PROGRAMAÇÃO DINÂMICA

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Introdução

Programação dinâmica: estratégia para resolver problemas cuja solução consiste em resolver subproblemas não-disjuntos

Sequência de Fibonacci

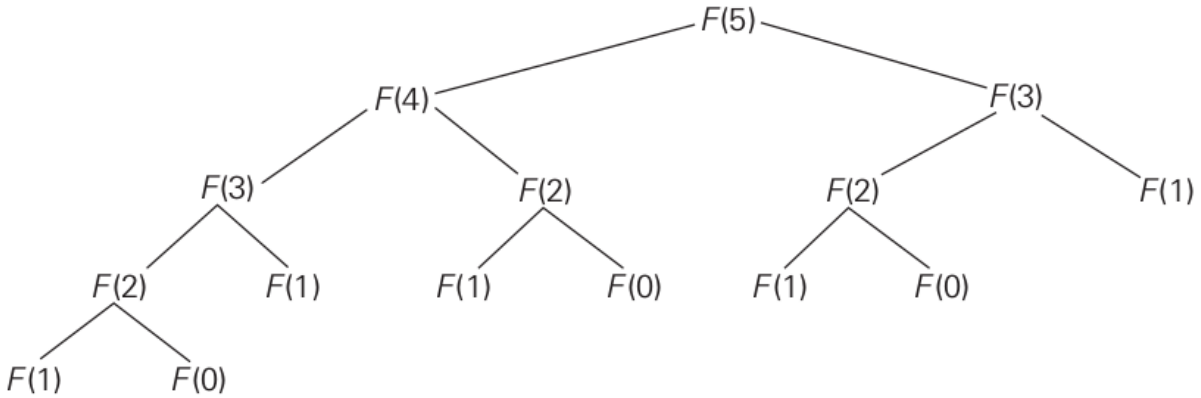
- $F(n) = F(n-1) + F(n-2)$, para $n > 1$
- $F(0) = 0$ e $F(1) = 1$

Considerando que $n \geq 0$:

Algoritmo: int Fib(n)

```
1  if n ≤ 1 then return n;
2  else return Fib(n − 1) + Fib(n − 2);
```

Introdução



[1]

Considerando que $n \geq 0$:

Algoritmo: int Fib2(n)

```
1  F[0], F[1] ← 0, 1;
2  for i ← 2 to n do
3  |   F[i] ← F[i − 1] + F[i − 2];
4  return F[n];
```

[1] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Introdução

Abordagens diretas de programação dinâmica podem ser vistas como uma variação de space-for-time trade-off.

Em alguns casos, é possível evitar o uso de espaço extra

- Em Fibonacci, armazenar somente os dois últimos elementos

Abordagens para programação dinâmica

- Bottom-up: resolve todos os subproblemas
- Top-down: resolve alguns subproblemas

Considerando que $n \geq 0$:

Algoritmo: int Fib3(F[0..n], n)

```
1  if F[n] = −1 then
2      if n ≤ 1 then F[n] ← n;
3      else F[n] ← Fib3(F, n − 1) + Fib3(F, n − 2);
4  return F[n];
```

Problema da fila de moedas

Considerando uma fila de n moedas com valores $c_1, c_2, ..., c_n$ não necessariamente distintos, coletar a maior quantia sem poder escolher duas moedas adjacentes.

Relação de recorrência:

- $F(n) = max\{c_n + F(n-2), F(n-1)\}$, para $n > 1$
- $F(0) = 0$ e $F(1) = c_1$

Algoritmo: int CoinRow(C[1..n])

```
1  F[0], F[1] ← 0, C[1];
2  for i ← 2 to n do
3      F[i] ← max(C[i] + F[i − 2], F[i − 1]);
4  return F[n];
```

Problema da fila de moedas

Considerando: 5, 1, 2, 10, 6 e 2

$F[0] = 0,\ F[1] = c_1 = 5$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5					

$F[2] = \max\{1 + 0, 5\} = 5$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5				

$F[3] = \max\{2 + 5, 5\} = 7$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7			

$F[4] = \max\{10 + 5, 7\} = 15$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15		

$F[5] = \max\{6 + 7, 15\} = 15$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	

$F[6] = \max\{2 + 15, 15\} = 17$

index	0	1	2	3	4	5	6
C		5	1	2	10	6	2
F	0	5	5	7	15	15	**17**

[2]

[2] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Problema da fila de moedas

Reconstruindo a solução:

- $F(6) = c_6 + F(4)$, logo c_6 faz parte da solução
- $F(4) = c_4 + F(2)$, logo c_4 faz parte da solução
- $F(2) = F(1)$, logo, c_2 não faz parte da solução, mas c_1 faz

Problema do troco mínimo

Troco de valor n usando a menor quantidade de moedas ($d_1 < d_2 < ... < d_m$ e $d_1 = 1$), assumindo uma quantidade ilimitada de cada moeda.

