Projeto e Análise de Algoritmos I

Pedro Hokama

Fontes

- [*clrs*] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
- [*timr*] Algorithms Illuminated Series, Tim Roughgarden
- Desmistificando Algoritmos, Thomas H. Cormen.
- Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
- Stanford Algorithms
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V`
  `https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZ17A5HEK6`
- Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
- Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420

Qualquer erro é de minha responsabilidade.

Algoritmos lineares para ordenação

Estudaremos algoritmos de ordenação de tempo linear:

- Counting Sort
  - os elementos são inteiros pequenos
  - os valores são limitados por $O(n)$
- Radix Sort
  - representação numérica com comprimento constante
  - os valores dos elementos independente de n
- Bucket Sort
  - elementos do vetor são sorteados no intervalo [0..1)
  - os valores são distribuídos uniformemente

Counting Sort

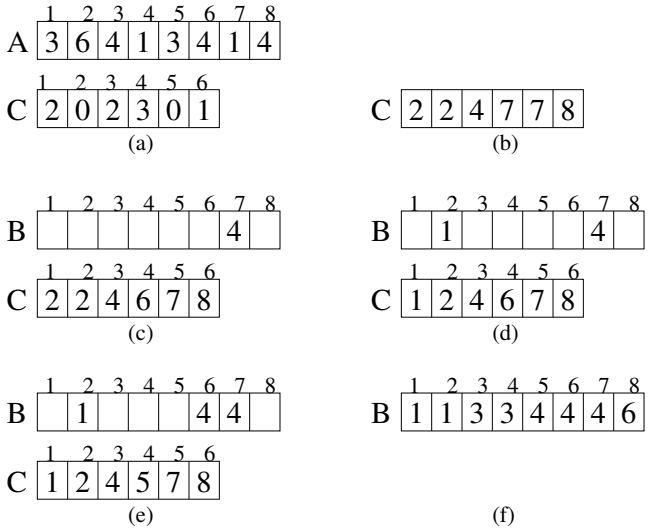
Ideia:

- entrada é um vetor $A[1 \ldots n]$ de inteiros
- saída será outro vetor $B[1 \ldots n]$ de inteiros
- supomos que o valores estão no intervalo entre 0 e k
- para cada inteiro i no vetor, contamos o número $C[i]$ de elementos menores ou iguais i em A
- então, na posição $C[i]$ do vetor B, deve haver o elemento i

Counting Sort - Algoritmo

```
COUNTING-SORT(A, B, n, k)
1  para i ← 0 até k
2      C[i] ← 0
3  para j ← 1 até n
4      C[A[j]] ← C[A[j]] + 1
5  para i ← 1 até k
6      C[i] ← C[i] + C[i − 1]
7  para j ← n decrescendo até 1
8      B[C[A[j]]] ← A[j]
9      C[A[j]] ← C[A[j]] − 1
```


Counting Sort - Exemplo



Counting Sort - Complexidade

Qual a complexidade de COUNTING-SORT?

- COUNTING-SORT realiza $O(n+k)$ instruções elementares
- quando $k \in O(n)$, ele tem complexidade $O(n)$

E a cota inferior de $\Omega(n \log n)$ para ordenação?

- não há comparações entre elementos de A
- a cota só vale para algoritmos baseados em comparação

Algoritmos in-place e estáveis

Algoritmos de ordenação in-place

- um algoritmo é in-place se a quantidade de memória adicional requerida independe de n
- são in-place INSERTION-SORT, HEAP-SORT
- não são in-place MERGE-SORT, COUNTING-SORT

Algoritmos de ordenação estáveis

- um algoritmo é estável se elementos com chaves iguais mantêm-se na ordem passada na entrada.
- são estáveis INSERTION-SORT, MERGE-SORT, QUICKSORT, COUNTING-SORT
- o HEAP-SORT não é estável

Radix Sort

Ideia:

- a entrada é um vetor $A[1 \ldots n]$ de inteiros
- cada inteiro é representado na base k-ária
- supomos que um inteiro tem d dígitos
- por exemplo, CEPs são inteiros de 8 dígitos na base 10
- ordenaremos um dígito por vez com um algoritmo estável
- ordenamos primeiro os dígitos menos significativos

Radix Sort - Algoritmo

RADIX-SORT(A, n, d)
1 **para** $i \leftarrow 1$ até d
2 ordene $A[1 \ldots n]$ 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
		$\uparrow$		$\uparrow$		$\uparrow$

Radix Sort - Correção

Demonstramos a correção do algoritmo por indução

- suponha que o vetor de números esteja ordenado em relação aos $i - 1$ dígitos menos significativos
- ordene o vetor pelo i-ésimo dígito com um método estável
- considere números a e b no vetor resultante com $a < b$:
  1. se os i-ésimos dígitos forem distintos, então eles estão em ordem em relação a este dígito, independentemente dos $i - 1$ primeiros dígitos; portanto, a aparece antes de b
  2. se os i-ésimos dígitos forem iguais, então por hipótese de indução, eles estavam em ordem pelos $i - 1$ primeiros dígitos antes da i-ésima iteração; como usamos um algoritmo estável nesta iteração, a ordem é mantida de forma que a aparecia antes de b e continua assim

Radix Sort - Complexidade

Qual é a complexidade do Radix-Sort?

- o algoritmo de ordenação usado tem tempo $\Theta(f(n))$
- então Radix-Sort tem tempo total $\Theta(d\,f(n))$
- quando d é constante, a complexidade é $\Theta(f(n))$
- se usarmos Counting-Sort, a complexidade é $\Theta(n+k)$
  - lembre-se de que $k-1$ é o maior valor do número na entrada do Counting-Sort (e.g., $k=9$ na base decimal).
  - se k é contante, então o tempo resultante é $\Theta(n)$

Radix Sort - Comparação

Vamos comparar Radix-Sort e Merge-Sort

- temos $n = 2^{20}$ números de 64 bits
- interpretamos os números na base $k = 2^{16}$ (cada 16 bits corresponde a um dígito)
- cada número tem $d = 4$ dígitos

1. Ordenando com Merge-Sort
   - executamos cerca de $n \log_2 n = 20 \times 2^{20}$ comparações
   - usamos um vetor auxiliar de tamanho 2^{20}
2. Ordenando com Radix-Sort
   - utilizamos Counting-Sort como sub-rotina de ordenação
   - executamos cerca de $d(n+k) = 4(2^{20} + 2^{16})$ operações
   - usamos dois vetores auxiliares de tamanhos 2^{16} e 2^{20}

Conclusão: Radix-Sort foi mais rápido, mas usa mais memória

Bucket Sort

Ideia:

- temos n números em $[0,1)$ distribuídos uniformemente
- dividimos $[0,1)$ em n segmentos de tamanhos iguais
- particionamos os elementos em listas (buckets) correspondentes aos segmentos
- o número esperado de elementos por lista é constante
- podemos ordenar cada lista independentemente
- e concatenar as listas já ordenadas

Bucket Sort - Algoritmo

```
BUCKET-SORT(A, n)
1  para i ← 0 até n − 1
2      crie uma lista ligada vazia B[i]
3  para i ← 1 até n
4      insira A[i] na lista ligada B[⌊n A[i]⌋]
5  para i ← 0 até n − 1
6      ordene a lista B[i] com INSERTION-SORT
7  concatene as listas B[0], B[1], ..., B[n − 1]
```

Bucket Sort - Exemplo

$A =$

1	.78
2	.17
3	.39
4	.26
5	.72
6	.94
7	.21
8	.12
9	.23
10	.68

$B =$

0	
1	.12,.17
2	.21,.23,.26
3	.39
4	
5	
6	.68
7	.72,.78
8	
9	.94

Bucket Sort - Correção

Considere dois elementos x e y da entrada com $x < y$:

