

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

К курсовому проектированию

По курсу «Л и ОА в ИЗ»

На тему: «Реализация алгоритма Прима»

Выполнил: ст. гр. 22ВВ4  
Жуков Илья

Приняли: Юрова О.В.  
Акифьев И.В.

## Содержание

Реферат .....	3
Введение .....	4
1. Постановка задачи .....	5
2. Теоретическая часть задания .....	6
3. Описание алгоритма программы .....	7
4. Описание программы .....	8
5. Тестирование .....	13
6. Ручной расчет задачи .....	25
Заключение .....	27
Список литературы .....	28
Листинг программы: .....	29

## **Реферат**

Отчет 28 стр, рисунков 24

### **ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ПРИМА, ОСТОВНОЕ ДЕРЕВО**

Цель исследования – разработка программы, способная рассчитывать минимальное остовное дерево, используя алгоритм Прима.

В работе рассмотрены различные способы реализации матрицы смежности для неориентированного графа. Случайная генерация, ручная и при помощи файла. Результат программы можно сохранить в файл и просмотреть в дальнейшем.

## **Введение**

Реализация алгоритма Прима является одним из методов решения задачи нахождения минимального остовного дерева во взвешенном неориентированном связном графе. Алгоритм Прима помогает найти такое остовное дерево, которое содержит все вершины и имеет минимальную сумму весов ребер.

Начиная с некоторой стартовой вершины, алгоритм Прима постепенно строит остовное дерево, добавляя в него ребра с минимальными весами и присоединяя новые вершины, которые еще не включены в дерево. Алгоритм продолжает выполняться до тех пор, пока все вершины не будут включены в остовное дерево.

Реализация алгоритма Прима является важным инструментом для решения задач, связанных с сетями и оптимизацией, таких как строительство дорог, проектирование сетей связи или распределение ресурсов. Она позволяет найти оптимальное решение на основе весов и связей между вершинами в графе.

В качестве среды разработки мною была выбрана среда Microsoft Visual Studio 2022, язык программирования – C#

Целью данной курсовой работы является разработка программы на языке C#, который является широко используемым. Именно с его помощью осуществляется поиск минимального остовного дерева.

## **Постановка задачи**

Требуется разработать программу, которая будет рассчитывать минимальное остовное дерево, используя алгоритм Прима.

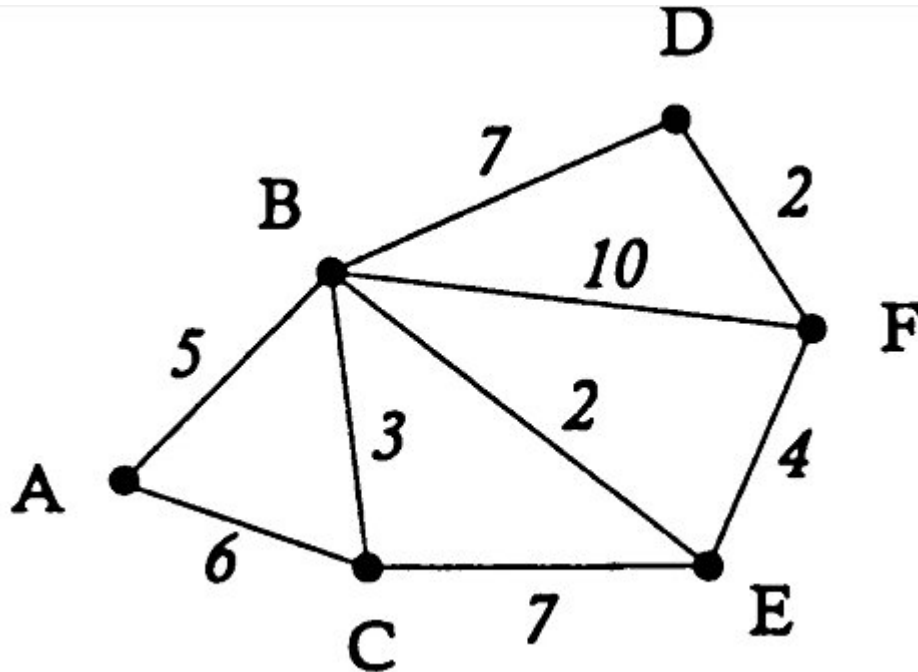
Исходный граф в программе должен задаваться матрицей смежности, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы смежности. После обработки этих данных на экран должна выводиться матрица смежности для неориентированного графа. Необходимо предусмотреть различные исходы поиска, чтобы программа не выдавала ошибок и работало правильно.

Устройство ввода – клавиатура и мышь.

Уровень сложности проекта «Базовый». Номер задания 18.

## Теоретическая часть задания

Граф  $G$  задается множеством вершин  $A, B, C, D, \dots$  и множеством ребер, соединяющих между собой определенные вершины. Ребра из множества  $A$  неориентированы, это означает, что ребра не имеют заданного направления, присвоенного им.



Для каждого ребра задаётся его вес. При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где присутствие пути из одной вершины в другую обозначается числом 1 или более, иначе нулем.

Существует множество алгоритмов на графах, в основе которых лежит систематический перебор вершин графа, такой, что каждая вершина графа просматривается только один раз, и переход от одной вершины к другой осуществляется по ребрам графа. Остановится на алгоритме Прима.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшим весом. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшего веса. Выбираемое на каждом шаге ребро присоединяется к дереву. 3 вершины, соединенные между собой ребрами, присоединяться к дереву не могут. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимального размера.

## Описание алгоритма программы

Переменная "matrix" представляет собой двумерный массив, хранящий веса ребер между вершинами графа. Веса ребер могут быть положительными целыми числами или 0, где 0 означает отсутствие ребра между вершинами.

Переменная "startVertex" указывает стартовую вершину, с которой начинается построение остоного дерева.

Алгоритм начинает работу путем инициализации массива "minimumSpanningTree" размера size x size, где size - количество вершин в графе. В этом массиве будут храниться найденные ребра остоного дерева.

Массив "visited" представляет собой массив флагов для каждой вершины графа. Начальная вершина (startVertex) помечается как посещенная.

Цикл while продолжает выполняться, пока количество посещенных вершин меньше общего числа вершин в графе.

Внутри цикла происходит поиск минимального ребра, соединяющего посещенную вершину с непосещенной вершиной. Для этого перебираются все вершины, которые уже были посещены (изначально только начальная), и проверяется соседняя вершина, которая еще не была посещена и имеет ненулевой вес ребра с текущей вершиной. Если найдено ребро с меньшим весом, чем текущее минимальное, то обновляются переменные "minWeight", "minFrom" и "minTo".

Если найдено ребро с минимальным весом (т.е. "minFrom" и "minTo" не равны -1), то оно добавляется в минимальное остоное дерево "minimumSpanningTree", обновляются массив "visited" и число посещенных вершин увеличивается на 1.

Если не найдено ребро с минимальным весом (т.е. все вершины уже посещены), то цикл прерывается.

По завершении работы алгоритма возвращается заполненный массив minimumSpanningTree, который представляет собой минимальное остоное дерево графа.

Таким образом, данный код реализации алгоритма Прима и позволяет найти минимальное остоное дерево в графе, основываясь на весах ребер и их связях между вершинами.

## Описание программы

Функция *FillAdjacencyMatrixSingle* которая заполняет матрицу смежности для заданного графа (0 и 1). Давайте рассмотрим, что происходит в этой программе.

Инициализация матрицы: Она начинается с инициализации двумерного массива *adjacencyMatrix* с размерностью *size* x *size*. Этот массив представляет собой матрицу смежности графа.

Цикл для заполнения матрицы: Вложенный цикл перебирает вершины графа. Внутри этого цикла другой вложенный цикл перебирает оставшиеся вершины, начиная с текущей вершины + 1. Это позволяет заполнить только половину матрицы без диагонали, так как граф неориентированный, и значения ребер сохраняются симметрично относительно главной диагонали матрицы.

Ввод значений для ребер: Для каждой пары вершин выводится сообщение с предложением ввести значение для ребра между ними. Пользователь должен ввести 0 или 1, представляющие собой отсутствие или наличие ребра между вершинами.

Проверка корректности ввода: Введенное значение проверяется на соответствие условиям. Если введенное значение не является 0 или 1, выводится сообщение об ошибке, и пользователю предлагается ввести значение заново.

Заполнение матрицы: Если введенное значение соответствует условиям, оно сохраняется в переменную *edgeValue*. Поскольку граф неориентированный, значения ребер сохраняются симметрично относительно главной диагонали матрицы.

Возвращение заполненной матрицы: После заполнения всех ребер матрица смежности возвращается из функции.

Функция *FillAdjacencyMatrix* которая заполняет матрицу смежности для заданного графа (1 и более). Давайте рассмотрим, что происходит в этой программе.

Программа начинает с инициализации двумерного массива *adjacencyMatrix*, размерностью *size* x *size*, предполагая, что *size* - это количество вершин в графе. Затем программа перебирает все вершины графа с помощью двух вложенных циклов *for*. Внутренний цикл перебирает оставшиеся вершины, начиная с текущей вершины *i* + 1. Выполняется это для того, чтобы заполнить только половину матрицы смежности, не включая



диагональ, так как граф неориентированный, и значения ребер сохраняются симметрично относительно главной диагонали матрицы.

Для каждой пары вершин выводится сообщение с предложением ввести значение для ребра между ними. Пользователю предлагается ввести значения для ребер между каждой парой вершин. В случае ошибочного ввода (например, введено отрицательное число или нечисловое значение), программа предлагает ввести корректное значение.

Наконец, поскольку граф неориентированный, значения ребер сохраняются симметрично относительно главной диагонали матрицы, то есть  $\text{adjacencyMatrix}[i, j] = \text{edgeValue}$ ; и  $\text{adjacencyMatrix}[j, i] = \text{edgeValue}$ ; равны. Введенная пользователем информация сохраняется в матрице смежности и возвращается в качестве результата, для дальнейшего использования в программе.

Функция *PrintMatrix*, которая используется для форматированного вывода двумерной матрицы.

В начале функции получается размерность массива *matrix* по первому измерению с помощью метода *GetLength(0)* и сохраняется в переменной *size*.

Далее создается пустая строка с именем *output*, которая будет использоваться для формирования вывода матрицы.

Затем следует вложенный цикл *for*, который дважды перебирает все элементы матрицы с помощью переменных *i* и *j*. Внутри этого цикла к текущему элементу матрицы *matrix[i, j]* добавляется форматированная строка с помощью интерполяции строк, где число будет отформатировано с выравниванием по левому краю и шириной в 4 символа. Данная строка добавляется к переменной *output*.

После завершения внутреннего цикла, добавляется символ новой строки с помощью *Environment.NewLine* для перехода на следующую строку в выводе.

В конце функция возвращает переменную *output*, содержащую отформатированный вывод матрицы.

Функция *PrimAlgorithm*, в котором происходит реализация алгоритма Прима для поиска минимального остовного дерева во взвешенном неориентированном графе

Переменная "matrix" представляет собой двумерный массив, хранящий веса ребер между вершинами графа. Веса ребер могут быть положительными целыми числами или 0, где 0 означает отсутствие ребра между вершинами.

Переменная "startVertex" указывает стартовую вершину, с которой начинается построение остовного дерева.

