CS 141 homework4

Celyna Su

TOTAL POINTS

100 / 100

QUESTION 1

1 Question1 **25 / 25**

✓ **+ 5 pts** **Explains correctly why the approach is a greedy algorithm.**

✓ **+ 5 pts** **Analyzes time complexity.**

✓ **+ 15 pts** **Complete proof of optimality (greedy choice and optimal substructure).**

+ 10 pts Proof of optimality is incorrect or incomplete (missing greedy choice or optimal substructure).

+ 0 pts Didn't prove optimal solution or Incorrect answer

+ 5 pts Proof of optimality is incorrect and incomplete (missing greedy choice or optimal substructure).

QUESTION 2

2 Question2 **25 / 25**

✓ **- 0 pts** **Correct**

- 5 pts No/Incorrect greedy approach

- 3 pts Unclear/Incomplete proof

- 5 pts No proof

- 23 pts Completely wrong answer

- 3 pts Unclear greedy approach

- 25 pts No answer

- 20 pts Programming code is not allowed

QUESTION 3

3 Question3 **25 / 25**

✓ **- 0 pts** **Correct**

- 10 pts Incorrect Algorithm

- 15 pts Proof not shown

- 10 pts Incorrect proof

- 10 pts One of the properties not proved

- 5 pts Algorithm not explained properly

- 3 pts Time complexity is incorrect/not shown

- 7 pts Proofs not explained properly

- 25 pts Incorrect Solution

QUESTION 4

4 Question4 **25 / 25**

✓ **- 0 pts** **Correct**

- 15 pts Explanation not valid or sufficient

- 23 pts Incorrect answer

- 3 pts Conclusion is not right

- 6 pts Reason not given for why we pair each of the leaf node

- 3 pts Why it will be a balanced tree?

- 6 pts Incorrect reasoning

CS 141, Spring 2019 Homework 4

Posted: May 2nd, 2019 Due: May 9th, 2019

Name: Celyna Su

Student ID #: 862037643

- You are expected to work on this assignment on your own
- Use pseudocode, Python-like or English to describe your algorithms. Absolutely no C++/C/Java
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in class, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your algorithms
- Homework has to be submitted electronically on Gradescope by the deadline. No late assignments will be accepted

Problem 1. (25 points)

Consider the Activity Selection problem we discussed in class. Suppose that instead of always selecting the first activity to finish, we instead select the last activity to start (as long as it is compatible with all the previously selected activities). In other words, instead of considering the activities by “earliest finish”, we consider them by “latest start”. Explain why this approach is a greedy algorithm, analyze its time complexity, and prove that it yields an optimal solution (greedy choice and optimal substructure).

Answer: It’s considered a greedy algorithm because we incrementally scheduled the activities that start the latest and end with our events that start the earliest. We maximize the set of remaining activities, which makes this algorithm greedy.

Greedy-Choice Property:
Proof. First, we assume that $A \subseteq S$ is an optimal solution. Then, we order the activities in A by decreasing start time.
Let k be the activity that starts the latest in A.
If $k = 1$, A starts with a greedy choice.
Otherwise, if $k \neq 1$, then there is another solution B which is sorted in a way similar to that of A, which also starts with a greedy choice.
In this situation, switch out the element that starts the latest in B and replace it with k.
$B = (A - \{k\}) \bigcup \{1\} \longleftarrow$ This is the first element since the array is sorted with decreasing start time.
Because all the activities in A do not conflict with each other, and all the activities in B do not conflict with each other, k is the last activity to start and $s_1 \leq s_k$.
A and B are the same size, which makes it optimal.

Optimal Substructure: We want to show that $A \subseteq S$ is an optimal solution, which can only be done if and only if $A' \bigcup \{1\}$ is an optimal solution for S.
Proof. Let A' be an optimal solution for S', which means $A' \bigcup \{1\}$ is an optimal solution for S because if we disagree, that means there must be a solution that is larger, implying $B' \bigcup \{1\}$. But, that will mean that B' is a more optimal solution, which contradicts the fact that A' is an optimal solution for S.

The time complexity is $O(n \log n)$ because we are sorting it the same way as stated in the slides, but flipped.

Problem 2. (25 points)

A server has n customer waiting to be served. The service time required by each customer is known in advance: it is t_i minutes for customer i. So if, for example, the customers are served in order of increasing i, then the i-th customer has to wait $\sum_{j=1}^{i} t_j$ minutes. We want to minimize the total waiting time:

$$T = \sum_{i=1}^{n} (\text{time spent waiting by customer } i)$$

Give a greedy (efficient) algorithm for computing the optimal order in which to process the customers. Prove why your algorithm is correct (i.e., always returns an optimal solution).

Answer: We start by populating an array with all the wait times for each customer, then sort it in increasing order, t_i.

