

Projeto e Análise de Algoritmos

Estatísticas de ordem

Lehilton Pedrosa

Primeiro Semestre de 2020

Estatísticas de Ordem

Estamos interessados no **problema da seleção**:

- ▶ **Entrada:** vetor A com n números reais e um inteiro i
- ▶ **Saída:** 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)

Estatísticas de Ordem

- ▶ Mínimo e máximo

```
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  $mín$ 
```

Análise:

- ▶ realiza $n - 1$ comparações
- ▶ não é possível fazer menos

Mínimo e máximo

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

Análise:

- ▶ realiza $2(n - 1)$ comparações
- ▶ é possível fazer melhor!

Mínimo e máximo (melhorado)

Ideia para melhorar algoritmo

- ▶ inicializamos *mín* e *máx*
 - ▶ com o primeiro elemento se n for ímpar
 - ▶ com o mínimo e o máximo dos dois primeiros se n for par
- ▶ comparamos os demais elementos em *pares*
 - ▶ o menor deles com *mín*
 - ▶ e o maior deles com *máx*
- ▶ fazemos 3 comparações para cada 2 elementos

Análise:

- ▶ o número de comparações é dado por

$$\begin{cases} 3\lfloor n/2 \rfloor & \text{se } n \text{ for ímpar} \\ 3\lfloor n/2 \rfloor - 2 & \text{se } n \text{ for par} \end{cases}$$

- ▶ não é possível fazer melhor (exercício de CLRS)

Estatísticas de Ordem

- ▶ Problema da seleção

Algoritmo para problema da seleção

Podemos selecionar o i -ésimo elemento ordenando o vetor.

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

Análise:

- ▶ usamos MERGE-SORT ou HEAP-SORT como ORDENE
- ▶ o algoritmo realiza $O(n \lg n)$ comparações

É possível fazer melhor que isso?

- ▶ achamos mínimo e máximo com $O(n)$ comparações
- ▶ vamos tentar descobrir o i -ésimo em **tempo linear**

Relembrando particionamento

Problema:

- ▶ rearranjar um vetor $A[p \dots r]$
- ▶ devolver um índice q , com $p \leq q \leq r$, tal 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

PARTICIONE(A, p, r)

```
1   $x \leftarrow A[r]$   $\triangleright$   $x$  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  $i + 1$ 
```

Selecionando via particionamento

Ideia:

- ▶ particionamos o vetor de forma que

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

- ▶ verificamos o índice q do pivô
 1. se $i = q$, então o i -ésimo menor é $A[q]$!
 2. se $i < q$, então o i -ésimo menor está em $A[1 \dots q - 1]$
 3. se $i > q$, então o i -ésimo menor está em $A[q + 1 \dots n]$

Algoritmo baseado em particionamento

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

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

- ▶ p e r são os índices de limite do vetor
- ▶ k é a posição relativa do pivô

Análise do algoritmo melhorado

SELECT-NL (A, p, r, i)	Tempo
1 se $p = r$ então	$\Theta(1)$
2 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 $i = k$ então	$\Theta(1)$
6 devolva $A[q]$	$O(1)$
7 senão se $i < k$ então	$O(1)$
8 devolva SELECT-NL ($A, p, q - 1, i$)	$T(k - 1)$
9 senão	$O(1)$
10 devolva SELECT-NL ($A, q + 1, r, i - k$)	$T(n - k)$

Seja $n := r - p + 1$

- ▶ no pior caso $T(n) = \max\{T(k - 1), T(n - k)\} + \Theta(n)$
- ▶ já sabemos que $T(n) = \Theta(n^2)$

Análise do algoritmo melhorado (cont)

Comparando

- ▶ sabemos que `SELECT-ORD` gasta tempo $O(n \log n)$
- ▶ mas, no pior caso, `SELECT-NL` tem complexidade $\Theta(n^2)$

Seria melhor usar `SELECT-ORD`?

- ▶ não, `SELECT-NL` é eficiente na prática
- ▶ no **caso médio**, ele tem complexidade $O(n)$

Versão aleatorizada

- ▶ o pior caso ocorre devido a escolhas infelizes do pivô
- ▶ podemos minimizar isso usando aleatoriedade
- ▶ vamos reutilizar **PARTICIONE-ALEATÓRIO**

PARTICIONE-ALEATÓRIO(A, p, r)

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

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

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

Algoritmo aleatorizado

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

Análise do tempo esperado

- ▶ Defina a seguinte variável indicadora

$$X_k = \begin{cases} 1 & \text{se } A[p \dots q] \text{ tem exatamente } k \text{ elementos} \\ 0 & \text{caso contrário} \end{cases}$$

- ▶ Podemos limitar o tempo de execução por

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 0, 1 \\ \sum_{k=1}^n X_k T(\max\{k-1, n-k\}) + \Theta(n) & \text{se } n \geq 2 \end{cases}$$

- ▶ Queremos calcular $E[T(n)]$

Análise do tempo esperado (cont)

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

pois

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

Análise do tempo esperado (cont)

Vamos demonstrar por indução que $E[T(n)] \leq cn$

$$\begin{aligned} E[T(n)] &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + an \\ &\stackrel{\text{h.i.}}{\leq} \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an \\ &= \frac{2c}{n} \left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k \right) + an \\ &= \frac{2c}{n} \left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2} \right) + an \end{aligned}$$

Análise do tempo esperado (cont)

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

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

Estatísticas de Ordem

- ▶ Algoritmo BFPRT

Algoritmo linear para seleção

Veremos um algoritmo linear para o problema da seleção.

