22.4 Analysis of union by rank with path compression

\[ A(4,1) \text{. But we also have that} \]
\[ A(4,1) = A(3,2) \]
\[ = 2^{2^{\cdot\cdot\cdot^{2}}} \text{,} \]

which is far greater than the estimated number of atoms in the observable universe (roughly \(10^{80}\)). It is only for impractically large values of \( n \) that \( A(4,1) \leq \lg n \), and thus \( \alpha(m,n) \leq 4 \) for all practical purposes. Note that the \( O(m \lg^* n) \) bound is only slightly weaker than the \( O(m \alpha(m,n)) \) bound; \( \lg^* 65536 = 4 \) and \( \lg^* 2^{65536} = 5 \), so \( \lg^* n \leq 5 \) for all practical purposes.

**Properties of ranks**

In the remainder of this section, we prove an \( O(m \lg^* n) \) bound on the running time of the disjoint-set operations with union by rank and path compression. In order to prove this bound, we first prove some simple properties of ranks.

**Lemma 22.2**

For all nodes \( x \), we have \( \text{rank}[x] \leq \text{rank}[p[x]] \), with strict inequality if \( x \neq p[x] \). The value of \( \text{rank}[x] \) is initially 0 and increases through time until \( x \neq p[x] \); from then on, \( \text{rank}[x] \) does not change. The value of \( \text{rank}[p[x]] \) is a monotonically increasing function of time.

**Proof** The proof is a straightforward induction on the number of operations, using the implementations of \textsc{Make-Set}, \textsc{Union}, and \textsc{Find-Set} that appear in Section 22.3. We leave it as Exercise 22.4.1.

We define \( \text{size}(x) \) to be the number of nodes in the tree rooted at node \( x \), including node \( x \) itself.

**Lemma 22.3**

For all tree roots \( x \), \( \text{size}(x) \geq 2^{\text{rank}[x]} \).

**Proof** The proof is by induction on the number of \textsc{Link} operations. Note that \textsc{Find-Set} operations change neither the rank of a tree root nor the size of its tree.

**Basis:** The lemma is true before the first \textsc{Link}, since ranks are initially 0 and each tree contains at least one node.

**Inductive step:** Assume that the lemma holds before performing the operation \textsc{Link}(\( x, y \)). Let \( \text{rank} \) denote the rank just before the \textsc{Link}, and let \( \text{rank}' \) denote the rank just after the \textsc{Link}. Define \( \text{size} \) and \( \text{size}' \) similarly.

If \( \text{rank}[x] \neq \text{rank}[y] \), assume without loss of generality that \( \text{rank}[x] < \text{rank}[y] \). Node \( y \) is the root of the tree formed by the \textsc{Link} operation, and

\[
\text{size}'(y) = \text{size}(x) + \text{size}(y)
\]
\[ \geq 2^{\text{rank}[x]} + 2^{\text{rank}[y]} \]
\[ \geq 2^{\text{rank}[y]} \]
\[ = 2^{\text{rank}'[y]} . \]

No ranks or sizes change for any nodes other than \( y \).
If \( \text{rank}[x] = \text{rank}[y] \), node \( y \) is again the root of the new tree, and

\[
\text{size}'(y) = \text{size}(x) + \text{size}(y)
\geq 2^{\text{rank}[x]} + 2^{\text{rank}[y]}
= 2^{\text{rank}[y]+1}
= 2^{\text{rank}'[y]} .
\]

**Lemma 22.4**

For any integer \( r \geq 0 \), there are at most \( n/2' \) nodes of rank \( r \).

**Proof** Fix a particular value of \( r \). Suppose that when we assign a rank \( r \) to a node \( x \) (in line 2 of \textsc{Make-Set} or in line 5 of \textsc{Link}), we attach a label \( x \) to each node in the tree rooted at \( x \). By Lemma 22.3, at least \( 2' \) nodes are labeled each time. Suppose that the root of the tree containing node \( x \) changes. Lemma 22.2 assures us that the rank of the new root (or, in fact, of any proper ancestor of \( x \)) is at least \( r+1 \). Since we assign labels only when a root is assigned a rank \( r \), no node in this new tree will ever again be labeled. Thus, each node is labeled at most once, when its root is first assigned rank \( r \). Since there are \( n \) nodes, there are at most \( n \) labeled nodes, with at least \( 2' \) labels assigned for each node of rank \( r \). If there were more than \( n/2' \) nodes of rank \( r \), then more than \( 2' \cdot (n/2') = n \) nodes would be labeled by a node of rank \( r \), which is a contradiction. Therefore, at most \( n/2' \) nodes are ever assigned rank \( r \).

