ТЕОРИЯ АЛГОРИТМОВ

1 ОСНОВЫ АЛГОРИТМИЗАЦИИ

1.1 АЛГОРИТМЫ И ВЕЛИЧИНЫ

Слово "алгоритм" произошло от латинской формы имени величайшего среднеазиатского математика Мухаммеда ибн Муса аль-Хорезми (Alhorithmi), жившего в 783—850 гг. В своей книге "Об индийском счете" он изложил правила записи натуральных чисел с помощью арабских цифр и правила действий над ними "столбиком", знакомые теперь каждому школьнику. В XII веке эта книга была переведена на латынь и получила широкое распространение в Европе.

В течение длительного времени термин «алгоритм» употребляли преимущественно математики. При этом они пользовались так называемым интуитивным понятием алгоритма - заранее заданное, понятное и точное предписаниевозможному исполнителю совершить определенную последовательность действий для получения решения задачи за конечное число шагов.[1]

Решая различные задачи, математики столкнулись с рядом неразрешимых проблем, т.е. задач, для которых не удавалось создать решающий их алгоритм. Это стало предпосылкой к обоснованию сущности алгоритмов формальным, т.е. математическим путём.

Основные результаты теории алгоритмов были получены в 30-60 годах 20 века Э.Черчем, А. Тьюрингом, А. Постом, А. Колмогоровым, А. Марковым. Введенные в рассмотренные алгоритмические модели, машины Тьюринга, Поста дали возможность устанавливать алгоритмическую разрешимость проблемы путём построения соответствующих машин логического действия.

В настоящее время понятие алгоритма является не только одним из главных понятий математики, но одним из главных понятий современной науки. Более того, с наступлением эры информатики алгоритмы становятся одним из важнейших факторов цивилизации. [1]

*Определение 1.*Алгоритм – конечная последовательность детерминированных арифметических и логических действий над исходными и промежуточными данными задачи обработки информации, выполнение которых приводит к правильному ее решению, т.е. получению требуемых выходных данных. [2]

Свойства алгоритмов.

1. Понятность. Исполнитель алгоритма должен понимать, как его выполнять. Иными словами, имея алгоритм и произвольный вариант исходных данных, исполнитель должен знать, как надо действовать для выполнения этого алгоритма.

2. Дискретность (прерывность, раздельность). Алгоритм должен состоять из отдельных элементарных шагов. Количество шагов должно быть конечным, т.е. после выполнения этих шагов исполнитель должен остановиться. Если действия повторяются бесконечно, говорят, что алгоритм зациклился.

3. Определённость. Последовательность шагов алгоритма должна быть детерминированной (определённой). После каждого шага либо указывается, какой шаг выполнить дальше, либо даётся команда остановки, после этого действия алгоритма считаются законченными.

4. Результативность (конечность). После выполнения конечного количества шагов алгоритм должен выдавать правильный результат решения той или иной информационной задачи.

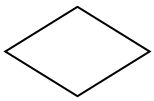
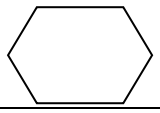
5. Массовость. Алгоритм решения задачи должен разрабатываться в общем виде, т.е. он должен быть применим для некоторого класса задач, различающихся лишь исходными данными. При этом исходные данные могут выбираться из некоторой области, так называемой областиприменимости алгоритма.[2]

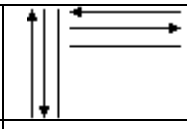
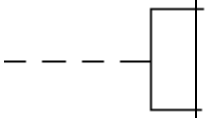
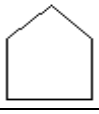
Способы представления алгоритмов.

1. Графический (в виде блок-схемы).
2. Словесный, текстовый (в виде последовательности шагов, описывающих действия естественным языком).
3. Программный (в виде программы, полностью или сжато представленной на каком-либо алгоритмическом языке).

Наиболее наглядным считается графический способ. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий.

Таблица 1 – Символы блок-схем алгоритмов

Процесс		Выполнение операций или групп операций, в результате которых изменяется значения, форма представления или расположение данных.
Решение		Выбор направления выполнения алгоритма или программы в зависимости от некоторых переменных условий
Модификация		Выполнение операций, меняющих команды ил группу команд, т.е. изменяющий программу
Предопределенный процесс		Использование ранее созданных и отдельно описанных алгоритмов и программ
Ручной ввод		Ввод данных оператором в процессе обработки при помощи устройства, непосредственно сопряжённого с вычислительной машиной
Дисплей		Ввод-вывод данных в случае, если непосредственно подключенное к процессору устройство воспроизводит данные и позволяет оператору вносить изменения в процессе их обработки
Документ		Ввод-вывод данных, носителем которых служит бумага.

Линия потока		Указание последовательности связей между символами
Соединитель		Указание связи между прерванными линиями потока, связывающими символы
Пуск-остановка		Начало, конец, прерывание процесса обработки данных или выполнения программы
Комментарий		Связь между элементом схемы и пояснением
Межстраничный соединитель		Указание связи между разъединительными частями схем алгоритмов и программ, расположенных на разных листах.

Размеры символов должны удовлетворять соотношению b=1,5a, рис. 1. На этом же рисунке продемонстрирован пример использования символа «комментарий».

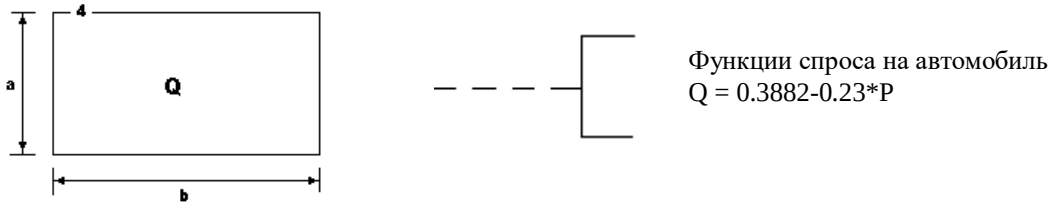
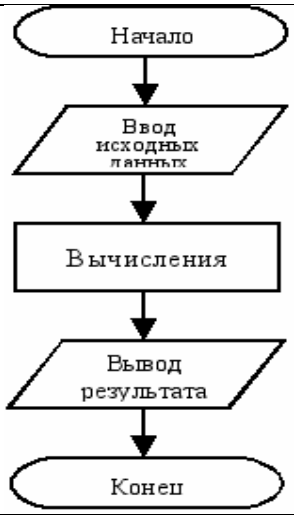
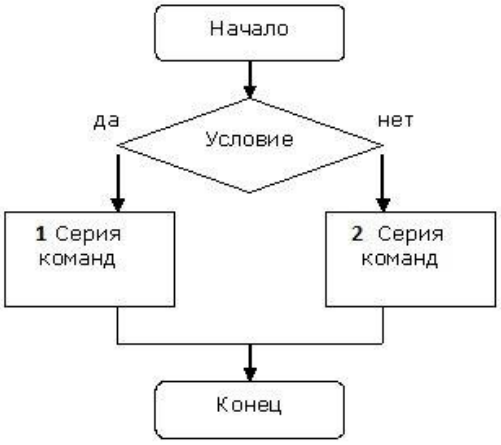
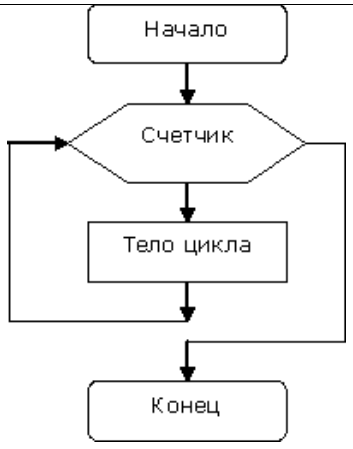
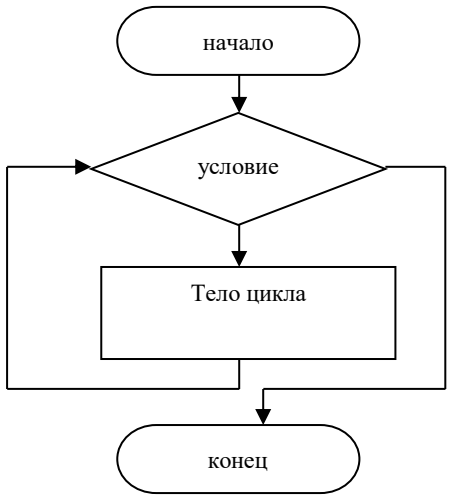
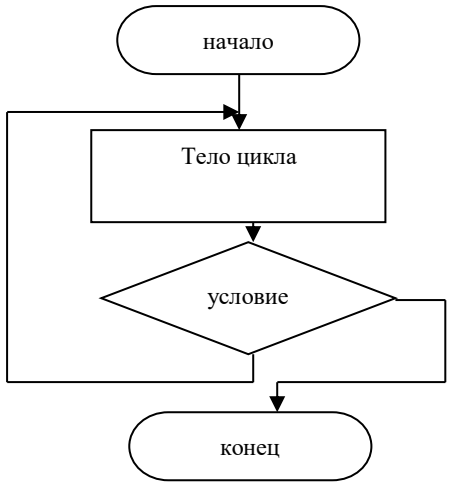


Рис. 1 Фрагмент блок-схемы алгоритма

Таблица 2 – Унифицированные структуры

	Следование
	Полное ветвление
	Цикл с параметром

Продолжение таблицы 2

	Цикл с предусловием
	Цикл с постусловием

В зависимости от употребляемых унифицированных структур алгоритмы программных модулей, составляющих программный комплекс, могут быть линейными, разветвляющимися, циклическими и сложными.

ЛАБОРАТОРНЫЕ ЗАДАНИЯ

Этапы решения задач на ЭВМ.

1. **Постановка задачи**
2. **Анализ и исследование задачи, модели**
3. **Разработка алгоритма**
4. **Программирование**
5. **Тестирование и отладка**
6. **Анализ результатов решения задачи и уточнение в случае необходимости математической модели с повторным выполнением этапов 2-5**
7. **Сопровождение программы**

В данном материале на примерах решения конкретных задач рассматриваются этапы решения на ЭВМ с первого по четвертый.

На лабораторных занятиях при выполнении заданий необходимо представить этапы решения с первого по шестой.

Для иллюстрации этапов решения задач на ЭВМ (1-3) рассмотрим следующую задачу.

ЛАБОРАТОРНОЕ ЗАДАНИЕ 1

Задача 1.

1. Составить алгоритм приближенного вычисления квадратного корня $x = \sqrt{a}$ с заданной точностью ε методом Ньютона.

2. Очередное значение корня x_n вычисляется по формуле$x_n = x_{n-1} + \frac{1}{2}\left(\frac{a}{x_{n-1}} - x_{n-1}\right)$,n=1, 2, …при условии, что задано начальное значение корня x_0. Первое значение корня будет равно $x_1 = x_{0,} + \frac{1}{2}(\frac{a}{x_0} - x_0)$, второе – $x_2 = x_{01,} + \frac{1}{2}(\frac{a}{x_1} - x_1)$и т.д. Корень можно считать вычисленным с заданной точностью ε, если модуль очередного уточнения корня $|\frac{1}{2}\left(\frac{1}{x_n} - x_n\right)| \leq \ \varepsilon$.

3. Блок-схема алгоритма на рис. 2.

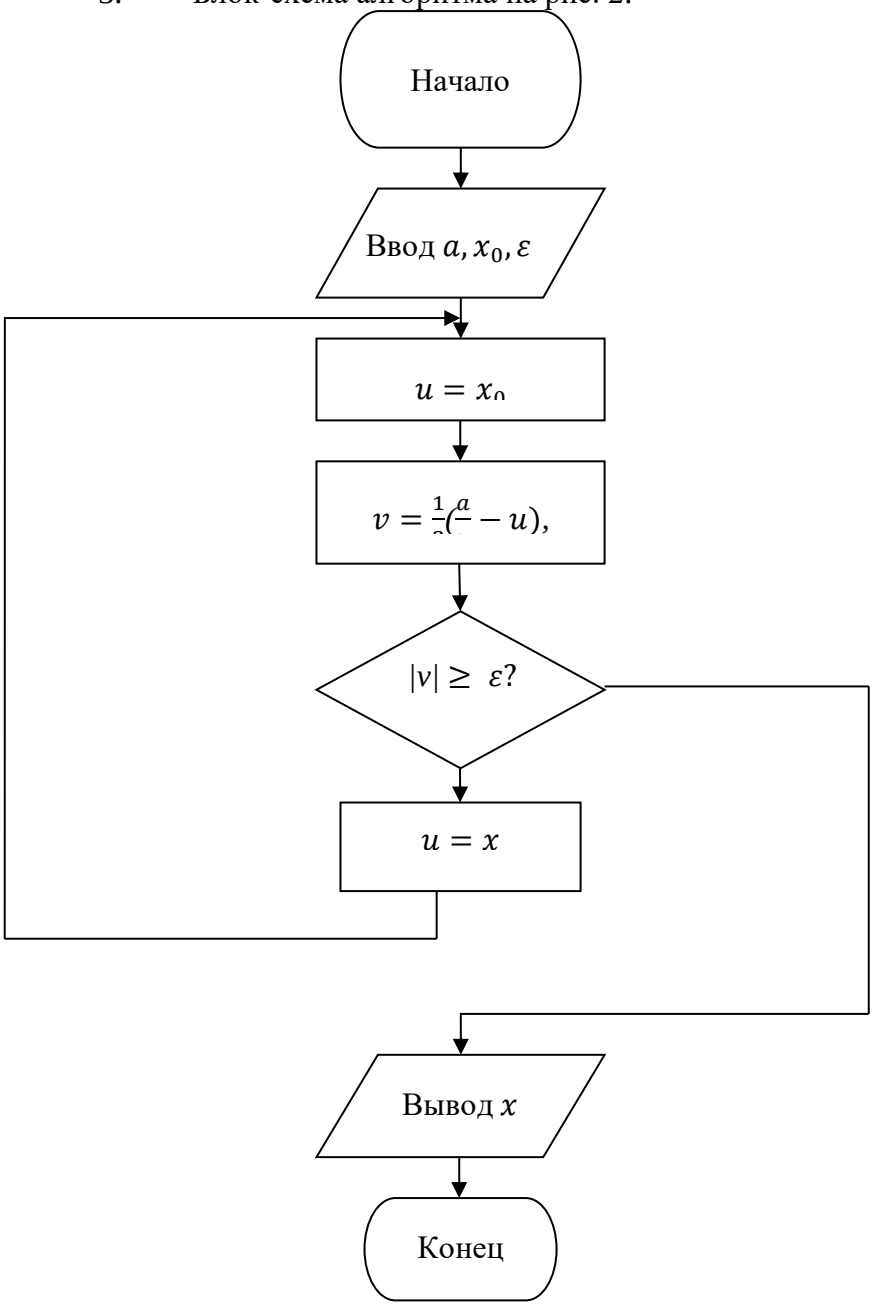


Рис. 2 Алгоритм вычисления квадратного корня методом Ньютона.

1.2 ЛИНЕЙНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ

*Определение 2.*Линейный алгоритм – это алгоритм, содержащий алгоритмическую структуру «следование.

Общий вид блок-схемы линейного алгоритма представлен в таблице 2.Рассмотрим 1-3 этапы решения задачи на ЭВМ с помощью линейного алгоритма.[2]

Задача 2.

1. Даны два действительных положительных числа a и b. Найти среднее арифметическое и среднее геометрическое этих чисел.
2. Среднее арифметическое чисел a и b определяется как $S = \frac{(a+b)}{2}$, а среднее геометрическое как $G = \sqrt{ab}$.
3. Блок-схема алгоритма представлена на рис.3.

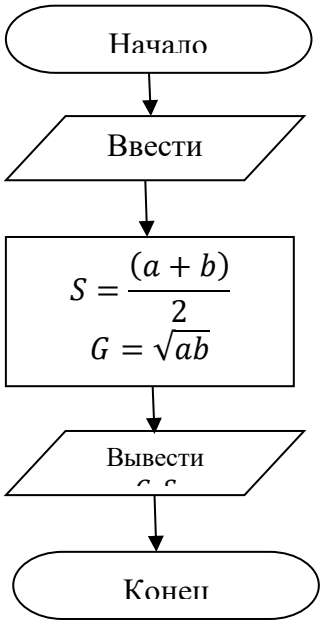


Рис. 3 Алгоритм вычисления среднего арифметического и среднего геометрического двух чисел.

1.3 ВЕТВЛЕНИЕ В ВЫЧИСЛИТЕЛЬНЫХ АЛГОРИТМАХ

*Определение 3.*Конструкция ветвления - это часть алгоритма, в которой в зависимости от выполнения или невыполнения некоторого условия выполняется либо одна, либо другая последовательность действий. Алгоритм, в котором используется конструкция ветвления, называется алгоритмом с ветвлением.

Общий вид блок-схем алгоритма ветвления представлен в таблице 2.

Для реализации алгоритмов с разветвленной структурой в языках программирования используются условные операторы.

Рассмотрим пример решения задачи с помощью алгоритма ветвления

Задача 3.

1. Составить алгоритм вычисления корней квадратного уравнения $ax^2 + bx + c = 0$ с действительными коэффициентами $a, b, c \neq 0$, когда $a \neq 0, b \neq 0, c \neq 0$.
2. Дискриминант квадратного уравнения$ax^2 + bx + c = 0$может иметь три типа корней: разные действительные корни, если$d > 0$, равные действительные корни, если $d = 0$, и комплексные сопряженные корни, если$d < 0$. Разные действительные корни вычисляются по формулам$x_1 = \frac{-b+\sqrt{d}}{2a}, x_2 = \frac{-b-\sqrt{d}}{2a}$. Равные корни определяются так:$x_1 = x_2 = \frac{-b}{2a}$. Комплексные сопряженные корни вычисляются так же, как и действительные, только представляются двойкой чисел$\frac{-b}{2a} \pm i\frac{\sqrt{|d|}}{2a}$, где знак$i$ указывает на то, что$\frac{\sqrt{|d|}}{2a}$ – мнимая часть значения корня. В алгоритме вычисления корней на первом этапе должно быть предусмотрено вычисление значения дискриминантаd и дальнейшая проверка его знака.
3. Блок-схема алгоритма вычисления корней квадратного уравнения представлена на рис.4.

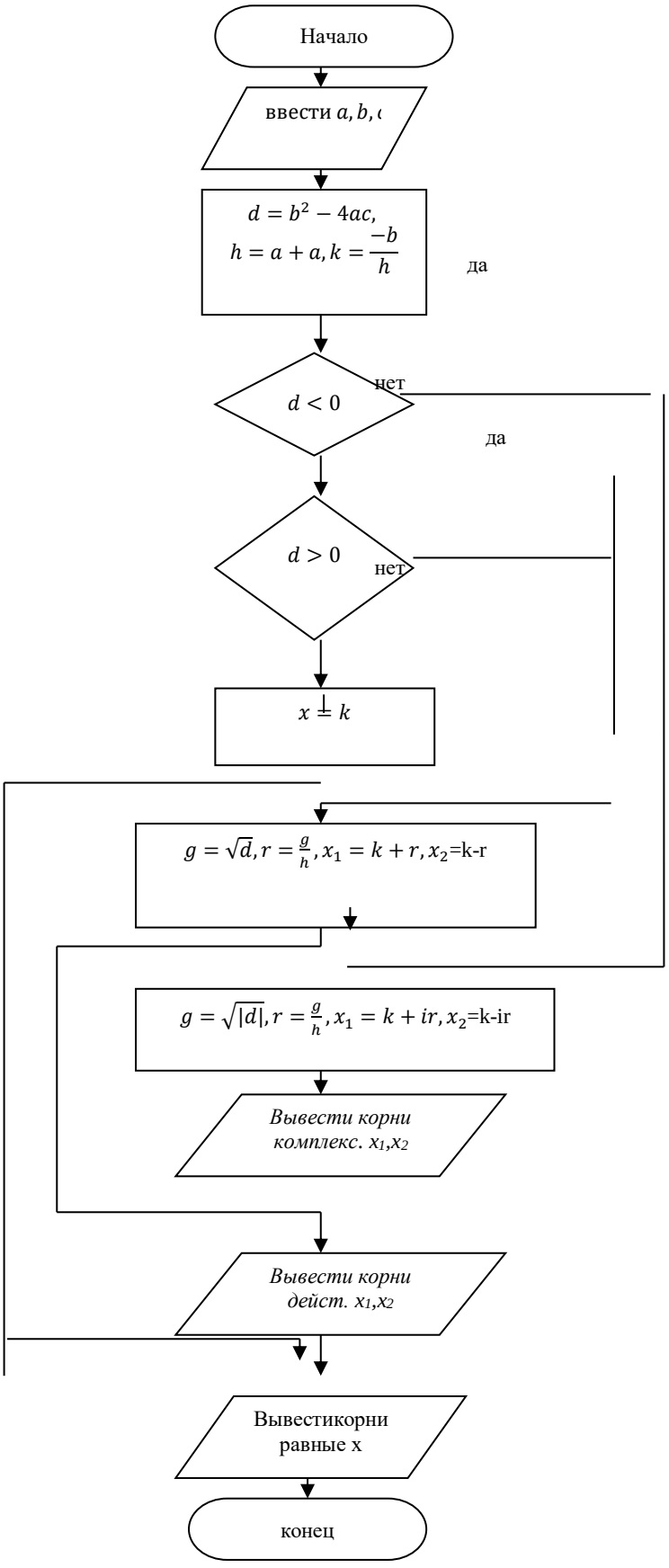


Рис. 4 Алгоритм вычисления корней квадратного уравнения.

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №1. ВЕТВЛЕНИЕ В ВЫЧИСЛИТЕЛЬНЫХ АЛГОРИТМАХ

Вариант 1.

1. Написать программу, которая по паролю будет определять уровень доступа сотрудника к секретной информации в базе данных. Доступ к базе имеют только шесть человек, разбитых на три группы по степени доступа. Они имеют следующие пароли: 9583, 1747 — доступны модули баз A, B, C; 3331, 7922 — доступны модули баз B, C; 9455, 8997 — доступен модуль базы C.

2. Пользователь вводит с клавиатуры три целых числа a,b,c. Необходимо вывести на экран наибольшее из этих чисел.

Вариант 2

1. Вычислить значение функции:

$$F(x)=\begin{cases} x^2-3x+9, x\le 3; \\ \dfrac{1}{x^3+6}, x\ge 3. \end{cases}$$

2. Дано трехзначное число. Написать программу, определяющую кратна ли шести сумма его цифр.

Вариант 3

1. В ЭВМ поступают результаты соревнований по плаванию для трех спортсменов. Составить программу, которая выбирает лучший результат и выводит его на экран с сообщением, что это результат победителя заплыва.

2. Пользователь вводит с клавиатуры три целых числа a,b,c. Необходимо вывести на экран наибольшее из этих чисел.

Вариант 4

1. Даны три положительных числа. Определить, можно ли построить треугольник со сторонами, длины которых равны этим числам. Если возможно, то ответить на вопрос, является ли он прямоугольным.

2. Напишите программу, запрашивающую три вещественных числа и выводящую их на экран в упорядоченном по возрастанию виде.

Вариант 5.

1. Составьте программу, которая уменьшает первое число в пять раз, если оно больше второго по абсолютной величине.

2. Даны три числа a, b, c. Определить, какое из них равно d. Если ни одно не равно d, то найти max (d-a, d-b, d-c).

Вариант 6.

1. Вычислить значение функции:

$$F(x)=\begin{cases} 4x^2+2x-19, x\ge -3{,}5; \\ -\dfrac{2x}{-4x+1}, x<3{,}5. \end{cases}$$

2. Дано четырехзначное число. Написать программу определения больше ли цифра сотен цифры единиц.

Вариант 7

1. Определить, является ли номер автобусного билета счастливым числом. Номер задается шестизначным числом.

2. Даны три вещественных числа a,b,c. Напишите программу, определяющую, могут ли данные числа являться длинами сторон равностороннего треугольника.

Вариант 8.

