CSC373H1S 2014 **Assignment A2**
Due: Thursday, February 13, 10PM Total Score: 10 points

Please follow the instructions provided on the course website to submit your assignment. You may submit the assignments in pairs. Also, if you use *any* sources (textbooks, online notes, friends) please cite them for your own safety.

You can use any data-structures and algorithms discussed in CSC263 (e.g. merge-sort, heaps, etc.) and in the lectures of this class by stating their name. You do not need to provide any explanation or pseudo-code for their implementation. You can also use their running time without proving them: for example, if you are using the merge-sort in your algorithm you can simply state that merge-sort's running time is $O(n \lg n)$.

For each problem below, give an algorithm which solves that problem and prove that your algorithm is correct (that is, it outputs the optimal solution for every possible input). Also give a bound on the time complexity of the algorithm.

If you give a dynamic programming algorithm for any of the problems posed below, be sure to include:

1. The definition of the recurrence implemented by the dynamic program, along with a high-level description of what each cell in the recurrence is holding.
2. A proof that the recurrence is correct.
3. A dynamic programming algorithm using memoization computing the recurrence.

For an example of a "high-level description", in the Weighted Interval Scheduling problem (WIS) from Lecture 7, a high-level description of a cell in the recurrence is

$D[i] :=$ the optimal solution to WIS when considering the first i intervals $\{I_1, I_2, \cdots, I_i\}$ in the EFT ordering.

Questions

1. **Treepaths!**

   Give an efficient algorithm for the following problem.

   Input: A weighted, rooted tree $T = (V, E)$ with root r and arbitrary integer (positive or negative) vertex weights $w : V \to \mathbb{Z}$.

   Output: The simple path P from the root r to a leaf ℓ which maximizes the sum of weighted vertices visited on P. That is, a simple path P from the root to a leaf such that $\sum_{v \in P} w(v)$ is maximized.

   To receive full marks, your algorithm must run in time linear in the number of nodes in the tree (that is, $O(|V|)$). For this question assume that the addition of any two integers can be performed in $O(1)$ time.

2. **Line Intersections**

   Suppose you are given a set L of n line segments in the plane, where each segment has one endpoint on the line $y = 0$ and one endpoint on the line $y = 1$, and all $2n$ endpoints are distinct. Give an algorithm to compute the largest subset of L in which no pair of segments intersects. To obtain full marks your algorithm should run in time $O(n \log n)$.

3. **Line Intersections (Redux)**

   Now, suppose you are given a set L of n line segments in the plane, but the endpoints of the segment lie on the unit circle (e.g. defined by the equation $x^2 + y^2 = 1$), and all $2n$ endpoints are distinct. Give an algorithm to compute the largest subset of L in which no pair of segments intersects. To obtain full marks your algorithm should run in time $O(n^3)$[1].

[1] $O(n^3)$ is not the best running time possible for this problem. I will offer a bonus (up to 5%) for anyone who can beat the $O(n^3)$ algorithm.

4. **Percival the Opulent**

The most daring and ostentatious knight in the Checkerboard Kingdom, Percival the Opulent, is on the run! He is being chased by a cavalry of enemy knights after sneaking deep into enemy lands to save as many damsels as possible. Percival needs to escape *fast*. Of course, as a knight, he moves as all knights move: in an "L" shape. For example, if Percival is the "P" in Figure 1, then the arrows will determine where he can move in one step (one of the squares 1, 2, 3 or 4). Note that *Percival can only move to these squares* – he cannot perform the other "usual" knight moves to the top or to the left. It is up to you to plot Percival's escape route. We will

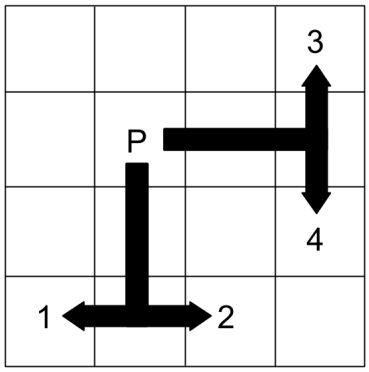

Figure 1: Poor Percival!

assume that there is an $n \times n$ checkerboard, and Percival can start in *any* square in the topmost row. You want to find a route for Percival using only the "knight moves" displayed above (that is, moves that go to the right or down) that will get Percival from his starting point to *any* square in the bottom row.

