Data Structures and Algorithms

Mohammad GANJTABESH

Department of Computer Science,
School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.

mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

Disjoint Set

Data Structures and Algorithms
Undergraduate course

Mohammad GANJTABESH
mgtabesh@ut.ac.ir

Dept. of Computer Science
University of Tehran

Motivations

Application :

- Find reachable points in a maze.
- Find Minimum Spanning Tree in graphs.
- Efficient way to work with sets ...

Operations

1) MakeSet (x) : creates a singleton set $\{x\}$

2) Find (x) : returns ID of the set containing x such that
   - if x and y lie in the same set then Find(x) = Find(y)
   - otherwise, Find$(x) \neq$ Find(y)

3) Union (x, y) : Merges two sets containing x and y.

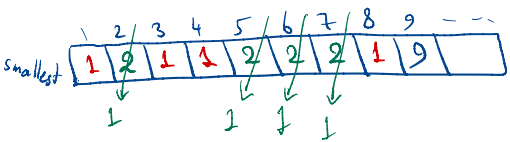
Array Implementation

- Assume that the elements are integers, say $1, 2, 3, \ldots n$
- Let the smallest element represents the set ID.

$\{5, 7, 2, 6\} \rightsquigarrow ID = 2$

$\{8, 1, 3, 4\} \rightsquigarrow ID = 1$

$\{9\} \rightsquigarrow ID = 9$




```
MakeSet (i) {                    O(1)
    smallest[i] ← i;
}
Find (i) {                       O(1)
    return (smallest[i])
}
```

```
Union (i, j) {                   O(n)
    i_id ← Find (i);
    j_id ← Find (j);
    if (i_id = j_id) return;
    m ← min (i_id, j_id);
    for (k ← 1 to n) {
        if (smallest[k] ∈ {i_id, j_id})
            smallest[k] ← m;
    }
}
```

Linked List Implementation

*idea : represent a set as a linked list and use the list tail as ID of the set.

$\{5,7,2,6\} \Longrightarrow$ 

ID = 6

$\{1,4,3\} \Longrightarrow$ 

ID = 3

MakeSet $(x) \rightsquigarrow O(1)$

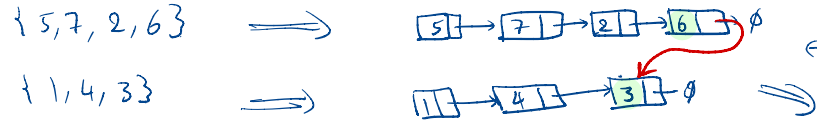
Find $(x) \rightsquigarrow O(n)$

Union $(x,y) \rightsquigarrow \begin{cases} O(1) & \text{if tail is available.} \\ O(n) & \text{otherwise.} \end{cases}$

Pros
- Union is $O(1)$
- well-defined ID (automatically updated)

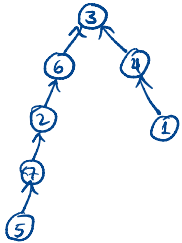
Cons
- Find is $O(n)$
- Union (x,y) require tail and head pointers

Tree Implementation

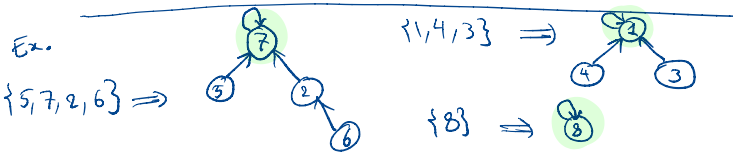


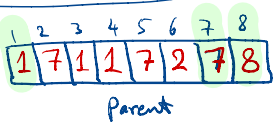
- Represent each set as a rooted tree
- ID of a set is the root of the tree
- Use array of size n to represent the whole structure.

$$\text{Parent}[i] = \begin{cases} i & \text{if } i \text{ is root,} \\ \text{parent of } i & \text{o.w.} \end{cases}$$



Ex.



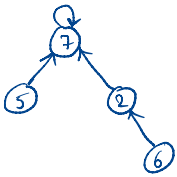


Tree Implementation

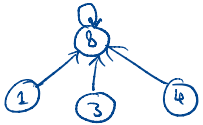
```
MakeSet (i) { O(1)
    parent[i] ← i ;
    H[i] ← 0 ;
}
```


```
Find (i) { O(height of the tree)
    while (i ≠ parent[i])
        i ← parent[i];
    return (i)
}
```

{5,7,2,6}

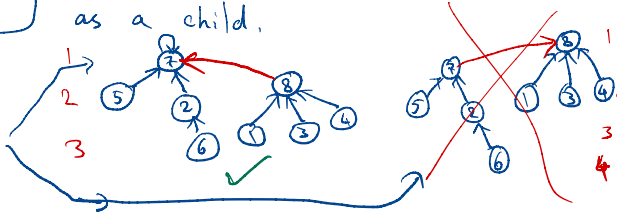


{1,3,4,8}



* To keep the height of the trees as small as possible, use the shallow tree as a child.

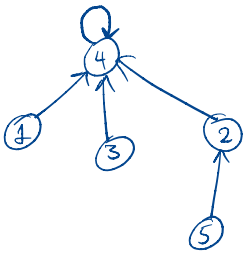
Union (6, 4)



Tree Implementation

* To quickly find the height of a tree, keep the height of each subtree in an array:

H[i] ⟵ the height of subtree rooted at i

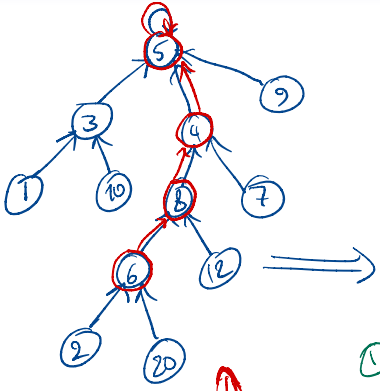


	1	2	3	4	5
H	0	1	0	2	0

```
Union(i,j){
    i_id ← Find(i);
    j_id ← Find(j);
    if(i_id = j_id) return;
    if(H[i_id] > H[j_id])
        Parent[j_id] ← i_id;
    else
        Parent[i_id] ← j_id;
    if(H[i_id] = H[j_id])
        H[j_id]++;
}
```

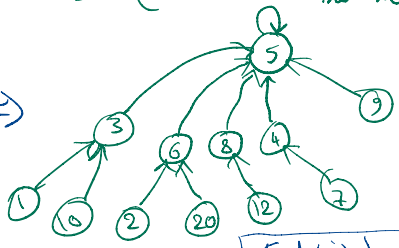
O(height of the trees)

Gain more efficiency: path compression



Find (6) : traverse the nodes 6, 8, 4, 5

Use these info. to rearrange the tree (to reduce the height).



{1, 3, 10, 5, 9, 4, 8, 7, 6, 12, 2, 20}

log* : iterative logarithm

$n = 1 \Rightarrow \log^*(n) = 0$

$n = 2 \Rightarrow \log^*(n) = 1$

$n = 3, 4 \Rightarrow \log^*(n) = 2$

$n = 5, 6, 7, 8 \cdots 16 \Rightarrow \log^*(n) = 3$

$n = 17, 18, \ldots$

```
Find (i) {
    if (i ≠ parent [i])
        Parent [i] ← Find (Parent [i]);
    return (Parent [i]);
}
```