Введение в теорию алгоритмов (часть 1)

проф. А.В. Цыганов, СПбГУ, 2008

С понятием алгоритма были знакомы уже ученые древних цивилизаций.

Алгоритм Евклида нахождения наибольшего общего делителя двух целых чисел был описан в книге VII «Начал» Евклида, датирующейся 330-320 гг. до н.э.

Прообраз метода исключения Гаусса был описан в китайском источнике «Девять книг по арифметике» (202 г. до н.э. — 9 г.н.э.).

Эратосфен (приблиз. 276–194 гг. до н.э.) предложил алгоритм проверки простоты числа - (раба) Эратосфена.

Для нахождения всех простых чисел не больше заданного числа **n** нужно :

1. Выписать подряд все целые числа от двух до **n** (**2, 3, 4, …, n**).
2. Пусть переменная **p** изначально равна двум — первому простому числу.
3. Вычеркнуть из списка все числа **2p, 3p, 4p, …**
4. Найти первое не вычеркнутое число, большее чем **p**, и присвоить значению переменной **p** это число.
5. Повторять шаги 3 и 4 до тех пор, пока **p** не станет больше, чем **n**
6. Все не вычеркнутые числа в списке — простые числа.

Тем не менее, различные формализации понятия алгоритма были предложены только в середине 30-х годов XX столетия, когда и стала складываться современная теория алгоритмов.

Потребность в точном определении вызвана внутренними тенденциями развития математики, поскольку некоторые открытые проблемы заключались в выяснении **существования** (или **не существования**) **алгоритма для решения конкретных задач**.

«Я принадлежу к тем кибернетикам, которые не видят никаких принципиальных ограничений в кибернетическом подходе к проблеме жизни и полагают, что можно анализировать жизнь во всей ее полноте, в том числе и человеческое сознание со всей его сложностью, методами кибернетики».

А. Н. Колмогоров «Автоматы и жизнь».

Почти одновременно в 30-х годах XX в. независимо (и в разных формах) было формализовано понятие вычислимости.

- **Пост и Тьюринг** определили алгоритм как вычисление на ***абстрактных машинах***.
- **Гедель, Черч и Клини** определили понятие вычислимости на языке ***рекурсивных функций***.
- **Марков** ввел понятие нормального ***алгорифма***.

Все эти определения, совершенно различные по форме, описывают некое единое математическое понятие — понятие **алгоритма**.

Доказательство **алгоритмической неразрешимости** многих известных задач и получение ряда других отрицательных результатов послужило хорошим стимулом развития классической теории алгоритмов.

Эти исследования непосредственно предшествовали появлению первых компьютеров в 40-х годах XX века, а теория алгоритмов явилась теоретическим фундаментом для создания и использования вычислительных машин.

С практической точки зрения часто нет никакой разницы между **неразрешимой** задачей и задачей, решаемой за **экспоненциальное** от длины входа время.

Постепенно сформировалось понятие **«эффективного»** алгоритма, под которым к 1970 г. стали понимать любой **полиномиальный** алгоритм, т.е. алгоритм, время выполнения которого ограничено некоторым **полиномом** от длины записи входных данных.

Задачи, разрешимые такими алгоритмами, образуют класс, который стали обозначать через $\mathcal{P}$.

В начале 1970-х годов в работах С. Кука, Р. Карпа и Л. Левина была разработана математическая теория **недетерминированных** вычисления и введен класс $\mathcal{NP}$ задач. Задачи, принадлежащие классу $\mathcal{NP}$, можно охарактеризовать как задачи, имеющие «короткое доказательство».

К классу $\mathcal{P}$ относятся задачи, которые могут быть решены за время, **полиномиально** зависящее от объёма исходных данных, с помощью **детерминированной** вычислительной машины (например, машины Тьюринга)

К классу $\mathcal{NP}$ — задачи, которые могут быть решены за **полиномиально** выраженное время с помощью **недетерминированной** вычислительной машины, то есть машины, следующее состояние которой не всегда однозначно определяется предыдущими.

Другое определение класса $\mathcal{NP}$ - задачи, решение которых с помощью дополнительной информации полиномиальной длины, данной нам свыше, **мы можем проверить за полиномиальное время**.

В частности, к классу $\mathcal{NP}$ относятся все задачи, решение которых можно проверить за полиномиальное время.

Класс $\mathcal{P}$ содержится в классе $\mathcal{NP}$.

Классическим примером $\mathcal{NP}$ -задачи является задача о коммивояжёре.

Пример: **Алгоритмы устойчивой сортировки**

- Сортировка **пузырьком** (англ. Bubble sort) — сложность **O(n^2)**;
- Сортировка **перемешиванием** (Шейкерная, Cocktail sort, bidirectional bubble sort) — сложность **O(n^2)**;
- **Гномья** сортировка — сложность алгоритма — **O(n^2)**;
- Сортировка **вставками** (Insertion sort) — сложность **O(n^2)**;
- **Блочная** сортировка (Корзинная сортировка, Bucket sort) — сложность алгоритма: **O(n);** требуется **O(k)** дополнительной памяти.
- Сортировка **подсчётом** (Counting sort) — Сложность алгоритма: **O(n+k)**; требуется **O(n+k)** дополнительной памяти
- Сортировка **слиянием** (Merge sort) — Сложность алгоритма: **O(n log n);** требуется **O(n)** дополнительной памяти;
- Сортировка с помощью **двоичного дерева** (англ. Tree sort) — Сложность алгоритма: **O(n log n);** требуется **O(n)** дополнительной памяти

Теория $\mathcal{NP}$ алгоритмов выработала и ряд прагматических рекомендаций для решения прикладных задач.

В тех случаях, когда интересующая разработчика практических алгоритмов задача оказывается $\mathcal{NP}$ полной, имеет смысл попробовать построить **эффективный алгоритм** для какой-либо ее модификации или частного случая, приемлемых с практической точки зрения.

Когда не удается найти и такую модификацию, имеет смысл попробовать построить для задачи **приближенный эффективный алгоритм**, который гарантирует нахождение решения.

Парадоксы Зенона – Ахиллес и черепаха – причаливание, стыковка, посадка.

Математическое моделирование (общие слова)

Для разработки программы, решающей задачу, необходимо пройти несколько весьма важных этапов.

1. **Содержательная задача**
2. **Математическая модель**
3. **Задача на математической модели**
4. **Алгоритм решения задачи**
5. **Программа**
6. **Программный продукт**

Роль формального описания на каждом этапе (даже на этапе формулирования содержательной задачи) весьма велика.

Причина заключается в том, что разработка программ в настоящее время ведется не программистом-одиночкой, а целым коллективом.

Если до 90-х годов акцент в обучении программированию падал на изучение инструментов (языки программирования, операционные системы, базы данных), то сейчас все больше и больше обращается внимание на первые четыре этапа.

Основной отличительной особенностью человека является способность ставить и решать задачи. **Что же такое задача?**

Задача предполагает необходимость сознательного поиска средства для достижения ясно видимой, но непосредственно недоступной цели. Решение есть нахождение этого средства.

Дьердь Пойа (1887—1985) «Математическое открытие»

Процесс решения Пойа представлял как путь, ведущий к цели, в пространстве, где находятся препятствия.

Математика претендует на построение в некотором смысле универсального механизма решения любых задач вне зависимости от их физической природы.

Вся история развития математики говорит о том, что такие претензии весьма обоснованы ☺

Обычно математика «расправляется» с задачами по следующей схеме:

Классификация -> **Типизация** -> **Формализация**

Классификация задач по Евклиду.

Древнегреческий математик и философ Евклид (III в. до н. э.) выделил два основных типа задач:

1. **Построение фигур (проблема).**

2. **Доказательство свойств фигур (теорема).**

Любая конкретная задача из геометрии решается правильно подобранной комбинацией проблем (построений) и теорем.

К середине XIX века геометрия разделилась на множество плохо согласованных разделов: евклидова, сферическая, гиперболическая, проективная, аффинная, риманова, многомерная, комплексная и т. д.

«*Эрлангенская программа*» Клейна или алгебраизация геометрии позволила получить глубокие результаты, **для старых инструментов крайне затруднительные или вовсе недостижимые**.

Проблема «**Построить равнобедренный треугольник**», требует формулировки:

1) **свойств** равнобедренного треугольника (геометрических отношений;
2) **процедуры** в виде применения последовательности правил (алгоритма) построения чертежа геометрического объекта при помощи процессоров (циркуля и линейки).

Известные свойства класса равнобедренных треугольников, выделяющих их из множества всех треугольников, суть **математическая модель**

$$M = \{T, TP \subset T\},$$

где **T** – множество всех треугольников, а **TP** – множество равнобедренных треугольников. **Порождающая функция** имеет **входными данными** (аргументами) точки и прямые, заданные на плоскости, а **выходными данными** – треугольники. Правила «**вычисления**» функции формируются в виде «команд» процессору, состоящего только из циркуля и линейки.

