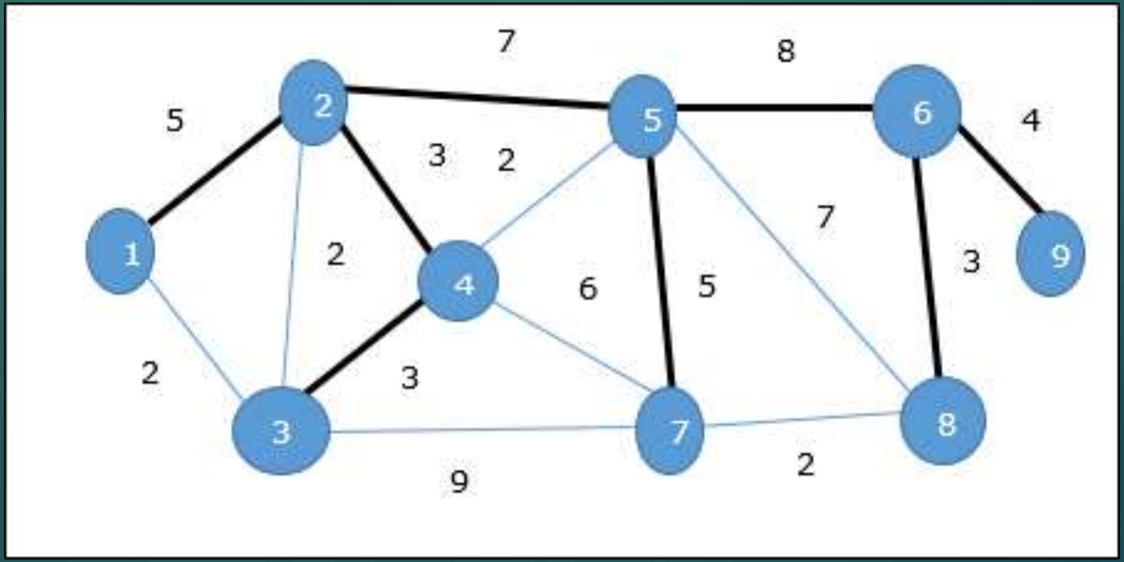
CS-101 Project

Group Members
AAYAN SONI
ADITYA GARG
RAKSHIT KAUSHAL
AMITOJ SINGH

Contents

Minimum Spanning Tree (MST)

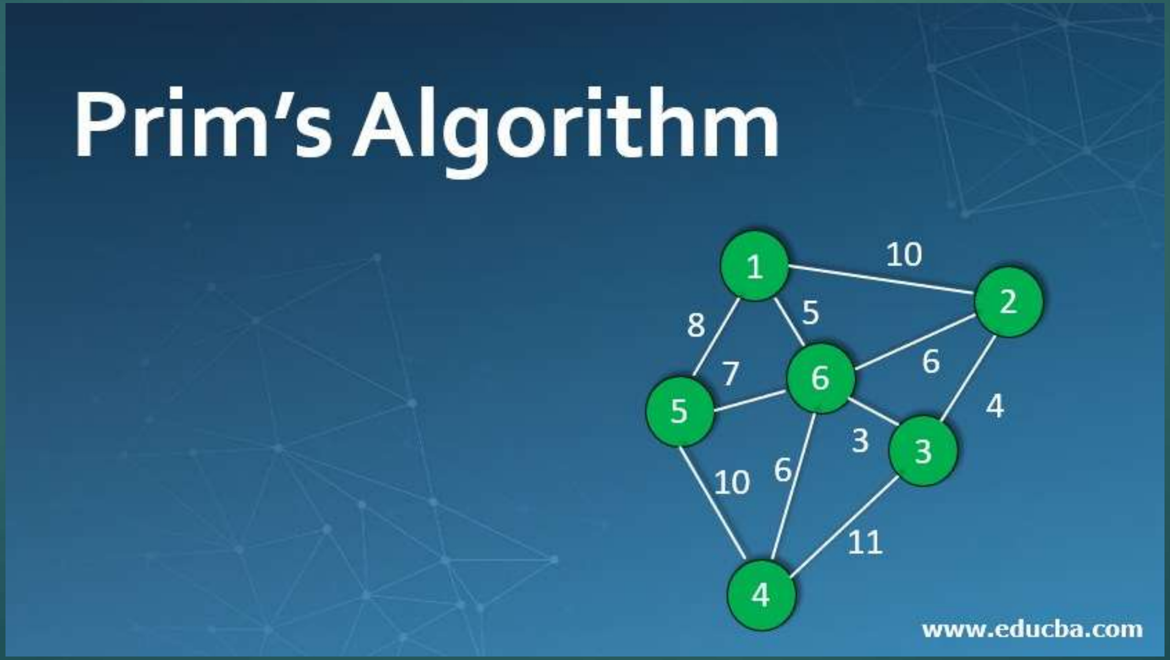
- An MST is a subset of edges of a connected, weighted, and undirected graph that contains all the vertices such that the total weight of the edges is minimum.
- An MST has no cycles in it.



