**Лабораторная работа 1. Вспомогательные функции**

**ЦЕЛЬ РАБОТЫ:** приобретение навыков составления и отладки программ с использованием пользовательских функций для замера продолжительности процесса вычисления.

***Задание 1.*** Разработайте три функции (start, dget и iget), используя следующие спецификации:

**Auxil.cpp**

|  |
| --- |
| //-- Auxil.cpp  //#include "stdafx.h"  #include "Auxil.h"  #include <ctime>  #include <iostream>  namespace auxil  {  void start() // старт генератора сл. чисел  {  srand((unsigned)time(NULL));  };  double dget(double rmin, double rmax) // получить случайное число  {  return ((double)rand() / (double)RAND\_MAX) \* (rmax - rmin) + rmin;  };  int iget(int rmin, int rmax) // получить случайное число  {  return (int)dget((double)rmin, (double)rmax);  };  } |

**Auxil.h**

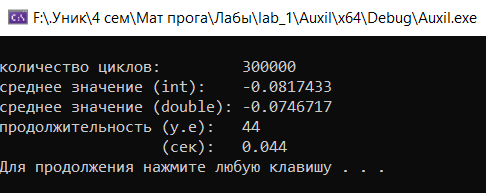
|  |
| --- |
| #pragma once  //-- Auxil.h  #pragma once  #include <cstdlib>  namespace auxil  {  void start(); // старт генератора сл. чисел  double dget(double rmin, double rmax); // получить случайное число  int iget(int rmin, int rmax); // получить случайное число  }; |

***Задание 2***

Для проверки работоспособности разработанных функций и приобретения навыков замера продолжительности процесса вычисления реализуется программа ниже.

|  |
| --- |
| #include "Auxil.h" // вспомогательные функции  #include <iostream>  #include <ctime>  #include <locale>  #include <stdlib.h>  #define CYCLE 300000 // количество циклов  int main()  {  double av1 = 0, av2 = 0;  clock\_t t1 = 0, t2 = 0;  setlocale(LC\_ALL, "rus");  auxil::start(); // старт генерации  t1 = clock(); // фиксация времени  for (int i = 0; i < CYCLE; i++)  {  av1 += (double)auxil::iget(-100, 100); // сумма случайных чисел  av2 += auxil::dget(-100, 100); // сумма случайных чисел  }  t2 = clock(); // фиксация времени  std::cout << std::endl << "количество циклов: " << CYCLE;  std::cout << std::endl << "среднее значение (int): " << av1 / CYCLE;  std::cout << std::endl << "среднее значение (double): " << av2 / CYCLE;  std::cout << std::endl << "продолжительность (у.е): " << (t2 - t1);  std::cout << std::endl << " (сек): " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl;  system("pause");  return 0;  } |

Результат выполнения



**Задание 3**

Проведём необходимые эксперименты (разработаем кодом) и построим график зависимости (Excel) продолжительности процесса вычисления от количества циклов.

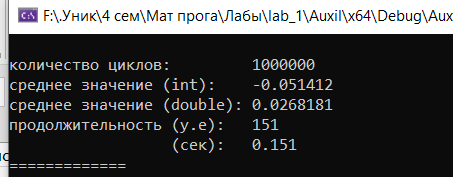


График зависимости:

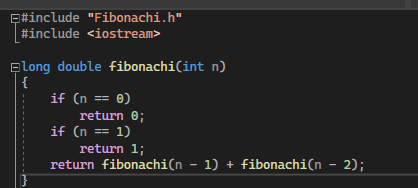


Вывод: Линейная зависимость O(n)

Вывод: скорость выполнения программы линейно зависит от количества итераций цикла.

В качестве второго алгоритма был выбран алгоритм подсчета Фибоначчи

Исполнительный файл для решения алгоритма Фибоначчи.



Результат выполнения алгоритма Фибоначчи.

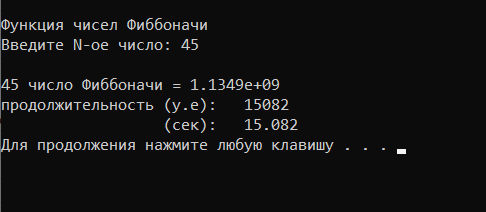


График зависимости продолжительности от порядкового номера Фибоначчи



Проанализировав график было выявлено, что сложность данного алгоритма является экспоненциальной.

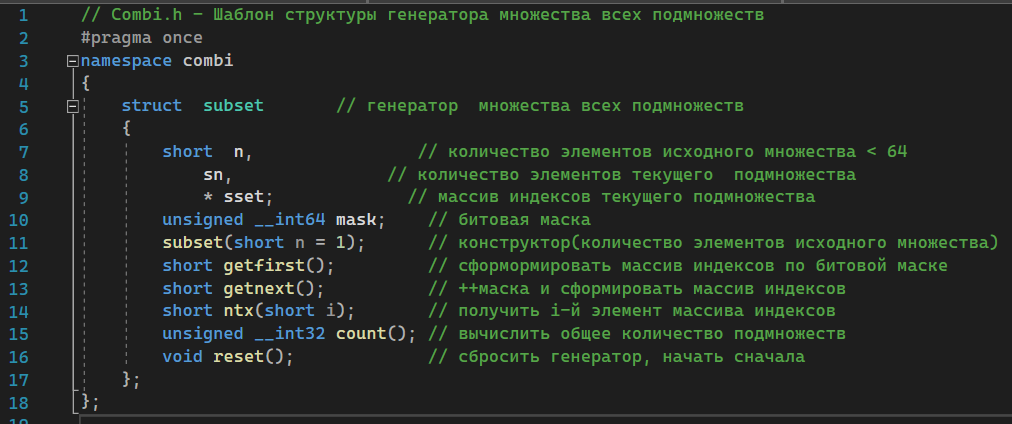
Вывод: В ходе выполнения данной лабораторной работы были приобретены навыки составления и отладки программ с использованием пользовательских функций для замера продолжительности процесса вычисления, а также был изучен и описан алгоритм Фибоначчи.

**Лабораторная работа 2. Комбинаторные алгоритмы решения оптимизационных задач**

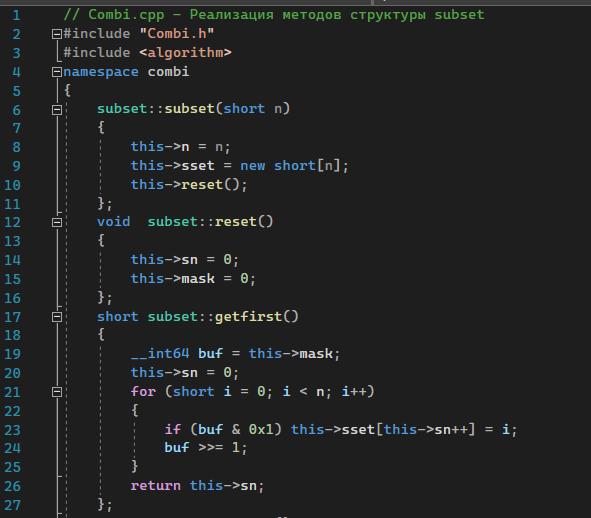
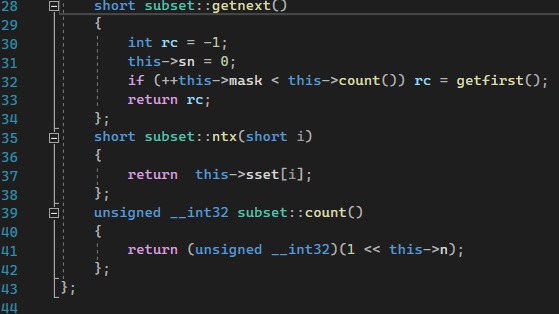
**ЦЕЛЬ РАБОТЫ:** приобрести навыки разработки генераторов подмножеств, перестановок, сочетаний и размещений на С++; научиться применять разработанные генераторы для решения задач о рюкзаке (упрощенную, коммивояжера, об оптимальной загрузке судна и об оптимальной загрузке судна с центровкой.

**Задание 1.** Разобрать и разработать генератор подмножеств заданного множества.

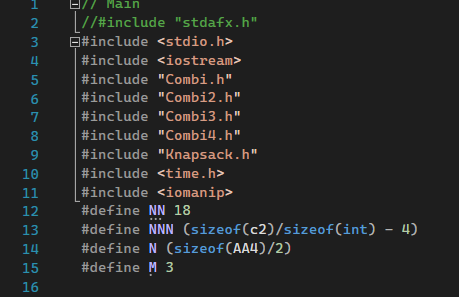
Разобрала и разработала код генератора всех подмножества заданного множества из методического пособия.



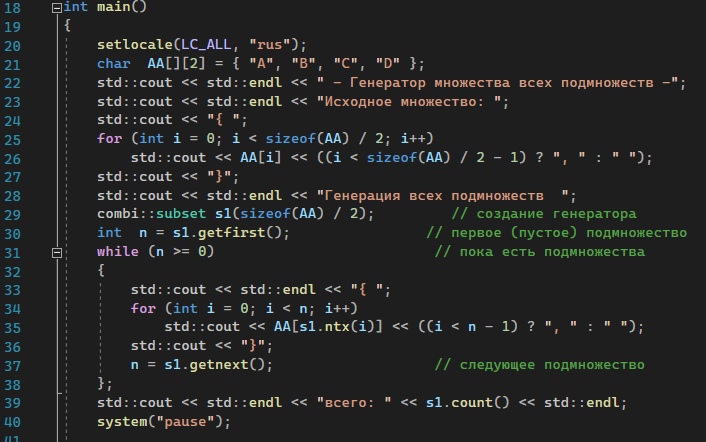
Заголовочный файл для данного генератора

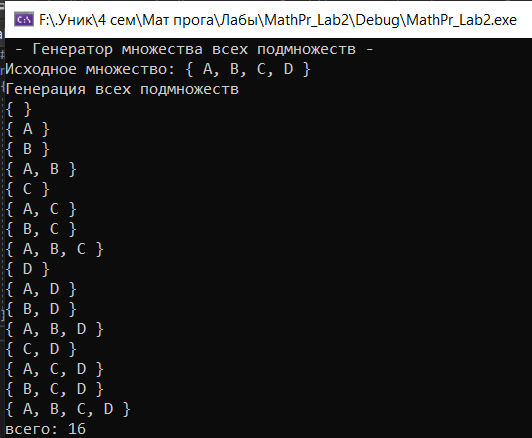
Исполнительный файл для данного генератора



Все директивы, используемые при выполнении лабораторной работы



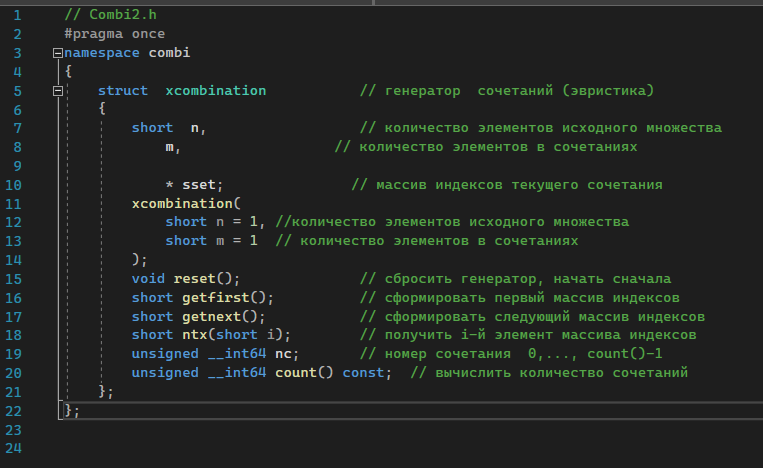
Реализация генератора в функции main.



