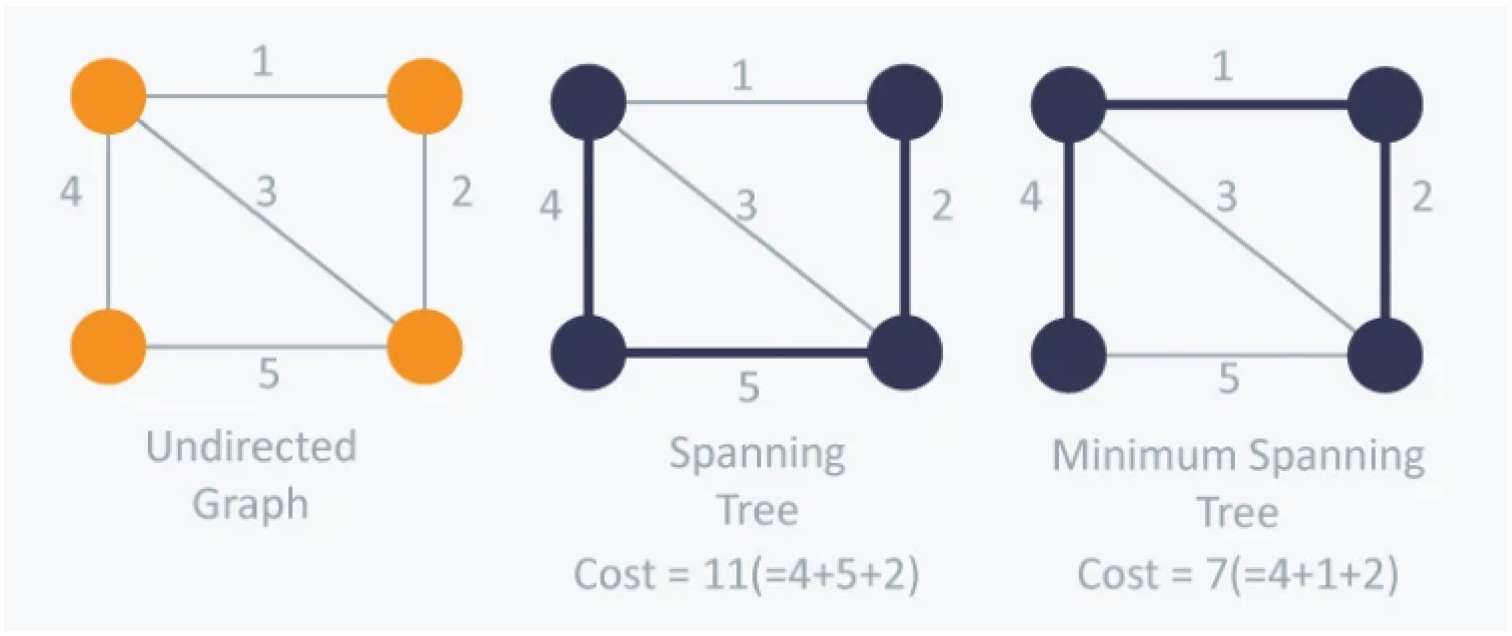


Kruskal's Algorithm and Prim's minimum spanning tree algorithm are two popular algorithms to find the minimum spanning trees.

Kruskal's algorithm uses the greedy approach for finding a minimum spanning tree.

Kruskal's algorithm treats every node as an independent tree and connects one with another only if it has the lowest cost compared to all other options available.

Step to Kruskal's algorithm:

- Sort the graph edges with respect to their weights.
- Start adding edges to the minimum spanning tree from the edge with the smallest weight until the edge of the largest weight.
- Only add edges which don't form a cycle—edges which connect only disconnected components.

Or as a simpler explanation,

Step 1 – Remove all loops and parallel edges

Step 2 – Arrange all the edges in ascending order of cost

Step 3 – Add edges with least weight

But how do you check whether twovertices are connected or not? That's where the real-life example of Disjoint Sets come into use.