Алгоритм начинает работу путем инициализации массива "minimumSpanningTree" размера size x size, где size - количество вершин в графе. В этом массиве будут храниться найденные ребра остовного дерева.

Массив "visited" представляет собой массив флагов для каждой вершины графа. Начальная вершина (startVertex) помечается как посещенная.

Цикл while продолжает выполняться, пока количество посещенных вершин меньше общего числа вершин в графе.

Внутри цикла происходит поиск минимального ребра, соединяющего посещенную вершину с непосещенной вершиной. Для этого перебираются все вершины, которые уже были посещены (изначально только начальная), и проверяется соседняя вершина, которая еще не была посещена и имеет ненулевой вес ребра с текущей вершиной. Если найдено ребро с меньшим весом, чем текущее минимальное, то обновляются переменные "minWeight", "minFrom" и "minTo".

Если найдено ребро с минимальным весом (т.е. "minFrom" и "minTo" не равны -1), то оно добавляется в минимальное остовное дерево "minimumSpanningTree", обновляются массив "visited" и число посещенных вершин увеличивается на 1.

Если не найдено ребро с минимальным весом (т.е. все вершины уже посещены), то цикл прерывается.

По завершении работы алгоритма возвращается заполненный массив minimumSpanningTree, который представляет собой минимальное остовное дерево графа.

Функция *CalculateTotalDistance* принимает матрицу matrix, начальную вершину startVertex и объект StreamWriter writer. Она начинает с получения размерности матрицы и создания массива visited для отслеживания посещенных вершин. Затем вызывает CalculateDistanceRecursive для рекурсивного вычисления расстояний.

Функция *CalculateDistanceRecursive* осуществляет рекурсивный обход графа, начиная с текущей вершины currentVertex. Она отмечает текущую вершину как посещенную, затем перебирает смежные вершины и вычисляет расстояние до каждой непосещенной вершины. При этом он увеличивает

totalDistance на вес ребра между текущей и смежной вершинами. Для каждого перехода в следующую вершину он также записывает информацию о переходе в файл с помощью объекта writer, а также выводит это в консоль.

Функция *GenerateAdjacencyMatrix*, которая генерирует случайную матрицу смежности для неориентированного графа.

В начале функции создается объект Random `r = new Random()` для генерации случайных чисел. Затем создается двумерный массив (матрица) `matrix` размерности `size x size` для хранения смежности вершин графа.

Далее следуют два вложенных цикла `for`, которые используются для заполнения этой матрицы. Первый цикл отвечает за итерацию по столбцам матрицы, второй цикл - по строкам.

Внутри второго цикла элементы матрицы `[i, j]` и `[j, i]` получают случайное значение 0 или 1 с помощью `r.Next(0, 2)` для обеспечения случайной симметричной матрицы смежности. Значения ребер между вершинами проверяются на равенство, чтобы обеспечить симметричность графа. Если между вершинами есть ребро (`[i, j] = 1`), то и обратное ребро (`[j, i]`) также устанавливается в 1. Аналогично, если не существует ребра (`[i, j] = 0`), то и обратное ребро (`[j, i]`) также устанавливается в 0.

В конце функции возвращает сгенерированную случайную матрицу смежности для неориентированного графа.

Функция *ConvertToWeightedMatrix*, которая принимает матрицу смежности `adjacencyMatrix` в качестве аргумента и создает новую матрицу с весами `weightedMatrix`.

Сначала функция получает размерность матрицы смежности с помощью вызова `adjacencyMatrix.GetLength(0)` и создает новую пустую матрицу `weightedMatrix` такого же размера.

Затем функция использует вложенный цикл `for` для обхода каждого элемента матрицы смежности. Внутри цикла проверяется, если значение элемента `[i, j]` равно 1, что означает наличие ребра между вершинами `i` и `j`. В этом случае элемент `weightedMatrix[i, j]` заполняется случайным значением от 1 до 10, представляющим вес ребра. Также элемент `weightedMatrix[j, i]` устанавливается равным `weightedMatrix[i, j]` для обеспечения симметричности взвешенной матрицы, так как граф предполагается неориентированным.

В конце функция возвращает новую матрицу смежности с весами `weightedMatrix`, в которой ребра установлены весами от 1 до 10 в зависимости от созданных случайных значений.

## Тестирование

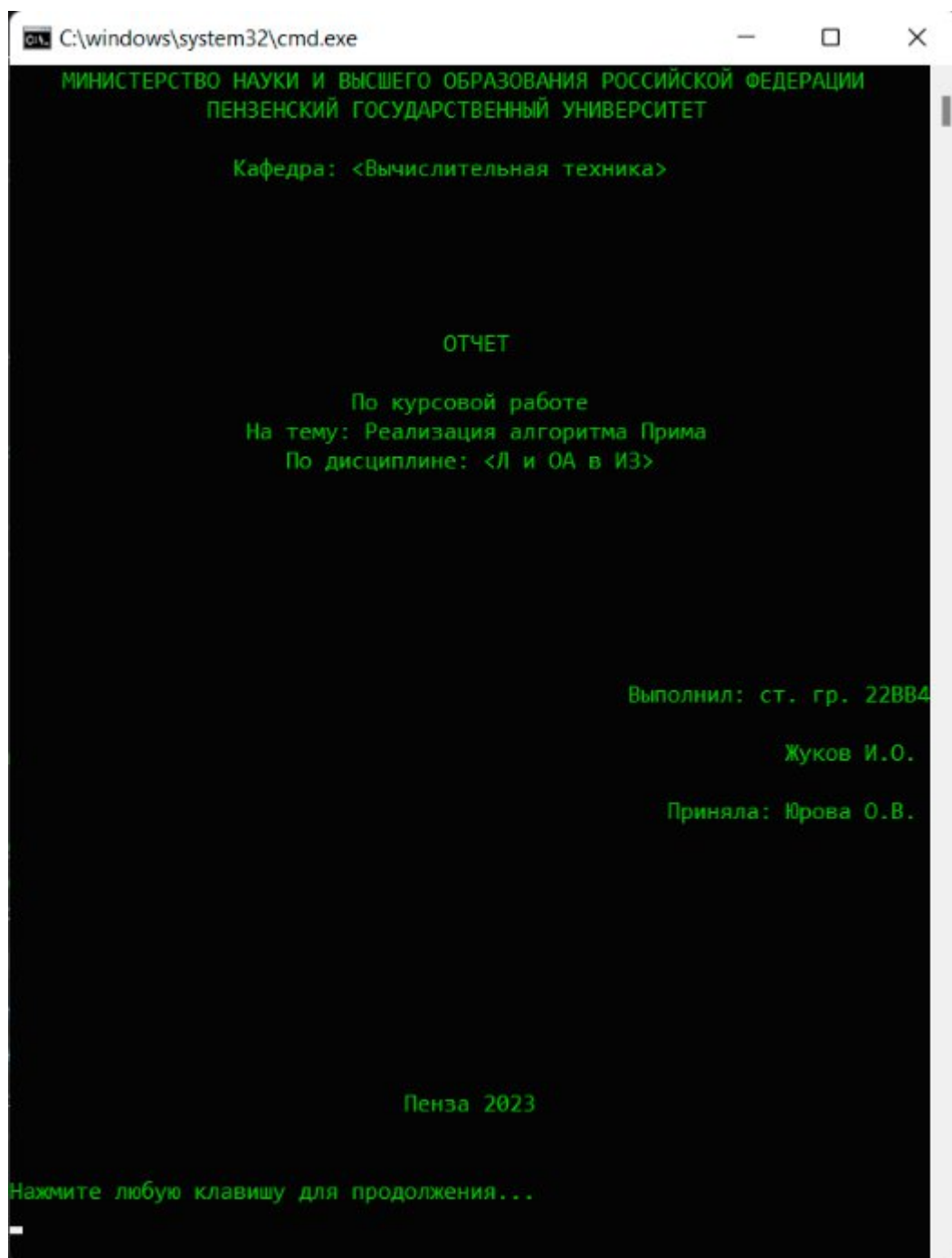
Для написания данной программы использован язык программирования С#. С# (произносится как "си шарп") — современный объектно-ориентированный и типобезопасный язык программирования. С# позволяет разработчикам создавать разные типы безопасных и надежных приложений, выполняющихся в .NET. С# относится к широко известному семейству языков С.

Среда разработки Microsoft Visual Studio 2022 предоставляет все средства, необходимые при разработке и отладке многомодульной программы.

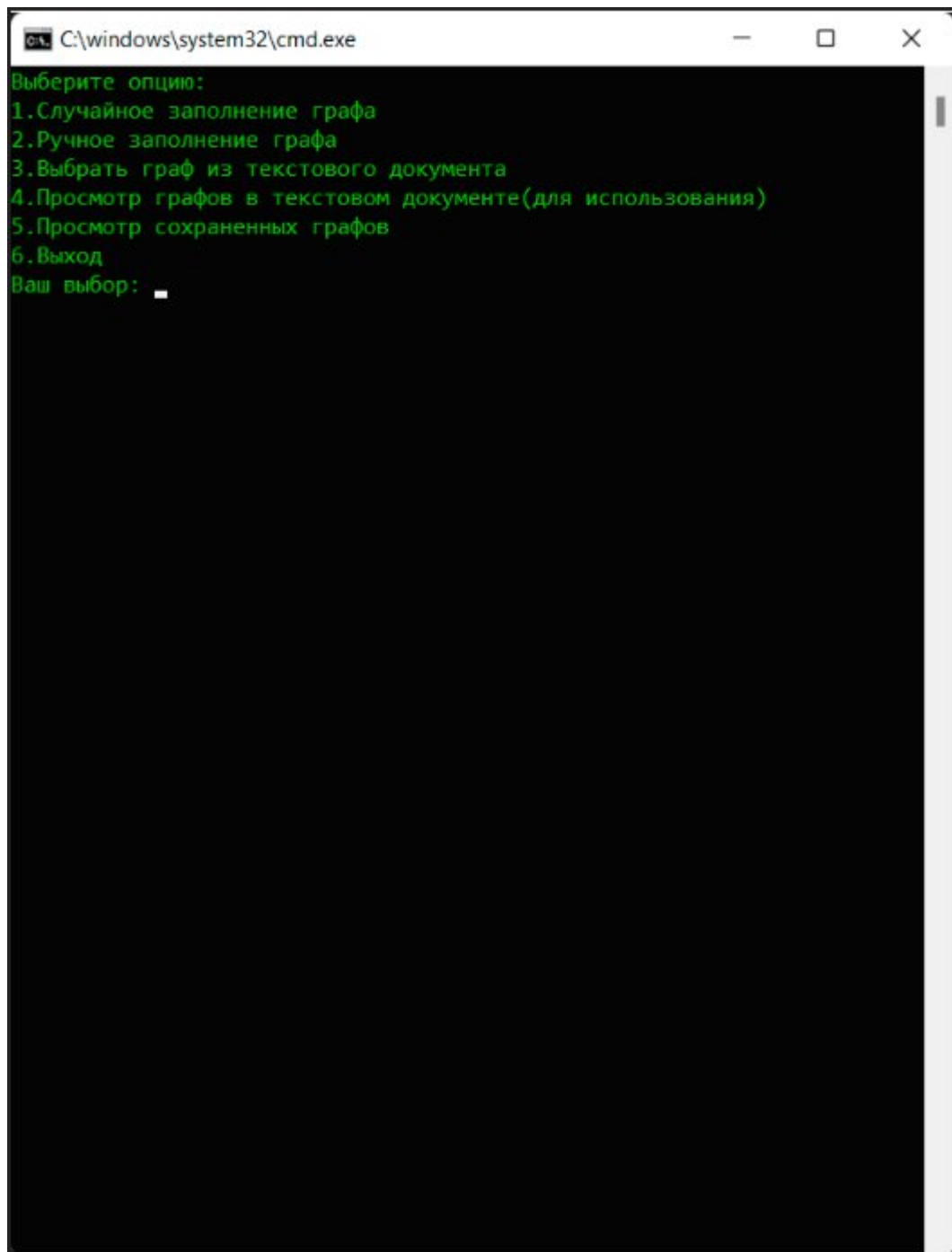
Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы. В ходе тестирования было выявлено и исправлено множество проблем с реализацией алгоритма, вводом данных и т.п.

