Дисциплина: Теоретические основы информатики 3 курс 5 семестр
Лекционный курс

Вопросы экзаменационных билетов

1.Информатика как наука и вид практической деятельности. Место информатики в системе наук.
2.Информация, основные виды информации. Непрерывная и дискретная информация.
3.Количество информации. Единицы измерения информации. Кодирование информации.
4.Теория кодирования. 3 подхода к определению количества информации.
5.Теория кодирования. Оптимальное кодирование. Теоремы Шеннона.
6.Теория кодирования. Методы сжатия информации. Коды Шеннона-Фано.
7.Теория кодирования. Методы сжатия информации. Коды Хаффмана.
8.Теория кодирования. Методы сжатия информации. Кодирование методом Лемпел-Зива.
9.Теория кодирования. Методы восстановления информации. Биты четности и дублирование информации.
10.Теория кодирования. Методы восстановления информации. Коды Рида-Соломона. Расстояние Хэмминга. Коды Хэмминга.
11.Понятие системы счисления. Позиционные и непозиционные системы счисления. Примеры.
12.Представление чисел в различных системах счисления.
13.Системы счисления. Преобразование чисел в различных системах счисления. Методы преобразования чисел из десятичной системы счисления в двоичную.
14.Системы счисления, используемые в ЭВМ. Особенности систем счисления с основанием 2,8,16.
15.Математические операции в различных системах счисления. Примеры.
16.Представление информации в ЭВМ. Текстовая и графическая информация.
17.Представление чисел в ЭВМ. Прямой, обратный и дополнительный код. Числа с фиксированной и плавающей запятой, нормализованный код.
18.Представление информации в ЭВМ. Графическая и мультимедиа информация.
19.Понятие алгоритма. Принцип потенциальной осуществимости. Основные свойства алгоритмов. Понятие исполнителя алгоритмов.
20.Классификация алгоритмов. Блок-схемы описания алгоритмов. Формы записи алгоритмов.
21. Рекурсия и итерация в алгоритмах. Понятие о типах данных
22.Принципы программирования. Методы разработки и анализа алгоритмов.
23.Сложность алгоритмов. Варианты оценки сложности. Асимптотическая сложность алгоритма.
24.Не полиномиальные алгоритмы. Примеры задач НП. Замкнутость класса задач НП. Понятие неразрешимой задачи. Экстраалгоритм.
25.Основные методы разработки эффективных алгоритмов: итерационные формулы, рекурсивные алгоритмы, метод балансировки дерева, динамическое программирование
26.Реально выполнимые алгоритмы. Совпадение классов полиномиальных и реально выполнимых алгоритмов. Примеры полиномиальных алгоритмов.
27.Основные методы эффективного представления данных – основные модели данных.
28.Основные методы эффективного представления данных - динамические структуры данных.
29.Моделирование как основной метод научного познания. Понятие модели, классификация моделей.
30.Понятие автомата. Дискретный характер ЭВМ.
31.Понятие жадного алгоритма. Алгоритмы оптимизации на сетях и графах. Алгоритмы Прима и Краскала.
32.Алгоритмы оптимизации на сетях и графах. Алгоритмы Дейкстры и Флойда.
33.Алгоритмы оптимизации на сетях и графах. Задача Форда-Фалкерсона о потоках в сетях. Алгоритмы решения задачи о максимальном потоке.
34.Понятие о кибернетике. Система управления и ее реализация. Обратная связь в системе управления. Системы прогноза.
35. Основные задачи искусственного интеллекта
36.Понятие жадного алгоритма. Матроиды и их свойства.

Лекция №1

Раздел №1. Основы теории информации

Тема 1.1 Введение в теоретическую информатику

Содержание: Введение в курс «Теоретические основы информатики». Информатика как наука и вид практической деятельности. Место информатики в системе наук. Информация и ее виды. Непрерывная и дискретная информация. Количество информации. Единицы измерения информации.

Введение в курс «Теоретические основы информатики»

Целью освоения дисциплины «Теоретические основы информатики» является:

– формирование систематических знаний о современных методах информатики, её месте и роли в системе наук;

– расширение и углубление понятий теоретической информатики, теории кодирования, алгоритмизации и программирования;

– развитие абстрактного мышления, пространственных представлений, вычислительной, алгоритмической культур и общей математической и информационной культуры.

Дисциплина «Теоретические основы информатики» относится к вариативной части профессионального цикла. Она изучается после дисциплин «Дискретная математика»,«Математическая логика», «Программирование». Для ее освоения студенты также используют знания, умения, навыки, сформированные в ходе изучения основных математических курсов: «Математический анализ» «Алгебра», «Геометрия». Всего планируется проведение 13 лекций, 20 практических занятий, промежуточная аттестация — экзамен.

Изучаемые темы и разделы:

Тема	Название
Раздел 1. Основы теории информации	
1_1	Введение в теоретическую информатику.
1_2	Основы теории кодирования.
Раздел 2. Методы теоретической информатики	
2_1	Системы счисления и представление информации в ЭВМ.
2_2	Основы кибернетики, моделирования и теории искусственного интеллекта
Раздел 3. Основы теории алгоритмизации	
3_1	Основы теории алгоритмизации задач.
3_2	Алгоритмы оптимизации на сетях и графах.

Курс "Теоретические основы информатики" содержит лекционные и практические занятия.

На лекционные занятия выносятся общетеоретические темы. Рассматриваются базовая теоретические сведения о разделах теоретической информатики, теории кодирования, систем счисления и представления информации в ЭВМ, методология кибернетики, моделирования и теории искусственного интеллекта. Рассматриваются так же основы теории алгоритмизации задач, блок схемы алгоритмов, алгоритмы оптимизации на сетях и графах Учитывая незначительное количество учебных часов, выделяемых на данный курс, базовые формулы математических методов и их обоснование дается в основном без строгих математических доказательств.

На практических занятиях (которые, как правило, построены по типу семинарских занятий) разбираются методы решения задач, связанных с различными темами данного курса.

Используемая литература

При изучении данного курса необходимо привлечение дополнительного материала и проработка лекционного материала с помощью следующей учебной литературы:

Основная литература

1. Теоретические основы информатики : учебник / Р.Ю. Царев, А.Н. Пупков, В.В. Самарин и др. . - Красноярск : Сибирский федеральный университет, 2015. - 176 с. : табл., схем., ил. - Библиогр.: с. 140. - ISBN 978-5-7638-3192-4 ; То же [Электронный ресурс]. - URL: //biblioclub.ru/index.php?page=book&id=435850

2. Чернышев, А.Б. Теория информационных процессов и систем : учебное пособие / А.Б. Чернышев, В.Ф. Антонов, Г.Б. Суюнова ; Министерство образования и науки Российской Федерации, Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северо-Кавказский федеральный университет». - Ставрополь : СКФУ, 2015. - 169 с. : ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=457890

3. Душин, В.К. Теоретические основы информационных процессов и систем : учебник / В.К. Душин. - 5-е изд. - Москва : Издательско-торговая корпорация «Дашков и К°», 2016. - 348 с. : ил. - Библиогр. в кн. - ISBN 978-5-394-01748-3 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=453880

4. Волкова, В.Н. Теоретические основы информационных систем / В.Н. Волкова. - Санкт-Петербург. : Издательство Политехнического университета, 2014. - 300 с. : схем., табл., ил. - Библиогр. в кн. - ISBN 978-5-7422-3478-4 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=363073

5. Умняшкин, С.В. Основы теории цифровой обработки сигналов : учебное пособие / С.В. Умняшкин. - Москва : Техносфера, 2016. - 528 с. : ил., табл., схем. - (Мир цифровой обработки). - Библиогр. в кн.. - ISBN 978-5-94836-424-7 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=444859

6. Горелик, В.А. Пособие по дисциплине «Теоретические основы информатики» : учебное пособие / В.А. Горелик, О.В. Муравьева, О.С. Трембачева ; Министерство образования и науки Российской Федерации, Московский педагогический государственный университет. - Москва : МПГУ, 2015. - 120 с. : ил. - Библиогр. в кн. - ISBN 978-5-4263-0220-4 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=472092

7. Быкова, В.В. Комбинаторные алгоритмы: множества, графы, коды : учебное пособие / В.В. Быкова ; Министерство образования и науки Российской Федерации, Сибирский Федеральный университет. - Красноярск : Сибирский федеральный университет, 2015. - 152 с. : табл., ил. - Библиогр.: с. 120-121. - ISBN 978-5-7638-3155-9 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=435666

Дополнительная литература

1. Системы и сети передачи информации / Ю. Громов, И.Г. Карпов, Г.Н. Нурутдинов и др. - Тамбов : Издательство ФГБОУ ВПО «ТГТУ», 2012. - 128 с. : схем., ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: //biblioclub.ru/index.php?page=book&id=277938

2. Математические методы и модели исследования операций : учебник / под ред. В.А. Колемаева. - Москва : Юнити-Дана, 2015. - 592 с. : ил., табл., граф. - Библиогр. в кн. - ISBN 978-5-238-01325-1 ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=114719

3. Шевелев, Ю.П. Сборник задач по дискретной математике (для практических занятий в группах) [Электронный ресурс] : учебное пособие / Ю.П. Шевелев, Писаренко Л. А., Шевелев М. Ю. — СПб. : Лань, 2013. — 524 с. — Режим доступа URL: http://e.lanbook.com/books/element.php?pl1_id=5251.

4. Штарьков, Ю.М. Универсальное кодирование. Теория и алгоритмы [Электронный ресурс] / Ю. М. Штарьков — М. : Физматлит, 2013. — 280 с. — Режим доступа: URL: http://e.lanbook.com/books/element.php?pl1_id=59667.
5. Асанов, М.О. Дискретная математика: графы, матроиды, алгоритмы [Электронный ресурс] : учеб. пособие / М.О. Асанов, В.А. Баранский, В.В. Расин. — Электрон. дан. — Санкт-Петербург : Лань, 2010. — 368 с. — Режим доступа: https://e.lanbook.com/book/536.
6. Горлач, Б.А. Математическое моделирование. Построение моделей и численная реализация [Электронный ресурс] : учеб. пособие / Б.А. Горлач, В.Г. Шахов. — Электрон. дан. — Санкт-Петербург : Лань, 2016. — 292 с. — Режим доступа: https://e.lanbook.com/book/74673.
7. Биллиг, В. Подготовка к ЕГЭ по информатике : курс / В. Биллиг. - 2-е изд., исправ. - Москва : Национальный Открытый Университет «ИНТУИТ», 2016. - 51 с. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=429191

Некоторые интернет-ссылки

- Алгоритм Дейкстры. algolist.manual.ru/maths/graphs/shortpath/dijkstra.php
- Алгоритм Флойда. algolist.manual.ru/maths/graphs/shortpath/floyd.php
- Алгоритмы нахождения максимального потока. algolist.manual.ru/maths/graphs/maxflows/
- Белова И.М. Компьютерное моделирование: Учебно-методическое пособие для студентов направления "Прикладная математика и информатика" и специальности "Математическое обеспечение и администрирование информационных систем". window.edu.ru/window/
- Ватолин Д.С. Алгоритмы сжатия изображений. algolist.manual.ru/compress/image/fractal/
- Крис Касперски, Могущество кодов Рида-Соломона или информация, воскресшая из пепла www.insidepro.com/kk/027/027r.shtml
- Нахождение максимального пропускного потока. algolist.manual.ru/maths/graphs/netflow.php
- Нахождение на графе минимального остовного дерева. algolist.manual.ru/maths/graphs/span.php
- Сергиевская И.М. Математическая логика и теория алгоритмов: Учебное пособие. window.edu.ru/window/
- Тарасевич Ю.Ю. Элементы дискретной математики для программистов. window.edu.ru/window/
- Нахождение на графе минимального остовного дерева. algolist.manual.ru/maths/graphs/span.php
- Алгоритм Дейкстры. algolist.manual.ru/maths/graphs/shortpath/dijkstra.php
- Алгоритм Флойда. algolist.manual.ru/maths/ graphs/shortpath/ floyd.php
- Нахождение максимального пропускного потока. algolist.manual.ru/maths/graphs/netflow.php
- Алгоритмы нахождения максимального потока. algolist.manual.ru/maths/graphs/maxflows/
- С.Д. Кузнецов Методы сортировки и поиска. algolist.manual.ru/
- Сергиевская И.М. Математическая логика и теория алгоритмов: Учебное пособие. window.edu.ru/window/
- Нахождение на графе минимального остовного дерева. algolist.manual.ru/maths/graphs/span.php

1.Информатика как наука и вид практической деятельности. Место информатики в системе наук.

Информатика как наука и вид практической деятельности

ТОИ объединяют в себе результаты некоторых математических и инженерных наук, которые нашли широкое применение в информатике. Здесь находятся математические описания тех информационных технологий, которые широко распространены в приложениях информатики.

Связь ТОИ с другими дисциплинами и науками

Нет возможности полностью в одном курсе интегрировать все ТОИ. Наиболее важные части выделены в отдельные дисциплины: численные методы, теория алгоритмов, дискретная

математика, исследования определений, компьютерное моделирование. В свою очередь ТОИ является частью большой информатики.

Термин «информатика» возник в XX в. 50-60 г. на одном из конгрессов посвященных кибернетике. Словом «информатика» – как объединение ряда наук, связанных с компьютером, вначале называлась кибернетика.

В нашем курсе мы ограничимся изучением:

1) основных проблем и приложений теории кодирования;

2) прикладной области теории алгоритмов: проектирование алгоритмов и оценка их сложности;

3) некоторые алгоритмические задачи дискретной математики.

Термин ***"информатика"*** (франц. *informatique*) происходит от французских слов *information* (информация) и *automatique* (автоматика) и дословно означает ***"информационная автоматика"***.

Широко распространён также англоязычный вариант этого термина — ***"Computer science"***, что означает буквально ***"компьютерная наука"***.

Информатика — это основанная на использовании компьютерной техники дисциплина, изучающая структуру и общие свойства информации, а также закономерности и методы её создания, хранения, поиска, преобразования, передачи и применения в различных сферах человеческой деятельности.

В 1978 году международный научный конгресс официально закрепил за понятием *"информатика"* области, связанные с **разработкой, созданием, использованием и материально-техническим обслуживанием систем обработки информации, включая компьютеры и их программное обеспечение, а также организационные, коммерческие, административные и социально-политические аспекты компьютеризации — массового внедрения компьютерной техники во все области жизни людей**.

Таким образом, информатика базируется на компьютерной технике и немыслима без нее.

Информатика — научная дисциплина с широчайшим диапазоном применения. Её **основные направления**:

- **разработка вычислительных систем и программного обеспечения**;
- **теория информации**, изучающая процессы, связанные с передачей, приёмом, преобразованием и хранением информации;
- **методы искусственного интеллекта**, позволяющие создавать программы для решения задач, требующих определённых интеллектуальных усилий при выполнении их человеком (логический вывод, обучение, понимание речи, визуальное восприятие, игры и др.);
- **системный анализ**, заключающийся в анализе назначения проектируемой системы и в установлении требований, которым она должна отвечать;
- **методы машинной графики, анимации, средства мультимедиа**;
- **средства телекоммуникации**, в том числе, **глобальные** компьютерные сети, объединяющие всё человечество в единое информационное сообщество;
- **разнообразные приложения**, охватывающие производство, науку, образование, медицину, торговлю, сельское хозяйство и все другие виды хозяйственной и общественной деятельности.

Информатику обычно представляют состоящей из двух частей:

- **технические средства**;
- **программные средства**.

Технические средства, то есть ***аппаратура компьютеров***, в английском языке обозначаются словом ***Hardware***, которое буквально переводится как ***"твёрдые изделия"***.

А для ***программных средств*** выбрано (а точнее, создано) очень удачное слово ***Software*** (буквально — ***"мягкие изделия"***), которое подчёркивает равнозначность программного обеспечения и самой машины и вместе с тем подчёркивает способность программного обеспечения модифицироваться, приспосабливаться, развиваться.

Программное обеспечение — это совокупность всех программ, используемых

компьютерами, а также вся область деятельности по их созданию и применению.

Помимо этих двух общепринятых ветвей информатики выделяют ещё одну существенную ветвь — ***алгоритмические средства***. Для неё российский академик А.А. Дородницин предложил название ***Brainware*** (от англ. *brain* — интеллект). **Эта ветвь связана с разработкой алгоритмов и изучением методов и приёмов их построения**.

Алгоритмы — это правила, предписывающие выполнение последовательностей действий, приводящих к решению задачи.

Нельзя приступить к программированию, не разработав предварительно алгоритм решения задачи.

Роль информатики в развитии общества чрезвычайно велика. С ней связано начало революции в области накопления, передачи и обработки информации. Эта революция, следующая за революциями в овладении веществом и энергией, затрагивает и коренным образом преобразует не только сферу материального производства, но и интеллектуальную, духовную сферы жизни.

Рост производства компьютерной техники, развитие информационных сетей, создание новых информационных технологий приводят к значительным изменениям во всех сферах общества: в производстве, науке, образовании, медицине и т.д.

Место информатики в системе наук

Информатику принято делить на:

1) *языки и методы программирования*
2) *вычислительную технику*
3) *программное обеспечение*
4) *ТОИ*
5) *моделирование и искусственный интеллект*

В результате такого разделения можно проследить связь информатики с другими науками. Все без исключения науки связанны с математикой. Теория вычислительной техники имеет широкую связь с электроникой, физикой и техническими дисциплинами.

Прослеживая связь с математическими дисциплинами важно историческое развитие математики, которое привело к ТОИ.

Со средних веков прослеживаются работы по дискретной математике, она содержится частично в алгебре и частично в геометрии.

В начале XX в. в связи с кризисом математики (теории множеств) возникла математическая теория алгоритмов.

Основой ТОИ являются работы по теории оптимизации и управления, которые дали в конце 40-х годов сформировать кибернетику.

Теоретические основы компьютерного моделирования и искусственного интеллекта разрабатываются до сих пор.

2.Информация, основные виды информации. Непрерывная и дискретная информация.

Информация и ее виды. Непрерывная и дискретная информация

Мы живем в материальном мире. Все, что нас окружает, и с чем мы сталкиваемся ежедневно, относится либо к физическим телам, либо к физическим полям. Из курса физики мы знаем, что состояния абсолютного покоя не существует, и физические объекты находятся в состоянии непрерывного движения и изменения, которое сопровождается обменом энергией и ее переходом из одной формы в другую.

Все виды энергообмена сопровождаются появлением сигналов, то есть, все сигналы имеют в своей основе *материальную энергетическую природу*. При взаимодействии сигналов с физическими телами, в последних возникают определенные изменения свойств – это явление называется **регистрацией сигналов.** Такие изменения можно наблюдать, измерять или

фиксировать иными способами – при этом возникают и регистрируются новые сигналы, то есть, образуются данные.

Данные – это зарегистрированные сигналы.

Данные несут в себе **информацию** о событиях, происходящих в материальном мире, поскольку они являются регистрацией сигналов, возникающих в результате этих событий. Однако данные не тождественны информации, станут ли эти данные информацией, зависит от очень многих обстоятельств. Рассмотрим пример. Наблюдая за состязаниями бегунов, мы с помощью механического секундомера регистрируем начальное и конечное положение стрелки прибора. В итоге мы замеряем величину ее перемещения за время забега – это регистрация данных. Однако информацию о времени преодоления дистанции мы пока не получим. Для того чтобы данные о перемещении стрелки дали информацию о времени забега, необходимо наличие **метода** пересчета одной физической величины в другую. Надо знать цену деления шкалы секундомера (или знать метод ее определения) и надо также знать, как умножается цена деления прибора на величину перемещения, то есть надо еще обладать математическим методом умножения.

Несмотря на то, что с понятием информация мы сталкиваемся ежедневно, строгого и общепризнанного ее определения до сих пор не существует, поэтому вместо определения обычно используют **понятие об информации.** Информация – это настолько общее и глубокое понятие, что его нельзя объяснить одной фразой. В это слово вкладывается различный смысл в технике, науке и в житейских ситуациях.

Для информатики как технической науки понятие информации не может основываться на таких антропоцентрических понятиях, как **знание**, и не может опираться только на объективность фактов и свидетельств. Средства вычислительной техники обладают способностью обрабатывать информацию автоматически, без участия человека, и ни о каком знании или незнании здесь речь идти не может. Эти средства могут работать с искусственной, абстрактной и даже ложной информацией, не имеющей объективного отражения ни в природе, ни в обществе.

Информация – это продукт взаимодействия данных и адекватных им методов.

Применительно к компьютерной обработке данных под информацией понимают некоторую последовательность символических обозначений (букв, цифр, закодированных графических образов и звуков и т.п.), несущую смысловую нагрузку и представленную в понятном компьютеру виде. Каждый новый символ в такой последовательности символов увеличивает информационный объем сообщения.

Свойства информации

Адекватность информации может выражаться в трех формах: семантической, синтаксической, прагматической.

Синтаксическая адекватность. Она отображает формально-структурные характеристики информации и не затрагивает ее смыслового содержания. На синтаксическом уровне учитываются тип носителя и способ представления информации, скорость передачи и обработки, размеры кодов представления информации, надежность и точность преобразования этих кодов и т.п. Информацию, рассматриваемую только с синтаксических позиций, обычно называют данными, так как при этом не имеет значения смысловая сторона. Эта форма способствует восприятию внешних структурных характеристик, т.е. синтаксической стороны информации.

Семантическая (смысловая) адекватность. Эта форма определяет степень соответствия образа объекта и самого объекта. Семантический аспект предполагает учет смыслового содержания информации. На этом уровне анализируются те сведения, которые отражает информация, рассматриваются смысловые связи. В информатике устанавливаются смысловые связи между кодами представления информации. Эта форма служит для формирования понятий и представлений, выявления смысла, содержания информации и ее обобщения.

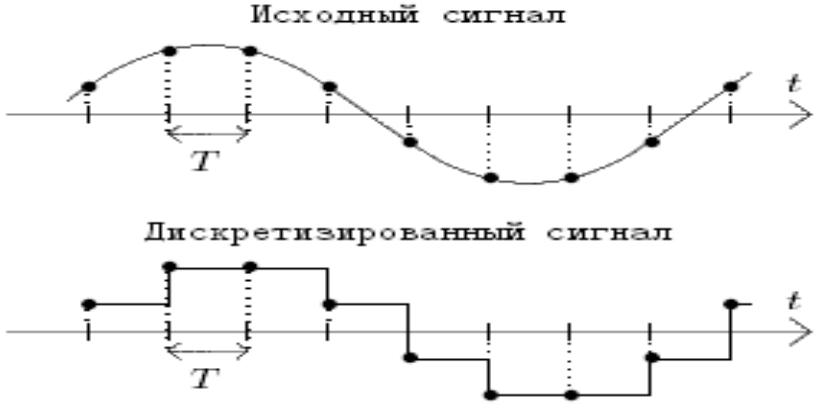
Прагматическая (потребительская) адекватность. Она отражает отношение информации и ее потребителя, соответствие информации цели управления, которая на ее основе реализуется. Проявляются прагматические свойства информации только при наличии единства

информации (объекта), пользователя и цели управления. Прагматический аспект рассмотрения связан с ценностью, полезностью использования информации при выработке потребителем решения для достижения своей цели. С этой точки зрения анализируются потребительские свойства информации. Эта форма адекватности непосредственно связана с практическим использованием информации, с соответствием ее целевой функции деятельности системы.

Информация всегда связана с источником и получателем сообщения по следующей схеме:

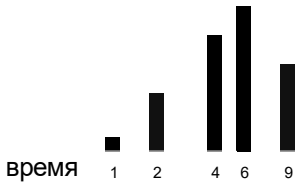
Передача сообщения от источника к получателю происходит в материально-энергетической форме: электромагнитной, световой, звуковой и т. д. Прием информации связан с изменением во времени какой-то величины, характеризующей изменение параметров физической среды – вещественной функции. Если эта функция непрерывна, то имеет место **аналоговая информация**. Если эта функция задана только в некоторых точках (отрезках), то это **дискретная информация**.

Непрерывную информацию может, например, выдавать датчик атмосферного давления или датчик скорости автомашины. Дискретную информацию можно получить от любого цифрового индикатора: электронных часов, счетчика магнитофона и т. п. Дискретная информация удобнее для обработки человеком, но непрерывная информация часто встречается в практической работе, поэтому необходимо уметь переводить непрерывную информацию в дискретную (дискретизация) и наоборот.



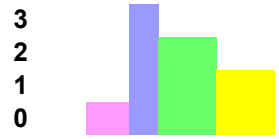
Для дискретной информации определяется понятие дискретности по уровню и по времени. **Дискретность по времени** означает, что сигнал задается только в отдельные дискретные моменты времени. Такой сигнал фактически имеет уже импульсную природу, а не непрерывную.

Дискретное время



Дискретность по уровню означает, что величина или амплитуда сигнал задается только для определенного **дискретного количества значений (уровней**).

4 уровня сигналов



Можно определить и **одновременно дискретность по времени и уровню** (это фактически уже **цифровой сигнал**).

Выделение уровней сигнала позволяет ввести цифровую запись сигнала. Таким образом, введение дискретности по времени и уровню позволяет сделать сигнал дискретным **(дискредитировать или оцифровать)**. В современных ЭВМ есть специальные устройства как для оцифровки непрерывной информации (например - звуковая плата, при оцифровке записи звука с микрофона), так и для превращения дискретного сигнала в непрерывный (например – модем, при передаче данных по телефонной линии). Модем (это слово происходит от слов модуляция и демодуляция) представляет собой устройство для такого перевода: он переводит цифровые данные от компьютера в звук или электромагнитные колебания-копии звука и наоборот.

3.Количество информации. Единицы измерения информации. Кодирование информации.

Количество информации. Единицы измерения информации

Как и любое вещество в физике, так и информация в информатике должна иметь количественные характеристики. Возможно ли объективно измерить количество информации? Важнейшим результатом теории информации является следующий вывод: в определенных, весьма широких условиях можно пренебречь качественными особенностями информации и выразить ее количество числом, а также сравнить количество информации, содержащейся в различных группах данных. В настоящее время получили распространение подходы к определению понятия «количество информации», основанные на том, что информацию, содержащуюся в сообщении, можно нестрого трактовать в смысле ее новизны, или, иначе, уменьшения неопределенности наших знаний об объекте.

В современной информатике приняты следующие варианты измерения информации (или по научному - меры):

- *энтропийная (вероятностная);*
- *объемная (семантическая);*
- *алгоритмическая*.

Любое измерение требует создания шкалы (меры), в которой вводится некая единица измерения. Единицей измерения информации считается бит (его смысл поясним позже). В вычислительной технике вся обрабатываемая информация не зависимо от ее природы (текст, число, изображение и т.д.) представляется в двоичной форме записи числа, т.е. состоящая из двух символов 0 и 1. Объем информации, который занимает один такой символ и называется битом. Бит слишком мелкая единица измерения. На практике чаще применяется более крупная единица – **байт, равная восьми битам.**

Широко используются также еще более крупные производные единицы информации:

- 1 Килобайт (Кбайт) = 1024 байт = 2^{10} байт,
- 1 Мегабайт (Мбайт) = 1024 Кбайт = 2^{20} байт,
- 1 Гигабайт (Гбайт) = 1024 Мбайт = 2^{30} байт,

- 1 Терабайт (Тбайт) = 1024 Гбайт = 2^{40} байт,
- 1 Петабайт (Пбайт) = 1024 Тбайт = 2^{50} байт.

Скорость передачи информации измеряется в Кбайт/с, бит/с, байт/с и т.д.

Лекция №2

Раздел №1. Основы теории информации

Тема 1.2 Основы теории кодирования

Содержание: Кодирование информации. Измерение информации – 3 базовых подхода. Количество информации и вероятность. Оптимальное кодирование. Теоремы Шеннона. Основные задачи теории кодирования.

Кодирование информации

Информация представляется в ЭВМ с помощью записи, которую принято называть закодированной. В этом названии содержится понимание того, что записать информацию и считать ее возможно только в том случае, если есть правило, определяющее порядок ее записи. Такой процесс принято называть кодированием. Речь, письменность, рисунок, фотография, математическая формула – все это варианты кодирования информации.

В теоретической информатике принципы и методы кодирования описываются специальной теорией кодирования. В нашем курсе теория кодирования будет занимать значительную часть содержания (что определяется как ее важностью, так и отсутствием данной тематики в других дисциплинах).

Для кодирования информации необходимо разработать **код – язык на котором можно записать любую информацию данного типа**. В любом языке есть синтаксис и семантика. **Синтаксис** – правило записи предложения языка. **Семантика** – понимание смысла конструкции языка.

Следовательно, для кодирования нужны **формальные языки**, у которых синтаксис строгий, а семантика однозначная. При этом кодирование не должно вести к потере информации.

Поэтому для любой системы кодирования кроме формальной стороны важно свойство **однозначного кодирования и декодирования** без потерь информации, что возможно не в каждом коде.

Основной проблемой практического применения методов кодирования стала проблема кодирования информации в ЭВМ. При кодировании необходимо учитывать форму и вид информации. Форма может быть дискретной и непрерывной.

