Disjoint sets

- A Disjoint set S is a collection of sets $S_1, \ldots\ldots S_n$ where $\forall_{i \neq j} S_i \cap S_j = \phi$
- Each set has a representative which is a member of the set (Usually the minimum if the elements are comparable) .

Operations

- Make-Set(x)

  Creates a new set where x is it's only element (and therefore it is the representative of the set).

- Union(x, y)

  Unites the dynamic sets that contain x and y say S_x and S_y into a new set which is the union of two sets.

- Find(x)

  Returns the representative of the set containing x.

Example

- Maintain a set of pairwise disjoint sets.
  - {3,5,7} , {4,2,8}, {9}, {1,6}
- Each set has a unique name, one of its members
  - {3,5,7} , {4,2,8}, {9}, {1,6}
- Union(x,y) – take the union of two sets named x and y
  - {3,5,7} , {4,2,8}, {9}, {1,6}
  - Union(5,1)

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

- Find(x) – return the name of the set containing x.

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

  Find(1) = 5

  Find(4) = 8

  Find(9) = 9

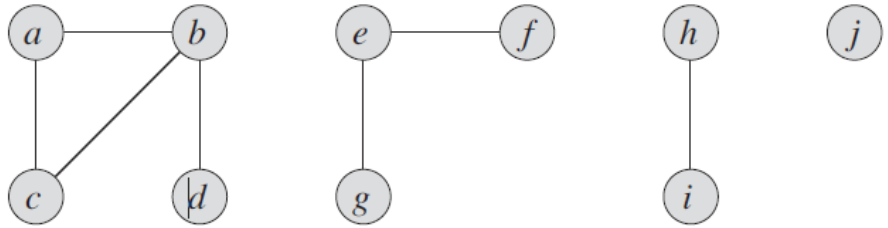
Applications of Disjoint-set

- Number of connected components in a graph.
- Minimum spanning tree

```
CONNECTED-COMPONENTS(G)
1 for each vertex  v  ϵ  G.V
2         MAKE-SET(v)
3 for each edge (u , v) ϵ  G:E
4         if FIND-SET(u) ≠ FIND-SET(v)
5                 UNION(u, v);

SAME-COMPONENT(u, v)
1 if  FIND-SET(u) == FIND-SET(v)
2         return TRUE
3 else
          return FALSE
```

Application of Disjoint-set: Connected components



Edge Processed	Collection of Disjoint sets									
Initial sets	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b , d)	{a}	{b, d}	{c}		{e}	{f}	{g}	{h}	{i}	{j}
(e , g)	{a}	{b, d}	{c}		{e, g}	{f}		{h}	{i}	{j}
(a , c)	{a, c}	{b, d}			{e, g}	{f}		{h}	{i}	{j}
(h , i)	{a, c}	{b, d}			{e, g}	{f}		{h, i}		{j}
(a , b)	{a, b, c, d}				{e, g}	{f}		{h, i}		{j}
(e , f)	{a, b, c, d}				{e, g, f}			{h, i}		{j}
(b , c)	{a, b, c, d}				{e, g, f}			{h, i}		{j}

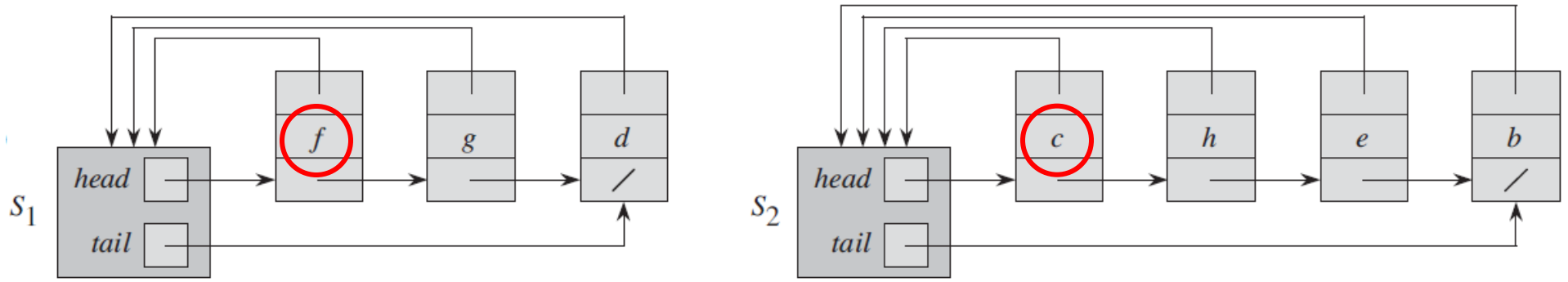
Data structure/Representation of Disjoint sets

