**Белорусский государственный технологический университет**

**Факультет информационных технологий**

**Специальность программная инженерия**

Отчёты по лабораторным работам

По дисциплине «Математическое программирование»

Выполнил:

Студент 2 курса 8 группы

Соленок Анастасия Александровна

Преподаватель: асс. Ромыш А.С.

2025, Минск

**Содержание**

[Лабораторная работа №1. Вспомогательные функции 3](#_Toc193543675)

[Лабораторная работа №2. Комбинаторные алгоритмы 9](#_Toc193543676)

[Лабораторная работа №3. Метод ветвей и границ. Задача коммивояжера и методы её решения 29](#_Toc193543677)

[Лабораторная работа №4. Динамическое программирование 40](#_Toc193543678)

[Лабораторная работа №5. Транспортная задача 52](#_Toc193543679)

[Лабораторная работа №6. Оптимизационные задачи на графах 59](#_Toc193543680)

[Лабораторная работа №7. Сетевые модели 74](#_Toc193543681)

[Лабораторная работа №8. Графический метод решения задач оптимизации 76](#_Toc193543682)

**Лабораторная работа №1. Вспомогательные функции**

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

**Ход работы**

## **1.1. Написание кода программы.**

Код файла main.cpp приведен в листинге 1.1.

|  |
| --- |
| #include "stdafx.h"  #include "Auxil.h" // вспомогательные функции  #include <iostream>  #include <ctime>  #include <locale>  #define CYCLE 1000000 // количество циклов  int Fib(int i)  {  int value = 0;  if (i < 1)  return 0;  if (i == 1)  return 1;  return Fib(i - 1) + Fib(i - 2);  }  int \_tmain(int argc, \_TCHAR\* argv[])  {  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;  for (int n = 20; n < 41; n++)  {  t1 = clock();  int num = Fib(n);  t2 = clock();  std::cout << std::endl << n << "-е число Фибоначчи " << t2 - t1 << " у.е.";  }  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 1.1. Содержание файла main.cpp

Код файла Auxil.h приведен в листинге 1.2.

|  |
| --- |
| #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  }; |

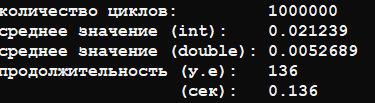
Листинг 1.2. Содержание файла Auxil.h

Код файла Auxil.cpp приведет в листинге 1.3.

|  |
| --- |
| #include "stdafx.h"  #include "Auxil.h"  #include <ctime>  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);  };  } |

Листинг 1.3. Содержание файла Auxil.cpp

Результат работы программы представлен на рисунке 1.1.



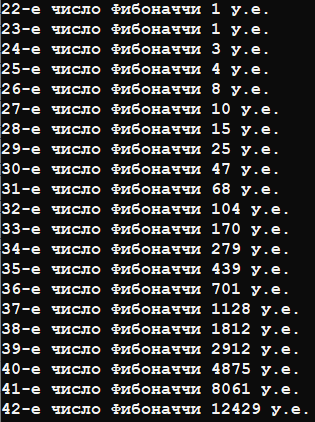


Рисунок 1.1 Результаты работы программы

## **1.2. Измерение скорости выполнения функции генерации случайных чисел.**

Результаты измерений и соответствующий график приведены на рисунке 1.2.

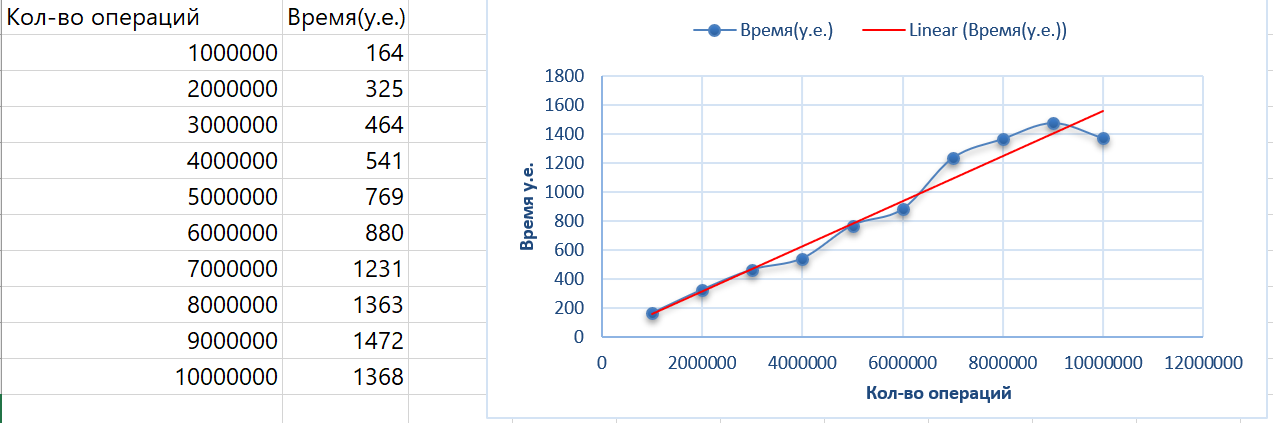


Рисунок 1.2. Результаты измерений и их график

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

## **1.3. Измерение скорости выполнения функции подсчета n-го числа Фибоначчи.**

Результаты измерений и соответствующий график приведены на рисунке 1.3.

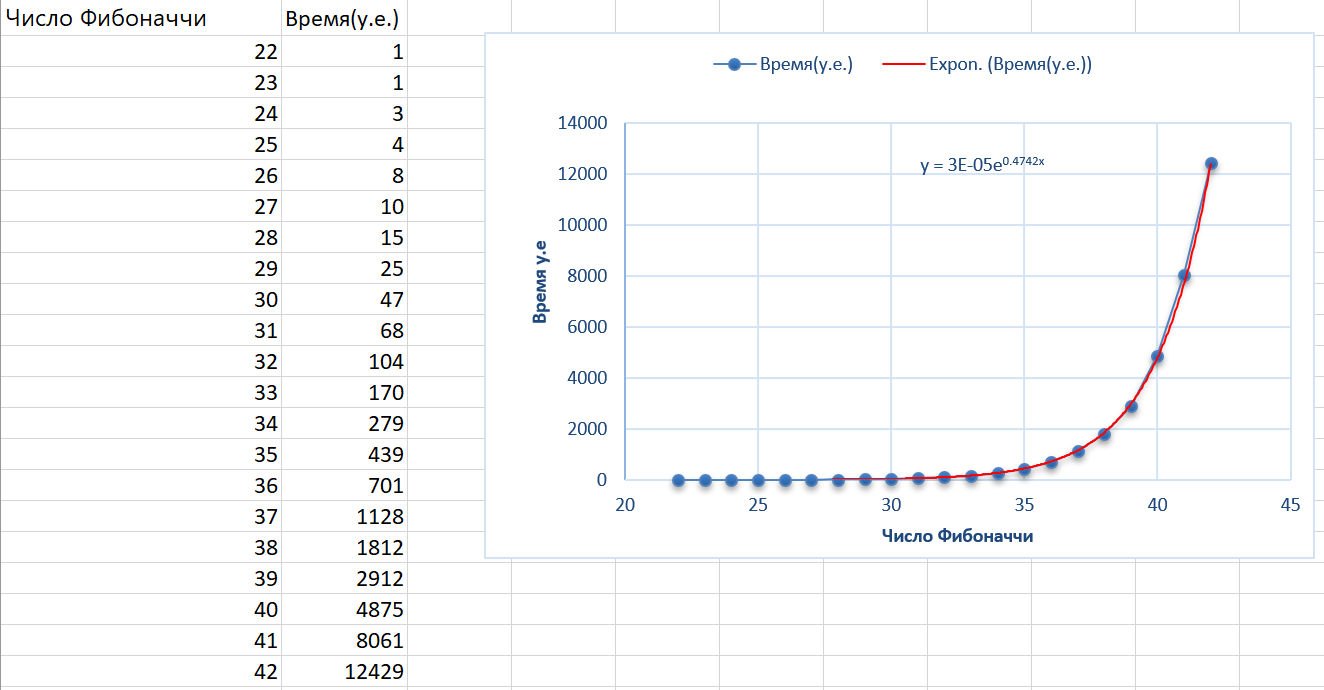


Рисунок 1.3. Результаты измерений и их график

Вывод: скорость выполнения программы экспоненциально зависит от порядкового номера числа Фибоначчи.

**Лабораторная работа №2. Комбинаторные алгоритмы**

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

В этом задании было необходимо разработать генератор подмножеств заданного множества. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

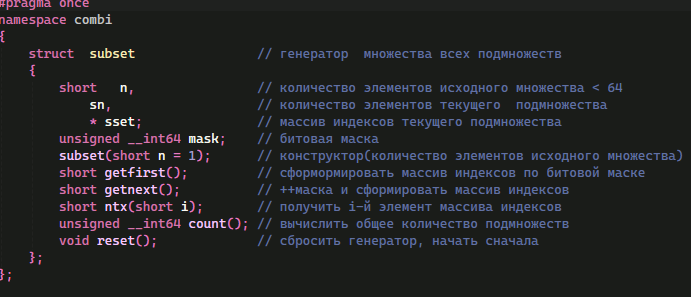


Рисунок 1 – Combi.h

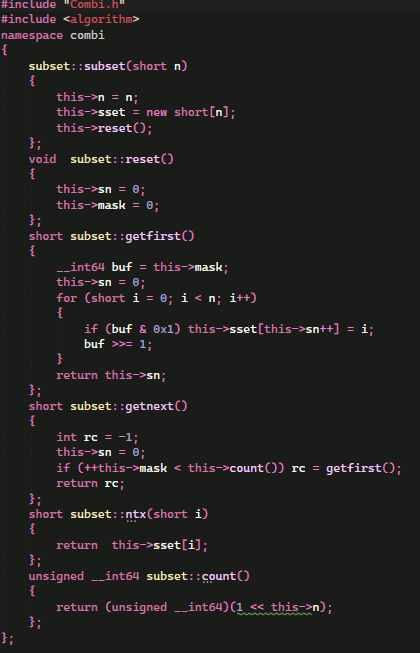


Рисунок 2 - Combi.cpp

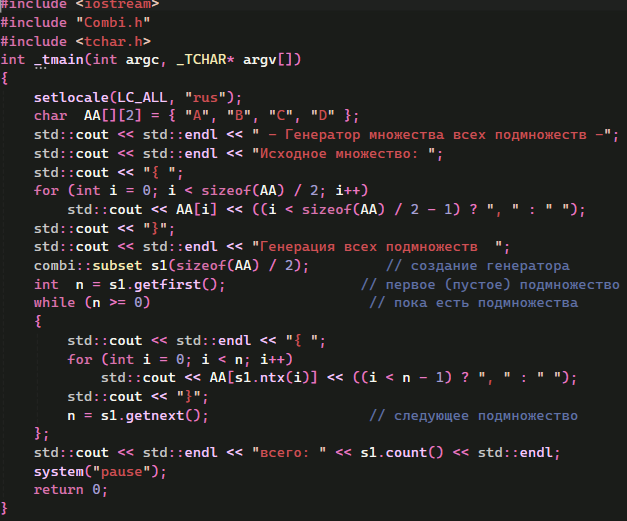


Рисунок 3 - Subsets.cpp

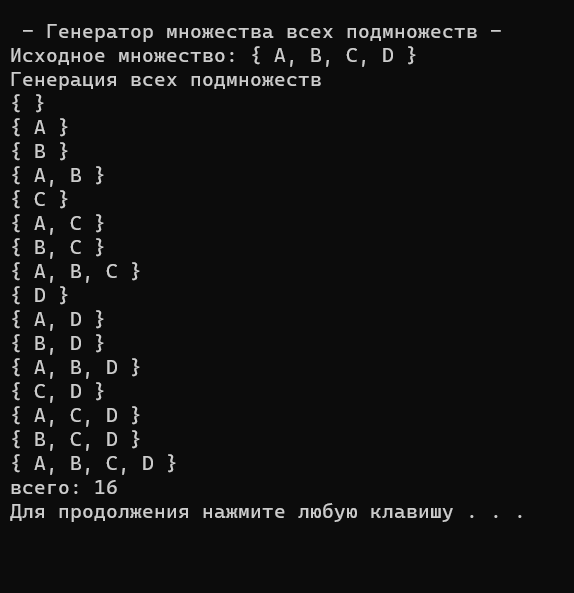


Рисунок 4 – Результат выполнения

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

В этом задании было необходимо разработать генератор сочетаний. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

