Projeto e Análise de Algoritmos

Apresentação da Disciplina

Edirlei Soares de Lima
<edirlei@iprj.uerj.br>

Por que Estudar Algoritmos?

- **Razões Práticas e Teóricas:**
  - Devemos conhecer um conjunto de algoritmos de diferentes áreas.
  - Devemos ser capazes de projetar novos algoritmos e analisar suas capacidades.
  - O estudo de algoritmos é reconhecidamente a pedra fundamental da computação.
  - Programas de computadores não existiriam sem algoritmos.

Objetivos da Disciplina

- Examinar a teoria e a arte de resolver **eficientemente** problemas computacionais.
  - Introduzir conceitos mais avançados de desenvolvimento de algoritmos;
  - Ensinar noções de complexidade em problemas computacionais;
  - Avaliar a eficiência computacional de algoritmos;
  - Comparar diferentes algoritmos para a solução de um mesmo problema;
  - Descrever e empregar os princípios, métodos e técnicas fundamentais para o projeto de algoritmos corretos e eficientes.

Exemplo: Busca em Vetor

- Problema:

Entrada: vetor v com n elementos e um elemento d a procurar

Saída: m se o elemento procurado está em v[m] ou -1 se o elemento procurado não está no vetor;

Exemplo: Busca em Vetor

- **Um primeiro algoritmo:** Percorrer o vetor `vet`, elemento a elemento, verificando se `elem` é igual a um dos elementos de `vet`:

```
int busca(int n, int *vet, int elem)
{
  int i;
  for (i=0; i<n; i++)
  {
    if (elem == vet[i])
      return i;
  }
  return -1;
}
```

Exemplo: Busca em Vetor

- **Pior Caso:** o elemento não está no vetor
  - Neste caso são necessárias n comparações
  - T(n) = n → O(n) - Linear!
- **Melhor Caso:** o elemento é o primeiro
  - T(n) = O(1)
- **Caso Médio:**
  - n/2 comparações
  - T(n) = n/2 → O(n) - Linear!

Complexidade de algoritmos:
Na análise de complexidade, analisamos o "Pior Caso", o "Melhor Caso" e o "Caso Médio"

Exemplo: Busca em Vetor

- **E se o vetor estiver ordenado?**
  - Temos um segundo algoritmo:

```
int busca_ord(int n, int *vet, int elem)
{
  int i;
  for (i=0; i<n; i++)
  {
    if (elem == vet[i])
      return i;
    else if (elem < vet[i])
      return -1;
  }
  return -1;
}
```

Exemplo: Busca em Vetor

- Qual a complexidade do algoritmo de busca linear em vetor ordenado?

- **Melhor Caso:**
  - T(n) = O(1)

- **Pior Caso:**
  - T(n) = n → O(n) - Linear!

Um algoritmo pode ter a mesma complexidade de um outro, porém pode ser mais, ou menos, eficiente. Eficiência e Complexidade são coisas diferentes!

Exemplo: Busca em Vetor

- **É possível melhorar o algoritmo?**
  - **Sim!**

- **Procedimento:**
  - Compare o elemento d com o elemento do meio de v;
  - Se o elemento d for menor, pesquise a primeira metade do vetor;
  - Se o elemento d for maior, pesquise a segunda parte do vetor;
  - Se o elemento d for igual, retorne a posição;
  - Continue o procedimento subdividindo a parte de interesse até encontrar o elemento d ou chegar ao fim.

Exemplo: Busca em Vetor

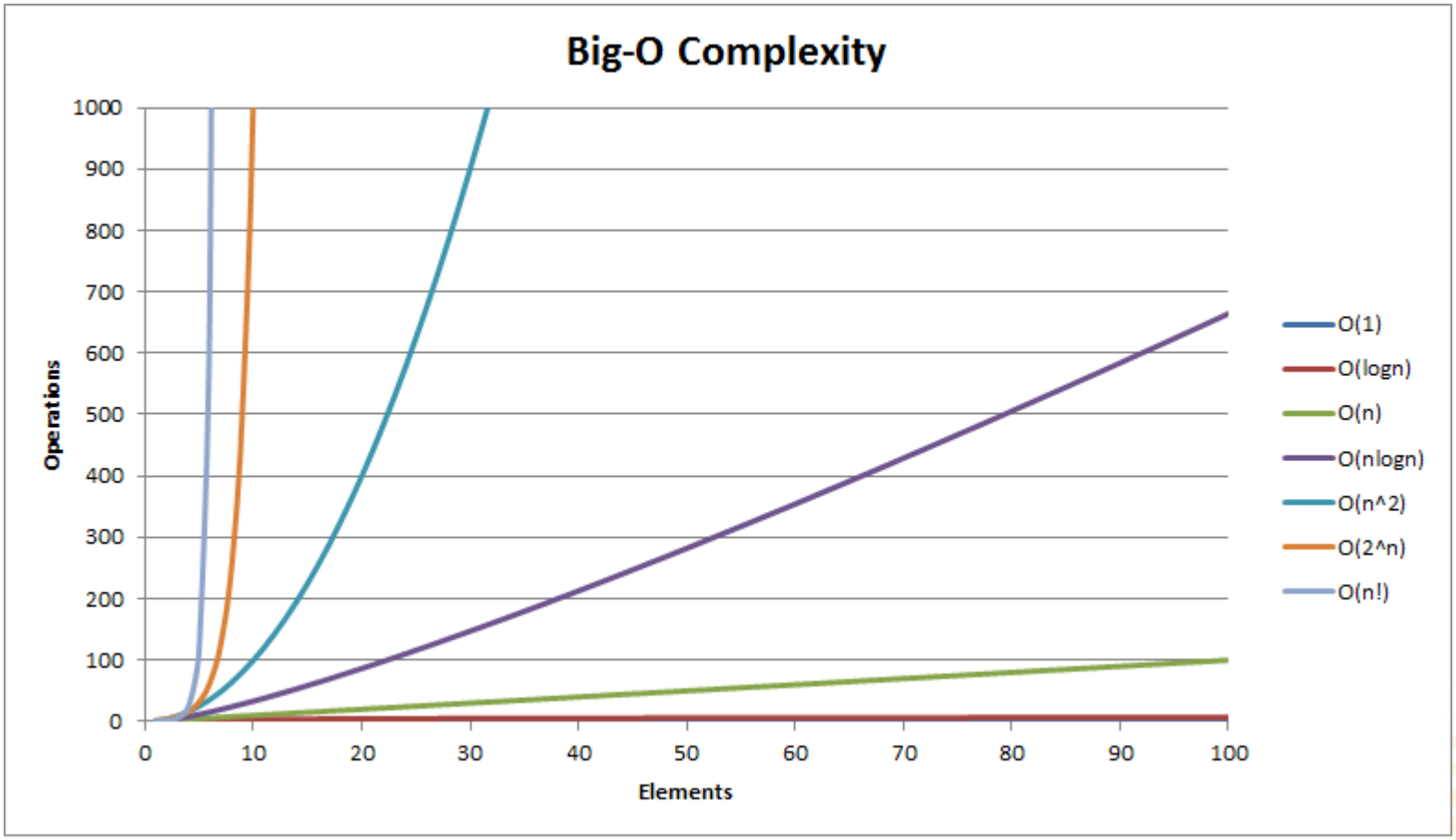
```
int busca_bin(int n, int *vet, int elem){
  int ini = 0;
  int fim = n-1;
  int meio;
  while(ini <= fim){
    meio = (ini + fim) / 2;
    if (elem < vet[meio])
      fim = meio - 1;
    else if (elem > vet[meio])
      ini = meio + 1;
    else
      return meio;
  }
  return -1;
}
```

Busca Binária em Vetor Ordenado

- **Pior caso:** elemento não está no vetor
  - 2 comparações são realizadas a cada ciclo;
  - a cada repetição, a parte considerada na busca é dividida na metade;
  - T(n) = **O(log n)**

Repetição	**Tamanho do Problema**
1	n
2	n/2
3	n/4
4	n/8
…	…
log n	1

Analise de Algoritmos – Tipos de Crescimentos



Diferença entre O(n) e O(log n)

Tamanho	O(n)	O(log n)
10	10 seg	3 seg
60	1 min	6 seg
600	10 min	9 seg
3 600	1 hora	12 seg
86 400	1 dia	16 seg
2 592 000	1 mês	21 seg
946 080 000	1 ano	30 seg
94 608 000 000	100 anos	36 seg

