МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

ПОСТРОЕНИЕ И АНАЛИЗ АЛГОРИТМОВ

Методические рекомендации к лабораторным работам для студентов специальности 1-53 01 02 «Автоматизированные системы обработки информации» очной и заочной форм обучения

Могилев 2022

УДК 604.43
ББК 32.973-018.1
П43

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Автоматизированные системы управления» «22» ноября 2022 г., протокол № 4

Составитель ст. преподаватель И. А. Беккер

Рецензент канд. техн. наук, доц. С. К. Крутолевич

Методические рекомендации к лабораторным работам содержат краткие теоретические сведения, задания, общие требования к отчету. Предназначены для студентов специальности 1-53 01 02 «Автоматизированные системы обработки информации» очной и заочной форм обучения.

Учебно-методическое издание

ПОСТРОЕНИЕ И АНАЛИЗ АЛГОРИТМОВ

Ответственный за выпуск	А. И. Якимов
Корректор	А. А. Подошевко
Компьютерная верстка	Е. В. Ковалевская

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Целью учебной дисциплины является формирование у студентов теоретических знаний о построении и анализе алгоритмов, знакомство с практическими основами исследования алгоритмов.

Задачами учебной дисциплины являются:

– формирование у студентов знаний о способах построения и записи алгоритма, типах сложности алгоритма, классах вычислительной сложности задач;

– обучение методике подсчета трудоемкости и временных затрат алгоритма и анализу алгоритма через функцию его роста.

В результате освоения учебной дисциплины студент будет:

– знать:

а) основы теории построения и анализа алгоритмов;

б) асимптотические оценки алгоритмов;

в) классы вычислительной сложности задач.

– уметь:

а) выполнять исследование алгоритма;

б) оценивать характер роста вычислительной сложности алгоритмов.

– владеть:

а) средствами измерения времени в программных реализациях алгоритмов;

б) инструментами оценки трудоемкости алгоритмов.

Лабораторные работы предназначены для знакомства с основами теории алгоритмов, практическими методиками исследования алгоритмов.

В лабораторных работах предполагается написание на языке C# программы, которая выполняет логически законченную задачу, предоставляя пользователю удобный интерфейс, наглядно отображая результаты работы.

Алгоритмы, заложенные в программу, должны решать задачу полностью и корректно, адекватно реагируя на нестандартные воздействия.

При защите лабораторных работ студент описывает используемые типы, структуры данных и обосновывает их выбор, отвечает на вопросы по логике и реализации алгоритма, на контрольные вопросы.

Для записи алгоритма в методических рекомендациях используются псевдокод и обобщенный язык программирования Pascal.

1 Лабораторная работа № 1. Машина Тьюринга и рекурсивные функции

Цель работы: сформировать умения применять программу машины Тьюринга к входному слову на ленте; научиться строить машину Тьюринга; получать примитивно рекурсивную функцию.

Теоретические сведения

Машина Тьюринга и рекурсивные функции являются примерами формального алгоритма.

Машина Тьюринга состоит из внешней памяти (бесконечной ленты) с заданным внешним алфавитом A и управляющего устройства с набором его состояний (внутренним алфавитом Q).

Работа машины Тьюринга осуществляется посредством команд вида $q_i a_m \rightarrow q_j L a_n$: если управляющее устройство в состоянии q_i обозревает ячейку с символом a_m, то оно меняет свое состояние на q_j, а содержимое ячейки на a_n и сдвигается влево.

Обозначение сдвига вправо – R, не смещаться – S.

Состояние Стоп обозначается как q_0. Машина Тьюринга остановится, когда управляющее устройство придет в состояние Стоп.

Команды машины Тьюринга выполняются в соответствии заданной программой в зависимости от условий: текущего состояния управляющего устройства q_i и текущего символа a_m. Последовательность выполняемых команд вовсе не должна соответствовать их последовательности в программе машины Тьюринга.

Эквивалентной машине Тьюринга формальной моделью алгоритма являются рекурсивные функции Черча.

Теория рекурсивных функций исследует класс вычислимых функций на базе некоторой системы аксиом.

Говорят, что (n + 1)-местная функция f получена из n-местной функции g и (n + 2)-местной функции h с помощью оператора примитивной рекурсии, если для любых $x_1, \ldots, x_n$, y справедливы равенства

$$f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n),$$

$$f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots x_n, y)).$$

Причем независимо от числа аргументов в φ рекурсия ведется только по одной переменной y.

Остальные n переменных $x_1, \ldots, x_n$ на момент применения схемы примитивной рекурсии зафиксированы и играют роль параметров.

Пример 1 – Машина Тьюринга задается следующей программой:

$q_1 0 \rightarrow q_1 0R$, $q_1 1 \rightarrow q_2 0R$, $q_2 0 \rightarrow q_0 1S$, $q_2 1 \rightarrow q_1 0R$.

Найти результат применения заданной машины Тьюринга к записи на ленте: 0111010.

Решение

$0\ \underline{1}11010 \to 00\ \underline{1}1010 \to 000\ \underline{1}010 \to 0000\ \underline{0}10 \to 0000\ \underline{1}10.$
$q_1 \qquad q_2 \qquad q_1 \qquad q_2 \qquad q_0$

Пример 2 – Построить машину Тьюринга, правильно вычисляющую функцию $f(x, y) = x + y$.

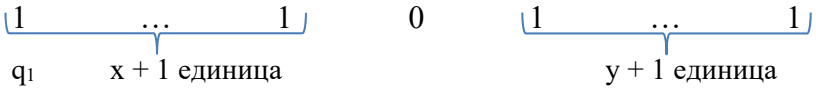
Решение

Для записи чисел в этой модели представления данных на ленте будем использовать бинарную систему счисления, в которой 0 представляет собой пустой символ ленты.

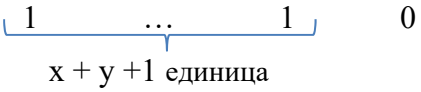
Тогда 0 как значащий символ обозначится на ленте так: 1.

1 обозначится на ленте так: 11, а число 2 через набор из трех единиц – 111.

Число n будем записывать с помощью последовательности из n + 1 единицы, поэтому пара (x; y) в начальном состоянии на ленте будет задаваться следующим образом:



После работы машины Тьюринга на ленте должно быть выходное слово следующего вида:



Нужно, чтобы машина Тьюринга вначале, двигаясь вправо, заменяла нуль, разделяющий массивы единиц, единицей. Затем, пройдя массив единиц, нужно начинать движение влево, заменяя две последние единицы нулями.

Программа такой машины может иметь следующий набор команд:

$q_1 1 \to q_1 1R$, $q_1 0 \to q_2 1R$, $q_2 1 \to q_2 1R$, $q_2 0 \to q_3 0L$, $q_3 1 \to q_4 0L$, $q_4 1 \to q_5 0L$, $q_5 1 \to q_5 1L$, $q_5 0 \to q_0 0R$.

Пример 3 – Найти функцию f, получаемую с помощью операции примитивной рекурсии из функций $g(x) = x$, $h(x, y, z) = zy + z$.

Решение

Функцию f в общем случае строят по схеме

$$f(x_1,\ldots,x_n, 0) = g(x_1, \ldots, x_n),$$
$$\ldots$$
$$f(x_1\ldots x_n, y+1)= h(x_1\ldots x_n, y, f(x_1\ldots x_n, y)). \quad (1)$$

Функцию f от двух переменных будем строить по схеме

$$f(x, 0) = g(x),$$
$$\ldots$$
$$f(x, y+1)= h(x, y, f(x, y)). \quad (2)$$

$f(x, 0) = x.$
$f(x, 1)= h(x, 0, f(x, 0)) = h(x, 0, x) = x.$
$f(x, 2)= h(x, 1, f(x, 1)) = h(x, 1, x) = 2x.$
$f(x, 3)= h(x, 2, f(x, 2)) = h(x, 2, 2x) = 6x.$
$f(x, 4)= h(x, 3, f(x, 3)) = h(x, 3, 6x) = 24x.$
…
Явное задание построенной рекурсивно функции f:
$f(x, y) = y!x.$

Варианты заданий. Выполните задания согласно выданному преподавателем варианту (таблица 1).

