

# Técnicas de Projeto (Parte 4)

## Projeto e Análise de Algoritmo

Adaptado: Felipe Cunha

# Técnicas de Projeto

## 1) Algoritmos Gulosos

# Algoritmos Gulosos

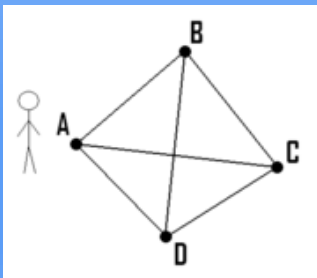
- Tipicamente algoritmos gulosos são utilizados para resolver problemas de otimização.
- Uma característica comum dos problemas onde se aplicam algoritmos gulosos é a existência de subestrutura ótima, semelhante à programação dinâmica:
  - Programação dinâmica: tipicamente os subproblemas são resolvidos quanto à otimalidade antes de se proceder a escolha de um elemento que irá compor a solução ótima
  - Algoritmo guloso: primeiramente é feita a escolha de um elemento que irá compor a solução ótima e só depois um subproblema é resolvido.

# Algoritmos Gulosos

- Um algoritmo guloso sempre faz a escolha que parece ser a melhor a cada iteração, ou seja, de acordo com um critério guloso. É uma decisão localmente ótima.
- Propriedade da escolha gulosa: garante que a cada iteração é tomada uma decisão que irá levar a um ótimo global.
- Em um algoritmo guloso uma escolha que foi feita nunca é revista, ou seja, não há qualquer tipo de retrocesso.

# Exemplo: Problema do Caixeiro Viajante

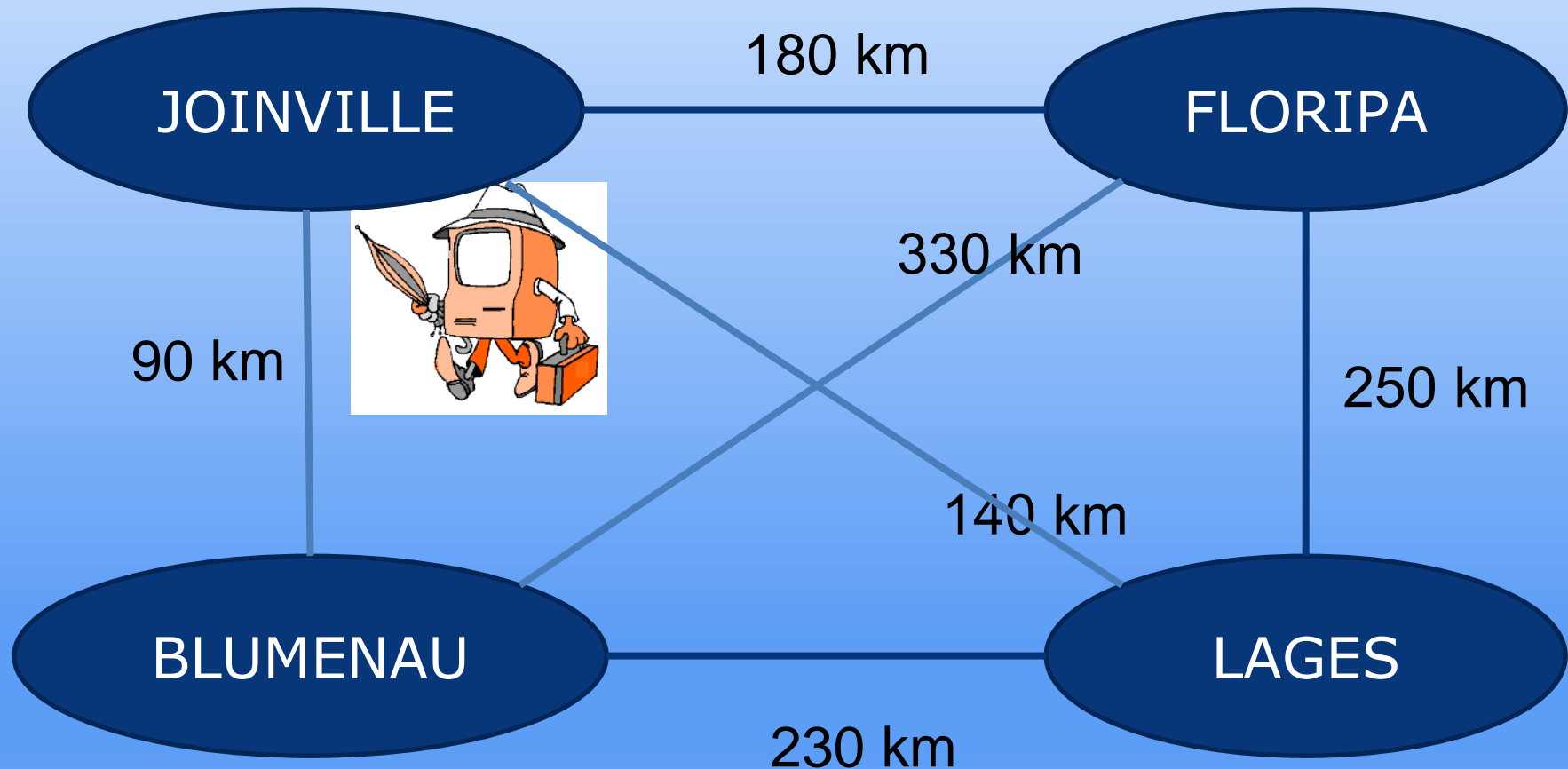
- Um caixeiro viajante deseja visitar  $N$  cidades e entre cada par de cidades existe uma rota;
- Cada rota possui uma distância (ou o custo necessário) para percorrê-la;
- O caixeiro viajante deseja encontrar um caminho que passe por cada cidade apenas uma vez, e além disso que tenha um custo menor que certo valor.



