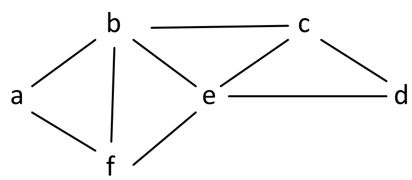
- Tree: A connected graph that does not contain any non-trivial circuit (It is circuit-free).
- Rooted Tree: A Tree in which one vertex is distinguished from the others and is called ROOT.
- Binary Tree: A rooted tree in which every vertex has at most two children.

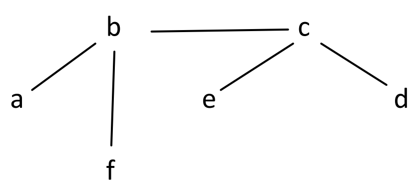
Spanning Tree:

- A spanning tree for a graph G is a subgraph of G that contains every vertex of G and is a tree.
- Every connected graph has a spanning tree.
- A graph may have more than one spanning trees.
- Any two spanning trees of a graph G have the same number of edges.
- If a graph is a tree then its ONLY spanning tree is itself.



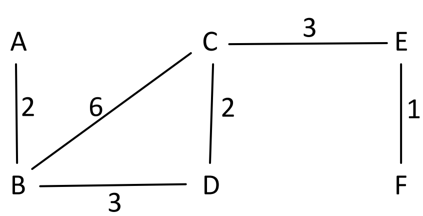
- Delete af
- Delete fe
- Delete be
- Delete ed

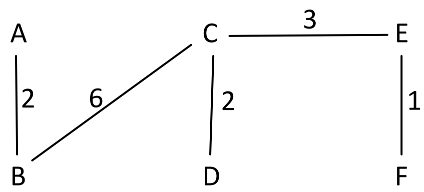
Spanning Tree:



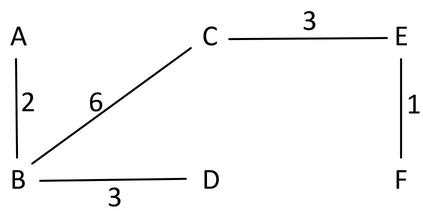
Minimal Spanning Tree:

- A spanning tree for a graph that has the least possible weight compared to all other spanning trees of the same graph.

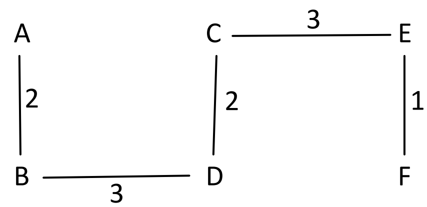




Total Weight: 14



Total weight: 15



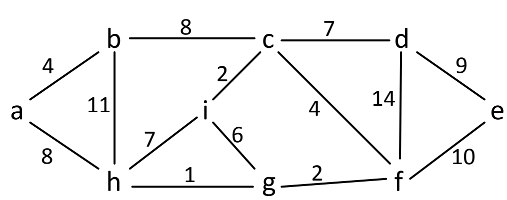
Total weight: 11 (Minimal Spanning Tree)

Finding Minimal Spanning Tree using Prim's Algorithm:

- A greedy algorithm.
- Finds a minimum spanning tree for a weighted undirected graph.

Step 1: Remove all loops.
Step 2: Remove all parallel edges and keep the edge that has least weight.

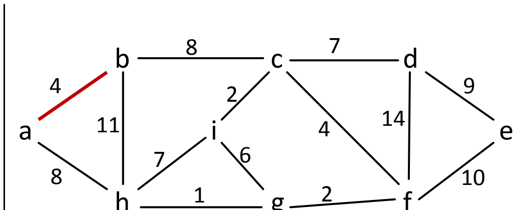
EXAMPLE 1:



Starting Point: a
Options for a:

- ab: 4, ah: 8

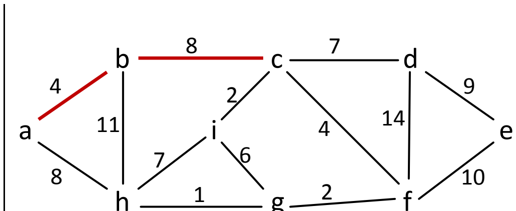
Select the smaller value: 4 (ab)



Current Point: b
Options for a, b:

- ah: 8
- bh: 11, bc: 8

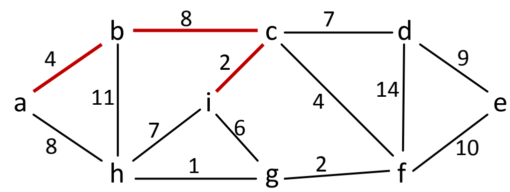
Select the smaller value: 8 (bc)



Current Point: c
Options for a, b, c:

- ah: 8
- bh: 11
- ci: 2, cd: 7, cf: 4

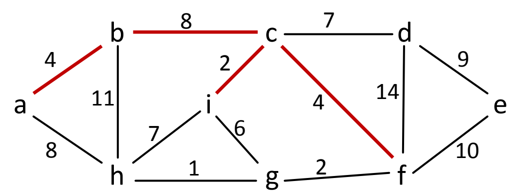
Select the smallest value: 2 (ci)



Current Point: i
Options for a, b, c, i:

- ah: 8
- bh: 11
- cd: 7, cf: 4
- ih: 7, ig: 6

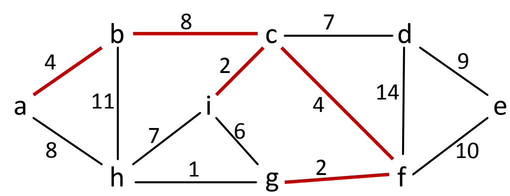
Select the smallest value: 4 (cf)



Current Point: f
Options for a, b, c, i, f:

- ah: 8
- bh: 11
- cd: 7
- ih: 7
- fd: 14, fe: 10, fg: 2

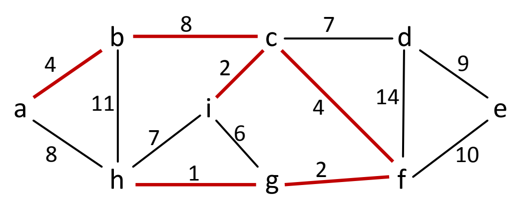
Select the smallest value: 2 (fg)



Current Point: g
Options for a, b, c, i, f, g:

- cd: 7
- fd: 14, fe: 10
- gh: 1

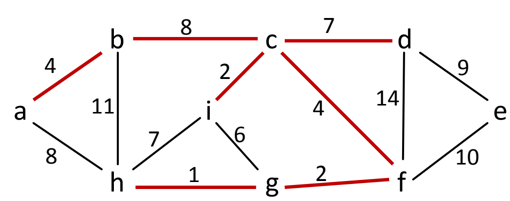
Select the smallest value: 1 (gh)



Current Point: d
Options for a, b, c, i, f, g,d:

- cd: 7
- fd: 14, fe: 10
- de: 9

Select the smallest value: 7 (cd)

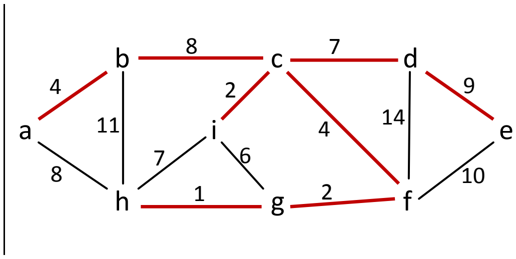


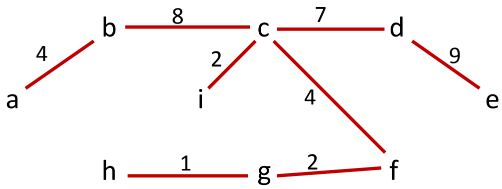
Current Point: e

Options for a, b, c, i, f, g,d,e:

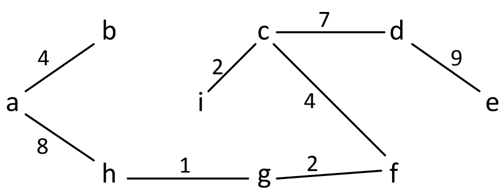
- fe: 10
- de: 9

Select the smallest value: 9 (de)



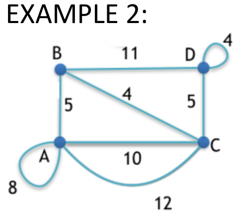


Total weight: 37

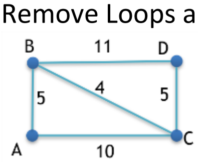


Total weight: 37

EXAMPLE 2:



Remove Loops and Parallel Edges:



Create a Table:

- Place the starting vertex in Row 1.
- Put 0 in cells having same row and column name.
- Find the edges that directly connect two vertices and fill the table with the weight of the edge.
- If no direct edge exists then fill the cell with infinity.