- Linked list representation
- Rooted trees representation

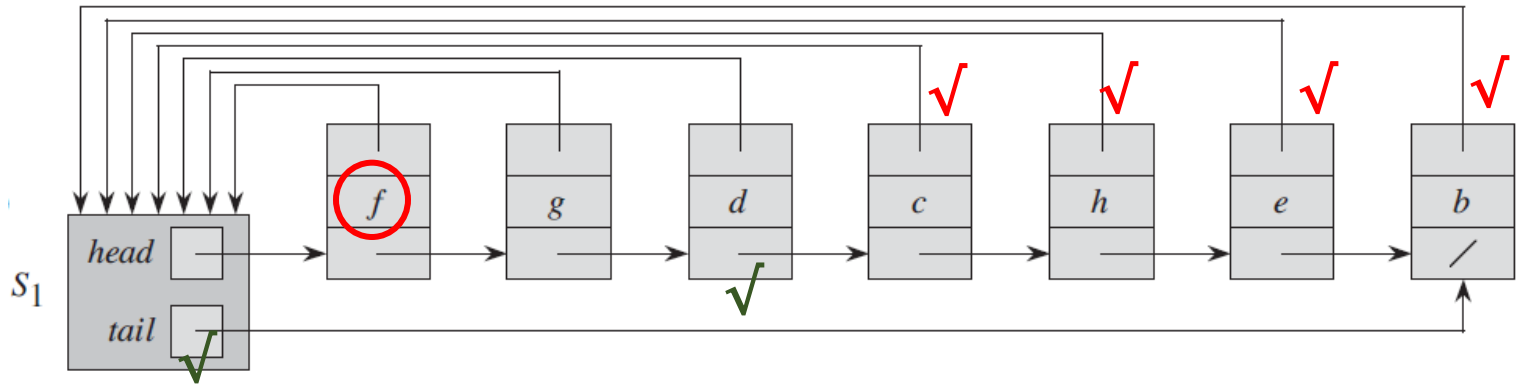
Linked list representation

The representative is the set member in the first object in the list.

Representation of set S_1 and S_2

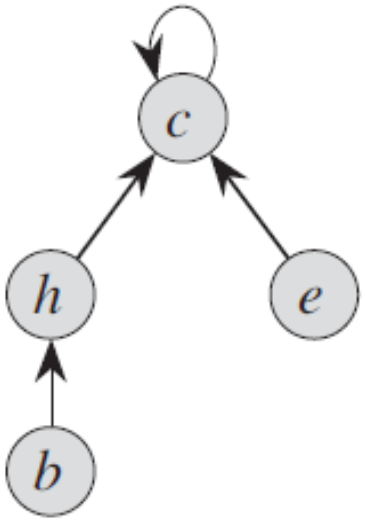


Union(g, e)

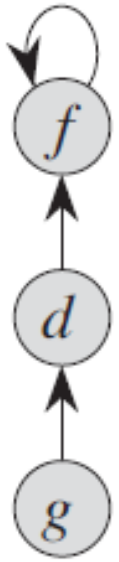


Rooted tree representation

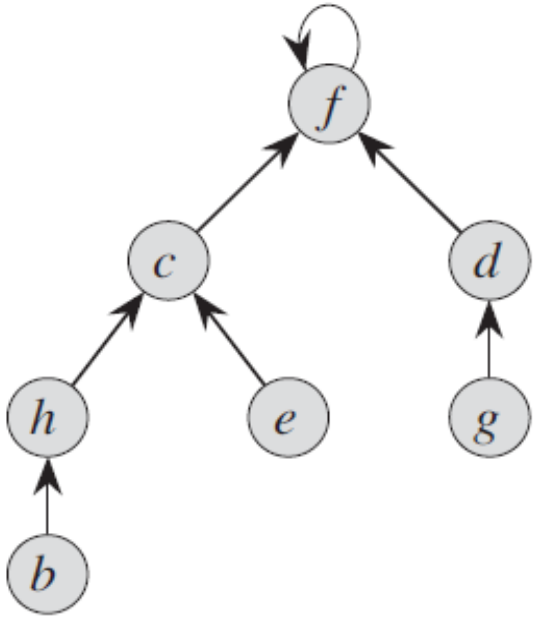
The root of each tree contains the representative and is its own parent



