Projeto e Análise de Algoritmos

Aula 05 – Técnicas de Projeto de Algoritmos (Programação Dinâmica)

Edirlei Soares de Lima

<edirlei@iprj.uerj.br>

Estratégias de Projeto de Algoritmos

- **Força Bruta (Brute Force)**
- **Dividir e Conquistar (Divide and Conquer)**
- Diminuir e Conquistar (Decrease and Conquer)
- Transformar e Conquistar (Transform and Conquer)
- Compromisso Tempo–Espaço (Space and Time Tradeoffs)
- **Estratégia Gulosa (Greedy)**
- **Programação Dinâmica (Dynamic Programming)**
- Voltando Atrás (Backtracking)
- Ramificar e Limitar (Branch and Bound)
- Algoritmos Aproximados

Programação Dinâmica

- A programação dinâmica tem como objetivo reduzir o tempo de execução de um programa utilizando soluções ótimas a partir de subproblemas previamente calculados.
  - Resolvem-se os problemas de pequena dimensão e guardam-se as soluções;
  - A solução de um problema é obtida combinando as de problemas de menor dimensão.

Programação Dinâmica

- Passos gerais de um algoritmo baseado em Programação Dinâmica:
  - **Dividir o problema** em sub problemas;
  - **Computar** os valores de uma solução de forma bottom-up e **armazená-los** (memorização);
  - **Construir a solução** ótima para cada subproblema utilizado os valores computados.

- **Quando usar Programação Dinâmica?**
  - Quando a solução ótima de um subproblema pode ser composta pelas soluções ótimas dos subproblemas;
  - Quando o calculo da solução ótima implica muitas vezes no calculo do mesmo subproblema.

Exemplo 1: Série de Fibonacci

- Série de Fibonacci:

$$fib(n) = \begin{cases} 0 & se\ n = 0 \\ 1 & se\ n = 1 \\ fib(n-1) + fib(n-2) & se\ n > 1 \end{cases}$$

- Algoritmo Simples:

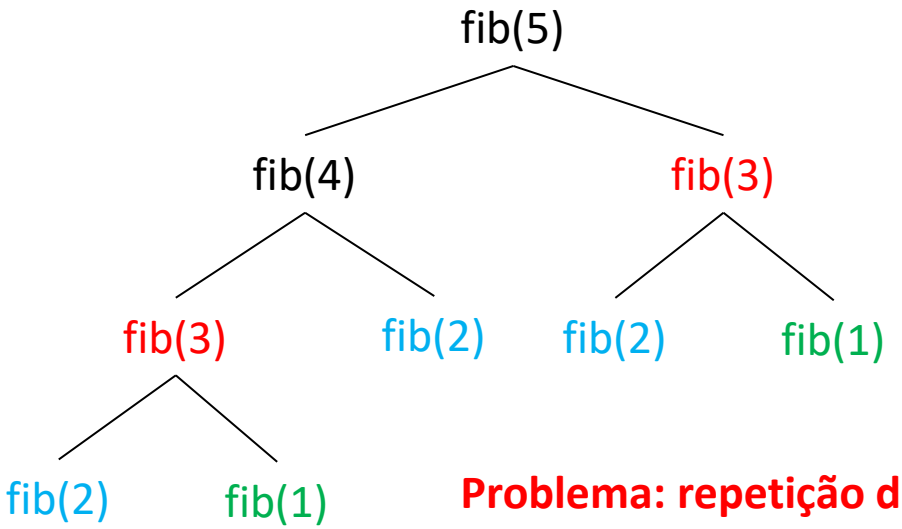
```
1. Fibonacci(n)
2.   se n < 2 então
3.     retorne n
4.   senão
5.     retorne fibonacci(n - 1) + fibonacci(n - 2)
```

O(2^n)

Exemplo 1: Série de Fibonacci

- Algoritmo Simples:

```
1. fib(n)
2.   se n < 2 então
3.     retorne n
4.   senão
5.     retorne fib(n - 1) + fib(n - 2)
```



Problema: repetição de chamadas!

