Lecture 26

Greedy Algorithms: Disjoint Subsets & Union-Find Algorithms.

Disjoint Sets

- Disjoint Sets and Operations
- Detecting a cycle
- Graphical Representation
- Union-Find Algorithm
- Array Implementation & Collapsing

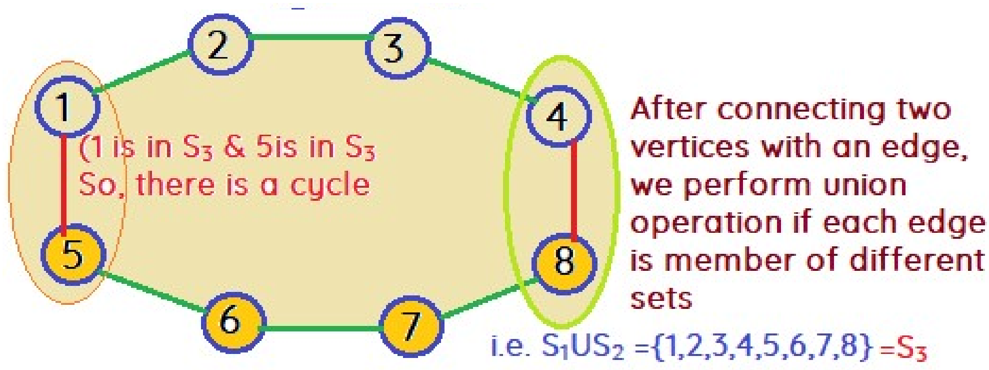
(1) Disjoint Set Operation

- Disjoint sets are similar to sets in Mathematics but not exactly.
- Disjoint sets a little bit changed for making them useful in algorithms
- A well-known algorithm for disjoint sets is Kruskal'a algorithm that detect a cycle in a graph

- Let's see the difference of disjoint sets from Mathematics

(1) Disjoint Set Operation

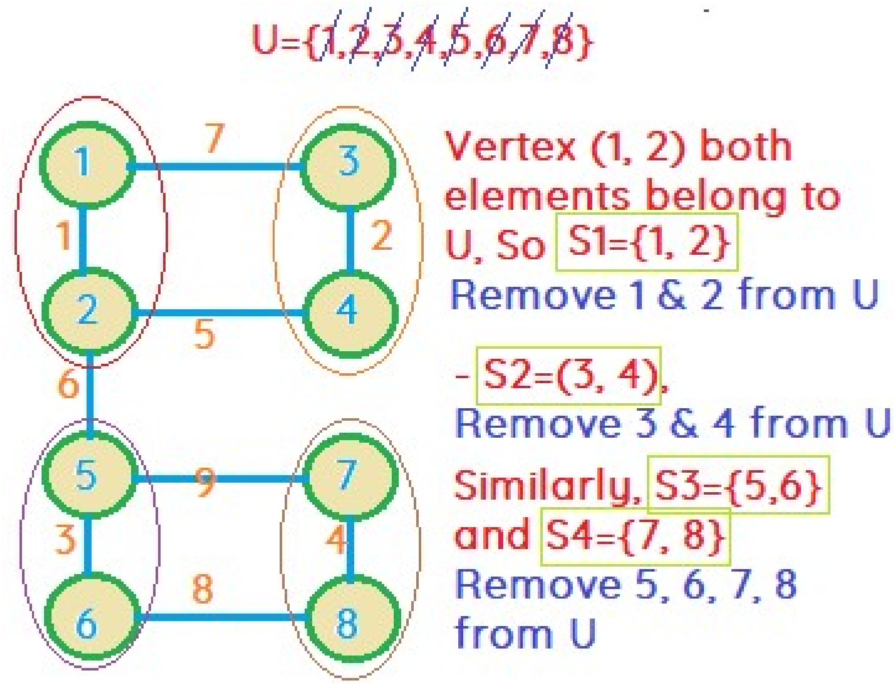
- Find & Union
- $S_1=\{1, 2, 3, 4\}$, $S_2=\{5, 6, 7, 8\}$
- $S_1 \cap S_2 = \varphi$
- Set membership operation is
  - $1 \varepsilon S_1, 6 \varepsilon S_2, 8 \varepsilon S_2$ $etc.$



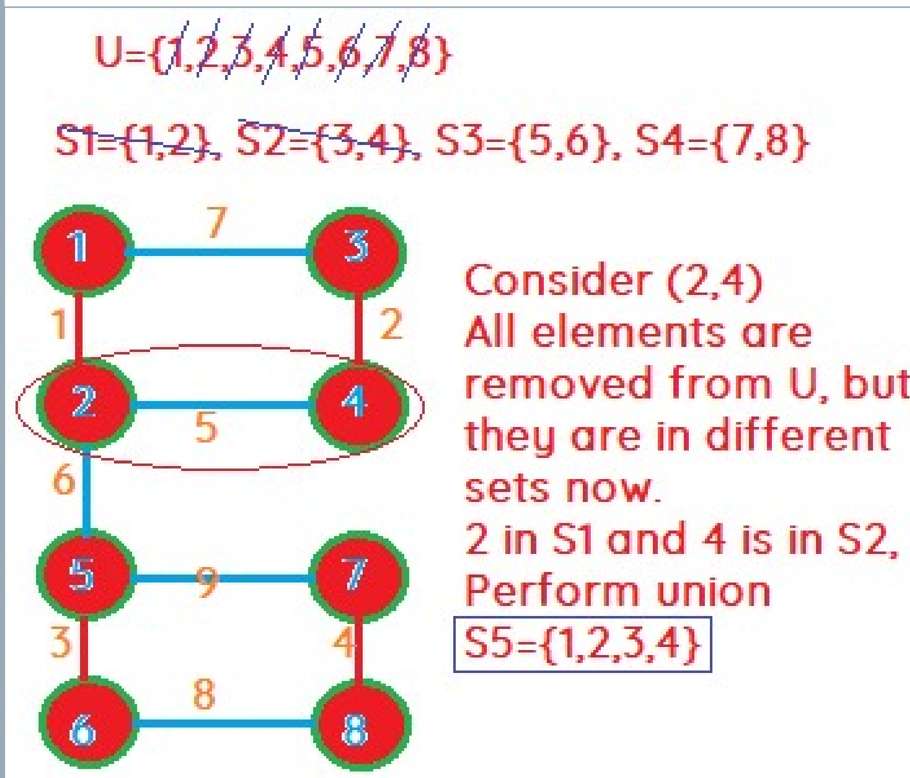
- Union Operation by connecting (4, 8) and (1,5) if each vertex is belonging to different set.
  - Let's Say $S_1US_2=S_3=\{1,2,3,4,5,6,7,8\}$, thus S_1 and S_2 are deleted.
  - If both vertex belong to the same set, then there is a cycle.
  - Similarly, for (1,5).
- This is the way to find out cycle in a graph

