Prim's Minimum Spanning Tree Algorithm

What is Minimum Spanning Tree?

Given a connected and undirected graph, a spanning tree of that graph is a subgraph that is a tree and connects all the vertices together. A single graph can have many different spanning trees. A

minimum spanning tree (MST)

or minimum weight spanning tree for a weighted, connected and undirected graph is a spanning tree with weight less than or equal to the weight of every other spanning tree. The weight of a spanning tree is the sum of weights given to each edge of the spanning tree.

Number of edges in a minimum spanning tree:

A minimum spanning tree has (V – 1) edges where V is the number of vertices in the given graph.

Prim's Algorithm

Prim's algorithm is also a Greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST, the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets, and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

A group of edges that connects two set of vertices in a graph is called cut in graph theory

So, at every step of Prim's algorithm, we find a cut (of two sets, one contains the vertices already included in MST and other contains rest of the vertices), pick the minimum weight edge from the cut and include this vertex to MST Set (the

Dash
All
Articles
Videos
Problems
Quiz
Contest

set that contains already included vertices).

How does Prim's Algorithm Work?

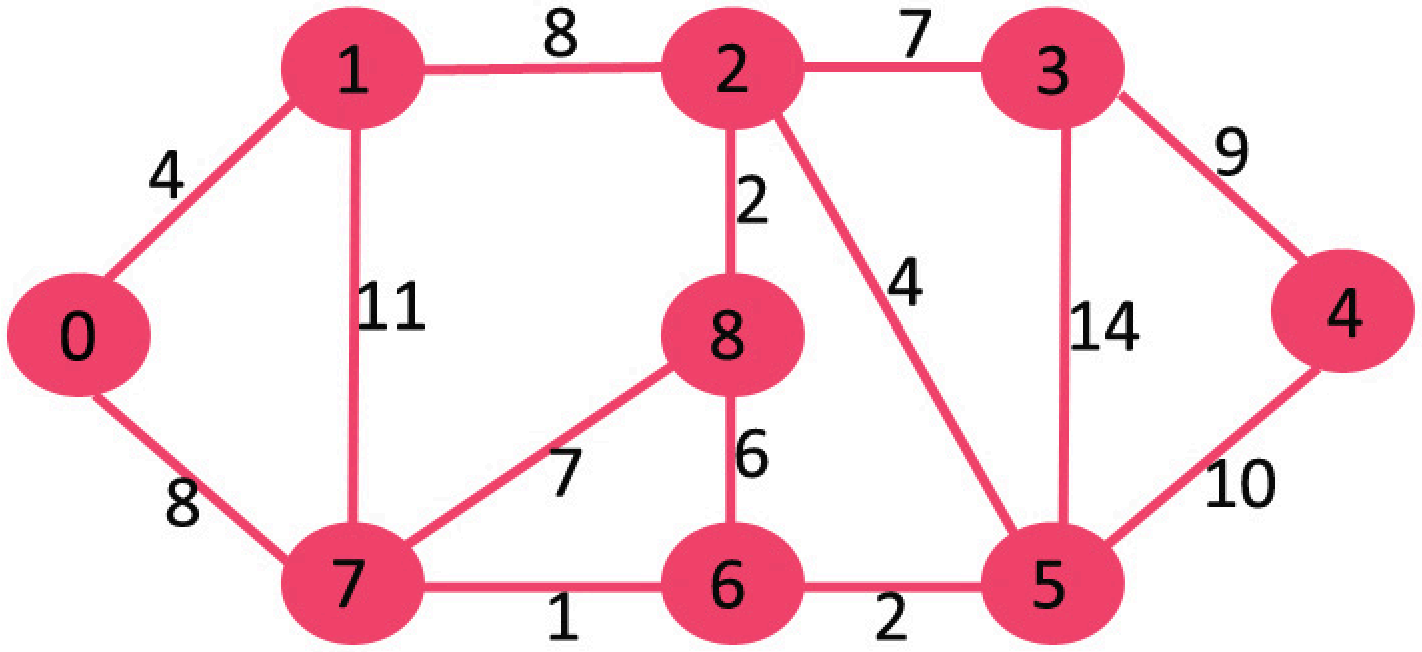
The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a *Spanning* Tree. And they must be connected with the minimum weight edge to make it a *Minimum* Spanning Tree.

Algorithm:

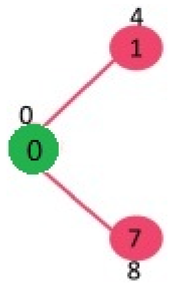
1. Create a set *mstSet* that keeps track of vertices already included in MST.
2. Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
3. While mstSet doesn't include all vertices:
   - Pick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

The idea of using key values is to pick the minimum weight edge from cut

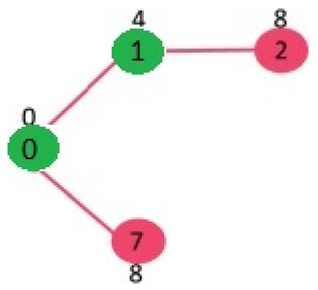
The key values are used only for vertices which are not yet included in MST, the key value for these vertices indicate the minimum weight edges connecting them to the set of vertices included in MST. Let us understand this with the help of following example:



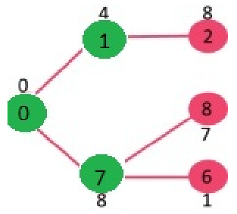
The set *mstSet* is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with the minimum key value. The vertex 0 is picked, include it in *mstSet*. So *mstSet* becomes {0}. After including to *mstSet* , update key values of adjacent vertices. Adjacent vertices of 0 are 1 and 7. The key values of 1 and 7 are updated as 4 and 8. Following subgraph shows vertices and their key values, only the vertices with finite key values are shown. The vertices included in MST are shown in green color.



