CS180 Spring 2017 - Midterm

Monday, May 1, 2017

You will have 110 minutes to take this exam. This exam is closed-book and closed-notes. There are 6 questions for a total of 100 points. **Please write your name and student ID on every page of your solutions. Please use separate pages for each question.**

Question	Points
1	/10
2	/20
3	/20
4	/10
5	/20
6	/20
Total	/100

1. [10 points]

   (a) Prove formally that $2^n = O(n!)$.

   (b) You work for a company and one of your colleagues claims that he (or she) invented a new sorting algorithm whose running time is $f(n) = 16f(\frac{n}{16}) + \frac{1}{2}n$ where n is the size of input to the algorithm and that this algorithm is way better than all existing comparison based sorting algorithm (which is $n \log n$ at best) in terms of asymptotic running time. Is he or she right or wrong? State your answer and prove it formally.

a) $n!$ grows faster than 2^n such that $\lim_{n \to \infty} \frac{2^n}{n!} = 0$

therefore we can say $2^n \leq K \cdot n!$; $K > 0 \Rightarrow 2^n = O(n!)$

b) We can analyze this algorithm using master theorem

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)^d$$

if $d > \log_b a \Rightarrow T(n) = \Theta(n^d)$

if $d = \log_b a \Rightarrow T(n) = \Theta(n^d \log n)$

if $d < \log_b a \Rightarrow T(n) = \Theta(n^{\log_b a})$

In this problem:

$a = 16$

$b = 16$

$d = 1$

$\log_b a = \log_{16} 16 = 1 = d$

Therefore the running time of this algorithm is

$$\Theta(n^d \log n) = \Theta(n \log n)$$

So we say this person is <u>wrong</u> because their algorithm is not better than the best comparison based sorting algorithm

7

(cont)

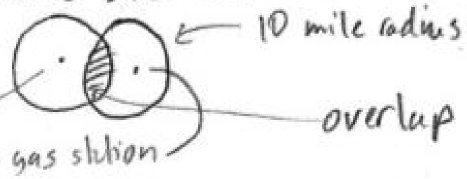
2. **[20 points]** Your millionaire-friend decide to open pizza-parlors on a particular stretch of highway from Los Angeles to Las Vegas, since he knows that there are no pizza restaurants on this deserted highway stretch, and many drivers who love pizza go through it. He wants to open at least one pizza-parlor within 10 mile distance of each gas-station on the highway in order to dominate the inferior food quality offerings at gas stations, and he wants to dominate that particular highway stretch with high quality pizza offerings. Since he heard that you are taking an algorithms class at UCLA, he asks you for an algorithm to places as few pizza-parlors as possible to cover each gas station.

   Explain the algorithm that you would use to decide where on the highway to place pizza restaurants for your rich friend. He says that he will accept your solution only if you could prove to him that it is an absolute minimum number of restaurants to cover all gas stations, so you should prove that as well.

if every gas station from LA to Las Vegas is at least 10 miles apart from each other, then simply place one pizza parlor within 10 miles of every gas station

if there is at least one overlap* between gas stations
sort gas stations in order of number of overlaps from greatest to least
go through the list and assign a pizza parlor at each overlap until all gas stations are covered

*overlap =>



The first part of the algorithm yields the minimum number because if all gas stations are at least 10 miles apart from each, there is no overlap and you must place at least one pizza parlor at each gas station

The second part of the algorithm also yields the minimum number because at each selection of pizza parlor placement, the algorithm does better than any other algorithm.
At each selection of pizza parlor placement, we place a pizza parlor that covers as many gas stations as possible
Therefore at each step, we use the minimum number of restaurants to cover the most gas stations

14

$$\begin{tikzcd}
 & & b \arrow[dr, dash] & \\
s \arrow[r, dash] & a \arrow[ur, dash] \arrow[dr, dash] & & t \\
 & & d &
\end{tikzcd}$$

3. **[20 points]** Often, there are multiple shortest paths between two nodes of a graph. These shortest paths may share edges between them. That is s → a → b → t and s → a → d → t are two distinct shortest paths, even though they both use the edge s → a. Give a linear-time algorithm for the following problem:

   Input: An undirected graph G = (V,E) with unit edge length, nodes s and t.

