Projeto e Análise de Algoritmos

Introdução a algoritmos aleatorizados e ordenação por particionamento

Lehilton Pedrosa

Primeiro Semestre de 2020

Revisão de probabilidade

Espaço amostral

Consideramos a noção de espaço amostral S

- conjunto cujos elementos são eventos elementares
- cada elementos é o resultado de um experimento

Estamos interssados em investigar eventos

- é 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

- não há nada de especial com $\Pr\{S\} = 1$
- o evento complementar de A é $\bar{A} = S \setminus A$
- assim, $\Pr\{\bar{A}\} = 1 - \Pr\{A\}$

Propriedades

Algumas conssequências 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
- Se um evento é $A = \{\text{exatamante } k \text{ caras e } n-k \text{ coroas}\}$
- 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 associada é uma função que associa cada elemento do espaço amostral a uma valor real.

- para um número real x podemos definir o evento em algum elemnto s ocorra 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

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

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
- mas 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:

```
ContratarAssistente(n)
1   melhor ← 0    ▷ candidato 0 é um candidato dummy
2   para i ← 1 até n faça
3       entreviste candidato i
4       se candidato i é melhor que melhor
5           então melhor ← i
6                   contrate candidato i
```

Custo dessa estratégia

Quanto irá custar contratar de acordo com esse algoritmo?

- sempre entrevistamos n candidatos
- suponha que contratamos m assistentes
- assim 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?
  - analogamente, supomos uma distribuição uniforme
  - então cada permutação tem a mesma probabilidade

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

Análise de ContratarAssistente

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 ContratarAssistente (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

