محمد اسامة محمد حلمي ذكاء اصطناعي

Prim's Algorithm Project

Prim's Algorithm is a greedy algorithm that finds the Minimum Spanning Tree (MST) of a connected, weighted, and undirected graph. It ensures all vertices are connected with the minimum total edge weight.

(a) Required Algorithms for Prim's Algorithm

To find the MST using Prim’s algorithm, the following steps are performed:

1. **Initialize the MST:** Start with an arbitrary vertex and add it to the MST. Track vertices already included.
2. **Choose the minimum edge:** At each step, choose the edge with the smallest weight that connects a vertex in the MST to a vertex outside the MST.
3. **Repeat until all vertices are included.**

PRIM(G, V):

1. Initialize a Min-Heap (Priority Queue) for edge weights.
2. Pick an arbitrary starting vertex (u).
3. Mark u as visited and add its edges to the Min-Heap.
4. While Min-Heap is not empty:
    a. Extract the edge with the smallest weight (u, v).
    b. If v is not visited:
        i. Add edge (u, v) to the MST.
        ii. Mark v as visited.
        iii. Add all edges of v to the Min-Heap.
5. Return the MST edges and their total weight.

Analysis of Prim's Algorithm

1. Time Complexity

- **Edge Insertions into the Min-Heap:** O(Elog⁡V)O(E \log V)O(ElogV), where EEE is the number of edges.
- **Extracting Minimum Weight Edge:** O(Vlog⁡V)O(V \log V)O(VlogV), as there are VVV vertices.
- **Total Complexity:** O((V+E)log⁡V)O((V + E) \log V)O((V+E)logV). In a dense graph, EEE can be close to V2V^2V2, making the complexity O(Elog⁡V)O(E \log V)O(ElogV).

2. Space Complexity

- **Graph Representation:** O(V+E)O(V + E)O(V+E) for adjacency list.
- **Priority Queue:** O(E)O(E)O(E), as all edges are stored.

3. Stability

- Prim's Algorithm always produces the correct MST for connected, undirected graphs.

4. Constraints

- The graph must be connected, weighted, and undirected.

Implementation in C++

```
#include <iostream>
#include <vector>
#include <queue>
#include <tuple>
#include <climits>

using namespace std;

typedef pair<int, int> Edge; // (weight, destination)

void prims_algorithm(vector<vector<Edge>>& graph, int start) {
  int n = graph.size();
  vector<bool> visited(n, false);
  priority_queue<Edge, vector<Edge>, greater<Edge>> min_heap;

  vector<pair<int, int>> mst;
  int total_weight = 0;

    visited[start] = true;
  for (auto& edge : graph[start]) {
```

```
        min_heap.push(edge);
    }

        while (!min_heap.empty()) {
        auto [weight, v] = min_heap.top();
        min_heap.pop();

        if (!visited[v]) {
            visited[v] = true;
            total_weight += weight;
            mst.push_back({weight, v});


            for (auto& edge : graph[v]) {
                if (!visited[edge.second]) {
                    min_heap.push(edge);
                }
            }
        }
    }

        cout << "Edges in MST:" << endl;
    for (auto& edge : mst) {
        cout << "Weight: " << edge.first << ", Vertex: " << edge.second << endl;
    }
    cout << "Total Weight of MST: " << total_weight << endl;
}

int main() {
    vector<vector<Edge>> graph = {
        {{2, 1}, {6, 3}},         {{2, 0}, {3, 2}, {8, 3}},
        {{3, 1}, {5, 3}},
        {{6, 0}, {8, 1}, {5, 2}}       };

    prims_algorithm(graph, 0);

    return 0;
}
```