Projeto e Análise de Algoritmos

Prof. Dr. Ednaldo B. Pizzolato

www.dc.ufscar.br/~ednaldo ednaldo@dc.ufscar.br

RELAÇÃO ESPAÇO X TEMPO E PROGRAMAÇÃO DINÂMICA

Agenda

- Parte A – Relação Espaço x Tempo
  - Introdução
  - Algoritmo de ordenação
  - String matching
    - Algoritmo de Horspool
    - Algoritmo de Boyer-Moore (intro)
  - Hashing (intro)
  - B-Trees (intro)
- Parte B – Programação Dinâmica

Programação Dinâmica

Programação Dinâmica

- Programação dinâmica (PD) é um paradigma de programação que tem como objetivo reduzir o tempo de execução de um programa utilizando soluções ótimas a partir de subproblemas previamente calculados.
- De maneira informal, é uma maneira de implementar algoritmos recursivos utilizando uma abordagem bottom-up.

Programação Dinâmica

- A abordagem é iterativa, começando por instâncias menores, e usando uma tabela para guardar os resultados das instâncias anteriores.
- O uso da tabela para guardar os resultados das instâncias anteriores evita a necessidade de computar estas instâncias de maneira repetida.

Programação dinâmica

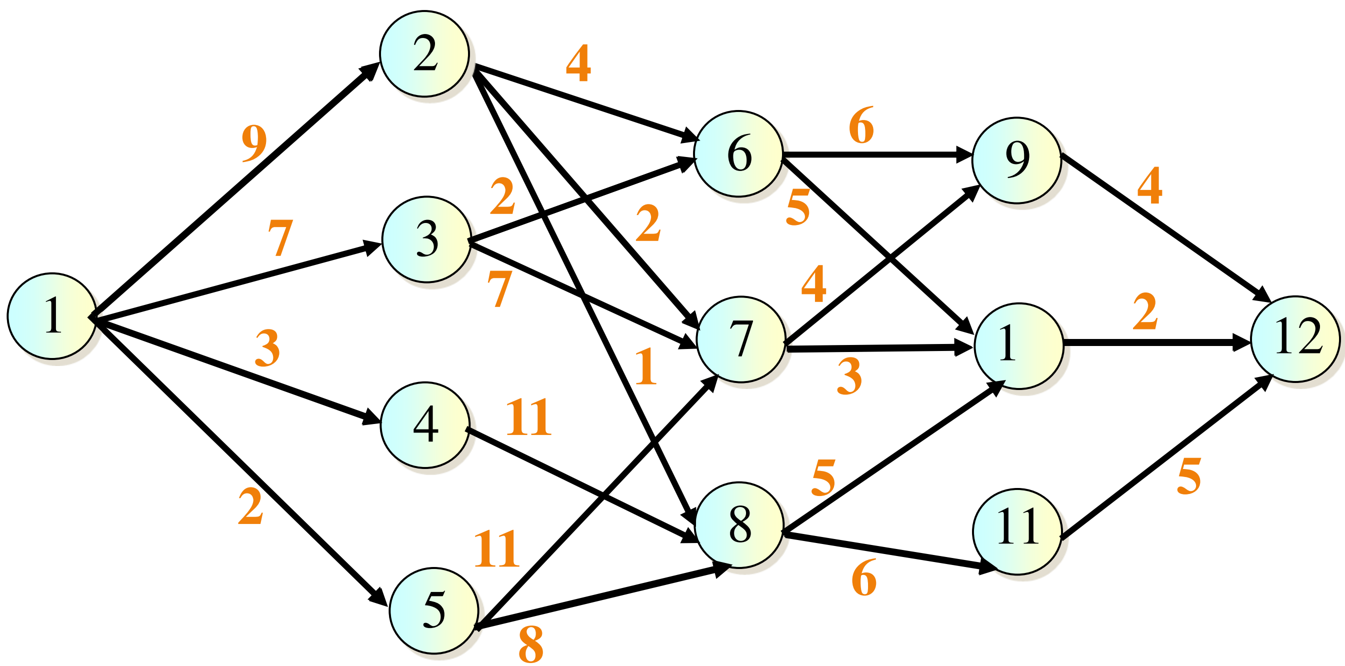
- Principio de Optimalidade

Numa sequência ótima de escolhas ou decisões cada subsequência deve também ser ótima.

