**Белорусский государственный технологический университет**

**Факультет информационных технологий**

**Специальность программная инженерия**

Отчёты по лабораторным работам

По дисциплине «Математическое программирование»

Выполнил:

Студент 2 курса 8 группы

Мамонько Денис Александрович

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.

***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.

Перед тем, как начать работу, мы должны установить предкомпилированный заголовок. Для этого нужно создать файлы pch.h и pch.cpp. По умолчанию данных файлов проекта нет. В pch.h пишем все используемые в данное проекте библиотеки. В pch.cpp пишем только подключенный предкомпилированный файл. В свойствах проекта предкомпилированный заголовок stdafx.h меняем на pch.h, а в свойствах pch.cpp устанавливаем галочку на Yc (Создать). Далее приступаем непосредственно к работе. В задании 1 у нас были прототипы функций и сама их реализация, в листинге ниже приведен пример кода.

|  |
| --- |
| #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

|  |
| --- |
| #include "pch.h"  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 1.2 – 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 "pch.h"  #define CYCLE 1000000 // количество циклов  int main()  {  setlocale(LC\_ALL, "rus");  double av1 = 0, av2 = 0;  clock\_t t1 = 0, t2 = 0;  auxil::start(); // старт генерации  t1 = clock(); // фиксация времени  for (int i = 0; i < CYCLE; i++)  {  av1 += (double)auxil::iget(-100, 100); // сумма случайных чисел  av2 += auxil::dget(-100, 100); // сумма случайных чисел  }  t2 = clock(); // фиксация времени  std::cout << std::endl << "количество циклов: " << CYCLE;  std::cout << std::endl << "среднее значение (int): " << av1 / CYCLE;  std::cout << std::endl << "среднее значение (double): " << av2 / CYCLE;  std::cout << std::endl << "продолжительность (у.е): " << (t2 - t1);  std::cout << std::endl << " (сек): "  << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 2.1 – Файл main.cpp

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

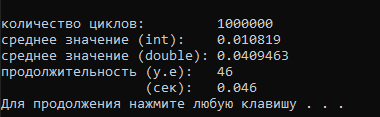


Рисунок 1 – Результат работы программы

**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.

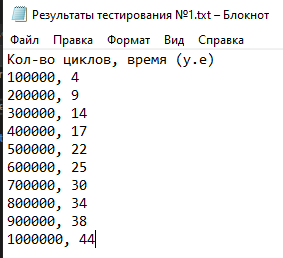


Рисунок 2 – Результаты тестирования №1



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  int fibonachi(int n); |

Листинг 3.1 – Файл Fibonachi.h

|  |
| --- |
| #include "pch.h"  using namespace std;  int fibonachi(int n)  {  if (n == 0)  return 0;  if (n == 1)  return 1;  return fibonachi(n - 1) + fibonachi(n - 2);  } |

Листинг 3.2 – Файл Fibonachi.cpp

|  |
| --- |
| #include "pch.h"  #define CYCLE 1000000 // количество циклов  int main()  {  setlocale(LC\_ALL, "rus");  cout << "\t\tЧисла Фибоначчи" << endl;  int N;  clock\_t t3 = 0, t4 = 0;  cout << "Введите количество чисел N: ";  cin >> N;  t3 = clock();  int result = fibonachi(N);  cout << "Результат функции чисел Фибоначчи: " << result;  t4 = clock();  std::cout << std::endl << "продолжительность (у.е): " << (t4 - t3);  std::cout << std::endl << " (сек): "  << ((double)(t4 - t3)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 3.3 – Файл main.cpp

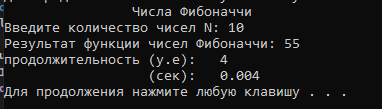


Рисунок 4 – Результат работы программы

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 10 до 40 с шагом 5.

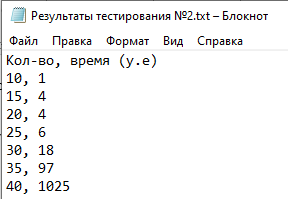


Рисунок 5 – Результаты тестирования №2

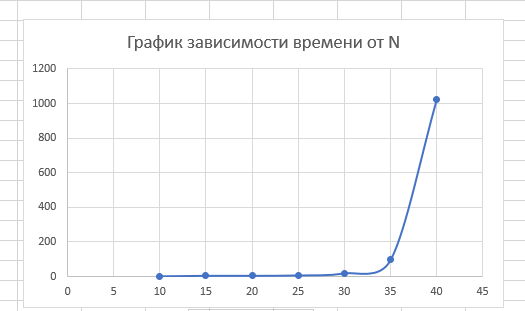


Рисунок 6 – График зависимости времени от N

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

**ИТОГ**

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**

**Вариант 7**

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

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

**Генератор подмножеств** – это инструмент или алгоритм, который помогает создать все возможные подмножества заданного множества. **Подмножество** – это набор элементов, которые принадлежат исходному множеству, но не обязательно включают все элементы этого 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 "Combi.h"  int main() // с помощью инкременирования битовой маски  {  setlocale(LC\_ALL, "rus");  char AA[][2] = { "A", "B", "C", "D" };  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 Main.cpp

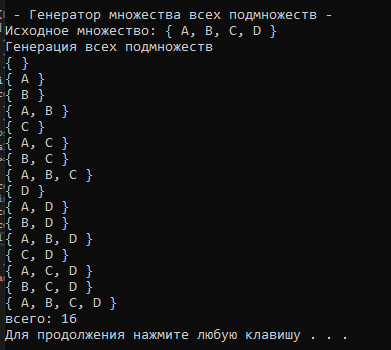


Рисунок 1 – Результат работы программы

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

**Генератор сочетаний** – это инструмент или алгоритм, который помогает создать все возможные комбинации элементов заданного множества определенного размера. Сочетания не зависят от порядка элементов.

|  |
| --- |
| #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 "Combi.h"  int main() // тоже битовая маска, но берем только подмножества с N двоичными единицами  {  setlocale(LC\_ALL, "rus");  char AA[][2] = { "A", "B", "C", "D", "E" };  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 Main.cpp

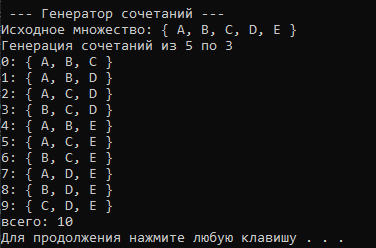


Рисунок 2 – Результат работы программы

**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 <iomanip>  int main() // алгоритм Джонсона – Троттера  {  setlocale(LC\_ALL, "rus");  char AA[][2] = { "A", "B", "C", "D" };  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 Main.cpp

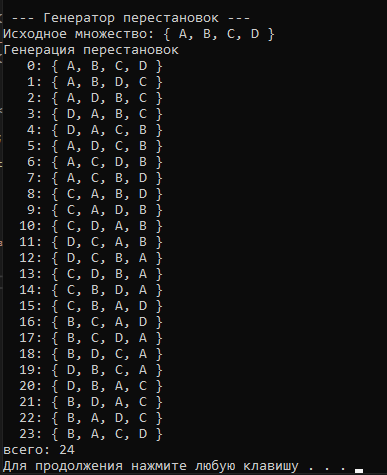


Рисунок 3 – Результат работы программы

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

**Генератор размещений** – это инструмент или алгоритм, который помогает создать все возможные размещения элементов заданного множества определенного размера.

**Размещение** – это упорядоченная комбинация элементов множества, где важен порядок и количество элементов в каждом размещении. В отличие от сочетаний, размещения зависят от порядка и могут повторять элементы (если это допускается).

|  |
| --- |
| #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::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 <iomanip>  #include "Combi.h"  #define N (sizeof(AA)/2)  #define M 3  int main()  {  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 < M; 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 Main.cpp

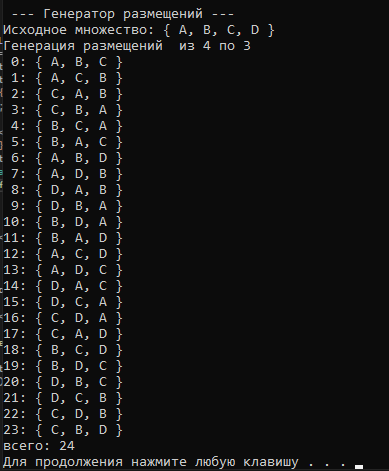


Рисунок 4 – Результат работы программы

**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 "../Combinations/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 "../Combinations/Combi.cpp"  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 <iomanip>  #include "Boat.h"  #define NN (sizeof(v)/sizeof(int))  #define MM 3  int wmain()  {  setlocale(LC\_ALL, "rus");  int V = 1000,  v[] = { 100, 200, 300, 400, 500, 150 },  c[NN] = { 10, 15, 20, 25, 30, 25 };  short r[MM];  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  );  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;  system("pause");  return 0;  } |

Листинг 5.3 – Файл Main.cpp

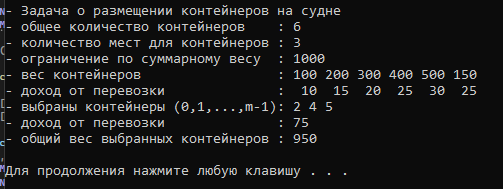


Рисунок 5 – Результат работы программы

**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.

В этой задаче описание файлов Boat.cpp и Boat.h будет совпадать, отличаться будет лишь Main.cpp.

