CS 141 midterm2A

Celyna Su

TOTAL POINTS

72 / 90

QUESTION 1

1 Question1 **15 / 15**

✓ + **15 pts** **Huffman tree and codes are correct (A: one bit, B,C,D: four bits, E: two bits, F,G: five bits)**

+ **13 pts** Minor mistakes (8-13 points)

+ **7 pts** Major mistakes (2-7 points)

+ **1 pts** Completely incorrect

+ **0 pts** no answer or answer makes no sense

QUESTION 2

2 Question2 **15 / 15**

✓ + **5 pts** **(1) Algorithm sorts by bi/wi in decreasing order**

✓ + **5 pts** **(2) Algorithm takes as much as possible of the item with the highest bi/wi value**

✓ + **5 pts** **(3) Algorithm updates the residual capacity of the knapsack and repeat on the next item in sorted order if residual capacity > 0**

+ **0 pts** No answer or answer makes no sense

QUESTION 3

3 Question3 **10 / 15**

+ **15 pts** Greedy choice correct. Shows that a1^{b1} * ai^{bi} >= a1^{bi} * ai^{b1} because a1 > ai and b1 > bi using math

✓ + **10 pts** **Greedy choice proof somewhat correct, but not mathematically precise**

+ **5 pts** Some intuitive arguments about optimality or greedy choice

+ **0 pts** No answer or answer makes no sense

No sum, but products.

QUESTION 4

4 Question4 **15 / 15**

✓ + **13 pts** **Correct recurrence relation A) C[i,jk]=0 for i=j=k=0; B) C[i,j,k]= C[i-1,j-1,k-1] + 1 if i >0,j>0,k>0 and X[i]=Y[j]=Z[k]; C) C{i,j,k] = max {C[i-1,j,k],C[i,j-1,k],C[i,j,k-1]} otherwise**

+ **10 pts** Minor mistakes in the recurrence relation (6-10 points)

+ **5 pts** Major mistakes in the recurrence relations (1-5 points)

✓ + **1 pts** **Correct time complexity O(l m n) [providing this without the recurrence relation C[i,j,k] is just guessing]**

✓ + **1 pts** **Correct space complexity O(l m n) [no need for recurrence relation in this case]**

+ **0 pts** No answer or answer makes no sense

QUESTION 5

5 Question5 **12 / 15**

✓ + **15 pts** **Correct: number of symbols to insert = n - LCS(x,x^R)**

+ **15 pts** Correct: break the string into two halves (with a common middle element if odd), compute the LCS between the first half and the second half reversed, number of symbols to insert = n - LCS(first half x, second half x^R)

+ **15 pts** Correct dynamic programming algorithm

+ **10 pts** Computes LCS(x,x^R) but does not explicitly gives a formula for the number of symbols to be inserted or the algorithm is incorrect/unclear/inefficient

+ **5 pts** Mentions LCS

+ **0 pts** no answer or answer makes no sense

- 3 Point adjustment

overly-complicated and probably wrong as is, but correct intuition

QUESTION 6

6 Question6 5 / 15

+ **13 pts** Correct. C[i]=min_{0<=k<i} C{k]+1 if X[k+1..i] is palindrome

+ **10 pts** Minor mistakes in the recurrence relation (6-10 points)

✓ + **5 pts** **Major mistakes in the recurrence relations (1-5 points)**

+ **1 pts** Correct time complexity O(n^3) [providing this without the recurrence relation C[i] is just guessing]

✓ + **1 pts** **Correct space complexity O(n) [no need for recurrence relation in this case]**

+ **0 pts** No answer, or answer makes no sense

- 1 Point adjustment

💬 how can you get C[i] from C[i-1] and C[i+1]?

