Name: Vera Duong

ID: 109431166

CSCI 3104, Algorithms **Due March 5, 2021**

Problem Set 6 (50 points) **Spring 2021, CU-Boulder** Collaborators: Nathan Straub, Matt Hartnett

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.
Advice 2: Verbal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solution:

- The solutions **should be typed** and we cannot accept hand-written solutions. Here's a short intro to Latex.
- You should submit your work through **Gradescope** only.
- The easiest way to access Gradescope is through our Canvas page. There is a Gradescope button in the left menu.
- Gradescope will only accept **.pdf** files.
- It is vital that you match each problem part with your work. Skip to 1:40 to just see the matching info.

Name: $\boxed{\text{Vera Duong}}$

ID: $\boxed{109431166}$

CSCI 3104, Algorithms **Due March 5, 2021**

Problem Set 6 (50 points) **Spring 2021, CU-Boulder** Collaborators: Nathan Straub, Matt Hartnett

1. (a) Consider designing an algorithm for issuing coins from a vending machine. This algorithm should take in an amount of change needed x and issues the minimum number of coins whose value sums to x. Given an ordered set of coin denominations, $C = \{c_1, c_2, ...\}$, a typical greedy approach is to choose the largest coin c_k such that $c_k \leq x$, then set $x = x - c_k$ and recurse. Using the set $C = \{1, 5, 10, 25\}$, list the coins returned for the given values of x.

   Example $x = 24$. *Solution:* [10, 10, 1, 1, 1, 1]

   i. $x = 16$
   ii. $x = 49$
   iii. $x = 31$

   (b) Provide a set C of coin denominations for which the greedy approach given in (a) is **not** always optimal. Give an example value of x for which the greedy solution and optimal solution differ, and show that these two solutions differ by listing the coins the greedy solution would pick versus the optimal set of coins.

   (c) Suppose you are late for (an in-person!) class and are trying to cram important items from your apartment into your backpack to take to campus. However, your backpack only has k total space and you have a variety of items $I = \{i_1, i_2, ...\}$, each of which has a value $v(i)$ and size $s(i)$. We want to choose the set of items I^* that maximizes the sum of values $\sum_{i \in I^*} v(i)$ while still fitting in the backpack: $\sum_{i \in I^*} s(i) \leq k$.

   One greedy approach to the problem is to take the item i_n with max $v(i_n)$ such that $s(i_n) \leq k$, then set $k = k - s(i_n)$, remove i_n from I and recurse. Given the following set of items, and with a backpack size of $k = 20$, provide the optimal and greedy solutions and note whether or not they differ.

Item Name	Value	Size
Laptop	20	12
Phone Charger	13	8
Notebook	18	11
Pencil	3	2
Textbook	22	15

   (d) Another greedy approach to this problem is to first calculate each item's value *density*, defined as $d(i) = \frac{v(i)}{s(i)}$, then choose items in order of max $d(i)$ until the backpack is full. More specifically, this approach chooses the item i_n with max $d(i_n)$ such that $s(i_n) \leq k$, then sets $k = k - s(i_n)$, removes i_n from I, and recurses. Given the same set of items above, this time with a backpack size of $k = 21$, provide the optimal solution and this greedy solution and note whether or not they differ.

Solution:

(a) i. Using the typical greedy approach, we would first use the dime, nickle, and penny. This adds 10 + 5 + 1 = 16.

   ii. Using the typical greedy approach, we would first use the quarter, 2 dimes, and 4 pennies. This adds 25 + 10 + 10 + 1 + 1 + 1 + 1.

   iii. We would first use the quarter, nickle, and one penny. This adds 25 + 5 + 1.

(b) For my chosen set of C coin denominations, I have $C = \{1, 2, 10, 15\}$ and my example value is $x = 22$. The greedy approach would give us 15 + 2 + 2 + 2 + 1, which is 5 coins. The optimal solution would just give us 10 + 10 + 2, which is only 3 coins and differs by 2 coins compared to what the greedy one returns.

(c) For the Greedy Solution we first have the Textbook because it has the largest value and with a size of 15, and we next pick the pencil (size 2) because it's the only other item which doesn't add over size 20 and has a value of 3. The Greedy Solution would give us a value of 25. For the optimal solution, we have the Laptop (size 12) and Phone Charger (size 8) which totals the size limit equal to $k = 20$ and has a combined value of 33. These differ because for the optimal we reached the maximum size and had the largest value possible 33 without going over size 20, and for the greedy solution we couldn't max out the size and only found a combined value of 25. This means that choosing the largest value for an item isn't always the best option.