Ниже продемонстрирован результат тестирования программы:

Вначале пользователя будет встречать титульник по курсовой работе:



В нем пользователю предлагается нажать любую клавишу для продолжения. После ввода пользователем любой клавиши перед ним появляется окно с выбором дальнейшего действия:



При вводе пользователем цифры «1» будет предложено ввести предварительный размер матрицы. После ввода размера появятся 2 матрицы (невзвешенная и взвешенная). На основе невзвешенной матрицы на тех местах, где стояли «1» будут генерироваться случайные значения от 1 до 10, тем самым получится взвешенная матрица, с которой пользователь будет работать в дальнейшем. Далее ему будет предложено ввести вершину, с которой начать обход.

```
C:\windows\system32\cmd.exe
6.Выход
Ваш выбор: 1
Введите размер матрицы: 7
0 0 0 1 0 0 1
0 0 1 1 0 0 1
0 1 0 0 0 0 0
1 1 0 0 0 0 1
0 0 0 0 0 1 1
0 0 0 0 1 0 1
1 1 0 1 1 1 0

Взвешенная матрица смежности для неориентированного графа:
0 0 0 10 0 0 6
0 0 10 9 0 0 4
0 10 0 0 0 0 0
10 9 0 0 0 0 3
0 0 0 0 0 6 9
0 0 0 0 6 0 7
6 4 0 3 9 7 0

Введите вершину, с которой начать обход: 1
Минимальное остовное дерево по алгоритму Прима:
0 0 0 0 0 0 6
0 0 10 0 0 0 4
0 10 0 0 0 0 0
0 0 0 0 0 0 3
0 0 0 0 0 6 0
0 0 0 0 6 0 7
6 4 0 3 0 7 0

Обход графа через самые минимальные ребра:
1 (size 6) -> 7
    7 (size 4) -> 2
        2 (size 10) -> 3
            7 (size 3) -> 4
                7 (size 7) -> 6
                    6 (size 6) -> 5
Итоговая сумма минимального расстояния обхода графа: 36
Хотите сохранить результат работы кода? (да/нет):
```

После ввода вершины реализуется алгоритм Прима, в результате чего появится новая матрица смежности, остовное дерево и итоговая сумма минимального расстояния обхода графа с выбранной пользователем вершины.

Далее пользователю будет предложено сохранить результат. При положительном ответе:

```
Хотите сохранить результат работы кода? (да/нет): да
Результат работы кода сохранен в файле output.txt.
```

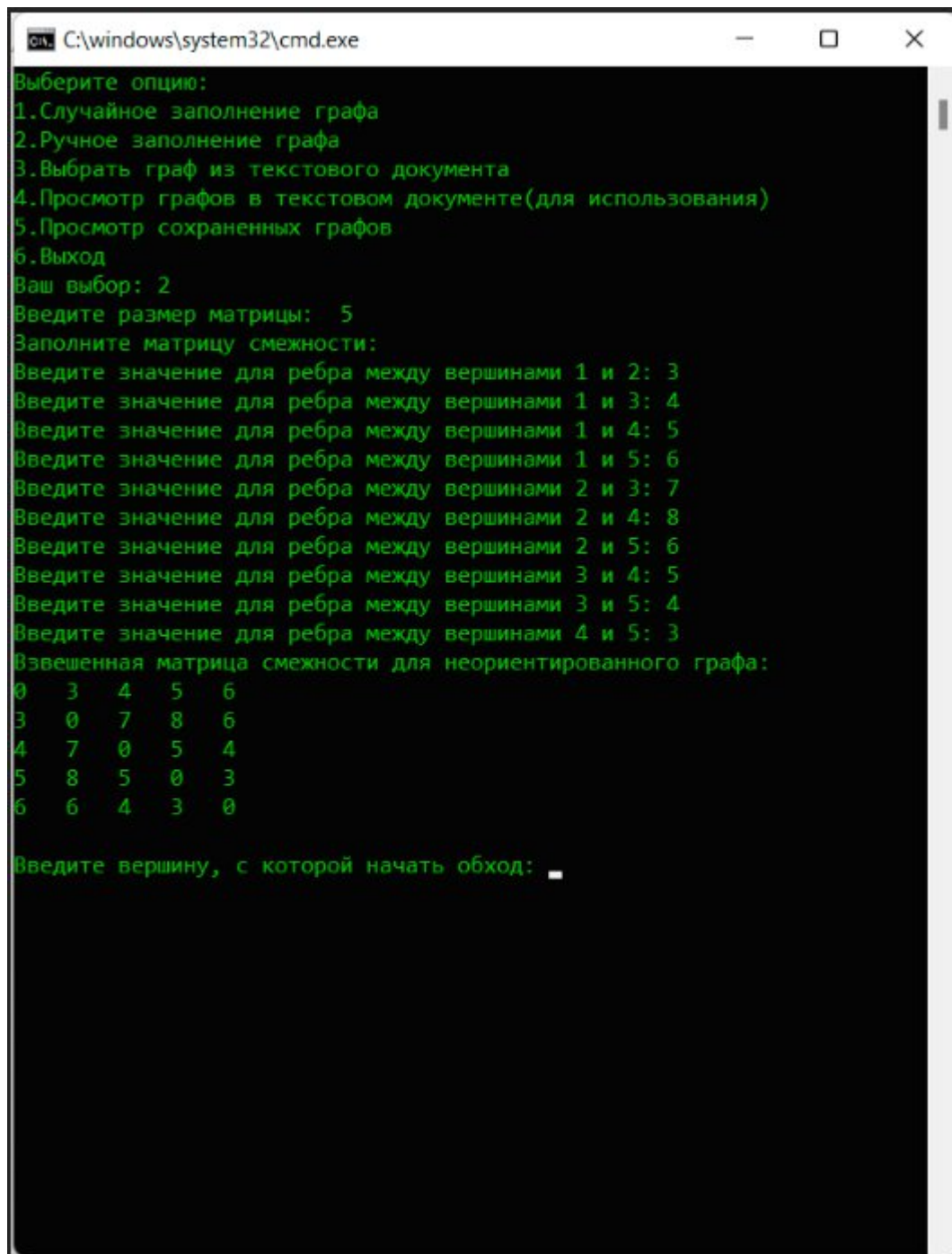
Иначе:

```
Хотите сохранить результат работы кода? (да/нет): нет
Результат работы кода не сохранен.
```

Следующая опция «Ручное заполнение графа»:



При вводе «2» пользователю будет предоставлена возможность заполнить матрицу смежности по своему желанию. Вначале он введет размер матрицы, которую хочет создать, и перейдет к её заполнению:



```
C:\windows\system32\cmd.exe
Выберите опцию:
1.Случайное заполнение графа
2.Ручное заполнение графа
3.Выбрать граф из текстового документа
4.Просмотр графов в текстовом документе(для использования)
5.Просмотр сохраненных графов
6.Выход
Ваш выбор: 2
Введите размер матрицы: 5
Заполните матрицу смежности:
Введите значение для ребра между вершинами 1 и 2: 3
Введите значение для ребра между вершинами 1 и 3: 4
Введите значение для ребра между вершинами 1 и 4: 5
Введите значение для ребра между вершинами 1 и 5: 6
Введите значение для ребра между вершинами 2 и 3: 7
Введите значение для ребра между вершинами 2 и 4: 8
Введите значение для ребра между вершинами 2 и 5: 6
Введите значение для ребра между вершинами 3 и 4: 5
Введите значение для ребра между вершинами 3 и 5: 4
Введите значение для ребра между вершинами 4 и 5: 3
Взвешенная матрица смежности для неориентированного графа:
0  3  4  5  6
3  0  7  8  6
4  7  0  5  4
5  8  5  0  3
6  6  4  3  0

Введите вершину, с которой начать обход: _
```

Далее ему будет предложено ввести вершину, с которой начать обход. После этого реализуется алгоритм Прима, после чего выведется новая матрица смежности, остовное дерево и итоговая сумма минимального расстояния обхода графа с выбранной пользователем вершины:

```

Введите вершину, с которой начать обход: 1
Минимальное остовное дерево по алгоритму Прима:
0  3  4  0  0
3  0  0  0  0
4  0  0  0  4
0  0  0  0  3
0  0  4  3  0

Обход графа через самые минимальные ребра:
1 (size 3) -> 2
1 (size 4) -> 3
    3 (size 4) -> 5
        5 (size 3) -> 4
Итоговая сумма минимального расстояния обхода графа: 14
Хотите сохранить результат работы кода? (да/нет):

```

Как и в первом случае пользователю будет предложено сохранить результат.

При положительном ответе:

```

Хотите сохранить результат работы кода? (да/нет): да
Результат работы кода сохранен в файле output.txt.

```

Иначе:

```

Хотите сохранить результат работы кода? (да/нет): нет
Результат работы кода не сохранен.

```

Следующая опция «Выбрать граф из текстового документа»:

При вводе «3» пользователь может использовать матрицу смежности, которая уже сохранена в текстовом документе, предварительно просмотрев сохраненные в документе матрицы для использования (опция «4»).

В консоли появятся 2 матрицы (невзвешенная и взвешенная), как и в 1 случае, на основе 1-ой заполняется 2-ая. Ему будет предложено ввести вершину, с которой начать обход. После ввода реализуется алгоритм Прима после чего выведется матрица смежности, остовное дерево и итоговая сумма:

```
C:\windows\system32\cmd.exe

3.Выбрать граф из текстового документа
4.Просмотр графов в текстовом документе(для использования)
5.Просмотр сохраненных графов
6.Выход
Ваш выбор: 3
0 0 1 1 0 1 1
0 0 1 0 1 0 1
1 1 0 1 0 1 1
1 0 1 0 1 1 0
0 1 0 1 0 1 1
1 0 1 1 1 0 1
1 1 1 0 1 1 0

Взвешенная матрица смежности для неориентированного графа:
0 0 6 1 0 6 3
0 0 3 0 2 0 2
6 3 0 7 0 3 4
1 0 7 0 10 1 0
0 2 0 10 0 9 9
6 0 3 1 9 0 10
3 2 4 0 9 10 0

Введите вершину, с которой начать обход: 1
Минимальное остовное дерево по алгоритму Прима:
0 0 0 1 0 0 3
0 0 3 0 2 0 2
0 3 0 0 0 0 0
1 0 0 0 0 1 0
0 2 0 0 0 0 0
0 0 0 1 0 0 0
3 2 0 0 0 0 0

Обход графа через самые минимальные ребра:
1 (size 1) -> 4
  4 (size 1) -> 6
1 (size 3) -> 7
  7 (size 2) -> 2
    2 (size 3) -> 3
    2 (size 2) -> 5
Итоговая сумма минимального расстояния обхода графа: 12
Хотите сохранить результат работы кода? (да/нет):
```

Как и в прошлых случаях пользователю будет предложено сохранить результат.