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include "Boat.h"  #include <time.h>  #define NN (sizeof(v)/sizeof(int))  #define MM 6  #define SPACE(n) std::setw(n)<<" "  int wmain()  {  setlocale(LC\_ALL, "rus");  int V = 1000,  v[] = { 250, 560, 670, 400, 200, 270, 370, 330, 330, 440, 530, 120,  200, 270, 370, 330, 330, 440, 700, 120, 550, 540, 420, 170,  600, 700, 120, 550, 540, 420, 430, 140, 300, 370, 310, 120 };  int c[NN] = { 15,26, 27, 43, 16, 26, 42, 22, 34, 12, 33, 30,  42,22, 34, 43, 16, 26, 14, 12, 25, 41, 17, 28,  12,45, 60, 41, 33, 11, 14, 12, 25, 41, 30, 40 };  short r[MM];  int maxcc = 0;  clock\_t t1, t2;  std::cout << std::endl << "-- Задача об оптимальной загрузке судна -- ";  std::cout << std::endl << "- ограничение по весу : " << V;  std::cout << std::endl << "- количество мест : " << MM;  std::cout << std::endl << "-- количество ------ продолжительность -- ";  std::cout << std::endl << " контейнеров вычисления ";  for (int i = 24; i <= NN; i++)  {  t1 = clock();  int maxcc = boat(V, MM, i, v, c, r);  t2 = clock();  std::cout << std::endl << SPACE(7) << std::setw(2) << i  << SPACE(15) << std::setw(5) << (t2 - t1);  }  std::cout << std::endl << std::endl;  system("pause");  return 0;  } |

Листинг 6.1 – Файл Main.cpp

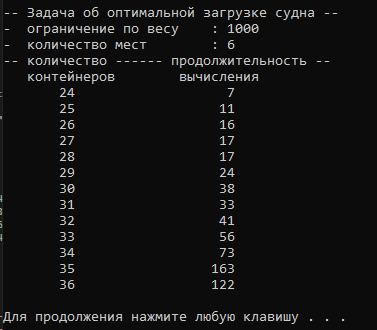


Рисунок 6 – Результат работы программы



Рисунок 7 – График зависимости времени вычисления от количества контейнеров

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

**Вариант 7**

**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.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Город | 1 | 2 | 3 | 4 | 5 |
| 1 | INF | 14 | 28 | INF | 7 |
| 2 | 7 | INF | 22 | 61 | 77 |
| 3 | 9 | 21 | INF | 86 | 56 |
| 4 | 24 | 51 | 28 | INF | 21 |
| 5 | 86 | 73 | 52 | 20 | INF |

**Алгоритм коммивояжёра** – это классическая задача оптимизации, целью которой является нахождение кратчайшего маршрута, который позволяет посетить каждый город ровно один раз и вернуться в исходный город. В нашей задаче мы будем работать с матрицей городов, где в качестве городов выступают вершины, а расстояния между ними – ребра с каким-либо числовым весом. Также на некоторых рёбрах вес отсутствует и это значение помечено как INF. Это значит, что расстояние между городами отсутствует. Матрицу расстояний по-другому еще называют ***графом***.

**Граф** – структура данных, состоящая из объектов, называемых вершинами, и связей между этими вершинами, которые называются ребрами.

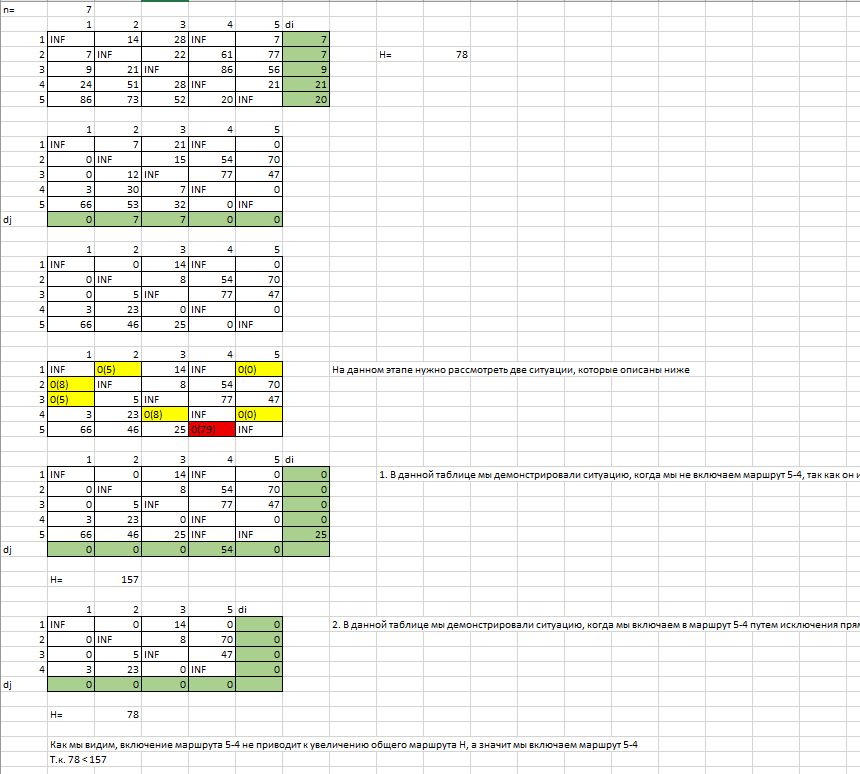
Решение задачи коммивояжера с параметром включает в себя использование ***цикла Гамильтона***.

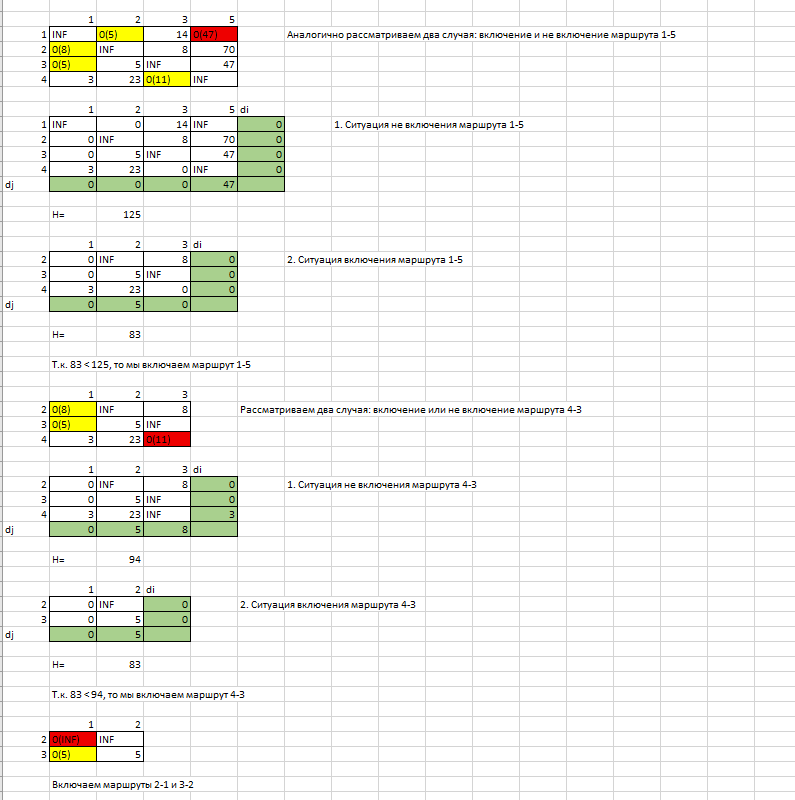
**Цикл Гамильтона** – маршрут, который проходит каждую вершину графа ровно один раз и возвращается в исходную вершину.  
Результатом задачи является нахождения кратчайшего пути в цикле Гамильтона.

**Стоимость пути (кратчайший путь)** – сумма длин всех ребер в цикле Гамильтона.

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

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





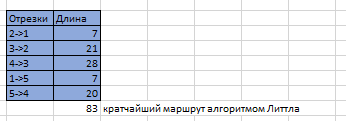


Рисунок 1 – Решение алгоритма Литтла вручную

В результате решения данной задачи, мы получили, что оптимальным маршрутом является следующий: **1->5->4->3->2->1** и его длина составляет **83**.

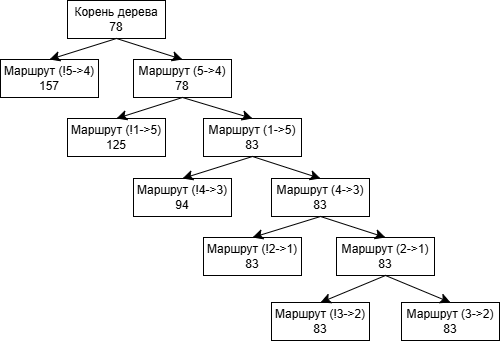


Рисунок 2 – Обход исходной матрицы

**Задание 2\*.** Предложить решение задачи коммивояжера методом вервей и границ на языке C++. Код, анализ и вывод включить в отчет.

|  |
| --- |
| #include <iostream>  #include <limits>  using namespace std;  const int N = 5;  int final\_path[N + 1];  bool visited[N];  int final\_res = INT\_MAX;  void copyToFinal(int curr\_path[])  {  for (int i = 0; i < N; i++)  final\_path[i] = curr\_path[i];  final\_path[N] = curr\_path[0];  }  int firstMin(int matrix[N][N], int i)  {  int min = INT\_MAX;  for (int k = 0; k < N; k++)  if (matrix[i][k] < min && i != k)  min = matrix[i][k];  return min;  }  int secondMin(int matrix[N][N], int i)  {  int first = INT\_MAX, second = INT\_MAX;  for (int j = 0; j < N; j++)  {  if (i == j)  continue;  if (matrix[i][j] <= first)  {  second = first;  first = matrix[i][j];  }  else if (matrix[i][j] <= second &&  matrix[i][j] != first)  second = matrix[i][j];  }  return second;  }  void TSPRec(int matrix[N][N], int curr\_bound, int curr\_weight,  int level, int curr\_path[])  {  if (level == N)  {  if (matrix[curr\_path[level - 1]][curr\_path[0]] != 0)  {  int curr\_res = curr\_weight +  matrix[curr\_path[level - 1]][curr\_path[0]];  if (curr\_res < final\_res)  {  copyToFinal(curr\_path);  final\_res = curr\_res;  }  }  return;  }  for (int i = 0; i < N; i++)  {  if (matrix[curr\_path[level - 1]][i] != 0 &&  visited[i] == false)  {  int temp = curr\_bound;  curr\_weight += matrix[curr\_path[level - 1]][i];  if (level == 1)  curr\_bound -= ((firstMin(matrix, curr\_path[level - 1]) +  firstMin(matrix, i)) / 2);  else  curr\_bound -= ((secondMin(matrix, curr\_path[level - 1]) +  firstMin(matrix, i)) / 2);  if (curr\_bound + curr\_weight < final\_res)  {  curr\_path[level] = i;  visited[i] = true;  TSPRec(matrix, curr\_bound, curr\_weight, level + 1,  curr\_path);  }  curr\_weight -= matrix[curr\_path[level - 1]][i];  curr\_bound = temp;  memset(visited, false, sizeof(visited));  for (int j = 0; j <= level - 1; j++)  visited[curr\_path[j]] = true;  }  }  }  void TSP(int matrix[N][N])  {  int curr\_path[N + 1];  int curr\_bound = 0;  memset(curr\_path, -1, sizeof(curr\_path));  memset(visited, 0, sizeof(curr\_path));  for (int i = 0; i < N; i++)  curr\_bound += (firstMin(matrix, i) +  secondMin(matrix, i));  curr\_bound = (curr\_bound & 1) ? curr\_bound / 2 + 1 :  curr\_bound / 2;  visited[0] = true;  curr\_path[0] = 0;  TSPRec(matrix, curr\_bound, 0, 1, curr\_path);  }  int main()  {  setlocale(LC\_ALL, "RU");  int matrix[N][N] = {  {0, 14, 28, 0, 7},  {7, 0, 22, 61, 77},  {9, 21, 0, 86, 56},  {24, 51, 28, 0, 21},  {86, 73, 52, 20, 0},  };  TSP(matrix);  cout << "Длина минимального маршрута : " << final\_res << endl;  cout << "Последовательность вершин : ";  for (int i = 0; i <= N; i++)  cout << final\_path[i] << " ";  return 0;  } |

