**Отчёт по лабораторным работам**

**Выполнил:**

**Халалеенко Андрей Николаевич**

**Студент 2 курса 3 группы**

**Лабораторная работа 1**

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

Важно:

**stdafx.h** - это стандартное название предварительно скомпилированного заголовочного файла в Visual Studio 2017 и более ранних версиях. Начиная с VS2019, вместо этого используется **pch.h**. Его не нужно ниоткуда скачивать, в него вписываются часто включаемые в коде заголовочные файлы для ускорения компиляции. Сам файл должен создаваться автоматически при создании проекта; но если его нет, можно добавить его вручную. Содержимое должно может выглядеть, например, так:

#pragma once

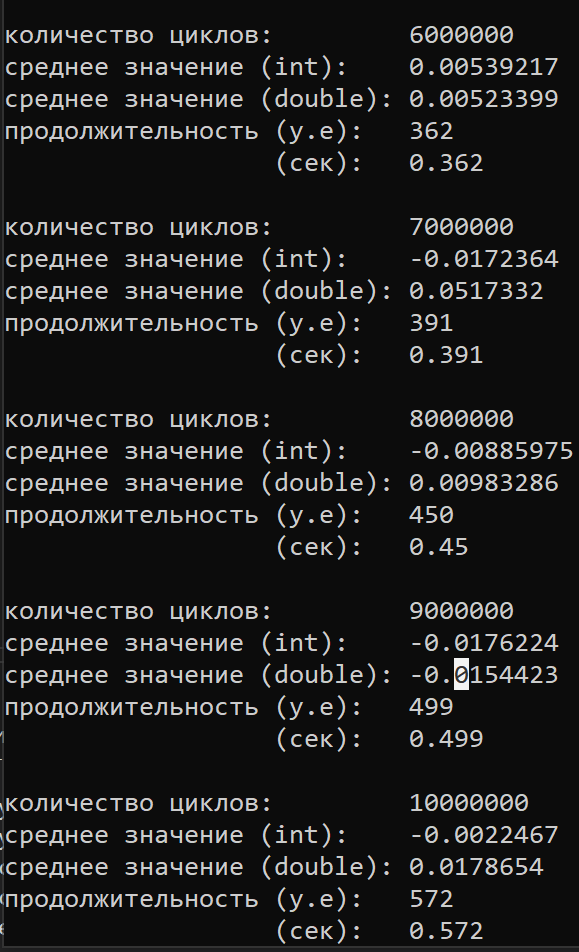
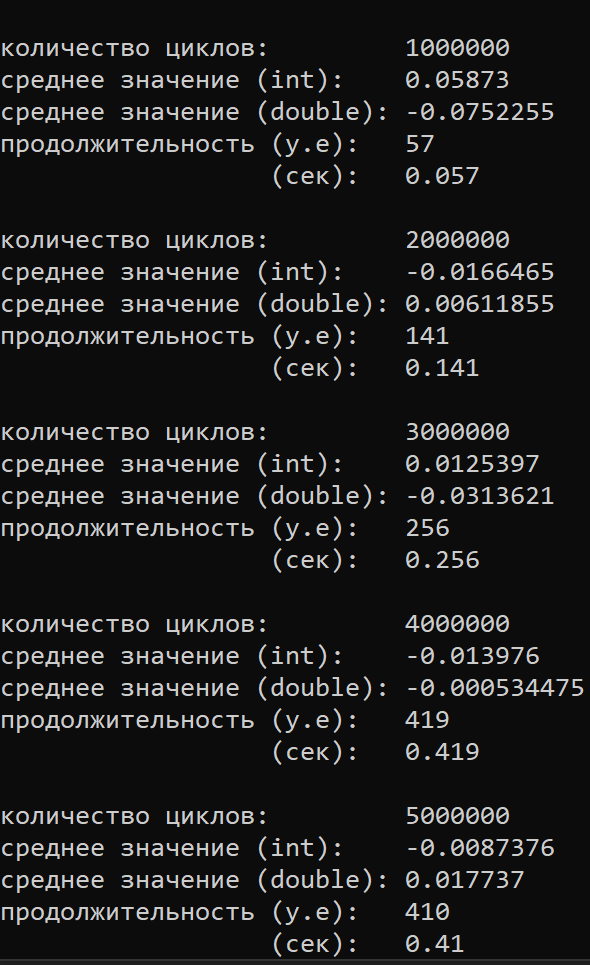
#include <stdio.h>

//другие директивы include...

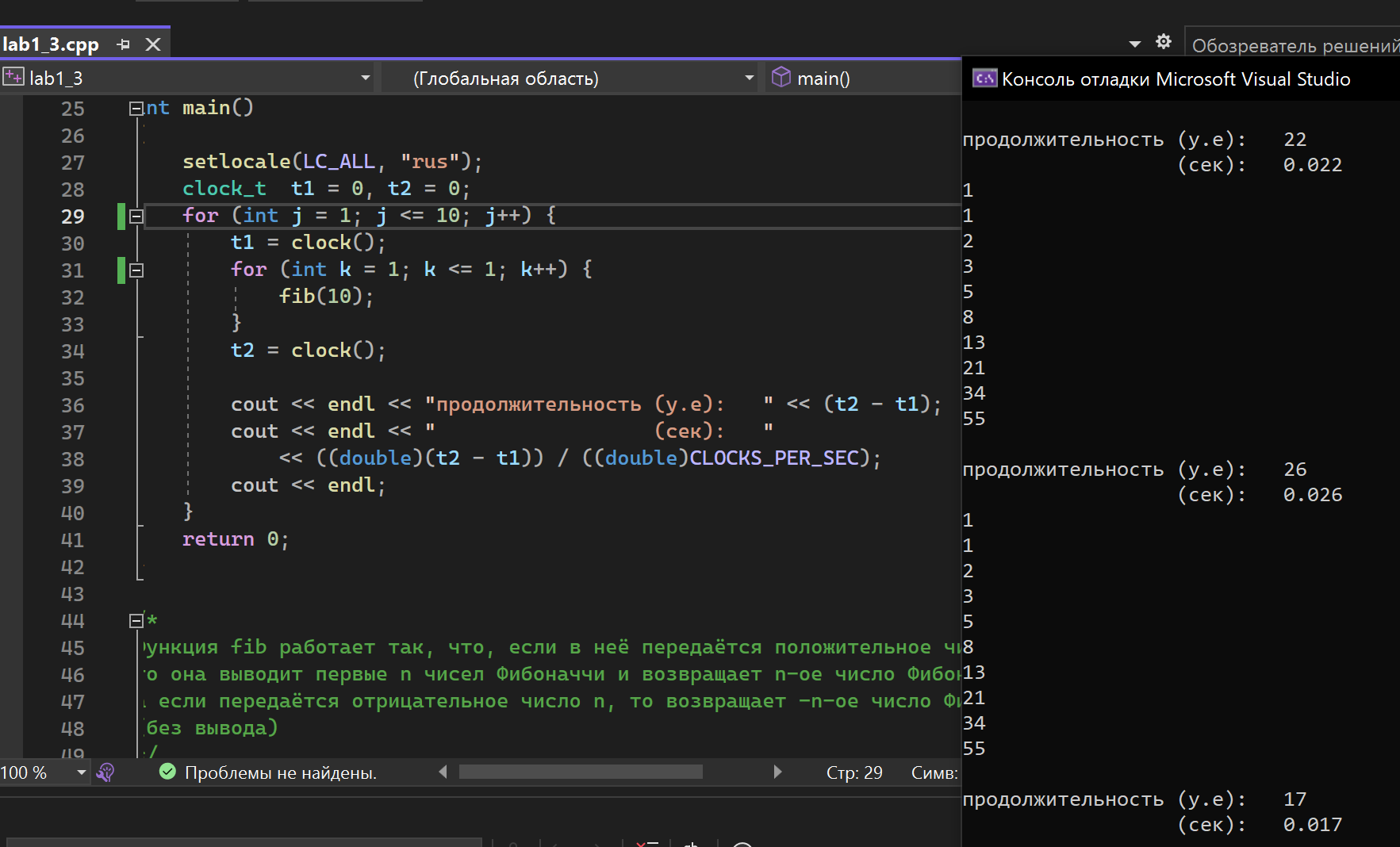
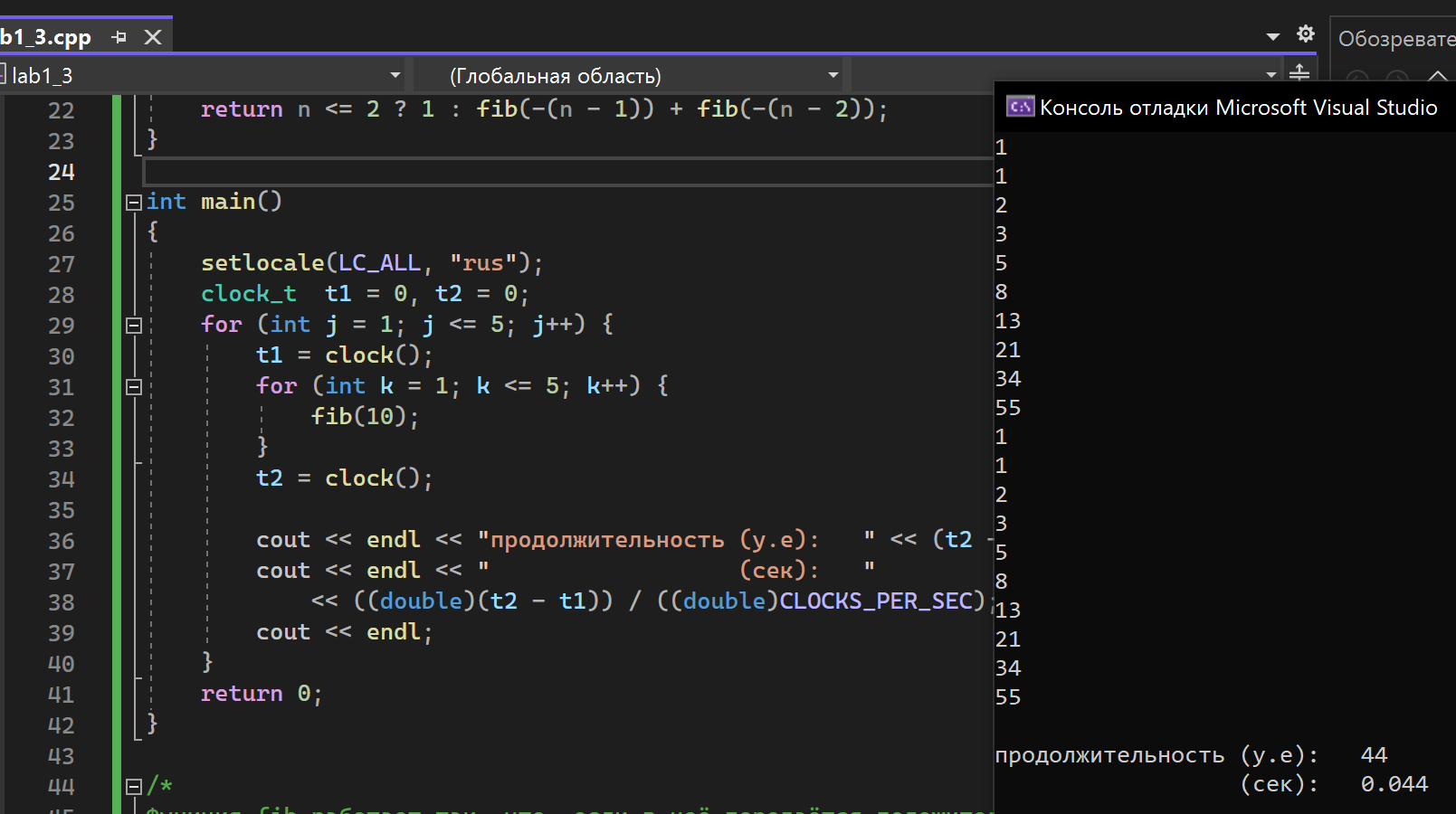
Совместно с ним в проекте должен находится **pch.cpp**, состоящий из единственной строки:

#include "pch.h"

Для использования предварительно скомпилированных заголовков их необходимо включить параметром /Y или через страницу свойств проекта **Configuration Properties > C/C++ > Precompiled Headers**. После этого в каждом cpp-файле в самом начале (до любых директив препроцессора или строк кода) нужно добавить #include "pch.h".

****

Для текущего задания специально была реализована данная работа циклов для нагрузки системы.

****

****

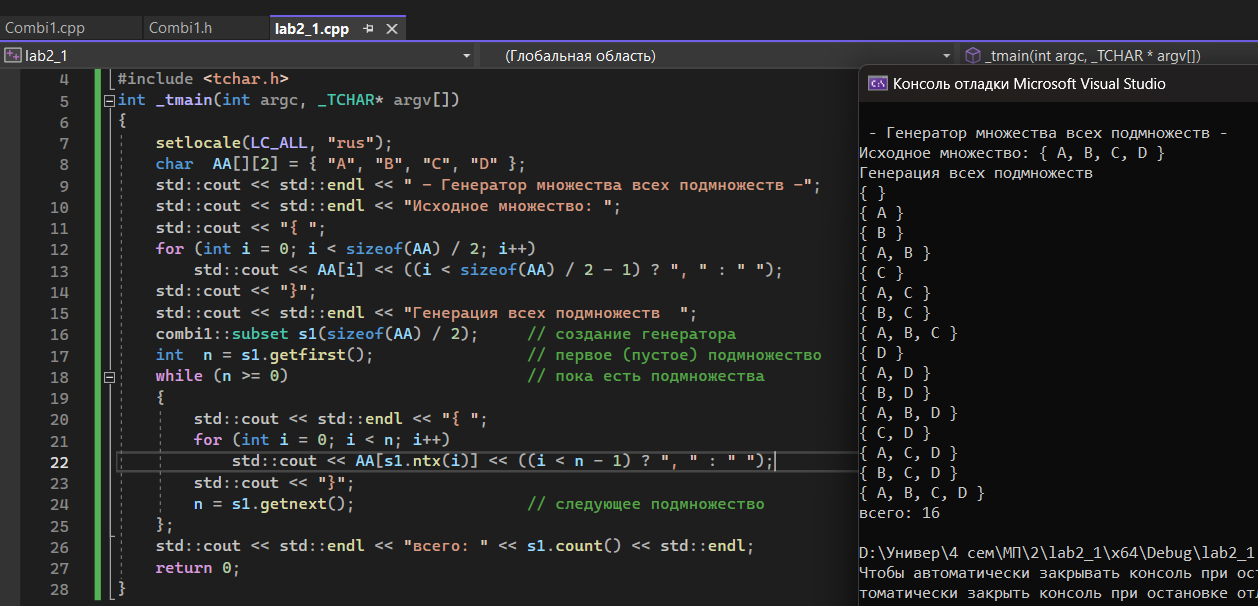
Данная диаграмма отображает графическое представление работоспособности алгоритмов. Как можно заметить, ввиду реализации неоптимизированного цикла, на графике можно заметить низкий показатель работоспособности алгоритма.

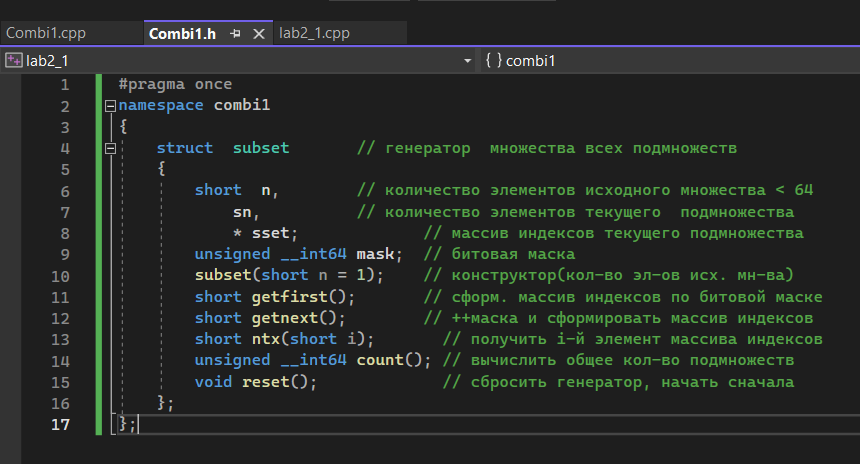
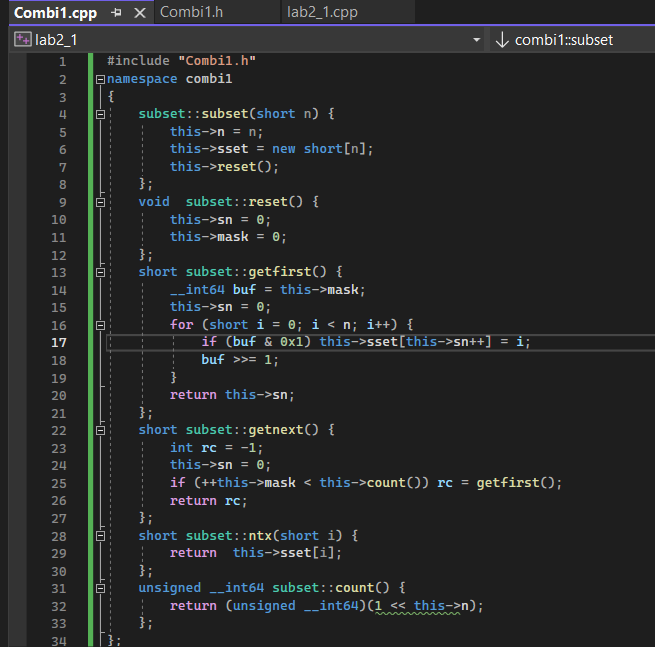
**Вывод**: в данной лабораторной работе были приобретены навыки составления и отладки программ с использованием пользовательских функций для замера продолжительности процесса вычисления и продемонстрированы на графике диаграммы.

