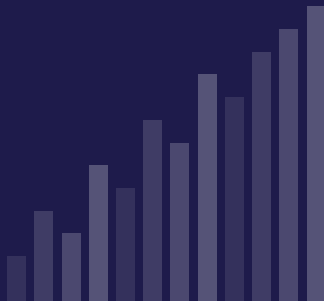


COMPLEMENTO 5 · BLOCO COMPLEMENTAR: ORDENAÇÃO AVANÇADA (OPCIONAL)

Bucket sort e seleção em tempo linear

Seção 8.4 e capítulo 9

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Ordenar dados uniformes em $[0, 1)$ em tempo esperado linear com baldes
- ▶ Achar o i -ésimo menor sem ordenar, em tempo esperado $\Theta(n)$
- ▶ Conhecer a seleção determinística linear (mediana das medianas)
- ▶ Medir $\approx 3,39n$ comparações para a mediana



Teoria

Bucket sort e a hipótese de distribuição;
RANDOMIZED-SELECT; SELECT



Prática em C

Seleção aleatorizada em C



Experimento

Mediana em $3,39n$; bucket sort uniforme contra concentrado



Conexões

Estatística, bancos de dados, redes e aprendizado de máquina

Bucket sort

- ▶ Supõe entrada **uniforme** em $[0, 1)$.
- ▶ n baldes; o valor x vai para o balde $\lfloor nx \rfloor$.
- ▶ Cada balde é ordenado por inserção; depois os baldes são concatenados.
- ▶ Com n_i elementos no balde i :
 $E[n_i^2] = 2 - 1/n$, então o tempo esperado é $\Theta(n)$.

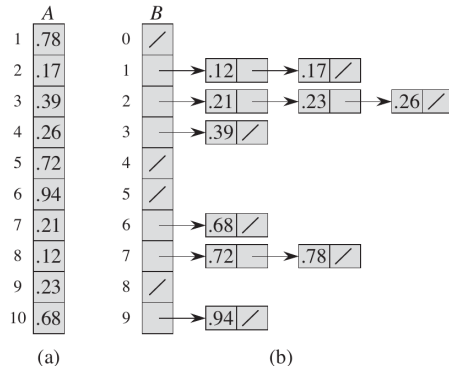


Figura 8.4 de Cormen et al. (2009) — 10 valores em $[0, 1)$ distribuídos em 10 baldes, cada um ordenado e

O i -ésimo menor

- ▶ Como o quicksort, mas recursão em **um** lado só.
- ▶ Tempo esperado $\Theta(n)$: em média, o subproblema encolhe em proporção constante.
- ▶ Para a mediana:
 $\approx 2(1 + \ln 2) n \approx 3,39 n$ comparações.
- ▶ Pior caso: $\Theta(n^2)$, mas só com sorteios muito azarados.

Randomized-Select(A, p, r, i)

```
1 se  $p = r$  então
2   retorne  $A[p]$ 
3  $q \leftarrow \text{RANDOMIZED-PARTITION}(A, p, r)$ 
4  $k \leftarrow q - p + 1$  // posição do pivô
5 se  $i = k$  então
6   retorne  $A[q]$ 
7 senão se  $i < k$  então
8   retorne Randomized-Select( $A, p, q - 1, i$ )
9 senão
10  retorne Randomized-Select( $A, q + 1, r, i - k$ )
```

Seleção linear no pior caso: mediana das medianas

SELECT (Blum, Floyd, Pratt, Rivest e Tarjan, 1973):

1. Divida em grupos de 5 e ache a mediana de cada um.
2. Ache recursivamente a mediana x dessas medianas.
3. Particione em torno de x : pelo menos $\frac{3n}{10} - 6$ elementos ficam de cada lado.
4. Recursão no lado certo.

$$T(n) \leq T(\lceil n/5 \rceil) + T(7n/10 + 6) + O(n) = O(n).$$

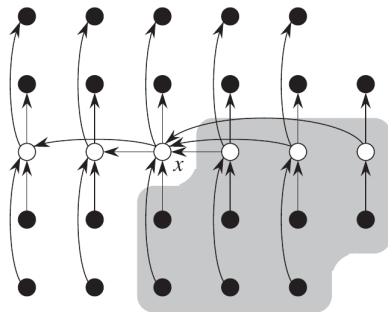


Figura 9.1 de Cormen et al. (2009) — grupos de 5; x é a mediana das medianas. A região sombreada é garantidamente maior que x .



