gianfrancojoel_condoriluna_aula5Pratico

September 10, 2024

1 Aula 05 - Exercício prático Ordenação linear

Aluno: Gian Franco Joel Condori Luna

1.0.1 1) (1,0) Gere um vetor aleatório contendo 1000 datas (dia/mês/ano), sendo que dia contém valores de 1 a 31; mês de 1 a 12; e ano de 2000 a 2024. Você pode elaborar a melhor estratégia para armazenar a data (ex: TAD com dia, mês e ano).

- Ordene as datas usando radix sort combinado com dois métodos (contagem + um método estável e contagem + um não estável).
- Calcule o tempo que o algoritmo levou em cada caso.
- Comente o motivo de a ordenação não funcionar em um dos casos.

1.0.2 Solução:

Passos:

- Criaremos um vetor de 1000 datas aleatórias (dia/mês/ano).
- Usaremos tuplas (dia, mês, ano) para armazenar as datas.
- Vamos implementar o Radix Sort, combinado com:
  - Um método estável (Ordenação por contagem).
  - Um método não estável (QuickSort).
- Cálculo do tempo: Usaremos time para medir o tempo de execução.

[4]:
```
# PYTHON version 3.10.12
import numpy as np # type: ignore
import time

# Seed para que os dados sejam sempre os mesmos e o
# exercício possa ser replicado.
np.random.seed(42)
```

