Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»
Кафедра прикладного и системного программирования

ОСНОВЫ КОНСТРУИРОВАНИЯ ПРОГРАММ

Методические рекомендации

Витебск
ВГУ имени П.М. Машерова
2023

УДК 004.42:004.432(075.8)
ББК 32.973.22я73
О-75

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 7 от 26.04.2023.

Составитель*:* заведующий кафедрой прикладного и системного программирования ВГУ имени П.М. Машерова, кандидат физико-математических наук, доцент **Е.А. Корчевская**

Рецензент:
заведующий кафедрой информационных систем и технологий УО «ВГТУ», кандидат технических наук, доцент *В.Е. Казаков*

О-75 **Основы конструирования программ :** методические рекомендации / сост. Е.А. Корчевская. – Витебск : ВГУ имени П.М. Машерова, 2023. – 33 с.

В методических рекомендациях изложены общие подходы и основные принципы конструирования программ на языке С++. Даны краткие теоретические сведения, приведен список работ для самостоятельного выполнения.

Предназначаются для студентов специальностей «Прикладная информатика» (дисциплина «Алгоритмы и структуры данных»), «Информационные системы и технологии» (дисциплина «Основы конструирования программ»).

УДК 004.42:004.432(075.8)
ББК 32.973.22я73

© ВГУ имени П.М. Машерова, 2023

СОДЕРЖАНИЕ

Введение 4

1. Алгоритм и его свойства. Формализация понятия «алгоритм». Принципы разработки алгоритмов 5

1.1. Понятие алгоритма 5

1.2. Элементы блок-схем алгоритмов 8

2. Операторы управления программой 12

2.1. Оператор ветвления if 12

2.2. Оператор выбора switch 12

2.3. Циклический оператор for 14

2.4. Циклические операторы while и do/while 15

3. Способы упорядочивания информации 16

3.1. Массивы 16

3.2. Алгоритмы сортировок 18

4. Динамические структуры данных 20

4.1. Линейные динамические структуры данных 20

4.2. Нелинейные структуры данных. Графы 27

Литература 32

ВВЕДЕНИЕ

Методические рекомендации посвящены актуальному в настоящее время направлению программирования на языке С++. Материал разбит на 4 раздела.

Первый раздел посвящен понятию алгоритма, его свойствам. Здесь также приведены элементы блок-схем алгоритмов.

Во втором разделе описаны операторы управления программой. Даны примеры программ на языке С++, которые демонстрируют основные операторы.

В третьем разделе описаны алгоритмы сортировок (метод прямого выбора и метод прямого обмена), приведен код на языке программирования С++.

Четвертый раздел посвящен динамическим структурам данных.

Материал соответствует отдельным темам учебных программ курсов: «Алгоритмы и структуры данных» (специальность «Прикладная информатика»), «Основы конструирования программ» (специальность «Информационные системы и технологии»).

1. АЛГОРИТМ И ЕГО СВОЙСТВА. ФОРМАЛИЗАЦИЯ ПОНЯТИЯ «АЛГОРИТМ». ПРИНЦИПЫ РАЗРАБОТКИ АЛГОРИТМОВ

1.1. ПОНЯТИЕ АЛГОРИТМА

Для решения задачи исполнителю необходимо указать последовательность действий, которые он должен выполнить для достижения цели-получения результата.

Алгоритм – это заранее заданная последовательность однозначно описанных и трактуемых действий, позволяющих получить за конечное число шагов решение задачи, определяемое исходными данными.

Основные свойства алгоритма:

1. Детерминированность (определённость). Предписываемые алгоритмом действия должны быть однозначно описаны и не допускать различные трактовки. В каждый момент времени следующее действие должно однозначно определяться текущим состоянием системы. Таким образом, алгоритм выдаёт один и тот же результат для одних и тех же исходных данных.
2. Конечность (результативность) – при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат после выполнения конечного числа действий.
3. Массовость – возможность применять многократно один и тот же алгоритм для любой задачи одного класса или для решения одной задачи при различных исходных данных.
4. Выполнимость – алгоритм должен содержать описание только таких действий, которые являются выполнимыми в рамках средств, используемых для решения задачи.
5. Дискретность – алгоритм должен быть представлен как последовательное выполнение простых шагов. Шагом называется каждое действие алгоритма.

Оценка качества (эффективности) алгоритма определяется:

- скоростью сходимости,
- временем выполнения,
- удобством обращения к алгоритму,
- простотой,
- удобочитаемостью.

Для описания алгоритмов существуют различные способы. Наиболее распространенными являются: запись на подмножестве естественного языка, на псевдокоде, графическое представление (с помощью блок-схемы или диаграммы активности). Кроме того, текст программы на языке программирования также является формой записи алгоритма.

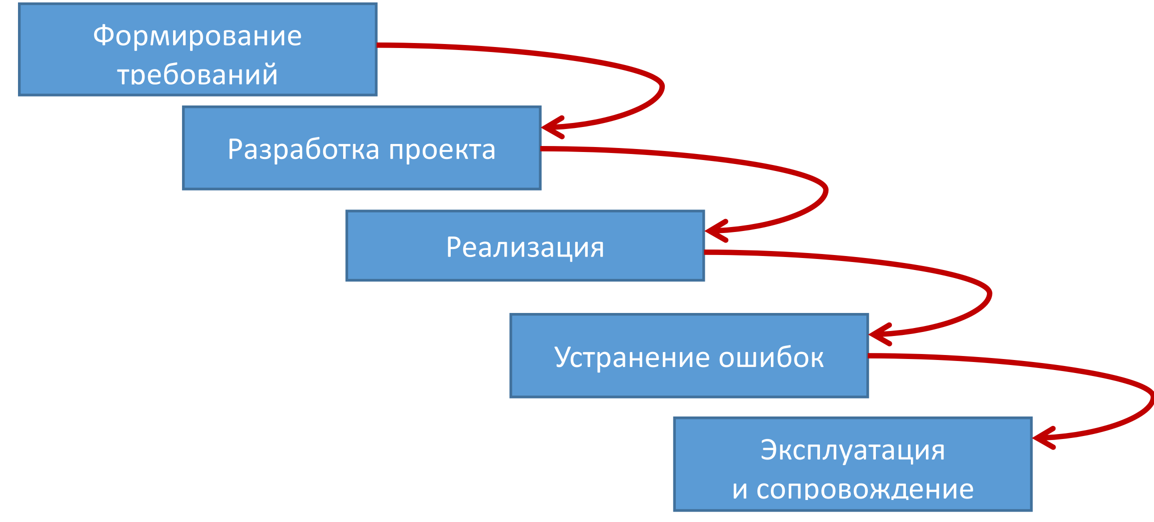
Типы алгоритмов: Линейные, Разветвленные, Циклические.

