МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждения высшего образования
«МИРЭА - Российский технологический университет»
РТУ МИРЭА

Отчет по выполнению практической работы № 7.2

Тема:

«Графы: создание, алгоритмы обхода, важные задачи теории графов»

Дисциплина: «Структуры и алгоритмы обработки данных»

Выполнил студент: Туляшева А.Т.

Группа: ИКБО-42-23

Москва - 2024

Содержание

ЦЕЛЬ РАБОТЫ

Получение практических навыков по выполнению операций над структурой данных граф.

ЗАДАНИЕ 1

Ответьте на вопросы.

1. Определения понятий:

- Ориентированный граф: Граф, в котором каждому ребру приписано направление, то есть каждое ребро соединяет начальную и конечную вершину. Ребра такого графа называются дугами.
- Неориентированный граф: Граф, в котором ребра не имеют направления. Ребро соединяет две вершины, и их порядок не имеет значения.
- Взвешенный граф: Граф, в котором каждому ребру (или вершине) приписан вес — числовое значение, которое может представлять, например, стоимость, длину, или время.
- Связный граф: Граф, в котором существует путь между любыми двумя вершинами. Для ориентированного графа связность подразумевает, что можно пройти из любой вершины в любую другую с учетом направлений ребер.
- Центр графа: Множество вершин графа, у которых наименьшее значение эксцентриситета. Эксцентриситет вершины — это максимальное расстояние (длина кратчайшего пути) от этой вершины до любой другой.
- Диаметр графа: Наибольшее значение расстояния (длины кратчайшего пути) между любыми двумя вершинами графа. Другими словами, это максимум всех эксцентриситетов в графе.
- Матрица смежности: Квадратная матрица AAA, где элемент A[i][j]A[i][j]A[i][j] обозначает наличие (и, возможно, вес) ребра между вершинами iii и jjj.
  - Для неориентированного графа матрица симметрична.
  - Для ориентированного графа симметрия не обязательна.

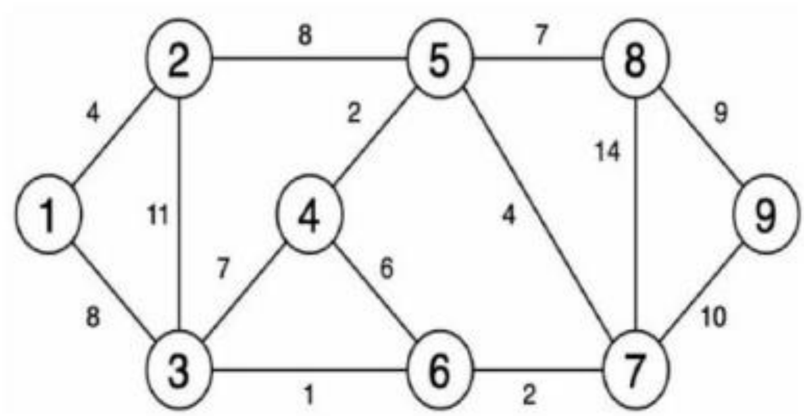
2. Остовное дерево графа:

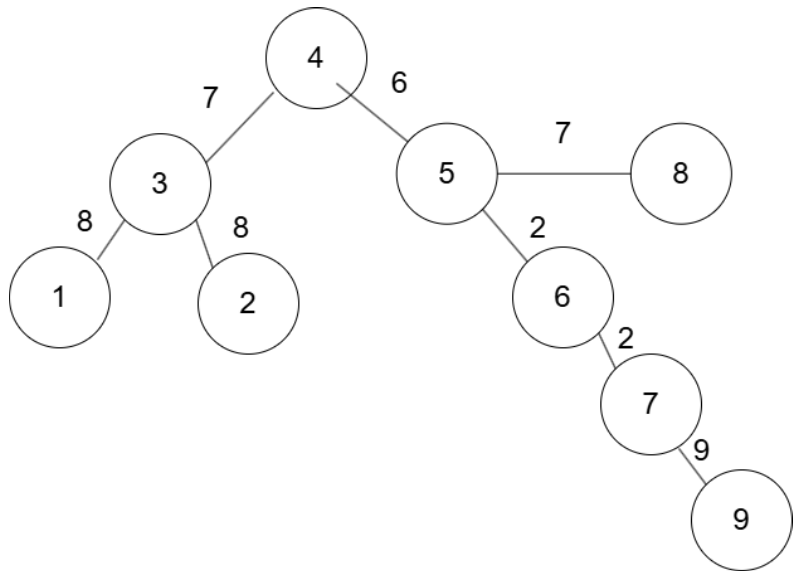
Остовное дерево (минимальное остовное дерево) — это подграф связного графа, который включает все вершины исходного графа, является деревом (не содержит циклов) и соединяет все вершины с минимальным числом ребер.

