МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра ИС

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмы и структура данных»

Тема: Графы

Студент гр. 3374 _____ Лобачев И.М.

Преподаватель _____ _____

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Лобачев И.М.

Группа 3374

Тема работы: Графы

Исходные данные:

Реализовать алгоритм поиска минимального остова на основе алгоритма Краскала.

Дата выдачи задания: _____

Дата сдачи реферата: _____

Дата защиты реферата: _____

Студент _____ Лобачев И.М.

Преподаватель _____ _____

АННОТАЦИЯ

Код, представленный в данной работе, разработан для работы с графовыми структурами данных. В центре внимания находится реализация алгоритма Краскала, который служит для построения минимального остовного дерева (MST). Этот алгоритм играет ключевую роль в различных задачах теории графов и оптимизации, позволяя эффективно находить минимальные подмножества рёбер, соединяющие все вершины графа с минимальными затратами.

Кроме того, в рамках реализации алгоритма Краскала используются несколько вспомогательных методов и алгоритмов, которые значительно упрощают и ускоряют процесс. Среди них можно выделить алгоритм сортировки, необходимый для упорядочивания рёбер по весу, что является важным шагом перед применением самого алгоритма Краскала. Также важным компонентом является система непересекающихся множеств (СНМ), которая позволяет эффективно управлять и объединять компоненты графа, обеспечивая при этом быструю проверку наличия циклов.

Таким образом, код данной работы представляет собой комплексное решение для работы с графами, включающее как основную логику алгоритма Краскала, так и необходимые вспомогательные инструменты, что позволяет достичь высокой эффективности и надежности в построении минимального остовного дерева.

Алгоритмы и их алгоритмическая сложность

1. Сортировка ребер графа

Для сортировки массива рёбер в алгоритме Краскала применяется стандартная библиотечная функция std::sort, которая основана на алгоритме Introsort. Этот алгоритм сочетает в себе несколько методов сортировки, включая быструю сортировку, сортировку слиянием и сортировку вставками, что позволяет ему эффективно справляться с различными сценариями. Сложность Introsort составляет O(E log E), где E — это общее количество рёбер в графе.

Такой подход к сортировке обеспечивает надежную и быструю обработку данных, что особенно важно при работе с графами, где количество рёбер может быть значительным. Эффективность алгоритма Introsort позволяет минимизировать время выполнения сортировки, что, в свою очередь, способствует ускорению общего процесса построения минимального остовного дерева.

2. Алгоритм Краскала

Алгоритм Краскала представляет собой метод, используемый для построения минимального остовного дерева (MST) в контексте взвешенного неориентированного графа. Минимальное остовное дерево — это особый подграф, который соединяет все вершины исходного графа, при этом не создавая циклов, и обладает минимальной общей суммой весов своих рёбер.

Работа алгоритма Краскала основывается на жадном подходе, что означает, что на каждом этапе выбирается ребро с наименьшим весом среди доступных, которое не приводит к образованию циклов в уже сформированном остовном дереве. Этот процесс повторяется до тех пор, пока в остовном дереве не окажется ровно $V-1$ рёбер, где V обозначает общее количество вершин в графе.

Алгоритм начинается с того, что все рёбра графа сортируются по возрастанию их весов. Затем в процессе итерации по отсортированному списку рёбер выбираются те, которые могут быть добавлены в остовное дерево без нарушения его свойств. Для проверки того, образует ли добавляемое ребро цикл, обычно используется структура данных, известная как "система непересекающихся множеств" (union-find). Эта структура позволяет эффективно отслеживать и объединять компоненты связности.

Таким образом, алгоритм Краскала обеспечивает эффективное построение минимального остовного дерева, что является важной задачей в теории графов и имеет множество практических применений в различных областях, таких как сети, планирование и оптимизация.

Цель работы

Основной целью разработанного программного обеспечения является работа с графами. Программа реализует:

1. Построение минимального остовного дерева с использованием алгоритма Краскала.
2. Представление графа в различных формах (список смежности, матрица инцидентности).
3. Реализацию обхода графа двумя способами — в глубину (DFS) и в ширину (BFS).