Процесс создания и использования программного обеспечения, представленный в виде последовательности этапов и выполняемых на этих этапах процессов называется жизненным циклом программного обеспечения.

Основные этапы жизненного цикла:

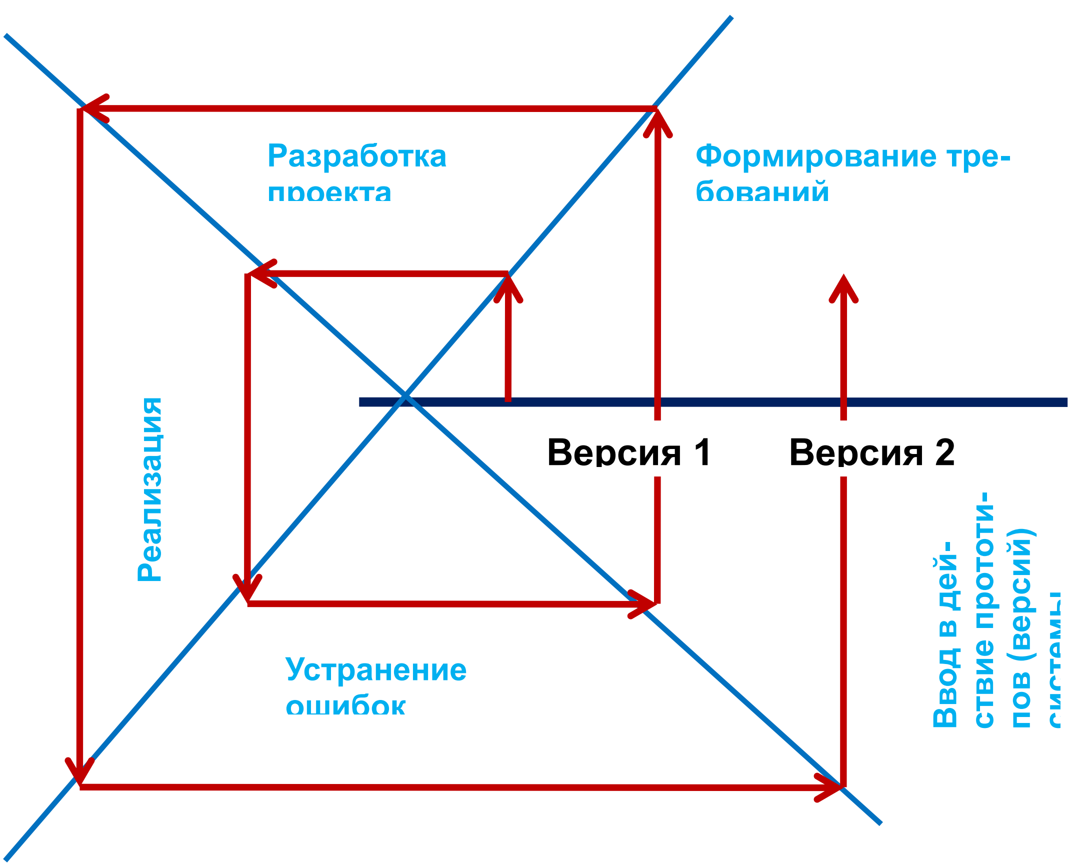
- Формирование требований – процесс сбора требований к системе, их систематизации, документирования, анализа, выявления противоречий и неполноты, разрешения конфликтов.
- Разработка проекта – деятельность по созданию проекта, то есть воспроизводимой модели программного обеспечения.
- Реализация – этап жизненного цикла программного обеспечения, объединяющий последовательные фазы создания программы в виде исходного кода, объектного кода и исполнимого кода. Результатом реализации является программа, которая может быть исполнена на компьютере.
- Устранение ошибок - процесс устранения причин того, что программное обеспечение не работает, либо результат его работы не соответствует выработанным требованиям.
- Эксплуатация – деятельность по использованию программного обеспечения для решения практических задач.
- Сопровождение – модификация программного обеспечения с целью устранения ошибок, реализации потребностей заказчика в улучшении тех или иных характеристик, а также его адаптации к использованию в модифицированном окружении.

Каскадная модель жизненного цикла («модель водопада») предусматривает последовательное выполнение всех этапов проекта в строго фиксированном порядке. Переход на следующий этап означает полное завершение работ на предыдущем этапе.



На практике этапы каскадной модели реализуются итерационно, с циклами обратной связи между этапами.

Спиральная модель жизненного цикла предусматривает спиралеобразное совершенствование системы путем последовательного создания прототипов (новых версий) этой системы. На каждом витке спирали при создании очередной версии продукта, уточняются требования проекта и планируются работы этого витка.



При разработке алгоритма и составлении программы мы можем столкнуться с тремя типами ошибок:

1. Синтаксические.

Ошибка в записи текста; например, пропущена закрывающая скобка в арифметическом выражении: (a*(b*(c+d))+a. Здесь три открывающиеся скобки и только 2 - закрывающиеся.

2. Семантические.

Например, пусть дана символьная строка S и число N. Запись вида Str1/Num будет семантически неверна, так как типы числителя и знаменателя несовместимы.

3. Логические.

Например, при вычислении расстояния S, которое пройдет автомобиль, движущийся со скоростью v за время t, мы вместо формулы S=v*t запишем S=v+t.

1.2. ЭЛЕМЕНТЫ БЛОК-СХЕМ АЛГОРИТМОВ

Блок-схема представляет собой совокупность символов, соответствующих этапам работы алгоритма и соединяющих их линий. Пунктирная линия используется для соединения символа с комментарием. Сплошная линия отражает зависимости по управлению между символами и может снабжаться стрелкой. Стрелку можно не указывать при направлении дуги слева направо и сверху вниз.

начало	Терминатором начинается и заканчивается любая функция. Тип возвращаемого значения и аргументов функции обычно указывается в комментариях к блоку терминатора.
Ввод (str)	Если источник данных не принципиален, обычно используется символ параллелограмма для ввода/вывода. Подробности ввода/вывода могут быть указаны в комментариях.
str := "hello world'	В блоке операций обычно размещают одно или несколько операций присваивания, не требующих вызова внешних функций.
a < b * c	Блок в виде ромба имеет один вход и несколько подписанных выходов. В случае, если блок имеет 2 выхода (соответствует оператору ветвления), на них подписывается результат сравнения – «да/нет». Если из блока выходит большее число линий (оператор выбора), внутри него записывается имя переменной, а на выходящих дугах – значения этой переменной.
buff := foo(1, tmp)	Вызов внешних процедур и функций помещается в прямоугольник с дополнительными вертикальными линиями.
LoopName a < b LoopName a < b	Символы начала и конца цикла содержат имя и условие. Условие может отсутствовать в одном из символов пары. Расположение условия, определяет тип оператора, соответствующего символам на языке высокого уровня – оператор с предусловием (while) или постусловием (do … while).

i := 1 .. 100	Символ «подготовка данных» в произвольной форме, задает входные значения. Используется обычно для задания циклов со счетчиком.
A	В случае, если блок-схема не умещается на лист, используется символ соединителя, отражающий переход потока управления между листами. Символ может использоваться и на одном листе, если по каким-либо причинам тянуть линию не удобно.
foo(i, str) выделяет i-тый символ строки str	Комментарий может быть соединен как с одним блоком, так и группой. Группа блоков выделяется на схеме пунктирной линией.

