FCT/Unesp – Presidente Prudente
Departamento de Matemática e Computação

Projeto e Análise de Algoritmos: Programação Dinâmica

Prof. Danilo Medeiros Eler
danilo.eler@unesp.br

(apresentação adaptada: ver referências no final)

Departamento de Matemática e Computação
Faculdade de Ciências e Tecnologia
Unesp – Universidade Estadual Paulista
Presidente Prudente/SP, Brasil

Programação Dinâmica

- Tem como objetivo reduzir o tempo de execução de um programa utilizando soluções ótimas a partir de **subproblemas** previamente calculados
- O termo programação, neste caso, não está relacionado com programa de computador, mas com método de solução baseado em tabela

Programação Dinâmica

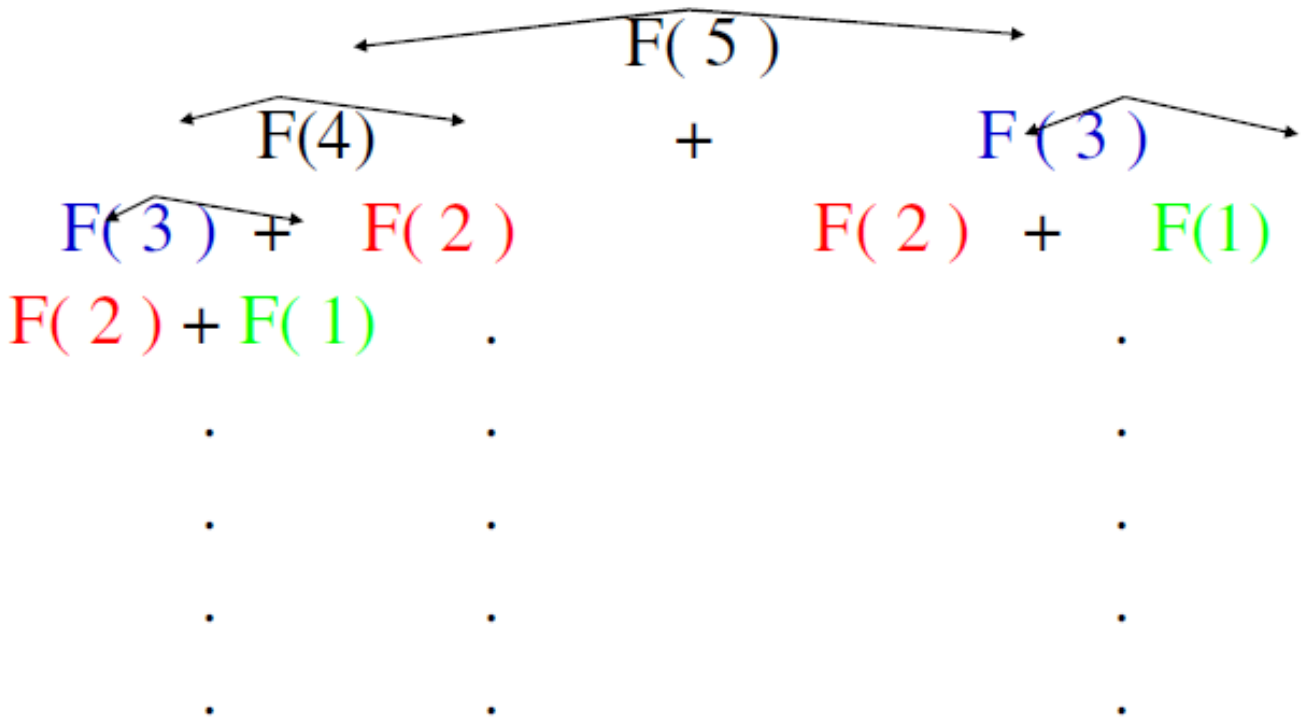
- Visa evitar o recálculo desnecessário das soluções dos subproblemas
  - Para isso, elas são armazenadas em tabelas

