УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет информационных технологий

Кафедра программной инженерии

**Отчёт по лабораторным работам**

**по дисциплине «Математическое программирование»**

Выполнил:

Студент 2 курса 8 группы

Дмитроченко Кирилл Денисович

Преподаватель: асс. Ромыш А.С.

Минск

2025

**Содержание**

[**Лабораторная работа №1 «Вспомогательные функции». 3**](#_Toc195352538)

[**Лабораторная работа №2 «Комбинаторные алгоритмы». 9**](#_Toc195352539)

[**Лабораторная работа №3 «Метод ветвей и границ. Задача коммивояжера и методы её решения». 25**](#_Toc195352540)

[**Лабораторная работа №4 «Динамическое программирование». 28**](#_Toc195352541)

[**Лабораторная работа №5 «Транспортная задача». 34**](#_Toc195352542)

[**Лабораторная работа №6. «Алгоритм на графах» 41**](#_Toc195352543)

[**Лабораторная работа №7. «Сетевые модели» 63**](#_Toc195352544)

[**Лабораторная работа №8. «Графический метод решения задач оптимизации» 66**](#_Toc195352545)

# **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:

start – функция установки начального числа как текущего значения для генератора.

dget – функция возвращает действительное число.

iget – функция возвращает целое число.

В данной лабораторной я не использовал предкомпилированные заголовки, так как в этом не было необходимости. В задании 1 у нас были прототипы функций и сама их реализация, в листинге ниже приведен пример кода.

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

Листинг 1.1 – Файл Auxil.h

|  |
| --- |
| //-- Auxil.cpp  #include "Auxil.h"  #include <ctime>  namespace auxil  {  void start()  {  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 1.2 – Файл Auxil.cpp

***Задание 2***

1. Реализовать пример 2.
2. Для проверки работоспособности разработанных функций и приобретения навыков замера продолжительности процесса вычисления реализуйте программу, приведенную в примере 2.

В данном примере проверяем работоспособность наших трех функций, прототипы которых мы описывали ранее.

|  |
| --- |
| #include "Auxil.h" // вспомогательные функции  #include <iostream>  #include <ctime>  #include <locale>  #define CYCLE 1000000 // количество циклов(Почему не итераций???????)  void 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");  } |

Листинг 2.1 – Файл NomeroUno.cpp

Программа работает корректно, результат программы со всеми вычислениями представлен на рисунке 1

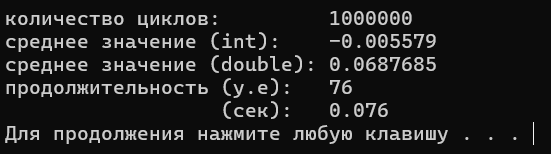


Рисунок 1 – Результат работы программы

**Задание 3**

Проведите необходимые эксперименты и постройте график зависимости (Excel) продолжительности процесса вычисления от количества циклов в примере 2. Проанализируйте характер зависимости. Проведите исследование любого другого рекурсивного алгоритма, например, вычисления факториала или генератора чисел Фибоначчи (прим. – например вычислите каким будет 100-е, 200-е, 300-е и т.д число), и включите в отчет график.

Для начала нам нужно найти зависимость продолжительности вычислений от количества циклов. Для этого просто каждый раз будем менять вручную количество циклов и проверим за какое время они будут выполняться. Диапазон циклов будет от 100000 до 1000000.

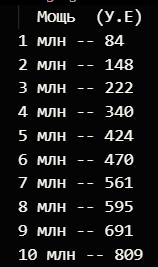


Рисунок 2 – Результаты тестирования №1

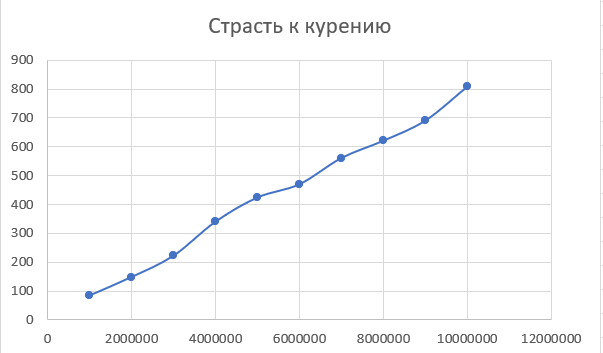


Рисунок 3 – График зависимости времени от цикла

Как мы видим, зависимость у нас линейная, так как количество операций прямо пропорционально числу циклов.

Проведем теперь исследование с числами Фибоначчи. Напишем программу, реализующую этот алгоритм.

|  |
| --- |
| #pragma once  #include <iostream>  int fibonachi(int n) {  if (n == 0)return 0;  if (n == 1)return 1;  return fibonachi(n - 1) + fibonachi(n - 2);  } |

Листинг 3.1 – Файл Fibonachi.h

|  |
| --- |
| #include "pch.h"  #include "Fibonachi.h"  using namespace std;  void main()  {  int n;  clock\_t t3 = 0, t4 = 0;  cout << "N:";  cin >> n;  t3 = clock();  int result = fibonachi(n);  cout << "result:" << result;  t4 = clock();  cout << std::endl << "time:" << (t4 - t3)<<" AUE";  std::cout << std::endl << " (sec): "  << ((double)(t4 - t3)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl;  } |

Листинг 3.2 – Файл Source.cpp

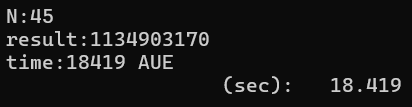


Рисунок 4 – Результат работы программы

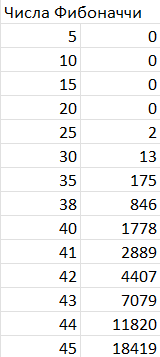
Для того, чтобы построить график и определить зависимость, мы также должны проверить за какое время будет выполняться алгоритм чисел Фибоначчи исходя из входного параметра N. Значения N будем менять вручную в диапазоне от 5 до 35 с шагом 5,после чего изменим шаг для более подробных измерений.

Рисунок 5 – Результаты тестирования №2



Рисунок 6 – График зависимости времени от N

На графике видно большое отклонение, так что данную зависимость можно считать экспоненциальной.

**ИТОГИ:**

1. В первом эксперименте зависимость времени выполнения от количества циклов линейная.
2. Во втором эксперименте зависимость времени выполнения рекурсивного алгоритма Фибоначчи экспоненциальная.

# **Лабораторная работа №2 «Комбинаторные алгоритмы».**

**ЦЕЛЬ РАБОТЫ:** приобрести навыки разработки генераторов подмножеств, перестановок, сочетаний и размещений на С++; научиться применять разработанные генераторы для решения следующих задач:

- о рюкзаке (упрощенную)

- коммивояжера

- об оптимальной загрузке судна

- об оптимальной загрузке судна с центровкой.

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

|  |
| --- |
| #pragma once  namespace combi  {  struct subset // генератор множества всех подмножеств  {  short n, // количество элементов исходного множества < 64  sn, // количество элементов текущего подмножества  \* sset; // массив индексов текущего подмножества  unsigned \_\_int64 mask; // битовая маска  subset(short n = 1); // конструктор(количество элементов исходного множества)  short getfirst(); // сформормировать массив индексов по битовой маске  short getnext(); // ++маска и сформировать массив индексов  short ntx(short i); // получить i-й элемент массива индексов  unsigned \_\_int64 count(); // вычислить общее количество подмножеств  void reset(); // сбросить генератор, начать сначала  };  }; |

Листинг 1.1 – Файл Combi.h

|  |
| --- |
| #include "Combi.h"  #include <algorithm>  namespace combi  {  subset::subset(short n)  {  this->n = n;  this->sset = new short[n];  this->reset();  };  void subset::reset()  {  this->sn = 0;  this->mask = 0;  };  short subset::getfirst()  {  \_\_int64 buf = this->mask;  this->sn = 0;  for (short i = 0; i < n; i++)  {  if (buf & 0x1) this->sset[this->sn++] = i;  buf >>= 1;  }  return this->sn;  };  short subset::getnext()  {  int rc = -1;  this->sn = 0;  if (++this->mask < this->count()) rc = getfirst();  return rc;  };  short subset::ntx(short i)  {  return this->sset[i];  };  unsigned \_\_int64 subset::count()  {  return (unsigned \_\_int64)(1 << this->n);  };  }; |

Листинг 1.2 – Файл Combi.cpp

|  |
| --- |
| #include <iostream>  #include <tchar.h>  #include "Combi.h"  int \_tmain(int argc, \_TCHAR\* argv[])  {  setlocale(LC\_ALL, "rus");  char AA[][2] = { "A", "B", "C", "D","E"};  std::cout << std::endl << " - Генератор множества всех подмножеств -";  std::cout << std::endl << "Исходное множество: ";  std::cout << "{ ";  for (int i = 0; i < sizeof(AA) / 2; i++)  std::cout << AA[i] << ((i < sizeof(AA) / 2 - 1) ? ", " : " ");  std::cout << "}";  std::cout << std::endl << "Генерация всех подмножеств ";  combi::subset s1(sizeof(AA) / 2); // создание генератора  int n = s1.getfirst(); // первое (пустое) подмножество  while (n >= 0) // пока есть подмножества  {  std::cout << std::endl << "{ ";  for (int i = 0; i < n; i++)  std::cout << AA[s1.ntx(i)] << ((i < n - 1) ? ", " : " ");  std::cout << "}";  n = s1.getnext(); // cледующее подмножество  };  std::cout << std::endl << "всего: " << s1.count() << std::endl;  system("pause");  return 0;  } |

Листинг 1.3 – Файл Source.cpp

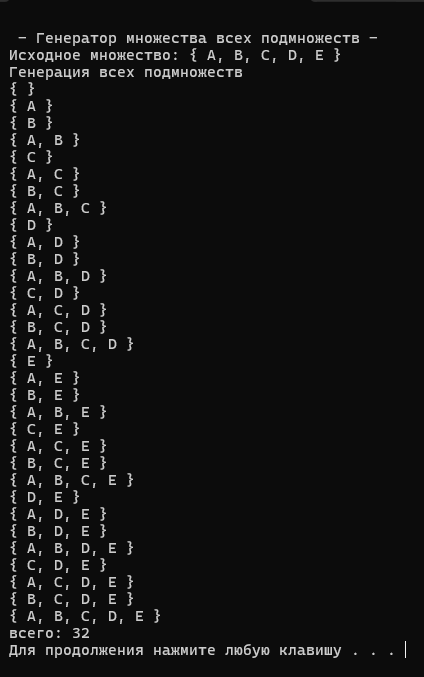


Рисунок 1 – Результат работы программы

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

|  |
| --- |
| #pragma once  namespace combi  {  struct xcombination // генератор сочетаний (эвристика)  {  short n, // количество элементов исходного множества  m, // количество элементов в сочетаниях  \* sset; // массив индексов текущего сочетания  xcombination(  short n = 1, //количество элементов исходного множества  short m = 1 // количество элементов в сочетаниях  );  void reset(); // сбросить генератор, начать сначала  short getfirst(); // сформировать первый массив индексов  short getnext(); // сформировать следующий массив индексов  short ntx(short i); // получить i-й элемент массива индексов  unsigned \_\_int64 nc; // номер сочетания 0,..., count()-1  unsigned \_\_int64 count() const; // вычислить количество сочетаний  };  }; |

Листинг 2.1 – Файл Combi.h

|  |
| --- |
| #include "Combi.h"  #include <algorithm>  namespace combi  {  xcombination::xcombination(short n, short m)  {  this->n = n;  this->m = m;  this->sset = new short[m + 2];  this->reset();  }  void xcombination::reset() // сбросить генератор, начать сначала  {  this->nc = 0;  for (int i = 0; i < this->m; i++) this->sset[i] = i;  this->sset[m] = this->n;  this->sset[m + 1] = 0;  };  short xcombination::getfirst()  {  return (this->n >= this->m) ? this->m : -1;  };  short xcombination::getnext() // сформировать следующий массив индексов  {  short rc = getfirst();  if (rc > 0)  {  short j;  for (j = 0; this->sset[j] + 1 == this->sset[j + 1]; ++j)  this->sset[j] = j;  if (j >= this->m) rc = -1;  else {  this->sset[j]++;  this->nc++;  };  }  return rc;  };  short xcombination::ntx(short i)  {  return this->sset[i];  };  unsigned \_\_int64 fact(unsigned \_\_int64 x) { return(x == 0) ? 1 : (x \* fact(x - 1)); };  unsigned \_\_int64 xcombination::count() const  {  return (this->n >= this->m) ?  fact(this->n) / (fact(this->n - this->m) \* fact(this->m)) : 0;  };  }; |

Листинг 2.2 – Файл Combi.cpp

|  |
| --- |
| #include <iostream>  #include <tchar.h>  #include "Combi.h"  int \_tmain(int argc, \_TCHAR\* argv[])  {  setlocale(LC\_ALL, "rus");  char AA[][2] = { "A", "B", "C", "D"};  std::cout << std::endl << " --- Генератор сочетаний ---";  std::cout << std::endl << "Исходное множество: ";  std::cout << "{ ";  for (int i = 0; i < sizeof(AA) / 2; i++)  std::cout << AA[i] << ((i < sizeof(AA) / 2 - 1) ? ", " : " ");  std::cout << "}";  std::cout << std::endl << "Генерация сочетаний ";  combi::xcombination xc(sizeof(AA) / 2, 3);  std::cout << "из " << xc.n << " по " << xc.m;  int n = xc.getfirst();  while (n >= 0)  {  std::cout << std::endl << xc.nc << ": { ";  for (int i = 0; i < n; i++)  std::cout << AA[xc.ntx(i)] << ((i < n - 1) ? ", " : " ");  std::cout << "}";  n = xc.getnext();  };  std::cout << std::endl << "всего: " << xc.count() << std::endl;  system("pause");  return 0;  } |

