CSE 102 3-10-20

Disjoint set ADTs

Amno-tized Analysis

chap 21 : sec 1-3 in CLRS

Disjoint set ADT :

- state : collection of disjoint sets

$\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$

Pairwise disjoint: $S_i \cap S_j = \phi$ for $i \neq j$.

each set in $\mathcal{S}$ has a distinguished element, called representative.

Ex.

$$\mathcal{S} = \{ \{a, b, c\}, \{d, e\}, \{f\}, \{g, h\} \}$$

notation: S_x is the (unique) member of $\mathcal{S}$ containing x. note x need not be representative.

above example:

$$\mathcal{S} = \{ S_a, S_d, S_f, S_g \}$$

Operations

- MakeSet(x) : creates $\{x\}$ in $\mathcal{S}$

  i.e. $\mathcal{S} = \mathcal{S} \cup \{\{x\}\}$

  Pre : $x \notin$ any other element of $\mathcal{S}$

- Union(x, y) : unites S_x & S_y

  $$\mathcal{S} = \mathcal{S} - \{S_x\} - \{S_y\} \cup \{S_x \cup S_y\}$$

  Pre : $S_x \neq S_y$

  note #sets in $\mathcal{S}$ is reduced by 1

- FindSet(x): returns (a pointer to) the representative of S_x.

Notation: will analyze runtimes using

n = # of MakeSet ops.

and

m = (#MakeSet) + (#Union) + (#FindSet)

observe that

(#union) $\leq n-1$

Two questions

(1) why?

(2) how? — Linked lists (2a)
— D.S. formats (2b)

To answer (1): let $G=(V,E)$ be an (undirected) graph.

Problem: find # of components in G, and given $x, y \in V$, determine whether x, y in same component.

Components(G)

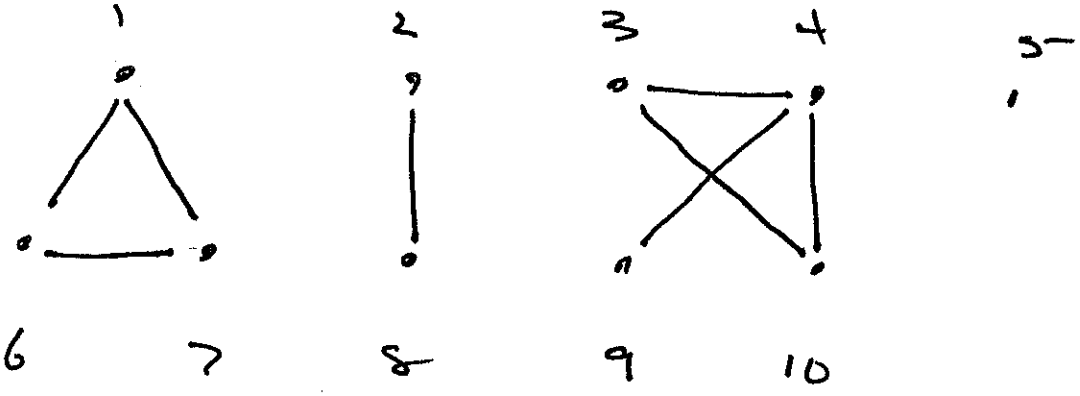
```
1. for each x ∈ V(G)
2.     MakeSet(x)
3. for each {x,y} ∈ E(G)
4.     if FindSet(x) ≠ FindSet(y)
5.         Union(x,y)
```

when done $|\mathcal{S}|$ = # connected components.

SameComponent(x,y)

```
1. if FindSet(x) == FindSet(y)
2.     return true
3. else
4.     return false
```

Ex.

