KRUSKAL'S ALGORITHM: Minimum Spanning Tree

→ What is a Spanning Tree?

Given an undirected and connected graph $G = (V, E)$, a spanning tree of the graph G is a tree that spans G (that is, it includes every vertex of G) and is a subgraph.

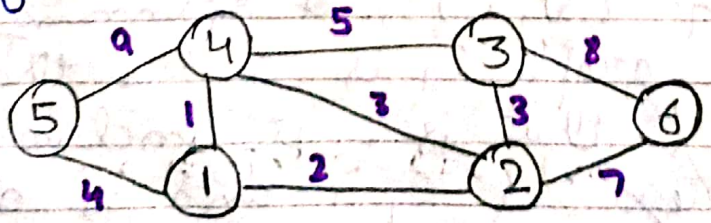
of G (every edge in the tree belongs to G).

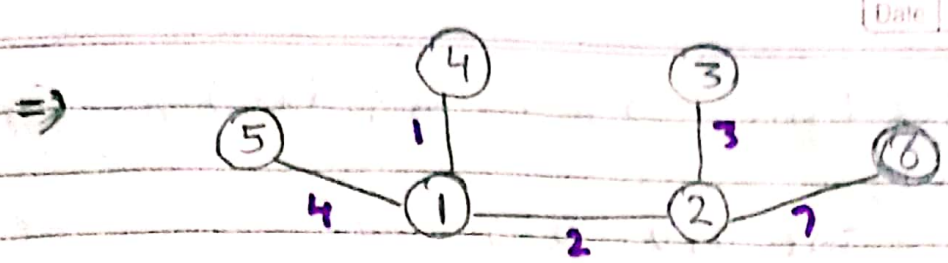
→ What is a Minimum Spanning Tree?
The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees. There also can be minimum spanning trees.

KRUSKAL'S ALGO:

1) Sort all the edges in increasing order of their weight.
2) Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If the cycle is not formed, include this edge. Else, discard it. (For this cycle check, we will be using DSU).
3) Repeat step 2 until there are (V-1) edges in the spanning tree.

e.g: Find MST for this:





CODE

```
// DSU to check if adding an
   edge will form a cycle or not
VOID MAKE_SET (INT V)
{
}

INT FIND_SET (INT V)
{
}

VOID UNION_SETS (INT A, INT B)
{
}

INT MAIN()
{
   INT n, m;
   CIN >> n >> m;
   VECTOR < PAIR < INT, PAIR < INT, INT >>> EDGES;
   // weight, edges with that weight
```

```
FOR (INT i=0; i<m; i++)
{
    INT U, V, WT;
    CIN >> U >> V >> WT;
    EDGES.PB({WT, {U, V}});
}
```

-> Sorted according to weights

```
SORT(EDGES.BEGIN(), EDGES.END());
```

-> Making all the nodes in DSU

```
FOR (INT i=1; i<=n; i++) MAKE_SET(i);
```

-> Sum of weights in forming Minimum Spanning Tree

```
INT TOTAL_COST = 0;
FOR (AUTO &EDGE : EDGES)
{
    INT WT = EDGE.F;
    INT U = EDGE.S.F;
    INT V = EDGE.S.S;
    -> If they have a common parent (means cycle is forming) continue.
    IF (FIND_SET(U) == FIND_SET(V))
    { CONTINUE;
    UNION_SETS(U, V);
    TOTAL_COST += WT;
    COUT << U << " " << V << "\n"
    // Just to visualize
}
COUT << TOTAL_COST << "\n" // Just to visualize
```

```
    RETURN 0;
}
```

INPUT :

```
6 9
5 4 9
1 4 1
5 1 4
4 3 5
4 2 3
1 2 2
3 2 3
3 6 8
2 6 7
```

OUTPUT :

```
1 4
1 2
3 2
5 1
2 6
17
```