Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Математическое программирование  
(вариант 11)

Студент: Трубач Д. С.

ФИТ 2 курс 5 группа

Лаборант: Степанова Л. П.

Минск 2023

**Содержание**

[Лабораторная работа №1. Вспомогательные функции 3](#_Toc132498378)

[Лабораторная работа №2. Комбинаторные алгоритмы решения оптимизационных задач 7](#_Toc132498379)

[Лабораторная работа №3. Метод ветвей и границ. Задача коммивояжера и методы её решения. 25](#_Toc132498380)

[Лабораторная работа №4. Динамическое программирование 33](#_Toc132498381)

[Лабораторная работа №5. Транспортная задача 46](#_Toc132498382)

[Лабораторная работа №6. Алгоритмы на графах 54](#_Toc132498383)

# 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

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

Листинг 1.1 — Файл Auxil.cpp

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

Листинг 1.2 — Файл Auxil.h

***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.

|  |
| --- |
| #include "stdafx.h"  #include "Auxil.h" // вспомогательные функции  #include "Fibonachi.h"  #include <iostream>  #include <ctime>  #include <locale>  #define CYCLE 1000000 // количество циклов  int main(int argc, char\* argv[])  {  double av1 = 0, av2 = 0;  clock\_t t1 = 0, t2 = 0;  setlocale(LC\_ALL, "rus");  auxil::start(); // старт генерации  t1 = clock(); // фиксация времени  for (int i = 0; i < CYCLE; i++)  {  av1 += (double)auxil::iget(-100, 100); // сумма случайных чисел  av2 += auxil::dget(-100, 100); // сумма случайных чисел  }  t2 = clock(); // фиксация времени  std::cout << std::endl << "количество циклов: " << CYCLE;  std::cout << std::endl << "среднее значение (int): " << av1 / CYCLE;  std::cout << std::endl << "среднее значение (double): " << av2 / CYCLE;  std::cout << std::endl << "продолжительность (у.е): " << (t2 - t1);  std::cout << std::endl << " (сек): "  << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC);  std::cout << std::endl;  system("pause");  return 0;  } |

Листинг 1.3 — Пример 2

**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.

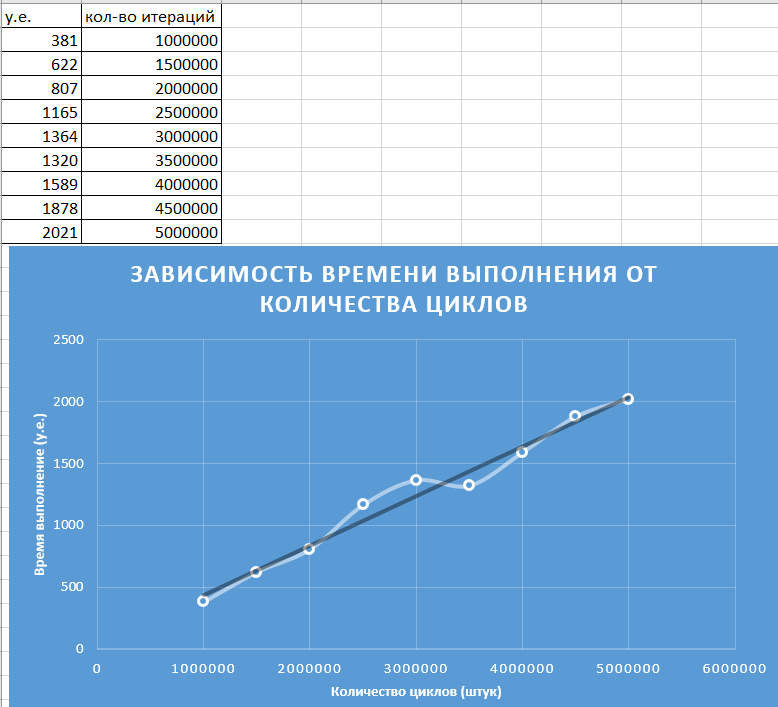


Рисунок 1.1 — График зависимости продолжительности процесса вычисления от количества циклов

Найдем коэффициент корреляции для данной зависимости:



Рисунок 1.2 — Коэффициент корреляции

Так как коэффициент приблизительно равен единице, можем сделать вывод о том, что зависимость продолжительности процесса вычисления от количества цикла имеет линейный вид.

**Числа Фибоначчи:**

Последовательность чисел Фибоначчи определяется формулой Fn = Fn-1 + Fn-2. То есть, следующее число получается как сумма двух предыдущих.

Первые два числа равны 1, затем 2(1+1), затем 3(1+2), 5(2+3) и так далее: 1, 1, 2, 3, 5, 8, 13, 21....

Реализацию данного алгоритма и график зависимости времени выполнения от заданного числа можно посмотреть в листинге 1.4 и на рисунке 1.3., соответственно.

|  |
| --- |
| //-- Fibonachi.cpp  #include "Fibonachi.h"  #include <iostream>  // Посчитать N-ое число Фибоначчи  long double fibonachi(int n)  {  if (n == 0)  return 0;  if (n == 1)  return 1;  return fibonachi(n - 1) + fibonachi(n - 2);  }  //-- Lab1.cpp  std::cout << "=============\n\nФункция чисел Фиббоначи";  clock\_t t3 = 0, t4 = 0; int a;  std::cout << std::endl << "Введите N-ое число: ";  std::cin >> a;  t3 = clock();  long double result = fibonachi(a);  std::cout << std::endl << a << " число Фиббоначи = " << 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; |

Листинг 1.4 — Реализация алгоритма чисел Фибоначчи и тестирование

Рисунок 1.3 — График зависимости продолжительности процесса вычисления от количества циклов

Вывод: скорость выполнения программы линейно зависит от количества итераций цикла. Скорость нахождения N-го числа Фибоначчи имеет вид показательной зависимости.

# Лабораторная работа №2. Комбинаторные алгоритмы решения оптимизационных задач

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

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

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

|  |
| --- |
| // Combi.h  #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 2.1 — файл Combi.h

|  |
| --- |
| // Combi.cpp  #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 2.2 — файл Combi.cpp

|  |
| --- |
| // Main  #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 2.3 — файл Subsets(Task1).cpp

Результат работы генератора подмножеств заданного множества предоставлен на рисунке 2.1.

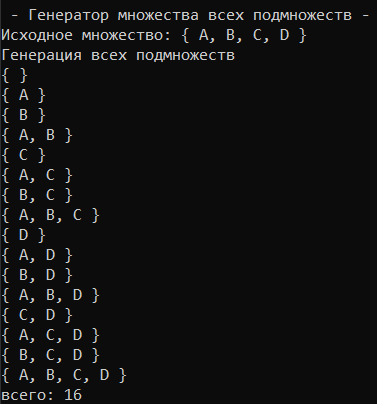


Рисунок 2.1 — результат работы генератора подмножеств заданного множества

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

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

|  |
| --- |
| // Combi.h  #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 2.4 — файл Combi.h

|  |
| --- |
| // Combi.cpp  #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 2.5 — файл Combi.cpp

|  |
| --- |
| // Main  #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 2.6 — файл Combinations(Task2).cpp

Результат работы генератора сочетаний предоставлен на рисунке 2.2.

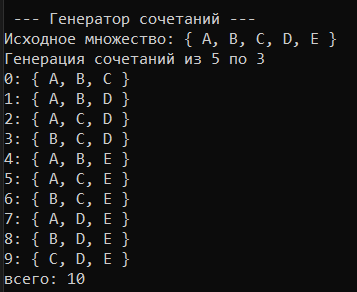


Рисунок 2.2 — результат работы генератора сочетаний

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

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

|  |
| --- |
| // Combi.h  #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 2.7 — файл Combi.h

|  |
| --- |
| // Combi.cpp  #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 2.8 — файл Combi.cpp

|  |
| --- |
| // --- Main  #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 2.9 — файл Permutations(Task3).cpp

Результат работы генератора перестановок предоставлен на рисунке 2.3.

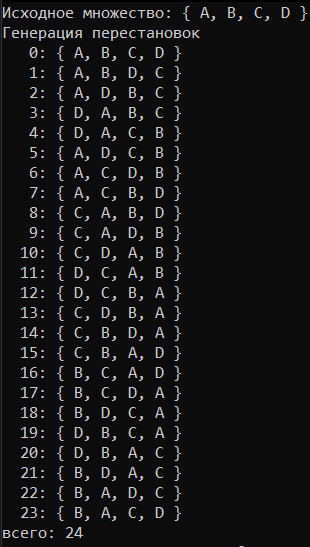


Рисунок 2.3 — результат работы генератора перестановок

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

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

|  |
| --- |
| // Combi.h  #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  };  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  };  struct accomodation // генератор размещений  {  short n, // количество элементов исходного множества  m, // количество элементов в размещении  \* sset; // массив индесов текущего размещения  xcombination\* cgen; // указатель на генератор сочетаний  permutation\* pgen; // указатель на генератор перестановок  accomodation(short n = 1, short m = 1); // конструктор  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 na; // номер размещения 0, ..., count()-1  unsigned \_\_int64 count() const; // общее количество размещений  };  } |

Листинг 2.10 — файл Combi.h

|  |
| --- |
| // Combi.cpp  #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() // ñáðîñèòü ãåíåðàòîð, íà÷àòü ñíà÷àëà  {  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() // ñôîðìèðîâàòü ñëåäóþùèé ìàññèâ èíäåêñîâ  {  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 2.11 — файл Combi.cpp

|  |
| --- |
| // --- main  #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 2.12 — файл Accommodations(Task4).cpp

Результат работы генератора размещений предоставлен на рисунке 2.4.

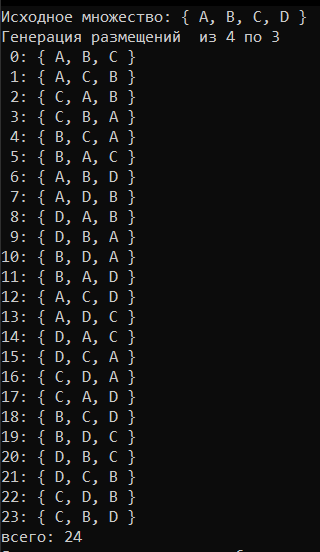


Рисунок 2.4 — результат работы генератора размещений

**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.);

Ниже в листингах представлены заголовочные файлы и файлы .cpp с реализацией поставленной задачи.

|  |
| --- |
| // --- Вoat.h  // -- решение задачи об оптимальной загрузке судна  // функция возвращает доход от перевози выбранных контейнеров  #pragma once  #include "../Combinations(Task2)/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 2.13 — файл Boat.h

|  |
| --- |
| // --- Boat.cpp  #include "Boat.h"  #include "../Combinations(Task2)/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 2.14 — 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 2.15 — файл Boat(Task5).cpp

Решение задачи об оптимальной загрузке судна показан на рисунке 2.5.

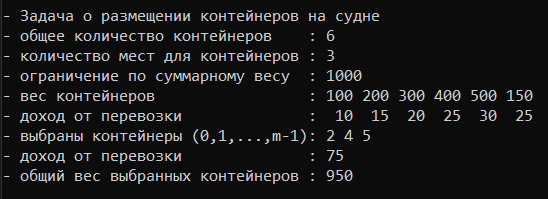


Рисунок 2.5 — решение задачи об оптимальной загрузке судна

**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:

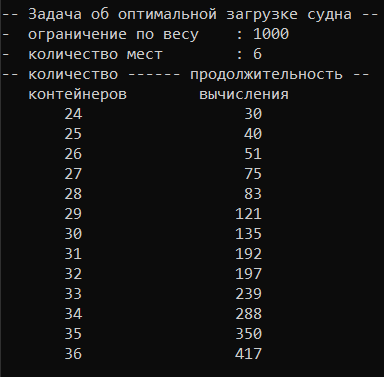
3, 7, 11, 15) об оптимальной загрузке судна (количество мест на судне для контейнеров – 6, количество контейнеров 25 – 35

Зависимость времени вычисления от количества контейнеров вычисляется с помощью кода, показанном в листинге 2.16.

|  |
| --- |
| #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 2.16

Результат выполнения кода и график зависимости приведён на рисунке 2.6.



