Отчёт

По лабораторным работам

По предмету: Математическое программирование

Студент

Авчинник Е. С.

ФИТ 2 курс, 1/2 группа

Преподаватель

Бурмакова А. В.

Оглавление

[**Лабораторная работа 1. Вспомогательные функции** 3](#_Toc196314533)

[**Задание 1.** 3](#_Toc196314534)

[**Задание 2.** 4](#_Toc196314535)

[**Задание 3** 5](#_Toc196314536)

[**Лабораторная работа 2. Комбинаторный алгоритмы** 7](#_Toc196314537)

[**Задание 1.** Разобрать и разработать генератор подмножеств заданного множества. 7](#_Toc196314538)

[**Задание 2.** Разобрать и разработать генератор Блчетаний. 9](#_Toc196314539)

[**Задание 3**. Разобрать и разработать генератор перестановок. 10](#_Toc196314540)

[**Задание 4** Разобрать и разработать генератор размещений 12](#_Toc196314541)

[**Задание 5** Решить задачу коммивояжера 14](#_Toc196314542)

[**Задание 6** Исследовать зависимость времени вычисления необходимое для решения задачи 15](#_Toc196314543)

[**Лабораторная работа 3** 19](#_Toc196314544)

[**Задание 1.** 19](#_Toc196314545)

[**Задание 2.** 19](#_Toc196314546)

[**Задание 3** 24](#_Toc196314547)

[**Лабораторная работа 4** 26](#_Toc196314548)

[**Лабораторная работа 5** 38](#_Toc196314549)

[**Лабораторная работа 6 ВАРИАНТ 1** 45](#_Toc196314550)

[**Лабораторная работа 7** 50](#_Toc196314551)

[**Лабораторная работа 8** 53](#_Toc196314552)

# **Лабораторная работа 1. Вспомогательные функции**

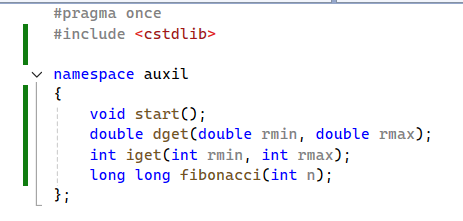
**ЦЕЛЬ РАБОТЫ:** приобретение навыков Блставления и отладки программ с использованием пользовательских функций для замера продолжительности процесБе вычисления.

## **Задание 1.**

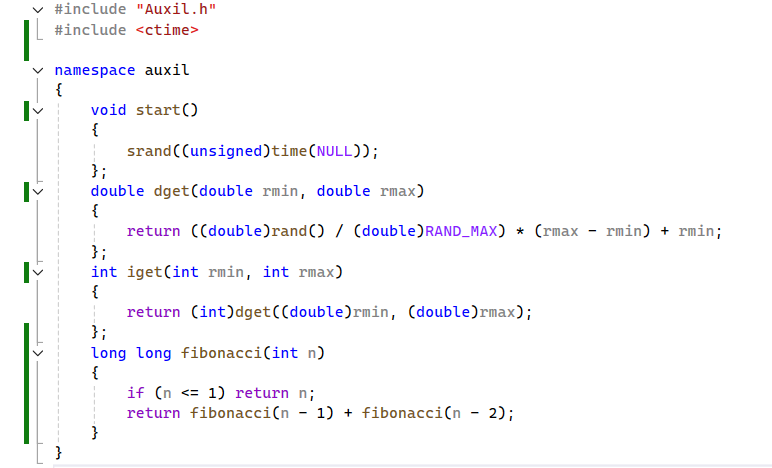
Разработайте три функции (start, dget и iget), используя следующие спецификации. **Примечание**: разработанные функции должны располагаться в файле **Auxil.cpp**,  а в файле **Auxil.h –** прототипы функций.



1. pch.h – файл, Блдержащий часто встречаемые заголовочные файлы для ускорения компиляции
2. pch.cpp – файл, Блдержащий единственную строку #include "pch.h"
3. Auxil.h – заголовочный файл, который Блдержит прототипы функций start, dget и iget:



1. Auxil.cpp – файл с реализацией функций start, dget и iget:



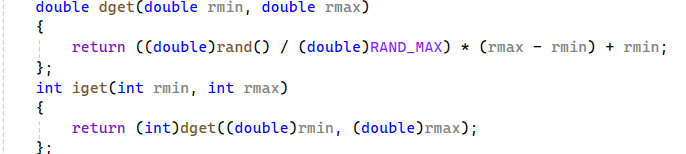
1. main.cpp – файл с функцией main(), где находятся экземпляры класБе и вывод всех заданий.

Реализация функций генератора псевдослучайных чисел

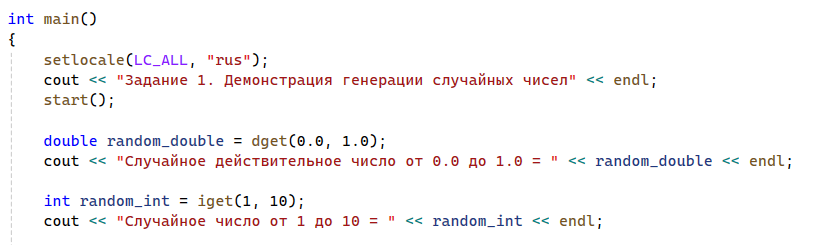
1. Инициализация генератора случайных чисел в помощью метода start(). Функция устанавливает в качестве начального числа для генератора псевдослучайных чисел текущее значение системного времени в формате функции time().



1. Генерируем псевдослучайные числа. Функция **dget()** возвращает действительное псевдослучайное число в диапазоне от **rmin** до **rmax.** Функция i**get()** возвращает псевдослучайное число в заданном диапазоне от **rmin** до **rmax.**



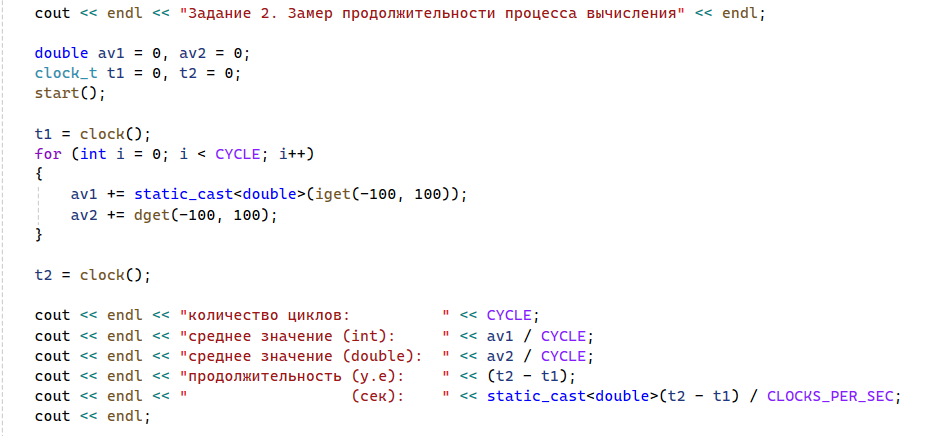
1. Вывод псевдослучайных чисел в функции main:





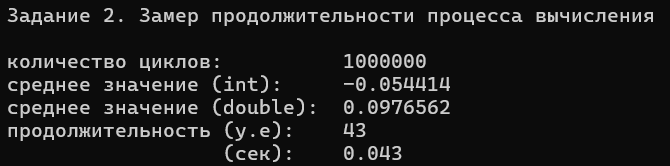
## **Задание 2.**

Реализовать пример 2.Для проверки работоспоБлбности разработанных функций и приобретения навыков замера продолжительности процесБе вычисления реализуйте программу, приведенную в примере 2.



Инициализируем генератор случайных чисел с помощью функции start(). Используем функцию clock() для замера времени выполнения. Время начала замеряется с помощью t1 = clock(), а время окончания – с помощью t2 = clock().

Среднее значение и продолжительность вычисления выводятся в конБлль.

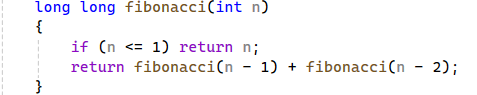


**Задание 3**

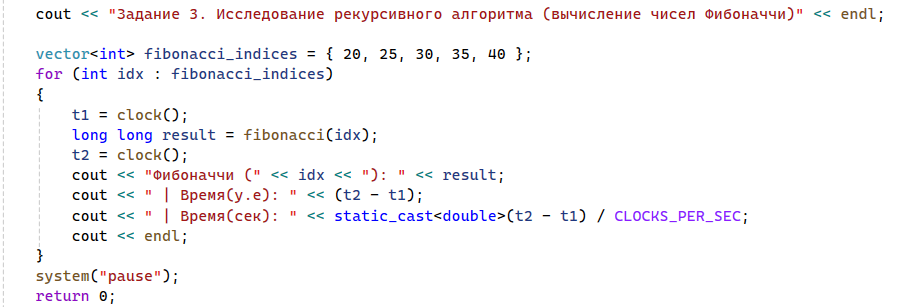
Проведите необходимые эксперименты и постройте график зависимости (Excel) продолжительности процесБе вычисления от количества циклов в примере 2. Проанализируйте характер зависимости. Проведите исследование любого другого рекурсивного алгоритма, например, вычисления факториала или генератора чисел Фибоначчи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

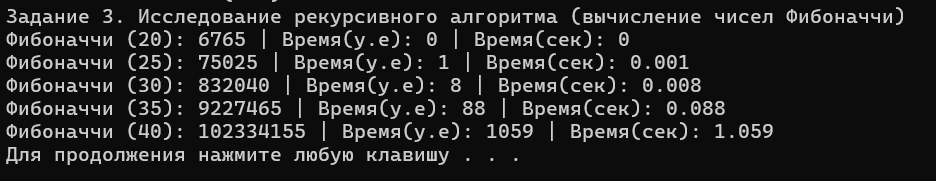
**Примечание**: продолжительность вычисления измерять в условных единицах процесБлрного времени (функция **clock**).

1. Реализация метода для вычисления чисел Фибоначчи:



1. Вычисление значений чисел Фибоначчи для различных индекБлв, и измерение времени выполнения для каждого вычисления.







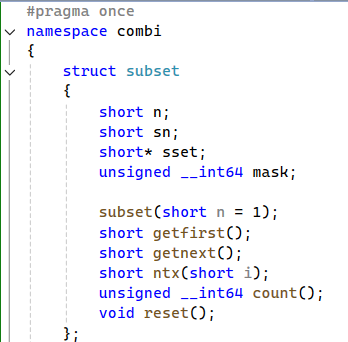


Оба графика показывают что с увеличением данных увеличивается время выполнения программы.

**Лабораторная работа 2. Комбинаторный алгоритмы**

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

1. Необходимо было Блздать структуру subset, которая будет генерировать подмножества заданного множества.



2. Поля и методы структуры:

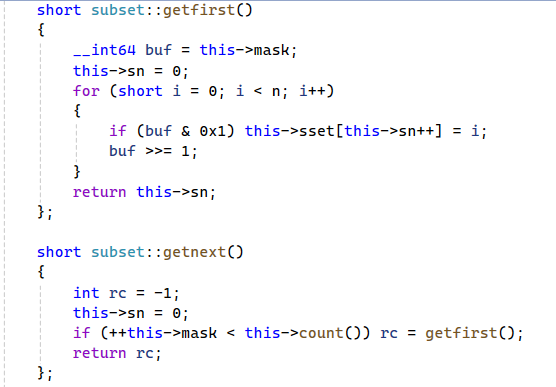
1. short n: количество элементов исходного множества.
2. short sn: количество элементов текущего подмножества.
3. short\* sset: массив индекБлв текущего подмножества.
4. unsigned \_\_int64 mask: битовая маска для генерации подмножеств.
5. Конструктор subset(short n = 1): инициализация структуры.
6. short getfirst(): формирование массива индекБлв по битовой маске.
7. short getnext(): увеличение маски и формирование массива индекБлв.
8. short ntx(short i): получение i-го элемента массива индекБлв.
9. unsigned \_\_int64 count(): вычисление общего количества подмножеств.
10. void reset(): сброс генератора, чтобы начать сначала.

3. Блздаём файл Combi.cpp, который будет Блдержать реализацию методов структур, используемых для генерации подмножеств.

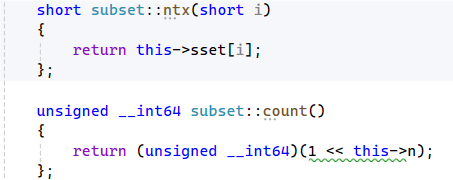
Конструктор инициализирует структуру subset с количеством элементов n и выделяет память для массива индекБлв текущего подмножества. Метод reset сбрасывает Блстояние структуры, обнуляя количество элементов текущего подмножества и устанавливая битовую маску в 0.



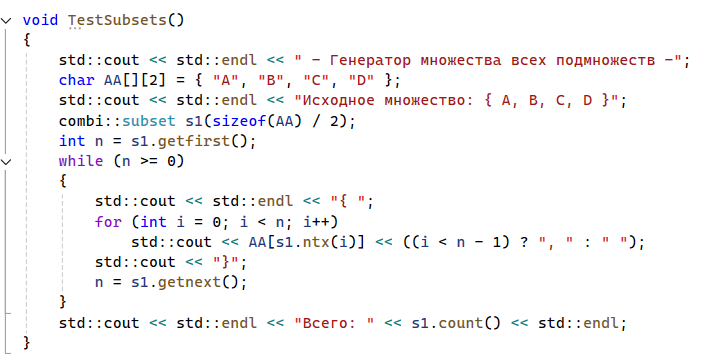
Метод getfirst формирует массив индекБлв текущего подмножества на основе битовой маски. Он проходит по битам маски и, если бит установлен, добавляет Блответствующий индекс в массив. Метод getnext увеличивает битовую маску и вызывает метод getfirst для формирования нового подмножества. Если маска превышает количество подмножеств, метод возвращает -1.

