

Classificação e Pesquisa de Dados

Aula 11

Classificação em Tempo Linear $O(n)$: Counting Sort, Radix Sort e Bucket Sort

UFRGS

INF01124

1

Classificação de Dados em Tempo Linear

- ◆ Heapsort e Merge Sort apresentam desempenho de $\Theta(n \log_2 n)$
- ◆ Quicksort tem custo médio de $\mathcal{O}(n \log_2 n)$
- ◆ Estes algoritmos se baseiam em comparações entre os valores de chaves a serem ordenados e, portanto, são algoritmos de classificação por comparação
- ◆ Todo algoritmo de classificação por comparação está limitado por $\mathcal{O}(n \log_2 n)$. Portanto, Heapsort e Merge sort são assintoticamente ótimos
- ◆ Existem algoritmos que apresentam custo linear (i.e., $\mathcal{O}(n)$)

2

Classificação de Dados em Tempo Linear

Algoritmos de classificação em tempo linear exploram determinadas propriedades do conjunto de dados a ser ordenado

Principais Algoritmos

- Counting sort
- Radix sort
- Bucket sort

3

Classificação de Dados por Distribuição de Chaves – Counting Sort

◆ Princípio de classificação

- Assume que cada um dos n elementos de entrada é um inteiro no intervalo de 1 a k
- Quando $k = \mathcal{O}(n)$, o algoritmo executa em um tempo $\mathcal{O}(n)$
- Ideia básica: determinar para cada elemento x da entrada, o número de elementos menores que x
- Pequeno ajuste necessário para suportar valores de chaves repetidos
- Utiliza dois outros vetores auxiliares

◆ O algoritmo é estável

4

Procedimento Counting Sort

```
Proc counting-sort (A, B, k)
/* B: vetor que conterá a saída ordenada;
k: tamanho do intervalo */
begin
  for i ← 1 to k
    do C[i] ← 0; /* inicializa acumuladores */
  for i ← 1 to length[A]
    do C[A[i]] ← C[A[i]] + 1; /* C[i] = numero de elementos iguais a i */
  for i ← 2 to k
    do C[i] ← C[i] + C[i-1]; /* C[i] = numero de elementos menores ou iguais a i */
  for j ← length[A] downto 1 do
    begin
      B[C[A[j]]] ← A[j]; /* Armazena A[j] em sua posição final */
      C[A[j]] ← C[A[j]] - 1; /* Decrementa acum. p/ suportar chaves repetidas */
    end
end end
```

