Applied Algorithm Design: Exam

Prof. Pietro Michiardi

Rules and suggestions

- The idea is to complete first the questions and then the exercises.
- Having **the good** answer for all questions will guarantee to pass the exam with 10/20.
- Exercises are difficult if you are not familiar with the course contents. Suggestion: pick one exercise that you think is more feasible for you and focus on that one.

Questions

1. Given a digraph $G = (V, E)$ and two nodes s and t, find the minimum number of edges whose removal disconnects t from s.
   (a) Given this problem, which algorithm can be used to find the solution?
   (b) Which is the running time of this algorithm?
   (c) Why is this question is important in real systems?

2. Given an undirected graph $G = (V, E)$, name a simple algorithm to check for bipartiteness.

3. Draw a simple (e.g. 5 nodes) graph, and illustrate (in a sequence of graphs) at least two iterations of the PageRank algorithm: in particular, each node in the drawings should indicate what is the current PageRank value.

4. In your home works, you had to implement the Fair Sojourn Protocol (FSP).
   - What is the motivation for FSP? Why FIFO or processor sharing are not enough?
   - What is the fundamental assumption of FSP, and by the way for any size-based scheduling protocol?

- In your opinion, what is the impact of wrong estimates on the job sizes?

5. Why is it important for the Gale-Shapley algorithm to output the same result no matter what the proposal order is? Be brief.

Exercise 1

You work for a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, you can only obtain those licenses at the rate of at most one per month.

Each license is currently selling for a price of \$100. However, they are all becoming more expensive according to exponential growth curves: in particular, the cost of license j increases by a factor of $r_j > 1$ each month, where r_j is a given parameter. This means that if license j is purchased t months from now, it will cost $100 \cdot r_j^t$. We will assume that all the price growth rates are distinct; that is, $r_i \neq r_j$ for licenses $i \neq j$ (even though they start at the same price \$100).

The question is: Given that the company can only buy at most one license a month, in which order should it buy the licenses so that the total amount of money it spends is as small as possible?

Give an algorithm that takes the n rates of price growth $r_1, r_2, ...r_n$ and computes an order in which to buy the licenses so that the total amount of money spent is minimized. **The running time of your algorithm should be polynomial** in n.

[Hint] Here we look for a greedy algorithm. Now, since you have to prove optimality, it is helpful to recall that when a greedy algorithm works optimally by putting a set of things in an optimal order, it is often effective to try proving correctness using an **exchange argument.**]

Exercise 2

You're doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with n rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the *highest safe rung*.

It might be natural to try binary search: drop a jar from the middle rung, see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. But this has the drawback that you could break a lot of jars in finding the answer.

If your primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first rung, then the second rung, and so forth, climbing one higher each time until the jar breaks. In this way, you only

need a single jar – at the moment it breaks, you have the correct answer – but you may have to drop it n times (rather than $logn$ as in the binary search solution).

So here is the trade-off: it seems you can perform fewer drops if you're willing to break more jars. To understand better how this tradeoff works at a quantitative level, let's consider how to run this experiment given a fixed "budget" of $k \geq 1$ jars. In other words, you have to determine the correct answer – the highest safe rung – and can use at most k jars in doing so.

(a) Suppose you are given a budget of $k = 2$ jars. Describe a strategy for finding the highest safe rung that requires you to drop a jar at most $f(n)$ times, for some function $f(n)$ that grows slower than linearly.

`[Hint] It should be the case that` $\lim_{n\to\infty} f(n)/n = 0$.

(b) Now suppose you have a budget of $k > 2$ jars, for some given k. Describe a strategy for finding the highest safe rung using at most k jars. If $f_k(n)$ denotes the number of times you need to drop a jar according to your strategy, then the functions $f_1, f_2, f_3, ...$ should have the property that each grows asymptotically slower than the previous one.

`[Hint]` $lim_{n\to\infty} f_k(n)/f_{k-1}(n) = 0$ `for each` k.