Что такое алгоритм Краскала

Метод Краскала, является одним из методов анализа многомерных данных,используемым для визуализации и обработки данных с большим числом переменных.Он принадлежит к классу алгоритмов, известных как метод главных компонент или метод многомерного шкалирования.

Основная идея метода Краскала заключается в создании двухмерного или трехмерного представления многомерных данных,сохраняя при этом максимально возможные расстояния между объектами. Это позволяет выявлять или визуализировать скрытые структуры и закономерности в данных.

Метод Краскала часто применяется в таких областях, как психологические исследования, социология, маркетинг и биостатистика.

Краткая реализация этого метода включает следующие шаги:

1. Сбор данных: Получение многомерного массива данных.

2. Определение матрицы расстояний: Вычисление расстояний или различий между всеми парами объектов.

3. Оптимизация: Использование итерационных методов для минимизации функции потерь,

чтобы расположить объекты на плоскости или в пространстве с учетом их взаимных расстояний.

4. Визуализация: Построение графиков или диаграмм, чтобы наглядно представить полученные результаты.

Метод Краскала является мощным инструментом для анализа сложных наборов данных и помогает исследователям визуально интерпретировать свои результаты.

Что делает код

Данный код реализует набор алгоритмов и структур данных для работы с графами. Основная задача программы - считывание графа из файла, его представление в разных формах и выполнение ряда операций, таких как построение минимального остовного дерева (алгоритм Краскала), обходы графа в глубину (DFS) и ширину (BFS).

Описание работы кода

1. **Считывание данных из файла:**

   Программа считывает список вершин и матрицу смежности графа из текстового файла. На основании данных формируется список рёбер с указанием их весов.

2. **Представление графа:**

   Список смежности: создаётся для представления графа и удобства выполнения обходов.

   Матрица инцидентности: отображает взаимосвязь вершин и рёбер.

3. **Алгоритм Краскала:**

   Реализуется построение минимального остовного дерева (МОД) на основе алгоритма Краскала. Используется система непересекающихся множеств для объединения вершин и предотвращения циклов. Результатом является список рёбер МОД и его общий вес.

4. **Обходы графа:**

   DFS (поиск в глубину): реализован рекурсивно, с выводом последовательности вершин.

   BFS (поиск в ширину): выполняется с использованием очереди, также выводит последовательность вершин.

Функционал кода

1. **Ввод данных**

   Из файла считываются вершины графа (их имена) и матрица смежности.

2. **Формирование структуры графа**

   Генерируются список рёбер, список смежности и матрица инцидентности.

3. **Алгоритм Краскала**

   Вычисляется минимальное остовное дерево, выводятся его рёбра и общий вес.

4. **Обходы**

   Выполняются обходы графа (DFS и BFS) с выводом порядка посещения вершин.