Листинг 2.3 – Файл Source.cpp

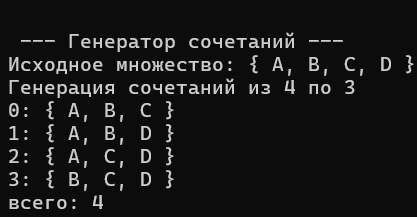


Рисунок 2 – Результат работы программы

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

Генератор перестановок – это инструмент или алгоритм, который создаёт все возможные перестановки (порядки) элементов заданного множества. Перестановка представляет собой упорядоченное расположение элементов множества.

|  |
| --- |
| #pragma once  namespace combi  {  struct permutation // генератор перестановок  {  const static bool L = true; // левая стрелка  const static bool R = false; // правая стрелка  short n, // количество элементов исходного множества  \* sset; // массив индексов текущей перестановки  bool\* dart; // массив стрелок (левых-L и правых-R)  permutation(short n = 1); // конструктор (количество элементов исходного множества)  void reset(); // сбросить генератор, начать сначала  \_\_int64 getfirst(); // сформировать первый массив индексов  \_\_int64 getnext(); // сформировать случайный массив индексов  short ntx(short i); // получить i-й элемент масива индексов  unsigned \_\_int64 np; // номер перествновки 0,... count()-1  unsigned \_\_int64 count() const; // вычислить общее кол. перестановок  };  } |

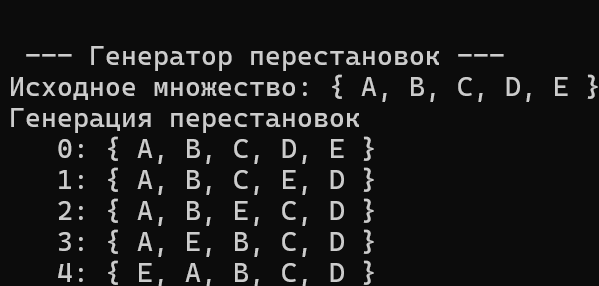
Листинг 3.1 – Файл Combi.h

|  |
| --- |
| #include "Combi.h"  #include <algorithm>  #define NINF ((short)0x8000)  namespace combi  {  permutation::permutation(short n)  {  this->n = n;  this->sset = new short[n];  this->dart = new bool[n];  this->reset();  };  void permutation::reset()  {  this->getfirst();  };  \_\_int64 permutation::getfirst()  {  this->np = 0;  for (int i = 0; i < this->n; i++)  {  this->sset[i] = i; this->dart[i] = L;  };  return (this->n > 0) ? this->np : -1;  };  \_\_int64 permutation::getnext() //  {  \_\_int64 rc = -1;  short maxm = NINF, idx = -1;  for (int i = 0; i < this->n; i++)  {  if (i > 0 &&  this->dart[i] == L &&  this->sset[i] > this->sset[i - 1] &&  maxm < this->sset[i]) maxm = this->sset[idx = i];  if (i < (this->n - 1) &&  this->dart[i] == R &&  this->sset[i] > this->sset[i + 1] &&  maxm < this->sset[i]) maxm = this->sset[idx = i];  };  if (idx >= 0)  {  std::swap(this->sset[idx],  this->sset[idx + (this->dart[idx] == L ? -1 : 1)]);  std::swap(this->dart[idx],  this->dart[idx + (this->dart[idx] == L ? -1 : 1)]);  for (int i = 0; i < this->n; i++)  if (this->sset[i] > maxm) this->dart[i] = !this->dart[i];  rc = ++this->np;  }  return rc;  };  short permutation::ntx(short i) { return this->sset[i]; };  unsigned \_\_int64 fact(unsigned \_\_int64 x) { return (x == 0) ? 1 : (x \* fact(x - 1)); };  unsigned \_\_int64 permutation::count() const { return fact(this->n); };  } |

Листинг 3.2 – Файл Combi.cpp

|  |
| --- |
| #include <iostream>  #include "Combi.h"  #include <tchar.h>  #include <iomanip>  int \_tmain(int argc, \_TCHAR\* argv[])  {  setlocale(LC\_ALL, "rus");  char AA[][2] = { "A", "B", "C", "D","E"};  std::cout << std::endl << " --- Генератор перестановок ---";  std::cout << std::endl << "Исходное множество: ";  std::cout << "{ ";  for (int i = 0; i < sizeof(AA) / 2; i++)  std::cout << AA[i] << ((i < sizeof(AA) / 2 - 1) ? ", " : " ");  std::cout << "}";  std::cout << std::endl << "Генерация перестановок ";  combi::permutation p(sizeof(AA) / 2);  \_\_int64 n = p.getfirst();  while (n >= 0)  {  std::cout << std::endl << std::setw(4) << p.np << ": { ";  for (int i = 0; i < p.n; i++)  std::cout << AA[p.ntx(i)] << ((i < p.n - 1) ? ", " : " ");  std::cout << "}";  n = p.getnext();  };  std::cout << std::endl << "всего: " << p.count() << std::endl;  system("pause");  return 0;  } |

Листинг 3.3 – Файл Source.cpp



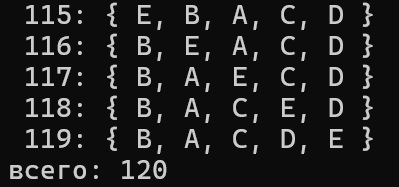


Рисунок 3 – Результат работы программы

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

|  |
| --- |
| #pragma once  namespace combi  {  struct xcombination  {  short n,  m,  \* sset;  xcombination(  short n = 1,  short m = 1  );  void reset();  short getfirst();  short getnext();  short ntx(short i);  unsigned \_\_int64 nc;  unsigned \_\_int64 count() const;  };  struct permutation  {  const static bool L = true;  const static bool R = false;  short n,  \* sset;  bool\* dart;  permutation(short n = 1);  void reset();  \_\_int64 getfirst();  \_\_int64 getnext();  short ntx(short i);  unsigned \_\_int64 np;  unsigned \_\_int64 count() const;  };  struct accomodation  {  short n,  m,  \* sset;  xcombination\* cgen;  permutation\* pgen;  accomodation(short n = 1, short m = 1);  void reset();  short getfirst();  short getnext();  short ntx(short i);  unsigned \_\_int64 na;  unsigned \_\_int64 count() const;  };  } |

Листинг 4.1 – Файл Combi.h

|  |
| --- |
| #include "Combi.h"  #include <algorithm>  #define NINF ((short)0x8000)  namespace combi  {  accomodation::accomodation(short n, short m)  {  this->n = n;  this->m = m;  this->cgen = new xcombination(n, m);  this->pgen = new permutation(m);  this->sset = new short[m];  this->reset();  }  void accomodation::reset()  {  this->na = 0;  this->cgen->reset();  this->pgen->reset();  this->cgen->getfirst();  };  short accomodation::getfirst()  {  short rc = (this->n >= this->m) ? this->m : -1;  if (rc > 0)  {  for (int i = 0; i <= this->m; i++)  this->sset[i] = this->cgen->sset[this->pgen->ntx(i)];  };  return rc;  };  short accomodation::getnext()  {  short rc;  this->na++;  if ((this->pgen->getnext()) > 0) rc = this->getfirst();  else if ((rc = this->cgen->getnext()) > 0)  {  this->pgen->reset(); rc = this->getfirst();  };  return rc;  };  short accomodation::ntx(short i)  {  return this->sset[i];  };  // факториал из структуры accomodation  unsigned \_\_int64 fact(unsigned \_\_int64 x) { return (x == 0) ? 1 : (x \* fact(x - 1)); };  unsigned \_\_int64 accomodation::count() const  {  return (this->n >= this->m) ?  fact(this->n) / fact(this->n - this->m) : 0;  };  // ================================== PERMUTATION ==================================  permutation::permutation(short n)  {  this->n = n;  this->sset = new short[n];  this->dart = new bool[n];  this->reset();  };  void permutation::reset()  {  this->getfirst();  };  \_\_int64 permutation::getfirst()  {  this->np = 0;  for (int i = 0; i < this->n; i++)  {  this->sset[i] = i; this->dart[i] = L;  };  return (this->n > 0) ? this->np : -1;  };  \_\_int64 permutation::getnext()  {  \_\_int64 rc = -1;  short maxm = NINF, idx = -1;  for (int i = 0; i < this->n; i++)  {  if (i > 0 &&  this->dart[i] == L &&  this->sset[i] > this->sset[i - 1] &&  maxm < this->sset[i]) maxm = this->sset[idx = i];  if (i < (this->n - 1) &&  this->dart[i] == R &&  this->sset[i] > this->sset[i + 1] &&  maxm < this->sset[i]) maxm = this->sset[idx = i];  };  if (idx >= 0)  {  std::swap(this->sset[idx],  this->sset[idx + (this->dart[idx] == L ? -1 : 1)]);  std::swap(this->dart[idx],  this->dart[idx + (this->dart[idx] == L ? -1 : 1)]);  for (int i = 0; i < this->n; i++)  if (this->sset[i] > maxm) this->dart[i] = !this->dart[i];  rc = ++this->np;  }  return rc;  };  short permutation::ntx(short i) { return this->sset[i]; };  unsigned \_\_int64 permutation::count() const { return fact(this->n); };  // ================================== XCOMBINATION ==================================  xcombination::xcombination(short n, short m)  {  this->n = n;  this->m = m;  this->sset = new short[m + 2];  this->reset();  }  void xcombination::reset()  {  this->nc = 0;  for (int i = 0; i < this->m; i++) this->sset[i] = i;  this->sset[m] = this->n;  this->sset[m + 1] = 0;  };  short xcombination::getfirst()  {  return (this->n >= this->m) ? this->m : -1;  };  short xcombination::getnext()  {  short rc = getfirst();  if (rc > 0)  {  short j;  for (j = 0; this->sset[j] + 1 == this->sset[j + 1]; ++j)  this->sset[j] = j;  if (j >= this->m) rc = -1;  else  {  this->sset[j]++;  this->nc++;  }  }  return rc;  };  short xcombination::ntx(short i)  {  return this->sset[i];  };  unsigned \_\_int64 xcombination::count() const  {  return (this->n >= this->m) ?  fact(this->n) / (fact(this->n - this->m) \* fact(this->m)) : 0;  };  } |

Листинг 4.2 – Файл Combi.cpp

|  |
| --- |
| #include <iostream>  #include <tchar.h>  #include <iomanip>  #include "Combi.h"  #define N (sizeof(AA)/2)  #define M 3  int \_tmain(int argc, \_TCHAR\* argv[])  {  setlocale(LC\_ALL, "rus");  char AA[][2] = { "A", "B", "C", "D" };  std::cout << std::endl << " --- Генератор размещений ---";  std::cout << std::endl << "Исходное множество: ";  std::cout << "{ ";  for (int i = 0; i < N; i++)  std::cout << AA[i] << ((i < N - 1) ? ", " : " ");  std::cout << "}";  std::cout << std::endl << "Генерация размещений из " << N << " по " << M;  combi::accomodation s(N, M);  int n = s.getfirst();  while (n >= 0)  {  std::cout << std::endl << std::setw(2) << s.na << ": { ";  for (int i = 0; i < 3; i++)  std::cout << AA[s.ntx(i)] << ((i < n - 1) ? ", " : " ");  std::cout << "}";  n = s.getnext();  };  std::cout << std::endl << "всего: " << s.count() << std::endl;  system("pause");  return 0;  } |

Листинг 4.3 – Файл Source.cpp

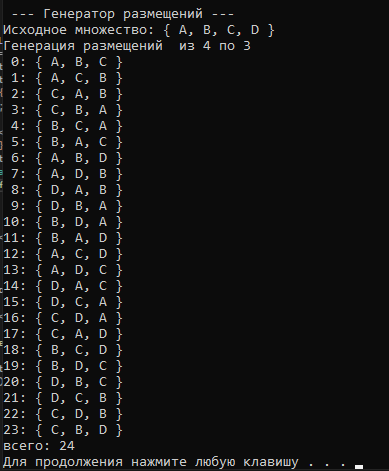


Рисунок 4 – Результат работы программы

**Задание 5.**  Решить в соответствии с вариантом задачу и результат занести в отчет (Вариант распределяется по списку). (3, 7, 11, 15) об оптимальной загрузке судна (веса контейнеров сгенерировать случайным образом: ограничение по общему весу – 1500 кг., количество мест на судне для контейнеров – 5, количество контейнеров 25, веса контейнеров 100 – 900 кг., доход от перевозки 10 – 150 у.е.);

|  |
| --- |
| #pragma once  #include "Combi.h"  int boat(  int V, // [in] максимальный вес груза  short m, // [in] количество мест для контейнеров  short n, // [in] всего контейнеров  const int v[], // [in] вес каждого контейнера  const int c[], // [in] доход от перевозки каждого контейнера  short r[] // [out] результат: индексы выбранных контейнеров  ); |

Листинг 5.1 – Файл Boat.h

|  |
| --- |
| #include "Boat.h"  #include "Combi.h"  namespace boatfnc  {  int calcv(combi::xcombination s, const int v[]) // вес  {  int rc = 0;  for (int i = 0; i < s.m; i++) rc += v[s.ntx(i)];  return rc;  };  int calcc(combi::xcombination s, const int c[]) // доход  {  int rc = 0;  for (int i = 0; i < s.m; i++) rc += c[s.ntx(i)];  return rc;  };  void copycomb(short m, short\* r1, const short\* r2) // копировать  {  for (int i = 0; i < m; i++) r1[i] = r2[i];  };  }  int boat(  int V, // [in] максимальный вес груза  short m, // [in] количество мест для контейнеров  short n, // [in] всего контейнеров  const int v[], // [in] вес каждого контейнера  const int c[], // [in] доход от перевозки каждого контейнера  short r[] // [out] результат: индексы выбранных контейнеров  )  {  combi::xcombination xc(n, m);  int rc = 0, i = xc.getfirst(), cc = 0;  while (i > 0)  {  if (boatfnc::calcv(xc, v) <= V)  if ((cc = boatfnc::calcc(xc, c)) > rc)  {  rc = cc; boatfnc::copycomb(m, r, xc.sset);  }  i = xc.getnext();  };  return rc;  }; |

