Disjoint Sets

Dr Dayal Kumar Behera

What is Disjoint Sets

- A disjoint Set is a data structure Maintains a collection

  $S = \{S_1, S_2, \ldots., S_k\}$ of disjoint dynamic Sets

  where $S_i \cap S_j = \emptyset \quad \forall\, i \neq j$

- Here each element in the set is an object.
- Each set is identified by a representative, which can be any member of the set.

Disjoint Sets Operations

The operation it supports

- MAKE_SET(x): Creates a new set containing x as the only member Hence, x is also representative.
- UNION(x, y): Replaces the Sets S_x and S_y $with$ $S_x \cup S_y$

  One of the element of $S_x \cup S_y$ becomes the representative.

  (Since sets are disjoint, it is required that after union of S_x and S_y, S_x and S_y must be deleted from the collection)

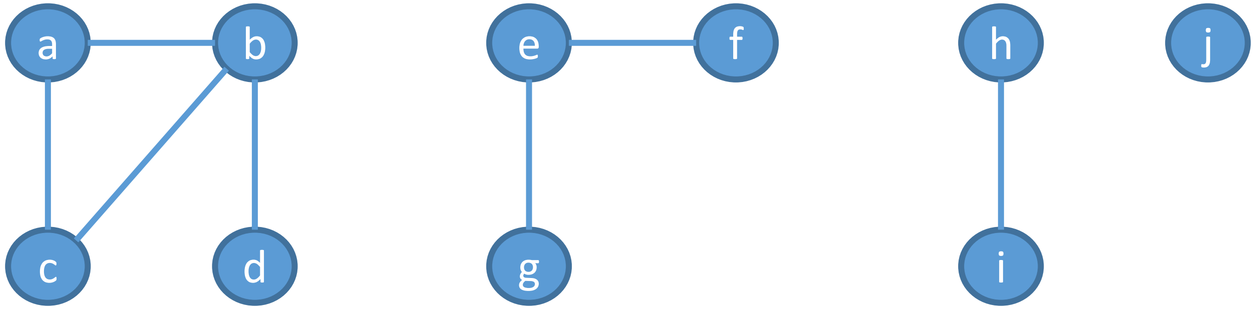
- FIND_SET(x): Returns the representative of the Set containing x

Disjoint Sets : Analysis

- We assume that MAKE_SET are the first n operations.
- Each UNION operation decreases the no of sets by one hence at max n-1 number of UNION operations are possible.

Application of Disjoint Set

- There are many applications of Disjoint Sets.
- One such application is determining the connected components of an undirected graph



Graph G is represented as

G.V as set of vertices : G.V = {a, b, c, d, e, f, g, h, i, j}

and G.E as set of Edges: G.E = {(a,b), (a,c),(b,c), (b,d), (e,f), (e,g), (h,i)}

Connected Components

G.V = {a, b, c, d, e, f, g, h, i, j}

G.E = {(a, b), (a, c),(b, c), (b, d), (e, f), (e, g), (h, i)}

Edge Processed	**Collection of Disjoint Sets**									
Initial Sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(a,b)	{a,b}		{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(a,c)	{a,b,c}			{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,c): no change	{a,b,c}			{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a,b,c,d}				{e}	{f}	{g}	{h}	{i}	{j}
(e,f)	{a,b,c,d}				{e, f}		{g}	{h}	{i}	{j}
(e,g)	{a,b,c,d}				{e, f, g}			{h}	{i}	{j}
(h,i)	{a,b,c,d}				{e, f, g}			{h, i}		{j}

A graph with four connected components

Algorithm for Connected Components

Algo CONNECTED_COMPONENT(G)

1. for each vertex v $\in G.V$
2. MAKE_SET(v)
3. For each Edge (u, v) $\in G.E$
4. if (FIND_SET(u) ≠ FIND_SET(v))
5. UNION(u, v)