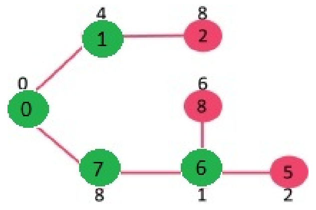
Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex 1 is picked and added to mstSet. So mstSet now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



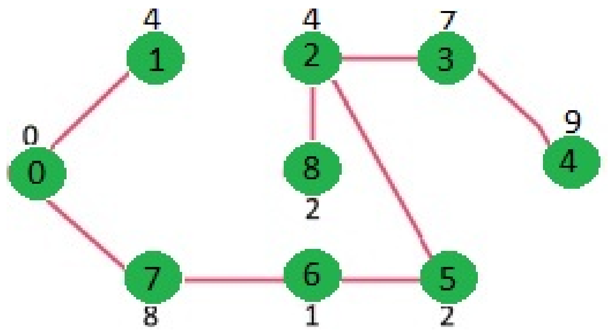
Pick the vertex with minimum key value and not already included in MST (not in mstSET). We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So mstSet now becomes {0, 1, 7}. Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (1 and 7 respectively).



Pick the vertex with minimum key value and not already included in MST (not in mstSET). Vertex 6 is picked. So mstSet now becomes {0, 1, 7, 6}. Update the key values of adjacent vertices of 6. The key value of vertex 5 and 8 are updated.



We repeat the above steps until *mstSet* includes all vertices of given graph. Finally, we get the following graph.



How to implement the above algorithm?

We use a boolean array mstSet[] to represent the set of vertices included in MST. If a value mstSet[v] is true, then vertex v is included in MST, otherwise not. Array key[] is used to store key values of all vertices. Another array parent[] to store indexes of parent nodes in MST. The parent array is the output array which is used to show the constructed MST.

C++ Java

```
// A Java program for Prim's Minimum Spanning Tree (MST) algorithm.
// The program is for adjacency matrix representation of the graph

import java.util.*;
import java.lang.*;
import java.io.*;

class MST {
    // Number of vertices in the graph
    private static final int V = 5;

    // A utility function to find the vertex with minimum key
    // value, from the set of vertices not yet included in MST
    int minKey(int key[], Boolean mstSet[])
    {
        // Initialize min value
        int min = Integer.MAX_VALUE, min_index = -1;
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
        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[][])
    {
        System.out.println("Edge \tWeight");
        for (int i = 1; i < V; i++)
            System.out.println(parent[i] + " - " + i + "\t" + graph[i][parent[i]]);
    }

    // Function to construct and print MST for a graph represented
    // using adjacency matrix representation
    void primMST(int graph[][])
    {
        // Array to store constructed MST
        int parent[] = new int[V];

        // Key values used to pick minimum weight edge in cut
        int key[] = new int[V];

        // To represent set of vertices not yet included in MST
        Boolean mstSet[] = new Boolean[V];

        // Initialize all keys as INFINITE
        for (int i = 0; i < V; i++) {
            key[i] = Integer.MAX_VALUE;
            mstSet[i] = false;
        }

        // Always include first 1st vertex in MST.
        key[0] = 0; // Make key 0 so that this vertex is
        // picked as first vertex
```

Dash
All
Articles
Videos
Problems
Quiz
Contest

```
        parent[0] = -1; // First node is always root of MST

        // The MST will have V vertices
        for (int count = 0; count < V - 1; count++) {
            // Pick thd 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] != 0 && mstSet[v] == false && graph[u][v] < key[v]) {
                    parent[v] = u;
                    key[v] = graph[u][v];
                }
        }

        // print the constructed MST
        printMST(parent, graph);
    }

    public static void main(String[] args)
    {
        /* Let us create the following graph
        2 3
        (0)--(1)--(2)
        | / \ |
        6| 8/ \5 |7
        | /     \ |
        (3)-------(4)
            9          */
        MST t = new MST();
        int graph[][] = new int[][] { { 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
        t.primMST(graph);
    }
}
```

Output

```
Edge   Weight
0 - 1    2
1 - 2    3
0 - 3    6
1 - 4    5
```

Time Complexity of the above program is O(V^2). If the input graph is represented using adjacency list, then the time complexity of Prim's algorithm can be reduced to O(E log V) with the help of binary heap.
Auxiliary Space: O(V)

Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.