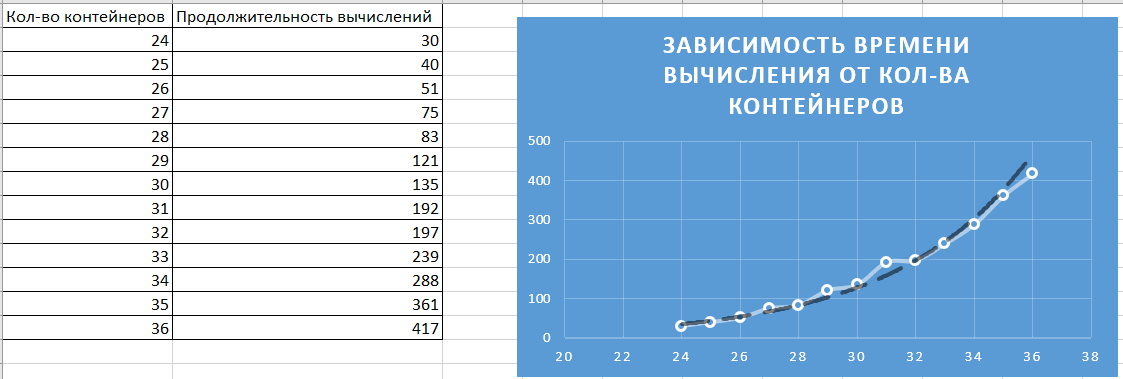


Рисунок 2.6 — Зависимость времени вычисления от кол-ва контейнеров

Вывод: исходя из полученных данных и графика, можно заметить, что скорость выполнения программы плавно возрастает при добавлении количества контейнеров.

# 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 (вариант 11):**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 22 | 32 |  | 11 |
| **2** | 11 |  | 26 | 57 | 73 |
| **3** | 13 | 33 |  | 86 | 60 |
| **4** | 28 | 47 | 44 |  | 33 |
| **5** | 82 | 77 | 52 | 24 |  |

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

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

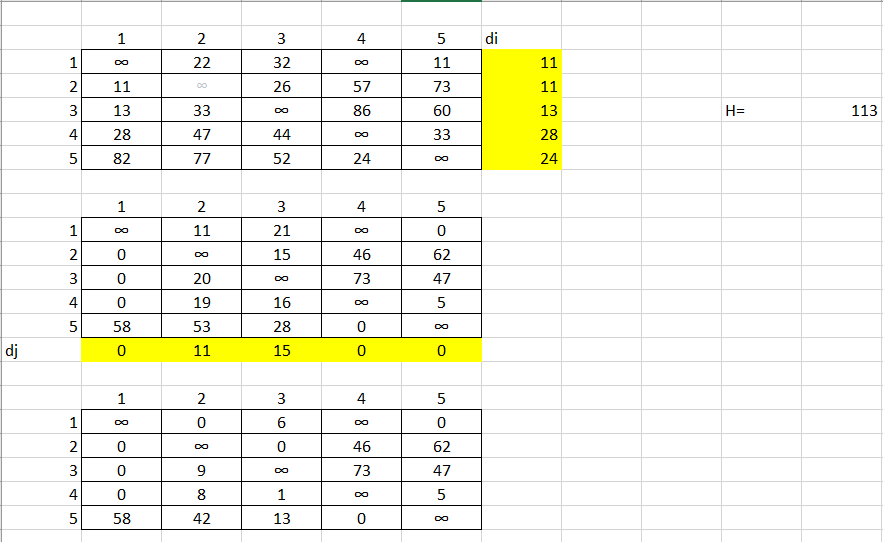
Гамильтонов цикл – такой путь, который проходит через все вершины графа ровно один раз.

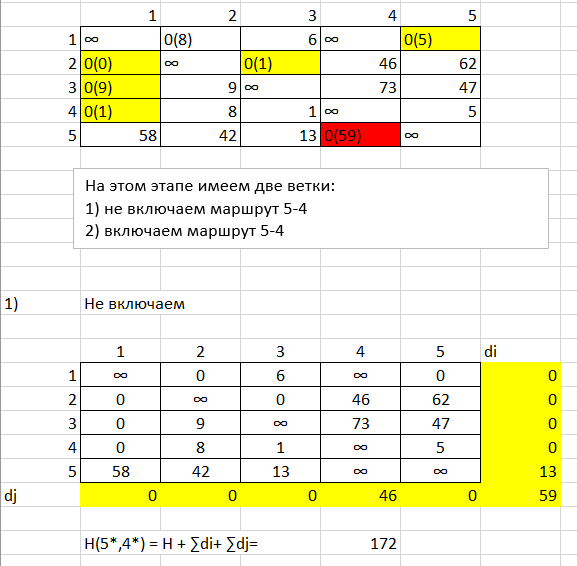
Параметр в задаче коммивояжера с параметром - это число, которое ограничивает вес ребра, которое может быть включено в найденный гамильтонов цикл.

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

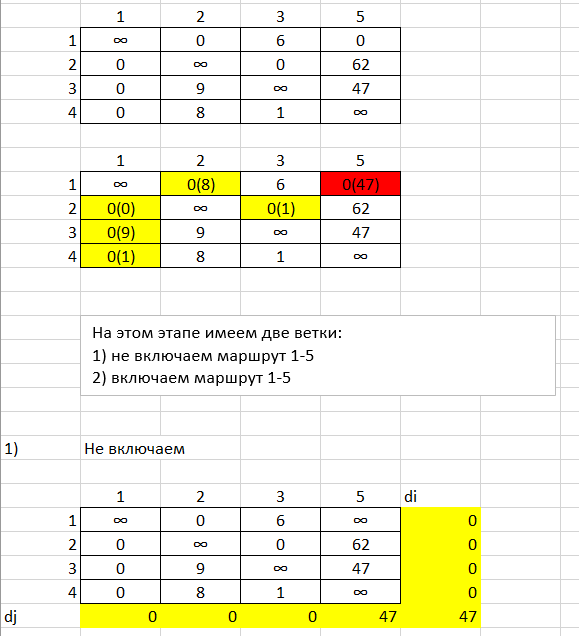
Цель задачи коммивояжера с параметром - найти гамильтонов цикл минимального веса в заданном графе, который удовлетворяет ограничению на максимальный вес ребра. Значение параметра может быть задано заранее или получено в результате вычислений, и влияет на решение задачи.

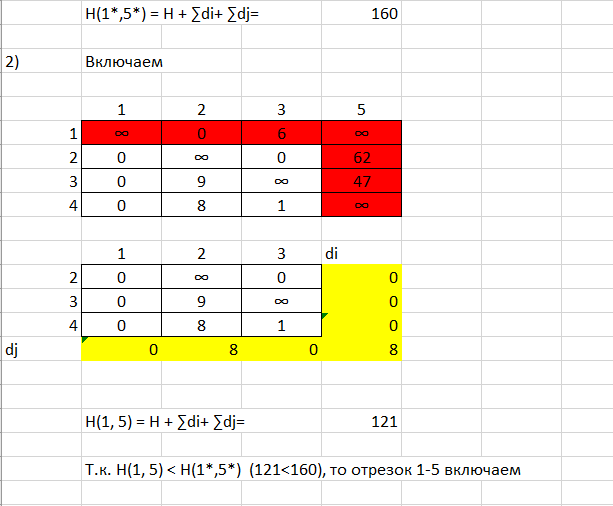
**Задание 2:**

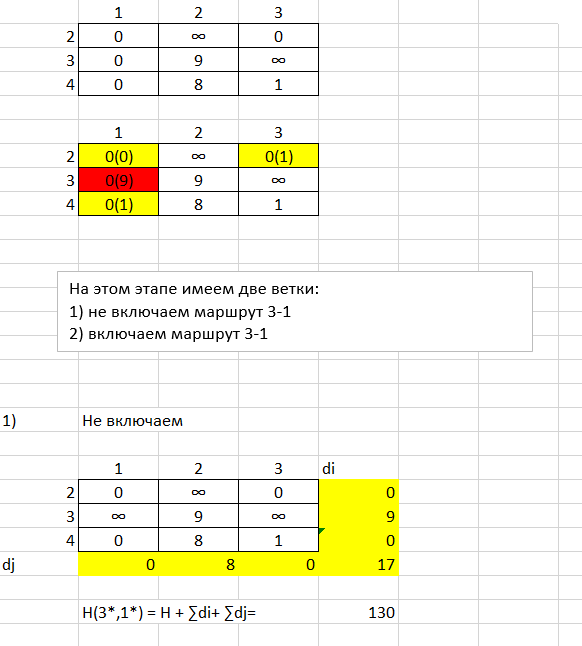
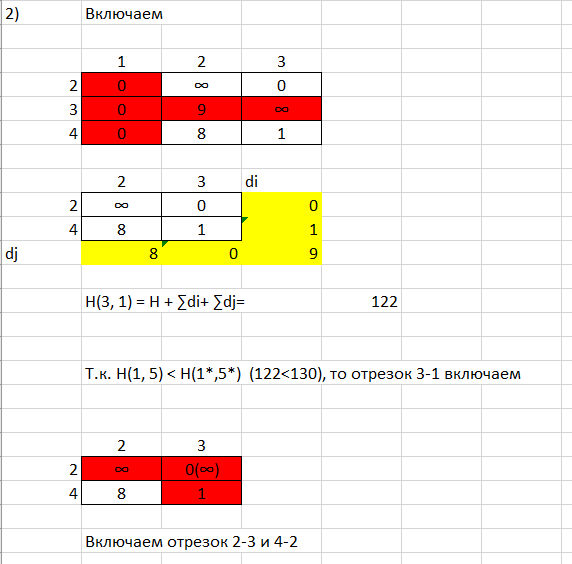












В результате мы получили следующий оптимальный маршрут: **121 (1->5->4->2->3->1)**

Граф обхода матрицы представлен на рисунке 1.

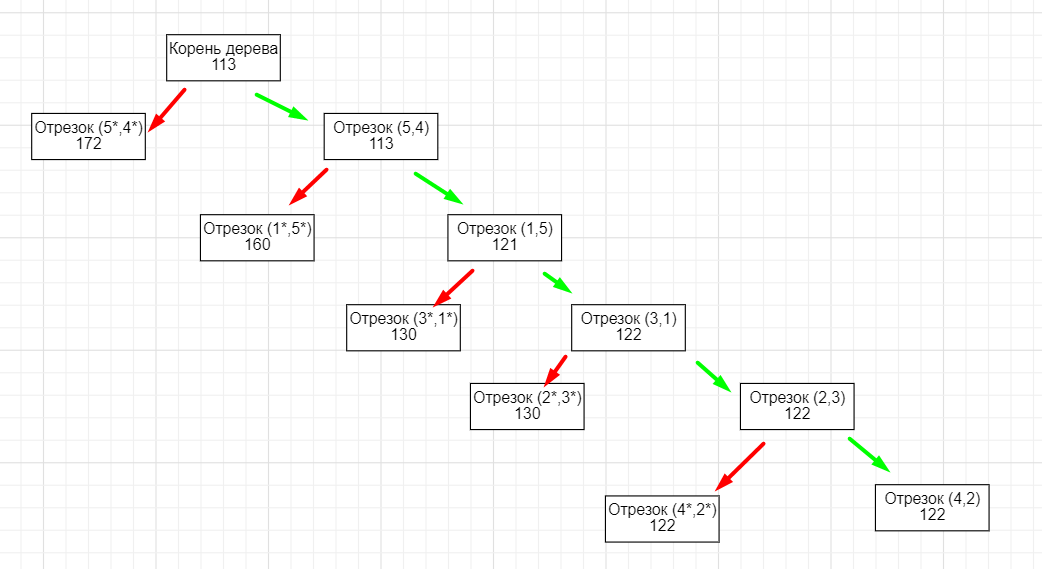


Рисунок 1 — обход исходной матрицы

**Задание 3.**

После проверки результата при помощи генератора перестановок из лабораторной работы 2, полученные ответы совпадали с исходным решением. Демонстрация выполнения программы представлена на рисунке 2.

