CPSC 320 Midterm #1 Sample Solution

February 4, 2015

1 A Capital Idea [11 marks]

1. In each row below, circle the correct statement if we know that for all positive integers n, $f(n) < g(n)$. **[8 marks]**

   SOLUTION: We can easily prove $f(n) \in O(g(n))$ by setting our $c = 1$ and our $n_0 = 1$.

   If $f(n) = 1$ and $g(n) = n + 1$, then $f(n) < g(n)$ for positive n, but $f(n) \notin \Omega(g(n))$. However, if $f(n) = 1$ and $g(n) = 2$, then we **also** have $f(n) < g(n)$, but $f(n) \in \Omega(g(n))$. So, we don't know about Ω (and unless we knew O was false, we also wouldn't know about Θ).

   We don't know about o either. The Ω examples establish that. (In the first example, $f(n) \in o(g(n))$, but in the second example, that's not true.)

   We **do** know about ω. It cannot be the case for **all** constants c that $f(n) > c \cdot g(n)$ for sufficiently large n because it's not true for $c = 1$. So, $f(n) \notin \omega(g(n))$.

2. Consider the following pseudocode:

```
Given an array A with length n:

Let TestCount = 0
For each index i from 1 to the length of A:
  For each index j from 1 to the length of A:
    If UnknownTest(A, A[i], A[j]) returns true:
      Increment TestCount
Halt and return TestCount
```

   If you're given no further information about `UnknownTest`, what can you say about the **asymptotic worst-case runtime** of this function? **Briefly** justify your answer. **[3 marks]**

   SOLUTION: The outer loop runs exactly n times. The inner loop runs exactly n times. That means the inner loop's body is executed n^2 times. We don't know how long it takes to execute (maybe `UnknownTest` is really slow asymptotically?), but it has to take at least constant time just for loop maintenance, handling the `if`, calling the `UnknownTest` function, etc.

   So, we can only give a lower-bound (i.e., an Ω bound). Since all the other steps besides the loop take constant time, that bound is $\Omega(n^2)$.

   Note: just annotating the loops, saying $\Omega(n^2)$ and saying "because `UnknownTest` may take more than constant time" would have been plenty.

2 Demi-Glace [15 marks]

The minimum spanning tree problem becomes somewhat strange in the presence of negative edge weights. Imagine, for example, that you are a telecommunications company creating a communications network by connecting particular cities with fiber-optic cable. You want to ensure that all cities are connected by some path (i.e., that you've created a spanning tree). There is a cost to laying the cable, but some pairs of cities are also willing to pay you to do the job; so, the **net** cost of a particular connection may be positive, zero, or even negative.

1. An **apparently unrelated** problem: We have a graph G with 25 nodes and 150 edges. We compute a spanning tree T of G with total cost 100 on its edges. We then add 2 to the cost of **every edge** in G. Explain why the new cost of T is 148. *Hint:* Draw a small example to gain insight! **[3 marks]**

   SOLUTION: Every tree in a graph with $|V|$ nodes has exactly $|V| - 1$ edges; so, T does as well.

   We increased each edge's cost by 2; so, we increased T's overall cost by $2 * (|V| - 1)$.

2. Back to the problem above: Assuming that you insist on producing a fiber-optic network representing a minimum spanning **tree**, give a reduction from this problem to the problem of computing a minimum spanning tree over a graph with strictly **positive** edge weights. **[6 marks]**

   Remember: a reduction is two algorithms, not just one. *Hint:* the previous problem is **not** unrelated.

   SOLUTION: To reduce an instance of the any-weight problem to the positive-weight problem, we find the weight $-e$ of the lowest-weight edge and add $e + 1$ to all edges' weights. (At that point, the lowest-weight edge has weight $-e + e + 1 = 1$, and all other edges have higher weight.)

   To "restore" a solution from the positive-weight problem to the any-weight problem, we just give back the same edges. (If we need to "fix" the weights, we just subtract $e + 1$ from each one.)

   We cut this problem from the exam, but only for time balance and because we didn't want too many interdependent problems, not because it was a bad problem: Find a bound on the worst-case performance of your reduction (not including the time to solve the underlying problem) in terms of $|V|$ and $|E|$. That's a **good** problem to solve!

