Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Отчёт по лабораторным работам

По дисциплине «Математическое программирование»

Выполнила:

Студентка 2 курса 10 группы

Рублевская Маргарита Владимировна

Преподаватель: асс. Ромыш А.С.

2025, Минск

**Содержание**

[**Лабораторная работа № 1 3**](#_Toc195204743)

[**Лабораторная работа № 2 7**](#_Toc195204744)

[**Лабораторная работа № 3 19**](#_Toc195204745)

[**Лабораторная работа № 4 23**](#_Toc195204746)

[**Лабораторная работа № 5 30**](#_Toc195204747)

[**Лабораторная работа № 6 41**](#_Toc195204748)

[**Лабораторная работа № 7 55**](#_Toc195204749)

[**Лабораторная работа № 8 57**](#_Toc195204750)

# **Лабораторная работа № 1**

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

**Ход работ**

1. **Написание кода программы (выполнение заданий № 1-2)**

Код файла main.cpp приведен в листинге 1.1

#include "stdafx.h"

#include "Auxil.h" // вспомогательные функции

#include <iostream>

#include <ctime>

#include <locale>

#define CYCLE 1000000 // количество циклов

int Fib(int i)

{

if (i < 1)

return 0;

if (i == 1)

return 1;

return Fib(i - 1) + Fib(i - 2);

}

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

{

double av1 = 0, av2 = 0;

clock\_t t1 = 0, t2 = 0;

setlocale(LC\_ALL, "rus");

auxil::start(); // старт генерации

t1 = clock(); // фиксация времени

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

{

av1 += (double)auxil::iget(-100, 100); // сумма случайных чисел

av2 += auxil::dget(-100, 100); // сумма случайных чисел

}

t2 = clock(); // фиксация времени

std::cout << std::endl << "Количество циклов: " << CYCLE;

std::cout << std::endl << "Среднее значение (int): " << av1 / CYCLE;

std::cout << std::endl << "Среднее значение (double): " << av2 / CYCLE;

std::cout << std::endl << "Продолжительность (у.е): " << (t2 - t1);

std::cout << std::endl << " (сек): " << ((double)(t2 - t1)) / ((double)CLOCKS\_PER\_SEC);

std::cout << std::endl;

for (int n = 22; n < 43; n++)

{

t1 = clock();

int num = Fib(n);

t2 = clock();

std::cout << std::endl << n << "-е число Фибоначчи " << (t2 - t1) << " у.е.";

}

std::cout << std::endl;

system("pause");

return 0;

}

Листинг 1.1. — Содержание файла main.cpp

Код файла Auxil.h приведен в листинге 1.2.

#pragma once

#include <cstdlib>

namespace auxil

{

void start(); // старт генератора сл. чисел

double dget(double rmin, double rmax); // получить случайное число

int iget(int rmin, int rmax); // получить случайное число

};

Листинг 1.2. — Содержание файла main.cpp

Код файла Auxil.cpp приведен в листинге 1.3.

#include "stdafx.h"

#include "Auxil.h"

#include <ctime>

namespace auxil

{

void start() // старт генератора сл. чисел

{

srand((unsigned)time(NULL));

};

double dget(double rmin, double rmax) // получить случайное число

{

return ((double)rand() / (double)RAND\_MAX) \* (rmax - rmin) + rmin;

};

int iget(int rmin, int rmax) // получить случайное число

{

return (int)dget((double)rmin, (double)rmax);

};

}

Листинг 1.3. — Содержание файла main.cpp

Результат работы программы представлен на рисунке 1.1.

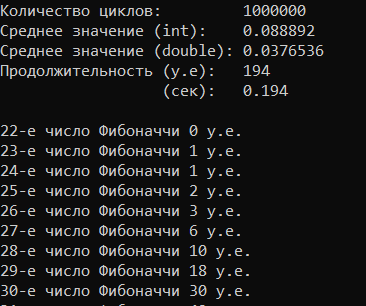
****

Рисунок 1.1 — Результаты работы программы

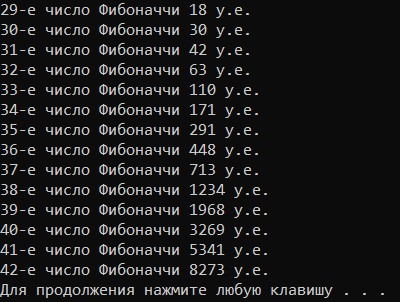
****

Рисунок 1.1 (продолжение) — Результаты работы программы

1. **Выполнение задания № 3**
2. Проведите необходимые эксперименты и постройте график зависимости (Excel) продолжительности процесса вычисления от количества циклов в примере 2.

Сначала найдем зависимость продолжительности вычислений от циклов. Результаты измерений и соответствующий график, представлены на рисунке 1.2.

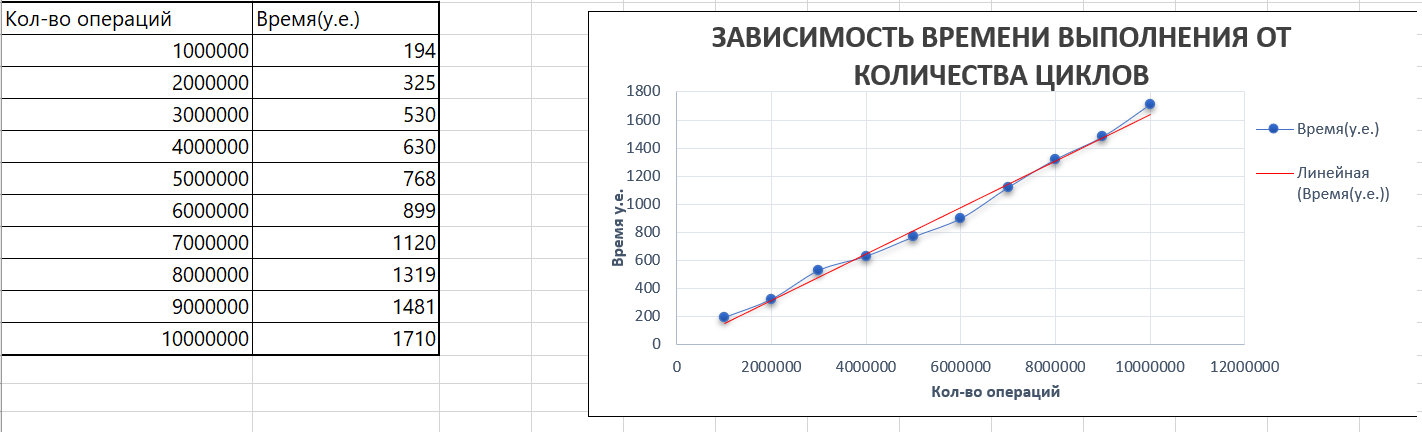
****

Рисунок 1.2 — Результаты измерений

Как мы видим, зависимость у нас линейная, так как кол-во операций прямо пропорционально числу циклов. Вывод: скорость выполнения программы линейно зависит от количества итераций цикла.

1. 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.

Результаты измерений и соответствующий график, представлены на рисунке 1.3.

****

Рисунок 1.3 — Результаты измерений

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

# **Лабораторная работа № 2**

**Вариант 10**

**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.

**Ход работы**

**Написание кода генератора подмножеств заданного множества**

Генератор подмножеств заданного множества – это вывод всех возможных комбинаций элементов множества. Код файла Combi.h приведен в листинге 1.1.

#pragma once

namespace combi

{

struct subset // генератор множества всех подмножеств

{

short n, // количество элементов исходного множества < 64

sn, // количество элементов текущего подмножества

\* sset; // массив индексов текущего подмножества

unsigned \_\_int64 mask; // битовая маска

subset(short n = 1); // конструктор(количество элементов исходного множества)

short getfirst(); // сформормировать массив индексов по битовой маске

short getnext(); // ++маска и сформировать массив индексов

short ntx(short i); // получить i-й элемент массива индексов

unsigned \_\_int64 count(); // вычислить общее количество подмножеств

void reset(); // сбросить генератор, начать сначала

};

};

Листинг 1.1 — Содержание файла Combi.h

Код файла Combi.cpp приведен в листинге 1.2.

#include "Combi.h"

#include <algorithm>

namespace combi

{

subset::subset(short n)

{

this->n = n;

this->sset = new short[n];

this->reset();

};

void subset::reset()

{

this->sn = 0;

this->mask = 0;

};

short subset::getfirst()

{

\_\_int64 buf = this->mask;

this->sn = 0;

for (short i = 0; i < n; i++)

{

if (buf & 0x1) this->sset[this->sn++] = i;

buf >>= 1;

}

return this->sn;

};

short subset::getnext()

{

int rc = -1;

this->sn = 0;

if (++this->mask < this->count()) rc = getfirst();

return rc;

};

short subset::ntx(short i)

{

return this->sset[i];

};

unsigned \_\_int64 subset::count()

{

return (unsigned \_\_int64)(1 << this->n);

};

};

Листинг 1.2 — Содержание файла Combi.cpp

1. **Написание кода генератора сочетаний**

Сочетания выбор элементов из множества, в котором порядок не важен. Код файла Combi.h приведен в листинге 2.1.

#pragma once

namespace combi

{

struct xcombination // генератор сочетаний (эвристика)

{

short n, // количество элементов исходного множества

m, // количество элементов в сочетаниях

\* sset; // массив индексов текущего сочетания

xcombination(

short n = 1, //количество элементов исходного множества

short m = 1 // количество элементов в сочетаниях

);

void reset(); // сбросить генератор, начать сначала

short getfirst(); // сформировать первый массив индексов

short getnext(); // сформировать следующий массив индексов

short ntx(short i); // получить i-й элемент массива индексов

unsigned \_\_int64 nc; // номер сочетания 0,..., count()-1

unsigned \_\_int64 count() const; // вычислить количество сочетаний

};

};

Листинг 2.1 — Содержание файла Combi.h

Код файла Combi.cpp приведен в листинге 2.2.

