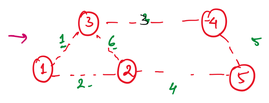
Graphs - MST DSU

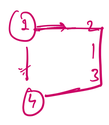
Friday, 30 July 2021 7:58 PM

There's a state that has been hit by a cyclone. V E



5 cities

Aim of govt:
↓
connect all cities by reconstructing min roads.
↓
connecting all the nodes with min no of edges

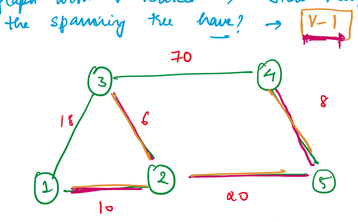


ans = 4 roads →
1 2 3 4
2 3 4 5
1 6 4 5

* Spanning Tree

* No loops
↓
connected acyclic graph

For a graph with V vertices → How many edges will the spanning tree have? → V-1 → Tree

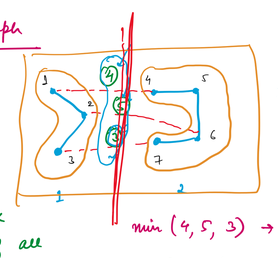


What will be min cost required to connect all cities?

6 + 8 + 20 + 10
= 44

MST → minimum spanning tree

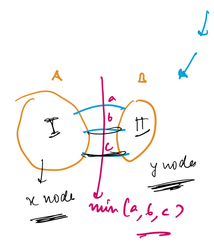
Cuts in a graph



7 nodes
↓
graph

We will pick minimum of all edges that pass through this cut

min (4, 5, 3) → 3

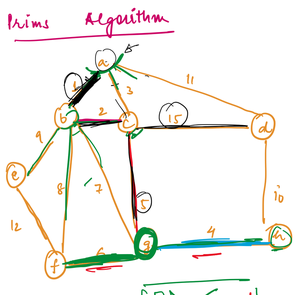


x node

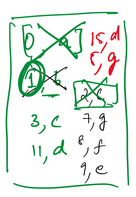
min (a, b, c)

(*) We will pick only one edge passing through the cut

Prims Algorithm



a ✓
b ✓
c ✓



ans+1

visited array
↓
node that have been added to the component will be mark visited

a → (b, 1) (c, 3) (d, 11)

c → d, 15
g, 5

what all edges are there going out of this component

Algo

1. Take a node (from pq.top) and mark it visited
2. We will iterate over its neighbour and add edge in min heap

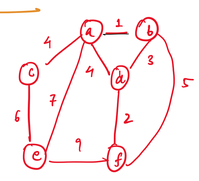
(edge weight, destination node) + visited

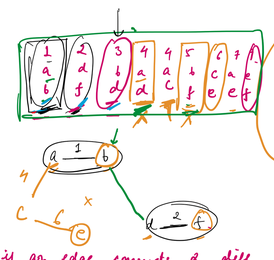
pq.push((0, 1))
while (! pq.empty())

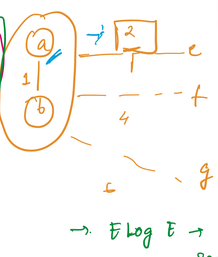
Dijkstra
↓
you push distance from src node

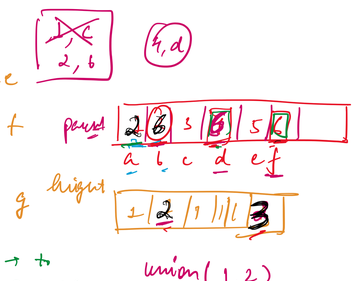
Prim
↓
you push weights of neighbs

Kruskal









parent | 2 | 6 | 3 | 6 | 5 | 6 |
a b c d e f

height | 1 | 2 | 1 | 1 | 1 | 3 |

union(1, 2)

par[1] = 2

if an edge connects a diff component then take it

if an edge connects node which are in same component then we use in taking it

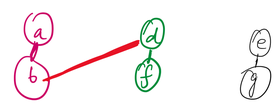
→ E log E → to sort edges

→ E*(V + E)

E²

Why can't visited be used

a	b	d	f
T	T	T	T

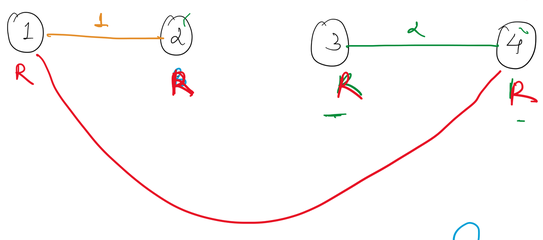


DSU → Disjoint set union

wt	1	2	3	4
u	1	3	1	2
v	2	4	4	4

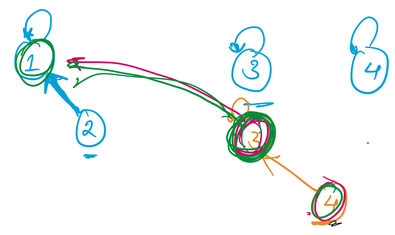
ans += edge weight

sort
for
if (fav = =)
else (union)
=ans=



1	1	1	3
1	2	3	4

connect nodes →



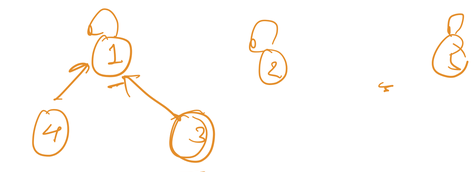
union (a, b)
p1 = id (a)
p2 = id (b)
if (p1 ! = p2)
id[p1] = p2
p1=3 p2=1