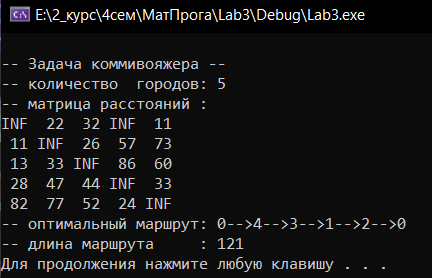


Рисунок 2 — результат выполнения программы

**Вывод**: результаты после решения вручную и после использования генератора совпали.

# Лабораторная работа №4. Динамическое программирование

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

**Ход выполнения работы**

**Задание 1.** На языке С++ сгенерировать случайным образом строку букв латинского алфавита  длиной  символов и длиной .

**Решение:**

Исходный код программы приведен в листинге 3.1, а результат показан на рисунке 3.1.

|  |
| --- |
| #include <iostream>  #include <ctime>  #include <Windows.h>  using namespace std;  #define FIRST\_LEN 300  #define SECOND\_LEN 250  char\* GenerateRandomString(int size)  {  char\* str = (char\*)malloc(sizeof(char) \* size);  for (int i = 0; i < size; i++) {  str[i] = rand() % 26 + 'a'; // 26 букв в алфавите  }  return str;  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  srand(time(NULL));  char\* s1 = GenerateRandomString(FIRST\_LEN);  cout << "S1: " << endl;  for (int i = 0; i < FIRST\_LEN; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s1[i];  }  cout << endl << endl;  char\* s2 = GenerateRandomString(SECOND\_LEN);  cout << "S2: " << endl;  for (int i = 0; i < SECOND\_LEN; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s2[i];  }  cout << endl << endl;  } |

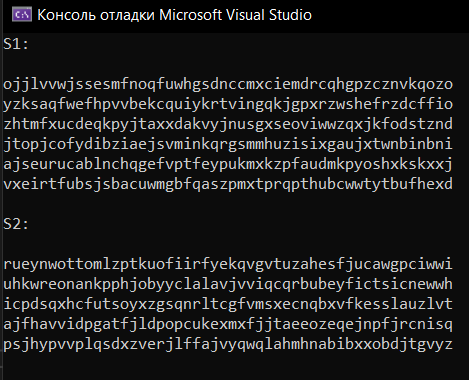
Листинг 3.1 — файл Task1(String\_generator).cpp  


Рисунок 3.1 — результат

**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).

**Решение:**

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

Исходный код реализации через динамическое программирование:

|  |
| --- |
| #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); // минимальное из трех  }  int levenshtein(int lx, const char x[], int ly, const char y[])  {  // длина слова х, слово длиной lx  // длина слова y, слово длиной ly  int\* d = new int[(lx + 1) \* (ly + 1)];  for (int i = 0; i <= lx; i++) DD(i, 0) = i; // длина строки х = i (кол-во символов в слове x), а DD возвращ. саму строку  for (int j = 0; j <= ly; j++) DD(0, j) = j; // длина строки y = 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); // возвращает длину слова и строки  } |

Листинг 3.2 — вычисление дистанции Левенштейна с помощью динамического программирования

Пример реализации рекурсивным методом:

|  |
| --- |
| #include <iomanip>  #include <algorithm>  #include "Levenshtein.h"  #define DD(i,j) d[(i)\*(ly+1)+(j)]  int levenshtein\_r( // рекурсия  int lx, const char x[],  int ly, const char y[]  )  {  int rc = 0; // результат, дистанция  if (lx == 0) rc = ly; // если длина x = 0, то дистанция = длине слова y  else if (ly == 0) rc = lx;  else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0; // если длина x = длина y = 1 и первые буквы совапали, дистанция = 0  else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1; // если длина x = длина y = 1 и первые буквы не совапали, дистанция = 1  else rc = min3(  levenshtein\_r(lx - 1, x, ly, y) + 1,  levenshtein\_r(lx, x, ly - 1, y) + 1,  levenshtein\_r(lx - 1, x, ly - 1, y) + (x[lx - 1] == y[ly - 1] ? 0 : 1)  );  return rc;  }; |

Листинг 3.3 — вычисление дистанции Левенштейна с помощью рекурсии

Полный код программы находится в листингах 3.4, 3.5, 3.6.

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

Листинг 3.4 — файл Levenshtein.h

|  |
| --- |
| // - Levenshtein.cpp  #include <iomanip>  #include <algorithm>  #include "Levenshtein.h"  #define DD(i,j) d[(i)\*(ly+1)+(j)]  int min3(int x1, int x2, int x3)  {  return std::min(std::min(x1, x2), x3); // минимальное из трех  }  int levenshtein(int lx, const char x[], int ly, const char y[])  {  // длина слова х, слово длиной lx  // длина слова y, слово длиной ly  int\* d = new int[(lx + 1) \* (ly + 1)];  for (int i = 0; i <= lx; i++) DD(i, 0) = i; // длина строки х = i (кол-во символов в слове x), а DD возвращ. саму строку  for (int j = 0; j <= ly; j++) DD(0, j) = j; // длина строки y = 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); // возвращает длину слова и строки  }  int levenshtein\_r( // рекурсия  int lx, const char x[],  int ly, const char y[]  )  {  int rc = 0; // результат, дистанция  if (lx == 0) rc = ly; // если длина x = 0, то дистанция = длине слова y  else if (ly == 0) rc = lx;  else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0; // если длина x = длина y = 1 и первые буквы совапали, дистанция = 0  else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1; // если длина x = длина y = 1 и первые буквы не совапали, дистанция = 1  else rc = min3(  levenshtein\_r(lx - 1, x, ly, y) + 1,  levenshtein\_r(lx, x, ly - 1, y) + 1,  levenshtein\_r(lx - 1, x, ly - 1, y) + (x[lx - 1] == y[ly - 1] ? 0 : 1)  );  return rc;  }; |

Листинг 3.5 — файл Levenshtein.cpp

|  |
| --- |
| // --- main  // вычисление дистанции (расстояния) Левенштейна  #include <algorithm>  #include <iostream>  #include <ctime>  #include <iomanip>  #include "Levenshtein.h"  #include <Windows.h>  #define FIRST\_LEN 300  #define SECOND\_LEN 250  using namespace std;  char\* GenerateRandomString(int size)  {  char\* str = (char\*)malloc(sizeof(char) \* size);  for (int i = 0; i < size; i++) {  str[i] = rand() % 26 + 'a'; // 26 букв в алфавите  }  return str;  }  int main()  {  SetConsoleCP(1251);  SetConsoleOutputCP(1251);  char\* s1 = GenerateRandomString(FIRST\_LEN);  cout << "S1: " << endl;  for (int i = 0; i < FIRST\_LEN; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s1[i];  }  cout << endl << endl;  srand(time(NULL) + 1);  char\* s2 = GenerateRandomString(SECOND\_LEN);  cout << "S2: " << endl;  for (int i = 0; i < SECOND\_LEN; i++) {  if (i % 50 == 0)  {  cout << "\n";  }  cout << s2[i];  }  cout << endl << endl;  clock\_t t1 = 0, t2 = 0, t3 = 0, t4 = 0;  int lx = sizeof(s1);  int ly = sizeof(s2);  int s1\_size[]{ FIRST\_LEN / 25, FIRST\_LEN / 20, FIRST\_LEN / 15, FIRST\_LEN / 10, FIRST\_LEN / 5, FIRST\_LEN / 2, FIRST\_LEN };  int s2\_size[]{ SECOND\_LEN / 25, SECOND\_LEN / 20, SECOND\_LEN / 15, SECOND\_LEN / 10, SECOND\_LEN / 5, SECOND\_LEN / 2, SECOND\_LEN };  cout << "\n\n-- расстояние Левенштейна -----";  cout << "\n\n--длина --- рекурсия -- дин.програм. ---\n";  for (int i = 0; i < min(lx, ly); i++)  {  t1 = clock();  levenshtein\_r(s1\_size[i], s1, s2\_size[i], s2);  t2 = clock();  t3 = clock();  levenshtein(s1\_size[i], s1, s2\_size[i], s2);  t4 = clock();  cout << right << setw(2) << s1\_size[i] << "/" << setw(2) << s2\_size[i]  << " " << left << setw(10) << (t2 - t1)  << " " << setw(10) << (t4 - t3) << endl;  }  system("pause");  return 0;  } |

Листинг 3.6 — файл Task2(Levenshtein).cpp

На рисунке 3.2 представлены дистанции Левенштейна вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

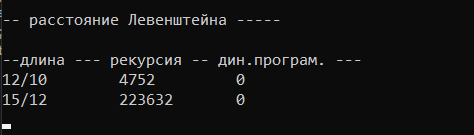


Рисунок 3.2 — результат программы

**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).

**Решение:**

На графике, представленном на рисунке 3.3, можно заметить, что выполненные с помощью динамического алгоритма, вычисления производятся в разы быстрее, чем с помощью рекурсивного алгоритма.

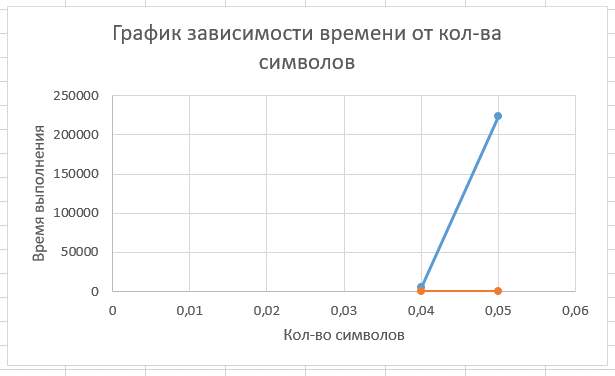


Рисунок 3.3 — сравнительный анализ времени выполнения

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

|  |  |
| --- | --- |
| Задание 4 (Вариант 11) | |
| Лот | Полет |

**Решение:**

1. 
2. 
3. 
4. 
5.   = 5  = 4
6.   = 4  = 3
7. 
8. 
9. 
10.   = 3  = 2
11.   = 3  = 2
12.  = 1
13.  = 2  = 1
14.   = 2 = 1
15.  =   = 1 = 1 = 0
16. 
17. 
18. 
19. 
20. 
21. 
22. 
23. 
24. 
25. 
26. 
27. 
28. 
29. 
30. 

**Задание 5. (нет решения, есть excel)**

**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.

|  |  |
| --- | --- |
| Задание 5 (вариант 11) | |
| TOUEXAZ | HIEHXZ |

**Решение:**

|  |
| --- |
| // - LCS.h  // -- рекурсивное вычисление длины LCS  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

|  |
| --- |
| // - LCS.cpp  // -- рекурсивное вычисление длины LCS  #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

|  |
| --- |
| // - main  // -- вычисления длины LCS  #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[] = "TOUEXAZ";  char Y[] = "HIEHXZ";  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, "TOUEXAZ", sizeof(Y) - 1, "HIEHXZ");  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 Task5(LCS)

На рисунке 5.1 представлена наибольшая общая подпоследовательность последовательностей Х и У, вычисленные при помощи метода динамического программирования, а также рекурсивным алгоритмом.

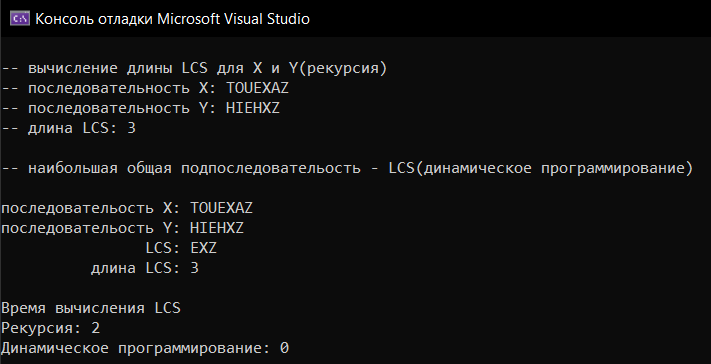


Рисунок 5.1 — результат выполнения программы

На рисунке 5.2 представлен график зависимости от суммарной длинны двух подстрок и необходимом времени выполнения при помощи динамического алгоритма:

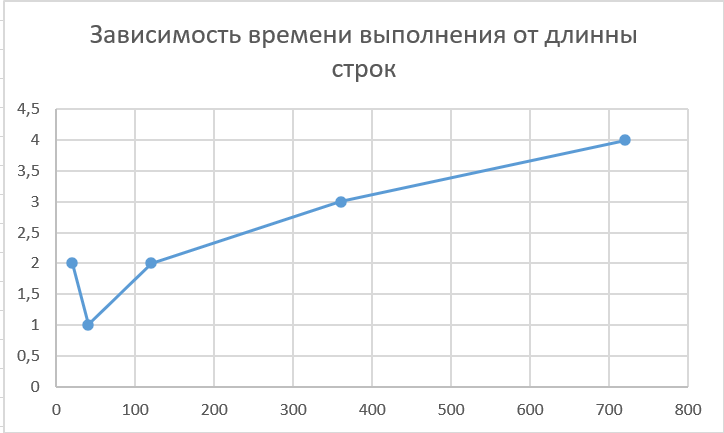


Рисунок 5.2 – График зависимости

**Алгоритм прохождения:**

1. Создаем двумерный массив размером (len(TOUEXAZ) + 1) на (len(HIEHXZ) + 1), где каждый элемент равен 0.