Листинг 5.2 – Файл Boat.cpp

|  |
| --- |
| #include <iostream>  #include <chrono>  #include <tchar.h>  #include <iomanip>  #include "Boat.h"  #define NN 35  #define MM 6  int \_tmain(int argc, \_TCHAR\* argv[])  {  setlocale(LC\_ALL, "rus");  int V = 1500,  v[] = { 345, 678, 123, 890, 456, 210, 567, 789, 134, 678,  345, 890, 234, 567, 456, 789, 123, 678, 345, 890,  567, 234, 456, 789, 678,427,715,195,682,829,723,278,414,318,681},  c[NN] = { 45, 120, 75, 30, 110, 60, 90, 15, 105, 50,  80, 25, 95, 40, 85, 20, 100, 35, 70, 55,  65, 10, 115, 45, 130,11,82,120,34,100,14,145,67,92,23};  short r[MM];  auto start = clock();  int cc = boat(  V, // [in] максимальный вес груза  MM, // [in] количество мест для контейнеров  NN, // [in] всего контейнеров  v, // [in] вес каждого контейнера  c, // [in] доход от перевозки каждого контейнера  r // [out] результат: индексы выбранных контейнеров  );  auto end = clock();  std::cout << std::endl << "- Задача о размещении контейнеров на судне";  std::cout << std::endl << "- общее количество контейнеров : " << NN;  std::cout << std::endl << "- количество мест для контейнеров : " << MM;  std::cout << std::endl << "- ограничение по суммарному весу : " << V;  std::cout << std::endl << "- вес контейнеров : ";  for (int i = 0; i < NN; i++) std::cout << std::setw(3) << v[i] << " ";  std::cout << std::endl << "- доход от перевозки : ";  for (int i = 0; i < NN; i++) std::cout << std::setw(3) << c[i] << " ";  std::cout << std::endl << "- выбраны контейнеры (0,1,...,m-1): ";  for (int i = 0; i < MM; i++) std::cout << r[i] << " ";  std::cout << std::endl << "- доход от перевозки : " << cc;  std::cout << std::endl << "- общий вес выбранных контейнеров : ";  int s = 0; for (int i = 0; i < MM; i++) s += v[r[i]]; std::cout << s;  std::cout << std::endl << std::endl;  std::cout << "-------------------------------------------------------------" << std::endl;  std::cout << std::endl << "Время выполнения: " << (double)(end - start) / CLOCKS\_PER\_SEC << " секунд"<<std::endl;  system("pause");  return 0;  } |

Листинг 5.3 – Файл Source.cpp

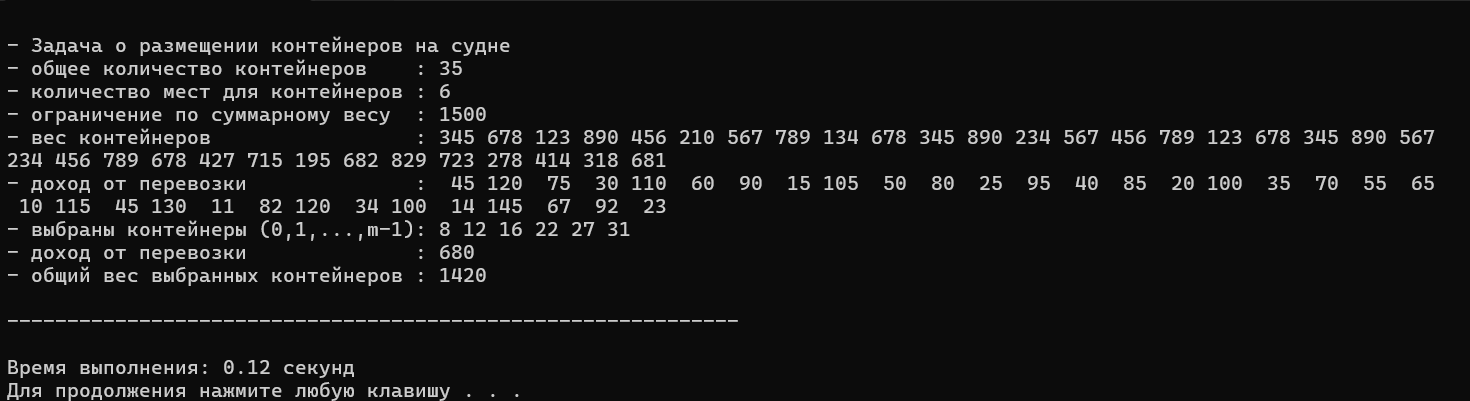


Рисунок 5 – Результат работы программы

**Задание 6.** Исследовать зависимость времени вычисления необходимое для решения задачи (в соответствии с вариантом) от размерности задачи и результат в виде графика с небольшим пояснением занести в отчет: (3, 7, 11, 15) об оптимальной загрузке судна (количество мест на судне для контейнеров – 6, количество контейнеров 25 – 35.

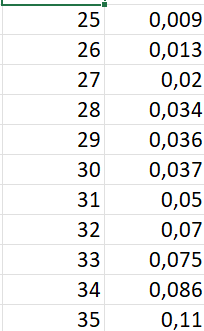


Рисунок 6 – Полученные экспериментальные данные



Рисунок 7 – График зависимости времени вычисления от количества контейнеров

**Вывод:** Исходя из полученных данных и графика зависимости можно сделать вывод, что зависимость линейная.

# **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. Для этого:

* принять элементы матрицы расстояний равными:

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

где *n* – номер варианта или номер по журналу;

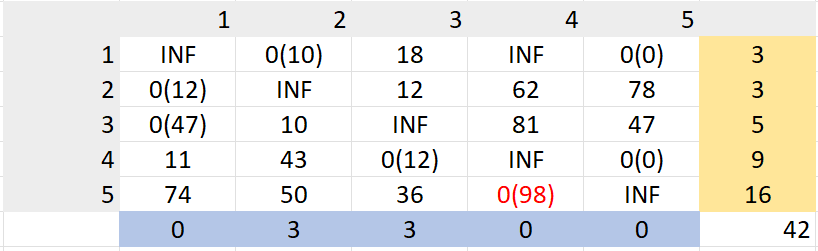
Взяв в качестве n число 3, получаем следующую таблицу:

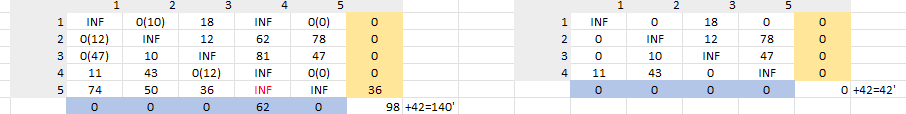
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 6 | 24 |  | 3 |
| **2** | 3 |  | 18 | 65 | 81 |
| **3** | 5 | 9 |  | 86 | 52 |
| **4** | 20 | 55 | 4 \* n |  | 9 |
| **5** | 90 | 69 | 52 | 16 |  |

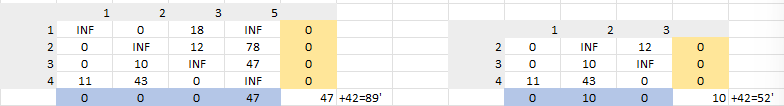
**Задача коммивояжера** — одна из самых известных оптимизационных задач. Ее цель заключается в нахождении самого выгодного маршрута (кратчайшего, самого быстрого, наиболее дешевого), проходящего через все заданные точки (пункты, города) по одному разу, с последующим возвратом в исходную точку.

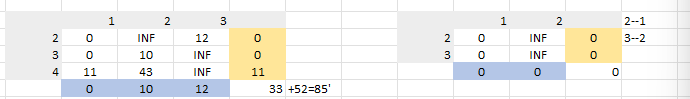
**Задание 2.** Решить сформулированную задачу методом ветвей и границ.

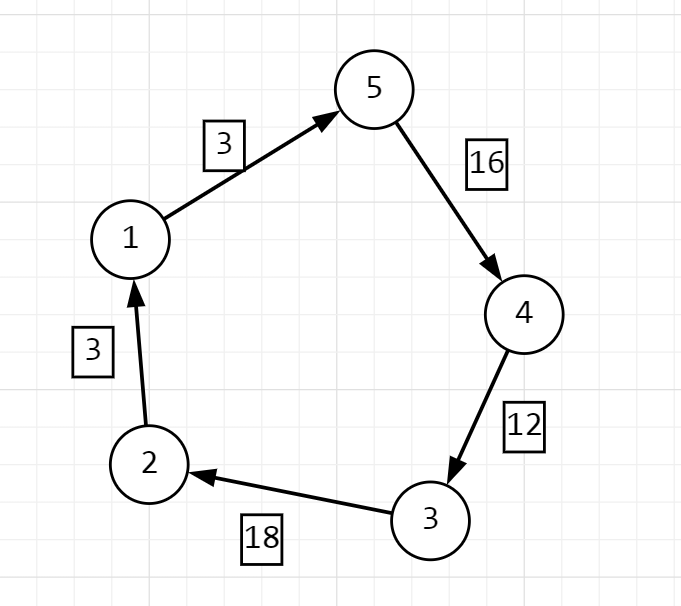
Этапы выполнения:



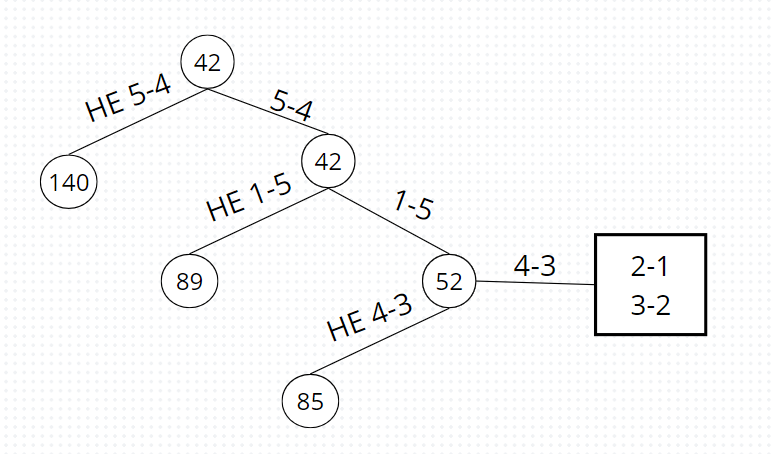






Полученный граф: 

Полученное дерево:



Полученный маршрут: **5 – 4 – 3 – 2 – 1**

Длина маршрута: 52

**Вывод**: Получен оптимальный маршрут путём решения задачи коммивояжера методом ветвей и границ

# **Лабораторная работа №4 «Динамическое программирование».**

**1.**

|  |
| --- |
| random\_device rd;  mt19937 gen(rd());  const char alphabet[] = "abcdefghijklmnopqrstuvwxyz";  uniform\_int\_distribution<> gg(0, 25);  string S1, S2;  for (int i = 0; i < 300; i++) {  S1 += alphabet[gg(gen)];  }  cout << "S1: " << S1 << "\n\n";  for (int i = 0; i < 200; i++) {  S2 += alphabet[gg(gen)];  }  cout << "S2: " << S2 << "\n\n"; |

Листинг 1 – генерация S1 и S2

2.

|  |
| --- |
| #include <algorithm>  #include <iostream>  #include <ctime>  #include <iomanip>  #define DD(i,j) d[(i)\*(ly+1)+(j)]  using namespace std;  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;  };  int levenshtein\_recursive(int lx, const char x[], int ly, const char y[]) {  if (lx == 0) return ly;  if (ly == 0) return lx;  int cost = (x[lx - 1] == y[ly - 1]) ? 0 : 1;  return min3(  levenshtein\_recursive(lx - 1, x, ly, y) + 1, // удаление  levenshtein\_recursive(lx, x, ly - 1, y) + 1, // вставка  levenshtein\_recursive(lx - 1, x, ly - 1, y) + cost // замена  );  }  int main()  {  setlocale(LC\_ALL, "rus");  clock\_t t1 = 0, t2 = 0, t3, t4;  char x[] = "abcdefghklmnoxm", y[] = "xyabcdefghomnkm";  int lx = sizeof(x) - 1, ly = sizeof(y) - 1;  std::cout << std::endl;  std::cout << std::endl << "-- расстояние Левенштейна -----" << std::endl;  std::cout << std::endl << "--длина --- рекурсия -- дин.програм. ---"  << std::endl;  for (int i = 8; i < std::min(lx, ly); i++)  {  t1 = clock(); cout<<levenshtein\_r(i, x, i - 2, y)<<" "; t2 = clock();  t3 = clock(); cout<<levenshtein(i, x, i - 2, y); t4 = clock();  std::cout << std::right << std::setw(3) << i - 2 << "/" << std::setw(2) << i  << " " << std::left << std::setw(10) << (t2 - t1)  << " " << std::setw(10) << (t4 - t3) << std::endl;  }  system("pause");  return 0;  } |

Листинг 2 – Вычисление расстояния Левенштейна и замер времени выполнения при каждом k

3.