При положительном ответе:

```
Хотите сохранить результат работы кода? (да/нет): да
Результат работы кода сохранен в файле output.txt.
```

Иначе:

```
Хотите сохранить результат работы кода? (да/нет): нет
Результат работы кода не сохранен.
```

Следующая опция «Просмотр графов в текстовом документе(для использования)

При вводе «4» в консоли появиться содержащая в файле матрица смежности, которую можно удалить и перезаписать:

```
Содержимое файла matrixReady.txt:
0 0 1 1 0 1 1
0 0 1 0 1 0 1
1 1 0 1 0 1 1
1 0 1 0 1 1 0
0 1 0 1 0 1 1
1 0 1 1 1 0 1
1 1 1 0 1 1 0

Удалить содержимое файла matrixReady.txt? (да/нет):
```

При удалении содержимое текстового документа очищается. При перезаписи пользователю будет предложено заполнить матрицу случайными значениями(тот же принцип как в опции «1»). Пользователь вводит размер матрицы и генерируется невзвешенная матрица смежности, которая сохраняется в текстовый документ, для дальнейшего пользования в опции «3»:

```
Удалить содержимое файла matrixReady.txt? (да/нет): нет
Содержимое файла matrixReady.txt не было удалено.
Перезаписать матрицу в файле matrixReady.txt? (да/нет): да
Случайное заполнение матрицы? (да/нет): да
Введите размер матрицы: 7
0  0  1  1  1  0  1
0  0  1  1  1  1  0
1  1  0  0  1  0  0
1  1  0  0  1  0  1
1  1  1  1  0  0  0
0  1  0  0  0  0  1
1  0  0  1  0  1  0

Матрица смежности успешно сохранена в файле C:/Users/Илюха/source/repos/example2228/matrixReady.txt.
```

Иначе пользователю будет предоставлена возможность заполнить матрицу смежности вручную:

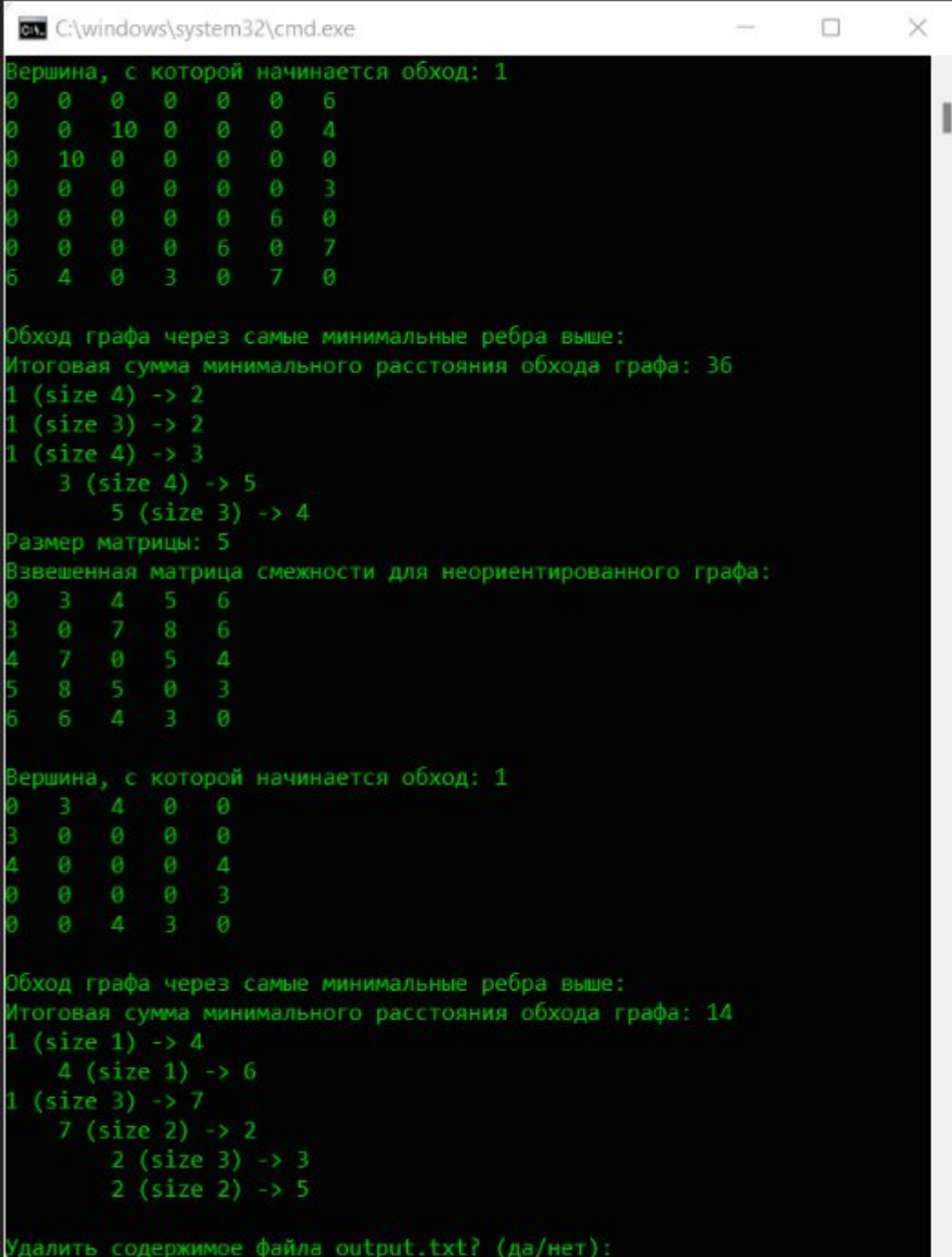
```
Случайное заполнение матрицы? (да/нет): нет
Введите размер матрицы: 4
Заполните матрицу смежности:
Введите значение для ребра между вершинами 1 и 2: 5
Введите значение для ребра между вершинами 1 и 3: 4
Введите значение для ребра между вершинами 1 и 4: 3
Введите значение для ребра между вершинами 2 и 3: 2
Введите значение для ребра между вершинами 2 и 4: 6
Введите значение для ребра между вершинами 3 и 4: 7
0  5  4  3
5  0  2  6
4  2  0  7
3  6  7  0

Матрица смежности успешно сохранена в файле C:/Users/Илюха/source/repos/example2228/matrixReady.txt.
```



Далее опция «просмотр сохраненных графов»:

При вводе «5» пользователь может просмотреть все прошлые действия, которые он сохранял. По своему желанию ему предоставлена возможность удаления истории сохранений:



```
C:\windows\system32\cmd.exe
Вершина, с которой начинается обход: 1
0 0 0 0 0 0 6
0 0 10 0 0 0 4
0 10 0 0 0 0 0
0 0 0 0 0 0 3
0 0 0 0 0 6 0
0 0 0 0 6 0 7
6 4 0 3 0 7 0

Обход графа через самые минимальные ребра выше:
Итоговая сумма минимального расстояния обхода графа: 36
1 (size 4) -> 2
1 (size 3) -> 2
1 (size 4) -> 3
    3 (size 4) -> 5
        5 (size 3) -> 4
Размер матрицы: 5
Взвешенная матрица смежности для неориентированного графа:
0 3 4 5 6
3 0 7 8 6
4 7 0 5 4
5 8 5 0 3
6 6 4 3 0

Вершина, с которой начинается обход: 1
0 3 4 0 0
3 0 0 0 0
4 0 0 0 4
0 0 0 0 3
0 0 4 3 0

Обход графа через самые минимальные ребра выше:
Итоговая сумма минимального расстояния обхода графа: 14
1 (size 1) -> 4
    4 (size 1) -> 6
1 (size 3) -> 7
    7 (size 2) -> 2
        2 (size 3) -> 3
        2 (size 2) -> 5
Удалить содержимое файла output.txt? (да/нет):
```

## Проверка на несведущего человека:

При вводе пользователем числа, большего чем 6:

```
Ваш выбор: 7
Неверная опция, попробуйте ещё раз

Выберите опцию:
1.Случайное заполнение графа
2.Ручное заполнение графа
3.Выбрать граф из текстового документа
4.Просмотр графов в текстовом документе(для использования)
5.Просмотр сохраненных графов
6.Выход
Ваш выбор: _
```

При вводе размера матрицы  $<1$ :

```
Введите размер матрицы: -1
Некорректный ввод. Пожалуйста, введите целое положительное число.
```

При вводе вершины для обхода, которой не существует в графе:

```
Введите вершину, с которой начать обход: 0
Некорректный ввод. Пожалуйста, введите целое положительное ненулевое число.
Введите вершину, с которой начать обход:
```

При вводе числа или иного ответа на вопрос:

```
Хотите сохранить результат работы кода? (да/нет): 1
Некорректный ввод. Пожалуйста, введите 'да' или 'нет'.
```

Ручное заполнение матрицы, при вводе буквы/отрицательного/вещественного числа:

```
Введите значение для ребра между вершинами 1 и 3: д
Некорректный ввод. Пожалуйста, введите целое положительное число.
```

