

Projeto e Análise de Algoritmos*

Estatísticas de ordem

Lehilton Pedrosa

Segundo Semestre de 2017

*Criado por C. de Souza, C. da Silva, O. Lee, F. Miyazawa et al.

Estatísticas de Ordem

Estatísticas de Ordem (Problema da Seleção)

- ▶ Estamos interessados em resolver o

Problema da Seleção:

Dado um conjunto A de n números reais e um inteiro i , determinar o i -ésimo menor elemento de A .

- ▶ Casos particulares importantes:

Mínimo : $i = 1$

Máximo : $i = n$

Mediana : $i = \lfloor \frac{n+1}{2} \rfloor$ (mediana inferior)

Mediana : $i = \lceil \frac{n+1}{2} \rceil$ (mediana superior)

Mínimo

Recebe um vetor $A[1 \dots n]$ e devolve o **mínimo** do vetor.

```
MÍNIMO( $A, n$ )  
1   $mín \leftarrow A[1]$   
2  para  $j \leftarrow 2$  até  $n$  faça  
3      se  $mín > A[j]$   
4          então  $mín \leftarrow A[j]$   
5  devolva  $mín$ 
```

Número de comparações: $n - 1 = \Theta(n)$

Aula passada: não é possível fazer com menos comparações.

Mínimo e máximo

Recebe um vetor $A[1 \dots n]$ e devolve o **mínimo** e o **máximo** do vetor.

```
MINMAX( $A, n$ )  
1   $mín \leftarrow máx \leftarrow A[1]$   
2  para  $j \leftarrow 2$  até  $n$  faça  
3      se  $A[j] < mín$   
4          então  $mín \leftarrow A[j]$   
5      se  $A[j] > máx$   
6          então  $máx \leftarrow A[j]$   
7  devolva ( $mín, máx$ )
```

Número de comparações: $2(n - 1) = 2n - 2 = \Theta(n)$

É possível fazer melhor!

Mínimo e máximo

- ▶ Processe os elementos em pares. Para cada par, compare o menor com o mínimo atual e o maior com o máximo atual. Isto resulta em 3 comparações para cada 2 elementos.
- ▶ Se n for ímpar, inicialize o mínimo e o máximo como sendo o primeiro elemento.
- ▶ Se n for par, inicialize o mínimo e o máximo comparando os dois primeiros elementos.

Número de comparações:

$3\lfloor n/2 \rfloor$ se n for ímpar

$3\lfloor n/2 \rfloor - 2$ se n for par

Pode-se mostrar que isto é o melhor possível.

(Exercício ★ do CLRS)

Problema da Seleção – primeira solução

Recebe $A[1 \dots n]$ e i tal que $1 \leq i \leq n$
e devolve valor do i -ésimo menor elemento de $A[1 \dots n]$

```
SELECT-ORD( $A, n, i$ )  
1  ORDENE( $A, n$ )  
2  devolva  $A[i]$ 
```

ORDENE pode ser *MergeSort* ou *HeapSort*.

A complexidade de tempo de SELECT-ORD é $O(n \lg n)$.

Será que não dá para fazer melhor que isso?

Afinal, consigo achar o mínimo e/ou máximo em tempo $O(n)$.

Relembrando – PARTIÇÃO

Problema: reorganizar um dado vetor $A[p \dots r]$ e devolver um índice q , $p \leq q \leq r$, tais que

$$A[p \dots q - 1] \leq A[q] < A[q + 1 \dots r]$$

Entrada:

| | | | | | | | | | | |
|---|-----|----|----|----|----|----|----|----|-----|----|
| | p | | | | | | | | r | |
| A | 99 | 33 | 55 | 77 | 11 | 22 | 88 | 66 | 33 | 44 |

Saída:

| | | | | | | | | | | |
|---|-----|----|----|-----|----|----|----|----|-----|----|
| | p | | | q | | | | | r | |
| A | 33 | 11 | 22 | 33 | 44 | 55 | 99 | 66 | 77 | 88 |

Relembrando – Particione

Rearranja $A[p \dots r]$ de modo que $p \leq q \leq r$ e $A[p \dots q-1] \leq A[q] < A[q+1 \dots r]$.

```
PARTICIONE( $A, p, r$ )
1   $x \leftarrow A[r]$   ▷  $x$  é o “pivô”
2   $i \leftarrow p-1$ 
3  para  $j \leftarrow p$  até  $r-1$  faça
4      se  $A[j] \leq x$ 
5          então  $i \leftarrow i+1$ 
6                   $A[i] \leftrightarrow A[j]$ 
7   $A[i+1] \leftrightarrow A[r]$ 
8  devolva  $i+1$ 
```

Problema da Seleção – segunda solução

Suponha que queremos achar o i -ésimo menor de $A[1 \dots n]$.

