Change Making Problem

June 5, 2018

Student: Vieru Bogdan

Computers and Information Technology
(Engligh Language)

Group 1.3 B

1st Year

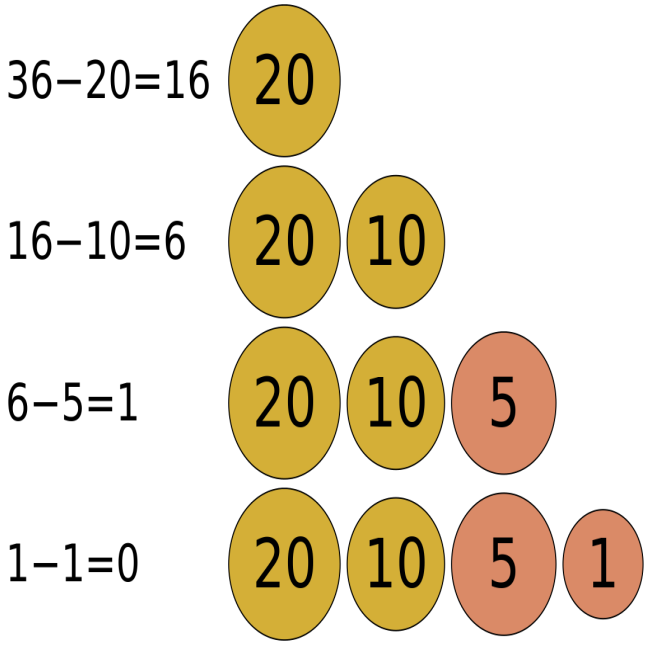
Introduction

Dynamic programming is both a mathematical optimization method and a computer programming method. The method was developed by Richard Bellman in the 1950s and has found applications in numerous fields, from aerospace engineering to economics. In both contexts it refers to simplifying a complicated problem by breaking it down into simpler sub-problems in a recursive manner. While some decision problems cannot be taken apart this way, decisions that span several points in time do often break apart recursively. Likewise, in computer science, a problem that can be solved optimally by breaking it into sub-problems and then recursively finding the optimal solutions to the sub-problems is said to have optimal substructure. If sub-problems can be nested recursively inside larger problems, so that dynamic programming methods are applicable, then there is a relation between the value of the larger problem and the values of the sub-problems.[1] In the optimization literature this relationship is called the Bellman equation.

A greedy algorithm is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage[1] with the intent of finding a global optimum. In many problems, a greedy strategy does not usually produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a globally optimal solution in a reasonable amount of time.

Problem statement

Implement 2 methods to find the minimum number of coins of given denominations needed for a given amount of money.



Pseudocode

```
coinChange(int denomination[NCoins+1], int nrcoins[Amount+1], int
    lastcoin[Amount+1])
1.      nrcoins[0] <- 0
2.      lastcoin[0] <- 0
3.      for iterator1 <- 1 to Amount do
4.          min <- INF;
5.      for iterator2 <- 1 to NCoins do
6.          if denomination[iterator2] <= iterator1 do
7.              if(1 + nrcoins[iterator1 - denomination[iterator2]] <
    min) do
8.                      min <- 1 +nrcoins[iterator1 -
    denomination[iterator2]]
9.                       coin <- iterator2
10.     nrcoins[iterator1] <- min
11.     lastcoin[iterator] <- coin
```

```
coinSet(int denominations[NCoins+1], int lastcoin[Amount+1])
1.  while iterator > 0 do
2.      print "Use coin of denomination %d " ,
    denominations[lastcoin[iterator]]
3.      iterator <- iterator 0 denominations[lastcoin[iterator]]
```

```
greedy(int denomination[NCoins+1] , int coins[Amount+1])
1.  iterator_coins <-
2.  amount_aux <- Amount
3.  for iterator <- 1 to NCoins+1 do
4.      idenomination[iterator]<-denomination[NCoins+1 - iterator]
5.  iterator <- 1
6.  while iterator <= Ncoins do
7.      if idenomination[iterator] <= amount_aux do
8.          no_coins <- amount_aux / idenomination[iterator]
9.          print idenomination[iterator] is used no_coins times
10.         amount_aux <- amount_aux - no_coins*idenomination[iterator]
11.         iterator_coins <- iterator_coins + no_coins
12.     iterator++
13. print The min no. of coins that can be used : iterator_coins
```