Prática em C

- ▶ Partição aleatorizada do complemento 2
- ▶ Laço em vez de recursão: só um lado continua
- ▶ Lê o vetor e i ; imprime o i -ésimo menor

Seleção aleatorizada em C (1/3)

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void troca(int A[], int i, int j) { int t = A[i]; A[i] = A[j]; A[j] = t; }
5
6  int particiona_aleatorio(int A[], int p, int r) {
7      troca(A, p + rand() % (r - p + 1), r);
8      int x = A[r], i = p - 1;
9      for (int j = p; j < r; j++)
10         if (A[j] <= x) troca(A, ++i, j);
11     troca(A, i + 1, r);
12     return i + 1;
13 }
```

Seleção aleatorizada em C (2/3)

```
14
15 /* RANDOMIZED-SELECT: como o quicksort, mas desce só no lado que contém a resposta. */
16 int seleciona(int A[], int p, int r, int i) {
17     while (p < r) {
18         int q = particiona_aleatorio(A, p, r);
19         int k = q - p + 1; // o pivô é o k-ésimo menor de A[p..r]
20         if (i == k) return A[q];
21         if (i < k) r = q - 1;
22         else { p = q + 1; i -= k; }
23     }
24     return A[p];
25 }
```

Seleção aleatorizada em C (3/3)