Таблица 1 – Задания по вариантам

Вариант	1	2	3	4	5	6	7	8	9	10	11	12
Задание	1-1	1-2	1-8	1-6	1-7	1-5	1-10	1-11	1-9	1-3	1-4	1-8
	2-1	2-4	2-9	2-10	2-5	2-2	2-12	2-3	2-11	2-6	2-8	2-7
	3-3	3-4	3-5	3-7	3-6	3-8	3-1	3-2	3-10	3-12	3-11	3-9

Продолжение таблицы 1

Вариант	1	2	3	4	5	6	7	8	9	10	11	12
Задание	1-11	1-10	1-12	1-9	1-5	1-8	1-1	1-3	1-7	1-2	1-4	1-6
	2-5	2-2	2-10	2-12	2-11	2-6	2-4	2-9	2-7	2-3	2-8	2-1
	3-1	3-2	3-7	3-8	3-10	3-3	3-4	3-11	3-12	3-6	3-9	3-5

Продолжение таблицы 1

Вариант	13	14	15	16	17	18	19	20	21	22	23	24
Задание	1-7	1-3	1-11	1-10	1-8	1-7	1-12	1-5	1-4	1-1	1-6	1-2
	2-8	2-12	2-6	2-2	2-3	2-4	2-1	2-5	2-7	2-12	2-9	2-10
	3-4	3-9	3-1	3-8	3-12	3-	3-2	3-5	3-11	3-7	3-3	3-6

Задание 1

Согласно выданному варианту (таблица 2) найдите результат применения машины Тьюринга к записи на ленте К, считая, что управляющее устройство установлено под первой слева единицей и находится в состоянии q_1.

Таблица 2 – Программы работы машины Тьюринга

Вариант	Машина Тьюринга 1	Машина Тьюринга 2
1	а) $q_1 1 \to q_1 1 R$ $q_1 0 \to q_2 1 L$ $q_2 1 \to q_2 1 L$ $q_2 0 \to q_0 0 R$. К: 0111101011	б) $q_1 1 \to q_1 1 R$ $q_1 0 \to q_2 0 R$ $q_2 1 \to q_3 1$ $q_2 0 \to q_2 0 R$ $q_3 1 \to q_3 1 R$ $q_3 0 \to q_0 0$. К:1101010011
2	а) $q_1 1 \to q_1 1 R$ $q_1 0 \to q_2 0 R$ $q_2 1 \to q_3 1$ $q_2 0 \to q_2 0 R$ $q_3 1 \to q_3 1$ R $q_3 0 \to q_0 0$. К:11001101	б) $q_1 1 \to q_1 1 R$ $q_1 0 \to q_2 1 L$ $q_2 1 \to q_2 1 L$ $q_2 0 \to q_0 0 R$. К:1101101001
3	а) $q_1 0 \to q_1 1 R$ $q_1 1 \to q_2 1 L$ $q_2 1 \to q_2 1$ $q_2 0 \to q_0 0 R$. К:01001100110	б) $q_1 1 \to q_1 1$ $q_1 0 \to q_2 0 R$ $q_2 1 \to q_3 1$ $q_2 0 \to q_2 0 R$ $q_3 1 \to q_3 1 R$ $q_3 0 \to q_0 0 L$. К:100110001
4	а) $q_1 1 \to q_1 0$ $q_1 0 \to q_2 1 L$ $q_2 1 \to q_2 1 L$ $q_2 0 \to q_0 0 R$. К:011011101010	б) $q_1 1 \to q_2 1 R$ $q_1 0 \to q_1 0$ $q_2 1 \to q_1 0 R$ $q_2 0 \to q_3 0 L$ $q_3 1 \to q_0 1 L$ $q_3 0 \to q_3 1$. К:10111001
5	а) $q_1 1 \to q_1 1 R$ $q_1 0 \to q_2 0 L$ $q_2 1 \to q_2 1 L$ $q_2 0 \to q_0 1$. К:11001101110	б) $q_1 1 \to q_1 1 R$ $q_1 0 \to q_2 0 R$ $q_2 1 \to q_3 1 R$ $q_2 0 \to q_2 0$ $q_3 1 \to q_3 1$ $q_3 0 \to q_0 0 L$. К:10100101

Окончание таблицы 2

Вариант	Машина Тьюринга 1	Машина Тьюринга 2
6	а) $q_1 1 \to q_2 1$ $q_1 0 \to q_2 0 R$ $q_2 1 \to q_0 1 R$ $q_2 0 \to q_1 1 L$. K:0110101011	б) $q_1 1 \to q_2 1 R$ $q_1 0 \to q_1 0 R$ $q_2 1 \to q_1 0$ $q_2 0 \to q_3 0 L$ $q_3 1 \to q_2 1 L$ $q_3 0 \to q_0 0 R$. K:11110101
7	а) $q_1 1 \to q_2 1$ L $q_1 0 \to q_0 0 R$ $q_2 1 \to q_2 1 R$ $q_2 0 \to q_1 1 R$. K:01010011110	б) $q_1 1 \to q_2 1 R$ $q_1 0 \to q_1 0 R$ $q_2 1 \to q_1 0$ $q_2 0 \to q_3 0 L$ $q_3 1 \to q_2 1 L$ $q_3 0 \to q_0 0 R$. K:10100101
8	а) $q_1 1 \to q_2 1$ R $q_1 0 \to q_2 0 L$ $q_2 1 \to q_2 1 R$ $q_2 0 \to q_1 1 L$. K:01011110110	б) $q_1 1 \to q_2 1 R$ $q_1 0 \to q_1 0$ $q_2 1 \to q_1 0$ $q_2 0 \to q_0 0 L$ $q_3 1 \to q_2 1 L$ $q_3 0 \to q_2 0$. K:11010001
9	а) $q_1 1 \to q_1 1 R$ $q_1 0 \to q_2 0 L$ $q_2 1 \to q_0 1 L$ $q_2 0 \to q_2 0 R$. K:11011001	б) $q_1 1 \to q_1 1 R$ $q_1 0 \to q_2 0 R$ $q_2 1 \to q_3 1 R$ $q_2 0 \to q_2 0$ $q_3 1 \to q_1 1 R$ $q_3 0 \to q_0 1 L$. K:0111101110
10	а) $q_1 1 \to q_1 1 R$ $q_1 0 \to q_2 0 R$ $q_2 1 \to q_3 1 R$ $q_2 0 \to q_0 0$ $q_3 1 \to q_3 1$ R $q_3 0 \to q_0 0$. K:0101011010	б) $q_1 1 \to q_1 1 R$ $q_1 0 \to q_2 0 L$ $q_2 1 \to q_2 1 L$ $q_2 0 \to q_0 0$. K:10011101
11	а) $q_1 1 \to q_1 1$ $q_1 0 \to q_3 0 L$ $q_2 1 \to q_3 1 R$ $q_2 0 \to q_0 1$ $q_3 1 \to q_3 1$ R $q_3 0 \to q_0 0$. K:101101110	б) $q_1 1 \to q_3 1 L$ $q_1 0 \to q_0 0 R$ $q_2 1 \to q_3 1 R$ $q_2 0 \to q_2 0$ $q_3 1 \to q_1 1 L$ $q_3 0 \to q_1 0$. K:10100011

Задание 2

Постройте машину Тьюринга, правильно вычисляющую функцию:

1) f(x) = x + 2;
2) f(x) = x – 1;
3) f(x) = 10 – x;
4) f(x) = x – 2;
5) f(x) = 1 + x;
6) f(x) = y – x;
7) f(x) = x + 3;
8) f(x,y) = y – x – 1;
9) f(x, y) = x – y;
10) f(x, y) = x + y – 2;
11) f(x, y) = x – y + 1;
12) f(x, y) = x – y + 3;
13) f(x, y) = x + y – 1;
14) f(x,y) = x + y + 2.