Prim’s Algorithm

Introduction

Prim's algorithm is a popular algorithm used in the field of graph theory. It is designed to find a weighted undirected graph's minimum spanning tree (MST).

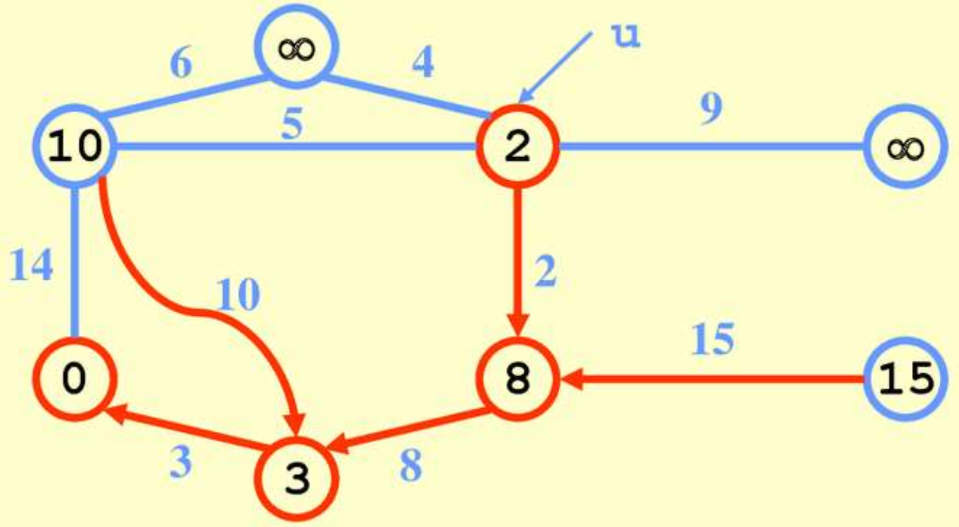


Algorithm

- We have an undirected weighted connected graph G(V,E).
- We declare an empty MST and select an arbitrary vertex to start with.
- We use a visited array to keep track of which vertices have been included in the MST and a parent array to keep track of the corresponding parent vertex in the MST.
- Then find the vertex with the minimum weight that is not yet included in the MST and is connected to a vertex that has been already visited.
- Assign the parent value and key value of the above vertex.
- Repeat the above two steps till all the vertices are visited.

Prim's Algorithm

```
MST-Prim(G, w, r)
    Q = V[G];
    for each u ∈ Q
        key[u] = ∞;
    key[r] = 0;
    p[r] = NULL;
    while (Q not empty)
        u = ExtractMin(Q);
        for each v ∈ Adj[u]
            if (v ∈ Q and w(u,v) < key[v])
                p[v] = u;
                key[v] = w(u,v);
```



Code of Prim’s Algorithm

- The code of Prim’s algorithm is attached with the file.
- The input format is as follows >>
- The first line contains an integer v, which represents the number of vertices in the graph.
- The next v lines contain v integers each representing the weight of the edges, where 0 denotes no edge.
- The output contains the edges present in the MST and their corresponding weights.

Time complexity

- The time complexity of Prim's algorithm is as follows:
- We iterate through all the vertices connected to the vertex being visited in O(V).
- We have to do the above step by iterating through all the vertices in the graph which also takes O(V) time.

This makes time complexity $O(V^2)$.