Для дальнейшего знакомства с теорией кодирования дадим несколько упрощенных вариантов определений базовых терминов:

- Код – метод представления и записи информации. Как правило, кодирование – перевод записи информации из одного вида в другой. Кодирование информации состоит в представлении чисел и слов соответствующими им комбинациями символов. Совокупность всех комбинаций из определенного количества символов, которые избраны для представления информации, называют кодом. Каждую такую комбинацию символов будем называть кодовой комбинацией. Общее количество кодовых комбинаций может быть меньше или равно числу всевозможных комбинаций из заданного количества символов.
- Декодирование – обратное кодирование информации, своего рода «расшифровка» информации.
- Обратимое кодирование – кодирование позволяющее выполнить однозначное декодирование.
- *Равномерный код* – код, все комбинации которого содержат одинаковое количество символов.
- *Неравномерный код* – код с различным количеством символов для различных комбинаций (например, азбука Морзе).

4.Теория кодирования. 3 подхода к определению количества информации.

Измерение информации – 3 базовых подхода

Рассмотрим подробнее 3 базовых меры измерения информации:

Объемный (символьный) подход к определению количества информации

Объемная характеристика информации представляет собой количество символов, содержащихся в конкретном сообщении.

Например, одно и тоже число можно записать разными способами:

1-й способ – 20;

2-й способ – “двадцать”;

3-й способ – XX;

4-й способ – 011 000.

Любой из этих способов чувствителен к форме представления (записи) числа.

Символьный подход не связывает количество информации в сообщении с его содержанием. Как правило в этом подходе определена шкала переводящая любой символ в единицы информационного объема (например в биты), причем не обязательно равномерный.

Алфавит – все множество символов данного языка, включая цифры, знаки препинания и даже пробел. Полное количество символов – мощность алфавита N. В данной мере принимается, что появление символов равновероятно. Сравнить объем можно только для равномощных алфавитов. Тогда для этой меры удобнее двоичная форма записи и использование битов.

Алгоритмическая оценка количества информации

Алгоритмическая оценка количества информации характеризуется сложностью (размером) программы, которая позволяет ее произвести.

Так, например, компьютерная программы, печатающее слово из одних нулей достаточно проста, а программа, печатающее слово из 0 и 1 уже более сложная. При разных машинах и разных языках программирования (алгоритмах) это все разное. Поэтому задаются некоторой вычислительной машиной (чаще всего машиной Тьюринга), а предлагаемая количественная оценка информации определяется сложностью слова, как минимальное число внутренних состояний машины, требуемой для его воспроизведения. В теоретической информатике разработана тория меры Колмогорова, которая дает математические основы расчета алгоритмической меры.

Вероятностный подход к измерению информации

Наиболее продуктивна для ТОИ вероятностная мера. Она наиболее близка теоретическому понятию информации как меры уменьшающей незнание (в древней Греции функция незнания была названа энтропией).

Количество информации и вероятность

Энтропийная характеристика информации принята в теории информации и кодирования. Она представляет собой меру неопределенности в появлении некоторых событий и выражается математической зависимостью от совокупности вероятности этих событий. Количество информации в сообщении определяется при энтропийном методе оценки тем, насколько уменьшится эта мера после получения сообщения.

Подход Хартли

Если считать, что события равновероятны, а функция объема информации обладает аддитивным свойством (т.е. объем информации независимых событий складывается). То можно сделать предположение о виде функциональной зависимости:

$$F(a \text{ и } b) = F(a) + F(b)$$

Пусть мы бросаем монету с 2 вариантами исходов. При одном броске вариантов 2, при 2-х бросках вариантов 4, при к бросках вариантов $2^к$ => $F(X^n) = n*F(X)$. Таким свойством обладает функция логарифма. Основание логарифма определяется единицей измерения. Если это бит, то $log_m(2)=1$ => формула Хартли:

$$H = \log_2 N,$$

Данную формулу предложил в 1928 г Хартли, США.

Подход Шеннона

Шеннон предложил усовершенствовать формула Хартли на случай разновероятных событий. Для этого он использовал принятый в теории вероятности метод усреднения взвешенного среднего. Если исходов N, но их вероятности различны: $p_1, p_2 \ldots p_N$, то взвешенное среднее $Xcp = \Sigma p_k * X_k$ аналогично получаем формулу Шеннона:

$$H = -(p_1 \log_2 p_1 + p_2 \log_2 p_2 + \ldots + p_N \log_2 p_N). = \Sigma p_k * H_k$$

При равных вероятностях формулы Хартли и Шеннона совпадают

5.Теория кодирования. Оптимальное кодирование. Теоремы Шеннона.

Оптимальное кодирование

Одной из задач теории кодирования является задача выбрать такой код, чтобы объем информации в результате кодирования был как можно меньше. Такой код принято называть оптимальным. Можно оптимизировать процессы кодирования и с точки зрения других критериев – скорости кодирования, криптографической стойкости, объема алгоритма, выполняющего кодирование, возможности восстановления информации при наличии ее потерь в линиях связи.

Теоремы Шеннона. Основные задачи теории кодирования

Работы Клода Шеннона явились основанием для выделения теории кодирования как отдельной науки. Его заслуга перед теорией кодирования заключается прежде всего в том, что он определил основные задачи теории кодирования, сформулировал их в качестве теорем и доказал основные из них. Таким образом, теоремы Шеннона являются основными классификаторами задач теории кодирования. Не вдаваясь в математическую сторону доказательства этих теорем, приведем основной смысл этих теорем и их значение для теории кодирования.

Шеннон сформулировал три основные проблемы и три теоремы на основе этих проблем:

1 проблема: для записи информации необходим язык кодирования, который позволял бы кодировать, (то есть записывать в одном языке информацию) и декодировать эту информацию (расшифровывать из одного языка информацию). Отсюда проблема существования для любой информации какого-либо способа кодирования и декодирования.

Теорема 1: для любой информации всегда может быть построен код позволяющий однозначно кодировать и декодировать.

2 проблема: проблема избыточности кода и поиска оптимального кода с помощью различных алгоритмов сжатия.

Теорема 2: всегда можно создать систему кодирования эффективность, которой будет сколь угодно близкой к оптимальному коду.

Может оказаться так, что оптимального кода нет, тогда мы всегда можем улучшать сколь угодно долго постепенно, приближаясь к оптимальному коду. Таким образом получается, что не существует барьера для сжатия кода.

3 проблема: возникла при передаче информации, когда возможные помехи и соответствуют потере информации.

Таким образом, помехи канала связи приводят к потере части информации, следовательно, для восстановления информации код должен быть избыточным. Возникает проблема создания помехоустойчивых кодов и кодов, которые могли бы самовостанавливать информацию за счет своей избыточности.

Теорема 3: при наличии помех в канале связи всегда можно построить такую систему кодирования, чтобы информация была восстановлена с данной вероятностью.

Теория кодирования возникла при изучении понятий информации и информационных технологий. Эти информационные технологии потребовали своего математического описания. Потребовалось так же определить понятие информации. При этом оказалось, что проблемы хранения, преобразования и передачи информации напрямую зависят от формы записи информации, т.е. от ее кодирования.

Это позволило (Шеннон) сформировать 3 основные задачи теории кодирования.

1) *построение оптимального кода, в котором информация занимала бы минимальный объем;*

2) *при передаче информации по линии связи возможно появление ошибок (необходимо разработать кодировку, которая позволяла бы находить и исправлять ошибки);*
3) *при кодировании и декодировании перевода из одного кода в другой информация должна однозначно преобразовываться без потерь.*

Еще одной важной задачей связанной с третьей проблемой является задача математического исчисления. Компьютер должен производить действия над числами. Результаты в двоичном виде. Отсюда проблемы кодировки и исчисления чисел.

6.Теория кодирования. Методы сжатия информации. Коды Шеннона-Фано.

Лекция №3

Раздел №1. Основы теории информации

Тема 1.2 Основы теории кодирования

Содержание: Основные методы сжатия информации – коды Шеннона-Фано, Хаффмана, Лемпел-Зива. Средняя длина кода. Примеры кодирования с помощью кодов Шеннона-Фано, Хаффмана, Лемпел-Зива.

Основные методы сжатия информации

Алгоритмы сжатия информации делятся на алгоритмы с потерей информации и без потери информации.

- алгоритмы с потерей информации не дают возможность декодирования. К таким относятся технологии: JPEG, MPEG, некоторые технологии звукового кодирования.
- алгоритмы без потери информации: данные алгоритмы могут сжать не более теоретического предела, вычисленного по вероятностному подходу, то есть это алгоритм, который удаляет избыточность кодировки.

Известны три основных подхода к построению архиваторов:

- RLE – учет повторяющихся последовательностей;
- KWE – создание таблицы-словаря с кодировкой повторяющихся слов;
- Хаффмана – метод неравномерного кода – чем чаще встречается знак, тем короче его код.

Подход RLE

В этом случае необходимо искать элементы, которые повторяются подряд, тогда их можно заменить на образец элемента и число повторений.

Данный алгоритм очень удобен для векторной графики, но не очень для сжатия текстовой информации. Архиваторы, построенные по этому принципу, одну информацию сжимают хорошо, а другую плохо. Его используют отдельно в графике и для общего вида файлов, только для части алгоритма сжатия.

Подход KWE

Если дан текст, то в нем можно найти либо повторяющиеся слова, либо по крайней мере корни слов. В данном случае алгоритм предполагает составить динамические словарь, размер которого меняется. Динамический словарь содержит информацию наподобие кодовой таблицы ЭВМ, то есть код – образец слова. Коды подбираются так, чтобы их нельзя было спутать с другими частями текста. Обычным стандартом такого архиватора является двухбайтовая кодировка. Двухбайтовые числа называются тугены. 2 байта обеспечивают размер словаря до 256000 слов. Используют двухбайтовые тугены и соответствующий словарь.

Алгоритм состоит в том, что мы сканируем информацию и находим повторяющиеся конструкции, желательно большого размера. В результате они запоминаются в динамический словарь. Слова небольшого размера смысла кодировать не имеет. Найденные слова удаляются из текста и заменяются кодом из словаря.

Данный алгоритм наиболее удобен для текстов. Однако его можно использовать для произвольного двоичного кода. Но в нем надо искать одинаковые комбинации из 20 и более бит. Однако в этом случае качество кодирования снижается. Если заданы произвольные битовые данные, то размер комбинации можно устанавливать самому. Наибольшую эффективность на практике оказывают комбинации размером 18-20 бит. Слишком короткие комбинации (по битам) встречаются очень часто, но степень сжатия будет невысока, слишком длинные комбинации

(более 3 байт) встречаются редко. Значит, есть некий оптимальный выбор длины буквы. Как можно убедиться алгоритмы RLE и KWE дополняют друг друга, поэтому их комбинируют.

Коды Шеннона-Фано

Исторически первым подходом к построению алгоритмов сжатия информации были работы Шеннона (и независимо от него работы Фано) о возможности создания оптимального кода, который можно было бы использовать для сжатия информации. Шеннон предложил вариант такого метода, основанный на построении неравномерного кода. Тогда для более часто встречающегося элемента информации можно выбрать более короткий код, что даст уменьшение общего объема кода.

Понятно, что кодирование информации допускается тогда, когда возможно последующее однозначное декодирование. Если для неравномерного кода декодирование возможно, то его принято называть префиксным. Шеннон предложил один из вариантов построения такого префиксного неравномерного кода, который называется кодом Шеннона-Фано. В дальнейшем, были построены более оптимальные неравномерные коды (например Хафмена), поэтому код Шеннона-Фано используется только для иллюстрации теории сжатия с помощью неравномерных кодов.

Код Шеннона−Фэно строится следующим образом:

- буквы алфавита сообщений выписываются в таблицу в порядке убывания вероятностей;
- затем они разделяются на две группы так, чтобы суммы вероятностей в каждой из групп были по возможности одинаковы;
- всем буквам верхней половины в качестве первого символа приписывается 1, а всем нижним 0;
- каждая из полученных групп, в свою очередь, разбивается на две подгруппы с одинаковыми суммарными вероятностями и т. д. ;
- процесс повторяется до тех пор, пока в каждой подгруппе останется по одной букве.

Пример – пусть заданы 8 букв исходного алфавита и вероятности их появления в тексте, построим код Шеннона для двоичного кода (алфавит 0 и 1):

Буквы	Вероят-ности	Ступени разбиения							Кодовые комбинации
		1	2	3	4	5	6	7	
Z1	1/2	1							1
Z2	1/4	0	1						01
Z3	1/8	0	0	1					001
Z4	1/16	0	0	0	1				0001
Z5	1/32	0	0	0	0	1			00001
Z6	1/64	0	0	0	0	0	1		000001
Z7	1/128	0	0	0	0	0	0	1	0000001
Z8	1/128	0	0	0	0	0	0	0	0000000

Качество кодирования можно оценить по средней длине кода, которая вычисляется как сумма произведений вероятностей букв на длины их кодовых комбинаций. В этом примере получим: ***L=1/2·1+1/4·2+1/8·3+1/16·4+1/32·5+1/64·6+1/128·7·2=(32·1+16·2+8·3+4·4+2·5+6+7)/64=(64+24+16+10+13)/64=127/64=1,984 (бит)***. Это означает, что по сравнению с объемным кодированием (где потребуется 3 бита на букву) мы имеем (1-1.984/3)*100% =33,8% уменьшения объема.

7.Теория кодирования. Методы сжатия информации. Коды Хаффмана.

Коды Хаффмана

Хаффман предложил строить коды по типу бинарного дерева. Он исходил из того, что кодировать придется в двоичном коде через 0 и 1, поэтому он предложил разместить буквы какого-то текста в порядке убывания его частоты или вероятности (частота – количество букв в тексте; вероятность – отношение количества данной буквы к общему количеству букв).

Подход Хаффмана хорош тем, что получаем неравномерный код в соответствии с частотами каждой буквы. Доказана теорема: данный код является оптимальным, то есть любой другой неравномерный код не может сжать информацию более, чем данный.

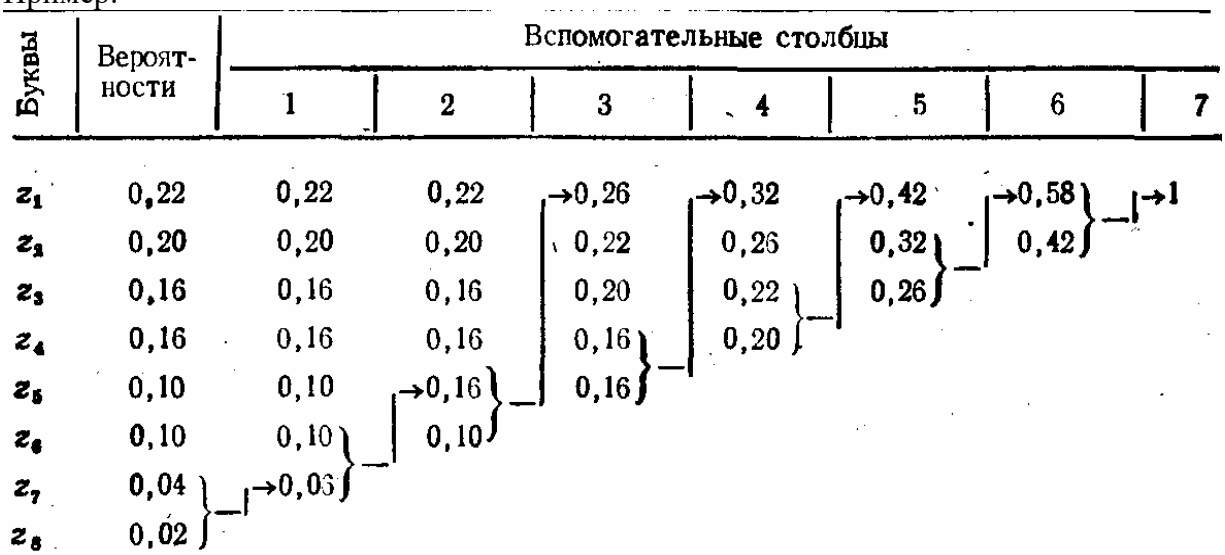
Данный принцип позволяет строить в двоичной системе координат коды для алфавитов, число букв которых не есть степень двойки.

Алгоритм Хаффмана легко реализуется в ЭВМ и воспроизводится. Образуется в результате кодирования дерево, выражается в качестве табличного соответствия и прилагается к сжатому документу. Это самый быстрый алгоритм.

Алгоритм построения оптимального кода, предложенный в 1952 году Хаффменом:

1. буквы первичного алфавита выписываются в основной столбец в порядке убывания вероятностей;
2. две последние буквы объединяются в одну вспомогательную букву, которой приписывается суммарная вероятность;
3. вероятности букв, не участвовавших в объединении, и полученная суммарная вероятность снова располагаются в порядке убывания вероятностей в дополнительном столбце, а две последние объединяются;
4. процесс продолжается до тех пор, пока не получим единственную вспомогательную букву с вероятностью, равной единице.

Пример:



В результате построения получим:

Z1	Z2	Z3	Z4	Z5	Z6	Z7	Z8
01	00	111	110	100	1011	10101	10100

Самостоятельно: вычислите среднюю длину кода Хафмена и % сжатия информации.

8.Теория кодирования. Методы сжатия информации. Кодирование методом Лемпел-Зива.

Коды Лемпел-Зива

Существуют 2 реализации метода Лепел-Зива - *Алгоритм Лемпел–Зива (LZ)*, или *Лемпел–Зива–Уэлча (LZW)*

Это еще одна схема сжатия, основанная на сведении к минимуму избыточности кодирования. Вместо кодирования каждого отдельного символа LZ-алгоритм кодирует часто встречающиеся символьные последовательности. Например, можно заменить краткими кодами слова "который" "так же", "там", оставляя имена собственные и другие слова, встречающиеся один раз, незакодированными. Программа, реализующая LZ-алгоритм, просматривает файл, содержащий текст, символы или байты графической информации, и выполняет статистический анализ для построения своей кодовой таблицы.

Если заменить 60–70 % текста символами, длина кода которых составляет менее половины первоначальной длины, можно достигнуть сжатия приблизительно в 50 %. При попытке применить этот алгоритм к исполняемым файлам получаются менее впечатляющие результаты

(от 10 до 20 % сжатия), так как избыточность кода, создаваемого компиляторами, значительно меньше избыточности текста на естественном языке. Файлы баз данных – еще один трудный случай: записи, содержащие имена, адреса и номера телефонов, редко повторяются, а поэтому плохо поддаются сжатию.

Именно алгоритмы Лемпеля-Зива лежат в основе наиболее известных архиваторов: zip, arj, rar.

Сжатие ведется до тех пор пока имеет смысл записывать это сжатие.

Таким образом, для реального сжатия информации используются комбинированные методы с целью оптимизации для любого варианта информации. Метод Лепел-Зива является методом кодирования со словарем, поэтому здесь важно выделять повторяющиеся последовательности (слова), которые помещаются в словарь. Рассмотрим один из вариантов метода, который имеет динамический словарь и предназначен для сжатия двоичных кодов большой длины. Эта задача характерна для компьютерных архиваторов, которые фактически сжимают двоичное представление графической, текстовой или кодовой информации, хранящейся в файлах. Например (метод LZ77) можно использовать специальные триплеты – тройки, содержащие 2 цифры и знак. Первая цифра триплета – начальный номер фрагмента, вторая цифра – длина фрагмента, знак – фрагмент, который помещают в конец строки. Триплетов может быть много и они распаковываются последовательно.

Пример: 000100110101010101001111101011001010101010011001100010 10

1) 000100110101010101001111101011001010101010011001100010 10
2) 000100110101010101001111101011001010101010011001100110(30,6,'')
3) 000100110101010101001111101011001010101010100110(4,6,'')(30,6,'')
4) 000100110101010101001111101011000(8,14,'')(4,6,'')(30,6,'')
5) 000100110101010101001111101010(6,4,0)(8,14,'')(4,6,'')(30,6,'')
6) 000100110101010101001111(7,5,'')(6,4,0)(8,14,'')(4,6,'')(30,6,'')

0001001101010101(3,6,1)(7,5,'')(6,4,0)(8,14,'')(4,6,'')(30,6,'')

9.Теория кодирования. Методы восстановления информации. Биты четности и дублирование информации.

Лекция №4

Раздел №1. Основы теории информации

Тема 1.2 Основы теории кодирования

Содержание: Проблема восстановления информации – биты четности, расстояние Хэмминга и коды Хэмминга, коды Рида-Соломона. Проблема криптографической защиты информации. Методы шифровки данных. Система PGP, технология электронной подписи.

Проблема восстановления информации – биты четности, коды Рида-Соломона

Одной из важнейших проблем теории кодирования является проблема передачи данных через канал, в котором на сигнал накладывается некий шум. На практике любая передача данных может привести к искажению передаваемой информации, т.е. к добавлению к ней случайной составляющей (шум). Именно случайный характер искажения и делает эту задачу сравнительно сложной.

Будем считать, что сигнал представлен в двоичном виде как последовательность нулей и единиц. На практике сигнал всегда разбит на сегменты определенной длины – пакеты. Длина пакета строго определена, поэтому искажения можно разбить на ряд вариантов – изменение в пакете 1 бита, изменение 2-х битов, изменение 3-х битов и т.д.. При этом вариант изменения 1 бита гораздо более вероятен, чем остальные (пример с вероятностью изменения бита p=0.001). Если пакет передан с искажением, то могут быть поставлены 2 проблемы:

а) Обнаружение «испорченных» битов.

б) Восстановление «испорченных» битов.

Первая задача как правило проще второй.

Проблема обнаружения и исправления ошибок в последовательностях двоичных сообщений, которые передаются по каналам связи решается путем увеличения избыточности кода. В зависимости от назначения и возможностей помехо защиты в системе кодирования различают коды самокорректирующиеся (позволяющие автоматически исправлять ошибки) и самоконтролирующиеся (позволяющие автоматически обнаруживать наиболее вероятные ошибки). Можно указать несколько вариантов реализации защиты от помех и сбоев в ЭВМ:

- Вариант «зеркала». Практически этот вариант предполагает дублирование информации (например на зеркальном сервере), тогда любой потерянный бит или байт можно восстановить с помощью его копии. Одновременное поражение одного и того же элемента сообщения в разных частях памяти при этом крайне маловероятна. При передаче сигнала в канале связи это вариант реализуется повторной передачей пакета.
- Вариант кода с проверкой на четность (бит четности) образуется добавлением к группе информационных двоичных знаков одного контрольного. Значение его выбирается таким образом, чтобы общее число единиц в слове было четным или нечетным. После чтения или записи данных в память проверяется, сохранен ли принцип записи. Обнаруживаются одиночные ошибки и групповые с нечетной кратностью. Как правило качество этого метода определяется числом битов группы(пакета) на 1 бит четности. Чем больше это значение, тем больше вероятность пропуска ошибки. Строгое математическое решение этой задачи приводит к построению сложных кодов типа кодов Рида-Соломона.
- Вариант кодов Хэмминга. Наиболее известные из самоконтролирующихся и самокорректирующихся кодов – коды Хемминга. Они основаны на выборе кодов специальным образом. В принципе это аналог предшествующего варианта, но построение таких кодов существенно проще, чем коды Рида-Соломона.

Коды Хэмминга дали возможность проанализировать теорию самовосстанавливающих кодов, но с точки зрения оптимальности эти коды не идеальны. Строгая постановка математической задачи построения помехозащитных кодов дает множество более качественных вариантов построения таких кодов. Примером могут служить коды Рида-Соломона, широко использующиеся в устройствах передачи и хранения данных для обнаружения и **исправления** как одиночных так и групповых ошибок. Область их применения необычайно широка - кодеры/декодеры Рида-Соломона можно найти и в ленточных запоминающих устройствах, и в контроллерах оперативной памяти, и в модемах, и в жестких дисках, и в CD-ROM/DVD приводах и т.д.

Как правило, теорию кодов Рида-Соломона представляют алгебраически как задачу размещения среди информационных битов особых контрольных битов (аналог битов четности). При изменении 1 или нескольких информационных битов результат алгебраического преобразования полученной последовательности битов (как двоичного числа) будет отличаться от необходимого. Практически алгоритм заключается в умножении информационного слова, представленного в виде полинома D, на некий полином G известный обоим сторонам, в результате чего получается кодовое слово C, опять-таки представленное в виде полинома. Декодирование осуществляется с точностью до наоборот: если при делении кодового слова C на полином G декодер внезапно получает остаток, то он может рапортовать наверх об ошибке. Соответственно, если кодовое слово разделилось нацело, его передача завершилась успешно. Таким образом, этот метод оптимизирует с точки зрения избыточности коды Хэмминга.

10.Теория кодирования. Методы восстановления информации. Коды Рида-Соломона. Расстояние Хэмминга. Коды Хэмминга.

Расстояние Хэмминга и коды Хэмминга

Для построения кодов Хэмминга необходимо создать такое алфавитное кодирование, при котором расстояние Хэмминга между кодами больше заданной величины. Расстояние Хэмминга – число бит, которые различаются между любыми двумя кодами буквы. Для вычисления расстояния необходимо сравнить все коды букв между собой и определить число бит, которые не совпадают в этих кодах. Затем среди найденных чисел выбирается наименьшее. Таким образом,

между любыми кодами букв число отличий в битах равно или больше расстояния. Теперь можно легко понять идею кодов Хэмминга:

- Если есть код с расстоянием 2, то при любом изменении кода буквы на 1 бит получится комбинация бит, которая не совпадает с другими кодами букв (так как они отличаются на 2 бита). Это дает основание обнаружить ошибку.
- Если есть код с расстоянием 3, то при любом изменении кода буквы на 1 бит можно не только обнаружить ошибку и определить какой из кодов (который отличается на 1 бит от «искаженного» кода) испорчен. Тогда ошибку можно исправить.

  Аналогично код с расстоянием 4 исправляет ошибки в 2-х битах и т.д

Минимальное кодовое расстояние (d) – это минимальное расстояние между двумя любыми кодовыми комбинациями в заданном коде. Для всех корректирующих кодов d>1. Для обнаружения ошибки кратности t требуется, чтобы d=t+1, а для ее исправления — d=t+2. С увеличением значения d растет корректирующая способность кода, которая количественно может быть выражена как вероятность обнаружения или исправления ошибок различных типов. Коды Хемминга имеют бóльшую относительную избыточность, чем коды с проверкой на четность, и предназначены либо для исправления одиночных ошибок (при d=3), либо для исправления одиночных и обнаружения без исправления двойных ошибок.

Первоначально эти коды предложены Хеммингом в таком виде, при котором контрольные знаки занимают особые позиции: позиция i-го знака имеет номер 2^{i-1}. При этом каждый контрольный знак входит лишь в одну проверку на четность. Рассмотрим код Хемминга, предназначенный для исправления одиночных ошибок, т. е. код с минимальным кодовым расстоянием d=3. Ошибка возможна и в одной из n позиций. Следовательно, число контрольных знаков, а значит, и число разрядов регистра ошибок должно удовлетворять условию превышения расстояния Хэмминга 2. В результате получаем, что коды Хэмминга дают возможность искать и исправлять ошибки при искажении сигнала при передачи данных. При этом, чем больше расстояние Хэмминга, тем больше избыточность, но при этом растут возможности по восстановлению информации.

Проблема криптографической защиты информации — не выносится на экзамен

Проблема криптографической защиты информации

Криптография (тайнопись) - это раздел математики, в котором изучаются и разрабатываются системы изменения письма с целью сделать его непонятным для непосвященных лиц. Известно, что еще в V веке до нашей эры тайнопись использовалась в Греции. В современном мире, где все больше и больше услуг предоставляется через использование информационных технологий, проблема защиты информации методами криптографии имеет первостепенное значение. Сегодня большая часть обмена информацией проходит по компьютерным сетям и часто (в бизнесе, военным и прочее) нужно обеспечивать конфиденциальность такого обмена. Теоретические основы классической криптографии впервые были изложены Клодом Шенноном в конце 1940-х годов. Простейшая система шифрования - это замена каждого знака письма на другой знак по выбранному правилу. Юлий Цезарь, например, заменял в своих секретных письмах первую букву алфавита на четвертую, вторую - на пятую, последнюю - на третью и т.п., т.е. A на D, B на E, Z на C и т.п. Октавиан Август заменял каждую непоследнюю букву алфавита на следующую, а последнюю на первую. Подобные шифры, называемые простой заменой или подстановкой, описаны в рассказах "Пляшущие человечки" А. К. Дойла, "Золотой жук" Э. По и других. Шифры простой замены легко поддаются расшифровке, при знании исходного языка сообщения, т.к. каждый письменный язык характеризуется частотой встречаемости своих знаков. Например, в английском языке чаще всего встречается буква E, а в русском - О. Таким образом, в шифрованном подстановкой сообщении на русском языке самому частому знаку будет с большой вероятностью соответствовать буква О. Вероятность будет расти с ростом длины сообщения.

Усовершенствованные шифры-подстановки используют возможность заменять символ исходного сообщения на любой символ из заданного для него множества символов, что позволяет выровнять частоты встречаемости различных знаков шифра, но подобные шифры

удлиняют сообщение и замедляют скорость обмена информацией. В шифрах-перестановках знаки сообщения специальным образом переставляются между собой, например, записывая сообщение в строки заданной длины и беря затем последовательность слов в столбцах в качестве шифра. Пример – квадрат Полибия:

1	2	3	4	5	6	
A	C	D	H	J	R	1
P	B	G	E	K	A	2
M	N	O	I	S	E	3
X	F	V	W	Z	T	4
M	Z	N	L	Q	Y	5
S	I	N	C	O	U	6

Здесь каждая буква заменяется на пару цифр – номер строки и столбца. Так как буквы могут повторяться. То можно выровнять или исказить частоту буквы, если она встречается более 1 раза.