- se ambos terminam na mesma lista
  - x aparecerá antes de y já que a lista foi ordenada
  - e se manterão ordenados após a concatenação
- se terminam em listas $B[i]$ e $B[j]$, respectivamente
  - como $x < y$, temos $i = \lfloor nx \rfloor \leq \lfloor ny \rfloor = j$ e então $i < j$
  - assim, x aparecerá antes de y após a concatenação

Bucket Sort - Complexidade

O tempo de execução é uma váriável aleatória

- denote por $T(n)$ o tempo de execução do algoritmo
- denote por n_i o tamanho de $B[i]$
- observe que o número de elementos é $n = \sum_{i=1}^{n} n_i$

Somando o tempo das operações

- particionamos os elementos em tempo $\Theta(n)$
- ordenamos cada lista em tempo $\Theta(n_i^2)$
- concatenamos todas as listas em tempo $\Theta(n)$

$$T(n) = \sum_{i=0}^{n-1} \Theta(n_i^2) + \Theta(n)$$

Bucket Sort - Pior caso

Pior caso

$$T(n) \leq \sum_{i=0}^{n-1} cn_i^2 + \Theta(n) \leq cn^2 + \Theta(n) = O(n^2)$$

- um pior caso ocorre se todos números caem em uma lista
- o tempo de execução é maior se há listas muito grandes
- mas o número esperado em cada lista é pequeno

Bucket Sort - Tempo esperado

Tempo esperado

$$
\begin{aligned}
E[T(n)] &= E\left[\sum_{i=0}^{n-1} cn_i^2\right] + \Theta(n) \\
&= \sum_{i=0}^{n-1} E[cn_i^2] + \Theta(n) \\
&= \sum_{i=0}^{n-1} cE[n_i^2] + \Theta(n)
\end{aligned}
$$

Bucket Sort - Tempo esperado (cont)

Queremos calcular $E[n_i^2]$

- seja X_{ij} a variável que indica se $A[j]$ está em $B[i]$
- assim, temos $n_i = \sum_{j=1}^{n} X_{ij}$

$$
\begin{aligned}
E[n_i^2] &= E\left[\left(\sum_{j=1}^{n} X_{ij}\right)^2\right] \\
&= E\left[\left(\sum_{j=1}^{n}\sum_{k=1}^{n} X_{ij}X_{ik}\right)\right] \\
&= E\left[\sum_{j=1}^{n} X_{ij}^2 + \sum_{j=1}^{n}\sum_{k=1,k\neq j}^{n} X_{ij}X_{ik}\right] \\
&= \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{j=1}^{n}\sum_{k=1,k\neq j}^{n} E[X_{ij}X_{ik}]
\end{aligned}
$$

Bucket Sort - Tempo esperado (cont)

Como X_{ij} e X_{ik} são independentes

$$
\begin{aligned}
E[n_i^2] &= \sum_{j=1}^{n} E[X_{ij}^2] + \sum_{j=1}^{n}\sum_{k=1,k\neq j}^{n} E[X_{ij}X_{ik}] \\
&= \sum_{j=1}^{n} E[X_{ij}] + \sum_{j=1}^{n}\sum_{k=1,k\neq j}^{n} E[X_{ij}]E[X_{ik}] \\
&= \sum_{j=1}^{n} \frac{1}{n} + \sum_{j=1}^{n}\sum_{k=1,k\neq j}^{n} \frac{1}{n}\frac{1}{n} \\
&= 1 + n(n-1)\frac{1}{n^2} \\
&= 2 - \frac{1}{n} = O(1)
\end{aligned}
$$

Bucket Sort - Tempo esperado (cont)

Voltando à estimativa de $E[T(n)]$, temos

$$
\begin{aligned}
E[T(n)] &= \sum_{i=0}^{n-1} cE[n_i^2] + \Theta(n) \\
&= \sum_{i=0}^{n-1} cO(1) + \Theta(n) \\
&= \Theta(n)
\end{aligned}
$$