## Traveling Salesman Problem - TSP



# Exemplo: Problema do Caixeiro Viajante



## Exemplo: Problema do Caixeiro Viajante

N	Rotas por Segundo	( n - 1 )!	Cálculo Total
5	250 milhões	24	Insignificante
10	110 milhões	362 880	0.003 seg
15	71 milhões	87 bilhoes	20 min
20	53 milhões	$1.2 \times 10^{17}$	73 anos
25	42 milhões	$6.2 \times 10^{23}$	470 milhões de anos

# Exemplo: Problema do Caixeiro Viajante

## Vizinho mais próximo (Abordagem Gulosa)

- 1 - Selecione arbitrariamente uma cidade inicial
- 2 - Selecione a menor rota até qualquer cidade. Repita até todas as cidades terem sido visitadas.

- **PROBLEMA:** se a distância da última cidade até a primeira foi muito grande, a solução é obrigada a escolher esse caminho.



# Características de Algoritmos Gulosos

- Quando funciona corretamente, a primeira solução encontrada é sempre ótima.
- Se o objetivo é:
  - Maximizar: provavelmente escolherá o candidato restante que proporcione o maior ganho individual.
  - Minimizar: então será escolhido o candidato restante de menor custo.
- O algoritmo nunca muda de ideia:
  - Um candidato escolhido e adicionado à solução passa a fazer parte dessa solução permanentemente.
  - Um candidato excluído do conjunto solução, não é mais reconsiderado.

# Características de Algoritmos Gulosos

- Para construir a solução ótima existe um conjunto ou lista de candidatos.
- São acumulados um conjunto de candidatos considerados e escolhidos, e o outro de candidatos considerados e rejeitados.
- Existe uma função verifica se um conjunto de candidatos é viável
- Uma *função de seleção* indica a qualquer momento quais dos candidatos restantes é o mais promissor.
- Uma *função objetivo* fornece o valor da solução encontrada, como o comprimento do caminho construído (não aparece de forma explícita no algoritmo guloso).

