Sample Solution to Midterm

Problem 1. (3 × 5 = 15 points)

(a) For all monotonically increasing functions $f(n)$ and $g(n)$), does $f(n) = O(g(n))$ imply $2^{f(n)} = O(2^{g(n)})$? Give a proof or a counterexample.

Answer: False. The relationship $f(n) = O(g(n))$ indicates that f's growth rate can exceed that of $g(n)$ by only a constant. For instance, if $g(n)$ is n, then $f(n)$ can be n, $2n$, $100n$, but cannot be n^2 or $n \log n$. A constant-factor blowup, however, when applied to the exponent makes a huge difference. Let $f(n) = 2n$ and $g(n) = n$. Then, we have $f(n) = O(g(n))$. But $2^{f(n)} \neq O(2^{g(n)})$ because

$$\lim_{n\to\infty} \frac{2^{2n}}{2^n} = \lim_{n\to\infty} 2^n = \infty.$$

(b) Solve the following recurrence relation. (Assume that $T(1) = 1$.)

$$T(n) = 2T(n/3) + n.$$

Answer:

$$\begin{aligned}
T(n) &= n + 2T(n/3) \\
&= n + 2n/3 + 4T(n/9) \\
&= n + 2n/3 + 4n/9 + 8T(n/27) \\
&= n(1 + 2/3 + 4/9 + \ldots (2/3)^i + \ldots) + 2^{\log_3 n} T(1) \\
&= n(1 + 2/3 + 4/9 + \ldots (2/3)^i + \ldots) + n^{\log_3 2} \\
&\leq \frac{n}{1 - 2/3} + n^{\log_3 2} \\
&= 3n + n^{\log_3 2} \\
&= O(n).
\end{aligned}$$

Clearly $T(n) \geq n$, so $T(n) = \Omega(n)$. We thus have $T(n) = \Theta(n)$.

An alternative argument, using the recursion tree approach yields a contribution of n for the first level, $2n/3$ for the second, $4n/9$ for the third, and so we see a pattern. Since this is a geometrically decreasing series, the first term dominates the sum, and we obtain $T(n) = \Theta(n)$.

(c) Let G be a undirected graph with distinct weights on the edges. Let T be an MST and (u, v) be an edge of T. Is it true that for any cut C of G that separates u from v, (u, v) is a minimum-weight edge of the cut. Give a proof or a counterexample.

Answer: False. Note that the cut property about MSTs refers to a specific cut, for each edge of the MST. So it does not apply here. Consider a graph with three vertices, u, v, and x, with two edges (u, v) and (u, x) of weights 2 and 1, respectively. There is only one spanning tree, which is also an MST. Consider the cut with $\{u\}$ on one side and $\{v, x\}$ on the other. The edge (u, v) is not the min-weight edge in the cut.

Problem 2. (2 + 8 = 10 points) Chip testing

You have n supposedly identical VLSI chips that are in principle capable of testing each other. Unfortunately, not all chips are good. You have a test machine that tests two chips at a time. If you load chips A and B into the test machine, each chip reports whether the other chip is good or bad. A good chip always reports accurately whether the other chip is good or bad, but the answer of a bad chip cannot be trusted. Thus, for example, if chip A is good and chip B is bad, then A will report accurately that the other chip is bad, but B's answer could be either of "good" or "bad".

(a) Using a single good chip, describe how you can determine all the good chips from your set using at most a linear number of pairwise tests.

Answer: We run $n-1$ pairwise tests, each using the given good chip and one of the $n-1$ other chips. The report given by the good chip identifies the other good chips. Number of pairwise tests done is $n-1$.

(b) Give an algorithm to find a single good chip from among n chips, assuming that more than $n/2$ of the chips are good. Make your algorithm as efficient as you can, in terms of the number of pairwise tests the algorithm conducts. Derive a bound on the number of pairwise tests conducted by your algorithm.

(*Hint:* One approach is to show that at most $n/2$ pairwise tests are sufficient to reduce the problem to one of nearly half the size. One can then use this idea to design a divide-and-conquer algorithm.)

Answer: Let S be the given set of n chips. Let g be the number of good chips and b be the number of bad chips. We group the given set S chips into pairs. If n is odd, then there may be an unpaired chip. Thus, we have $\lfloor n/2 \rfloor$ pairs of two chips each. We load each pair into the test machine and perform the pairwise test.

We will construct a set S' of at most $\lceil n/2 \rceil$ chips such that at least half of them are good. We go through the results of the pairwise tests. If the test result is good-good, then we add to S' any one of the chips and throw the other. Otherwise, we throw out both the chips. Finally, if the number of chips thus included is an even number, then we add the unpaired chip to S'.

Since, at most chip from each pair is included in S', it is clear that the number of chips in S' is at most $\lfloor n/2 \rfloor$. Taking into account the unpaired chip, we obtain $|S'| \leq \lceil n/2 \rceil$. We argue below that the set S' has more than half of its chips as good chips. Using this, we can repeat the process until we are left with exactly one chip, which by definition will be good. We first give the analysis and then a correctness proof (not needed for the exam).

Analysis. Using $\lfloor n/2 \rfloor$ pairwise tests, we have reduced the number of chips to $\lceil n/2 \rceil$ and retained the assumption that more than half of the chips are good. So the total number of pairwise tests we do is given by the recurrence.

$$T(n) = n/2 + T(n/2) = n/2 + n/4 + T(n/4) = O(n),$$

following a calculation very similar to the one in part (b) of problem 1 of this exam.

Correctness proof. We now prove that the number of good chips in S' is more than half of S'. Let gg, bb, and gb be the number of pairwise tests involving two good chips, two bad chips, and one good chip and a bad chip, respectively. (Note that the terms "good" and "bad" in the preceding sentence refer to the actual quality of the chip, not the outcome of the test.) Let x and y denote the number of good chips and bad chips, respectively, that were picked during the pairwise tests. When two good chips pair together, at least one of the chips is included in S'. And the only way a bad chip can be included is when both the chips are bad and report that each is good; in this case, the other bad chip is eliminated. So $x \geq gg$, and $y \leq bb$.

If n is even, we have $gg > bb$, yielding the desired claim $x > y$. If n is odd, then we have $gg \geq bb$, yielding $x \geq y$. In this case, if the number of chips we select in the pairwise tests $(x + y)$ is odd, then $x > y$, and we do not include the unpaired chip, proving the desired claim. On the other hand, if $x + y$ is even, we do add the unpaired chip to S' and have two cases. Either the unpaired chip is good, in which case the number of good chips in S' clearly exceeds the majority. Or the unpaired chip is bad, in which case $x > y$ means $x \geq y + 2$; adding the unpaired bad chip leaves the good chips with a clear majority. This completes the proof.

Problem 3. (2 + 8 = 10 points) Complete paths in directed acyclic graphs

You are given a directed graph G and asked to determine whether there exists a *complete path* in the graph; that is, a path that visits every vertex of the graph *exactly once*.

(a) Give an example of a directed graph that has a complete path. Give an example of a directed graph that does not have a complete path.

Answer: Let $V = \{u, v, w\}$ be a set of three vertices. If we have a graph with two edges (u, v) and (v, w), then we have a complete path $u \to v \to w$. Instead, if we have two edges (u, v) and (u, w), then there is no complete path.

(b) Determining the existence of complete paths in general directed graphs is notoriously hard. The problem is much easier for *directed acyclic graphs*. Give a polynomial-time algorithm to determine whether a complete path exists in a given directed acyclic graph. If such a path exists, your algorithm must also return the path. The more efficient your algorithm is in terms of its worst-case running time, the more credit you will get. Justify the running time of your algorithm.

Answer: Obtain a topological sort of the graph G. Let $v_1, v_2, \ldots, v_n$ denote the sequence of vertices in the topologically sorted order. A hamiltonian path exists if and only if there exists a directed edge (v_i, v_{i+1}) for each i, $1 \leq i < n$. We go through the vertices in sequence and check whether these edges exist. If all of these exist, then the graph has a complete path; otherwise, not.

The topological sort takes linear time. So does the walk through the vertices in topological order and the check for the particular edges. Therefore, the algorithm is linear-time.

Problem 4. (10 points) Subsequences

Comparing two sequences of characters arises in several applications including the life sciences and information retrieval. Suppose you are given two sequences of characters drawn from a common alphabet: S of length m and T of length n, each possibly containing a character more than once. We say that S is a *subsequence* of T if S can be obtained from T by deleting some characters from T. For instance, the sequence A, T, C, T, G is a subsequence of $G, \underline{A}, A, A, \underline{T}, \underline{C}, G, G, \underline{T}, T, \underline{G}$. (Positions indicating the subsequence property are underlined.)

Give an algorithm that determines whether S is a subsequence of T. The more efficient your algorithm is in terms of its worst-case running time, the more credit you will get. Justify the running time of your algorithm.

Answer: We present a greedy algorithm for the problem. Let $S = s_1, s_2, \ldots, s_m$ and $T = t_1, t_2, \ldots, t_n$ be the two sequences. We find the first position j such that $s_1 = t_j$, and then recurse with the sequences $s_2, \ldots, s_m$ and $t_{j+1}, \ldots t_n$. An easy iterative implementation is as follows.

```
let i = j = 1 and M be empty
while i ≤ m and j ≤ n do
        if t_j = s_i
           append t_j to M
           i = i + 1
        endif
        j = j + 1
if i = m + 1
        return M
else
        return "Subsequence not found"
```

Problem 5. (5 points) Shortest paths in directed graphs with both vertex and edge weights

Suppose you are given a directed graph with positive weights on both edges and vertices. The length of a path in the graph is the sum of the weights on the edges and the vertices along the path. Give a polynomial-time algorithm to determine the shortest path from a given source s to all other vertices in the graph.

Answer: There are several ways of reducing the given problem to one in which we have edge weights only (something we know how to solve, using Dijkstra's algorithm or Bellman-Ford's algorithm).

- One reduction is to remove the weights on the vertices and, for all vertices u, increase the weight of every outgoing edge (u, v) by the weight on u. Now, the shortest path from s to t in the original graph is the same as the shortest path from s to t in the new graph. (Note that the path lengths are not quite the same. Why?)
- A different approach is to replace each vertex u by two vertices u_i and u_o, have all edges coming into u point to u_i, all edges coming out of u come from u_o, and finally add an edge (u_i, u_o) with weight the same as the weight of the vertex u. Again, the shortest path from s to t in the original graph is the same as the shortest path from s to t in the new graph (with the same lengths).

The above reduction takes time $O(n + m)$ for a graph with n vertices and m edges. Invoking Dijkstra takes $O(m \log n)$ time. So the total running time is $O(m \log n)$.

Another different approach is to modify Dijkstra's algorithm. In particular, if the weight of an edge (u, v) is $w(u, v)$ and the weight of a vertex u is $w(u)$, all we need to do is to change the line defining $d'(v)$ to

$$d'(v) = \min_{\text{edge } (u,v): u \in S} d(u) + w(u, v) + w(v).$$

Bonus Problem. (6 points) (Splitting a heist)

Two art thieves share a collection of n paintings. They have agreed on a value for each painting, say $x_1, x_2, \ldots, x_n$. They would like to split the paintings into two halves of equal value.

Design an algorithm that determines whether it is possible for the thieves to fairly split the paintings. Your algorithm should run in $O(nT)$ time, where T is the sum of the x_i's. You may assume that T is even.

(*Hint:* Use dynamic programming. One approach is to derive a recurrence for determining whether it is possible to assign paintings to one thief that add up to a value of i, for i from 1 to $T/2$.)

Answer: We will maintain an $n \times T$ matrix S, where $S[i, j]$ is 1 if one of the thieves can get a total value of i using a subset of the paintings 1 through j. We will obtain a recurrence for computing S and use S to determine whether the paintings can be split equally.

We can calculate $S[i, j]$ as follows. First we set $S[i, 0]$ to be 0 for all i. For any other i and j, we set $S[i, j]$ to 1 if either $i = x_j$, or $i > x_j$ and $S[i - x_j, j - 1] = 1$; otherwise, we set $S[i, j]$ to 0.

The desired problem can be solved as follows now. If T is odd, then the answer is trivial: it is not possible to fairly split the paintings. If T is even, then we would like each thief to have a total value of $T/2$. So we check whether $S[T/2, n]$ is 1. If it is, then the answer is yes; otherwise, the answer is no.