   Output: The number of distinct shortest paths from s to t.

   Give a outline of algorithm, its proof of correctness, and the running time analysis.

Main idea: use depth first search (DFS)

Algorithm: initialize a counter: number of shortest paths (NSP) = 0

- start at node s and perform a DFS until you reach node t
- once node t is reached, increment NSP
  then retrace your steps until you reach a node that has unvisited edges
- go through that unvisited edge and traverse the graph in a DFS until is t is reached again.
  repeat this logic
- once you retrace your steps all the way back to s you have the number of shortest paths.

Running time: in the worst case, we visit every edge
$O(|E|)$

5

4. **[10 points]** Your high-school buddy goes to lunch with you one day, and confidentially tells you that he made an amazing discovery: that he can reduce in polynomial time the Minimum Spanning Tree (MST) problem to Traveling Salesman Problem (TSP). That is, MST $\leq_p$ TSP. He plans to write up his solution carefully and send it to the most prestigious journal in computer science, but he does not want to share with you any details of his intricate solution.

Assume that he did not make any mistakes, and proved correctly that MST $\leq_p$ TSP. Do you feel that this result is important enough to be published in prestigious journal in computer science? Explain your answer in detail.

This result is Not important

MST $\leq_p$ TSP => if TSP can be solved in polynomial time then MST can be solved in polynomial time

However TSP is NP-complete and there is no known polynomial time algorithm to solve it, so the claim MST $\leq_p$ TSP is meaningless

On the other hand if the claim was

TSP $\leq_p$ MST, TSP can be reduced in polynomial time to MST

This would be significant.

TSP $\leq_p$ MST => if MST can be solved in polynomial time then TSP can be solved in polynomial time

this is significant because there already exists polynomial time algorithms that solve MST (Kruskal's, Reverse-Delete, etc...)

This would mean that TSP can be solved in polynomial time, which means that every NP-complete problem could be solved in polynomial time, which would imply P = NP

This would be a significant discovery

10

5. **[20 points]** Recall the Traveling Salesman problem, TSP:

18

Input: A matrix of distances D (all distances are positive integers), a budget B.
Output: A tour which passes through all the cities and has length less than or equal to B, if such a tour exists.

The optimization version of this problem asks directly for the shortest tour TSP-OPT:

Input: A matrix of distances D(all distances are positive integers).
Output: The shortest path which passes through all the cities.

Show that if TSP can be solved in polynomial time, then so can TSP-OPT.

Basically, we want to prove that TSP-OPT is NP-complete such that $\text{TSP-OPT} \leq_p \text{TSP}$

Goal: reduce in polynomial time TSP-OPT to TSP

Main idea: assume there exists some black-box algorithm that solves TSP in polynomial time.
We will use this black-box algorithm as a subroutine to solve TSP-opt. This subroutine will be invoked a polynomial number of times.

Algorithm outline:
- Establish some upper bound, which will be the max budget B_{max}; Let's say B_{max} = the total distance in matrix
- set budget = B_{max}
- invoke black-box as a subroutine with the following input: Black-box (matrix, budget)
- The output will confirm or reject if such a tour given that budget exists
- if it exists, decrement the budget, and repeat black box (i.e. Blackbox(matrix, (budget-1)))
- if it does not exist, then we have the minimum budget for which a valid tour exists (i.e. the shortest path which passes all cities)

Conclusion: if TSP can be solved in poly time using a black-box algorithm then we can invoke that algorithm a poly # of times to solve TSP-OPT

6. **[20 points]** Show that for any problem Q in NP, there exists an algorithm which solves Q in time $O(2^{p(n)})$, where n is the size of the input and $p(n)$ is a polynomial dependent on input n.

For a problem to be in NP, this implies that this problem does not have a polynomial time solution but if a solution were given to a verifier, it could be verified in polynomial time.

to solve Q in time $O(2^{p(n)})$ we simply calculate all possible combintions of an input of size n and send them to a verifier.

A verifier would verify that solution in polynomial time, $p(n)$

Since calculating all possible combinations takes exponential time and verifying that solution takes $p(n)$ time, we say that any problem $Q \in NP$ can be solved in time $O(2^{p(n)})$

10