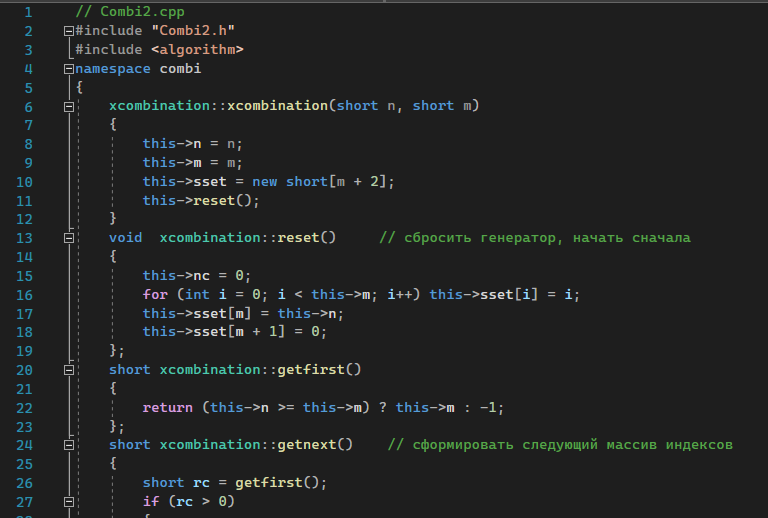
Вывод выполнения генератора всех подмножеств на консоль

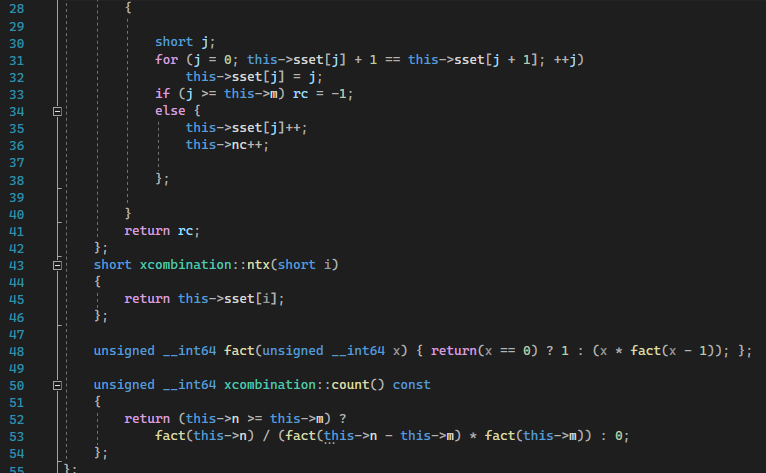
**Задание 2.** Разобрать и разработать генератор сочетаний.

Разобрала и разработала код генератора сочетаний для заданного множества из методического пособия.

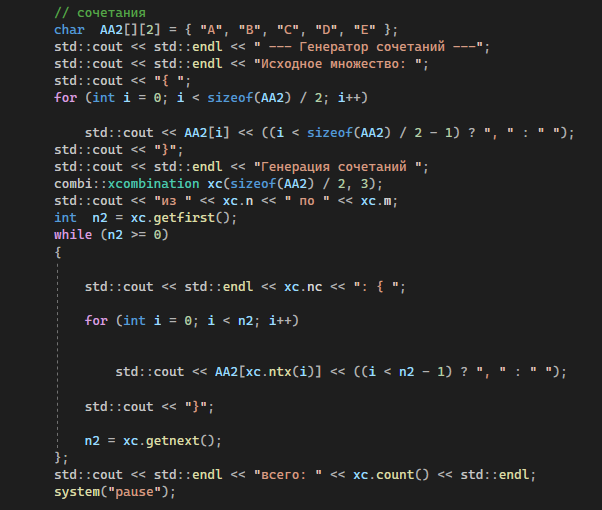


Заголовочный файл для данного генератора

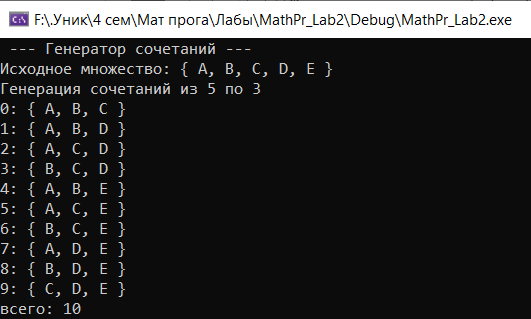




Исполнительный файл для данного генератора



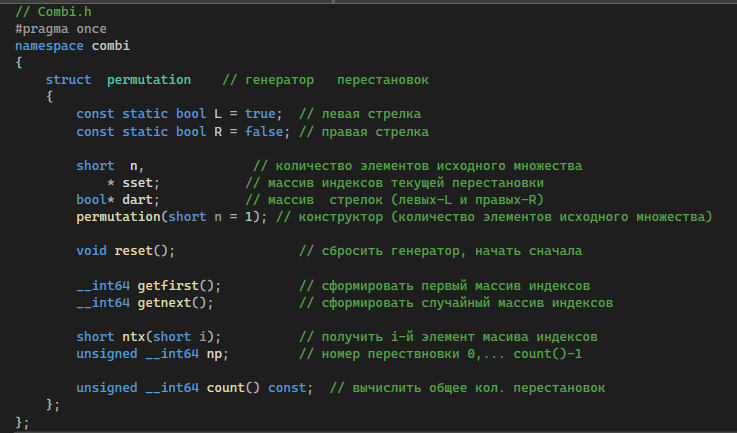
Реализация генератора в функции main.



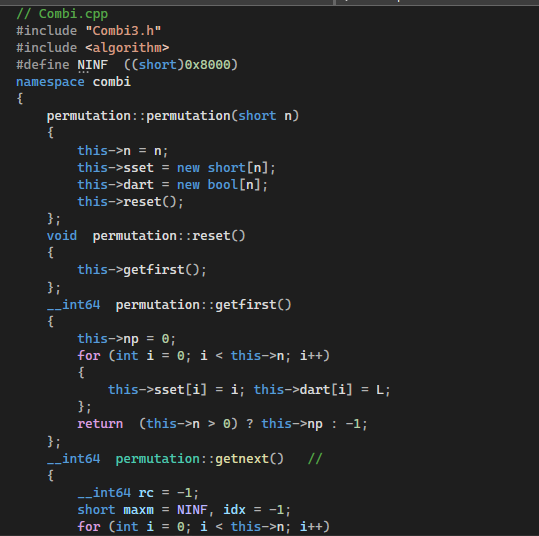
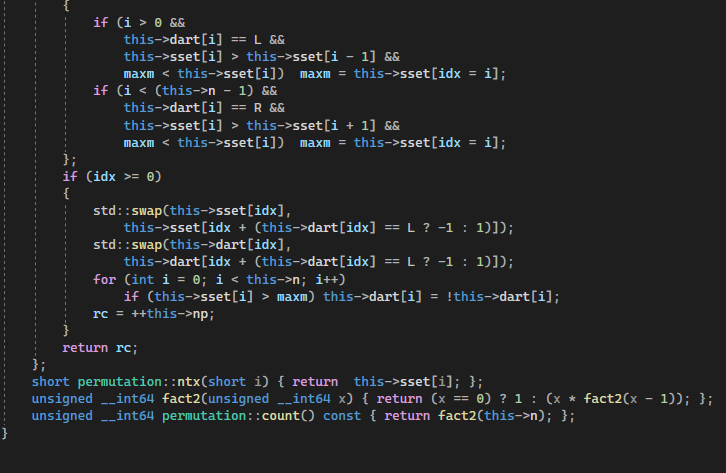
Вывод генератора сочетаний на консоль

**Задание 3.** Разобрать генератор перестановок.

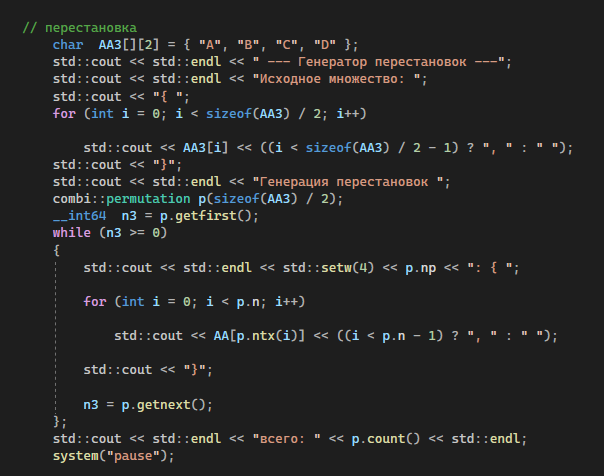
Разобрала код генератора перестановок для заданного множества из методического пособия.



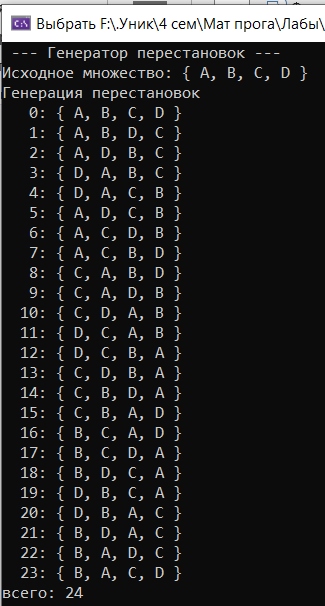
Заголовочный файл для данного генератора

Исполнительный файл для данного генератора



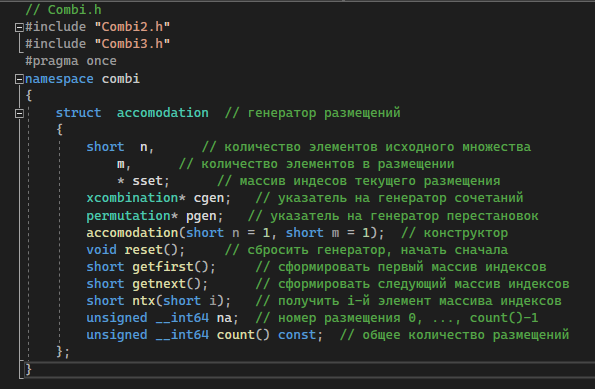
Реализация генератора в функции main.



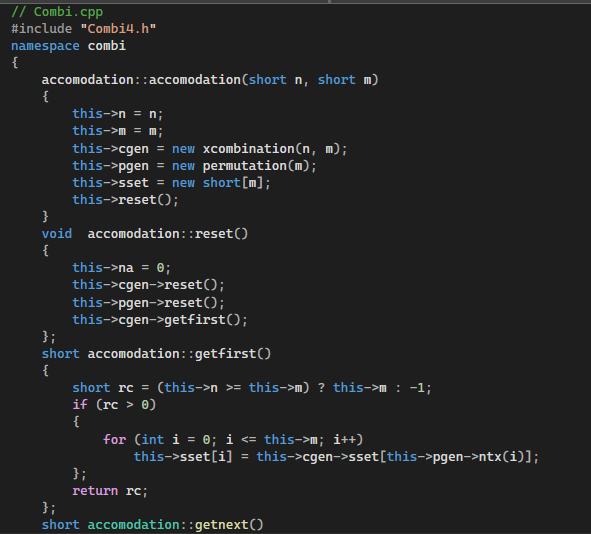
Вывод выполнения генератора перестановок на консоль.

**Задание 4.** Разобрать и доработать генератор размещений.

Разобрала и доработала код генератора всех размещений заданного множества из методического пособия.

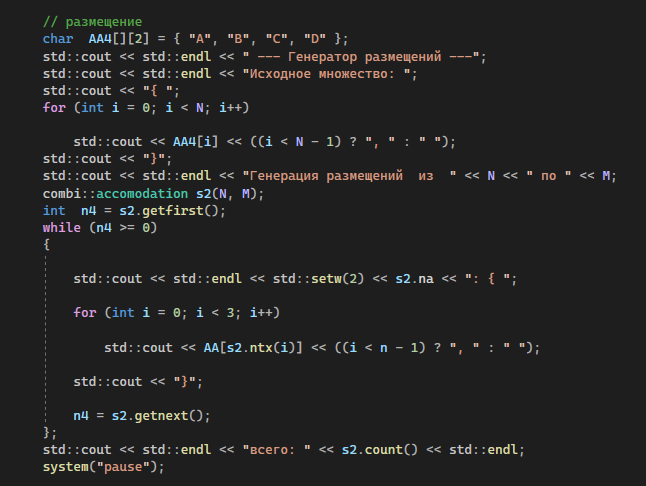


Заголовочный файл для данного генератора

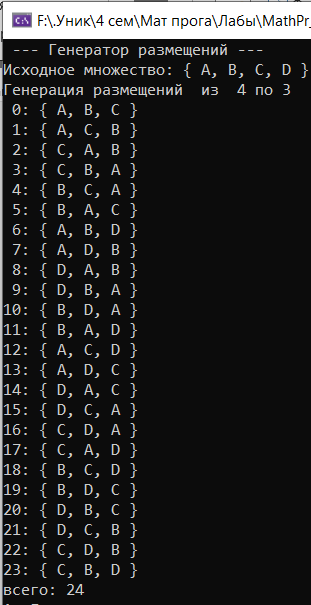




Исполнительный файл для данного генератора



Реализация генератора в функции main.