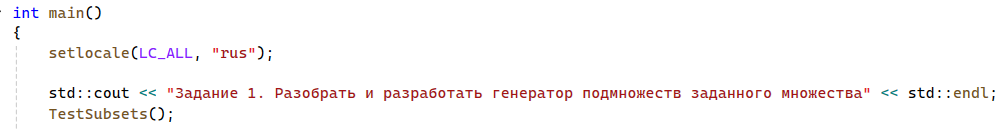


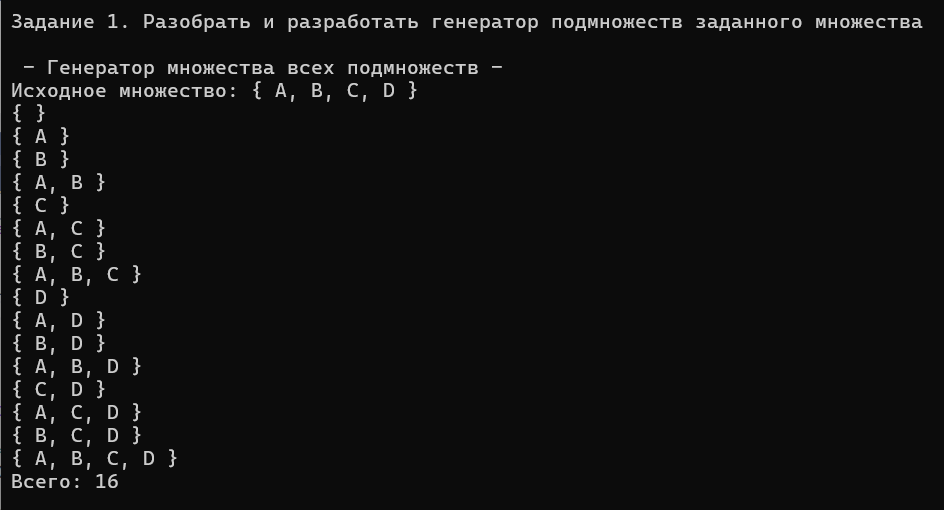
Метод ntx возвращает i-й элемент массива индекБлв текущего подмножества. Метод count возвращает общее количество подмножеств, которое равно 2^n.



4. Вызываем функции генератора подмножества в основном файле

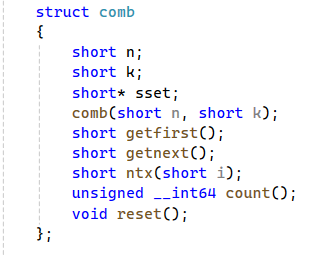






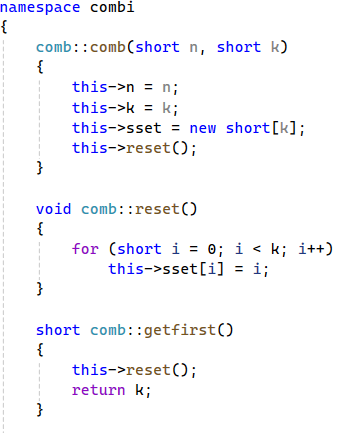
**Задание 2.** Разобрать и разработать генератор Блчетаний.

1. Структура comb, которая будет генерировать Блчетания из заданного множества.

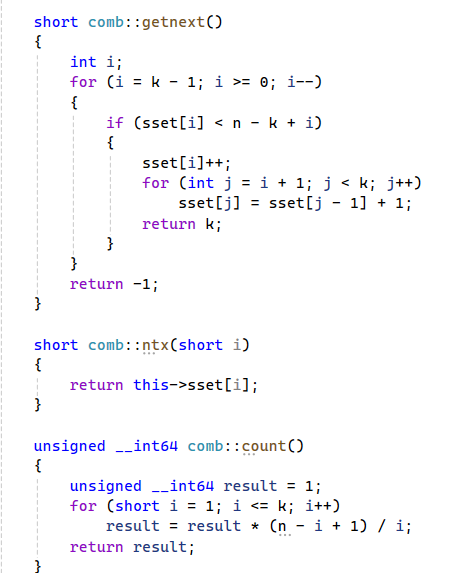


1. Поля и методы структуры.
2. short n: количество элементов исходного множества.
3. short k: количество элементов в Блчетании.
4. short\* sset: массив индекБлв текущего Блчетания.
5. Конструктор comb(short n, short k): инициализация структуры.
6. short getfirst(): формирование первого Блчетания.
7. short getnext(): генерация следующего Блчетания.
8. short ntx(short i): получение i-го элемента массива индекБлв.
9. unsigned \_\_int64 count(): вычисление общего количества Блчетаний.
10. void reset(): сброс генератора, чтобы начать сначала.
11. Реализация структуры comb для генерации Блчетаний

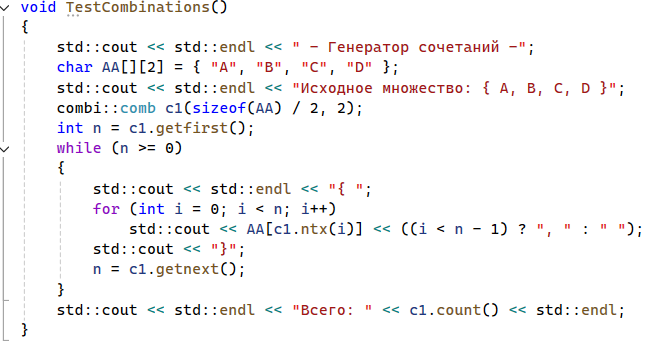
Конструктор инициализирует структуру comb с количеством элементов n и длиной Блчетания k. Метод reset инициализирует массив индекБлв начальными значениями. Метод getfirst сбрасывает генератор и возвращает количество элементов в Блчетании.

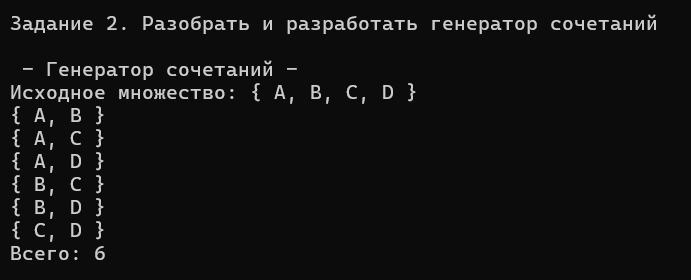


Метод getnext генерирует следующее Блчетание, увеличивая индексы элементов. Метод ntx возвращает i-й элемент массива индекБлв текущего Блчетания. Метод count возвращает общее количество Блчетаний, которое вычисляется как n! / (k! \* (n-k)!).



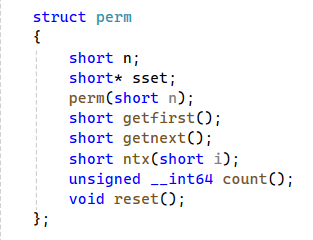
1. Вызываем функцию в основном файле



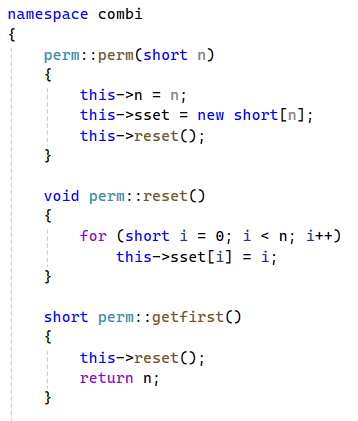


## **Задание 3**. Разобрать и разработать генератор перестановок.

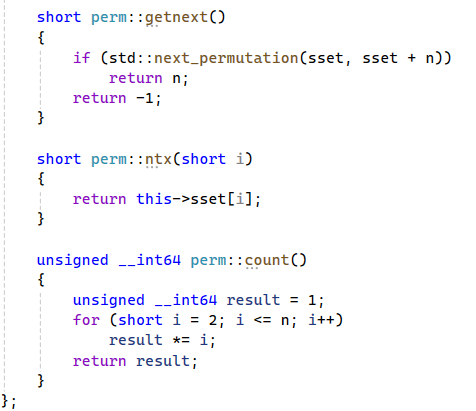
1. Структура perm



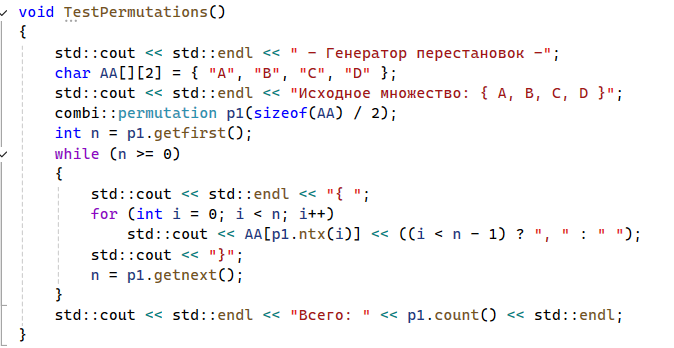
Конструктор инициализирует структуру perm с количеством элементов n. Метод reset инициализирует массив индекБлв начальными значениями. Метод getfirst сбрасывает генератор и возвращает количество элементов в перестановке.

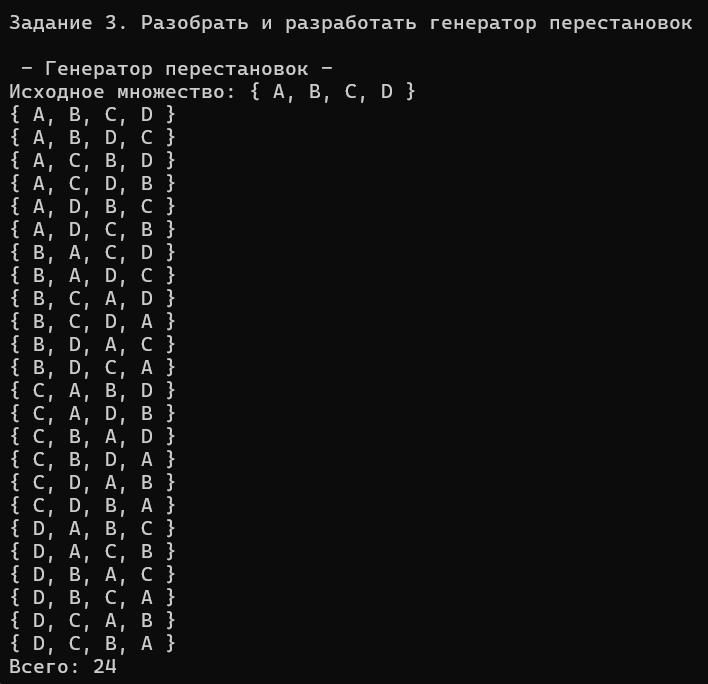


Метод getnext генерирует следующую перестановку, используя функцию std::next\_permutation. Метод ntx возвращает i-й элемент массива индекБлв текущей перестановки. Метод count возвращает общее количество перестановок, которое вычисляется как n!.



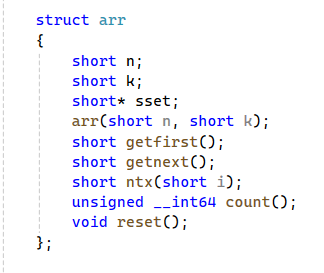
1. Вызов генерации в главном файле



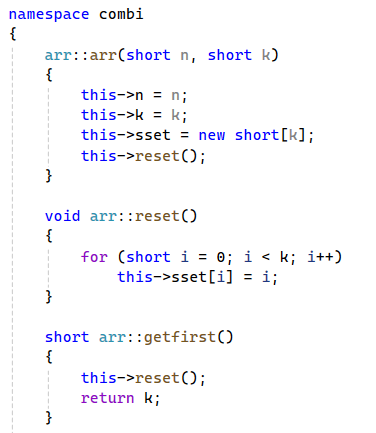


**Задание 4** Разобрать и разработать генератор размещений

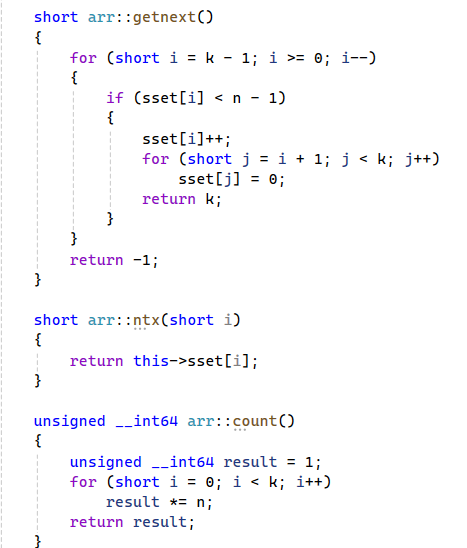
1. Структура arr



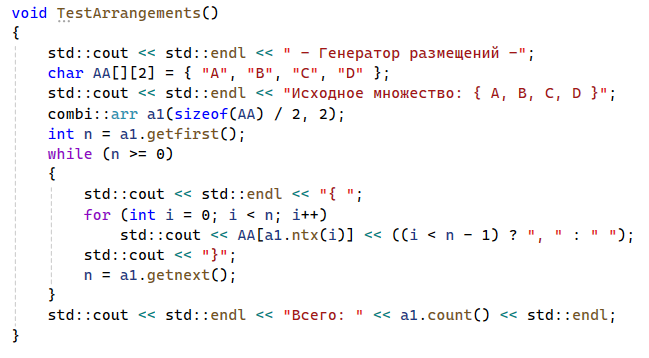
Конструктор инициализирует структуру arr с количеством элементов n и длиной размещения k. Метод reset инициализирует массив индекБлв начальными значениями. Метод getfirst сбрасывает генератор и возвращает количество элементов в размещении.

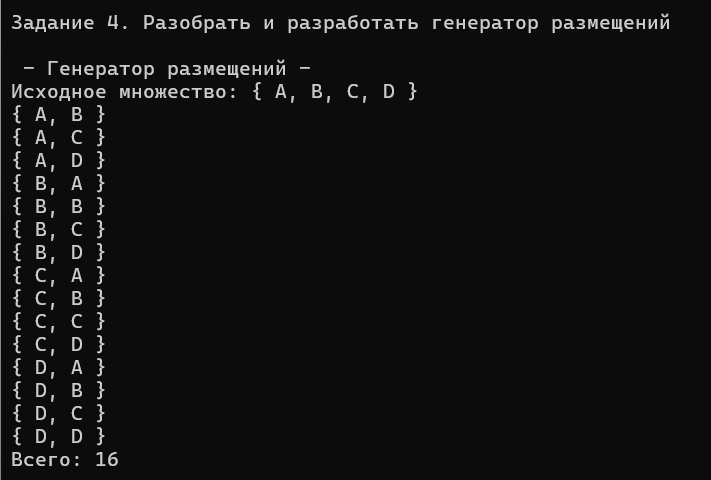


Метод getnext() Метод getnext генерирует следующее размещение, увеличивая индексы элементов. Метод ntx возвращает i-й элемент массива индекБлв текущего размещения. Метод count возвращает общее количество размещений, которое вычисляется как n^k.



