15-451/651 Algorithms, Spring 2018

Homework #6 **Due: April 17, 2018**

Same instructions as previous written HWs.

(25 pts) 1. **Competing with the Best.**

Here's an alternative online algorithm for the List Update problem discussed in lecture.

> **Move-Half-Way-To-Front:** When an element x is accessed, swap x with its predecessors until its distance to the front is at most half what it was before. To be more precise, suppose when we access x there there are k elements in front of it. After the operation there will be $\lfloor k/2 \rfloor$ elements in front of it.

Is there a universal constant c such that this algorithm (MHWTF) is c-competitive? (That is, the cost for MHWTF should be at most c times the optimal cost, for all starting lists, and all input sequences.)

If the answer to the above question is "yes", then find the constant c such that: (1) MHWTF is c-competitive, and (2) MHWTF is not c'-competitive for any $c' < c$ (for sufficiently large n). Prove both parts.

If the answer to the above question is "no", then give an example showing that the competitive ratio goes to infinity.

(25 pts) 2. **Algorithms from another Planet.**

Your spaceship has landed on LineWorld, which as the name suggests, looks like the real line $\mathbb{R}$. You start at the origin $x_0 = 0$. At each day $t \geq 1$, the adversary puts a linear force field which is centered at some location $c_t \in \mathbb{R}$ and has slope $s_t \geq 0$ and offset $b_t \geq 0$. This force field means that if you are standing at location $x \in \mathbb{R}$, you spend $f_t(x) := b_t + s_t \cdot |x - c_t|$ of fuel for this day. Moreover, you spend 1 unit of fuel for each unit of distance you move along the line. So on day t, you observe the force field for this timestep (i.e., you get to know s_t, b_t, c_t). Then you may move from the previous location x_{t-1} to some location x_t (it takes you zero time to travel), and then spend day t at this location x_t, hence burning $|x_t - x_{t-1}| + f_t(x_t)$ amount of fuel on this day. The total fuel usage until time T would hence be

$$\sum_{t=1}^{T} \Big(|x_t - x_{t-1}| + f_t(x_t) \Big).$$

You cannot see the future, so you make decisions on each day t without knowing the force fields on days $t + 1$ or later. You want to give an online algorithm that is 4-competitive. I.e., for each T, the fuel usage of the algorithm until day T is at most 4 times the fuel usage of any other algorithm B until that same day.

Here's an algorithm: on day t, move from x_{t-1} towards the center c_t, until either (a) you reach c_t, or (b) xthe distance you have traveled (i.e., $|x_{t-1} - x|$) equals $f_t(x)$ at the point x you reach. This is your new point x_t. Show this algorithm is 4-competitive.

Hint: What would be a good potential to use?

(In case you can correctly show a better competitive ratio, that is great, we're looking for a ratio of 4.)

(25 pts) 3. **Adapt, Adopt, Improve.**

Here is a variation on the deterministic Weighted-Majority algorithm, designed to make it more adaptive.

(a) Each expert begins with weight 1 (as before).

(b) We predict the result of a weighted-majority vote of the experts (as before).

(c) If an expert makes a mistake, we penalize it by dividing its weight by 2, but *only* if its weight was at least 1/4 of the average weight of experts.

Prove that in any contiguous block of trials (e.g., the 51st day through the 77th day), the number of mistakes made by the algorithm is at most $O(m + \log n)$, where m is the number of mistakes made by the best expert *in that block*, and n is the total number of experts.

(25 pts) 4. **Programming Problem: Ski Slopes**

You are building a new ride at Kennywood. This involves a collection of slides between various locations. In order to make the ride as safe as possible, you want to minimize the slope of the slides. But to make the ride interesting, the endpoints of some of the slides are fixed at certain heights.

We model this problem as an undirected graph $G = (V, E)$ where the edges are the slides, so that each edge $e \in E$ has a positive integer edge length $\ell(e)$, and a subset of vertices have a specified (possibly negative) integer height $h(v)$. Your program will assign real valued heights to all the remaining vertices so as to minimize the maximum *slope* of all the edges.

Given an assignment of heights to all the vertices, the slope of an edge $e = (u, v)$ is defined as follows:

$$\text{slope}(e) = \frac{|h(u) - h(v)|}{\ell(e)}$$

Your program should be called `slope.c` or something analogous. Also, we'll provide code for the simplex algorithm in some common languages, which you can use.

Finally, 5 of the 25 points are for giving a short explanation of what your algorithm does (in the comments above the code). We understand that the simplex algorithm takes exponential time in the worst-case, but you should argue that had you used a poly-time LP solver, you'd have got a correct poly-time algorithm for the above slope-assignment problem.

Input:

The first line of input contains n and m the number of vertices and edges of G. $1 \leq n \leq 1000$, $1 \leq m \leq 2000$. The following n lines each contain either the character "$*$", or an integer. The $*$ indicates that the hight is unspecified, and is to be computed by your program.

The following m lines each contain a pair of vertex numbers in the range $[0, n-1]$, and an integer representing the length of that edge. There are no multi-edges or self-loops. The length will be an integer in the range $[1, 10000]$. There will be no edges connecting a pair of vertices with known heights.

Output:

Output the minimum obtainable slope with six digits after the decimal point. Your answer should be within a relative or absolute error of at most 10^{-6} of the correct answer. The time limit is 10 seconds.

Examples:

```
3 2                                          slope = 5.000000
0
*
-100
0 1 1
1 2 19

6 5                                          slope = 3.500000
*
8
1
2
5
4
0 1 1
0 2 1
0 3 1
0 4 1
0 5 1
```