Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

SIN 343
Desafios de Programação

João Batista Ribeiro

joao42lbatista@gmail.com

Slides baseados no material do prof. Guilherme C. Pena

Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

Aula de Hoje

Projeto de Algoritmos

Algoritmos Gulosos

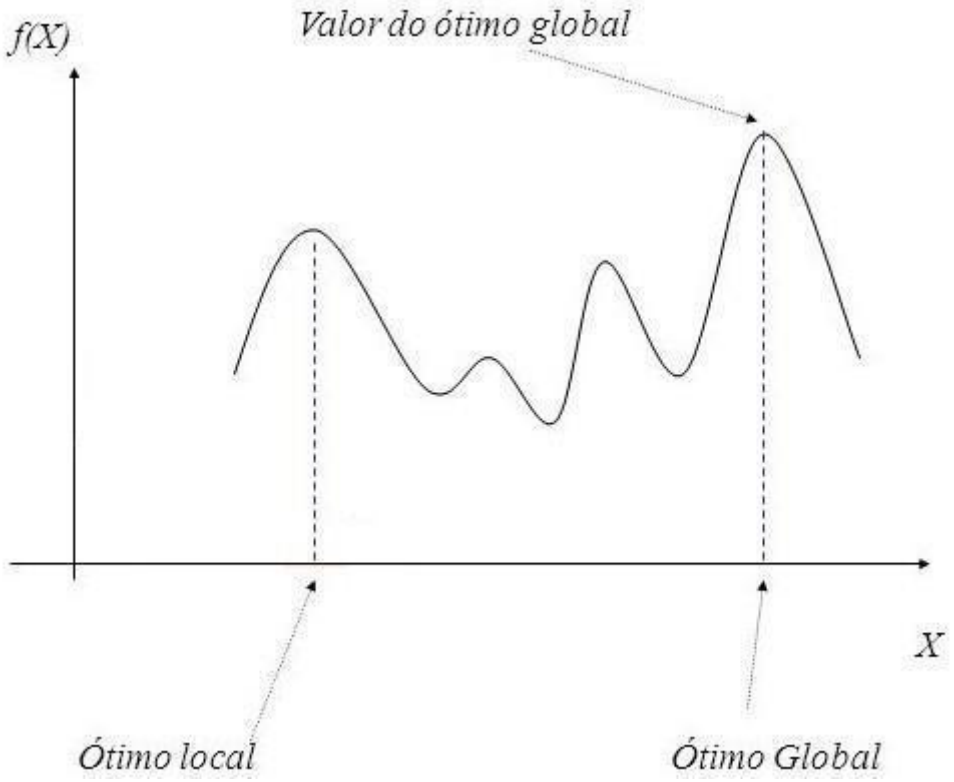
O **método guloso** é uma estratégia normalmente usada para resolver problemas de otimização.

Ele consiste em construir uma solução de forma incremental, expandindo a sub(solução) já obtida até o momento, pela inclusão de um o mais elementos.

Algoritmos Gulosos

Nesta construção, o algoritmo guloso **sempre** faz a escolha que parece ser a melhor no momento em questão.

Isto é, faz uma escolha "***localmente ótima***", na esperança de que essa escolha leve a uma solução **globalmente ótima**.



Algoritmos Gulosos

Assim, em cada passo, a escolha deve ser:

- **Viável**: a (sub)solução deve satisfazer os requisitos do problema
- **Localmente ótima**: deve ser a escolha que leve à melhor (sub)solução viável naquele momento
- **Irreversível**: uma vez feita a escolha, ela não pode ser desfeita num passo posterior

Algoritmos Gulosos

Os requisitos explicam o nome da técnica: em cada etapa, é feita a escolha "***gulosa***" da melhor alternativa disponível naquele momento.

Em alguns problemas, essa estratégia **produz uma solução ótima** para todas as instâncias.

Para outros, ela não leva a uma solução ótima, mas pode ser útil para gerar, de forma rápida, **soluções aproximadas**.

Algoritmos Gulosos

Problemas onde é possível obter soluções ótimas:

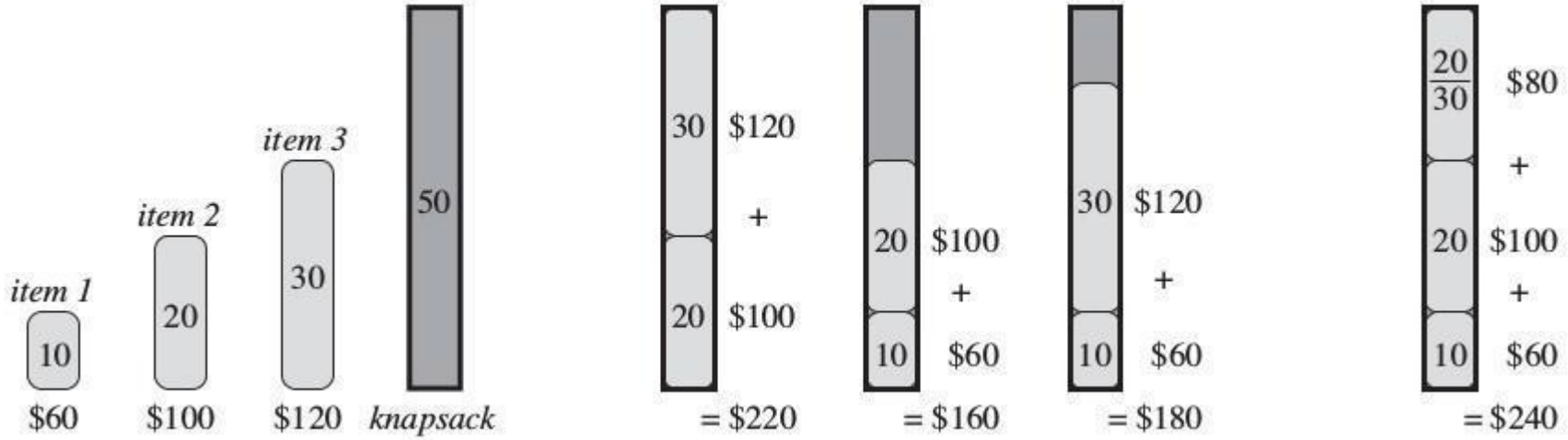
- Árvore geradora mínima (MST);
- Caminhos mais curtos;
- Escalonamento de tarefas;
- Códigos de Huffman.

Problemas onde só é possível obter aproximações:

- Problema do caixeiro viajante (TSP);
- Problema (geral) da mochila;
- Vários outros problemas de otimização combinatória.

Algoritmos Gulosos

Problema da Mochila (Fracionário):
Dado n objetos, cada um com o peso w_i (um número real positivo), um custo c_i e dada uma mochila que suporta um peso W (um número real positivo). O objetivo é determinar como preencher a mochila, **permitindo que se leve porções dos objetos**, tal que o custo final seja máximo.



Algoritmos Gulosos

Problema da Mochila (Fracionário):

Exemplo: Considere a instância do problema onde a mochila tem capacidade W = 20 e há n = 3 objetos com custos: $(c_1 , c_2 , c_3) = (25, 24, 15)$ e pesos: $(w_1, w_2, w_3) = (18, 15, 10)$. Assim, o objetivo é

$$\max (25x_1 + 24x_2 + 15x_3)$$
$$\text{s.a. } 18x_1 + 15x_2 + 10x_3 <= 20$$

Algoritmos Gulosos

Problema da Mochila (Fracionário):

$$\max (25x_1 + 24x_2 + 15x_3)$$

$$\text{s.a. } 18x_1 + 15x_2 + 10x_3 <= 20$$

(x_1, x_2, x_3)	Peso total	Custo Total
(1/2, 1/3, 1/4)	16,5	24,25
(1, 2/15, 0)	20	28,2
(0, 2/3, 1)	20	31
(0, 1, 1/2)	20	31,5
(1/2, 1/2, 1/2)	21,5 (inviável)	32

Algoritmos Gulosos

Problema da Mochila (Fracionário):

Estratégia 1:
A cada passo, escolher o objeto de maior valor.
Ordem de preenchimento: objeto 1, objeto 2, objeto 3
(1, 2/15, 0), custo total = 28,2 (não é a melhor solução)

Estratégia 2:
A cada passo, escolher o objeto de menor peso.
Ordem de preenchimento: objeto 3, objeto 2, objeto 1
(0, 2/3, 1), custo total = 31 (não é a melhor solução)

Algoritmos Gulosos

Problema da Mochila (Fracionário):

Estratégia 3:
A cada passo, escolher o objeto de maior valor por unidade de peso, isto é, com maior relação c_i/w_i .

Como c_i/w_i = (1.39, 1.6, 1.5) então

Ordem de preenchimento: objeto 2, objeto 3, objeto 1
(0, 1, 1/2) , custo total = 31,5.

A estratégia 3 conduzirá a uma *solução ótima* do problema.

Algoritmos Gulosos

Problema da Mochila (Fracionário):

Estratégia 3: Complexidade **Θ(n log n)**

```
Mochila (n, C[], w[], W ){
   Ordene os objetos em ordem decrescente de c[i]/w[i];
   Faça X = (0, 0,...,0);
   i = 0;
   while ( i < n && w[i] < W ){
     X[i] = 1;
     W = W – w[i];
     i = i + 1;
   }
   if ( i < n )
       X[i] = W / w[i];
   return X;
}
```

Ver código: **mochilaFrac.cpp**

Algoritmos Gulosos

Problema da Seleção de Atividades:

Seja A um conjunto com n atividades que necessitam usar um mesmo recurso, que não pode ser usado simultaneamente por mais de uma tarefa (uma sala de reuniões, um computador de pesquisa, ...).

Suponha que cada tarefa i inicia-se num instante s_i e encerra-se no instante imediatamente anterior a t_i . Mais precisamente, uma tarefa i deve ser executada no intervalo de tempo $[s_i, t_i)$.

Algoritmos Gulosos

Problema da Seleção de Atividades:

Dizemos que duas atividades i e j são compatíveis se os intervalos $[s_i, t_i)$ e $[s_j, t_j)$ não se sobrepõem.

O problema de seleção de atividades consiste em **selecionar o subconjunto máximo de atividades compatíveis**.

Algoritmos Gulosos

Problema da Seleção de Atividades:

Tarefa	a	b	c	d	e	f	g	h	o	j	k
Início	1	3	0	5	3	5	6	8	8	2	12
Término	4	5	6	7	9	9	10	11	12	14	16

Estratégia 1: A cada passo, escolher a tarefa que começa mais cedo. S = {c, g, k}

Estratégia 2: A cada passo, escolher a tarefa de menor duração. S = {b, h, k}

Estratégia 3: A cada passo, escolher a tarefa que termina mais cedo. S = {a, d, h, k}

Algoritmos Gulosos

Problema da Seleção de Atividades:

Algoritmo Estratégia 3: Complexidade **Θ(n log n)**

Entrada: um vetor A contendo as atividades. Em cada posição do vetor há um (s_i , t_i).

Saida: vetor S contendo as atividades a serem atendidas.

```
Atividades (A[], n){
  Ordene em ordem crescente de A[].termino;
  Faça X = (0, 0,...,0);
  i = 0; k = 0;
  X[i++] = 1; //Atividade 1
  for (j = 1; j < n; j++)
    if(A[j].s >= A[k].t){
      X[i++] = j+1;
      k = j;
    }
  return X;
}
```

Ver código:
atividades.cpp

Algoritmos C++

Biblioteca <algorithm>

http://www.cplusplus.com/reference/algorithm/

A linguagem C++ fornece uma série de algoritmos mais comuns para variados proprósitos em um intervalo de elementos:

- Busca
- Ordenação
- Contagem
- Manipulação
- dentre outros.

Algoritmos C++

Biblioteca <algorithm>

http://www.cplusplus.com/reference/algorithm/

Para a maioria deles, o site apresenta um esboço do seu funcionamento (implementação) além da sua complexidade.

Algoritmos C++

Biblioteca <algorithm>

Algumas operações que não alteram a sequência:

Find Θ(n)

Search Θ(n²)

Count Θ(n)

Ver arquivo: **naoModificam.cp**

Algoritmos C++

Biblioteca <algorithm>

Algumas operações que alteram a sequência:

Replace Θ(n)

Remove Θ(n)

Rotate Θ(n)

Ver arquivo: **modificam.cpp**

Algoritmos C++

Biblioteca <algorithm>

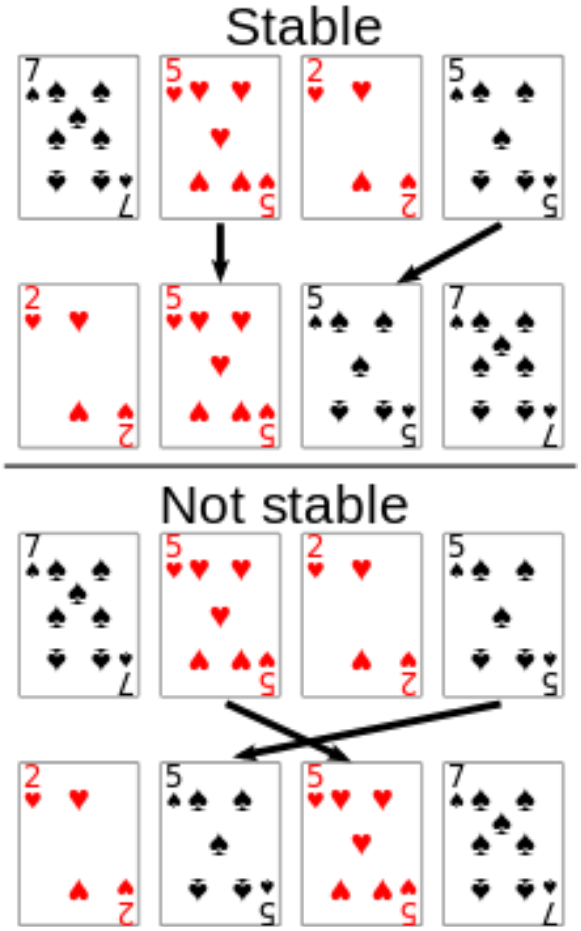
Ordenação e Busca Binária

Sort Θ(n log n)

Stable_sort Θ(n log n)

Binary_search Θ(log n)

Ver arquivo: **ordenacao.cpp**



Algoritmos C++

Exemplo:

Maria não tem problema em arranjar um parceiro para dançar. Aliás, seu maior problema é na verdade selecionar os melhores pretendentes. Ela sabe que um programa simplificaria muito esta seleção.

Ela gosta muito de dançar, e sabe que seu parceiro ideal teria 180 cm de altura. Seu primeiro critério é achar alguém o mais próximo dessa altura. Não importa se a diferença é pra mais ou pra menos.

Algoritmos C++

Exemplo:

Entre aqueles de mesma altura, ela prefere o mais próximo de 75 Kg, sem passar disso. Entre todos os acima desse peso, ela prefere o mais leve.

Se duas ou mais pessoas empatarem nessas características, ela quer seus nomes em ordem alfabética de sobrenome, e se houver mesmo sobrenome, em ordem alfabética de nome.

Ela quer apenas os nomes dos pretendentes, ordenados de acordo com os critérios estabelecidos.

Algoritmos C++

Exemplo:

Entrada:

George Bush	195	110
Harry Truman	180	75
Bill Clinton	180	75
John Kennedy	180	65
Ronald Reagan	165	110
Richard Nixon	170	70
Jimmy Carter	180	77

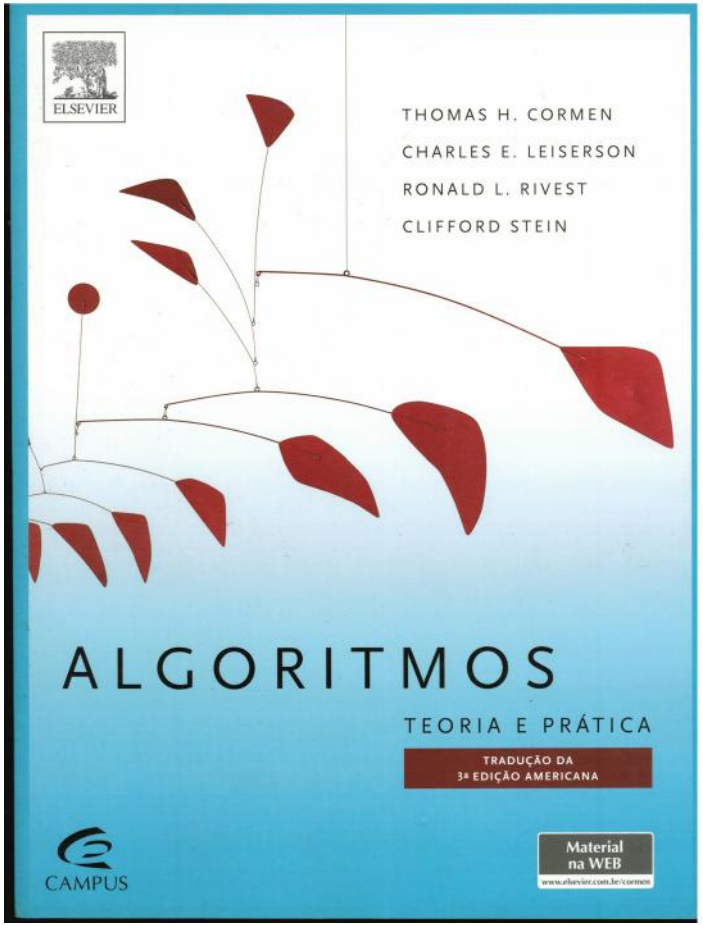
Saída Esperada:

Clinton, Bill
Truman, Harry
Kennedy, John
Carter, Jimmy
Nixon, Richard
Bush, George
Reagan, Ronald

Ver: **pretendentes.cpp**

Referência Bibliográfica

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: teoria e prática. Tradução da 2. ed. Americana. Rio de Janeiro: Campus, 2002.



Referência Bibliográfica

- SKIENA, S.S. REVILLA, M. A. Programming challenges: the programming contest training manual. Springer, 2003.