Aula 11: Algoritmos de Ordenação

Luís Felipe

UFF

03 de Novembro de 2025

Luís Felipe

03/11/25

Algoritmos de Ordenação

- O problema da ordenação é caracterizado pela organização de um conjunto de elemento do mesmo tipo segundo um critério de ordenação.
- Sem perda de generalidade, o problema será atacado considerando-se que:
  - Os elementos a serem ordenados são numéricos e estão armazenados em um vetor;
  - Deseja-se ordenar os elementos não decrescentemente:

    se `i` $<$ `j`, então `valores[i]` $\leq$ `valores[j]`
- Note que, ao realizar uma busca em uma lista ordenada, é possível obter um ganho de eficiência em comparação com a busca em uma lista não ordenada.
  - Aula passada vimos o Algoritmo de Busca Binária.
- Hoje veremos algumas formas de ordenar elementos de uma lista:
  - Selection Sort
  - Bubble Sort
  - Insertion Sort

Luís Felipe

03/11/25

Selection Sort

Ideia geral:

- Para cada posição i do vetor valores (desde a 1a até a última), o algoritmo procura pelo i-ésimo menor elemento e o coloca na posição i.
  - A colocação desse elemento se dá pela troca dele com quem estava na posição i.
  - Após fazer essa troca, o algoritmo continua da posição $i + 1$, caso não tenha concluído.

Luís Felipe

03/11/25

Selection Sort

```
# Operação que troca o conteúdo de duas células do vetor
def trocar(vals, posX, posY):
    temp = vals[posX]
    vals[posX] = vals[posY]
    vals[posY] = temp
    return None


# Operação que encontra o local do menor elemento do vetor
# considerando as células a partir de um dado início

def selecionarMenor(vals, inicio):
    localMenor = inicio
    for pos in range(inicio+1, len(vals)):
        if vals[pos] < vals[localMenor]:
            localMenor = pos
    return localMenor


# Método da Seleção (Selection Sort)

def ordenar(valores):
    for ind in range(len(valores)-1):
        menor = selecionarMenor(valores, ind)
        trocar(valores, ind, menor)
    return None
```

Luís Felipe

03/11/25

Utilizando Selection Sort

```
1  # Operação que troca o conteúdo de duas células do vetor
2  def trocar(vals, posX, posY):
3      temp = vals[posX]
4      vals[posX] = vals[posY]
5      vals[posY] = temp
6      return None
7
8
9  # Operação que encontra o local do menor elemento do vetor
10 # considerando as células a partir de um dado início
11 def selecionarMenor(vals, inicio):
12     localMenor = inicio
13     for pos in range(inicio+1, len(vals)):
14         if vals[pos] < vals[localMenor]:
15             localMenor = pos
16     return localMenor
17
18
19 # Método da Seleção (Selection Sort)
20 def ordenar(valores):
21     for ind in range(len(valores)-1):
22         menor = selecionarMenor(valores, ind)
23         trocar(valores, ind, menor)
24     return valores
25
26 # Saída no formato igual da entrada
27 def main():
28     lista = input().split()
29     lista_ordenada = ordenar(lista)
30     for i in range(len(lista_ordenada)):
31         print(lista_ordenada[i], end=' ')
32
33 main()
```

Luís Felipe

03/11/25

Exemplo de Execução — Selection Sort

Vetor inicial: [29, 10, 14, 37, 13]

Passo 1: menor elemento entre posições 0–4 é 10 (pos 1) Troca com posição 0 ⇒ [10, 29, 14, 37, 13]

Passo 2: menor elemento entre posições 1–4 é 13 (pos 4) Troca com posição 1 ⇒ [10, 13, 14, 37, 29]

Passo 3: menor elemento entre posições 2–4 é 14 (pos 2) Sem troca (já está no lugar) ⇒ [10, 13, 14, 37, 29]

Passo 4: menor elemento entre posições 3–4 é 29 (pos 4) Troca com posição 3 ⇒ [10, 13, 14, 29, 37]

Vetor ordenado: [10, 13, 14, 29, 37]

Luís Felipe

03/11/25

Análise de Complexidade — Selection Sort

No algoritmo anterior, há basicamente duas estruturas de repetição `for` aninhadas.

- A mais externa executa, para cada elemento do vetor, comparações com os elementos seguintes e, em seguida, uma troca.
  - Desta forma, aproximadamente $n(n+1)/2$ elementos são acessados.
- Ou seja, o número de elementos avaliados é da ordem de n^2.
- Portanto, sua complexidade é: $O(n^2)$

Luís Felipe

03/11/25

Ordenação pelo Método da Bolha (Bubble Sort)

- Esta estratégia executa $n - 1$ iterações, controladas por uma repetição mais externa.
- Em cada iteração, por meio de uma repetição interna:
  - Percorre-se todo o vetor, comparando cada par de elementos `valores[i]` e `valores[i+1]`.
  - Caso `valores[i] > valores[i+1]`, realiza-se a troca de posição.