#include "Combi.h"

#include <algorithm>

namespace combi

{

xcombination::xcombination(short n, short m)

{

this->n = n;

this->m = m;

this->sset = new short[m + 2];

this->reset();

}

void xcombination::reset() // сбросить генератор, начать сначала

{

this->nc = 0;

for (int i = 0; i < this->m; i++) this->sset[i] = i;

this->sset[m] = this->n;

this->sset[m + 1] = 0;

};

short xcombination::getfirst()

{

return (this->n >= this->m) ? this->m : -1;

};

short xcombination::getnext() // сформировать следующий массив индексов

{

short rc = getfirst();

if (rc > 0)

{

short j;

for (j = 0; this->sset[j] + 1 == this->sset[j + 1]; ++j)

this->sset[j] = j;

if (j >= this->m) rc = -1;

else {

this->sset[j]++;

this->nc++;

};

}

return rc;

};

short xcombination::ntx(short i)

{

return this->sset[i];

};

unsigned \_\_int64 fact(unsigned \_\_int64 x) { return(x == 0) ? 1 : (x \* fact(x - 1)); };

unsigned \_\_int64 xcombination::count() const

{

return (this->n >= this->m) ?

fact(this->n) / (fact(this->n - this->m) \* fact(this->m)) : 0;

};

};

Листинг 2.2 — Содержание файла Combi.cpp

1. **Написание кода генератора перестановок**

Код файла Combi.h приведен в листинге 3.1.

#pragma once

namespace combi

{

struct permutation // генератор перестановок

{

const static bool L = true; // левая стрелка

const static bool R = false; // правая стрелка

short n, // количество элементов исходного множества

\* sset; // массив индексов текущей перестановки

bool\* dart; // массив стрелок (левых-L и правых-R)

permutation(short n = 1); // конструктор (количество элементов исходного множества)

void reset(); // сбросить генератор, начать сначала

\_\_int64 getfirst(); // сформировать первый массив индексов

\_\_int64 getnext(); // сформировать случайный массив индексов

short ntx(short i); // получить i-й элемент масива индексов

unsigned \_\_int64 np; // номер перествновки 0,... count()-1

unsigned \_\_int64 count() const; // вычислить общее кол. перестановок

};

};

Листинг 3.1 — Содержание файла Combi.h

Код файла Combi.cpp приведен в листинге 3.2.

#include "Combi.h"

#include <algorithm>

#define NINF ((short)0x8000)

namespace combi

{

permutation::permutation(short n)

{

this->n = n;

this->sset = new short[n];

this->dart = new bool[n];

this->reset();

};

void permutation::reset()

{

this->getfirst();

};

\_\_int64 permutation::getfirst()

{

this->np = 0;

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

{

this->sset[i] = i; this->dart[i] = L;

};

return (this->n > 0) ? this->np : -1;

};

\_\_int64 permutation::getnext()

{

\_\_int64 rc = -1;

short maxm = NINF, idx = -1;

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

{

if (i > 0 &&

this->dart[i] == L &&

this->sset[i] > this->sset[i - 1] &&

maxm < this->sset[i]) maxm = this->sset[idx = i];

if (i < (this->n - 1) &&

this->dart[i] == R &&

this->sset[i] > this->sset[i + 1] &&

maxm < this->sset[i]) maxm = this->sset[idx = i];

};

if (idx >= 0)

{

std::swap(this->sset[idx],

this->sset[idx + (this->dart[idx] == L ? -1 : 1)]);

std::swap(this->dart[idx],

this->dart[idx + (this->dart[idx] == L ? -1 : 1)]);

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

if (this->sset[i] > maxm) this->dart[i] = !this->dart[i];

rc = ++this->np;

}

return rc;

};

short permutation::ntx(short i) { return this->sset[i]; };

unsigned \_\_int64 fact(unsigned \_\_int64 x) { return (x == 0) ? 1 : (x \* fact(x - 1)); };

unsigned \_\_int64 permutation::count() const { return fact(this->n); };

}

Листинг 3.2 — Содержание файла Combi.cpp

1. **Написание кода генератора размещений**

Код файла Combi.h приведен в листинге 4.1.

#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 4.1 — Содержание файла Combi.h

Код файла Combi.cpp приведен в листинге 4.2.

#include "Combi.h"

#include <algorithm>

#define NINF ((short)0x8000)

namespace combi

{

// ================== ACCOMODATION ==================

accomodation::accomodation(short n, short m)

{

this->n = n;

this->m = m;

this->cgen = new xcombination(n, m);

this->pgen = new permutation(m);

this->sset = new short[m];

this->reset();

}

void accomodation::reset()

{

this->na = 0;

this->cgen->reset();

this->pgen->reset();

this->cgen->getfirst();

};

short accomodation::getfirst()

{

short rc = (this->n >= this->m) ? this->m : -1;

if (rc > 0)

{

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

this->sset[i] = this->cgen->sset[this->pgen->ntx(i)];

};

return rc;

};

short accomodation::getnext()

{

short rc;

this->na++;

if ((this->pgen->getnext()) > 0) rc = this->getfirst();

else if ((rc = this->cgen->getnext()) > 0)

{

this->pgen->reset(); rc = this->getfirst();

};

return rc;

};

short accomodation::ntx(short i)

{

return this->sset[i];

};

// факториал из структуры accomodation

unsigned \_\_int64 fact(unsigned \_\_int64 x) { return (x == 0) ? 1 : (x \* fact(x - 1)); };

unsigned \_\_int64 accomodation::count() const

{

return (this->n >= this->m) ?

fact(this->n) / fact(this->n - this->m) : 0;

};

// ================ PERMUTATION ======================

permutation::permutation(short n)

{

this->n = n;

this->sset = new short[n];

this->dart = new bool[n];

this->reset();

};

void permutation::reset()

{

this->getfirst();

};

\_\_int64 permutation::getfirst()

{

this->np = 0;

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

{

this->sset[i] = i; this->dart[i] = L;

};

return (this->n > 0) ? this->np : -1;

};

\_\_int64 permutation::getnext()

{

\_\_int64 rc = -1;

short maxm = NINF, idx = -1;

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

{

if (i > 0 &&

this->dart[i] == L &&

this->sset[i] > this->sset[i - 1] &&

maxm < this->sset[i]) maxm = this->sset[idx = i];

if (i < (this->n - 1) &&

this->dart[i] == R &&

this->sset[i] > this->sset[i + 1] &&

maxm < this->sset[i]) maxm = this->sset[idx = i];

};

if (idx >= 0)

{

std::swap(this->sset[idx],

this->sset[idx + (this->dart[idx] == L ? -1 : 1)]);

std::swap(this->dart[idx],

this->dart[idx + (this->dart[idx] == L ? -1 : 1)]);

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

if (this->sset[i] > maxm) this->dart[i] = !this->dart[i];

rc = ++this->np;

}

return rc;

};

short permutation::ntx(short i) { return this->sset[i]; };

unsigned \_\_int64 permutation::count() const { return fact(this->n); };

// =============== XCOMBINATION ==================

xcombination::xcombination(short n, short m)

{

this->n = n;

this->m = m;

this->sset = new short[m + 2];

this->reset();

}

void xcombination::reset()

{

this->nc = 0;

for (int i = 0; i < this->m; i++) this->sset[i] = i;

this->sset[m] = this->n;

this->sset[m + 1] = 0;

};

short xcombination::getfirst()

{

return (this->n >= this->m) ? this->m : -1;

};

short xcombination::getnext()

{

short rc = getfirst();

if (rc > 0)

{

short j;

for (j = 0; this->sset[j] + 1 == this->sset[j + 1]; ++j)

this->sset[j] = j;

if (j >= this->m) rc = -1;

else

{

this->sset[j]++;

this->nc++;

}

}

return rc;

};

short xcombination::ntx(short i)

{

return this->sset[i];

};

unsigned \_\_int64 xcombination::count() const

{

return (this->n >= this->m) ?

fact(this->n) / (fact(this->n - this->m) \* fact(this->m)) : 0;

};

}

Листинг 4.2 — Содержание файла Combi.cpp

1. **Решение задачи упрощенную о рюкзаке и исследование зависимости времени вычисления необходимого для решения задачи от размерности задачи**

Код файла Combi.h приведен в листинге 5.1.

#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 5.1 — Содержание файла Combi.h

Код файла Combi.cpp приведен в листинге 5.2.

#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 5.2 — Содержание файла Combi.cpp

Код файла Knapsack.h приведен в листинге 5.3.

#pragma once

#include "Combi.h"

int knapsack\_s(

int V, // [in] вместимость рюкзака

short n, // [in] количество типов предметов

const int v[], // [in] размер предмета каждого типа

const int c[], // [in] стоимость предмета каждого типа

short m[] // [out] количество предметов каждого типа

);

Листинг 5.3 — Содержание файла Knapsack.h

Код файла Knapsack.cpp приведен в листинге 5.4.

#include "Knapsack.h"

#define NINF 0x80000000 // самое малое int-число

int calcv(combi::subset s, const int v[]) // объем в рюкзаке

{

int rc = 0;

for (int i = 0; i < s.sn; i++) rc += v[s.ntx(i)];

return rc;

};

int calcc(combi::subset s, const int v[], const int c[]) //стоимость в рюкзаке

{

int rc = 0;

for (int i = 0; i < s.sn; i++) rc += (v[s.ntx(i)] \* c[s.ntx(i)]);

return rc;

};

void setm(combi::subset s, short m[]) //отметить выбранные предметы

{

for (int i = 0; i < s.n; i++) m[i] = 0;

for (int i = 0; i < s.sn; i++) m[s.ntx(i)] = 1;

};

int knapsack\_s(

int V, // [in] вместимость рюкзака

short n, // [in] количество типов предметов

const int v[], // [in] размер предмета каждого типа

const int c[], // [in] стоимость предмета каждого типа

short m[] // [out] количество предметов каждого типа {0,1}

)

{

combi::subset s(n);

int maxc = NINF, cc = 0;

short ns = s.getfirst();

while (ns >= 0)

{

if (calcv(s, v) <= V)

if ((cc = calcc(s, v, c)) > maxc)

{

maxc = cc;

setm(s, m);

}

ns = s.getnext();

};

return maxc;

};

Листинг 5.4 — Содержание файла Knapsack.cpp

Код файла main.cpp приведен в листинге 5.5.