Задание 3

Найдите функцию f(x, y), получаемую из g(x) и h(x, y, z) с помощью операции примитивной рекурсии:

1) g(x) = x, h(x, y, z) = x + 2z;
2) g(x) = 3x, h(x, y, z) = x + z;
3) g(x) = 3, h(x, y, z) = 2xy + z;
4) g(x) = 1, h(x, y, z) = xyz + z;
5) g(x) = x + 2, h(x, y, z) = x + xy + 2z;
6) g(x) = 4x, h(x, y, z) = x + y;
7) g(x) = 5, h(x, y, z) = xyz;
8) g(x) = x + 1, h(x, y, z) = x + 2yz;
9) g(x) = 2x, h(x, y, z) = xy + 2z;
10) g(x) = x, h(x, y, z) = x + y + z;
11) g(x) = x + 3, h(x, y, z) = x + yz;
12) g(x) = 2, h(x, y, z) = xy + xz.

Контрольные вопросы

1 Дайте определение формального алгоритма, разработанное А. Тьюрингом.
2 Дайте определение формального алгоритма, принадлежащее А. Черчу.
3 Как связаны эти определения?
4 Что такое рекурсивная функция? Какие есть виды рекурсивных функций?
5 Как они строятся? Какие операторы для этого используются?
6 Какие функции берутся в качестве исходных при построении класса рекурсивных функций?
7 Докажите, что функция следования вычислима по Тьюрингу.
8 Дайте определение оператору суперпозиции.
9 Как по-другому называется этот оператор? Приведите пример сложной функции из алгебры.
10 Приведите тезис Черча.

2 Лабораторная работа № 2. Построение и анализ алгоритма вычисления значения полиномиальной функции

Цель работы: сформировать знания и умения использовать блок-схемы, язык программирования для записи алгоритма; развить умения анализировать алгоритм с помощью функции трудоемкости.

Теоретические сведения

Для анализа алгоритмов применим следующий подход подсчета трудоемкости: договоримся считать элементарными операциями следующие операции алгоритма

1) арифметические;
2) логические;
3) индексация, обращение по каждому индексу считается за одну операцию;
4) присваивание;
5) сравнение;
6) вызов свойства;
7) вызов метода, причём каждый входной параметр даёт одну элементарную операцию.

Трудоемкость конструкции «Следование» есть сумма трудоемкостей блоков, следующих друг за другом. Таким образом, для вычисления теоретически трудоемкости f_A алгоритма A суммируют трудоемкости последовательно идущих друг за другом блоков кода.

С учетом договоренности о том, какие операции в алгоритме мы считаем элементарными, трудоемкость строки кода

```
a[1] = a[0] + 2*(1 + itog)
```

будет равна 6.

Основными блоками алгоритма являются «Ветвление» и «Цикл».

Теоретическая трудоемкость F_{If} конструкции If вычисляется как

$$F_{If} = f_{условия} + f_{+} \cdot p_{+} + f_{-} \cdot p_{-},$$

где f_{+} и f_{-} – трудоемкости ветвей;

p_{+} и p_{-} – вероятности попадания на конкретную ветвь блока условия.

Экспериментальная трудоемкость блока, алгоритма – это количество элементарных операций, подсчитанных счетчиком в коде.

Для конструкции If теоретически подсчитанное и полученное экспериментально значения трудоемкости могут отличаться.

Трудоемкость F_{For} конструкции For зависит от количества итераций и в общем случае определяется по формуле

$$F_{For} = 1 + 3N + N \cdot f_{тела\ цикла},$$

где N – количество итераций цикла.

Трудоемкость F_{For} стандартной конструкции For в C# определяется по формуле

$$F_{For} = 2 + 2N + N \cdot f_{\text{тела цикла}},$$

где N – количество итераций цикла.

В лабораторной работе исследованию и анализу подлежит алгоритм вычисления значения полиномиальной функции по схеме Горнера – без использования возведения в степень.

Полиномиальную функцию в общем случае удобно задавать набором коэффициентов. Предположим, что последовательность коэффициентов находится в массиве. В реальных программах она также может читаться из файла или из более сложных структур данных, например, из списка.

Отметим следующие составные части алгоритма, вычисляющего функцию на последовательности элементов, например, при суммировании элементов последовательности:

1) инициализация значения функции s для пустой последовательности s : = 0.0;

2) вычисление нового значения функции по прочтению очередного элемента последовательности. Новое значение вычисляется по старому значению и очередному прочитанному элементу. В данном случае это суммирование s : = s + a[i].

Эти две части присутствуют в любом алгоритме, вычисляющем функцию на последовательности. Рассмотрим пример функции на последовательности.

Дан многочлен $p(x) = a_0x^n + a_1x^{n-1} + ... + a_n$.

Нужно вычислить значение многочлена в точке x = t.

Алгоритм, основанный на просмотре последовательности коэффициентов в направлении от старшего к младшему, не использующий возведение в степень явно, называется схемой Горнера.

Возьмем многочлен третьей степени: $p(x) = ax^3 + bx^2 + cx + d$.

Его по схеме Горнера можно представить как $p(x) = ((ax + b)x + c)x + d$.

Для вычисления значения многочлена достаточно трех умножений и трех сложений.

В общем случае многочлен представляется по схеме Горнера в следующем виде:

$$p(x) = (...((a_0x + a_1)x + a_2)x + ... + a_{n-1})x + a_n.$$

Обозначим через $p_k(x)$ многочлен k-й степени, вычисленный по коэффициентам $a_0, a_1, ..., a_k$:

$$p_k(x) = a_0x^k + a_1x^{k-1} + ... + a_k.$$

Тогда $p_{k+1}(x) = p_k(x)x + a_{k+1}$, т. е. при считывании нового коэффициента многочлена надо старое значение многочлена умножить на значение x, а затем прибавить к нему новый коэффициент.

Варианты заданий. Варианты заданий приведены в таблице 3. В качестве N берется Ваш номер в журнале, его нужно разделить с остатком на 3, M = N%3.

Таблица 3 – Задания по вариантам

Значение M	Задание
0	1, 2
1	1, 3
2	2, 3

Задание

Требуется разработать алгоритм по схеме Горнера согласно варианту, составить его блок-схему, реализовать свой алгоритм в программной среде.

Вариант содержит два метода, которые должны вычислять один и тот же многочлен в конкретной точке. Данные должны задаваться один раз.

Выполнить анализ алгоритма: вычислить его трудоемкость теоретически и проверить ее экспериментально, введя в код программы счетчик элементарных операций. Сделать выводы о соответствии значений нескольких пар чисел теоретической и полученной экспериментально трудоемкости для разных входных значений.

1 Разработать алгоритм схемы Горнера с использованием арифметического цикла.

2 Используя арифметический цикл с отрицательным шагом, разработать алгоритм схемы Горнера в случае, когда коэффициенты многочлена заданы по возрастанию, а не по убыванию степеней.

3 Выполнить алгоритм схемы Горнера с использованием цикла ПОКА.

3 Лабораторная работа № 3. Сравнение эффективности разных видов сортировки

Цель работы: сформировать знания и умения организации сортировки с выполнением анализа алгоритма с учетом рассмотрения лучшего, среднего и худшего случаев.

Теоретические сведения

Будем считать, что сортируемые объекты являются записями, содержащими одно или несколько полей. Одно из полей, называемое ключом, имеет такой тип данных, что на нем определено отношение линейного порядка «<» или «≤».

