15-451 Algorithms, Spring 2009

Homework # 3 **Due Date: Tuesday, February 24th, 2009**

Please hand in each problem on a separate sheet and put your **name** and **recitation** (time or letter) at the top of each sheet. You will be handing each problem into a separate box, and we will then give homeworks back in recitation.

Remember: written homeworks are to be done **individually**. Group work is only for the oral-presentation assignments.

Problems:

(25 pts) 1. **Hashing.** As discussed in class, the notion of *universal* hashing gives us guarantees that hold for *arbitrary* (i.e., worst-case) sets S, in expectation over our choice of hash function. In this problem, you will work out what some of these guarantees are.

(a) Describe an explicit universal hash function family from $U = \{0, 1, 2, 3, 4, 5, 6, 7\}$ to $\{0, 1\}$. Hint: you can do this with a set of 4 functions.

(b) Find all subsets $S \subset U$ of size 2, such that for S there is no perfect hash function from your proposed universal hash function family H. Explain.

(c) Let's generalize the result we obtained in (a) a little bit. Suppose that we are looking for a smallest size universal hash family from $U = \{0, 1, \ldots, 2^n - 1\}$ to $\{0, 1\}$. Find a lower bound on the size of such a hash family. Provide a proof or an explanation.
Hint: You may use the following fact: $\sum_{p=0}^{q} \binom{q}{p} = 2^q$.

(d) Let H be a universal family of hash functions from some universe U into a table of size m. Let $S \subseteq U$ be a set of m elements we wish to hash. Prove that if we choose h from H at random, the expected number of pairs (x, y) in S that collide is $\le \frac{m-1}{2}$.

(e) Prove that with probability at least 7/8, no bin gets more than $1 + 2\sqrt{2m}$ elements. Hint: use part (d).
To solve this question, you should use "Markov's inequality". Markov's inequality is a fancy name for a pretty obvious fact: if you have a non-negative random variable X with expectation $\mathbf{E}[X]$, then for any $k > 0$, $\mathbf{Pr}(X > k\mathbf{E}[X]) \le 1/k$. For instance, the chance that X is more that 100 times its expectation is at most 1/100. You can see that this has to be true just from the definition of "expectation".

(25 pts) 2. **Doing more with binary search trees.**
Let B be a binary search tree with distinct values, and let h be the height of B.

a. Write an algorithm **findFirstGE**(x) that finds the smallest value in B that is greater than or equal to x. If no such value exists in B, return ∞.
As an example, suppose B contains the values $1, 2, 3, 4, 5$. Then:

  - **findFirstGE**(2) returns 2,
  - **findFirstGE**(2.5) returns 3, and
  - **findFirstGE**(6) returns ∞

  What is the running time of your algorithm in terms of h?

b. Write an algorithm **getAllBetween**(x, y) that returns all values v in B, such that $x \leq v \leq y$, *in sorted order.*
   For example, with B containing $1, 2, 3, 4, 5$:
  - **getAllBetween**(2, 4) returns $[2, 3, 4]$,
  - **getAllBetween**(-10, 2) returns $[1, 2]$, and
  - **getAllBetween**(3, 4.5) returns $[3, 4]$

  What is the running time of your algorithm?

c. What if we just want to know the *number* of elements between x and y?
   For example, with B containing $1, 2, 3, 4, 5$:
  - **countAllBetween**(2, 4) returns 3,
  - **countAllBetween**(-10, 2) returns 2, and
  - **countAllBetween**(3, 4.5) returns 2

  Can you find a faster algorithm?
  Hint: Recall from recitation that one way to work out the balance factor of a tree is to store the height at each node. You may want to use a similar idea in this problem, i.e. consider storing an additional piece of information at each node to help your algorithm Be careful though, you need to show how to maintain this information during an **insert** into B, and also that this extra information does not cause **insert** to take more than $\Theta(h)$ time.

(25 pts) 3. **The list-update problem**

Suppose we have n data items $x_1, x_2, \ldots, x_n$ that we wish to store in a linked list in some order. Let's say the cost for performing a *lookup*(x) operation is \$1 if x is in the head of the list, \$2 if x is the second element in the list, and so on.

For instance, say there are 4 items and it turns out that we end up accessing x_1 3 times, x_2 5 times, x_3 once, and x_4 twice. Then, the cost of using the list (x_2, x_1, x_4, x_3) will \$21.

The *Move-to-Front* (MTF) strategy is the following algorithm for organizing the list if we don't know in advance how many times we will access each element. We begin with the elements in their initial order $(x_1, x_2, \ldots, x_n)$. Then, whenever we perform a *lookup*(x) operation, we move the item accessed to the front of the list. Let us say that performing the movement is free. For instance, if the first operation was *lookup*(x_3), then we pay \$3, and afterwards the list will look like $(x_3, x_1, x_2, x_4 \ldots)$.

i. Suppose $n = 4$ and we use MTF starting from the order (x_1, x_2, x_3, x_4). If we perform the following 4 operations:

$$lookup(x_4), lookup(x_2), lookup(x_4), lookup(x_2).$$

   What does the list look like in the end and what was the total cost?

ii. What is the best static list[1] in hindsight,i.e, what is the best static list assuming that the requests are known in advance? Give a proof for your solution.

iii. Prove that the total cost of the MTF algorithm on a sequence of m operations (think of m as much larger than n) is at most $2C_{static} + n^2$ where C_{static} is the cost of the best static list in hindsight for those m operations.
Hint: Assume that the best static list is $x_1, x_2, \ldots, x_n$. If $i < j$ but x_j is in front of x_i in the MTF list, let's say that x_j has "cut in line" in front of x_i. Now, imagine that each element x_i has a piggy bank with \$1 for everyone that is currently cutting in line in front of it.

iv. So far we have assumed that the cost of moving an element to the front of the list is zero. Lets remove this assumption. Lets say that the cost of moving an element from position j to the front of the list is $j - 1$. How does the bound on the cost of the MTF algorithm change in this model?
Hint: Go through the same analysis as in the previous model but now keeping track of the additional cost.

Note: one nice use of this is for *data compression*. You store each ascii character in a list in this way, and then when reading a string of text, for each character you output its index i in the list before moving the character to the front (this requires only $O(\log i)$ bits, which will be small if the item was close to the front of the list).

(25 pts) 4. **Tunneling Towns**

Imagine that you have n pairs of towns: (w_1, e_1), (w_2, e_2)... (w_n, e_n). A large mountain range divides all the pairs, with each e_i lying east of the range, and each w_i lying to the west.

You are given the order of the towns on the west of the range - from north to south - and the order of the towns on the east of the range. For example we may have the western order given as: w_1, w_3, w_2, w_4, w_5, and the eastern order as: e_2, e_1, e_4, e_3, e_5. This is illustrated in Figure 1.
Each pair of towns, (w_i, e_i), wishes to build a tunnel between its two members. Unfortunately, this is not always possible, since no pair of tunnels may cross. For example, in the above diagram, we could connect w_1 and e_1 via a tunnel, or we could connect w_2 and e_2, but not both, since those tunnels would cross. This is illustrated in Figure 2.

i. Let us define an optimal placement of tunnels for some set of towns T as any placement which maximizes the total number of pairs that are connected (without crossing any tunnels). Find an optimal placement of tunnels for the previous example (Figure 1).

ii. Using dynamic programming, give an algorithm that finds the longest path in a directed acyclic graph.

[1]By a static list we mean that the list is not allowed to change after every request.

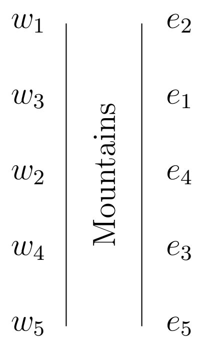
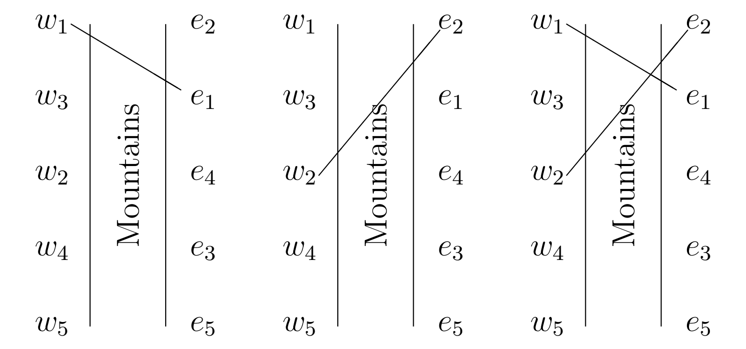


Figure 1: The towns, unconnected.



(a) Pair 1 connected. (b) Pair 2 connected. (c) Pairs 1 & 2 connected. Note that this is illegal, because the tunnels cross.

Figure 2: The towns with various connections.

iii. Give an algorithm that solves the tunnel placement problem in $O(n^2)$ time by creating a DAG and finding the longest path in it.