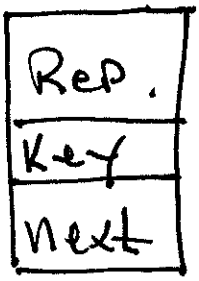


Edge	$\mathcal{A}$
	{1} {2} {3} {4} {5} {6} {7} {8} {9} {10}
{1,7}	{1,7} {2} {3} {4} {5} {6} {9} {10} {8}
{2,8}	{1,7} {2,8} {3} {4} {5} {6} {9} {10}
{3,4}	{1,7} {2,8} {3,4} {5} {6} {9} {10}
{6,7}	{1,6,7} {2,8} {3,4} {5} {9} {10}
{9,4}	{1,6,7} {2,8} {3,4,9} {5} {10}
{4,10}	{1,6,7} {2,8} {3,4,9,10} {5}
{1,6}	" " " "
{3,10}	" " " "

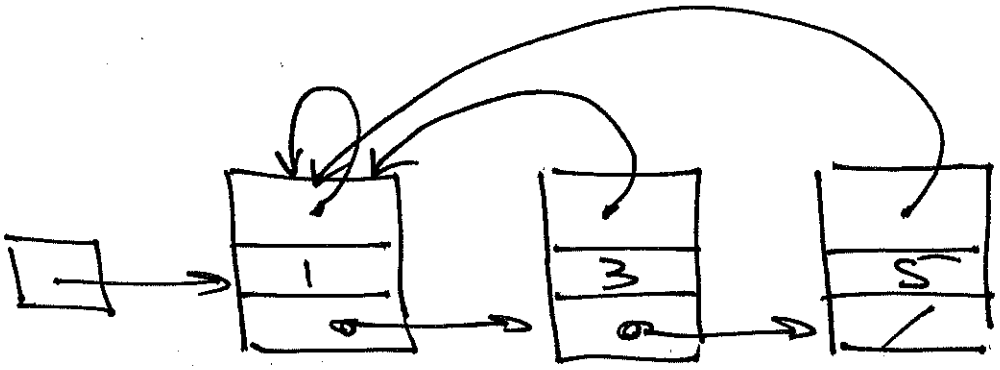
21.2 Linked list Representation

$\mathcal{S}$ is rep. by a collection of linked lists, first object in each list is representative of its set.

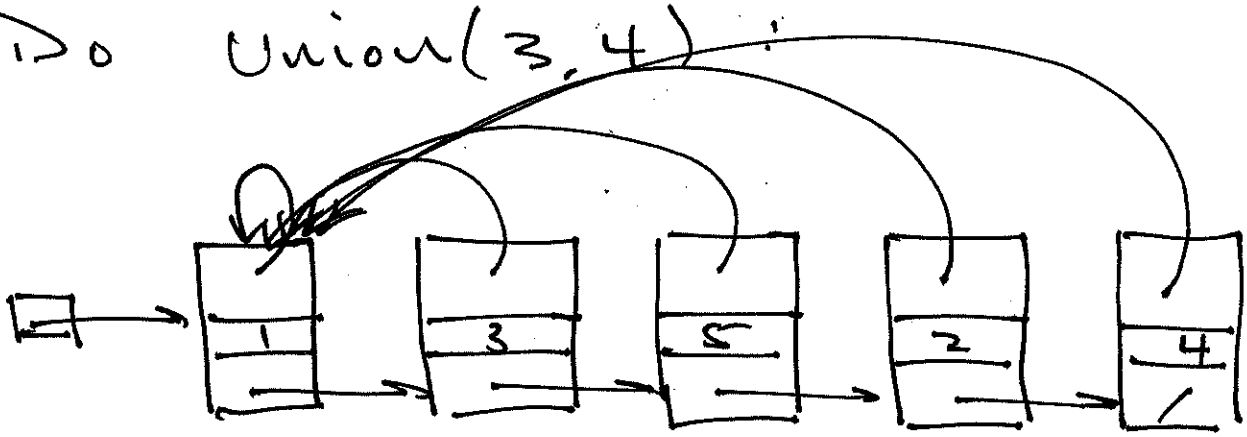
Node



Ex $\mathcal{S} = \{\{1,3,5\}, \{2,4\}\}$

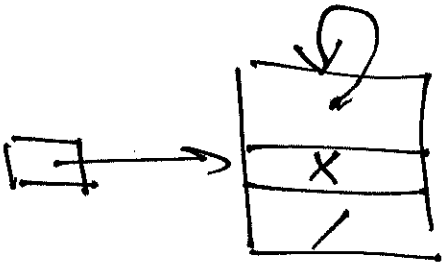


Do Union(3, 4):



MakeSet(x) $\Theta(1)$

1. Create new list with sole object x



FindSet(x) $\Theta(1)$

1. return rep[x]

Union(x, y) runtime?

