Amortized Analysis of Union-Find

Ruijie Fang

1 Union-Find

The union-find algorithm we 'll be analyzing supports union and find operations, and implements them via union-by-rank as well as path compression. Let n be the total number of singleton sets in the beginning of a sequence σ of m operations. Let $T_t(u)$ denote the set of vertices in the subtree rooted at vertex u at operation $t, 1 \leq t \leq m$. Let $h(T(u))$ return the height of a subtree rooted at u. We define our rank function at vertex u at step t as

$$r(u) = 2 + h(T_t(u))$$

We call a vertex a root if its parent is itself. Initially, the parent of all vertices point to themselves.

1.1 Union

The union algorithm is straightforward:

If $r(x) \leq r(y)$, Union(x, y) links the root of x to the root of y and returns.

Else, $r(y) > r(x)$, and Union(x, y) links the root of y to the root of x and returns.

1.2 Find

Using path compression, we can implement find as

Find(x)

{ parent(x)$\leftarrow$Find(parent(x)); return parent(x); }

The Find($\cdot$)-operation can be decomposed into two operations: 1) An operation that takes time proportional to the length of the path from x to a root to return that root; 2) Another operation that takes the same number of steps, which resets the parent pointers of x and its ancestors to the root.

2 Ackermann Function and its Inverse

We define the Ackermann function as:

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

and $A(k) = A_k(2)$ for all $k \in \mathbb{N}$. Observe that A_k is monotone, i.e. $A_k(y) \geq A_k(x)$ if $y \geq x$.

The Ackermann function grows faster than all primitive recursive functions. We define $\alpha(n)$, the inverse Ackermann function, as

$$\alpha(n) = \min\{k; A_k(2) \geq n\}$$

For all practical purposes, $\alpha(n) \leq 4$.

3 Three Basic Lemmas

3.1 Lemma I: $|T_t(u)| \geq 2^{h(T_t(u))}$ for all $1 \leq t \leq m$ and any u.

Proof: By induction on m.

Base case: $m = 1$, $|T_1(u)| = 1$ for all u and $h(T_1(u)) = 0$.

Inductive case: Assume $|T_{t-1}(u)| \geq 2^{h(T_{t-1}(u))}$. If we do a find operation at step t, then $h(T_t(u)) = 1 \leq 2^{h(T_{t-1}(u))} \leq |T_{t-1}(u)| = |T_t(u)|$. If we do a union operation with vertex v:

1) if $r(u) \leq r(v)$, then we link subtree at u into subtree at v, and $|T_t(u)| = |T_{t-1}(u)| \geq 2^{h(T_{t-1}(u))} = 2^{h(T_t(u))}$ is invariant.

2) else, we link some smaller tree rooted at v into u, and since

$$|T_{t-1}(u)| \geq |T_{t-1}(v)|$$

so we have two situations, if $h(T_{t-1}(u)) \geq h(T_{t-1}(v)) + 1$ then $h(T_t(u)) =$

$h(T_{t-1}(u))$, and therefore we have

$$\begin{aligned} |T_t(u)| &= |T_{t-1}(u)| + |T_{t-1}(v)| \\ &\geq 2^{h(T_{t-1}(u))} \\ &= 2^{h(T_t(u))} \end{aligned}$$

or, $h(T_{t-1}(v)) + 1 > h(T_{t-1}(u))$ then $h(T_t(u)) = h(T_{t-1}(v)) + 1$ (one extra depth due to linking induced by union operation), and

$$\begin{aligned} |T_t(u)| &= |T_{t-1}(u)| + |T_{t-1}(v)| \\ &\geq 2 \cdot |T_{t-1}(v)| \\ &= 2 \cdot 2^{h(T_{t-1}(v))} \\ &= 2^{h(T_{t-1}(v))+1} \\ &= 2^{h(T_t(u))} \end{aligned}$$

QED.

3.2 Lemma II: The maximum rank after executing sequence σ is at most $\lfloor \log n \rfloor + 2$.

Proof:

Let $r_m(v)$ denote the rank of vertex v at step m.