**Лабораторная работа 2**

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

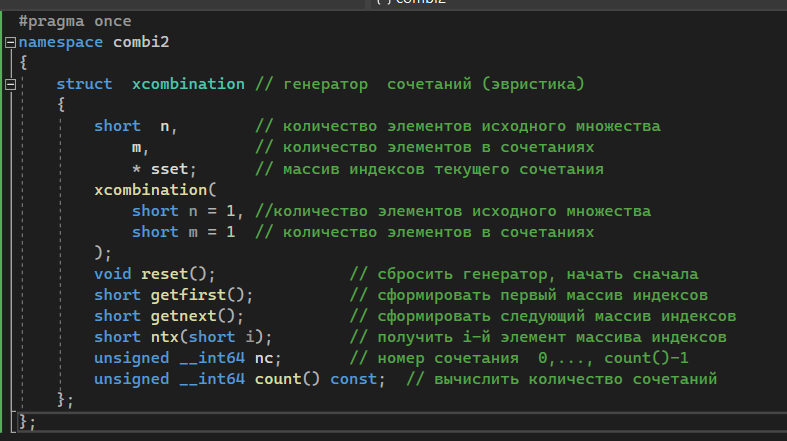
Задание 1:

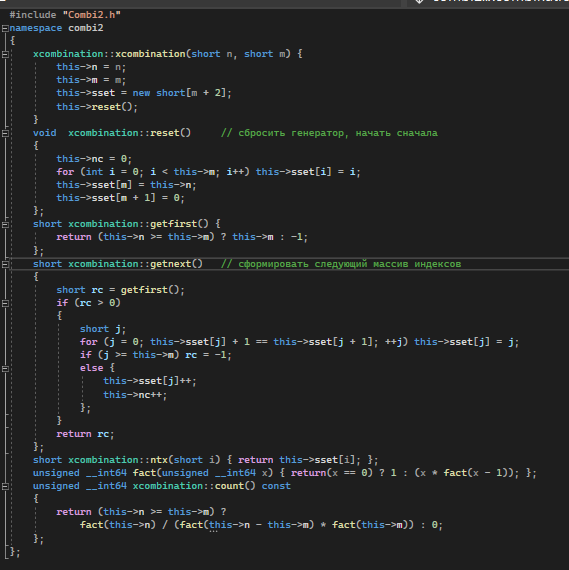


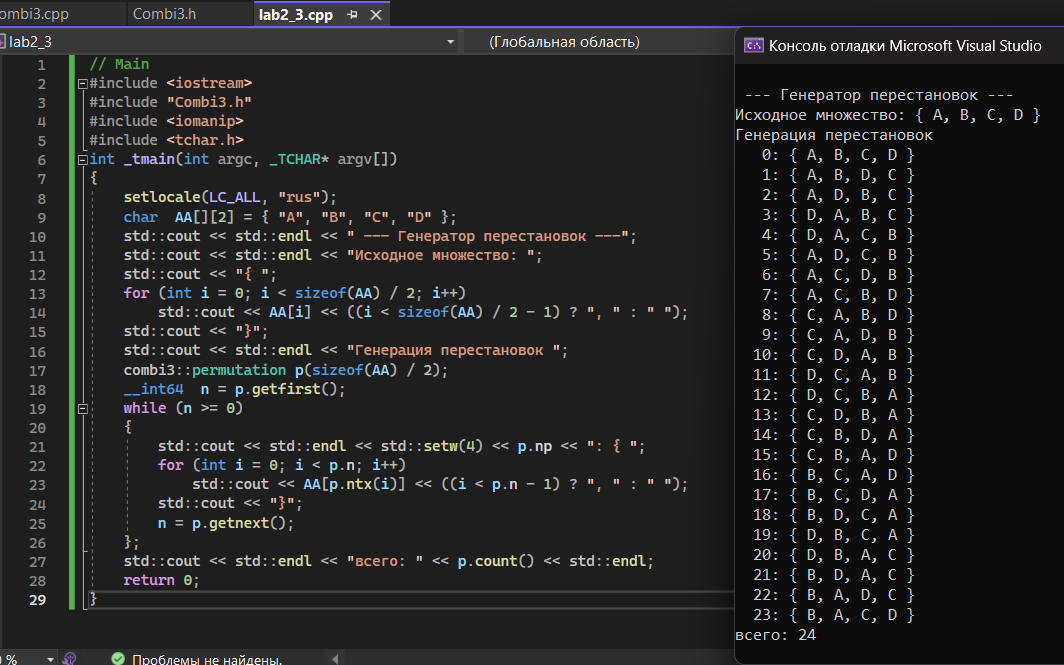
Задание 2:

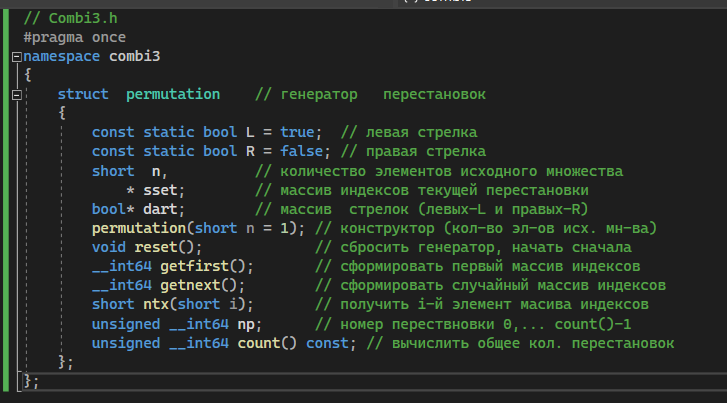


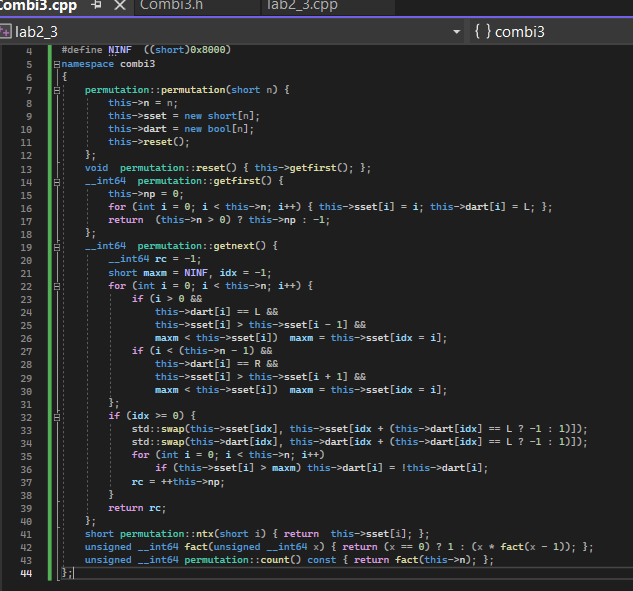


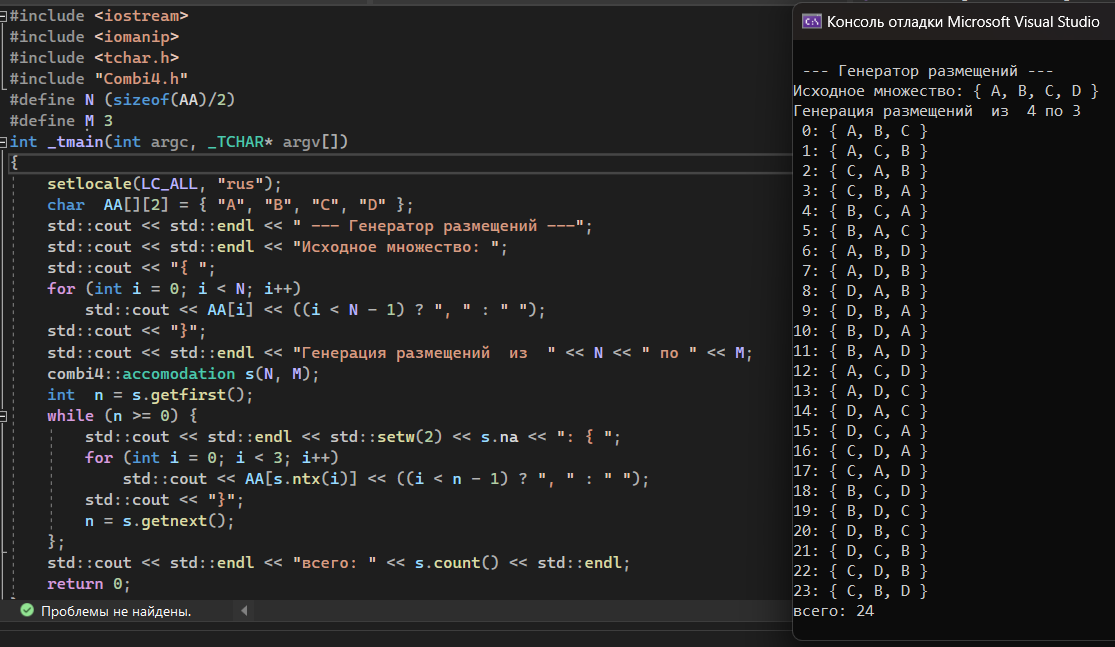


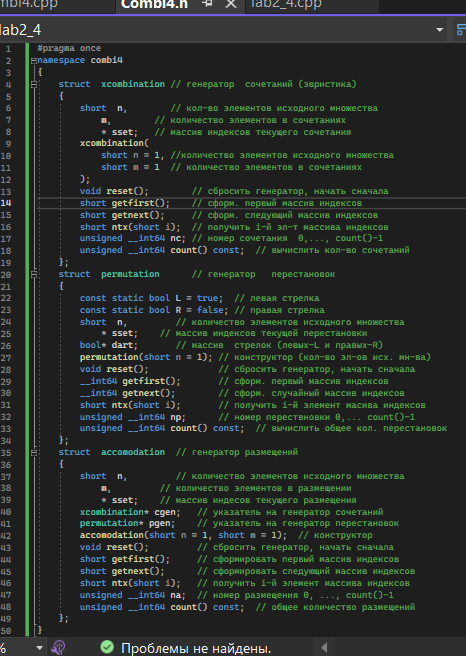
Задание 3:

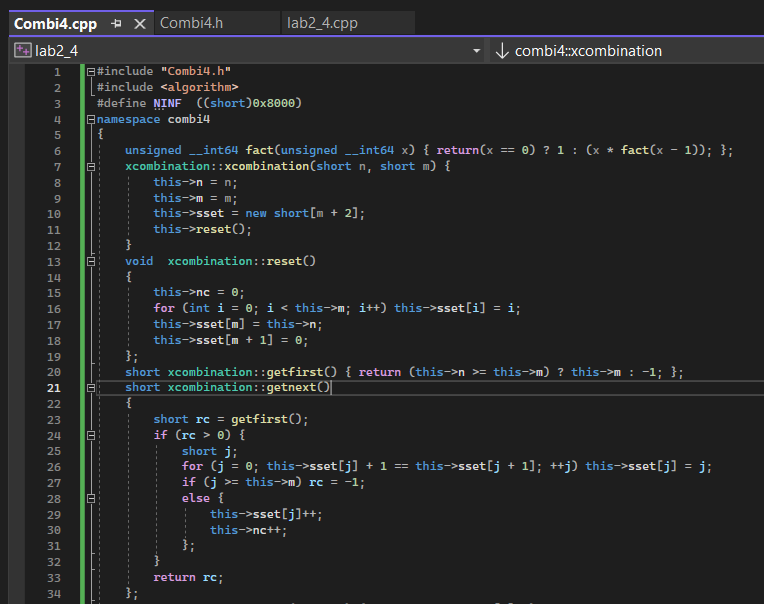
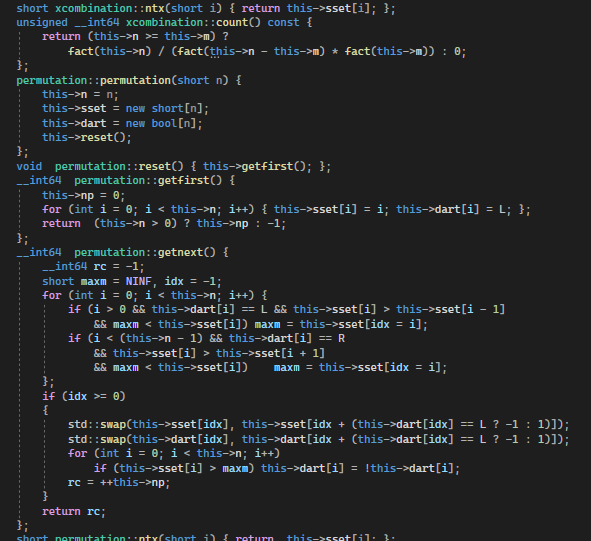
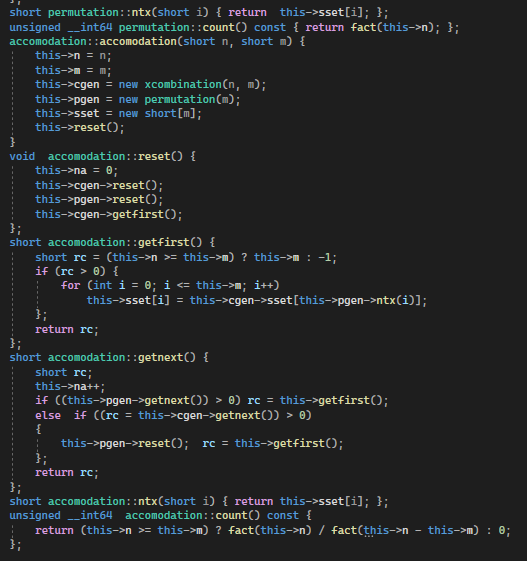






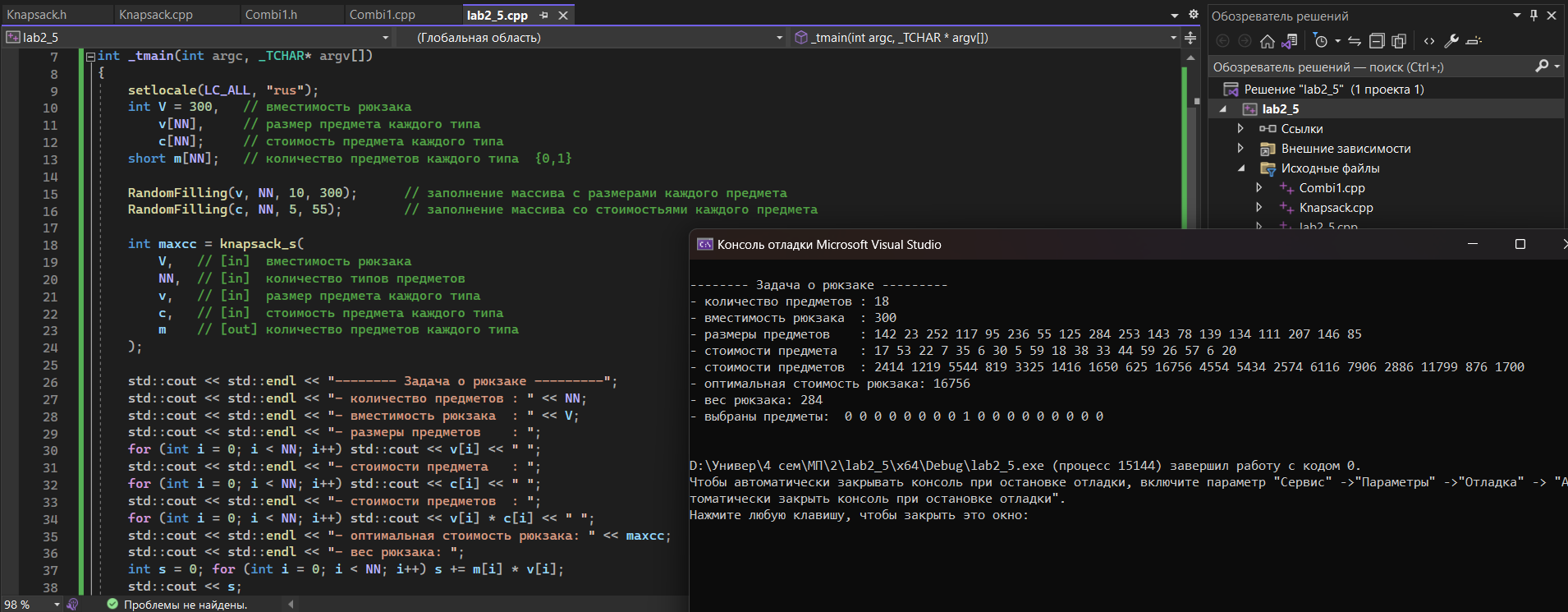
Задание 4: 

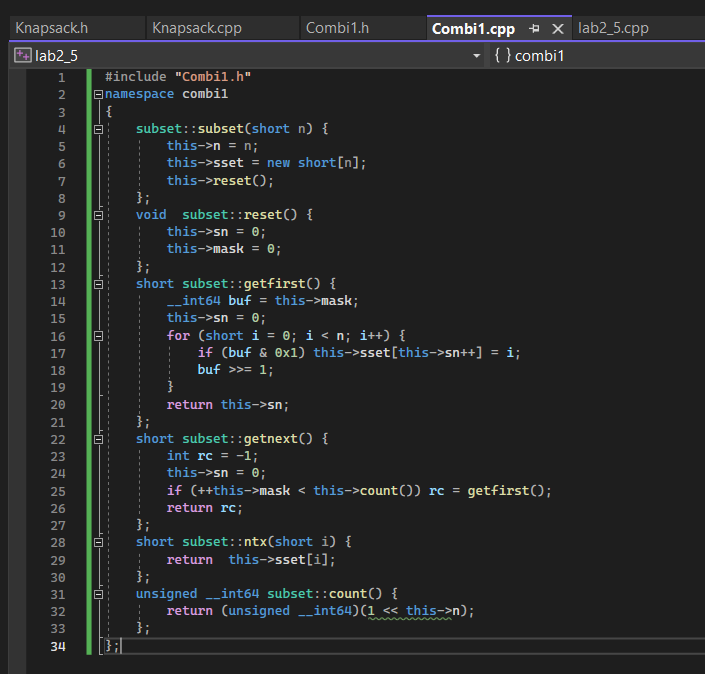


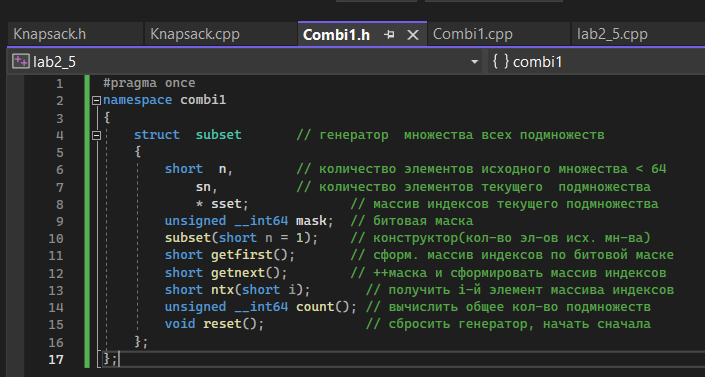
  

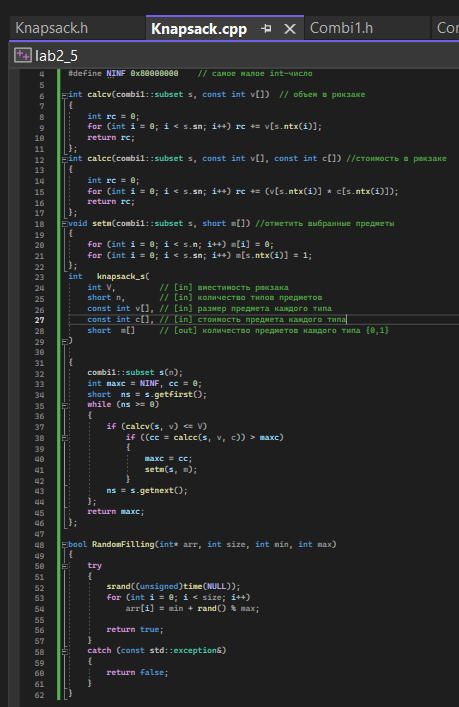
Задание 5:

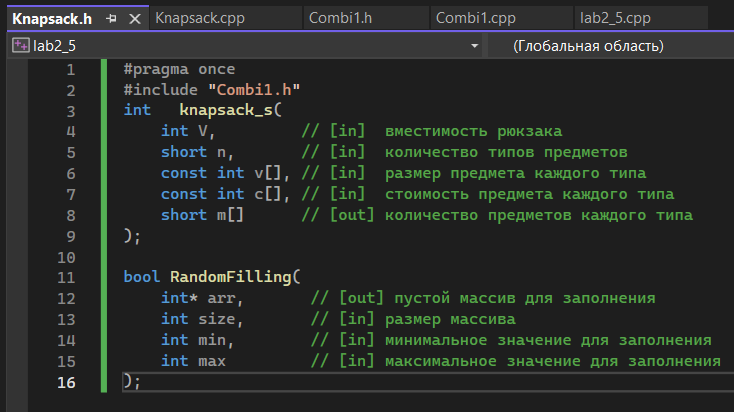
// Упрощенную о рюкзаке (веса предметов и их стоимость сгенерировать случайным образом: вместимость рюкзака 300 кг, веса предметов 10 – 300 кг, стоимость предметов 5 – 55 у.е.;



