Paradigmas de Resolução de Problemas

Programação Dinâmica – Problema do Troco: Exercícios Resolvidos

Prof. Edson Alves - UnB/FGA

2020

Sumário

OJ 166 – Making Change

Problema

Given an amount of money and unlimited (almost) numbers of coins (we will ignore notes for this problem) we know that an amount of money may be made up in a variety of ways. A more interesting problem arises when goods are bought and need to be paid for, with the possibility that change may need to be given. Given the finite resources of most wallets nowadays, we are constrained in the number of ways in which we can make up an amount to pay for our purchases – assuming that we can make up the amount in the first place, but that is another story.

Problema

The problem we will be concerned with will be to minimise the number of coins that change hands at such a transaction, given that the shopkeeper has an adequate supply of all coins. (The set of New Zealand coins comprises $5c, 10c, 20c, 50c, \$1$ and $\$2$.) Thus if we need to pay $55c$, and we do not hold a $50c$ coin, we could pay this as $2 \times 20c + 10c + 5c$ to make a total of 4 coins. If we tender $\$1$ we will receive $45c$ in change which also involves 4 coins, but if we tender $\$1.05(\$1 + 5c)$, we get $50c$ change and the total number of coins that changes hands is only 3.

Write a program that will read in the resources available to you and the amount of the purchase and will determine the minimum number of coins that change hands.

Entrada e saída

Input

Input will consist of a series of lines, each line defining a different situation. Each line will consist of 6 integers representing the numbers of coins available to you in the order given above, followed by a real number representing the value of the transaction, which will always be less than $\$5.00$. The file will be terminated by six zeroes $(0\ 0\ 0\ 0\ 0\ 0)$. The total value of the coins will always be sufficient to make up the amount and the amount will always be achievable, that is it will always be a multiple of $5c$.

Output

Output will consist of a series of lines, one for each situation defined in the input. Each line will consist of the minimum number of coins that change hands right justified in a field 3 characters wide.

Exemplo de entradas e saídas

Sample Input

```
2 4 2 2 1 0 0.95
2 4 2 0 1 0 0.55
0 0 0 0 0 0
```

Sample Output

```
2
3
```

Solução $O(NM)$

- Seja $m \geq C$ a quantidade paga ao caixa, onde C é o valor da conta, em centavos
- Defina $G(m, xs)$ como o número mínimo de moedas utilizadas para pagar m usando xs, onde

$$xs = \{x_1, x_2, \ldots, x_N\}$$

  e x_i é o número de moedas disponíveis do tipo c_i
- Para o cidadão xs é dado na entrada
- Para o caixa, $xs = ys$, com $y_i = \infty$ para todo $i \in [1, N]$
- Caso não seja possível dar um troco para m usando xs, faça $G(m, xs) = \infty$

Solução $O(NM)$

- Quando o cidadão paga m ao caixa, ele recebe $C - m$ de troco
- Utilizando a notação descrita, a solução S do problema é dada por

$$S = \min_i \{G(C + 5i, xs) + G(5i, ys)\},$$

  onde $i \in [0, (M - C)/5]$ e M é o maior valor possível para a conta
- Como $G(m, xs)$ tem complexidade $O(N)$, a solução proposta tem complexidade $O(NM)$ por caso de teste

Solução $O(NM)$

```cpp
#include <bits/stdc++.h>

using namespace std;

const int oo { 1000000010 }, N = 6;
const vector<int> cs { 5, 10, 20, 50, 100, 200 }, ys(cs.size(), oo);

int greedy(int m, const vector<int>& qs)
{
    int res = 0;

    for (int i = N - 1; i >= 0; --i)
    {
        auto k = min(qs[i], m / cs[i]);
        m -= k*cs[i];
        res += k;
    }

    return m > 0 ? oo : res;
}