Задача сортировки заключается в упорядочивании последовательности записей таким образом, чтобы значения ключевого поля составляли неубывающую последовательность. Другими словами, записи r_1, r_2, …, r_n со значениями ключей k_1, k_2, ..., k_n соответственно надо расположить в порядке r_{i1}, r_{i2}, ..., r_{in}, таком, что $k_{i1} \le k_{i2} \le \ldots \le k_{in}$. В общем случае необязательно, чтобы все записи были различными, и если есть записи с одинаковыми значениями ключей, то в упорядоченной последовательности они располагаются рядом друг с другом в любом порядке.

Будем использовать различные критерии оценки алгоритмов внутренней сортировки. Первой и наиболее общей мерой времени выполнения является количество шагов алгоритма, необходимых для упорядочивания n записей.

Другой общей мерой служит количество сравнений между значениями ключей, выполняемых при сортировке списка из n записей. Эта мера особенно информативна, когда ключи являются строками символов, и поэтому самым «трудоемким» оператором будет оператор сравнения ключей. Если размер записей большой, то следует также учитывать время, необходимое на их перемещение.

Простые схемы сортировки.

Рассмотрим несколько простых методов, называемых прямыми, где требуется порядка n^2 сравнений элементов.

Сортировка вставками Insertion sort (***с помощью прямого включения***).

Первый прямой метод, который будем рассматривать, называется сортировкой **вставками**, т. к. на i-м этапе i-й элемент A[i] вставляется в нужную позицию среди элементов A[1], A[2], ..., A[i – 1], которые уже упорядочены. После этой вставки первые i элементов будут упорядочены.

При сортировке элементов массива вставками массив делят на две части: отсортированную – a_1, …, a_{i-1} и «исходную» – a_i, …, a_n.

В начале работы в качестве отсортированной части берем только один первый элемент, а в качестве неотсортированной – все остальные элементы. На каждом шаге, начиная с i = 2, из неотсортированной последовательности извлекается i-й элемент и вставляется в отсортированную так, чтобы не нарушить в ней упорядоченности элементов.

Каждый шаг алгоритма включает четыре действия.

1 Взять i-й элемент массива (он же является первым элементом в неотсортированной части) и сохранить его в дополнительной переменной.

2 Найти позицию j в отсортированной части массива, куда будет вставлен i-й элемент, значение которого теперь хранится в дополнительной переменной. Вставка этого элемента не должна нарушить упорядоченности элементов отсортированной части массива.

3 Сдвинуть элементы массива с i – 1 позиции по j-ю на один элемент вправо, чтобы освободить найденную позицию для вставки.

4 Вставить взятый элемент в найденную j-ю позицию.

Сказанное можно записать в виде псевдопрограммы:

```
for i:= 2 to n do
переместить A[i] на позицию j ≤ i такую, что
A[i] < A[k] для j < k < i  и
либо A[i] > A[j – 1], либо j = 1
```

Чтобы сделать процесс перемещения элемента A[i] более простым, можно ввести элемент A[0], чье значение ключа будет меньше значения ключа любого элемента A[1], ..., A[n]. Если такую константу нельзя применить, то при вставке A[i] в позицию j – 1 надо проверить, не будет ли j = 1, если нет, тогда сравнивать элемент A[i] (который сейчас находится в позиции j) с элементом A[j –1]. Описанный алгоритм показан в листинге 1. Стандартная процедура swap используется для перестановки элементов местами.

Листинг 1. Сортировка вставками

```
(1) A[0].key := -∞;
(2)  for i:= 2 to n do begin
(3)    j:= i;
(4)  while A[j] < A[j – 1] do begin
(5)    swap(A[j], A[j – 1]);
(6)    j:= j – 1
(7)  end;
(8)  end.
```

Число сравнений C_i при i-м проходе в худшем случае равно i – 1, в лучшем – единице. Если считать, что все перестановки из n элементов равновероятны, то среднее число сравнений i/2.

Число перестановок $M_i = C_i + 2$.

Общее число сравнений и число перестановок таковы:

$C_{min} = n - 1$; $M_{min} = 3(n - 1)$;

$C_{ave} = (n^2 + n - 2)/4$; $M_{ave} = (n^2 + 9n - 10)/4$;

$C_{max} = (n^2 + n - 4)/4$; $M_{max} = (n^2 + 3n - 4)/2$.

Данный алгоритм показывает наилучшие результаты работы в случае уже упорядоченной исходной части массива, наихудшие – когда элементы первоначально расположены в обратном порядке. Он эффективен на небольших наборах данных, на наборах данных до десятков элементов может оказаться лучшим; также он эффективен на наборах данных, которые уже частично отсортированы.

Это устойчивый алгоритм сортировки (не меняет взаимного расположения равных элементов). Если элемент состоит только из одного ключа (т. е. значения, по которому и производится сортировка), то устойчивость сортировки не важна, но при сортировке записей (например, с полями Фамилия и Имя) в поддержании исходного порядка может быть смысл, т. к. равные по ключу элементы различны.

Устойчивыми алгоритмами сортировки также являются пузырьковая, слиянием.

Неустойчивые алгоритмы сортировки: быстрая сортировка Хоара, пирамидальная, Шелла.

Сортировка выбором имеет как устойчивые, так и неустойчивые реализации.

Устойчивость сортировки всегда может быть достигнута путем удлинения исходных ключей, если включить в них информацию о первоначальном порядке значений.

Пример 1 – Пусть даны список названий и годы знаменитых извержений вулканов (рисунок 1). Они как записи с ключевым полем **Название** будут подлежать сортировке по алфавиту.

Название	**Год**
Пили	1902
Этна	1669
Кракатау	1883
Агунг	1963
Св. Елена	1980
Везувий	79

Рисунок 1 – Список для сортировки

Для этого примера мы применим следующее объявление типов данных:

```
type
keytype = array[1..10] of char;
recordtype = record
key: keytype; { название вулкана }
year: integer { год извержения }
 end;
```

На рисунке 2 показаны последовательные этапы алгоритма вставкой для i = 2, 3, …, 6. После каждого этапа алгоритма сортировки вставками элементы, расположенные выше линии, уже упорядочены, хотя между ними на последующих этапах могут быть вставлены элементы, которые сейчас находятся ниже линии.

-∞	-∞	-∞	-∞	-∞	-∞
Пили	Пили	Кракатау	Агунг	Агунг	Агунг
Этна	Этна	Пили	Кракатау	Кракатау	Везувий
Кракатау	Кракатау	Этна	Пили	Пили	Кракатау
Агунг	Агунг	Агунг	Этна	Св. Елена	Пили
Св. Елена	Св. Елена	Св. Елена	Св. Елена	Этна	Св. Елена
Везувий	Везувий	Везувий	Везувий	Везувий	Этна
Начало	**i = 2**	**i = 3**	**i = 4**	**i = 5**	**i = 6**

Рисунок 2 – Этапы сортировки методом вставки

***Сортировка выбором* (*Selection sort*).**

При сортировке с помощью **прямого выбора** массив также делится на две части: отсортированную или «готовую» последовательность – a_1, …, a_{i-1} и «исходную» – a_i, …, a_n. Происходит поиск одного элемента из исходной последовательности, который обладает наименьшим (наибольшим) значением ключа, и уже найденный элемент помещается в конец готовой последовательности.

На момент начала сортировки методом прямого выбора готовая последовательность считается пустой, соответственно, исходная последовательность включает в себя все элементы массива.

Алгоритм сортировки с помощью прямого выбора.

1 Из всего массива выбирается элемент с наименьшим значением ключа.

2 Он меняется местами с первым элементом a_1.

3 Затем этот процесс повторяется с оставшимися n – 1 элементами, n – 2 элементами и т. д. до тех пор, пока не останется один элемент с наибольшим значением.

