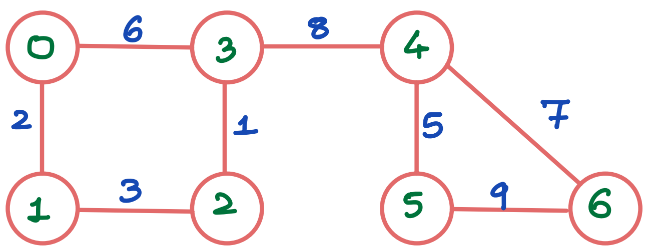
GRAPH

↓ Greedy

MST: Prims
Kruskal

GRAPH:

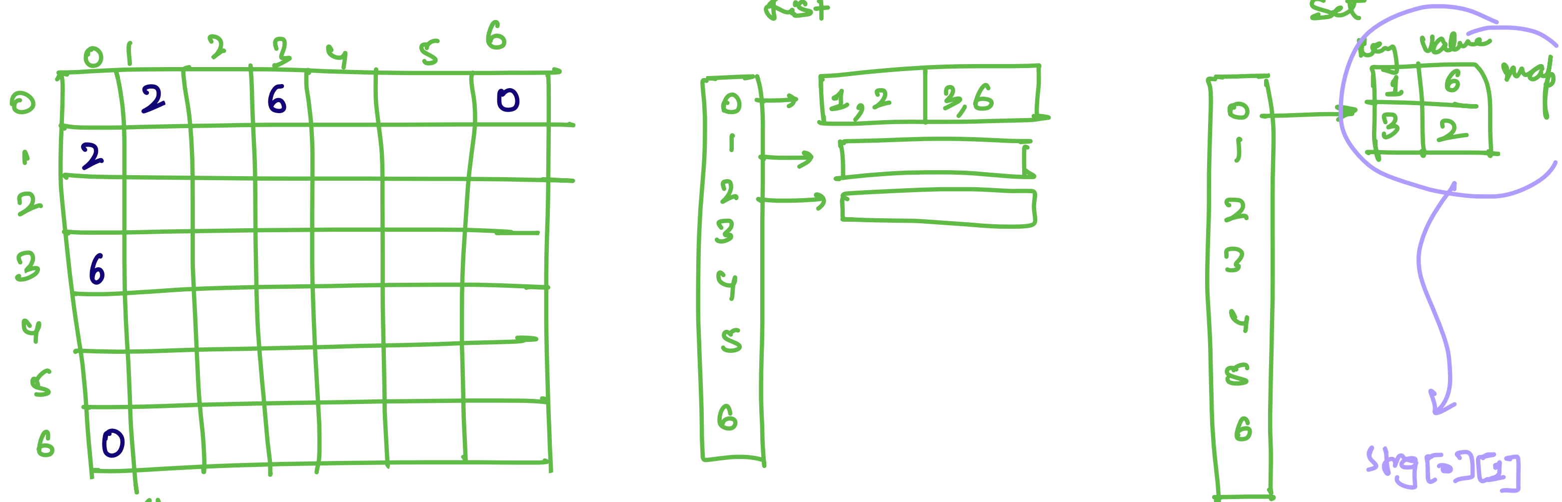


- nodes/vertices
- edges

(undirected weighted graph)

Store ?

