Quicksort e Select Aleatorizados

CLRS Secs 7.3, 7.4 e 9.2

Relembremos o Particione

Rearranja $A[p\,..\,r]$ de modo que $p \le q \le r$ e
$A[p\,..\,q{-}1] \le A[q] < A[q{+}1\,..\,r]$

```
PARTICIONE (A, p, r)
1  x ← A[r]        ▷ x é o "pivô"
2  i ← p−1
3  para j ← p até r − 1 faça
4      se A[j] ≤ x
5          então i ← i + 1
6                A[i] ↔ A[j]
7  A[i+1] ↔ A[r]
8  devolva i + 1
```

Invariantes: no começo de cada iteração de 3–6,

(i0) $A[p\,..\,i] \le x$ (i1) $A[i{+}1\,..\,j{-}1] > x$ (i2) $A[r] = x$

Consumo de tempo: $\Theta(n)$ onde $n := r - p$.

Quicksort aleatorizado

```
PARTICIONE-ALEA(A, p, r)
  i ← RANDOM(p, r)
  A[i] ↔ A[r]
  devolva PARTICIONE(A, p, r)

QuickSort-Ale (A, p, r)
  se p < r
      então q ← PARTICIONE-ALEA(A, p, r)
            QuickSort − Ale(A, p, q − 1)
            QuickSort − Ale(A, q + 1, r)
```


Como analizar?

Vamos estimar o tempo médio...do que, mesmo?

- Fixe uma permutação
- Considere as várias instâncias do QuickSort-Ale em cima dela, com a distribuição de probabilidade dada pelos sorteios ao longo da execução.
- Determine o tempo médio para essa permutação.

Quicksort aleatorizado

```
PARTICIONE-ALEA(A, p, r)
i ← RANDOM(p, r)
A[i] ↔ A[r]
devolva PARTICIONE(A, p, r)
```

```
QuickSort-Ale (A, p, r)
se p < r
    então q ← PARTICIONE-ALEA(A, p, r)
          QuickSort − Ale(A, p, q − 1)
          QuickSort − Ale(A, q + 1, r)
```

Consumo esperado de tempo?

Basta contar o número esperado de comparações na linha 4 do PARTICIONE.

Consumo esperado de tempo

Basta contar o número esperado de comparações na linha 4 do PARTICIONE.

```
PARTICIONE (A, p, r)
1  x ← A[r]          ▷ x é o "pivô"
2  i ← p−1
3  para j ← p até r − 1 faça
4      se A[j] ≤ x
5          então i ← i + 1
6                A[i] ↔ A[j]
7  A[i+1] ↔ A[r]
8  devolva i + 1
```


Consumo de tempo esperado

Suponha $A[p\,..\,r]$ permutação de $1\,..\,n$.

X_{ab} = número de comparações entre a e b na linha 4 do PARTICIONE do QuickSort-Ale;

Queremos calcular

$$\begin{aligned} X &= \text{total de comparações "}A[j] \le x\text{"} \\ &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{ab} \end{aligned}$$

Consumo de tempo esperado

Supondo $a < b$,

$$X_{ab} = \begin{cases} 1 & \text{se primeiro pivô em } \{a,\ldots,b\,|\,\} \text{ é } a \text{ ou } b \\ 0 & \text{caso contrário} \end{cases}$$

Qual a probabilidade de X_{ab} valer 1?

$$\Pr\{X_{ab}{=}1|\ \} = \frac{1}{b-a+1} + \frac{1}{b-a+1} = \mathrm{E}[X_{ab}]$$

$$X = \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} X_{ab}$$

$\mathrm{E}[X] = ????$

Consumo de tempo esperado

$$
\begin{aligned}
\mathrm{E}[X] &= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \mathrm{E}[X_{ab}] \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \Pr\{X_{ab}{=}1\} \\
&= \sum_{a=1}^{n-1} \sum_{b=a+1}^{n} \frac{2}{b-a+1} \\
&= \sum_{a=1}^{n-1} \sum_{k=1}^{n-a} \frac{2}{k+1} \\
&< 2 \sum_{a=1}^{n-1} \left(\tfrac{1}{2} + \cdots + \tfrac{1}{n}\right) \\
&< 2n\,(H_n - 1) \;<\; 2n \ln n
\end{aligned}
$$

Conclusões

O consumo de tempo esperado do algoritmo QuickSort-Ale é $\mathrm{O}(n \log n)$.

Do exercício 7.4-4 do CLRS temos que

O consumo de tempo esperado do algoritmo QuickSort-Ale é $\Theta(n \log n)$.

Prática

1. Eliminar recursão de cauda e começar pela “metade menor”, para economizar pilha.
2. Parar a recursão quando n chega num certo ponto ($n \approx 10$). No final, fazer uma passada do ORDENA-POR-INSERÇÃO.
3. TIMSORT. Algo parecido, mas baseado no MERGESORT. Usado no python. Não está em livros ainda, mas está na Wikipedia e no código do python, super-explicado.

k-ésimo menor elemento

CLRS 9

k-ésimo menor

Problema: Encontrar o k-ésimo menor elemento de $A[1\,..\,n]$.

