Базовые алгоритмические структуры. Данные, их типы, структуры и обработка

Алгоритм

- "Алгоритм" является базовым основополагающим понятием информатики (и математики), а алгоритмизация (программирование) – основным разделом курса информатики.
- Понятие алгоритма, как и понятие информации, точно определить невозможно. Поэтому встречаются самые разнообразные определения – от "наивно-интуитивных" ("алгоритм – это план решения задачи") до "строго формализованных" (нормальные алгоритмы Маркова).

Алгоритм

- *Алгоритм* – это **конечная** совокупность точных (формализованных) и полных команд исполнителю алгоритма (человек, ЭВМ), задающих порядок и содержание действий, которые он должен выполнить для нахождения решения любой задачи из рассматриваемого класса задач.

Базовые алгоритмические структуры

- Различают три *базовые* алгоритмические структуры: *следование, ветвление, повторение.*
- Структура ***следование*** состоит из двух команд с указанной очередностью их выполнения и имеет вид:

  <команда–предшественник>;<команда–преемник>.
- Структура типа ***ветвления*** в полной форме состоит из некоторого условия, проверяемого на истинность при выполнении структуры; команды, выполняемой при выполнении проверяемого условия, и команды, выполняемой при невыполнении условия. Структура имеет вид

если <условие>

тогда <команда, выполняемая при выполнении условия>

иначе <команда, выполняемая при невыполнении условия>

Базовые алгоритмические структуры

- Структура ***повторения*** (цикл) служит для компактной записи одного и того же набора команд, повторяемых для различных значений параметров команд.
- Телом цикла называется последовательность повторяемых команд, которая может быть и пустой.

Данные

- *Данные* – это некоторые сообщения, слова в некотором заданном алфавите.
- *Пример:*
  - число 123 – *данное*, представляющее собой слово в алфавите из десяти натуральных цифр;
  - число 12,34 – *данное*, представляющее собой слово в алфавите из десяти натуральных цифр и десятичной запятой;
  - текст "математика и информатика – нужные дисциплины", – *данное* в алфавите из символов русского языка и знаков препинания, включая пробел.
- Текущее (то есть рассматриваемое в данный момент времени) состояние *данных* называют текущим *значением данных* или просто *значением* .

- До разработки алгоритма (программы) необходимо выбрать оптимальную для реализации задачи *структуру данных*.
- На структуру данных влияет выбранный метод решения.
- *Тип* данных характеризует область определения значений данных.
- Типы данных задаются
  - простым перечислением *значений* типа, например как в *простых типах данных*,
  - объединением (структурированием) ранее определенных каких-то типов – *структурированные типы данных* .

Пример:

- Зададим простые типы данных *"специальность"*, *"студент"*, *"вуз"* следующим перечислением:

специальность = (филолог, историк, математик, медик);

студент = (Петров, Николаев, Семенов, Иванова, Петрова);

вуз = (СПбГУ, ГУАП, ИТМО).

Значением типа *"студент"* может быть *Петров.*

- Опишем *структурированный тип данных "специальность_студента":*

специальность_студента=(специальность, студент).

Значением типа *"специальность_студента"* может быть пара (*историк, Семенов*).

- Для обозначения *текущих* значений данных используются константы – числовые, текстовые, логические.

Пример:

- 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*, который называется ***индексом***.

Пример:

- Последовательность чисел 89, –65, 9, 0, –1.7 может образовывать одномерный вещественный *массив* размерности 5, например, с именем x вида: x[1] = 89, x[2] = –65, x[3] = 9, x[4] = 0, x[5] = –1.7.
- *Значение* порядкового номера элемента *массива* называется *индексом* элемента.
- Можно ссылаться на элемент x[4], элемент x[i], элемент x[4+j] *массива* x. При текущих *значениях* переменных i = 2 и j = 1 эти индексы определяют, соответственно, 4-й, 2-й и 5-й элементы *массива*.

- 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, но в отличие от вектора, *матрица* имеет описание двух индексов, разделяемых запятыми: первый определяет начальное и конечное *значение* номеров строк, а второй – столбцов.

