CSE 421 Intro to Algorithms Autumn 2017

Homework 5

Due Monday November 13, 4:00 pm

Problem 1:

Suppose we want to replicate a file over a collection of n servers, labeled $S_1, S_2, \ldots, S_n$. To place a copy of the file at server S_i results in a *placement cost* of c_i, for an integer $c_i > 0$.

Now, if a user requests the file from server S_i, and no copy of the file is present at S_i, then the servers $S_{i+1}, S_{i+2}, S_{i+3} \ldots$ are searched in order until a copy of the file is finally found, say at server S_j, where $j > i$. This results in an *access cost* of $j - i$. (Note that the lower-indexed servers $S_{i-1}, S_{i-2}, \ldots$ are not consulted in this search.) The access cost is 0 if S_i holds a copy of the file. We will require that a copy of the file be placed at server S_n, so that all such searches will terminate, at the latest, at S_n.

We'd like to place copies of the files at the servers so as to minimize the sum of placement and access costs. Formally, we say that a *configuration* is a choice, for each server S_i with $i = 1, 2, \ldots, n-1$, of whether to place a copy of the file at S_i or not. (Recall that a copy is always placed at S_n.) The *total cost* of a configuration is the sum of all placement costs for servers with a copy of the file, plus the sum of all access costs associated with all n servers.

Give a polynomial-time algorithm to find a configuration of minimum total cost.

Problem 2:

In a word processor, the goal of "pretty-printing" is to take text with a ragged right margin, like this,

```
Call me Ishmael.
Some years ago,
never mind how long precisely,
having little or no money in my purse,
and nothing particular to interest me on shore,
I thought I would sail about a little
and see the watery part of the world.
```

and turn it into text whose right margin is as "even" as possible, like this.

```
Call me Ishmael. Some years ago, never
mind how long precisely, having little
or no money in my purse, and nothing
particular to interest me on shore, I
thought I would sail about a little
and see the watery part of the world.
```

To make this precise enough for us to start thinking about how to write a pretty-printer for text, we need to figure out what it means for the right margins to be "even." So suppose our text consists of a sequence of *words*, $W = \{w_1, w_2, \ldots, w_n\}$, where w_i consists of c_i characters. We have a maximum line length of L. We will assume we have a fixed-width font and ignore issues of punctuation or hyphenation.

A *formatting* of W consists of a partition of the words in W into *lines*. In the words assigned to a single line, there should be a space after each word except the last; and so if $w_j, w_{j+1}, \ldots, w_k$ are assigned to one line, then we should have

$$\left[\sum_{i=j}^{k-1}(c_i + 1)\right] + c_k \leq L.$$

We will call an assignment of words to a line *valid* if it satisfies this inequality. The difference between the left-hand side and the right-hand side will be called the *slack* of the line—that is, the number of spaces left at the right margin.

Give an efficient algorithm to find a partition of a set of words W into valid lines, so that the sum of the *squares* of the slacks of all lines (including the last line) is minimized.

Problem 3:

The problem of searching for cycles in graphs arises naturally in financial trading applications. Consider a firm that trades shares in n different companies. For each pair $i \neq j$, they maintain a trade ratio r_{ij}, meaning that one share of i trades for r_{ij} shares of j. Here we allow the rate r to be fractional; that is, $r_{ij} = \frac{2}{3}$ means that you can trade three shares of i to get two shares of j.

A *trading cycle* for a sequence of shares $i_1, i_2, \ldots, i_k$ consists of successively trading shares in company i_1 for shares in company i_2, then shares in company i_2 for shares i_3, and so on, finally trading shares in i_k back to shares in company i_1. After such a sequence of trades, one ends up with shares in the same company i_1 that one starts with. Trading around a cycle is usually a bad idea, as you tend to end up with fewer shares than you started with. But occasionally, for short periods of time, there are opportunities to increase shares. We will call such a cycle an *opportunity cycle*, if trading along the cycle increases the number of shares. This happens exactly if the product of the ratios along the cycle is above 1. In analyzing the state of the market, a firm engaged in trading would like to know if there are any opportunity cycles.

Give a polynomial-time algorithm that finds such an opportunity cycle, if one exists.

Problem 4 (Extra Credit):

In this problem you are given as input a binary tree **T=(V,E)** with each leaf **w** labelled by a symbol $\mathbf{s_w}$ from some fixed alphabet **A**, as well as a two-dimensional non-negative array of costs indexed by pairs of elements of **A** such that **c[s,t]** is the cost of changing symbol **s** to symbol **t** and this satisfies **c[s,s]=0** and **c[s,t]=c[t,s]** for all **s,t** in **A**. Design an algorithm to label each internal node **v** of the binary tree by a symbol $\mathbf{s_v}$ from **A** so as to minimize the sum over all **(u,v)** in **E** of $\mathbf{c[s_u,s_v]}$.

(This kind of problem arises in computational biology when one wishes to assess a potential evolutionary tree and reconstruct properties of potential ancestral species given this tree. In this case each "symbol" might be a very short sequence - or even just a letter - of DNA or protein that might at a given position within a longer sequence, and **c[s,t]** is the cost of the evolutionary change in going from **s** to **t**.)

Problem 5 (Extra Credit):

Use ideas similar to those for finding the closest pair in two dimensional space to give a divide and conquer algorithm for finding the closest pair among **n** points in three dimensional space that is much more efficient than the naïve $\mathbf{O(n^2)}$ time algorithm. Prove the running time of your algorithm. Better running times will yield better credit for this problem.