Информатика (основы программирования)

Лекция 1.

Алгоритмы. Определение, оценка качества

Автор: Бабалова И.Ф.

Доцент, каф.12 2023 г.

Введение (1)

- Язык программирования С позиционируется среди множества языков как универсальный язык, который оказался удобным средством при практическом использовании для разработки программ самого разнообразного назначения.
- Язык был создан **Денисом Ритчи** для написания операционной системы UNIX и реализован в этой же операционной системе.

Введение (2)

- Но язык C оказался не привязанным к конкретной аппаратуре или системе, на нем легко писать программы, которые без каких-либо изменений переносятся на системы, где осуществляется его поддержка.
- Последний из стандартов языка C был принят в 1999 году, и получилась современная версия языка C99. Разработка программного интерфейса на основе стандартной библиотеки функций языка C в соответствии со стандартом C99 гарантирует мобильность исходного кода прикладного программного обеспечения в любых операционных средах.

Введение (3)

- В языке заложена возможность работать непосредственно с битами, байтами, указателями и словами. Этот уровень данных позволяет разрабатывать программы для системного программирования, таких как редакторы, трансляторы, компоновщики.
- Язык С был создан и апробирован активно работающими программистами. В результате С обеспечивает то, чего и ждут от него именно программисты: небольшое количество ограничений, блочную структуру, автономные функции и малое количество ключевых слов.

Введение (4)

- Программы, написанные на языке С, обладают эффективностью программ, написанных на языке ассемблера, и структурированностью, присущей программам, созданным на языках типа Pascal. Поэтому неудивительно, что во всем мире С стал универсальным языком программирования.
- Решающим фактором успеха языка С стало то, что во многих случаях он может быть использован вместо ассемблера, который основан на символическом представлении бинарного кода, непосредственно выполняемого компьютером. Все следующие расширения языка типа C++ , C# и его визуальные варианты только доказывают жизнеспособность языка С.

Рекомендуемая литература

ОСНОВНАЯ ЛИТЕРАТУРА:

1. Кормен Т. Х. и др. Алгоритмы: построение и анализ, 3-е изд. : Пер. с англ. – М. : ООО «И. Д. Вильямс», 2013. – 1328 с. : ил. – Парал. тит. англ.
2. Керниган Б. У., Ритчи Д. М. Язык программирования С, 2-е изд. : Пер. с англ. – М. : Издательство «Вильямс», 2013. – 304 с. : ил. – Парал. тит. англ.
3. Шилдт Г. С: полное руководство, классическое издание, 4-е изд. : Пер. с англ. – М. : Издательство «Вильямс», 2018. – 699 с. : ил. – Парал. тит. англ.
4. Робачевский А. М., Немнюгин С. А., Стесик О. Л. Операционная система UNIX. – 2-е изд., перераб. и доп. – СПб. : БХВ-Петербург, 2010. – 656 с. : ил.
5. Вавренюк А. Б., Кутепов С. В., Курышева О. К., Макаров В. В. Операционные системы. Основы UNIX. Учебное пособие – М. : ИНФРА-М, 2019. – 160 с. : ил.
6. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона + CD / Пер. с англ. Ткачев Ф. В. – М. : ДМК Пресс, 2010. – 272 с. : ил.
7. http://bigor.bmstu.ru/?cnt/?doc=OP2/OP_T.cou

ДОПОЛНИТЕЛЬНАЯ ЛИТЕРАТУРА:

1. Д. Кнут т.1, 2, 3 Искусство программирования. Электронная версия.

Цель курса

Научиться:

- Анализировать формулировку задачи
- Определять типы данных для решения задачи на компьютере
- Разрабатывать алгоритм решения поставленной задачи
- Разрабатывать программу на языке программирования в соответствии с созданным алгоритмом
- Отлаживать решение задачи на достаточном количестве тестов
- Доказывать правильность решения задачи
- Изучить язык Си и правила работы в ОС Linux.