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.

Восходящий метод,

- 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".

Структурированный алгоритм

- *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.

Модуль

- 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.
- Модуль - *вспомогательный алгоритм (подалгоритм).* 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. В программировании используются синонимы – процедура, подпрограмма, метод.

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, т.е. обеспечивающим проверку каждого отдельного типа наборов входных данных, особенно исключительных случаев.
- *Пример.* Для задачи решения квадратного уравнения

$$ax^2 + bx + c = 0$$

такими исключительными случаями, например, будут:

1) $a = b = c = 0$;

2) $a = 0$, b, c – отличны от нуля;

3) $D = b^2 - 4ac < 0$ и др.

Тестирование алгоритма

- Полную гарантию правильности алгоритма может дать описание работы и результатов алгоритма с помощью системы аксиом и правил вывода или *верификация* алгоритма.
- Для несложных алгоритмов грамотный подбор *тестов* и полное *тестирование* может дать полную картину работоспособности (неработоспособности).
- *Трассировка* – это метод пошаговой фиксации динамического состояния алгоритма на некотором *тесте*. Часто осуществляется с помощью трассировочных таблиц, в которых каждая строка соответствует определённому состоянию алгоритма, а столбец – определённому состоянию параметров алгоритма (входных, выходных и промежуточных). *Трассировка* облегчает отладку и понимание алгоритма.

Тестирование алгоритма

- Процесс поиска и исправления (явных или неявных) ошибок в алгоритме называется *отладкой* алгоритма.
- Некоторые (скрытые, труднообнаруживаемые) 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 алгорима (программы).

Алгоритм

Алгоритм и его свойства

Понятие алгоритма – одно из фундаментальных понятий информатики.

Алгоритмизация наряду с моделированием выступает в качестве общего метода информатики.

К реализации определенных алгоритмов сводятся процессы управления в различных системах, что делает понятие алгоритма близким и кибернетике.

Алгоритмы являются объектом систематического исследования пограничной между математикой и информатикой научной дисциплины, примыкающей к математической логике – **теории алгоритмов**.

История понятия «алгоритм»

Само слово «**алгоритм**» происходит от **algorithmi** - латинской формы написания имени великого математика IX века аль-Хорезми, который сформулировал правила выполнения арифметических действий.

Первоначально под алгоритмами и понимали только правила выполнения четырех арифметических действий над многозначными числами.

Определение алгоритма

1. Конечная последовательность действий (операций) и правил их выполнения или команд, предназначенных для решения определенной задачи или группы задач;

2. Предписание, определяющее ход вычислительного процесса, связанного с преобразованием данных от некоторого их исходного состояния к требуемому результату.

Формальные описания алгоритмов аналогичны представлениям основных частей программ, которые их реализуют, поэтому многое, что относят к описанию конкретных программ, применимо к алгоритму и наоборот.

Понятие исполнителя алгоритма

- Понятие **исполнителя** невозможно определить с помощью какой-либо формализации.

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.

Вся совокупность команд, которые данный исполнитель умеет выполнять, называется **системой команд исполнителя** (СКИ).

Исполнитель алгоритма

- Одно из принципиальных обстоятельств состоит в том, что исполнитель не вникает в смысл того, что он делает, но получает необходимый результат.
- В таком случае говорят, что исполнитель действует **формально,** т. е. отвлекается от содержания поставленной задачи и только строго выполняет некоторые правила, инструкции.
- Наличие алгоритма формализует процесс решения задачи, исключает рассуждение исполнителя. Использование алгоритма дает возможность решать задачу формально, механически исполняя команды алгоритма в указанной последовательности.
- Введение в рассмотрение понятия «исполнитель» позволяет определить алгоритм как понятное и точное предписание исполнителю совершить последовательность действий, направленных на достижение поставленной цели.