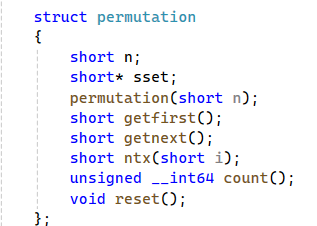
1. Реализация



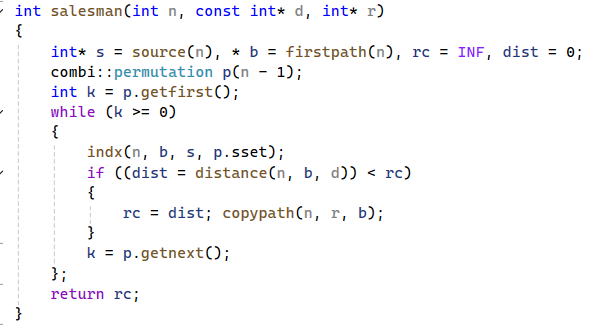


**Задание 5** Решить задачу коммивояжера

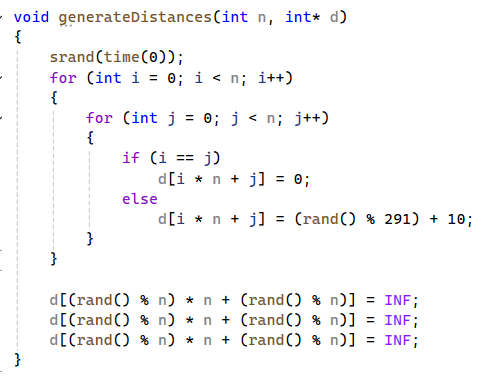
1. Структура permutation, которая генерирует перестановки городов для решения задачи коммивояжера.



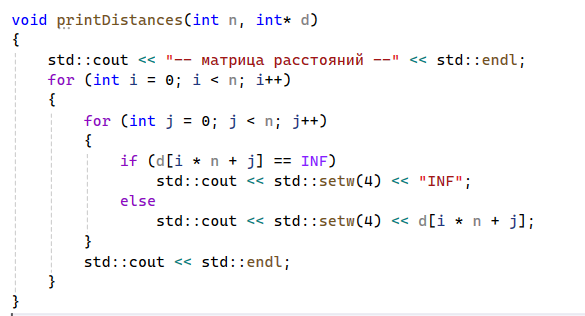
2. Определение функции salesman, которая находит оптимальный маршрут коммивояжера, используя перестановки и вычисляя расстояния между городами.

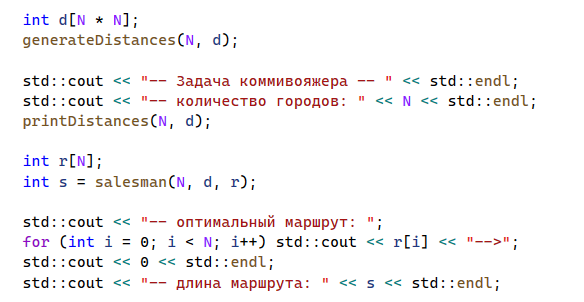


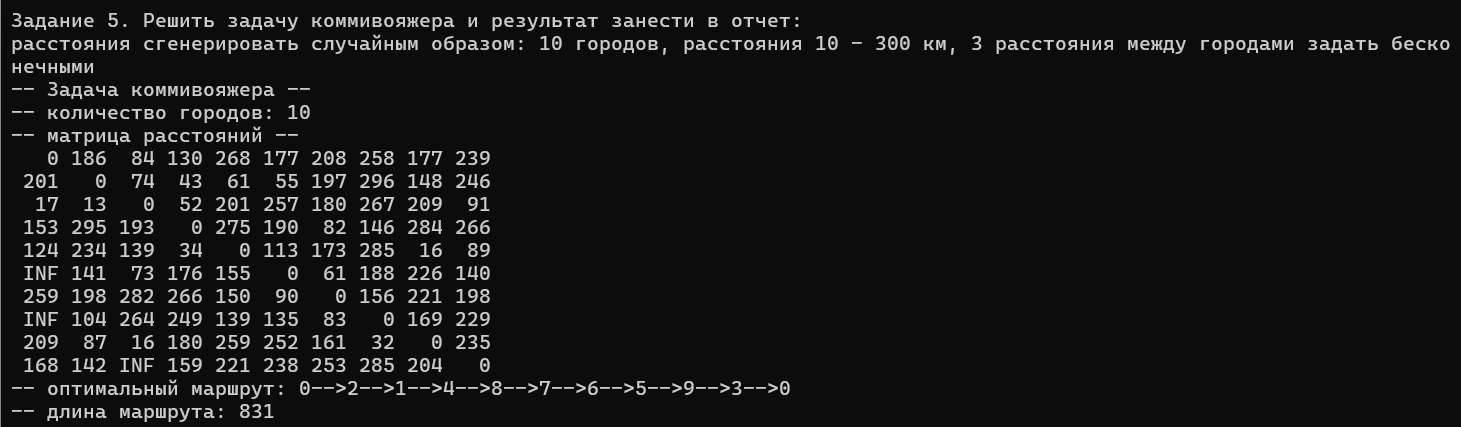
1. Блздание функции generateDistances, которая случайным образом генерирует расстояния между 10 городами в диапазоне от 10 до 300 км, и задает три расстояния как бесконечные.



1. Реализация задачи

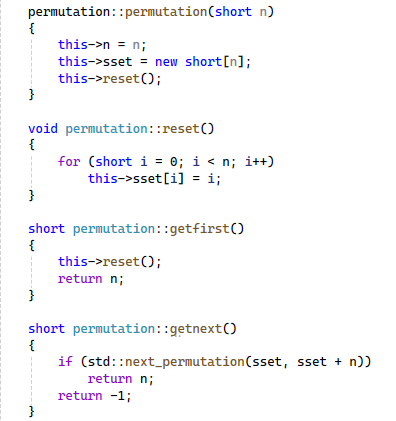
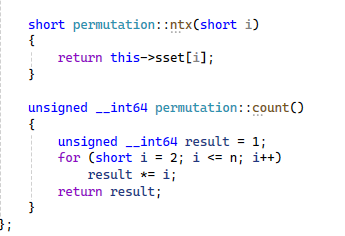




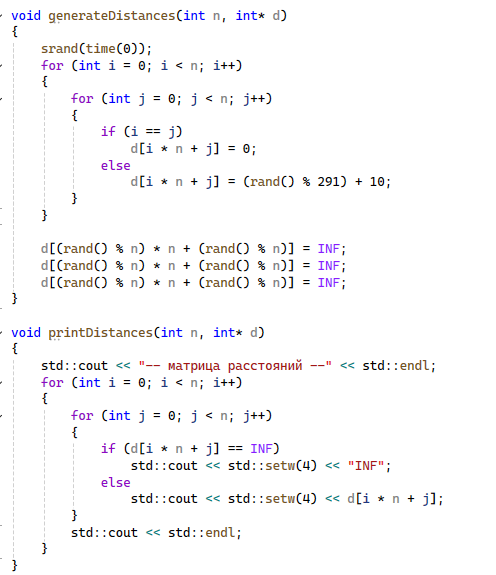


**Задание 6** Исследовать зависимость времени вычисления необходимое для решения задачи

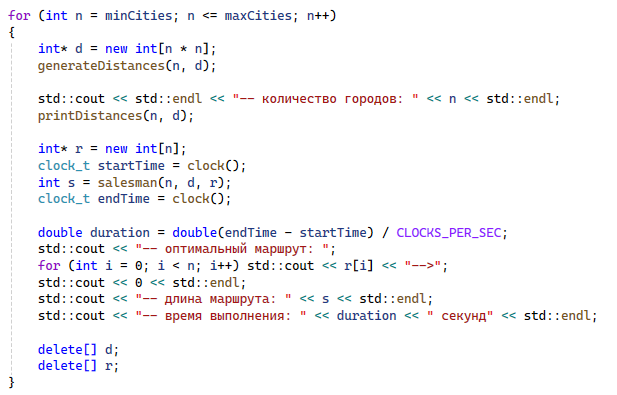
1. Выполнено тестирование задачи коммивояжера для различных количеств городов в диапазоне от 6 до 12, время выполнения при каждом количестве городов было замерено и построен график зависимости времени вычисления от количества городов. Добавлен код для замера времени выполнения задачи коммивояжера для различных количеств городов.

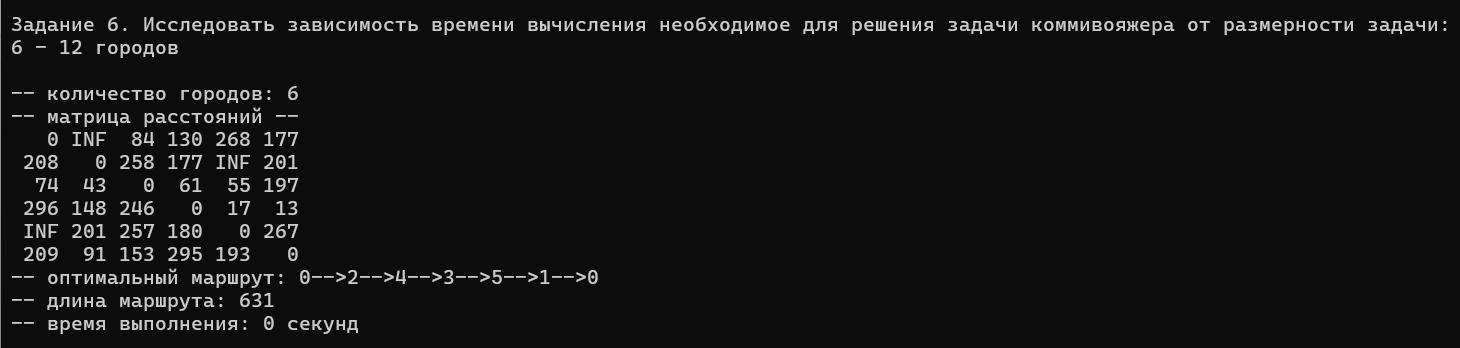
Использована функцию generateDistances для генерации случайных расстояний между городами.



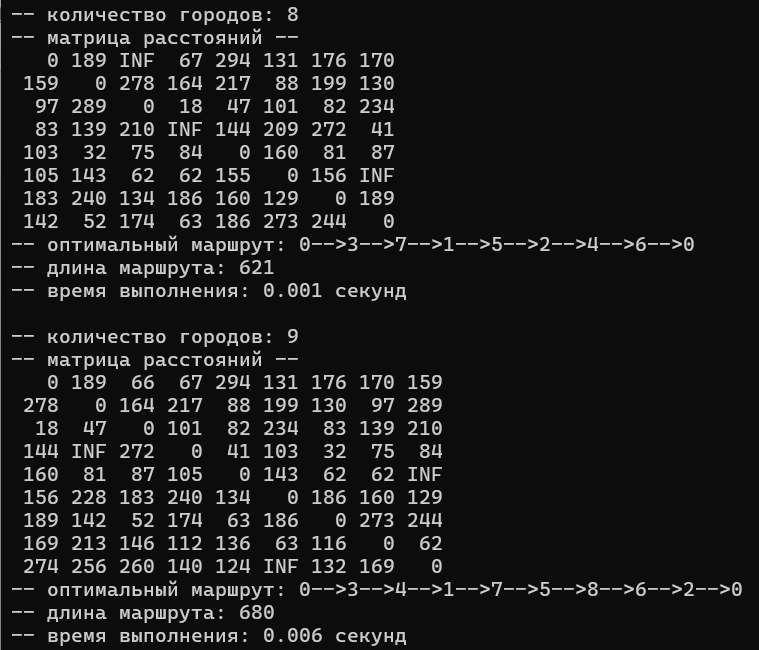
Использование функции clock, startTime, endTime для замера времени выполнения задачи.



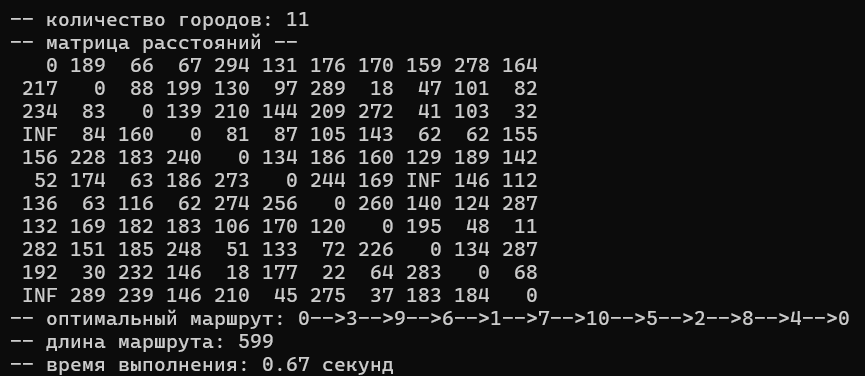
2. Запуск программы

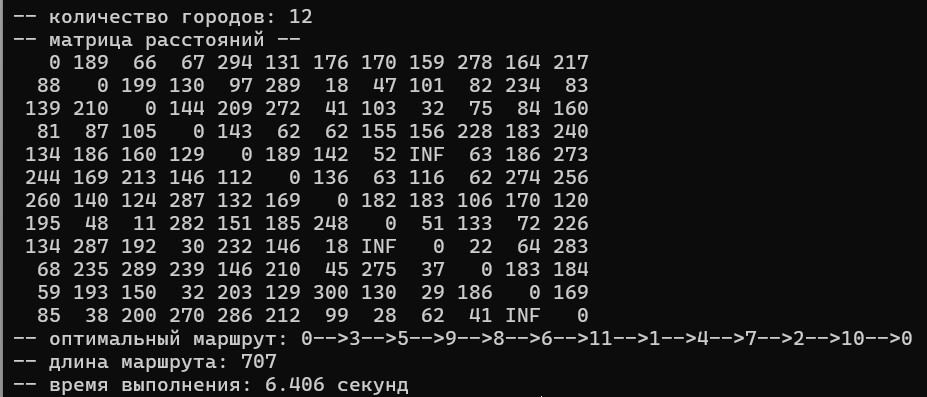












3. График зависимости времени выполнения программы о количества городов

Вывод: с увеличением городов увеличивается время выполнения программы

**Лабораторная работа 3**

## **Задание 1.**

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;

Матрица расстояний при n=1:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Город** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 2 | 22 |  | 1 |
| **2** | 1 |  | 16 | 67 | 83 |
| **3** | 3 | 3 |  | 86 | 50 |
| **4** | 18 | 57 | 4 |  | 3 |
| **5** | 92 | 67 | 52 | 14 |  |

Коммивояжеру нужно посетить каждый из этих 5 городов ровно один раз и вернуться в исходный город, при этом минимизировав общее расстояние пути. Необходимо найти маршрут, который будет минимизировать общую длину пути с учетом приведенной матрицы расстояний.

Для этого мы будем использовать метод ветвей и границ, а затем сравним полученное решение с комбинаторным методом перестановок.

## **Задание 2.**

Решить сформулированную задачу методом ветвей и границ.

**Решение методом ветвей и границ:**

В Блответствии с утверждением 1: если все элементы первой строки таблицы уменьшить на 1 (наименьшее значение в строке), то это не повлияет на порядок городов в кратчайшем кольцевом маршруте, проходящем через все города по одному разу, а лишь Блкратит его длину на 1.