5

Counting Sort Exemplo

```
Proc counting-sort (A, B, k)
/* B: vetor que conterá a saída ordenada;
k: tamanho do intervalo */
begin
  for i ← 1 to k
    do C[i] ← 0;
  for i ← 1 to length[A]
    do C[A[i]] ← C[A[i]] + 1;
  for i ← 2 to k
    do C[i] ← C[i] + C[i-1];
  for j ← length[A] downto 1 do
    begin
      B[C[A[j]]] ← A[j];
      C[A[j]] ← C[A[j]] - 1;
    end
end end
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| A | 3 | 6 | 4 | 1 | 3 | 4 | 1 | 4 |
| C | 2 | 0 | 2 | 3 | 0 | 1 | | |
| C | 2 | 2 | 4 | 7 | 7 | 8 | | |

6

Counting Sort Exemplo (Cont.)

Diagram illustrating the execution of the **for** loop in the **isSorted** function. The array **A** is initially **[3, 6, 4, 1, 3, 4, 1, 4]**. The loop iterates from **j = 7** down to **j = 1**, comparing **A[j]** with **A[j-1]**. The array is updated in place during each iteration.

| Iteration | j | A[j] | A[j-1] | Comparison | Result |
|-----------|---|------|--------|------------|--------|
| 1 | 7 | 4 | 1 | 4 > 1 | True |
| 2 | 6 | 1 | 3 | 1 < 3 | False |
| 3 | 5 | 3 | 4 | 3 < 4 | False |
| 4 | 4 | 1 | 3 | 1 < 3 | False |
| 5 | 3 | 4 | 6 | 4 < 6 | False |
| 6 | 2 | 3 | 4 | 3 < 4 | False |
| 7 | 1 | 6 | 3 | 6 > 3 | True |

The final array **A** is **[3, 2, 2, 4, 7, 7, 8, 8]**.

Exercícios

- ◆ Realize a ordenação do vetor abaixo usando counting sort, mostrando os valores intermediários para os vetores B e C :

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 1 | 8 | 7 | 5 | 2 | 2 | 4 | 3 |
|---|---|---|---|---|---|---|---|---|---|

Classificação de Dados por Distribuição de Chaves – Radix Sort

Princípio de classificação

- Seja um conjunto de n chaves numéricas, cada uma composta por d dígitos decimais
- Utiliza 10 escaninhos representando os dígitos de 0 a 9
- Classifica os números de acordo com o dígito menos significativo e combina os resultados, preservando a ordem obtida
- Utilizando o resultado do passo anterior como entrada, repete-se o procedimento para a casa decimal menos significativa ainda não visitada, até que todas tenham sido visitadas
- São necessários d passos para classificar o conjunto de n chaves

Classificação de Dados por Distribuição de Chaves – Radix Sort

- É essencial que os dígitos sejam classificados utilizando um algoritmo estável

| | | | |
|-----|-----|-----|-----|
| 329 | 720 | 720 | 329 |
| 457 | 355 | 329 | 355 |
| 657 | 436 | 436 | 436 |
| 839 | 457 | 839 | 457 |
| 436 | 657 | 355 | 657 |
| 720 | 329 | 457 | 720 |
| 355 | 839 | 657 | 839 |

Procedimento Radix Sort

```

Proc radix-sort (A, d)
begin
  for i ← 1 to d do
    use um algoritmo de sort estável para ordenar o vetor A segundo o dígito i
  end

```

Exercícios

- ◆ Realize a ordenação dos elementos dos vetores abaixo usando o radix sort:

| | |
|-----|-----|
| 278 | BOI |
| 827 | SIM |
| 501 | COM |
| 489 | XIS |
| 263 | MEL |
| 012 | CEM |
| 575 | LAR |

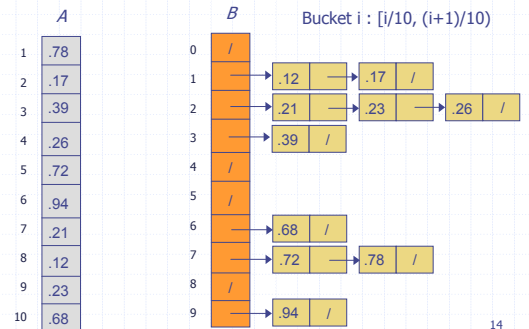
Classificação de Dados por Distribuição de Chaves – Bucket Sort

Princípio de classificação

- Assume que cada um dos n elementos de entrada foi gerado por um processo randômico que distribui valores uniformemente no intervalo $[0,1)$
- Subdivide o intervalo $[0,1)$ em n sub-intervalos (*buckets*) de mesmo tamanho
- Distribui os valores de entrada nos buckets
- Para produzir a sequência ordenada:
 - Dentro de cada bucket, os valores são classificados utilizando insertion sort
 - Visita-se os buckets em ordem crescente (de valores do sub-intervalo), listando-se os elementos em cada bucket visitado

13

Bucket Sort Exemplo



14

Procedimento Bucket Sort

```

Proc bucket-sort (A)
begin
  n ← length[A]
  for i ← 1 to n
  do insira A[i] na lista B[⌊nA[i]⌋]
  for i ← 0 to n-1
  do ordene a lista B[i] usando insertion sort
  concatene as listas B[0], B[1], ..., B[n-1], nesta ordem
end
    
```

Apesar do custo do algoritmo insertion sort ser $O(n^2)$, devido à probabilidade de um dado elemento "cair" em um bucket $B[i]$ ser $1/n$, o custo do algoritmo é $O(n)$. Detalhes da prova podem ser encontrados no livro *Introduction to Algorithms*, by Cormen et al.

15