Учреждение образования

«Белорусский государственный технологический университет»

Кафедра ИСИТ

**«Математическое программирование»**

**Лабораторная работа №1**

**Вспомогательные функции**

Выполнил:

Студент 2 курса 7 группы ФИТ

Воликов Дмитрий Анатольевич

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

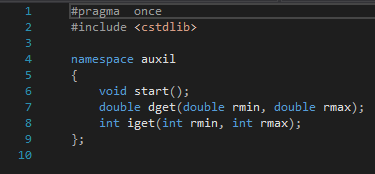
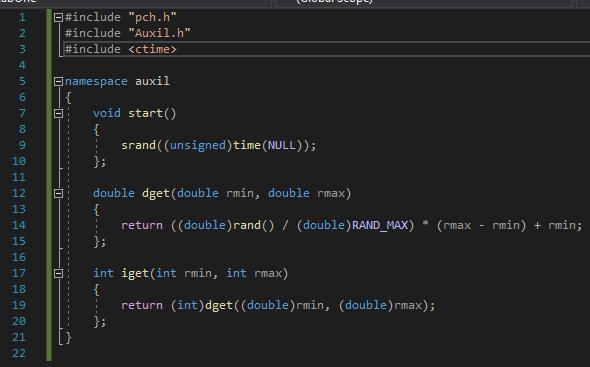
**Задание 1.**Разработайте три функции (start, dget и iget), используя следующие спецификации:

Рисунок 1.1 – Заголовочный файл Auxil.h

Рисунок 1.2 – Файл Auxil.cpp

Рисунок 1.3 – Результат выполнения функции получения случайного числа

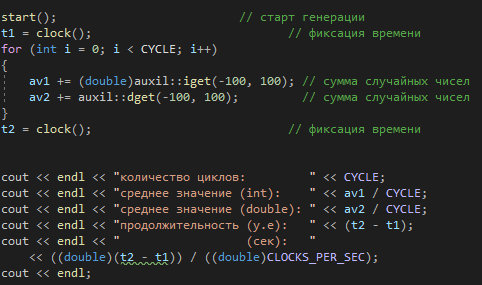
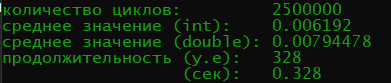
**Задание 2.**Реализовать пример*.* Для проверки работоспособности разработанных функций и приобретения навыков замера продолжительности процесса вычисления реализуйте программу, приведенную в примере 2.

Рисунок 1.4 – Реализация примера

Рисунок 1.5 – Результат выполнения реализации примера

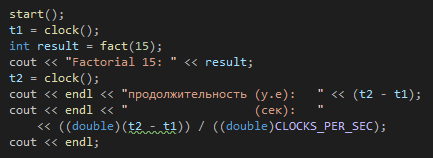
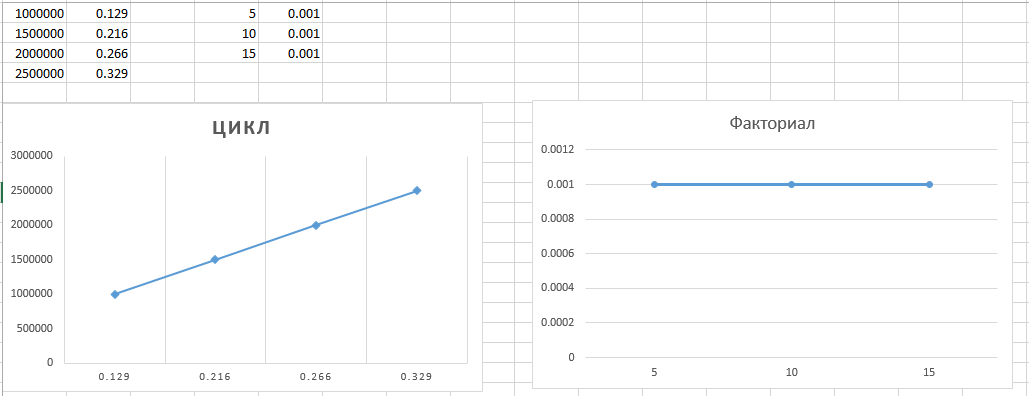
**Задание 3.** Проведите необходимые эксперименты и постройте график зависимости (Excel) продолжительности процесса вычисления от количества циклов в примере 2. Проанализируйте характер зависимости. Проведите исследование любого другого рекурсивного алгоритма, например, вычисления факториала или генератора чисел Фибоначчи (прим. – например вычислите каким будет 100-е, 200-е, 300-е и т.д число), и включите в отчет график.

Рисунок 1.6 – Проверка на скорость рекурсивного алгоритма вычисления факториала

Рисунок 1.7 – Графики Excel на основе полученных данных в результате выполнения программы

**Вывод**: в ходе этой лабораторной работы я научился использовать стандартные библиотеки времени для определения времени работы алгоритмов. Создал свои файлы заголовков с реализациями функций.

Учреждение образования

«Белорусский государственный технологический университет»

Кафедра ИСИТ

**«Математическое программирование»**

**Лабораторная работа №2**

**Комбинаторные алгоритмы решения оптимизационных задач**

Выполнил:

Студент 2 курса 7 группы ФИТ

Воликов Дмитрий Анатольевич

**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.

#pragma once

namespace combi

{

//подмножества множества

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

void reset();

unsigned \_\_int64 count();

short ntx(short i);

};

//сочетания

struct xcombination // генератор сочетаний (эвристика)

{

short n, // количество элементов исходного множества

m, nc, // 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 count() const;

};

//перестановки

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

};

//размещения

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 2.1 – Код заголовочного файла Combi.h

#include "stdafx.h"

#include "Combi.h"

#include <algorithm>

#define NINF ((short)0x8000)

namespace combi

{

subset::subset(short n)

{

this->n = n;

sset = new short[n];

reset();

};

void subset::reset()

{

sn = 0;

mask = 0;

};

short subset::getfirst()

{

long buf = mask;

sn = 0;

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

{

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

buf >>= 1;

}

return sn;

};

short subset::getnext()

{

int rc = -1;

sn = 0;

if (++mask < count()) rc = getfirst();

return rc;

};

short subset::ntx(short i)

{

return sset[i];

};

unsigned \_\_int64 subset::count()

{

return (unsigned long)(1 << n);

};

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;

};

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

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

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];

};

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

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

{

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

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

};

}

Листинг 2.2 – Код файла Combi.cpp

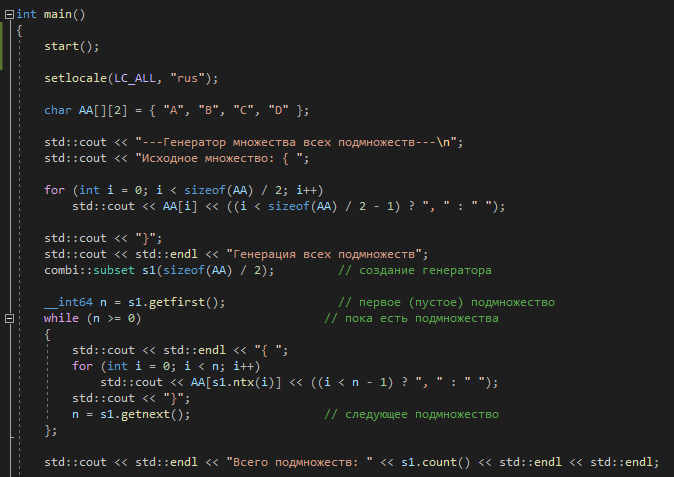
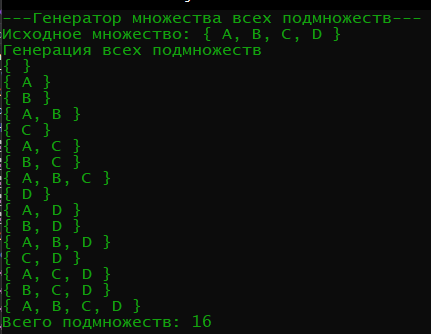
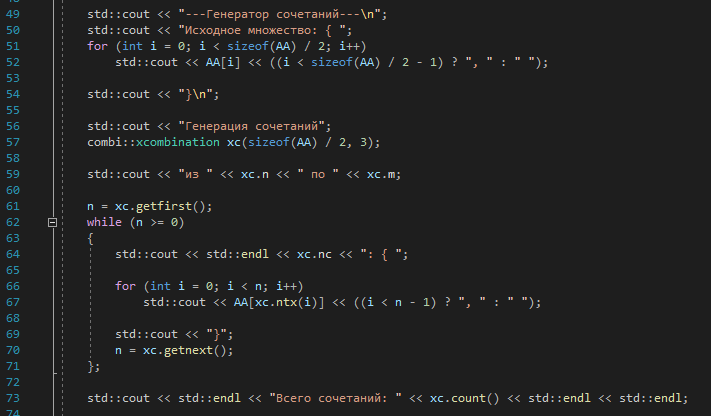
Задание 1. Разобрать и разработать генератор подмножеств заданного множества.

