

Министерство науки и высшего образования Российской Федерации

Пензенский Государственный университет

Кафедра "Вычислительная техника"

Пояснительная записка

К курсовому проектированию
По курсу «Логика и основы алгоритмизации в
инженерных задачах »
На тему: «Реализация алгоритма Прима»

Выполнил студент группы 22BVB3:

Кузьмин Д.В.

Принял:

Юрова О.В.

Акифьев И.В.

Хорошо
09.01.24

Пенза 2023

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет Вычислительной техники

Кафедра "Вычислительная техника"

"УТВЕРЖДАЮ"

Зав. кафедрой ВТ

«__»__20__

ЗАДАНИЕ

на курсовое проектирование по курсу

Тема и основы алгоритмизации вычислительных задач
Студенту Кузьмину Ванилу Владимировичу Группа 22 ВВВЗ (22ВВВЗ2)
Тема проекта Реализация алгоритма Гутмана

Исходные данные (технические требования) на проектирование

- Разработка алгоритмов и программного обеспечения в соответствии с данными заданием курсового проекта. Пояснительная записка должна содержать:
1. Постановку задачи;
 2. Теоретическую часть задачи;
 3. Описание алгоритма поставленной задачи;
 4. Пример ручного расчета задачи и вычислений (на небольшом фрагменте работы алгоритма);
 5. Описание самой программы;
 6. Тесты;
 7. Список литературы;
 8. Источники программного;
 9. Результаты работы программы;

Объем работы по курсу

1. Расчетная часть

Ручной расчет работы алгоритма

2. Графическая часть

схема алгоритма в формате блок-схем.

3. Экспериментальная часть

Тестирование программы;
Результаты работы программы на тестовых данных

Срок выполнения проекта по разделам

- 1 Исследование теоретической части курсового
- 2 Разработка алгоритмов программы
- 3 Разработка программы
- 4 Тестирование и завершение разработки программы
- 5 Оформление пояснительной записки
- 6
- 7
- 8

Дата выдачи задания " 6 " сентября

Дата защиты проекта " " "

Руководитель Андреев Игорь Владимирович

Задание получил " 25 " сентября

2023 г.

Студент Кузьмук Данила Владиславович

ОГЛАВЛЕНИЕ

Реферат	5
Введение	6
1.Постановка задачи.....	7
2.Теоретическая часть	8
3.Описание алгоритма программы.....	9
4.Описание программы.....	11
5.Тестирование	15
6.Ручной расчёт программы	18
Заключение	19
Список литературы	20
7.Приложение А.....	21
Листинг программы.....	21

Реферат

Отчет 23 стр., 12 рисунков, 1 таблица, 1 приложение.

ГРАФ, ТЕОРИЯ ГРАФОВ, АЛГОРИТМ ПРИМА

Цель исследования – реализация алгоритма Прима для нахождения минимального остовного дерева.

В работе рассмотрены основные шаги алгоритма Прима для построения минимального остовного дерева взвешенного графа.

Установлено, что с помощью данного алгоритма можно выделить минимальное остовное дерево независимо от степени связности и весов рёбер исходного графа.

Введение

Алгоритм Прима - это алгоритм минимального остовного дерева, что принимает граф в качестве входных данных и находит подмножество ребер этого графа, который формирует дерево, включающее в себя каждую вершину, а также имеет минимальную сумму весов среди всех деревьев, которые могут быть сформированы из графа.

Алгоритм Прима строит минимальное остовное дерево, добавляя на каждом шаге к строящемуся остову безопасное ребро минимальной длины.

Ребро называется безопасным, если при добавлении его к строящемуся остову не нарушается свойство ацикличности.

В качестве среды разработки мною была выбрана Microsoft Visual Studio 2019, язык программирования – Си.

Целью данной курсовой работы является разработка программы на языке Си, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм Прима, осуществляющий нахождение минимального остовного дерева.

1. Постановка задачи

Требуется разработать программу, которая выделит минимальное остовное дерево для неориентированного взвешенного графа, используя алгоритм Прима.

