**ET3272: Design and Analysis of Algorithms**

**Name of the student: Aditya Jannawar Roll No.: 02**

**Div.: A Batch: 1**

**Date of performance: /0 /2023**

**Experiment No. 15**

# Title: Prim MST

**Theory/Description of the Problem Statement:**

Prim's algorithm is a greedy algorithm used to find the minimum spanning tree (MST) of a weighted undirected graph. The algorithm starts with a single vertex and grows the tree by adding the minimum weight edge at each step until all vertices are covered in the MST.

Prim's algorithm uses a priority queue to keep track of the minimum weight edge to add at each step. It maintains two sets of vertices: one set that is already included in the MST, and another set that is not yet included. At each step, it adds the minimum weight edge that connects a vertex in the included set to a vertex in the not included set.

The time complexity of Prim's algorithm is O(E log V), where E is the number of edges and V is the number of vertices in the graph. The space complexity is O(V), as it stores a priority queue of size V and an array to keep track of the vertices included in the MST.

**Algorithm :**

* Create a set mstSet that keeps track of vertices already included in MST.
* Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign the key value as 0 for the first vertex so that it is picked first.
* While mstSet doesn’t include all vertices
* Pick a vertex u that is not there in mstSet and has a minimum key value.
* Include u in the mstSet.
* Update the key value of all adjacent vertices of u. To update the key values, iterate through all adjacent vertices.
* For every adjacent vertex v, if the weight of edge u-v is less than the previous key value of v, update the key value as the weight of u-v.

**Pseudo Code :**

* Prim(G, w, r):
* for each u ∈ V[G]
* key[u] ← ∞ // Initialize key values to infinity
* parent[u] ← NIL // Initialize parent values to NIL
* key[r] ← 0 // Make the key value of root node to 0
* Q ← V[G] // Add all vertices to a min-priority queue
* while Q is not empty
* u ← Extract-Min(Q) // Extract the vertex with the minimum key value
* for each v ∈ Adj[u]
* if v ∈ Q and w(u, v) < key[v]
* parent[v] ← u
* key[v] ← w(u, v) // Update the key value of vertex v in Q
* Decrease-Key(Q, v, key[v])
* return parent // Return the set of edges in MST

**Analysis of the Algorithm**

**Time Complexity:**

The time complexity of Prim's algorithm using adjacency matrix representation is O(V^2), where V is the number of vertices in the graph. This is because, in the worst-case scenario, we will visit each vertex and each edge in the graph.

**Space Complexity:**

The space complexity of Prim's algorithm using adjacency matrix representation is O(V^2), where V is the number of vertices in the graph. This is because we use an adjacency matrix to represent the graph, which requires O(V^2) space.

**Experiment and result:**

Code:

#include <bits/stdc++.h>

using namespace std;

// Number of vertices in the graph

#define V 5

int minKey(int key[], bool mstSet[])

{

    // Initialize min value

    int min = INT\_MAX, min\_index;

    for (int v = 0; v < V; v++)

        if (mstSet[v] == false && key[v] < min)

            min = key[v], min\_index = v;

    return min\_index;

}

// A utility function to print the

// constructed MST stored in parent[]

void printMST(int parent[], int graph[V][V])

{

    cout << "Edge \tWeight\n";

    for (int i = 1; i < V; i++)

        cout << parent[i] << " - " << i << " \t"

            << graph[i][parent[i]] << " \n";

}

// Function to construct and print MST for

// a graph represented using adjacency

// matrix representation

void primMST(int graph[V][V])

{

    // Array to store constructed MST

    int parent[V];

    // Key values used to pick minimum weight edge in cut

    int key[V];

    // To represent set of vertices included in MST

    bool mstSet[V];

    // Initialize all keys as INFINITE

    for (int i = 0; i < V; i++)

        key[i] = INT\_MAX, mstSet[i] = false;

    // Always include first 1st vertex in MST.

    // Make key 0 so that this vertex is picked as first

    // vertex.

    key[0] = 0;

    // First node is always root of MST

    parent[0] = -1;

    // The MST will have V vertices

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

        // Pick the minimum key vertex from the

        // set of vertices not yet included in MST

        int u = minKey(key, mstSet);

        // Add the picked vertex to the MST Set

        mstSet[u] = true;

        // Update key value and parent index of

        // the adjacent vertices of the picked vertex.

