Prim's Minimum Spanning Tree (MST) | Greedy Algo-5

Difficulty Level : Medium ● Last Updated : 03 Nov, 2020

We have discussed Kruskal's algorithm for Minimum Spanning Tree. Like Kruskal'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 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

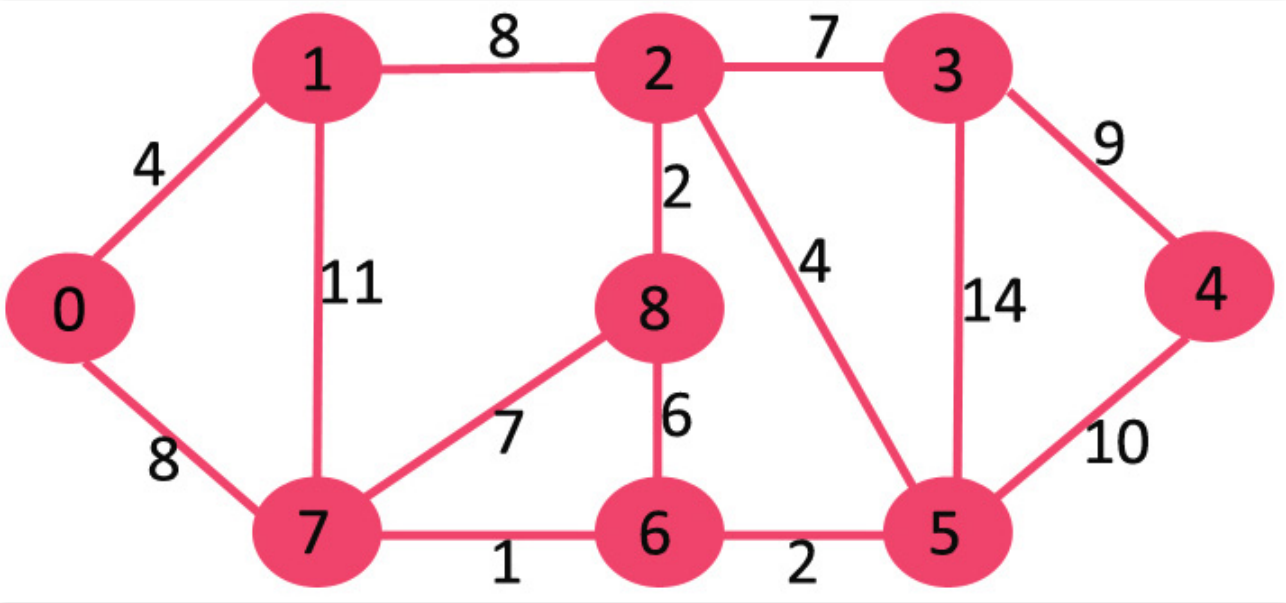
1) Create a set *mstSet* that keeps track of vertices already included in MST.

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.

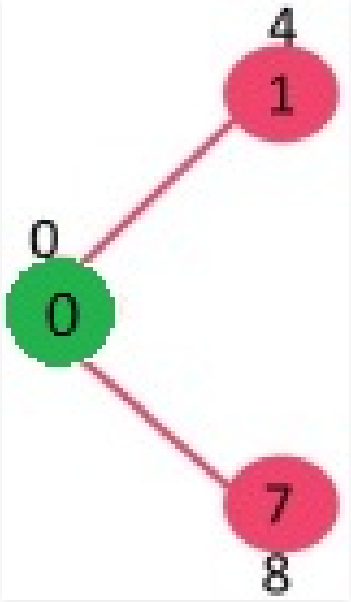
3) While mstSet doesn't include all vertices

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.