Por exemplo: O menor caminho de São Carlos a São Paulo passando por Campinas é dado pelo menor caminho de São Carlos a Campinas com o menor caminho de Campinas a São Paulo.

Programação Dinâmica

Problema: determinar o caminho mais curto de 1 a 12 no grafo abaixo



Programação Dinâmica

- Passos:
  - 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.

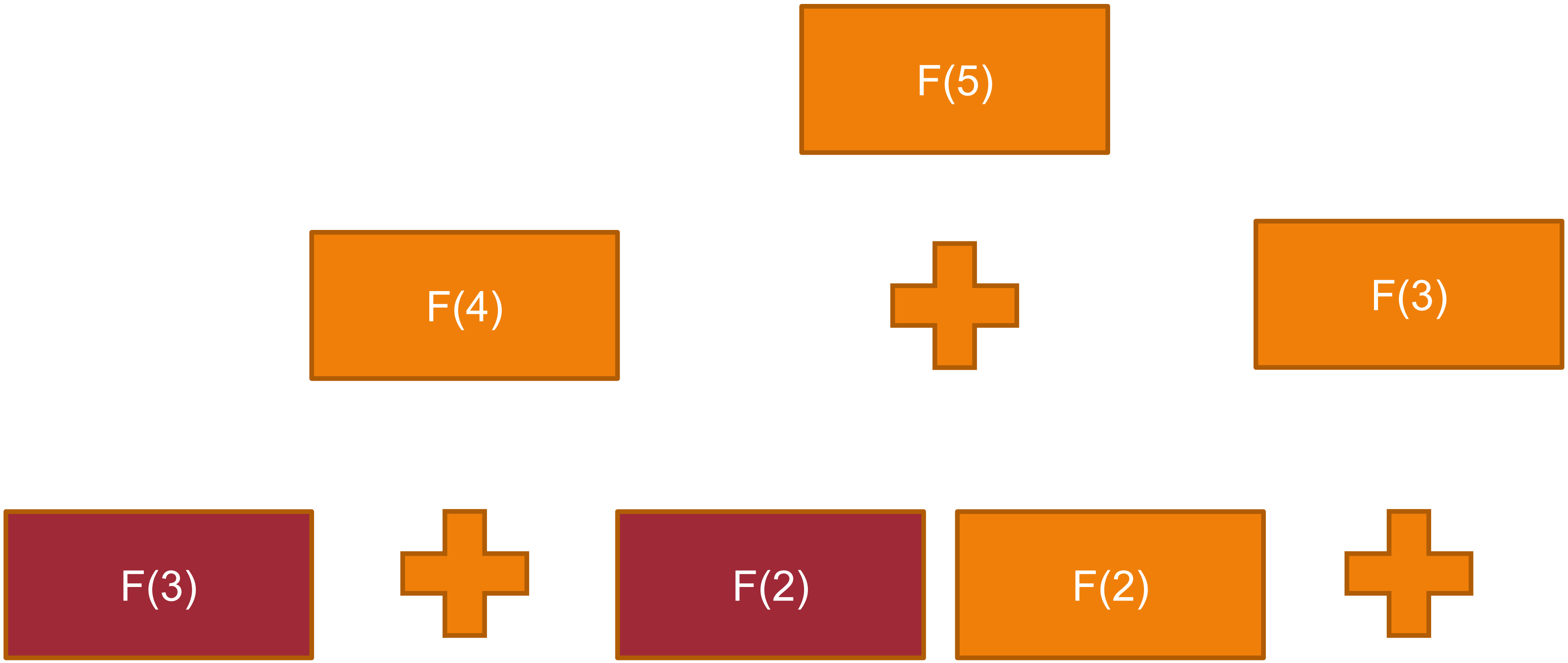
Programação Dinâmica

- Quando é possível utilizar Programação Dinâmica?
  - R1: Quando a solução ótima de um subproblema pode ser composta das soluções ótimas dos subproblemas;
  - R2: Quando o cálculo da solução ótima implica muitas vezes o cálculo do mesmo subproblema (lembra de Fibonacci?).