Приведение таблицы по строке:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** | **αi** |
| **1** |  | 2 | 22 |  | 1 | **-1** |
| **2** | 1 |  | 16 | 67 | 83 | **-1** |
| **3** | 3 | 3 |  | 86 | 50 | **-3** |
| **4** | 18 | 57 | 4 |  | 3 | **-3** |
| **5** | 92 | 67 | 52 | 14 |  | **-14** |

22

Результат:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 1 | 21 |  | 0 |
| **2** | 0 |  | 15 | 66 | 82 |
| **3** | 0 | 0 |  | 83 | 47 |
| **4** | 16 | 54 | 1 |  | 0 |
| **5** | 78 | 53 | 38 | 0 |  |
| **Вычет** | 0 | 0 | 1 | 0 | 0 |

В Блответствии с утверждением 2: если все элементы второго столбца таблицы уменьшить на 1 (наименьшее значение в столбце), то это не повлияет на порядок городов в кратчайшем кольцевом маршруте, проходящем через все города по одному разу, а лишь Блкратит его длину на 1.

Результат приведения таблицы по столбцу:

1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 1 | 20 |  | 0 |
| **2** | 0 |  | 14 | 66 | 82 |
| **3** | 0 | 0 |  | 83 | 47 |
| **4** | 16 | 54 | 0 |  | 0 |
| **5** | 78 | 53 | 37 | 0 |  |
| **βi** | 0 | 0 | 1 | 0 | 0 |

**γ = 1+22=23 –** сумма констант приведения

**φ(R) = γ(R) –** нижняя граница длины маршрута

**φ = 1+22 = 23 –** нижняя граница длины кратчайшего кольцевого маршрута

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 1 | 20 |  | 0 |
| **2** | 0 |  | 14 | 66 | 82 |
| **3** | 0 | 0 |  | 83 | 47 |
| **4** | 16 | 54 | 0 |  | 0 |
| **5** | 78 | 53 | 37 | 0 |  |

Дальнейшие операции производятся с полностью приведённой таблицей. Анализируем дуги имеющие нулевой вес, так как только их удаление может повлиять на нижнюю границу.

**Шаг 1**. Заменяем на **INF** **ноль** в первой строке, тем Бемым проверяю дугу (1,5) и вычисляем сумму констант уже для этой таблицы.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 1 | 20 |  |  |
| **2** | 0 |  | 14 | 66 | 82 |
| **3** | 0 | 0 |  | 83 | 47 |
| **4** | 16 | 54 | 0 |  | 0 |
| **5** | 78 | 53 | 37 | 0 |  |

14

1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 1 | 20 |  |  |
| **2** |  |  | 14 | 66 | 82 |
| **3** | 0 | 0 |  | 83 | 47 |
| **4** | 16 | 54 | 0 |  | 0 |
| **5** | 78 | 53 | 37 | 0 |  |

0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 1 | 20 |  |  |
| **2** |  |  | 14 | 66 | 82 |
| **3** |  | 0 |  | 83 | 47 |
| **4** | 16 | 54 | 0 |  | 0 |
| **5** | 78 | 53 | 37 | 0 |  |

1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 1 | 20 |  |  |
| **2** |  |  | 14 | 66 | 82 |
| **3** |  |  |  | 83 | 47 |
| **4** | 16 | 54 | 0 |  | 0 |
| **5** | 78 | 53 | 37 | 0 |  |

14

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 1 | 20 |  |  |
| **2** |  |  | 14 | 66 | 82 |
| **3** |  |  |  | 83 | 47 |
| **4** | 16 | 54 |  |  | 0 |
| **5** | 78 | 53 | 37 | 0 |  |

0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 1 | 20 |  |  |
| **2** |  |  | 14 | 66 | 82 |
| **3** |  |  |  | 83 | 47 |
| **4** | 16 | 54 |  |  |  |
| **5** | 78 | 53 | 37 | 0 |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **4** | **5** |
| **1** |  | 1 | 20 |  |  |
| **2** |  |  | 14 | 66 | 82 |
| **3** |  |  |  | 83 | 47 |
| **4** | 16 | 54 |  |  |  |
| **5** | 78 | 53 | 37 |  |  |

103

Все допустимые кольцевые маршруты не будут включать путь из города 5 в город 4, а длина этих кольцевых маршрутов не будет меньше, чем нижняя граница, построенная для  ***полностью приведенной таблицы***.

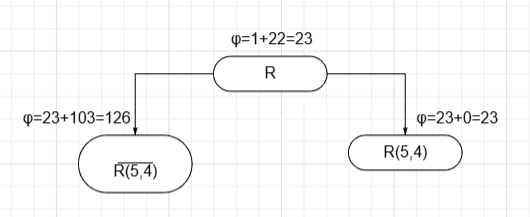
*Кроме того, очевидно, что допустимый кольцевой маршрут, Блдержащий дугу (5, 4), не может Блдержать дугу (4, 5).*

Вычеркиваем 5 строку и 4 столбец, а значение дуги (4, 5) заменяем на INF и считаем

0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **5** |
| **1** |  | 1 | 20 |  |
| **2** |  |  | 14 | 82 |
| **3** |  |  |  | 47 |
| **4** | 16 | 54 |  |  |

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



Продолжаем применять алгоритм к новой таблице:

48

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **5** |
| **1** |  | 1 | 20 |  |
| **2** |  |  | 14 | 82 |
| **3** |  |  |  | 47 |
| **4** | 16 | 54 |  |  |

14

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **5** |
| **1** |  | 1 | 20 |  |
| **2** |  |  | 14 | 82 |
| **3** |  |  |  | 47 |
| **4** | 16 | 54 |  |  |

0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **5** |
| **1** |  | 1 | 20 |  |
| **2** |  |  | 14 | 82 |
| **3** |  |  |  | 47 |
| **4** | 16 | 54 |  |  |
| **Города** | **1** | **2** | **3** | **5** |
| **1** |  | 1 | 20 |  |
| **2** |  |  | 14 | 82 |
| **3** |  |  |  | 47 |
| **4** | 16 | 54 |  |  |

30

1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** | **5** |
| **1** |  | 1 | 20 |  |
| **2** |  |  | 14 | 82 |
| **3** |  |  |  | 47 |
| **4** | 16 | 54 |  |  |

Удаляем строку 1 и столбец 5:

|  |  |  |  |
| --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** |
| **2** |  |  | 14 |
| **3** |  |  |  |
| **4** | 16 | 54 |  |

Продолжаем применять алгоритм

14

|  |  |  |  |
| --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** |
| **2** |  |  | 14 |
| **3** |  |  |  |
| **4** | 16 | 54 |  |

0

|  |  |  |  |
| --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** |
| **2** |  |  | 14 |
| **3** |  |  |  |
| **4** | 16 | 54 |  |

54

|  |  |  |  |
| --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** |
| **2** |  |  | 14 |
| **3** |  |  |  |
| **4** | 16 | 54 |  |

30

|  |  |  |  |
| --- | --- | --- | --- |
| **Города** | **1** | **2** | **3** |
| **2** |  |  | 14 |
| **3** |  |  |  |
| **4** | 16 | 54 |  |

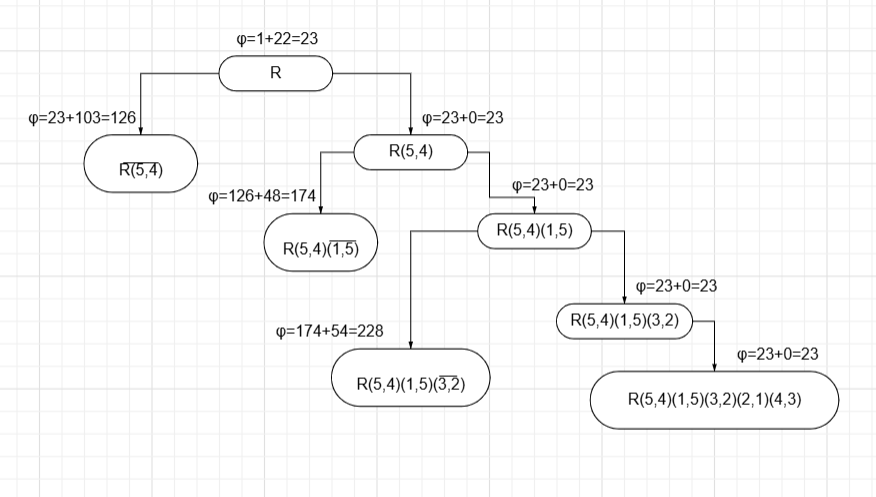
Удаляем 3 строку и 2 столбец

Дугу (2,3) делаем

|  |  |  |
| --- | --- | --- |
| **Города** | **1** | **3** |
| **2** |  |  |
| **4** | 16 |  |

Дуги (2,1) и (4,3) – два последних звена кольцевого маршрута.

0



Итоговый путь: 5 → 4 → 3 → 2 → 1→5

## **Задание 3**

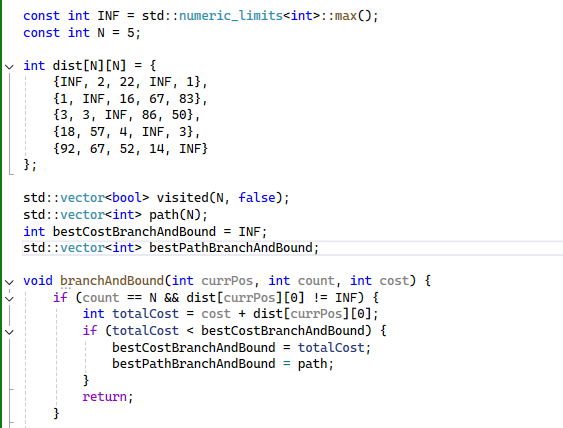
Проверить полученное решение при помощи генератора перестановок (см. лаб. 2, задание 5.1.) и включить копию экрана с решением в отчет.

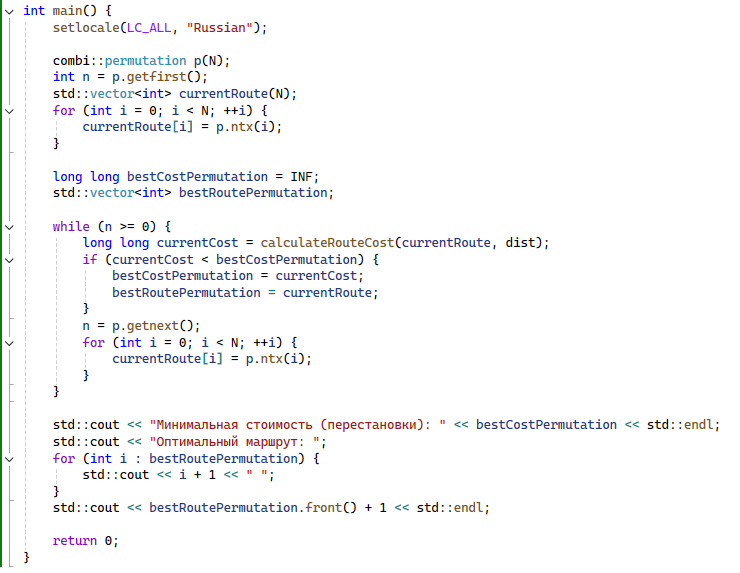
Решение при помощи генератора перестановок Блвпадает с личным решением.



Реализация алгоритма ветвей и границ для решения задачи коммивояжера:

1. Инициализация массива расстояний: определяем матрицу расстояний dist, где INF используется для обозначения недостижимых маршрутов.
2. Инициализация переменных и структуры данных: Блздаем вектор visited для отслеживания посещенных городов. Вектор path для хранения текущего пути. Переменные bestCost и bestPath для отслеживания наилучшего найденного решения.
3. Рекурсивная функция branchAndBound: функция перебирает все возможные пути с использованием рекурсии. Если путь завершен, вычисляем его общую стоимость. Обновляем наилучшее решение, если найден путь с меньшей стоимостью.
4. Вызов функции branchAndBound из main: начинаем с первого города (0). Вызываем функцию branchAndBound, которая находит наилучший путь и минимальную стоимость.





Реализация генератора перестановок была взята из предыдущей лабораторной работы.

Для проверки правильности решения ветвей и границ, реализуем генератор перестановок, который перебирает все возможные перестановки городов и вычисляет их стоимость:

1. Реализация класБе permutation в файле Combi.h: класс permutation реализует методы для генерации всех перестановок заданного множества. Включает методы reset, getfirst, getnext, ntx, и count.
2. Использование генератора перестановок: Блздаем объект permutation и генерируем все возможные перестановки маршрутов. Для каждой перестановки вычисляем стоимость маршрута с использованием функции calculateRouteCost. Обновляем наилучшее решение, если находим маршрут с меньшей стоимостью.

Результаты генератора перестановок:

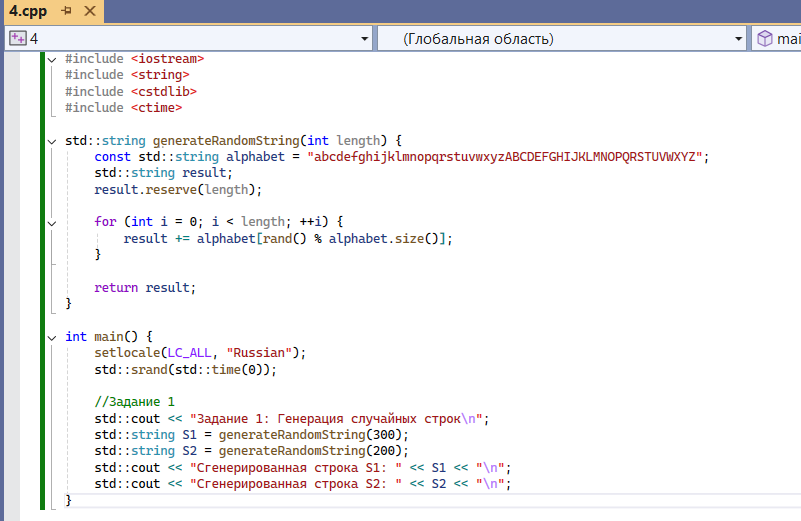


# **Лабораторная работа 4**

***Задание 1.***

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

Для Блздания генератора случайных строк была Блздана функция - **generateRandomString.**



1. Объявление функции для генерации случайной строки:

Функция **generateRandomString(int length)** генерирует случайную строку указанной длины из символов латинского алфавита

**std::string result:** переменная для хранения сгенерированной строки.

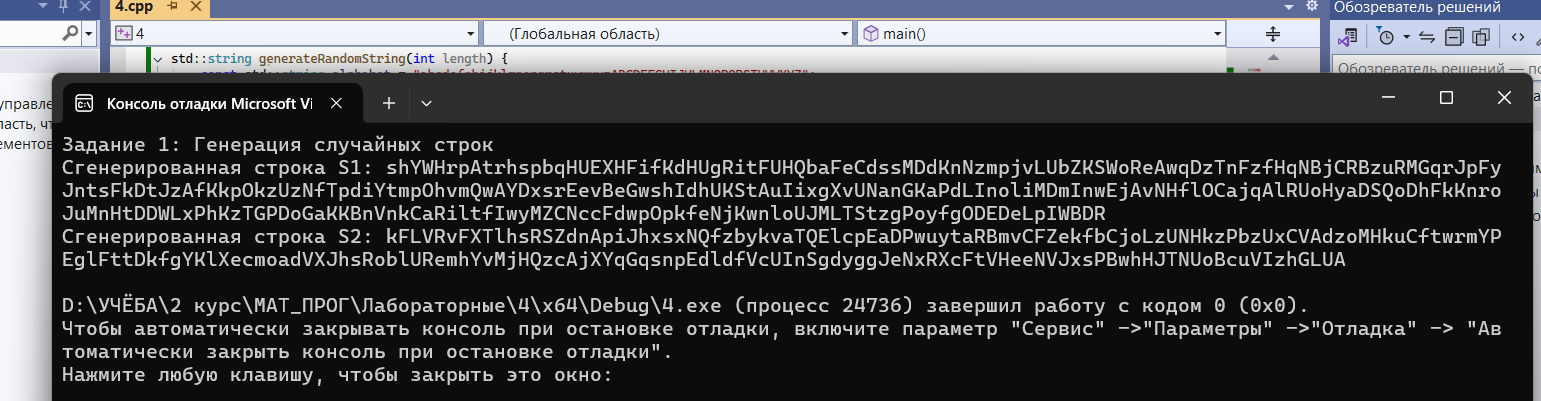
**result.reserve(length):** зарезервировать память для строки заданной длины.