However, Percival does not want to rush home empty-handed. Each square of the input checkerboard will come with a positive integer that represents how many damsels Percival can save from that square if he lands there during his escape. Of course, Percival wants to maximize the number of damsels in his damsel-sack by the time he escapes.

So, here is the problem. You will be given a positive integer n, representing the size of the checkerboard on which you must plan Percival's escape route. You are also given an $n \times n$ array D where for all $1 \leq i, j \leq n$ the value $D[i, j]$ is a non-negative integer representing how many damsels are on the square in row i and column j of the checkerboard (assume that the top row of the checkerboard is row 1 and the leftmost column of the checkerboard is column 1). A *square* S in the checkerboard will be defined by a pair of integers $S = (i, j)$, where S lies in the ith row and the jth column. An *escape route* for Percival is defined as a sequence of squares $\mathcal{E} = \{P_1, P_2, \cdots, P_\ell\}$ where

(a) The starting square P_1 lies in row 1 of the checkerboard.

(b) The last square P_ℓ lies in row n of the checkerboard.

(c) Each square P_i is reachable from the square P_{i-1} by one of the knight moves depicted in Figure 1.

(d) No square is visited twice in an escape route.

For any escape route $\mathcal{E}$, the number of damsels saved by Percival on $\mathcal{E}$ is simply the sum of the number of damsels appearing on each square in $\mathcal{E}$.

Give an efficient algorithm which, given a positive integer n and array D as above, outputs the escape route $\mathcal{E}$ which maximizes the number of damsels that Percival can save. To achieve full marks your algorithm must run in time polynomial in n.

5. **A Lesson in Economics**

Consider the following simple model of a modern "supply chain". We want to supply a product (call it Skub[2]) to a collection of consumers. Of course, we (the suppliers of Skub) don't interact directly with the consumers. Rather, we interact with a collection of *distributors* $\mathcal{D} = \{D_1, D_2, \cdots, D_\ell\}$. The distributors interact with a set of *business owners* $\mathcal{B} = \{B_1, B_2, \cdots, B_m\}$, and the business owners sell directly to the consumers $\mathcal{C} = \{C_1, C_2, \cdots, C_n\}$.

Suppose that we have an infinite supply of Skub to deliver to the consumers. For each $1 \leq i \leq \ell$, the distributor D_i will buy *at most* d_i units of Skub from us. For each i and $1 \leq j \leq m$, the business owner B_j will buy *at most* b_{ij} units of Skub from distributor D_i. For each j and $1 \leq k \leq n$, the consumer C_k will buy *at most* c_{jk} units of Skub from the business owner B_j. Assume that every distributor or business owner in the supply chain will sell exactly as much Skub as they can buy.

Give an efficient algorithm that, given any set of distributors $\mathcal{D}$, business owners $\mathcal{B}$, and consumers $\mathcal{C}$ and the upper bounds on the number of units that each person can buy from the previous person in the supply chain, outputs the following:

(a) The maximum number of units of Skub that can be pushed through the supply chain to the consumers.

(b) In this supply chain, the consumer responsible for buying the most units of Skub.

(c) Also in this supply chain, a sequence D_i, B_j, C_k of distributor, business owner, and consumer such that D_i sells some non-zero amount of Skub to B_j and B_j similarly sells some non-zero amount of Skub to C_k, but the amount of Skub sold in total $d_i + b_{ij} + c_{jk}$ is minimized.

To receive full marks, your algorithm must run in time that is polynomial in $|\mathcal{D}|$, $|\mathcal{B}|$, and $|\mathcal{C}|$.

6. **This fighting is tearing us apart!**

Here is a problem that we have all likely dealt with before: *chore distribution*. You are given a set of roommates $R = \{r_1, r_2, \cdots, r_n\}$ as well as a set of chores $C = \{c_1, c_2, \cdots, c_m\}$. Each roommate r_i has a set of preferred chores $P_i \subseteq C$, as well as a non-negative integer z_i which represents how many chores the roommate r_i is willing to do. You must give an algorithm which, given any set of roommates R (along with their individual chore preferences P_i and chore limits z_i) and any set of chores C, outputs an assignment of roommates to chores such that:

(a) Every roommate r_i is assigned only to chores that he or she prefers (i.e. chores in P_i).

(b) Each roommate r_i is assigned at most z_i chores.

(c) Each chore is assigned to at most one roommate.

(d) The total number of chores which end up being assigned are maximized.

To receive full marks your algorithm must run in time that is polynomial in $|R|$ and $|C|$.

[2] http://pbfcomics.com/20/