Kruskal's algorithm example in detail

I am sure very few of you would be working for acable network company, so let's make the Kruskal's minimum spanning tree algorithm problem more relatable.

On your trip to Venice, you plan to visit all the important world heritage sites but are short on time. To make your itinerary work,you decide to use Kruskal's algorithm using disjoint sets.

Here is amap of Venice.

SHARE

Let’s simplify the map by converting it into a graph as below and naming important locations on the map with lettersand distance in meters (x 100).

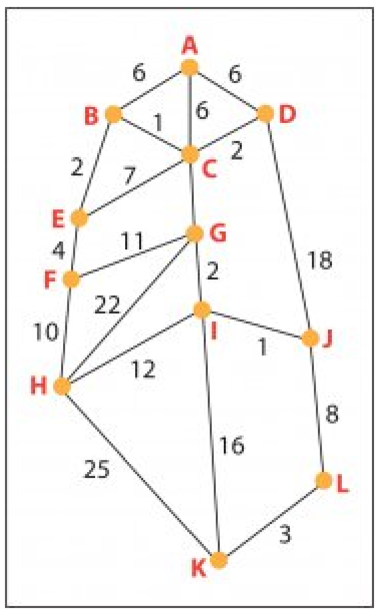
Cannaregio	Ponte Scalzi	Santa Corce	Dell ‘Orto	Ferrovia	Piazzale Roma	San Polo	Dorso Duro	San Marco	St. Mark Basilica	Castello	Arser
A	B	C	D	E	F	G	H	I	J	K	L

Let’s understand how Kruskal’s algorithm is used in the real-world example using the above map.

Step 1- Remove all loops and parallel edges

So for the given map, we have a parallel edge running between Madonna dell’Orto (D) to St. Mark Basilica (J), which is of length 2.4kms(2400mts).

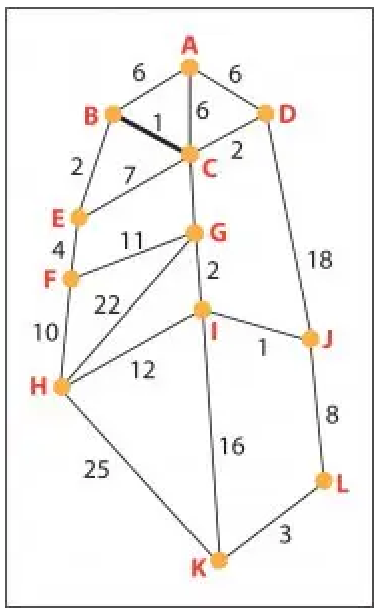
We will remove the parallel road and keep the 1.8km (1800m) length for representation.



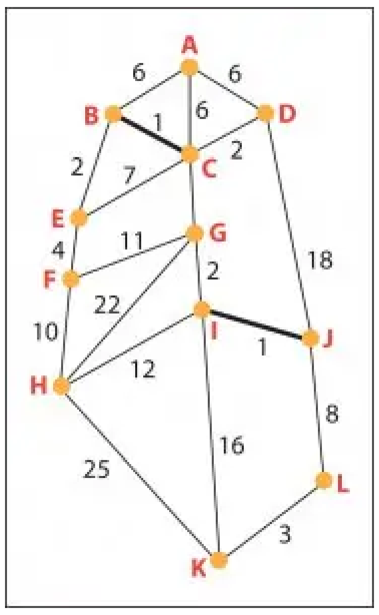
Step 2 – Arrange all the edges on the graph in ascending order. Kruskal’s algorithm considers each group as a tree and applies disjoint sets to check how many of the vertices arepart of other trees.

C,D	2
K,L	3
E,F	4
A ,B	6
A,C	6
A,D	6
E,C	7
J,L	8
F,H	10
F,G	11
H,I	12
I,K	16
D,J	18
G,H	22
H,K	25

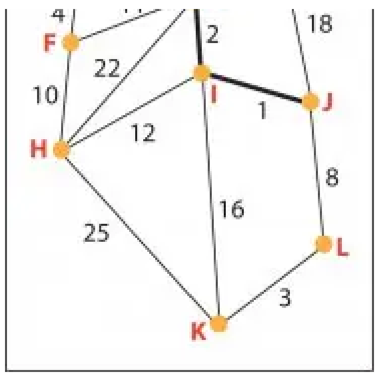
Step 3 –Add edges with least weight; we begin with the edges with least weight/cost. Hence, B, C is connected first considering their edge cost only 1.