1. Даны три вещественных числа a,b,c. Напишите программу, определяющую, могут ли данные числа являться длинами сторон любого треугольника.

2. Дана точка с координатами (x,y), требуется определить принадлежность точки отрезку (a,b).

Вариант 9.

1. Даны три вещественных числа a,b,c. Напишите программу, определяющую, могут ли данные числа являться длинами сторон прямоугольного треугольника.

2. В точке (x0,y0) находится центр круга радиусом R. Напишите программу, определяющую, находится ли точка с заданными координатами (x,y) внутри или за пределами круга.

Вариант 10.

1. Дана точка с координатами (x,y), определите, принадлежит ли точка осям координат.

2. Напишите программу, запрашивающую три вещественных числа и выводящую их на экран в упорядоченном по убыванию виде.

ЛАБОРАТОРНОЕ ЗАДАНИЕ 2 ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ (ЦИКЛ С ПАРАМЕТРОМ)

1.4 ЦИКЛЫ В ВЫЧИСЛИТЕЛЬНЫХ АЛГОРИТМАХ

Определение 4. Циклические алгоритмы – алгоритмы, содержащие фрагменты повторения вычислений. Выделяют циклы арифметические и итерационные.В *арифметических циклах* число повторений вычислений известно и определяется счетчиком цикла. При каждом очередном вычислении значение счетчика изменяется на заданную величину и сравнивается с установленным количеством повторений. Если эти величины совпадают, то происходит выход из цикла по счетчику. В противном случае повторения вычислений продолжаются. Если перед началом цикла значение счетчика превышает заданное число повторений, то цикл не выполняется вообще. Возможен принудительный выход из цикла по некоторому наперед заданному значению.

В *итерационных циклах* число повторений неизвестно, выход из цикла осуществляется при выполнении некоторого условия. В случае, когда условие проверяется до начала повторений, циклы называются с предусловием, когда же проверка происходит после очередной итерации, циклы называются с постусловием.

Все арифметические циклы – это циклы с предусловием. [2]

Общий вид блок-схем циклических алгоритмов представлен в таблице 2.

Рассмотрим примеры решения задач с помощью циклических алгоритмов.

Задача4.

1. Составить алгоритм вычисления функции$F = n$!.

2. $F = n$!. – произведение n натуральных чисел 1*2*3…n, $1! = 1$. Вычисление значения факториала можно рассматривать как последовательное повторение операции умножения предшествующего значения факториала F на очередное натуральное число.

3. Блок −схема алгоритма вычисления факториала $n!$ на рис.5.

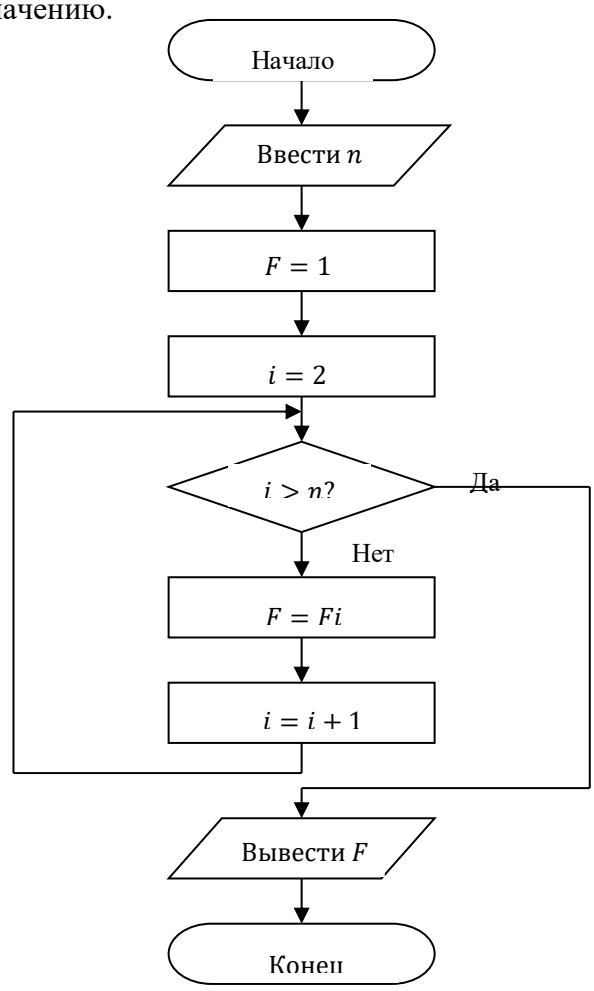


Рис. 5 Алгоритм вычисления функции n!

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №2. *ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ (ЦИКЛ С ПАРАМЕТРОМ)*

Все задачи требуется решить, используя цикл с параметром.

Вариант 1.

1. Найти сумму всех n-значных чисел ($1 \le n \le 4$).
2. Даны два целых числа A и B (A < B). Найти сумму всех целых чисел от A до B включительно.

Вариант 2.

1. Дано действительное число a, натуральное число n. Вычислить:

P = a(a – n)(a – 2n) x … x (a – n2).

2. Даны два целых числа A и B (A < B). Найти произведение всех целых чисел от A до B включительно.

Вариант 3.

1. Дано натуральное число n. Вычислить:

S = 1! + 2! + 3! + … + n! (n>1).

2. Даны два целых числа A и B (A < B). Найти сумму квадратов всех целых чисел от A до B включительно.

Вариант 4.

1. Дано натуральное число n. Вычислить:

$S = 1/3^2 + 1/5^2 + 1/7^2 + \ldots + 1/(2n + 1)^2$.

2. Даны два целых числа A и B (A < B). Найти частное, получаемое при делении A на все делители из диапазона целых чисел от A до B включительно.

Вариант 5.

1. Написать программу, которая вычисляет сумму n- первых членов ряда 1+1/2+1/3+1/4+… Количество суммируемых членов ряда задается во время работы программы.
2. Дано целое число N (> 0). Найти произведение

$N! = 1 \cdot 2 \cdot \ldots \cdot N$

Чтобы избежать целочисленного переполнения, вычислять это произведение с помощью вещественной переменной и вывести его как вещественное число.

Вариант 6.

1. Написать программу, которая вводит с клавиатуры последовательность из пяти дробных чисел и после ввода каждого числа выводит среднее арифметическое полученной части последовательности.
2. Дано целое число N (> 0). Найти сумму

$N^2 + (N + 1)^2 + (N + 2)^2 + \ldots + (2 \cdot N)^2$

Вариант 7.

1. Написать программу, которая генерирует три последовательности из десяти случайных чисел в диапазоне от 1 до 10, выводит каждую последовательность на экран и вычисляет среднее арифметическое каждой последовательности.
2. Дано целое число N (> 0). Найти сумму

1 + 1/2 + 1/3 + … + 1/N

Вариант 8.

1. Составить программу, которая печатает таблицу умножения и сложения натуральных чисел в десятичной системе счисления.
2. Дано вещественное число — цена 1 кг яблок. Вывести стоимость 1.2, 1.4, …, 2 кг конфет.

Вариант 9.

1. Даны целые числа K и N (N > 0). Вывести N раз число K.
2. Выполнить табулирование функции y = cos(x + a) на отрезке [1, 10] с шагомh=1.

Вариант 10.

1. Дано вещественное число — цена 1 кг конфет. Вывести стоимость 1, 2, …, 10 кг конфет.

2. Вычислить сумму значений функции y = x2 на отрезке [1, 5] с шагом 1.

ЛАБОРАТОРНОЕ ЗАДАНИЕ 3. ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ (ЦИКЛЫ С ПРЕДУСЛОВИЕМ И ПОСТУСЛОВИЕМ)

Задача 5.

1. Составить алгоритм вычисления выражения $S = \sin x + sin^2 x \ldots + sin^n x$.

2. Если взять начальное значение суммы $S = \sin x$, то повторяющимся действием будет $S + S \sin x$.

3. Блок −схема алгоритма представлена на рис.6.

Задача 6.

1. Составить алгоритм нахождения наибольшего общего делителя двух целых чисел a и b. [6]

2. Наибольший общий делитель (НОД) двух целых чисел *a*и *b* – это наибольшее целое число, которое делит нацело оба числа. Рассмотрим способ, который называется алгоритмом Евклида. Пусть *a*и*b*одновременно не равные нулю целые неотрицательные числа и a≥b. Если *b* = 0, то НОД(*a, b)* =*a*, а если *b*≠0, то для чисел *a,b*и*r,* где *r* – остаток от деления *a*на *b,* выполняется равенство НОД (*a, b)* = НОД(*b, r*). Действительно, *r*:=*a*Mod*b, r*:= *a* – (*a*Div*b*) * *b*. Если какое-то число делит нацело и *a*, и *b*, то из приведенного равенства следует, что оно делит нацело и число *r*.

3. Блок-схема алгоритма Евклида нахождения НОД и НОК двух целых чисел на рис.7.

Задача 7.

1. Дано натуральное число *n.* Требуется подсчитать количество цифр данного числа.

2. Количество цифр в числе *n* неизвестно, поэтому необходимо использовать цикл с предусловием. Подсчёт количества цифр можно начать с последней цифры числа, далее увеличить изначально нулевой счётчик цифр на единицу. Повторять уменьшение числа в 10 раз, пока оно не станет равным 0.

3. Блок-схема на рис.8.

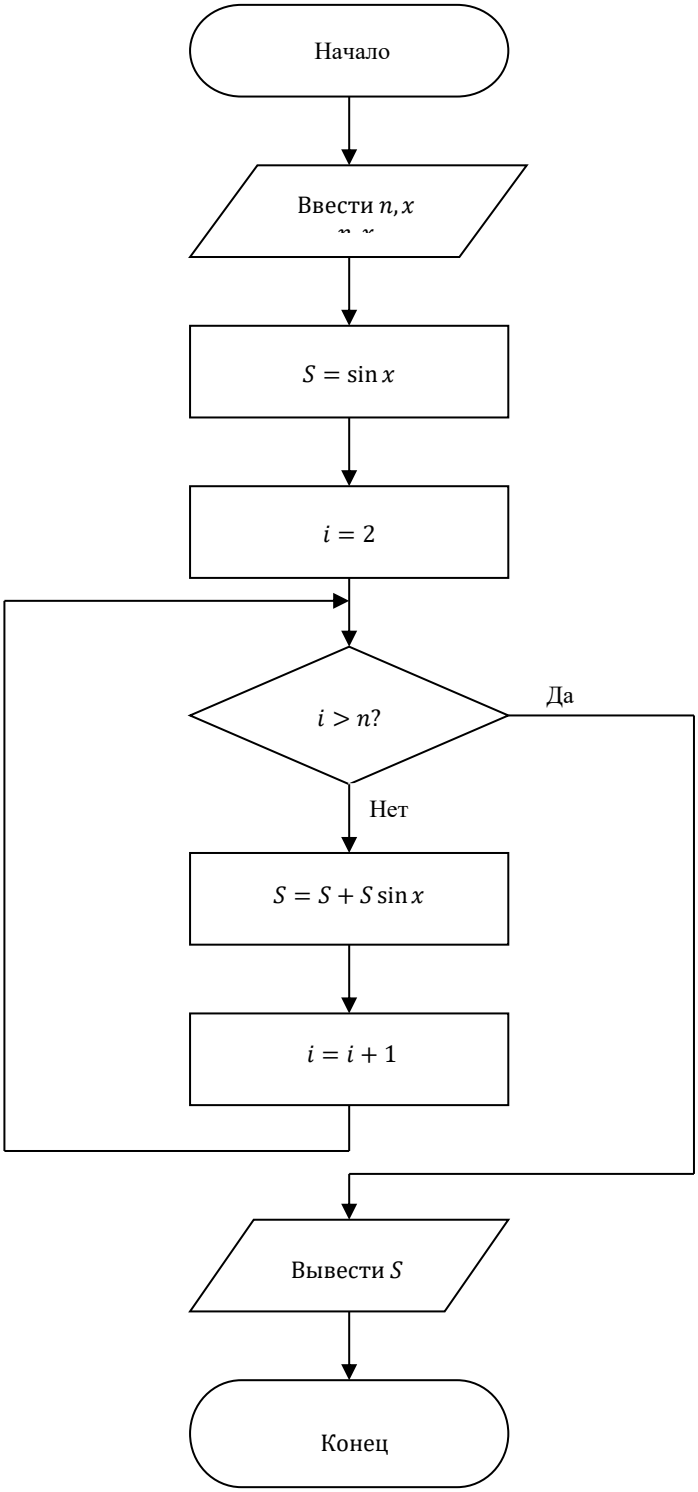


Рис. 6 Алгоритм вычисления выражения $S = \sin x + sin^2 x \ldots + sin^n x$.

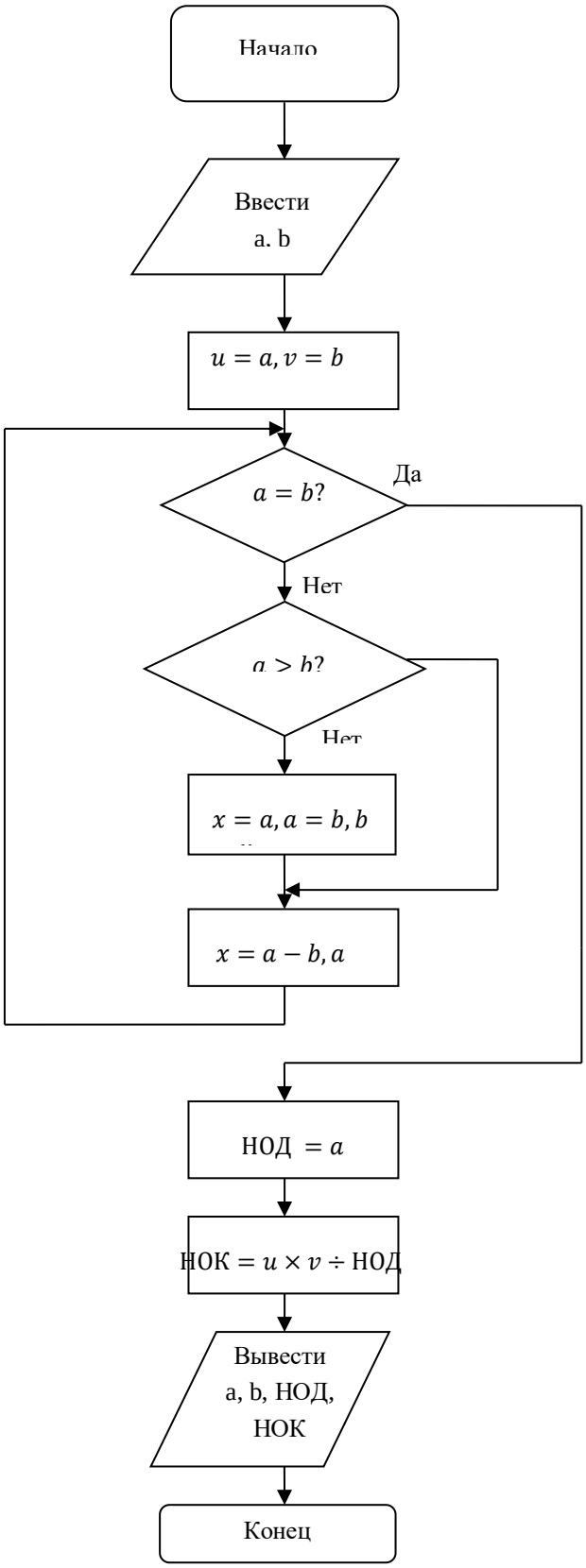


Рис.7 Алгоритм Евклида нахождения НОД и НОК двух целых чисел

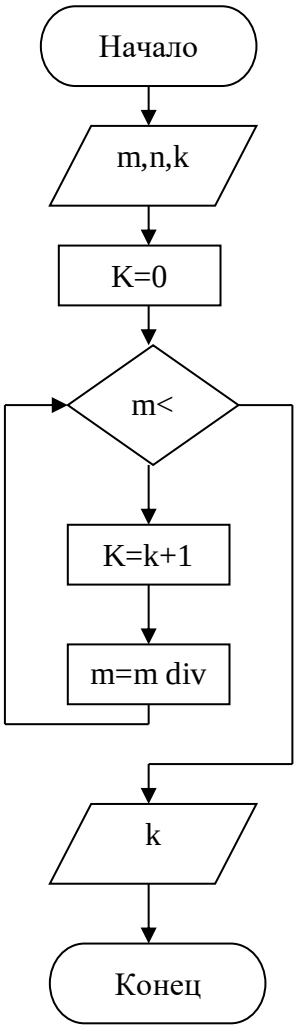


Рис.8 Алгоритм подсчета количества цифр в натуральном числе.

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №3. *ЦИКЛИЧЕСКИЕ АЛГОРИТМЫ (ЦИКЛЫ С ПРЕДУСЛОВИЕМ И ПОСТУСЛОВИЕМ)*

Составить алгоритмы для следующих задач, используя циклы с предусловием и постусловием.

Вариант 1.

1. Начав тренировки, спортсмен в первый день пробежал 10 км. Каждый день он увеличивал дневную норму на 10% нормы предыдущего дня. Какой суммарный путь пробежит спортсмен за 7 дней?
2. Определить, какие различные цифры входят в заданное целое число.

Вариант 2.

1. Одноклеточная амеба каждые 3 часа делится на 2 клетки. Определить, сколько амеб будет 3, 6, 9, 12, …, 24 часа.
2. Вывести все квадраты натуральных чисел, не превосходящие данного числа N.

Вариант 3.

1. Написать программу, вычисляющую сумму и среднее арифметическое последовательности положительных чисел, которые вводятся с клавиатуры.
2. Вычислить НОД двух чисел A и B.

Вариант 4.

1. Написать программу, которая определяет максимальное число из введенной с клавиатуры последовательности положительных чисел (длина последовательности не ограничена).

2. Дано число. Найти сумму и произведение его цифр.

Вариант 5.

1. Написать программу, которая проверяет, является ли целое число, введенное пользователем, простым.

2. Написать программу для решения следующей задачи: рост ребенка на начало года – 120 см. За месяц он подрастает на 2%. Через сколько месяцев его рост станет больше или равным 150 см.?

Вариант 6.

1. Составьте программу для нахождения всех автоморфных чисел в отрезке [m,n]. Автоморфным называется целое число, которое равно последним числам своего квадрата. Например: 52=25, 62=36, 252=625.

2. Начальный вклад в банке равен 1000 руб. Через каждый месяц размер вклада увеличивается на P процентов от имеющейся суммы (P — вещественное число, 0 < P < 25). По данному P определить, через сколько месяцев размер вклада превысит 1100 руб., и вывести найденное количество месяцев K (целое число) и итоговый размер вклада S (вещественное число).

Вариант 7.

1. Написать программу, которая вычисляет НОК двух целых чисел.

2. Дано целое число N (> 0). Используя операции деления нацело и взятия остатка от деления, вывести все его цифры, начиная с самой правой (разряда единиц).

Вариант 8.

1. Написать программу, вычисляющую произведение положительных четных чисел до 10.

2. Дано целое число N (> 0). С помощью операций деления нацело и взятия остатка от деления определить, имеется ли в записи числа N цифра «2». Если имеется, то вывести True, если нет — вывести False.

Вариант 9.

1. Написать программу, вычисляющую значение выражения $y=\sqrt{k}$ для k=1,3,5,7,9.

2. Дано целое число N (> 0). С помощью операций деления нацело и взятия остатка от деления определить, имеются ли в записи числа N нечетные цифры. Если имеются, то вывести True, если нет — вывести False.

Вариант 10.

1. Натуральные числа a, b, c называются числами Пифагора, если выполняется условие a2 + b2 = c2. Напечатать все числа Пифагора меньшие N.

2. Данon вещественных чисел. Найти их среднее арифметическое.

ЛАБОРАТОРНАЯ РАБОТА №4. ОДНОМЕРНЫЕ МАССИВЫ

Массив - это упорядоченный набор однотипных элементов, обозначаемых одним именем; доступ к элементу массива осуществляется по его номеру.

Для записи элементов массива в память компьютера нужно выделить для их хранения необходимое количество ячеек памяти, которое определяется *размером* массива.

В программе для каждого массива должны быть указаны его параметры: имя, размерность и размер. Бывают одномерные, двумерные и т.д. массивы. Это называется *размерностью.*

A: 3,-4,0,3,-5,10,0

A[1]=3, A[3]=0, A[7]=0

I - номер элемента, A[I] - элемент массива, стоящий на I-ом месте

Пример 1.

Сформировать и вывести на экран последовательность из n элементов, заданных датчиком случайных чисел на интервале [-23, 34].

```
Program posled;
Var a: array[1..100] of integer;
i, n: integer;
Begin
  Write (‘Сколькоэлементов? ’);  Readln (n);
   For i=1 to n do
begin
a[i]:= Random(58)-23;
writeln (a[i],’ ‘);
end;
End.
```

Пример 2.

Найти произведение элементов одномерного массива, состоящего из n элементов. Элементывводятсясклавиатуры.

```
Program proisveden;
Var a: array[1..100] of integer;
i, n, p: integer;
Begin
  Write (‘Сколькоэлементов? ’);  Readln (n);
p:=1;
  For i:=1 to n do
begin
write (‘введитечисло’); readln (a[i]);
p:=p*a[i];
end;
writeln(‘произведение элементов равно:  ‘,p);
End.
```

Пример 3.

Найти сумму элементов одномерного массива. Размер произвольный. Элементы вводятся с клавиатуры.

```
Program summa;
Var a: array[1..100] of real;
i, n: integer;
s: real;
Begin
  Write (‘n=’); Readln (n);
s:=0;
  For i:=1 to n do
begin
write (‘введитечисло’); readln (a[i]);
s:=s+a[i];
end;
```

```
writeln(‘суммаэлементовравна  ‘,s);
End.
```

<u>Пример 4</u>

Задан массив A, состоящий из n чисел. Найти среднее арифметическое его элементов. Элементывводятсясклавиатуры.

```
Program srednee;
Var a: array[1..100] of  real;
i, n: integer;
s,sred: real;
Begin
  Write (‘n=’); Readln (n);
s:=0;
  For i:=1 to n do
begin
write (‘введитечисло’); readln (a[i]);
s:=s+a[i];
end;
sred:=s/n;
writeln(‘среднее арифметическое  элементов: ‘,s);
End.
```

<u>Пример 5.</u>

Найти сумму элементов массива с четными номерами, содержащего N элементов. Элементывводятсясклавиатуры.

```
Program sumshet;
Var a: array[1..100] of  real;
i, n: integer;
s,sred: real;
Begin
  Write (‘n=’); Readln (n);
s:=0;
  For i:=1 to n do
begin
write (‘введитечисло’); readln (a[i]);
if i mod 2 = 0 then s:=s+a[i];
end;
writeln(‘сумма элементов с четными номерами: ‘,s);
End.
```

<u>Пример 6.</u>

Сформировать и вывести на экран массив, элементы которого заданы датчиком случайных чисел на интервале [-19, 26] (размер произвольный). Найти произведение элементов с нечетными номерами.

```
Programproisvednechet;
Vara: array[1..100] ofinteger;
i, n, p: integer;
Begin
  Write (‘Сколькоэлементов? ’);  Readln (n);
```

```
  P:=1;
  For i=1 to n do
begin
a[i]:= Random(46)-19;
writeln (a[i],' ');
if i mod 2 <> 0 then P=P*a[i];
end;
Writeln('Произведение элементов с нечетными номерами:', P);
End.
```

Пример 7.

Сформировать и вывести на экран массив, элементы которого заданы датчиком случайных чисел на интервале [-56, 47] (размер произвольный). Найти произведение элементов с четными номерами, которые превосходят некоторое число t.

```
Program proisvedchetbolt;
Var a: array[1..100] of integer;
i, n, p, t: integer;
Begin
  Write ('Сколькоэлементов? ');  Readln (n);
  P:=1;
  For i=1 to n do
begin
a[i]:= Random(104)-56;  writeln (a[i],' ');
if (i mod 2 = 0) and (a[i]>t) then P=P*a[i];
end;
Writeln('Произведение элементов с четными номерами, превосходящие число t:', P);
End.
```

Пример 8.