1. concatenate S_x & S_y,
2. update rep. pointers of S_y to point to Rep[x].

Runtime of Union(x, y) is

$\Theta(|S_y|)$

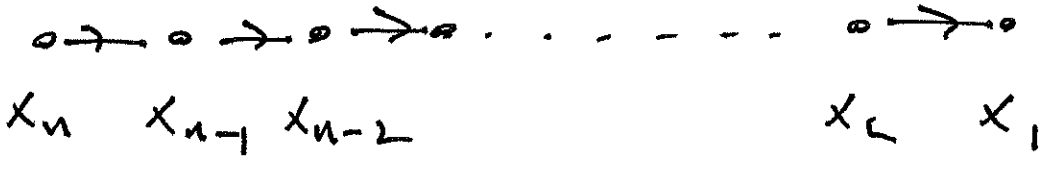
Ex

MakeSet(x_1)
⋮
MakeSet(x_n)
} n ← cost $= n \cdot \Theta(1) = \Theta(n)$

$m = n + (n-1) = 2n-1$

Union(x_2, x_1)
Union(x_3, x_2)
⋮
Union(x_n, x_{n-1})
} $n-1$

This seq. created



cost of $n-1$ Union$(\cdot,\cdot)$ is

$$1 + 2 + 3 + \cdots + (n-1) = \frac{n(n-1)}{2} = \Theta(n^2)$$

$$\text{total cost} = \Theta(n) + \Theta(n^2)$$

$$= \Theta(n^2)$$

$$= \Theta(m^2)$$

Since $m = 2n - 1$ so $n = \frac{m+1}{2}$

Ammortized Runtime

$$\frac{\Theta(m^2)}{m} = \Theta(m) \quad \text{cost Per operation.}$$

Weighted Union Heuristic

- store length of each list
- always append shorter to longer list : longer → shorter
- get fewer rep. updates.

Theorem

Using W.U.H. with linked lists, any seq. of m MakeSet, FindSet & Union ops., n of which are MakeSets

Costs : $O(m + n \lg n)$

Proof: p. 566.

Prev. example

MakeSet(x_1)
⋮
MakeSet(x_n)
} n cost $= n \cdot \Theta(1)$

Union(x_2, x_1)
⋮
Union(x_n, x_{n-1})
} $n-1$ cost $= (n-1) \cdot \Theta(1)$

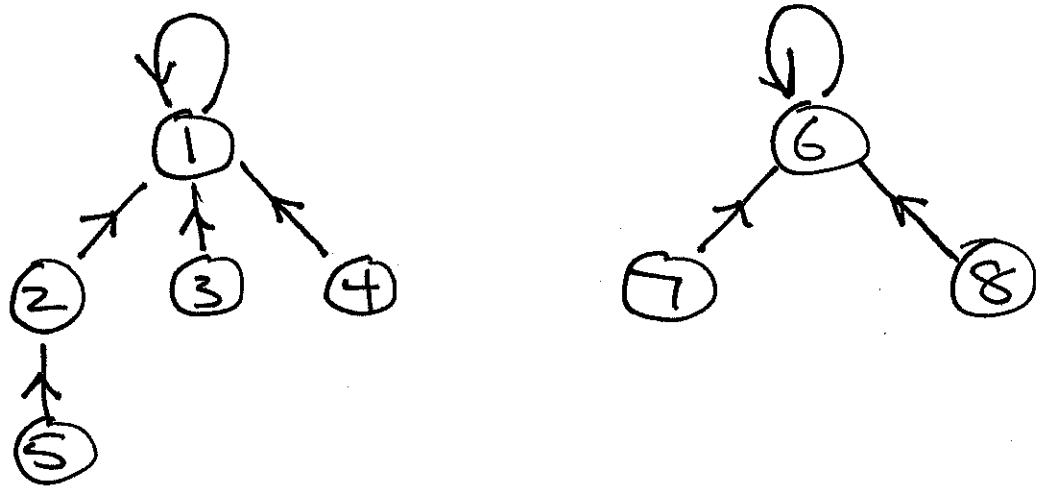
total cost $= n + n - 1 = 2n - 1 = m$

amortized cost $= \frac{\Theta(m)}{m} = \Theta(1)$.