**Corollary 22.5**

Every node has rank at most \([\lg n]\).

**Proof** If we let \( r > \lg n \), then there are at most \( n/2' < 1 \) nodes of rank \( r \).
Since ranks are natural numbers, the corollary follows.

**Proving the time bound**

We shall use the aggregate method of amortized analysis (see Section 18.1) to prove the \( O(m \lg^* n) \) time bound. In performing the amortized analysis, it is convenient to assume that we invoke the \textsc{Link} operation rather than the \textsc{Union} operation. That is, since the parameters of the \textsc{Link} procedure are pointers to two roots, we assume that the appropriate \textsc{Find-Set} operations are performed if necessary. The following lemma shows that even
if we count the extra FIND-SET operations, the asymptotic running time remains unchanged.

**Lemma 22.6**

Suppose we convert a sequence $S'$ of $m'$ MAKE-SET, UNION, and FIND-SET operations into a sequence $S$ of $m$ MAKE-SET, LINK, and FIND-SET operations by turning each UNION into two FIND-SET operations followed by a LINK. Then, if sequence $S$ runs in $O(m' \lg^* n)$ time, sequence $S'$ runs in $O(m' \lg^* n)$ time.

**Proof** Since each UNION operation in sequence $S'$ is converted into three operations in $S$, we have $m' \leq m \leq 3m'$. Since $m = O(m')$, an $O(m' \lg^* n)$ time bound for the converted sequence $S'$ implies an $O(m' \lg^* n)$ time bound for the original sequence $S'$.

In the remainder of this section, we shall assume that the initial sequence of $m'$ MAKE-SET, UNION, and FIND-SET operations has been converted to a sequence of $m$ MAKE-SET, LINK, and FIND-SET operations. We now prove an $O(m \lg^* n)$ time bound for the converted sequence and appeal to Lemma 22.6 to prove the $O(m' \lg^* n)$ running time of the original sequence of $m'$ operations.

**Theorem 22.7**

A sequence of $m$ MAKE-SET, LINK, and FIND-SET operations, $n$ of which are MAKE-SET operations, can be performed on a disjoint-set forest with union by rank and path compression in worst-case time $O(m \lg^* n)$.

**Proof** We assess charges corresponding to the actual cost of each set operation and compute the total number of charges assessed once the entire sequence of set operations has been performed. This total then gives us the actual cost of all the set operations.

The charges assessed to the MAKE-SET and LINK operations are simple: one charge per operation. Since these operations each take $O(1)$ actual time, the charges assessed equal the actual costs of the operations.

Before discussing charges assessed to the FIND-SET operations, we partition node ranks into blocks by putting rank $r$ into block $\lg^* r$ for $r = 0, 1, \ldots, \lfloor \lg n \rfloor$. (Recall that $\lfloor \lg n \rfloor$ is the maximum rank.) The highest-numbered block is therefore block $\lg^* (\lfloor \lg n \rfloor) = \lg^* n - 1$. For notational convenience, we define for integers $j \geq -1$,

$$B(j) = \begin{cases} 
-1 & \text{if } j = -1, \\
1 & \text{if } j = 0, \\
2 & \text{if } j = 1, \\
2^2 \ldots 2^{j-1} & \text{if } j \geq 2. 
\end{cases}$$

Then, for $j = 0, 1, \ldots, \lg^* n - 1$, the $j$th block consists of the set of ranks
\{B(j - 1) + 1, B(j - 1) + 2, \ldots, B(j)\}.

We use two types of charges for a **Find-Set** operation: **block charges** and **path charges**. Suppose that the **Find-Set** starts at node \(x_0\) and that the find path consists of nodes \(x_0, x_1, \ldots, x_l\), where for \(i = 1, 2, \ldots, l\), node \(x_i\) is \(p[x_{i-1}]\) and \(x_l\) (a root) is \(p[x_l]\). For \(j = 0, 1, \ldots, \lg^* n - 1\), we assess one block charge to the last node with rank in block \(j\) on the path. (Note that Lemma 22.2 implies that on any find path, the nodes with ranks in a given block are consecutive.) We also assess one block charge to the child of the root, that is, to \(x_{l-1}\). Because ranks strictly increase along any find path, an equivalent formulation assesses one block charge to each node \(x_i\) such that \(p[x_i] = x_i\) (\(x_i\) is the root or its child) or \(\lg^* \text{rank}[x_i] < \lg^* \text{rank}[x_{i+1}]\) (the block of \(x_i\)'s rank differs from that of its parent). At each node on the find path for which we do not assess a block charge, we assess one path charge.

