Algoritimos e Estruturas de Dados III
CIC210
Programação Dinâmica

Haroldo Gambini Santos

Universidade Federal de Ouro Preto - UFOP

3 de setembro de 2009

Notas

Conteúdo

Notas

Seção

Notas

Números de Fibonacci

Relação de Recorrência

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

$$\{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots\}$$

Aplicações

- Biologia, Demografia
- Arquitetura, Música
- ...

Notas

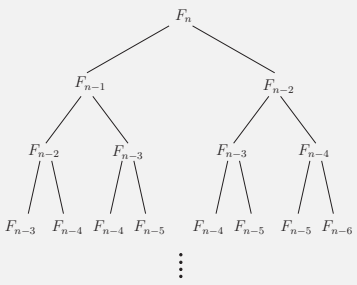
Números de Fibonacci

Um Algoritmo Simples

```
1 função fibonacci( n )
2 se  n < 2 então
3     retorne n;
4 senão
5     retorne fibonacci ( n − 1 ) + fibonacci ( n − 2 );
6 fim
```


Notas

Números de Fibonacci



Complexidade Exponencial !

Notas

Números de Fibonacci

Calculando em $O(n)$

```
1  função fibonacci( n )
2  anterior = 0; corrente = 0;
3  se  n < 2 então
4      retorne n;
5  senão
6      para  i = 1 até n − 1 faça
7          novo = anterior + corrente;
8          anterior = corrente;
9          corrente = novo;
10     fim
11 fim
12 retorne corrente;
```


Notas

Programação Dinâmica

Estratégia *Bottom Up*

- Casos menores são inicialmente calculados
- Resultados são armazenados (usualmente em uma tabela)
- Cálculos subseqüentes utilizam valores pré-calculados

Notas

Programação Dinâmica

Estratégia *Top Down*

- Utiliza métodos recursivos
- Métodos recursivos são instrumentados para salvar e reutilizar resultados pré-calculados
- **Memoização**

Notas

Programação Dinâmica

Aplicabilidade em problemas com:

- Sub-estruturas ótimas
  - A solução ótima para um determinado problema pode ser obtida combinando-se soluções ótimas dos seus sub-problemas
- Sub-problemas sobrepostos
  - Necessidade dos mesmos problemas serem resolvidos muitas vezes

Notas

DC × PD

- Divisão e Conquista
  - Eficiente quando subproblemas são independentes
  - Alguns subproblemas podem ser resolvidos muitas vezes
- Programação Dinâmica
  - Apropriado quando subproblemas compartilham subproblemas
  - Cada subproblema é resolvido somente uma vez
  - Resultados são armazenados
  - Troca-se **memória** por **tempo de processamento**

Notas

O Problema da Mochila

Exemplo

item	A	B	C	D	E
tamanho	3	4	7	8	9
lucro	4	5	10	11	13

Mochila com capacidade 17

Soluções de lucro ótimo

- Valor ótimo: 24
  - D E
  - A C C
  - A A B C
  - A A A B B

Notas

O Problema da Mochila

Implementação Recursiva

```
1  função mochila( cap, n, tamanho, lucro )
2  lucroMax = 0;
3  para i = 1 até n faça
4      espaco = cap − tamanho[i];
5      se espaco ≥ 0 então
6          lucroComItem =
           mochila(espaco, n, tamanho, lucro) + lucro[i];
7          se lucroComItem > lucroMax então
8              lucroMax = lucroComItem;
9          fim
10     fim
11 fim
12 retorna lucroMax;
```


Notas

O Problema da Mochila

Implementação com Programação Dinâmica - Memoização

```
1  lucroMaxCap[c] = desconhecido     ∀c;
2  função mochila( cap, n, tamanho, lucro )
3  lucroMax = 0;
4  se lucroMaxCap[cap] ≠ desconhecido então
5      retorna lucroMaxCap[cap];
6  senão
7      para i = 1 até n faça
8          espaco = cap − tamanho[i];
9          se espaco ≥ 0 então
10             lucroComItem = mochila(espaco, n, tamanho, lucro) + lucro[i];
11             se lucroComItem > lucroMax então
12                 lucroMax = lucroComItem;
13             fim
14         fim
15     fim
16     lucroMaxCap[cap] = lucroMax;
17 fim
18 retorna lucroMax;
```


Notas

O Problema da Mochila

Implementação com Programação Dinâmica - *Bottom Up*

```
1  função mochila( cap, n, tamanho, lucro )
2  lucroMaxCap[0] = 0;
3  para c = 1 até cap faça
4      lucroMaxCap[c] = 0;
5      para i = 1 até n faça
6          se (tamanho[i] ≤ c) então
7              capRestante = c − tamanho[i];
8              lucroComItem = lucroMaxCap[capRestante] + lucro[i];
9              se (lucroComItem > lucroMaxCap[c]) então
10                 lucroMaxCap[c] = lucroComItem;
11             fim
12         fim
13     fim
14 fim
15 retorna lucroMaxCap[cap];
```


Notas

O Problema da Mochila - Função Recursiva

Dados

- Items $i = \{1, \ldots, n\}$
  - peso p_i
  - lucro l_i

Função

$M(p)$ máximo de lucro que pode ser obtido com mochila de capacidade p

$$M(p) = \max_{i:p_i \leq p} \{M(p - p_i) + l_i\}$$

Notas

O Problema da Mochila sem Repetições

- Consideração adicional:
  - Um dado item j será incluso ou não na solução ?

Função

$$M(p, j) = \max\{$$
$$M(p - p_i, j - 1) + l_i \ ,$$
$$M(p, j - 1)$$
$$\}$$

Máximo de lucro que pode ser obtido COM ou SEM a inserção do item j.

Notas

Notas