Força Bruta, Programação Dinâmica e Algoritmos Gulosos
URI: Troco(2446)

Luiz Fellipe Machi Pereira(RA 103491)
Diogo Fernando de Melo Sales(RA 93814)

25 de junho de 2018

1 Descrição do Problema

Você está num supermercado e está na fila do caixa para comprar alguns produtos. Assim que você termina de passar as compras pelo caixa, se lembra que tem várias moedas em seu bolso, algumas repetidas, e fica pensando se com elas dá para pagar exatamente o valor das compras (para assim se livrar destas moedas e ficar com os bolsos mais leves). Você consegue pagar o valor exato da conta usando estas moedas?

1.1 Entrada

A primeira linha da entrada contém dois números inteiros **V** ($1 \leq V \leq 10^5$) e **M** ($1 \leq M \leq 10^3$), indicando, respectivamente, o valor final da compra e o número de moedas que você tem em seu bolso. A segunda linha contém M números inteiros que descrevem o valor **M**i ($1 \leq Mi \leq 10^5$) de cada moeda existente em seu bolso.

1.2 Saída

Seu programa deve imprimir apenas uma linha, contendo apenas um caractere: S caso seja possível pagar o valor exato da conta usando apenas suas moedas, ou N caso contrário.

2 Formulação recursiva

$$opt(m, V) = \begin{cases} True & \text{se } V = 0 \\ False & \text{se } m < 0 \text{ e V} > 0 \\ opt(m-1, V - M[m]) & \\ \vee\ opt(m-1, V) & \text{se } M \geq 0 \text{ e V} > 0 \end{cases}$$

3 Substrutura ótima

Para mostrar que há subestrutura ótima devemos analisar dois casos:
A) $m \in S$ (Considere agora que o valor $M[m]$ faça parte da solução ótima). Assumimos que $S^* = opt(m-1, V - M[m]) \cup M[M]$ onde m é quantidade de moedas que você tem no bolso, V é o valor final da compra e o vetor M representa o valor de cada moeda, é a solução ótima para o problema de tamanho m. Suponha que exista outra solução $A^* = opt'(m-1, V - M[m])$ que devolve um valor estritamente menor do que $S^* - M[m]$, ou seja, exista uma forma de obter o valor V utilizando menos moedas e que devolve $Verdadeiro$. Portanto, há como melhorar a solução ótima utilizando $A^* \cup M[m]$, o que é uma contradição.
B) $m \in S = S$ e $M[m]$ não faz parte da solução ótima então $S = [1...m-1]$ tem que ser a solução ótima para o subproblema. A prova é análoga ao caso anterior.

4 Sobreposição de problemas

Para demonstrar que o problema apresenta sobreposição de problemas foi usado o método de árvore de recursão Figura 1.

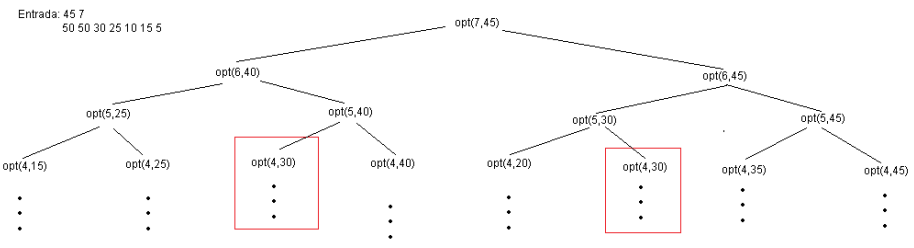

Figura 1: Árvore de Recursão

5 Análise de Complexidade

5.1 Força Bruta

A recorrência força bruta T(V) é dada pelo seguinte somatório, onde V é o valor do troco, m é quantidade de moedas disponíveis indo de $[M[1]...M[m]]$ e $M[i]$ é o valor de cada moeda. Para determinar sua complexidade mais precisamente é necessário encontrar uma fórmula fechada para o somatório das recorrências abaixo.

$$\sum_{i=0}^{m} T(V - M[i])$$

5.2 Programação dinâmica/memoizado

Para determinar qualquer complexidade de um algoritmo memoizado, é necessário fazer o cálculo do Número de subproblemas distintos vezes o Custo de cada subproblema, em nosso caso, a quantidade de subproblemas é $(m.V)$ (memo), onde m é a quantidade de moedas disponíveis e V o valor desejado, e o custo de cada linha é constante, visto que as chamadas recursivas têm custo constante e as comparações também. Portanto a complexidade do algoritmo é $O(m.V)$.