21.3 Disjoint Set Forests

each set S is a rooted tree, each node points only to its parent. root is the representative, and is its own parent, and points to itself.

Ex. $\mathcal{S} = \{\{1,2,3,4,5\}, \{6,7,8\}\}$

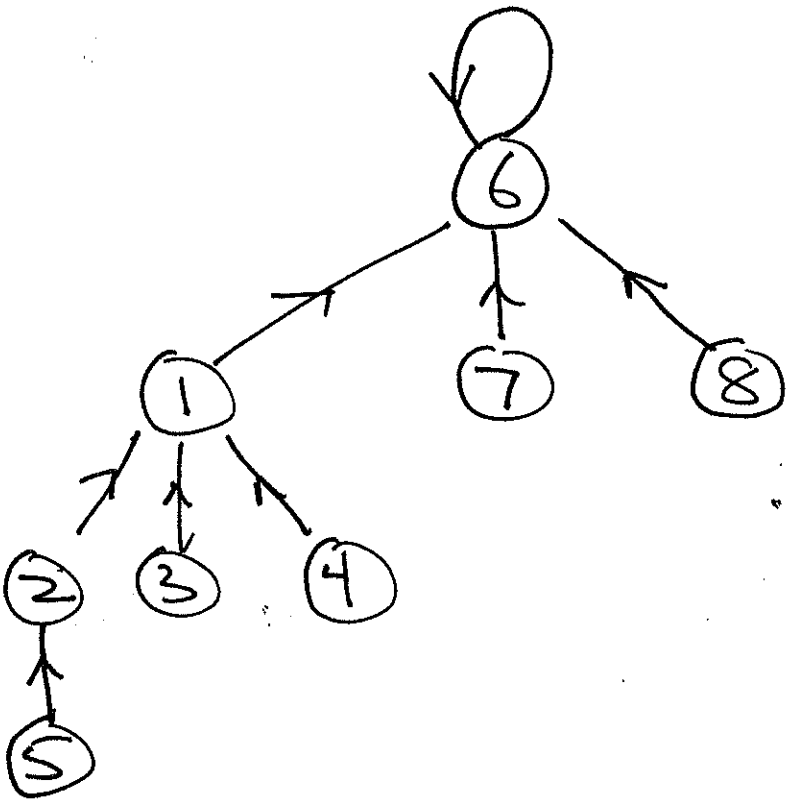


Do:

Union(3, 8)

Convention: Union(x, y) causes rep[x] to point rep[y]

result



MakeSet: creates new tree with root only

FindSet: follows parents up to root. call this a "Find-Path"

Union costs $\Theta(1)$

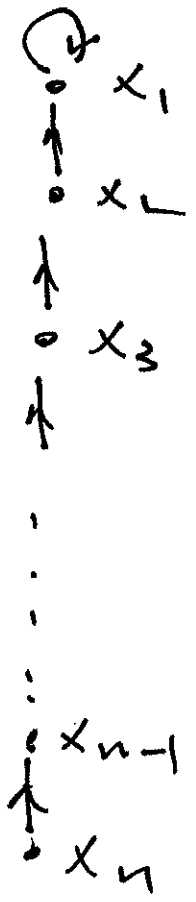
MakeSet costs $\Theta(1)$

FindSet costs Θ(length of Find-Path)

Ex

MakeSet(x_1)
⋮
MakeSet(x_n)
Union(x_n, x_{n-1})
Union(x_{n-1}, x_{n-2})
⋮
Union(x_3, x_2)
Union(x_2, x_1)

result :



total cost = $m = n + n - 1 = 2n - 1$

Two Heuristics:

Union by rank

for each node x, maintain an estimate rank[x] of height(x)

$$height(x) \leq rank[x]$$