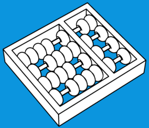
Introdução a algoritmos aleatorizados e ordenação por particionamento

MO417 - Complexidade de Algoritmos I

10

04/24



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP

“Teoria de probabilidades não é mais que sentido comum reduzido a cálculo.”

Pierre-Simon Laplace.

Revisão de probabilidade

Espaço amostral

Consideramos a noção de **espaço amostral** S:

- Conjunto cujos elementos são eventos elementares.
- Cada elemento é o resultado de um experimento.

Estamos interessados em investigar **eventos**:

- Um subconjunto do espaço amostral S.
- O subconjunto vazio $\emptyset$ é o evento nulo.
- O conjunto completo S é o evento certo.
- Eventos disjuntos são **mutuamente exclusivos**.
- Conjuntos unitários são mutuamente exclusivos.

Probabilidade

Uma **distribuição de probabilidade** $\Pr\{\,\}$ em S é uma função mapeando eventos em reais satisfazendo axiomas:

1. $\Pr\{A\} \geq 0$ para todo evento A.
2. $\Pr\{S\} = 1$.
3. $\Pr\{A \cup B\} = \Pr\{A\} + \Pr\{B\}$ para eventos mutuamente exclusivos A e B.

Chamamos $\Pr\{A\}$ a probabilidade do evento A:

- O evento complementar de A é $\overline{A} = S \setminus A$.
- Assim, $\Pr\{\overline{A}\} = 1 - \Pr\{A\}$.

Propriedades

Algumas consequências da definição:

1. Se $A_1, A_2, \ldots,$ são eventos mutuamente exclusivos
$$\Pr\left\{\bigcup_i A_i\right\} = \sum_i \Pr\{A_i\}.$$
2. Para quaisquer dois eventos A, B:
$$\Pr\{A \cup B\} = \Pr\{A\} + \Pr\{B\} - \Pr\{A \cap B\}.$$
3. E portanto:
$$\Pr\{A \cup B\} \leq \Pr\{A\} + \Pr\{B\}.$$

Distribuições discretas

Vamos considerar espaços amostrais **discretos**.

- Para qualquer evento: $\Pr\{A\} = \sum_{s} \Pr\{s\}$.
- Em uma distribuição **uniforme** em S, temos: $\Pr\{s\} = 1/|S|$

Exemplo: Uma moeda não viciada.

- Espaço amostral $S = \{H, T\}$ (cara ou coroa).
- Temos $\Pr\{H\} = \Pr\{T\} = 1/|S| = 1/2$.

Exemplo: Uma sequência de n lançamentos de moeda.

- O espaço amostral são strings com H ou T de tamanho n.
- Considere um evento
  $A = \{\text{exatamente } k \text{ caras e } n - k \text{ coroas}\}$.
- Então $\Pr\{A\} = \binom{n}{k}/2^n$.

Probabilidade condicional e independência

A **probabilidade condicional** de um evento A dado outro B é:

$$\Pr\{A|B\} = \frac{\Pr\{A \cap B\}}{\Pr\{B\}}.$$

Dois eventos A e B são **independentes** se:

$$\Pr\{A \cap B\} = \Pr\{A\}\Pr\{B\}.$$

Variável aleatória

Uma **variável aleatória** X discreta é uma função que associa cada elemento do espaço amostral a uma valor real.

- Dado um número real x estamos interessados na probabilidade de que X tenha valor x.
- Temos que considerar eventos s tal que que $X(s) = x$.
- Esse evento tem a seguinte probabilidade:

$$\Pr\{X = x\} = \sum_{s \in S : X(s) = x} \Pr\{s\}.$$

Esperança

O **valor esperado** ou **esperança** de uma variável aleatória X é:

$$E[X] = \sum_x x \cdot \Pr\{X = x\}.$$

A esperança satisfaz propriedades de linearidades:

- Para variáveis aleatórias X e Y:

$$E[X + Y] = E[X] + E[Y]$$

- Para uma variável aleatória X e real a:

$$E[aX] = aE[X]$$

Variáveis indicadoras

A variável indicadora de um evento A é a variável aleatória:

$$I\{A\} = \begin{cases} 1 & \text{se evento } A \text{ ocorreu} \\ 0 & \text{se evento } A \text{ não ocorreu} \end{cases}$$

Exemplo: quantas caras esperamos ao lançarmos n moedas?

