DISJOINT SET MANUPULATION AND RECAPITULATION

CSE [AIML], 6TH SEMESTER

DESIGN & ANALYSIS OF ALGORITHM

SET & DISJOINT SET

- Set: set is a collection of distinct elements. The Set can be represented, for examples, as S1={1,2,5,10}.
- Disjoint Sets: The disjoints sets are those do not have any common element. For example S1={1,7,8,9} and S2={2,5,10}, then we can say that S1 and S2are two disjoint sets.

DISJOINT SET OPERATIONS

The disjoint set operations are

1. Union
2. Find

Disjoint set Union:

If Si and Sj are two disjoint sets, then their union Si U Sj consists of all the elements x such that x is in Si or Sj.

Example:

S1={1,7,8,9} S2={2,5,10}

S1 U S2={1,2,5,7,8,9,10}

DISJOINT SET OPERATIONS CONTINUES…

Find: Given the element I, find the set containing I.

Example:

S1={1,7,8,9}

Then,

S2={2,5,10} S3={3,4,6}

Find(4)=S3 Find(5)=S2 Find(7)=S1

SET REPRESENTATION

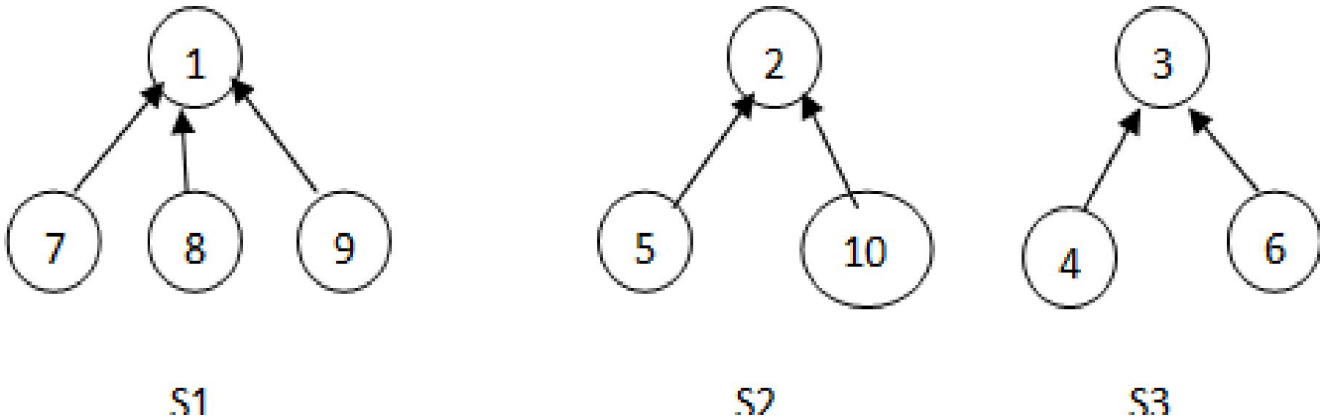
The set will be represented as the tree structure where all children will store the address of parent / root node. The root node will store null at the place of parent address. In the given set of elements any element can be selected as the root node, generally we select the first node as the root node.

SET REPRESENTATION CONTINUES…

Example:

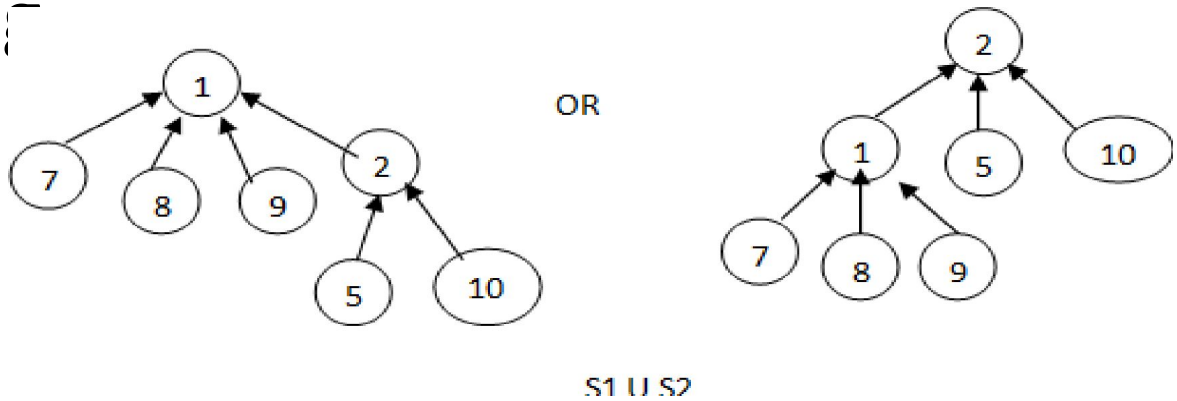
S1={1,7,8,9} S2={2,5,10} s3={3,4,6}

Then these sets can be represented as



DISJOINT UNION

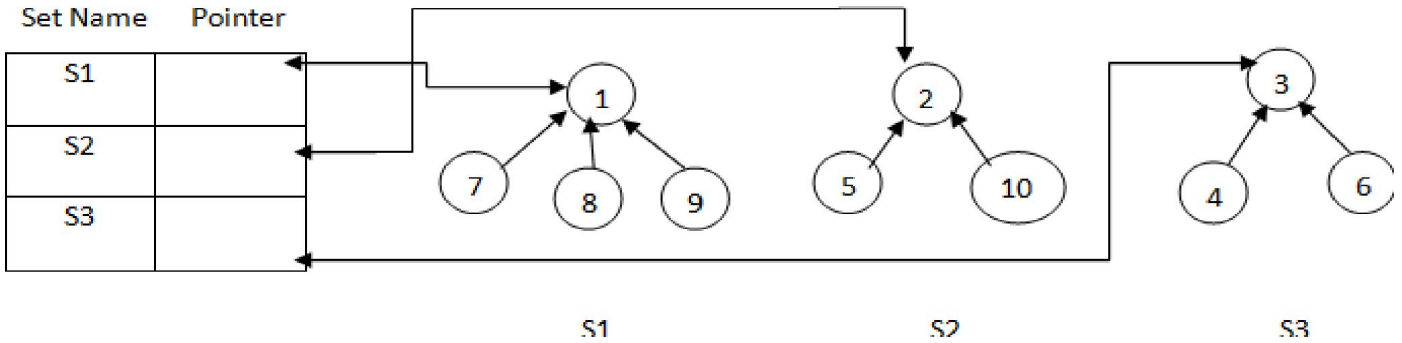
To perform disjoint set union between two sets Si and Sj can take any one root and make it sub-tree of the other. Consider the above example sets S1 and S2 then the union of S1 and S2 can be represented as any one of the following



S1 U S2

FIND

To perform find operation, along with the tree structure we need to maintain the name of each set. So, we require one more data structure to store the set names. The data structure contains two fields. One is the set name and the other one is the pointer to root.



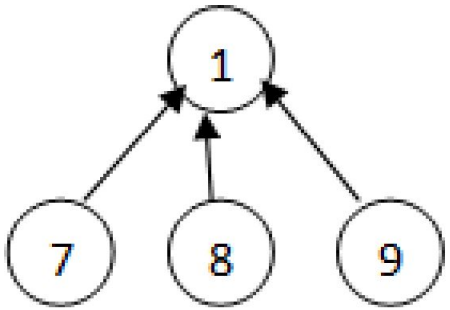
UNION ALGORITHM

To represent the sets, we use an array of 1 to n elements where n is the maximum value among the elements of all sets. The index values represent the nodes (elements of set) and the entries represent the parent node. For the root value the entry will be‘-1’.

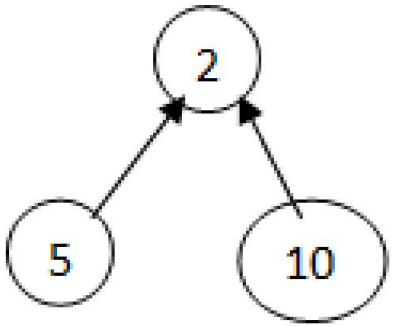
UNION ALGORITHM CONTINUES…

Example:

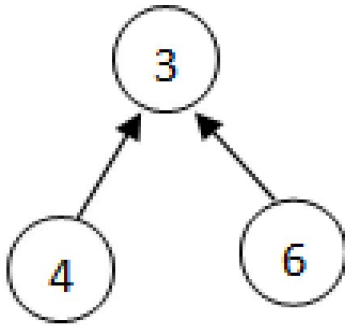
For the following sets the array representation is as shown below.