Методы шифровки данных. Система PGP, технология электронной подписи

Шифровка данных – кодирование данных с помощью специального ключа так, чтобы декодировать сообщение при использовании ключа можно было бы во много раз быстрее, чем без ключа. Дешифровка – восстановление данных с помощью ключа. Расшифровка - восстановление данных без помощи ключа. Теоретически любое сообщение можно расшифровать, но усилия, которые при этом могут потребоваться (время и ресурсы ЭВМ), определяют криптостойкость метода шифровки. Чем больше эти усилия, тем крепче шифр.

В ЭВМ в основном используются методы алгебраического шифрования, когда информацию в двоичной форме делят на блоки, которые подвергают арифметическим операциям с использованием специальных цифр (цифровых ключей). Так как основная цель шифровки – затруднить расшифровку, то используются такие операции, которые обращаются при помощи очень сложных алгоритмов (например, нахождения всех простых сомножителей).

Как правило, системы алгебраического шифрования делят на системы с тайными и открытыми ключами. Система с тайным ключом основана на том, что размер и значение ключа в принципе не известно. Более сложна идея системы с открытым ключом. Фактически здесь 2 ключа – открытый и тайный. Обмениваясь открытой частью ключа, пользователи могут передавать друг другу информацию, которую легко декодировать с помощью своего тайного и чужого открытого ключа. Таким образом, собеседники отсекают от доступа к информации других пользователей, которые могут перехватить сообщение. Криптостойкость таких систем зависит прежде всего от длины ключа – число бит в цифре ключа. Во многих странах спецслужбы ограничивают использование ключей больше определенной длины.

Система PGP, технология электронной подписи

Первую и наиболее известную систему с открытым ключом разработали в 1978 году американцы Р. Ривест (Rivest R.), Э. Шамир (Shamir A.) и Л. Адлеман (Adleman L.). По их именам эта система получила название RSA. Пусть абоненты А и В решили организовать для себя возможность секретной переписки. Для этого каждый из них независимо выбирает по два больших простых числа и на их основе строят открытый и секретный ключи. Открытыми ключами можно обменяться. Тогда сообщение от А к В строится на основе операций над секретным ключом А и открытым ключом В, а дешифруется с помощью секретного ключа В и открытого ключа А. Если некто С знает открытые ключи А и В, то задача расшифровки для него будет во много раз дольше и сложнее чем для В.

На основе таких алгоритмов разрабатывается так же технология электронной подписи. В системе с электронной подписью сообщение необходимо "подписывать", т.е. явно указывать на отправителя из книги открытых паролей. Так как правильно расшифровать можно только то сообщение, которое зашифровано с помощью тайного ключа отправителя, то расшифрованное с помощью открытого ключа А послание можно считать посланным именно А. На этом алгоритме построена процедура удостоверения правильности цифровой подписи, которая во многих странах имеет те же юридические права, что и обычная финансовая подпись человека.

Лекция №5

Раздел №2. Методы теоретической информатики

Тема 2.1 Системы счисления и представление информации в ЭВМ

Содержание: Системы счисления. Математические операции в различных системах счисления. Системы счисления, используемые в ЭВМ и их особенности. Примеры решения задач на системы счисления.

11.Понятие системы счисления. Позиционные и непозиционные системы счисления. Примеры.

Системы счисления

Под системой счисления понимается способ представления любого числа посредством некоторого алфавита символов, называемых цифрами.

Система счисления должна обязательно удовлетворять следующим требованиям:

- *однозначная запись числа;*
- *определенность диапазона, который может занимать число (целые, рациональные, иррациональные, действительные);*
- *в системе должно быть определено конечное число математических операций, результат которых должен быть записан в этой же системе;*
- *для записи любого числа должно использоваться конечное число знаков.*

В зависимости от способа изображения чисел с помощью цифр **системы счисления делятся на позиционные и непозиционные.**

Позиционной называется система счисления, в которой количественное значение каждой цифры зависит от ее места (позиции) в числе. В таких системах значение единицы цифры каждого разряда имеет постоянный вес. Этот вес определяется позицией, которую разряд занимает по отношению к запятой. Так, в числе десятичной системы счисления 100,01 используются только цифры 0 и 1, но 1 стоящая слева от запятой, определяет количество сотен, а 1, стоящая справа от запятой, определяет количество сотых долей единицы.

В непозиционной системе счисления цифры не меняют своего количественного значения при изменении положения в записи числа. Примером является римская система. Из-за отсутствия однозначной связи между значениями цифры и ее позиции, большого количества цифр, необходимых для изображения чисел римская система счисления применяется редко. Другим примером непозиционной системы является система счета палочками (применяемая при обучении детей). Существуют так же смешанные системы, где смешиваются разные позиционные системы (или даже позиционные и не позиционные) пример – система записи даты/времени (года, месяцы, дни, часы, минуты, секунды).

В позиционной системе счисления числа записываются в виде последовательности цифр

$$A = a_{n-1}\, a_{m-2} \ldots a_1 a_0, a_{-1} a_{-2} a_{-3} \ldots a_{-n}. \qquad (1)$$

Позиции, пронумерованные индексами m и n, называются разрядами соответственно целой и дробной части числа (индекс m определяет число разрядов целой части числа, а индекс n – дробной). Обычно разряды имеют имена, например в десятичной системе это сотни, десятки, единицы, десятые, сотые и т.д.

Каждая цифра a_k (для целой части числа $0 \le k \le m-1$, для дробной - - $n \le k \le -1$) в записанной последовательности может принимать одно из значений некоторого алфавита символов, называемых цифрами. Обозначив количество цифр алфавита через q, имеем:

$$q - 1 \ge a_k \ge 0 .$$

Количество различных цифр (q), используемых для изображения чисел в позиционной системе счисления, называется основанием системы счисления. Поскольку цифра a_k соответствует количеству единиц k-го разряда, содержащихся в числе, то основание системы счисления q позиционной системы указывает, во сколько раз единица (k+1)-го разряда больше единицы k-го разряда.

Записанную выше последовательность цифр (1), соответствующую числу A, можно представить в виде полинома от основания q:

$$A = a_{m-1} \cdot q^{m-1} + a_{m-2} \cdot q^{m-2} + \ldots + a_1 \cdot q^1 + a_0 \cdot q^0 + a_{-1} \cdot q^{-1} + a_{-2} \cdot q^{-2} + \ldots + a_{-n} \cdot q^{-n}$$

или $\grave{A} = \sum_{i=0}^{m} a_i q^i + \sum_{t=-1}^{-n} a_t q^t$.(2)

Такая запись называется развернутой формой записи числа.

Каждая цифра данной записи занимает свой разряд. Положительные разряды и нулевой дают целую часть. Через «,» записывается десятичная часть. Величина разряда говорит, в какую степень надо возвести основание. Вес равен основанию в степени разряда.

Основание системы счисления определяет ее название: q = 10 – десятичная, q = 2 - двоичная.

В дальнейшем **для обозначения** используемой системы счисления будем число заключать в скобки и в индексе указывать основание системы счисления: $(A)_2$, $(A)_{10}$, $(A)_8$, $(A)_{16}$.

В математике рассматривается общая теория систем счисления, в которой доказана возможность создания систем счисления с самыми различными основаниями:

- Отрицательными основаниями.
- Рациональными основаниями типа q = N/M, где N и M – целые числа.
- Симметричными основаниями, где набор цифр например такой -2,-1,1,2.
- Иррациональными основаниями – например с основанием π или e.

Замечание: При использовании рациональных (в том числе целых) оснований не все числа могут быть представлены в этой системе счисления. Такие числа и называются иррациональными. То что они не могут быть записаны не значит что они не существуют (например число π). Аналогично при выборе иррационального основания все обычные рациональные числа не могут быть в ней записаны. Интересно, что и иррациональные числа тоже могут быть записаны в такой системе не все. Это говорит о том, что иррациональных чисел гораздо больше чем рациональных.

12.Представление чисел в различных системах счисления.

Представление чисел в различных системах счисления

Для представления числа в какой-то системе счисления необходимо:

- Установить алфавит цифр для данной системы счисления. Например для 8-й системы счисления необходимо иметь 8 цифр – 0, 1, 2, 3, 4, 5, 6, 7. Это общее правило – основание системы счисления не может быть цифрой этой системы счисления. Если основание системы больше 10, то обычных цифр нам уже не достаточно – используют большие латинские буквы A=10, B=11, C=12 и т.д. В принципе изображение цифр достаточно условно, можно было бы использовать произвольные значки, но обычные «арабские» (или правильнее индийские) цифры нам более привычны. Арабские цифры можно использовать и для рациональных оснований. Например записать число в системе с основанием 1/10 очень просто – нужно записать его в десятичной системе, а затем «перевернуть» - изменив знак разряда на противоположный. При этом нулевой разряд (единицы) и запятая останется на месте, все положительные разряды (целая часть) станут отрицательными (дробными) и наоборот. Число как бы симметрично отобразится относительно нулевого разряда. Аналогично можно поступать и с любой системой с основанием 1/N. Для других рациональных систем получение необходимой формы записи более сложно.
- Представить число в развернутой форме в данной системе счисления с использованием выбранных цифр.
- Записать число в виде последовательной записи цифр в порядке убывания разрядов и отделить дробную часть запятой.
- ***Замечание.*** Конечной десятичной дроби в другой системе счисления может соответствовать бесконечная (периодическая) дробь. В этом случае используется специальная форма записи – период дроби (повторяющиеся цифры дробной части) указывают 1 раз в круглых скобках. Пример: $1/3 = 0,(3)_{10}$

Наиболее простым способом получения записи числа считается следующий алгоритм:

- Записать число в десятичной системе.
- Выполнить преобразование целой части из десятичной системы в нужную.
- Выполнить преобразование дробной части из десятичной системы в нужную.
- Записать результат в виде единого числа.

Наибольшее распространение в ЭВМ получила двоичная система счисления. В этой системе используются только две цифры: 0 и 1.

В двоичной системе счисления любое число в соответствии с (1) и (2) может быть представлено последовательностью двоичных цифр

$$X = a_m a_{m-1} a_{m-2} \ldots a_1 a_0, a_{-1} a_{-2} a_{-3} \ldots, \text{ где } a_i \in \{0,1\}; \quad (3)$$

или суммой степеней числа 2, взятых с указанными в ней коэффициентами

$$X = a_{m-1} \cdot 2^{m-1} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 + a_{-1} \cdot 2^{-1} + a_{-2} \cdot 2^{-2} + \ldots + a_{-n} \cdot 2^{-n}. \quad (4)$$

Например, двоичное число

$$(1010{,}101)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 + 1 \cdot 2^{-1} + 0 \cdot 2^{-2} + 1 \cdot 2^{-3}$$

Применение двоичной системы счисления позволяет уменьшить общее количество аппаратуры вычислительных устройств и создает большие удобства для их проектирования, так как для представления в машине двоичного разряда числа может быть использован простой элемент, имеющий лишь два устойчивых состояния (реле, триггерные схемы). Кроме того, в двоичной системе счисления очень просто выполняются арифметические и логические операции.

В восьмеричной системе счисления используется восемь цифр: 0,1,2,3,4,5,6,7. Любое число в восьмеричной системе может быть представлено последовательностью цифр, где $a_i \in \{0 \div 7\}$ или суммой степеней числа 8.

$$(A)_8 = (157{,}34)_8 = 1 \cdot 8^2 + 5 \cdot 8^1 + 7 \cdot 8^0 + 3 \cdot 8^{-1} + 4 \cdot 8^{-2}. \quad (5)$$

В шестнадцатеричной системе счисления для изображения чисел употребляются 16 цифр от 0 до 15. При этом, чтобы одну цифру не изображать двумя знаками, введены обозначения для цифр, больших девяти, латинскими буквами: десять – A, одиннадцать – B, двенадцать – C, тринадцать - D, четырнадцать – E, пятнадцать – F.

$$(2AF)_{16} = 2 \cdot 16^2 + 10 \cdot 16^1 + 15 \cdot 16^0. \quad \textbf{(6)}$$

13.Системы счисления. Преобразование чисел в различных системах счисления. Методы преобразования чисел из десятичной системы счисления в двоичную.

Преобразование чисел в различных системах счисления

Рассмотрим общие **правила перевода числа** из системы счисления с основанием q в систему счисления с основанием S.

Прежде всего необходимо выделить основные варианты перевода:

1) Перевод из системы с основанием M в десятичную. Здесь нужно просто воспользоваться развернутой формой записи числа.

2) Перевод из десятичной системы в систему с основанием M. Здесь для перевода целой части используется правило деления, а для перевода дробной части правило умножения. В некоторых системах (например двоичной) есть упрощенные варианты перевода.

3) Перевод из системы с основанием K в систему с основанием M. Здесь рекомендуется использовать десятичную систему как промежуточную (K->10->M). В некоторых системах есть упрощенные варианты перевода (например правило триад, тетрад).

Метод деления

Метод деления применяется для преобразования целой части чисел. Ниже приведен его алгоритм.

- Разделим нацело десятичное число на новое основание. Если есть остаток, запишем его в младший разряд результата, а если нет – нуль.
- Повторяем деление до тех пор пока, пока окончательный результат не обнулится.

Метод умножения

Метод применяется для преобразования дробной части числа. Ниже приведен его алгоритм.

- Дробная часть умножается на новое основание и результат делится на целую и дробную часть.
- Целую часть (представляемую 1 цифрой) добавляем в дробную часть результата.
- Повторяем умножение новой дробной части и вновь получаем новую дробную цифру результата

Этот процесс повторяется до обнуления дробной части. Такой процесс при наличии периодической дроби бесконечен. Поэтому необходимо следить за результатом и при повторе дробной цифры в результате прерывать процесс с указанием полученного периода.

Необходимость в преобразовании чисел из одной системы счисления в другую возникают из-за того, что ЭВМ работает в двоичной системе счисления, программа записывается в шестнадцатеричной системе счисления, а исходные данные и результат удобнее представлять в десятичной системе счисления.

Обычно преобразования чисел из одной системы счисления в другую выполняются автоматически специальными устройствами машины. Однако в ряде случаев возникает необходимость и ручного перевода отдельного числа или небольшой группы чисел из одной системы счисления в другую. Таким образом, наибольший интерес представляет преобразование, использующее 2-ю систему. Рассмотрим такие примеры:

Совершенно очевидно, что двоичное число представляется последовательностью нулей и единиц – разрядов. Как и в любой позиционной системе, каждому разряду присвоен определенный вес – показатель степени основания системы. Веса первых 10 позиций представлены в таблице 1.

Таблица 4.1 *Веса первых десяти позиций двоичной системы счисления*

Позиция	9	8	7	6	5	4	3	2	1	0
Вес	512	256	128	64	32	16	8	4	2	1
Образование	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0

В двоичной системе счисления даже сравнительно небольшие числа занимают много позиций.

Как и в десятичной системе, в двоичной системе счисления для отделения дробной части используется точка (*двоичная точка*). Каждая позиция слева от этой точки также имеет свой вес – вес разряда дробной части числа. Значение веса в этом случае равно основанию системы счисления (т.е. двойке), возведенному в отрицательную степень.

Получить десятичное число из двоичного чрезвычайно просто.

Для этого необходимо записать двоичное число в развернутой форме и выполнить действия возведения в степень, умножения и сложения. В результате получим десятичное число.

Пример 1. *Перевод двоичного числа* $(101011{,}1)_2$ *в десятичное.*

$$(101011{,}1)_2 = 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} =$$

$$= 32 + 8 + 2 + 1 + 0{,}5 = (43{,}5)_{10}$$

Аналогично можно рассмотреть и перевод из другой системы:

$174{,}2_8 = 1*8^2+7*8^1+4*8^0+2*8^{-1} = 64+7*8+4+2/8 = 124{,}25_{10}$

$7C{,}4_{16} = 7*16^1+12*8^0+4*16^{-1} = 112+12+4/16 = 124{,}25_{10}$

Преобразование десятичных чисел в двоичные

Перевод из двоичной системы в десятичную несколько сложнее. Рассмотрим несколько алгоритмов.

Метод вычитания

Из десятичного числа вычитаются наибольшая возможная степень двойки, в соответствующий разряд двоичного числа записывается единица, если разность меньше следующей степени двойки, то далее записывается нуль, а если больше записывается единица и опять производится вычитание, и так до тех пор, пока исходное число не уменьшится до нуля.

Метод деления

Другим методом является так называемый метод деления. Он применяется для преобразования целой части чисел. Ниже приведен его алгоритм.

Разделим нацело десятичное число на двойку. Если есть остаток, запишем в младший разряд единицу, а если нет – нуль и снова разделим результат от первого деления. Повторим процедуру так до тех пор, пока окончательный результат не обнулиться.

Метод умножения

Метод применяется для преобразования десятичных дробей (чисел меньших единицы).

Число умножается на 2, если результат не меньше 1, то в старший разряд записывается единица, если нет, то нуль. Умножаем на 2 дробную часть результата и повторяем процедуру. И так далее до получения нужной степени точности или до обнуления результата.

Существует экономный способ преобразования целой части десятичного числа по схеме Горнера.

Существует схема Горнера, которая позволяет вычислить полином

$$\grave{A} = (((a_m p + a_{m-1})p + a_{m-2})p + ...) + a_0$$

по рекуррентной формуле $S_k=S_{k+1}\,p+a_{m-k}$, $S_0 = a_m$ (S_k – цифры числа)

Схему Горнера можно использовать и в любой другой системе, особенно если преобразование ведется из системы с большим основанием в систему с меньшим основанием.

Пример 2. *Перевод десятичного числа* $(149{,}5)_{10}$ *в двоичное методом вычитания.*

$(149{,}5)_{10} = (10010101{,}1)_2$

$-\ \underline{128 = 2^7}$

$21{,}5$

$-\ \underline{16 = 2^4}$

$5{,}5$

$-\ \underline{4 = 2^2}$

$1{,}5$

$-\ \underline{1 = 2^0}$

$0{,}5$

$-\ \underline{0{,}5 = 2^{-1}}$

0

Пример 3. *Перевод десятичного числа* $(149)_{10}$ *в двоичное методом деления.*

$(149)_{10}$	2	
148	74	2
1	74	37
0	36	18
1	18	9
0	8	4
1	4	2
0	2	1
0	0	0
1	← старший разряд	

$(10010101)_2=(149)_{10}$ ← ответ

Пример 4. *Перевод десятичного числа* $(0{,}5625)_{10}$ *в двоичное методом умножения.*

$0{,}5625 \cdot 2 = 1{,}1250 \to 1$

$0{,}125 \cdot 2 = 0{,}25 \to 0$

$0{,}25 \cdot 2 = 0{,}5 \to 0$

$0{,}5 \cdot 2 = 1{,}0 \to 1$

$0{,}0 \cdot 2 = 0 \to 0$

$(0{,}5625)_{10} = (0{,}1001)_2$

Пример на схему Горнера. Перевести двоичное число $(101111101)_2$ в десятичную систему счисления.

$S_0 = 1$

$S_1 = 1*2 + 0 = 2$

$S_2 = 2*2 + 1 = 5$

$S_3 = 5*2 + 1 = 11$

$S_4 = 11*2 + 1 = 23$

$S_5 = 23*2 + 1 = 47$

$S_6 = 47*2 + 1 = 95$
$S_7 = 95*2 + 0 = 190$
$S_8 = 190*2 + 1 = 381$
В результате получаем десятичное число 381.
Проверим этот результат: переведем число $(381)_{10}$ в двоичную систему.
Учтем, что $(10)_{10}=(1010)_2$
$S_0 = 3 = (11)_3$
$S_1 = S_0 * 10 + 8 = 11*1010 + 1000 =100110$
$S_2 = S_1 * 10 + 1 = 100110 * 1010 + 1 = (110000111)_2$

15.Математические операции в различных системах счисления. Примеры.

Математические операции в различных системах счисления

Сложение.

При сложении двух чисел в одной системе счисления используют правило столбика: числа записываются друг над другом, так чтобы места одинаковых разрядов совпадали.

Вычисления ведутся последовательно, начиная с самого наименьшего разряда. Суммирование ведется по правилам принятым в данной системе. При этом иногда производится перенос единицы в больший разряд. Для осуществления суммирования и вычитания необходима таблица сложения (подобие таблицы умножения). Мы привыкли автоматически вычислять сложение и вычитание в десятичной системе, но такая таблица нужна и там.

Таблица сложения в двоичной системе счисления
0+0=0
0+1=1
1+0=1
1+1=10

Пример 7. Найти сумму двоичных чисел $10000000100_{(2)}$ и $111000010_{(2)}$.

Решение

$$\begin{array}{cccccccccccc} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ + & & & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ \hline & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{1} & \mathbf{0} \end{array}$$

Вычитание.

Пример 8. Найти разность двоичных чисел $10010110_{(2)}$ и $1101011_{(2)}$.

Решение

$$\begin{array}{cccccccc} & 0 & 1 & & 0 & & 0 & 0 \\ & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ - & & 1 & 1 & 0 & 1 & 0 & 1 & 1 \\ \hline & & & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{1} & \mathbf{1} \end{array}$$

Умножение.

Если производить умножение и деление, то потребуется дополнительно и таблица умножения:

Пример: система с основанием 4

цифры	0	1	2	3
0	0	0	0	0
1	0	1	2	3
2	0	2	10	12
3	0	3	12	21

Используя эту таблицу и таблицу сложения можно выполнить умножение или деление по аналогии с десятичной системы (столбиком) так как эти методы годятся для любой системы счисления.

Особенный вариант умножение и деление имеет для 2-й системы:

Операцию производят в два действия:

1) поразрядное умножение одного числа на цифру другого

2) размещение полученных результатов в виде столбика со сдвигом разрядов до последующего сложения.

Удобно умножать по разрядно, на то число, которое имеет меньше разрядов.

Рассмотрим пример в системе с основанием 3. Нам необходимо знать правило умножения и сложения.

0+0=0 0*0=0
1+1=2 1*0=0
1+2=10 2*2=11
2+2=11

X $122_{(3)}$
$11_{(3)}$
_____
122
122
_____
$2112_{(3)}$

Деление.

Деление заменяется последовательным вычитанием из старших разрядов делимого произведения делителя на цифру в данной системе счисления.

p=4

0+0=0 2+2=10 0*0=0
1+1=2 2+3=11 1*0=0
1+2=3 2*2=10
1+3=10 3+3=12 2*3=12
3*3=21

_ 1023 / 11
33 33
_____
033
33
_____
0

Операции умножения и деления могут быть сведены в двоичной системе к двум последовательным операциям суммирования и сдвига, или вычитания и сдвига.

Операция сдвига – это смещение разрядов вправо или влево на единицу.

Поразрядное умножение на ноль или единицу обязательно ведет к сдвигу разряда на единицу и в случае нуля пропуска числа, в случае 1 – копируются числа.

Таким образом, в 2-й системе операции умножения и деления могут быть заменены на сложение (вычитание тоже заменяется сложением) и сдвиг разрядов). В ЭВМ, где все операции производятся с двоичными числами, таблица умножения не нужна, так как умножение производит операцию сдвига и суммирования. Остаются только операции суммирования: 0+0=0, 0+1=1.

14.Системы счисления, используемые в ЭВМ. Особенности систем счисления с основанием 2,8,16.

Системы счисления, используемые в ЭВМ и их особенности

В ЭВМ применяют позиционные системы счисления с десятичным основанием: двоичную, восьмеричную, шестнадцатеричную системы.

Основная система счисления двоичная, так как ЭВМ воспринимают вводимую в них информацию в виде двоичных символов: 0 и 1. Это обусловливается сравнительной простотой электронных элементов с двумя устойчивыми состояниями: ключевым режимом ламп и полупроводников, переключением состояний магнитных сердечников с прямоугольной петлей гистерезиса и т. д.

Восьмеричная и шестнадцатеричную системы введены в ЭВМ для упрощения работы с двоичными числами. Каждая восьмеричная цифра заменяет 3 двоичных, а каждая шестнадцатеричная – 4 двоичных. При такой замене операции преобразования практически не нужны. Особенно удобны шестнадцатеричные цифры, так 2 таких цифры как раз формируют 1 байт данных.

Десятичная система используется в ЭВМ для облегчения работы пользователей. Ввод и вывод цифр осуществляется с помощью десятичной системы. При этом может использоваться двоично-десятичная система.

Особенности систем счисления с основанием 2,8,16

В информатике не все системы счисления равноправны. Наибольшее значение имеет 2-ая система, так как вся информация ЭВМ хранится в 2-ой системе.

Двоичная система на ЭВМ технически наиболее удобно реализуема, другие системы реализовывать сложнее так как числа 2-ой системы имеют большую длину, поэтому для удобства используются системы основания которых есть 2 в степени, а именно в 8-ой и 16-ой.

Наиболее удобная в современном компьютере 16- ричная и обратно, существует экономное правило триад и тетрад.

Правило триад

Суть правила триад: каждому разряду восьмеричного числа соответствует ровно 3 разряда двоичного числа и это соответствие взаимнооднозначно. Разряды отчитываются от запятой влево – целая часть, вправо – дробная.

Пример 5. $\underline{001}\ \underline{101}\ \underline{111},\ \underline{101}\ \underline{110}_{(2)}$
1 5 7 5 6

при необходимости добавляются незначащие нули.

000 – 0
001 – 1
010 – 2
011 – 3
100 – 4
101 – 5
110 – 6
111 – 7

Правило тетрад

По аналогии с методом триад вводится метод тетрад, в котором установлено взаимнооднозначное соответствие между четырьмя двоичными разрядами и одним 16-ричным разрядом.

0000 – 0
0001 – 1
0010 – 2
0011 – 3
0100 – 4
0101 – 5
0110 – 6
0111 – 7
1000 – 8
1001 – 9
1010 – A
1011 – B
1100 – C
1101 – D
1110 – E
1111 – F

Пример 6. $\underline{0010}\ \underline{1110}\ \underline{1001},\ \underline{1110}\ \underline{1101}\ \underline{1010}_{(2)}$
2 E 9 E D $A_{(16)}$

6 F 3, 2 1 $A_{(16)}$ = 0110 1111 0011, 0010 0001 1010

Замечание: правила триад и тетрад – это частный случай более широкого правила, в котором связываются системы счисления с основанием P и Q, которые должны быть связаны следующим соотношением $p^n = Q$, более того используя это правило, можно делать промежуточные преобразования : $p^n = Q$ $p^m = T$. $Q \leftrightarrow T$, $Q \rightarrow P \rightarrow T$, $T \rightarrow P \rightarrow Q$.

Примеры решения задач на системы счисления

Пример 1. *Перевод двоичного числа* $(101011,1)_2$ *в десятичное.*

$$(101011,1)_2 = 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 + 1\cdot 2^{-1} =$$

$$= 32 + 8 + 2 + 1 + 0{,}5 = (43{,}5)_{10}$$

Аналогично можно рассмотреть и перевод из другой системы:

$174{,}2_8 = 1*8^2 + 7*8^1 + 4*8^0 + 2*8^{-1} = 64 + 7*8 + 4 + 2/8 = 124{,}25_{10}$

$7C{,}4_{16} = 7*16^1 + 12*8^0 + 4*16^{-1} = 112 + 12 + 4/16 = 124{,}25_{10}$

Пример 2. *Перевод десятичного числа* $(149,5)_{10}$ *в двоичное методом вычитания.*

$(149,5)_{10} = (10010101,1)_2$

$-$ $\underline{128 = 2^7}$

$21{,}5$

$-$ $\underline{16 = 2^4}$

$5{,}5$

$-$ $\underline{4 = 2^2}$

$1{,}5$

$-$ $\underline{1 = 2^0}$

$0{,}5$

$-$ $\underline{0{,}5 = 2^{-1}}$

0

Пример 3. *Перевод десятичного числа* $(149)_{10}$ *в двоичное методом деления.*

$(149)_{10}$	2
148	74 | 2
1	74	37 | 2
	0	36	18 | 2
		1	18	9 | 2
			0	8	4 | 2
				1	4	2 | 2
					0	2	1 | 2
						0	0	0
							1	← старший разряд

$(10010101)_2 = (149)_{10}$ ← ответ

Пример 4. *Перевод десятичного числа* $(0{,}5625)_{10}$ *в двоичное методом умножения.*

$0{,}5625\cdot 2 = 1{,}1250 \to 1$

$0{,}125\cdot 2 = 0{,}25 \to 0$

$0{,}25\cdot 2 = 0{,}5 \to 0$

$0{,}5\cdot 2 = 1{,}0 \to 1$

$0{,}0\cdot 2 = 0 \to 0$

$(0{,}5625)_{10} = (0{,}1001)_2$

Рассмотрим пример 5. Перевести двоичное число $(101111101)_2$ в десятичную систему счисления методом Горнера.

$S_0 = 1$

$S_1 = 1*2 + 0 = 2$

$S_2 = 2*2 + 1 = 5$
$S_3 = 5*2 + 1 = 11$
$S_4 = 11*2 + 1 = 23$
$S_5 = 23*2 + 1 = 47$
$S_6 = 47*2 + 1 = 95$
$S_7 = 95*2 + 0 = 190$
$S_8 = 190*2 + 1 = 381$
В результате получаем десятичное число 381.
Проверим этот результат: переведем число $(381)_{10}$ в двоичную систему.
Учтем, что $(10)_{10}=(1010)_2$
$S_0 = 3 = (11)_3$
$S_1 = S_0 * 10 + 8 = 11*1010 + 1000 =100110$

$$\mathbf{S_2 = S_1 * 10 + 1 = 100110 * 1010 + 1 = (110000111)_2}$$

Лекция №6

Раздел №2. Методы теоретической информатики

Тема 2.1 Системы счисления и представление информации в ЭВМ

Содержание: Представление информации в ЭВМ – текстовой, графической, мультимедийной. Представление чисел в ЭВМ. Прямой, обратный и дополнительный код. Числа с плавающей и фиксированной запятой. Мантисса и порядок числа. Нормализованный код.

