

## CSci 280, Spring 2009, Exam 2

This take-home exam is due **Friday, April 3**, at 7am. You should submit your solution on paper, either handwritten or typed. E-mailed solutions will not be accepted.

**Answer only four of the following five problems.** If you answer all problems, I will grade only the first four. Each problem is worth 20 points, so your solutions will be graded out of 80 points.

The quality of writing and mathematical argument will be a central component in how your solutions are graded. When I return your graded solution, I will identify a problem that you should rewrite to be eligible for the final 20 points. (If your initial submission has no significant flaws, the final 20 points will require no additional work.) In submitting the rewrite, you should also include your original, graded test submission. Note that I will grade the rewrite more harshly than I graded the original; simply resubmitting your original solution will not be worth anything.

You should not obtain help from anybody or anything, except your instructor, your written class notes, the textbook (Dasgupta, Papadimitriou, & Vazirani, available on paper or on-line), and the course Web site.

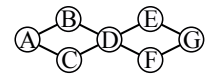
1. Suppose we have  $m$  machines, each with startup time  $s_i$ . Only one machine can be running at a time. We also have  $n$  jobs, each of which can be performed on any of the machines; on machine  $i$ , job  $j$  takes  $t_{i,j}$  time. We want to find the time taken by the fastest assignment of jobs to machines.

The below table gives an example involving 3 machines and 5 jobs. The best schedule is for machine 0 to perform the first two jobs, then machine 1 for job 2, then machine 2 for the last two jobs. This schedule takes  $(2 + 1) + 1 + (1 + 2) + (2 + 1) + 1 = 11$  time units. However, the best schedule for just the first four jobs is for machine 0 to do them all, taking  $(2 + 1) + 1 + 4 + 1 = 9$  time units.

	$s_i$	$t_{i,0}$	$t_{i,1}$	$t_{i,2}$	$t_{i,3}$	$t_{i,4}$
machine 0	2	1	1	4	1	5
machine 1	1	3	3	2	3	3
machine 2	2	3	3	4	1	1

Using dynamic programming, create an algorithm taking  $O(mn)$  time for finding time taken by the best assignment. Present the solution in pseudocode, and argue that it is correct and has the required big-O bound.

2. Given a connected, undirected graph, we can define an **essential vertex** as one whose removal splits the graph into two or more components. In the example graph at right, vertex D is essential, since removing it leaves the graph in two components, A/B/C and E/F/G. No other vertices are essential.



Create an  $O(m)$  algorithm that identifies an essential vertex in a graph if one exists. Present the algorithm in pseudocode, and argue that it is correct and has the required big-O bound. Hint: Think about the depth-first-search tree and its back edges.

3. The SEAT ASSIGNMENT problem is as follows: Suppose we are throwing a dinner party at which  $n$  people will be assigned seats at a round table. However, we also know of some bad relationships among our guests, and we want to make sure we don't assign two guests to sit next to each other if they have a bad relationship.

Prove that SEAT ASSIGNMENT is NP-hard by reducing it from another problem known to be NP-hard.

*Continued on next page...*

4. Recall that for the KNAPSACK problem, we are given a collection of weights  $\{w_0, w_1, \dots, w_{n-1}\}$  and a capacity  $c$ , and we wish to find the subset of weights with the largest sum without exceeding  $c$ . The most intuitive approach to selecting weights is a greedy algorithm: We first sort the weights in descending order, then go through each weight and select each that fits into the knapsack.

For example, given the weights  $\{9, 7, 6, 3\}$  and a capacity of 14, we first see whether the largest weight 9 fits; it does, so we place it into the knapsack. We then see that 7 and then 6 don't fit into the knapsack. But the 3 does, so we place it into the knapsack, for a total weight of  $9 + 3 = 12$ . This isn't optimal, though: We could have selected 7 and 6 to get a weight of 13. But it's close to optimal.

Prove that the greedy algorithm is a 2-approximation algorithm. Also, show that 2 is the best possible approximation ratio for this algorithm; that is, for any ratio less than 2, you can give an example where the greedy algorithm does worse than that ratio.

5. A few years ago a utility contracted you to write a program for designing telecommunications networks between towns. Your program's input is a list of possible town connections along with the cost of laying a wire between each pair of towns. You naturally chose to use Prim's algorithm to identify a minimum spanning tree among the towns.

But they have now identified that often there are some possible connections that are politically very sensitive, and the upper management is unwilling to deal with more than one politically sensitive connection. A list of connections that are politically sensitive will also be fed as input to your program.

Design an algorithm so that it chooses at most one politically sensitive connection. Your algorithm should take  $O(n^2 + m \log n)$  time. One approach is to use Prim's algorithm in essentially unmodified form and then work with the resulting spanning tree to arrive at the best possible answer. Present your algorithm in pseudocode, and argue that it is correct and has the required big-O bound.