Modification to improve time complexity

- In this approach we maintain a min heap that contains all the vertices and their key values(which are initially INF for all vertices other than the starting one).
- Pop the vertex with the min key value. Let this vertex be v.
- Now if u is a vertex adjacent to v and the weight of edge u-v > key value of u, replace the weight of u-v as the key value of u.
- We do this for all the vertices adjacent to v.
- We repeat the above three steps until the min heap is empty and we get our minimum spanning tree.
- The time complexity of the above algorithm is O(ElogV).

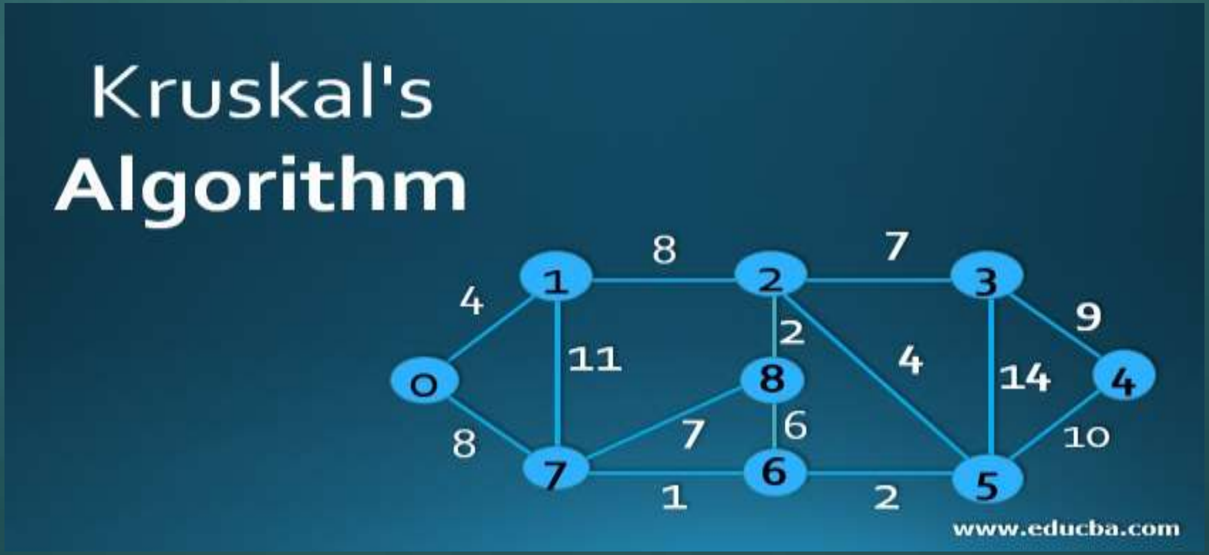
Drawbacks

- The algorithm has a time complexity of O(V^2), V being the number of vertices in the graph. The algorithm may not be very time efficient if we have a large number of vertices.
- The algorithm may give different MSTs depending upon the first chosen edge. Hence we cannot get a specific MST.
- It requires the graph to be connected. If the graph is not connected, the algorithm will only find the minimum spanning tree of one of its connected components.

KRUSKAL’s ALGORITHM

INTRODUCTION

- Kruskal's Algorithm is a very popular algorithm to generate a Minimum spanning tree (MST) of an undirected, weighted, and connected graph.
- It is a Greedy algorithm in which we makes a locally optimal choice in order to find the global optimal solution.