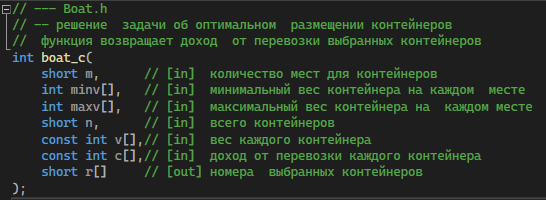


Вывод результата генератора размещений на консоль.

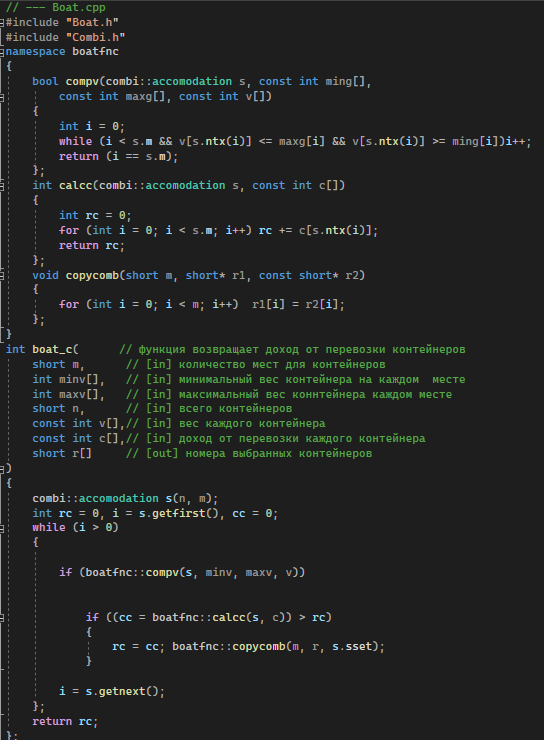
**Задание 5.**  Решить задачу об оптимальной загрузке судна с условием центровки

Для решения данной задачи был использован и доработан код из методического пособия.

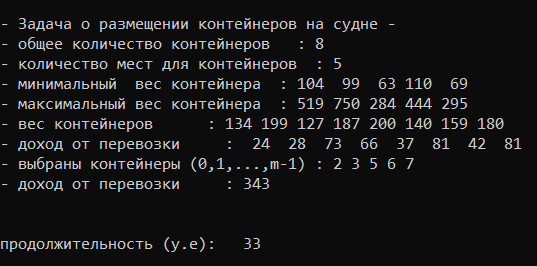
Заголовочный файл для данной задачи:



Исполнительный файл для данной задачи:



Результат выполнения исполнительного файла задачи об оптимальной загрузке судна с условием центровки:



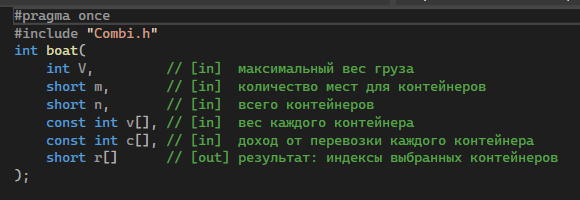
**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи об оптимальной загрузке судна с условием центровки.

Для исследования зависимости времени вычисления задачи об оптимальной загрузке судна с условием центровки был использован и доработан код из методического пособия.

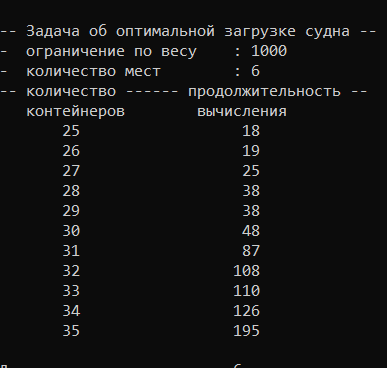
Исполнительный файл для данной задачи:



Заголовочный файл для данного программного блока:



Результат выполнения программного кода задачи об оптимальной загрузке судна с условием центровки:



Был построен график зависимости продолжительности вычислений от количества мест на судне:



Проанализировав данный график было выявлено, что он является экспоненциальным.

Вывод: В ходе выполнения лабораторной работы были приобретены навыки разработки генераторов подмножеств, перестановок, сочетаний и размещений на С++, а также применены разработанные генераторы для решения задач об оптимальной загрузке судна.

**Лабораторная работа 3. Метод ветвей и границ. Задача коммивояжера и методы её решения.**

**ЦЕЛЬ РАБОТЫ:** освоить общие принципы решения задач методом ветвей и границ, решить задачу о коммивояжере данным методом, сравнить полученное решение задачи с комбинаторным методом перестановок.

**Задание 1.** Сформулировать условие задачи коммивояжера с параметром.

Задача коммивояжера с параметром заключается в поиске гамильтонова цикла минимального веса в полном взвешенном графе, заданном матрицей расстояний, при условии, что в этом цикле есть ребро с максимальным весом не больше заданного параметра.

В данном случае матрица расстояний содержит расстояния между 5 городами, заданными номерами от 1 до 5. Значение INF означает отсутствие ребра между соответствующими городами. Задача коммивояжера с параметром для данной матрицы заключается в поиске гамильтонова цикла минимального веса в графе, при условии, что в этом цикле есть ребро с максимальным весом не больше заданного параметра.

Гамильтонов цикл – такой путь, который проходит через все вершины графа ровно один раз.

Параметр в задаче коммивояжера с параметром - это число, которое ограничивает вес ребра, которое может быть включено в найденный гамильтонов цикл.

Более конкретно, параметр определяет максимально допустимый вес ребра в гамильтоновом цикле. Если в графе есть несколько ребер, вес которых больше параметра, то цикл, содержащий любое из этих ребер, не будет оптимальным.

Цель задачи коммивояжера с параметром - найти гамильтонов цикл минимального веса в заданном графе, который удовлетворяет ограничению на максимальный вес ребра. Значение параметра может быть задано заранее или получено в результате вычислений, и влияет на решение задачи.

**Задание 2.** Решить сформулированную задачу методом ветвей и границ.

12

12

14

29

25

α = 12+12+14+29+25=92

Приведённая по строкам матрица:

0 12 15 0 0

β= 0+12+15+0+0=27

Полностью приведённая матрица:

Нижняя граница длины кольцевого маршрута φ = 92 + 27 = 119

01,2 = 3; 01,5 = 7; 02,1 = 0; 02,3 = 4;

03,1 = 10; 04,1 = 3; 05,4 = 56;

Выбираем максимальное значение и получаем путь из города 5 в 4 и соответственно получим граф:

175

119

В случае если мы идём по маршруту (5, 4) то расстояние будет равно нижней границе кольцевого маршрута, то есть 119, а если не пойдём, то расстояние будет равно 119 + 56 = 175.

Так как меньшее расстояние 119, то мы идём из города 5 в город 4. Следовательно, для дальнейших вычислений вычёркиваем 5 строку и 4 столбец из матрицы и делаем обратный путь (4, 5) равным INF:

1 2 3 5

1

2

3

4

Полученная матрица уже является полностью приведённой, поэтому нижняя граница кольцевого маршрута останется неизменной и равной 119.

01,2 = 3, 01,5 = 47, 02,1 = 0, 02,3 = 4,

03,1 = 10, 04,1 = 3,

Выбираем максимальное значение и получаем путь из города 1 в 5 и соответственно получим граф:

119

166

В случае если мы идём по маршруту (1, 5) то расстояние будет равно нижней границе кольцевого маршрута, то есть 119, а если не пойдём, то расстояние будет равно 119 + 47 = 166.

Так как меньшее расстояние 119, то мы идём из города 1 в город 5. Следовательно, для дальнейших вычислений вычёркиваем 1 строку и 5 столбец из матрицы:

1 2 3

2

3

4

Данная матрица не является полностью приведённой, поэтому её надо привести по столбцам и соответственно она примет вид:

1 2 3

2

3

4

β = 0 + 3 + 0 = 3;

Следовательно, изменится нижняя граница кольцевого маршрута и соответственно: φ = 119 + 3 = 122.

02,1 =0, 02,3=4, 03,1 = 0, 04,1 = 0, 04,2 = 7;

Выбираем максимальное значение и получаем путь из города 4 в 2 и соответственно получим граф:

122

129

В случае если мы идём по маршруту (4, 2) то расстояние будет равно нижней границе кольцевого маршрута, то есть 122, а если не пойдём, то расстояние будет равно 122 + 7 = 129.

Так как меньшее расстояние 122, то мы идём из города 4 в город 2. Следовательно, для дальнейших вычислений вычёркиваем 4 строку и 2 столбец из матрицы:

1 3

2

3

После анализа данной матрицы к нашему графу добавятся пути (2, 1), (2, 3) и (3, 1). Соответственно минимальное расстояние будет равно 122, и граф будет иметь следующий вид:

122

119

119

129

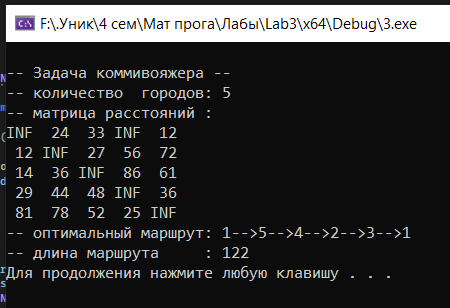
166

175

Расставим переходы между городами в правильной последовательности и соответственно получим (1, 5), (5, 4), (4, 2), (2, 3), (3, 1).

**Задание №3**

Проверка правильности решения:



**Вывод**: Мной были освоены общие принципы решения задач методом ветвей и границ, и была решена задача о коммивояжере данным методом. Полученное решение задачи было сверено с комбинаторным методом перестановок.

**Лабораторная работа 4**

**ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ**

**ЦЕЛЬ РАБОТЫ:** освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.

**Ход выполнения работы**

**Задание 1.**

На языке С++ была сгенерирована случайным образом строка букв латинского алфавита S1 длиной 300 символов и S2 длиной 200.

**Код программы:**#include <iostream>

#include <string>

#include <stdlib.h> /\* srand, rand \*/

#include <ctime>

using namespace std;

string RandomString(int len)

{

srand(time(0));

string str = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";

string newstr;

int pos;

while (newstr.size() != len) {

pos = ((rand() % (str.size() - 1)));

newstr += str.substr(pos, 1);

}

return newstr;

}

int main()

{

cout << "String 300" << endl;

string random\_str = RandomString(300);

cout << random\_str << endl;

cout << "String 200" << endl;

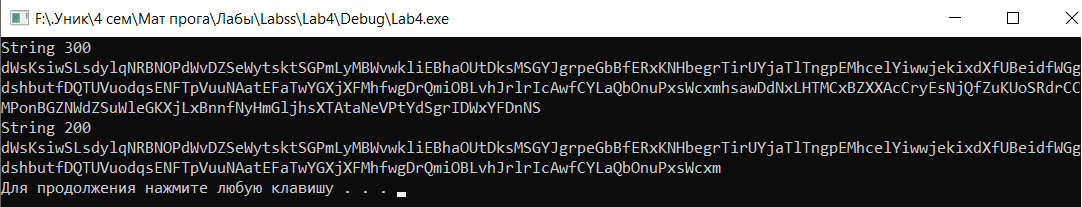
random\_str = RandomString(200);

cout << random\_str << endl;

system("pause");

}}

Результат выполнения программы:



**Задание 2.**

Были вычислены двумя способами (рекурсивно и с помощью динамического программирования) – Левенштейна для , где - длина строки ,  - строка, состоящая из первых  символов строки . (копии экрана и код вставить в отчет).

**Решение:**

Ниже приведены варианты реализации нахождения дистанции Левенштейна при помощи динамического программирования и при помощи рекурсивного алгоритма.