Лабораторная работа № 1
Алгоритмизация вычислительных процессов.
Блок-схемы. Разветвляющийся вычислительный процесс.
Циклический алгоритмический процесс»

Составить блок-схемы для решения следующих задач

Задание 1

1. Значения переменных X, Y, Z поменять местами так, чтобы они оказались упорядоченными по возрастанию.
2. Значения переменных X, Y, Z поменять местами так, чтобы они оказались упорядоченными по убыванию.
3. Даны две переменные целого типа: A и B. Если их значения не равны, то присвоить каждой переменной сумму этих значений, а если равны, то присвоить переменным нулевые значения.
4. Даны две переменные целого типа: A и B. Если их значения не равны, то присвоить каждой переменной максимальное из этих значений, а если равны, то присвоить переменным нулевые значения.
5. Даны три переменные: X, Y, Z. Если их значения упорядочены по убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное.
6. Даны три переменные: X, Y, Z. Если их значения упорядочены по возрастанию или убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное.
7. Даны целочисленные координаты точки на плоскости: Если точка не лежит на координатных осях, то вывести 0. Если точка совпадает с началом координат, то вывести 1. Если точка не совпадает с началом координат, но лежит на оси OX или OY, то вывести соответственно 2 или 3.

8. Даны вещественные координаты точки, не лежащей на координатных осях OXиOY. Вывести номер координатной четверти, в которой находится данная точка.

9. На числовой оси расположены три точки: A, B, C. Определить, какая из двух последних точек (B или C) расположена ближе к A, и вывести эту точку и ее расстояние отточки A.

10. Даны четыре целых числа, одно из которых отлично от трех других, равных между собой. Вывести порядковый номер этого числа.

Задание 2

1. Единицы длины пронумерованы следующим образом: 1 – дециметр, 2 – километр, 3 – метр, 4 – миллиметр, 5 – сантиметр. Дан номер единицы длины и длина отрезка L в этих единицах (вещественное число). Вывести длину данного отрезка в метрах.

2. Единицы массы пронумерованы следующим образом: 1 – килограмм, 2 – миллиграмм, 3 - грамм, 4 – тонна, 5 ' центнер. Дан номер единицы массы и масса тела M в этих единицах (вещественное число). Вывести массу данного тела в килограммах.

3. Робот может перемещаться в четырех направлениях ("С" – север, "З" – запад. "Ю" – юг, "В" – восток) и принимать три цифровые команды: 0 продолжать движение, 1 – поворот налево, 1 – поворот направо. Дан символ C – исходное направление робота и число N посланная ему команда. Вывести направление робота после выполнения полученной команды.

4. Локатор ориентирован на одну из сторон света ("С" – север, "З" – запад, "Ю" – юг, "В" – восток) и может принимать три цифровые команды: 1 – поворот налево, -1 – поворот направо, 2 – поворот на 180 градусов. Дан символ C – исходная ориентация локатора и числа N1 и N2 – две посланные ему команды. Вывести ориентацию локатора после выполнения данных команд.

5. Элементы окружности пронумерованы следующим образом:1 – радиус (R), 2 – диаметр (D), 3 – длина (L), 4 – площадькруга(S). Дан номер одного из этих элементов и его значение.Вывести значения остальных элементов данной окружности (в том же порядке). В качестве значения Pi использовать 3.14.

6. Элементы равнобедренного прямоугольного треугольника пронумерованы следующим образом: 1 – катет (a), 2– гипотенуза (c), 3 – высота, опущенная па гипотенузу (h), 4 – площадь (S). Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данного треугольника (втом же порядке).

7. Элементы равностороннего треугольника пронумерованы следующим образом: 1 – сторона (a), 2 – радиус вписанной окружности (R1), 3 – радиус описанной окружности (R2), 4 –площадь (S). Дан номер одного из этих элементов и его значение. Вывестизначения остальных элементов данного треугольника (в том же порядке).

8. Даны два целых числа: D (день) и M (месяц), определяющие правильную дату невисокосного года. Вывести значения D и M для даты, предшествующей указанной.

9. Даны два целых числа: D (день) и M (месяц), определяющие правильную дату невисокосного года. Вывести значения D и M для даты, следующей за указанной.

10. Дано целое число в диапазоне 20 - 69, определяющее возраст (в годах). Вывести строку – словесное описание указанного возраста, обеспечив правильное согласование числа со словом "год", например: 20 – "двадцать лет". 32 – "тридцать два года", 41–"сорок один год".

Задание 3

1. Дано вещественное число A и целое число N (> 0). Вывести $1 - A + A^2 - A^3 + ... + (-1)^N A^N$.

2. Дано целое число N (>1). Вывести наименьшее целое К, при котором выполняется неравенство $3^к > N$, и само значение $3^к$.

3. Дано целое число N (> 1). Вывести наибольшее целое К, при котором выполняется неравенство $3^к < N$, и само значение $3^к$.

4. Дано вещественное число A (> 1). Вывести наименьшее из целых чисел N, для которых сумма 1 + 1/2 + ... + 1/N будет больше A, и саму эту сумму.

5. Дано вещественное число A (> I). Вывести наибольшее из целых чисел N, для которых сумма 1 + 1/2 + ... + 1/N будет меньше A, и саму эту сумму.

6. Дано целое число N (> 0). Вывести произведение 1 * 2 * ... * N. Чтобы избежать целочисленного переполнения, вычислять это произведение с помощью вещественной переменной и выводить его как вещественное число.

7. Дано целое число N (> 0). Если N – нечетное, то вывести произведение 1 * 3 * ... * N; если N – четное, то вывести произведение 2 * 4* ... * N. Чтобы избежать целочисленного переполнения, вычислять это произведение с помощью вещественной переменной и выводить его как вещественное число.

8. Дано целое число N (> 0). Вывести сумму 2 + 1/(2!) +1/(3!) + ... + 1/(N!) (выражение N! – "N факториал" обозначает произведение всех целых чисел от 1 доN: N! = 1 * 2 * ... * N). Полученное число является приближенным значением константы e= exp(1) (e= 2.71828183...).

9. Дано вещественное число X и целое число N (>0). Вывести $1 + X ++X^2/2! + ... + X^N/N!$ (N! = 1 * 2 * ... * N). Полученное число является приближенным значением функции exp в точке X.

