Midterm Examination

CS 4104 (Spring 2017)

Assigned: Friday, March 17, 2017.
PDF solutions due on Canvas before the start of class on Monday, March 27, 2017.

Instructions

1. **You must work on your own for this examination, i.e., you are not allowed to have a partner.**
2. Use pseudo-code in the style of the slides or the textbook to describe any algorithm. If you write code, I will not award you any points.
3. For every algorithm you describe, prove its correctness, and state and prove the running time of the algorithm. I am looking for clear descriptions of algorithms and for the most efficient algorithms and analysis that you can come up with. Except for one problem, I am not specifying the desired running time for each algorithm. I will give partial credit to non-optimal algorithms, as long as they are correct.
4. You may consult the textbook, your notes, or the course web site to solve the problems in the examination. Of course, the TA and I are available to answer your questions. You **may not** work on the exam with anyone else, ask anyone questions, or consult other textbooks or sites on the Web for answers. **Do not use** concepts from Chapters 6 and later in the textbook.
5. You must prepare your solutions digitally, i.e., do not hand-write your solutions.
6. I prefer that you use LaTeX to prepare your solutions. However, I will not penalise you if you use a different system. To use LaTeX, you may find it convenient to download the LaTeX source file for this document from the link on the course web site. At the end of each problem are three commented lines that look like this:

   ```
   % \solution{
   %
   % }
   ```

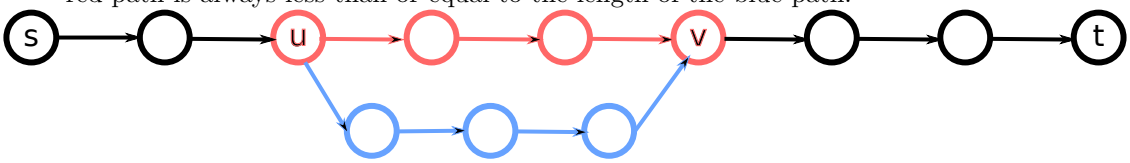
   You can uncomment these lines and type in your solution within the curly braces.

Good luck!

Problem 1 (15 points) Let us start with some quickies. For each statement below, say whether it is true or false. You do not have to provide a proof or counter-example for your answer.

1. Every tree is a bipartite graph.
2. $\sum_{i=1}^{n} i \log i = \Theta(n^2 \log n)$.
3. Dijkstra's algorithm may not terminate if the graph contains negative-weight edges.
4. You solve a problem on an input of size n by dividing it into three subsets of size $n/3$, solving each sub-problem recursively, and combining the solutions in $O(n)$ time. The running time of your algorithm is $O(n \log n)$.
5. In a directed graph, $G = (V, E)$, each edge has a weight between 0 and 1. To compute the length of the *longest* path that starts at u and ends at v, we can change the weight of each edge to be the reciprocal of its weight and then apply Dijkstra's algorithm.
6. Suppose π is the shortest s-t path in a directed graph. Then, there can be two nodes u and v in π such that length of the portion of π connecting u to v is larger than the length of the shortest u-v path in the graph. Here π is the name I am giving to the shortest path between s and t; it is not the mathematical constant.

   In the figure below, π is the path composed of black and red edges. As an example, I have marked two nodes u and v in π and denoted the portion of π connecting u to v using red edges. The blue edges denote another path from u to v in the graph. The figure does not indicate edge costs. If you answer "yes" to this question, you are saying that there are graphs where the length of the red path can be larger than the length of the blue path (knowing that π is the shortest s-t path). If you say "no", then you are saying that for every pair of nodes u and v in π, the length of the red path is always less than or equal to the length of the blue path.



Problem 2 (10 points) Consider the problem of minimising lateness that we discussed in class. We are given n jobs. For each job $i, 1 \leq i \leq n$, we are given a time $t(i)$ and a deadline $d(i)$. Let us assume that all the deadlines are distinct. We want to schedule all jobs on one resource. Our goal is to assign a starting time $s(i)$ to each job such that each job is delayed as little as possible. A job i is *delayed* if $f(i) > d(i)$; the *lateness of the job* is $\max(0, f(i) - d(i))$.