#include <iostream>

#include "Combi.h"

#include "Knapsack.h"

#include <ctime>

#define NN 18

int main()

{

setlocale(LC\_ALL, "rus");

int V = 300, // вместимость рюкзака

v[] = { 25, 30, 60, 20, 21, 14, 36, 85, 45, 35, 74, 58, 67, 25, 19, 46, 36, 89}, // размер предмета каждого типа

c[] = { 25, 10, 20, 30, 5, 15, 55, 36, 47, 50, 32, 52, 12, 11, 30, 40, 7, 21}; // стоимость предмета каждого типа

short m[NN]; // количество предметов каждого типа {0,1}

clock\_t t1 = 0, t2 = 0;

t1 = clock();

int maxcc = knapsack\_s(

V, // [in] вместимость рюкзака

NN, // [in] количество типов предметов

v, // [in] размер предмета каждого типа

c, // [in] стоимость предмета каждого типа

m // [out] количество предметов каждого типа

);

t2 = clock();

std::cout << "-------- Задача о рюкзаке --------- " << std::endl;

std::cout << std::endl << "- количество предметов : " << NN;

std::cout << std::endl << "- вместимость рюкзака : " << V;

std::cout << std::endl << "- размеры предметов : ";

for (int i = 0; i < NN; i++) std::cout << v[i] << " ";

std::cout << std::endl << "- стоимости предметов : ";

for (int i = 0; i < NN; i++) std::cout << v[i] \* c[i] << " ";

std::cout << std::endl << "- оптимальная стоимость рюкзака: " << maxcc;

std::cout << std::endl << "- вес рюкзака: ";

int s = 0; for (int i = 0; i < NN; i++) s += m[i] \* v[i];

std::cout << s;

std::cout << std::endl << "- выбраны предметы: ";

for (int i = 0; i < NN; i++) std::cout << " " << m[i];

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

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 5.5 — Содержание файла main.cpp

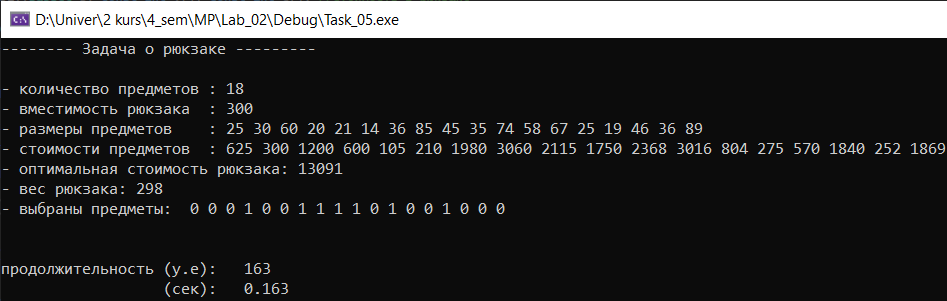
Результат работы программы представлен на рисунке 5.1. 

Рисунок 5.1. Результат работы программы

Исследование зависимости представлено на рисунке 5.2.



Рисунок 5.2. Исследование зависимости

# **Лабораторная работа № 3**

**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**

Условие задачи коммивояжёра:

**Задание №2**

Приведённая по строкам матрица:

α = 10+10+12+27+23=82;

Полностью приведённая матрица:

β = 10+13=23;

Нижняя граница длины кольцевого маршрута: φ = 82 + 23 = 105.

01,2 = 0+8=8; 01,5 = 0+3=3; 02,1 = 0+2=2;

03,1 = 0+8=8; 04,1 = 0+0=0; 04,3 = 0+2=2;

05,4 = 48+10=58.

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

163

105

В случае если мы идём по маршруту (5, 4) то расстояние будет равно нижней границе кольцевого маршрута, то есть 105, а если не пойдём, то расстояние будет равно 105 + 58 = 163.

Так как меньшее расстояние 105, то мы идём из города 5 в город 4. Следовательно, для дальнейших вычислений вычёркиваем 5 строку и 4 столбец из матрицы и делаем обратный путь (4, 5) равным INF:

1 2 3 5

1

2

3

4

Полученная матрица уже является полностью приведённой, поэтому нижняя граница кольцевого маршрута останется неизменной и равной 105.

01,2 = 0+8=8, 01,5 = 0+47=47, 02,1 = 0+2=2,

03,1 = 0+8=8, 04,1 = 0+0=0, 04,3 =0+2=2.

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

105

152

В случае если мы идём по маршруту (1, 5) то расстояние будет равно нижней границе кольцевого маршрута, то есть 105, а если не пойдём, то расстояние будет равно 105 + 47 = 152.

Так как меньшее расстояние 105, то мы идём из города 1 в город 5. Следовательно, для дальнейших вычислений вычёркиваем 1 строку и 5 столбец из матрицы:

1 2 3

2  
3  
4

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

β = 0 + 8 + 0 = 8;

Следовательно, изменится нижняя граница кольцевого маршрута и соответственно: φ = 105 + 8 = 113.

02,1 = 2, 03,1 = 0, 03,2 = 3;

04,1 = 0, 04,3 = 2;

Выбираем максимальное значение и получаем путь из города 3 в 2 и соответственно получим граф:

113

116

В случае если мы идём по маршруту (3, 2) то расстояние будет равно нижней границе кольцевого маршрута, то есть 113, а если не пойдём, то расстояние будет равно 113 + 3 = 116.

Так как меньшее расстояние 113, то мы идём из города 3 в город 2. Следовательно, для дальнейших вычислений вычёркиваем 3 строку и 2 столбец из матрицы:

После анализа данной матрицы к нашему графу добавятся пути (2, 1) и (4,3). Граф, представлен на рисунке 1.

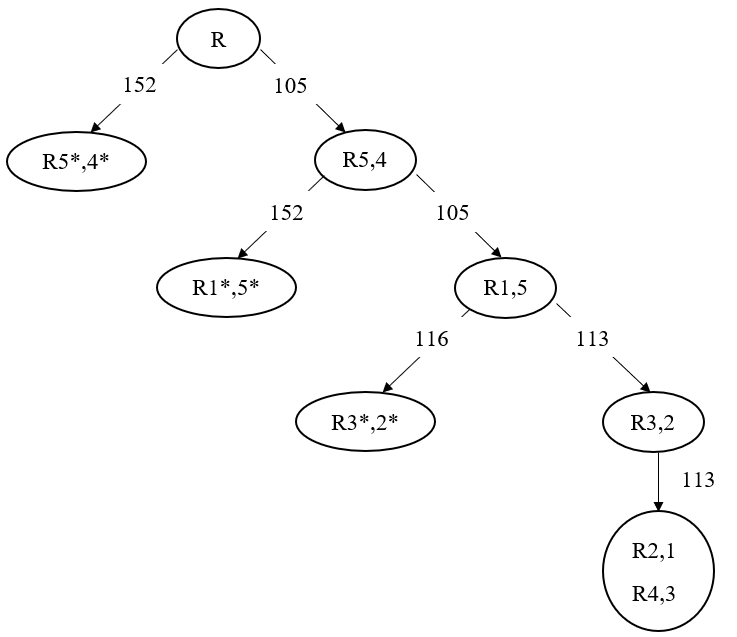


Рисунок 1 — Результат работы программы

Расставим переходы между городами в правильной последовательности и соответственно получим (1, 5), (5, 4), (4, 3), (3, 2), (2, 1). Итого длина маршрута будет равна 113.

**Задание №3**

Результат работы программы представлен на рисунке 5.1.

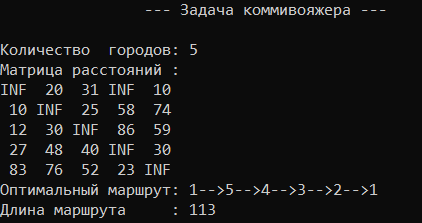


Рисунок 2 — Результат работы программы

**Вывод**: в ходе лабораторной работы были освоены принципы решения задач методом ветвей и границ, решена задача о коммивояжере данным методом, а также подтверждена правильность решения задачи с помощью метода полного перебора.

# **Лабораторная работа № 4**

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

**Задание № 1**

На языке С++ сгенерировать случайным образом строку букв латинского алфавита *S1* длиной 300 символов и *S2*длиной 200. Код приведен в листинге 1.1.

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

void main()

{

srand(time(NULL));

char abc[25];

char s1[300], s2[250];

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

{

abc[n] = (char)i;

}

std::cout << "S1 = ";

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

{

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

if (i % 100 == 0)

std::cout << std::endl;

std::cout << s1[i];

}

std::cout << std::endl << std::endl << "S2 =";

for (int i = 0; i < 251; i++)

{

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

if (i % 100 == 0)

std::cout << std::endl;

std::cout << s2[i];

}

std::cout << std::endl;

}

Листинг 1.1. – Генерация строк

Результат выполнения программы представлен на рисунке 1.1.

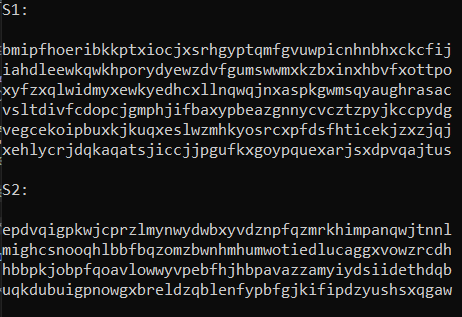
****

Рисунок 1.1. – Результат работы программы

**Задание № 2**

Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – Левенштейна для , где - длина строки ,  - строка, состоящая из первых  символов строки . (копии экрана и код вставить в отчет). Код приведен в листинге 1.2.

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

{

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

}

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

{

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

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

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

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

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

{

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

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

}

return DD(lx, ly);

}

int levenshtein\_r( int lx, const char x[],

int ly, const char y[] )

{

int rc = 0;

if (lx == 0) rc = ly;

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

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

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

else rc = min3(

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

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

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

);

return rc;

};

Листинг 1.2. – Вычисление расстояния Левенштейна

Результат выполнения программы представлен на рисунке 1.2.

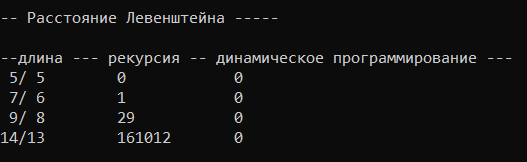


Рисунок 1.2. – Результат работы программы

**Задание № 3**

Выполнить сравнительный анализ времени, затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . На рисунке 1.3 представлены графики зависимости времени вычисления от *k*.

****

Рисунок 1.3. – Графики зависимости

**Задание № 4**

Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом).

|  |  |
| --- | --- |
| Задание 4 | |
| Ель | Дрель |

1.  
2.  
3.  
4.  
5.  

