disjoint sets

CIS 315

warning: attempting to follow the logic of this analysis may cause your brain to hurt
aside: disjoint sets

**Figure 5.5** A directed-tree representation of two sets \{B, E\} and \{A, C, D, F, G, H\}.

from Dasgupta-Papadimitriou-Vazirani (also our chap 21)
union-find by rank with path compression

```plaintext
procedure makeset(x)
    \( \pi(x) = x \)
    rank(x) = 0

function find(x)
    while \( x \neq \pi(x) \):
        \( x = \pi(x) \)
    return x

procedure union(x, y)
    \( r_x = \text{find}(x) \)
    \( r_y = \text{find}(y) \)
    if \( r_x = r_y \): return
    if rank\(r_x\) > rank\(r_y\):
        \( \pi(r_y) = r_x \)
    else:
        \( \pi(r_x) = r_y \)
        if rank\(r_x\) = rank\(r_y\): \( \text{rank}(r_y) = \text{rank}(r_y) + 1 \)
```

Any sequence of \( m \) operations, \( n \) of which are makeset, takes time \( O(m \lg^* n) \)
- \( \lg^* n \) is minimum \( k \) such that \( \lg \lg \lg \ldots \lg n \leq 1 \) (\( k \) iterations)
- actually better -- \( O(m\alpha(n)) \) -- \( \alpha(n) \) is inverse Ackermann function
- both \( \lg^* n \) and \( \alpha(n) \) are very very slow growing, essentially constant
text code (p 571)

MakeSet(x)
1  x.p = x
2  x.rank = 0

Union(x,y)
1  Link(FindSet(x),FindSet(y))

Link(x,y)
1  if x.rank > y.rank
2    y.p = x
3  else x.p = y
4    if x.rank = y.rank
5      y.rank = y.rank+1

FindSet(x)
1  if x ≠ x.p
2    x.p = FindSet(x.p)
3  return x.p
A_k(j) = \begin{cases} 
  j + 1 & \text{if } k = 0 \\
  A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1 
\end{cases}

f^{(j+1)}(j) = f(f(\ldots f(j)\ldots))
is j+1-way iteration

\alpha(n) = \min\{k: A_k(1) \geq n\}

very fast growing: A_4(1) = 16^{512}, A_4(3) has 10^{19,727} digits

very hard to explain – analysis uses Ackermann’s function

originally designed to show separation between partial and primitive recursive functions

we’re not going to define potential function here, it would take all day to describe
main theorem

1) Any sequence of $m$ disjoint set operations, $n$ of which are MakeSets, uses time $O(m \cdot \alpha(n))$.

2) Furthermore, this bound is tight: for any large $m,n$, there exists a sequence of $m$ disjoint set operations (of which $n$ are MakeSets) that uses time $\Omega(m \cdot \alpha(n))$.

the optimal bound of the text is too hard – we will follow DPV and show a $O(m \cdot \lg^* n)$ upper bound
following DPV

• there are three properties of rank

• prop1: for any $x$, $\text{rank}(x) < \text{rank}(\pi(x))$

• prop2: any root node of rank $k$ has at least $2^k$ nodes in its tree

• prop3: if there are $n$ elements overall, there can be at most $n/2^k$ nodes of rank $k$

note: $\pi(x)$ is the parent of $x$ in DPV, in CLRS it’s $x.p$
intervals for ranks

- interval 0: \{1\}
- interval 1: \{2\}
- interval 2: \{3,4\}
- interval 4: \{5,6,...,16\}
- interval 16: \{17, 18,..., 2^{16}=65536\}
- interval 65536: \{65537, 65538, ..., 2^{65536}\}

interval \(k\) is of the form \(\{k+1, k+2, ..., 2^k\}\)

we look at the ranks of nodes as they pass through the intervals

at most \(\text{lg}*n\) intervals
accounting for find

• each node gets some pocket money
• total pocket money is $n \lg^* n$ dollars
• each find takes $O(\lg^* n)$ steps, plus some additional steps paid for by pocket money
• (remember: one step = one dollar)
• so overall time for $m$ finds is $O(m \lg^* n)$, plus the $n \lg^* n$ extra
observation:
by prop3, the number of nodes with rank > k is at most
\[ \frac{n}{2^{k+1}} + \frac{n}{2^{k+2}} + \ldots \leq \frac{n}{2^{k+1}} \cdot (1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \ldots) = \frac{n}{2^{k+1}} \cdot (2) = \frac{n}{2^k} \]

- nodes in interval k get \(2^k\) dollars each as pocket money
- total allocation to each interval is at most n dollars
- there are at most \(\lg n\) intervals
- total pocket money at most \(n\lg n\) dollars

- pocket money is given to a node when it stops being a root
- once it stops being a root, it will never again become a root
allocation of costs for a find

• during a find on x, look at chain of parent nodes
• if rank of $\pi(x)$ is in same interval as x, then x pays for that link from its pocket money
• if rank of $\pi(x)$ is in different interval, then cost is charged to the find operation
• at most $\lg^* n$ nodes of this latter type
• so amortized cost of find is $\lg^* n$
why this works

• each time x pays a dollar, the rank of its parent increases
• there are at most $2^k$ nodes in this interval, and x has $2^k$ dollars
• its parent will be in the next interval before x runs out of money, and then x never has to pay again
• (note: once a node is a non-root, its rank never changes)
BONUS: Fibonacci Heap

notation: in a Fibonacci heap $H$

- $m$ is the number of marked nodes
- $t$ is the number of trees in the root list
- $d$ is the maximum degree of any node in an $n$ node heap $H$ (fact: $d=\Theta(\log n)$, hard proof)
- $e$ is a constant (chosen later)

potential function: $\varphi(H) = e(2m+t)$
extractMin (consolidate)

actual cost: \( c = O(t+d) \)
potential before: \( e(2m+t) \)
new \( m \): \( m \) (no change)
new \( t \): \( d \)
new potential: \( e(2m+d) \)

amortized cost:

\[
\hat{c} = O(t+d) + e(2m+d) - e(2m+t)
\]

\[
= O(t) + O(d) + e \cdot d - e \cdot t = O(d)
\]

pick \( e \) carefully to cancel the \( O(t) \)
decreaseKey

causes k cuts (cascading cuts)
actual cost: O(k)
new m: m-k
new t: t+k
new potential: e(2(m-k)+(t+k))

amortized cost:
\[ \hat{c} = O(k) + e(2(m-k)+(t+k))-e(2m+t) \]
\[ = O(k) - e \cdot k = O(1) \]

again, pick e carefully