Aditya Shirode, Rhythm Shah, Shrenuj Gandhi

3. [5 points]

   Suppose that you wish to add, to the operations of a disjoint set forest, the operation PRINT-SET(x), which prints all the elements of the set containing x, in any order. Give enough details to show that we can add a single attribute to each node so that the time for PRINT-SET(x) is linear in the number of elements printed and all other operations are unchanged. In particular it should take constant time to print each element.

Answer:
In order to achieve PRINT-SET(x) operation,
we add an attribute $c[]$ which stores the list of neighbours.
This list is updated every time the LINK(x, y) function is called.

Thus the new LINK() function is as follows :

```
LINK(x, y)
        $p[x] \rightarrow y$
        $c[x] \rightarrow c[x] \cup y$          ▷ achieved in constant time
        $c[y] \rightarrow c[y] \cup x$          ▷ achieved in constant time
```

This change does not affect the growth class of LINK() function.

Now, when PRINT-SET() is called on node x,
we can use either BFS or DFS to traverse and print the nodes.

Moreover, we know that both BFS and DFS take O($v + e$), where v is #Vertices and e is #Edges. This implies that using BFS or DFS on disjoint set in the worst case would take O(n) as $v = n$ and $e = n - 1$ in worst case.

Let's assume the following operations are to be performed on a completely disjoint set with nodes $1, 2, 3, 4$.

```
Union(1,2)
Union(3,4)
Find(1)
Union(1,4)
Print-Set(2)
```

We have added the attribute $c[]$, which will store neighbours of node i.
After performing $Union(1, 4)$, we have the data structure contents as follows -

p[i]	4	2	4	4
i	1	2	3	4
c[i]	[4,2]	[1]	[4]	[1,3]

Now we encounter PRINT-SET(2).
$Print - Set(2)$ would call a BFS (Or DFS) on 2.
We go to 2. Print it (print 2). Then insert contents of it's adjacent list $c[2]$ in queue.

Output: 2

We get 1 from the queue and we print it. We insert adjacent list $c[1] = [4, 2]$ in queue.

Output: $2, 1$

We move on to next in queue, 4, print 4, find out it's adjacency list ($c[4] = [1, 3]$) and insert it in queue.

Output: $2, 1, 4$

We move on to 2, but it has already been visited. Next up in queue is 1, which also has been visited. So in comes 3. We print 3, and enter $c[3] = [4]$ in queue.

Output: $2, 1, 4, 3$

We see that 4 has already been visited, we move on. As no more nodes to be visited, we return.

Thus, we have seen that we can visit and print all the nodes in linear time by using the adjacency list $c[]$ in a disjoint set.