Programação Dinâmica

$$Fib(n) = \begin{cases} 0 \\ 1 \\ Fib(n-1) + Fib(n-2) \end{cases}$$

Programação Dinâmica

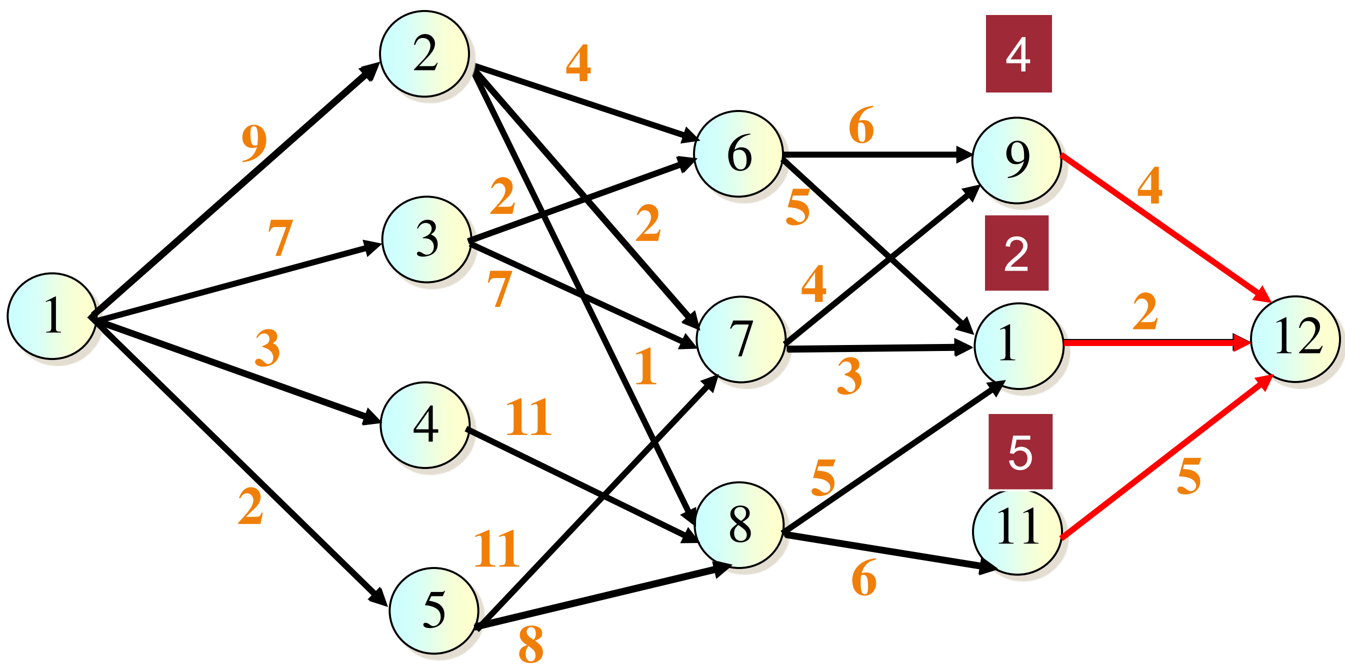


Programação Dinâmica

```
int Fibonacci ( int n )
{
     int V[n+1];
     int i;
     if (n==0)
            return 0;
    else if (n==1)
               return 1;
           else{
                    V[0]=0;
                    V[1]=1;
                   for(i=2; i<=n;i++)
                      V[i]=V[i-1] + V[i-2];
                 }
    return V[i];
}
```