Найти наименьший элемент одномерного массива, состоящего из n элементов. Элементывводятсясклавиатуры.

```
Program minim;
Var a: array[1..100] of real;
i, n: integer;
min: real;
Begin
  Write ('n='); Readln (n);
  For i:=1 to n do
begin
write('a[',i,']='); readln (a[i]);
end;
min:=a[1];
  For i:=2 to n do
    If a[i]< min then min:=a[i];
Writeln('наименьшее число:   ',min);
End.
```

Пример 9.

Найти номер наименьшего элемента в массиве, заданного датчиком случайных чисел на интервале [-20, 25]. Размерпроизвольный.

```
Program numberminim;
Var a: array[1..100] of integer;
i, n, num, min: integer;
Begin
  Write (‘n=’); Readln (n);
  For i:=1 to n do
begin
a[i]:= Random(46)-20;
writeln (a[i]);
end;
min:=a[1];
num:=1;
  For i:=2 to n do
      If a[i]< min then
begin
min:=a[i];
num:=i;
end;
Writeln(‘ номер наименьшего элемента:  ‘,num);
End.
```

<u>Пример 10.</u>

В заданном одномерном массиве, состоящем из n целых чисел, подсчитать количество нулей.

```
Program  kolv0;
Var a: array[1..100] of integer;
i, n, k: integer;
Begin
  Write (‘n=’); Readln (n);
  For i:=1 to n do
begin
Write(‘a[‘,i,’]=’); readln (a[i]);
if a[i]=0 then k:=k+1;
end;
Writeln(‘количество 0 равно  ’, k);
end.
```

<u>Пример 11.</u>

В заданном одномерном массиве, состоящем из n целых чисел, подсчитать количество четных элементов.

```
Program  kolvcchet;
Var a: array[1..100] of integer;
i, n, k: integer;
Begin
  Write (‘n=’); Readln (n);
  For i:=1 to n do
begin
Write(‘a[‘,i,’]=’); readln (a[i]);
if  a[i] mod 2=0 then k:=k+1;
end;
```

```
Writeln(‘количество четных элементов: ’, k);
end.
```

Пример 12.

Найдите среднее арифметическое элементов массива, состоящего из 10 чисел, которые превышают по величине число С. Элементы вводятся с клавиатуры.

```
Program sredarifmet;
Var a: array[1..10] of  real;
i, k: integer;
     C, S, sred: real;
Begin
  For i:=1 to 10 do
begin
write(‘a[‘,i,’]=’);     readln (a[i]);
end;
write(‘введитеС: ‘); readln (C );
  For i:=1 to 10 do
begin
      If a[i]>C then
begin      S=S+a[i];     K=K+1;   end;
end;
sred=S/k;
Writeln(‘среднее арифметическое чисел, превосходящих ‘,C,’  равно  ‘,sred);
End.
```

Пример 13

Найти произведение элементов целочисленного одномерного массива с четными номерами, состоящего из n элементов. Элементывводятсясклавиатуры.

```
Program proizved_chet;
Var a: array [1..100] of integer;
i, n, p: integer;
Begin
p:=1;
write ('n=');  readln (n);
for i:=1 to n do
begin
write ('a[',i,']='); readln (a[i]);
if i mod 2=0 then p:=p*a[i];
end;
Writeln ('произведение элементов массива с четными номерами равно ',p);
End.
```

Пример 14.

Массив А вводится с клавиатуры. Сформировать новый массив В, состоящий из четных элементов массива А. Элементы вводятся с клавиатуры. Размер n.

```
Program newmasiv;
Var a: array[1..100] of  integer;
b: array[1..100] of  integer;
n, i, k: integer;
```

```
Begin
kol:=0; k:=0;
write ('n='); readln (n);
  For i:=1 to n do
begin
write(‘a[‘,i,’]=’);     readln (a[i]);
if a[i] mod 2=0 then
begin
k:=k+1; b[k]:=a[i]; kol:=kol+1;
end;
ifkol=0 then writeln(‘четныхэлементовнет’) else
for k:=1 to kol do  write(‘b[‘,k,’]=’,b[k]);
end;
```

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №4. ОДНОМЕРНЫЕ МАССИВЫ

Вариант 1. Массив А вводится с клавиатуры. Найти сумму его элементов. Размер произвольный.Массив задан датчиком случайных чисел на интервале [-37, 66]. Найти наименьший нечетный элемент. Размер произвольный.

Вариант 2. Найти произведение элементов, кратных 3 в массиве, заданном датчиком случайных чисел на интервале [-28, 27]. Размер произвольный.

Вариант 3. Массив А вводится с клавиатуры. Найти среднее арифметическое его элементов с нечетными номерами. Размер произвольный.

Вариант4. Массив А вводится с клавиатуры. Найти сумму его элементов с четными номерами, произведение отрицательных элементов, количество нечетных элементов. Размер произвольный.

Вариант5. Найти наибольший элемент и его номер в последовательности, элементы которой вводятся с клавиатуры. Размер произвольный.

Вариант6. Найти средне арифметическое элементов последовательности, превосходящих некоторое число С. Массив задан датчиком случайных чисел на интервале [-44, 35]. Размер произвольный. Значение С вводится с экрана.

Вариант7. Массив А вводится с клавиатуры. Вывести только нечетные элементы. Размер произвольный.Массив задан датчиком случайных чисел на интервале [-31, 45]. Сформировать новый массив В, состоящий из нечетных элементов массива А. Размер произвольный.

Вариант8. Упорядочить данную последовательность по убыванию. Элементы вводятся с клавиатуры. Размер произвольный.Массив А вводится с клавиатуры. Сформировать новый массив В, состоящий из положительных элементов массива А и найти в нем наибольший элемент. Размер произвольный.

Вариант9. порядочить данную последовательность по убыванию. Массив задан датчиком случайных чисел на интервале [-54, 33]. Размер произвольный.

Вариант 10. Массив А вводится с клавиатуры. Сформировать новый массив В, состоящий из положительных элементов массива А. Размер произвольный.

ЛАБОРАТОРНАЯ РАБОТА № 5 ДВУМЕРНЫЕ МАССИВЫ (МАТРИЦЫ)

Пример 1.

Сформировать с помощью датчика случайных чисел и вывести на экран матрицу, размером MxN. Элементызадаютсянаинтервале [-20, 25].

```
Var a: array[1..50,1..50] of integer;
```

```
i, j, n, m: integer;
Begin
Write(‘сколькострок?’); Readln(m);
Write(‘сколькостолбцов?’); Readln(n);
For i:=1 to m do
begin
      For j:=1 to n do
begin
a[i,j]:=int(rnd*46)-20;
write(a[i,j],’ ‘);
end;
writeln;
end;
End.
```

Пример 22.

В двумерном массиве, состоящем из n целых чисел, найти сумму элементов в каждой строке. Размерпроизвольный.

```
Program summastrok;
Var a: array[1..50,1..50] of integer;
i, j, n, m,S: integer;
Begin
Write(‘сколькострок?’); Readln(m);
Write(‘сколькостолбцов?’); Readln(n);
For i:=1 to m do
  For j:=1 to n do
begin
write(‘a[‘,i,’,’,j,’)=’);     readln (a[i,j]);
end;
For i:=1 to m do
begin
    S:=0;
    For j:=1 to n do
      S:=S+a[i,j];
Writeln(‘суммаэлементовв ‘,i,’ строкеравна  ‘,S);
end;
End.
```

Пример 3.

Найти наименьший элемент двумерного массива. Размер MXN. Элементы задаются на интервале [-30, 45].

```
Program minim;
Var a: array[1..50,1..50] of integer;
i, j, n, m, min: integer;
Begin
Write(‘сколькострок?’); Readln(m);
Write(‘сколькостолбцов?’); Readln(n);
For i:=1 to m do
begin
      For j:=1 to n do
begin
a[i,j]:=int(rnd*76)-30;  write(a[i,j],’ ‘);
```

```
end;
writeln;
end;
min:=a[1,1];
  For i:=1 to m do
  For j:=1 to n do
if a[i,j]< min then min:=a[i,j];
Writeln('наименьшее число   ',min);
End.
```

Пример 4.

В двумерном массиве, состоящем из целых чисел, найти наименьший элемент и номер строки, в которой он находится. Элементывводятсясклавиатуры. Размер MXN.

```
Program minim;
Var a: array[1..50,1..50] of integer;
i, j, m, n, min, K: integer;
Begin
Write('сколькострок?'); Readln(m);
Write('сколькостолбцов?'); Readln(n);
For i:=1 to m do
For j:=1 to n do
      begin       write('a[',i,',',j,']=');  readln (a[i,j]);  end;
min:=a[1,1]; K:=1;
  For i:=1 to m do
  For j:=1 to n do
    If a[i,j]< min then
begin
min:=a[i,j];    K:=i;
end;
Writeln('наименьшее число   ',min,'  находится в ', k , ' строке');
End.
```

Пример 5.

Найти сумму элементов в каждой строке двумерного массива, состоящего из целых чисел. Размер MXN. Элементызадаютсянаинтервале [-19, 30].

```
programsumstr;
Var a: array[1..50,1..50] of integer;
i, j, n, m,sum: integer;
Begin
Write('сколькострок?'); Readln(m);
Write('сколькостолбцов?'); Readln(n);
For i:=1 to m do
begin
      For j:=1 to n do
begin
a[i,j]:=int(rnd*50)-19;  write(a[i,j],' ');
end;
writeln;
end;
for i:=1 to m do
begin
```

```
sum:=0;
for j:=1 to n do sum:=sum+a[i,j];
writeln('сумма элементов в ',i,' строке: ',sum);
end;
end.
```

Пример 6

Подсчитать количество положительных элементов в каждой строке матрицы размером MxN, элементы которой вводятся с клавиатуры.

```
programkolpolvstr;
Var a: array[1..50,1..50] of integer;
i, j, m, n, kol: integer;
Begin
Write('сколькострок?'); Readln(m);
Write('сколькостолбцов?'); Readln(n);
For i:=1 to m do
begin
   For j:=1 to n do
     begin      write('a[',i,',',j,']=');  readln (a[i,j]);  end;
for i:=1 to m do
begin
kol:=0;
for j:=1 to n do if a[i,j]>0 then kol:=kol+1;
writeln('количество положительных элементов в ',i,' строке: ',kol);
end;
writeln;
end;
end.
```

Пример 7.

Сформировать матрицу типа

1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1

```
programformir;
Var a: array[1..50,1..50] of integer;
i, j, m, n: integer;
Begin
Write('сколькострок?'); Readln(m);
Write('сколько столбцов?'); Readln(n);
For i:=1 to m do
begin
   For j:=1 to n do
begin
if i=j then a[i,j]:=1 else a[i,j]:=0;
write(a[i,j]);
end;
writeln;
End.
```

Пример 8.

Найти номер столбца массива размером MxN, в котором находится наибольшее количество отрицательных элементов. Элементывводятсясклавиатуры.

```
programnomerstolb;
Var a: array[1..50,1..50] of integer;
b: array[1..50] of integer;
i, j, m, n, max, jmax: integer;
Begin
Write('сколькострок?'); Readln(m);
Write('сколькостолбцов?'); Readln(n);
 For i:=1 to m do
begin
   For j:=1 to n do
begin
          write('a[',i,',',j,']='); readln (a[i,j]);
end;
for j:=1 to m do
begin
b[j]:=0;
for i:=1 to n do
if a[i,j]<0 then b[j]:=b[j]+1;
end;
max:=b[1]; jmax:=1;
  For j:=2 to n do
begin
if b[j]>max then
begin
max:=b[j]; jmax:=j;
end;
end;
writeln('Наибольшее количество отрицательных элементов в ';jmax ; 'столбце');
end;
end.
```

Пример 9.

Упорядочить каждый столбец матрицы по возрастанию. Массив размером MxN, элементы которого задаются датчиком случайных чисел на интервале [-17;26].

```
programporydok;
Var a: array[1..50,1..50] of integer;
i, j, n, m,t,r: integer;
Begin
Write('сколькострок?'); Readln(m);
Write('сколькостолбцов?'); Readln(n);
For i:=1 to m do
begin
     For j:=1 to n do
begin
a[i,j]:=int(rnd*44)-17;
write(a[i,j],' ');
end;
```

```
Writeln;
end;
For j:=1 to n do
  For r:=1 to m do
    For i:=1 to m-1 do
if a[i,j]> a[i+1,j] then
begin
t:= a[i,j];
a[i,j]:= a[i+1,j];
a[i+1,j]:= t;
end;
For i:=1 to m do
begin
   For j:=1 to n do write(a[i,j],' ');
Writln;
end;
End.
```

Пример 10.

Сформировать матрицу **1 1 1 1**
2 2 2 2
3 3 3 3

```
program former;
Var a: array[1..3,1..4] of integer;
i, j: integer;
Begin
For i:=1 to 3 do
begin
   For j:=1 to 4 do
begin
a[i,j]:=i; write(a[i,j]);
end;
writln;
end;
End.
```

Пример 11.

Найти наибольшее нечетное число в матрице размером MxN, элементы которой задаются датчиком случайных чисел на интервале [-27, 38].

```
programmaxnechet;
Var a: array[1..50,1..50] of integer;
b: array[1..50] of integer;
i, j, n, m, p, max, k: integer;
Begin
Write('сколькострок?'); Readln(m);
Write('сколькостолбцов?'); Readln(n);
For i:=1 to m do
begin
     For j:=1 to n do
begin
a[i,j]:=int(rnd*66)-27;
write(a[i,j],' ');
```

```
end;
writeln;
end;
kol:=0; p:=0;
For i:=1 to m do
    For j:=1 to n do
        If a[i;j]mod 2 <> 0 then
begin
p:=p+1; b[p]:=a[i,j]; k:=k+1:
end;
if k=0 then writeln('нечетныхэлементовнет') else
                        begin
write('нечетные элементы:');
max:=b[1];
for p:=1 to k do
begin
writ(b[p],' ');
if b[p]>max then max:=b[p];
end;
writeln('наибольшее нечетное число', max);
end;
```

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ № 5
ДВУМЕРНЫЕ МАССИВЫ (МАТРИЦЫ)

Вариант1.

1. Подсчитать количество положительных элементов в каждом столбце матрицы размером MxN, элементы которой вводятся с клавиатуры.
2. Найти наибольшее число, кратное 3, в матрице размером MxN, элементы которой вводятся с клавиатуры.

Вариант2.

1. Подсчитать количество отрицательных элементов в каждой строке матрицы размером MxN, элементы которой задаются с помощью датчика случайных чисел на интервале [-35; 65].
2. Найти номер столбца массива размером MxN, в котором находится наибольшее количество элементов, кратных 5. Элементы задаются датчиком случайных чисел на интервале [-27;43].

Вариант3.

1. Подсчитать количество четных элементов в каждом столбце матрицы размером MxN, элементы которой задаются с помощью датчика случайных чисел на интервале [-98; 54].
2. Найти сумму элементов, стоящих на побочной диагонали массива размером MxN, элементы которого вводятся с клавиатуры

Вариант4.

1. Подсчитать количество четных отрицательных элементов в матрице размером MxN, элементы которой вводятся с клавиатуры.
2. Найти номер строки массива размером MxN, в котором находится наибольшее количество четных элементов. Элементы задаются датчиком случайных чисел на интервале [-54;61].

Вариант5.

1. Сформировать матрицу **1 1 1 1**
2 2 2
3 3 3
4 4 4

2. Найти наименьшее четное число в матрице размером MxN, элементы которой задаются датчиком случайных чисел на интервале [-65, 45].

Вариант6.

1Сформировать матрицу **1 2 3 4**
1 2 3 4
1 2 3 4
1 2 3 4

2. Найти произведение диагональных элементов массива размером MxN, элементы которого вводятся с клавиатуры.

Вариант7.

1Сформировать матрицу **0 0 0 1**
0 0 1 0
0 1 0 0
1 0 0 0

2Найти произведение элементов в каждой строке массива размером MxN, элементы которого вводятся с клавиатуры.

Вариант8.

1. Найти номер наибольшего элемента массива размером MxN, элементы которого задаются датчиком случайных чисел на интервале [-67;23].

2.Найти произведение элементов в каждом столбце массива размером MxN, элементы которого вводятся с клавиатуры.

Вариант9.

1Найти номер наибольшего элемента массива размером MxN, элементы которого вводятся с клавиатуры.

2.Найти сумму элементов в каждом столбце массива размером MxN, элементы которого задаются датчиком случайных чисел на интервале [-19;20].

Вариант10.

1. Найти наибольший элемент массива размером MxN, элементы которого задаются датчиком случайных чисел на интервале [-25;19].

2.Найти наименьший элемент массива размером MxN, элементы которого вводятся с клавиатуры.

ЛАБОРАТОРНАЯ РАБОТА № 6. МАШИНА ПОСТА

В 1936 году американский математик и логик Эмиль Леон Пост (1897–1954) предложил абстрактную вычислительную конструкцию, позволяющую формально определить алгоритм и названную впоследствии машиной Поста**.** При разработке вычислительной конструкции Пост руководствовался принципом создания максимально простой абстракции: минимумом операций при обработке информации, входная информация должна быть закодирована с использованием минимального набора символов.

В теории алгоритмов существует так называемый “тезис Поста”: “Всякий алгоритм представим в форме машины Поста”. Этот тезис одновременно является формальным

определением алгоритма. Алгоритм (по Посту) — программа для машины Поста, приводящая к решению поставленной задачи.

Тезис Поста является гипотезой. Его невозможно строго доказать (так же, как и тезис Тьюринга), потому что в нем фигурируют, с одной стороны, интуитивное понятие “всякий алгоритм”, а с другой стороны — точное понятие “машина Поста”. Для того чтобы опровергнуть гипотезу Поста, необходимо придумать алгоритм, который невозможно записать в виде программы для машины Поста. На сегодняшний день такого алгоритма не существует. [12]

Состав машины Поста.

Машина Поста состоит из ленты и каретки. Лента бесконечна и разделена на секции одинакового размера — ячейки.

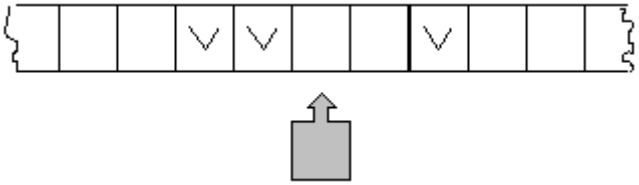


Рис. 9. Состав машины Поста

В каждой ячейке ленты может стоять метка **V** либо ничего не записано.. Состояние ленты — это распределение меток по ячейкам. Состояние ленты меняется в процессе работы машины.

Каретка может передвигаться вдоль ленты влево и вправо. Когда она неподвижна, она стоит против ровно одной ячейки ленты, т.е. каретка обозревает одну ячейку. За единицу времени каретка может совершить одно из трех действий: стереть метку, поставить метку, совершить движение на соседнюю ячейку. Состояние машины Поста складывается из состояния ленты и положения каретки.

Команды машины Поста:

1. записать 1 (метку), перейти к *i*-й строке программы;
2. записать 0 (стереть метку), перейти к *i*-й строке программы;
3. сдвиг влево, перейти к *i*-й строке программы;
4. сдвиг вправо, перейти к *i*-й строке программы;
5. останов;
6. если 0, то перейти к *i*, иначе перейти к *j*.

Недопустимые действия, ведущих к аварийной остановке машины:

- попытка записать 1 (метку) в заполненную ячейку;
- попытка стереть метку в пустой ячейке;
- бесконечное выполнение (зацикливание).

Команды машины обозначают следующим образом (рис.10):

→	Шаг вправо
←	Шаг влево
V	Записать отметку
X	Стереть отметку
? a; b	Просмотреть ячейку; если в ячейке находится 0, то перейти на команду с номером *a*, иначе на команду с номером *b*
!	Останов

Рис. 10 Команды машины Поста

Рассмотрим несколько арифметических задач для машины Поста и их решение.[12]

С помощью простейших операций, которыми располагает машина Поста, можно выполнять арифметические операции — основу любого современного процессора.

1. На ленте задан массив меток. Увеличить длину массива на 2 метки. Каретка находится либо слева от массива, либо над одной из ячеек самого массива. (увеличение числа на 2).

Решение:

1. ? 2; 3 (команды 1 и 2 — передвигаем каретку к массиву)
2. → 1
3. → 4 (команды 3 и 4 — передвигаем каретку к концу массива)
4. ? 5; 3
5. V 6 (команды 5–7 — ставим 2 метки в конце массива)
6. → 7
7. V 8
8. !

2. Даны два массива меток, которые находятся на некотором расстоянии друг от друга. Требуется соединить их в один массив. Каретка находится над крайней левой меткой первого массива. (сложение двух чисел)

Решение.

1. X 2	6. ? 8; 7
2. → 3	7. !
3. ? 4; 2	8. ← 9
4. V 5	9. ? 10; 8
5. → 6	10. → 1

3. На ленте задана последовательность массивов, включающая в себя один и более массивов. При этом два соседних массива отделены друг от друга одной пустой ячейкой. Необходимо на ленте оставить один массив длиной равной сумме длин массивов, присутствовавших изначально. Каретка находится над крайней левой меткой первого (левого) массива.

Решение.

1. X 2	6. ? 10; 7
2. → 3	7. ← 8
3. ? 4; 2	8. ? 9; 7
4. V 5	9. → 1
5. → 6	10. !

4. На ленте заданы два массива — *m* и *n*, *m* > *n*. Вычислить разность этих массивов. Каретка располагается над левой ячейкой правого массива.

Решение:

1. → 2 (команды 1–3: ищем левую метку массива *m*)
2. ? 3; 1
3. ← 4
4. X 5 (стираем левую метку массива *m*)
5. ? 6; 7
6. → 5
7. X 8 (стираем левую метку массива *n*)
8. → 9
9. ? 12; 10 (стерли последнюю метку в массиве *n*?)
10. ←11 (ищем левый край массива *m*)
11. ? 10; 4
12. !

5. На ленте задан массив. Удвоить массив в два раза. Каретка располагается над первой ячейкой массива.

Решение.

В результате работы программы справа от исходного массива будет сформирован

1. ← 2	9. V 10
2. ? 3; 1	10. → 11
3. → 4	11. V 12
4. X 5	12. ← 13
5. → 6	13. ? 14; 12
6. ? 7; 5	14. ← 15
7. → 8	15. ? 16; 1
8. ? 9; 7	16. ! 16

новый массив удвоенной длины, исходный массив будет стерт.

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ № 6. МАШИНА ПОСТА

Вариант 1. На ленте имеется некоторое множество меток (общее количество меток не менее 1). Между метками множества могут быть пропуски, длина которых составляет одну ячейку. Заполнить все пропуски метками.

Вариант 2. На ленте имеется массив из *n* отмеченных ячеек. Каретка обозревает крайнюю левую метку. Справа от данного массива на расстоянии в *m* ячеек находится еще одна метка. Составьте для машины Поста программу, придвигающую данный массив к данной ячейке.

Вариант 3. Известно, что на ленте машины Поста находится метка. Напишите программу, которая находит ее.

Вариант 4. Дан массив меток. Каретка располагается где-то над массивом, но не над крайними метками. Стереть все метки, кроме крайних, и поставить каретку в исходное положение.

Вариант 5. На ленте машины Поста расположено *n* массивов меток, отделенных друг от друга свободной ячейкой. Каретка находится над крайней левой меткой первого массива. Определить количество массивов.

Вариант 6. На ленте расположены два массива разной длины. Каретка обозревает крайний элемент одного из них. Составьте программу для машины Поста, сравнивающую длины массивов и стирающую больший из них. Отдельно продумайте случай, когда длины массивов равны.

Вариант 7. На ленте машины Поста находятся два массива в *m* и *n* меток. Составить программу выяснения, одинаковы ли массивы по длине.

Вариант 8. Дано *N* массивов меток. Массивы разделены тремя пустыми ячейками. Количество меток в массиве не меньше двух. Если количество меток в массиве кратно трем, то стереть метки в этом массиве через одну, в противном случае стереть весь массив. Каретка находится над крайней левой меткой первого массива.