[14]:
```
# Função para gerar 1000 datas aleatórias
def gerar_datas(n=1000):
    datas = []
    for _ in range(n):
        dia = np.random.randint(1, 31)
        mes = np.random.randint(1, 12)
```

```
        ano = np.random.randint(2000, 2024)
        datas.append((dia, mes, ano))
    return datas

# Função estável (Ordenação por contagem)
# Fonte: CHATGPT 3.5
def counting_sort(arr, index):
    max_val = max(arr, key=lambda x: x[index])[index]
    min_val = min(arr, key=lambda x: x[index])[index]
    range_of_values = max_val - min_val + 1

    count = [0] * range_of_values
    output = [0] * len(arr)

    for i in arr:
        count[i[index] - min_val] += 1

    for i in range(1, len(count)):
        count[i] += count[i - 1]

    for i in range(len(arr) - 1, -1, -1):
        output[count[arr[i][index] - min_val] - 1] = arr[i]
        count[arr[i][index] - min_val] -= 1

    return output

# Função não estável (QuickSort)
# Fonte: CHATGPT 3.5
def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    else:
        pivot = arr[len(arr) // 2]
        left = [x for x in arr if x < pivot]
        middle = [x for x in arr if x == pivot]
        right = [x for x in arr if x > pivot]
        return quick_sort(left) + middle + quick_sort(right)

# Radix Sort com Counting Sort (estável)
def radix_sort_estavel(arr):
    # Ordenar por dia, depois por mês, depois por ano
    arr = counting_sort(arr, 0)  # Dia
    arr = counting_sort(arr, 1)  # Mês
    arr = counting_sort(arr, 2)  # Ano
    return arr

# Radix Sort com Quick Sort (não estável)
```

```python
def radix_sort_nao_estavel(arr):
    # Ordenar por dia, depois por mês, depois por ano
    arr = quick_sort(arr)  # Quick Sort diretamente
    return arr

# Função principal
def main():
    datas = gerar_datas()

    # Ordenar com Radix Sort + Ordenação por contagem (estável)
    inicio_estavel = time.time()
    ordenado_estavel = radix_sort_estavel(datas.copy())
    fim_estavel = time.time()
    tempo_estavel = fim_estavel - inicio_estavel

    # Ordenar com Radix Sort + Quick Sort (não estável)
    inicio_nao_estavel = time.time()
    ordenado_nao_estavel = radix_sort_nao_estavel(datas.copy())
    fim_nao_estavel = time.time()
    tempo_nao_estavel = fim_nao_estavel - inicio_nao_estavel

    # Exibir os tempos
    print(f"Tempo de Radix Sort com método estável (Ordenação por contagem ):␣
 ↪{tempo_estavel:.6f} segundos")
    print(f"Tempo de Radix Sort com método não estável (QuickSort):␣
 ↪{tempo_nao_estavel:.6f} segundos")

    # Verificação do comportamento
    print("\nComportamento:")
    print("")
    print("Método estável")
    print(ordenado_estavel[0:50])
    print("")
    print("Método não estável")
    print(ordenado_nao_estavel[0:50])

if __name__ == "__main__":
    main()
```

```
Tempo de Radix Sort com método estável (Ordenação por contagem ): 0.004780
segundos
Tempo de Radix Sort com método não estável (QuickSort): 0.008690 segundos

Comportamento:

Método estável
[(14, 1, 2000), (26, 1, 2000), (27, 1, 2000), (7, 2, 2000), (11, 2, 2000), (18,
```

```
2, 2000), (20, 2, 2000), (23, 2, 2000), (4, 3, 2000), (7, 3, 2000), (16, 3,
2000), (18, 3, 2000), (26, 3, 2000), (27, 3, 2000), (28, 3, 2000), (3, 4, 2000),
(30, 4, 2000), (14, 5, 2000), (24, 5, 2000), (29, 5, 2000), (17, 6, 2000), (22,
6, 2000), (27, 6, 2000), (28, 6, 2000), (29, 6, 2000), (5, 7, 2000), (16, 7,
2000), (23, 7, 2000), (1, 8, 2000), (15, 8, 2000), (10, 9, 2000), (10, 9, 2000),
(3, 10, 2000), (13, 10, 2000), (6, 11, 2000), (29, 11, 2000), (29, 11, 2000),
(3, 1, 2001), (13, 1, 2001), (27, 1, 2001), (5, 2, 2001), (12, 2, 2001), (22, 2,
2001), (7, 3, 2001), (26, 3, 2001), (5, 4, 2001), (22, 4, 2001), (30, 4, 2001),
(8, 5, 2001), (11, 5, 2001)]

Método não estável
[(1, 1, 2008), (1, 1, 2012), (1, 2, 2004), (1, 2, 2009), (1, 2, 2011), (1, 2,
2018), (1, 3, 2017), (1, 4, 2011), (1, 4, 2014), (1, 5, 2003), (1, 5, 2015), (1,
5, 2018), (1, 6, 2011), (1, 7, 2010), (1, 7, 2023), (1, 8, 2000), (1, 8, 2019),
(1, 9, 2009), (1, 9, 2012), (1, 9, 2020), (1, 9, 2023), (1, 10, 2007), (1, 10,
2017), (1, 10, 2020), (1, 10, 2023), (2, 1, 2004), (2, 2, 2008), (2, 2, 2010),
(2, 3, 2006), (2, 3, 2015), (2, 3, 2017), (2, 4, 2005), (2, 5, 2017), (2, 5,
2017), (2, 5, 2022), (2, 6, 2005), (2, 6, 2013), (2, 6, 2016), (2, 7, 2005), (2,
7, 2006), (2, 7, 2007), (2, 7, 2008), (2, 7, 2010), (2, 7, 2012), (2, 7, 2015),
(2, 8, 2001), (2, 8, 2004), (2, 8, 2014), (2, 9, 2023), (2, 10, 2020)]
```

1.0.3 Comentário sobre porque a ordenação não funcionar em um dos casos:

- O método estável ORDENOU corretamente as datas.
- O método não estável NÃO ORDENOU corretamente as datas.
- A razão pela qual o Radix Sort com Quick Sort pode falhar é porque Quick Sort não preserva a ordem relativa dos elementos iguais, o que pode quebrar a ordenação de datas que dependem da estabilidade (como quando dias ou meses são iguais, mas queremos preservar a ordem entre os anos).