# Prog. Dinâmica X Algoritmos Gulosos

- Possuem sub-estrutura ótima.
- Programação dinâmica:
  - Faz uma escolha a cada passo.
  - Escolha depende das soluções dos sub-problemas.
  - Resolve os problemas bottom-up.
- Técnica gulosa:
  - Trabalha na forma top-down.

# Exemplo: Problema da Mochila

- Problema da Mochila (enunciado):
  - Um ladrão acha  $n$  itens numa loja.
  - Item  $i$  vale  $v_i$  unidades (dinheiro, e.g., R\$, US\$, etc).
  - Item  $i$  pesa  $w_i$  unidades (kg, etc).
  - $v_i$  e  $w_i$  são inteiros.
  - Conseguir carregar  $W$  unidades no máximo.
  - Deseja carregar a “carga” mais valiosa.

# Exemplo: Problema da Mochila

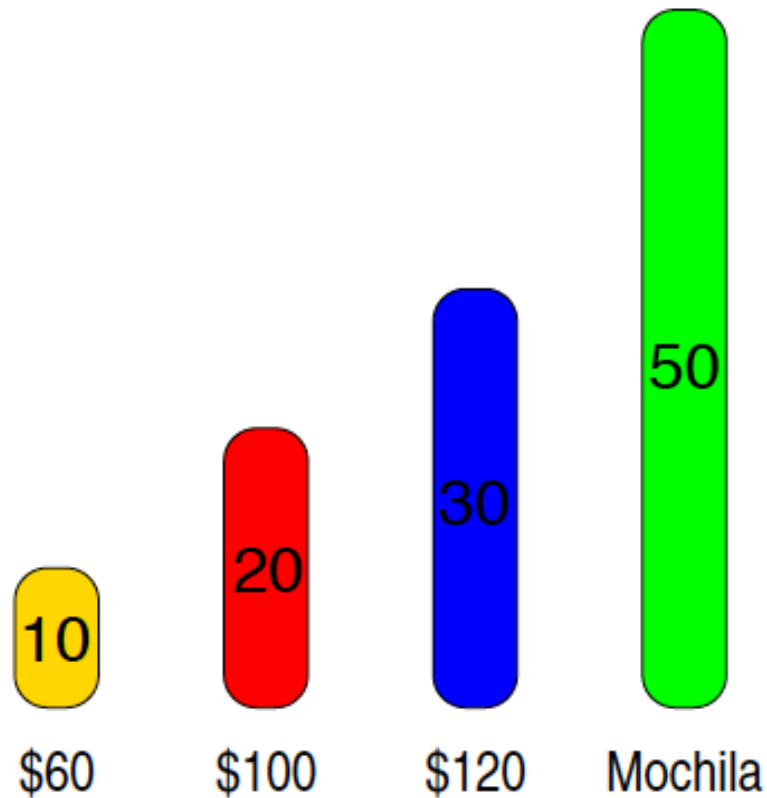
- Problema da Mochila 0 –1 ou (0 –1 Knapsack Problem):
  - O item  $i$  é levado integralmente ou é deixado.
- Problema da Mochila Fracionário:
  - Fração do item  $i$  pode ser levada.

# Exemplo: Problema da Mochila

- Possuem a propriedade de sub-estrutura ótima.
- Problema inteiro:
  - Considere uma carga que pesa no máximo  $W$  com  $n$  itens.
  - Remova o item  $j$  da carga (específico mas genérico).
  - Carga restante deve ser a mais valiosa pesando no máximo  $W - w_j$  com  $n - 1$  itens.
- Problema fracionário:
  - Considere uma carga que pesa no máximo  $W$  com  $n$  itens.
  - Remova um peso  $w$  do item  $j$  da carga (específico mas genérico).
  - Carga restante deve ser a mais valiosa pesando no máximo  $W - w$  com  $n - 1$  itens mais o peso  $w_j - w$  do item  $j$ .

