- Here we consider the disjoint-set forest with union by rank and path compression heuristics. We prove the amortized time complexity of m number of make-set, union, find-set operations, in which n are make-set operations, on an initially empty data strcture is $O(m \lg^* n)$.

- The *rank* of any node v of a disjoint-set forest is an upper bound on the height of v. For the sake of completeness, the pseudocode from [CLRS] is listed at the end of this note. The following obvious properties are derived from the pseudocode:

  – None of the make-set, union, and find-set operations cause the rank of any node to decrease.

    Only the link operation could change the rank of a node.

  – If any node v becomes a child of another node, then onwards, rank of v won't change. Hence, only ranks of tree roots could be modified by the link operation.

  – The link operation increases the rank of $T.root$ by at most one. This increase is exactly one whenever another tree T', whose root's rank equal to $T.root$, is linked as a child of T's root.

  – For any node v, the ranks of nodes that occur along the simple path from v to root strictly increase.

  – A node's parent may change or the parent's rank may change: the former happens via a path compression whereas the latter occurs when the parent is a root and its rank got increased via a link operation.

  – Each union operation instantiates two find-set operations and at most one link operation. Hence, m make-set, union, and find-set operations are effectively $O(m)$ make-set, link, and find-set operations.

- Lemma 1: For any tree root v, the number of nodes in T_v is lower bounded by $2^{v.rank}$.

  - Proof is by induction on the number of link operations.

    As part of induction step, in linking a tree rooted at v' as a child of a tree rooted at v, there are two cases to consider: $v'.rank < v.rank$ and $v'.rank = v.rank$.

  Lemma 2: If x is a non-root node in a tree rooted at v when $v.rank$ is set to r, then from there on, x can never be in a tree whose root's rank gets set to r.

  - If v is linked as a child of another root, then new root's rank is either already greater than r or is equal to $r + 1$ after linking.

    When some other tree's root is linked as a child of v, either $v.rank$ remains same or it increases by one. In the former case, the root of x does not change, whereas in the latter, the root's rank is greater than r.

    That is, except for v, no root v' exists such that x is a descendent of v' and the rank of v' is r.

  Theorem 1: In executing $O(m)$ make-set, link, and find-set operations, in which n are make-set operations, for any non-negative integer r, there are at most $\frac{n}{2^r}$ nodes of rank r.

  - Suppose there are greater than $\frac{n}{2^r}$ nodes of rank r. Then, from Lemma 1 and Lemma 2, the total number of nodes in the disjoint-set forest is at least $(> \frac{n}{2^r})(\geq 2^r)$, which is strictly greater than n.

  Corollary: The rank of any node is at most $\lfloor \lg n \rfloor$.

  - Substituting $r' > \lfloor \lg n \rfloor$ in Theorem 1, the number of nodes of rank r' is strictly less than 1.

- The iterated logarithm function, $\lg^* n = \begin{cases} \min\{i \geq 0 : \lg^{(i)}(n) \leq 1\} & \text{if } n > 1, \\ 0 & \text{otherwise.} \end{cases}$

  This is a very slowly growing function after the inverse Ackermann function.

  For any non-negative integer r, r is said to be in *block*-i whenever $\lg^* r = i$. A node v is in block-i if the rank of v is in block-i. We say the block id of block-i is i. Since node ranks are integers in $[0, \lfloor \lg n \rfloor]$, block id's are integers in $[0, (\lg^* n) - 1]$.

- It is immediate, n make-set operations together take $O(n)$ time, and the $O(m)$ link operations together take $O(m)$ time. ——— (1a)

  The find-set is essentially a find-path together with path compression. Since the time for path compression can be charged to number of nodes visited in a find-path, the time complexity of a find-set operation is the number of nodes visited in the corresponding find-path. To analyze the amortized time complexity of all the find-paths among $O(m)$ operations, we categorize nodes along any find-path P:

  (i) root and its child on P (these are the nodes whose parents won't change due to a find-path),

  (ii) every node v on P whose parent belongs to a different block to v, and

  (iii) every node v on P whose parent belongs to the same block as v.

  Since there are $O(m)$ find-paths and each such path has at most two nodes of type-(i), the amortized cost of accessing all type-(i) nodes together is $O(m)$. ——— (1b)

  Since block ids are in $[0, (\lg^* n) - 1]$ and since nodes of ranks along any find-path increase, there are at most $\lg^* n$ nodes of type-(ii) along any find-path. Since there are $O(m)$ find-paths, the amortized cost of accessing all type-(ii) nodes together is $O(m \lg^* n)$. ——— (1c)

  From here on, we focus on upper bounding the total number of type-(iii) nodes visited due to $O(m)$ find-path operations.

- Once v is determined to be a type-(ii) node, then it continues to be a type-(ii) node in subsequent find-paths as well. This is due to v's parent's rank would either remains same or increases; in both the cases, v's parent is in a different block to v.

  Indeed, for a node v with its rank belonging to block-i, the worst case arises when the following two events occur alternately: a find-path on v and linking root of the tree in which v resides as a child of another root. Again, in the worst case, with each such find-path on v, v's parent's rank could increase. Since v's parent's ranks strictly increase, eventually, the rank of parent of v could belong to a block that is different from the block to which v belongs. From the definition of type-(ii) nodes, when this happens, v becomes a node of type-(ii).

  Since the number of type-(ii) nodes is upper bounded, it suffices to account for how many times any type-(iii) node v could be visited among $O(m)$ find-paths before v becomes a type-(ii) node.

- The number of type-(iii) nodes when all the $O(m)$ find-paths with n make-sets and $O(m)$ link operations considered equals to $\sum_{i=0}^{(\lg^* n)-1}$ (number of nodes whose ranks are in block-i) $*$ (for any node v in block-i, maximum number of times v's parent's rank is incremented by one while v's parent's rank continues to lie in block-i). ——— (2)

Let $minr_i$ be the minimum rank possible in block-i. Also, let $maxr_i$ be the maximum rank possible in block-i. From Theorem 1, the first term of (2) is at most $\frac{n}{2^{minr_i}} + \frac{n}{2^{minr_i+1}} + \ldots + \frac{n}{2^{maxr_i}} < \frac{n}{2^{minr_i-1}} = \frac{n}{maxr_i}$. The last equality is due to the following: since maximum rank possible in any block is a tower of 2s, $2^{maxr_{i-1}} = maxr_i$; however, $minr_i = maxr_{i-1} + 1$.

The second term of (2) is maximized if v has rank $minr_i$ and its parents' ranks increase amid find-paths in increments of one, from $minr_i + 1$ to $maxr_i$.

Hence, (2) is at most $\sum_{i=0}^{(\lg^* n)-1} (\frac{n}{maxr_i} * (maxr_i - (minr_i + 1) + 1)) = O(n \lg^* n)$.

- Combining (1a), (1b), (1c), with (2), the amortized time complexity of m make-set, union, find-set operations in which n are make-set operations is $O(m \lg^* n)$.

References:

Set Merging Algorithms. J. E. Hopcroft and J. D. Ullman. SIAM Journal on Computing, Vol. 2(4): 294-303, 1973. (This note only covers pages 7-8 of this paper.)

Appendix

make-set(x):
set x as x's parent
initialize $x.rank$ to 0

union(x', y'):
if $((x \leftarrow find\text{-}set(x'))! = (y \leftarrow find\text{-}set(y'))$ **then**
    $link(x, y)$

$link(x, y)$:
if $x.rank < y.rank$ **then**
    link x as a child of y
else
    link y as a child of x
    if $x.rank$ *is equal to* $y.rank$ **then**
        increase the rank of y by one

find-set(x):
foreach *node* x' *on the simple path from* x *to root* v **do**
    //visiting nodes along this path is called a *find-path* on x
    make x' as a child of v //called *path compression*
end