CPSC 320 Sample Midterm 1
February, 2016

1. [20 points] Short Answers

   (a) [4 points] Consider the Gale-Shapley stable matching algorithm. Is it possible for every woman to get the worst partner on her list? If so, what preference lists would result in such outcome?

   Solution: Yes, it is. It suffices for every man to have a different first choice (in which case pairing men with their first choice is stable, since none of them has a reason to switch), and for each man's first choice to be a woman who ranks him last.

   (b) [8 points] Rank the following functions by order of growth. Use $f \ll g$ if $f \in o(g)$ and $f \approx g$ if $f \in \Theta(g)$.

   $$n \log n \qquad \log^3 n \qquad 2^{\log_2 n} \qquad n^2/\log n$$

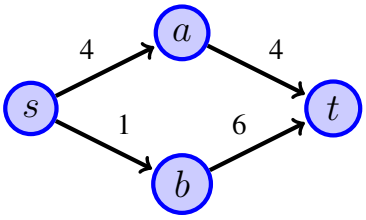
   You **don't** need to justify your answer.

   Solution: $\log^3 n \ll 2^{\log_n} = n \ll n \log n \ll n^2/\log n$

   (c) [4 points] Let P be a shortest path from s to t in a directed graph $G = (V, E)$ with distance function $\ell : E \to \mathbb{R}^+$. Let $\ell' : E \to \mathbb{R}^+$ is defined as follows: $\ell'(e) = [\ell(e)]^2$ for every $e \in E$. **Decide whether the following statement is true or false (justify your answer shortly).**

   Statement: *P is always a shortest path from s to t in G with distance function ℓ'.*

   Solution: *False*. Consider G, ℓ in the following example:

   The shortest path from s to t is s, b, t with the cost 7, as the only alternative path has distance 8. In G, ℓ', s, b, t is longer (37) than alternative path s, a, t (32).

   (d) [4 points] In the decision tree build for an algorithm $\mathcal{A}$ solving a problem $\mathcal{P}$ on inputs of size n,

   (i) what is the number of children (outgoing edges) of each internal node in the tree?

   (ii) what can we say about the number of leaves of the tree with respect to the problem $\mathcal{P}$?

   Solution:

   (i) The number of children corresponds to the number of answers each query/test on the input can give.

   (ii) The number of leaves is (at least) the number of possible outcomes of the problem $\mathcal{P}$.

2. [8 points] Consider three functions $f, g, h : \mathbb{N} \to \mathbb{R}^+$, where $f(n) \in \Omega(g(n))$.

   What can we say about asymptotic relation between $f(n)h(n)$ and $g(n)h(n)$? **Justify** your answer (showing an example of functions that satisfy this relation does not count).

Solution: We have $f(n)h(n) \in \Omega(g(n)h(n))$.
Proof using definitions: We have $f(n) \geq cg(n)$ for all $n \geq n_0$ for some constants c, n_0. Then $f(n)h(n) \geq cg(n)h(n)$ for $n \geq n_0$.

Proof using limits:

$$\lim_{n\to\infty} \frac{f(n)h(n)}{g(n)h(n)} = \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \text{ or a constant}$$

It follows that $f(n)h(n) \in \Omega(g(n)h(n))$.

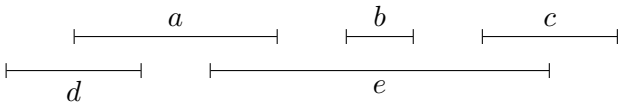
3. [24 points] Interval Scheduling.

   (a) [6 points] Consider interval scheduling problem from the class (selecting the maximum number of compatible intervals from set $\mathcal{I}$). Alex has proposed a greedy algorithm (different from the one considered on the lecture) that attempts to solve this problem. In attempt to prove correctness of his algorithm, Alex proved the following claim:

   > **Claim.** Let $\mathcal{G} = \{I_1, \ldots, I_k\}$ be a greedy solution produced by Alex's algorithm and $\mathcal{O} = \{J_1, \ldots, J_m\}$ with $k \leq m$ be an optimal solution. Both sequences are sorted by the starting times. Then
   > - for every $x = 1, \ldots, k$, $s(I_x) \leq s(J_x)$;
   > - $\mathcal{G}$ is a compatible subset of $\mathcal{I}$; and
   > - every interval in $\mathcal{I}$, overlaps at least one interval in $\mathcal{G}$.

   Can this claim be used to show that $\mathcal{G}$ is optimal? **Either prove it or give a counterexample.**

   Solution: The Claim does not imply that the greedy solution is optimal (has the maximum number of elements).
   Counterexample: Consider the following instance of the problem:

   Assume that the greedy solution is $\mathcal{G} = \{d, e\}$, and an optimal solution is $\mathcal{O} = \{a, b, c\}$. Then intervals in $\mathcal{G}$ are compatible and satisfy the Claim (d starts before a, e starts before b), yet the greedy solution is not optimal.

   (b) [12 points] Consider the following interval scheduling problem:

   > Your boss has just given you a list of n jobs to perform. Each job J has a starting time $s(J)$ and finishing time $f(J)$. You can never do more than one job at a time. You are tired, so you would like to do as few jobs as possible, but cannot just do "nothing". More precisely, you want to find a list of the smallest number of jobs you can do so that every **other** jobs overlaps with at least one job on your list (so the boss cannot say that you can add another job to your list).

   Propose a reasonably efficient greedy algorithm. Your algorithm **does not need** to always return the smallest list, but it should make a good attempt it. Write a pseudocode for your algorithm.

   Solution: First, note that this is different from the Interval Scheduling problem from the class (since we want to select a smallest size set of intervals), and it's also different from the safety committee problem from the assignment, since the jobs you are selecting cannot overlap (while committee members could have overlapping shifts).

There are many greedy algorithms that we can design. For example, we could use the algorithm proposed in Question 3(a) of Assignment 2, or we could use the following algorithm:

(1) Compute for each job the number of conflicting jobs.

(2) Sort the jobs by decreasing number of conflicts.

(3) Repeat: until there are no jobs left: choose to first job in the list, add it to the solution and then remove it and all jobs it conflicts with from the sorted list.

Comment: None of these algorithms is producing an optimal solution (smallest set of jobs). Can you find a counterexample for each of these two algorithms?

(c) [6 points] **Analyze the time complexity** (find a good asymptotic upper bound on the *worst-case* running time) of your algorithm from part (b). Specify only as much of your implementation (for instance, data structures used) as needed for the analysis. Justify your answer!
Remarks: You don't need to show that your bound is tight. Remember you are not going to be judged by how low your upper bound is, but by whether it's correct and by how good it is. For instance, if your algorithm requires time $O(n^3)$, but you will claim its running time is in $O(n \log n)$, you will most likely get 0 points. But also if it works in time $\Theta(n \log n)$, and your bound is $O(n^3)$, you will not get many points either.

Solution: For the above algorithm:

(1) Check each pair of jobs, whether they conflict and increase the count for each if they do. This can be done in $O(n^2)$ time.

(2) Sorting can be done $O(n \log n)$.

(3) Processing the list will take $O(n)$ iterations. In each iteration we need to traverse the remainder of the list to mark conflicting jobs as deleted (which are then ignored when picking jobs from the list), which takes time $O(n)$. Hence, the total time in this step is $O(n^2)$.

The total time is in $O(n^2)$.

4. [10 points] **Optimal caching.**

(a) [4 points] In case 3 of the proof of the theorem that we later used to show that the greedy caching strategy is optimal, we assumed that S_{FF} evicts an element e from the cache, whereas S_j ejects a different element f. We then stated that one of three things **must** happen before a request for e comes along:

- A request for f, where S_j evicts e.
- A request for f, where S_j evicts an element other than e.
- A request for some other element x, where S_j evicts e.

Why must one of these situations occur before any request for e?

Solution: Since FF algorithm ejects e for request d_{j+1}, e is the element that is accessed again furthest in the future in the following requests. So f will be requested before e.

(b) [6 points] Consider the following online algorithm for caching:

Least-Recently-Used (LRU): Evict the least recently used item from the cache, when there is a cache miss.

Design an initial state of the cache and a sequence of requests that will require an eviction for each request if we follow the LRU algorithm. Justify why your answer is correct.

Solution: Let assume that the cache initially contains some "old" items $o_1, o_2, \ldots, o_k$ and that the sequence of requests contains the following items $i_1, i_2, \ldots, i_{k+1}, i_1, i_2, \ldots, i_{k+1}, \ldots$. During the first k steps, items $o_1, \ldots, o_k$ are evicted and $i_1, \ldots, i_k$ are brought in to the cache. In each following step we evict exactly the item that is needed in the next step (since it's the least recently used item).

5. [8 points] (a) [2 points] Explain what's the different between a walk and a path in a graph.

Solution: A path is a walk that cannot repeat any nodes.

(b) [6 points] Analyze the running time of the following algorithm:

```
1:  function DIJKSTRA(G = (V, E), ℓ, s)
2:      S ← {s}                                          ▷ explored nodes
3:      d(s) ← 0                                         ▷ distance from s
4:      while S ≠ V do
5:          for all v ∈ V \ S do
6:              d(v) ← min_{u∈S: (u,v)∈E} [d(u) + ℓ(u, v)]
7:              P(v) ← arg min_{u∈S: (u,v)∈E} [d(u) + ℓ(u, v)]
8:          end for
9:          select u ∈ V \ S with the minimum d(u)
10:         add u to S
11:     end while
12:     return d, P
13: end function
```

in terms of n (the number of nodes) and m (the number of edges). Find a tight asymptotic upper bound.

Solution: There are n iterations of the **while** loop. To store the information which element is in S, we will use a bitmap array, so adding a node to S and checking if a node is in S can be done in constant time ($O(1)$).
In each iteration:

- In lines 5–8, for each vertex v of the graph, the min is calculated for at most $\deg(v)$ values (here, we assume that we check the adjacency list of u and for each neighbor check if it is in S or not, not the other way round). Hence, the time spent on these lines is in $O(\sum_{v \in V} \deg(v)) = O(m)$.
- Line 9 takes $O(n)$ time and line 10 $O(1)$ time.

Since the graph is connected $m \geq n-1$. Hence, the total running time is in $O(n(m+n+1)) = O(nm)$.

Comments: Showing that this bound is *tight* is a bit tricky (and not required, the question was supposed to say "a good asymptotic upper bound"). But here it is: Assume the graph is directed and has an edge from each node to each node, in which case $m = \Theta(n^2)$ and we need to show that the running time is in $\Omega(n^3)$. Then during the i-th iteration of the **while** loop, the time spent in lines 5–7 is in Θ of the number of edges between sets S and $V \setminus S$, which is $i(n-i)$. So, the running time is in $\Omega(\sum_{i=0}^{n-1} i(n-1))$. Now to see that the sum is in $\Omega(n^3)$, consider only part of the sum from $n/4$ to $3n/4$ and lower bound each term with $n^2/4$.

Points total: 70

End of exam.