Исходный код реализации через динамическое программирование:

#include "stdafx.h"

#include <iostream>

#include <algorithm>

#include <ctime>

#include <iomanip>

#include <locale>

#include "Levenshtein.h"

using namespace std;

#define \_rand(min, max) ( rand() % ((max) - (min) + 1) + (min) )

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

char abc[25]; // алфавит

char s1[300];

char s2[200];

// заполняем массив

for (int i = 97, n = 0; i <= 122; ++i, ++n)

{

abc[n] = (char)i;

}

cout << "S1 = ";

for (int i = 0; i < 301; i++)

{

s1[i] = abc[\_rand(0, 25)];

if (i % 50 == 0)

cout << "\n";

cout << s1[i];

}

cout <<"\nS2 =";

for (int i = 0; i < 201; i++)

{

s2[i] = abc[\_rand(0, 25)];

if (i% 50 == 0)

cout << "\n";

cout << s2[i];

}

cout << "\n";

clock\_t t1 = 0, t2 = 0, t3 = 0, t4 = 0;

int lx = sizeof(s1) ;

int ly = sizeof(s2) ;

int S1\_size[]{ 12,15,20,30,60,150,300 };

int S2\_size[]{ 8,10,14,20,40,100,200 };

std::cout << "\n\n-- расстояние Левенштейна -----";

std::cout << "\n\n--длина --- рекурсия -- дин.програм. ---\n";

for (int i = 0; i < min(lx, ly); i++)

{

t1 = clock();

levenshtein\_r(S1\_size[i], s1 , S2\_size[i], s2);

t2 = clock();

t3 = clock();

levenshtein(S1\_size[i], s1, S2\_size[i], s2);

t4 = clock();

cout << right << setw(2) << S1\_size[i] << "/" << setw(2) << S2\_size[i]

<< " " << left << setw(10) << (t2 - t1)

<< " " << setw(10) << (t4 - t3) << endl;

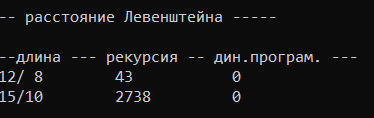
}

system("pause");

return 0;

}

Результат вычисления расстояния Левенштейна вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.



**Задание 3.** Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . (копии экрана и график вставить в отчет).

**Решение:**

На графике, представленном на рисунке 3, можно заметить, что выполненные с помощью динамического алгоритма, вычисления производятся в разы быстрее, чем с помощью рекурсивного алгоритма.



Рис. 3 – сравнительный анализ времени выполнения

**Задание 4.** Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом) (каждый шаг алгоритма по примеру из лекции вставить в отчет).

|  |  |
| --- | --- |
| Задание 4 | |
| Эхо | Хорек |

**Решение:**

1.  
2.  
3.  
4.  
5.  

 = 5.

 = 4.

1.  

 = 4.

 = 3.

1.  
2.  
3.  

 = 3.

 = 2.

1.  
2.  
3.  

 = 2.

 = 1.

1.  

 = 3.

 = 2.

1.  

 = 2.

 = 1.

 = 1.

 = 1.

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 

**Задание 5.**

Выполнила сравнительный анализ времени затраченного на решение задачи об оптимальной расстановке скобок при умножении нескольких матриц для двух методов решения (рекурсивное решение, динамическое программирование). Размерность матриц взята в соответствии с вариантом.

|  |
| --- |
| Задание 5 |
| 8\*11, 11\*19, 19\*22, 22\*29, 29\*39, 39\*50 |

**Решение:**

Программный код:

// расстановка скобок (рекурсия)

#define INFINITY 0x7fffffff

#define NINFINITY 0x80000000

int OptimalM(int i, int j, int n, const int c[], int \*s){

#define OPTIMALM\_S(x1,x2) (s[(x1-1)\*n+x2-1])

int o = INFINITY;

int bo = INFINITY;

if (i<j){

for (int k = i; k<j; k++){

bo = OptimalM(i, k, n, c, s) + OptimalM(k + 1, j, n, c, s) + c[i - 1] \* c[k] \* c[j];

if (bo < o){

o = bo;

OPTIMALM\_S(i, j) = k;

}

}

}

else o = 0;

return o;

#undef OPTIMALM\_S

};

// расстановка скобок (динамическое программирование)

int OptimalMD(int n, const int c[], int\* s){

#define OPTIMALM\_S(x1,x2) (s[(x1-1)\*n+x2-1])

#define OPTIMALM\_M(x1,x2) (M[(x1-1)\*n+x2-1])

int \*M = new int[n\*n], j = 0, q = 0;

for (int i = 1; i <= n; i++)

OPTIMALM\_M(i, i) = 0;

for (int l = 2; l <= n; l++){

for (int i = 1; i <= n - l + 1; i++){

j = i + l - 1;

OPTIMALM\_M(i, j) = INFINITY;

for (int k = i; k <= j - 1; k++){

q = OPTIMALM\_M(i, k)+OPTIMALM\_M(k + 1, j)+c[i - 1]\*c[k]\* c[j];

if (q < OPTIMALM\_M(i, j)){

OPTIMALM\_M(i, j) = q;

OPTIMALM\_S(i, j) = k;

}

}

}

}

return OPTIMALM\_M(1, n);

#undef OPTIMALM\_M

#undef OPTIMALM\_S

};

Результат выполнения программного кодв:

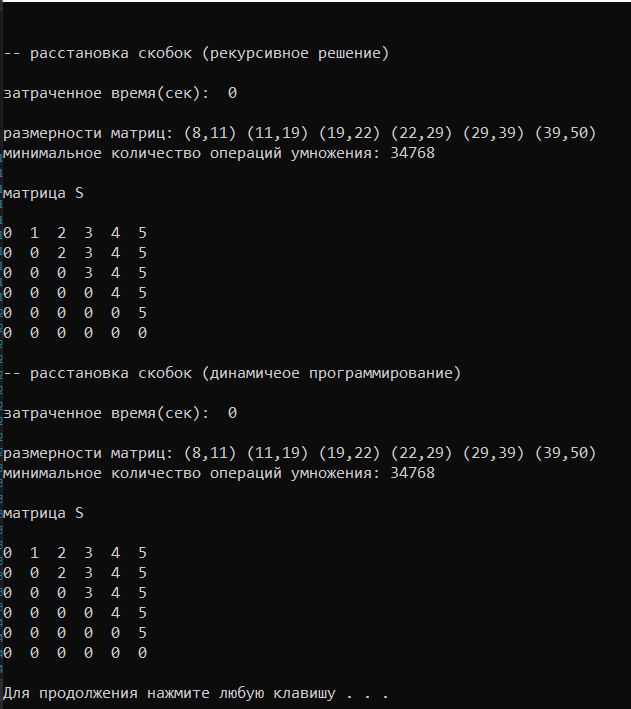


Рис. 4 – результат выполнения программы

Принцип расстановки скобок по итоговой матрице:

Скобки расставляются по принципу «сначала внешние – затем внутренние». Имеется 6 матриц, вот их размерность:

А1=8\*11,

А2=11\*19,

А3=19\*22,

А 4 =22\*29,

А 5 =29\*39,

А 6 =39\*50.

Матрица S:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** |
| **1** | 0 | 1 | 2 | 3 | 4 | 5 |
| **2** | 0 | 0 | 2 | 3 | 4 | 5 |
| **3** | 0 | 0 | 0 | 3 | 4 | 5 |
| **4** | 0 | 0 | 0 | 0 | 4 | 5 |
| **5** | 0 | 0 | 0 | 0 | 0 | 5 |
| **6** | 0 | 0 | 0 | 0 | 0 | 0 |

Найдем элемент (1,6) в матрице S, он равен 5. Это означает, что точка разрыва между 1-ой и 6-ой матрицей находится после 5-ой матрицы. Что позволяет расставить скобки следующим образом:

(A1\*A2\*A3\*A4\*A5)\*A6

Точку разрыва между второй и шестой матрицей определяет элемент (1,5). Он равен 4. Следовательно разрыв будет после четвертой матрицы.

((A1\*A2\*A3\*A4)\*A5)\*A6

Далее берем элемент (1,4) и получаем, что он равен 3. Следовательно получаем:

(((A1\*A2\*A3)\*A4)\*A5)\*A6

И на последнем шаге мы возьмем элемент (1,3) и он равен 2:

((((A1\*A2)\*A3)\*A4)\*A5)\*A6

И на последнем шаге мы возьмем элемент (1,2) и он равен 1:

((((A1\*A2)\*A3)\*A4)\*A5)\*A6

Это выражение и есть конечное.

Полученная расстановка скобок позволяет получить минимальное количество операций умножения, равное 34768.

Вывод: в ходе выполнения лабораторной работы были освоены общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом, а также была вычислена дистанции Левенштейна при помощи рекурсивного алгоритма.

**Лабораторная работа 5. ТРАНСПОРТНАЯ ЗАДАЧА**

**Цель работы:** Приобретение навыков решения открытой транспортной задачи

**Задание.** Решить транспортную задачу. Имеется 5 поставщиков продукции и 6 потребителей. Величина запасов, потребностей и стоимость затрат на перевозку продукции взять в соответствии с вариантом (*N*). Оформить отчет.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **N+12** | **N+2** | **N+6** | **N+3** | **N+11** | **N+1** | **168+N** |
| 2 | **N+10** | **N** | **N+8** | **N+5** | **N+7** | **N+13** | **113+N** |
| 3 | **N+1** | **N+5** | **N+11** | **N+8** | **N+2** | **N+11** | **150+N** |
| 4 | **N+4** | **N+10** | **N+10** | **N+3** | **N+13** | **N+2** | **159+N** |
| 5 | **N+3** | **N+11** | **N+9** | **N** | **N+10** | **N+4** | **100+N** |
| ПОТРЕБНОСТИ | **143+N** | **107+N** | **131+N** | **193+N** | **95+N** | **163+N** |  |

Условие задачи

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 180 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 125 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 162 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 171 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 112 |
| Потребность | 155 | 119 | 143 | 205 | 107 | 175 |  |

Для разрешимости транспортной задачи необходимо, чтобы суммарные запасы продукции у поставщиков равнялись суммарной потребности потребителей. Проверим это условие.

∑a = 180 + 125 + 162 + 171 + 112 = 750

∑b = 155 + 119 + 143 + 205 + 107 + 175 = 904

Задача является открытой (с нарушенным балансом).

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 180 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 125 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 162 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 171 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 112 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154 |
| Потребность | 155 | 119 | 143 | 205 | 107 | 175 |  |

Так как запасы поставщиков меньше потребности потребителей, введем фиктивного поставщика 6, с запасом продукции равным 904-750=154. Стоимость доставки единицы продукции от фиктивного поставщика ко всем потребителям примем равной нулю.

Теперь выполняется условие.

Общая транспортная задача имеет 12 ограничений в виде равенств, по одному на каждый пункт отправления и назначения. Т. к. транспортная задача д.б. сбалансированной, то одно из этих равенств избыточно. Т.о. транспортная задача имеет 13 независимых ограничений, отсюда вытекает, что начальное базисное решение состоит из 13 базисных переменных.

**Этап I**

***Метод наименьшей стоимости***

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 180 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 125-119=6 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 162 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 171 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 112 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154 |
| Потребность | 155 | 119-119=0 | 143 | 205 | 107 | 175 |  |

Суть метода заключается в том, что из всей таблицы стоимостей выбирают клетку с наименьшей стоимостью, для этой ячейки присваиваем меньшее из чисел ai, или bj. Затем, из рассмотрения исключают либо строку, соответствующую поставщику, запасы которого полностью израсходованы, либо столбец, соответствующий потребителю, потребности которого полностью удовлетворены, либо строку и столбец, если израсходованы запасы поставщика и удовлетворены потребности потребителя. Повторяем, пока все запасы не будут распределены, а потребности удовлетворены.

Искомый элемент равен c22=12.

x22 = min(125,119) = 119.

Искомый элемент равен c54=12. Для этого элемента запасы равны 112, потребности 205. Поскольку минимальным является 112, то вычитаем его.  
x54 = min(112,205) = 112.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 180 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 6 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 162 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 171 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 112-112=0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154 |
| Потребность | 155 | 0 | 143 | 205-112=93 | 107 | 175 |  |

Искомый элемент равен c16=13. Для этого элемента запасы равны 180, потребности 175.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 180-175=5 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 6 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 162 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 171 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154 |
| Потребность | 155 | 0 | 143 | 93 | 107 | 175-175=0 |  |

Поскольку минимальным является 175, то вычитаем его.  
x16 = min(180,175) = 175.