Вариант 9. На ленте машины Поста расположен массив из *n* меток. Составить программу, действуя по которой машина выяснит, делится ли число *n* на 3. Если да, то после массива через одну пустую ячейку поставить метку.

Вариант 10. Дано несколько массивов меток. Удалить четные массивы. Каретка находится над первым массивом.

ЛАБОРАТОРНАЯ РАБОТА №7. МАШИНА ТЬЮРИНГА

Абстрактная вычислительная машина, предложенная в 1936 году А. Тьюрингом для уточнения понятия алгоритма. Доказано, что машина Тьюринга по своим возможностям эквивалентна машине Поста.

Состав Машины Тьюринга.

Машина Тьюринга состоит из каретки и бесконечной ленты, разбитой на ячейки. Каждая ячейка ленты может содержать символ из некоторого алфавита $A=\{a_0,a_1,\ldots,a_N\}$.

Любой алфавит содержит символ «пробел», который обозначается как a_0 или Λ. При вводе команд пробел заменяется знаком подчеркивания «_».

Машина Тьюринга — это автомат,управляемый таблицей. Строки в таблице соответствуют символам выбранного алфавита **A**, а столбцы — состояниям автомата $Q=\{q_0,q_1,\ldots,q_M\}$. В начале работы машина Тьюринга находится в состоянии q_1. Состояние q_0 — это конечное состояние: попав в него, автомат заканчивает работу.[11]

В каждой клетке таблицы, соответствующей некоторому символу a_i и некоторому состоянию q_j, находится команда, состоящая из трех частей:

1. символ из алфавита **A**;
2. направление перемещения: > (вправо), < (влево) или **.** (на месте);
3. новое состояние автомата.

Примеры решения задач с помощью машины Тьюринга рассмотрим далее.

1. Составить алгоритм, прибавляющий единицу к последней цифре заданного числа, расположенного на ленте. Входные данные – слово – цифры целого десятичного числа, записанные в последовательные ячейки на ленту. В первоначальный момент устройство располагается напротив самого правого символа – цифры числа.

Решение. В случае если последняя цифра равняется 9, то ее нужно заменить на 0 и затем прибавить единицу к предшествующему символу. Программа в этом случае для данного устройства Тьюринга может быть написана так:

	a_0	0	1	2	3	…	7	8	9
q_1	1 Н q_0	1 Н q_0	2 Н q_0	3 Н q_0	4 Н q_0		8 Н q_0	9 Н q_0	0 λ q_0

Здесь q_1 — состояние изменения цифры, q_0 — остановка. Если в q_1 автомат фиксирует элемент из ряда 0..8, то он замещает ее на один из 1..9 соответственно и затем переключается в состояние q_0, то есть устройство останавливается. В случае если же каретка фиксирует число 9, то замещает ее на 0, затем перемещается влево, останавливаясь в состоянии q_1. Такое движение продолжается до того момента, пока устройство не зафиксирует цифру, меньшую 9. Если все символы оказались равными 9, они замещаются нулями, на месте старшего элемента запишется 0, каретка переместится влево и запишет 1 в пустую клетку. Следующим шагом будет переход в состояние q_0– остановка.

2. Дан ряд из символов, обозначающих открывающие и закрывающие скобки. Требуется создать программу для машины Тьюринга, которая выполняла бы удаление пары взаимных скобок, то есть элементов, расположенных подряд – “( )”. Например, исходные данные: “) (( ) (( )”, ответ должен быть таким: “) . . . ((”.

Решение.Механизм, находясь в q_1, анализирует крайний слева элемент в строке.

Состояние q_1: если встречен символ “(”, то совершить сдвиг вправо и переход в положение q_2; если определен “a_0”, то остановка.

Состояние q_2: проводится анализ скобки “(” на наличие парности, в случае совпадения должно получиться “)”. Если элемент парный, то сделать возврат каретки влево и перейти в q_3.

Состояние q_3: осуществить удаление сначала символа “(”, а затем “)” и перейти в q_1.

	a_0	(	)
q_1	a_0Нq_0	(Пq_2	) Пq_1
q_2	a_0Нq_0	(Пq_2	) λq_3
q_3	a_0 Н q_0	a_0П q_3	a_0 П q_1

Для проверки и отладки программ для машины Поста и машины Тьюринга можно использовать тренажёр «Машина Поста» и «Машина Тьюринга». В Интернет можно найти свободно распространяемые имитаторы как машины Поста, так и машины Тьюринга. [9]

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №7. МАШИНА ТЬЮРИНГА

Вариант 1.На ленте машины Тьюринга содержится последовательность символов “+”. Напишите программу для машины Тьюринга, которая каждый второй символ “+” заменит на “–”. Замена начинается с правого конца последовательности. Автомат в состоянии q_1 обозревает один из символов указанной последовательности. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Вариант 2.Дано число n в восьмеричной системе счисления. Разработать машину Тьюринга, которая увеличивала бы заданное число n на 1. Автомат в состоянии q_1 обозревает некую цифру входного слова. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Вариант 3.Дана десятичная запись натурального числа $n > 1$. Разработать машину Тьюринга, которая уменьшала бы заданное число n на 1. Автомат в состоянии q_1 обозревает правую цифру числа. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Вариант 4.Дано натуральное число $n > 1$. Разработать машину Тьюринга, которая уменьшала бы заданное число n на 1, при этом в выходном слове старшая цифра не должна быть 0. Например, если входным словом было “100”, то выходным словом должно быть “99”, а не “099”. Автомат в состоянии q_1 обозревает правую цифру числа. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Вариант 5. Дан массив из открывающих и закрывающих скобок. Построить машину Тьюринга, которая удаляла бы пары взаимных скобок, т.е. расположенных подряд “( )”. Например, дано “) (( ) (( )”, надо получить “) . . . ((”. Автомат в состоянии q_1 обозревает крайний левый символ строки. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Вариант 6. Дана строка из букв “a” и “b”. Разработать машину Тьюринга, которая переместит все буквы “a” в левую, а буквы “b” — в правую части строки. Автомат в состоянии $q1$ обозревает крайний левый символ строки. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Вариант 7.На ленте машины Тьюринга находится число, записанное в десятичной системе счисления. Умножить это число на 2. Автомат в состоянии q_1 обозревает крайнюю левую цифру числа. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Вариант 8. Даны два натуральных числа m и n, представленные в унарной системе счисления. Соответствующие наборы символов “|” разделены пустой клеткой. Автомат в состоянии q_1обозревает самый правый символ входной последовательности. Разработать машину Тьюринга, которая на ленте оставит сумму чисел m и n. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Вариант 9.Даны два натуральных числа m и n, представленных в унарной системе счисления. Соответствующие наборы символов “|” разделены пустой клеткой. Автомат в состоянии q_1 обозревает самый правый символ входной последовательности. Разработать машину Тьюринга, которая на ленте оставит разность чисел m и n. Известно, что $m > n$. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

Вариант 10.На ленте машины Тьюринга находится десятичное число. Определить, делится ли это число на 5 без остатка. Если делится, то записать справа от числа слово “да”, иначе — “нет”. Автомат обозревает некую цифру входного числа. Кроме самой программы-таблицы, описать словами, что выполняется машиной в каждом состоянии.

ЛАБОРАТОРНАЯ РАБОТА №8. ВСПОМОГАТЕЛЬНЫЕ АЛГОРИТМЫ. ПРОЦЕДУРЫ И ФУНКЦИИ

В современном программировании применяется технология нисходящего программирования «сверху вниз». Суть этой технологии заключается дробления сложной

задачи на более простые подзадачи (декомпозиция) до тех пор, пока не прояснятся все детали решения. Такие подзадачи оформляются в виде вспомогательных алгоритмов.

Реализация вспомогательных алгоритмов осуществляется посредством подпрограмм.

Определение 5. Подпрограмма – обособленная, оформленная в виде отдельной синтаксической конструкции и снабженная именем часть программы. Использование подпрограмм позволяет, сосредоточив в них подробное описание некоторых операций, в остальной программе указывать только имена подпрограмм, чтобы выполнить эти операции. [7]

Подпрограммы в процедурно-ориентированных языках программирования представляют собой процедуры и функции. Имея один и тот же смысл и аналогичную структуру, процедуры и функции различаются назначением и способом их использования.

*Определение 6.*Процедура – независимая именованная часть программы, которую можно вызвать по имени для выполнения определенных действий.

*Определение 7.*Функция – подпрограмма, которая передает в точку вызова скалярное значение.

Рассмотрим технологию создания и использования процедур и функций, руководствуясь учебником Окулова С.М. «Основы программирования».

Процедуры.

Структура процедуры схожа со структурой программы.

```
Procedure <Имя процедуры> (<параметры>) ;
Label <метки> ;
Const <описание типов данных> ;
Var <описание переменных> ;
Begin
<основное тело процедуры> ;
End.
```

Вызов процедуры

Для наглядности объяснения, обратимся к примеру:

```
Procedure Sum(x,y:Integer;Var z:Integer) ;
Begin
z:=x+y
End;
...
ReadLn(a,b,c);
Sum(a,b,c);
WriteLn(c);
...
End.
```

В данном фрагменте программы вызывается процедура вычисления суммы двух целых чисел. Процедура вызывается по имени.

При вызове процедуры Sum(a,b,c) из основной программы значение переменной aприсваивается переменной x процедуры *Sum*, а значение переменной x – переменной y.

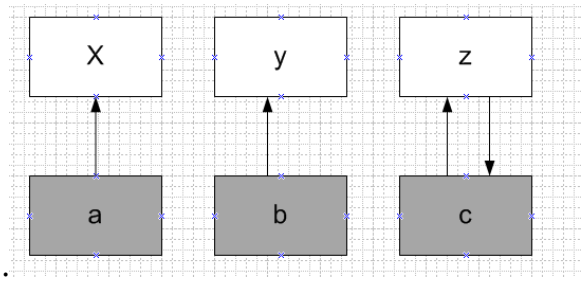


Рис. 11. Соответствие параметров процедуры

В любой программе все переменные делятся на глобальные и локальные.

```
Program TR;
Var k: Integer;
    p: Real;
Procedure Sum;
Var  d: Char;
     f: Integer;
  Begin
  ...
  End;
  ...
End.
```

В данном фрагменте программы переменные k, p – глобальные, ad,f - локальные. Глобальные переменные описываются в разделе описаний основной части программы, а локальные – только в том, где описываются переменные. Локальные переменные существуют только во временя работы процедуры, определяются (создаются) при её вызове и «исчезают» после завершения работы процедуры.

При описании процедуры указывается список формальных параметров. Каждый параметр является локальным, к нему можно обращаться только в пределах данной процедуры (в примере x,y,z - формальные параметры, рис.11). Фактические параметры – это параметры, которые передаются процедуре при обращении к ней. (a,b,c – фактические параметры, рис.11). Количество и типы формальных и фактических параметров должны совпадать.

Параметры-значения**,** т.е. передача параметров по значению. При таком способе передачи параметров значение фактического параметра становится значением соответствующего формального параметра. Внутри процедуры можно производить любые действия с данным формальным параметров (допустимые для его типа), но эти изменения никак не отражаются на значении фактического параметра, т. е. каким он был до вызова процедуры, таким же и останется после завершения её работы (x,y – формальные параметры-значения).

Параметры-переменные - формальные параметры, перед которыми стоит служебное слово Var. При таком способе передачи параметров в процедуру передаётся не значение, а адрес фактического параметра (обязательно переменной). Любые операции с формальным параметром выполняются непосредственно над фактическим параметром. [6]

Функции.

Структура функции.

```
Function <имя функции> {(список параметров)} :
<тип результата>;
```

В теле функции обязательно должен быть хотя бы один оператор присвоения, где в левой части стоит имя функции, а в правой — ее значение. Иначе значение функции не будет определено. Обращение к функции осуществляется по имени с необязательным указанием списка аргументов. Каждый аргумент должен соответствовать формальным параметрам, указанным в заголовке, и иметь тот же тип.

В качестве примера приведем алгоритм вычисления выражения

$Z=(A^5+A^{-3})/2*A^M$, в котором возведение в степень выполняется функцией Step.[7] На рис.12 представлена блок-схема основного алгоритма.

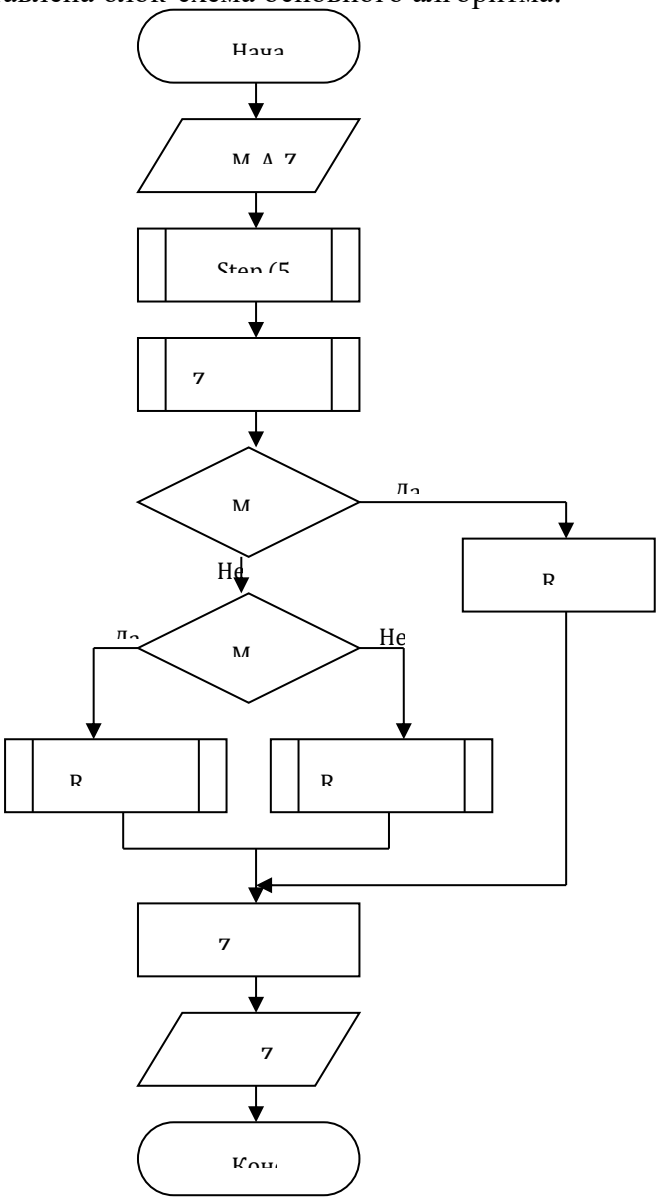


Рис.12 Алгоритм вычисления выражения $Z=(A^5+A^{-3})/2*A^M$

В первом разделе «Основы алгоритмизации» рассмотрены понятия алгоритм, свойства алгоритма, способы представления алгоритмов. Виртуальные машины Поста и Тьюринга иллюстрируют формализацию алгоритмов, а так же описаны вспомогательные алгоритмы на примерах процедур и функций.

Для успешного усвоения материала рекомендуется выполнить задания для самостоятельной работы в соответствии с номером своего варианта. В подборке заданий для данного пособия использованы материалы следующих источников [3], [7],[8].

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №8. ВСПОМОГАТЕЛЬНЫЕ АЛГОРИТМЫ. ПРОЦЕДУРЫ И ФУНКЦИИ.

Вариант 1. В квадратной матрице размером 5x5, заполненной случайными целыми числами из диапазона (–45,+45) найти количество положительных элементов.

Вариант 2. В квадратной матрице размером 5x5, заполненной случайными целыми числами из диапазона (–40,+40) найти соответственно максимальный отрицательный и минимальный положительный элементы.

Вариант 3. Описать процедуру Minmax(X, Y), записывающую в переменную X минимальное из значений X и Y, а в переменную Y — максимальное из этих значений (X и Y — вещественные параметры, являющиеся одновременно входными и выходными). Используя четыре вызова этой процедуры, найти минимальное и максимальное из данных чисел A, B, C, D.

Вариант 4.Даны три квадратных матрицы A, B, C n-го порядка. Вывести на печать ту из них, норма которой наименьшая. Пояснение. Нормой матрицы назовем максимум из абсолютных величин ее элементов.

Вариант 5. Пусть даны две вещественные матрицы порядка n. Получите новую матрицу прибавлением к элементам каждого столбца первой матрицы, произведения элементов соответствующих строк второй матрицы.

Вариант 6. Пусть даны две вещественные матрицы порядка n. Получите новую матрицу умножением минимального элемента каждой строки первой матрицы на наибольший элемент соответствующего столбца второй матрицы.

Вариант 7.Описать процедуру Mean(X, Y, AMean, GMean), вычисляющую среднее арифметическое AMean = (X + Y)/2 и среднее геометрическое GMean = sqrt(XY) двух положительных чисел X и Y (X и Y — входные, AMean и GMean — выходные параметры вещественного типа). С помощью этой процедуры найти среднее арифметическое и среднее геометрическое для пар (A, B), (A, C), (A, D), если даны A, B, C, D.

Вариант 8.Описать процедуру TrianglePS(a, P, S), вычисляющую по стороне a равностороннего треугольника его периметр P и площадь S. (a — входной, P и S — выходные параметры; все параметры являются вещественными). С помощью этой процедуры найти периметры и площади трех равносторонних треугольников с данными сторонами.

Вариант 9.Описать процедуру DigitCountSum(K, C, S), находящую количество C цифр целого положительного числа K, а также их сумму S (K — входной, C и S — выходные параметры целого типа). С помощью этой процедуры найти количество и сумму цифр для каждого из пяти данных целых чисел.

Вариант 10.Описать процедуру Swap(X, Y), меняющую содержимое переменных X и Y (X и Y — вещественные параметры, являющиеся одновременно входными и выходными). С ее помощью для данных переменных A, B, C, D последовательно поменять содержимое следующих пар: A и B, C и D, B и C и вывести новые значения A, B, C, D.

ЛАБОРАТОРНАЯ РАБОТА №9. РЕКУРСИВНЫЕ МЕТОДЫ ПОСТРОЕНИЯ АЛГОРИТМОВ

Определение 6. Рекурсия – способ организации вычислительного процесса, при котором функция в ходе выполнения составляющих ее операторов обращается сама к себе.[7]

Использование рекурсии сокращает программу, в этом ее преимущество перед использованием итерационных циклов, но при выполнении может вызвать переполнение оперативной памяти компьютера.

Вид рекурсивной процедуры:

```
Procedure Rec (t:Integer);
Begin
<действия на вход в рекурсию>; -
if <Проверка условий> Then Rec (t-1);
<действия на выходе из рекурсии>
End;
```

Рассмотрим типовой пример задачи вычисления факториала числа.

```
Function Factorial ( n: Integer) : Longint;
Begin
  If n=1 Then Factorial:=1
         Else Factorial:=n* Factorial (n-1);
End;
```

На рис. 13 показаны прямой и обратный ход рекурсии. [6] В любой рекурсивной подпрограмме обязательно должна быть нерекурсивная ветвь:

```
If n = 1 Then Factorial := 1
```

Пусть переменной *a* присваивается значение 5!:

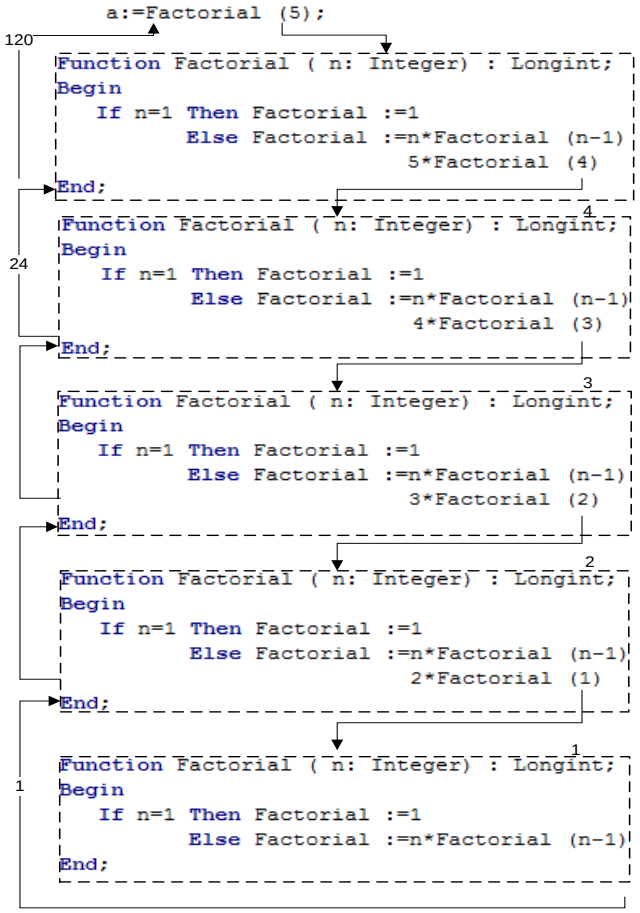


Рис. 13. Прямой и обратный ход рекурсии.

Рассмотрим некоторые примеры использования рекурсии.[6]

1. Сложение двух чисел $a + b$. Рекурсивное определение операции сложения двух чисел:

$$a + b = \begin{cases} a \text{ при } b = 0, \\ (a+1) + (b-1) \text{ при } b > 0 \\ (a-1) + (b+1) \text{ при } b < 0 \end{cases}$$

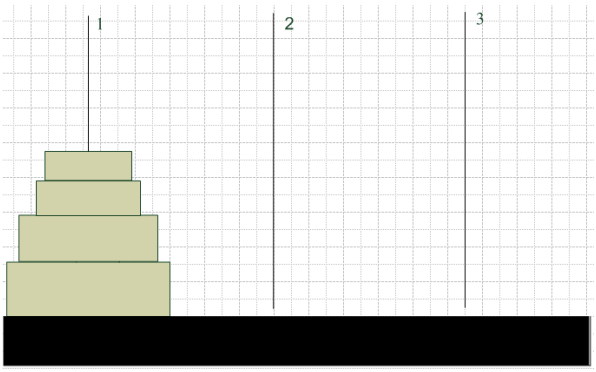
```
Function Sum ( a, b: Integer) : Integer;
 Begin
  If b=0 Then Sum:=a
         Else If b>0 Then Sum:=Sum (a+1, b-1)
                     Else Sum:=Sum (a-1, b+1)
 End;
```

2. Каждое число Фибоначчи равно сумме двух предыдущих чисел при условии, что первые два равны 1 (1, 1, 2, 3, 5, 8, 13, 21,…).В общем виде n-е число Фибоначчи можно определить так:

$$\Phi(n)=\begin{cases} 1 \text{ при } n=1 \text{ или } n=2, \\ \Phi(n-1)+\Phi(n-2) \text{при } n>2. \end{cases}$$

```
Function Fib (n:Integer) :Integer;
 Begin
  If n<=2 Then Fib:=1
          Else Fib:=Fib (n-1) +Fib (n-2) ;
 End;
```

3. «Ханойские башни». В конце XIX века в Европе появилась игра под названием «Ханойские башни». Реквизит игры состоит из 3 игл, на которых размещается башня из колец. Цель игры – перенести башню с левой иглы (1) на правую (3), причем за один раз можно перенести только одно кольцо. Кроме того, запрещается помещать большее кольцо над меньшим.



Программа, которая печатает последовательность переноса колец посредством рекурсивной процедуры, представлена вПриложении 1, Hanoy.pas.

Данная задача своим происхождением обязана легенде, в которой рассказывается, что в большом храме Бенареса на бронзовой плите установлены три алмазных стержня, на один из которых бог нанизал во время сотворение мира 64 золотых диска. С тех пор день и ночь монахи, сменяя друг друга каждую секунду, перекладывают по одному диску согласно описанным выше правилам. Конец мира наступит тогда, когда все 64 диска будут перемещены, на что потребуется чуть больше 58 млрд. лет.