- ▶ Executamos **PARTICIONE** e este rearranja o vetor e devolve um índice k tal que

$$A[1 \dots k - 1] \leq A[k] < A[k + 1 \dots n].$$

- ▶ Eis a ideia do algoritmo:
 - ▶ Se $i = k$, então o pivô $A[k]$ é o i -ésimo menor! (Yeesss!)
 - ▶ Se $i < k$, então o i -ésimo menor está em $A[1 \dots k - 1]$;
 - ▶ Se $i > k$, então o i -ésimo menor está em $A[k + 1 \dots n]$.

Problema da Seleção – segunda solução

Recebe $A[p\dots r]$ e i tal que $1 \leq i \leq r-p+1$
e devolve o i -ésimo menor elemento de $A[p\dots r]$.

```
SELECT-NL( $A, p, r, i$ )
```

```
1 se  $p = r$ 
```

```
2     então devolva  $A[p]$ 
```

```
3  $q \leftarrow$  PARTICIONE( $A, p, r$ )
```

```
4  $k \leftarrow q - p + 1$ 
```

```
5 se  $i = k$   $\triangleright$  pivô é o  $i$ -ésimo menor!
```

```
6     então devolva  $A[q]$ 
```

```
7     senão se  $i < k$ 
```

```
8         então devolva SELECT-NL( $A, p, q - 1, i$ )
```

```
9         senão devolva SELECT-NL( $A, q + 1, r, i - k$ )
```

Segunda solução – complexidade

| | SELECT-NL(A, p, r, i) | Tempo |
|---|---|-------------|
| 1 | se $p = r$ | $\Theta(1)$ |
| 2 | então devolva $A[p]$ | $O(1)$ |
| 3 | $q \leftarrow \text{PARTICIONE}(A, p, r)$ | $\Theta(n)$ |
| 4 | $k \leftarrow q - p + 1$ | $\Theta(1)$ |
| 5 | se $i = k$ | $\Theta(1)$ |
| 6 | então devolva $A[q]$ | $O(1)$ |
| 7 | senão se $i < k$ | $O(1)$ |
| 8 | então devolva SELECT-NL($A, p, q - 1, i$) | $T(k - 1)$ |
| 9 | senão devolva SELECT-NL($A, q + 1, r, i - k$) | $T(n - k)$ |

$$T(n) = \max\{T(k - 1), T(n - k)\} + \Theta(n)$$

Exercício: mostre que $T(n) \in \Theta(n^2)$

Segunda solução – complexidade

- ▶ A complexidade de `SELECT-NL` no pior caso é $\Theta(n^2)$.
- ▶ Então é melhor usar `SELECT-ORD`?
- ▶ Não, `SELECT-NL` é muito eficiente na prática.
- ▶ Vamos mostrar que no **caso médio** `SELECT-NL` tem complexidade $O(n)$.

SELECT aleatorizado

O pior caso do **SELECT-NL** ocorre devido a uma escolha infeliz do pivô.

Um modo de evitar isso é usar aleatoriedade (como no **QUICKSORT-ALEATÓRIO**).

PARTICIONE-ALEATÓRIO(A, p, r)

1 $j \leftarrow \text{RANDOM}(p, r)$

2 $A[j] \leftrightarrow A[r]$

3 devolva **PARTICIONE**(A, p, r)

Algoritmo SELECT-ALEAT

Recebe $A[p \dots r]$ e i tal que $1 \leq i \leq r - p + 1$
e devolve o i -ésimo menor elemento de $A[p \dots r]$

SELECT-ALEAT(A, p, r, i)

```
1  se  $p = r$ 
2    então devolva  $A[p]$ 
3   $q \leftarrow$  PARTICIONE-ALEATÓRIO( $A, p, r$ )
4   $k \leftarrow q - p + 1$ 
5  se  $i = k$   $\triangleright$  pivô é o  $i$ -ésimo menor
6    então devolva  $A[q]$ 
7  senão se  $i < k$ 
8    então devolva SELECT-ALEAT( $A, p, q - 1, i$ )
9  senão devolva SELECT-ALEAT( $A, q + 1, r, i - k$ )
```

Análise do tempo esperado

Recorrência para o tempo esperado de SELECT-ALEAT.