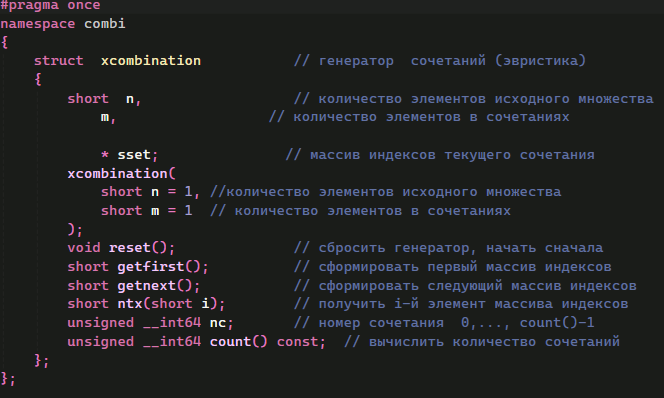


Рисунок 5 - Combi.h

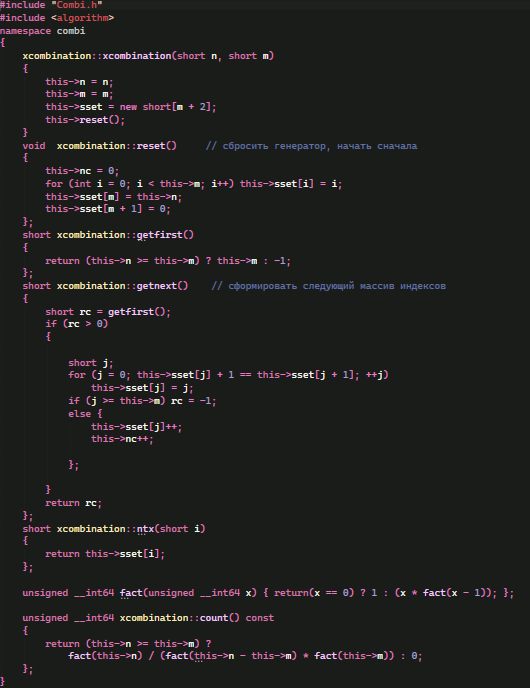


Рисунок 6 - Combi.cpp

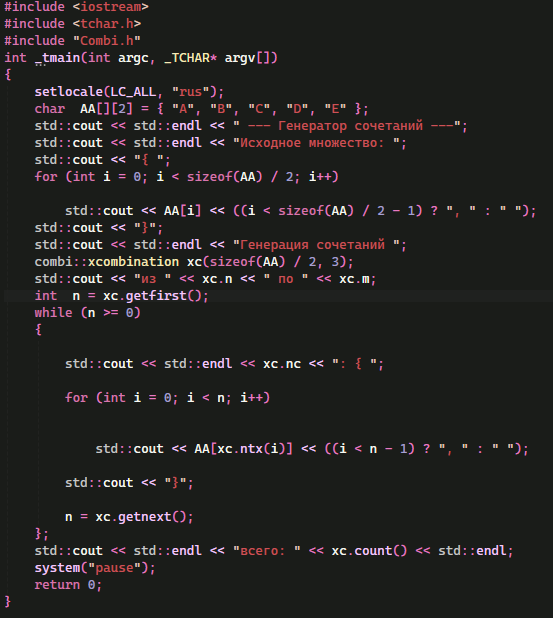


Рисунок 7 -Combinations.cpp

Результат выполнения представлен на рисунке 8.

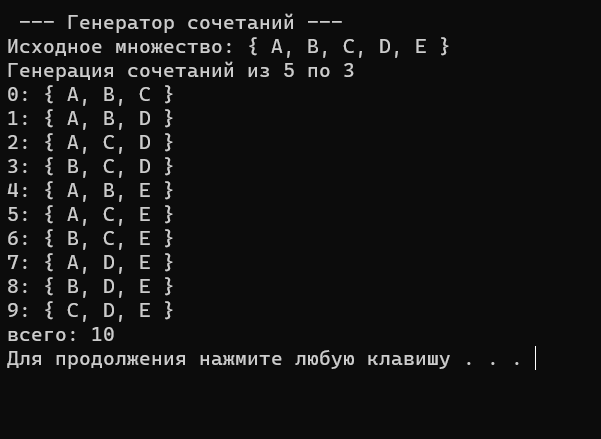


Рисунок 8 – результат выполнения программы

**Задание 3.** Разобрать и разработать генератор перестановок.

В этом задании было необходимо разработать генератор перестановок. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

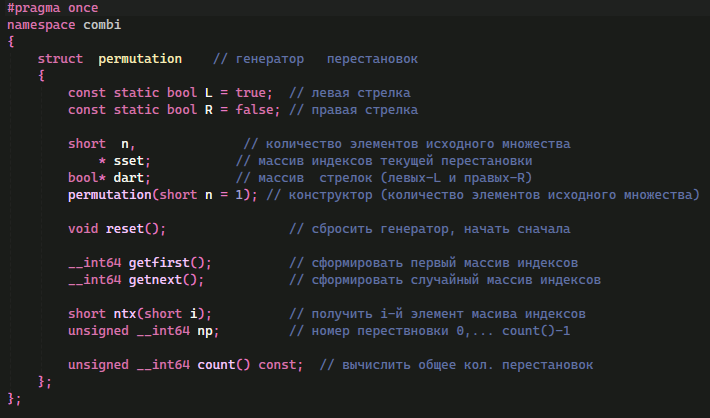


Рисунок 9 – Combi.h

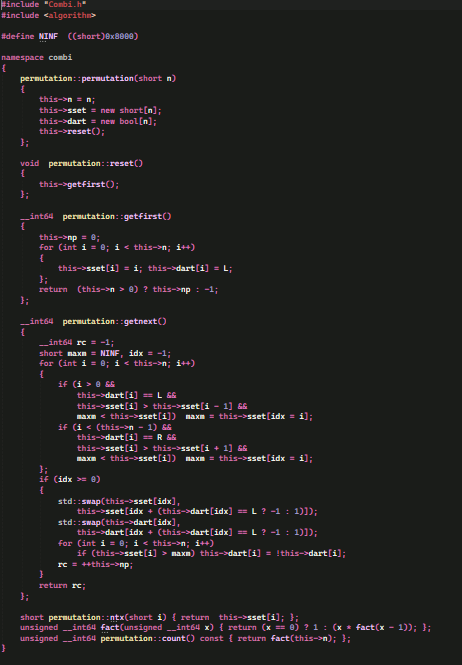


Рисунок 10 – Combi.cpp

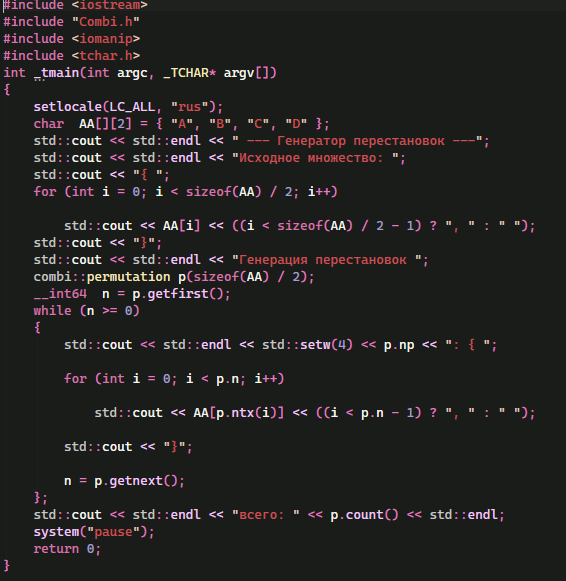


Рисунок 11 –Permitations.cpp

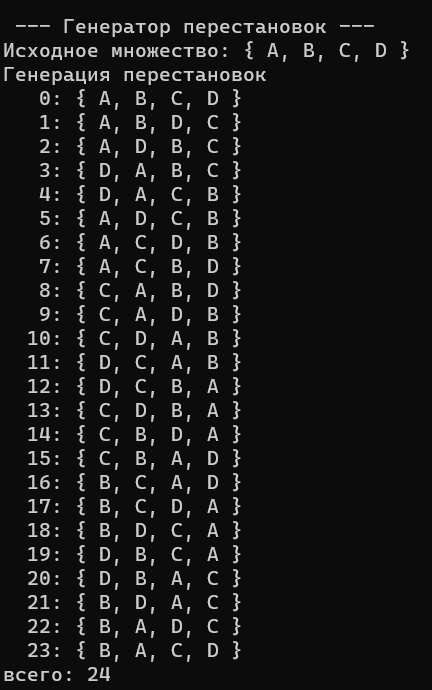


Рисунок 12 – результат выполнения программы

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

В этом задании было необходимо разработать генератор размещений. Ниже на рисунках представлены заголовочные файлы и файлы cpp с реализацией поставленной задачи.

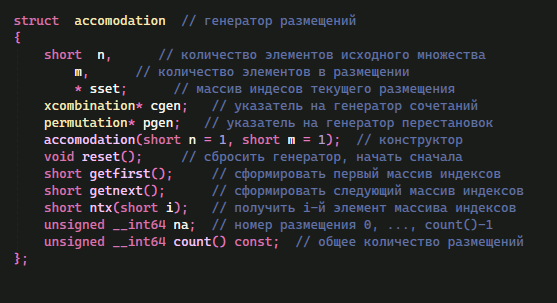


Рисунок 13 – Combi.h

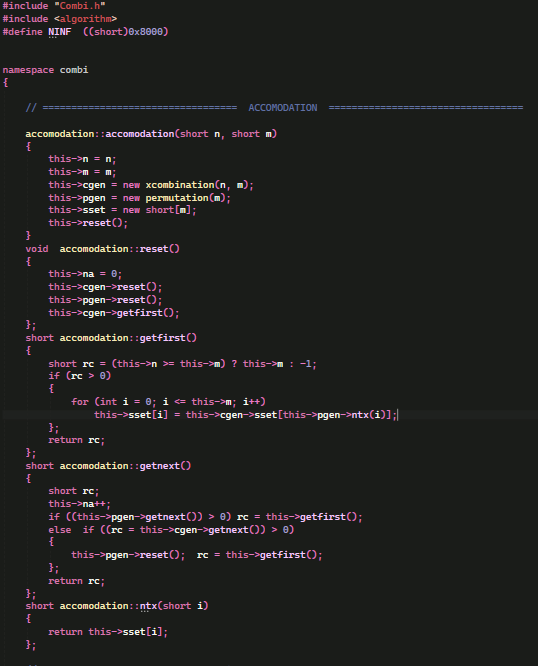


Рисунок 14 – Combi.cpp

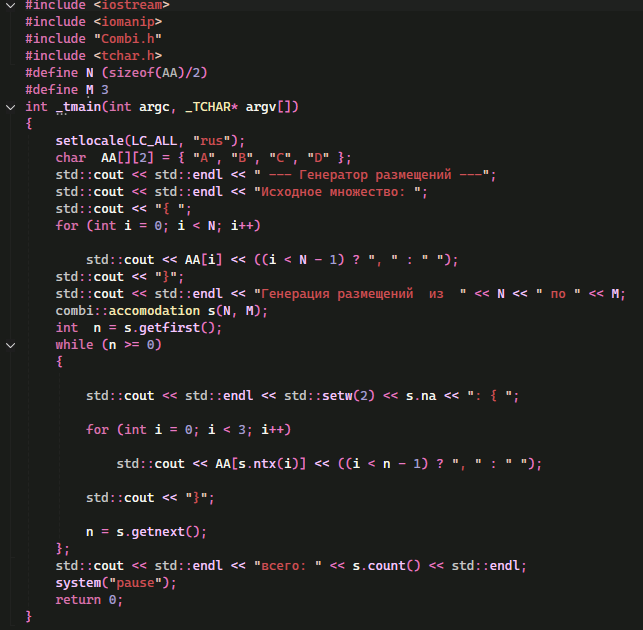


Рисунок 15 –Accommodations.cpp

Результат выполнения программы представлен на рисунке 16.

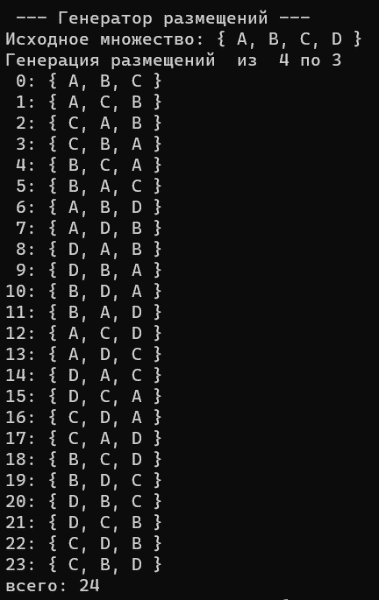


Рисунок 16 - результат выполнения программы

**Задание 5.** Решить в соответствии с вариантом задачу и результат занести в отчет.

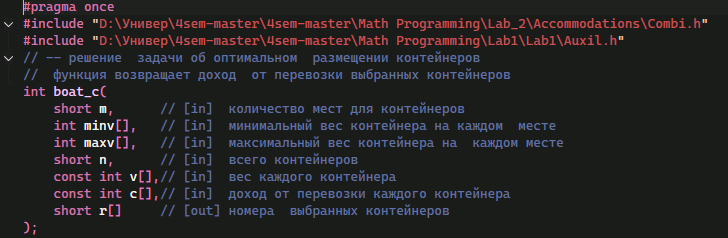
Об оптимальной загрузке судна с условием центровки (веса контейнеров сгенерировать случайным образом: количество мест на судне для контейнеров – 5, количество контейнеров 8, веса контейнеров 100 – 200 кг, доход от перевозки 10 – 100 у.е; минимальный вес контейнера для каждого места 50 – 120 кг, максимальный вес контейнера для каждого места 150 – 850 кг); 

Рисунок 17 – Boat.h

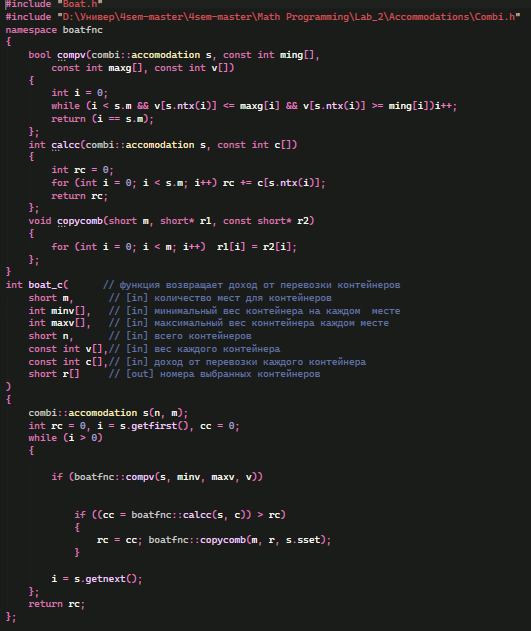


Рисунок 18 – Boat.cpp

Результат выполнения программы представлен на рисунке 19.