На i-м этапе сортировки **выбором** выбирается запись с наименьшим ключом среди записей A[i], ..., A[n] и меняется местами с записью A[i]. В результате после i-го этапа все записи A[1], ..., A[i] будут упорядочены.

Сортировку посредством выбора можно описать следующим образом:

```
for i:= 1 to n - 1 do
выбрать среди A[i], . .., A[n] элемент с наименьшим ключом и поменять его местами с A[i];
```

Листинг 2. Сортировка посредством выбора

```
1  var lowkey: keytype; {текущий наименьший ключ, найденный при проходе по элементам A[i], … A[n]}
2  lowindex: integer; {позиция элемента с ключом lowkey}
3  begin
4    for i:= 1 to n - 1 do begin
5      lowindex: = i;
6      lowkey:= A[i].key;
7      for j:= i + 1 to n do begin
8      {сравнение ключей с текущим ключом lowkey}
9        if A[j].key < lowkey then begin
10         lowkey:= A[j].key;
11         lowindex:= j
12       end;
13     swap(A[i], A[lowindex])
14     end;
15   end;
16 end.
```

Пример 2 – На рисунке 3 показаны этапы сортировки посредством выбора для списка из рисунка 1. Например, на первом этапе значение lowindex равно 4, т. е. позиции Агунга, который меняется с Пили, элементом A[1].

-∞	-∞	-∞	-∞	-∞	-∞
Пили	Пили	Кракатау	Агунг	Агунг	Агунг
Этна	Этна	Пили	Кракатау	Кракатау	Везувий
Кракатау	Кракатау	Этна	Пили	Пили	Кракатау
Агунг	Агунг	Агунг	Этна	Св. Елена	Пили
Св. Елена	Св. Елена	Св. Елена	Св. Елена	Этна	Св. Елена
Везувий	Везувий	Везувий	Везувий	Везувий	Этна
Начало	**1-й этап**	**2-й этап**	**3-й этап**	**4-й этап**	**5-й этап**

Рисунок 3 – Этапы сортировки методом выбора

Линии на рисунке показывают, что элементы, расположенные выше, имеют наименьшие значения ключей и уже упорядочены.

После (n – 1)-го этапа элемент A[n] также стоит на правильном месте, т. к. выше его все записи имеют меньшие значения ключей.

При работе данного алгоритма число сравнений элементов C не зависит от начального порядка элементов $C = (n^2 – n)/2$.

Лучший случай: $M_{min} = 3(n – 1)$.

Худший случай: $M_{max} = n^2/4 + 3(n – 1)$.

В общем случае алгоритм с прямым выбором предпочтительнее алгоритма прямого включения. Однако если элементы вначале упорядочены (почти упорядочены), алгоритм с прямым включением выполнит сортировку быстрее.

Сортировка «пузырьком».

Самым простым методом сортировки является **метод «пузырька».** Он относится к методам прямого обмена, как и шейкерная сортировка.

Чтобы описать основную идею этого метода, представим, что записи, подлежащие сортировке, хранятся в массиве, расположенном вертикально. Записи с малыми значениями ключевого поля более «легкие» и «всплывают» вверх наподобие пузырька.

При первом проходе вдоль массива, начиная проход снизу, берется первая запись массива и ее ключ поочередно сравнивается с ключами последующих записей. Если встречается запись с более «тяжелым» ключом, то эти записи меняются местами. При встрече с записью с более «легким» ключом эта запись становится эталоном для сравнения, и все последующие записи сравниваются с этим новым ключом. В результате запись с наименьшим значением ключа окажется в самом верху массива.

Во время второго прохода вдоль массива находится запись со вторым по величине ключом, которая помещается под записью, найденной при первом проходе массива, т. е. на вторую сверху позицию, и т. д.

Отметим, что во время второго и последующих проходов вдоль массива нет необходимости просматривать записи, найденные за предыдущие проходы, т. к. они имеют ключи, меньшие, чем у оставшихся записей. Другими словами,

во время i-го прохода не проверяются записи, стоящие на позициях выше i. В листинге 3 приведен описываемый алгоритм, в котором через A обозначен массив из n записей (тип данных recordtype). Здесь и далее предполагаем, что одно из полей записей называется key (ключ) и содержит значения ключей.

Листинг 3. Алгоритм «пузырька»

```
(1) for i:= 1 to n – 1 do begin
(2)    for j:= 2 to i + 1 do begin
(3)       if A[j].key < A[j – 1].key then
(4)         swap(A[j], A[j – 1]);
(5)    end;
(6) end.
```

Пример 3 – Применим алгоритм «пузырька» для упорядочивания списка вулканов в алфавитном порядке их названий, считая, что отношением линейного порядка в данном случае является отношение лексикографического порядка над полем ключей. На рисунке 4 показано пять проходов алгоритма. Линии указывают позицию, выше которой записи уже упорядочены. После пятого прохода все записи, кроме последней, стоят на нужных местах, но последняя запись не случайно оказалась последней – она также уже стоит на нужном месте. Поэтому сортировка заканчивается.

-∞	-∞	-∞	-∞	-∞	-∞
Пили	Пили	Кракатау	Агунг	Агунг	Агунг
Этна	Этна	Пили	Кракатау	Кракатау	Везувий
Кракатау	Кракатау	Этна	Пили	Пили	Кракатау
Агунг	Агунг	Агунг	Этна	Св. Елена	Пили
Св. Елена	Св. Елена	Св. Елена	Св. Елена	Этна	Св. Елена
Везувий	Везувий	Везувий	Везувий	Везувий	Этна
Начало	**1-й проход**	**2-й проход**	**3-й проход**	**4-й проход**	**5-й проход**

Рисунок 4 – Этапы сортировки пузырьком

На первом проходе Везувий и Св. Елена меняются местами, но далее Везувий не может поменяться местами с Агунгом. Агунг, поочередно меняясь местами с Кракатау, Этной и Пили, поднимается на самый верх. На втором этапе Везувий поднимается вверх и занимает вторую позицию. На третьем этапе это же проделывает Кракатау, а на четвертом – Этна и Св. Елена меняются местами, Пили стоит на нужном месте.

Все названия вулканов расположились в алфавитном порядке, сортировка закончилась. Пятый этап также выполнен в соответствии с алгоритмом листинга 1, но он уже не меняет порядок записей.

Алгоритм прямого обмена основывается на сравнении и перестановке пары соседних элементов и продолжении этого процесса до тех пор, пока не будут упорядочены все элементы.

Обмен местами двух элементов представляет собой характерную особенность данного метода, хотя в предыдущих методах переставляемые элементы также обменивались местами.

Число сравнений в алгоритме сортировки «пузырьком» $C = (n^2 - n)/2$, а минимальное и максимальное число перестановок элементов равны

$$M_{min} = 0,$$

$$M_{max} = 3(n^2 - n)/4.$$

Шейкерная сортировка.

Пример пузырьковой сортировки отражает асимметрию работы алгоритма: легкий «пузырек» всплывает сразу – за один проход, а тяжелый тонет очень медленно – за один проход на одну позицию.

Так, массив **15 29 31 55 70 93 8** с помощью сортировки пузырьком будет упорядочен за один проход, а для массива **93 8 15 29 31 55 70** потребуется шесть проходов. Это наводит на мысль о следующем улучшении: чередовать направление просмотра на каждом последующем проходе (рисунок 5).

