Disjoint Sets

- Maintaining Disjoint Sets
- Complexity Analysis

Disjoint Sets - Definition

- Set representatives called **canonical elements**.
- Elements are integers beween 1 and *n*.
- Each element can be accessed in $O(1)$ time.
- We look for a data structure supporting:
  - makeset(*x*): cretion of a set with one element *x*.
  - find(*x*): returns canonical element of the set containing *x*.
  - link(*x*,*y*): forms a union of two sets with *x* and *y* as their canonical elements.

Problem

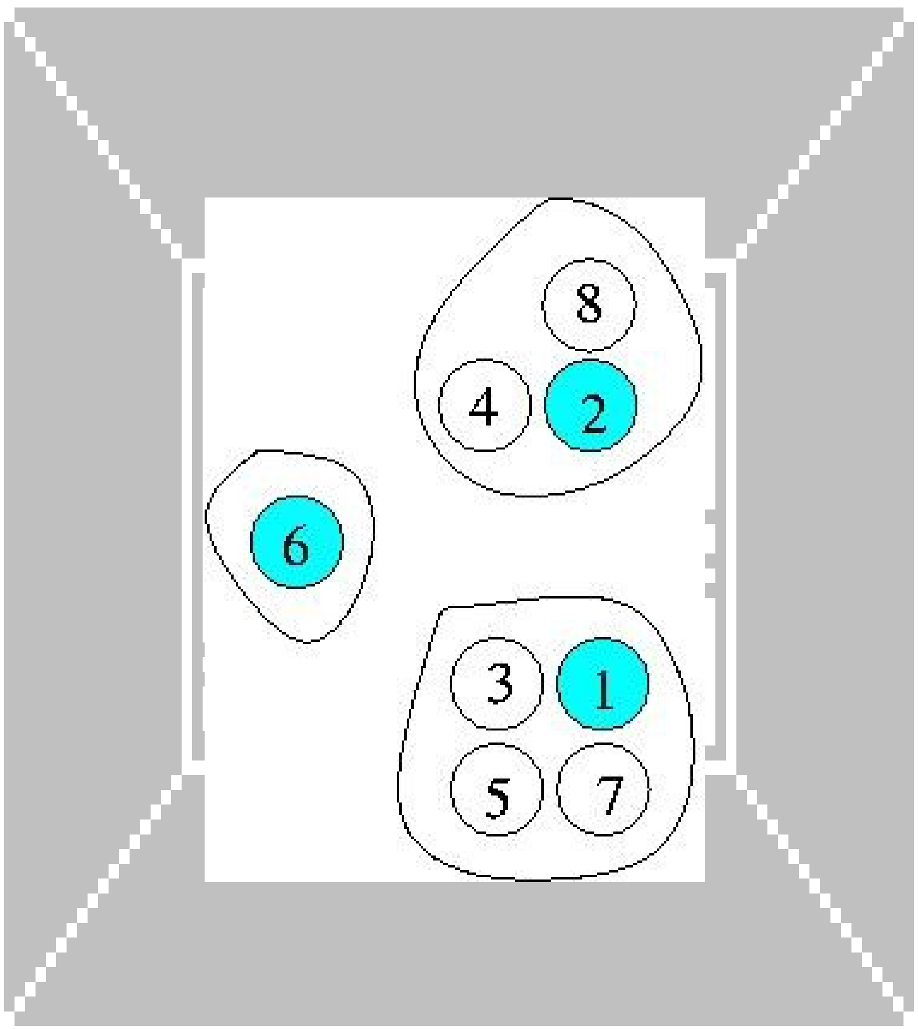
- How to represent disjoint sets in order to be able to carry out:
  - n makeset,
  - m find,
  - k link, $k \leq n-1$,

in any feasible order as quickly as possible?

Applications of Disjoint Sets

- Many algorithm (including many graph algorithms).
- Equivalence of symbolic addresses (Fortran).
- Special kind of sorting.

Vector Representation



SET	NEXT	FIRST	SIZE
1	3	1	4
2	4	2	3
1	5		
2	8		
1	7		
6	0	6	1
1	0		
2	0		

Vector Representation - makeset

- makeset(x)
  - set(x)=x;
  - first(x)=x;
  - next(x)=0; or next(x)=x;
  - size(x)=1
- One makeset takes O(1) time.
- n makesets takes O(n) time.

Vector Representation - find

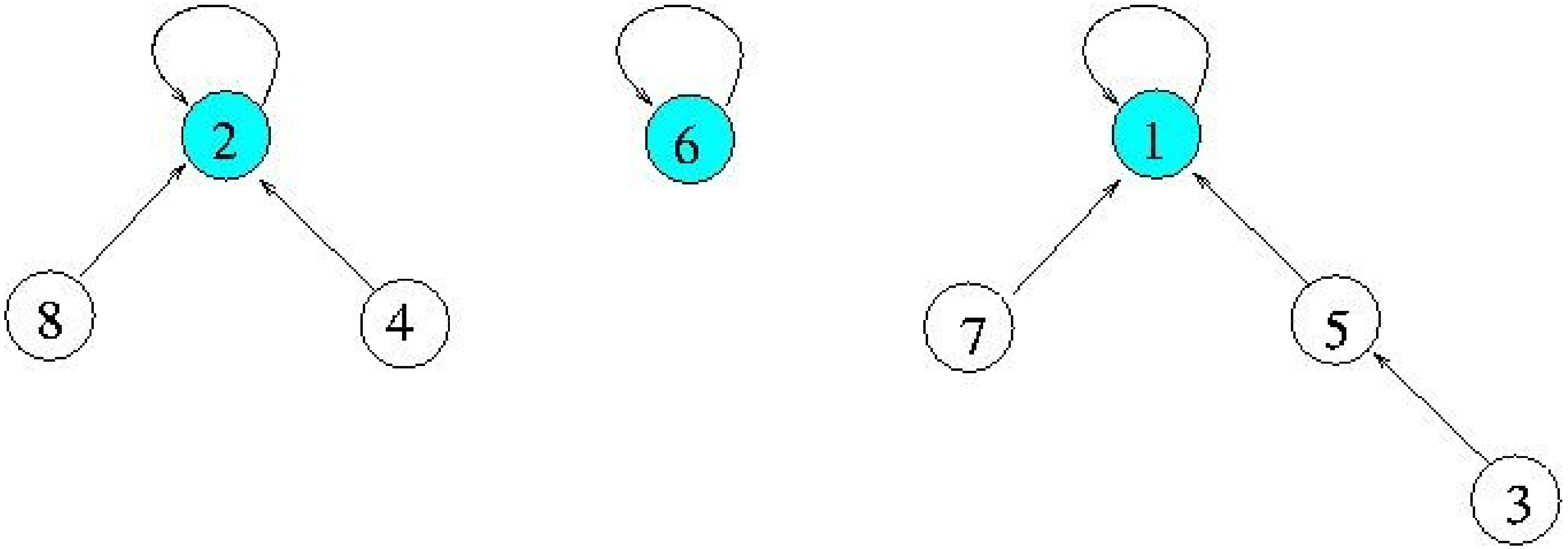
- find(x) is trivial: return set(x).
- One find takes $O(1)$ time.
- m find take $O(m)$ time.

Vector Representation - link

- link(x,y): Elements of the smaller set are added to the larger set by scanning + pointer update.
- One link takes $O(n)$ time.
- n-1 links take $O(n^2)$ time. Is this a tight bound?
- When an element is scanned, it ends up in a set that is at least twice as big.
- No element cannot be scanned more than $O(\log_2 n)$ times.
- n-1 links take $O(n\log_2 n)$ time.