Suponha $A[1\,..\,n]$ sem elementos repetidos.

Exemplo: 33 é o 4º menor elemento de:

1									10	
22	99	32	88	34	**33**	11	97	55	66	A

1			4						10	
11	22	32	**33**	34	55	66	88	97	99	ordenado

Mediana

Mediana é o $\left\lfloor \frac{n+1}{2} \right\rfloor$-ésimo menor ou o $\left\lceil \frac{n+1}{2} \right\rceil$-ésimo menor elemento.

Exemplo: a mediana é 34 ou 55:

1									10	
22	99	32	88	**34**	33	11	97	**55**	66	A

1				5	6				10	
11	22	32	33	**34**	**55**	66	88	97	99	ordenado

k-ésimo menor

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

SELECT-ORD (A, n, k)
ORDENE (A, n)
devolva $A[k]$

O consumo de tempo do SELECT-ORD é $\Theta(n \lg n)$.

Dá para fazer melhor?

Menor

Recebe um vetor $A[1\,..\,n]$ e devolve o valor do menor elemento.

MENOR (A, n)
menor $\leftarrow A[1]$
para $k \leftarrow 2$ **até** n **faça**
  se $A[k] <$ menor
    então menor $\leftarrow A[k]$
devolva menor

O consumo de tempo do algoritmo MENOR é $\Theta(n)$.

Segundo menor

Recebe um vetor $A[1\,..\,n]$ e devolve o valor do segundo menor elemento, supondo $n \geq 2$.

```
SEG-MENOR (A, n)
  menor ← min{A[1], A[2]}     segmenor ← max{A[1], A[2]}
  para k ← 3 até n faça
      se A[k] < menor
          então segmenor ← menor
                menor ← A[k]
      senão se A[k] < segmenor
          então segmenor ← A[k]
  devolva segmenor
```

O consumo de tempo do SEG-MENOR é $\Theta(n)$.

Algoritmo linear?

Será que conseguimos fazer um algoritmo linear

para a mediana?

para o k-ésimo menor?

Sim!

Usaremos o PARTICIONE do QUICKSORT!

Select aleatorizado

```
PARTICIONE-ALEA(A, p, r)
  k ← RANDOM (p, r)
  A[k] ↔ A[r]
  devolva PARTICIONE (A, p, r)
```

```
SELECT-ALEA (A, p, r, k)
  se p = r então devolva A[p]
  q ← PARTICIONE-ALEA (A, p, r)
  se k = q − p + 1
      então devolva A[q]
  se k < q − p + 1
      então devolva SELECT-ALEA (A, p, q − 1, k)
      senão devolva SELECT-ALEA (A, q+1, r, k−(q−p+1))
```

Consumo esperado de tempo?

$X =$ número de comparações no PARTICIONE-ALEA

Quebrar em fases

KT 13.5

Uma fase é uma sucessão de chamadas recursivas

a fase muda quando o tamanho do vetor cai de pelo menos 3/4

Na fase k, o tamanho do vetor é $\leq \left(\frac{3}{4}\right)^k n$

Se X_k é o número de chamadas da fase k, vale

$$X \leq \sum_{k \geq 0} X_k \left(\frac{3}{4}\right)^k n$$

Logo,

$$E(X) \leq n \sum_{k \geq 0} E(X_k) \left(\frac{3}{4}\right)^k$$

Com pivô sortudo

Suponha que cada pivô caia na faixa boa

25%	50%	25%

a chamada recursiva faz mudar de fase.
Difícil dar sorte sempre, mas quanto tempo precisa esperar?

Um pouco mais de probabilidade

Considere uma sequência de sorteios de BOM ou MAU, onde a probabilidade de BOM é $p > 0$ a cada sorteio.

quantos sorteios em média para aparecer o primeiro BOM?

X = instante do primeiro sorteio favorável

$$\Pr[X = j] = (1-p)^{j-1} p$$

$$E(X) = \sum_{j \geq 1} j \Pr[X = j] = \sum_{j \geq 1} j (1-p)^{j-1} p$$

$$\leq p \sum_{j=1}^{\infty} j (1-p)^{j-1} = \frac{1}{p}$$

Terminando

Para um pivô cair na faixa boa, $p = 1/2$. Logo, o número esperado de sorteios para que o tamanho do vetor caia por $3/4$ é ≤ 2.
Da análise anterior,

$$E(X) \leq n \sum_{k \geq 0} E(X_k) \left(\frac{3}{4}\right)^k$$

$$\leq 2n \sum_{k \geq 0} \left(\frac{3}{4}\right)^k$$

$$\leq 8n$$

Conclusões

O consumo de tempo esperado do algoritmo SELECT-ALEA é $O(n)$.

Próxima aula

CLRS Cap 6 Heapsort