Homework 4

CS 325

Daniel Kim

Problem 1.
Write a concrete example of the knapsack problem where you specify a set of *at least* 5 objects, their dollar values (i.e., benefits) and their weights, as well as the weight of the knapsack, denoted W. Now, consider the greedy approach of *sorting items based on decreasing benefit/weight ratios and picking items from the beginning of the list.* In the context of your example, show that

Problem 1.a. (2 points)

- The greedy approach works for fractional knapsack.

Problem 1.b. (2 points)

- The greedy approach may fail for 0-1 knapsack.

---

a) **The greedy approach for fractional knapsack**

Sorted Decreasing Order

w_i / v_i	w_i	v_i	i
15.00	1	15	**3**
10.25	4	**41**	**4**
10.20	5	**51**	**5**
10.00	2	20	**2**
8.33	3	25	**1**

Fractional Knapsack

w_i / v_i	w_i	v_i	i
15	1	15	**3**
10.25	4	41	**4**
10.20	1	**1/5 * (51) = 10.20**	**5**

Total Weight 6
Total Value **66.2**

b) **Greedy approach fails 0/1 knapsack problem**. These values are only slightly different from last homework's values for the 0/1 Knapsack problem. Values for items 4 and 5 were incremented by 1 to make sorting in decreasing w/v order easier. **We see that the optimal value is 66 with items indexed at 3 and 5 with weights of 1 and 5** respectively, **but the greedy approach produces a value of 56** using only items of weight 1 and 4.

				Capacity = j						
	w_i	v_i	i	**0**	**1**	**2**	**3**	**4**	**5**	**6**
			0	**0**	0	0	0	0	0	0
8.33	3	25	**1**	0	0	0	25	25	25	25
10.00	2	20	**2**	0	0	20	25	25	45	45
15.00	1	15	**3**	0	**15**	20	**35**	40	45	**60**
10.25	4	41	**4**	0	**15**	20	35	40	**56**	**61**
10.20	5	51	**5**	0	15	20	35	40	**56**	**66**

Sorted Decreasing Order

w_i / v_i	w_i	v_i	i
15.00	1	15	**3**
10.25	4	**41**	**4**
10.20	5	**51**	**5**
10.00	2	20	**2**
8.33	3	25	**1**

0/1 Knapsack

w_i / v_i	w_i	v_i	i
15	1	15	**3**
10.25	4	41	**4**

Total Weight 5
Total Value **56**

Problem 2.
Consider the following symbols with their corresponding frequencies:

$A : 1, B : 1, C : 2, D : 3, E : 5, F : 8, G : 13, H : 21$

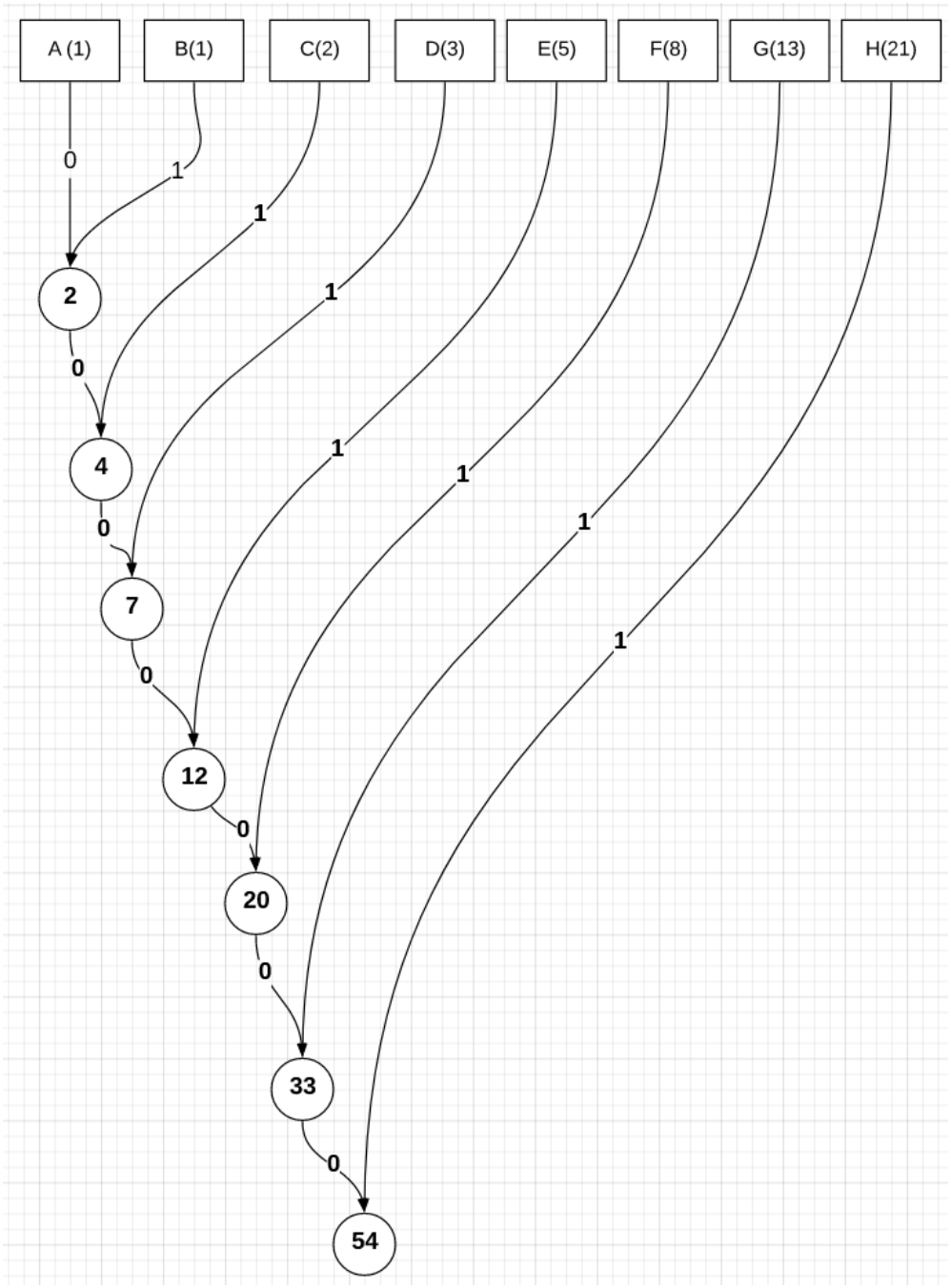
Problem 2.a. (3 points)

- Construct the Huffman coding of these symbols along with its optimal coding tree.

Problem 2.b. (3 points)

- Use your coding tree to decode 000100100001000000000001001

2a)



2b)

0001 001 00001 0000000 001 001

E F D A F F

Decoded message: EFDAFF

Char	Count	Code
A	1	0000000
B	1	0000001
C	2	000001
D	3	00001
E	5	0001
F	8	001
G	13	01
H	21	1

Problem 3.
Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

Problem 3.a. (4 points)

- Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are $c^0, c^1, \cdots, c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm of *picking the largest denomation first* always yields an optimal solution. You are expected to reason about why this approach gives an optimal solution. (*Hint*: Show that for each denomination c^i, the optimal solution must have less than c coins.)

Problem 3.b. (4 points)

- Design an $O(nk)$ time algorithm that makes change for any set of k different coin denominations, assuming that one of the coins is a penny.

For $c^0, c^1, \ldots, c^{k-1}, c^k$ denominations, the optimal solution will have the fewest coins to produce change for any amount **N**. To accomplish this, we can choose the highest denomination coins first. In other words, the number of coins of any denomination of c^i excluding c^k used is less than c where i < k.