## Описание поведения программы при тестировании

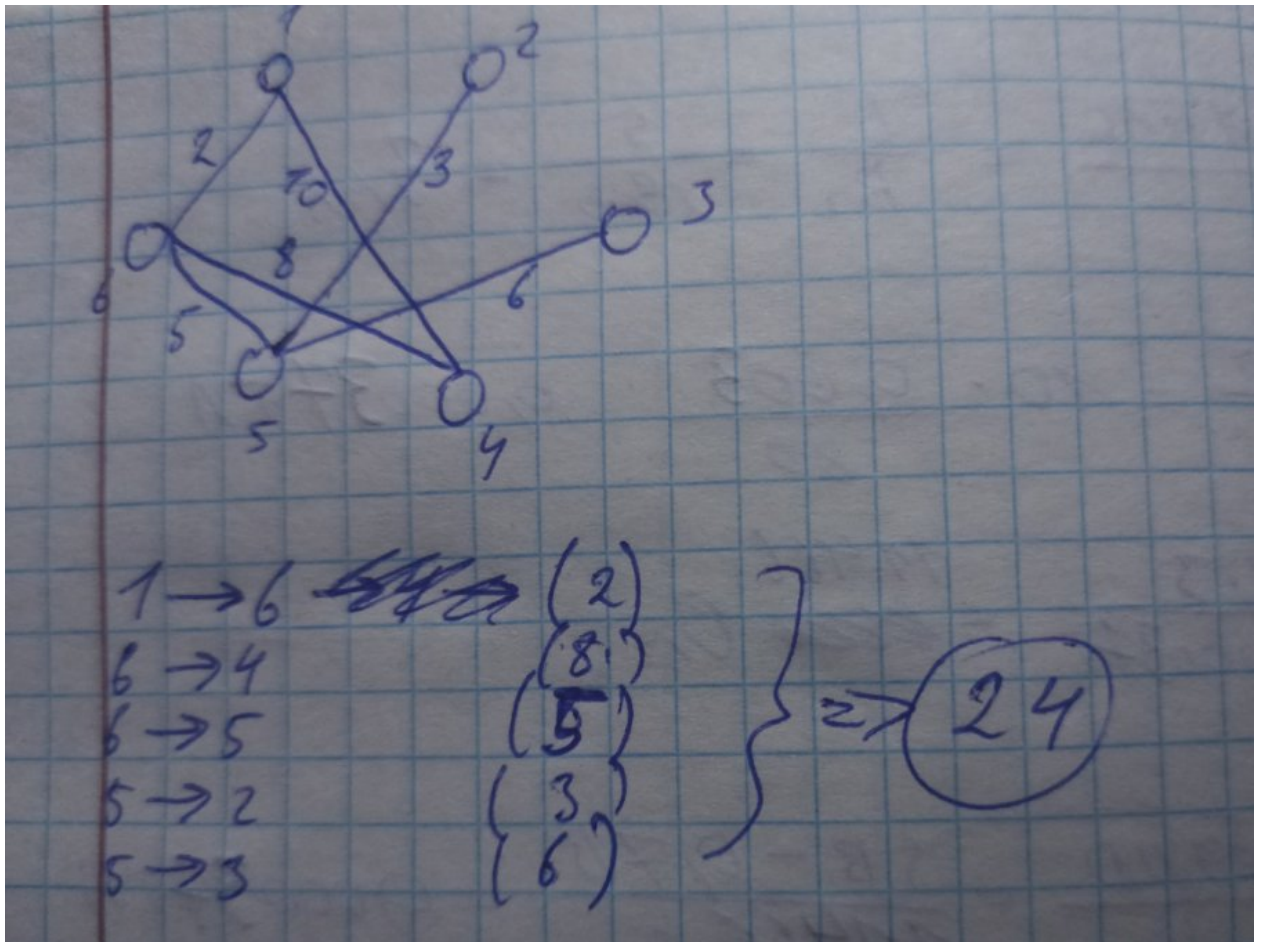
Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод титульника	верно
Выбор случайного заполнения матрицы	Предложение ввести размер матрицы	верно
Выбор ручного заполнения матрицы	Предложение ввести размер матрицы	верно
Просмотр сохраненных графов	Содержимое текстового документа	верно

В результате тестирования было выявлено, что программа успешно проверяет данные на соответствие необходимым требованиям.



## Ручной расчет задачи

Проведем проверку программы посредством ручных вычислений на примере неориентированного взвешенного графа:



Таким образом, можно сделать вывод, что программа работает верно:

Взвешенная матрица смежности для неориентированного графа:

0	0	0	10	0	2
0	0	0	0	3	0
0	0	0	0	6	0
10	0	0	0	0	8
0	3	6	0	0	5
2	0	0	8	5	0

Введите вершину, с которой начать обход: 1

Минимальное остовное дерево по алгоритму Прима:

0	0	0	0	0	2
0	0	0	0	3	0
0	0	0	0	6	0
0	0	0	0	0	8
0	3	6	0	0	5
2	0	0	8	5	0

Обход графа через самые минимальные ребра:

```
1 (size 2) -> 6
    6 (size 8) -> 4
    6 (size 5) -> 5
        5 (size 3) -> 2
        5 (size 6) -> 3
```

Итоговая сумма минимального расстояния обхода графа: 24

## **Заключение**

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Прима, для поиска минимального остовного дерева в Microsoft Visual Studio 2022.

При выполнении данной курсовой работы были получены навыки разработки программ, освоены приемы создания матриц смежности разными способами (случайным, ручным, из текстового документа). Приобретены навыки по осуществлению алгоритма Прима. Углублены знания языка программирования C#.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

## Список литературы

1. C# 9.0. Карманный справочник (2021)
2. "C# 7.0 in a Nutshell: The Definitive Reference" автора Josephy Albahari и Ben Albahari
3. "C# Programming Yellow Book" автора Rob Miles
4. "Introduction to Algorithms" авторов Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest и Clifford Stein
5. "Algorithms Unlocked" автора Thomas H. Cormen
6. "Cracking the Coding Interview: 189 Programming Questions and Solutions" автора Gayle Laakmann McDowell

## Листинг программы:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Security.Policy;
using System.Text;
using System.Threading.Tasks;

namespace examplee
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.SetWindowSize(70, 40); // Устанавливаем размер консоли на
            70 столбцов и 40 строк
            Console.SetBufferSize(70, 40); // Устанавливаем размер буфера консоли
            на 70 столбцов и 40 строк
            Console.BufferHeight = Int16.MaxValue - 1; // Устанавливаем
            максимальную высоту буфера консоли
            Console.ForegroundColor = ConsoleColor.Green; // Устанавливает цвет
            для символов

            string title = "МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
            ФЕДЕРАЦИИ";
            string university = "ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ\r\n\r\n";
            string department = "Кафедра: «Вычислительная
            техника»\r\n\r\n\r\n\r\n\r\n";
            string report = "ОТЧЕТ\r\n\r\n";
            string name = "По курсовой работе";
            string named = "На тему: Реализация алгоритма Прима";
            string discipline = "По дисциплине: «Л и ОА в
            ИЗ»\r\n\r\n\r\n\r\n\r\n\r\n";
            string group = "Выполнил: ст. гр. 22ВВ4";
            string student = "Жуков И.О.\r\n\r\n";
            string teacher = "Приняли: Юрова О.В.\r\n\r\n";
            string city = "Пенза 2023";
            string teacher2 = "Акифьев И.В.\r\n\r\n\r\n\r\n\r\n\r\n\r\n\r\n";
```

```

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 2)
+ (title.Length / 2)) + "}", title));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 2)
+ (title.Length / 3)) + "}", university));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 2)
+ (department.Length / 2)) + "}", department));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 2)
+ (department.Length / 8)) + "}", report));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 2)
+ (name.Length / 2)) + "}", name));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 2)
+ (name.Length / 1)) + "}", named));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 2)
+ (discipline.Length / 2)) + "}", discipline));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 1)
+ (group.Length / 25)) + "}", group));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 1)
+ (group.Length / 20)) + "}", student));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 1)
+ (teacher.Length / 35)) + "}", teacher));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 1)
+ (teacher2.Length / 3)) + "}", teacher2));

        Console.WriteLine(String.Format("{0," + ((Console.WindowWidth / 2)
+ (city.Length / 2)) + "}", city));

```

```

        Console.WriteLine("\n\n" + "Нажмите любую клавишу для
продолжения...");

```

```

        Console.Read();
        Console.Clear();
        Console.ReadLine();

```

```

        string menuOption;
        do
        {
            Console.WriteLine("Выберите опцию:");
            Console.WriteLine("1.Случайное заполнение графа");
            Console.WriteLine("2.Ручное заполнение графа");
            Console.WriteLine("3.Выбрать граф из текстового документа");
            Console.WriteLine("4.Просмотр графов в текстовом документе(для
использования)");
            Console.WriteLine("5.Просмотр сохраненных графов");
            Console.WriteLine("6.Выход");

            Console.Write("Ваш выбор: ");

```

```

menuOption = Console.ReadLine();

switch (menuOption)
{
    case "1":
        //создается новый список строк с именем "output".
        List<string> output2 = new List<string>();
        int size2;
        while (true)
        {
            Console.Write("Введите размер матрицы: ");

            string input = Console.ReadLine();

            //Если введенное значение может быть успешно преобразовано
            //в целое число и не равно 0, -1, то условие "if" будет выполнено и цикл будет
            //прерван с помощью "break".
            if (int.TryParse(input, out size2) && size2 != -1
&& size2 != 0)
            {
                break;
            }

            Console.WriteLine("Некорректный ввод. Пожалуйста,
введите целое положительное число.");
        }
        output2.Add("Размер матрицы: " + size2.ToString());

        int[,] adjacencyMatrix = GenerateAdjacencyMatrix(size2);
        output2.Add("Невзвешенная матрица смежности для
неориентированного графа:");
        string matrixOutput2 = PrintMatrix(adjacencyMatrix);
        output2.Add(matrixOutput2);
        Console.WriteLine(matrixOutput2);

        int[,] weightedMatrix =
ConvertToWeightedMatrix(adjacencyMatrix);
        Console.WriteLine("Взвешенная матрица смежности для
неориентированного графа:");
        string weightedMatrixOutput = PrintMatrix(weightedMatrix);

```

```

        output2.Add("Взвешенная матрица смежности для
неориентированного графа:");
        output2.Add(weightedMatrixOutput);
        Console.WriteLine(weightedMatrixOutput);

        int startVertex2;
        while (true)
        {
            Console.Write("Введите вершину, с которой начать
обход: ");

            string input = Console.ReadLine();

            //Если введенное значение может быть успешно преобразовано
            в целое число и не равно 0, -1 и меньше значения переменной "size2", то условие
            "if" будет выполнено и цикл будет прерван с помощью "break".
            if (int.TryParse(input, out startVertex2) &&
startVertex2 != 0 && startVertex2 != -1 && startVertex2 < size2)
            {
                break;
            }

            Console.WriteLine("Некорректный ввод. Пожалуйста,
введите целое положительное ненулевое число.");
        }

        output2.Add("Вершина, с которой начинается обход: " +
startVertex2.ToString());
        int[,] minimumSpanningTree2 = PrimAlgorithm(weightedMatrix,
startVertex2);

        Console.WriteLine("Минимальное остовное дерево по алгоритму
Прима:");

        string minimumSpanningTreeOutput2 =
PrintMatrix(minimumSpanningTree2);
        output2.Add(minimumSpanningTreeOutput2);
        Console.WriteLine(minimumSpanningTreeOutput2);

        output2.Add("Обход графа через самые минимальные ребра
выше:");
        Console.WriteLine("Обход графа через самые минимальные
ребра:");

```



```

        //используется StreamWriter для записи результата работы
        //кода в файл output.txt.
        using (StreamWriter writer = new
StreamWriter("C:/Users/Илюха/source/repos/example2228/output.txt", true))
        {
            int totalDistance2 =
CalculateTotalDistance(minimumSpanningTree2, startVertex2, writer);
            output2.Add("Итоговая сумма минимального расстояния
обхода графа: " + totalDistance2.ToString());
            Console.WriteLine("Итоговая сумма минимального расстояния
обхода графа: " + totalDistance2.ToString());
        }

        string saveChoice;
        while (true)
        {
            Console.Write("Хотите сохранить результат работы
кода? (да/нет): ");

            string input = Console.ReadLine();

            //Если введенное значение равно "да" или "нет" (без
учета регистра), то условие "if" будет выполнено и цикл будет прерван с помощью
"break".
            if (StringComparison.OrdinalIgnoreCase) || (input.Equals("да",
StringComparison.OrdinalIgnoreCase) || input.Equals("нет",
StringComparison.OrdinalIgnoreCase))
            {
                saveChoice = input;
                break;
            }

