AMORTIZED ANALYSIS

INSTRUCTOR: VALERIE KING
SCRIBE: VALERIE KING

```
Data structures for disjoint sets
chapter 22

X={x1, x2,x3,....xn} set of n elements
S= {S1, S_2,..S_k} set of subsets of X
Each set is identified by a representative member of the set

makeset(x)
union(x,y) let Sx be set containing x, Sy set containing y,
          destroy Sx and Sy and replace with their union
findset(x) returns pointer to representative of set containing x.


Note: no more than n-1 unions, n makesets, any number of findsets
How to implement this:

Attempt 1:

represent each set by linked list, pointer from each element to
to first element=representative.

to union, append Sx to Sy:

Cost of makeset O(1)
cost of findset O(1)
cost of union |Sx|

could be high
for example:  theta(n^2) for unions



Attempt #2:
A weighted union:
```

```
Always add smaller list to back of  longer list:
Total cost of unions then is O(nlog n)
O(m) for m other operations.

Weighted union takes O(nlogn) over course of algorithm:
proof:
let x be any element. How many times can x's pointer be updated?
Each time x is updated, size of list containing x increases by at least 2.
So can only update x log n times.

Attempt #3
Disjoint forest

represent each set by a rooted tree, parent pointers
node contains one element
root is representative of set and points to itself

findset: need to trace up parent pointers to root
union: make one tree child of root of other tree
makeset
Could get very long trees

2 ideas:
union by rank
path compression

union by rank
 Idea: make root of tree with fewer nodes point to tree with more nodes
 use rank of tree as upper bound on its height.

path compression
 when you do findset, set each node you pass to point to root

pseudocode:

makeset(x) :
p(x) <-x
rank(x)<- 0

----

union(x,y)
link(findset(x), findset(y))
```

```
link(x,y)
 if rank[x]> rank[y]
  then p[y] <- x
  else p[x] <- y
  if rank[x]=rank[y] then rank[y] <-rank[y]+1


findset(x)
if x ~=p(x)
 then p[x]<-findset(p[x])
return p[x]

(sets all parent pointers on path to root, to the root)

Let m be the number of operations and n the number of elements
total cost is O(m alpha(m,n)) where alpha(m,n) is the inverse
Ackermann's function

Ackermann's function
A(1,j) = 2^j for all j
A(i,1)=A(i-1,2) i>1
A(i,j)=A(i-1,A(i,j-1)) i,j >1


i\j =     1            2            3                 4

 1        2           2^2           2^3                2^4

 2        2^2       2^2^2         2^(2^(2^2))    2^(2^2^2^2)

 3        2^2^2    2^{2^....^2}    2^{2^...^2}    2^{2^...^2}
                   16 of 'em          2^{2^...^2}    2^{2^...^2}
                                            16           2^{2^...^2}
                                                              16

e.g.
2^{2^2^2^2} = 2^(2^(2^(2^2)= 2^65536
    4

Let alpha = inverse ackermann's(m,n) = min{i >=1| A(i, floor(m/n)) > lg n)
A(4, m/n) > A(4,1) ~ 10^80 so never more than 4, in reality.
```

Theorem: If m >= n disjoint sets operations are performed on n elements then the amortiz

Lemma 1: for any node x, rank(x) <= rank(p(x))
equal only if p(x)=x.
rank increases until x is no longer a root. Then it
is unchanged.

Lemma 2: The number of nodes in a tree with root x
is at least 2^rank(x)
proof by induction

Lemma 3: The number of nodes of rank k is at most n/2^k.
(every node has rank at most lg n).

proof: When a node x is assigned the rank k, label every node in
its subtree by x. Every node can be so labelled once.
Since there are 2^k such nodes, there can be no more than
n/2^k nodes labelled rank k.

partition the set of ranks into blocks.

B(0,j)=j
B(i,0)=0
B(1,j)=2^j for j>0.
B(i,1)= A(i,1)= A(i-1,2) i>1
B(i,j)=A(i,j) i,j>1, = A(i-1, A(i,j-1)).
B(alpha(m,n)+1, 1) =A(alpha(m,n), floor(m/n)) > lg n
alpha(m,n) is the minimum i s.t. A(i, floor(m/n)) > log n.

