Name: Willem Scott

ID: 108759616

Collaborators: Enrico Blackwell and Ethan Miles

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

Problem Set 6 (50 points) **Spring 2021, CU-Boulder**

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: Willem Scott

ID: 108759616

Collaborators: Enrico Blackwell and Ethan Miles

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

Problem Set 6 (50 points) **Spring 2021, CU-Boulder**

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. $[10, 5, 1]$
   ii. $[25, 10, 10, 1, 1, 1, 1]$
   iii. $[25, 5, 1]$

(b) $[1, 16, 20]$ will give a non optimal greedy solution for 32, yielding $[20, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$ instead of $[16, 16]$

(c) The greedy algorithm will yield a score of 25 with $[Textbook, Pencil]$ where the optimal solution would be $[Laptop, PhoneCharger]$ with a score of 33

(d) This greedy solution would yield $[Laptop, PhoneCharger]$ with a score of 33, where the optimal solution is $[PhoneCharger, Notebook, Pencil]$ with a score of 34

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 solution is $[B, C, E]$

(b) The greedy algorithm yields $[A, D]$, which is not optimal.

(c)

Event Index	Interval
A	$[1, 4]$
B	$[3, 5]$
C	$[4, 8]$
D	$[7, 9]$
E	$[8, 12]$

The optimal solution is $[A, C, E]$, but this greedy algorithm would prioritize $[B, D]$, which is not optimal.

(d)

The optimal solution grabs the earliest non-conflicting end time at each step. With the previous example, it would produce the optimal solution.

Name: Willem Scott
ID: 108759616
Collaborators: Enrico Blackwell and Ethan Miles

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) This is a suspicious code because e is already the prefix for c, which is ambiguous.

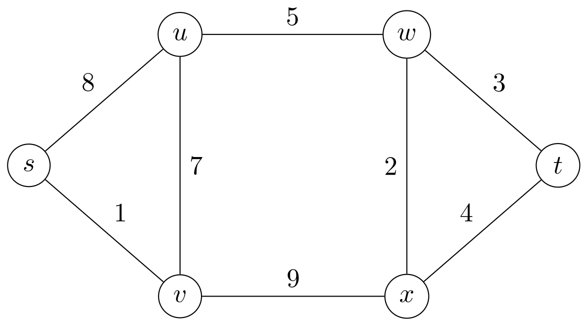
(b) i. (ABCDE, 1)
   ii. (AE, .6), (BCD, .4)
   iii. (DB, .2), (C, .2), (A, .25), (E, .35)
   iv. (D, .2), (B, .05)

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

Name: Willem Scott
ID: 108759616
Collaborators: Enrico Blackwell and Ethan Miles

CSCI 3104, Algorithms **Due March 5, 2021**
Problem Set 6 (50 points) **Spring 2021, CU-Boulder**

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) i. $(suw, 13)$
ii. $(svuw, 13)$
iii. $(svxw, 12)$
iv. $(svxtw, 17)$
v. $(suvxw, 26)$
vi. $(suvxtw, 31)$

(b) This greedy solution finds $svuwxt$ as its first path. This is not the optimal path, as it has a cost of 19 where the optimal has a cost of 14 by $svxt$.

(c) $uwxtvs$ is the order of nodes found using a depth first search.