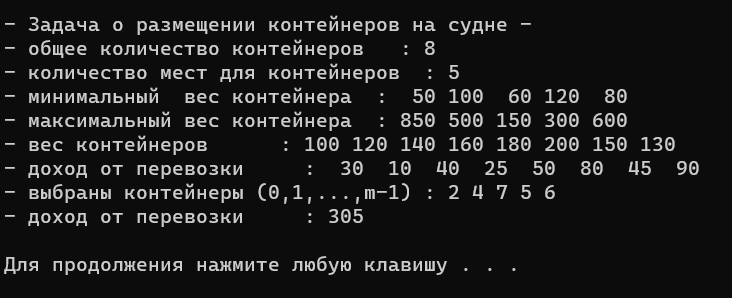


Рисунок 19 – результат работы программы

**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи от размерности задачи и результат в виде графика:

об оптимальной загрузке судна с условием центровки (количество мест на судне для контейнеров 4 – 8);

Результат работы программы на рисунке 20.

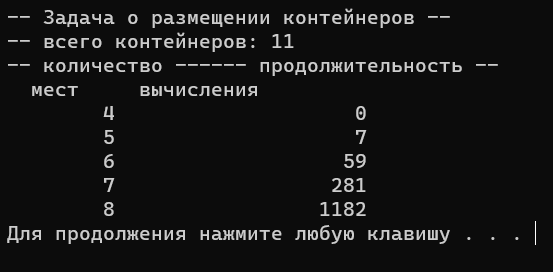


Рисунок 20 – Зависимость времени выполнения от количества мест для контейнеров



Рисунок 21 – график зависимости

**Вывод:** исходя из полученных данных и графика, можно заметить, что скорость выполнения программы экспоненциально возрастает при увеличении количества мест для контейнеров.

**Лабораторная работа №3. Метод ветвей и границ. Задача коммивояжера и методы её решения**

**Цель:** 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.

Задачей коммивояжера является задача, в которой даны города и расстояния между ними, которые можно представить узлами графа. Решением данной задачи является число – минимальное расстояние, которое необходимо пройти, чтобы посетить все города ровно 1 раз.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** | INF | 2 \* n | 21 + n | INF | n |
| **2** | n | INF | 15 + n | 68 - n | 84 - n |
| **3** | 2 + n | 3 \* n | INF | 86 | 49 + n |
| **4** | 17 + n | 58 - n | 4 \* n | INF | 3 \* n |
| **5** | 93 - n | 66 + n | 52 | 13 + n | INF |

Таблица 1 – исходная таблица расстояний с параметром

На основе данной таблицы, используя параметр «n», необходимо составить таблицу расстояний, на основе которой будет решаться задача. Поля «INF» в данной таблице обозначают, что из вершины «i» в вершину «j» нет пути. Учитывая, что мой вариант 12, было принято решения принять параметр «n» равным 12. С учетом этого была получена следующая таблица.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** | ∞ | 24 | 33 | ∞ | 12 |
| **2** | 12 | ∞ | 27 | 56 | 72 |
| **3** | 14 | 36 | ∞ | 86 | 61 |
| **4** | 29 | 46 | 48 | ∞ | 36 |
| **5** | 81 | 78 | 52 | 25 | ∞ |

Таблица 2 – исходная таблица расстояний с подставлен параметром

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

Ниже представлены шаги решения задачи, граф разбора представлен в конце.

Проводим редукцию матрицы, находим минимальный элемент в каждой строке, а затем из каждой строки отнимаем её минимальный элемент.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 4 | 5 | di |
| 1 | ∞ | 24 | 33 | ∞ | 12 | 12 |
| 2 | 12 | ∞ | 27 | 56 | 72 | 12 |
| 3 | 14 | 36 | ∞ | 86 | 61 | 14 |
| 4 | 29 | 46 | 48 | ∞ | 36 | 29 |
| 5 | 81 | 78 | 52 | 25 | ∞ | 25 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 4 | 5 |
| 1 | ∞ | 12 | 21 | ∞ | 0 |
| 2 | 0 | ∞ | 15 | 44 | 60 |
| 3 | 0 | 22 | ∞ | 72 | 47 |
| 4 | 0 | 17 | 19 | ∞ | 7 |
| 5 | 56 | 53 | 27 | 0 | ∞ |

Проводим редукцию по столбам.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 4 | 5 |
| 1 | ∞ | 12 | 21 | ∞ | 0 |
| 2 | 0 | ∞ | 15 | 44 | 60 |
| 3 | 0 | 22 | ∞ | 72 | 47 |
| 4 | 0 | 17 | 19 | ∞ | 7 |
| 5 | 56 | 53 | 27 | 0 | ∞ |
| dj | 0 | 12 | 15 | 0 | 0 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 4 | 5 |
| 1 | ∞ | 0 | 6 | ∞ | 0 |
| 2 | 0 | ∞ | 0 | 44 | 60 |
| 3 | 0 | 10 | ∞ | 72 | 47 |
| 4 | 0 | 5 | 4 | ∞ | 7 |
| 5 | 56 | 41 | 12 | 0 | ∞ |

Определяем нижнюю границу H, которая равна сумме всех di и dj. H = 12 + 12 + 14 + 29 + 25 + 0 + 12 + 15 + 0 + 0 = 119.

Далее следует определить ребро ветвления, для этого каждую клетку, со значением 0 заменяем на INF и находит сумму минимальных элементов в её строке и столбце.

Эти суммы находятся в скобках после значения ячейки.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 4 | 5 |
| 1 | ∞ | 0(5) | 6 | ∞ | 0(13) |
| 2 | 0(0) | ∞ | 0(4) | 44 | 60 |
| 3 | 0(10) | 10 | ∞ | 72 | 47 |
| 4 | 0(4) | 5 | 4 | ∞ | 7 |
| 5 | 56 | 41 | 12 | 0(56) | ∞ |

Выбираем ячейку с максимальной суммой, это ячейка 5-4 и сумма равна 56. Нам следует определить, следует ли включить это ребро в маршрут или нет.

Предположим, что мы не взяли это ребро. Заменяем его на INF и снова находим суммы di и dj.

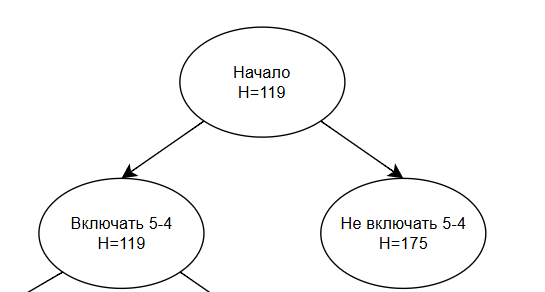
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 4 | 5 | di |
| 1 | ∞ | 0 | 6 | ∞ | 0 | 0 |
| 2 | 0 | ∞ | 0 | 44 | 60 | 0 |
| 3 | 0 | 10 | ∞ | 72 | 47 | 0 |
| 4 | 0 | 5 | 4 | ∞ | 7 | 0 |
| 5 | 56 | 41 | 12 | ∞ | ∞ | 12 |
| dj | 0 | 0 | 0 | 44 | 0 | 56 |

Сумма вышла равной 56, добавляем предыдущую границу H, 56 + 119 = 175.

Теперь возьмем ребро 5-4. Удалим 5 строку и 4 столбец, а также заменим ребро 4-5 на INF, чтобы не пойти туда.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 5 | di |
| 1 | ∞ | 0 | 6 | 0 | 0 |
| 2 | 0 | ∞ | 0 | 60 | 0 |
| 3 | 0 | 10 | ∞ | 47 | 0 |
| 4 | 0 | 5 | 4 | ∞ | 0 |
| dj | 0 | 0 | 0 | 0 | 0 |

Сумма всех di и dj равна 0, прибавляем прошлую границу, получаем H=119+0=119. Попадаем в следующую ситуацию.



Следует выбрать тот путь, где граница будет минимальной, то есть включим 5-4. Повторяем предыдущие шаги.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 5 |
| 1 | ∞ | 0(5) | 6 | 0(47) |
| 2 | 0(0) | ∞ | 0(4) | 60 |
| 3 | 0(10) | 10 | ∞ | 47 |
| 4 | 0(4) | 5 | 4 | ∞ |

Исключаем ребро 1-5.

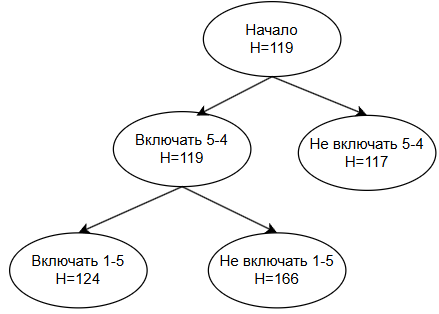
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | 5 | di |
| 1 | ∞ | 0 | 6 | ∞ | 0 |
| 2 | 0 | ∞ | 0 | 60 | 0 |
| 3 | 0 | 10 | ∞ | 47 | 0 |
| 4 | 0 | 5 | 4 | ∞ | 0 |
| dj | 0 | 0 | 0 | 47 | 47 |

Граница равна H=47+119=166.

Включаем ребро 1-5.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | di |
| 2 | 0 | ∞ | 0 | 0 |
| 3 | 0 | 10 | ∞ | 0 |
| 4 | 0 | 5 | 4 | 0 |
| dj | 0 | 5 | 0 | 5 |

Граница равна H=5+119=124.



Идем по пути включения 1-5.

|  |  |  |  |
| --- | --- | --- | --- |
| i j | 1 | 2 | 3 |
| 2 | 0(0) | ∞ | 0(4) |
| 3 | 0(10) | 10 | ∞ |
| 4 | 0(4) | 5 | 4 |

Ребром ветвления является ребро 3-1. Предположим, что мы не включили его.

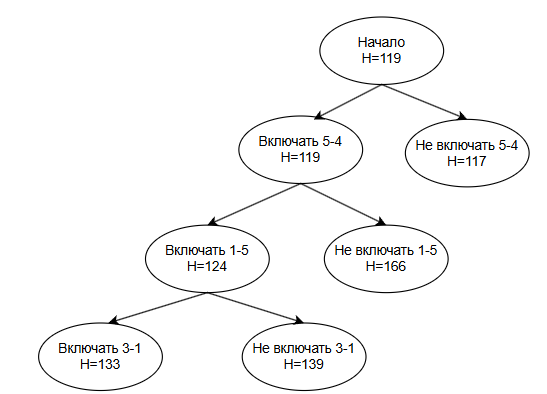
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| i j | 1 | 2 | 3 | di |
| 2 | 0 | ∞ | 0 | 0 |
| 3 | ∞ | 10 | ∞ | 10 |
| 4 | 0 | 5 | 4 | 0 |
| dj | 0 | 5 | 0 | 15 |

В данном случае граница равна H=124+15=139

Если же ребро взять, то:

|  |  |  |  |
| --- | --- | --- | --- |
| i j | 2 | 3 | di |
| 2 | ∞ | 0 | 0 |
| 4 | 5 | 4 | 4 |
| dj | 5 | 0 | 9 |

H = 124+9=133.

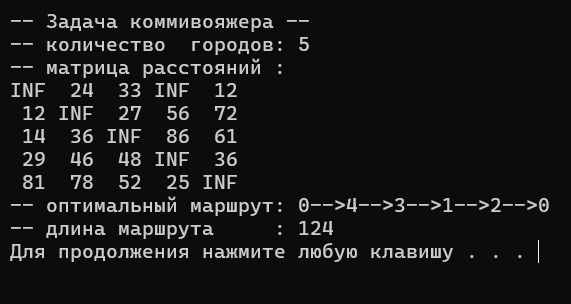


Так как необходимо посетить все вершины, то включаем рёбра 2-3 и 4-2.

В результате получим маршрут 1-5, 5-4, 4-2, 2-3, 3-1. Длина маршрута равна 124.

**Задание 3.** Проверить полученное решение при помощи генератора перестановок и включить копию экрана с решением.

Ниже представлена копия экрана с решением задачи коммивояжера при помощи генератора перестановок.



**Лабораторная работа №4. Динамическое программирование**

**Задание 1**

На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной 200.