Определение программы

- Программа – это записанная на языке, понятном компьютеру, последовательность действий для получения конкретного результата
- Алгоритм + структура данных (Определение по Никлаусу Вирту) = программа

Определение алгоритма

- Алгоритм – это конечное множество правил, определяющее процесс переработки одной, входной системы данных, в другую, выходную, систему данных.

- Аналогичные термины:
  - Процесс
  - Рецепт
  - Метод
  - Способ

- Алгоритм должен строго подчиняться следующим свойствам:

Свойства алгоритма

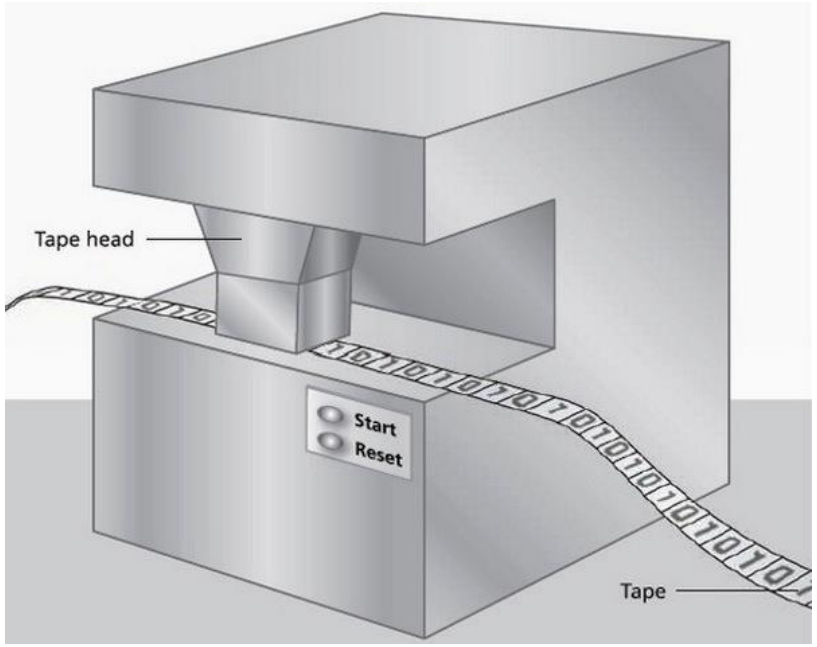
1. Переход от одной (входной) системы данных к другой (выходной) осуществляется за конечное число шагов - **конечность алгоритма**.
2. Процесс решения задачи обеспечивается отдельными операциями, следующими друг за другом – **дискретность алгоритма**.
3. Каждое правило по своей входной системе данных однозначно определяет выходную систему данных, независимо от времен и средств, использованных для решения задачи - **однозначность алгоритма**.
4. Исходная система данных для алгоритма выбирается из некоторого потенциально бесконечного множества данных – **массовость алгоритма**.

Универсальность алгоритма

- Свойство **массовости** алгоритма ни в коем случае не подразумевает, что можно разработать один алгоритм для решения всех задач.
- Алгоритм для решения конкретной задачи должен быть применим к любой совокупности данных, – вот в чем его **массовость**.
- Понятие **универсального** алгоритма – это утопия.
- Алгоритм разрабатывается для решения определенной задачи и подзадачи некоторого класса задач. В теории алгоритмов сформулирован постулат:

Универсальный алгоритм не существует.

Машина Тьюринга



Предельно простая и общая условная схема автоматической вычислительной машины

- Любой вычислительный или логический процесс, для которого существует алгоритм, может быть автоматизирован с помощью такой примитивной машины.
- И наоборот: все, что можно сделать с помощью этой машины, подчинено алгоритму.
- Задачи, которые этой машиной не решаются – алгоритмически неразрешимы, для любой, даже самой мощной машины (не только сегодняшнего дня, но и будущего).
- С алгоритмически неразрешимыми задачами способен справиться только мозг человека.

Определение инварианта (1)

- Для реализации повторяющихся действий введено понятие цикла. Цикл может быть реализован только при наличии инварианта.

- **Инвариант** (Invariant – неизменяющийся) – это числа, алгебраические выражения, величины, связанные с каким-либо математическим объектом и остающиеся неизменными при преобразованиях этого объекта.