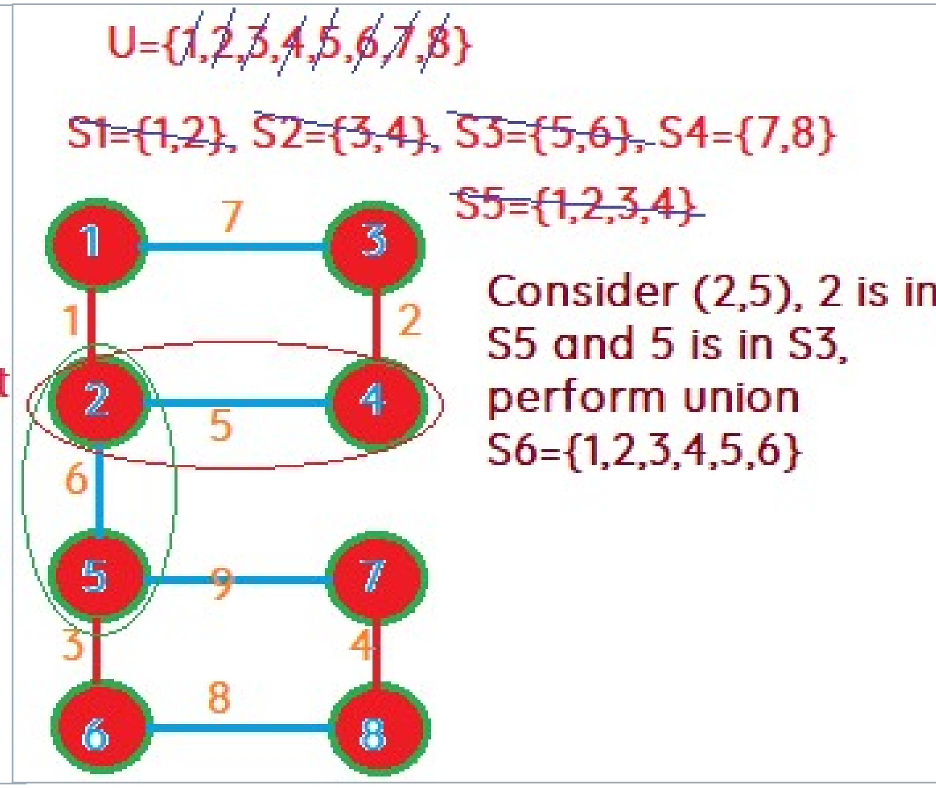
(2) Detecting Cycles

- Our target is to detect cycle in a graph
  - U={1, 2, 3,, 4, 5, ,6, 7, 8}
  - How to take help of disjoint sets, to find cycles?
  - Consider 8 vertices with U
  - Each element is considered as a set



(2) Detecting Cycles





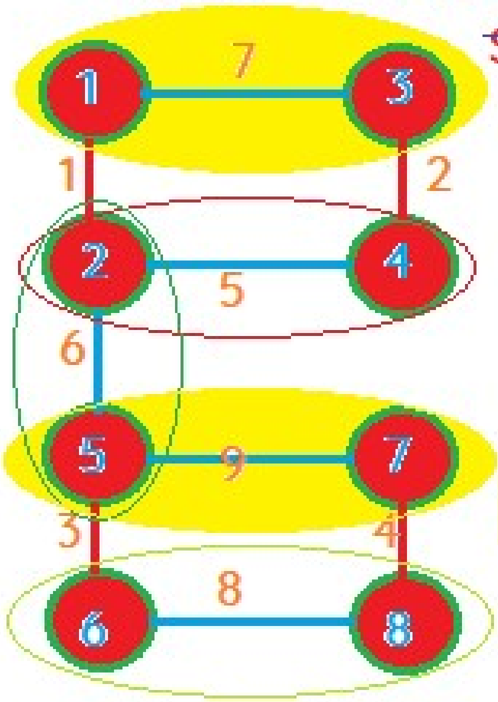
(2) Detecting Cycles

U={1,2,3,4,5,6,7,8}

~~S1={1,2}~~, ~~S2={3,4}~~, ~~S3={5,6}~~, ~~S4={7,8}~~

~~S5={1,2,3,4}~~

~~S6={1,2,3,4,5,6}~~



1 & 3 belongs to S6 (same set), it means there is a cycle. So, do not consider or include it.