$T(n)$ = complexidade de tempo de SELECT-ALEAT.

$$T(0) = \Theta(1)$$

$$T(1) = \Theta(1)$$

$$T(n) \leq \sum_{k=1}^n X_k T(\max\{k-1, n-k\}) + \Theta(n),$$

onde X_k é variável aleatória binária que é igual a 1 sse o vetor $A[p \dots q]$ tem k elementos

$E[T(n)]$ é $\Theta(???)$.

Análise do tempo esperado

$$\begin{aligned} E[T(n)] &\leq E\left[\sum_{k=1}^n X_k T(\max\{k-1, n-k\}) + an\right] \\ &\leq \sum_{k=1}^n E[X_k] E[T(\max\{k-1, n-k\})] + an \\ &\leq \sum_{k=1}^n \frac{1}{n} E[T(\max\{k-1, n-k\})] + an \\ &\leq \frac{2}{n} \sum_{k=\lceil n/2 \rceil}^{n-1} E[T(k)] + an \end{aligned}$$

pois

$$\max\{k-1, n-k\} = \begin{cases} k-1 & \text{se } k > \lceil n/2 \rceil, \\ n-k & \text{se } k \leq \lceil n/2 \rceil. \end{cases}$$

Demonstração: $E[T(n)] \leq cn$

$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + an$$

$$\stackrel{\text{hi}}{\leq} \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an$$

$$= \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an$$

$$= \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an$$

Demonstração: $E[T(n)] \leq cn$

$$\begin{aligned} E[T(n)] &\leq \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an \\ &\leq \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2} \right) + an \\ &= \frac{c}{n} \left(\frac{3n^2}{4} + \frac{n}{2} - 2 \right) + an \\ &\leq \frac{3cn}{4} + \frac{c}{2} + an \\ &= cn - \left(\frac{cn}{4} - \frac{c}{2} - an \right) \leq cn. \end{aligned}$$

Isto funciona se $c > 4a$ e $n \geq 2c/(c - 4a)$.

Logo, $E[T(n)] = O(n)$.

O tempo esperado de `SELECT-ALEAT` é $O(n)$.

Na verdade,

O tempo esperado de `SELECT-ALEAT` é $\Theta(n)$.

Veremos depois:

Algoritmo que resolve o Problema da Seleção em tempo linear (no pior caso).

Problema da Seleção – terceira solução

Problema da Seleção:

Dado um conjunto A de n números reais e um inteiro i , determinar o i -ésimo menor elemento de A .

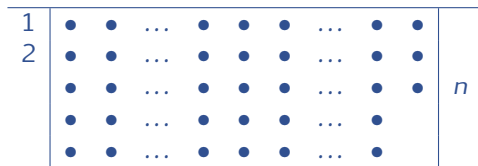
Veremos um **algoritmo linear** para o Problema da Seleção.

BFPRT = Blum, Floyd, Pratt, Rivest e Tarjan

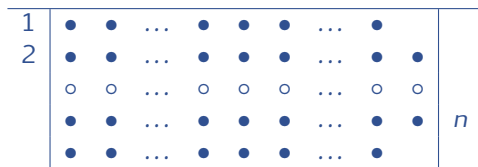
Para simplificar a exposição, vamos supor que os elementos em A são distintos.

Problema da Seleção – terceira solução

1. Divida os n elementos em $\lfloor \frac{n}{5} \rfloor$ subconjuntos de 5 elementos e um subconjunto de $n \bmod 5$ elementos.



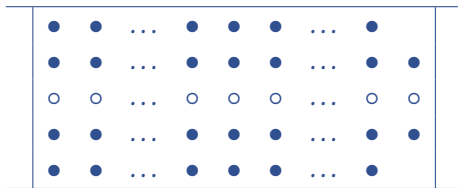
2. Encontre a **mediana** de cada um dos $\lfloor \frac{n}{5} \rfloor$ subconjuntos.



Na figura acima, cada subconjunto está em ordem crescente, de cima para baixo.

Problema da Seleção – terceira solução

3. Determine, recursivamente, a **mediana x** das **medianas** dos subconjuntos de no máximo 5 elementos.



A figura acima é a mesma que a anterior, com as colunas ordenadas pela mediana de cada grupo. A ordem dos elementos em cada coluna permanece inalterada. Por simplicidade de exposição, supomos que a última coluna permanece no mesmo lugar.

Note que o algoritmo não ordena as medianas!

Problema da Seleção - terceira solução

4. Usando x como pivô, particione o conjunto original A criando dois subconjuntos $A_{<}$ e $A_{>}$, onde
- ▶ $A_{<}$ contém os elementos $< x$ e
 - ▶ $A_{>}$ contém os elementos $> x$.

Se a posição final de x após o particionamento é k , então $|A_{<}| = k - 1$ e $|A_{>}| = n - k$.

Problema da Seleção - terceira solução

5. Finalmente, para encontrar o i -ésimo menor elemento do conjunto, compare i com a posição k de x após o particionamento:
 - ▶ Se $i = k$, x é o elemento procurado;
 - ▶ Se $i < k$, então determine recursivamente o i -ésimo menor elemento do subconjunto $A_{<}$;
 - ▶ Senão, determine recursivamente o $(i - k)$ -ésimo menor elemento do subconjunto $A_{>}$.

Note que esta parte é idêntica ao feito em **SELECT-NL** e em **SELECT-ALEAT**. O que diferencia este algoritmo dos outros é a escolha do **pivô**. Escolhendo-se a mediana das medianas, vamos poder garantir que nenhum dos lados é muito “grande”.

Terceira solução – complexidade

$T(n)$: complexidade de tempo no pior caso

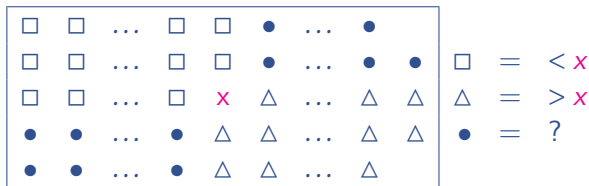
1. Divisão em subconjuntos de 5 elementos. $\Theta(n)$
2. Encontrar a mediana de cada subconjunto. $\Theta(n)$
3. Encontrar x , a mediana das medianas. $T(\lceil n/5 \rceil)$
4. Particionamento com pivô x . $O(n)$
5. Encontrar o i -ésimo menor de $A_{<}$ $T(k-1)$
OU encontrar o $i-k$ -ésimo menor de $A_{>}$. $T(n-k)$

Temos então a recorrência

$$T(n) = T(\lceil n/5 \rceil) + T(\max\{k-1, n-k\}) + \Theta(n)$$

Terceira Solução - Complexidade

O diagrama abaixo classifica os elementos da última figura.



Veja que o número de elementos $> x$, isto é \triangle s, é no mínimo $\frac{3n}{10} - 6$.

Isto porque no mínimo $\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil$ grupos contribuem com 3 elementos $> x$, exceto possivelmente o último e aquele que contém x . Portanto, $3 \left(\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2 \right) \geq \frac{3n}{10} - 6$.

Terceira Solução - Complexidade

Da mesma forma, o número de elementos $< x$, isto é \square s, é no mínimo $\frac{3n}{10} - 6$.

Assim, no passo 5 do algoritmo,

$$\max\{k - 1, n - k\} \leq n - \left(\frac{3n}{10} - 6\right) \leq \frac{7n}{10} + 6.$$

A recorrência $T(n)$ está agora completa:

$$T(n) \leq \begin{cases} \Theta(1), & n \leq 140 \\ T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor + 6) + \Theta(n), & n > 140, \end{cases}$$

140 é um “número mágico” que faz as contas funcionarem...