Определение инварианта (2)

Рассмотрим цикл поиска некоторого числа x в заданной последовательности чисел. Дано: [Ai] = A1,A2,…,An

1. Перед входом в цикл определяем **начало поиска** и его **конец**:
i=1 и количество чисел в последовательности n.

2. Если действие выполняется и находится x=A[i], то число найдено. Если значение не возвращается, то числа нет в последовательности. Перед каждым следующим действием i увеличивается на 1.

3. Цикл должен завершиться при условиях:
а) i>n и отсутствием значения в результате. Это утверждение должно сопровождаться или признаком, или комментарием об отсутствии искомого числа в последовательности.
б) Найдено число x=A[i]. Получен искомый результат.

Приведённая последовательность действий является инвариантом для решения любой задачи на компьютере.

Оценка алгоритма – правильность

- Метод доказательства – **инвариант** цикла. Это утверждение, для которого демонстрируется истинность инварианта в начале каждого действия цикла. Каждый цикл должен содержать три его действия:
- **Инициализация**. Инвариант цикла истинен перед первым выполнением цикла.
- **Сохранение**. Если инвариант цикла истинен перед первым циклом, то он остаётся истинным и после его выполнения.
- **Завершение**. После завершения цикла вместе с причиной завершения цикла получен искомый результат работы алгоритма.

Оценка алгоритма – конечность

- Наличие **инварианта** в алгоритме обеспечивает свойство конечности алгоритма

Оценка алгоритма – эффективность

Решение конкретной задачи должно **выполняться** за наименьшее время и **требовать** наименьший объем компьютерной памяти

1. Арифметические действия различны по времени – сложение и деление, например.
2. Количество действий должно быть минимальным
3. Оценка времени работы алгоритма зависит от количества обрабатываемых данных n. Эта оценка может находиться в диапазоне

от O(n) до O (n^2).

4. Время выполнения алгоритма зависит и от используемого типа данных и места хранения данных.

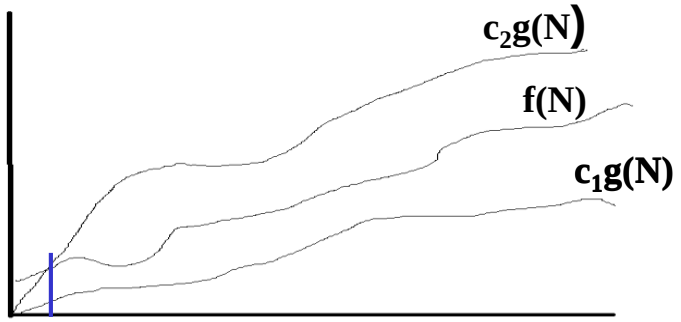
5. Размер памяти, которую может потребовать алгоритм для реализации его на компьютере, может быть недостаточным.

Оценка скорости работы алгоритма

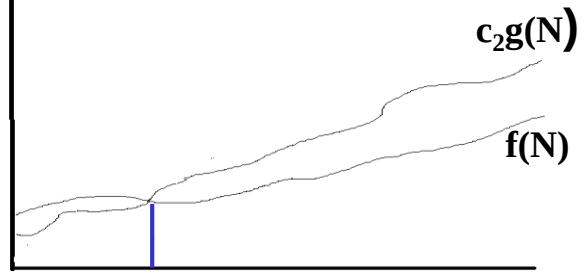
Функция f(N) имеет порядок сложности Θ(g(N)), если существуют постоянные переменные c1, c2 и N1, такие, что для всех N > N1

0 <= c1*g(N) <= f(N) <= c2*g(N).