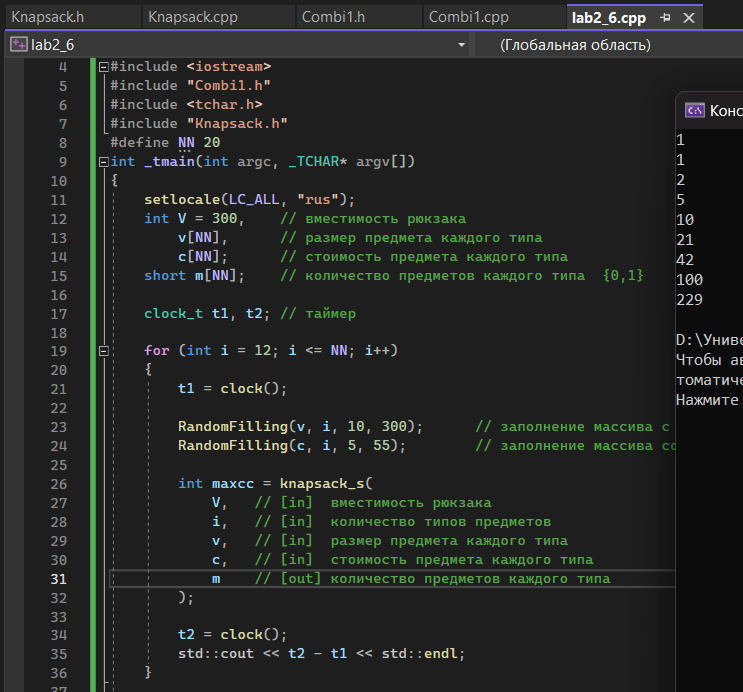


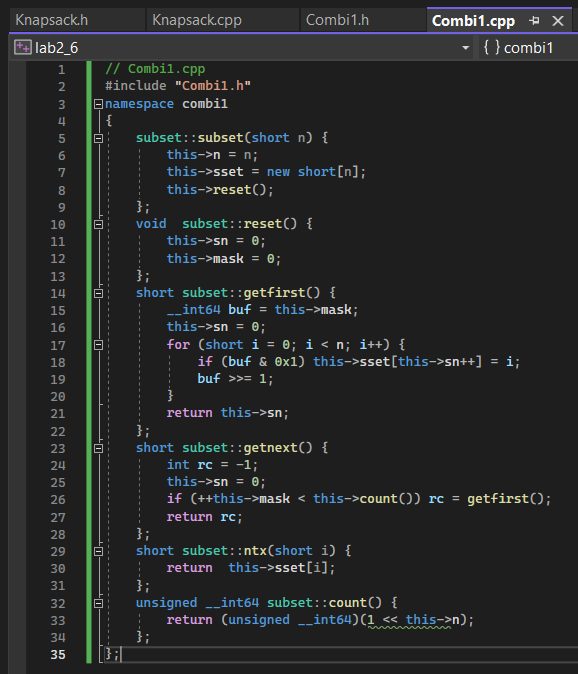


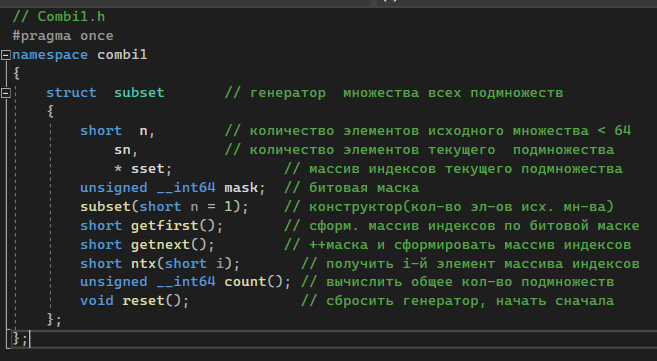


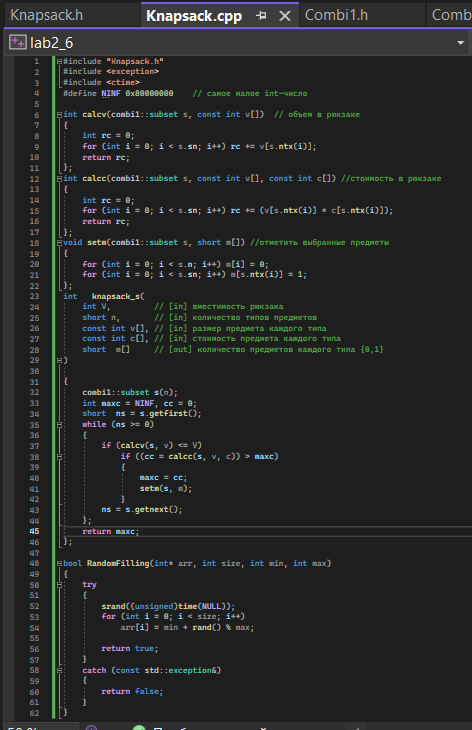


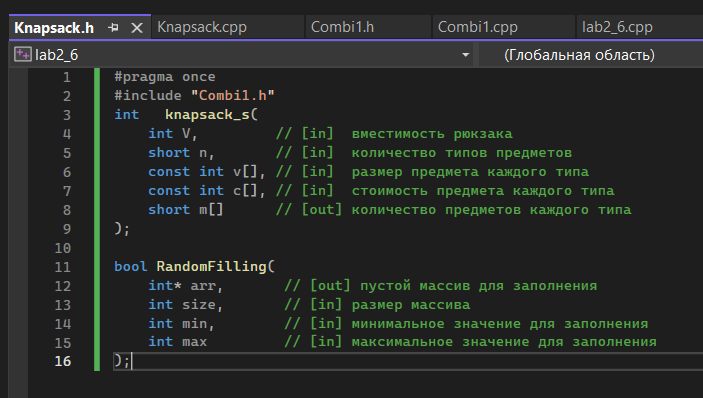
Задание 6:

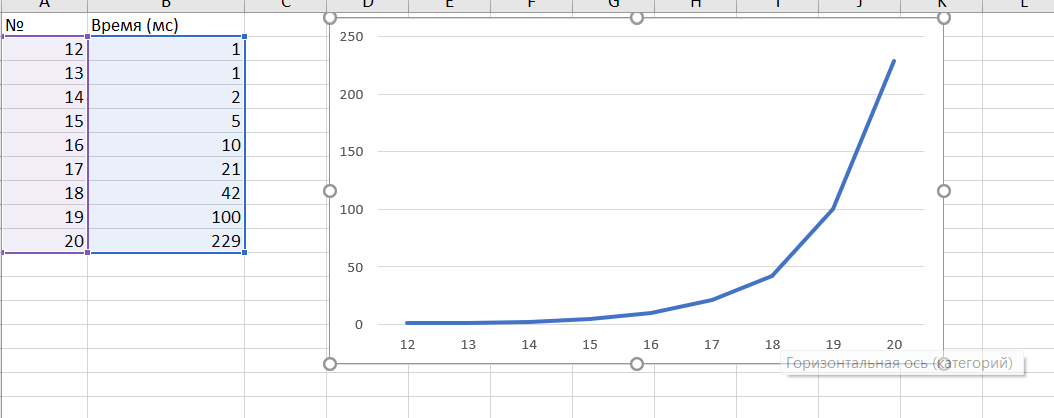












Как видно из графика, время выполнения увеличивается, по началу, с небольшой скоростью. Однако, после выполнения почти десятка операций, время выполнения заметно увеличивается.

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

**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.

**Примечание:** Задания и вопросы со знаком (\*), выполняются в необязательном порядке, но их выполнение поощряется.

Сформулировать условие задачи коммивояжера с параметром. Для этого:

* принять элементы матрицы расстояний равными:

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

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

**Ход решения:**

Имеем 5 городов, построим матрицу расстояний между городами:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INF | 20 | 31 | INF | 10 |
| 10 | INF | 25 | 58 | 74 |
| 12 | 30 | INF | 86 | 59 |
| 27 | 48 | 40 | INF | 30 |
| 83 | 76 | 52 | 23 | INF |

Возьмем в качестве произвольного маршрута:  
X0 = (1,2);(2,3);(3,4);(4,5);(5,1)  
Тогда F(X0) = 20 + 25 + 86 + 30 + 83 = 244  
Для определения нижней границы множества воспользуемся **операцией редукции** или приведения матрицы по строкам, для чего необходимо в каждой строке матрицы D найти минимальный элемент.  
di = min(j) dij

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| INF | 20 | 31 | INF | 10 | 10 |
| 10 | INF | 25 | 58 | 74 | 10 |
| 12 | 30 | INF | 86 | 59 | 12 |
| 27 | 48 | 40 | INF | 30 | 27 |
| 83 | 76 | 52 | 23 | INF | 23 |

Затем вычитаем di из элементов рассматриваемой строки. В связи с этим во вновь полученной матрице в каждой строке будет как минимум один ноль.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INF | 10 | 21 | INF | 0 |
| 0 | INF | 15 | 48 | 64 |
| 0 | 18 | INF | 74 | 47 |
| 0 | 21 | 13 | INF | 3 |
| 60 | 53 | 29 | 0 | INF |

Такую же операцию редукции проводим по столбцам, для чего в каждом столбце находим минимальный элемент:  
dj = min(i) dij

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INF | 10 | 21 | INF | 0 |
| 0 | INF | 15 | 48 | 64 |
| 0 | 18 | INF | 74 | 47 |
| 0 | 21 | 13 | INF | 3 |
| 60 | 53 | 29 | 0 | INF |
| 0 | 10 | 13 | 0 | 0 |

После вычитания минимальных элементов получаем полностью редуцированную матрицу, где величины di и dj называются **константами приведения**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INF | 0 | 8 | INF | 0 |
| 0 | INF | 2 | 48 | 64 |
| 0 | 8 | INF | 74 | 47 |
| 0 | 11 | 0 | INF | 3 |
| 60 | 43 | 16 | 0 | INF |

Сумма констант приведения определяет нижнюю границу H:  
H = ∑di + ∑dj  
H = 10+10+12+27+23+0+10+13+0+0 =105 (**корневая вершина**)  
Для всех клеток матрицы с нулевыми элементами заменяем поочередно нули на INF(бесконечность) и определяем для них сумму образовавшихся констант приведения, они приведены в скобках.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| INF | 0(8) | 8 | INF | 0(3) | 0 |
| 0(2) | INF | 2 | 48 | 64 | 2 |
| 0(8) | 8 | INF | 74 | 47 | 8 |
| 0(0) | 11 | 0(2) | INF | 3 | 0 |
| 60 | 43 | 16 | 0(64) | INF | 16 |
| 0 | 8 | 2 | 48 | 3 |  |

**Исключение ребра** (5,4) проводим путем замены элемента d54 = 0 на INF, после чего осуществляем очередное приведение матрицы расстояний для образовавшегося подмножества (5\*,4\*), в результате получим редуцированную матрицу.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| INF | 0 | 8 | INF | 0 | 0 |
| 0 | INF | 2 | 48 | 64 | 0 |
| 0 | 8 | INF | 74 | 47 | 0 |
| 0 | 11 | 0 | INF | 3 | 0 |
| 60 | 43 | 16 | INF | INF | 16 |
| 0 | 0 | 0 | 48 | 0 | 64 |

Нижняя граница гамильтоновых циклов этого подмножества:  
H(5\*,4\*) = 105 + 64 = 169

Включение ребра (5,4) проводится путем исключения всех элементов 5-ой строки и 4-го столбца, в которой элемент d45 заменяем на INF, для исключения образования негамильтонова цикла.  
В результате получим другую сокращенную матрицу (4 x 4), которая подлежит операции приведения.  
После операции приведения сокращенная матрица будет иметь вид:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INF | 0 | 8 | 0 | 0 |
| 0 | INF | 2 | 64 | 0 |
| 0 | 8 | INF | 47 | 0 |
| 0 | 11 | 0 | INF | 0 |
| 0 | 0 | 0 | 0 | 0 |

Сумма констант приведения сокращенной матрицы:  
∑di + ∑dj = 0  
Нижняя граница подмножества (5,4) равна:  
H(5,4) = 105 + 0 = 105 ≤ 169  
Поскольку нижняя граница этого подмножества (5,4) меньше, чем подмножества (5\*,4\*), то ребро (5,4) включаем в маршрут с новой границей H = 105

**Определяем ребро ветвления**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INF | 0(8) | 8 | 0(47) | 0 |
| 0(2) | INF | 2 | 64 | 2 |
| 0(8) | 8 | INF | 47 | 8 |
| 0(0) | 11 | 0(2) | INF | 0 |
| 0 | 8 | 2 | 47 | 0 |

Исключаем ребро (1, 5)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| INF | 0 | 8 | INF | 0 |
| 0 | INF | 2 | 64 | 0 |
| 0 | 8 | INF | 47 | 0 |
| 0 | 11 | 0 | INF | 0 |
| 0 | 0 | 0 | 47 | 47 |

Нижняя граница гамильтоновых циклов этого подмножества:  
H(1\*,5\*) = 105 + 47 = 152

**Включение ребра** (1,5) проводится путем исключения всех элементов 1-ой строки и 5-го столбца, в которой элемент d51 заменяем на INF, для исключения образования негамильтонова цикла.  
В результате получим другую сокращенную матрицу (3 x 3), которая подлежит операции приведения.  
После операции приведения сокращенная матрица будет иметь вид:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | INF | 2 | 0 |
| 0 | 8 | INF | 0 |
| 0 | 11 | 0 | 0 |
| 0 | 8 | 0 | 8 |

Нижняя граница подмножества (1,5) равна:  
H(1,5) = 105 + 8 = 113 ≤ 152  
Чтобы исключить подциклы, запретим следующие переходы: (4,1),  
Поскольку нижняя граница этого подмножества (1,5) меньше, чем подмножества (1\*,5\*), то ребро (1,5) включаем в маршрут с новой границей H = 113

**Определяем ребро ветвления**

|  |  |  |  |
| --- | --- | --- | --- |
| 0(2) | INF | 2 | 2 |
| 0(0) | 0(3) | INF | 0 |
| INF | 3 | 0(5) | 3 |
| 0 | 3 | 2 | 0 |

Наибольшая сумма констант приведения равна (3 + 2) = 5 для ребра (4,3).

**Исключаем ребро (4, 3)**

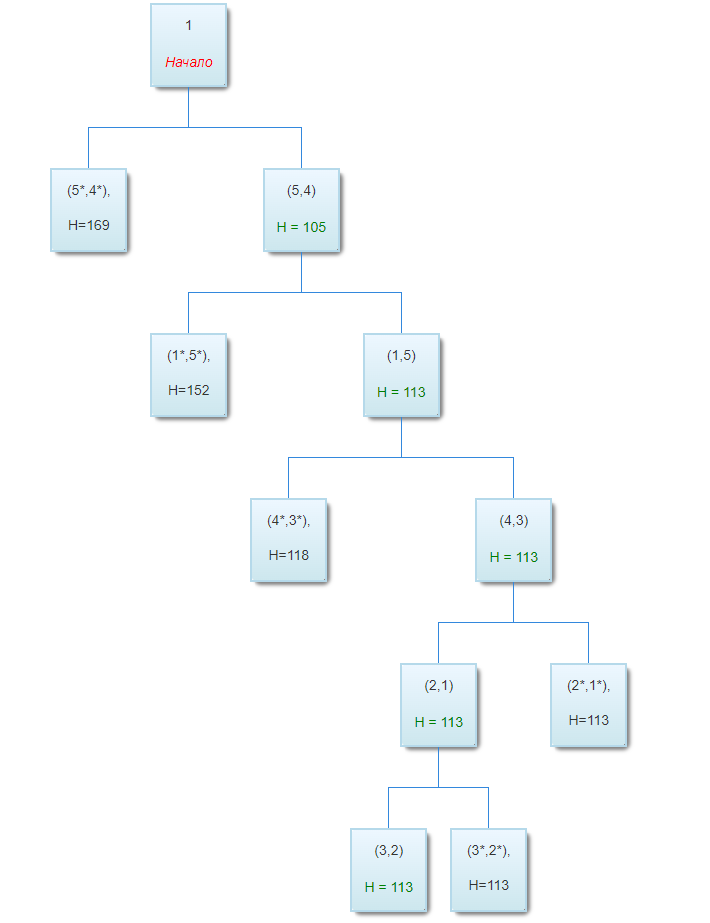
|  |  |  |  |
| --- | --- | --- | --- |
| 0 | INF | 2 | 0 |
| 0 | 0 | INF | 0 |
| INF | 3 | INF | 3 |
| 0 | 0 | 2 | 5 |

Нижняя граница гамильтоновых циклов этого подмножества:  
H(4\*,3\*) = 113 + 5 = 118

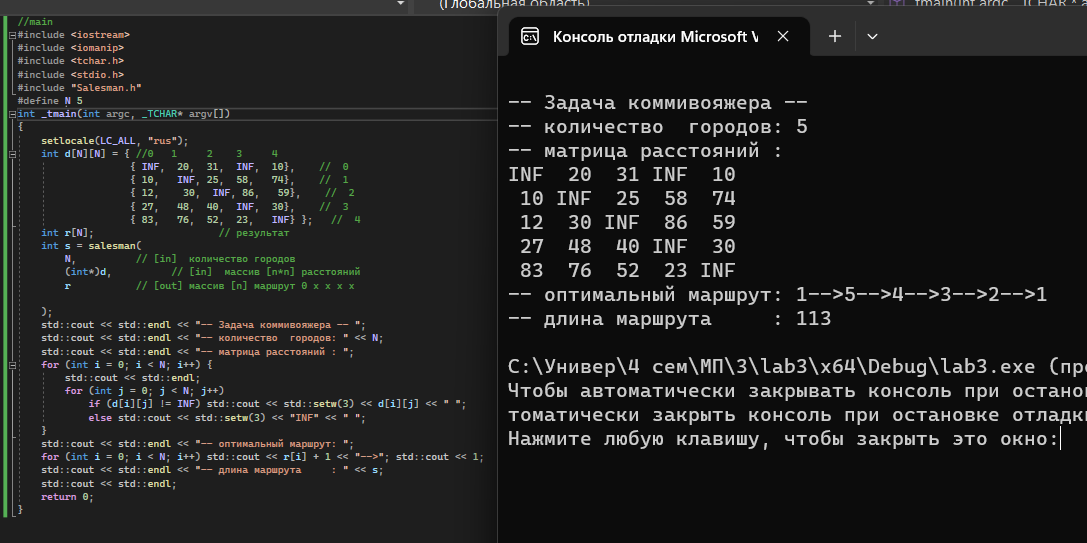
**Включение ребра** (4,3) проводится путем исключения всех элементов 4-ой строки и 3-го столбца, в которой элемент d34 заменяем на INF, для исключения образования негамильтонова цикла.  
В результате получим другую сокращенную матрицу (2 x 2), которая подлежит операции приведения.  
После операции приведения сокращенная матрица будет иметь вид:

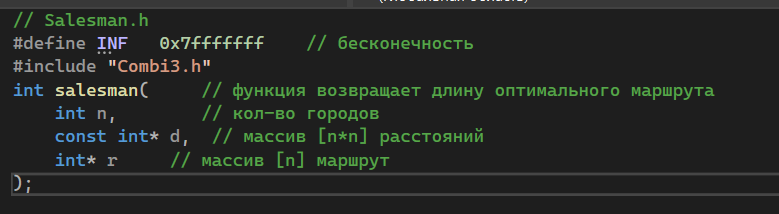
|  |  |  |
| --- | --- | --- |
| 0 | INF | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |

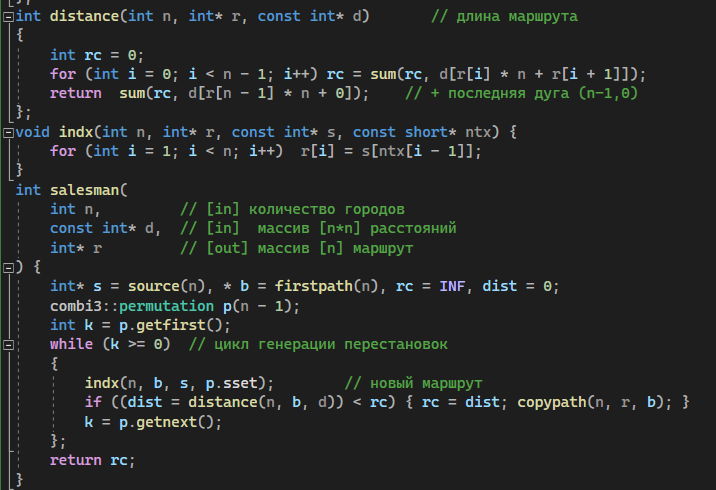
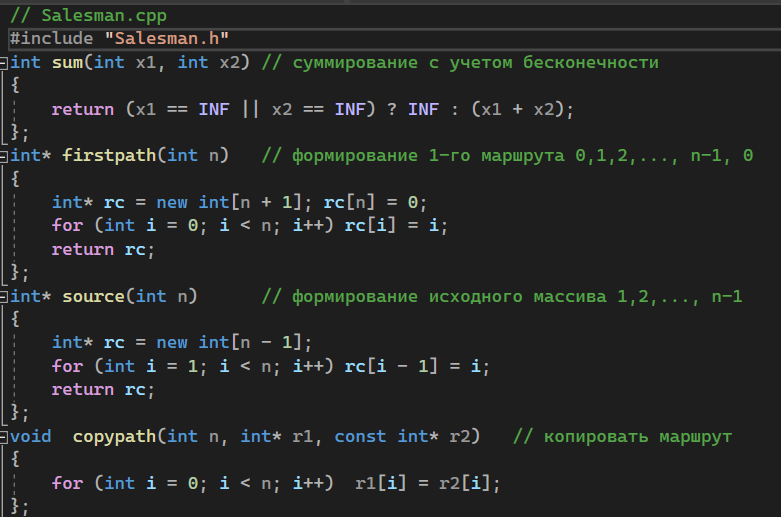
Нижняя граница подмножества (4,3) равна:  
H(4,3) = 113 + 0 = 113 ≤ 118  
Поскольку нижняя граница этого подмножества (4,3) меньше, чем подмножества (4\*,3\*), то ребро (4,3) включаем в маршрут с новой границей H = 113  
В соответствии с этой матрицей включаем в гамильтонов маршрут ребра (2,1) и (3,2).  
В результате по дереву ветвлений гамильтонов цикл образуют ребра:  
(5,4), (4,3), (3,2), (2,1), (1,5),  
**Длина маршрута равна F(Mk) = 113**

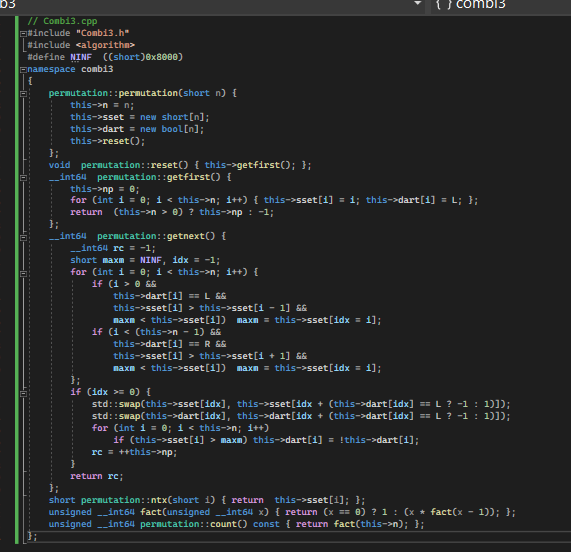
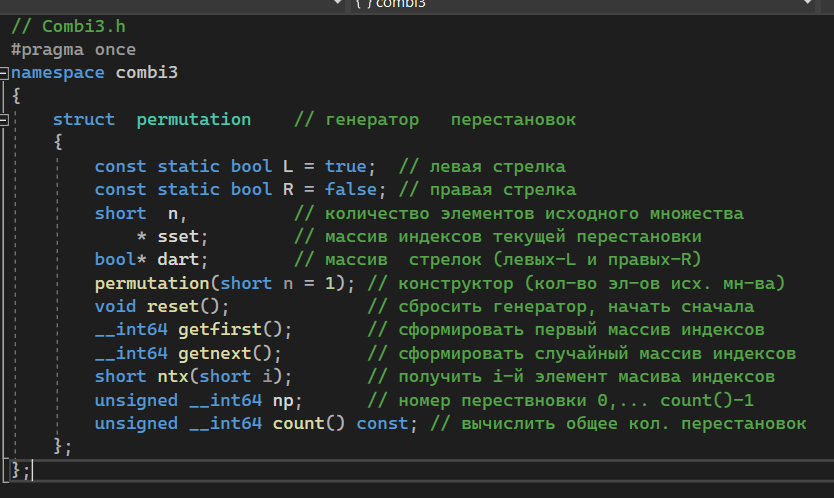


**Проверка генератором перестановок**









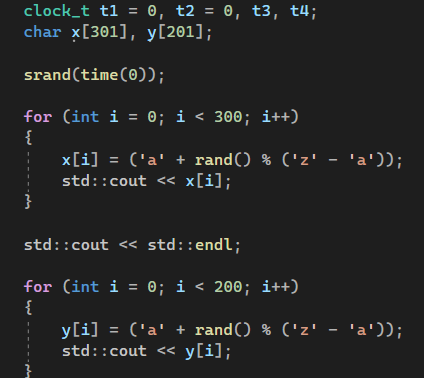
Вывод: освоили общие принципы решения задач методом ветвей и границ, решили задачу о коммивояжере данным методом, сравнили полученное решение задачи с комбинаторным методом перестановок.

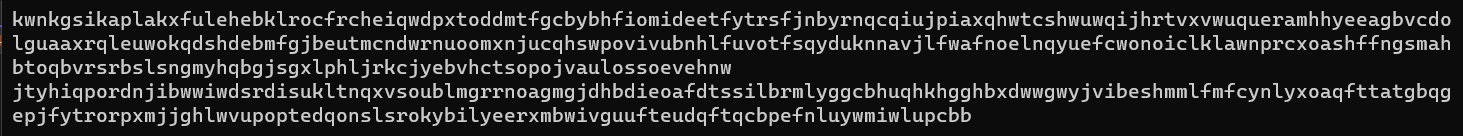
**Лабораторная работа 4**

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

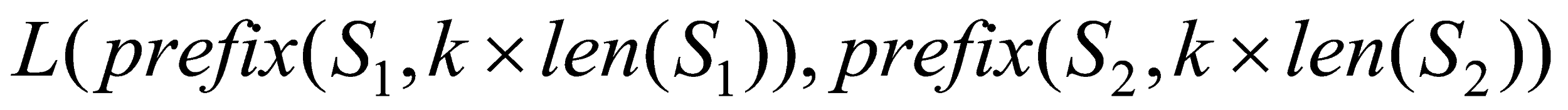
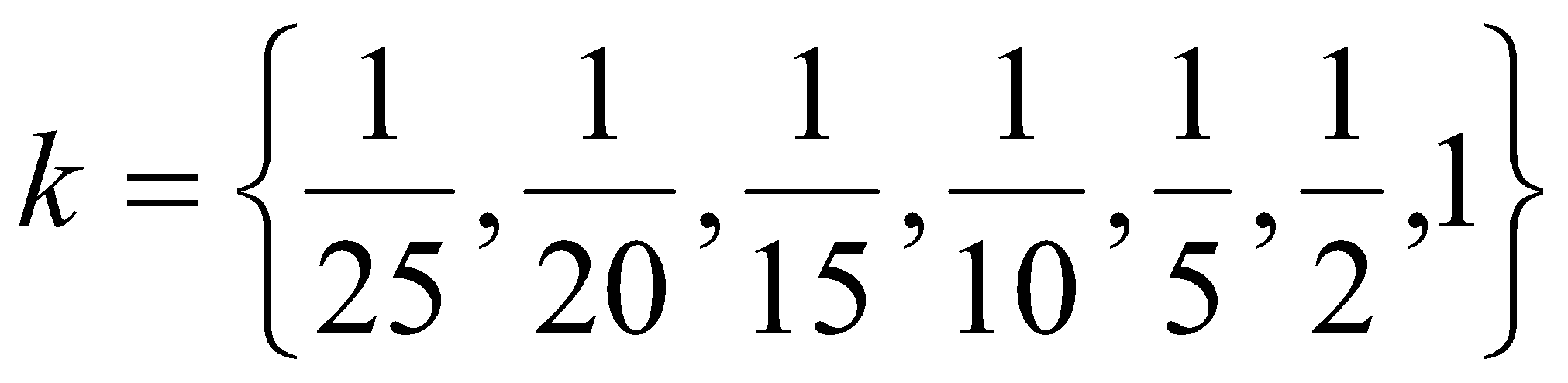
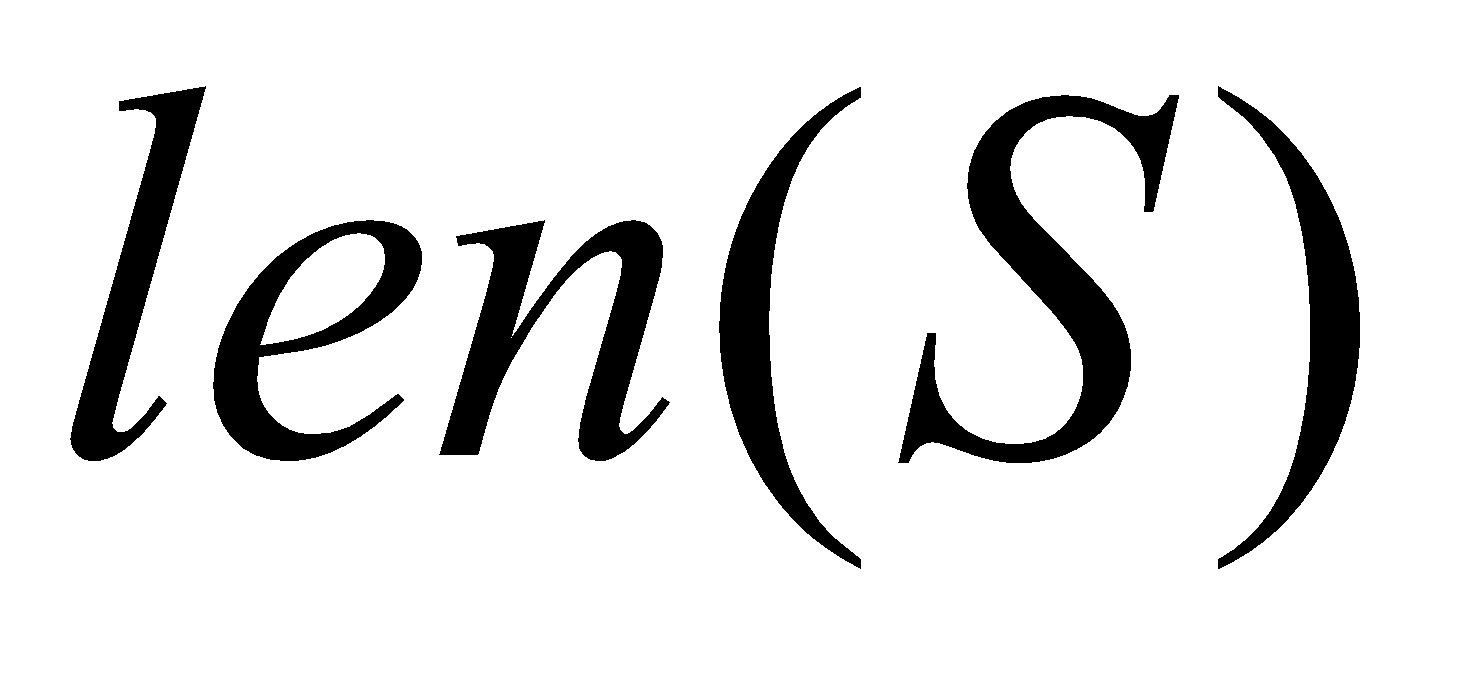
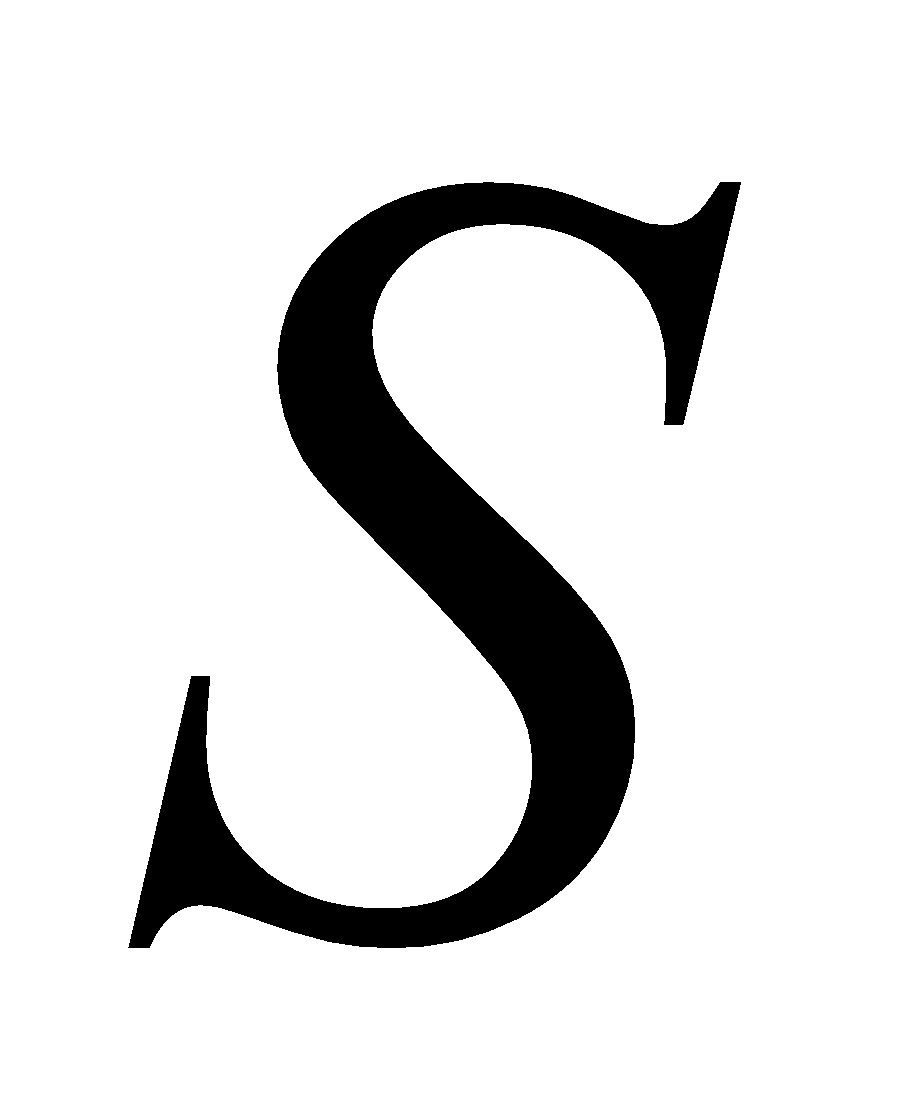
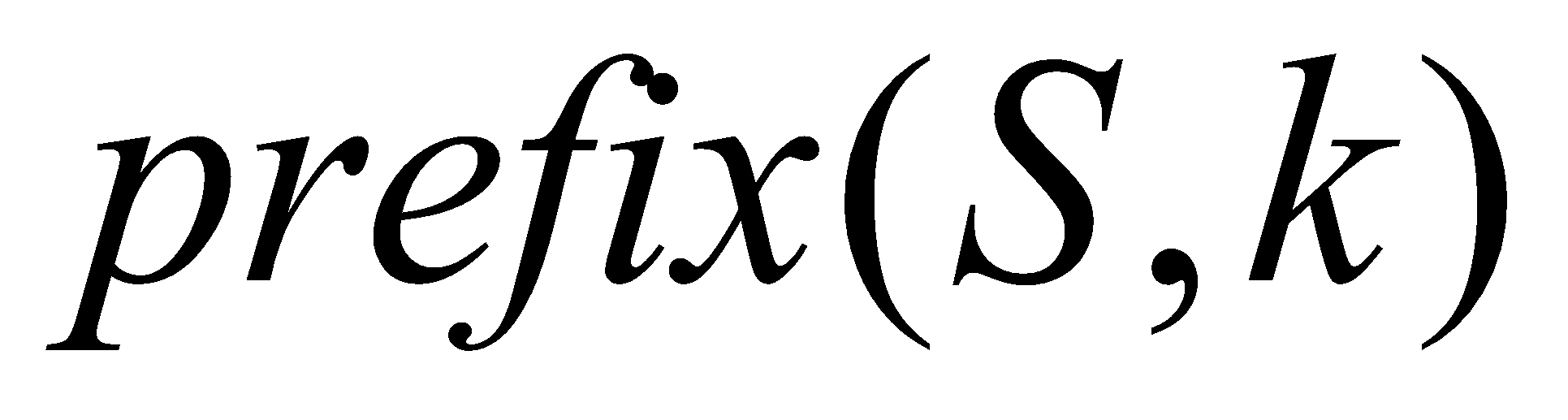
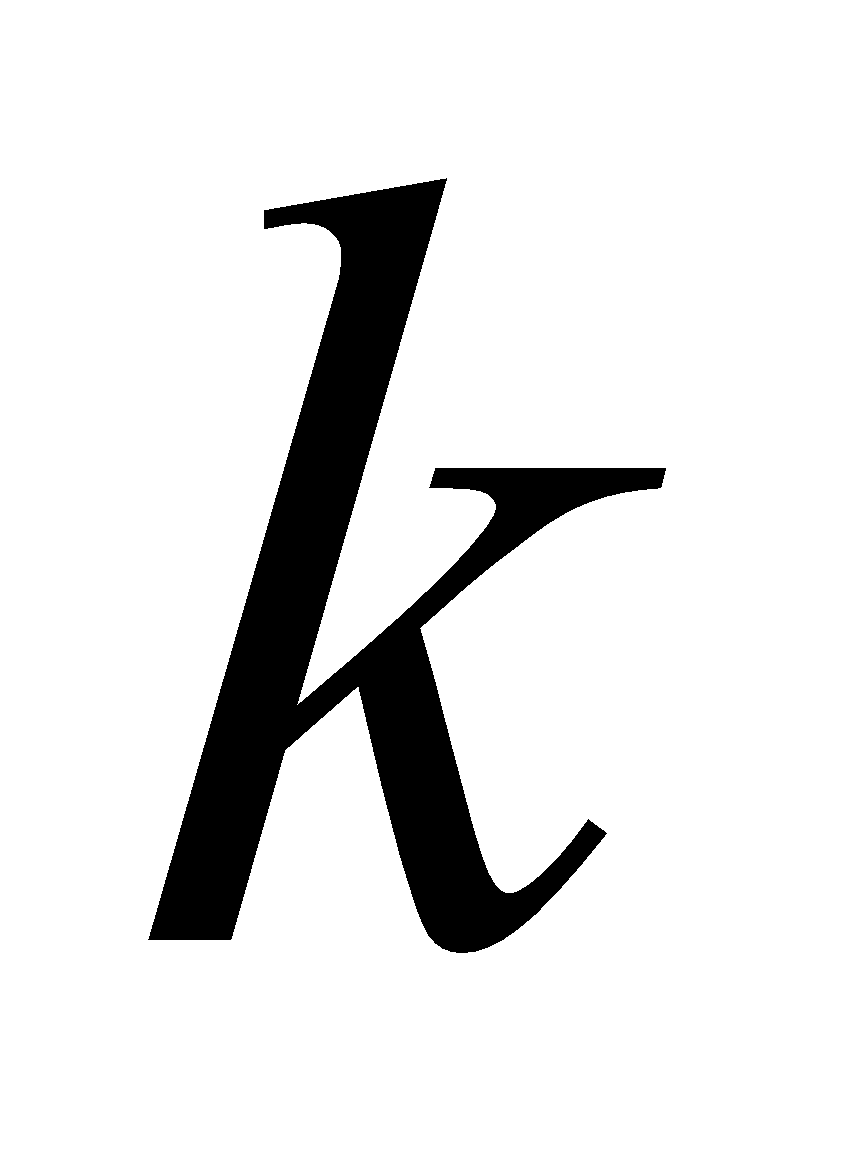
***Задание 1.***