2. Проходим по каждой строке и столбцу массива, начиная с первого и до последнего.

3. Если значение строки или столбца равно 0, то элемент массива на пересечении этой строки и столбца остается равным 0.

4. Если символ строки и столбца равны, то элемент массива на пересечении этой строки и столбца равен значению элемента на предыдущей диагонали плюс 1, т.е. с\_(𝑛,𝑝) = c\_(𝑛−1,𝑝−1) + 1.

5. Если символ строки и столбца не равны, то элемент массива на пересечении этой строки и столбца равен максимуму между значением элемента на предыдущей строке и значением элемента на предыдущем столбце, т.е. c\_(𝑛,𝑝) = max⁡( с\_(𝑛,𝑝"−1" ), c\_(𝑛−1,𝑝)).

6. После прохода по всем элементам массива, наибольшая общая подпоследовательность будет равна значению элемента на пересечении последней строки и последнего столбца.

Алгоритм прохода будет выглядеть следующим образом:

1)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | T | O | U | E | X | A | Z |
| H | 0 |  |  |  |  |  |  |
| I |  |  |  |  |  |  |  |
| E |  |  |  |  |  |  |  |
| H |  |  |  |  |  |  |  |
| X |  |  |  |  |  |  |  |
| Z |  |  |  |  |  |  |  |

2)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | T | O | U | E | X | A | Z |
| H | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| E | 0 | 0 | 0 | 1 |  |  |  |
| H |  |  |  |  |  |  |  |
| X |  |  |  |  |  |  |  |
| Z |  |  |  |  |  |  |  |

3)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | T | O | U | E | X | A | Z |
| H | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| E | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| H | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X | 1 | 1 | 1 | 1 | 2 |  |  |
| Z |  |  |  |  |  |  |  |

4)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | T | O | U | E | X | A | Z |
| H | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| I | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| E | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| H | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| X | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| Z | 2 | 2 | 2 | 2 | 2 | 2 | 3 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | T | O | U | E | X | A | Z |
| H |  |  |  |  |  |  |  |
| I |  |  |  |  |  |  |  |
| E |  |  |  | ⬉ |  |  |  |
| H |  |  |  | ⬆ |  |  |  |
| X |  |  |  |  | ⬉ | ⬅ |  |
| Z |  |  |  |  |  |  | ⬉ |

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

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

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

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

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

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

# Лабораторная работа №5. Транспортная задача

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

**Задание для выполнения:**

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

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

Вариант 11:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **23** | **13** | **17** | **14** | **22** | **12** | **179** |
| 2 | **21** | **11** | **19** | **16** | **18** | **24** | **124** |
| 3 | **12** | **16** | **22** | **19** | **13** | **22** | **161** |
| 4 | **15** | **21** | **21** | **14** | **24** | **13** | **170** |
| 5 | **14** | **22** | **20** | **11** | **21** | **15** | **111** |
| ПОТРЕБНОСТИ | **154** | **118** | **142** | **204** | **106** | **174** |  |

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

∑запасы = 179 + 124 + 161 + 170 + 111 = 745

∑потребности = 154 + 118 + 142 + 204 + 106 + 174 = 898

Так как запасы поставщиков меньше потребности потребителей, введем фиктивного поставщика 6, с запасом продукции равным 898-745=153. Стоимость доставки единицы продукции от фиктивного поставщика ко всем потребителям примем равной нулю.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14 | 22 | 12 | 179 |
| 2 | 21 | 11 | 19 | 16 | 18 | 24 | 124 |
| 3 | 12 | 16 | 22 | 19 | 13 | 22 | 161 |
| 4 | 15 | 21 | 21 | 14 | 24 | 13 | 170 |
| 5 | 14 | 22 | 20 | 11 | 21 | 15 | 111 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| Потребность | 154 | 118 | 142 | 204 | 106 | 174 |  |

Теперь выполняется условие .

Этап I

*Метод наименьшей стоимости*

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

Искомый элемент равен c22=11

x22 = min(124,118) = 118.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14 | 22 | 12 | 179 |
| 2 | 21 | **11** | 19 | 16 | 18 | 24 | 124-118=6 |
| 3 | 12 | 16 | 22 | 19 | 13 | 22 | 161 |
| 4 | 15 | 21 | 21 | 14 | 24 | 13 | 170 |
| 5 | 14 | 22 | 20 | 11 | 21 | 15 | 111 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| Потребность | 154 | 118-118=0 | 142 | 204 | 106 | 174 |  |

Искомый элемент равен c54=11.

x54 = min(111,204) = 111.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14 | 22 | 12 | 179 |
| 2 | 21 | 11|118 | 19 | 16 | 18 | 24 | 6 |
| 3 | 12 | 16 | 22 | 19 | 13 | 22 | 161 |
| 4 | 15 | 21 | 21 | 14 | 24 | 13 | 170 |
| 5 | 14 | 22 | 20 | **11** | 21 | 15 | 111-111=0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| Потребность | 154 | 0 | 142 | 204-111=93 | 106 | 174 |  |

Искомый элемент равен c16=12.

x16 = min(179,174) = 174.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14 | 22 | **12** | 179-174=5 |
| 2 | 21 | 11|118 | 19 | 16 | 18 | 24 | 6 |
| 3 | 12 | 16 | 22 | 19 | 13 | 22 | 161 |
| 4 | 15 | 21 | 21 | 14 | 24 | 13 | 170 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| Потребность | 154 | 0 | 142 | 93 | 106 | 174-174=0 |  |

Искомый элемент равен c31=12.

x31 = min(161,154) = 154.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14 | 22 | 12|174 | 5 |
| 2 | 21 | 11|118 | 19 | 16 | 18 | 24 | 6 |
| 3 | **12** | 16 | 22 | 19 | 13 | 22 | 161-154=7 |
| 4 | 15 | 21 | 21 | 14 | 24 | 13 | 170 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| Потребность | 154-154=0 | 0 | 142 | 93 | 106 | 0 |  |

Искомый элемент равен c35=13.

x35 = min(7,106) = 7.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14 | 22 | 12|174 | 5 |
| 2 | 21 | 11|118 | 19 | 16 | 18 | 24 | 6 |
| 3 | 12|154 | 16 | 22 | 19 | **13** | 22 | 7-7=0 |
| 4 | 15 | 21 | 21 | 14 | 24 | 13 | 170 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| Потребность | 0 | 0 | 142 | 93 | 106-7=99 | 0 |  |

Искомый элемент равен c14=14.

x14 = min(5,93) = 5.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | **14** | 22 | 12|174 | 5-5=0 |
| 2 | 21 | 11|118 | 19 | 16 | 18 | 24 | 6 |
| 3 | 12|154 | 16 | 22 | 19 | 13|7 | 22 | 0 |
| 4 | 15 | 21 | 21 | 14 | 24 | 13 | 170 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| Потребность | 0 | 0 | 142 | 93-5=88 | 99 | 0 |  |

Искомый элемент равен c44=14.

x44 = min(170,88) = 88.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | 0 |
| 2 | 21 | 11|118 | 19 | 16 | 18 | 24 | 6 |
| 3 | 12|154 | 16 | 22 | 19 | 13|7 | 22 | 0 |
| 4 | 15 | 21 | 21 | **14** | 24 | 13 | 170-88=82 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| Потребность | 0 | 0 | 142 | 88-88=0 | 99 | 0 |  |

Искомый элемент равен c25=18.

x25 = min(6,99) = 6.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | 0 |
| 2 | 21 | 11|118 | 19 | 16 | **18** | 24 | 6-6=0 |
| 3 | 12|154 | 16 | 22 | 19 | 13|7 | 22 | 0 |
| 4 | 15 | 21 | 21 | 14|88 | 24 | 13 | 82 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| Потребность | 0 | 0 | 142 | 0 | 99-6=93 | 0 |  |

Искомый элемент равен c43=21.

x43 = min(82,142) = 82.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | 0 |
| 2 | 21 | 11|118 | 19 | 16 | 18|6 | 24 | 0 |
| 3 | 12|154 | 16 | 22 | 19 | 13|7 | 22 | 0 |
| 4 | 15 | 21 | **21** | 14|88 | 24 | 13 | 82-82=0 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 153 |
| Потребность | 0 | 0 | 142-82=60 | 0 | 93 | 0 |  |

Искомый элемент равен c63=0.

x63 = min(153,60) = 60.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | 0 |
| 2 | 21 | 11|118 | 19 | 16 | 18|6 | 24 | 0 |
| 3 | 12|154 | 16 | 22 | 19 | 13|7 | 22 | 0 |
| 4 | 15 | 21 | 21|82 | 14|88 | 24 | 13 | 0 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | **0** | 0 | 0 | 0 | 153-60=93 |
| Потребность | 0 | 0 | 60-60=0 | 0 | 93 | 0 |  |

Искомый элемент равен c65=0.

x65 = min(93,93) = 93.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | 0 |
| 2 | 21 | 11|118 | 19 | 16 | 18|6 | 24 | 0 |
| 3 | 12|154 | 16 | 22 | 19 | 13|7 | 22 | 0 |
| 4 | 15 | 21 | 21|82 | 14|88 | 24 | 13 | 0 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0|60 | 0 | **0** | 0 | 93-93=0 |
| Потребность | 0 | 0 | 0 | 0 | 93-93=0 | 0 |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | 0 |
| 2 | 21 | 11|118 | 19 | 16 | 18|6 | 24 | 0 |
| 3 | 12|154 | 16 | 22 | 19 | 13|7 | 22 | 0 |
| 4 | 15 | 21 | 21|82 | 14|88 | 24 | 13 | 0 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0|60 | 0 | 0|93 | 0 | 0 |
| Потребность | 0 | 0 | 0 | 0 | 0 | 0 |  |

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

Подсчитаем число базисных переменных. Их должно быть m + n - 1 = 11

Значение целевой функции для этого опорного плана равно:

F(x) = 14\*5 + 12\*174 + 11\*118 + 18\*6 + 12\*154 + 13\*7 + 21\*82 + 14\*88 + 11\*111 + 0\*60 + 0\*93 = 10874

Этап II

*Метод потенциалов*

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

ui + vj = cij

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

u1 + v4 = 14; 0 + v4 = 14; v4 = 14

u1 + v6 = 11; 0 + v6 = 11; v6 = 11

u4 + v4 = 14; 14 + u4 = 14; u4 = 0

u4 + v3 = 21; 0 + v3 = 21; v3 = 21

u6 + v3 = 0; 21 + u6 = 0; u6 = -21

u6 + v5 = 0; -21 + v5 = 0; v5 = 21

u2 + v5 = 18; 21 + u2 = 18; u2 = -3

u2 + v2 = 11; -3 + v2 = 11; v2 = 14

u5 + v4 = 11; 14 + u5 = 11; u5 = -3

u3 + v5 = 13; 21 + u3 = 13; u3 = -8

u3 + v1 = 12; -8 + v1 = 12; v1 = 20

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | U |
| b 1 | b 2 | b3 | b 4 | b 5 | b 6 |
| a 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | u1 = 0 |
| a 2 | 21 | 11|118 | 19 | 16 | 18|6 | 24 | u2 = -3 |
| a 3 | 12|154 | 16 | 22 | 19 | 13|7 | 22 | u3 = -8 |
| a 4 | 15 | 21 | 21|82 | 14|88 | 24 | 13 | u4 = 0 |
| a 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | u5 =  -3 |
| a 6 | 0 | 0 | 0|60 | 0 | 0|93 | 0 | u6 = -21 |
| V | v1 = 20 | v2 = 14 | v3 = 21 | v4 = 14 | v5 = 21 | v6 = 11 |  |

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

(1;2): ∆12 = 0 + 14 - 13 = 1 > 0

(1;3): ∆13 = 0 + 21 - 17 = 4 > 0

(4;1): ∆41 = 0 + 20 - 15 = 5 > 0

(5;1): ∆51 = -3 + 20 - 14 = 3 > 0

max(1,4,5,3) = 5

Выбираем максимальную оценку свободной клетки (4;1): 15. Для этого в перспективную клетку (4;1) поставим знак «+», а в остальных вершинах многоугольника чередующиеся знаки «-», «+», «-».