        // Consider only those vertices which are not

        // yet included in MST

        for (int v = 0; v < V; v++)

            // graph[u][v] is non zero only for adjacent

            // vertices of m mstSet[v] is false for vertices

            // not yet included in MST Update the key only

            // if graph[u][v] is smaller than key[v]

            if (graph[u][v] && mstSet[v] == false

                && graph[u][v] < key[v])

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

    }

    // Print the constructed MST

    printMST(parent, graph);

}

int main()

{

    int graph[V][V] = { { 0, 2, 0, 6, 0 },

                        { 2, 0, 3, 8, 5 },

                        { 0, 3, 0, 0, 7 },

                        { 6, 8, 0, 0, 9 },

                        { 0, 5, 7, 9, 0 } };

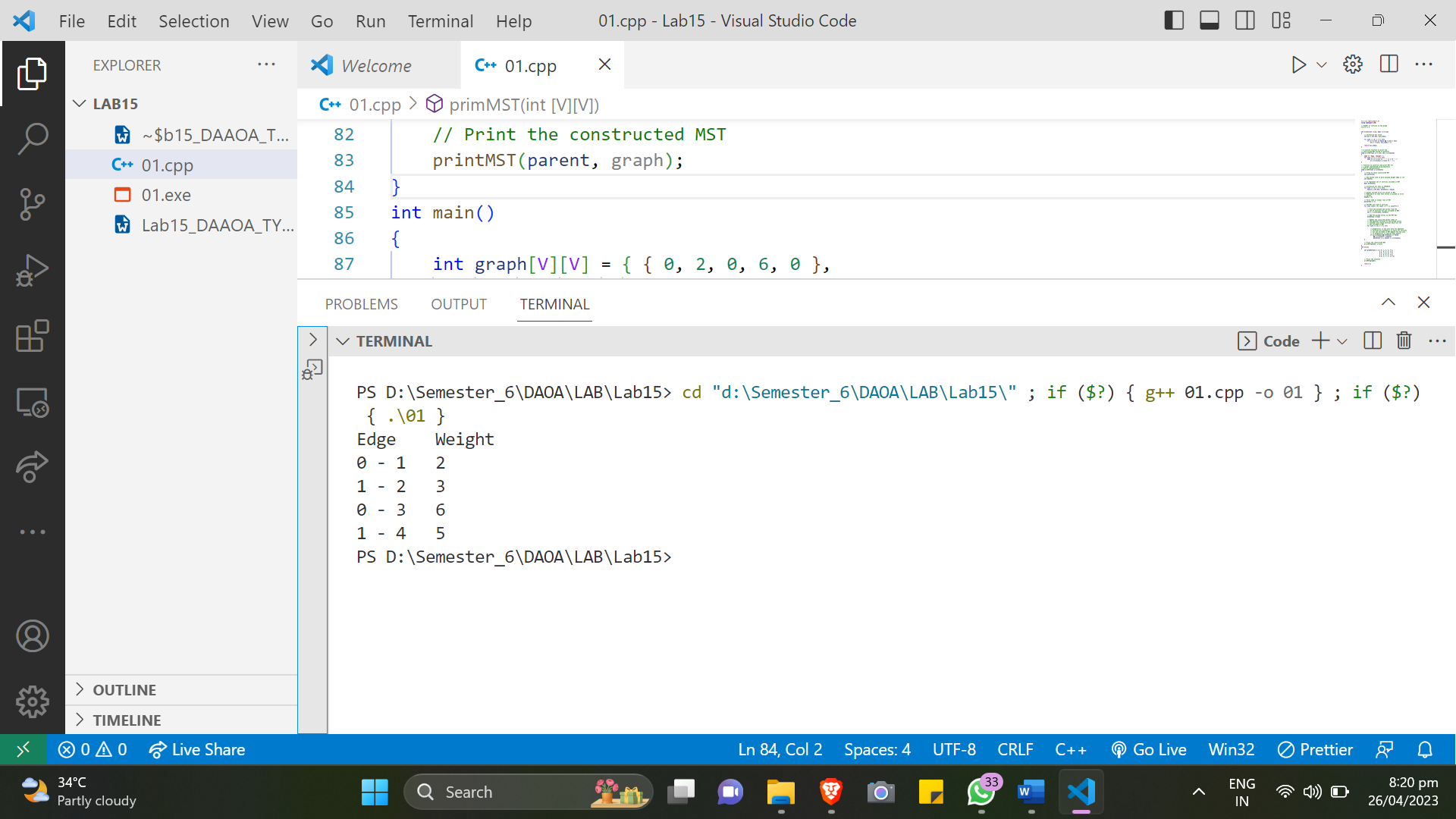
    // Print the solution

    primMST(graph);

    return 0;

}

Output:



**Conclusions:**

In conclusion, Prim's algorithm is a greedy algorithm used to find the minimum spanning tree of a connected weighted undirected graph. The algorithm works by starting from an arbitrary vertex and adding the closest vertex not already in the tree until all vertices are in the tree. The time complexity and space complexity of the algorithm using an adjacency matrix representation are O(V^2), where V is the number of vertices in the graph.