Exemplo de Fibonacci

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

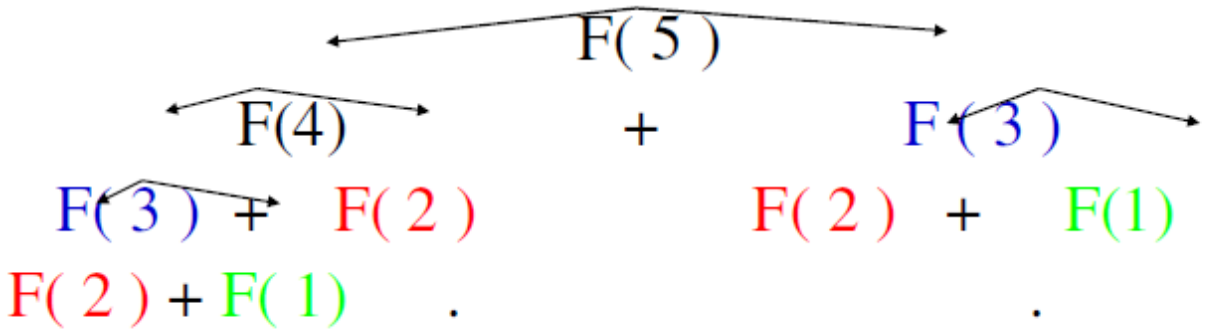
Programação Dinâmica

Exemplo de Fibonacci



Programação Dinâmica

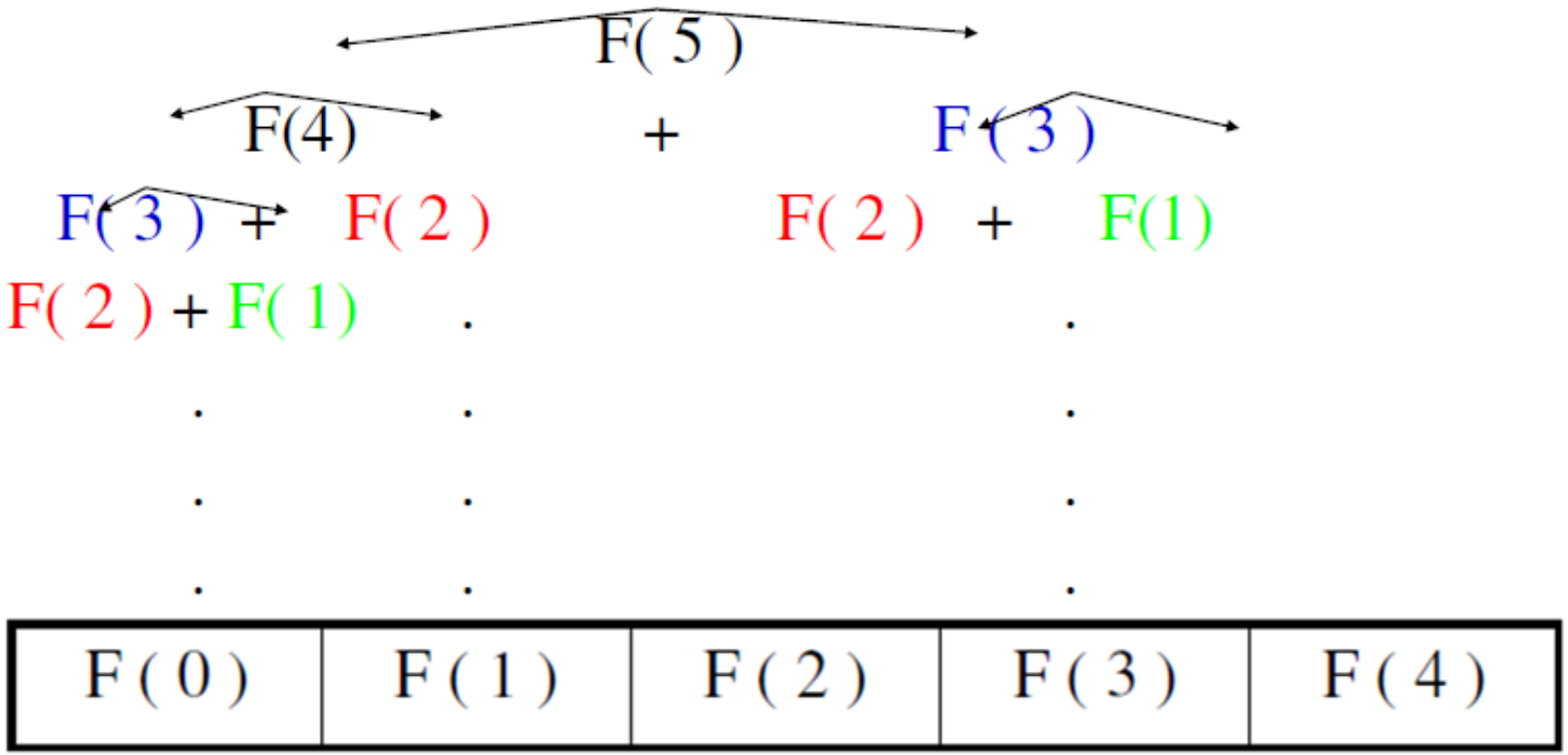
Exemplo de Fibonacci




```
int fibonacci (int n) {
    if (n < 2) {
        return n;
    } else {
        return fibonacci (n - 1) + fibonacci (n - 2);
    }
}
```

Programação Dinâmica

Exemplo de Fibonacci



Programação Dinâmica

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]; }
```

Programação Dinâmica

Um problema para ser resolvido com programação dinâmica (PD) requer apresentar as seguintes propriedades:

- **Sub-estrutura ótima:** a solução ótima é composta da solução ótima de instâncias menores (*Fibonacci(n)* é composto pela soma de *Fibonacci(n-1)* e *Fibonacci(n-2)*);
- **Sobreposição de subproblemas:** ao calcular *Fibonacci(n-1)* se calcula *Fibonacci(n-2)*, que é usado para calcular *Fibonacci(n)*.

Solução com PD para fibonacci é memorizar num vetor os valores/estados (usar espaço) para não re-calcular (ganhar em velocidade).

Programação Dinâmica

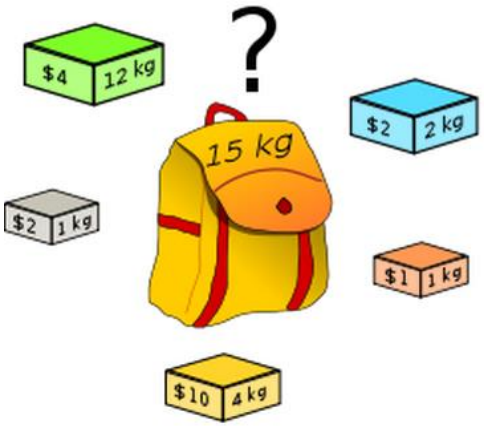
- Passos a seguir
  - Dividir o problema em sub problemas
  - Computar os valores de uma solução de forma *bottom-up e* armazená-los (memorização)
    - Sub-estrutura ótima
  - Construir a solução ótima para cada subproblema utilizando os valores computados
    - Sobreposição de subproblemas

Programação Dinâmica

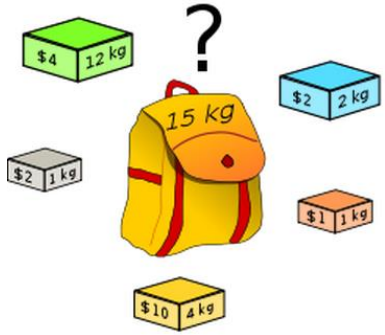
- Programação dinâmica X Divisão e conquista
  - **Divisão e conquista** particiona o problema em sub-problemas menores
    - Utiliza soluções menores para compor a solução final, dependendo do cenário
  - **Programação dinâmica** resolve os sub-problemas, partindo dos menores para os maiores
    - Armazena os resultados e somente reusa as soluções ótimas

Problema 1

- Dados uma mochila com capacidade *C* e *N* itens disponíveis, cada item tem um peso (w) e um valor (*v*), deseja-se selecionar os itens para colocar na mochila de modo que o seu valor seja maximizado, sem exceder a capacidade da mochila (C).



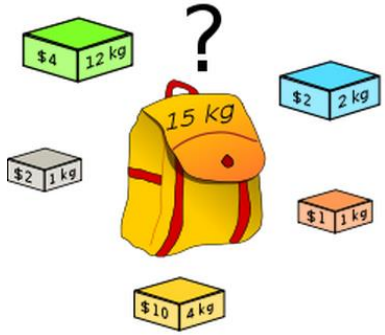
Problema 1



- Exemplo de configuração
  - C = 5;
  - N = 4
  - Itens
    - Item 1 – v = 50; w = 4;
    - Item 2 – v = 40; w = 2;
    - Item 3 – v = 30; w = 1;
    - Item 4 – v = 45; w = 3;

Mochila
Valor: 0
Peso: 0
Itens:

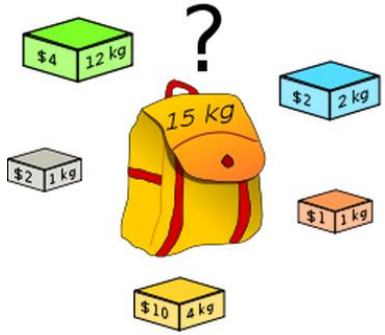
Problema 1



- Exemplo de configuração
  - C = 5;
  - N = 4
  - Itens
    - Item 1 – v = 50; w = 4;
    - Item 2 – v = 40; w = 2;
    - Item 3 – v = 30; w = 1;
    - Item 4 – v = 45; w = 3;

Mochila
Valor: 50
Peso: 4
Itens: • 1

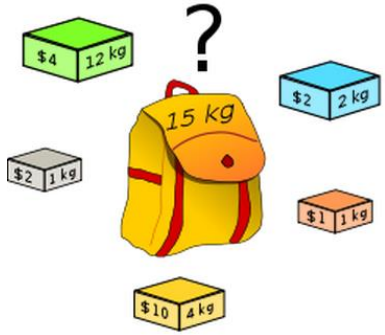
Problema 1



- Exemplo de configuração
  - C = 5;
  - N = 4
  - Itens
    - Item 1 – v = 50; w = 4;
    - Item 2 – v = 40; w = 2;
    - Item 3 – v = 30; w = 1;
    - Item 4 – v = 45; w = 3;

Mochila
Valor: 80
Peso: 5
Itens: • 1 • 3

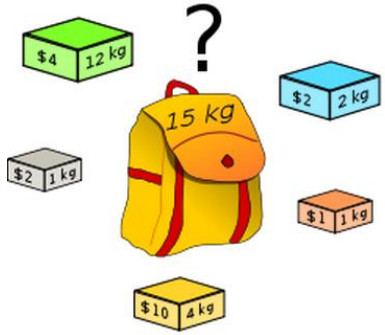
Problema 1



- Exemplo de configuração
  - C = 5;
  - N = 4
  - Itens
    - Item 1 – v = 50; w = 4;
    - Item 2 – v = 40; w = 2;
    - Item 3 – v = 30; w = 1;
    - Item 4 – v = 45; w = 3;

Mochila
Valor: 45
Peso: 3
Itens: • 4

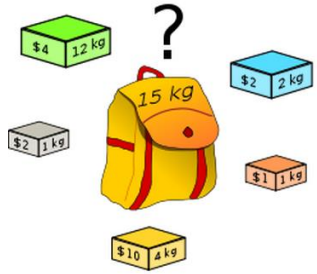
Problema 1



- Exemplo de configuração
  - C = 5;
  - N = 4
  - Itens
    - Item 1 – v = 50; w = 4;
    - Item 2 – v = 40; w = 2;
    - Item 3 – v = 30; w = 1;
    - Item 4 – v = 45; w = 3;

Mochila
Valor: 85
Peso: 5
Itens: • 4 • 2

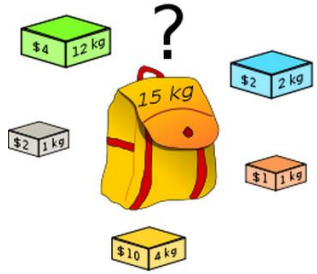
Problema 1



- Dados uma mochila com capacidade *C* e *N* itens disponíveis, cada item tem um peso (w) e um valor (*v*), deseja-se selecionar os itens para colocar na mochila de modo que o seu valor seja maximizado, sem exceder a capacidade da mochila (C).

Resolução por Combinação → $O(2^n)$

Problema 1

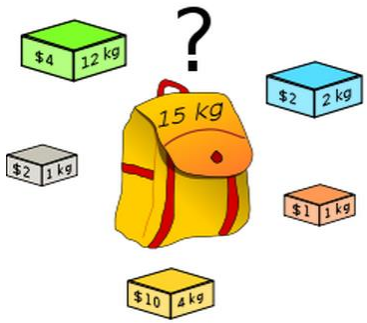


- Seja uma mochila com capacidade *C* e n itens, cada qual tem um peso (w*[ ]*) e um valor (*v[ ]*). Deseja-se maximizar o valor da mochila, selecionando ou não um item da lista
- n = 3
- v = {9, 7, 8}
- w = {4, 3, 2}
- C = 5

Combinação → $O(2^n)$

Programação Dinâmica → $O(nC)$

Problema 1



- n = 3
- v = {9, 7, 8}
- w = {4, 3, 2}
- C = 5
- Com um algoritmo guloso a solução seria (máximo local)
  - Item 1 ➔ Valor 9 e peso 4
    - Apesar de não ter ocupado toda a mochila, não cabe mais nenhum item
- A solução ótima (máximo global)
  - Item 2 → Valor 7 e peso 3
  - Item 3 → Valor 8 e peso 2
  - Utilizou toda a capacidade da mochila
    - Valor total = 15
- Com programação dinâmica o problema é resolvido em O(nC)

