CS 341, Fall 2025 T. Brown, É. Schost

ASSIGNMENT 2

DUE: Wed Oct 8, 11:59 PM. DO NOT COPY. ACKNOWLEDGE YOUR SOURCES.

Please read `http://www.student.cs.uwaterloo.ca/~cs341` for general instructions and policies.

Note: All logarithms default to base 2 (i.e., $\log x$ is defined as $\log_2 x$). **Note:** "Giving" an algorithm means doing the four parts (i)–(iv) as described on the course web page.

1. [10 marks] **Greedy / Divide & Conquer.**

   Suppose there are n valuable gemstones embedded in the ground, arranged in a line. The gemstones have positive integer values $v_1, v_2, ..., v_n$. You can *extract* gemstones from the ground using your pickaxe, but in the process of extracting gemstone v_i, you will *destroy* gemstones v_{i-1} and v_{i+1}. The exceptions are v_1 and v_n. Extracting v_1 will only destroy one gemstone, v_2, and extracting v_n will only destroy v_{n-1}. The goal is to extract gemstones in such a way as to **maximize the total value** you obtain.

   Prove that the following greedy algorithms for this problem are **incorrect**.

   (a) [1 mark] Consider gems in the input order, and take every gemstone that you can. (In other words, at each step, if the gemstone being considered is not already destroyed, take it, destroying its neighbours.)

   (b) [1 mark] Sort from most valuable to least valuable (i.e., such that $v_1 \geq v_2 \geq ... \geq v_n$) then take every gemstone you can in that order. (Assume we also keep track of where the gems were in the input order, i.e., before the sort, so we know which gems are destroyed when we extract a gem.)

   (c) [1 mark] Do preprocessing to determine the sums S_{odd} of all odd v_i, i.e., $v_1, v_3, ...$, and S_{even} of all even v_i, i.e., $v_2, v_4, ...$ If S_{odd} is larger, extract all of the odd gems. Otherwise extract the even ones.

   (d) [1 mark] For each gem v_i, $1 < i < n$, let $f_i = \frac{v_i}{v_{i-1}+v_{i+1}}$, and let $f_1 = \frac{v_1}{v_2}$ and $f_n = \frac{v_n}{v_{n-1}}$. Sort from largest to smallest f_i (breaking ties arbitrarily), remembering where each gem was in the input order. Then, take every gem you can (i.e., that is not destroyed) in that order.

   (e) [1 mark] For each gemstone v_i, $1 < i < n$, let $f_i = v_i - v_{i-1} - v_{i+1}$ (i.e., the value you would get from the gem, minus the values of the gems that would be destroyed). For v_1 and v_n, let $f_1 = v_1 - v_2$ and $f_n = v_n - v_{n-1}$, respectively. Sort from largest to smallest f_i (breaking ties arbitrarily), then take every gemstone you can (i.e., that is not destroyed by another gem you extract) in that order.

   (f) [5 marks] Design a divide & conquer algorithm that returns the maximum value that can be achieved by extracting gems. Briefly justify correctness. Give a Θ bound for your algorithm's runtime in the unit cost model. To make the analysis a easier, you can ignore floors/ceilings, or differences of +/- 1 or 2 between subproblem sizes.

2. [10 marks] **Greedy.**

We are given n jobs (we'll just refer to them by their indices, from 1 to n); each one takes one unit of time to complete. We are also given deadlines $d_1, \ldots, d_n$ (positive integers) and rewards $r_1, \ldots, r_n$. We'll say that a set of jobs $F \subseteq \{1, \ldots, n\}$ is satisfiable if we can find an ordering of it for which all its jobs are done before, or at the time of, their respective deadlines. If that's the case, the total reward of F is $R(F) = \sum_{j \in F} r_j$.

Your task will be to design a greedy algorithm that finds a satisfiable set F that maximizes $R(F)$; the first questions will possibly help you prove correctness.

(a) [1 mark] For a set of jobs $F \subseteq \{1, \ldots, n\}$, and $t \in \{0, \ldots, n\}$, we define the counting function $c(F, t)$ as the number of jobs in F with deadline at most t. Prove that F is satisfiable if and only if $c(F, t) \leq t$ for $t = 0, \ldots, n$.

(b) [3 marks] Let G and F be satisfiable sets, with F not included in G, and suppose that j is a job in G, but not of F. Prove that there is a job i in F, but not in G, such that $F \cup \{j\} - \{i\}$ is satisfiable. Suggestion: use the previous question.

(c) [3 marks] Give a greedy algorithm that finds a satisfiable set G that maximizes the total reward. For this question, a description in words and a correctness proof are sufficient; don't worry about implementation details yet.

Suggestion: prove that if G is the greedy output and F any other satisfiable set, you can build F' satisfiable with $R(F) \leq R(F')$ and $|G \cap F'| > |G \cap F|$. First, though: assuming that this suggested claim is valid, make sure that you can explain why this would prove that the greedy output is optimal (if you do this last step correctly, even without proving the suggested claim, you will get partial credit).

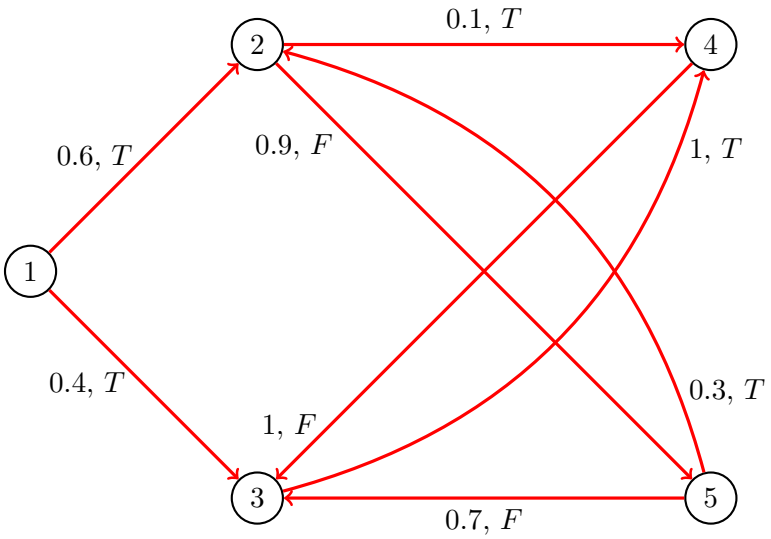
(d) [3 marks] Explain how to implement the strategy from (c): tell us what data structure(s) you use and how you will use them to perform the required operations. Give a big-O bound for the resulting runtime; $O(n^2)$ is enough for full marks on this question.

3. [10 marks] **Dynamic Programming.**

We consider a directed graph $G = (V, E)$, where edges e are labeled by both a *probability* $p[e]$ and a character $c[e]$. For any vertex v, the sum of all $p[e]$, for $e = (v, w)$, is equal to 1.

We can build a string S by starting from an arbitrary vertex s, and repeating L times the following: from the current vertex v, choose an edge $e = (v, w)$ at random (using the probability distribution given by the probabilities $p[e]$, for e outgoing edge at v), append its character $c[e]$ to S, and repeat from w.

The goal here is to somehow reverse the process: you are given the string S, and your task is to determine what was the likeliest walk we followed to produce this string: you should return the probability of this walk (which is the product of the probabilities of its edges) and the walk itself. For example, consider the following graph, with edges labeled with their probabilities and characters T or F (at $v = 1$, both outgoing edges have the same label; this is fine).



Starting from $v = 1$, there are two ways we can obtain the string $S = TT$, through the walks 124 and 134. The former has probability $0.6 \times 0.1 = 0.06$, the latter $0.4 \times 1 = 0.4$. There is only one other walk labeled $S = TT$, namely 524, with probability $0.3 \times 0.1 = 0.03$. So in this case, the output of the algorithm should be the walk 134, with its probability 0.4.

Describe an algorithm that performs this task and analyze its runtime in terms of n (number of vertices), m (number of edges) and L (length of the string S). You can assume that all operations you do use arguments that fit in a word (this includes floating-point arithmetic, to manipulate the probabilities) and you can do the analysis in the unit cost model. You can also assume that the vertex set is $V = \{1, \ldots, n\}$, and that you are given the graph as an array $A = A[1..n]$, with each $A[i]$ being the list of all j such that (i, j) is an edge.