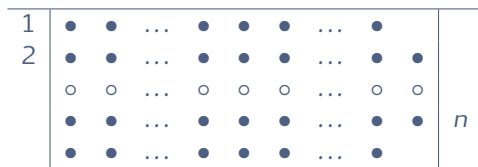
- ▶ chamaremos de **algoritmo BFPRT**
- ▶ devido aos autores Blum, Floyd, Pratt, Rivest e Tarjan
- ▶ vamos supor que os elementos em A são distintos

Algoritmo BFPRT

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



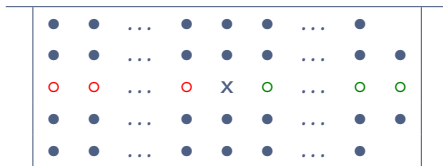
2. Encontre a mediana de cada um dos $\lfloor \frac{n}{5} \rfloor$ subconjuntos.



- ▶ na figura acima, cada subconjunto está em ordem crescente, de cima para baixo.

Algoritmo BFPRT (cont)

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



- ▶ a figura acima é a mesma que a anterior, mas com as colunas ordenadas pela mediana de cada grupo
- ▶ a ordem dos elementos em cada uma das colunas permanece inalterada
- ▶ por simplicidade de exposição, suponha que última coluna permanece no mesmo lugar.
- ▶ **note que o algoritmo não ordena as medianas!**

Algoritmo BFPRT (cont)

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

Se a posição final de x após o particionamento for k , então

$$|A_{<}| = k - 1 \quad \text{e} \quad |A_{>}| = n - k.$$

Algoritmo BFPRT (cont)

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

Análise do algoritmo BFPRT

Seja $T(n)$ a complexidade de tempo no pior caso.

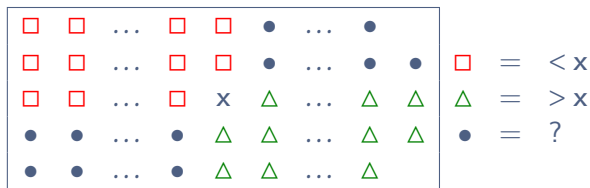
1. divisão em subconjuntos de 5 elementos $\Theta(n)$
2. encontrar a mediana de cada subconjunto $\Theta(n)$
3. encontrar x , a mediana das medianas $T(\lceil n/5 \rceil)$
4. Particionamento com pivô x . $O(n)$
5. ou encontrar o i -ésimo menor de $A_{<}$, $T(k-1)$
ou encontrar o $i-k$ -ésimo menor de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

Obtemos a recorrência

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

Análise do algoritmo BFPRT (cont)

O diagrama abaixo classifica os elementos da última figura.



Podemos contar o número de elementos maiores que x:

- ▶ contêm pelo menos os triângulos \triangle da figura
- ▶ há no mínimo $\frac{3n}{10} - 6$ tantos elementos
 - ▶ há $\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil$ grupos com 3 elementos maiores que x
 - ▶ exceto talvez o último e o que contém x
 - ▶ assim o número de elementos é $3 \left(\lceil \frac{1}{2} \lceil \frac{n}{5} \rceil \rceil - 2 \right) \geq \frac{3n}{10} - 6$.

Análise do algoritmo BFPRT (cont)

Repetindo o argumento para os quadrados \square

- ▶ há pelo menos $\frac{3n}{10} - 6$ elementos menores que x
- ▶ portanto

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

Encontramos uma recorrência limitada por

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

- ▶ o limiar 140 foi escolhido para as contas funcionarem
- ▶ a solução da recorrência é $T(n) \in \Theta(n)$

Análise do algoritmo BFPRT (cont)

Vamos demonstrar por indução que $T(n) \leq cn$

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

se $c \geq 20a$.

Pseudocódigo do algoritmo BFPRT

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

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

- ▶ devolve o índice do elemento, **não** o valor

Pseudocódigo do particionamento de 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   $k \leftarrow$  SELECT-BFPRT( $A, p, p + \lceil n/5 \rceil - 1, \lfloor (\lceil n/5 \rceil + 1)/2 \rfloor$ )  
  
8   $A[k] \leftrightarrow A[r]$   
9  devolva PARTICIONE( $A, p, r$ )
```