CS 4349.501 Final Exam—Problems and Instructions

December 11, 2017

Please read the following instructions carefully before you begin.

- **DON'T PANIC!**
- Write your name and Net ID on the *answer sheets* cover page and your Net ID on each additional page. Answer each of the six questions on the answer sheets provided. One sheet was intentionally left blank to provide you with scratch paper.
- You're allowed to bring in one 8.5" by 11" piece of paper with notes written or printed on front and back.
- You have 2 hours and 45 minutes to take the exam.
- Please turn in these problem sheets, your answer sheets, scratch paper, and notes by the end of the exam period.
- If asked to design and analyze an algorithm, you should describe your algorithm clearly and give its asymptotic running time using big-oh notation in terms of the input size. ***You do not have to write any proofs unless the question explicitly says otherwise.***
- Feel free to ask for clarification on any of the problems.

1. (a) Recall the algorithm MergeSort from class, where the subroutine Merge($A[1 .. n], m$) sorts the array A assuming subarrays $A[1 .. m]$ and $A[m+1 .. n]$ are already sorted.

```
MergeSort(A[1..n]):
  if n > 1
      m ← ⌊n/2⌋
      MergeSort(A[1 .. m])
      MergeSort(A[m + 1 .. n])
      Merge(A[1 .. n], m)
```

In one sentence, describe what MergeSort($A[1 .. n]$) does to A if we remove the last line of the algorithm so that we never call Merge.

For parts (b) and (c), consider the following alternative sorting algorithm (and please do not try to prove its correctness during the exam):

```
StoogeSort(A[1..n]):
  if n = 2 and A[1] > A[2]
      swap A[1] ↔ A[2]
  else if n > 2
      m ← ⌈2n/3⌉
      StoogeSort(A[1 .. m])
      StoogeSort(A[n − m + 1 .. n])
      StoogeSort(A[1 .. m])
```

(b) The worst-case running time of the ***original*** MergeSort algorithm given in lecture follows the recurrence $T(n) \leq 2T(n/2) + O(n)$. Give a recurrence for the running time of StoogeSort($A[1 .. n]$). *[Hint: Feel free to ignore the ceiling.]*

(c) What is the running time of StoogeSort($A[1 .. n]$) obtained by solving your recurrence?

2. Recall the Backpack problem from Homework 11 in which you are given a collection of books of different costs and are tasked with storing as expensive a subset of books as possible in your backpack at once. Unfortunately, the books are very heavy, and you are limited in how much total weight you can carry.

   Formally, you are given two arrays $C[1 .. n]$ and $W[1 .. n]$ of positive integers where $C[i]$ is the cost of book i in dollars and $W[i]$ is the weight of book i in pounds. You are also given a positive integer M which is the maximum load you can carry in pounds. Your goal is to compute the maximum total cost of any subset of books you can carry from books 1 through n.

   Let $T = \sum_{i=1}^{n} C[i]$ be the total cost of all given books. Given integers i and c with $0 \leq i \leq n$ and $0 \leq c \leq T$, let $MinWeight(i, c)$ denote the minimum total weight of any subset of books 1 through i that have total cost ***exactly*** c. Let $MinWeight(i, c) = \infty$ if there is no such subset of books. This is ***not*** the function defined in Homework 11.

   (a) Give a recurrence definition for $MinWeight(i, c)$. Don't forget the base cases!

   (b) Design and analyze a dynamic programming algorithm to build a two-dimensional array $MinWeight[0 .. n][0 .. T]$ where for each pair of indices i and c, $MinWeight[i][c] = MinWeight(i, c)$ as defined above. Your algorithm should have a running time of $O(nT)$.

   (c) Design and analyze an algorithm for the Backpack problem that runs in $O(nT)$ time. You may assume your solution to part (b) is correct and use it as a black box if you would like.

3. Recall the class scheduling problem from lecture in which you want to find a maximum cardinality subset of classes such that no two classes have overlapping time periods.