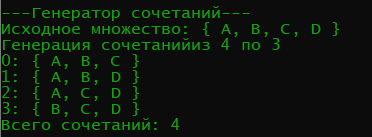
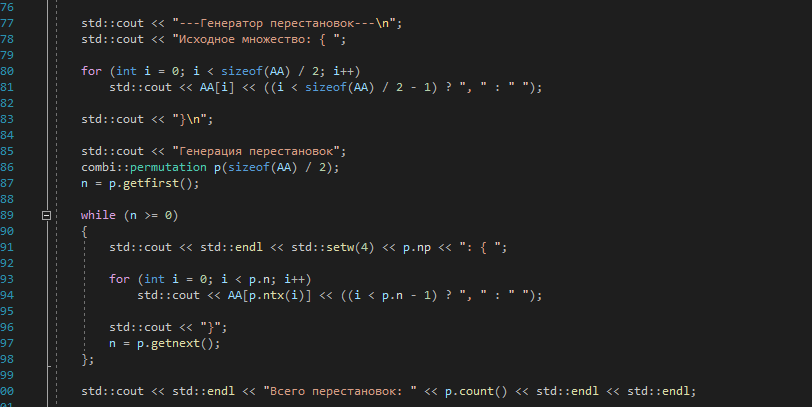
Рисунок 2.1 – Код расчёта подмножеств исходного множества



Рисунок 2.2 – Результат выполнения программы

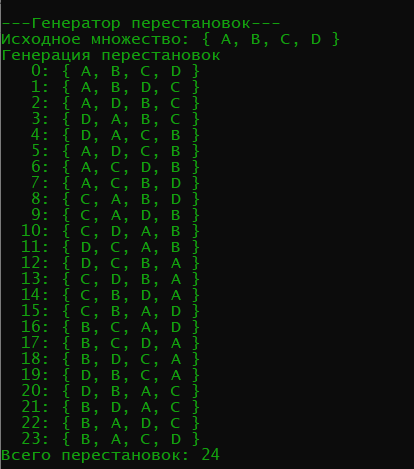
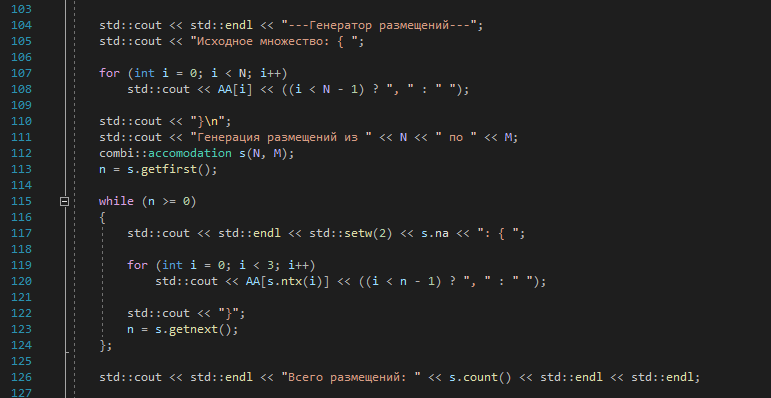
**Задание 2**. Разобрать и разработать генератор сочетаний.

Рисунок 2.3 – Код генерации сочетаний из исходного подмножества

Рисунок 2.4 – Результат генерации сочетаний из исходного подмножества

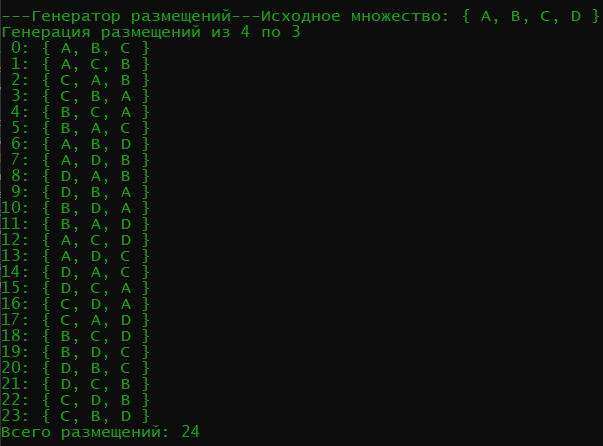
**Задание 3**. Разобрать и разработать генератор перестановок.

Рисунок 2.4 – Код генерации перестановок исходного множества

Рисунок 2.5 – Результат вычисления перестановок исходного множества

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

Рисунок 2.6 – Код генерации размещений из исходного множества

Рисунок 2.7 – Результат генерации размещений исходного множества

**Задание 5**. 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.);

#include "stdafx.h"

#include "Boat.h"

namespace boatfnc

{

int calcv(combi::xcombination s, const int v[]) // вес

{

int rc = 0;

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

return rc;

};

int calcc(combi::xcombination s, const int c[]) // доход

{

int rc = 0;

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

return rc;

};

void copycomb(short m, short\* r1, const short\* r2) // копировать

{

for (int i = 0; i < m; i++) r1[i] = r2[i];

};

}

int boat(

int V, // [in] максимальный вес груза

short m, // [in] количество мест для контейнеров

short n, // [in] всего контейнеров

const int v[], // [in] вес каждого контейнера

const int c[], // [in] доход от перевозки каждого контейнера

short r[] // [out] результат: индексы выбранных контейнеров

)

{

combi::xcombination xc(n, m);

int rc = 0, i = xc.getfirst(), cc = 0;

while (i > 0)

{

if (boatfnc::calcv(xc, v) <= V)

{

if ((cc = boatfnc::calcc(xc, c)) > rc)

{

rc = cc; boatfnc::copycomb(m, r, xc.sset);

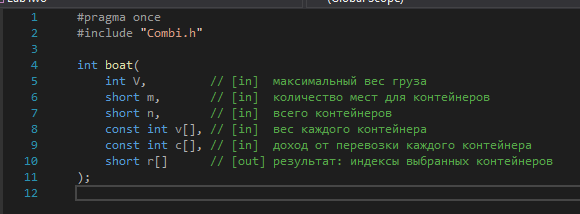
}

}

i = xc.getnext();

};

return rc;

};

Листинг 2.3 – Код файла Boat.cpp

Рисунок 2.8 – Заголовочный файл Boat.h

int V = 1500,

v[] = { 100, 200, 300, 400, 500, 150,

100, 200, 300, 400, 500, 150,

100, 200, 300, 400, 500, 150,

100, 200, 300, 400, 500, 150,

300, 670, 200, 350, 250, 400,

150, 250, 300, 400, 350},

c[NN] = { 10, 10, 20, 54, 111, 25,

11, 15, 22, 120, 67, 26,

14, 18, 100, 90, 83, 27,

19, 99, 61, 87, 30, 25,

50, 79, 20, 25, 15, 30,

150, 100, 75, 50, 40};

short r[MM];

clock\_t t1 = 0, t2 = 0;

t1 = clock();

int cc = boat(

V, // [in] максимальный вес груза

MM, // [in] количество мест для контейнеров

NN, // [in] всего контейнеров

v, // [in] вес каждого контейнера

c, // [in] доход от перевозки каждого контейнера

r // [out] результат: индексы выбранных контейнеров

);

t2 = clock();

std::cout << std::endl << "- Задача о размещении контейнеров на судне";

std::cout << std::endl << "- общее количество контейнеров : " << NN;

std::cout << std::endl << "- количество мест для контейнеров : " << MM;

std::cout << std::endl << "- ограничение по суммарному весу : " << V;

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

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

std::cout << std::setw(3) << v[i] << " ";

std::cout << std::endl << "- доход от перевозки : ";

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

std::cout << std::setw(3) << c[i] << " ";

std::cout << std::endl << "- выбраны контейнеры (0,1,...,m-1): ";

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

std::cout << r[i] << " ";

std::cout << std::endl << "- доход от перевозки : " << cc;

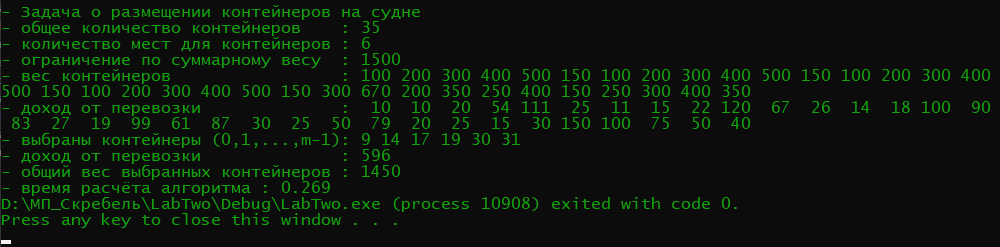
std::cout << std::endl << "- общий вес выбранных контейнеров : ";

int sum = 0;

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

sum += v[r[i]];

std::cout << sum;

