Ordenação e Análise: Selection sort e Bubble sort

Rafael Alves da Costa

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
FATEC Carapicuíba

Aula 14 - Estrutura de Dados

11/2025

Sumário

Algoritmos: Força Bruta (Conceitos)

Força Bruta

- Força bruta é uma abordagem direta para resolver problemas, baseada na declaração do problema e nas definições envolvidas.
- Utiliza o poder computacional, não o intelecto humano: "Apenas faça!".
- Exemplo: calcular a^n para um número não nulo a e um número inteiro não negativo n pela multiplicação repetida de a.
- Aplicada com frequência em algoritmos simples, como o cálculo do mdc e multiplicação de matrizes.

Características e Aplicações da Força Bruta

- Aplicável a uma ampla variedade de problemas; única abordagem que resolve virtualmente qualquer problema.
- Útil para problemas como:
  - Ordenação
  - Pesquisa
  - Multiplicação de matrizes
  - Correspondência de strings
- Eficaz para problemas de instâncias pequenas ou ocasionais, onde algoritmos mais eficientes não são justificáveis.

Importância e Comparações

- Embora ineficiente para alguns problemas.
- É uma base para comparar e avaliar algoritmos mais eficientes.
- Vantagens:
  - Simplicidade e aplicabilidade geral.
  - Valor prático em problemas específicos e pequenos.

Selection Sort

Selection Sort

- **Passo Inicial:** Varre a lista para encontrar o menor elemento e o troca com o primeiro, colocando-o na posição final.
- **Passos Seguintes:** A cada nova passagem, procura-se o menor entre os últimos $n - i$ elementos e o troca com o elemento A_i.
- Em cada passo i, a lista assume a forma:

$$A_0 \leq A_1 \leq \cdots \leq A_{i-1} \,|\, A_i, \ldots, A_{min}, \ldots, A_{n-1}$$

- **Finalização:** Após $n - 1$ passagens, a lista estará completamente ordenada.
- **Vamos ver uma animação!!!**

Algoritmo Selection Sort

Algorithm SelectionSort

```
1:  Entrada: Um array A[0..n − 1] de elementos ordenáveis
2:  Saída: Array A[0..n − 1] ordenado em ordem crescente
3:  for i ← 0 to n − 2 do
4:      min ← i
5:      for j ← i + 1 to n − 1 do
6:          if A[j] < A[min] then
7:              min ← j
8:          end if
9:      end for
10:     Troque A[i] e A[min]
11: end for
```

Análise do Custo de Cada Linha

- **Linha 1 e 2 (Entrada e Saída):** Não contribui significativamente para o custo de execução, então o custo é $O(1)$.
- **Linha 3 (for $i \leftarrow 0$ to $n-2$):**
  - Executa $n-1$ vezes, pois i varia de 0 a $n-2$.
  - Custo por iteração: $O(1)$.
  - Custo total: $O(n-1) = O(n)$.
- **Linha 4 (min $\leftarrow i$):**
  - Executa $n-1$ vezes (uma vez para cada iteração de i).
  - Custo por iteração: $O(1)$.
  - Custo total: $O(n)$.

Análise do Custo de Cada Linha (Continuação)

- **Linha 5 (for $j \leftarrow i + 1$ to $n - 1$):**
  - Executada uma vez para cada valor de i.
  - Para cada i, j varia de $i + 1$ a $n - 1$, resultando em aproximadamente $n - i - 1$ iterações.
  - Custo total:

$$\sum_{i=0}^{n-2}(n - i - 1) = \frac{n(n-1)}{2} = O(n^2)$$

- **Linha 6 (if $A[j] < A[min]$):**
  - Executa aproximadamente $n - i - 1$ vezes para cada i, dentro do loop interno.
  - Custo total: $O(n^2)$.

Análise do Custo de Cada Linha (Continuação)

- **Linha 7 (min $\leftarrow j$):**
  - Executa em média metade das vezes que a linha 5 é avaliada, ou seja, aproximadamente $\frac{n^2}{2}$.
  - Custo total: $O(n^2)$.
- **Linha 10 (swap $A[i]$ and $A[min]$):**
  - Executa $n-1$ vezes (uma vez para cada iteração de i no loop externo).
  - Custo por iteração: $O(1)$.
  - Custo total: $O(n)$.