Исходный граф в программе должен задаваться матрицей весов рёбер, причем при генерации данных должны быть предусмотрены граничные условия. Программа должна работать так, чтобы пользователь вводил количество вершин для генерации матрицы взвешенного графа. После обработки этих данных на экран должна выводиться матрица взвешенного графа и минимальное остовное дерево, полученное алгоритмом Прима. Необходимо предусмотреть различные исходы, такие как случаи отсутствия связности в графе или некорректного ввода данных, чтобы программа не выдавала ошибок и работала правильно. Устройство ввода – клавиатура и мышь.

2. Теоретическая часть задания

Граф G (рисунок 1) задается множеством вершин X_1, X_2, \dots, X_n и множеством ребер, соединяющих между собой определенные вершины. Ребра в этом графе имеют веса, показывающие стоимость прохождения между вершинами. Граф с такими ребрами называется взвешенным.

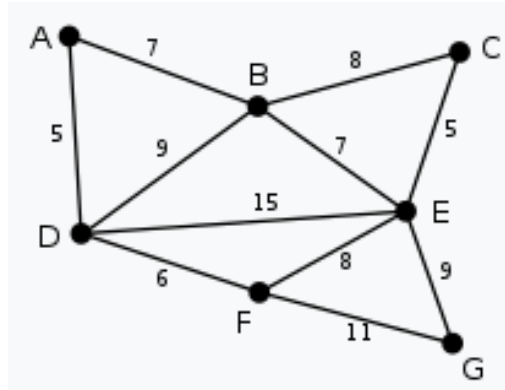


Рисунок 1 – Пример взвешенного графа

При представлении графа матрицей смежности информация о ребрах графа хранится в квадратной матрице, где пути из одной вершины в другую и обратно обозначаются единицей, иначе нулем.

На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. 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 наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимальной стоимости.

3. Описание алгоритма программы

Цель алгоритма Прима в графе заключается в построении минимального остовного дерева путем пошагового добавления ребер с наименьшим весом. Начиная с начальной вершины, мы постепенно расширяем остовное дерево, выбирая ребро минимального веса, которое связывает уже выбранные вершины с вершиной из оставшихся. Этот процесс продолжается, пока все вершины не будут включены в остовное дерево или пока не будет достигнут критерий останова. Ниже представлен псевдокод:

функция Алгоритм Прима(граф[`MAX_SIZE`][`MAX_SIZE`], размер,
родитель[`MAX_SIZE`])

минВес[`MAX_SIZE`]

посещено[`MAX_SIZE`]

для $i = 0$ пока $i < \text{размер}$

минВес[i] = `INT_MAX`

посещено[i] = `false`

родитель[i] = -1

минВес[0] = 0

для count = 0 пока count < размер - 1

минВершина = -1

для $v = 0$ пока $v < \text{размер}$

если не посещено[v] и (минВершина == -1 или

минВес[v] < минВес[минВершина])

минВершина = v

посещено[минВершина] = `true`

для $v = 0$ пока $v < \text{размер}$

если граф[минВершина][v] != 0 и не посещено[v] и

```
граф[минВершина][v] < минВес[v]
    родитель[v] = минВершина
    минВес[v] = граф[минВершина][v]

вывод("\nМинимальное остовное дерево:\n")
для i = 1 пока i < размер
    если родитель[i] != -1
        вывод("Ребро: " + родитель[i] + " - " + i + ", Вес:
" + граф[i][родитель[i]])
    иначе
        вывод("Ребро: " + findUnconnectedNode(родитель,
размер) + " - " + i + ", не существует")
```

4. Описание программы

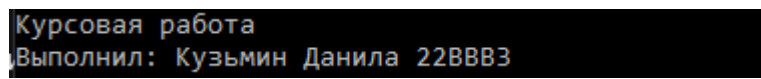
Для написания данной программы использован язык программирования Си. Язык программирования Си – универсальный язык программирования, который завоевал особую популярность у программистов, благодаря сочетанию возможностей языков программирования высокого и низкого уровней.

Проект был создан в виде консольного приложения Win32(VisualC++).

Данная программа многомодульная, поскольку состоит из нескольких функций: saveGraphToFile, savePrimResultToFile, fileExists. loadGraphFromFile, fillUndirectedGraphRandom, printGraph, findUnconnectedNode, primAlgorithm, main(void), struct Node, void print_tree.