 = 5.

 = 4.

1.  

 = 4.

 = 3.

1.  
2.  
3.  

 = 3.

 = 2.

1.  
2.  
3.  

 = 2.

 = 1.

1.  

 = 3.

 = 2.

1.  

 = 2.

 = 1.

1.  

 = 1.

 = 1.

 = 0.

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

Дистанция Левенштейна для «ель» и «дрель» равно 2.

**Задание № 5**

Выполнить сравнительный анализ времени, затраченного на решение задачи об оптимальной расстановке скобок при умножении нескольких матриц для двух методов решения (рекурсивное решение, динамическое программирование). Код приведен в листинге 1.3.

#include "stdafx.h"

#include <memory.h>

#include "MultyMatrix.h"

// расстановка скобок (рекурсия)

#define INFINITY 0x7fffffff

#define NINFINITY 0x80000000

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

{

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

int o = INFINITY; int bo = INFINITY;

if (i < j)

{

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

{

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

if (bo < o)

{

o = bo; OPTIMALM\_S(i, j) = k;

}

}

}

else o = 0; return o;

#undef OPTIMALM\_S

};

// расстановка скобок (динамическое программирование)

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

{

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

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

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

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

OPTIMALM\_M(i, i) = 0;

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

{

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

{

j = i + l - 1;

OPTIMALM\_M(i, j) = INFINITY;

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

{

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

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

{

OPTIMALM\_M(i, j) = q; OPTIMALM\_S(i, j) = k;

}

}

}

}

return OPTIMALM\_M(1, n);

#undef OPTIMALM\_M

#undef OPTIMALM\_S

};

Листинг 1.3. – Решение задачи о расстановке скобок

Результат выполнения программы представлен на рисунке 1.4.

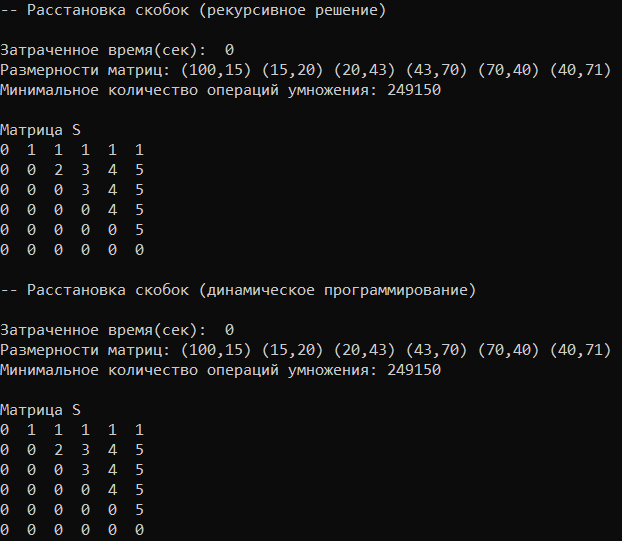
****

Рисунок 1.4. – Результат работы программы

Принцип расстановки скобок по итоговой матрице:

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

А1=100\*15,

А2=15\*20,

А3=20\*43,

А4 =43\*70,

А5 =70\*40,

А6 =40\*71.

Матрица S:

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

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

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

Точку разрыва между второй и шестой матрицей определяет элемент (2,6). Он равен 5. Следовательно разрыв будет после 5-ой матрицы.

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

Далее берем элемент (2,5) и получаем, что он равен 4. Следовательно получаем:

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

И на последнем шаге мы возьмем элемент (2,4) и он равен 3:

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

Это выражение и есть конечное.

Полученная расстановка скобок позволяет получить минимальное количество операций умножения, равное 249150.

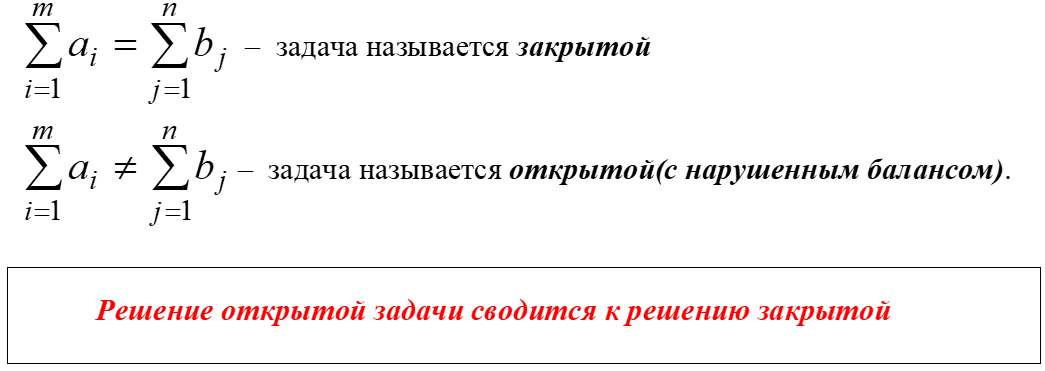
**Вывод:** в результате выполнения лабораторной работы были освоены общие принципы решения задач методом динамического программирования. Были изучены его основные этапы и принципы работы алгоритмов. Были рассмотрены примеры решения задач методом динамического программирования и сравнены с рекурсивным методом.

# **Лабораторная работа № 5**

**Задание 1:** решить транспортную задачу. Имеется 5 поставщиков продукции и 6 потребителей.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13** | **21** | **11** | **178** |
| 2 | **20** | **10** | **18** | **15** | **17** | **23** | **123** |
| 3 | **11** | **15** | **21** | **18** | **12** | **21** | **160** |
| 4 | **14** | **20** | **20** | **13** | **23** | **12** | **169** |
| 5 | **13** | **21** | **19** | **10** | **20** | **14** | **110** |
| ПОТРЕБНОСТИ | **153** | **117** | **141** | **203** | **105** | **173** |  |

Производим проверку является ли задача закрытой.



Из результатов видно, что потребность превышает количество на складах, т.е. задача является открытой.

Чтобы получить закрытую модель, введем дополнительный пункт отправления 6 c запасами 892-740 = 152. Тарифы перевозки из пункта отправления 6 во все пункты назначения полагаем равными нулю. В результате получим закрытую модель транспортной задачи:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13** | **21** | **11** | **178** |
| 2 | **20** | **10** | **18** | **15** | **17** | **23** | **123** |
| 3 | **11** | **15** | **21** | **18** | **12** | **21** | **160** |
| 4 | **14** | **20** | **20** | **13** | **23** | **12** | **169** |
| 5 | **13** | **21** | **19** | **10** | **20** | **14** | **110** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **153** | **117** | **141** | **203** | **105** | **173** |  |

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.

Выбор наименьшей ячейки С22= 10

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13** | **21** | **11** | **178** |
| 2 | **20** | **10** | **18** | **15** | **17** | **23** | **123** |
| 3 | **11** | **15** | **21** | **18** | **12** | **21** | **160** |
| 4 | **14** | **20** | **20** | **13** | **23** | **12** | **169** |
| 5 | **13** | **21** | **19** | **10** | **20** | **14** | **110** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **153** | **117** | **141** | **203** | **105** | **173** |  |

x22=min(117,123)=117

Выбор наименьшей ячейки С54= 10.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13** | **21** | **11** | **178** |
| 2 | **20** | **10/117** | **18** | **15** | **17** | **23** | **6** |
| 3 | **11** | **15** | **21** | **18** | **12** | **21** | **160** |
| 4 | **14** | **20** | **20** | **13** | **23** | **12** | **169** |
| 5 | **13** | **21** | **19** | **10** | **20** | **14** | **110** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **153** | **0** | **141** | **203** | **105** | **173** |  |

x54=min(203,110)=110

Выбор наименьшей ячейки С31= 11

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13** | **21** | **11** | **178** |
| 2 | **20** | **10/117** | **18** | **15** | **17** | **23** | **6** |
| 3 | **11** | **15** | **21** | **18** | **12** | **21** | **160** |
| 4 | **14** | **20** | **20** | **13** | **23** | **12** | **169** |
| 5 | **13** | **21** | **19** | **10/110** | **20** | **14** | **0** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **153** | **0** | **141** | **93** | **105** | **173** |  |

x31=min(153,160)=153;

Выбор наименьшей ячейки С16= 11

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13** | **21** | **11** | **178** |
| 2 | **20** | **10/117** | **18** | **15** | **17** | **23** | **6** |
| 3 | **11/153** | **15** | **21** | **18** | **12** | **21** | **7** |
| 4 | **14** | **20** | **20** | **13** | **23** | **12** | **169** |
| 5 | **13** | **21** | **19** | **10/110** | **20** | **14** | **0** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **0** | **0** | **141** | **93** | **105** | **173** |  |

x16=min(173,178)=173;

Выбор наименьшей ячейки С35= 12

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13** | **21** | **11/173** | **5** |
| 2 | **20** | **10/117** | **18** | **15** | **17** | **23** | **6** |
| 3 | **11/153** | **15** | **21** | **18** | **12** | **21** | **7** |
| 4 | **14** | **20** | **20** | **13** | **23** | **12** | **169** |
| 5 | **13** | **21** | **19** | **10/110** | **20** | **14** | **0** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **0** | **0** | **141** | **93** | **105** | **0** |  |

x35=min(105,7)=7;

Выбор наименьшей ячейки С14= 13

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13** | **21** | **11/173** | **5** |
| 2 | **20** | **10/117** | **18** | **15** | **17** | **23** | **6** |
| 3 | **11/153** | **15** | **21** | **18** | **12/7** | **21** | **0** |
| 4 | **14** | **20** | **20** | **13** | **23** | **12** | **169** |
| 5 | **13** | **21** | **19** | **10/110** | **20** | **14** | **0** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **0** | **0** | **141** | **93** | **98** | **0** |  |

x14=min(93,5)=5;

Выбор наименьшей ячейки С44= 13

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13/5** | **21** | **11/173** | **0** |
| 2 | **20** | **10/117** | **18** | **15** | **17** | **23** | **6** |
| 3 | **11/153** | **15** | **21** | **18** | **12/7** | **21** | **0** |
| 4 | **14** | **20** | **20** | **13** | **23** | **12** | **169** |
| 5 | **13** | **21** | **19** | **10/110** | **20** | **14** | **0** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **0** | **0** | **141** | **88** | **98** | **0** |  |