Искомый элемент равен c31=13. Для этого элемента запасы равны 162, потребности 155. Поскольку минимальным является 155, то вычитаем его.  
x31 = min(162,155) = 155.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 5 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 6 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 162-155=7 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 171 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154 |
| Потребность | 155-155=0 | 0 | 143 | 93 | 107 | 0 |  |

Искомый элемент равен c35=14. Для этого элемента запасы равны 7, потребности 107. Поскольку минимальным является 7, то вычитаем его.  
x35 = min(7,107) = 7.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 5 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 6 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 7-7=0 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 171 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154 |
| Потребность | 0 | 0 | 143 | 93 | 107-7=100 | 0 |  |

Искомый элемент равен c14=15. Для этого элемента запасы равны 5, потребности 93. Поскольку минимальным является 5, то вычитаем его.  
x14 = min(5,93) = 5.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
|  | 1 | 2 | 3 | 4 | 5 | 6 |  |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 5-5=0 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 6 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 0 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 171 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154 |
| Потребность | 0 | 0 | 143 | 93-5=88 | 100 | 0 |  |

Искомый элемент равен c44=15. Для этого элемента запасы равны 171, потребности 88. Поскольку минимальным является 88, то вычитаем его.  
x44 = min(171,88) = 88.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
|  | 1 | 2 | 3 | 4 | 5 | 6 |  |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 0 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 6 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 0 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 171-88=83 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154 |
| Потребность | 0 | 0 | 143 | 88-88=0 | 100 | 0 |  |

Искомый элемент равен c25=19. Для этого элемента запасы равны 6, потребности 100. Поскольку минимальным является 6, то вычитаем его.  
x25 = min(6,100) = 6.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 0 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 6-6=0 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 0 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 83 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154 |
| Потребность | 0 | 0 | 143 | 0 | 100-6=94 | 0 |  |

Искомый элемент равен c43=22. Для этого элемента запасы равны 83, потребности 143. Поскольку минимальным является 83, то вычитаем его.  
x43 = min(83,143) = 83.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 0 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 0 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 0 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 83-83=0 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154 |
| Потребность | 0 | 0 | 143-83=60 | 0 | 94 | 0 |  |

Искомый элемент равен c63=0. Для этого элемента запасы равны 154, потребности 60. Поскольку минимальным является 60, то вычитаем его.  
x63 = min(154,60) = 60.

Искомый элемент равен c65=0. Для этого элемента запасы равны 94, потребности 94. Поскольку минимальным является 94, то вычитаем его.  
x65 = min(94,94) = 94.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 0 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 0 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 0 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 0 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 154-60=94 |
| Потребность | 0 | 0 | 60-60=0 | 0 | 94 | 0 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15 | 23 | 13 | 0 |
| 2 | 22 | 12 | 20 | 17 | 19 | 25 | 0 |
| 3 | 13 | 17 | 23 | 20 | 14 | 23 | 0 |
| 4 | 16 | 22 | 22 | 15 | 25 | 14 | 0 |
| 5 | 15 | 23 | 21 | 12 | 22 | 16 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 94-94=0 |
| Потребность | 0 | 0 | 0 | 0 | 94-94=0 | 0 |  |

В результате получен первый опорный план, который является допустимым, так как все грузы из баз вывезены, потребность потребителей удовлетворена, а план соответствует системе ограничений транспортной задачи.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15|5 | 23 | 13|175 | 180 |
| 2 | 22 | 12|199 | 20 | 17 | 19|6 | 25 | 125 |
| 3 | 13|155 | 17 | 23 | 20 | 14|7 | 23 | 162 |
| 4 | 16 | 22 | 22|83 | 15|88 | 25 | 14 | 171 |
| 5 | 15 | 23 | 21 | 12|112 | 22 | 16 | 112 |
| 6 | 0 | 0 | 0|60 | 0 | 0|94 | 0 | 154 |
| Потребность | 155 | 119 | 143 | 205 | 107 | 175 |  |

Подсчитаем число занятых клеток таблицы(Количество базисных переменных), их 11, а должно быть m + n - 1 = 11. Следовательно, опорный план является невырожденным.

Значение целевой функции для этого опорного плана равно:

F(x) = 15\*5 + 13\*175 + 12\*119 + 19\*6 + 13\*155 + 14\*7 + 22\*83 + 15\*88 + 12\*112 + 0\*60 + 0\*94 = 10495

**Этап II**

***Метод потенциалов***

Каждому поставщику ai ставим в соответствие некоторое число - ui, называемое *потенциалом поставщика*. Каждому потребителю bj ставим в соответствие некоторое число - vj, называемое *потенциалом потребителя*. Для базисной ячейки (задействованного маршрута), сумма потенциалов поставщика и потребителя должна быть равна тарифу данного маршрута.

ui + vj = cij

Найдем предварительные потенциалы ui, vj. по занятым клеткам таблицы, полагая, что u1 = 0.

u1 + v4 = 15; 0 + v4 = 15; v4 = 15  
u4 + v4 = 15; 15 + u4 = 15; u4 = 0  
u4 + v3 = 22; 0 + v3 = 22; v3 = 22  
u6 + v3 = 0; 22 + u6 = 0; u6 = -22  
u6 + v5 = 0; -22 + v5 = 0; v5 = 22  
u2 + v5 = 19; 22 + u2 = 19; u2 = -3  
u2 + v2 = 12; -3 + v2 = 12; v2 = 15  
u3 + v5 = 14; 22 + u3 = 14; u3 = -8  
u3 + v1 = 13; -8 + v1 = 13; v1 = 21  
u5 + v4 = 12; 15 + u5 = 12; u5 = -3  
u1 + v6 = 13; 0 + v6 = 13; v6 = 13

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15|5 | 23 | 13|175 | u1=0 |
| 2 | 22 | 12|199 | 20 | 17 | 19|6 | 25 | u2=-3 |
| 3 | 13|155 | 17 | 23 | 20 | 14|7 | 23 | u3=-8 |
| 4 | 16 | 22 | 22|83 | 15|88 | 25 | 14 | u4=0 |
| 5 | 15 | 23 | 21 | 12|112 | 22 | 16 | u5=-3 |
| 6 | 0 | 0 | 0|60 | 0 | 0|94 | 0 | u6=-22 |
| Потребность | v1 = 21 | v2 = 15 | v3 = 22 | v4 =15 | v5 =22 | v6 = 13 |  |
|  |  |  |  |  |  |  |  |

Опорный план не является оптимальным, так как существуют оценки свободных клеток, для которых ui + vj > cij  
(1;2): 0 + 15 > 14; ∆12 = 0 + 15 - 14 = 1 > 0  
(1;3): 0 + 22 > 18; ∆13 = 0 + 22 - 18 = 4 > 0  
(4;1): 0 + 21 > 16; ∆41 = 0 + 21 - 16 = 5 > 0  
(5;1): -3 + 21 > 15; ∆51 = -3 + 21 - 15 = 3 > 0  
max(1,4,5,3) = 5

Выбираем максимальную оценку свободной клетки (4;1): 16  
Для этого в перспективную клетку (4;1) поставим знак «+», а в остальных вершинах многоугольника чередующиеся знаки «-», «+», «-».

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15|5 | 23 | 13|175 | u1=0 |
| 2 | 22 | 12|199 | 20 | 17 | 19|6 | 25 | u2=-3 |
| 3 | 13|155| - | 17 | 23 | 20 | 14|7|+ | 23 | u3=-8 |
| 4 | 16| + | 22 | 22|83|- | 15|88 | 25 | 14 | u4=0 |
| 5 | 15 | 23 | 21 | 12|112 | 22 | 16 | u5=-3 |
| 6 | 0 | 0 | 0|60|+ | 0 | 0|94|- | 0 | u6=-22 |
| Потребность | v1 = 21 | v2 = 15 | v3 = 22 | v4 =15 | v5 =22 | v6 = 13 |  |
|  |  |  |  |  |  |  |  |

Цикл приведен в таблице (4,1 → 4,3 → 6,3 → 6,5 → 3,5 → 3,1).  
Из грузов хij стоящих в минусовых клетках, выбираем наименьшее, т.е. у = min (4, 3) = 83. Прибавляем 83 к объемам грузов, стоящих в плюсовых клетках и вычитаем 83 из Хij, стоящих в минусовых клетках. В результате получим новый опорный план.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15|5 | 23 | 13|175 | 180 |
| 2 | 22 | 12|199 | 20 | 17 | 19|6 | 25 | 125 |
| 3 | 13|72 | 17 | 23 | 20 | 14|90 | 23 | 162 |
| 4 | 16| 83 | 22 | 22 | 15|88 | 25 | 14 | 171 |
| 5 | 15 | 23 | 21 | 12|112 | 22 | 16 | 112 |
| 6 | 0 | 0 | 0|143 | 0 | 0|11 | 0 | 154 |
| Потребность | 155 | 119 | 143 | 205 | 107 | 175 |  |
|  |  |  |  |  |  |  |  |

Проверим оптимальность опорного плана. Найдем *предварительные потенциалы* ui, vj. по занятым клеткам таблицы, в которых ui + vj = cij, полагая, что u1 = 0.  
u1 + v4 = 15; 0 + v4 = 15; v4 = 15  
u4 + v4 = 15; 15 + u4 = 15; u4 = 0  
u4 + v1 = 16; 0 + v1 = 16; v1 = 16  
u3 + v1 = 13; 16 + u3 = 13; u3 = -3  
u3 + v5 = 14; -3 + v5 = 14; v5 = 17  
u2 + v5 = 19; 17 + u2 = 19; u2 = 2  
u2 + v2 = 12; 2 + v2 = 12; v2 = 10  
u6 + v5 = 0; 17 + u6 = 0; u6 = -17  
u6 + v3 = 0; -17 + v3 = 0; v3 = 17  
u5 + v4 = 12; 15 + u5 = 12; u5 = -3  
u1 + v6 = 13; 0 + v6 = 13; v6 = 13

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 24 | 14 | 18 | 15|5 | 23 | 13|175 | u1=0 |
| 2 | 22 | 12|199 | 20 | 17 | 19|6 | 25 | u2=2 |
| 3 | 13|72 | 17 | 23 | 20 | 14|90 | 23 | u3=-3 |
| 4 | 16| 83 | 22 | 22 | 15|88 | 25 | 14 | u4=0 |
| 5 | 15 | 23 | 21 | 12|112 | 22 | 16 | u5=-3 |
| 6 | 0 | 0 | 0|143 | 0 | 0|11 | 0 | u6=-17 |
| Потребность | v1 = 16 | v2 = 10 | v3 = 17 | v4 =15 | v5 =17 | v6 = 13 |  |
|  |  |  |  |  |  |  |  |

Опорный план является оптимальным, так все оценки свободных клеток удовлетворяют условию ui + vj ≤ cij.  
Минимальные затраты составят:

F(x) = 15\*5 + 13\*175 + 12\*119 + 19\*6 + 13\*72 + 14\*90 + 16\*83 + 15\*88 + 12\*112 + 0\*143 + 0\*11 = 10080  
**Анализ оптимального плана**.  
Из 1-го склада необходимо груз направить в 4-й магазин (5 ед.), в 6-й магазин (175 ед.)  
Из 2-го склада необходимо груз направить в 2-й магазин (119 ед.), в 5-й магазин (6 ед.)  
Из 3-го склада необходимо груз направить в 1-й магазин (72 ед.), в 5-й магазин (90 ед.)  
Из 4-го склада необходимо груз направить в 1-й магазин (83 ед.), в 4-й магазин (88 ед.)  
Из 5-го склада необходимо весь груз направить в 4-й магазин.  
Потребность 3-го магазина остается неудовлетворенной на 143 ед.  
Оптимальный план является вырожденным, так как базисная переменная x63=0.  
Потребность 5-го магазина остается неудовлетворенной на 11 ед.  
Оптимальный план является вырожденным, так как базисная переменная x65=0.

**Лабораторная работа 6. АЛГОРИТМЫ НА ГРАФАХ**

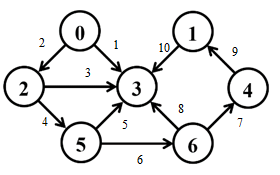
**ЦЕЛЬ РАБОТЫ:** Освоить сущность и программную реализацию: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов. Разобрать алгоритм Прима и алгоритм Крускала.

***Задание 1.*** Ориентированный граф **G** взять в соответствии с вариантом. Представить его в отчете в виде матрицы смежности, матрицы инцидентности, списка смежных вершин.



Матрица смежности

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| 6 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |



Матрица идентичности

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 1 |
| 2 | 0 | -1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | -1 | 0 | -1 | 0 | -1 | 0 | 0 | -1 | 0 | -1 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | -1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | -1 | 1 | 1 | 0 | 0 |

Cписок смежных вершин



S0={2,3}, S1={3}, S2={3,5}, S3={⌀}, S4={1}, S5={3,6}, S6={3,4};

***Задание 2.*** Осуществить алгоритмы поиска в ширину и глубину, а также алгоритма топологической сортировки аналогично примерам, рассмотренным на лекциях.

