University of Washington
Department of Computer Science and Engineering
CSE 421, Fall 2005

December 12, 2005

Practice Final Exam - Answer Key

NAME: ____________________

Instructions:

- Closed book, closed notes, no calculators
- Time limit: 1 hour 50 minutes
- Answer the problems on the exam paper.
- If you need extra space use the back of a page
- Problems are not of equal difficulty, if you get stuck on a problem, move on.
- This practice exam is based on the CSE 521 exam from Spring 1998, some changes have been made to reflect different coverage and content.
- Concise answers are provided. The answers I am providing may be less detailed than necessary for full credit. (I am keeping the answers short, so I don't have to stay up all night typesetting this thing!).

1	/15
2	/10
3	/20
4	/10
5	/25
6	/10
7	/10
Total	/100

Problem 1 (15 points):

Give solutions to the following recurrences. Justify your answers.

a)

$$T(n) = \begin{cases} 3T(n/3) + n^2 & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$

$O(n^2)$, unroll the recurrence to show that the first term is dominant.

b)

$$T(n) = \begin{cases} 4T(n/3) + n & \text{if } n > 0 \\ 1 & \text{if } n = 0 \end{cases}$$

The amount of work for the recurrence is increasing with levels. Unrolling the recurrence shows that the number of terms increases by a factor of 4 at each level. There are $\log_3 n$ levels, so the total amount of work on the bottom level is $4^{\log_3 n} = n^{\log_3 4}$.

c)

$$T(n) = \begin{cases} T(n/3) + 1 & \text{if } n > 1 \\ 0 & \text{if } n = 0 \end{cases}$$

$O(\log n)$, the recurrence just counts the number of level until the base case, which is $\log_3 n$.

Problem 2 (10 points):
Let $G = (V, E)$ be an undirected, bipartite graph. A matching M is said to be *maximal* if every edge of E shares at least one endpoint with an edge of M. Let M_{opt} be a maximum cardinality matching for G.

a) Give an example of a graph G with a maximal matching M_{greed} where $|M_{greed}| = \frac{1}{2}|M_{opt}|$.

The smallest possible example is $V = \{v_1, v_2, w_1, w_2\}$, $E = \{(v_1, w_1), (v_1, w_2), (v_2, w_2)\}$. $\{(v_1, w_2)\}$ is a maximal matching, and $\{(v_1, w_1), (v_2, w_2)\}$ is a maximum cardinality matching.

b) For a bipartite graph G, and maximal matching M_{greed}, prove that $|M_{greed}| \geq \frac{1}{2}|M_{opt}|$.

Proof sketch: Every edge in M_{opt} must be share a vertex with at least one edge of M_{greed} (if not, then M_{greed} is not maximal). Each edge of M_{greed} can share vertices of at most two edges from M_{opt}. Hence $|M_{greed}| \geq \frac{1}{2}|M_{opt}|$.

Problem 3 (20 points):
How can you:

a) Find a maximum weight spanning tree using an algorithm which computes a minimum weight spanning tree.

Negate all edges, find a minimum spanning tree.

b) Find a shortest path in an undirected graph, using an algorithm which finds a shortest path in a directed graph.

Replace all edges by pairs of anti-parallel edges.

c) Find a female optimal stable marriage, given an algorithm for a male optimal stable marriage.

Construct a new instance of the problem, with the M and W sets of vertices interchanged.

d) Find a maximum flow in a graph with capacities on the vertices (meaning a bound on the flow that can go into each vertex), using an algorithm for maximum flow where the capacities are on the edges.

Split each vertex into two vertices, with an edge between them giving the vertix capacity.

Problem 4 (10 points):
Let $G = (V, E)$ be a directed graph with edge costs, K an integer, and s and t vertices of V. Describe an algorithm which finds the most expensive path from s to t that uses exactly K edges. Use a dynamic programming algorithm based on the following recurrence:

$$Opt[k, w] = \max_v (Opt[k-1, v] + cost(v, w)).$$

Problem 5 (25 points):

What is the fastest known algorithm for each of the following problems? Give a short description or citation (no more than two sentences each). What is the run time of the fastest algorithm? (Note: I was looking for the fastest algorithm discussed in class - not the fastest algorithm presented in the theoretical literature.)

1. Determining if an undirected graph with n vertices and m edges is bipartite.

   Use Breadth First Search to divide the graphs into two sets of vertices. $O(n + m)$.

2. Computing the longest common subsequence of a pair of strings each of length n.

   Dynamic Programming. $O(n^2)$.

3. Solving the single source shortest paths problem on a graph with n vertices and m edges.

   Bellman-Ford algorithm. $O(nm)$ time. (Note that nothing is said about the edges being non-negative.).

4. Solving the single source shortest paths problem on an *acyclic* graph with n vertices and m edges.

   Topological sort. $O(n + m)$ (This algorithms wasn't discussed for shortest paths in acyclic graphs - so it probably wouldn't have been a fair question for this year's final.)

5. Checking whether or not an undirected graph with n vertices and m edges has a cycle.

   Use depth first search, and report a cycle if non-tree edges are detected. $O(n + m)$ time.

6. Given n points in the plane, find the closest pair of points.

   Divide and conquer by splitting geometrically, finding closest pairs recursively, and looking for closest "cross border" pair. $O(n \log n)$

7. Solving the knapsack problem with n items, and a bound of K on the size of the knapsack.

   Dynamic programming. $O(nK)$ time.

8. Find a maximum cardinality matching in a bipartite graph with n vertices and m edges.

   Convert to network flow, and use the Ford-Fulkerson algorithm. $O(nm)$ time.

Problem 6 (10 points):

Give short answers to the following questions about network flow:

a) Is the Ford-Fulkerson algorithm a polynomial time algorithm? Why or why not.

   No. In the worst case, the number of phases of the Ford-Fulkerson algorithm depends on the capacities, not on the number of edges and vertices.

   (Note: this issue has not been discussed in class yet - but is likely to be come up in the last week during discussion on NP-completeness.)

b) How do you find the minimum cut of a graph, after a flow algorithm has found the maximum flow?

   Construct the residual graph, and then determine which vertices are reachable in the residual graph by edges of non-zero capacity. This gives one side of the cut. The edges leaving this set are all filled to capacity.

Problem 7 (10 points):

Suppose that you have an algorithm for determining if a graph with vertex demands and edge capacities has a feasible circulation. How would you use this algorithm to find the value of the maximum flow in a graph with edge capacities and a source and a sink.
An algorithm for the circulation problem allows us to test if a flow graph has a flow of a particular value K by assigning a demand of $-K$ to s, and a demand of K to t. We can find a maximum flow by doing a binary search to find the maximum K. This will take $O(\log K)$ calls to the circulation algorithm.