find (a) {
while (id[a] ! = a) {
a = id[a]
}
return a
}

id/parent | 1 | 1 | 1 | 1 | 2 |
1 2 3 4 5

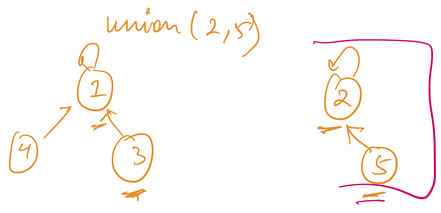
find_parent(5)

union(1, 3)

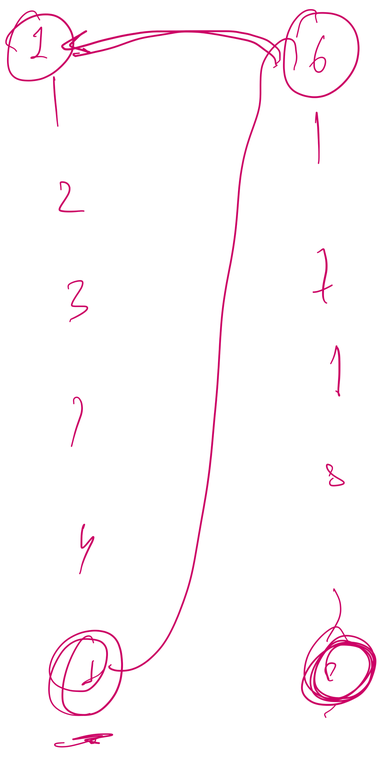


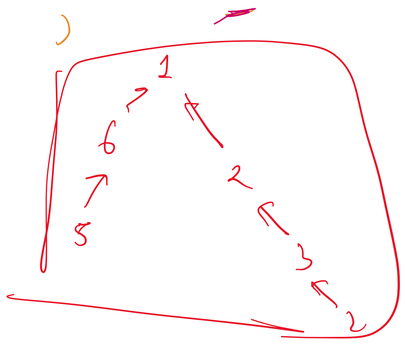
union (3, 4)

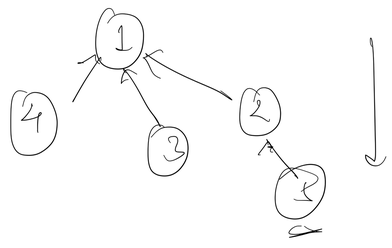
union (2, 5)



union (3, 5)





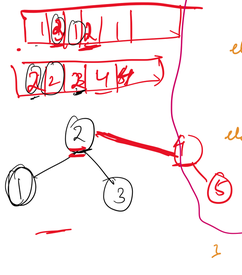


parent

height

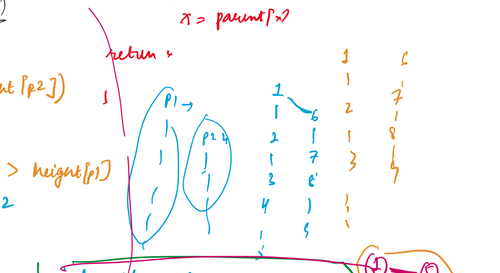
union (x, y) {
p1 = find(x) //O(1)
p2 = find(y)
if (p1 = = p2) return;
if (height[p1] > height[p2])
parent[p2] = p1
else if (height[p2] > height[p1])
parent[p1] = p2
else
parent[p1] = p2
height[p2]++;

Optimisation no. 1



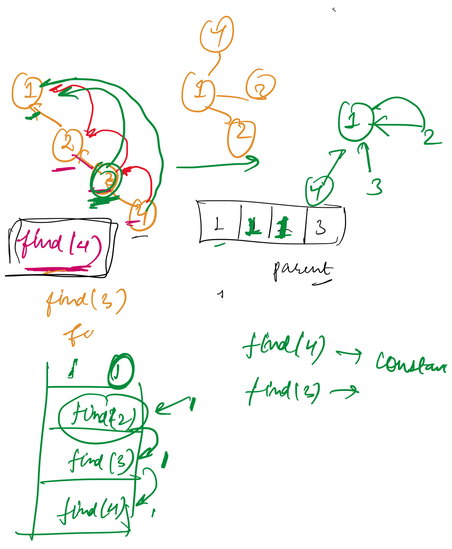
find (x) {
while (parent[x] ! = x)
x = parent[x]
return x

Optimisation 2



int find (x)
if (parent[x] = = x)
return x
p = find(parent[x])
parent[x] = p
return p
}

Amortized complexity
↓
O(1)



find(4) → constant
find(2) →

(1) E log E

(2) E