Adjacency Matrix | Adjacency List | Adjacency Set



Sparse: mostly values 0

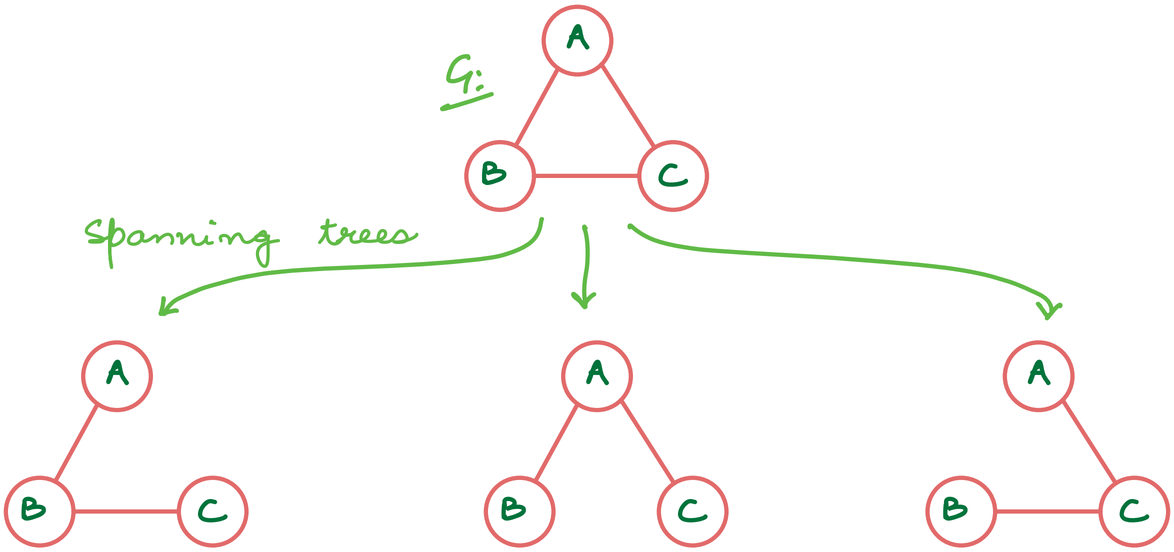
map<int, map<int, int>> strg

a-b@c

SPANNING TREE

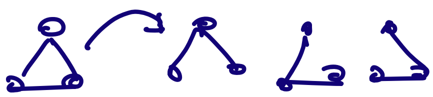
- A spanning tree is a subset of graph G, which has all vertices covered with minimum possible number of edges.

- Spanning tree doesnot have cycle and it cannot be disconnected.



3 vertex (n vertex)
2 edges (n-1 edges)

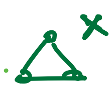
Properties:



- A connected graph G can have more than one spanning tree.

- Spanning tree has n-1 edges where n is the number of nodes (vertices).

- All possible spanning trees of graph G, have the same number of edges and vertices. — n vertex — n-1 edges

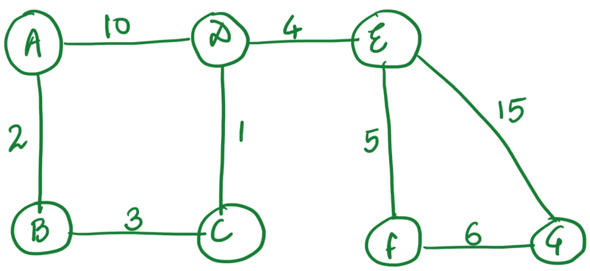
- Spanning tree doesnot have any cycle (loops). 

- Removing one edge from spanning tree will make the graph disconnected i.e. spanning tree is minimally connected.

- Adding one edge to the spanning tree will create a circuit or loop i.e. the spanning tree is maximally acyclic.

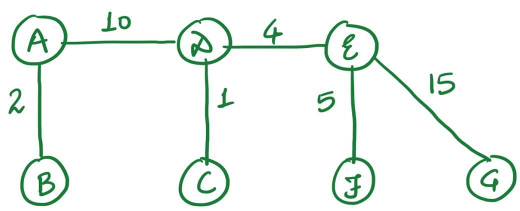
MINIMUM SPANNING TREE (MST):

In a weighted graph, a minimum spanning tree is a spanning tree that has minimum weight than all other spanning trees of the same graph.



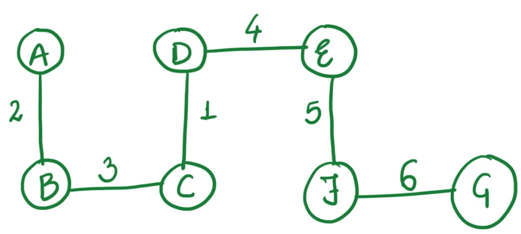
Different spanning trees possible :-

1.



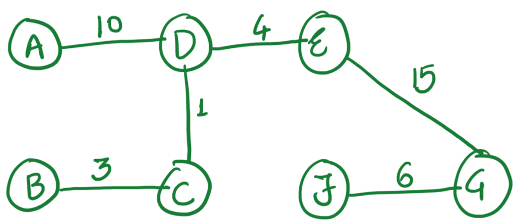
Cost :
2+10+1+4+5+15
= 37

2.

