Лекция 13: Базовые алгоритмические структуры. Данные, их типы, структуры и обработка

Информатика

Цель: рассмотреть

- основные понятия об алгоритме в программах и алгоритмизации решения задач.
- основные понятия о данных к алгоритмам, их базовые типы и структуры, вопросы их использования в алгоритмизации задач.

Алгоритм

- "Алгоритм" является базовым основополагающим понятием информатики, а алгоритмизация (программирование) – основным разделом курса информатики (ядром курса).
- 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 упорядоченная 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.

Алгоритм удовлетворяет следующим основным свойствам:

- Конечность (дискретность) команд и выполняемых по ним действий *алгоритма*.
- Выполнимость в определенной операционной среде (в определенном классе исполнителей).
- Результативность отдельных команд и всего *алгоритма*.
- Применимость *алгоритма* ко всем возможным входным данным конкретного класса задач.
- Определенность (детерминированность) команд и всего *алгоритма* для всех входных данных.
- Формализованное, конструктивное описание (представление) команд *алгоритма*.
- Минимальная полнота системы команд *алгоритм*.
- Непротиворечивость любых команд *алгоритма* на любом наборе входных данных.

- Любой *алгоритм* ориентирован на некоторый общий метод решения класса задач и представляет собой формализованную запись метода, процедуры.
- *Алгоритм*, записанный на некотором алгоритмическом, формальном языке, состоит из *заголовка алгоритма* (описания параметров, спецификаций класса задач) и *тела алгоритма* (последовательности команд исполнителя, преобразующих входные параметры в выходные).
- Для записи, исполнения, обмена и хранения *алгоритмов* существуют различные средства, языки, псевдокоды – блок-схемы, структурограммы (схемы Нэсси-Шнайдермана), Р-схемы, школьный алгоритмический язык (ШАЯ), различные языки программирования.

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

if <условие> then <команда, выполняемая при выполнении условия> else <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 = (МГУ, РГУ, КБГУ).

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.

- В зависимости от задачи рассматривают *данные*, которые имеют не только "линейную" (как приведенные выше), но и иерархическую структуру.

- *Пример.* Структуру "вуз" можно задать иерархической структурой, состоящей, например, из следующих уровней: "Ректорат", "Деканаты и подразделения", "Кафедры", "Отделы", "Преподаватели и сотрудники".

- 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*.

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 - *вспомогательным алгоритмом (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.

Свойства модулей:

- *функциональная целостность и завершенность* (каждый модуль реализует одну функцию, но реализует хорошо и 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);
- *корректность и надежность;*
- *ссылка на тело модуля происходит только по имени модуля*, то есть вызов и актуализация модуля возможны только через его заголовок.

Свойства (преимущества) модульного проектирования алгоритмов:

- возможность *разработки* алгоритма большого объема (алгоритмического комплекса) различными исполнителями;
- *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.
- *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 = b2 – 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).

Общая структура алгоритмического обеспечения

Основные формы использования алгоритмов – автономное, библиотечное, пакетное

- *Автономный алгоритм* определяется решаемой задачей, структурой используемых данных, структурой логических связей частей (модулей) алгоритма и языком {псевдокодов}, на котором представлен, описан алгоритм.
- *Библиотека алгоритмов* определяется множеством задач, решаемых с помощью библиотеки, множеством алгоритмов для решения типовых задач некоторой предметной области и структурой используемых данных.
- *Пакет алгоритмов*, как и библиотека, определяется множеством задач, решаемых с помощью пакета, множеством алгоритмов для решения типовых задач или их составных частей из некоторой предметной области, структурой используемых данных и обменов данными между задачами (модулями), специальным языком, на котором формулируется задание (последовательность этапов решаемой задачи, последовательность задач