(d) The density for each item listed:
Laptop: $20/12 = 1.66$
Phone Charger: $13/8 = 1.625$
Notebook: $18/11 = 1.636$
Pencil: $3/2 = 1.5$
Textbook: $22/15 = 1.46$

For the Greedy Solution we would first pick the Laptop because it has the largest density. Next we would pick the phone charger because it has the next largest density that doesn't go over the size limit, making the two items with a combined size of 20 (less than the limit size of 21) and a value of 33.

For the Optimal Solution, we would pick the Phone Charger, Notebook, and Pencil because their sizes add up to 21, which is exactly the size limit, and their combined values are 34.

The Greedy approach and the Optimal approach are also different in this case because for the greedy, we ended up with a size of 20, with space for size 1 leftover in the backpack and a combined value of 33. For the Optimal, we were able to utilize all the space of size 21 and still get a combined value of 34, which is larger than 33. This also shows that it's not always the best option to pick the largest density because with that idea, this greedy situation still gave us a lower value and room leftover of size 1.

2. Suppose we have a number of events m_i. Each event starts at time s_i and finishes at time e_i, where $0 \leq s_i < e_i$. We represent the event m_i with the closed interval $[s_i, e_i]$. Our goal is to construct a maximum size set of events, where no two events in the set overlap (note that two events m_i and m_j where $m_i = [\cdot, x]$ and $m_j = [x, \cdot]$ are not considered to overlap).

   Suppose the following intervals are provided.

Event Index	Interval
A	$[1, 4]$
B	$[2, 3]$
C	$[3, 6]$
D	$[5, 9]$
E	$[7, 11]$

   (a) Give the optimal set of events for this example set of intervals.

   (b) Consider the following greedy approach: at each step, select the event with the earliest starting time that does not conflict with any event already in our set. Show that this approach is not optimal by providing the greedy solution generated by this approach for the above example intervals.

   (c) Consider a second greedy approach: first order the events by length. Then, at each step, select the event with the shortest length that does not conflict with any event already in our set. While this approach generates the optimal set for the above example, it is not guaranteed to do so for this problem generally. Give a set of 5 intervals for which this second greedy approach fails to generate the optimal solution. Explicitly state both the optimal solution and the solution generated by this approach for your example.

   (d) State a greedy approach to this problem that always finds the optimal solution.

Solution:

(a) The Optimal set of events for this example would be $\{B, C, E\}$. This is because the most events possible is 3 events, and these three don't overlap with eachother.

(b) The Greedy approach where we select the events with the earliest starting time (without time conflicts) are events $\{A, D\}$. This is not optimal because there are only 2 events, whereas in part (a) we showed that the max is 3. Picking the earliest starting time is not always the best option.

(c) By selecting the events by length, we also get $\{B, C, E\}$ which is optimal in this example. Below I will give a set of 5 intervals that isn't optimal if we select by length:

Event Index	Interval
A	$[1, 4]$
B	$[4, 7]$
C	$[7, 10]$
D	$[3, 5]$
E	$[7, 9]$

If we chose to select events by length, our solution would be $\{D, E\}$ and only gives us 2 events whereas the optimal solution gives us 3 events, in this case $\{A, B, C\}$

(d) A Greedy approach to this problem that always finds the optimal solution would be to select by the earliest order of ending times (as long as it's legal).

Name: Vera Duong

ID: 109431166

CSCI 3104, Algorithms **Due March 5, 2021**

Problem Set 6 (50 points) **Spring 2021, CU-Boulder** Collaborators: Nathan Straub, Matt Hartnett

3. The following problems deal with the concept of Huffman Coding.

   (a) Suppose your friend tells you that they have written a program to generate Huffman codes. You test it out by entering the text of *Wuthering Heights*, and it generates the following codes for the first several letters of the alphabet:

Letter	Code
a	01
b	00101
c	101
d	00111
e	1
f	000
g	0011
...	...

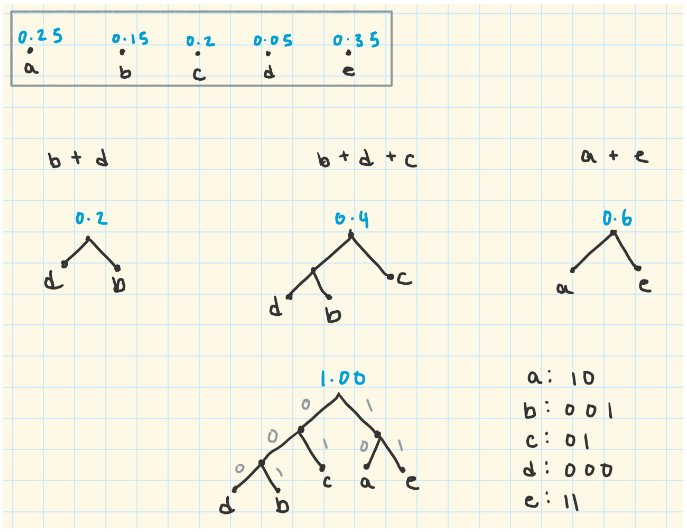
   Even without knowing the exact distribution of letters in *Wuthering Heights*, you are immediately suspicious of your friend's program. Explain why this is not a possible valid set of Huffman codes.

