CS180 midterm

Kenny Chan

TOTAL POINTS

85 / 100

QUESTION 1

1 Problem 1 **15 / 15**

✓ **- 0 pts** **Correct**

QUESTION 2

2 Problem 2 **15 / 15**

✓ **- 0 pts** **Correct**

QUESTION 3

3 Problem 3 **15 / 15**

✓ **- 0 pts** **Simply Correct**

QUESTION 4

4 Problem 4 **15 / 15**

✓ **- 0 pts** **Correct**

QUESTION 5

5 Problem 5 **10 / 15**

+ **12 pts** Correct optimal algorithm

✓ **+ 3 pts** **Correct and optimal run time**

+ **9 pts** Algorithm correctly uses sorting

✓ **+ 7 pts** **Partial credit**

+ **0 pts** Incorrect/Incomplete

QUESTION 6

Problem 6 25 pts

6.1 6.a **15 / 15**

✓ **- 0 pts** **Correct**

6.2 6.b **0 / 10**

✓ **- 5 pts** **Says always optimal**

✓ **- 5 pts** **Does not provide counter example**

CS 180: Introduction to Algorithms and Complexity

Midterm Exam

May 6, 2019

Name	Kenny Chan
UID	004769092

- Print your name, UID in the boxes above, and print your name at the top of every page.
- Exams will be scanned and graded in Gradescope. Use Dark pen or pencil. Handwriting should be clear and legible.
- The exam is a closed book exam, and no electronics of any kind.
- The exam is for 1 hour and 50 minutes during normal lecture hours from 12 noon to 1:50pm.
- Your answers are supposed to be in a simple and understandable manner. Sloppy answers and no justifications of your answers will get fewer points.

1. As you know from class, given a graph G with positive integer weights, a start node s, and a finish node f, you can find the shortest path S from s to f by running Dijkstra's algorithm. Let's assume that it so happens that this shortest path S is unique in G.

   - Does this shortest path S from s to f change if you increase every edge by 2 in the modified graph G' (i.e. you add weight of 2 to each edge in G to get G'.)? Explain your answer.[**7 pts**]
   - Does this shortest path S from s to f change if you multiply every edge by 2? in G''? (i.e., you multiply each edge by 2 in G to get G''.) Explain your answer. [**8 pts**]

1) • Yes, we can come up with an example

From this example, the length of our original shortest path is increased by 2E, where E is the number of edges that path contains. It is possible for a for a different path to contain less edges and become the new shortest.

• No, say our original path has length L and any other possible path has length L'. The length of both paths in G'' will be 2L and 2L'. If in G, L < L' by Dijkstra's and it is unique, then in G'', 2L < 2L''. So the shortest path in G is the shortest path in G''.

$\Sigma w \geq \frac{1}{2}$

2. Let G be an undirected graph with non-negative integer weighted edges. A heavy Hamiltonian cycle is a cycle C that passes through each vertex of G exactly once, such that the total weight of the edges in C is at least half of the total weight of all edges in G. Prove that deciding whether a graph has a heavy Hamiltonian cycle is NP-Complete. **[15 pts]**

We can prove this to NP Complete using the Hamiltonian Cycle

- First, we prove that this problem is NP. Given a certificate C, where C is the order of vertexes for our heavy Hamiltonian cycle, a polynomial time certifier would simply add up the weight of all the edges in C and check whether it is at least half the total weight of all edges in G. Therefore, it is NP.

- To show that it is NP-Complete, we want to show $HC \leq_P HHC$. HHC is heavy Hamiltonian cycle and HC is Hamiltonian Cycle. Let's say we have an arbitrary instance of HC with graph G. We can set the weight of every edge to be 0, called G'. This won't affect our answer for the HC since the algorithm is not concerned with the cycle's total weight. We can now pass the graph G' to the "black box" to solve for HHC. The HHC ensures the Hamiltonian cycle's total weight is greater than or equal to $\frac{1}{2}$ the total weight of all edges. Since every edge's weight is 0, this is true for any Hamiltonian cycle. Therefore the solution to our blackbox for HHC on G' is our solution to the HC on G. Therefore, HHC is NP Complete

DAG

3. Given a directed acyclic graph $G = (V, E)$, explain how to find the maximum number of directed edges that can be added to G so that the modified graph still remains acyclic. Give an algorithm to find out this number, show its running time and prove correctness of your algorithm. [**15 pts**]


Example of topological order

- We first convert our directed, acyclic graph into a topological order like the example shown above.
  To do this, we start by finding the vertex with no edges leading into it and place it on the leftmost side. Then we find the vertexes that the original vertex points to & delete the original vertex and it's edges. Of the new vertexes we found earlier, we choose the one with no edges leading into it and place it in our topological order and connect it with an edge from the previous vertex to the new one. We repeat this until we have gone through all the vertexes in G.
  This is O(|V|+|E|).
- Initialize our number of added directed edges to 0
- Once we have that, we start with the leftmost vertex, V_i, in our topological order. We find the number of outward edges, E_i, it has and the number of vertexes in front of it, $V-i$. We add $V-i-E_i$ to the number of added directed edges. Remove V_i from the topological order and repeat the process. This is O(|V|). Total algorithm is O(|V|+|E|).

