Cover page for
CSC263 Homework #3

(fill and attach this page to your homework)

SUBMITTED BY:

(1) Family Name: Kozakov

Given Name: Michael

Student Number: 999301018

(2) Family Name: __________

Given Name: __________

Student Number: __________

Graded Homework should be returned in Tutorial: _______(tutorial room number)

FOR HELP WITH YOUR HOMEWORK YOU MAY CONSULT ONLY THE INSTRUCTOR, THE TEACHING ASSISTANTS, YOUR HOMEWORK PARTNER (IF YOU HAVE ONE), YOUR TEXTBOOK AND YOUR CLASS NOTES. **You may not consult any other source**.

By virtue of submitting this homework I/we acknowledge that I am/we are aware of the policy on homework collaboration for this course.

Assignment 3

Michael Kozakov

Mar 10, 2013

1. a) Since no path compression is used, the FIND operation will not change the rank or height of the tree. We then only need to consider the n - 1 UNIONs.
   - PREDICATE: Let P(x) = after x UNION operations, rank(q) = height(Q)
   - SHOW: P(n-1)
   - BASE CASE: Show P(0)
     - Since in the beginning Q is a singleton set, it has only one node q, so $height(Q) = 0$. We're given that for a single node, $rank(q) = 0$. So P(0) holds, since $height(Q) = rank(q)$;
   - INDUCTION HYPOTHESIS: Assume P(1), P(2), ..., P(n-2)
   - INDUCTION STEP: Show P(n-1)
     - Let T1 with root t1 and T2 with root t2 be two trees resulting after n-2 UNION operations.
     - Let h_1 and h_2 be the respective heights of T1 and T2.
     - Then (by IH) $rank(t1)$ is h_1 and $rank(t2)$ is h_2.
     - WLOG, assume $h_1 \leq h_2$
     - We then perform a UNION between T1 and T2 and call the resulting tree Q.
     - Case 1: $h_1 < h_2$
       - We make $t1$ the child of $t2$.
       - The height of T2 and rank of t2 remain unchanged.
       - The root q of Q is $t2$.
       - $height(Q) = height(T2) = rank(t2) = rank(q)$
       - Therefore P(n-1) holds.
     - Case 2: $h_1 = h_2$
       - We make t1 the child of t2.
       - The height of the resulting tree Q is $h_1 + 1$ (the original height of T1 + the new root node).
       - The rank of t2 is incremented, and becomes $h_2 + 1$.
       - Since $h_1 = h_2$, $rank(q) = h_2 + 1 = h_1 + 1 = height(Q)$.
       - Therefore P(n-1) holds.
   - CONCLUSION: After performing n-1 UNION operations using union-by-rank on n distinct elements (each in a singleton set), we end up with a tree Q, such that height(Q) = rank(q).
     Since FIND does not affect the height or rank of the resulting tree, showing P(n-1) is sufficient to prove that for any tree Q formed during the execution of σ, height(Q) = rank(q)

b) As was shown in part a, for any tree Q formed during the execution of σ, rank(q) = height(Q). To prove that $rank(q) \leq \lfloor \log_2 n \rfloor$, we need to prove that for any Q, $height(Q) \leq \lfloor \log_2 n \rfloor$. Since no path-compression is used, only the UNION operations can affect the height of Q.

- PREDICATE: Let P(x) = for input of x singleton sets, x-1 UNION operations will result in a tree Q, such that $height(Q) = \lfloor \log_2 x \rfloor$
- SHOW: P(n), $n \geq 1$
- BASE CASES:
  - show P(1)
    - After $1 - 1 = 0$ UNION operations, the height of Q is 0.
    - Then $height(Q) = 0 \leq \lfloor \log_2 1 \rfloor$
    - Then P(0) holds.
  - show P(2)
    - After 2 - 1 = 1 UNION operations, the height of Q is 1.
    - Then $height(Q) = 1 \leq \lfloor \log_2 2 \rfloor$
    - Then P(1) holds.
- INDUCTION HYPOTHESIS: Assume P(3), P(4), ..., P(n-1)
- INDUCTION STEP: Show P(n)
  - Let T1 and T2 be two trees formed as a result of n-2 UNIONs.
  - WLOG, assume such that $size(T1) \leq size(T2)$
  - Let h_1 and h_2 be the respective heights of T1 and T2.
  - Then since $size(T1) < n$ and $size(T2) < n$, by IH

$$\begin{aligned} h1 &\leq \lfloor \log_2(size(T1)) \rfloor \\ h2 &\leq \lfloor \log_2(size(T2)) \rfloor \end{aligned}$$