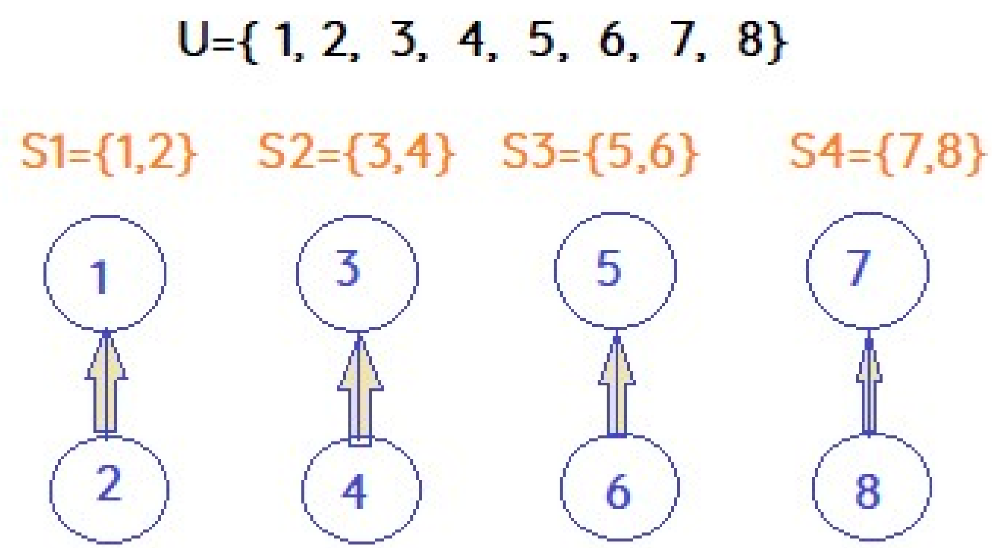
Continue...

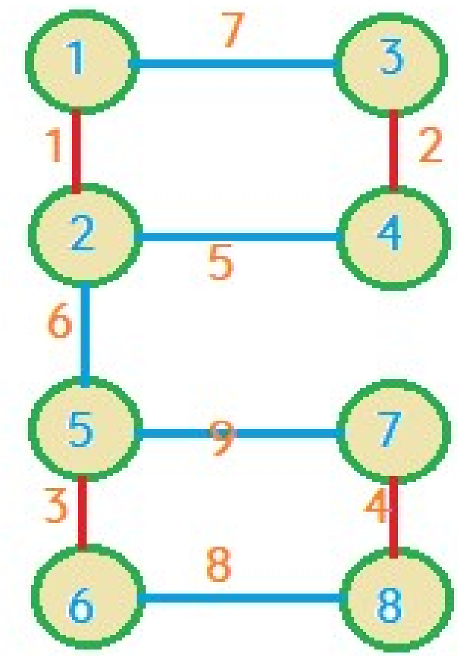
Consider (6, 8)
6 is in S6 and 8 is in S4, So perform union

S7={1,2,3,4,5,6,7,8}

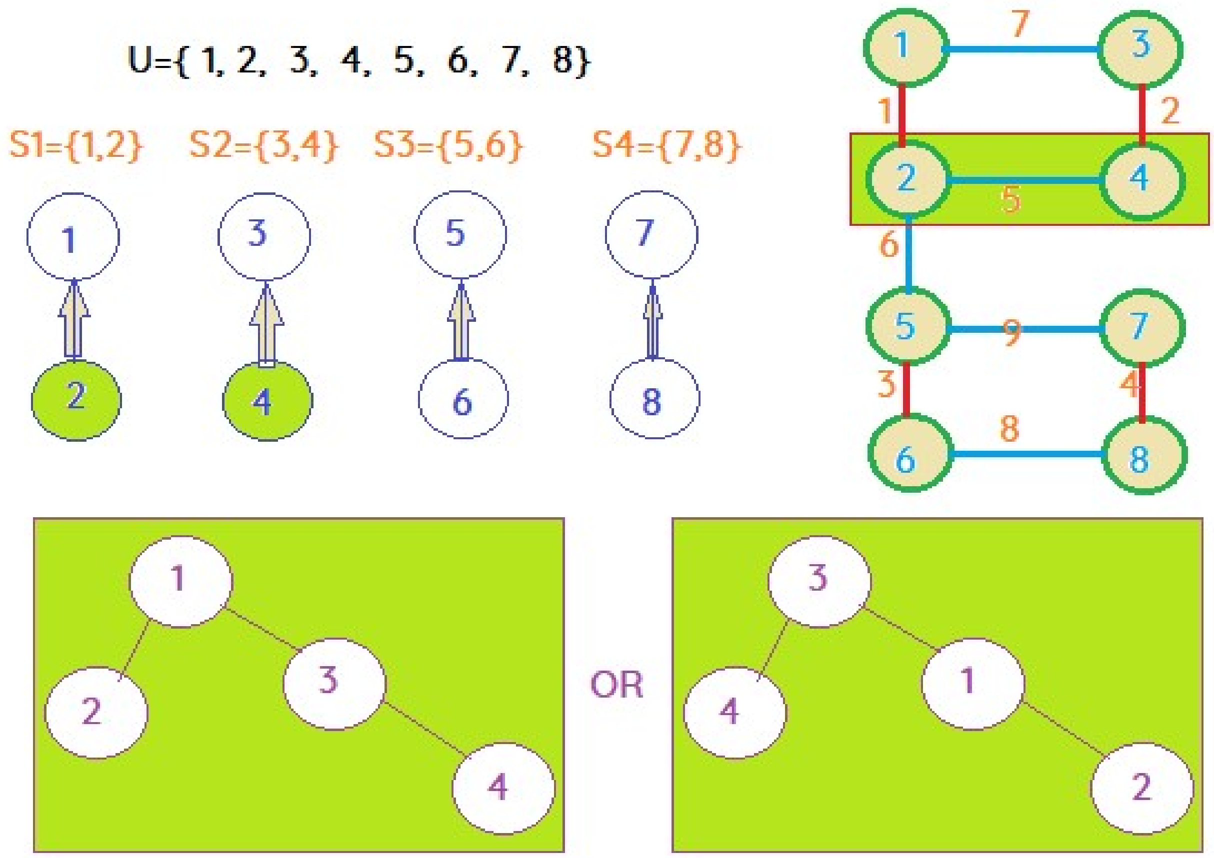
Consider (5,7)
5 & 7 are in same set S7, so there is a cycle.