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

****

****

***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).

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

Исходный код реализации через динамическое программирование:

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);

}

Пример реализации рекурсивным методом:

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

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

**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**,

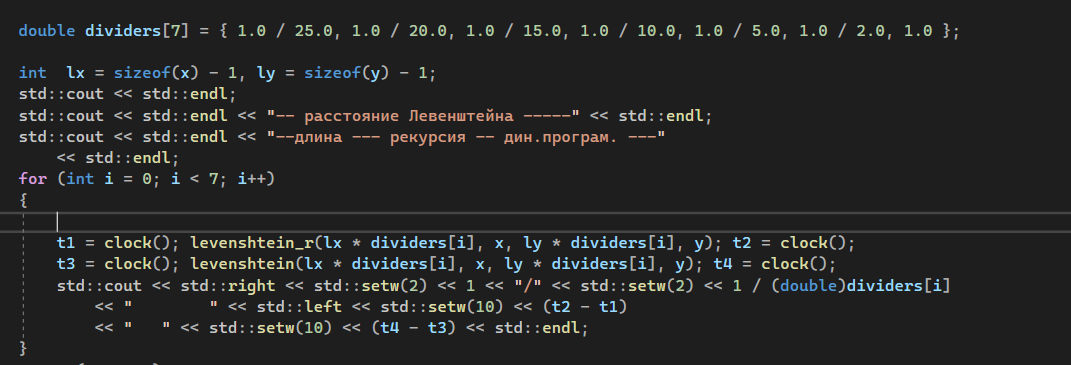
levenshtein\_r(lx, x, ly - **1**, y) + **1**,

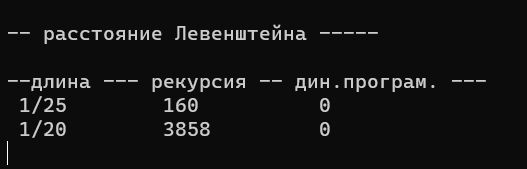
levenshtein\_r(lx - **1**, x, ly - **1**, y) + (x[lx - **1**] == y[ly - **1**] ? **0** : **1**)

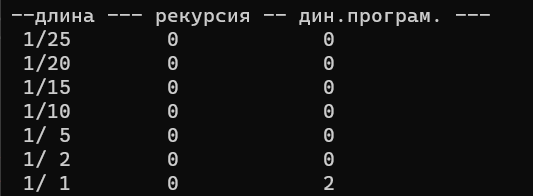
);

**return** rc;

};

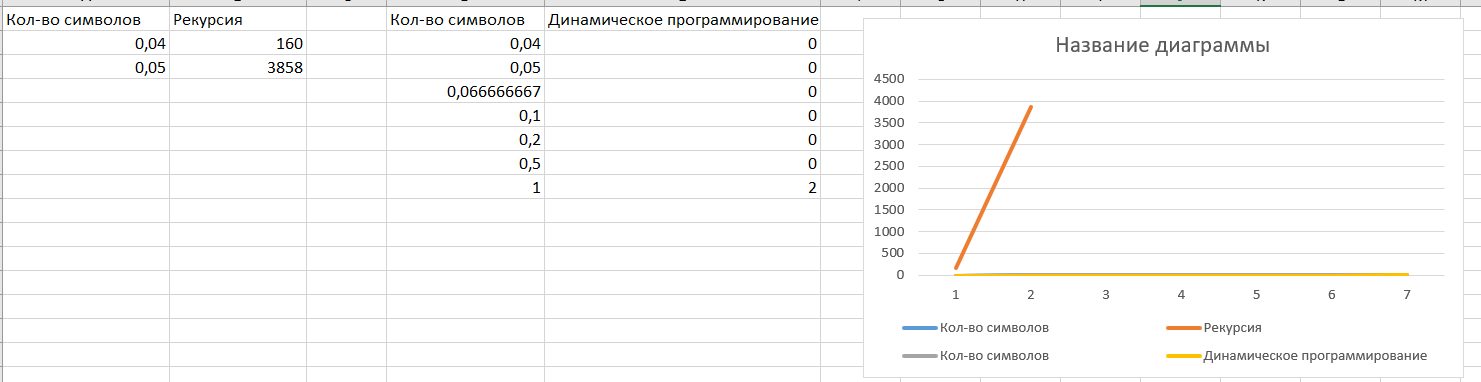






***Задание 3.***

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).

****

Метод динамического программирования значительно эффективнее рекурсивного метода, т.к. выполняется намного быстрее (моментально).

***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).



**Вычисление дистанции Левенштейна**

– количество символов в заданной строке. Например,

– заданная строка без последнего символа. Например,

– последний символ заданной строки. Например,

Пусть необходимо вычислить . Тогда имеем следующую последовательность шагов:



class Programm

{

static int Minimum(int a, int b, int c) => (a = a < b ? a : b) < c ? a : c;

static int LevenshteinDistance(string text1, int len1, string text2, int len2)

{

if (len1 == 0)

{

return len2;

}

if (len2 == 0)

{

return len1;

}

var substitutionCost = 0;

if (text1[len1 - 1] != text2[len2 - 1])

{

substitutionCost = 1;

}

var deletion = LevenshteinDistance(text1, len1 - 1, text2, len2) + 1;

var insertion = LevenshteinDistance(text1, len1, text2, len2 - 1) + 1;

var substitution = LevenshteinDistance(text1, len1 - 1, text2, len2 - 1) + substitutionCost;

return Minimum(deletion, insertion, substitution);

}

static void Main(string[] args)

{

Console.Write("Первое слово: Ель");

string s1 = "Ель";

Console.Write("\tВторое слово: Дрель");

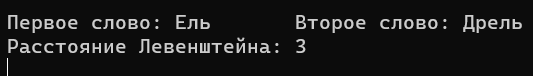
string s2 = "Дрель";

Console.WriteLine("\nРасстояние Левенштейна: {0}", LevenshteinDistance(s1, 3, s2, 5));

Console.ReadLine();

}

}



Таким образом дистанция Левенштейна для данных строк равна 3.

***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). Размерность матриц взять в соответствии с вариантом. Объяснить в отчете принцип расставления скобок по итоговой матрице + код + копии экрана.

Код программы:

// --- MultyMatrix.h

// расстановка скобок

#pragma once

// расстановка скобок при умножении матриц

// функции возвращают минимальное количество операций умножения

#define OPTIMALM\_PARM(x) ((int\*)x) // для представления 2мерного массива

int OptimalM( // рекурсия

int i, // [in] номер первой матрицы

int j, // [in] номер последней матрицы

int n, // [in] количество матриц

const int c[], // [in] массив размерностей

int\* s // [out] результат: позиции скобок

);

int OptimalMD( // динамическое программирование

int n, // [in] количество матриц

const int c[], // [in] массив размерностей

int\* s // [out] результат: позиции скобок

);

// --- MultiMatrix.cpp

// расстановка скобок (рекурсия)

#include <memory.h>

#include "MultiMatrix.h"

#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, 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

};

// --- MultyMatrix.cpp (продолжение)

// 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

};

#include <cmath>

#include <memory.h>

#include <iostream>

#include "MultiMatrix.h" // умножение матриц

#define N 6

int main()

{

int Mc[N + 1] = { 7,10,18,21,28,38,49 }, Ms[N][N], r = 0, rd = 0;

memset(Ms, 0, sizeof(int) \* N \* N);

r = OptimalM(1, N, N, Mc, OPTIMALM\_PARM(Ms));

setlocale(LC\_ALL, "rus");

std::cout << std::endl;

std::cout << std::endl << "-- расстановка скобок (рекурсивное решение) "

<< std::endl;

std::cout << std::endl << "размерности матриц: ";

for (int i = 1; i <= N; i++) std::cout << "(" << Mc[i - 1] << "," << Mc[i] << ") ";

std::cout << std::endl << "минимальное количество операций умножения: " << r;

std::cout << std::endl << std::endl << "матрица S" << std::endl;

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

{

std::cout << std::endl;

for (int j = 0; j < N; j++) std::cout << Ms[i][j] << " ";

}

std::cout << std::endl;

memset(Ms, 0, sizeof(int) \* N \* N);

rd = OptimalMD(N, Mc, OPTIMALM\_PARM(Ms));

std::cout << std::endl

<< "-- расстановка скобок (динамичеое программирование) " << std::endl;

std::cout << std::endl << "размерности матриц: ";

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

std::cout << "(" << Mc[i - 1] << "," << Mc[i] << ") ";

std::cout << std::endl << "минимальное количество операций умножения: "

<< rd;

std::cout << std::endl << std::endl << "матрица S" << std::endl;

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

{

std::cout << std::endl;

for (int j = 0; j < N; j++) std::cout << Ms[i][j] << " ";

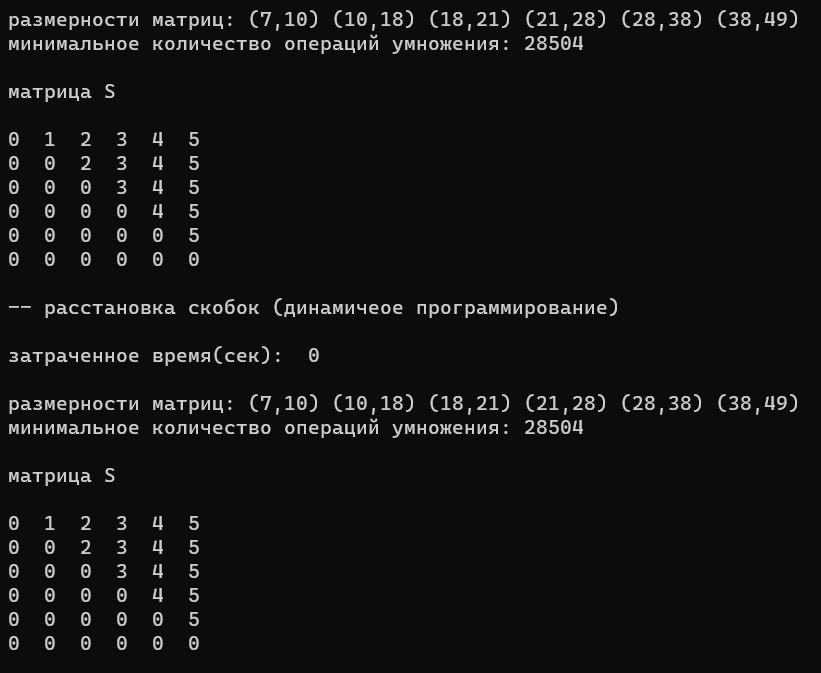
}

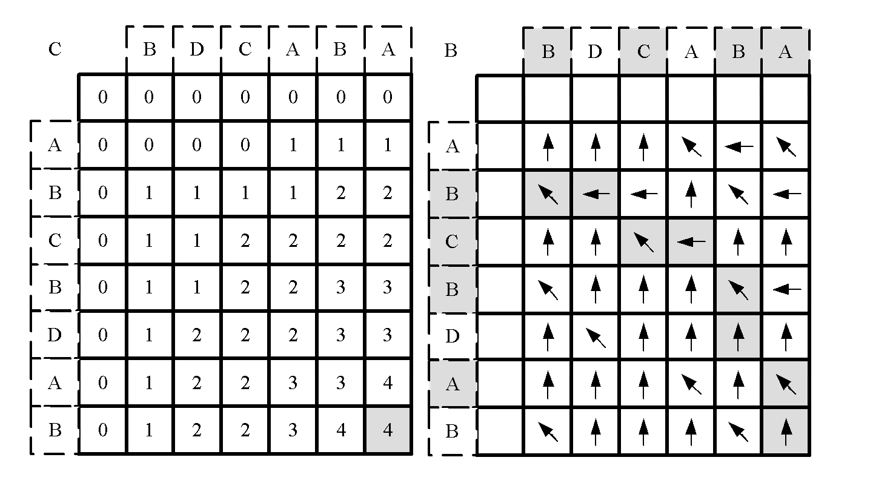
std::cout << std::endl << std::endl;

system("pause");

return 0;

}



****

**Принцип расстановки скобок по итоговой матрице:**

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

А1=7\*10,

А2=10\*18,

А3=18\*21,

А4 =21\*28,

А5 =28\*38,

А6 =38\*49.

Матрица S:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **1** | **2** | **3** | **4** | **5** | **6** |
| **1** | 0 | 1 | 2 | 3 | 4 | 5 |
| **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 5. Это означает, что точка разрыва между 1-ой и 6-ой матрицей находится после 5-ой матрицы. Что позволяет расставить скобки следующим образом:

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

Точку разрыва между первой и пятой матрицей определяет элемент (1,5). Он равен 4. Следовательно разрыв будет после четвертой матрицы.

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

Далее берем элемент (1,4) и получаем, что он равен 3. Следовательно получаем:

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

И на последнем шаге мы возьмем элемент (1,3) и он равен 2:

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

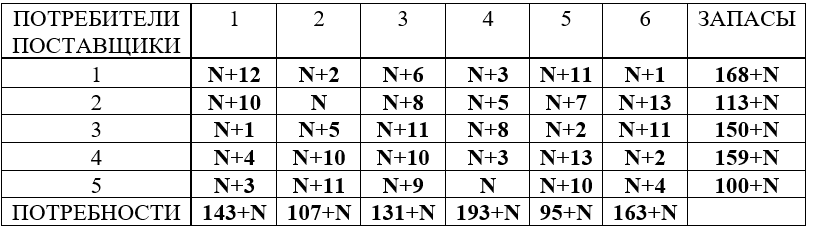
Это выражение и есть конечное.

Полученная расстановка скобок позволяет получить минимальное количество операций умножения, равное 28504.

**Вывод:** в ходе работы были освоены общие принципы решения задач методом динамического программирования, проведено сравнение полученных решений задач с рекурсивным методом.

**Лабораторная работа 5. ТРАНСПОРТНАЯ ЗАДАЧА**

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



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 22 | 12 | 16 | 13 | 13 | 21 | 169 |
| 20 | 10 | 18 | 15 | 17 | 23 | 123 |
| 11 | 15 | 21 | 18 | 12 | 21 | 160 |
| 14 | 20 | 20 | 13 | 23 | 12 | 169 |
| 13 | 21 | 19 | 10 | 20 | 14 | 110 |
| 153 | 117 | 141 | 203 | 105 | 173 |  |

Проверим необходимое и достаточное условие разрешимости задачи.  
∑a = 169 + 123 + 160 + 169 + 110 = 731 (запасы)  
∑b = 153 + 117 + 141 + 203 + 105 + 173 = 892 (потребности)

Как видно, суммарная потребность груза в пунктах назначения превышает запасы груза на базах. Следовательно, модель исходной транспортной задачи является открытой. Чтобы получить закрытую модель, введем дополнительную (фиктивную) базу с запасом груза, равным 161 (892-731). Тарифы перевозки единицы груза из базы во все магазины полагаем равны нулю.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 22 | 12 | 16 | 13 | 21 | 11 | 169 |
| 20 | 10 | 18 | 15 | 17 | 23 | 123 |
| 11 | 15 | 21 | 18 | 12 | 21 | 160 |
| 14 | 20 | 20 | 13 | 23 | 12 | 169 |
| 13 | 21 | 19 | 10 | 20 | 14 | 110 |
| 0 | 0 | 0 | 0 | 0 | 0 | 161 |
| 153 | 117 | 141 | 203 | 105 | 173 |  |

**Поиск первого опорного плана (метод наименьшей стоимости)**  
Суть метода заключается в том, что из всей таблицы стоимостей выбирают наименьшую, и в клетку, которая ей соответствует, помещают меньшее из чисел ai, или bj.  
Затем, из рассмотрения исключают либо строку, соответствующую поставщику, запасы которого полностью израсходованы, либо столбец, соответствующий потребителю, потребности которого полностью удовлетворены, либо и строку и столбец, если израсходованы запасы поставщика и удовлетворены потребности потребителя.  
Из оставшейся части таблицы стоимостей снова выбирают наименьшую стоимость, и процесс распределения запасов продолжают, пока все запасы не будут распределены, а потребности удовлетворены.  
Искомый элемент (2,2) равен 10. Для этого элемента запасы равны 123, потребности 117. Поскольку минимальным является 117, то вычитаем его.  
x22 = min(123,117) = 117.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 22 | x | 16 | 13 | 21 | 11 | 169 |
| 20 | 10 | 18 | 15 | 17 | 23 | 123-117=6 |
| 11 | x | 21 | 18 | 12 | 21 | 160 |
| 14 | x | 20 | 13 | 23 | 12 | 169 |
| 13 | x | 19 | 10 | 20 | 14 | 110 |
| 0 | x | 0 | 0 | 0 | 0 | 161 |
| 153 | 117–117=0 | 141 | 203 | 105 | 173 |  |

Искомый элемент (5,4) равен 10. Для этого элемента запасы равны 110, потребности 203. Поскольку минимальным является 110, то вычитаем его.  
x54 = min(110,203) = 110.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 22 | x | 16 | 13 | 21 | 11 | 169 |
| 20 | x | 18 | 15 | 17 | 23 | 6 |
| 11 | x | 21 | 18 | 12 | 21 | 160 |
| 14 | x | 20 | 13 | 23 | 12 | 169 |
| x | x | x | 10 | x | x | 110-110=0 |
| 0 | x | 0 | 0 | 0 | 0 | 161 |
| 153 | 0 | 141 | 203-110 = 93 | 105 | 173 |  |