**Алгоритм поиска в ширину(BFS)**

Исходный граф: (0,2), (0,3), (1,3), (2,3), (2,5), (4,1), (5,3), (5,6), (6,4)

По условию, граф имеет 7 вершин, пронумерованных начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

Текущее состояние алгоритма хранится в следующих структурах памяти:

Q – очередь вершин,

C – массив окраски вершин,

D – массив расстояний,

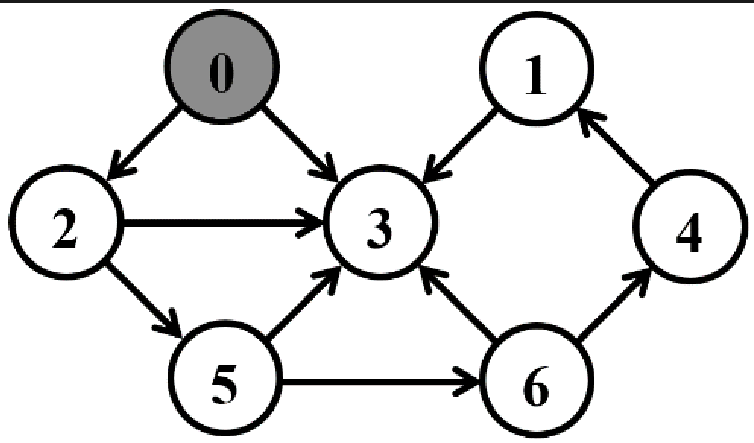
P – массив предшествующих вершин

W-White G-Gray B-Black N-Nil

**Шаг 1.**

В качестве стартовой вершины выбираем вершину с номером 0.

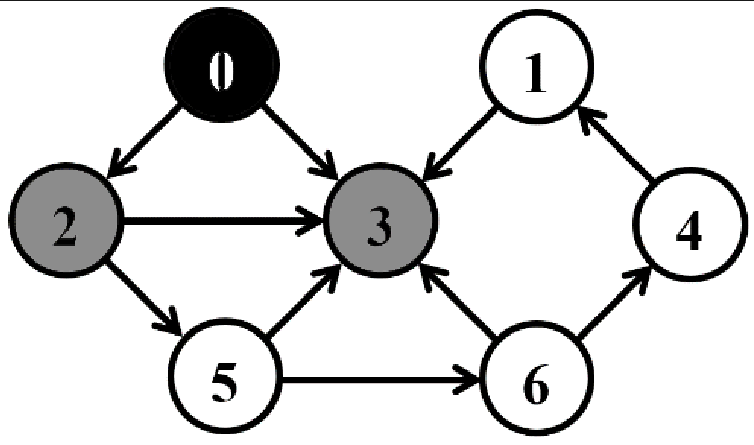
|  |  |
| --- | --- |
| Q | 0 |
| C | G | W | W | W | W | W | W |
| D | 0 | I | I | I | I | I | I |
| P | N | N | N | N | N | N | N |

в

**Шаг 2.**

У вершины 0 две смежные вершины. Для последующего пути выбираем вершину с наименьшим весом – 2, а 3-ю добавляем в начало очереди. Закрашиваем 0 в черный цвет как пройденную. Смежные в серый.

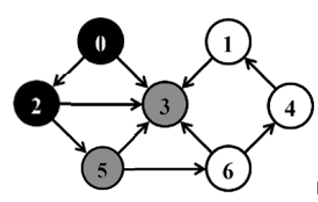
|  |  |  |
| --- | --- | --- |
| Q | 2 | 3 |
| C | B | W | | G | G | W | W | W |
| D | 0 | I | | 1 | 1 | I | I | I |
| P | N | N | | 0 | 0 | N | N | N |



**Шаг 3.**

Переходим в смежную вершину, вершину номер 2. С ней смежные вершины номер 3 и 5. Закрашиваем 3 и 5 в серый цвет, а вершину номер 2 в черный, как пройденную

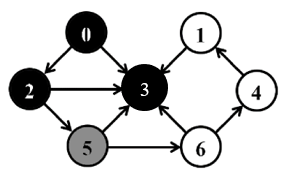
|  |  |  |
| --- | --- | --- |
| Q | 3 | 5 |
| C | B | W | | B | G | W | G | W |
| D | 0 | I | | 1 | 1 | I | 2 | I |
| P | N | N | | 0 | 0 | 1 | N | N |



**Шаг 4.**

Переходим в смежную вершину, вершину номер 3. Вершина номер 3 не имеет смежных вершин поэтому просто окрашиваем ее в черный, как пройденную.

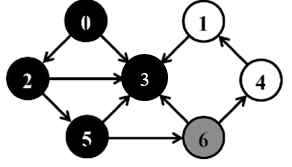
|  |  |
| --- | --- |
| Q | 5 |
| C | B | W | B | B | W | G | W |
| D | 0 | I | 1 | 1 | I | 2 | I |
| P | N | N | 0 | 0 | 1 | 3 | N |



**Шаг 5.**

Переходим в смежную вершину, вершину номер номер 5. Вершина номер 5 имеет одну смежную вершину, номер 6, закраиваем ее в серый цвет, а вершину 5 в черный.

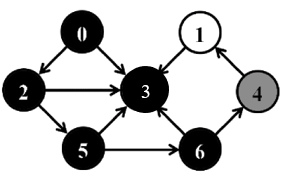
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 6 |  |  |  |  |  |  |
| C | B | W | B | B | W | B | G |
| D | 0 | I | 1 | 1 | I | 2 | 3 |
| P | N | N | 0 | 0 | 1 | 3 | 4 |



**Шаг 6.**

Переходим смежную вершину 6, закрашиваем ее черным цветом, как пройденную, а смежную вершину 4 в серый как смежную

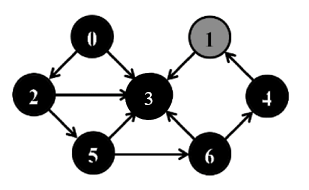
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 4 |  |  |  |  |  |  |
| C | B | W | B | B | G | B | B |
| D | 0 | I | 1 | 1 | 4 | 2 | 3 |
| P | N | N | 0 | 0 | 5 | 3 | 4 |



**Шаг 7.**

Закрашиваем вершину номер 4 в черный, как пройденную, а вершину номер 1 в серый как смежную

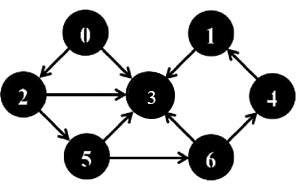
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q | 1 |  |  |  |  |  |  |
| C | B | G | B | B | B | B | B |
| D | 0 | 5 | 1 | 1 | 4 | 2 | 3 |
| P | 0 | 6 | 0 | 0 | 5 | 3 | 4 |



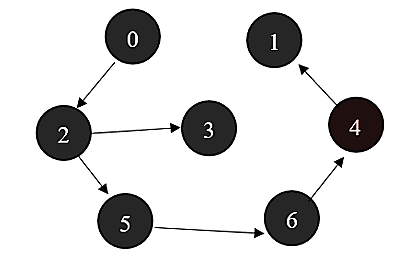
**Шаг 8.**

Закрашиваем вершину номер 1 в черный цвет, как пройденную. Больше вершин не осталось

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Q |  |  |  |  |  |  |  |
| C | B | B | B | B | B | B | B |
| D | 0 | 5 | 1 | 1 | 4 | 2 | 3 |
| P | N | 6 | 0 | 0 | 5 | 3 | 4 |



В результате получили BFS-дерево:



Порядок обхода:

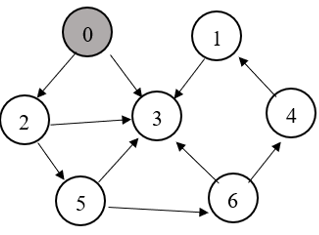
0, 2, 3, 5, 6, 4, 1.

**Алгоритм поиска в глубину (DFS).**

**Шаг 1.**

По условию граф имеет 7 вершин, пронумерованных с нуля. В качестве стартовой вершины возьмем вершину с номером 0.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | W | W | W | W | W |
| D | 1 | I | I | I | I | I | I |
| P | N | N | N | N | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

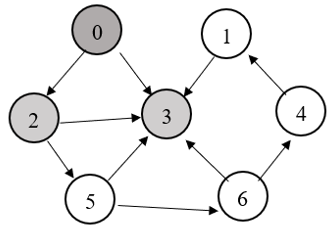


**Шаг 2.**

Вершина 0 имеет 2 смежные вершины 2 и 3. Закрашиваем их серым цветом.

И выбираем те что с наименьшим весом.

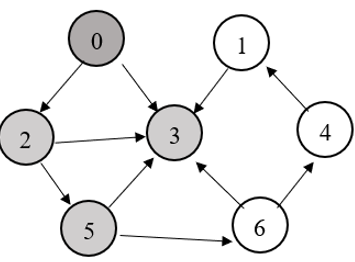
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | G | G | W | W | W |
| D | 1 | I | 2 | 2 | I | I | I |
| P | N | N | 0 | 0 | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



**Шаг 3.**

Вершина 2 имеет одну не отмеченную смежную вершину номер 5, закрашиваем ее в серый цвет.

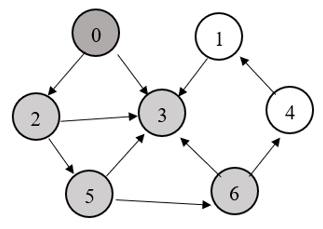
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | G | G | W | G | W |
| D | 1 | I | 2 | 2 | I | 3 | I |
| P | N | N | 0 | 0 | N | 3 | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



**Шаг 4.**

Вершина номер 5 имеет, одну смежную неокрашенную вершину, это 6. Закрашиваем ее в серый цвет.

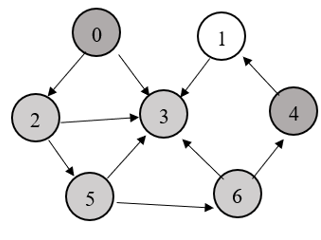
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | G | G | W | G | G |
| D | 1 | I | 2 | 2 | I | 3 | 4 |
| P | N | N | 0 | 0 | N | 3 | 4 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



**Шаг 5.**

Вершина номер 6 имеет одну смежную не закрашенную вершину, это номер 4, закрашиваем ее, в серый цвет.

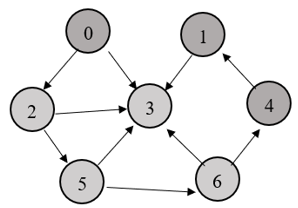
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | G | G | G | G | G |
| D | 1 | I | 2 | 2 | 5 | 3 | 4 |
| P | N | N | 0 | 0 | 5 | 3 | 4 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



**Шаг 6.**

Вершина 4 имеет одну смежную вершину, это один, закрашиваем ее в серый цвет.