            //Если введенное значение не прошло проверку, выводится
сообщение об ошибке "Некорректный ввод. Пожалуйста, введите 'да' или 'нет'.",
и цикл продолжает выполняться заново.
            Console.WriteLine("Некорректный ввод. Пожалуйста,
введите 'да' или 'нет'.");
        }
        if (saveChoice.ToLower() == "да")
        {
            string outputPath2 =
"C:/Users/Илюха/source/repos/example2228/output.txt";

            //Используя класс "StreamWriter", создается объект
"writer" для записи в файл по указанному пути.

```

```

        using (StreamWriter writer = new StreamWriter(outputPath2,
true))
        {
            //Происходит запись каждой строки из массива
"output2" в файл с помощью цикла "foreach" и метода "WriteLine()".
            foreach (string line in output2)
            {
                //Закрывается объект "writer".
                writer.WriteLine(line);
            }
            writer.Close();
        }
        Console.WriteLine("Результат работы кода сохранен
в файле output.txt.");
    }
    else if (saveChoice.ToLower() == "нет")
    {
        Console.WriteLine("Результат работы кода не сохранен.");
    }
    break;
case "2":
    //создается новый список строк с именем "output".
    List<string> output = new List<string>();
    int size;

    while (true)
    {
        Console.Write("Введите размер матрицы: ");

        string input = Console.ReadLine();

        //Если введенное значение может быть успешно преобразовано
в целое число и не равно 0, -1, то условие "if" будет выполнено и цикл будет
прерван с помощью "break".
        if (int.TryParse(input, out size) && size != -1 &&
size != 0)
        {
            break;
        }
    }

```

```

        Console.WriteLine("Некорректный ввод. Пожалуйста,
введите целое положительное число.");
    }
    output.Add("Размер матрицы: " + size.ToString());

    int[,] weightedMatrix2 = FillAdjacencyMatrix(size);
    output.Add("Взвешенная матрица смежности для
неориентированного графа:");
    Console.WriteLine("Взвешенная матрица смежности для
неориентированного графа:");

    string matrixOutput = PrintMatrix(weightedMatrix2);
    output.Add(matrixOutput);
    Console.WriteLine(matrixOutput);

    int startVertex;
    while (true)
    {
        Console.Write("Введите вершину, с которой начать
обход: ");

        string input = Console.ReadLine();

        //Если введенное значение может быть успешно преобразовано
        в целое число и не равно 0, -1 и меньше значения переменной "size2", то условие
        "if" будет выполнено и цикл будет прерван с помощью "break".
        if (int.TryParse(input, out startVertex) &&
startVertex != 0 && startVertex != -1 && startVertex < size)
        {
            break;
        }

        Console.WriteLine("Некорректный ввод. Пожалуйста,
введите целое положительное ненулевое число.");
    }

    output.Add("Вершина, с которой начинается обход: " +
startVertex.ToString());
    int[,] minimumSpanningTree = PrimAlgorithm(weightedMatrix2,
startVertex);

```

```

        Console.WriteLine("Минимальное остовное дерево по алгоритму
Прима:");

        matrixOutput = PrintMatrix(minimumSpanningTree);
        output.Add(matrixOutput);
        Console.WriteLine(matrixOutput);

        output.Add("Обход графа через самые минимальные ребра
выше:");
        Console.WriteLine("Обход графа через самые минимальные
ребра:");

        //используется StreamWriter для записи результата работы
кода в файл output.txt.
        using (StreamWriter writer = new
StreamWriter("C:/Users/Илюха/source/repos/example2228/output.txt", true))
        {
            int totalDistance2 =
CalculateTotalDistance(minimumSpanningTree, startVertex, writer);
            output.Add("Итоговая сумма минимального расстояния
обхода графа: " + totalDistance2.ToString());
            Console.WriteLine("Итоговая сумма минимального расстояния
обхода графа: " + totalDistance2.ToString());
        }

        string saveChoice2;
        while (true)
        {
            Console.Write("Хотите сохранить результат работы
кода? (да/нет): ");

            string input = Console.ReadLine();

            //Если введенное значение равно "да" или "нет" (без
учета регистра), то условие "if" будет выполнено и цикл будет прерван с помощью
"break".
            if (StringComparison.OrdinalIgnoreCase) || (input.Equals("да",
StringComparison.OrdinalIgnoreCase) || input.Equals("нет",
StringComparison.OrdinalIgnoreCase))
            {
                saveChoice2 = input;
                break;
            }
        }
    }
}

```

```

        Console.WriteLine("Некорректный ввод. Пожалуйста,
введите 'да' или 'нет'.");
    }

    if (saveChoice2.ToLower() == "да")
    {
        //Используя класс "StreamWriter", создается объект
"writer" для записи в файл по указанному пути.
        using (StreamWriter writer = new
StreamWriter("C:/Users/Илюха/source/repos/example2228/output.txt", true))
        {
            foreach (string line in output)
            {
                //Происходит запись каждой строки из массива
"output" в файл с помощью цикла "foreach" и метода "WriteLine()".
                writer.WriteLine(line);
            }
            //закрывается объект "writer"
            writer.Close();
        }
        Console.WriteLine("Результат работы кода сохранен
в файле output.txt.");
    }
    else
    {
        Console.WriteLine("Результат работы кода не сохранен.");
    }
    break;
case "3":
    //Создается пустой список строк с именем output3.
    List<string> output3 = new List<string>();

    //Считывается содержимое файла matrixReady.txt и сохраняется
в массив строк с именем lines.
    string[] lines =
File.ReadAllLines("C:/Users/Илюха/source/repos/example2228/matrixReady.txt");

    //Получается размерность size3, равная количеству строк
в файле.
    int size3 = lines.Length;

```

```

size3.                                //Создается двумерный массив matrix размером size3 на
int[,] matrix = new int[size3, size3];

//Запускается цикл, который проходит по каждой строке
массива lines.
for (int i = 0; i < size3; i++)
{
    //Внутри этого цикла строка разделяется на отдельные
значения, используя разделитель ' ', и сохраняется в массив строк с именем
values.

    string[] values = lines[i].Split(' ');

    //Запускается вложенный цикл, который проходит по
каждому элементу массива values.
    for (int j = 0; j < size3; j++)
    {
        //Каждый элемент, представляющий строкой,
преобразуется в целое число с помощью метода int.Parse() и сохраняется в
соответствующую ячейку массива matrix.
        matrix[i, j] = int.Parse(values[j]);
    }
}

output3.Add("Невзвешенная матрица смежности для
неориентированного графа:");
string matrixOutput3 = PrintMatrix(matrix);
output3.Add(matrixOutput3);
Console.WriteLine(matrixOutput3);

int[,] weightedMatrix3 = ConvertToWeightedMatrix(matrix);
Console.WriteLine("Взвешенная матрица смежности для
неориентированного графа:");
string weightedMatrixOutput3 = PrintMatrix(weightedMatrix3);
output3.Add("Взвешенная матрица смежности для
неориентированного графа:");
output3.Add(weightedMatrixOutput3);
Console.WriteLine(weightedMatrixOutput3);

int startVertex3;
while (true)
{

```

```

        Console.WriteLine("Введите вершину, с которой начать
обход: ");

        string input = Console.ReadLine();

        //Если введенное значение может быть успешно преобразовано
        в целое число и не равно 0, -1 и меньше значения переменной "size3", то условие
        "if" будет выполнено и цикл будет прерван с помощью "break".
        if (int.TryParse(input, out startVertex3) &&
startVertex3 != 0 && startVertex3 != -1 && startVertex3 < size3)
        {
            break;
        }

        Console.WriteLine("Некорректный ввод. Пожалуйста,
введите целое положительное ненулевое число.");
    }

    output3.Add("Вершина, с которой начинается обход: " +
startVertex3.ToString());
    int[,] minimumSpanningTree3 = PrimAlgorithm(weightedMatrix3,
startVertex3);

    Console.WriteLine("Минимальное остовное дерево по алгоритму
Прима:");

    string minimumSpanningTreeOutput3 =
PrintMatrix(minimumSpanningTree3);
    output3.Add(minimumSpanningTreeOutput3);
    Console.WriteLine(minimumSpanningTreeOutput3);

    output3.Add("Обход графа через самые минимальные ребра
выше:");
    Console.WriteLine("Обход графа через самые минимальные
ребра:");

    //используется StreamWriter для записи результата работы
    кода в файл output.txt.
    using (StreamWriter writer = new
StreamWriter("C:/Users/Илюха/source/repos/example2228/output.txt", true))
    {
        int totalDistance2 =
CalculateTotalDistance(minimumSpanningTree3, startVertex3, writer);
        output3.Add("Итоговая сумма минимального расстояния
обхода графа: " + totalDistance2.ToString());
    }

```

```

        Console.WriteLine("Итоговая сумма минимального расстояния
обхода графа: " + totalDistance2.ToString());
    }

    string saveChoice3;
    while (true)
    {
        Console.Write("Хотите сохранить результат работы
кода? (да/нет): ");

        string input = Console.ReadLine();

        //Если введенное значение равно "да" или "нет" (без
учета регистра), то условие "if" будет выполнено и цикл будет прерван с помощью
"break".

        if (StringComparison.OrdinalIgnoreCase == input.Equals("да",
StringComparison.OrdinalIgnoreCase) || input.Equals("нет",
StringComparison.OrdinalIgnoreCase))
        {
            saveChoice3 = input;
            break;
        }

        Console.WriteLine("Некорректный ввод. Пожалуйста,
введите 'да' или 'нет'.");
    }

    if (saveChoice3.ToLower() == "да")
    {
        string outputPath2 =
"C:/Users/Илюха/source/repos/example2228/output.txt";

        //Используя класс "StreamWriter", создается объект
"writer" для записи в файл по указанному пути.
        using (StreamWriter writer = new StreamWriter(outputPath2,
true))
        {
            foreach (string line in output3)
            {
                //Происходит запись каждой строки из массива
"output" в файл с помощью цикла "foreach" и метода "WriteLine()".
                writer.WriteLine(line);
            }
        }
    }

```



```

        //закрывается объект "writer"
        writer.Close();
    }
    Console.WriteLine("Результат работы кода сохранен
в файле output.txt.");
}
else
{
    Console.WriteLine("Результат работы кода не сохранен.");
}
break;
case "4":
    //Создается пустой список строк с именем output4
    List<string> output4 = new List<string>();

    string viewChoice;
    while (true)
    {
        Console.Write("Хотите посмотреть доступные графы?
(да/нет): ");

        string input = Console.ReadLine();

        //Если введенное значение равно "да" или "нет" (без
учета регистра), то условие "if" будет выполнено и цикл будет прерван с помощью
"break".
        if (StringComparison.OrdinalIgnoreCase == (input.Equals("да",
StringComparison.OrdinalIgnoreCase) || input.Equals("нет",
StringComparison.OrdinalIgnoreCase)))
        {
            viewChoice = input;
            break;
        }

        Console.WriteLine("Некорректный ввод. Пожалуйста,
введите 'да' или 'нет'.");
    }

    if (viewChoice.ToLower() == "да")
    {

```

```

        string outputPath2 =
"C:/Users/Илюха/source/repos/example2228/matrixReady.txt";