Observação: Após cada iteração externa, o maior elemento ainda não ordenado "bolha" para o final do vetor.

Luís Felipe

03/11/25

Código — Bubble Sort

```python
def bubble_sort(valores):
    n = len(valores)
    for i in range(n-1):                    # loop externo (n-1 iterações)
        for j in range(n-1-i):              # loop interno
            if valores[j] > valores[j+1]:   # faz a troca
                temp = valores[j]
                valores[j] = valores[j+1]
                valores[j+1] = temp
    return None
```

Complexidade: No pior e no médio caso: $O(n^2)$.

Luís Felipe

03/11/25

Utilizando o Bubble Sort

```
1  def bubble_sort(valores):
2      n = len(valores)
3      for i in range(n-1):                    # loop externo (n-1 iterações)
4          for j in range(n-1-i):              # loop interno
5              if valores[j] > valores[j+1]:   # faz a troca
6                  temp = valores[j]
7                  valores[j] = valores[j+1]
8                  valores[j+1] = temp
9      return valores
10
11 def main():
12     lista = input().split()
13     lista_ordenada = bubble_sort(lista)
14     for i in range(len(lista_ordenada)):
15         print(lista_ordenada[i], end=' ')
16
17 main()
```

Luís Felipe

03/11/25

Exemplo de Execução — Bubble Sort

Vetor inicial: $[5, 3, 4, 1, 2]$

Iteração 1:

$[3, 5, 4, 1, 2] \rightarrow [3, 4, 5, 1, 2] \rightarrow [3, 4, 1, 5, 2] \rightarrow [3, 4, 1, 2, 5]$

Iteração 2:

$[3, 4, 1, 2, 5] \rightarrow [3, 1, 4, 2, 5] \rightarrow [3, 1, 2, 4, 5]$

Iteração 3:

$[3, 1, 2, 4, 5] \rightarrow [1, 2, 3, 4, 5]$

Iteração 4:

$[1, 2, 3, 4, 5]$ (ordenado)

Resultado final: $[1, 2, 3, 4, 5]$

Luís Felipe

03/11/25

Ordenação pelo Método da Inserção (Insertion Sort)

- Esta estratégia constrói o vetor ordenado de forma incremental.
- Para cada elemento (a partir da segunda posição), insere-o na posição correta dentro da parte já ordenada do vetor.
- Para inserir, desloca os elementos maiores uma posição à frente, abrindo espaço para o elemento atual.

Observação: Funciona bem para listas pequenas ou quase ordenadas.
Complexidade:

Pior e médio caso: $O(n^2)$ Melhor caso: $O(n)$

Luís Felipe

03/11/25

Código — Insertion Sort

```python
def insertion_sort(valores):
    n = len(valores)
    for i in range(1, n):
        atual = valores[i]
        j = i - 1
        # desloca elementos maiores que 'atual'
        while j >= 0 and valores[j] > atual:
            valores[j+1] = valores[j]
            j -= 1
        # insere o elemento na posição correta
        valores[j+1] = atual
    return None
```

Complexidade:

- Pior e médio caso: $O(n^2)$
- Melhor caso (já ordenado): $O(n)$

Luís Felipe

03/11/25

Utilizando o Insertion Sort

```python
def insertion_sort(valores):
    n = len(valores)
    for i in range(1, n):
        atual = valores[i]
        j = i - 1
        # desloca elementos maiores que 'atual'
        while j >= 0 and valores[j] > atual:
            valores[j+1] = valores[j]
            j -= 1
        # insere o elemento na posição correta
        valores[j+1] = atual
    return valores

# Saída no formato igual da entrada
def main():
    lista = input().split()
    lista_ordenada = insertion_sort(lista)
    for i in range(len(lista_ordenada)):
        print(lista_ordenada[i], end=' ')

main()
```

Luís Felipe

03/11/25

Exemplo de Execução — Insertion Sort

Vetor inicial: [5, 3, 4, 1, 2]

Passo 1: (i=1) Insere 3 antes de 5 [3, 5, 4, 1, 2]

Passo 2: (i=2) Insere 4 entre 3 e 5 [3, 4, 5, 1, 2]

Passo 3: (i=3) Insere 1 na posição inicial [1, 3, 4, 5, 2]

Passo 4: (i=4) Insere 2 entre 1 e 3 [1, 2, 3, 4, 5]

Resultado final: [1, 2, 3, 4, 5]

Luís Felipe

03/11/25

Vídeos com execução dos algoritmos

- Selection sort: https://www.youtube.com/watch?v=hFhf9djnM5A&list=RDhFhf9djnM5A&start_radio=1
- Bubble sort: https://www.youtube.com/watch?v=Iv3vgjM8Pv4&list=RDIv3vgjM8Pv4&start_radio=1
- Insertion sort: https://www.youtube.com/watch?v=QdQmAdyfmDI&list=RDQdQmAdyfmDI&start_radio=1