Intro to the problem :

The efficiency of an algorithm sometimes depends on using an efficient data structure. A good choice of data structure can reduce the execution time of an algorithm and Union-Find is a data structure that falls in that category.

Let’s say, you have a set of N elements which are partitioned into further subsets, and you have to keep track of connectivity of each element in a particular subset or connectivity of subsets with each other. To do this operation efficiently, you can use Union-Find Data Structure.

Let’s say there are 5 people A, B, C, D E. A is a friend of B, B is a friend of C and D is a friend of E. As we can see:

1) A, B and C are connected to each other.
2) D and E are connected to each other.

So we can use Union Find Data Structure to check whether one friend is connected to another in a direct or indirect way or not. We can also determine the two different disconnected subsets. Here 2 different subsets are {A, B, C} and {D, E}.

You have to perform two operations here :

Union (A, B) - connect two elements A and B. Find (A, B) - find, is there any path connecting two elements A and B

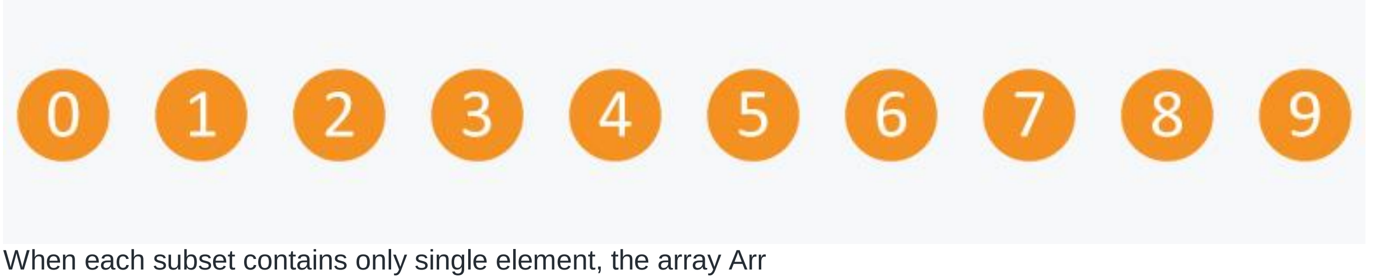
Example: You have a set of elements S = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Here you have 10 elements (N = 10).We can use an array **Arr** to manage the connectivity of elements. Arr[ ] indexed by elements of set, having size of N (as N elements in set) and can be used to manage the above operations.

Assumption: A and B objects are connected only if Arr[A] = Arr [B].

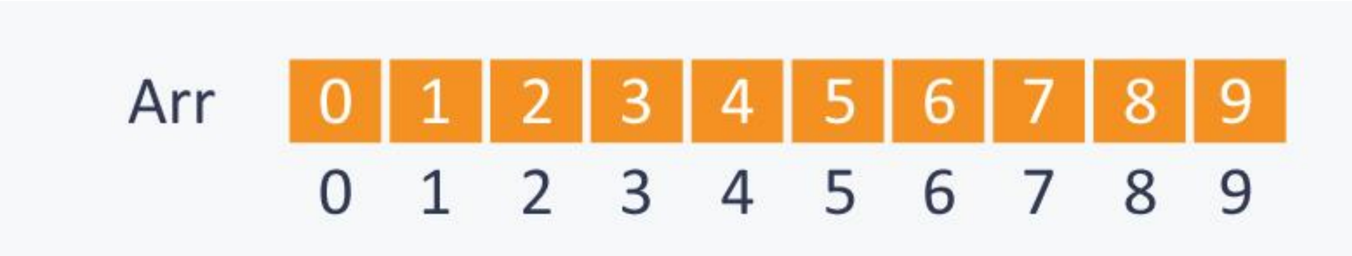
Now how we will implement above operations :

Find (A, B) - check if Arr[A] is equal to Arr[B] or not. Union (A, B) - Connect A to B and merge the components having A and B by changing all the elements ,whose value is equal to Arr[A], to Arr[B].

