Minimum Coin change Problem

Dynamic Programming Approach

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

In this approach we use a dynamic table ranging from 1 to amount for which change is required. This array contains the optimal number of coins required for each index. For calculating the coins for any amount k, starting from 1 through k, we use the values already computed to compute the next values thus avoiding re-computation of the same subproblem. We choose the minimum from $\{1 + C[p - di]\}$ if $p > 0$ where di is the denomination coin used, p is the change amount and C is the dynamic table. The base case is handled as 0 coins for 0 change amount.

Proof of correctness:

Let us consider the change amount to be 5, the possible denominations are pennies and nickels. As per the above approach we can either select 5 pennies or 1 nickel. The dynamic table entry for 4 is 4 coins of 1 penny each. So we will have to choose min{1+C[0], 1+C[4]} which is min{1,5}. Hence choosing a nickel is optimal since we have a coin of that denomination.

Let us consider the change amount to be 6, the possible denominations are pennies and nickels. If we choose a penny we would have 1+C[5] = 2 coins. If we choose a nickel we would have 1+C[1] = 2coins. Here either case we end up choosing 1 nickel and 1 penny. This trend continues till change amount 9.

Let us now consider a change amount of 10, the possible denominations that can be used are dimes, nickels and pennies. If we choose a penny we get 1+C[9] =1 +4 =5 coins. If we choose a nickel we get 1+C[5]=1+1 = 2coins. If we choose a dime we get 1+C[0] = 1 coin. We choose the option with minimum coins that is we choose a dime. Hence the Dynamic programming approach is optimal.

Time complexity is $\Theta(nk)$ where n is the amount for which change is required and k is total number of denominations.

Space complexity is $\Theta(n)$.

Greedy Algorithm

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

The approach used for solving this problem is very similar to what we do in real-life. We use the greatest value coins for the change amount first. This coin is used as many times as possible so that it does not exceed the change amount. The sum of these coins is deducted from the change amount. This remainder is the new change amount. The process is repeated. This is optimal since the largest amount coins are used first which eventually leads to a optimal solution with minimum number of coins.

Proof of Correctness:

Let us consider the case of using pennies and nickels. When the change amount is 5, either I can use 5 pennies or 1 nickel. 1 nickel is optimal. This operation would reduce the count of coins by 4 (if we were to choose pennies). In other words, when the amount is greater than or equal to 5 and we are allowed to choose only nickels and pennies we choose as many nickels as possible before checking for pennies.

On similar lines as above if we consider the case of using pennies, nickels and dimes. We can use at the most 1 nickel as 2 nickels equal a dime, so using a dime would be optimal. This operation would reduce the count of coins by 1 (if we were to choose only nickels). In other words, when the amount is greater than or equal to 10 and we are allowed to choose only dimes, nickels and pennies we choose as many dimes as possible before checking for nickels.

On similar lines as above if we consider the case of using pennies, nickels, dimes and quarters. We can use at the most 2 dimes as 3 dimes is greater than a quarter and so using a quarter would be optimal. This operation would reduce the count of coins by 1 (if we were to choose only dimes). In other words, when the amount is greater than or equal to 30 and we are allowed to choose only quarters, dimes, nickels and pennies we choose as many quarters as possible before checking for nickels.

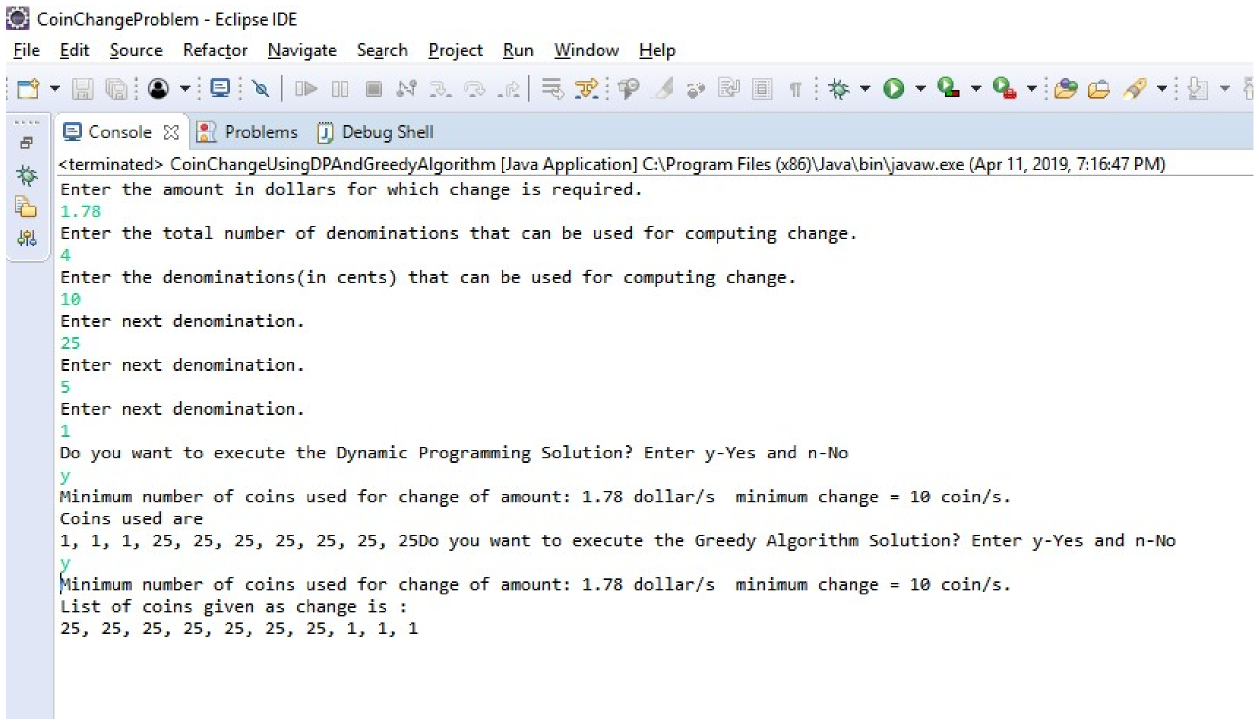
For the gap between 25 and 30, we could choose 2 dimes, 1 nickel and n-25 pennies or 1 quarter and n-25 pennies. But in line with the approach we would choose the latter, which is an optimal choice and reduces the coin total by 2. Proving that greedy algorithm is optimal.

Time complexity is Θ(klogk +n) where n is the amount for which change is required and k is total number of denominations. Which is approximately Θ(n).

C. Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are c^0, c^1, ..., c^k for some integers c>1 and k>= 1. Show that the greedy algorithm always yields an optimal solution.

1. The reasoning is very similar to the proof of greedy algorithm in the previous question. If we consider the first two types of coins that is c^0 and c^1. This means c^0 equals pennies and c^1 is the coin that is c cents. Let us call c^1 as c'. In line with the above discussion we can say, we can use at most c −1 pennies. As for c pennies or greater it is optimal to use the c' coin. This operation would reduce the total number of coins c −1. In other words, when the amount is greater than or equal to c and we are allowed to choose only c' and pennies we choose as many c' as possible before checking for pennies.
2. In continuation to this if we consider c^{n-1}' and c^{n}' coins. we can use at the most (c −1) c^{n-1}' coins. If the change amount is c^{n}' we would choose c^{n}' instead of c * c^{n-1}'coins . This operation would reduce the total number of coins c −1. In other words, when the amount is greater than or equal to c^{n}' and we are allowed to choose only c^{n}' , c^{n-1}' and smaller denomination coins, we choose as many c^{n}' as possible before checking for c^{n-1}'.
3. Thus we can conclude that greedy algorithm approach yields an optimal solution in this case as well.

Console Output


CoinChangeProblem - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Console Problems Debug Shell

```
<terminated> CoinChangeUsingDPAndGreedyAlgorithm [Java Application] C:\Program Files (x86)\Java\bin\javaw.exe (Apr 11, 2019, 7:16:47 PM)
Enter the amount in dollars for which change is required.
1.78
Enter the total number of denominations that can be used for computing change.
4
Enter the denominations(in cents) that can be used for computing change.
10
Enter next denomination.
25
Enter next denomination.
5
Enter next denomination.
1
Do you want to execute the Dynamic Programming Solution? Enter y-Yes and n-No
y
Minimum number of coins used for change of amount: 1.78 dollar/s  minimum change = 10 coin/s.
Coins used are
1, 1, 1, 25, 25, 25, 25, 25, 25, 25Do you want to execute the Greedy Algorithm Solution? Enter y-Yes and n-No
y
Minimum number of coins used for change of amount: 1.78 dollar/s  minimum change = 10 coin/s.
List of coins given as change is :
25, 25, 25, 25, 25, 25, 25, 1, 1, 1
```