Estrutura de Dados

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

12 de julho de 2016

Algoritmos Lineares de Ordenação

Algoritmos lineares de Ordenação

- Os seguintes algoritmos de ordenação têm complexidade $O(n)$:
- **Counting Sort**: Elementos são números inteiros *pequenos*, mais precisamente, $x \in O(n)$.
- **Radix Sort**: Elementos são números inteiros de *comprimento máximo constante*, isto é, independe de n.
- **Bucket Sort**: Elementos são números reais *uniformemente distribuídos* no intervalo [0..1).

Counting Sort: Conceitos

- Considere o problema de ordenar um vetor $A[1..n]$ de inteiros quando se sabe que todos os inteiros estão no *intervalo* entre 0 e k.
- Podemos ordenar o vetor simplesmente contando, para cada inteiro i no vetor, quantos elementos do vetor são menores que i.
- É exatamente o que faz o algoritmo *Counting Sort*.

Counting Sort: Algoritmo

Algoritmo COUNTING-SORT(A,n)

para $i \leftarrow 0$ **até** k **faça**
$C[i] \leftarrow 0$

para $j \leftarrow 1$ **até** n **faça**
▷ $C[i]$ é o número de js tais que $A[j] = i$
$C[A[j]] \leftarrow C[A[j]] + 1$

para $i \leftarrow 1$ **até** k **faça**
▷ $C[i]$ é o número de js tais que $A[j] \leqslant i$
$C[i] \leftarrow C[i] + C[i-1]$

para $j \leftarrow n$ **até** 1 **faça**
$B[C[A[j]]] \leftarrow A[j]$
$C[A[j]] \leftarrow C[A[j]] - 1$

Counting Sort - Simulação: n = 6, k = 9

Os valores: 5, 4, 8, 7, 2, 9.

$C[i] \leftarrow 0$

C[0..9]:

0	0	0	0	0	0	0	0	0	0

$C[A[j]] \leftarrow C[A[j]] + 1$

C[0..9]:

0	0	1	0	1	1	0	1	1	1

$C[i] \leftarrow C[i] + C[i-1]$

C[0..9]:

0	0	1	1	2	3	3	4	5	6

$B[C[A[j]]] \leftarrow A[j]$

B[1..6]:

2	4	5	7	8	9

Radix Sort

- Podemos evitar o uso excessivo de memória adicional começando pelo dígito menos significativo.
- Esta é a proposta do *Radix Sort*
- Para funcionar é necessário um método de ordenação estável:
  - Por exemplo, o Counting-Sort.

Radix Sort: Algoritmo

- Suponha que os elementos do vetor A a ser ordenado sejam números inteiros de até d dígitos. O Radix Sort é simplesmente:

Algoritmo RADIX-SORT(A,n,d)
para $i \leftarrow 1$ **até** d **faça**
Ordene $A[1..n]$ pelo i-ésimo dígito usando um método estável.

Radix-Sort: Exemplo

329		720		720		329
457		355		329		355
657		436		436		436
839	$\rightarrow$	457	$\rightarrow$	839	$\rightarrow$	457
436		657		355		657
720		329		457		720
355		839		657		839
$\uparrow$		$\uparrow$		$\uparrow$		

Bucket Sort: Conceitos

- Supõe que os n elementos da entrada estão distribuídos uniformemente no intervalo $[0, 1)$.
- A ideia é dividir o intervalo $[0, 1)$ em n segmentos de mesmo tamanho (buckets) e distribuir os n elementos nos seus respectivos segmentos. Como os elementos estão distribuídos uniformemente, espera-se que o número de elementos seja aproximadamente o mesmo em todos os segmentos.
- Em seguida, os elementos de cada segmento são ordenados por um método qualquer. Finalmente os segmentos ordenados são concatenados em ordem crescente.

Bucket Sort: Algoritmo

Algoritmo BucketSort(A, n)
  para $icor \leftarrow 1$ **até** n **faça**
    Insira $A[i]$ na lista ligada $B[[nA[i]]]$
  para $i \leftarrow 0$ **até** $n-1$ **faça**
    Ordene a lista $B[i]$ com Insertion-Sort
  Concatene as listas $B[0], B[1], \ldots, B[n-1]$

Bucket Sort: Exemplo

$$A = \begin{array}{r|l} 1 & .78 \\ 2 & .17 \\ 3 & .39 \\ 4 & .26 \\ 5 & .72 \\ 6 & .94 \\ 7 & .21 \\ 8 & .12 \\ 9 & .23 \\ 10 & .68 \end{array} \qquad B = \begin{array}{r|l} 0 & \\ 1 & .12,\ .17 \\ 2 & .21,\ .23,\ .26 \\ 3 & .39 \\ 4 & \\ 5 & \\ 6 & .68 \\ 7 & .72,\ .78 \\ 8 & \\ 9 & .94 \end{array}$$