University of New Mexico
Department of Computer Science

Final Examination

CS 362 Data Structures and Algorithms
Spring, 2007

Name:	
Email:	

- Print your name and email, *neatly* in the space provided above; print your name at the upper right corner of *every* page. Please print legibly.
- This is an *closed book* exam. You are permitted to use *only* two pages of "cheat sheets" that you have brought to the exam and a calculator. *Nothing else is permitted.*
- Do all the problems in this booklet. *Show your work!* You will not get partial credit if we cannot figure out how you arrived at your answer.
- Write your answers in the space provided for the corresponding problem. Let us know if you need more paper.
- Don't spend too much time on any single problem. The questions are weighted equally. If you get stuck, move on to something else and come back later.
- If any question is unclear, ask us for clarification.

Question	Points	Score	Grader
1	20		
2	20		
3	20		
4	20		
5	20		
Total	100		

1. **Short Answer**

   (a) Consider the greedy algorithm for activity selection discussed in class. If there is some activity that has finish time earlier than the start time of all other activities, will this activity always be selected by the greedy algorithm? Justify your answer.

   Solution: Yes the activity will always be selected since it conflicts with no other activities.

   (b) Is the minimum weight edge in a graph always in any minimum spanning tree for a graph? Is it always in any single source shortest path tree for the graph? Justify your answers.

   Solution: The minimum weight edge is always in the MST for the graph. You can see this by noting that this edge is always included by Kruskal's algorithm. The minimum weight edge need not be in a single source shortest path (SSSP) tree. Consider the graph consisting of nodes a,b,c where $w(a,b) = 2$, $w(a,c) = 2$, $w(b,c)=1$. Then the SSSP tree rooted at a consists of the edges (a,b) and (a,c)

(c) Assume you have an implementation of the Union-Find data structure that has an amortized cost of $\Theta(\log n)$ for all operations (note that this is a suboptimal implementation of Union-Find). If you use this data structure to implement Kruskal's algorithm on a connected graph with n nodes and m edges, what will be the run time of Kruskal's? Explain your answer. *Solution: It will still take O(m log m) time since sorting the edges will still be the bottleneck for the algorithm.*

(d) Consider the problem 3-SAT discussed in class. If someone proves that this problem can be solved in polynomial time, what are the implications for the complexity classes P and NP? If someone shows that the problem can *not* be solved in polynomial time, what does this imply about P and NP? What would it imply about all the NP-Hard problems?

Solution: If someone shows that SAT can be solved in polynomial time, then this means that $P = NP$. If someone shows that 3-SAT can not be solved in polynomial time, it implies that $P \neq NP$ and indeed that all the NP-Hard problems are not in P. This is the case since if we could solve any NP-Hard problem in polynomial time, we can solve 3-SAT in polynomial time.

2. **Recurrences and Asymptotics**

(a) Give an asymptotic solution for the following recurrence: $T(n) = 4T(n/2) + 1$ *Solution: $f(n) = 1$ and $af(n/b) = 4$ so $f(n) \leq c * af(n/b)$ for $c > 1$. Thus the leaf nodes dominate and so $T(n) = \Theta(4^{\log_2 n}) = \Theta(n^2)$*

(b) Solve the following recurrence $T(n) = 3T(n-1) + 4T(n-2)$. Give the solution in general form i.e. do not solve for the constants *Solution: The annihilator is $\boldsymbol{L}^2 - 3\boldsymbol{L} - 4$ which factors to $(\boldsymbol{L}-4)(\boldsymbol{L}+1)$. This implies that the general solution is $T(n) = c_1(4)^n + c_2(-1)^n$*

(c) Assume the run time of some algorithm is given by the recurrence $T(n) = 2T(\sqrt{n}) + \log n$. What is the asymptotic run time of this algorithm? *Solution: We need to do a transformation. Let $n = 2^i$. Then we have that $T(2^i) = 2T(2^{i/2}) + i$. Now let $t(i) = T(2^i)$. Then we have $t(i) = 2t(i/2) + i$. But this is easy to solve, it's just the recurrence for mergesort. So we know that $t(i) = i \log i$. This implies that $T(2^i) = i \log i$. This implies that $T(n) = \log n(\log \log n)$.*

(d) Prove that $\log n = o(\log^2 n)$.
Solution: Goal: Want to show that for all positive constants c, there exists a n_0 such that $\log n < c \log^2 n$ for all $n \geq n_0$. The inequality we want then is:

$$\begin{aligned} \log n &\leq c \log^2 n \\ 1 &\leq c \log n \\ 1/c &\leq \log n \end{aligned}$$

Thus we need to choose n_0 such that $\log n_0 \geq 1/c$.

3. P **and** NP

   Consider the following problem which is based on a real-world problem occurring in computational biology. We want to build a "representative set" for a large collection of protein molecules that we don't understand. The idea is to intensively study the proteins in the representative set and thereby learn something about all the proteins in the large collection. To be useful, the representative set must have the following two properties: it must be small so it's not too expensive to study it; and every protein in the collection must either be in the representative set or all proteins similar to it must be in the representative set. Thus the representative set problem can be defined as follows. You are given a large set S of proteins and you are also given a list L of all pairs of proteins that are similar to each other. Your goal is to find a subset S' of S such that 1) every protein p in S is either in S' or all proteins similar to p are in S' and 2) S' is the smallest subset with this property.