Искомый элемент (1,6) равен 11. Для этого элемента запасы равны 169, потребности 173. Поскольку минимальным является 169, то вычитаем его.  
x16 = min(169,173) = 169.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | 11 | 169-169=0 |
| 20 | x | 18 | 15 | 17 | 23 | 6 |
| 11 | x | 21 | 18 | 12 | 21 | 160 |
| 14 | x | 20 | 13 | 23 | 12 | 169 |
| x | x | x | 10 | x | x | 0 |
| 0 | x | 0 | 0 | 0 | 0 | 161 |
| 153 | 0 | 141 | 93 | 105 | 173-169=4 |  |

Искомый элемент (3,1) равен 11. Для этого элемента запасы равны 160, потребности 153. Поскольку минимальным является 153, то вычитаем его.  
x31 = min(160,153) = 153.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | 11 | 0 |
| x | x | 18 | 15 | 17 | 23 | 6 |
| 11 | x | 21 | 18 | 12 | 21 | 160-153=7 |
| x | x | 20 | 13 | 23 | 12 | 169 |
| x | x | x | 10 | x | x | 0 |
| **x** | x | 0 | 0 | 0 | 0 | 161 |
| 153-153=0 | 0 | 141 | 93 | 105 | 4 |  |

Искомый элемент (3,5) равен 12. Для этого элемента запасы равны 7, потребности 105. Поскольку минимальным является 7, то вычитаем его.  
x35 = min(7,105) = 7.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | 11 | 0 |
| x | x | 18 | 15 | 17 | 23 | 6 |
| 11 | x | x | x | 12 | x | 7-7=0 |
| x | x | 20 | 13 | 23 | 12 | 169 |
| x | x | x | 10 | x | x | 0 |
| x | x | 0 | 0 | 0 | 0 | 161 |
| 0 | 0 | 141 | 93 | 105-7=98 | 4 |  |

Искомый элемент (4,6) равен 12. Для этого элемента запасы равны 169, потребности 4. Поскольку минимальным является 4, то вычитаем его.  
x46 = min(169,4) = 4.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | 11 | 0 |
| x | x | 18 | 15 | 17 | x | 6 |
| 11 | x | x | x | 12 | x | 0 |
| x | x | 20 | 13 | 23 | 12 | 169-4=165 |
| x | x | x | 10 | x | x | 0 |
| x | x | 0 | 0 | 0 | x | 161 |
| 0 | 0 | 141 | 93 | 98 | 4-4=0 |  |

Искомый элемент (4,4) равен 13. Для этого элемента запасы равны 165, потребности 93. Поскольку минимальным является 93, то вычитаем его.  
x44 = min(165,93) = 93.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | 11 | 0 |
| x | x | 18 | x | 17 | x | 6 |
| 11 | x | x | x | 12 | x | 0 |
| x | x | 20 | 13 | 23 | 12 | 165-93=72 |
| x | x | x | 10 | x | x | 0 |
| x | x | 0 | 0 | 0 | x | 161 |
| 0 | 0 | 141 | 93-93=0 | 98 | 0 |  |

Искомый элемент (2,5) равен 17. Для этого элемента запасы равны 6, потребности 98. Поскольку минимальным является 6, то вычитаем его.  
x25 = min(6,98) = 6.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | 11 | 0 |
| x | x | x | x | 17 | x | 6-6=0 |
| 11 | x | x | x | 12 | x | 0 |
| x | x | 20 | 13 | 23 | 12 | 72 |
| x | x | x | 10 | x | x | 0 |
| x | x | 0 | 0 | 0 | x | 161 |
| 0 | 0 | 141 | 0 | 98-6=92 | 0 |  |

Искомый элемент (4,3) равен 20. Для этого элемента запасы равны 72, потребности 141. Поскольку минимальным является 72, то вычитаем его.  
x43 = min(72,141) = 72.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | 11 | 0 |
| x | x | x | x | 17 | x | 0 |
| 11 | x | x | x | 12 | x | 0 |
| x | x | 20 | 13 | x | 12 | 72-72=0 |
| x | x | x | 10 | x | x | 0 |
| x | x | 0 | 0 | 0 | x | 161 |
| 0 | 0 | 141-72=69 | 0 | 92 | 0 |  |

Искомый элемент (6,3) равен 0. Для этого элемента запасы равны 161, потребности 69. Поскольку минимальным является 69, то вычитаем его.  
x63 = min(161,69) = 69.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | 11 | 0 |
| x | x | x | x | 17 | x | 0 |
| 11 | x | x | x | 12 | x | 0 |
| x | x | 20 | 13 | x | 12 | 0 |
| x | x | x | 10 | x | x | 0 |
| x | x | 0 | 0 | 0 | x | 161-69=92 |
| 0 | 0 | 69-69=0 | 0 | 92 | 0 |  |

Искомый элемент (6,5) равен 0. Для этого элемента запасы равны 92, потребности 92. Поскольку минимальным является 92, то вычитаем его.  
x65 = min(92,92) = 92.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| x | x | x | x | x | 11 | 0 |
| x | x | x | x | 17 | x | 0 |
| 11 | x | x | x | 12 | x | 0 |
| x | x | 20 | 13 | x | 12 | 0 |
| x | x | x | 10 | x | x | 0 |
| x | x | 0 | 0 | 0 | x | 92-92=0 |
| 0 | 0 | 0 | 0 | 92-92=0 | 0 |  |

Итог

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | Запасы |
|  | 22 | 12 | 16 | 13 | 21 | 11(169) | 169 |
|  | 20 | 10(117) | 18 | 15 | 17(6) | 23 | 123 |
|  | 11(153) | 15 | 21 | 18 | 12(7) | 21 | 160 |
|  | 14 | 20 | 20(72) | 13(93) | 23 | 12(4) | 169 |
|  | 13 | 21 | 19 | 10(110) | 20 | 14 | 110 |
|  | 0 | 0 | 0(69) | 0 | 0(92) | 0 | 161 |
| Потребности | 153 | 117 | 141 | 203 | 105 | 173 |  |

В результате получен первый опорный план, который является допустимым, так как все грузы из баз вывезены, потребность магазинов удовлетворена, а план соответствует системе ограничений транспортной задачи.  
Подсчитаем число занятых клеток таблицы, их 11, а должно быть m+n -1 = 11 (m – число строк, n – число столбцов). Следовательно, опорный план является *невырожденным*.  
Значение целевой функции для этого опорного плана равно:  
F(x) = 11\*169 + 10\*117 + 17\*6 + 11\*153 + 12\*7 + 20\*72 + 13\*93 + 12\*4 + 10\*110 + 0\*69 + 0\*92 = 8695.

**Улучшение опорного плана**.

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

u1 + v6 =11; 0 + v6 = 11; v6 = 11  
u4 + v6 = 12; 11 + u4 = 12; u4 = 1  
u4 + v3 = 20; 1 + v3 = 20; v3 = 19  
u6 + v3 = 0; 19 + u6 = 0; u6 = -19  
u6 + v5 = 0; -19 + v5 = 0; v5 = 19  
u2 + v5 = 17; 19 + u2 = 17; u2 = -2  
u2 + v2 = 10; -2 + v2 = 10; v2 = 12  
u3 + v5 = 12; 19 + u3 = 12; u3 = -7  
u3 + v1 = 11; -7 + v1 = 11; v1 = 18  
u4 + v4 = 13; 1 + v4 = 13; v4 = 12  
u5 + v4 = 10; 12 + u5 = 10; u5 = -2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | V1=18 | V2=12 | V3=19 | V4=12 | V5=19 | V6=11 |
| U1=0 | 22 | 12 | 16 | 13 | 21 | 11(169) |
| U2=-2 | 20 | 10(117) | 18 | 15 | 17(6) | 23 |
| U3=-7 | 11(153) | 15 | 21 | 18 | 12(7) | 21 |
| U4=1 | 14 | 20 | 20(72) | 13(93) | 23 | 12(4) |
| U5=-2 | 13 | 21 | 19 | 10(110) | 20 | 14 |
| U6=-19 | 0 | 0 | 0(69) | 0 | 0(92) | 0 |

Опорный план не является оптимальным, так как существуют оценки свободных клеток, для которых ui + vj > cij  
(1;3): 0 + 19 > 16; ∆13 = 0 + 19 - 16 = 3 > 0  
(4;1): 1 + 18 > 14; ∆41 = 1 + 18 - 14 = 5 > 0  
(5;1): -2 + 18 > 13; ∆51 = -2 + 18 - 13 = 3 > 0  
max(3,5,3) = 5

Выбираем максимальную оценку свободной клетки (4;1): 14  
Для этого в перспективную клетку (4;1) поставим знак «+», а в остальных вершинах многоугольника чередующиеся знаки «-», «+», «-».

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | Запасы |
| 1 | 22 | 12 | 16 | 13 | 21 | 11[169] | 169 |
| 2 | 20 | 10[117] | 18 | 15 | 17[6] | 23 | 123 |
| 3 | 11[153][-] | 15 | 21 | 18 | 12[7][+] | 21 | 160 |
| 4 | 14[+] | 20 | 20[72][-] | 13[93] | 23 | 12[4] | 169 |
| 5 | 13 | 21 | 19 | 10[110] | 20 | 14 | 110 |
| 6 | 0 | 0 | 0[69][+] | 0 | 0[92][-] | 0 | 161 |
| Потребности | 153 | 117 | 141 | 203 | 105 | 173 |  |

Цикл приведен в таблице (4,1 → 4,3 → 6,3 → 6,5 → 3,5 → 3,1).  
Из грузов хij стоящих в минусовых клетках, выбираем наименьшее, т.е. у = min (4, 3) = 72. Прибавляем 72 к объемам грузов, стоящих в плюсовых клетках и вычитаем 72 из Хij, стоящих в минусовых клетках. В результате получим новый опорный план.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  | Запасы |
|  | 22 | 12 | 16 | 13 | 21 | 11[169] | 169 |
|  | 20 | 10[117] | 18 | 15 | 17[6] | 23 | 123 |
|  | 11[81] | 15 | 21 | 18 | 12[79] | 21 | 160 |
|  | 14[72] | 20 | 20 | 13[93] | 23 | 12[4] | 169 |
|  | 13 | 21 | 19 | 10[110] | 20 | 14 | 110 |
|  | 0 | 0 | 0[141] | 0 | 0[20] | 0 | 161 |
| Потребности | 153 | 117 | 141 | 203 | 105 | 173 |  |

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

u1 + v6 = 11; 0 + v6 = 11; v6 = 11  
u4 + v6 = 12; 11 + u4 = 12; u4 = 1  
u4 + v1 = 14; 1 + v1 = 14; v1 = 13  
u3 + v1 = 11; 13 + u3 = 11; u3 = -2  
u3 + v5 = 12; -2 + v5 = 12; v5 = 14  
u2 + v5 = 17; 14 + u2 = 17; u2 = 3  
u2 + v2 = 10; 3 + v2 = 10; v2 = 7  
u6 + v5 = 0; 14 + u6 = 0; u6 = -14  
u6 + v3 = 0; -14 + v3 = 0; v3 = 14  
u4 + v4 = 13; 1 + v4 = 13; v4 = 12  
u5 + v4 = 10; 12 + u5 = 10; u5 = -2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | v1=13 | v2=7 | v3=14 | v4=12 | v5=14 | v6=11 |
| u1=0 | 22 | 12 | 16 | 13 | 21 | 11[169] |
| u2=3 | 20 | 10[117] | 18 | 15 | 17[6] | 23 |
| u3=-2 | 11[81] | 15 | 21 | 18 | 12[79] | 21 |
| u4=1 | 14[72] | 20 | 20 | 13[93] | 23 | 12[4] |
| u5=-2 | 13 | 21 | 19 | 10[110] | 20 | 14 |
| u6=-14 | 0 | 0 | 0[141] | 0 | 0[20] | 0 |

Опорный план является оптимальным, так все оценки свободных клеток удовлетворяют условию ui + vj ≤ cij.  
Минимальные затраты составят: F(x) = 11\*169 + 10\*117 + 17\*6 + 11\*81 + 12\*79 + 14\*72 + 13\*93 + 12\*4 + 10\*110 + 0\*141 + 0\*20 = 8335.

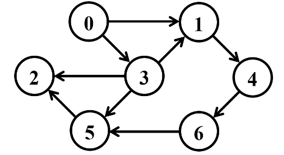
**Анализ оптимального плана**.  
Из 1-го склада необходимо весь груз направить в 6-й магазин.  
Из 2-го склада необходимо груз направить в 2-й магазин (117 ед.), в 5-й магазин (6 ед.)  
Из 3-го склада необходимо груз направить в 1-й магазин (81 ед.), в 5-й магазин (79 ед.)  
Из 4-го склада необходимо груз направить в 1-й магазин (72 ед.), в 4-й магазин (93 ед.), в 6-й магазин (4 ед.)  
Из 5-го склада необходимо весь груз направить в 4-й магазин.  
Потребность 3-го магазина остается неудовлетворенной на 141 ед.  
Оптимальный план является вырожденным, так как базисная переменная x63=0.  
Потребность 5-го магазина остается неудовлетворенной на 20 ед.  
Оптимальный план является вырожденным, так как базисная переменная x65=0.

**Вывод**: приобретены навыки решения открытой транспортной задачи.

**Лабораторная работа 6. АЛГОРИТМЫ НА ГРАФАХ**

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

Граф:



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

Матрица смежности:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| 0 |  | 1(a) |  | 1(b) |  |  |  |
| 1 |  |  |  |  | 1(c) |  |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  | 1(d) | 1(e) |  |  | 1(f) |  |
| 4 |  |  |  |  |  |  | 1(g) |
| 5 |  |  | 1(h) |  |  |  |  |
| 6 |  |  |  |  |  | 1(i) |  |

Матрица инцидентности:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | A | B | C | D | E | F | G | H | I |
| 0 | 1 | 1 |  |  |  |  |  |  |  |
| 1 | -1 |  | 1 | -1 |  |  |  |  |  |
| 2 |  |  |  |  | -1 |  |  | -1 |  |
| 3 |  | -1 |  | 1 | 1 | 1 |  |  |  |
| 4 |  |  | -1 |  |  |  | 1 |  |  |
| 5 |  |  |  |  |  | -1 |  | 1 | -1 |
| 6 |  |  |  |  |  |  | -1 |  | 1 |

Список смежных вершин:

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

***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.

1. **Алгоритм поиска в ширину (BFS)**

По условию, граф имеет 5 вершин, пронумерованных начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

Текущее состояние алгоритма хранится в следующих структурах памяти:

Q – очередь вершин,

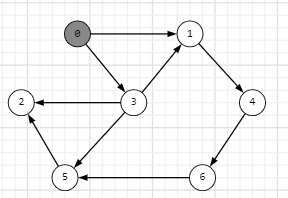
C – массив окраски вершин,

D – массив расстояний,

P – массив предшествующих вершин.

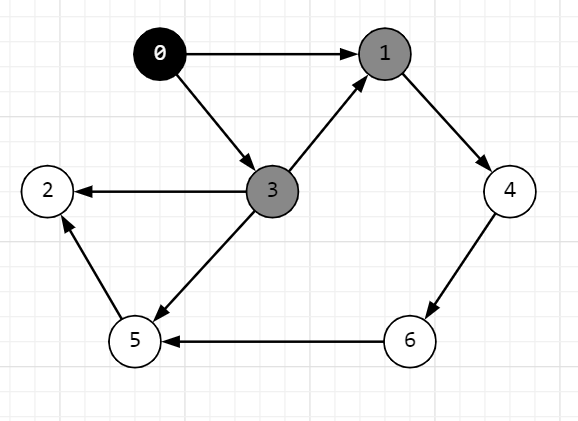
Шаг 1.

В качестве стартовой вершины выберем вершину с номером 0.

****

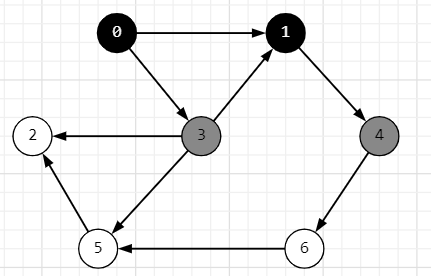
|  |  |
| --- | --- |
| Q | 0 |
| C | G | W | W | W | W | W | W |
| D | 0 | I | I | I | I | I | I |
| P | N | N | N | N | N | N | N |

Шаг 2. У вершины 0 две смежные вершины; для последующего пути выбираю вершину с наименьшим весом из двух – первую, а третью добавляю в начало очереди. Закрашиваю нулевую в черный цвет, как пройденную. А смежные (первая и третья) – в серый.



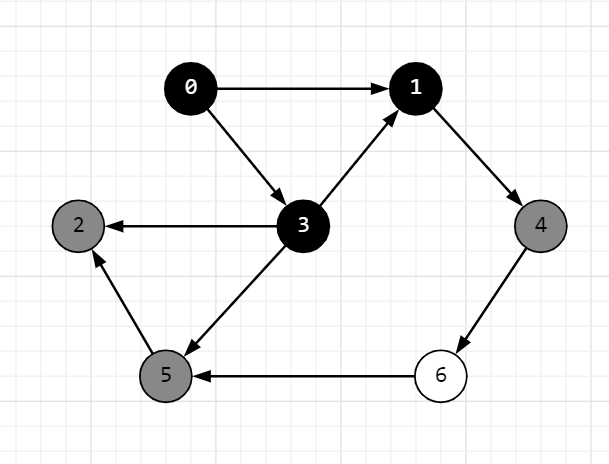
|  |  |  |
| --- | --- | --- |
| Q | 1 | 3 |
| C | B | G | W | G | W | W | W |
| D | 0 | 1 | I | 1 | I | I | I |
| P | N | 0 | N | 0 | N | N | N |

Шаг 3. Перехожу в смежную вершину – вершина номер один. Единственная смежная с ней вершина номер четыре, закрашиваю ее в серый цвет для попадания в очередь, а первую в черный цвет как пройденную.



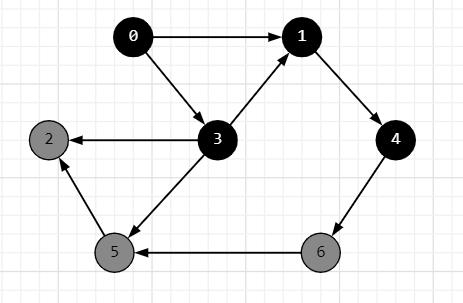
|  |  |  |
| --- | --- | --- |
| Q | 3 | 4 |
| C | B | B | W | G | G | W | W |
| D | 0 | 1 | I | 1 | 2 | I | I |
| P | N | 0 | N | 0 | 1 | N | N |

Шаг 4. Перехожу в смежную вершину – вершина номер три. Смежные с ней вершины: номер два и номер пять, закрашиваю их в серый цвет для попадания в очередь, а третью в черный цвет как пройденную.



|  |  |  |  |
| --- | --- | --- | --- |
| Q | 4 | 2 | 5 |
| C | B | B | G | | B | G | G | W |
| D | 0 | 1 | 3 | | 1 | 2 | 3 | I |
| P | N | 0 | 3 | | 0 | 1 | 3 | N |

Шаг 5. Перехожу в смежную вершину – вершина номер четыре. У вершины четыре смежная вершина одна: вершина номер шесть. Закрашиваю четвёртую вершину в черный цвет, как пройденную, а вершину номер 6 окрашиваю в серый цвет.

