CSE 830: Design and Theory of Algorithms
Sample Questions for Exam #2
Fall 2011

Below are 25 example problems that may appear on your second exam. Note that you should be prepared to provide an intuitive explanation for any algorithms you describe. If you use dynamic programming, the description should include an example chart of partial solutions that you store.

I. Optimization Problems

1. Index-Element Matching. Suppose that you are given a sorted sequence of n distinct integers $\{a_1, a_2, \ldots, a_n\}$. Give an O(log n) algorithm to determine whether there exists an index i such that $a_i = i$. For example, in {-10, -3, 3, 5, 7}, $a_3 = 3$. In {2, 3, 4, 5, 6, 7}, there is no such i.

2. The Currency Problem. Consider a country who has coins with denominations of $\{d_1, d_2, \ldots, d_k\}$ units (you may assume that d_1 is 1 unit). Give an efficient algorithm to determine the minimum number of coins required to make change for n units of currency. Analyze the running time for this algorithm.

3. The Currency Problem II. Formulate an efficient algorithm to determine the total number of possible ways to make change for n units of currency given coins of denominations of $\{d_1, d_2, \ldots, d_k\}$ units (you may assume that d_1 is 1 unit). Analyze the running time for this algorithm.

4. Longest Common Substrings. Given two strings of lengths m and n, give an efficient algorithm to determine the longest common sub-string. For example, the strings "asymptotic" and "unsympathetic" have the longest common sub-string of "symp". Analyze the time complexity for this algorithm.

5. Shuffled Strings. The string Z is a *shuffle* of strings X and Y if Z contains all the letters of X and Y in their original order but interspersed. For example, "chcohciolpaste" is a shuffle of "chocolate" and "chips". Give an efficient algorithm to determine if a string Z is a shuffle of strings X and Y. Analyze the time complexity for this algorithm.

6. The Knapsack Problem. A greedy algorithm will often not produce the optimal answer for the most general form of the Knapsack Problem. Give examples of where the following greedy algorithms would fail:

a. Always choose the most valuable items first.
b. Always choose the smallest items first.
c. Always choose the items of maximum value/size first.

7. String Alignment. Construct the dynamic programming table that would be used to calculate the minimum number of insertions, deletions, and mutations it would take to go from the word "bear" to the word "flea" using the algorithm explained in class. What is the optimal alignment between these words?

8. Recursion v.s. Dynamic Programming. Given the equation T(a, b) = min(T(a-1, b) ; T(a-1, b-1) ; T(a, b-1)) + ab, with boundary conditions of T(a, 0) = a and T(0, b) = b, compare the running time of a recursive implementation of this equation to that of an efficient dynamic programming implementation.

9. The bookshelf problem. In one of your homework problems, you constructed an algorithm to find the minimum height of a bookshelf where each book b_i has a thickness (t_i) and a height (h_i), and all of the shelves have a maximum width W and a height equal to its tallest book. Given the books:

i	t_i	h_i
1	1	1
2	2	2
3	1	5
4	1	4
5	2	3
6	1	1

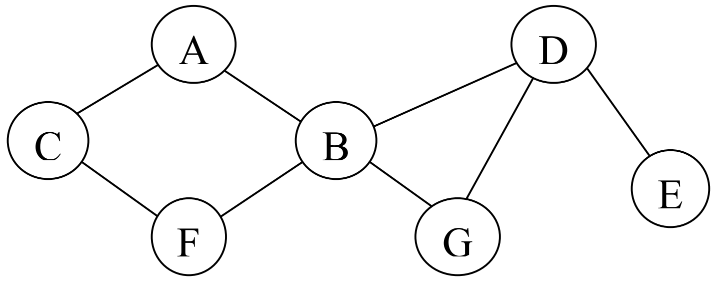
Determine the placement of these books when W=4 that would result in the minimal height. What height would the greedy algorithm have produced?

10. Text Justification. You are given a body of text with n words in it, and you want to print out that text to minimize the number of large gaps of whitespace wasted at the end of lines. Assume that word i is c_i characters long, and a full line can hold up to L characters in it. If k characters are unused in any line, than that line carries a cost of k^2. Thus many small gaps will typically be better than one big one. Design an algorithm to minimize the total whitepace cost of printing this body of text. Analyze the worst-case time complexity of this algorithm.

11. The Approximate Knapsack. Given the standard knapsack problem, construct an approximation algorithm that will guarantee that you take at least one-half of the maximum possible value that can fit in your knapsack. Prove that this is the case.

II. Graph Theory Problems

12. Graph Traversal. Given the following graph, determine the orders that both Breadth-First-Search and Depth-First-Search would traverse (mark as 'discovered') the vertices, starting from 'A'. Break all ties in alphabetical order.



13. Switching Underlying Data Types. Give an O(n^2) algorithm to convert from an adjacency matrix to adjacency lists.

14. Bipartite Gaphs. If the maximum degree in a graph is 2, must it be bipartite? If a graph has cycles of only even length, must it be bipartite? Prove or give a counter example for each.

15. Hamiltonian paths. A Hamiltonian path is a simple path that passes through every vertex in a graph exactly once. Finding a Hamiltonian path in an arbitrary graph can be hard, but in a DAG (directed acyclic graph), it can be found (or decided that no such path exists) in only O(n+m) time. Describe how this is possible.

16. Vertex Cover. The vertex cover problem (choosing a minimum set of vertices where each edge has at least one vertex in the set) is known to be a hard problem, but some instances of it are solvable in polynomial time. Describe three such algorithms if we know that the graph is **(a)** a ring (a connected graph where all vertices are degree 2.), **(b)** a tree. or **(c)** a grid.

17. Articulation Vertices. Describe a graph with *n* vertices that has the maximum number of articulation vertices. Describe a graph with only one articulation vertex that, if deleted, will break the graph into the maximum number of unconnected components.

18. Negative Weighted Graphs. Prim's algorithm generates a minimum spanning tree by starting with a single vertex and repeatedly adding the minimum-weighted edges that include a new vertex into the tree one at a time. Will this algorithm still work if some of the edges have negative weights? Why or why not?

19. Updating Minimum Spanning Trees. If you know the minimum spanning tree for a graph, give an O(n) algorithm to update the MST when a single edge is added.

20. The Traveling Salesman. In the traveling salesman problem, you are provided with a list of *n* cities and a matrix of distances between all pairs of cities. You must output the minimum cost (total distance) route for the salesman to take that will pass through all *n* cities, returning back to the first when he is done. This is known to be a hard problem that must be solved by brute force. Describe the approach that you would use to *correctly* sand efficiently solve this problem. Specify how you handle branching, bounding, backtracking, node ordering and other optimizations you would perform to lead you to the best possible solution.

21. Double Graph Dilemma. You are presented with two graphs G_1 and G_2, which contain the same set of vertices, but with different edges connecting them. Design an algorithm that will find the shortest path between two vertices in G_1 such that no vertices connected by an edge in G_2 both appear in the path. This is known to be a hard problem that must be solved by brute force. Describe the approach that you would use to *correctly* solve this problem as fast as possible. Specify how you handle branching, bounding, backtracking, node ordering, and other optimizations you would perform to lead you to the best possible solution.

22. Matching a Graph. A matching in a graph is a set of disjoint edges (i.e. edges that do not share any vertices in common), which are used commonly in approximation algorithms. Give a linear-time algorithm to find a maximum size matching in a *tree*.

23. Doing the Impossible. Your boss calls you into his office and asks you to implement an algorithm for him that will compute the *longest path* in a graph, and it must run in less than five minutes, even for graphs with 10,000 vertices. You balk, knowing that this is an NP-Complete problem, but he doesn't want to hear any complaints about it. Outline an algorithm that you would use to try to solve this problem for him to the best of your ability, given the time constraint. Remember that you do have five minutes, so don't just come up with an okay solution that returns an answer instantly if you can do better by taking a little bit of time.

24. Spanning distance. Is the path between a pair of vertices in a minimum spanning tree necessarily the shortest path between the two vertices in the full graph? Give a proof or a counter-example.

25. A Bridge to DFS. Suppose G is a connected undirected graph. An edge *e* whose removal disconnects the graph is called a *bridge*. Must every bridge *e* be an edge in a depth-first search tree of G, or can e be a back edge? Give a proof or a counter-example.