PICAT: Uma Linguagem de Programação Multiparadigma

Claudio Cesar de Sá

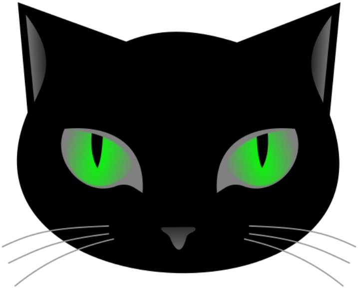
claudio.sa@udesc.br

Departamento de Ciência da Computação – DCC
Centro de Ciências e Tecnológias – CCT
Universidade do Estado de Santa Catarina – UDESC

9 de maio de 2019

Programação Dinâmica

- O que é a PD?
- Características
- Importância
- Exemplo



Programação Dinâmica (PD) – I

- A recursão embora elegante, esta é ineficiente. Ver a árvore de expansão de Fibonacci.

- A recursão embora elegante, esta é ineficiente. Ver a árvore de expansão de Fibonacci.
- Uma poderosa *técnica de programação* que contorna a complexidade de certos problemas exponenciais

Programação Dinâmica (PD) – I

- A recursão embora elegante, esta é ineficiente. Ver a árvore de expansão de Fibonacci.
- Uma poderosa *técnica de programação* que contorna a complexidade de certos problemas exponenciais
- O problema **deve** apresentar uma **regra de recorrência**, a qual torna-se uma estratégia para se **armazenar sequencialmente** resultados temporários/intermediários

Programação Dinâmica (PD) – I

- A recursão embora elegante, esta é ineficiente. Ver a árvore de expansão de Fibonacci.
- Uma poderosa *técnica de programação* que contorna a complexidade de certos problemas exponenciais
- O problema **deve** apresentar uma **regra de recorrência**, a qual torna-se uma estratégia para se **armazenar sequencialmente** resultados temporários/intermediários
- Estes cálculos de *instâncias menores* são armazenados numa **tabela dinâmica**

Programação Dinâmica (PD) – I

- A recursão embora elegante, esta é ineficiente. Ver a árvore de expansão de Fibonacci.
- Uma poderosa *técnica de programação* que contorna a complexidade de certos problemas exponenciais
- O problema **deve** apresentar uma **regra de recorrência**, a qual torna-se uma estratégia para se **armazenar sequencialmente** resultados temporários/intermediários
- Estes cálculos de *instâncias menores* são armazenados numa **tabela dinâmica**
- Esta *técnica de programação* utiliza uma *tabela dinâmica* nos cálculos intermediários, evitando a repetição do que já foi calculado anteriormente, é conhecida como: Programação Dinâmica, ou simplesmente: **PD**

Programação Dinâmica (PD) – II

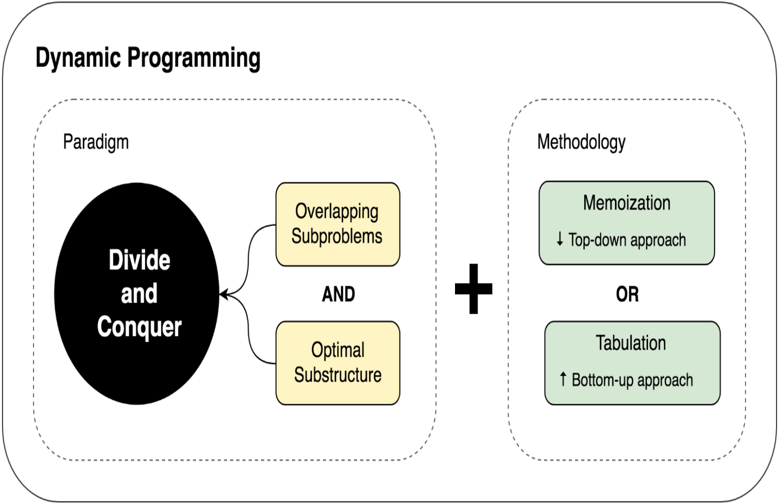


Figura 1: Conceitos da Programação Dinâmica – (PD) – Resumos

Requisitos para PD:

Programação Dinâmica (PD) – III

Requisitos para PD:

- **Subestrutura ótima**: um cálculo incremental, tal que os melhores sejam encontrados e armazenados para posterior reuso.

  Exemplo:

  1. $fat(0) = 1$
  2. $fat(1) = 1$

  Estes são os melhores resultados até então, pois o problema apresenta em seu interior soluções ótimas para subproblemas.

- **Sobreposição de problemas menores (subproblemas)**: de modo recorrente, possamos construir uma tabela para armazenar as soluções dos subproblemas, afim de evitar que elas sejam recalculadas.
  Exemplo:
  $fat(7) \rightarrow fat(6) \rightarrow ...fat(0)$ (*top-down*)
  $fat(0) \rightarrow fat(1) \rightarrow ...fat(7)$ (*botton-up*)
  onde esta sobreposição é descrita por:
  $fat(N) = N.fat(N-1)$

Aproximadamente, a PD é uma recursão apoiada por uma tabela de cálculos intermediários

Dica: `https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/dynamic-programming.html`

Programação Dinâmica (PD) – V

- Como Picat usa a recursão, na programação em lógica, nada mais natural do que esta ter a PD disponível

Programação Dinâmica (PD) – V

- Como Picat usa a recursão, na programação em lógica, nada mais natural do que esta ter a PD disponível
- O comando que cria uma tabela para um determinado predicado é o ***table***

Programação Dinâmica (PD) – V

- Como Picat usa a recursão, na programação em lógica, nada mais natural do que esta ter a PD disponível
- O comando que cria uma tabela para um determinado predicado é o ***table***
- O *table* é um dos elementos fortes do planejador do Picat (módulo *planner*)

Programação Dinâmica (PD) – V

- Como Picat usa a recursão, na programação em lógica, nada mais natural do que esta ter a PD disponível
- O comando que cria uma tabela para um determinado predicado é o ***table***
- O *table* é um dos elementos fortes do planejador do Picat (módulo *planner*)
- Assim a PD, faz a complexidade ser espacial devido o uso de memória em seus cálculos intermediários

Programação Dinâmica (PD) – V

- Como Picat usa a recursão, na programação em lógica, nada mais natural do que esta ter a PD disponível
- O comando que cria uma tabela para um determinado predicado é o ***table***
- O *table* é um dos elementos fortes do planejador do Picat (módulo *planner*)
- Assim a PD, faz a complexidade ser espacial devido o uso de memória em seus cálculos intermediários
- O exemplo escolhido para ilustrar a PD em Picat, veio do texto *Modeling and Solving AI Problems in Picat*, de Roman Barták e Neng-Fa

Exemplo de Uso da Programação Dinâmica – (PD)

- Seja o binômio $(x + y)^n$, conhecido como *Binômio de Newton*

Exemplo de Uso da Programação Dinâmica – (PD)

- Seja o binômio $(x+y)^n$, conhecido como *Binômio de Newton*
- Casos particulares são:
- $(x+y)^0 = 1$
- $(x+y)^1 = x+y$
- $(x+y)^2 = x^2 + 2xy + y^2$

Exemplo de Uso da Programação Dinâmica – (PD)

- Seja o binômio $(x+y)^n$, conhecido como *Binômio de Newton*
- Casos particulares são:
- $(x+y)^0 = 1$
- $(x+y)^1 = x + y$
- $(x+y)^2 = x^2 + 2xy + y^2$
- $(x+y)^2 = x^2y^0 + 2x^1y^1 + x^0y^2$
- $(x+y)^3 = x^3y^0 + 3x^2y^1 + 3x^1y^2 + x^0y^3$
- $(x+y)^4 = x^4y^0 + 4x^3y^1 + 6x^2y^2 + 4x^1y^3 + x^0y^4$.
- ....

Exemplo de Uso da Programação Dinâmica – (PD)

- Seja o binômio $(x+y)^n$, conhecido como *Binômio de Newton*
- Casos particulares são:
- $(x+y)^0 = 1$
- $(x+y)^1 = x + y$
- $(x+y)^2 = x^2 + 2xy + y^2$
- $(x+y)^2 = x^2y^0 + 2x^1y^1 + x^0y^2$
- $(x+y)^3 = x^3y^0 + 3x^2y^1 + 3x^1y^2 + x^0y^3$
- $(x+y)^4 = x^4y^0 + 4x^3y^1 + 6x^2y^2 + 4x^1y^3 + x^0y^4$.
- ....
- Como obter estes coeficientes polinômios?

Exemplo de Uso da Programação Dinâmica – (PD)

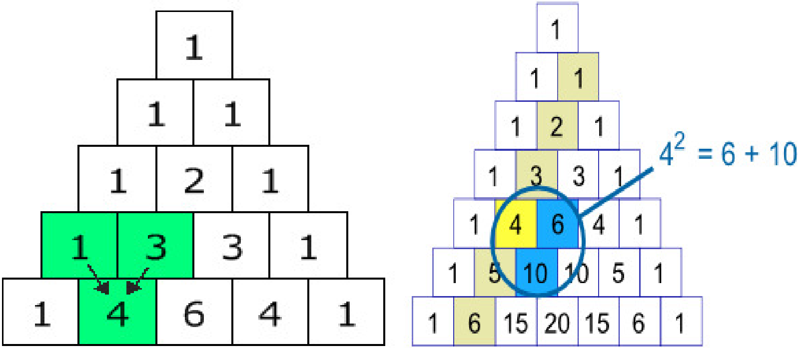


Figura 2: O triângulo de Pascal – suas propriedades

A Triangulo de Pascal

N \ p	0	1	2	3	4	5	6	
0	$\binom{0}{0}$							1
1	$\binom{1}{0}$	$\binom{1}{1}$						1 1
2	$\binom{2}{0}$	$\binom{2}{1}$	$\binom{2}{2}$					1 2 1
3	$\binom{3}{0}$	$\binom{3}{1}$	$\binom{3}{2}$	$\binom{3}{3}$				1 3 3 1
4	$\binom{4}{0}$	$\binom{4}{1}$	$\binom{4}{2}$	$\binom{4}{3}$	$\binom{4}{4}$			1 4 6 4 1
5	$\binom{5}{0}$	$\binom{5}{1}$	$\binom{5}{2}$	$\binom{5}{3}$	$\binom{5}{4}$	$\binom{5}{5}$		1 5 10 10 5 1
6	$\binom{6}{0}$	$\binom{6}{1}$	$\binom{6}{2}$	$\binom{6}{3}$	$\binom{6}{4}$	$\binom{6}{5}$	$\binom{6}{6}$	1 6 15 20 15 6 1

Figura 3: O triângulo de Pascal – Coeficientes Binomiais

Volte a Stiffel: 12

- O *coeficiente binomial*, também chamado de *número binomial*, de um número n, na classe k, consiste no número de combinações de n termos, k a k.

Formulação Matemática – I

- O *coeficiente binomial*, também chamado de *número binomial*, de um número n, na classe k, consiste no número de combinações de n termos, k a k.
- O número binomial de um número n, na classe k, pode ser escrito como:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k!}$$

- Alternativa ao cálculo do fatorial, tem-se a relação de Stiffel:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

Formulação Matemática – II

- Alternativa ao cálculo do fatorial, tem-se a relação de Stiffel:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

- O coeficiente binomial é muito utilizado no Triângulo de Pascal, onde o termo na linha n e coluna k é dado por: $\binom{n-1}{k-1}$

Formulação Matemática – II

- Alternativa ao cálculo do fatorial, tem-se a relação de Stiffel:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

- O coeficiente binomial é muito utilizado no Triângulo de Pascal, onde o termo na linha n e coluna k é dado por: $\binom{n-1}{k-1}$
- Complementado a relação de Stiffel, tem-se ainda:
  - $\binom{n}{0} = 1$ com $k = 0$
  - $\binom{n}{n} = 1$ com $k = n$
- Veja o triângulo novamente: 10

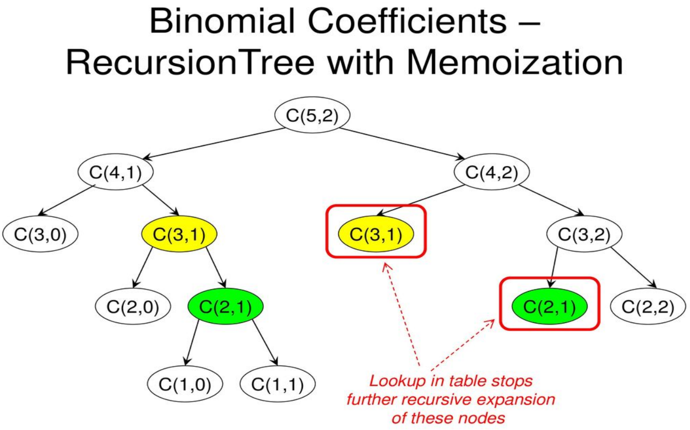


Figura 4: A árvore expandida de busca – *memoization*

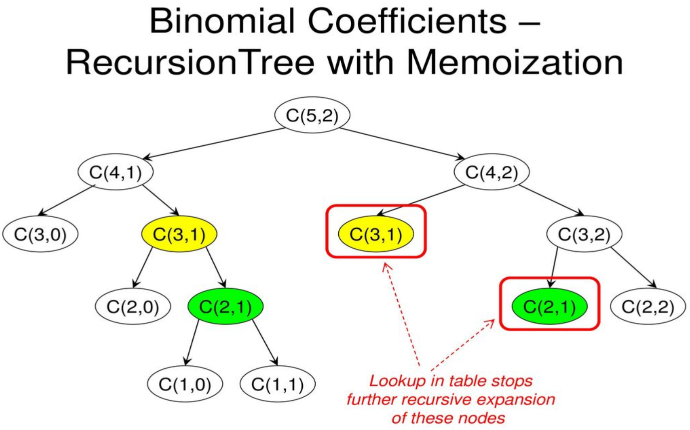


Figura 4: A árvore expandida de busca – *memoization*

A fórmula de Stiffel é recorrente e diretamente escrita em Picat.

Código em Partes

```
import datetime.   %%% para o statistics
import util.

table
c(_, 0) = 1.
c(N, N) = 1.
c(N,K) = c(N-1, K-1) + c(N-1, K).
```

- Relembrando: $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$
- Esta fórmula é semelhante com a sequência de Fibonacci, vista na seção de recursividade, mas aqui temos 2 argumentos em `c(N,K)`

```
main  ?=>
    statistics(runtime,_), % faz uma marca do 1o. statistics
    N = 10, %% ateh uns 30 ... são números grandes ... fatorial
     foreach(I in 0  .. N)
        foreach(J in 0  ..  I)
             printf("  %d", c(I,J))
           end,
          printf(" \n"),
      end,
    statistics(runtime, [T_Picat_ON, T_final]),
    T = (T_final) / 1000.0, %%% está em milisegundos
    printf("\n CPU time %f em SEGUNDOS ", T),
    printf("\n OVERALL PICAT CPU time %f em SEGUNDOS ", T_Picat_ON/1000
    printf(" \n =========================================\n ")
    %%% , fail descomente para multiplas solucoes
    .
main => printf("\n Para uma solução .... !!!!" ) .
```

- Acompanhar as explicações do código de: https://github.com/claudiosa/CCS/blob/master/picat/coeficiente_binomial_PD.pi
- Confira a execução

```
[ccs@gerzat picat]$ picat coeficiente_binomial_PD.pi
  1
  1   1
  1   2   1
  1   3   3   1
  1   4   6   4   1
  1   5   10   10   5   1
  1   6   15   20   15   6   1
  1   7   21   35   35   21   7   1
  1   8   28   56   70   56   28   8   1
  1   9   36   84   126   126   84   36   9   1
  1   10   45   120   210   252   210   120   45   10   1

 CPU time 0.000000 em SEGUNDOS
 OVERALL PICAT CPU time 0.009000 em SEGUNDOS
 =========================================
```

- Há outros métodos para se resolver estes problemas

Reflexões sobre PD

- Há outros métodos para se resolver estes problemas
- O comando *table* é a base do módulo *planner*, usado para resolver problemas de planejamento

Reflexões sobre PD

- Há outros métodos para se resolver estes problemas
- O comando *table* é a base do módulo *planner*, usado para resolver problemas de planejamento
- A PD é uma estratégia de programação bem poderosa

- Há outros métodos para se resolver estes problemas
- O comando *table* é a base do módulo *planner*, usado para resolver problemas de planejamento
- A PD é uma estratégia de programação bem poderosa
- Uso: sub-sequência máxima, menor distância entre 2 pontos num grafo, problema da mochila, soma de sub-conjuntos etc

Reflexões sobre PD

- Há outros métodos para se resolver estes problemas
- O comando *table* é a base do módulo *planner*, usado para resolver problemas de planejamento
- A PD é uma estratégia de programação bem poderosa
- Uso: sub-sequência máxima, menor distância entre 2 pontos num grafo, problema da mochila, soma de sub-conjuntos etc
- Assunto das próximas seções: Planejamento e PR