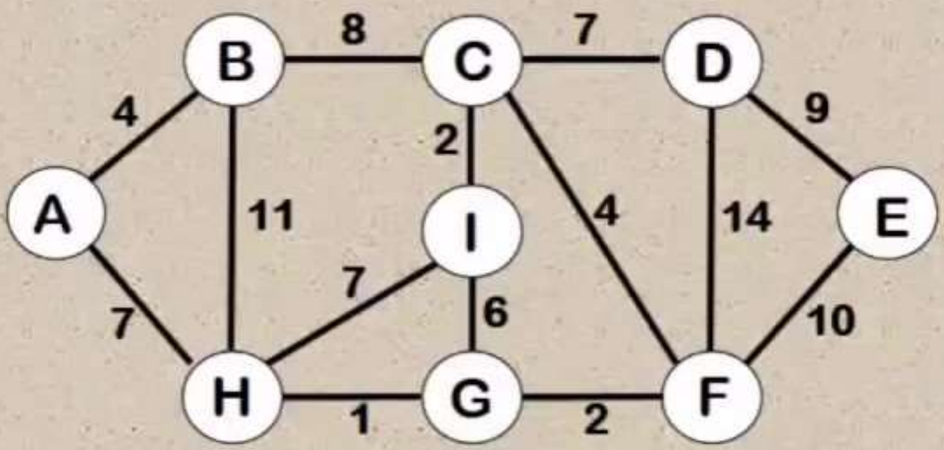
Algorithm

- Here we have an undirected weighted connected graph G(V,E) whose weights are assigned with each edge.
- Sort all the edges in ascending order according to their weights.
- Assign each node a distinct parent such that later on it will help in loop detection.
- Starting from the minimum weighted edge we keep on adding them to the answer such that the added edge doesn’t lead to a cycle in the graph.
- We check that there is no cycle by checking if their parents’ are not same.
- Repeat the process until we have v-1 edges where v is the number of vertices

Kruskal's Algorithm (Minimum Spanning Tree)

Graph:

- Connected
- Weighted
- Undirected



Minimum Spanning Tree:

- Minimum possible total edge weight
- All the vertices connected
- No cycles
- Edges are a sub set of the graph edges

Number of Vertices = 9

Stop when number of edges of the spanning tree = (# of vertices -1) = 9-1 = 8

Code of Kruskal’s Algorithm

- The code of Kruskal’s Algorithm is attached with the file.
- The input format is as follows->
- The first line contains 2 integers n, e which represent the number of vertices and the number of edges in the graph.
- The next e lines contain three integers each-first vertex, second vertex and the weight of the edge connecting these vertices.
- The output contains n-1 lines with three integers each, which tell the vertices and the weight of that edge in the minimum spanning tree of the graph.

Time Complexity

- The time complexity of the shared code of Kruskal’s Algorithm is as follows:

1. For sorting the edges we have used an inbuilt sort function which has a time complexity of O(ElogE).
2. In the last step to determine which edge to include in the MST , we checked each edge whether to include it or not and this process had a time complexity of O(E).

Hence the overall time complexity of the algorithm becomes O(ElogE).

Another approach of Kruskal’s Algorithm

- The code for Kruskal’s algorithm can also be implemented using union-find data structure which has a little poor time complexity than the earlier code presented.
- In this method we have used rank and parent arrays to detect cycle formation.
- On checking that no cycle forms on adding we use unite function to change the parents and update the information of added edge.
- We continue the process till we get n-1 edges.

Drawbacks

- **Complexity**:- The Algorithm has a time complexity of O(ElogE) , E being the number of edges in graph. The Algorithm may not be very time efficient if we have a large number of edges.
- **Lack of Edge priority Consideration**:-The Algorithm considers only those edges in the MST which obviously decrease the overall cost and do not result in a cycle, but it may happen a particular problem or a condition requires an edge to be present in the solution which normally algorithm will not consider because of basic conditions of the algorithm. Thus it is not applicable in cases where we have to construct an MST or a graph satisfying some particular conditions.

- **Memory usage-** If merge sort algorithm is used in sorting then a lot of memory wastage is there. Also in storing the parent array and the weights of edges a lot of memory is used. If there are large number of vertices and edges then the a lot of space is utilised.

Applications of Prim's & Kruskal's Algorithms

- **Cost reduction in Cable Installations**:- When installing network cables involves a cost associated with each cable invention, the above algorithms can be implemented to determine an MST where all nodes are connected and with minimum total cost.
- **Transportation of drinking water using Pipes**:- Here we consider the vertices of the graph as places where water is to be transported and the edges as potential pipe connections between two places where each pipe connection has some cost. Again we can observe that algorithms can come in very handy in designing such a system and the MST will have minimum cost setup and connections between all places.

- **<u>Connecting places via roads, bridges, etc</u>**:--This problem can be stated in many ways whether it is to connect two cities with each other or install a metro which connects all sectors in a city where all connections have some cost involved. The main condition remains the same:- We need to have connections between all places (vertices/nodes) and also that the total cost involved is minimum (total cost of edges in MST). It is again a direct application of the algorithms where we can generate an MST which adheres to the given conditions.

THANK YOU