Листинг 2.1 – Алгоритм Литтла



Рисунок 3 – Результат работы программы

**Задание 3.** Проверить полученное решение при помощи генератора перестановок (см. лаб. 2, задание 5.1.) и включить копию экрана с решением в отчет.

|  |
| --- |
| #pragma once  namespace combi  {  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;  };  }; |

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

|  |
| --- |
| #pragma once  #define INF 0x7fffffff  #include "Combi.h"  int salesman(  int n,  const int\* d,  int\* r  ); |

Листинг 3.3 – Файл TSP.h

|  |
| --- |
| #include "TSP.h"  int sum(int x1, int x2)  {  return (x1 == INF || x2 == INF) ? INF : (x1 + x2);  };  int\* firstpath(int n)  {  int\* rc = new int[n + 1];  rc[n] = 0;  for (int i = 0; i < n; i++)  rc[i] = i;  return rc;  };  int\* source(int n)  {  int\* rc = new int[n - 1];  for (int i = 1; i < n; i++) rc[i - 1] = i;  return rc;  };  void copypath(int n, int\* r1, const int\* r2)  {  for (int i = 0; i < n; i++)  r1[i] = r2[i];  };  int distance(int n, int\* r, const int\* d)  {  int rc = 0;  for (int i = 0; i < n - 1; i++)  rc = sum(rc, d[r[i] \* n + r[i + 1]]);  return sum(rc, d[r[n - 1] \* n + 0]);  };  void indx(int n, int\* r, const int\* s, const short\* ntx)  {  for (int i = 1; i < n; i++)  r[i] = s[ntx[i - 1]];  }  int salesman(  int n,  const int\* d,  int\* r  )  {  int\* s = source(n), \* b = firstpath(n), rc = INF, dist = 0;  combi::permutation p(n - 1);  int k = p.getfirst();  while (k >= 0)  {  indx(n, b, s, p.sset);  if ((dist = distance(n, b, d)) < rc)  {  rc = dist; copypath(n, r, b);  }  k = p.getnext();  };  return rc;  } |

Листинг 3.4 – Файл TSP.cpp

|  |
| --- |
| #include <iostream>  #include <iomanip>  #include "TSP.h"  #define N 5  using namespace std;  int main()  {  setlocale(LC\_ALL, "rus");  int d[N][N] = {  { INF, 14, 28, INF, 7 },  { 7, INF, 22, 61, 77 },  { 9, 21, INF, 86, 56 },  { 24, 51, 28, INF, 21 },  { 86, 73, 52, 20, INF }  };  int r[N];  int s = salesman(  N,  (int\*)d,  r  );  cout << "\n-- Задача коммивояжера -- ";  cout << "\n-- количество городов: " << N;  cout << "\n-- матрица расстояний : ";  for (int i = 0; i < N; i++)  {  cout << "\n";  for (int j = 0; j < N; j++)  if (d[i][j] != INF) cout << setw(3) << d[i][j] << " ";  else cout << setw(3) << "INF" << " ";  }  cout << "\n-- оптимальный маршрут: ";  for (int i = 0; i < N; i++)  cout << r[i] + 1 << "-->";  cout << 1;  cout << "\n-- длина маршрута : " << s << "\n";  system("pause");  return 0;  } |

Листинг 3.5 – Решение задачи коммивояжера методом генератора перестановок

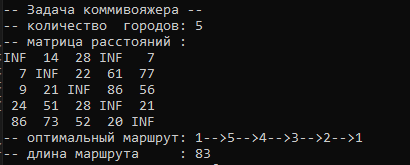


Рисунок 4 – Результат работы программы

Исходя из результатов программ, можно заметить, что решения всеми 3 способами, которые были упомянуты выше, совпали. Был найден одинаковый маршрут и его длина.

**Вывод**: результаты после решения вручную, после использования генератора перестановок и после решения методом ветвей и границ совпали.

**Лабораторная работа №4. Динамическое программирование**

**Вариант 7**

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

***Задание 1.***

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

|  |
| --- |
| #include <iostream>  #include <ctime>  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';  }  return str;  }  int main()  {  setlocale(LC\_ALL, "RU");  char\* str1 = generateRandomString(300);  cout << "Первая строка: ";  for (int i = 0; i < 300; i++) {  if (i % 50 == 0) {  cout << endl;  }  cout << str1[i];  }  cout << endl;  char\* str2 = generateRandomString(200);  cout << "Вторая строка: ";  for (int i = 0; i < 200; i++) {  if (i % 50 == 0) {  cout << endl;  }  cout << str2[i];  }  } |

Листинг 1.1 – Листинг firstTask.cpp

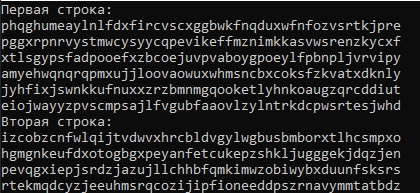
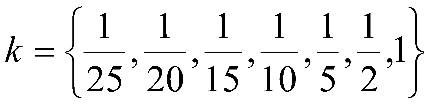


Рисунок 1 – Результат программы

***Задание 2.***

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

Сначала разберем пример динамического программирования.

|  |
| --- |
| #pragma once  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 2.1 – Файл Levenshtein.h

|  |
| --- |
| #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 2.2 – Файл Levenshtein.cpp

|  |
| --- |
| #include <algorithm>  #include <iostream>  #include <ctime>  #include <iomanip>  #include "Levenshtein.h"  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(); levenshtein\_r(i, x, i - 2, y); t2 = clock();  t3 = clock(); levenshtein(i, x, i - 2, y); t4 = clock();  std::cout << std::right << std::setw(2) << 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 2.3 – Файл secondTask.cpp

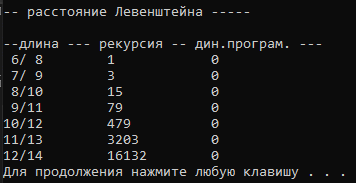


Рисунок 2 – Результат работы программы

Теперь рассмотрим пример исключительно рекурсивного алгоритма, а затем сравним результаты.

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <iomanip>  #include <algorithm>  int min3(int x1, int x2, int x3) {  return std::min(std::min(x1, x2), x3);  }  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, t2;  char x[] = "abcdefghklmnoxm", y[] = "xyabcdefghomnkm";  int lx = sizeof(x) - 1, ly = sizeof(y) - 1;  std::cout << "\n-- расстояние Левенштейна (рекурсия) --\n";  for (int i = 8; i < std::min(lx, ly); i++) {  t1 = clock();  int distance = levenshtein\_recursive(i, x, i - 2, y);  t2 = clock();  std::cout << "Длина " << i - 2 << "/" << i << " Время: " << (t2 - t1) << " мс" << std::endl;  }  system("pause");  return 0;  } |

Листинг 2.4 – Файл secondTask.cpp

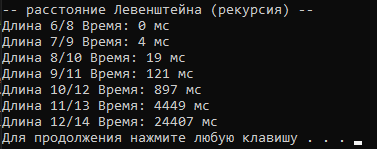


Рисунок 3 – Результат работы программы

***Задание 3.***

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

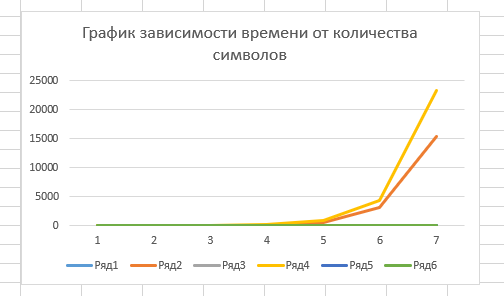


Рисунок 4 – График зависимости времени от количества символов

На графике, который представлен выше, можно сдедать вывод, что алгоритм, вычисленный способом динамического программирования, является более оптимальным и вычисления проходят в разы быстрее, чем рекурсивным алгоритмом.

***Задание 4.***

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



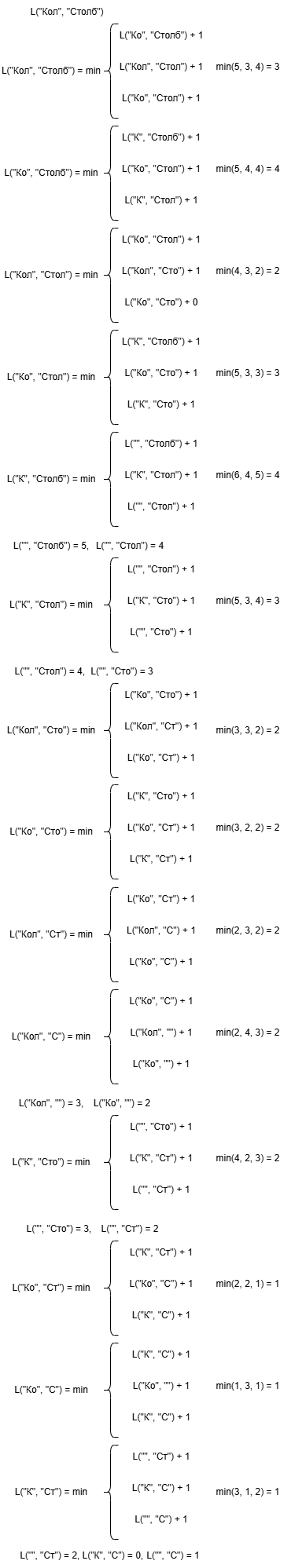


Рисунок 5 – Ручной пример вычисления дистанции Левенштейна

***Задание 5.***

**Нечетные варианты**. Выполнить сравнительный анализ времени затраченного на решение задачи о наибольшей общей подпоследовательности для двух методов решения (рекурсивное решение, динамическое программирование). Две последовательности взять в соответствии с вариантом. Построить графики зависимости времени вычисления от . **Отобразить ход решения в отчете**(по примеру из лекции) + код и копии экрана.