Листинг кода генерации строк:

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <Windows.h>  using namespace std;  #define FIRST\_LEN 300  #define SECOND\_LEN 250  char\* GenerateRandomString(int size)  {  char\* str = (char\*)malloc(sizeof(char) \* size);  for (int i = 0; i < size; i++) {  str[i] = rand() % 26 + 'a'; // 26 букв в алфавите  }  return str;  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  srand(time(NULL));  char\* s1 = GenerateRandomString(FIRST\_LEN);  cout << "S1: " << endl;  for (int i = 0; i < FIRST\_LEN; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s1[i];  }  cout << endl << endl; |

Листинг 1 – генерация строк

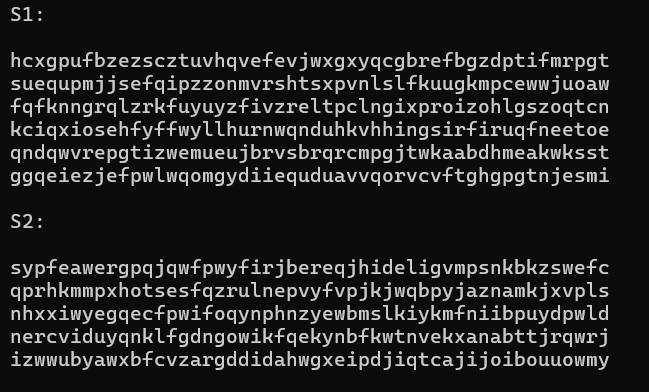
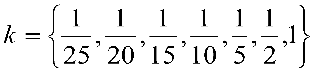


Рисунок 1 – результат работы генератора

**Задание 2**

Необходимо при помощи двух способов вычислить дистанцию Левенштейна для . Код выполнение программы представлен на рисунках ниже.

|  |
| --- |
| #pragma once  #include <tchar.h>  // - Levenshtein.h  // -- дистанции Левенштeйна (динамическое программирование)  int levenshtein(  int lx, // длина слова x  const char x[], // слово длиной lx  int ly, // длина слова y  const char y[] // слово y  );  // -- дистанции Левенштeйна (рекурсия)  int levenshtein\_r(  int lx, // длина строки x  const char x[], // строка длиной lx  int ly, // длина строки y  const char y[] // строка y  ); |

Листинг 2 – Заголовочный файл с прототипами функций

// - Levenshtein.cpp

#include <iomanip>

#include <algorithm>

#include "Levenshtein.h"

#define DD(i,j) d[(i)\*(ly+1)+(j)]

int min3(int x1, int x2, int x3)

{

return std::min(std::min(x1, x2), x3);

}

int levenshtein(int lx, const char x[], int ly, const char y[])

{

int\* d = new int[(lx + 1) \* (ly + 1)];

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

for (int j = 0; j <= ly; j++) DD(0, j) = j;

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

for (int j = 1; j <= ly; j++)

{

DD(i, j) = min3(DD(i - 1, j) + 1, DD(i, j - 1) + 1,

DD(i - 1, j - 1) + (x[i - 1] == y[j - 1] ? 0 : 1));

}

return DD(lx, ly);

}

int levenshtein\_r(

int lx, const char x[],

int ly, const char y[]

)

{

int rc = 0;

if (lx == 0) rc = ly;

else if (ly == 0) rc = lx;

else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0;

else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1;

else rc = min3(

levenshtein\_r(lx - 1, x, ly, y) + 1,

levenshtein\_r(lx, x, ly - 1, y) + 1,

levenshtein\_r(lx - 1, x, ly - 1, y) + (x[lx - 1] == y[ly - 1] ? 0 : 1)

);

return rc;

};

Листинг 3 – Реализация функций

// --- main

// вычисление дистанции (расстояния) Левенштейна

#include <algorithm>

#include <iostream>

#include <ctime>

#include <iomanip>

#include "Levenshtein.h"

#include <Windows.h>

using namespace std;

char\* GenerateRandomString(int size)

{

char\* str = (char\*)malloc(sizeof(char) \* size);

for (int i = 0; i < size; i++) {

str[i] = rand() % 26 + 'a'; // 26 букв в алфавите

}

return str;

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

char\* s1 = GenerateRandomString(300);

cout << "S1: " << endl;

for (int i = 0; i < 300; i++) {

if (i % 50 == 0)

{

cout << "\n";

}

cout << s1[i];

}

cout << endl << endl;

srand(time(NULL) + 1);

char\* s2 = GenerateRandomString(200);

cout << "S2: " << endl;

for (int i = 0; i < 200; i++) {

if (i % 50 == 0)

{

cout << "\n";

}

cout << s2[i];

}

cout << endl << endl;

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

int lx = sizeof(s1);

int ly = sizeof(s2);

int s1\_size[]{ 300 / 25, 300 / 20, 300 / 15, 300 / 10, 300 / 5, 300 / 2, 300 };

int s2\_size[]{ 200 / 25, 200 / 20, 200 / 15, 200 / 10, 200 / 5, 200 / 2, 200 };

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

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;

}

Листинг 4 – функция main

Результат выполнения программы представлен на рисунке 4.

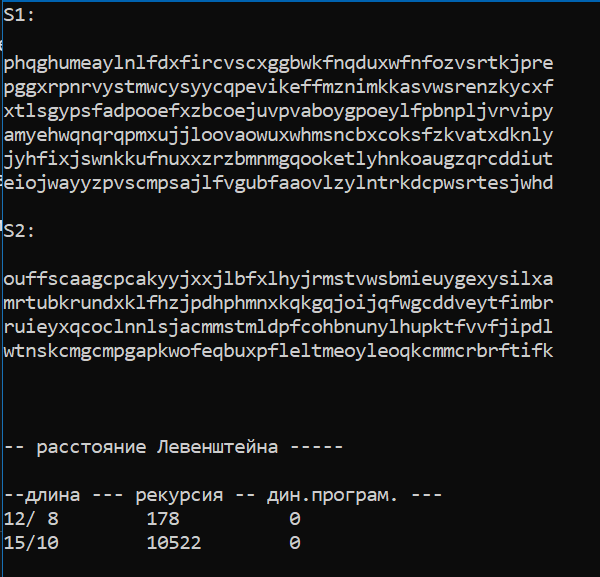


Рисунок 2 – Результат выполнения программы

**Задание 3**

Необходимо провести сравнительный анализ по затраченному на выполнение программы времени. На графике, который изображён на рисунке 5, нетрудно заметить, что использование динамического алгоритма в несколько раз эффективнее по затраченному времени, нежели рекурсивное выполнение.



Рисунок 3 – График зависимости выполнения

**Задание 4**

Реализовать в соответствии с вариантом (рисунок 6) пример вычисления дистанции Левенштейна вручную, при помощи рекурсивного алгоритма. Ниже представлен алгоритм решения последовательно:



Рисунок 4 – Вариант для вычисления дистанции Левенштейна



= 5.

= 4.



= 4.

= 3.



= 3.

= 2.



= 2.

= 1.



= 3.

= 2.



= 2.

= 1.



= 1.

= 1.

= 0.

1. L(“Э”, “Х”) = min(2,2,1) = 1
2. L(“Эх”, “Х”) = min(2,3,2) = 2
3. L(“Эхо”, “Х”) = min(3,4,3) = 3
4. L(“Э”, “Хо”) = min(3,2,2) = 2
5. L(“Эх”, “Хо”) = min(3,3,2) = 2
6. L(“Эхо”, “Хо”) = min(3,4,2) = 2
7. L(“Э”, “Хор”) = min(4,3,3) = 3
8. L(“Эх”, “Хор”) = min(4,3,3) = 3
9. L(“Эхо”, “Хор”) = min(4,3,3) = 3
10. L(“Э”, “Хоре”) = min(5,4,4) = 4
11. L(“Э”, “Хорек”) = min(6,5,5) = 5
12. L(“Эх”, “Хоре”) = min(5,4,4) = 4
13. L(“Эхо”, “Хоре”) = min(5,5,4) = 4
14. L(“Эх”, “Хорек”) = min(6,5,5) = 5
15. L(“Эхо”, “Хорек”) = min(6,5,5) = 5

**Задание 5**

Выполнить 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 взять в соответствии с вариантом. Объяснить в отчете принцип расставления скобок по итоговой матрице + код + копии экрана.

Листинги кода представлены ниже:

// --- MultyMatrix.h

// расстановка скобок

#pragma once

// расстановка скобок при умножении матриц

// функции возвращают минимальное количество операций умножения

#define OPTIMALM\_PARM(x) ((int\*)x) // для представления 2мерного массива

int OptimalM( // рекурсия

int i, // [in] номер первой матрицы

int j, // [in] номер последней матрицы

int n, // [in] количество матриц

const int c[], // [in] массив размерностей

int\* s // [out] результат: позиции скобок

);

int OptimalMD( // динамическое программирование

int n, // [in] количество матриц

const int c[], // [in] массив размерностей

int\* s // [out] результат: позиции скобок

);

Листинг 5 – MultiMatrix.h

// --- MultiMatrix.cpp

// расстановка скобок (рекурсия)

#include <memory.h>

#include "MultiMatrix.h"

#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, 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

};

// --- MultyMatrix.cpp (продолжение)

// 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

};

Листинг 6 - MultiMatrix.cpp

// --- main

// расстановка скобок

#include <cmath>

#include <memory.h>

#include <chrono>

#include <iostream>

#include "MultiMatrix.h" // умножение матриц

#define N 6

int main()

{

int Mc[N + 1] = { 8,11,19,22,29,39,50 }, Ms[N][N], r = 0, rd = 0;

auto t1 = std::chrono::system\_clock::now();

memset(Ms, 0, sizeof(int) \* N \* N);

r = OptimalM(1, N, N, Mc, OPTIMALM\_PARM(Ms));

auto t2 = std::chrono::system\_clock::now();

setlocale(LC\_ALL, "rus");

std::cout << std::endl;

std::cout << std::endl << "-- расстановка скобок (рекурсивное решение) "

<< std::endl;

std::cout << std::endl << "размерности матриц: ";

for (int i = 1; i <= N; i++) std::cout << "(" << Mc[i - 1] << "," << Mc[i] << ") ";

std::cout << std::endl << "минимальное количество операций умножения: " << r;

std::cout << std::endl << std::endl << "матрица S" << std::endl;

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

{

std::cout << std::endl;

for (int j = 0; j < N; j++) std::cout << Ms[i][j] << " ";

}

std::cout << std::endl;

std::chrono::duration<double> elapsed = t2 - t1;

std::cout << "Время выполнения: " << elapsed.count() \* 1000000 << "мкс." << std::endl;

t1 = std::chrono::system\_clock::now();

memset(Ms, 0, sizeof(int) \* N \* N);

rd = OptimalMD(N, Mc, OPTIMALM\_PARM(Ms));

t2 = std::chrono::system\_clock::now();

std::cout << std::endl

<< "-- расстановка скобок (динамичеое программирование) " << std::endl;

std::cout << std::endl << "размерности матриц: ";

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

std::cout << "(" << Mc[i - 1] << "," << Mc[i] << ") ";

std::cout << std::endl << "минимальное количество операций умножения: "

<< rd;

std::cout << std::endl << std::endl << "матрица S" << std::endl;

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

{

std::cout << std::endl;

for (int j = 0; j < N; j++) std::cout << Ms[i][j] << " ";

}

std::cout << std::endl << std::endl;

elapsed = t2 - t1;

std::cout << "Время выполнения: " << elapsed.count() \* 1000000 << "мкс." << std::endl;

system("pause");

return 0;

}

Листинг 7 – main.cpp

Выполнив сравнительный анализ времени затраченного на решение задачи об оптимальной расстановке скобок при умножении нескольких матриц для двух методов решения, можно заметить, что динамический алгоритм работает быстрее, однако, в данном ситуации оба метода затрачивают для выполнения минимально времени. Результаты представлены на рисунке 7:

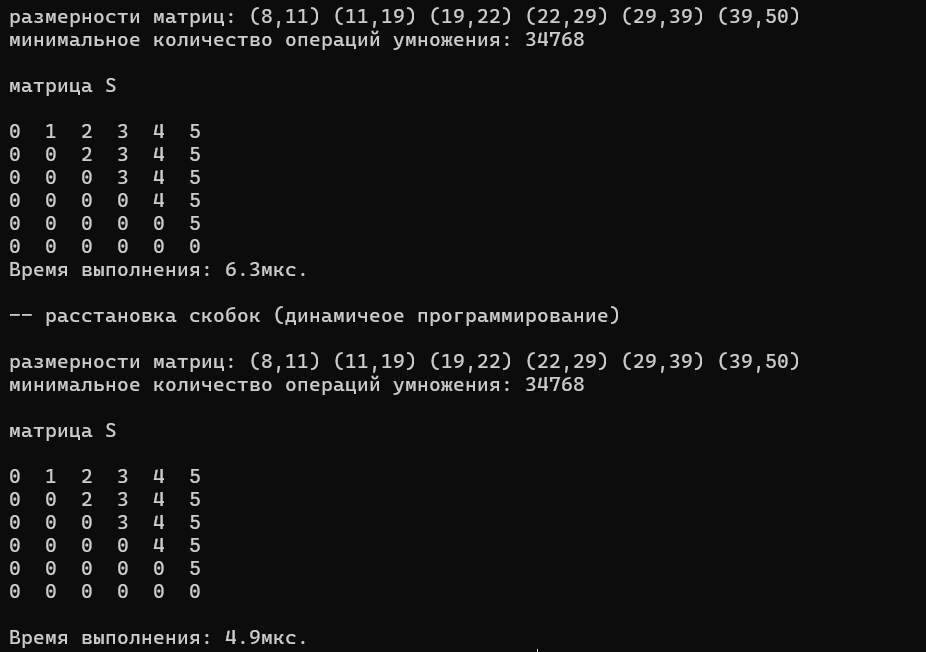


Рисунок 5 – выполнение программы по оптимальной расстановке скобок при умножении нескольких матриц

По данной матрице мы можем оптимально расставить скобки. Для умножения шести входных матриц достаточно выполнить m[1,6]. Оптимальная последовательность умножений имеет вид:

A16 = (s[1,6] = 5) = A15 \* A6 =

(s[1,5] = 4) = (A13 \* A45)\*A66 =

(s[1,3] = 2, s[4,5] = 4) = (A12 \* A33 \* A44 \* A55)\*A66 =

((A1 \* A2 )\* A3 \* A4 \* A5) \* A6

Данная расстановка скобок никак не влияет на последовательность перемножения матриц, значит матрицы уже были расставлены оптимально для наименьшего количества операций умножения.

