

Greedy Algorithms | Set 5 (Prim's Minimum Spanning Tree (MST)) - GeeksforGeeks

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 verices), pick the minimum weight edge from the cut and include this vertex to MST Set (the set that contains already included vertices).*

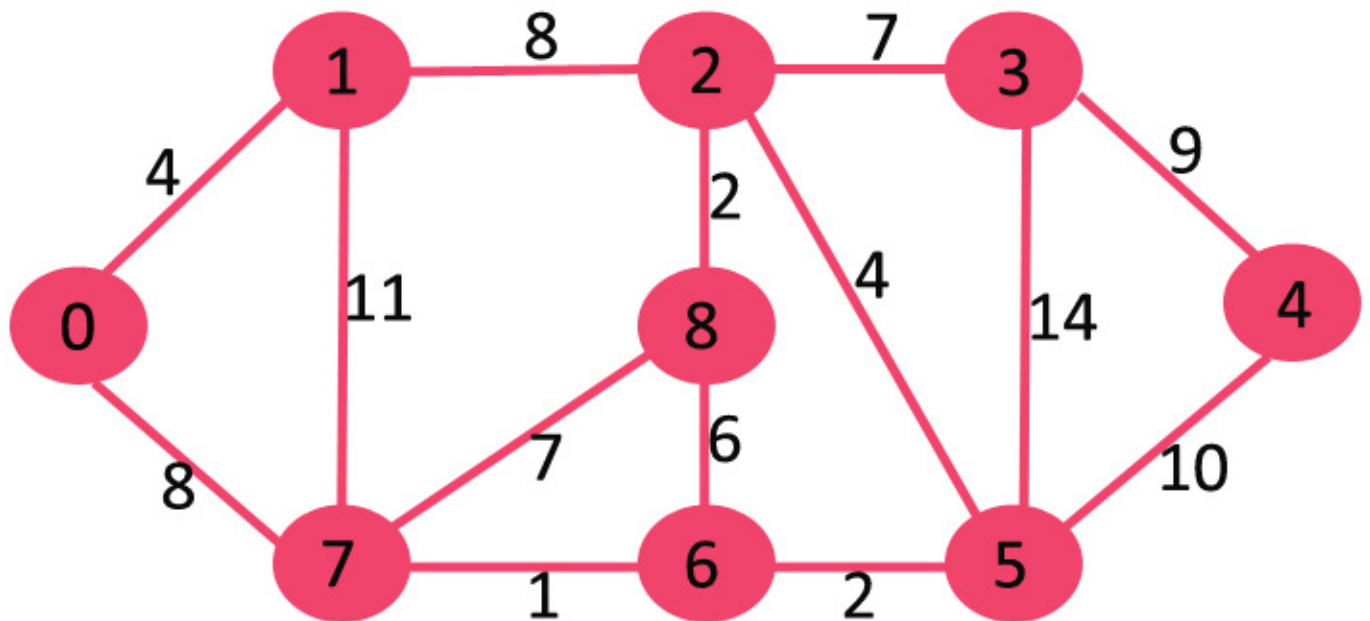
How does Prim's Algorithm Work? The idea behind Prim's algorithm is simple, a spanning tree means all vertices must be connected. So the two disjoint subsets (discussed above) of vertices must be connected to make a *Spanning Tree*. And they must be connected with the minimum weight edge to make it a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

Algorithm

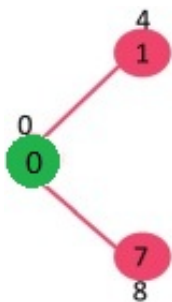
- 1) Create a set *mstSet* that keeps track of vertices already included in MST.
- 2) Assign a key value to all vertices in the input graph. Initialize all key values as INFINITE. Assign key value as 0 for the first vertex so that it is picked first.
- 3) While *mstSet* doesn't include all vertices
 -a) Pick a vertex *u* which is not there in *mstSet* and has minimum key value.
 -b) Include *u* to *mstSet*.
 -c) 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.

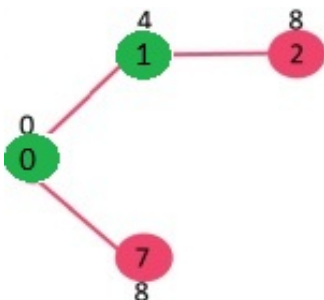
Let us understand with the following example:



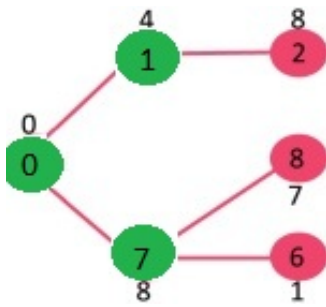
The set *mstSet* is initially empty and keys assigned to vertices are {0, INF, INF, INF, INF, INF, INF, INF, INF} where INF indicates infinite. Now pick the vertex with minimum key value. The vertex 0 is picked, include it in *mstSet*. So *mstSet* becomes {0}. After including to *mstSet*, update key 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.



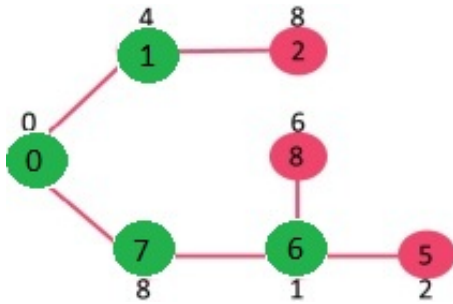
Pick the vertex with minimum key value and not already included in MST (not in *mstSet*). The vertex 1 is picked and added to *mstSet*. So *mstSet* now becomes {0, 1}. Update the key values of adjacent vertices of 1. The key value of vertex 2 becomes 8.



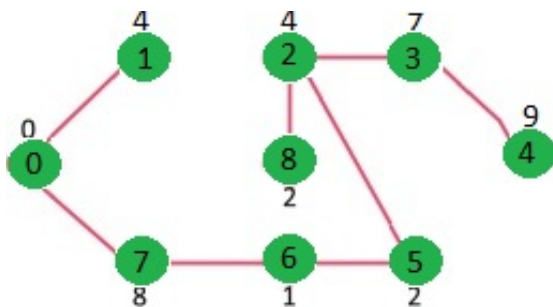
Pick the vertex with minimum key value and not already included in MST (not in *mstSet*). We can either pick vertex 7 or vertex 2, let vertex 7 is picked. So *mstSet* now becomes {0, 1, 7}. Update the key values of adjacent vertices of 7. The key value of vertex 6 and 8 becomes finite (7 and 1 respectively).



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/C++
- Java

```
// A C / C++ program for Prim's Minimum Spanning Tree (MST) algorithm. //
// The program is for adjacency matrix representation of the graph
#include <stdio.h>
#include <limits.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 n, int graph[V][V])
{
    printf("Edge   Weight\n");
    for (int i = 1; i < V; i++)
        printf("%d - %d   %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])
{
    int parent[V]; // Array to store constructed MST
    int key[V];    // Key values used to pick minimum weight edge in cut
    bool mstSet[V]; // To represent set of vertices not yet included in
MST
    // 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.
    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 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 u
            // 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, V, 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);
    return 0;
}

```