|  |
| --- |
| #pragma once  int lcs(int lenx, // длина последовательности X  const char x[], // последовательность X  int leny, // длина последовательности Y  const char y[] // последовательность Y  );  // -- динамическое вычисление LCS  int lcsd(const char x[], // последовательность X  const char y[], // последовательность Y  char z[] // наибольшая общая подпоследовательность  ); |

Листинг 5.1 – Файл LCS.h

|  |
| --- |
| #include <algorithm>  #include "LCS.h"  int lcs(int lenx, const char x[],  int leny, const char y[])  {  int rc = 0;  if (lenx > 0 && leny > 0)  {  if (x[lenx - 1] == y[leny - 1]) rc = 1 + lcs(lenx - 1, x, leny - 1, y);  else rc = std::max(lcs(lenx, x, leny - 1, y), lcs(lenx - 1, x, leny, y));  }  return rc; //длина LCS  }  #define LCS\_C(x1,x2) (C[(x1)\*(leny+1)+(x2)])  #define LCS\_B(x1,x2) (B[(x1)\*(leny+1)+(x2)])  #define LCS\_X(i) (x[(i)-1])  #define LCS\_Y(i) (y[(i)-1])  #define LCS\_Z(i) (z[(i)-1])  enum Dart { TOP, LEFT, LEFTTOP };  void getLCScontent(int lenx, int leny, const char x[],  const Dart\* B,  int n, int i, int j, char z[])  {  if ((i > 0 && j > 0 && n > 0))  {  if (LCS\_B(i, j) == LEFTTOP)  {  getLCScontent(lenx, leny, x, B, n - 1, i - 1, j - 1, z);  LCS\_Z(n) = LCS\_X(i);  LCS\_Z(n + 1) = 0;  }  else if (LCS\_B(i, j) == TOP)  getLCScontent(lenx, leny, x, B, n, i - 1, j, z);  else getLCScontent(lenx, leny, x, B, n, i, j - 1, z);  }  };  int lcsd(const char x[], const char y[], char z[])  {  int n;  int lenx = strlen(x), leny = strlen(x),  \* C = new int[(lenx + 1) \* (leny + 1)];  Dart\* B = new Dart[(lenx + 1) \* (leny + 1)];  memset(C, 0, sizeof(int) \* (lenx + 1) \* (leny + 1));  for (int i = 1; i <= lenx; i++)  for (int j = 1; j <= leny; j++)  if (LCS\_X(i) == LCS\_Y(j))  {  LCS\_C(i, j) = LCS\_C(i - 1, j - 1) + 1;  LCS\_B(i, j) = LEFTTOP;  }  else if (LCS\_C(i - 1, j) >= LCS\_C(i, j - 1))  {  LCS\_C(i, j) = LCS\_C(i - 1, j);  LCS\_B(i, j) = TOP;  }  else  {  LCS\_C(i, j) = LCS\_C(i, j - 1);  LCS\_B(i, j) = LEFT;  }  getLCScontent(lenx, leny, x, B, LCS\_C(lenx, leny), lenx, leny, z);  return LCS\_C(lenx, leny);  }  #undef LCS\_Z  #undef LCS\_C  #undef LCS\_B  #undef LCS\_X  #undef LCS\_Y |

Листинг 5.2 – Файл LCS.cpp

|  |
| --- |
| #include <iostream>  #include "LCS.h"  #include <Windows.h>  #include <ctime>  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  clock\_t t1 = 0, t2 = 0, t3 = 0, t4 = 0;  char z[100] = "";  t1 = clock();  char X[] = "QVTWNHO";  char Y[] = "RQTWYK";  std::cout << std::endl << "-- вычисление длины LCS для X и Y(рекурсия)";  std::cout << std::endl << "-- последовательность X: " << X;  std::cout << std::endl << "-- последовательность Y: " << Y;  int s = lcs(sizeof(X) - 1, "QVTWNHO", sizeof(Y) - 1, "RQTWYK");  std::cout << std::endl << "-- длина LCS: " << s << std::endl;  t2 = clock();  // наибольшая общая подпоследовательность  t3 = clock();  char x[] = "TOUEXAZ";  char y[] = "HIEHXZ";  int l = lcsd(x, y, z);  t4 = clock();  std::cout << std::endl  << "-- наибольшая общая подпоследовательость - LCS(динамическое"  << " программирование)" << std::endl;  std::cout << std::endl << "последовательость X: " << x;  std::cout << std::endl << "последовательость Y: " << y;  std::cout << std::endl << " LCS: " << z;  std::cout << std::endl << " длина LCS: " << l;  std::cout << std::endl;  std::cout << std::endl << "Время вычисления LCS";  std::cout << std::endl << "Рекурсия: " << (t2 - t1);  std::cout << std::endl << "Динамическое программирование: " << (t4 - t3) << std::endl;  return 0;  } |

Листинг 5.3 – Файл fifthTask.cpp

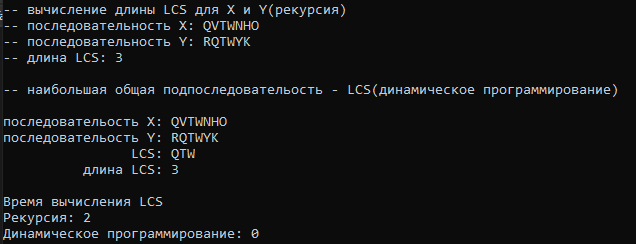


Рисунок 6 – Результат работы программы

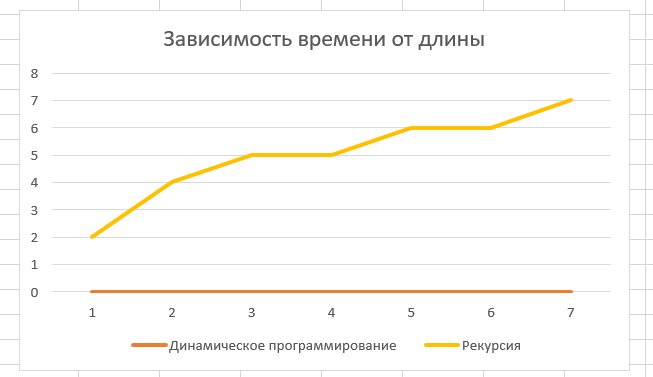


Рисунок 7 – График зависимости времени от длины

На рисунке 7 представлен график зависимости времени от длины последовательности. Сам рисунок представлен двумя графиками, где первый график это поиск общей подпоследовательности рекурсивным алгоритмом за определенное время, а второй график – это результат работы алгоритма, который работал по принципу динамического программирования. При построении графика, мы руководствовались принципом, когда посимвольно сравнивали строки, то есть каждый раз мы добавляли по символу и смотрели за какое время выполняется алгоритм. Данную операцию мы проделывали, пока не дошли до строки в исходном виде. Исходя из данного графика, можно сделать вывод, что поиск общей подпоследовательности лучше реализовывать с помощью динамического программирования, данный подход является более эффективным.

**Матричный способ:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | Q | V | T | W | N | H | O |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| R | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Q | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| T | 0 | 1 | 1 | 2 | 2 | 2 | 2 | 2 |
| W | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |
| Y | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |
| K | 0 | 1 | 1 | 2 | 3 | 3 | 3 | 3 |

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

**Лабораторная работа №5. Транспортная задача**

**Вариант 7**

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

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Потребители Поставщики | B1 | B2 | B3 | B4 | B5 | B6 | Запасы |
| A1 | 19 | 9 | 13 | 10 | 18 | 8 | 175 |
| A2 | 17 | 7 | 15 | 12 | 14 | 20 | 120 |
| A3 | 8 | 12 | 18 | 15 | 9 | 18 | 157 |
| A4 | 11 | 17 | 17 | 10 | 20 | 9 | 166 |
| A5 | 10 | 18 | 16 | 7 | 17 | 11 | 107 |
| Потребности | 150 | 114 | 138 | 200 | 102 | 170 |  |

**Ход решения:**

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

Проверим необходимое и достаточное условие разрешимости задачи.

∑a = 175 + 120 + 157 + 166 + 107 = 725

∑b = 150 + 114 + 138 + 200 + 102 + 170 = 874

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 149 (725—874). Тарифы перевозки единицы груза из базы во все магазины полагаем равны нулю.

Занесем исходные данные в распределительную таблицу.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | B1 | B2 | B3 | B4 | B5 | B6 | Запасы |
| A1 | 19 | 9 | 13 | 10 | 18 | 8 | 175 |
| A2 | 17 | 7 | 15 | 12 | 14 | 20 | 120 |
| A3 | 8 | 12 | 18 | 15 | 9 | 18 | 157 |
| A4 | 11 | 17 | 17 | 10 | 20 | 9 | 166 |
| A5 | 10 | 18 | 16 | 7 | 17 | 11 | 107 |
| A6 | 0 | 0 | 0 | 0 | 0 | 0 | 149 |
| Потребности | 150 | 114 | 138 | 200 | 102 | 170 |  |

**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=7. Для этого элемента запасы равны 120, потребности 114. Поскольку минимальным является 114, то вычитаем его.

x22 = min(120,114) = 114.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 19 | x | 13 | 10 | 18 | 8 | 175 |
| 17 | **7** | 15 | 12 | 14 | 20 | **120 - 114 = 6** |
| 8 | x | 18 | 15 | 9 | 18 | 157 |
| 11 | x | 17 | 10 | 20 | 9 | 166 |
| 10 | x | 16 | 7 | 17 | 11 | 107 |
| 0 | x | 0 | 0 | 0 | 0 | 149 |
| 150 | **114 - 114 = 0** | 138 | 200 | 102 | 170 |  |

Искомый элемент равен c54=7. Для этого элемента запасы равны 107, потребности 200. Поскольку минимальным является 107, то вычитаем его.

x54 = min(107,200) = 107.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 19 | x | 13 | 10 | 18 | 8 | 175 |
| 17 | 7 | 15 | 12 | 14 | 20 | 6 |
| 8 | x | 18 | 15 | 9 | 18 | 157 |
| 11 | x | 17 | 10 | 20 | 9 | 166 |
| x | x | x | **7** | x | x | **107 - 107 = 0** |
| 0 | x | 0 | 0 | 0 | 0 | 149 |
| 150 | 0 | 138 | **200 - 107 = 93** | 102 | 170 |  |

Искомый элемент равен c16=8. Для этого элемента запасы равны 175, потребности 170. Поскольку минимальным является 170, то вычитаем его.