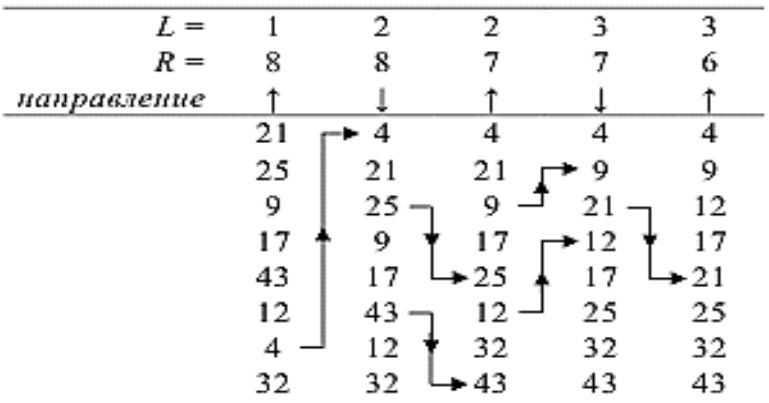


Рисунок 5 – Этапы сортировки методом вставки

Алгоритм, реализующий такой подход, называется шейкерной сортировкой. Переменные L и R содержат индексы элементов, до которых должен происходить просмотр при движении влево (или вверх) и вправо (или вниз) соответственно.

Анализ шейкерной сортировки довольно сложен. Шейкерная сортировка с успехом используется лишь в тех случаях, когда известно, что элементы почти упорядочены, что на практике бывает весьма редко. Анализ показывает, что «обменная» сортировка и ее усовершенствования фактически оказываются хуже сортировок с помощью включений и с помощью выбора

Сортировка Шелла.

Все методы прямой сортировки фактически передвигают каждый элемент на всяком элементарном шаге на одну позицию. Поэтому они требуют порядка n^2 таких шагов. Следовательно, в основу любых улучшений алгоритмов сортировки

должен быть положен принцип перемещения на каждом такте элементов на большие расстояния. Среднее расстояние, на которое должен продвигаться каждый из n элементов во время сортировки, равно n/3 позиций. Это число является целью в поиске более эффективных методов сортировки.

В 1959 г. Д. Шеллом было предложено усовершенствование алгоритма сортировки вставками. Рассмотрим его работу на примере следующего массива:

35 28 49 79 45 11 89 70 91 67 54 19 13 24.

Сначала отдельно группируются элементы, отстоящие друг от друга на расстоянии 4. Таких групп будет четыре, они показаны на рисунке 6. Элементы, принадлежащие одной группе, объединены дугами.

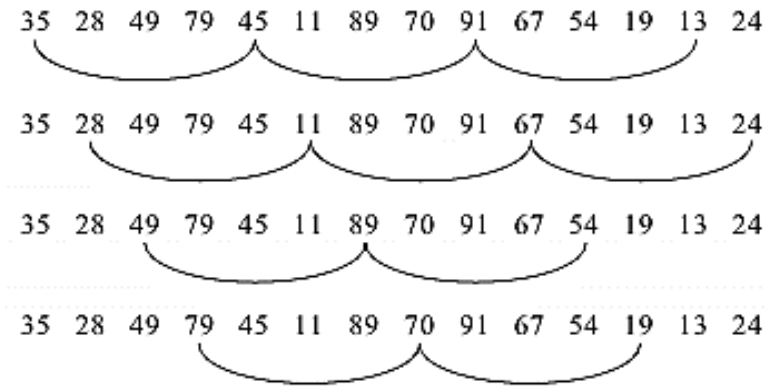


Рисунок 6 – Группы сортируемых элементов

Следующим шагом выполняется сортировка внутри каждой из групп методом прямого включения. Сначала сортируются элементы **35, 45, 91, 13**, затем **28, 11, 67, 24**, следом **49, 89, 54** и, наконец, **79, 70, 19**. В результате получаем массив **13 11 49 19 35 24 54 70 45 28 89 79 91 67.**

Такой процесс называется четвертной сортировкой.

Следующим проходом элементы группируются так, что теперь в одну группу входят элементы, отстоящие друг от друг на две позиции (рисунок 7). Таких групп две:

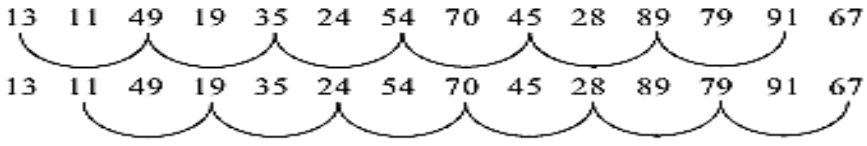


Рисунок 7 – Группы сортируемых элементов через две позиции

Вновь в каждой группе выполняется сортировка с помощью прямого включения. Это называется двойной сортировкой, ее результатом будет массив:

13 13 11 35 19 45 24 49 28 54 67 89 70 91 79.

И, наконец, на третьем проходе (рисунок 8) идет обычная или одинарная сортировка с помощью прямого включения.

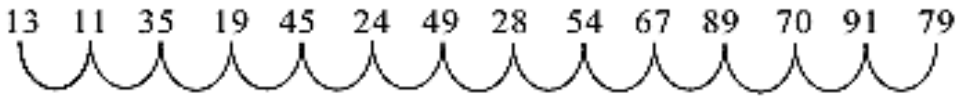


Рисунок 8 – Группы сортируемых элементов через одну позицию

Результатом работы алгоритма сортировки Шелла является отсортированный массив **11 13 19 24 28 35 45 49 54 67 70 79 89 91.**

Расстояния в группах можно уменьшать по-разному, лишь бы последнее было единичным, ведь в самом плохом случае последний проход и сделает всю работу. Однако совсем не очевидно, что такой прием «уменьшающихся расстояний» может дать лучшие результаты, если расстояния не будут степенями двойки.

При выполнении лабораторной работы рекомендуется начальный шаг взять равным $h_{нач} = n/3$, где n – количество элементов массива, и уменьшать его на каждом проходе вдвое: $h_{i-1} = h_i/2$, $h_{конеч} = 1$.

Математический анализ показывает, что для сортировки n элементов методом Шелла затраты пропорциональны $\mathbf{n^{1.2}}$, что значительно лучше $\mathbf{n^2}$, необходимых для методов прямой сортировки.

Сортировка расческой.

Сортировка расческой улучшает сортировку пузырьком, и конкурирует с алгоритмами, подобными быстрой сортировке. Основная идея – устранить маленькие значения в конце списка, которые крайне замедляют сортировку пузырьком (интересно, что большие значения в начале списка не представляют проблемы для сортировки пузырьком).

В сортировке пузырьком, когда сравниваются два элемента, расстояние между ними (разность индексов) равно 1. Основная идея сортировки расческой в том, что этот промежуток может быть гораздо больше, чем единица.

Сначала расстояние между элементами максимально, и равно размеру массива минус один. Затем, пройдя массив с этим шагом, необходимо поделить шаг на фактор уменьшения (его оптимальное значение 1,247) и пройти по списку вновь. Так продолжается до тех пор, пока разность индексов не достигнет единицы. В этом случае сравниваются соседние элементы, как и в сортировке пузырьком, но такая итерация будет только одна.

Эта сортировка является неустойчивой, т. е. она может менять порядок одинаковых элементов.

Такой подход позволяет снизить вычислительную сложность алгоритма: худшее время $O(n^2)$, лучшее время $\Omega(n \log n)$.

Сравнение различных алгоритмов сортировки.

Эффективность различных алгоритмов сортировки массивов, состоящих из n сортируемых элементов, проведем по двум критериям (таблица 1): числу необходимых сравнений элементов C и числу перестановок элементов M.