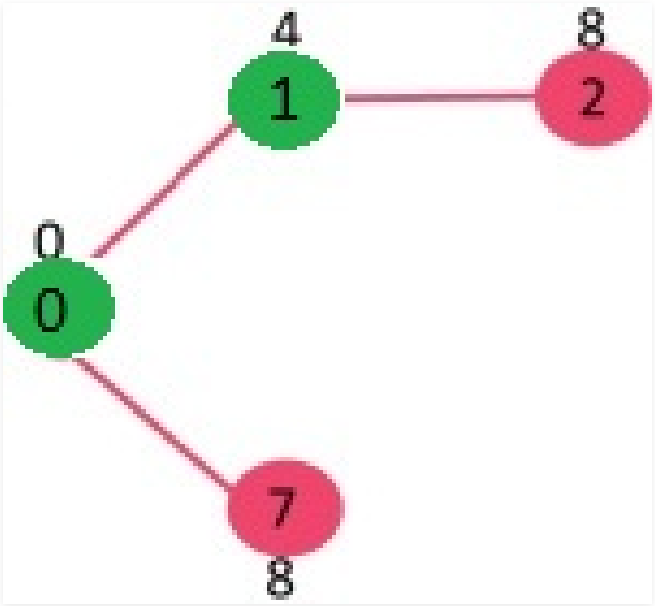
Let us understand with the following example:



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.

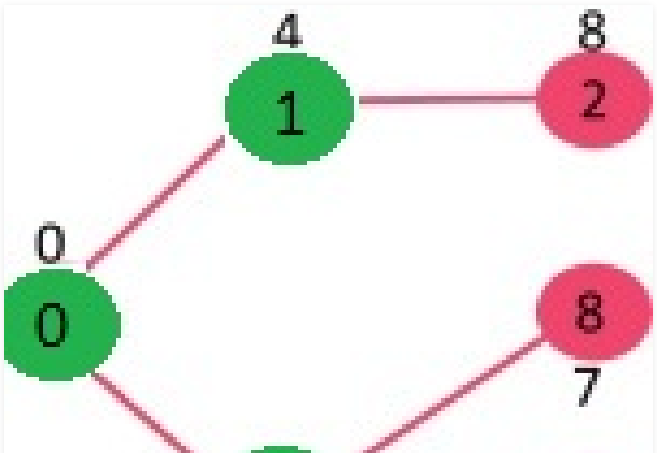


Pick the vertex with minimum key value and not already included in MST (not in mstSET). The vertex 1 is picked and added to mstSet. So mstSet now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



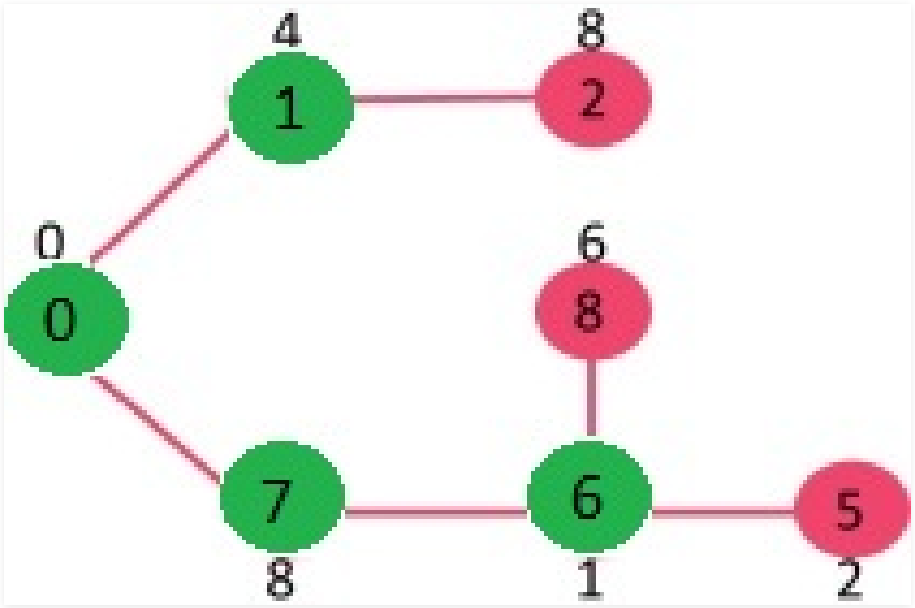
Pick the vertex with minimum key value and not already included in MST (not in mstSET). We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So mstSet now becomes {0,

becomes finite (1 and 7 respectively).