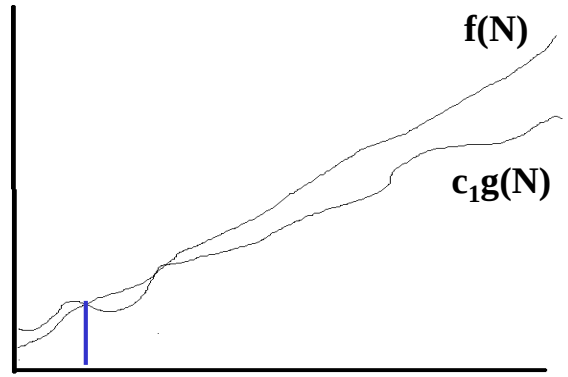
В этой оценке мы рассматриваем ограничения скорости, как снизу, так и сверху



Сложность Θ-teta



Сложность O-micron



Сложность Ω-omega

Примеры оценки эффективности алгоритмов (1)

1. $f(N) = N^5 + 7N^2 - 14N$, тогда можно сказать, что алгоритмическая сложность будет оцениваться, как $\Theta(N^5)$.
2. $f(N) = N^{18} + 10 \log_2 N = \Theta(N^{18})$

Примеры оценки эффективности алгоритмов (2)

3. Задан массив A длиной N элементов. Найти номер первого вхождения элемента со значением Z. Сколько операций потребуется, чтобы обнаружить искомый номер с помощью алгоритма, который просматривает последовательность элементов массива?

Самый первый элемент массива может оказаться Z, следовательно, минимальное количество операций поиска будет $K_{min} = 1$. Элемента в массиве может не быть совсем, и тогда для поиска потребуется ровно N операций, $K_{max} = N$. А какое среднее значение количества поисков? Предполагая, что количество итераций алгоритма, требуемых для поиска, равномерно распределено по всем числам от 1 до N, получаем формулу:

$$\kappa_m = \frac{N*(N+1)}{2*N} = \frac{N+1}{2}$$

Для данного алгоритма нам проще будет использовать O-нотацию: f(N) = O(N). Эта нотация ограничена только сверху.

Примеры интересных алгоритмов (1)

Решение задачи о вычислении степени некоторого числа X.

- Число X может быть целым или вещественным. Степень числа n – целое число. Возведение в степень обычно выполняется умножением n-1 раз на число X, следовательно, оценка сложности алгоритма будет определяться как f(N)=O(N).

Примеры интересных алгоритмов (2)

Рассмотрим формулировку возведения в степень, называемую в литературе «Индийским алгоритмом»

При n=1 $x^n = x$;

При n>1 $x^n = x^{n\ mod 2} \left(x^{n\ div\ 2}\right)^2$

- Эта запись не относится ни к одному языку программирования. Деление по модулю – это mod, а деление нацело – div (сокр. от divide)
- Сокращение количества умножений обеспечивается за счёт того, что при каждом выполнении вызова операции происходит не более одного деления, одного возведения в квадрат и одного умножения, поэтому общее количество арифметических операций не больше $3\log_2 n$. При больших значениях n это число будет существенно меньше обычных n - 1 умножений. Например, при n = 1000 это примерно 30 операций.

$25^{35} = 25 * (25^{17})^2$ → .... $25*(25^1)^2$

Декомпозиция алгоритма (1)

Решение каждой конкретной задачи разбивается на последовательность простых действий – этапов, понятных компилятору

- Каждый этап должен давать конкретный результат
- Из множества этапов должно быть получено решение задачи любой сложности

Декомпозиция алгоритма (2)

Этот процесс разложения задач формулируется алгоритмической парадигмой «Разделяй и властвуй»:

1. **Разделение** - задача разбивается на несколько подзадач, которые представляют собой смысловые части решения основной задачи.

2. **Властвование** - решаются подзадачи. Порядок решения всех подзадач определяется алгоритмом решения всей задачи.

3. **Объединение** - решения всех подзадач объединяются в решение всей задачи.

