CSEP 521 – Applied Algorithm
Spring 2003

Practice Problems for Final

Not to be turned in!

1. Suppose that an algorithm A runs in worst-case time $f(n)$ and that algorithm B runs in worst-case time $g(n)$. For each of the following questions, answer either *yes, no,* or *can't tell*.

- Is A faster than B for all n greater than some n_0 if $g(n) = \Omega(f(n) \log n)$?
- Is A faster than B for all n greater than some n_0 if $g(n) = O(f(n) \log n)$?
- Is A faster than B for all n greater than some n_0 if $g(n) = \Theta(f(n) \log n)$?

2. Suppose that 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 the depth-first search tree of G or can e be a back edge? Give a proof or a counterexample.

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

4. True or false: Suppose that E' is a subset of the edges of a graph such that there is a minimum spanning tree containing all edges in E' and let e be the minimum weight edge in the graph that is not contained in E'. Then $E' \cup e$ is also contained in some minimum spanning tree.

5. True or false: Let (u,v) be the minimum weight edge in the entire graph. Then (u,v) belongs to some minimum spanning tree.

6. True or false: A flow f in a flow network $G=(V,E)$ with source s and sink t is maximum if the value of the flow equals the capacity of some (s,t) cut in the graph.

7. Suppose you have a job that has been partitioned into a large number of smaller tasks, each with associated task time (the time it takes to complete the tasks). Suppose also that there are constraints on the tasks - for each pair of tasks, you are told whether it is essential that one task be performed before the other. Give an efficient algorithm for determining the minimum completion time for the entire job assuming an unlimited number of parallel processors (each able to complete a task the moment its precedence constraints have been satisfied).

8. Consider the following packing problem: Given an integer B and a set of *n* different items, where the *i*th item has size *s*[*i*] (an integer), the problem is to find a subset of the items whose sizes sum is maximize but not more than B *(note that this is the knapsack problem with profits=sizes).*

8.a. Give an integer programming formulation of this problem.

8.b. (Dynamic Programming) Assume that each item occurs in unlimited supply. The problem is to pack items of the given sizes *s*[*i*] in a knapsack of size B, but each item may appear many times in the knapsack. We want to find out if we can fil the knapsack to full capacity.

We formulate this as follows. We have *n types* of items, with an infinite supply of items of each type. Each item of the *i*th type has size *s*[*i*]. We wish to determine if there is a subset of the items whose sizes sum to exactly *B*. (There can be an arbitrary number of items of any type in this subset.)

- For any nonnegative integer j, define *f*(j) to be 1 if there is a subset of the items whose sizes sum to exactly j, and 0 otherwise (so f(B) is the solution we are seeking). Give a recursive formulation for f(j) (Don't forget the base case.
- Give pseudocode for computing f(B).

9. Explain in at most two sentences what is wrong, if anything, with the following induction proof

Claim: All horses are the same color.
Proof: We prove this by showing that any set of horses contains only horses of a single color; in particular, this is true of the set of all horses. Let H be an arbitrary set of horses. We show by induction on n, the number of horses in H, that all horses in H are the same color.
Base: The cases n = 0 and n = 1 are immediately seen to be true.
Induction step: Consider any number n of horses in H. Call these horses h1, h2, h3, ..., hn. By the induction hypothesis, any set of n-1 horses contains only horses of a single color. Consider the set H1 obtained by removing horse h1 from H, and the set H2 obtained by removing horse h2 from H. There are n-1 horses in each of these sets, hence the induction hypothesis applies to each: H1 has all horses of a single color (say, c1), and H2 has all horses of a single color (say, c2). But, since horse hn is common to *both* sets H1 and H2, the two colors c1 and c2 must be the same. This completes the induction step.

10. Show that if there is a polynomial time algorithm that determines whether a graph has a Hamiltonian cycle, then there is a polynomial time algorithm to find Hamiltonian cycles (in other words, a black box for the decision problem can be used to solve in poly-time the reporting problem).

11. The decision vertex-cover problem is to take a graph *G* and an integer *k* and decide if *G* has a vertex cover of size *k* or not. The optimization problem takes a graph *G*, and returns a smallest vertex cover in *G*. Show that if the decision problem has a polynomial

time algorithm then the optimization problem also has a polynomial time algorithm. Recall that a vertex cover is a collection *S* of vertices with the property that every edge is incident to a vertex in *S*.

12. Consider the following 2Clique problem:
INPUT: A undirected graph *G* and an integer *k*.
OUTPUT: 1 if *G* has two vertex disjoint cliques of size *k*, and 0 otherwise.
Show that this problem is *NP*-hard. Use the fact that the clique problem in *NP*-complete. The input to the clique problem is an undirected graph *H* and an integer *j*. The output should be 1 if *H* contains a clique of size *j* and 0 otherwise.

13. Consider the following problem. The input is an undirected graph *G* and an integer *k*. The problem is to determine if *G* contains a clique of size *k* **AND** an independent set of size *k*. Recall that an independent set is a collection of mutually nonadjacent vertices. Show by reduction that if this problem has a polynomial time algorithm then the clique problem has one.

14. It is known that the scheduling problem $1||\Sigma_j T_j$ is NP-hard. Can you use this fact to prove that $1||\Sigma_j L_j$ is also NP-hard?

on-line algorithms

15. Consider a version of bin packing in which in addition to the regular constraint (each bin can accommodate items of total size at most one), we can pack at most c items in each bin.
For this version a possible greedy online algorithm is: Given the next item a_i, if some open bin includes less than c items of total size at most 1-size(a_i), pack a_i in such a bin. Otherwise, open a new a bin for a_i.

15.a. Let n be the total number of item, give two lower bounds on OPT, one based on the volume constraint and one based on the cardinality constraint.
15.b Use the above bounds to show that the number of bins used is at most 3OPT+1

16. Prove that the paging rule LFU (least frequently used) is not competitive. In LFU we have a counter for each page, counter[i] is increased whenever the page i is accesses. When a page fault occurs, we evict from the cache the page whose counter has the lower value. You need to prove that for any c there exists a sequence in which the number of cache faults of LFU is at least c times the number of misses of an optimal algorithm.

17. Prove that FIFO is k-competitive.
Hint: Partition the sequence into maximal phases with exactly k faults of FIFO per phase. Consider an arbitrary phase and the next request after that phase.