

Amortized Cost

- Not an algorithm design mechanism per se
- For analyzing cost of algorithms

Example: Exercise 17.3-6

Implement a queue with two stacks

Why?

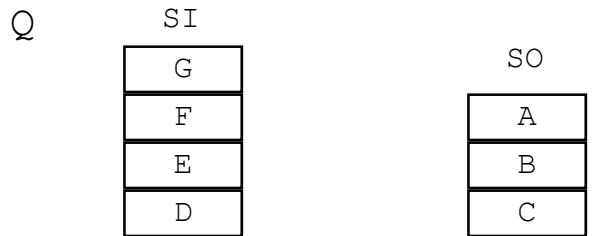
Queue Operations

`enqueue(x, Q)`

`dequeue(Q)`

`return ()`

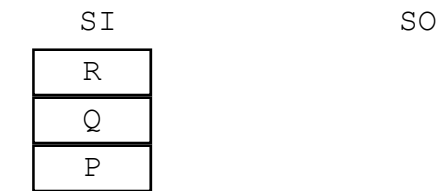
`pop(SO)`



A Complication

```
dequeue(Q)
  if empty(SO) then
    while not empty(SI) do

  return(top(SO))
pop(SO)
```



Complexity

- operation `enqueue` is always
- in a sequence of n `queue` ops, one `dequeue` could be as bad as
- but total cost of sequence isn't so bad as

Consider: each element has \$4

- pay \$1 for
- pay \$2 for
- pay \$1 for

Union-Find

Assumptions

- 1.
- 2.
- 3.
- 4.

Naming by an element is not a limitation

Usually want to know if

Critical Assumption: the number of ops is proportional to

First Hack

Array $R[1..n]$

$R[i] \equiv$ name of set

Initially, $R[i] =$

Union(i, j): Scan R . For every element k with $R[k] = i$, set

i	1	2	3	4	5
$R[i]$	1	2	3	4	5

Each union costs

Improving Efficiency

More efficiency

- find just the elements of the set
- always choose

$R[i]$ as before

If j is a "name", then $LIST[j]$ points to

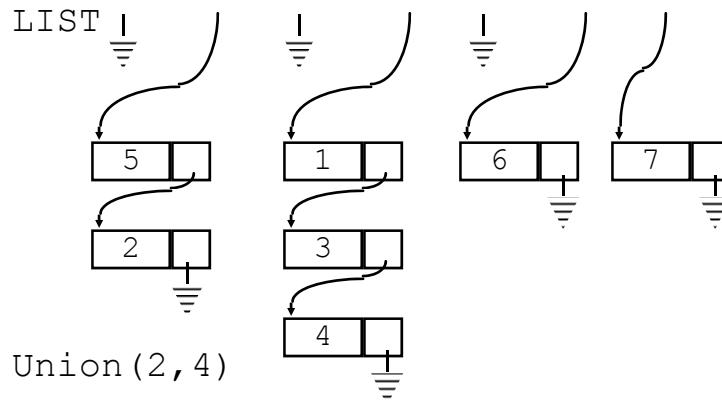
and $SIZE[j] =$

$Union(i, j)$ {assume $i =$, $j =$ }

1. Assume $SIZE[i] \leq SIZE[j]$
2. Traverse $LIST[i]$ and
3. Add $LIST[i]$ to
4. Adjust $SIZE[j]$

Example

i	1	2	3	4	5	6	7
$R[i]$	4	2	4	4	2	6	7
$SIZE$	0	2	0	3	0	1	1



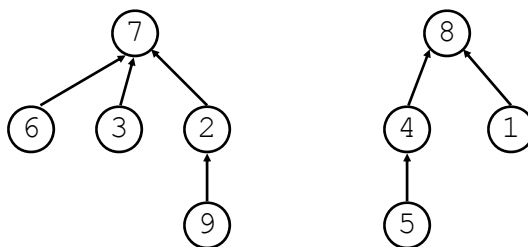
Time Complexity

Theorem: Can execute $n-1$ Union operations in $O(n \cdot \log n)$ time.

- Time for Union is proportional to
- Charge the cost to the elements moved when they go
- How many times may a single element be moved?

Tree-Structured Union-Find

Data structure: forest of trees, one per
Almost linear time for n ops



Find(i) -

Union(i, j) -

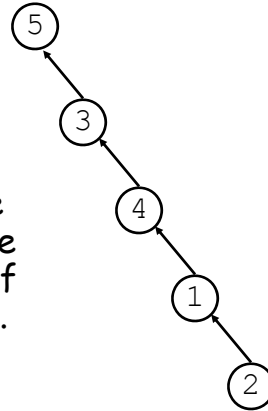
Balancing

Can get bad trees

Keep trees somewhat balanced:
always merge

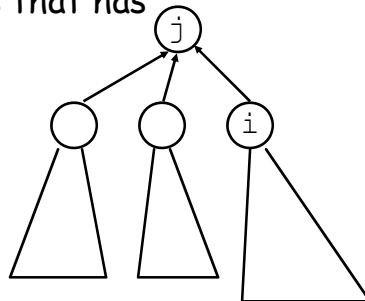
Lemma: If for every Union, the smaller tree is made a subtree of the larger, then any tree of height h has at least 2^h nodes.

