CS 4820, Summer 2010 Out: July 16. **Due: Tuesday, July 20, 8:30am**

Homework 4

Reading: Chapter 6 (all sections except Section 6.10)

Problem 1. So, you did great in the time trials last week. ☺ Now you are looking for something even more unconventional! And you do find a different time trial which is as follows.

So, it goes on for n hours, in each hour specifying how much you are allowed to bike in that hour (that is, in i-th hour, you can bike at the most x_i miles). Owing to your hill-less training, you find it hard to bike for a long time continuously, and you need breaks — long breaks — to charge yourself up. So, in an hour $i : 1 \leq i \leq n$, you can either decide to bike or decide to rest and *charge up* for the future hours! After doing some research on your weekly rides data, you have figured out that the number of miles you can bike in a given hour goes down if you have been biking continuously for longer time in the past. That is, in the first hour after rest, you can bike s_1 miles, in the second hour after rest, you can bike $s_2 < s_1$ miles (if you have not rested since) and so on. That is, if you have not rested for i hours in the past, you can ride s_i miles in an hour, where naturally, $s_1 > s_2 > s_3 > \cdots > s_n$. (Notice that if you rest in hour i, then you are allowed to ride 0 miles in that hour. In this case, in hour $i+1$, we say that you have not rested for last one hour, so you can ride s_1 miles.) So, in any single day j, if you have not rested for the last i hours, you can bike $\min\{s_i, x_j\}$ miles (this comes from constraints on how long you are allowed to bike, and how long your stamina allows you to bike).

Here is the problem: Given numbers $x_i : 1 \leq i \leq n$ indicating how many miles you are allowed to bike at the most on i-th day, and your biking profile in the form of $s_i : 1 \leq 1 \leq n$ indicating how many miles you are capable of riding if you have not rested for last i hours, choose the hours in which to bike and in which hours to rest so as to maximize the distance you travel. You can assume that you have rested in the hour preceding the first hour of the time trial, so you are totally refreshed in the first hour.

Example: Suppose $n = 4$, and the values of x_i and s_i are given by

	Hour 1	Hour 2	Hour 3	Hour 4
x	27	10	23	28
s	25	20	12	5

The best solution here is to bike in the first hour, rest in the second, and bike in third and fourth. This way, you can bike a total of $\min\{27, 25\} + 0$ (for rest) $+ \min\{23, 25\} + \min\{28, 20\} = 25 + 23 + 20 = 68$.

(a) Give an example of an instance where you are allowed to bike as much as you can (that is, $x_i \geq s_1$ for all i), but in the optimal solution, you rest at least twice. In addition to the example, you should say what the optimal solution is. You don't have to prove that the solution is optimal, but if it is not obvious, please give a sketch of why you think it is optimal.

(b) Give an efficient algorithm that takes values $x_i : 1 \leq i \leq n$ and $s_i : 1 \leq i \leq n$, and returns the total *number* of miles you can bike in the optimal solution.

Problem 2. Okay, let us take a break from biking. You are at work and you don't quite like it because of (i) the hierarchical nature of your company, and (ii) the work keeps you away from biking. But forget about biking. Back to work! The president of your company wants to organize a summer party. As we mentioned, there is a elaborate hierarchical structure among the employees, which can be expressed as a tree in which employees are nodes and the "parent" of a node is the supervisor of the corresponding employee. The president is of course at the root of the tree. With some background research, the public relations office has assigned a *friendliness* score to each employee, which is a real number (let us call this score f_u for employee u). In order to make the party fun for all attendees, the president does not want both an employee and his or her immediate supervisor to attend.

(a) Describe an algorithm to make up the guest list. The goal should be to maximize the sum of the friendliness score of all invited guests. Prove the correctness of your algorithm and analyze its running time.

(b) Give an example where the optimal solution does not invite the president. Now look at the solutions in which president is always invited, and modify your algorithm such that it gives the optimal solution with the constraint that the president is always invited.

Problem 3. In this problem, we want to use dynamic programming on a directed graph $G = (V, A)$ for speech recognition. Each arc $a \in A$ is labelled with a sound $\sigma(a)$ from a finite set of sounds Σ in the language. The labelled graph is a formal model of a person speaking a restricted language (with sounds Σ). Each path in the graph starting from a *special vertex* $v_0 \in V$ corresponds to a possible sequence of sounds produced by the model. The label of a directed path is defined to be the concatenation of the labels of the arcs on that path.

(a) Describe an efficient algorithm that, given an arc-labelled graph G with special vertex v_0 and a sequence $s = (s_1, s_2, ..., s_k)$ of sounds from the language, returns a path in G that begins at v_0 and has s as its label, if such a path exists. Otherwise, the algorithm should indicate that no such path exists. Prove the correctness of your algorithm and analyze its running time.

To extend the model, suppose that every arc $a = (u, v) \in A$ has also been associated a nonnegative probability $p(a)$ of traversing the arc a from vertex u (its tail) and producing the corresponding sound. The sum of probabilities of the arcs leaving any vertex equals 1. The probability of a path is defined to be the product of the probabilities of its arcs. We can view the probability of a path beginning at v_0 as the probability that a "random walk" beginning at v_0 will follow the specified path, where the choice of which arc to take at a vertex u is made probabilistically according to the probabilities of the available arcs leaving u.

(b) Change the algorithm from part **(a)** (or give a different algorithm) so that if a path is returned, it is a *most probable path* starting at v_0 and having label s. You can assume that if a path exists, there is at least one path which has positive probability. Prove the correctness of your algorithm and analyze its running time.