Kruskal's algorithm

Before moving directly towards the algorithm, we should first understand the basic terms such as spanning tree and minimum spanning tree.

Spanning tree - A spanning tree is the subgraph of an undirected connected graph.

Minimum Spanning tree - 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.

Now, let's start with the main topic.

Kruskal's Algorithm is used to find the minimum spanning tree for a connected weighted graph. The main target of the algorithm is to find the subset of edges by using which we can traverse every vertex of the graph. It follows the greedy approach that 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.

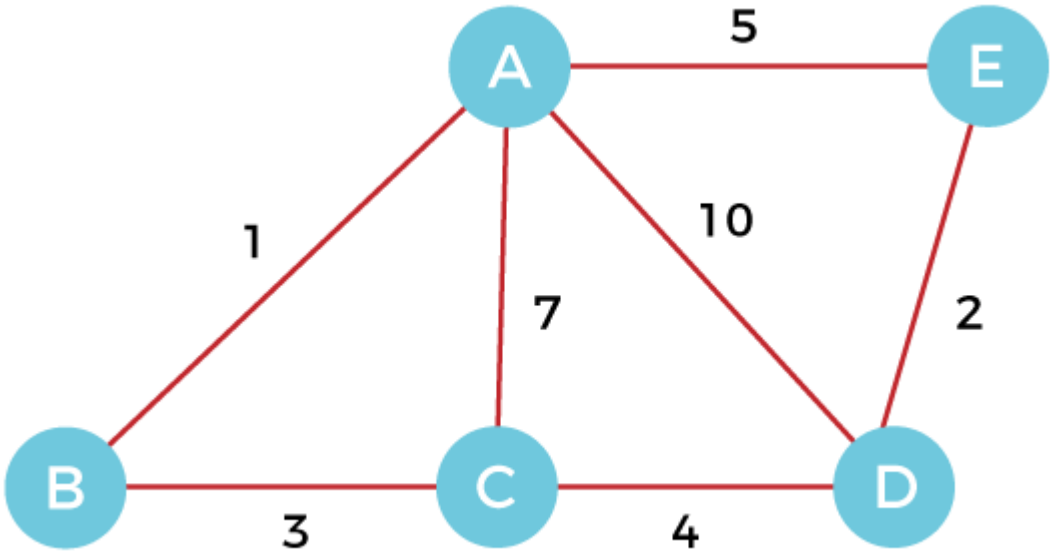
The applications of Kruskal's algorithm are -

- 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 -

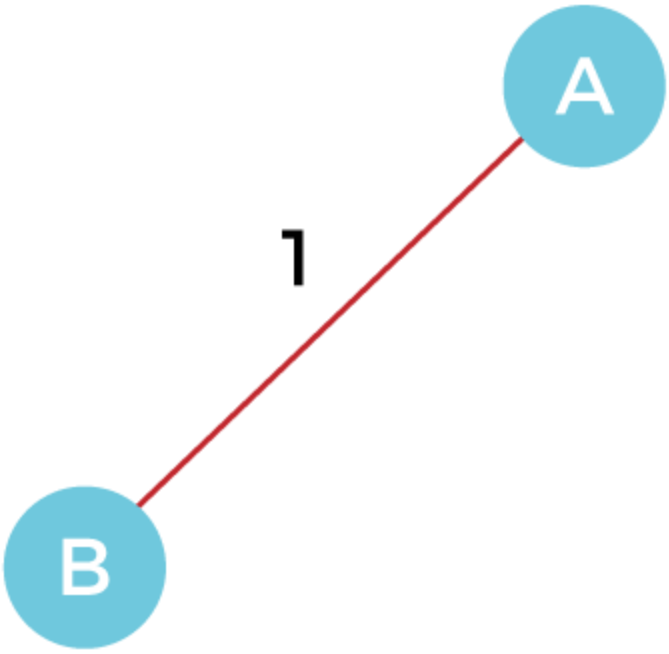
Edge	AB	AC	AD	AE	BC	CD	DE
Weight	1	7	10	5	3	4	2