10. Дано вещественное число X и целое число N (> 0). Вывести $1 - X^2/2! +... + (-1)^N X^{2N}/(2N)!$ (N! =1 * 2 * ... * N). Полученное число является приближенным значением функции cos в точке X.

2. ОПЕРАТОРЫ УПРАВЛЕНИЯ ПРОГРАММОЙ

2.1. ОПЕРАТОР ВЕТВЛЕНИЯ IF

Стандартная форма оператора if следующая:

```
if (выражение) {
   последовательность операторов 1;
}
else {
   последовательность операторов 2;
}
```

Если выражение истинно (любое значение, кроме 0), выполняется блок операторов, следующий за if, иначе выполняется блок операторов, следующих за else. Всегда выполняется код, ассоциированный или с if, или с else, но никогда не выполняются оба кода одновременно. Если требуется проверить несколько условий, их объединяют знаками логических операций.

2.2. ОПЕРАТОР ВЫБОРА SWITCH

Язык С++ имеет оператор принятия решений switch, выполняющий действия, основываясь на сравнении значения со списком констант символов или целых чисел. При обнаружении совпадения выполняется оператор или операторы, ассоциированные с данным значением. Оператор switch имеет следующий вид:

```
switch (выражение) {
   case константа1:
      последовательность операторов
        break;
   case константа2:
       последовательность операторов
        break;
        …
  case константаn:
       последовательность операторов
        break;
  default:
       последовательность операторов
 }
```

Оператор default выполняется, если не найдено соответствий, default необязателен и, если его нет, то в случае отсутствия совпадений ничего не происходит. Когда обнаруживается совпадение, операторы, ассоциированные с соответствующим case, выполняются до тех пор, пока не встретится

оператор break. В случае default (или последнего case, если отсутствует default), оператор switch заканчивает работу при обнаружении конца.

Следует знать о трех важных моментах оператора switch:

1. switch отличается от if тем, что он может выполнять только операции проверки строгого равенства, в то время как if может вычислять логические выражения и отношения.
2. Не может быть двух констант в одном операторе switch, имеющих одинаковые значения. Конечно, оператор switch, включающий в себя другой оператор switch, может содержать аналогичные константы.
3. Если в операторе switch используются символьные константы, они автоматически преобразуются к целочисленным значениям.

Пример: Простейший калькулятор.

```
#include <iostream>
using namespace std;
int main() {
   double x, y, result;
   char operation;
   bool ok = true;
   cin >> x >> operation >> y;
   switch (operation){
      case '+': result = x + y; break;
      case '-': result = x - y; break;
      case '*': result = x * y; break;
      case '/':  if (y !=  0)
                     result = x / y;
                  else
                     ok = false;
                 break;
     default : cout<< "Error" ; ok = false;
  }
 if (ok)
     cout <<  "result : " << result  << endl;
 return 0;
}
```

2.3. ЦИКЛИЧЕСКИЙ ОПЕРАТОР FOR

Операторы цикла используются для организации многократно повторяющихся вычислений. Любой цикл состоит из тела цикла, то есть тех операторов, которые выполняются несколько раз, начальных установок, модификации параметра цикла и проверки условия продолжения выполнения цикла.

Один проход цикла называется итерацией. Проверка условия выполняется на каждой итерации либо до тела цикла (тогда говорят о цикле с предусловием), либо после тела цикла (цикл с постусловием). Разница между ними состоит в том, что тело цикла с постусловием всегда выполняется хотя бы один раз, после чего проверяется, надо ли его выполнять еще раз. Проверка необходимости выполнения цикла с предусловием делается до тела цикла, поэтому возможно, что он не выполнится ни разу.

Переменные, изменяющиеся в теле цикла и используемые при проверке условия продолжения, называются параметрами цикла. Целочисленные параметры цикла, изменяющиеся с постоянным шагом на каждой итерации, называются счетчиками цикла.

Стандартный вид цикла for следующий:

```
for (инициализация; условие; изменение) {
    тело цикла;
}
```

Оператор for имеет три главные части:

1. Инициализация – это место, где обычно находится оператор присваивания, используемый для установки начального значения переменной цикла.
2. Условие– это место, где находится выражение, определяющее условие работы цикла.
3. Изменение – это место, где определяется характер изменения переменной цикла на каждой итерации.

Пример: Программа осуществляет ввод положительного целого числа и вычисление суммы n нечетных чисел.

```
#include <iostream>
using namespace std;
int main(){
  int Sum, i, n;
  cin  >> n ;
  Sum = 0;
  if (n > 0)  {
     for (i = 1; i <= n; i++)
         Sum += 2*i +1;
  }
   else
       cout  <<  "Error"  <<  '\n';
    return 0;
}
```

2.4. ЦИКЛИЧЕСКИЕ ОПЕРАТОРЫ WHILE И DO/WHILE

Циклический оператор while с предусловием, в котором сначала проверяется условие, и, если оно ложно, то цикл не выполняется и управление передается на следующую инструкцию после тела цикла while. Если условие истинно, то выполняется инструкция, после чего условие проверяется снова, и снова выполняется инструкция. Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей инструкции после цикла.

Синтаксис цикла с предусловием:

while (условие){

тело цикла

}

В следующем примере цикл используется для того, чтобы найти количество знаков в десятичной записи целочисленной переменной n.

```
int Ndigits = 0;
while(n! = 0) {
     Ndigitы++;
     N /= 10;
}
```

Внутри цикла значение переменной n уменьшается в 10 раз до тех пор, пока она не станет равна 0. Уменьшение целочисленной переменной в 10 раз (с использованием целочисленного деления) эквивалентно отбрасыванию последней цифры этой переменной.

Циклический оператор do/while в противоположность циклам for и while, сначала проверяющим условие, проверяет условие в конце. То есть цикл do/while всегда выполняется, по крайней мере, один раз. Стандартный вид цикла do/while следующий:

do {

последовательность операторов;

} while (условие);

Лабораторная работа № 2
Разветвляющийся вычислительный процесс.
Циклический алгоритмический процесс

На языке программирования С++ написать программы для решения задач из лабораторной работы № 1.

3. СПОСОБЫ УПОРЯДОЧИВАНИЯ ИНФОРМАЦИИ

3.1. МАССИВЫ

Массив – это последовательная группа ячеек памяти, имеющих одинаковое имя и одинаковый тип. Чтобы сослаться на отдельную ячейку или элемент массива, мы указываем имя массива и номер позиции отдельного элемента массива.

Имена массивов должны удовлетворять тем же требованиям, которые предъявляются к другим именам переменных. Номер позиции, указанный внутри квадратных скобок, называется индексом. Индекс должен быть целым числом или целым выражением.

