Name: Cody Hubbard

Student ID: 004 843 389

TA - 1B (12-2)

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

Name: Cody Hubbard

Student ID: 004 843 389

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) Proof: Consider $\lim_{n \to \infty} \frac{2^n}{n!} = 0$, so $2^n = O(n!)$ ∎

Additionally let $c = 2$ $n_0 = 3$ then $\forall n \geq n_0$ $2^n \leq 2 \cdot n!$

(b) masters thm $a = 16$ $b = 16$ $c = 1$ $k = 0$ $\frac{1}{2}n = O(n \log^k n)$ ✓

$1 = \log_{16} 16^1$ so $\boxed{f(n) = O(n \log n)}$

He is false, by the masters' theorem his algorithm is $\Theta(n \log n)$ and thus is not "better" than the other sorting algorithms

Name: Cody Hubbard

Student ID: 004 843 389

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.

I would use a greedy algorithm. Set each gas station to not having a pizza parlor. Go 10 miles from the first gas station and make a pizza place. Set it and any other gas stations this pizza place is 10 miles from to having a pizza place. Go 10 miles past the next gas station which does not have a pizza place and make a new one, Setting nearby gas stations to having pizza places as before. Continue this method until every gas station has a pizza place.

Proof by Induction on G the number of Gas stations

Case G=1. trivial, 1 pizza place is needed.

By Inductive hypothesis. the Greedy method will give optimal pizza places for G=k gas stations, Say m pizza places.

Consider the k+1th gas station.

If this gas station is more than 10 miles from the last pizza place my algorithm will result in (m+1) total pizza places, one additional.

Name: Cody Hubbard

Student ID: 004 843 389

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.

on G use Dijkstras algorithm to find the shortest path from S to T. once this path is found store its length.
For every vertex in this path run Dijkstras on each of the neighbors of that vertex (which were not in the first found path.
(I.E. S→A→B→T run dijkstras from all neighbors of A to T excluding B) (shortest)
For each path found, if the length is the same as the stored length increase the count of distinct paths. Lastly return # distinct paths +1
(+1 is for the original path)

This is correct because it searches for every shortest path from S→T. . . .

Dijkstras S to T
store path.
store length.
for each node in path
  Dijkstra each neighbor
  Chek if length = shortest
  inc # paths (dont inc #paths)
return #paths+1

Dijkstras
1
7
6(Dijkstras + 1 + 1) + 1

The running time of this problem relies entirely on the running time of Dijkstras, the complexity is

Dijkstras + 1 + 1 + C(Dijkstras + 1 + 1) = Dijkstras + C(Dijkstras + 2) + 2

where C is just some constant number of nodes. Thus = Dijkstras(C+2C) + 2

= O(Dijkstras) thus if Dijkstra's Algorithm runs in linear time this algorithm does as well.

Name: Cody Hubbard

Student ID: 004 843 389

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.

No, I do not believe that this result is important enough to be published in a prestigious Journal. This is because TSP is an NP-complete problem. Proving that TSP is as hard as or harder than MSP is almost trivial as there are several algorithms (Primm, Krushal) which give solutions in polynomial time or better and are checkable in polynomial time, this cannot be said for TSP. If he had proved TSP $\leq_p$ MST then it would be worth publishing.

Name: Cody Hubbard

Student ID: 004 845 389

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

   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.

NTS $TSP\text{-}OPT \leq_p TSP$

Suppose D, the input of TSP-OPT is also D', the input of TSP. Consider B, the budget for TSP. There is no suitable input for B from the TSP-OPT problem so let it be a custom variable b.

My idea is that if we have a black box that solves TSP then we can keep lowering b until there exists no solution; thus finding the "shortest" tour

Suppose we have a black box that solves TSP in polynomial time, then use a binary search type procedure on b to find the point where no-solution meets solution (like a zero of this problem). This finds the lowest possible budget and thus The "shortest" path which passes through all the cities, which solves TSP-OPT.

Suppose we have a black box which solves TSP-OPT in polynomial time. Then using this black box, any solution to TSP-OPT using the input D of TSP will yield a solution to TSP with some Budget, B = total cost of the TSP-OPT solution.

Name: Cody Hubbard

Student ID: 004 843 389

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.

Given $Q \in NP$, Then $Q \leq_p SAT$ by definition/theorem(?).
It is known that SAT is solvable in time $O(2^{p(n)})$ as well.
And thus by definition of reducable $Q = O(2^{p(n)})$.