By Lemma I:

$$\begin{aligned} n &\geq |T_m(v)| \\ &\geq 2^{h(T_m(v))} \\ \lfloor \log n \rfloor &\geq h(T_m(v)) \\ &= r_m(v) - 2 \\ r_m(v) &= \lfloor \log n \rfloor + 2 \end{aligned}$$

3.3 Lemma III: The number of vertices that have rank $r \leq n/2^{r-2}$.

Another way to state this is

$$|\{v; r(v) = r\}| \leq n/2^{r-2}.$$

Proof:
Observe that if $r(u) = r(v)$ then $T_m(u)$ and $T_m(v)$ are disjoint, so

$$
\begin{aligned}
n &\geq |\bigcup_{r(u)=r} T_m(u)| \\
&= \sum_{r(u)=r} |T_m(u)| \\
&\geq \sum_{r(u)=r} 2^{h(T_m(u))} \text{ by Lemma I} \\
&= \sum_{r(u)=r} 2^{r-2} \\
&= 2^{r-2}|\{u; r(u) = r\}| \\
|\{u; r(u) = r\} &\leq n/2^{r-2}.
\end{aligned}
$$

4 Analysis

4.1 A distance metric

Observe that $r(\text{parent}(u)) \geq r(u) + 1$ at all times. Let $\delta(u)$ be the greatest k such that

$$r(\text{parent}(u)) = A_k(r(u))$$

Note that such k always exists, since we can always let $k = 0$, and in which case $r(\text{parent}(u)) = r(u) + 1$. The larger $\delta(u)$, the larger the difference between the height of subtree at u and the subtree at parent(u); the large difference indicates the subtree at u must be among the shallowest subtrees of children of parent(u); if $\delta(u) = 0$, then the subtree at u must be the deepest subtree among the subtrees of children of parent(u). As we shall see, the setup of our charging scheme will depend on this intuition.

4.2 An upper bound on $\delta(u)$

How big can $\delta(u)$ be? Intuitively, it shouldn't be too large, which would indicate the tree is very deep at parent(u), which is not good. We now show an upper bound for $\delta(u) = k$.

$$
\begin{aligned}
n &> \lfloor \log n \rfloor + 2 \\
&\geq r(\text{parent}(u)) \text{ by Lemma II} \\
&\geq A_k(r(u)) \\
&\geq A_k(2)
\end{aligned}
$$

since $r(u) \geq 2$ by monotonicity of A_k. This implies

$$\alpha(n) > k = \delta(u)$$

4.3 The charging scheme

Union-type operations take constant time in our union find algorithm, so we focus on analyzing a sequence of m find operations (assuming we're operating on a forest of trees with nontrivial structure — i.e. they are not all singleton sets).

1) Find(x) for which there exists y in a path from x to root such that $\delta(y) = \delta(x)$. This models the situation like

$$x \rightarrow \dots \rightarrow y \rightarrow \dots \rightarrow \text{root}$$

In this case, we charge 1 time unit to the vertex x itself.

2) If there is no such y on a path from x to root, then we charge the time unit to the entire find operation.

4.4 Analysis of the charging scheme

2) is relatively easy to analyze. Assume that all m find operations cover vertices of type 2. But there are no more than $\alpha(n)$ distinct δ-values on a path from x to root (since no δ value occurs twice), so each charge is also upper bounded by $\alpha(n)$, and the total running time is bounded by $O((m+n)\alpha(n))$.

For 1), the situation requires a bit more work: Let x be the vertex which we initiate the find-operation. We know that

$$r(\text{parent}(y)) \geq A_k(r(y))$$
$$r(\text{parent}(x)) \geq A_k(r(x))$$

Suppose, in fact that $r(\text{parent}(x)) \geq A_k^i(r(x))$ for any $i \geq 1$.

for $k = \delta(x) = \delta(y)$. Let v be the root node (i.e. the last node on the path). Since the tree at $v$is larger than its subtrees,

$$\begin{aligned} r(v) &\geq r(\text{parent}(x)) \\ &\geq A_k(r(y)) \\ &\geq A_k(r(\text{parent}(x)) \text{ by monotonicity,} \\ &\geq A_k(A_k^i(r(x)) \\ &= A_k^{i+1}(r(x)) \end{aligned}$$

At most $r(x)$ charges are charged to x before $r(\text{parent}(x)) \geq A_k^{r(x)}(r(x)) = A_{k+1}(r(x))$, which causes $\delta(x) \geq k+1$.

$\therefore$ $\delta(x)$ increases by 1 after at most $r(x)$. Since $\delta(x)$ increases at most $\alpha(n) - 1$ times, there can be at most $r(x)\alpha(n)$ charges to every such vertex x of rank r. In total, this amounts to

$$r \cdot \alpha(n) \frac{n}{2^{r-2}}$$

chargest against vertices of rank r, and

$$\begin{aligned} \sum_{r=0}^{\infty} \alpha(n) \frac{nr}{2^{r-2}} &= n\alpha(n) \sum_{r=0}^{\infty} \frac{r}{2^{r-2}} \\ &= n\alpha(n) \cdot 8 \\ &= O(n\alpha(n)) \end{aligned}$$

total chargest against all such vertices. Taking the worst case of the two situations above, we arrive at the following theorem:

4.5 Theroem I: A sequence of m union and find operations starting from n singleton sets take $O((m+n)\alpha(n))$ worst-case time.