University Of Rochester

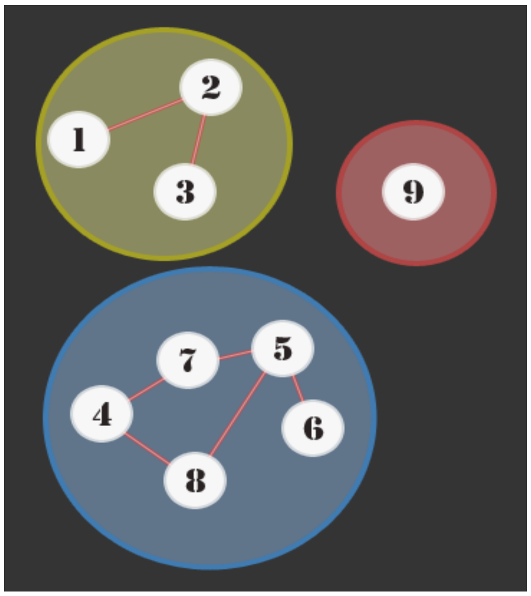
Rajat Kuthiala

CSCS 172

April 13, 2016

Data Structure: Disjoint Sets

"Two sets A_1 and A_2 are disjoint if their intersection $A_1 \bigcap A_2 \equiv \emptyset$, where $\emptyset$ is the empty set. n sets A_1, A_2, ..., A_n are disjoint if $A_i \bigcap A_j \equiv \emptyset$ for $i \neq j$. For example, $\{A, B, C\}$ and $\{D, E\}$ are disjoint, but $\{A, B, C\}$ and $\{C, D, E\}$ are not. Disjoint sets are also said to be mutually exclusive or independent." –[*http://mathworld.wolfram.com/DisjointSets.html]* meaning that no item is more than one set. Eg:

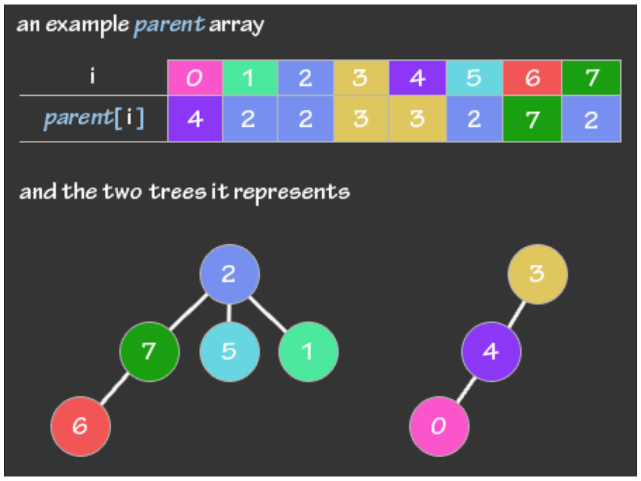


Disjoint Sets

The disjoint set data structure allows us to very quickly determine if two items are in the same set and also to very quickly unite two sets. This data structure is used by the Boost Graph Library to implement its Incremental Connected Components functionality. It is also used for implementing Kruskal's algorithm to find the minimum spanning tree of a graph. Another application of Disjoint data structure is during the computation of relation where the related objects unionize in same set. Two useful operation of disjoint algorithm are:

- **Find:** Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.
- **Union:** Join two subsets into a single subset.

In its simplest form, disjoint data structure is just an array of integers, called **parent**. If we are dealing with **n** items, then parent is an array of size n, and the **i**th element of the array represents the **i**th item. These relationships create one, or more than one virtual trees.



i	0	1	2	3	4	5	6	7
parent[i]	4	2	2	3	3	2	7	2

Each tree is a disjoint set. If two elements are in the same tree, then they are in the same disjoint set. The root node (or the topmost node) of each tree is called the representative of the set. There is always a single unique representative of each set. As you can see, if **I** is the representative of a set, then **parent[i] = i.** If **I** is not the representative of the set, then it can be found by travelling up the tree until we find the representative. This is how the **find** operation work of the Data Structure.

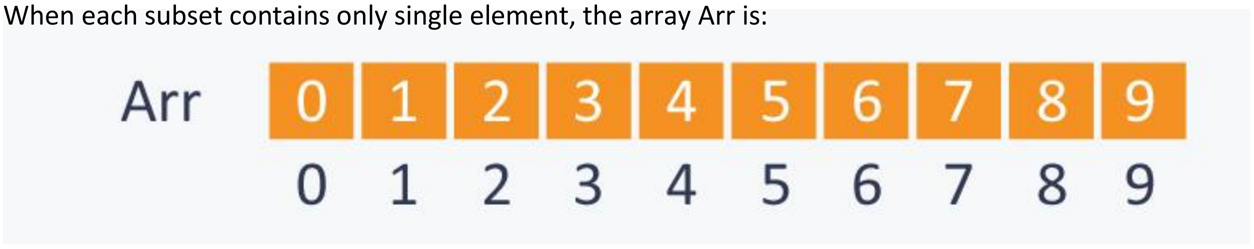
As for the implementation of **Union** operation, it takes two elements as input and then find their representative using find operation. Then it finally puts either one of the trees under the root node of the other tree, effectively merging the trees and the sets.