6 Escolha Gulosa

Nossa proposta de algoritmo guloso é ordenar o vetor de moedas, percorrer o vetor em busca de encontrar uma moeda de valor igual a V, caso não exista, o algoritmo pega as menores moedas e vai somando, se a soma for igual a V o algoritmo retorna $Verdadeiro$, se a soma for maior que V retorna-se $Falso$. O algoritmo falha quando a única solução é um valor grande de uma moeda somado com um valor pequeno de moeda.

```
1 def troco(coins,qtd,val):
2         soma = 0
3         for i in range(len(coins)):
4                 if soma == val:
5                         return True
6                 elif soma > val:
7                         return False
8                 else:
```

```
                        soma += coins[i]

val = [int(i) for i in input().split()]
v = val[0]
m = val[1]
coins = [int(i) for i in input().split()]
coins.sort()
for x in range(len(coins)):
        if coins[x] == v:
                flag = 1
                break
if flag == 1:
        print('S')
else:
        if troco(coins,m-1,v) == True:
                print('S')
        else:
                print('N')
```

7 Casos de teste

Entrada 1 :
```
16 4
25 10 5 1
```
Saída 1: S

Entrada 2 :
```
20 4
25 10 5 1
```
Saída 2: N

Entrada 3 :
```
35 10
33 33 32 31 30 25 20 14 7 1
```
Saída 3: S

Entrada 4 :
```
21 4
25 10 5 1
```
Saída 4: N

Entrada 5 :
```
9270 20
269 657 124 367 150 961 290 856 304 929 665 231 877 377 447 965 82 661 462 617
```
Saída 5: S

8 Comparação de algorítimos

Para efetivar a melhora trazida pela PD foram comparadas as versões em modo Força Bruta, Memoizado e Iterativo do problema Troco(2446). A tabela a seguir apresenta os resultados Tabela 1.

Tabela 1: Comparação dos Algoritmos no quesito tempo.

Tipo	Força Bruta	Memoizado	Iterativo
Real	0,039s	0,039s	0,005s
User	0,026s	0,032s	0,000s
Sys	0,013s	0,007s	0,005s

No caso para todos os algoritmos foi usada a Entrada 1 especificada na sessão 7.

Força Bruta

```
1  def troco(coins,qtd,val):
2          if val >= 0 and qtd >= 0:
3                  if val == 0:
4                          return True
5                  elif(qtd < 0 and val > 0):
6                          return False
7                  else:
8                          return troco(coins,qtd-1,val
                               - coins[qtd]) or troco(
                              coins,qtd-1,val)
9          else:
10                 return False
11
12 val = [int(i) for i in input().split()]
13 v = val[0]
14 m = val[1]
15 coins = [int(i) for i in input().split()]
16 if troco(coins,m-1,v) == True:
17         print('S')
18 else:
19         print('N')
```

Memoizado

```
def troco(coins,qtd,val,memo):
        if qtd < 0 or val <0:
                return False
        else:
                if memo[qtd][val] == -1:
                        if val == 0:
                                ans = True
                        elif(qtd < 0 and val > 0):
                                ans = False
                        else:
                                ans = troco(coins,
                                   qtd-1,val - coins
                                   [qtd],memo) or
                        troco(coins,qtd-1,val,memo)
                                memo[qtd][val] = ans
        return memo[qtd][val]

val0 = [int(i) for i in input().split()]
v = val0[0]
m = val0[1]
coins = [int(i) for i in input().split()]
memo = []
for x in range(m+1):
        lines = []
        for x in range(v+1):
                lines.append(-1)
        memo.append(lines)

if troco(coins,m-1,v,memo) == True:
        print('S')
else:
        print('N')
```

Iterativo

```
#include <iostream>

using namespace std;

int troco(int coins[], int m, int v){
        int ans,memo[m+1][v+1];
```

```
7
8           for (int i = 0; i <= m; ++i){
9                   memo[i][0] = 1;
10          }
11          for (int i = 1; i <= v; ++i){
12                  memo[0][i] = 0;
13          }
14          for (int i = 1; i <= m; i++){
15                  for (int j = 1; j <= v; j++){
16                          if (j < coins[i-1]){
17                                  memo[i][j] = memo[i
                                       -1][j];
18                          }
19                          else{
20                                  memo[i][j] = memo[i
                                       -1][j] || memo[i
                                       -1][j-coins[i
                                       -1]];
21                          }
22                  }
23          }
24          return memo[m][v];
25  }
26
27  int main(){
28          int v = 0,m = 0,s=0;
29          cin >> v >> m;
30          int coins[m];
31
32          for (s = 0; s < m; s++){
33                  cin >>coins[s];
34          }
35
36          if (troco(coins,m,v) == 1){
37                  printf("S\n");
38          }else{
39                  printf("N\n");
40          }
41  }
```

9 Referências

O PROBLEMA subset-sum. Disponível em: https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/mochila-subsetsum.html. Acesso em: 23 jun. 2018.
DYNAMIC Programming — Set 25 (Subset Sum Problem). Disponível em: https://www.geeksforgeeks.org/dynamic-programming-subset-sum-problem/. Acesso em: 23 jun. 2018.
URI Online Judge — 2446. Disponível em: https://www.urionlinejudge.com.br/judge/pt/problems/view/2446. Acesso em: 23 jun. 2018.