Работа программы начинается с вывода общей информации о курсовой работе(рис.2).

```
printf("Курсовая работа\n");  
printf("Выполнил: Кузьмин Данила 22ВВВ3\n");
```

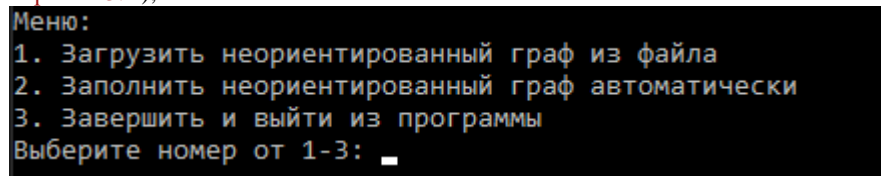


```
Курсовая работа  
Выполнил: Кузьмин Данила 22ВВВ3
```

Рисунок 2 – Меню.

Далее выводятся основные действия программы, которые пользователь может выбрать (рис. 3).

```
printf("Меню:\n");  
printf("1. Загрузить неориентированный граф из файла\n");  
printf("2. Заполнить неориентированный граф автоматически\n");  
printf("3. Завершить и выйти из программы\n");  
printf("Выберите номер от 1-3: ");
```



```
Меню:  
1. Загрузить неориентированный граф из файла  
2. Заполнить неориентированный граф автоматически  
3. Завершить и выйти из программы  
Выберите номер от 1-3: _
```

Рисунок 3 – Основные действия

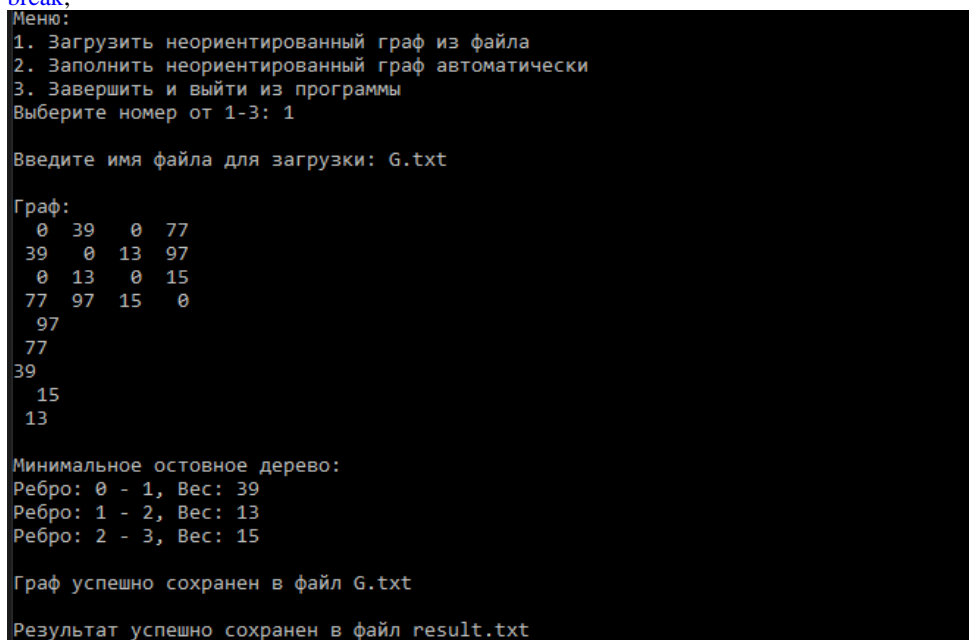
Если пользователь выбрал первый пункт (рис. 4), то сначала программа просит ввести название файла. Затем она загружает этот файл (вызывая функцию loadGraphFromFile), выводит граф на экран (вызывая функцию printGraph), выполняет алгоритм Прима и выводит минимальное остовное дерево на экран (вызывая функцию primAlgorithm), сохраняет граф и

результат в файлы (вызывая функции saveGraphToFile и savePrimResultToFile).