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | 0 |
| 2 | 21 | 11|118 | 19 | 16 | 18|6 | 24 | 0 |
| 3 | 12|154[-] | 16 | 22 | 19 | 13|7[+] | 22 | 0 |
| 4 | 15[+] | 21 | 21|82[-] | 14|88 | 24 | 13 | 0 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0|60[+] | 0 | 0|93[-] | 0 | 0 |
| Потребность | 0 | 0 | 0 | 0 | 0 | 0 |  |

Из грузов хij стоящих в минусовых клетках, выбираем наименьшее, т.е. прибавляем 82 к объемам грузов, стоящих в плюсовых и вычитаем 82 из xij, стоящих в минусовых клетках. В результате получим новый опорный план.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | 0 |
| 2 | 21 | 11|118 | 19 | 16 | 18|6 | 24 | 0 |
| 3 | 12|72 | 16 | 22 | 19 | 13|89 | 22 | 0 |
| 4 | 15|82 | 21 | 21 | 14|88 | 24 | 13 | 0 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0|142 | 0 | 0|11 | 0 | 0 |
| Потребность | 0 | 0 | 0 | 0 | 0 | 0 |  |

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

u1 + v4 = 14; 0 + v4 = 14; v4 = 14

u1 + v6 = 12; 0 + v6 = 12; v6 = 12

u4 + v4 = 14; 14 + u4 = 14; u4 = 0

u5 + v4 = 11; 14 + u5 = 11; u5 = -3

u4 + v1 = 15; 0 + v1 = 15; v1 = 15

u3 + v1 = 12; 15 + u3 = 12; u3 = -3

u3 + v5 = 13; -3 + v5 = 13; v5 = 16

u2 + v5 = 18; 16 + u2 = 18; u2 = 2

u2 + v2 = 11; 2 + v2 = 11; v2 = 9

u6 + v5 = 0; 16 + u6 = 0; u6 = -16

u6 + v3 = 0; -16 + v3 = 0; v3 = 16

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | U |
| b 1 | b 2 | b3 | b 4 | b 5 | b 6 |
| a 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | u1 = 0 |
| a 2 | 21 | 11|118 | 19 | 16 | 18|6 | 24 | u2 = 2 |
| a 3 | 12|72 | 16 | 22 | 19 | 13|89 | 22 | u3 = -3 |
| a 4 | 15|82 | 21 | 21 | 14|88 | 24 | 13 | u4 = 0 |
| a 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | u5 = -3 |
| a 6 | 0 | 0 | 0|142 | 0 | 0|11 | 0 | u6 = -16 |
| V | v1 = 15 | v2 = 9 | v3 = 16 | v4 = 14 | v5 = 16 | v6 = 12 |  |

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

Минимальные затраты составят: F(x) = 14\*5 + 12\*174 + 11\*118 + 18\*6 + 12\*72 + 13\*89 + 15\*82 + 14\*88 + 11\*111 + 0\*142 + 0\*11 = 11468

*Анализ оптимального плана.*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Поставщик | Потребитель | | | | | | Запас |
| 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 23 | 13 | 17 | 14|5 | 22 | 12|174 | 0 |
| 2 | 21 | 11|118 | 19 | 16 | 18|6 | 24 | 0 |
| 3 | 12|154 | 16 | 22 | 19 | 13|7 | 22 | 0 |
| 4 | 15 | 21 | 21|82 | 14|88 | 24 | 13 | 0 |
| 5 | 14 | 22 | 20 | 11|111 | 21 | 15 | 0 |
| 6 | 0 | 0 | 0|60 | 0 | 0|93 | 0 | 0 |
| Потребность | 0 | 0 | 0 | 0 | 0 | 0 |  |

Из 1-го склада необходимо доставить 5 ед. товара 4-му потребителю и 174 ед. товара 6-му потребителю.

Из 2-го склада необходимо доставить 118 ед. товара 2-му потребителю и 6 ед. товара 5-му потребителю.

Из 3-го склада необходимо доставить 154 ед. товара 1-му потребителю и 7 ед. товара 5-му потребителю.

Из 4-го склада необходимо доставить 82 ед. товара 3-му потребителю и 88 ед. товара 4-му потребителю.

Из 5-го склада необходимо доставить 111 ед. товара 4-му потребителю.

Потребность 3-го потребителя остается неудовлетворенной на 60 ед.

Потребность 5-го потребителя остается неудовлетворенной на 93 ед.

# Лабораторная работа №6. Алгоритмы на графах

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

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

|  |  |
| --- | --- |
| Вариант | Граф G |
| 11 |  |

Матрица смежности - это квадратная матрица размера n x n, где n - число вершин в графе. Эта матрица используется для представления графа в виде таблицы, где каждый элемент матрицы a\_ij показывает, существует ли ребро между вершинами i и j.

Если в графе есть ребро между вершинами i и j, то a\_ij равно 1. Если ребра между этими вершинами нет, то a\_ij равно 0. Обратим внимание, что для неориентированного графа матрица смежности будет симметричной относительно главной диагонали.

**Матрица смежности:**

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

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

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

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

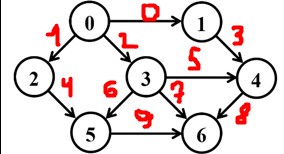
{0, 0, 0, 0, 0, 0, 1},

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

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



G = (V, E), V = {0, 1, 2, 3, 4, 5, 6}, E = {<0, 1>0, <0, 2>1, <0, 3>2, <1, 4>3, <2, 5>4, <3, 4>5, <3, 5>6, <3, 6>7, <4, 6>8, <5, 6>9}



\*если в двух словах, если ребро исходит от вершины, то 1, если входит, то -1, 0 – если ничего

**Матрица инцидентности:**

0 1 2 3 4 5 6 7 8 9

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

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

2{0, -1, 0, 1, 0, 0, 0, 0, 0, 0},

3{0, 0, -1, 0, 0, 1, 1, 1, 0, 0},

4{0, 0, 0, -1, 0, -1, 0, 0, 1, 0},

5{0, 0, 0, 0, -1, 0, -1, 0, 0, 1},

6{0, 0, 0, 0, 0, 0, 0, -1, -1, -1}

Список смежных вершин - это список всех вершин, с которыми заданная вершина в графе имеет ребра. Другими словами, это список вершин, которые непосредственно соединены с данной вершиной ребрами.

Например, если в графе есть вершина A, которая имеет ребра, соединяющие ее с вершинами B, C и D, то список смежных вершин для вершины A будет содержать вершины B, C и D.

Список смежных вершин может быть представлен в виде массива, списка или любой другой структуры данных, в которой хранятся вершины, с которыми данная вершина имеет ребра.

**Список смежных вершин:**

S0 = {1, 2, 3}

S1 = {4}

S2 = {5}

S3 = {4, 5, 6}

S4 = {6}

S5 = {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.

**Алгоритм поиска в ширину (англ. breadth-first search, BFS)** позволяет найти кратчайшие пути из одной вершины невзвешенного графа до всех остальных вершин

**Поиск в ширину:**

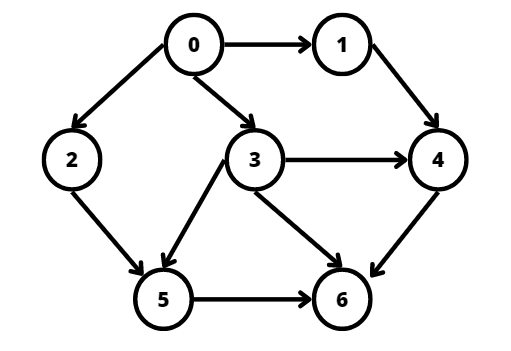
Q- для промежуточного хранения вершин(очередь)

массивы:

С- массив окраски вершин ( Б – не добавлена в очередь , С – добавлена в очередь , Ч – вышла из очереди )

D- массив расстояний

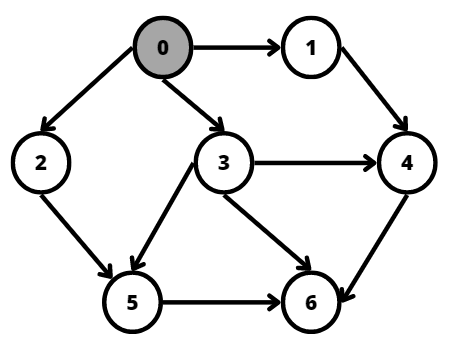
Р- массив предшествующих вершин



Шаг 1

Очередь : 0

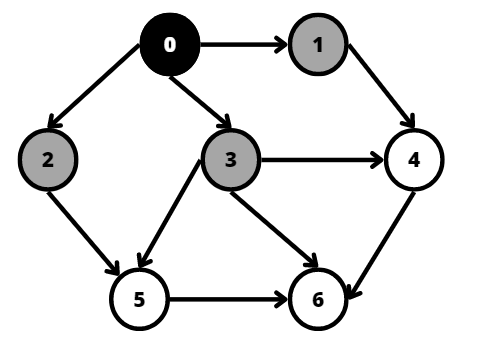
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Б | Б | Б | Б | Б | Б |
| Расстояния | 0 | - | - | - | - | - | - |
| Массив предыдущих вершин | - | - | - | - | - | - | - |



Шаг 2

Очередь : 1 2 3

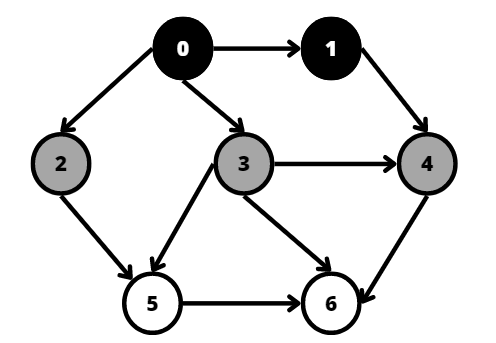
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | С | С | С | Б | Б | Б |
| Расстояния | 0 | 1 | 1 | 1 | - | - | - |
| Массив предыдущих вершин | - | 0 | 0 | 0 | - | - | - |



Шаг 3

Очередь : 2 3 4

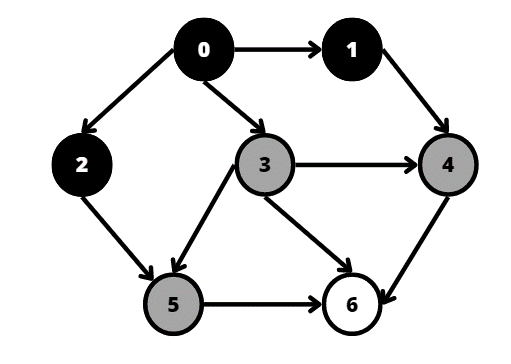
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Ч | С | С | С | Б | Б |
| Расстояния | 0 | 1 | 1 | 1 | 2 | - | - |
| Массив предыдущих вершин | - | 0 | 0 | 0 | 1 | - | - |



Шаг 4

Очередь : 3 4 5

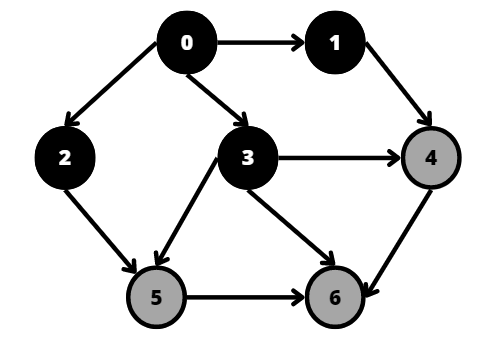
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Ч | Ч | С | С | С | Б |
| Расстояния | 0 | 1 | 1 | 1 | 2 | 1 | - |
| Массив предыдущих вершин | - | 0 | 0 | 0 | 1 | 2 | - |



Шаг 5

Очередь: 4 5 6

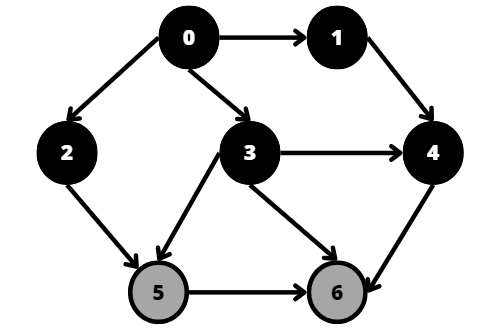
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Ч | Ч | Ч | С | С | С |
| Расстояния | 0 | 1 | 1 | 1 | 2 | 1 | 1 |
| Массив предыдущих вершин | - | 0 | 0 | 0 | 1 | 2 | 3 |