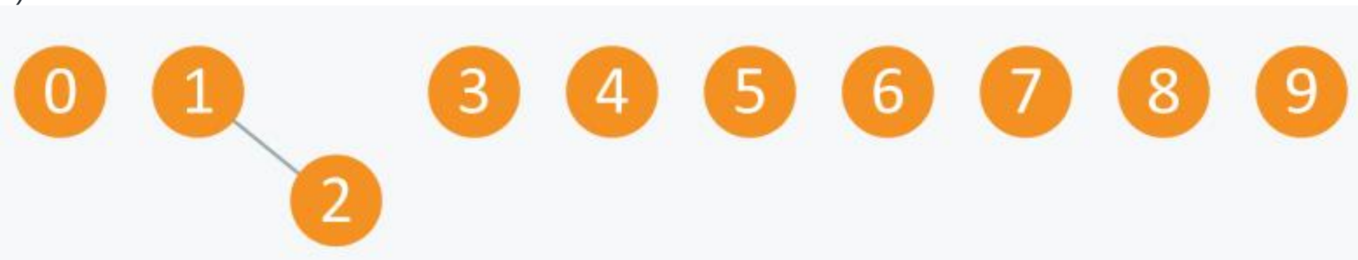
Initially there are 10 subsets and each subset has single element in it.



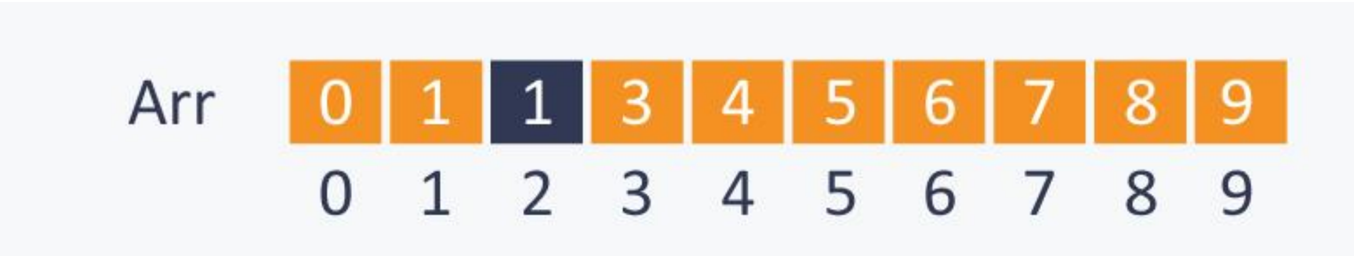
When each subset contains only single element, the array Arr is:



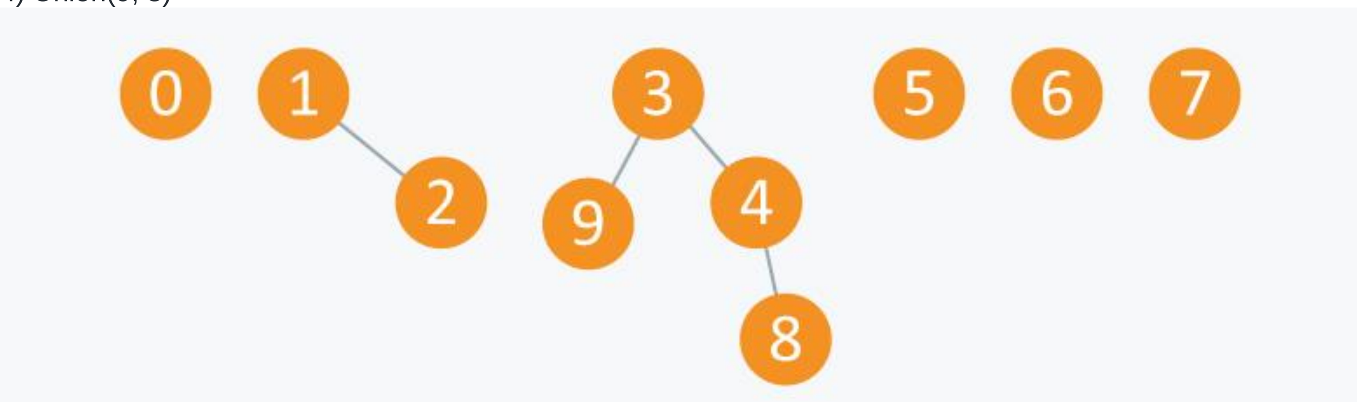
Let’s perform some Operations: 1) Union(2, 1)



