disjoint sets

CIS 413/513
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
union-find by rank with path compression (from DPV)

```
procedure makeset(x)
    π(x) = x
    rank(x) = 0

function find(x)
    while x ≠ π(x)
        x = π(x)
    return x

procedure union(x, y)
    r_x = find(x)
    r_y = find(y)
    if r_x = r_y: return
    if rank(r_x) > rank(r_y):
        π(r_y) = r_x
    else:
        π(r_x) = r_y
        if rank(r_x) = rank(r_y): rank(r_y) = 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
code from CLRS text

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
CLRS uses potential method

very hard to explain — analysis uses Ackermann’s function

\[ A_k(j) = \begin{cases} 
  j + 1 & \text{if } k = 0 \\
  A_{k-1}^{(j+1)}(j) & \text{if } k \geq 1
\end{cases} \]

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

originally designed to show separation between partial and primitive recursive functions

inverse Ackermann:
\[ \alpha(n) = \min \{ k: A_k(1) \geq n \} \]

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}\}

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

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

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

- each node gets some pocket money
- total pocket money is $n/lgn \times n$ dollars
- each find takes $O(lgn)$ steps, plus some additional steps paid for by pocket money

(remember: one step = one dollar)

- so overall time for $m$ finds is $O(mlgn)$, plus the $n/lgn \times n$ extra
observation:
by prop3, the number of nodes with rank > k is at most
$n/2^{k+1} + n/2^{k+2} + ... \leq n/2^{k+1} \cdot (1 + 1/2 + 1/2^2 + 1/2^3 + ...) = n/2^{k+1} \cdot (2) = n/2^k$

- 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)