- Seja X_i a variável indicadora do evento **"o i-ésimo lançamento deu cara"**.
- Seja X o número de caras sorteadas.
- Assim, o número de caras esperado é:

$$E[X] = E\left[\sum_{i=1}^{n} X_i\right] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} 1/2 = n/2.$$

- Note que $E[X_i] = \Pr\{X_1 = 1\} \cdot 1 + \Pr\{X_1 = 0\} \cdot 0 = 1/2$.

Série Harmônica

- A série harmônica é:

$$H_n = \sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}.$$

- Esta série é limitada superiormente da seguinte forma:

$$H_n \leq \ln n + 1.$$

Análise probabilística e algoritmo aleatorizado

O problema do assistente

Suponha que queremos contratar um assistente:

- Vamos sempre entrevistar n candidatos.
- Entrevistar um candidato tem um custo pequeno c_i.
- Contratar um candidato tem um custo elevado c_h.

Como garantir que teremos contratado o melhor assistente?

Um algoritmo aleatorizado

Se estivermos determinados a contratar o melhor:

Algoritmo: CONTRATAR-ASSISTENTE(n)

```
melhor ← 0        ▷ candidato 0 é um candidato dummy
para i ← 1 até n
    entreviste candidato i
    se candidato i é melhor que melhor
        melhor ← i
        contrate candidato i
```

Custo dessa estratégia

Quanto irá custar contratar de acordo com esse algoritmo?

- Sempre entrevistamos n candidatos.
- Suponha que contratamos m assistentes.
- O custo associado ao algoritmo é $c_i n + c_h m$.
- O custo varia com a ordem das entrevistas.

Análise probabilística

Tradicionalmente, analisamos o **pior caso**:

- Qual o maior valor que pode ser gasto?
- Em um pior caso, contratamos todos os assistentes.
- Assim o custo é $O(c_h n)$.

Vamos usar **probabilidade** para fazer outros tipos de análise.

- **Tempo de execução esperado:**
  - Quanto tempo esperamos gastar?
  - A entrada é uma variável aleatória.
  - Precisamos da distribuição de probabilidade da entrada.
- **Tempo de execução médio:**
  - Qual o tempo médio entre todas as entradas?
  - Supomos uma distribuição uniforme.
  - Então, cada permutação tem a mesma probabilidade.

Análise de Contratar-Assistente

Começamos definindo uma variável indicadora:

$$X_i = \begin{cases} 1 & \text{se o candidato } i \text{ é contratado} \\ 0 & \text{se o candidato } i \text{ não é contratado} \end{cases}$$

Precisamos calcular:

$$E[X_i] = \Pr\{\text{candidato } i \text{ ser contratado}\}.$$

- O candidatos i é contratado na linha 6 do algoritmo.
- Isso acontece se i é melhor entre candidatos $1, 2, \ldots, i$.
- Como supomos uma distribuição, cada um entre eles tem a mesma chance ser o melhor.
- Assim $E[X_i] = \Pr\{\text{candidato } i \text{ ser contratado}\} = 1/i$.

Análise de Contratar-Assistente (cont)

Agora podemos estimar o custo esperado.

- Seja X a variável que guarda o número de contratações.
- Então, $X = \sum_{i=1}^{n} X_i$.

$$
\begin{aligned}
E[X] &= E\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} E[X_i] \\
&= \sum_{i=1}^{n} 1/i \\
&= \ln n + O(1).
\end{aligned}
$$

Assim, o custo médio de contratação é $O(c_h \ln n)$!