(3) Graphical Representation

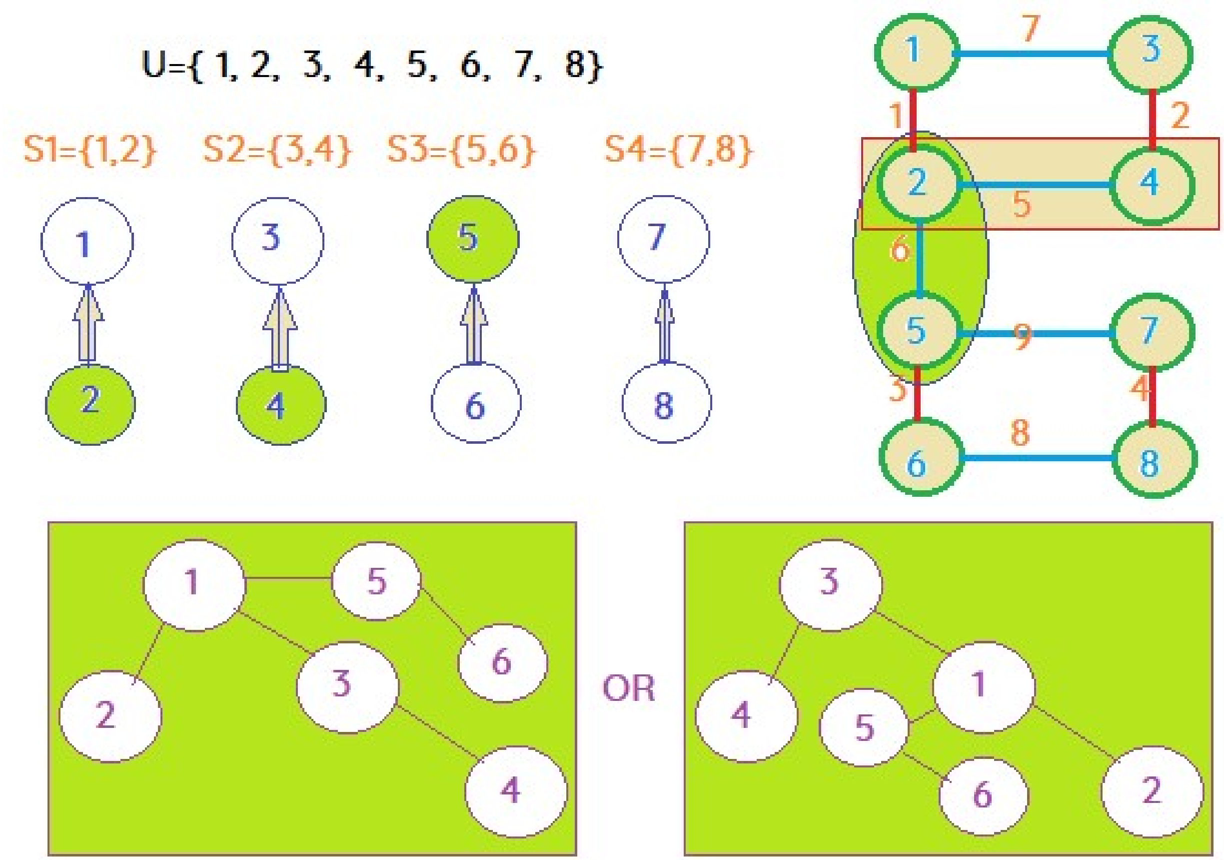




(3) Graphical Representation



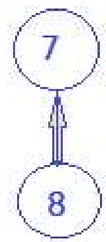
(3) Graphical Representation

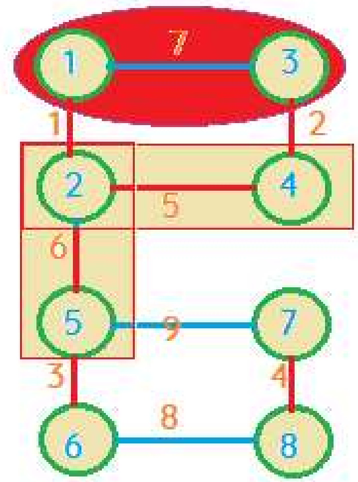


(3) Graphical Representation

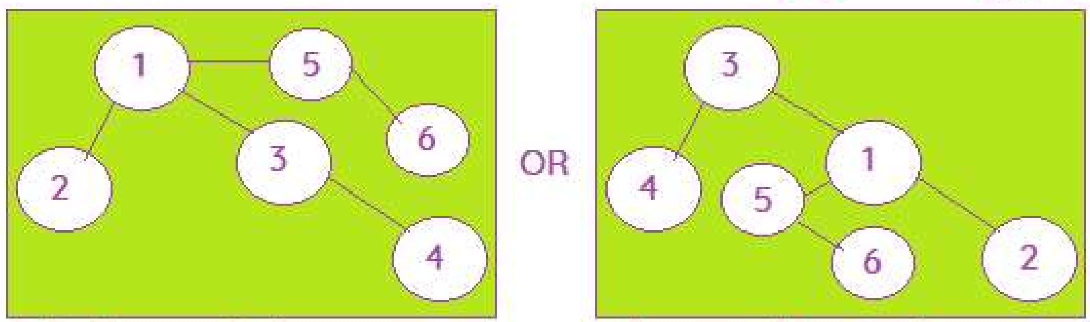
U={ 1, 2, 3, 4, 5, 6, 7, 8}

S1={1,2} S2={3,4} S3={5,6} S4={7,8}





Not Included (1,3)



We leave it here as we now know the cycle detection from Graph

Union-Find Algorithm

Parents itself for each element

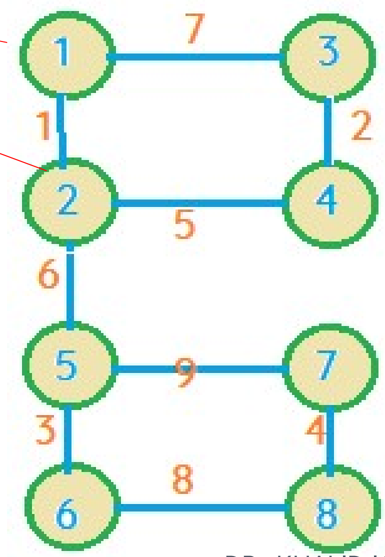
U	1	2	3	4	5	6	7	8
Parent	-1	-1	-1	-1	-1	-1	-1	-1
Vertex	1	2	3	4	5	6	7	8

Union-Find Algorithm

U	1	2	3	4	5	6	7	8
Parent	-1	-1	-1	-1	-1	-1	-1	-1
Vertex	1	2	3	4	5	6	7	8

Parents itself for each element

Parent of 1 and 2 can be found in a constant time due to -1 (shows 1 & 2 are Parents of itself.

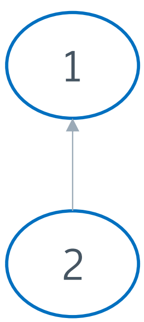


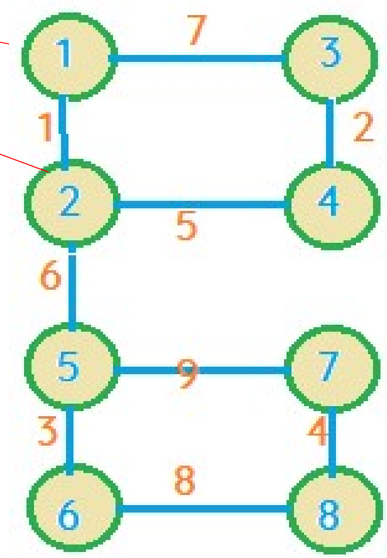
Union-Find Algorithm

Parents itself for each element

U	1	2	3	4	5	6	7	8
Parent	-2	1	-1	-1	-1	-1	-1	-1
Vertex	1	2	3	4	5	6	7	8

Update Parent of 2 to 1 after performing Union Operation.
-2 represents that there are two nodes and 1 is parent of itself.



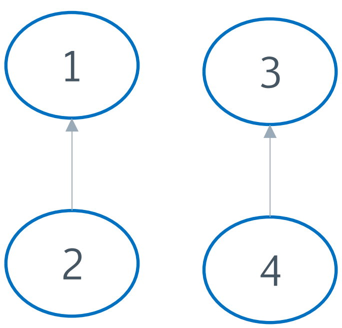


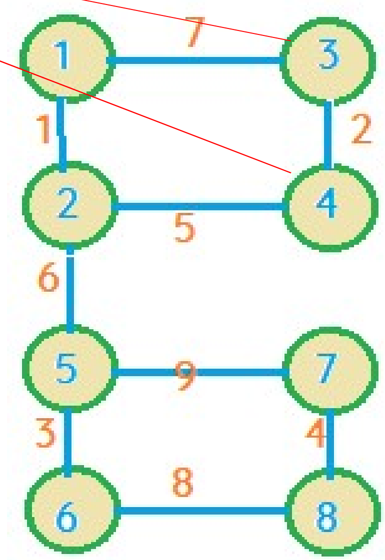
Union-Find Algorithm

Parents itself for each element

U	1	2	3	4	5	6	7	8
Parent	-2	1	-2	3	-1	-1	-1	-1
Vertex	1	2	3	4	5	6	7	8

Update Parent of 4 to 3 after performing Union Operation. -2 represents that there are two nodes and 3 is parent of itself.



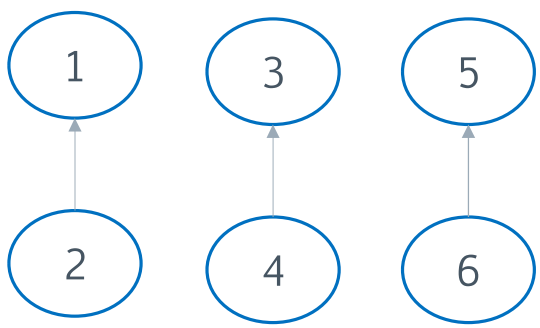


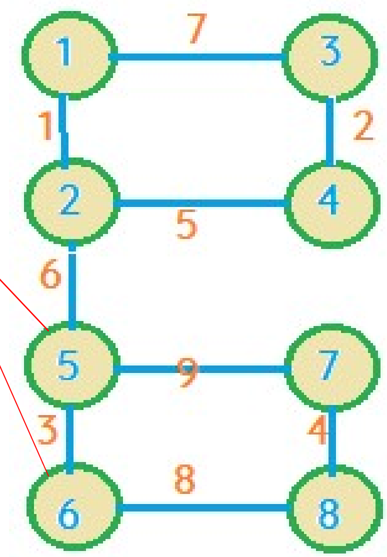
Union-Find Algorithm

Parents itself for each element

U	1	2	3	4	5	6	7	8
Parent	-2	1	-2	3	-2	5	-2	7
Vertex	1	2	3	4	5	6	7	8

Similar process for (5,6) and (7,8)