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

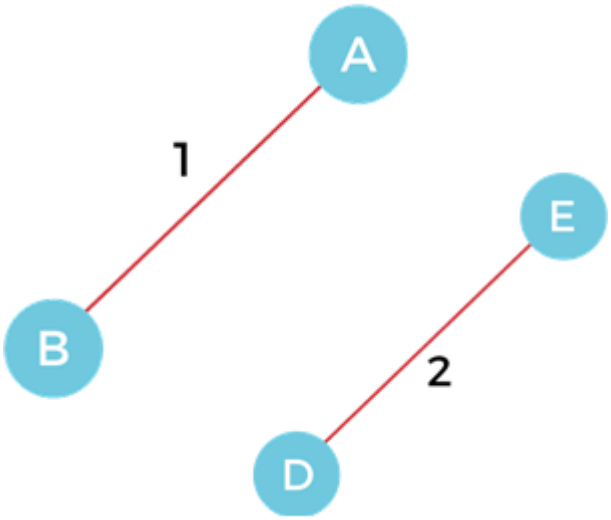
Edge	AB	DE	BC	CD	AE	AC	AD
Weight	1	2	3	4	5	7	10

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

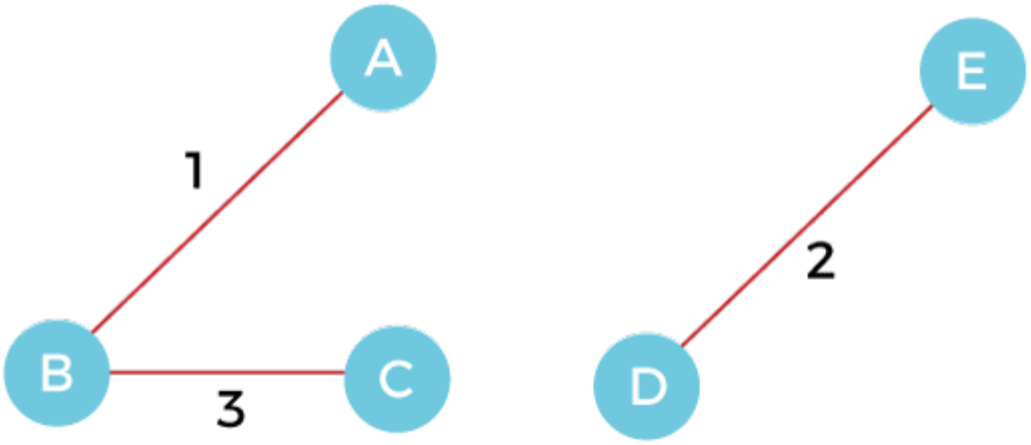
Step 1 - First, add the edge **AB** with weight **1** to the MST.



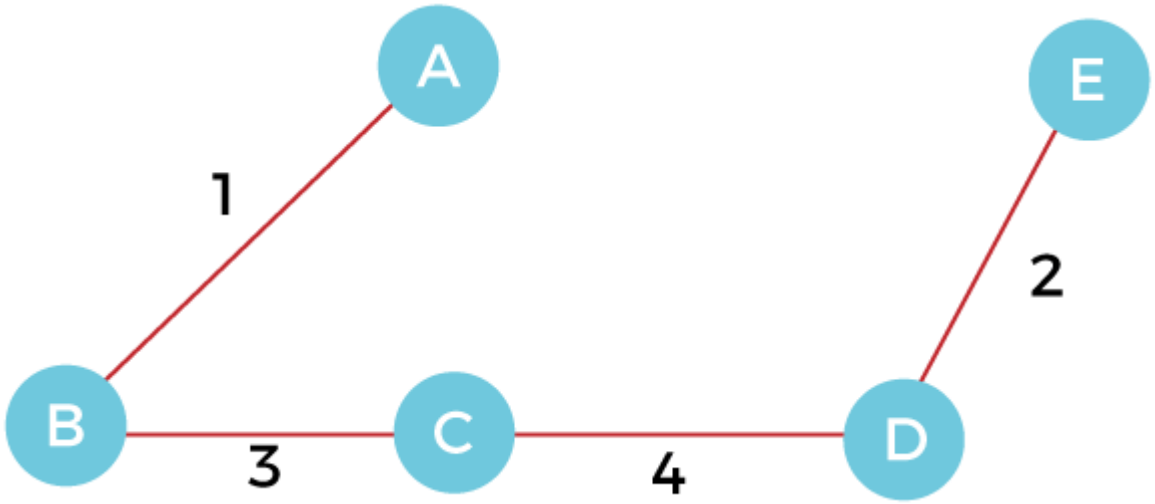
Step 2 - Add the edge **DE** with weight **2** to the MST as it is not creating the cycle.