case 1:

```
printf("\nВведите имя файла для загрузки: ");
scanf("%s", filename);
if (!fileExists(filename)) {
    printf("\nВведён несуществующий файл\n\n");
    break;
}
loadGraphFromFile(graph, &size, filename);
printGraph(graph, size);
///
for (int i = 0; i < size; i++) {
    for (int j = i; j < size; j++) {
        D = graph[i][j];
        if (D == 0) continue;
        root = CreateTree(root, D);
    }
}

print_tree(root, 0);
root = NULL;
D = 0;
///
primAlgorithm(graph, size, parent);
saveGraphToFile(graph, size, "G.txt");
savePrimResultToFile(parent, graph, size, "result.txt");
break;
```



```
Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3: 1

Введите имя файла для загрузки: G.txt

Граф:
  0  39  0  77
 39  0  13  97
  0  13  0  15
 77  97  15  0
 97
 77
 39
 15
 13

Минимальное остовное дерево:
Ребро: 0 - 1, Вес: 39
Ребро: 1 - 2, Вес: 13
Ребро: 2 - 3, Вес: 15

Граф успешно сохранен в файл G.txt
Результат успешно сохранен в файл result.txt
```

Рисунок 4 – Загрузка графа из файла

Если пользователь выбрал второй пункт (рис. 5), то сначала программа просит ввести размер графа. Затем она заполняет этот граф случайными числами (вызывая функцию fillUndirectedGraphRandom), если вводите число. Далее выводит граф на экран (вызывая функцию printGraph), если вы вводите

буквы, то будет выдавать ошибку. После выполняет алгоритм Прима и выводит минимальное остовное дерево на экран (вызывая функцию `primAlgorithm`), сохраняет граф и результат в файлы (вызывая функции `saveGraphToFile` и `savePrimResultToFile`).

case 2:

```
printf("\nВведите размер неориентированного графа: ");

while (1) {
    cin >> buffer;

    if (atoi(buffer) == 0 && buffer[0] != '0') {

        cout << "Введите только целое число" << endl;
        continue;
    }
    else {
        size = atoi(buffer);
        break;
    }

    // scanf("%d", &size);
}

if (size <= 0) {
    printf("\nВведён неверный размер графа\n\n");
    break;
}

fillUndirectedGraphRandom(graph, size);
printGraph(graph, size);

//
for (int i = 0; i < size; i++) {
    for (int j = i; j < size; j++) {
        D = graph[i][j];
        if (D == 0) continue;
        root = CreateTree(root, D);
    }
}

print_tree(root, 0);
root = NULL;
D = 0;

//
primAlgorithm(graph, size, parent);
saveGraphToFile(graph, size, "G.txt");
savePrimResultToFile(parent, graph, size, "result.txt");
break;
```



```

Курсовая работа
Выполнил: Кузьмин Данила 228ВВВ3
Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3: 2

Введите размер неориентированного графа: 4

Граф:
  0  39  0  77
 39  0  13  97
  0  13  0  15
 77  97  15  0
  97
 77
39
 15
 13

Минимальное остовное дерево:
Ребро: 0 - 1, Вес: 39
Ребро: 1 - 2, Вес: 13
Ребро: 2 - 3, Вес: 15

Граф успешно сохранен в файл G.txt
Результат успешно сохранен в файл result.txt

```

Рисунок 5 – Заполнение графа автоматически

Если пользователь выбрал пункт «3», то программа завершит свою работу (рис. 6).

```

Курсовая работа
Выполнил: Кузьмин Данила 228ВВВ3
Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3: 3

E:\C++\cursovaya\x64\Debug\cursovaya.exe (процесс 16164) завершил работу с кодом 3.
Нажмите любую клавишу, чтобы закрыть это окно: _

```

Рисунок 6 – Проверка на завершение программы

5.Тестирование

Среда разработки Microsoft Visual Studio 2019 представляет все средства, необходимые при разработке и отладки многомодульных программ.

Тестирование проводилось в рабочем порядке , в процессе разработки , после завершения написания программы. В ходе тестирования было выявлено и исправлено множество ошибок и проблем , связанных с вводом данных, изменением дизайна вводимых данных, алгоритмом программы, взаимодействием функций.