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

Листинг 2.4 – Код расчёта размещения контейнеров

Рисунок 2.9 – Результат выполнения расчёта размещения контейнеров в лодке

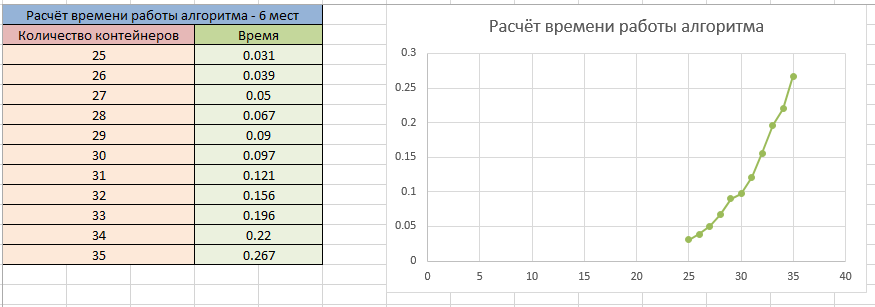
Задание 6. Исследовать зависимость времени вычисления необходимое для решения задачи (в соответствии с вариантом) от размерности задачи и результат в виде графика с небольшим пояснением занести в отчет: 3, 7, 11, 15) об оптимальной загрузке судна (количество мест на судне для контейнеров – 6, количество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

Рисунок 2.10 – Графики по рассчитанным результатам вычисления

**Вывод**: в ходе этой лабораторной работы я изучил элементы комбинаторики, реализовал возможность подсчёта размещения, сочетания и перестановки из исходного множества. А также научился вычислять подмножества исходного множества.

Учреждение образования

«Белорусский государственный технологический университет»

Кафедра ИСИТ

**«Математическое программирование»**

**Лабораторная работа №3**

**Метод ветвей и границ. Задача коммивояжера и методы её решения**

Выполнил:

Студент 2 курса 7 группы ФИТ

Воликов Дмитрий Анатольевич

**ЦЕЛЬ РАБОТЫ:** освоить общие принципы решения задач методом ветвей и границ, решить задачу о коммивояжере данным методом, сравнить полученное решение задачи с комбинаторным методом перестановок.

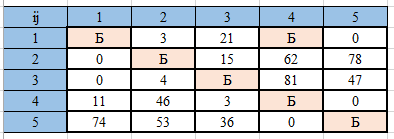
**Задание 1.** Сформулировать условие задачи коммивояжера с параметром. Для этого принять элементы матрицы расстояний равными:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** | Б | 6 | 24 | Б | 3 |
| **2** | 3 | Б | 18 | 65 | 81 |
| **3** | 5 | 9 | Б | 86 | 52 |
| **4** | 20 | 55 | 12 | Б | 9 |
| **5** | 90 | 69 | 52 | 16 | Б |

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

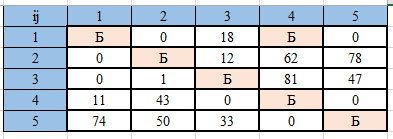
**Шаг 1.** Приведение матрицы по строкам, затем по столбцам и нахождение сумм констант приведения.

Вид матрицы после приведения по строкам:



Сумма констант по строкам – 36.

Вид матрицы после приведения по столбцам:



Сумма констант по столбцам – 6.

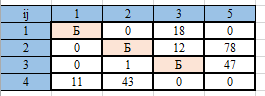
γ = 36 + 6 = 42 – длина нижней границы.

**Шаг 2.** Вычисление дуги, которая как можно сильнее повлияет на нижнюю границу длины допустимых кольцевых маршрутов.

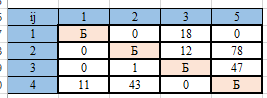


По вычислениям мы вывели, что дуга 5 -> 4 является той самой искомой дугой, так что выкинем её из нашей матрицы. Все допустимые кольцевые маршруты не будут меньше, чем нижняя граница, построенная для полностью приведенной таблицы увеличенная на 95.

Удалим из полностью приведенной матрицы строку 5 и столбец 4:

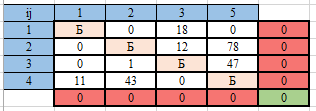


Кроме того, очевидно, что допустимый кольцевой маршрут, содержащий дугу (5, 4), не может содержать дугу (4, 5), поэтому на этом месте мы ставим знак «Б»:

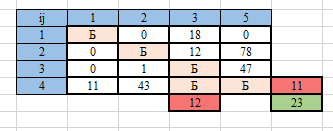


**Шаг 3.** Повторяем шаг 1 для результативной матрицы из шага 2.

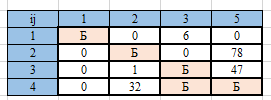
Приведение матрицы по строкам и столбцам не даёт никакого изменения, так все промежуточные константы равны 0.



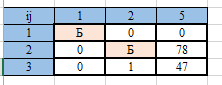
Согласно алгоритму дальнейший поиск решения следует осуществлять во множестве ***R***(5,4), поскольку именно здесь на текущий момент нижняя граница является меньшей.



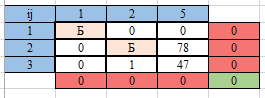
Приведём матрицу к полному виду:



Удалим строку 4 и столбец 3:

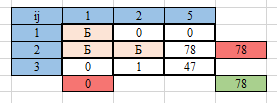


Посчитаем константы приведения и убедимся, что они снова равны 0:

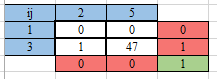


**Шаг 4**. Дальнейший поиск решения целесообразно осуществлять во множестве ***R*** (5, 4) (4, 3), так как здесь на текущий момент нижняя граница является меньшей.

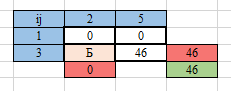
Анализ таблицы позволяет выявить дугу (2, 1), удаление которой приводит к максимальной сумме констант приведения (78).



Удалим строку 2 и столбец 1 и посчитаем константу приведения:



Приведём таблицу и найдём тот нуль, суммы констант которых дадут наибольшее значение. В нашем случае это дуги (3, 2) и (1, 5). В этом случае можно удалить любую из дуг:



После удаления дуги (3, 2), мы получим окончательную дугу, значение которой будет (1, 5):



В итоге мы получаем следующие дуги:

(5, 4) –> (4, 3) –> (2, 1) –> (3, 2) –> (1, 5)

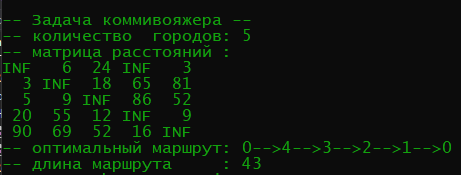
Расставим дуги в правильном порядке:

(1, 5) -> (5, 4) -> (4, 3) -> (3, 2) -> (2, 1).

Посчитаем сумму значений дуг:

Φ = 43.

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



**Вывод**: в ходе выполнения данной лабораторной работы я изучил опытным путём метод ветвей и границ на основе решения задачи о коммивояжёре. Решение задачи «руками» дало точно такое же решение, как то, что выдаёт нам программа с готовым алгоритмом на основе перестановок.

Учреждение образования

«Белорусский государственный технологический университет»

Кафедра ИСИТ

**«Математическое программирование»**

**Лабораторная работа №4**

**Динамическое программирование**

Выполнил:

Студент 2 курса 7 группы ФИТ

Воликов Дмитрий Анатольевич

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

**ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ**

***Задание 1.***На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной 200.

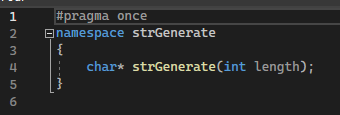


Рисунок 1 – strGenerate.h

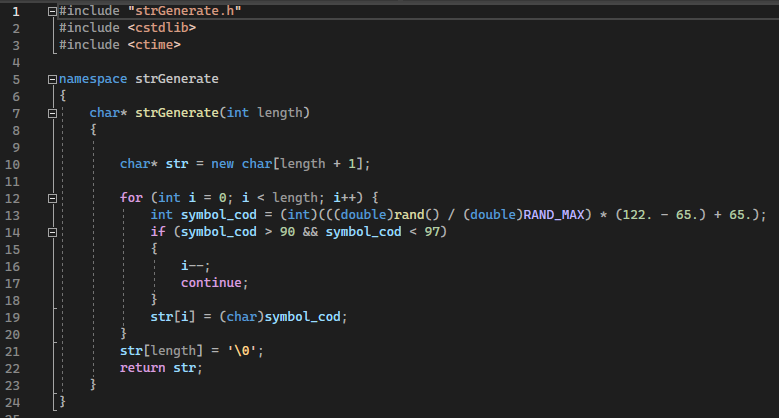


Рисунок 2 – strGenerate.cpp

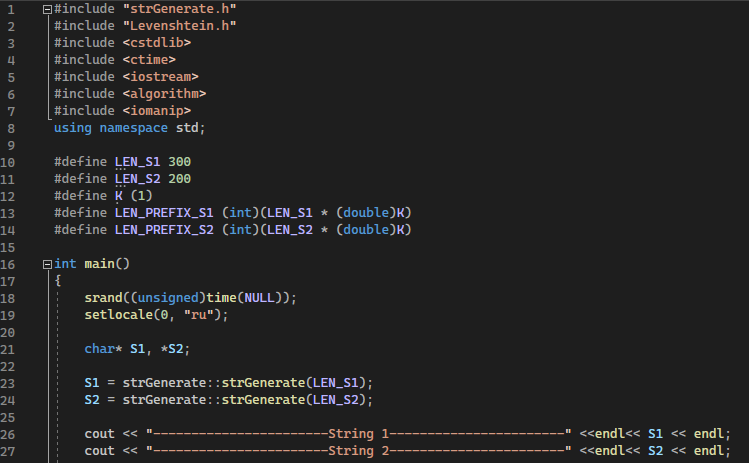


Рисунок 3 – main.cpp

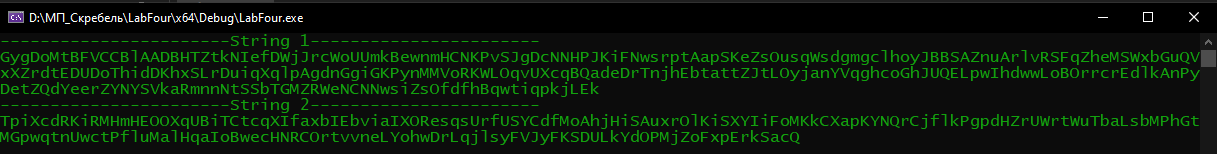
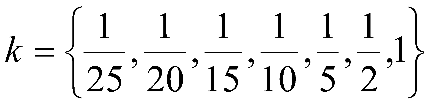


Рисунок 4 – Сгенерированные строки

***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).

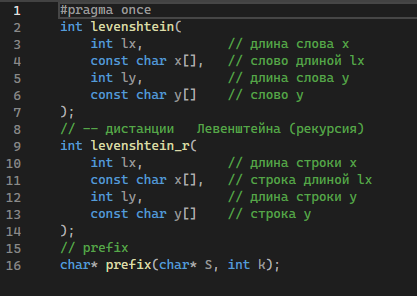


Рисунок 5 – Levenstein.h

#include <iomanip>

#include <algorithm>

#include "Levenshtein.h"

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

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

{

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

}

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

{

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

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

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

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

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

{

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

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

}

return DD(lx, ly);

}

int levenshtein\_r(

int lx, const char x[],

int ly, const char y[]

)

{

int rc = 0;

if (lx == 0) rc = ly;

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

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

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

else rc = min3(

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

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

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

);

return rc;

};

char\* prefix(char\* S, int k)

{

char\* str = new char[k + 1];

for (size\_t i = 0; i < k; i++)

{

str[i] = S[i];

}

str[k] = '\0';

return str;

}

Листинг 1 – Levenstein.cpp

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

cout << "\n\n";

cout << "Prefix1: " << prefix(S1, LEN\_PREFIX\_S1) << endl;

cout << "Prefix2: " << prefix(S2, LEN\_PREFIX\_S2) << endl;

cout << endl;

cout << endl << "-- Расстояние Левенштейна -----" << endl;

cout << endl << "--Длина --- Рекурсия -- Дин.програм. ---"

<< endl;

int lev = 0, lev\_r = 0;

int sum = 0, sum\_r = 0;

for (int i = 0; i < min(LEN\_PREFIX\_S1, LEN\_PREFIX\_S2); i++)

{

t1 = clock();

lev = levenshtein\_r(i, prefix(S1, LEN\_PREFIX\_S1), i, prefix(S2, LEN\_PREFIX\_S2));

t2 = clock();

t3 = clock();

lev\_r = levenshtein(i, prefix(S1, LEN\_PREFIX\_S1), i, prefix(S2, LEN\_PREFIX\_S2));

t4 = clock();

cout << right << setw(2) << i << "/" << setw(2) << i

<< " " << left << setw(10) << (t2 - t1)

<< " " << setw(10) << (t4 - t3) << endl;

sum\_r += t2 - t1;

sum += t4 - t3;

}

cout << lev << endl << lev\_r << endl;

cout << sum\_r << "\t" << sum << endl;

Листинг 2 – main.cpp (задача на нахождение расстояния Левенштейна)

**Задание 3**. Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от k (копии экрана и график вставить в отчет).

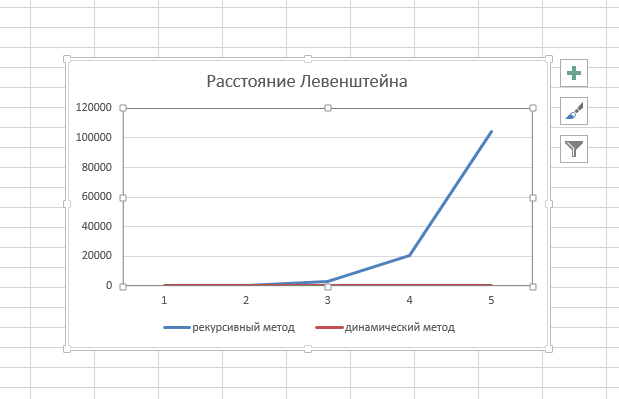


Рисунок 6 – график зависимости времени вычислений от k

***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).

Вариант 3: строка №1 – лом, строка №2 – гомон.



Рисунок 7 – алгоритм вычисления расстояния Левенштейна

Основные функции алгоритма Левенштейна:

* Len(X) – длина строки X.
* Cut(X) – строка X без последнего символа.
* Last(X) – последний символ строки X.

Последовательность алгоритма:

5. L(“”, “гомон”) = len(“гомон”) = 5  
   L(“”, “гомо”) = len(“гомо”) = 4
6. L(“”, “гомо”) = len(“гомо”) = 4  
   L(“”, “гом”) = len(“гом”) = 3
7. L(“”, “гом”) = len(“гом”) = 3  
   L(“”, “го”) = len(“го”) = 2



12. L(“”, “го”) = len(“го”) = 2  
    L(“”, “г”) = len(“г”) = 1
13. L(“лом”, “”) = len(“лом”) = 3  
    L(“ло”, “”) = len(“ло”) = 2
14. L(“ло”, “”) = len(“ло”) = 2  
    L(“л”, “”) = len(“л”) = 1
15. L(“л”, “”) = len(“л”) = 1  
    L(“”, “г”) = len(“г”) = 1  
    L(“”, “”) = 0
16. = 4

Следовательно, чтобы получить из слова «лом» слово «гомон» нужно сделать 3 операции: заменить «л» на «г», добавить «о» и «н».

***Задание 5.*** 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 k. Отобразить ход решения в отчете (по примеру из лекции) + код и копии экрана.

**Вариант 3**: последовательность символов №1 – ABCDFGI, последовательность символов №2 – EATUFI.

Ход решения задачи:

Следовательно, наибольшей общей подпоследовательностью двух исходных наборов символов является 3, а именно символы AFI.

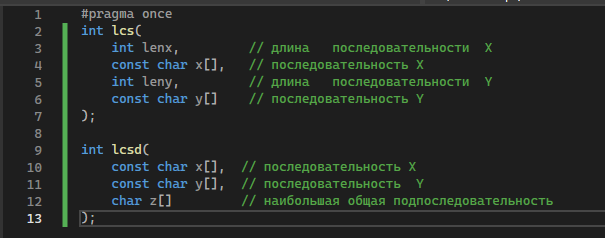


Рисунок 7 – LCS.h

#include <algorithm>

#include "LCS.h"

#include <cstring>

#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

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

}

Листинг 3 – LCS.cpp

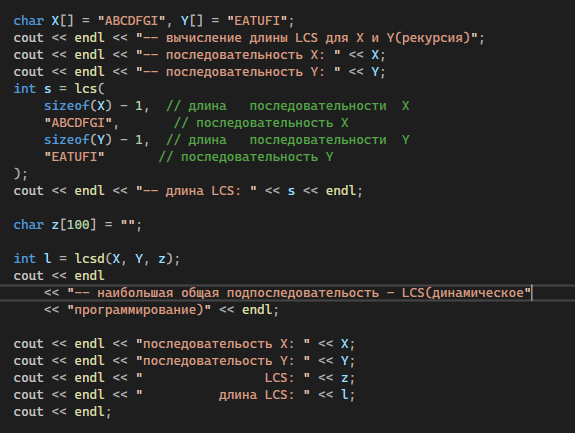


Рисунок 8 – main.cpp

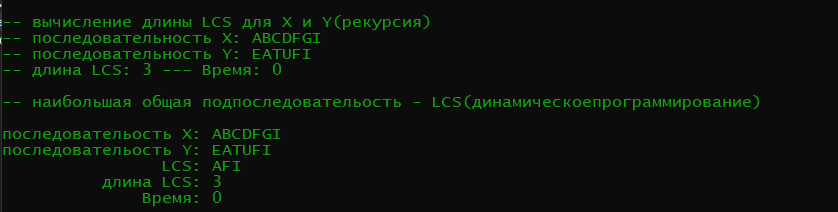


Рисунок 9 – результат вычислений программы

**Вывод**: в ходе данной лабораторной работы мною были изучены несколько новых алгоритмов: алгоритм нахождения расстояния Левенштейна и алгоритм нахождения наибольшей общей последовательности. Код изучен и добавлен в отчёт, как и графики, которые отображают время вычисления алгоритмов двумя разными методами: рекурсивным и динамическим.