**Вывод:** динамический подход к решению задач позволяет выполнять их значительно быстрее, чем рекурсивный, особенно это будет заметно при решении задач с большим объёмом информации. Так же, я сделал следующие выводы:

1. Динамическое программирование - это мощный метод решения задач, позволяющий эффективно решать широкий спектр задач, которые не могут быть решены простыми алгоритмами.

2. Метод динамического программирования заключается в разбиении сложной задачи на более простые подзадачи, решение которых затем комбинируется в общее решение задачи.

3. Решение задач методом динамического программирования отличается высокой скоростью выполнения благодаря использованию кэширования вычислений и быстрой обработке данных.

4. В ходе выполнения лабораторной работы был исследован один из наиболее популярных методов динамического программирования - расстояние Левенштейна. Результаты экспериментов показали, что данная техника может быть очень эффективна для решения задач, связанных с обработкой текстовых данных.

5. Решение задач методом динамического программирования может быть осуществлено не только с помощью программирования на языке C++, но и на других языках, таких как Python или Java.

**Лабораторная работа №5. Транспортная задача**

Приобретение навыков решения открытой транспортной задачи

Необходимо составить таблицу (исходя из своего варианта) для решения транспортной задачи. Таблица представлена на таблице 1.

Таблица 1 – Исходная таблица для решения транспортной задачи

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 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** |
| ПОТРЕБНОСТИ | **155** | **119** | **143** | **205** | **107** | **175** |  |

**Ход решения:**

* Проверить, открытая задача или закрытая;
* Составить опорный план;
* Применить метод потенциалов;

Решим задачу. Ниже будет приведен алгоритм решения задачи для таблицы 1:

Проверим необходимое и достаточное условие разрешимости задачи.  
∑a = 180 + 125 + 162 + 171 + 112 = 750  
∑b = 155 + 119 + 143 + 205 + 107 + 175 = 904  
Как видно, суммарная потребность груза в пунктах назначения превышает запасы груза на базах. Следовательно, модель исходной транспортной задачи является открытой. Чтобы получить закрытую модель, введем дополнительную (фиктивную) базу с запасом груза, равным 154 (750—904). Тарифы перевозки единицы груза из базы ко всем потребителям полагаем равны нулю. Занесем исходные данные в распределительную таблицу.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 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** |
| ПОТРЕБНОСТИ | **155** | **119** | **143** | **205** | **107** | **175** | **904** |

**Этап I. Поиск первого опорного плана**.  
1. Используя *метод наименьшей стоимости*, построим первый опорный план транспортной задачи. Суть метода заключается в том, что из всей таблицы стоимостей выбирают наименьшую, и в клетку, которая ей соответствует, помещают 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 и строку и столбец, если израсходованы запасы поставщика и удовлетворены потребности потребителя.  
Из оставшейся части таблицы стоимостей снова выбирают наименьшую стоимость, и процесс распределения запасов продолжают, пока все запасы не будут распределены, а потребности удовлетворены.  
Искомый элемент равен c22=12. Для этого элемента запасы равны 125, потребности 119. Поскольку минимальным является 119, то вычитаем его.  
x22 = min(125,119) = 119.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **24** | **x** | **18** | **15** | **23** | **13** | **180** |
| 2 | **22** | **12 | 119** | **20** | **17** | **19** | **25** | **6** |
| 3 | **13** | **x** | **23** | **20** | **14** | **23** | **162** |
| 4 | **16** | **x** | **22** | **15** | **25** | **14** | **171** |
| 5 | **15** | **x** | **21** | **12** | **22** | **16** | **112** |
| 6 | **0** | **x** | **0** | **0** | **0** | **0** | **154** |
| ПОТРЕБНОСТИ | **155** | **0** | **143** | **205** | **107** | **175** | **904** |

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **24** | **x** | **18** | **15** | **23** | **13** | **180** |
| 2 | **22** | **12 | 119** | **20** | **17** | **19** | **25** | **6** |
| 3 | **13** | **x** | **23** | **20** | **14** | **23** | **162** |
| 4 | **16** | **x** | **22** | **15** | **25** | **14** | **171** |
| 5 | **x** | **x** | **x** | **12 | 112** | **x** | **x** | **0** |
| 6 | **0** | **x** | **0** | **0** | **0** | **0** | **154** |
| ПОТРЕБНОСТИ | **155** | **0** | **143** | **93** | **107** | **175** | **904** |

Искомый элемент равен c16=13. Для этого элемента запасы равны 180, потребности 175. Поскольку минимальным является 170, то вычитаем его.  
x16 = min(180,175) = 175.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **24** | **x** | **18** | **15** | **23** | **13 | 175** | **5** |
| 2 | **22** | **12 | 119** | **20** | **17** | **19** | **x** | **6** |
| 3 | **13** | **x** | **23** | **20** | **14** | **x** | **162** |
| 4 | **16** | **x** | **22** | **15** | **25** | **x** | **171** |
| 5 | **x** | **x** | **x** | **12 | 112** | **x** | **x** | **0** |
| 6 | **0** | **x** | **0** | **0** | **0** | **x** | **154** |
| ПОТРЕБНОСТИ | **155** | **0** | **143** | **93** | **107** | **0** | **904** |

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **x** | **x** | **18** | **15** | **23** | **13 | 175** | **5** |
| 2 | **x** | **12 | 119** | **20** | **17** | **19** | **x** | **6** |
| 3 | **13 | 155** | **x** | **23** | **20** | **14** | **x** | **7** |
| 4 | **x** | **x** | **22** | **15** | **25** | **x** | **171** |
| 5 | **x** | **x** | **x** | **12 | 112** | **x** | **x** | **0** |
| 6 | **x** | **x** | **0** | **0** | **0** | **x** | **154** |
| ПОТРЕБНОСТИ | **0** | **0** | **143** | **93** | **107** | **0** | **904** |

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **x** | **x** | **18** | **15** | **23** | **13 | 175** | **5** |
| 2 | **x** | **12 | 119** | **20** | **17** | **19** | **x** | **6** |
| 3 | **13 | 155** | **x** | **x** | **x** | **14 | 7** | **x** | **0** |
| 4 | **x** | **x** | **22** | **15** | **25** | **x** | **171** |
| 5 | **x** | **x** | **x** | **12 | 112** | **x** | **x** | **0** |
| 6 | **x** | **x** | **0** | **0** | **0** | **x** | **154** |
| ПОТРЕБНОСТИ | **0** | **0** | **143** | **93** | **100** | **0** | **904** |

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **x** | **x** | **x** | **15 | 5** | **x** | **13 | 175** | **0** |
| 2 | **x** | **12 | 119** | **20** | **17** | **19** | **x** | **6** |
| 3 | **13 | 155** | **x** | **x** | **x** | **14 | 7** | **x** | **0** |
| 4 | **x** | **x** | **22** | **15** | **25** | **x** | **171** |
| 5 | **x** | **x** | **x** | **12 | 112** | **x** | **x** | **0** |
| 6 | **x** | **x** | **0** | **0** | **0** | **x** | **154** |
| ПОТРЕБНОСТИ | **0** | **0** | **143** | **88** | **100** | **0** | **904** |

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **x** | **x** | **18** | **15 | 5** | **23** | **13 | 175** | **0** |
| 2 | **x** | **12 | 119** | **20** | **17** | **19** | **x** | **6** |
| 3 | **13 | 155** | **x** | **x** | **x** | **14 | 7** | **x** | **0** |
| 4 | **x** | **x** | **22** | **15 | 88** | **25** | **x** | **83** |
| 5 | **x** | **x** | **x** | **12 | 112** | **x** | **x** | **0** |
| 6 | **x** | **x** | **0** | **x** | **0** | **x** | **154** |
| ПОТРЕБНОСТИ | **0** | **0** | **143** | **0** | **100** | **0** | **904** |

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **x** | **x** | **x** | **15 | 5** | **x** | **13 | 175** | **0** |
| 2 | **x** | **12 | 119** | **x** | **x** | **19 | 6** | **x** | **0** |
| 3 | **13 | 155** | **x** | **x** | **x** | **14 | 7** | **x** | **0** |
| 4 | **x** | **x** | **22** | **15 | 88** | **25** | **x** | **83** |
| 5 | **x** | **x** | **x** | **12 | 112** | **x** | **x** | **0** |
| 6 | **x** | **x** | **0** | **0** | **0** | **x** | **154** |
| ПОТРЕБНОСТИ | **0** | **0** | **143** | **0** | **94** | **0** | **904** |

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **x** | **x** | **x** | **15 | 5** | **x** | **13 | 175** | **0** |
| 2 | **x** | **12 | 119** | **x** | **x** | **19 | 6** | **x** | **0** |
| 3 | **13 | 155** | **x** | **x** | **x** | **14 | 7** | **x** | **0** |
| 4 | **x** | **x** | **22 | 83** | **15 | 88** | **x** | **x** | **0** |
| 5 | **x** | **x** | **x** | **12 | 112** | **x** | **x** | **0** |
| 6 | **x** | **x** | **0** | **x** | **0** | **x** | **154** |
| ПОТРЕБНОСТИ | **0** | **0** | **60** | **0** | **94** | **0** | **904** |

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **x** | **x** | **x** | **15 | 5** | **x** | **13 | 175** | **0** |
| 2 | **x** | **12 | 119** | **x** | **x** | **19 | 6** | **x** | **0** |
| 3 | **13 | 155** | **x** | **x** | **x** | **14 | 7** | **x** | **0** |
| 4 | **x** | **x** | **22 | 83** | **15 | 88** | **x** | **x** | **0** |
| 5 | **x** | **x** | **x** | **12 | 112** | **x** | **x** | **0** |
| 6 | **x** | **x** | **0 | 60** | **x** | **0** | **x** | **94** |
| ПОТРЕБНОСТИ | **0** | **0** | **0** | **0** | **94** | **0** | **904** |

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **24** | **14** | **18** | **15 | 5** | **23** | **13 | 175** | **0** |
| 2 | **22** | **12 | 119** | **20** | **17** | **19 | 6** | **25** | **0** |
| 3 | **13 | 155** | **17** | **23** | **20** | **14 | 7** | **23** | **0** |
| 4 | **16** | **22** | **22 | 83** | **15 | 88** | **25** | **14** | **0** |
| 5 | **15** | **23** | **21** | **12 | 112** | **22** | **16** | **0** |
| 6 | **0** | **0** | **0 | 60** | **0** | **0 | 94** | **0** | **0** |
| ПОТРЕБНОСТИ | **0** | **0** | **0** | **0** | **0** | **0** | **904** |

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.  
2. Подсчитаем число занятых клеток таблицы, 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\*110 + 19\*6 + 13\*155 + 14\*7 + 22\*83 + 15\*88 + 12\*112 + 0\*60 + 0\*94 = 10387

**Этап II. Улучшение опорного плана** **методом потенциалов**  
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 +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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | v1 = 21 | v2 = 15 | v3 = 22 | v4 = 15 | v5 = 22 | v6 = 13 |
| u1 = 0 | **24** | **14** | **18** | **15 | 5** | **23** | **13 | 175** |
| u2 = -3 | **22** | **12 | 119** | **20** | **17** | **19 | 6** | **25** |
| u3 = -8 | **13 | 155** | **17** | **23** | **20** | **14 | 7** | **23** |
| u4 = 0 | **16** | **22** | **22 | 83** | **15 | 88** | **25** | **14** |
| u5 = -3 | **15** | **23** | **21** | **12 | 112** | **22** | **16** |
| u6 = -22 | **0** | **0** | **0 | 60** | **0** | **0 | 94** | **0** |

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): 11  
Для этого в перспективную клетку (4;1) поставим знак «+», а в остальных вершинах многоугольника чередующиеся знаки «-», «+», «-».