Ниже продемонстрирован результат тестирования программы при вводе пользователем различных количеств вершин.

```
Курсовая работа
Выполнил: Кузьмин Данила 22BVB3
Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3: _
```

Рисунок 7 – Проверка программы на запуск

```
Курсовая работа
Выполнил: Кузьмин Данила 22BVB3
Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3: 4

Сделан неверный выбор. Пожалуйста, выберите снова.

Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3: _
```

Рисунок 8–Проверка на неправильный выбор номер

```
Курсовая работа
Выполнил: Кузьмин Данила 228883
Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3: 1

Введите имя файла для загрузки: dada

Введён несуществующий файл

Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3:
```

Рисунок 9–Проверка на несуществующий файл

```
Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3: 2

Введите размер неориентированного графа: -3

Введён неверный размер графа

Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3: █
```

Рисунок 10–Проверка на введение правильного размера графа

```
Меню:
1. Загрузить неориентированный граф из файла
2. Заполнить неориентированный граф автоматически
3. Завершить и выйти из программы
Выберите номер от 1-3: 2

Введите размер неориентированного графа: караван
Введите только целое число
```

Рисунок 11–Проверка на введение только целых чисел

Описание теста	Ожидаемый результат	Полученный результат
Запуск программы	Вывод общей информации и основных действий	Верно
Проверка на неправильный выбор основного действия	Программа должна оповестить пользователя об ошибке, предложить снова варианты	Верно
Проверка на введение несуществующего файла	Программа должна оповестить пользователя об ошибке, предложить снова варианты	Верно
Проверка на введение неправильного размера графа	Программа должна оповестить пользователя об ошибке, предложить снова варианты	Верно
Вывод результата	Вывод правильного результата на разно-размерных графах, идентичность с ручным расчетом	Верно
Правильность работы алгоритма	Совпадение ручных расчетов с результатом работы алгоритма	Верно
Проверка на наличие изолированных вершин	Должна выполняться проверка на наличие изолированных вершин	Верно

Таблица 1 – Описание поведения программы при тестировании

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

6. Ручной расчёт программы

Граф:				
0	30	32	97	61
30	0	66	89	53
32	66	0	0	20
97	89	0	0	97
61	53	20	97	0

Минимальное остовное дерево:				
Ребро:	0 - 1,	Вес:	30	
Ребро:	0 - 2,	Вес:	32	
Ребро:	1 - 3,	Вес:	89	
Ребро:	2 - 4,	Вес:	20	

Рисунок 12 – Граф и результат

Начинаем с вершины 0. Выбираем ребро с минимальным весом: ребро 0 - 1 с весом 30. Далее ищем следующие рёбра с минимальным весом по такому же принципу. Из вершины 0 можно пройти в вершину 2 и получить следующее ребро: ребро 0 - 2 с весом 32. Из вершины 2 можно перейти в вершину 4, получаем ребро: ребро 2 - 4 с весом 20. Из вершины 1 можно перейти в вершину 3, получаем ребро: ребро 1 - 3 с весом 89. В итоге мы получаем минимальное остовное дерево: 3 - 1 - 0 - 2 - 4. Таким образом, результат ручных расчетов совпадает с результатом работы алгоритма, таким образом можно сделать вывод, что программа работает верно.

Заключение

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм Прима в Microsoft Visual Studio 2019.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания матриц смежности, а также работы с новыми алгоритмами. Углублены навыки знания языка программирования Си.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

Список литературы

1. Уилсон Р. Введение в теорию графов. Пер. с англ. 1977. 208с.
2. Герберт Шилдт «полный справочник по С++» - вильямс, 2006
3. Оре О. Графы и их применение: Пер. с англ. 1965. 176с.
4. Брайан Керниган, Деннис Ритчи «Язык программирования Си»

7. Приложение А

Листинг программы

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <stdbool.h>
#include <locale.h>
#include <time.h>
#include <vector>
#include <fstream>
// #include <sstream>

using namespace std;

#define MAX_SIZE 100

int findUnconnectedNode(int parent[MAX_SIZE], int size) {
    for (int i = 0; i < size; ++i) {
        if (parent[i] == -1) {
            return i;
        }
    }
    return -1; // Если не найдено несвязанных вершин
}

void printTreeStructure(ofstream& test, const vector<vector<int>>& children, int vertex, int depth = 0) {
    for (int i = 0; i < depth; ++i) {
        cout << " ";
        test << " ";
    }

    cout << vertex << "\n";
    test << vertex << "\n";

    // рекурсивно выводятся все смежные вершины для текущей вершины
    for (int child : children[vertex]) {
        printTreeStructure(test, children, child, depth + 1);
    }
}

// создание непосредственной структуры
void buildTreeStructure(const int parent[], int size, vector<vector<int>>& children) {
    for (int i = 0; i < size; ++i) {
        if (parent[i] != -1) {
            children[parent[i]].push_back(i);
        }
    }
}