Учреждение образования

«Белорусский государственный технологический университет»

Кафедра ИСИТ

**«Математическое программирование»**

**Лабораторная работа №5**

**ТРАНСПОРТНАЯ ЗАДАЧА**

Выполнил:

Студент 2 курса 7 группы ФИТ

Воликов Дмитрий Анатольевич

**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 (*3*). Оформить отчет.

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

**Ход решения**

**Тип задачи.** Найдём сумму потребностей, обозначив каждую за bj. Также сделаем для запасов, приняв каждый за ai.

Так, как суммы потребностей и запасов не равны, то задача является открытого типа. Из сумм потребностей и запасов можем выяснить, что потребностей больше, а значит надо добавить ещё одного поставщика, запасы которого будут составлять разницу потребностей и запасов (145).

**Составить опорный план**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 3 | 146 | 5 | 9 | 6 | 14 | 4 | 25 | 171 |
| 2 | 13 | 3 | 110 | 11 | 6 | 8 | 10 | 16 | 116 |
| 3 | 4 | 8 | 14 | 11 | 55 | 5 | 98 | 14 | 153 |
| 4 | 7 | 13 | 13 | 6 | 21 | 16 | 5 | 141 | 162 |
| 5 | 6 | 14 | 12 | 3 | 103 | 13 | 7 | 103 |
| 6 | 0 | 0 | 0 | 128 | 0 | 17 | 0 | 0 | 145 |
| ПОТРЕБНОСТИ | 146 | 110 | 134 | 196 | 98 | 166 |  |

Первое допустимое решение:

x11 = 3; x16 = 4; x22 = 3; x23 = 11; x34 = 11; x35 = 5; x44 = 6; x46 = 5; x54 = 3; x63 = 0; x64 = 0

Значение функции цели:

В итоге выходит 11 уравнений, что удовлетворяет условию m+n-1, где m – число поставщиков, n – число запасов.

Составим эти уравнения:

u1 + v1 = 3

u1 + v6 = 4

u2 + v2 = 3

u2 + v3 = 11

u3 + v4 = 11

u3 + v5 = 5

u4 + v4 = 6

u4 + v6 = 5

u5 + v4 = 3

u6 + v3 = 0

u6 + v4 = 0

Примем u1 = 0, тогда:

u1 = 0 u2 = 6 u3 = 6 u4 = 1 u5 = -2 u6 = -5

v1 = 3 v2 = -3 v3 = 5 v4 = 5 v5 = -1 v6 = 4

x12 = u1 + v2 – c12 = 0 – 3 – 5 = -8

x13 = u1 + v3 – c13 = 0 + 5 – 9 = -4

x14 = u1 + v4 – c14 = 0 + 5 – 6 = -1

x15 = u1 + v5 – c15 = 0 – 1 – 14 = -15

x21 = u2 + v1 – c21 = 6 + 3 – 13 = -4

x24 = u2 + v4 – c24 = 6 + 5 – 8 = 5

x25 = u2 + v5 – c25 = 6 – 1 – 10 = -5

x26 = u2 + v6 – c26 = 6 + 4 – 16 = -6

x31 = u3 + v1 – c31 = 6 + 3 – 4 = 5

x32 = u3 + v2 – c32 = 6 – 3 – 8 = -5

x33 = u3 + v3 – c33 = 6 + 5 – 14 = -3

x36 = u3 + v6 – c36 = 6 + 4 – 14 = -4

x41 = u4 + v1 – c41 = 1 + 3 – 7 = -3

x42 = u4 + v2 – c42 = 1 – 3 – 13 = -17

x43 = u4 + v3 – c43 = 1 + 5 – 13 = -7

x45 = u4 + v5 – c45 = 1 – 1 – 16 = -16

x51 = u5 + v1 – c51 = -2 + 3 – 6 = -5

x52 = u5 + v2 – c52 = -2 – 3 – 14 = -19

x53 = u5 + v3 – c53 = -2 + 5 – 12 = -9

x55 = u5 + v5 – c55 = -2 – 1 – 13 = -16

x56 = u5 + v6 – c56 = -2 + 4 – 7 = -5

x61 = u6 + v1 – c61 = u6 + v1 = -5 + 3 = -2

x62 = u6 + v2 – c62 = u6 + v1 = -5 – 3 = -8

x65 = u6 + v5 – c65 = u6 + v1 = -5 – 1 = -6

x66 = u6 + v6 – c66 = u6 + v1 = -5 + 4 = -1

Получилось два одинаковых положительных значения: 5. Т.е. можно менять значения циклически от любого (я выбрал x24)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 3 | 146 | 5 | 9 | 6 | 14 | 4 | 25 | 171 |
| 2 | 13 | 3 | 110 | 11 | 8 | 6 | 10 | 16 | 116 |
| 3 | 4 | 8 | 14 | 11 | 55 | 5 | 98 | 14 | 153 |
| 4 | 7 | 13 | 13 | 6 | 21 | 16 | 5 | 141 | 162 |
| 5 | 6 | 14 | 12 | 3 | 103 | 13 | 7 | 103 |
| 6 | 0 | 0 | 0 | 134 | 0 | 11 | 0 | 0 | 145 |
| ПОТРЕБНОСТИ | 146 | 110 | 134 | 196 | 98 | 166 |  |

u1 + v1 = 3

u1 + v6 = 4

u2 + v2 = 3

u2 + v4 = 8

u3 + v4 = 11

u3 + v5 = 5

u4 + v4 = 6

u4 + v6 = 5

u5 + v4 = 3

u6 + v3 = 0

u6 + v4 = 0

Примем u1 = 0, тогда:

u1 = 0 u2 = 3 u3 = 6 u4 = 1 u5 = -2 u6 = -5

v1 = 3 v2 = 0 v3 = 5 v4 = 5 v5 = -1 v6 = 4

x12 = u1 + v2 – c12 = 0 + 0 – 5 = -5

x13 = u1 + v3 – c13 = 0 + 5 – 9 = -4

x14 = u1 + v4 – c14 = 0 + 5 – 6 = -1

x15 = u1 + v5 – c15 = 0 – 1 – 14 = -15

x21 = u2 + v1 – c21 = 3 + 3 – 13 = -7

x23 = u2 + v3 – c23 = 3 + 5 – 11 = -3

x25 = u2 + v5 – c25 = 3 – 1 – 10 = -8

x26 = u2 + v6 – c26 = 3 + 4 – 16 = -9

x31 = u3 + v1 – c31 = 6 + 3 – 4 = 5

x32 = u3 + v2 – c32 = 6 + 0 – 8 = -2

x33 = u3 + v3 – c33 = 6 + 5 – 14 = -3

x36 = u3 + v6 – c36 = 6 + 4 – 14 = -4

x41 = u4 + v1 – c41 = 1 + 3 – 7 = -3

x42 = u4 + v2 – c42 = 1 + 0 – 13 = -12

x43 = u4 + v3 – c43 = 1 + 5 – 13 = -7

x45 = u4 + v5 – c45 = 1 – 1 – 16 = -16

x51 = u5 + v1 – c51 = -2 + 3 – 6 = -5

x52 = u5 + v2 – c52 = -2 + 0 – 14 = -16

x53 = u5 + v3 – c53 = -2 + 5 – 12 = -9

x55 = u5 + v5 – c55 = -2 – 1 – 13 = -16

x56 = u5 + v6 – c56 = -2 + 4 – 7 = -5

x61 = u6 + v1 – c61 = u6 + v1 = -5 + 3 = -2

x62 = u6 + v2 – c62 = u6 + v1 = -5 + 0 = -5

x65 = u6 + v5 – c65 = u6 + v1 = -5 – 1 = -6

x66 = u6 + v6 – c66 = u6 + v1 = -5 + 4 = -1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 3 | 91 | 5 | 9 | 6 | 14 | 4 | 80 | 171 |
| 2 | 13 | 3 | 110 | 11 | 8 | 6 | 10 | 16 | 116 |
| 3 | 4 | 55 | 8 | 14 | 11 | 5 | 98 | 14 | 153 |
| 4 | 7 | 13 | 13 | 6 | 76 | 16 | 5 | 86 | 162 |
| 5 | 6 | 14 | 12 | 3 | 103 | 13 | 7 | 103 |
| 6 | 0 | 0 | 0 | 134 | 0 | 11 | 0 | 0 | 145 |
| ПОТРЕБНОСТИ | 146 | 110 | 134 | 196 | 98 | 166 |  |

