/\*

    Name : Ayush Pandey

    Roll No : 3317

    ASSIGNMENT-3

    Problem Statement :

        Implement Greedy search algorithm for any of the following application:

            I.   Selection Sort

            II.  Minimum Spanning Tree

            III. Single-Source Shortest Path Problem

            IV.  Job Scheduling Problem

            V.   Prim's Minimal Spanning Tree Algorithm

            VI.  Kruskal's Minimal Spanning Tree Algorithm

            VII. Dijkstra's Minimal Spanning Tree Algorithm

\*/

#include <iostream>

#include <vector>

#include <algorithm>

#include <climits>

using namespace std;

struct Edge {

    int src, dest, weight;

};

struct Job {

    char id;

    int deadline, profit;

};

void selectionSort(vector<int> &arr) {

    int n = arr.size();

    for (int i = 0; i < n - 1; i++) {

        int minIndex = i;

        for (int j = i + 1; j < n; j++) {

            if (arr[j] < arr[minIndex])

                minIndex = j;

        }

        swap(arr[i], arr[minIndex]);

    }

    cout << "Sorted array: ";

    for (int num : arr) cout << num << " ";

    cout << endl;

}

void primMST(vector<vector<int>> &graph) {

    int V = graph.size();

    vector<int> key(V, INT\_MAX), parent(V, -1);

    vector<bool> inMST(V, false);

    key[0] = 0;

    for (int count = 0; count < V - 1; count++) {

        int minKey = INT\_MAX, u = -1;

        for (int v = 0; v < V; v++)

            if (!inMST[v] && key[v] < minKey)

                minKey = key[v], u = v;

        inMST[u] = true;

        for (int v = 0; v < V; v++)

            if (graph[u][v] && !inMST[v] && graph[u][v] < key[v])

                key[v] = graph[u][v], parent[v] = u;

    }

    cout << "Minimum Spanning Tree (Prim's Algorithm):\n";

    for (int i = 1; i < V; i++)

        cout << parent[i] << " - " << i << "  " << graph[i][parent[i]] << endl;

}

bool edgeComparison(Edge a, Edge b) {

    return a.weight < b.weight;

}

int findParent(vector<int> &parent, int i) {

    if (parent[i] == -1)

        return i;

    return findParent(parent, parent[i]);

}

void kruskalMST(vector<Edge> &edges, int V) {

    sort(edges.begin(), edges.end(), edgeComparison);

    vector<int> parent(V, -1);

    vector<Edge> mst;

    for (Edge edge : edges) {

        int srcParent = findParent(parent, edge.src);

        int destParent = findParent(parent, edge.dest);

        if (srcParent != destParent) {

            mst.push\_back(edge);

            parent[srcParent] = destParent;

        }

    }

    cout << "Minimum Spanning Tree (Kruskal's Algorithm):\n";

    for (Edge e : mst)

        cout << e.src << " - " << e.dest << "  " << e.weight << endl;

}

void dijkstra(vector<vector<int>> &graph, int src) {

    int V = graph.size();

    vector<int> dist(V, INT\_MAX);

    vector<bool> visited(V, false);

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {

        int minDist = INT\_MAX, u = -1;

        for (int v = 0; v < V; v++)

            if (!visited[v] && dist[v] < minDist)

                minDist = dist[v], u = v;

        visited[u] = true;

        for (int v = 0; v < V; v++)

            if (graph[u][v] && !visited[v] && dist[u] + graph[u][v] < dist[v])

                dist[v] = dist[u] + graph[u][v];

    }

    cout << "Shortest Path from Source " << src << ":\n";

    for (int i = 0; i < V; i++)

        cout << "To " << i << "  Distance: " << dist[i] << endl;

}

bool jobComparison(Job a, Job b) {

    return a.profit > b.profit;

}

void jobScheduling(vector<Job> &jobs, int maxDeadline) {

    sort(jobs.begin(), jobs.end(), jobComparison);

    vector<char> schedule(maxDeadline, '-');

    for (Job job : jobs) {

        for (int j = min(maxDeadline, job.deadline) - 1; j >= 0; j--) {

            if (schedule[j] == '-') {

                schedule[j] = job.id;

                break;

            }

        }

    }

    cout << "Job Sequence for Maximum Profit: ";

    for (char c : schedule)

        if (c != '-')

            cout << c << " ";

    cout << endl;

}

int main() {

    int choice;

    do {

        cout << "\nMenu:\n";

        cout << "1. Selection Sort\n";

        cout << "2. Minimum Spanning Tree - Prim's Algorithm\n";

        cout << "3. Minimum Spanning Tree - Kruskal's Algorithm\n";

        cout << "4. Dijkstra's Single Source Shortest Path Algorithm\n";

        cout << "5. Job Scheduling Problem\n";

        cout << "6. Exit\n";

        cout << "Enter your choice: ";

        cin >> choice;

        switch (choice) {

            case 1: {

                int n;

                cout << "Enter number of elements: ";

                cin >> n;

                vector<int> arr(n);

                cout << "Enter elements: ";

                for (int &num : arr) cin >> num;

                selectionSort(arr);

                break;

            }

            case 2: {

                int V;

                cout << "Enter number of vertices: ";

                cin >> V;

                vector<vector<int>> graph(V, vector<int>(V));

                cout << "Enter adjacency matrix:\n";

                for (int i = 0; i < V; i++)

                    for (int j = 0; j < V; j++)

                        cin >> graph[i][j];

                primMST(graph);

                break;

            }

            case 3: {

                int V, E;

                cout << "Enter number of vertices and edges: ";

                cin >> V >> E;

                vector<Edge> edges(E);

                cout << "Enter edges (src, dest, weight):\n";

                for (Edge &edge : edges)

                    cin >> edge.src >> edge.dest >> edge.weight;

                kruskalMST(edges, V);

                break;

            }

            case 4: {

                int V, src;

                cout << "Enter number of vertices: ";

                cin >> V;

                vector<vector<int>> graph(V, vector<int>(V));

                cout << "Enter adjacency matrix:\n";

                for (int i = 0; i < V; i++)

                    for (int j = 0; j < V; j++)

                        cin >> graph[i][j];

                cout << "Enter source vertex: ";

                cin >> src;

                dijkstra(graph, src);

                break;

            }

            case 5: {

                int n, maxDeadline = 0;

                cout << "Enter number of jobs: ";

                cin >> n;

                vector<Job> jobs(n);

                cout << "Enter job id, deadline, and profit:\n";

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

                    cin >> jobs[i].id >> jobs[i].deadline >> jobs[i].profit;

                    maxDeadline = max(maxDeadline, jobs[i].deadline);

                }

                jobScheduling(jobs, maxDeadline);

                break;

            }

        }

    } while (choice != 6);

    return 0;

}