S1



S2



S3

UNION ALGORITHM CONTINUES…

Algorithm for Union operation:

To perform union the SimpleUnion(i,j) function takes the inputs as the set roots i and j . And make the parent of i as j i.e, make the second root as the parent of first root.

```
Algorithm SimpleUnion(i,j)
{
P[i]:=j;
}
```

ALGORITHM FOR FIND OPERATION

The SimpleFind(i) algorithm takes the element i and finds the root node of i.It starts at i until it reaches a node with parent value-1.

```
Algorithm SimpleFind(i)
{
    while( P[i]≥0)
        i:=P[i];
    return i;
}
```

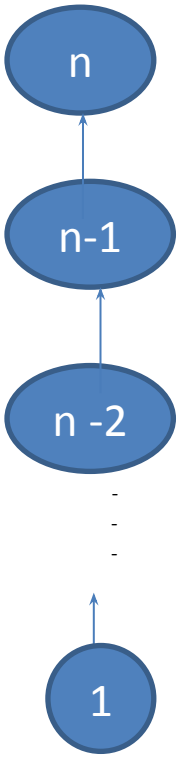
ANALYSIS OF SIMPLE UNION & SIMPLE FIND

Although the SimpleUnion(i,j) and SimpleFind(i) algorithms are easy to state, their performance characteristics are not very good. For example, consider the set – {1, 2, 3, 4, …, n}.

Then if we want to perform following sequence of operations Union(1,2), Union(2,3)……. Union(n-1,n) and sequence of Find(1), Find(2)……Find(n).

ANALYSIS OF SIMPLE UNION & SIMPLE FIND CONTINUES…

The sequence of Union operations results the degenerate tree as below.

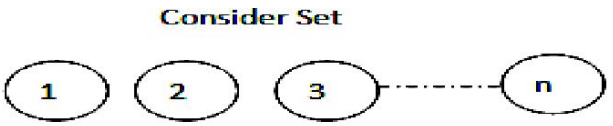


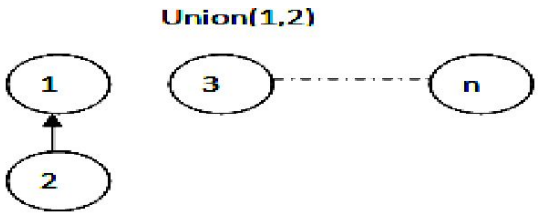
ANALYSIS OF SIMPLE UNION & SIMPLE FIND CONTINUES…

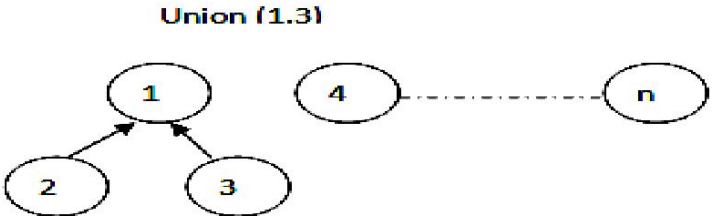
- Since, the time taken for a Union is constant, the n-1 sequence of union can be processed in time O(n).And for the sequence of Find operations it will take O(n^2). We can improve the performance of union and find by avoiding the creation of degenerate tree by applying weighting rule for Union.

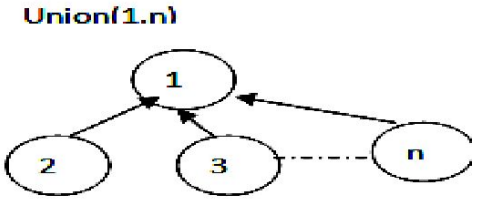
WEIGHTED UNION

If the number of nodes in the tree with root I is less than the number in the tree with the root j, then make ‘j’ the parent of i; otherwise make ‘i' the parent of j.









WEIGHTED UNION CONTINUES…

To implement weighting rule we need to know how many nodes are there in every tree. To do this we maintain “count” field in the root of every tree. If ‘i' is the root then count[i] equals to number of nodes in tree with rooti.