- Let the optimal solution be represented by $(x_0, x_1, \ldots, x_k)$ where x_i is the number of coins of denomination c_i .
- We will show that we must have $x_i < c$ for $every\ i <$ k
- Suppose we had some optimal solution such that $x_i \geq c$, then we can decrease x_i by c and increase x_{i+1} by 1 (all denominations are divisible by c).
- The new collection of coins has c - 1 fewer coins. Thus, the original

solution must have not been optimal.

- This configuration is achieved and is the same when following our greedy choice of picking the largest possible denomination coins first.
- This is so because for a total value of N, we would pick x_k = floor[(N / c^k)] and for values of i < k , x_i = floor[(N mod c^{i+1})/ c^i]
- Therefore, the greedy algorithm will always produce the optimal solution for the denominations c^0, c^1, ..., c^{k-1}, c^k

3b)

c = array

c[n] = minimum number of coins required to represent a total value of n

denomination = array is used to trace back from index n the denominations of the coins used to represent the total value of n.

mChange(d, n)

for i <-- to n
  c[i] <-- α
  for j <-- 1 to k
    if i ≥ d_j
      if 1 + c[i – d_j] < c[i]
        c[i] <-- 1 + c[i -d_j]
    denomination[i] <-- d_j

return c and denomination

There are 2 for loops. The first look goes from i to n. The second for loop goes from 1 to k. Total running time is O(n*k)

Problem 4. (7 points)
Implementation: Implement the make change algorithm you designed in the previous problem. Your program should read a text file "data.txt" where each line in "data.txt" contains three values c, k and n. Please make sure you take your input in the specified order c, k and n. For example, a line in "data.txt" may look like the following:

3 4 38

where $c = 3, k = 4, n = 38$. That is, the set of denominations is $\{3^0, 3^1, 3^2, 3^3, 3^4\} = \{1, 3, 9, 27, 81\}$, and we would like to make change for $n = 38$. The file "data.txt" may include multiple lines like above.

The output will be written to a file called "change.txt", where the output corresponding to each input line contains a few lines. Each line has two numbers, where the first number denotes a denomination and the second number represents the cardinality of that denomination in the solution. For example, for the above input line '3 4 38', the optimal solution is the multiset $\{27, 9, 1, 1\}$, and the output in the file "change.txt" is as follows:

27 1

9 1

1 2

which means the solution contains 1 coin of denomination 27, one coin of 9 and two coins of denomination 1. You can use a delimiter line to separate the outputs generated for different input lines.

---

4. **Uploaded to TEACH. Thank you!**

Problem 5. (3 points)
Extra credit: Can you generalize the results you found in the construction of Problem 2 (i.e., the Huffman coding tree)? Write a statement/theorem that captures your generalization.

Submit a copy of all your code files and a README file that explains how to compile and run your code in a ZIP file to TEACH. We will only test execution with an input file named data.txt.

5. If you look at the values below, the frequencies of the characters in increasing order are the Fibonacci numbers where A starts as $\mathbf{F_1 = 1}$ and B is $\mathbf{F_2 = 1}$. The maximum digits of 0, 1's is the [total number of unique characters = n] – 1. We can write the recurrence for frequencies for the characters as $\sum_{i=0}^{n}(F_i + 1)$ where i = first index = 0 of character A.

As for the **n = total # of unique characters - 1** (n = 8 -1 =7for question 2).

Starting at i = 0 for the first character with least count, the first digits of 0's can be represented by **(n - i)** , where n is the number of unique characters - 1 and i is the index of the character.

The digit place of 1's can be represented by n – (i-1) if i > 0. For example, B is i = 1,

So the number of 0's is 8-1-1 = 6

# of unique characters

8			
# of zeroes	**digit location of 1**	**Index**	**Char**
n = **7**			a
6	**7**	**1**	b
5	**6**	**2**	c
4	**5**	**3**	d
3	**4**	**4**	e
2	**3**	**5**	f
1	**2**	**6**	g
0	**1**	**7**	h

Char	Count	Code
A	1	0000000
B	1	0000001
C	2	000001
D	3	00001
E	5	0001
F	8	001
G	13	01
H	21	1