   What is the difficulty of the representative set problem? I.e. is it NP-Hard or is it in P? If it's NP-Hard, show this is the case and describe how you might approximate it. If it's in P, give a polynomial time algorithm to solve the problem.

   Solution: The problem is NP-Hard - it can be made equivalent to vertex cover as follows. Let each protein be a node in a graph G and let there be an edge between two nodes in the graph if their corresponding proteins are similar. Now let Z be the set of all nodes of degree zero and let G' be G with the nodes Z removed. The vertex cover for G' plus all nodes Z is the representative set. To see this, note that the requirement that each node either be in S' or have all its neighbors in S' is equivalent to all zero degree nodes being in the representative set and all edges in the graph having one vertex in the representative set. Note that we need to remove all vertices of degree 0 to make the problem equivalent to vertex cover - we took off points on this problem if this critical change was not stated. *Since vertex cover is NP-Hard, so is this problem. Moreover, we can approximate vertex cover by using the greedy 2-approximation algorithm described in class. This gives us a 2-approximation algorithm for the representative set problem since all the zero degree vertices must be in the optimal solution.*

4. **Trees**

The minimum bottleneck spanning tree (MBST) is a spanning tree that seeks to minimize the most expensive edge in the tree. More specifically, for a tree T over a graph G, we say that e is a bottleneck edge of T if it's an edge with maximal cost. The, the tree T is a minimum bottleneck spanning tree if T is a spanning tree and there is no other spanning tree of G with a cheaper bottleneck edge.

(a) Assume you are given a graph G and a weight w and that G has n nodes and m edges. Give a simple $O(n+m)$ time algorithm that determines if there is a MBST with bottleneck edge with cost no more than w.

Solution: Remove all edges with cost more than w from the graph. Then run DFS on the graph starting at an arbitrary vertex to determine if there is a tree on the remaining edges that spans all the nodes.

(b) Is a MBST for a graph G always a minimum spanning tree for G? If so, prove it. If not, give a counter example. *Solution: The answer is no. Consider a graph over 4 vertices a,b,c,d, with the following weight function over the edges. w(a,b) = 1, w(b,c) = 2, w(c,d) = 3, w(a,c) = 2. Then (a,c), (b,c), (c,d) is a MBST but it is not a minimum spanning tree since (a,b), (b,c), (c,d) has minimum total cost.*

(c) Is a minimum spanning tree for a graph G always a MBST for G? If so, prove it. If not, give a counter example. Hint: Use the safe edge theorem. *Solution: The answer is yes. Let T be a MST for G and assume by way of contradiction that T is not a MBST. Then there must be some edge $e = (x, y)$ that is more costly than the minimum cost bandwidth edge. But then consider a cut that respects $T - e$ and thus has x on one side and y on the other. There must then be some edge e' with weight less than e that crosses this cut. Why? Because we know that the MBST uses edges with weight less than e and we know the MBST must have one edge that crosses this cut. Finally consider the tree T' which is T with e removed and e' added. T' is a spanning tree and it has less total weight than T. But this contradicts our assumption that T is a MST. Thus, if T is a MST, T is also a MBST.*

5. **Summer Fishing**

   Note: This problem is similar to a job interview question asked at Facebook.com.

   After your hard work during the year at UNM, you want to take a long deserved fishing break this summer. However, you want to plan your fishing trip so that you catch as many fish as possible and have thus formulated the following problem. You have a map of the fishing spots of New Mexico which you represent as a graph G. Each node in G is a fishing spot and two nodes x and y are connected if and only if it's possible to travel from x to y in an evening (after a day of fishing). Let n be the number of nodes in this graph. Your fishing trip will last for ℓ days and you've created a n by ℓ prediction matrix c, based on your uncanny and infallible knowledge of fishing conditions, such that $c(v,t)$ is the number of fish you will catch at spot v on day t. Note that for the same fishing spot v, and for different days t and t', $c(v,t)$ may not equal $c(v,t')$ because of different weather conditions, and so forth. Your problem is the following. Design an algorithm that given a graph G, a start node s in G, and a matrix c, will plan a fishing trip that ensures you catch the maximum number of fish. In other words, your algorithm will return a path of ℓ nodes, starting at s, and going through G such that this path ensures that the maximum number of fish will be caught from among all paths of length ℓ. What is the runtime of your algorithm?

   Solution: This is a dynamic programming problem. For node v and day t, let $m(v,t)$ be the maximum number of fish that can be caught in a trip that starts at s and ends after t days at node v. First we will define a recurrence for m. Base case: $m(s,1) = c(s,1)$ and for all $v \neq s$, $m(v,1) = -\infty$ (this is to enforce the fact that we must start at node s). Recurrence: For $1 < t \leq l$, $m(v,t) = c(v,t) + max_{v' \in N(v)} m(v',t)$, where $N(v)$ is the set of all neighbors of v in G. The value we want in the end is the maximum over all v of $m(v,l)$. It's easy to transform this recurrence into a dynamic program. We just fill in the array $m(v,t)$ for increasing values of t. This will require three nest loops and the total run time of the algorithm will be $O(n^2\ell)$

5. **Summer Fishing, continued.**