Код программы

```
#include <iostream>
#include <fstream>
#include <vector>
#include <tuple>
#include <algorithm>
#include <string>
#include <queue>
#include <unordered_map>
using namespace std;
struct Edge {
    int weight;
    int u;
    int v;
};
int find(int parent[], int i) {
    if (parent[i] == -1)
        return i;
    return parent[i] = find(parent, parent[i]);
}
void unionSets(int parent[], int rank[], int x, int y) {
    int xroot = find(parent, x);
    int yroot = find(parent, y);
    if (xroot != yroot) {
        if (rank[xroot] < rank[yroot]) {
            parent[xroot] = yroot;
        } else if (rank[xroot] > rank[yroot]) {
            parent[yroot] = xroot;
        } else {
            parent[yroot] = xroot;
            rank[xroot]++;
        }
    }
```

```
}
void kruskal(int vertices, vector<Edge>& edges, const vector<string>& vertexNames) {
    sort(edges.begin(), edges.end(), [](Edge a, Edge b) { return a.weight < b.weight; });
    int parent[vertices], rank[vertices];
    fill_n(parent, vertices, -1);
    fill_n(rank, vertices, 0);
    vector<Edge> result;
    int totalWeight = 0;
    for (const auto& edge : edges) {
        if (find(parent, edge.u) != find(parent, edge.v)) {
            result.push_back(edge);
            unionSets(parent, rank, edge.u, edge.v);
            totalWeight += edge.weight;
        }
    }
    cout << "Минимальное остовное дерево:" << endl;
    for (const auto& edge : result) {
        cout << vertexNames[edge.u] << " " << vertexNames[edge.v] << endl;
    }
    cout << totalWeight << endl;
}
void dfs(int vertex, vector<bool>& visited, const vector<vector<int>>& adjacencyList, const
vector<string>& vertexNames) {
    visited[vertex] = true;
    cout << vertexNames[vertex] << " ";
    for (int i = adjacencyList[vertex].size() - 1; i >= 0; --i) {
        int neighbor = adjacencyList[vertex][i];
        if (!visited[neighbor]) {
            dfs(neighbor, visited, adjacencyList, vertexNames);
        }
    }
}
void bfs(int startVertex, const vector<vector<int>>& adjacencyList, const vector<string>&
vertexNames) {
    vector<bool> visited(adjacencyList.size(), false);
```

```
    queue<int> q;
    q.push(startVertex);
    visited[startVertex] = true;
    while (!q.empty()) {
        int vertex = q.front();
        q.pop();
        cout << vertexNames[vertex] << " ";
        for (int neighbor : adjacencyList[vertex]) {
            if (!visited[neighbor]) {
                visited[neighbor] = true;
                q.push(neighbor);
            }
        }
    }
}
vector<vector<int>> buildAdjacencyList(const vector<Edge>& edges, int vertices) {
    vector<vector<int>> adjacencyList(vertices);
    for (const auto& edge : edges) {
        adjacencyList[edge.u].push_back(edge.v);
        adjacencyList[edge.v].push_back(edge.u);
    }
    return adjacencyList;
}
vector<vector<int>> buildIncidenceMatrix(const vector<Edge>& edges, int vertices) {
    vector<vector<int>> incidenceMatrix(vertices, vector<int>(edges.size(), 0));
    for (int i = 0; i < edges.size(); ++i) {
        incidenceMatrix[edges[i].u][i] = 1;
        incidenceMatrix[edges[i].v][i] = 1;
    }
    return incidenceMatrix;
}
int main() {
    system("chcp 65001");
    system("cls");
    ifstream input("C:\\Users\\IVAN\\Documents\\AIST\\graph.txt");
```

```
    string line;
    getline(input, line);
    vector<string> vertexNames;
    size_t pos = 0;
    while ((pos = line.find(' ')) != string::npos) {
        vertexNames.push_back(line.substr(0, pos));
        line.erase(0, pos + 1);
    }
    vertexNames.push_back(line);
    int numVertices = vertexNames.size();
    vector<Edge> edges;
    for (int i = 0; i < numVertices; ++i) {
        for (int j = 0; j < numVertices; ++j) {
            int weight;
            input >> weight;
            if (weight > 0 && i < j) {
                edges.push_back({weight, i, j});
            }
        }
    }
    input.close();
    vector<vector<int>> adjacencyList = buildAdjacencyList(edges, numVertices);
    vector<vector<int>> incidenceMatrix = buildIncidenceMatrix(edges, numVertices);
    kruskal(numVertices, edges, vertexNames);
    cout << endl;
    cout << "Обход в глубину (DFS): \t";
    vector<bool> visited(numVertices, false);
    dfs(0, visited, adjacencyList, vertexNames);
    cout << endl;
    cout << "Обход в ширину (BFS): \t";
    bfs(0, adjacencyList, vertexNames);
    cout << endl;
    cout << endl;
    return 0;
}
```