1. Генерация случайных символов:

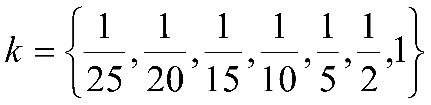
Цикл for проходит по всем символам строки и добавляет случайный символ из строки **alphabet**. **rand() % alphabet.size()** используется для генерации случайного индекБе символа.

1. Основная функция main

**std::srand(std::time(0)):** инициализация генератора случайных чисел текущим временем.



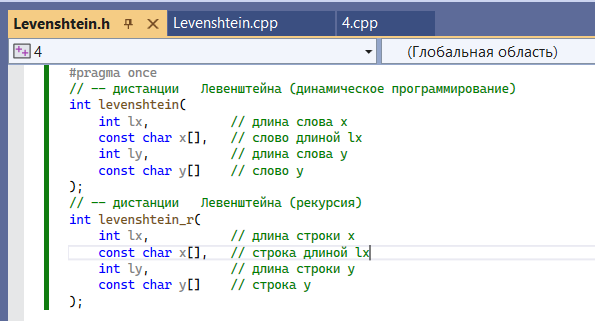
***Задание 2.***

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

***Задание 3.***

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

Для выполнения этого задания воспользуемся предоставленными в методическом поБлбии класБеми.



**Функция levenshtein (динамическое программирование):** это оптимизированный подход, который использует таблицу (обычно двумерный массив) для Блхранения промежуточных результатов. Он предотвращает многократное вычисление одинаковых подзадач, что существенно ускоряет выполнение.

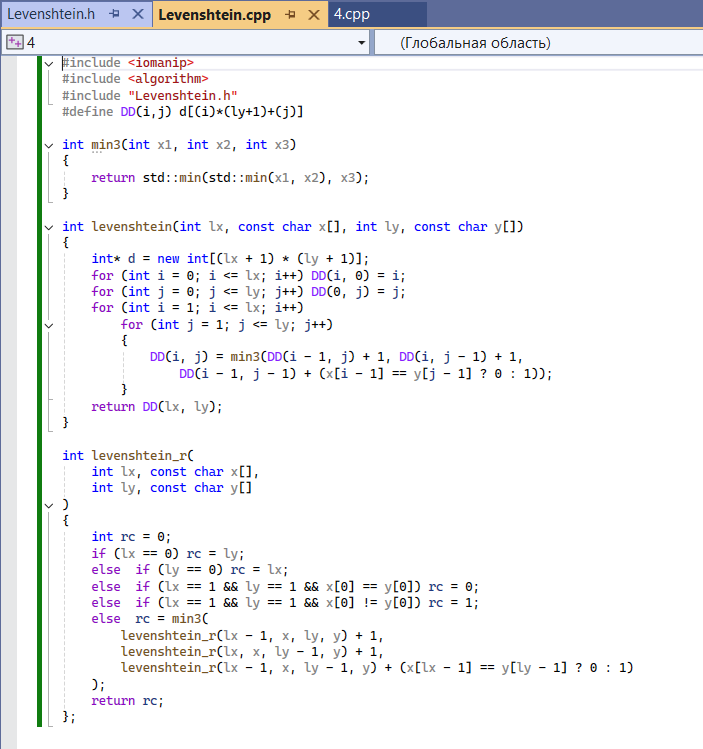
Параметры:

1. lx: длина первой строки x.
2. x[]: массив символов, представляющий первую строку.
3. ly: длина второй строки y.
4. y[]: массив символов, представляющий вторую строку.

**Функция levenshtein\_r (рекурсия):** решение на основе рекурсивного подхода, который повторно вычисляет подзадачи, вызывая Бем себя.

Параметры:

Аналогичны функции levenshtein (lx, x[], ly, y[]).



**Функция levenshtein\_r (рекурсия):** рекурсивно сравниваются последние символы строк и рассчитывается минимальная стоимость операций (вставка, удаление, замена).

В обычных случаях: если одна из строк пуста, расстояние равно длине другой строки. Если обе строки Блстоят из одного символа, возвращается 0 (если символы равны) или 1 (если они разные). Вызывает себя трижды для всех возможных преобразований.

Ключевая оБлбенность: простота реализации, но отсутствие оптимизации (нет кеширования промежуточных результатов).

Производительность: требует значительных ресурБлв (оБлбенно стековой памяти) для обработки большого количества рекурсивных вызовов.

**Функция levenshtein (динамическое программирование):**

Блздается двумерная таблица d размером (lx+1)×(ly+1)(lx + 1) \times (ly + 1), где lxlx и lyly — длины строк x и y. Таблица заполняется минимальными значениями операций (вставка, удаление, замена) для каждой пары подстрок. Итеративно обрабатываются все символы обоих строк с расчетом минимального количества операций для преобразования.

Ключевая оБлбенность: min3 используется для выбора минимального значения из трех операций: удаление символа, вставка символа, замена (или отсутствие изменений, если символы Блвпадают).

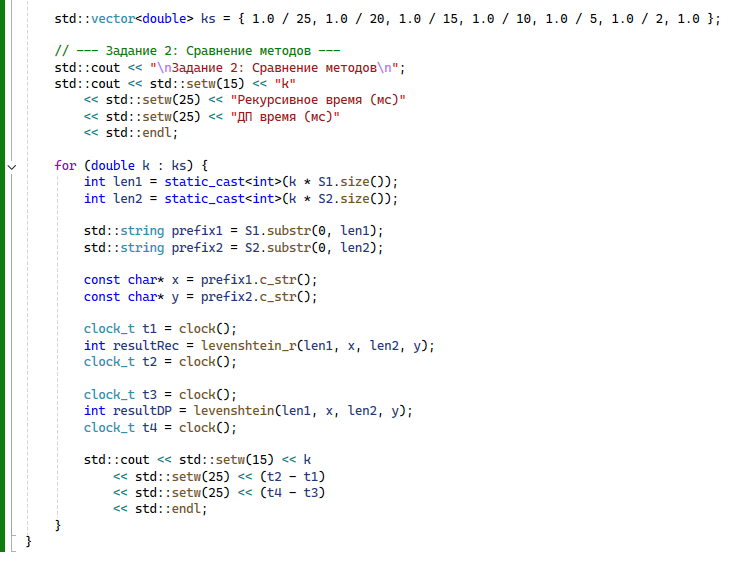
Производительность: временная сложность: O(lx×ly)O(lx \times ly) (эффективно для больших строк).

Пространственная сложность: также O(lx×ly)O(lx \times ly) из-за использования таблицы.

Общие аспекты: min3(int x1, int x2, int x3):

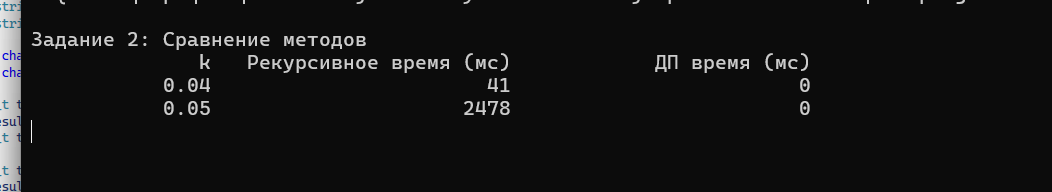
Вспомогательная функция для выбора минимального значения из трех возможных операций позволяет сделать код компактным и понятным.

Оптимизация через динамическое программирование: метод динамического программирования (levenshtein) значительно быстрее для реальных приложений и справляется с длинными строками за разумное время. Метод рекурсии (levenshtein\_r) больше подходит для демонстрации работы алгоритма на учебных примерах.



В коде представлены значения k (коэффициенты длины строк), которые варьируются от 1/25 до 1. Для каждого k:

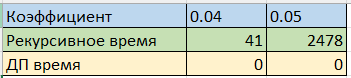
1. длина подстрок пропорциональна k, и как следствие, влияние на время выполнения будет различным:
2. маленькие значения k: время выполнения у обоих методов относительно невелико.
3. большие значения k: рекурсивный метод начинает показывать резкое увеличение времени выполнения, в то время как метод динамического программирования Блхраняет нормальную производительность.



**Вывод:**

Динамическое программирование более эффективно для больших строк. Рекурсивный метод не практичен для длинных строк из-за экспоненциального роста времени выполнения.

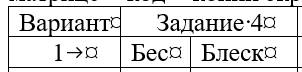
**Преимущества динамического программирования:** подходит для реальных приложений и время выполнения увеличивается линейно в зависимости от длины строк.

****

Из графиков видно, что время выполнения программы значительно выше при рекурсивном методе и мало при динамическом.

***Задание 4.***

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



Расстояние Левенштейна считается как минимальное количество операций (вставка, удаление или замена символов), необходимых для преобразования одной строки в другую.

Рекурсивная формула:

1. Если одна строка пуста, результат — длина второй строки.

2. Если последние символы одинаковы, смотрим подстроки без последних символов.

3. Если последние символы разные, то рассчитываем стоимость для:

1. удаления символа (удаляем последний символ первой строки)
2. вставки символа (добавление последнего символа второй строки к первой строке)
3. замены символа (последний символ первой строки заменяем на последний символ второй)

И берем минимальную стоимость из трёх.

**Обозначим:**

1. **L(S1, S2)** — расстояние Левенштейна между строками S1 и S2.
2. **cut(S)** — строка без последнего символа.
3. **last(S)** — последний символ строки S.

Применим алгоритм.



*= 5.*

*= 4.*



*= 4.*

*= 3.*



*= 3.*

*= 2.*



*= 2.*

*= 1.*



*= 3.*

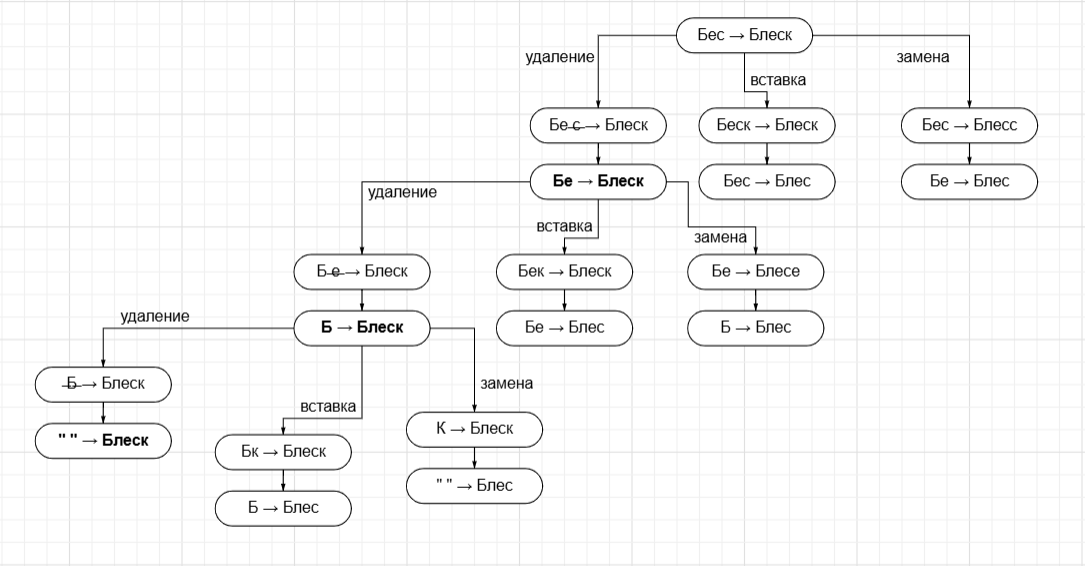
*= 2.*



*= 2.*

*= 1.*





**Шаги выполнения:**

**Шаг 1. L("Бес", "Блеск")**

Последние символы: last("Бес") = с, last("Блеск") = к, разные. Поэтому есть 3 возможные операции:

1. Удаление: L("Бе ̶с̶ ", "Блеск") => (("Бе ", "Блеск") => +1

2. Вставка: L("Беск", "Блеск")=>("Бес", "Блес")=> + 1

3. Замена: L("Бес", "Блесс") =>("Бе", "Блес")=> + 1

Выбираем **удаление.**

**Шаг 2. L("Бе", "Блеск")**

Последние символы: last("Бе") = е, last("Блеск") = к, разные. Возможные операции:

1. Удаление: L("Б ̶е̶", "Блеск")=>("Б ", "Блеск") => + 1

2. Вставка: L("Бек", "Блеск") =>("Бе", "Блес") =>+ 1

3. Замена: L("Бе", "Блесе") =>("Б", "Блес") =>+ 1

Выбираем **удаление.**

**Шаг 3. L("Б", "Блеск")**

Последние символы: last("Б") = б, last("Блеск") = к, разные. Возможные операции:

1. Удаление: L("̶Б̶", "Блеск") =>(" ", "Блеск") =>+ 1

2. Вставка: L("Бк", "Блеск") => ("Б", "Блес")=>+ 1

3. Замена: L("К", "Блеск") =>(" ", "Блес") =>+ 1

Выбираем **удаление.** Первая строка пуста.

**Итоговое расстояние:**

Объединяя все шаги, получаем результат: ( L("Бес", "Блеск") = 3 ).

***Задание 5.***

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



**1. Рекурсивное решение**

Рекурсивный метод уже представлен в коде в lcs. Этот метод использует сравнение последних символов и разветвление на три возможных пути: если символы Блвпадают: длина увеличивается на 1, а строки уменьшаются. Если символы разные: выбирается максимальное значение из двух подзадач — либо удаление из первой строки, либо из второй.