x16 = min(175,170) = 170.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 19 | x | 13 | 10 | 18 | **8** | **175 - 170 = 5** |
| 17 | 7 | 15 | 12 | 14 | x | 6 |
| 8 | x | 18 | 15 | 9 | x | 157 |
| 11 | x | 17 | 10 | 20 | x | 166 |
| x | x | x | 7 | x | x | 0 |
| 0 | x | 0 | 0 | 0 | x | 149 |
| 150 | 0 | 138 | 93 | 102 | **170 - 170 = 0** |  |

Искомый элемент равен c31=8. Для этого элемента запасы равны 157, потребности 150. Поскольку минимальным является 150, то вычитаем его.

x31 = min(157,150) = 150.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | 13 | 10 | 18 | 8 | 5 |
| x | 7 | 15 | 12 | 14 | x | 6 |
| **8** | x | 18 | 15 | 9 | x | **157 - 150 = 7** |
| x | x | 17 | 10 | 20 | x | 166 |
| x | x | x | 7 | x | x | 0 |
| x | x | 0 | 0 | 0 | x | 149 |
| **150 - 150 = 0** | 0 | 138 | 93 | 102 | 0 |  |

Искомый элемент равен c35=9. Для этого элемента запасы равны 7, потребности 102. Поскольку минимальным является 7, то вычитаем его.

x35 = min(7,102) = 7.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | 13 | 10 | 18 | 8 | 5 |
| x | 7 | 15 | 12 | 14 | x | 6 |
| 8 | x | x | x | **9** | x | **7 - 7 = 0** |
| x | x | 17 | 10 | 20 | x | 166 |
| x | x | x | 7 | x | x | 0 |
| x | x | 0 | 0 | 0 | x | 149 |
| 0 | 0 | 138 | 93 | **102 - 7 = 95** | 0 |  |

Искомый элемент равен c14=10. Для этого элемента запасы равны 5, потребности 93. Поскольку минимальным является 5, то вычитаем его.

x14 = min(5,93) = 5.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | **10** | x | 8 | **5 - 5 = 0** |
| x | 7 | 15 | 12 | 14 | x | 6 |
| 8 | x | x | x | 9 | x | 0 |
| x | x | 17 | 10 | 20 | x | 166 |
| x | x | x | 7 | x | x | 0 |
| x | x | 0 | 0 | 0 | x | 149 |
| 0 | 0 | 138 | **93 - 5 = 88** | 95 | 0 |  |

Искомый элемент равен c44=10. Для этого элемента запасы равны 166, потребности 88. Поскольку минимальным является 88, то вычитаем его.

x44 = min(166,88) = 88.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | 10 | x | 8 | 0 |
| x | 7 | 15 | x | 14 | x | 6 |
| 8 | x | x | x | 9 | x | 0 |
| x | x | 17 | **10** | 20 | x | **166 - 88 = 78** |
| x | x | x | 7 | x | x | 0 |
| x | x | 0 | x | 0 | x | 149 |
| 0 | 0 | 138 | **88 - 88 = 0** | 95 | 0 |  |

Искомый элемент равен c25=14. Для этого элемента запасы равны 6, потребности 95. Поскольку минимальным является 6, то вычитаем его.

x25 = min(6,95) = 6.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | 10 | x | 8 | 0 |
| x | 7 | x | x | **14** | x | **6 - 6 = 0** |
| 8 | x | x | x | 9 | x | 0 |
| x | x | 17 | 10 | 20 | x | 78 |
| x | x | x | 7 | x | x | 0 |
| x | x | 0 | x | 0 | x | 149 |
| 0 | 0 | 138 | 0 | **95 - 6 = 89** | 0 |  |

Искомый элемент равен c43=17. Для этого элемента запасы равны 78, потребности 138. Поскольку минимальным является 78, то вычитаем его.

x43 = min(78,138) = 78.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | 10 | x | 8 | 0 |
| x | 7 | x | x | 14 | x | 0 |
| 8 | x | x | x | 9 | x | 0 |
| x | x | **17** | 10 | x | x | **78 - 78 = 0** |
| x | x | x | 7 | x | x | 0 |
| x | x | 0 | x | 0 | x | 149 |
| 0 | 0 | **138 - 78 = 60** | 0 | 89 | 0 |  |

Искомый элемент равен c63=0. Для этого элемента запасы равны 149, потребности 60. Поскольку минимальным является 60, то вычитаем его.

x63 = min(149,60) = 60.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | 10 | x | 8 | 0 |
| x | 7 | x | x | 14 | x | 0 |
| 8 | x | x | x | 9 | x | 0 |
| x | x | 17 | 10 | x | x | 0 |
| x | x | x | 7 | x | x | 0 |
| x | x | **0** | x | 0 | x | **149 - 60 = 89** |
| 0 | 0 | **60 - 60 = 0** | 0 | 89 | 0 |  |

Искомый элемент равен c65=0. Для этого элемента запасы равны 89, потребности 89. Поскольку минимальным является 89, то вычитаем его.

x65 = min(89,89) = 89.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | 10 | x | 8 | 0 |
| x | 7 | x | x | 14 | x | 0 |
| 8 | x | x | x | 9 | x | 0 |
| x | x | 17 | 10 | x | x | 0 |
| x | x | x | 7 | x | x | 0 |
| x | x | 0 | x | **0** | x | **89 - 89 = 0** |
| 0 | 0 | 0 | 0 | **89 - 89 = 0** | 0 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Потребители Поставщики | B1 | B2 | B3 | B4 | B5 | B6 | Запасы |
| A1 | 19 | 9 | 13 | 10[5] | 18 | 8[170] | 175 |
| A2 | 17 | 7[114] | 15 | 12 | 14[6] | 20 | 120 |
| A3 | 8[150] | 12 | 18 | 15 | 9[7] | 18 | 157 |
| A4 | 11 | 17 | 17[78] | 10[88] | 20 | 9 | 166 |
| A5 | 10 | 18 | 16 | 7[107] | 17 | 11 | 107 |
| A6 | 0 | 0 | 0[60] | 0 | 0[89] | 0 | 149 |
| Потребности | 150 | 114 | 138 | 200 | 102 | 170 |  |

В результате получен первый опорный план, который является допустимым, так как все грузы из баз вывезены, потребность магазинов 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) = 10∙5 + 8∙170 + 7∙114 + 14∙6 + 8∙150 + 9∙7 + 17∙78 + 10∙88 + 7∙107 + 0∙60 + 0∙89 = 6510

**Этап II. Улучшение опорного плана**.

Проверим оптимальность опорного плана. Найдем *предварительные потенциалы* ui, vj. по занятым клеткам таблицы, в которых ui + vj = cij, полагая, что u1 = 0.

u1 + v4 = 10; 0 + v4 = 10; v4 = 10

u4 + v4 = 10; 10 + u4 = 10; u4 = 0

u4 + v3 = 17; 0 + v3 = 17; v3 = 17

u6 + v3 = 0; 17 + u6 = 0; u6 = -17

u6 + v5 = 0; -17 + v5 = 0; v5 = 17

u2 + v5 = 14; 17 + u2 = 14; u2 = -3

u2 + v2 = 7; -3 + v2 = 7; v2 = 10

u3 + v5 = 9; 17 + u3 = 9; u3 = -8

u3 + v1 = 8; -8 + v1 = 8; v1 = 16

u5 + v4 = 7; 10 + u5 = 7; u5 = -3

u1 + v6 = 8; 0 + v6 = 8; v6 = 8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | v1=16 | v2=10 | v3=17 | v4=10 | v5=17 | v6=8 |
| u1=0 | 19 | 9 | 13 | 10[5] | 18 | 8[170] |
| u2=-3 | 17 | 7[114] | 15 | 12 | 14[6] | 20 |
| u3=-8 | 8[150] | 12 | 18 | 15 | 9[7] | 18 |
| u4=0 | 11 | 17 | 17[78] | 10[88] | 20 | 9 |
| u5=-3 | 10 | 18 | 16 | 7[107] | 17 | 11 |
| u6=-17 | 0 | 0 | 0[60] | 0 | 0[89] | 0 |

Опорный план не является оптимальным, так как существуют оценки свободных клеток, для которых ui + vj > cij

(1;2): 0 + 10 > 9; ∆12 = 0 + 10 - 9 = 1 > 0

(1;3): 0 + 17 > 13; ∆13 = 0 + 17 - 13 = 4 > 0

(4;1): 0 + 16 > 11; ∆41 = 0 + 16 - 11 = 5 > 0

(5;1): -3 + 16 > 10; ∆51 = -3 + 16 - 10 = 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 | 19 | 9 | 13 | 10[5] | 18 | 8[170] | 175 |
| 2 | 17 | 7[114] | 15 | 12 | 14[6] | 20 | 120 |
| 3 | 8[150][-] | 12 | 18 | 15 | 9[7][+] | 18 | 157 |
| 4 | 11[+] | 17 | 17[78][-] | 10[88] | 20 | 9 | 166 |
| 5 | 10 | 18 | 16 | 7[107] | 17 | 11 | 107 |
| 6 | 0 | 0 | 0[60][+] | 0 | 0[89][-] | 0 | 149 |
| Потребности | 150 | 114 | 138 | 200 | 102 | 170 |  |

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) = 78. Прибавляем 78 к объемам грузов, стоящих в плюсовых клетках и вычитаем 78 из Хij, стоящих в минусовых клетках. В результате получим новый опорный план.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Потребители Поставщики | B1 | B2 | B3 | B4 | B5 | B6 | Запасы |
| A1 | 19 | 9 | 13 | 10[5] | 18 | 8[170] | 175 |
| A2 | 17 | 7[114] | 15 | 12 | 14[6] | 20 | 120 |
| A3 | 8[72] | 12 | 18 | 15 | 9[85] | 18 | 157 |
| A4 | 11[78] | 17 | 17 | 10[88] | 20 | 9 | 166 |
| A5 | 10 | 18 | 16 | 7[107] | 17 | 11 | 107 |
| A6 | 0 | 0 | 0[138] | 0 | 0[11] | 0 | 149 |
| Потребности | 150 | 114 | 138 | 200 | 102 | 170 |  |

Проверим оптимальность опорного плана. Найдем *предварительные потенциалы* ui, vj. по занятым клеткам таблицы, в которых ui + vj = cij, полагая, что u1 = 0.

