CSci 242: Algorithms and Data Structures **Fall, 2019**

Instructor: Dr. M. E. Kim Date: November 8th, 2019

**Due: by the end of day, November 15th (Fri.), 2019.**

**Home Assignment 7: Greedy Algorithm and Huffman Codes (150 points)**

**Q1. [30]** **Job Scheduling**

Suppose a hair stylist has several customers waiting for different treatments. The treatments don’t all take the same amount of time, but the stylist knows how long each takes. A reasonable goal would be to schedule the customers in such a way as to minimize the total time they spend both waiting and being served, which is called the time in the system. This is called *a problem of minimizing the total time in the system.:*  total time = waiting time + service time.

There are five jobs and the service times for these jobs are:

|  |  |  |
| --- | --- | --- |
| **Jobs** | **Service Time** |  |
| A | 15 |  |
| B | 6 |  |
| C | 5 |  |
| D | 9 |  |
| E | 10 |  |

1. [10] Write a recursive greedy algorithm to decide the optimal sequence of jobs to minimize the total time spent in the system.

I=1

total =

hold= 0

recursiveGreedy(I, hold, total):

Hold = hold + job[I].time

Total = total + hold

If I < N then recursiveGreedy(I + 1, total)

Return(job, total)

1. [10] Design an iterative greedy algorithm for the same problem of 1.

I=1

Hold = 0

Total = 0

IterativeGreedy(job)

SortedArray

While(I <= N)

Hold = hold + job[I].time

Total = total + hold

I = I + 1

Return job, total

1. [10] (A) What is the optimal solution of the above problem, i.e. the optimal sequence of job? and (B) What is its minimum total time?

Jobs in Increasing Order

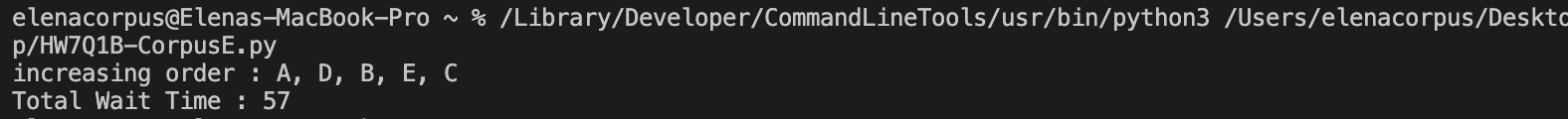
A, D, B, E, C

Shortest job First Algorihtm :

Waiting time for A = 0 = 0  
Waiting time for D = 0+4 = 4  
Waiting time for B = 0+4+5 = 9  
Waiting time for E = 0+4+5+8 = 17  
Waiting time for C = 0+4+5+8+10 = 27  
Total waiting time = 0+4+9+17+27 = 57

**Q1B [20, optional] Implementation of Q1.**

Write the Python (or Java) program for Q1.(1) - (3) to find both the optimal sequence of jobs and the minimum total time.



**Q2. [10] Prefix Code**

A code for a, b, c, d, e is given by ‘a=11’, ‘b=10’, ‘c=010’, ‘d=*x*01’, ‘e=*yz*0’, where *x, y, z* are either 0 or 1. Determine *x, y* and *z* so that the given code is a prefix code.

The prefix property says that No code is the prefix of any other code i.e among a, b, c, d, e none of them will be the prefix of each other.

So here we have, a = 11 , b = 10, c = 010 which are given already

The following code d and e are 3 digit codes so none of a, b, c should match the prefix of d and e.

d = x01, So what can we use here?

d = 101 (x=1 ?), but that matches the prefix b=10. So that's not right

d = 001 (x= 0 ?), yes since none of a= 11, b =10, c = 010 can match the prefix of d= 001.

So, x = 0.

e = yz0 , So what can we use here?

e = 100 , (y=1? and z = 0 ?), but that matches the prefix b = 10. So not right.

e = 110, (y=1? and z = 1 ?), but that matches the prefix a = 11. So not right.

e = 010, (y=0? and z = 1 ?), but that matches the prefix c = 010. So not right.

e = 000, (y=0? and z = 0 ?), which doesn't match any prefix in the data above. So that's right.

y=0 and z=0.

So, final answers : x = 0 , y =0 , z = 0.

**Q3. [30] Fractional Knapsack Problem**

An edited book has 6 articles. The table shows the lengths of the articles and their importance,

where the scale of importance is 1(low) to 10(high). The book must be at most 150 pages long.

The problem is ***to edit the book*** so that **the overall importance is maximized.**

1. [20] Edit the book by *choosing articles* whose pages and importance are given in the table. i.e. Give
2. [8] the list of the chosen articles with their chosen number of pages in the order,

C : 60

E : 10

B : 20

D : 15

F : 40

A : 30

1. [6] the importance for each chosen article and

C : 4

E : 8

B : 13

D : 16

F : 22

A : .5

1. [6] the total maximum importance of the edited book.