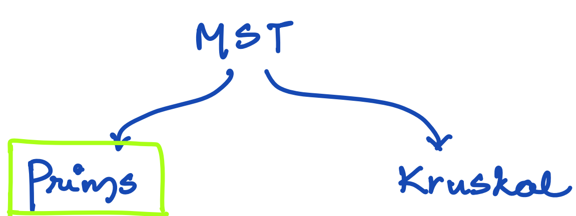


Cost :
2+3+1+4+5+6
= 21 → least cost } MST

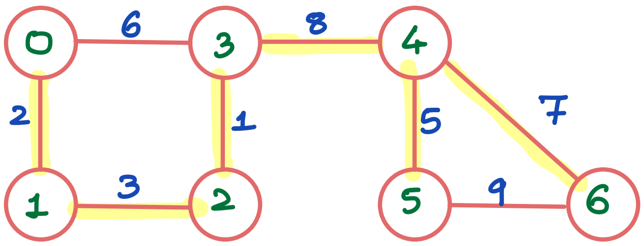
3.

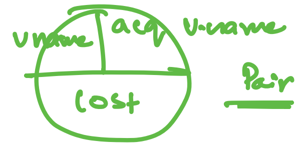


Cost :
10+1+3+4+15+6
= 39



PRIMS ALGORITHM :



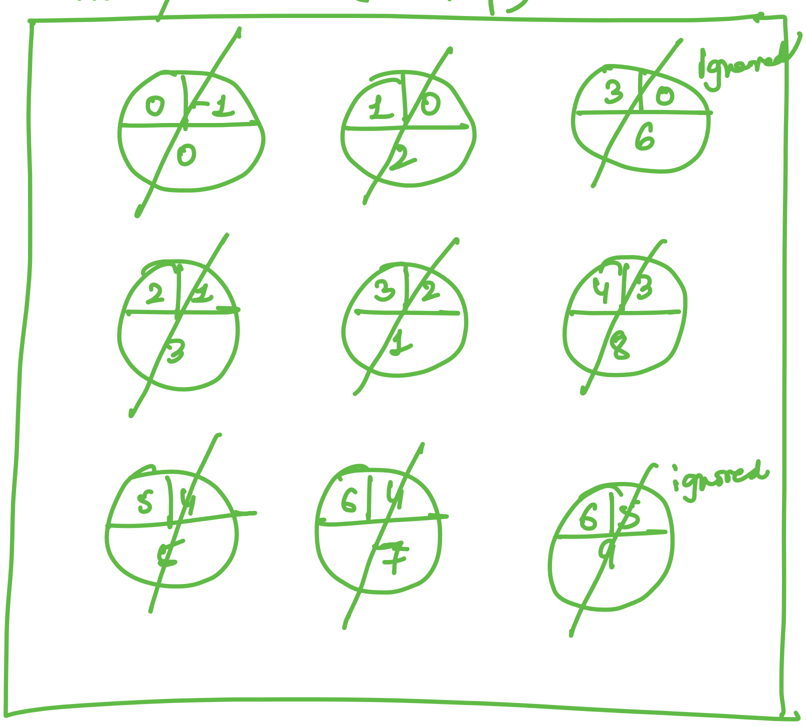


visited

0 ✓
1 ✓
2 ✓
3 ✓
4 ✓
5 ✓
6 ✓

Pick any starting node

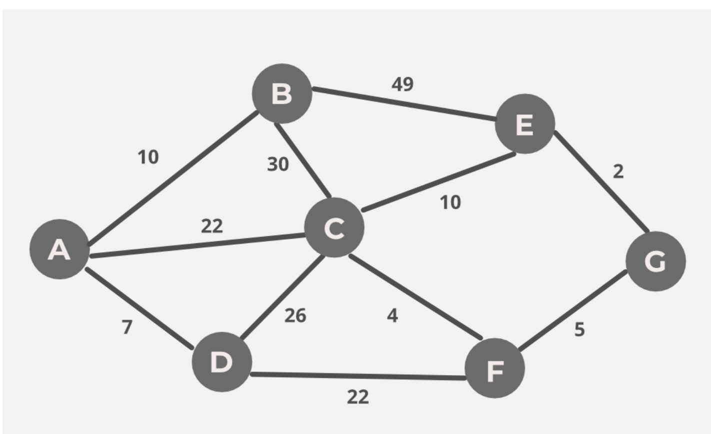
Priority Queue (Min Heap)



already visited Ignore

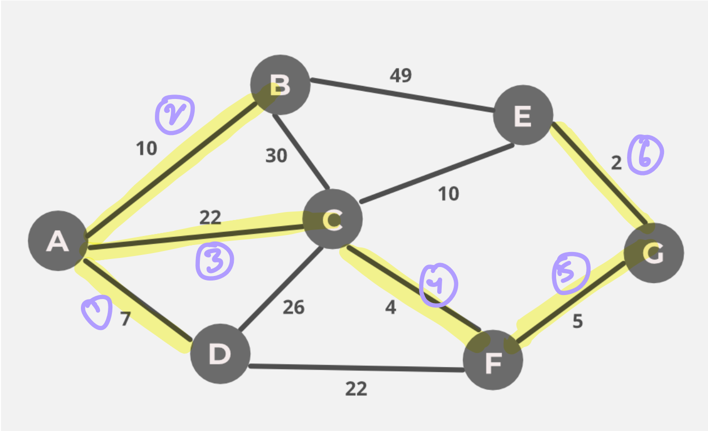
→ Remove min cost node
→ visited
→ Print
→ nbrs unvisited