```

Solução $O(NM)$

```
22 int solve(int M, const vector<int>& xs)
23 {
24     int total = 0, ans = oo;
25
26     for (int i = 0; i < N; ++i)
27         total += xs[i] * cs[i];
28
29     for (int m = M; m <= total; m += 5)
30         ans = min(ans, greedy(m, xs) + greedy(m - M, ys));
31
32     return ans;
33 }
34
35 int main()
36 {
37     while (true)
38     {
39         vector<int> xs(6);
40
41         for (int i = 0; i < N; ++i)
42             scanf("%d", &xs[i]);
```

Solução $O(NM)$

```
43
44          if (accumulate(xs.begin(), xs.end(), 0) == 0)
45              break;
46
47          int d, c;
48          scanf("%d.%d", &d, &c);
49
50          auto ans = solve(100*d + c, xs);
51
52          printf("%3d\n", ans);
53      }
54
55      return 0;
56  }
```

OJ 147 – Dollars

Problema

New Zealand currency consists of $\$100, \$50, \$20, \$10$, and $\$5$ notes and $\$2, \$1, 50c, 20c, 10c$ and $5c$ coins. Write a program that will determine, for any given amount, in how many ways that amount may be made up. Changing the order of listing does not increase the count. Thus $20c$ may be made up in 4 ways: $1 \times 20c, 2 \times 10c, 10c + 2 \times 5c$, and $4 \times 5c$.

Entrada e saída

Input

Input will consist of a series of real numbers no greater than $\$300.00$ each on a separate line. Each amount will be valid, that is will be a multiple of $5c$. The file will be terminated by a line containing zero (0.00).

Output

Output will consist of a line for each of the amounts in the input, each line consisting of the amount of money (with two decimal places and right justified in a field of width 6), followed by the number of ways in which that amount may be made up, right justified in a field of width 17.

Exemplo de entradas e saídas

Sample Input

```
0.20
  2.00
0.00
```

Sample Output

```
   0.20                 4
   2.00               293
```

Solução $O(NM)$

- Este problema é uma variante do problema do troco, onde se deseja obter o número de maneiras distintas de se dar um mesmo troco m utilizando-se as moedas disponíveis
- Deste modo, é preciso adaptar o algoritmo de programação dinâmica para obter este resultado
- Importante notar que a ordem não é levada em consideração, de modo que $\{0.25, 0.50, 0.25\}$ e $\{0.50, 0.25, 0.25\}$ são consideradas uma mesma maneira de dar o troco de $\$1$
- Seja $dp(m, i)$ o número de maneiras distintas de se dar o troco m utilizando-se apenas as i menores moedas
- São dois casos-base: $dp(0, 0) = 1$ e $dp(m, 0) = 0$, se $m > 0$

Solução $O(NM)$

- São duas transições possíveis: utilizar a i-ésima menor moeda ou não, sendo que ambas alternativas devem ser totalizadas
- Assim,

$$dp(m, i) = dp(m - c_i, i) + dp(m, i - 1),$$

  se $c_i \leq m$, ou $dp(m, i) = dp(m, i - 1)$, caso contrário
- Se as moedas forem processadas em ordem crescente, é possível deixar ímplicita a segunda dimensão do estado, reduzindo o uso de memória e simplificando a implementação
- A tabela dos estados pode ser preenchida em $O(NM)$, e após este preenchimento cada caso de teste pode ser respondido em $O(1)$, onde M é o maior troco possível e N o número de moedas distintas

Solução $O(NM)$

```cpp
#include <bits/stdc++.h>

using namespace std;
using ll = long long;

const int MAX { 30010 };
const vector<int> cs { 10000, 5000, 2000, 1000, 500, 200,
    100, 50, 20, 10, 5 };

ll st[MAX];

void precomp()
{
    st[0] = 1;

    for (auto c : cs)
        for (int m = c; m < MAX; m += 5)
                st[m] += st[m - c];
}
```

Solução $O(NM)$

```
20
21 int main()
22 {
23     precomp();
24
25     int d, c;
26
27     while (scanf("%d.%d", &d, &c) == 2 and d + c)
28         printf("%3d.%02d%17lld\n", d, c, st[100*d + c]);
29
30     return 0;
31 }
```

Codeforces Beta Round #10 – Problem E: Greedy Change

Problema

Billy investigates the question of applying greedy algorithm to different spheres of life. At the moment he is studying the application of greedy algorithm to the problem about change. There is an amount of n coins of different face values, and the coins of each value are not limited in number. The task is to collect the sum x with the minimum amount of coins. Greedy algorithm with each its step takes the coin of the highest face value, not exceeding x. Obviously, if among the coins' face values exists the face value 1, any sum x can be collected with the help of greedy algorithm.