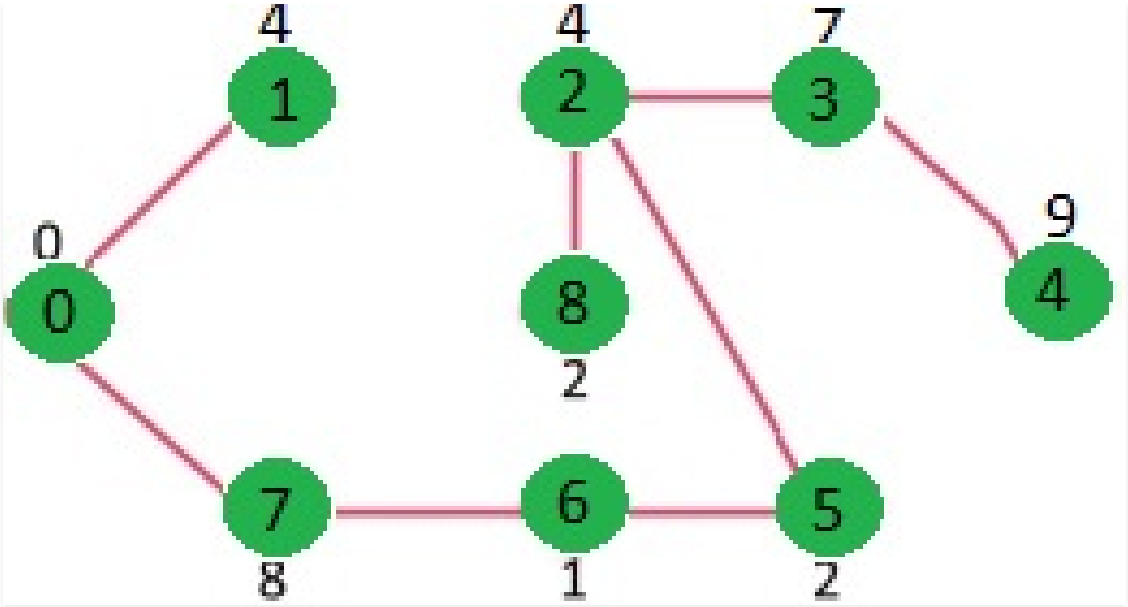


Related Articles

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



Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

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++

```
// A C++ program for Prim's Minimum
// Spanning Tree (MST) algorithm. The program is
// for adjacency matrix representation of the graph
#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;
    parent[0] = -1; // First node is always root of MST

    // The MST will have V vertices
    for (int count = 0; count < V - 1; count++)
    {
```

```
        // 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);
}

// Driver code
int main()
{
    /* Let us create the following graph
        2 3
    (0)--(1)--(2)
    | / \ |
    6| 8/ \5 |7
    | / \ |
    (3)-------(4)
            9     */
    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;
}

// This code is contributed by rathbhupendra
```

C

```c
// for adjacency matrix representation of the graph
#include <limits.h>
#include <stdbool.h>
#include <stdio.h>
// Number of vertices in the graph
#define 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[], 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[]
int printMST(int parent[], int graph[V][V])
{
    printf("Edge \tWeight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d \t%d \n", parent[i], i, graph[i][parent[i]]);
}

// 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;
```

```
    // 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);
}

// driver program to test above function
int main()
{
    /* Let us create the following graph
        2 3
    (0)--(1)--(2)
    | / \ |
    6| 8/ \5 |7
    | /     \ |
    (3)-------(4)
            9         */
    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);
```

Java

```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;

        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
```

```
        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
        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
```

```
        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);
    }
}
// This code is contributed by Aakash Hasija
```

Python

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

import sys # Library for INT_MAX

class Graph():

    def __init__(self, vertices):
        self.V = vertices
        self.graph = [[0 for column in range(vertices)]
                      for row in range(vertices)]

    # A utility function to print the constructed MST stored in parent[]
    def printMST(self, parent):
        print "Edge \tWeight"
        for i in range(1, self.V):
            print parent[i], "-", i, "\t", self.graph[i][ parent[i] ]