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **24** | **14** | **18** | **15 | 5** | **23** | **13 | 175** | **180** |
| 2 | **22** | **12 | 119** | **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** |
| ПОТРЕБНОСТИ | **155** | **119** | **143** | **205** | **107** | **175** | **0** |

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **24** | **14** | **18** | **15 | 5** | **23** | **13 | 175** | **180** |
| 2 | **22** | **12 | 119** | **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** |
| ПОТРЕБНОСТИ | **155** | **119** | **143** | **205** | **107** | **175** | **0** |

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 = 10; 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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | v1 = 16 | v1 = 10 | v1 = 17 | v1 = 15 | v1 = 17 | v1 = 13 |
| u1 = 0 | **24** | **14** | **18** | **15 | 5** | **23** | **13 | 175** |
| u2  = 2 | **22** | **12 | 119** | **20** | **17** | **19 | 6** | **25** |
| u3  = -3 | **13 | 72** | **17** | **23** | **20** | **14 | 90** | **23** |
| u4  = 0 | **16 | 83** | **22** | **22** | **15 | 88** | **25** | **14** |
| u5  = -3 | **15** | **23** | **21** | **12 | 112** | **22** | **16** |
| u6  = -17 | **0** | **0** | **0 | 143** | **0** | **0 | 11** | **0** |

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 к 4-у потребителю (5 ед.), к 6-у потребителю (175 ед.)  
Из 2-го склада необходимо груз направить к 2-у потребителю (119 ед.), к 5-у потребителю (6 ед.)  
Из 3-го склада необходимо груз направить к 1-у потребителю (72 ед.), к 5-у потребителю (90 ед.)  
Из 4-го склада необходимо груз направить к 1-у потребителю (83 ед.), к 4-у потребителю (88 ед.)  
Из 5-го склада необходимо весь груз направить к 4-у потребителю.  
Потребность 3-го потребителя остается неудовлетворенной на 143 ед.  
Оптимальный план является вырожденным, так как базисная переменная x63=0.  
Потребность 5-го потребителя остается неудовлетворенной на 11 ед.  
Оптимальный план является вырожденным, так как базисная переменная x65=0.

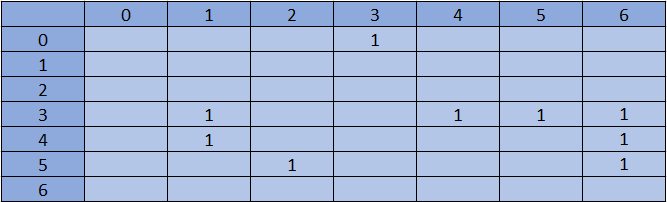
**Вывод:** в ходе выполнения данной лабораторной работы были приобретены навыки решения открытой транспортной задачи. Для решения задачи был применен метод наименьшей стоимости и метод потенциалов, которые позволяют определить оптимальный план перевозок при минимальных затратах. Были проведены расчеты с использованием данных методов, что позволило получить оптимальный план перевозок.

**Лабораторная работа №6. Оптимизационные задачи на графах**

**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:

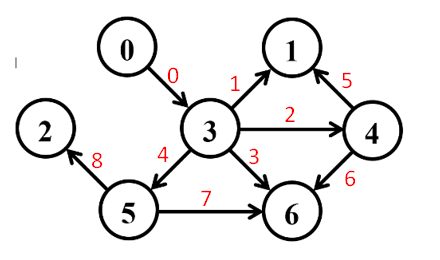


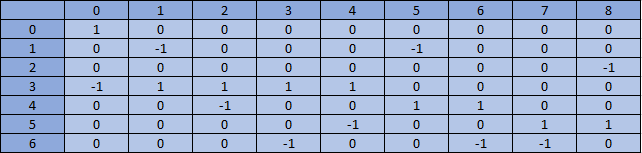
Список смежных вершин:

0 {3} 1 {-} 2 {-} 3 {1, 4, 5, 6} 4 {1, 6}

5 {2, 6}

Матрица инцидентности





***Задание 2.*** Осуществить алгоритмы поиска в ширину и глубину, а также алгоритма топологической сортировки аналогично примерам, рассмотренным на лекциях. Оформить отчет, включив в него **каждый** шаг выполнения алгоритмов.

***1.Поиск в ширину***



1.Посещённые вершины: {}

Очередь: {0}

Текущая вершина: {}

2.Посещённые вершины: {0}

Очередь: {}

Текущая вершина: {0}

3.Посещённые вершины: {0}

Очередь: {3}

Текущая вершина: {0}

4.Посещённые вершины: {0, 3}

Очередь: {1, 4, 5, 6}

Текущая вершина: {3}

5.Посещённые вершины: {0, 3, 1}

Очередь: {4, 5, 6}

Текущая вершина: {1}

6.Посещённые вершины: {0, 3, 1, 4}

Очередь: {5, 6}

Текущая вершина: {4}

7.Посещённые вершины: {0, 3, 1, 4, 5}

Очередь: {6, 2}

Текущая вершина: {5}

8.Посещённые вершины: {0, 3, 1, 4, 5, 6}

Очередь: {2}

Текущая вершина: {6}

9.Посещённые вершины: {0, 3, 1, 4, 5, 6, 2}

Очередь: {}

Текущая вершина: {2}

***2.Поиск в глубину***



1.Посещённые вершины: {}

Стек: {0}

Текущая вершина: {}

2.Посещённые вершины: {0}

Стек: {3}

Текущая вершина: {0}

3.Посещённые вершины: {0, 3}

Стек: {6, 5, 4, 1}

Текущая вершина: {3}

4.Посещённые вершины: {0, 3, 6}

Стек: {5, 4, 1}

Текущая вершина: {6}

5.Посещённые вершины: {0, 3, 6, 5}

Стек: {2, 4, 1}

Текущая вершина: {5}

6.Посещённые вершины: {0, 3, 6, 5, 2}

Стек: {4, 1}

Текущая вершина: {2}

7.Посещённые вершины: {0, 3, 6, 5, 2, 4}

Стек: {1}

Текущая вершина: {4}

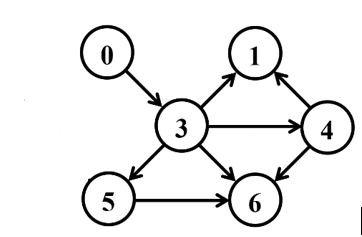
8.Посещённые вершины: {0, 3, 6, 5, 2, 4, 1}

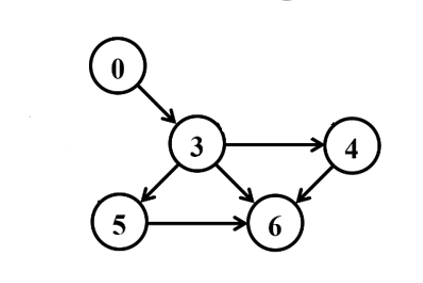
Стек: {}

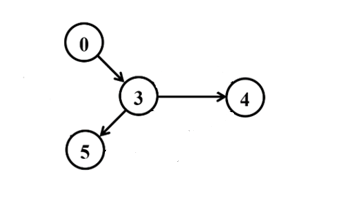
Текущая вершина: {1}

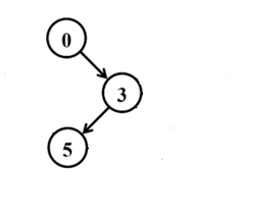
***3.Топологическая сортировка:***

2

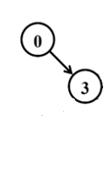
1

6

4



5



3

0

**Результат: 2-1-6-4-5-3-0**

***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.

Заголовочный файл для структур представления матричным и списковым способом представлен на рисунке 1:

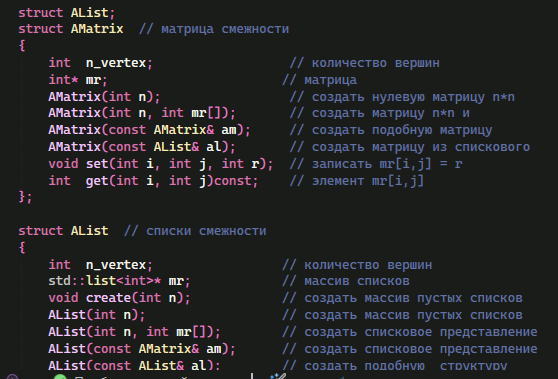


Рисунок 1 – struct of AList and AMatrix

В листинге 1 демонстрируется реализация структур AMatrix и AList:

#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;

};

};

Листинг 1 – Graph.cpp

В листинге 2 показана реализация функции BFS:

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;

}

Листинг 2 – BFS.cpp

Результат выполнения программы представлен на рисунке 2:

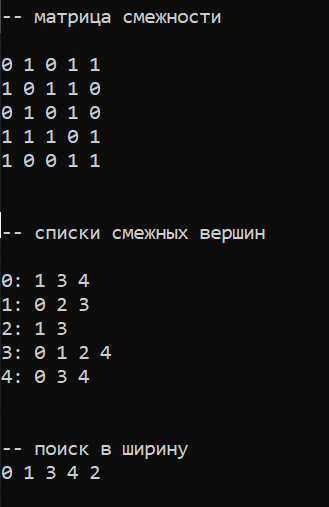


Рисунок 2 – результат выполнения программы

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

Заголовочный файл функции DFC.h:

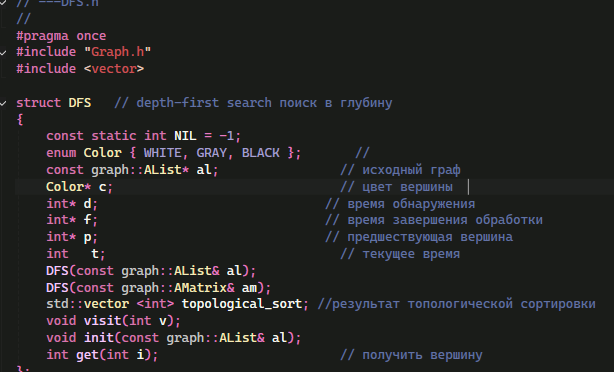


Рисунок 3 – DFC.h

Реализация заголовка представлена в листинге 3:

#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;

};

Листинг 3 - DFC.cpp

Результат выполнения программы представлен на рисунке 4:

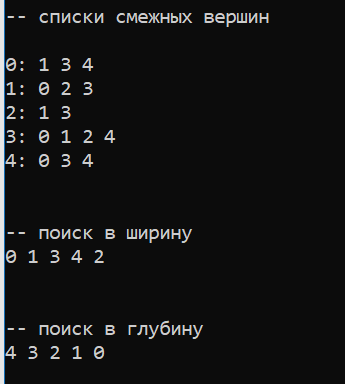


Рисунок 4 – результат выполнения программы

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

Реализация кода представлена в предыдущем задании, а результат выполнения – на рисунке 6.

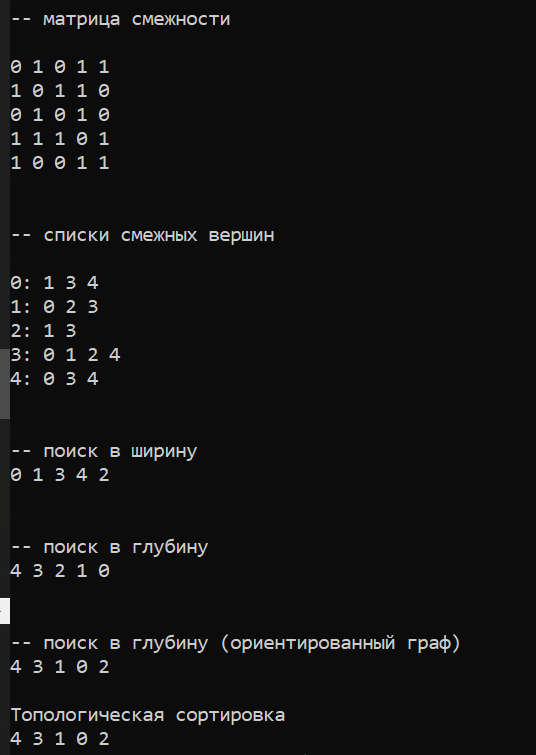
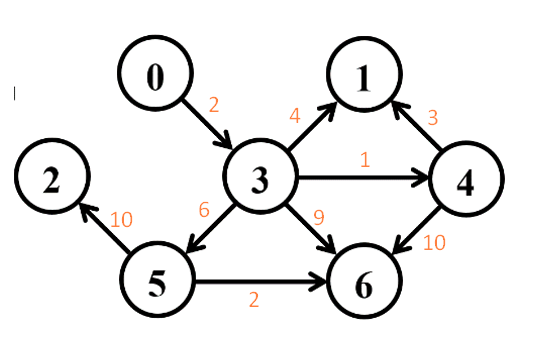
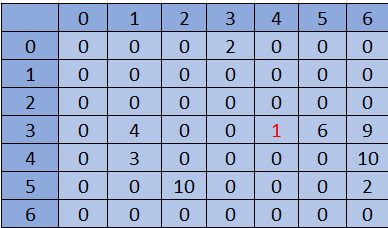


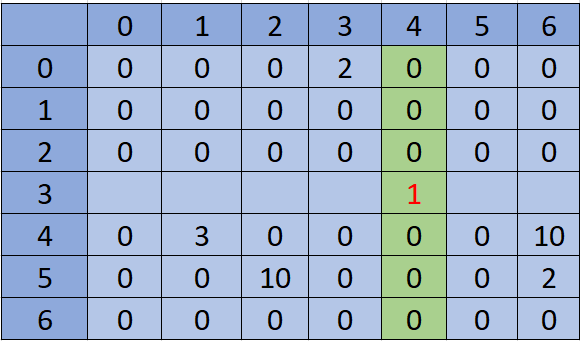
Рисунок 6 – результат выполнения программы

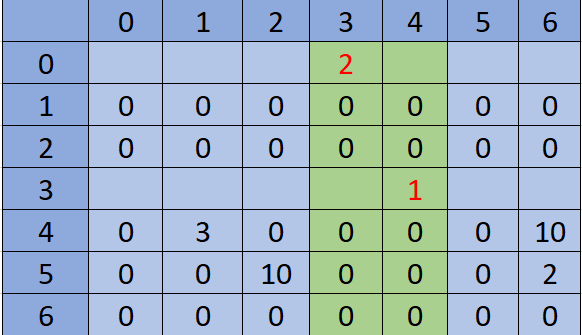
***Задание 6.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Прима. Шаги построения отразить в отчете.