Representation of set S_1 with ***c*** as representative



Representation of set S_2 ***f*** as representative

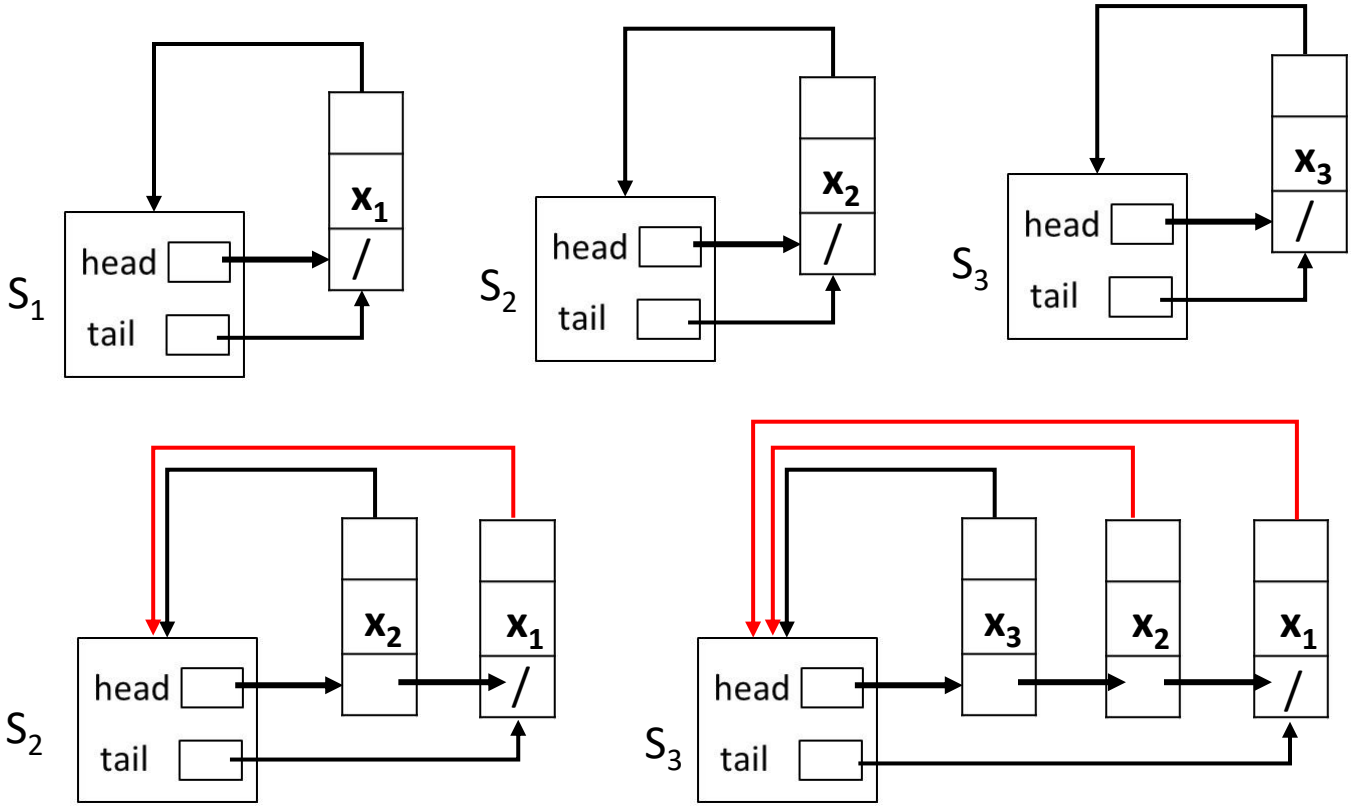


Union(g, e)

Simple Implementation of Union

- Consider Linked List representation of Disjoint sets.
- For suppose to perform a sequence of m operations on n objects as follows

Operation	Number of objects 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})	$n-1$



$$\sum_{i=1}^{n-1} i \quad = n * (n-1) \ / \ 2 \ = \Theta(n^2)$$

In the worst case, the above implementation of the UNION procedure requires an average of $\Theta(n)$ time per call because we may be appending a longer list onto a shorter list.

We can improve time using ***weighted-union heuristic***

- Maintain a number of elements in the set
- In union append the shortest list to longer list, which results in less number of objects is updated.

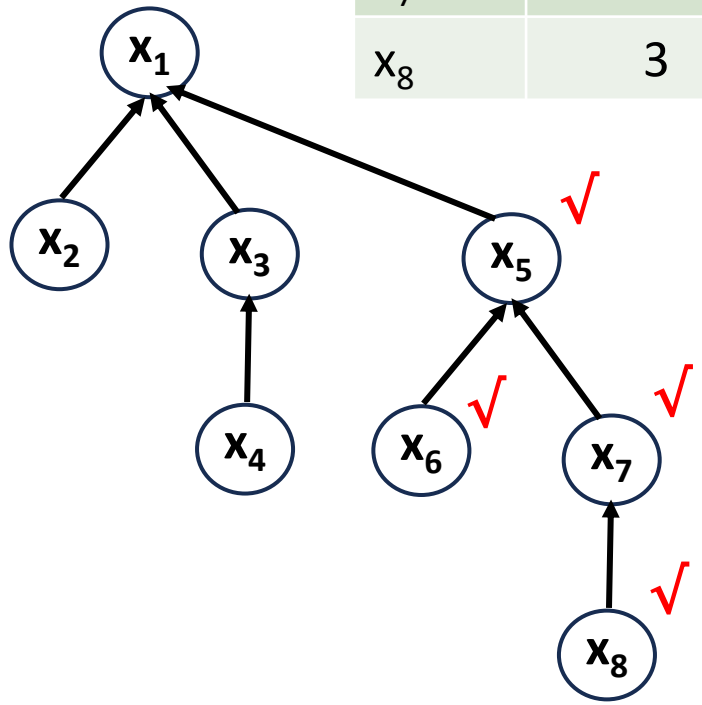
With this simple weighted-union heuristic, a single UNION operation can still $\Omega(n)$ time if both sets have $\Omega(n)$ members. A sequence of m MAKE-SET, UNION, and FIND-SET operations, n of which are MAKE-SET operations, takes $O(m + n \log n)$ time.