CS 141, Spring 2019 Midterm 2/A

First name: celyna

Last name: Su

Student ID: 862037643

- This exam is **closed book, closed notes**, 80 minutes long
- Read the questions carefully
- No electronic equipment allowed (cell phones, tablets, computers, ...)
- Write legibly. What can't be read will not be graded
- Use pseudocode, Python, or English to describe your algorithms (no C/C++/Java)
- When designing an algorithm, you are allowed to use any algorithm or data structure we explained in CS 141 or CS 14, without giving its details, unless the question specifically requires that you give such details
- Always remember to analyze the time complexity of your solution
- If you have a question about the meaning of a question, come to the front of the class

Problem 1. (15 points [greedy])

Draw the Huffman tree and find the optimal prefix code for the symbols in the following frequency table

symbol	frequency	code
A	20	0
B	5	1111
C	4	1110
D	3	1100
E	10	10
F	2	11011
G	1	11010

A 20
B 5
C 4
D 3
E 10
F 2
G 1 →

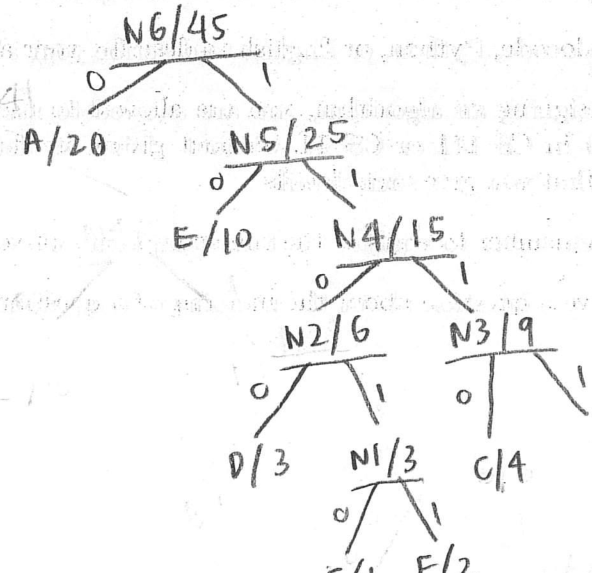
A 20
B 5
C 4
D 3
E 10
N1 3 →
(F+G

A 20
B 5
C 4 →
E 10
N2 6
(N1+D)

A 20
N3 9
(B+C)
E 10
N2 6 →

A 20
N4 15
(N3+N2)
E 10 →

A 20
N5 25 → N6 45
(N4+E)



Problem 2. (15 points [greedy])

In the fractional knapsack problem we discussed in class, we are supposed to choose among n items, where each item i has a positive benefit b_i and a positive weight w_i; we are also given the size of the knapsack W. The problem is to find the amount x_i of each item i which maximizes the total benefit $\sum_{i=1}^{n} x_i(b_i/w_i)$ under the condition that $0 \leq x_i \leq w_i$ and $\sum_{i=1}^{n} x_i \leq W$.

Write the pseudo-code for the greedy algorithm for fractional knapsack we discussed in class.

① Find the ratio $\overbrace{(b_i / w_i)}^{X_i}$ for each of n items

② sort and order in descending order (most valuable first)

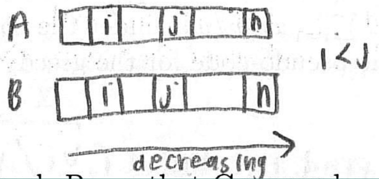
③ Take the item w/ the highest ratio and add to knapsack, and continuing doing so until you cannot add the next item as a whole (as it will exceed W)

④ Add the last item as much as possible (break it, fraction)

Problem 3. (15 points [greedy proof])

You are given two unsorted arrays $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$ of n distinct positive integers. The objective is to find an ordering of A and B so that $W = \prod_{i=1}^{n} a_i^{b_i}$ is maximized. Consider the following greedy algorithm.

```
Algorithm GREEDY(A : array, B : array)
    sort A in decreasing order
    sort B in decreasing order
    return (A, B)
```



We claim that the ordering computed by GREEDY is optimal. Prove that GREEDY has the greedy choice property for this problem.

$$\boxed{a_i \cdot b_i + a_j \cdot b_j \geq a_i \cdot b_j + a_j \cdot b_i}$$

$$a_i b_i - a_i b_j \geq a_j b_i - a_j b_j$$

$$a_i (b_i / b_j) \geq a_j (b/-b_j)$$

$$\boxed{a_i} \geq a_j$$

optimal

Greedy choice property:

$\boxed{\text{proof.}}$ let a_i be an optimal solution.