Шаг 6

Очередь: 5 6

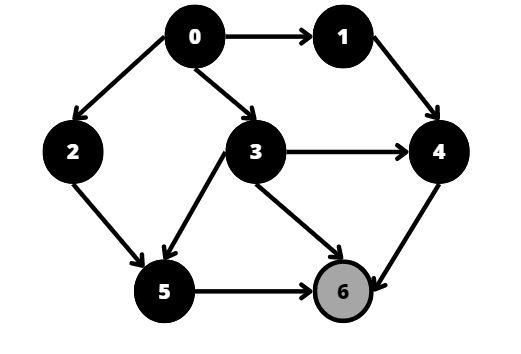
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Ч | Ч | Ч | Ч | С | С |
| Расстояния | 0 | 1 | 1 | 1 | 2 | 1 | 1 |
| Массив предыдущих вершин | - | 0 | 0 | 0 | 1 | 2 | 3 |



Шаг 7

Очередь: 6

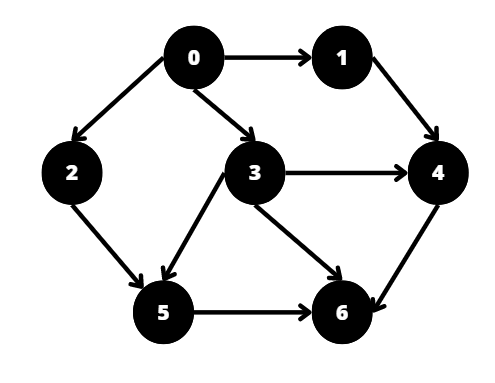
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Ч | Ч | Ч | Ч | Ч | С |
| Расстояния | 0 | 1 | 1 | 1 | 2 | 1 | 1 |
| Массив предыдущих вершин | - | 0 | 0 | 0 | 1 | 2 | 3 |



Шаг 8

Очередь: …

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | Ч | Ч | Ч | Ч | Ч | Ч | Ч |
| Расстояния | 0 | 1 | 1 | 1 | 2 | 1 | 1 |
| Массив предыдущих вершин | - | 0 | 0 | 0 | 1 | 2 | 3 |



Очередь пуста – следует конец обхода.

**Алгоритм BFS** сводится к следующей последовательности шагов.

1. Инициализировать массивы **С**, **D**, **P**. Стартовую вершину **s** поместить в очередь **Q**. и окрасить в серый цвет: **C[s] = G**. Для стартовой вершины установить расстояние, равное нулю: **D[s] = 0**.
2. Если очередь **Q** пуста, то работа алгоритма завершена, в противном случае перейти к следующему шагу.
3. Выбрать из очереди **Q** вершину **k** и окрасить ее в черный цвет: **С[k] = B**.
4. Построить множества **J** вершин белого цвета смежных вершине **k**. Если таких вершин нет, то перейти к шагу 2, иначе – к следующему шагу.
5. Каждую вершину **j** из множества **J** поместить в очередь **Q**. Обычно (но не обязательно) в очередь вершины помещаются в порядке возрастания номеров.
6. Каждую вершину **j** из множества **J** окрасить в серый цвет: **С[j] = G**.
7. Для каждой вершины **j** из множества **J** вычислитьрасстояние: **D[j] = D[k] + 1**.
8. Для каждой вершины **j** из множества **J** указать предшествующую вершину: **P[j] = k**.
9. Перейти к шагу 3.

**Алгоритм поиска (или обхода) в глубину** (англ. depth-first search, DFS) позволяет построить обход графа, при котором посещаются все вершины, доступные из начальной вершины.

**Алгоритм поиска в глубину:**

T-шаг

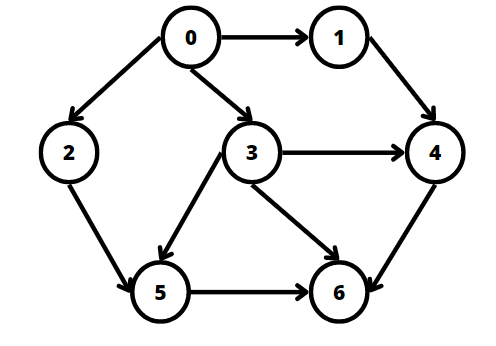
Стек посещённых вершин

Массив окраски вершин ( Б – не посещена , С – в стеке , Ч – вышла из стека)

Массив расстояний

Массив предшествующих вершин

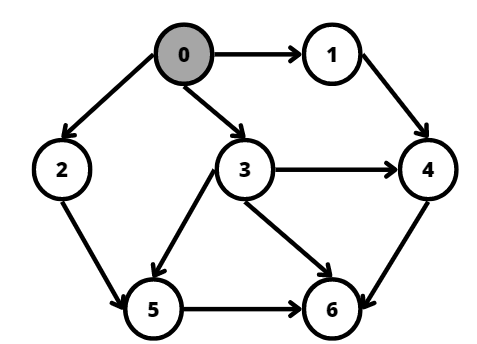
Массив шаг на котором вершина окрашивается в черный цвет



Шаг 1

Стек :

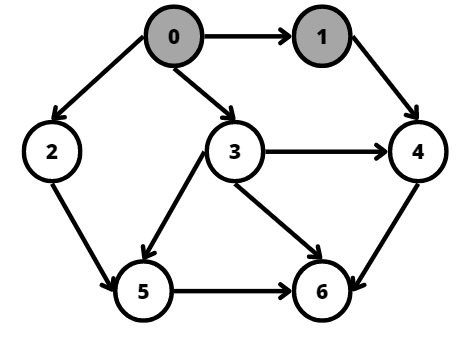
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Б | Б | Б | Б | Б | Б |
| Расстояния | 0 |  |  |  |  |  |  |
| Номера предыдущих вершин |  |  |  |  |  |  |  |
| Номер шага , когда вершина выходит из алгоритма |  |  |  |  |  |  |  |



Шаг 2

Стек : 0

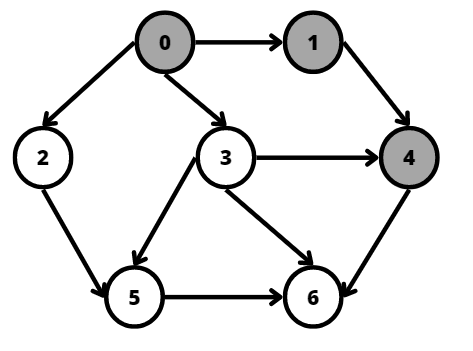
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | С | Б | Б | Б | Б | Б |
| Расстояния | 0 | 1 |  |  |  |  |  |
| Номера предыдущих вершин |  | 0 |  |  |  |  |  |
| Номер шага , когда вершина выходит из алгоритма |  |  |  |  |  |  |  |



Шаг 3

Стек : 0 1

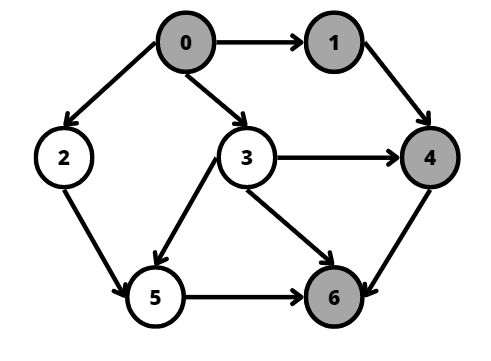
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | С | Б | Б | С | Б | Б |
| Расстояния | 0 | 1 |  |  | 2 |  |  |
| Номера предыдущих вершин |  | 0 |  |  | 1 |  |  |
| Номер шага , когда вершина выходит из алгоритма |  |  |  |  |  |  |  |



Шаг 4

Стек : 0 1 4

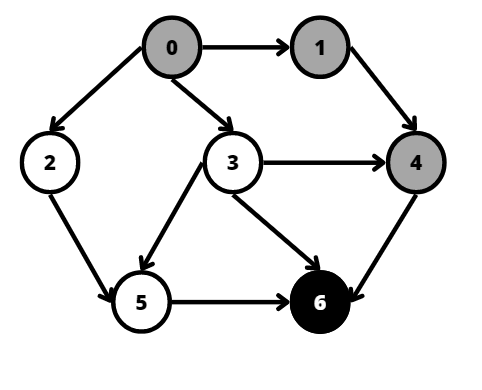
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | С | Б | С | С | Б | С |
| Расстояния | 0 | 1 |  |  | 2 |  | 3 |
| Номера предыдущих вершин |  | 0 |  |  | 1 |  | 4 |
| Номер шага , когда вершина выходит из алгоритма |  |  |  |  |  |  |  |



Шаг 5

Стек : 0 1

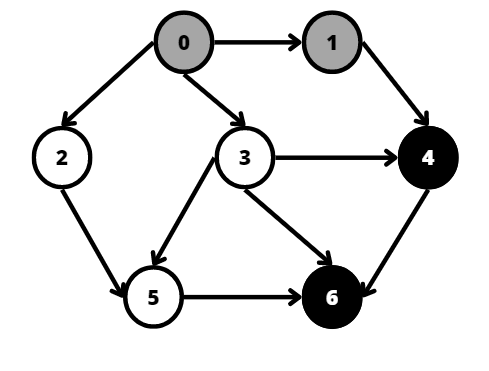
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | С | Б | С | С | Б | Ч |
| Расстояния | 0 | 1 |  | 2 | 3 |  | 4 |
| Номера предыдущих вершин |  | 0 |  | 1 | 3 |  | 4 |
| Номер шага , когда вершина выходит из алгоритма |  |  |  |  |  |  | 5 |



Шаг 6

Стек : 0

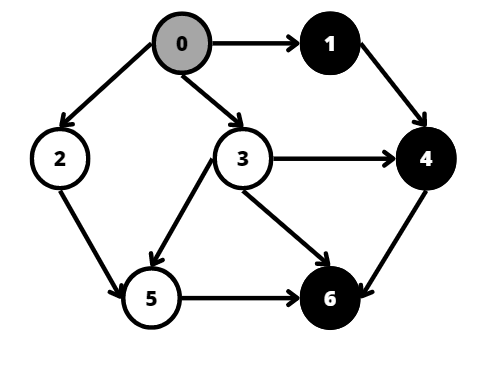
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | С | Б | С | Ч | Б | Ч |
| Расстояния | 0 | 1 |  | 2 | 3 |  | 4 |
| Номера предыдущих вершин |  | 0 |  | 1 | 3 |  | 4 |
| Номер шага , когда вершина выходит из алгоритма |  |  |  |  | 6 |  | 5 |



Шаг 7

Стек : 0

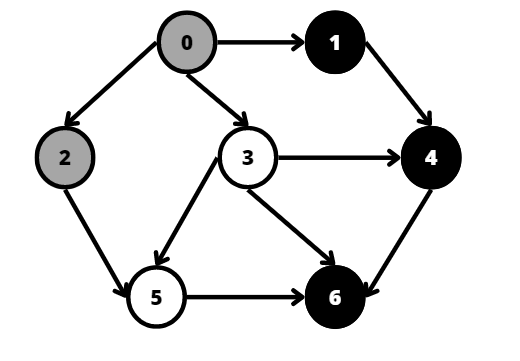
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | Б | С | Ч | Б | Ч |
| Расстояния | 0 | 1 |  | 2 | 3 |  | 4 |
| Номера предыдущих вершин |  | 0 |  | 1 | 3 |  | 4 |
| Номер шага , когда вершина выходит из алгоритма |  | 7 |  |  | 6 |  | 5 |



Шаг 8

Стек : 0

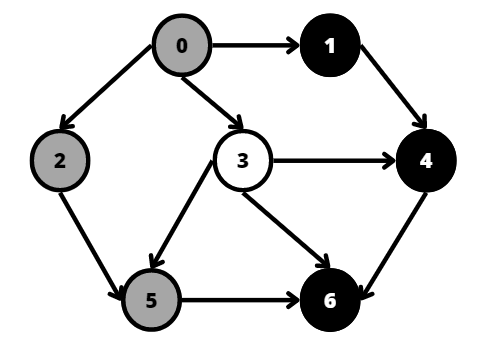
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | С | С | Ч | Б | Ч |
| Расстояния | 0 | 1 | 1 | 2 | 3 |  | 4 |
| Номера предыдущих вершин |  | 0 | 0 | 1 | 3 |  | 4 |
| Номер шага , когда вершина выходит из алгоритма |  | 7 |  |  | 6 |  | 5 |