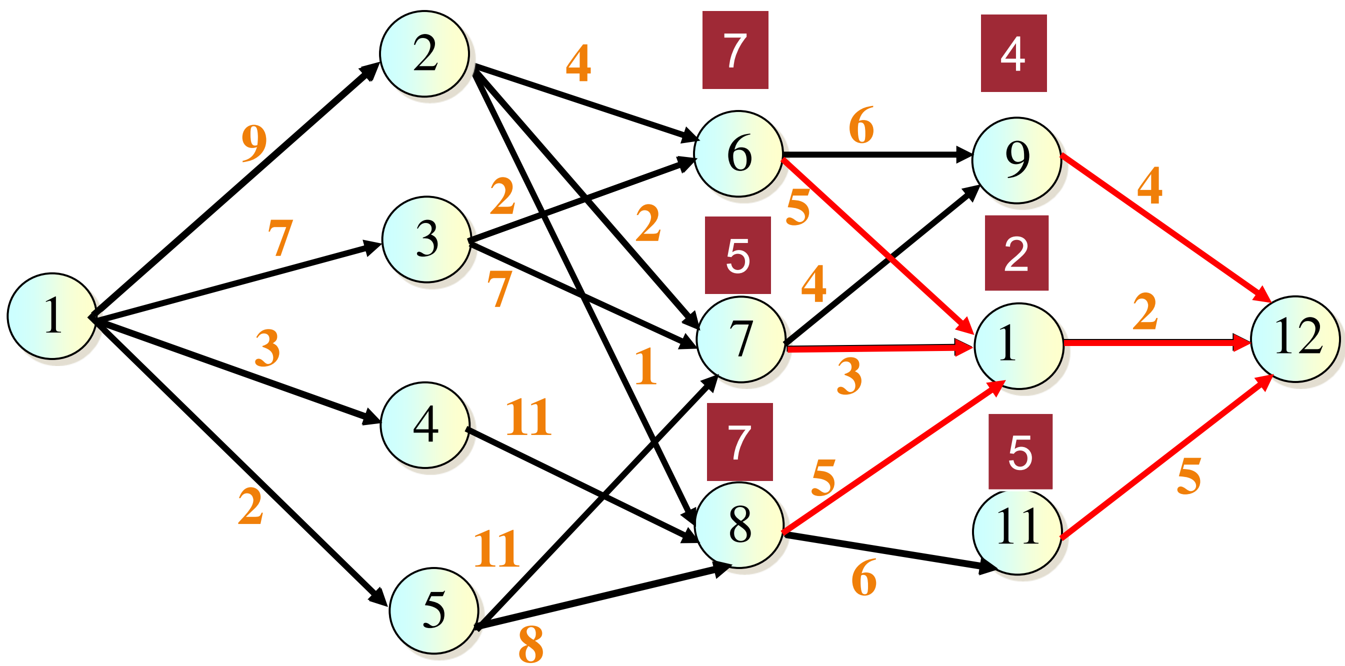
Programação Dinâmica

- Lembra do problema do menor caminho entre 1 e 12?



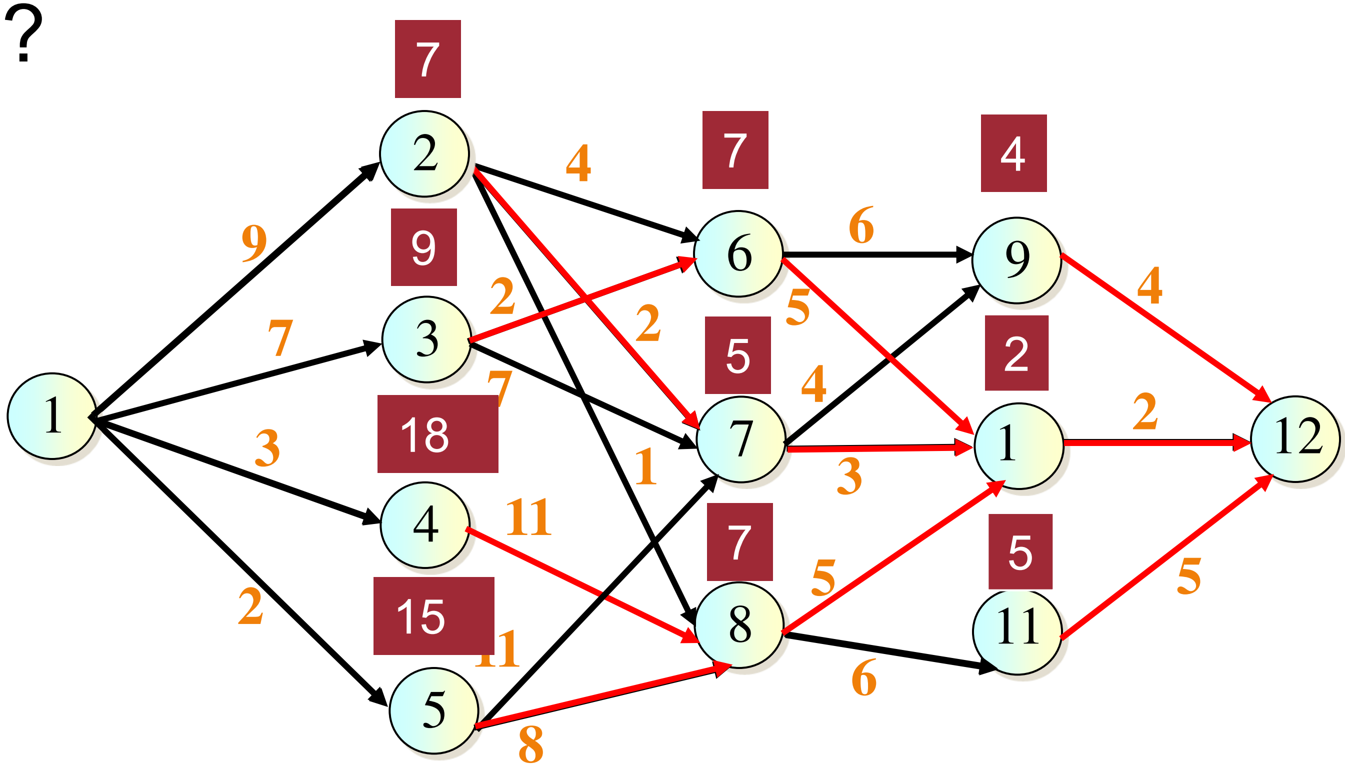
Programação Dinâmica

- Lembra do problema do menor caminho entre 1 e 12?



Programação Dinâmica

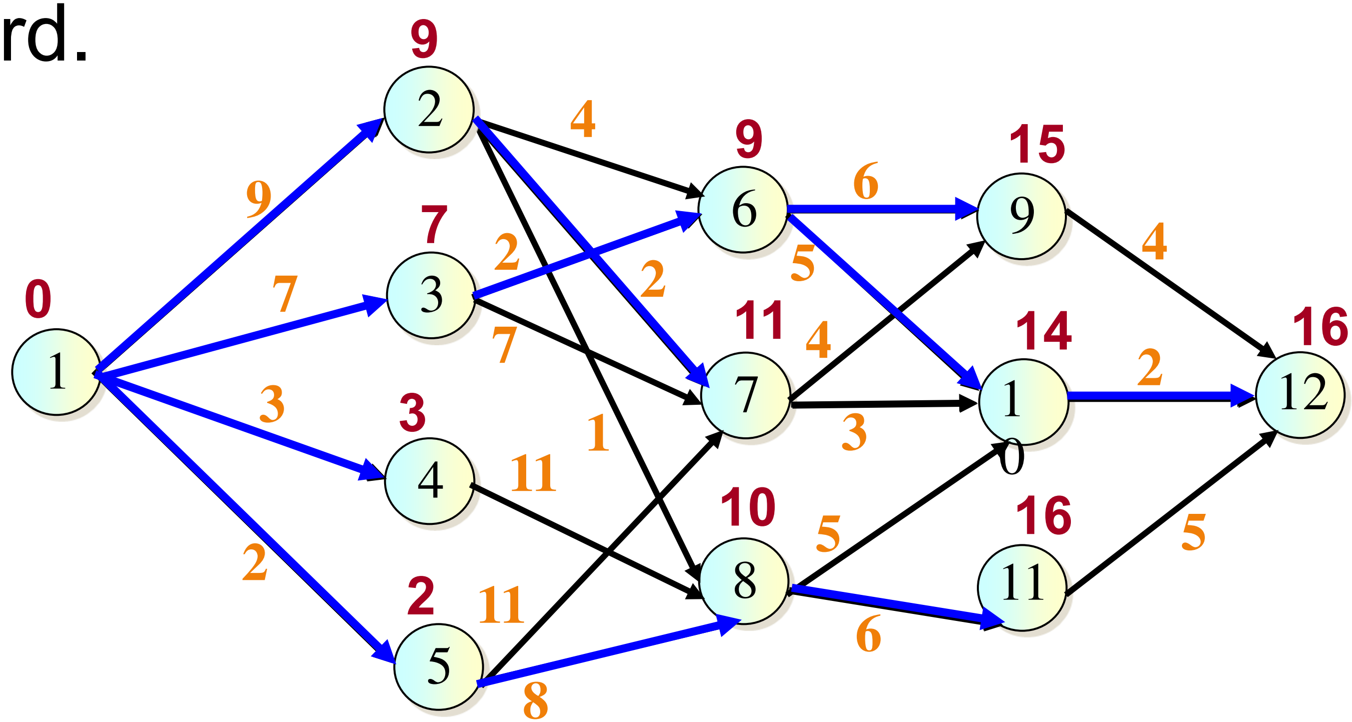
- Lembra do problema do menor caminho entre 1 e 12?



Procedimento Backwards

Programação Dinâmica

- De forma semelhante também é possível construir o caminho mais curto por procedimento forward.



Programação Dinâmica

- Aplicação a problemas de decisões seqüenciais: cada decisão aplicada a um estado em determinado estágio leva a um estado do estágio imediatamente seguinte.

- Princípio da otimalidade: uma seqüência ótima de decisões tem a propriedade de que quaisquer que sejam o estado e a decisão inicial, as decisões remanescentes constituem uma seqüência ótima de decisões com relação ao estado decorrente da primeira decisão.

- Alternativamente: “toda subtrajetória da trajetória ótima é ótima com relação a suas extremidades inicial e final”.

Programação Dinâmica

- Método exato para resolver problemas de programação inteira que envolvem apenas decisões sequenciais, em que cada nova decisão depende apenas do estado do sistema, mas não das decisões anteriores (isto é, da forma como este estado foi atingido).

Programação Dinâmica

- Principais conceitos envolvidos:
  - estágios (etapas)
  - estados
  - decisões
  - função critério a ser otimizada

Programação Dinâmica

- Roteiro de aplicação:
  - Identificar um modelo de decisões seqüenciais através de seus estágios.
  - Assegurar-se de que cada solução viável (ou trajetória) pode ser vista como uma seqüência de decisões tomadas a cada estágio, de modo tal que seu custo seja igual à soma dos custos das decisões individuais.
  - Definir o conceito de estado como a resultante de todas as decisões relevantes tomadas no passado (caso forward).