Understanding this:

block(i,j)= [B(i,j)...B(i,j+1)-1]

Note that each block at level 0 contains one number
In general, to figure out where block(i,j) starts, look at the first
number B(i,j-1) in block(i,j-1) and count 0,1,2, B(i,j-1)-1 blocks
on level i-1. Start block(i,j) where the B(i,j-1)^th block on level i-1
starts.

There is only one block for level alpha(m,n)+1 by the definition of alpha.

Let bij = the number of level i-1 blocks partitioning block (i,j).
b_i0=2 for i \in [1,...,alpha(m,n)]
b_ij <= A(i,j) for i \in [1,...,alpha(m,n)]

DEF: the LEVEL of node x is the minimum i such that
rank(x) and rank(p(x)) are in a common bloack of the level i partition.

If x=p(x), level(x)=0.

after a while, p(x)'s rank goes up while x's rank stays stable.

Allocate 1 credit for each makeset, 1 for each link,
and alpha(m,n) +2 for each find.

on the find path, need one credit on the path to
the root.

If level i <1 then x is a root.

For each level i>0, can assign 1 credit to the last node on level i in the
path.

What about the other nodes?
A node only incurs a cost if it has a parent, its rank is fixed.

Consider a node x in level i>0 which is not the last in its level.
rank(x) and rank(p(x) are in different blocks on level i-1.
and p(x) and the last node on the path are in different blocks on
level i-1 before the find, since x isn't the last node on its level.
So after the find, the new p(x) is in a new
block on level i-1.

Recall:
 bij = the number of level i-1 blocks partitioning block (i,j).
Then this can happen at most bij-1 times while rank(x) is in block(i,j)
while x is in level i.

After that, rank(x) and rank(p(x)) are in different level i blocks.

```
Let nij = the number of nodes with rank in block(i,j).
The total number of credits used is
  sum i_{1}^{alpha(m,n)+1} sum {j>=0}   n_ij (bij - 1)

For j=0, n_i0 <= n.
         b_i0=2.
For j>0,

 nij <= sum_{k=B(i,j)}^{B(i,j+1)-1} n/2^k.
     <= 2 n/(2^{B(i,j)-1})= n/2^{A(i,j)-1} for i\in [1,...,alpha(m,n)], j>0.

b_i0=2 for i \in [1,...,alpha(m,n)]

b_ij <= A(i,j)  for  i \in [1,...,alpha(m,n)]

and

B_{\alpha(m,n)+1, 0} = A(alpha(m,n), floor(m/n)), which is the min
i such that A(i, m/n)> lg n.
i.e., B_{\alpha(m,n)+1, 0} >= lg n.

So b_{\alpha(m,n)+1, 0} <= m/n.



 sum i_{1}^{alpha(m,n)+1} sum {j>=0}   n_ij (bij - 1)

<= n(2-1) + n(m/n-1) + n sum i_{1}^{alpha(m,n)} sum {j>=1}  A(i,j)/2^{A(i,j)-1}
<= n + m +n  alpha(m,n) sum r=1.. lg ceil(A(1,j)) 2^r/2^{2^r}
<= n + m + n (alpha(m,n) sum k=1..lg ceil(A(i,j) k/2^k
=O(m + n(alpha(m,n))).

???
 <= n + m + n sum i_{1}^{alpha(m,n)} sum r=1.. lg ceil(A(i,l)) 2^r/2^{A(i,j)-1}
since  sum r=1.. lg A(i,l) 2^r <= 2 A(i,l) where l is the highest j on level i.

 <= n + m + n sum i_{1}^{alpha(m,n)} 2 (A(i,l)+1)/ 2^{A(i,j)-1}
 <= n + m + n sum i_{1}^{alpha(m,n)}  A(i,l)+1/ 2^{A(i,j)-1}
?????
 <= n + m + n sum i_{1}^{alpha(m,n)} k+1/ 2^{k-2}
```

1,0-1 Top