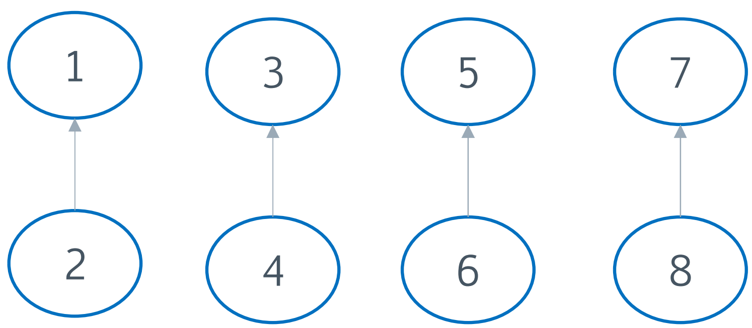


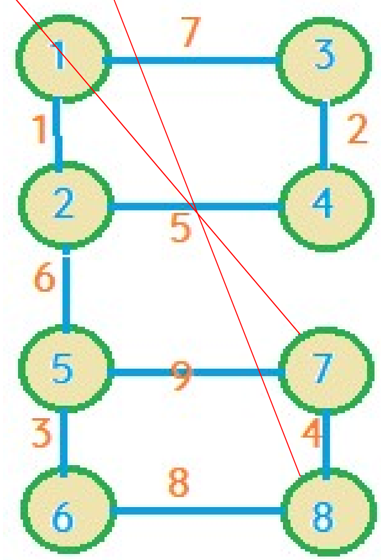
Union-Find Algorithm

Parents itself for each element

U	1	2	3	4	5	6	7	8
Parent	-2	1	-2	3	-2	5	-2	7
Vertex	1	2	3	4	5	6	7	8

Similar process for (5,6) and (7,8)



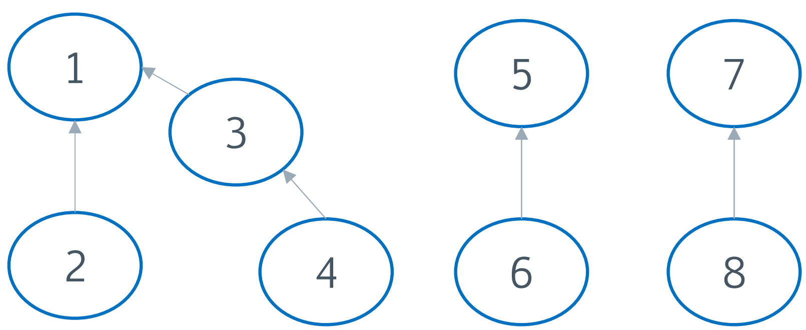


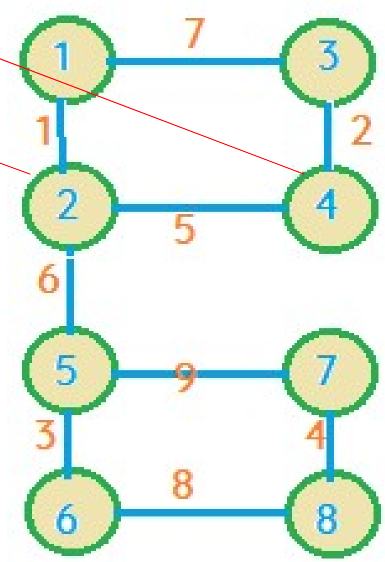
Union-Find Algorithm

Parents itself for each element

U	1	2	3	4	5	6	7	8
Parent	-4	1	1	3	-2	5	-2	7
Vertex	1	2	3	4	5	6	7	8

Consider (2,4) – Parent of 2 is 1 and 1 is parent of itself. Also, parent of 4 is 3 and 3 is parent of itself. Different Parents then perform Union – Select 1 as a parent.





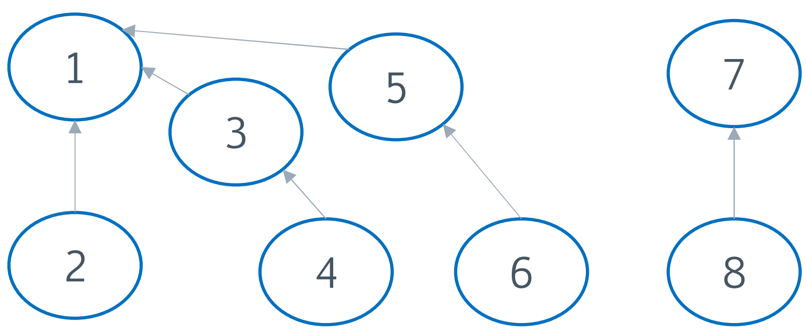
Union-Find Algorithm

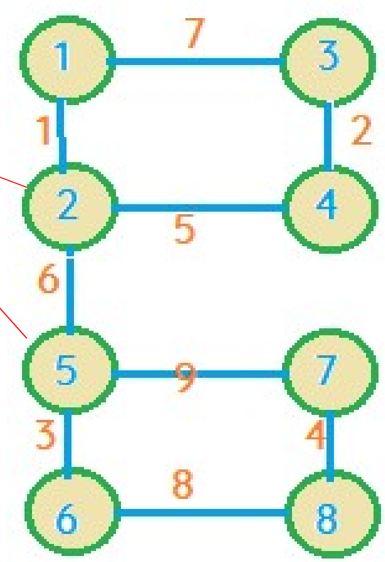
Parents itself for each element

U	1	2	3	4	5	6	7	8
Parent	-6	1	1	3	1	5	-2	7
Vertex	1	2	3	4	5	6	7	8

Consider (2,5) – Parent of 2 is 1 and 1 is parent of itself. Also, 5 itself is a parent.

Different Parents then perform Union – Select 1 as a parent due to its high rank or weight.