****

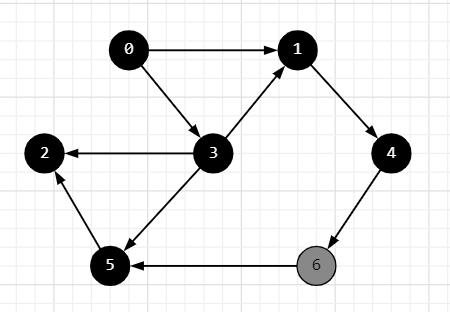
|  |  |  |  |
| --- | --- | --- | --- |
| Q | 2 | 5 | 6 |
| C | B | G | B | | B | B | G | G |
| D | 0 | 1 | 3 | | 1 | 2 | 3 | 4 |
| P | N | 0 | 3 | | 0 | 1 | 3 | 4 |

Шаг 6. Перехожу в смежную вершину – вершина номер два. У неё нет смежных вершин. Закрашиваю вершину два в черный как пройденную.

****

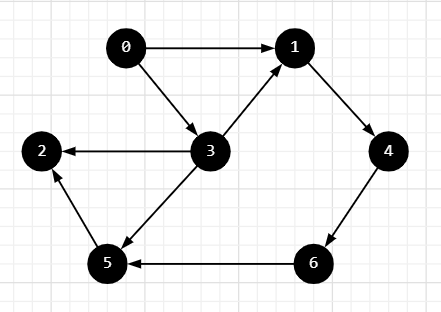
|  |  |  |
| --- | --- | --- |
| Q | 5 | 6 |
| C | B | B | B | B | B | G | G |
| D | 0 | 1 | 3 | 1 | 2 | 3 | 4 |
| P | N | 0 | 3 | 0 | 1 | 3 | 4 |

Шаг 7. Перехожу в смежную вершину – вершина номер два. У неё нет смежных вершин. Закрашиваю вершину пять в черный как пройденную.

****

|  |  |
| --- | --- |
| Q | 2 |
| C | B | B | B | B | B | B | G |
| D | 0 | 1 | 3 | 1 | 2 | 3 | 4 |
| P | N | 0 | 3 | 0 | 1 | 3 | 4 |

Шаг 8. Перехожу в вершину номер шесть. Закрашиваем ее в черный цвет, так как других действий нет, Алгоритм окончен.



|  |
| --- |
| Q |
| C | B | B | B | B | B | B | G |
| D | 0 | 1 | 3 | 1 | 2 | 3 | 4 |
| P | N | 0 | 3 | 0 | 1 | 4 | 3 |

Порядок обхода: 0 1 3 4 2 5 6.

1. **Алгоритм поиска в глубину (DFS).**

По условию, граф имеет 7 вершин, пронумерованных, начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

Текущее состояние алгоритма хранится в следующих структурах памяти:

C – массив окраски вершин,

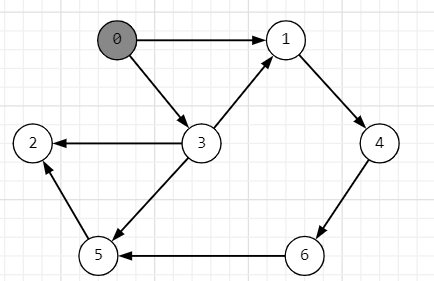
D – время окраски вершин в серый цвет,

P – массив предшествующих вершин,

F – время окраски в чёрный цвет.

Шаг 1.

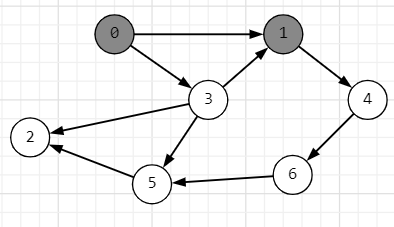
В качестве стартовой вершины выбираем вершину с номером 0. Далее будем осуществлять проход по смежным вершинам, пока не сможет достичь того, чтобы не было возможности осуществить дальнейший проход.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | W | W | W | W | W | W |
| D | 1 | I | I | I | I | I | I |
| P | N | N | N | N | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Шаг 2.

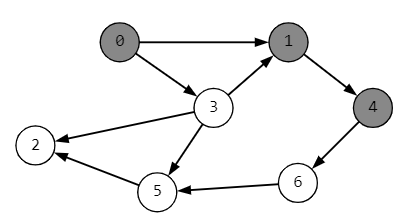
Вершина ноль имеет две смежные вершины, переходим в вершину первую, так как она с наименьшим весом и окрашиваем ее в серый цвет.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | W | W | W | W | W |
| D | 1 | 2 | I | I | I | I | I |
| P | N | 0 | N | N | N | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Шаг 3.

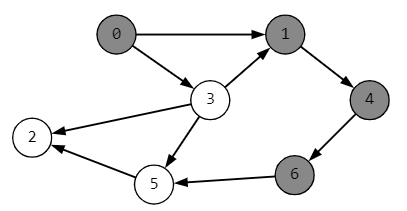
Вершина первая имеет одну смежную вершину – четыре, переходим и окрашиваем ее в серый цвет.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | W | W | G | W | W |
| D | 1 | 2 | I | I | 3 | I | I |
| P | N | 0 | N | N | 1 | N | N |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Шаг 4.

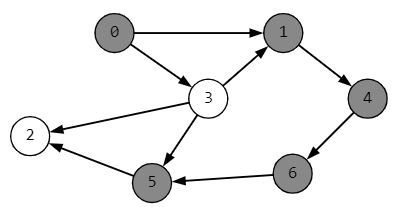
Вершина третья имеет четыре смежные вершины, переходим туда, где вес наименьший – в вершину два, окрашиваем ее в серый цвет.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | W | W | G | W | G |
| D | 1 | 2 | I | I | 3 | I | 4 |
| P | N | 0 | N | N | 1 | N | 4 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Шаг 5.

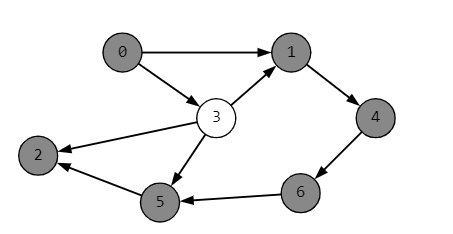
Вершина шестая имеет смежную вершину пять, переходим туда, окрашиваем ее в серый цвет.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | W | W | G | G | G |
| D | 1 | 2 | I | I | 3 | 5 | 4 |
| P | N | 0 | N | N | 1 | 6 | 4 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Шаг 6.

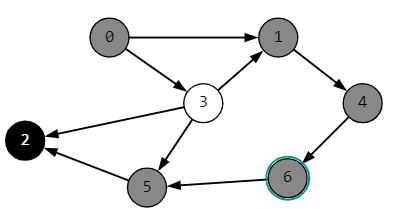
Вершина пятая имеет смежную вершину два, переходим туда, окрашиваем ее в серый цвет.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | G | W | G | G | G |
| D | 1 | 2 | 6 | I | 3 | 5 | 4 |
| P | N | 0 | 5 | N | 1 | 6 | 4 |
| F | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Шаг 7.

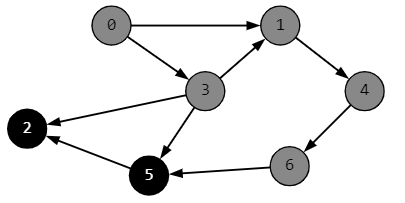
Вершина вторая не имеет смежных белых вершин, следовательно, мы погрузились в глубину. Закрашиваем вторую вершину в чёрный цвет.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | B | W | G | G | G |
| D | 1 | 2 | 6 | I | 3 | 5 | 4 |
| P | N | 0 | 5 | N | 1 | 6 | 4 |
| F | 0 | 0 | 7 | 0 | 0 | 0 | 0 |

Шаг 8.

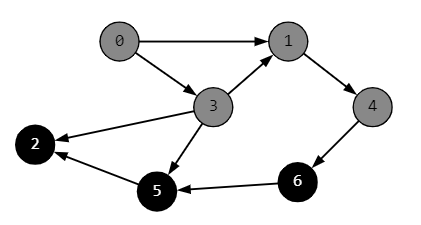
У нас имеется одна не пройденная вершина 3, закрашиваем ее в серый цвет. Возвращаемся обратно, окрашивая вершину 5 в чёрный цвет, ибо она смежная с вершиной 2.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | B | G | G | B | G |
| D | 1 | 2 | 6 | I | 3 | 5 | 4 |
| P | N | 0 | 5 | 0 | 1 | 6 | 4 |
| F | 0 | 0 | 7 | 0 | 0 | 8 | 0 |

Шаг 9.

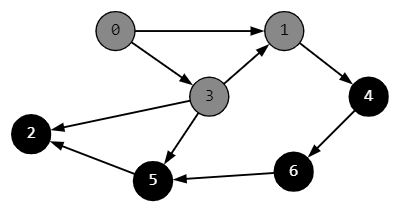
Находим смежные вершины для пятой вершины – это шестая вершина. Закрашиваем ее в чёрный цвет.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | B | G | G | B | B |
| D | 1 | 2 | 6 | 7 | 3 | 5 | 4 |
| P | N | 0 | 5 | 0 | 1 | 6 | 4 |
| F | 0 | 0 | 7 | 0 | 0 | 8 | 9 |

Шаг 10.

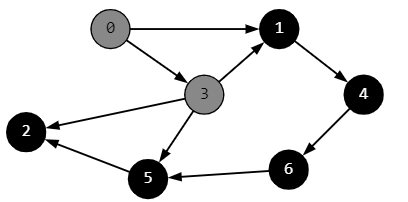
Находим смежные вершины для шестой вершины – это четвёртая вершина. Закрашиваем ее в чёрный цвет.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | G | B | G | B | B | B |
| D | 1 | 2 | 6 | 7 | 3 | 5 | 4 |
| P | N | 0 | 5 | 0 | 1 | 6 | 4 |
| F | 0 | 0 | 7 | 0 | 10 | 8 | 9 |

Шаг 11.

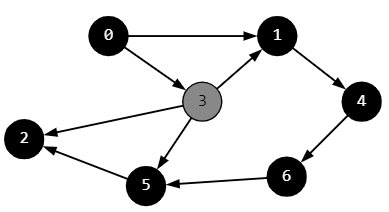
Находим смежные вершины для четвёртой вершины – это первая вершина. Закрашиваем ее в чёрный цвет.

****

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | G | B | B | G | B | B | B |
| D | 1 | 2 | 6 | 7 | 3 | 5 | 4 |
| P | N | 0 | 5 | 0 | 1 | 6 | 4 |
| F | 0 | 11 | 7 | 0 | 10 | 8 | 9 |

Шаг 12.

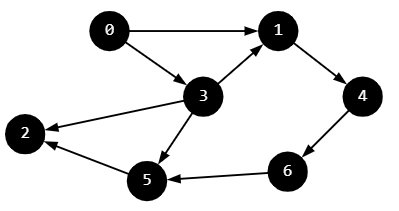
Находим смежные вершины для первой вершины – это нулевая вершина. Закрашиваем ее в чёрный цвет.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | B | B | B | G | B | B | B |
| D | 1 | 2 | 6 | 7 | 3 | 5 | 4 |
| P | N | 0 | 5 | 0 | 1 | 6 | 4 |
| F | 12 | 11 | 7 | 0 | 10 | 8 | 9 |

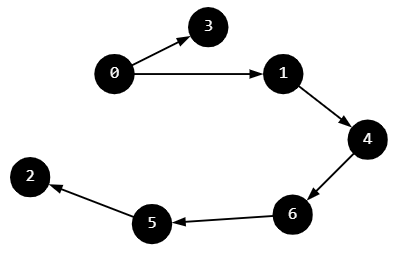
Шаг 13.

Находим смежные вершины для нулевой вершины – это третья вершина. Закрашиваем ее в чёрный цвет.



|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| C | B | B | B | B | B | B | B |
| D | 1 | 2 | 6 | 7 | 3 | 5 | 4 |
| P | N | 0 | 5 | 0 | 1 | 6 | 4 |
| F | 12 | 11 | 7 | 13 | 10 | 8 | 9 |

В результате получили DFS-дерево:



Порядок обхода: 0, 1, 4, 6 5, 2, 3.

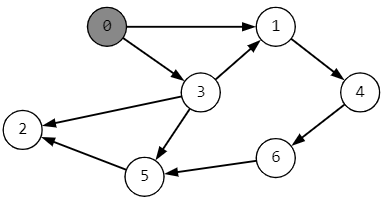
1. **Алгоритм топологической сортировки.**

По условию, граф имеет 7 вершин, пронумерованных начиная с нуля. В качестве стартовой вершины выбрана вершина с номером 0.

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

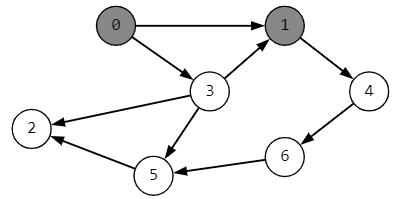
Шаг 1.

В качестве стартовой вершины выбираем вершину с номером 0. Окрашиваем ее в серый цвет.



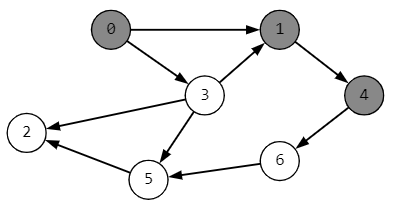
Шаг 2.

Из нулевой идем в первую вершину, красим ее в серый цвет.



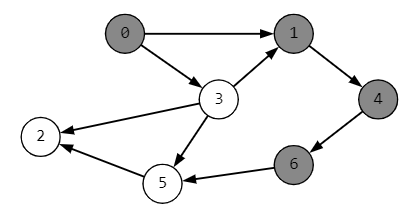
Шаг 3.

Из первой идем в четвёртую вершину, красим ее в серый цвет.



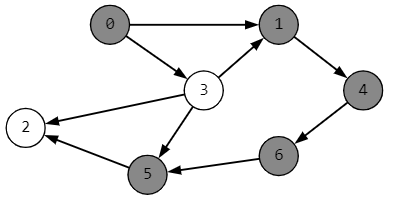
Шаг 4.

Из четвёртой идем в шестую вершину, красим ее в серый цвет.



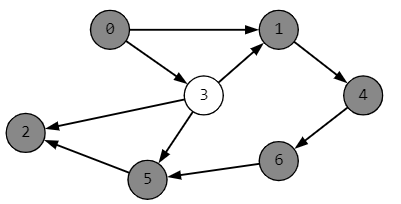
Шаг 5.

Из шестой идем в пятую вершину, красим ее в серый цвет.



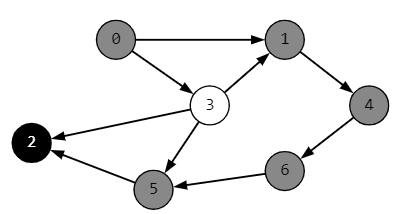
Шаг 6.

Из пятой идем во вторую вершину, красим ее в серый цвет.



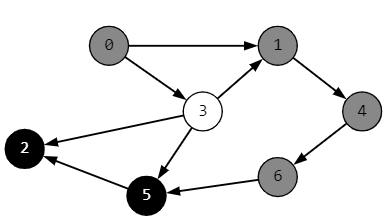
Шаг 7.

Переходим во вторую, она не имеет смежных, тогда красим ее в черный цвет и кладем вторую вершину в стек.



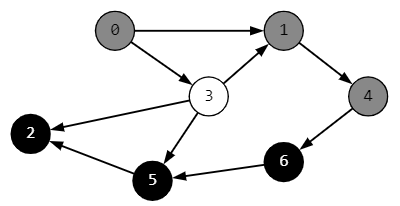
Шаг 8.

Идём в обратном направлении. Пятую красим в черный цвет и кладём её в стек.



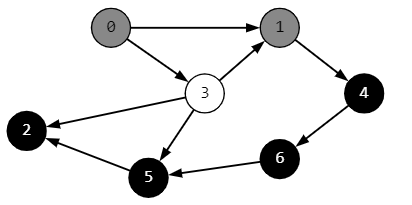
Шаг 9.

Идём в обратном направлении. Шестую красим в черный цвет и кладём её в стек.



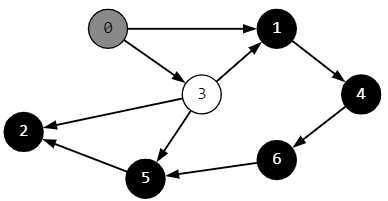
Шаг 10.

Четвёртую вершину красим в черный цвет и кладём её в стек.



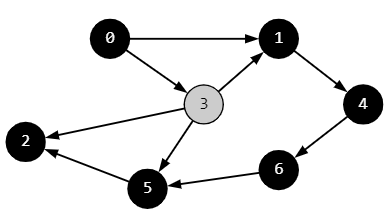
Шаг 11.

Первую вершину красим в черный цвет и кладём её в стек.



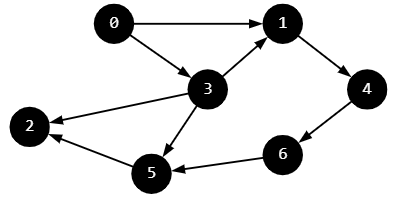
Шаг 12.

Нулевую вершину красим в черный цвет и кладём её в стек. У неё есть смежная незакрашенная вершина 3. Красим её в серый цвет



Шаг 13.

Красим третью вершину в черный цвет.

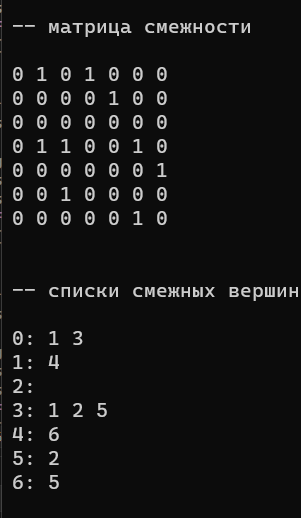


В итоге имеем сортировку: 0, 3, 1, 4, 6, 5, 2.

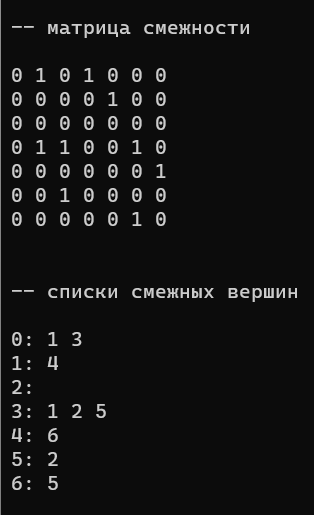
***Задание 3.*** Осуществить программную реализацию алгоритмов на C++. Разработать структуры **AMatrix** и **АList**  для представления ориентированного графа матричным и списковым способом. Разработать функции преобразования из одного способа представления в другой. Разработать функцию **BFS** обхода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

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

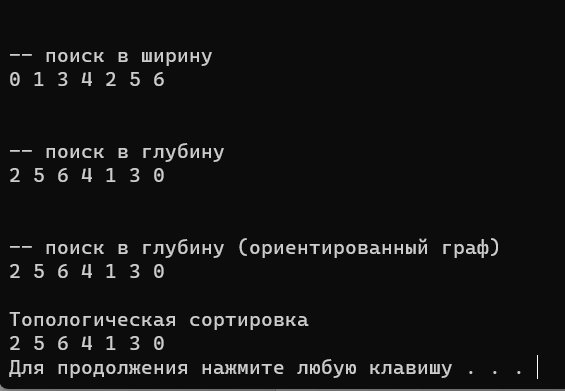
***Задание 5.*** Доработайте функцию **DFS**,для выполнения топологической сортировки графа. Продемонстрировать работу функции. Копии экрана вставить в отчет.