        //Считывается содержимое файла matrixReady.txt и
сохраняется в строку fileContent
        string fileContent = File.ReadAllText(outputPath2);
        Console.WriteLine("Содержимое файла matrixReady.txt:");
        Console.WriteLine(fileContent);

        string deleteChoice;
        while (true)
        {
            Console.Write("Удалить содержимое файла
matrixReady.txt? (да/нет): ");

            string input = Console.ReadLine();

            //Если введенное значение равно "да" или "нет"
(без учета регистра), то условие "if" будет выполнено и цикл будет прерван с
помощью "break".
            if (input.Equals("да",
StringComparison.OrdinalIgnoreCase) ||
input.Equals("нет",
StringComparison.OrdinalIgnoreCase))
            {
                deleteChoice = input;
                break;
            }

            Console.WriteLine("Некорректный ввод. Пожалуйста,
введите 'да' или 'нет'.");
        }

        if (deleteChoice.ToLower() == "да")
        {
            //перезапись существующего файла с именем "outputPath2"
и записи в него пустой строки.
            File.WriteAllText(outputPath2, string.Empty);
            Console.WriteLine("Содержимое файла matrixReady.txt
успешно удалено.");
        }
        else
        {

```

```

        Console.WriteLine("Содержимое файла matrixReady.txt
не было удалено.");
    }

    string newMatChoise;
    string method;
    while (true)
    {
        Console.Write("Перезаписать матрицу в файле
matrixReady.txt? (да/нет): ");

        string input = Console.ReadLine();

        //Если введенное значение равно "да" или "нет"
        (без учета регистра), то условие "if" будет выполнено и цикл будет прерван с
        помощью "break".

        if (input.Equals("да",
StringComparison.OrdinalIgnoreCase) || input.Equals("нет",
StringComparison.OrdinalIgnoreCase))
        {
            newMatChoise = input;
            break;
        }

        Console.WriteLine("Некорректный ввод. Пожалуйста,
введите 'да' или 'нет'.");
    }

    if (newMatChoise.ToLower() == "да")
    {
        while (true)
        {
            Console.Write("Случайное заполнение матрицы?
(да/нет): ");

            string input = Console.ReadLine();

            //Если введенное значение равно "да" или
            "нет" (без учета регистра), то условие "if" будет выполнено и цикл будет прерван
            с помощью "break".

            if (input.Equals("да",
StringComparison.OrdinalIgnoreCase) || input.Equals("нет",
StringComparison.OrdinalIgnoreCase))

```

```

        {
            method = input;
            break;
        }

        Console.WriteLine("Некорректный ввод. Пожалуйста,
введите 'да' или 'нет'.");
    }

    if (method.ToLower() == "да")
    {
        int size4;
        while (true)
        {
            Console.Write("Введите размер матрицы:
");

            string input = Console.ReadLine();

            //Если введенное значение может быть
            //успешно преобразовано в целое число и не равно 0, -1, то условие "if" будет
            //выполнено и цикл будет прерван с помощью "break".
            if (int.TryParse(input, out size4) &&
size4 != -1 && size4 != 0)
            {
                break;
            }

            Console.WriteLine("Некорректный ввод.
Пожалуйста, введите целое положительное число.");
        }

        int[,] adjacencyMatrix4 =
GenerateAdjacencyMatrix(size4);
        string matrixOutput4 =
PrintMatrix(adjacencyMatrix4);
        output4.Add(matrixOutput4);
        Console.WriteLine(matrixOutput4);

        //Создается экземпляр класса StringBuilder
        под названием `matrixString`.

```

```

        //StringBuilder представляет изменяемую строку
        для эффективной конкатенации и сборки строк.
        StringBuilder matrixString = new StringBuilder();

        //выполняется двойной цикл for для итерации
        по элементам матрицы смежности.
        for (int i = 0; i < size4; i++)
        {
            for (int j = 0; j < size4; j++)
            {
                //При каждой итерации, значение элемента
                `adjacencyMatrix4[i, j]` добавляется в `matrixString` с помощью метода
                `Append()`.
                //Символ пробела также добавляется
                после каждого значения, чтобы создать отступ между числами.
                matrixString.Append(adjacencyMatrix4[i,
j]);

                matrixString.Append(" ");
            }
            //После каждой строки матрицы добавляется
            символ новой строки с помощью метода `AppendLine()`.
            matrixString.AppendLine();
        }
        //вызывается метод `ToString()` у объекта
        `matrixString` для получения результирующей строки матрицы.
        //Полученная строка матрицы записывается
        в файл с помощью метода `File.WriteAllText()`.
        //Путь к файлу предоставляется в переменной
        `outputPath2`.
        File.WriteAllText(outputPath2,
matrixString.ToString());
        Console.WriteLine($"Матрица смежности успешно
сохранена в файле {outputPath2}.");
    }
    else
    {
        int size4;
        while (true)
        {
            Console.Write("Введите размер матрицы:
");

            string input = Console.ReadLine();

```

```

//Если введенное значение может быть
успешно преобразовано в целое число и не равно 0, -1, то условие "if" будет
выполнено и цикл будет прерван с помощью "break".
if (int.TryParse(input, out size4) &&
size4 != -1 && size4 != 0)
{
    break;
}

Console.WriteLine("Некорректный ввод.
Пожалуйста, введите целое положительное число.");
}

int[,] weightedMatrix4 =
FillAdjacencyMatrixSingle(size4);

string matrixOutput4 =
PrintMatrix(weightedMatrix4);

output4.Add(matrixOutput4);
Console.WriteLine(matrixOutput4);

//Создается экземпляр класса StringBuilder
под названием `matrixString`.
//StringBuilder представляет изменяемую строку
для эффективной конкатенации и сборки строк.
StringBuilder matrixString = new StringBuilder();

//выполняется двойной цикл for для итерации
по элементам матрицы смежности.
for (int i = 0; i < size4; i++)
{
    for (int j = 0; j < size4; j++)
    {
        //При каждой итерации, значение элемента
`weightedMatrix4[i, j]` добавляется в `matrixString` с помощью метода
`Append()`.
        //Символ пробела также добавляется
после каждого значения, чтобы создать отступ между числами.
matrixString.Append(weightedMatrix4[i,
j]);

matrixString.Append(" ");
}
}

```

```

        //После каждой строки матрицы добавляется
        символ новой строки с помощью метода `AppendLine()`.
        matrixString.AppendLine();
    }
    //вызывается метод `ToString()` у объекта
    `matrixString` для получения результирующей строки матрицы.
    //Полученная строка матрицы записывается
    в файл с помощью метода `File.WriteAllText()`.
    //Путь к файлу предоставляется в переменной
    `outputPath2`.
    File.WriteAllText(outputPath2,
matrixString.ToString());
    Console.WriteLine($"Матрица смежности успешно
сохранена в файле {outputPath2}.");
    }

    }
    else
    {
        Console.WriteLine($"Файл {outputPath2} не был
изменен.");
    }
}
break;
case "5":
    string viewChoice2;
    while (true)
    {
        Console.Write("Просмотреть содержимое файла output.txt?
(да/нет): ");

        string input = Console.ReadLine();

        //Если введенное значение равно "да" или "нет" (без
        учета регистра), то условие "if" будет выполнено и цикл будет прерван с помощью
        "break".
        if
        StringComparison.OrdinalIgnoreCase) || (input.Equals("да",
StringComparison.OrdinalIgnoreCase)) || input.Equals("нет",
        {
            viewChoice2 = input;
            break;
        }
    }
}

```

```

        Console.WriteLine("Некорректный ввод. Пожалуйста,
введите 'да' или 'нет'.");
    }

    if (viewChoice2.ToLower() == "да")
    {
        string outputPath2 =
"C:/Users/Илюха/source/repos/example2228/output.txt";
        string fileContent = File.ReadAllText(outputPath2);
        Console.WriteLine("Содержимое файла output.txt:");
        Console.WriteLine(fileContent);

        string deleteChoice;
        while (true)
        {
            Console.Write("Удалить содержимое файла output.txt?
(да/нет): ");

            string input = Console.ReadLine();

            //Если введенное значение равно "да" или "нет"
            (без учета регистра), то условие "if" будет выполнено и цикл будет прерван с
            помощью "break".

            if (input.Equals("да",
StringComparison.OrdinalIgnoreCase) ||
            input.Equals("нет",
StringComparison.OrdinalIgnoreCase))
            {
                deleteChoice = input;
                break;
            }

            Console.WriteLine("Некорректный ввод. Пожалуйста,
введите 'да' или 'нет'.");
        }

        if (deleteChoice.ToLower() == "да")
        {
            //перезапись существующего файла с именем "outputPath2"
            и записи в него пустой строки.

            File.WriteAllText(outputPath2, string.Empty);

```



```

        Console.WriteLine("Содержимое файла output.txt
успешно удалено.");
    }
    else
    {
        Console.WriteLine("Содержимое файла output.txt
не было удалено.");
    }
}
break;
case "6":
    Console.WriteLine("Программа завершена");
    break;
default:
    {
        Console.WriteLine("Неверная опция, попробуйте ещё
раз");
        break;
    }
}
Console.WriteLine();
} while (menuOption != "6");
}
//Метод, который заполняет матрицу смежности для графа.
private static int[,] FillAdjacencyMatrixSingle(int size)
{
    //происходит инициализация двумерного массива adjacencyMatrix с
размерностью size x size
    int[,] adjacencyMatrix = new int[size, size];
    Console.WriteLine("Заполните матрицу смежности:");

    //начинается цикл, который перебирает все вершины графа.
    //Внутри цикла есть еще один цикл, который перебирает оставшиеся
вершины, начиная с текущей вершины + 1.
    //Это позволяет заполнить только половину матрицы, не включая
диагональ, поскольку граф неориентированный
    //и значения ребер сохраняются симметрично относительно главной
диагонали матрицы.
    for (int i = 0; i < size; i++)
    {

```

```

        for (int j = i + 1; j < size; j++) // Измененное условие для
заполнения только половины матрицы, не включая диагональ
        {
            //Для каждой пары вершин выводится сообщение с предложением
ввести значение для ребра между ними.
            Console.WriteLine($"Введите значение для ребра между вершинами
{i + 1} и {j + 1}: ");

            //предлагается ввести значения для ребер между каждой парой
вершин

            int edgeValue;

            //используется цикл while для ввода значения ребра. Если
вводимое значение является целым положительным числом,
            //то оно сохраняется в переменную edgeValue и цикл прерывается.
Если введенное значение не соответствует условиям,
            //выводится сообщение об ошибке и требуется ввести корректное
значение.

            while (true)
            {
                string input = Console.ReadLine();

                if (input == "0" || input == "1") // Проверяем, является
ли введенное значение 0 или 1
                {
                    edgeValue = int.Parse(input);
                    break;
                }

                Console.WriteLine("Некорректный ввод. Пожалуйста, введите
только 0 или 1.");
            }

            //Поскольку граф неориентированный, значения ребер сохраняются
симметрично относительно главной диагонали матрицы.
            adjacencyMatrix[i, j] = edgeValue;
            adjacencyMatrix[j, i] = edgeValue;
        }
    }

    return adjacencyMatrix;
}

//Метод, который заполняет матрицу смежности для графа.
private static int[,] FillAdjacencyMatrix(int size)

```

