Computer Science B63 — February 15, 2007
Scarborough Campus — University of Toronto

Homework Assignment #3
Due: March 7, 2007, by 1:10 pm

1. *Please complete and attach (with a staple) an assignment cover page to the front of your assignment. You may work alone or with one other student. If you work in a group, write both your names on the cover sheet and submit only one copy of your homework.*
2. *If you do not know the answer to a question, and you write "I (We) do not know the answer to this question", you will receive 20% of the marks of that question. If you just leave a question blank with no such statement, you get 0 marks for that question.*
3. *Unless we explicitly state otherwise, you should justify your answers. Your paper will be marked based on the correctness and completeness of your answers, and the clarity, precision and conciseness of your presentation.*

Question 1. (15 marks)

Give an efficient algorithm to compute the size of the intersection of two sets A and B. More precisely, the input to your algorithm are two sets A and B, each containing n distinct integers, represented by arrays $A[1..n]$ and $B[1..n]$. Your algorithm must compute $|A \cap B|$.

Your algorithm should run in $\Theta(n)$ ***expected*** time, under some assumptions.

Note: Do **not** assume that the sets contain integers that are small or have a small range. In fact, they could be 256-bit integers, and the integers in the sets may range from 0 to $2^{256} - 1$. So, a solution based on a direct access table or on linear time sorting is **not** acceptable.

a. (10 marks) Describe your algorithm in clear English and give the corresponding pseudo-code.

b. (5 marks) Explain why your algorithm runs in **expected** $\Theta(n)$ time. Explicitly state every assumption that you need for your time complexity analysis.

Question 2. (20 marks)

Let T be a hash table with m slots using chaining in which we store 256-bit integers. An integer key k maps to a slot in T via the hash function $h(k) = k \bmod m$. Assume that the keys stored in T are uniformly distributed in the range 0 to $2^{256} - 1$ and that m is much smaller than 2^{256}.

If many elements have been inserted in the hash table, chains will become long and performance may become unacceptable, meaning we need to use a larger hash table. Building a brand new table would require rehashing all the stored keys, which could take a long time. Let us consider what happens if we choose to expand T by one slot. If we want most elements to stay where they are, what elements can we move to the new slot, and what new hash function can we use? For this question we consider the following technique.

Initially T contains m slots, numbered 0 through $m - 1$. When expanding the table we add one slot to the end of T (numbered m) and redistribute the elements in slot $T[0]$ by rehashing them according to the hash function $h'(k) = k \bmod 2m$. Let us call the resulting hash table T_1. Whenever we do an operation on T_1, to find the proper slot for a key k, we first determine whether $h(k) < 1$; if so, we map k into slot $h'(k)$, otherwise we map k into slot $h(k)$.

a. (6 marks) Suppose T had n elements, and we expand T into T_1 as described above. For each i, $0 \le i \le m$, what is the expected length of the chain at slot $T_1[i]$? Justify your answer.

We now generalize this process as follows. Let s be any integer $1 \le s \le m$. After s expansions, s slots have been added and the keys in slots 0 through $s - 1$ have been redistributed using $h'(k) = k \bmod 2m$. We call the resulting table T_s (note that T_s contains $m + s$ slots, numbered 0 through $m + s - 1$). Whenever we do an operation on T_s, to find the proper slot for a key k, we first determine whether $h(k) < s$; if so, we map k into slot $h'(k)$, otherwise we map k into slot $h(k)$. More precisely, we use the following hash function:

$$h_s(k) = \begin{cases} k \bmod 2m & \text{if } h(k) < s \\ k \bmod m & \text{if } h(k) \geq s \end{cases}$$

b. (6 marks) Prove that h_s always gives a value in the range 0 to $m + s - 1$.

c. (8 marks) Suppose T had n elements, and we expand T into T_s as described above. What is the expected number of comparisons performed when doing a SEARCH for a key that is *not* present in the hash table? Justify your answer.

Question 3. (15 marks)

Let $G = (V, E)$ be a connected undirected graph with n nodes $V = \{1, 2, \ldots, n\}$ and m edges. Let $e_1, e_2, \ldots, e_m$ be all the edges of G listed in some specific order. Suppose that we remove the edges from G one at a time, *in this order*. Initially the graph is connected and at the end of this process the graph is disconnected. Therefore, there is an edge e_i, $1 \leq i \leq m$, so that before removing e_i the graph was connected but after removing e_i the graph became disconnected. Give an algorithm that determines this edge e_i. Your algorithm should run in $O(m \log^* n)$ time. Explain why your algorithm has this time complexity. You may assume that G is given to the algorithm as a linked list of the edges appearing in the order $e_1, e_2, \ldots, e_m$.

Question 4. (20 marks) This question is a programming assignment. To see its description follow the link given in the "Assignments" section of the course web page.