3. Finish this proof that your reduction—paired with an optimal solution to the positive edge weights MST problem—produces a **minimum** spanning tree. (Note: you may assume that your reduction produces a spanning tree, just not necessarily a **minimum** spanning tree.) **[6 marks]**

   Proof: Assume for contradiction that my reduction (paired with the optimal solution to the positive edge weight MST problem) produces a spanning tree T with cost c_T in the original graph but some other spanning tree U exists with cost $c_U < c_T$ in the original graph.

   Note: You're free to continue as you like from there, but if you want to also use the following, just circle it: "T's cost in the graph produced by my reduction is c'_T, and U's cost in the graph produced by my reduction is c'_U. We can compute c'_T and c'_U as follows..."

   SOLUTION: We'll use that quote above and continue with:

   ... $c'_T = c_T + (|V| - 1)b$ and $c'_U = c_U + (|V| - 1)b$, where b is the constant we added to each edge's weight. That's because each tree has exactly $|V| - 1$ edges; so, adding b to all edges' weights adds exactly $(|V| - 1)b$ to the trees' costs.

   We know T is optimal in the new graph. So, $c'_T \leq c'_U$. But then $c'_T - (|V| - 1)b \leq c'_U - (|V| - 1)b$ (subtracting the same quantity from both sides), and so $c_T \leq c_U$. That contradicts $c_U < c_T$.

   QED

3 Greedy (Yet Plausible) Deniability [18 marks]

You're solving the interval scheduling problem except **minimizing** the number of jobs performed rather than maximizing it. In particular, we define a *conflict set* to be the set of all jobs that conflict with a particular job. Your solution should minimize the number of jobs performed while still performing exactly one job from each conflict set.

UNECESSARY FLAVOR TEXT: Your boss has just given you a list of jobs to perform. Each job has a start time and an end time. You can never do more than one job at a time. You're kind of tired; so, you'd like to do as few jobs as possible, but you can't just do **nothing** or you'll get fired. So, you want to find a list of the smallest number of jobs you can do so that every **other** job conflicts with (has times that overlap) at least one of the jobs you **are** doing.

1. Here is a greedy strategy that does **not** always choose the smallest number of jobs: (1) Sort the jobs in order by increasing **start** time. (2) Repeat until there are no jobs left: choose to do the first job left and then cross out it and all jobs with which it conflicts.

   Now, explain which jobs are in the conflict set of the job j with the first start time and where those jobs appear in the array that is sorted by start time. **[2 marks]**

   SOLUTION: The conflicting jobs are all those with start time $\leq j$'s finish time (and start time $\geq$ j's start time, but since j is first, that's every job). Since the jobs are sorted by start time, all of those jobs are at the start of the array.

   We initially developed this problem thinking that this led to a greedy solution, but oops, it doesn't. However, the fact that those jobs are at the start of the array does make the implied solution nice and efficient :).

2. Draw an example with **four jobs** that shows that this broken greedy strategy can succeed. Indicate what the strategy selects. **[1 mark]**

   SOLUTION: I am but a novice at drawing in LaTeX. So, I'll just say that this is a pretty trivial problem. For example, if all four jobs conflict, this algorithm wins (picks one). If none of the four jobs conflicts (i.e., they're one after the other), this algorithm also wins (picks all four).

3. Draw an example with **four jobs** that shows that this broken greedy strategy can fail. Indicate (1) what the strategy selects and (2) what the optimal solution is. **[2 marks]**

   SOLUTION: See above. Only three jobs are really necessary to illustrate the problem. Have the first job conflict with the second but **not** the third. Have the second job conflict with both the first and third. (So, draw the second job spanning from before the end of the first to after the start of the third.) Then, this algorithm picks the first, crosses off the second, and picks the third, but the optimal solution picks only the second.

   We need a fourth job to be responsive to the problem. To keep things simple, we'll have it start after all the other jobs finish, and both greedy and optimal must pick it.