u1 + v1 = 3

u1 + v6 = 4

u2 + v2 = 3

u2 + v4 = 8

u3 + v1 = 4

u3 + v5 = 5

u4 + v4 = 6

u4 + v6 = 5

u5 + v4 = 3

u6 + v3 = 0

u6 + v4 = 0

Примем u1 = 0, тогда:

u1 = 0 u2 = 3 u3 = 1 u4 = 1 u5 = -2 u6 = -5

v1 = 3 v2 = 0 v3 = 5 v4 = 5 v5 = 4 v6 = 4

x12 = u1 + v2 – c12 = 0 + 0 – 5 = -5

x13 = u1 + v3 – c13 = 0 + 5 – 9 = -4

x14 = u1 + v4 – c14 = 0 + 5 – 6 = -1

x15 = u1 + v5 – c15 = 0 + 4 – 14 = -10

x21 = u2 + v1 – c21 = 3 + 3 – 13 = -7

x23 = u2 + v3 – c23 = 3 + 5 – 8 = 0

x25 = u2 + v5 – c25 = 3 + 4 – 10 = -3

x26 = u2 + v6 – c26 = 3 + 4 – 16 = -9

x32 = u3 + v2 – c32 = 1 + 0 – 8 = -7

x33 = u3 + v3 – c33 = 1 + 5 – 14 = -8

x34 = u3 + v4 – c34 = 1 + 5 – 11 = -5

x36 = u3 + v6 – c36 = 1 + 4 – 14 = -9

x41 = u4 + v1 – c41 = 1 + 3 – 7 = -3

x42 = u4 + v2 – c42 = 1 + 0 – 13 -12

x43 = u4 + v3 – c43 = 1 + 5 – 13 = -7

x45 = u4 + v5 – c45 = 1 + 4 – 16 = -11

x51 = u5 + v1 – c51 = -2 + 3 – 6 = -5

x52 = u5 + v2 – c52 = -2 + 0 – 14 = -16

x53 = u5 + v3 – c53 = -2 + 5 – 12 = -9

x55 = u5 + v5 – c55 = -2 + 4 – 13 = -11

x56 = u5 + v6 – c56 = -2 + 4 – 7 = -5

x61 = u6 + v1 – c61 = -5 + 3 = -2

x62 = u6 + v2 – c62 = -5 + 0 = -5

x65 = u6 + v5 – c65 = -5 + 4 = -1

x66 = u6 + v6 – c66 = -5 + 4 = -1

Так, как все значения вышли отрицательными, то смело можно заявить, что составленный план является оптимальным.

Допустимое решение

x11 = 3; x16 = 4; x22 = 3; x24 = 8; x31 = 4; x35 = 5; x44 = 6; x46 = 5; x54 = 3; x63 = 0; x64 = 0

Значение функции цели

**Вывод:** в ходе данной лабораторной работы я изучил принципы транспортных задач, этапы их решения и методы решения, которые используются на этих этапах. Методы: поиск наименьшей стоимости, а также метод потенциалов. Работу этих методов я продемонстрировал, решив заданную мне транспортную задачу.

Учреждение образования

«Белорусский государственный технологический университет»

Кафедра ИСИТ

**«Математическое программирование»**

**Лабораторная работа №6**

**АЛГОРИТМЫ НА ГРАФАХ**

Выполнил:

Студент 2 курса 7 группы ФИТ

Воликов Дмитрий Анатольевич

**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 ДЛЯ ВЫПОЛНЕНИЯ**

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

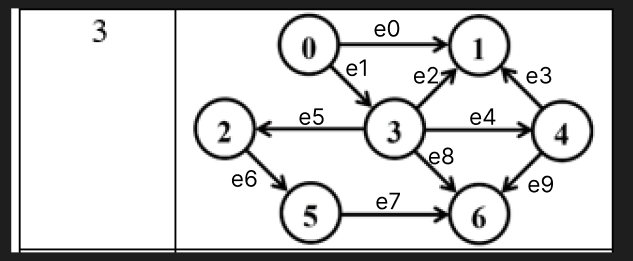


Рисунок 1 – Ориентированный граф

Матрица смежности:

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

Матрица инцидентности:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| МИнц | e0 | e1 | e2 | e3 | e4 | e5 | e6 | e7 | e8 | e9 |
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | -1 | 0 | -1 | -1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | -1 | 1 | 0 | 0 | 0 |
| 3 | 0 | -1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 4 | 0 | 0 | 0 | 1 | -1 | 0 | 0 | 0 | 0 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | -1 | -1 | -1 |

Список смежных вершин:

1. 1, 3
2. –
3. 5
4. 1, 2, 4, 6
5. 1, 6
6. 6
7. –

**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 глубину в исходном ориентированном графе по шагам:

1. 0 –> 1, а так, как вершина 1 не имеет путей к другим вершинам, то возвращаемся к вершине 0.
2. 0 –> 3, так, как вершина 1 уже была просмотрена, то пойдём в вершину 4
3. 3 –> 4, так, как вершина 1 уже была просмотрена, то пойдём в вершину 6
4. 4 –> 6, так, как вершина 6 не имеет путей к другим вершинам, то возвращаемся к вершине 3.
5. 3 –> 2
6. 2 –> 5
7. 5
8. Все вершины отмечены просмотренными, значит поиск в глубину по графу закончился следующим циклом: 0 –> 1 –> 3 –> 4 –> 6 –> 2 –> 5.

Алгоритм поиска в ширину в исходном ориентированном графе по шагам:

1. 0 –> 1, вершину 1 помечаем пройденной. Несложно заметить, что у неё нет других путей, значит в эту вершину возвращаться мы не будем
2. 0 –> 3, вершины 0 и 3 отмечаются как помеченными. Далее возможные выходы из вершины 3 – это 1, 4, 6, 2. Но вершину 1 мы откидываем, так как она уже была пройдена
3. 3 –> 2, вершину 2 помечаем пройденной
4. 3 –> 4, вершину 4 помечаем пройденной
5. 3 –> 6, вершину 6 помечаем пройденной
6. Далее из вершин 2, 4, 6 делаем проход, но так, как вершина 4 ведёт к уже помеченным вершина 1 и 6, её мы не трогаем. Вершина 6 никуда не ведёт, поэтому её мы тоже не трогаем, а значит остаётся вершина 5.
7. 2 –> 5, вершину 5 помечаем пройденной.
8. Из вершины 5 есть путь до вершины 6, которая уже отмечена пройденной, значит больше путей нет
9. Граф пройден следующим циклом: 0 –> 1 –> 3 –> 2 –> 4 –> 6 –> 5.

Алгоритм топологической сортировки в исходном ориентированном графе по шагам:

1. Выбираем вершину, которая не имеет входных дуг. В нашем случае – вершина 0. Удаляем её из графа и переносим в граф решения.
2. Далее выбираем вершину 3, удаляем её из графа и переносим в граф решения.
3. Далее выбираем вершину 2, и переносим её в граф решения.
4. Выбираем вершину 5, удаляем из графа и переносим в граф решения.
5. Выбираем вершину 4, удаляем из графа и переносим в граф решения.
6. Выбираем вершину 6, удаляем из графа и переносим в граф решения.
7. Выбираем вершину 1, удаляем из графа и переносим в граф решения.
8. Теперь расставляем дуги таким образом, чтобы меньшее приходило к большему, но не наоборот.
9. В результате у нас вышел следующий граф:

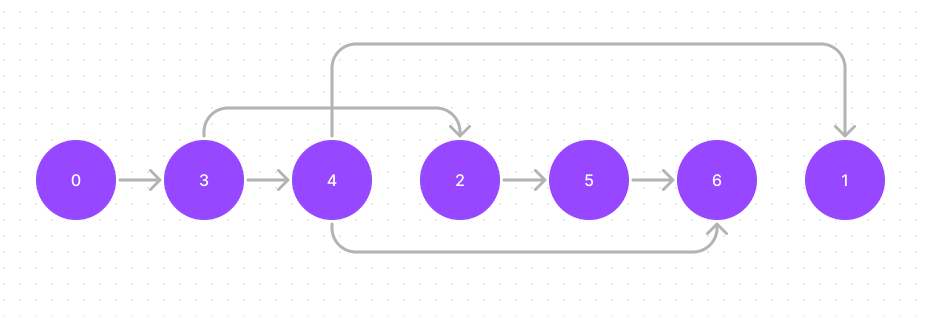


Рисунок 2 – отсортированный граф

**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.

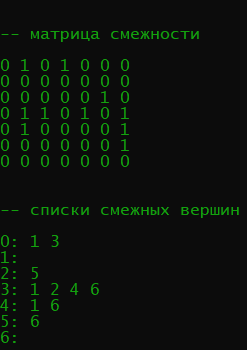


Рисунок 3 – Подсчёт смежных вершин для исходного графа



Рисунок 4 –Расчёт поиска в ширину исходного графа

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

#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 1 – DFS.h

#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 2 – DFS.cpp



Рисунок 5 – Поиск в глубину исходного графа

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

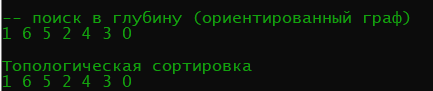


Рисунок 6 – Топологическая сортировка для исходного графа

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

Веса ребер принять:

W(e0,1)=8,

W(e0,3)=2,

W(e3,1)=4,

W(e4,1)=3,

W(e3,2)=9,

W(e2,5)=11,

W(e3,4)=1,

W(e4,6)=10,

W(e5,6)=2,

W(e3,6)=9.

