CS 170 Efficient Algorithms and Intractable Problems
Spring 2016 Alessandro Chiesa and Umesh Vazirani **HW 7**

Due March 7, 11:59pm

**1. (20 pts.) The Quest Begins**

You're on a quest to visit the magical city of Pimentel, and the only way to get there is through a narrow path across a mountain. A dragon lives on the mountaintop, and eats all those who dare intrude upon his mountain – unless they pay a sufficiently high bribe. You'll need to earn as many gold coins as you can in the $n$ days before you have to face the dragon.

There are two nearby villages where you can earn some money doing odd jobs. The villages publish lists $A[1..n]$ and $B[1..n]$, where $N[i]$ is the nonnegative integer number of gold coins available to be earned by working at village $N$ on day $i$. (You're confident that you're a hard enough worker that you can always earn all $N[i]$ coins available.)

The travel time between the two villages is a day, so that if you're working at $A$ on day $d$ and decide to switch to $B$, you can start working there on day $d+2$. You can start at either village on day 1.

Design an efficient algorithm to find the highest number of gold coins you could earn in the next $n$ days.

**2. (25 pts.) Giant's Riddle**

Having successfully bribed the dragon, you come upon a bridge over a huge chasm; you can just see the gates of Pimentel on the other side. A huge giant armed with a club blocks the entrance of the bridge. He asks you the following riddle, and promises to let you pass if you can solve it.

(a) **(20 pts.)** Consider the "longest common increasing subsequence" problem: given two sequences $a = a_1, a_2, \ldots, a_n$, and $b = b_1, b_2, \ldots, b_m$, find the longest possible subsequence that is both (1) a subsequence of both $a$ and $b$ and (2) is strictly increasing.

For example, the longest common increasing subsequence of $(1,2,3,5,6,2,4)$ and $(2,1,3,0,7,4,5,6)$ is $(1,3,5,6)$. We can verify that there is no longer subsequence of both sequences that is also strictly increasing.

Devise an efficient algorithm to return the length of the longest common increasing subsequence (algorithms can optionally find and return the subsequence as well).

*Input:* Two sequences of integers $a, b$.

*Output:* A single integer, denoting the length of the longest common increasing subsequence.

The runtime of your algorithm should be at most $O(nm(n+m))$.

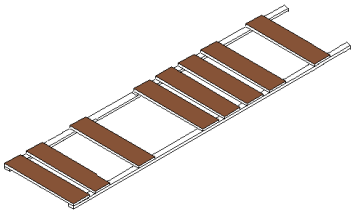
(*The giant taunts you, telling you it's possible to solve this even in $O(nm)$ time.*)

(b) **(5 pts.)** The giant wants to double-check you truly understand your answer. Calculate the longest common increasing subsequence (not just its length) on the following two lists; show your work by writing the list/table of subproblems you solved to find the answer.

(*YOUR CAL ID*) and (3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8).

**3. (15 pts.) Bridge Hop**

Having outwitted the giant's defenses, it's now time to cross the bridge. You notice the bridge is constructed of a single row of planks. Originally there had been $n$ planks; unfortunately, some of them are now missing, and you're no longer sure if you can make it to the other side. For convenience, you define an array $V[1..n]$ so that $V[i]$ = TRUE iff the $i$th plank is present. You're at one side of the bridge, standing still; in other words, your *hop length* is 0 planks. Your bridge-hopping skills are as follows: with each hop, you can increase or decrease your hop length by 1, or keep it constant.



For example, the image above has planks at indices [1, 2, 4, 7, 8, 9, 10, 12], and you could get to the other side with the following hops: [0, 1, 2, 4, 7, 10, 12, 14].

Clarifications: You start at location 0, just before the first plank. Arriving at any location greater than $n$ means you've successfully crossed. Due to your winged shoes, there is no maximum hop length. But you can only hop forward (hop length cannot be negative).

Devise an efficient algorithm to determine whether you can make it to the other side.

**4. (20 pts.) A Sisyphean Task?**

You've crossed the bridge and arrived at the gates of Pimentel. They're locked, but you notice a giant-sized scale in front of them, with the positive integer $k$ written on it. You deduce that the gates only unlock when the giant, who weighs $k$ pounds, steps on the scale.

You look around for anything that might help, and you notice $n$ boulders conveniently arrayed nearby, each boulder $i$ helpfully labeled with its positive integer weight $w_i$. You jot down the list of weights $W[1..n]$. You'd like to determine if there is any set of boulders that together weigh exactly $k$ pounds, so that you can place them on the scale to impersonate the giant, unlock the gates, and enter the city.

(a) **(18 pts.)** Design an algorithm to do this.

(b) **(2 pts.)** Is your algorithm polynomial in the *size* of the input?

**5. (20 pts.) City Tour**

You're finally inside, and you can't wait to tour Pimentel and all of its wondrous attractions! Its layout can be represented as an undirected graph with positive lengths on all of the edges, where the length of an edge represents the time it takes to travel along that edge. You have an ordered sequence of $m$ attractions which you would like to visit. Each attraction is a vertex in the graph. There may be vertices in the graph that are not attractions.

Due to time constraints, you can only visit $k$ of these attractions, but you still want to visit them in the same order as originally specified. What is the minimum travel time you incur?

Design an efficient algorithm for the following problem:

*Input:* a graph $G = (V, E)$, with lengths $\ell : E \to \mathbb{R}_{>0}$, attractions $a_1, \ldots, a_m \in V$, and $k \in \mathbb{N}$

*Output:* The length of the shortest path that visits $k$ of the attractions, in the specified order

In other words, we want to find a subsequence of $a_1, \ldots, a_m$ that contains $k$ attractions, and find a path that visits each of those $k$ attractions in that order—while minimizing the total length of that path. More precisely, we want to find the length of the shortest path of the form $a_{i_1} \rightsquigarrow a_{i_2} \rightsquigarrow \cdots \rightsquigarrow a_{i_k}$, where the indices $i_1, \ldots, i_k$ can be freely chosen as long as they satisfy $1 \leq i_1 < i_2 < \cdots < i_k \leq m$. You don't need to output the path or the subsequence—just the length is enough.

Clarification: Suppose the shortest path from $a_1$ to $a_2$ goes through some other attraction, say $a_5$. That's OK. If the subsequence starts $a_1, a_2, \ldots$, it is permissible for the path from $a_1$ to $a_2$ to go through $a_5$ along the way to $a_2$, if this is the shortest way to get from $a_1$ to $a_2$ (this doesn't violate the ordering requirement, and $a_5$ doesn't count towards the $k$ attractions).

**6. (* pts.) Programming Assignment: Finding Counterexamples to Greedy Algorithms**

*Grading: This problem will be worth 40 points total, with 20 points coming from Homework 6 and 20 points coming from homework 7. So if you get 30 points on this problem then you will get 15 points for it on Homework 6 and 15 points on Homework 7. Thus, the other problems on this homework will contribute to 80% of your score for it, with the other 20% coming from the programming assignment.

*Please see HW 6 for the problem text and submission instructions.*