Análise e Projeto de Algoritmos

Humberto Longo

Instituto de Informática
Universidade Federal de Goiás

Bacharelado em Ciências da Computação, 2013

Problema da Mochila – *Knapsack Problem*

Exemplo 4.8

- Como armazenar. em um compartimento qualquer, muitos itens com valores agregados, de forma que o somatório dos valores de tais itens seja o máximo possível dentre as possibilidades de combinação de itens?

- Matriz bidimensional (M), de ordem $(N + 1) \times (C + 1)$, onde:

  N: Número de itens;

  C: Capacidade da mochila.

Problema da Mochila – *Knapsack Problem*

Exemplo 4.8

- Idéia do algoritmo:
  1. Começar com uma mochila de capacidade 1 e descobrir o valor máximo possível para esta mochila.
  2. Passar para uma mochila de capacidade 2 e aproveitar as informações da mochila de capacidade atual - 1 (nesse caso, 1) para descobrir o valor máximo para uma mochila de capacidade 2.
  3. Repetir esse procedimento até capacidade atual ser igual a C. Além da capacidade ir crescendo, também vai se adicionando novos itens no processo.

Problema da Mochila – *Knapsack Problem*

Exemplo 4.8

- Idéia do algoritmo:
  1. Percorrendo a tabela por linha (por item), e para a coluna j atual (capacidade atual), analisar se o item atual i (linha atual) cabe na mochila de capacidade j.
  2. Se couber é preciso escolher o maior valor entre:
     - 2.1 Valor na mochila de mesma capacidade j que não tinha esse item (essa informação estará em $M[i-1][j]$).
     - 2.2 Soma do valor do item com o valor na mochila de capacidade (j – peso do item i) e que não tinha esse item ($M[i-1][j$ – peso do item $i]$).
  3. Se não couber, o maior valor para essa mochila será o valor da mochila de mesma capacidade, mas que não tem esse item ($M[i][j] = M[i-1][j]$).

Problema da Mochila – *Knapsack Problem*

Exemplo 4.8

Item	Peso	Valor
1	4	2
2	2	1
3	1	3
4	2	4
5	2	1

Programação Dinâmica

Algoritmo

Knapsack(N, C)

Entrada: Número de produtos (N) e Capacidade da mochila (C).
Saída: Valor máximo para a capacidade C.

1 **para** $i \leftarrow 0$ **até** N **faça** $M[i][0] \leftarrow 0$;

2 **para** $j \leftarrow 1$ **até** C **faça** $M[0][j] \leftarrow 0$;

3 **para** $i \leftarrow 1$ **até** N **faça**
4 **para** $j \leftarrow 1$ **até** C **faça**
5 **se** $itens[i].peso \leq j$ **então**
6 $M[i][j] \leftarrow \max \left(\begin{array}{l} M[i-1][j], \\ M[i-1][j - itens[i].peso] + itens[i].valor \end{array} \right)$;
7 **senão**
8 $M[i][j] \leftarrow M[i-1][j]$;

9 **retorna** $M[N][C]$;

Problema da Mochila – *Knapsack Problem*

- **Mochila 0-1 sem pesos**:
  - $S = \{s_1, s_2, \ldots, s_n\}$.
  - $s_i \in \mathbb{Z}^+,\ i = 1, \ldots, n$.
  - $K \in \mathbb{Z}^+$.
  - $\exists\, S' \subset S$ tal que $\sum_{s_i \in S'} s_i = K$?
- $M(n, K) \rightarrow$ problema da mochila com n itens e mochila de tamanho K.
- $M(i, k),\ i \leq n \rightarrow$ problema da mochila com os i primeiros itens de S e mochila de tamanho k.

Problema da Mochila – *Knapsack Problem*

- Solução: hipótese de indução simples (errado!)
  - **Suponha que sabe-se resolver** $M(n-1, K)$.
  - Base: há uma solução se e somente se $n = 1$ e $s_1 = K$.
  - $M(n-1, K)$ tem solução $\rightarrow$ problema está terminado.
    - item s_n não é utilizado.
  - $M(n-1, K)$ não tem solução.
    - Solução de $M(n, K)$ deve incluir o item s_n.
    - $n-1$ primeiros itens em uma mochila de tamanho $K - s_n$.
  - $M(n, K)$ reduzido para $M(n-1, K)$ e $M(n-1, K-s_n)$.
  - $M(n-1, K-s_n) \Rightarrow$ problema não coberto pela hipótese de indução!