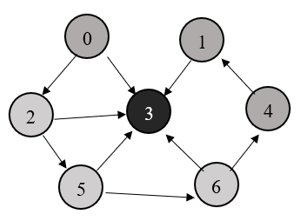
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | G | G | G | G |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



**Шаг 7.**

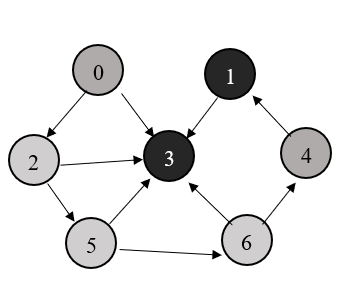
Вершина 1 имеет смежную вершину номер 3, но она уже закрашена в серый цвет и не имеет смежную вершин, следовательно, закрашиваем вершину номер 3 в черный цвет и таким образом двигаемся по обратному пути.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | B | G | G | G |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 0 | 0 | 7 | 0 | 0 | 0 |



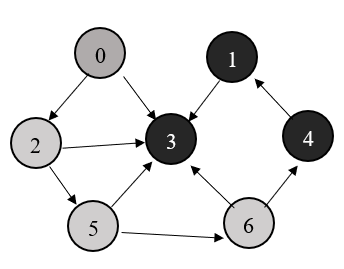
**Шаг 8**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | B | G | G | G |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 8 | 0 | 7 | 0 | 0 | 0 |



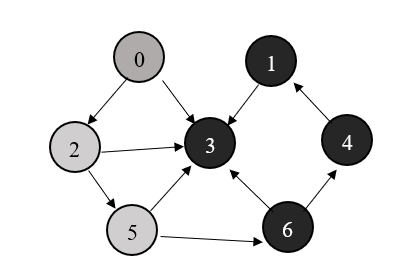
**Шаг 9.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | B | B | G | G |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 8 | 0 | 7 | 9 | 0 | 0 |



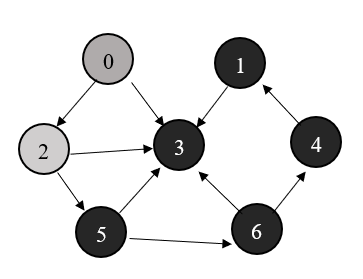
**Шаг 10**.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | B | B | G | B |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 8 | 0 | 7 | 9 | 0 | 10 |



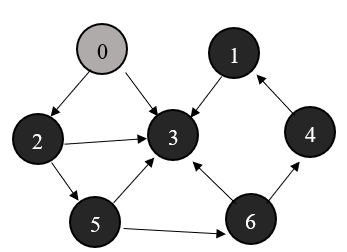
**Шаг 11.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | G | B | B | B | B |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 8 | 0 | 7 | 9 | 11 | 10 |



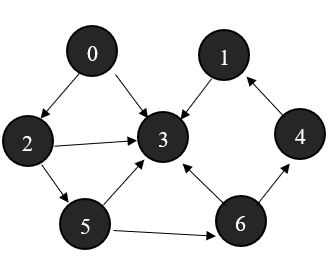
**Шаг 12.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | B | B | B | B | B |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 0 | 8 | 12 | 7 | 9 | 11 | 10 |

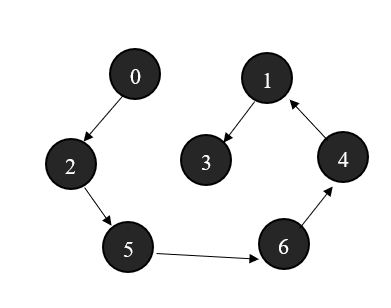


**Шаг 13.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | B | B | B | B | B |
| D | 1 | 6 | 2 | 2 | 5 | 3 | 4 |
| P | N | 6 | 0 | 0 | 5 | 2 | 4 |
| F | 13 | 8 | 12 | 7 | 9 | 11 | 10 |



В результате получили DFS-дерево:



Порядок обхода: 3,1,4,6,5,2,0.

**Алгоритм топологической сортировки.**

Топологическая сортировка – это процедура упорядочивания вершин ориентированного графа, не имеющего циклов.

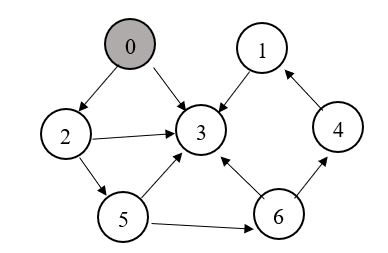
При реализации топологической сортировки с помощью алгоритма поиска в глубину используется массив меток вершин, с помощью которого моделируется удаление вершин из графа и сохраняются новые номера вершин.



По условию, граф имеет 7 вершин, пронумерованных начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

**Шаг 1.**

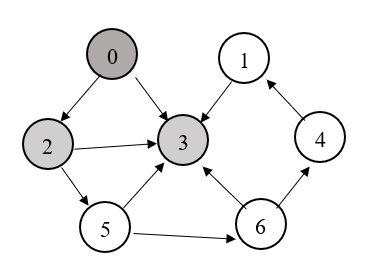
В качестве стартовой вершины выбираем вершину с номером 0. Окрашиваем ее в серый цвет.



**Шаг 2.**

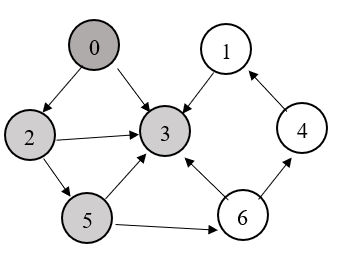
Вершина 0 имеет 2 смежные вершины 2 и 3. Закрашиваем их серым цветом.

И выбираем т что с наименьшим весом.



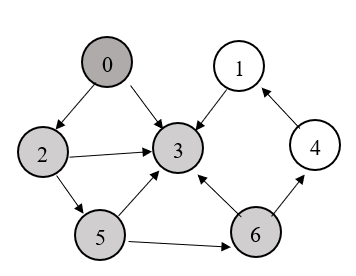
**Шаг 3.**

Вершина 2 имеет одну не отмеченную смежную вершину номер 5, закрашиваем ее в серый цвет.



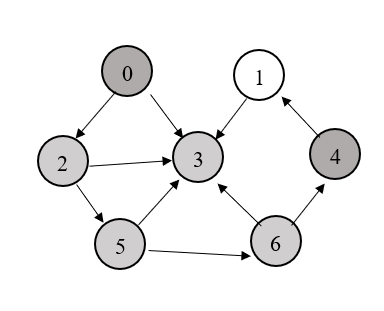
**Шаг 4.**

Вершина номер 5 имеет, одну смежную неокрашенную вершину, это 6. Закрашиваем ее в серый цвет.



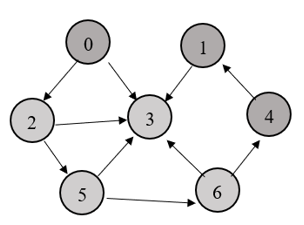
**Шаг 5.**

Вершина номер 6 имеет одну смежную не закрашенную вершину, это номер 4, закрашиваем ее, в серый цвет.



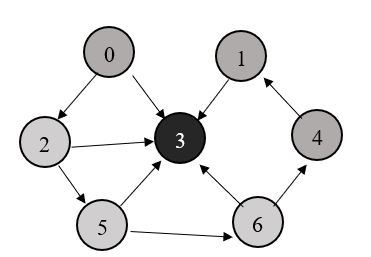
**Шаг 6.**

Вершина 4 имеет одну смежную вершину, это один, закрашиваем ее в серый цвет.



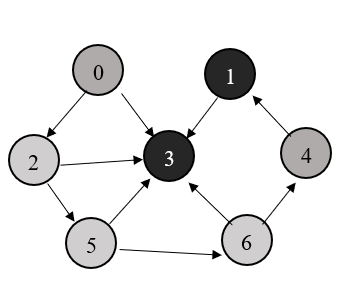
**Шаг 7.**

Вершина 1 имеет смежную вершину номер 3, но она уже закрашена в серый цвет и не имеет смежную вершин следовательно закрашиваем вершину номер 3 в черный цвет и таким образом двигаемся по обратному пути. Добавляем вершину 3 в стек.



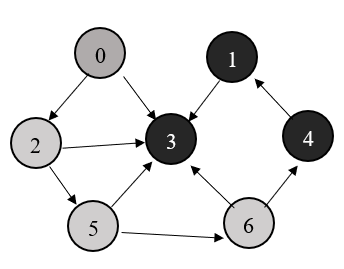
**Шаг 8.**

Добавляем вершину 1 в стек.



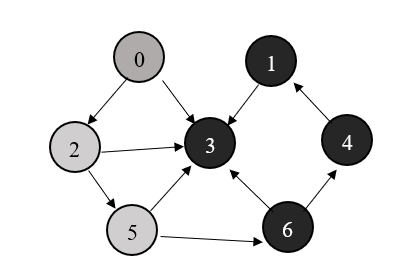
**Шаг 9.**

Добавляем вершину 4 в стек.



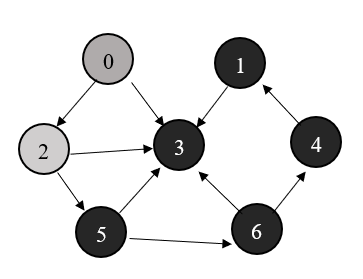
**Шаг 10.**

Добавляем вершину 6 в стек.



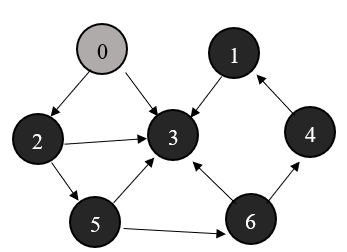
**Шаг 11.**

Добавляем вершину 5 в стек.



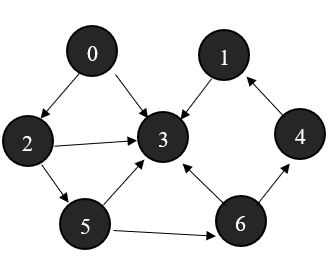
**Шаг 12.**

Добавляем вершину 2 в стек.



**Шаг 13.**

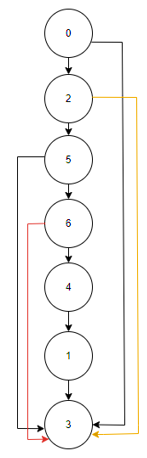
Добавляем вершину 0 в стек.



Таким образом имеем сортировку:

3, 1, 4, 6, 5, 2, 0.

Схема:



***Задание 3.*** Осуществить программную реализацию алгоритмов на C++. Разработать структуры **AMatrix** и **АList**  для представления ориентированного графа матричным и списковым способом. Разработать функции преобразования из одного способа представления в другой. Разработать функцию **BFS** обхода вершин графа, используя метод поиска в ширину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

Код:

// --- Graph.h

//

#pragma once

#include <list>

#include "Graph.h"

namespace graph

{

struct AList;

struct AMatrix // матрица смежности

{

int n\_vertex; // количество вершин

int \*mr; // матрица

AMatrix(int n); // создать нулевую матрицу n\*n

AMatrix(int n, int mr[]); // создать матрицу n\*n и

AMatrix(const AMatrix& am); // создать подобную матрицу

AMatrix(const AList& al); // создать матрицу из спискового

void set(int i, int j, int r); // записать mr[i,j] = r

int get(int i, int j)const; // элемент mr[i,j]

};

struct AList // списки смежности

{

int n\_vertex; // количество вершин

std::list<int> \*mr; // массив списков

void create(int n); // создать массив пустых списков

AList(int n); // создать массив пустых списков

AList(int n, int mr[]); // создать списковое представление

AList(const AMatrix& am); // создать списковое представление

AList(const AList& al); // создать подобную структуру

void add(int i, int j); // добавить в i-ый список

int size(int i) const; // размер i-го списка

int get(int i, int j)const; // j-ый элемент i-го списка

};

};

// --- Graph.cpp

//

#include "stdafx.h"

#include "Graph.h"

namespace graph

{

AMatrix::AMatrix(int n)

{

this->n\_vertex = n;

this->mr = new int [this->n\_vertex\*this->n\_vertex];

for(int i = 0; i < n\*n; i++)mr[i] = 0;

};

AMatrix::AMatrix(int n, int mr[])

{

this->n\_vertex = n;

this->mr = mr;

};

AMatrix::AMatrix(const AMatrix& am)

{

this->n\_vertex = am.n\_vertex;

this->mr = new int [this->n\_vertex\*this->n\_vertex];

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

this->set(i,j, am.get(i,j));

};

AMatrix::AMatrix(const AList& al)

{

this->n\_vertex = al.n\_vertex;

this->mr = new int [this->n\_vertex \* this->n\_vertex];

for(int k = 0; k < this->n\_vertex\*this->n\_vertex; k++)mr[k] = 0;

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < al.size(i); j++) this->set(i,al.get(i,j),1);

};

void AMatrix::set(int i, int j, int r) {this->mr[i\*this->n\_vertex+j] = r;};

int AMatrix::get(int i, int j)const

{return this->mr[i\*this->n\_vertex+j];};

void AList::create(int n)

{this->mr = new std::list<int>[this->n\_vertex = n];};

AList::AList(int n) {create(n);}

AList::AList(const AMatrix& am)

{

this->create(am.n\_vertex);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

if (am.get(i,j)!=0) this->add(i,j);

};

AList::AList(const AList& al)

{

this->create(al.n\_vertex);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < al.size(i); j++) this->add(i, al.get(i,j));

};

AList::AList(int n, int mr[])

{

this->create(n);

for (int i = 0; i < this->n\_vertex; i++)

for (int j = 0; j < this->n\_vertex; j++)

if (mr[i\*this->n\_vertex+j]!= 0) this->add(i,j);

};

void AList::add(int i, int j){this->mr[i].push\_back(j);};

int AList::size(int i) const {return (int)this->mr[i].size();};

int AList::get(int i, int j)const

{

std::list<int>::iterator rc = this->mr[i].begin();

for (int k = 0; k < j; k++) rc++;

return (int)\*rc;