**2. Динамическое программирование**

Метод DP (lcsd) уже опиБес. Основные шаги: Блздание двумерного массива для хранения промежуточных результатов, вычисление максимальной длины LCS через итерации, извлечение Бемой наибольшей подпоследовательности с помощью направления движения по таблице.

В программе для расчёта используется код из методического поБлбия для вычисления рекурсивного и динамического вычисления дины LCS.

**Заполнение и вычисление матриц:**

**1. Левая матрица: Таблица значений длины LCS.**

Эта матрица хранит промежуточные результаты вычисления LCS. Строки и столбцы Блответствуют символам двух последовательностей. Значения в ячейках таблицы указывают длину наибольшей общей подпоследовательности между Блответствующими подстроками.

Как заполняется:

1. **Инициализация**: первая строка (верхняя) и первый столбец (слева) заполняются нулями, так как, если одна из строк пустая, длина LCS равна 0.

2. **Основное правило**: для каждой ячейки dp[i][j] (где i и j — индексы строки и столбца): если символы строк Блвпадают (X[i-1] == Y[j-1]):

– dp[i][j] = dp[i-1][j-1] + 1, добавляем 1 к длине LCS предыдущих подстрок. Если символы не Блвпадают:

– dp[i][j] = max(dp[i-1][j], dp[i][j-1]), берем максимальное значение между удалением символа из одной строки или другой.

**Пример заполнения:**

Для строк X = "ALBDACD" и Y = "CDLDCA":

Столбцы и строки Блответствуют символам (с дополнительным "пустым" начальным символом). Каждый шаг вычисления учитывает Блвпадение символов и строит LCS по мере заполнения.

**2. Правая матрица: Направления движения (Backtracking)**

Эта матрица хранит информацию о том, каким образом был вычислен каждый элемент в левой таблице. Она используется для восстановления Бемой LCS.

Типы стрелок:

– стрелка вверх (TOP): значение взято из ячейки сверху (dp[i-1][j]), что означает исключение символа из последовательности X.

– стрелка влево (LEFT): значение взято из ячейки слева (dp[i][j-1]), что означает исключение символа из последовательности Y.

– диагональная стрелка (LEFTTOP): Значение взято из ячейки по диагонали сверху-слева (dp[i-1][j-1]), что указывает на Блвпадение символов и их включение в LCS.

**Как используется:**

1. Начинаем с последней ячейки (правый нижний угол).

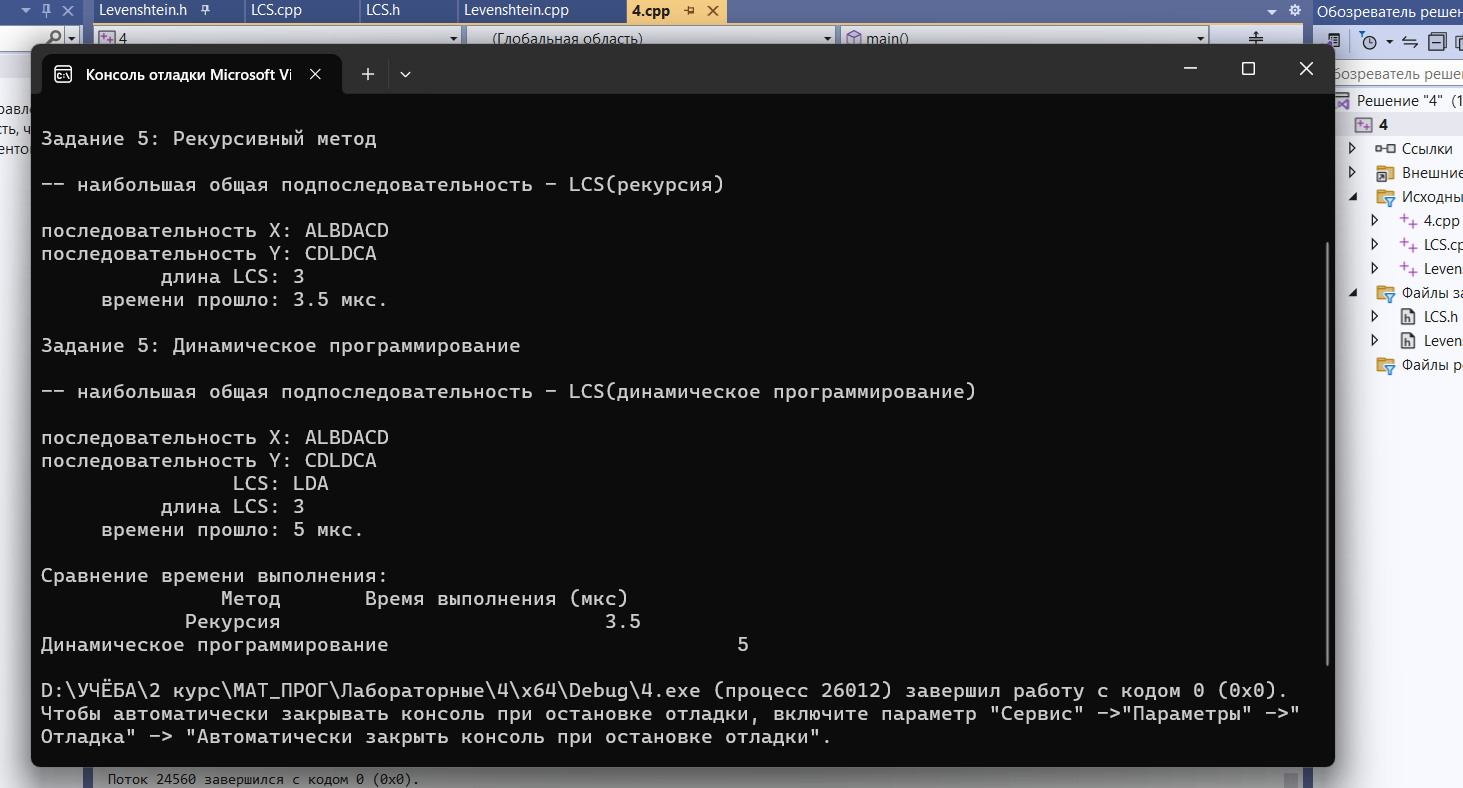
2. Двигаемся по стрелкам:

– если стрелка диагональная, символ добавляется в LCS.

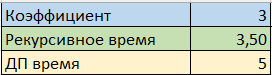
– если стрелка вверх или влево, пропускаем Блответствующий символ.

**Что означают матрицы: левая матрица** – показывает числовые значения, отражающие длину LCS для каждой пары подстрок. **Правая матрица** – определяет путь для восстановления Бемой LCS.

****



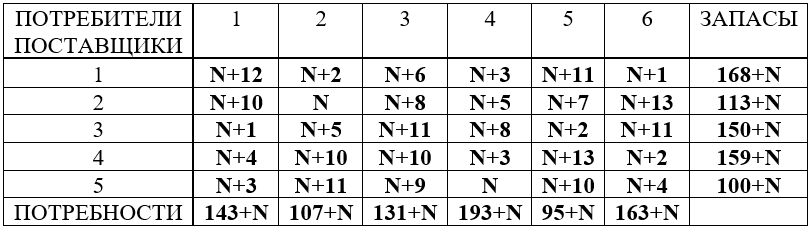
Из выводов решения, можно подвести итоги: затраченное время на динамический алгоритм превышает рекурсивный алгоритм в 1,5 раза. Из этого следует, что использовать динамический алгоритм в подобного вида задачах, не Бемое оптимальное решение.



# **Лабораторная работа 5**

**Вариант 1**

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



**Ход решения:**

* Проверить, открытая задача или закрытая;



Суммарный объём запасов у поставщиков:

**169 + 114 + 151 + 160 + 101 = 695**

Суммарный объём потребностей у потребителей:

**144 + 108 + 132 + 194 + 96 + 164 = 838**.

Так как 695 не равно 838, задача носит открытый характер. Поскольку сумма предложения меньше суммы спроса, возникает недостаток товаров. Чтобы сбалансировать задачу, добавляем фиктивного поставщика с объёмом поставок:

838 - 695 = 143.

Стоимость перевозки от фиктивного поставщика до каждого потребителя устанавливается равной нулю. После добавления фиктивного поставщика задача становится закрытой и готовой для решения.

* Составить опорный план;

Построение начального базисного решения: метод северо-западного угла, метод наименьшей стоимости (минимального элемента), метод Фогеля

**Метод наименьшей стоимости заключается в том, что мы сначала заполняем ячейки с наименьшей стоимостью перевозки.**

Этот метод используется для минимизации стоимости транспортировки:

Пусть m=5, поставщиков продукции,

n=6, потребителей продукции.

Запасы A=(169, 114,151,160,101)

Потребности B=(144, 108, 132, 194, 96, 164)

Затраты на перевозку продукции C=()=

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4 | 12 | 2 | 169 |
| 2 | 11 | 1 | 9 | 6 | 8 | 14 | 114 |
| 3 | 2 | 6 | 12 | 9 | 3 | 12 | 151 |
| 4 | 5 | 11 | 11 | 4 | 14 | 3 | 160 |
| 5 | 4 | 12 | 10 | 1 | 11 | 5 | 101 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 144 | 108 | 132 | 194 | 96 | 164 |  |

Выбор ячейки с наименьшим значением и =1

=min (114, 108)=108

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4 | 12 | 2 | 169 |
| 2 | 11 | 1| 108 | 9 | 6 | 8 | 14 | 6 |
| 3 | 2 | 6 | 12 | 9 | 3 | 12 | 151 |
| 4 | 5 | 11 | 11 | 4 | 14 | 3 | 160 |
| 5 | 4 | 12 | 10 | 1 | 11 | 5 | 101 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 144 | 0 | 132 | 194 | 96 | 164 |  |

=min (194, 101)=101

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4 | 12 | 2 | 169 |
| 2 | 11 | 1| 108 | 9 | 6 | 8 | 14 | 6 |
| 3 | 2 | 6 | 12 | 9 | 3 | 12 | 151 |
| 4 | 5 | 11 | 11 | 4 | 14 | 3 | 160 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 144 | 0 | 132 | 93 | 96 | 164 |  |

=min (164, 169)=164

Выбор ячейки с наименьшим значением =2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4 | 12 | 2|164 | 5 |
| 2 | 11 | 1| 108 | 9 | 6 | 8 | 14 | 6 |
| 3 | 2 | 6 | 12 | 9 | 3 | 12 | 151 |
| 4 | 5 | 11 | 11 | 4 | 14 | 3 | 160 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 144 | 0 | 132 | 93 | 96 | 0 |  |

=min (144, 151)=144

Выбор ячейки с наименьшим значением =2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4 | 12 | 2|164 | 5 |
| 2 | 11 | 1| 108 | 9 | 6 | 8 | 14 | 5 |
| 3 | 2|144 | 6 | 12 | 9 | 3 | 12 | 7 |
| 4 | 5 | 11 | 11 | 4 | 14 | 3 | 160 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 0 | 0 | 132 | 93 | 96 | 0 |  |

=min (96, 7)=7

Выбор ячейки с наименьшим значением =3

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4 | 12 | 2|164 | 5 |
| 2 | 11 | 1| 108 | 9 | 6 | 8 | 14 | 6 |
| 3 | 2|144 | 6 | 12 | 9 | 3|7 | 12 | 0 |
| 4 | 5 | 11 | 11 | 4 | 14 | 3 | 160 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 0 | 0 | 132 | 93 | 89 | 0 |  |

=min (93, 5)=5

Выбор ячейки с наименьшим значением =5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4|5 | 12 | 2|164 | 0 |
| 2 | 11 | 1| 108 | 9 | 6 | 8 | 14 | 6 |
| 3 | 2|144 | 6 | 12 | 9 | 3|7 | 12 | 0 |
| 4 | 5 | 11 | 11 | 4 | 14 | 3 | 160 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 0 | 0 | 132 | 88 | 89 | 0 |  |

=min (89, 6)=6

Выбор ячейки с наименьшим значением =5

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4|5 | 12 | 2|164 | 0 |
| 2 | 11 | 1| 108 | 9 | 6 | 8|6 | 14 | 0 |
| 3 | 2|144 | 6 | 12 | 9 | 3|7 | 12 | 0 |
| 4 | 5 | 11 | 11 | 4 | 14 | 3 | 160 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 0 | 0 | 132 | 88 | 83 | 0 |  |

=min (160, 88)=88

Выбор ячейки с наименьшим значением =4

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4|5 | 12 | 2|164 | 0 |
| 2 | 11 | 1| 108 | 9 | 6 | 8|6 | 14 | 0 |
| 3 | 2|144 | 6 | 12 | 9 | 3|7 | 12 | 0 |
| 4 | 5 | 11 | 11 | 4|88 | 14 | 3 | 72 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 0 | 0 | 132 | 0 | 83 | 0 |  |

=min (72, 132)=72

Выбор ячейки с наименьшим значением =11

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4|5 | 12 | 2|164 | 0 |
| 2 | 11 | 1| 108 | 9 | 6 | 8|6 | 14 | 0 |
| 3 | 2|144 | 6 | 12 | 9 | 3|7 | 12 | 0 |
| 4 | 5 | 11 | 11|72 | 4|88 | 14 | 3 | 0 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 0 | 0 | 60 | 0 | 83 | 0 |  |

=min (60, 143)=60

Выбор ячейки с наименьшим значением =60

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4|5 | 12 | 2|164 | 0 |
| 2 | 11 | 1| 108 | 9 | 6 | 8|6 | 14 | 0 |
| 3 | 2|144 | 6 | 12 | 9 | 3|7 | 12 | 0 |
| 4 | 5 | 11 | 11|72 | 4|88 | 14 | 3 | 0 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 0 |
| 6 | 0 | 0 | 0|60 | 0 | 0 | 0 | 83 |
| ПОТРЕБНОСТИ | 0 | 0 | 0 | 0 | 83 | 0 |  |

=min (83, 83)=83

Выбор ячейки с наименьшим значением =83

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4|5 | 12 | 2|164 | 0 |
| 2 | 11 | 1| 108 | 9 | 6 | 8|6 | 14 | 0 |
| 3 | 2|144 | 6 | 12 | 9 | 3|7 | 12 | 0 |
| 4 | 5 | 11 | 11|72 | 4|88 | 14 | 3 | 0 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 0 |
| 6 | 0 | 0 | 0|60 | 0 | 0|83 | 0 | 0 |
| ПОТРЕБНОСТИ | 0 | 0 | 0 | 0 | 0 | 0 |  |

Должно быть базовых m+n-1=5+6-1= 10 переменных. Выбираем ячейку =11

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4|5 | 12 | 2|164 | 169 |
| 2 | 11 | 1| 108 | 9 | 6 | 8|6 | 14 | 114 |
| 3 | 2|144 | 6 | 12 | 9 | 3|7 | 12 | 151 |
| 4 | 5 | 11 | 11|72 | 4|88 | 14 | 3 | 160 |
| 5 | 4 | 12 | 10 | 1|101 | 11 | 5 | 101 |
| 6 | 0 | 0 | 0|60 | 0 | 0|83 | 0 | 143 |
| ПОТРЕБНОСТИ | 144 | 108 | 132 | 194 | 96 | 164 |  |

Первое допустимое решение: =5, =164, =108, =6, =144, =7, =72, =88, =101, =60, =83.