Proof: If $h = 0$, then the tree must be



Proof, Continued

If $h > 0$, choose height h tree formed by Union's that has



Look at last tree added, must have height

Before the addition, T_j has height

Algorithm Design & Analysis

Path Compression

Further improvement on Find time

Lecture 3 David Maier **13**

Algorithm Design & Analysis

Complexity

Need funny functions to prove complexity

$F(0) =$
 $F(i) =$

i	0	1	2	3	4	5
-----	---	---	---	---	---	---

$F(i)$

G is kind of an inverse

$G(n) =$ the least k such that

In this universe, $G(n) \leq$ for all
 practical n

Lecture 3 David Maier **14**

Complexity 2

Show that a sequence of $c \cdot n$ Union/Find's can be done in

Assume: Union takes

Find takes "units" proportional to

Definition: The u-height (union height) of an element relative to

1. Let σ' be σ with
2. Construct a forest using
3. u-height (v) is height of v in

Complexity 3

Lemma: There are at most

Proof: Node of u-height = r has at least descendents in forest F .

If u-height (v) = u-height (w), sets of descendents are

There are at most distinct sets of 2^r elements, so

Corollary:

Complexity 4

Lemma: If during execution of σ (not σ^{-1}), w is a proper descendent of v , then

Proof: w a proper descendent of v under σ implies

Put nodes into groups by u-height

u-height = r goes in

u-height 0-1 2 3-4 5-16 17-65536 65537-2⁶⁵⁵³⁶
group

Complexity 5

How to count cost of Find's in $c \cdot n$ operations.

Split cost of Find between

1.

2. certain nodes

In the end have

sum over

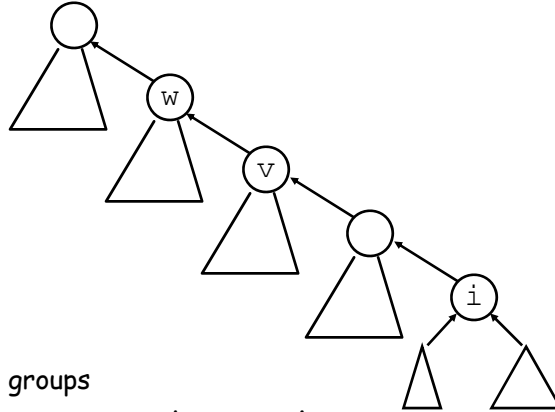
+

sum over

Suppose Find(i) traverses k nodes

- How to split up k "units"?

Complexity 6



Consider each node v on path

1. If

v is root

w is root

v, w in different groups

2. If v, w in same group and w not the root,

Complexity 7

Along any upward path, u-heights increase

At most $\frac{1}{2}n$ different groups
so case 1 applies to at most

So a Find operation gets charged at most

Complexity 8

What happens when case 2 applies?

v gets a new parent x

and $u\text{-height}(x) \leq u\text{-height}(w)$

- How many times can v get a new parent before its parent is in a higher group, and case 1 applies?

At most

Complexity 9

Charges to nodes:

$\sum_{g=0}^{F(g)} (\max \text{ nodes in } g) (\max \text{ moves for } g)$

$$N(g) \leq \sum_{r=F(g-1)+1}^{F(g)}$$

max moves for a node in a group \leq

Graphs

An undirected graph G is a pair (N, E)

N

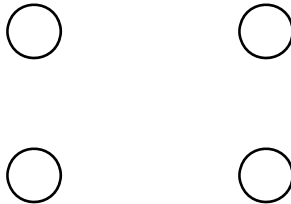
E

Example

$N = \{a, b, c, d\}$

$E =$

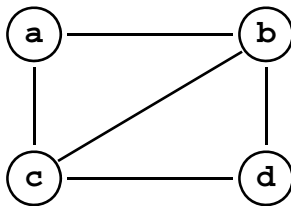
$\{\{a,b\}, \{b,c\}, \{a,c\}, \{b,d\}, \{c,d\}\}$



Paths

A path in graph $G = (N, E)$

A sequence n_1, n_2, \dots, n_k $k \geq 2$ of nodes such that



A path is simple if

A path is a cycle if

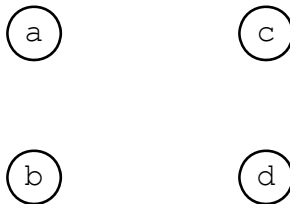
and $k \geq 3$

Directed Graphs

In a directed graph (digraph) E is

Example

$$E = \{(a,b), (b,c), (a,c), (b,d), (c,d), (d,c)\}$$



Representation

Time and space complexity depends on how we capture the graph.