Problema da Mochila – *Knapsack Problem*

- Solução: hipótese de indução forte.
  - **Suponha que sabe-se resolver** $M(n-1,k), \forall\ 0 \leq k \leq K$.
  - Base:
    - Se $k = 0$ existe uma solução.
    - Se $k = 1$ há uma solução se e somente se $n = 1$ e $s_1 = K$.

Problema da Mochila – *Knapsack Problem*

- Passo da indução:
  - $M(n-1, K)$ tem solução $\rightarrow$ problema está terminado.
    - item s_n não é utilizado.
  - $M(n-1, K)$ não tem solução.
    - solução de $M(n, K)$ deve incluir o item s_n.
    - $n-1$ primeiros itens em uma mochila de tamanho $K - s_n$.
  - $M(n, K)$ reduzido para $M(n-1, K)$ e $M(n-1, K-s_n)$.
  - Se $K - s_n < 0$, $M(n-1, K-s_n)$ é ignorado.

Programação Dinâmica

Problema da Mochila – *Knapsack Problem*

Mochila-Decisão$(n, K, S = (s_1, \ldots, s_n))$

```
M[0,0] ← verdadeiro;
para i ← 1 até K faça M[0,i] ←sem_solução;
para i ← 1 até n faça
    para k ← 0 até K faça
        M[i,k] ← sem_solução;
        se (tem_solução(M[i − 1,k])) então
            M[i,k] ← tem_solução_não_pertence;
        senão
            se (k − S[i] ≥ 0) então
                se (tem_solução(M[i − 1,k − S[i]])) então
                    M[i,k] ← tem_solução_pertence;
retorna M[n,K];
```

Programação Dinâmica

Problema da Mochila – *Knapsack Problem*

- Algoritmo recursivo $\Rightarrow$ complexidade exponencial em m.
- Máximo de $n.K$ problemas distintos.
  - $M(i,k)$ para todos os valores possíveis de i e k.
- Programação Dinâmica:
  - Tabela $n \times K$.
  - Elemento $[i,k] \rightarrow$ problema $M(i,k)$.
  - Para $i = 1,\ldots,n$ e $k = 0\ldots,K \rightarrow$ nenhum problema é computado duas vezes.
  - Solução na posição $[n,K]$.
  - Complexidade $O(n.K)$.

Exemplo 4.9

- $S = \{s_1, s_2, s_3, s_4\} = \{2, 3, 5, 6\}$ e $K = 6$:

Tam. Mochila $\longmapsto$	0	1	2	3	4	5	6
$S_0 = \emptyset$	$\notin$	–	–	–	–	–	–
$s_1 = 2, \quad S_1 = \{s_1\} \cup S_0$	$\notin$	–	$\in$	–	–	–	–
$s_2 = 3, \quad S_2 = \{s_2\} \cup S_1$	$\notin$	–	$\notin$	$\in$	–	$\in$	–
$s_3 = 5, \quad S_3 = \{s_3\} \cup S_2$	$\notin$	–	$\notin$	$\notin$	–	$\notin$	–
$s_4 = 6, \quad S_4 = \{s_4\} \cup S_3$	$\notin$	–	$\notin$	$\notin$	–	$\notin$	$\in$

Problema da Mochila – *Knapsack Problem*

- Este algoritmo **não é polinomial**:
  - Entrada do problema: $(K, n, s_1, s_2, \ldots, s_n)$.
  - Se K fosse fixo, o algoritmo seria polinomial (linear).
  - Caso geral — instância pode ser dividida em duas partes:
    - uma parte de tamanho $O(n)$.
    - uma parte de tamanho $O(\log K)$.
  - Se $K >> n$, $O(n.K)$ é exponencial em relação a $O(\log K)$.

Livros Texto


T. H. Cormen, C. E. Leiserson e R. L. Rivest.
Introduction to Algorithms.
McGraw-Hill, New York, 1990.


C. H. Papadimitriou, U. V. Vazirani e S. Dasgupta.
Algoritmos.
Mcgraw-Hill Brasil, 2009.


U. Manber.
Algorithms: A Creative Approach.
Addison-Wesley, 1989.


D. E. Knuth.
The Art of Computer Programming. Volume 1 – Fundamental Algorithms.
Addison Wesley, 1998.


D. E. Knuth.
The Art of Computer Programming. Volume 2 – Sorting and Searching.
Addison Wesley, 1998.


N. Ziviani.
Projeto de Algoritmos com Implementações em Pascal e C.
Editora Thomson. 2a Edição. 2004.