   For each of the following alternative greedy algorithms for the class scheduling problem, write ***Correct*** if the algorithm always constructs an optimal schedule or write ***Wrong*** if there is some input for which the algorithm does not produce an optimal schedule. *You must clearly write exactly one of* **Correct** *or* **Wrong** *to get credit for each part.* ***Proofs/counterexamples are not required***, but you may want to do them on your scratch sheet to convince yourself that your answers are correct. Each wrong algorithm has a small counterexample.

   (a) Choose the course x that *ends last*, discard classes that conflict with x, and recurse.

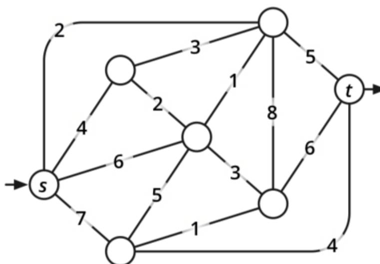
   (b) Choose the course x that *starts last*, discard all classes that conflict with x, and recurse.

   (c) Choose the course x with *shortest duration*, discard all classes that conflict with x, and recurse.

   (d) If no classes conflict, choose them all. Otherwise, discard the course with *longest duration* and recurse.

   (e) If any course x completely contains another course, discard x and recurse. Otherwise, choose the course y that *ends last*, discard all classes that conflict with y, and recurse.

4. Clearly indicate (draw) the following structures from the weighted graph pictured below. Some of the subproblems have more than one correct answer. For parts (c) and (e), you may treat every undirected edge uv as a pair of directed edges $u \to v$ and $v \to u$ with the same weight/capacity.



(a) A depth-first spanning tree rooted at s

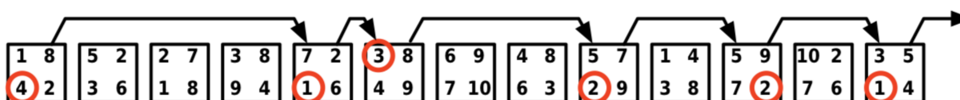
(b) A breadth-first spanning tree rooted at s

(c) A shortest-path tree rooted at s

(d) A minimum spanning tree

(e) A minimum (s,t)-cut

5. Consider the following solitaire game played on a row of n squares. Each square contains four positive integers. The player begins by placing a token on the leftmost square. On each move, the player chooses one of the numbers on the token's current square and then moves the token that number of squares to the right. The game ends when the token moves past the rightmost square. The object of the game is to make as many moves as possible before the game ends.



An instance of the puzzle that allows six moves. (This is ***not*** the longest legal sequence of moves.)

(a) Prove that the obvious greedy strategy (always choose the smallest number) does not give the largest possible number of moves for every instance of the puzzle. Your proof should be a small counterexample and a description of the optimal sequence of moves that does better than the greedy strategy.

(b) Describe and analyze an efficient algorithm to find the largest possible number of legal moves for a given instance of the puzzle. Your algorithm should ideally run in $O(n)$ time. *[Hint: Two possible strategies are to design a dynamic programming algorithm or to reduce to longest paths in directed acyclic graphs. Any correct $O(n)$ time algorithm is worth full credit.]*

6. The University of Northern Incredible has hired you to write an algorithm to schedule their final exams. Each semester, UNI offers n different classes. There are r different rooms on campus and t different time slots in which exams can be offered. You are given two arrays $E[1 .. n]$ and $S[1 .. r]$, where $E[i]$ is the number of students enrolled in the ith class, and $S[j]$ is the number of seats in the jth room. At most one final exam can be held in each room during each time slot. Class i can hold its final exam in room j only if $E[i] \leq S[j]$.

   Describe and analyze an efficient algorithm to assign a room-time slot pair to each class (or report correctly that no such assignment is possible). *[Hint: Matchings. Be sure to give your runtime in terms of n, r, and t.]*