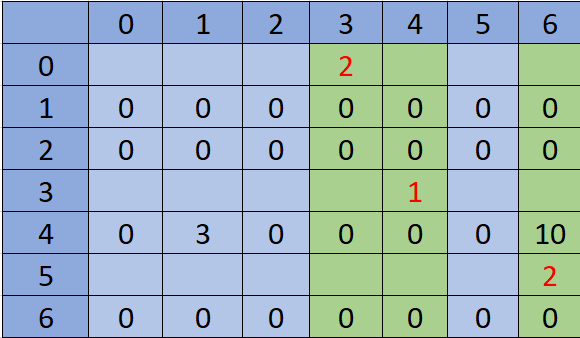


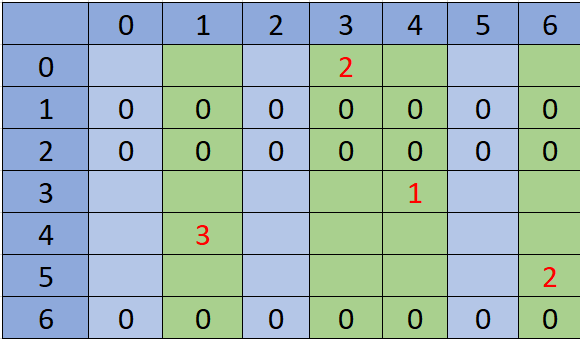
(Можно решать матрицей, вычёркивая строки и выделяя столбцы, можно через граф).

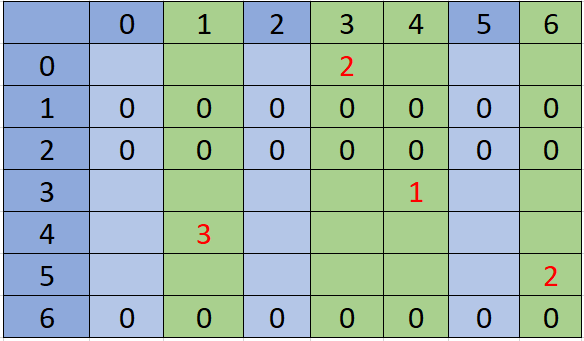












Т.е. 0-1-3-4-6

Теперь графически решим:

Для решения по алгоритму Прима необходимо выбрать произвольную начальную вершину и добавлять к остовному дереву ребра с наименьшей стоимостью, пока все вершины не будут включены в дерево.

Можно начать с вершины 0. Перечислим все ребра, связанные с этой вершиной и их стоимость:

* 0->3 (стоимость 2)

Добавляем это ребро к остовному дереву.

Теперь мы имеем две вершины в остовном дереве: 0 и 3. Необходимо выбрать следующее ребро с наименьшей стоимостью, которое связывает вершину из дерева и вершину, не принадлежащую дереву. Перечислим все такие ребра и их стоимость:

* 3->4 (стоимость 1)
* 3->5 (стоимость 6)
* 3->6 (стоимость 9)
* 3->1 (стоимость 4)

Выбираем ребро 3->4, так как оно имеет наименьшую стоимость. Добавляем вершину 4 в остовное дерево.

Теперь в дереве находятся вершины 0, 3 и 4. Необходимо выбрать следующее ребро с наименьшей стоимостью, которое связывает вершину из дерева и вершину, не принадлежащую дереву. Перечислим все такие ребра и их стоимость:

* 4->1 (стоимость 3)
* 4->6 (стоимость 10)
* 3->5 (стоимость 6)
* 3->6 (стоимость 9)
* 5->6 (стоимость 2)

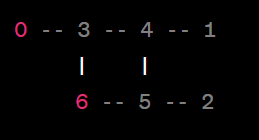
Выбираем ребро 5->6, так как оно имеет наименьшую стоимость. Добавляем вершину 6 в остовное дерево.

Теперь в дереве находятся вершины 0, 3, 4 и 6. Необходимо выбрать следующее ребро с наименьшей стоимостью, которое связывает вершину из дерева и вершину, не принадлежащую дереву. Перечислим все такие ребра и их стоимость:

* 5->2 (стоимость 10)
* 3->1 (стоимость 4)

Выбираем ребро 3->1, так как оно имеет наименьшую стоимость. Теперь в дереве находятся все вершины: 0, 3, 4, 6 и 1. Нет больше ребер, связывающих вершину из дерева и вершину, не принадлежащую дереву, поэтому алгоритм завершен.

Таким образом, остовное дерево, построенное по алгоритму Прима, имеет следующий вид:



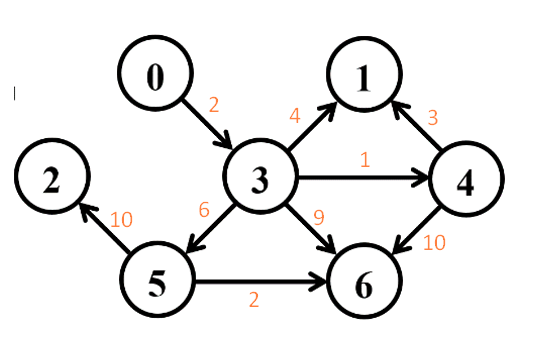
Общая стоимость дерева равна 16.

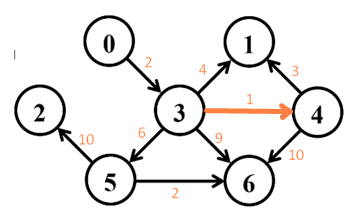
\*Остовное дерево - это подграф, содержащий все вершины исходного графа и являющийся деревом (т.е. связным графом без циклов).

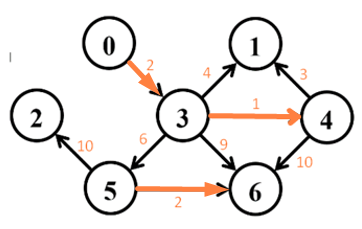
Однако, если вам нужно пройти все вершины графа, то минимальное остовное дерево может не включать все вершины. Это происходит, когда исходный граф не является связным, то есть имеет несколько компонент связности. В таком случае остовное дерево будет содержать только вершины из одной компоненты связности, а вершины из других компонент связности будут исключены из дерева.

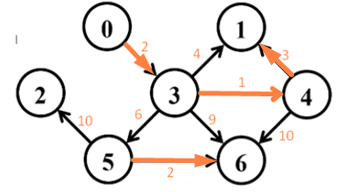
Если нужно пройти все вершины графа, вам необходимо обходить все компоненты связности графа. Для этого можно использовать алгоритм поиска в глубину или алгоритм поиска в ширину, который позволит пройти все вершины графа.

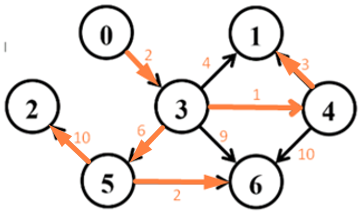
***Задание 7.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Крускала.

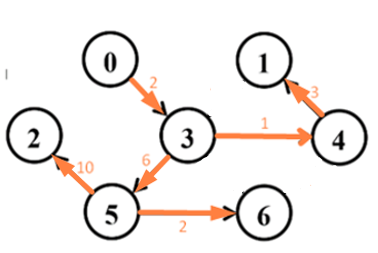












**Лабораторная работа №7. Сетевые модели**

**Цель:** Приобретение навыков сетевого планирования и составления сетевых графиков, приобретение опыта нахождения критического пути.

**Задание 1.** Подумайте и выделите в проекте, согласно вашему варианту не менее 4 этапов работ. Также разбейте полученные этапы на задачи, их количество в совокупности по этапам должно быть не менее 12. Пример оформления задания смотрите в приложении ниже и в лекционном материале по теме.

**Задание 2.** Распределите время, отпущенное на ваш проект согласно вариантам, на выделенные вами этапы. Скорректируйте сформулированные вами задачи, если это необходимо.

В соответствии с вариантом, для выполнения лабораторной работы был выбран проект «Создание компьютерной игры», продолжительность которого составляет 90 дней. На основе этих двух заданий была построена таблица 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Код  операции | Наименование операции | Предшествующие операции | t |
| I. Проектирование игрового сценария | | | |
| Z1 | Исследование рынка компьютерных игр и анализ трендов |  | 5 |
| Z2 | Формулировка концепции игрового сюжета | Z1 | 4 |
| Z3 | Разработка основных персонажей и их характеристик | Z1 | 6 |
| Z4 | Создание общей структуры игрового мира и уровней | Z1 | 6 |
| II. Разработка игрового движка и интерфейса | | | |
| Z5 | Выбор технологической платформы для разработки | Z2, Z3, Z4 | 5 |
| Z6 | Проектирование игрового движка с учетом особенностей сценария | Z1 | 10 |
| Z7 | Создание пользовательского интерфейса и управление игровым процессом | Z6 | 10 |
| Z8 | Тестирование игрового движка на работоспособность | Z5, Z6 | 5 |
| III. Графическое оформление и аудиодизайн | | | |
| Z9 | Разработка графического дизайна игровых объектов и персонажей | Z7 | 7 |
| Z10 | Создание анимаций и спецэффектов | Z7 | 8 |
| Z11 | Звуковое оформление игры: звуковые эффекты, музыкальное сопровождение | Z7 | 7 |
| Z12 | Интеграция графики и звука в игровой процесс | Z11 | 3 |
| IV. Тестирование и оптимизация | | | |
| Z13 | Проведение тестирования игры на различных устройствах и операционных системах | Z9, Z10 | 5 |
| Z14 | Анализ обратной связи тестеров и пользователей | Z13 | 5 |
| Z15 | Оптимизация производительности и устранение ошибок | Z13, Z14 | 3 |
| Z16 | Подготовка к выпуску и дистрибуция игры | Z12, Z15 | 2 |

Таблица 1 – календарное планирование задач проекта

**Задание 3.** Согласно составленному перечню задач и распределённому времени составьте сетевой график вашего проекта. Помните о правилах составления графика и вводите фиктивные операции и операции ожидания если это необходимо. Найдите критический путь в составленном вами сетевом графике и обоснуйте его нахождение. Критический путь может быть меньше, чем время, отведенное на выполнение всех задач. Выделите, какие операции принадлежат критическому пути.

Изобразим календарное планирование в виде ориентированного графа, где вершины представляют собой задачи, а дуги – время выполнения. Каждая вершина ведёт в вершины, которые являются задачами, последующими ей. Данный граф представлен на рисунке 1.

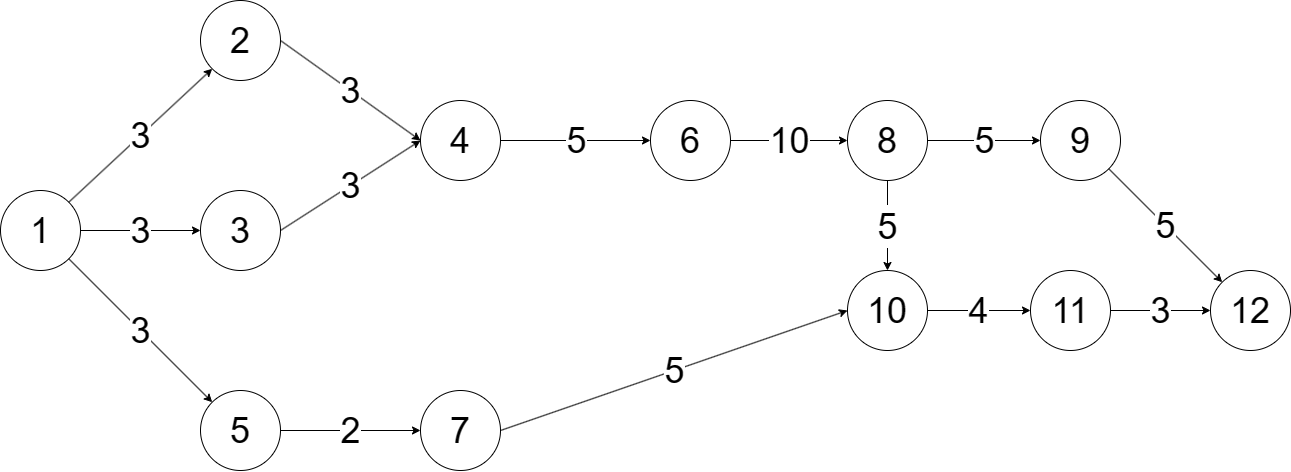


Рисунок 1 – сетевой график для заданного проекта

Для нахождения критического пути необходимо найти ранние сроки начала каждой задачи, а также поздние сроки начала каждой задачи.