Графическое представление алгоритма

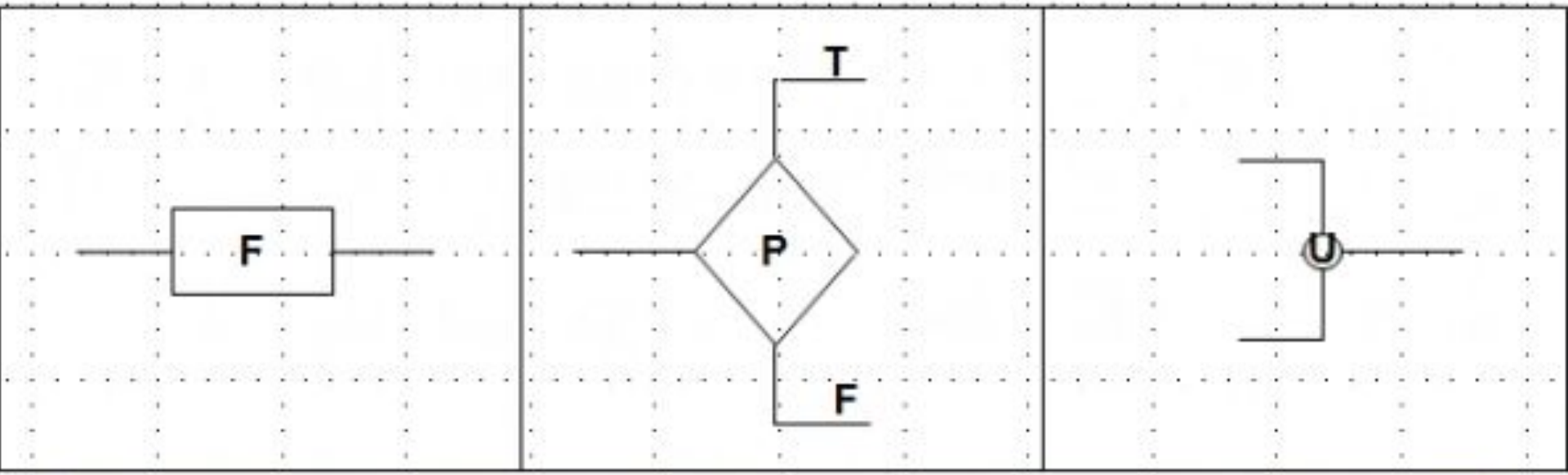
- Алгоритм, составленный для некоторого исполнителя, можно представить различными способами:
  - с помощью графического описания;
  - с помощью словесного описания;
  - в виде таблицы;
  - в виде последовательности формул.

- Графическое описание алгоритма, называется **блок-схемой**.
- **Блок-схема** – это ориентированный граф, указывающий порядок исполнения команд алгоритма.

Блок-схема

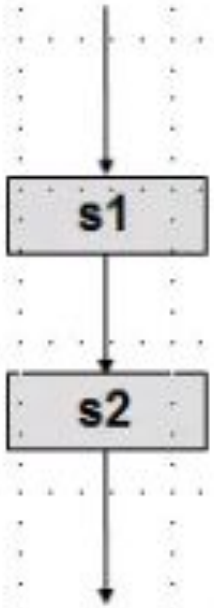
Вершины графа могут быть одного из трех типов:

- Функциональная (имеющая один вход и один выход);
- Предикатная (имеющая один вход и два выхода);
- Объединяющая (обеспечивающая передачу управления от одного из двух входов к выходу).