В результате после преобразования ориентированного графа в связный неориентированный, а также добавления весов рёбер, получим новый граф:

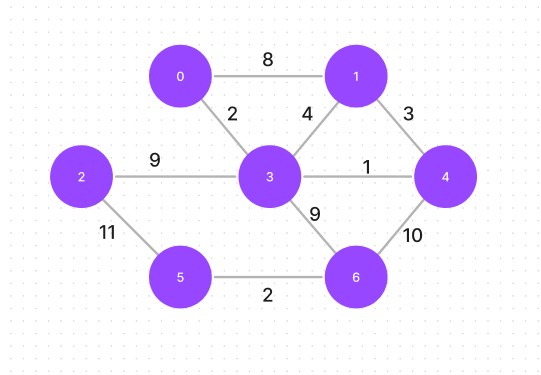


Рисунок 7 – Новый исходный граф

Начнём поиск остовного дерева с 3-ей вершины. Добавим к нашему дереву 4-ую вершину, ибо вес ребра W(e34) равен 1. Добавим вершину 1 к нашему дереву, по ребру W(e41), у которого вес равен 3. Также к дереву присоединиться вершина 0 по ребру W(e03), вес которого равен 2. Добавим вершину 2, по ребру W(e32) с весом 9. Следующей вершиной будет 6 по ребру W(e36) с весом 9. Окончательно добавим вершину 5 с ребром весом в 2. В итоге у нас получилось следующее остовное дерево:

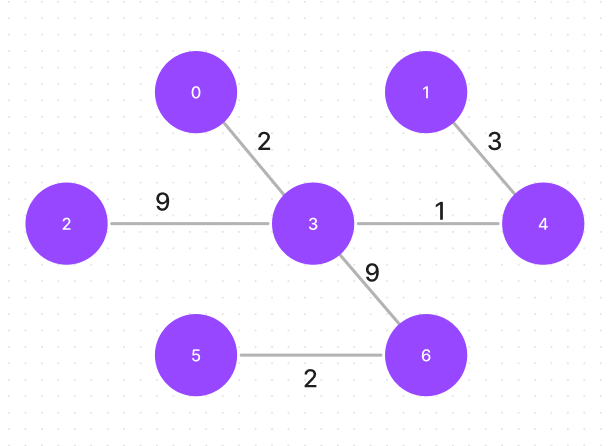


Рисунок 8 – Остовное дерево по алгоритму Прима

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

Исходный граф с заданными значениями веса для рёбер берём из прошлого задания:

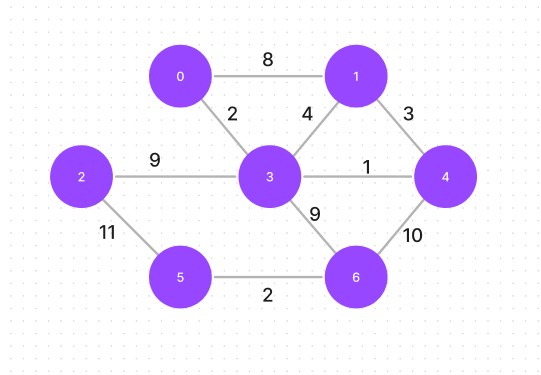


Рисунок 9 – Исходный граф для расчёта остовного дерева по алгоритму Крускала

Рассматриваем рёбра с самым минимальным весом, в нашем случае – e34. Так, как оно единственное и вообще самое первое для добавления в остовное дерево, то спокойно можем добавить вершины 3 и 4 в остов.

Следующие рёбра с самым минимальным весом – e03 и e56. Так, как добавление этих рёбер не приводит к появлению цикла в остове, тогда спокойно можем их добавлять.

На данный момент самым ребром с самым минимальным весом является ребро e14. Добавление его в остов не приводит к появлению цикла, оставляем.

Рёбра e31 и e01 нельзя добавлять в остов, так как они приводят к появлению цикла в нашем остовном дереве.

Теперь самыми «лёгкими» рёбрами являются e23 и e36. Добавление их не приводит к появлению циклов, а значит добавляем их к остовному дереву.

Так, как все вершины на данный момент входят в остовное дерево, а также у нас нет каких-либо циклов, то смело можно подытожить, что остовное дерево мы нашли правильно.

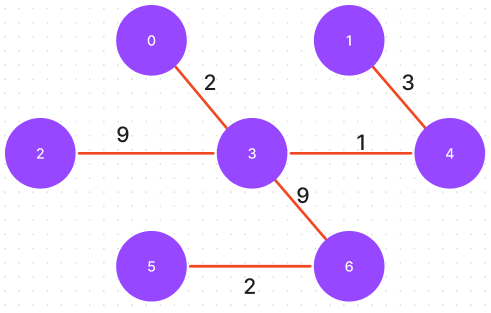


Рисунок 10 – Результат вычисления остовного дерева по алгоритму Крускала

**Вывод**: в ходе данной лабораторной работы были рассмотрены следующие операции над графом: расчёт матриц смежности и инцидентности, списка смежных вершин. Задачи решены как вручную, так и с помощью программы, расчёты сошлись. Были разобраны алгоритмы Прима и Крускала для расчёта остовного дерева.

Учреждение образования

«Белорусский государственный технологический университет»

Кафедра ИСИТ

**«Математическое программирование»**

**Лабораторная работа №7**

**Сетевые модели**

Выполнил:

Студент 2 курса 7 группы ФИТ

Воликов Дмитрий Анатольевич

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

**Задание для выполнения:**

Лабораторная работа базируется на исследовании различных тематик в проектировании программных продуктов, составлении сетевых графиков для разных тем, нахождении критических путей в составленных графиках. Каждый проект принять условным или обобщенным, но допустимо делать упор на конкретные примеры.



Рисунок 1 – Вариант задания

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

Подумайте и выделите в проекте, согласно вашему варианту не менее 4 этапов работ. Также разбейте полученные этапы на задачи, их количество в совокупности по этапам должно быть не менее 12. Пример оформления задания смотрите в приложении ниже и в лекционном материале по теме. Распределите время, отпущенное на ваш проект согласно вариантам, на выделенные вами этапы. Скорректируйте сформулированные вами задачи, если это необходимо.

|  |  |  |  |
| --- | --- | --- | --- |
| Код  операции | Наименование  операции | Предшествующие  операции | Время (дни) |
| **I. Анализ** | | | |
| Z1 | Системный анализ |  | 8 |
| Z2 | Анализ требований | Z1 | 3 |
| **II. Проектирование** | | | |
| Z3 | Проектирование пользовательских интерфейсов | Z2, Z9, Z10 | 3 |
| Z4 | Проектирование базы данных | Z2, Z10, Z11 | 3 |
| Z5 | Проектирование безопасности данных | Z2, Z10, Z11 | 5 |
| **III. Разработка** | | | |
| Z6 | Создание пользовательских интерфейсов | Z3, Z10, Z15 | 5 |
| Z7 | Разработка объектов базы данных | Z4, Z10, Z11, Z14 | 7 |
| Z8 | Настройка методов шифрования и маскирования данных | Z5, Z10, Z11, Z15 | 5 |
| **IV. Тестирование** | | | |
| Z9 | Создание UNIT-тестов | Z6, Z7, Z8 | 2 |
| Z10 | Проведение тестирования | Z6, Z7, Z8 | 5 |
| Z11 | Тесты на проникновение | Z9, Z10, Z8 | 5 |
| **V. Внедрение** | | | |
| Z12 | Разработка документации | Z6, Z7, Z8 | 3 |
| Z13 | Завершение работ | Z10, Z11, Z12 | 3 |
| **VI. Дополнительные требования** | | | |
| Z14 | Выбор и установка образа БД | Z1 | 4 |
| Z15 | Выбор и установка инструментария | Z1 | 4 |

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

Согласно составленному перечню задач и распределённому времени составьте сетевой график вашего проекта. Помните о правилах составления графика и вводите фиктивные операции и операции ожидания если это необходимо.