A solução é $T(n) \in \Theta(n)$

Solução da recorrência: $T(n) \leq cn$

$$\begin{aligned}T(n) &\leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor + 6) + an \\ &\stackrel{\text{hi}}{\leq} c\lceil n/5 \rceil + c(\lfloor 7n/10 \rfloor + 6) + an \\ &\leq c(n/5 + 1) + c(7n/10 + 6) + an \\ &= 9cn/10 + 7c + an \\ &= cn + (-cn/10 + 7c + an) \\ &\leq cn,\end{aligned}$$

Quero que $(-cn/10 + 7c + an) \leq 0$.

Isto equivale a $c \geq 10a(n/(n-70))$ quando $n > 70$. Como $n > 140$, temos $n/(n-70) \leq 2$ e assim basta escolher $c \geq 20a$.

Algoritmo SELECT

Recebe $A[p \dots r]$ e i tal que $1 \leq i \leq r - p + 1$
e devolve um índice q tal que $A[q]$ é o i -ésimo menor elemento
de $A[p \dots r]$.

```
SELECT( $A, p, r, i$ )
1  se  $p = r$ 
2      então devolva  $p$   $\triangleright p$  e não  $A[p]$ 
3   $q \leftarrow$ 
PARTICIONE-BFPRT( $A, p, r$ )
4   $k \leftarrow q - p + 1$ 
5  se  $i = k$ 
6      então devolva  $q$   $\triangleright q$  e não  $A[q]$ 
7  senão se  $i < k$ 
8      então devolva SELECT( $A, p, q - 1, i$ )
9  senão devolva SELECT( $A, q + 1, r, i - k$ )
```

Rearranja $A[p \dots r]$ e devolve um índice q , $p \leq q \leq r$, tal que $A[p \dots q-1] \leq A[q] < A[q+1 \dots r]$ e

$$\max\{k-1, n-k\} \leq \left\lfloor \frac{7n}{10} \right\rfloor + 6,$$

onde $n = r - p + 1$ e $k = q - p + 1$.

- ▶ Divida o vetor em $\lfloor n/5 \rfloor$ grupos de tamanho 5 e um grupo ≤ 5 ,
- ▶ ordene cada grupo e determine a mediana de cada um deles,
- ▶ determine a mediana das medianas chamando `SELECT` (!!)
- ▶ e particione o vetor em torno desse valor.

PARTICIONE-BFPRT

PARTICIONE-BFPRT(A, p, r) $\triangleright n := r - p + 1$

- 1 para $j \leftarrow p, p+5, p+5 \cdot 2, \dots$ até $p+5(\lceil n/5 \rceil - 1)$ faça
- 2 **ORDENE**($A, j, j+4$)
- 3 **ORDENE**($A, p+5\lfloor n/5 \rfloor, n$)

- 4 para $j \leftarrow 1$ até $\lceil n/5 \rceil - 1$ faça
- 5 $A[j] \leftrightarrow A[p+5j-3]$
- 6 $A[\lceil n/5 \rceil] \leftrightarrow A[\lfloor (p+5\lfloor n/5 \rfloor + n)/2 \rfloor]$

- 7 $k \leftarrow \text{SELECT}(A, p, p+\lceil n/5 \rceil - 1, \lfloor (\lceil n/5 \rceil + 1)/2 \rfloor)$

- 8 $A[k] \leftrightarrow A[r]$
- 9 devolva **PARTICIONE**(A, p, r)

Exercício 1 Mostre como modificar QUICKSORT de modo que tenha complexidade de tempo $\Theta(n \lg n)$ no **piores caso**.

Exercício 2 Suponha que você tenha uma subrotina do tipo “caixa-preta” que determina a mediana em **tempo linear** (no **piores caso**). Descreva um algoritmo linear simples que resolve o problema da seleção para todo i .

Exercício 3 Dado um conjunto de n números, queremos imprimir em ordem crescente os i maiores elementos deste usando um algoritmo baseado em comparações. Compare a complexidade dos seguinte métodos em função de n e i .

- ▶ Ordene o vetor e liste os i maiores elementos.
- ▶ Construa uma fila de prioridade (max-heap) e chame a rotina `EXTRACT-MAX` i vezes.
- ▶ Use um algoritmo de seleção para encontrar o i -ésimo maior elemento, particione o vetor em torno dele e ordene os i maiores elementos.