Introduction to Algorithms: 6.006
Massachusetts Institute of Technology September 23, 2010
Professors Konstantinos Daskalakis and Patrick Jaillet Handout 2

Problem Set 2

This problem set is divided into two parts: Part A problems are theory questions, and Part B problems are programming tasks.

Part A questions are due **Tuesday, October 5th** at **11:59PM**.

Part B questions are due **Thursday, October 7th** at **11:59PM**.

Solutions should be turned in through the course website. Your solution to Part A should be in PDF form using LaTeX or scanned handwritten solutions. Your solution to Part B should be a valid Python file, together with one PDF file containing your solutions to the two theoretical questions in Part B.

Templates for writing up solutions in LaTeX are available on the course website.

Remember, your goal is to communicate. Full credit will be given only to the correct solution which is described clearly. Convoluted and obtuse descriptions might receive low marks, even when they are correct. Also, aim for concise solutions, as it will save you time spent on write-ups, and also help you conceptualize the key idea of the problem.

Part A: Due Tuesday, October 5th

1. **(15 points)** Building a Balanced Search Tree from a Sorted List

   (a) **(5 points)** Given a list of n numbers, we can build a binary search tree containing these numbers by starting with an empty tree and inserting the numbers from the list one by one into the tree. By employing appropriate rebalancing procedures, e.g. as in AVL-trees, the total time needed to build this tree will be $O(n \log n)$.

   Now assume that the elements in this list of n numbers given to you are already sorted. Show how to construct in $O(n)$ time a binary search tree containing these numbers. The tree should be roughly balanced (its height should be $O(\log n)$)

   (b) **(5 points)** Argue that your algorithm returns a tree of height $O(\log n)$. Note: It is probably easier to prove an absolute bound (such as $1 + \log n$ or $2 \log n$) than to use asymptotic notation in the argument.

   (c) **(5 points)** Argue that the algorithm runs in $O(n)$ time (here it is hard to avoid the asymptotic notation, so use asymptotic notation).

2. **(20 points)** Implementing the Runway Reservation System

   (a) **(5 points)** Explain how to implement the runway reservation system such that all operations – checking whether a request to land at time t is valid, inserting a landing time into the system, and deleting a landing time from the system – take $O(\log n)$ time,

where n is the total number of scheduled landing times in the system before the operation. Remember, a requested time t is valid if there are no scheduled landings within < 3 minutes of t. Give algorithms for the three operations and analyze their running time.

(b) **(5 points)** Given two times t_1 and t_2 with $t_1 < t_2$, give an algorithm that returns all the scheduled landing times that are (inclusively) between t_1 and t_2. If the total number of such landing times is k, then the running time of your algorithm should be $O(k+\log n)$.

(c) **(5 points)** Given two times t_1 and t_2 with $t_1 < t_2$, give an algorithm to count the number of scheduled landing times that are (inclusively) between t_1 and t_2 in $O(\log n)$ time, independent of the number of such landing times. You are allowed to augment the binary search tree nodes.

(d) **(5 points)** Given a requested time t which is invalid, give an $O(\log n)$-time algorithm to find the smallest time t_2 such that $t_2 > t$ and t_2 is valid. You are allowed to augment the binary search tree nodes.

3. **(15 points)** Collision Resolution

Assume simple uniform hashing in the entire problem.

(a) **(5 points)** Consider a hash table with m slots that uses chaining for collision resolution. The table is initially empty. What is the probability that, after four keys are inserted, there is a chain of size exactly 3?

(b) **(5 points)** Consider a hash table with m slots that uses open addressing with linear probing. The table is initially empty. A key k_1 is inserted into the table, followed by key k_2, and then key k_3. What is the probability that the total number of probes while inserting these keys is at least 4?

(c) **(5 points)** Suppose you have a hash table where the load-factor α is related to the number n of elements in the table by the following formula:

$$\alpha = 1 - \frac{1}{\sqrt{n}}.$$

If you resolve collisions by open addressing, what is the expected time for an unsuccessful search in terms of n?

Part B: Due Thursday, October 7th

(50 points) Remote Error Correction

Prof. Daskalakis went to Mordor to give a talk. He was planning to work on Problem Set 3 (PS3) for 6.006 during his visit. Unfortunately, his computer broke and corrupted the latest copy of PS3.

A small fraction of the lines of the file was affected. Prof. Daskalakis could just download the latest copy of the file, but Mordor (The Land of Shady ISPs) is infamous for slow and pricey Internet access (2 magic rings per each transferred and received byte).

Help Prof. Daskalakis [redacted] recover the file and prepare PS3 on time! Design an interactive protocol for detecting and correcting corrupted characters that uses little communication. The file `file_transfer_skeleton.py` has three unimplemented functions. Your goal in this problem is to implement them. (Feel free to add helper methods.)

1. First, Prof. Daskalakis and Prof. Jaillet, who is in Cambridge, need a good hash function which would compute a hash value for any contiguous subset of lines of a file.

   (5 points) Implement a division hash in the function `hash_function`, which returns the hash value of a string.

   (20 points) The function `compute_node_values` should use your function `hash_function` to precompute all useful hash values for the old file and the new file. The hash values should be kept in `HashOldFile` and `HashNewFile`, respectively. In order to compute these values efficiently you should use the idea behind a rolling hash, i.e. instead of computing each value from scratch, exploit the fact that if you know the hash values of two blocks of lines, then with a constant number of operations you can compute the hash of the concatenation of these two blocks of lines.

2. **(4 points)** Suppose the two copies of the file differ in some number of consecutive lines. Let n be the total number of lines in the file. How can Prof. Daskalakis and Prof. Jaillet use binary search to find the start and end of the corruption by exchanging only $O(\log n)$ hash values?

   (4 points) Let W be the number of corrupted lines (not necessarily consecutive). What can they do to find all corrupted lines by exchanging only $O(W \cdot \log n)$ hash values?

   (17 points) The function `binary_check_hash` should simulate the protocol. It should call the function `compare` to compare two hash values, and the function `send_words` whenever a line is corrupted and has to be retrieved from the server.