Шаг 8

Стек : 0, 2

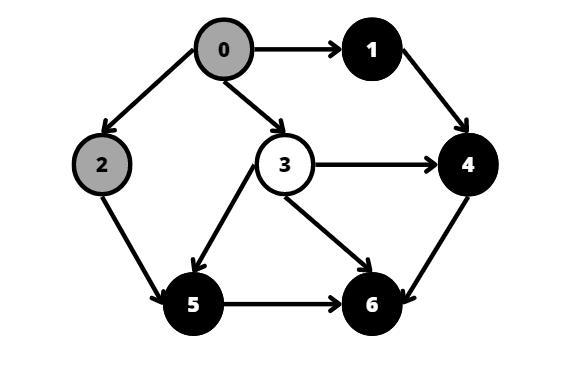
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | С | С | Ч | С | Ч |
| Расстояния | 0 | 1 | 1 | 2 | 3 | 2 | 4 |
| Номера предыдущих вершин |  | 0 | 0 | 1 | 3 | 2 | 4 |
| Номер шага , когда вершина выходит из алгоритма |  | 7 |  |  | 6 |  | 5 |



Шаг 9

Стек : 0

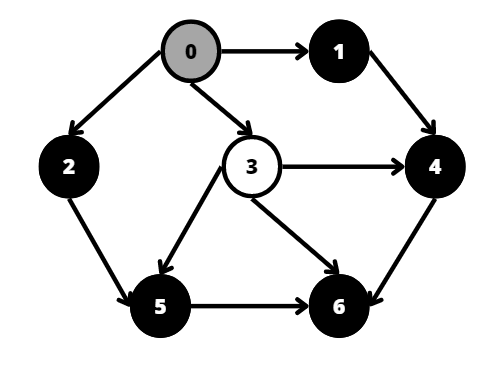
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | Б | С | Ч | Ч | Ч |
| Расстояния | 0 | 1 | 1 | 2 | 3 | 2 | 4 |
| Номера предыдущих вершин |  | 0 | 0 | 1 | 3 | 2 | 4 |
| Номер шага , когда вершина выходит из алгоритма |  | 7 |  |  | 6 | 9 | 5 |



Шаг 10

Стек : 0

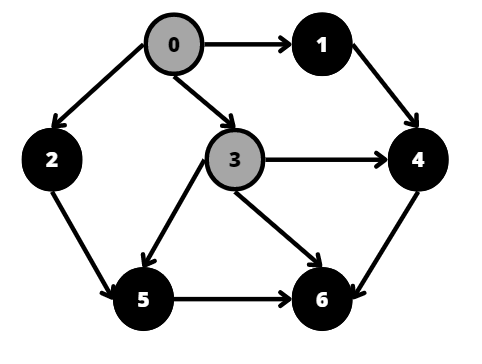
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | Ч | С | Ч | Б | Ч |
| Расстояния | 0 | 1 | 1 | 2 | 3 | 2 | 4 |
| Номера предыдущих вершин |  | 0 | 0 | 1 | 3 | 2 | 4 |
| Номер шага , когда вершина выходит из алгоритма |  | 7 | 10 |  | 6 | 9 | 5 |



Шаг 11

Стек : 0

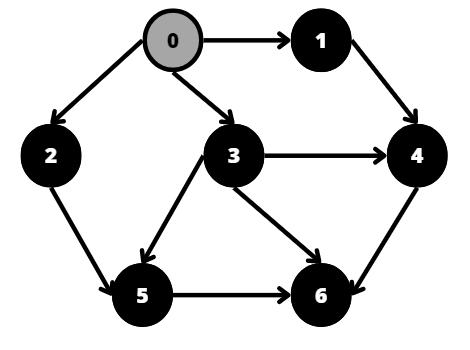
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | Ч | С | Ч | Ч | Ч |
| Расстояния | 0 | 1 | 1 | 2 | 3 | 2 | 4 |
| Номера предыдущих вершин |  | 0 | 0 | 0 | 3 | 2 | 4 |
| Номер шага , когда вершина выходит из алгоритма |  | 7 | 10 |  | 6 | 9 | 5 |



Шаг 12

Стек :

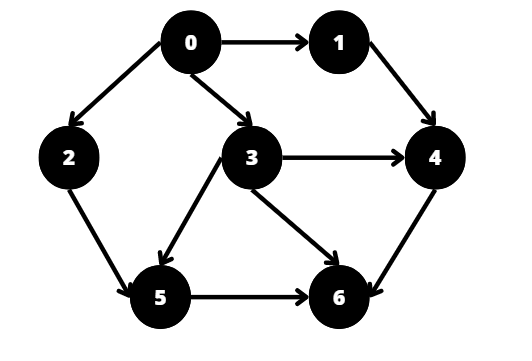
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | Ч | С | Ч | Ч | Ч |
| Расстояния | 0 | 1 | 1 | 2 | 3 | 2 | 4 |
| Номера предыдущих вершин |  | 0 | 0 | 0 | 3 | 2 | 4 |
| Номер шага , когда вершина выходит из алгоритма |  | 7 | 10 | 12 | 6 | 9 | 5 |



Шаг 13

Стек :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Окраска вершин | С | Ч | Ч | С | Ч | Ч | Ч |
| Расстояния | 0 | 1 | 1 | 2 | 3 | 2 | 4 |
| Номера предыдущих вершин |  | 0 | 0 | 0 | 3 | 2 | 4 |
| Номер шага , когда вершина выходит из алгоритма | 13 | 7 | 10 | 12 | 6 | 9 | 5 |



Результат

В основе алгоритма DFS лежит рекурсивная процедура **Visit**, имеющая один входной параметр **k** – вершину графа.

Опишем пошагово процедуру **Visit**.

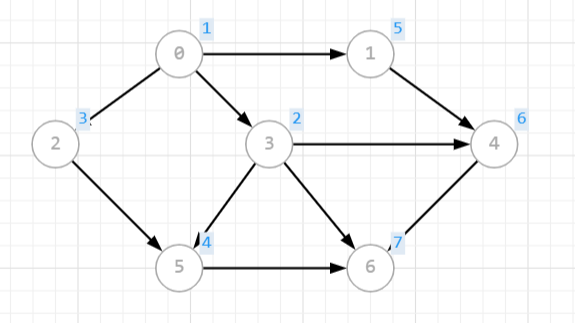
1. Принять параметр **k** – вершину графа.
2. Вершину **k**  окрасить в серый цвет: **C[k] = G**.
3. Увеличить номер шага: **t = t + 1**.
4. Подсчитать расстояние до вершины: **D[k] = t**. Расстояние до вершины в алгоритме DFS совпадает с номером шага, на котором эта вершина была обнаружена (окрашена в серый цвет).
5. Построить множества **J** вершин белого цвета, смежных вершине **k**. Если таких вершин нет, то перейти к шагу 8.
6. Для каждой вершины **j** из множества **J** указать предшествующую вершину: **P[j] = k**.
7. Для каждой вершины **j** из множества **J** выполнить процедуру **Visit**.
8. Вершину **k**  окрасить в черный цвет: **C[k] = B**.
9. Увеличить номер шага: **t = t + 1**.
10. Отметить время фиксации вершины: **F[k] = t**.

**Топологическая сортировка**

**Топологическая сортировка −** это процедура упорядочивания вершин бесконтурного ориентированного графа, не имеющего циклов (ациклического графа). В результате топологической сортировки для вершин графа определяется такой порядок, что если их расположить на рисунке в соответствии с этим порядком сверху вниз, то дуги будут направлены только от верхних вершин к нижним**.**

**Результат**





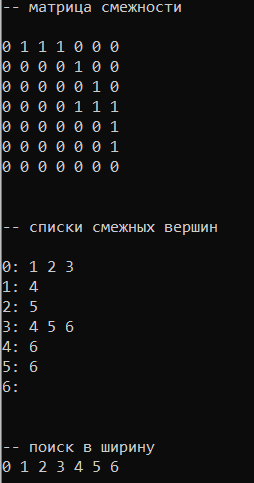
***Задание 3.*** Осуществить программную реализацию алгоритмов на C++. Разработать структуры **AMatrix** и **АList**  для представления ориентированного графа матричным и списковым способом. Разработать функции преобразования из одного способа представления в другой. Разработать функцию **BFS** обхода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

|  |
| --- |
| // ---BFS.h  //  #pragma once  #include "Graph.h"  #include <queue>  struct BFS // breadth-first search поиск в ширину (связный граф)  {  const static int INF = 0x7fffffff;  const static int NIL = -1;  enum Color { WHITE, GRAY, BLACK }; //  const graph::AList\* al; // исходный граф  Color\* c; // цвет вершины  int\* d; // расстояние до вершины  int\* p; // предшествующая вершина  std::queue<int> q; // очередь  BFS(const graph::AList& al, int s);  BFS(const graph::AMatrix& am, int s);  void init(const graph::AList& al, int s);  int get(); // получить следующую вершину  }; |

Листинг 3.1 — файл BFS.h

|  |
| --- |
| // ---BFS.cpp  //  #include "BFS.h"  void BFS::init(const graph::AList& al, int s)  {  this->al = &al;  this->c = new Color[this->al->n\_vertex];  this->d = new int[this->al->n\_vertex];  this->p = new int[this->al->n\_vertex];  for (int i = 0; i < this->al->n\_vertex; i++)  {  this->c[i] = WHITE;  this->d[i] = INF;  this->p[i] = NIL;  };  this->c[s] = GRAY;  this->q.push(s);  };  BFS::BFS(const graph::AList& al, int s) { this->init(al, s); };  BFS::BFS(const graph::AMatrix& am, int s)  {  this->init(\*(new graph::AList(am)), s);  };  int BFS::get()  {  int rc = NIL, v = NIL;  if (!this->q.empty())  {  rc = this->q.front();  for (int j = 0; j < this->al->size(rc); j++)  if (this->c[v = this->al->get(rc, j)] == WHITE)  {  this->c[v] = GRAY;  this->d[v] = this->d[rc] + 1;  this->p[v] = rc;  this->q.push(v);  };  this->q.pop();  this->c[rc] = BLACK;  };  return rc;  } |

Листинг 3.2 — файл BFS.cpp



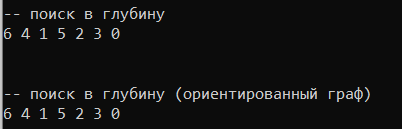
***Задание 4.*** Разработать функцию **DFS**  обхода вершин графа, используя метод поиска глубину. Продемонстрировать работу функции. Копии экрана вставить в отчет.

|  |
| --- |
| // ---DFS.h  //  #pragma once  #include "Graph.h"  #include <vector>  struct DFS // depth-first search поиск в глубину  {  const static int NIL = -1;  enum Color { WHITE, GRAY, BLACK }; //  const graph::AList\* al; // исходный граф  Color\* c; // цвет вершины  int\* d; // время обнаружения  int\* f; // время завершения обработки  int\* p; // предшествующая вершина  int t; // текущее время  DFS(const graph::AList& al);  DFS(const graph::AMatrix& am);  std::vector <int> topological\_sort; //результат топологической сортировки  void visit(int v);  void init(const graph::AList& al);  int get(int i); // получить вершину  }; |