16.Представление информации в ЭВМ. Текстовая и графическая информация.

18.Представление информации в ЭВМ. Графическая и мультимедиа информация.

Представление информации в ЭВМ – текстовой, графической, мультимедийной

Представление текстовой информации в ЭВМ

Любая информация в компьютере хранится в двоичном коде, следовательно, нужен способ кодировки, который бы преобразовывал знаки текста в двоичные коды. Такая система кодировки называется кодовой таблицей ЭВМ.

Для того, чтобы тексты переносились с одного компьютера на другой были установлены стандарты кодировок. Первоначально первый стандарт кодировки был 8-разрядным, 1 байт = 1 разряду, 256 знаков в таблице.

ASCII коды – американский стандарт. КОИ – российский стандарт.

Таблица ASCII коды содержит коды от 0 до 31 – управляющие коды, 32 – пробел. От 33 до 255 – видимые знаки: маленькие и большие буквы латинского алфавита, цифры десятичной системы 0 – 9, знаки препинания, математические операции. Выделяется место для национальной кодировки. Оставшиеся коды отданы для псевдографики и дополнительных знаков.

Псевдографика – элементы, с помощью которых можно изображать некоторые графические изображения. Она возникла из-за необходимости вывода на экран изображений таблиц в текстовом режиме.

Экран монитора может работать в двух режимах: текстовом и графическом. В текстовом режиме экран делится на строки и столбцы клеткой. Клетка такой сетки называется знакоместо и может содержать один знак кодовой таблицы. У знакоместа два цвета: цвет фона и цвет знака, что дает возможность представить экран в достаточно разнообразном виде.

Текстовый режим – один из вариантов графического, созданного для удобства вывода текстовой информации. Текстовый режим: 25 или 50 строк; 40 или 80 столбцов.

В результате оптимизации текстовое изображение выводится гораздо быстрее, чем графическое. Текстовый режим используется для набора и вывода текста, просмотра информации в текстовом режиме.

Развитие ЭВМ потребовало модернизации этой системы:

1) кодовой таблицы стало не хватать для представления всех необходимых знаков;

2) произошел переход к чисто графическому экрану, все системы стали графическими;

3) текст стали форматировать – нумерация страниц, отступы, шрифты.

В результате, из первой проблемы появились новые кодировки: ANSI – стандарт Windows и второй стандарт – двухбайтовый – Unicode. ANSI стандарт – модернизация ASCII кода, выброшена

псевдографика, а вместо неё добавили 30 других знаков. Unicode стандарт – кодируется двумя байтами, $2^{16} = 64*10^3$ знака.

По второй проблеме – наличие графического экрана потребовало встраивание в каждую систему ввода текста. По третьей проблеме – поскольку необходимо было хранить не только текст, но и форматирование к нему, то текстовые программы стали хранить текст совсем других форматов. Каждый редактор текста обзавелся своим форматом. Следовательно, появилась необходимость решения этой проблемы. Возникли 2 варианта:

1. Единый стандартный формат форматированого текста RTF – единый для разных систем. В дальнейшем появились и некоторые другие форматы – PDF и т.д.
2. Создание специальных гипертекстовых форматов, где текст и его формат хранится в одном текстовом файле с использованием специальной теговой формы (тег – блок текста с единым оформлением, где указаны начало тега и его конец). Первым вариантом такого формата стало создание TeX Давидом Кнутом, затем появились языки SGML и его подвиды HTML, DHTML, XML, XHTML и т.д.

Представление графической и мультимедийной информации в ЭВМ

Представление графической информации в ЭВМ.

В графическом режиме экран поделен на точки – пиксели, точки так же располагаются вдоль строк и столбцов, но их гораздо больше, чем знакомест. Количество пикселей по горизонтали и вертикали – это разрешение экрана. В настоящее время, это цифры порядка тысячи. Каждый пиксель может иметь свой цвет, в результате формируется мозаичное изображение, которое называется графическим. Графическая информация хранится в ЭВМ в растровом или векторном виде.

Растровое изображение – изображение в виде мозаики, для его хранения нужен массив содержащий код цвета каждого пикселя (пример формат BMP). Для записи растрового рисунка нужно задать местоположение каждого пикселя и его цвет. Для экономии объема растровый рисунок задается в виде прямоугольника пикселей. Указываются координаты верхнего левого угла ширина и высота. Далее последовательно блок за блоком цвета пикселей.

Векторное изображение – сжатие растровой графической информации с помощью выделения одноцветных элементов (отрезков – векторов, кривых – контуров, алгоритмических или фрактальных фигур). Поэтому существуют варианты векторного, контурного, алгоритмического и фрактального сжатия в векторной графике. Алгоритмическое сжатие по смыслу подобно общему алгоритмическому сжатию информации, когда вместо самой информации записывается алгоритм построения информации (в данном случае изображения). Фрактальное сжатие теоретически наиболее мощное, так как здесь используются специальные самоподобные изображения – фракталы. Такие изображения просто алгоритмизируются, но фактически могут быть очень сложными (напрмер так можно изображать деревья, ландшафт и т.д.). Таким образом, в векторной графике используется специальная упакованная (сжатая) запись изображения. Примеры – векторный формат графического пакета Corel Draw (cdr), файлы чертежных систем AutoCad и ArhiCad.

Простейший векторный формат представляет графическое изображение как набор отрезков прямых одного цвета, т.е. если у нас на одной прямой находятся хотя бы 4 пикселя, то для хранения такого отрезка достаточно знать направление, длину отрезка и один раз цвет. Следовательно, если цвет кодируется 3 байтами, то получаем выигрыш в сжатии информации. Ширина прямой – 1 пиксель.

Для коротких отрезков можно использовать отрезки по осям, в результате векторный формат позволяет сжать графическую информацию.

Контурный формат – одноцветные пиксели формируются в виде кривой. Такое расширение возможности с одной стороны увеличивает сжатие, а с другой стороны усложняет алгоритм.

Особенно удобными такие форматы оказались для представления шрифтов – набора знаков, изображающих текст. Особенностью шрифтов является масштабирование, растровые шрифты не масштабируются.

На сегодняшний день существует два основных стандарта Post Script и True Type.

Рисунок так же можно представить в виде алгоритма воспроизведения этого рисунка, это характерно для графических систем, которые работают с векторным форматом Auto Cad, Corel Draw.

Рисунки в этих системах хранятся в виде команд, которые могут воспроизвести этот рисунок. Это достаточно высокая система сжатия. К ним можно отнести формат *.wmf, который содержит

команды графического интерфейса Windows, что позволяет автоматически воспроизвести рисунок кнопки, окна и т.д.. Здесь хранят стандартное изображение.

Пример. Пусть разрешение экрана 640×480 . Каков объем файла для хранения экрана?

Для записи размера будем использовать $4_{числа} \times 4_{бт} = 16_{бт}$.

$640 \times 480 \approx 600 \times 500 = 3 * 10^5$.

черно-белое изображение: 1 бит на цвет. $3 * 10^5 \, бит / 8 \approx 4 * 10^4 \, бит \approx 40 Кб$

Палитра – это число цветов одновременно изображаемых на экране 4 цвета – 2 бита, $1стр = 80 Кб$. Палитра 8 цветов – 3 бита на цвет, $1стр = 120 Кб$.

16 цветов – 4 бита на цвет, $1стр = 160 Кб$.

256 цветов – 1 байт – 8 бит на цвет, $1стр = 300 Кб$.

2 байта на цвет High Color, 64000 цветов 600 Кб на страницу.

True Color – формируется как набор трех цветов: красный, зеленый и синий – RGB-формат, 3 байта на цвет. Наилучшее разрешение увеличивает объем на 1 страницу до 10 Мб растровое изображение. Такие объемы информации серьезно замедляли графическую работу монитора, а так же занимали слишком много места на диске, поэтому стали разрабатываться методы сжатия графических изображений – векторный формат.

Для сжатия графической информации был придуман физиологический способ: JPEG. Оказывается, человеческий глаз сглаживает контраст цветов, если точки находятся близко друг от друга. В результате, если рядом расположены разноцветные пиксели, то человек, сглаживая цвета, видит некий средний цвет. Следовательно, нет необходимости хранить цвет каждого пикселя, а надо объединять такие пиксели в цветовые пятна и делать их одного цвета.

Такое сжатие называется сжатием с потерей информации. На качество изображения это почти не влияет. Сжатие возможно от 10 до 20 раз в расширении *.jpg.

Представление видео и анимации в ЭВМ.

Такая информация требует разбиения показа по кадрам. Эксперименты показали, что глаз не замечает мелькание кадров, если их частота не меньше 24 кадра в секунду. Так 1 мин и 25 кадров в сек.

$60 * 25 = 1500$ кадров, $25 * 900 \approx 22 Ìá \;\; / ñ \;\; 1500 * 900 \approx 130 Мб$. 1 час $130 * 60 = 7{,}8 Гб$

Воспроизведение видео и анимации практически невозможно без сжатия информации. Для сжатия анимации придуман метод MPEG-(это инженерное решение). Его суть: хранить не все кадры, а только изменение от кадра к кадру. Такая технология дает сжатие от 10 до 100 раз.

Хранение звуковой информации в ЭВМ.

Звук – это непрерывный сигнал, поэтому необходима оцифровка – преобразование в дискретный сигнал. Оцифровку можно производить двумя способами:

- разложение непрерывного сигнала в спектр;
- выполнение дискретизации по уровню сигнала.

Звуковой сигнал – это непрерывный сигнал, который можно представить как набор гармонических сигналов (пример – нотная музыка), которые выражаются в тригонометрической форме, поэтому звук выгодно представлять в виде дискретного преобразования Фурье.

Амплитуды гармоник хранятся в специальном файле, который называется волновая таблица, так как спектр каждого источника звука уникален (существует теория Котельникова, которая позволяет оценить качество восстановления по дискретному спектру самой функции).

Использование волновых таблиц дает возможность моделировать звук и аранжировать музыку.

Во втором варианте непрерывный звуковой сигнал фиксируется в отдельных точках, количество этих точек называют частотой дискретизации. Значение амплитуды хранится как целое число. Следовательно, чем больше разрядов выделено на хранение такого числа, тем выше качество звука.

Количество возможных уровней для хранения амплитуды называется числом уровней дискретизации и определяет объем данных для хранения звукового файла.

По теории Котельникова частота дискретизации влияет на спектр, то есть частоты выше частоты дискретизации практически не запоминаются. Происходит искажение частотного характера звука. Количество уровней влияет на точность записи амплитуды, фактически это искажение громкости. Эти погрешности приводят к появлению постороннего шума.

Таким образом, при оцифровке звука приходится искать компромисс между увеличением числа уровней и частоты дискретизации с одной стороны и объемом звуковой информации с другой.

Таким образом, мы перечислили алгоритмы преобразования с помощью ЭВМ числовой, текстовой, графической, звуковой и видео информации, что практически включает в себя все виды информации, которой пользуется человек. Это делает ЭВМ универсальным средством хранения, передачи и обработки любой информации.

17.Представление чисел в ЭВМ. Прямой, обратный и дополнительный код. Числа с фиксированной и плавающей запятой, нормализованный код.

Представление чисел в ЭВМ

Представление чисел в ЭВМ. Прямой, обратный и дополнительный код

В зависимости от назначения и конструкции ЭВМ в них применяются три формы представления чисел:

1. представление целых чисел,
2. представление действительных чисел с фиксированной запятой (естественная)
3. представление действительных чисел с плавающей запятой (нормальная, полулогарифмическая).

Как правило, представление целых чисел имеет варианты прямого, обратного, дополнительного и двоично-десятичного варианта. Представление действительных чисел при этом формируется с помощью одного или 2-х целых чисел.

Прямой, обратный и дополнительный код

Прямой код чисел соответствует обычной записи чисел со своим знаком. Прямой код основан на представлении чисел в виде абсолютного значения с кодом соответствующего знака (для минуса выделена цифра 1, а для плюса 0).

$A_1 = +0{,}0101$, $[A_1]_{пр} = 00101$. $A_2 = -0{,}0101$, $[A_2]_{пр} = 10101$.

Нуль в прямом коде имеет два вариант +0 и -0+ 0 = 000…00 = $[0]_{пр}$ - 0 = 100…00 = $[0]_{пр}$

Прямой код используется в ЭВМ для выполнения арифметических операций над положительными числами, для записи положительных и отрицательных чисел в ЗУ, а также в устройствах ввода и вывода. Для упрощения выполнения арифметических операций в ЭВМ отрицательные числа представляются в обратном и дополнительном кодах. Чтобы представить двоичное отрицательное число в обратном коде, нужно поставить в знаковый разряд единицу, а во всех значащих разрядах единицы заменить нулями, а нули – единицами. Положительные числа во всех кодах записываются одинаково.

$A = -0{,}1010$. $[A]_{обр} = 10101$

Чтобы представить двоичное отрицательное число в обратном коде необходимо в знаковом разряде прямого кода сохранить единицу, а во всех значащих разрядах единицы заменить нулями, а нули – единицами (выполнить инверсию).

Примеры:

$A_1^{др} = -0{,}11001;\ [A_1^{др}]_{пр} = 111001;\ [A_1^{др}]_{обр} = 100110$

$A_2^{ц} = -10101;\ [A_2^{ц}]_{пр} = 110101;\ [A_2^{ц}]_{обр} = 101010$

Для представления отрицательного числа в дополнительном коде необходимо получить обратный код и добавить единицу в самом маленьком разряде. Знаковый разряд фактически это самый большой по номеру (располагается слева). Для кодирования знака используется знаковый разряд, который помещается, как правило, перед старшим значащим разрядом числа (крайний слева разряд). Нуль в этом разряде записывается для положительных чисел, а единица – для отрицательных.

Положительные числа в прямом, обратном и дополнительных кодах имеют одинаковую форму записи. Используя обратный и дополнительный код, можно операцию вычитания и сложения чисел различных знаков свести к арифметическому сложению кодов чисел. При этом появляется возможность оперировать со знаковыми разрядами так же, как и с цифровыми.

Двоично-десятичная форма представления целого числа формируется на основе двоичных тетрад. Десятичное число делится на отдельные десятичные цифры, каждая из которых записывается в отдельную двоичную тетраду (4 бита). Знак числа записывают в последнюю

тетраду с помощью цифр B и C. Эта форма удобна для быстрого вывода цифр на экран, где используются десятичные цифры.

Числа с плавающей и фиксированной запятой. Мантисса и порядок числа

Действительные числа в ЭВМ можно хранить с фиксированной запятой (например денежный формат) или с плавающей запятой.

При представлении чисел с фиксированной запятой положение запятой (точки) закрепляется в определенном месте относительно разрядов числа и сохраняется неизменным для всех чисел. Запятая может быть зафиксирована перед старшим значащим разрядом числа, после младшего или после любого из разрядов разрядной сетки. В первом случае в ЭВМ будут представлены только числа, которые по модулю меньше единицы (дробные числа), во втором – только целые числа, в третьем – смешанные дроби, в которых запятая разделяет целую и дробную часть. Так как запятая здесь фиксирована, то число можно хранить как целое.

Практически наиболее удобно в ЭВМ фиксировать положение запятой перед старшим значащим разрядом или после младшего. Фиксацией запятой после любого разряда мы пользуемся в повседневной жизни. Если для представления числа со знаком выделено n разрядов, то диапазон представления целых двоичных чисел в этом случае определяется выражением

$$1 \le \left| X_{\text{ф.з.}}^{\text{ц}} \right| \le 2^{n-1} - 1.$$

Наибольшее целое двоичное число, представленное в данном формате (n-разрядной сетке), будет содержать единицы во всех значащих разрядах, а наименьшее – единицу в (n-1)-м разряде, остальные нули. Код в знаковом разряде (0 или 1) будет определять знак представленного числа.

Диапазон представления в ЭВМ дробных двоичных чисел будет определяться неравенством

$$2^{-(n-1)} \le \left| X_{\text{ф.з.}}^{\text{др.}} \right| \le 1 - 2^{-(n-1)}$$

или приближенно

$$0 \le \left| X_{\text{ф.з.}} \right| < 1.$$

Наибольшее дробное двоичное число будет содержать единицы во всех значащих разрядах, а наименьшее – единицу только в младшем значащем разряде.

Использование представления чисел **с фиксированной запятой** (в естественной форме) позволяет упростить схемы устройств ЭВМ, повысить быстродействие вычислений, но создает определенные трудности при программировании. Недостатком представления чисел с фиксированной точкой является также и то, что относительная точность выполняемых расчетов зависит от величины чисел.

Если запятая в числе не фиксирована, то число можно хранить только в округленном (приближенном) виде. Числа $|X| < 1$, если запятая фиксирована после младшего значащего разряда, и $|X| < 2^{-(n-1)}$, если запятая фиксирована перед старшим значащим разрядом, не могут быть представлены n разрядами со знаком и принимаются равными нулю (**машинный нуль**). Если в знаковом разряде такого числа записан нуль, то представлен прямой код положительного машинного нуля, если единица, то отрицательного машинного нуля.

Двоичные числа $|X| > 2^{n-1} - 1$, если в ЭВМ представлены только целые числа, и $|X| \ge 1$, если в ЭВМ представлены дробные числа, также не могут быть представлены в принятой разрядной сетке. Такие числа выходят за пределы разрядной сетки влево, происходит **переполнение разрядной сетки**. В этом случае старшие разряды числа теряются, результат вычислений оказывается неверным. При переполнении машина, оперирующая числами с фиксированной запятой, обычно автоматически останавливается. Таким образом, число с плавающей запятой ограничено диапазоном между машинным нулем и максимальным по модулю числом.

Чтобы исходные данные и получающиеся в процессе решения задачи промежуточные и конечные величины не выходили за диапазон чисел, представленных в формате с фиксированной запятой, используется **масштабирование величин**, участвующих в вычислениях.

Мантисса и порядок числа

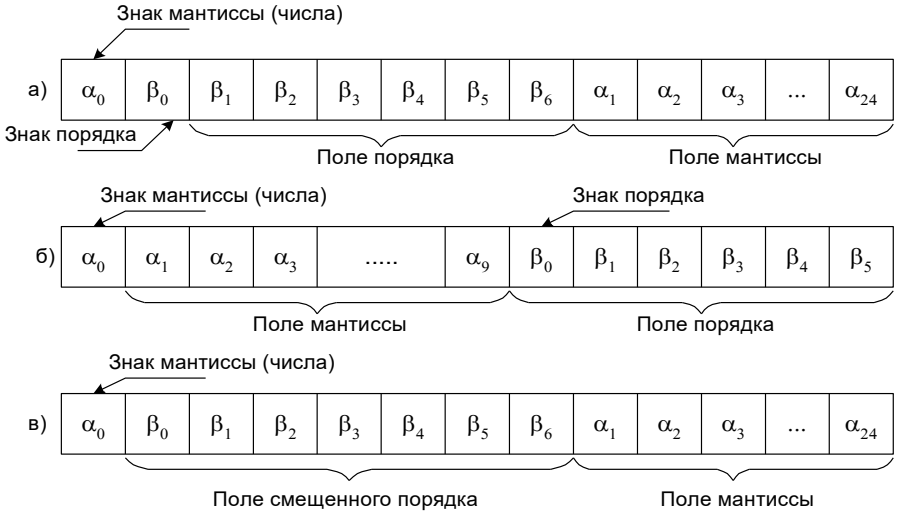
В ЭВМ, предназначенных для решения широкого круга задач, используется представление чисел **с плавающей запятой в нормальной, полулогарифмической форме**. В нормальной (полулогарифмической) форме число X представляется в виде $X = m_x \cdot q^{P_x}$, $|m_x| < 1$,

где m_x – мантисса числа X, определяющая значащие цифры числа; P_x – порядок (характеристика) числа X; q – основание системы счисления.

Порядок P может быть положительным или отрицательным целым числом (запятая при представлении порядка фиксируется после младшего разряда) или целым числом без знака, которое указывает положение запятой в числе. Мантисса m представляет собой правильную дробь, т.е. запятая при представлении мантиссы фиксируется перед старшим разрядом. Порядок P и мантисса m представляются в системе счисления с основанием q. Таким образом, для хранения числа с плавающей запятой нужно 2 целых числа – для мантиссы и для порядка. Для удобства создан специальный нормализованный формат, объединяющий оба эти числа в одном коде.

Нормализованный код

Рассмотрим форматы представления в ЭВМ чисел с плавающей запятой в нормализованном виде. Они содержат знаковые части и поля для мантиссы и порядка. Кодирование знаков остается таким же, как и при представлении чисел с фиксированной запятой.



Чтобы упростить операции над порядками, их сводят к действиям над целыми положительными числами путем использования так называемого **смещенного порядка**, который всегда положителен. Смещенный порядок $P_{см}$ образуется прибавлением к порядку P числа, равного 2^{K-1} (для двоичного представления порядка), где K – число разрядов порядка. В результате чего $P_{см}$ может принимать значения от 1 при максимальном по модулю отрицательном порядке до величины, равной (2^K-1) при максимальном по модулю положительном порядке. Для представления величины смещенного порядка используются все разряды поля порядка, включая знаковый.

Чтобы избежать неоднозначности представления чисел в форме с плавающей запятой, обеспечить наибольшую точность вычислений при заданной разрядности мантиссы, используют представление мантиссы в нормализованном виде. Модуль нормализованной мантиссы должен удовлетворять условию

$$\frac{1}{q} \le |m_X| < 1,$$

при котором старший разряд мантиссы в q-ичной системе счисления не должен быть равным нулю. Число с плавающей запятой, мантисса которого удовлетворяет этому условию, называется нормализованным. Двоичное число X будет нормализованным, если в старшем разряде мантиссы m_X всегда стоит единица, поэтому ее не хранят.

Установим диапазон изменения чисел, представленных с плавающей запятой.

Наибольшее нормализованное число в двоичной системе счисления, которое можно записать в разрядную сетку машины с плавающей запятой (n – число разрядов мантиссы со знаком, K – число разрядов порядка со знаком) будет иметь максимальную по модулю мантиссу и максимальный положительный порядок.

Лекция №7

Раздел №2. Методы теоретической информатики

Тема 2.2 Основы кибернетики, моделирования и теории искусственного интеллекта

Содержание: Моделирование как основной метод научного познания. Понятие модели, различные виды моделей, классификация моделей. Понятие об автоматах. Дискретный характер ЭВМ.

29.Моделирование как основной метод научного познания. Понятие модели, классификация моделей.

Основы кибернетики, моделирования и теории искусственного интеллекта

Моделирование как основной метод научного познания

Определение модели и процесса моделирования. Моделирование как основной метод научного познания.

Моделирование – присущий человеку метод познания. В случае моделирования человек заменяет реальный объект какой-то упрощенной моделью. Суть моделирования – упростить описание объекта до приемлемого уровня, чтобы описание касалось только тех свойств объекта, которые в данный момент нас интересуют. Модель – это объект, которым заменяют реальный объект. Главное условие, чтобы модель была проще объекта.

В таком определении любое абстрактное мышление можно считать моделированием. Однако можно сузить понятие моделирования, ограничив его только теми моделями которые могут быть воспроизведены с помощью ЭВМ. Такое моделирование называют компьютерным.

Хорошая модель должна в рамках изученных параметрах вести себя аналогично объекту следовательно, есть 2 основных проблемы моделирования:

- метод создания модели (реализация)
- методы проверки аналогичности объекта и модели (адекватность модели)

Каждая из этих проблем является своим разделом математических и прикладных дисциплин.

Анализ адекватности модели – это особая математическая проблема, которой занимаются такие разделы прикладной математики, как планирование эксперимента, математическая статистика, методы аппроксимации и численные методы.

Существует также раздел математики, который изучает модели. Здесь в отличие от компьютерного моделирования нет реального объекта. При этом вводятся некоторые важные понятия: эволюция модели, фазовое пространство, функция реакции модели и т.д.

Понятие модели, различные виды моделей, классификация моделей

Математическая модель представляется неким «черным ящиком», в котором имеются некие входные и выходные сигналы (параметры).

Математическая модель связывает функционально входные и выходные параметры. Совокупность значений входящих параметров определяют выходные параметры. Такая совокупность называется состоянием модели. Пространство состояний или фазовое пространство имеет размерность равную количеству выходных параметров. Каждая точка в этом пространстве – это одно составляющее модели. В фазовом пространстве поведение модели будет описываться некой кривой состояния. Если это изменение связано с течением времени, то такая кривая называется эволюцией модели. Кривая поведения модели называется фазовым портретом модели.

Классификация моделей – информационные и предметные, компьютерные и аналоговые, аналитические, информационные и имитационные. Математические модели : аналитические, информационные и имитационные. Структурные(дискретные) и непрерывные модели. Уровни сложности структурных моделей – макроуровень и метауровень. Модели так же делятся на детерминированные или стохастические модели.

В компьютерном моделирование принято выделять следующие типы моделей:

1. Физическая модель – реальный физический объект (например, лабораторная установка)

2. Макет – физический объект, который отражает отдельные свойства объекта в определенном масштабе.

3. Аналоговая модель – модель, которая использует процессы аналогичные процессам объектов. Часто под аналогичной моделью понимают модель созданную с помощью специальных электрических схем.

4. Математическая модель

5. Информационная модель

6. Имитационная модель (особый вид математической модели)

Существует классификация моделей по их сфере применения – это модель в химии, в физике, в биологии, в экономике, экологии.

Часто каждый из перечисленных типов моделей имеет свои существенные особенности, что делит компьютерное моделирование на разделы: аналоговое моделирование, математическое моделирование, информационное моделирование, имитационное моделирование.

Возможно также рассмотрение моделирования по области применения: моделирование в химии, в биологии.

Классификация по сложности моделей.

1. Деление на дискретные и непрерывные, детерминированные и стохастические, различные варианты сетевых представлений. Следовательно, можно строить комбинации из разных вариантов пар. Так есть непрерывные и детерминированные, дискретно детерминированные, непрерывно стохастические и дискретно стохастические. Отдельно рассмотрим более сложные модели, которые называются сетевыми (сетевые детерминированные, сетевые стохастические и т.д.).
2. Модели делятся по сложности на уровни сложности:

- микроуровень – непрерывная модель
- макроуровень – дискретная модель, которая состоит из нескольких моделей микроуровня.
- метауровень – модель, которая состоит из нескольких моделей макроуровня. Иногда модели 2,3 уровня объединяют в структурные модели.

Деление моделей по таким типам определяется различием методов их создания и исследования. Например, работу станка в цехе можно отнести к микроуровню. Работу цеха относят к макроуровню, т.к. в цехе работают несколько станков. А работу всего завода можно можно отнести к метауровню.

Математическое моделирование

Математическая модель – совокупность уравнений, неравенств, соотношений которые однозначно связывают входные и выходные параметры. Существуют различные варианты матмоделей в том числе:

а) Аналитическая модель, которая представлена в аналитических формах. Иногда именно ее называют математической моделью. В одних случаях аналитическая модель имеет решение явно, в других случаях необходимо использовать численные методы.

б) Имитационные модели реализуются только с помощью компьютера или специальных аналоговых машин. Аналоговая вычислительная машина (АВМ) – система, которая собрана из специальных электронных блоков, в отличие от обычных ЭВМ в них использована не цифровая электроника, а аналоговая. В цифровой электронике сигнал дискретный, а в аналоговой сигнал непрерывный. В аналоговой электронике придуманы схемы, позволяют складывать аналоговые сигналы, вычитать, получать и преобразовывать их спектр, дифференцировать, интегрировать. Это аналоговые сумматоры, инверторы, интеграторы. Комбинируя цифровые и аналоговые ЭВМ (гибридные системы), можно добиться получения численного решения практически любой математической задачи. Поэтому для имитационного моделирования существует 3 варианта:

4. использование аналогового вычисляющего устройства;
5. использование гибридного устройства;
6. использование цифровой ЭВМ.

В компьютерном моделировании (КМ) широко используются методы статистического моделирования и моделирования стохастических систем. Поясним разницу между этими понятиями.

О. Статистическое моделирование – моделирование с использованием случайных процессов и явлений.

О. Стохастическое моделирование – моделирование систем, имеющих случайные параметры или процессы.

Разница между этими вариантами моделирования в том, что методами статистического моделирования можно в принципе изучать детерминированные модели.

Существует 2 варианта использования статистического моделирования:

- в стохастических моделях может существовать случайные параметры или взаимодействия. Связь между параметрами носит случайный или очень сложный характер.
- для особых вариантов детерминированных моделей могут использоваться статистические методы. Практически всегда используются статическое моделирование в имитационных моделях.

О. Модели в которых между параметрами существует однозначная связь и нет случайных параметров называются *детерминированными.*

Понятие “детерминированное” возникло из детерминированных процессов. Детерминированные процессы – процессы, в которых всякие явления определены законами. Человек считал все процессы в природе детерминированными, однако со временем были обнаружены случайные процессы. Исследование таких процессов показало, что они бывают 2-х типов:

а) Случайные по своей природе процессы;

б) Очень сложные детерминированные процессы;

Доказана центральная теорема теории вероятности, в соответствии с которой сложение различных процессов увеличивает случайный характер. Так, если сложить совершенно разные последовательности, не связанные между собой, то в результате в пределе процесс будет стремиться к нормальному распределению, но известно, что нормальное распределение соответствует независимым событиям (наиболее случайным процессам), следовательно, объединение детерминированных событий в пределе ведет к их случайности.

