CS 373: Algorithms, Spring 2003
Homework 5 (due Thursday, April 17, 2003 at 23:59:59)

<table>
<thead>
<tr>
<th>Name:</th>
<th>Net ID:</th>
<th>Alias:</th>
<th>U G</th>
</tr>
</thead>
</table>

Name: 
Net ID: | Alias: | U G |

Name: 
Net ID: | Alias: | U G |

# | Score | Grader |
---|-------|--------|
1. |       |        |
2. |       |        |
3. |       |        |
4. |       |        |
5. |       |        |

Total |       |        |
"I have an idea," said George. "I take it you fellows are naturally anxious to improve your minds?"

I said, "We don’t want to become monstrosities. To a reasonable degree, yes, if it can be done without much expense and with little personal trouble."

– Three Men on the Bummel, Jerome K. Jerome

1 Required Problems

1. Hashing to Victory
   [20 Points]

In this question we will investigate the construction of hash table for a set $W$, where $W$ is static, provided in advance, and we care only for search operations.

(a) [2 Points] Let $U = \{1, \ldots, m\}$, and $p = m + 1$ is a prime. Let $W \subseteq U$, such that $n = |W|$, and $s$ an integer number larger than $n$. Let $g_k(x, s) = (kx \mod p) \mod s$.

Let $\beta(k, j, s) = \left| \{x \mid x \in W, g_k(x, s) = j\} \right|$. Prove that

$$\sum_{k=1}^{p-1} \sum_{j=1}^{s} \binom{\beta(k, j, s)}{2} < \frac{(p-1)n^2}{s}.$$

(b) [2 Points] Prove that there exists $k \in U$, such that

$$\sum_{j=1}^{s} \binom{\beta(k, j, s)}{2} < \frac{n^2}{s}.$$

(c) [2 Points] Prove that $\sum_{j=1}^{n} \beta(k, j, n) = |W| = n$.

(d) [3 Points] Prove that there exists a $k \in U$ such that $\sum_{j=1}^{n} (\beta(k, j, n))^2 < 3n$.

(e) [3 Points] Prove that there exists a $k' \in U$, such that the function $h(x) = (k'x \mod p) \mod n^2$ is one-to-one when restricted to $W$.

(f) [3 Points] Conclude, that one can construct a hash-table for $W$, of $O(n^2)$, such that there are no collisions, and a search operation can be performed in $O(1)$ time (note that the time here is worst case, also note that the construction time here is quite bad - ignore it).

(g) [3 Points] Using (d) and (f), conclude that one can build a two-level hash-table that uses $O(n)$ space, and perform a lookup operation in $O(1)$ time (worst case).

2. Find kth smallest number.
   [20 Points]

This question asks you to to design and analyze a randomized incremental algorithm to select the $k$th smallest element from a given set of $n$ elements (from a universe with a linear order).

In an incremental algorithm, the input consists of a sequence of elements $x_1, x_2, \ldots, x_n$. After any prefix $x_1, \ldots, x_{i-1}$ has been considered, the algorithm has computed the $k$th smallest element in $x_1, \ldots, x_{i-1}$ (which is undefined if $i \leq k$), or if appropriate, some other invariant
from which the $k$th smallest element could be determined. This invariant is updated as the next element $x_i$ is considered.

Any incremental algorithm can be randomized by first randomly permuting the input sequence, with each permutation equally likely.

(a) [5 Points] Describe an incremental algorithm for computing the $k$th smallest element.

(b) [5 Points] How many comparisons does your algorithm perform in the worst case?

(c) [10 Points] What is the expected number (over all permutations) of comparisons performed by the randomized version of your algorithm? (Hint: When considering $x_i$, what is the probability that $x_i$ is smaller than the $k$th smallest so far?) You should aim for a bound of at most $n + O(k \log(n/k))$. Revise (a) if necessary in order to achieve this.

3. ANOTHER LOWER BOUND
[20 Points]
Let $b_1 \leq b_2 \leq b_3 \leq \ldots \leq b_k$ be $k$ given sorted numbers, and let $A$ be a set of $n$ arbitrary numbers $A = \{a_1, \ldots, a_n\}$, such that $b_1 < a_i < b_k$, for $i = 1, \ldots, n$.

The rank $v = r(a_i)$ of $a_i$ is the index, such that $b_v < a_i < b_{v+1}$.

Prove, that in the comparison model, any algorithm that outputs the ranks $r(a_1), \ldots, r(a_n)$ must take $\Omega(n \log k)$ running time in the worst case.

4. ACKERMANN FUNCTION
[20 Points]
The Ackermann’s function $A_i(n)$ is defined as follows:

$$A_i(n) = \begin{cases} 
4 & \text{if } n = 1 \\
4n & \text{if } i = 1 \\
A_{i-1}(A_i(n-1)) & \text{otherwise}
\end{cases}$$

Here we define $A(x) = A_x(x)$. And we define $\alpha(n)$ as a pseudo-inverse function of $A(x)$. That is, $\alpha(n)$ is the least $x$ such that $n \leq A(x)$.

(a) [4 Points] Give a precise description of what are the functions: $A_2(n)$, $A_3(n)$, and $A_4(n)$.

(b) [4 Points] What is the number $A(4)$?

(c) [4 Points] Prove that $\lim_{n \to \infty} \frac{\alpha(n)}{\log^*(n)} = 0$.

(d) [4 Points] We define

$$\log^{**} n = \min \left\{ i \geq 1 \left| \log^* \ldots \log^* n \leq 2 \right. \right\} \text{ i times}$$

(i.e., how many times do you have to take $\log^*$ of a number before you get a number smaller than 2). Prove that $\lim_{n \to \infty} \frac{\sqrt{\alpha(n)}}{\log^{**}(n)} = 0$.

(e) [4 Points] Prove that $\log(\alpha(n)) \leq \alpha(\log^{**} n)$ for $n$ large enough.
5. **Divide-and-Conquer Multiplication**

[20 Points]

(a) **[7 Points]** Show how to multiply two linear polynomials $ax + b$ and $cx + d$ using only three multiplications. (Hint: One of the multiplications is $(a + b) \cdot (c + d)$.)

(b) **[7 Points]** Give two divide-and-conquer algorithms for multiplying two polynomials of degree-bound $n$ that run in time $\Theta(n \lg 3)$. The first algorithm should divide the input polynomial coefficients into a high half and a low half, and the second algorithm should divide them according to whether their index is odd or even.

(c) **[6 Points]** Show that two $n$-bit integers can be multiplied in $O(n \lg 3)$ steps, where each step operates on at most a constant number of 1-bit values.
2 Practice Problems

1. [10 Points]

   (a) [1 Points] With path compression and union by rank, during the lifetime of a Union-
       Find data-structure, how many elements would have rank equal to ⌊lg n − 5⌋, where
       there are n elements stored in the data-structure?

   (b) [1 Points] Same question, for rank ⌊(lg n)/2⌋.

   (c) [2 Points] Prove that in a set of n elements, a sequence of n consecutive FIND operations
       take O(n) time in total.

   (d) [1 Points] (Based on CLRS 21.3-2)
       Write a nonrecursive version of FIND with path compression.

   (e) [3 Points] Show that any sequence of m MAKESET, FIND, and UNION operations, where
       all the UNION operations appear before any of the FIND operations, takes only O(m)
       time if both path compression and union by rank are used.

   (f) [2 Points] What happens in the same situation if only the path compression is used?

2. [10 Points] Off-line Minimum

   (Based on CLRS 21-1)
   The off-line minimum problem asks us to maintain a dynamic set T of elements from the do-
   main \{1, 2, \ldots, n\} under the operations INSERT and EXTRACT-MIN. We are given a sequence
   S of n INSERT and m EXTRACT-MIN calls, where each key in \{1, 2, \ldots, n\} is inserted exactly
   once. We wish to determine which key is returned by each EXTRACT-MIN call. Specifically,
   we wish to fill in an array extracted[1 \ldots m], where for i = 1, 2, \ldots, m, extracted[i] is the key
   returned by the ith EXTRACT-MIN call. The problem is “off-line” in the sense that we are
   allowed to process the entire sequence S before determining any of the returned keys.

   (a) [2 Points]
       In the following instance of the off-line minimum problem, each INSERT is represented
       by a number and each EXTRACT-MIN is represented by the letter E:

       4, 8, E, 3, E, 9, 2, 6, E, E, 1, 7, E, 5.

       Fill in the correct values in the extracted array.

   (b) [4 Points]
       To develop an algorithm for this problem, we break the sequence S into homogeneous
       subsequences. That is, we represent S by
       \[I_1, E, I_2, E, I_3, \ldots, I_m, E, I_{m+1}\],
       where each E represents a single EXTRACT-MIN call and each \(I_j\) represents a (possibly
       empty) sequence of INSERT calls. For each subsequence \(I_j\), we initially place the keys
       inserted by these operations into a set \(K_j\), which is empty if \(I_j\) is empty. We then do
       the following.
Off-Line-Minimum(m,n)
1  for i ← 1 to n
2    do determine j such that i ∈ K_j
3    if j ≠ m + 1
4      then extracted[j] ← i
5    let l be the smallest value greater than j for which set K_l exists
6    K_l ← K_j ∪ K_l, destroying K_j
7  return extracted

Argue that the array extracted returned by Off-Line-Minimum is correct.

(c) [4 Points]
Describe how to implement Off-Line-Minimum efficiently with a disjoint-set data structure. Give a tight bound on the worst-case running time of your implementation.

3. [10 Points] Tarjan’s Off-Line Least-Common-Ancestors Algorithm
(Based on CLRS 21-3)
The least common ancestor of two nodes u and v in a rooted tree T is the node w that is an ancestor of both u and v and that has the greatest depth in T. In the off-line least-common-ancestors problem, we are given a rooted tree T and an arbitrary set P = \{\{u, v\}\} of unordered pairs of nodes in T, and we wish to determine the least common ancestor of each pair in P.

To solve the off-line least-common-ancestors problem, the following procedure performs a tree walk of T with the initial call LCA(root[T]). Each node is assumed to be colored WHITE prior to the walk.

LCA(u)
1  MakeSet(u)
2  ancestor[Find(u)] ← u
3  for each child v of u in T
4    do LCA(v)
5  Union(u, v)
6  ancestor[Find(u)] ← u
7  color[u] ← BLACK
8  for each node v such that \{u, v\} ∈ P
9    do if color[v] = BLACK
10       then print “The least common ancestor of” u “and” v “is”
11           ancestor[Find(v)]

(a) [2 Points] Argue that line 10 is executed exactly once for each pair \{u, v\} ∈ P.
(b) [2 Points] Argue that at the time of the call LCA(u), the number of sets in the disjoint-set data structure is equal to the depth of u in T.
(c) [3 Points] Prove that LCA correctly prints the least common ancestor of u and v for each pair \{u, v\} ∈ P.
(d) [3 Points] Analyze the running time of LCA, assuming that we use the implementation of the disjoint-set data structure with path compression and union by rank.