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 como

$$maximizar \sum_{i \in I} v_i$$

restrito à

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

tal que $\sum_{i \in I} v_i \geq V$.

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)

Uma solução em tempo pseudo-polinomial $O(nP)$ pode ser obtida por programação dinâmica.

Subproblema ótimo.

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

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 $i = 1 \rightarrow n$ **do**
  for $w = 0 \rightarrow P$ **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 $B[n, P]$

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 ρ_i. Após isso, a prenchimento da mochila ocorre usando-se o itens em ordem decrescente de valor ρ_i.