// Вывод остовного дерева
void printPrimMST(ofstream& test, int graph[MAX_SIZE][MAX_SIZE], int size, int parent[MAX_SIZE]) {
    vector<vector<int>> children(size);

    buildTreeStructure(parent, size, children);

    int root = findUnconnectedNode(parent, size);

    if (root != -1) {
        cout << "Визуализация остовного дерева:\n";
        printTreeStructure(test, children, root);
    } else {
```

```

        cout << "Граф не содержит ни одного ребра.\n";
    }
}

// Функция сохранения графа в файл
void saveGraphToFile(int graph[MAX_SIZE][MAX_SIZE], int size, const char* filename) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("Ошибка при открытии файла\n");
        return;
    }
    fprintf(file, "%d\n", size);
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            fprintf(file, "%d ", graph[i][j]);
        }
        fprintf(file, "\n");
    }
    fclose(file);
    printf("\nГраф успешно сохранен в файл G.txt\n\n");
}

// Функция сохранения результата алгоритма Прима в файл
void savePrimResultToFile(int parent[MAX_SIZE], int graph[MAX_SIZE][MAX_SIZE], int size, ofstream& test)
{
    //FILE* file = fopen(filename, "w");
    if (!test.is_open()) {
        printf("Ошибка при открытии файла для сохранения результата\n");
        return;
    }
    test << "Минимальное остовное дерево:" << endl;
    //fprintf(file, "Минимальное остовное дерево:\n");
    for (int i = 1; i < size; ++i) {
        //fprintf(file, "Ребро: %d - %d, Вес: %d\n", parent[i], i, graph[i][parent[i]]);
        test << "Ребро: " << parent[i] << " - " << i << ", Вес: " << graph[i][parent[i]] << endl;
    }
    //fclose(file);
    printf("Результат успешно сохранен в файл result.txt\n\n");
}

// Функция для проверки существования файла
bool fileExists(const char* filename) {
    FILE* file = fopen(filename, "r");
    if (file != NULL) {
        fclose(file);
        return true; // Файл существует
    }
    return false; // Файл не существует
}

// Загрузка графа из файла
void loadGraphFromFile(int graph[MAX_SIZE][MAX_SIZE], int* size, const char* filename) {
    FILE * file = fopen(filename, "r");
    if (file == NULL) {
        printf("\nОшибка при открытии файла\n");
        return;
    }
    fscanf(file, "%d", size);
    for (int i = 0; i < *size; ++i) {
        for (int j = 0; j < *size; ++j) {
            fscanf(file, "%d", &graph[i][j]);
        }
    }
    fclose(file);
}

// Функция для заполнения неориентированного графа случайными числами
void fillUndirectedGraphRandom(int graph[MAX_SIZE][MAX_SIZE], int size)

```

```

{
    srand(time(NULL));
    // Заполнение графа нулями и единицами на главной диагонали
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            if (i == j) {
                graph[i][j] = 0; // Главная диагональ остается нулевой
            }
            else {
                graph[i][j] = rand() % 2; // Заполнение ребер 0 или 1
                graph[j][i] = graph[i][j]; // Отражение изменений для неориентированного графа
            }
        }
    }
    // Замена единиц случайными числами
    for (int i = 0; i < size; ++i) {
        for (int j = 0; j < size; ++j) {
            if (i != j && graph[i][j] == 1) {
                graph[i][j] = rand() % 100; // Заполнение случайным числом вместо единицы
                graph[j][i] = graph[i][j]; // Отражение изменений для неориентированного графа
            }
        }
    }
}
// Вывод графа в консоль
void printGraph(int graph[MAX_SIZE][MAX_SIZE], int size)
{
    printf("\nГраф:\n");
    for (int i = 0; i < size; ++i)
    {
        for (int j = 0; j < size; ++j)
        {
            printf("%3d ", graph[i][j]);
        }
        printf("\n");
    }
}
// Функция для нахождения несвязных вершин

