CIS 621: Algorithms and Complexity

Assignment 2

Igor Burago

Problem 1

When a starting state of the binary counter is not zero, initial credit value equals to \( b \), the number of bits set to one. As we showed in class, to increment the counter once, two credit units are necessary: Ones becoming zeros pay for themselves; one unit is needed to flip the rightmost zero bit, another unit is associated with that newly set bit.

Thus, it takes \( b + 2m \) credit units to increment the counter \( m \) times starting from a state with \( b \) ones. Since \( m \in \Omega(b) \) iff there exists \( c > 0 \) such that

\[
\begin{align*}
    cb & \leq m, \\
    b & \leq \frac{1}{c} m, \\
    b + 2m & \leq \left( \frac{1}{c} + 2 \right) m,
\end{align*}
\]

then \( (b + 2m) \in O(m) \).

Problem 2

Without loss of generality, we assume that the number of tasks performed is a power of two: \( n = 2^k \). Then the total complexity of a sequence of the tasks is

\[
T(n) = n - k + \sum_{i=1}^{k} 2^i = n - k + 2^{k+1} - 2 = 3n - k - 2, \quad k > 0.
\]

Since \( \frac{1}{n} T(n) = 3 - \frac{k+2}{n} < 3 \), the amortized complexity of a sequence of the tasks is \( \Theta(1) \).
Problem 3

Let us assume, without any loss of generality, that there are $n = 2^k$ queues to meld, each of which contains at most $m$ items.

The “round robin” melding algorithm then consists of $k$ rounds. During each $i$-th round, queues of at most $m^{2^{i-1}}$ items are merged in pairs, taking the number of operations proportional to $\log(m^{2^i})$. There are $2^{k-i}$ such merges within $i$-th round. This gives us the total time complexity proportional to

$$\sum_{i=1}^{k} 2^{k-i} \log_2(m^{2^i}) = \log_2 m \sum_{i=1}^{k} 2^{k-i} + \sum_{i=1}^{k} i2^{k-i}.$$  

Considering that

$$\sum_{i=1}^{k} 2^{k-i} = 2^k \sum_{i=1}^{k} 2^{-i} = 2^k (1 - 2^{-k}) = 2^k - 1,$$

$$\sum_{i=1}^{k} i2^{k-i} = -2^{k+1} \sum_{i=1}^{k} -i2^{-i-1}$$

we obtain $O(n \log m) \in O(nm)$ total time complexity. Since $nm$ is an upper bound on the number of items in all queues, the algorithm is linear in the number of items in total.

Problem 4

a) Let us construct a worst-case sequence of operations for this implementation of disjoint-set forest. In doing so, we will rely on the following propositions:
1) Each of the \textsc{Find-Depth} operations in a worst-case sequence $S^*$ is applied to the deepest node in the forest, because this implementation of \textsc{Find-Depth} is linear in depth of a node.

2) All calls to \textsc{Find-Depth} are the last operations in $S^*$. Indeed, the maximum height of trees in the forest cannot decrease, regardless of operations performed (in this implementation).

3) In order to achieve the maximum possible height of trees in the forest, after the first node has been created, \textsc{Make-Tree} and \textsc{Graft} are called in pairs, so that the resulting forest degenerates to a chain:

\[
\begin{align*}
\text{\textsc{Make-Tree}}(v_1), \\
\text{\textsc{Make-Tree}}(v_2), & \quad \text{\textsc{Graft}}(v_2, v_1), \\
\text{\textsc{Make-Tree}}(v_3), & \quad \text{\textsc{Graft}}(v_3, v_2), \\
\vdots & \quad \vdots \\
\text{\textsc{Make-Tree}}(v_i), & \quad \text{\textsc{Graft}}(v_i, v_{i-1}), \\
\vdots & \quad \vdots \\
\text{\textsc{Make-Tree}}(v_n), & \quad \text{\textsc{Graft}}(v_n, v_{n-1}).
\end{align*}
\]

This way, a worst-case sequence $S^*$ breaks down into:

1) $n$ calls \textsc{Make-Tree}(v$_i$),
2) $n - 1$ calls \textsc{Graft}(v$_i$, v$_{i-1}$),
3) $m - 2n + 1$ calls \textsc{Find-Depth}(v$_n$).

Taking into account time complexity of each of the three operations, we obtain that the total time complexity of $S^*$ is proportional to

$$T_m(n) = n \cdot 1 + (n - 1) \cdot 1 + (m - 2n + 1) \cdot n = -2n^2 + (m + 3)n - 1.$$

The value of $n$ giving the worst running time can be easily found from the necessary condition of extremum:

$$\frac{d}{dn} T_m(n) = 0,$$

$$-4n + m + 3 = 0.$$
Hence, the integer \( n^* \) nearest to \( \frac{m+3}{4} \) maximizes \( T_m(n) \).

\[
n^* = \left\lfloor \frac{m + 3}{4} + \frac{1}{2} \right\rfloor = \left\lfloor \frac{m + 5}{4} \right\rfloor.
\]

Since, as a polynomial of degree two, \( T_m(n) \in \Theta(n^2) \) and \( n^* \in \Theta(m) \), the total running time for a worst-case sequence \( S^* \) is \( T_m(n^*) \in \Theta(m^2) \).

b) The only difference of \texttt{MAKE-Tree} from \texttt{MAKE-Set} is initialization of the pseudo-distance for a new node:

\[
\text{MAKE-Tree}(v) \\quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \ quad
compress the path \( v \rightarrow p[v] \rightarrow \text{root} \) to \( v \rightarrow \text{root} \), the sum \( d[v] + d[p[v]] + d[\text{root}] \), previously giving the correct depth for \( v \), changes to \( d[v] + d[\text{root}] \). To make it equal to the actual depth of \( v \), and maintain the key property, we include \( d[v] \) by the pseudo-distance of its former parent.

Given \textsc{Find}, implementation of \textsc{Find-Depth} is trivial:

\[
\textsc{Find-Depth}(v) \\
\begin{align*}
    r &\leftarrow \text{Find}(v) \\
    \text{if } v = r \\
    \quad \text{then return } d[v] \\
    \text{return } d[v] + d[p[v]]
\end{align*}
\]

d) We implement the \textsc{Graft} procedure the following way.

\[
\textsc{Graft}(u, v) \\
\begin{align*}
    r_u &\leftarrow \text{Find}(u) \\
    r_v &\leftarrow \text{Find}(v) \\
    \delta &\leftarrow \text{Find-Depth}(v) + 1 \\
    \text{if } \text{rank}[r_u] \leq \text{rank}[r_v] \\
    \quad \text{then } p[r_u] &\leftarrow r_v \\
    \quad d[r_u] &\leftarrow d[r_u] + \delta - d[r_v] \\
    \quad \text{if } \text{rank}[r_u] = \text{rank}[r_v] \\
    \quad \quad \text{then } \text{rank}[r_v] &\leftarrow \text{rank}[r_v] + 1 \\
    \text{else } p[r_v] &\leftarrow r_u \\
    d[r_u] &\leftarrow d[r_u] + \delta \\
    d[r_v] &\leftarrow d[r_v] - d[r_u]
\end{align*}
\]

First of all, we find roots of the trees in the disjoint-set forest to which nodes \( u \) and \( v \) belong, denoting them as \( r_u \) and \( r_v \), correspondingly. Then we determine the value to which depths of \( u \) and all of its children in the original forest increase—the depth of \( v \) plus one (for the link from \( u \) to \( v \)).

After that, we union the two trees with roots \( r_u \) and \( r_v \) the same way it is done for disjoint-set unions, including the union-by-rank heuristic. In addition, we update pseudo-distances to maintain the key property of the disjoint-set forest. To prove correctness of the updates, it is sufficient to show that depths of the roots \( r_u \) and \( r_v \) are correct after union; correctness of depths
for their children (in the disjoint-set forest) follows from that. Indeed, sums of $d[w], d[p[w]], d[p[p[w]]], \ldots, d[r_u]$ or $d[r_v]$, or both, do not change during unions, except for the last one or two terms (depending on the union case). If they are updated correctly, the whole sum is correct.

Let $d_0[w]$ and $d_1[w]$ be the values of pseudo-distances of a node $w$ before and after union, correspondingly. Depending on ranks, the following two union cases are possible.

1) Rank of $r_u$ is smaller than or equal to rank of $r_v$. The node $r_u$ becomes a child of the node $r_v$. The depth of $r_u$ after union is composed of

$$d_1[r_u] + d_1[r_v] = d_0[r_u] + \delta - d_0[r_v] + d_0[r_v] = d_0[r_u] + \delta.$$  

As it should, the depth increases by $\delta$. The depth of $r_v$ does not change and equals to

$$d_1[r_v] = d_0[r_v].$$

2) Rank of $r_u$ is greater than rank of $r_v$. The node $r_v$ becomes a child of the node $r_u$. After update, the depth of $r_u$ increases to

$$d_1[r_u] = d_0[r_u] + \delta.$$

The depth of $r_v$ after union equals to

$$d_1[r_v] + d_1[r_u] = d_0[r_v] - d_1[r_u] + d_1[r_u] = d_0[r_v].$$

That is, it remains unchanged, as expected.

Summing up all of the above, our implementation of GRAFT updates pseudo-distances correctly.

e) Comparing time complexity of MAKE-TREE, FIND-DEPTH, GRAFT, and MAKE-SET, FIND-SET, UNION, correspondingly, we see that in each of these three pairs of procedures asymptotic running times differ only by a constant factor.

MAKE-TREE vs MAKE-SET:

Additional $d[v] \leftarrow 0$ step.
**Find-Depth vs Find-Set:**

Two additional constant-time steps

\[ w \leftarrow p[v] \quad \text{and} \quad d[v] \leftarrow d[v] + d[w] \]

on every level of the downward pass.

**Graft vs Union:**

At most three additional constant-time steps: one or two for updating \(d[r_u]\) and \(d[r_v]\), as well as

\[ \delta \leftarrow \text{Find-Depth}(v) + 1. \]

The latter is also a constant-time operation, because of the path compression: Just before this call to Find-Depth, we call Find with the same node as an argument.

Thus, the sequence of \(m\) operations, among which \(n\) are call to Make-Tree, takes \(\Theta(m \alpha(n))\) time in the worst case.