

# Programação Dinâmica

---

Estagiária PAE: María Lucía Castro

## Definição

---

*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

## Princípio de Otimalidade

---

- 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.

## Passos a seguir em PD

---

- 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 utilizando os valores computados.

## Quando é possível usar PD?

---

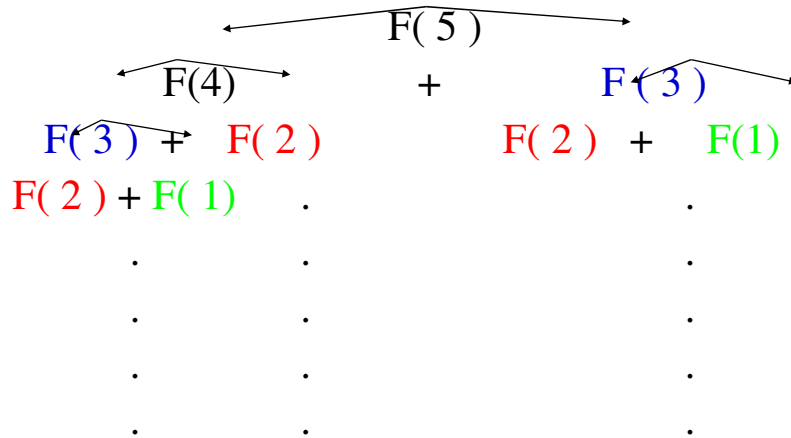
- Quando a solução ótima de um subproblema pode ser compor das soluções ótimas dos subproblemas
- Quando o calculo da solução ótima implica muitas vezes o calculo do mesmo subproblema.

## Exemplo de Fibonacci

---

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

## Exemplo de Fibonacci



## Utilizando PD no problema de Fibonacci

| F ( 0 ) | F ( 1 ) | F ( 2 ) | F ( 3 ) | F ( 4 ) |
|---------|---------|---------|---------|---------|
|---------|---------|---------|---------|---------|

```

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]; }
    
```

## O problema da multiplicação de matrizes

- Suponhamos que temos  $n$  matrizes ( $A_1, \dots, A_n$ ) e precisamos multiplicar elas. Para cada  $i$  ( $1 \leq i \leq n$ ) a matriz  $A_i$  tem dimensionalidade  $p_{i-1} \times p_i$ . Queremos encontrar a forma de multiplicar as matrizes de modo que o número de multiplicações escalares seja mínimo.
  - Obs: Para multiplicar uma matriz de  $p \times q$  dimensões, com uma matriz de  $q \times r$  dimensões,  $pqr$  operações serão necessárias

## Consideremos o seguinte exemplo

- $M = M_1[10,20] \times M_2[20,50] \times M_3[50,1] \times M_4[1,100]$
- A forma mais eficiente de multiplicá-las é:
$$M = (M_1 \times (M_2 \times M_3)) \times M_4$$
  - Requer apenas 2.200 operações
  - Outras opções podem precisar de até 125,000 operações

## O problema da multiplicação de matrizes

- Tentar todas as ordens possíveis de multiplicações para avaliar o produto de  $n$  matrizes é um processo exponencial em  $n$ ,  $f(n) \geq 2^{n-2}$
- Usando programação dinâmica é possível resolver este problema em  $O(n^3)$

## O problema da multiplicação de matrizes

- Modelando o Problema:
  - Seja  $m_{ij}$  o menor custo para computar o produto  $M_i \times M_{i+1} \times \dots \times M_j$ , para  $1 \leq i \leq j \leq n$
  - Nesse caso:

$$m_{ij} = \begin{cases} 0 & \text{se } i=j \\ \text{Min } i \leq k < j (m_{ik} + m_{k+1,j} + d_{i-1}d_kd_j) & \text{se } j > i \end{cases}$$