Таблица 4 – Количество сравнений и перестановок основных видов сортировок

Метод	Количество сравнений C и перестановок M		
	минимальное	среднее	максимальное
Прямое включение	$C_{min} = n - 1$ $M_{min} = 3(n - 1)$	$C_{ave} = (n^2 + n - 2)/ 4$ $M_{ave} = (n^2 + 9n - 10)/4$	$C_{max} = (n^2 + n - 4)/4$ $M_{max} = (n^2 + 3n - 4)/2$
Прямой выбор	$C_{min} = (n^2 - n)/2$ $M_{min} = 3(n - 1)$	$C_{ave} = (n^2 - n)/2$ $M_{ave} = n\cdot(\ln n + 0{,}57)$	$C_{max} = (n^2 - n)/2$ $M_{max} = n^2/4 + 3(n - 1)$
Прямой обмен	$C_{min} = (n^2 - n)/2$ $M_{min} = 0$	$C_{ave} = (n^2 - n)/2$ $M_{ave} = 0{,}75 \cdot (n^2 - n)$	$C_{max} = (n^2 - n)/2$ $M_{max} = 1{,}5 \cdot (n^2 - n)$

Для усовершенствованных методов нет простых и точных формул. Существенно, что в случае сортировки Шелла вычислительные затраты составляют $c \cdot n^{1.2}$, в то время как для прямых методов – $c \cdot n^2$. Очевидно преимущество сортировки Шелла как улучшенного метода по сравнению с прямыми методами сортировки.

Пузырьковая сортировка определенно наихудшая из всех сравниваемых. Ее усовершенствованная версия – шейкерная сортировка – продолжает оставаться плохой по сравнению с прямым включением и прямым выбором (за исключением уже упорядоченного массива).

Варианты заданий. Варианты заданий приведены в таблице 5. В качестве личного номера N берется Ваш номер в журнале.

Таблица 5 – Задания, соответствующие личному номеру

Номер	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Задание	а, б	а, в	а, г	а, д	а, е	а, ж	б, в	б, г	б, д	б, е	б, ж	в, г	в, д	в, е	в, ж

Продолжение таблицы 5

Номер	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Задание	а, б	а, в	а, г	а, д	а, е	а, ж	б, в	б, г	б, д	б, е	б, ж	в, г	в, д	в, е	в, ж

Задание

Реализовать в программе два алгоритма сортировки из указанных ниже согласно варианту и сравнить эффективность реализованных алгоритмов по их трудоемкости, числу перестановок и сравнений на различных входах (20, 50, 100, 400, 1000), а также сравнить в программе время выполнения каждого из методов сортировки:

а) сортировка с помощью прямого включения;

б) сортировка с помощью прямого выбора при помощи поиска минимального элемента;

в) сортировка с помощью прямого выбора при помощи поиска одновременно минимального и максимального элементов;

г) сортировка «пузырьком»;

д) шейкерная сортировка;

е) сортировка расческой;

ж) сортировка Шелла.

4 Лабораторная работа № 4. Эвристические алгоритмы решения NP-полных задач

Цель работы: экспериментально определить среднее время выполнения обобщенной элементарной операции в языке высокого уровня и получить прогноз времени выполнения программы для больших размерностей множества исходных данных (в зависимости от типов данных).

Теоретические сведения

Анализ трудоемкости алгоритмов позволяет получить функциональную зависимость между параметрами исходной задачи (мощностью множества исходных данных) и количеством выполненных элементарных операций в заданной среде реализации.

При этом переход от функции трудоемкости к времени выполнения программы для алгоритма с известной трудоемкостью зависит от времени выполнения элементарных операций в среде языка реализации алгоритма и специфики реализации алгоритма (квалификация программиста, структуры данных и др.) на данном языке программирования.

Имея аналитическую оценку трудоемкости данного алгоритма fa(N), ее экспериментальное подтверждение, зная среднее время выполнения обобщенной элементарной операции для данного языка и данного процессора t_cp, можно получить оценку времени выполнения программы:

$$ta(N) = t\_cp * fa(N). \quad (3)$$

Для получения средневзвешенной оценки t_cp необходимо получить значения ta(N) для различных значений N, что определяет следующий алгоритм нахождения среднего времени выполнения элементарной операции:

1) измерение времени ta(N) для различных значений N;

2) экспериментальное или теоретическое получение функции трудоемкости алгоритма fa(N);

3) определение t_cp = ta(N)/f(N) для каждого значения N;

4) усреднение полученных значений t_cp по количеству испытаний.

Эта методика дает возможность провести ряд дополнительных исследований и определить зависимость t_cp от типов данных. Полученные значения t_cp для различных типов данных позволяют при сравнительном анализе выявить качественные различия процессоров при обработке данных различного типа, что может являться основой одного из процессорных тестов.

Пересчет полученных значений t_cp в такты процессора дает возможность провести сравнительный анализ относительной скорости выполнения элементарных операций для различных типов данных у процессоров с различной тактовой частотой и различной внутренней архитектурой.

Таким образом, можно обнаружить преимущества той или иной архитектуры по отношению к данной задаче, т. е. решить проблему выбора наиболее рационального процессора по отношению к данной задаче.

Если тестовый алгоритм обладает достаточной устойчивостью по t_ср (значения t_ср, полученные для различных N, имеют малую дисперсию), то можно прогнозировать поведение программы для больших размерностей исходных данных, что позволяет определить границы применимости данного алгоритма на данном процессоре.

Реализуем программно задачу о сумме, определяющую время своего выполнения.

Задача о сумме формулируется как задача нахождения таких элементов S_j исходного массива S из N чисел, что $\sum S_j = V$.

Задача относится к классу NPC, условия существования решения

$$\text{Min } \{S[i], i = 1, N\} \le V \le \text{Sum}.$$

Поскольку исходный массив содержит N чисел, то проверке на равенство V перебором подлежат следующие варианты решений:

1) V содержит одно слагаемое $\Rightarrow C_N^1 = N$ вариантов;

2) V содержит два слагаемых $\Rightarrow C_N^2 = (N \cdot (N - 1))/(1 \cdot 2)$ вариантов;

3) V содержит три слагаемых $\Rightarrow C_N^3 = N \cdot (N - 1) \cdot (N - 2)/(1 \cdot 2 \cdot 3)$ вариантов и т. д. до проверки одного варианта с N слагаемыми.

Поскольку сумма биномиальных коэффициентов для степени N равна $(1+1)^N = \sum C_N^k = 2^N$ и для каждого варианта необходимо выполнить суммирование (с оценкой O(N)) для проверки на V, то оценка сложности алгоритма в худшем случае имеет вид $F^\wedge_A(N,V) = O(N \cdot 2^N)$.

Определим вспомогательный массив, хранящий текущее сочетание исходных чисел в массиве S, подлежащих проверке на V – массив Cnt[j]. Элемент массива равен 0, если число S[j] не входит в V, и равен 1, если число S[j] входит в V. Решение получено, если $V = \sum S[j] \cdot Cnt[j]$.

Задание

1 Подберите значение N_0, при котором время выполнения кода составляет порядка 0,5 нс на данном процессоре для целого типа данных.

2 Для $N_0 + h$, … $N_0 + 5h$ выполните замеры времени выполнения, полученные экспериментальные данные занесите в расчетную таблицу (шаг h определяется экспериментально как наименьший, дающий относительно стабильную разницу во времени).

3 Измените тип данных на длинное целое и проведите шесть замеров времени для тех же значений N_0, …, $N_0 + 5h$.

4 По известной трудоемкости и полученному времени выполнения программы рассчитайте время выполнения обобщенной элементарной операции t_ср в нс и, зная тактовую частоту процессора, t_ср в тактах процессора.

5 На основе полученных данных рассчитайте средневзвешенное значение t_ср по шести испытаниям и построить прогноз времени выполнения для значений $N \in [N_0 + 6h; N_0 + 9h]$.