- executamos 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 ContratarAssistente

```
ContratarAssistenteAleatorizado(n)
1  permute aleatoriamente a lista dos candidatos
2  melhor ← 0    ▷ candidato 0 é um candidato dummy
3  para i ← 1 até n faça
4      entreviste candidato i
5      se candidato i é melhor que melhor
6          então 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 ContratarAssistente

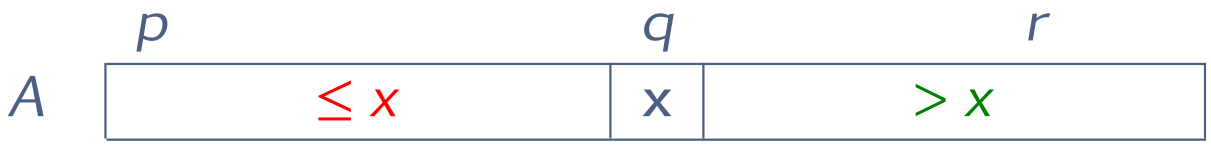
Quick Sort

QuickSort

QuickSort é baseado em divisão e conquista.

Divisão:

- divida em subvetores $A[p \ldots q-1]$ e $A[q+1 \ldots r]$
- tais que $A[p \ldots q-1] \leq A[q] < A[q+1 \ldots r]$



Conquista:

- ordene os subvetores recursivamente

Combinação:

- nada a fazer, o vetor está ordenado.

Problema do particionamento

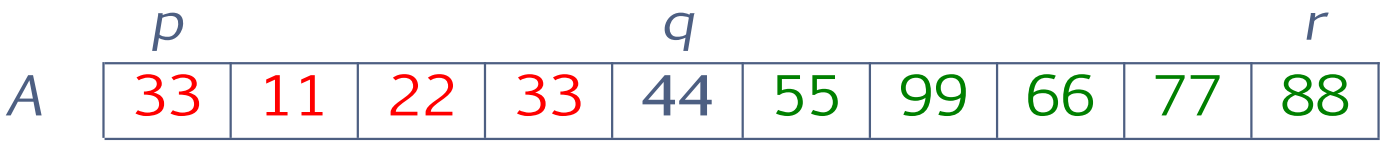
Problema:

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

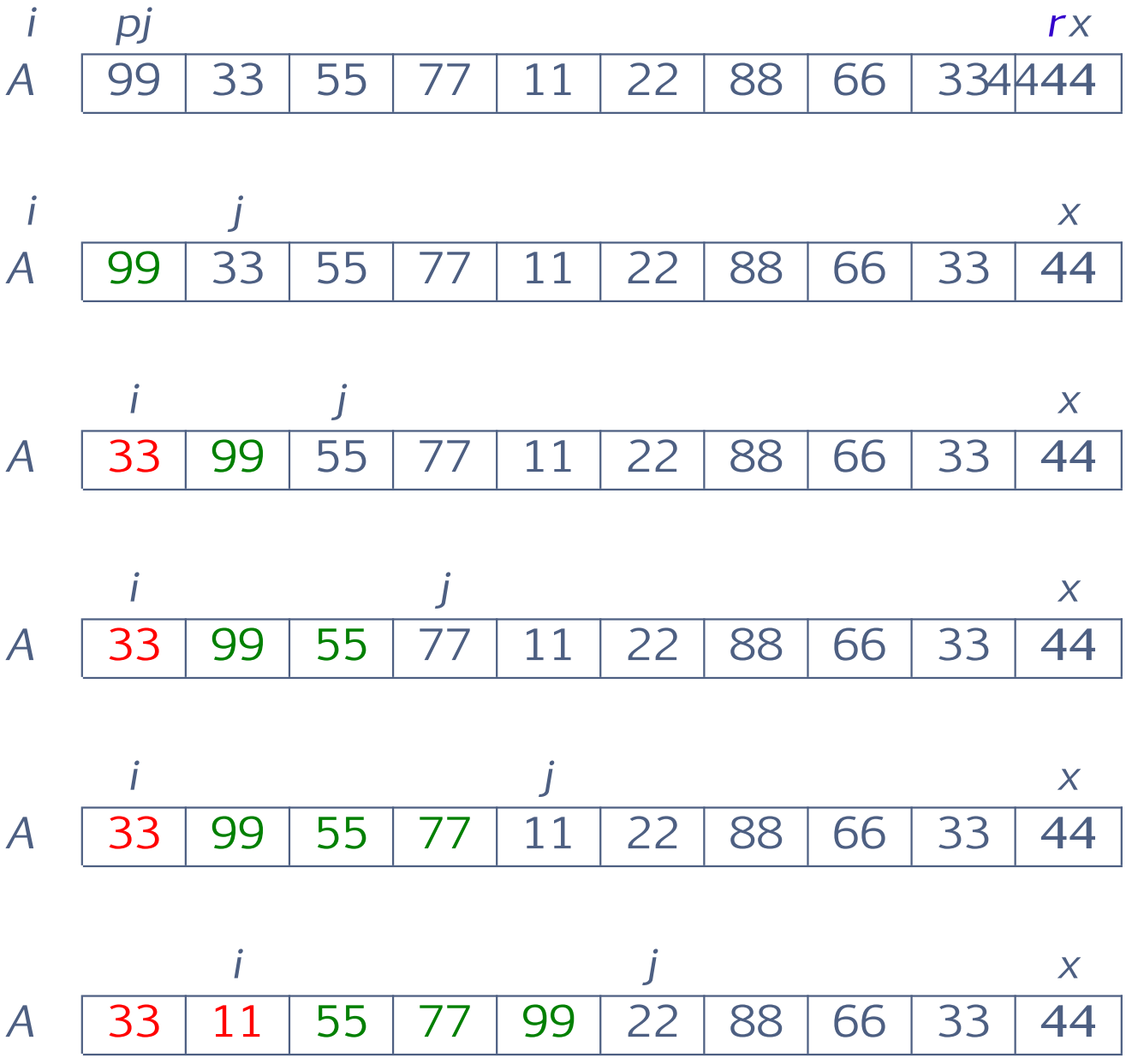
$$A[p \ldots q-1] \leq A[q] < A[q+1 \ldots r]$$

Entrada:

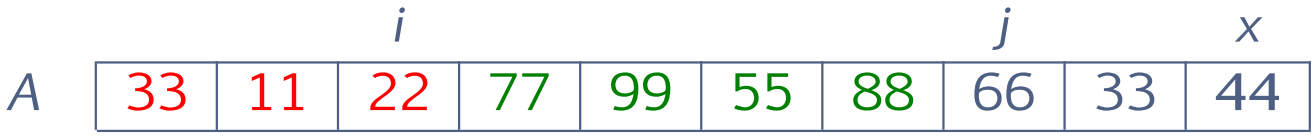
Saída:



Particionando



Particionando



Algoritmo 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
```

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
1	$x \leftarrow A[r]$ $\quad \triangleright\ x$ é o pivô	$\Theta(1)$
2	$i \leftarrow p-1$	$\Theta(1)$
3	**para** $j \leftarrow p$ **até** $r-1$ **faça**	$\Theta(n)$
4	$\quad$ **se** $A[j] \leq x$	$\Theta(n)$
5	$\quad\quad$ **então** $i \leftarrow i+1$	$O(n)$
6	$\quad\quad\quad A[i] \leftrightarrow A[j]$	$O(n)$
7	$A[i+1] \leftrightarrow A[r]$	$\Theta(1)$
8	**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

```
QuickSort(A, p, r)
1   se p < r
2       então q ← Particione(A, p, r)
3             QuickSort(A, p, q − 1)
4             QuickSort(A, q + 1, r)
```