Массивы занимают область в памяти. Программист указывает тип каждого элемента, количество элементов, требуемое каждым массивом, и компилятор может зарезервировать соответствующий объем памяти. Чтобы указать компилятору зарезервировать память для 12 элементов массива целых чисел с, используется объявление

int Array[12];

Пример нахождения суммы элементов массива.

```
    #include <iostream>
    using namespace std;
    int main(void) {
    // Описание массива
    int A[10];
    // Определение количества элементов в массиве
    int N = 10;
    // Заполнение массива
    for(int I = 0;i < N; i++) {
      cout << "A[" << i << "]= ";
      cin >> A[i];
    }
     // Обработка элементов массива
    int Sum=0;
    for(int i = 0;i < N;i++)
                Sum+=A[i];
    cout << "Сумма элементов массива равна " << Sum <<
endl;
     // Вывод элементов массива
    for(int i=0;i<N;i++) cout << A[i] << ',';
    cout << endl;
    system ("pause");
    return 0;
    }
```

Элементам массива можно присваивать начальные значения (инициализировать их) в объявлении массива с помощью следующего за объявлением списка (заключенного в фигурные скобки) одинаковых по смыслу и разделенных запятыми начальных значений.

int n[10] = {32, 27, 64, 18, 95, 14, 90, 70, 60, 37};

Если начальных значений меньше, чем элементов в массиве, оставшиеся элементы автоматически получают нулевые начальные значения. Например, элементам массива n можно присвоить нулевые начальные значения с помощью объявления

int n[10] = {0};

которое явно присваивает нулевое начальное значение первому элементу и неявно присваивает нулевые начальные значения оставшимся девяти элементам, потому что здесь начальных значений меньше, чем элементов массива.

Размерность массива вместе с типом его элементов определяет объем памяти, необходимый для размещения массива, которое выполняется на этапе компиляции, поэтому размерность может быть задана только целой положительной *константой или константным выражением.* Если при описании массива не указана размерность, должен присутствовать инициализатор, в этом случае компилятор выделит память по количеству инициализирующих значений.

Пример нахождения максимального из элементов массива.

```
#include <iostream>
using namespace std;
int main(void) {
// Описание массива
float A[10];
// Определение количества элементов в массиве
int N;
cout << "Введите количество элементов массива (не бо-
лее 10) ---> ";
cin >> N;
// Заполнение массива
for(int i=0;i<N;i++)
      A[i]=rand();
// Обработка элементов массива
float Max=A[0];
for(int i=1;i<N;i++)
  if (Max<A[i])
        Max=A[i];
cout << "Наибольший элемент массива равен " << Max <<
endl;
// Вывод элементов массива
for(int i=0;i<N;i++)
      cout << A[i] << endl;
return 0;
}
```

Двумерные массивы представляют собой таблицу значений, содержащую информацию в строках и столбцах. Чтобы определить отдельный табличный элемент, нужно указать два индекса: первый (по соглашению) указывает номер строки, а второй (по соглашению) указывает номер столбца. Таблицы или массивы, которые требуют двух индексов для указания отдельного элемента, называются двумерными массивами.

Пример:

```
#include<iostream>
using namespace std;
void main(){
 double a[20][2];
 for ( int i = 0; i <2 0; i++)
   for (int j  =  0; j  <  2; j++){
      a[i][j] = i*j;
      cout << a [i][ j ] << ' ' ;
   }
return;
}
```

3.2. АЛГОРИТМЫ СОРТИРОВОК

Под сортировкой массива подразумевается процесс перестановки элементов массива, целью которого является размещение элементов массива в определенном порядке. Например, если имеется массив целых чисел a, то после выполнения сортировки по возрастанию должно выполняться условие:

$a[1] \le a[2] \le .. . \le a[SIZE]$,

где SIZE – верхняя граница индекса массива.

При решении задач сортировки выдвигается требование минимальности использования памяти. Дополнительные массивы не заводят.

При оценки быстродействия алгоритма сортировки используют 2 показателя:

1. Количество присваиваний
2. Количество сравнений.

Все методы сортировки можно разделить на 2 группы: прямые и улучшенные.

К прямым относятся: метод прямого выбора, метод прямого обмена и метод вставкой. Улучшенные методы основаны на тех же принципах, что и прямые, но используют некоторые идеи, способствующие улучшению сортировки.

Сортировка методом прямого выбора

Алгоритм сортировки массива по возрастанию методом прямого выбора может быть представлен так:

1. Просматривая массив от первого элемента, найти минимальный элемент и поместить его на место первого элемента, а первый – на место минимального.
2. Просматривая массив от второго элемента, найти минимальный элемент и поместить его на место второго элемента, а второй – на место минимального.
3. И так далее до предпоследнего элемента.

Далее представлена программа сортировки массива целых чисел по возрастанию. Для демонстрации процесса сортировки программа выводит массив после каждого обмена элементов.

```
#include<iostream>
#include <iomanip>
using namespace std;
void main(){
    const int ArraySize = 10;
    int a[ArraySize] = {2,6,4,8,10,77,1,100,-1,0};
    int i, min, j, buf, k;
    for ( i = 0; I < ArraySize-1; i++){
         min = i;
        for ( j = i+1; j < ArraySize; j++)
              if (a[j] < a[min])
                   min = j;
       buf = a[i];
       a[i] = a[min];
       a[min] = buf;
      for (k = 0; k < ArraySize; k++)
       cout << setw(4) << a[k];
            cout << endl;
   }
  return;
}
```

Сортировка методом прямого обмена (пузырька)

В основе алгоритма лежит обмен соседних элементов массива. Каждый элемент массива, начиная с первого, сравнивается со следующим, и если он больше следующего, то элементы меняются местами. Таким образом, элементы с меньшим значением продвигаются к началу массива (всплывают), а элементы с большим значением – к концу массива (тонут), поэтому этот метод иногда называют методом «пузырька». Этот процесс повторяется на единицу меньше раз, чем элементов в массиве.

```
#include<iostream>
#include <iomanip>
using namespace std;
void main(){
   const int ArraySize = 10;
   int a[ArraySize] ={2, 6, 4, 8, 10, 77, 1, 100, -1,
0};
   int i, buf, k;
   for ( i =1;  i <= ArraySize-1; i++){
     for ( k = 0; k < ArraySize-i; k++)
          if (a[k] > a[k+1]) {
               buf = a[k];
               a[k] = a[k+1];
               a[k+1] = buf;
      }
     for (k = 0; k < ArraySize; k++)
       cout << setw(4) <<a[k];
     cout << endl;
   }
  return;
}
```

4. ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

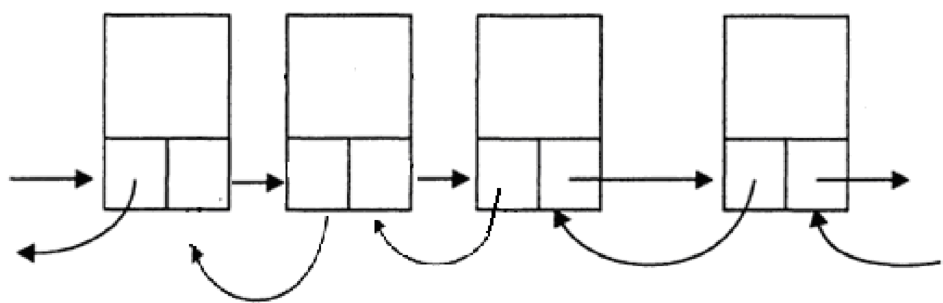
4.1. ЛИНЕЙНЫЕ ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

Если до начала работы с данными невозможно определить, сколько памяти потребуется для их хранения, то память выделяется по мере необходимости отдельными блоками, связанными друг с другом с помощью указателей. Такой способ организации данных называется динамическими структурами данных, поскольку их размер изменяется во время выполнения программы. Из динамических структур в программах чаще всего используются линейные списки, стеки, очереди и бинарные деревья. Они различаются способами связи отдельных элементов и допустимыми операциями. Динамическая структура может занимать несмежные участки оперативной памяти.

Элемент любой динамической структуры данных представляет собой структуру, содержащую по крайней мере два поля: для хранения данных и для указателя. Полей данных и указателей может быть несколько. Поля данных могут быть любого типа: основного, составного или типа указатель.

1. Линейные списки

Самый простой способ связать множество элементов – сделать так, чтобы каждый элемент содержал ссылку на следующий. Такой список называется однонаправленным (односвязным). Если добавить в каждый элемент вторую ссылку – на предыдущий элемент, получится двунаправленный список (двусвязный), если последний элемент связать указателем с первым, получится кольцевой список.



Над списками можно выполнять следующие операции:

• начальное формирование списка (создание первого элемента);

• добавление элемента в конец списка;

• чтение элемента с заданным ключом;

• вставка элемента в заданное место списка (до или после элемента с заданным ключом);

• удаление элемента с заданным ключом;

• упорядочивание списка по ключу.

Рассмотрим двунаправленный линейный список. Для формирования списка и работы с ним требуется иметь по крайней мере один указатель – на начало списка. Удобно завести еще один указатель – на конец списка. Для простоты допустим, что список состоит из целых чисел, то есть описание элемента списка выглядит следующим образом:

```
 struct Unit{
     int d;
     Unit *next;
     Unit *prev;
};
```

Ниже приведена программа, которая формирует список из 5 чисел, добавляет
число в список, удаляет число из списка и выводит список на экран. Указатель на начало списка обозначен begin, на конец списка – end, вспомогательные указатели – pv и pkey.

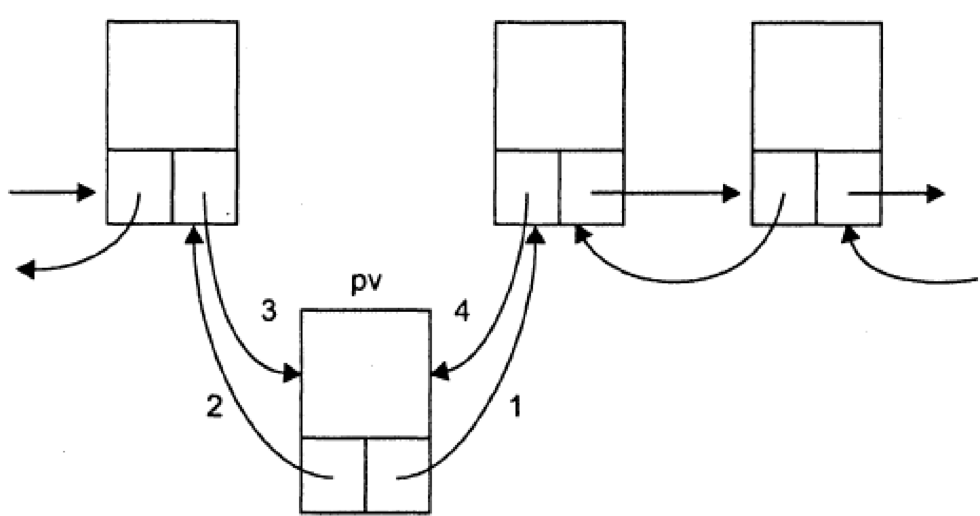
```
#include <iostream>
using namespace std;
struct Unit{
   int d;
```

```
   Unit *next;
   Unit *prev;
};
Unit * first(int d);
void add(Unit **end, int d);
Unit * find(Unit * const begin, int i);
bool remove(Unit **begin, Unit **end, int key);
Unit * insert(Unit * const begin, Unit **end, int
key, int d);
int main(){
Unit *begin = first(1);
Unit *end = begin;
for (int i = 2; i<6; i++)
     add(&end, i);
insert(begin, &end, 2, 200);
if(!remove (&begin, &end, 5))
     cout << "не найден";
Unit *pv = begin;
while (pv){
    cout << pv->d << ' ';
    pv = pv->next;
}
  cin.get();
return 0;
}
// Формирование первого элемента
Unit * first(int d){
   Unit *pv = new Unit;
   pv->d = d;
   pv->next = 0;
   pv->prev = 0;
   return pv;
}
// Добавление в конец списка
void add (Unit **end, int d){
   Unit *pv = new Unit;
   pv->d = d; pv->next = 0; pv->prev = *end;
   (*end)->next = pv;
   *end = pv;
}
// Поиск элемента по ключу
Unit * find(Unit * const begin, int d){
   Unit *pv = begin;
   while (pv){
   if(pv->d == d)
```

```
        break;
   pv = pv->next;
  }
return pv;
}
// Удаление элемента
bool remove(Unit **begin, Unit **end, int key){
   if(Unit *pkey = find(*begin, key)){
   if (pkey == *begin){
/
   *begin = (*begin)->next;
   (*begin)->prev =0;
   }
else
   if (pkey == *end){
    *end = (*end)->prev;
    (*end)->next =0;
  }
else{
   (pkey->prev)->next = pkey->next;
   (pkey->next)->prev = pkey->prev;
}
delete pkey;
return true;
}
return false;
}
// Вставка элемента
Unit * insert (Unit *const begin, Unit **end, int
key, int d){
   if(Unit *pkey = find(begin, key)){
     Unit *pv = new Unit;
     pv->d = d;
     pv->next = pkey->next;
    pv->prev = pkey;
    pkey->next = pv;
    if( pkey!= *end)
       (pv->next)->prev = pv;
   else
      *end = pv;
   return pv;
}
return 0;
}
```

Рассмотрим подробнее функцию удаления элемента из списка remove. Ее параметрами являются указатели на начало и конец списка и ключ элемента, подлежащего удалению. Сначала выделяется память под локальный указатель pkey, которому присваивается результат выполнения функции нахождения элемента по ключу find. Эта функция возвращает указатель на элемент в случае успешного поиска и 0, если элемента с таким ключом в списке нет. Далее рассматривается два случая: если pkey получает ненулевое значение, условие в операторе if становится истинным (элемент существует) и, если нет – выполняется возврат из функции со значением false. Удаление из списка происходит по-разному в зависимости от того, находится элемент в начале списка, в середине или в конце.

