Estrutura de Dados em Python

Prof. Nisston Moraes Tavares de Melo

uniesp
Centro Universitário

Estrutura de dados em Python

Dra. Ana Carolina Costa de Oliveira
prof2121@iesp.edu.br

EXPERT ACADEMY

Aula 07

Agenda

- Ordenação
  - Bubble sort
  - Selection sort
  - Quick sort
- Exemplos

1 Ordenação

ORDENAÇÃO

- Após uma base de dados estar construída, pode ser necessário ordená-la, como:
  - Nomes em ordem alfabética
  - Alunos por nota
  - Clientes por CEP
  - Vendas por preço
  - Cidades em ordem crescente de população
- Ordenar dados pode ser um passo preliminar para pesquisá-los (pesquisa binária X pesquisa linear)

2 Bubble Sort

BUBBLE SORT – BOLHA

- Notavelmente lento e é o mais simples dos algoritmos de ordenação
- Funcionamento
  - Comparação de dois números
  - Se o da esquerda for maior, os elementos devem ser trocados
  - Desloca-se uma posição à direita
- À medida que o algoritmo avança, os itens maiores “surgem como uma bolha” na extremidade superior do vetor
- Visualização on-line: https://visualgo.net/en/sorting

BUBBLE SORT – BOLHA

- O algoritmo com 10 elementos faz 9 comparações na primeira passagem, 8 na segunda, 7 na terceira, etc (n – 1, n – 2, n – 3). Para 10 itens:
  - 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 45
- Big-O: $O(n^2)$
- O algoritmo faz cerca de $N^2/2$ comparações
- Há menos trocas do que há comparações, pois dois elementos serão trocados somente se precisarem
- Se os dados forem aleatórios, uma troca será necessária mais ou menos $N^2/4$ (no pior caso, com os dados em modo inverso, uma troca será necessária a cada comparação)

```
import numpy as np

 def bubble_sort(vetor):
   n = len(vetor)

   for i in range(n):
     for j in range(0, n - i - 1):
       if vetor[j] > vetor[j + 1]:
         temp = vetor[j]
         vetor[j] = vetor[j + 1]
         vetor[j + 1] = temp
   return vetor
```

3 Selection Sort

SELECTION SORT

- Melhora a ordenação pelo método da bolha reduzindo o número de trocas necessárias de N^2 para N
- O número de comparações permanece $N^2/2$
- Funcionamento
  - Percorrer todos os elementos e selecionar o menor
  - O menor elemento é trocado com o elemento da extremidade esquerda do vetor (posições iniciais)
  - Os elementos ordenados acumulam-se na esquerda
- Visualização on-line: https://visualgo.net/en/sorting

SELECTION SORT

- O algoritmo com 10 elementos faz 9 comparações na primeira passagem, 8 na segunda, 7 na terceira, etc (n – 1, n – 2, n – 3). Para 10 itens:
  - 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1 = 45
- Big-O: $O(n^2)$
- O algoritmo faz cerca de $N^2/2$ comparações (mesmo número de comparações que o método da bolha)
- Com 10 elementos, são requeridas menos de 10 trocas (geralmente é feita uma troca a cada passagem)
- Com 100 itens, são requeridas 4.950 comparações, mas menos de 100 trocas

```
def selection_sort(vetor):
  n = len(vetor)

  for i in range(n):
    id_minimo = i
    for j in range(i + 1, n):
      if vetor[id_minimo] > vetor[j]:
        id_minimo = j
    temp = vetor[i]
    vetor[i] = vetor[id_minimo]
    vetor[id_minimo] = temp

  return vetor
```

4 Quick Sort

QUICK SORT

- Algoritmo rápido e eficiente criado em 1960 para traduzir um dicionário de inglês para russo
- Funcionamento
  - O vetor é dividido em subvetores que são chamados recursivamente para ordenar os elementos
  - Estratégia da divisão e conquista
- Visualização on-line: https://visualgo.net/en/sorting

QUICK SORT

- Pior caso: $O(n^2)$ – quando o elemento pivot é o maior ou menor elemento
- Melhor caso: O(n*log n)

Exercício.


uniesp
Centro Universitário