Define

1. the *lateness of a schedule* as $\max_i \left(\max \left(0, f(i) - d(i)\right)\right)$ and
2. the *delay of a schedule* as $\sum_{i=1}^{n} \left(\max \left(0, f(i) - d(i)\right)\right)$.

Note that although the words "lateness" and "delay" are synonyms, for the purpose of this problem we are defining them to mean different quantities: the lateness of a schedule is the *maximum* of the latenesses of the individual jobs, while the delay of a schedule is the *sum* of the latenesses of the individual jobs.

Consider the algorithm that we discussed in class for computing a schedule with the smallest lateness: we sorted all the jobs in increasing order of deadline and scheduled them in this order. We proved that the earliest-deadline-first algorithm correctly solves the problem of minimising lateness. If we were to use the *same proof* to try to demonstrate that the algorithm correctly solves the problem of minimising delay, what is the first point where the proof breaks down? Explain why the proof is incorrect here.

Problem 3 (30 points) This summer, you are working at a synthetic biology company. The company specialises in creating cocktails of microbes (e.g., bacteria and fungi) to synthesize new antibiotics in an effort to combat infectious diseases. Their idea is to start with a compound that is cheap to make and to convert this compound into the desired antibiotic (which is another compound) using a series

of chemical reactions that occur inside microbes. If you wonder why microbes are involved, know that several naturally occurring antibiotics are produced by microbes.

Here is how the science works. A given species of microbe has the ability to synthesize specific compounds. The microbe does so using reactions that convert one compound into another. For example, in Figure 1, where the arrow sign means conversion, the red microbe can convert the compound c_1 into the compounds c_2 or c_3 (the first two reactions). It can also convert the compound c_2 to c_4. A microbe can also chain together reactions. Therefore, if the red microbe is provided the compound c_1, then it can make c_2, and thereafter use c_2 to make c_4. These chains of reactions can be arbitrarily long, as long as all the reactions can take place in that microbe.

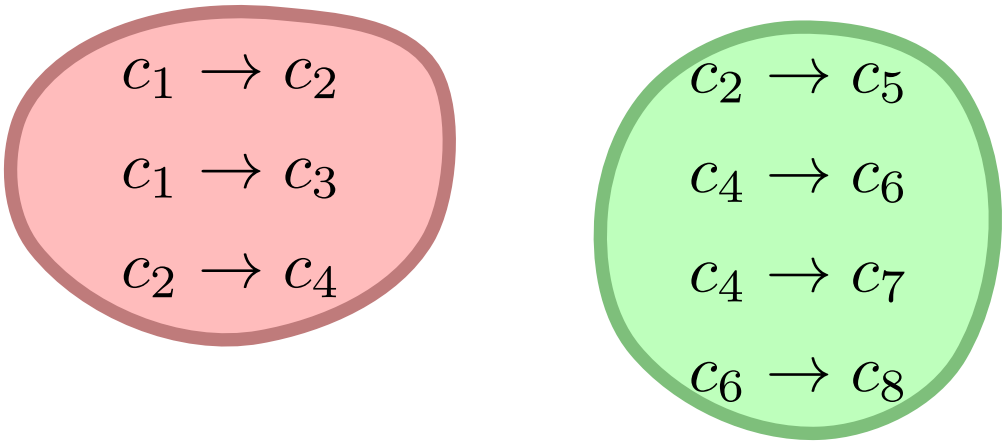


Figure 1: An illustration of microbes and reactions. The red microbe can perform three reactions and the green microbe is capable of four reactions. Starting from c_1, these two microbes can together make c_8 only if c_4 can be secreted by the red microbe and absorbed by the green microbe.

The difficulty is that different microbial species can perform different sets of reactions and hence produce different sets of compounds. For instance, the only reaction that can use c_4 to form another compound exists in the green microbe. If we started with c_1 in the red microbe, produced c_4 in this microbe, and then somehow managed to "ship" c_4 to the green microbe, then we could obtain the compounds c_6, c_7, and c_8 (via c_6). And c_8 may be the antibiotic we desire!

How can we give the green microbe some c_4? Fortunately, biology comes to our rescue once again. Microbes can secrete some compounds to the environment and can absorb some compounds from the environment. For example, if c_4 is such a compound, then (starting from c_1), the red microbe can make and secrete c_4. Subsequently, the green microbe can absorb c_4 and use it to make c_6, c_7, and c_8.