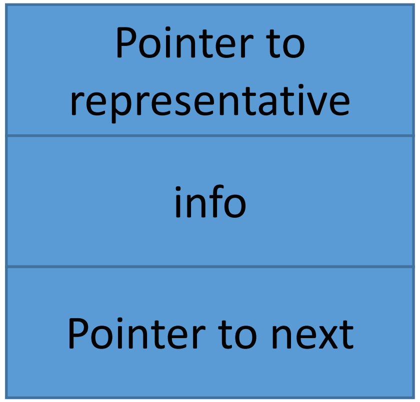
Algo SAME_COMPONENT(u, v)

1. If (FIND_SET(u) == FIND_SET(v))
2. return TRUE
3. else return FALSE

Representation of Disjoint-set

Linked-list Representation:

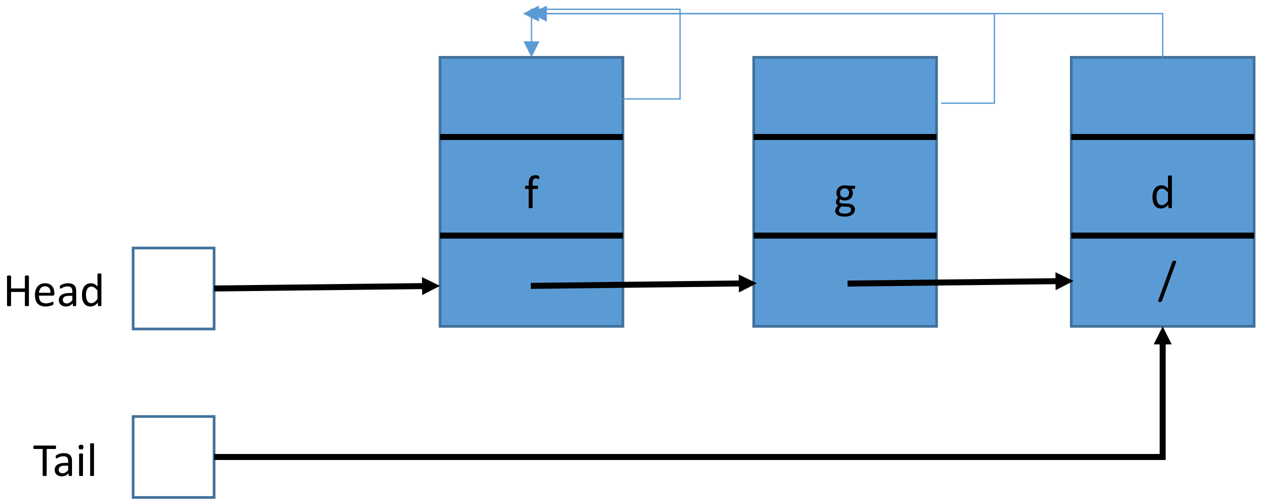
Each node can be represented as follows



Representation of Disjoint-set

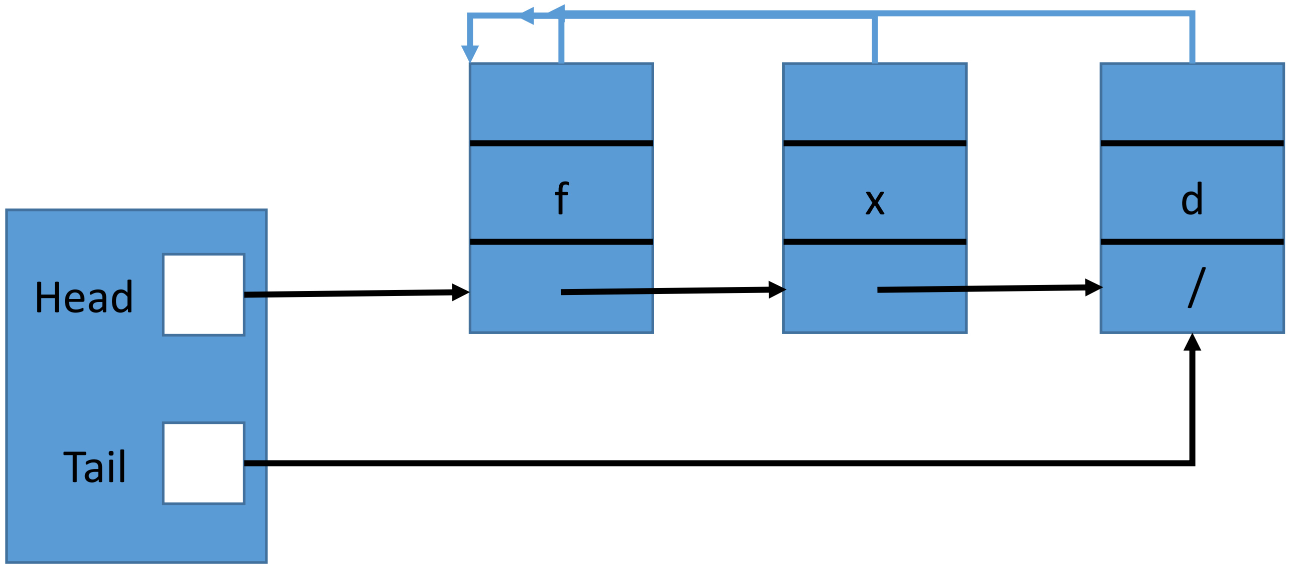
Linked-list Representation:

- Each set is represented by the same linked list
- Object of each set has attributes as head, Pointing to the First Object in the set and a tail pointing to the last object in the list.
- Each object in the list contains a set member, a pointer to the next member in the list.



Set S_1

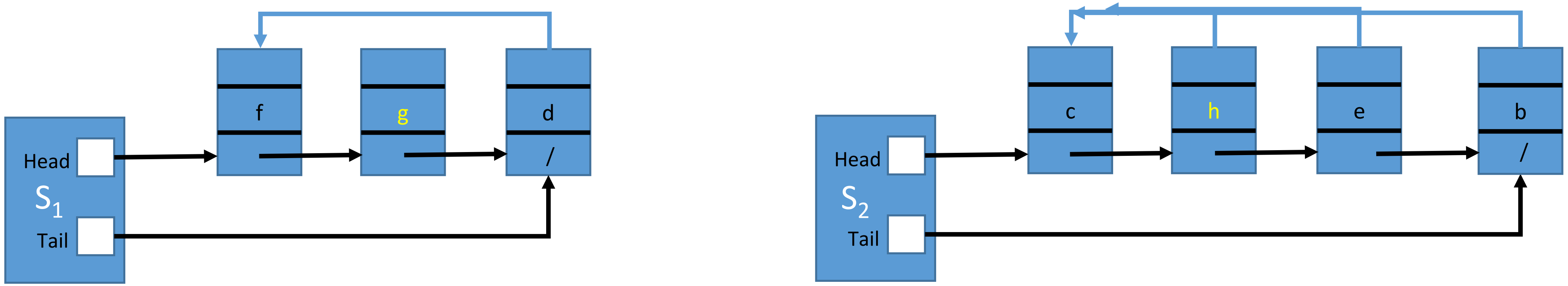
Basic Operations



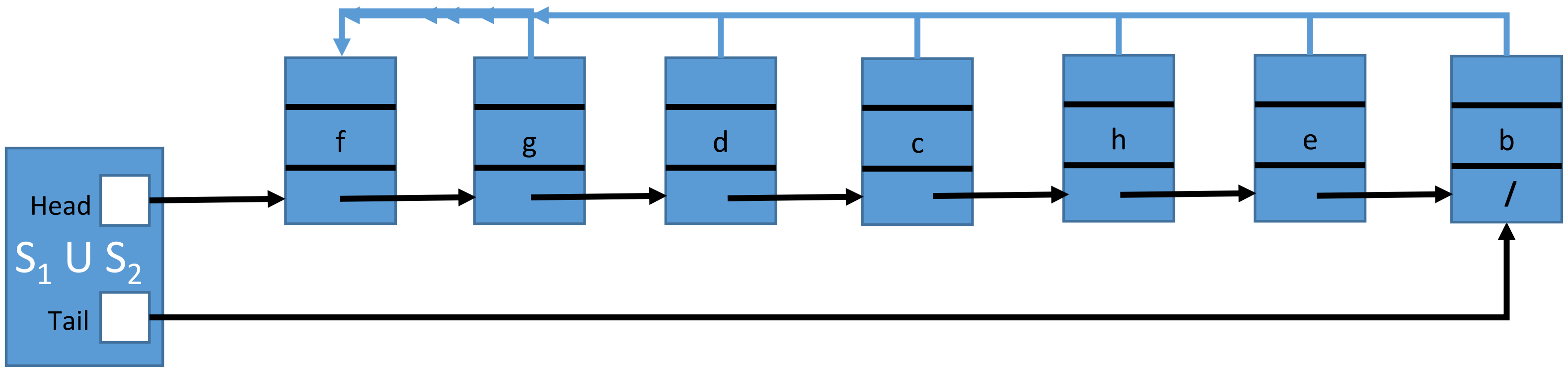
MAKE_SET(x) will work in O(1)

FIND_SET(x) should start from Head and traverse till Last Node,
If x is found it can use the address of it's object to reach the set object its belongs to and get the reference of the object that head points to.

UNION Operation



UNION(g, h)



STEPS for Union Operations

- UNION(g, h) will append h’s list (S_2) onto the end of g’s list (S_1)
- The Representative of g’s list (i.e. f) will be the representative of the union
- We will destroy the Set Object S_2

Sequence of steps

1. Identified the Last element of S_1 (using S_1.tail) and next of that object should update with S_2.head
2. S_1.tail = S_2.tail
3. Head of each object in S_2 should point to S_1 Set Object
4. Destroy S_2

Analysis of Operation

Operations	No of Object Updated
MAKE-SET(x_1)	1
⋮	⋮
MAKE-SET(x_2)	1
MAKE-SET(x_n)	1
UNION(x_2, x_1)	1
UNION(x_3, x_2)	2
UNION(x_4, x_3)	3
⋮	⋮
UNION(x_n, x_{n-1})	n-1

Suppose there are n objects
n operations for MAKE-SET

But for Union:
No of operations
= 1+2+3+……+ n-1
= n(n-1)/2
Which is $\theta(n^2)$

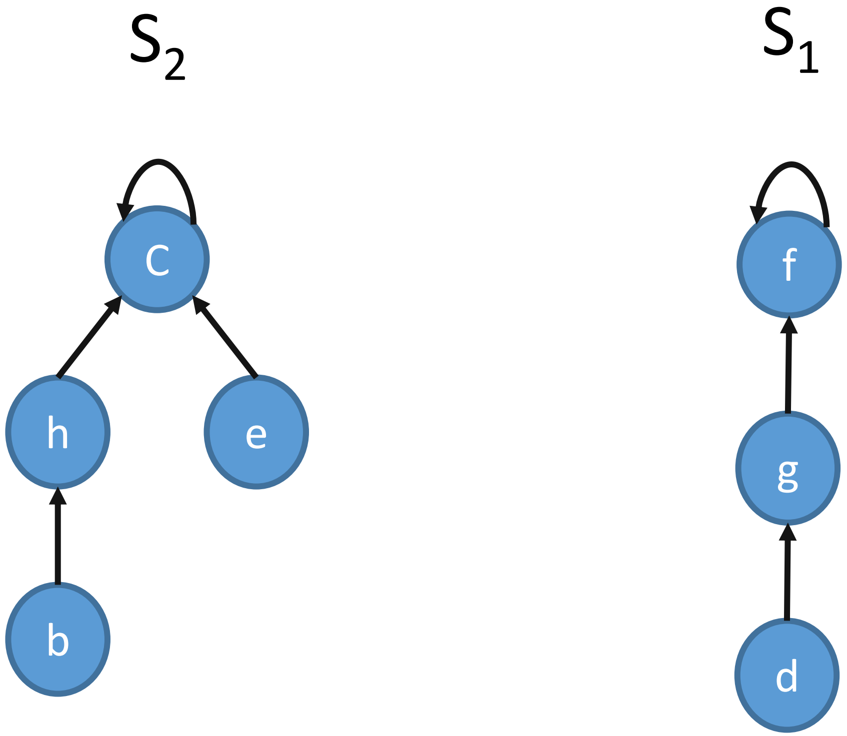
A Weighted Union Heuristic

Weighted Union: Suppose each list also includes the length of the list, and we will always append the shorter list onto a longer list

Disjoint Set Forest

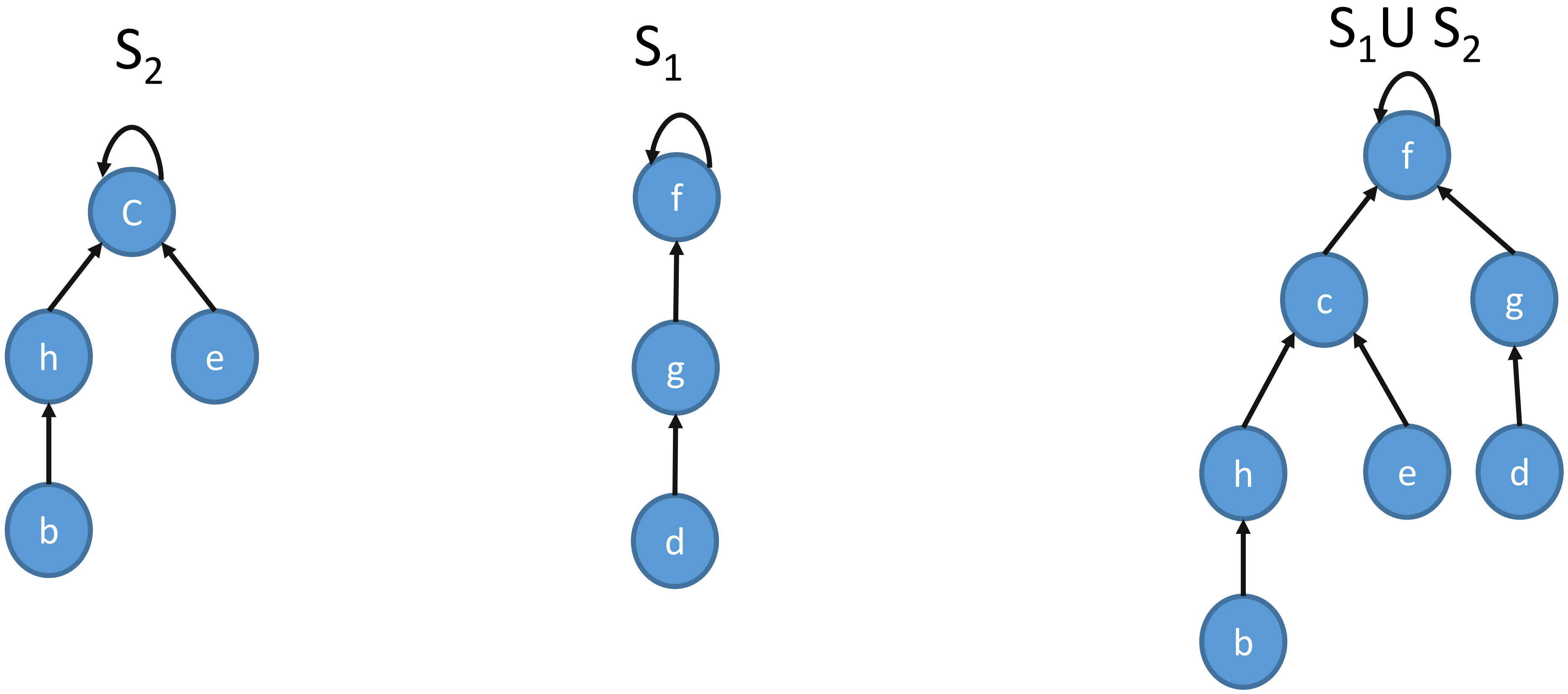
- For Faster implementation we represent sets by Rooted Trees.
- With each node containing one member and each tree represents one set
- In Disjoint Set Forest each member points only to its parent
- Root of each tree contains it representative.
- Root points to Itself, i.e. root is it’s own parent.

Tree Representation



1. MAKE_SET operation will create a Tree with One Node, i.e Root Node
2. FIND_SET operation will follow the parent pointer until it finds the root of the tree.
3. Nodes visited on this simple path during FIND_SET operation towards Route constitutes the **Find Path**

Tree Representation: Union Operation



For union operation root of one tree will point to the root of other

Analysis of all Operations

MAKE_SET is to create a set from 1 element : O(1)

FIND_SET depends of the depth of the tree

UNION is one operation of updating a root pointer

But n-1 union can simply create a tree that is a linear chain of n elements. Which will create a tree with depth n-1, and FIND_SET operation is going to suffer.

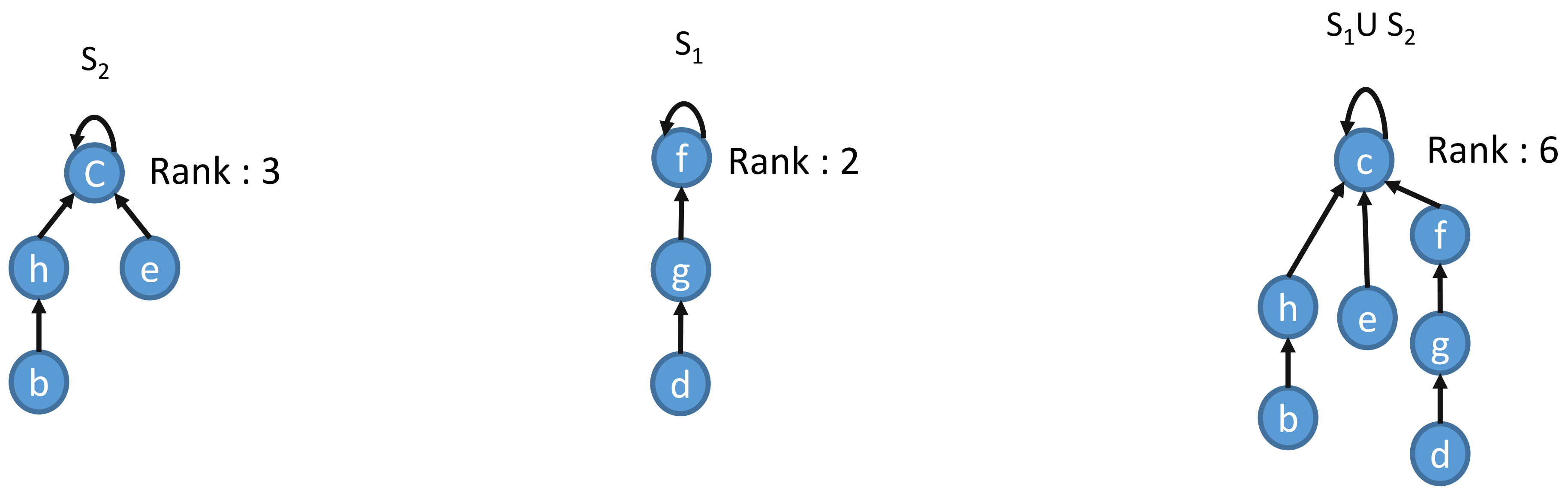
We will use to Heuristic Algorithms to improve the running time.