Входные данные

A B C D

0 5 3 0

5 0 0 4

3 0 0 7

0 4 7 0

Вывод данные после обработки

A C

B D

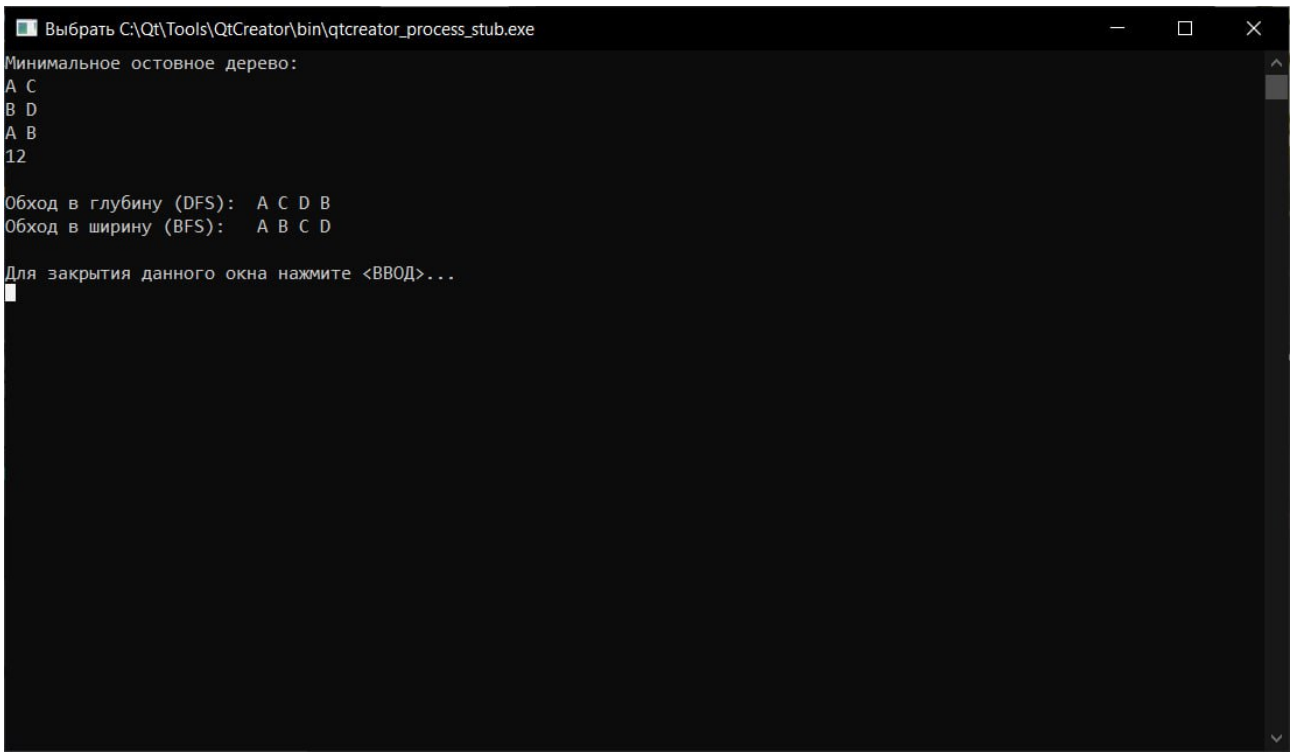
A B

12

Обход в глубину (DFS): A C D B

Обход в ширину (BFS): A B C D

Что выводится в консоль



Практическая значимость

Разработанная программа имеет широкую область применения:

Оптимизация сетей: нахождение минимального остовного дерева применимо в задачах оптимизации транспортных и коммуникационных сетей.

Моделирование процессов: анализ графов используется в логистике, экономике и других науках.

Обучение: программа может быть полезна для изучения базовых алгоритмов и структур данных, связанных с графами.

Вывод

В данной работе была выполнена реализация нескольких важных алгоритмов теории графов, включая алгоритм Краскала и методы обхода. Мы показали, как можно применять различные представления графа для решения прикладных задач. Программа является универсальным инструментом для изучения и анализа графовых структур, а также может служить основой для дальнейшего расширения, например, для работы с направленными графами, динамическими изменениями графа или использованием других алгоритмов (например, Прима или Дейкстры).