# Exemplo: Problema da Mochila

- Situação Inicial

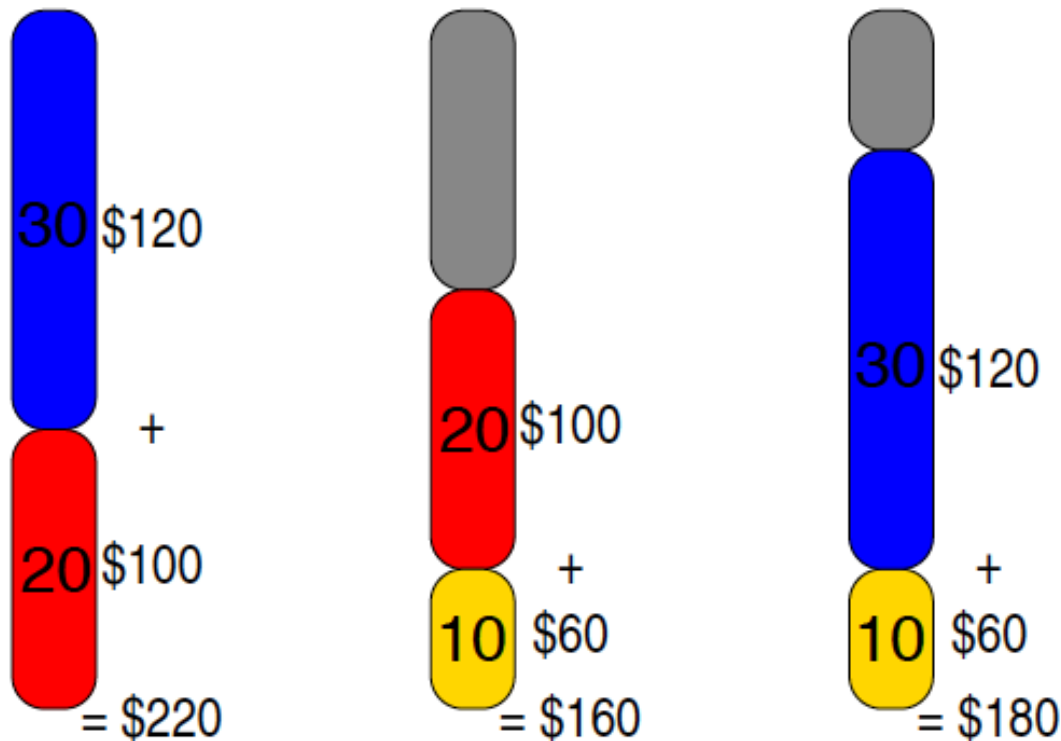


Item	Peso	Valor	V/P
1	10	60	6
2	20	100	5
3	30	120	4

Carga máxima da mochila: 50

# Exemplo: Problema da Mochila

- Estratégia Gulosa



Soluções possíveis:

#	Item (Valor)
1	2 + 3 = 100 + 120 = 220
2	1 + 2 = 60 + 100 = 160
3	1 + 3 = 60 + 120 = 180

→ Solução 2 é a gulosa.



## Exemplo: Problema da Mochila

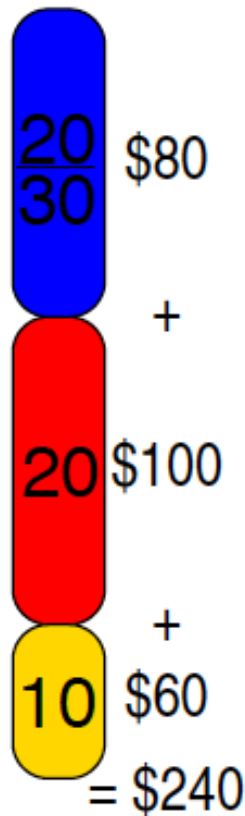
- Levar o item 1 faz com que a mochila fique com espaço vazio
- Espaço vazio diminui o valor efetivo da relação  $v/w$
- Neste caso deve-se comparar a solução do sub-problema quando:

Item é incluído na solução X Item é excluído da solução

- Passam a existir vários sub-problemas
- Programação dinâmica passa a ser a técnica adequada

# Exemplo: Problema da Mochila

- Estratégia Gulosa Problema Fracionário



Item	Peso	Valor	Fração
1	10	60	1
2	20	100	1
3	30	80	2/3

→ Total = 240.

→ Solução ótima!

# Exercícios

- Descreva como o algoritmo de PRIM utiliza a abordagem gulosa para construir a árvore geradora mínima de um grafo simples qualquer
- Projete uma abordagem gulosa para o problema do troco de moedas

## Exemplo: Seleção de Atividades

- $S = \{a_1 \dots a_n\}$ : conjunto de  $n$  atividades que podem ser executadas em um mesmo local. Exemplo: palestras em um auditório.
- Para todo  $i = 1 \dots n$ , a atividade  $a_i$  começa no instante  $s_i$  e termina no instante  $f_i$ , com  $0 \leq s_i < f_i$ . Ou seja, supõe-se que a atividade  $a_i$  será executada no intervalo de tempo (semi-aberto)  $[s_i ; f_i)$ .
- Definição: As atividades  $a_i$  e  $a_j$  são ditas compatíveis se os intervalos  $[s_i ; f_i)$  e  $[s_j ; f_j)$  são disjuntos.

## Exemplo: Seleção de Atividades

I	1	2	3	4	5	6	7	8	9	10	11
$S_i$	1	3	0	5	3	5	6	8	8	2	12
$F_i$	4	5	6	7	8	9	10	11	12	13	14

- Pares de atividades incompatíveis:  $(a_1; a_2)$ ,  $(a_1; a_3)$
- Pares de atividades compatíveis:  $(a_1; a_4)$ ,  $(a_4; a_8)$
- Conjuntos máximos de atividades compatíveis:  $(a_1; a_4; a_8; a_{11})$  e  $(a_2; a_4; a_9; a_{11})$
- As atividades estão ordenadas em ordem crescente de tempos de término.

## Exemplo: Seleção de Atividades

- Inicialmente verificaremos que o problema da seleção de atividades tem a propriedade da sub-estrutura ótima e, então, definiremos recursivamente o valor de uma solução ótima.
- Em seguida, mostraremos que há uma forma de resolver uma quantidade consideravelmente menor de subproblemas do que é feito na programação dinâmica.
- Isto será garantido por uma propriedade de escolha gulosa, a qual dará origem a um algoritmo guloso.

## Exemplo: Seleção de Atividades

- Definição:  $S_{ij} = \{a_k \in S : f_i \leq s_k < f_k \leq s_j\}$ , i.e., o conjunto de tarefas que começam depois do término de  $a_i$  e terminam antes do início de  $a_j$ .
- Tem-se que  $S = S_{0; n+1}$  e, com isso,  $S_{ij}$  está bem definido para qualquer par  $(i; j)$  tal que  $0 \leq i; j \leq n + 1$ .
- Supondo que  $f_0 \leq f_1 \leq f_2 \leq \dots f_n < f_{n+1}$ , ou seja, que as tarefas estão ordenadas em ordem crescente de tempos de término, pode-se concluir que  $S_{ij} = \emptyset$  para todo  $i \geq j$ .

# Exemplo: Seleção de Atividades

## Subestrutura ótima:

- Considere o subproblema da seleção de atividades definido sobre  $S_{ij}$ .
- Suponha que  $a_k$  pertence a uma solução ótima de  $S_{ij}$ .
- Como  $f_i \leq s_k < f_k \leq s_j$ , uma solução ótima para  $S_{ij}$  que contenha  $a_k$  será composta pelas atividades de uma solução ótima de  $S_{ik}$ , pelas atividades de uma solução ótima de  $S_{kj}$  e por  $a_k$ .



