Eric Yang

304263623

CS 180

Dis 1B

Homework 5

1. A. Let’s assume the coin denominations are d1 = 1, d2 = 5, and d3 = 8. If the input m is 10. The greedy algorithm would return 1 coin of value 8 and 2 coins of value 1. However, the optimal solution would have been two coins of value 5. So this counterexample shows that the greedy algorithm doesn’t work for all choices of coin denominations.

B. make array NumCoin with m entries

Initialize each entry as infinity

Make another array LastCoin with m entries

For i from 1 through k // input entries that only need 1 coin

NumCoin[ = 1

LastCoin[ = i

For each possible input n value up to m //find the min number of coins as change for each input up to m

For each i from 1 through k

If (NumCoin[n-] + 1) > NumCoin[n]

NumCoin[n] = NumCoin[n-] + 1

Create array Coin with each entry corresponding in order to

Initialize each entry in Coin to 0

While m is not 0 // find number of each coin to return

Coin[] = Coin[] + 1

m = m -

Return Coin

This algorithm works because we use dynamic programming to find the minimum number of coins to find change for m by looking at the minimum number of coins for each input less than m. By looking back, we can determine which coin to subtract from m. We use an array to hold all the minimum coin values. Then we keep another array to tell us the last coin used for each input and use this information to tell which value to subtract from the input m. We repeat this process until we get an array that tells us how many of each coin is used.

1. Given an array Tape with n entries for each file

Sort(Tape)

If size of Tape is 1, return Tape

Sort(first half of array)

Sort(second half of array)

Merge/Insert files into array in order of smallest to greatest

1. Given an array of the n words

Create table LineCost size n x n where LineCost[i][k] tells cost of having words n[i] to

n[k] on a single line

For each i from 1 to n //find number of extra spaces in each line in table

LineCost[i][i] = M -

For each k from 1 to n

LineCost[i][k] = LineCost[i][k-1] - – 1

For each i from 1 to n //find cost of each line in table

For each k from 1 to n

LineCost[i][k] = LineCost[i][k] \* LineCost[i][k] \* LineCost[i][k]

LineCost[i][n] = 0 // 0 cost if last line

Create array Print to hold index of first word of each line

Create array MinCost with n entries initialized to infinity

MinCost[0] = 0

For each i from 1 to n //find optimal paragraph

For each k from 1 to i

If MinCost[k-1] and LineCost[k][i] aren’t infinity

Temp = MinCost[k-1] + LineCost[k][i]

If Temp < MinCost[i]

MinCost[i] = Temp

Print [i] = k

Return Print

The complexity of this algorithm should be O(n^2) due to the nested for loops. The space requirement should be O(n^2) due to the table we use to keep all of the line costs.

1. This problem can be reduced to a Common Subsequence Problem

Given string of length n

Create an array X with each character of input string

Create second array Y with characters of input string in reverse

Create table Length of size n \* n

For k from 0 to n

Length[0][k] = 0

Length[k][0] = 0

For i from 1 to n

For j from 1 to n

If X[i] == Y[j]

Length[i][j] = Length[i-1][j-1] + 1

Else

Length[i][j] = max (Length[i-1][j], Length[i][j-1])

Return Length[n][n]