# CS 180: Introduction to Algorithms and Complexity

## Midterm Exam

May 6, 2019

| Name | Ali Mirabzadeh |
|---|---|
| UID | 305-179-067 |

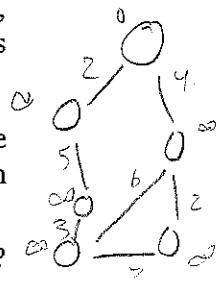
- Print your name, UID in the boxes above, and print your name at the top of every page.
- Exams will be scanned and graded in Gradescope. Use Dark pen or pencil. Handwriting should be clear and legible.
- The exam is a closed book exam, and no electronics of any kind.
- The exam is for 1 hour and 50 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.

5 6 = 11 a+b > c+d

10 10

**Name:** Ali Mirabzadeh **UID:** 305-179-067 **CS180 Midterm Exam**

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 $G$.

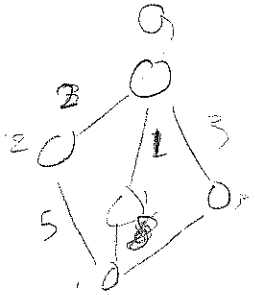



a/ • Does this shortest path $S$ from $s$ to $f$ change if you increase every edge by 2 in the modified graph $G'$ (i.e. you add weight of 2 to each edge in $G$ to get $G'$.)? Explain your answer.**[7 pts]**

b/ • 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]**

a/ NO, It doesn't change, since every edge is getting larger the same amount, adding 2 to each node, it wouldn't have any effect on it. As we know Dijkstra's algorithm would give a weight to all the neighbors of a the node it's visited and then compare the length to weigh of the node. Since adding 2 is a linear calculation it wouldn't change the shortest path; it would be the same but at the end would have a larger sum.

b/ NO, it doesn't change. The same reasoning happens here. Let's say from $s \to f$, it took $w_1+w_2+w_3$ in $G$ which is the shortest path. In $G$ $w_4+w_5+w_6$ was greater than $w_1+w_2+w_3$ so it wasn't considered as the shortest path.




Now let's go to $G''$; in other words each weight now is 2 times bigger. Hence $w_1+w_2+w_3$ in $G$ is $2w_1+2w_2+2w_3$ in $G''$ which is still smaller than $2w_4+2w_5+2w_6$

If

$$w_1+w_2+w_3 < w_4+w_5+w_6 \xrightarrow{\times 2} 2(w_1+w_2+w_3) < 2(w_4+w_5+w_6)$$

$$\Rightarrow 2w_1+2w_2+2w_3 < 2w_4+2w_5+2w_6$$

Hence, Dijkstra's algorithm would still choose $S$ in $G''$

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]**

$W_{HHC} \geq \frac{W_G}{2}$

1. clearly. Heavy Hamiltonian Cycle (HHC) is in NP as in there is a solution that can be solved in P-time, since it is derivative of Hamiltonian cycle problem, HC is also in NP.
- Certificate: Set of all nodes and edges in $G$
- Certifier: solution shows that, there is such a path that is call HHC. you have to sum weight of all edges in $G$ which means traversing through all the nodes, also you have to find the sum of edges for HHC which is basically a HC so it takes P-time to find solution

2. Pick a known problem that is NP-complete

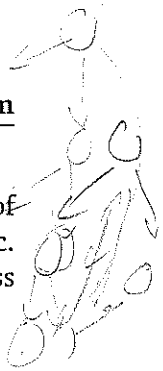
Hamiltonian Cycle $\leq_P$ Heavy Hamiltonian Cycle

3. I am going to prove that HC, a known NP-Complete problem can be reduced to HHC

We already know that HC would find the shortest path by visiting all the nodes, we can map this problem in way that we would provide the minimum ourselves such that the minimum be at least $\frac{\sum W_G}{2}$, half of the total weight of all edges in $G$.

In this way, HC algorithm tries to find HHC by comparing its sum to the give sum, $\frac{\sum W_G}{2}$, if matched then it would be HHC.