~~-1 → 0 @ 0~~
0 → 1 @ 2
1 → 2 @ 3
2 → 3 @ 1
3 → 4 @ 8
4 → 5 @ 5
4 → 6 @ 7



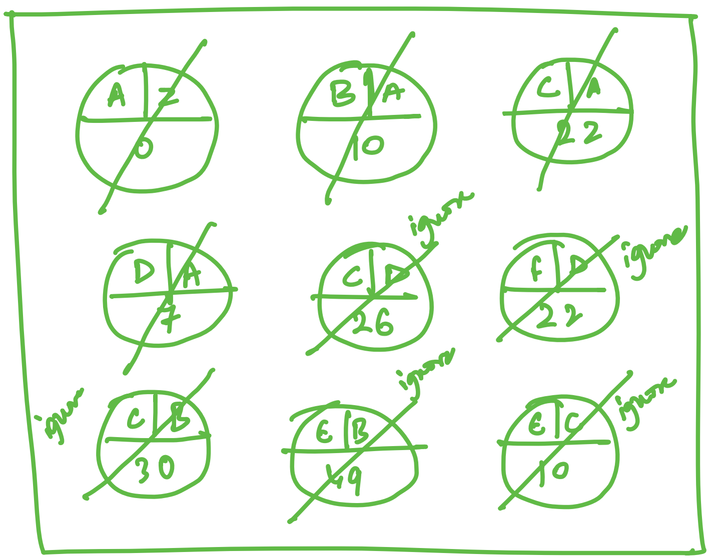
Using Prim's algorithm to construct a minimum spanning tree starting with node A, which one of the following sequences of edges represents a possible order in which the edges would be added to construct the minimum spanning tree?

(A) (E, G), (C, F), (F, G), (A, D), (A, B), (A, C) ✗
(B) (A, D), (A, B), (A, C), (C, F), (G, E), (F, G)
(C) (A, B), (A, D), (D, F), (F, G), (G, E), (F, C)
(D) (A, D), (A, B), (D, F), (F, C), (F, G), (G, E) ✓