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №9. РЕКУРСИВНЫЕ МЕТОДЫ ПОСТРОЕНИЯ АЛГОРИТМОВ

Вариант 1.Описать рекурсивную функцию DigitSum(K) целого типа, которая находит сумму цифр целого числа K без использования оператора цикла.

Вариант 2.Вычислить сумму: $1! + 2! + 3! + \cdots + n!$, используюя функцию вычисления факториала числа $k!$

Вариант 3.Описать рекурсивную функцию SqrtK(X, K, N) вещественного типа, находящую приближенное значение корня k-й степени из числа X по формуле:

$$Y_0 = 1, \qquad Y_{N+1} = Y_N - (Y_N - X/(Y_N)^{K-1})/K,$$

где Yn обозначает SqrtK(X, K, N) при фиксированных X и K. Параметры функции: X > 0 — вещественное число, K > 1 и N > 0 — целыт/

Вариант 4.Описать рекурсивную функцию NOD(A, B) целого типа, находящую наибольший общий делитель (НОД) двух целых положительных чисел A и B, используя алгоритм Евклида: NOD(A, B) = NOD(BmodA,B), если A<> 0; НОД(0, B) = B.

Вариант 5.Описать рекурсивную функцию$Fib(N)$целого типа, вычисляющую$N-$й элемент последовательности чисел Фибоначчи (N — целое число):Считать, что номер $N \leq 20$. Для уменьшения количества рекурсивных вызовов по сравнению с функцией создать вспомогательный массив для хранения уже вычисленных чисел Фибоначчи и обращаться к нему при выполнении функции Fib.

Вариант 6.Описать рекурсивную функцию Fact2(N) вещественного типа, вычисляющую значение двойного факториала N!! = N·(N-2)·(N-4)·. . . ., где N > 0 — параметр целого типа; последний сомножитель в произведении равен 2, если N — четное число, и 1, если N — нечетное.

Вариант 7.Напишите рекурсивную функцию SumTo(n), которая для данного n вычисляет сумму чисел от 1 don.

ЛАБОРАТОРНАЯ РАБОТА № 10,11. МЕТОДЫ СОРТИРОВКИ ДАННЫХ

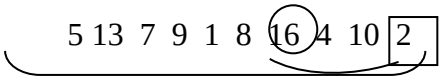
По методам сортировки, а также по методам поиска данных существует очень много публикаций.

Сортировка простым выбором

Рассмотрим идею на примере. Пусть исходный массив A состоит из 10 элементов: 5 13 7 9 1 8 16 4 10 2. После сортировки массив должен выглядеть так: 1 2 4 5 7 8 9 10 13 16.

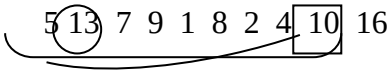
Процесс сортировки описан ниже. Максимальный элемент текущей части массива заключен в кружок, а элемент, с которым происходит обмен, в квадратик. Скобкой помечена рассматриваемая часть массива.

1-й шаг. Рассмотрим весь массив и найдем в нем максимальный элемент – 16 (на седьмом месте), поменяем его местами с последним элементом – с числом 2.



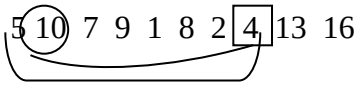
Максимальный элемент записан на свое место.

2-й шаг. Рассмотрим часть массива – с первого до девятого элемента. Максимальный элемент этой части – 13 (на втором месте). Поменяем его местами с последним элементом этой части – с числом 10.



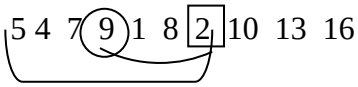
Отсортированная часть массива имеет два элемента.

3-й шаг. Уменьшим рассматриваемую часть массива на один элемент. Здесь нужно поменять местами второй элемент (его значение – 10) и последний элемент этой части – число 4.

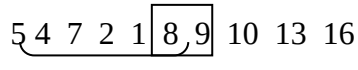


В отсортированной части массива – 3 элемента.

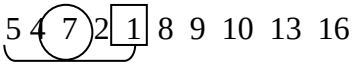
4-й шаг.



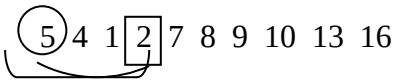
5-й шаг. Максимальный элемент этой части массива является последним в ней, поэтому его нужно оставить на старом месте.



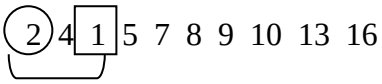
6-й шаг.



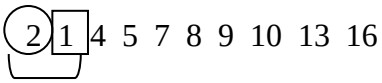
7-й шаг.



8-й шаг.



9-й шаг.



На рис. 14.представлена блок-схема алгоритма сортировки простым выбором.

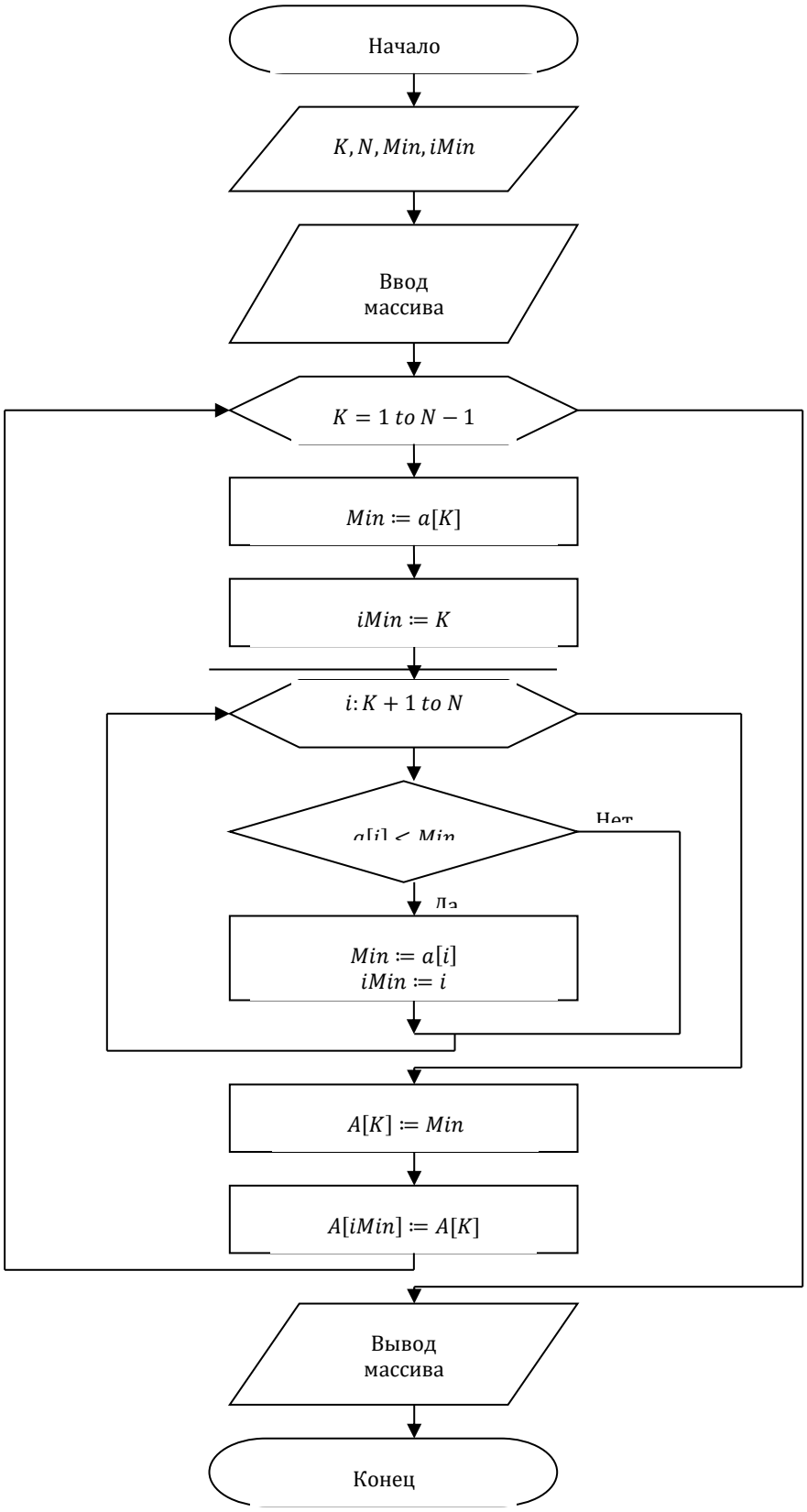


Рис. 14. Алгоритм сортировки методом выбора

2. Сортировка простым обменом. Метод «пузырька»

Рассмотрим идею метода на примере. Отсортируем по возрастанию массив из 5 элементов: 5 4 8 2 9. Длина текущей части массива – N-k+1, где k - номер просмотра, I – номер первого элемента проверяемой пары, номер последней пары – N-k.

Первый просмотр*:* рассматривается весь массив.

I=1 5 4 8 2 9

> меняем

I=2 4 5 8 2 9

< не меняем

I=3 4 5 8 2 9

> меняем

I=4 4 5 2 8 **9**

< не меняем

Цифра 9 находится на своем месте.

Второй просмотр:рассматриваем часть массива с первого до предпоследнего элемента.

I=1 4 5 2 8 9

< не меняем

I=2 4 5 2 8 9

> меняем

I=3 4 2 5 **8 9**

< не меняем

Цифра 8 на своем месте.

Третий просмотр:рассматриваемая часть массива содержит три первых элемента.

I=1 4 2 5 8 9

> меняем

I=2 2 4 **5 8 9**

< не меняем

Цифра 5 на своем месте.

Четвертый просмотр:рассматриваем последнюю пару элементов.

I=1 2 **4 5 8 9**

< не меняем

Цифра 4 на своем месте. Наименьший элемент – 2 оказывается на первом месте. Количество просмотров элементов массива равно N-1.

Итак, массив отсортирован. Этот метод также называют методом «пузырька», потому что при выполнении сортировки более «легкие» элементы (элементы с заданным свойством) постепенно всплывают на «поверхность».

На рис. 15. представлена блок-схема алгоритма сортировки методом «пузырька».

Сортировка простыми вставками.

Сортировка этим методом производится последовательно шаг за шагом. На $i-$м шаге считается, что часть массива, содержащая первые$(i-1)$элементов, уже упорядочена, то есть$A[1] \le A[2] \le \cdots A[i-1]$. Далее следует взять$i-$й элемент, и для него найти место в отсортированной части массива, такое, что после его вставки упорядоченность не нарушится, то есть требуется найти такоеj $(0 \le j \le i-1)$, что $A[j] \le A[i] \le A[j+1]$,(при $j=0$происходит только одно сравнение). Затем выполняется вставка элемента $A[i]$на место j. На каждом шаге отсортированная часть массива увеличивается. Для выполнения полной сортировки потребуется выполнить $(n-1)$шаг.

Рассмотрим этот процесс на примере. Таблица 3. Пусть требуется отсортировать массив из 10 элементов по возрастанию.

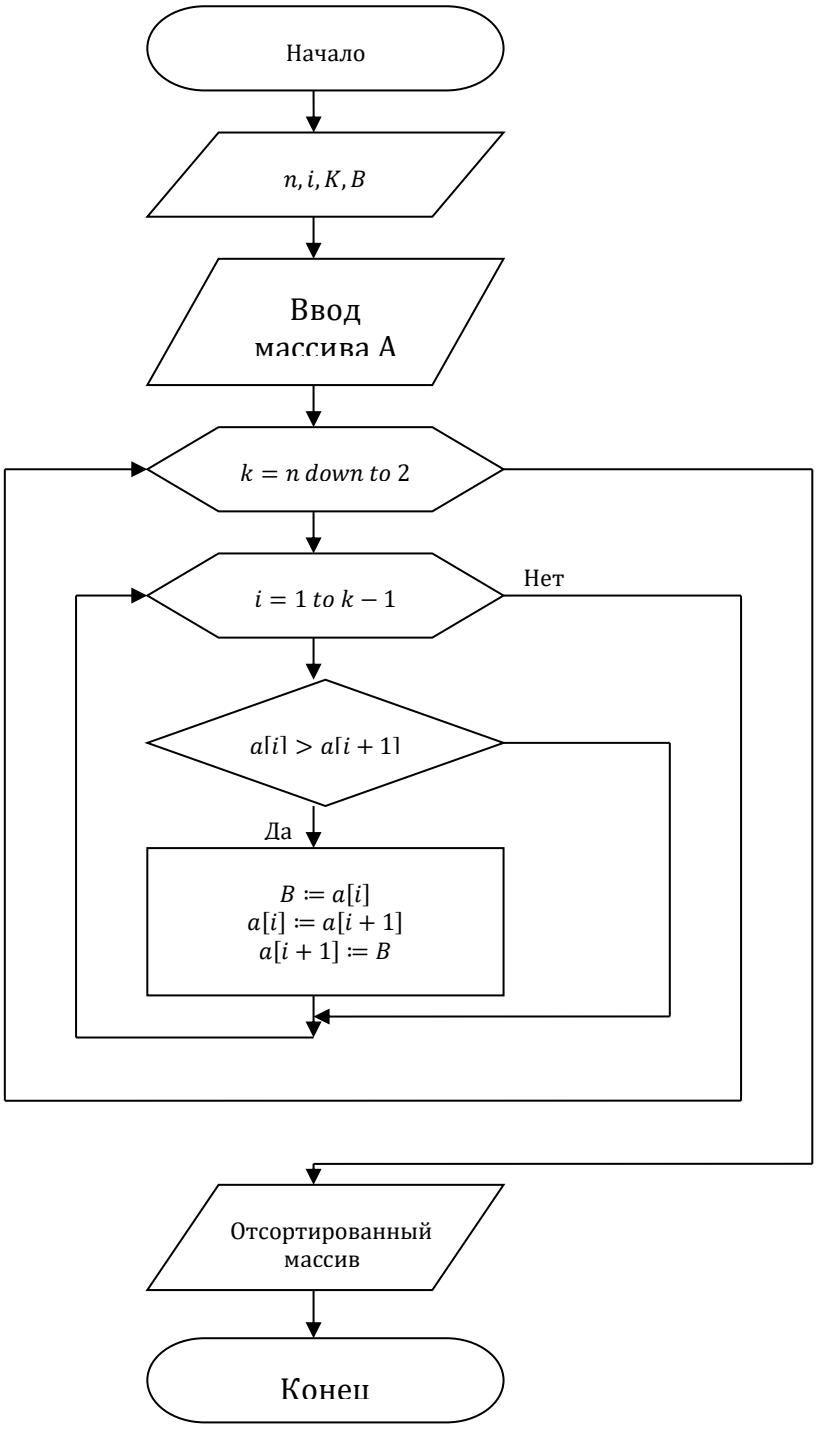


Рис. 15. Алгоритм сортировки методом «пузырька»

Таблица 3. Сортировка вставками

1-й шаг.	13 6 8 11 3 1 5 9 15 7	Рассматриваем часть массива из одного элемента 13. Вставляем второй элемент массива 6 так, чтобы упорядоченность сохранилась. Так как 6<13, записываем 6 на первое место. Отсортированная часть массива содержит два элемента (6 13).
2-й шаг.	6 13 8 11 3 1 5 9 15 7	Возьмем третий элемент массива 8 и подберем для него место в упорядоченной части массива. 8>6 и 8<13, следовательно, его место второе.
3-й шаг.	6 8 13 11 3 1 5 9 15 7	Следующий элемент – 11. Он записывается в упорядоченную часть массива на третье место, так как 11>8, но 11<13.
4-й шаг.	6 8 11 13 3 1 5 9 15 7	Далее, действуя аналогичным образом, определяем, что 3 необходимо записать на первое место.
5-й шаг.	3 6 8 11 13 1 5 9 15 7	По той же причине 1 записываем на первое место.
6-й шаг.	1 3 6 8 11 13 5 9 15 7	Так как 5>3, но 5<6, то место 5 в упорядоченной части – третье.
7-й шаг.	1 3 5 6 8 11 13 9 15 7	Место числа 9 – шестое.
8-й шаг.	1 3 5 6 8 9 11 13 15 7	Определяем место для предпоследнего элемента 15. оказывается, что этот элемент массива уже находится на своем месте.
9-й шаг.	1 3 5 6 8 9 11 13 15 7	Осталось подобрать подходящее место для последнего элемента 7.
	1 3 5 6 7 8 9 11 13 15	Массив отсортирован полностью.

Блок-схема алгоритма сортировки массива вставками на рис.16.

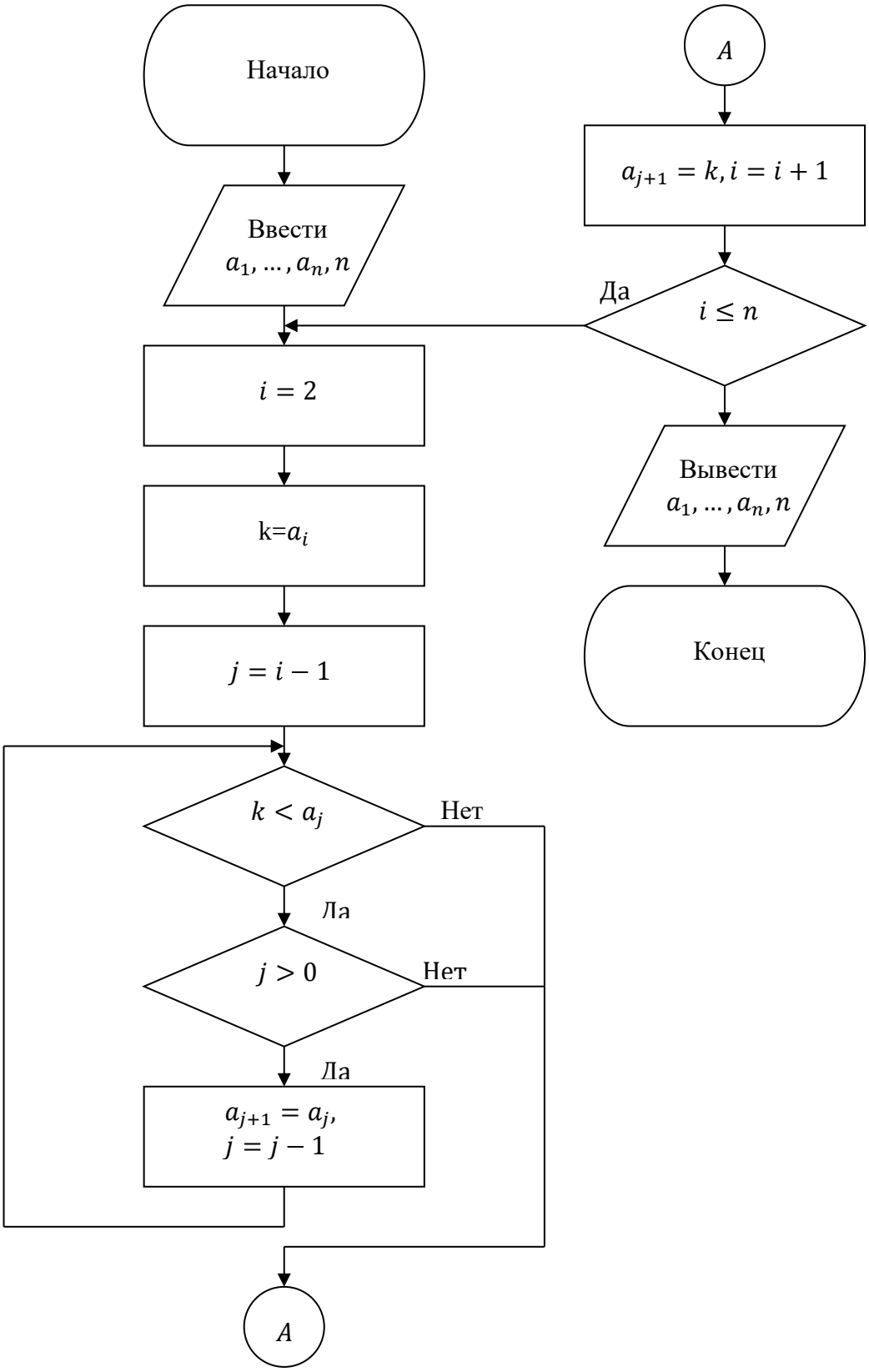


Рис.16 Алгоритм сортировки элементов массива вставками

Методы быстрой сортировки.

Метод сортировки слиянием.

Прежде, чем обсуждать метод, рассмотрим следующую задачу. Объединить («слить») упорядоченные фрагменты массива AA[k],…,A[m] и A[m+1],…,A[q] в один A[k],…,A[q], естественно, тоже упорядоченный (k<m<q). Основная идея решения состоит в сравнении

очередных элементов каждого фрагмента, выяснении, какой из элементов меньше, переносе его во вспомогательный массив D (для простоты) и продвижении по тому фрагменту массива, из которого взят элемент. При этом следует не забывать записать в D оставшуюся часть этого фрагмента, который не успел себя «исчерпать».

Рассмотрим сортировку массива методом слияния на примере.

Пример.Пусть первый фрагмент состоит из 5 элементов: 3 5 8 11 16, а второй – из 8: 1 5 7 9 12 13 18 20. Рисунок иллюстрирует логику объединения фрагментов.

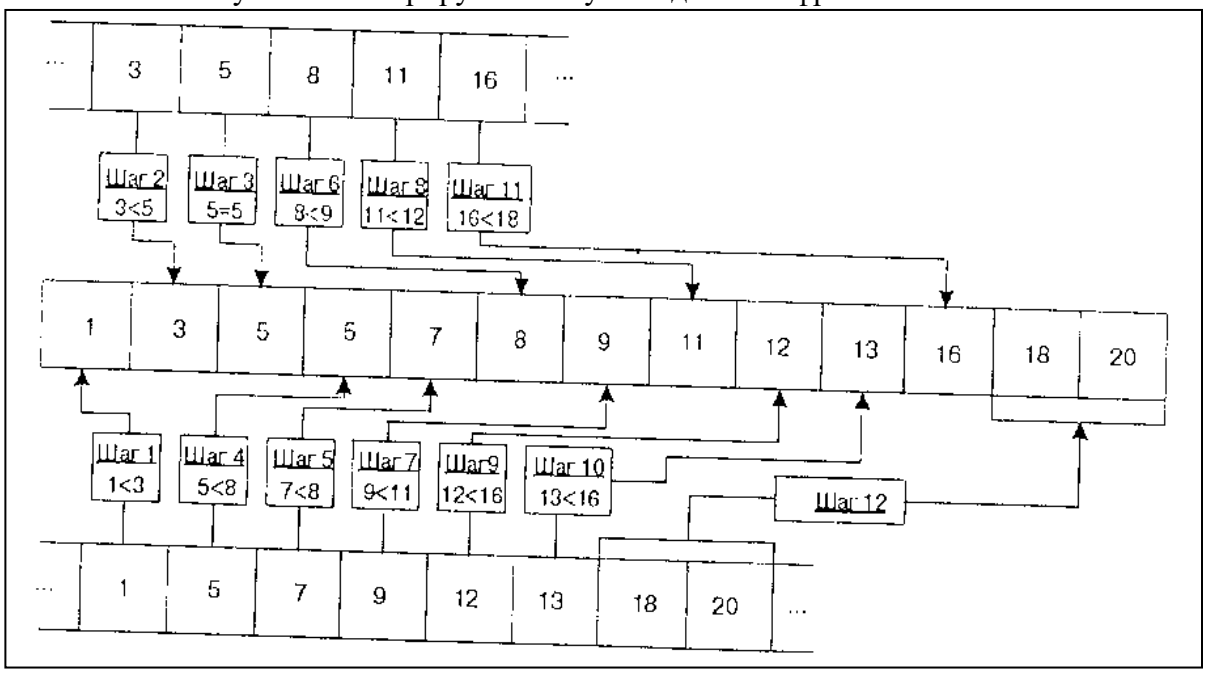


Рис. 17. Метод сортировки элементов массива слиянием

Метод слияний – один из первых в теории алгоритмов сортировки. Он предложен Дж. фон Нейманом в 1945 году. Недостатком сортировки слиянием является использование дополнительной памяти. Но при работе с файлами или списками, доступ к которым осуществляется только последовательно, очень удобно применение этого метода.[6, 10]

Метод быстрой сортировки. Сортировка Хоара.