∴ HHC is NP-Complete since it HC $\leq_P$ HHC



3. Given a directed acyclic (no cycle) 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]**

Algorithm:

We create list of visited node to be empty, R
Also, a list of all nodes in the graph, A

while R is not empty

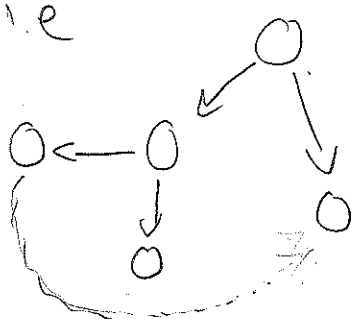
try to connect the current node visited, to any other node that is not connected yet to the current node, ie if it is not connected, check from A if the current node and node we are connecting to makes any cyles with other nodes in A if it doesn't make a cycle

increment the variable that count the number of edges

end while

return the counter

Proof: It is correct since it starts from a node and tries to connect it to a node that hasn't been connected to yet. It checks if makes a cycle by calling the union function, if it doesn't make a cycle then that means it can be added i.e



Running time $O(|V|+|E|)$

1 2 3 4 5 6

Name: Ali Mirabzadeh UID: 305-179-067 CS180 Midterm Exam

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]**

First sort the cables from the shortest to the longest; $O(n \log n)$
then Connect the first two cables, Call it CableSum
then
For each cable in the sorted list, start from the third cable, 3, 4, ... n : $O(n)$
add $C_i$ to CableSum
end for
return the sum

Proof: Let's say the say we have sorted list of integers representing the length of each cable. since we connect two cables at a time and we're looking to make a minimum total connection cost; we have to sum every two neighbor element in the list at a time to make the sum minimum at a time

i.e list = $[a_1, a_2, a_3, a_4]$ where $a_1 < a_2 < a_3 < a_4$

$a_1 + a_2 < a_1 + a_3$ or $a_1 + a_4$ or $a_2 + a_3$ so we first do $a_1 + a_2$ (two Cables)

then $(a_1 + a_2) + a_3$ (two Cables) then $((a_1 + a_2) + a_3) + a_4$ (two Cables)

There long the algorithm above would 'work' and it is efficient as it takes $O(n \log n)$, sorting part, to find the minimum total connection cost

running time: $O(n \log n)$

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]**

a) Since we have the arrival and departure time, we can easily find the interval that each train stays at a platform; like interval scheduling problem.

Therefore, we sort our list based on the interval : $O(n \log n)$

Also, we create a counter variable to keep track of #of platforms needed

Then we create an empty list to add intervals based on the needed platforms, R

while ( the sorted list of platform is not empty)

Check if the interval is not in R, if it is continue

Otherwise find the first compatible interval with the current interval

add the two intervals to R

increment the counter for platforms

endwhile

return the counter

b) It takes $O(n \log n)$

First we sort with the most efficient algorithm : $O(n \log n)$

the while loop takes $O(n)$

Therefore, at the end would be $O(n \log n)$

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]**

To make an optimal solution I make while loop that loops through until find all changes.

The rules I give to the algorithm is to use the most quarters then dimes then nickels and lastly cents to reduce the number of coins so within the loop, first I check if n divided by 25, quarters, is not zero, the if it is not, then I do the divison, take the result and it to the variable that sotores number of coins and store the remainder into n.

I will have three similar if statment following the first one with the same functionality except that I check for dimes, nickels and cents in order

I keep doing this till find all the changes, so I will exit the loop and return the number of coins

This algorithm is correct as it looks for number of coins by using the minimum number of coins. we already know that using coins with larger amount would decreas the number of coins

i.e for 35 cents you can use 2 coins, quarter and a dime which the algorithm does behave in that manner, first start with quarters, then dime, etc

(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]**

My algorithm is optimal for any given set of coin denominations including pennies, as within the loop it starts by checking for the largest coins to the smallest, pennies.

i.e for 4 cents it would return 4 as it is the only possible way to change

for 11 cents it would return 2, dime and a penny

for 16 cents, it would return 3, dime, nickels and a penny.

Therefore, despite the set, it would always return the minum number of coins and it's always optimal.

6 5