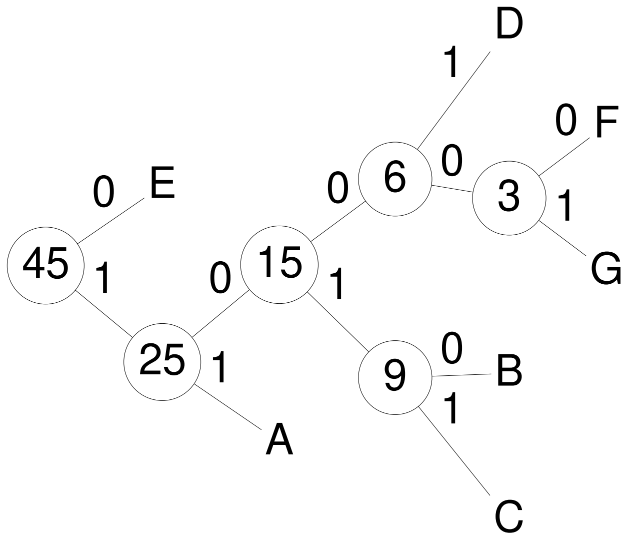
CS 141, Spring 2019 Midterm 2/B

Problem 1. (15 points [greedy])

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

Answer:

symbol	frequency	code
A	10	11
B	5	1010
C	4	1011
D	3	1001
E	20	0
F	1	10000
G	2	10001



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. push the n items in a priority queue Q sorted by b_i/w_i

```
2. let $w \leftarrow W$ be the residual capacity of the knapsack
3. while $w > 0$
4.     pop max item $j$ from $Q$ (the one with the largest benefit/weight ratio)
5.     take $x_j = \min\{w, w_j\}$ of item $j$
6.     let $w \leftarrow w - x_j$
```

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.

Greedy choice: The usual exchange argument. Consider any indices i and j such that $i < j$, and consider the terms $a_i^{b_i}$ and $a_j^{b_j}$. We want to show that the objective function W will not get worse by taking $a_i^{b_j}$ and $a_j^{b_i}$ instead. In other words, we need to show that $a_i^{b_i} a_j^{b_j} \geq a_i^{b_j} a_j^{b_i}$. Since A and B are sorted in decreasing order and $i < j$ we have $a_i \geq a_j$ and $b_i \geq b_j$. Since a_i and a_j are positive and $b_i - b_j$ is nonnegative, we have $a_i^{b_i - b_j} \geq a_j^{b_i - b_j}$. Multiplying both sides by $a_i^{b_j} a_j^{b_j}$ yields $a_i^{b_i} a_j^{b_j} \geq a_i^{b_j} a_j^{b_i}$.

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

$$C[i,j,k] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \text{ or k=0} \\ C[i-1, j-1, k-1] + 1 & \text{if } i > 0, j > 0, k > 0 \text{ and } X[i] = Y[j] = Z[k] \\ \max\{C[i-1,j,k], C[i,j-1,k], C[i,j,k-1]\} & \text{otherwise} \end{cases}$$

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`). Your algorithm just needs to report the number of characters (not which characters

or where they have to be inserted). 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.

Answer: Observe that $LCS(X, X^R)$ will give you the longest subsequence of X that is a palindrome. Therefore the number of characters that need to be inserted in X to make X a palindrome is $n - |LCS(X, X^R)|$. Time complexity if $O(n^2)$ because of LCS.

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.
Answer: We have

$$C[i] = \begin{cases} \min_{k \in [0, i-1]} \{C[k] + 1 : \text{if } X_{k+1\ldots i} \text{ is a palindrome}\} & \text{if } i > 0 \\ 0 & \text{if } i = 0 \end{cases}$$

This answer will give full credit, but runs in $O(n^3)$-time. Space is $O(n)$.

We can speed up the palindrome check if we precompute the following additional table. Define $P[i, j] =$ TRUE if $x_i x_{i+1} \ldots x_j$ and is a palindrome, for all $1 \leq i \leq j \leq n$.

$$P[i, j] = \begin{cases} x_i = x_j \text{ AND } P[i+1, j-1] & \text{if } i < j - 1 \\ x_i = x_j & \text{if } i = j - 1 \\ \text{TRUE} & \text{if } i = j \end{cases}$$

Then

$$C[i] = \begin{cases} \min_{k \in [0, i-1]} \{C[k] + 1 : P[k+1, i] = \text{TRUE}\} & \text{if } i > 0 \\ 0 & \text{if } i = 0 \end{cases}$$

The algorithm first computes $P[i, j]$ and then $C[i]$. The time complexity is $O(n^2)$. Space is $O(n^2)$.