CS 170 HW 6

Due on 2018-10-07, at 9:59 pm

1 (★) Study Group

List the names and SIDs of the members in your study group.

2 (★★★) Horn Formulas

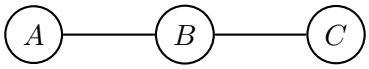
Describe an algorithm which finds a satisfying assignment for a Horn formula, or reports that none exists, in time linear in the size of the formula. Prove that your algorithm is correct and runs in linear time.

3 (★★) Graph Game

Given an undirected, unweighted graph G, with each node v having a value $\ell(v) \geq 0$, consider the following game.

1. All nodes are initially *unmarked* and your score is 0.
2. Choose an unmarked node u. Let $M(u)$ be the *marked* neighbours of u. Add $\sum_{v \in M(u)} \ell(v)$ to your score. Then mark u.
3. Repeat the last step for as many turns as you like, or until all the nodes are marked.

For instance, suppose we had the graph:



with $\ell(A) = 3$, $\ell(B) = 2$, $\ell(C) = 3$. Then, an optimal strategy is to mark A then C then B giving you a score of $0 + 0 + 6$. We can check that no other order will give us a better score.

(a) Is it ever better to leave a node unmarked? Briefly justify your answer.

(b) Give a greedy algorithm to find the order which gives the best score. Describe your algorithm, prove its optimality, and analyse its running time.

(c) Now suppose that $\ell(v)$ can be negative. Give an example where your algorithm fails.

(d) Your friend suggests the following modified algorithm: delete all v with $\ell(v) < 0$, then run your greedy algorithm on the resulting graph. Give an example where this algorithm fails.

4 (★★) Tree Perfect Matching

A *perfect matching* in an undirected graph $G = (V, E)$ is a set of edges $E' \subseteq E$ such that for every vertex $v \in V$, there is exactly one edge in E' which is incident to v.

Give an algorithm which finds a perfect matching *in a tree*, or reports that no such matching exists. Describe your algorithm, prove that it is correct and analyse its running time.

5 (★★) Steel Beams

You're a construction engineer tasked with building a new transit center for a large city. The design for the center calls for a T-foot-long steel beam for integer $T > 0$. Your supplier can provide you with an *unlimited* number of steel beams of integer lengths $0 < c_1 < \ldots < c_k$ feet. You can weld as many beams as you like together; if you weld together an a-foot beam and a b-foot beam you'll have an $(a + b)$-foot beam. Unfortunately, every weld increases the chance that the beam might break, so you want as few as possible.

Your task is to design an algorithm which outputs how many beams of each length you need to obtain a T-foot beam with the minimum number of welds, or 'not possible' if there's no way to make a T-foot beam from the lengths you're given. (If there are multiple optimal solutions, your algorithm may return any of them.)

(a) Consider the following greedy strategy. Start with zero beams of each type. While the total length of all the beams you have is less than T, add the longest beam you can without the total length going over T.

  (i) Suppose that we have 1-foot, 2-foot and 5-foot beams. Show that the greedy strategy always finds the optimum.

  (ii) Find a (short) list of beam sizes $c_1, \ldots, c_k$ and target T such that the greedy strategy fails to find the optimum. Briefly justify your choice.

(b) Give a dynamic programming algorithm which always finds the optimum. Describe your algorithm, including a clear statement of your recurrence, show that it is correct and prove its running time. How much space does your algorithm use?

6 (★) Longest Increasing Subsequence

Given an array A of n integers, here is a linear time algorithm to find the length of the longest **strictly** increasing subsequence, where $M[j]$ represents the length of the longest increasing subsequence of the first j integers of A.

$$M[1] = 1$$

$$M[i] = \begin{cases} M[i-1] & \text{if } A[i] \leq A[i-1] \\ M[i-1] + 1 & \text{otherwise} \end{cases}$$

Verify that this algorithm takes linear time. Is the algorithm correct? Prove it is or give a counter-example and explain.

7 (★★★★) Propositional Parentheses

You are given a propositional logic formula using only $\wedge$, $\vee$, T, and F that does not have parentheses. You want to find out how many different ways there are to *correctly parenthesize* the formula so that the resulting formula evaluates to true.

A formula A is correctly parenthesized if $A = T$, $A = F$, or $A = (B \wedge C)$ or $A = (B \vee C)$ where B, C are correctly parenthesized formulas. For example, the formula $T \vee F \vee T \wedge F$ can be correctly parenthesized in 5 ways:

$$(T \vee (F \vee (T \wedge F))) \quad (T \vee ((F \vee T) \wedge F)) \quad ((T \vee F) \vee (T \wedge F))$$
$$(((T \vee F) \vee T) \wedge F) \quad ((T \vee (F \vee T)) \wedge F)$$

of which 3 evaluate to true: $((T \vee F) \vee (T \wedge F))$, $(T \vee ((F \vee T) \wedge F))$, and $(T \vee (F \vee (T \wedge F)))$.

(a) Give a dynamic programming algorithm to solve this problem. Describe your algorithm, including a clear statement of your recurrence, show that it is correct, and prove its running time.

(b) Briefly explain how you could use your algorithm to find the probability that, under a *uniformly randomly chosen* correct parenthesization, the formula evaluates to true.