Once a node other than the root or its child is assessed block charges, it will never again be assessed path charges. To see why, observe that each time path compression occurs, the rank of a node \(x_i\) for which \(p[x_i] \neq x_i\) remains the same, but the new parent of \(x_i\) has a rank strictly greater than that of \(x_i\)'s old parent. The difference between the ranks of \(x_i\) and its parent is a monotonically increasing function of time. Thus, the difference between \(\lg^* \text{rank}[p[x_i]]\) and \(\lg^* \text{rank}[x_i]\) is also a monotonically increasing function of time. Once \(x_i\) and its parent have ranks in different blocks, they will always have ranks in different blocks, and so \(x_i\) will never again be assessed a path charge.

Since we have charged once for each node visited in each **Find-Set** operation, the total number of charges assessed is the total number of nodes visited in all the **Find-Set** operations; this total represents the actual cost of all the **Find-Set** operations. We wish to show that this total is \(O(m \lg^* n)\).

The number of block charges is easy to bound. There is at most one block charge assessed for each block number on the given find path, plus one block charge for the child of the root. Since block numbers range from 0 to \(\lg^* n - 1\), there are at most \(\lg^* n + 1\) block charges assessed for each **Find-Set** operation. Thus, there are at most \(m(\lg^* n + 1)\) block charges assessed over all **Find-Set** operations.

Bounding the path charges is a little trickier. We start by observing that if a node \(x_i\) is assessed a path charge, then \(p[x_i] \neq x_i\) before path compression, so that \(x_i\) will be assigned a new parent during path compression. Moreover, as we have observed, \(x_i\)'s new parent has a higher rank than its old parent. Suppose that node \(x_i\)'s rank is in block \(j\). How many times can \(x_i\) be assigned a new parent, and thus assessed a path charge, before \(x_i\) is assigned a parent whose rank is in a different block (after which \(x_i\) will never again be assessed a path charge)? This number of times is maximized if \(x_i\) has the lowest rank in its block, namely \(B(j - 1) + 1\), and its parents' ranks successively take on the values \(B(j - 1) + 2, B(j - 1) + 3, \ldots, B(j)\).
Since there are $B(j) - B(j-1) - 1$ such ranks, we conclude that a vertex can be assessed at most $B(j) - B(j-1) - 1$ path charges while its rank is in block $j$.

Our next step in bounding the path charges is to bound the number of nodes that have ranks in block $j$ for integers $j \geq 0$. (Recall that by Lemma 22.2, the rank of a node is fixed once it becomes a child of another node.) Let the number of nodes whose ranks are in block $j$ be denoted by $N(j)$. Then, by Lemma 22.4,

$$N(j) \leq \sum_{r=B(j-1)+1}^{B(j)} \frac{n}{2^r}.$$  

For $j = 0$, this sum evaluates to

$$N(0) = n/2^0 + n/2^1 = 3n/2 = 3n/2B(0).$$

For $j \geq 1$, we have

$$N(j) \leq \frac{n}{2B(j-1)+1} \sum_{r=0}^{B(j)-(B(j-1)+1)} \frac{1}{2^r}$$

$$< \frac{n}{2B(j-1)+1} \sum_{r=0}^{\infty} \frac{1}{2^r}$$

$$= \frac{n}{2B(j-1)}$$

$$= \frac{n}{B(j)}.$$ 

Thus, $N(j) \leq 3n/2B(j)$ for all integers $j \geq 0$.

We finish bounding the path charges by summing over all blocks the product of the maximum number of nodes with ranks in the block and the maximum number of path charges per node of that block. Denoting by $P(n)$ the overall number of path charges, we have

$$P(n) \leq \sum_{j=0}^{\log^* n - 1} \frac{3n}{2B(j)} (B(j) - B(j-1) - 1)$$

$$\leq \sum_{j=0}^{\log^* n - 1} \frac{3n}{2B(j)} \cdot B(j)$$

$$= \frac{3}{2} n \log^* n.$$ 

Thus, the total number of charges incurred by FIND-SET operations is $O(m (\log^* n + 1) + n \log^* n)$, which is $O(m \log^* n)$ since $m \geq n$. Since there are $O(n)$ MAKE-SET and LINK operations, with one charge each, the total time is $O(m \log^* n)$. 

\[\square\]