Since all nodes other than roots have positive numbers in parent (P) field, we can maintain count in P field of the root as negative number.

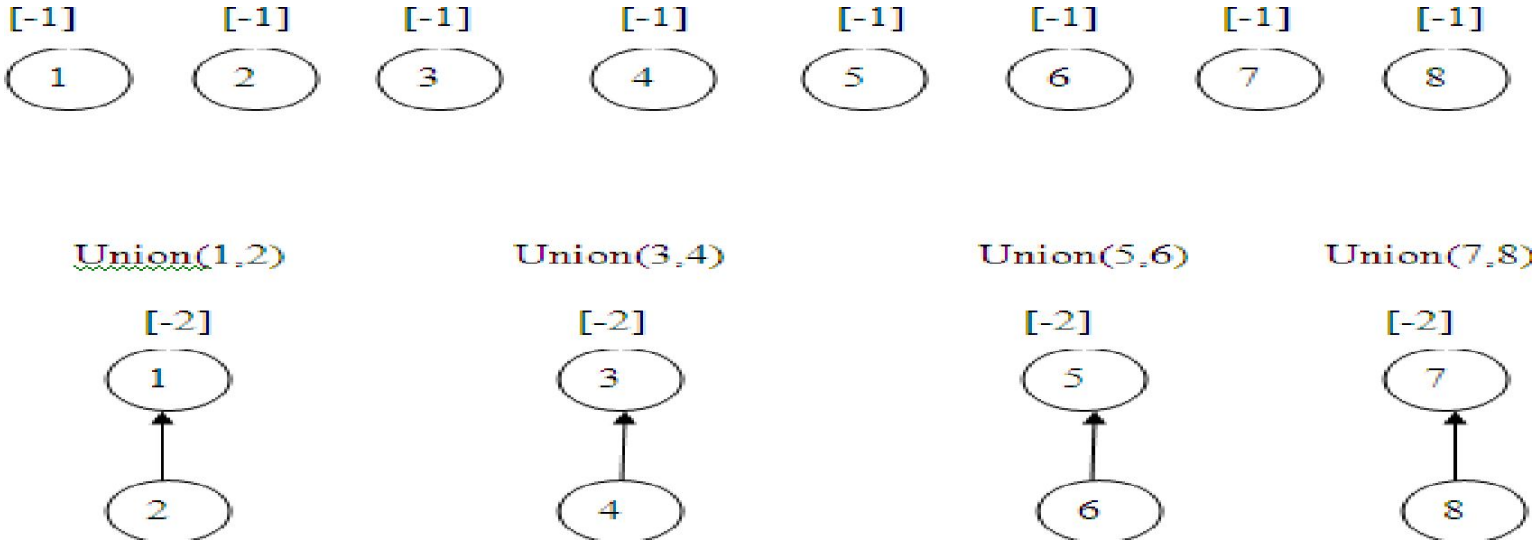
ALGORITHM OF WEIGHTED UNION

```
Algorithm WeightedUnion(i,j)
//Union sets with roots i and j, i≠j using the weighted rule
// P[i]=-count[i] andp[j]=-count[j]
{
temp:=P[i]+P[j];
if (P[i]>P[j])then
{
// i has fewer nodes P[i]:=j;
P[j]:=temp;
}
else
{
// j has fewer nodes P[j]:=i;
P[i]:=temp;
}
}
```