## Exemplo: Seleção de Atividades

- Definição: para todo  $0 \leq i, j \leq n + 1$ , seja  $c[i, j]$  o valor ótimo do problema de seleção de atividades para a instância  $S_{ij}$ . Deste modo, o valor ótimo do problema de seleção de atividades para instância  $S = S_{0; n+1}$  é  $c[0, n + 1]$ .
- Fórmula de recorrência:

$$c[i, j] = \begin{cases} 0 & S_{ij} = \phi \\ \max_{i < k < j: a_k \in S_{ij}} \{c[i, k] + c[k, j] + 1\} & S_{ij} \neq \phi \end{cases}$$

# Exemplo: Seleção de Atividades

- Teorema: (escolha gulosa)
- Considere o subproblema definido para uma instância não-vazia  $S_{ij}$ , e seja  $a_m$  a atividade de  $S_{ij}$  com o menor tempo de término, i.e.:

$$f_m = \min \{f_k : a_k \in S_{ij}\}$$

- Então:
  - existe uma solução ótima para  $S_{ij}$  que contém  $a_m$  e
  - $S_{im}$  é vazio e o subproblema definido para esta instância é trivial, portanto, a escolha de  $a_m$  deixa apenas um dos subproblemas com solução possivelmente não-trivial, já que  $S_{mj}$  pode não ser vazio.

# Exemplo: Seleção de Atividades

**SelecionaAtivGulosoRec**( $s$ ;  $f$  ;  $i$  ;  $j$ )

**Entrada:** vetores  $s$  e  $f$  com instantes de início e término das atividades  $i$  ,  $i + 1, \dots, j$ , sendo  $f_i \leq \dots \leq f_j$ .

**Saída:** conjunto de tamanho máximo de índices de atividades mutuamente compatíveis.

1.  $m = i + 1$ ;

// Busca atividade com menor tempo de término que pode estar em  $S_{ij}$

2. enquanto  $m < j$  e  $s_m < f_i$  faça  $m = m + 1$ ;

3. se  $m \geq j$  então retorne conjunto-vazio;

4. senão

5.     se  $f_m > s_j$  então retorne conjunto-vazio;     //  $a_m \notin S_{ij}$

6.     senão retorne  $\{a_m\} \cup \text{SelecionaAtivGulosoRec}(s, f, m, j)$ .

## Exemplo: Seleção de Atividades

- A chamada inicial será

`SelecionaAtivGulosoRec(s, f, 0, n+1) .`

- Complexidade:  $\Theta(n)$ .
- Ao longo de todas as chamadas recursivas, cada atividade é examinada exatamente uma vez no laço da linha 2. Em particular, a atividade  $a_k$  é examinada na última chamada com  $i < k$ .

# Exercícios

- Encontre uma colocação ótima de parênteses de um produto de cadeias de matrizes cuja sequência de dimensões é  $\langle 5, 10, 3, 12, 5, 50, 6 \rangle$
- Resolva o problema da mochila fracionária considerando uma mochila com capacidade 50 e 4 itens conforme peso e valor especificados na tabela abaixo

	A	B	C	D	E
Peso	40	30	20	10	20
Valor	840	600	400	100	300

# Exercícios

- Implemente um programa que resolva o problema do Ciclo Hamiltoniano, usando backtracking. Compare o desempenho do programa que usa a técnica de força bruta
- Enumere as diferenças e semelhanças entre a técnica de Divisão e Conquista e Programação Dinâmica
- Mostre que o tempo de execução de Quicksort é  $\Theta(n^2)$  quando todos os elementos do vetor A são distintos e estão ordenados em ordem decrescente