|  |
| --- |
| При построении сетевых графиков соблюдается ряд правил:   1. в сети не должно быть событий (кроме исходного), в которые не входит ни одна дуга; 2. не должно быть событий (кроме завершающего), из которых не выходит ни одной дуги; 3. сеть не должна содержать замкнутых контуров (циклов); 4. ***любая пара событий сетевого графика может быть соединена не более чем одной дугой;*** 5. ***номер*** начального ***события любой операции должен быть меньше номера ее*** конечного ***события.*** |

Найдите критический путь в составленном вами сетевом графике и обоснуйте его нахождение. Критический путь может быть меньше, чем время, отведенное на выполнение всех задач. Выделите, какие операции принадлежат критическому пути.

|  |  |  |  |
| --- | --- | --- | --- |
| Начальное событие | Код операции | Предшествующие операции | Конечное событие |
| 0 | Z1 |  | 1 |
| 2 | Z2 | Z1 | 3 |
| 4 | Z3 | Z2, Z9, Z10 | 5 |
| 6 | Z4 | Z2, Z10, Z11 | 7 |
| 8 | Z5 | Z2, Z10, Z11 | 9 |
| 10 | Z6 | Z3, Z10, Z15 | 11 |
| 12 | Z7 | Z4, Z10, Z11, Z14 | 13 |
| 14 | Z8 | Z5, Z10, Z11, Z15 | 15 |
| 16 | Z9 | Z6, Z7, Z8 | 17 |
| 18 | Z10 | Z6, Z7, Z8 | 19 |
| 20 | Z11 | Z8, Z9, Z10 | 21 |
| 22 | Z12 | Z6, Z7, Z8 | 23 |
| 24 | Z13 | Z10, Z11, Z12 | 25 |
| 26 | Z14 | Z1 | 27 |
| 28 | Z15 | Z1 | 29 |

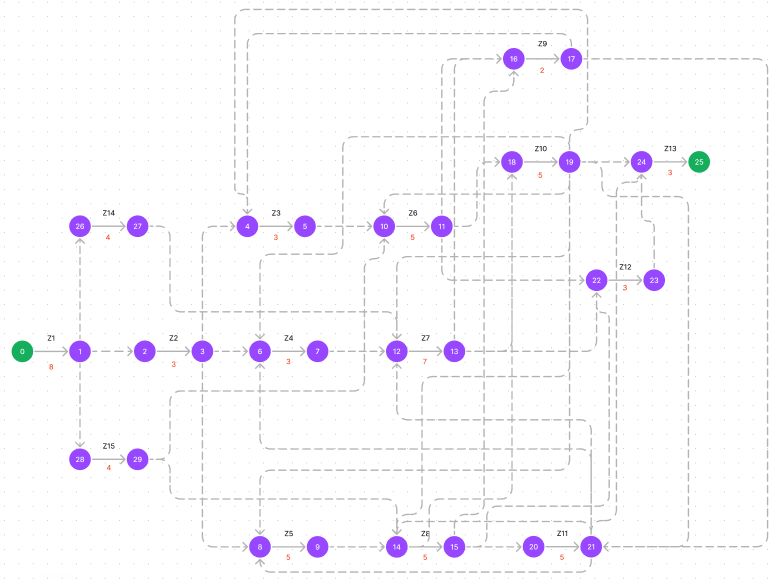


Рисунок 2 – Сетевой график

**Нахождение критического пути**: будем идти с начального узла к конечному только по самым дорогим для нас путям. Алгоритм критического пути выглядит следующим образом:

Z1 – Z14 – Z7 – Z9 – Z11 – Z5 – Z8 – Z12 – Z13, что даст нам следующее значение: 8 + 4 + 7 + 2 + 5 + 5 + 5 + 3 + 3 = 42 дня.

**Вывод**: в ходе этой лабораторной работы мною была изучена тема «Сетевые модели». На примере разработки банковского приложения я составил этапы выполнения задачи, разбил на подзадачи. К каждой задаче установил некоторые предшествующие задачи. В итоге всего был построен график сетевой модели, а также найден критический путь, значение которого равно 42 дням.

Учреждение образования

«Белорусский государственный технологический университет»

Кафедра ИСИТ

**«Математическое программирование»**

**Лабораторная работа №8**

**Графический метод решения оптимизационных задач**

Выполнил:

Студент 2 курса 7 группы ФИТ

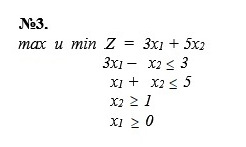
Воликов Дмитрий Анатольевич

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

**Задание для выполнения**:

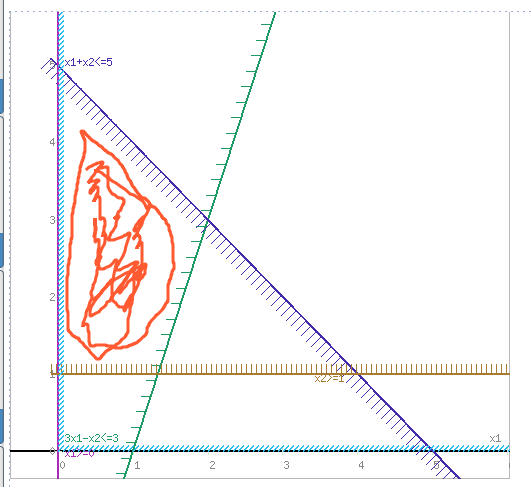
Задание рассчитано на повторение пройденного материала.

Номера задач принять за варианты – 1,11 вариант – задача №1, 2,12 вариант и так далее.



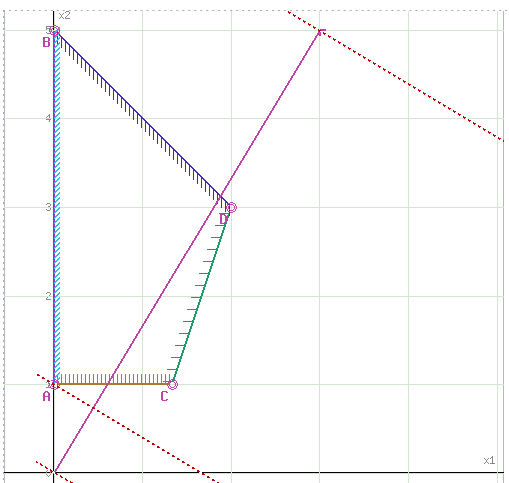
**Задания по шагам**:

**Шаг №1**. Построим область допустимых решений, т.е. решим графически систему неравенств. Для этого построим каждую прямую и определим полуплоскости, заданные неравенствами (полуплоскости обозначены штрихом).



Закрашенная красным цветом область ­– есть область наших допустимых значений, а значит искать решения надо в ней.

**Шаг №2**. Построим прямую, отвечающую значению функции F = 3x1+5x2 = 0. Вектор-градиент, составленный из коэффициентов целевой функции, указывает направление максимизации F(X). Начало вектора – точка (0; 0), конец – точка (3;5). 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.

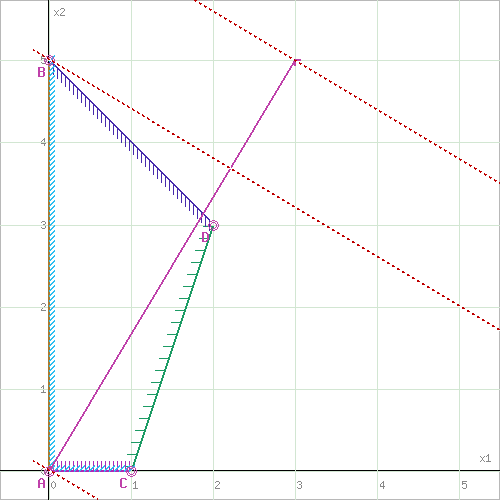


Прямая пересекает область в точке A. Так как точка A получена в результате пересечения прямых **(4)** и **(5)**, то ее координаты удовлетворяют уравнениям этих прямых:

x2=1, x1=0

Решив систему уравнений, получим: x1 = 0, x2 = 1  
Откуда найдем минимальное значение целевой функции:  
F(x) = 3\*0 + 5\*1 = 5

**Шаг №3**. Сделаем всё тоже самое, как во втором пункте, но только для максимального значения. Но так, как нас интересует максимальное значение, то будем двигать вектор градиент до последнего касания с раскрашенной областью.



Прямая пересекает область в точке B. Так как точка B получена в результате пересечения прямых **(2)** и **(4)**, то ее координаты удовлетворяют уравнениям этих прямых:

x1+x2=5, x1=0

Решив систему уравнений, получим: x1 = 0, x2 = 5  
Откуда найдем максимальное значение целевой функции:  
F(x) = 3\*0 + 5\*5 = 25

Вывод: данная лабораторная работа была самой легкой среди всех остальных и не составляла труда посчитать значения экстремумов для уравнения, которому были заданы ограничения. Минимальным значением уравнения является 5, максимальным – 25.