Значение функции цели:

Z=4∙5 + 2∙164 + 1∙108 + 8∙6 + 2∙144 + 3∙7 + 11∙72 + 4∙88 + 1∙101 + 0∙60 + 0∙83 = 2058.

* Применить метод потенциалов;

В методе потенциалов каждой строке i и каждому столбцу j транспортной таблицы ставятся в соответствие числа(потенциалы) (поставщики) и (потребители). Для каждой базисной переменной потенциалы и удовлетворяют уравнению. + =

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | 1 | 2 | 3 | 4 | 5 | 6 | ЗАПАСЫ |
| 1 | 13 | 3 | 7 | 4|169 | 12 | 2|164 | 169 |
| 2 | 11 | 1| 108 | 9|6 | 6 | 8 | 14 | 114 |
| 3 | 2|144 | 6 | 12 | 9 | 3|7 | 12 | 151 |
| 4 | 5 | 11 | 11|25 | 4|46 | 14|68 | 3|21 | 160 |
| 5 | 4 | 12 | 10|101 | 1 | 11 | 5 | 101 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 143 |
| ПОТРЕБНОСТИ | 144 | 108 | 132 | 194 | 96 | 164 |  |

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

u1 + v4 = 4; 0 + v4 = 4; v4 = 4

u4 + v4 = 4; 4 + u4 = 4; u4 = 0

u4 + v3 = 11; 0 + v3 = 11; v3 = 11

u6 + v3 = 0; 11 + u6 = 0; u6 = -11

u6 + v5 = 0; -11 + v5 = 0; v5 = 11

u2 + v5 = 8; 11 + u2 = 8; u2 = -3

u2 + v2 = 1; -3 + v2 = 1; v2 = 4

u3 + v5 = 3; 11 + u3 = 3; u3 = -8

u3 + v1 = 2; -8 + v1 = 2; v1 = 10

u5 + v4 = 1; 4 + u5 = 1; u5 = -3

u1 + v6 = 2; 0 + v6 = 2; v6 = 2

Уровней 10, переменных 10.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | V1=10 | V2=4 | V3=11 | v4=4 | v5=11 | v6=2 | ЗАПАСЫ |
| u1=0 | 13 | 3 | 7 | 4[5] | 12 | 2[164] | 169 |
| u2=-3 | 11 | 1[108] | 9 | 6 | 8[6] | 14 | 114 |
| u3=-8 | 2[144] | 6 | 12 | 9 | 3[7] | 12 | 151 |
| u4=0 | 5 | 11 | 11[72] | 4[88] | 14 | 3 | 160 |
| u5=-3 | 4 | 12 | 10 | 1[101] | 11 | 5 | 101 |
| u6=-11 | 0 | 0 | 0[60] | 0 | 0[83] | 0 | 143 |
| ПОТРЕБНОСТИ | 144 | 108 | 132 | 194 | 96 | 164 |  |

ui + vj > cij, значит решение можно улучшить. **Δij=ui+vj−cij**. Если **∆ij > 0**, то решение можно улучшить, переместив часть груза в эту клетку

(1;2): 0 + 4 > 3; ∆12 = 0 + 4 - 3 = 1 > 0

(1;3): 0 + 11 > 7; ∆13 = 0 + 11 - 7 = 4 > 0

(4;1): 0 + 10 > 5; ∆41 = 0 + 10 - 5 = 5 > 0

(5;1): -3 + 10 > 4; ∆51 = -3 + 10 - 4 = 3 > 0 max(1,4,5,3) = 5

Мы нашли ячейки с положительными оценками: **максимальная оценка: Δ(4,1) = 5.** Значит, перемещение в эту ячейку **снизит стоимость**. Выбираем максимальную оценку свободной клетки (4;1): 5. Для этого в перспективную клетку (4;1) поставим знак «+», а в остальных вершинах многоугольника чередующиеся знаки «-», «+», «-».

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | V1=10 | V2=4 | V3=11 | v4=4 | v5=11 | v6=2 | ЗАПАСЫ |
| u1=0 | 13 | 3 | 7 | 4[5] | 12 | 2[164] | **169** |
| u2=-3 | 11 | 1[108] | 9 | 6 | 8[6] | 14 | **114** |
| u3=-8 | 2[144][-] | 6 | 12 | 9 | 3[7][+] | 12 | **151** |
| u4=0 | 5[+] | 11 | 11[72][-] | 4[88] | 14 | 3 | **160** |
| u5=-3 | 4 | 12 | 10 | 1[101] | 11 | 5 | **101** |
| u6=-11 | 0 | 0 | 0[60][+] | 0 | 0[83][-] | 0 | **143** |
| ПОТРЕБНОСТИ | **144** | **108** | **132** | **194** | **96** | **164** |  |

Цикл приведенный 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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | V1=10 | V2=4 | V3=11 | v4=4 | v5=11 | v6=2 | ЗАПАСЫ |
| u1=0 | 13 | 3 | 7 | 4[5] | 12 | 2[164] | **169** |
| u2=-3 | 11 | 1[108] | 9 | 6 | 8[6] | 14 | **114** |
| u3=-8 | 2[72] | 6 | 12 | 9 | 3[79] | 12 | **151** |
| u4=0 | 5[72] | 11 | 11 | 4[88] | 14 | 3 | **160** |
| u5=-3 | 4 | 12 | 10 | 1[101] | 11 | 5 | **101** |
| u6=-11 | 0 | 0 | 0[132] | 0 | 0[11] | 0 | **143** |
| ПОТРЕБНОСТИ | **144** | **108** | **132** | **194** | **96** | **164** |  |

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

u1 + v4 = 4; 0 + v4 = 4; v4 = 4

u4 + v4 = 4; 4 + u4 = 4; u4 = 0

u4 + v1 = 5; 0 + v1 = 5; v1 = 5

u3 + v1 = 2; 5 + u3 = 2; u3 = 3

u3 + v5 = 3; -3 + v5 = 3; v5 = 6

u2 + v5 = 8; 6 + u2 = 8; u2 = 2

u2 + v2 = 1; 2 + v2 = 1; v2 = 1

u6 + v5 = 0; 6 + u6 = 0; u6 = 6

u6 + v3 = 0; -6 + v3 = 0; v3 = 6

u5 + v4 = 1; 4 + u5 = 1; u5 = 3

u1 + v6 = 2; 0 + v6 = 2; v6 = 2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| ПОТРЕБИТЕЛИ  ПОСТАВЩИКИ | V1=10 | V2=4 | V3=11 | v4=4 | v5=11 | v6=2 | ЗАПАСЫ |
| u1=0 | 13 | 3 | 7 | 4[5] | 12 | 2[164] | **169** |
| u2=-3 | 11 | 1[108] | 9 | 6 | 8[6] | 14 | **114** |
| u3=-8 | 2[72] | 6 | 12 | 9 | 3[79] | 12 | **151** |
| u4=0 | 5[72] | 11 | 11 | 4[88] | 14 | 3 | **160** |
| u5=-3 | 4 | 12 | 10 | 1[101] | 11 | 5 | **101** |
| u6=-11 | 0 | 0 | 0[132] | 0 | 0[11] | 0 | **143** |
| ПОТРЕБНОСТИ | **144** | **108** | **132** | **194** | **96** | **164** |  |

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

Минимальные затраты составят: F(x) = 4∙5 + 2∙164 + 1∙108 + 8∙6 + 2∙72 + 3∙79 + 5∙72 + 4∙88 + 1∙101 + 0∙132 + 0∙11 = 1698

**Анализ оптимального плана.**

Из 1-го склада необходимо груз направить в 4-й магазин (5 ед.), в 6-й магазин (164 ед.)

Из 2-го склада необходимо груз направить в 2-й магазин (108 ед.), в 5-й магазин (6 ед.)

Из 3-го склада необходимо груз направить в 1-й магазин (72 ед.), в 5-й магазин (79 ед.)

Из 4-го склада необходимо груз направить в 1-й магазин (72 ед.), в 4-й магазин (88 ед.)

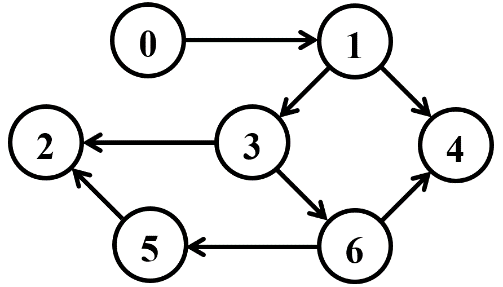
Из 5-го склада необходимо весь груз направить в 4-й магазин.

Потребность 3-го магазина остается удовлетворенной.

Оптимальный план является вырожденным, так как базисная переменная x63=0. Потребность 5-го магазина остается удовлетворенной. Оптимальный план является вырожденным, так как базисная переменная x65=0.

# **Лабораторная работа 6 ВАРИАНТ 1**

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



**Матрица смежности**

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

**Матрица инцедентности**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Вершины \ ребра | Е1 | Е2 | Е3 | е4 | Е5 | Е6 | Е7 | Е8 |
| 0 | -1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | -1 | -1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 3 | 0 | 1 | 0 | -1 | -1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | -1 |
| 6 | 0 | 0 | 0 | 0 | 1 | -1 | -1 | 0 |

**Список смежных вершин**

Список смежных вершин показывает, какие вершины соединены дугами для каждой вершины графа.

Вершина 0: [1]

Вершина 1: [3,4]

Вершина 2: -

Вершина 3: [2,6]

Вершина 4: -

Вершина 5: [2]

Вершина 6: [4,5]

***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**. **Поиск в ширину (BFS)**

Алгоритм обхода в ширину использует очередь для обхода вершин графа.

Старт с вершины 0:

Очередь: 0.

Посещаем вершину 1. Очередь становится: 1.

Обработка вершины 1:

Очередь: 1.

Из вершины 1 переходим в вершины 3 и 4. Очередь: 3, 4.

Обработка вершины 3:

Очередь: 3, 4.

Из вершины 3 переходим в вершины 2 и 6. Очередь: 4, 2, 6.

Обработка вершины 4:

Очередь: 4, 2, 6.

Вершина 4 не имеет исходящих рёбер.

Обработка вершины 2:

Очередь: 2, 6.

Вершина 2 не имеет исходящих рёбер.

Обработка вершины 6:

Очередь: 6.

Из вершины 6 переходим в вершины 4 и 5. Очередь: 5.

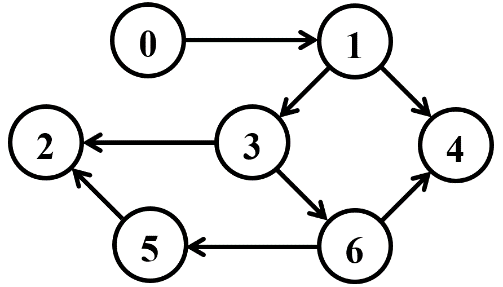
Обработка вершины 5:

Очередь: 5.

Из вершины 5 переходим в вершину 2. Очередь становится пустой.

Итог: BFS посещает вершины в следующем порядке:

0 → 1 → 3 → 4 → 2 → 6 → 5.



1. **Поиск в глубину (DFS)**

Алгоритм обхода в глубину использует стек для последовательного спуска по графу.

Шаги для графа (начинаем с вершины 0):

Старт с вершины 0:

Переходим в вершину 1.

Из вершины 1:

Углубляемся в вершину 3.

Из вершины 3:

Переходим в вершину 2. Так как у вершины 2 нет выходов, возвращаемся к вершине 3.

Из вершины 3 переходим в вершину 6.

Из вершины 6:

Углубляемся в вершину 4. Так как у вершины 4 нет выходов, возвращаемся к вершине 6.

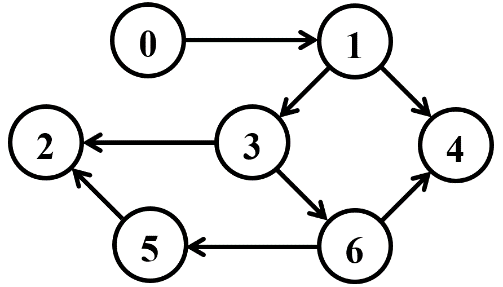
Затем переходим к вершине 5. Из вершины 5 переходим снова в вершину 2.

Завершаем обход, возвращаясь к вершине 1, а затем к вершине 0.

Итог: DFS обходит вершины в порядке: 2 → 4 → 5 → 6 → 3 → 1 → 0.

**3.Топологическая сортировка**

Топологическая сортировка используется для ориентированных ацикличных графов (DAG). Мы обходим граф в глубину и записываем вершины в обратном порядке завершения.



**Используем обход в глубину (DFS):**

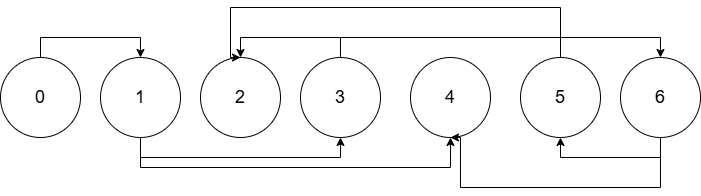
* Начинаем с вершины **0**, затем идём к **1**.
* Из **1** переходим к **3**, а потом к **2**.
* После **2** возвращаемся к **3** и продолжаем к **6**, потом к **5**.
* Вершина **4** тоже учитывается, так как она отдельная.

**Записываем вершины в обратном порядке завершения:**

* Сначала завершается **5**, потом **6**, затем **3**, **2**, **1**, **0**.
* Поскольку **4** тоже есть в графе, её добавляем перед **5**.

**Инвертируем порядок завершения:**

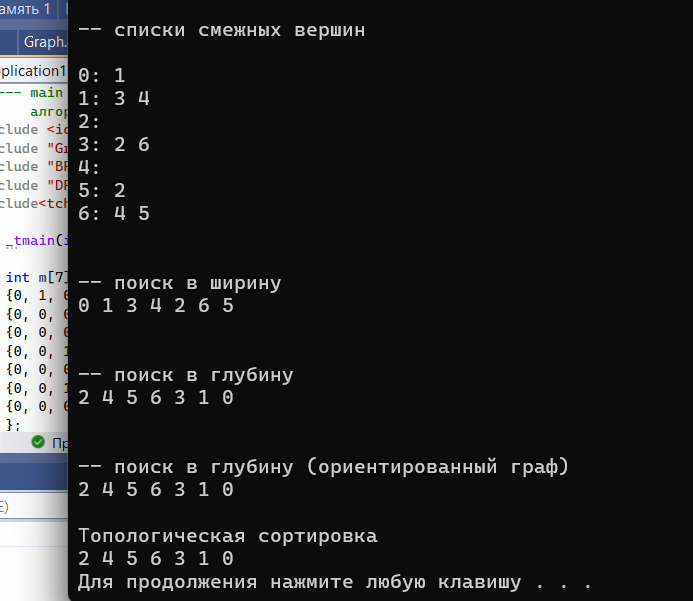
* Получаем итоговую последовательность: **2 → 4 → 5 → 6 → 3 → 1 → 0**.



