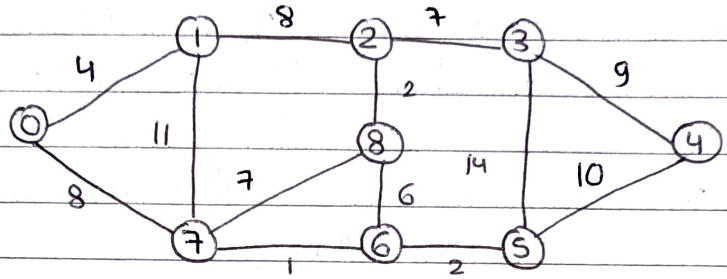
Page:
Date: / /

Day - 216

Graph - 20

* **Kruskal Algorithm:**

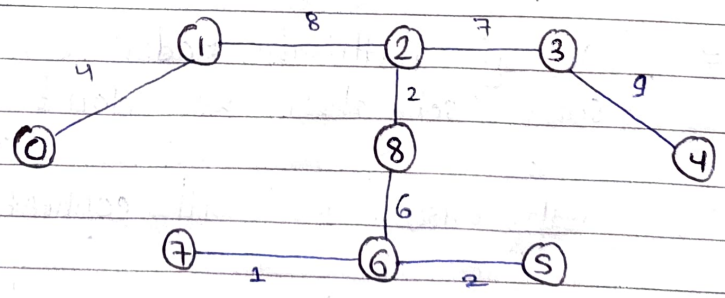


⇉ Kruskal uses just the basic understanding that if we want to connect something with min cost an distance.

⇉ Then, we generally arrange all the distance from min to max.

⇉ Then start to join without forming the cycle.

⇉ So,



⇒ So, for finding the cycle or finding that the node have another path before to get that node —

⇒ we can use BFS & DFS.

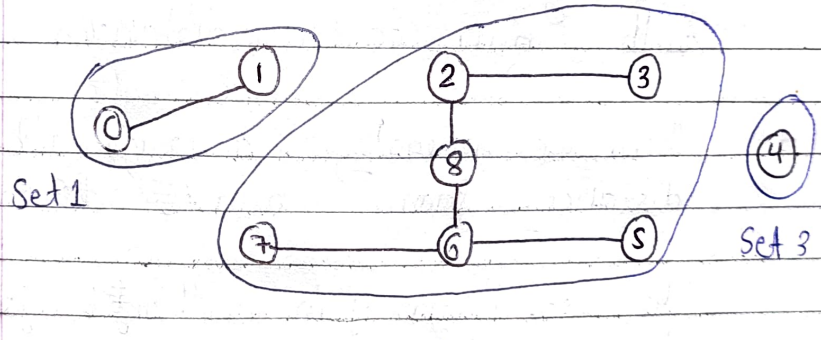
⇒ But this will increase our time compl..

⇒ So, can we do this in O(1) time?

⇒ Let's see this —

⇒ For this, we will use Disjoint sets.

⇒ First, we make sets —



⇒ So, if both the nodes are in the same set then we don't connect it.

⇒ Otherwise, we will connect them.

⇒ First, we make a array like this to denote sets —

0	1	2	3	4	5	6	7	8	9	→ set

=> Now all the nodes are in the different set.

=> Now, (7) — (6) (7 & 6 are in differ. sets)

=> So, array becomes —

0	1	2	3	4	5	6	6	8	9

=> Different set ⇐⇒ connect them

=> And so on...

⇒ Now, if two sets have 3 & 6 nodes respectively.

=> And we connect any node between these two sets nodes then we have to change the array.

=> But this will also increases the T.C.

=> So, to solve this problem, we use Union by rank or size method.

Page:
Date: / /

* Union by rank:

=> Here, we take two arrays, parent & rank.

=> Parent give you the current parent of the node.

=> Rank is used to select which set have to changed their set value.

(u,v)
(0,1)
(1,2)
(3,4)
(5,6)
(7,8)
(3,6)

	0	1	2	3	4	5	6	7	8
parent	0	1	2	3	4	5	6	7	8

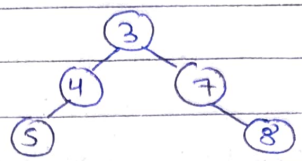
	0	1	2	3	4	5	6	7	8
rank	0	0	0	0	0	0	0	0	0

(kis set ko kis set mei merge kare)
(Height)

Steps:

1. Find the ultimate parent of u & v.
2. Find the rank of ultimate (P_u, P_v).
3. Merge the smaller set into larger set (rank).

⇒ When we are finding ultimate parent — i.e.

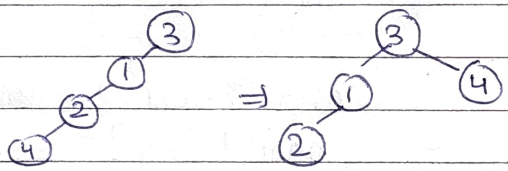


⇒ So, '5' → ultimate parent is — 3 but to find this we first have to go to 4 then 3.

⇒ This will also increase the time complexity.

⇒ For this, we will use path compression technique.

⇒ So,



⇒ If anyone asked about ultimate parent of 4 then we will do this

⇒ So, we require two functions — findParent & findRank.

Code

⇒ int findParent (int u, vector <int> &parent)
{
if (u == parent [u])
return u;
return parent[u] = findParent (Parent [u], parent);
}
↳ Path Compression

⇒ void unionByRank (int u, int v,
vector< int> parent, vector<int>
rank){
int pu = findParent (u, parent);
int pv = find Parent (v, parent);

if (rank [pu] > rank [pv]){
parent [pv] = pu;
}
else if (rank [pu] < rank [pv])
parent [pu] = pv;
else {
parent [pv] = pu
rank [pu]++;
}

⇒ T.C. → O(E log E)
S.C. → O(V+E)