A ✓
B ✓
C ✓
D ✓
E ✓
F ✓
G ✓

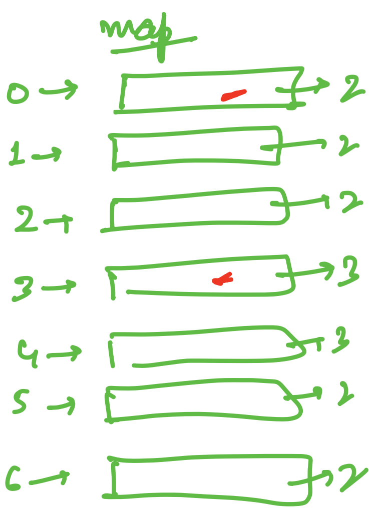
→ remove
→ visited
→ print
→ nbrs



Z → A : 0
A → D : 7
A → B : 10
A → C : 22
C → F : 4
F → G : 5
G → E : 2

$V + E(1 + \log E + 1 + 1 + 1) + 2E$

feel   top   pop



$E \log E$

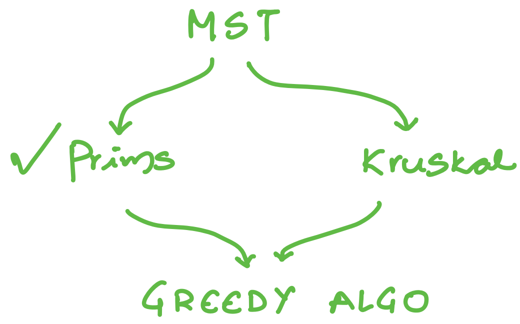
$E \log V^2$

$2E \log V$

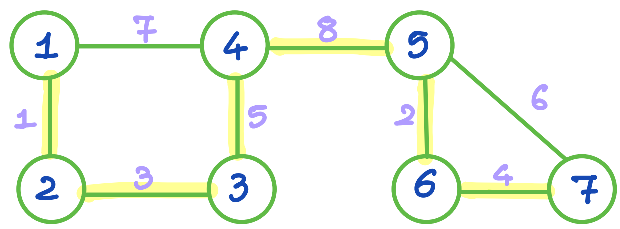
$O(E \log V)$

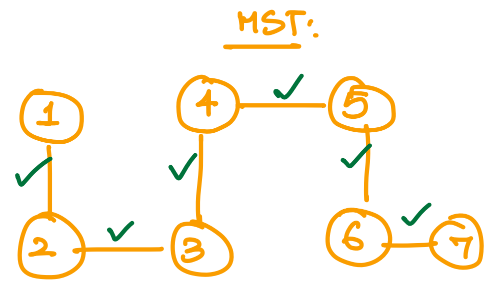
complete graph:
$E = V^2$

$2+2+2+3+3+2+2$ } $2E$
16

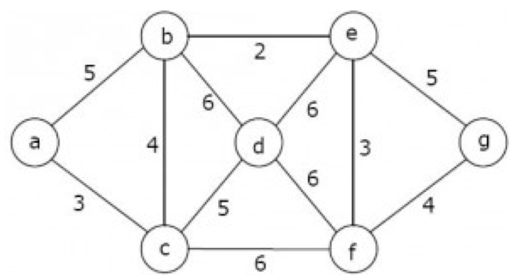


KRUSKAL ALGO:





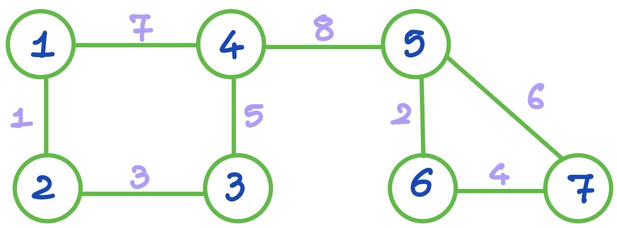
Consider the following graph:



Which one of the following is NOT the sequence of edges added to the minimum spanning tree using Kruskal's algorithm?

(A) (b,e)(e,f)(a,c)(b,c)(f,g)(c,d) ✓
(B) (b,e)(e,f)(a,c)(f,g)(b,c)(c,d) ✓
(C) (b,e)(a,c)(e,f)(b,c)(f,g)(c,d) ✓
(D) (b,e)(e,f)(b,c)(a,c)(f,g)(c,d)

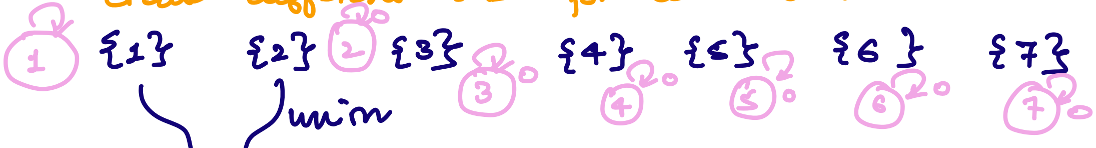
Prims	Kruskal
→ start with any node	→ pick edge with least weight
→ Priority Queue	→ Don't use priority queue
→ explore the nbrs (can't jump from 1 edge to another edge)	→ don't explore the nbrs (pick the edge with the least weight).



Edges inc order of wt:

1	2	3	4	5	6	7	8
✓	✓	✓	✓	✓	ignore	ignore	

Create different sets for each vertex



{1 2} {3} {4} {5} {6} {7}

↓ union

{1 2} {3} {4} {5 6} {7}

↓ union

{1 2 3} {4} {5 6} {7}

↓ union

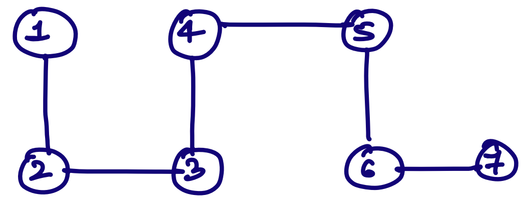
{1 2 3} {4} {5 6 7}

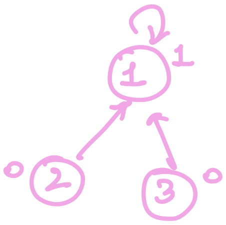
↓ union

{1 2 3 4} {5 6 7}

↓ union

{1 2 3 4 5 6 7}





→ Parent
→ Rank

	1	2	3	4	5	6	7
Parent :	1	1	1		5	5	
Rank :	1	0	0		1	0	

Root: if (i == parent[i]) i is root.