u1 + v4 = 10; 0 + v4 = 10; v4 = 10

u4 + v4 = 10; 10 + u4 = 10; u4 = 0

u4 + v1 = 11; 0 + v1 = 11; v1 = 11

u3 + v1 = 8; 11 + u3 = 8; u3 = -3

u3 + v5 = 9; -3 + v5 = 9; v5 = 12

u2 + v5 = 14; 12 + u2 = 14; u2 = 2

u2 + v2 = 7; 2 + v2 = 7; v2 = 5

u6 + v5 = 0; 12 + u6 = 0; u6 = -12

u6 + v3 = 0; -12 + v3 = 0; v3 = 12

u5 + v4 = 7; 10 + u5 = 7; u5 = -3

u1 + v6 = 8; 0 + v6 = 8; v6 = 8

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | v1=11 | v2=5 | v3=12 | v4=10 | v5=12 | v6=8 |
| u1=0 | 19 | 9 | 13 | 10[5] | 18 | 8[170] |
| u2=2 | 17 | 7[114] | 15 | 12 | 14[6] | 20 |
| u3=-3 | 8[72] | 12 | 18 | 15 | 9[85] | 18 |
| u4=0 | 11[78] | 17 | 17 | 10[88] | 20 | 9 |
| u5=-3 | 10 | 18 | 16 | 7[107] | 17 | 11 |
| u6=-12 | 0 | 0 | 0[138] | 0 | 0[11] | 0 |

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

Минимальные затраты составят: F(x) = 10∙5 + 8∙170 + 7∙114 + 14∙6 + 8∙72 + 9∙85 + 11∙78 + 10∙88 + 7∙107 + 0∙138 + 0∙11 = 6120

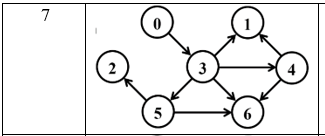
**Вывод:** Оба метода имеют свои области применения. Метод наименьшей стоимости удобен для быстрого получения начального плана, а метод потенциалов обеспечивает точное нахождение оптимального решения. В практических задачах часто используется комбинация этих методов: сначала строится начальный план методом наименьшей стоимости, а затем он оптимизируется с помощью метода потенциалов. Такой подход позволяет эффективно решать транспортные задачи с минимальными затратами времени и ресурсов.

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

**Вариант 7**

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

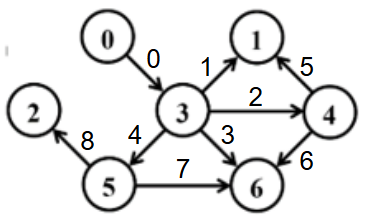
***Задание 1.*** Ориентированный граф **G** взять в соответствии с вариантом. Представить его в отчете в виде матрицы смежности, матрицы инцидентности, списка смежных вершин.



**Матрица смежности:**

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

Для того, чтобы построить матрицу инцидентности, нам нужно определить количество ребер в нашем ориентированном графе. В нашем графе их количество равно 9. Теперь отметим эти ребра на нашем рисунке:



**Матрица инцидентности:**

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

**Список смежных вершин:**  
S0 = {3}

S1 = ∅

S2 = ∅

S3 = {1, 4, 5, 6}

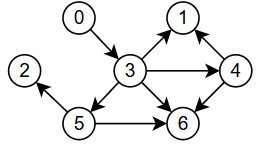
S4 = {1, 6}

S5 = {2, 6}

S6 = ∅

***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.

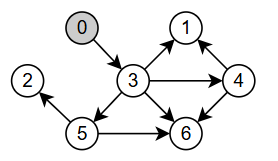
**Поиск в ширину**



Шаг 1

Очередь: 0

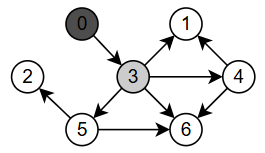
Последовательность: ∅



Шаг 2

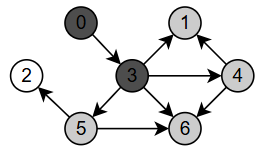
Очередь: 3

Последовательность: 0



Шаг 3

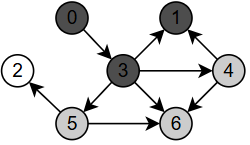
Очередь: 1, 4, 5, 6  
Последовательность: 0, 3



Шаг 4

Очередь: 4, 5, 6

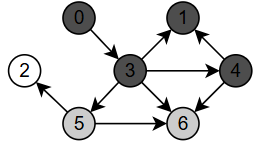
Последовательность: 0, 3, 1



Шаг 5

Очередь: 5, 6

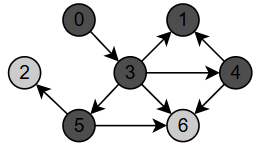
Последовательность: 0, 3, 1, 4



Шаг 6

Очередь: 6, 2

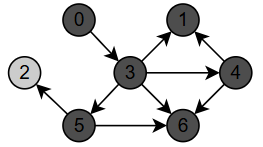
Последовательность: 0, 3, 1, 4, 5



Шаг 7

Очередь: 2

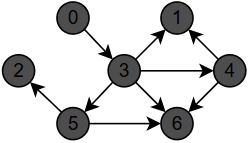
Последовательность: 0, 3, 1, 4, 5, 6



Шаг 8

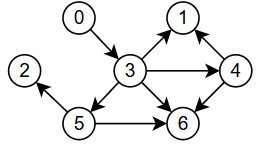
Очередь: ∅

Последовательность: 0, 3, 1, 4, 5, 6, 2



Обход в ширину: 0, 3, 1, 4, 5, 6, 2

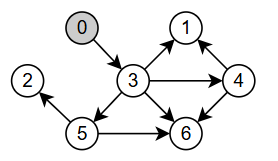
**Поиск в глубину**



Шаг 1

Стек: 0

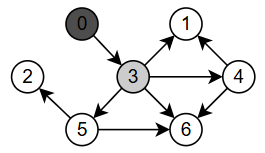
Последовательность: ∅



Шаг 2

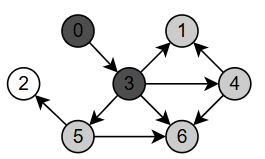
Стек: 3

Последовательность: 0



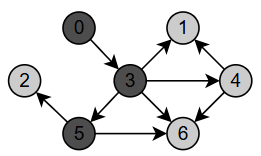
Шаг 3

Стек: 5,4,6,1  
Последовательность: 0, 3



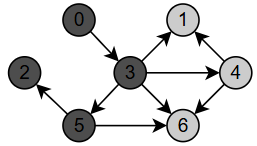
Шаг 4

Стек: 2,4,6,1  
Последовательность: 0, 3, 5



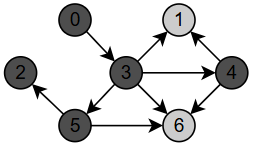
Шаг 5

Стек: 4,6,1  
Последовательность: 0, 3, 5, 2



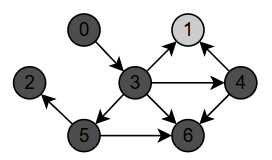
Шаг 6

Стек: 6,1  
Последовательность: 0, 3, 5, 2, 4



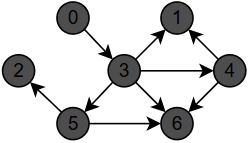
Шаг 7

Стек: 1  
Последовательность: 0, 3, 5, 2, 4, 6



Шаг 8

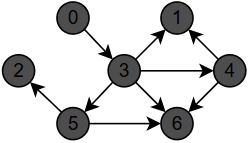
Стек: ∅  
Последовательность: 0, 3, 5, 2, 4, 6, 1



Обход в глубину: 0, 3, 5, 2, 4, 6, 1

**Топологическая сортировка**

Топологическая сортировка — это упорядочивание вершин ориентированного графа, при котором для каждого ребра (u→v) вершина u идет перед вершиной vв этом упорядочивании. Она возможна только для ациклических ориентированных графов (DAG). В нашем случае результат топологической сортировки будет совпадать с результатом DFS, так как топологическая сортировка использует тот же механизм, что и DFS: вершины добавляются в стек в порядке их завершения. Так как в графе нет циклов, и DFS обходит все вершины в порядке, который уже соответствует топологическому упорядочиванию, то результат топологической сортировки совпадает с порядком завершения вершин в DFS.



Результат топологической сортировки: 0, 3, 5, 2, 4, 6, 1

***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.

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

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

Данные 3 задания будут размещены в одном программном файле и результат данной программы будет прикреплен ниже снимком экрана:

|  |
| --- |
| #include <iostream>  #include <stack>  #include <queue>  #include <vector>  #include <algorithm>  using namespace std;  const int V = 7;  int matrix[V][V] =  {  {0, 0, 0, 1, 0, 0, 0},  {0, 0, 0, 0, 0, 0, 0},  {0, 0, 0, 0, 0, 0, 0},  {0, 1, 0, 0, 1, 1, 1},  {0, 1, 0, 0, 0, 0, 1},  {0, 0, 1, 0, 0, 0, 1},  {0, 0, 0, 0, 0, 0, 0}  };  void BFS\_by\_matrix(int start) {  queue<int> q;  bool visited[V] = { false };  q.push(start);  visited[start] = true;  while (!q.empty()) {  int current = q.front();  q.pop();  cout << current << " ";  for (int i = 0; i < V; i++) {  if (matrix[current][i] == 1 && !visited[i]) {  q.push(i);  visited[i] = true;  }  }  }  }  void DFS\_by\_matrix(int start, vector<bool>& visited, stack<int>& stack) {  visited[start] = true;  for (int i = 0; i < V; i++) {  if (matrix[start][i] == 1 && !visited[i]) {  DFS\_by\_matrix(i, visited, stack);  }  }  stack.push(start);  }  void topologicalSort() {  stack<int> stack;  vector<bool> visited(V, false);  for (int i = 0; i < V; i++) {  if (!visited[i]) {  DFS\_by\_matrix(i, visited, stack);  }  }  cout << "Топологическая сортировка: ";  while (!stack.empty()) {  cout << stack.top() << " ";  stack.pop();  }  cout << endl;  }  void printAdjacencyList() {  cout << "Список смежных вершин:" << endl;  for (int i = 0; i < V; i++) {  cout << i << ": ";  for (int j = 0; j < V; j++) {  if (matrix[i][j] == 1) {  cout << j << " ";  }  }  cout << endl;  }  }  void printEdgeList() {  cout << "Список ребер:" << endl;  for (int i = 0; i < V; i++) {  for (int j = 0; j < V; j++) {  if (matrix[i][j] == 1) {  cout << "(" << i << " -> " << j << ") ";  }  }  }  cout << endl;  }  int main() {  setlocale(LC\_ALL, "RU");  int start = 0;  cout << "Обход в ширину (BFS): ";  BFS\_by\_matrix(start);  cout << endl;  cout << "Обход в глубину (DFS): ";  vector<bool> visited(V, false);  stack<int> stack;  DFS\_by\_matrix(start, visited, stack);  while (!stack.empty()) {  cout << stack.top() << " ";  stack.pop();  }  cout << endl;  topologicalSort();  printAdjacencyList();  printEdgeList();  return 0;  } |