Метод предложен Ч. Э. Р. Хоаром в 1962 году. Эффективность метода достаточно высокая, поэтому автор назвал его «быстрой сортировкой».

Идея метода. В исходном массиве A выбирается некоторый элемент X (барьерный элемент). Следует записатьX «на свое место» в массиве, например, k, такое, что слева от X были элементы массива, меньшие или равные X, а справа – элементы массива, большие X, т.е. массив A будет иметь вид: $A[1], A[2], \dots A[k], X, A[k+1], \dots A[n]$.

В результате элемент $A[k]$находится на своем месте и исходный массив Aразделен на две неупорядоченные части, барьером между которыми является элемент $A[k]$. Далее сортируем полученные части по той же логике до тех пор, пока не останутся части массива, состоящие из одного элемента, то есть пока не будет отсортирован весь массив.

Рассмотрим сортировку массива методом Хоара на примере.[6,10]

Пример**.** Исходный массив состоит из 8 элементов: 8 12 3 7 19 11 4 16. В качестве барьерного элемента возьмем средний элемент массива (7).

Произведя необходимые перестановки, получим: (4 3) 7 (12 19 11 8 16), элемент 7 находится на своем месте. Продолжим сортировку.

Левая часть: (3) 4 7 (12 19 11 8 16)

3 4 7 (12 19 11 8 16)

Правая часть:

3 4 7 (8) 11 (19 12 16)

3 4 7 8 11 (19 12 16)

3 4 7 8 11 12 (19 16)
3 4 7 8 11 12 (16) 19
3 4 7 8 11 12 16 19

Из вышеизложенного описания явно просматривается рекурсивная схема реализации, параметрами которой являются нижняя и верхняя границы изменения индексов сортируемой части исходного массива. Приведем процедуру быстрой сортировки. [7]

```
ProcedureQuickSort (m, t :Integer); {*m- начало массива, t – конец массива.*}
Var i, j, x, w : Integer;
    Begin
      I:=m;  j:=t;  x:=A [(m+t) Div 2];
      Repeat
        While A[i] < x Do Inc (i);
        While A[j] > x Do Dec (j);
         If i<=j Then Begin w:=A[i];  A[i]:=A[j];  A[j]:=w;
Inc (i); Dec (j); End
      Until i>j;
      If m<j Then QuickSort (m, j);
      If i<t Then QuickSort (I, t);
    End;
```

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ № 10,11. МЕТОДЫ СОРТИРОВКИ ДАННЫХ

Реализуйте все простые сортировки линейного массива, используя процедуры, и оцените трудоемкость каждого алгоритма.

Вариант 1. Дан массив целых чисел осуществить сортировку четных элементов массива.

Вариант 2. Дан массив целых чисел осуществить сортировку элементов массива записанных на нечетных местах.

Вариант 3. Дан массив целых чисел осуществить сортировку отрицательных элементов массива.

Вариант 4. Дан массив целых чисел осуществить сортировку положительных элементов массива. Вариант 5. В упорядоченный по возрастанию значений элементов массив M, состоящий из целых чисел, необходимо вставить число, не нарушив упорядоченности исходного массива.

Вариант 6. Задан массив M, состоящий из N целочисленных элементов. Упорядочить элементы таким образом, чтобы вначале располагались все нечетные аргументы, а после них все четные.

Вариант 7. Задан массив M, состоящий из N целочисленных элементов. Упорядочить элементы таким образом, чтобы вначале располагались все четные аргументы, а после них все нечетные.

Вариант8. Задан массив M, состоящий из N целочисленных элементов. Упорядочить элементы таким образом, чтобы вначале располагались все положительные аргументы, а после них все отрицательные.

Вариант 9. Задан массив M, состоящий из N целочисленных элементов. Упорядочить элементы таким образом, чтобы вначале располагались все отрицательные аргументы, а после них все положительные.

Вариант 10. Массив M, состоящий из 30 элементов, переформировать так, чтобы вначале стояли все положительныеэлементы в порядке убывания их значений, а затем все отрицательные элементы в порядке возрастания их значений.

ЛАБОРАТОРНАЯ РАБОТА №12. МЕТОД ПЕРЕБОРА В ЗАДАЧАХ ПОИСКА

Задачи поиска предназначены для определения нахождения элемента, обладающего заданным свойством, в определенной совокупности данных, в частности, в массиве.

Линейный поиск.

Поиск наибольшего и наименьшего элемента в массиве.

Дан ряд чисел a_1, a_2, …, a_i, …, a_n. Разработать алгоритм поиска наибольшего и наименьшего числа в этом ряду с указанием номера (индекса) его расположения.

Очевидный способ поиска наибольшего (наименьшего) числа в заданном ряду *n* чисел включает следующие действия. Взять первое число ряда и сказать, что оно наибольшее (наименьшее), а индекс его равен 1. Затем взять второе число ряда и сравнить с предполагаемым максимальным (минимальным) первым числом. Если второе число больше предполагаемого (максимального) первого числа, взять третье число ряда и сравнить с первым. Так следует действовать до тех пор, пока не будет выбрано последнее a_n число. В результате на месте первого числа окажется наибольшее (наименьшее) число с указанным его номером в ряду чисел. [2]

Блок – схема алгоритма поиска наибольшего и наименьшего элемента на рис.18.

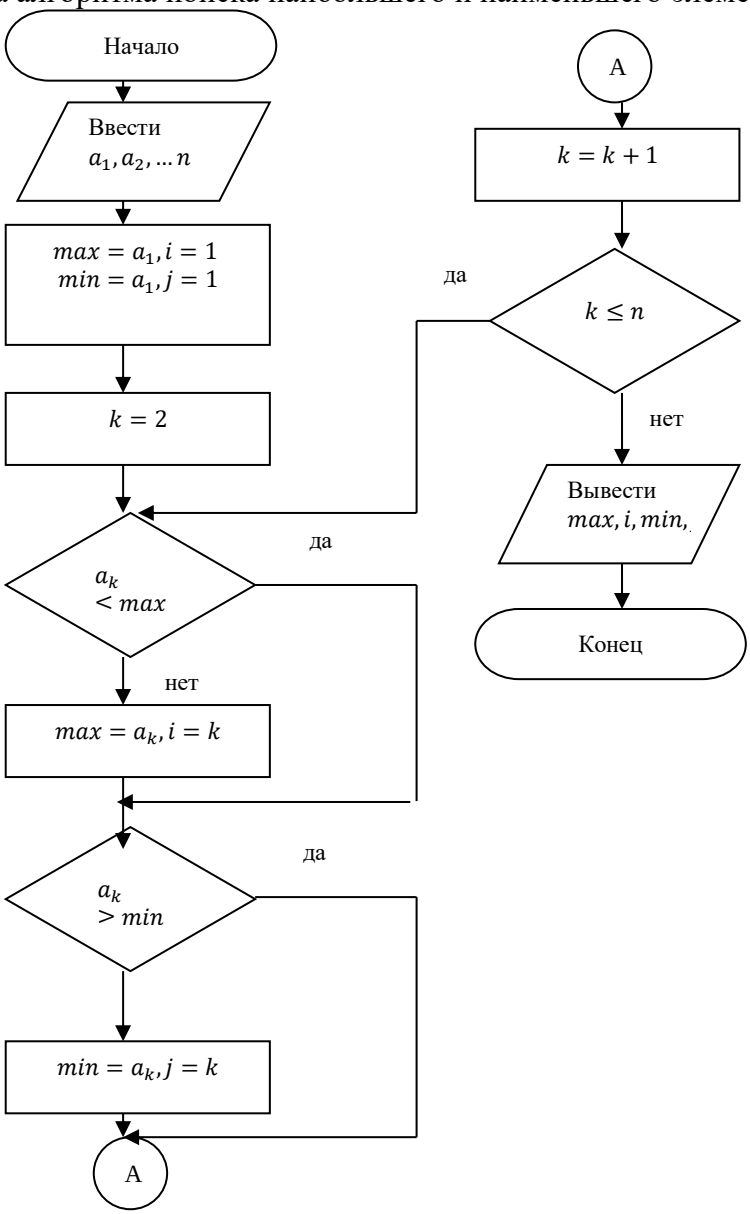


Рис. 18 Алгоритм нахождения наибольшего и наименьшего элемента в линейном массиве

Бинарный поиск.

Метод бинарного поиска можно применять уже в отсортированном массиве. Допустим, что массив А отсортирован в порядке неубывания. Это позволяет по результату сравнения со средним элементом массива исключить из рассмотрения одну из половин. С оставшейся частью процедура повторяется. И так до тех пор, пока не будет найден искомый элемент или не будет построен весь массив. [6,7]

Рассмотрим алгоритм бинарного поиска на примере.

Пример. Пусть X = 6, а массив А состоит из 10 элементов:

3 5 6 8 12 15 17 18 20 25.

1-й шаг. Найдем номер среднего элемента среднего элементов: $m = \left[\frac{1+10}{2}\right] = 5$.

Так как 6<A[5], то далее рассматриваются только элементы, индексы которых меньше 5.

3 5 6 8 ~~**12** 15 17 18 20 25.~~

2-й шаг. Рассматриваем лишь первые 4 элемента массива, находим индекс среднего элемента этой части :$m = \left[\frac{1+4}{2}\right] = 2$.

6>A[2], следовательно, первый и второй элементы из рассмотрения исключаются:

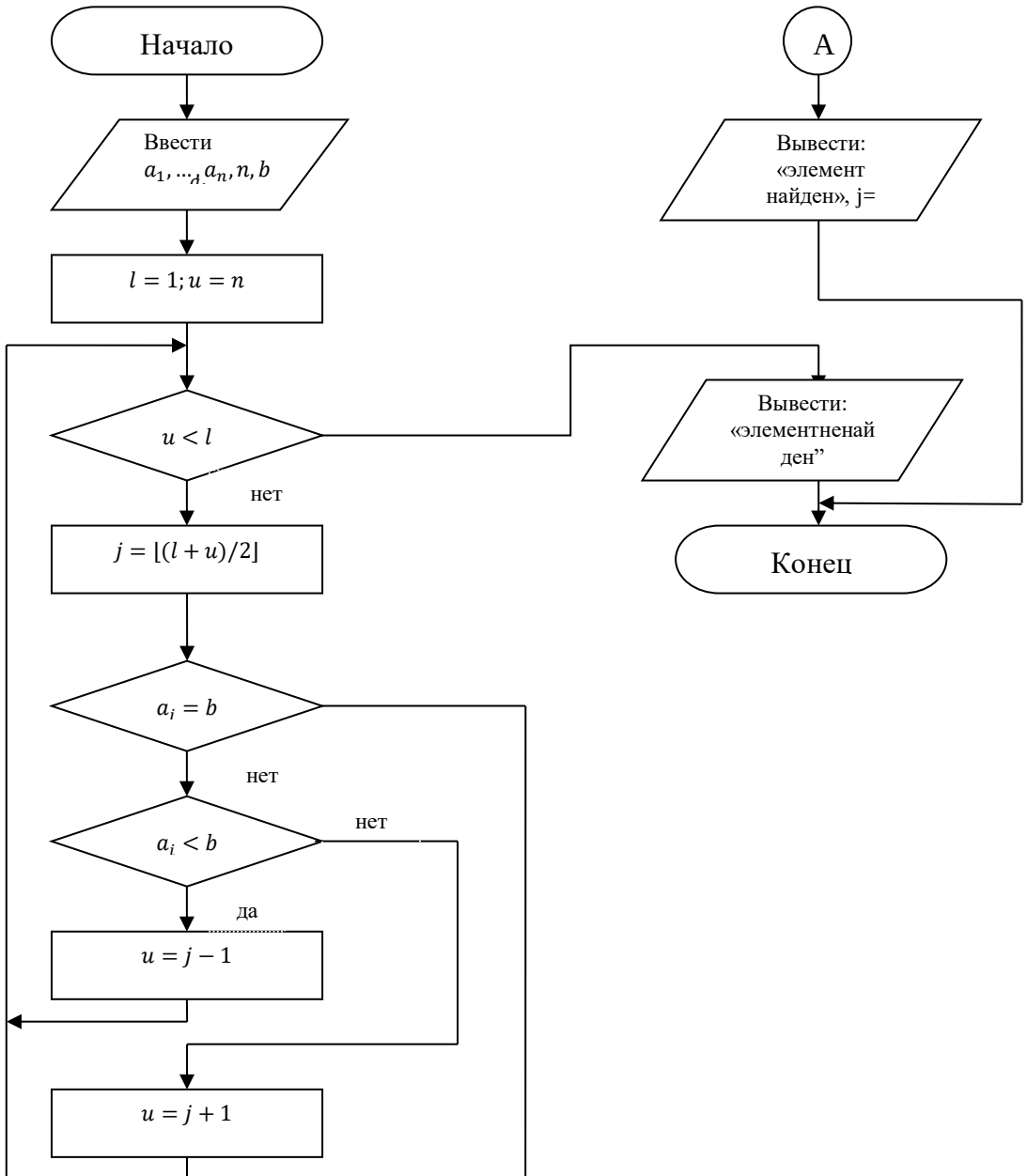
~~3 **5** 6 8 12 15 17 18 20 25~~;

3-й шаг. Рассматриваем два элемента, значение $m = \left[\frac{3+4}{2}\right] = 3$.

~~3 5~~**~~6~~** 8 ~~12 15 17 18 20 25~~;

A[3] = 6. Элемент найден, его номер – 3.

Блок - схема алгоритма бинарного поиска на рис.19:



Случайный поиск.

Организация поиска k-го элемента в неупорядоченном массиве Xвозможна следующим образом. Выбирается случайным образом элемент с номером q. Массив Xразбивается на три части: элементы, меньшие $X[q]$, равные $X[q]$и большие $X[q]$. А затем, в зависимости от количества элементов в каждой части, выбирается одна из частей для дальнейшего поиска. Теоретическая оценка числа сравнений имеет порядок $k*N$, т. е. для худшего случая N^2, но на практике он значительно быстрее.

СЛОЖНОСТЬ АЛГОРИТМОВ

Характеристики алгоритма, которые влияют на его применимость, принято называть характеристиками сложности алгоритма. Среди характеристик сложности наиболее важными являются две, характеризующие ресурсы исполнителя: время и память. Необходимо знать, как долго будет выполняться алгоритм и хватит ли ресурса памяти для этого. Время зависит от того, кто является исполнителем (человек, вычислительное устройство, компьютер), и от того, насколько быстро он делает операции (разные компьютеры обладают разной производительностью). Так как нужна объективная характеристика алгоритма, не зависящая от исполнителя, то вместо времени исполнения алгоритма будем рассматривать число шагов t выполнения алгоритма. Если $\bar{\tau}$ – среднее время одного шага исполнителя, то фактическое время работы алгоритма для этого исполнителя.

$$T_{\emptyset} = \bar{\tau} * t$$

Таким образом, t есть характеристика алгоритма, не зависящая от особенностей исполнителя, и потому математическая характеристика сложности алгоритма. Память S, используемая алгоритмом, также зависит от особенностей исполнителя. Если на каждом шаге алгоритма используется не более μ единиц памяти, то $S \leq \mu \cdot t$. Эта оценка очень грубая, так как t может значительно превосходить S. В большинстве случаев в качестве характеристики сложности алгоритма применяется характеристика t – число шагов выполнения алгоритма.

Трудоемкость алгоритмов.

Итак, t зависит от исходных данных задачи. Зависимость эту не всегда возможно анализировать непосредственно. Вследствие этого целесообразно будет определить временные рамки выполнения алгоритма (максимальное и минимальное время), сколько в среднем будет выполняться алгоритм (среднее время). Но для любых вариантов задачи время (число шагов) ничем не ограничено. Так, при сортировке массива время, как правило, зависит от длины массива и растет с ростом числа элементов массива. Принято входные данные Vалгоритма характеризовать одним параметром или несколькими параметрами. Одной из таких характеристик является объем входных данных – число элементов входных данных. Эта характеристика объективно характеризует входные данные так же, как и число шагов объективно характеризует исполнение алгоритма. В свою очередь, устанавливают зависимость объема входных данных от одного или нескольких параметров, характеризующих задачу. Так, в задаче сортировки массива таким параметром является длина n массива.

Так как число шагов алгоритма зависит не только от выбранных в задаче параметров$P = (n, m, \dots)$, характеризующих объем входных данных, но и от других характеристик входных данных$X = \left(X_{1,} X_{2,\dots}\right)$, то можно ввести оценку по всем этим характеристикам. Оценка наибольшего числа шагов, необходимых для выполнения алгоритма, в зависимости от параметров P: $t = (P, X) \leq \max_{X} T\,(P, X) \equiv T(P)$

называется максимальной трудоемкостью алгоритма или просто трудоемкостью алгоритма. Максимальная трудоемкость дает возможность оценить максимальное время, необходимое для исполнения алгоритма. Эта оценка может быть очень завышенной в некоторых случаях.

Поэтому важно иметь оценку наименьшего числа шагов, которую называют минимальной трудоемкостью:

$$t = (p, x) \geq \min_{x} t\,(P, X) \equiv Tmin(P)$$

и оценку среднего числа шагов, которую называют средней трудоемкостью:

$$t = t(P, X) \approx \frac{1}{K} \sum_{X} t(P, X) \equiv Tcp(P)$$

где k – число вариантов других характеристик входных данных.

Трудоемкость алгоритма позволяет оценить время выполнения алгоритма при решении той или иной задачи: $T_{\emptyset}\ (X) \leq \bar{\tau} * T(n)$

При этом коэффициент $\bar{\tau}$ статистически определяется для исполнителя или оценивается некоторой константой. Однако точный вид зависимости *T(n)* от аргумента *n* часто очень трудно установить. Поэтому вместо установления вида функции для трудоемкости оценивается быстрота роста этой функции при помощи некоторой простой функции f(n).

Говорят, что T(n) = O(f(n)),если |T(n)| ≤ C|f(n)| для всех значений n > n_0. Такая оценка роста функции T(n) является односторонней, так как функция f(n) может расти быстрее. Лучше оценивать рост трудоемкости функцией f(n), имеющей тот же порядок роста, т. е. также |T(n)| ≥ C1|f(n)|. В этом случае пишут
T(n) = Θ(f(n))и говорят, что рост *T(n)* оценивается ростом f(n). Наиболее простыми функциями, оценивающими рост трудоемкости, являются полиномы

$$p(n) = n^k + c_1 * n^{k-1} + c_2 * n^{k-2} * t \ldots + c_k$$

В случае T(n) = Θ(p(n)), учитывая, чтор(n) = Θ(n k), получаем T(n) = Θ(n k). Говорят, что в этом случае трудоемкость полиномиальна или алгоритм имеет полиномиальную трудоемкость. При k = 1 T(n) = Θ(n) и алгоритмы принято называть алгоритмами с линейной трудоемкостью.

Если есть два алгоритма A1 и A2 решения некоторой задачи и оба имеют полиномиальную трудоемкость, причем *k1 < k2*, то говорят, что первый алгоритм имеет меньшую трудоемкость. Но меньшая трудоемкость не означает, что время решения задачи первым алгоритмом будет меньше, чем вторым. Так, пусть

$$90n \leq T_1(n) \leq 100n\,, 9n^2 \leq T_2(n) \leq 10n^2$$

Тогда при n < 10 оказывается, что время решения задачи для первого алгоритма больше, чем для второго. Однако, из определения ясно, что найдется такое n_0 (в примере n_0 = 10), что время решения задачи при n > n0 будет всегда меньше для первого алгоритма.
Трудоемкость алгоритма может иметь скорость роста меньшую, чем линейная. Например, $T(n) = \theta\left(\sqrt{n}\right)$ или $T(n) = \log_2 n$.

Но и в этом случае принято говорить о полиномиальной трудоемкости. Алгоритмы, трудоемкость которых растет быстрее любого полинома, принято называть алгоритмами экспоненциальной трудоемкости, даже если скорость роста трудоемкости оценивается более медленной функцией, чем экспонента. Например, экспоненциальными являются все алгоритмы со следующими трудоемкостями:

$$T(n) = \theta(2^n), T(n) = \theta\left(2^{\sqrt{n}}\right) T(n) = \theta\left(n^{n^2}\right)$$

Причина, по которой используются только эти два названия трудоемкости (полиномиальная и экспоненциальная), состоит в том, что алгоритмы полиномиальной трудоемкости, как правило, эффективны, если показатель степени у полинома не слишком большой. А алгоритмы экспоненциальной трудоемкости не являются эффективными, так как время вычисления по этим алгоритмам растет очень быстро. В таблице показана скорость нарастания времени работы алгоритмов различной трудоемкости в секундах на компьютере с быстродействием 10^6 оп/сек.

n	10	20	30	40	50
n	0.00001	0.00002	0.00003	0.00004	0.00005
n^2	0.0001	0.0004	0.0009	0.00016	0.00025
n^3	0.001	0.008	0.0027	0.0064	0.125
n^5	0.1	3.2	24.3	1.7 мин	5.3 мин
2^n	0.001	1.0	17.9 мин	12,7 дн	35,7 лет
3^n	0.059	58 мин	6.5 лет	385500 лет	200×10^8лет

При нескольких параметрах входных данных трудоемкость полиномиального алгоритма растет как полином от нескольких аргументов. Например,

$$T(n,m)=\theta(n\times m^2), T(n,m,k)=\theta(n^2k\times \log_2 m), \sum_{i=1}^{n(A)} n_i\,(B)$$

Оценивание трудоемкости алгоритмов.

Процесс получения оценки трудоемкости называется оцениванием трудоемкости. Для этого следует анализировать алгоритм с точки зрения быстроты роста числа его шагов, при изменении параметров задачи (параметров входных данных). Прежде всего, в алгоритме следует выделить циклы. Если циклов нет, то число шагов линейной структуры алгоритма не зависит от параметров задачи и, следовательно, трудоемкость является константной, т. е. оценивается как *Θ*(1).

Циклическая структура алгоритма ведет к повторению выполнения его частей, что влияет на общее число шагов выполнения, т. е. на трудоемкость. Следует оценить для каждого цикла, от каких параметров задачи зависит число повторений цикла и как оно растет с ростом этих параметров.

Если цикл *B* с числом повторений *n(B)* вложен в цикл *A* с числом повторений *n(A)* и циклы независимы (число повторений цикла *B* не зависит от выполнения цикла *A*), то общее число повторений цикла *B* с учетом повторений цикла *A* составляет *n(A) · n(B).*

Отсюда правило: *для вложенных независимых циклов их трудоемкости перемножаются Θ(AB) = Θ(A) · Θ(B).*

Если вложенные циклы не являются независимыми, т. е. число повторений внутреннего цикла $n_i(B)$ зависит от номера *i* повторения при выполнении внешнего цикла, то нужно проанализировать, как зависит общее число повторений внутреннего цикла от параметров задачи.

Если циклы не являются вложенными, то трудоемкость определяется наибольшей из трудоемкостей циклов

$$\Theta(A + B) = \Theta(A) + \Theta(B) = max\{\Theta(A), \Theta(B)\}.$$

При оценке максимальной трудоемкости следует подбирать такие примеры входных данных для тех или иных параметров задачи, на которых реализуется максимальное число шагов алгоритма. При оценке минимальной трудоемкости следует подбирать примеры, на которых реализуется минимальное число шагов алгоритма. Ввиду сложности некоторых алгоритмов такие примеры не всегда удается построить, но в таких случаях для оценки трудоемкости бывает достаточно примеров и близких по числу операций к максимальному или соответственно к минимальному числу.[11]

Рассмотрим примеры оценивания трудоемкости на примере алгоритма сортировки массива методом «пузырька». Блок – схема алгоритма сортировки методом «пузырька» см. рис. 15

Алгоритм содержит вложенные циклы. Внешний цикл, в случае массива входных данных, упорядоченного по убыванию, будет выполняться максимальное число раз*: n − 1*, а в случае входного массива, упорядоченного по возрастанию, будет выполняться только 1 раз. Внутренний цикл во втором случае выполняется *n − 1* раз, а в первом случае циклы зависимы, но, внутренний цикл в среднем выполняется *n/2* раз. Поэтому максимальная трудоемкость (входные данные первого случая) оценивается как

$$\Theta(n) \cdot \Theta(n) = \Theta(n^2),$$

а минимальная трудоемкость (входные данные второго случая) – как

$\Theta(1) \cdot \Theta(n) = \Theta(n)$.

