Data Structures and Algorithms (EE 220): Homework 4

Submit first 3 problems to Ms. Spanner before 10am on Mar 31
Programing assignment is due on Apr 7

Posted 03/21/2003

Note: Please provide proof of correctness for the algorithms that you propose. Also provide the complexity of the proposed algorithm.

Problem 1: (10 pts) Give an efficient algorithm to implement Queue data structure (First In First Out) and Stack data structure (Last In First Out) using Priority Queues.

Problem 2: (10 pts) You have implemented certain functionality using Heaps. Recall that the standard implementation of Heap data structures supports following operations. (a) build_heap (b) insert (c) extract_min (d) decrease_key. For your application these standard operations are not sufficient. You need to add an operation called delete(x), which deletes an element with key=x if such element is present in the heap. If no element has key=x then it reports that the element with key value x is not present in the heap. Give $O(\log(n))$ implementation of the delete operation, where n is the maximum possible size of the heap.

Problem 3: (10 pts) You have given a list of n elements, where each element can take value either 0 or 1. The aim is to sort this list in the ascending order. The constraint here is that the additional memory available is only $\Theta(1)$. Give an $O(n)$ sorting algorithm that respects the memory constraint. (Partial credit will be given for the $O(n)$ algorithm that violates the memory constraint).

Programing Assignment: (50 pts) This programing assignment is designed to study the performance gain of the hashing over binary search trees using simulations. For this you need to implement binary search tree and the hash table. The problem setting is as follows.

Lets assume that the input data given to us is n integers between 1 and 10,000. The size of hash table is 1029. The hash value of an element $x \in \{1, 2, \ldots, 10000\}$

(denoted by $h(x)$) is given by the following formula.

$$h(x) = x \bmod 1029. \tag{1}$$

We use separate chaining for storing the data. Now, m integers in the range $\{1, 2, \ldots, 10000\}$ will be given and you have to search whether the given integer is present in the hash table. Let Σ be the sum of steps that you need to execute for a given set of m elements. Now, $\frac{\Sigma}{m}$ is the average complexity of the m searches.

The pseudo code for the described procedure is as follows.

For(k=1 to n)
{
    generate random integer uniformly from the range $\{1, 2, \ldots, 10000\}$;
    Store the element in the hash table at appropriate location;
}

$\Sigma = 0$;
for(k=1 to m)
{
    generate random integer uniformly from the range $\{1, 2, \ldots, 10000\}$;
    Check if the generated number is present in the hash table;
    Let σ_k be the number of steps to search k^{th} element, then
    $\Sigma = \Sigma + \sigma_k$
}

Average_complexity = $\frac{\Sigma}{m}$

Perform the experiment for various values of n, say in the increment of 100 starting from 1000 till 9000, and $m = 5000$. Plot the value of average complexity versus n.

Now, store the given n elements in the binary search tree and repeat the experiment. Plot the value of average complexity versus n for this data structure as well. Compare the average complexities in both the cased. Does hashing provide any advantage?

Contact PA in case of any doubt regarding random number generation or any other step.