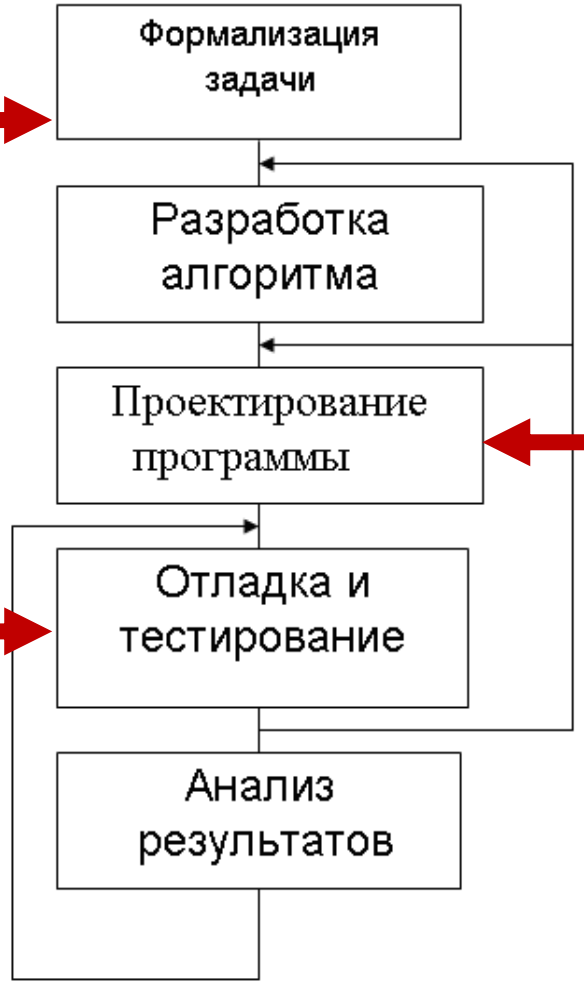
Алгоритмическая система

- Определение типов данных.
- Декомпозиция алгоритма.
- Разработка алгоритма для решения конкретной задачи.
- Описание действий алгоритма в форме, удобной для проверки человеком и понятной компьютеру.

Для публикации алгоритмов утверждены государственные и международные стандарты.

Этапы разработки программ

Формулировка метода решения задачи и описание данных ее решения



Написание кода программы в соответствии с технологией программирования

Проверка решения задачи

Изображение алгоритмов (1)

Для наглядного представления алгоритмов принята система геометрических фигур, каждая из которых относится к определенному алгоритмическому правилу.

Основные фигуры:

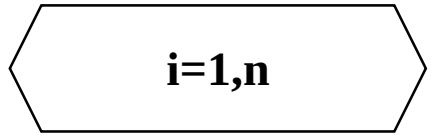
- Действие (процесс)
  - A = 20;
  - X = Y;

b=1.5 a

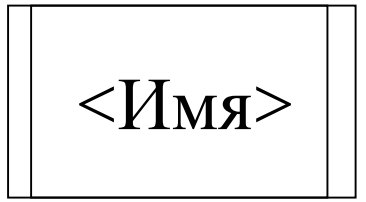
Изображение алгоритмов (2)

- Условие
  - X > Y
  - $x \in [1;2] \rightarrow (x \geq 1) and\ (x \leq 2)$

- Цикл
  - Заголовок цикла с заданным числом повторений
  - Для циклов с неизвестным числом повторений нет специальной фигуры

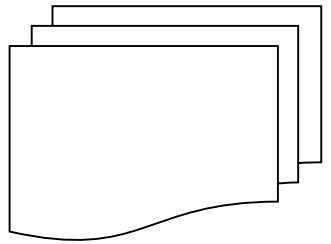


- Предопределённый процесс
  - Это ранее описанные подпрограммы (функции)



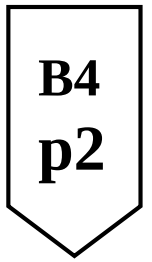
Изображение алгоритмов (3)

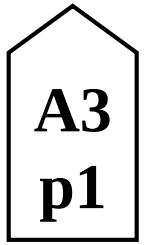
- Блок в алгоритме обозначает, где надо печатать или визуально представлять решение задачи



- Коннектор
  - Точка объединения частей алгоритма

- Межстраничный соединитель
  - Внутри – координаты блока на указанной странице

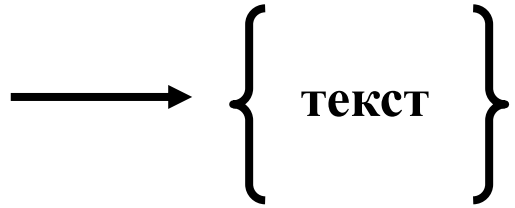




Изображение алгоритмов (4)

