# CSE 421 Intro to Algorithms Autumn 2017

# Homework 2

**Due Friday October 13, 4:00 pm**

**Problem 1:** A popular game a few years ago was the *Six Degrees of Kevin Bacon* game in which people would try to find the shortest path of co-appearances linking some arbitrary actor to Kevin Bacon; similarly, in Mathematics, people proudly mention their *Erdős number*, how far they are away in the co-authorship graph from Paul Erdős. In both of these there is a graph of people and edges between them if they are socially connected in some way. Of course a single short path between **v** and **w** does not necessarily describe a strong connection between them; it could be merely coincidental. However, if there are a lot of ways in which two people are connected by a shortest path then that suggests a much stronger connection. Suppose that the task instead is to compute the *number of shortest paths* between **v** and **w**.

It turns that this can be done efficiently. Suppose that you are given a graph **G=(V,E)** and two nodes **v** and **w** in **G**. Give an algorithm that computes the number of shortest paths between **v** and **w** in **G** in time **O(n+m)** where **n** is the number of vertices and **m** is the number of edges in **G**. (The algorithm will not be able to exactly list all such paths since there may be many more of them than the time bound.) Justify the running time bound of your algorithm.

**Problem 2:** Your company has been consulted by Homeland Security to do risk analysis for networks. In particular, Homeland Security is interested in determining whether there are single points of network failure **v** in the following sense, if node **v** stopped working then an emergency broadcast message sent from one of the nodes other than **v** would not make it to all the remaining nodes in the network (and, to find what single points of failure exist so that they can be corrected). The issue seems familiar and you recall that your Algorithms instructor suggested that an extension of recursive depth-first search will do the job for problems like this.

Show how to modify the code for recursive depth-first search of undirected graphs to (i) assign each node **v** a number, **dfsnum(v)**, indicating the sequence number for when it was first visited by depth-first search, and (ii) compute **smallest(v)** for each node **v**, the smallest dfsnum of any node that was encountered in the recursive call **DFS(v)**.

Show how to determine whether or not **v** is a single point of network failure in a connected network by comparing **dfsnum(v)** and **smallest(w)** for the children **w** of **v** in the DFS tree. (You will need a separate simpler condition for the root of the DFS.)

**Problem 3:**

Some of your friends have gotten into the burgeoning field of *time-series data mining*, in which one looks for patterns in sequences of events that occur over time. Purchases at stock exchanges—what's being bought—are one source of data with a natural ordering in time. Given a long sequence $S$ of such events, your friends want an efficient way to detect certain "patterns" in them—for example, they may want to know if the four events

```
buy Yahoo, buy eBay, buy Yahoo, buy Oracle
```

occur in this sequence $S$, in order but not necessarily consecutively.

They begin with a collection of possible *events* (e.g., the possible transactions) and a sequence $S$ of $n$ of these events. A given event may occur multiple times in $S$ (e.g., `Yahoo` stock may be bought many times in a single sequence $S$). We will say that a sequence $S'$ is a *subsequence* of $S$ if there is a way to delete certain of the events from $S$ so that the remaining events, in order, are equal to the sequence $S'$. So, for example, the sequence of four events above is a subsequence of the sequence

```
buy Amazon, buy Yahoo, buy eBay, buy Yahoo, buy Yahoo,
buy Oracle
```

Their goal is to be able to dream up short sequences and quickly detect whether they are subsequences of $S$. So this is the problem they pose to you: Give an algorithm that takes two sequences of events—$S'$ of length $m$ and $S$ of length $n$, each possibly containing an event more than once—and decides in time $O(m + n)$ whether $S'$ is a subsequence of $S$.

**Problem 4 (Extra Credit)**

One of the first things you learn in calculus is how to minimize a differentiable function such as $y = ax^2 + bx + c$, where $a > 0$. The Minimum Spanning Tree Problem, on the other hand, is a minimization problem of a very different flavor: there are now just a finite number of possibilities for how the minimum might be achieved—rather than a continuum of possibilities—and we are interested in how to perform the computation without having to exhaust this (huge) finite number of possibilities.

One can ask what happens when these two minimization issues are brought together, and the following question is an example of this. Suppose we have a connected graph $G = (V, E)$. Each edge $e$ now has a *time-varying edge cost* given by a function $f_e : \mathbf{R} \to \mathbf{R}$. Thus, at time $t$, it has cost $f_e(t)$. We'll assume that all these functions are positive over their entire range. Observe that the set of edges constituting the minimum spanning tree of $G$ may change over time. Also, of course, the cost of the minimum spanning tree of $G$ becomes a function of the time $t$; we'll denote this function $c_G(t)$. A natural problem then becomes: find a value of $t$ at which $c_G(t)$ is minimized.

Suppose each function $f_e$ is a polynomial of degree 2: $f_e(t) = a_e t^2 + b_e t + c_e$, where $a_e > 0$. Give an algorithm that takes the graph $G$ and the values $\{(a_e, b_e, c_e) : e \in E\}$ and returns a value of the time $t$ at which the minimum spanning tree has minimum cost. Your algorithm should run in time polynomial in the number of nodes and edges of the graph $G$. You may assume that arithmetic operations on the numbers $\{(a_e, b_e, c_e)\}$ can be done in constant time per operation.