UNIVERSIDADE FEDERAL DE RORAIMA

Programação Dinâmica

Prof. Dr. Herbert Oliveira Rocha
herberthb12@gmail.com

Programação Dinâmica

- Na aplicação da **técnica de decomposição**, frequentemente há casos em que um mesmo subproblema aparece diversas vezes ao longo do processo.
- A decomposição pura e simples é incapaz de reconhecer este fato.
- Nestes casos, é conveniente utilizar uma variação da decomposição denominada de **PROGRAMAÇÃO DINÂMICA**.

Programação Dinâmica

- Variação da **técnica de decomposição**
  - PD é basicamente um **esquema de enumeração** de soluções que visa, através de uma abordagem de **divisão-e-conquista** (**decomposição**), minimizar o montante de computação a ser feito.
- Aplicada quando há casos em que um mesmo subproblema aparece diversas vezes ao longo do processo, onde a decomposição pura e simples é incapaz de reconhecer este fato.
- A PD resolve o subproblema uma vez só e reutiliza a solução toda vez que o mesmo aparecer novamente.

Programação Dinâmica

- Uma das técnicas mais usadas para a resolução de **Problemas de Otimização Combinatória** (**POC**).

POC: objetivo a otimizar + restrições a satisfazer

- Pode ser aplicada tanto para **problemas** em que possuem **solução polinomial** quanto para problemas cujo melhor algoritmo conhecido possui complexidade de **tempo exponencial**.

- A abordagem envolve a resolução de uma série de subproblemas até se achar a solução do problema original.

Programação Dinâmica

- Começa no fim e **funciona de trás para a frente** (de problemas menores a problemas cada vez maiores, até se resolver o problema original).

- Deve-se **armar** equações que são sempre **relações de recorrência**.

- Utiliza-se uma **tabela auxiliar** que contém **uma entrada para cada subproblema distinto**.
  - Quando um certo subproblema surge na computação pela 1ª. vez, ele é resolvido e sua solução armazenada na tabela. Quando um suproblema ocorrer novamente, a solução dada previamente é extraída da tabela em tempo constante.

Programação Dinâmica

PD: recursão x iteração

- A PD...
  - modela o problema recursivamente (define-se uma **relação de recorrência** que representa a solução do problema),
  - ... mas resolve-o iterativamente (uma **tabela** é usada para guardar os valores gerados, que são reutilizados toda vez que o mesmo subproblema ocorrer).

- Dizem que PD é uma **maneira esperta** de transformar recursão em iteração com o apoio de uma tabela.

Programação Dinâmica

Problema – Fatorial de *n*

1º. Método: usando decomposição pura

fatorial (n)

se n<= 1 então retornar 1;

senão retornar n x fatorial (n-1);

Problemas maiores

Problemas menores

fat

Programação Dinâmica

Problema – Fatorial de *n*

1º. Método: usando decomposição pura

fatorial (n)

se n <= 1 então retornar 1;

senão retornar n x fatorial (n-1);

Problemas maiores

Problemas menores

2º. Método: aplicando PD

fat[0] := fat[1] := 1;

para j := 2 a n faça

fat[j] := j x fat[j-1];