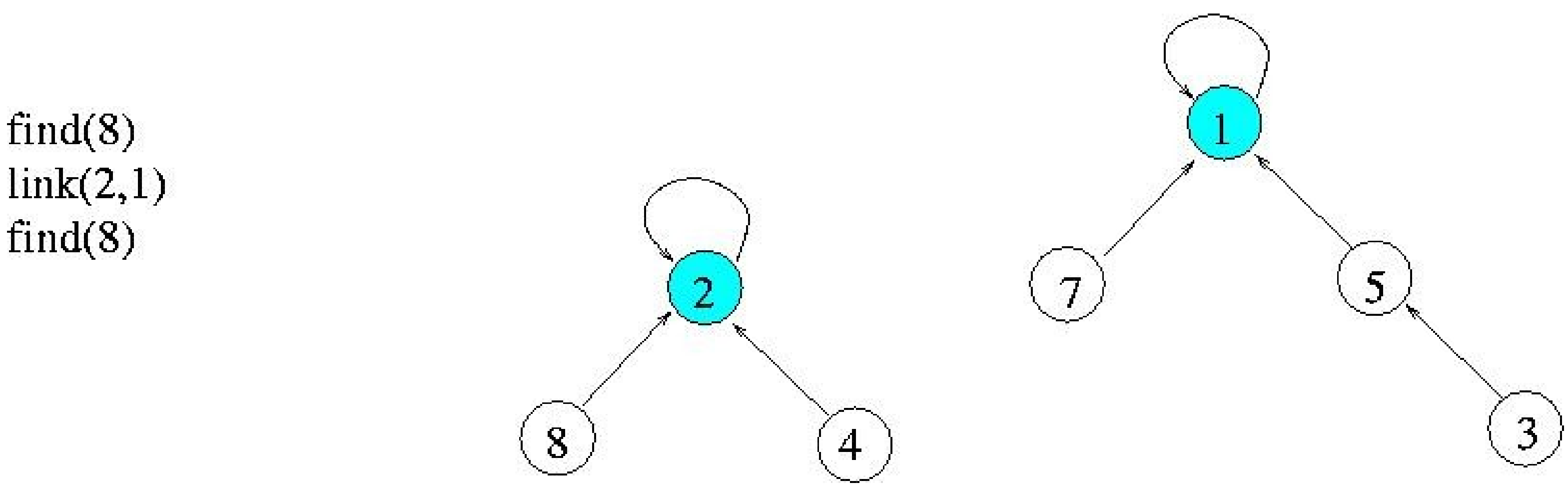
Rooted Tree Representation

- Nodes of trees contain elements, one set per tree.
- Roots contain canonical elements.
- Each node has a parent pointer. Roots point to itself.
- The same set can be represented by different trees.



Rooted Tree Representation

- makeset(*x*): create one-node tree in *O*(1) time.
- find(*x*): follow parent pointers from x to the root.
- link(*x*,*y*): let y be the parent of *x*, and let *y* be the canonical element of the union set. Requires *O*(1) time.

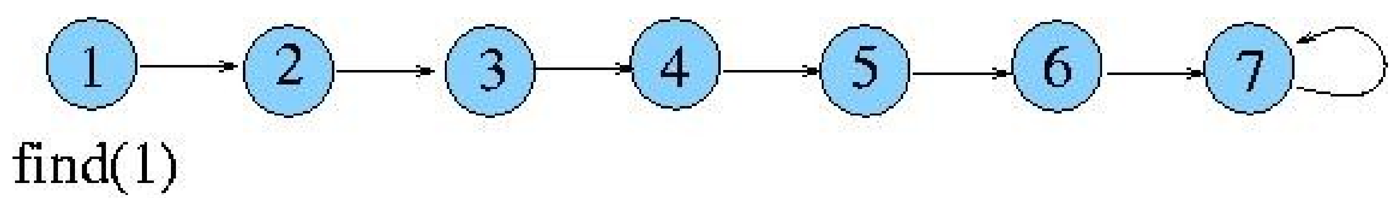


Problems with Rooted Trees

- High Trees

makeset(1), makeset(2), ..., makeset(n)

link(1,2), link(2,3), ..., link(n-1,n)

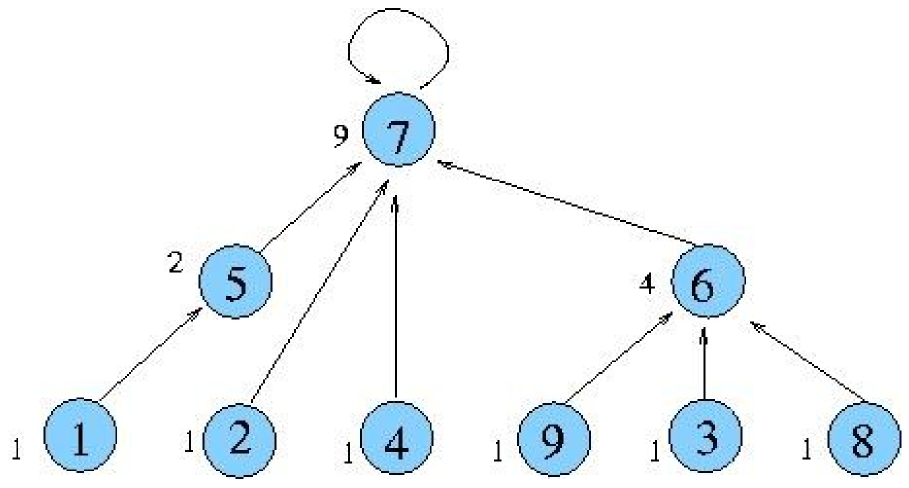


Linking by Size

- Root of the tree with more nodes is made the root of the union tree.