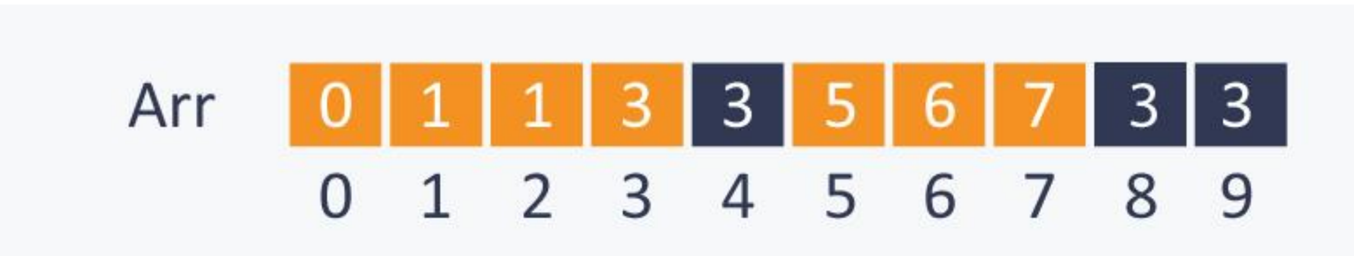
Arr will be:



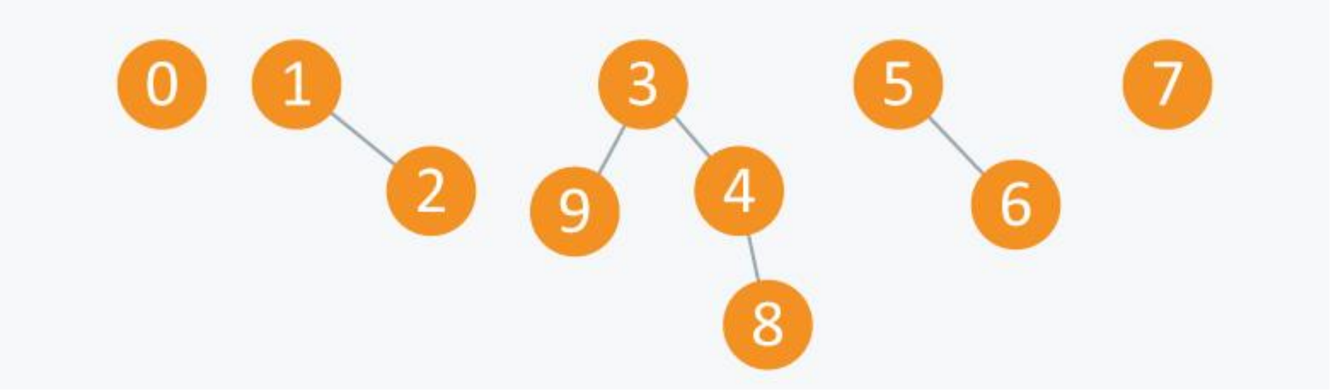
2) Union(4, 3)
3) Union(8, 4)
4) Union(9, 3)



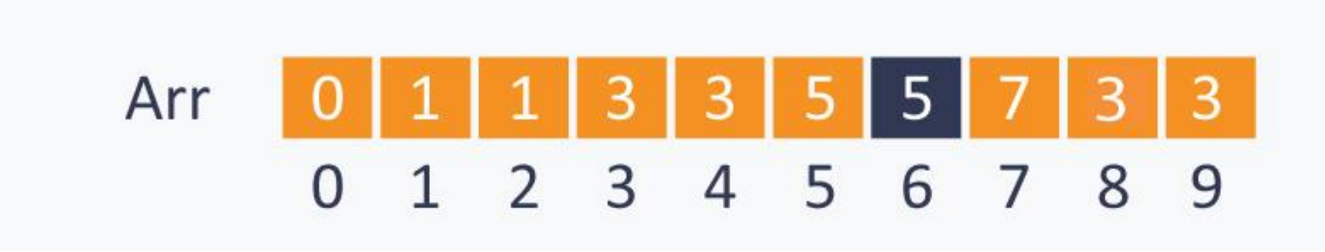
Arr will be:



5) Union(6, 5)



Arr will be:



After performing some operations of Union(A ,B), you can see that now there are 5 subsets. First has elements {3, 4, 8, 9}, second has {1, 2}, third has {5, 6}, fourth has {0} and fifth has {7}. All these subsets are said to be Connected Components.

