

5

Skip Treap and Streaming

5.1 Skip list

Uma *skip list* (Pugh, 1989) é uma estrutura de dados para representar conjuntos dinâmicos que usa aleatoriedade para obter, em tempo esperado, desempenho comparável ao de uma árvore de busca balanceada¹, com a vantagem de ser mais simples de implementar e de manter.

Numa *skip list* mantemos um conjunto S de n elementos distintos de um universo U dotado de ordem total. Os elementos de S são mantidos em uma lista ligada ordenada, a *lista do nível 0*, denotada S_0 ($S_0 = S$). Dada a lista S_i do nível $i \geq 0$, a *lista do nível $i + 1$* , denotada S_{i+1} , é um subconjunto aleatório de S_i no qual cada elemento de S_i é escolhido com probabilidade $1/2$, com as escolhas independentes. A lista S_{i+1} é ordenada e cada elemento seu tem um apontador para a sua cópia imediatamente abaixo, no nível S_i . No último nível, digamos H (é uma variável aleatória) temos $S_H = \emptyset$. A estrutura tem os níveis $0, 1, \dots, H$, dos quais só o último é vazio.

Além dos elementos de S mantemos dois sentinelas em cada nível, $-\infty$ no começo e $+\infty$ no fim de cada lista (inclusive da vazia S_H), que não pertencem a U . Todas as listas são duplamente ligadas. Um apontador topo aponta para o sentinela $-\infty$ do nível H . A Figura 5.1 descreve um exemplo.

As operações de dicionário são descritas a seguir.

busca: Dado $x \in U$, a busca devolve um apontador para o maior elemento de S_0 menor ou igual a x . Assim, devolve o próprio x se $x \in S$; caso contrário, devolve o antecessor de x caso x estivesse em S , e o próximo elemento de S_0 é o menor elemento maior que x . A busca começa em topo. Em cada nível, enquanto o próximo elemento da lista é menor ou igual a x , a busca avança por ele; quando o próximo é maior que x , ela desce um nível pela cópia abaixo, e termina quando não é possível descer, isto é, no nível

¹ Uma árvore binária balanceada é uma árvore com raiz r cujas subárvores esquerda e direita satisfazem algum critério de balanceamento como, por exemplo, a diferença entre as alturas ser de no máximo 1. Um critério bom garante que a altura da árvore é logarítmica no número de elementos, o que faz uma busca ser rápida. Em contrapartida, toda alteração exige o esforço de recompor o balanceamento.

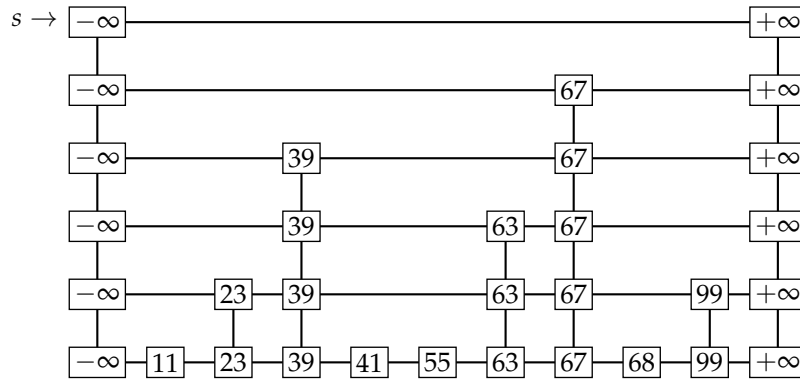


Figura 5.1: exemplo de uma skip list com 6 níveis que representa o conjunto $\{11, 23, 39, 41, 55, 63, 67, 68, 99\}$.

0. O último nó visitado em cada nível i é o maior elemento de S_i menor ou igual a x . A Figura 5.2 destaca um exemplo na estrutura da Figura 5.1;

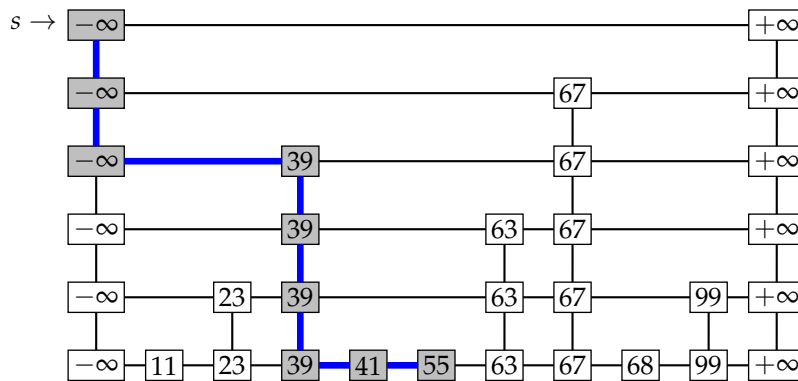


Figura 5.2: O caminho da busca por 55 na skip list da Figura 5.1.

inserção: Dado $x \in U$, a inserção coloca x no lugar apropriado da estrutura caso ele não esteja. Supondo $x \notin S$, uma busca por x determina, para cada nível i , o último nó a_i visitado nele, que é o antecessor de x nesse nível; esses nós são guardados durante a busca. Inserimos x em S_0 logo depois de a_0 . Em seguida jogamos uma moeda: se der cara, inserimos uma cópia de x no nível 1 logo depois de a_1 , ligada à cópia de baixo, e jogamos outra moeda para decidir se há inclusão no nível 2, e assim sucessivamente até sair coroa. Se a replicação chega a um nível que ainda não existe, criamos esses níveis (com sentinelas) e mais um nível vazio no topo, de modo que o último nível continue vazio; assim, H (o índice do último nível) pode aumentar numa inserção;

remoção: Dado $x \in s$, uma busca por x passa por todas as cópias de x , da mais alta, no nível $h(x)$, até a do nível 0. Cada cópia é retirada de sua lista usando os apontadores para o elemento anterior e o seguinte, que é o motivo de as listas serem duplamente ligadas.

Depois, eliminamos os níveis do topo que ficaram vazios; se x era o único elemento dos r níveis mais altos não vazios, então H diminui de r .

Em cada uma das operações, o custo é proporcional ao comprimento do caminho percorrido pela busca, que tem as H descidas mais os passos horizontais. Essas quantidades são variáveis aleatórias, e dependem da estrutura sorteada.

Como $\mathbb{E}|S_i| = n/2^i$ fica abaixo de 1 quando $i > \log_2 n$, esperamos que $H \approx \log_2 n$ e que nos níveis mais altos os nós sejam muito esparsos, de modo que uma busca caminha pouco na horizontal em cada um dos níveis mais altos e, no total, a quantidade de passos de uma busca seja de ordem logarítmica no tamanho do conjunto.

Observemos que sortear as listas nível a nível, ou jogar moedas a cada inserção, produz a mesma distribuição de alturas e que a remoção de um elemento não afeta as alturas dos demais. Por isso, nas análises seguintes supomos que a sequência de operações não depende dos sorteios feitos pela estrutura. Vamos estudar essa estrutura de dados e, para isso, começamos definindo alguns parâmetros importantes.

A *altura de x* , para todo $x \in S$, é denotada por $h(x)$ e é o número de sorteios realizados na inserção de x . Assim, $h(x) \sim \text{Geom}(1/2)$. Além disso, $h(x)$ também é o número de cópias de x :

$$x \in S_i \text{ se, e somente se, } i < h(x).$$

A *altura da skip list S* é

$$H := \max_{x \in S} h(x)$$

com $S_H = \emptyset$. No exemplo da Figura 5.1, $h(11) = 1$ e $h(67) = 5 = H$.