Листинг 4.1 — файл DFS.h

|  |
| --- |
| // ---DFS.cpp  //  #include "DFS.h"  #define NINF 0x80000000  #define INF 0x7fffffff  void DFS::init(const graph::AList& al)  {  this->al = &al;  this->c = new Color[this->al->n\_vertex];  this->d = new int[this->al->n\_vertex];  this->f = new int[this->al->n\_vertex];  this->p = new int[this->al->n\_vertex];  this->t = 0;  for (int i = 0; i < this->al->n\_vertex; i++)  {  this->c[i] = WHITE;  this->d[i] = this->f[i] = 0;  this->p[i] = NIL;  };  for (int i = 0; i < this->al->n\_vertex; i++)  if (this->c[i] == WHITE)  {  this->visit(i);  this->topological\_sort.push\_back(i);  }  };  DFS::DFS(const graph::AList& al) { this->init(al); };  DFS::DFS(const graph::AMatrix& am)  {  this->init(\*(new graph::AList(am)));  };  void DFS::visit(int u)  {  int v = NIL;  this->c[u] = GRAY;  this->d[u] = ++(this->t);  for (int j = 0; j < this->al->size(u); j++)  if (this->c[v = this->al->get(u, j)] == WHITE)  {  this->p[v] = u;  this->visit(v);  this->topological\_sort.push\_back(v);  }  this->c[u] = BLACK;  this->f[u] = ++(this->t);  };  int DFS::get(int i)  {  int j = 0, min1 = INF, min2 = NINF, ntx = NIL;  for (int j = 0; j <= i; j++) // iая статистика  {  for (int k = 0; k < this->al->n\_vertex; k++)  if (this->f[k] < min1 && this->f[k] > min2)  {  min1 = this->f[k]; ntx = k;  };  min2 = min1; min1 = INF;  };  return ntx;  }; |

Листинг 4.2 — файл DFS.cpp



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

|  |
| --- |
| // Функция для топологической сортировки  void topologicalSort(int v, bool visited[], std::stack<int>& Stack, graph::AList& g)  {  // Помечаем текущую вершину как посещенную  visited[v] = true;  // Рекурсивно вызываем функцию для всех смежных вершин  // Если смежная вершина не была посещена, то рекурсивно вызываем функцию  for (int i = 0; i < g.size(v); i++)  if (!visited[g.get(v, i)])  topologicalSort(g.get(v, i), visited, Stack, g);  // Помещаем текущую вершину в стек  Stack.push(v);  }  . . .  std::cout << std::endl << "-- топологическая сортировка" << std::endl;  // Создаем стек для хранения топологической сортировки  std::stack<int> Stack;  // Массив для хранения информации о посещении вершин  bool \*visited = new bool[g5.n\_vertex];  for (int i = 0; i < g5.n\_vertex; i++) visited[i] = false;  // Вызываем функцию для топологической сортировки  for (int i = 0; i < g5.n\_vertex; i++)  if (visited[i] == false)  topologicalSort(i, visited, Stack, g5);  // Выводим топологически отсортированный граф  while (Stack.empty() == false)  {  std::cout << Stack.top() << " ";  Stack.pop();  }  std::cout << std::endl; |

Листинг 5.1— функция топологической сортировки

***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;

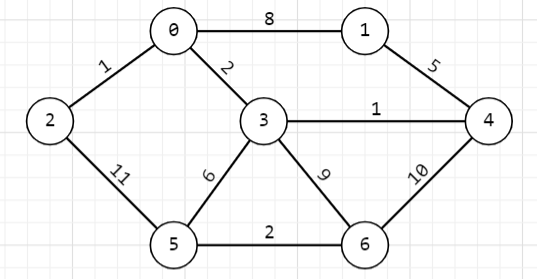
W(e4,3)=4; W(e3,4)=1;

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;

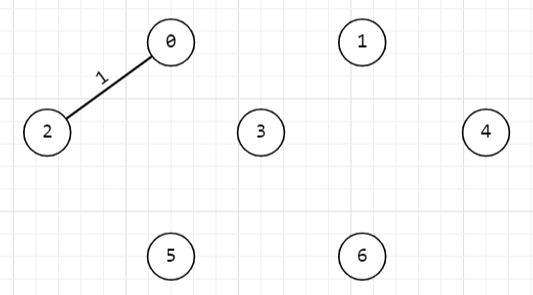


Шаг 1:

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}

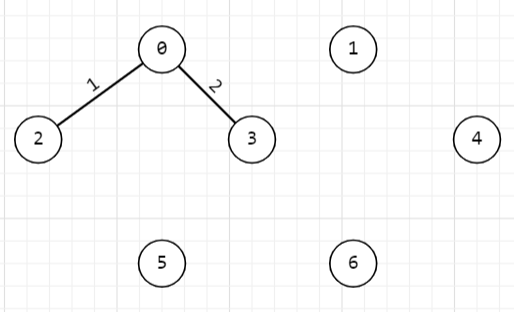
Вершина 0 соединена с вершинами 1, 2 и 3. Веса их ребер равно 8, 1 и 2 соответственно. Из этих ребер минимальной стоимостью обладает ребро 0-2 (вес = 1). Тогда, вершину 2 включаем в множество U, а ребро 0-2 — в множество T.



Шаг 2:

U = {0, 2}

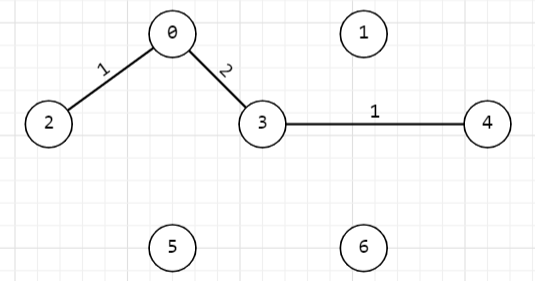
Вершины 0 и 2 соединены с вершинами 1, 3 и 5. Веса их ребер равно 8, 2 и 11 соответственно. Из этих ребер минимальной стоимостью обладает ребро 0-3 (вес = 2). Тогда, вершину 3 включаем в множество U, а ребро 0-3 — в множество T.



Шаг 3:

U = {0, 2, 3}

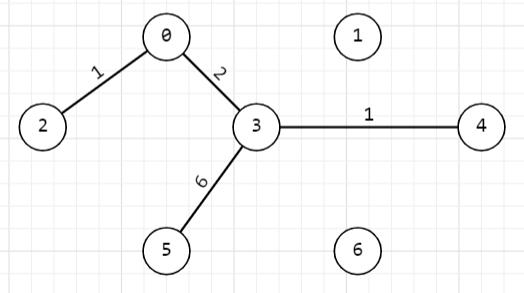
Вершины 0, 2 и 3 соединены с вершинами 1, 5(2-5), 4, 6, 5(3-5). Веса их ребер равно 8, 11, 1, 9 и 6 соответственно. Из этих ребер минимальной стоимостью обладает ребро 3-4 (вес = 1). Тогда, вершину 4 включаем в множество U, а ребро 3-4 — в множество T.



Шаг 4:

U = {0, 2, 3, 4}

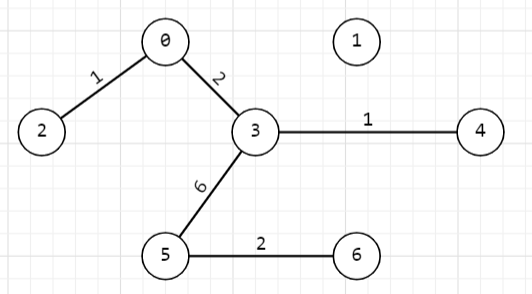
Вершины 0, 2, 3 и 4 соединены с вершинами 1, 5(2-5), 5(3-5), 6(3-6), 6(4-6). Веса их ребер равно 8, 11, 6, 9 и 10 соответственно. Из этих ребер минимальной стоимостью обладает ребро 3-5 (вес = 6). Тогда, вершину 5 включаем в множество U, а ребро 3-5 — в множество T.



Шаг 5:

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

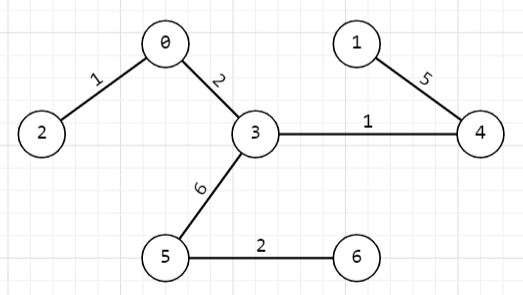
Вершины 0, 2, 3, 4 и 5 соединены с вершинами 1, 6(3-6), 6(4-6), 6(5-6) .Веса их ребер равно 8, 9, 10 и 2 соответственно. Из этих ребер минимальной стоимостью обладает ребро 5-6 (вес = 2). Тогда, вершину 6 включаем в множество U, а ребро 5-6 — в множество T.



Шаг 6:

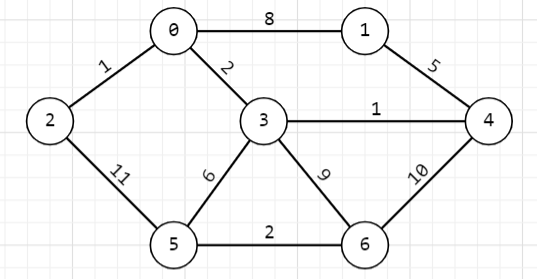
U = {0, 2, 3, 4, 5, 6}

Оставшаяся вершина 1 соединяется с вершиной 0 и 4, их веса равны 8 и 5 соответственно. Тогда, вершину 1 включаем в множество U, а ребро 1-4 — в множество T.



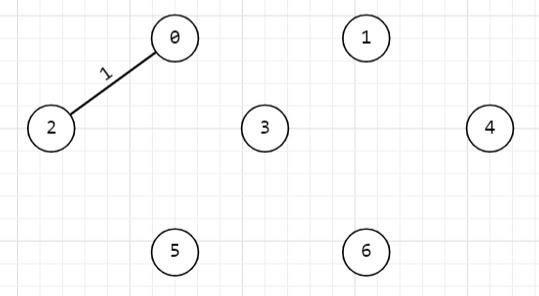
Вес минимального остовного дерева: 17

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

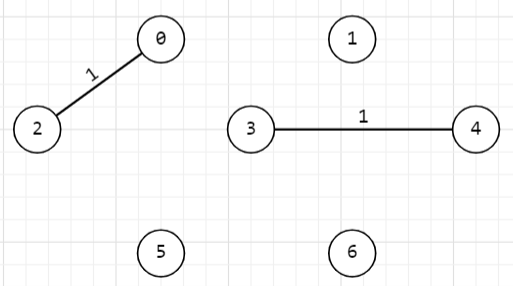


Шаг 1:

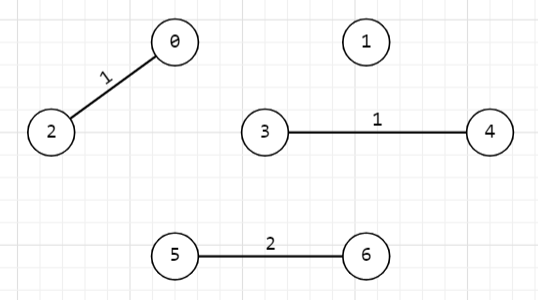
Из всех ребер выбираем ребро с минимальной стоимостью (весом) и включаем это ребро в остовное дерево. Повторяем так далее, однако смотрим на то, чтобы следующее ребро соединяло вершины из разных компонент.



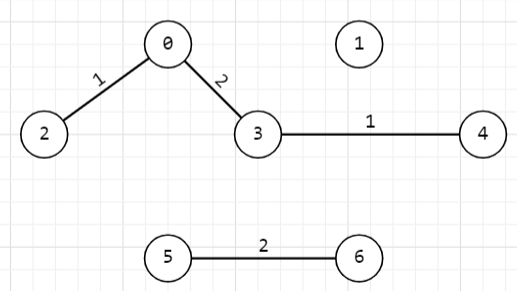
Шаг 2:



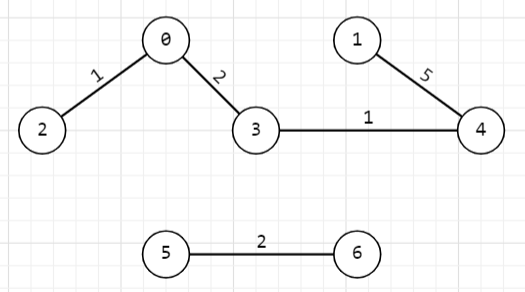
Шаг 3:



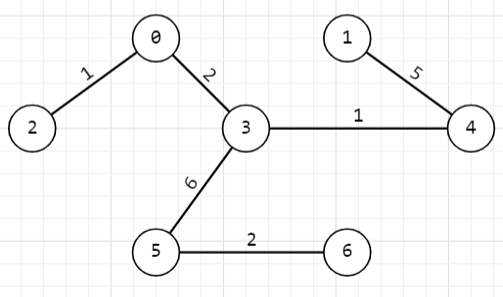
Шаг 4:



Шаг 5:



Шаг 6:



Вес минимального остовного дерева: 17