Menor quantidade de moedas para um valor

Explicação do problema – Menor quantidade de moedas para um valor

O problema do “troco”, como ficou mundialmente conhecido consiste em:

Dado um valor X de moedas, queremos dar um troco no valor de X reais. Suponhamos que temos quantidades infinitas de moedas de Troco = (T1, T2, T3 ..., Tn), de valores distintos. A pergunta chave desse problema é, qual seria a menor quantidade de moedas para o valor X.

Baseado nesse problema, temos alguns algoritmos de diferentes tipos para obter essa solução.

Nada melhor do que explicar esse problema com os exemplos:

Exemplos

Algoritmo Guloso

Para X moedas = {25, 4, 3, 1} e T = 31, sem moedas de 5, o algoritmo Guloso daria (25+4+1+1).

Porém podemos fazer melhor, utilizando programação dinâmica:

Usando o mesmo exemplo de cima, o algoritmo guloso usaria 4 moedas para dar de troco de 31 centavos, usando a programação dinâmica poderíamos resolver esse problema utilizando 3 moedas de troco.

Para X moedas = {25, 4, 3, 1} e T = 31, sem moedas de 5, **o algoritmo dinâmico daria (25+3+3)**, diferente do guloso que daria (25+4+1+1).

A diferença entre ambos, é que o algoritmo guloso resolveria da seguinte maneira:

- A estratégia será dividir o T = 31 sempre pelo maior valor do vetor de moedas, e o restante que sobrar da divisão pelo segundo maior valor do vetor de moedas, e assim consecutivamente.

  Exemplo

  31 / 25 = 1 → Faremos a divisão e ignoramos o resto, assumindo assim que 1 moeda de troco de 25 cabe dentro de 31.
  Sobra 6
  6 / 4 = 1 → Assumimos aqui que uma 1 moeda de troco de 4c dentro de 6.

Sobra 2
2 / 3 = 0 → Nessa conta podemos ver a diferença do Guloso para o Dinâmico, aonde no guloso nenhuma moeda de 3c é selecionada para o troco, pois, o algoritmo só assume que a moeda de troco é válida quando a divisão é => a 1.
Sobra 2
2 / 1 = 2 → Será usada duas moedas de troco de 1c.

Curiosidade : Esse problema é uma das variações discutidas no Problema de mudança de moeda. Aqui, em vez de acharmos o número total de soluções possiveis, iremos encontrar a solução com o número mínimo de moedas.

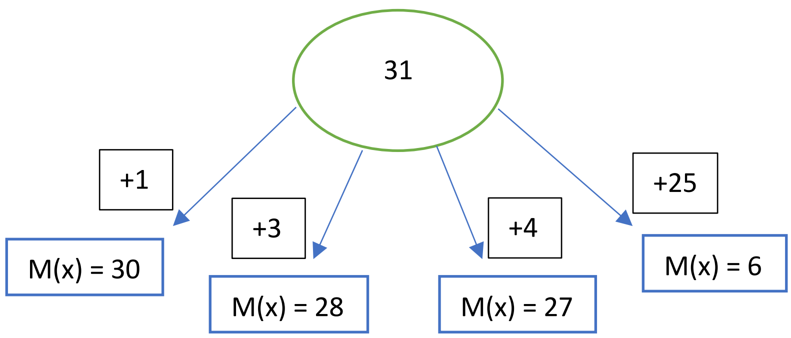
Explicação da solução dinâmica - Passo a Passo

Para entendermos a solução desse problema do jeito dinâmico, precisamos primeiro entender que para que a programação dinâmica seja aplicável em um problema, precisa existes duas principais características:

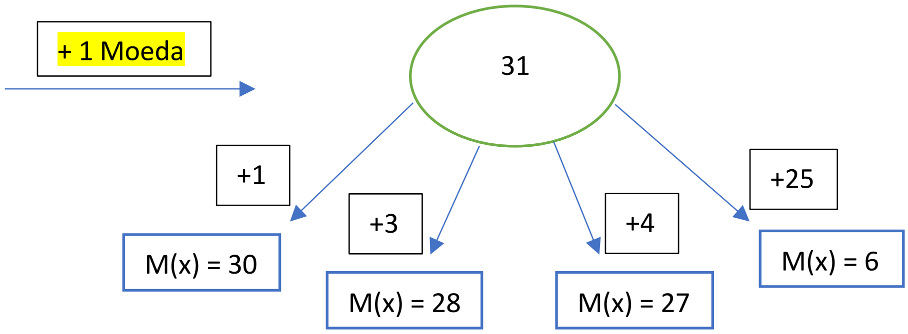
- **Subestrutura ótima**
  - Um problema apresenta uma subestrutura ótima quando uma solução ótima para o problema contém em seu interior, soluções ótimas para subproblemas.
- **Superposição de subproblemas**
  - A superposição de subproblemas acontece quando um algoritmo recursivo reexamina o mesmo problema muitas vezes.

No problema do troco, utilizando programação dinâmica, vamos cair na **Superposição de subproblemas,** pois:

- Primeiramente assumiremos um M(x) = Quantidade de Moedas.
- Depois, ao invés de eu dar o troco cheio de 31 eu poderia assumir que a moeda de troco de 25c faz parte da solução, e assim por diante, com todas as moedas de troco que eu tiver.



- Conseguimos representar um problema maior através do ganho de um problema menor, utilizando 1 moeda de troco.
- Estamos utilizando soluções menores de 1 moeda para tentar ganhar o problema principal, por isso, o que importa é esse valor que nós descontamos do valor total para assim propagar a árvore.



M(x) = Quantidade de Moedas

T = [25,4,3,1]

T = troco

X = valor

$$M(x) \begin{cases} 1 + M(x - 1) \\ 1 + M(x - 3) \\ 1 + M(x - 4) \\ 1 + M(x - 25) \end{cases}$$

- Com isso montaremos uma **equação de otimização**

$$M(x) = \underset{0 <= I <= 4}{\mathrm{ArgMin}} \ (1 + M(x - T)$$

Explicando a equação

- **ArgMin** = argumento mínimo. O argumento mínimo é o valores do vetor de troco T = [25,4,3,1].
- O numero 1 (**1** + M(x-t)) representa que eu devo utilizar pelo menos **uma** das moedas do valor do vetor de troco.
- (1 + **M (x – T)**) = essa é a nossa solução, que descobrimos ali em cima.
- (1 + M (x – **T**)) = o **T** é a moeda que nós tiramos (25,4,3,1).

No **exemplo de funcionamento**, colocaremos essa formula em prática.

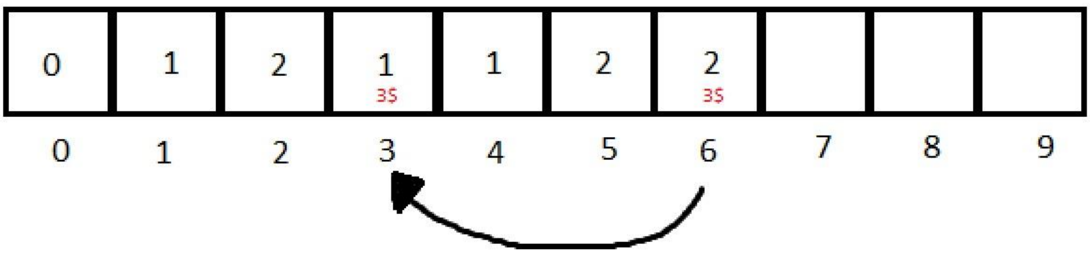
Exemplo de Funcionamento

M(x) = ArgMin (1 + M(x – T)

0 <= I <= 4

Usaremos essa formula para calcular a menor quantidade, como vimos ali em cima:

No Q(3), aplicaremos na formula as possibilidade de moeda de troco possíveis, que no caso são: 1 centavo e 3 centavos.



$$Q(3) = \begin{cases} \overset{1\$}{1} + Q(3-1) \Rightarrow 3 \\ \overset{3\$}{1} + Q(3-3) \Rightarrow 1 \end{cases} \qquad Q(6) = \begin{cases} \overset{1\$}{1} + Q(5) \Rightarrow 3 \\ \overset{3\$}{1} + Q(3) \Rightarrow 2 \\ \overset{4\$}{1} + Q(2) \Rightarrow 3 \end{cases}$$

Resolvendo a formula podemos ver que surgiu a possibilidade de utilizar 3 moedas de 1c ou utilizar 1 moeda de 3c.

E sempre pegaremos como resposta o **menor valor** encontrado**.**

No Q(6), aplicaremos na formula as possibilidades de moeda de troco possíveis, que no caso são: 1 centavos, 3 centavos, 4 centavos

Como o Q(6) será o último que analisaremos, para obtemos os resultados usaremos as respostas já encontradas nas analises anteriores, por exemplo:

1 + **Q(5),** o valor do Q(5) já encontramos e sabemos que é 2. Assim, substituímos o Q(5) por 2... 1 + 2 = 3

Isso se aplica para as outras equações do Q(6).

E sempre pegaremos como resposta o **menor valor** encontrado.

Implementação

Implementamos o algoritmo dinâmico em c++, utilizando o mesmo exemplo acima.

```cpp
#include <iostream>
using namespace std;

int minimoMoedas(int moedas[], int m, int V)

{
    //caso base
    if (V == 0) return 0;

    //inicializando o resultado
    int resultado = INT_MAX;

    // tente todas as moedas com o valor menor que V
    for (int i = 0; i < m; i++)
    {
      if (moedas[i] <= V)
      {
          int subR = minimoMoedas(moedas, m, V-moedas[i]);

          if (subR != INT_MAX && subR + 1 < resultado)
             resultado = subR + 1;
      }
    }
    return resultado;

}
```

A nossa **equação de otimização** é aplicada na **segunda condição** dentro do For...

```cpp
int main()
    {
        int moedas[] =  {25,4,3,1};

        int m = sizeof(moedas)/sizeof(moedas[0]);

        int V = 31;

        cout << "O minimo de moedass eh: " << minimoMoedas(moedas, m, V);

        return 0;
    }
```

```
C:\Users\Victor\Documents\Find minium.exe
O minimo de moedass eh: 3
--------------------------------
Process exited after 0.3315 seconds with return value 0
Pressione qualquer tecla para continuar. . .
```

Analise assintótica

```
Para i = 0 até i < m -----------------------→ n - 1
{
        Se moedas <= V ---------------→ C
        {
                subR = minimoMoedas(moedas,m,V-moedas[i]) ---------→ 1

                Se (subR != INT_MAX && subR + 1 < resultado) -----→ C
        }
}
Retorne resultado --------------→ 1
}
```