Greedy-Choice Property:
Proof. We assume that $A \subseteq T$ is an optimal solution.
Let a_j be the least wait time for customer j in A.
If the element for any customer j in A is a_j then A starts with a greedy choice.
Otherwise, if the element for any customer j in A is not a_j then there is another solution B which also begins with a greedy choice.
By having an element c that does not have the shortest wait time, the optimality of A is contradicted. Therefore, you have to switch that specific element with one where that element is the shortest wait time, denoted s.
$B = (A - \{c\}) \bigcup \{s\}$

Optimal Substructure: We want to show that $A \subseteq T$ is an optimal solution, which can only be down if and only if $A' - \{h\}$ is an optimal solution for $S - \{h\}$ where h is a customer with the shortest service time.
Proof. Let A' be an optimal solution for $S - \{h\}$.
Then, that means that $A' - \{h\}$ is an optimal solution for S because if we say that it is not, that means there must be a solution that results in a shorter wait time, $B' - \{h\}$.
Since the summation of shortest time is equal to the wait time of h, a_h, plus the time of all the customers $B' - \{h\}$ must be less than $A' - \{h\}$.
However, that means B' becomes a more optimal solution which contradicts the fact that A' is an optimal solution for S.

The time complexity is $O(n \log n)$ because we are sorting it in the fastest way possible.

Problem 3. (25 points)

Assume that you are given two arrays $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ of n real numbers.

1. Describe a greedy algorithm to determine an ordering of the elements of A and B such that $W = \sum_{i=1}^{n} |a_i - b_i|$ is minimized
2. Analyze the time complexity of your algorithm
3. State and prove the greedy-choice property of your algorithm
4. State and prove the optimal substructure property of your algorithm

Hint: consider using the following fact. Given real numbers $x_1 \leq x_2$ and $y_1 \leq y_2$, then

$$|x_1 - y_1| + |x_2 - y_2| \leq |x_1 - y_2| + |x_2 - y_1|.$$

Answer: We start by sorting both arrays in increasing order.
Let each sorted array be A_s and B_s. Let each of their respected elements be named e_s, where e is an element in the array. The time complexity is $O(n \log n)$.

Greedy-Choice Property:
Proof. We ssume that in solution W, the summation is $W = \sum_{i=1}^{n} |a_i - b_i| = |a_{s1} - b_{s1}| + |a_{s2} - b_{s2}| ... |a_{sn} - b_{sn}|$
The first choice of solution W is $|a_{s1} - b_{s1}| + |a_{s2} - b_{s2}|$.
We assume there is another optimal solution, W'.
W' has a similar algorithm. However, the first choice is $|a_{s1} - b_{s1}| + |a_{s2} - b_{s2}|$.
According to the hint, we know that the first choice of W results in less than or equal value to that of W'.
Since W' is optimal and W results in a smaller value, W must also be optimal.

Optimal Substructure:
Proof. Since W asks us to create an optimal solution in which the first elements of the array are present, that means there must be another optimal solution in which there is an iteration where the first elements are not included in the summation.
We can write that as $N = W - \{a_{s2} - b_{s2}\}$. The summation for this algorithm will then be $N = \sum_{i=2}^{n} |a_i - b_i| = |a_{s2} - b_{s2}| + |a_{s3} - b_{s3}| ... |a_{sn} - b_{sn}|$
We can also say that $N = |a_{s1} - b_{s1}| + \sum_{i=2}^{n} |a_i - b_i| = |a_{s1} - b_{s1}| + |a_{s2} - b_{s2}| + |a_{s3} - b_{s3}| ... |a_{sn} - b_{sn}|$
N must be the optimal choice, because if there is a more optimal function, N', would be able to create a function in which the summation is even less. If we were to create that function it would contradict the statement above. Thus, tihs proves that this is an optimal solution.

Problem 4. (25 points)

Suppose a data file contains a sequence of 8-bit characters, such that all the 256 characters are about as common. More precisely, the maximum character frequency is less than twice the minimum character frequency. Explain why Huffman coding in this case is no more effective than using an ordinary 8-bit fixed-length code.

Answer: Assume that the characters are sorted in ascending order of frequency. Thus, we would get $f_1 < f_2 < f_3 < ... < f_{256} < q_1$ in which $f_1 + f_2 = q_1$. We know that f_1 and f_2 will be joined first into f_1', and because $f_2 \geq f_1$ and $2 \cdot f_1 > f_{256}$, this will not be joined until after the last value, f_{256}, is joined with another element.
Continuing on, f_3 and f_4 will be joined into f_2' with $f'2 \geq f_1' \geq f_{256}$.
As we keep on going, we get f_{253} and f_{254} joined into $f_{127}' \geq ... \geq f_1' \geq f_{256}$.
Lastly, f_{255} and f_{256} are joined into $f'128 \geq f_{127}' \geq ... \geq f_1'$.
Using this information, we know that since $f_{256} < 2 \cdot f_1 \leq f_1'$ and $f_{128}' \leq 2 \cdot f_{256}$, $f_{128}' \leq 2 \cdot f_{256} \leq 4 \cdot f_1 \leq 2 \cdot f_1$.
Thus, $f_{128}' < 2 \cdot f_1'$, which is the same condition that held for the first round of the Huffman coding.

Because the condition held this round, it is safe to assume that it will hold on all rounds. Huffman will run 8 rounds in total until all nodes are joined to the root $(128, 64, 32, 16, 8, 4, 2,$ and finally, $1)$ to create a Huffman tree. Each branch will have the same length, 8, because at each stage each node is joined to another one that has also received the same treatment by the Huffman algorithm. In conclusion, this is why in this case, Huffman coding is no more effective than using an ordinary 8-bit fixed-length code.