o que se quer calcular := o que já se calculou

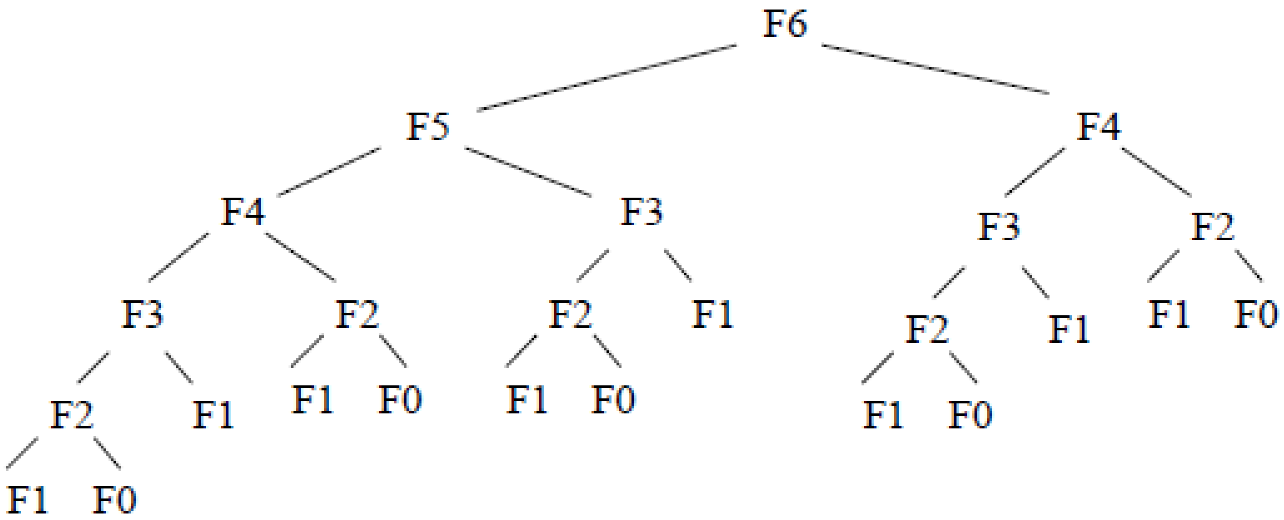
Problemas maiores

Problemas menores

	fat
n	
n-1	
...	
1	1
0	1

Programação Dinâmica

Problema – Fibonacci de *n*



Programação Dinâmica

Problema – Fibonacci de *n*

1º. Método: usando decomposição pura

fibo (n)

se n<= 2 então retornar n-1;

senão retornar fibo(n-1) + fibo(n-2) ;

Problemas maiores

Problemas menores

2º. Método: aplicando PD

fib[1] := 0; fib[2] := 1;

para j := 3 a n faça

fib[j] := fib[j-1] + fib[j-2] ;

o que se quer calcular **:=** o que já se calculou

Problemas maiores

Problemas menores

	fib
n	
n-1	
…	
2	**1**
1	**0**

ALGORITMOS GULOSOS

- Um **algoritmo guloso** sempre faz a melhor escolha no processo de decisão corrente.

- Espera-se que a **melhor escolha** a cada decisão **local** leve a uma melhor escolha para o problema global.

ALGORITMOS GULOSOS

- Algoritmos gulosos (do inglês, ***greedy***) utilizam o mesmo conceito de **subestrutura ótima** usado na Programação Dinâmica.
  - Fazem escolhas com **base em dados locais** para encontrar boas soluções.
  - Técnica pode ser usada para obter **resultados aproximados em alguns casos** (pois nem sempre fornecem o ótimo).
  - Tendem a ser mais fáceis de desenvolver do que os algoritmos de PD.

Programação Dinâmica

PROBLEMAS DE OTIMIZAÇÃO COMBINATÓRIA - POC

- Problemas que apresentam várias soluções, cada uma com um valor (custo) associado.
- Procura-se a solução com valor ótimo (mínimo ou máximo).
- Um solução geralmente apresenta uma estrutura: ela é composta de uma seqüência de escolhas. Tais escolhas devem ser feitas para se chegar à solução ótima.

objetivo a otimizar + condições a satisfazer = solução POC

função objetivo + restrições = solução ótima

PROGRAMAÇÃO DINÂMICA resolve POCs

- Caracterizar a estrutura de uma solução ótima.
- Definir recursivamente o valor de uma solução ótima.
- Calcular o valor da solução ótima em processo ***bottom-up*** (de problemas menores para maiores).
- Construir uma solução ótima a partir das informações calculadas.

Programação Dinâmica

Problema: Multiplicação de uma cadeia de matrizes

- Computar o produto de muitas matrizes eficientemente.
- Determinar uma ordem em que as matrizes sejam multiplicadas, de modo a minimizar o número de operações envolvidas!

Programação Dinâmica

Problema: Multiplicação de uma cadeia de matrizes

- Seja a sequência (cadeia) $<M_1, M_2, \ldots, M_n>$ de n matrizes. Computar o produto $M_1 x M_2 \ldots x M_n$ de forma a minimizar o número de multiplicações.

Programação Dinâmica

- As *n* matrizes são multiplicadas aos pares.
  - **Subproblema: multiplicação de duas matrizes**.
- Duas matrizes ***A:pxm*** *e* ***B:mxq*** podem ser multiplicadas usando ***pxmxq* multiplicações** escalares.
- Duas matrizes A e B podem ser multiplicadas se forem **compatíveis**.

  número de colunas de A = número de linhas de B

  - A(*pxm) x* B(*mxq*) *x* C(*pxq*)
    - O número de multiplicações é *pxmxq*

$$\begin{pmatrix} a_{11} & a_{12} \\ \boxed{a_{21} \; a_{22}} \\ a_{31} & a_{32} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix} = \begin{pmatrix} - & - & - \\ - & c_{22} & - \\ - & - & - \end{pmatrix}$$

Programação Dinâmica

- **Exemplo:**
  - Multiplicar as matrizes: A:10x100, B:100x5 e C:5x50.
    - <A, B, C> (10, 100, 5, 50)
  - Qual ordem escolher??
    - » **((AB)C)** = 10x100x5 + 10x5x50 = 5000 + 2500 = **7500**
    - » (A(BC)) = 100x5x50 + 10x100x50 = 25000 + 50000 = 75000
  - **A ordem das multiplicações faz muita diferença!**
- **Problema da "parentização"**
  - O custo da multiplicação é definido pela ordem em que os pares de matrizes são multiplicados.
  - O problema consiste em definir ONDE colocar os PARÊNTESES para agrupar as matrizes 2 a 2 e assim realizar o mínimo de multiplicações (para todas as possibilidades é exponencial).

