THURSDAY, JAN 21st

(Incrementall) Dynamic Connectivity Problem (DCP)

- Start with graph with vertices only
- Edges arrive sequentially
- Keep track of connected components as new edges arrive

Generic Alg. for D.C.P.

for each edge (u,v) in a sequence of edges **do**:

  if CONNECTED(u,v) == 0 **then**

    UNION(u,v) ← more later

CONNECTED(u,v) Find(u) == Find(v)

need FIND : Find(u) returns the component label of vertex u

$n = |V|$

$m = |E|$

Alg. 1 : <u>Quick-Find</u>

- array id of length n ($|V|$)
  - if vertex i belongs to component k,
    then $id[i] = k$

$Find(i) = id[i]$ // constant time $[O(1)]$

Initially

id	0	1	2	3	4	5
	0	1	2	3	4	5

0	1	2	3	4	5
0	0	0	0	0	0

$(id[i] = i$

$i = 0, 1, \ldots, n-1$

$CONNECTED(i, j) = (id[i] == id[j])$

(Quick-Find)

UNION (i, j) : For each element (of id) equal to $id[j]$, change value to $id[i]$

cost : $\Theta(n)$

total cost ($n-1$ union operations) : $O(n(n-1))$

$= O(n^2)$!!

Alg. 2 : Quick - Union

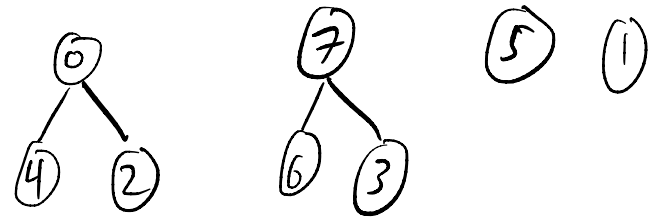
- still use array "id" → Find will be slower
  → Union will be fast : $O(1)$

- each vertex will now have parent, except for root (root is its own parent)

- if parent of vertex i is vertex j, then $id[i] = j$

Initially, $id[i] = i$ for $i = 0, 1, 2, \ldots, n-1$

(Quick-Union)



Find

Find(i):

repeatedly do $i \leftarrow id[i]$

until i stops changing

	0	1	2	3	4	5	6	7
id	0	1	0	7	0	5	7	7

- cost: length of leaf-to-root path

worst-case cost Find: $\Omega(n)$

Union(i,j)

Let a be the comp. id of root of vertex i and let b be comp. id of root of j

$a = id[id[\ldots id[i]\ldots]]$ and $b = id[id[\ldots id[j]\ldots]]$

set $id[b] = a$

Analysis of Quick-Union

Find : $\Omega(n)$ Total cost (all edges) : $\Omega(mn)$

Union : $O(1)$ Total cost : $O(n)$

$\Big] + \Omega(mn)$

Weighted Quick-Union

trick : do Quick-Union but always make tree with lower weight (lower number nodes = vertices) the child of the other

(if tie, break ties arbitrarily)

Claim:

Weighted quick-union:

Find(i) cost $O(\log_2 n)$

vertices

MONDAY, JAN 25th

Proposition: Max height among any tree in forest $\leq \underbrace{\log_2(n)}_{\lg(n)}$

Proof: Consider arbitrary node v. (we will show that $\text{depth}(v) \leq \lg(n)$)

depth k

sufficient

FACT: Observe depth of v is the same as # of times the comp. id. of v changed

- Let S_j be the # nodes in tree containing v after j^{th} label change. $(S_0 = 1)$

- weighted quick-union $\Rightarrow$ $S_j \geq 2 S_{j-1}$ $\quad S_{j+1} \geq 2 \cdot 2 S_{j-1}$

$$n \geq S_k \geq 2 S_{k-1} \geq 2^2 S_{k-2} \geq \dots \geq 2^k S_0 = 2^k$$

next step

$\log_2 n \geq k$

weighted union-find with path compression

- whenever we do Find, we make each node in the path a direct descendant of the root

Problem: worst-case time complexity of doing m union-find operations (processing m edges).

$$O(n + m\,\alpha(m,n))$$

$\alpha(m,n)$ is inverse Ackermann function

$$\alpha(m,n) = \min\left\{ i \geq 1 : A\left(i, \left\lfloor \frac{m}{n} \right\rfloor\right) \geq \log_2(n) \right\}$$

Ackermann function

$O(n + m\,\alpha(m,n))$ $\alpha(m,n)$ is inverse Ackermann function

$$\alpha(m,n) = \min\{ i \geq 1 : A(i, \lfloor \tfrac{m}{n} \rfloor) \geq \log_2(n) \}$$

Ackermann function

Examples

$A(2,1) = 5$ "Smallest i s.t. $5 \leq \log_2 n$" $\Leftrightarrow n \leq 2^5$

$\alpha(n,n) \leq 2$

$A(4,1) = 65{,}533$ "Smallest i s.t. $65{,}533 \leq \log_2 n$

$\Leftrightarrow n \leq 2^{65{,}533}$

$\alpha(m,n)$ is almost always ≤ 5

simpler bound : $O(n + m \log_2^* n)$

iterated logarithm

Definition : $\overbrace{\log_b^* n}^{f(n)} = \begin{cases} 0 & \text{if } n \le 1 \\ 1 + \log_b^*(\log_b n) \end{cases}$

all "practical" n : $\log^* n \le 5$