// Функция выполнения алгоритма Прима для графа
void primAlgorithm(ofstream& test, int graph[MAX_SIZE][MAX_SIZE], int size, int parent[MAX_SIZE]) {
    int minEdge[MAX_SIZE]; // Минимальные веса рёбер
    bool visited[MAX_SIZE]; // Посещали ли вершины
    // Инициализация значений
    for (int i = 0; i < size; ++i) {
        minEdge[i] = INT_MAX; // Установка начальных значений весов рёбер как максимальное
        значение
        visited[i] = false; // Начально ни одна вершина не посещена
        parent[i] = -1; // Начально нет родительской вершины
    }
    minEdge[0] = 0; // Стартовая вершина, вес ребра равен 0
    for (int count = 0; count < size - 1; ++count) {
        int minVertex = -1; // Минимальный индекс вершины
        for (int v = 0; v < size; ++v) {
            if (!visited[v] && (minVertex == -1 || minEdge[v] < minEdge[minVertex])) {
                minVertex = v; // Нахождение вершины с минимальным весом ребра
            }
        }
        visited[minVertex] = true; // Включаем найденную вершину в остоное дерево
        // Обновляем веса смежных вершин
        for (int v = 0; v < size; ++v) {
            if (graph[minVertex][v] != 0 && !visited[v] && graph[minVertex][v] <
                minEdge[v]) {
                parent[v] = minVertex;
                minEdge[v] = graph[minVertex][v];
            }
        }
    }
}

```



```

    }
}
// Вывод ребер минимального остовного дерева
printf("\nМинимальное остовное дерево:\n");

int D = 0;
struct Node* root = NULL;
for (int i = 1; i < size; ++i) {
    if (parent[i] != -1) {
        printf("Ребро: %d - %d, Вес: %d\n", parent[i], i, graph[i][parent[i]]);
    }
    else {
        printf("Ребро: %d - %d, не существует\n", findUnconnectedNode(parent, size), i);
    }
}

printPrimMST(test, graph, size, parent);
}

int main(void) {
    setlocale(LC_ALL, "RUS");
    int graph[MAX_SIZE][MAX_SIZE];
    int size, choice;
    char filename[50];
    int parent[MAX_SIZE];
    bool b = { };

    ///
    char buffer[20];
    ofstream ttt;
    ///

    printf("Курсовая работа\n");
    printf("Выполнил: Кузьмин Данила 22BBB3\n");

    do {
        printf("Меню:\n");
        printf("1. Загрузить неориентированный граф из файла\n");
        printf("2. Заполнить неориентированный граф автоматически\n");
        printf("3. Завершить и выйти из программы\n");
        printf("Выберите номер от 1-3: ");

        ///
        while (1) {
            cin >> buffer;

            if (atoi(buffer) == 0 && buffer[0] != '0') {

                cout << "Введите только целое число" << endl;
                continue;
            }

            if (buffer[1] != 0) {
                cout << "only integers" << endl;
                continue;
            }

            else {
                choice = atoi(buffer);
                break;
            }
        };

        ///
        switch (choice) {
            case 1:
                printf("\nВведите имя файла для загрузки: ");
                scanf("%s", filename);
                if (!fileExists(filename)) {
                    printf("\nВведён несуществующий файл\n\n");
                    break;
                }
            }
        }
    }
}

```

```

    }
    loadGraphFromFile(graph, &size, filename);
    printGraph(graph, size);
    ttt.open("result.txt");
    primAlgorithm(ttt, graph, size, parent);
    saveGraphToFile(graph, size, "G.txt");
    savePrimResultToFile(parent, graph, size, ttt);
    ttt.close();
    break;
case 2:
    printf("\nВведите размер неориентированного графа: ");

    while (1) {
        cin >> buffer;

        if (atoi(buffer) == 0 && buffer[0] != '0') {

            cout << "Введите только целое число" << endl;
            continue;
        }
        else {
            size = atoi(buffer);
            break;
        }
    }

    if (size <= 0) {
        printf("\nВведён неверный размер графа\n\n");
        break;
    }
    fillUndirectedGraphRandom(graph, size);
    printGraph(graph, size);
    ttt.open("result.txt");
    primAlgorithm(ttt, graph, size, parent);
    saveGraphToFile(graph, size, "G.txt");
    //cout << "jope" << endl;
    savePrimResultToFile(parent, graph, size, ttt);
    ttt.close();
    break;
case 3:
    return 3; // Завершение программы при выборе "3"
default:
    printf("\nСделан неверный выбор. Пожалуйста, выберите снова.\n\n");
    break;
    }
} while (true);
}

```