Для нахождения ранних сроков начала каждой задачи начнём с первой вершины, ей присвоим значение 0, остальным значение будет высчитываться как максимальный срок начала предшествующих + длина дуги.

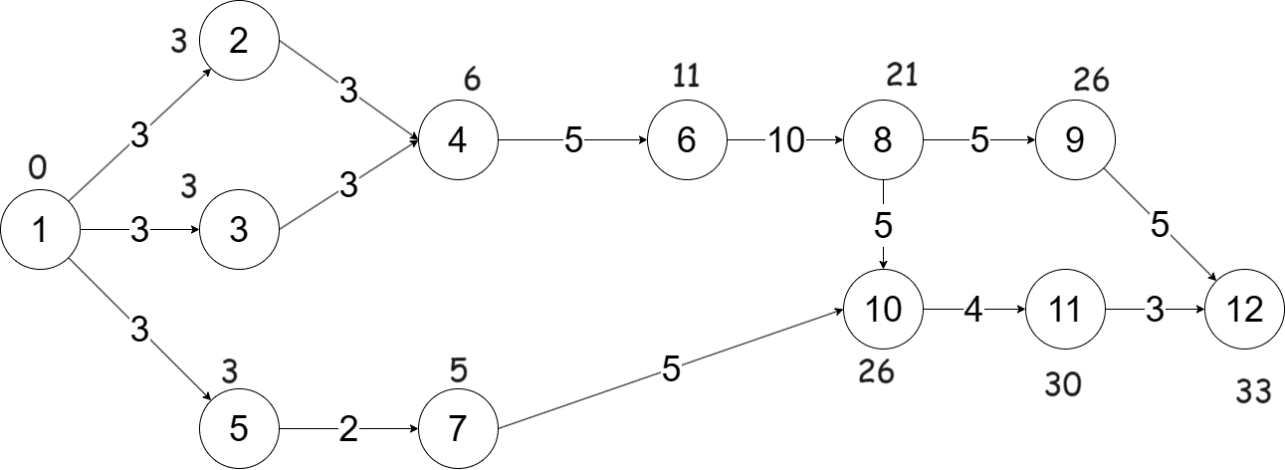


Рисунок 2 – ранние сроки для каждой задачи

Для нахождения поздних сроков начала каждой задачи начнём с последней вершины, ей присвоим значение, равное её раннему сроку начала, остальным значение будет высчитываться как минимальный срок начала последующих - длина дуги.

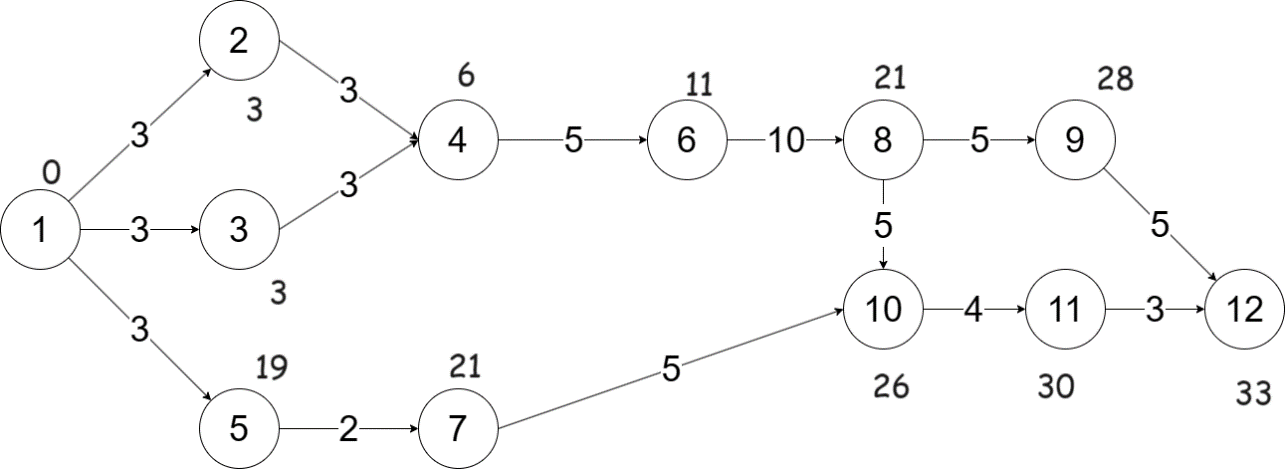


Рисунок 3 – поздние сроки для каждой задачи

Чтобы найти критический путь, необходимо построить путь по вершинам, у которых резерв по времени равен нулю, чтобы найти резерв необходимо от позднего срока отнять ранний, следовательно вершины, у которых они совпадают и будут составлять критический путь. В данном случае критических пути 2, так как резерв вершин 2 и 3 равен, и они являются связующими между вершинами 1 и 4. Критические пути представлены на рисунках 4 и 5.

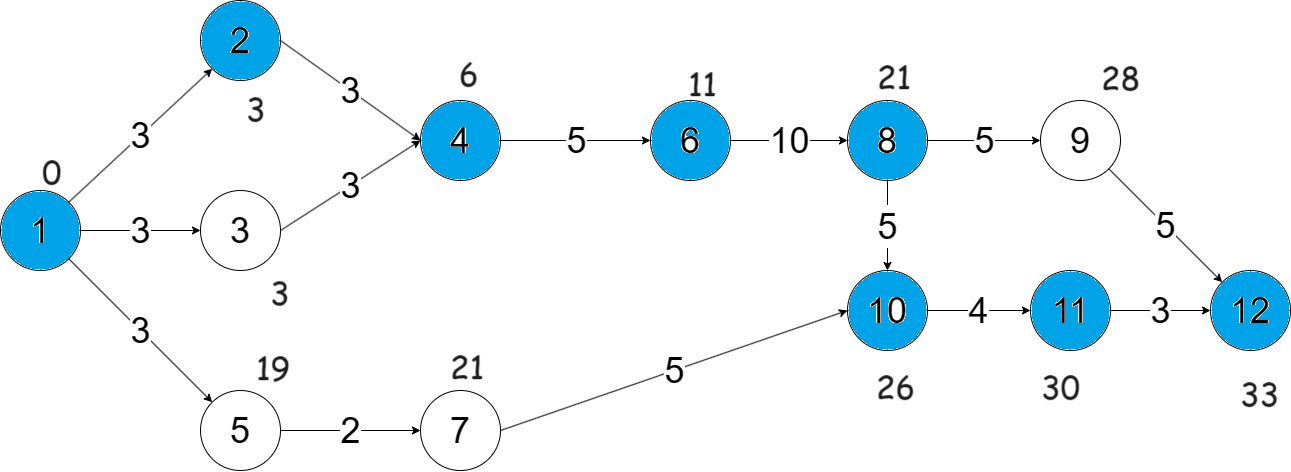


Рисунок 4 – критический путь №1

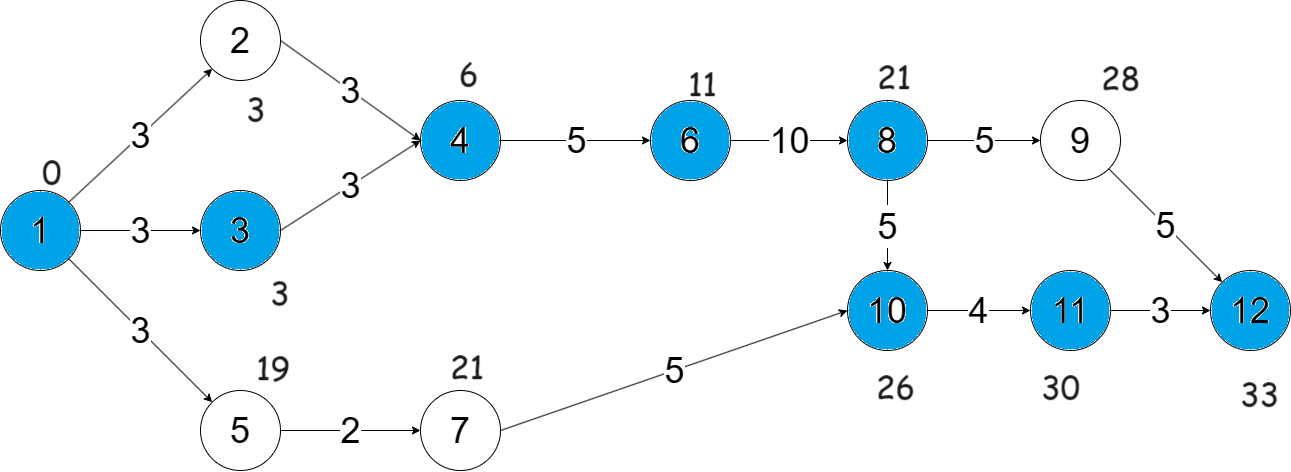


Рисунок 5 – критический путь №2

Из найденных путей, можно выделить следующее: критическому пути соответствуют задачи 1-2-4-6-8-10-11-12 или 1-3-4-6-8-10-11-12.

**Лабораторная работа №8. Графический метод решения оптимизационных задач**

**Цель:** освоить решение задач графическим методом.

**Задание.** Решить задачу графическим методом

В соответствии с вариантом была выбрана следующая задача:

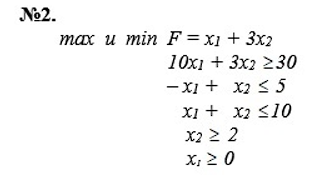


Рисунок 1 – исходная задача

Для начала построим на графике области ограничений. График представлен на рисунке 2.

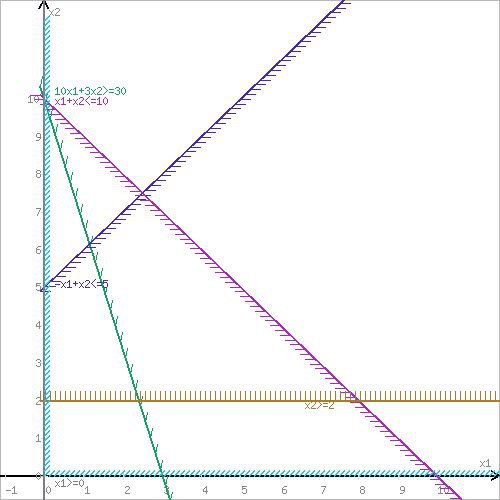


Рисунок 2 – область, ограниченная линиями

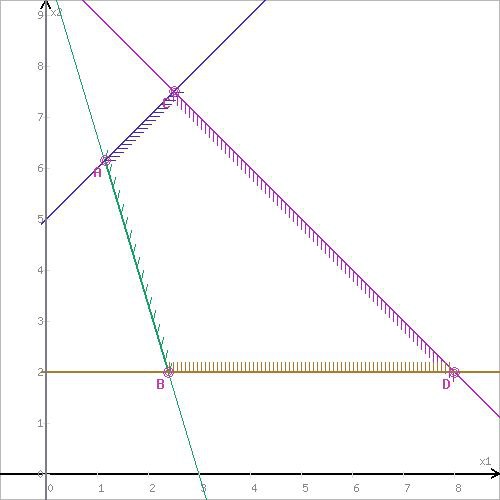


Рисунок 3 – полезная область

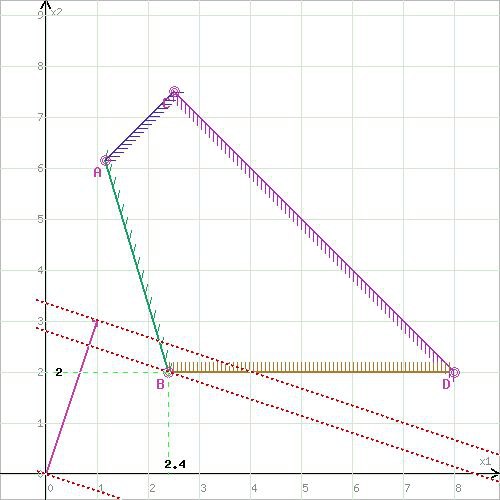


Рисунок 4 – нахождение минимального F

Параллельно сдвигая вектор решений, получим, что F достигает минимального значения в точке x1 = 2.4, x2 = 2, следовательно Fmin = 2.4 + 3 \* 2 = 8.4.

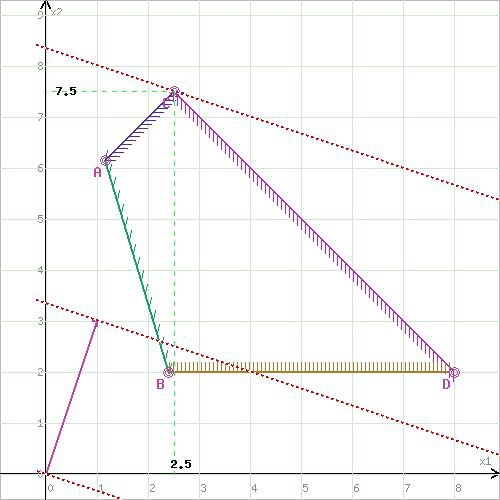


Рисунок 2 – нахождение максимального F

В данном случае решением является x1 = 2.5, x2 = 7,5. Fmax = 2.5 + 7.5 \* 3 = 24.