Algoritmos aleatorizados

Algumas considerações sobre a análise anterior:

- Entrevistamos na ordem dada.
- Presumimos que cada ordem tem a mesma probabilidade.
- Isso nem sempre é razoável.

Podemos garantir essa propriedade de outra forma:

- Primeiro recebemos toda a lista de candidatos.
- Depois permutamos **aleatoriamente** essa lista.
- Garantimos que cada ordem tem a mesma probabilidade.

Algoritmos aleatorizados:

- Executam uma ou mais instruções aleatórias.
- Para uma entrada fixa, pode haver diferentes execuções.
- O tempo de execução é em si uma variável aleatória.

Versão aleatorizada de CONTRATAR-ASSISTENTE

Algoritmo: CONTRATAR-ASSISTENTE-ALEATORIZADO(n)

```
1  permute aleatoriamente a lista dos candidatos
2  melhor ← 0      ▷ candidato 0 é um candidato dummy
3  para i ← 1 até n
4      entreviste candidato i
5      se candidato i é melhor que melhor
6          melhor ← i
7          contrate candidato i
```

O custo esperado de contratação é $O(c_h \ln n)$!

- Após a linha 1, cada ordem aparece com a mesma chance.
- Temos uma situação completamente análoga à da análise de CONTRATAR-ASSISTENTE.

QuickSort

QuickSort

QUICK-SORT é baseado em **divisão e conquista**.

Divisão:

- Divida em subvetores $A[p \dots q-1]$ e $A[q+1 \dots r]$.
- Tais que $\mathbf{A[p \dots q-1]} \leq A[q] < \mathbf{A[q+1 \dots r]}$.

	p	q	r
A	$\leq \mathbf{x}$	$\mathbf{x}$	$> \mathbf{x}$

Conquista:

- Ordene os subvetores recursivamente.

Combinação:

- Nada a fazer, o vetor está ordenado.

Problema do particionamento

Problema

Entrada: *Um vetor* $A[p \ldots r]$
Saída: *Um índice* q*, com* $p \leq q \leq r$*, tal que*

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

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

Particionando

Ideia:

- Pivô x será o último elemento em $A[r]$.
- Índices i e j definem dois subvetores.

$$\mathbf{A[p \ldots i] \leq x} \quad \text{e} \quad \mathbf{A[i+1 \ldots j-1] > x}$$

- Varremos com j da primeira até a penúltima posição.
  - Se $A[j] \leq x$, trocamos $A[i+1]$ com $A[j]$ e incrementamos i.
  - Se $A[j] > x$, apenas continuamos.

Algoritmo PARTICIONE

Algoritmo: 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** $r - 1$
4   **se** $A[j] \leq x$
5         $i \leftarrow i + 1$
6         $A[i] \leftrightarrow A[j]$
7 $A[i{+}1] \leftrightarrow A[r]$
8 **devolva** $i + 1$

Teorema (Invariantes)

No começo de cada iteração da linha 3 vale:

1. $A[p \ldots i] \leq x$.
2. $A[i{+}1 \ldots j - 1] > x$.
3. $A[r] = x$.

Complexidade de Particione

Particione(A, p, r)	Tempo
$x \leftarrow A[r]$ ▷ x é o pivô	$\Theta(1)$
$i \leftarrow p - 1$	$\Theta(1)$
para $j \leftarrow p$ **até** $r - 1$	$\Theta(n)$
se $A[j] \leq x$	$\Theta(n)$
$i \leftarrow i + 1$	$O(n)$
$A[i] \leftrightarrow A[j]$	$O(n)$
$A[i+1] \leftrightarrow A[r]$	$\Theta(1)$
devolva $i + 1$	$\Theta(1)$