Работа *функции вставки элемента* в список проиллюстрирована на рис. Номера около стрелок соответствуют номерам операторов в комментариях.



2. Стеки

Добавление элементов в стеке и выборка выполняются с одного конца, называемого вершиной стека. Другие операции со стеком не определены. При выборке элемент исключается из стека. Говорят, что стек реализует принцип обслуживания LIFO (last in – first out, последним пришел – первым ушел). Стек проще всего представить себе как закрытую с одного конца узкую трубу, в которую бросают мячи. Достать первый брошенный мяч можно только после того, как вынуты все остальные.

Рассмотрим пример формирования стека из пяти целых чисел (1,2, 3, 4, 5) и выводит его на экран. Функция помещения в стек по традиции называется push, а выборки – pop. Указатель для работы со стеком (top) всегда ссылается на его вершину.

```
#include <iostream>
using namespace std;
```

```
struct Unit{
    int d;
    Unit *p;
};
Unit * first( int d);
void push (Unit **top, int d);
int pop(Unit **top);

int main(){
Unit *top = first(1);
for (int i = 2; i<6; i++)push(&top, i);
while (top)
cout << pop(&top) << ' ';
  cin.get();
return 0;
}
// Начальное формирование стека
Unit * first (int d){
Unit *pv =new Unit;
pv->d = d;
pv->p = 0;
return pv;
}
// Занесение в стек
void push(Unit **top, int d){
Unit *pv = new Unit;
pv->d = d;
pv->p = *top;
*top = pv; }
// Выборка из стека

int pop (Unit **top){
int temp = (*top)->d;
Unit *pv = *top;
*top = (*top)->p;
delete pv;
return temp;
 }
```

3. Очереди

Очередь – это частный случай однонаправленного списка, добавление элементов в который выполняется в один конец, а выборка – из другого конца. Другие операции с очередью не определены. При выборке элемент исключается из очереди. Говорят, что очередь реализует принцип обслуживания FIFO (first in – first out, первым пришел – первым ушел).

Далее рассмотрим пример, в котором формируется очередь из пяти целых чисел и выводится на экран. Функция помещения в конец очереди называется add, а выборки– del. Указатель на начало очереди называется begin, указатель на конец –end.

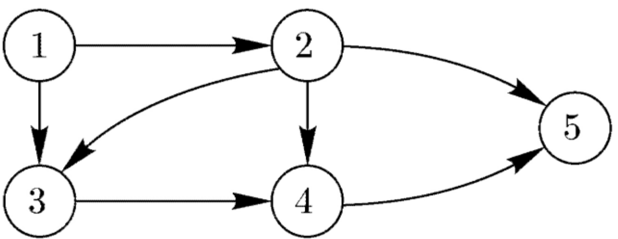
```
#include <iostream >
using namespace std;
struct Unit{
   int d;
   Unit *p;
};
Unit * first (int d);
void add (Unit **end, int d);
int del(Unit **begin);
int main(){
Unit *begin = first(1);
Unit *end = begin;
for (int i = 2; i<6; i++)add(&end, i);
while (begin)
cout << del (&begin) << ' ';
  cin.get();
return 0;
}
// Начальное формирование очереди
Unit * first(int d){
Unit *pv = new Unit;
pv->d = d;
pv->p = 0;
 return pv;
}
// Добавление в конец
void add(Unit **end, int d){
Unit *pv = new Unit;
pv->d = d;
pv->p = 0;
(*end)->p = pv;
*end = pv;}
// Выборка
int del(Unit **begin){
int temp = (*begin)->d;
Unit *pv = *begin;
*begin = (*begin)->p;
delete pv;
return temp;
}
```

4.2. НЕЛИНЕЙНЫЕ СТРУКТУРЫ ДАННЫХ. ГРАФЫ

Граф **G = (V, E)** состоит из непустого множества узлов (вершин) **V** и множества пар узлов **E**. Будем предполагать, что |V| = n и |E|=m.

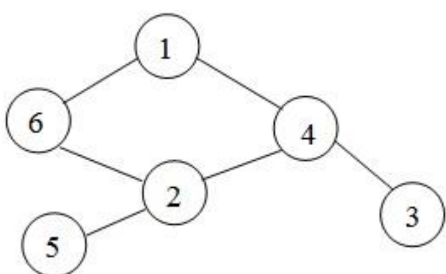
Если множество **E** представлено в виде упорядоченных пар узлов **(v,w)**, то

- ✓ граф называется **ориентированным** (орграфом),
- ✓ упорядоченная пара узлов (v, w) называется **дугой** (v – начало, w – конец дуги)



Если множество **E** представлено в виде неупорядоченных пар узлов **(v,w)**, то

- – граф называется **неориентированным**,
- – неупорядоченная пара узлов (v, w) называется **ребром**



Путем в ориентированном графе (**цепью** в неориентированном графе) называется последовательность вершин вида $v_1, v_2,\dots, v_{k-1}, v_k$, где $(v_i, v_{i+1})\in E$, $i=1..k-1$

Путь называется **простым**, если все узлы различны.

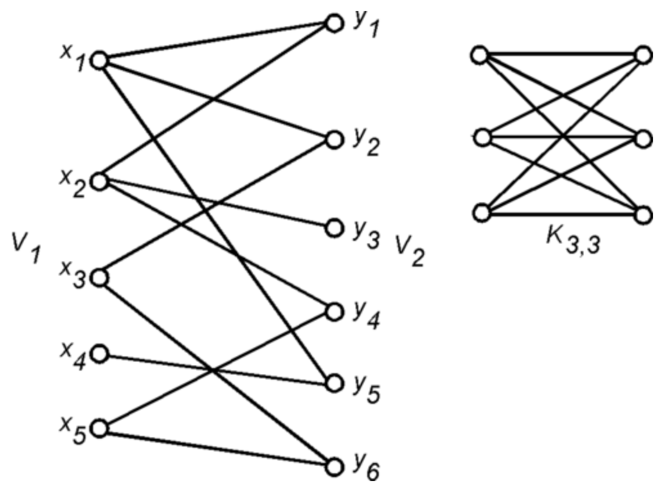
Цикл – это простой путь, который начинается и заканчивается в одном и том же узле.