3. Количество ребер в остовном дереве:

Если граф GGG имеет nnn вершин, то остовное дерево графа будет содержать ровно n−1n - 1n−1 ребро. Это свойство справедливо для любого связного графа, так как добавление еще одного ребра создаст цикл, а удаление ребра нарушит связность.

4. Постройте остовное дерево, используя алгоритм Прима. Стартовая вершина – 4.



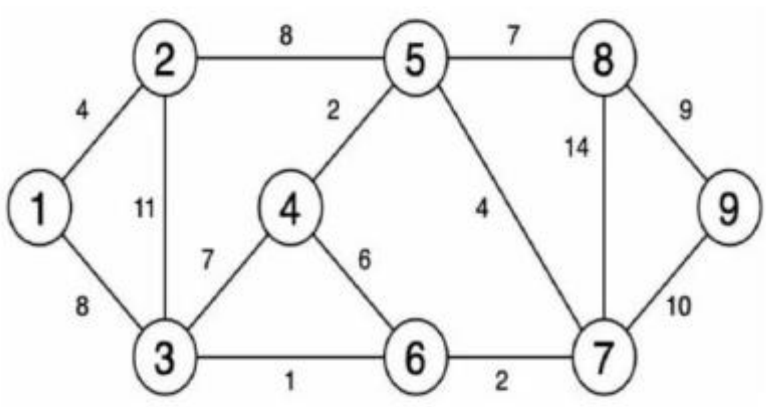


Вес остовного дерева: 6+2+2+7+7+9+8+8=49

5. Что такое кратчайший путь в графе?

Кратчайший путь в графе — это путь между двумя вершинами, сумма весов рёбер которого минимальна. Если граф не взвешенный, кратчайший путь измеряется количеством рёбер.

6. Найдите кратчайший путь от вершины 1 до вершины 9, используя алгоритм Дейкстры.



Алгоритм Дейкстры:

1. Инициализация:
   - Установить расстояние до стартовой вершины = 0, до остальных
   - ∞.
   - Создать множество *непосещённых* вершин.
2. Выбор вершины:
   - Выбрать непосещённую вершину с минимальным расстоянием.
3. Обновление соседей:
   - Для каждой соседней вершины обновить расстояние, если найден более короткий путь.
4. Пометить вершину как посещённую:
   - Добавить текущую вершину в список посещённых.
5. Повторить:
   - Повторять шаги 2–4, пока не обработаны все вершины или расстояния до целевых вершин не найдены.

6. Результат:
   - Минимальные расстояния от стартовой вершины до всех остальных.
7. Кратчайший путь:

   1 → 3 → 6 → 7 → 9.

   Длина пути: 21.

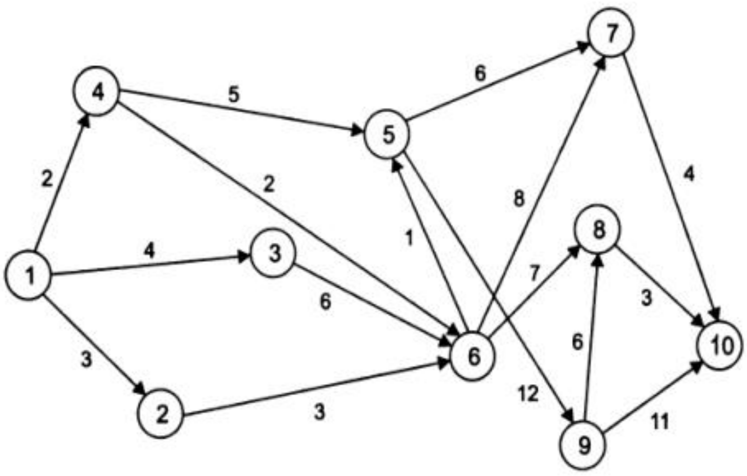
7. В чем отличие алгоритма Дейкстры от алгоритма Флойда-Уоршалла. Какова вычислительная сложность каждого алгоритма по времени и памяти.