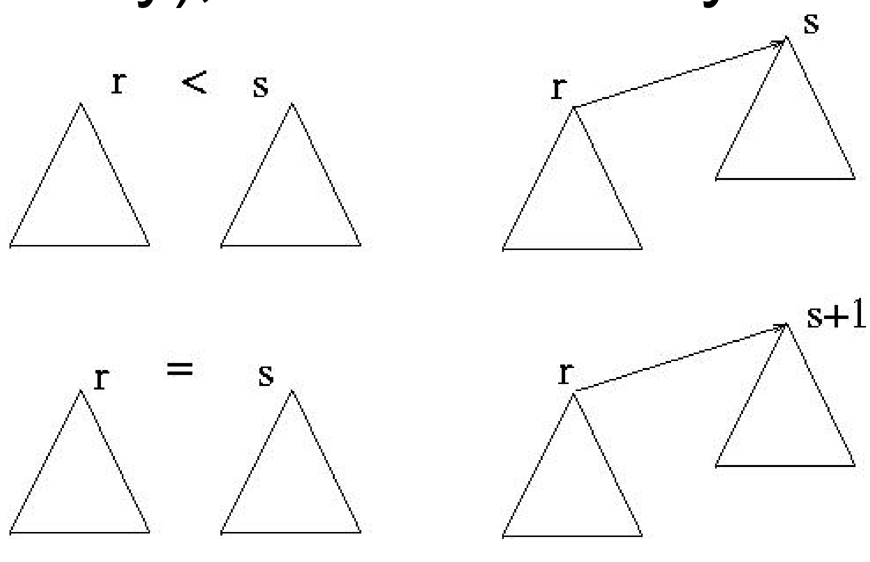
link(1,5), link(4,7), link(3,6), link(2,7)

link(6,8), link(7,5), link(9,6), link(6,7)



Linking by Rank

- Roots of one-element trees have rank 0.
- The root of the tree with higher rank is made the root of the union tree.
- If trees have the same rank, the rank of the new root (chosen arbitrarily), is increased by 1.



Linking by Rank - Examples

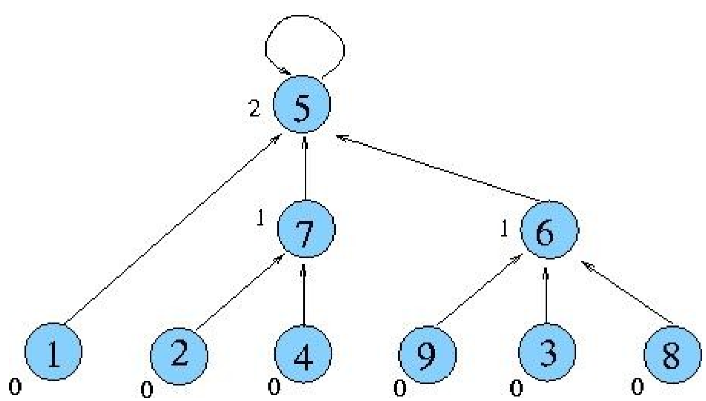
link(1,5), link(4,7)

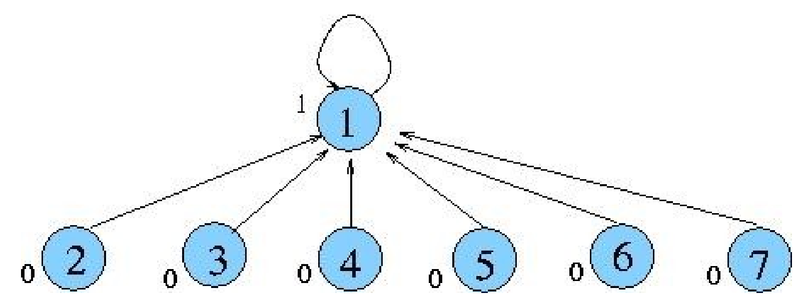
link(3,6), link(2,7)

link(8,6), link(5,7)

link(9,6), link(6,5)

link(1,2), link(2,3)
link(3,4), link(4,5)
link(5,6), link(6,7)





Linking by Rank – Basics

- Once an item seizes to be a root, it never becomes a root again. Its rank never changes.
- $r(x) \leq r(p(x))$ with strict inequality unless $p(x)=x$.
- $r(x)$ increases by at most 1 during each link.