Поскольку в природе не существует совершенно чисто детерминированных процессов, всегда есть смесь детерминированных и случайных процессов. Действие случайного фактора называется “шумом”. Источники шума в природе – очень сложные детерминированные процессы (броуновское движение молекул) и теоретически случайные процессы (радиоактивность, квантовые эффекты).

В имитационном моделировании часто сложные процессы заменяют случайными, следовательно, для того чтобы сделать имитационную модель, нужно научиться моделировать случайные процессы методами статического моделирования. Представляют случайные процессы в КМ последовательностью случайных чисел, величина которых случайно меняется.

Основой статистического моделирования является моделирование последовательностей случайных чисел.

Геометрическим моделированием называются методы моделирования:

- геометрического изображения объекта
- поверхности объекта
- среды объекта (освещение и т.д.)
- моделирование цветов, оттенков

Сущность геометрического моделирования – представить объект геометрически правильно в двумерной или трехмерной системе координат. Фигура может быть представлена в виде набора графических примитивов (отрезков, дуг, окружностей, эллипсов, сплайнов). Так как изображение формируется в системе координат, то задается некий массив точек для привязки примитивов, который хранит расположение в пространстве между ними.

Информационная модель – это совокупность большого объема информации об объекте и методов ее обработки. Для информационной модели важна и структура информации и методы ее

обработки. Характерный пример информационной модели – прогноз погоды. Здесь существует множество датчиков, приборов станций, которые собирают и передают в единую базу информацию о температуре, влажности, ветре в разных точках. Аналитическая связь между этими величинами теоретически очень сложна, поэтому разработали методы такой обработки информации, которые дали бы возможность с какой- то долей вероятности предсказать поведение системы.

Если рассматривать информационную модель как совокупность большого объема информации об объекте, то возникает вопрос о характере используемой информации. Человечество хранит большую часть своих актуальных данных в виде книг, рисунков, чертежей и других видах носителей. Такая информация должна быть преобразована для ввода в ЭВМ, а зачастую и значительно переработана. Принято называть вид такой не структурированной текстово-графической информации называть вербальной моделью. В отличии от нее понятие информационной модели предусматривает наличие организации четкой структуры данных, удобной для хранения, обработки и ввода в ЭВМ.

Классификация информационных моделей по структуре

Для классификации информационных моделей (ИнМ) важно прежде всего деление на текстографические и фактографические ИнМ. Первый тип ИнМ использует информацию представленную в стандартной текстовой форме. Фактографическая же информация имеет четкую внутреннюю структуру и организацию. Обычно такого рода совокупность информации, организованной по определенной структурной модели принято называть базой данных. С точки зрения структуры базы данных существуют стандартные классификации:

- сетевая
- иерархическая
- реляционная
- объектно-ориентированная
- многомерная

В настоящее время очень развиваются модели знаний, которые могут хранить не только данные, но и знания.

Классификация информационных моделей по методам обработки

Обработка большого количества данных требует использования специальных подходов к организации работы с ИнМ :

1. Использование специализированных методов хранения и доступа к БД.
2. Использование динамических структур (списки, индексирование и т.д.).
3. Использование специальных, быстрых алгоритмов сортировки, поиска, выборки.
4. Использование стандартных методов аппроксимации (аппроксимационные полиномы, сплайны и т.д.).
5. Использование специальных статистических методов обработки данных OLAP.
6. В настоящее время наиболее перспективным считается использование специальной интеллектуальной обработки ИАД.

30.Понятие автомата. Дискретный характер ЭВМ.

Понятие об автоматах

Автомат – термин, который активно используется в теории автоматов и автоматических систем. Характерно, что в процессе исследования автоматов образовались 2 отличных подхода к этому понятию – автомат с точки зрения математика и автомат с точки зрения инженера.

С точки зрения математика автомат – алгебраическая система, состоящая из 3 множеств и 2 отношений (функций) заданных на этих множествах. Множества – X – входных сигналов,Y – выходных сигналов, Z – состояний. Как правила эти множества конечны (тогда автомат конечный), но теоретически рассматриваются бесконечные автоматы. Отношения :

- Y=E(X,Z) – связывает входные и выходные сигналы (с учетом состояний) – выходная функция автомата.

- Y=T(Z) – связывает выходные сигналы и состояние автомата – переходная функция автомата.

С точки зрения инженера автомат рассматривается как дискретная или непрерывная модель, связывающая входные Xi и выходные сигналы Yj. Здесь, как в моделировании активно используется модель черного ящика, у которого есть лампочки, сигнализирующие о внутреннем состоянии:

Если фиксировать какое-то состояние, то можно снять зависимость выходных сигналов Yj от входных Xi. Этот набор зависимостей называют выходными характеристиками автомата. Если фиксировать входные сигналыXi и наблюдать как меняются выходные сигналы Yj в зависимости от состояний Zk, то можно построить переходные характеристики автомата.

Хотя автоматы могут иметь непрерывные сигналы, но наиболее важно изучать дискретные автоматы, у которых входные и выходные сигналы принимают только дискретное число состояний, дискретно и число состояний Zk. В наиболее важных приложениях число состоянии конечно – конечные автоматы.

Состояние конечного дискретного автомата описывается тройкой (Xi, Yj, Zk). Если внести все эти варианты в таблицу то она будет 3-х мерной. Однако можно ее представить набором таблиц выходной и переходной характеристик. Если автомат детерминирован, то таблицы однозначно определяют работу автомата (пример – машина Тьюринга, конечный детерминированный автомат с конечной памятью, память – число внутренних состояний). Однако существуют и вероятностные автоматы, переход которых из состояния в состояние определяется с какой-то вероятностью. В этом случае у таблиц автоматов появляется еще одна размерность – вероятность.

Иногда поведение автоматов представляют с помощью графа. Вершинами графа являются его состояния (описываемые Zk), ребрами – возможные входные сигналы. Под действием этих сигналов состояния меняются, что показывается направлением ребра (дуги). Это ориентированный граф состояний. Часто к вершинам у входа в них дуг записывают значения выходных сигналов, тогда граф состояний полностью описывает поведение автомата.

Дискретный характер ЭВМ

Представление ЭВМ как дискретного автомата.

ЭВМ как система обработки цифровой\дискретной информации представляется в виде автомата, перерабатывающего дискретную информацию и меняющего свои внутренние состояния лишь в допустимые моменты времени. Математической моделью при этом подходе является конечный автомат, характеризующийся конечным множеством X входных сигналов, конечным множеством Y выходных сигналов, конечным множеством Z внутренних состояний, начальным состоянием $Z_0 \in Z$; функцией переходов $g(z,x)$; функцией выходов $\nu(z,x)$. Автомат функционирует в дискретном автоматном времени, моментами которого являются такты (примыкающие друг к другу равные интервалы времени, каждому из которых соответствуют постоянные значения входного и выходного сигналов и внутренние состояния).

Здесь множество входных сигналов – двоичные данные, которые поступают на процессор (в виде единого множества битов – слова процессора). Выходные данные – результат работы процессора. ЭВМ обладает памятью (регистры процессора, КЭШ-память, оперативная память, внешняя память), что аналогично использованию множества внутренних состояний системы.

ЭВМ относится к программируемым автоматам, которые имеют одну или множество программ. При наступлении определенных условий программа запускается и определяет один из возможных вариантов выходных данных. С точки зрения теории автоматов это эквивалентно наличию определенного числа особых внутренних состояний (достоинство программы только в том, что одна программа может определять огромное число состояний, экономя тем самым память автомата).

Это описание характерно для моделей так называемых дискретных систем массового обслуживания. В этих системах есть поток заявок, которые обслуживает специальная система. Время обслуживания может быть разное или однородное. Заявки могут образовывать очереди или получать отказы от обслуживания, если система занята. Система может иметь несколько каналов обслуживания (обслуживать несколько заявок одновременно). Практически, все возникающие в ЭВМ процессы обработки цифровой информации могут быть представлены в таких моделях

Лекция №8

Раздел №2. Методы теоретической информатики

Тема 2.2 Основы кибернетики, моделирования и теории искусственного интеллекта

Содержание: Кибернетика как наука об управлении и управляющих системах. Системы автоматического управления. Основные задачи искусственного интеллекта. Понятие о методах представления знаний.

34.Понятие о кибернетике. Система управления и ее реализация. Обратная связь в системе управления. Системы прогноза.

Кибернетика как наука об управлении и управляющих системах

Кибернетика — наука о процессе управления, построении управляющих систем и автоматизации управления каким-то объектом управления (ОУ).

Управлящая система = ОУ+Система Управления (СУ). СУ бывают автоматизированными (жесткая) и гибкими.

Основой теории кибернетики является понятие - гомеостат. Гомеостат — система, которая поддерживает некое устойчивое состояние равновесия. Пример гомоестата — термостат, который поддерживает постоянную температуру. Другой пример гомеостата — автопилот. Для автопилота равносие заключается в отсутствии отклонений от некой функции — полетного курса. Это все примеры автоматизированных или жестких систем управления, использующих простую обратную связь.

Кибернетика установила, что управление присуще только системным объектам. Общим в процессах является его антиэнтропийный характер, направленность на упорядочение системы.

О. Теóрия управлéния — наука о принципах и методах управления различными системами, процессами и объектами. Основами теории управления являются кибернетика и теория информации. Суть теории управления: на основе системного анализа составляется математическая модель объекта управления (ОУ), после чего синтезируется алгоритм управления (АУ) для получения желаемых характеристик протекания процесса или целей управления. Данная область знаний хорошо развита и находит широкое применение в современной технике. В социально-экономических системах теория управления посвящена приемам и методам анализа, прогноза и возможностям регулирования деятельности различных общностей людей (мирового сообщества, региональных объединений, наций, общественно-хозяйственных групп).Теория управления, как любая наука, имеет свою методологию и методическое обеспечение. Однако в области естествознания и техники теория управления имеет гораздо больше успехов, чем в социально-экономической сфере, где, очевидно, действует ограничение, вытекающее из принципа — «система не может объяснить саму себя».

Первое самоуправляемое устройство было построено Ктезибием из Александрии (примерно в 250 году до н.э.). Его водяные часы использовали сифон как регулятор потока воды. До этого изобретения считалось, что только живые существа способны модифицировать свое поведение в ответ на изменения в окружающей среде. Следующим шагом в развитии саморегулирующихся систем управления с обратной связью стали регулятор паровой машины Джеймса Уатта (1736—1819), и термостат Корнелиса Дреббеля (1572—1633). Математическая теория устойчивых систем с обратной связью была разработана в XIX веке. В связи с развитием паровых машин, потребовались регуляторы, которые могли бы автоматически поддерживать установившийся режим их работы. Универсальность математических методов, полученных в данной теории, перевела ее в

область наук, занимающихся изучением абстрактных математических объектов, а не их конкретных технических реализаций. Родоначальником непосредственно «**математической теории управления**» можно считать Александра Михайловича Ляпунова — автора классической теории устойчивости движения (1892).

Системы автоматического управления

Теория автоматического управления (ТАУ) появилась во второй половине 19 века сначала как теория регулирования. Широкое применение паровых машин вызвало потребность в регуляторах, то есть в специальных устройствах, поддерживающих устойчивый режим работы паровой машины. Это дало начало научным исследованиям в области управления техническими объектами. Оказалось, что результаты и выводы данной теории могут быть применимы к управлению объектами различной природы с различными принципами действия. В настоящее время сфера ее влияния расширилась на анализ динамики таких систем, как экономические, социальные и т.п. Поэтому прежнее название “Теория автоматического регулирования” заменено на более широкое - “Теория автоматического управления”.

Управление каким-либо объектом (объект управления будем обозначать ОУ) есть воздействие на него в целях достижения требуемых состояний или процессов. В качестве ОУ может служить самолет, станок, электродвигатель и т.п. Управление объектом с помощью технических средств без участия человека называется *автоматическим управлением*. Совокупность ОУ и средств автоматического управления называется *системой автоматического управления (САУ)*.

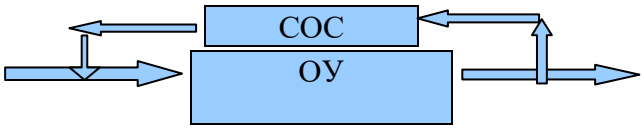
Основной задачей автоматического управления является поддержание определенного закона изменения одной или нескольких физических величин, характеризующих процессы, протекающие в ОУ, без непосредственного участия человека. Эти величины называются *управляемыми величинами*.

Системы управления разделяют на два больших класса:

Автоматизированные системы управления (АСУ) — с участием человека в контуре управления;

Системы автоматического управления (САУ)— без участия человека в контуре управления

Общая схема АСУ



ОУ- объект управления, СОС — Система обратной связи.

СОС передает на вход АСУ корректирующий сигнал, который пропорционален отклонению выходных параметров ОУ от состояния равновесия. Существуют 2 варианта обратной связи:

1. Положительная ОС. В этом варианте отклонение от равновесия будет увеличиваться при подаче корректирующего сигнала. Тогда система быстро уходит от состояния равновесия. Пример — мультивибратор, который таким образом генерирует импульсный сигнал.
2. Отрицательная ОС. Здесь корректирующий сигнал возвращает систему к равновесию. Этот вариант и реализуется в гомеостате.

Основная проблема АСУ — запаздывание ОС. Например — учебное заведение строит обратную связь по результатам контрольных работ, но коррекция будет с запозданием. Занятия уже завершены, а ученики не имеют знаний. Системы в которых запаздывание критично делают гибкими СУ. В таких СУ вводится система прогноза, которая должна заранее определять необходимость коррекции.

Примеры современных методов управления:

- Нелинейное управление
- Теория катастроф
- Адаптивное управление
- Построение оптимальных робастных регуляторов

- Игровые методы в управлении
- Интеллектуальное управление

САУ (системы автоматического управления)— формируются в основном в варианте жесткой системы АСУ, но существуют и варианты гибких САУ. В основном гибкие САУ используют в сложных системах (социально-экономические системы, неоднородные системы, структурные сложные системы). Разработка ПО для САУ в настоящее время остается популярнейшей областью информатики.

Замкнутые САУ

В замкнутых системах автоматического регулирования управляющее воздействие формируется в непосредственной зависимости от управляемой величины. Связь входа системы с его выходом называется обратной связью. Сигнал обратной связи вычитается из задающего воздействия. Такая обратная связь называется *отрицательной*.

Разомкнутые САУ

Сущность принципа разомкнутого управления заключается в *жестко* заданной программе управления. То есть управление осуществляется «вслепую», без контроля результата, основываясь лишь на заложенной в САУ модели управляемого объекта. Примеры таких систем: таймер, блок управления светофора, автоматическая система полива газона, автоматическая стиральная машина и т.п

В качестве примеров САУ можно привести:

- Системы дискретного действия или автоматы (торговые, игровые, музыкальные, узлы ЭВМ).

Системы стабилизации уровня звука, изображения или магнитной записи. Это могут быть управляемые комплексы летательных аппаратов, включающие в свой состав системы автоматического управления двигателя, рулевыми механизмами, автопилоты и навигационные системы.

35. Основные задачи искусственного интеллекта

Основные задачи искусственного интеллекта

Задача разработки ИИ может быть сформулирована как разработка компьютерной модели, которая обладала бы всеми способностями к мышлению и высшей нервной деятельности свойственной человеку. В процессе обсуждения этой проблемы возник вопрос о том, что именно относить к таким способностям мышления. Мнения в целом разделились.

Знаменитый ученый – информатик, Алан Тьюринг, предложил простой тест на «разумность» компьютерной модели. Необходим эксперимент с привлечением нескольких человек и этой компьютерной программы. В рамках эксперимента люди и программа общаются между собой с помощью общего интерфейса, например текстовых сообщений. Все участники – люди, стараются по результатам общения определить, кто из собеседников не человек. Если есть программа, которую не удается отличить от человека, то она действительно адекватно моделирует интеллект.

В целом, решения такой задачи достичь не удалось. Более того, появилась гипотеза (утверждение, которое пока не имеет доказательства) о том, что человек не в состоянии создать программу по сложности не уступающую ему самому. В аксиоматике была сформулирована гипотеза Геделя (на основе доказанных теорем теории множеств и логических исчислений) о том, что для полного и непротиворечивого описания системы необходимо использование средств, выходящих за рамки данной аксиоматической системы. Распространяя эту гипотезу на ИИ, получаем, что для создания полной модели человеческого разума необходим «сверхразум», который обладает средствами недоступными человеку. С другой стороны, возможности современного человека все более определяются коллективным разумом всего человечества. Возможно, коллективный разум человечества может выступать в роли «сверхразума». Интересна так же идея «киборга» - модернизации человеческого разума. Биологически человек развивается, но сравнительно медленно. Интеграция средств ИИ и человека может дать новые возможности для мышления человека, а в перспективе для создания «человеко-машинного разума» - киборга.

Таким образом, не добившись результата в главном, разработки ИИ преуспели в решении частных задач. Каждая такая задача характеризует одну из сторон человеческого мышления или возможностей, но не дает целостной модели (например, шахматные программы уже играют на уровне лучших гроссмейстеров мира, но полностью овладеть возможностями шахматистов пока не удалось).

Работы по созданию интеллектуальных систем ведутся в двух направлениях:

- Сторонники первого направления исходят из положения о том, что искусственные системы не обязаны повторять в своей структуре и функционировании структуру и протекающие в ней процессы, присущие биологическим системам. Важно лишь то, что теми или иными средствами удается добиться тех же результатов в поведении, какие характерны для человека и других биологических систем. Это направление получило название **информационное**. В рамках этого направления в основном разрабатывают реально действующие методы обработки информации и программы, моделирующие те или иные стороны интеллекта. Развитие информационного направления шло от задачи о рационализации рассуждений путем выяснения общих приемов быстрого выявления ложных и истинных высказываний в заданной системе знаний. Способность рассуждать и находить противоречия в различных системах взаимосвязанных ситуаций, объектов, понятий является важной стороной феномена мышления, выражением способности к дедуктивному мышлению. Результативность информационного направления бесспорна в области изучения и воспроизведения дедуктивных методов мышления. Для некоторых практических задач этого достаточно. Информационное направление - наука точная, строгая, вобравшая в себя основные результаты изысканий кибернетики и ее математическую культуру. Главные проблемы у информационного направления — ввести в свои модели внутреннюю активность и суметь представить индуктивные процедуры. Методы обработки данных OLAP(традиционная) и IAD(интеллектуальный анализ данных).
- Сторонники второго **«биологического»** направления считают, что как и в других областях моделирования необходимо опираться на особенности построения и функционирования реально существующих биологических систем, способных к высшей нервной деятельности. Феномены человеческого поведения, его способность к обучению и адаптации, по мнению этих специалистов, есть следствие именно биологической структуры и особенностей ее функционирования. В рамках этого направления возникли и активно развиваются такие области ИИ как бионика (наука о моделировании биологических существ, которая старается применить особенности строения и поведения животных в технике), моделирование нейронных сетей, адаптивного поведения живых существ, генетического механизма биологического приспособления и развития, процессов распознавания и т.д. В нейрофизиологии установлено, что целый ряд функций и свойств у живых организмов реализованы с помощью определенных нейронных структур. На основе воспроизведения таких структур в ряде случаев получены различные модели, которые активно используются в различных разработках систем ИИ. Проводятся исследования (работы Н.М. Амосова), связанные с построением действующих "интеллектуальных автоматов", в которые закладывается целый ряд представлений о человеческом разуме. Получены как программные, так и интересные физически реализованные автоматы. В модели введена активность элементов и критерии состояния, интерпретируются чувства, эмоции. В целом развитие области техники, связанной с созданием различных автоматов и управляемых систем во многом опирается на результаты исследований биологического направления в ИИ.

Классификация задач в области искусственного интеллекта

В рамках указанных выше направлений развития систем ИИ постепенно определились ряд перспективных направлений – задач:

В рамках информационного направления:

- Новые методы анализа данных – интеллектуальный анализ данных (ИАД), кластерный анализ и др.
- Нечеткая логика (включающая нечеткую логику, алгебру и теорию множеств).
- Представление знаний.
- Системы логического вывода, автоматического доказательства и т.д.
- Семантического анализа конструкций естественного языка, компьютерного перевода. Компьютерный естественно-языковой интерфейс.
- Интеллектуальные информационные системы и экспертные системы.
- Экспертные системы моделируют поведение человека-экспетра в какой-либо области знаний. Это особый вариант интеллектуальной информационной системы.
- В рамках биологического направления:
- Бионика.
- Генетические алгоритмы.
- Разработка автоматов и активных агентных систем.
- Проблемы адаптации, приспособления и активных эвристических методов получения знаний.

В настоящее время основные достижения в области создания компьютерных систем управления, которые способны самостоятельно адаптироваться в изменяющейся ситуации определяются использованием моделей поведения животных или человека. Особенно это актуально для обучения компьютерных программ активной деятельности, когда цель и методы достижения ставятся самой программой исходя из некой заложенной «сверх цели». При этом системе часто приходится самой добывать необходимые знания с помощью эвристических (экспериментальных) методов. Эвристика - получение знаний либо опытным путем, либо путем анализа. Фактически это задачи машинного обучения, т.е. получить знания на основе поступающих данных.

- Моделирование нейронных сетей.
- Задача распознав ания образов.

ЗРО делится на задачу кластеризации и классификации. Кластеризация — выделение образа из окружающей среды. Классификация — идентификация образа, определение к какому классу он относится.

Понятие о методах представления знаний

Всякая предметная область деятельности может быть описана в виде некоторой совокупности сведений о структуре этой области, основных ее характеристиках, процессах, протекающих в ней, а также о способах решения, возникающих в ней задач. Все эти сведения образуют знания о предметной области.

Знания традиционно делят на:

- Процедурные
- Декларативные.

Исторически первичными были процедурные знания, то есть знания, «растворенные» в алгоритмах. Они управляли данными. Для их изменения требовалось изменять программы. Однако с развитием искусственного интеллекта приоритет данных постепенно изменялся, и все большая часть знаний сосредоточивалась в структурах данных (таблицы, списки, абстрактные типы данных), то есть увеличивалась роль декларативных знаний. В интеллектуальных системах для хранения и использования знаний создаются специальные представления знаний, включающие в свой состав всю совокупность процедур, необходимых для записи знаний, извлечения их из памяти и поддержки хранилища знаний в рабочем состоянии. Системы представления знаний часто оформляются как базы знаний, являющиеся естественным развитием баз данных. Теория баз знаний составляет сейчас заметную часть искусственного интеллекта. Для представления знаний в памяти системы разрабатываются разнообразные модели представления знаний. В настоящее время в интеллектуальных системах используются четыре основных модели знаний:

- **1 - семантические сети**
- **2 – фреймы**
- **3 - логические системы**
- **4 — продукции**

Лекция №9

Раздел №3. Основы теории алгоритмизации

Тема 3.1 Основы теории алгоритмизации задач

Содержание: Понятие алгоритма и исполнителя алгоритма. Принцип потенциальной осуществимости. Запись алгоритмов. Основные свойства алгоритмов. Классификация алгоритмов. Способы представления алгоритмов.

19.Понятие алгоритма. Принцип потенциальной осуществимости. Основные свойства алгоритмов. Понятие исполнителя алгоритмов.

1. Понятие алгоритма и исполнителя алгоритма. Основные свойства алгоритмов.

Понятие алгоритма

Алгоритм относится к основным понятиям математики, а потому не имеет точного определения. Часто это понятие формулируют так: *«точное предписание о порядке выполнения действий из заданного фиксированного множества для решения всех задач заданного класса»*.

Рассмотрим подробнее ключевые слова в этой формулировке:

− ***«точное предписание»*** означает, что предписание однозначно и одинаково понимается всеми исполнителями алгоритма и при одних и тех же исходных данных любой исполнитель получает один и тот же результат;

− ***«из заданного фиксированного множества»*** означает, что множество действий, используемых в предписании, оговорено заранее и не может меняться в ходе исполнения алгоритма;

− ***«решения всех задач заданного класса»*** означает, что это предписание предназначено для решения класса задач, а не для одной отдельной задачи.

Алгоритм всегда определяет однозначно, какое действие должно быть выполнено следующим, равно как и то, какое действие должно быть выполнено следующим.

Применительно к ЭВМ алгоритм определяет вычислительный процесс, начинающийся с обработки некоторой совокупности возможных исходных данных и направленный на получение определённых этими исходными данными результатов.

Для задания алгоритма необходимо описать следующие его элементы:

- набор объектов, составляющих совокупность возможных исходных данных, промежуточных и конечных результатов;
- правило начала;
- правило непосредственной переработки информации (описание последовательности действий);
- правило окончания;
- правило извлечения результатов.

Понятие исполнителя алгоритма

Алгоритм всегда рассчитан на конкретного исполнителя. **Исполнитель** в информатике – человек или автоматическое устройство, которому поручается исполнить алгоритм или программу. Исполнителем может быть человек, группа людей, робот, станок, компьютер, язык программирования и т.д. Важнейшим свойством, характеризующим любого из этих исполнителей, является то, что исполнитель умеет выполнять некоторые команды. Так, исполнитель-человек умеет выполнять такие команды, как «встать», «сесть», «включить компьютер» и т.д., а исполнитель-язык программирования Бейсик – команды PRINT, END, LIST и другие аналогичные. Вся совокупность команд, которые данный исполнитель умеет выполнять, называется **системой команд исполнителя** (СКИ). Область, в пределах которой действует исполнитель, называется **средой исполнителя**. Одно из принципиальных обстоятельств состоит

в том, что исполнитель не вникает в смысл того, что делает, но получает необходимый результат. В таком случае говорят, что исполнитель действует **формально**, то есть отвлекается от поставленной задачи и только строго выполняет некоторые требования, инструкции.

Это важная особенность алгоритмов. Наличие алгоритма формализует процесс решение задачи, искажает рассуждения исполнителя. Использование алгоритма даёт возможность решать задачу формально, механически исполняя команды алгоритма в указанной последовательности. Целесообразность предусматриваемых алгоритмом действий обеспечивается точным анализом со стороны того, кто составляет этот алгоритм.

Введение в рассмотрение понятия «исполнитель» позволяет определить **алгоритм** как понятное и точное предписание исполнителю совершить последовательность действий, направленных на достижение поставленной цели. Наиболее распространенными и привычными являются алгоритмы работы с величинами – числовыми, символьными, логическими и т.д.

Свойства алгоритма

Свойства алгоритма принято делить на основные (необходимые по определению, без выполнения которых алгоритм не является алгоритмом) и сравнительные (с помощью которых можно сравнивать алгоритмы между собой).

Основные свойства:

- Дискретность. Алгоритм должен делить процесс решения задачи на последовательное выполнение простых (или ранее определенных) шагов (этапов).
- Детерминированность (определенность). Каждый шаг алгоритма должен быть четким, однозначным и не оставлять места для произвольного выбора.
- Результативность (конечность). За конечное число шагов алгоритм либо должен приводить к решению задачи, либо после конечного числа шагов останавливаться из-за невозможности получить решение с выдачей соответствующего сообщения.

Сравнительные свойства:

- Массовость – умение решать определенный класс задач.

Рекурсивность – использование процедур, функций, циклов и других повторяющихся конструкций

2. Принцип потенциальной осуществимости

Алгоритм должен давать результат за конечное число шагов. Таким образом, бесконечный алгоритм – это некая теоретическая абстракция, реальный алгоритм должен всегда иметь условие прерывания бесконечного повторения в таком алгоритме. Однако даже конечный алгоритм может быть настолько «сложным» (т.е. выполняться долго), что о его осуществимости можно говорить только потенциально. Кроме того, термин потенциальной осуществимостей может использоваться при условии наличия алгоритма, который невозможно или очень сложно исполнить в реальности. Таким образом, многие алгоритмы могут рассматриваться только как воображаемый процесс. Их осуществимость определяется как потенциальная в отличие от реальных алгоритмов, которые могут быть исполнены практически

20.Классификация алгоритмов. Блок-схемы описания алгоритмов. Формы записи алгоритмов.

3. Классификация алгоритмов.

Так как алгоритмов очень много существует множество вариантов их классификации. На практике наиболее употребительна классификация по типу решаемых задач.

- Вычислительный алгоритм (обработка некоторой совокупности возможных исходных данных и получение результата).
- Логический алгоритм (проверка условия).
- Моделирующий алгоритм (алгоритм создания и заданного функционирования математической модели). Данный тип алгоритмов используется для описания поведения модели объекта.
- Адаптивный алгоритм (обладает способностью настраиваться на решаемую задачу).
- Вероятностный алгоритм (использует случайные данные, результат его так же в каком-то смысле случайный).

Алгоритм формирования и функционирования объекта, объектно-ориентированное программирование. Описывает объект какого-то класса. От моделирующего отличается тем, что объект реальный (в моделирующем алгоритме) и объект класса различны по своей природе. Реальный объект – это объект, существующий в природе, для которого создаётся математическая модель и затем данная модель и её функционирование реализуются с помощью алгоритма. Объект класса – это просто структура данных.

4. Запись алгоритмов. Способы представления алгоритмов

Для записи алгоритмов используются следующие варианты:

- Текстовое представление в виде пронумерованной последовательности инструкций
- Графический метод (различные виды блок-схем)
- Использование языков программирования
- Использование языка псевдокода
- Использование программ специальных алгоритмических машин (Тьюринга, Поста) или алгорифмов Маркова

Язык Блок-схем содержит специальные геометрические конструкции – овал для начала, конца, возврата алгоритма, прямоугольник для обычных вычислений, ромб для условий, шестиугольник для циклов, круг для соединителя.