If there is an ordering of A and B s.t $W = \prod_{i=1}^{n} a_i^{b_i}$ is maximized then there must also be a a_j that is also greedy.

$B = (A - \{n\}) \cup \{1\}$

Problem 4. (15 points [dynamic programming])

We want to extend the LCS dynamic programming algorithm we covered in class to find the longest common subsequence between three strings X, Y and Z, where $|X| = l$, $|Y| = m$ and $|Z| = n$. Let X_i be a prefix of string X of length i where $0 \leq i \leq l$, Y_j be a prefix of string Y of length j where $0 \leq j \leq m$, and Z_k be a prefix of string Z of length k where $0 \leq k \leq n$. We define $C[i, j, k]$ as the length of the longest common subsequence between X_i, Y_j and Z_k. Then

$$C[i,j,k] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \text{ or k=0} \\ c[i-1, j-1, k-1] + 1 & \text{if } x_i = y_j = z_k \\ \max\{c[i-1, j, k], c[i, j-1, k], \\ \quad c[i, j, k-1]\} & \text{if otherwise} \end{cases}$$

The time complexity of this algorithm is $O(lmn)$

The space complexity of this algorithm is $O(lmn)$

Problem 5. (15 points [dynamic programming - black box])

A string Y is a *palindrome* if $Y^R = Y$, where Y^R is the reverse of Y. Given a string X of length n, we want to find the minimum number of characters that need to be inserted in X to make X a palindrome. For instance, X=`Ab3bd` can become `dAb3bAd` or `Adb3bdA` by inserting two characters (one `d`, one `A`). Give a $O(n^2)$-time dynamic programming algorithm for this problem.
Hint: Compute X^R and use one of the algorithms we discussed in class a black-box.

DP:
01 knapsack
lcs
counting combinations.

$O(\underset{\text{length}}{\underset{\uparrow}{n}} * \underset{\text{traversing str.}}{\underset{\uparrow}{n}})$

---

① Reverse the given string, denoted as X, and store as X^R.

② If empty, return 0.

③ else if $X[n-1] == X^R[\overset{\text{length of } X^R}{m}-1]$, return $1 + lcs(X, X^R, n-1, m-1)$

④ else: return max $(lcs(X, X^R, \underset{\overset{\wedge}{n-}}{n}, m-1), lcs(X, X^R, n-1, m))$

Problem 6. (15 points [dynamic programming])

A string Y is a *palindrome* if $Y^R = Y$, where Y^R is the reverse of Y. Given a string X a partitioning of X is a *palindrome partitioning* if every substring of the partition is a palindrome. For example, `aba|bbb|a|bb|a|b|aba` and `aba|b|bbabb|ababa` are two palindrome partitioning of X = `ababbbabbababa`. Design a dynamic programming algorithm to determine the coarsest (i.e., fewest cuts) palindrome partitioning of X. In the example, the second partition (3 cuts) is optimal. Remember to analyze the space- and time-complexity of your solution.

Hint: Define the dynamic programming table $C[i]$ to be number of cuts in the best palindrome partition of X_i, where X_i is the prefix of X of length i.

$$C[i] = \begin{cases} 0 & \text{If } i=0 \\ c[i-1]+1 & \text{if } X_i^R = X_i \\ \min\{c[i-1], c[i+1]\} & \text{If otherwise} \end{cases}$$

The time complexity is $0\ (n^2)$

The space complexity is $o\ (n)$