Relação de recorrência:

- $F(n) = min_{j:n \geq d_j}\{F(n - d_j)\} + 1$, para $n > 0$
- $F(0) = 0$

Algoritmo: int ChangeMaking(D[1..m], n)

```
1  F[0] ← 0;
2  for i ← 1 to n do
3      temp, j ← ∞, 1;
4      while j ≤ m ∧ i ≥ D[j] do
5          temp ← min(F[i − D[j]], temp);
6          j ← j + 1;
7      F[i] ← temp + 1;
8  return F[n];
```

Problema do troco mínimo

Considerando: $n = 6$ e moedas $1, 3$ e 4

$F[0] = 0$

n	0	1	2	3	4	5	6
F	0						

$F[1] = \min\{F[1-1]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1					

$F[2] = \min\{F[2-1]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2				

$F[3] = \min\{F[3-1], F[3-3]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1	2	1			

$F[4] = \min\{F[4-1], F[4-3], F[4-4]\} + 1 = 1$

n	0	1	2	3	4	5	6
F	0	1	2	1	1		

$F[5] = \min\{F[5-1], F[5-3], F[5-4]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	

$F[6] = \min\{F[6-1], F[6-3], F[6-4]\} + 1 = 2$

n	0	1	2	3	4	5	6
F	0	1	2	1	1	2	**2**

3

[3] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Problema da coleta de moedas

Considerando um grid (n,m) com no máximo 1 moeda por célula, coletar o máximo de moedas, saindo de (1,1) até (n,m), andando para a direita ou para baixo (sempre coletando as moedas no caminho).

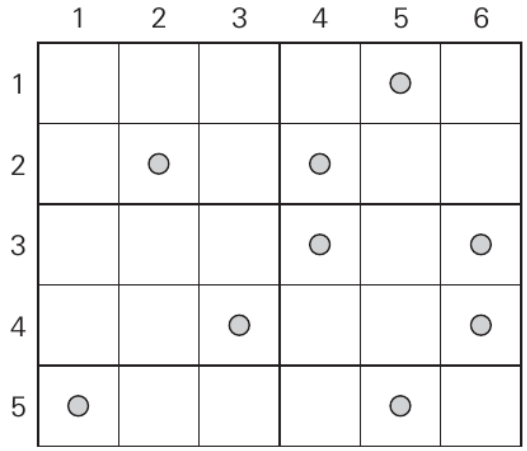
Relação de recorrência:

- $F(i,j) = max\{F(i-1,j), F(i,j-1)\} + c_{ij}$ para $1 \leq i \leq n, 1 \leq j \leq m$
- $F(0,j) = 0$ para $1 \leq j \leq m$, $F(i,0) = 0$ para $1 \leq i \leq n$

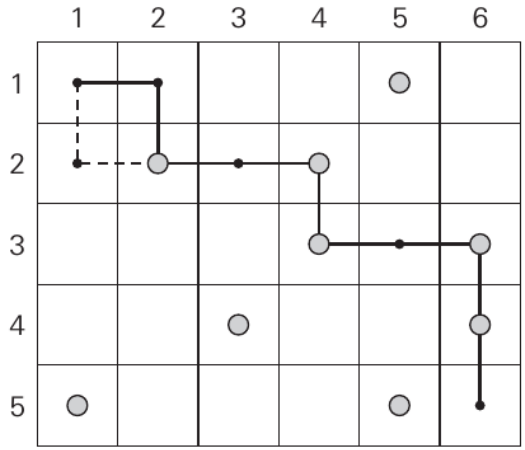
Algoritmo: int RobotCoinCollection(C[1..n, 1..m])

```
1  F[1,1] ← C[1,1];
2  for j ← 2 to m do  F[1,j] ← F[1,j−1] + C[1,j] ;
3  for i ← 2 to n do
4  |    F[i,1] ← F[i−1,1] + C[i,1];
5  |    for j ← 2 to m do
6  |    |    F[i,j] ← max(F[i−1,j], F[i,j−1]) + C[i,j];
7  return F[n,m];
```

Problema da coleta de moedas



	1	2	3	4	5	6
1	0	0	0	0	1	1
2	0	1	1	2	2	2
3	0	1	1	3	3	4
4	0	1	2	3	3	5
5	1	1	2	3	4	**5**