X44=min(88,169)=88;

Выбор наименьшей ячейки С24= 17

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13/5** | **21** | **11/173** | **0** |
| 2 | **20** | **10/117** | **18** | **15** | **17** | **23** | **6** |
| 3 | **11/153** | **15** | **21** | **18** | **12/7** | **21** | **0** |
| 4 | **14** | **20** | **20** | **13/88** | **23** | **12** | **81** |
| 5 | **13** | **21** | **19** | **10/110** | **20** | **14** | **0** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **0** | **0** | **141** | **0** | **98** | **0** |  |

x24=min(6,98)=6;

Выбор наименьшей ячейки С43= 20

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13/5** | **21** | **11/173** | **0** |
| 2 | **20** | **10/117** | **18** | **15** | **17/6** | **23** | **0** |
| 3 | **11/153** | **15** | **21** | **18** | **12/7** | **21** | **0** |
| 4 | **14** | **20** | **20** | **13/88** | **23** | **12** | **81** |
| 5 | **13** | **21** | **19** | **10/110** | **20** | **14** | **0** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **0** | **0** | **141** | **0** | **92** | **0** |  |

x43=min(81,141)=81;

Выбор наименьшей ячейки С63= 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13/5** | **21** | **11/173** | **0** |
| 2 | **20** | **10/117** | **18** | **15** | **17/6** | **23** | **0** |
| 3 | **11/153** | **15** | **21** | **18** | **12/7** | **21** | **0** |
| 4 | **14** | **20** | **20/81** | **13/88** | **23** | **12** | **0** |
| 5 | **13** | **21** | **19** | **10/110** | **20** | **14** | **0** |
| 6 | **0** | **0** | **0** | **0** | **0** | **0** | **152** |
| ПОТРЕБНОСТИ | **0** | **0** | **60** | **0** | **92** | **0** |  |

X63=min(60,154)=60;

Выбор наименьшей ячейки С65= 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13/5** | **21** | **11/173** | **0** |
| 2 | **20** | **10/117** | **18** | **15** | **17/6** | **23** | **0** |
| 3 | **11/153** | **15** | **21** | **18** | **12/7** | **21** | **0** |
| 4 | **14** | **20** | **20/81** | **13/88** | **23** | **12** | **0** |
| 5 | **13** | **21** | **19** | **10/110** | **20** | **14** | **0** |
| 6 | **0** | **0** | **0/60** | **0** | **0** | **0** | **92** |
| ПОТРЕБНОСТИ | **0** | **0** | **0** | **0** | **92** | **0** |  |

x65=min(92,92)=92;

Количество базисных переменных должно быть m+n-1= 6+6-1= 11. В таблице их также 11, следовательно план является невырожденным.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | **22** | **12** | **16** | **13/5** | **21** | **11/173** | **0** |
| 2 | **20** | **10/117** | **18** | **15** | **17/6** | **23** | **0** |
| 3 | **11/153** | **15** | **21** | **18** | **12/7** | **21** | **0** |
| 4 | **14** | **20** | **20/81** | **13/88** | **23** | **12** | **0** |
| 5 | **13** | **21** | **19** | **10/110** | **20** | **14** | **0** |
| 6 | **0** | **0** | **0/60** | **0** | **0/92** | **0** | **0** |
| ПОТРЕБНОСТИ | **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.

Первое допустимое решение

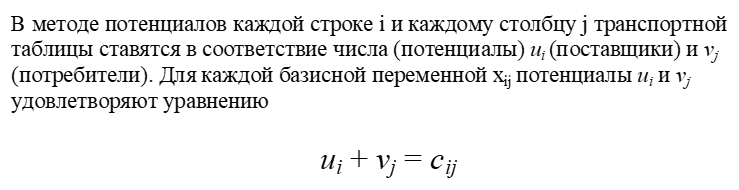
x31=153, x22=117, x43=81, x63=60, x14=5, x44=88, x54=110, x25=6, x35=7, x65=92, x16=173;

Значение функции цели тогда:

Z= 13\*5 + 11\*173 + 10\*117 + 17\*6 + 11\*153 + 12\*7 + 20\*81 + 13\*88 + 10\*110 + 0\*60 + 0\*92 = 8871

**Этап II**

***Метод потенциалов***

**

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

u1 + v4 = 13; 0 + v4 = 13; v4 = 13

u4 + v4 = 13; 13 + u4 = 13; u4 = 0

u4 + v3 = 20; 0 + v3 = 20; v3 = 20

u6 + v3 = 0; 20 + u6 = 0; u6 = -20

u6 + v5 = 0; -20 + v5 = 0; v5 = 20

u2 + v5 = 17; 20 + u2 = 17; u2 = -3

u2 + v2 = 10; -3 + v2 = 10; v2 = 13

u3 + v5 = 12; 20 + u3 = 12; u3 = -8

u3 + v1 = 11; -8 + v1 = 11; v1 = 19

u5 + v4 = 10; 13 + u5 = 10; u5 = -3

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

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | v1=19 | v2=13 | v3=20 | v4=13 | v5=20 | v6=11 | ЗАПАСЫ |
| u1=0 | **22** | **12** | **16** | **13[5]** | **21** | **11[173]** | **0** |
| u2=-3 | **20** | **10[117]** | **18** | **15** | **17[6]** | **23** | **0** |
| u3=-8 | **11[153]** | **15** | **21** | **18** | **12[7]** | **21** | **0** |
| u4=0 | **14** | **20** | **20[81]** | **13[88]** | **23** | **12** | **0** |
| u5=-3 | **13** | **21** | **19** | **10[110]** | **20** | **14** | **0** |
| u6=-20 | **0** | **0** | **0[60]** | **0** | **0[92]** | **0** | **0** |
| ПОТРЕБНОСТИ | **0** | **0** | **0** | **0** | **0** | **0** |  |

Для свободных клеток xij = ui + vj - cij

|  |  |
| --- | --- |
| Небазисная переменная |  |
| x11 | **0+19-22=-3** |
| x21 | **(-3)+19-20=-4** |
| x41 | **0+19-14=5** |
| x51 | **(-3)+19-13=3** |
| x61 | **-20+19-0=-1** |
| x12 | **0+13-12=1** |
| x32 | **(-8)+13-15=-10** |
| x42 | **0+13-20=-7** |
| x52 | **(-3)+13-21=-11** |
| x62 | **-20+13+0=-7** |
| x13 | **0+20-16=4** |
| x23 | **(-3)+20-18=-1** |
| x33 | **(-8)+20-21=-9** |
| x53 | **(-3)+20-19=-2** |
| x24 | **(-3)+13-15=-5** |
| x34 | **(-8)+13-18=-13** |
| x15 | **0+20-21=-1** |
| x45 | **0+20-23=-3** |
| x55 | **(-3)+20-20=-3** |
| x26 | **(-3)+11-23=-15** |
| x36 | **(-8)+11-21=-18** |
| x46 | **0+11-12=-1** |
| x56 | **(-3)+11-14=-6** |

Вводимой в базис будет переменная, имеющая наибольшее положительное значение -x41.

Определив вводимую в базис переменную, следует определить исключаемую из базиса переменную. Обозначим через θ количество груза, перевозимого по маршруту (4,1). Максимально возможное значение θ определяем из следующих условий:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 22 | 12 | 16 | 13[5] | 21 | 11[173] | 178 |
| 2 | 20 | 10[117] | 18 | 15 | 17[6] | 23 | 123 |
| 3 | 11[153]  [-] | 15 | 21 | 18 | 12[7]  [+] | 21 | 160 |
| 4 | 14[+] | 20 | 20[81]  [-] | 13[88] | 23 | 12 | 169 |
| 5 | 13 | 21 | 19 | 10[110] | 20 | 14 | 110 |
| 6 | 0 | 0 | 0[60]  [+] | 0 | 0[92]  [-] | 0 | 152 |
| ПОТРЕБНОСТИ | 153 | 117 | 141 | 203 | 105 | 173 |  |

Перемещаем товары по циклу 81 единиц (минимальный в минусовом клетке (4,3)) цикл (4,1) →(4,3) →(6,3) →(6,5) →(3,5) →(3,1)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 22 | 12 | 16 | 13[5] | 21 | 11[173] | 178 |
| 2 | 20 | 10[117] | 18 | 15 | 17[6] | 23 | 123 |
| 3 | 11[72] | 15 | 21 | 18 | 12[88] | 21 | 160 |
| 4 | 14[81] | 20 | 20 | 13[88] | 23 | 12 | 169 |
| 5 | 13 | 21 | 19 | 10[110] | 20 | 14 | 110 |
| 6 | 0 | 0 | 0[141] | 0 | 0[11] | 0 | 152 |
| ПОТРЕБНОСТИ | 153 | 117 | 141 | 203 | 105 | 173 |  |
|  | B1 | B2 | B3 | B4 | B5 | B6 | ЗАПАСЫ |

Повторим вычисление потенциалов.

u1 = 0.

u1 + v4 = 13; 0 + v4 = 13; v4 = 13

u4 + v4 = 13; 13 + u4 = 13; u4 = 0

u4 + v1 = 14; 0 + v1 = 14; v1 = 14

u3 + v1 = 11; 14 + u3 = 11; u3 = -3

u3 + v5 = 12; -3 + v5 = 12; v5 = 15

u2 + v5 = 17; 15 + u2 = 17; u2 = 2

u2 + v2 = 10; 2 + v2 = 10; v2 = 8

u6 + v5 = 0; 15 + u6 = 0; u6 = -15

u6 + v3 = 0; -15 + v3 = 0; v3 = 15

u5 + v4 = 10; 13 + u5 = 10; u5 = -3

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | v1=14 | v2=8 | v3=15 | v4=13 | v5=15 | v6=11 |
| u1=0 | 22 | 12 | 16 | 13[5] | 21 | 11[173] |
| u2=2 | 20 | 10[117] | 18 | 15 | 17[6] | 23 |
| u3=-3 | 11[72] | 15 | 21 | 18 | 12[88] | 21 |
| u4=0 | 14[81] | 20 | 20 | 13[88] | 23 | 12 |
| u5=-3 | 13 | 21 | 19 | 10[110] | 20 | 14 |
| u6=-15 | 0 | 0 | 0[141] | 0 | 0[11] | 0 |

|  |  |
| --- | --- |
| Небазисная переменная |  |
| x11 | **0+14-22=-8** |
| x21 | **2+14-20=-4** |
| x51 | **(-3)+14-13=-2** |
| x61 | **-15+14-0=-1** |
| x12 | **0+8-12=-4** |
| x32 | **(-3)+8-15=-10** |
| x42 | **0+8-20=-12** |
| x52 | **(-3)+8-21=-16** |
| x62 | **-15+8+0=-7** |
| x13 | **0+15-16=-1** |
| x23 | **2+15-18=-1** |
| x33 | **(-3)+15-21=-9** |
| x53 | **(-3)+15-19=-7** |
| x24 | **2+13-15=0** |
| x34 | **(-3)+13-18=-8** |
| x15 | **0+15-21=-6** |
| x45 | **0+15-23=-8** |
| x55 | **(-3)+15-20=-8** |
| x26 | **2+11-23=-10** |
| x36 | **(-3)+11-21=-12** |
| x46 | **0+11-12=-1** |
| x56 | **(-3)+11-14=-6** |
| x64 | **(-15)+13-0=-2** |
| x66 | **(-15)+11-0=-4** |

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

Минимальные затраты составят: F(x) = 13\*5 + 11\*173 + 10\*117 + 17\*6 + 11\*72 + 12\*88 + 14\*81 + 13\*88 + 10\*110 + 0\*141 + 0\*11 = 8466.

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

# **Лабораторная работа № 6**

**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.

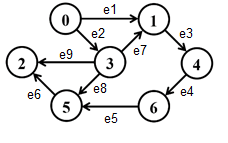
**Способы представления графов**

Исходный граф:



Матрица смежности:

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



Матрица инцидентности:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | e1 | e2 | e3 | e4 | e5 | e6 | e7 | e8 | e9 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | -1 | 0 | 1 | 0 | 0 | 0 | -1 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | -1 | 0 | 0 | -1 |
| 3 | 0 | -1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 4 | 0 | 0 | -1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | -1 | 1 | 0 | -1 | 0 |
| 6 | 0 | 0 | 0 | -1 | 1 | 0 | 0 | 0 | 0 |

Список смежных вершин:

0: 1 3

1: 4

2: -

3: 1 2 5

4: 6

5: 2

6: 5

**Алгоритм поиска в ширину (BFS)**

Исходный граф:



Шаг 1

Текущая вершина: 0

Состояние очереди: q = {1, 3}.

Шаг 2

Текущая вершина: 1.

Состояние очереди: q = {3, 4}.

Шаг 3

Текущая вершина: 3.

Состояние очереди: q = {4, 2, 5}.

Шаг 4

Текущая вершина: 4.

Состояние очереди: q = {2, 5, 6}.

Шаг 5

Текущая вершина: 2.

Состояние очереди: q = {5, 6}.

Шаг 6

Текущая вершина: 5.

Состояние очереди: q = {6}.

Шаг 7

Текущая вершина: 6.

Состояние очереди: q = {}.

Итоговая последовательность вершин: 0-1-3-4-2-5-6

**Алгоритм поиска в глубину (DFS) и топологическая сортировка**

Исходный граф:



Алгоритм обхода графа в глубину (Depth-First Search, DFS) — это алгоритм, который используется для обхода всех вершин графа. Он начинает с одной вершины и идет вглубь графа, пока не достигнет конца пути. Затем он возвращается на предыдущий уровень и продолжает обход до тех пор, пока не пройдет все вершины.

Шаг 1

Текущая вершина: 0

Состояние стека: q = {1, 3}.

Шаг 2

Текущая вершина: 1

Состояние стека: q = {4, 3}.

Шаг 3

Текущая вершина: 4

Состояние стека: q = {6, 3}.

Шаг 4

Текущая вершина: 6

Состояние стека: q = {5, 3}.

Шаг 5

Текущая вершина: 5.

Состояние стека: q = {2, 3}.

Шаг 6

Текущая вершина: 2.

Состояние стека: q = {3}.

Шаг 7

Текущая вершина: 3.

Состояние стека: q = {}.

Итоговая последовательность вершин: 0-1-4-6-5-2-3.

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

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

Шаг 1

Текущая вершина: 0.

Состояние стека: s = {1, 3, 0}.

Массив топологически отсортированных вершин: d = {}.

Шаг 2

Текущая вершина: 1.

Состояние стека: s = {4, 1, 3, 0}.

Массив топологически отсортированных вершин: d = {}.

Шаг 3

Текущая вершина: 4.

Состояние стека: s = {6, 4, 1, 3, 0}.

Массив топологически отсортированных вершин: d = {}.

Шаг 4

Текущая вершина: 6.

Состояние стека: s = {5, 6, 4, 1, 3, 0}.

Массив топологически отсортированных вершин: d = {}.

Шаг 5

Текущая вершина: 5.

Состояние стека: s = {2, 5, 6, 4, 1, 3, 0}.

Массив топологически отсортированных вершин: d = {}.

Шаг 6

Текущая вершина: 2.

Состояние стека: s = {5, 6, 4, 1, 3, 0}.

Массив топологически отсортированных вершин: d = {2}.

Шаг 7

Текущая вершина: 5.

Состояние стека: s = {6, 4, 1, 3, 0}.

Массив топологически отсортированных вершин: d = {2, 5}.

Шаг 8

Текущая вершина: 6.

Состояние стека: s = {4, 1, 3, 0}.

Массив топологически отсортированных вершин: d = {2, 5, 6}.

Шаг 9

Текущая вершина: 4.

Состояние стека: s = {1, 3, 0}.

Массив топологически отсортированных вершин: d = {2, 5, 6, 4}.

Шаг 10

Текущая вершина: 1.

Состояние стека: s = {3, 0}.

Массив топологически отсортированных вершин: d = {2, 5, 6, 4, 1}.

Шаг 11

Текущая вершина: 3.

Состояние стека: s = {0}.

Массив топологически отсортированных вершин: d = {2, 5, 6, 4, 1, 3}.

Шаг 12

Текущая вершина: 0.

Состояние стека: s = {}.

Массив топологически отсортированных вершин: d = {2, 5, 6, 4, 1, 3, 0}.

Итоговая последовательность вершин: 2-5-6-4-1-3-0

**Задание №3**

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

Код файла Graph.h представлен в листинге 1.1.

|  |
| --- |
| #pragma once  #include <list>  #include "Graph.h"  namespace graph  {  struct AList;  struct AMatrix // матрица смежности  {  int n\_vertex; // количество вершин  int\* mr; // матрица  AMatrix(int n); // создать нулевую матрицу n\*n  AMatrix(int n, int mr[]); // создать матрицу n\*n и  AMatrix(const AMatrix& am); // создать подобную матрицу  AMatrix(const AList& al); // создать матрицу из спискового  void set(int i, int j, int r); // записать mr[i,j] = r  int get(int i, int j)const; // элемент mr[i,j]  };  struct AList // списки смежности  {  int n\_vertex; // количество вершин  std::list<int>\* mr; // массив списков  void create(int n); // создать массив пустых списков  AList(int n); // создать массив пустых списков  AList(int n, int mr[]); // создать списковое представление  AList(const AMatrix& am); // создать списковое представление  AList(const AList& al); // создать подобную структуру  void add(int i, int j); // добавить в i-ый список  int size(int i) const; // размер i-го списка  int get(int i, int j)const; // j-ый элемент i-го списка  };  } |

Листинг 1.1 – Код файла Graph.h

Реализация методов структур Alist и AMatrix представлена в листинге 1.2.

|  |
| --- |
| #include "Graph.h"  namespace graph  {  AMatrix::AMatrix(int n)  {  this->n\_vertex = n;  this->mr = new int[this->n\_vertex \* this->n\_vertex];  for (int i = 0; i < n \* n; i++)mr[i] = 0;  };  AMatrix::AMatrix(int n, int mr[])  {  this->n\_vertex = n;  this->mr = mr;  };  AMatrix::AMatrix(const AMatrix& am)  {  this->n\_vertex = am.n\_vertex;  this->mr = new int[this->n\_vertex \* this->n\_vertex];  for (int i = 0; i < this->n\_vertex; i++)  for (int j = 0; j < this->n\_vertex; j++)  this->set(i, j, am.get(i, j));  };  AMatrix::AMatrix(const AList& al)  {  this->n\_vertex = al.n\_vertex;  this->mr = new int[this->n\_vertex \* this->n\_vertex];  for (int k = 0; k < this->n\_vertex \* this->n\_vertex; k++)mr[k] = 0;  for (int i = 0; i < this->n\_vertex; i++)  for (int j = 0; j < al.size(i); j++) this->set(i, al.get(i, j), 1);  };  void AMatrix::set(int i, int j, int r) { this->mr[i \* this->n\_vertex + j] = r; };  int AMatrix::get(int i, int j)const  {  return this->mr[i \* this->n\_vertex + j];  };  void AList::create(int n)  {  this->mr = new std::list<int>[this->n\_vertex = n];  };  AList::AList(int n) { create(n); }  AList::AList(const AMatrix& am)  {  this->create(am.n\_vertex);  for (int i = 0; i < this->n\_vertex; i++)  for (int j = 0; j < this->n\_vertex; j++)  if (am.get(i, j) != 0) this->add(i, j);  };  AList::AList(const AList& al)  {  this->create(al.n\_vertex);  for (int i = 0; i < this->n\_vertex; i++)  for (int j = 0; j < al.size(i); j++) this->add(i, al.get(i, j));  };  AList::AList(int n, int mr[])  {  this->create(n);  for (int i = 0; i < this->n\_vertex; i++)  for (int j = 0; j < this->n\_vertex; j++)  if (mr[i \* this->n\_vertex + j] != 0) this->add(i, j);  };  void AList::add(int i, int j)  {  this->mr[i].push\_back(j);  };  int AList::size(int i) const  { return (int)this->mr[i].size(); };  int AList::get(int i, int j)const  {  std::list<int>::iterator rc = this->mr[i].begin();  for (int k = 0; k < j; k++) rc++;  return (int)\*rc;  };  } |

Листинг 1.2 – Реализация методов структур AList и AMatrix

Код файла BFS.h представлен в листинге 1.3.

|  |
| --- |
| #pragma once  #include "Graph.h"  #include <queue>  struct BFS  {  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 1.3 – Код файла BFS.h

Реализация методов структуры BFS представлена в листинге 1.4.

|  |
| --- |
| #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 1.4 – Реализация методов структуры BFS

Результат работы функций представлен на рисунке 1.1.