Способы представления алгоритмов

Любой возможный алгоритм может быть представлен в виде совокупности трёх основных конструкций: линейной, разветвлённой, циклической.

Линейные алгоритмы

Линейным называется алгоритм, в котором все этапы решения задачи выполняются последовательно. Каждая операция является самостоятельной, независимой от каких- либо условий. На схеме блоки, отображающие эти операции, располагаются в линейной последовательности.

Данный алгоритм имеет следующий формат:

Начало <последовательность выполняемых команд>; Конец.

Разветвленные алгоритмы

Ветвящимся называется такой алгоритм, в котором выбирается один из нескольких возможных путей (вариантов) вычислительного процесса. Каждый подобный путь называется **ветвью алгоритма**. Признаком ветвящегося алгоритма является наличие операций проверки условия. Обычно различают два вида условий – простые и составные [2].

Простым условием (отношением) называется выражение, составленное из двух арифметических выражений или двух текстовых величин, связанных одним из знаков: >, <, $\leq$, $\geq$, =, $\neq$.

Из отношений можно образовать **сложные (составные)** выражения, с помощью логических операций И, ИЛИ, НЕ.

Например, x<7 и x>2; y>0 или (y<0 и x<0).

В схеме алгоритма операцию проверки выполняет **логический блок**. Он изображается ромбом, внутри которого указывается проверяемое условие (отношение), и имеет два выхода: Да и Нет.

Если условие (отношение) истинно (выполняется), то выходим из блока по выходу «Да»; если ложно (не выполняется) – по выходу «Нет».

Ветвящийся алгоритм, включающий в себя две ветви, называется простым, более двух ветвей – сложным. Сложный ветвящийся алгоритм можно представить с помощью простых ветвящихся алгоритмов.

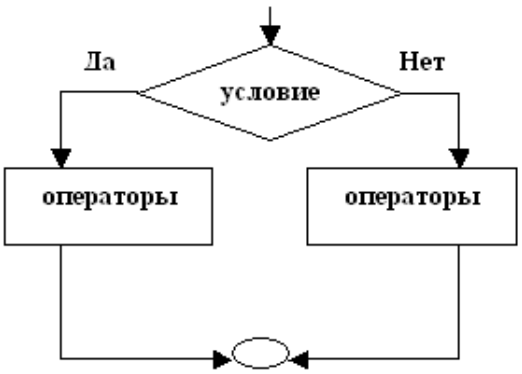
Ветвление реализуется в виде команды:

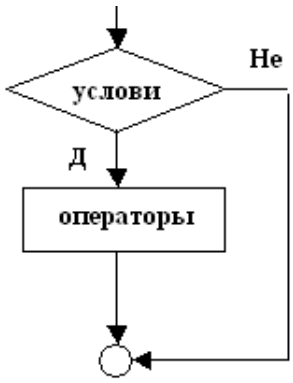
если <ЛВ> то <Серия 1> иначе <Серия 2>.

Здесь **<ЛВ>** – это логическое выражение;

<Серия 1> – описание последовательности действий, которые должны выполниться, когда **<ЛВ>** принимает значение «истина»;

<Серия 2> – описание последовательности действий, которые должны выполниться, когда **<ЛВ>** принимает значение «ложь».





Любая из этих серий может быть пустой. В этом случае ветвление называется неполным. В противном случае ветвление называется полным.

Циклические алгоритмы

Циклом называется последовательность действий, выполняемых многократно, каждый раз при новых значениях параметров.

Алгоритмы, включающие в себя циклы, называются **циклическими**.

Различают циклические алгоритмы с параметром, с предусловием и с постусловием.

1. **Цикл с параметром** (со счётчиком) применяется в тех случаях, когда в программе какие-то действия (операторы) повторяются и при этом некоторая величина меняется с постоянным шагом.

Формат команды на алгоритмическом языке имеет следующий вид:

НЦ

Для <параметр цикла> от <начальное значение> до <конечное значение>

Выполнить <оператор>

КЦ

Схема, иллюстрирующая работу оператора цикла с параметром, выглядит следующим образом:

2. Оператор **цикла с предусловием** используется в тех случаях, когда число повторений действий в программе неизвестно. Его общий вид:

НЦ

Пока <условие>

<тело оператора цикла>

КЦ

Где <условие> – условие, при котором выполняется <тело оператора цикла>.

Схема, иллюстрирующая работу данного, цикла имеет вид:

3. Оператор **цикла с постусловием** применяется тогда, когда число повторений действий заранее не известно. Данный оператор отличается от предыдущего тем, что тело цикла повторяется не менее одного раза.

Общий вид этого оператора:

НЦ

<тело оператора цикла>

КЦ при <условие>,

где <условие> – условие, при котором оператор прекращает свою работу.

Схема, иллюстрирующая работу оператора с постусловием:

Цикл называется **детерминированным**, если число повторений тела цикла заранее известно или оговорено. Примером может служить цикл с использованием оператора FOR.

Цикл называется **итерационным,** если число повторений тела цикла заранее не известно, а зависит от значений параметров (некоторых переменных), участвующих в вычислениях.

Например, цикл с оператором WHILE DO или REPEAT UNTIL.

Лекция №10

Раздел №3. Основы теории алгоритмизации

Тема 3.1 Основы теории алгоритмизации задач

Содержание: Рекурсия и итерация. Понятие о типах данных. Принципы программирования. Сложность алгоритма, оценка сложности алгоритма. Понятие о полиномиальных и реально выполнимых алгоритмах. Примеры полиномиальных алгоритмов. Класс NP – алгоритмов.

21. Рекурсия и итерация в алгоритмах. Понятие о типах данных

1. Рекурсия и итерация

Алгоритм, в состав которого входит итерационный цикл, называется **итерационным** алгоритмом.

Такие алгоритмы используются при реализации итерационных численных методов.

В итерационных алгоритмах необходимо обеспечить обязательное достижение условия выхода из цикла – **сходимость итерационного процесса**. В противном случае произойдет зацикливание алгоритма, т.е. не выполнится свойство результативности.

При составлении новых алгоритмов могут использоваться алгоритмы, составленные ранее. Алгоритмы, целиком используемые в составе других алгоритмов, называют вспомогательными алгоритмами. Алгоритм может содержать обращение к себе самому как вспомогательному, и в этом случае его называют **рекурсивным**. Рекурсия является средством программирования, при котором процедура или функция прямо или косвенно вызывает сама себя. Если команда обращения алгоритма к самому себе находится в самом алгоритме, то такую рекурсию называют **прямой**. Возможны случаи, когда рекурсивный вызов данного алгоритма происходит из вспомогательного алгоритма, к которому в данном алгоритме имеется обращение. Такая рекурсия называется **косвенной**. Косвенный вызов может быть организован и более сложным образом, то есть в рекурсию могут быть вовлечены несколько подпрограмм.

Рекурсия позволяет, например, очень просто запрограммировать вычисление рекуррентных соотношений. Рассмотрим следующий способ вычисления факториала: $f_0 = 1$, $f_n = f_{n-1} \cdot n$.

При анализе рекурсивной программы обычно возникают два вопроса:

- почему программа заканчивает работу?
- почему она работает правильно?

Чтобы доказать пункт а), обычно проверяют, что с каждым рекурсивным вызовом значение какого-то параметра изменяется (чаще всего уменьшается), и это не может продолжаться бесконечно. Для ответа на вопрос b) достаточно проверить, что содержащая рекурсивный вызов программа работает правильно, предположив, что вызываемая ею одноимённая подпрограмма работает правильно. В самом деле, в этом случае в цепочке рекурсивно вызываемых программ все программы работают правильно (в этом можно убедиться, двигаясь от конца цепочки к началу).

2. Понятие о типах данных

До разработки алгоритма (программы) необходимо выбрать оптимальную для реализации задачи структуру *данных*. Неудачный выбор *данных* и их описания может не только усложнить решаемую задачу и сделать ее плохо понимаемой, но и привести к неверным результатам. На структуру *данных* влияет и выбранный метод решения.

Тип *данных* характеризует область определения *значений данных*. Задаются типы *данных* простым перечислением *значений* типа, например как в *простых типах данных* , либо объединением (структурированием) ранее определенных каких-то типов – *структурированные типы данных*. Для обозначения текущих *значений данных* используются константы – числовые, текстовые, логические. Часто (в зависимости от задачи) рассматривают *данные*, которые имеют не только "линейную" (как приведенные выше), но и иерархическую структуру. В алгоритмических языках есть **стандартные типы**, например, целые, вещественные, символьные, тестовые и логические типы. Они в этих языках не уточняются (не определяются, описываются явно) и имеют соответствующие описания с помощью служебных слов. Каждый тип *данных* допускает использование определенных операций со *значениями* типа ("с типом").

Для описания величин, значение которых может изменяться вводятся переменные. К структурным типам данных можно отнести – массивы, множества, строки, списки, деревья, таблицы, файлы. Наиболее часто используемая структура данных – *массив*. Одномерный *массив* (вектор, *ряд*, линейная таблица) – это совокупность *значений* некоторого простого типа (целого, вещественного, символьного, текстового или логического типа), перенумерованных в каком-то порядке и имеющих общее имя. Для выделения конкретного элемента *массива* необходимо указать его порядковый номер в этом *ряду*. *Значение* порядкового номера элемента *массива* называется индексом элемента. Двумерный *массив* (*матрица*, прямоугольная таблица) – совокупность одномерных векторов, рассматриваемых либо "горизонтально" (векторов-строк), либо "вертикально" (векторов-столбцов) и имеющих одинаковую размерность, одинаковый тип и общее имя. *Матрицы*, как и векторы, должны быть в алгоритме описаны служебным словом (например, таб или array), но в отличие от вектора, *матрица* имеет описание двух индексов, разделяемых запятыми: первый определяет начальное и конечное *значение* номеров строк, а второй – столбцов. Для актуализации элемента двумерного *массива* нужны два его индекса – номер строки и номер столбца, на пересечении которых стоит этот элемент.

Работа с файлами

Файл — это именованная структура данных, представляющая собой последовательность элементов данных одного типа, причем количество элементов последовательности практически не ограничено. В первом приближении файл можно рассматривать как массив переменной длины неограниченного размера.

Файлы могут хранить в себе все, что поддается кодированию:

- исходные тексты программ или входные данные (тесты);
- машинные коды выполняемых программ (игры, вирусы, обучающие и сервисные программы, др.);
- информацию о текущем состоянии какого-либо процесса;
- различные документы, в том числе и Интернет-страницы;
- картинки (рисунки, фотографии, видео);
- музыку;
- и т.д. и т.п.

Для ОС подразделяется работа с тремя видами файлов:

- *текстовыми*;
- *типизированными;*
- *нетипизированными*.

Последние два типа объединяются под названием ***бинарные***: информация в них записывается по байтам и потому не доступна для просмотра или редактирования в удобных для человека текстовых редакторах, зато такие файлы более компактны, чем *текстовые*.

Файл, компонентами которого являются данные символьного типа, называется символьным, или текстовым. В отличие от *бинарных*, *текстовые файлы* возможно создавать, просматривать и редактировать "вручную" - в любом доступном текстовом редакторе. Кроме того, при считывании данных из *текстового файла* нет необходимости заботиться об их преобразовании.

Имя файла задается согласно принятым в OS правилам. Оно может быть полным, т. е. состоять не только непосредственно из имени файла, но и включать путь к файлу (имя диска, каталогов и подкаталогов). Перед выводом в файл его необходимо открыть. Если программа, формирующая выходной файл, уже использовалась, то возможно, что файл с результатами работы программы уже есть на диске. Поэтому программист должен решить, как поступить со

старым файлом: заменить старые данные новыми или новые данные добавить к старым. Способ использования старого варианта определяется во время открытия файла. Возможны следующие режимы открытия файла для записи в него данных:- перезапись (запись нового файла поверх существующего или создание нового файла);- добавление в существующий файл.

Аналогично построена работа с типизированными и не типизированными файлами. Главное отличие их – они файлы прямого доступа (текстовые файлы имеют последовательный доступ, строка за строкой). Такие файлы состоят из блоков равного обьема. В типизированном файле они так же пронумерованы. Это записи, которые имеют номер. Запись как положено в Паскале может состоять из разнотипных полей с собственными именами и доступом как у полей класса через указание точки. В не типизированном файле блоки просто равной длины в байтах (эту длину можно менять).

22.Принципы программирования. Методы разработки и анализа алгоритмов

3. Принципы программирования

Разработка программы состоит из двух различных действий – создания алгоритма и представления его в виде программы. При этом, создание алгоритма – это, как правило, наиболее сложный этап, т.к. создать алгоритм – значит, найти метод решения задачи. Поэтому, чтобы понять, как создать алгоритм, необходимо понять процесс решения задачи.

Рассмотрение методов решения задач и их подробное изучение не являются аспектами только информатики. Это важно практически для всех областей науки. Тесная связь между процессом создания алгоритмов и общей проблемой поиска решения задачи привела к сотрудничеству специалистов разных областей науки. Но оказалось невозможным свести проблему решения задачи только к созданию алгоритма. Таким образом, способность решать задачи является профессиональным навыком, а не точной наукой. Следовательно, его нужно развивать, научиться этому невозможно.

В 1945 году математик Полиа предложил фазы решения задачи, которые остаются теми основными принципами, на которых основываются и сегодня при обучении навыкам решения задач.

Фаза 1. Понять сущность задачи.

Фаза 2. Разработать план решения задачи.

Фаза 3. Выполнить план.

Фаза 4. Оценить точность решения, а также его потенциал в качестве средства для решения других задач.

Применительно к программированию эти фазы могут выглядеть так:

Фаза 1. Понять сущность задачи.

Фаза 2. Предложить идею, какая алгоритмическая процедура позволит решить задачу.

Фаза 3. Сформулировать алгоритм и представить его в виде программы.

Фаза 4. Оценить точность программы и ее потенциал в качестве средства для решения других задач.

Важно отметить, что эти фазы не являются этапами, это фазы, которые должны быть когда-либо выполнены в процессе решения задачи.

Подходы к созданию алгоритмов и требования к ним изменялись в ходе эволюции компьютеров.

Изначально программы составлялись из команд, непосредственно исполнявшихся компьютерами:

- операция присваивания;
- простейшие арифметические операции;
- операции сравнения чисел;
- операторы безусловного и условного перехода;
- операторы вызова подпрограммы.

Такой подход, ориентированный на непосредственно выполняемые компьютером операции, называется **операциональным**.

Отметим ***основные недостатки*** алгоритмов, к которым приводил операциональный подход:

- злоупотребление командой условного и безусловного переходов зачастую приводило к очень запутанной структуре программы, напоминавшей по образному сравнению «блюдо спагетти»;
- вместе с разнообразными уловками, направленными на повышение эффективности программы (т.е. минимальных требований к оперативной памяти и минимального времени выполнения), это приводило к непонятности программ, их ненадежности, трудностям в отладке и модификации, делая программирование трудоемким, сложным и чрезвычайно дорогостоящим.

Необходимость ориентироваться на ограниченный набор команд компьютера, на его скромные возможности приводила к огромной трудоемкости, к сложности программ, к проблемам, связанным с ошибками в них. В результате узким местом в развитии вычислительной техники оказалось именно программирование.

Структурное программирование

Появившийся в начале 1970-х годов новый подход к разработке алгоритмов получил название структурного. С появлением структурного программирования описанные выше трудности были во многом преодолены. В основе технологических принципов структурного программирования лежит утверждение о том, что логическая структура программы может быть выражена комбинацией трех базовых структур: следования, ветвления и цикла (это содержание теоремы Бема-Якопини).

Следование - самая важная из структур. Она означает, что действия могут быть выполнены друг за другом

Рис. 1. Структура «следование»

Эти прямоугольники могут представлять как одну единственную команду, так и множество операторов, необходимых для выполнения сложной обработки данных.

Ветвление - это структура, обеспечивающая выбор между двумя альтернативами. Выполняется проверка, а затем выбирается один из путей (рис. 2).

Эта структура называется также «ЕСЛИ - ТО - ИНАЧЕ», или «развилка». Каждый из путей (ТО или ИНАЧЕ) ведет к общей точке слияния, так что выполнение программы продолжается независимо от того, какой путь был выбран.

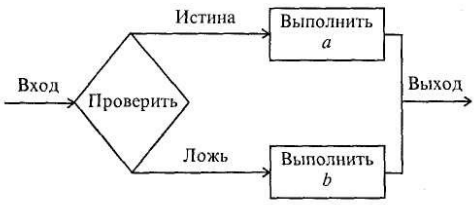


рис. 2.

Цикл (или повторение) предусматривает повторное выполнение некоторого набора команд программы. Если бы циклы не существовали, вряд ли занятие программированием было бы оправданным: циклы позволяют записать длинные последовательности операций обработки данных с помощью небольшого числа повторяющихся команд.

Эти структуры можно комбинировать одну с другой - как путем организации их следований, так и путем создания суперпозиций (вложений одной структуры в тую) - сколь угодно разнообразно для выражения логики алгоритма решения любой задачи. Используя описанные структуры, можно полностью исключить использование каких-либо еще операторов условного и безусловного перехода, что является важным признаком структурного программирования.

Умение образовывать из базовых структур их суперпозиции в соответствии с условиями конкретной задачи - одно из важнейших в программировании.

Еще одним важным компонентом структурного подхода к разработке алгоритмов является **модульность.** Модуль - это последовательность логически связанных операций, оформленных как отдельная часть программы. Использование модулей имеет следующие преимущества:

- возможность создания программы несколькими программистами;
- простота проектирования и последующих модификаций программы;
- упрощение отладки программы - поиска и устранения в ней ошибок;
- возможность использования готовых библиотек наиболее употребительных модулей.

Но, пожалуй, самым важным достижением структурного подхода к разработке алгоритмов является **нисходящее проектирование программ**, основанное на идее уровней абстракции, которые становятся уровнями модулей в разрабатываемой программе. На этапе проектирования строится схема иерархии, изображающая эти уровни. Схема иерархии позволяет программисту сначала сконцентрировать внимание на определении того, что надо сделать в программе, а лишь затем решать, как это зада делать. При нисходящем проектировании исходная, подлежащая решению задача разбивается на ряд подзадач, подчиненных по своему содержанию главной задаче. Такое разбиение называется детализацией или декомпозицией. На следующем этапе эти задачи, в свою очередь, разбиваются на более мелкие подчиненные подзадачи и так далее, до уровня относительно небольших подзадач, которые требуют для решения небольших модулей в 3 - 5 строк. Такой метод проектирования программ позволяет преодолевать проблему сложности разработки программы (и ее последующей отладки и сопровождения).

Объектно-ориентированное и декларативное программирование

Сегодня базовым методом программирования стало объектно-ориентированное программирование, а противостоящим ему при решении определенных классов задач является декларативное программирование, выраженное двумя разными подходами - функциональным и логическим.

Ограничимся кратчайшей характеристикой этих направлений.

Объект - основное понятие объектного программирования – объединяющее в себе информационные структуры (поля) и действия над ними (методы) Объекты могут перенимать свойства друг у друга с помощью наследования и реагировать на события (событийное управление). В объектно-ориентированном подходе исходная задача представляется как совокупность взаимодействующих объектов.

Декларативный подход в разработке компьютерных программ появился в начале 70-х годов. Он не получил столь широкого применения как процедурный, поскольку был направлен на относительно узкий круг задач искусственного интеллекта. При его применении программист описывает свойства исходных данных, их взаимосвязи, свойства, которыми должен обладать результат, а не алгоритм получения результата. Алгоритм решения порождается автоматически той системой, которая поддерживает декларативно-ориентированный язык программирования (ядро). При ***логическом*** варианте такого подхода (Пролог) задача описывается совокупностью фактов и правил в некотором формальном логическом языке, при ***функциональном*** варианте - в виде функциональных соотношений между фактами (язык Лисп).

Методы разработки и анализа алгоритмов

Нисходящим проектированием алгоритмов, *проектированием* алгоритмов "сверху вниз" или методом последовательной (пошаговой) *нисходящей разработки* алгоритмов называется такой метод составления алгоритмов, когда исходная задача (алгоритм) разбивается на ряд вспомогательных подзадач (подалгоритмов), формулируемых и решаемых в терминах более простых и элементарных операций (процедур). Последние, в свою очередь, вновь разбиваются на более простые и элементарные, и так до тех пор, пока не дойдём до команд исполнителя. В терминах этих команд можно представить и выполнить полученные на последнем шаге разбиений подалгоритмы (команд системы команд исполнителя).

Восходящий метод , наоборот, опираясь на некоторый, заранее определяемый корректный набор подалгоритмов, строит функционально завершенные подзадачи более общего назначения, от них переходит к более общим, и так далее, до тех пор, пока не дойдем до уровня, на котором можно записать решение поставленной задачи. Этот метод известен как метод *проектирования* "снизу вверх".

Структурные принципы алгоритмизации (***структурные методы*** алгоритмизации) – это принципы формирования алгоритмов из базовых структурных алгоритмических единиц (следование, ветвление, повторение), используя их последовательное соединение или вложение

друг в друга с соблюдением определённых правил, гарантирующих читабельность и исполняемость алгоритма сверху вниз и последовательно.

Структурированный алгоритм – это алгоритм, представленный как следования и вложения базовых алгоритмических структур. У структурированного алгоритма статическое состояние (до актуализации алгоритма) и динамическое состояние (после актуализации) имеют одинаковую логическую структуру, которая прослеживается сверху вниз ("как читается, так и исполняется"). При структурированной *разработке* алгоритмов правильность алгоритма можно проследить на каждом этапе его построения и выполнения.

Теорема (о структурировании) Бема-Якопини. Любой алгоритм может быть эквивалентно представлен структурированным алгоритмом, состоящим из базовых алгоритмических структур (линейная, ветвление, цикл).

Одним из широко используемых методов *проектирования* и *разработки* алгоритмов (программ) является *модульный метод* (модульная технология).

Модуль – это некоторый алгоритм или некоторый его блок, имеющий конкретное наименование, по которому его можно выделить и актуализировать. Иногда модуль называется **вспомогательным алгоритмом**, хотя все алгоритмы носят вспомогательный характер. Это название имеет смысл, когда рассматривается динамическое состояние алгоритма; в этом случае можно назвать вспомогательным любой алгоритм, используемый данным в качестве блока (составной части) тела этого динамического алгоритма. Используют и другое название модуля – **подалгоритм**. В программировании используются синонимы – процедура, подпрограмма.

Свойства модулей:

- *функциональная целостность и завершенность* (каждый модуль реализует одну функцию, но реализует полностью);
- *автономность и независимость от других модулей* (независимость работы модуля-преемника от работы модуля-предшественника; при этом их связь осуществляется только на уровне передачи/приема параметров и управления);
- *эволюционируемость* (развиваемость);
- *открытость* для пользователей и разработчиков (для модернизации и использования);

Свойства (преимущества) модульного *проектирования* алгоритмов:

- возможность *разработки* алгоритма группой исполнителей;
- *возможность создания и ведения библиотеки* наиболее часто используемых алгоритмов (подалгоритмов);
- облегчение *тестирования* алгоритмов и обоснования их правильности ;
- упрощение *проектирования* и модификации алгоритмов ;
- уменьшение сложности *разработки* (*проектирования*) алгоритмов (или комплексов алгоритмов);
- *наблюдаемость вычислительного процесса* при реализации алгоритмов.

Тестирование алгоритма – это проверка правильности или неправильности работы алгоритма на специально заданных *тестах* или тестовых примерах – задачах с известными входными данными и результатами (иногда достаточны их приближения). Тестовый набор должен быть минимальным и полным, то есть обеспечивающим проверку каждого отдельного типа наборов входных данных, особенно исключительных случаев.

Трассировка – это метод пошаговой фиксации динамического состояния алгоритма на некотором *тесте*. *Трассировка* облегчает отладку и понимание алгоритма.

Процесс поиска и исправления (явных или неявных) ошибок в алгоритме называется **отладкой алгоритма**.

Некоторые (скрытые, трудно обнаруживаемые) ошибки в сложных программных комплексах могут выявиться только в процессе их эксплуатации, на последнем этапе поиска и исправления ошибок – этапе сопровождения. На этом этапе также уточняют и улучшают документацию, обучают персонал использованию алгоритма (программы).

По одной из классификаций методы программирования делятся на следующие типы:

- Операциональные
- Процедурные, называемые также директивные (directive) или императивными (imperative),
- Декларативные (declarative) языки,
- Объектно-ориентированные (object-oriented).

Операциональное программирование обычно рассматривается на примере машинно-ориентированного языка Ассемблер (хотя оно присутствовало и в первых языках высокого уровня). К процедурным языкам относятся такие классические языки программирования, как Algol, Fortran, Basic, Pascal, C. Наиболее существенными классами декларативных языков являются **функциональные** (functional), и реляционные - **логические** (logic) языки. К категории функциональных языков относятся, например, Lisp и Haskell. Самым известным языком логического программирования является Prolog (Пролог). Существуют реляционные языки, которые не являются логическими – например язык SQL. Среди объектно-ориентированных языков программирования (языков ООП) отметим C++, Java, Object Pascal, Python.

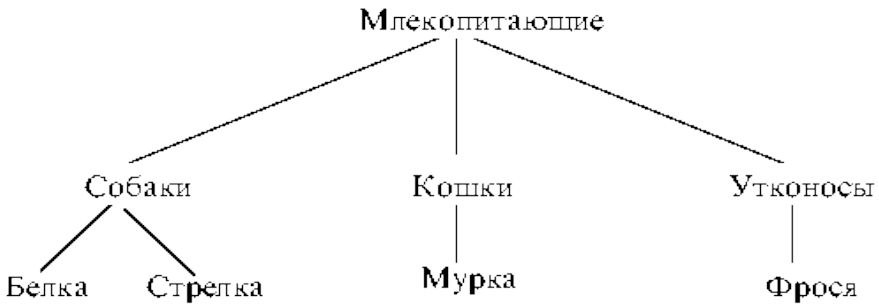
Особенность операционального программирования – ориентация на построение инструкции для исполнителя – процессора ЭВМ в терминах его операций. Главный недостаток – сложность программ, злоупотребление оператором перехода GOTO.

Особенность процедурного программирования – переход к более строгому, логически структурному программированию. Основные принципы процедурного программирования – использование стандартных технологий проектирования, структурного и модульного принципов, исключение GOTO, максимальное использование стандартных алгоритмов. Главный недостаток – постепенное загромождение модулей процедурами и функциями, недостаточная читабельность и понимаемость больших программ.

Особенность декларативного программирования – программист только описывает (создает декларацию) задачу, а методы и средства ее решения подбирает специальная программа – ядро языка программирования. Главный недостаток – недостаточное развитие среды и средств программирования для данных языков.

Особенность ООП – использование иерархии классов, соединение в объектах полей и методов работы с полями, полиморфизм используемых методов. Появление ООП стимулировало развитие визуальных сред программирования и переход к событийно-управляемым программам.

Объектно-ориентированное программирование (ООП) - это результат естественной эволюции более ранних методологий программирования. Потребность в ООП связана со стремительным усложнением разрабатываемых программ и, как следствие, их недостаточной надежностью. Модульное программирование, рассмотренное выше, оказалось не способным решить эту проблему. Можно сказать, что **ООП** - это моделирование объектов посредством иерархически связанных классов. При этом малозначащие детали объекта скрыты от нас, и если мы даем команду какому-то объекту, то он "знает", как ее выполнить. Фундаментальной концепцией в ООП является понятие обязанности или **ответственности** за выполнение действия. Все объекты являются представителями, или **экземплярами**, классов. Метод, активизируемый объектом в ответ на сообщение, определяется классом, к которому принадлежит получатель сообщения. Все объекты одного класса используют одни и те же методы в ответ на одинаковые сообщения. Классы представляются в виде иерархической древовидной структуры, в которой классы с более общими чертами располагаются в корне дерева, а специализированные классы и в конечном итоге индивидумы располагаются в ветвях. На рисунке показана одна из возможных иерархий классов, включающая в себя собак Белку и Стрелку, кошку Мурку и утконоса Фросю.



Классы собак, кошек и утконосов являются дочерними по отношению к классу млекопитающих, следовательно наследуют его свойства. При программной реализации этой иерархии логично метод "кормление детенышей" реализовывать в родительском классе, вместо того, чтобы несколько раз дублировать его в каждом из подклассов. **Наследование** свойств родительского класса позволяет использовать их в дочерних классах. Немного другая ситуация с рождением детенышей, ведь утконосы откладывают яйца, а не являются живородящими животными? В этой ситуации выручает свойство **полиформизма**: различные реализации методов могут носить одинаковые имена, а система сама определит какую из реализаций использовать в том или ином случае. В нашем примере следует в классе млекопитающих реализовать метод "потомство" (родить детеныша), в классах собак и кошек этот метод будет отсутствовать (система будет искать его в родительском классе и найдет его там), а в классе утконосов нужно написать новый метод, с тем же именем, но другой реализацией (отложить яйца).

Таким образом в основе ООП лежат три основных понятия:

- инкапсуляция (сокрытие данных в классе или методе);
- наследование;
- полиморфизм.

Инкапсуляцию можно представить, как защитную оболочку вокруг кода данных, с которыми этот код работает. Оболочка задает поведение и защищает код от произвольного доступа извне.
Наследование - это процесс, в результате которого один тип наследует свойства другого типа.
Полиморфизм - это концепция, позволяющая иметь различные реализации для одного и того же метода, которые будут выбираться в зависимости от типа объекта, переданного методу при вызове.

23.Сложность алгоритмов. Варианты оценки сложности. Асимптотическая сложность алгоритма.
24.Не полиномиальные алгоритмы. Примеры задач НП. Замкнутость класса задач НП. Понятие неразрешимой задачи. Экстраалгоритм.
26.Реально выполнимые алгоритмы. Совпадение классов полиномиальных и реально выполнимых алгоритмов. Примеры полиномиальных алгоритмов.

Сложность алгоритма, оценка сложности алгоритма. Понятие о полиномиальных и реально выполнимых алгоритмах.

Известно, что правильность — далеко не единственное качество, которым должна обладать хорошая программа. Одним из важнейших является эффективность. Эффективность – сравнительная характеристика алгоритма и определяется неким критерием эффективности. Это может быть время выполнения алгоритма, число операций процессора, размер памяти используемой программы и т.д.. Этот критерий определяется как зависимость или функция от различных входных данных (параметра N). В тоже время можно говорить о эффективности алгоритма и сложности задачи. Можно сравнивать разные алгоритмы для одной задачи, а можно для разных задач. Поэтому принято говорить о сложности или скорости алгоритма и о сложности задачи. Так как эти величины есть функции, то можно их сравнивать по асимптотическим оценкам при стремлении параметра $N \to \infty$. Нахождение точной зависимости критерия эффективности для конкретной программы — задача достаточно сложная. По этой причине обычно ограничиваются **асимптотическими оценками** этой функции, то есть описанием ее

примерного поведения при больших значениях параметра N. При этом для асимптотических оценок используют традиционное отношение O (читается "O большое") между двумя функциями f(n)=O(n). Асимптотические оценки функции при N->∞ можно представить выражением O(f(N)), которое означает const* f(N). Пусть заданы 2 оценки O(f(N)) и O(g(N)). Будем считать, что:

Функция f(N) растет медленнее g(N), если $\lim_{N\to\infty} [O(f(N)) / O(g(N))] = 0$.

Функция f(N) растет быстрее g(N), если $\lim_{N\to\infty} [O(g(N)) / O(f(N))] = 0$.

Функции f(N) и g(N) имеют одинаковую скорость роста, если $\lim_{N\to\infty} [O(f(N)) / O(g(N))] = const \neq 0$.

Аналогично по асимптотике роста критерия эффективности можно говорить о скорости роста алгоритма. Причем, чем больше скорость роста алгоритма, тем он фактически медленнее и наоборот. Соответственно по функции асимптотики можно различать линейные O(N), квадратичные $O(N^2)$, кубические $O(N^3)$ экспоненциальные $O(e^N)$ и др. алгоритмы.

Сложность задачи можно оценить по скорости наиболее эффективного для нее алгоритма. При этом, может оказаться, что наиболее эффективный алгоритм нам пока не известен. Поэтому реальная сложность задачи может оказаться ниже чем представляемая нами, но не может оказаться больше чем текущая.

В качестве примера рассмотрим алгоритм нахождения факториала числа. Легко видеть, что количество операций, которые должны быть выполнены для нахождения факториала N! числа N в первом приближении прямо пропорционально этому числу, ибо количество повторений цикла (итераций) при вычислении по формуле N!=(N-1)!*N программе равно N. В подобной ситуации принято говорить, что алгоритм имеет ***линейную сложность*** (сложность O(N)). Можно ли вычислить факториал быстрее? Оказывается, да. Можно написать такую программу, которая будет давать правильный результат для тех же значений N с логарифмической скоростью. Про алгоритмы, в которых количество операций примерно пропорционально log(N) (в информатике обычно используют для основания двоичный логарифм) говорят, что они имеет ***логарифмическую сложность*** (O(log(N)).

Полиномиальным алгоритмом называют алгоритм, скорость которого может быть представлена полиномом какой-либо конечной степени.

Реально-выполнимым алгоритмом называют алгоритм, у которого время решения задачи возможно для реального использования на практике. Так как время решения зависит не только от алгоритма, но и от задачи (то есть от N), то некоторые задачи могут быть решены при малых N, но не решаются для больших значений N.

Когда начинающие программисты тестируют свои программы, то значения параметров, от которых они зависят, обычно невелики. Поэтому даже если при написании программы был применен неэффективный алгоритм, это может остаться незамеченным. Однако, если подобную программу попытаться применить в реальных условиях, то ее практическая непригодность проявится незамедлительно.

С увеличением быстродействия компьютеров возрастают и значения параметров, для которых работа того или иного алгоритма завершается за приемлемое время. Таким образом, увеличивается среднее значение величины N, и, следовательно, возрастает величина отношения времен выполнения быстрого и медленного алгоритмов. Исходя из практики использования различных алгоритмов для решения задач было принято использовать понятие реально выполнимых алгоритмов только для задач полиномиальной сложности и более простых. Таким образом, класс полиномиальных (P) алгоритмов совпадает с классом реально выполнимых алгоритмов.

Полиномиальные и не полиномиальные алгоритмы. Класс NP – алгоритмов

Задачи, которые решаются различными полиномиальными алгоритмами, очень разнообразны, но для информатики наиболее популярны задачи поиска и сортировки. Простой поиск в массиве реализуется алгоритмом линейной скорости от числа элементов массива O(N). Возможен вариант быстрого бинарного поиска, если массив был заранее отсортирован. Тогда скорость алгоритма выше полиномиальной ($O(\log_2(N))$).

Простые алгоритмы сортировок

К ***простым внутренним сортировкам*** относят методы, сложность которых пропорциональна квадрату размерности входных данных. Иными словами, при сортировке массива, состоящего из N компонент, такие алгоритмы будут выполнять $C*N^2$ действий, где C - некоторая константа. Количество действий, необходимых для упорядочения некоторой последовательности данных, конечно же, зависит не только от длины этой последовательности, но и от ее структуры. Например, если на вход подается уже упорядоченная последовательность (о чем программа, понятно, не знает), то количество действий будет значительно меньше, чем в случае перемешанных входных данных. Как правило, сложность алгоритмов подсчитывают раздельно по количеству сравнений и по количеству перемещений данных в памяти (пересылок), поскольку выполнение этих операций занимает различное время. Однако точные значения удается найти редко, поэтому для оценки алгоритмов ограничиваются лишь понятием "пропорционально", которое не учитывает конкретные значения констант, входящих в итоговую формулу. Общую же эффективность алгоритма обычно оценивают "в среднем": как среднее арифметическое от сложности алгоритма "в лучшем случае" и "в худшем случае", то есть (Eff_best + Eff_worst)/2. Таким образом, этот алгоритм имеет сложность $O(N^2)$ ("порядка эн квадрат").

Сортировка информации имеет смысл, если информация структурирована. Тогда сортировка предполагает изменение структуры так, чтобы она располагалась по направлению возрастания\убывания некоторого числового критерия сортировки. Соответственно можно говорить о сортировке линейных структур (массивы, таблицы, линейные списки), сортировки деревьев, сортировки различных сетей и т.д.. В простейшем случае сортировки линейной структуры алгоритмы принято делить на простые и быстрые алгоритмы. Простые (медленные) алгоритмы имеют квадратичную по числу элементов скорость и называются квадратичными. К ним относят методы «пузырька» или обменной сортировки, выбора, вставки, дополнительного массива и т.д..

Метод выбора использует алгоритм поиска max\min. Например, для сортировки по возрастанию используем поиск минимального элемента. Найдем min элемент и поменяем его местами с первым, затем повторим это действие с частью массива без 1-го элемента и т.д.. Когда останется 1 элемент, прекратим вычисления – массив отсортирован.

Помимо класса полиномиальных алгоритмов (P – класса), могут существовать задачи не имеющие алгоритма решения полиномиальной скорости. Такие задачи принято называть NP задачами, а алгоритмы их решения NP- алгоритмами.

Доказана теорема о полноте класса NP задач:

Если существует полиномиальное решение хотя бы одной NP-задачи, то на его основе можно построить алгоритмы решения для любой NP – задачи. Таким образом, NP-задачи являются не полиномиальными только в совокупности. Это свидетельствует о полноте класса NP-задач.

Скорость роста NP-алгоритмов представляется либо как $O(e^N)$, $O(K^N)$ или O(N!) (в соответствии с формулой Стирлинга, их скорости близки и выше любой фиксированной степени).

Примерами NP-задач являются задачи:

- Разложение числа на простые сомножители. Если считать за N – число сомножителей, то скорость роста алгоритма O(N!).

- Задача коммивояжера. В данной задаче заданы N городов и расстояния между ними, необходимо построить замкнутый путь, который удовлетворял условиям – минимальности суммарного расстояния, проходил бы через все города и только по одному разу. Фактически это задача поиска минимального гамильтонова пути в графе с заданными длинами ребер. Скорость роста такого алгоритма выше полиномиальной.

Поскольку к классу реально выполнимых алгоритмов относятся только полиномиальные и более быстрые алгоритмы, то NP-алгоритмы выполнимы только потенциально. Это означает, что при увеличении числа элементов в задаче время выполнения такого алгоритма быстро выходит за пределы возможностей его реализации. Поэтому для NP- задач очень важно правильно оценить требуемое количество операций для ее реального решения.

Пример: Оценка времени решения задачи выбранным ЭВМ для разных алгоритмов

Сравнительная таблица времен выполнения алгоритмов

Сложность алгоритма	n=10	$n=10^3$	$n=10^6$
Log_2 (N)	0.2 сек.	0.6 сек.	1.2 сек.
N	0.6 сек.	1 мин.	16.6 час.
N^2	6 сек.	16.6 час.	1902 года
2^N	1 мин.	10^{295} лет	10^{300000} лет

Неразрешимые задачи и экстраалгоритм

Ранее уже рассматривались алгоритмы и их сложность. При этом был определен класс NP-алгоритмов, который имел максимальную сложность. Однако можно определить класс еще облее сложных алгоритмов, которые невозможны с точки зрения основных свойств алгоритмов. Если NP-алгоритмы можно рассматривать как потенциально осуществимые алгоритмы, то эти алгоритмы не осуществимы даже потенциально или требуют бесконечного числа операций. В теории алгоритмов они получили название экстраалгоритмов. Как правило экстраалгоритмы возникают при попытке решения задач очень большой общности. Например попытка построить алгоритм, который мог бы проверять возможность использования произвольного алгоритма для произвольного данного является экстраалгоритмом.

Принято задачи, которые решаются экстраалгоритмом называть неразрешимыми. В теории алгоритмов особой проблемой остается доказательство неразрешимости тех или иных задач. По крайней мере, такое доказательство позволило бы прекратить поиски метода ее решения.

Лекция №11

Раздел №3. Основы теории алгоритмизации

Тема 3.1 Основы теории алгоритмизации задач

Содержание: Методы построения эффективных алгоритмов: итерационные формулы, метод бинарных деревьев и их балансировки, рекурсивные алгоритмы, динамическое программирование. Основные методы эффективного представления данных – основные модели данных, динамические структуры данных.

25.Основные методы разработки эффективных алгоритмов: итерационные формулы, рекурсивные алгоритмы, метод балансировки дерева, динамическое программирование

Методы построения эффективных алгоритмов

итерационные формулы, метод бинарных деревьев и их балансировки, рекурсивные алгоритмы, динамическое программирование

Основные методы эффективного представления данных

Существует ряд методов, которые могут быть использованы для повышения эффективности разрабатываемых алгоритмов. Рассмотрим некоторые из таких методов:

- **Итерационные формулы.** Метод заключается в представлении алгоритма вычисления функции в виде итерационной формулы. Во многих случаях такое представление упрощает построение алгоритма.

Итерационные вычисления производятся по итерационным формулам, которые бывают двух видов:

- явная формула имеет в общем случае вид $x=f(x)$
- неявная формула всегда может быть представлена в виде $F(x)=0$.

Неявные формулы могут быть преобразованы к явному виду. Например линейное преобразование: $x=C*F(x)+x$, где С – некоторая константа, которая выбирается из условия сходимости получаемой итерационной последовательности. Явная формула далее может быть представлена в виде рекуррентной формулы, когда элемент итерационной последовательности x_k выражается в виде

$$x_k = f(x_{k-1}, x_{k-2} \ldots)$$

. Результатом вычисления по такой формуле является итерационная последовательность x_k. Если эта последовательность имеет предел, то он, как правило, является решением задачи итерационного вычисления. Другим вариантом задачи может быть вычисление суммы или произведения (конечных или бесконечных) элементов итерационной последовательности. Количество элементов x_{k-j} в правой части формулы $x_k = F(x_{k-1}, x_{k-2} \ldots x_{k-m})$ определяет m - число начальных значений элементов последовательности, которые потребуются для вычисления. Обычно m=1 и задача требует одно начальное условие, но иногда это не так:

Пример. Вычисление последовательности чисел Фибоначчи по формуле $x_k = x_{k-1} + x_{k-2}$, требует использования 2-х начальных значений $x_0 = 1$ и $x_1 = 1$.

Алгоритм вычисления предела по итерационной формуле:

- Определение начального приближения, которое либо задается, либо определяется из дополнительного условия (например условия сходимости последовательности).
- Организация вычисления последовательности по правилу: $x_1 = f(x_0) \Rightarrow x_2 = f(x_1) \Rightarrow \ldots x_n = f(x_{n-1})$
- Прерывание вычислений производится при достижении косвенной оценки заданной погрешности ε: $|x_{k+1} - x_k| < \varepsilon$

Вычисленный предел является приближенной величиной с заданной погрешностью. Примеры методов и алгоритмов, использующих вычисление предела очень многообразны – итерационные методы решения линейных и нелинейных уравнений, сеточных уравнений, вычисление бесконечных сумм, произведений, комбинаторные вычисления и т.д.. В более сложных случаях последовательность содержит не числа, а некие структуры – вектора, матрицы, строки и т.д.. Однако общий принцип вычисления остается неизменным. Следует учитывать, что итерационная последовательность может сходиться к некоторому пределу или расходится (уход значений на бесконечность). Поэтому вычисление предела последовательности содержит опасность «зацикливания» программы.

Кроме вычисления предела, задача может требовать вычисления суммы, прозведения элементов последовательности и т.д.. Характерно, что эти задачи сводятся к тем же итерационным. Например, сумма последовательности может вычисляться по итерационной формуле для сумм $S_{k+1} = S_k + x_k$, начальное условие $S_0=0$. Аналогично произведение $P_{k+1} = P_k * x_k$, начальное условие $P_0=1$. В этом случае мы вычисляем сразу 2 последовательности S_k и x_k , вторая последовательность называется подпоследовательностью. Выделение подпоследовательностей часто дает простое решение для сложных задач:

Пример: Вычисление функции $\sin(x) = \sum(-1)^n x^{2n+1} / (2n+1)!$

Здесь выделяется подпоследовательность элементов ряда.

Обозначим элемент ряда $A_k = A_{k-1} * g$ тогда => $g = A_k / A_{k-1} = (-1) x^2 / [(2n+1)*2*n]$. В результате имеем: $S_{k+1} = S_k + A_k$, $A_k = A_{k-1} * g$, $g = (-1) x^2 / [(2n+1)*2*n]$, $S_0=0$, $A_0 = x$.

Пример. Вычисление полинома по формуле Горнера. Составим алгоритм вычисления *значения* полинома $P_n(x) = a_0x^n + a_1x^{n-1} + ... + a_n$ для заданного *значения* x. Метод решения – метод (схема) Горнера. Опишем его. Заметим, что:

1) при $n = 0$, $P_0(x) = a_0$;

2) при n = 1, $P_1(x) = a_0x + a_1 = P_0(x)\,x + a_1$;

3) при n = 2, $P_2(x) = a_0x^2 + a_1x + a_2 = (a_0x + a_1)x + a_2 = P_1(x)x + a_2$;

. . .

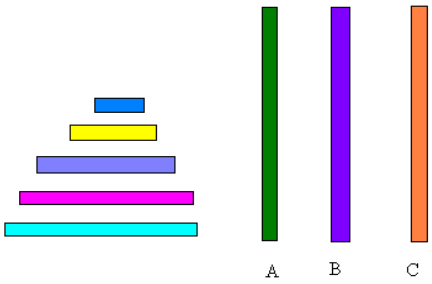
n) $P_n(x) = a_0x^n + a_1^{xn-1} + ... + a_{n-1}x + a_n = (a_0x^{n-}1 + a_1x^{n-2} + ... + a_{n-1})x + a_n = P_{n-1}(x)x + a_n$.

Таким образом, всегда верно рекуррентное соотношение Горнера:

$$P_{n+1} = P_n * x + a_n$$

- **Рекурсивные алгоритмы.** Метод заключается в представлении алгоритма вычисления функции в виде рекурсивной формулы. Примером является задача о Ханойских башнях.

Есть 3 стержня и кольца разных диаметров, которые надеваются на стержни. Причем разрешается помещать кольцо только на кольцо большего диаметра:



Задача: нужно переместить кольца со стержня A на C, используя стержень B как вспомогательный. Для перемещения 2-х колец алгоритм простой — A->B, A->C,B->C. Для большего числа колец алгоритм начинает усложняться и составить процедурый алгоритм для N-колец очень сложно. При рекурсии задача решается просто:

а) Перенесем N-1 кольцо с A на B.

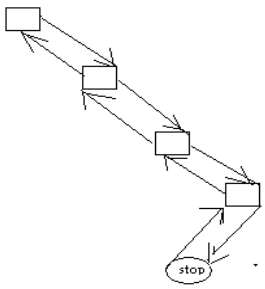
б) Перенесем 1 кольцо с A на C.

в) Перенесем N-1 кольцо с B на C.

т.е. Задача с N кольцом решается через задачу с N-1 кольцом. Тогда N->N-1->N-2->N-3->...2 => задача решена.

Рекурсия – вызов процедуры или функции самой себя. Существует много задач легко решаемых рекурсией, более того, доказано, что любую алгоритмическую задачу можно решить рекурсивно. Это методика рекурсивного программирования.

Действие рекурсии можно представлять как вызов процедуры 1 , затем такой же процедуры 2 и т.д. этот вызов должен быть конечным, следовательно, должно быть условие остановки рекурсии. Перед каждым вызовом новой итерации рекурсии это условие должно проверяться. После остановки возвращаемся назад, таким образом, идет движение по кругу: вниз потом вверх (спуск и подъем рекурсии).



Рекурсия необходима для выполнения каких-либо действий, поэтому выделяют следующие варианты использования рекурсии:

1. Рекурсия с выполнением действий на спуске (до вызова рекурсии).
2. Рекурсия с выполнением действий на подъеме (после вызова рекурсии).
3. Рекурсия с выполнением действий на спуске и подъеме.

Если сравнить рекурсию и цикл, то главное отличие в том, что цикл – это повторение действий, рекурсия – новый вызов (новый вариант). Это два принципиально разных подхода к выполнению действий. Теоретически доказано, что оба подхода эквивалентны, но на самом деле простые задачи проще понимаются циклом, а сложные – рекурсией. Таким образом, рекурсивный подход к логическому решению более удобен и логичен.

Следует учитывать, что рекурсивное программирование и циклическое программирование настраивает программистов на разное понимание алгоритмизации. Поэтому навыки программирования в процедурном языке могут быть даже вредны для рекурсивного программирования.

- **Метод бинарных деревьев.** Суть метода в представлении решения задачи в виде поиска по бинарному дереву. Примером является задача бинарного поиска. Данный метод является так же усложненной формой метода деления задачи на подзадачи. Например, в алгоритме «ветвей и границ» используемого для поиска целочисленного решения задачи оптимизации (поиск максимального или минимального значения многомерной функции), исходную задачу делят на 2 задачи с помощью 2-х дополнительных условий. Эти условия подбирается так, чтобы целочисленные варианты решения удовлетворяли одному из условий, а наилучшее не целочисленное решение не удовлетворяло ни одному из условий. Таким образом, при построении новой ветки бинарного дерева удаляется хотя бы одно не целочисленное решение. Алгоритм работает до тех пор, пока не будет получено целочисленное решение.
- **Метод балансировки бинарных деревьев.** При построении бинарных деревьев желательно сделать его полностью сбалансированным. Сбалансированное бинарное дерево ускоряет алгоритм поиска по дереву. Время выполнения операций пропорционально высоте дерева. Если двоичное дерево плотно заполнено, то его высота пропорциональна логарифму (по основанию 2) от числа вершин и скорость то время $O(\log_2(n))$. Если дерево представляется как простая цепочка, то время поиска линейно от числа вершин $O(n)$. Таким образом, балансировка или «уплотнение» двоичного дерева существенно ускоряет алгоритм. В настоящее время разработаны различные варианты алгоритмов балансировки двоичных деревьев поиска (метод Адельсон-Вельского и Ландиса, Хопкрофта, Байера для красно-черных деревьев и т.д.).
- Метод динамического программирования. Данный метод обычно применяется в задачах поиска оптимального решения, которое зависит от времени. Время, как правило, можно представить дискретным набором значений – этапов или шагов. Таким образом, динамическая задача представляется как многоэтапная. Характерным примером метода решения многоэтапной задачи является алгоритм динамического программирования Беллмана. В нем задача представляется в обратном порядке, от последнего этапа до первого (метод обратного планирования). На каждом шаге строится набор вариантов для этапа An, для совокупности этапов (An-1;An), для совокупности этапов (An-2, An-1;An) и т.д.. При достижении алгоритмом 1-го этапа определяется оптимальное решение. Скорость используемого алгоритма будет существенно выше алгоритма полного перебора.
- **Метод последовательного перехода.** Суть метода – оптимальное решение находится в одной точке из конечного множества точек многомерного пространства. Можно организовать полный перебор вариантов. Однако быстрее будет работать алгоритм, который будет проверять подмножество смежных (наиболее близких) точек. Наилучшая из смежных точек, на следующем шаге алгоритма становится новой опорной точкой, которая формирует подмножество смежных точек. Скорость используемого алгоритма будет существенно выше алгоритма полного перебора. Примером может служить алгоритм Симплекс-метода решения оптимизационной задачи линейного программирования.
- **Метод жадного алгоритма.** В данном случае производится проверка на существование жадного алгоритма. Если такой алгоритм решения задачи существует, то его используют. Если жадного алгоритма решения задачи не существует, то часто задачу преобразуют так, чтобы она решалась жадным алгоритмом.

27.Основные методы эффективного представления данных – основные модели данных.

Основные модели данных

Одним из методов эффективного представления данных в ЭВМ – использование баз данных.

О. База данных (БД) – совокупность специальным образом организованных данных, хранимых в памяти ЭВМ и отображающих состояние объектов и их взаимосвязей в рассматриваемой предметной области.

Таким образом, БД всегда связана с предметной областью и вопрос разработки базы данных обязательно требует изучения этой предметной области. Технология изучения предметной области опирается на методы системного анализа, которые прежде всего требуют определения или выделения главной структуры, основных сущностей и связей между сущностями в изучаемой области. Для наиболее полного и точного описания таких структур для БД принято использовать различную логическую структуру организации данных, которая называется *моделью представления данных (информационной моделью)*. К числу классических относятся следующие модели данных:

- иерархическая;
- сетевая;
- реляционная.

Кроме того, в последние годы появились и стали более активно внедряться на практике следующие модели данных:

- постреляционная;
- многомерная;
- объектно-ориентированная.

В иерархической модели данные представляются в виде древовидной (иерархической) структуры. Главная особенность такой структуры – все элементы данных разделены по уровням иерархии. При этом данные связаны только между соседними уровнями. Данные одного уровня или уровней, не являющихся соседними не связываются между собой.

Сетевая модель организует представление данных в виде произвольного графа. Она имеет те же основные составляющие (узел, уровень, связь), однако характер их отношений принципиально иной. В сетевой модели принята свободная связь между элементами разных уровней.

Реляционная модель данных (РМД) название получила от английского термина *relation* – отношение. Термин указывает, что такая модель хранения данных построена на установке отношений между составляющих ее частей. Практически структура РМД представляет собой двухмерный массив (двухмерную таблицу) либо совокупность связанных между собой таблиц. Каждая строка таблицы называется записью, а столбец – полем.

Постреляционная модель данных представляет собой расширенную реляционную модель, в которой отменено требование не делимости поля. Здесь возможны многозначные поля, которые сами фактически являются таблицей. Возможны так же объединение полей в группы (ассоциации). Размер и количество полей при этом может динамически меняться.

Достоинством постреляционной модели является представление множества таблиц в виде одной, что обеспечивает наглядность и упрощает анализ и разработку БД. Недостатком этой модели является сложность решения проблемы обеспечения целостности и непротиворечивости данных.

Многомерная модель фактически является частным случаем постреляционной. Главное отличие состоит в фиксировании типов возможных структур. Так выделяют объемные структуры, «снежинка», «звезда», «созвездие». Простейший вариант трехмерной объемной структуры можно представить как куб, разделенный на малые кубики содержащие информацию. Каждое измерение куба – отдельный параметр (например цена, марка, год выпуска). Практически могут использоваться и 4-х, 5-ти и более размерные гиперкубы. В вариантах «снежинка», «звезда», «созвездие» таблицы объединяются в структуры, геометрия которых соответствует названию.

Объектно-ориентированные БД формируют сеть на основе объектов. Объекты имеют набор полей (аналогично записи) и набор методов для работы с данными полями. Наличие методов обеспечивает более корректную и правильную работу с данными полей. Связи между

объектами носят характер ссылки в памяти, что облегчает реализацию БД в ЭВМ и оптимизацию обработки данных. Фактически эта модель объединяют в себе достоинства двух моделей -реляционной и сетевой.

Более сложные структуры данных могут быть описаны с помощью баз знаний. В компьютерных интеллектуальных системах для хранения и использования знаний создаются специальные представления знаний, включающие в свой состав всю совокупность процедур, необходимых для записи знаний, извлечения их из памяти и поддержки хранилища знаний в рабочем состоянии. Системы представления знаний часто оформляются как базы знаний, являющиеся естественным развитием баз данных. Теория баз знаний составляет сейчас заметную часть искусственного интеллекта.

Для представления знаний в памяти системы разрабатываются разнообразные модели представления знаний. В настоящее время в интеллектуальных системах используются четыре основных модели знаний:

1 - семантические сети; 2 – фреймы: 3 - логические системы; 4 - продукции

Семантическая модель наиболее близка к тому, как представляются знания в текстах на естественном языке. В ее основе лежит идея о том, что вся необходимая информация может быть описана как совокупность троек вида: (aRb), где a и b два объекта или понятия, а R – двоичное отношения между ними. Такая модель может графически представляться в виде сети, в которой вершинам соответствуют объекты или понятия, а дугам – отношения между ними. Такая модель носит название ***семантической сети***. Впервые такое представление знаний под названием синтагматических цепей было использовано в работах по ситуационному управлению, получивших развитие в СССР в середине шестидесятых годов.

Основными элементами семантической сети являются вершины и дуги. Вершины сети – объекты, которые определяют сущности и понятия предметной области, дуги – отношения между ними. Связи в сети выражаются глаголами или прилагательными.

Логическая модель знаний. Наиболее распространенная модель знаний опирается на классическую аксиоматическую систему. Это либо логические исчисление, типа исчисления предикатов и его расширений, либо дедуктивная система, основанная на использовании дедуктивных правил и метода резолюции (метода резолюции - аналог доказательства от противного). Эти две модели знаний отличаются ярко выраженной процедурной формой. Поэтому часто говорят, что они описывают процедурные знания, а модели знаний отличаются явно выраженной процедурной формой. В отличии от этой системы модели знаний, опирающиеся на семантические сети описывают декларативные знания.

Системы продукций, то есть правил вида: “Если А, то В”, задающих элементарные шаги преобразований и умозаключений могут служить основой для представления знаний. Продукционная модель в целом схожа с логической, главное ее отличие в замене правил вывода продукционными правилами. Если правила вывода опираются на классические законы логики, то продукционные правила строятся как синтаксические генераторы новых форм. Классическим примером продукционных правил являются нотации Бекуса-Наура, разработанные в 60-е годы для описания алгоритмов работы синтаксических анализаторов.

Фреймовые представления знаний широко распространенны в ИИ. Понятие «*фрейм*» в искусственном интеллекте было введено М. Минским, который под фреймом объекта или явления понимал то его минимальное описание, которое содержит всю существенную информацию об этом объекте или явлении, и обладает тем свойством, что удаление из описания любой части приводит к потере существенной информации, без которой описание объекта или явления не может быть достаточным для идентификации. В дальнейшем структурам фрейма была стандартизирована. Сегодня структура фрейма содержит слоты и демоны, где каждый слот есть пара вида: (Имя слота; Значение слота). Таким образом, слот можно сравнить с полем класса. В качестве значений слота могут выступать переменные, константы, любые выражения в

выбранной предметной области, ссылки на другие фреймы и слоты. Таким образом, фрейм представляет собой достаточно гибкую конструкцию, позволяющую отображать в памяти интеллектуальной системы разнообразные знания. Под демонами понимают процедуры, которые связаны с слотами или всем фреймом (демоны слота и демоны фрейма). Аналогия фреймов с классами, позволяет сравнить демоны с методами класса.

28.Основные методы эффективного представления данных - динамические структуры данных.

Динамические структуры данных

Данные в программировании принято делить на статические и динамические. Статические данные описываются привычным образом и не могут менять свою структуру в процессе исполнения программы. Традиционно операционная система выделяет программе 3 сегмента (данных, кода и стека). Сегмент данных хранит статические данные программы. При наличии динамических структур они будут помещаться в любом свободном месте оперативной памяти. Таким образом, динамические данные обладают двумя важными преимуществами – возможность модификации структуры и дополнительная память. Современные системы программирования это активно используют, так Delphi все классы делает динамическими, поэтому их объекты размещаются вне статической памяти.

Для доступа к динамическому объекту используют специальный объект – указатель. Фактически указатель только хранит адрес первой ячейки памяти объекта и его общий размер. Размер зависит от типа объекта, поэтому указатель связан с определенным типом объекта. Этот указатель называют типизированный. Существует и особая форма не типизированного указателя TPointer, которая используется только в особых случаях. Есть и особая константа Nil для пустого указателя.

Синтаксис описания указательного типа пользователя:

type

Имя_типа=^тип;

Синтаксис описания переменной указательного типа:

Var R:^integer;

Динамические переменные и значения их указателей создаются с помощью стандартных процедур New(имя переменной). Освобождение памяти с помощью Dispose(имя). Для работы с не типизированными указателями используются процедуры GetMem и FreeMem. Такой сложный вариант предусмотрен для того, чтобы выделение памяти для динамического объекта происходило под управлением операционной системы.

Для работы с указателями вводятся особые операции:

^ - используется при описании (стоит перед именем типа, на который ссылаются), используется для доступа к содержимому ячейки на который ссылается указатель (стоит после имени указателя).

@ - получение адреса объекта (используется для направления указателя на реальный объект).

+\- - указатели можно складывать или вычитать как целые числа, можно прибавлять или вычитать из указателя целое число (фактически это сдвиг адреса на определенное количество байт).

С помощью указателей можно создавать динамические структуры данных – списки, стеки, очереди.

Стеком называется *динамическая структура данных*, у которой в каждый момент времени доступен только верхний (последний) элемент. Лучше всего понятие *стека* можно проиллюстрировать примером стопки книг: в стопку можно добавить еще одну книжку, положив ее сверху; из стопки можно взять, не разрушив ее, только верхнюю книгу. А для того, чтобы достать книжку из середины стопки, необходимо сначала убрать по одной все лежащие выше нее. Последовательность обработки элементов *стека* хорошо отражают аббревиатуры **LIFO**

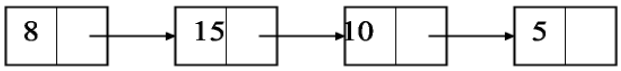
(**L**ast **I**n **F**irst **O**ut - "последним вошел, первым вышел") и **FILO** (**F**irst **I**n **L**ast **O**ut - "первым вошел, последним вышел").

Очередью называется *динамическая структура*, у которой в каждый момент времени доступны два элемента: первый и последний. Из начала *очереди* элементы можно удалять, а к концу - добавлять. Примером *очереди* программистской вполне может служить *очередь* обывательская: скажем, в продуктовом магазине. Последовательность обработки элементов *очереди* хорошо отражают аббревиатуры **LILO** (**L**ast **I**n **L**ast **O**ut - "последним вошел, последним вышел") и **FIFO** (**F**irst **I**n **F**irst **O**ut - "первым вошел, первым вышел").Список – это более общая структура, где возможен доступ к любому элементу, но по определенному правилу и через указатели.

Списком называется упорядоченная структура, каждый элемент которой содержит ссылку, связывающую его со следующим элементом. Для организации списков используются самоадресующиеся записи. Пример:

```
Type
   DT = Record
   I: integer;D:^DT;
End;
```

В примере указывается описание типа записи, внутри которой есть поле d которое ссылается на такую же запись. Появляется возможность связать (с помощью указателей) например 4 записи, которые содержат информацию (8,15,10,5). Таким образом, список имеет в каждом элементе указатель для связи и информационную часть.



Приведенный пример показывает вариант односвязного списка, где элемент имеет только 1 указатель для связи. Это осложняет работу с списком, так как перемещение возможно только от начала к концу списка. Другой вариант записи, дает возможность создать двусвязный список, где ходить можно в обе стороны:

```
Type
   DT = Record
   I: integer;F:^DT;E:^DT;
End;
```

Две связи дают возможность образовывать дерево и другие нелинейные структуры. На практике одно- и двух- связные списки могут быть основой таких структур как линейный список, список-кольцо, список-дерево и т.д.. Для работы с списками нужна разработка процедур добавления и удаления элементов, соединения и разделения списков, поиска и сортировки данных в списке. В результате *освобождения памяти* значение *указателя*, хранившего *адрес* освобожденной области, становится неопределенным. Для того чтобы сохранить информацию обо всем списке, достаточно только одной переменной - *указателя* на первый элемент этого списка. Обычно его называют ***головой списка***. *Указатель* на *голову* должен быть выделенным: с ним нельзя производить никаких действий, которые могут стать причиной утери всего списка. Для работы со списком обычно заводят вспомогательный *указатель*.

Лекция №12

Раздел №3. Основы теории алгоритмизации

Тема 3.2 Алгоритмы оптимизации на сетях и графах

Содержание: Алгоритмы оптимизации на сетях и графах. Понятие жадного алгоритма. Алгоритмы Прима и Краскала. Алгоритмы Дейкстры и Флойда. Примеры решения задач.

31.Понятие жадного алгоритма. Алгоритмы оптимизации на сетях и графах. Алгоритмы Прима и Краскала.

Алгоритмы оптимизации на сетях и графа

Граф можно представить как совокупность двух множеств – множества вершин и множества ребер, связанных с вершинами (инцидентных вершинам). Таким образом, граф можно

было бы описать в ЭВМ двумя связанными массивами, однако существуют более удобные варианты:

- Матрица смежности. Квадратная матрица, в которой каждая строка и столбец соответствуют одной вершине графа. Если вершины связаны ребром, то на пересечении соответствующих строк и столбцов записывается число. Если связи нет, то записывается ноль. Есть вариант, когда записывают только число 1 (нет длин ребер) или число равное длине ребра. Матрица смежности симметрична относительно главной диагонали, на главной диагонали все значения равны нулю, количество ненулевых элементов строки или столбца равно степени соответствующей вершины.
- Матрица инциндентности. Матрица в которой строки связаны с вершинами графа, а столбцы с ребрами графа. В общем случае не квадратная. Как правило содержит нули и единицы. Ноль при отсутствии соединения вершины и ребра, 1 при наличии инциндентности. Главное свойство этой матрицы – число ненулевых элементов любого столбца равна 2. Количество ненулевых элементов строки равно степени соответствующей вершины.
- Список ребер. Пронумерованный список пар вершин, которые связаны с ребром данного номера.

Наличие нескольких вариантов описания графа требует прежде всего построения алгоритмов проверки правильности введенных данных и возможности конвертации данных из одного варианта в другой.

Следующей задачей является визуализация графа. Так как любой граф можно изобразить на плоскости, то подобная задача имеет простой смысл: изображаем в случайных, несовпадающих точках вершины-круги и соединяем их отрезками, соответствующими ребрам графа.

Алгоритмы решения задач на графах

Среди множества задач, связанных с графами выделим 3 важнейших задачи оптимизации:

- Задача Прима-Краскала
- Задача Дейкстры
- Задача Форда-Фалкерсона

О. Задача оптимизации — поиск максимума или минимума некой целевой функции. Задача оптимизации на графе означает, что целевая функция задается на графе. Это может быть взвешенный граф, или функция определяется в вершинах, или просто функция зависит от числа ребер (вершин) на некотором пути. Иногда функция формируется путем подсчетов числа маршрутов, циклов, путей, неких структур (например треугольников одного цвета) в графе. Такими являются многие комбинаторные задачи. Наличие графа здесь фактически усложняет поиск, так как в связанном графе не все все пути возможны.

Понятие жадного алгоритма

Жадным называется алгоритм, который для получения оптимального общего решения на каждом шаге выбирает из всех возможностей самый оптимальный. Так если мы ищем максимальное решение, то на каждом шаге мы должны выбирать вариант дающий максимальный вклад в общую сумму. Очевидно, что ни каждая задача может быть решена жадным алгоритмом. Но можно выделить класс задач, для решения которых существует тот или иной жадный алгоритм.

Поиск жадного алгоритма важен простотой его реализации. Очевидно, что выбор вариант по максимальному вкладу дает простейшую алгоритмизацию задачи. При этом к таким задачам относятся многие практически важные задачи, которые на первый взгляд не могут решаться жадным алгоритмом. В частности все 3 указанные выше оптимизационные задачина графах имеют жадные алгоритмы, хотя для некоторых задач они не самые эффективные.

Примером эффективного жадного алгоритма является алгоритм кодирования Хафмена. Фактически алгоритм Хафмена – поиск оптимального кодирования с помощью двоичного дерева. С помощью двоичного дерева реализуется двоичное префиксное кодирование,

которое не требует разделяющих знаков и однозначно декодируется. Поскольку в методе Хафмена выбирается для объединения всегда минимальные элементы, то данный алгоритм имеет явный «жадный» характер.

Метод жадного алгоритма. В данном случае производится проверка на существование жадного алгоритма. Если такой алгоритм решения задачи существует, то его используют. Если жадного алгоритма решения задачи не существует, то часто задачу преобразуют так, чтобы она решалась жадным алгоритмом.

Алгоритмы Прима и Краскала

Задача Прима-Краскала связана с поиском остовного дерева графа минимальной длины. Граф должен иметь длины ребер, быть связным. Длиной графа будем считать сумму длин его ребер. По алгоритму Прима искомое остовное дерево получается путем последовательного подсоединения к вершинам графа его ребер. На первом шаге все ребра удаляются. Затем ребра сортируются в порядке возрастания длины. Первое добавленное ребро – кратчайшее, далее ребра выбираются по особой процедуре:

- Выделяется множество вершин графа которые уже связаны с установленными ребрами (множество А). Множество В – все остальные вершины.
- Выделяем множество ребер, которые соединяют вершины множеств А и В, но не соединяют вершины одного множества.
- Из выделенного множества ребер находим кратчайшее, его добавляем в граф.

Алгоритм повторяют до тех пор, пока множество В не станет пустым.

Алгоритм Краскала более сложен для реализации. Его действие противоположно алгоритму Прима. Берем исходный граф со всеми существующими ребрами и затем последовательно удаляем ребра. Проблема здесь в том, что нельзя просто удалить длиннейшее ребро, надо сначала убедиться, что оно циклическое. Это значит, что это ребро образует с другими замкнутый путь. Для этого нужно удалить изучаемое ребро из графа и проверить, есть ли последовательность ребер соединяющая теперь вершины этого ребра. Здесь так же используются 2 множества вершин. В первое множество помещают одну вершину ребра, добавляют все вершины с ней связанные, далее все вершины связанные с вновь добавленными вершинами и так пока не найдется связь с другой вершиной ребра или добавление вершин прекратится, что говорит о нарушении связности графа.

32.Алгоритмы оптимизации на сетях и графах. Алгоритмы Дейкстры и Флойда.

Алгоритмы Дейкстры и Флойда

Задача Дейкстры связана с поиском кратчайших путей между вершинами графа при наличии длин ребер.

Алгоритм использует три массива из n чисел каждый. Первый массив А содержит метки с двумя значениями: 0 и 1, второй массив В содержит расстояния - текущие кратчайшие расстояния между соответствующими вершинами, третий массив содержит обозначения вершин. Матрица расстояний d_{ik} задает длины дуг d_{ik}, если такой дуги нет, то d_{ik} присваивается большое число, равное «сумме всех расстояний ребер графа». Начиная с заданной вершины К алгоритм заполняет массив В. На первом шаге вносятся в В расстояния от К до всех вершин. Вершина К в массиве А помечается. Среди непомеченных вершин ищут вершину, которая имеет минимальное число в В. Далее эту вершину помечают и изменяют значения в массиве В для непомеченных вершин по правилу:

> Если сумма расстояния для последней помеченной вершины и расстояние от нее до данной вершины меньше записанного раннее значения, то записывают в массив В эту сумму. Теперь эта сумма считается новым расстоянием для проверяемой вершины.

Алгоритм выполняют до того момента, когда не остается не помеченных вершин.

Алгоритм Уоршелла-Флойда – усовершенствование метода Дейкстры для решения задачи поиска всех расстояний между всеми вершинами. Его можно представить в виде программы Паскаля:

```
For k:=1 to N do
For i:=1 to N do
```

For j:=1 to N do

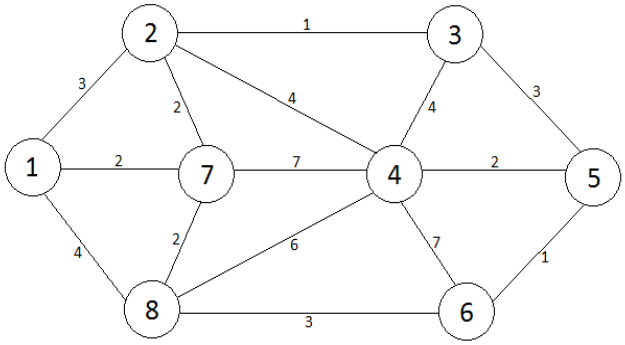
d[i,j]:= min(d[i,j], d[i,k]+d[k,j]);

Здесь d[i,j] – массив расстояний, min – функция вычисляющая минимальное из 2 значений.

Примеры решения задач

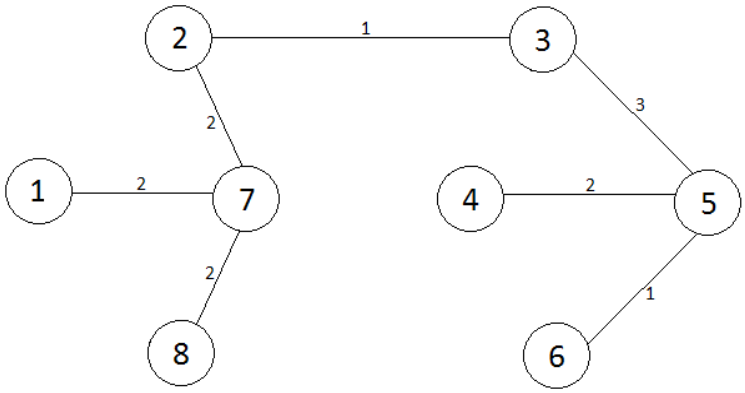
Задача Прима-Краскала

Строим граф. Найти кратчайший путь и построить остовное дерево минимальной длины, используя алгоритм Прима и Краскала.



Перепишем ребра в порядке возрастания. Строим граф согласно алгоритму.

23	1+
56	1+
17	2+
27	2+
78	2+
45	2+
12	3-
35	3+
68	3
24	4
18	4
34	4
48	6
46	7
47	7



Для алгоритма Краскала процесс обратный, но дерево получается то же.

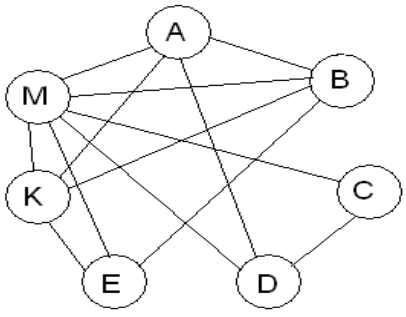
Решение задачи Дейкстра

Найти расстояния от A до всех остальных точек.

Условие: Вершины:A, B, C, D, E, K, M

AB=7 AD=12 AM=5 BE=11 BK=9 BM=7 CM=6 CD=10 EM=5 EK=12 DM=5 KM=6 AK=6.

Изобразим граф задачи:



Составляем таблицу: По горизонтали, в первой строке(шапка) указываем вершины от А до М, по вертикали указываются пути, от точки А до всех остальных.

	A	B	C	D	E	K	M
A	0	7	∞	12	∞	6	5
5+M	-	7	11	10	10	6	5
6+K	-	7	11	10	10	6	-
7+B	-	7	11	10	10	-	-
10+D	-	-	11	10	10	-	-
10+E	-	-	11	-	10	-	-

По строкам заполняем таблицу, учитывая растояния до отмеченных вершин. По заполненной таблице формируем ответ.

Ответ: A→A , d=0 A→B, d=7 A→M→C, d=11 A→M→D, d=10 A→M→E, d=10 A→K, d=6 A→M, d=5

Лекция №13

Раздел №3. Основы теории алгоритмизации

Тема 3.2 Алгоритмы оптимизации на сетях и графах

Содержание: Задача Форда-Фалкерсона о потоках в сетях. Примеры решения задач. Матроиды. Основные свойства матроидов, теорема Радо-Эдмондса.

33.Алгоритмы оптимизации на сетях и графах. Задача Форда-Фалкерсона о потоках в сетях. Алгоритмы решения задачи о максимальном потоке.

Задача Форда-Фалкерсона о потоках в сетях

Сетью называется связный граф с выделенной парой вершин – входом и выходом. Если есть несколько входов или выходов, то создают виртуальный вход или выход, соединенный со всеми входами или выходами. Транспортной сетью называется сеть (чаще всего с дугами, как ориентированный граф) с заданной на дугах специальной функцией – пропускной способностью (которая должна быть положительной).

Вершина начала транспортной сети, из которой дуги только выходят. Вершина выхода транспортной сети, в которую дуги только входят. На множестве дуг b задана целочисленная функция $c(b) > 0$, где $c(b)$ – пропускная способность дуги b.

Определение. Потоком по транспортной сети называется положительно определенная функция $\varphi(b) \geq 0$, заданная на множестве дуг b и обладающая следующими свойствами:

А) Для любой дуги $c(b) \geq \varphi(b)$. Б) Сумма входных потоков в любой вершина сети равна сумме выходных потоков.

Свойство Б утверждает, что поток, входящий в вершину, равен выходящему потоку (поток в вершинах не скапливается). Это свойство называют правилом (законом) Кирхгофа.

Дуга называется насыщенной если для нее $c= \varphi$.

Набор дуг, удаление которых ведет к разрыву связи между входом и выходом (связь есть, если есть путь между вершинами) называют сечением сети.

Простым сечением называют сечение в котором нельзя удалить ни одну дугу без потери свойства сечения (нет лишних дуг).

Насыщенным сечением является сечение в котором все дуги насыщены.

Пропускной способностью сечения называют сумму пропускных способностей ее дуг.

Задача Форда- Фалкерсона (ФФ) – поиск максимально возможного потока в транспортной сети.

Теорема Форда- Фалкерсона – максимальный поток в сети равен минимальной пропускной способности сечений этой сети.

Методы решения задачи Форда-Фалкерсона

Для решения задачи нужно найти набор дуг (сечение), которое обладает следующими свойствами:

- Все дуги набора(сечения) насыщены.
- Удаление дуг набора(сечения) приводит к разрыву связей между входом и выходом.
- Для разрыва связи нужны все дуги набора(сечения). Если удалить любую дугу, то сечение не будет удовлетворять 2-му свойству.

Поиск решения имеет множество различных алгоритмов. Рассмотрим 2 из них:

А) Метод обратного планирования Беллмана. Суть метода определяется алгоритмом:

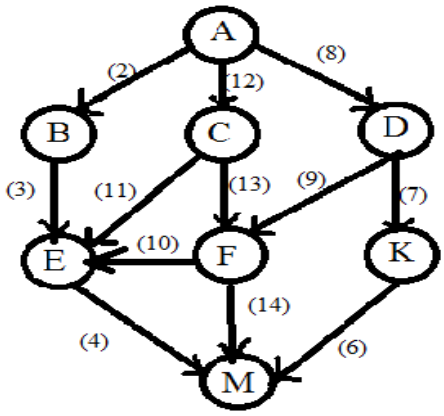
- Выбираем выходную вершину как слой номер Т.
- Рассматриваем все дуги, входящие в слой Т. Насыщаем их.
- Рассматриваем все вершины, откуда начинаются дуги, входящие в слой Т. Определяем их как слой Т-1.
- Рассматриваем все дуги, входящие в слой Т-1. Определяем их потоки, так чтобы для вершин слоя Т-1 выполнялось правило Киргофа. Если это не удается, то нужно уменьшить выходные потоки в вершины слоя Т.
- Алгоритм повторяется до тех пор пока потоки всех дуг не будут определены.
- Результат распределения должен быть максимальным потоком. Убеждаемся в этом находя насыщенное сечение сети. Поток в слое Т будет тогда равен максимальному потоку.

Б) Метод последовательного перехода. Суть метода определяется алгоритмом:

- 1) Находим опорный поток. Например выбираем поток равный 0 везде.
- 2) Находим путь П между входом и выходом, в котором все дуги не насыщенны.
- 3) Находим дугу В на пути П для которой разница с-φ минимальна.
- 4) Увеличиваем поток по дуге П так, чтобы дуга В была насыщенна.
- 5) Помечаем насыщенную дугу В (можно даже для наглядности удалять ее из сети).
- 6) Добавляем В к списку насыщенных дуг и проверяем не является ли полученный список сечением. Если список сечение, то проверяем – нельзя ли удалить из него дуги и получить простое сечение. Если список еще не сечение, то повторяем п.3-6.
- 7) Вычисляем пропускную способность полученного простого сечения. По теореме Форда-Фалкерсона эта величина равна максимальному потоку.

Примеры решения задач

Задача Форда-Фалкерсона с построением графа сети. По данным строим граф. Вход А, выход М:



Пропускные способности ребер:

AB=2, AD=8, AC=12, BE=3, CE=11, CF=13, DF=9, DK=7, FE=10, EM=4, FM=14, KM=6

1 шаг Начальный поток P=0

Ищем путь от входа к выходу. Путь: A->B->E->M

Находим в этом пути самую минимальную пропускную способность(с учетом потока). Min: AB=2 ==>увеличиваем P на 2 P=P+2 ==> Насыщаем AB

2 шаг Поток P=2

Ищем путь, который не проходит через насыщенные ребра. Путь: A->D->K->M

min: KM=6 ==>Увеличиваем P на 6 P=P+6 ==>Насыщаем KM

3 шаг Поток P=8 Путь A->C->F->M min: AC=12 P=P+12 ==> Насыщаем AC

4 шаг Поток P=20. Путь: A->D->F->M min: AD=2 P=P+2 ==> Насыщаем AD, FM. Найденное насыщенное сечение S=AC+AD+AB=2+12+8=22 => максимальный поток 22. Ответ: P=22.

36.Понятие жадного алгоритма. Матроиды и их свойства.

Матроиды

В теории графов играют важную роль жадные алгоритмы. Они просты для понимания и реализации, работают сравнительно быстро, известно много разнообразных задач, которые можно решить с помощью жадных алгоритмов. Однако не всегда можно доказать возможность применимости жадного алгоритма для нахождения точного решения многих задач.

Жадными (градиентными) называют алгоритмы, действующие по принципу "максимальный выигрыш на каждом шаге". Такая стратегия не всегда ведет к конечному успеху - иногда выгоднее сделать не наилучший, казалось бы, выбор на очередном шаге с тем, чтобы в итоге получить оптимальное решение. Но для некоторых задач применение жадных алгоритмов оказывается оправданным.

Рассмотрим теоретическую основу жадных алгоритмов – теорию матроидов. При помощи нее можно довольно часто установить возможность применимости жадного алгоритма. Для этого необходимо, чтобы исследуемое множество являлось матроидом. Когда возник вопрос о том, каковы обстоятельства, при которых жадный алгоритм результативен, или иначе - что особенного в тех задачах, для которых он дает точное решение, оказалось, что математическая теория, с помощью которой что-то в этом можно прояснить, уже существует. Это теория матроидов, основы которой были заложены Уитни в работе 1935 г. Целью создания теории матроидов было изучение комбинаторного аспекта линейной независимости, а в дальнейшем обнаружились разнообразные применения понятия матроида, первоначально совсем не имевшиеся в виду.

Матроид — классификация подмножеств некоторого множества, представляющая собой обобщение идеи независимости элементов, аналогично независимости элементов линейного пространства, на произвольное множество.

Матроид — пара множеств X и K, где X — конечное множество, называемое носителем матроида (база), а K — некоторое множество подмножеств X, называемое семейством *независимых* множеств (2 особых условия). ***Рангом*** матроида называется мощность его базового множества.

Матроиды – совокупность 2-х множеств (базовое множество и множество «независимых» подмножеств). Подмножества удовлетворяют условиям – если B принадлежит T и A подмножество B => A принадлежит T, A + B принадлежит T и |A|<|B| => существует такой X, что он принадлежит B\A и (A u X) принадлежит T, X – элемент, который добавляют.

Матроидом называется пара множеств E, I, состоящая из конечного множества E, называемого **базовым множеством** матроида, и множества его подмножеств I, называемого **множеством независимых множеств** матроида, причем I удовлетворяет трем свойствам:

1.Множество I не пусто. (Даже если исходное множество E было пусто – $E = \varnothing$, то I будет состоять из одного элемента – множества, содержащего пустое $I = \{\{\varnothing\}\}$).

2.Любое подмножество любого элемента множества I также будет элементом этого множества. (Это свойство понятно – если некоторый набор векторов линейно-независим, то линейно-независимым будет также любой его поднабор).

3.Если X, Y принадлежат I, причем $|X| = |Y| + 1$, тогда существует элемент x принадлежащий $X - Y$, такой что объединение $Y \cup \{x\}$ принадлежит I. (Если мощности двух линейно-

независимых подмножеств X,Y принадлежащих I, отличаются на единицу $|X| = |Y| + 1$, то найдется такой вектор a принадлежащий E и X\Y, который при добавлении к Y также даст линейно-независимое подмножество принадлежащее I).

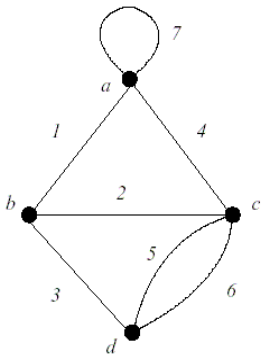
Стоит отметить, что свойство №2 еще называют свойством наследственности.

Основные свойства матроидов, теорема Радо-Эдмондса

Существуют матроиды 2 видов: матричные (множество строк матрицы и подмножества линейно-независимых строк), графовые (множество ребер и подмножества антициклических ребер (в том числе и остовные деревья). Графовый матроид графа Γ будем обозначать М(Г). Векторный матроид матрицы A будем обозначать М[A].

Примеры:

Графовый матроид: Рассмотрим граф на рисунке.



Граф неориентированный и состоит из четырех вершин и семи ребер, одно из которых является петлей. Пусть E будет множеством, состоящим из ребер этого графа, E= {1, 2, 3, 4, 5, 6, 7}, а I будет множеством подмножеств E, таких что каждый элемент I не содержит в себе циклов графа. Эта пара множеств E, I является матроидом, ее называют графовым матроидом и обозначают $M(\Gamma)$.

Рассмотрим матрицу.

1	0	0	1	0	0	0
0	1	0	1	1	1	0
0	0	1	0	1	1	0

Определим множество E, как множество, состоящее из {1, 2, 3, 4, 5, 6, 7} – номеров столбцов матрицы, а множество I, как множество, состоящее из подмножеств E, таких, что векторы, определяемые ими, являются линейно-независимыми. Построенное множество I обладает 3-мя свойствами матроида, т.е будет матроидом.

Теорема. Пусть $\boldsymbol{\Gamma}$ – граф, а $\boldsymbol{A_\Gamma}$ – его матрица инценденций. Если рассматривать $\boldsymbol{A_\Gamma}$ как матрицу, то векторный матроид, построенный на $\boldsymbol{A_\Gamma}$, в качестве независимых множеств будет содержать множества ребер, не содержащих в себе циклов, и М($\boldsymbol{\Gamma}$) = М[$\boldsymbol{A_\Gamma}$].

В матроиде можно определить весовую функцию, которая даст возможность сравнивать подмножества. Максимум или минимум значения этой функции будет определять оптимальный или наилучший набор элементов (подмножество). Таким условиям удовлетворяют многие задачи теории оптимизации, где нужно найти максимум или минимум какой-то целевой функции на множестве возможных значений аргументов. Если это множество удовлетворяет условиям матроида, то можно искать решение жадным алгоритмом. Это доказано в Теорема Радо-Эдмондса. В соотвествии с теоремой Радо-Эдмондса – оптимальное подмножество матроида получается «жадным» алгоритмом следующего вида:

- Сортируются элементы в порядке весов
- Перебираем элементы и если его можно добавить к результату, то добавляем.

Жа́дный алгори́тм Ра́до — Э́дмондса — алгоритм нахождения в матроиде базы минимального веса. Если каждому элементу носителя матроида сопоставлен его вес, и вес подмножества носителя определяется как сумма весов элементов этого подмножества, то алгоритм Радо — Эдмондса позволяет найти среди всех баз матроида базу минимального веса. Алгоритм Радо—

Эдмондса обобщает жадные алгоритмы. Например, будучи применённым к графическому матроиду, он превращается в алгоритм Краскала поиска остовного леса минимального веса.

Реализация

X — носитель матроида, I — семейство независимых множеств.

Для всех i от 0 до ранга матроида строится множество $A_i \in I$ мощности i, вес которого является минимальным среди весов независимых подмножеств той же мощности.

$A_0 = \varnothing$ — очевидно.

Строятся эти множества так: $A_i = A_{i-1} + \{x\}$, где x — минимальный из элементов $y \in X \setminus A_i$ таких, что $A_i \cup \{y\} \in I$.

Ответ задачи — A_n, где n — ранг матроида.

Теорема Радо-Эдмонса

Если M(Г)=(E, I) - матроид, то для любой весовой функции f(b) найдется множество A, которое будет множеством наибольшего веса из I. Если же M(Г)=(E, I) не является матроидом, то найдется такая функция f: E→R, что A не будет множеством наибольшего веса из I.