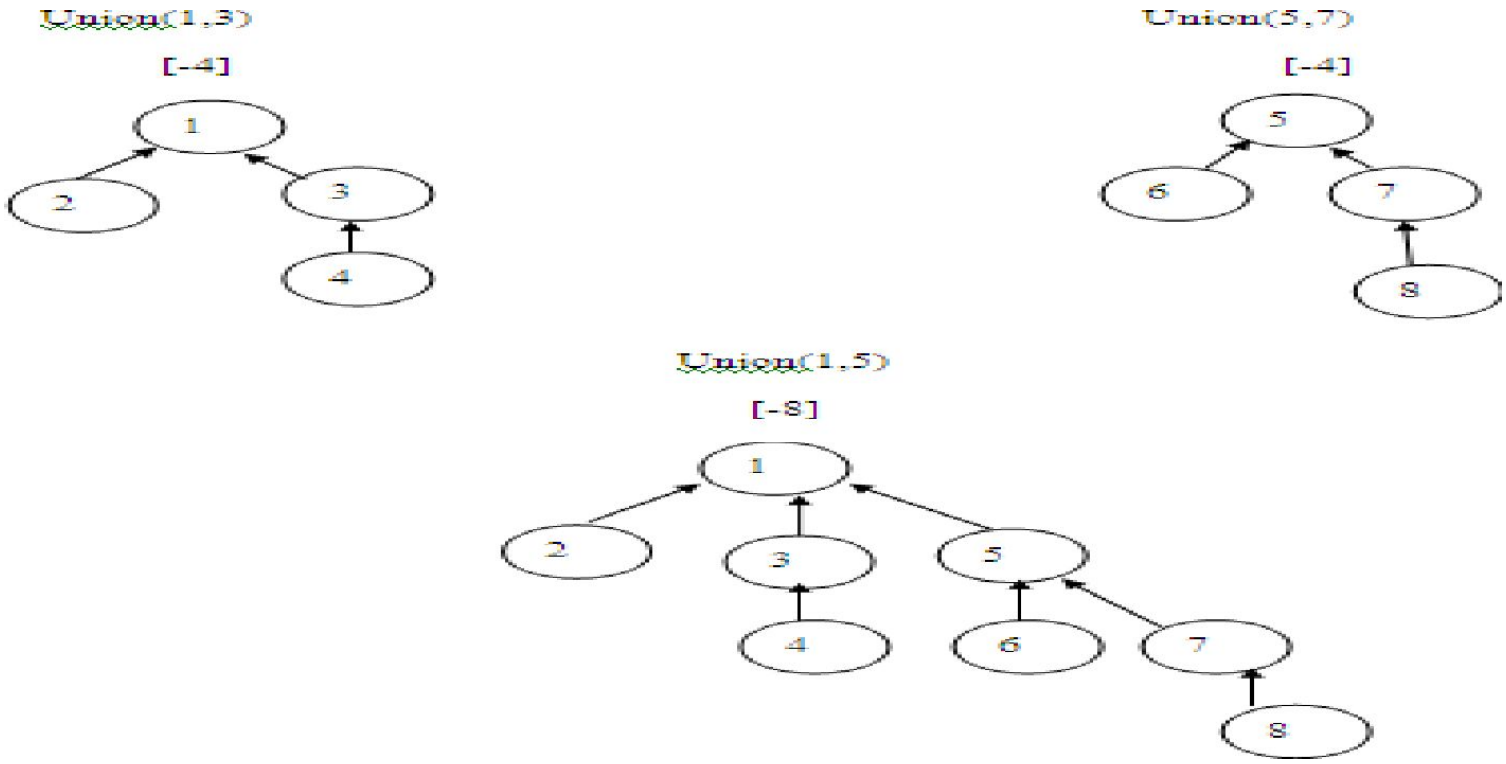
COLLAPSING FIND

If j is a node on the path from i to its root and p[i] ≠root[i], then set P[j] to root[i].

Consider the tree created by WeightedUnion() on the sequence of1≤i≤8. Union(1,2), Union(3,4), Union(5,6) and Union(7,8)



COLLAPSING FIND CONTINUES…



Now process the following eight find operations Find(8),Find(8) .....Find(8)
If SimpleFind() is used each Find(8) requires going up three parent link fields for a total of 24 moves.

COLLAPSING FIND CONTINUES…

When Collapsing find is used the first Find(8) requires going up three links and resetting three links. Each of remaining seven finds require going up only one link field. Then the total cost is now only 13 moves.(3 going up + 3 resets + 7 remaining finds).

ALGORITHM COLLAPSING FIND

```
Algorithm CoIlapsingFind(i)
// Find the root of the tree containing element i. Use the
//collapsing rule to collapse all nodes from i to the root .
{
    r := i;
    while (p[r] >0) do
        r := p[r]; // Find the root,
    while ( i< r) do // Collapse nodes from i to root r ,
    r:=p[i];
    return r;
}
```

END OF PRESENTATION

THANK YOU