Spanning tree

Spanning tree and the minimum spanning tree. But before moving directly towards the spanning tree, let's first see a brief description of the graph and its types.

Graph

A graph can be defined as a group of vertices and edges to connect these vertices. The types of graphs are given as follows -

* **Undirected graph:** An undirected graph is a graph in which all the edges do not point to any particular direction, i.e., they are not unidirectional; they are bidirectional. It can also be defined as a graph with a set of V vertices and a set of E edges, each edge connecting two different vertices.
* **Connected graph:** A connected graph is a graph in which a path always exists from a vertex to any other vertex. A graph is connected if we can reach any vertex from any other vertex by following edges in either direction.
* **Directed graph:** Directed graphs are also known as digraphs. A graph is a directed graph (or digraph) if all the edges present between any vertices or nodes of the graph are directed or have a defined direction.

Now, let's move towards the topic spanning tree.

What is a spanning tree?

A spanning tree can be defined as the subgraph of an undirected connected graph. It includes all the vertices along with the least possible number of edges. If any vertex is missed, it is not a spanning tree. A spanning tree is a subset of the graph that does not have cycles, and it also cannot be disconnected.

68M

1.1K

Hello Java Program for Beginners

A spanning tree consists of (n-1) edges, where 'n' is the number of vertices (or nodes). Edges of the spanning tree may or may not have weights assigned to them. All the possible spanning trees created from the given graph G would have the same number of vertices, but the number of edges in the spanning tree would be equal to the number of vertices in the given graph minus 1.

A complete undirected graph can have **nn-2** number of spanning trees where **n** is the number of vertices in the graph. Suppose, if **n = 5**, the number of maximum possible spanning trees would be **55-2 = 125.**

Applications of the spanning tree

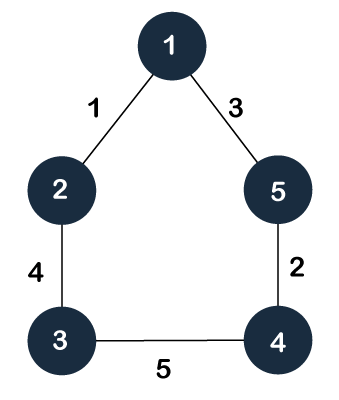
Basically, a spanning tree is used to find a minimum path to connect all nodes of the graph. Some of the common applications of the spanning tree are listed as follows -

* Cluster Analysis
* Civil network planning
* Computer network routing protocol

Now, let's understand the spanning tree with the help of an example.

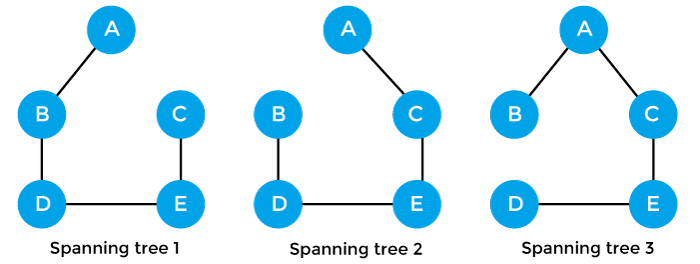
Example of Spanning tree

Suppose the graph be -



As discussed above, a spanning tree contains the same number of vertices as the graph, the number of vertices in the above graph is 5; therefore, the spanning tree will contain 5 vertices. The edges in the spanning tree will be equal to the number of vertices in the graph minus 1. So, there will be 4 edges in the spanning tree.

Some of the possible spanning trees that will be created from the above graph are given as follows -



Properties of spanning-tree

Some of the properties of the spanning tree are given as follows -

* There can be more than one spanning tree of a connected graph G.
* A spanning tree does not have any cycles or loop.
* A spanning tree is **minimally connected,** so removing one edge from the tree will make the graph disconnected.
* A spanning tree is **maximally acyclic,** so adding one edge to the tree will create a loop.
* There can be a maximum **nn-2** number of spanning trees that can be created from a complete graph.
* A spanning tree has **n-1** edges, where 'n' is the number of nodes.
* If the graph is a complete graph, then the spanning tree can be constructed by removing maximum (e-n+1) edges, where 'e' is the number of edges and 'n' is the number of vertices.