Листинг 1 – Main.cpp

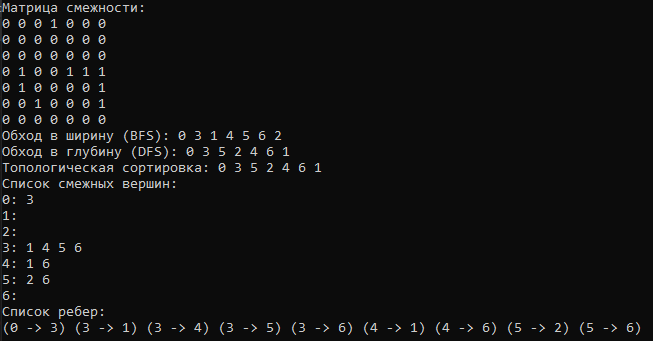


Рисунок 1 – Результат программы

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

Веса ребер принять:

W:

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

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

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

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

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

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

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

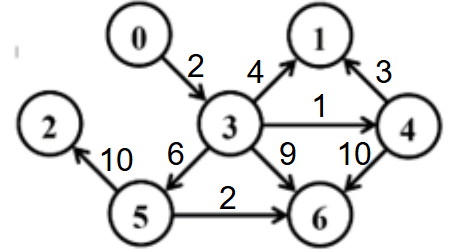
W(e4,3)=4; W(e3,4)=1;

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

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

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

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

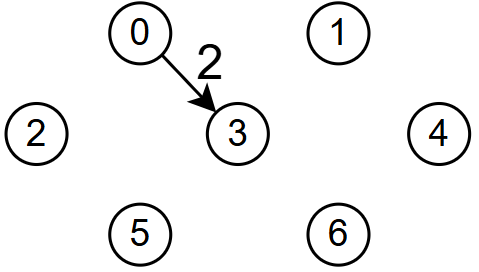


Шаг 1:

Предположим, что в множество U пока только входит вершина 0, нужно найти ребро с минимальной стоимостью, которое соединяет вершину 0 с вершиной, которая в множество U не входит.

U = {0}

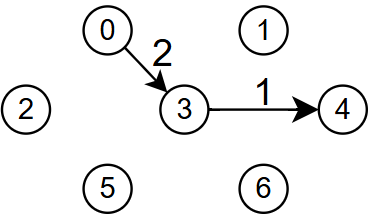
Вершина 0 соединена только с вершиной 3. Вес данной вершины равен 2, включаем данную вершину в множество U.



Шаг 2:

U = {0, 3}

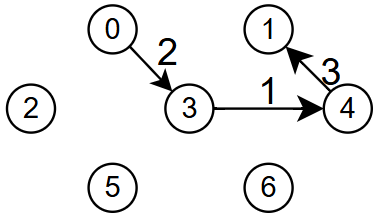
Вершина 3 соединена с вершинами 1, 4, 5, 6. Веса этих рёбер равны 4, 1, 9, 6. Из всех этих рёбер минимальной стоимостью обладает ребро 3-4 (вес = 1). Включаем вершину 4 в множество U.



Шаг 3:

U = {0, 3, 4}

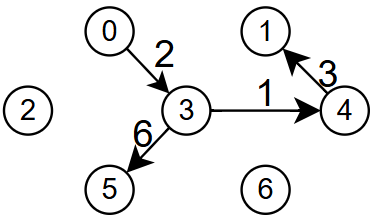
Вершины 3 соединена с вершинами 1, 5, 6, а вершина 4 с вершинами 1 и 6. Среди данных рёбер наименьшую стоимость имеет ребро 4-1 (вес = 3). Вершину 1 включаем в множество U.



Шаг 4:

U = {0, 3, 4, 1}

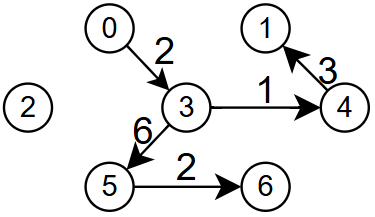
Вершина 3 соединена с вершинами 1, 5, 6, но так как мы вершину 1 уже посетили, ребро 3-1 мы не рассматриваем. Вершина 4 соединена с вершиной 6. Среди данных рёбер наименьшую стоимость имеет ребро 3-5 (вес = 6). Включаем вершину 5 в множество U.



Шаг 5:

U = {0, 3, 4, 1, 5}

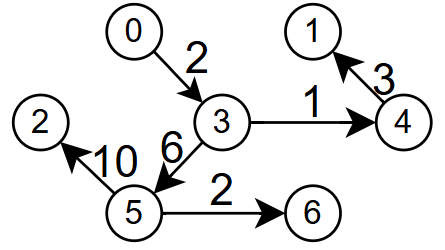
Вершина 3 соединена с вершиной 6, а вершина 5 с вершиной 2 и 6. Среди данных рёбер наименьшую стоимость имеет ребро 5-6 (вес = 2). Добавляем вершину 6 в множество U.



Шаг 6:

U = {0, 3, 4, 1, 5, 6}

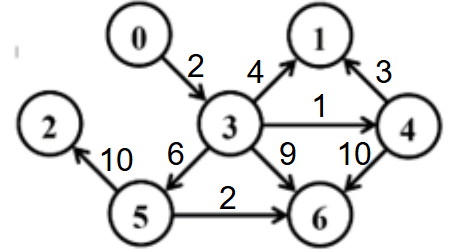
Оставшаяся вершина 2 соединена с вершиной 5, вес данного ребра равен 10. Так что просто добавляем вершину 2 в множество U.



U = {0, 3, 4, 1, 5, 6, 2}

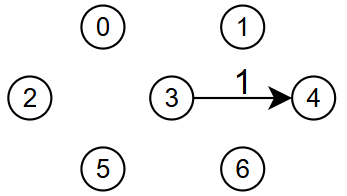
Вес минимального остовного дерева: 24

***Задание 7.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Крускала. Шаги построения отразить в отчете.

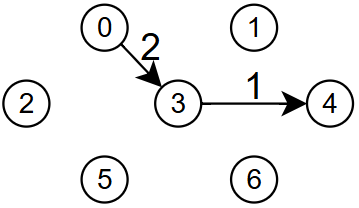


Шаг 1:

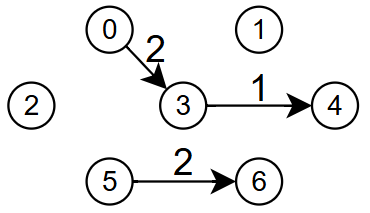
Из всех ребер выбираем ребро с минимальной стоимостью (весом) и включаем это ребро в остовное дерево. Повторяем так далее, однако смотрим на то, чтобы следующее ребро соединяло вершины из разных компонент.



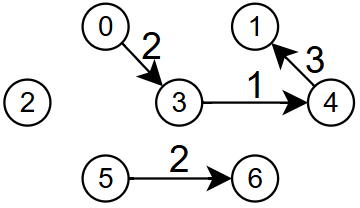
Шаг 2:



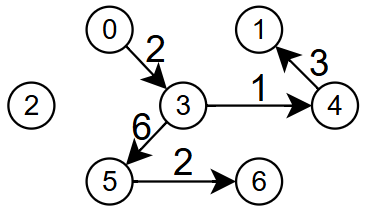
Шаг 3:



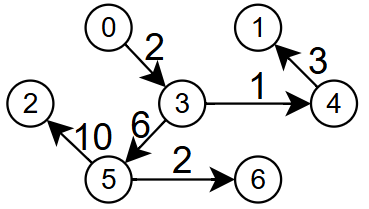
Шаг 4:



Шаг 5:



Шаг 6:



Вес минимального остовного дерева: 24

**Лабораторная работа №7. Сетевые модели**

**Вариант 7**

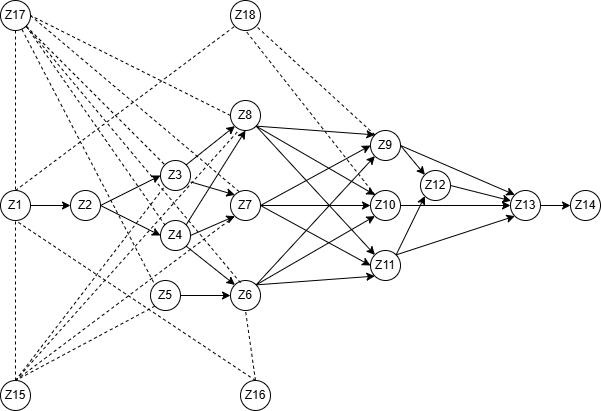
**Цель работы:** Приобретение навыков сетевого планирования и составления сетевых графиков, приобретение опыта нахождения критического пути.

**Задание 1-2. Структурное планирование. Календарное планирование.**

Тема: «Создание десктопного приложения».

|  |  |  |  |
| --- | --- | --- | --- |
| Код операции | Наименование операции | Предшествующие операции | t |
| I. АНАЛИЗ | | | |
| Z1 | Системный анализ |  | 2 |
| Z2 | Анализ требований | Z1 | 3 |
| II. ПРОЕКТИРОВАНИЕ | | | |
| Z3 | Проектирование базы данных | Z2, Z15, Z17 | 3 |
| Z4 | Проектирование классов | Z2, Z17 | 5 |
| Z5 | Проектирование интерфейсов пользователей | Z15, Z17 | 3 |
| III. КОДИРОВАНИЕ | | | |
| Z6 | Кодирование интерфейсов пользователей | Z4, Z5, Z16, Z17 | 10 |
| Z7 | Кодирование процедур СУБД | Z3, Z4, Z15, Z17 | 3 |
| Z8 | Кодирование классов | Z3, Z4, Z15, Z17 | 6 |
| IV. ТЕСТИРОВАНИЕ | | | |
| Z9 | Функциональное тестирование | Z6, Z7, Z8, Z18 | 7 |
| Z10 | Структурное тестирование | Z6, Z7, Z8, Z18 | 3 |
| V. ВНЕДРЕНИЕ | | | |
| Z11 | Разработка документации | Z6, Z7, Z8, Z9 | 4 |
| Z12 | Обучение пользователей | Z9, Z11 | 3 |
| Z13 | Испытание | Z9,Z10,Z11,Z12 | 5 |
| Z14 | Завершение работ | Z13 | 3 |
| VI. ДОПОЛНИТЕЛЬНЫЕ РАБОТЫ | | | |
| Z15 | Установка СУБД | Z1 | 1 |
| Z16 | Установка web-сервера | Z1 | 1 |
| Z17 | Установка инструментария | Z1 | 1 |
| Z18 | Подготовка полигона | Z1 | 2 |

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



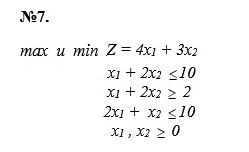
Длина критического пути: 42 дня. Последовательность задач, принадлежащих критическому пути: **Z1 → Z2 → Z4 → Z6 → Z9 → Z11 → Z12 → Z13 → Z14.**

**Вывод:** В ходе выполнения лабораторной работы был составлен структурный и календарный план создания десктопного приложенпя. Кроме того, был составлен сетевой график и найден критический путь.