Disjoint Set: 2 sets are called as disjoint when there is nothing is common.

{1 2 3 4} {5 6 7} → disjoint ✓

{1 2 3} {3 4} → disjoint ? ✗

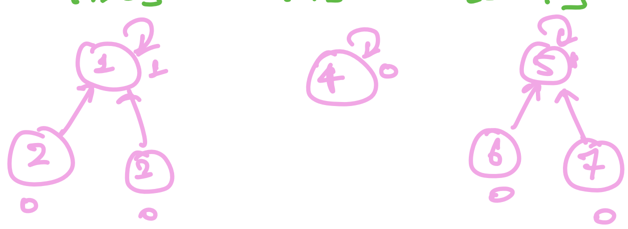
Operations:

1) union

{1} {2} → {1, 2}

2) find

[1] {1 2 3}   [4] {4}   [5] {5 6 7}



Representative Element

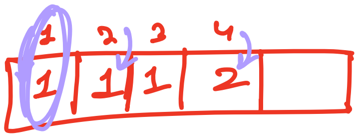
Edge: 5 & 6 vertex

figure out the set to which 5 & 6 belongs?

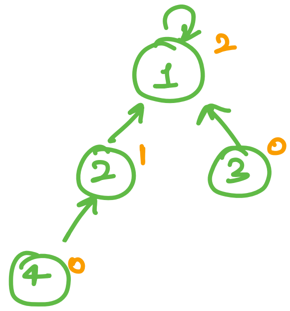
ask 5 vertex who is ur Rt ? 5
ask 6 ——————— ? 5
} same (same set)

Visualise set in the form of tree

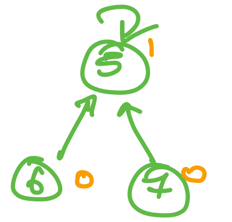
(store array)