Exemplo 1: Série de Fibonacci

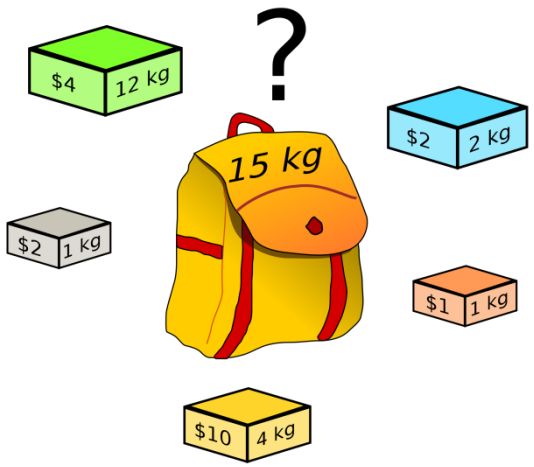
- Algoritmo Baseado em Programação Dinâmica:

```
1. fib(n)
2.   se n < 2 então
3.     retorne n
4.   senão
5.     V[0] ← 0
6.     V[1] ← 1
7.     para i ← 2 até n faça
8.       V[i] ← V[i-1] + V[i-2]
9.     retorne V[i]
```

O(n)

Exemplo 2: Problema da Mochila

- **Dados n itens:**
  - Pesos: w_1, w_2, ..., w_n
  - Valores: v_1, v_2, ..., v_n
  - Uma mochila de capacidade W



- **Problema:** encontrar o subconjunto mais valioso de itens que caibam dentro da mochila (Knapsack Problem).

Exemplo 2: Problema da Mochila

- **Problema:** encontrar o subconjunto mais valioso de itens que caibam dentro da mochila (Knapsack Problem).

- Solução Baseada em Programação Dinâmica:

$$c[i,w] = \begin{cases} 0 & se\ i = 0\ ou\ w = 0 \\ c[i-1,w] & se\ w_i > w \\ \max\{v_i + c[i-1, w-w_i], c[i-1,w]\} & se\ i > 0\ e\ w_i \leq w \end{cases}$$

Exemplo 2: Problema da Mochila

$$c[i,w] = \begin{cases} 0 & se\ i = 0\ ou\ w = 0 \\ c[i-1,w] & se\ w_i > w \\ \max\{v_i + c[i-1, w-w_i], c[i-1,w]\} & se\ i > 0\ e\ w_i \leq w \end{cases}$$

Item 1	Item 2	Item 3	Item 4
$V_1 = 5$	$V_2 = 4$	$V_2 = 7$	$V_2 = 7$
$W_1 = 5$	$W_2 = 6$	$W_2 = 8$	$W_2 = 4$

MAX = 13

w

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0														
1														
2														
3														
4														

Exemplo 2: Problema da Mochila

$$c[i,w] = \begin{cases} 0 & se\ i = 0\ ou\ w = 0 \\ c[i-1,w] & se\ w_i > w \\ \max\{v_i + c[i-1, w-w_i], c[i-1,w]\} & se\ i > 0\ e\ w_i \leq w \end{cases}$$

Item 1	Item 2	Item 3	Item 4
$V_1 = 5$	$V_2 = 4$	$V_2 = 7$	$V_2 = 7$
$W_1 = 5$	$W_2 = 6$	$W_2 = 8$	$W_2 = 4$

MAX = 13

i \ *w*	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	5	5	5	5	5	5	5	5	5
2	0	0	0	0	0	5	5	5	5	5	5	9	9	9
3	0	0	0	0	0	5	5	5	7	7	7	9	9	12
4	0	0	0	0	7	7	7	7	7	12	12	12	14	14

