1 Union Find

We want to maintain a collection of sets, under the operations of:

1. MakeSet(x) - create a set $x$
2. Find(x) - return the set that contains $x$.
3. Union(A,B) - return the set which is the union of $A$ and $B$. Namely $A \cup B$.

1.1 Amortized analysis

We use a data-structure as a black-box inside an algorithm (for example Union-Find in Kruskal algorithm). So far, when we design a data-structure we cared about worst case time for operation.

Note, that we care about the OVERALL running time of the data-structure. Less about its running time for a single operation. Amortized running time of operation = (overall running time)/(number of operations).

To implement this operations, we are going to use Reversed Trees.

\[
\begin{align*}
\text{MakeSet}(x) \\
& \quad \text{parent}(x) \leftarrow x \\
& \quad \text{rank}(x) \leftarrow 0
\end{align*}
\]

\[
\begin{align*}
\text{Find}(x) \\
& \quad \text{if } x \neq \text{parent}(x) \text{ then} \\
& \quad \quad \text{parent}(x) \leftarrow \text{Find}(	ext{parent}(x)) \\
& \quad \text{return parent}(x)
\end{align*}
\]
This is known as union by rank and path compression.

**Definition 1.1** A node in the UF data-structure is a leader if it is the root of a tree.

**Lemma 1.2** Once a node stop being a leader (i.e., the node in top of a tree), it can never become a leader again.

**Lemma 1.3** Once a node stop being a leader than its rank is fixed.

**Lemma 1.4** Ranks are monotonically increasing in the reversed trees, as we travel for a node to the root of the tree.

**Lemma 1.5** When a node gets rank $k$ than there are at least $\geq 2^k$ elements in its subtree.

*Proof:* The proof is by induction. For $k = 0$ it is obvious. Next observe that a node gets rank $k$ only if the merged two roots has rank $k - 1$. By induction, they have $2^{k-1}$ nodes (each one of them), and thus the merged tree has $\geq 2^{k-1} + 2^{k-1} = 2^k$ nodes.

**Lemma 1.6** The number of nodes of rank $k$ during all the execution of the Union-Find data-structure is at most $n/2^k$.

*Proof:* Again, by induction. For $k = 0$ it is obvious. We charge a node $v$ of rank $k$ to the two elements of rank $k - 1$ that were leaders that were used to create it – one of them is $v$ having degree $k - 1$, the other one is some other node $u$. After the merge $v$ is of rank $k$ and $u$ is of rank $k - 1$ and it is no longer a leader (it can not participate in a union as a leader). Thus we can charge this event to the two no longer active nodes of degree $k - 1$. Namely, $u$ and $v$. By induction, we have $n/2^{k-1}$such nodes, and thus $\leq (n/2^{k-1})/2 = n/2^k$ such nodes of degree $k$.

**Lemma 1.7** The time to perform a single Find operation when we perform union by rank and path compression is $O(\log n)$ time.

*Proof:* The rank of the leader bounds the depth of a tree $T$ in the Union Find data-structure. By the above lemma, if we have $n$ elements, the maximum rank is $\log n$ and thus the depth of a tree is at most $O(\log n)$. 
Theorem 1.8 If we perform a sequence of \( m \) operations over \( n \) elements, the overall running time of the Union-Find data-structure is \( O((n + m) \log^* n) \).

Definition 1.9 \( \text{Tower}(b) = 2^{\text{Tower}(b-1)} \) and \( \text{Tower}(0) = 1 \).

Definition 1.10 \( \text{Block}(i) = [\text{Tower}(i - 1) + 1, \text{Tower}(i)] \). Namely, \( \text{Block}(i) = [z, 2^z - 1] \) for \( z = \text{Tower}(i - 1) + 1 \).

Observation 1.11 The running time of Find\((x)\) is proportional to the length of the path from \( x \) to the root of the tree that contains \( x \). Indeed, we start from \( x \) and we visit the sequence: \( x_1 = x, x_2 = \text{parent}(x) = \text{parent}(x_1), \ldots, x_i = \text{parent}(x_{i-1}), \ldots, x_m = \text{root} \).

Clearly, we have for this sequence:
\[ \text{rank}(x_1) < \text{rank}(x_2) < \text{rank}(x_3) < \ldots < \text{rank}(x_m). \]

Note, that the time to perform find, is proportional to \( m \).

Definition 1.12 A node \( x \) is in the \( i \)-th block if \( \text{rank}(x) \in \text{Block}(i) \).

We are now looking for ways to pay for the find operation.

Observation 1.13 The rank of a node \( v \) is \( O(\log n) \), and the number of blocks is \( O(\log^* n) \).

Observation 1.14 During a find operation, since the ranks of the nodes we visit are monotone increasing, once we pass through from a node \( v \) in the \( i \)-th block into a node in the \( (i+1) \)-th block, we can never go back to the \( i \)-th block (i.e., visit elements with rank in the \( i \)-th block).

Lemma 1.15 During a Find operation, the number of jumps between blocks is \( O(\log^* n) \).

Observation 1.16 If \( x \) and \( \text{parent}(x) \) are in the same block and we perform a find operation that passes through \( x \). Let \( r_{\text{before}} = \text{rank}(\text{parent}(x)) \) before the find operation, and let \( r_{\text{after}} \) be \( \text{rank}(\text{parent}(x)) \) after the Find operation. Then because of path compression, we have \( r_{\text{after}} > r_{\text{before}} \).

Namely, when we jump inside a block, we do some work: we make the parent point jump forward.

Definition 1.17 A jump during a find operation inside the \( i \)-th block is called an internal jump.

Lemma 1.18 At most \( |\text{Block}(i)| \) find operations can pass through an element \( x \) which is in the \( i \)-th block (i.e., \( \text{rank}(x) \in \text{Block}(i) \)) before \( \text{parent}(x) \) is no longer in the \( i \)-th block.

Lemma 1.19 There are at most \( n/\text{Tower}(i) \) nodes that have ranks in the \( i \)-th block throughout the algorithm execution.
Proof: Clearly,
\[
\sum_{i=\text{Tower}(i-1)+1}^{\text{Tower}(i)} \frac{n}{2^i} = n \cdot \sum_{i=\text{Tower}(i-1)+1}^{\text{Tower}(i)} \frac{1}{2^i} \leq \frac{n}{\text{tower}(i-1)} = \frac{n}{\text{Tower}(i)}.
\]

**Lemma 1.20** The number of inner block jumps performed inside the i-th block performed during the lifetime of the union-find data-structure is \(O(n)\).

Proof: An element \(x\) in the \(i\)-th block, can have \(|\text{Block}(i)|\) jumps. There are \(n/\text{Tower}(i)\) such elements. Thus, the total number of internal jumps is
\[
|\text{Block}(i)| \cdot \frac{n}{\text{Tower}(i)} \leq \text{Tower}(i) \cdot \frac{n}{\text{Tower}(i)} = n.
\]

**Lemma 1.21** The number of internal jumps performed by the Union-Find data-structure overall is \(O(n \log^* n)\).

Proof: For every block, we perform at most \(O(n)\) internal jumps. We have \(O(\log^* n)\) blocks. Thus, the overall number of internal jumps is at most \(O(n \log^* n)\).

**Lemma 1.22** The overall time spent on \(m\) find operations is \(O((m + n) \log^* n)\).