Iterated Functions

\[ f^*(n) = \begin{cases} 
0 & \text{if } n \leq 1 \\
1 + f^*(f(n)) & \text{if } n > 1
\end{cases} \]

\[ = \min \left\{ i \geq 0 : f \left( f \left( f \left( \ldots f(n) \ldots \right) \right) \right) \leq 1 \right\} \]

\[ = \min \{ i \geq 0 : f^{(i)}(n) \leq 1 \}, \]

where \( f^{(i)}(n) = \begin{cases} 
n & \text{if } i = 0 \\
f \left( f^{(i-1)}(n) \right) & \text{if } i > 0
\end{cases} \)

**Example:** If \( f = \log \), we have:

\[
\begin{align*}
\log^{(0)}(65536) &= 65536 & \log^{(3)}(65536) &= 2 \\
\log^{(1)}(65536) &= 16 & \log^{(4)}(65536) &= 1 \\
\log^{(2)}(65536) &= 4 & \therefore \log^*(65536) &= 4
\end{align*}
\]
## Iterated Functions

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$f^*(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n - 1$</td>
<td>$n - 1$</td>
</tr>
<tr>
<td>$n - 2$</td>
<td>$\frac{n}{2}$</td>
</tr>
<tr>
<td>$n - c$</td>
<td>$\frac{n}{c}$</td>
</tr>
<tr>
<td>$n - c$</td>
<td>$\log_2 n$</td>
</tr>
<tr>
<td>$\sqrt{n}$</td>
<td>$\log \log n$</td>
</tr>
<tr>
<td>$\log n$</td>
<td>$\log^* n$</td>
</tr>
</tbody>
</table>
### The Inverse Ackermann Function: $\alpha(n)$

<table>
<thead>
<tr>
<th>$f(n)$</th>
<th>$f^*(n)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\log n$</td>
<td>$\log^* n$</td>
</tr>
<tr>
<td>$\log^* n$</td>
<td>$\log^{**} n$</td>
</tr>
<tr>
<td>$\log^{**} n$</td>
<td>$\log^{***} n$</td>
</tr>
<tr>
<td>$\log^{k-2} n$</td>
<td>$\log^{k-1} n$</td>
</tr>
<tr>
<td>$\log^{k-1} n$</td>
<td>$\log^k n$</td>
</tr>
</tbody>
</table>

$k = \alpha(n)$

\[ \alpha(n) = \min \left\{ k \geq 1 : \log^{k-1} n \leq 3 \right\} \]
Union-Find: A Disjoint-Set Data Structure
Disjoint Set Operations

A disjoint-set data structure maintains a collection of disjoint dynamic sets. Each set is identified by a representative which must be a member of the set.

The collection is maintained under the following operations:

**MAKE-SET( x ):** create a new set \( \{x\} \) containing only element \( x \).

Element \( x \) becomes the representative of the set.

**FIND( x ):** returns a pointer to the representative of the set containing \( x \)

**UNION( x, y ):** replace the dynamic sets \( S_x \) and \( S_y \) containing \( x \) and \( y \), respectively, with the set \( S_x \cup S_y \)
Union-Find Data Structure
with Union by Rank and Find with Path Compression

**MAKE-SET** (x)
1. \( \pi(x) \leftarrow x \)
2. \( \text{rank}(x) \leftarrow 0 \)

**LINK** (x, y)
1. \( \text{if} \ \text{rank}(x) > \text{rank}(y) \ \text{then} \ \pi(y) \leftarrow x \)
2. \( \text{else} \ \pi(x) \leftarrow y \)
3. \( \text{if} \ \text{rank}(x) = \text{rank}(y) \ \text{then} \ \text{rank}(y) \leftarrow \text{rank}(y) + 1 \)

**UNION** (x, y)
1. **LINK**( FIND (x), FIND (y) )

**FIND** (x)
1. \( \text{if} \ x \neq \pi(x) \ \text{then} \ \pi(x) \leftarrow \text{FIND} (\pi(x)) \)
2. return \( \pi(x) \)
Some Useful Properties of Rank

- If $x$ is not a root then $\text{rank}(x) < \text{rank}(\pi(x))$
- Node ranks strictly increase along any simple path towards a root
- Once a node becomes a non-root its rank never changes
- If $\pi(x)$ changes from $y$ to $z$ then $\text{rank}(z) > \text{rank}(y)$
- If the root of $x$'s tree changes from $y$ to $z$ then $\text{rank}(z) > \text{rank}(y)$
- If $x$ is the root of a tree then $\text{size}(x) \geq 2^{\text{rank}(x)}$
- If there are only $n$ nodes the highest possible rank is $\lceil \log_2 n \rceil$
- There are at most $\frac{n}{2^r}$ nodes with rank $r \geq 0$
Some Useful Properties of Rank