  - Since $size(T1) \leq size(T2)$ $h_1 \leq h_2$
  - Case 1: $h_1 < h_2$
    - We make t1 the child of t2.
    - The height of the resulting tree Q is h_2.
    - $height(T2) = h_2 \leq \lfloor \log_2(size(T2)) \rfloor \leq \lfloor \log_2(size(T2) + size(T1)) \rfloor \leq \lfloor \log_2 n \rfloor$,
    - Therefore P(n) holds.
  - Case 2: $h_1 = h_2$
    - We make t1 the child of t2.
    - The height of the resulting tree Q is $height(Q) = h_1 + 1$ (the original height + the new root node).
    - Let s be size(T1)

- Then

$$\begin{aligned} h1 &\le \lfloor \log_2 s \rfloor \\ h1+1 &\le \lfloor \log_2 s \rfloor + 1 \\ &\le \lfloor \log_2 s \rfloor + \lfloor \log_2 2 \rfloor \\ &\le \lfloor \log_2 (2s) \rfloor \end{aligned}$$

- Since s is the size of the smaller tree, $2s \le size(T1) + size(T2) \le n$.
- Then,

$$\begin{aligned} h_1 + 1 &\le \lfloor \log_2 n \rfloor \\ height(Q) &\le \lfloor \log_2 n \rfloor \end{aligned}$$

- Therefore P(n) holds

- CONCLUSION: By induction, we show that after performing n-1 UNIONS on n singleton sets, for the resulting tree Q, $height(Q) \le \lfloor \log_2 n \rfloor$
  Since FIND does not affect the height or rank of the resulting tree, showing P(n-1) is sufficient to prove that for any tree Q formed during the execution of σ, $height(Q) = rank(q)$

c)
- The UNION operation performs exactly one comparison between the ranks of the roots of the two trees. Therefore performing n-1 UNIONs costs n-1 comparisons and is $\theta(n)$.
- The FIND x operation goes up the tree Q from node x to the root q. Therefore in the worst case, FIND has to perform exactly height(Q) comparisons.
- In the worst case, m FIND operations are performed after the n - 1 UNION operations, so the cost of performing m FINDs is $\theta(m \log_2 n)$, since we've shown that after n-1 UNIONS the height will be $\le \log_2 n$.
- Therefore the cost of performing σ is $\theta(n + m \log_2 n)$. Since $m >= n$,

$$\begin{aligned} n + m \log_2 n &\le m + m \log_2 n \\ &\le m \log_2 n + m \log_2 n \\ &\le 2 \times m \log_2 n \end{aligned}$$

- Then, the cost of performing σ is $\theta(m \log_2 n)$.

2. a)
- In the worst case, the pivot is the largest (or smallest) element.
- In a sequence of n elements that means that in the first round we perform n-1 comparisons, and then call quicksort on a list of size n-1.
- Therefore, for a sequence of size 7, the worst case total number of comparisons is

$$6 + 5 + 4 + 3 + 2 + 1 = 21$$

b) ∘ In the best case, the pivot is the median value in the sequence. That means that when we perform quicksort on a sequence of size n, we would perform n-1 comparisons, and then call quicksort on two sequences of respective sizes $\lfloor \frac{n-1}{2} \rfloor$ and $\lceil \frac{n-1}{2} \rceil$.

∘ Therefore, for a sequence of size 7, the best case total number of comparisons is

$$6 + 2 + 2 = 10$$

c) ∘ The sorted version of A is [2, 3, 4, 7, 8, 10, 11, 12, 15]. The index i of 4 is 3, the index j of 11 is 7.

∘ According to the formula shown in CLRS (pg. 183), the probability of comparing two elements located at i and j is $\frac{2}{j-i+1}$.