Листинг 4. Программа определения времени выполнения процедуры решения задачи о сумме

```
PROGRAM TaskSum;
Uses Dos,Crt;
Type
    fp = Extended;
    Value = Word {Extended} {Longint};
    Loop = Word;
    Vector = Array [0..100] of Value;
Var
   Number,Count    : Vector;
   N               : Loop;
   V               : Value;
   Flag            : Boolean;
   Tbegin,Tend,Tfm :fp;
   Function Time : Fp;
   Var
      t          : fp;
      h,m,s,s100 : Word;
   Begin
      GetTime(h,m,s,s100);
      t:=h*3600.0;
      t:=t+(m*60.0);
      t:=t+(s*1.0);
      t:=t+( (s100*1.0)/100.0 );
      Time:=t;
   End;
Procedure SetNumber;
   Var
       i : Loop;
   Begin
      ClrScr;
      Write('N=');
      Readln(N);
      Write('V=');
      Readln(V);
      For i:=1 to N do
       begin
          Number[i]:=Random(100)+50;
          Writeln(Number[i]:5);
       end;
   End;
Procedure FindSum(N: Loop;
V: Value; Var S,Cnt: Vector;
Var Flag: Boolean);
   Var
      i,j : Loop;
      Sum : Value;
   Begin
      Flag:=False;
      i:=1;
```

```
      Repeat
       begin
       Cnt[i]:=0; Вспомогательный массив коэффициентов при Sj
       для проверки суммы
       i:=i+1;
       end;
      Until i>N;
      Cnt[N]:=1;
      Repeat
       begin
          sum:=0;
          i:=1;
          Repeat
           begin
              sum:=sum + Cnt[i]*S[i];
              i:=i+1;
           end;
          Until i>N;
          If sum = V
            then
             begin
                Flag:=True;
                Write('OK');
                Readln;
                Halt;
              end;
           j:=n;
           While Cnt[j]=1 do
            begin
               Cnt[j]:=0;
               j:=j-1;
            end;
           Cnt[j]:=1;
        end;
    Until Cnt[0] = 1;
  End;

BEGIN
   Randomize;
   SetNumber;
   Tbegin:=Time;
   FindSum(N,V,Number,Count,Flag);
   Tend:= Time;
   Tfm := Tend-Tbegin;
   Writeln('ta(',N:2,')=',Tfm:8:2,' sec.');
   Readln;
END.
```

Обработка результатов экспериментов и построение прогноза результатов выполняются в MS Excel (рисунок 9).

Функция f(n) подбирается исследователем путем замены коэффициентов так, чтобы значения функции примерно соответствовали полученным экспериментально. Общий вид функции, задающий степень ее роста, будет иметь примерно такой же вид.

Вводится тактовая частота процессора, на котором проводился эксперимент, и на основе полученных данных выполняется задание 4 по расчету времени выполнения обобщенной элементарной операции.

C17 =8*A17*B17+16*B17-3*A17-12

	A	B	C	D	E	F	G	H
1	f(n)= 8*n*2^n + 16*2^n - 3*n - 12				Процессор		2200	МГц
2								
3		Экспериментальные данные						
4	n	2^n	f(n) теория	f(n) эксперимент	TYPE = WORD	Time -s	top - ns	top - takt
5	16	65 536	9 437 124	9 437 138		0,22	23,31	4,66
6	17	131 072	19 922 881	19 922 988		0,44	22,09	4,42
7	18	262 144	41 942 974	41 942 942		0,93	22,17	4,43
8	19	524 288	88 080 315	88 080 349		1,98	22,48	4,5
9	20	1 048 576	184 549 304	184 549 273		4,17	22,60	4,52
10	21	2 097 152	385 875 893	385 875 854		8,62	22,34	4,47
11	22	4 194 304	805 306 290	805 306 226		18,02	22,38	4,48
12						Средние значения	**22,48**	**4,50**
13								
14	Прогнозируемые результаты							
15	n	2^n	f(n) теория	f(n) эксперимент	Прогноз	Прогноз		
16					h:m:s	sec		
17	23	8 388 608	1 677 721 519		0:00:38	37,72		
18	24	16 777 216	3 489 660 844		0:01:18	78,45		
19	25	33 554 432	7 247 757 225		0:02:43	162,93		
20	26	67 108 864	15 032 385 446		0:05:38	337,93		
21	27	134 217 728	31 138 512 803		0:11:40	700		t эксп
22	28	268 435 456	64 424 509 344		0:24:08	1448,3		37,62
23	29	536 870 912	133 143 986 077		0:49:53	2993,1		
24	30	1 073 741 824	274 877 906 842		1:42:59	6179,3		delta %
25	31	2 147 483 648	566 935 682 967		3:32:25	12745		0,25%
26								

Рисунок 9 – Обработка результатов экспериментов и построение прогноза результатов

Контрольные вопросы

1 Как оценивается алгоритм задачи о сумме в худшем случае?

2 Опишите идею реализации полного перебора вариантов достижения суммы.

3 Поясните содержание ячейки A1.

4 Будет ли у Вас содержание ячейки A1 таким же?

5 Объясните смысл данных, хранящихся в столбце B.

6 Как заполняются столбцы C и D?

7 Поясните смысл числовых значений в столбцах F, G, H.

8 Как выполнено построение прогноза, вручную или с использованием встроенных средств?

9 Какой смысл имеет величина delta?

10 Какой тип данных в C# соответствует типу WORD в языке Pascal?

11 Аналогично, какой тип данных в C# соответствует типу LONG в языке Pascal?

12 Сравните результаты, полученные при работе программы с двумя типами данных.

Список литературы

1 **Авдошин, С. М.** Дискретная математика. Формально-логические системы и языки / С. М. Авдошин, А. А. Набебин. – Москва : ДМК Пресс, 2018. – 390 с.

2 **Игошин, В. И.** Сборник задач по математической логике и теории алгоритмов : учебное пособие / В. И. Игошин. – Москва : КУРС; ИНФРА-М, 2019. – 392 с.

3 **Игошин, В. И.** Математическая логика : учебное пособие / В. И. Игошин. – Москва : ИНФРА-М, 2020. – 399 с.

4 **Гуц, А. К.** Математическая логика и теория алгоритмов / А. К. Гуц. – 3-е изд., испр. – Москва : URSS; Либроком, 2016. – 117 с.

5 **Таран, Т. А.** Сборник задач по дискретной математике / Т. А. Таран, Н. А. Мыценко, Е. Л. Темникова. – 2-е изд., перераб. и доп. – Киев: Инрес, 2005. – 64 с.

6 **Светлов, В. А.** Логика : учебное пособие / В. А. Светлов. – Москва : Логос, 2020. – 432 с.

Приложение А
(обязательное)

Общие требования к содержанию отчета

Отчет по лабораторной работе № 1 принимается в письменном виде (в тетради по дисциплине), к лабораторным работам № 2–4 – в печатном.

На титульном листе отчета указываются название учреждения образования, название закрепленной за дисциплиной кафедры, название дисциплины; необходимые данные о студенте: ФИО студента, группа; должность, ФИО преподавателя.

Печатный отчет должен содержать стандартные составные части.

1 Титульный лист (рисунок А.1) с указанием следующих реквизитов: название учреждения образования, название закрепленной за дисциплиной кафедры, номер и название лабораторной работы, название дисциплины, вариант, кто выполнил лабораторную работу (ФИО студента, группа), кто проверяет работу (должность, ФИО преподавателя), место и дата составления отчета.

2 Цель работы и постановка задачи.

3 Выполненное задание согласно варианту (код программы, реализующей данный алгоритм, с необходимыми комментариями).

4 Скриншоты с входными и выходными данными. Обычно программа тестируется на нескольких вариантах входных данных для проверки ее корректности.

5 Выводы по теме лабораторной работы.

Отчет оформляется шрифтом гарнитуры Times New Roman, кегль 12 пт, межстрочный интервал – полуторный, абзацный отступ – 1,25 см.

Страницы должны быть пронумерованы вверху посередине. Титульный лист при нумерации считается, но не нумеруется.

МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Автоматизированные системы управления»

ПОСТРОЕНИЕ И АНАЛИЗ АЛГОРИТМОВ

Лабораторная работа № 2
Построение и анализ алгоритма
вычисления значения полиномиальной функции

Выполнил:
Светлов Владислав Сергеевич,
студент группы АСОИ-221

Проверила:
старший преподаватель
Беккер Инга Александровна

Могилев, 2022

Рисунок А.1 – Титульный лист отчета