{1 2 3 4}



{5 6 7}



find(4) ? 1

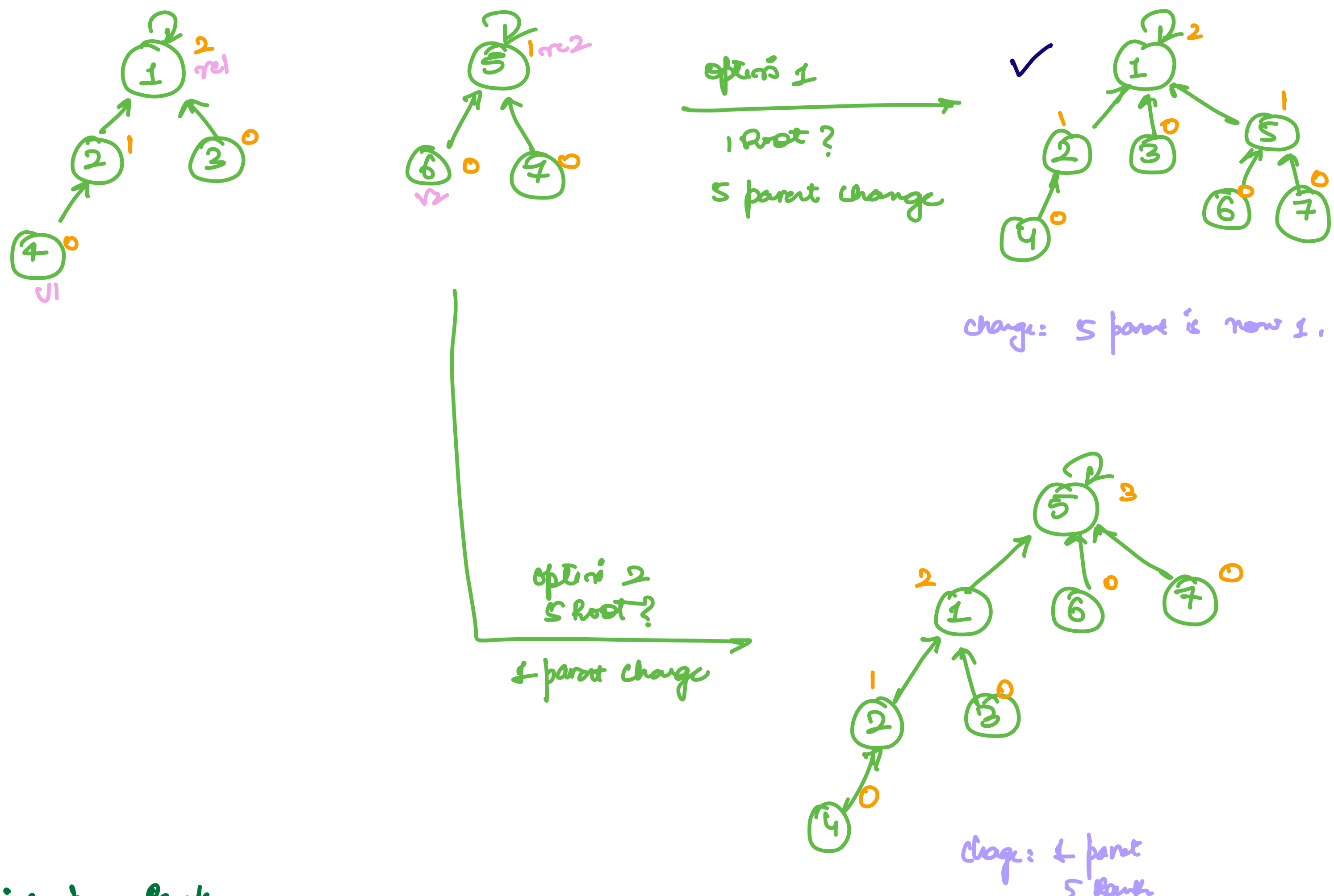
↓

op: representative element

find(7) ? 5

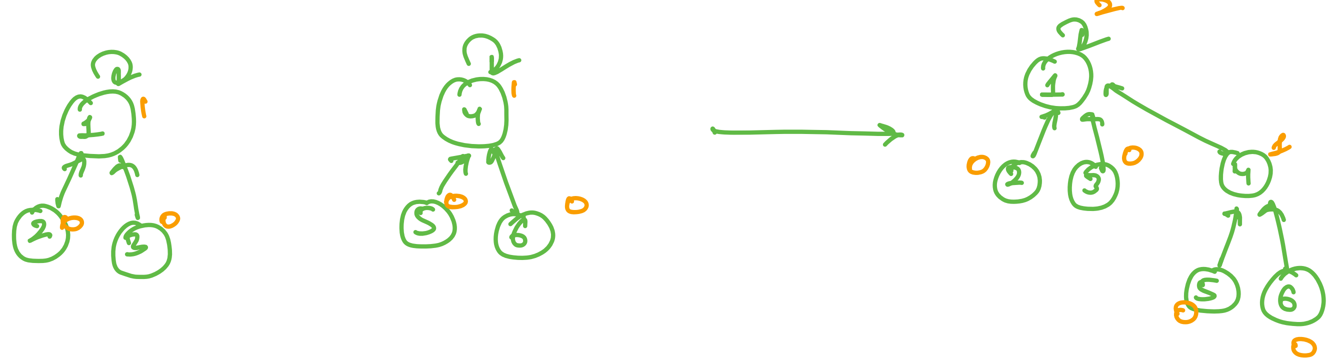
Union ?

{1 2 3 4} ∪ {5 6 7} {1 2 3 4 5 6 7}

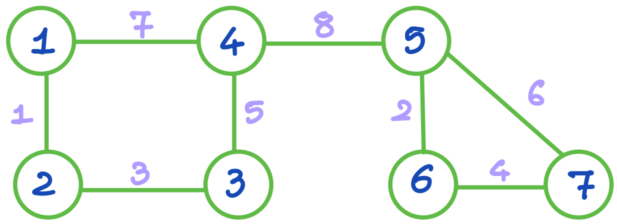


Union by Rank

Root whose rank is lower its parent will be changed.



