EE3980 Algorithms

Homework 10. Coin Set Design II

By 105061212 王家駿

1. Introduction

In this homework, we are solving for the same problem as the previous homework, the coin set design problem. We used the greedy method in the last homework, but we could get the conclusion that this kind of method doesn't provide an optimal solution of the problem. Thus, we replace the greedy method by dynamic programming method (DP) to solve the problem, and compare the difference between the two results.

2. Implementation

Since we are solving for the same problem and four sub-problem, we just revise the code in finding the minimum coins needed for all prices in D and with all coin values known, which is the NCoinGreedy function in the previous homework. And the rest of our code remains the same.

2.1. Dynamic programming (DP)

```
1.  Algorithm NCoinDP(D, NCoin, Coins[])     // DP algorithm
2.  {
3.      // initialize
4.      cnt := 0;
5.      for i := 1 to D do table[i] = 0;
6.
```

```
7.      for i := 1 to D do {                // for value from 1 to D
8.          res := findMin(i, NCoin, Coins);// minimum coins needed
9.          table[i] := res;                // record value
10.         cnt := cnt + res;
11.     }
12.
13.     return cnt;
14. }
```

To implement the dynamic programming algorithm, we use an array table to store the number of minimum coins needed with regard to a specific price value. So, once we want to find the value, we could just reach the table directly if the value had found before.

In this function, we want to get the sum of minimum coins needed, so we iterate from 1 to D, and calculating the minimum value of each specific price. Once the value is found, we record it in the table. After the iteration goes to the end, the total minimum sum is found.

2.2. Find the minimum coins needed for a specific value

```
1.  Algorithm findMin(D, NCoin, Coins[])        // recursive function
2.  {
3.      if (table[D] != 0) return table[D];     // if value has been recorded
4.      if (NCoin = 1) return D;                // terminal condition
5.      res = findMin(D, NCoin - 1, Coins);     // number of target coins = 0
6.
7.      i = 1;
8.      // all possibility of number of target coins
9.      while (i * Coins[NCoin - 1] <= D) do {
10.         // recursive relation
```

```
11.         res_tmp := i + findMin(D - i * Coins[NCoin - 1], NCoin - 1, Coins);
12.         if (res_tmp < res) res = res_tmp;    // find minimum
13.         i := i + 1;
14.     }
15.
16.     return res;
17. }
```

In this function, we get the minimum number of coins needed for a specific price value.

To solve the problem, we use a recurrence relation:

$$g_1(\mathrm{D}) = \mathrm{D}$$

$$g_n(\mathrm{D}) = \min_{x_n \text{ from 1 to } \mathrm{D}/c_n} \{x_n + g_{n-1}(\mathrm{D} - x_n c_n)\} \quad \text{for n} > 1$$

where g_n represents the minimum number of coins needed, using n types of coins from Coins[0] to Coins[n-1], x_n represents the number of the target coin, and c_n represents the value of the target coin.

If there is only one type of coins, the minimum number of coins is obvious D because we could only select one-dollar coins.

If there are n types of coins, since we have known the minimum value of the case of n-1 types, we could only focus on the nth type of coin, the target coin. To ensure all cases are concerned, we list all the possibility of number of the target coin. In the case we take x_n target coins, there are $\mathrm{D} - x_n c_n$ dollars remained, so the next step is to

find the minimum number of coins of the rest value. And then we find the minimum above all cases of x_n. That's the result.

The characteristic of dynamic programming is added at line 3. If the value had found before and recorded in the table, we could just reach the table and skip the rest of the function, which could reduce the instructions to be executed.

3. Executing results

We run the program and record the results of minimum values found, and compare the result with the one we got in the previous homework.

Sub-problem	**Greedy**		**Dynamic Programming**	
	Value of each coin	Average	Value of each coin	Average
1	{1, 5, 10, 50}	5.05051	{1, 5, 10, 50}	5.05051
2	{1, 5, 10, 28}	4.54545	{1, 5, 10, 22}	4.30303
3	{1, 5, 14, 50}	4.52525	{1, 5, 12, 50}	4.32323
4	{1, 5, 13, 42}	4.24242	{1, 5, 18, 25}	3.92929

4. Result analysis and conclusion

From the results, we could observe that we get the same results in the sub-problem 1 and different result in the rest sub-problems, which meets the conclusion we got in the last homework.

For the last three cases, the dynamic programming method could reduce the average coins needed further. We are not able to find the optimal solution by greedy method, but we could use dynamic programming method to solve the problem, and we also show that it is exactly the optimal solution in section 2.1 and 2.2. So, we could easily solve the problem by merely revising the part of finding the minimum coins needed for a specific price value.

However, to calculate the minimum coins needed for a specific price value, the dynamic method would be much slower than the greedy method. In the greedy method we provided in the last homework, there were just some integer calculations like divisions and remainders; while in the dynamic programming method, we need more time and space complexity since we need recursive function calls, even though we use dynamic programming to reduce the instructions to be executed. Thus, in the case of large D or large NCoin, if we are more concerned about the time consumed, greedy method is still a feasible solution to effectively reduce the number of coins needed to some extent.