- We will analyze the total running time of $m'$ MAKE-SET, UNION and FIND operations of which exactly $n \ (\leq m')$ are MAKE-SET
- But each UNION can be replaced with two FIND and one LINK
- Hence, we can simply analyze the total running time of $m$ MAKE-SET, LINK and FIND operations of which exactly $n \ (\leq m)$ are MAKE-SET and where $m' \leq m \leq 3m'$
Compress

```plaintext
Compress (x, y) \{ y is an ancestor of x \}
1. if x \neq y then \pi(x) \leftarrow Compress (\pi(x), y)
2. return \pi(x)
```

- We will analyze the total running time of \( m \) MAKE-SET, LINK and FIND operations of which exactly \( n (\leq m) \) are MAKE-SET.
- But FIND\( (x) \) is nothing but Compress\( (x, y) \), where \( y \) is the root of the tree containing \( x \).
- Hence, we can analyze the total running time of \( m \) MAKE-SET, LINK and Compress operations of which exactly \( n (\leq m) \) are MAKE-SET.
Compress

\[ \text{Compress} \left( x, y \right) \quad \text{\{ } y \text{ is an ancestor of } x \text{ \}} \]

1. \text{if } x \neq y \text{ then } \pi(x) \leftarrow \text{Compress} \left( \pi(x), y \right) \\
2. \text{return } \pi(x) \\

We can reorder the sequence of Link and Compress operations so that all Link’s are performed before all Compress operations without changing the number of parent pointer reassignments!
\textsc{Shatter}(x)

1. \textit{if} $x \neq \pi(x)$ \textit{then} \textsc{Shatter}(\pi(x))
2. $\pi(x) \leftarrow x$

\hspace{1cm}

$w$
\hspace{1cm}
\hspace{1cm}
\hspace{1cm}
$z$
\hspace{1cm}
\hspace{1cm}
$y$
\hspace{1cm}
\hspace{1cm}
$x$

$w$
\hspace{1cm}
\hspace{1cm}
\hspace{1cm}
$z$
\hspace{1cm}
\hspace{1cm}
$y$
\hspace{1cm}
\hspace{1cm}
$x$
Let $T(m, n, r) =$ worst-case number of parent pointer assignments

- during any sequence of at most $m$ COMPRESS operations
- on a forest of $n$ nodes
- with maximum rank $r$

**Bound 0:** $T(m, n, r) \leq nr$.

**Proof:** Since there are at most $r$ distinct ranks, and each new parent of a node has a higher rank than its previous parent, any node can change parents fewer than $r$ times.
Bound 1

Bound 1: $T(m, n, r) \leq m + 2n \log^* r$.

Proof: Let $F$ be the forest, and $C$ be the sequence of COMPRESS operations performed on $F$.

Let $T(F, C)$ be the number of parent pointer assignments by $C$ in $F$.

Let $s$ be an arbitrary rank. We partition $F$ into two subforests:

- $F_b$ containing all nodes with rank $\leq s$, and
- $F_t$ containing all nodes with rank $> s$.

\[ F_b \quad \text{containing all nodes with rank } \leq s, \quad \text{and} \]
\[ F_t \quad \text{containing all nodes with rank } > s. \]
Bound 1: \( T(m, n, r) \leq m + 2n \log^* r \).

Proof: Let \( s \) be an arbitrary rank. We partition \( F \) into two subforests:

- \( F_b \) containing all nodes with rank \( \leq s \), and
- \( F_t \) containing all nodes with rank \( > s \).

Let \( n_t = \#\text{nodes in } F_t \), and \( n_b = \#\text{nodes in } F_b \)

Let \( m_t = \#\text{COMPRESS operations with at least one node in } F_t \), and

\( m_b = m - m_t \)
Bound 1

Bound 1: \( T(m, n, r) \leq m + 2n \log^* r. \)

Proof: The sequence \( C \) on \( F \) can be decomposed into

- a sequence of \textsc{Compress} operations in \( F_t \), and
- a sequence of \textsc{Compress} and \textsc{Shatter} operations in \( F_b \)

Suppose, this decomposition partitions \( C \) into two subsequences

- \( C_t \) in \( F_t \), and
- \( C_b \) in \( F_b \)
Bound 1

**Bound 1:** $T(m, n, r) \leq m + 2n \log^* r$.

**Proof:** We get the following recurrence:

$$T(F, C) \leq T(F_t, C_t) + T(F_b, C_b) + m_t + n_b$$

<table>
<thead>
<tr>
<th>Cost on Left Side</th>
<th>Corresponding Cost on Right Side</th>
</tr>
</thead>
<tbody>
<tr>
<td>node $\in F_t$ gets new parent $\in F_t$</td>
<td>$T(F_t, C_t)$</td>
</tr>
<tr>
<td>node $\in F_b$ gets new parent $\in F_b$</td>
<td>$T(F_b, C_b)$</td>
</tr>
<tr>
<td>node $\in F_b$ gets new parent $\in F_t$</td>
<td>$n_b$</td>
</tr>
<tr>
<td>( for the first time )</td>
<td></td>
</tr>
<tr>
<td>node $\in F_b$ gets new parent $\in F_t$</td>
<td>$m_t$</td>
</tr>
<tr>
<td>( again )</td>
<td></td>
</tr>
</tbody>
</table>
Bound 1

Bound 1: \( T(m, n, r) \leq m + 2n \log^* r \).

Proof: We get the following recurrence:

\[
T(F, C) \leq T(F_t, C_t) + T(F_b, C_b) + m_t + n_b
\]

Now \( n_t \leq \sum_{i>s} \frac{n}{2^i} = \frac{n}{2^s} \), and \( r_t = r - s < r \).

Hence, using bound 0: \( T(F_t, C_t) \leq n_t r_t < \frac{nr}{2^s} \)

Let \( s = \log r \). Then \( T(F_t, C_t) < n \).

Hence, \( T(F, C) \leq T(F_b, C_b) + m_t + 2n \)

\[\Rightarrow T(F, C) - m \leq T(F_b, C_b) - m_b + 2n\]
**Bound 1**

**Bound 1:** \( T(m, n, r) \leq m + 2n \log^* r \).

**Proof:**

We got \( T(F, C) - m \leq T(F_b, C_b) - m_b + 2n \)

Let \( T_1(m, n, r) = T(m, n, r) - m \)

Then \( T_1(m, n, r) \leq T_1(m_b, n_b, r_b) + 2n \)

\[ \Rightarrow T_1(m, n, r) \leq T_1(m, n, \log r) + 2n \]

Solving, \( T_1(m, n, r) \leq 2n \log^* r \)

Hence, \( T(m, n, r) \leq m + 2n \log^* r \)
Bound 2: $T(m, n, r) \leq 2m + 3n \log** r$.

Proof: Similar to the proof of bound 1.

But we solve $T(F_t, C_t)$ using bound 1, instead of bound 0!

We fix $s = \log^* r$ (instead of $\log r$ for bound 1)

Then using bound 1: $T(F_t, C_t) \leq m_t + 2n_t \log^* r_t$

$\leq m_t + 2 \frac{n}{2\log^* r} \log^* r$

$\leq m_t + 2n$

Then from $T(F, C) \leq T(F_t, C_t) + T(F_b, C_b) + m_t + n_b$, we get

$T(F, C) \leq T(F_b, C_b) + 2m_t + 3n_b$
Bound 2: \( T(m, n, r) \leq 2m + 3n \log^{**} r. \)

Proof: Our recurrence:

\[
T(F, C) \leq T(F_b, C_b) + 2m_t + 3n_b
\]

\[
\Rightarrow T(F, C) - 2m \leq T(F_b, C_b) - 2m_b + 3n_b
\]

Let \( T_2(m, n, r) = T(m, n, r) - 2m \)

Then \( T_2(m, n, r) \leq T_2(m_b, n_b, r_b) + 3n \)

\[
\Rightarrow T_2(m, n, r) \leq T_2(m, n, \log^* r) + 3n
\]

Solving, \( T_2(m, n, r) \leq 3n \log^{**} r \)

Hence, \( T(m, n, r) \leq 2m + 3n \log^{**} r \)
**Bound $k$**

**Bound $k$:** $T(m, n, r) \leq km + (k + 1)n \log^{k} r$.

**Observation:** As we increase $k$:
- the dependency on $m$ increases
- the dependency on $r$ decreases

When $k = \alpha(r)$, we have $\log^{k} r \leq 3$!

**Bound $\alpha$:** $T(m, n, r) \leq m\alpha(r) + 3(\alpha(r) + 1)n$. 
The $\alpha$ Bound

**Bound $\alpha$:** $T(m,n,r) \leq m\alpha(r) + 3(\alpha(r) + 1)n$.

Observing that $r < n$, we have:

**Bound $\alpha$:** $T(m,n,r) \leq (m + 3n)\alpha(n) + 3n = O((m + n)\alpha(n))$.

Assuming $m \geq n$, we have:

**Bound $\alpha$:** $T(m,n,r) = O(m\alpha(n))$.

So, amortized complexity of each operation is only $O(\alpha(n))$!