Различия между алгоритмом Дейкстры и алгоритмом Флойда-Уоршалла:

Алгоритм Дейкстры используется для поиска кратчайшего пути от одной заданной вершины ко всем остальным в графе с неотрицательными весами. Вычислительная сложность — O(V^2) для реализации на основе матриц и O((V+E) logV) для реализаций на основе куч. Алгоритм Флойда-Уоршалла находит кратчайшие пути между всеми парами вершин и применяется ко всем взвешенным графам (включая отрицательные веса, если нет отрицательных циклов). Вычислительная сложность.

8. Обойдите граф, используя метод поиска а) в ширину; б) в глубину.

Стартовая вершина – 1.



а) Поиск в ширину (BFS)

Алгоритм поиска в ширину использует очередь и исследует узлы на текущем уровне перед переходом на следующий уровень.

Порядок обхода:

1 → 4 → 3 → 2 → 5 → 6 → 7 → 8 → 9 → 10

б) Поиск в глубину (DFS)

Поиск в глубину использует стек (или рекурсию), углубляясь по каждому пути, пока не достигнет конца.

Порядок обхода:

1 → 4 → 5 → 6 → 7 → 8 → 10 → 9 → 3 → 2

ЗАДАНИЕ 2

1. Разработать класс «Граф», обеспечивающий хранение и работу со структурой данных «граф», в соответствии с вариантом индивидуального задания: Реализовать метод ввода графа с клавиатуры, наполнение графа осуществлять с помощью метода добавления одного ребра; Реализовать методы, выполняющие задачи, определенные вариантом индивидуального задания; Разработать доступный способ (форму) вывода результирующего дерева на экран монитора.

2. Разработать программу, демонстрирующую работу всех методов класса.

3. Произвести тестирование программы на одном из графов, предложенных в таблице.

4. Составить отчет, отобразив в нем описание выполнения всех этапов разработки, тестирования и код всей программы со скриншотами результатов тестирования.

Вариант 1:

1	Матрица смежности	Определить центр графа. Составить программу реализации алгоритма Крускала построения остовного дерева минимального веса.

2.1 Математическая модель решения (описание алгоритма)

1. addEdge(int u, int v, int weight) - добавление одного ребра. Добавляет ребро между вершинами u и v с указанным весом weight в граф.

2. addEdgesBulk(int edges) – добавление нескольких ребер (для удобства). Вызывает функцию добавления одно ребра addEdge.

3. display() – вывод матрицы смежности. Два вложенных цикла перебирают строки и столбцы матрицы и выводят значения.

4. find_center() – нахождение центра графа. После того, как мы получаем массив расстояний для каждой вершины, мы находим максимальное расстояние от этой вершины до всех остальных. Затем мы сравниваем это расстояние с текущим минимальным максимальным расстоянием и обновляем его и индекс центра, если это необходимо.

5. find(int v, vector<int>& parent) – поиск корня вершины. Метод для нахождения представителя (корня) вершины v в структуре объединений. Если parent[v] не равен v, рекурсивно ищется корень и обновляется родительский узел, чтобы оптимизировать дальнейшие запросы.

6. unite(int v1, int v2, vector<int>& parent) – объединение двух вершин. Метод для объединения двух вершин v1 и v2 в одном поддереве. Находит корни для обеих вершин и устанавливает родителя v1 как родителя v2.

7. pair<vector<pair<int, int>>, int> kruskal() – реализация алгоритма Краскала. Сначала инициализируется вектор parent. Далее собираются все ребра с их весами в вектор edges. Ребра сортируются по возрастанию весов. В цикле добавляются ребра в MST, если они не образуют цикл (проверяется с помощью методов find и unite), обновляется вес MST. Метод возвращает пару: вектор с ребрами MST и их общий вес.