- Пуск – останов
  - Фигура, обозначающая вход в алгоритм или его завершение

- Комментарий



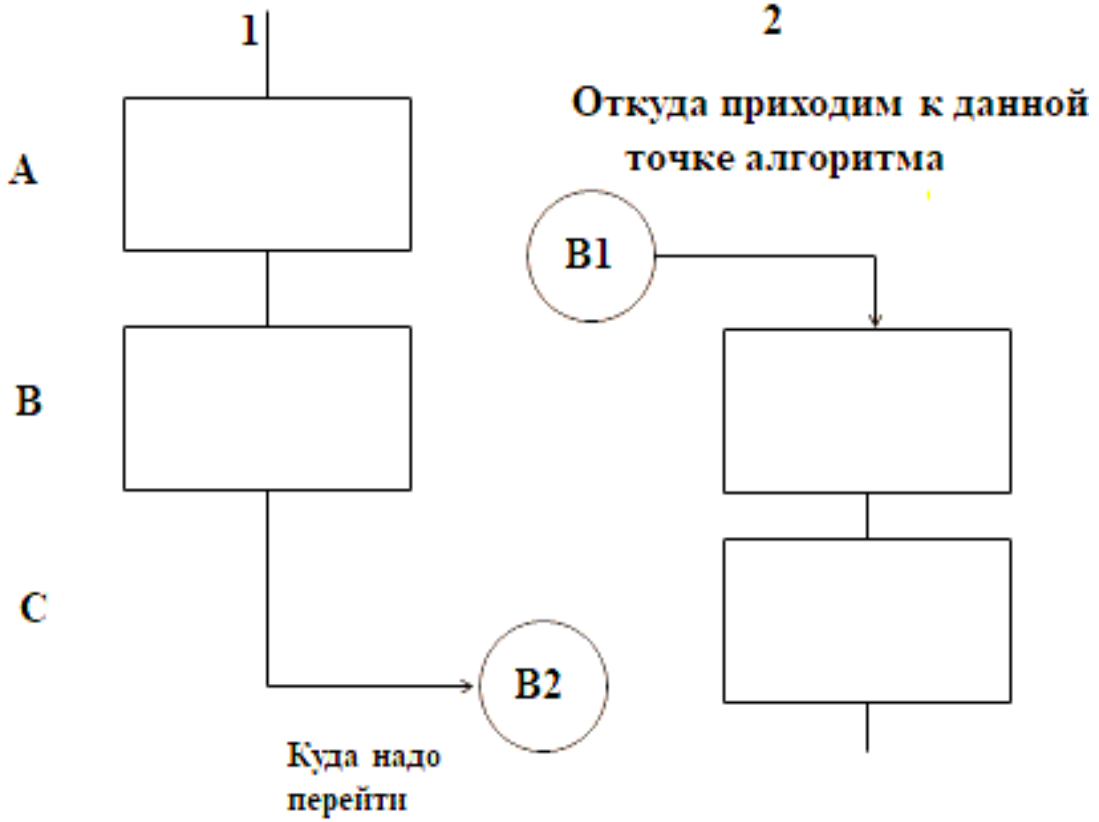
- Ввод / вывод

Изображение алгоритмов (5)

- Соединение фигур в изображении алгоритмов обеспечивается **только вертикальными** и **горизонтальными** линиями соединения.
- Допускаются линии соединения **ломаные**, но с прямым углом.
- Все линии соединения указывают на путь решения задачи, то есть **направление движения** по алгоритму.
- Поэтому следует избегать изображения **длинных** линий соединения, **пересекающихся** линий и линий с **произвольным углом** наклона.

Правила соединения частей алгоритма

- Все блоки имеют координаты
- Горизонталь – буква латинского алфавита, вертикаль - цифра
- Линии только горизонтальные и вертикальные.
- Точки соединения обозначаются координатами блоков, которые они связывают. Своих координат они не имеют.
- Войти в блок можно только через его начало



Спасибо за внимание