CSL356 Aug 4,5,7

TUTORIAL SHEET 1

1. Given two positive numbers a and b, use Euclid's algorithm to find integers s, t such that $s \cdot a + t \cdot b = GCD(a, b)$. Prove the correctness of your algorithm by induction.
   Solution: We use induction. We we call $GCD(a, b)$, and assuming $a \geq b$, two cases arise: (i) b divides a: if $a = \alpha b$, then $s = 1, t = (1 - \alpha)$, (ii) b does not divide a: let r denote the remainder when a is divided by b. By induction/recursion, there are constants s', t' such that $s'b + t'r = GCD(b, r) = GCD(a, b)$. Now substitute $r = a - \alpha b$ for some positive integer α, to find the desired s and t.

2. (KT-Chapter 1) Decide whether the following statement is true or false: "In every instance of the stable matching problem, there is a stable matching containing a pair (m, w) such that m is ranked first in the preference list for w, and w is ranked first in the preference list of m".
   Solution: There may not even exist such a matching. For example suppose there are two men m_1, m_2 and two women w_1, w_2. The list of m_1 is w_1, w_2, and that of m_2 is w_2, w_1. The list of w_1 is m_2, m_1 and that of w_2 is m_1, m_2.

3. Give an instance of the stable matching problem for which the algorithm discussed in class takes $\Omega(n^2)$ time.

   Solution: The men have identical rankings of all women (the ranking list of women can be arbitrary.

4. (KT-Chapter 1) Gale and Shapley published their paper on the stable marriage problem in 1962; but a version of their algorithm had already been in use for ten years by the National Resident Matching Program, for the problem of assigning medical residents to hospitals.

   Basically, the situation was the following. There were m hospitals, each with a certain number of available positions for hiring residents. There were n medical students graduating in a given year, each interested in joining one of the hospitals. Each hospital had a ranking of the students in order of preference, and each student had a ranking of the hospitals in order of preference. We will assume that there were more students graduating than there were slots available in the m hospitals. The interest, naturally, was in finding a way of assigning each student to at most one hospital, in such a way that all available positions in all hospitals were filled. (Since we are assuming a surplus of students, there would be some students who do not get assigned to any hospital.) We say that an assignment of students to hospitals is stable if neither of the following situations arises.

   - **First type of instability:** There are students s and s', and a hospital h, so that (i) s is assigned to h, (ii) s' is unassigned, and (iii) h prefers s' to s.

– **Second type of instability:** There are students s and s', and hospitals h and h', so that (i) s is assigned to h and s' is assigned to h', (ii) h prefers s' to s, and s' prefers h to h'.

So we basically have the stable marriage problem, except that (i) hospitals generally want more than one resident, and (ii) there is a surplus of medical students. Show that there is always a stable assignment of students to hospitals, and give an efficient algorithm to find one. The input size is $\theta(mn)$; ideally, you would like to find an algorithm with this running time.

Solution: The algorithm is similar to the stable matching algorithm. The hospitals propose to the candidates in order of preference. A candidate, if not already assigned or assigned to a less preferable hospital, accepts the proposal; otherwise rejects it. There can be at most mn proposals, and so we are done. Check that all of the operations can be implemented using arrays.

A common pitfall is that if candidates propose to hospitals, then hospitals need to check if they have a less preferable candidate. Since a hospital can have large number of positions, it is not clear how to perform this operation in constant time.

5. (KT-Chapter 4) Let us consider a long, quiet country road with houses scattered very sparsely along it. (We can picture the road as a long line segment, with an eastern endpoint and a western endpoint.) Further, let's suppose the residents of all these houses are avid cell phone users. You want to place cell phone base stations at certain points along the road, so that every house is within 4 kilometers of one of the base stations. Give an efficient algorithm that achieves this goal, using as few base stations as possible. Prove the correctness of your algorithm.

   Solution: There is a simple greedy algorithm here. Let h denote the left-most house. Then we place a base station 4km to the right of h. Now remove all houses which are covered by this base station, and repeat. It is easy to show that if the algorithm locates base stations at $b_1, \ldots, b_k$, and some other algorithm (which could be the optimal algorithm) places base stations at $b'_1, \ldots, b'_{k'}$ (from left to right), then $b_1 \geq b'_1, b_2 \geq b'_2$, and so on. Therefore $k \leq k'$.

6. (KT-Chapter 4) Consider the following variation on the Interval Scheduling Problem from lecture. You have a processor that can operate 24 hours a day, every day. People submit requests to run daily jobs on the processor. Each such job comes with a start time and an end time; if the job is accepted to run on the processor, it must run continuously, every day, for the period between its start and end times. (Note that certain jobs can begin before midnight and end after midnight; this makes for a type of situation different from what we saw in the Interval Scheduling Problem.)

   Given a list of n such jobs, your goal is to accept as many jobs as possible (regardless of their length), subject to the constraint that the processor can run at most one job at any given point in time. Provide an algorithm to do this with a running time that is polynomial in n, the number of jobs. You may assume for simplicity that no two jobs have the same start or end times.

Example: Consider the following four jobs, specified by (start-time, end-time) pairs: (6 pm, 6 am), (9 pm, 4 am), (3 am, 2 pm), (1 pm, 7 pm). The unique solution would be to pick the two jobs (9 pm, 4 am) and (1 pm, 7 pm), which can be scheduled without overlapping.

Solution: This is like the interval scheduling problem discussed in class except that jobs are periodic. One way of thinking about this is that time is not like an interval, but is a circle (think of it as the circle in a clock). Now, each job corresponds to an interval in this circle. Now, suppose we *knew* the job which are being done at midnight – call this j. Then, we could remove all jobs overlapping with j, and solve the remaining problem. The remaining problem looks like the interval scheduling problem done in class (and so can be solved using a greedy algorithm). However, we do not know the job j. One way out is to try out all possibilities for such a job: for each job which contains the mid-night time, we select it and solve the resulting interval scheduling problem. Finally, we pick the best such solution.