QuickSort

```
QuickSort(A, p, r)
1   se p < r
2       então q ← Particione(A, p, r)
3             QuickSort(A, p, q − 1)
4             QuickSort(A, q + 1, r)
```

QuickSort

```
QuickSort(A, p, r)
1   se p < r
2       então q ← Particione(A, p, r)
3             QuickSort(A, p, q − 1)
4             QuickSort(A, q + 1, r)
```

QuickSort

```
QuickSort(A, p, r)
1   se p < r
2       então q ← Particione(A, p, r)
3             QuickSort(A, p, q − 1)
4             QuickSort(A, q + 1, r)
```

Complexidade de QuickSort

QuickSort(A,p,r)		Tempo
1	**se** $p < r$	$\Theta(1)$
2	**então** $q \leftarrow$ Particione(A,p,r)	$\Theta(n)$
3	QuickSort$(A,p,q-1)$	$T(k)$
4	QuickSort$(A,q+1,r)$	$T(n-k-1)$

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

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

Se o vetor estiver ordenado, então

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

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

Limitando o pior caso

Podemos limitar o pior caso com o seguinte limitante:

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

Queremos mostrar que $T(n) = \Theta(n^2)$.

Demonstrando $T(n) = O(n^2)$

Vamos mostrar $T(n) \leq cn^2$ por indução em n (grande).

$$
\begin{aligned}
T(n) &= \max_{0 \leq k \leq n-1} \left\{ T(k) + T(n-k-1) \right\} + bn \\
&\leq \max_{0 \leq k \leq n-1} \left\{ ck^2 + c(n-k-1)^2 \right\} + bn \\
&= c \max_{0 \leq k \leq n-1} \left\{ k^2 + (n-k-1)^2 \right\} + bn \\
&= c(n-1)^2 + bn && \text{(exercício)} \\
&= cn^2 - 2cn + c + bn \\
&\leq cn^2 ,
\end{aligned}
$$

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

Limitando o melhor caso

A complexidade de melhor caso pode ser limitada por:

$$M(n) = \begin{cases} \Theta(1) & \text{se } n = 0, 1 \\ \min\limits_{0 \le k \le n-1} \left\{ M(k) + M(n-k-1) \right\} + \Theta(n) & \text{se } n = 2, 3, \ldots \end{cases}$$

A expressão é minimizada quando $k \approx (n-1)/2$.

- assim podemos resolver uma recorrência

$$R(n) = R(\left\lfloor \frac{n-1}{2} \right\rfloor) + R(\left\lceil \frac{n-1}{2} \right\rceil) + \Theta(n)$$

- resolvendo, obtemos $R(n) = \Omega(n \lg n)$.
- pode-se dar um exemplo em que isso sempre acontece

Algumas conclusões

Até agora, concluímos que QuickSort

- no pior caso, leva tempo $\Theta(n^2)$
- no melhor caso, leva tempo é $\Theta(n \lg n)$