Let’s see how it works:

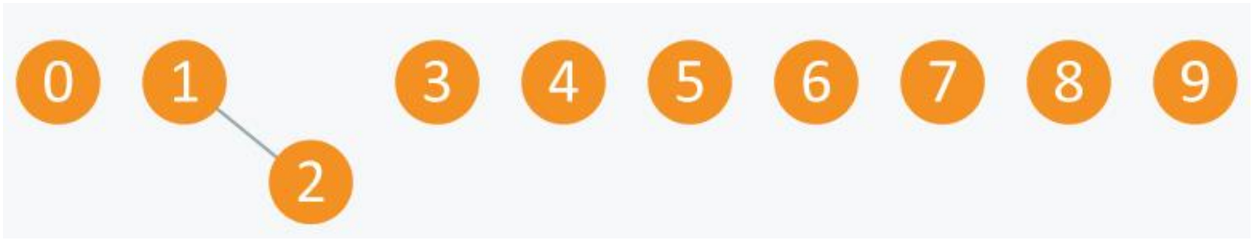
Suppose there are 10 subsets and each subset has single element in it.

0
1
2
3
4
5
6
7
8
9

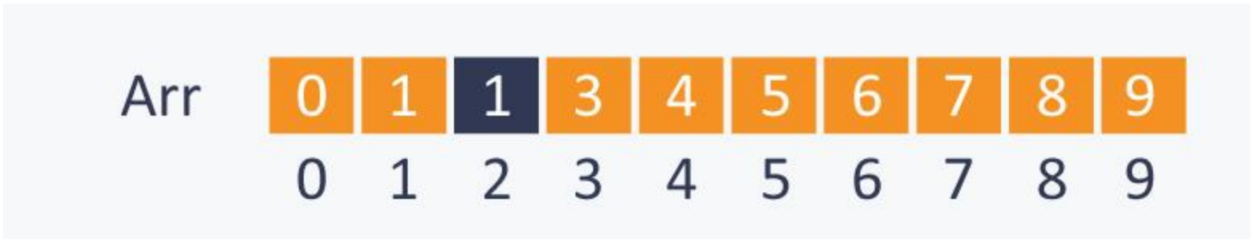
When each subset contains only single element, the array Arr is:



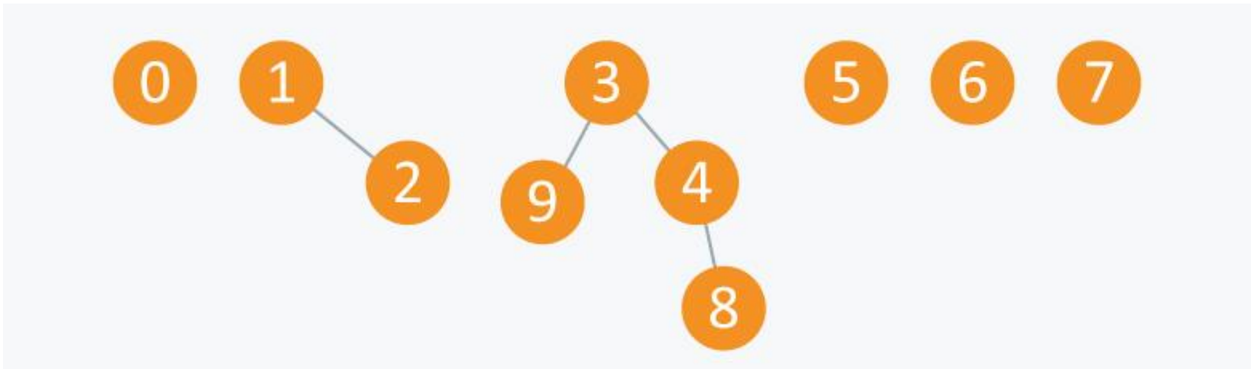
After performing an Union operation (e.g.: Union(2,1)), the elements will look something like:



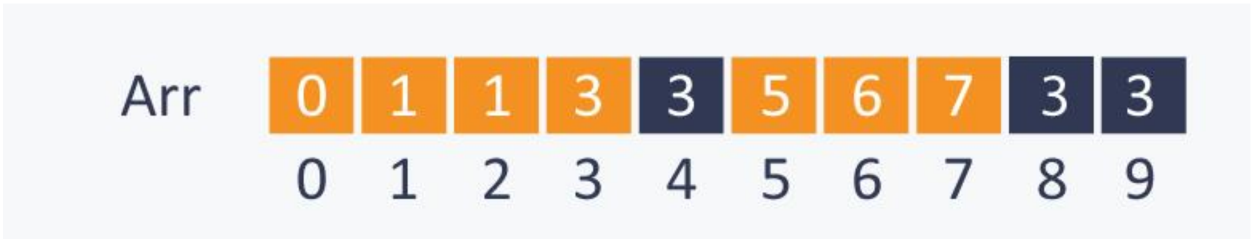
And the array Arr looks like:



Performing more operations(Union(4,3), Union(8,4), Union(9,3)), we get elements:



And the array Arr will transform to:



We 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 Disjoint Data Structure is useful in graphs for performing various operations like connecting nodes, finding connected components, etc.

As for the running time analysis, each time the depth of a node i increases by 1, the size of the tree containing i is at least doubled. Thus, running time of each operation is log n. The average over a sequence of operation is almost constant, that is, a sequence of M union/find operation takes a total of O(M) time on average.