2.2 Код программы

```
#include ...
using namespace std;
const int INF = numeric_limits<int>::max();

class Graph {
private:
    int vertices; //кол-во вершин в графе
    vector<vector<int>> adjacency_matrix; //матрица смежности для представления графа

    int find(int v, vector<int>& parent) { //поиск корня вершины
        if (parent[v] != v) {
            parent[v] = find(parent[v], parent);
        } //рекурсивно ищется корень и обновляется родительский узел
        return parent[v];
    }

    void unite(int v1, int v2, vector<int>& parent) { //объединение двух вершин
        parent[find(v1, parent)] = find(v2, parent);
    } //находит корни для обеих вершин и устанавливает родителя v1 как родителя v2

public:
    Graph(int vertices) { //конструктор
        this->vertices = vertices; //устанавливаем кол-вол вершин
        adjacency_matrix.resize(vertices, vector<int>(vertices, 0)); //инициализируем матрицу нулями
    }

    void addEdge(int u, int v, int weight) { //добавление одного ребра
        if (u <= vertices && v <= vertices && u > 0 && v > 0) {
            adjacency_matrix[u - 1][v - 1] = weight; //индексация с 0
            adjacency_matrix[v - 1][u - 1] = weight; //для неориентированного графа
        }
        else {
            cout << "Некорректные вершины!\n";
        }
    }

    void addEdgesBulk(int edges) { //для массового ввода ребер
        cout << "Введите рёбра в формате 'начало конец вес':\n";
        for (int i = 0; i < edges; ++i) {
            int u, v, weight;
            cout << "Ребро " << i + 1 << ": ";
            cin >> u >> v >> weight;
            addEdge(u, v, weight);
        }
    }
```

Рисунок 1 – Код программы

```
void display() { //вывод
    for (int i = 0; i < vertices; i++) { //по строкам матрицы
        for (int j = 0; j < vertices; j++) { //по столбцам
            cout << adjacency_matrix[i][j] << " ";
        }
        cout << endl;
    }
}

int find_center() {
    for (int k = 0; k < vertices; k++) {
        for (int i = 0; i < vertices; i++) {
            for (int j = 0; j < vertices; j++) {
                if (adjacency_matrix[i][k] < INF && adjacency_matrix[k][j] < INF) {
                    adjacency_matrix[i][j] = min(adjacency_matrix[i][j],
                        adjacency_matrix[i][k] + adjacency_matrix[k][j]);
                }
            }
        }
    }
    vector<int> max_distances(vertices);
    for (int i = 0; i < vertices; i++) {
        max_distances[i] = *max_element(adjacency_matrix[i].begin(), adjacency_matrix[i].end());
    }
    int center_index = distance(max_distances.begin(), min_element(max_distances.begin(), max_distances.end()));
    return center_index;
}
```

Рисунок 2 – Код программы

```
pair<vector<pair<int, int>>, int> kruskal() { //алгоритм Крускала
    vector<int> parent(vertices); //для хранения родителей вершин
    for (int i = 0; i < vertices; i++) { //инициализируем родителей,
        parent[i] = i;          //каждая вершина – родитель сама себе
    }
    vector<pair<int, pair<int, int>>> edges; //для хранения всех ребер
    for (int i = 0; i < vertices; i++) { //проходим по всем вершинам
        for (int j = i; j < vertices; j++) { //по всем остальным вершинам
            if (adjacency_matrix[i][j] != 0) { //есть ли ребро
                edges.push_back({ adjacency_matrix[i][j], {i, j} }); //добавляем ребро с весом в вектор
            }
        }
    }
    sort(edges.begin(), edges.end()); //сортируем ребра по весу
    int mst_weight = 0; //общий вес остовного дерева
    vector<pair<int, int>> mst_edges; //для хранения ребер остовного дерева
    for (auto edge : edges) { //по всем ребрам
        int weight = edge.first; //вес ребра
        int start = edge.second.first; //стартовая вершина
        int end = edge.second.second;  //конечная вершина
        if (find(start, parent) != find(end, parent)) { //если вершины разных компонентов
            unite(start, end, parent); //объединям их
            mst_weight += weight;//общий вес +
            mst_edges.push_back({ start, end }); //добавляем ребро в остовное дерево
        }
    }
    return { mst_edges, mst_weight }; //ребра и вес остовного дерева
}
```