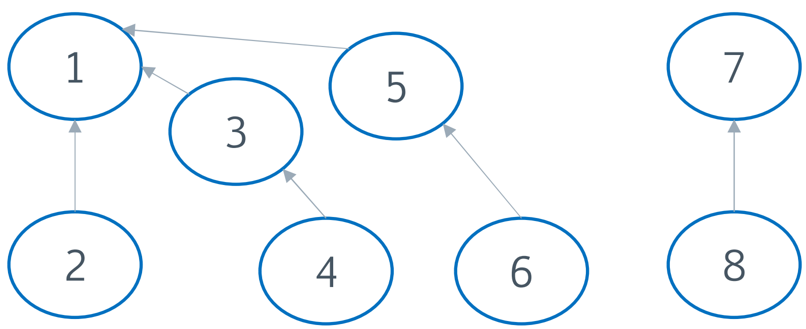


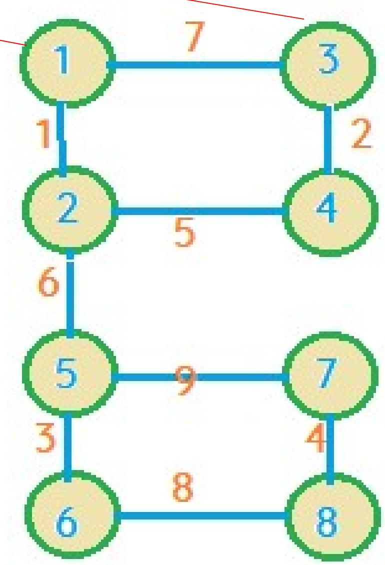
Union-Find Algorithm

Parents itself for each element

U	1	2	3	4	5	6	7	8
Parent	-6	1	1	3	1	5	-2	7
Vertex	1	2	3	4	5	6	7	8

Consider (1,3) – 1 is parent of itself. Parent of 3 is also 1. SAME PARENT. It means that it's a CYCLE. Do not include it.



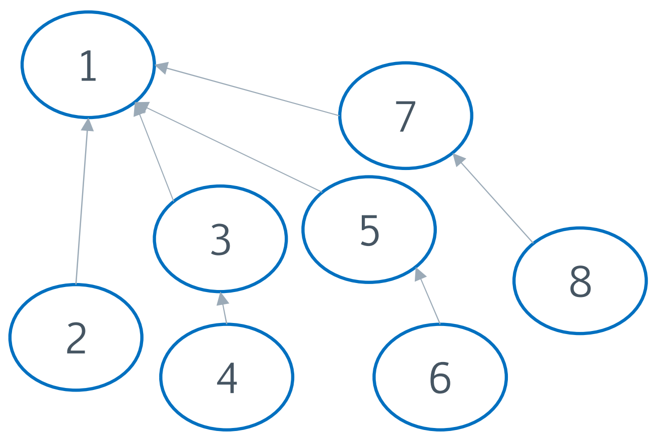


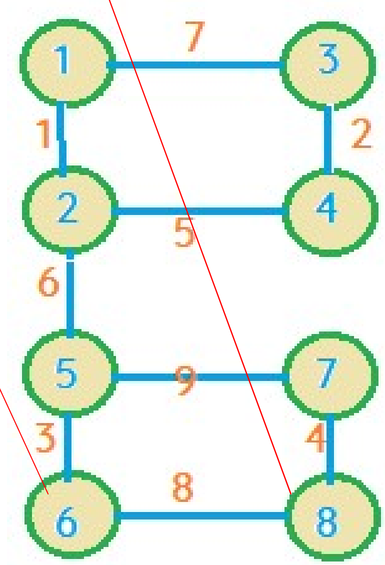
Union-Find Algorithm

Parents itself for each element

U	1	2	3	4	5	6	7	8
Parent	-8	1	1	3	1	5	1	7
Vertex	1	2	3	4	5	6	7	8

Consider (6,8) – 6 parent is 1. Parent of 8 is also 7. Different PARENT. Perform UNION Operation.





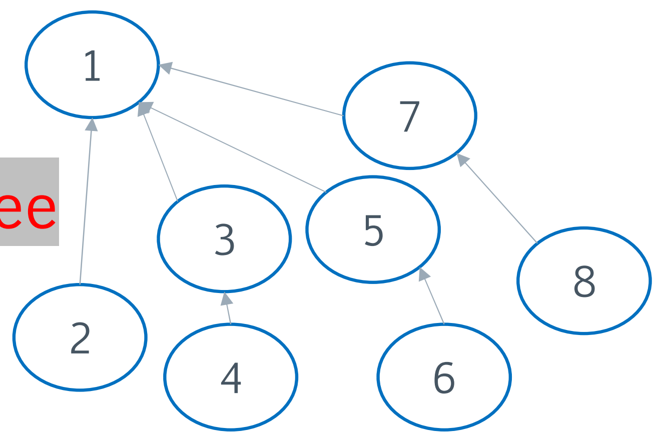
Union-Find Algorithm

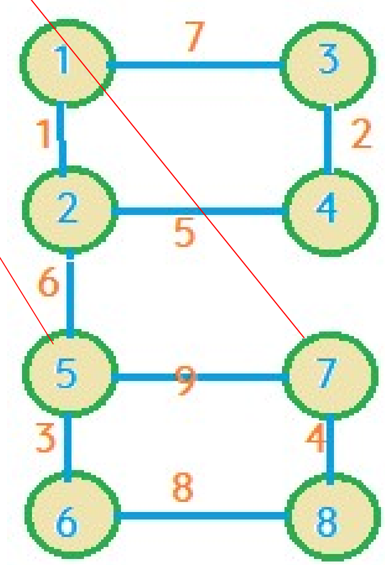
Parents itself for each element

U	1	2	3	4	5	6	7	8
Parent	-8	1	1	3	1	5	1	7
Vertex	1	2	3	4	5	6	7	8

Consider (5,7) – 5 parent is 1. Parent of 7 is also 1. SAME PARENT. It's a CYCLE.