```

    {
        //происходит инициализация двумерного массива adjacencyMatrix с
размерностью size x size
        int[,] adjacencyMatrix = new int[size, size];
        Console.WriteLine("Заполните матрицу смежности:");

        //начинается цикл, который перебирает все вершины графа.
        //Внутри цикла есть еще один цикл, который перебирает оставшиеся
вершины, начиная с текущей вершины + 1.
        //Это позволяет заполнить только половину матрицы, не включая
диагональ, поскольку граф неориентированный
        //и значения ребер сохраняются симметрично относительно главной
диагонали матрицы.
        for (int i = 0; i < size; i++)
        {
            for (int j = i + 1; j < size; j++) // Измененное условие для
заполнения только половины матрицы, не включая диагональ
            {
                //Для каждой пары вершин выводится сообщение с предложением
ввести значение для ребра между ними.
                Console.Write($"Введите значение для ребра между вершинами
{i + 1} и {j + 1}: ");

                //предлагается ввести значения для ребер между каждой парой
вершин

                int edgeValue;

                //используется цикл while для ввода значения ребра. Если
вводимое значение является целым положительным числом,
                //то оно сохраняется в переменную edgeValue и цикл прерывается.
Если введенное значение не соответствует условиям,
                //выводится сообщение об ошибке и требуется ввести корректное
значение.

                while (true)
                {
                    string input = Console.ReadLine();

                    if (int.TryParse(input, out edgeValue) && edgeValue !=
-1)
                    {
                        break;
                    }
                }
            }
        }
    }

```

```

        Console.WriteLine("Некорректный ввод. Пожалуйста, введите
целое положительное число.");
    }

    //Поскольку граф неориентированный, значения ребер сохраняются
симметрично относительно главной диагонали матрицы.
    adjacencyMatrix[i, j] = edgeValue;
    adjacencyMatrix[j, i] = edgeValue;
}
}
return adjacencyMatrix;
}
//Вывод матрицы на экран
static string PrintMatrix(int[,] matrix)
{
    //Получаем размерность массива matrix по первому измерению и сохраняет
в переменной size.
    int size = matrix.GetLength(0);

    //Создаем пустую строку с именем output, которая будет использоваться
для формирования вывода матрицы.
    string output = "";
    for (int i = 0; i < size; i++)
    {
        for (int j = 0; j < size; j++)
        {
            //К текущему элементу матрицы matrix[i, j] добавляется
форматированная строка,
            //где число будет отформатировано с выравниванием по левому
краю и шириной в 4 символа.
            //Затем эта строка добавляется к output.
            output += $"{matrix[i, j],-4}";
        }
        //После каждой строки матрицы добавляется символ новой строки
для перехода на следующую строку в выводе.
        output += Environment.NewLine;
    }
    return output;
}

//Реализация алгоритма Прима для поиска минимального остовного дерева
во взвешенном неориентированном графе

```

```

static int[,] PrimAlgorithm(int[,] matrix, int startVertex)
{
    //Получаем размерность (количество вершин) графа из переданной матрицы
    смежности matrix и сохраняем его в переменной size

    int size = matrix.GetLength(0);

    //Создаем новый двумерный массив minimumSpanningTree, который будет
    хранить минимальное остовное дерево. Размерность массива такая же, как у
    переданной матрицы matrix

    int[,] minimumSpanningTree = new int[size, size];

    //Создаем новый одномерный массив visited, который будет отслеживать,
    посещена ли каждая вершина. Изначально все элементы массива установлены в false.

    bool[] visited = new bool[size];

    //Создаем переменную numOfVisited, которая будет отслеживать
    количество посещенных вершин. Изначально установлено значение 1, так как
    начальная вершина уже посещена.

    int numOfVisited = 1;

    //Помечаем начальную вершину, соответствующую startVertex, как
    посещенную.

    visited[startVertex - 1] = true;

    //Входим в цикл while, который будет выполняться до тех пор, пока
    количество посещенных вершин меньше размерности графа.

    while (numOfVisited < size)
    {
        //Создаем переменную minWeight и устанавливаем ее значение на
        максимально возможное значение типа int. В этой переменной будет храниться
        минимальный вес ребра при каждой итерации цикла.

        int minWeight = int.MaxValue;

        //Создаем переменные minFrom и minTo, которые будут хранить номера
        вершин, соединенных минимальным ребром. Изначально задаем им значение -1, чтобы
        отслеживать, есть ли вершины, соединенные минимальным ребром.

        int minFrom = -1;
        int minTo = -1;

        //Входим в цикл for, который перебирает все вершины графа.

        for (int i = 0; i < size; i++)
        {

```

```

        //Проверяем, была ли вершина i уже посещена. Если да, выполняем
код внутри этого условия.
        if (visited[i])
        {
            //Входим во внутренний цикл for, который перебирает все
вершины графа для поиска ребра с минимальным весом.
            for (int j = 0; j < size; j++)
            {
                //Проверяем, является ли вершина j непосещенной,
есть ли ребро между вершинами i и j, и является ли его вес меньше текущего
минимального веса minWeight.
                if (!visited[j] && matrix[i, j] > 0 && matrix[i,
j] < minWeight)
                {

                    //Если условие выполнено, обновляем значение
minWeight на вес найденного ребра.
                    minWeight = matrix[i, j];

                    //Обновляем значения minFrom и minTo на соответствующие
номера вершин, найденных ребром с минимальным весом.
                    minFrom = i + 1;
                    minTo = j + 1;
                }
            }
        }

        //Если найдено ребро с минимальным весом(т.е. "minFrom" и "minTo"
не равны -1), то оно добавляется в минимальное остовное дерево
"minimumSpanningTree",

        //обновляются массив "visited" и число посещенных вершин
увеличивается на 1.
        if (minFrom != -1 && minTo != -1)
        {
            minimumSpanningTree[minFrom - 1, minTo - 1] = minWeight;
            minimumSpanningTree[minTo - 1, minFrom - 1] = minWeight;
            visited[minTo - 1] = true;
            numOfVisited++;
        }

        //Если не найдено ребро с минимальным весом (т.е. все вершины
уже посещены), то цикл прерывается.
        else
        {

```

```

        break;
    }
}

//В конце функция возвращает минимальное остовное дерево
"minimumSpanningTree".

return minimumSpanningTree;

}

//Метод реализации алгоритма обхода графа в глубину с подсчетом общего
расстояния.

static int CalculateTotalDistance(int[,] matrix, int startVertex,
StreamWriter writer)
{
    //Получает размерность массива matrix по первому измерению и сохраняет
    в переменной size.

    int size = matrix.GetLength(0);

    //Создает массив булевых значений visited длиной size, который будет
    использоваться для отслеживания посещенных вершин.

    bool[] visited = new bool[size];

    //Инициализирует переменную totalDistance нулевым значением, которая
    будет содержать общее расстояние между вершинами.

    int totalDistance = 0;

    //Вызывает рекурсивную функцию CalculateDistanceRecursive для
    вычисления расстояния между вершинами, начиная с startVertex.

    //По умолчанию startVertex индексируется с 1, поэтому происходит
    смещение на 1 при вызове рекурсивной функции.

    CalculateDistanceRecursive(matrix, visited, startVertex - 1, 0, ref
    totalDistance, writer);

    //Возвращает общее расстояние между вершинами.

    return totalDistance;

}

//Это рекурсивная функция, которая вычисляет расстояние между вершинами
в графе. Принимает двумерный массив matrix, массив булевых значений visited,
//текущую вершину currentVertex, уровень отступа indentLevel, переменную
totalDistance по ссылке и объект StreamWriter writer.

static void CalculateDistanceRecursive(int[,] matrix, bool[] visited,
int currentVertex, int indentLevel, ref int totalDistance, StreamWriter writer)
{

```

```

        //Устанавливает текущую вершину currentVertex в массиве visited в
        значение true, чтобы отметить ее как посещенную.
        visited[currentVertex] = true;

        //Запускает цикл, проходящий по всем столбцам в строке currentVertex
        матрицы matrix.
        for (int i = 0; i < matrix.GetLength(0); i++)
        {
            //Проверяет, существует ли ребро между текущей вершиной
            currentVertex и вершиной i, а также не была ли вершина i уже посещена.
            if (matrix[currentVertex, i] != 0 && !visited[i])
            {
                //Увеличивает значение переменной totalDistance на значение
                ребра между currentVertex и i.
                totalDistance += matrix[currentVertex, i];

                //Записывает строку в файл writer, содержащую информацию
                о ребре между вершинами currentVertex + 1 и i + 1, включая размер ребра.
                writer.WriteLine(new string(' ', indentLevel * 4) +
                $"{currentVertex + 1} (size {matrix[currentVertex, i]}) -> {i + 1}");
                Console.WriteLine(new string(' ', indentLevel * 4) +
                $"{currentVertex + 1} (size {matrix[currentVertex, i]}) -> {i + 1}");

                //Рекурсивно вызывает функцию CalculateDistanceRecursive
                для вершины i, чтобы продолжить вычисление расстояния между вершинами.
                CalculateDistanceRecursive(matrix, visited, i, indentLevel
                + 1, ref totalDistance, writer);
            }
        }
    }

    //Метод GenerateAdjacencyMatrix генерирует матрицу смежности для
    заданного размера графа.
    private static int[,] GenerateAdjacencyMatrix(int size)
    {
        Random r = new Random();

        //создает двумерный массив (матрицу) размерности size x size для
        хранения смежности вершин.
        int[,] matrix = new int[size, size];

        //Внутри циклов происходит заполнение этой матрицы. Первый цикл
        отвечает за итерацию по столбцам матрицы, а второй цикл - по строкам.
        for (int i = 0; i < size; i++)

```



```

        {
            for (int j = i + 1; j < size; j++)
            {
                //Каждый элемент [i, j] получает случайное число 0 или 1,
                а элемент [j, i] получает то же значение для обеспечения симметричности графа.
                matrix[i, j] = r.Next(0, 2);
                matrix[j, i] = matrix[i, j];
            }
        }
        return matrix;
    }

    //Метод ConvertToWeightedMatrix принимает матрицу смежности и преобразует
    ее во взвешенную матрицу.
    private static int[,] ConvertToWeightedMatrix(int[,] adjacencyMatrix)
    {
        int size = adjacencyMatrix.GetLength(0);

        //Создается новая матрица weightedMatrix с размерами такими же, как
        у матрицы смежности.
        int[,] weightedMatrix = new int[size, size];
        Random r = new Random();

        //Затем используется цикл for для проверки каждого элемента матрицы
        смежности.
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                //Если значение элемента [i, j] равно 1 (т.е. есть связь
                между вершинами i и j),
                //то элемент[i, j] взвешивается(получает случайное значение
                от 1 до 10),
                if (adjacencyMatrix[i, j] == 1)
                {
                    weightedMatrix[i, j] = r.Next(1, 11);

                    //а элемент [j, i] принимает то же самое значение, чтобы
                    обеспечить симметричность графа.
                    weightedMatrix[j, i] = weightedMatrix[i, j];
                }
            }
        }
    }

```

```
        }  
        return weightedMatrix;  
    }  
}  
}
```