Application design

The library contains the header coinChange.h which has the functions prototypes required for the Dynamic Programming method :

-void coinChange(int denomination[NCoins+1], int nrcoins[Amount+1], int lastcoin[Amount+1])

-void coinSet(int denominations[NCoins+1], int lastcoin[Amount+1])

The source file coinChange.c contains the implementations for the above functions:

Function coinChange allows us to determine the minimum number of coins needed for a given amount of money storing it in an array called nrcoins and the last coin used to form every amount from 1 to the given amount storing it in the lastcoin array . It computes the minimum number of coins needed for the amount by going through every value from 1 to the given amount and through every denomination checking if there is a denomination smaller than the current value and how many times it fits in the value , in the end it will store the minimum number of coins needed for an amount and the last used coin.

Function coinSet allows us to go through the lastcoin array to print the optimal solution , the denominations that were used and how many times were them used . It starts at the final element of the lastcoin array and then it subtracts from the iterator the current element from the array until our iterator is smaller or equal to 0.

The header greedy.h contains the function prototype for the Greedy method: -void greedy(int denomination[NCoins+1] , int coins[Amount+1])

The source file greedy.c contains the implementation for the above function:

Function greedy first puts the denominations in descending order , then goes through the denominations , if the current denomination is smaller the amount we need to form we find how many times it fits in the amount by dividing the amount by the denomination and saving it in the nocoins variable , then we subtract the amount that we get by multiplying the nocoins with the denomination .