```
27 int main(void) {
28     srand(15);
29     int n, i;
30     if (scanf("%d", &n) != 1 || n <= 0) return 1;
31     int *A = malloc(n * sizeof(int));
32     for (int k = 0; k < n; k++)
33         if (scanf("%d", &A[k]) != 1) return 1;
34     if (scanf("%d", &i) != 1 || i < 1 || i > n) return 1;
35     printf("%dº menor: %d\n", i, seleciona(A, 0, n - 1, i));
36     free(A);
37     return 0;
38 }
```



Experimento

- ▶ Mediana: média de no mínimo 200 execuções por n
- ▶ Bucket sort: dados uniformes contra todos no mesmo balde
- ▶ Deslocamentos da inserção dentro dos baldes

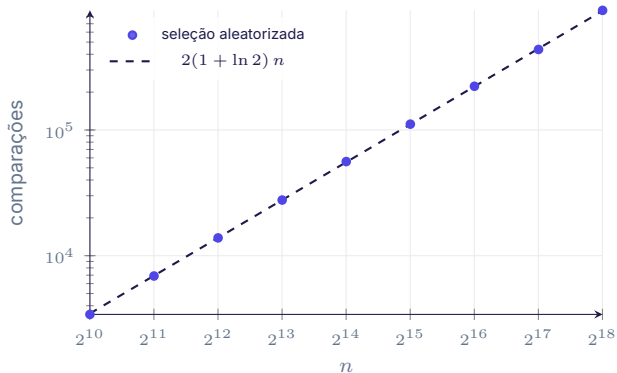
Seleção linear e baldes sob duas distribuições

Hipótese. A seleção aleatorizada da mediana faz em média $2(1 + \ln 2)n \approx 3,39n$ comparações: **linear**. No bucket sort, com dados uniformes cada balde tem $O(1)$ elementos e os deslocamentos somam $\approx n/4$; se todos caem no mesmo balde, vira uma inserção: $\approx n^2/4$.

Método

1. Seleção: $n = 2^{10}$ a 2^{18} ; muitas execuções por n (uma só varia de $2n$ a $5n$)
2. Bucket sort: n baldes, n de 1.000 a 16.000
3. Concentrada: todos os valores em $[0, 1/n)$
4. Contadores de comparações e deslocamentos

Comparações para achar a mediana



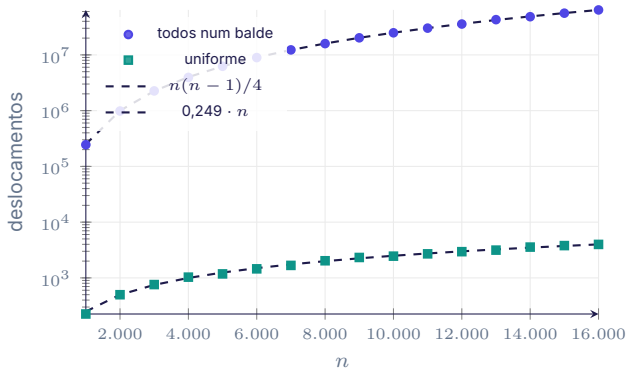
n	medido	razão
1.024	3.412	0,984
2.048	6.895	0,994
4.096	13.852	0,999
8.192	27.706	0,999
16.384	56.064	1,011
262.144	894.455	1,008

razão = medido $\div$ $2(1 + \ln 2) n$

✓ desvio máx. 1,5%

Achar a mediana custa só 3,4 comparações por elemento; ordenar, $1,39 \lg n$ por elemento (mais de 25 para $n = 2^{18}$).

Bucket sort: a distribuição importa



n	medido	razão
1.000	244.686	0,980
4.000	3.943.630	0,986
7.000	12.297.851	1,004
10.000	24.905.365	0,996
13.000	42.870.998	1,015
16.000	64.025.802	1,000

razão = medido $\div$ $n(n-1)/4$

✓ desvio máx. 1,5%

Uniforme: cerca de $0,25 \cdot n$ deslocamentos (linear). Concentrada: $n^2/4$, a inserção pura.

EXPERIMENTO

O que o experimento mostrou

A seleção da mediana fica em $\approx 3,39n$ comparações em média: **linear**, sem ordenar. O bucket sort é linear **se** a hipótese de uniformidade vale; com todos os dados num balde, volta a ser quadrático. **Algoritmos de caso médio dependem da distribuição da entrada; aleatorizados, só da sorte.**

>_ Reproduza: `gcc -O2 compl5_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Estatística e dados
- ▶ Bancos de dados
- ▶ Aprendizado de máquina
- ▶ Tabelas hash

Onde isso aparece na computação

Estatística e dados

Mediana, quartis e percentis (p50, p99) são seleções. Ferramentas de monitoramento calculam percentis de latência sem ordenar milhões de amostras.

Bancos de dados

`ORDER BY ... LIMIT k` e `PERCENTILE_CONT` usam seleção; histogramas em baldes guiam o otimizador de consultas.

Aprendizado de máquina

Árvores de decisão (k-d trees) dividem pela mediana de cada coordenada; a mediana é um estimador robusto contra valores extremos.

Tabelas hash

O bucket sort é uma tabela hash ordenada: a análise dos baldes ($E[n_i^2]$) é a mesma do encadeamento da aula 19.

Para casa

Exercícios

1. (8.4-1) Ilustre o bucket sort em $\langle .79, .13, .16, .64, .39, .20, .89, .53, .71, .42 \rangle$.
2. (8.4-2) Qual o pior caso do bucket sort? Como torná-lo $O(n \lg n)$ sem perder o caso médio?
3. (9.2-3) Escreva RANDOMIZED-SELECT iterativo.
4. (9.3-1) No SELECT, grupos de 7 funcionam? E grupos de 3?
5. Implemente o SELECT determinístico e meça a constante. Por que ninguém o usa na prática?

Leitura

Cormen et al., seção 8.4 e capítulo 9.

Código

josedihego.net/estruturas