Граф называется **связным**, если для любой пары вершин существует простой путь, их соединяющий; в противном случае, граф называется **несвязным**.

Если ***v*** – некоторая вершина графа, то максимальное количество вершин, в которые существует путь из ***v*** порождают **компоненту связности** графа.

Граф называется **двудольным**, если множество его вершин можно разбить на два подмножества (доли) таким образом, что каждое ребро соединяет только вершины различных подмножеств.

Граф называется **полным двудольным**, если каждая вершина одной доли соединена с каждой вершиной другой доли.



Представление графа в памяти

1. Матрица смежности
2. Матрица инцидентности
3. Список пар, соответствующим ребрам
4. Список смежности

Матрицей смежности графа **G** называется такая матрица **A** размера **n**x**n**, в которой:

– $A_{vw} = 1$, если существует ребро (дуга) из вершины ***v*** в ***w***
– $A_{vw} = 0$, в противном случае

Преимущество - за O(1) можно дать ответ на вопрос "существует ли ребро (дуга) (v, w) ?".

Недостаток – объем требуемой памяти вне зависимости от количества ребер (дуг) составляет n^2.

	1	2	3	4
1	0	1	0	1
2	0	0	1	1
3	0	1	0	0
4	1	0	1	0

Матрицей инцидентности графа **G** называется такая матрица **A** размера **n**x**m** (n – количество вершин, m – количество ребер).

Для орграфа столбец, соответствующий дуге (v, w), содержит :

- 1 в строке соответствующей вершине v
- –1 в строке, соответствующей вершине w
- 2 - петля
- 0 во всех остальных строках

Список пар

- Объем памяти в этом случае составляет O(m).
- Неудобство заключается в большом количестве шагов (в худшем случае порядка m), необходимом для получения множества вершин, к которым ведут ребра из данной вершины.

Список смежности

- Массив graf – статический массив, элемент graf[i] – указатель на начало линейного списка вершин, смежных с i-ой вершиной; если из вершины i нет выходящих дуг, то graf[i] = null.
- Граф представ виде n списков смежностей, по одному для каждой вершины.
- Объем памяти в этом случае O (m + n).

Лабораторная работа № 3
Задачи на графах

1. Задан набор неповторяющихся пар (A_i,A_j), A_i, A_j принадлежат множеству $A=\{A_1, A_2, ..., A_n\}$. Необходимо составить цепочку максимальной длины по правилу
$(A_i,A_j)+(A_j,Ak)=(A_i,A_j,A_k)$.
При образовании этой цепочки любая пара может быть использована не более одного раза.

2. Найти кратчайшее расстояние между двумя вершинами в графе. Найти все возможные пути между этими двумя вершинами в графе не пересекающиеся по вершинам.

3. Лабиринт задается матрицей смежности N*N, где C(i,j)=1, если узел i связан узлом j посредством дороги. Часть узлов назначается входами, часть – выходами. Входы и выходы задаются последовательностями узлов X(1),..,X(p) и Y(1),..,Y(k) соответственно.

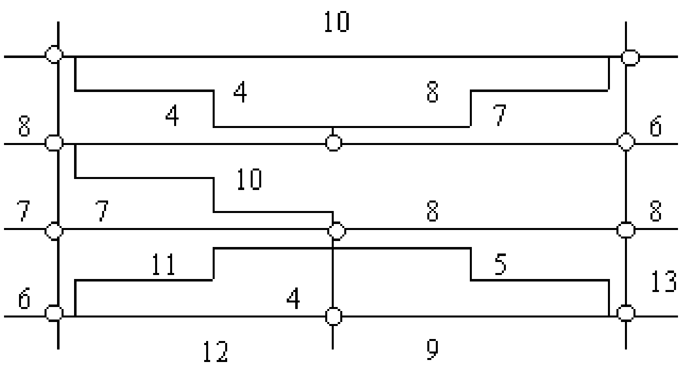
Найти максимальное число людей, которых можно провести от входов до выходов таким образом, чтобы:

а) их пути не пересекались по дорогам, но могут пересекаться по узлам;

б) их пути не пересекались по узлам;

4. Задан граф. Необходимо найти наибольшее число не пересекающихся по ребрам маршрутов между заданными вершинами υ и ω.

5. Следующая фигура показывает запутанную сеть дорог района города. Представьте, что мусорная машина должна пройти по всем дорогам хотя бы один раз, чтобы собрать мусор. Число на каждой стороне показывает время, которое должна потратить мусорная машина, чтобы проехать этот интервал. На перекрестках машина должна ждать время, равное числу пересекающихся дорог.



Составьте программу, показывающую как выбрать необходимый путь для сбора мусора в кратчайшее время.
Есть 11 остановок.
от 1 до 2 путь 10 мин.
от 1 до 3 4
от 1 до 4 8
от 2 до 3 8
от 2 до 5 6
от 3 до 4 4
от 3 до 5 7
от 4 до 7 7
от 4 до 6 10
от 4 до 8 7
от 8 до 6 7
от 8 до 10 6
от 10 до 6 11
от 6 до 9 4
от 10 до 9 12
от 6 до 11 5
от 6 до 4 8
от 5 до 4 8
от 4 до 11 13
от 9 до 11 5

6. Вводится N – количество домов и K – количество дорог. Дома пронумерованы от 1 до N. Каждая дорога определяется тройкой чисел – двумя номерами домов – концов дороги и длиной дороги. В каждом доме живет по

одному человеку. Найти точку – место встречи всех людей, от которой суммарное расстояние до всех домов будет минимальным.
Если точка лежит на дороге, то указать номера домов - концов этой дороги и расстояние от первого из этих домов. Если точка совпадает с домом, то указать номер этого дома. Примечание: длины дорог – положительные целые числа.

7. Найти кратчайшее расстояние между двумя вершинами в графе. Найти все возможные пути между этими двумя вершинами в графе не пересекающиеся по ребрам.

8. Реализовать алгоритм поиска в глубину.

9. Реализовать алгоритм поиска в ширину.

10. Реализовать топологическую сортировку для ориентированного ациклического графа.

ЛИТЕРАТУРА

1. Страуструп, Б. Язык программирования С++ / Б. Страуструп. – М.: Бином, 2022.
2. Шупляк, В.И. С++. Практический курс: учеб. пособие / В.И. Шупляк. – Минск: Новое знание, 2011.
3. Шилдт, Г. Искусство программирования на С++/ Г. Шилдт. – СПб.: БХВ-Петербург, 2005.
4. Керниган, Б. Язык программирования С / Б. Керниган, У. Ритчи, М. Денис; пер. с англ. – 2-е изд. – М.: Издат. дом «Вильямс», 2008.
5. Павловская, Т.А. С/С++. Программирование на языке высокого уровня / Т.А. Павловская. – СПб.: Питер, 2021.