	A	B	C	D
A	0	5	10	∞
B	5	0	4	11
C	10	4	0	5
D	∞	11	5	0

Smallest unmarked value in row A: 5 (AB)
So mark both AB and BA

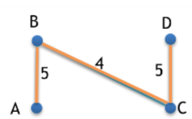
	A	B	C	D
A	0	**5**	10	∞
B	**5**	0	4	11
C	10	4	0	5
D	∞	11	5	0

Smallest unmarked value in row A and row B: 4 (BC)
So mark both BC and CB

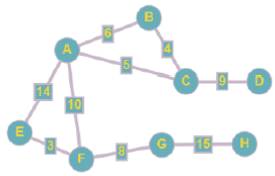
	A	B	C	D
A	0	**5**	10	∞
B	**5**	0	**4**	11
C	10	**4**	0	5
D	∞	11	5	0

Smallest unmarked value in row A, row B and row C: 5 (CD)
So mark both CD and DC

	A	B	C	D
A	0	**5**	10	∞
B	**5**	0	**4**	11
C	10	**4**	0	**5**
D	∞	11	**5**	0



Prim's Algorithm



1. Starting Node: E

	A	B	C	D	E	F	G	H
K	∞	∞	∞	∞	0	∞	∞	∞
P					Ø			

2. Minimum : 0 (E)
   Neighbors :
   - A, Weight: 14 < Existing key
   - F, Weight: 3 < Existing key

	A	B	C	D	E	F	G	H
K	14	∞	∞	∞	0	3	∞	∞
P	E				Ø	E		

3. Minimum : 3 (F)
   Neighbors :
   - A, Weight: 10 < Existing key
   - G, Weight: 8 < Existing key

	A	B	C	D	E	F	G	H
K	10	∞	∞	∞	0	3	8	∞
P	F				Ø	E	F	

4. Minimum : 8 (G)
   Neighbors :
   - F, Weight: 8 > Existing key (Discard)
   - H, Weight: 15 < Existing key

	A	B	C	D	E	F	G	H
K	10	∞	∞	∞	0	3	8	15
P	F				Ø	E	F	G

5. Minimum : 10 (A)
   Neighbors :
   - E, Weight: 14 > Existing key (Discard)
   - F, Weight: 10 > Existing key (Discard)
   - B, Weight: 6 < Existing key
   - C, Weight: 5 < Existing key

	A	B	C	D	E	F	G	H
K	10	6	5	∞	0	3	8	15
P	F	A	A		Ø	E	F	G

6. Minimum : 5 (C)
   Neighbors :
   - A, Weight: 5 < Existing key
   - B, Weight: 4 < Existing key
   - D, Weight: 9 < Existing key

	A	B	C	D	E	F	G	H
K	5	4	5	9	0	3	8	15
P	C	C	A	C	Ø	E	F	G

7. Minimum : 4 (B)
   Neighbors :
   - A, Weight: 6 > Existing key (Discard)
   - C, Weight: 4 < Existing key

	A	B	C	D	E	F	G	H
K	5	4	4	9	0	3	8	15
P	C	C	B	C	Ø	E	F	G

8. Minimum : 9 (D)
   Neighbors :
   - C, Weight: 9 > Existing key (Discard)

	A	B	C	D	E	F	G	H
K	5	4	4	9	0	3	8	15
P	C	C	B	C	Ø	E	F	G

9. Minimum : 15 (H)
   Neighbors :
   - G, Weight: 15 > Existing key (Discard)

	A	B	C	D	E	F	G	H
K	5	4	4	9	0	3	8	15
P	C	C	B	C	Ø	E	F	G

```
#include <bits/stdc++.h>
using namespace std;

#define V 5  // Number of vertices in the graph

// 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[]
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])
{
   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.
   // 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++)
    {
        // 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()
{
    /* 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 } };

    primMST(graph);  // Print the solution

    return 0;
}
```