Mochila Binária - solução ótima

Técnicas de Programação - 2025/02

Última atualização: 08/10/2025 11:19

Neste roteiro iremos responder à pergunta:

> Dada uma configuração da mochila com as variáveis abaixo, conseguimos uma solução com valor maior que X?

Temos as seguintes informações:

1. `N` - número de objetos
2. `C` - capacidade máxima da nossa mochila
3. `V` - array com o valor de cada objeto
4. `P` - array com o peso de cada objeto

e a saída do nosso algoritmo é

1. `O` - array de `VERDADEIRO/FALSO` indicando se o objeto `I` foi selecionado
2. `VALOR_TOTAL`
3. `PESO_TOTAL`, que deverá ser menor que `C`

Revisão da parte expositiva

Vamos rever algumas características do nosso problema.

Exercício: Em uma mochila com 13 objetos, quantas são soluções possíveis?

Exercício: Quais são os valores de `N` que conseguimos resolver sem envolver recursão?

A cada passo do algoritmo decidimos se selecionamos ou não o objeto atual. Se selecionamos, resolvemos um novo problema da mochila com capacidade menor (em função do objeto que acabamos de selecionar) e com somente os objetos restantes. Se não mantemos a capacidade e eliminamos o objeto atual da lista de objetos. Suponha, para os próximos exercícios, que já escolhemos $i - 1$ objetos e estamos decidindo se selecionamos o objeto i, de peso 10 e valor 5. Nossa mochila ainda tem 50 de capacidade e o valor atual é 43. Os objetos selecionados estão colocados em um array `O`.

Exercício: Se desejarmos **selecionar** o objeto i, como nossa solução parcial será modificada? Qual será a capacidade restante e o novo valor da mochila?

Exercício: Se desejarmos **não selecionar** o objeto i, como nossa solução parcial será modificada? Qual será a capacidade restante e o novo valor da mochila?

Montamos juntos, em aulas anteriores, o algoritmo para criar sequências de DNA de tamanho `N`. Reveja ele abaixo.

```
DNA(I, atual, N)

if I = N then
  print(atual)
end

DNA(I+1, atual + "A", N)
DNA(I+1, atual + "T", N)
DNA(I+1, atual + "C", N)
DNA(I+1, atual + "G", N)
```

Exercício: Como você poderia adaptá-lo para este novo problema? Considere que receberá os seguintes parâmetros:

1. `N` - número de objetos
2. `C` - capacidade máxima da nossa mochila
3. `V` - array com o valor de cada objeto
4. `P` - array com o peso de cada objeto
5. `O` - array de `VERDADEIRO/FALSO` indicando se o objeto `I` foi selecionado. Representa a solução atualmente encontrada
6. `VO` - valor atual dos objetos selecionados em `O`
7. `I` - objeto atual

Considere que a primeira chamada é feita com `O = [false, false, ....], VO=0, I=0`.

Por enquanto, seu programa deve dar `print` de todas as mochilas possíveis, junto com seu valor e peso. O algoritmo não devolve valor nenhum.

```
MOCHILA(N, V, P, O, C, VO, I)

# Caso em que não precisa fazer chamada recursiva vai aqui




# Caso em que adicionamos o elemento I




# Caso em que não adicionamos o elemento I
```

E para guardar a solução?

Guardar a solução ótima é "fácil". Vamos começar com uma ideia simples: adicione um novo parâmetro `MelhorO` ao seu algoritmo e, toda vez que completar uma solução, compare-a com `MelhorO`. Se a solução atual tiver valor maior, copie-a em `MelhorO`.

Exercício: com base na descrição acima, qual o tipo da variável `MelhorO`?

Exercício: qual seria o valor inicial da variável `MelhorO`?

Exercício: agora reescreva o caso em que não precisa fazer chamada recursiva para atualizar `MelhorO`.

```
MOCHILA(N, V, P, MelhorO, O, C, VO, I)

# Caso em que não precisa fazer chamada recursiva vai aqui
```

```
# O resto continua igual, mas passando o parâmetro MelhorO
```

Dá pra fazer melhor?

É possível melhorar um pouco esse programa, mas infelizmente não conseguimos diminuir sua complexidade computacional para menos que $\mathcal{O}(2^N)$.