Source Code

```
//----------------------------coinChange.h-------------------------------
#ifndef COINCHANGE_H_INCLUDED
#define COINCHANGE_H_INCLUDED

#define INF 32517
#define NCoins 4 // number of coins
#define Amount 68 // amount we want to change


void coinChange(int denomination[NCoins+1], int nrcoins[Amount+1], int
    lastcoin[Amount+1]);
void coinSet(int denominations[NCoins+1], int lastcoin[Amount+1]);



#endif // COINCHANGE_H_INCLUDED
```

```
//----------------------------coinChange.c-------------------------------

#include <stdio.h>
#include "coinChange.h"

#define INF 32517
#define NCoins 4 // number of coins
#define Amount 68 // amount we want to change



void coinChange(int denomination[NCoins+1], int nrcoins[Amount+1], int
    lastcoin[Amount+1]){
       int iterator1, iterator2, min, coin;
   /**
   * no. of coins used when the amount is 0
   * last coin used when the amount is 0
   */
       nrcoins[0] = 0;
       lastcoin[0] = 0;
   /**
   * iterator1 is helping us going through every value from 1 to the
       given amount
   */
       for(iterator1 = 1; iterator1 <= Amount; iterator1 ++) {
              min = INF;
              /**
              * iterator2 is used to go through every coin value of the
                  denomination
```

```
            */
            for( iterator2 = 1; iterator2 <= NCoins; iterator2++) {
                   if(denomination[iterator2] <= iterator1) {
                          if(1 + nrcoins[iterator1 -
                              denomination[iterator2]] < min) {
                                 min = 1 + nrcoins[iterator1 -
                                     denomination[iterator2]];
                      coin = iterator2 ;
                          }
                      }
            }
            nrcoins[iterator1] = min;
            lastcoin[iterator1] = coin;
      }
```

```
//-----------------------------greedy.h------------------------------

#ifndef GREEDY_H_INCLUDED
#define GREEDY_H_INCLUDED


#define INF 32517
#define NCoins 4 // number of coins
#define Amount 68 // amount we want to change

void greedy(int denomination[NCoins + 1] , int coins[Amount + 1]);



#endif // GREEDY_H_INCLUDED
```

```
//-----------------------------greedy.c------------------------------


#include <stdio.h>
#include "greedy.h"

#define INF 32517
#define NCoins 4 // number of coins
#define Amount 68 // amount we want to change

/**
* Function greedy rst puts the denominations in descending order , then
    goes through the denominations
* if the current denomination is smaller the amount we need to form we
    nd how many times it ts in the amount by dividing the amount by the
    denomination and saving it in the nocoins variable ,
* then we subtract the amount that we get by multiplying the nocoins
```

```
    with the denomination .
*/

void greedy(int denomination[NCoins+1] , int coins[Amount+1]){
    int iterator_coins=0,iterator , no_coins , idenomination[NCoins+1] ,
        amount_aux=Amount ;

    for(iterator = 1 ; iterator <= NCoins+1 ; iterator ++)
        idenomination[iterator]=denomination[NCoins + 1 - iterator];

    iterator = 1;
    while(iterator <= NCoins){
        if(idenomination[iterator] <= amount_aux){
            no_coins = amount_aux/idenomination[iterator];
            printf("%d coin is used %d times
                \n",idenomination[iterator],no_coins);
            amount_aux = amount_aux - no_coins*idenomination[iterator];
            iterator_coins = iterator_coins + no_coins;
        }
        iterator++;
    }
    printf("\n The min no. of coins that can be used : %d
        \n",iterator_coins);
}
```

```
//--------------------------------main.c--------------------------------

#include <stdio.h>
#include <stdlib.h>
#include "coinChange.h"
#include "greedy.h"
#define INF 32517
#define NCoins 4 // number of coins
#define Amount 68 // amount we want to change



int main()
{
    printf("\n===  Amount=%d  ===\n",Amount);
    /**
    * value of the coins , we start from index 1 so first value is 0
    */

        int denomination[NCoins+1] = {0, 1, 5, 10, 25};
    /**
    * min. no. of coins required
    */
```

```
        int nrcoins[Amount+1];
    /**
    * last coin we used to make change for index amount
    */
        int lastcoin[Amount+1];

        int coins[Amount+1];

        printf("\n The denominations : \n");

        arr_display(denomination , NCoins);

        printf("\n===Greedy Method===\n");

        greedy(denomination,coins);

        printf("\n===Dynamic Programming===\n");
    /**
    *  compute minimum number of coins required
    */
        coinChange(denomination, nrcoins, lastcoin);
        printf("Min. no. of coins required to change %d = %d\n", Amount,
            nrcoins[Amount]);
    /**
    * print the coins used to make change.
    */
        coinSet(denomination, lastcoin);


        return 0;
}



void arr_display(int *arr , int arr_length){
    int iterator=0;

    for(iterator=1;iterator<=arr_length;iterator++){
        printf("%d \n",*(arr + iterator));
    }

}
```

Experiments and results

```
===   Amount=189   ===

 The denominations :
1
5
12
25
50
61

===Greedy Method===
61 coin is used 3 times
5 coin is used 1 times
1 coin is used 1 times

 The min no. of coins that can be used : 5

===Dynamic Programming===
Min. no. of coins required to change 189 = 5
Use coin of denomination: 1
Use coin of denomination: 5
Use coin of denomination: 61
Use coin of denomination: 61
Use coin of denomination: 61

Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

```
===   Amount=36   ===

 The denominations :
1
5
10
25

===Greedy Method===
25 coin is used 1 times
10 coin is used 1 times
1 coin is used 1 times

 The min no. of coins that can be used : 3

===Dynamic Programming===
Min. no. of coins required to change 36 = 3
Use coin of denomination: 1
Use coin of denomination: 10
Use coin of denomination: 25

Process returned 0 (0x0)   execution time : 0.030 s
Press any key to continue.
```

Conclusions

This project made me realise that no matter how simple a problem seems , it can always cause a lot of trouble implementing an optimal solution .

References

1)www.stackoverflow.com

2)https://www.dyclassroom.com

3)https://www.geeksforgeeks.org

4)http://www.sharelatex.com