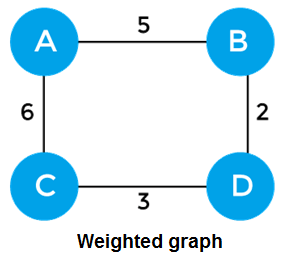
So, a spanning tree is a subset of connected graph G, and there is no spanning tree of a disconnected graph.

Minimum Spanning tree

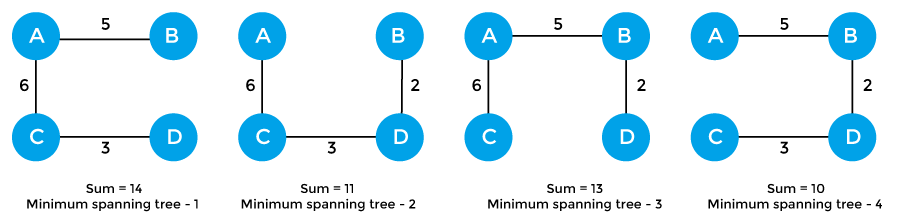
A minimum spanning tree can be defined as the spanning tree in which the sum of the weights of the edge is minimum. The weight of the spanning tree is the sum of the weights given to the edges of the spanning tree. In the real world, this weight can be considered as the distance, traffic load, congestion, or any random value.

Example of minimum spanning tree

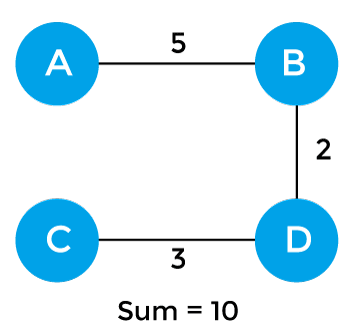
Let's understand the minimum spanning tree with the help of an example.



The sum of the edges of the above graph is 16. Now, some of the possible spanning trees created from the above graph are -



So, the minimum spanning tree that is selected from the above spanning trees for the given weighted graph is -



Applications of minimum spanning tree

The applications of the minimum spanning tree are given as follows -

* Minimum spanning tree can be used to design water-supply networks, telecommunication networks, and electrical grids.
* It can be used to find paths in the map.

Algorithms for Minimum spanning tree

A minimum spanning tree can be found from a weighted graph by using the algorithms given below -

* Prim's Algorithm
* Kruskal's Algorithm

Let's see a brief description of both of the algorithms listed above.

**Prim's algorithm -** It is a greedy algorithm that starts with an empty spanning tree. It is used to find the minimum spanning tree from the graph. This algorithm finds the subset of edges that includes every vertex of the graph such that the sum of the weights of the edges can be minimized.

Prim's algorithm starts with the single node and explores all the adjacent nodes with all the connecting edges at every step. The edges with the minimal weights causing no cycles in the graph got selected.

How does the prim's algorithm work?

Prim's algorithm is a greedy algorithm that starts from one vertex and continue to add the edges with the smallest weight until the goal is reached. The steps to implement the prim's algorithm are given as follows -

* First, we have to initialize an MST with the randomly chosen vertex.
* Now, we have to find all the edges that connect the tree in the above step with the new vertices. From the edges found, select the minimum edge and add it to the tree.
* Repeat step 2 until the minimum spanning tree is formed.

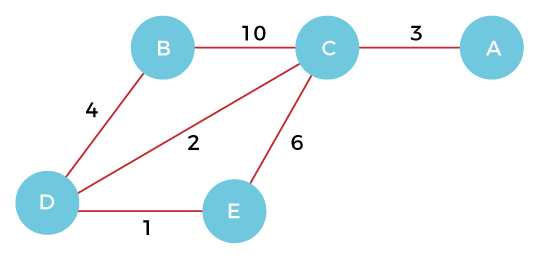
The applications of prim's algorithm are -

* Prim's algorithm can be used in network designing.
* It can be used to make network cycles.
* It can also be used to lay down electrical wiring cables.

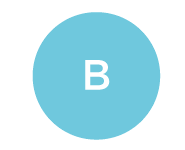
Example of prim's algorithm

Now, let's see the working of prim's algorithm using an example. It will be easier to understand the prim's algorithm using an example.

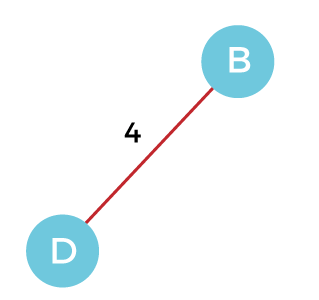
Suppose, a weighted graph is -



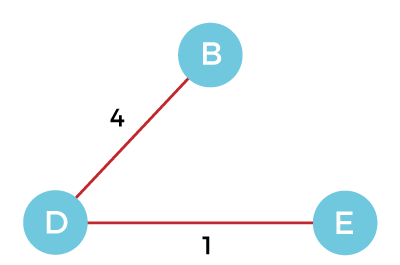
**Step 1 -** First, we have to choose a vertex from the above graph. Let's choose B.



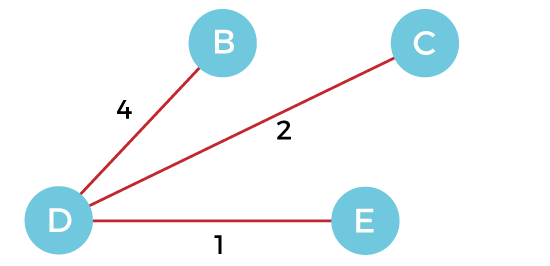
**Step 2 -** Now, we have to choose and add the shortest edge from vertex B. There are two edges from vertex B that are B to C with weight 10 and edge B to D with weight 4. Among the edges, the edge BD has the minimum weight. So, add it to the MST.



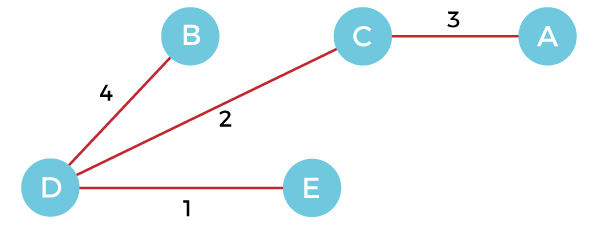
**Step 3 -** Now, again, choose the edge with the minimum weight among all the other edges. In this case, the edges DE and CD are such edges. Add them to MST and explore the adjacent of C, i.e., E and A. So, select the edge DE and add it to the MST.



**Step 4 -** Now, select the edge CD, and add it to the MST.



**Step 5 -** Now, choose the edge CA. Here, we cannot select the edge CE as it would create a cycle to the graph. So, choose the edge CA and add it to the MST.



So, the graph produced in step 5 is the minimum spanning tree of the given graph. The cost of the MST is given below -

Cost of MST = 4 + 2 + 1 + 3 = 10 units.

Algorithm

1. Step 1: Select a starting vertex
2. Step 2: Repeat Steps 3 and 4 until there are fringe vertices
3. Step 3: Select an edge 'e' connecting the tree vertex and fringe vertex that has minimum weight
4. Step 4: Add the selected edge and the vertex to the minimum spanning tree T
5. [END OF LOOP]
6. Step 5: EXIT

Complexity of Prim's algorithm

Now, let's see the time complexity of Prim's algorithm. The running time of the prim's algorithm depends upon using the data structure for the graph and the ordering of edges. Below table shows some choices -

* **Time Complexity**

|  |  |
| --- | --- |
| **Data structure used for the minimum edge weight** | **Time Complexity** |
| Adjacency matrix, linear searching | O(|V|2) |
| Adjacency list and binary heap | O(|E| log |V|) |
| Adjacency list and Fibonacci heap | O(|E|+ |V| log |V|) |

Prim's algorithm can be simply implemented by using the adjacency matrix or adjacency list graph representation, and to add the edge with the minimum weight requires the linearly searching of an array of weights. It requires O(|V|2) running time. It can be improved further by using the implementation of heap to find the minimum weight edges in the inner loop of the algorithm.