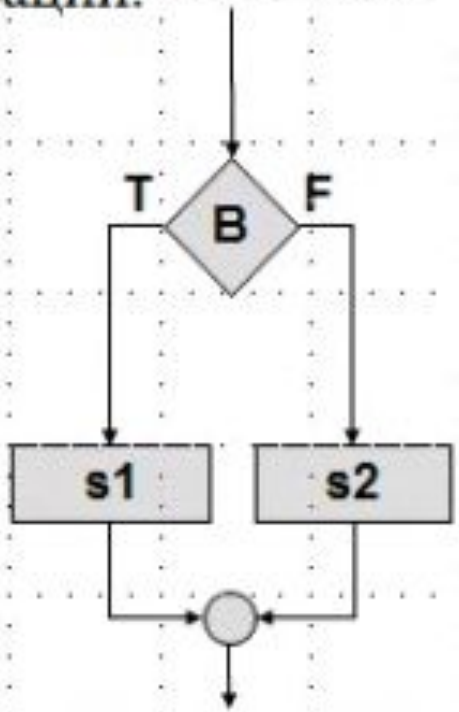


Блок-схема

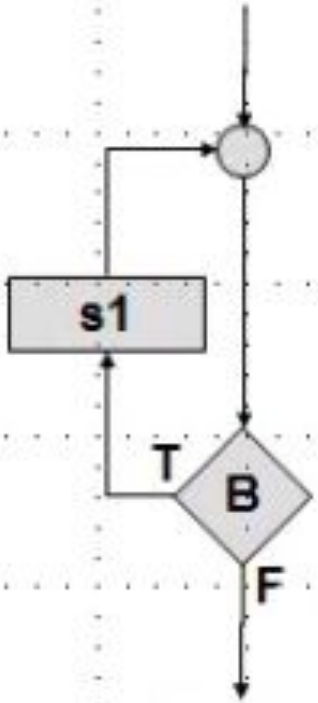
- Из данных элементарных блоков можно построить четыре блок-схемы, имеющих особое значение для практики алгоритмизации.