Во втором разделе рассмотрены методы сортировки элементов массива: метод простого выбора, метод «пузырька», сортировка слиянием и вставками. Разобран типовой пример нахождения максимального и минимального элементов в массив и принцип бинарного поиска в упорядоченном массиве. Для закрепления навыков создания алгоритмов сортировки можно рекомендовать задания для самостоятельной работы.

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ №12. МЕТОД ПЕРЕБОРА В ЗАДАЧАХ ПОИСКА

Вариант 1.Дана вещественная матрица A(N,M). Составить программу замены всех положительных элементов матрицы на элемент, имеющий минимальное значение.

Вариант 2.Дана вещественная матрица A(N,M). Составить программу нахождения максимального отрицательного элемента матрицы и нахождения его местоположения.

Вариант 3.Дана вещественная матрица A(N,M). Составить программу замены всех отрицательных элементов матрицы на элемент, имеющий максимальное значение.

Вариант 4.Составить программу замены всех отрицательных элементов матрицы A(N,N) на элемент этой матрицы, имеющий минимальное значение. Скорректированную матрицу напечатать.

Вариант 5.Дана вещественная матрица A(N,M).Составить программу нахождения минимального положительного элемента матрицы и нахождения его местоположения.

Вариант 6.Составить программу замены всех отрицательных элементов матрицы A(6,6) на 0, если сумма минимального и максимального элементов этой матрицы окажется меньше P.

Вариант 7. В массиве хранятся значения роста двенадцати человек. С помощью датчика случайных чисел заполнить массив целыми значениями, лежащими в диапазоне от 163 до 190 включительно. Отсортировать массив по возрастанию методом перебора.

Вариант 8. Заполнить массив A[1..18] числами из диапазона 20 150 . Отсортировать массив по убыванию методом пузырька.

Вариант 9. В массиве хранится информация о количестве осадков, выпавших за каждый день ноября. Отсортировать данные по возрастанию методом быстрой сортировки.

Вариант 10. В массиве хранятся сведения об оценках 25 учеников по химии. Отсортировать массив по убыванию методом пузырька.

ЛАБОРАТОРНОЕ ЗАДАНИЕ № 13, 14.

Графы возникли в восемнадцатом столетии, когда известный математик, Леонард Эйлер, пытался решить теперь уже классическую задачу о Кенигсбергских мостах. В то время в городе Кенигсбергебыло два острова, соединенных семью мостами с берегами реки Преголь и друг с другом так, как показано на рис. 7.1. Задача состоит в следующем: осуществить прогулку по городу таким образом, чтобы, пройдя ровно по одному разу по каждому мосту, вернуться в то же место, откуда начиналась прогулка. Решая эту задачу, Эйлер изобразил Кенигсберг в виде графа, отождествив его вершины с частями города, а ребра - с мостами, которыми связаны эти части. Как мы покажем в § 7.1, Эйлеру удалось доказать, что искомого маршрута обхода города не существует.

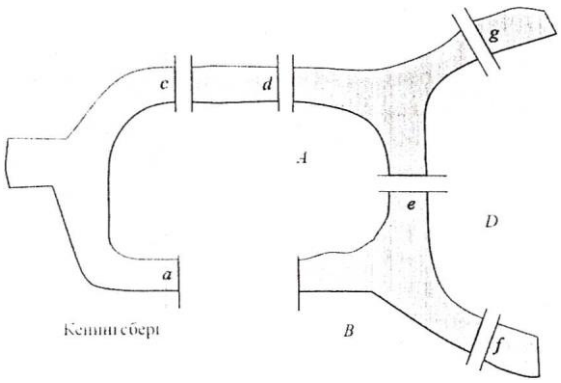


Рисунок 7.1. Схема старого Кенигсберга

В этой главе мы вводим стандартную терминологию, используемую в теории графов, и разбираем несколько конкретных задач решаемых с помощью графов. В частности, мы познакомимся с классом графов, называемых деревьями. Деревья – естественная модель, представляющая данные, организованные в иерархичную систему. Поиск по дереву для выделения отдельных предметов и сортировка данных в дереве представляет собой важные точки приложения усилий в информатике. В приложении к этой главе мы займемся сортировкой и поиском данных, организованных в деревья.

Графы и терминология

На рис. 7.1 изображены семь мостов Кенигсберга так.как они были расположены в восемнадцатом столетии. В задаче, к которой обратился Эйлер, спрашивается: можно ли найти маршрут прогулки, который проходит ровно один раз по каждому из мостов и начинается и заканчивается в одном и том же месте города?

Модель задачи - это *граф,* состоящий из множества *вершин* и множества *ребер,* соединяющих вершины. Вершины A, B,C и *D*символизируют берега реки и острова, а ребра *a,в, c,d,f*и *g*обозначают семь мостов (см. рис. 7.2). Искомый маршрут (если он существует) соответствует обходу ребер графа таким образом, что каждое из них проходится только один раз. Проход ребра, очевидно, соответствует пересечению реки по мосту.

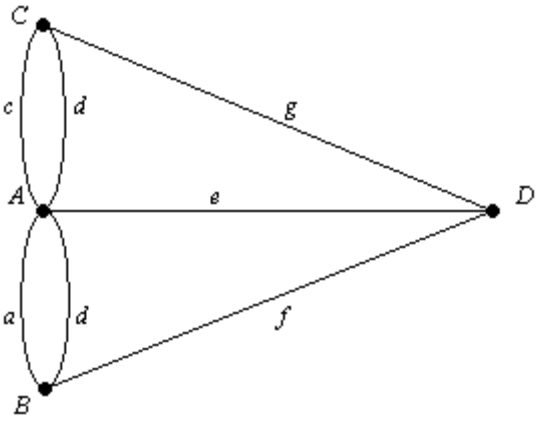


Рисунок 7.2. Модель задачи о мостах Кенигсберга

Граф, в котором найдется маршрут, найдется маршрут, начинающийся и заканчивающийся в одной вершине, и проходящий по всем ребрам графа ровно один раз,

называется *эйлеровым графом.* Последовательность вершин (может быть и с повторениями), через которые проходит искомый маршрут, как и сам маршрут, называется*эйлеровымциклом*. Эйлер заметил, что если в графе есть эйлеров цикл, то для каждого ребра, ведущего в какую-то вершину, должно найтись другое ребро, выходящее из этой вершины[1], и получил из этого простого наблюдсния такой вывод: если в данном графе существует эйлеров цикл, то к каждой вершине должно подходить четное число ребер.

Кроме того, Эйлеру удалось доказать и противоположное утверждение, так что граф, в котором любая пара вершин связана некоторой последовательностью ребер, является Эйлеровым тогда и только тогда, когда все его вершины имеют четную степень. *Степенью* вершины vназывается число δ(v)ребер, ей *инцидентных*[2] .

Теперь совершенно очевидно, что в графе, моделирующем задачу о мостах Кенигсберга, эйлерова цикла найти нельзя. Действительно, степени всех его вершин нечетны: *δ(B)* = *δ(C)* = δ(D) = 3 и *δ(A)* = 5. С легкой руки Эйлера графы, подобные тому, который мы исследовали при решении задачи о мостах, стали использовать при решении многих практических задач, а их изучение выросло в значительную область математики.

Простой граф определяется как пара G = (V, *E),* где V - конечное множество вершин, а *E* - конечное множество ребер, причем не может содержать *петель* (ребер, начинающихся и заканчивающихся в одной вершине) и *кратных ребер* (кратными называются несколько ребер, соединяющих одну и ту же пару вершин). Граф, изображенный на рис. 7.2. не является простым, поскольку, например, вершины*A*и *B* соединяются двумя ребрами (как раз эти ребра и называются кратными).

Две вершины *u*и *v* в простом графе называются *смежными,* если они соединяются каким-то ребром *e*, про которое говорят, что оно *инцидентно* вершине u(и v). Таким образом, мы можем представлять себе множество*E*ребер как множество пар смежных вершин, определяя тем самым нерефлексивное, симметричное отношение на множестве *V.* Отсутствие рефлексивности связано с тем, что в простом графе нет петель, т. е. ребер, оба конца которых находятся в одной вершине. Симметричность же отношения вытекает из того факта, что ребро *e*, соединяющее вершину *u* с *v,* соединяет и *v* с *u* (иначе говоря, ребра не ориентированы, т. е. не имеют направления). Единственное ребро простого графа, соединяющее пару вершин *u*и *v,* мы будем обозначать как *uv*(или *vu).*

Логическая матрица отношения на множестве вершин графа, которое задается его ребрами, называется*,матрицей смежности.* Симметричность отношения в терминах матрицы смежности M означает, что Мсимметрична относительно главной диагонали. А из-за нерефлексивности этого отношения на главной диагонали матрицы Мстоит символ «Л».

Пример 7.1. Нарисуйте граф G(V, E)с множеством вершин V= {a, b, c, d, e} и множеством ребер E= {ab, ae, bc, bd, ce, de}. Выпишите его матрицу смежности.

Решение. Граф G показан на рис. 7.3.

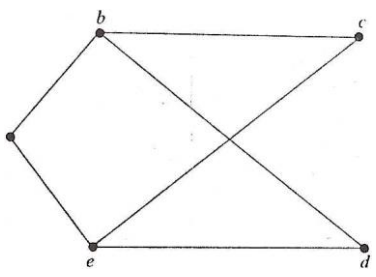


Рисунок 7.3.

Его матрица смежности имеет вид:

	a	b	c	d	e
a	Л	И	Л	Л	И
b	И	Л	И	И	Л
c	Л	И	Л	Л	И
d	Л	И	Л	Л	И
e	И	Л	И	И	Л

Для восстановления графа нам достаточно только тех элементов матрицы смежности, которые стоят над главной диагональю.

Подграфом графа G = (V, E) называется граф G’ = (V’, E’), в котором E’ CE и V’ CV.

Пример 7.2Найдите среди графиков H, Ки L, изображенных на рис. 7.4, подграфы графа G.

Решение. Обозначим вершины графов G, H и K как показано на рис. 7.5. Графы Ни K – подграфы в G, как видно из наших обозначений. Граф Lне является подграфом в G, поскольку у него есть вершина индекса 4, а у графа G такой нет.

Маршрутом длины *k*в графе Gназывается такая последовательность вершин v_0 , v_1 , …, v_k ,что для каждого i =1, …, kпара $v_{i-1}v_i$образует ребро графа. Мы будем обозначать такой маршрут через v_0 v_1 … v_k .Например 1 4 3 2 5 – это маршрут длины 4 в графе G из примера 7.2.

GH

KL

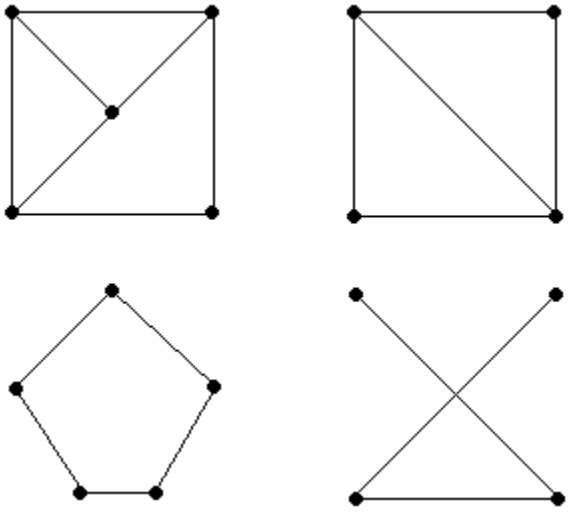


Рисунок 7.4.

Циклом в графе принято называть последовательность вершин v_0 , v_1 , … , v_k ,каждая пара которых является концами одного ребра, причем $v_0 = v_1$, а остальные вершины (и ребра) не повторяются. Иными словами, цикл – это замкнутый маршрут, проходящий через каждую

свою вершину и ребро только один раз

GHK

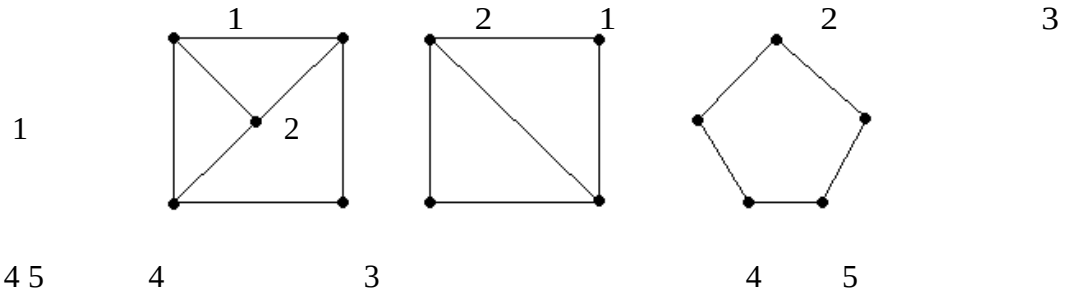


Рисунок 7.5

Пример 7.3. Найдите циклы в графе G из примера 7.2.

Решение. В этом графе есть два разных цикла длины 5:

1 3 2 5 4 1 и 1 2 5 4 3 1

Мы можем пройти эти циклы как в одном направлении так и в другом, начиная с произвольной вершины цикла. Кроме того, в графе есть три разных цикла длины 4:

1 2 5 4 1, 1 2 3 4 1 и 2 5 4 3 2,

и два цикла длины 3:

1 2 3 1 и 1 3 4 1.

Граф, в котором нет циклов, называется *ацикличным.* Структуры деревьев, которые возникают в вычислениях , представляют собой частный случай ацикличных графов. Позже в этой главе мы ими займемся.

Граф, называют *связным,* если любую пару его вершин соединяет какой – нибудь маршрут. Любой общий граф можно разбить на подграфы, каждый из которых окажется связным. Минимальное число таких связных компонент называется *числом связности* графа и обозначается через *c(G)*. Вопросы связности имеют важное значение в приложениях теории графов к компьютерным сетям. Следующий алгоритм применяется для определения числа связности графа.

Алгоритм связности.

Пусть G = (V, E) – граф. Алгоритм предназначен для вычисления значения *c = c(G),* т.е. числа компонент связности данного графа G.

begin
V' :=V;
c:=0;
while*V'≠ ø* **do**
begin
Выбратьy Є *V'*
Найти вершины, соединяющие маршрутом с y;
Удалить вершину y из V' и

соответствующие ребра из E;

c:=c+1;

end

end

Пример 7.4. Проследите за работой алгоритма связности на графе, изображенном на рис. 7.6.

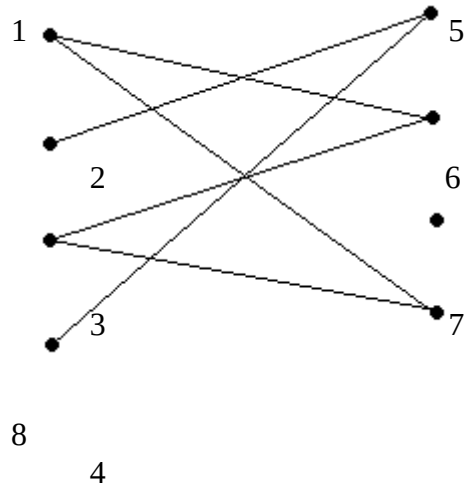


Рисунок 7.6.

Решение. Смотри табл. 7.1.

Таблица 7.1.

	V’
Исходные значения	{1,2,3,4,5,6,7,8}
Выбор y = 1	{2,4,5,7}
Выбор y = 2	{7}
Выбор y = 7	ø

Итак,*c(G)* = 3. Соответствующие компоненты связности приведены на рис. 7.7.

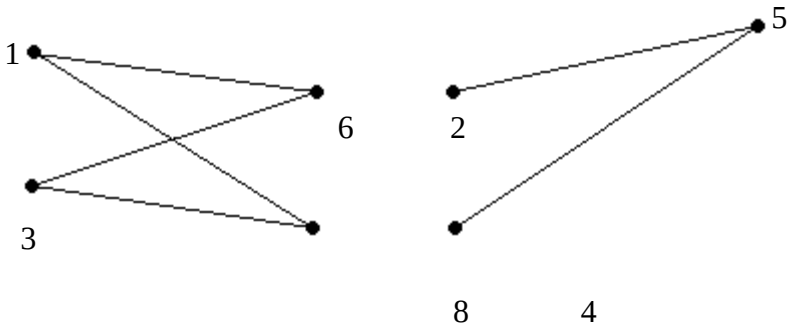


Рисунок 7.7.

Гамильтоновы графы

Мы начали эту главу с изучения эйлеровых графов, обладающих замкнутым маршрутом, который проходит по всем ребрам графа ровно один раз. Похожая задача состоит в поиске цикла, проходящего через каждую вершину графа в точности один раз. Такой цикл, если он существует, называется *гамильтоновым*, а соответствующий граф – *гамильтоновым графом.*

Гамильтоновы графы служат моделью при составлении расписания движения поездов, для телекоммуникационных сетей и т.д. В отличие от задачи Эйлера, простого критерия гомильтоновости графа пока не известно. Поиск хорошего критерия остается одной из главных нерешенных задач теории графов.

Полный граф K_5 изображен на рис. 7.8. Егоцикл*abcdea*, очевидно, является гамильтоновым. В нем есть и другие гамильтоновы циклы. Поскольку каждая вершина смежна с остальными, то начиная с вершины *a*, в качестве второй вершины цикла мы можем выбрать любую из четырех оставшихся. Далее у нас будет три варианта для выбора третьей вершины и два для четвертой, после чего мы вернемся в вершину *a.* Таким образом, у нас есть $4 \cdot 3 \cdot 2 = 24$ цикла. Поскольку каждый цикл можно проходить как в одном направлении, так и в другом, то реально в графе K_5 есть только 12 разных гамильтоновых циклов[1].

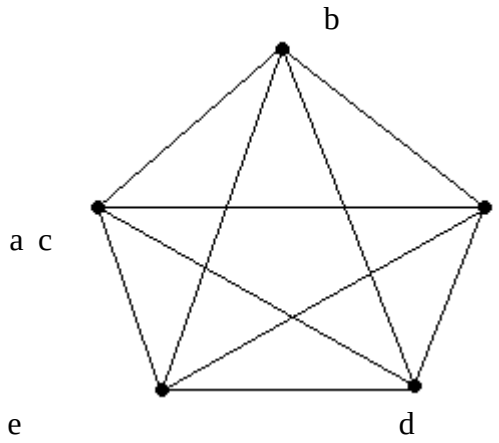


Рисунок 7.8. Полный граф K_5

Поиск гамильтонова цикла (если он существует) в произвольном (связном) графе – задача далеко не всегда простая. Ответ на вопрос о гамильтоновости графа может оказаться довольно трудоемким.

Пример 7.5. Покажите что граф, изображенный на рис. 7.9, не является гамильтоновым.

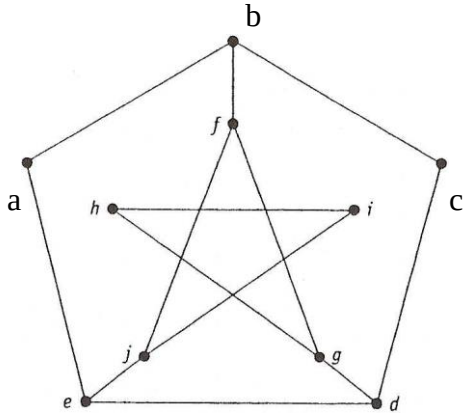


Рисунок 7.9. Пример не гамильтонова графа

Решение. Предположим, что в связном графе найдется гамильтонов цикл. Каждая вершина *v*включается в гамильтонов цикл*C* выбором двух инцидентных с ней ребер, а значит, степень каждой вершины в гамильтоновом цикле (после удаления лишних ребер) равна 2. Степени вершин данного графа - 2 или 3. вершины степени 2 входят в цикл вместе с обоими инцидентными с ними ребрами. Следовательно, ребра *ab, ae, cd, cb, hi, hgиij* в том или ином порядке входят в гамильтонов цикл*C (*см. рис. 7.10.).

Ребро *bf*не может быть часть цикла*C*, поскольку каждая вершина такого цикла должна иметь степень 2. Значит, ребра *fi*и *fg* обязаны входить в цикл*C*, чтобы включить в него вершину *f*.Но тогда ребра *jeigd*никак не могут принадлежать циклу*C*, поскольку в противном случае в нем появятся вершины степени три. Это вынуждает нас включить в цикл ребро *ed*, что приводит нас к противоречию: ребра, которые мы были вынуждены выбрать, образуют два несвязных цикла, а не один, существование которого мы предполагали. Вывод: граф, изображенный на рисунке 7.10, не является гамильтоновым.

Гамильтоновы графы применяются для моделирования многих практических задач. Основой всех таких задач служит классическая *задача коммивояжера.*

Коммивояжер должен совершить поездку по городам и вернуться обратно, побыв в каждом городе ровно один раз, сведя при этом затраты на передвижение к минимуму.

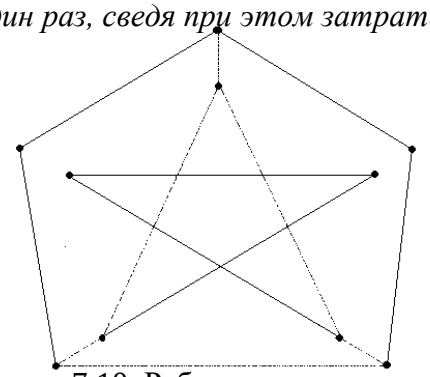


Рисунок 7.10. Ребра входящие в гамильтонов цикл*C*

Графическая модель задачи коммивояжера состоит из гамильтонова графа, вершины которого изображают города, а ребра связывающие их дороги. Кроме того, каждое ребро оснащено весом, обозначающим транспортные затраты, необходимые для путешествия по соответствующей дороге, такие, как, например, расстояние между городами или время движения по дороге[2]. Для решения задачи нам необходимо найти гамильтонов цикл минимального общего веса.

К сожалению, эффективный алгоритм решения данной задачи пока не известен. Для сложных сетей число гамильтоновых циклов, которые необходимо просмотреть для выделения минимального, непомерно огромно. Однако существуют алгоритмы поиска *субоптимального решения.* Субоптимальное решение необязательно даст цикл минимального общего веса, но найденный цикл будет, как правило, значительно меньшего веса, чем большинство произвольных гамильтоновых циклов! Один из таких алгоритмов мы сейчас и изучим.

Алгоритм ближайшего соседа.

Этот алгоритм выдает субоптимальное решение задачи коммивояжера, генерируя гамильтоновы циклы в нагруженном графе с множеством вершин V. Цикл, полученный в результате работы алгоритма, будет совпадать с конечным значением переменной *маршрут,* а его общая длина – конечное значение переменной *w.*

begin
Выбрать v Є V;
маршрут:= v;
w:=0;

v':=v;
Отметитьv':
while*остаются неотмеченными вершины* **do**
begin
Выбрать неотмеченную вершину u,
ближайшую к v';
маршрут:=маршрут u;
w:=w+вес ребра v'u;
v':=u;
Отметить v';
end
маршрут:= маршрут v;
w:=w+вес ребра v'u;
end

Пример 7.6. Примените алгоритм ближайшего соседа к графу, изображенному на рисунке 7.11. За исходную вершину возьмите вершину D.