Programação Dinâmica

- **Exemplo 2:**
  - Multiplicar as matrizes <A, B, C, D>:
    - » A:30x1, B:1x40, C:40x10 e D:10x25
      - <A, B, C, D> (30, 1, 40, 10, 25)
  - Qual ordem escolher??
    - » $(AB)C)D$ = 1200 + 12000 + 7500 = 20700
    - » $(AB)(CD)$ = 1200 + 10000 + 30000 = 41200
    - » $A((BC)D)$ = 400 + 250 + 750 = **1400**
  - **A ordem das multiplicações faz muita diferença!**
- Deseja-se uma sequência ótima para multiplicar:
  - $M_1 \times M_2 \times \ldots \times M_n$ onde M_i é uma matriz $d_{i-1} \times d_i$

Programação Dinâmica

1º. Método: força bruta (enumeração explícita)

- Testar todas as ordens possíveis.
    - parentizar 2 a 2;
    - equivalente ao problema de triangularizar um polígono convexo;
    - complexidade exponencial

$$P(n) = \begin{cases} 1, & se \quad n = 1 \\ \sum_{k=1}^{n-1} P(k)P(n-k), & se \quad n \geq 2 \end{cases}$$

$P(n)$ – nº de alternativas para colocação dos parênteses de uma sequência de n matrizes.

Programação Dinâmica

2º Método: Aplicando PD

$$M = M_1 \text{ x } \ldots \text{ x } M_n$$

Qualquer forma de colocar parêntesis em $M_i M_{i+1} \ldots M_j$ deve dividir a cadeia entre M_k e M_{k+1} , para algum inteiro k, $i \le k < j$.

$$M \;=\; (M_1 \text{ x } \ldots \text{ x } M_k) \quad \text{x} \quad (M_{k+1} \text{ x } \ldots \text{ x } M_n)$$

Custo de computar M ?

custo de computar $M_{i..k}$ + custo de computar $M_{k+1..j}$ + custo de multiplicar $M_{i..k}$ e $M_{k+1..j}$

– A subcadeia $M_i M_{i+1} \ldots M_k$ deve ter parentização ótima.
– A subcadeia $M_{k+1} M_{i+1} \ldots M_j$ deve ter parentização ótima.

Programação Dinâmica

2º Método: Aplicando PD

Seja k o índice da matriz mais à direita em M'.

Então, o problema de determinar M fica **decomposto em dois subproblemas**:

1 – determinar a ordem ótima de multiplicação de:

$$M' = M_1 \times \ldots \times M_k$$

$$M'' = M_{k+1} \times \ldots \times M_n$$

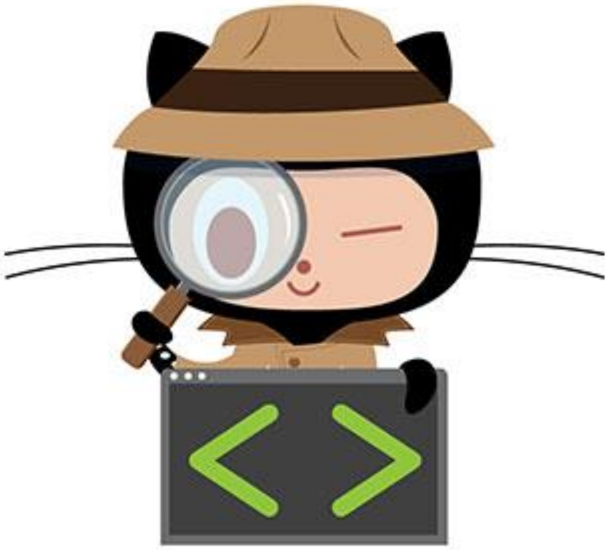
para k fixo, $1 \leq k \leq n$.

2 – calcular o nº de operações para obter $M_{1..n}$:

$$\left\{\begin{array}{l} \text{\# op. p/ obter } M' \\ \text{\# op. p/ obter } M'' \\ \text{\# op. p/ obter } M' \times M'' = d_0 d_k d_n \end{array}\right.$$

Notação: $M_{i..j}$ = resultado da avaliação de $M_i M_{i+1} \ldots M_j$ $(i \leq j)$

See you



Perguntas?