Design and Analysis of Algorithms **out:** February 12, 2015

Problem Set 3

Lecturer: Daniel A. Spielman **due: 2:30** PM, February 19, 2015

Instructions. Read Carefully

1. Make sure that your yale **netid** appears on **every** page of your problem set, and that it is written clearly.

2. The problem sets are due at the beginning of class. Solution sets will be provided in class. For this reason, late problem sets are not accepted.

3. You will **hand in each problem separately**, so make sure that they are not stapled together and that no problem appears on the back of another. There will be 4 boxes in the front of the class, one for each problem. Put each problem in the correct box.

4. Alternatively, you may submit the problem set electronically via the Classes server. If you do so, submit it as a **pdf** and make sure that your **netid** appears on **every** page. **Each problem must be in its own document.**

5. Cite any resources that you use, other than the textbook, TA and instructor.

6. You **must not** search the web for solutions to similar problems given out in other classes.

Collaboration Policy

You must write the solutions to the problems independently and in your own words.

However, you are allowed to discuss the problem sets in small groups of no more than 4 students. To ensure that you understand the solutions you submit, you are forbidden from taking written notes during your discussions. You must list at the top of the problem set everyone (other than course staff) with whom you have discussed the problem set. Failure to list people with whom you have discussed a problem set is considered a violation of academic honesty.

Problem 1: Maximum Matching in the Line

If you've ever read Flatland, you are familiar with the limitations of life in Lineland. Do not pity the Linelanders. Their limitations greatly simplify algorithmic challenges.

You are given a list of n people, who we will refer to as 1 through n, all of whom are arranged in a line. Your job is to construct a Maximum Weight Matching among these people, subject to the restriction that you are only allowed to match person i with person $i-1$ or person $i+1$. Since this is a matching, you cannot match person i with both $i-1$ and $i+1$. But, you are welcome to leave person i unmatched.

For each $1 \leq i < n$, you are given a number $w_{i,i+1} > 0$ indicating the benefit of matching person i with person $i+1$. The benefit of a matching is the sum of the benefits over all pairs that are matched.

Give a polynomial-time algorithm that computes a matching of maximum benefit. In your answer, be sure to clearly identify the subproblems, clearly explain how you solve each subproblem using other subproblems, and to explain why your solution is correct. If it is not clear that your algorithm runs in polynomial time, explain why it does.

Problem 2: Revisiting Segmented Least Squares

In this problem you will solve a variation of the Segmented Least Squares problem (see Section 6.3) in which you are *required* to use *exactly* k segments.

That is, you are given as input a number k along with n increasing x values $x_1 < \ldots < x_n$ and y values $y_1, \ldots, y_n$. Assume that for every $1 \leq s \leq f \leq n$ you are given $\text{err}_{s,f}$, the error made by the best-fit line to points s through f. You will choose the start points of segments 2 through k, $s_2, \ldots, s_k$. Given these, we know that segments 1 through $k-1$ finish at points $f_1, \ldots, f_{k-1}$ where $f_i + 1 = s_{i+1}$. It is always the case that $s_1 = 1$ and $f_k = n$. The error made by a choice of start points is then

$$\sum_{j=1}^{k} \text{err}_{s_j,f_j}.$$

Give a polynomial-time algorithm that minimizes the error in this version of the Segmented Least Squares problem. If it is not clear that your algorithm runs in polynomial time, explain why it does. In your answer, be sure to clearly identify the subproblems, clearly identify the recurrence you are using to solve the problem, and to explain why your solution is correct.

You do not need to compute the values $\text{err}_{s,f}$. If you like, you may assume that these numbers are part of the input.

Problem 3: Survivor Yale

After exhausting all possible exotic locations, the nth season of Survivor will take place at Yale. The most difficult challenge will involve teams consisting of two players each.

Let's call the players on a team Player 1 and Player 2. Player 1 will be required to traverse a sequence of platforms $a_1, \ldots, a_m$, in order. Player 2 will be required to traverse a sequence of platforms $b_1, \ldots, b_n$, also in order. Unconstrained, it would be easy for either player to traverse their platforms. However, the players are connected by a rope of length l, and so cannot occupy platforms that are distance greater than l apart.

They are also given a table $D(i, j)$ of the distances between platforms a_i and b_j. Player 1 can move from platform a_i to platform a_{i+1} while Player 2 is on platform b_j if both $D(i, j)$ and $D(i + 1, j)$ are at most l. We will assume for simplicity that only one player moves at any moment, and that no player is ever allowed to move backwards.

The players must determine a schedule of movements that will allow them to go from platforms (a_1, b_1) to platforms (a_m, b_n). But, I will allow you to just solve the problem of determining whether or not this is even possible.

Design a polynomial time algorithm that takes as input the table D and determines if there exists a sequence of moves that allow the players to reach the end. If it is not clear that your algorithm runs in polynomial time, explain why it does. Prove that your algorithm is correct.

Problem 4: A Xylophone-Playing Robot

We will consider the problem of designing an algorithm to help a primitive robot play the xylophone. There are many types of xylophones (if you are unfamiliar with these, Google them.) For the purpose of this problem, we assume that it consists of one row of equally spaced bars. You should treat the number of bars as a constant, although some xylophones have many (check out vibraphones). A note is played by striking a bar with a mallet. Our robot will have two robotic arms, each of which will hold one mallet. The robot can only hit a bar with a mallet if the mallet is over that bar.

We will divide time into small steps, let's say of 1/10 of a second. At each time step, each robotic arm can

1. Either hit the note under it or not, and
2. Stay put, move one bar left, or move one bar right.

Your algorithm should take a piece of music and determine whether or not the robot can play it. If there were only one arm, this problem would be easy. You would just have to

figure out if it could get from one note to the next that it needs to play in the amount of time between notes. The problem is more complicated when there are two arms because there are more choices. The arms can cross, so a note could be played by either arm.

More formally, a piece of music should consist of n time steps. Each step can have up to one note (bar on the xylophone) that should be played. Let's call the bars $1, \ldots, k$. So, a piece of music will be a sequence of numbers in $\{0, 1, \ldots, k\}$, with 0 indicating that no note is played at that time step.

Design a polynomial time algorithm that takes as input a piece of music and determines whether or not it can be played by the robot. If it is not clear that your algorithm runs in polynomial time, explain why it does. Prove that your algorithm is correct.

Hint: I suggest thinking about the configuration of the robot at each time step. By configuration, I mean where the arms are.