4

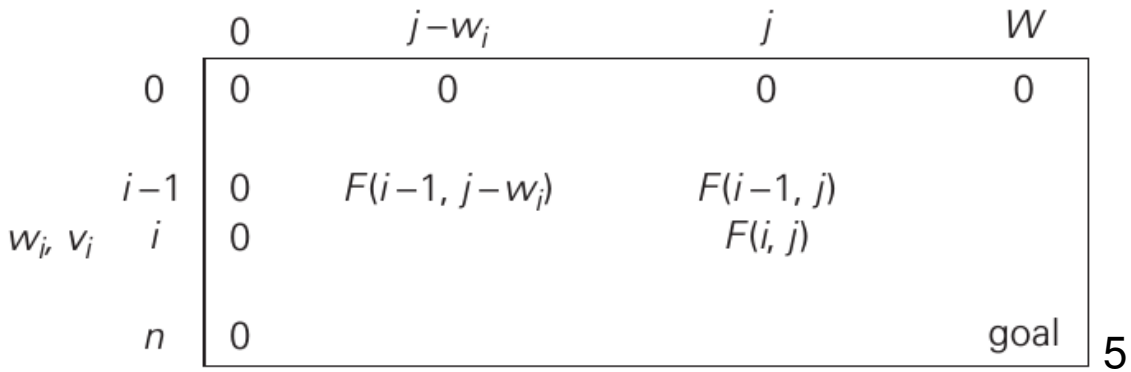
[4] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Knapsack (0-1)

Dado n itens com peso w_i e valor v_i, onde $i = 1, 2, \cdots, n$, e uma capacidade W, encontrar o subconjunto de itens mais valioso que cabe na mochila.

Relação de recorrência:

- $F(i,j) = max\{F(i-1,j), v_i + F(i-1,j-w_i)\}$, se $j - w_i \geq 0$
- $F(i,j) = F(i-1,j)$, se $j - w_i < 0$
- $F(0,j) = 0$, para $j \geq 0$, $F(i,0) = 0$, para $i \geq 0$



[5]

[5] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Knapsack (0-1): bottom-up

		capacity j					
	i	0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2,\ v_1 = 12$	1	0	0	12	12	12	12
$w_2 = 1,\ v_2 = 10$	2	0	10	12	22	22	22
$w_3 = 3,\ v_3 = 20$	3	0	10	12	22	30	32
$w_4 = 2,\ v_4 = 15$	4	0	10	15	25	30	**37**

Algoritmo: int Knapsack(n,W,w[1..n],v[1..n],F[0..n,0..W])

```
1  for i ← 0 to n do
2      for j ← 0 to W do
3          if i = 0 ∨ j = 0 then F[i][j] ← 0;
4          else if w[i] ≤ j then
               F[i][j] ← max(F[i − 1][j], v[i] + F[i − 1][j − w[i]]) ;
5          else F[i][j] ← F[i − 1][j] ;
6  return F[n][W];
```

[6] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Knapsack (0-1): top-down | 0,j e i,0 = 0, outras = -1

		capacity j					
	i	0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2,\ v_1 = 12$	1	0	0	12	12	12	12
$w_2 = 1,\ v_2 = 10$	2	0	—	12	22	—	22
$w_3 = 3,\ v_3 = 20$	3	0	—	—	22	—	32
$w_4 = 2,\ v_4 = 15$	4	0	—	—	—	—	**37**

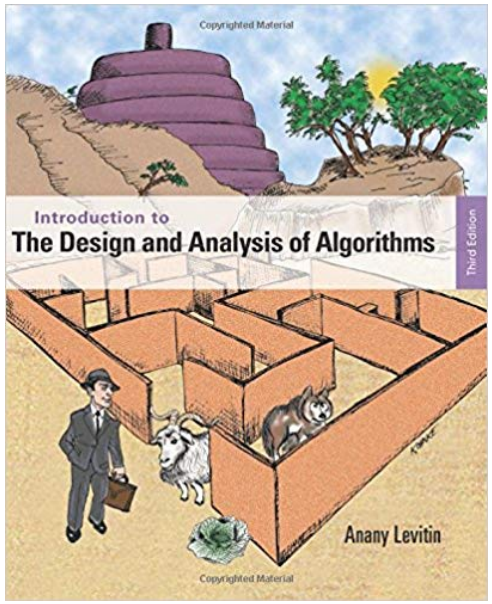
[7]

Algoritmo: int MFKnapsack(i,j,w[1..n],v[1..n],F[0..n,0..W])

```
1  if F[i, j] < 0 then
2      if j < w[i] then value ← MFKnapsack(i − 1, j, w, v, F) ;
3      else
4          value ← max(MFKnapsack(i − 1, j, w, v, F),
5                      v[i] + MFKnapsack(i − 1, j − w[i]);
6      F[i, j] ← value;
7  return F[i, j];
```

[7] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Bibliografia + leitura recomendada



Capítulo 9 (pp. 283–296)
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3a edição. Pearson. 2011.

PROGRAMAÇÃO DINÂMICA

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil