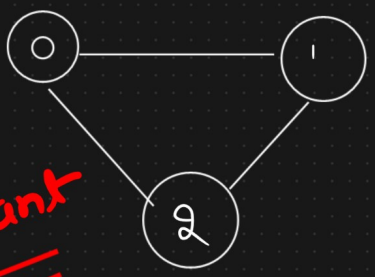
Path Exists or Not



Important Interviews

source = 0

destination = 2

BFT → ⊖ (V+E)

	0	1	2
Visited =	~~false~~ True	~~false~~ True	~~false~~ True

Queue
↳ fifo → append & popleft

~~0~~ ~~1~~ ~~2~~

node = ~~0~~ ~~1~~ 2
↳ True → Exists Path from source to destination

Application of DFT/BFT

DFT/BFT (Graph Traversal Algorithm)
↳ to check the existence of any path
↳ whether graph is connected or not

Disconnected graph

source = 1
Dst = 4

invertBinary(root) Invert Binary Tree

— Recursion

leftSubtree = self.invertBinary(root.left)

rightSubtree = self.invertBinary(root.right)

Inversion {
root.left = rightSubtree
root.right = leftSubtree

return root

Graph cycle Detection

Union Rank
&
Path compression
$\Rightarrow$ optimized time complexity $\rightarrow O(1)$

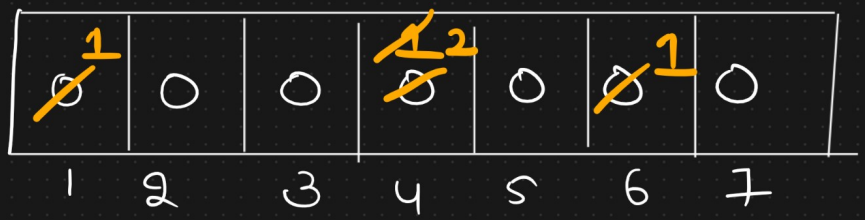
(1,2)
u = 1
v = 2

Parent

(1,2) ✓
(2,3) ✓
(4,5) ✓
(6,7) ✓
(5,6) ✓
(3,7)

Initialization

Rank



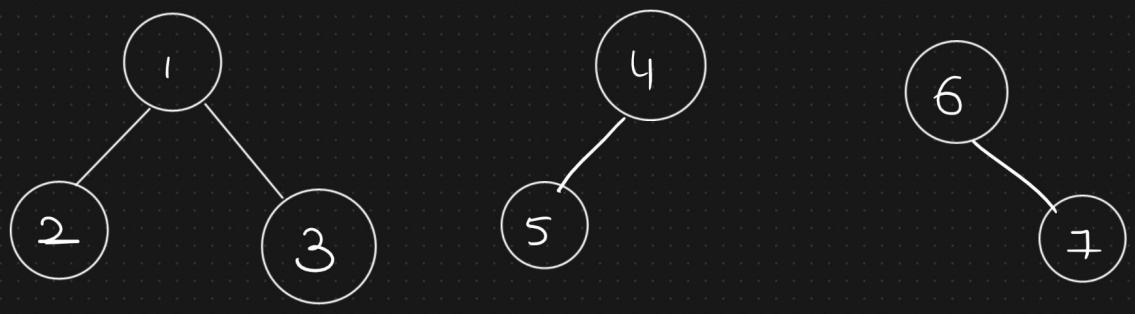
u v
(1, 2) ⟶ par_u = 1 — rank_u = 0
par_v = 2 — rank_v = 0

1) find ultimate parent of u & v
par_u
par_v

Algorithm

2) find rank of par_u & par_v

3) Connect smaller rank to larger rank node.



u v
(2, 3) — par_u = 1 — $Rank_1 = 1$

par_v = 3 — $Rank_3 = 0$

u v
(4, 5) — par_u = 4 — $Rank_4 = 0$

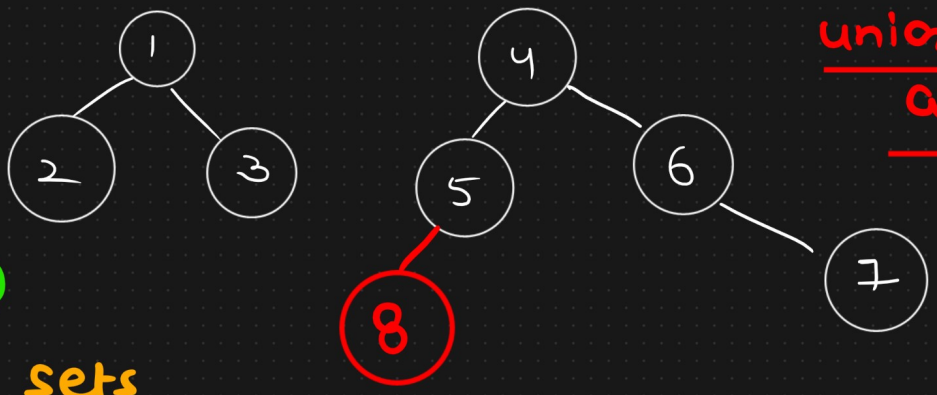
par_v = 5 — $Rank_5 = 0$

u v
(6, 7) — par_u = 6 — $Rank_6 = 0$

par_v = 7 — $Rank_7 = 0$

u v
(5, 6) — par_u = 4 — $Rank_4 = 1$

par_v = 6 — $Rank_6 = 1$



union Rank algo

Application

1) Disjoint sets

↳ {3,7}

↳ Not connected

Ultimate Parent {3,7}

{3} → 1

{7} → 4

Not equal

⇓ 100% sure

Not connected

{8,7}

↳ Connected

{8} = 4

{7} = 4

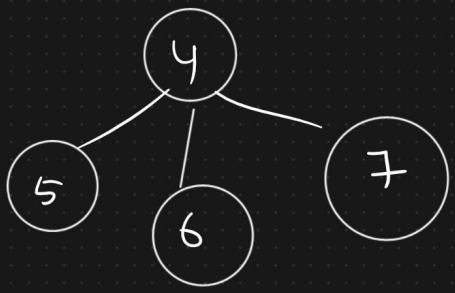
100% sure

⇓

Connected

Path compression

↳ ultimate parent of every node & storing that only



(3,7) —— par_u = 1 —— $Rank_1 = 1$

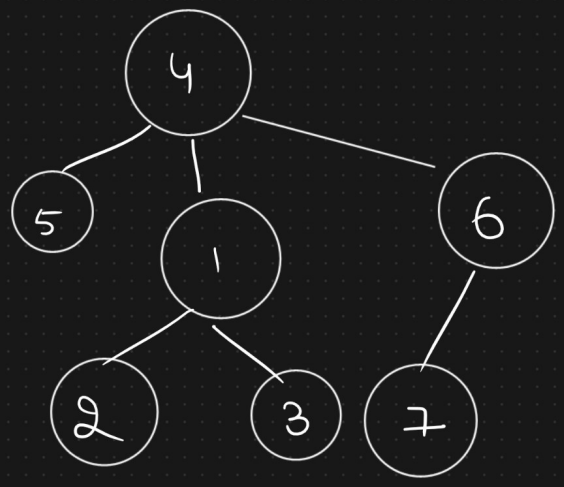
par_v = 4 —— $Rank_4 = 2$

<3,7> ⇒

<3> → 4

<7> → 4

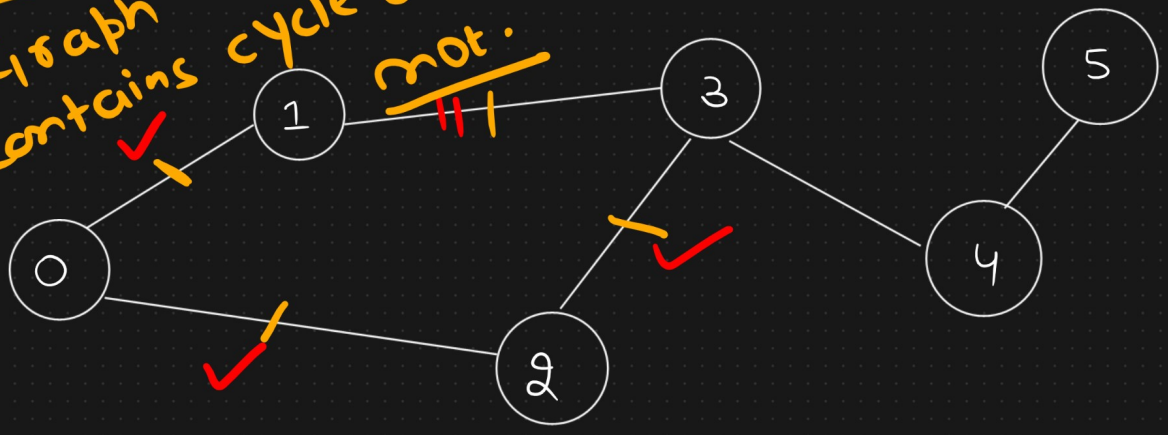
Connected



Graph cyclic or acyclic

Application

↳ Graph contains cycle or not:



Rank =

0	1	2	3	4	5
0	~~0~~ 1	0	0	0	0

Parent =

0	1	2	3	4	5
~~0~~ 1	1	~~2~~ 1	~~3~~ 1	4	5

Ultimate Parent (Path compression)

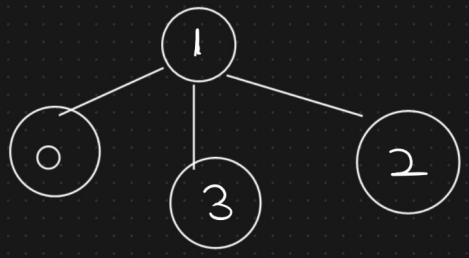
Parent of u & v — different → union rank

Same → cycle in the graph

(0, 1)

Par_0 = 0 — Rank 0 = 0

Par_1 = 1 — Rank 1 = 0



(0, 2)

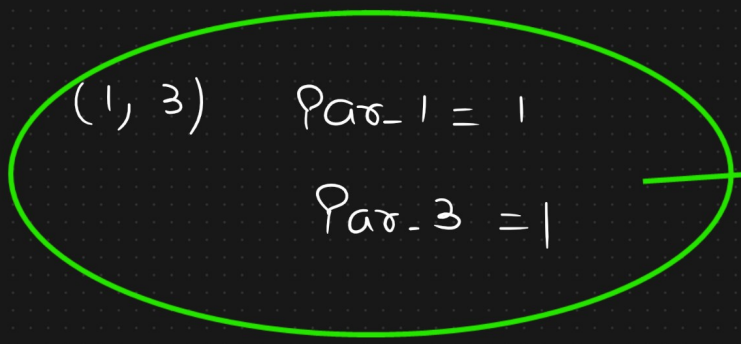
Par_0 = 1 — Rank 1

Par_2 = 2

Rank 2 = 0

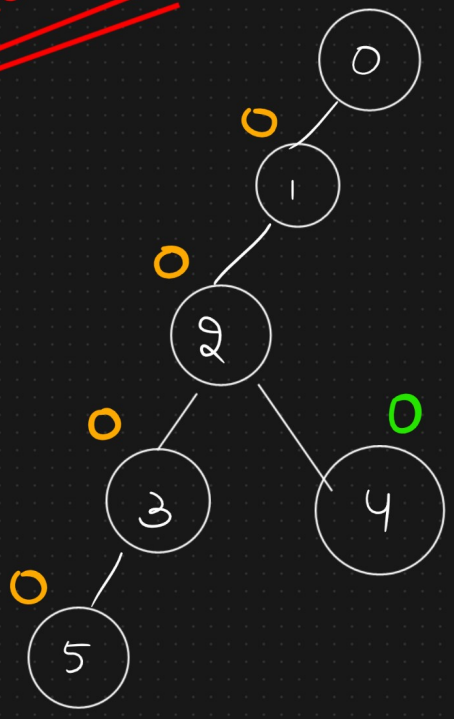
(2, 3) Par_2 = 1 Rank 1 = 1

Par_3 = 3 Rank 3 = 0



Graph contain cycle

Recursion



Parent(5) = 0

Path compression

Parent[5] = find(5)

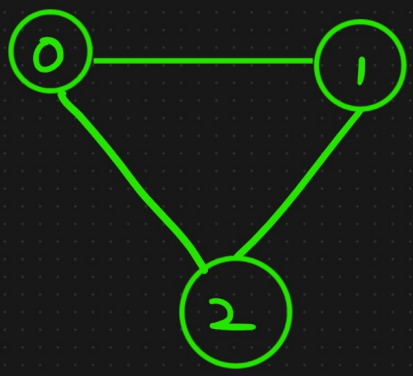
Parent[3] = find(3)

Parent[2] = find(2)

Parent(1) = find(1)

Parent[0] = find(0)

↳ 0



Python → dict — Hashing

↳ Key : Value

Hashiy[2] = 4

Hashiy[10] = KeyError

2 : 4
4 : 8
8 : 16

collections

Defaultdict

↳ avoid KeyError Exception

Hashiy(10) = 'Default value'

0/None