4 parent change
1 rank change



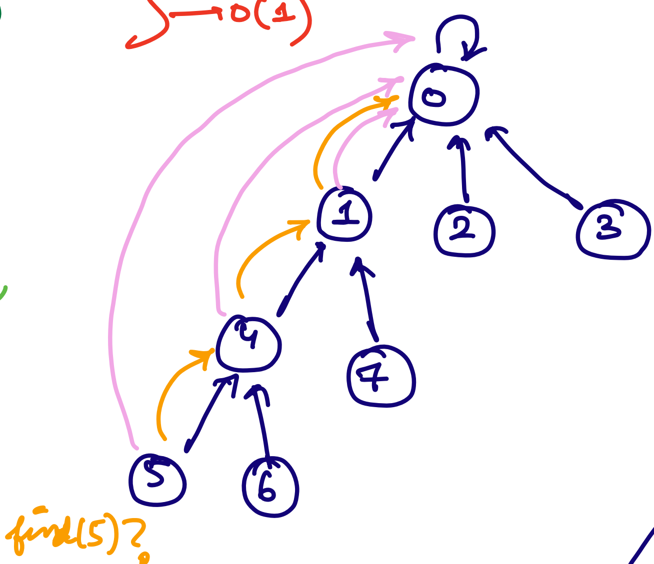
[[1,2,1], [2,3,2], [3,4,5], [1,4,7] .... [6,7,4]]

vectors: [1,4,7] → u, v, cost edge

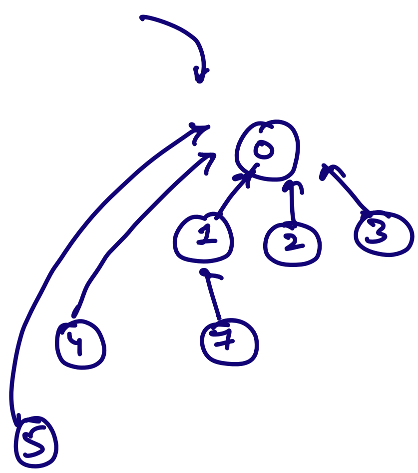
vector: [6,7,4]

Union: TC: find + O(1) → O(1)
find: TC: O(h) → O(1)

Union → Rank, Path Compression



find(5) :



TC: $E \log E + V + E(1 + 1)$

- $E \log E$ → sort edges
- V → initialization
- $1 + 1$ → union, find

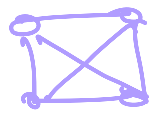
$E \log E + V + 2E$

$O(E \log E)$

$O(E \log V^2)$

$O(2E \log V)$

$O(E \log V)$



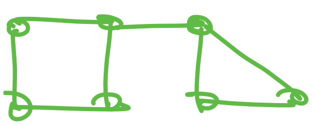
$E = V^2$

$E = {}^{V}C_2$

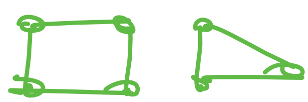
Prims, Kruskal] $O(E \log V)$

MTE: unit 1, 2, 3 → except Dijkstra, Bellman ford

Connected Components:

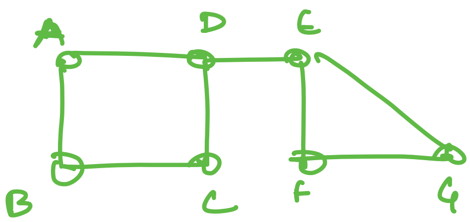


1 component

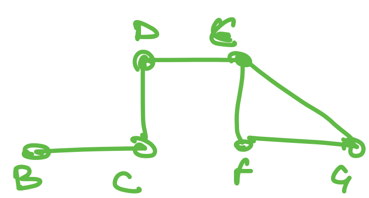


2 components

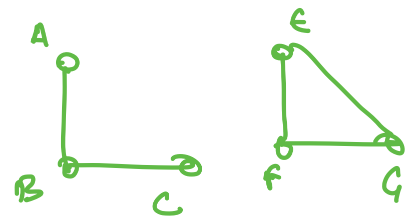
Cut vertex:



A vertex remove

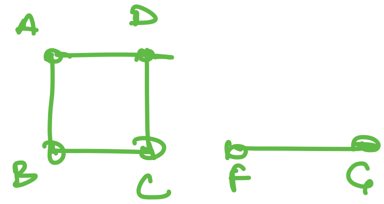


D vertex remove → Cut vertex

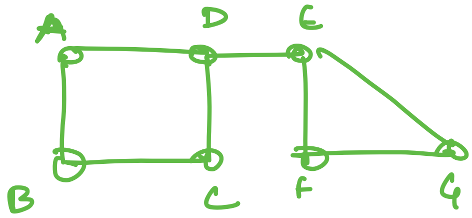


Result: D, E

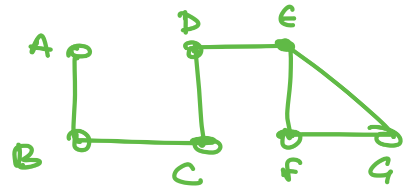
E vertex remove



Bridge:



AD edge remove



DE edge remove
↓
Bridge.