## O problema da multiplicação de matrizes

- O termo  $m_{ik}$  representa o custo mínimo para calcular:
  - $M' = M_i \times M_{i+1} \times \dots \times M_k$
- O termo  $m_{k+1,j}$  representa o custo mínimo para calcular:
  - $M'' = M_{k+1} \times M_{k+2} \times \dots \times M_j$
- O termo  $d_{i-1,k}d_j$  representa o custo de multiplicar  $M'[d_{i-1},d_k]$  por  $M''[d_k,d_j]$

## O problema da multiplicação de matrizes

|              |              |              |              |
|--------------|--------------|--------------|--------------|
| $m_{11} = 0$ | $m_{22} = 0$ | $m_{33} = 0$ | $m_{44} = 0$ |
| $m_{12} = ?$ | $m_{23} = ?$ | $m_{34} = ?$ |              |
| $m_{13} = ?$ | $m_{24} = ?$ |              |              |
| $m_{14} = ?$ |              |              |              |

## O problema da multiplicação de matrizes

|            |           |           |        |
|------------|-----------|-----------|--------|
| m11= 0     | m22= 0    | m33= 0    | m44= 0 |
| m12= 10000 | m23= 1000 | m34= 5000 |        |
| m13= 1200  | m24= 3000 |           |        |
| m14= 2200  |           |           |        |

## Algoritmo para o problema das matrizes

```
void avaliamultmatrizes(int d[], int n){
    int m[n][n];
    int j;
    for( int i=0; i<n; i++) m[i][i]=0;
    for( int h=1;h<n;h++){
        for( i=1;i<=n-h; i++){
            j=i + h;
            m[i-1][j-1]= 1000000000000000;
            for(int k=i;k<j;k++){
                int temp= m[i-1][k-1] + m[k][j-1] + d[i-1] * d[k] * d[j];
                if ( temp < m[i-1][j-1]) m[i-1][j-1]=temp; }
        }
    }
    Printf("%d",m[0][n-1]);
}
```

## Problema do troco de moedas

- Dado um conjunto de **n moedas**, cada uma com valor  **$c_i$** , e, dada uma quantidade **P**, precisamos achar o **numero mínimo de moedas** para obter aquela quantidade
- Definimos a função  **$D(i, Q)$**  como o número mínimo de moedas necessário para obter uma quantidade  $Q$ , usando os  $i$  primeiros tipos de moedas  $(1 \dots i)$ .
- A solução da função Troco pode utilizar 0 ou mais moedas do tipo  $i$ 
  - Se não usa nenhuma moeda do tipo  $i$  então:  $D(i, Q) = D(i-1, Q)$
  - Se usa sim,  $k$  moedas do tipo  $i$ , então:  $D(i, Q) = D(i-1, Q - K * c_i) + k$

## Exemplo para o problema do número das moedas

- $N = 3$
- $P = 8$
- $C = \{1, 4, 6\}$
- ?

## Problema das moedas

|         | Q |   |   |   |   |   |   |   |   |
|---------|---|---|---|---|---|---|---|---|---|
| Moedas  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| $C_1=1$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| $C_2=4$ | 0 | 1 | 2 | 3 | 1 | 2 | 3 | 4 | 2 |
| $C_3=6$ | 0 | 1 | 2 | 3 | 1 | 2 | 1 | 2 | 3 |

## Problema do número mínimo de moedas

- Qual é a função para achar o mínimo numero de moedas em cada etapa?
  - $D(i, Q) = \min_{k=0,1,\dots,Q/c_i} \{D(i-1, Q), D(i-1, Q-k*c_i) + k\}$

## Algoritmo do problema das moedas

□ ?

## Algoritmo do problema das moedas

```
Moedas ( int P, int N, int C[], int D[][]){
    int i,j;
    For ( i=0;i <=N; i++)
        D[i][0]=0;
    For ( i=1;i <=P; i++)
        D[0][i]=NUM_MAX;
    For (i=1; i<=N; i++){
        For(j=1;j<=P;j++){
            If (C[i] > j ) D[i][j] = D[ i-1][j];
            Else if (D[i-1][j] < (D[i][j]-C[i] +1) ) D[i][j]=D[i-1][j];
            Else D[i][j]=D[i][j]-C[i] +1;
        }
    }
}
```