Se $n := r - p + 1$, então a complexidade no pior caso é

$$T(n) = \Theta(2n + 4) + O(2n) = \Theta(n).$$

QuickSort

Algoritmo: $\textsc{Quick-Sort}(A, p, r)$

se $p < r$
- $q \leftarrow \textsc{Particione}(A, p, r)$
- $\textsc{Quick-Sort}(A, p, q-1)$
- $\textsc{Quick-Sort}(A, q+1, r)$

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

QuickSort

Algoritmo: $\textsc{Quick-Sort}(A, p, r)$

---

se $p < r$
- $\underline{q \leftarrow \textsc{Particione}(A, p, r)}$
- $\textsc{Quick-Sort}(A, p, q-1)$
- $\textsc{Quick-Sort}(A, q+1, r)$

---

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

QuickSort

Algoritmo: $\textsc{Quick-Sort}(A, p, r)$

```
se p < r
    q ← PARTICIONE(A, p, r)
    QUICK-SORT(A, p, q − 1)
    QUICK-SORT(A, q + 1, r)
```

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

QuickSort

Algoritmo: $\textsc{Quick-Sort}(A, p, r)$

```
se p < r
    q ← PARTICIONE(A, p, r)
    QUICK-SORT(A, p, q − 1)
    QUICK-SORT(A, q + 1, r)
```

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

Complexidade de QUICK-SORT

QUICK-SORT(A, p, r)	Tempo
se $p < r$	$\Theta(1)$
$q \leftarrow$ PARTICIONE(A, p, r)	$\Theta(n)$
QUICK-SORT$(A, p, q-1)$	$T(k)$
QUICK-SORT$(A, q+1, r)$	$T(n-k-1)$

Seja $n := r - p + 1$ o número total de elementos:

- Desses, $k := q - p$ são "pequenos".
- Enquanto $n - k - 1$ são "grandes".

O tempo de execução é:

$$T(n) = T(k) + T(n-k-1) + \Theta(n).$$

Recorrência

Encontramos uma recorrência:

$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 0, 1 \\ T(k) + T(n-k-1) + \Theta(n) & \text{se } n = 2, 3, \ldots \end{cases}$$

- O parâmetro da recorrência é apenas n.
- O valor de k depende do caso.

Limitando inferiormente o pior caso

Se o vetor estiver ordenado, então:

$$T(n) = T(n-1) + T(0) + \Theta(n).$$

Resolvendo para esse caso, obtemos $T(n) = \Theta(n^2)$.

Limitando superiormente o melhor caso

Se Particione sempre dividir em duas partes "iguais", temos:

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil - 1) + \Theta(n).$$

Simplificando para:

$$T(n) = 2T(n/2) + \Theta(n).$$

Obtemos $T(n) = \Theta(n \log n)$.

Algumas conclusões

Até agora, concluímos que Quick-Sort:

- No pior caso, leva tempo pelo menos $\Omega(n^2)$.
- No melhor caso, leva tempo no máximo $O(n \log n)$.

Mas na média, qual dos dois limitantes está mais próximo do comportamento típico do algoritmo?

Caso médio

Na média, QUICK-SORT leva tempo $O(n \log n)$!

- Casos ruins ocorrem quando os subvetores obtidos por PARTICIONE são **desbalanceados**:
  - Muitos poucos elementos pequenos e muitos grandes.
  - Ou vice-versa.
- Na maioria das instâncias, cada subvetor obtido tem alguma fração constante dos elementos do vetor.

Podemos ilustrar esse fato assim:

- Suponha que sempre dividimos na proporção $\frac{1}{10}$ para $\frac{9}{10}$.
- A recorrência seria da forma:

$$T(n) = T(\left\lfloor \frac{n-1}{10} \right\rfloor) + T(\left\lceil \frac{9(n-1)}{10} \right\rceil) + \Theta(n)$$

- Cuja a solução é $T(n) = \Theta(n \log n)$ (faça como **exercício**).