Классификация задач по Декарту.

Французский математик и философ Рене Декарт (1596—1650) в своей работе «Правила для руководства ума» предложил схему, которая, по его мнению, может быть применима к решению любых задач:

1. **Задача любого вида сводится к математической.**

2. **Любая математическая задача сводится к алгебраической.**

3. **Любая алгебраическая задача сводится к решению единственного алгебраического уравнения.**

Создание аналитической геометрии позволило перевести исследование геометрических свойств кривых и тел на алгебраический язык, то есть анализировать уравнение кривой в некоторой **системе координат**.

Классификация задач по Пойа.

1. **Задачи на нахождение** (имеется в виду не выбор элемента, а его вычисление), то есть нахождение объекта (неизвестного), удовлетворяющего условиям задачи, которые связывают неизвестные с данными.

2. **Задачи на доказательство.** Эти задачи связаны с проверкой (вычислением) некоторого условия и принятием соответствующего заключения.

Классификация задач по Клини.

два класса основных задач, требующих для своего разрешения **построения алгоритма**:

1. **Задача на перечисление** элементов некоторого множества (рекурсивно-перечислимое множество).

2. **Задачи на распознавание** принадлежности элемента некоторому множеству (рекурсивное множество).

Классификация задач в работах по искусственному интеллекту.

Один тип задач — задача поиска решения, которая состоит всегда из последовательного решения двух подзадач:

1. **порождение или задание элементов пространства поиска**

2. **идентификация (узнавание) элемента, обладающего нужными свойствами.**

Задача поиска тогда будет состоять из последовательности шагов: на каждом шаге порождается элемент некоторого множества (пространства), а затем определяется, соответствует ли он некоторым (целевым) свойствам.

Примеры – стол, стул - на них сидят, на них едят ……..

Практическая (машинная) информатика должна уметь решать такие типовые задачи:

1. задачу на порождение (вычисление) некоторого объекта;

2. задачу на распознавание (доказательство) некоторых свойств объекта;

3. задачу поиска объекта (решения) в пространстве решений, состоящую из задач порождения и распознавания.

Задачи имеющие решения за конечное время с помощью доступных в настоящее время средств и называются **содержательными.**

Примеры – хочу луну и т.д.

Пример.

1. **Содержательная задача** – расчёт токов и напряжений в электрической цепи.
2. **Математическая модель** – отношения, заданы **законами Ома и Кирхгофа** в виде системы **линейных уравнений**, определяющих электрическую сеть.
3. **Задача порождения** суть нахождение **решения системы линейных уравнений** в виде значений токов и напряжений в элементах сети.
4. **Порождающая (вычисляющ**ая) система функций, например, **алгоритм Гаусса**.
5. **Программа** – стандартная программа, например, из пакета «MatLab».

Модель

Слово «модель» произошло от латинского слова «modulus», означает «мера», «образец». Слово модель всегда требует дополнительного разъяснения—модель чего (или кого), то есть модель какого явления, объекта, процесса мы имеем в виду.

Модель — это образ, схема реальной действительности .

Модель – это такой материальный или **мысленно представляемый объект**, который в процессе исследования **замещает объект-оригинал** так, что его непосредственное изучение дает новые знания об объекте-оригинале.

Под моделированием понимается процесс построения, изучения и применения моделей. Оно тесно связано с такими категориями, как абстракция, аналогия, гипотеза и др. Процесс моделирования обязательно включает и построение абстракций, и умозаключения по аналогии, и конструирование научных гипотез.

Главная особенность моделирования в том, что это **метод опосредованного познания** с помощью **объектов-заместителей**.

Модель выступает как своеобразный **инструмент познания**, который исследователь **ставит между собой и объектом**, и с помощью которого изучает интересующий его объект.

Основное требование, предъявляемое к моделям – это их адекватность реальным процессам или объектам, которые замещает модель.

Ошибки - рыбак, который ловил рыбу только одной сетью, решил, разглядывая свои уловы, что наименьшие среди пойманных рыб — это самые маленькие рыбы в море; он допустил грубую ошибку, не учитывая важную особенность ситуации — определённый размер ячеек сети **- (пример про уборщицу).**

Основным подтверждением адекватности принятой модели является её согласие с известными из **эксперимента** свойствами моделируемого объекта. При этом, чем больше окажется таких независимых подтверждений, тем большее доверие приобретает модель.

Модель может быть похожей копией объекта, выполненной из другого материала, в другом масштабе, с отсутствием ряда деталей. Например, это игрушечный кораблик, домик из кубиков, деревянная модель самолета в натуральную величину, используемая в авиаконструировании и др.

Модели такого рода называют натурными.

Модель может, однако, отображать реальность более абстрактно – словесным описанием в свободной форме, описанием, формализованным по каким-то правилам, математическими соотношениями, чертежами, программами и т.п.

Модели такого рода называют абстрактными.

Классификация абстрактных моделей:

1. **Вербальные (текстовые) модели**. Эти модели используют последовательности предложений на формализованных диалектах естественного языка для описания той или иной области действительности (примерами такого рода моделей являются милицейский протокол, правила дорожного движения).

2. **Математические модели** – очень широкий класс знаковых моделей (основанных на формальных языках над конечными алфавитами), использующих те или иные математические методы. Например, математическая модель звезды будет представлять собой сложную систему уравнений, описывающих физические процессы, происходящие в недрах звезды.

3. **Информационные модели** – класс знаковых моделей, описывающих информационные процессы (получение, передачу, обработку, хранение и использование информации) в системах самой разнообразной природы. Примерами таких моделей могут служить **OSI** – семиуровневая модель взаимодействия открытых систем в компьютерных сетях, или **машина Тьюринга** – универсальная алгоритмическая модель.

Граница между вербальными, математическими и информационными моделями может быть проведена весьма условно.

· **концептуальное моделирование**, при котором совокупность уже известных фактов или представлений относительно исследуемого объекта или системы истолковывается с помощью некоторых специальных знаков, символов, операций над ними или помощью естественного или искусственного языков;

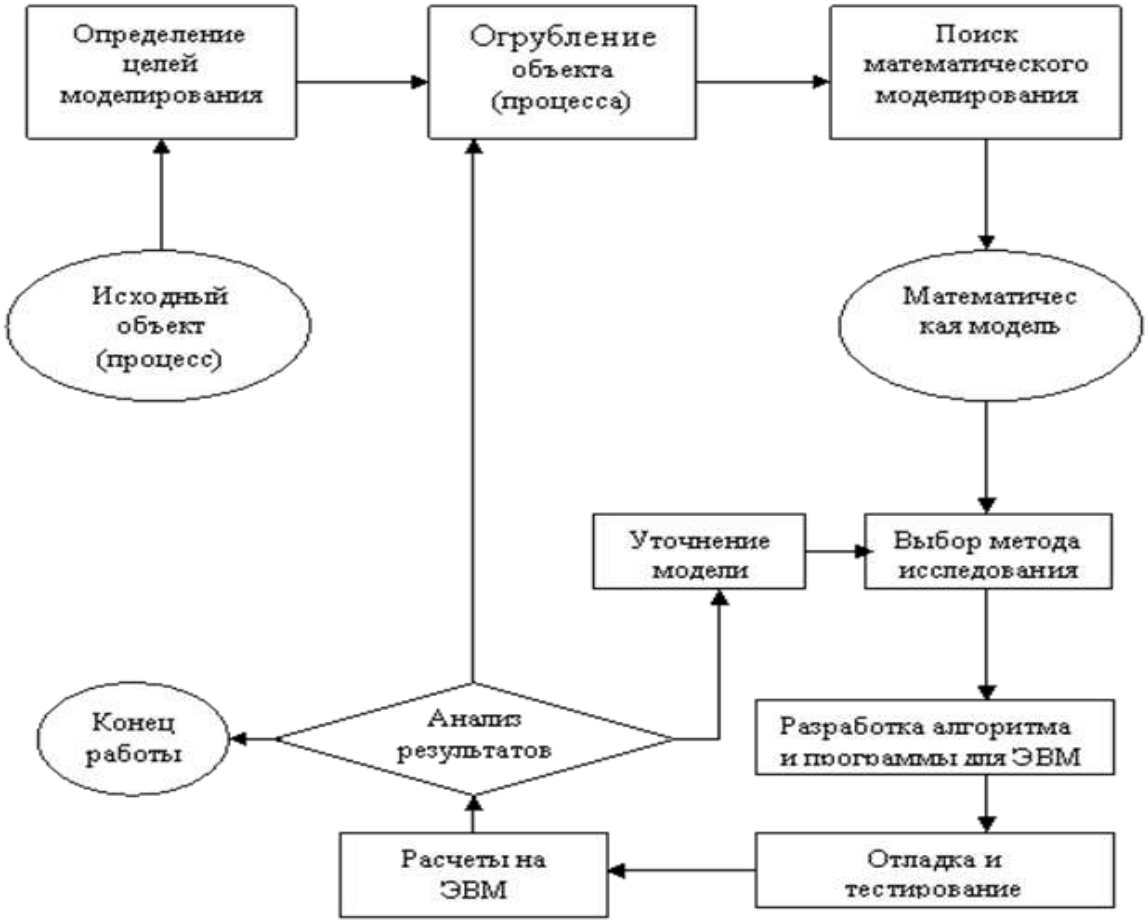
· **физическое (натурное) моделирование**, при котором модель и моделируемый объект представляют собой реальные объекты или процессы единой или различной физической природы, между процессами в объекте-оригинале и в модели выполняются некоторые соотношения подобия, вытекающие из схожести физических явлений (**числа Рейнольдса**);

· **структурно-функциональное моделирование**, при котором моделями являются схемы (блок-схемы), графики, чертежи, диаграммы, таблицы, рисунки, дополненные специальными правилами их объединения и преобразования;

· **математическое (логико-математическое) моделирование**, при котором моделирование, включая построение модели, осуществляется средствами математики и логики;

· **имитационное (компьютерное) моделирование**, при котором логико-математическая модель исследуемого объекта представляет собой **алгоритм функционирования объекта**, реализованный в виде программного комплекса для компьютера.

ЭТАПЫ И ЦЕЛИ КОМПЬЮТЕРНОГО МАТЕМАТИЧЕСКОГО МОДЕЛИРОВАНИЯ



Модель адекватна реальному процессу, если некоторые характеристики процесса, полученные на компьютере, совпадают с экспериментальными с заданной степенью точности.

Когда говорят, что некоторый объект ***M*** является **моделью** объекта ***N***, то всегда предполагают, что каждому выделенному элементу из ***N*** однозначно соответствует элемент из ***M***, каждой выделенной связи между элементами (во времени и пространстве) из ***N*** однозначно соответствует некоторая связь в модели ***M***.

Мы, как правило, будем рассматривать специфические модели, элементами и связями в которых выступают **математические объекты**.

Определение:

Пусть ***A*** - некоторое множество элементов,

P - множество отношений между элементами из ***A***,

F - множество операций над элементами из ***A***.

Математической моделью ***M*** называется тройка ***M=<A,P,F>.***

Множества

Г. Кантор - «**множество - единое имя для совокупности всех объектов, обладающих данным свойством**» или **«множество есть многое, мыслимое как единое»**

В 1903 году английский математик Бертран Рассел предложил антиномию в рамках языка классической («наивной») теории множеств Георга Кантора:

Пусть K — множество всех множеств, которые не содержат сами себя в качестве своего подмножества.

Ответ на вопрос «содержит ли ***K*** само себя в качестве подмножества?» не может быть дан в принципе.

*Если ответом является «да», то, по определению, такое множество не должно быть элементом **K**.*

Если же «нет», то, опять же по определению, оно должно быть элементом самого себя.

(Работы Лузина по теории пустых множеств)

Данная антиномия (более известная под названием «парадокс Рассела») поколебала основы математики и формальной логики, что вынудило ведущих математиков того времени начать поиск методов её разрешения.

Было предложено несколько направлений, начиная от банального **отказа от теоретико-множественного подхода в математике** и ограничения в использовании кванторов (интуиционизм, основоположником которого был голландский математик Лёйтзен Брауэр), до попыток **аксиоматической формализации теории множеств** (аксиоматика Цермело — Френкеля, аксиоматика Неймана — Бернайса — Гёделя и некоторые другие).

Позже австрийский философ Курт Гёдель показал, что **для достаточно сложных формальных систем всегда найдутся формулы, которые невозможно вывести (доказать) в рамках данной формальной системы —** первая теорема Гёделя о неполноте.

Данная теорема позволила ограничить поиски формальных систем, дав математикам и философам понимание того, что в сложных системах всегда будут появляться антиномии, подобные той, что предложил Б. Рассел.

Лучшее пояснение природы множества все же принадлежит основателю теории множеств Георгу Кантору:

*Под множеством мы понимаем любое соединение **A** в одно целое определённых вполне различаемых объектов **a**, которые существуют в нашем представлении или мыслях, которые называются элементами **A**.*

Конструктивное понятие множества в информатике

Множество суть совокупность (собрание, группа) элементов, обладающих общим свойством (природой, семантикой).

Теория типов описывает области определений и области значений функций — такие множества элементов, которые могут быть значениями входных параметров и возвращаемыми результатами функций.

Общее понимание теории типов в рамках информатики заключается в том, что обрабатываемые данные имеют тот или иной тип, то есть принадлежат определённому множеству возможных значений

Множества в информатике требуют уточнения **конструктивного** характера.

1. **Порождающий механизм для каждого элемента множества.**

2. **Каждый элемент множества должен отличаться от другого.**

3. **Интерпретация множеств** суть приписывание некоторого свойства (набора свойств) именно той совокупности элементов, которые объединены в множество.

Кроме этого в информатике множество (класс) и его элементы относятся к различным логическим типам, тип множества выше типа его элементов – что позволяет, например, сразу разрешить парадокс Рассела.

Понятие множеств и типов в информатике основано на математическом понятии, но не совсем тождественно ему

Два способа порождения множеств:

а) для конечных множеств – **перечисление** элементов;

б) для бесконечных множеств – **алгоритм или правила порождения**

Обычно для описания отличных друг от друга элементов применяется способ кодирования, такой, что код каждого элемента уникален.

Пример : Множество натуральных чисел – ***N* = {1, 2, 3, …, 437, …, 1521,}**.

Каждый элемент множества представляет собой код, построенный из алфавита цифр ***Z* = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.**

Известны способы кодирования двоичных и целых чисел (с плавающей запятой, с фиксированной запятой, обратных и дополнительных кодов).

Множество натуральных чисел, как указывал французский математик Пеано, «имеет счетную природу».

Эту природу можно увидеть, если ввести унарный код для описания натуральных чисел ***N* = {I, II, III, IIII, …, IIIIIIIIIIIIIII, …}.**
Каждое натуральное число кодируется определённым количеством палочек.

В этом случае порождающее правило задаётся операцией приписывания «палочки» к предыдущей последовательности «палочек».

Интерпретация множества натуральных чисел в виде свойства «счетности» формально описывается через операцию «прибавления» единицы и распространяется на двоичные коды или кодирование римскими цифрами.

Интерпретация множества четных чисел добавляет к основным свойствам свойство деления элементов множества на число «два».

Пример нетривиальной интерпретации:

В множестве групп первого курса выделяется множество групп ПМФ, поступивших в 2009 г. = **{111, 112, 113, 114,115,116}**.

Элемент множества – натуральное число, которое выступает только в виде кода, но не обладает всеми свойствами натуральных чисел (**эти коды бессмысленно складывать или умножать**).

В современных языках программирования (особенно в объектно-ориентированных и функциональных языках) механизм кодирования объектов, составляющих множества, и операций над объектами определяет сущность языка.

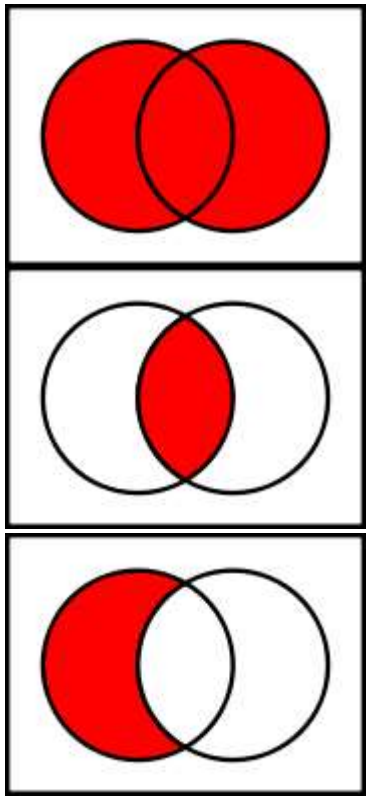
Операции над множествами и их свойства

1. Объединение $A \cup B = \{x \mid x \in A \vee x \in B\}$

2. Пересечение $A \cap B = \{x \mid x \in A \wedge x \in B\}$

3. Разность $A \setminus B = A \cap \overline{B} = \{x \mid x \in A \wedge x \notin B\}$

4. Дополнение $\overline{A} = \{x \mid x \notin A\}$



все элементы берутся из некоторого «универсального» множества, или универсума, обозначаемого **U** или **1**.

На совокупности этих операций определяется **булева алгебра**, которая позволяет производить эквивалентные преобразования формул, описывающие множества, сконструированные из исходных множеств.

Сделаем одно очень важное замечание об интерпретации (свойствах) множеств сконструированных из исходных (базовых).

! Множество, сконструированное из базовых и заданное формулой, в общем случае не наследует свойства исходных базовых множеств!

Специальные множества

* **Пустое множество** — множество, не содержащее ни одного элемента.
* **Универсальное множество** — множество, содержащее все мыслимые объекты.
* **Упорядоченное множество** — множество, на котором задано отношение порядка.

Булевой алгеброй называется непустое множество **M** с двумя бинарными операциями

$\cap$ (аналог конъюнкции),

$\cup$ (аналог дизъюнкции),

унарной операцией $\neg$ (аналог отрицания)

и двумя выделенными элементами:

0 (пустое множество или Ложь)

1 (универсум или Истина)

такими, что для всех **A**, **B** и **C** из множества верны аксиомы:

1°. Ассоциативность:

$$(A \cup B) \cup C = A \cup (B \cup C),$$

$$(A \cap B) \cap C = A \cap (B \cap C).$$

2°. Коммутативность:

$$A \cup B = B \cup A,$$

$$A \cap B = B \cap A.$$

3°. Дистрибутивность:

$$A \cap (B \cup C) = (A \cap B) \cup (A \cap C),$$

$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$

4°. Для пустого множества и универсума:

$$\varnothing \cup A = A,\ \varnothing \cap A = \varnothing,\ \varnothing \setminus A = \varnothing,\ A \setminus \varnothing = A,$$

$$\mathbf{1} \cup A = \mathbf{1},\ \mathbf{1} \cap A = A,\ \mathbf{1} \setminus A = \neg A,\ A \setminus \mathbf{1} = \varnothing.$$

5°. Правила де Моргана.

$$A \cup B = \neg(\neg A \cap \neg B),$$

$$A \cap B = \neg(\neg A \cup \neg B).$$

Алгебраический тип данных неформально можно определить как множество значений, представляющих собой некоторые контейнеры, внутри которых могут находиться значения каких-либо иных типов.

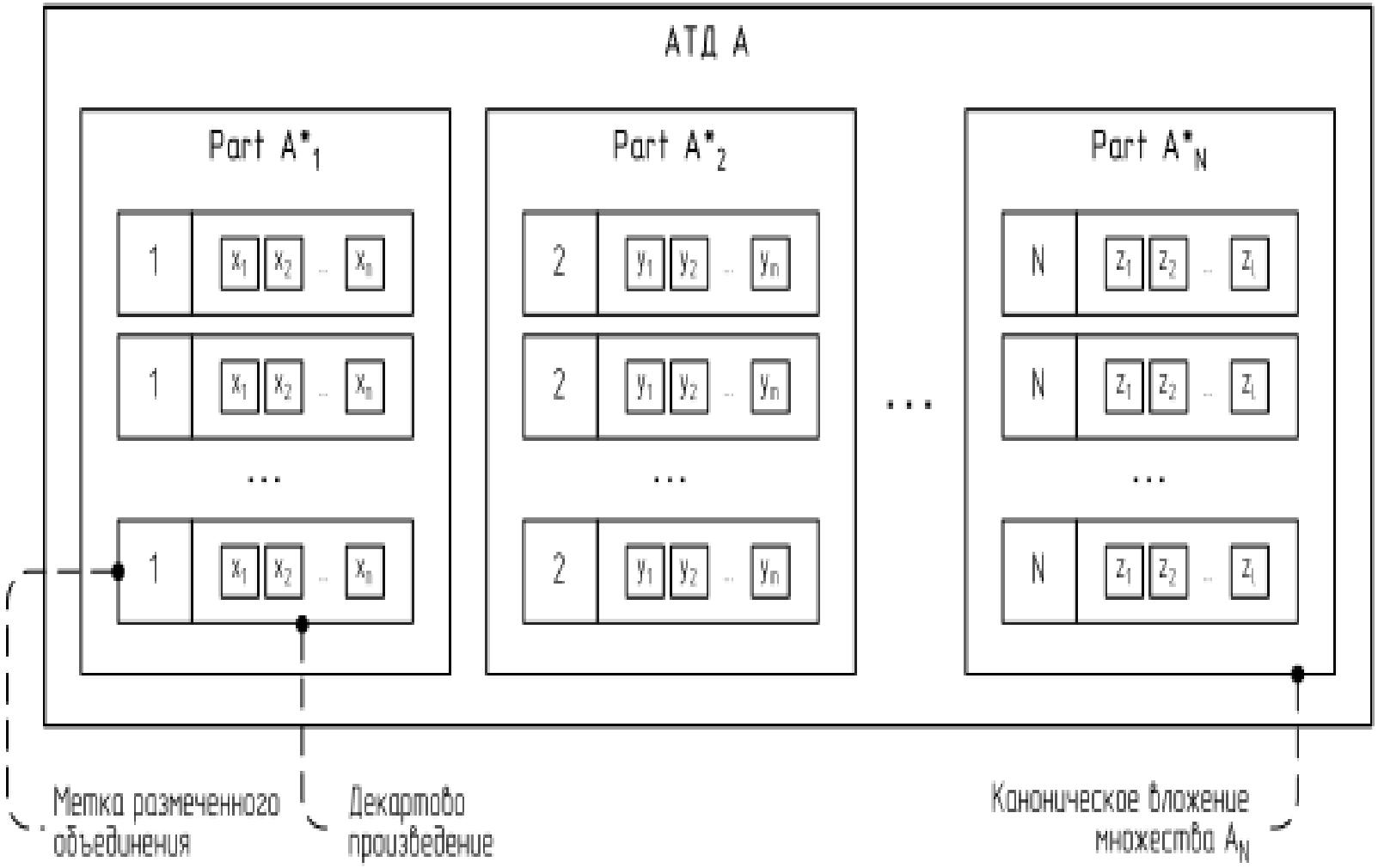
Множество таких контейнеров и составляет сам тип данных, множество его значений.

Алгебраический тип данных — **размеченное** объединение декартовых произведений множеств или, другими словами, размеченная сумма прямых произведений множеств.

$$\coprod_{i \in I} A_i = \bigcup_{i \in I} \{(x, i) \mid x \in A_i\}.$$

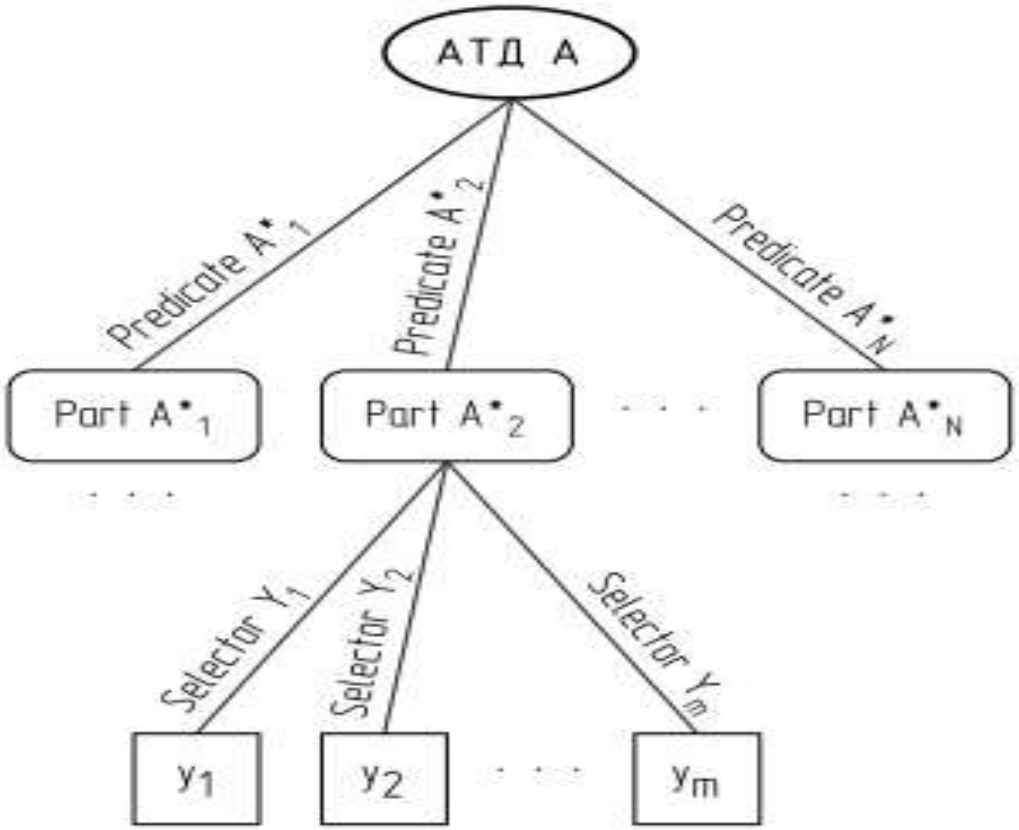
Здесь ***(x, i)*** — упорядоченная пара, в которой элементу ***x*** приписан индекс множества, из которого элемент попал в размеченное объединение.

Теоретико-множественное описание АДТ



Древовидное представление (нотация Ч. Э. Хоара)

синтаксически-ориентированное конструирование АТД



Чтобы найти ….. после третьего поворота налево у дуба сверни направо …….

Число три это два раза после нуля……

Пример:

Надо определить 3 типа фигур прямоугольник, круг или треугольник.

Прямоугольник определяется левым верхним и правым нижним углом:

```
struct Rectangle { Point topLeft, bottomRight; };
```

Круг — центром и радиусом:

```
struct Circle {Point center; double radius;};
```

Треугольник — тремя точками.

```
struct Triangle {Point a,b,c;};
```

Никакого **«общего интерфейса»**, как принято иллюстрировать в учебниках по ООП, у геометрических фигур нет и быть не может: геометрическая фигура — это всего лишь набор разных координат и длин.

Но как же описать тип данных, совмещающий в себе указание формы фигуры и параметров, необходимых для задания данной конкретной формы?

Первый способ, приходящий на ум программисту, имеющему опыт разработки на C, Java и т. п. таков: воспользоваться **составной структурой данных**, хранящей и форму, и параметры - т.е. помещение разных параметров в поля одной и той же структуры или класса.

Но при таком способе нельзя учесть, что для формы CIRCLE осмыслен только набор параметров типа Circle, и т. п.

Второй способ заключается в том, чтобы **отказаться от разделения на «форму» и «параметры»**, и описать тип данных явно «либо прямоугольник (задаваемый двумя точками), либо круг (задаваемый точкой и числом), либо треугольник (задаваемый тремя точками)» - т.е. если, то он описывается центром и радиусом…….

Вот код на Haskell:

```
data Shape = Rectangle {topLeft::Point, bottomRight::Point}
| Circle      {center::Point, radius::Double}
| Triangle    {a::Point, b::Point, c::Point}
```

Отношения,

функции,

операции

Отношения являются сущностью и основным объектом исследования математики.

Пусть задано множество $\mathcal{A}$ (конечное или бесконечное), введем понятие декартова произведения – $\mathcal{A} \times \mathcal{A}$, которое представляет собой множество всех пар

$\mathcal{D}^2=\{<a_i, a_j>\}$, где $(i, j) = 1, 2, 3, \ldots, (a_i, a_j) \in \mathcal{A}$, и допускается $i = j$.

Говорят, что множество $\mathcal{D}^2$ задает декартово пространство, элементами которого являются все возможные пары.

Любое подмножество $\alpha \subseteq \mathcal{A} \times \mathcal{A} = \mathcal{D}^2$ называется **бинарным отношением**.

Заметим (и это важно), «α» также является множеством – со своим правилом порождения и интерпретацией.

Пример.

Пусть $\mathcal{A} = \{a, b, c, d, e\}$ – множество кодов городов, для определённости

a – Астрахань,

b – Волгоград,

c – Самара,

d – Дудинка,

e – Екатеринбург.

Множество пар

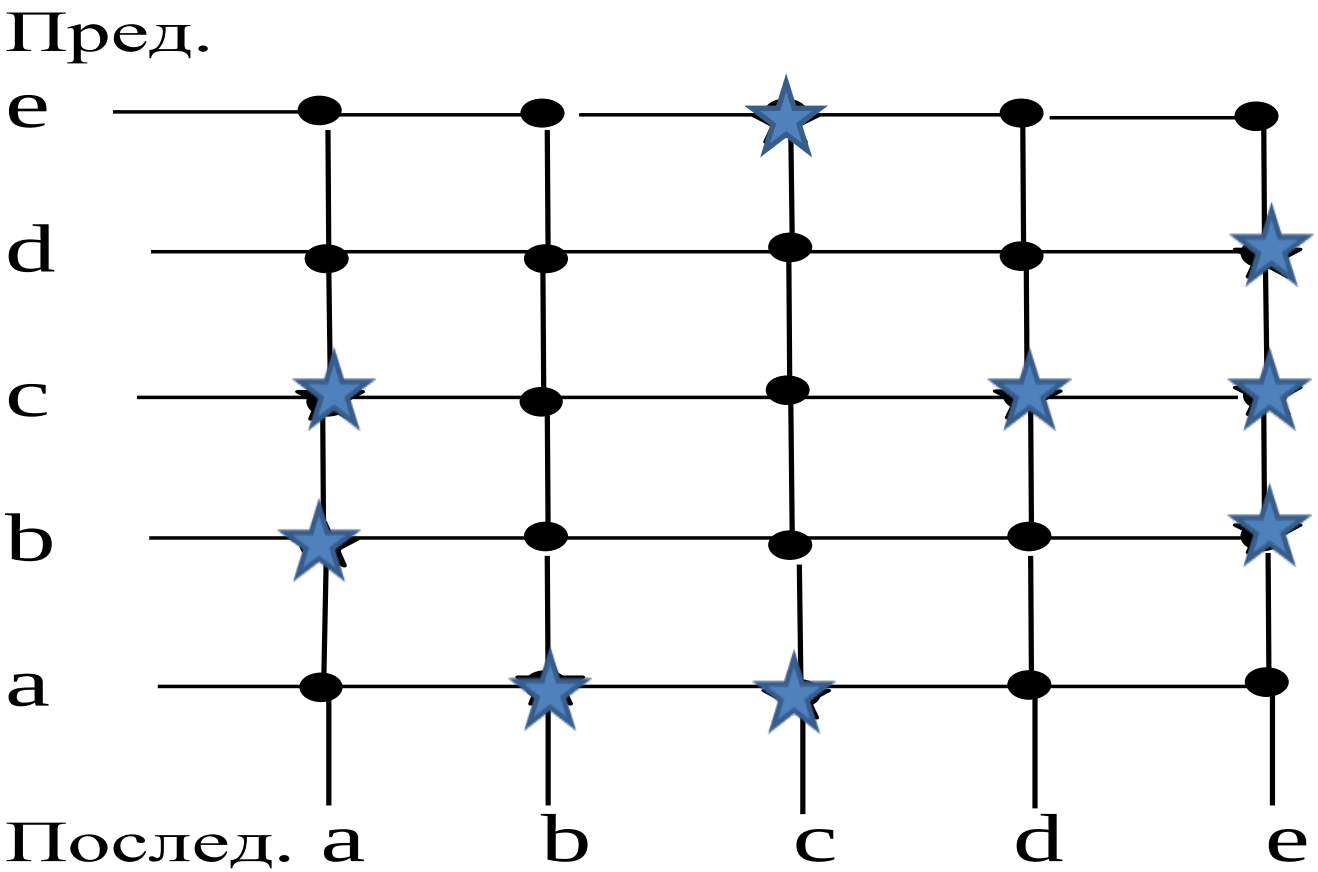
$\alpha = \{ac, ca, ав, ва, ce, ec, ве, cd, de\}$,

$\alpha \subset \mathcal{A} \times \mathcal{A}$

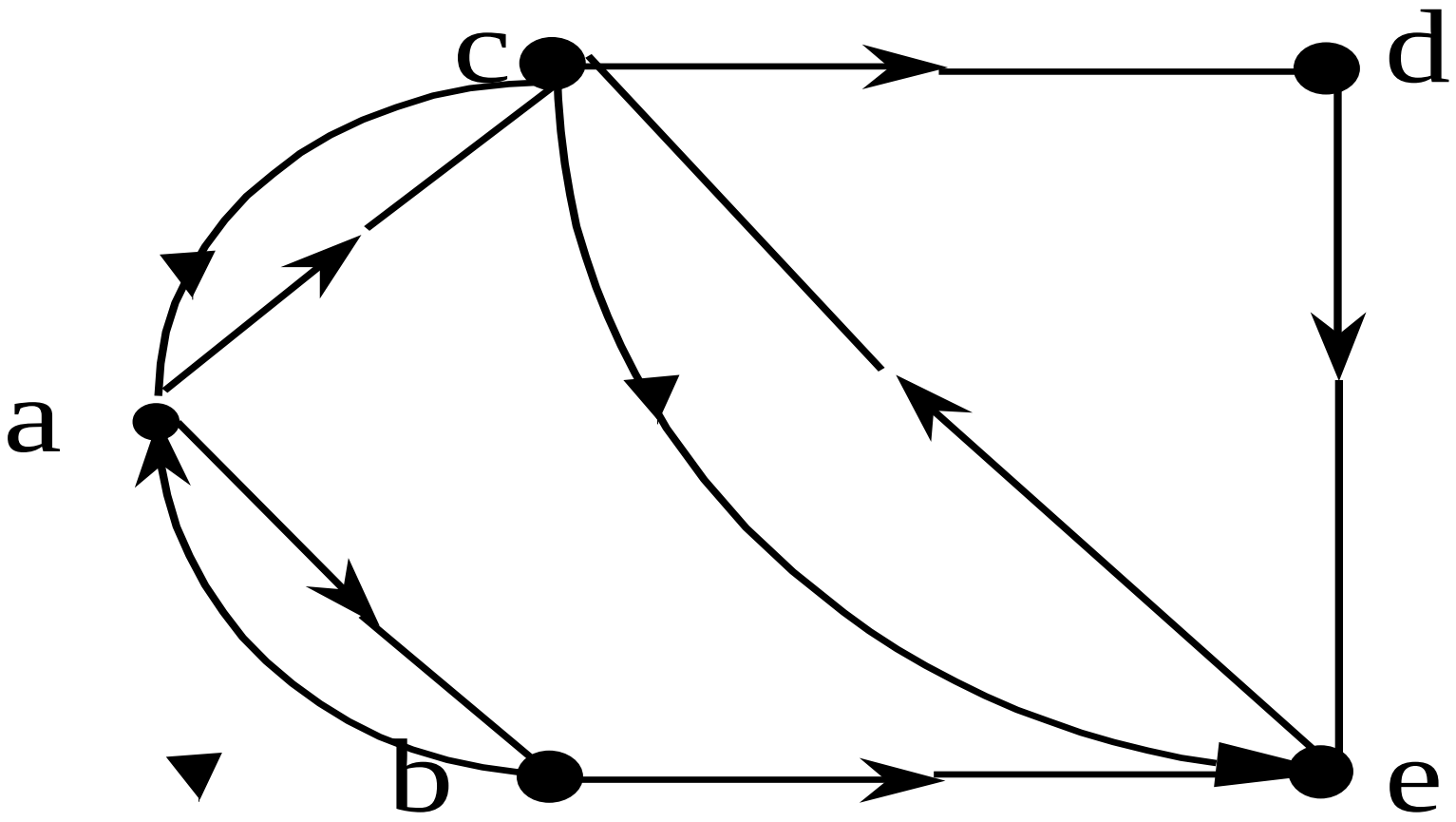
можно назвать **отношением** с интерпретацией «**город *i* непосредственно связан с городом *j* авиатрассой**».

На рисунке показано декартово пространство в виде двумерной решетки, узлы которой суть все возможные пары городов.

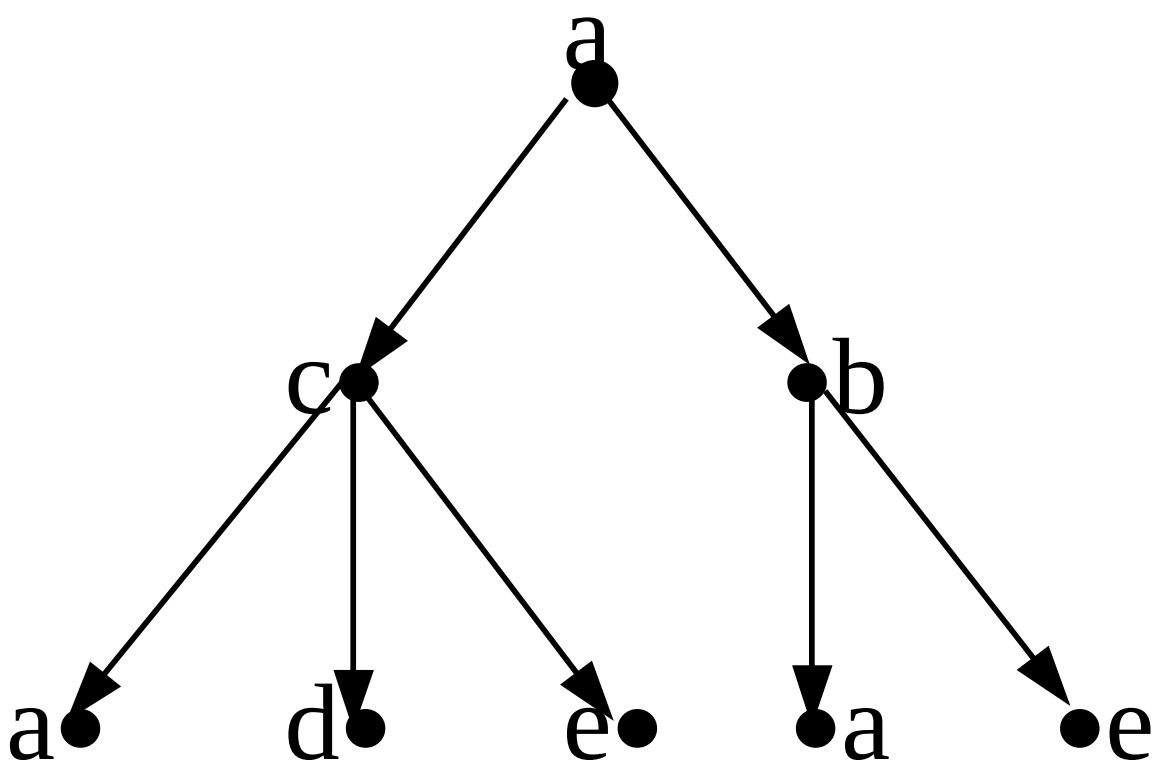
На рисунке пары были выделены «*».



Бинарное отношение может быть изображено графом, в котором вершины суть элементы алфавита «$\mathcal{A}$», а дуги соответствуют выделенным парам $\langle i, j \rangle$ отношения α, направление дуги указывается стрелкой от «*i*» к «*j*».



Дерево достижимости 2-го порядка из вершины «a»



Математическая модель авиационных трасс.

Формальные языки

Пусть $\mathcal{A}=\{a, b,$ и т.д. все малые латинские буквы$\}$ (**или буквы другого любого алфавита**).

Введем процедуру, порождающую все возможные слова в алфавите $\mathcal{A}$, сначала слова длины «1», далее – «2» и т.д. – длины «n».

Полученное множество слов обозначается как $\mathcal{A}^*$ («$*$» – знак операции итерации).

Понятно, что $\mathcal{A}^*$ есть бесконечное множество слов.

Процедура порождения слов описывается индуктивной схемой с единственной операцией, которая называется «конкатенация» (приписывание) и обозначается «●».

1. Вводится пустое слово – $\varnothing$ ($\mathcal{A}^0 = \varnothing$)

2. К пустому слову приписываются последовательно все буквы из алфавита $\mathcal{A}$, получается слово длины «1», которые составляют множество $\mathcal{A}' = \{a, b, \ldots\}$.

(n-1). Пусть порождено множество слов длины «$n-1$» – $\mathcal{A}^{n-1}$

n. Каждое слово $y \in \mathcal{A}^n$ получается из $x \in \mathcal{A}^{n-1}$ приписыванием букв из алфавита.

Формальным языком $\mathcal{L}$ называется любое подмножество $\mathcal{A}^*$, т.е. $\mathcal{L} \subseteq \mathcal{A}$, таким образом язык $\mathcal{L}$ является отношением на $\mathcal{A}^*$.

В математической логике и информатике **формальный язык** — это множество **каких-то конечных слов** над **конечным алфавитом**.

Понятие языка чаще всего используется в теории автоматов, теории вычислимости и теории алгоритмов. Научная теория, которая имеет дело с этими объектами, называется теорией формальных языков.

Пример Задан двух буквенный алфавит $\mathcal{A} = \{a, b\}$.

$\mathcal{A}^*$ – множество слов, состоящих из двух букв, приписанных в произвольном порядке.

$\mathcal{A}^2 = \{aa, bb, ab, ba\}$,

$\mathcal{A}^3 = \{aaa, aab, bba, bbb, aba, abb\ baa, bab\}$ и т.д.

а) Язык $\mathcal{L}_1 = \{aab, aabb\}$ – состоит из двух слов (конечный язык).

б) Язык $\mathcal{L}_2 = a^n b^m$; $n = 1, 2, \ldots, m = 1,2,\ldots.$ $bab \notin \mathcal{L}_2$.

в) Язык $\mathcal{L}_3 = a^n b^n$; $n = 1, 2, \ldots$ содержит все слова, которые имеют одинаковое число букв a и b :

$aaabbb \in \mathcal{L}_3$, $aaaabbbb \in \mathcal{L}_3$, но $abab \notin \mathcal{L}_3$, $bbaa \notin \mathcal{L}_3$.

г) Язык $\mathcal{L}_4 = x \cdot x^{-1}$ – так называемый «зеркальный» язык, где $x \in \mathcal{A}^*$.

Функции

Пусть ***A*** и ***B*** – множества, (вообще говоря, ***A*** и ***B*** могут быть и отношения, т.к. отношения тоже множество), а x_i, y_j элементы этих множеств

$$x_i \in A, \quad y_j \in B, \quad i = 1, 2, \ldots;$$

Бинарной функцией называется такое отношение

$$f \subseteq A \times B,$$

такое, что если существует пара

$$< x_i , y_i > \in f,$$

то

$$< x_i , y_j > \notin f, \quad i \neq j.$$

Арность функции — количество множеств, декартово произведение которых, является её областью определения.

По нашему определению для одного и того же ***x*** не могут существовать два и более значений ***y***. Это свойство называется **детерминированностью** **(поговорить о стохастические модели и мат. ожидание)**.

Говорят, что $x_i \in A$ – аргумент функции, $y_i \in B$ – значение функции.

Детерминированность означает, что в функциональном отношении для каждого значения аргумента x_i может быть только единственное значение функции y_i (**поговорить о ветвлениях в Maple**).

Понятно, что функциональное отношение может быть любой **арности**.

Например, функция $+(x, y) = z$ (арифметическое сложение на N) – **тренарная** функция.

Пример

Пусть A и B конечные алфавиты и есть два множества слов $L_1 \subset A^*$ и $L_2 \subset B^*$ - т.е. два формальных языка с разными алфавитами (кодами)

Отображение

$$T(L_1) = L_2;\ T \subseteq (L_1 \times L_2)$$

называется **транслятором** L_1 в L_2, когда каждому слову $x \in L_1$ ставится в соответствие слово $y \in L$ **детерминированно!!!!!!**.

Иерархия Хомского — классификация формальных языков и формальных грамматик, согласно которой они делятся на 4 типа по их условной сложности.

Предикаты

Предикат есть *характеристическая* функция, которая выделяет некоторое отношение.

Аргументами являются ***n***-ки, составляющие множество точек декартова пространства, а значениями «истина» (1), если ***n***-ка попадает в отношение «ложь» (0), если не попадает.

Пусть задано отношение $\alpha \subseteq \{X_1 \times X_2 \times .,.. \times X_n\}$

тогда соответствующий отношению α ***предикат*** определяется как функция

$$P(X_1, X_2, \ldots, X_n) = \begin{cases} \text{истина } (1), \text{если } \langle x_1, x_2, \ldots, x_n \rangle \in \alpha \\ \text{ложь } (0), \text{ если } \langle x_1, x_2, \ldots, x_n \rangle \notin \alpha \end{cases}$$

где .

$$x_1 \in X_1, x_2 \in X_2, \ldots, x_n \in X_n$$

Пример

Чётное число. Пусть любое натуральное число «***n***» представляется двоичным кодом, тогда предикат «четное число» $P_r(n)$ определяется признаком чётности.

$$P_r(n) = \begin{cases} и, \text{если } 0\text{-й разряд его кода равен "0"} \\ л, \text{если } 0\text{-й разряд его кода равен "1"} \end{cases}$$

Синтаксический анализатор в трансляторах реализует сложный ***предикат***, отвечающий на вопрос о принадлежности словарной конструкции, описывающей программу пользователя, к синтаксически правильным конструкциям языка программирования.

Высказывания с предикатом

Структура высказывания – «S есть P», где S называется субъектом, а P предикатом.

Субъект есть сущность, предмет, набор (множество) предметов.

Предикат P определяет свойство «принадлежать».

В математической записи – $S \in P(S)$.

Высказывание принимает значение **«истина»,** если значение **«истина»** принимает соответствующий предикат $P(S)$ со значением S .

Пример Высказывание: «4 есть чётное число» – истинно, т.к. при S=4, предикат $P(S)$ принимает значение **«истина»**

Предикативные высказывания в отличии от обычных высказываний содержат внутреннюю структуру (субъект и предикат).

Но если субъект определён и истинность/ложность предиката может быть вычислена, тогда предикативные высказывания ничем не отличаются от обычных и из них могут быть при помощи известных логических связок сконструированы сложные высказывания.

Обычно применяют следующие логические связки :

$\neg$ – **отрицание**,

$\vee$ – **дизъюнкция**,

$\&$ – **конъюнкция**,

$\rightarrow$ -- **импликация** с известной интерпретацией истинностными таблицами.

Пример.

Высказывание:

Студент Иванов спит на лекции, поэтому он не является отличником.

Элементарные высказывания:

S (субъект – Иванов),

P_1 (предикат – студент, т.е. имеет свойство «быть студентом»). Иванов принадлежит к множеству «студенты» и это истинно. .

P_2 (предикат – свойство «спать на лекции»).

P_3 (предикат – свойство «быть отличником»).

Сложное высказывание , где x = Иванов, читается так:

«Если Иванов есть студент и спит на лекции , то он не отличник ».

Для субъекта = Иванов, это высказывание истинно.

Заметим, что тождественная истинность высказывания, относится только к студентам, спящим на лекции.

Кванторы.

Пусть задан предикат , который означает, что ***x*** обладает свойством ***P***.

Рассмотрим утверждение: «Для всякого предмета ***x*** свойство выполнено».

Это утверждение (высказывание) будет истинным, если на самом деле все возможные предметы, объединённые в множество, обладают свойством ***P***.

Пример: $X = \{2, 4, 6, 8\}$ – все предметы из ***X*** обладают свойством чётности .

Это означает, что $P(2)\&P(4)\&P(6)\&P(8)$ – истинное утверждение.

В случае бесконечного множества ***X*** вводится специальный знак – квантор общности (всеобщности) – $\forall$.

Тогда наше утверждение будет иметь вид $\forall_x$.

Таким образом квантор всеобщности служит обобщением операции конъюнкции.

Операции

По определению, операция суть функция, где значения **аргументов** и значения **функции** берутся **из одного и того же множества**.

Примером могут служить операции арифметического сложения, умножения, деления для действительных чисел**.**

Для операций могут быть характерны следующие свойства:

* **Коммутативность** **x*y = y*x** (**матрицы**),
* **Антикоммутативность** **x*y = -y*x** (**операторы**),
* **Ассоциативность** **(x*y) *z = x*(y*z)**
* **Дистрибутивность** **x*(y&z) = x*y & x*z** (**слева и справа**)
* **Аддитивность** **f(x+y) = f(x)+f(y)**
* **Идемпотентность** **-** повторное действие над объектом не изменяет его

В программировании **ассоциативностью** операторов называют последовательность их выполнения (или направление вычисления), реализуемое, когда операторы имеют одинаковый приоритет и отсутствует явное (с помощью скобок) указание на очерёдность их выполнения.

При этом различается левая ассоциативность, при которой вычисление выражения происходит слева-направо, и правая ассоциативность — справа-налево.

Соответствующие операторы называют левоассоциативными и правоассоциативными.

Идемпотентная операция в информатике — действие, многократное повторение которого не приводит к изменениям иным, нежели при однократном.

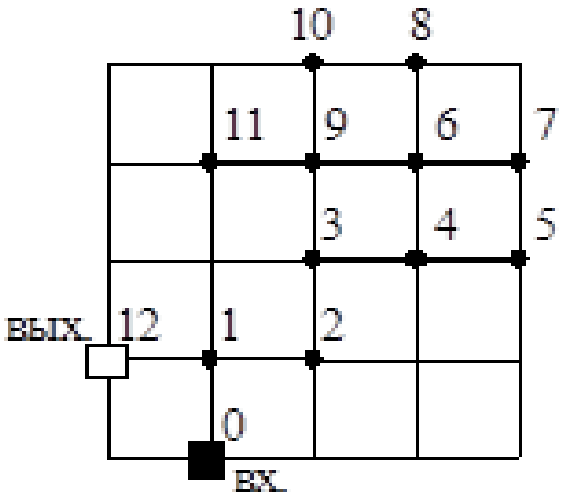
Примером такой операции могут служить **GET**-запросы в протоколе **HTTP**. По спецификации сервер должен возвращать одни и те же ответы на идентичные запросы (при условии что ресурс не изменился между ними по иным причинам). Такая особенность позволяет кэшировать ответы, снижая нагрузку на сеть.

Пример. **Лабиринтная задача.**

Блуждание Тезея по лабиринту по правилам и с нитью, данными ему Ариадной.

1) **Содержательная задача** – нахождение (порождение) пути в лабиринте от точки входа до точки выхода.