    (alternativamente, definir o conceito de estado como a resultante de todas as decisões relevantes tomadas no futuro no caso backwards)
  - Determinar as transições de estado possíveis.
  - Atribuir o custo de cada transição de estado à decisão correspondente.
  - Escrever uma recursão que defina o custo ótimo do estado inicial até o estado final.

Programação Dinâmica – exemplos

- Exemplo 1: Existe uma sequencia de n moedas cujos valores são inteiros positivos c_1, c_2, ... c_n. O objetivo é pegar a maior quantidade de dinheiro de tal forma que não se pode pegar duas moedas adjacentes.

5	1	2	10	6	2

Programação Dinâmica – exemplos

Seja F(n) o valor máximo que pode ser pego em uma fileira de n moedas. Podemos gerar uma recorrência criando 2 grupos: a) que contêm a última moeda ou b) que não a contêm.

5	1	2	10	6	2

Programação Dinâmica – exemplos

Está claro que F(0) é 0 (nenhuma moeda foi pega) e F(1) = c_1 (a moeda 1 foi pega). O maior valor que se pode pegar é o máximo de 2 situações:

1) o valor da moeda atual + tudo o que foi pego antes da moeda vizinha (anterior); ou

2) o valor de tudo o que foi pego até a moeda vizinha (anterior).

5	1	2	10	6	2

Programação Dinâmica – exemplos

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

$$F(0) = 0; F(1) = 5$$

Programação Dinâmica – exemplos

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

$$F(2) = \max\{1 + 0, 5\} = 5$$

Programação Dinâmica – exemplos

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

$$F(3) = \max\{2+5, 5\} = 7$$

Programação Dinâmica – exemplos

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

$$F(4) = \max\{10 + 5, 7\} = 15$$

Programação Dinâmica – exemplos

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

$$F(5) = \max\{6+7, 15\} = 15$$

Programação Dinâmica – exemplos

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

$$F(6) = \max\{2 + 15, 15\} = 17$$

Programação Dinâmica – exemplos

```
Coin-row (C[1..n]
  F[0] ← 0
  F[1] ← C[1]
  para i  ← 2 até n faça
      F[i] ← max{C[i]+F[i-2],F[i-1])
  return F[n]
```

Programação Dinâmica – exemplos

- Exemplo 2: Várias moedas estão espalhadas em um tabuleiro n x m, sendo que não existe mais que uma moeda por célula.

				x	
	x		x		
			x		x
		x			x
x				x	

Programação Dinâmica – exemplos

- Exemplo 2: ... Um robô localizado na célula do topo esquerdo deve pegar o máximo de moedas possível e levá-las ao canto inferior direito.

				x	
	x		x		
			x		x
		x			x
x				x	

Programação Dinâmica – exemplos

- Exemplo 2: ... O robô só pode se mover uma célula para a direita ou uma para baixo. E se visitar uma célula com moeda deve pegá-la.

				x	
	x		x		
			x		x
		x			x
x				x	

Programação Dinâmica – exemplos

- Exemplo 2: ... O robô só pode se mover uma célula para a direita ou uma para baixo. E se visitar uma célula com moeda deve pegá-la.
- Solução: Seja F(i,j) o maior número de moedas que o robô consegue pegar e levar até a célula $c_{i,j}$. E, claro, só poderá chegar à referida célula ou vindo da célula esquerda ou vindo de cima.
- O maior número de moedas em cada caso é:
  - F(i-1,j)
  - F(i,j-1)

  E, claro, não existem células acima do topo e nem à esquerda da primeira coluna. Para estas células, assume-se que F(i-1,j) ou F(i,j-1) será igual a 0 (zero).