The time complexity of the prim's algorithm is O(E logV) or O(V logV), where E is the no. of edges, and V is the no. of vertices.

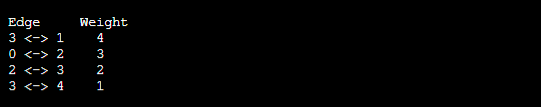
Implementation of Prim's algorithm

Now, let's see the implementation of prim's algorithm.

**Program:** Write a program to implement prim's algorithm in C language.

1. #include <stdio.h>
2. #include <limits.h>
3. #define vertices 5  /\*Define the number of vertices in the graph\*/
4. /\* create minimum\_key() method for finding the vertex that has minimum key-value and that is not added in MST yet \*/
5. **int** minimum\_key(**int** k[], **int** mst[])
6. {
7. **int** minimum  = INT\_MAX, min,i;
9. /\*iterate over all vertices to find the vertex with minimum key-value\*/
10. **for** (i = 0; i < vertices; i++)
11. **if** (mst[i] == 0 && k[i] < minimum )
12. minimum = k[i], min = i;
13. **return** min;
14. }
15. /\* create prim() method for constructing and printing the MST.
16. The g[vertices][vertices] is an adjacency matrix that defines the graph for MST.\*/
17. **void** prim(**int** g[vertices][vertices])
18. {
19. /\* create array of size equal to total number of vertices for storing the MST\*/
20. **int** parent[vertices];
21. /\* create k[vertices] array for selecting an edge having minimum weight\*/
22. **int** k[vertices];
23. **int** mst[vertices];
24. **int** i, count,edge,v; /\*Here 'v' is the vertex\*/
25. **for** (i = 0; i < vertices; i++)
26. {
27. k[i] = INT\_MAX;
28. mst[i] = 0;
29. }
30. k[0] = 0; /\*It select as first vertex\*/
31. parent[0] = -1;   /\* set first value of parent[] array to -1 to make it root of MST\*/
32. **for** (count = 0; count < vertices-1; count++)
33. {
34. /\*select the vertex having minimum key and that is not added in the MST yet from the set of vertices\*/
35. edge = minimum\_key(k, mst);
36. mst[edge] = 1;
37. **for** (v = 0; v < vertices; v++)
38. {
39. **if** (g[edge][v] && mst[v] == 0 && g[edge][v] <  k[v])
40. {
41. parent[v]  = edge, k[v] = g[edge][v];
42. }
43. }
44. }
45. /\*Print the constructed Minimum spanning tree\*/
46. printf("\n Edge \t  Weight\n");
47. **for** (i = 1; i < vertices; i++)
48. printf(" %d <-> %d    %d \n", parent[i], i, g[i][parent[i]]);
50. }
51. **int** main()
52. {
53. **int** g[vertices][vertices] = {{0, 0, 3, 0, 0},
54. {0, 0, 10, 4, 0},
55. {3, 10, 0, 2, 6},
56. {0, 4, 2, 0, 1},
57. {0, 0, 6, 1, 0},
58. };
59. prim(g);
60. **return** 0;
61. }

**Output**



**Kruskal's algorithm -** This algorithm is also used to find the minimum spanning tree for a connected weighted graph. Kruskal's algorithm also follows greedy approach, which finds an optimum solution at every stage instead of focusing on a global optimum.

How does Kruskal's algorithm work?

In Kruskal's algorithm, we start from edges with the lowest weight and keep adding the edges until the goal is reached. The steps to implement Kruskal's algorithm are listed as follows -

* First, sort all the edges from low weight to high.
* Now, take the edge with the lowest weight and add it to the spanning tree. If the edge to be added creates a cycle, then reject the edge.
* Continue to add the edges until we reach all vertices, and a minimum spanning tree is created.

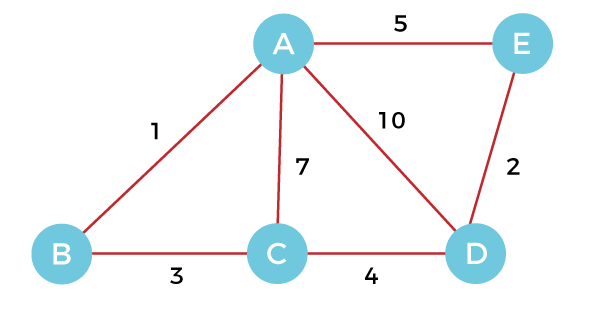
The applications of Kruskal's algorithm are -

* Kruskal's algorithm can be used to layout electrical wiring among cities.
* It can be used to lay down LAN connections.

Example of Kruskal's algorithm

Now, let's see the working of Kruskal's algorithm using an example. It will be easier to understand Kruskal's algorithm using an example.

Suppose a weighted graph is -



The weight of the edges of the above graph is given in the below table -

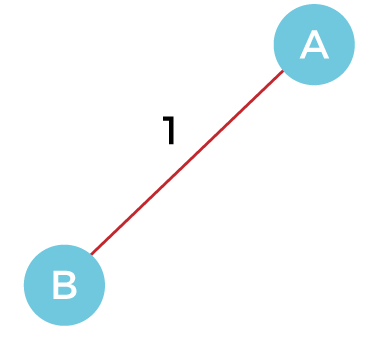
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Edge** | AB | AC | AD | AE | BC | CD | DE |
| **Weight** | 1 | 7 | 10 | 5 | 3 | 4 | 2 |

Now, sort the edges given above in the ascending order of their weights.

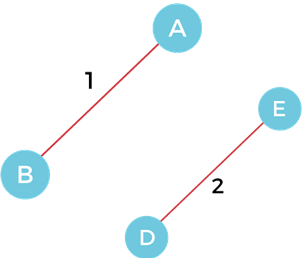
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Edge** | AB | DE | BC | CD | AE | AC | AD |
| **Weight** | 1 | 2 | 3 | 4 | 5 | 7 | 10 |

Now, let's start constructing the minimum spanning tree.

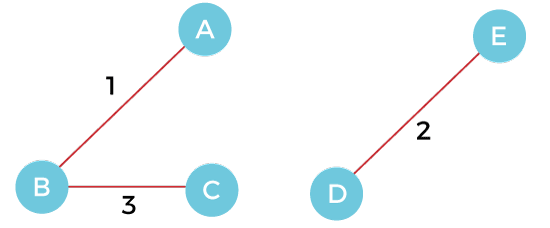
**Step 1 -** First, add the edge **AB** with weight **1** to the MST.



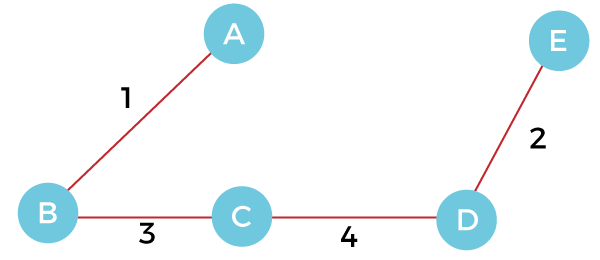
**Step 2 -** Add the edge **DE** with weight **2** to the MST as it is not creating the cycle.