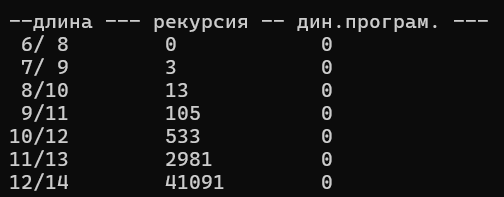


Рис.1 Результат выполнения программы

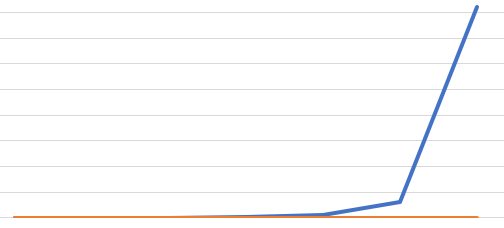


Рис.2 График зависимости

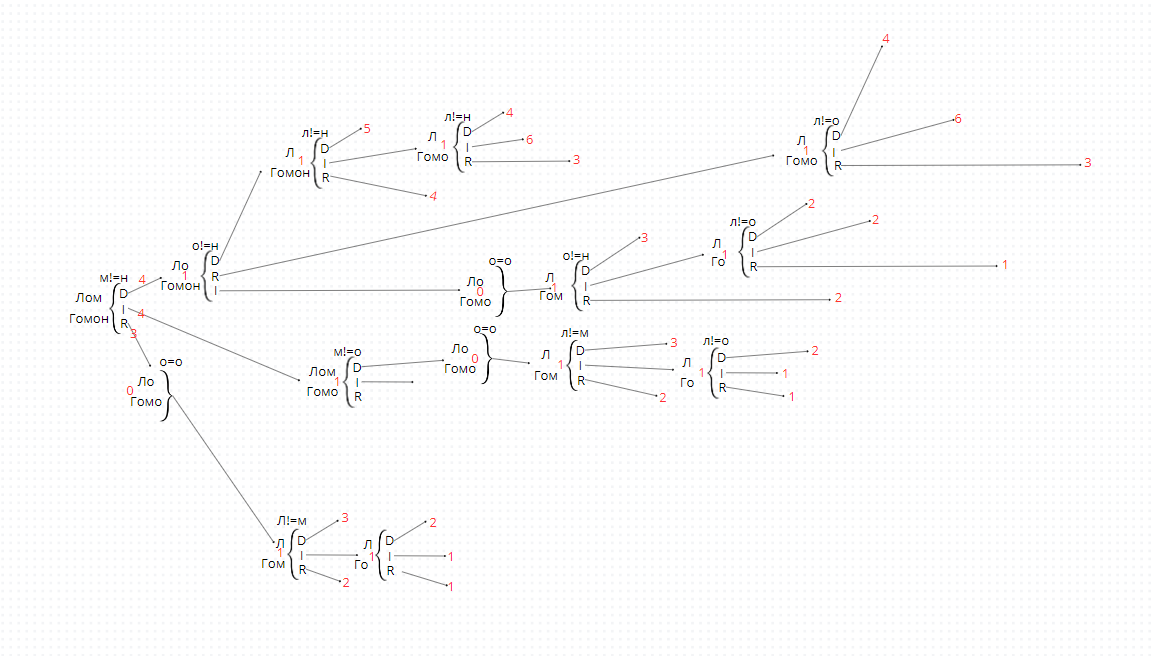
4.

Рис.3 Ручное вычисление расстояния Левенштейна

5.

|  |
| --- |
| #include <iostream>  #include <vector>  #include <algorithm>  #include <chrono>  using namespace std;  // Рекурсивный метод  int lcs\_recursive(int lenx, const string& x, int leny, const string& y) {  if (lenx == 0 || leny == 0)  return 0;  if (x[lenx - 1] == y[leny - 1])  return 1 + lcs\_recursive(lenx - 1, x, leny - 1, y);  else  return max(lcs\_recursive(lenx, x, leny - 1, y), lcs\_recursive(lenx - 1, x, leny, y));  }  // Динамическое программирование  int lcs\_dp(const string& x, const string& y) {  int lenx = x.size(), leny = y.size();  vector<vector<int>> dp(lenx + 1, vector<int>(leny + 1, 0));  for (int i = 1; i <= lenx; i++) {  for (int j = 1; j <= leny; j++) {  if (x[i - 1] == y[j - 1])  dp[i][j] = dp[i - 1][j - 1] + 1;  else  dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);  }  }  return dp[lenx][leny];  }  int main() {  string S1 = "ABCDFGI";  string S2 = "EATUFI";  int len1 = S1.size();  int len2 = S2.size();  vector<double> k\_values = { 1.0 / 25, 1.0 / 20, 1.0 / 15, 1.0 / 10, 1.0 / 5, 1.0 / 2, 1.0 };  for (double k : k\_values) {  int len\_prefix1 = max(1, static\_cast<int>(k \* len1));  int len\_prefix2 = max(1, static\_cast<int>(k \* len2));  string prefix1 = S1.substr(0, len\_prefix1);  string prefix2 = S2.substr(0, len\_prefix2);  // Динамическое программирование  auto start = chrono::high\_resolution\_clock::now();  int dp\_result = lcs\_dp(prefix1, prefix2);  auto end = chrono::high\_resolution\_clock::now();  chrono::duration<double> time\_dp = end - start;  cout << "DP (" << prefix1 << ", " << prefix2 << "): " << dp\_result << " | Time: " << time\_dp.count() << "s" << endl;  // Рекурсивный метод  start = chrono::high\_resolution\_clock::now();  int rec\_result = lcs\_recursive(prefix1.size(), prefix1, prefix2.size(), prefix2);  end = chrono::high\_resolution\_clock::now();  chrono::duration<double> time\_rec = end - start;  cout << "Recursive (" << prefix1 << ", " << prefix2 << "): " << rec\_result << " | Time: " << time\_rec.count() << "s" << endl;  cout << "------------------------------------------------------" << endl;  }  return 0;  } |

Листинг 3 – Код для нахождения длины наибольшей общей последовательности

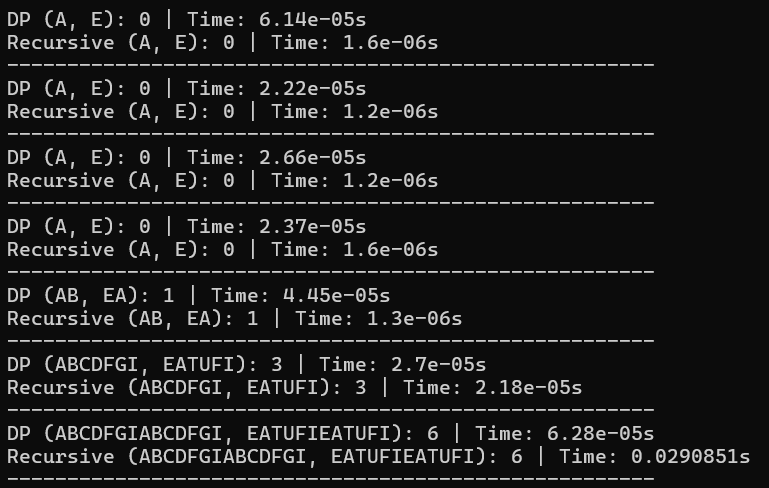
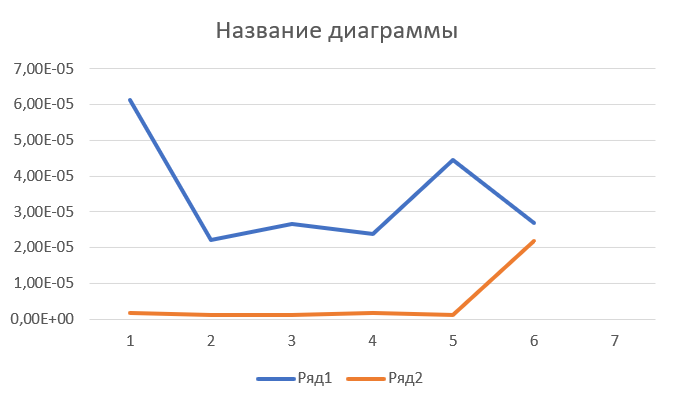


Рис.4 результат работы программы





**Вывод:** В лабораторной работе №4 были освоены общие принципы решения задач методом динамического программирования.

# **Лабораторная работа №5 «Транспортная задача».**

**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.

В качестве N берём число 3 и получаем следующую матрицу:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **15** | **5** | **9** | **6** | **14** | **4** | **171** |
| 2 | **13** | **3** | **11** | **8** | **10** | **16** | **116** |
| 3 | **4** | **8** | **14** | **11** | **5** | **14** | **153** |
| 4 | **7** | **13** | **13** | **6** | **16** | **5** | **162** |
| 5 | **6** | **14** | **12** | **3** | **13** | **7** | **103** |
| ПОТРЕБНОСТИ | **146** | **110** | **134** | **196** | **98** | **166** |  |

Таблица 1 Исходные данные

Sum a = 850, Sum b =705 , следовательно транспортная задача открытая

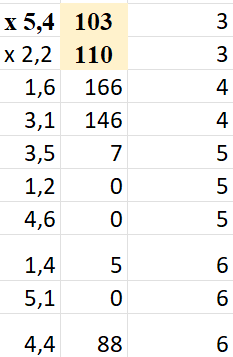


Рис.1 Конечные переменные

Этапы выполнения:



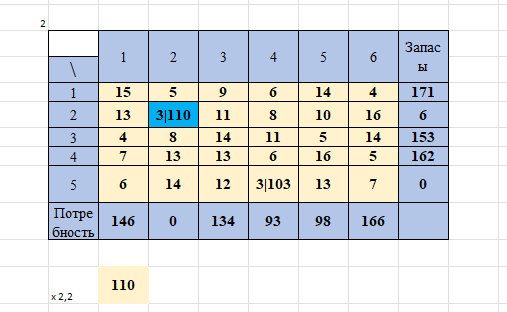
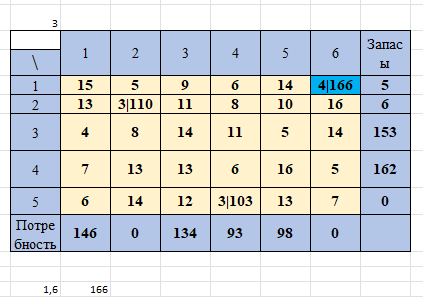
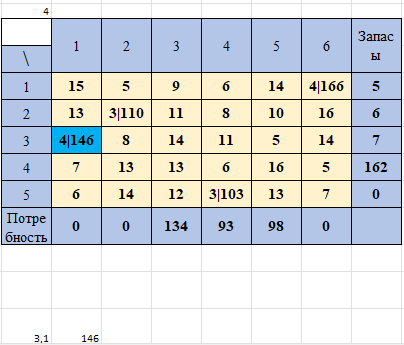
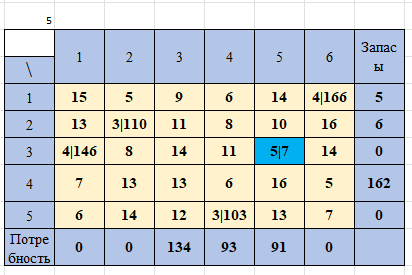
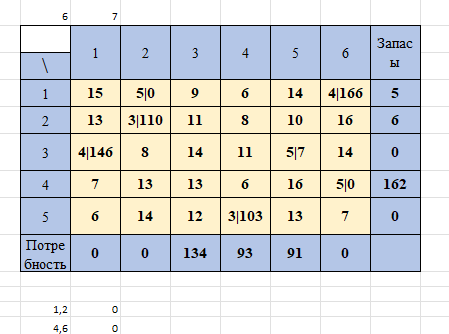
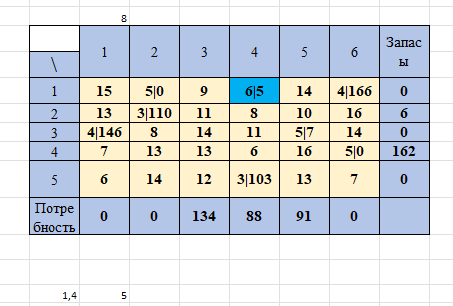
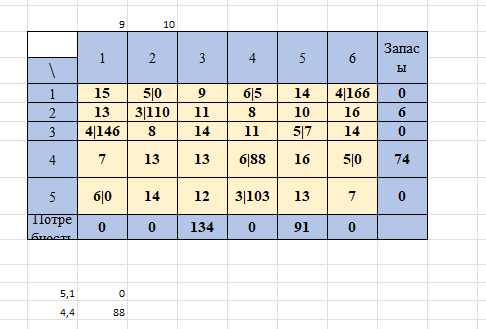
      



Рис.2 Длина маршрута

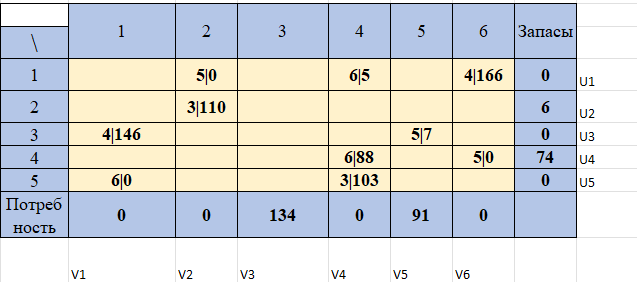


Рис.3 Финальный вид таблицы

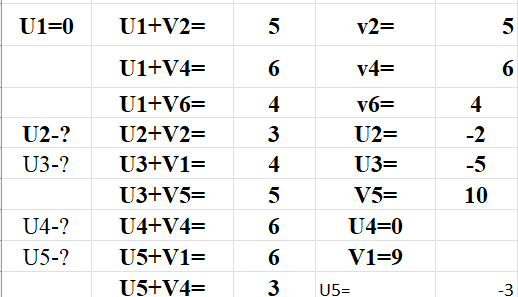


Рис.4 Система уравнений

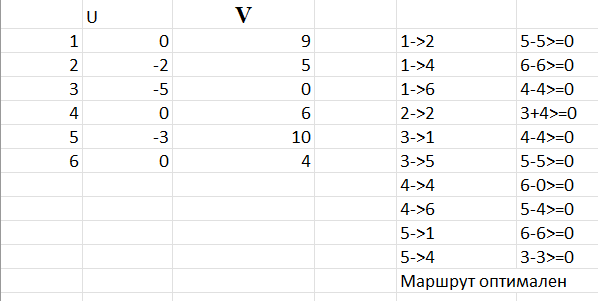


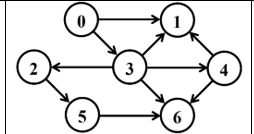
Рис.5 Проверка маршрута

**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. «Алгоритм 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

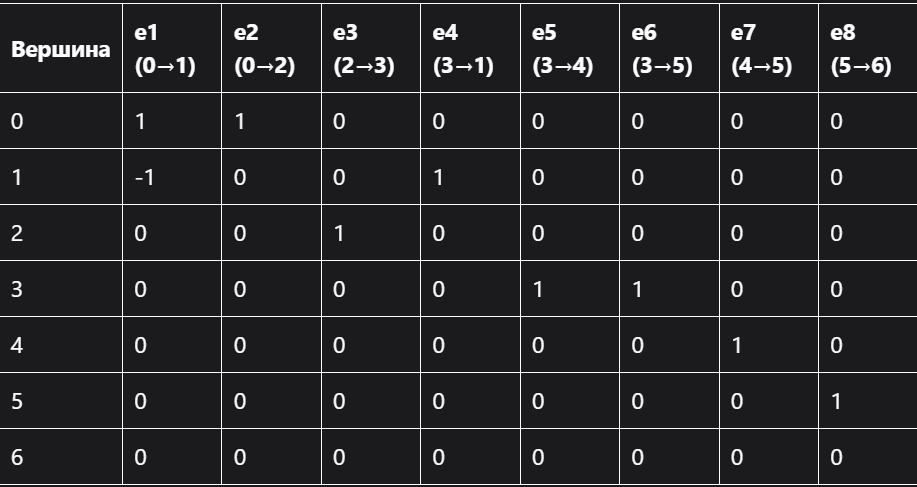
1. Граф:



Матрица смежности:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| gg | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 | 0 | 8 | 0 | 2 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 11 | 0 |
| 3 | 0 | 4 | 9 | 0 | 1 | 0 | 9 |
| 4 | 0 | 3 | 0 | 0 | 0 | 0 | 10 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Матрица инцидентности:



Список вершин:

* 1. 0-3

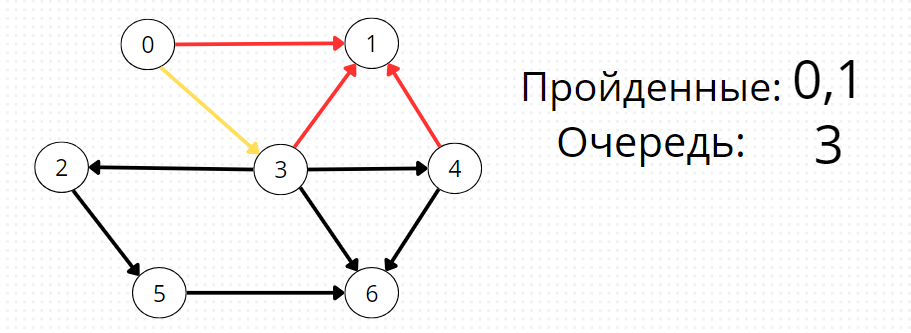
2-5

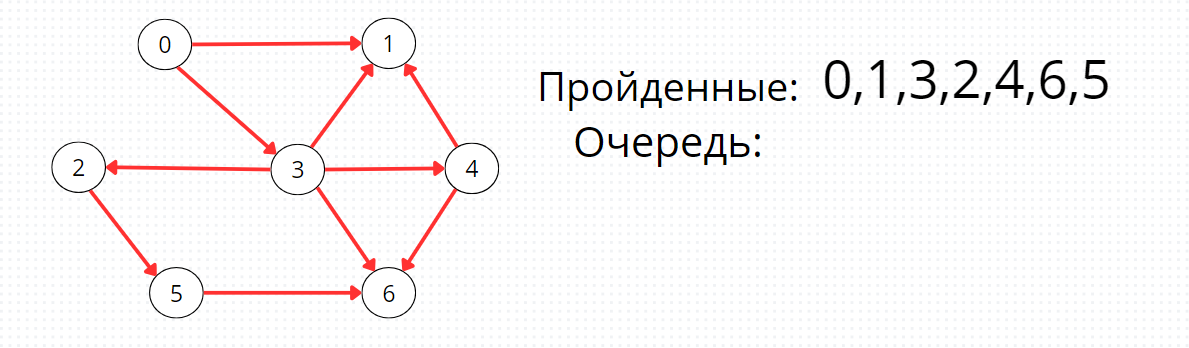
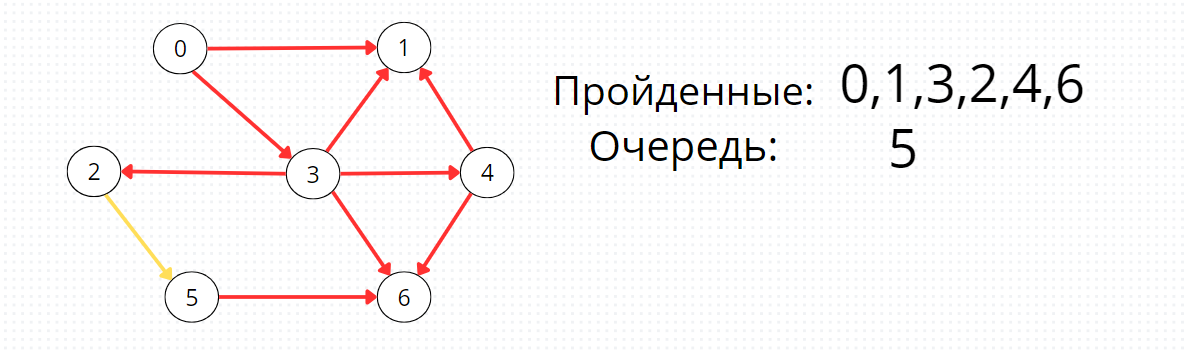
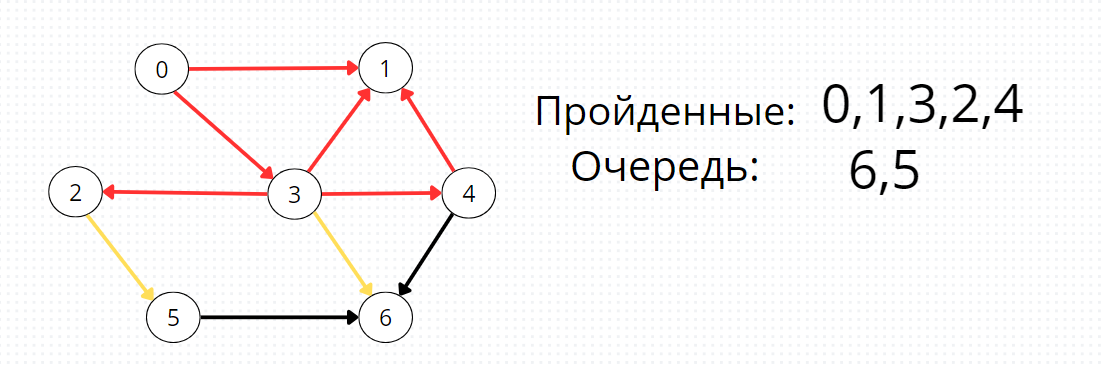
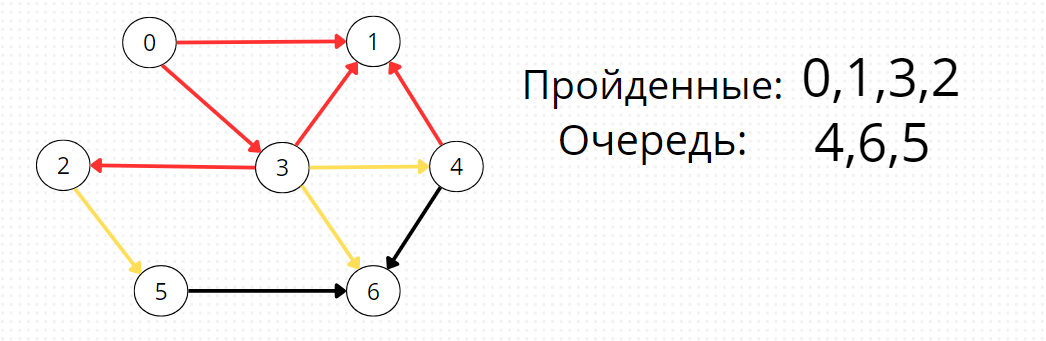
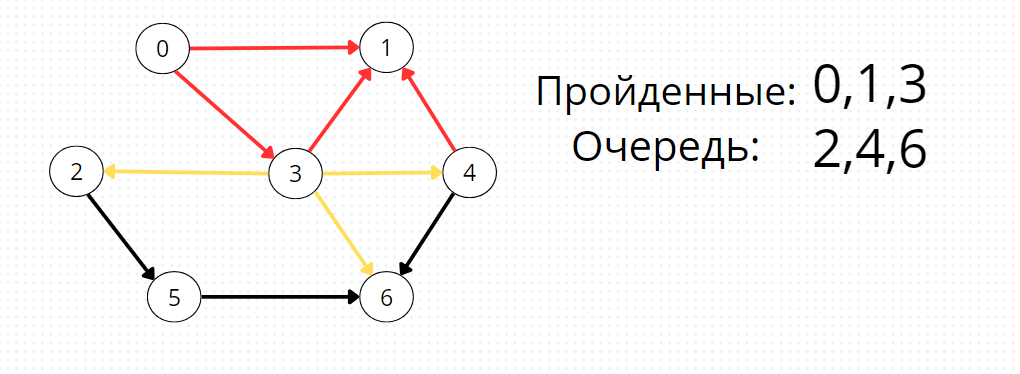
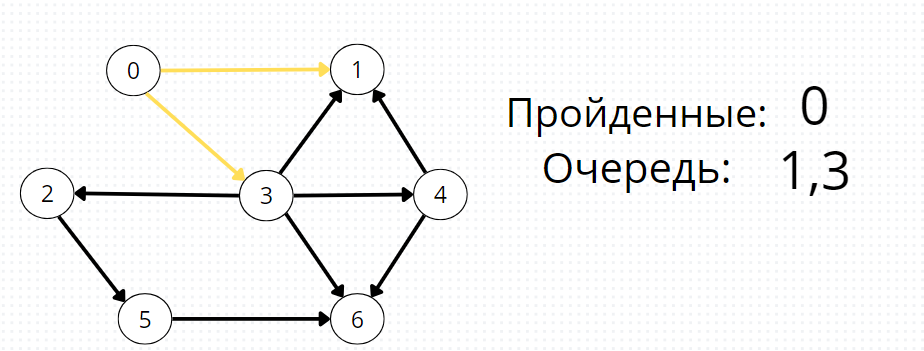
3-1 3-2 3-4 3-6

4-1 4-6

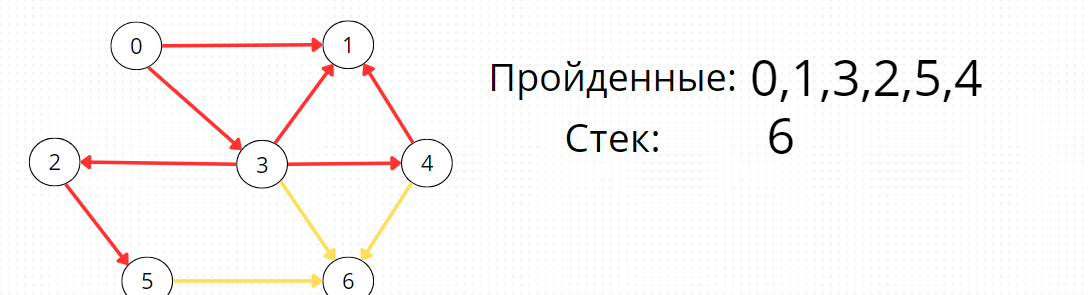
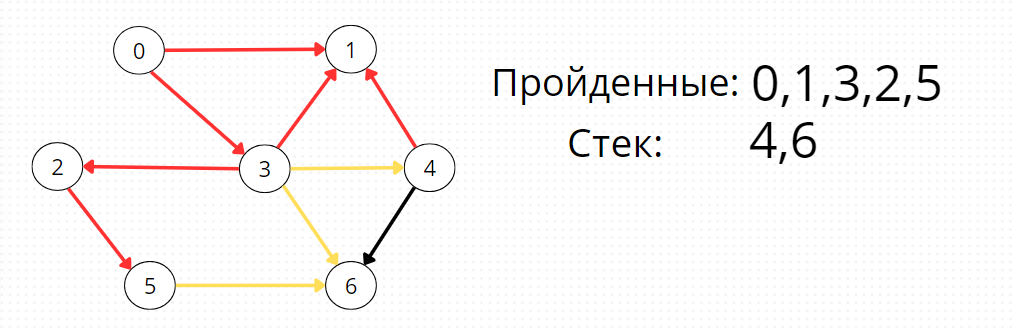
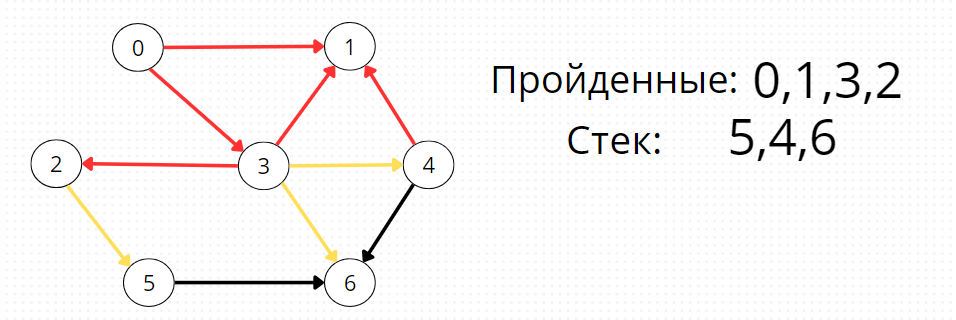
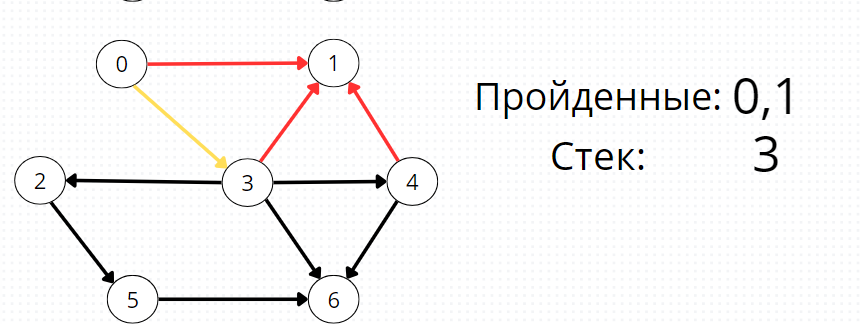
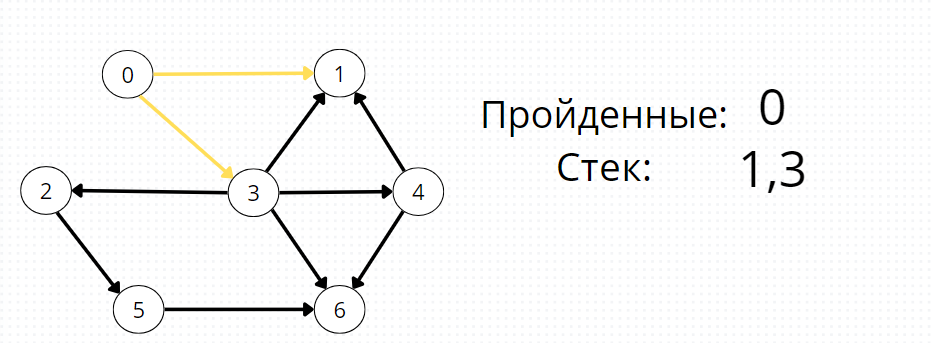
5-6