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



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

Веса ребер принять:

W:

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

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

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

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

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

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

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

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

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

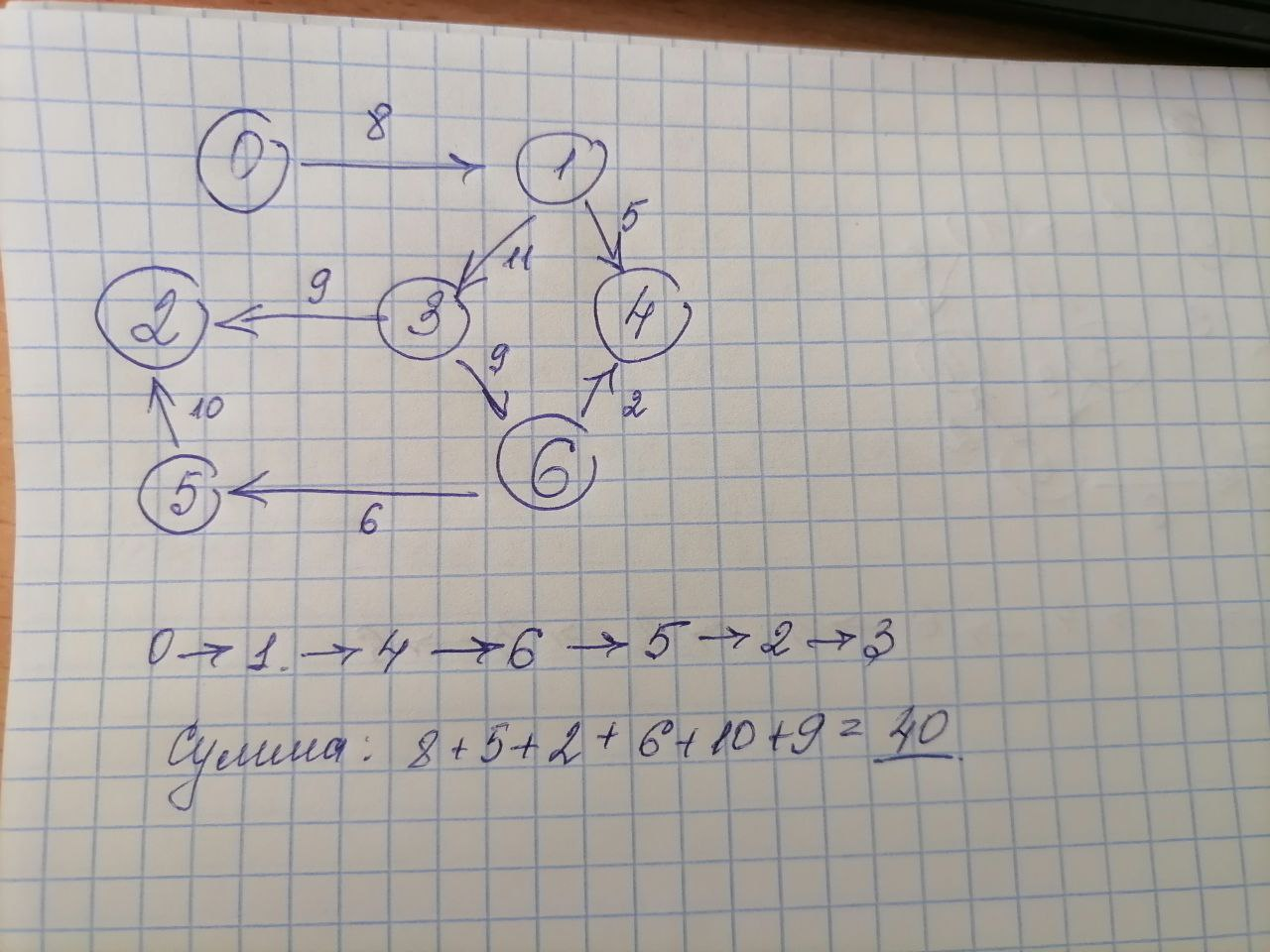
W(e5,6)=2; W(e6,5)=6;

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

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

**Шаги алгоритма Прима**

1. **Выбираем начальную вершину** — пусть это будет **0**.
2. **Добавляем минимальное ребро, соединяющее текущие вершины с новыми вершинами**:
   * Из **0** возможные рёбра: *(0 → 1, вес 8)* → выбираем **1**.
3. **Повторяем процесс, выбирая минимальное ребро**:
   * Из уже добавленных вершин {0, 1} минимальное ребро: *(1 → 4, вес 5)* → добавляем **4**.
   * Из {0, 1, 4} минимальное ребро: *(4 → 6, вес 2)* → добавляем **6**.
   * Из {0, 1, 4, 6} минимальное ребро: *(6 → 5, вес 6)* → добавляем **5**.
   * Из {0, 1, 4, 6, 5} минимальное ребро: *(5 → 2, вес 10)* → добавляем **2**.
   * Из {0, 1, 4, 6, 5, 2} минимальное ребро: *(2 → 3, вес 9)* → добавляем **3**.

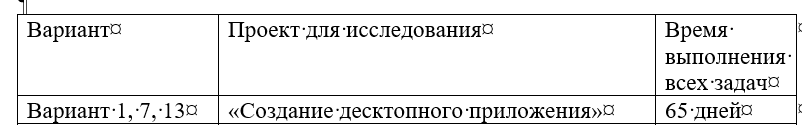


***Задание 7.*** По графу, соответствующему варианту составить минимальное остовное дерево по алгоритму Крускала. Шаги построения отразить в отчете.

**Шаги алгоритма Крускала**

1. **Сортируем рёбра по весу (от меньшего к большему):**
   * **4 → 6 (2)**
   * **1 → 4 (5)**
   * **6 → 5 (6)**
   * **0 → 1 (8)**
   * **2 → 3 (9)**
   * **3 → 4 (9)**
   * **5 → 2 (10)**
   * **1 → 3 (11)**
2. **Добавляем рёбра, если они не образуют цикл:**
   * **4 → 6 (2)** → добавляем, соединяет вершины.
   * **1 → 4 (5)** → добавляем.
   * **6 → 5 (6)** → добавляем.
   * **0 → 1 (8)** → добавляем.
   * **2 → 3 (9)** → добавляем.
   * **5 → 2 (10)** → добавляем.
3. **Итоговое минимальное остовное дерево:**
   * **Рёбра: {4 → 6, 1 → 4, 6 → 5, 0 → 1, 2 → 3, 5 → 2}**
   * **Общая сумма весов:** 2 + 5 + 6 + 8 + 9 + 10 = **40**.

# **Лабораторная работа 7**



**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.

***Тема «Создание десктопного приложения»***

|  |  |  |  |
| --- | --- | --- | --- |
| Код  операции | Наименование операции | Предшествующие операции | t |
| I. АНАЛИЗ | | | |
| Z1 | Сбор и анализ требований от пользователей |  | 5 |
| Z2 | Проектирование интерфейса приложения (UI/UX) | Z1 | 7 |
| Z3 | Создание технического задания и общей архитектуры приложения | Z1 | 5 |
| II. ПРОЕКТИРОВАНИЕ | | | |
| Z4 | Проектирование базы данных | Z2 | 10 |
| Z5 | Проектирование классов | Z3, Z4 | 15 |
| Z6 | Проектирование пользовательского интерфейса | Z2, Z5 | 10 |
| III. КОДИРОВАНИЕ | | | |
| Z7 | Кодирование интерфейсов пользователей | Z6 | 15 |
| Z8 | Кодирование процедур работы с базой данных | Z4, Z5 | 20 |
| Z9 | Кодирование классов | Z5 | 25 |
| IV. ТЕСТИРОВАНИЕ | | | |
| Z10 | Функциональное тестирование | Z7, Z8, Z9 | 15 |
| Z11 | Структурное тестирование | Z7, Z8, Z9 | 20 |
| V. ВНЕДРЕНИЕ | | | |
| Z12 | Разработка документации | Z10 | 5 |
| Z13 | Обучение пользователей | Z12 | 10 |
| Z14 | Испытание | Z11, Z13 | 15 |
| Z15 | Завершение работ | Z14 | 5 |

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

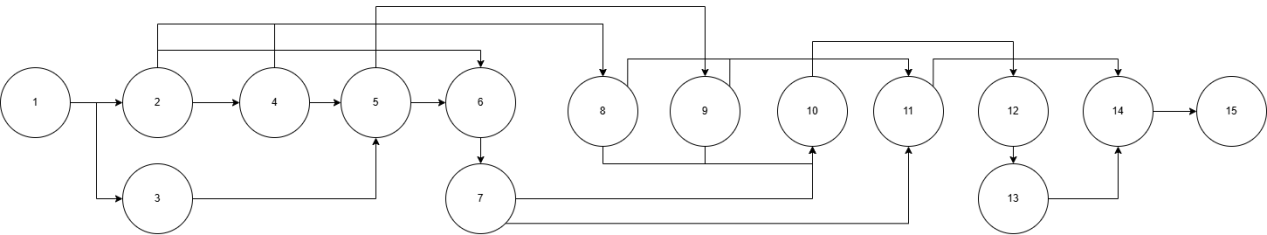
Согласно составленному перечню задач и распределённому времени составьте сетевой график вашего проекта. Помните о правилах составления графика и вводите фиктивные операции и операции ожидания если это необходимо.

При построении сетевых графиков соблюдается ряд правил:

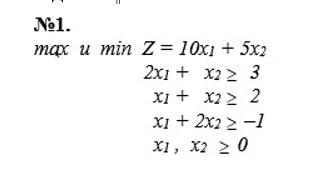
1. в сети не должно быть событий (кроме исходного), в которые не входит ни одна дуга;
2. не должно быть событий (кроме завершающего), из которых не выходит ни одной дуги;
3. сеть не должна содержать замкнутых контуров (циклов);
4. ***любая пара событий сетевого графика может быть соединена не более чем 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.

Z1 → Z2 → Z3 → Z4 → Z5 → Z6 → Z9 → Z10 → Z14 → Z15



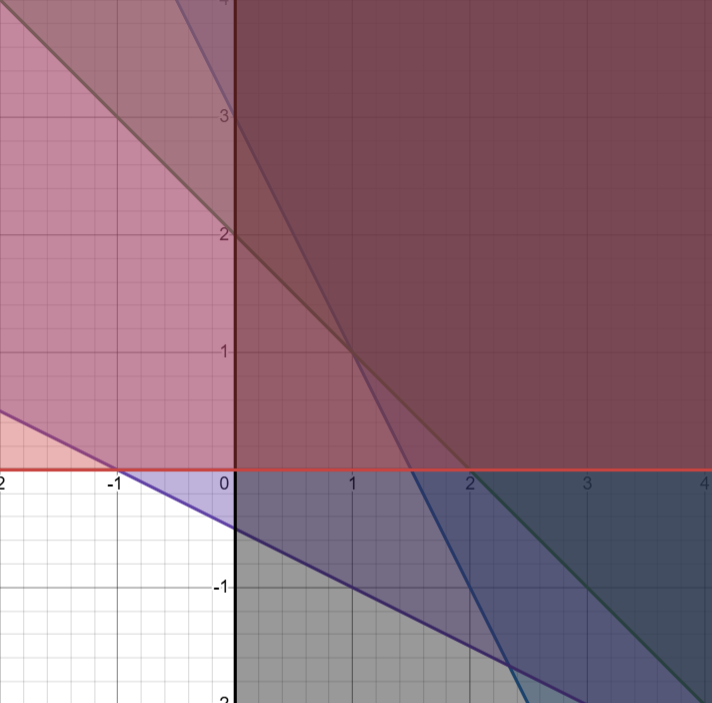
# **Лабораторная работа 8**

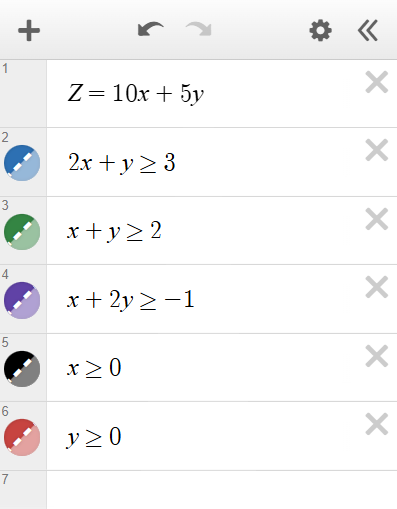


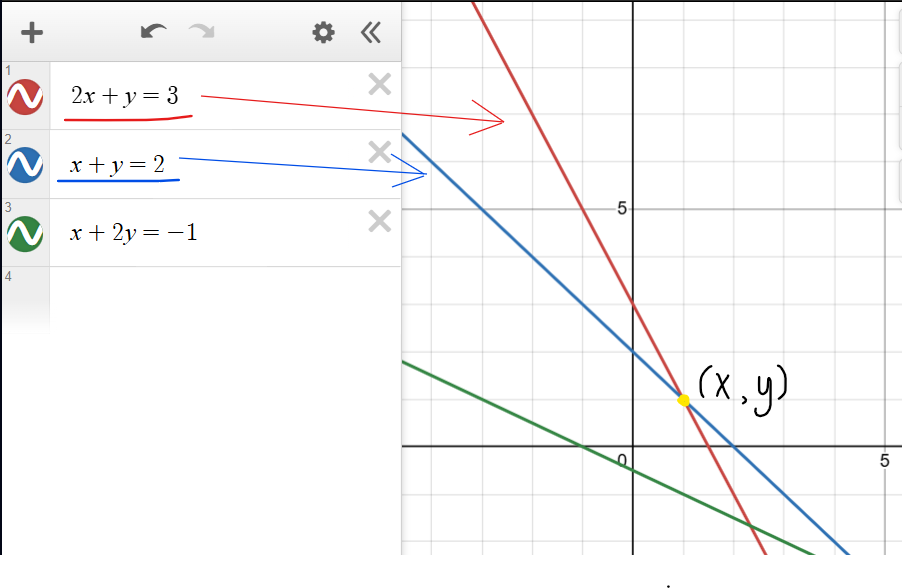
**Методика решения:**

1. Строим область допустимых решений, т.е. решаем графически систему неравенств. Для этого строим каждую прямую и определяем полуплоскости, заданные неравенствами (полуплоскости обозначаем штрихом).
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, если это возможно.

Ход решения и график отобразить в отчете.





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

Решаем систему двух уравнений с двумя неизвестными, чтобы найти точку пересечения прямых:

Зная пару (x, y), можно найти минимум Z:

Zmin = 10x+5y = 15