Collapsing in the tree



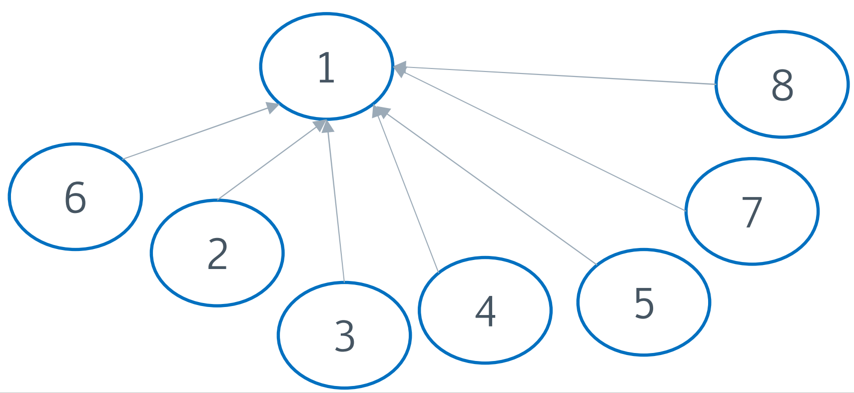


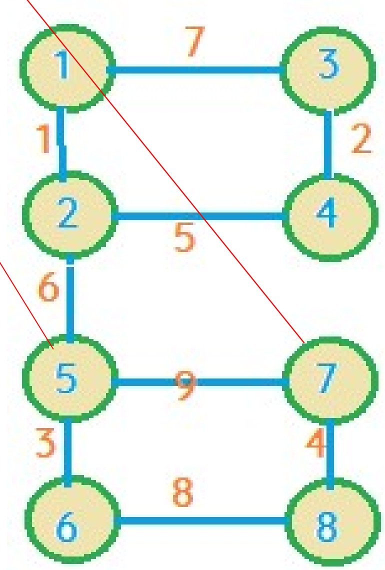
Union-Find Algorithm

Parents itself for each element

U	1	2	3	4	5	6	7	8
Parent	-8	1	1	1	1	1	1	1
Vertex	1	2	3	4	5	6	7	8

Consider (5,7) – 5 parent is 1. Parent of 7 is also 1. SAME PARENT. It's a CYCLE.





Collapsing in the tree: If we came to know the parent is the root

Union-Find Algorithm

```
function MakeSet(x)
    x.parent = x
    x.rank = 0

function Union(x, y)
    xRoot = Find(x)
    yRoot = Find(y)
    if xRoot == yRoot
        return

    // x and y are not already in the same set. Merge them.
    if xRoot.rank < yRoot.rank
        xRoot.parent = yRoot
    else if xRoot.rank > yRoot.rank
        yRoot.parent = xRoot
    else
        yRoot.parent = xRoot
        xRoot.rank = xRoot.rank + 1
```

MAKE-SET(x)
$x.p = x$
$x.rank = 0$

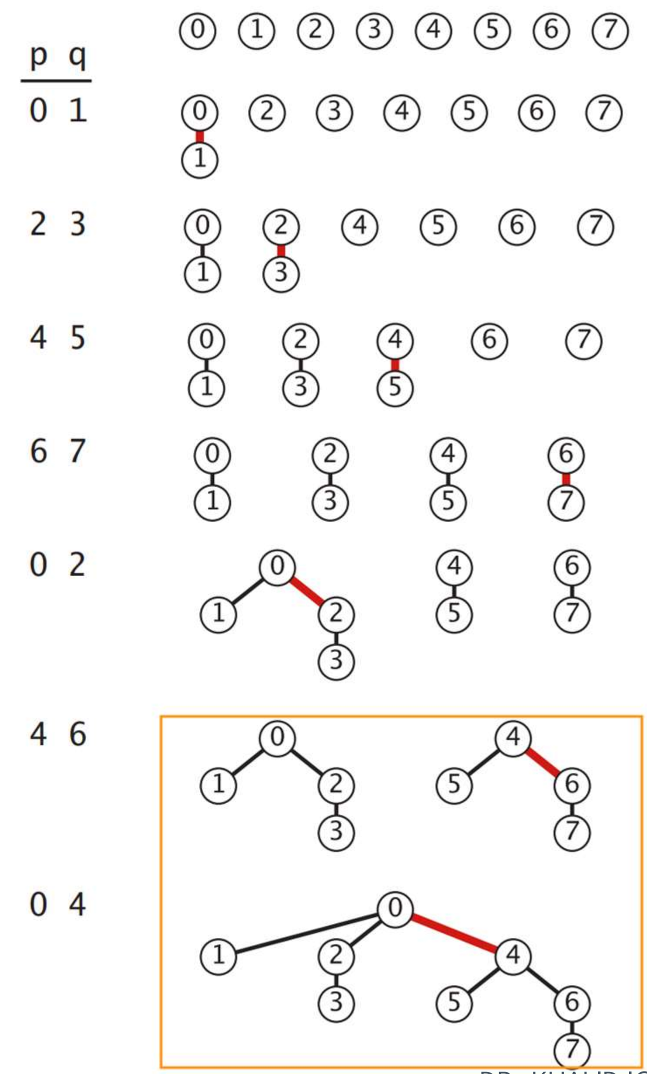
UNION(x, y)
LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
if $x.rank > y.rank$
    $y.p = x$
else $x.p = y$
    if $x.rank == y.rank$
        $y.rank = y.rank + 1$

FIND-SET(x)
if $x \neq x.p$
    $x.p =$ FIND-SET($x.p$)
return $x.p$

It is possible to create a **tree of depth n - 1** (Skew Trees). The worst-case **running time** of a **FIND is O(n)** and **m consecutive FIND operations** take **O(mn)** time in the **worst case.**

worst-case input



Thank You!!!

Have a good day