One can also relate these elements with nodes of a graph. The elements in one subset can be considered as the nodes of the graph which are connected to each other directly or indirectly, therefore each subset can be considered as **connected component.**

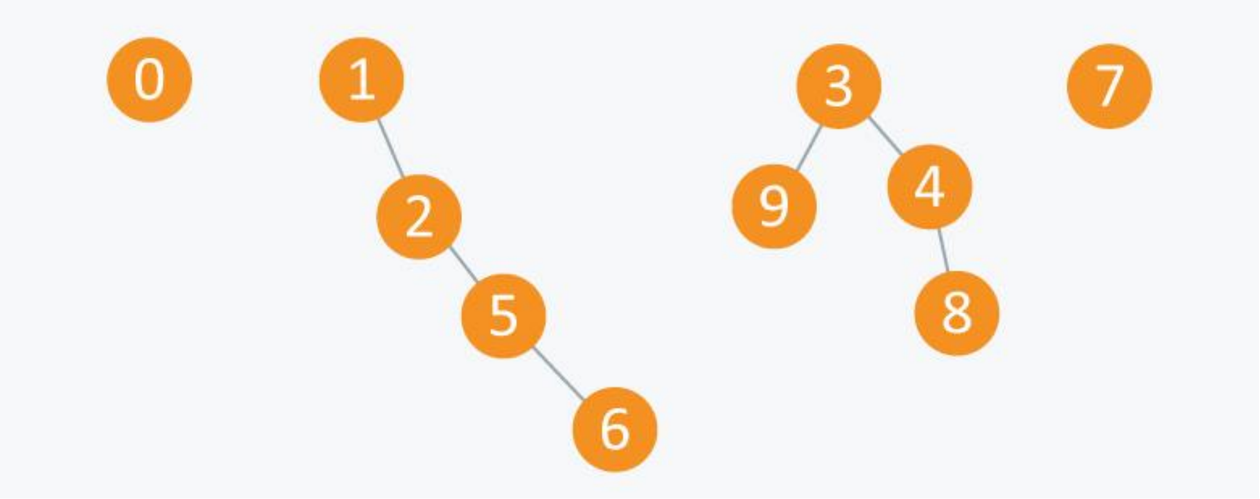
From this, we can infer that Union-Find data structure is useful in Graphs for performing various operations like connecting nodes, finding connected components etc.

Let's perform some Find(A, B) operations. 1) Find (0, 7) - as 0 and 7 are disconnected ,this will gives false result.
2) Find (8, 9) -though 8 and 9 are not connected directly ,but there exist a path connecting 8 and 9, so it will give us true result.

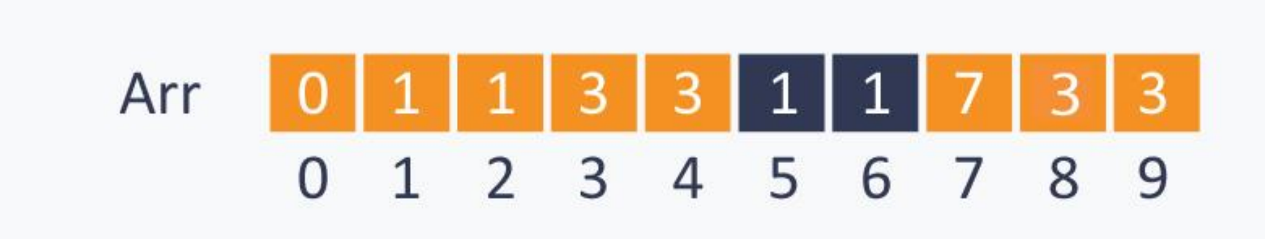
When we see above operations in terms of components, then :

Union(A, B) - Replace components containing two objects A and B with their union.
Find(A, B) - check if two objects A and B are in same component or not.

So if we perform operation Union(5, 2) on above components, then it will be :



Now the Arr will be:



Application :

- Network connectivity.
- Percolation.
- Image processing.
- Least common ancestor.
- Equivalence of finite state automata.
- Hinley-Milner polymorphic type inference.
- Kruskal's minimum spanning tree algorithm.
- Games (Go, Hex).
- Compiling equivalence statements in Fortran.

Reference :

https://www.hackerearth.com/practice/notes/disjoint-set-union-union-find/