2) **Структура данных** – множество пар залов, соединённых переходами



***L* = {0-1, 1-2, 2-3, 3-4, 4-5, 3-9, 9-6, 9-10, 9-11, 6-8, 6-7}.**

Предполагается, что переходы проходимы в любую сторону, поэтому граф переходов симметричный.

Функция нахождения пути состоит из комбинации трёх функций на каждом шаге (когда Тезей находится в одном из залов):

a. функции, порождающей все имеющиеся переходы из зала;

b. распознающей функции, которая распознает свободные переходы для дальнейшего движения;

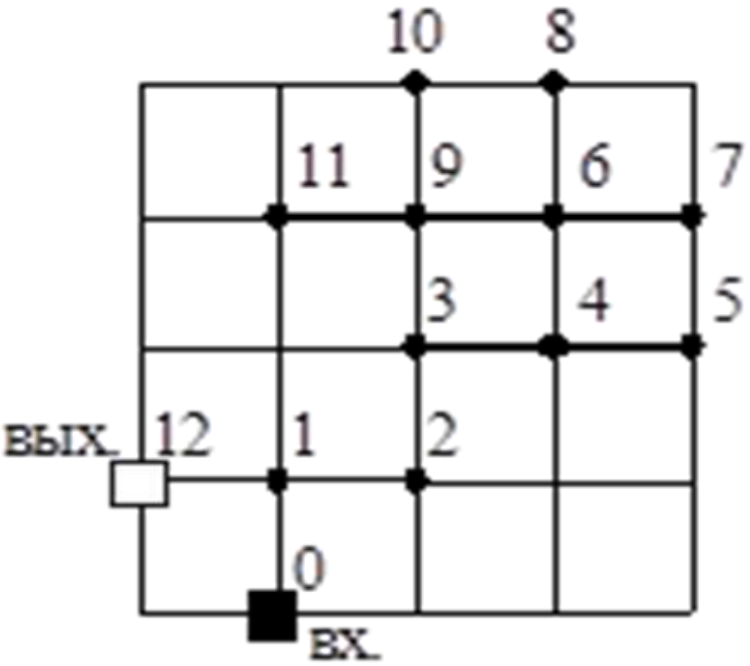
c.функции, порождающей метки пройдённого пути

Последовательность (траектория) прохождения залов Тезеем кодируется так называемым кодом Тезея (Т-кодом).

Для правила правой руки для выбора зала,

траектория блуждания будет

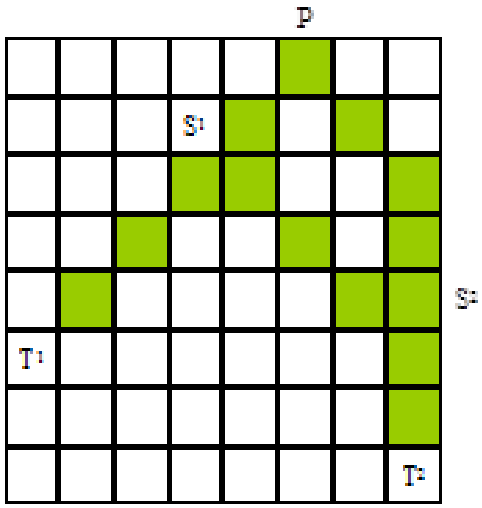
***T* = 0·1·2·3·4·5·4·6·7·6·8·6·9·10·9·11·9·3·2·1·12.**



Пример. **Распознавание букв русского алфавита.**

Буква проектируется на экран (ретину), состоящий из фотоэлементов

Необходимо распознать, какая буква отображается на экране.

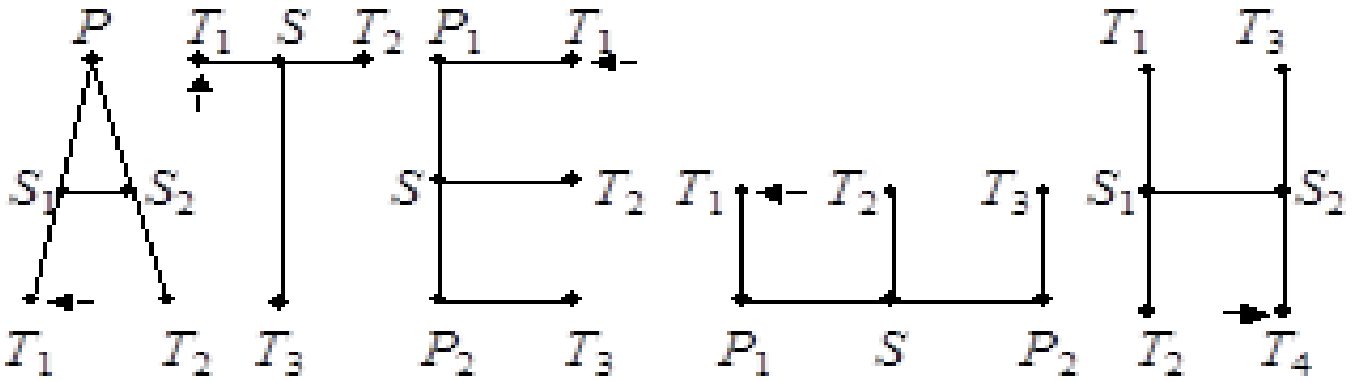


Структурное распознавание отличается от «обучающего» распознавания тем, что образ объекта определяется конечным набором **известных заранее признаков**.

Такими известными признаками для букв являются характерные точки: **T** – терминальные (оконечные), ***P*** – точки перелома, ***S*** – точки сопряжения.

Вместо того, чтобы распознавать 30 букв, нужно распознавать всего 3 образа – ***T, P, S*** и коды букв, которые состоят из тех же ***T, P, S***.

Если «запустить» Тезея в такой лабиринт, где вход и выход находятся в одной и той же терминальной точке, то каждой букве алфавита будет соответствовать свой **T-код**, который не зависит от расположения и размеров буквы на ретине фотоэлементов.



Буква А = {$T_1 \cdot S_1$, $T_2 \cdot S_2$, $S_1 \cdot P$, $P \cdot S_2$} суть **симметричное бинарное отношение связности** и может трактоваться как лабиринт с залами {T_1, T_2, S_1, S_2, P} и соответствующими переходами.

$T_A = T_1 \cdot S_1 \cdot S_2 \cdot T_2 \cdot S_2 \cdot P \cdot S_1 \cdot T_1$;

$T_T = T_1 \cdot S \cdot T_3 \cdot S \cdot T_2 \cdot S \cdot T_1$;

$T_E = T_1 \cdot P_1 \cdot S \cdot P_2 \cdot T_3 \cdot P_2 \cdot S \cdot P_1 \cdot T_1$;

$T_Ш = T_1 \cdot P_1 \cdot S \cdot P_2 \cdot T_3 \cdot P_2 \cdot S \cdot T_2 \cdot S \cdot P_1 \cdot T_1$;

$T_H = T_4 \cdot S_2 \cdot S_1 \cdot T_1 \cdot S_1 \cdot T_2 \cdot S_1 \cdot S_2 \cdot T_4$.

Итак, формально наша задача распадается на две подзадачи.

1) Задача порождения Т-кодов для букв алфавита (лабиринтная задача) и создание справочника Т-кодов букв – т.е. порождения формального языка.

Процедура её решения состоит из снятия изображения известной заранее буквы на ретине и превращения его в Т-код.

2) Задача распознавания буквы состоит из снятия Т-кода изображения и сравнения с Т-кодами справочника.

Это по сути решение задачи идентификации (распознавания эквивалентности) двух слов формального языка Т-кодов – т.е**. вычисление предиката**.

Заметим, что все эти задачи решаются на очень простых алгебраических типах данных – матричных и словарных.

Источники

1. А. Н. Колмогоров, Теория информации и теория алгоритмов. — М.: Наука, 1987. — 304 с.
2. Клини С. К. Введение в метаматематику. — М.: ИЛ, 1957.
3. Джордж Пойа. Математическое открытие. М., Наука, 1976 — 448 с
4. Фридланд А.Я. Информатика: процессы, системы, ресурсы. М.: БИНОМ. Лаборатория знаний, 2003.
5. Шеннон Р. Имитационное моделирование систем – искусство и наука. М.: Мир, 1990.
6. Вольфенгаген В. Э. Методы и средства вычислений с объектами. Аппликативные вычислительные системы. — М.: АО «Центр ЮрИнфоР»,2004. — 78С.
7. С. А. Абрамов Лекции о сложности алгоритмов, МЦНМО, 2009 г., 256 стр.
8. А. К. Гуц, Математическая логика и теория алгоритмов, Либроком, 2009 г., 120 стр.
9. Лекции В.М. Абрамова "Алгоритмы и алгоритмические языки”, МФТИ.
10. Лекции Л.Н. Столярова "Информатика и применение компьютеров в научных исследованиях”, МФТИ.