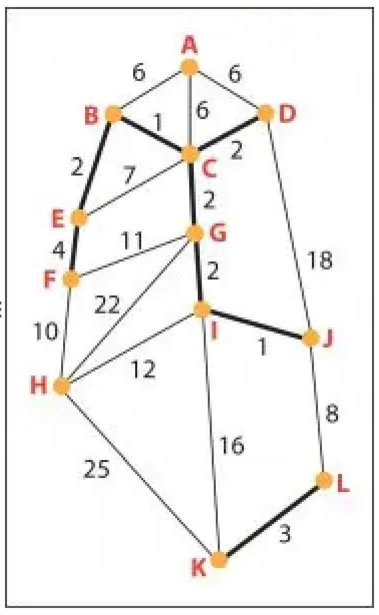
I, J has cost 1; it is the edge connected next.



Then, we connect edges with weight = 2.

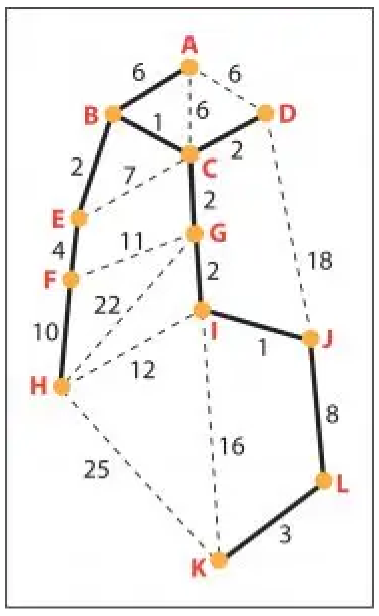


Similarly, we connect node K, Lwhich has an edge with weight = 3.



As given in the table above, all the edges are connected in ascending order, ensuring no loop or cycle is formed between 2 vertices.

Thisgives us the following graph, which is the minimum spanning tree forthe given problem.



Here Kruskal’s algorithm using C++

```
using namespace std;
const int MAX = 1e4 + 5;
int id[MAX], nodes, edges;
pair <long long, pair<int, int> > p[MAX];

void initialize()
{
    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)
    {
        // Selecting edges one by one in increasing order from the beginning
        x = p[i].second.first;
        y = p[i].second.second;
        cost = p[i].first;
        // Check if the selected edge is creating a cycle or not
        if(root(x) != root(y))
        {
            minimumCost += cost;
            union1(x, y);
        }
    }
    return minimumCost;
}

int main()
{
    int x, y;
    long long weight, cost, minimumCost;
    initialize();
    cin >> nodes >> edges;
    for(int i = 0;i < edges;++i)
    {
        cin >> x >> y >> weight;
    cout << minimumCost << endl;
    return 0;
}
```

After understanding how Kruskal's algorithm works, it's important to understand the difference between MST and TSP.

Minimum Spanning Tree vs. Traveling Salesman problem

A minimum spanning tree helps you build a tree which connects all nodes, or as in the case above, all the places/cities with minimum total weight.

Whereas, a traveling salesman problem (TSP) requires you to visit all the places while coming back to your starting node with minimum total weight.

SHARE

Following are some of the other real-life applications ofKruskal's algorithm:

1. Landing Cables
2. TV Network
3. Tour Operations

If you understood the example and working with disjoint sets, you are all set to join the CodeMonk challenge on the Disjoint Sets Union.

Arpit Mishra
Author
50 BLOGS

Related reads

The Impact Of Remote And Hybrid Working Models On DevOps

Here Is How You Can Identify And Close Skill Gaps In Software

Get advanced recruiting insights delivered every month