};

};

// ---BFS.h

//

#pragma once

#include "Graph.h"

#include <queue>

struct BFS // breadth-first search поиск в ширину (связный граф)

{

const static int INF = 0x7fffffff;

const static int NIL = -1;

enum Color {WHITE, GRAY, BLACK}; //

const graph::AList \*al; // исходный граф

Color \*c; // цвет вершины

int \*d; // расстояние до вершины

int \*p; // предшествующая вершина

std::queue<int> q; // очередь

BFS(const graph::AList& al, int s);

BFS(const graph::AMatrix& am, int s);

void init(const graph::AList& al, int s);

int get(); // получить следующую вершину

};

// ---BFS.cpp

//

#include "stdafx.h"

#include "BFS.h"

void BFS::init(const graph::AList& al, int s)

{

this->al = &al;

this->c = new Color[this->al->n\_vertex];

this->d = new int[this->al->n\_vertex];

this->p = new int[this->al->n\_vertex];

for (int i = 0; i < this->al->n\_vertex; i++)

{

this->c[i] = WHITE;

this->d[i] = INF;

this->p[i] = NIL;

};

this->c[s] = GRAY;

this->q.push(s);

};

BFS::BFS(const graph::AList& al, int s){this->init(al,s);};

BFS::BFS(const graph::AMatrix& am, int s)

{

this->init(\*(new graph::AList(am)),s);

};

int BFS::get()

{

int rc = NIL, v = NIL;

if (!this->q.empty())

{

rc = this->q.front();

for (int j = 0; j < this->al->size(rc); j++)

if (this->c[v = this->al->get(rc,j)] == WHITE)

{

this->c[v] = GRAY;

this->d[v] = this->d[rc]+1;

this->p[v] = rc;

this->q.push(v);

};

this->q.pop();

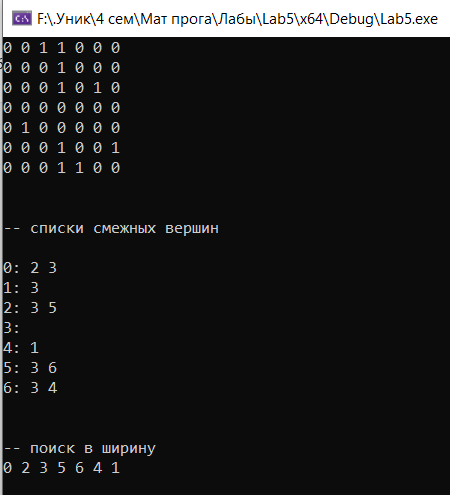
this->c[rc] = BLACK;

};

return rc;

}

Результат выполнения кода:



***Задание 4.*** Разработать функцию **DFS**  обхода вершин графа, используя метод поиска глубину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

// ---DFS.h

//

#pragma once

#include "Graph.h"

#include <vector>

struct DFS // depth-first search поиск в глубину

{

const static int NIL = -1;

enum Color {WHITE, GRAY, BLACK}; //

const graph::AList \*al; // исходный граф

Color \*c; // цвет вершины

int \*d; // время обнаружения

int \*f; // время завершения обработки

int \*p; // предшествующая вершина

int t; // текущее время

DFS(const graph::AList& al);

DFS(const graph::AMatrix& am);

std::vector <int> topological\_sort; //результат топологической сортировки

void visit(int v);

void init(const graph::AList& al);

int get(int i); // получить вершину

};

// ---DFS.cpp

//

#include "stdafx.h"

#include "DFS.h"

#define NINF 0x80000000

#define INF 0x7fffffff

void DFS::init(const graph::AList& al)

{

this->al = &al;

this->c = new Color[this->al->n\_vertex];

this->d = new int[this->al->n\_vertex];

this->f = new int[this->al->n\_vertex];

this->p = new int[this->al->n\_vertex];

this->t = 0;

for (int i = 0; i < this->al->n\_vertex; i++)

{

this->c[i] = WHITE;

this->d[i] = this->f[i] = 0;

this->p[i] = NIL;

};

for (int i = 0; i < this->al->n\_vertex; i++)

if (this->c[i] == WHITE)

{this->visit(i);

this->topological\_sort.push\_back (i);}

};

DFS::DFS(const graph::AList& al){this->init(al);};

DFS::DFS(const graph::AMatrix& am)

{

this->init(\*(new graph::AList(am)));

};

void DFS::visit(int u)

{

int v = NIL;

this->c[u] = GRAY;

this->d[u]= ++(this->t);

for (int j = 0; j < this->al->size(u); j++)

if (this->c[v = this->al->get(u,j)] == WHITE)

{

this->p[v] = u;

this->visit(v);

this->topological\_sort.push\_back (v);

}

this->c[u] = BLACK;

this->f[u]= ++(this->t);

};

int DFS::get(int i)

{

int j = 0, min1 = INF , min2 = NINF, ntx = NIL;

for (int j = 0; j <= i; j++) // iая статистика

{

for (int k = 0; k < this->al->n\_vertex; k++)

if (this->f[k] < min1 && this->f[k] > min2)

{min1 = this->f[k]; ntx = k;};

min2 = min1; min1 = INF;

};

return ntx;

};

// Graphs\_Optimization.cpp: определяет точку входа для консольного приложения.

// --- main

// алгоритмы BFS и DFS

#include "stdafx.h"

#include <iostream>

#include "Graph.h"

#include "BFS.h"

#include "DFS.h"

int \_tmain(int argc, \_TCHAR\* argv[])

{

int m[7][7] ={

{0, 0, 1, 1, 0, 0, 0},

{0, 0, 0, 1, 0, 0, 0},

{0, 0, 0, 1, 0, 1, 0},

{0, 0, 0, 0, 0, 0, 0},

{0, 1, 0, 0, 0, 0, 0},

{0, 0, 0, 1, 0, 0, 1},

{0, 0, 0, 1, 1, 0, 0}

};

setlocale(LC\_ALL, "rus");

graph::AMatrix g1(7,(int\*) m);

std::cout<<std::endl;

std::cout<<std::endl<< "-- матрица смежности "<< std::endl;

for (int i = 0; i < g1.nV; i++)

{

std::cout<<std::endl;

for (int j = 0; j < g1.nV; j++)

std::cout<< g1.get(i,j) << " ";

};

std::cout<<std::endl;

graph::AList g2(g1);

std::cout<<std::endl;

std::cout<<std::endl<< "-- списки смежных вершин "<< std::endl;

for (int i = 0; i < g1.nV; i++)

{

std::cout<<std::endl<< i << ": ";

for (int j = 0; j < g2.size(i); j++)

std::cout<<g2.get(i,j)<< " ";

}

std::cout<<std::endl;

BFS b1(g2,0);

std::cout<<std::endl;

std::cout<<std::endl<< "-- поиск в ширину "<< std::endl;

int k1;

while ((k1 = b1.get())!= BFS::NIL)

std::cout<< k1 << " ";

std::cout<<std::endl;

DFS b2(g2);

std::cout<<std::endl;

std::cout<<std::endl<< "-- поиск в глубину "<< std::endl;

for (int i = 0; i < g2.nV; i++)

std::cout<< b2.get(i) << " ";

std::cout<<std::endl;

std::cout << std::endl << "Топологическая сортировка" << std::endl;

for (std::vector <int>::iterator i (b2.topological\_sort.begin ()); i != b2.topological\_sort.end (); ++i)

std::cout << \*i << ' ';

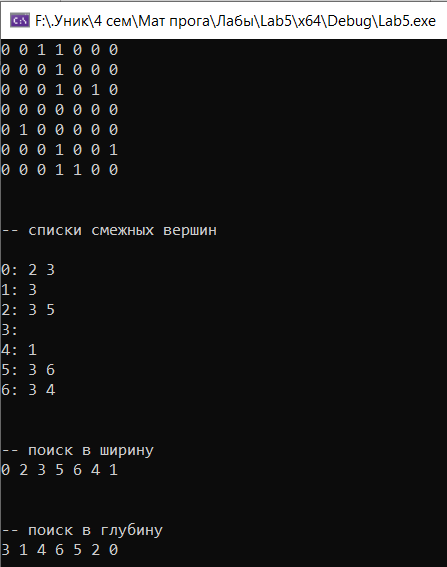
std::cout << std::endl;

system("pause");

return 0;

}

Результат выполнения кода:



***Задание 5.*** Доработайте функцию **DFS**,для выполнения топологической сортировки графа. Продемонстрировать работу функции. Копии экрана вставить в отчет.

сpp

#include "stdafx.h"

#include "DFS.h"

#define NINF 0x80000000

#define INF 0x7fffffff

void DFS::init(const graph::AList& al)

{

this->al = &al;

this->c = new Color[this->al->nV];

this->d = new int[this->al->nV];

this->f = new int[this->al->nV];

this->p = new int[this->al->nV];

this->t = 0;

for (int i = 0; i < this->al->nV; i++)

{

this->c[i] = WHITE;

this->d[i] = this->f[i] = 0;

this->p[i] = NIL;

};

for (int i = 0; i < this->al->nV; i++)

if (this->c[i] == WHITE)

{

this->visit(i);

this->topological\_sort.push\_back (i);

}

};

DFS::DFS(const graph::AList& al)

{

this->init(al);

};

DFS::DFS(const graph::AMatrix& am)

{

this->init(\*(new graph::AList(am)));

};

void DFS::visit(int u)

{

int v = NIL;

this->c[u] = GRAY;

this->d[u]= ++(this->t);

for (int j = 0; j < this->al->size(u); j++)

if (this->c[v = this->al->get(u,j)] == WHITE)

{

this->p[v] = u;

this->visit(v);

this->topological\_sort.push\_back (v);

}

this->c[u] = BLACK;

this->f[u]= ++(this->t);

};

int DFS::get(int i)

{

int j = 0, min1 = INF , min2 = NINF, ntx = NIL;

for (int j = 0; j <= i; j++)

{

for (int k = 0; k < this->al->nV; k++)

if (this->f[k] < min1 && this->f[k] > min2)

{

min1 = this->f[k];

ntx = k;

};

min2 = min1; min1 = INF;

};

return ntx;

};

dfs.h

#pragma once

#include "Graph.h"

#include <vector>

struct DFS // depth-first search поиск в глубину

{

const static int NIL = -1;

enum Color { WHITE, GRAY, BLACK }; //

const graph::AList \*al; // исходный граф

Color \*c; // цвет вершины

int \*d; // время обнаружения

int \*f; // время завершения обработки

int \*p; // предшествующая вершина

int t; // текущее время

DFS(const graph::AList& al);

DFS(const graph::AMatrix& am);

std::vector <int> topological\_sort; //результат топологической сортировки

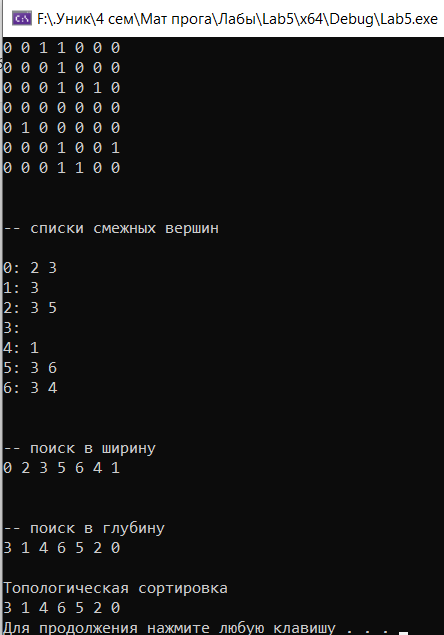
void visit(int v);

void init(const graph::AList& al);

int get(int i); // получить вершину

};

Результат выполнения кода:



***Задание 6.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Прима. Шаги построения отразить в отчете.

Веса ребер принять:

W:

W(e0,1)=8; W(e1,0)=5;

W(e0,2)=1; W(e2,0)=3;

W(e0,3)=2; W(e3,0)=8;

W(e1,3)=11; W(e3,1)=4;

W(e1,4)=5; W(e4,1)=3;

W(e2,3)=7; W(e3,2)=9;

W(e2,5)=11; W(e5,2)=10;

W(e4,3)=4; W(e3,4)=1;

W(e4,6)=10; W(e6,4)=2;

W(e5,6)=2; W(e6,5)=6;

W(e5,3)=3; W(e3,5)=6;

W(e6,3)=7; W(e3,6)=9;