Programação Dinâmica – exemplos

Solução:

Assim, o máximo de moedas que o robô consegue levar até a célula $c_{i,j}$ (contemplando a eventual situação de haver moeda na célula $c_{i,j}$) é:

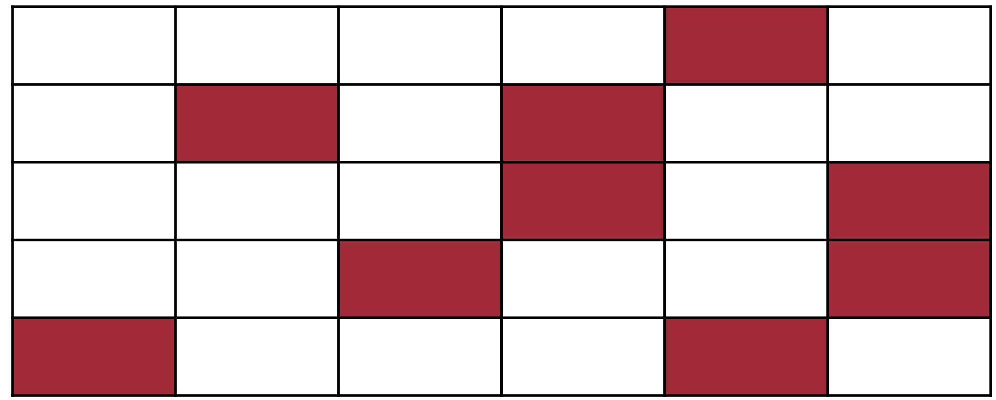
$$F(i,j) = \max\{F(i-1,j), F(i,j-1)\} + T_{i,j} \quad \text{para } 1 \leq i \leq n \text{ e } 1 \leq j \leq n$$

$$F(0,j) = 0 \quad \text{para } 1 \leq j \leq m$$

$$F(i,0) = 0 \quad \text{para } 1 \leq i \leq n$$

Onde $T_{i,j} = 1$ se há moeda na célula $c_{i,j}$ ou $T_{i,j} = 0$ caso contrário.

Programação Dinâmica – exemplos



```
Algoritmo Robo-Moeda(T[1..n,1..m])
F[1,1] ← T[1,1]
para j ← 2 até m faça
    F[1,j] ← F[1,j-1] + T[1,j]
para i ← 2 até n faça
    F[i,1] ← F[i-1,1] + T[i,1]
    para j ← 2 até m faça
        F[i,j] ← max(F[i-1],j], F[i,j-1]) + T[i,j]
retorna F[n,m]
```

Programação Dinâmica – exemplos

Algoritmo Robo-Moeda(T[1..n,1..m])

F[1,1] ← T[1,1]

para j ← 2 até m faça

  F[1,j] ← F[1,j-1] + T[1,j]

para i ← 2 até n faça

  F[i,1] ← F[i-1,1] + T[i,1]

  para j ← 2 até m faça

    F[i,j] ← max(F[i-1],j], F[i,j-1]) + T[i,j]

retorna F[n,m]

0	0	0	0	1	1
0	1	1	2	2	2
0	1	1	3	3	4
0	1	2	3	3	5
1	1	2	3	4	5

Programação Dinâmica – exemplos

Algoritmo Robo-Moeda(T[1..n,1..m])

F[1,1] ← T[1,1]

para j ← 2 até m faça

    F[1,j] ← F[1,j-1] + T[1,j]

para i ← 2 até n faça

    F[i,1] ← F[i-1,1] + T[i,1]

    para j ← 2 até m faça

        F[i,j] ← max(F[i-1],j], F[i,j-1]) + T[i,j]

retorna F[n,m]

0	0	0	0	1	1
0	1	1	2	2	2
0	1	1	3	3	4
0	1	2	3	3	5
1	1	2	3	4	5

Programação Dinâmica

- Vários outros exemplos podem ser obtidos através do estudo dos links no ambiente do moodle.

Resumo

- Programação Dinâmica é uma técnica para resolução de problemas que têm subproblemas com soluções em comum (overlapping).
- Tipicamente tais subproblemas surgem da recorrência relacionando sua solução com soluções de seus subproblemas menores e de mesmo tipo.
- A aplicação de programação dinâmica em problemas de otimização requer que os problemas satisfaçam o principio da optimalidade.

THE END