Why n union operations are n*(log n) instead of $O(n^2)$?

Union

Union

Object	#pointer updated
x_1	0
x_2	1
x_3	1
x_4	2
x_5	1
x_6	2
x_7	2
x_8	3



Each object pointer back is updated at most ceil(log n)
i.e., x_8 pointer is updated 3 times

Example

Suppose that CONNECTED-COMPONENTS is run on the undirected graph G= (V, E) Where V ={a, b, c, d, e, f, g, h, i, j, k} and the edges are processed in the order (d,i), (f, k), (g, i), (b, g), (a, h), (i, j) , (d, k), (b, j), (d, f), (g, j), (a, e). List the vertices in each connected component after each iteration of lines 3–5.

Heuristics to improve running time

- Union by rank
- Path compression

Union by rank

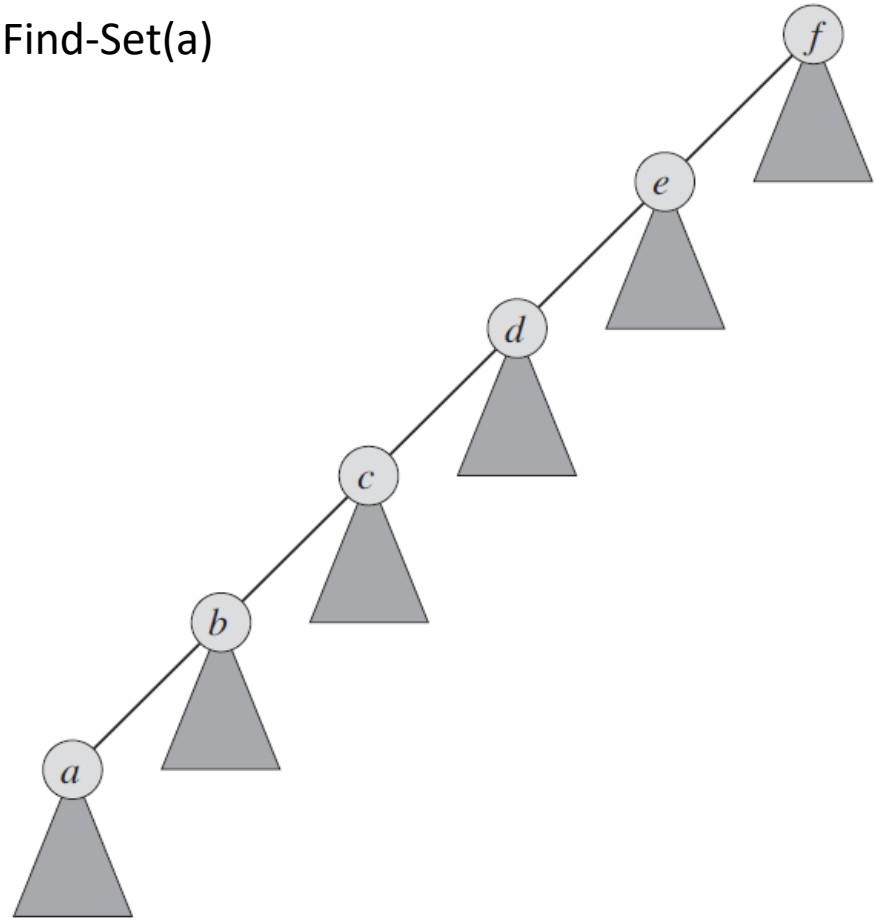
For each node, we maintain a ***rank***, which is an upper bound on the height of the node. In union by rank, the root with smaller rank point to the root with larger rank during a UNION operation

Path compression

During FIND-SET operations to make each node on the find path point directly to the root. Path compression does not change any ranks.

Path compression:

Find-Set(a)



It maintains a list of objects visited while traversing to find the root.

- list : [a, b, c, d, e]
- Update the root of [a, b, c, d, e] to f