Quick-Sort aleatorizado

- O pior caso ocorre devido a escolhas infelizes do pivô.
- Podemos minimizar isso usando aleatoriedade.
- Criamos uma versão de Particione aleatorizada.

Algoritmo: Particione-Aleatorizado(A, p, r)

$i \leftarrow$ Random(p, r)
$A[i] \leftrightarrow A[r]$
devolva Particione(A, p, r)

Algoritmo: Quick-Sort-Aleatorizado(A, p, r)

se $p < r$
    $q \leftarrow$ Particione-Aleatorizado(A, p, r)
    Quick-Sort-Aleatorizado$(A, p, q - 1)$
    Quick-Sort-Aleatorizado$(A, q + 1, r)$

Tempo esperado de Quick-Sort-Aleatorizado

Começamos com algumas definições:

- Vamos supor que todos os elementos são distintos.
- Sejam $z_1 < z_2 < \ldots < z_n$ esses elementos ordenados.
- Seja $Z_{ij} = \{z_i, z_{i+1}, \ldots, z_j\}$.

Queremos estimar o número esperado de comparações.

- Seja X_{ij} a variável que diz se z_i foi comparado com z_j:

$$X_{ij} = \begin{cases} 1 & \text{se } z_i \text{ foi comparado com } z_j \\ 0 & \text{caso contrário.} \end{cases}$$

- Então, o número total de comparações X é:

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}.$$

- Nosso objetivo é calcular $E[X]$.

Tempo esperado de QUICK-SORT-ALEATORIZADO

Pela linearidade da esperança:

$$E[X] = E[\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}].$$

Como X_{ij} é uma variável indicadora:

$$E[X_{ij}] = Pr\{z_i \text{ ser comparado com } z_j\}.$$

Comparação de dois elementos.

Considere a escolha do pivô e a comparação entre z_i e z_j:

$$\underbrace{z_1, z_2, \ldots, z_{i-1}}_{\text{posterga}}, \mathbf{z_i}, \underbrace{z_{i+1}, \ldots, z_{j-1}}_{\text{não comp.}}, \mathbf{z_j}, \underbrace{z_{j+1}, \ldots, z_n}_{\text{posterga}}.$$

Assim,

$$\begin{aligned}
& Pr\{z_i \text{ ser comparado com } z_j\} \\
= & Pr\{z_i \text{ ou } z_j \text{ ser escolhido como pivô primeiro em } Z_{ij}\} \\
= & Pr\{z_i \text{ ser escolhido como pivô primeiro em } Z_{ij}\} + \\
& Pr\{z_j \text{ ser escolhido como pivô primeiro em } Z_{ij}\} \\
= & \frac{1}{j-i+1} + \frac{1}{j-i+1} = \frac{2}{j-i+1}.
\end{aligned}$$

Tempo esperado de QUICK-SORT-ALEATORIZADO

Portanto,

$$\begin{aligned}
E[X] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} \\
&< \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k} \\
&= \sum_{i=1}^{n-1} O(\log n) \\
&= O(n \log n).
\end{aligned}$$

Conclusão

O consumo de tempo esperado pelo QUICK-SORT-ALEATORIZADO para **itens distintos** é $O(n \log n)$.

Algumas observações:

1. Qual o tempo esperado de QUICK-SORT-ALEATORIZADO quando todos os elementos são iguais?

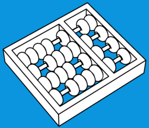
2. Podemos modificar QUICK-SORT-ALEATORIZADO para executar em tempo esperado $O(n \log n)$ quando há elementos iguais.
   - Modificamos PARTICIONE-ALEATORIZADO para dividir o vetor em três partes: menores, iguais e maiores que o pivô.
   - Faça isso como **exercício**.

Introdução a algoritmos aleatorizados e ordenação por particionamento

MO417 - Complexidade de Algoritmos I

04/24

10



Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP