CS 170 HW 6

Due **2020-10-12, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer "Yes", "Yes but anonymously", or "No"

2 Maximum Coverage

In the maximum coverage problem, we have m subsets of the set $\{1, 2, \ldots n\}$, denoted $S_1, S_2, \ldots S_m$. We are given an integer k, and we want to choose k sets whose union is as large as possible.

Give an efficient algorithm that finds k sets whose union has size at least $(1 - 1/e) \cdot OPT$, where OPT is the maximum number of elements in the union of any k sets. In other words, $OPT = \max_{i_1, i_2, \ldots i_k} |\cup_{j=1}^{k} S_{i_j}|$. Just the algorithm description and justification for the lower bound on the number of elements your solution contains is needed.

(Recall the set cover algorithm from lecture, and use that $(1 - 1/n)^n \leq 1/e$ for all integers n)

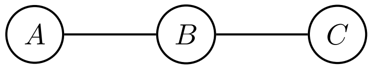
3 Graph Game

This is a solo question.

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 the following statement true or false? **For any graph, an optimal strategy which marks all the vertices always exists.** Briefly justify your answer.

(b) Give a greedy algorithm to find the order which gives the best score. **Please give a 3-part solution for this part.**

(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.

DP Solution Guidelines

Try to follow the following 3-part template when writing your solutions.

- Define a function $f(\cdot)$ in words, including how many parameters are and what they mean, and tell us what inputs you feed into f to get the answer to your problem.
- Write the "base cases" along with a recurrence relation for f.
- Prove that the recurrence correctly solves the problem.
- Analyze the runtime of your final DP algorithm.

4 Maximum weight independent set

This is a solo question.

Let $G = (V, E)$ be an undirected graph, with nonnegative weights $w(v)$ for each vertex $v \in V$. A subset of nodes $S \subset V$ is an *independent set* of G if no edge has both its endpoints in S.

Assuming that G is a tree, find an efficient dynamic programming algorithm for finding the maximum weight independent set in G, i.e. an independent set S of G such that $\sum_{v \in S} w(v)$ is maximized.

Please provide a 3-part solution.

5 Egg Drop

This is a solo question.

You are given k identical eggs and an n story building. You need to figure out the highest floor $\ell \in \{0, 1, 2, \ldots n\}$ that you can drop an egg from without breaking it. Each egg will never break when dropped from floor ℓ or lower, and always breaks if dropped from floor $\ell + 1$ or higher. ($\ell = 0$ means the egg always breaks). Once an egg breaks, you cannot use it any more.

Let $f(n, k)$ be the minimum number of egg drops that are needed to find ℓ (regardless of the value of ℓ).

(a) Find $f(1, k)$, $f(0, k)$, $f(n, 1)$, and $f(n, 0)$.

(b) Find a recurrence relation for $f(n, k)$. *Hint: Whenever you drop an egg, call whichever of the egg breaking/not breaking leads to more drops the "worst-case event". Since we need to find ℓ regardless of its value, you should assume the worst-case event always happens.*

6 Motel Choosing

You are traveling along a long road, and you start at location $r_0 = 0$. Along this road, there are n motels at location $\{r_i\}_{i=1}^n$ with $0 < r_1 < r_2 < \cdots < r_n$. The only places you may stop are these motels, but you can choose which to stop at. You must stop at the final motel (at distance r_n), which is your destination.

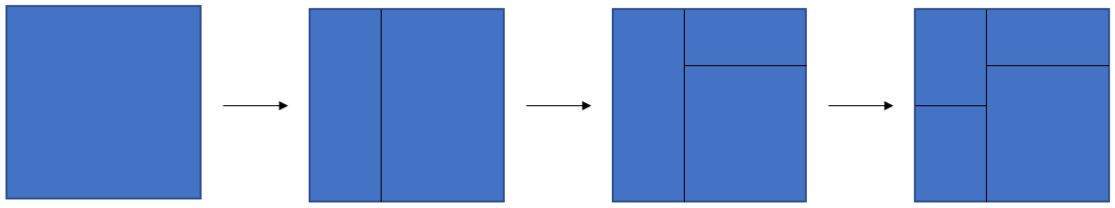
Ideally, you want to travel exactly T miles a day and stop at a motel at the end of the day, but this may not be possible (depending on the spacing of the motels). Instead, you receive a *penalty* of $(T - x)^8$ each day, if you travel x miles during the day. The goal is to plan your stops to minimize the total penalty (over all travel days).

Describe and analyze an algorithm that outputs the minimum penalty, given the locations $\{r_i\}$ of the motels and the value of T.

7 Geometric Knapsack

Suppose we have a piece of cloth with side lengths X, Y, where X, Y are positive integers, and a set of n *products* we can make out of the cloth. Each product is a rectangle of dimensions $a_i \times b_i$ and of value c_i, where all these numbers are positive integers.

We want to cut our large piece of cloth into multiple smaller rectangles to sell as products. Any rectangle not matching the dimensions of one of these products gets us no value. To cut the cloth, we are using a machine that takes one piece of cloth and cuts it into two, where the dividing line between the two pieces must be a vertical or horizontal line going all the way through the cloth. For example, the following is a valid series of cuts:



Give an efficient algorithm that determines the maximum value of the products that can be made out of the single $X \times Y$ piece of cloth. You may produce a product multiple times, or none if you wish. Give a three-part solution.

8 (Extra Credit) Job Scheduling Redux

We have k supercomputers, and n jobs to run on these supercomputers. The ith job has processing time t_i. We want to come up with an assignment of jobs to supercomputers, so that the *maximum* total processing time of jobs assigned to any supercomputer is minimized. That is, if J_ℓ is the set of jobs assigned to the ℓ-th super computer, then we want to minimize $T_{max} := \max_\ell \sum_{i \in J_\ell} t_i$.

Let T^*_{max} be the minimum value of T_{max} across all assignments. Give an efficient algorithm that finds an assignment of jobs to machines such that T_{max} for this assignment is at most c times T^*_{max} for some constant $c \geq 1$. Give a three-part solution, and specify in the proof of correctness what c is for your solution.

Solutions where c is larger than the value of c obtained by the staff's solution will not get credit.