Структура AMatrix



Структура AList



Веса ребер принять W:

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

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

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

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

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

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

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

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

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

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

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

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

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

Начинаем с вершины 0.

Далее, двигаемся в вершину 3. Получаем W(e0,3)=2;  
Двигаемся в вершину 1. Получаем W(e3,1)=4;

Двигаемся в вершину 4. Получаем W(e4,1)=3;

Двигаемся в вершину 6. Получаем W(e4,6)=10;

Двигаемся в вершину 5. Получаем W(e6,5)=6;

Двигаемся в вершину 2. Получаем W(e2,5)=11;

Итоговый вес: 36.

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

Алгоритм Крускала — алгоритм построения минимального остовного дерева взвешенного связного неориентированного графа. Алгоритм основан на жадной стратегии: на каждом шаге алгоритм выбирает ребро с минимальным весом из всех доступных, которое не образует цикл с уже выбранными ребрами.

1. Сортируем все ребра по возрастанию веса:

(e0,3) = 2

(e1,0) = 5

(e1,3) = 4

(e1,4) = 5

(e2,3) = 7

(e2,5) = 11

(e3,5) = 6

(e4,1) = 3

(e4,6) = 10

(e5,3) = 6

(e5,6) = 2

(e5,2) = 10

(e6,4) = 2

(e6,5) = 6

1. Создаем пустой список ребер MST (минимального остовного дерева).
2. Проходим по отсортированным ребрам, добавляя каждое ребро в список MST, если оно не создает цикла с уже добавленными ребрами.

(e0,3) добавляем, MST = {(e0,3)}

(e1,0) добавляем, MST = {(e0,3), (e1,0)}

(e1,3) пропускаем, так как добавление приведет к циклу

(e1,4) добавляем, MST = {(e0,3), (e1,0), (e1,4)}

(e2,3) пропускаем, так как добавление приведет к циклу

(e2,5) добавляем, MST = {(e0,3), (e1,0), (e1,4), (e2,5)}

(e3,5) пропускаем, так как добавление приведет к циклу

(e3,6) пропускаем, так как добавление приведет к циклу

(e4,1) пропускаем, так как добавление приведет к циклу

(e4,3) пропускаем, так как добавление приведет к циклу

(e4,6) добавляем, MST = {(e0,2), (e0,3), (e1,0), (e1,4), (e2,5), (e4,6)} (e5,3)

(e5,6) добавляем, MST = {(e0,3), (e1,0), (e1,4), (e2,5), (e4,6), (e5,6)}

(e5,2) пропускаем, так как добавление приведет к циклу

(e6,4) добавляем, MST = {(e0,3), (e1,0), (e1,4), (e2,5), (e4,6), (e5,6), (e6,4)}

(e6,5) пропускаем, так как добавление приведет к циклу

Таким образом, получили минимальное остовное дерево: {(e0,3), (e1,0), (e1,4), (e2,5), (e4,6), (e5,6), (e6,4)}.

Каждое ребро в дереве соответствует одной из вершин: {0, 1, 2, 3, 4, 5, 6}, а матрица смежности показывает, какие вершины соединены ребрами. Выводим соответствующие пары вершин для каждого ребра: (e0,3), (e1,0), (e1,4), (e2,5), (e4,6), (e5,6), (e6,4)

Это означает, что минимальное остовное дерево содержит следующие ребра:

(0, 3) веса 2

(1, 0) веса 5

(1, 4) веса 5

(2, 5) веса 11

(4, 6) веса 10

(5, 6) веса 2

(6, 4) веса 2

Общий вес – 37.

**Лабораторная работа 7. Сетевые модели**

**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.



**Задание 1. Структурное 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.

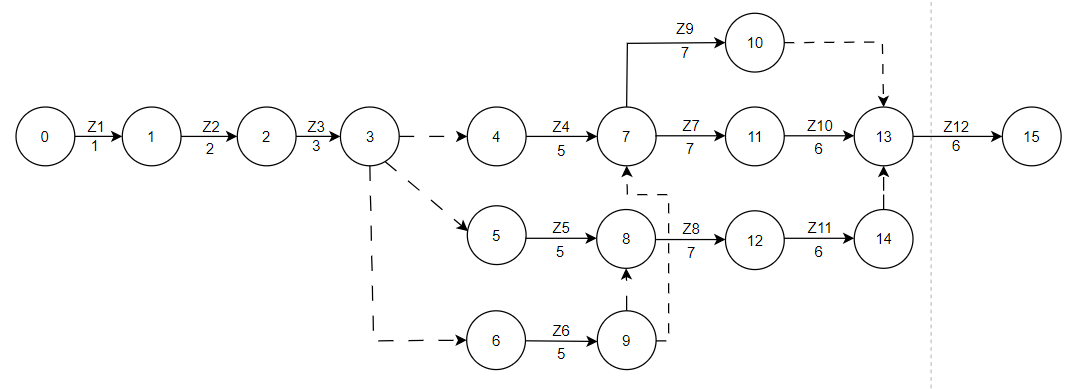
**Задание 2. Календарное планирование.**

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 выполнения** |
| I. АНАЛИЗ | | | |
| Z1 | Исследование требований к хранилищу |  | 1 |
| Z2 | Анализ существующих решений и технологий | Z1 | 2 |
| Z3 | Анализ возможностей облачных хранилищ | Z2 | 3 |
| II. ПРОЕКТИРОВАНИЕ | | | |
| Z4 | Проектирование структуры базы данных | Z3 | 5 |
| Z5 | Проектирование пользовательского интерфейса | Z3 | 5 |
| Z6 | Проектирование механизмов защиты данных | Z3 | 5 |
| III. РАЗРАБОТКА | | | |
| Z7 | Разработка серверной части хранилища | Z4, Z6 | 7 |
| Z8 | Разработка клиентской части хранилища | Z5, Z6 | 7 |
| Z9 | Разработка механизмов резервного копирования | Z4, Z6 | 7 |
| IV. ТЕСТИРОВАНИЕ | | | |
| Z10 | Тестирование серверной части хранилища | Z7 | 6 |
| Z11 | Тестирование клиентской части хранилища | Z8 | 6 |
| V. ВНЕДРЕНИЕ | | | |
| Z12 | Развёртывание облачного хранилища | Z10, Z11, Z9 | 6 |

**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.



Критический путь проекта может быть представлен следующей последовательностью операций:

I -> Z1 -> Z2 -> Z3 -> Z4 -> Z7 -> Z10 -> Z12

Общее время выполнения проекта можно определить как сумму времени выполнения всех операций в критическом пути:

1 + 2 + 3 + 5 + 7 + 6 + 6 = 30 дней.

**\*Задание 4. Оптимизация**

* Инвестиции: Привлечение дополнительных денежных средств от инвесторов может помочь расширить проект, улучшить его функциональность и повысить производительность. Инвесторы могут внести капитал в проект в обмен на долю прибыли или долю в компании. Это позволит финансировать дальнейшую разработку и масштабирование облачного хранилища.
* Партнерство с поставщиками облачных услуг: Установление партнерских отношений с крупными провайдерами облачных услуг, такими как Amazon Web Services, Microsoft Azure или Google Cloud, может помочь в привлечении дополнительного финансирования и ресурсов. Это также может обеспечить доступ к инфраструктуре и технологиям, которые могут значительно ускорить разработку и внедрение облачного хранилища.
* Наем новых специалистов: При привлечении дополнительного человеческого ресурса можно повысить эффективность работы и ускорить разработку проекта. Найм специалистов с опытом в области облачных технологий и разработки может привести к повышению производительности, оптимизации кода, а также развитию новых функциональностей для облачного хранилища.

**Ответы на вопросы:**

Основные разновидности методов сетевого планирования:

* метод критического пути (Critical Path Method - СРМ)
* метод оценки и обзора программ (Program Evaluation and Review Technique - PERT).

Метод критического пути (СРМ) применяется тогда, когда операции, входящие в состав комплекса работ, имеют известные строго определенные продолжительности (являются детерминированными).

В свою очередь, метод оценки и обзора программ (РЕRТ) применяется при планировании проектов, для которых характерна неопределенность в оценке затрат времени, необходимого для выполнения отдельных операций

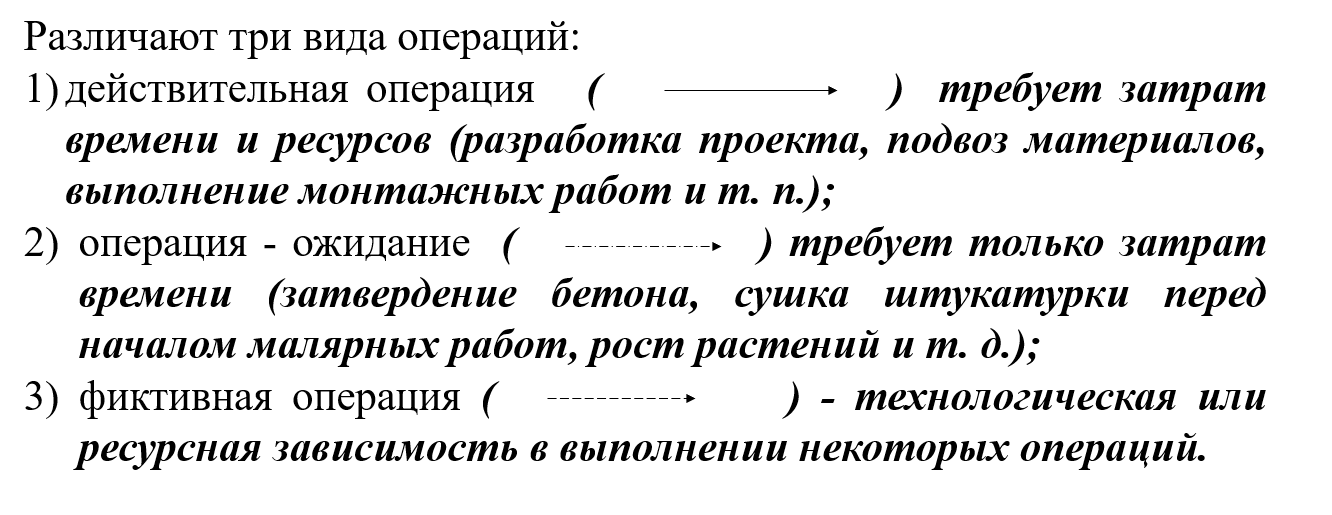
СПУ включает три основных этапа:

* Структурное планирование
* Календарное планирование
* Оперативное управление.

Сетевой моделью называется модель, отражающая комплекс работ (операций) и событий, связанных с реализацией некоторого проекта в их логической и технологической последовательности и связи.

Различают три вида событий: исходное, завершающее и промежуточное.

Сетевые графики с несколькими завершающими событиями называются многоцелевыми.

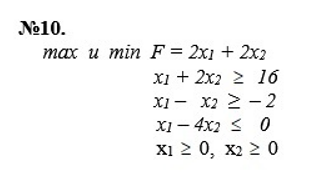


Разница между событием и операцией состоит в том, что моментом свершения события считается момент окончания выполнения всех входящих в это событие операций

Коэффициентом дополнительных затрат называется такой коэффициент, который показывает, насколько увеличится стоимость работы при уменьшении её продолжительности в единицу времени (при оптимизации проекта по стоимости).

**Лабораторная работа 8. Графический метод решения оптимизационных задач**

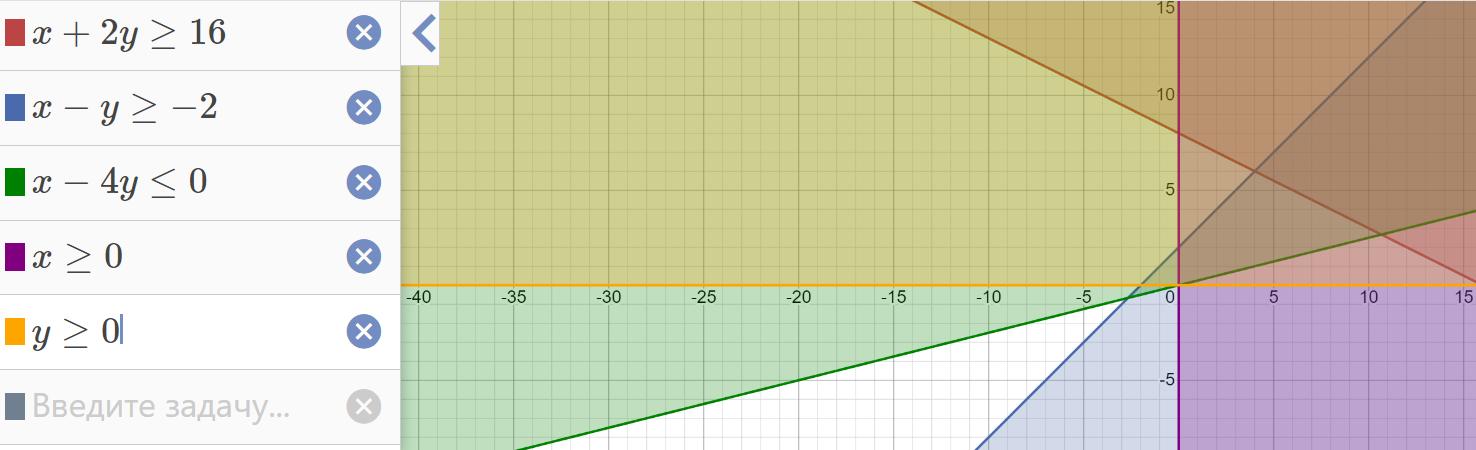
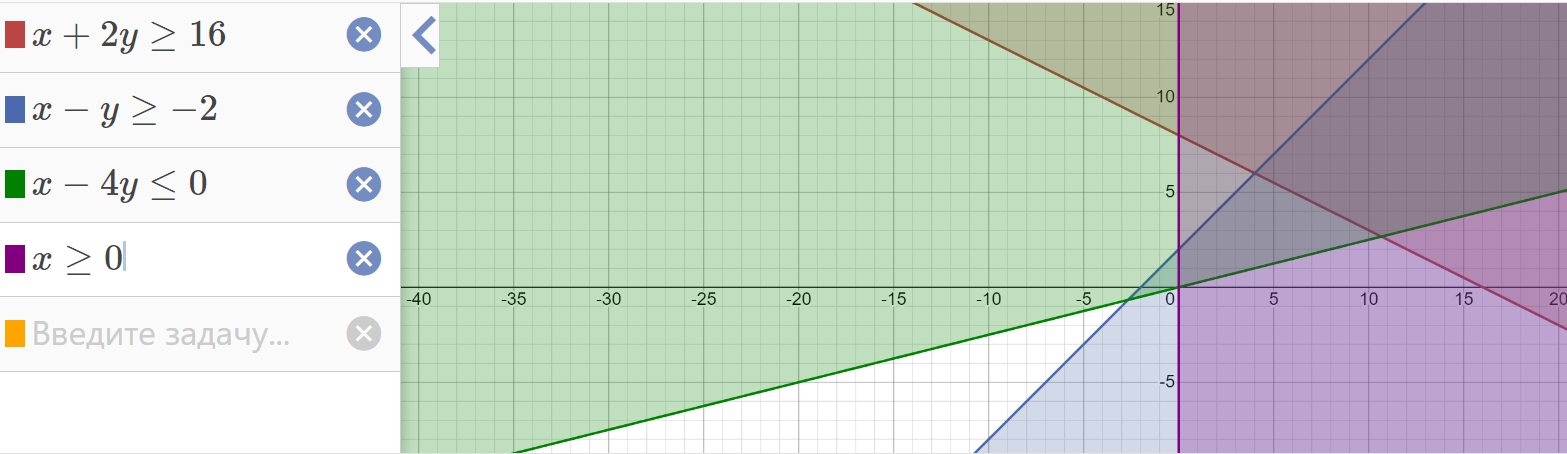
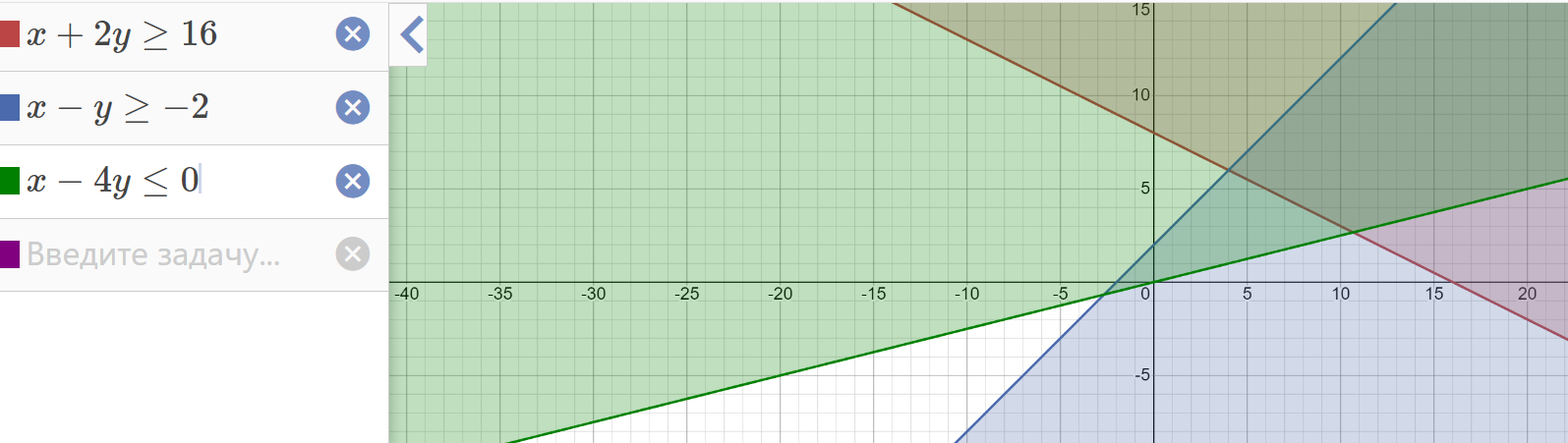
**Цель работы:** Освоить решение задач графическим методом.

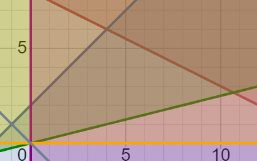


1. Строим область допустимых решений, т.е. решаем графически систему неравенств. Для этого строим каждую прямую и определяем полуплоскости, заданные неравенствами.

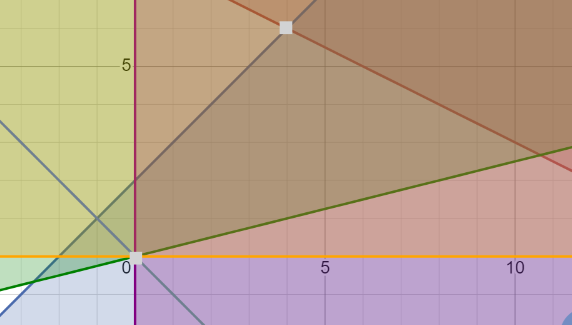








2. Строим прямую, соответствующую задаче, или целевой функции, приравненной к нулю. Область допустимых решений может представлять бесконечное множество. Поэтому ищем max и min в области ограничений.



По рисунку определяем, что min = (0; 0), max = (6; 4).

F(min) = 0; F(max) = 2\*6 + 2\*4 = 20.