**Step 3 -** Add the edge **BC** with weight **3** to the MST, as it is not creating any cycle or loop.



**Step 4 -** Now, pick the edge **CD** with weight **4** to the MST, as it is not forming the cycle.

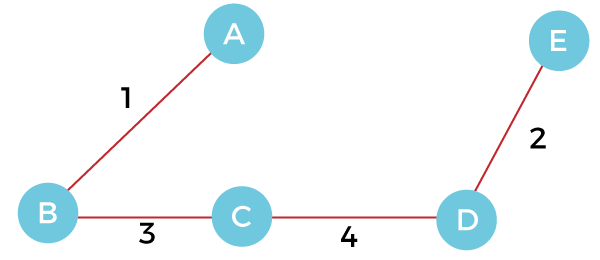


**Step 5 -** After that, pick the edge **AE** with weight **5.** Including this edge will create the cycle, so discard it.

**Step 6 -** Pick the edge **AC** with weight **7.** Including this edge will create the cycle, so discard it.

**Step 7 -** Pick the edge **AD** with weight **10.** Including this edge will also create the cycle, so discard it.

So, the final minimum spanning tree obtained from the given weighted graph by using Kruskal's algorithm is -



The cost of the MST is = AB + DE + BC + CD = 1 + 2 + 3 + 4 = 10.

Now, the number of edges in the above tree equals the number of vertices minus 1. So, the algorithm stops here.

Algorithm

1. Step 1: Create a forest F in such a way that every vertex of the graph is a separate tree.
2. Step 2: Create a set E that contains all the edges of the graph.
3. Step 3: Repeat Steps 4 and 5 **while** E is NOT EMPTY and F is not spanning
4. Step 4: Remove an edge from E with minimum weight
5. Step 5: IF the edge obtained in Step 4 connects two different trees, then add it to the forest F
6. (**for** combining two trees into one tree).
7. ELSE
8. Discard the edge
9. Step 6: END

Complexity of Kruskal's algorithm

Now, let's see the time complexity of Kruskal's algorithm.

* **TimeComplexity**  
  The time complexity of Kruskal's algorithm is O(E logE) or O(V logV), where E is the no. of edges, and V is the no. of vertices.

Implementation of Kruskal's algorithm

Now, let's see the implementation of kruskal's algorithm.

**Program:** Write a program to implement kruskal's algorithm in C++.

1. #include <iostream>
2. #include <algorithm>
3. **using** **namespace** std;
4. **const** **int** MAX = 1e4 + 5;
5. **int** id[MAX], nodes, edges;
6. pair <**long** **long**, pair<**int**, **int**> > p[MAX];
7. **void** init()
8. {
9. **for**(**int** i = 0;i < MAX;++i)
10. id[i] = i;
11. }
12. **int** root(**int** x)
13. {
14. **while**(id[x] != x)
15. {
16. id[x] = id[id[x]];
17. x = id[x];
18. }
19. **return** x;
20. }
21. **void** union1(**int** x, **int** y)
22. {
23. **int** p = root(x);
24. **int** q = root(y);
25. id[p] = id[q];
26. }
27. **long** **long** kruskal(pair<**long** **long**, pair<**int**, **int**> > p[])
28. {
29. **int** x, y;
30. **long** **long** cost, minimumCost = 0;
31. **for**(**int** i = 0;i < edges;++i)
32. {
33. x = p[i].second.first;
34. y = p[i].second.second;
35. cost = p[i].first;
36. **if**(root(x) != root(y))
37. {
38. minimumCost += cost;
39. union1(x, y);
40. }
41. }
42. **return** minimumCost;
43. }
44. **int** main()
45. {
46. **int** x, y;
47. **long** **long** weight, cost, minimumCost;
48. init();
49. cout <<"Enter Nodes and edges";
50. cin >> nodes >> edges;
51. **for**(**int** i = 0;i < edges;++i)
52. {
53. cout<<"Enter the value of X, Y and edges";
54. cin >> x >> y >> weight;
55. p[i] = make\_pair(weight, make\_pair(x, y));
56. }
57. sort(p, p + edges);
58. minimumCost = kruskal(p);
59. cout <<"Minimum cost is "<< minimumCost << endl;
60. **return** 0;
61. }

**Output**