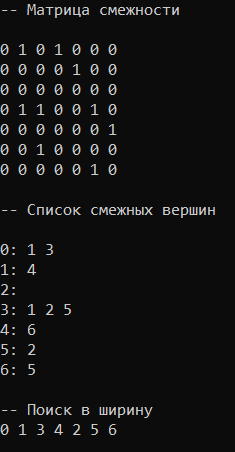


Рисунок 1.1 – Результат работы функций

**Задание №4-5**

Разработать функцию **DFS** обхода вершин графа, используя метод поиска глубину. Продемонстрировать работу функции.

Доработайте функцию **DFS**,для выполнения топологической сортировки графа.

Код файла DFS.h представлен в листинге 1.5.

|  |
| --- |
| #pragma once  #include "Graph.h"  #include <vector>  struct DFS  {  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 1.5 – Код файла DFS.h

Реализация методов структуры DFS представлена в листинге 1.6.

|  |
| --- |
| #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++)  {  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 1.6 – Реализация методов структуры DFS

Результат работы функций представлен на рисунке 1.2.

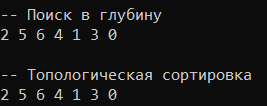
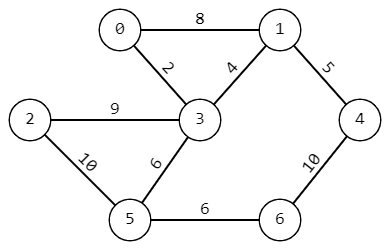


Рисунок 1.2 – Результат работы функций

**Задание №6**

По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Прима.

Исходный граф:



Шаг 1

Текущие вершины дерева: 0

Выбираем ребро 0-3

Шаг 2

Текущие вершины дерева: 0, 3

Выбираем ребро 3-1

Шаг 3

Текущие вершины дерева: 0, 3, 1

Выбираем ребро 1-4

Шаг 4

Текущие вершины дерева: 0, 3, 1, 4

Выбираем ребро 3-5

Шаг 5

Текущие вершины дерева: 0, 3, 1, 4, 5

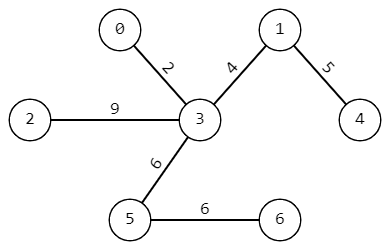
Выбираем ребро 3-2

Шаг 6

Текущие вершины дерева: 0, 3, 1, 4, 5, 2

Выбираем ребро 5-6

Полученное минимальное остовное дерево:



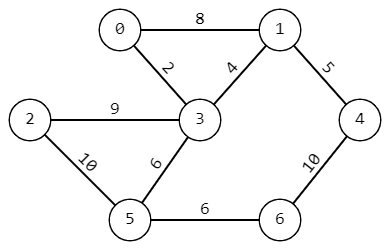
Вес дерева: 32.

**Задание №7**

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

В начале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовым деревом минимального веса.

Исходный граф:



Отсортируем все ребра по возрастанию веса:

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

Начальное ребро – это ребро с минимальным весом, а именно 0-3.

Шаг 1

Множество ребер остовного дерева: {(0, 3)}.

Шаг 2

Множество ребер остовного дерева: {(0, 3), (1, 3)}.

Шаг 3

Множество ребер остовного дерева: {(0, 3), (1, 3), (1, 4)}.

Шаг 4

Множество ребер остовного дерева: {(0, 3), (1, 3), (1, 4), (3, 5)}.

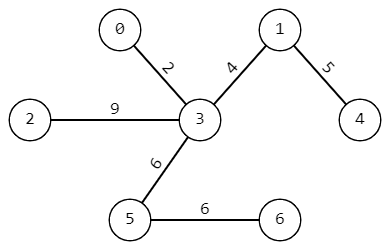
Шаг 5

Множество ребер остовного дерева: {(0, 3), (1, 3), (1, 4), (3, 5), (5, 6)}.

Шаг 6

Множество ребер остовного дерева: {(0, 3), (1, 3), (1, 4), (3, 5), (5, 6), (2, 3)}.

Полученное минимальное остовное дерево:



Вес дерева: 32.

**Вывод:** в ходе лабораторной работы были освоены сущность и программная реализация: а) способов представления графов; б) алгоритмов поиска в ширину и глубину; в) алгоритма топологической сортировки графов и разобрала алгоритм Прима и алгоритм Крускала.

# **Лабораторная работа № 7**

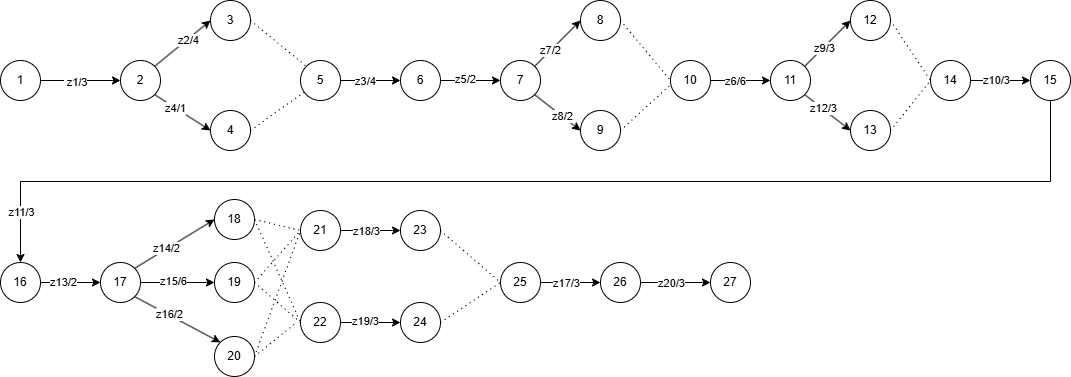
**Цель работы:** Приобретение навыков сетевого планирования и составления сетевых графиков, приобретение опыта нахождения критического пути.

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

Тема: «Создание облачного хранилища».

|  |  |  |  |
| --- | --- | --- | --- |
| Код операции | Наименование операции | Предшествующие операции | t |
| 1. Планирование | | | |
| Z1 | Определение требований к хранилищу |  | 3 |
| Z2 | Анализ бизнес-потребностей | Z1 | 4 |
| Z3 | Выбор облачной платформы | Z2, Z4 | 4 |
| Z4 | Определение бюджета и графика работ | Z1 | 1 |
| 2. Дизайн и настройка | | | |
| Z5 | Выбор технологий и оборудования для хранения и обработки данных | Z3 | 2 |
| Z6 | Создание архитектуры хранилища | Z7, Z8 | 6 |
| Z7 | Настройка облачной инфраструктуры | Z5 | 2 |
| Z8 | Установка программного обеспечения для управления хранилищем | Z5 | 2 |
| 3. Тестирование и оптимизация | | | |
| Z9 | Проведение тестирования системы | Z6 | 3 |
| Z10 | Выявление и устранение проблем | Z9 Z12 | 3 |
| Z11 | Оптимизация системы | Z10 | 3 |
| Z12 | Проведение интеграционных тестов | Z6 | 3 |
| 4. Развёртывание и поддержка | | | |
| Z13 | Развёртывание хранилища на серверах облачной платформы | Z11 | 2 |
| Z14 | Поддержка и обслуживание системы после запуска | Z13 | 2 |
| Z15 | Обеспечение безопасности и защиты данных в хранилище | Z13 | 6 |
| Z16 | Резервное копирование и восстановление данных | Z13 | 2 |
| 5. Обучение пользователей и техническая поддержка | | | |
| Z17 | Проведение обучения пользователей | Z18 Z19 | 3 |
| Z18 | Подготовка руководств пользователя | Z14 Z15 Z16 | 3 |
| Z19 | Техническая поддержка | Z14 Z15 Z16 | 3 |
| Z20 | Проведение обновлений и улучшений | Z17 | 3 |

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



Длина критического пути: 47 дней. Задачи, принадлежащие критическому пути: z1, z2, z3, z5, z7, z6, z9, z10, z11, z13, z15, z18, z17, z20.

**Вывод к 3 заданию:**

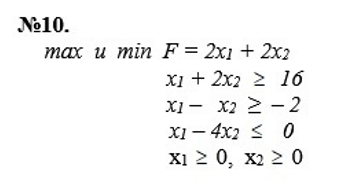
1. Общая продолжительность проекта по критическому пути: 47 дней.
2. Это минимальное время, за которое можно завершить проект.
3. Любая задержка на этих задачах повлечет сдвиг всего проекта.
4. Операции вне критического пути (например, Z4, Z12, Z15, Z16, Z19, Z22 и др.) — имеют резерв времени (float), и могут быть немного сдвинуты без влияния на срок всего проекта.