Tempo Total de Processamento

- Somando os custos principais:
  - Linhas 2, 3, e 7: $O(n)$
  - Linhas 4, 5, e 6: $O(n^2)$
- Como $O(n^2)$ domina $O(n)$, o tempo total de processamento do algoritmo Selection Sort é:

$$O(n^2)$$

- Complexidade final: $O(n^2)$ para melhor, pior e caso médio.

Bubble Sort

Bubble Sort

- O Bubble Sort compara elementos adjacentes da lista e os troca se estiverem fora de ordem.
- Ao repetir esse processo, o maior elemento "sobe"para a última posição da lista.
- O processo continua, movendo o segundo maior elemento para sua posição, até que após $n-1$ passadas, a lista esteja ordenada.
- Passada i $(0 \leq i \leq n-2)$ do algoritmo pode ser representada assim:

$$A_0, \ldots, A_j \leftrightarrow A_{j+1}, \ldots, A_{n-i-1} \,|\, A_{n-i} \leq \cdots \leq A_{n-1}$$

- Cada elemento "borbulha"para sua posição correta ao final de cada passada.

Algoritmo Bubble Sort

Algorithm BubbleSort

1: **Entrada:** Um array $A[0..n-1]$ de elementos ordenáveis
2: **Saída:** Array $A[0..n-1]$ ordenado em ordem crescente
3: **for** $i \leftarrow 0$ **to** $n-2$ **do**
4: **for** $j \leftarrow 0$ **to** $n-2-i$ **do**
5: **if** $A[j+1] < A[j]$ **then**
6: Troque $A[j]$ e $A[j+1]$
7: **end if**
8: **end for**
9: **end for**

Análise do Custo de Cada Linha

- **Linha 1 e 2 (Entrada e Saída):**
  - Não contribuem significativamente para o custo de execução.
  - Custo: $O(1)$.
- **Linha 3 (for $i \leftarrow 0$ to $n-2$):**
  - O laço externo executa $n-1$ vezes, pois i varia de 0 a $n-2$.
  - Custo por iteração: $O(1)$.
  - Custo total: $O(n)$.

Análise do Custo de Cada Linha (Continuação)

- **Linha 4 (for $j \leftarrow 0$ to $n-2-i$):**
  - O laço interno executa de $n-1$ a 1 vez, dependendo do valor de i.
  - Para cada i, o número de iterações do laço interno é aproximadamente $n-i-1$.
  - Custo total do laço interno, somando todas as iterações:

$$\sum_{i=0}^{n-2}(n-i-1) = \frac{(n-1)n}{2} = O(n^2)$$

  - Portanto, o custo total para essa linha é $O(n^2)$.

Análise do Custo de Cada Linha (Continuação)

- **Linha 5 (if $A[j+1] < A[j]$):**
  - Executado uma vez para cada iteração do laço interno.
  - Como o laço interno executa $O(n^2)$ vezes, o custo total para essa linha também é $O(n^2)$.
- **Linha 6 (swap $A[j]$ and $A[j+1]$):**
  - A troca é realizada apenas quando a condição do 'if' é satisfeita.
  - No pior caso, pode ocorrer a cada iteração do laço interno.
  - Custo total: $O(n^2)$.

Tempo Total de Processamento

- Somando os custos principais do algoritmo:
  - Linhas 3, 4, 5, e 6 contribuem com $O(n^2)$.
- Como o custo total é dominado por $O(n^2)$, o tempo total de processamento do algoritmo Bubble Sort é:

$$O(n^2)$$

Resumo da Complexidade

- **Melhor caso:** $O(n)$
  - Se o array já estiver ordenado, o Bubble Sort pode parar após a primeira iteração.
- **Pior caso:** $O(n^2)$
  - Ocorre em uma lista invertida, onde cada elemento precisa ser trocado em cada iteração.
- **Vamos ver uma animação!!!**

Considerações

Considerações

- **Recordando!!!**
  - Selection Sort.
  - Bubble Sort.
  - Exercícios.

Considerações

- **Recordando!!!**
  - Selection Sort.
  - Bubble Sort.
  - Exercícios.
- **Referências utilizadas!!!**

  - Cormen, T. H. et al. Algoritmos: Teoria e prática, 3a edição. Elsevier, 2012.
  - Levitin, A. Introduction to the Design and Analysis of Algorithms, 2007.
  - Goodrich, M. T., Tamassia, R., Goldwasser, M. H. (2013). Data structures and algorithms in Python.