   (b) Generate Huffman codes for the following set of letters with their given frequencies. Show your work by writing out the list of nodes and their respective frequencies available at each step. You can refer to nodes containing multiple letters in their subtree using a capitalized list of letters in alphabetical order; for example, if after some step you had a node representing a subtree containing a and e with frequency 0.6, you could refer to it in your list as $(AE, 0.6)$. Don't forget to list the final codes at the end!

Letter	Frequency
a	0.25
b	0.15
c	0.2
d	0.05
e	0.35

Solution:

(a) The Huffman codes provided is not a valid set because some codes are prefixes to another code. We want an encoding where no code is a prefix for another because it makes it ambiguous and difficult to decipher. For example, g is a prefix for d. If we wanted to write "ge" it could be interpreted as the letter "d." We can also see that the code for the letter "e" is a prefix for "c." So if I wanted to write "ea" it could be misinterpreted as the letter "c."



a: 10
b: 001
c: 01
d: 000
e: 11

(b)

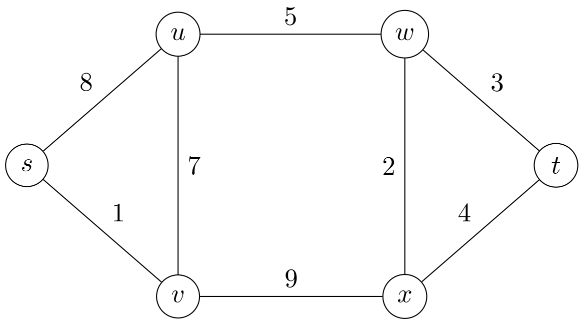
(3b continued)

Our encodings from the tree are as follows: a(10), b(001), c(01), d(000), e(11)

To generate these codes, we first begin by adding two nodes with the lowest frequencies, so we will have (db, 0.2) from doing $0.15 + 0.05 = 0.2$. d is on the left since is lower than b. Next, we add 0.2 to $c = 0.2$ for next two smallest nodes, which is basically calculating $d + b + c$. We will end up with (dbc, 0.4) with c on the right and db on the left. The next two smallest are a and e, so we add $0.25 + 0.35$ to get another node resulting in (ae, 0.6). We combine (dbc, 0.4) to (ae, 0.6) to get (dbcae, 1). Going left to right we will assign 0 and 1 to the branch in order to calculate a path which generates the Huffman encodings given.

Letter	Code
a	10
b	001
c	01
d	000
e	11

4. The questions in this problem make use of the following weighted graph:



(a) List all simple paths from s to w, along with their associated costs. (Note: A simple path is a path with no repeated vertices)

To refer to a path, list the vertices in order in which they are encountered; for example, if you wanted to list the path from s to t along the top of the graph, you would write $(suwt, 16)$.

(b) Consider a form of depth first search that uses a greedy heuristic to decide which edge to traverse next from a given node. Once a node is selected for expansion, this approach orders the edges from that node by minimum cost. You can also assume this implementation of depth first search remembers previously-visited nodes and does not select edges that would visit them again. Starting from node s, what is the first path found from s to t and what is its cost? Is this the optimal path from s to t? If not, give the optimal path.

(c) Using the version of depth first search described in (b), list the order in which depth first search visits all of the nodes if we start the algorithm from u.

Solution:

(a) (s u w, 13)

(s v x w, 12)

(s v u w, 13)

(s v x t w, 17)

(s u v x w, 26)

(s u v x t w, 31)

(b) The first path found is (s v u w x t, 19)

We start at s and travel to v since it's cost 1 is less than 8. Next, we travel to u because it's cost is $7 < 9$ and we already visited s. Next, u travels to w because we already visited its other two vertices, v and s. Next, we travel to x because the cost 2 is less than 3, its other vertex t. Finally, we go x to t because we already visited its other vertices w and v. This gives a combined cost of 19.

This is not the optimal path from s to t because we can travel s to v to x to t and still get a lower cost of 14. This comes to show that choosing the lowest path cost is not always optimal.

(c) If we start at u we first travel to w because its cost 5 is lower than the other two connected vertices (s and v). Next we go to x because its cost 2 is less than 3 (connected to t). From x, we make it to t since its cost 4 is less than 9 and we already visited w. We can't go anywhere from t because w and x have already been visited. So we back track to node x and travel to v since w and t were already visited. Next, we go to s since it's unvisited, and thus the list of the order of nodes where dfs visits all the nodes is u, w, x, t, (track back to x) then v, and s.