There are only two more rules. Each reaction takes some time to complete; different reactions can have different times. Secretion and ingestion of a compound also take time; these times vary from one compound to another.

What your company gives you as input are the following:

(i) the names of the microbes $M_1, M_2, \ldots M_k$,

(ii) the set C of compounds,

(iii) for each microbe M_i, where $1 \leq i \leq k$, a set S_i of reactions and their times,

(iv) the subset $S \subseteq C$ of compounds that can be secreted and absorbed, and the secretion and absorption times for each such compound,

(v) a source compound $\sigma \in C$ and a target compound $\tau \in C$.

To be clear about item (iii), for each microbe M_i, a reaction in S_i is of the form (c, d, t). This triple represents a reaction where compound $c \in C$ is converted to compound $d \in C$ in time t units in the microbe M_i. The sizes of the sets S_i can vary from microbe to microbe.

Finally, we can state the problems the company asks you to solve:

1. Starting from compound σ, can the microbes produce the target compound τ? If the answer is yes, which set of reactions accomplishes this task in the least amount of time?
2. Solve the first problem in the special case when every reaction in every microbe takes one unit of time and for every compound in S, the total secretion and absorption time is two units.

Note: The problem is not difficult to solve using techniques you have learnt so far in this course. I am looking for clear descriptions of the approach and careful attention to the running time. There are many elements in the input. You have to decide what factors will govern the size of the input and, therefore, the running time of your algorithm. For part 2, I am expecting a faster algorithm than for part 1.

Problem 4 (20 points) You are given a list of n real numbers (the list can contain both positive and negative numbers). Give an efficient algorithm to determine the contiguous sub-list with the largest sum. More formally, suppose the numbers are $l_1, l_2, \ldots, l_{n-1}, l_n$. Your mission, should you choose to accept it, is to compute two indices $1 \leq i \leq j \leq n$ such that

$$s(i,j) = \sum_{k=i}^{j} l_k$$

is the largest over all possible choices of i and j. Note that to compute $s(i,j)$ we sum up all the numbers between indices i and j inclusive.

Hint: I am looking for an $O(n \log n)$ time algorithm. It is possible that you have seen this problem before and are aware of a solution with an $O(n)$ running time. If you present this algorithm, it is even more essential than ever that you prove its correctness. I will not award any points for an $O(n)$ algorithm without a proof of correctness.

Problem 5 (25 points) After graduating from Virginia Tech, you start working at a company that makes games for phones. You are assigned to work on a game called PoodleJump. The goal here is for a player to move a character (a Poodle, to be precise) so that it jumps from one ledge to another. The position of a ledge is specified by its x-coordinate and its z-coordinate. The Poodle can only jump to the left. A jump is *good* if the poodle jumps to the left and if the ledge l that the poodle lands upon has the following property: every ledge to the right of ledge l, i.e., with x-coordinate larger than that of l, is below l, i.e., has a smaller z-coordinate than that of l; Otherwise, the jump is bad. See Figure 2. The poodle starts at the rightmost ledge, i.e., the one with the largest x-coordinate. The game ends as soon as the Poodle makes a jump that is bad. Given a set of n ledges (and their positions), devise an efficient algorithm to compute the largest number of good jumps a player can make and the sequence of ledges that comprise these jumps.

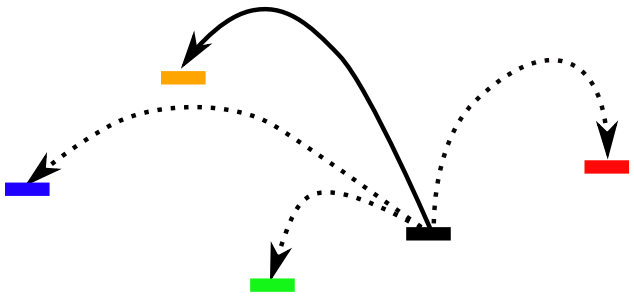


Figure 2: An illustration of good and bad jumps. The jumps from the black ledge to the red, green, and blue ledges are bad: red because the poodle jumps to the right; green because the black ledge is to the right of and above the green ledge; and blue because the orange ledge is to the right of and above the blue ledge. The only good jump from is from the black ledge to the orange ledge.