Edge list

$$\{a,b\}, \{b,c\}, \{a,c\}, \{b,d\}, \{c,d\}$$

Adjacency List

a →
 b →
 c →
 d →

Representation 2

Adjacency matrix

	a	b	c	d
a	1			
b	1			
c	1			
d	0			

Depth-First Search–Undirected Graphs

A means to visit all nodes of an undirected graph

```

at node x
  pick edge {x,y}
  if y visited, pick another
  edge
  else
  if no more edges, we're done
    
```


Time Complexity

Theorem: DFS requires $O(\max(n, e))$ steps on a graph with n nodes and e edges (given as an adjacency list).

Time spent looking for unmarked nodes

DFS(v) is called once for each node.

time spent, apart from recursive calls, is

$$\sum_{v \in N} (\# \text{nodes adjacent to } v)$$

Back Edges

If $\{v, w\} \in B$, then v is an ancestor of w in the spanning forest (or vice versa).

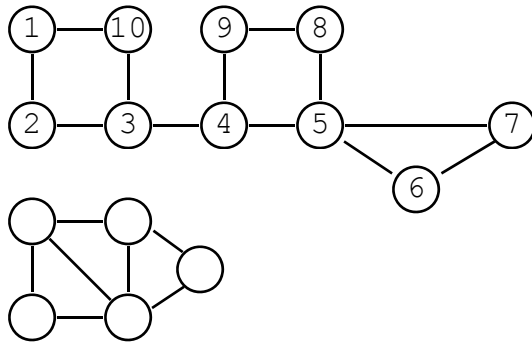
If v an ancestor of w in DFS tree, then

$$DFNum[v] < DFNum[w].$$

Biconnectivity (Problem 22-2)

Let $G = (N, E)$ be a connected

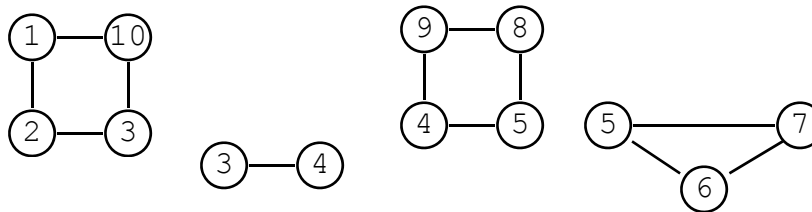
A node $a \in N$ is an articulation point if there are v and w different from a such that every path from



Biconnected Component

Maximal subgraph that is still connected after

Express in terms of edges

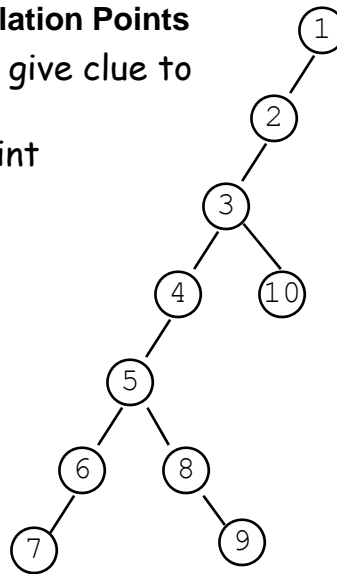


Articulation point must lie in two components

DFS and Articulation Points

Tree, back edges in DFS give clue to articulation points

If a is an articulation point that splits v and w , then



DFS and Articulation Points 2

Lemma:

$G = (N, E)$ is

$S = (N, T)$ is

Node a is an articulation point if and only if

1. a is the root of S and

or

2. a is not the root and a has some child c in S where there is no

Low Numbers

For graph $G = (N, E)$, let

$T =$

$B =$

For simplicity, assume $DFNum[v] =$

How high (lowest $DFNum$) can we hop from a node on a back edge?

$LOW[v] =$

$\min(\{v\} \cup \{w \mid \text{there is } \{x, w\} \text{ in } B \text{ where}$

Calculating Low

If $\{x, w\}$ in B and $w < v$ then

Also, v is an articulation point if for some child c of v ,

$LOW[c]$

Calculate LOW during DFS

$LOW[v]$ is minimum of

- 1.
- 2.
- 3.

Biconnected Components Algorithm

```
Count ←  
Parent[v] ←  
BI(v)  
  mark v  
  DFNum[v] ← Count, increment Count  
  Low[v] ← DFNum[v]
```

Algorithm Cont.

```
for each edge {v,w} do  
  if w not marked then  
  
    if LOW[w] ≥ DFNum[v]  
    and v not root then  
  
      LOW[v] ←  
    else if PARENT[v] ≠ w then  
      LOW[v] ←
```

Need a check at end if root is an articulation point.