Programa da Disciplina

1. **Complexidade de Algoritmos**
   - Algoritmos
   - Eficácia vs Eficiência
   - Complexidade de Algoritmos
   - Análise Assintótica

2. **Técnicas de Projeto de Algoritmos**
   - Força Bruta/Busca Completa/Backtracking
   - Divisão e Conquista
   - Método Guloso
   - Programação Dinâmica

Programa da Disciplina

3. **Algoritmos de Processamento de Texto**
   - Busca de Padrões em Texto
     - Algoritmo de Força Bruta
     - Algoritmo de Boyer-Moore
     - Tries
   - Maior Subsequência Comum
     - Algoritmo de Força Bruta
     - Algoritmo de Programação Dinâmica

Programa da Disciplina

4. **Algoritmos de Ordenação**
   - Métodos de Ordenação
     - Bubble Sort
     - Selection Sort
     - Insertion Sort
     - Merge Sort
     - Quick Sort
   - Métodos de Ordenação de Complexidade Linear
     - Counting Sort
     - Radix Sort
     - Bucket Sort

Programa da Disciplina

5. **Algoritmos em Grafos**
   - Busca em Profundidade e Busca Largura;
   - Ordenação Topológica;
     - Algoritmo de Kahn;
     - Algoritmo baseado na busca em profundidade;
   - Componentes Fortemente Conectados;
     - Algoritmo de Kosaraju;
     - Algoritmo de Tarjan;
   - Árvores Geradoras Mínimas;
     - Algoritmo de Prim;
     - Algoritmo de Kruskal;
   - Distâncias Mínimas;
     - Algoritmo de Dijkstra;

Critério de Avaliação

- **Avaliação Teórica:**
  - Prova teórica envolvendo o conteúdo teórico e prático apresentado durante as aulas;

- **Avaliação Prática:**
  - Trabalhos e exercícios desenvolvidos em grupo ou individualmente;
  - Apresentação em aula;

Critério de Avaliação

- **G1:**
  - Prova: 7.0
  - Trabalhos e Exercícios: 3.0
- **G2:**
  - Prova: 7.0
  - Trabalhos e Exercícios: 3.0
- **MP = (G1 + G2)/2**

Critério de Avaliação

- **Nota Extra:**
  - https://www.urionlinejudge.com.br (ACADEMIC)
  - Problemas selecionados
  - Pontos extras nas notas da G1 e G2 para os alunos que resolverem mais problemas:
    - 1˚ colocado: 2.0
    - 2˚ colocado: 1.5
    - 3˚ colocado: 1.5
    - 4˚ colocado: 1.0
    - 5˚ colocado: 0.5

Critério de Avaliação

- Se a frequência nas aulas for < 75% o aluno será REPROVADO POR FALTA;

- Se a frequência nas aulas for ≥ 75%, então:

  Se MP ≥ 7.0, o aluno será aprovado e MF = MP;

  Se MP < 4.0, o aluno será reprovado e MF = MP;

  Se MP < 7.0 e MP ≥ 4.0, o aluno irá para o exame final e então:

  MF = (MP + PF)/2;

  Se MF < 5.0 o aluno será reprovado;

  Se MF ≥ 5.0 o aluno será aprovado;

Controle de Turma

- **Presença obrigatória!!!**
  - Chamada em qualquer momento da aula;
  - Alunos com menos de 75% de presença serão reprovados automaticamente e não poderão fazer prova final (independente da nota);

  - Não será aberta nenhuma exceção!

Pré-Requisitos

- Programação (qualquer linguagem)
- Algoritmos e Estruturas de Dados

Material das Aulas

- **Página do Curso:**
  - www.inf.puc-rio.br/~elima/paa/

- **Contato:**
  - edirlei@iprj.uerj.br

Bibliografia Principal

- Cormen, Leiserson, Rivest e Stein. **Algoritmos – Teoria e Prática**, 2ª. Edição, Editora Campus, 2002.

- Halim e Halim. **Competitive Programming**, 3rd Edition, 2003.

- Levitin. **Introduction to the Design and Analysis of Algorithms**, 3rd Edition, 2011.