Universidade Federal de Ouro Preto
Departamento de Computação

# Programação Dinâmica

decom
departamento de computação

Prof. Haroldo Gambini Santos

## Introdução

- Fibonacci, relação de recorrência

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

$$\{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots\}$$



## Útil para modelar

- Biologia, Demografia
- Arquitetura, Música
- ...



## Fibonacci, algoritmo simples:

```
função fibonacci(n)
    se n<2 então:
        retorne n;
    senão
        retorne fibonacci(n-1)+fibonacci(n-2)
```

Complexidade ????

## Fibonacci em $O(n)$

```
função fibonacci(n)
    anterior = 0; corrente = 0;
    se n<2 então:
        retorne n;
    senão
        para i de 1 até n-1 faça:
            novo = anterior + corrente;
            anterior = corrente;
            corrente = novo;
        fim
    fim
    retorne corrente;
```



## Programação Dinâmica

- Estratégia ***Bottom Up***
  - Casos menores são inicialmente calculados
  - Resultados são armazenados
  - Cálculos subseqüentes utilizam valores pré-calculados



## Programação Dinâmica

- Estratégia ***Top Down***
  - Utiliza métodos recursivos
  - Métodos recursivos são **instrumentados** para salvar e reutilizar resultados pré-calculados
  - **Memoização**



## Divisão e Conquista × Programação Dinâmica

- Divisão e Conquista
  - Eficiente quando subproblemas são independentes
  - Alguns subproblemas podem ser resolvidos muitas vezes
- Programação Dinâmica
  - Apropriado quando subproblemas compartilham subproblemas
  - Cada subproblema é resolvido somente uma vez
  - Resultados são armazenados
  - Troca-se **memória** por tempo de **processamento**

## O Problema da Mochila

- Exemplo

| item | A | B | C | D | E |
|---|---|---|---|---|---|
| peso | 3 | 4 | 7 | 8 | 9 |
| lucro | 4 | 5 | 10 | 11 | 13 |

Capacidade da Mochila: 17

Soluções ótimas (lucro 24):
DE
ACC
AABC
AAABB



## Problema da Mochila

- Entrada:
  - Items $i$={1,...,$n$}
    - peso $p_i$
    - Lucro $l_i$
- Função que especifica a solução ótima para uma mochila de tamanho $c$ em função da solução ótima de problemas menores:

$$M(p) = \max_{i=1,\ldots,n: p_i \leq c} \{ M(\ c\text{-}p_i\ ) + l_i \}$$



## Problema da Mochila – Sol. *Top Down* ineficiente

```
função mochila( c, n, p, l )
    m = 0
    para i de 1 até n faça:
        c' = c – p_i
        se c' ≥ 0
            m' = mochila( c', n, p, l ) + l_i
            se m' > m então:
                m = m'
            fim
        fim
    fim
    retorne m
```



## Problema da Mochila – Sol. *Top Down*

```
m[] = [nulo]^n

função mochila( c, n, p, l )
    se m[c] ≠ nulo
        retorne m[c]
    senão
        m[c] = 0
        para i de 1 até n faça:
            c' = c – p_i
            se c' ≥ 0
                m' = mochila( c', n, p, l ) + l_i
                se m' > m[c] então:
                    m[c] = m'
                fim
            fim
        fim
    retorne m[c]
```

## Problema da Mochila, Sol. *Bottom Up*

```
função mochila( c, n, p, l )
    m[] = [0]^c
    para k de 1 até c faça:
        m[k] = 0
        para i de 1 até n faça:
            c' = k − p_i
            se c' ≥ 0 então
                m' = m[c'] + l_i
                se m' ≥ m[k] então:
                    m[k] = m'
                fim
            fim
        fim
    fim
    retorne m[c]
```



## Problema da Mochila sem Repetições

- Consideração adicional:
  - Um dado item $i$ será incluso ou não na solução ?

$$M(\ c,\ i\ ) = \max\{\ M(\ c\ -\ p_i\ ) + l_i\ ,\ M(\ c,\ i\text{-}1\ )\ \}$$

Máximo de lucro que pode ser obtido com ou sem a inclusão do item $i$



## Maior substring comum

- Entrada:
  - $x = (x_1, x_2, \ldots, x_n)$
  - $y = (y_1, y_2, \ldots, y_m)$
- Saída:
  - O maior $k$ tal que existam $i$ e $j$ satisfazendo:
  - $(\ x_i,\ x_{i+1},\ \ldots,\ x_{i+k-1}\ ) = (y_j,\ y_{j+1}\ ,\ldots,\ y_{j+k-1})$



## Maior substring comum

- Formulação:

$$d(i,j) = \begin{cases} d(i-1, j-1) + 1 & \text{se } x_i = y_j \\ 0 & \text{caso contrário} \end{cases}$$