# Knapsack

Thiago Rafael Becker

Novembro de 2011

## Introdução

*Definição do problema.*O problema, em sua forma original, é definido como: *Dado um conjunto de itens, cada um com um peso e um valor, como encher uma mochila como tal que o peso total não ultrapasse o limite de peso da mochila, e o valor total seja o maior possível.*

*Importância do problema.* Problemas similares aparecem em economia (alocação de recursos com limitação financeira, busca de estratégias com limitação de risco), combinatória, matemática aplicada, logística (*bin packing*), criptografia (*subset sum problem*), corte de materiais com despedício mínimo, .

*Problema de decisão.* O problema, como proposto, não serve para a análise de sua complexidade, pois não é um problema de decisão. Para torná-lo um problema de decisão, estabelecemos um limite mínimo $V$ ao valor máximo dos itens na mochila, levando ao seguinte problema de decisão: *Dado um conjunto de $n$ itens $I$, cada um com um peso $p_i$ e um valor $v_i$ ($i \in I$), é possível encher uma mochila de forma que o peso total do itens não ultrapasse o limite de peso $P$ e o valor total seja pelo menos $V$.*

Existem diversas variações do problema, mas vamos nos focar nas duas abaixo.

*Knapsack-0,1.* Este problema surge quando os itens são indivisíveis. Este problema é NP-Completo fraco.

*Knapsack fracionário.* Este problema surge quando os itens podem ser divididos em parte fracionárias e pode ser resolvido eficientemente por uma estratégia gulosa.

## Knapsack-0,1

Com o problema de decisão definido acima, podemos definir o problema com as seguintes restrições

$$\left(\sum_{i \in I} v_i\right) \geq V \tag{1}$$

$$\left(\sum_{i \in I} p_i\right) \leq P \tag{2}$$

### Solução por força bruta

Como existem $n$ itens, existem $2^n$ subconjuntos de $I$. O algoritmo iria procurar por cada um dos conjuntos de itens pela solução ótima, o que levaria a um problema de complexidade $O(2^n)$.

[[Algoritmo aqui]]

### Solução em tempo pseudo-polinomial (programação dinâmica)

Por programação dinâmica, é possível obter uma solução de complesidade $O(nP)$.

*Subestrutura ótima.*

$$B[i,w] = \begin{cases} B[i-1,w] & \text{se } p_i > w \\ max(B[i-1,w], B[i-1,w-p_i]+v_i) & \end{cases}$$

A resposta para o problema de decisão se encontrará na célula $B[n,P]$ da matriz resultante. Se $B[n,P] \geq V$, o problema está satisfeito.

*Algoritmo.* O pseudo-código da solução encontra-se abaixo.

**for** $i = 1 \to n$ **do**
  **for** $w = 0 \to P$ **do**
    **if** $p_i \leq w$ **then**
      **if** $v_i + B[i-1, w-p_i] > B[i-1,w]$ **then**
        $B[i,w] = v_i + B[i-1, w-p_i]$
      **else**
        $B[i,w] = B[i-1,w]$
      **end if**
    **else**
      $B[i,w] = B[i-1,w]$
    **end if**
  **end for**
**end for**
**return** $B[n,P]$

## Complexidade do problema

A complexidade do algoritmo acima é $O(nP)$, na qual $n$ é a quantidade de elementos do conjunto de entrada, e $P$ é o peso máximo suportado pela mochila. É, portanto, pseudo-polinomial, pois não depende exclusivamente do tamanho do conjunto que compõe o preoblema.

## Prova da plenitude-NP

*Subset sum.* O problema *subset sum* será usado para provar que problema da mochilaé NP-Completo. O problema é proposto da seguinte maneira: dado um conjunto de inteiros $X$, existe um subconjunto $S$ de $X$ ($X$ incluso) tal que $(\sum S_i) = c$?

*Caso especial de problema da mochila.* O caso especial do problema da mochilausado para a prova será tal que $P = V$ e $(\forall i \in I)(v_i = p_i)$. Com essa equivalencia, é possível transformar a restrição 2 na seguinte restrição:

$$\left(\sum_{i \in I} v_i\right) \leq V \tag{3}$$

e igualando as restrições 2 e 3 pelo somatório comum, chegamos a conclusão de que a restrição do caso especial é

$$\left(\sum_{i \in I} v_i\right) = V \tag{4}$$

Analogamente,

$$\left(\sum_{i \in I} p_i\right) = P \tag{5}$$

Este problema é equivalente ao problema de *subset sum*.

## Knapsack fracionário.

Este problema pode ser resolvido dividindo-se o valor total de cada item pelo seu peso, obtendo-se o valor por unidade de peso do item $\rho_i$. Após isso, a prenchimento da mochila ocorre usando-se o itens em ordem decrescente de valor $\rho_i$.