    # A utility function to find the vertex with
    # minimum distance value, from the set of vertices
    # not yet included in shortest path tree
    def minKey(self, key, mstSet):

        # Initilaize min value
        min = sys.maxint

        for v in range(self.V):
            if key[v] < min and mstSet[v] == False:
                min = key[v]
                min_index = v
```

```
    # represented using adjacency matrix representation
    def primMST(self):

        # Key values used to pick minimum weight edge in cut
        key = [sys.maxint] * self.V
        parent = [None] * self.V # Array to store constructed MST
        # Make key 0 so that this vertex is picked as first vertex
        key[0] = 0
        mstSet = [False] * self.V

        parent[0] = -1 # First node is always the root of

        for cout in range(self.V):

            # Pick the minimum distance vertex from
            # the set of vertices not yet processed.
            # u is always equal to src in first iteration
            u = self.minKey(key, mstSet)

            # Put the minimum distance vertex in
            # the shortest path tree
            mstSet[u] = True

            # Update dist value of the adjacent vertices
            # of the picked vertex only if the current
            # distance is greater than new distance and
            # the vertex in not in the shotest path tree
            for v in range(self.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 self.graph[u][v] > 0 and mstSet[v] == False and key[v] > self.gr
                        key[v] = self.graph[u][v]
                        parent[v] = u

        self.printMST(parent)

g = Graph(5)
g.graph = [ [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]]

g.primMST();
```

C#

```
// A C# program for Prim's Minimum
// Spanning Tree (MST) algorithm.
// The program is for adjacency
// matrix representation of the graph
using System;
class MST {

    // Number of vertices in the graph
    static int V = 5;

    // A utility function to find
    // the vertex with minimum key
    // value, from the set of vertices
    // not yet included in MST
    static int minKey(int[] key, bool[] mstSet)
    {

        // Initialize min value
        int min = int.MaxValue, min_index = -1;

        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[]
    static void printMST(int[] parent, int[, ] graph)
    {
        Console.WriteLine("Edge \tWeight");
        for (int i = 1; i < V; i++)
            Console.WriteLine(parent[i] + " - " + i + "\t" + graph[i, parent[i]]);
    }

    // Function to construct and
    // print MST for a graph represented
    // using adjacency matrix representation
    static void primMST(int[, ] graph)
    {
```

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

// To represent set of vertices
// included in MST
bool[] mstSet = new bool[V];

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

// Always include first 1st vertex in MST.
// Make key 0 so that this vertex is
// picked as first vertex
// First node is always root of MST
key[0] = 0;
parent[0] = -1;

// 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
```

```
                    key[v] = graph[u, v];
                }
        }

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

    // Driver Code
    public static void Main()
    {

        /* Let us create the following graph
        2 3
        (0)--(1)--(2)
        | / \ |
        6| 8/ \5 |7
        | / \ |
        (3)-------(4)
            9 */

        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
        primMST(graph);
    }
}

// This code is contributed by anuj_67.
```

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. Please see Prim's MST for Adjacency List Representation for more details.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Previous Next

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Properties of Minimum Spanning Tree (MST)**
01, Apr 21

05 **Boruvka's algorithm for Minimum Spanning Tree**
29, Mar 11

30, Oct 12

03 **Prim's MST for Adjacency List Representation | Greedy Algo-6**
23, Nov 12

04 **Applications of Minimum Spanning Tree Problem**
04, Mar 11

16, May 16

07 **Minimum Product Spanning Tree**
12, Dec 16

08 **Reverse Delete Algorithm for Minimum Spanning Tree**
18, Feb 17

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : vt_m, AnkurKarmakar, udkumar249, GlitchFinder, rathbhupendra, POuryaKordi, user_9781

Article Tags : Amazon, Cisco, Minimum Spanning Tree, Prim's Algorithm.MST, Samsung, Graph, Greedy

Practice Tags : Amazon, Samsung, Cisco, Greedy, Graph

Improve Article | Report Issue

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

GeeksforGeeks

5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org