4. Someone proposes a **new** greedy algorithm. They have **already** proven: "For any problem instance, an arbitrary optimal solution $\mathcal{O}$ **must** include at least one job c in common with the solution $\mathcal{G}$ created by their greedy algorithm."

   Here is a theorem that might be part of their proof of correctness: "Consider a new problem instance that is the same as the original one except that it excludes everything in c's conflict set. We now show that $\mathcal{O}$ with c removed (i.e., $\mathcal{O} - \{c\}$) is an optimal solution to this new problem instance."

   Finish these steps on the way to proving this theorem:

   (a) First, we show that $\mathcal{O} - \{c\}$ does not include two conflicting jobs. We know $\mathcal{O}$ itself does not include two conflicting jobs, and... **[2 marks]**

SOLUTION: $\mathcal{O}-\{c\}$ has no additional jobs beyond $\mathcal{O}$. So, $\mathcal{O}-\{c\}$ cannot have two conflicting jobs.

(b) Next, we show that $\mathcal{O}-\{c\}$ includes at least one job from each conflict set. Assume for contradiction that $\mathcal{O}-\{c\}$ does not include a job from the conflict set of some job in the new instance. That job also appears in the original instance; so, ... **[5 marks]**

SOLUTION: $\mathcal{O}$ includes some job from this conflict set. Since $\mathcal{O}-\{c\}$ doesn't include any such job, the job must be the only one that differs, c. But, we removed all jobs that conflict with c from the new instance. So, the job cannot have c in its conflict set, which is a contradiction.

QED

(c) Finally, we show tht $\mathcal{O}-\{c\}$ is optimal for the new problem instance. Assume for contradiction that it is not. Then, some other solution $\mathcal{O}'$ exists that uses fewer jobs to solve the new instance, but... **[6 marks]**

SOLUTION: $\mathcal{O}' \cup \{c\}$ must then be a solution to the original instance, since it includes something in the conflict set of everything that conflicts with c (i.e., c) as well as something in every other conflict set (i.e., the members of $\mathcal{O}'$).

However, $|\mathcal{O}' \cup \{c\}| = |\mathcal{O}'| + 1 < |\mathcal{O} - \{c\}| + 1 = |\mathcal{O}|$ and thus $\mathcal{O}' \cup \{c\}$ is a better solution than $\mathcal{O}$, but that contradicts the fact that $|\mathcal{O}|$ is optimal.

QED

4 Marriage Counselling [6 marks]

In this problem, we consider the Gale-Shapley algorithm with men proposing. For each statement, circle **one** answer to indicate whether the statement is **always** true, **never** true, or **sometimes** true (i.e., true for some instances but not for others).

Note: in some cases we restrict attention to just certain types of instances, in which case we're asking whether the statement is always, never, or sometimes true for instances **of that type**.

WARNING: one version of the exam asked slightly different questions below.

1. The last proposal made is to a woman who was previously engaged.

   SOLUTION: This must be false. The last proposal ends the loop. Thus, the proposal must be accepted (else the man would still be free and the loop would continue), and the woman cannot accept it and "dump" a former fiancé (else that fiancé would be free, and the loop would continue). So, the woman was not previously engaged. (Note, the fact that she wasn't engaged **at that moment** does indeed indicate that she was never previously engaged, since women in the algorithm have no mechanism to become free after becoming engaged.)

2. Each man makes only one proposal.

   SOLUTION: This can be true. Consider the trivial example where there's one man and one woman. It can also be false, consider a case with two men and women. Both men prefer the same women w_1, but only one can marry her. The other must propose to w_1 (and be rejected or later have the engagement broken) before proposing to w_2.

3. For any instance in which two women w_1 and w_2 both most prefer one man m, the one that m prefers **less** of these women does **not** marry m.

   SOLUTION: This is always true. Assume for contradiction that it were false. Then, m marries w_2, but m prefers w_1 to w_2, and since w_1 most prefers m, she prefers m to whomever she may have married. That's an instability, but the algorithm generates a stable solution. QED

5 Pairs of Apples and Oranges [10 marks]

For each of the following, indicate the most restrictive true answer of $f(n) \in O(g(n))$, $f(n) \in \Theta(g(n))$, and $f(n) \in \Omega(g(n))$. **NOTE:** there are only three options here (only the "big" bounds, not the "little" ones).

SOLUTION: Inline below..

$\frac{\lg n}{\log n}$	$\in \Theta$ (	2^{1000}	)
$n^3 - 2n^2 + 7$	$\in \Omega$ (	$5n^2 + 53$	)
$\frac{n^2}{\lg n}$	$\in \Omega$ (	$n \lg n$	)
$(n!)^2$	$\in O$ (	$(2n)!$	)
$(\lg 4)^n$	$\in \Omega$ (	$(\lg 3)^n$	)

Some brief justifications.

For the first, $\lg n$ is some constant times $\log n$. Thus, their ratio is also a constant, and both sides are constants.

The second should be clear.

Take the ratio for the third with the left side on top, and you get: $\frac{n^2}{(\lg n)n(\lg n)} = \frac{n}{(\lg n)^2}$. Linear time dominates any power of a log. You can verify that with derivatives without **too** much trouble by changing $\lg n$ to $\ln n$ to make your life easier! (You'll end up with a new ratio where both top and bottom approach infinity. Just take the derivative of the top and of the bottom again.)

For the fourth, if we take the ratio, one of the $n!$'s cancels and leaves a product of ratios like $2n/n$, $(2n-1)/(n-1)$, etc. Each of these is **at least** 2; so, multiplied together, they're at least 2^n, which goes to infinity.

For the last, this is just two exponentials with different bases. The one with the smaller base is dominated by the other.

Note: in **these** examples, every one that says O is also o and every one that says Ω is also ω.

6 Bonus [Up to 7 Bonus Marks]

Bonus marks add to your exam total and also to your course bonus total. What course bonus points are worth still isn't clear, however! **WARNING**: These questions are too hard for their point values. We are free to mark these questions harshly. Finish the rest of the exam before attempting these questions. Do not **taunt** these questions.

1. In the practice exam problems, we found that the independent set of a graph G with n vertices where the maximum degree of any vertex is $d_{\max}$ is lower-bounded by $\lceil \frac{n}{d_{\max}+1} \rceil$. Give an algorithm that, given an n and a $d_{\max}$, constructs a graph with the given parameters whose largest independent set has exactly $\lceil \frac{n}{d_{\max}+1} \rceil$ nodes. (I.e., show that this lower-bound is tight.) You must clearly explain why the graph produced has the appropriate sized independent set. **[2 bonus marks]**

   SOLUTION: All we'll do for the moment is give two strong hints. (1) There's no need for the graph to be connected. (2) At most one member of a clique can be in an independent set. Don't know what a clique is? It's a bonus, my friend: http://lmgtfy.com/?q=clique.

2. The **minimal** interval scheduling problem above suggests the following greedy approach: "Sort the jobs by start time and repeat until there are no jobs left: Of the jobs that conflict with the first job, choose the one with the last finish time and cross out all jobs that conflict with it."

Prove or disprove that this strategy is optimal. **[2 bonus marks]**

SOLUTION: All we'll do for the moment is to say that the strategy most definitely is not always optimal.

Now, if you have an efficient, greedy, optimal solution, post about it! Alternatively, if you have a lower-bound on the complexity of the problem, post about that, too! We're not sure but suspect it may be NP-complete (whatever that means... we haven't studied it yet!).

3. In this problem you'll prove a tight, non-asymptotic upper bound on the number of iterations of the loop in G-S in terms of the number of men n in the input. (1) Give and prove the upper bound. (2) Give an algorithm to produce an instance of size n that achieves that bound and show that it does so, which establishes that the bound is tight. **[3 bonus marks]**

   Note: we will give 1 bonus mark for the first part for a **tight** bound, even if you do not prove that it's tight (i.e., finish the second part).

   SOLUTION: All we'll do for the moment is give you a strong hint towards the bound: The last proposal must be to a woman who was previously unengaged. So, she can only ever receive a single proposal. (Perhaps surprisingly, it **is** possible to achieve the bound that suggests.)