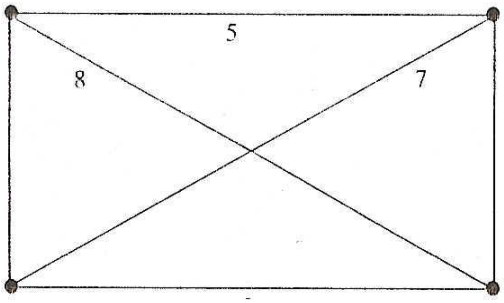


Рисунок 7.11.

Решение. Смотрите табл.7.2

Таблица 7.2

	u	*маршрут*	*w*	*v'*
Исходные значения	-	D	0	D
	C	DC	3	C
	A	DCA	9	A
	B	DCAB	14	B
Последний проход	B	DCABD	24	B

В результате работы алгоритма был найден гамильтонов цикл DCABDобщего веса 24. Делая полный перебор всех циклов в этом маленьком графе, можно обнаружить еще два других гамильтоновых цикла: ABCDA общего веса 23 и ACBDAобщего веса 31. В полном графе с двадцатью вершинами существует приблизительно $6{,}1 \cdot 10^{16}$ гамильтоновых циклов, перечисление которых требует чрезвычайно много машинной памяти и времени.

Деревья

Как уже упоминалось, есть класс графов, называемых деревьями, которые особенно интенсивно используются в вычислительных приложениях. Граф G = (V, E) называется *деревом*, если он связен и ацикличен(т.е. не содержит циклов).

Пусть G = (V, E) – граф с *n*вершинами и *m*ребрами. Можно сформулировать несколько необходимых и достаточных условий, при которых G является деревом:

- Любая пара вершин в G соединена единственным путем.

- G связен и $m = n - 1$.
- Gсвязен, а удаление хотя бы одного его ребра нарушает связность графа.
- Gацикличен, но если добавить хотя бы одно ребро, то в G появится цикл.

Эквивалентность большинства из этих условий устанавливается без особого труда. Наиболее сложно разобраться со вторым из них. В следующем примере мы докажем, что дерево с nвершинами имеет ровно$n - 1$ ребро.

Пример 7.7. Докажите с помощью индукции по числу вершин, что для дерева T с nвершинами и mребрами выполнено соотношение: $m = n - 1$.

Решение. Поскольку дерево с единственной вершиной вообще не содержит ребер, то доказываемое утверждение справедливо при $n = 1$.

Рассмотрим деревоT с nвершинами (и m ребрами), где $n > 1$и предположим, что любое дерево с $k < n$вершинами имеет ровно $k - 1$ ребро.

Удалим ребро из T. По третьему свойству дерево T после этой процедуры превратится в несвязный граф. Получится ровно две компоненты связности, ни одна из которых не имеет циклов (в противном случае исходный графT тоже содержал бы циклы и не мог бы быть деревом). Таким образом, полученные компоненты связности – тоже деревья.

Обозначим новые деревья T_1 и T_2. Пусть n_1– количество вершин у дерева T_1, а n_2– уT_2. Поскольку $n_1 + n_2 = n$, то$n_1 < n$и$n_2 < n$.

По предположению индукции дерево T_1имеет $n_1 - 1$ ребро, а $T_2 - n_2 - 1$. Следовательно, исходное дерево T насчитывало (с учетом одного удаленного)
$(n_1 - 1) + (n_2 - 1) + 1 = n - 1$ ребро, что и требовалось доказать.

Несложно доказать, что в любом связном графе найдется подграф, являющийся деревом. Подграф в G, являющийся деревом и включающий в себя все вершины G, называется *остовным деревом*. Остовное дерево в графе Gстроится просто: выбираем произвольное его ребро и последовательно добавляем другие ребра, не создавая при этом циклов, до тех пор, пока нельзя будет добавить никакого ребра, не получив при этом цикла. Благодаря примеру 7.7, мы знаем, что для построения остовного дерева в графе из nвершин необходимо выбрать ровно $n - 1$ ребро.

Пример 7.8. Найдите два разных остовных дерева в граве, изображенном на рис. 7.12.

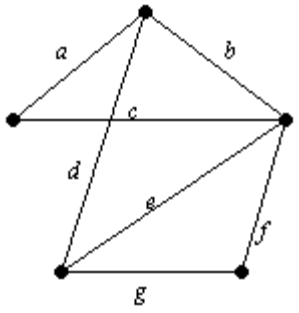


Рисунок 7.12. Связный граф G

Решение. В этом графе существует несколько остовных деревьев. Одно из них получается последовательным выбором ребер: *a, b, d,* и *f.*Другое – *b, c, e*и*g.* Названные деревья показаны на рисунке 7.13.

Процесс, описанный в примере 7.8, можно приспособить для решения *задачи кратчайшего соединения:*

Нужно построить железную сеть, связывающую некоторое число городов. Известна стоимость строительства отрезка путей между любой парой городов. Требуется найти сеть минимальной стоимости.

На языке теории графов нам нужно в нагруженном графе найти остовное дерево наименьшего общего веса. Такое дерево принято называть *минимальным остовным деревом* или, сокращенно *МОД*. В отличие от задачи коммивояжера, здесь есть эффективный алгоритм, находящий действительно минимальное остовное дерево. Он похож на алгоритм Прима.

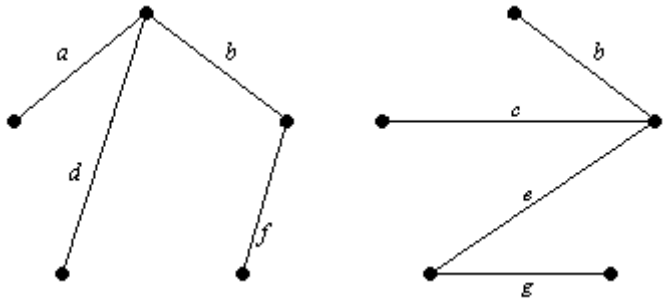


Рисунок 7.13. Остовные деревья графа *G*

Алгоритм поиска минимального остовного дерева. Пусть *G = (V, E)* – связной взвешенный граф. Алгоритм строит МОД в графе G, последовательно выбирая ребра наименьшего возможного веса до образования остовного дерева. МОД в памяти компьютера хранится в виде множества T ребер.

begin
e:=ребро графа G с наименьшим весом;
T:={e};
E':=E\{e}
while*E' ≠ø*
begin
e':= ребро из E' наименьшего веса;
*T:= T*U*{e'};*
E' := множество ребер из E' \{T},
чьё добавление к T не ведет
к образованию циклов;
end
end

Пример 7.9. В таблице 7.3 дано расстояние (в милях) между пятью деревнями A, B, C, Dи E. Найдите минимальное остовное дерево.

Таблица 7.3

	A	B	C	D	E
A	-	13	3	9	9
B	13	-	11	11	13
C	3	11	-	9	7
D	9	11	9	-	2
E	9	13	7	2	-

Решение. Ребра выбираются следующим образом: первое – 1 DEвеса 2; второе – ACвеса 3; третье – CE веса 7. на этой ступени строящееся дерево выглядит так, как на рис. 7.14

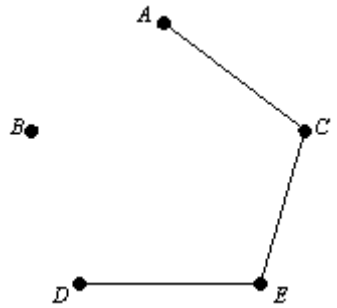


Рисунок 7.14. Вид дерева после трех шагов

Следующие повесу ребра – AD, AEи CD, каждое из которых имеет вес 9. Однако какое бы из них мы не добавили, получится цикл. Поэтому перечисленные ребра следует исключить из доступных для строительства. Далее идут ребра BC и BDвеса 11. Можно присоединить любое из них, получив при этом два разных МОД: {AC, BC, CE, DE} или {AC, BD, CE, DE} веса 23 каждое.

Зачастую, нам хотелось бы иметь деревья, представляющие собой формацию с учетом естественной иерархической структуры, такие как, например, генеалогическое дерево (рис.7.15). На нем показаны некоторые члены семьи Бернулли, каждый из которых был известным швейцарским математиком.

Генеалогическое дерево можно изобразить и более сжато. Схема приведенная на рисунке 7.16, представляет собой пример так называемого дерева с корнем. *Деревом с корнем* называется дерево с одной выделенной вершиной. Именно эта выделенная вершина и является *корнем* дерева. Вершины дерева, лежащие непосредственно под корнем, называются *сыновьями*. С другой стороны, вершина, стоящая непосредственно перед сыном, называется ее *отцом*.

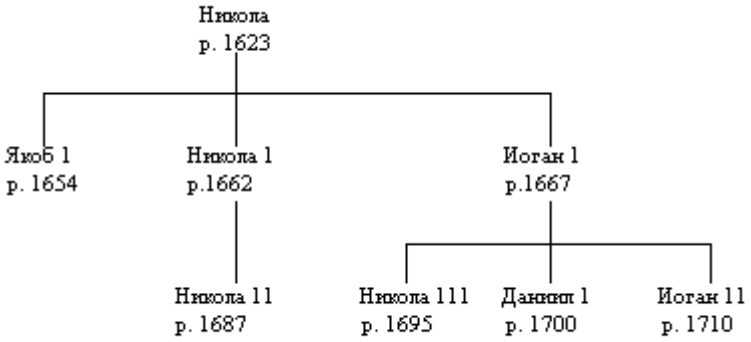


Рисунок 7.15. Династия Бернулли

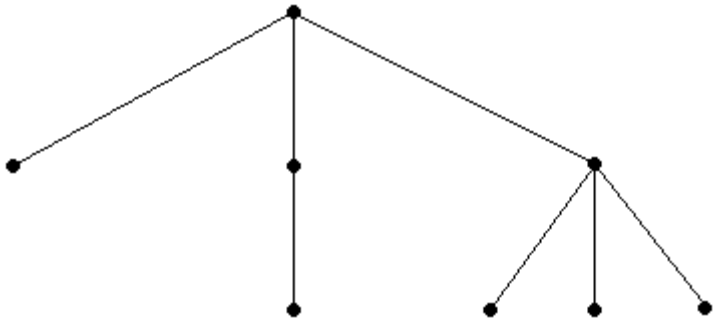
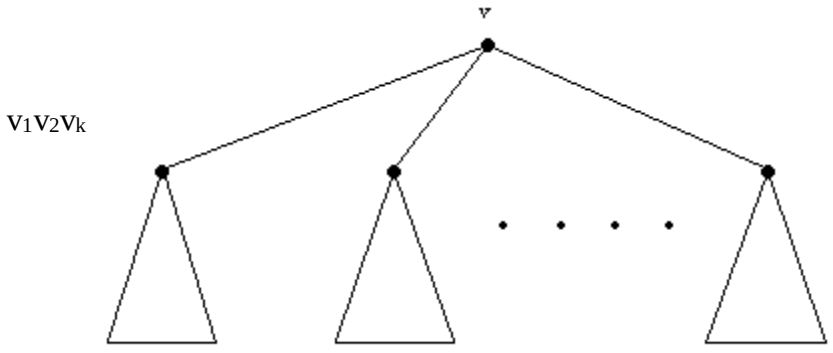


Рисунок 7.16. Схема генеалогического древа Бернулли

Древо с корнем можно определить рекуррентным образом. Отдельная вершина является деревом с корнем (она же служит и корнем такого дерева). Если T_1, T_2, …, T_k - несвязные друг с другом деревья с корнями v_1, v_2, …, v_k, то граф, получающийся присоединением новой вершины vк каждой из вершин v_1, v_2, …, v_k отдельным ребром, является деревом T с корнем v. Вершины v_1, …, v_k графа T – это сыновья корня v. Мы изображаем такое дерево с корнем, расположенным наверху, и сыновьями, стоящими ниже, непосредственно под корнем (см. рис. 1.17). Каждую вершину дерева с корнем можно рассматривать как корень другого дерева, которое «растет» из него. Мы будем называть его *поддеревом* дерева T.

Как мы уже говорили, вершина на самом верху дерева – его корень, а вот те, которые находятся в самом низу дерева (не имеют сыновей) принято называть *листьями.* Остальные вершины, отличные от корня и листьев, называют *внутренними*.

Область применения деревьев с корнями обширна. Это, например, и информатика, и биология, и менеджмент. Для приложения к информатике наиболее важны так называемые *двоичные* или *бинарные* деревья с корнем. Двоичное дерево отличает от остальных то, что каждая его вершина имеет не более двух сыновей. В двоичном дереве с корнем вниз от каждой вершины идет не более двух ребер.



T_1 T_2T_k

Рисунок 7.17.

Таким образом, можно сказать, что каждой вершине двоичного дерева с корнем соответствует не более, чем два поддерева, которые принято называть *левыми* и *правыми* поддеревьями этой вершины. Если оказалось, что у какой то вершины дерева отсутствует потомок слева, то ее левое поддерево называют *нулевым деревом* (т.е. нулевое дерево – это дерево без единой вершины). Аналогично, если у вершины отсутствует правый потомок, то ее правое поддерево будет нулевым.

Пример 1.10. Пусть T – двоичное дерево с корнем, изображенное на рисунке 7.18.

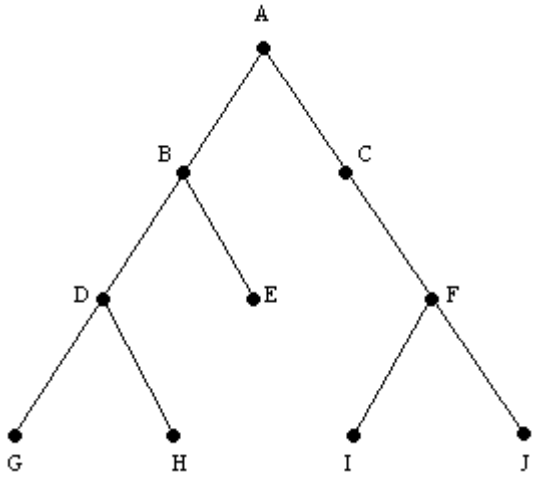


Рисунок 7.18. Двоичное дерево с корнем T

Определите

(а) корень T;
(б) корень левого поддерева вершины B;
(в) листья T;
(г) сыновей вершины C.

Нарисуйте двоичное дерево с корнем T', полученное из T перестановкой левых и правых поддеревьев к каждой вершины.

Решение. (а) A; (б) D; (в) G, H, E, I и J; (г)F.
Двоичное дерево с корнем T' начерчено на рис. 7.19.

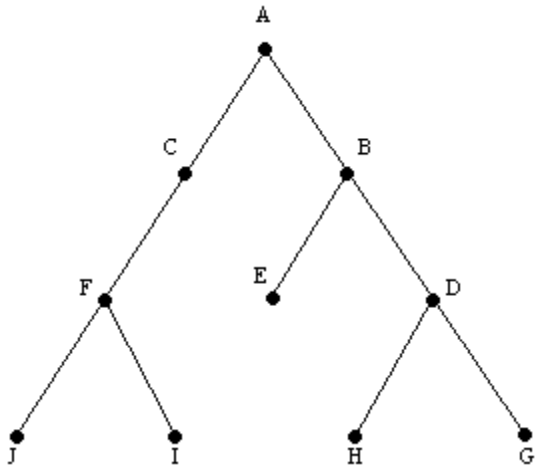


Рисунок 7.19. Двоичное дерево с корнем T'

КОНТРОЛЬНЫЕ ЗАДАНИЯ К ЛАБОРАТОРНОЙ РАБОТЕ № 13, 14.

Вариант 1
Нарисуйте граф, чья матрица смежности имеет вид:

$$\begin{bmatrix} Л & И & Л & И & Л & И \\ И & Л & И & Л & И & Л \\ Л & И & Л & И & Л & И \\ И & Л & И & Л & И & Л \\ Л & И & Л & И & Л & Л \\ И & Л & И & Л & Л & Л \end{bmatrix}$$

Опишите матрицу смежности полного графа K_n.

Вариант 2

Введя подходящие обозначения вершин, для каждого из графов на рисунке 7.20 подберите соответствующую матрицу смежности из перечисленных ниже.

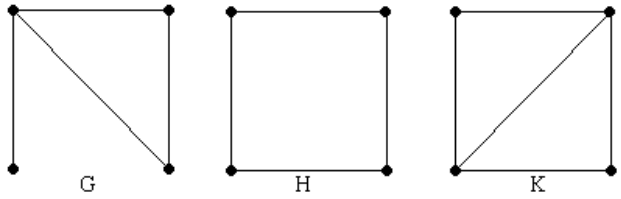


Рисунок 7.20.

$$\begin{bmatrix} Л & И & И & Л \\ И & Л & Л & И \\ И & Л & Л & И \\ Л & И & И & Л \end{bmatrix} \quad \begin{bmatrix} Л & И & И & Л \\ И & Л & И & И \\ И & И & Л & И \\ Л & И & И & Л \end{bmatrix} \quad \begin{bmatrix} Л & И & Л & Л \\ И & Л & И & И \\ Л & И & Л & И \\ Л & И & И & Л \end{bmatrix}$$

(а) (б) (с)

Вариант 3

Какие из графов на рис.7.12 могут являться подграфами графа из упражнения 7.3

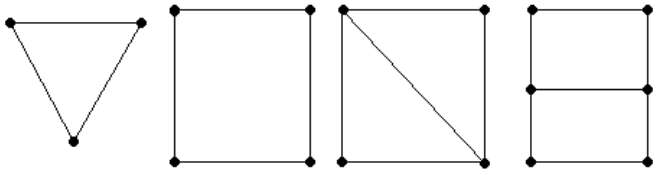


Рисунок 7.21. Кандидаты в подграфы

Вариант 4

Найдите гамильтоновы циклы в графе на рис.7.22.

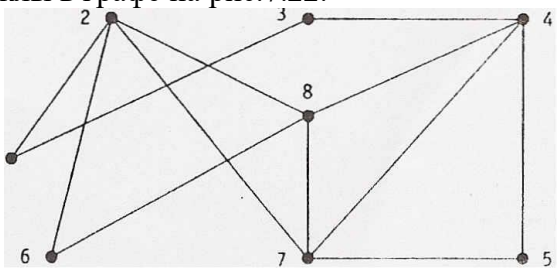
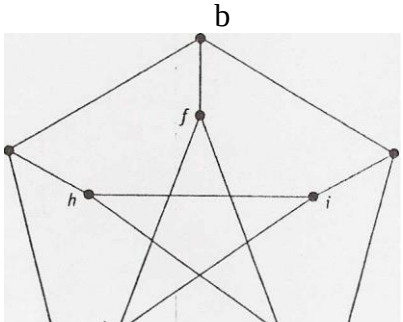


Рисунок 7.22.

Найдите в нем циклы длины 3, 4, 5, 6 и 7.

Вариант 5

На рисунке 7.23 изображен граф Петерсена P.



ac

Рисунок 7.23. Граф Петерсена

Найдите в нем цикл длины 9. Покажите, что P не является гамильтоновым.

Вариант 6

Используйте алгоритм ближайшего соседа для поиска гамильтонова цикла в нагруженном графе (рис.7.24), взяв за исходную

(а) вершину A;

(б) вершину D.

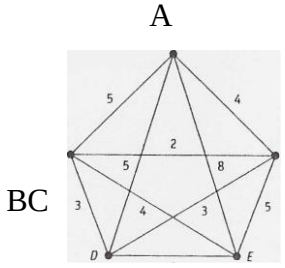


Рисунок 7.24. Нагруженный граф

Вариант 7

Выясните, являются ли графы, задаваемые следующими матрицами смежности, деревьями:

(а)
$$\begin{bmatrix} Л & Л & Л & Л & Л & И \\ Л & Л & Л & И & Л & И \\ Л & Л & Л & Л & И & И \\ Л & И & Л & Л & И & Л \\ Л & Л & И & И & Л & Л \\ И & И & И & Л & Л & Л \end{bmatrix};$$

(б)
$$\begin{bmatrix} Л & Л & Л & Л & Л & И \\ Л & Л & И & Л & Л & Л \\ Л & И & Л & И & Л & И \\ Л & Л & И & Л & И & Л \\ Л & Л & Л & И & Л & Л \\ И & Л & И & Л & Л & Л \end{bmatrix}.$$

Вариант 8

Известно, что дерево T имеет три вершины степени 3 и четыре вершины степени 2. Остальные вершины дерева имеют степень 1. Сколько вершин степени 1 есть у дерева T? (Указание: обозначьте число вершин дерева T через n и примените лемму об эстафете.)

Вариант 9

Лесом называют граф (не обязательно связный), каждая компонента связности которого – дерево. Пусть G – лес с n вершинами и k компонентами связности.

(а) Докажите, что G имеет n – kребер.

(б) Покажите, что если в каждой компоненте связности леса Gесть более одной вершины,

то Gсодержит по крайней мере 2kвершин степени 1.

(в) Нарисуйте лес с девятью вершинами и шестью ребрами, в котором не больше пяти вершин степени 1.

Вариант 10

Найдите мини0мальное остовное дерево графа, изображенного на рис.7.25.

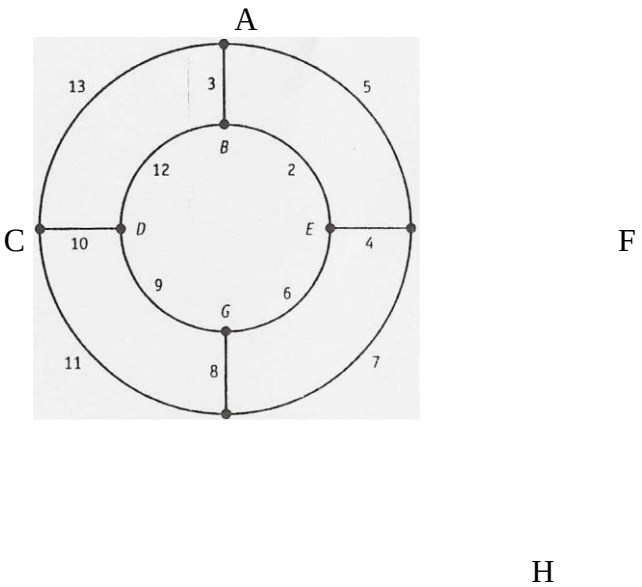


Рисунок 7.25.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

1. Игошин В.И.Теория алгоритмов: Учебное пособие. – М.:ИНФРА-М, 2013.- 318 с.
2. Канцедал С.А. Алгоритмизация и программирование: учебное пособие. – М.: ИД «ФОРУМ»: ИНФРА-М, 2014. – 352 с.
3. Колдаев В.Д., Павлова Е.Ю. Сборник задач и упражнений по информатике. - М.: ИД «ФОРУМ»: ИНФРА-М, 2010. – 256 с.
4. Марченко А.И., Марченко Л.А. Программирование в среде TurboPascal 7.0. – К.: ВЕК+, 1999. – 464 с.
5. Немцова Т.И., Голова С.Ю., Абрамова И.В. Программирование на языке высокого уровня. Программирование на языке ObjectPascal. - М.: ИД «ФОРУМ»: ИНФРА-М, 2012. – 496 с.
6. Окулов С.М. Основы программирования. – М.: БИНОМ. Лаборатория знаний, 2010. – 440 с.
7. Попов В. Б. TurboPascal для школьников. – М.: Финансы и статистика, 2000. – 528 с.
8. Цымбалюк Л.Н. Основы алгоритмизации и программирования: Практикум «Работа в программе BorlandPascal». – П.-К.: Камчатский кооперативный техникум, 2009. – 111 с.
9. Методические материалы и программное обеспечение: [Электронный ресурс], http://kpolyakov.narod.ru/, (Дата обращения 24.02 2015).
10. Кнут Д. Искусство программирования. Т.3. [Электронный документ], www.eknigi.org. (Дата обращения 15.04.2015).
11. Рублев В.С. Основы теории алгоритмов: учебное пособие. – Ярославль, 2005. [Электронный документ], www.ivt.corp7.uniyar.ac.ru . (Дата обращения 24.03.2015)
12. Фалина И.Н., Радченко Е.Л. «Изучение машины Поста в школьном курсе информатики». [Электронный документ], www.inf.1september.ru. (Дата обращения 30.03.2015)
13. Рабочая программа дисциплины «Теория алгоритмов».