Homework 5. Due May 29

CS180: Algorithms and Complexity
Spring 2018

GUIDELINES:

- Upload your assignments to Gradescope by 5:59 PM.
- Follow the instructions mentioned on the course webpage for uploading to Gradescope very carefully (including starting each problem on a new page and matching the pages with the assignments); this makes it easy and smooth for everyone. As the guidelines are simple enough, bad uploads will not be graded.
- You may use results and algorithms from class without proofs or too many details as long as you state what you are using clearly. You may have to wait till Wednesday's class for Problems 3, 4.
- Additional Guidelines for this assignment
  - For any universe U and your choice of a range size r, you have access to random hash functions $h : U \to \{1, \ldots, r\}$ as we discussed in class. For any $u \in U$, computing $h(u)$ takes $O(1)$ time and you don't have to take into account the space for computing h.
  - You can use the running-times of the dictionary data structures we discussed in class. Make sure you explicitly state what your U is.
- Most importantly, make sure you adhere to the policies for academic honesty set out on the course webpage. The policies will be enforced strictly. Homework is a stepping stone for exams; keep in mind that reasonable partial credit will be awarded and trying the problems will help you a lot for the exams.

1* Given two strings $X = x_1x_2 \ldots x_m$, $Y = y_1y_2 \ldots y_n$ of integers, a *common interlacing subsequence* is a sequence of indices $(i_1, i_2, \ldots, i_k)$ and $(j_1, j_2, \ldots, j_k)$, $1 \leq i_1 < i_2 < \cdots < i_k \leq m$ and $1 \leq j_1 < j_2 < \cdots < j_k \leq n$ such that $x_{i_1} < y_{j_1} < x_{i_2} < y_{j_2} < \cdots < x_{i_k} < y_{j_k}$.

Give a polynomial-time algorithm that given X, Y as inputs finds a longest common interlacing subsequence between X and Y. You don't have to prove correctness or analyze the time-complexity of the algorithm.

For instance, if $X = [1, 3, 2, 5]$ and $Y = [2, 6, 4, 7]$, then if you take $i_1 = 1, i_2 = 2, i_3 = 4$ and $j_1 = 1, j_2 = 3, j_3 = 4$, you get an interlacing subsequence—$1 < 2 < 3 < 4 < 5 < 7$—of length 6. In fact, in this case 6 is the best possible answer. [.75 points]

[Hint: One approach is to sub-problems like we did for edit distance (with one modification) and then try to develop a recurrence relation based on a case-analysis of what the last two symbols of the interlacing sequence would be.]

2 Consider the complete rooted *quaternary* tree of depth n. That is take a rooted tree where the root has four children, each child has exactly 4 children and so on for n levels. So for example for $n = 1$, you have the root connected to its four children (with 5 nodes in total); for $n = 2$ you have 21 nodes in total; and more generally, the total number of nodes in the tree of depth n is exactly $1 + 4 + 4^2 + \cdots + 4^n = (4^{n+1} - 1)/3$.

Now, suppose that you delete each edge of the tree independently with probability 1/2. Let X be the number of nodes which are still connected to the root after the deletion of the edges. Compute the expectation of X (with proof). [.75 points]

[Hint: One clean approach is to write X as a sum of 'simpler' random variables and use linearity of expectation to compute the expectation.]

3 Suppose you are given n apples and n oranges. The apples are all of different weights and all the oranges have different weight. However, for each apple there is a corresponding orange of the same weight and vice versa. You are also given a weighing machine that (counter to common intuition) will **only** compare apples to oranges: that is, if you feed an apple and an orange to the machine it will tell you whether the apple or the orange is heavier or if they have the same weight. Your goal is to use this machine to pair up each apple with the orange of the same weight with as few uses of the machine as possible.

Give a randomized algorithm to determining the pairing. For full-credit, your algorithm should only use the machine $O(n \log n)$ times in expectation. You do not need to prove correctness or analyze the time-complexity of the algorithm. [.75 points]

(Remember that you cannot compare an apple to an apple or an orange to an orange.)

4 Suppose you are writing a plagiarism detector. Students submit documents as part of a homework and each document is an (ordered) sequence of words. For some parameter m decided by the provost, we say two documents are copies of each other if one of them uses a sequences of m words (in that given order) from the other. Give an algorithm which given an integer m and N documents $D_1, \ldots, D_N$ as input, flags all submissions which are copies of some other submission. Your algorithm should run in expected $O(m + N +$ total-length of documents) time (i.e., expected $O(m + N + n_1 + \ldots + n_N)$ time where n_j is the length of j'th document) and be always correct. [.75 points]

Additional Problems. DO NOT turn in answers for the following problems - they are meant for your curiosity and understanding.

1. There are four types of brackets: (,), <, and >. We define what it means for a string made up of these four characters to be *well-nested* in the following way:
   (a) The empty string is well-nested.
   (b) If A is well-nested, then so are <A> and (A).
   (c) If S, T are both well-nested, then so is their concatenation ST.

For example, (), < () >, ((< >)), () <> are all well-nested. Meanwhile, (, <>),)(, (<)>, and (<(> are not well-nested.

Devise an algorithm that takes as input a string $s = (s_1, s_2, ..., s_n)$ of length n made up of these four types of characters. The output should be the length of the shortest well-nested string that contains s as a subsequence. For example, if (<(> is the input, then the answer is 6; a shortest string containing s as a subsequence is ()<()>. For full-credit, your algorithm should run in time $O(n^3)$ but you don't have to prove its correctness or analyze the time complexity.

(Hint: For this problem, try to use sub-problems that solve the problem for substrings of the original s: $(s_i, s_{i+1}, \ldots, s_j)$ for all $i < j$.)

2. Call a sequence of coin tosses "monotone" if the sequence never changes from Heads to Tails when parsed left to right. For example, the sequences $TTTHHHH$, $TTTT$ are monotone whereas $TTTHHHT$ is not. Consider a sequence of n coin tosses of a fair coin. For integer $k > 0$, let Y_k be the number of monotone sub-sequences of length k in the n coin tosses. Compute the expectation of Y_k (with proof).

   Recall that a sequence x is a sub-sequence of y if x can be obtained from y by deleting some symbols of y without changing the order of the remaining elements.

3. Define a random graph $G = (V, E)$ as follows. The vertex set of G is $V = \{1, \ldots, n\}$. Now for each pair $\{i, j\}$ with $i \neq j \in [n]$, add the pair $\{i, j\}$ to E with probability $1/2$ independent of the choice for every other pair. Let X_5 be the number of independent sets of size 5 in the graph G. Compute the expectation of X_5 (with proof).

   Recall that an independent set is a collection of vertices I in G such that no two vertices in I have an edge between them.