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

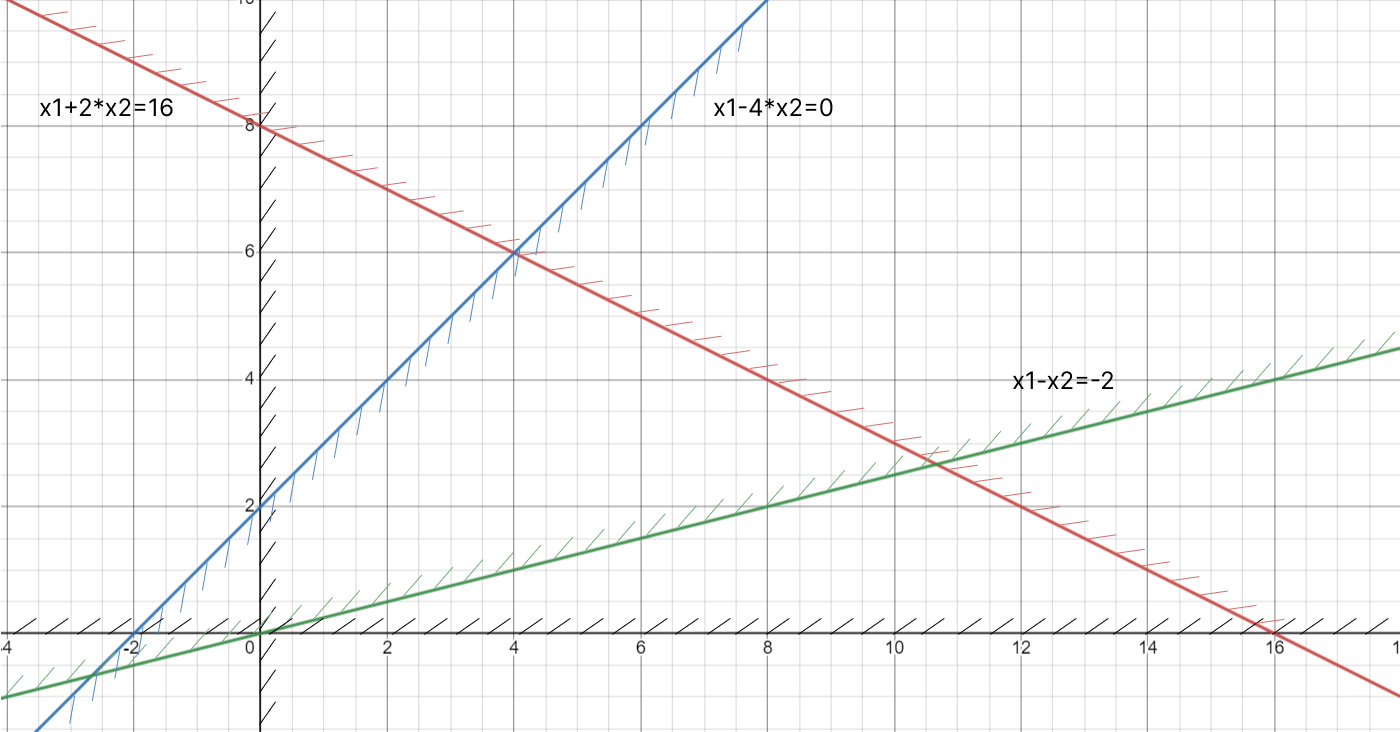
# **Лабораторная работа № 8**

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

**Задание**

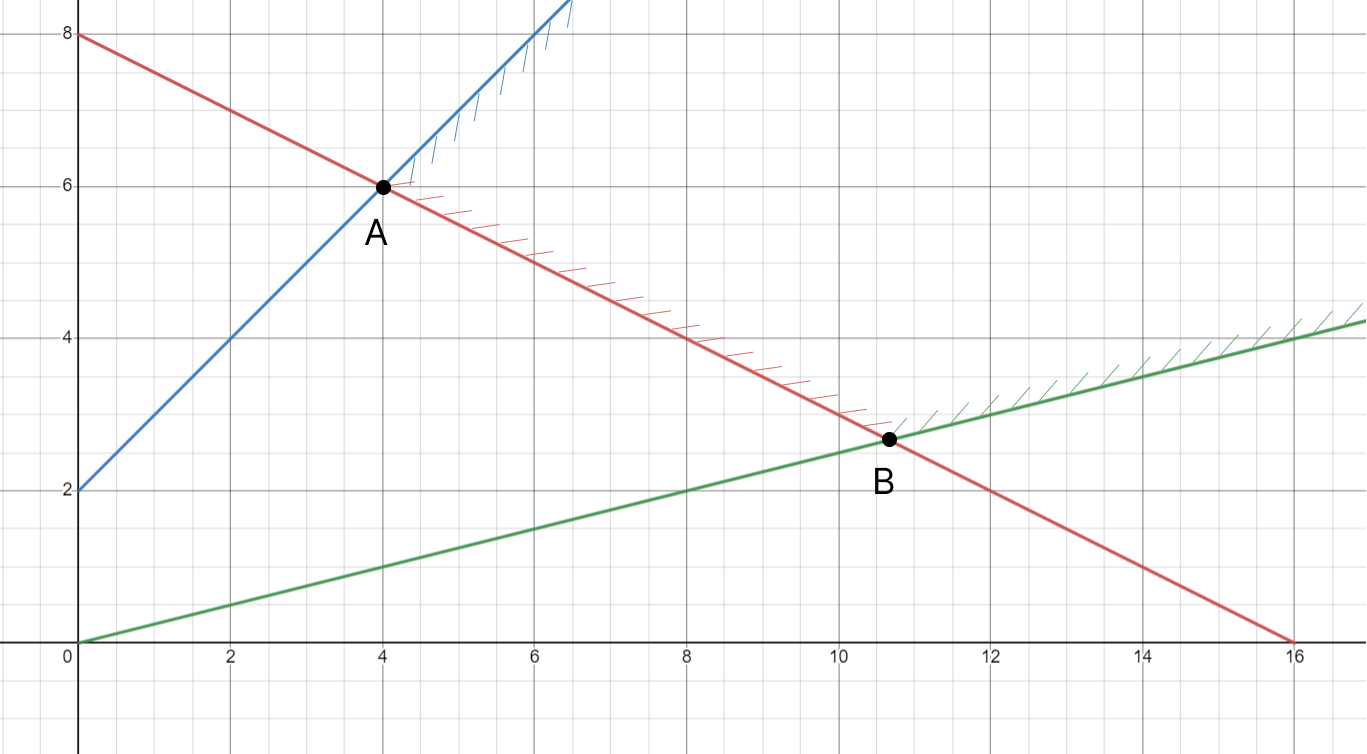


Шаг №1. Построить область допустимых решений, т.е. решить графически систему неравенств. Для этого построить каждую прямую и определить полуплоскости, заданные неравенствами (полуплоскости обозначены штрихом).



Шаг №2. Границы области допустимых решений.

Пересечением полуплоскостей будет являться область допустимых решений, координаты точек которого удовлетворяют условию неравенствам системы ограничений задачи. Обозначим границы области многоугольника решений.

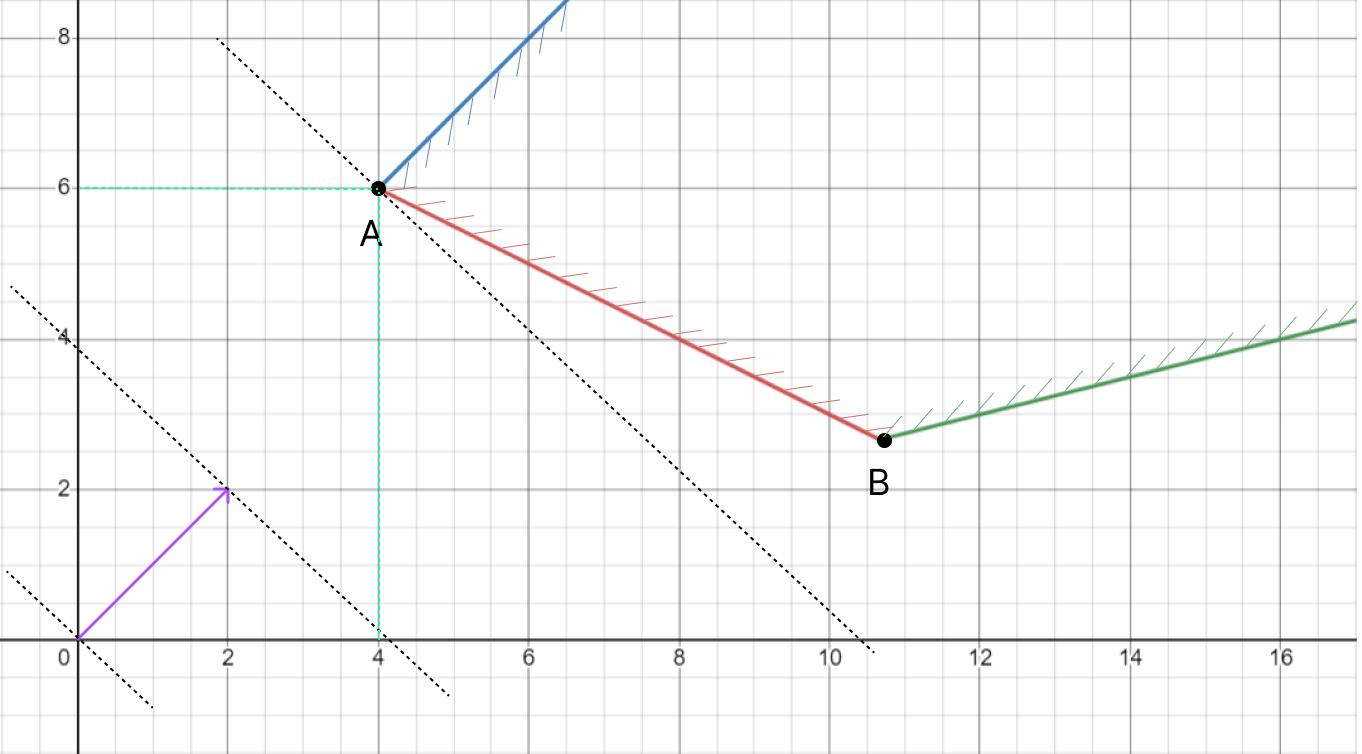


Оптимальные решения отсутствуют, так как система ограничений образует неограниченное сверху множество. Функция F в данном случае стремится к бесконечности, так как прямую функции можно передвигать в направлении вектора градиента как угодно далеко.

Однако, можно найти минимум.

Шаг №3. Целевая функция задачи F = 2x1+2x2 → min.

Вектор-градиент, составленный из коэффициентов целевой функции, указывает направление максимизации F(x). Начало вектора – точка (0; 0), конец – точка (2;2). Будем двигать эту прямую параллельным образом. Поскольку нас интересует минимальное решение, поэтому двигаем прямую до первого касания обозначенной области. На графике эта прямая обозначена пунктирной линией.



Прямая пересекает область в точке A. Так как точка A получена в результате пересечения прямых x1+2\*x2=16 и x1-x2=-2, то ее координаты удовлетворяют уравнениям этих прямых.

Решив систему уравнений, получим: x1 = 4, x2 = 6, откуда найдем минимальное значение целевой функции:

F(x) = 2\*4 + 2\*6 = 20

Таким образом, Fmin = 20, Fmax =+∞.

**Вывод:** в ходе выполнения лабораторной работы был освоен графический метод решения задач.