2/11/98 day 10 Qoute of the day, "Someone has to be bigger than the other."

1 Union find

1.1 operations

- *union(u, v)*
- *is* $u = v$? = (find(v)=find(u))

used by Kruskal's algorithm

1.2 implementation

Have a large array (one entry per element) which points towards a representative member of each equivalence class. Each equivalence class has some root representative member.

1.2.1 implementing union

Union is implemented by linking the root of one tree to the root of another tree.

Improvement: (weighted union) Link the smaller tree to the larger tree to improve balance. It's necessary to keep along a 'size' field.

1.2.2 find

Recurse towards the root and return the root.

Improvement: (path compression) When you find the representative from an element, adjust every pointer in the path to point directly towards the root.

1.3 analysis

1.3.1 Tree height limitation

Let T, a tree. height(T)=#edge on max length path

lemma: with weighted union $|T| \geq 2^{height(T)}$

proof: assume $|T_1| \geq 2^{h(T_1)}$, $|T_2| \geq 2^{h(T_2)}$ and without loss of generality $|T_1| \geq |T_2|$

$T = union(T_1, T_2)$.

case 1: $height(T) = height(T_1)$. Ok, because the height hasn't increased while the size has.

tcase 2: $height(T) = 1 + height(T_2)$. Ok, because $2^{1+height(T_2)} = 2 * 2^{height(T_2)} \leq 2 * |T_2| \leq |T_2| + |T_1| = |T|$

1.3.2 Ackermann function

- $A_0(x) = x + 1$
- $A_{k+1}(x) = A_k^x(x)$ where $A_k^i(x) = A_k(A_k(...(A_k(x))..)$

This is pretty fast.

- $A_0(x) = x + 1$
- $A_1(x) = A_0^x(x) = 2x$
- $A_2(x) = A_1^x(x) = 2^x x$
- $A_3(x) = A_2^x(x) \geq 2^{2^{\cdot^{\cdot^{2}}}} x$ etc...

Now, define the ackermann function as $A(k) = A_k(2)$.

Define the inverse Ackermann as: $\alpha(n) = min\{k : A(k) \geq n\}$. This grows pretty slow.

1.3.3 union find analysis

Theorem: with m union find and n objects, the timing is $O((m + n)\alpha(n))$.

proof:

- $T_t(u)$ = subtree rooted at u at time t
- $rank_t(u) = 2 + height(T_t(u))$

Rank increases while a node is a root, but never increases when it is not a root.

lemma: if $\exists$ time s.t. u is a descendent of v, then $rank(u) \leq rank(v)$ then it will be true forever after.

lemma: max rank $\leq \lfloor log(n) \rfloor + 2$

proof: $n \geq |T(u)| \geq 2^{h(T(u))} \geq 2^{rank(u)-2}$

lemma: $|\{u|rank(u) = v\}| \leq \frac{n}{2^{v-2}}$ **proof: a matter of counting** define: $0 \leq \delta(u) = max\{k|rank(parent(u)) \geq A(rank(u))$. Then for $n \geq 5$, $\delta(n) \leq \alpha(n)$

Charging rules for find:

- if $\exists$ ancestor(x)=y s.t. $\delta(y) = \delta(x)$ then charge x
- else charge 'Find'

'Find' has $\alpha(n)$ cost because it can only be charged once for each value of δ.

Charge to x over all finds: let x' = parent(x), v = root

- $rank(x') \geq A_k(rank(x))$
- max i of rank(x') $\geq A_k^i(rank(x))$
- $rank(v) \geq rank(y') \geq A_k(rank(y) \geq A_k(rank(x')) \geq A_k(A_k^i(rank(x))) = A_k^{i+1}(rank(x))$

If x gets $rank(x)$ charges then $\delta(x)$ increases. This means it can in incur at most $rank(x)\alpha(n)$ charges. This is a rather large number in comparison to $\alpha(n)$ but we are save by the lemma limiting the number of nodes at a particular rank.

total charges $< \sum_{i=1}^{log(n)} r\alpha(n)(\frac{n}{2^{r-2}}) \leq n\alpha(n)\sum_{i=1}^{infinity} \frac{r}{2^{r-2}} = O(n\alpha(n))$

This means with m union-finds on a set of size n, the timing is $O((m+n)\alpha(n))$