Problema

However, greedy algorithm does not always give the optimal representation of the sum, i.e. the representation with the minimum amount of coins. For example, if there are face values $\{1, 3, 4\}$ and it is asked to collect the sum 6, greedy algorithm will represent the sum as $4 + 1 + 1$, while the optimal representation is $3 + 3$, containing one coin less. By the given set of face values find out if there exist such a sum x that greedy algorithm will collect in a non-optimal way. If such a sum exists, find out the smallest of these sums.

Entrada e saída

Input

The first line contains an integer n ($1 \leq n \leq 400$) – the amount of the coins' face values. The second line contains n integers a_i ($1 \leq a_i \leq 10^9$), describing the face values. It is guaranteed that $a_1 > a_2 > \ldots > a_n$ and $a_n = 1$.

Output

If greedy algorithm collects any sum in an optimal way, output -1. Otherwise output the smallest sum that greedy algorithm collects in a non-optimal way.

Exemplo de entradas e saídas

Sample Input	Sample Output
5 25 10 5 2 1	-1
3 4 3 1	6

Solução $O(N^3)$

- O problema consiste em identificar o menor dentre os contraexemplos, caso a base seja não-canônica
- O algoritmo de Pearson resolve justamente este problema, com complexidade $O(N^3)$
- Em linhas gerais ele avalia $O(N^2)$ candidatos à menor contraexemplo, checando cada um deles em $O(N)$
- Se nenhum deles se confirmar, a base é canônica e a resposta será -1
- Dentre os candidatos que se confirmarem como contraexemplos, o menor deles será a solução do problema

Solução $O(N^3)$

```cpp
#include <bits/stdc++.h>

using namespace std;

const int oo { 2000000007 };

vector<int> greedy(int x, int N, const vector<int>& xs)
{
    vector<int> res(N, 0);

    for (int i = 0; i < N; ++i)
    {
        auto q = x / xs[i];
        x -=  q*xs[i];

        res[i] = q;
    }

    return res;
}

```

Solução $O(N^3)$

```
22 int value(const vector<int>& M, int N, const vector<int>& xs)
23 {
24     int res = 0;
25
26     for (int i = 0; i < N; ++i)
27         res += M[i]*xs[i];
28
29     return res;
30 }
31
32 int solve(int N, const vector<int>& xs)
33 {
34     if (N <= 2)
35         return -1;
36
37     int ans = oo;
38
39     for (int i = N - 2; i >= 0; --i)
40     {
41         auto g = greedy(xs[i] - 1, N, xs);
42         vector<int> M(N, 0);
```

Solução $O(N^3)$

```
43
44          for (int j = 0; j < N; ++j)
45          {
46              M[j] = g[j] + 1;
47              auto w = value(M, N, xs);
48              auto G = greedy(w, N, xs);
49
50              auto x = accumulate(M.begin(), M.end(), 0);
51              auto y = accumulate(G.begin(), G.end(), 0);
52
53              if (x < y)
54                  ans = min(ans, w);
55
56              M[j]--;
57          }
58      }
59
60      return ans == oo ? -1 : ans;
61  }
62
```

Solução $O(N^3)$

```
63 int main()
64 {
65     ios::sync_with_stdio(false);
66
67     int N;
68     cin >> N;
69
70     vector<int> xs(N);
71
72     for (int i = 0; i < N; ++i)
73         cin >> xs[i];
74
75     auto ans = solve(N, xs);
76
77     cout << ans << '\n';
78
79     return 0;
80 }
```

Referências

1. OJ 166 - Making Change
2. OJ 147 – Dollars
3. Codeforces Beta Round #10 – Problem E: Greedy Change