∘ Therefore the probability of comparing 4 and 7 is $\frac{2}{7-3+1} = 2/5$

d) i.

$$\begin{aligned}
E_1 &= 0 \\
E_2 &= 1 \\
E_3 &= \frac{1}{3}(2 + E_1 + E_1) + \frac{2}{3}(2 + E_2) \\
&= \frac{2}{3} + \frac{2}{3}(2 + 1) = 8/3 \\
E_4 &= \frac{2}{4}(3 + E_1 + E_2) + \frac{2}{4}(3 + E_3) \\
&= \frac{2}{4}(4) + \frac{2}{4}(3 + 8/3) = 29/6
\end{aligned}$$

ii.

$$\begin{aligned}
E_5 &= \frac{1}{5}(4 + E_2 + E_2) + \frac{2}{5}(4 + E_1 + E_3) + \frac{2}{5}(4 + E_4) \\
&= \frac{1}{5}(4 + 1 + 1) + \frac{2}{5}(4 + 0 + 8/3) + \frac{2}{5}(4 + 29/6) \\
&= 37/5 \\
E_5 - E_4 &= \frac{37}{5} - \frac{29}{6} \\
&= \frac{77}{30}
\end{aligned}$$

3.
- ASSUMPTION: We are conducting $R \bowtie S$, where R has attributes A and C, and S has attributes B and C.
- ASSUMPTION: For a specific row, the values in column A, B, and C are referred to as a', b' and c'.

- ASSUMPTION: Since $R \bowtie S$ is the same as $S \bowtie R$, WLOG assume that $size(R) \leq size(S)$
- ASSUMPTION: Appending a row to a relation is O(1)
- Let |R| be the size of R.
- Create an empty relation RESULT with attributes A, B and C.

a)
- Algorithm:
  - Construct an AVL tree T out of the elements of S in the following way:
    * The key of each node C' will be c', and an attached value will be a linked list
    * For each row of S, start performing an AVL INSERT operation with the node with key c' and and an attached linked list containing a single node b' .
    * If in the process of the insert you discover a node C'' with the same key (c'), append b' to the linked list attached to C'', and don't insert C' into the AVL.
  - Now for each of the rows in R perform an AVL FIND operation in T, looking for a node with the key c'. If the node was found, then for each value b' in the attached linked list, append a row to RESULT containing a', b' and c'.
- Analysis:
  - The worst case for the first part happens when no duplicate value was found throughout the traversal, and a regular AVL INSERT was performed, costing $\theta(\log N)$. Since we are conducting one INSERT for every one of N elements in S, creating an AVL tree will take $\theta(N \log N)$ steps.
  - Conducting |R| searches is going to cost |R| $\log N$, since each FIND takes $\theta(\log N)$ steps in the worst case.
  - For every one of the successful matches, we traverse the attached linked list and append a new row to RESULT. Since appending a new row costs $\theta(1)$, and the total number of rows in RESULT is k, we say constructing RESULT will take $\theta(k)$
  - In total the algorithm takes $\theta(k + N \log N + |R| \log N)$. Since $|R| \leq N$, we can say that the algorithm takes $\theta(k + 2N) = \theta(k + N)$ steps.

b) ASSUMPTION: Uniform distribution of elements (SUHA).
- Algorithm:
  - For each of the elements in S, insert the key c' with the linked list value containing a single value b' into a hash table. If a value with such a key already exists, simply append b' to the linked list at that key.
  - For each of the elements in R, search for key c' in the hash table, and if found, for each of the elements in the linked list value, append the row a', b', c' to RESULT.
- Analysis:

- In the worst case we will be performing N inserts into a hash table takes an *expected* $\theta(N)$ steps.
- Conducting |R| searches takes an *expected* $\theta(|R|)$.
- For every one of the successful matches, we need to append a new row. Assuming that appending to a relation costs $\theta(1)$, conducting k appends costs $\theta(k)$.
- In total the algorithm takes an expected $\theta(k + |R| + N)$. Since $|R| \leq N$, we can say that the algorithm takes an expected $\theta(k + 2N) = \theta(k + N)$ steps.