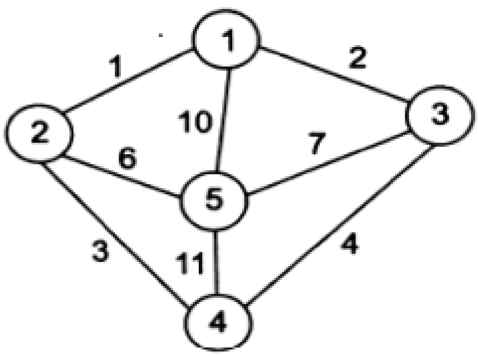
Рисунок 3 – Код программы

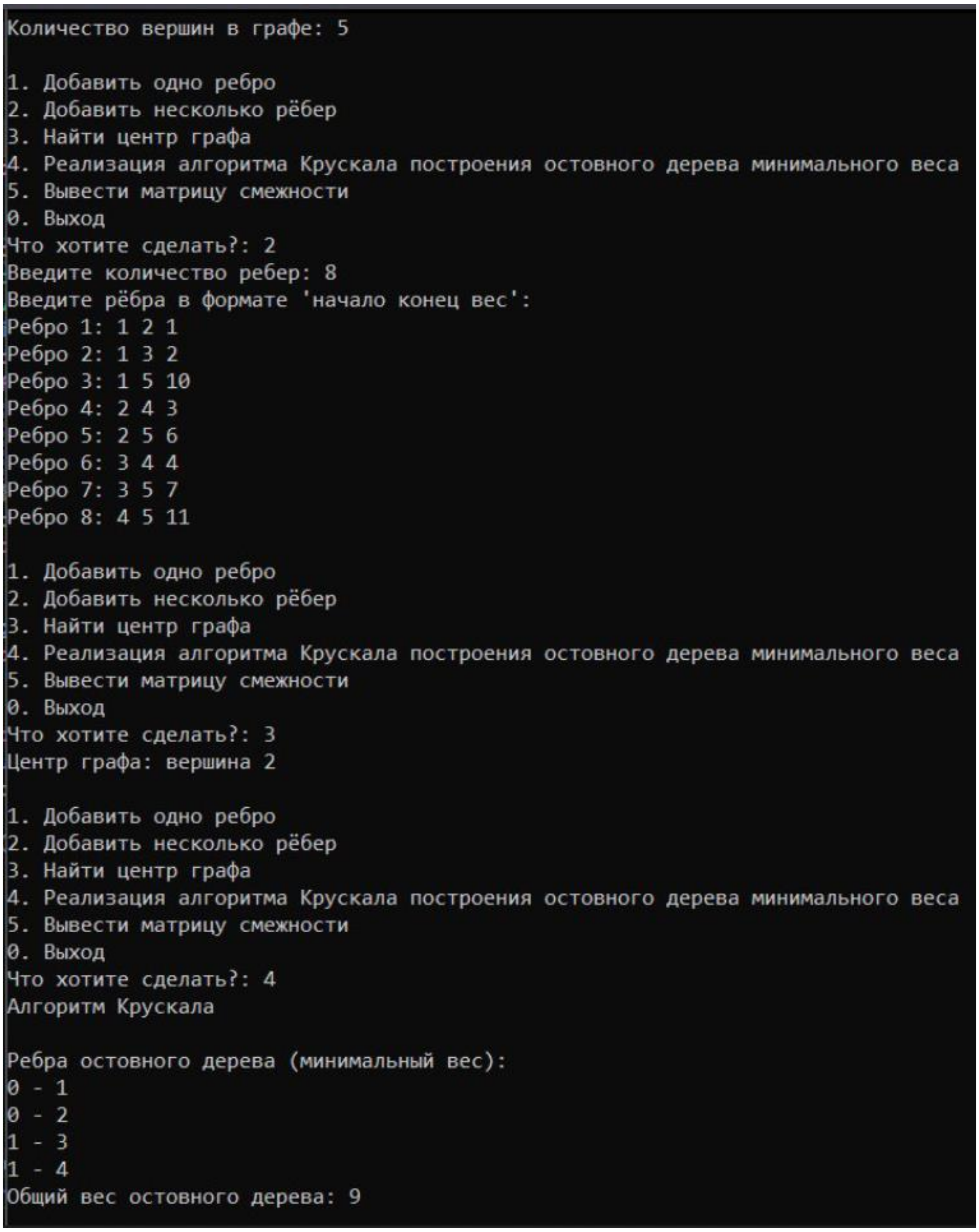
```
int main() {
    setlocale(LC_ALL, "rus");
    int vertices;
    cout << "Количество вершин в графе: ";
    cin >> vertices;
    Graph graph(vertices);
    int choice;
    do {
        cout << "\n1. Добавить одно ребро\n";
        cout << "2. Добавить несколько рёбер\n";
        cout << "3. Найти центр графа\n";
        cout << "4. Реализация алгоритма Крускала построения остовного дерева минимального веса\n";
        cout << "5. Вывести матрицу смежности\n";
        cout << "0. Выход\n";
        cout << "Что хотите сделать?: ";
        cin >> choice;
        switch (choice) {
        case 1: {
            int u, v, weight;
            cout << "Введите ребро (начало конец вес): ";
            cin >> u >> v >> weight;
            graph.addEdge(u, v, weight);
            break;
        }
        case 2: {
            int edges;
            cout << "Введите количество ребер: ";
            cin >> edges;
            graph.addEdgesBulk(edges);
            break;
        }
        case 3: {
            int center = graph.find_center();
            cout << "Центр графа: вершина " << center + 1 << endl;
            break;
        }
        case 4: {
            cout << "Алгоритм Крускала\n";
            pair<vector<pair<int, int>>, int> result = graph.kruskal();
            vector<pair<int, int>> mst_edges = result.first;
            int mst_weight = result.second;
            cout << "\nРебра остовного дерева (минимальный вес):\n";
            for (const auto& edge : mst_edges) {
                int start = edge.first;
                int end = edge.second;
                cout << start << " - " << end << endl;
            }
            cout << "Общий вес остовного дерева: " << mst_weight << endl;
            break;
        }
        case 5: {
            graph.display();
            break;
        }
        case 0:
            break;
        default:
            cout << "Попробуйте снова\n";
        }
    } while (choice != 0);
}
```