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.

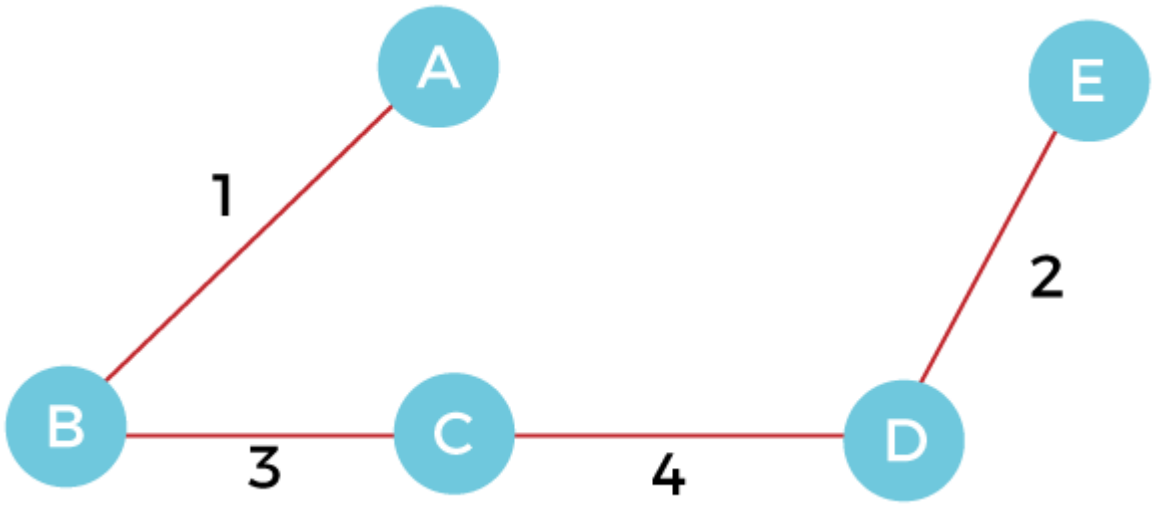


Step 5 - After that, pick the edge **AE** with weight **5.** Including this edge will create the cycle, so discard it.

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 f orest 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.

- **Time Complexity**
  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++.

```
#include <iostream>
#include <algorithm>
using namespace std;
const int MAX = 1e4 + 5;
int id[MAX], nodes, edges;
pair <long long, pair<int, int> > p[MAX];
void init()
{
    for(int i = 0;i < MAX;++i)
        id[i] = i;
}
int root(int x)
{
    while(id[x] != x)
    {
        id[x] = id[id[x]];
        x = id[x];
    }
    return x;
}
void union1(int x, int y)
{
    int p = root(x);
    int q = root(y);
    id[p] = id[q];
}
long long kruskal(pair<long long, pair<int, int> > p[])
{
    int x, y;
    long long cost, minimumCost = 0;
    for(int i = 0;i < edges;++i)
    {
        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;
        if(root(x) != root(y))
        {
            minimumCost += cost;
            union1(x, y);
        }
    }
    return minimumCost;
}
int main()
{
    int x, y;
    long long weight, cost, minimumCost;
    init();
    cout <<"Enter Nodes and edges";
    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

```
Enter Nodes and edges 5 7
Enter the value of X, Y and edges 1 2 1
Enter the value of X, Y and edges 1 3 7
Enter the value of X, Y and edges 1 4 10
Enter the value of X, Y and edges 1 5 5
Enter the value of X, Y and edges 2 3 3
Enter the value of X, Y and edges 3 4 4
Enter the value of X, Y and edges 4 5 2
Minimum cost is 10
```