Total Maximum of importance of the edited book is 22.5

|  |  |  |  |
| --- | --- | --- | --- |
| **Article** | **Importance of article** |  | **Pages** |
| A | 3 |  | 30 |
| B | 5 |  | 20 |
| C | 4 |  | 60 |
| D | 3 |  | 15 |
| E  F | 4  6 |  | 10  40 |

1. [10] Consider a greedy rule for the above Fractional Knapsack Problem that selects the articles in non-decreasing order of pages. If the capacity of the knapsack is not exceeded, we take all of the pages. Otherwise, we take whatever portion of the article fills the knapsack and stop. Give an example to show that this greedy algorithm does not necessarily maximize the importance.

0 remaining pages

Importance of 22.33

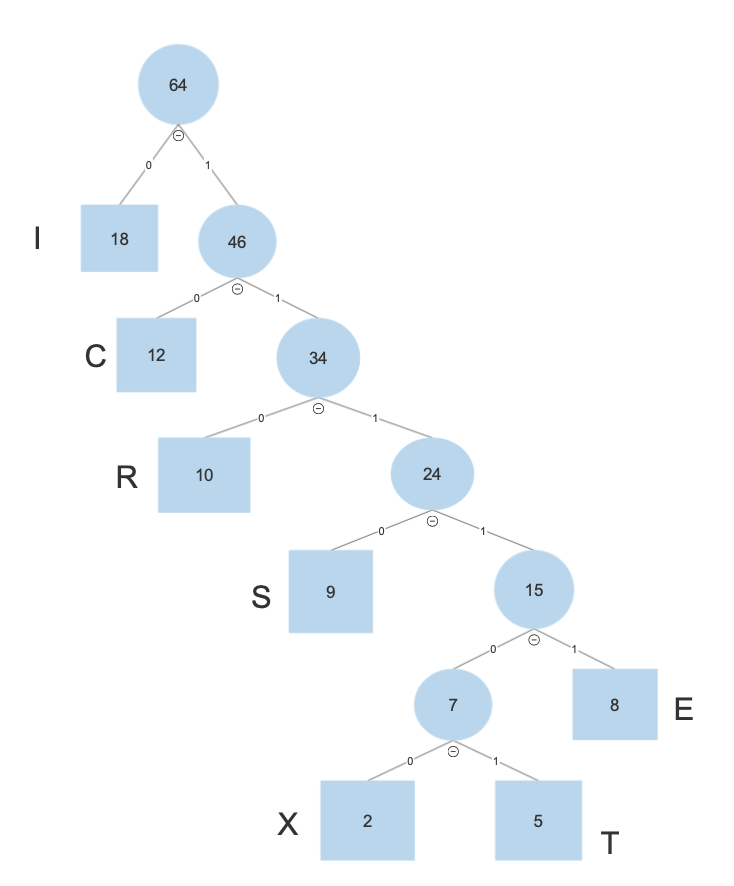
But maximum is is 22.5, so this greedy algorithm does not give the maximum imporance

**Q4. [50]** **Huffman Code**

1. [20] Using Huffman Algorithm, construct an optimal binary prefix code for the letters in the given frequency table.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Letter | C | E | I | R | S | T | X |
| Frequency | 12 | 8 | 18 | 10 | 9 | 5 | 2 |

1. [8] Draw your Huffman Tree (i.e. Encoding Tree), and



1. [7] Give an *optimal binary prefix code* for each letter.

I : 0

C : 10

R : 110

S : 1110

E : 11111

X : 111100

T : 111101

1. [5] Give the *total number of bits* required to encode the string given in the frequency table, i.e. the total path weight

12\*2+8\*5+18\*1+10\*3+9\*4+5\*6+2\*6 = 190

1. [20] Decode each bit string using the Huffman code in 1.

Assumption: The Huffman tree is constructed with the left branches and the right branches which are labeled with 0 and 1, respectively.

1. 011000101010100

IRIICCCCI

1. 1000100001010100

CIICIIICCCI

1. 111001001111110

SICIEC

1. 1000010011100

CIIICISI

1. [10] Encode each word using the Huffman code in 1.
2. RISE

1100111011111

1. EXIT

111111111000111101

1. TEXT

11110111111111100111101

1. EXERCISE

1111111110011111110100111011111

**Q4B. [30, optional] Implementation of Q4.**

Write a Python/Java program of Huffman coding to

1. generate the optimal Huffman codes for the letters in Q4.(1) with their frequencies,
2. encode the words in Q4.(3) and
3. decode the bit strings in Q4.(2).