Рисунок 4 – main

2.3 Тестирование

Было проведено тестирование на одном из предложенных графов.





```
Количество вершин в графе: 5

1. Добавить одно ребро
2. Добавить несколько рёбер
3. Найти центр графа
4. Реализация алгоритма Крускала построения остовного дерева минимального веса
5. Вывести матрицу смежности
0. Выход
Что хотите сделать?: 2
Введите количество ребер: 8
Введите рёбра в формате 'начало конец вес':
Ребро 1: 1 2 1
Ребро 2: 1 3 2
Ребро 3: 1 5 10
Ребро 4: 2 4 3
Ребро 5: 2 5 6
Ребро 6: 3 4 4
Ребро 7: 3 5 7
Ребро 8: 4 5 11

1. Добавить одно ребро
2. Добавить несколько рёбер
3. Найти центр графа
4. Реализация алгоритма Крускала построения остовного дерева минимального веса
5. Вывести матрицу смежности
0. Выход
Что хотите сделать?: 3
Центр графа: вершина 2

1. Добавить одно ребро
2. Добавить несколько рёбер
3. Найти центр графа
4. Реализация алгоритма Крускала построения остовного дерева минимального веса
5. Вывести матрицу смежности
0. Выход
Что хотите сделать?: 4
Алгоритм Крускала

Ребра остовного дерева (минимальный вес):
0 - 1
0 - 2
1 - 3
1 - 4
Общий вес остовного дерева: 9
```

Рисунок 5 – Тестирование программы

```
1. Добавить одно ребро
2. Добавить несколько рёбер
3. Найти центр графа
4. Реализация алгоритма Крускала построения остовного дерева минимального веса
5. Вывести матрицу смежности
0. Выход
Что хотите сделать?: 5
0 1 1 0 7
1 0 0 1 6
1 0 0 1 6
0 1 1 0 7
7 6 6 7 0

1. Добавить одно ребро
2. Добавить несколько рёбер
3. Найти центр графа
4. Реализация алгоритма Крускала построения остовного дерева минимального веса
5. Вывести матрицу смежности
0. Выход
Что хотите сделать?: 1
Введите ребро (начало конец вес): 1 4 3

1. Добавить одно ребро
2. Добавить несколько рёбер
3. Найти центр графа
4. Реализация алгоритма Крускала построения остовного дерева минимального веса
5. Вывести матрицу смежности
0. Выход
Что хотите сделать?: 5
0 1 1 3 7
1 0 0 1 6
1 0 0 1 6
3 1 1 0 7
7 6 6 7 0
```

Рисунок 6 – Тестирование программы

ВЫВОД

В ходе выполнения практической работы был создан класс для представления взвешенного графа, использующего список смежных вершин. Графы представляют собой структуры данных, состоящие из вершин и рёбер, которые могут быть как направленными, так и ненаправленными. В этой задаче использован список смежных вершин графа, где веса рёбер могут отражать, например, стоимость или расстояние между вершинами. Также были реализованы методы вывода всех цепочек в графе, используя метод поиска в ширину и нахождения кратчайших путей в графе.

ИНФОРМАЦИОННЫЕ ИСТОЧНИКИ

1. Вирт Н. Алгоритмы и структуры данных. Новая версия для Оберона, 2010.
2. Практические занятия Сорокин А.В. - РТУ МИРЭА 2024
3. Лекции Бузыкова Ю.С. – РТУ МИРЭА 2024
4. Практические занятия преподавателя Муравьевой Е. А., 2024.

1. Лекции Лозовский В.В. – РТУ МИРЭА 2024