



Coin Changing

Javier Esparza

Computer Science
School of Informatics
The University of Edinburgh

The problem

Given a coin set consisting of k coins with values $c_0 > c_1 > \dots c_{k-1}$, and an amount a to be changed, find non-negative integers $n_0, n_1 \dots n_{k-1}$ so that $\sum_{i=0}^{k-1} n_i c_i = a$. In addition there should be no other non-negative integers m_0, \dots, m_{k-1} such that $\sum_{i=0}^{k-1} m_i c_i = a$ and $\sum_{i=0}^{k-1} m_i < \sum_{i=0}^{k-1} n_i$.

The divide-and-conquer approach

We can split the potential solutions into two disjoint classes:

Use c_0 at least once. In this case the number of coins required is 1 (for a use of c_0) plus the number of coins needed to solve changing $a - c_0$ using c_0, \dots, c_{k-1} .

Do not use c_0 . In this case we just need to know the number of coins needed to change a using c_1, \dots, c_{k-1} .

The solution is the minimum of these two quantities.
(In the case $a < c_0$ only the 'do not use c_1 ' option is available)

-
- Two basis cases for the recursion:
 - When a is zero we need no coins.
 - When a is non-zero and we have no coins to offer change with the answer is infinity. In our implementation we just use `Integer.MAX_VALUE`.
 - Termination guarantee:
 - The parameter $(a + k)$, always decreases for either of the steps (use c_0 or not), and it cannot decrease for ever without reaching one of the basis cases.
-

```
class Change {  
    private static int[] c;  
  
    public static int change(int amount, int[] coins){  
        c = coins;  
        return change(amount, 0);  
    }  
    private static int change(int amount, int j) {  
        if (amount == 0) return(0);  
        if (j == c.length) return(Integer.MAX_VALUE);  
        if (amount < c[j]) return(change(amount, j+1));  
        else {  
            int c1 = change(amount, j+1);  
            int c2 = 1 + change(amount-c[j], j);  
            if (c1 < c2) return(c1);  
            else return(c2);  
        }  
    }  
}
```

The greedy strategy

Always try to give as many large coins as possible before considering smaller coins.

```
public static int change(int amount, int[] coins){
    int noCoins = 0;

    for (int i=0; i<coins.length; i++) {
        if (amount >= coins[i]) {
            noCoins += amount / coins[i];
            amount %= coins[i];
        }
    }
    if (amount > 0) return(Integer.MAX_VALUE);
    else return(noCoins);
}
```

```
public static int change(int amount, int[] coins){
    int[][] tab = new int[amount+1][coins.length+1];

    for(int i = 0; i <= amount; i++)
        tab[i][coins.length] = Integer.MAX_VALUE;
    for(int j = 0; j < coins.length; j++)
        tab[0][j] = 0;

    for(int i = 1; i <= amount; i++) {
        for(int j = coins.length-1; j >= 0; j--) {
            if (i < coins[j]) tab[i][j] = tab[i][j+1];
            else {
                int c1 = tab[i][j+1];
                int c2 = 1 + tab[i-coins[j]][j];
                if (c1 < c2) tab[i][j] = c1;
                else tab[i][j] = c2;
            }
        }
    }
    return(tab[amount][0]);
}
```

```
public static int change(int amount, int[] coins){
    int[] tab = new int[amount+1];

    tab[0]=0;
    for(int i = 1; i <= amount; i++) {
        tab[i] = Integer.MAX_VALUE;
        for(int j = coins.length-1; j >= 0 ; j--) {
            if (i >= coins[j]){
                int c = 1 + tab[i-coins[j]];
                if (c < tab[i]) tab[i] = c;
            }
        }
    }
    return(tab[amount]);
}
```

```
public static int change(int amount, int[] coins){
    int k = coins[0]+1;
    int[] tab = new int[k];

    tab[0]=0;
    for(int i = 1; i <= amount; i++) {
        tab[i%k] = Integer.MAX_VALUE;
        for(int j = coins.length-1; j >= 0 ; j--) {
            if (i >= coins[j]){
                int c = 1 + tab[(i-coins[j])%k];
                if (c < tab[i%k]) tab[i%k] = c;
            }
        }
    }
    return(tab[amount%k]);
}
```