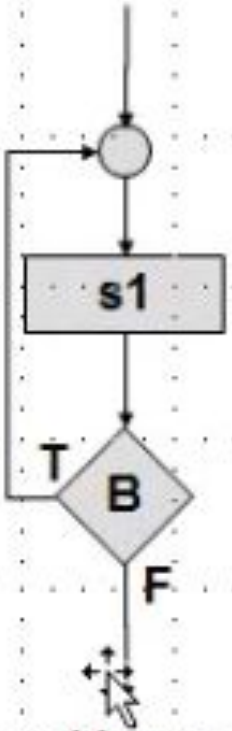
Композиция,
Следование



Альтернатива,
Развилка



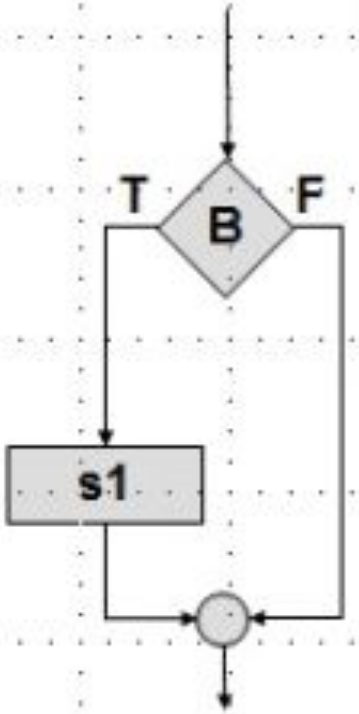
Цикл
с предусловием



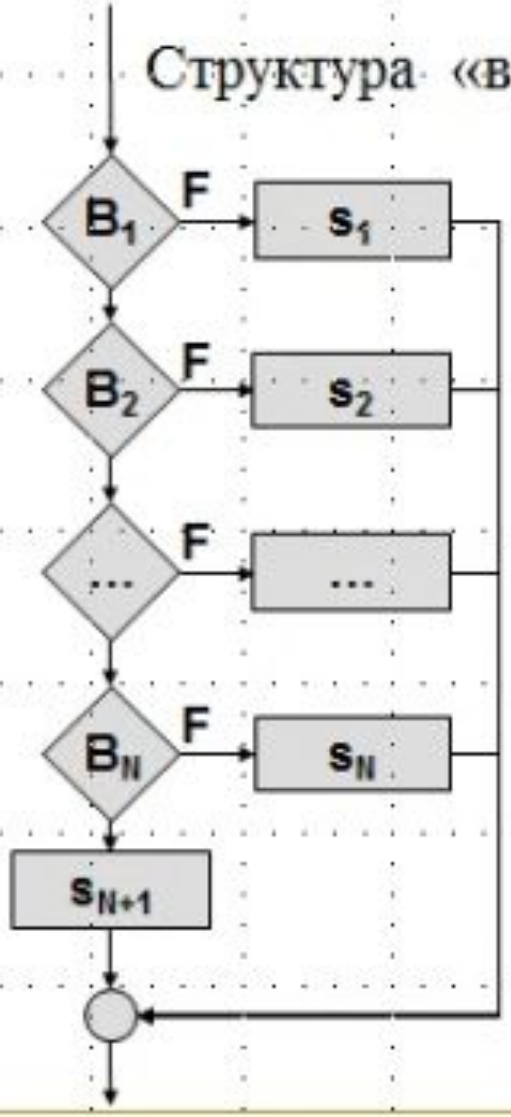
Цикл
с постусловием

Развитие структуры «альтернатива»

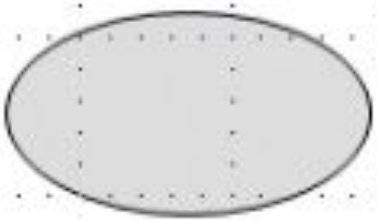
Неполная развилка (без else)



Структура «выбор»



Дополнительные блоки блок-схемы



Начало (конец) алгоритма

Выполнение операций, изменяющих команды

Вызов вспомогательного алгоритма

Ввод-вывод данных

Свойства алгоритмов

- Алгоритм должен быть составлен таким образом, чтобы исполнитель, в расчете на которого он создан, мог однозначно и точно следовать командам алгоритма и эффективно получать определенный результат.

Первое требование к алгоритмам

- Описываемый процесс должен быть разбит на конечную последовательность отдельных шагов.
- Возникающая в результате такого разбиения запись представляет собой упорядоченную совокупность четко разделенных друг от друга предписаний, образующих прерывную структуру алгоритма.
- Только выполнив требования одного предписания, можно приступить к выполнению следующего.
- Дискретная структура алгоритмической записи может, например, подчеркиваться сквозной нумерацией отдельных команд алгоритма, хотя это требование не является обязательным. Рассмотренное свойство алгоритмов называют **дискретностью**.

Второе требование к алгоритмам

- Используемые на практике алгоритмы составляются с ориентацией на определенного исполнителя.
- Чтобы составить для него алгоритм, нужно знать, какие команды этот исполнитель **может понять** и **исполнить**, а какие - **не может**.
- У каждого исполнителя имеется своя система команд.
- Составляя запись алгоритма для определенного исполнителя, можно использовать лишь те команды, которые имеются в его системе команд исполнения.

- Это свойство алгоритмов будем называть **понятностью**.

Третье требование к алгоритмам

- Будучи понятным, алгоритм не должен содержать предписаний, смысл которых может восприниматься неоднозначно, т.е. одна и та же команда, будучи понятна разным исполнителям, после исполнения каждым из них должна давать одинаковый результат.
- Запись алгоритма должна быть настолько четкой, полной и продуманной в деталях, чтобы у исполнителя не могло возникнуть потребности в принятии решений, не предусмотренных составителем алгоритма. Говоря иначе, алгоритм не должен оставлять места для произвола исполнителя.
- Кроме того, в алгоритмах недопустимы также ситуации, когда после выполнения очередной команды алгоритма исполнителю неясно, какая из команд алгоритма должна выполняться на следующем шаге.

- Отмеченное свойства алгоритмов называют **определенностью** или **детерминированностью.**

Четвертое требование к алгоритмам

- Обязательное требование к алгоритмам - **результативность.**
- Смысл этого требования состоит в том, что при точном исполнении всех предписаний алгоритма процесс должен прекратиться за конечное число шагов и при этом должен получиться определенный результат.
- Вывод о том, что решения не существует - тоже результат.

Пятое требование к алгоритмам

- Наиболее распространены алгоритмы, обеспечивающие решение не одной конкретной задачи, а некоторого класса задач данного типа.
- Это свойство алгоритма называют **массовостью.**
- В простейшем случае массовость обеспечивает возможность использования различных исходных данных.