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

Caso médio

Ná média, QuickSort leva tempo $O(n \lg n)$

- 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

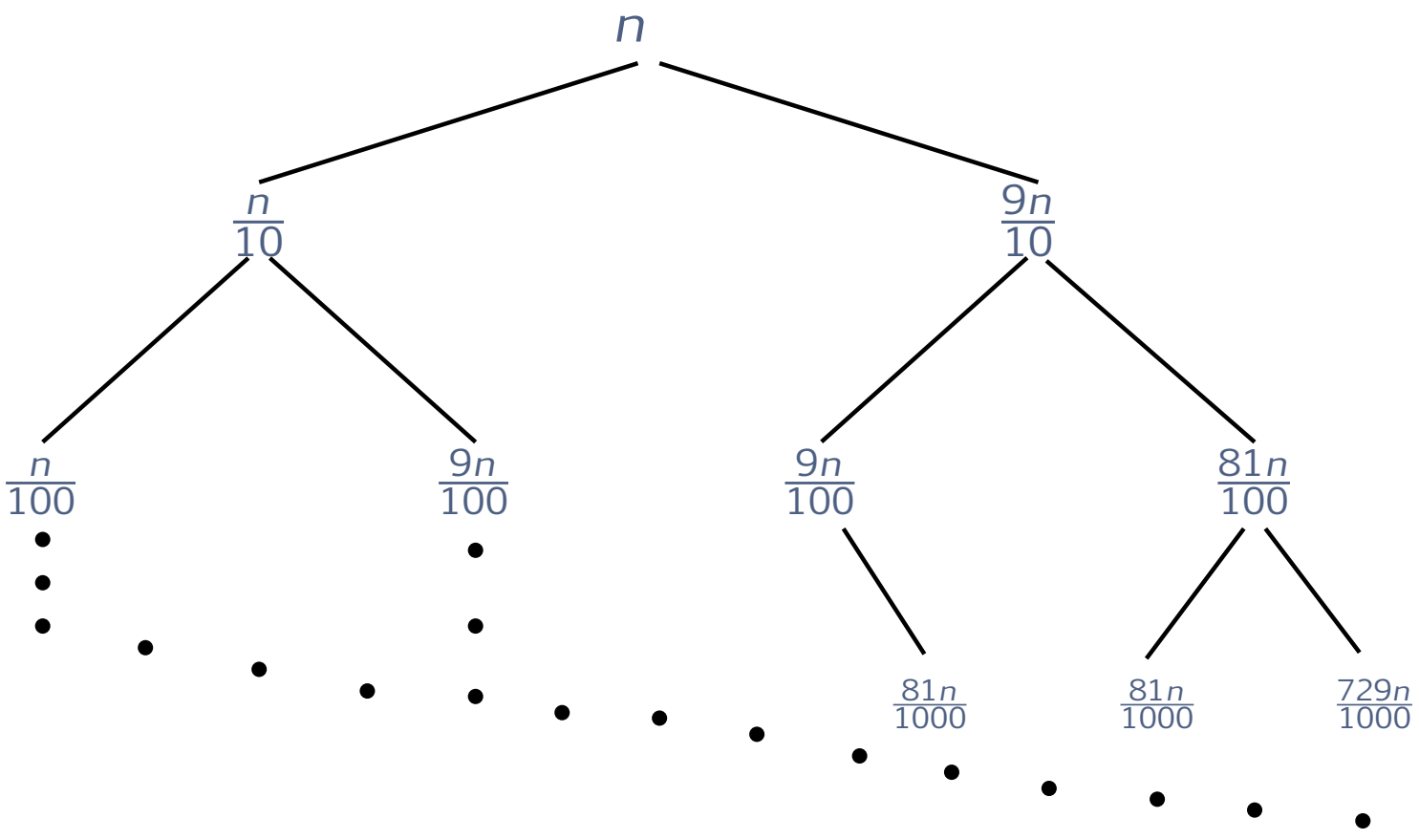
Podemos ilustrar esse fato assim:

- suponha que sempre dividimos na proporção $\frac{1}{9}$ para $\frac{9}{10}$
- a recorrência seria da forma

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

- cuja a solução é $T(n) = \Theta(n \lg n)$.

Árvore de recorrência



- o número de níveis é $\leq \log_{10/9} n$
- em cada nível o custo é $\leq n$
- então, o custo total é $O(n \log n)$

QuickSort aleatorizado

- o pior caso ocorre devido a escolhas infelizes do pivô
- podemos minimizar isso usando aleatoriedade
- criamos uma versão de Particione aleatorizada

```
PARTICIONE-ALEATÓRIO(A, p, r)
1   i ← RANDOM(p, r)
2   A[i] ↔ A[r]
3   devolva PARTICIONE(A, p, r)
```

```
QUICKSORT-ALEATÓRIO(A, p, r)
1   se p < r
2       então q ← PARTICIONE-ALEATÓRIO(A, p, r)
3              QUICKSORT-ALEATÓRIO(A, p, q − 1)
4              QUICKSORT-ALEATÓRIO(A, q + 1, r)
```

Tempo esperado de QuickSort-Aleatório

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 QuickSort-Aleatório (cont)

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:

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

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 QuickSort-Aleatório (cont)

Portanto,

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

Conclusão

O consumo de tempo esperado pelo QuickSort-Aleatório para itens distintos é $O(n \lg n)$.

Algumas observações:

1. qual o tempo esperado de QuickSort-Aleatório quando todos os elementos são iguais?

2. podemos morificar QuickSort-Aleatório para executar em tempo esperado $O(n \lg n)$ quando há elementos iguais.

   - modificamos Particione-Aleatório para dividir o vetor em três partes: menores, iguais e maiores que o pivô
   - faça isso como exercício