$$V[i,w] = \max(V[i-1,w], v_i + V[i-1, w-w_i])$$

Problema 2

- Com programação dinâmica é possível resolver o problema da Subsequência Comum Máxima (*Longest Common Subsequence*) em um tempo O(mn), em que *m* é o comprimento da sequência X e *n* é o comprimento da sequência Y

Problema 2

- Considere que X e Y sejam duas sequências de elementos. Qual é a subsequência comum máxima entre elas?
- Exemplo:
  - X = ABCBDAB
  - Y = BDCABA
- Algumas subsequências
  - BCAB – 4
  - DAB – 3
  - CAB – 3
  - AB – 2
- Força bruta tem complexidade exponencial

Problema 3

- Calcular o número mínimo de operações de multiplicação (escalar) necessário para computar a matriz *M* dada por

  M = M1 x M2 x ... X Mn

- Matrizes são multiplicadas aos pares sempre. Então, é preciso encontrar uma *parentização* (agrupamento) ótimo para a cadeia de matrizes

Problema 3

- Qual é o mínimo de multiplicações escalares necessário para computar *M*

  $M = M1_{200,2} \times M2_{2,30} \times M3_{30,20} \times M4_{20,5}$

- A ordem com que as matrizes são multiplicadas pode influenciar no desempenho do algoritmo?

Problema 3

- Qual é o mínimo de multiplicações escalares necessário para computar *M*

  $M = M1_{200,2} \times M2_{2,30} \times M3_{30,20} \times M4_{20,5}$

- Possibilidades de *parentização*

$$M = (M_1 \times (M_2 \times (M_3 \times M_4)))$$
$$M = (M_1 \times ((M_2 \times M_3) \times M_4))$$
$$M = ((M_1 \times M_2) \times (M_3 \times M_4))$$
$$M = ((M_1 \times (M_2 \times M_3)) \times M_4)$$
$$M = (((M_1 \times M_2) \times M_3) \times M_4)$$

Problema 3

- Qual é o mínimo de multiplicações escalares necessário para computar *M*

  $M = M1_{200,2} \times M2_{2,30} \times M3_{30,20} \times M4_{20,5}$

- Possibilidades de *parentização*

$$M = (M_1 \times (M_2 \times (M_3 \times M_4))) \rightarrow \text{5.300 multiplicações}$$
$$M = (M_1 \times ((M_2 \times M_3) \times M_4)) \rightarrow \text{3.400 multiplicações}$$
$$M = ((M_1 \times M_2) \times (M_3 \times M_4)) \rightarrow \text{45.000 multiplicações}$$
$$M = ((M_1 \times (M_2 \times M_3)) \times M_4) \rightarrow \text{29.200 multiplicações}$$
$$M = (((M_1 \times M_2) \times M_3) \times M_4) \rightarrow \text{152.000 multiplicações}$$

- A ordem das multiplicações faz muita diferença

Problema 3

- Calcular o número mínimo de operações de multiplicação (escalar) necessário para computar a matriz *M* dada por

  M = M1 x M2 x ... X Mn

- Matrizes são multiplicadas aos pares sempre. Então, é preciso encontrar uma *parentização* (agrupamento) ótimo para a cadeia de matrizes
- Com programação dinâmica é possível resolver em $O(n^2)$ operação, onde n é o número de matrizes

Extraído de

- Análise de Algoritmos I
  - Cid Carvalho de Souza, Cândida Nunes da Silva e Orlando Lee (IC/Unicamp)
- Projeto de Algoritmos: paradigmas
  - Moacir Ponti Junior (ICMC/USP)
- Programação Dinâmica I
  - Lucas Schmidt Cavalcante e Rosane Minghim (ICMC/USP)

Bibliografia

LEVITIN, A. **Introduction to The Design & Analysis of Algorithms**, Addison Wesley, 2003, 497p

BIBLIOGRAFIA BÁSICA

1. CORMEN, T. H.; LEISERSON C. E.; RIVEST, R. L.; STEIN, C. **Algoritmos**: teoria e prática, 2ª ed. Rio de Janeiro: Editora Campus, 2002. 916p.
2. SEDGEWICK, R. **Algorithms in C**: graph algorithms, 3ª ed. Part 5, Addison Wesley, 2002. 482p.
3. NETTO, P. O. B. **Grafos**: teoria, modelos, algoritmos. 2ª ed. São Paulo: Editora Edgard Blücher Ltda, 2001. 304p.
4. SZWARCFITER, J. L. **Grafos e Algoritmos Computacionais**. Rio de Janeiro: Editora Campus, 1984. 216p.
5. ZIVIANI, N. **Projeto de algoritmos**: com implementações em PASCAL e C. 2ª ed. São Paulo: Pioneira Thomson Learning Ltda, 2004. 552p.

BIBLIOGRAFIA COMPLEMENTAR

1. DROZDEK, A. **Estrutura de Dados e Algoritmos em C++**. São Paulo: Pioneira Thomson Learning Ltda, 2002. 579p.
2. GOODRICH, M. T.; TAMASSIA, R. **Estruturas de Dados e Algoritmos em JAVA**. 2ª ed. Bookman Companhia Editora, 2002. 584p.
3. SCHEINERMAN, E. R. **Matemática discreta**: uma introdução. São Paulo: Pioneira Thomson Learning, 2003. 532 p.
4. WILSON, R. J.; WATKINS, J. J. **Graphs**: an introductory approach. New York: John Wiley & Sons, Inc, 1990. 340p.