CSOR W4231.001 – Spring, 2020

Homework 3 (140 points)

Out: Monday, February 24, 2020
Due: 11:59pm, Monday, March 9, 2020

You should always describe your algorithm clearly in English. You should always give pseudocode for all algorithms that you design.

For all deterministic algorithms you design, you must argue correctness and give the best upper bound that you can for the running time. You are encouraged to analyze the space required by your algorithm.

If you give a DP algorithm, you should *also* clearly define the subproblems, give the recurrence and the boundary conditions, analyze time *and* space, and explain the order to compute subproblems. Proof of correctness is recommended but optional.

You should not use any external resources while working on this homework, not even your textbook. Failure to follow these instructions may affect your performance in exams (and possibly in interviews). For the same reason, you should think through the problems on your own extensively before you collaborate with your classmates. I also encourage you to work on all the recommended exercises.

You must submit your assignment as a pdf file. Other file formats will not be graded, and will automatically receive a score of 0. I recommend you type your solutions using LaTeX: you will earn 5 extra credit points if you type your solutions. If you do not type your solutions, make sure that your hand-writing is very neat, your scan is high-quality and your name and UNI are clearly written on your homework.

You should write up the solutions **entirely on your own**. Collaboration is limited to discussion of ideas only (see also course syllabus). Similarity between your solutions and solutions of your classmates, or solutions posted online will result in receiving a 0 in this assignment and possibly further disciplinary actions.

1. (20 points) You want to determine the probability of obtaining exactly k heads when n biased coins are tossed independently, where the i-th coin has known probability p_i of coming up heads.

   Given n, k, and $\{p_1, \ldots, p_n\}$, design and analyze an algorithm for this task that runs in time $O(nk)$. (Assume that you can multiply and add two numbers in $[0, 1]$ in $O(1)$ time.)

2. (20 points) You are given a string of n characters $s[1, \ldots, n]$, which you believe to be a corrupted text document in which all punctuation has vanished (e.g., "itwasthebestoftimes"). You wish to reconstruct the document using a dictionary available in the form of a Boolean function `dict(·)`: for any string w,

$$\texttt{dict}(w) = \begin{cases} 1, & \text{if } w \text{ is a valid word} \\ 0, & \text{otherwise} \end{cases}$$

   Give an $O(n^2)$ dynamic programming algorithm that determines if the string $s[\cdot]$ can be reconstituted as a sequence of valid words (assume calls to `dict` take constant time). You should also output the corresponding sequence of words, if the string is valid.

3. (26 points) There are n tasks to complete. Each task first needs to be *preprocessed* on a supercomputer and then finished on one of n processors. Task i requires p_i seconds of preprocessing on the supercomputer followed by f_i seconds on some processor to complete.

   Note that tasks are fed to the supercomputer sequentially but can be executed in parallel on different processors once they have been preprocessed. The process *terminates* when all tasks have been completed. The *duration* of the process is the total time until termination.

   Give an efficient algorithm that determines an ordering of the tasks for the supercomputer that minimizes the duration of the process, as well as the minimum duration.

4. (34 points) Given integers $a_1, \ldots, a_n$, we want to determine whether it is possible to partition $\{1, \ldots, n\}$ into three disjoint subsets I, J, K such that

$$\sum_{i \in I} a_i = \sum_{j \in J} a_j = \sum_{k \in K} a_k = \frac{1}{3} \sum_{i=1}^{n} a_i$$

   For example, for input $\{1, 2, 3, 4, 4, 5, 8\}$ the answer is **yes**, because there is the partition $(1, 8), (4, 5), (2, 3, 4)$. On the other hand, for input $\{2, 2, 3, 5\}$ the answer is **no**.

   Design and analyze a dynamic programming algorithm for this problem that runs in time polynomial in n and $\sum_{i=1}^{n} a_i$.

5. (40 points) You are given some data to analyze. You can spend D dollars to perform the analysis. You have organized the process of analyzing the data so that it consists of n tasks that have to be performed sequentially by using dedicated hardware: you will use a processor P_i to perform task i, for every i. Each processor is relatively cheap but may fail to complete its task with some probability, independently of the other processors. Specifically, P_i costs c_i dollars and succeeds to complete its task with probability s_i, while it fails with probability $1 - s_i$.

   (a) (3 points) What is the probability that the process of analyzing the data will be completed successfully?

   (b) Note that you can improve this success probability by using p_i identical processors P_i for task i instead of just one.

      i. (8 points) What is the probability that task i will be completed successfully now?

      ii. (4 points) What is the probability that the process of analyzing the data will be completed successfully?

      iii. (25 points) Given $s_1, \ldots, s_n$, integers $c_1, \ldots, c_n$ and integer D, give an algorithm to compute $p_1, \ldots, p_n$ such that the success probability of the entire process is maximized while you do not spend more than D dollars. Your algorithm should run in time polynomial in n and D.

RECOMMENDED exercises: *do NOT return, they will not be graded.*

1. Problem 16.1 from your textbook (pp. 446-447).

2. A server has n customers waiting to be served. The service time for customer i is t_i minutes. So if the customers are served in order of increasing i, the i-th customer spends $\sum_{j=1}^{i} t_j$ minutes waiting to be served.

   Given n, $\{t_1, t_2, \ldots, t_n\}$, design an efficient algorithm to compute the optimal order in which to process the customers so that the total waiting time below is minimized:

   $$T = \sum_{i=1}^{n} \text{(time spent waiting by customer } i\text{)}$$

3. Exercise 15.4-5 from your textbook (p. 397).

4. Consider an array A with n numbers, some of which (but not all) may be negative. We wish to find indices i and j such that

   $$\sum_{k=i}^{j} A[k]$$

   is maximized. Give an efficient algorithm for this problem.

5. Given two strings $x = x_1 x_2 \cdots x_m$ and $y = y_1 y_2 \cdots y_n$, we wish to find the length of their longest common substring, that is, the largest k for which there are indices i and j such that

   $$x_i x_{i+1} \cdots x_{i+k-1} = y_j y_{j+1} \cdots y_{j+k-1}.$$

   Give an efficient algorithm for this problem.