*Coin Change Dynamic Programming Assignment*

**Coin Changing Problem**

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

**a. Describe a dynamic programming to make change consisting of quarters, dimes, nickels, and pennies and prove that your algorithm yields an optimal solution. Implement your algorithm and test your solution.**

**Algorithm :**

**Objective**: Write a function to make change consisting of quarters, dimes, nickels and pennies for given ‘n’ value.

**Input**: integer n (change value) and denominations and length of denominations array

**Output**: minimum no. of coins after coin change.

**Assumptions**: All input values are integer and presented in form of cents.

MIN\_Coins(denom[], length, value)

instantiate an array of length (value+1); //table

for i = 1 to (value – 1)

table[i] = Integer.MaxValue

for i = 1 to (value - 1)

for j = 0 to (length – 1)

if coins[j] <= j

temp = table[i - coins[j]]  
 table[j] = minOf(temp+1, table[i])

return table[value]

**Output Results:**

**Input:**

denom[] = {1,2,4,8}  
value = 4

**Output:**

Total coins = 1

Proof:

The algorithm unlike greedy approach estimates the i+1 value using the previous values instead of calculating the previous values again. It stores the previously calculated values in an array for reusing them. For eg. To get the num of coins for 4 value it just adds +1 to table[i – count[j]] i.e. table[0] here.

The possible number of coins that can be returned can be 1(denom =4), 2(denom= 2,2), 4(denom = 4 of 1 each). But, the above algorithm returns only the optimum solution of 1.

**b. Describe a greedy algorithm to make change consisting of quarters, dimes, nickels, and pennies. Prove that your algorithm yields an optimal solution. Implement your algorithm and test your solution.**

**Algorithm :**

**Objective**: Write a function to make change consisting of quarters, dimes, nickels and pennies for given ‘n’ value.

**Input**: integer n (change value) and denominations

**Output**: minimum no. of coins after coin change.

**Assumptions**: All input values are integer and presented in form of cents.

Coin\_Change(denom[], value)

coinCount = 0   
While value > 0

For i 0 to denom[].length

If value > = denom[i]

Value -= denom[j]  
coinCount++  
break

Return coinCount

**Output Results:**

**Input:**

denom[] = {1,5,10,25}  
value = 10

**Output:**

Total coins = 1

Proof:

The algorithm compares the value and the denominations if it is greater than the denominations if not checks t=with the next denomination value and so on. Everytime it gets the expression true it increments the coincount value and decreases its value by the denomination value. It continues till the remaining value is 0.

The possible number of coins that can be returned for the above example can be 1(denom =10), 2(denom= 5,5), 6(denom = 5, 5 coins of 1 each) and so on but, the above algorithm returns only the optimum solution of 1.

**b. Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are c0, c1, ...., ck for some integers c>1 and k>= 1. Show that the greedy algorithm always yields an optimal solutions.**

**Ans.**

Assuming that available coins are of denomination c^0, c^1..c^k for some integers c>1 and k>=1 then for any given input say ‘n’, greedy algorithm always starts by choosing some denomination c^p such that p<k and p>0 ,that closest to input value ‘n’. And it continues to do so to find next closest fit for denomination such that final number of coins are minimun. Hence, we could conclude that it gives optimal solution.

**Code Files:**

**Dynamic Prog : DP\_CoinChange.java**

**Greedy Approach: CoinsChangeProblem.java**