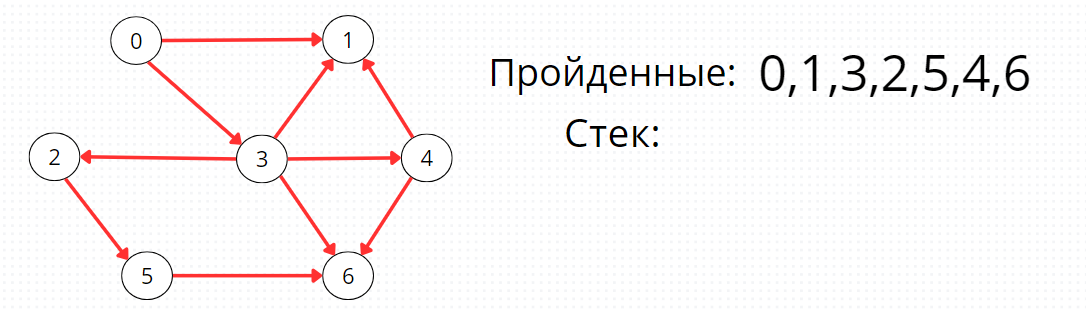
1. BFS:



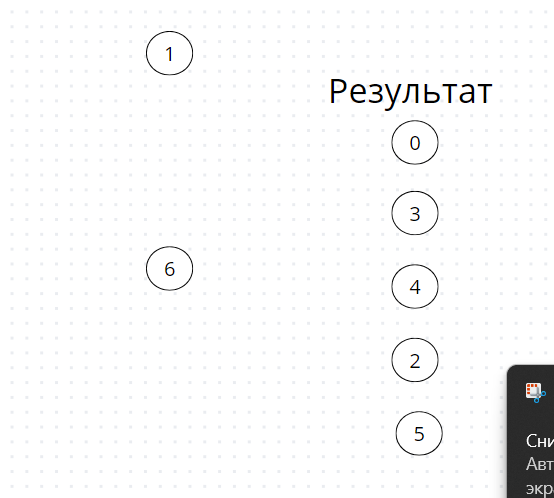
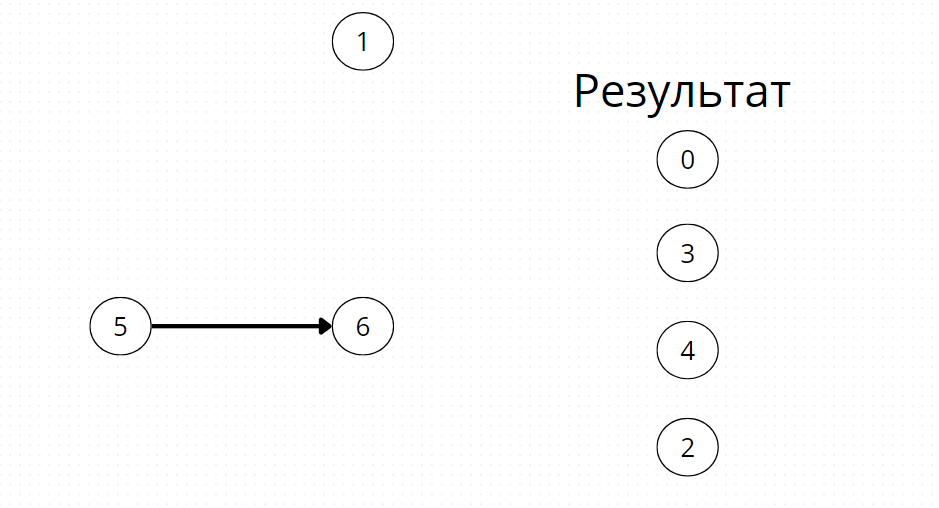
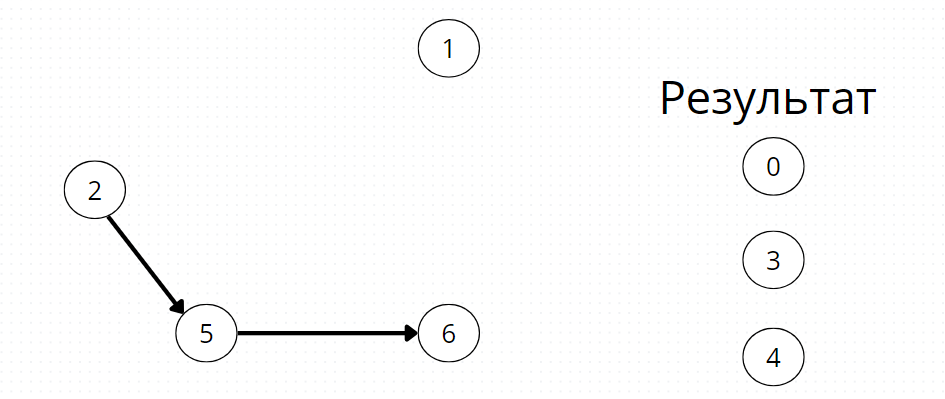
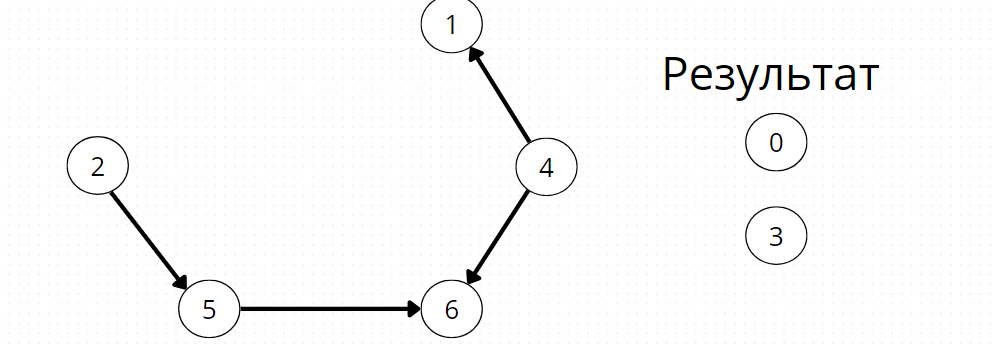
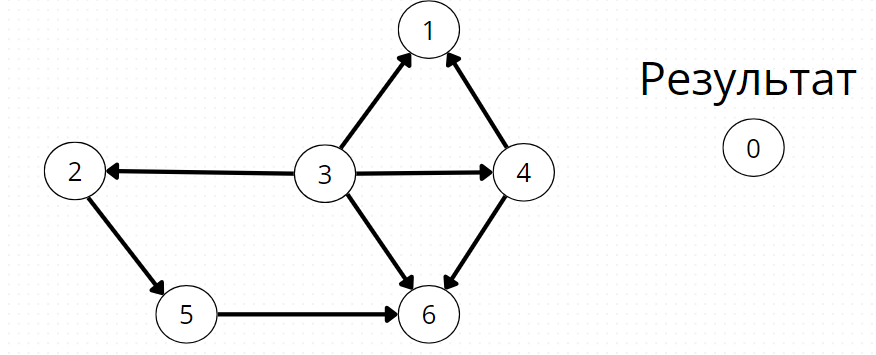
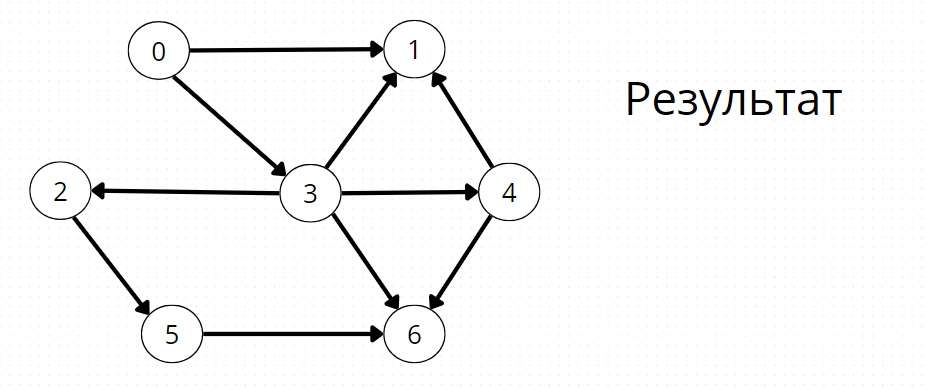


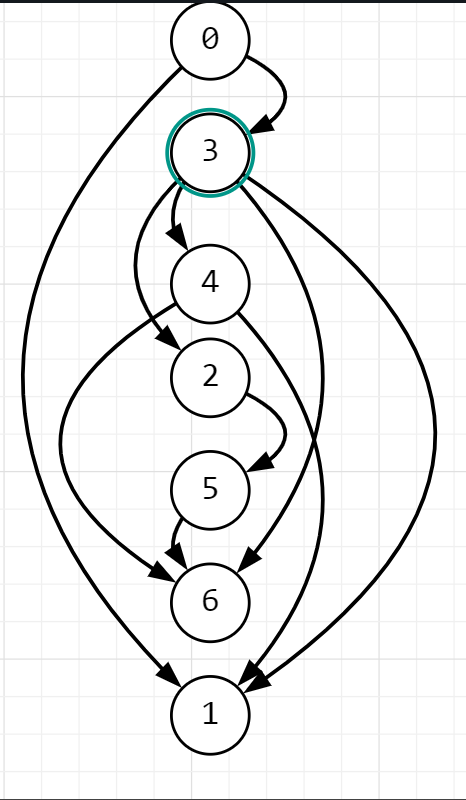
DFS:





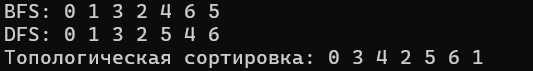
Топологическая сортировка:





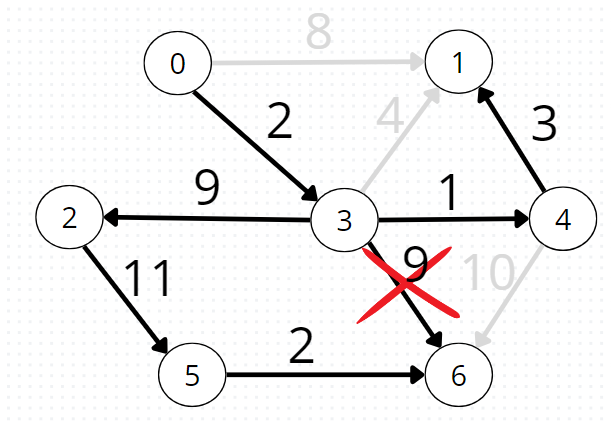
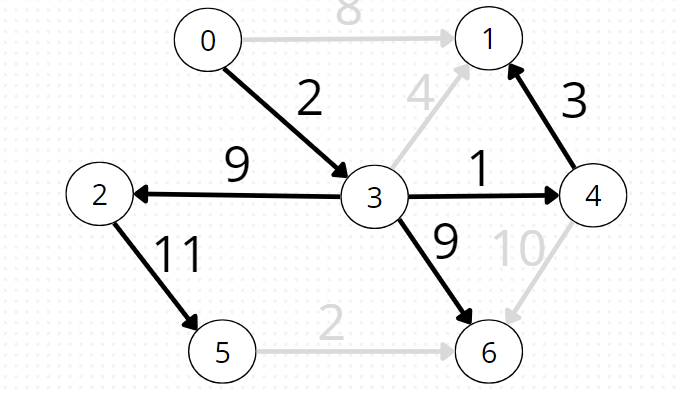
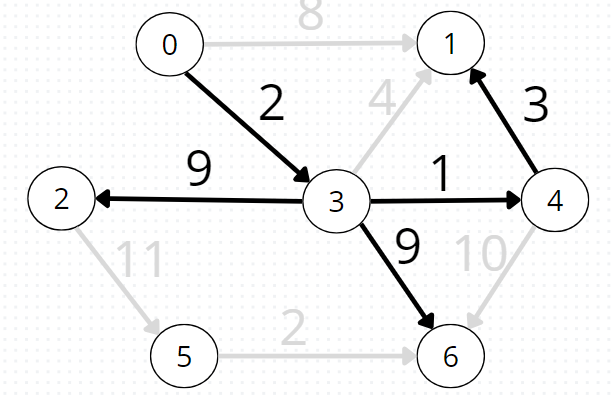
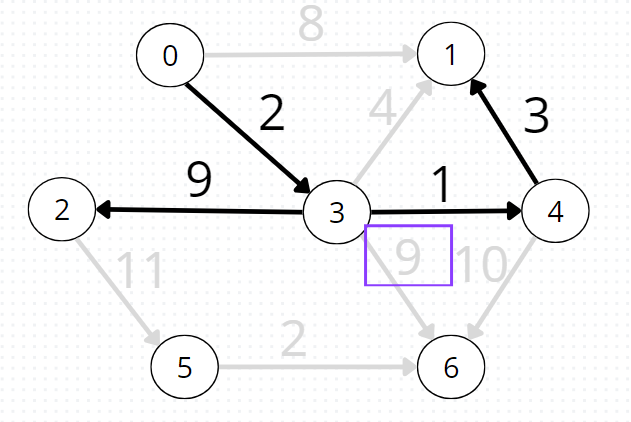
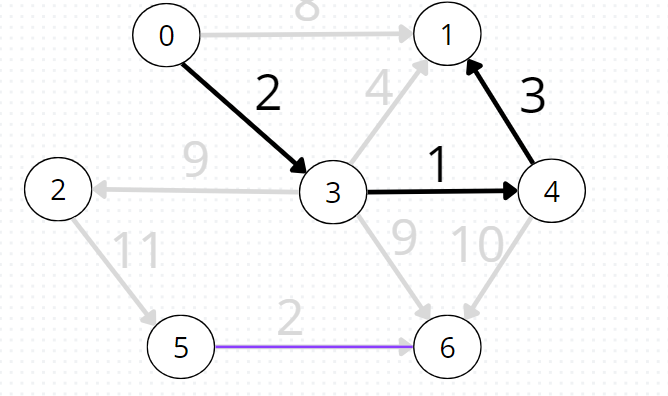
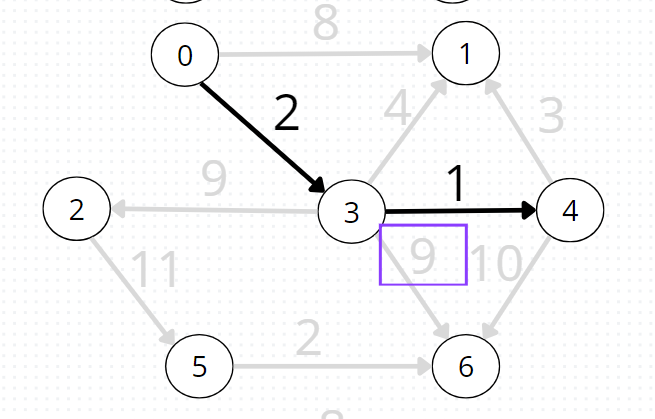
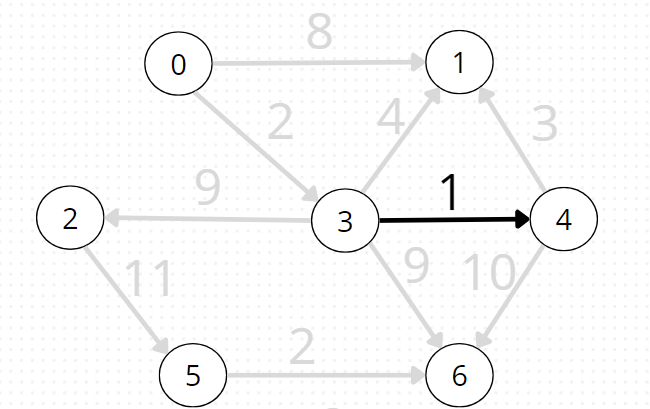
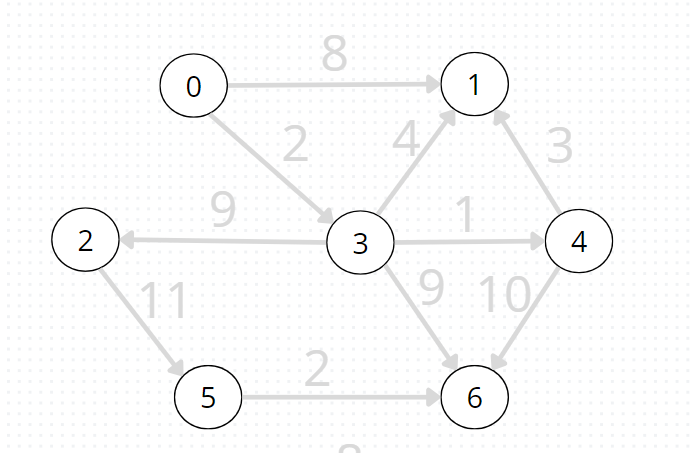
3,4,5:

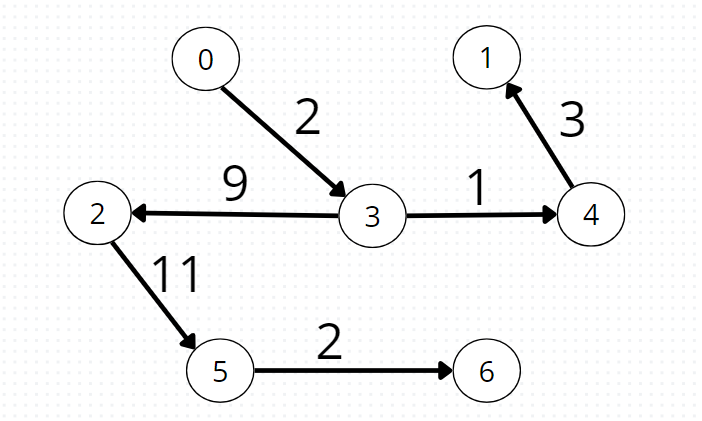
|  |
| --- |
| #include <iostream>  #include <vector>  #include <queue>  #include <stack>  using namespace std;  struct AMatrix  {  vector<vector<bool>> mat;  int vertices;  AMatrix(int n) : vertices(n), mat(n, vector<bool>(n, false)) {}  void addEdge(int u, int v)  {  mat[u][v] = true;  }  void print()  {  cout << "Матрица смежности:\n";  for (int i = 0; i < vertices; ++i)  {  for (int j = 0; j < vertices; ++j)  {  cout << mat[i][j] << " ";  }  cout << "\n";  }  }  };  struct AList  {  vector<vector<int>> adj;  int vertices;  AList(int n) : vertices(n), adj(n) {}  void addEdge(int u, int v)  {  adj[u].push\_back(v);  }  void print()  {  cout << "Список смежности:\n";  for (int i = 0; i < vertices; ++i)  {  cout << i << ": ";  for (int v : adj[i])  {  cout << v << " ";  }  cout << "\n";  }  }  };  AMatrix listToMatrix(const AList& list)  {  AMatrix matrix(list.vertices);  for (int u = 0; u < list.vertices; ++u)  {  for (int v : list.adj[u])  {  matrix.addEdge(u, v);  }  }  return matrix;  }  AList matrixToList(const AMatrix& matrix)  {  AList list(matrix.vertices);  for (int u = 0; u < matrix.vertices; ++u)  {  for (int v = 0; v < matrix.vertices; ++v)  {  if (matrix.mat[u][v])  {  list.addEdge(u, v);  }  }  }  return list;  }  void BFS(const AList& graph, int start)  {  vector<bool> visited(graph.vertices, false);  queue<int> q;  q.push(start);  visited[start] = true;  cout << "BFS: ";  while (!q.empty())  {  int u = q.front();  q.pop();  cout << u << " ";  for (int v : graph.adj[u])  {  if (!visited[v])  {  visited[v] = true;  q.push(v);  }  }  }  cout << "\n";  }  void DFS(const AList& graph, int start)  {  vector<bool> visited(graph.vertices, false);  stack<int> s;  s.push(start);  visited[start] = true;  cout << "DFS: ";  while (!s.empty())  {  int u = s.top();  s.pop();  cout << u << " ";  for (auto it = graph.adj[u].rbegin(); it != graph.adj[u].rend(); ++it)  {  int v = \*it;  if (!visited[v])  {  visited[v] = true;  s.push(v);  }  }  }  cout << "\n";  }  void topologicalSortUtil(const AList& graph, int u, vector<bool>& visited, stack<int>& st)  {  visited[u] = true;  for (int v : graph.adj[u])  {  if (!visited[v])  {  topologicalSortUtil(graph, v, visited, st);  }  }  st.push(u);  }  void topologicalSort(const AList& graph)  {  vector<bool> visited(graph.vertices, false);  stack<int> st;  for (int i = 0; i < graph.vertices; ++i)  {  if (!visited[i])  {  topologicalSortUtil(graph, i, visited, st);  }  }  cout << "Топологическая сортировка: ";  while (!st.empty())  {  cout << st.top() << " ";  st.pop();  }  cout << "\n";  }  int main()  {  setlocale(LC\_ALL, "rus");  AList list(7);  list.addEdge(0, 1);  list.addEdge(0, 3);  list.addEdge(1, 4);  list.addEdge(3, 1);  list.addEdge(3, 2);  list.addEdge(3, 5);  list.addEdge(4, 6);  list.addEdge(5, 2);  list.addEdge(6, 5);  AMatrix matrix = listToMatrix(list);  AList newList = matrixToList(matrix);  list.print();  matrix.print();  newList.print();  BFS(list, 0);  DFS(list, 0);  topologicalSort(list);  return 0;  } |

Пример работы: 

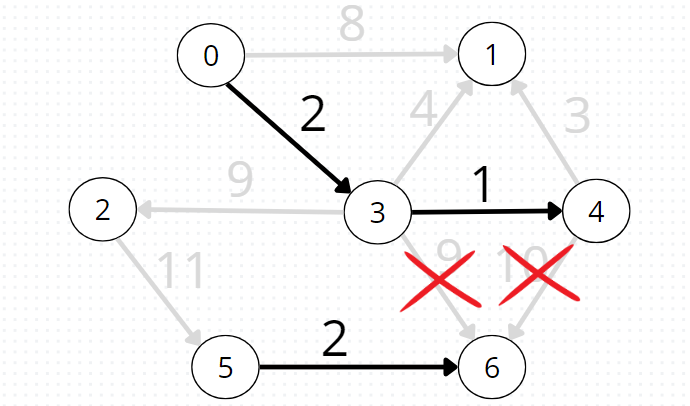
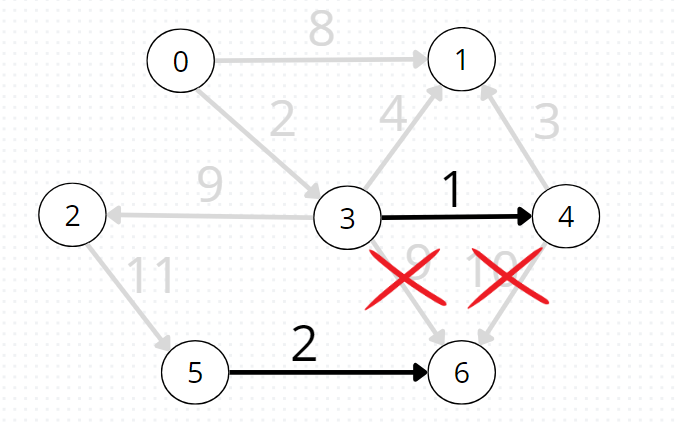
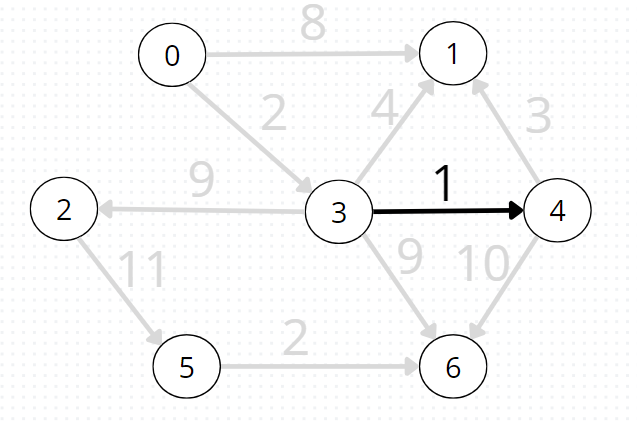
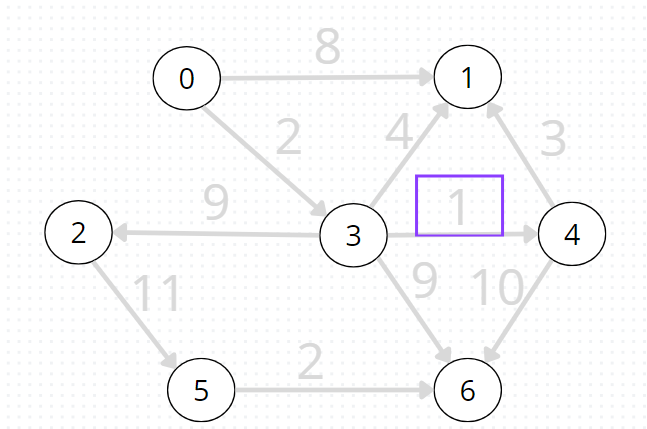
6:

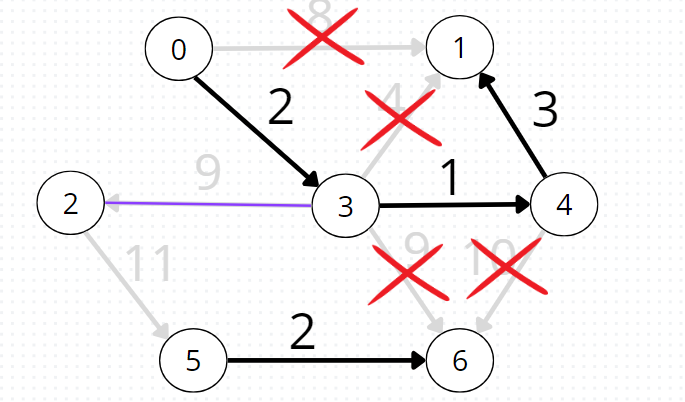
Алгоритм Примма:



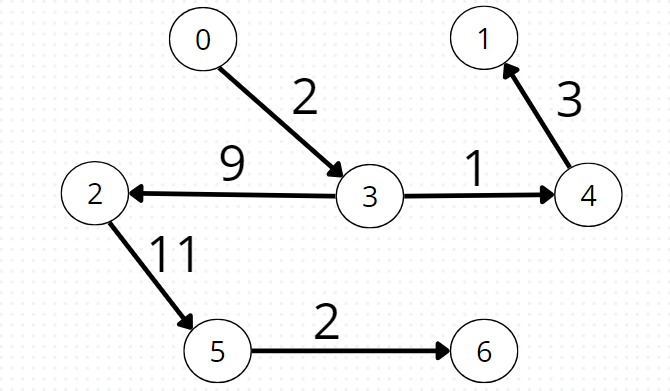
Итоговое минимальное остовное дерево 

Алгоритм Крускала:





Итоговое остовное дерево:



**Лабораторная работа №7. «Сетевые модели»**

**1,2. Структурное и календарное планирование.**

Тема: «Создание банковского приложения».

|  |  |  |  |
| --- | --- | --- | --- |
| Код  операции | Наименование операции | Предшествующие операции | t |
| I. Анализ | | | |
| Z1 | Системный анализ | - | 5 |
| Z2 | Анализ требований | Z1 | 5 |
| Z3 | Выбор поддерживаемых операций и данных | Z1 | 2 |
| Z4 | Определение бюджета и графика работ | Z2, Z3 | 3 |
| II. ПРОЕКТИРОВАНИЕ 15 | | | |
| Z5 | Проектирование архитектуры системы | Z4 | 3 |
| Z6 | Проектирование системы транзакций | Z4 | 3 |
| Z7 | Проектирование базы данных | Z2 | 4 |
| Z8 | Проектирование интерфейсов пользователей | Z3 | 2 |
| III. КОДИРОВАНИЕ 27 | | | |
| Z9 | Создание основного функционала | Z5, Z8 | 5 |
| Z10 | Реализация транзакций | Z5, Z6, Z19 | 7 |
| Z11 | Кодирование процедур СУБД | Z5, Z6,Z7 | 4 |
| IV. ТЕСТИРОВАНИЕ 43 | | | |
| Z12 | Функциональное тестирование | Z9, Z10, Z11, Z19 | 3 |
| Z13 | Структурное тестирование | Z10,Z11 | 2 |
| Z14 | Оптимизация системы | Z12,Z13 | 5 |
| V. ВНЕДРЕНИЕ 53 | | | |
| Z15 | Разработка документации | Z12,Z13 | 2 |
| Z16 | Обучение пользователей | Z15 | 1 |
| Z17 | Испытание | Z12, Z13, Z14 Z15, Z16 | 5 |
| Z18 | Завершение работ | Z17 | 1 |
| VI. ДОПОЛНИТЕЛЬНЫЕ РАБОТЫ 62 | | | |
| Z19 | Криптографическая защита | Z1,Z6 | 3 |

65

**3. Сетевой график, нахождение критического пути.**

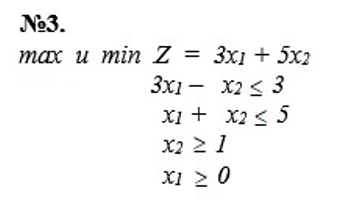


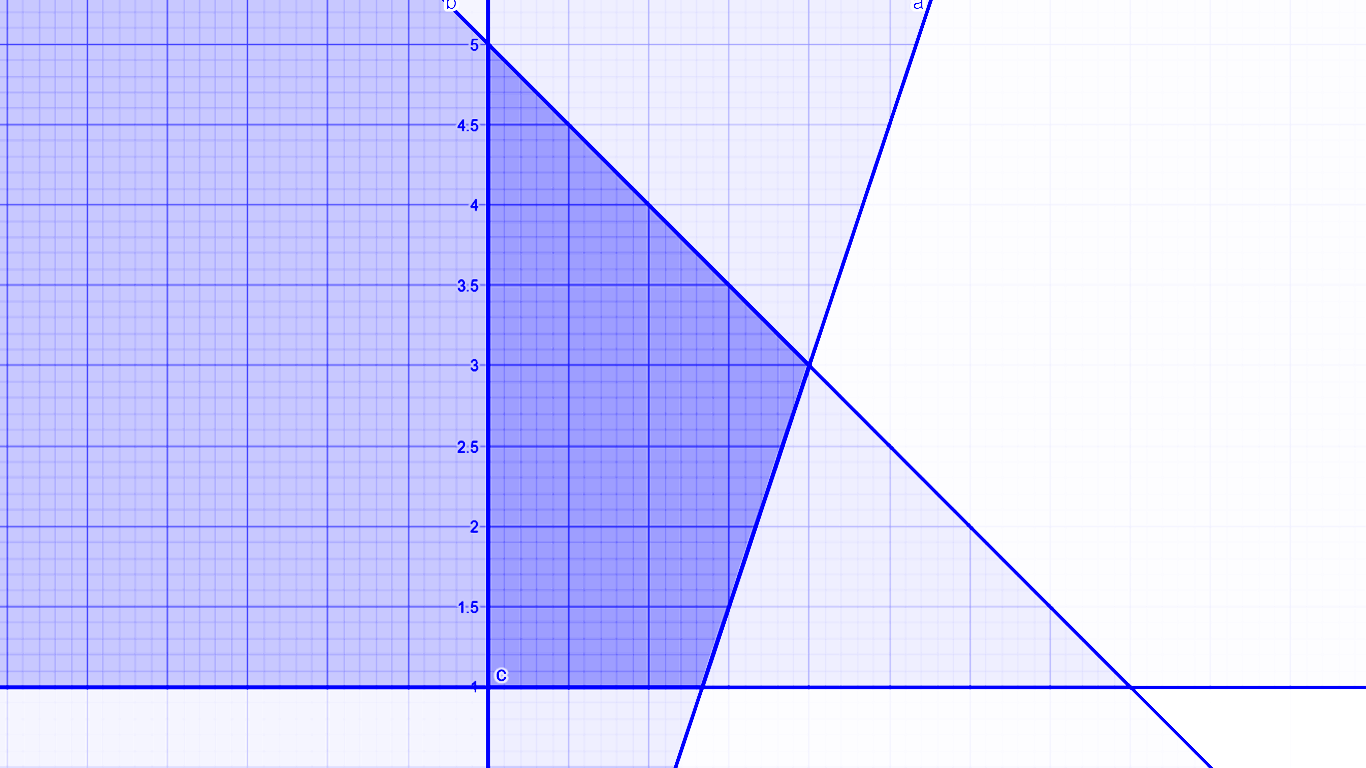
Длина критического пути: **45 дней**. Последовательность задач, принадлежащих критическому пути: **Z1 → Z2 → Z4 → Z6 → Z19 → Z10 → Z12 → Z14 → Z17 → Z18**

**Вывод:** в ходе выполнения лабораторной работы был составлен структурный и календарный план создания банковского приложения. Кроме того, был составлен сетевой график и найден критический путь

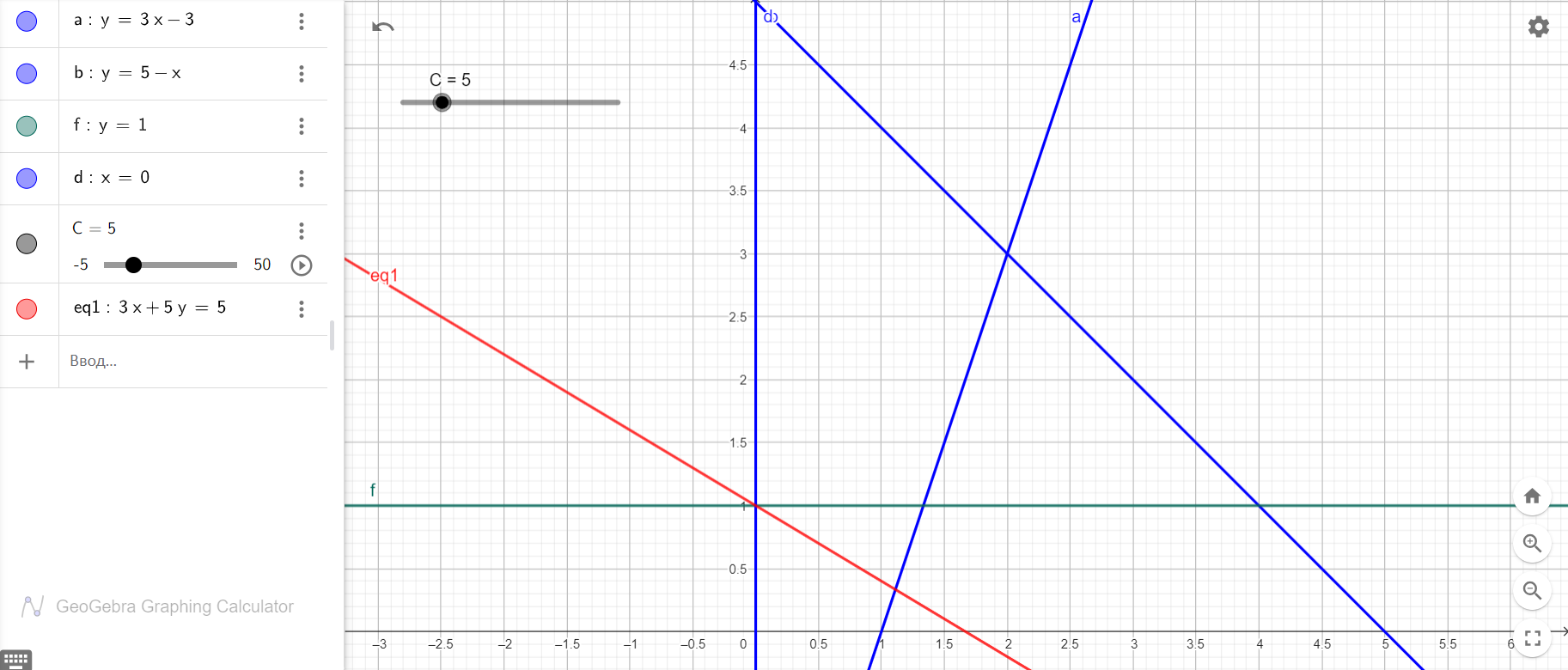
**Лабораторная работа №8. «Графический метод решения задач оптимизации»**

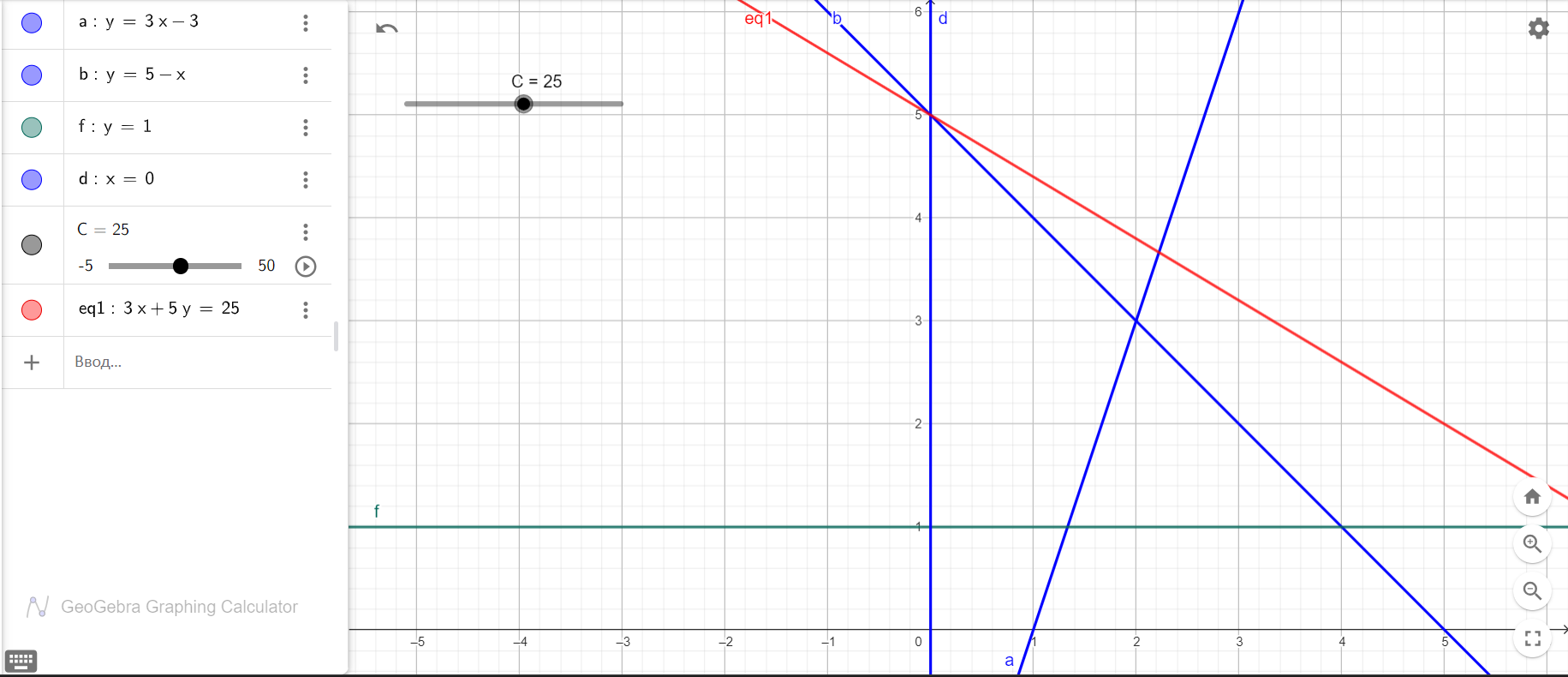
**Цель работы:** освоить решение задач графическим методом.

****

****

Для того что определить минимальное значение в данной области, нужно построить график прямой 3x+5y и приравнять к C. Затем построенную прямую двигаем влево к самой крайней точке выделенной фигуры и ищем координаты точки, в которой находится минимальное значение. Данная точка имеет координаты (0; 1). Теперь данную точку подставляем в 3x+5y. Получаем:5. Минимальное значение данной функции получилось 5. Оно совпадает с тем значением, которое получилось графически.

 Для того что определить максимальное значение в данной области, нужно построить график прямой 3x+5y и приравнять к C. Затем построенную прямую двигать вправо к самой крайней точке выделенной фигуры и ищем координаты точки, в которой находится максимальное значение. Данная точка имеет координаты (0;5). Теперь данную точку подставляем в 3x+5y. Получаем: 25 Максимальное значение данной функции получилось 25. Оно совпадает с тем значением, которое получилось графически.



**Вывод:** научились решать задачи графическим способом.