Exemplo 2: Problema da Mochila

```
1. Mochila_DP(w_i, v_i, n, W)
2.   para w ← 0 até W faça
3.     V[0,w] ← 0
4.   para i ← 1 até n faça
5.     V[i,0] ← 0
6.   para i ← 1 até n faça
7.     para w ← 0 até W faça
8.       se w_i ≤ w então
9.         se v_i + V[i-1, w-w_i] > V[i-1, w] então
10.          V[i,w] ← v_i + V[i-1, w-w_i]
11.        senão
12.          V[i,w] ← V[i-1,w]
13.      senão
14.        V[i,w] ← V[i-1,w]
15.  retorna V[n, W]
```

O(nW)

Exemplo 2: Problema da Mochila

- O algoritmo encontra o máximo valor possível que pode ser carregado na mochila. Como descobrir quais itens devem ser carregados?
  - Toda a informação necessária para descobrir quais são os itens está na tabela.

```
1. i ← n
2. k ← W
3. enquanto i > 0 e k > 0 faça
4.   se V[i, k] ≠ V[i-1, k] então
5.     coloque o item i na mochila
6.     i ← i - 1
7.     k ← k - w_i
8.   senão
9.     i ← i-1
```

Exemplo 2: Problema da Mochila

```
1. i ← n
2. k ← W
3. enquanto i > 0 e k > 0 faça
4.    se V[i, k] ≠ V[i-1, k] então
5.       coloque o item i na mochila
6.       i ← i - 1
7.       k ← k - w_i
8.    senão
9.       i ← i-1
```

Item 1	Item 2	Item 3	Item 4
$V_1 = 5$	$V_2 = 4$	$V_2 = 7$	$V_2 = 7$
$W_1 = 5$	$W_2 = 6$	$W_2 = 8$	$W_2 = 4$

MAX = 13

i \ *w*	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	5	5	5	5	5	5	5	5	5
2	0	0	0	0	0	5	5	5	5	5	5	9	9	9
3	0	0	0	0	0	5	5	5	7	7	7	9	9	12
4	0	0	0	0	7	7	7	7	7	12	12	12	14	14

Exemplo 2: Problema da Mochila

```
1. i ← n
2. k ← W
3. enquanto i > 0 e k > 0 faça
4.    se V[i, k] ≠ V[i-1, k] então
5.       coloque o item i na mochila
6.       i ← i - 1
7.       k ← k - w_i
8.    senão
9.       i ← i-1
```

Item 1	Item 2	**Item 3**	**Item 4**
$V_1 = 5$	$V_2 = 4$	**$V_2 = 7$**	**$V_2 = 7$**
$W_1 = 5$	$W_2 = 6$	**$W_2 = 8$**	**$W_2 = 4$**

MAX = 13

Solução: {Item 3, Item 4}

w

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	5	5	5	5	5	5	5	5	5
2	0	0	0	0	0	5	5	5	5	5	5	9	9	9
3	0	0	0	0	0	5	5	5	7	**7**	7	9	9	12
4	0	0	0	0	7	7	7	7	7	12	12	12	14	**14**

Programação Dinâmica

- Outros algoritmos importantes baseados em programação dinâmica que veremos ao longo do curso:
  - Maior Subsequência Comum

Técnicas de Projeto de Algoritmos

- Infelizmente, não existe uma técnica que seja a melhor dentre todas.

- Um problema pode ser resolvido de maneira mais eficiente adotando-se uma técnica do que outra.

- Uma técnica pode levar a um algoritmo $O(2^n)$ e outra técnica a um algoritmo $O(n^2)$ na resolução de um mesmo problema.

Exercícios

Lista de Exercícios 05 – Programação Dinâmica

http://www.inf.puc-rio.br/~elima/paa/

Leitura Complementar

- Levitin. **Introduction to the Design and Analysis of Algorithms**, 3rd Edition, 2011.

- **Capítulo 8: Dynamic Programming**