1st Heuristic: Union By Rank

2nd Heuristic: Path Compression

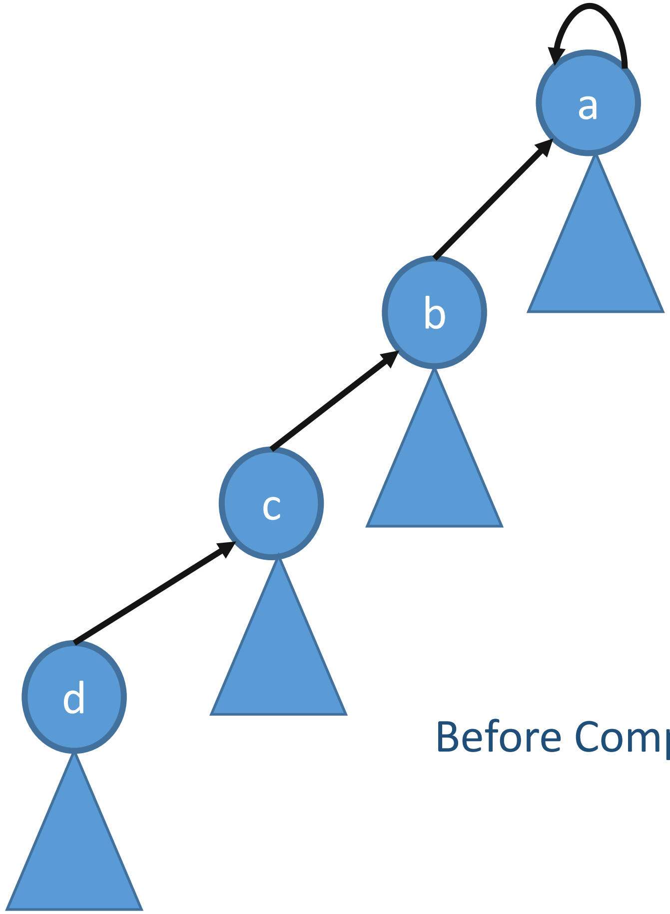
Union by Rank

- This is similar to Weighed-Union Heuristic.
- Here we keep a rank attribute of each head, which indicates how many nodes are under this root node or set.
- During Union root of the set with small rank will point to the root with larger rank.

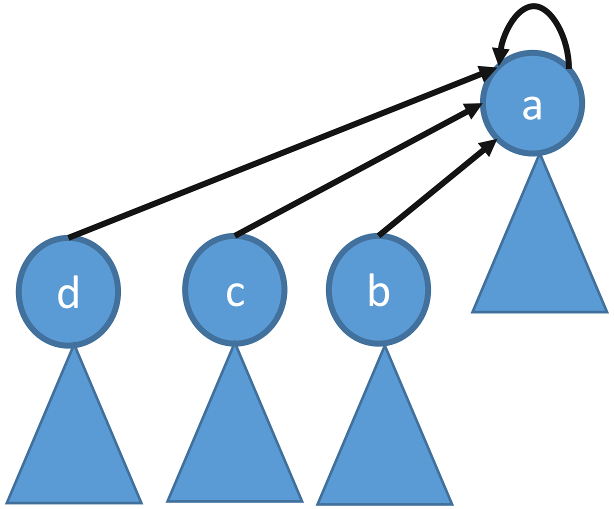


Path Compression

- During FIND_SET Operation : Each node on the **find path** is updated to directly point to the root node.



Before Compression



After Compression

This is not going to affect the Rank

Pseudo Code

Algo MAKE_SET(x)

1. x.p=x;
2. x.rank=0;

Algo UNION(x, y)

1. LINK(FIND_SET(x), FIND_SET(y))

Algo FIND_SET(x)

1. if(x != x.p)
2.   x.p = FIND_SET(x.p)
3. Return x.p

Algo LINK(x, y)

1. x.rank > y.rank
2.   y.p = x
3.   update rank of x
4. else x.p = y
5.   update rank of y

Thank You