Lower Bound on Number of Elements

- $s(x)$ = # of elements in a tree with x as root.
- Claim: $s(x) \geq 2^{r(x)}$
- Proof by induction on the number of link operations.
- True before first link: $s(x) = 1$ and $2^{r(x)} = 1$

Lower Bound Continued

- Assume that claim holds before the *i*-th link: link(x,y).
- Let $r_i(x)$ and $r_i(y)$ be rank values before the *i*-th link.
- If $r_i(x) < r_i(y)$, then y becomes root, $r_{i+1}(y)=r_i(\mathrm{y})$, and $s_{i+1}(y) > s_i(y) \geq 2^{ri(y)} = 2^{ri+1(y)}$
- If $r_i(x) > r_i(y)$, symmetric situation.
- If $r_i(x) = r_i(y)$, then y becomes the root and $s_{i+1}(y) = s_i(x) + s_i(y) \geq 2^{ri(x)} + 2^{ri(y)} = 2^{ri(\mathrm{y})+1} = 2^{ri+1(\mathrm{y})}$

Upper Bound on find

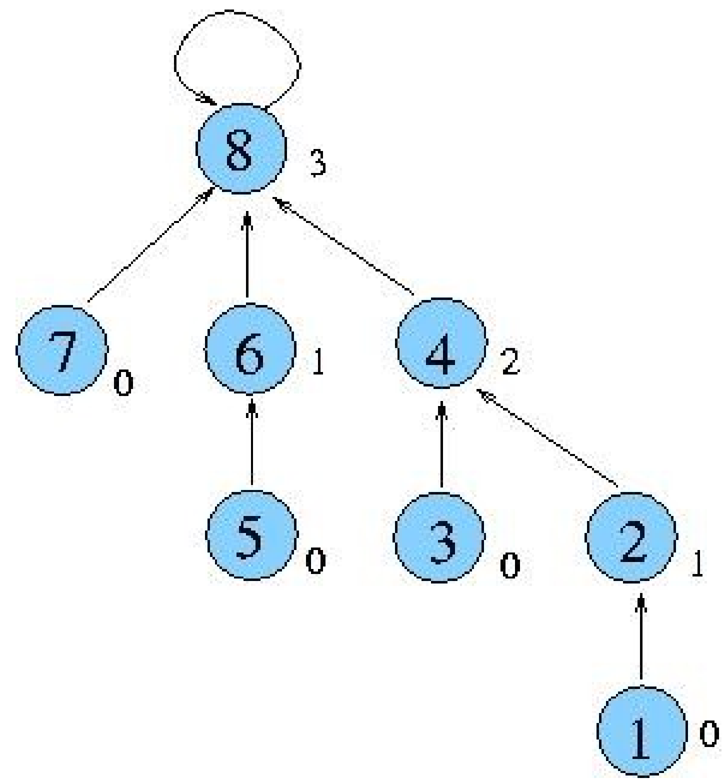
- $n \geq s_i(\mathrm{x}) \geq 2^{ri(x)}$ for all x and for all i, $0 \leq i < n$.
- $\log n \geq r_i(x)$.
- Rank is strictly increasing when going up the tree. Conclusion: find requires $O(\log n)$ time.
- Overall complexity for n makeset, n-1 link, m find is $O(n + m\log n + n-1) = O(n + m\log n)$.

Linking by Rank – Bound is Tight

- A binominal tree B_0 consists of a single node.
- A binominal tree B_i, $i > 0$, consists of two binominal trees B_{i-1} with root of one being the parent of the root of the other.
- Bi has size 2^i and height i. Proof by induction on height.

Linking by Rank – Bound is Tight

- makeset(1), makeset(2), ..., makeset(8)
- link(1,2), link(3,4), link(5,6), link(7,8)
- link(2,4), link(6,8)
- link(4,8)
- *m* times find(1)



Path Compression

- During find operation all traversed nodes are made direct sons of the root.

Disjoint Sets - Summary

	Makeset	**Find**	**Link**	**Total**
Vector	O(1)	O(1)	O(n)	O(m+nlogn)
Tree	O(1)	O(n)	O(1)	O(mn)
Link by rank	O(1)	O(logn)	O(1)	O(n+mlogn)
Compression	O(1)	O(logn)	O(1)	O(n+mlog*n)