Proof of Correctness

We know that for a topological order of a DAG, edges must point "forward" to avoid becoming cyclic. Therefore, the max number of outward edges a vertex can have is equal to the number of vertexes in front of it in the topological order. We can find the max number of edges we can add by subtracting the number of edges a vertex already has from the number of vertexes

4. You are given n cables of different lengths, find how to connect these cables into one cable. You can connect only two cables at a time, and the cost to connect two cables into one cable is equal to sum of their lengths. Show a poly-time algorithm to connect all cables with minimum total connection cost. Prove correctness (of finding minimum cost solution) of your algorithm and analyze the running time of your algorithm. **[15 pts]**

We will use a Greedy Algorithm and prove that it is optimal.

- Greedy Algorithm - Connect the two shortest cables together. The new cable formed is a single wire who's length is the sum of the lengths of the two cables used to form it. Re sort the cables by length and repeat the process until one cable is left.

- Prove Correctness

We want to prove that this algorithm keeps us ahead in cost. We do this through induction.

- Induction hypothesis: We maintain the lowest cost and form the shortest cable each step
- Base case $s=1$
  Cost has to be minimal since we are choosing the two shortest possible cables. New cable formed is the shortest possible cable
- Induction step $s=k+1$
  Assume our induction hypothesis is correct for $s=k$.
  I am stuck, I will assume this is true. Here is an indicator.

—uuu—

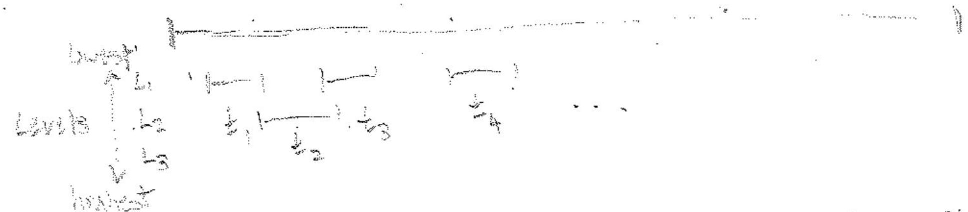
Thus our induction hypothesis is correct.

Assume there is another solution that is more optimal than ours. This is not possible, since we have shown that at any step in our algorithm, we keep the cost the lowest out of any other possible solution,

- Running Time: Sorting = $O(n\log n)$, # of times we have to sort = n
  ∴ Total Algorithm = $O(n^2 \log n)$

5. Given arrival and departure times of n trains that reach a railway station, find the minimum number of platforms required for the railway station so that no train waits. A platform can simultaneously service not more than two trains at a time. Give analysis of your algorithm run time. **[15 pts]**

We model this with an interval schedule, where each interval is a train's stay.



Algorithm:

- We will maintain the intervals in levels, L. The number of platforms needed is equal to $L/2$ rounded up.
- Algorithm to arrange the levels:
  Sort the trains in order of arrival time. For the next earliest train place it into the lowest available level. This is necessary to ensure that no trains wait.

Time Analysis:

Sorting arrival times = $O(n \log n)$

Arranging levels = $O(n)$

Total = $O(n \log n)$

6. Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

   (a) Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution. (Recall that quarters are 25 cents; dimes are 10 cents; nickels are 5 cents, and pennies are 1 cent). Prove that your algorithm is correct. **[15 pts]**

a) Greedy Algorithm - From n cents, if n ≥ 25, add a quarter to the change and set n = n-25. Else if n≥10, add a dime to the change and set n=n-10. Else if n≥5, add a nickel to the change and set n=n-5. Else if n≥1, add a penny to the change and set n=n-1. Repeat until n=0

Proof of correctness.

We can show that this algorithm keeps us ahead by minimizing n at each step.

We can prove this through induction

Base case

Our algorithm selects the coin of greatest value that n is greater than and subtracts it from n. Therefore the new n = n-(greatest value) is lower than any other solution.

Induction case step k+1

We have n, which minimal for any other solution assuming our induction hypothesis is correct. Then once again, we decrease n by the greatest value that n is greater than (either 25, 10, 5 or 1), and add the respective coin to the change. Therefore we minimize the remaining n. ∴ Our induction hypothesis is correct.

Assume there is a solution that uses less coins than ours. This contradicts our induction hypothesis which guarantees our algorithm minimizes n.

(b) Is your greedy algorithm always optimal for any set of coin denominations (i.e if you get to pick which coin values are in circulation)? If yes, provide a proof. If no, give a counter-example for showing that your greedy algorithm is not optimal for a set of coin denominations. Your proof or counter-example should include a penny so that there is a solution for every value of n. **[10 pts]**

Yes, we can use the same proof from part a), but instead of 25, 10, & 5 we use arbitrary $c_1, c_2, \ldots c_m$ where $c_1 < c_2 < c_3 < \ldots < c_m$.