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

**Вариант 7**

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



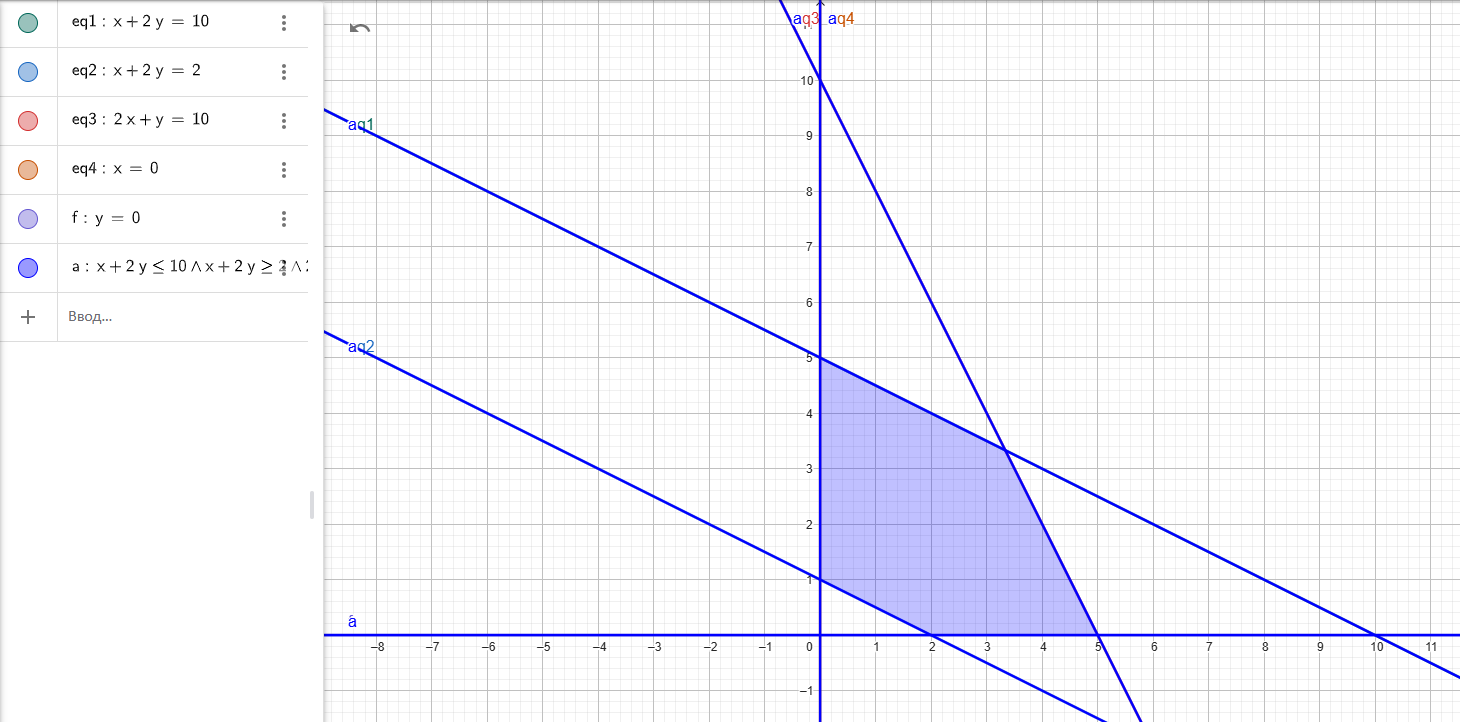


Рисунок 1 – Область ограничений

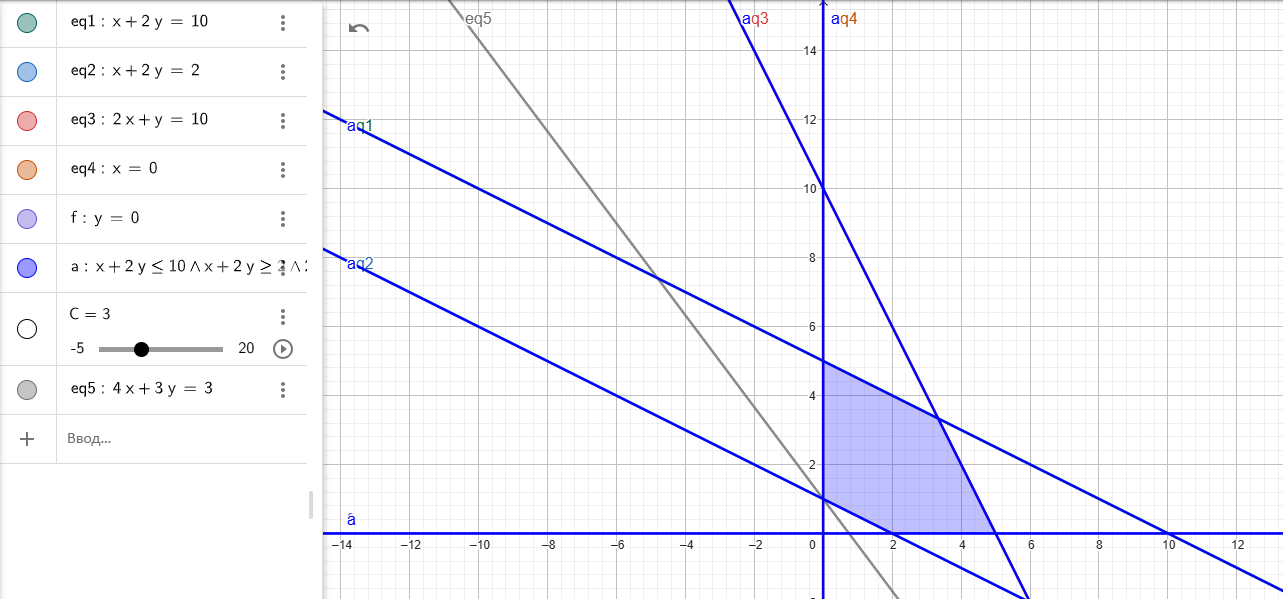


Рисунок 2 – Нахождение минимального значения

Для того что определить минимальное значение в данной области, нужно построить график прямой 4x + 3y и приравнять к C. Затем построенную прямую двигать 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 4x + 3y. Получаем: 4 \* 0 + 3 \* 1 = 3. Минимальное значение данной функции получилось 3. Оно совпадает с тем значением, которое получилось графически.

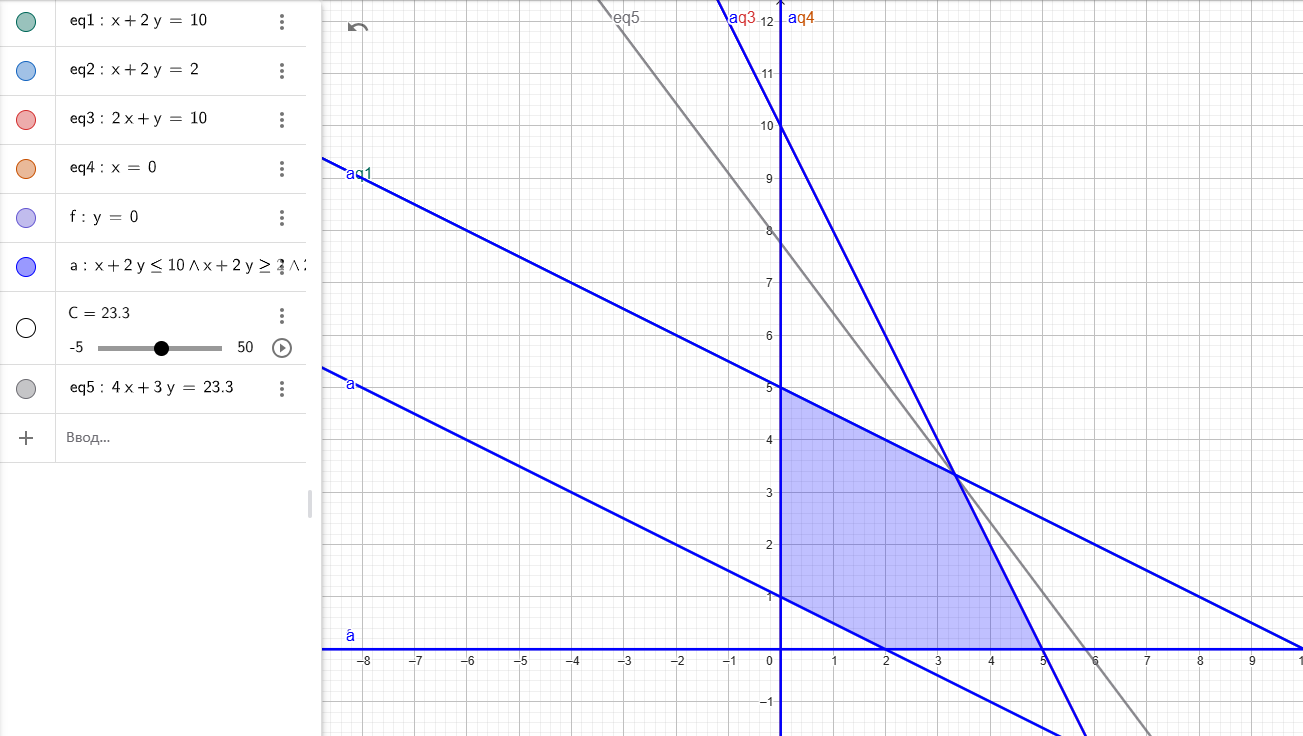


Рисунок 3 – Нахождение максимального значения

Для того что определить максимальное значение в данной области, нужно построить график прямой 4x + 3y 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 (3,33;3,33). Теперь данную точку подставляем в 4x + 3y. Получаем: 4 \* 3,33 + 3 \* 3,33 = 23,3. Максимальное значение данной функции получилось 23,3. Оно совпадает с тем значением, которое получилось графически.

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