De $h(x) \sim \text{Geom}(1/2)$ temos

$$\mathbb{P}[h(x) = t] = 2^{-t} = \mathbb{P}[h(x) > t] \quad (5.1)$$

para todo inteiro $t \geq 1$. Como $x \in S_i$ equivale a $h(x) > i$ e as alturas são independentes,

$$|S_i| \sim \text{Binomial}(n, 2^{-i}).$$

Podemos calcular a esperança de H , que é uma variável aleatória inteira positiva, pela Proposição 2.6

$$\begin{aligned} \mathbb{E}H &= \sum_{i \geq 0} \mathbb{P}[H > i] \\ &= \sum_{i=0}^m \mathbb{P}[H > i] + \sum_{i > m} \mathbb{P}[H > i], \quad m := \lfloor \log_2 n \rfloor. \end{aligned} \quad (5.2)$$

No primeiro somatório limitamos, trivialmente, cada termo por $\mathbb{P}[H > i] \leq 1$. No segundo, usamos a subaditividade e a equação (5.1), para obtermos

$$\mathbb{P}[H > i] \leq \mathbb{P}\left[\bigcup_{x \in S} [h(x) > i]\right] \leq \sum_{x \in S} \mathbb{P}[h(x) > i] = n 2^{-i},$$

para cada i . Agora, $\sum_{i>m} 2^{-i} = 2 - \sum_{i \leq m} 2^{-i} = 2 - (2 - 2^{-m}) = 2^{-m}$, e juntando as estimativas

$$\mathbb{E}H \leq m + 1 + n \sum_{i>m} 2^{-i} = m + 1 + n 2^{-m} < m + 3, \quad (5.3)$$

em que a última desigualdade vale porque $m > \log_2 n - 1$ implica $2^m > n/2$, isto é, $n 2^{-m} < 2$. Ou seja, como intuímos alguns parágrafos acima, de fato a estrutura tem altura esperada logarítmica no número de elementos de S (como numa árvore balanceada de busca)

$$\mathbb{E}H < \log_2 n + 3.$$

Mais que isso, a altura é logarítmica no número de elementos de S com alta probabilidade.

Como $h(x)$ é inteira, para $\alpha \in \mathbb{R}_+$ vale $h(x) > \alpha$ se e somente se $h(x) > \lfloor \alpha \rfloor$, logo

$$\mathbb{P}[h(x) > 2 \log_2(n) + 1] = 2^{-\lfloor 2 \log_2(n) + 1 \rfloor} < 2^{-2 \log_2(n)} = n^{-2}$$

(pois $\lfloor \alpha \rfloor + 1 > \alpha$) e, pela subaditividade,

$$\begin{aligned} \mathbb{P}\left[\bigcup_{x \in S} [h(x) > 2 \log_2 n + 1]\right] &\leq \sum_{x \in S} \mathbb{P}[h(x) > 2 \log_2 n + 1] \\ &< n \cdot \frac{1}{n^2} = \frac{1}{n}, \end{aligned} \quad (5.4)$$

ou seja, $H \leq 2 \log_2 n + 1$ com alta probabilidade. Note que, nas contas acima, só usamos a distribuição *marginal* de cada $h(x)$, e não a independência das alturas.

Proposição 5.1: A altura é logarítmica com alta probabilidade

Se S é uma *skip list* para um conjunto com n elementos então a altura $H = H(S)$ satisfaz

$$\mathbb{E}H < \log_2 n + 3 \quad \text{e} \quad \mathbb{P}[H > 2 \log_2 n + 1] < \frac{1}{n}.$$

O TAMANHO DA ESTRUTURA

Seja $N := 2(H + 1) + \sum_{i=0}^H |S_i|$ o número de nós na *skip list* (cada nível tem dois sentinelas). Como todo $x \in S$ tem $h(x)$ cópias,

$$\sum_{i=0}^H |S_i| = \sum_{x \in S} h(x).$$

Da linearidade da esperança, da igualdade acima e de $\mathbb{E}h(x) = 2$,

$$\mathbb{E}N = 2\mathbb{E}H + 2 + \sum_{x \in S} \mathbb{E}(h(x)) = 2n + 2\mathbb{E}H + 2 \quad (5.5)$$

e como $\mathbb{E}H < \log_2 n + 3$, temos $\mathbb{E}N = (2 + o(1))n$. Mais que isso, $\mathbb{E}N = O(n)$ com alta probabilidade. De fato, $\sum_{x \in S} h(x)$ é soma de n variáveis $\text{Geom}(1/2)$ independentes (cada elemento é sorteado com moedas próprias). Para $a = 2$, a cota exata de Chernoff para soma de geométricas, equação (4.24), dá

$$\mathbb{P}\left[\sum_{x \in S} h(x) \geq 4n\right] \leq \left(2\left(\frac{2}{3}\right)^3\right)^n = \left(\frac{16}{27}\right)^n < (0,6)^n < e^{-0,5n}$$

portanto, temos $\sum_{x \in S} h(x) < 4n$ com probabilidade $< \exp(-n/2)$ e, pela Proposição 5.1, $2H + 2 > 4\log_2 n + 4$ com probabilidade $\leq 1/n$. Como $N \leq 4n + 4\log_2 n + 4 \leq 5n$ para todo $n \geq 24$, temos $N > 5n$ com probabilidade pelo menos $1 - 1/n - \exp(-n/2)$, uma *skip list* que representa um conjunto com n elementos usa espaço $O(n)$.

Proposição 5.2: O tamanho é linear com alta probabilidade

Se S é uma *skip list* para um conjunto com n elementos então o tamanho N satisfaz

$$\mathbb{E}N = (2 + o(1))n \quad \text{e} \quad \mathbb{P}[N > 5n] = O\left(\frac{1}{n}\right).$$

5.1.1 O tempo de uma busca

Para determinar o número esperado de passos numa busca sobre uma *skip list* vamos limitar o número de comparações em dois grupos de níveis: os mais altos e os mais baixos. Em cada nível visitado, a busca compara x com o próximo elemento da lista: se ele é menor ou igual a x , a busca *avança* no mesmo nível; caso contrário, *desce* um nível (ou termina, no nível 0). Assim, o número de comparações no nível i é o número a_i de avanços nele mais uma comparação, a que provoca a descida ou o término, totalizando

$$\sum_{i=0}^H (a_i + 1)$$

comparações.

Fixado $\ell := \lfloor \log_2 n \rfloor$, os níveis altos são os níveis $\ell, \ell + 1, \dots, H$ e os níveis baixos $0, 1, \dots, \ell - 1$.

Como $|S_i| = 0$ para $i > H$, a soma $\sum_{i=0}^H |S_i|$ coincide com $\sum_{i \geq 0} |S_i|$, e a linearidade se aplica. Já o termo $2(H + 1)$ dos sentinelas depende de H , que é *aleatório*; por isso entra $\mathbb{E}H$ em $\mathbb{E}N$ (veja o Exercício 36 no final deste capítulo).

Proposição 5.3: Comparações nos níveis altos

O número esperado de comparações feitas, durante uma busca, nos níveis $\ell, \dots, H$ de uma *skip list* com n elementos é menor que 9.

Demonstração. Seja $N' := \sum_{i \geq \ell} |S_i|$ o número de cópias de elementos nesses níveis ($S_i = \emptyset$ para $i \geq H$). O número de comparações é

$$= \underbrace{\left(\sum_{i \geq \ell} a_i \right)}_{\text{avanços}} + \underbrace{\text{níveis visitados}}_{\text{uma comparação por nível}}.$$

em cada nível a busca visita cada nó no máximo uma vez, logo o número total de avanços nesses níveis é no máximo N' . O número de níveis visitados é no máximo $H - \ell + 1$. Se $H > \ell$, os níveis $\ell, \dots, H - 1$ são não vazios, e então $H - \ell \leq N'$; só o nível H , vazio, não é contado em N' . Cada nível visitado contribui com uma comparação além dos avanços, de modo que o número de comparações nos níveis $\geq \ell$ é

$$\leq N' + (H - \ell + 1) \leq N' + (N' + 1) = 2N' + 1.$$

Resta estimar $\mathbb{E}N'$. Como $x \in S_i$ se, e somente se, $i < h(x)$, cada elemento x tem $(h(x) - \ell)^+$ cópias nesses níveis, logo $N' = \sum_{x \in S} (h(x) - \ell)^+$ e, por (5.1),

$$\mathbb{E}(h(x) - \ell)^+ = \sum_{i \geq \ell} \mathbb{P}[h(x) > i] = \sum_{i \geq \ell} 2^{-i} = 2 \cdot 2^{-\ell}.$$

Como $2^\ell > n/2$, vale $n 2^{-\ell} < 2$ e, pela linearidade, $\mathbb{E}N' = 2n 2^{-\ell} < 4$. Logo o número esperado de comparações nos níveis mais altos é menor que $2 \cdot 4 + 1 = 9$. $\square$

$$^2 x^+ = \max\{x, 0\}.$$

O segundo passo é estimar o número de comparações nos ℓ níveis mais baixos, $0, \dots, \ell - 1$, onde se concentram a maior parte dos elementos. A ideia-chave é percorrer o caminho da busca *de trás para frente*.

Ideia importante

O caminho reverso: suponha que o caminho da busca de x termina, no nível 0, no nó y que é o maior elemento de S_0 menor ou igual a x (ou $-\infty$). O *caminho reverso* começa em y e, estando em um nó v do nível i , o próximo passo

- **sobe**, se v tem cópia no nível $i + 1$, isto é, se $v \in S_{i+1}$;
- **para a esquerda**, ao elemento anterior de S_i , caso contrário.

Isso reproduz exatamente o caminho da busca: se v ocorre em S_{i+1} , então v é o último elemento de S_{i+1} menor ou igual a x , e a busca chegou a v descendo do nível $i + 1$; se $v \notin S_{i+1}$, então a busca chegou a v avançando no nível i .

Cada passo reverso para cima corresponde a uma descida e cada passo para a esquerda, a um avanço. O número de comparações nos níveis $0, \dots, \ell - 1$ é

$$\sum_{i=0}^{\ell-1} (a_i + 1) = T,$$

o número de passos do caminho reverso do nível 0 ao nível ℓ (as ℓ subidas e os $\sum_{i<\ell} a_i$ passos à esquerda). Se $H < \ell$, a busca só tem os níveis $0, \dots, H$ e o número de comparações é menor.

Qual é a probabilidade de um passo reverso ser para cima? Ela é a probabilidade de $v \in S_{i+1}$, dado que $v \in S_i$, que é $1/2$. Além disso, o caminho reverso nunca visita duas vezes o mesmo par (elemento, nível), então cada passo depende de uma moeda que ainda não foi examinada (princípio das decisões adiadas³). Logo cada passo é “para cima” com probabilidade $1/2$, independentemente dos anteriores.

Para ir do nível 0 ao nível ℓ são necessárias ℓ subidas. O número de passos de cada trecho “esquerda, $\dots$, esquerda, sobe” é o número de moedas até a primeira cara, ou seja, tem distribuição $\text{Geom}(1/2)$, com valores em $\{1, 2, \dots\}$ e média 2.

O caminho reverso pode, porém, alcançar o sentinela $-\infty$ (ou o último nível antes de completar as ℓ subidas) e daí em diante só sobe, sem passos horizontais. Imaginando que o processo continua com moedas fictícias (como nos particionamentos fictícios do *quicksort*), o número T de passos reverso até o nível ℓ satisfaz

$$T \leq G_1 + \dots + G_\ell, \quad G_j \sim \text{Geom}(1/2) \text{ independentes},$$

e, pela linearidade da esperança, $\mathbb{E}T \leq 2\ell$. O custo esperado nos níveis mais baixos é, portanto, no máximo 2ℓ .

Teorema 5.1: O tempo esperado de uma busca

O número esperado de comparações feitas por uma busca em uma *skip list* que representa um conjunto com n elementos é $O(\log n)$.

Demonstração. As comparações dos níveis $0, \dots, \ell - 1$ têm esperança no máximo 2ℓ , e as dos níveis $\ell, \dots, H$ têm esperança menor que 9. Pela linearidade da esperança, o total é menor que

$$2\lceil \log_2 n \rceil + 9 = O(\log n).$$

□

³ Quando um algoritmo ou uma análise depende de muitas escolhas aleatórias, não é preciso sortear todas de antemão. Podemos sortear cada uma só no momento em que ela é consultada pela primeira vez. Se a escolha, no momento da consulta, ainda é uniformemente distribuída e independente de tudo o que foi observado até ali, o resultado é idêntico em lei ao de sortear tudo no início.

Teorema 5.2

Numa *skip list* que representa um conjunto S com cardinalidade $n \geq 3$, para todo $x \in U$ fixado, uma busca por x faz mais de $15 \log_2 n$ comparações com probabilidade no máximo $2n^{-2}$. Além disso, com probabilidade pelo menos $1 - 3/n$, toda busca, por qualquer $x \in U$, faz no máximo $15 \log_2 n$ comparações.

Demonstração. Sejam

$$\begin{aligned} L &:= \log_2 n, \\ a &:= \lceil 3L \rceil, \\ c &:= \lceil 15L \rceil. \end{aligned}$$

Seja T o número de comparações da busca, que é o número de passos do caminho reverso.

Como T é inteiro, $T > 15L$ implica $T \geq c$. Os eventos $[H \leq a]$ e $[H > a]$ particionam o espaço amostral; logo

$$\begin{aligned} [T > 15L] &\subseteq [T \geq c] \\ &= ([T \geq c] \cap [H > a]) \cup ([T \geq c] \cap [H \leq a]) \\ &\subseteq [H > a] \cup ([T \geq c] \cap [H \leq a]) \end{aligned}$$

e, pela subaditividade,

$$\mathbb{P}[T > 15L] \leq \mathbb{P}[H > a] + \mathbb{P}([T \geq c] \cap [H \leq a]) \quad (5.6)$$

Como $\mathbb{P}[h(x) > a] = 2^{-a} \leq 2^{-3L} = n^{-3}$, a subaditividade sobre os n elementos dá

$$\mathbb{P}[H > a] \leq \sum_{x \in S} \mathbb{P}[h(x) > a] \leq n \cdot n^{-3} = n^{-2}. \quad (5.7)$$

Percorremos o caminho reverso a partir do nó final em S_0 , completado por moedas fictícias depois que chega a $-\infty$. Pelo princípio das decisões adiadas, os passos são determinados por uma sequência $f_1, f_2, \dots$ de moedas justas independentes (cara = subir), e, se

$$\tau_j \text{ é o instante da } j\text{-ésima cara em } (f_i)_i,$$

então $T \leq \tau_H$, pois o caminho tem H subidas e cada uma é precedida, em $(f_i)_i$, pelos passos à esquerda, os avanços, que lhe correspondem. Como τ_j é crescente em j ,

$$H \leq a \Rightarrow T \leq \tau_a$$

e

$$[T \geq c] \cap [H \leq a] \subseteq [\tau_a \geq c] \quad (5.8)$$

Se a altura fosse um número fixo $H = k$ e o caminho nunca chegasse a $-\infty$ antes do topo, o caminho reverso seria: lançar moedas justas até obter a k -ésima cara $\tau_k \sim \text{BN}(k, 1/2)$. Entretanto, (1) quando o caminho chega a $-\infty$ antes de completar as subidas, os passos restantes são só subidas, sem moeda, $T \leq \tau_k$; (2) H é variável aleatória e está correlacionada com as moedas que o caminho, condicionar em $H = k$ muda a lei das moedas (por exemplo, força que algum elemento tenha $h(x) = k$ e nenhum tenha mais). Daí a estratégia da prova: usar a monotonicidade de τ_j em j para a inclusão em (5.8) em que τ_a é binomial negativa com parâmetro fixo a $(f_i)_i$ é i.i.d. por construção e a aleatoriedade de H é tratada à parte

Além disso, $\tau_a \geq c$ equivale a haver menos de a caras em $f_1, \dots, f_{c-1}$. Tome B como o número de caras nesses $c - 1$ lançamentos. A variável B tem distribuição Binomial($c - 1, 1/2$) e temos

$$[\tau_a \geq c] = [B < a]$$

Retomando (5.6) e (5.7), temos

$$\mathbb{P}[T > 15L] \leq n^{-2} + \mathbb{P}[B < a]$$

e para fechar a prova, usamos a desigualdade de Chernoff–Hoeffding (Teorema 4.4) fazendo $a = \mathbb{E}B - t$, ou seja, tomando

$$\begin{aligned} t &:= a - \mathbb{E}B \\ &< 3L + 1 - (c - 1)/2 \\ &< 3L + 1 - (15L - 1)/2 \\ &= -4,5L + 0,5 \end{aligned}$$

de modo que a probabilidade de $B < a$ é

$$\leq \mathbb{P}[B \leq \mathbb{E}B - t] \leq \exp\left(-\frac{2t^2}{c-1}\right) \leq \exp\left(-\frac{2(4,5L - 1,5)^2}{15L}\right).$$

No lado direito, no argumento da função exponencial, usando que $1,4 \ln(n) < \log_2(n) = L < 1,5 \ln(n)$, temos que o expoente pode ser limitado superiormente por

$$\begin{aligned} \frac{-40,5L^2 + 27L - 4,5}{15L} &= -2,7L + 1,8 - \frac{0,3}{L} \\ &< -2,7 \cdot (1,4 \ln(n)) + 1,8 \\ &= -3,78 \ln(n) + 1,8 \end{aligned}$$

logo Aplicando a função exponencial, obtemos:

$$\exp\left(-\frac{2(4,5L - 1,5)^2}{15L}\right) < \exp(-3,78 \ln(n) + 1,8) = e^{1,8} \cdot n^{-3,78} = O(n^{-3,78}).$$

Como $3,78 > 2$, para n suficientemente grande, este termo é estritamente menor ou igual a n^{-2} . Portanto, substituindo de volta na cota total da probabilidade

$$\mathbb{P}[T > 15L] \leq n^{-2} + n^{-2} = 2n^{-2} = O(n^{-2}),$$

o que prova que a cauda da distribuição decai polinomialmente, concluindo a demonstração.

Todas as buscas. Uma busca por x termina, no nível 0, no nó $y_0 := \max\{u \in S : u \leq x\}$ (com $y_0 = -\infty$ se não houver tal u). Para todo $u \in S$ vale

$$u \leq x \iff u \leq y_0$$

e $+\infty$ é maior que ambos; logo todas as comparações da busca por x têm o mesmo resultado que as da busca por y_0 , e as duas buscas percorrem o mesmo caminho. Assim, o caminho (e o número T de seus passos) depende de x apenas por meio de y_0 , que pertence a $S \cup \{-\infty\}$, um conjunto com $n + 1 \leq 2n$ elementos. Isso vale para todo $x \in U$, esteja ou não em S .

O evento $[H > a]$ não depende de y_0 e entra uma só vez; o evento $[B < a]$ entra uma vez para cada y_0 . Logo

$$\mathbb{P}[\exists \text{ busca com } > 15L \text{ passos}] \leq n^{-2} + 2n \cdot n^{-2} \leq \frac{3}{n}. \quad \square$$

5.2 Treap

Como no caso de *skip lists*, denotamos por U o conjunto universo e $S \subset U$ deve ser representado de modo a permitir as operações de dicionário: inserção, busca e remoção. Uma *treap* é uma árvore binária em que a cada nó v estão associados $\text{chave}(v)$ e $\text{prioridade}(v)$, de modo que a árvore se comporta como árvore binária de busca com relação às chaves e como uma *heap* de máximo com relação às prioridades (no sentido de árvore ordenada por prioridades, e não de uma *heap* implícita em vetor: aqui a forma da árvore não é determinada por um critério de completude). Supomos as chaves distintas entre si e as prioridades distintas entre si. Um nó v tem a propriedade *heap* se a sua prioridade é maior que a de todos os seus descendentes próprios, e tem a propriedade de árvore de busca se sua chave é maior que as chaves de todos os nós da sua subárvore esquerda e menor que as de todos os nós da sua subárvore direita.

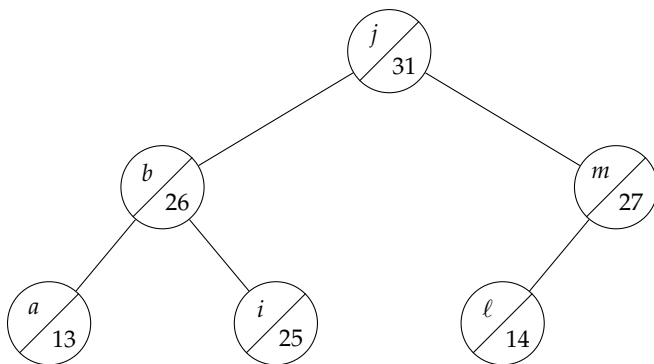


Figura 5.3: Exemplo de *treap* com 6 elementos. As letras são as chaves e os números as prioridades.

As *treaps* foram propostas por (Vuillemin, 1980), que as chamou de árvores cartesianas. Outra forma de descrever uma *treap* é que a estrutura é a árvore binária de busca que resulta da inserção dos nós, um por um, em uma árvore inicialmente vazia, em ordem *decrecente* de prioridade, usando o algoritmo padrão de inserção (o nó de maior prioridade é inserido primeiro e fica na raiz).

Proposição 5.4

Dado um conjunto de pares (chave, prioridade) com chaves distintas e prioridades distintas, existe uma única *treap* que o representa.

Demonstração. Por indução no número de nós. A propriedade *heap* obriga a raiz a ser o nó de maior prioridade; a propriedade de busca obriga a subárvore esquerda a conter exatamente os nós de chave menor que a da raiz e a direita, os de chave maior; e cada subárvore é, ela própria, uma *treap*, única pela hipótese de indução. $\square$

Em particular, a forma da árvore depende apenas das chaves e das prioridades, e não da ordem em que os nós foram inseridos. Seja S um conjunto de elementos de um universo U representados por uma *treap*. As operações de dicionário são descritas a seguir.

busca — Dado $x \in U$ queremos saber se $x \in S$. A busca é feita como em árvore binária de busca: a partir da raiz comparamos x com a chave do nó; se x for igual, a busca termina com sucesso; se x for menor, continuamos na subárvore esquerda; se for maior, na subárvore direita; e a busca falha ao chegar a uma subárvore vazia. O custo é proporcional à altura da árvore;

inserção — Dado $x \in U$ queremos inserir x em S . Fazemos uma busca por x e, caso falhe, inserimos x como folha, no lugar em que a busca terminou, com uma prioridade sorteada para x . Isso mantém a propriedade de árvore de busca, mas pode destruir a propriedade *heap*, que é refeita por rotações: enquanto x tem prioridade maior que a de seu pai y , fazemos a rotação simples que coloca x no lugar de y (uma rotação à direita em y se x é filho esquerdo de y , à esquerda se é filho direito), como na figura 5.4. Durante o processo, a propriedade *heap* só pode ser violada entre x e seu pai; cada rotação preserva a propriedade de busca e, como x tem prioridade maior que a de y , que por sua vez já era maior que as de todos os nós das subárvores B e C , nenhuma nova violação é criada. O número de rotações é no máximo a profundidade de x ;

remoção — Dado $x \in S$ queremos remover x de S . Após uma busca por x , enquanto x não for uma folha rotacionamos x com o seu filho de *maior* prioridade (ou com o único filho, se só houver um), que passa a ser o pai de x e, portanto, tem prioridade maior que a do outro filho, o que preserva a propriedade *heap*. Quando x é uma folha, é removido. O número de rotações é no máximo a altura da subárvore de x .

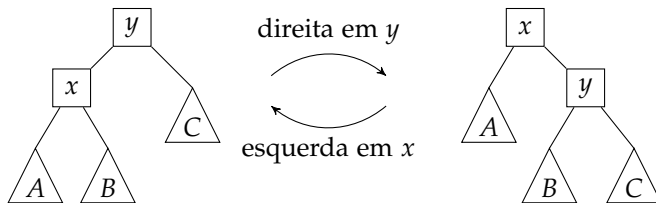


Figura 5.4: Rotações. A rotação à direita em y (da esquerda para a direita) sobe x e desce y ; a rotação à esquerda em x (da direita para a esquerda) faz o inverso. Em ambas, a ordem em-ordem A, x, B, y, C é preservada, apenas a subárvore B muda de pai, e o custo é $O(1)$.

Essas operações têm custo proporcional à altura da árvore e usamos aleatoriedade para garantir altura logarítmica: a prioridade atribuída a um nó quando ele é inserido é sorteada uniformemente em $[0, 1]$, independentemente das demais (em particular, ao reinserir uma chave removida sorteamos uma prioridade nova). Supomos também que a sequência de operações é escolhida independentemente das prioridades (adversário *oblívio*): um adversário que enxergasse as prioridades poderia, por exemplo, remover sempre os nós de menor prioridade, e as prioridades restantes deixariam de ser i.i.d. uniformes. Com probabilidade 1 as prioridades são distintas (a prioridade de um nó não precisa ser um número real; basta que as prioridades sejam distintas e a ordem relativa seja uniforme). Pela Proposição 5.4, depois de qualquer sequência (oblívia) de operações a árvore é a *treap* do conjunto atual, com prioridades independentes e uniformes, pois a remoção de um nó não altera as prioridades dos demais. Por isso basta analisar uma *treap* com n nós e prioridades i.i.d. uniformes.

Seja x_k o k -ésimo menor elemento do conjunto S . Vamos estimar a que distância esse nó está da raiz. Dizemos que x_j é *ancestral próprio* de x_k , e escrevemos $x_j \prec x_k$, se $x_j \neq x_k$ está no caminho da raiz até x_k . Seja $I_j := \mathbb{1}_{[x_j \prec x_k]}$. A *profundidade* de x_k é o seu número de ancestrais próprios,

$$D_k = \sum_{j \neq k} I_j.$$

Ela é o comprimento, em arestas, do caminho da raiz até x_k , e a *altura* da árvore é $\max_k D_k$. Para $j < k$ defina $X(j, k) = \{x_j, x_{j+1}, \dots, x_k\}$ e $X(k, j) = X(j, k)$.

Proposição 5.5

Seja $j \neq k$. Então x_j é ancestral próprio de x_k se, e somente se, x_j tem a *maior prioridade* entre os elementos de $X(j, k)$.

Demonstração. Nesta demonstração convencionamos que todo nó é ancestral de si mesmo. Seja a o ancestral comum mais baixo de x_j e x_k . Vamos mostrar que a é o nó de maior prioridade em $X(j, k)$.

(i) $a \in X(j, k)$: se $a \notin \{x_j, x_k\}$, então x_j e x_k estão em subárvores distintas de a e, pela propriedade de busca, a chave de a está entre as de x_j e x_k .

(ii) $X(j, k)$ está contido na subárvore de a : seja x_i com chave entre as de x_j e x_k . Em cada ancestral próprio b de a , x_j e x_k estão na mesma subárvore de b , logo a chave de b não está entre as deles e x_i fica do mesmo lado de b . Assim, a busca por x_i a partir da raiz segue o mesmo caminho que a busca por x_j até a , e x_i está na subárvore de a .

(iii) Pela propriedade *heap*, a tem prioridade maior que a de todos os demais nós de sua subárvore, em particular que a dos de $X(j, k)$. Como as prioridades são distintas, a é o único nó de maior prioridade em $X(j, k)$.

Agora, como $j \neq k$, x_j é ancestral próprio de x_k se, e somente se, $a = x_j$, o que por (i)–(iii) equivale a x_j ter a maior prioridade em $X(j, k)$. $\square$

Como as prioridades são i.i.d. contínuas, todo elemento de $X(j, k)$ tem a mesma probabilidade de ser o de maior prioridade, e $|X(j, k)| = |k - j| + 1$; logo

$$\mathbb{P}[I_j = 1] = \frac{1}{|k - j| + 1},$$

e, somando separadamente $j < k$ e $j > k$,

$$\mathbb{E}D_k = \sum_{j < k} \frac{1}{k - j + 1} + \sum_{j > k} \frac{1}{j - k + 1} = (H_k - 1) + (H_{n-k+1} - 1) = H_k + H_{n-k+1} - 2,$$

onde $H_n = \sum_{i=1}^n 1/i = \ln n + \gamma + \Theta(n^{-1})$ é o n -ésimo número harmônico e $\gamma \approx 0,577$ é a constante de Euler–Mascheroni (veja ?).

O número de nós no caminho da raiz até x_k é $D_k + 1$, de esperança $H_k + H_{n-k+1} - 1$. Como $H_m \leq 1 + \ln m$ (pois $1/i \leq \int_{i-1}^i dx/x$ para $i \geq 2$), vale

$$\mathbb{E}D_k \leq 2H_n - 2 \leq 2\ln n = O(\log n)$$

para todo k : a profundidade de qualquer nó tem esperança logarítmica.

Para obter limitantes com alta probabilidade usamos a desigualdade de Chernoff. Ela exige independência, e os indicadores I_j não são todos independentes (veja a observação abaixo), mas são independentes de cada lado de k separadamente.

Lema 5.1

Fixado k , os indicadores I_j , $j < k$, são independentes, com $\mathbb{P}[I_j = 1] = 1/(k - j + 1)$. O mesmo vale para os I_j , $j > k$.

Demonstração. Seja y_i a prioridade de x_{k-i+1} , para $i = 1, \dots, k$ (o caso $i = 1$ corresponde ao próprio x_k , para o qual I_k não é definido, e serve apenas para completar a permutação). Pela Proposição 5.5, para $i \geq 2$, $I_{k-i+1} = 1$ se, e somente se, $y_i > \max\{y_1, \dots, y_{i-1}\}$.

Seja $\rho_i \in \{1, \dots, i\}$ o posto de y_i entre $y_1, \dots, y_i$ (posto i é o maior). A ordem relativa de $y_1, \dots, y_k$ é uma permutação uniforme, e a aplicação que leva a permutação a $(\rho_1, \dots, \rho_k)$ é injetora (reconstruímos a permutação inserindo $y_1, y_2, \dots$ nas posições indicadas pelos postos); como os dois conjuntos têm $k!$ elementos, ela é bijetora. Logo $\rho_1, \dots, \rho_k$ são independentes, com ρ_i uniforme em $\{1, \dots, i\}$. Como $I_{k-i+1} = \mathbb{1}_{[\rho_i=i]}$ depende só de ρ_i , os indicadores são independentes, com $\mathbb{P}[I_{k-i+1} = 1] = 1/i$. O caso $j > k$ é simétrico. $\square$

Observação 10. Os indicadores de lados opostos não são independentes. Com $n = 3$ e $k = 2$, $\mathbb{P}[x_1 \prec x_2, x_3 \prec x_2] = \mathbb{P}[\text{a prioridade de } x_2 \text{ é a menor}] = 1/3$, enquanto $\mathbb{P}[x_1 \prec x_2] \mathbb{P}[x_3 \prec x_2] = 1/4$. Por isso separamos $D_k = L_k + R_k$, com $L_k = \sum_{j < k} I_j$ e $R_k = \sum_{j > k} I_j$.

Teorema 5.3

Numa *treap* com $n \geq 2$ nós e prioridades i.i.d. uniformes, com probabilidade pelo menos $1 - 2n^{-7}$ a profundidade de todo nó é menor que $2e^2 \ln n < 15 \ln n$. Em particular, a altura é $O(\log n)$ com probabilidade $1 - O(n^{-7})$, e, nesse instante, cada operação de dicionário tem custo $O(\log n)$ com essa probabilidade.

Demonstração. Fixe k . Pelo Lema 5.1, L_k é soma de $k - 1$ variáveis de Bernoulli independentes, com parâmetros $1/i$, $2 \leq i \leq k$, e $\mathbb{E}L_k = H_k - 1 \leq H_n - 1 \leq \ln n$ (pois $H_n \leq 1 + \ln n$). Para $s > 0$, usando $1 + x \leq e^x$,

$$\mathbb{E}e^{sL_k} = \prod_{i=2}^k \left(1 + \frac{e^s - 1}{i}\right) \leq \exp((e^s - 1) \mathbb{E}L_k) \leq \exp((e^s - 1) \ln n).$$

Pela desigualdade de Markov aplicada a e^{sL_k} , com $s = 2$ e o limiar $e^2 \ln n$,

$$\mathbb{P}[L_k \geq e^2 \ln n] \leq e^{-2e^2 \ln n} e^{(e^2 - 1) \ln n} = n^{-(e^2 + 1)} < n^{-8}.$$

O mesmo vale para R_k . Se $L_k < e^2 \ln n$ e $R_k < e^2 \ln n$, então $D_k < 2e^2 \ln n$. Pela subaditividade, sobre os dois lados e os n nós,

$$\mathbb{P}[\exists k : D_k \geq 2e^2 \ln n] \leq 2n \cdot n^{-8} = 2n^{-7}.$$

Fora desse evento, todo nó tem profundidade menor que $2e^2 \ln n < 15 \ln n$ (note que $2e^2 \approx 14,78$), e portanto a altura, medida em arestas, é menor que $15 \ln n$ (e, medida em nós, é no máximo $15 \ln n + 1$). A hipótese $n \geq 2$ garante $\ln n > 0$; para $n = 1$ a desigualdade estrita $0 < 0$ falharia. $\square$

Observação 11. O Teorema 5.3 refere-se a uma *treap* num instante fixo. Para uma sequência (oblívia) de m operações, cada estado intermediário é uma *treap* com prioridades i.i.d. uniformes. Seja $N \geq 2$ o maior número de nós ao longo da sequência. Como $\mathbb{E}L_k \leq \ln n \leq \ln N$ em cada estado, a demonstração do teorema vale com $\ln N$ no lugar de $\ln n$, e a subaditividade sobre os m estados mostra que todas as operações têm custo menor que $15 \ln N + 1$ com probabilidade pelo menos $1 - 2m N^{-7}$. Em particular, isso vale com probabilidade $1 - O(N^{-5})$ para $m = O(N^2)$.

5.3 Streaming

Um *fluxo* (*streaming*) de dados é uma sequência extremamente longa de itens provenientes de um universo, que pode ser lida apenas uma vez, em ordem. Há dados demais para armazenar, e eles chegam rápido demais para qualquer computação mais elaborada. Três exemplos:

- os pacotes que passam por um roteador de rede, em que queremos saber quantos endereços de origem distintos aparecem (o que pode indicar um ataque) ou quais enviam mais tráfego;
- as requisições a um servidor web ou as consultas a um mecanismo de busca, em que queremos contar visitantes únicos ou descobrir os termos mais buscados;
- as leituras de sensores, como os de temperatura ou de vibração de máquinas, que geram dados continuamente e muitas vezes em dispositivos com memória muito limitada.

No modelo de *streaming*, os dados chegam um a um, como uma sequência $\sigma = (x_1, x_2, \dots, x_m)$, em geral com m muito grande e possivelmente desconhecido, e cada x_i é um elemento do universo $\{1, \dots, n\}$. O algoritmo dispõe de uma memória muito menor que m , idealmente polilogarítmica em m e n .⁴ Com tão pouco espaço, resultados exatos costumam ser impossíveis, e recorremos a aleatoriedade e a aproximações do tipo (ε, δ) : com probabilidade pelo menos $1 - \delta$, o erro relativo é no máximo ε .

Alguns problemas básicos nesse modelo são:

- Elementos distintos*: estimar quantos valores diferentes aparecem no fluxo, como os endereços de origem distintos num roteador ou os visitantes únicos de um servidor web.
- Estimativa de frequências*: dado um elemento i , estimar quantas vezes ele apareceu, sem manter um contador por elemento. Isso

⁴ Isto é, da ordem de $(\log m + \log n)^{O(1)}$ bits, para ε e δ fixos. O termo $\log n$ é o custo de escrever um elemento do universo, e $\log m$ o de escrever um contador que chega a m ; o algoritmo pode guardar $(\log(nm))^{O(1)}$ desses objetos, mas não uma quantidade que cresça como potência de m ou de n . Para comparação, memorizar o fluxo inteiro custaria $m \log n$ bits, e guardar um bit por elemento do universo, n bits.

permite identificar os itens mais frequentes (*heavy hitters*), como os termos mais buscados ou as *hashtags* em alta.

- c) *Momentos de frequência*: se f_i é o número de ocorrências do elemento i em σ , o k -ésimo momento é $F_k = \sum_{i=1}^n f_i^k$ (com $0^0 = 0$). Assim, F_0 é o número de elementos distintos, F_1 é o comprimento do fluxo e F_2 mede a não uniformidade da distribuição. O problema é estimar F_k para cada $k \geq 0$.
- d) *Amostragem em fluxo*: manter uma amostra uniforme de ℓ elementos de um fluxo de tamanho desconhecido. Relaciona-se à prática de inspecionar apenas uma amostra dos pacotes que passam por um roteador.
- e) *Contagem aproximada*: contar o número m de itens usando muito menos que $\log_2 m$ bits. É o problema mais simples da lista e o que abordamos em seguida. Em vez do valor exato, guarda-se um contador que cresce probabilisticamente, de modo que seu valor é da ordem de $\log m$ e ocupa apenas $O(\log \log m)$ bits, para ε e δ fixos.

Começemos pela contagem exata: dado um fluxo de comprimento em $\{1, \dots, m\}$, com o limite m conhecido de antemão, determinar seu comprimento. Uma solução é manter um contador c , inicialmente nulo, e aumentá-lo de uma unidade a cada item lido. Basta guardar o resto de c na divisão por m , que assume m valores distintos e pode ser escrito com $\lceil \log_2 m \rceil$ bits; ao final, o resto 0 é interpretado como m . É possível usar bem menos? Para algoritmos determinísticos que devolvem a resposta exata, não: a proposição a seguir mostra que nem um único bit a menos basta.

Proposição 5.6

Seja $m \geq 2$. Nenhum algoritmo determinístico que usa no máximo $\lceil \log_2 m \rceil - 1$ bits de memória devolve o comprimento exato de toda sequência de comprimento em $\{1, \dots, m\}$. O mesmo vale para algoritmos aleatorizados que nunca erram.

Observe que o limite da Proposição 5.6 é justo: o contador módulo m descrito acima usa exatamente $\lceil \log_2 m \rceil$ bits.

Demonstração. Seja $b = \lceil \log_2 m \rceil - 1 \geq 0$. Como $\lceil \log_2 m \rceil < \log_2 m + 1$, temos $2^b < m$, isto é, há menos de m estados de memória possíveis. Em um algoritmo determinístico o estado inicial é fixo, o estado após ler uma sequência depende apenas dela, e a saída é função do estado final.

Para cada $i \in \{1, \dots, m\}$ seja S_i o estado após ler a sequência constante $(1, \dots, 1)$ de comprimento i . Como há m sequências e menos de m estados, existem $i \neq j$ com $S_i = S_j$. O algoritmo devolve então a mesma resposta para as duas sequências, mas as respostas corretas, i e j , são diferentes.

Suponha agora que o algoritmo é aleatorizado, que seus bits aleatórios vêm de uma semente r (não contada na memória) e que ele nunca erra, isto é, responde corretamente para toda semente e toda sequência. Fixada r , o algoritmo torna-se determinístico, usa no máximo b bits e acerta em todas as sequências, o que acabamos de ver ser impossível. $\square$

Observe que o limite da Proposição 5.6 é justo: o contador módulo m descrito acima usa exatamente $\lceil \log_2 m \rceil$ bits.

Permitindo estimativas e aleatorização, a economia de memória pode ser enorme. A ideia é de Morris (1978), que a propôs em 1978 para contar muitos eventos distintos em registradores de apenas 8 bits.

Definição 5.1: (ε, δ) -aproximação

Dados os números reais $\varepsilon, \delta \in (0, 1)$, chamamos um algoritmo aleatorizado de uma (ε, δ) -aproximação se, para todo fluxo de comprimento m , calcula uma estimativa $\hat{m}$ de m tal que

$$\mathbb{P}[|\hat{m} - m| \leq \varepsilon m] \geq 1 - \delta,$$

em que a probabilidade é sobre as escolhas aleatórias do algoritmo. Ou seja, $1 - \varepsilon \leq \hat{m}/m \leq 1 + \varepsilon$ com probabilidade $\geq 1 - \delta$. Equivalentemente, $\mathbb{P}[|\hat{m} - m| > \varepsilon m] \leq \delta$.

Veja o Exercício 30, pág. 109, e seguintes.

Naturalmente, quanto menores ε e δ , melhor a garantia, e em geral maior a memória necessária.

Algoritmo 5.1: Algoritmo de Morris

Instância: um fluxo $(x_1, \dots, x_m)$, $x_i \in \{1, 2, \dots, n\}$.

Resposta: uma estimativa $\hat{m}$ de m .

- 1 $X \leftarrow 0$;
- 2 **para cada** item lido **faça**
- 3 com probabilidade 2^{-X} , faça $X \leftarrow X + 1$;
- 4 **responda** $2^X - 1$.

A saída $2^X - 1$ é a estimativa $\hat{m}$. Quando $X = j$, o contador é incrementado com probabilidade 2^{-j} a cada item, de modo que o

número de itens até o incremento é geométrico, com média 2^j . Assim X passa de j para $j + 1$ depois de cerca de 2^j itens, e 2^X acompanha o número de itens vistos⁵. O algoritmo guarda apenas X , que, com alta probabilidade, é da ordem de $\log_2 m$ e portanto pode ser escrito com $O(\ln \ln m)$ bits. Os dois resultados a seguir tornam isso preciso. Antes, fixamos alguma notação.

Seja X_i o valor de X após processar i itens, com $X_0 = 0$. Dado $X_{i-1} = j$,

$$X_i = \begin{cases} j+1 & \text{com probabilidade } 2^{-j} \\ j & \text{com probabilidade } 1 - 2^{-j}, \end{cases}$$

sendo o sorteio do i -ésimo item independente do passado. Para uma função $f: \mathbb{R} \rightarrow \mathbb{R}$ defina

$$g_f(j) := (1 - 2^{-j})f(j) + 2^{-j}f(j+1).$$

Condicionando em X_{i-1} e usando a lei da esperança total, para $i \geq 1$ temos

$$\mathbb{E}f(X_i) = \mathbb{E}g_f(X_{i-1}). \quad (5.9)$$

Como $X_i \in \{0, 1, \dots, i\}$, as esperanças acima são somas finitas e existem para qualquer f .

Teorema 5.4

Para toda entrada de comprimento $m \geq 0$, a esperança sobre as escolhas aleatórias do algoritmo satisfaz $\mathbb{E}2^{X_m} = m + 1$. Em particular, a saída $\hat{m} = 2^{X_m} - 1$ satisfaz $\mathbb{E}\hat{m} = m$, isto é, $\hat{m}$ é um estimador não viesado de m .

Demonstração. A prova é por indução em m . Para $m = 0$, $X_0 = 0$ e $\mathbb{E}2^{X_0} = 1$. Seja $m \geq 1$ e suponha $\mathbb{E}2^{X_{m-1}} = m$. Para $f(j) = 2^j$,

$$g_f(j) = (1 - 2^{-j})2^j + 2^{-j}2^{j+1} = 2^j - 1 + 2 = 2^j + 1,$$

e por (5.9), pela linearidade da esperança e pela hipótese de indução,

$$\mathbb{E}2^{X_m} = \mathbb{E}g_f(X_{m-1}) = \mathbb{E}2^{X_{m-1}} + 1 = m + 1.$$

Por fim, $\mathbb{E}\hat{m} = \mathbb{E}2^{X_m} - 1 = m$. □

Logo o algoritmo de Morris é um estimador não viesado de m . Falta controlar a dispersão pois ueremos também que $\hat{m}$ se concentre em torno de sua esperança, e para isso precisamos da variância, que calculamos a partir do segundo momento de 2^{X_m} .

⁵ Como $1 + 2 + \dots + 2^{j-1} = 2^j - 1$, chegar a $X = j$ leva em média $2^j - 1$ itens, o que explica a saída $2^X - 1$.

Lema 5.2

Para toda entrada de comprimento $m \geq 0$,

$$\mathbb{E}4^{X_m} = \frac{3}{2}m^2 + \frac{3}{2}m + 1.$$

Demonstração. Para $f(j) = 4^j$,

$$g_f(j) = (1 - 2^{-j})4^j + 2^{-j}4^{j+1} = (4^j - 2^j) + 4 \cdot 2^j = 4^j + 3 \cdot 2^j.$$

Seja $i \geq 1$. Por (5.9), pela linearidade da esperança e pelo Teorema 5.4, aplicado ao comprimento $i - 1$,

$$\mathbb{E}4^{X_i} = \mathbb{E}4^{X_{i-1}} + 3 \mathbb{E}2^{X_{i-1}} = \mathbb{E}4^{X_{i-1}} + 3i.$$

Aplicando essa recorrência para $i = m, m - 1, \dots, 1$ e usando $\mathbb{E}4^{X_0} = 1$,

$$\mathbb{E}4^{X_m} = 1 + 3 \sum_{i=1}^m i = 1 + \frac{3m(m+1)}{2} = \frac{3}{2}m^2 + \frac{3}{2}m + 1. \quad \square$$

Corolário 5.1

Para toda entrada de comprimento $m \geq 0$,

$$\text{Var}(\hat{m}) = \frac{m(m-1)}{2}.$$

Demonstração. Como $\hat{m} = 2^{X_m} - 1$, temos $\text{Var}(\hat{m}) = \text{Var}(2^{X_m})$.

Usando $4^{X_m} = (2^{X_m})^2$, o Teorema 5.4 e o Lema 5.2,

$$\begin{aligned} \text{Var}(2^{X_m}) &= \mathbb{E}4^{X_m} - (\mathbb{E}2^{X_m})^2 \\ &= \frac{3}{2}m^2 + \frac{3}{2}m + 1 - (m+1)^2 = \frac{m^2 - m}{2}. \quad \square \end{aligned}$$

Pelo Corolário 5.1, para $m \geq 1$ temos $\text{Var}(\hat{m}) = m(m-1)/2 \leq m^2/2$. Em particular, o desvio padrão de $\hat{m}$ é da ordem de $m/\sqrt{2}$, isto é, comparável ao próprio m . Como $\hat{m}$ é não viesado (Teorema 5.4), a desigualdade de Chebyshev dá, para todo $\varepsilon > 0$,

$$\mathbb{P}[|\hat{m} - m| \geq \varepsilon m] \leq \frac{\text{Var}(\hat{m})}{\varepsilon^2 m^2} \leq \frac{1}{2\varepsilon^2}.$$

Esse limitante só é não trivial se $\varepsilon > 1/\sqrt{2}$. Como na definição tomamos $\varepsilon \in (0, 1)$, ele é sempre maior que $1/2$: não garante $\delta < 1/2$ para nenhuma precisão $\varepsilon < 1$, e é inútil na prática. Precisamos melhorar a concentração, e veremos dois truques comuns para isso. O primeiro é tomar a média de s cópias independentes do algoritmo, o que divide a variância por s . O segundo é tomar a mediana de várias dessas médias, o que amplifica a confiança, isto é, diminui a probabilidade de falha δ .

5.3.1 O truque da média

Pela desigualdade de Chebyshev, a probabilidade de erro é controlada pela variância, então podemos melhorar a concentração reduzindo-a. Lembremos que $\text{Var}(cX) = c^2 \text{Var}(X)$ para toda constante c , e que $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$ para variáveis independentes X e Y .

Executamos o algoritmo de Morris t vezes sobre o mesmo fluxo, em paralelo e com sorteios independentes, obtendo saídas $\hat{m}_1, \dots, \hat{m}_t$, independentes e com a mesma distribuição, e devolvemos

$$\hat{m}^* := \frac{1}{t} \sum_{i=1}^t \hat{m}_i.$$

Pela linearidade da esperança, $\mathbb{E}\hat{m}^* = m$ e, pela independência,

$$\text{Var}(\hat{m}^*) = \frac{1}{t^2} \sum_{i=1}^t \text{Var}(\hat{m}_i) = \frac{\text{Var}(\hat{m}_1)}{t} \leq \frac{m^2}{2t}$$

(Corolário 5.1). Para $m \geq 1$, a desigualdade de Chebyshev dá

$$\mathbb{P}[|\hat{m}^* - m| \geq \varepsilon m] \leq \frac{1}{2t\varepsilon^2}.$$

Tomando $t = \lceil 1/(\varepsilon^2 \delta) \rceil$, a probabilidade de que $\hat{m}^*$ erre por mais que εm é no máximo δ , isto é, $\hat{m}^*$ é uma (ε, δ) -aproximação. A memória é a de t contadores, com $O(\ln \ln m)$ bits cada (com alta probabilidade), ou seja, $O(\varepsilon^{-2} \delta^{-1} \ln \ln m)$ bits. A dependência em $1/\delta$ é ruim: queremos uma probabilidade de falha muito pequena sem pagar tanto. É o que faz o próximo truque.

5.3.2 O truque da mediana

Escolhemos $t = \lceil 3/(\varepsilon^2) \rceil$ e executamos o algoritmo da média, independentemente e em paralelo, s vezes, com s ímpar. Denotemos as saídas por $\hat{m}_1^*, \dots, \hat{m}_s^*$. Como $\text{Var}(\hat{m}_i^*) \leq m^2/(2t)$, Chebyshev dá, para cada i ,

$$\mathbb{P}[|\hat{m}_i^* - m| \geq \varepsilon m] \leq \frac{1}{2t\varepsilon^2} \leq \frac{1}{3}.$$

Devolvemos a mediana $\hat{m}^{**}$, isto é, o $\frac{s+1}{2}$ -ésimo menor valor.

Seja Y_i o indicador do evento *bom*

$$|\hat{m}_i^* - m| < \varepsilon m.$$

As Y_i são independentes e $\mathbb{P}[Y_i = 1] \geq 2/3$; logo $Y := \sum_{i=1}^s Y_i$ satisfaz $\mathbb{E}Y \geq 2s/3$.

Se a mediana é ruim, isto é, $|\hat{m}^{**} - m| \geq \varepsilon m$, então pelo menos $(s+1)/2$ valores são ruins: se $\hat{m}^{**} \geq (1+\varepsilon)m$, todos os $\frac{s+1}{2}$ valores

maiores ou iguais à mediana são ruins, e analogamente se $\hat{m}^{**} \leq (1 - \varepsilon)m$ com os menores ou iguais. Logo $Y \leq s - \frac{s+1}{2} < \frac{s}{2}$ e, como $\mathbb{E}Y \geq 2s/3$,

$$Y - \mathbb{E}Y < \frac{s}{2} - \frac{2s}{3} = -\frac{s}{6}.$$

Pela cota aditiva de Chernoff–Hoeffding (Teorema 4.4) para as s variáveis $Y_i \in [0, 1]$,

$$\mathbb{P}[|\hat{m}^{**} - m| \geq \varepsilon m] \leq \mathbb{P}\left[Y - \mathbb{E}Y \leq -\frac{s}{6}\right] \leq \exp\left(-\frac{2(s/6)^2}{s}\right) = \exp\left(-\frac{s}{18}\right).$$

(A cota aditiva depende só do desvio $s/6$, e não do valor exato de $\mathbb{E}Y$; isso evita $\sup \mathbb{E}Y = 2s/3$.) Para que a falha de precisão seja $\leq \delta/2$, tomamos s o menor ímpar com $s \geq 18 \ln(2/\delta)$.

5.3.3 Espaço e resumo das garantias

Até aqui controlamos apenas o *erro* das estimativas. Esta subseção mostra que os contadores, de fato, ocupam pouca memória, e reúne as garantias das duas variantes. Supomos conhecido um limite superior para m , com logaritmo da ordem de $\log m$, e escrevemos m nas cotas; é ele que permite reservar o espaço dos contadores antes de ler o fluxo.

Observação 12 (O espaço dos contadores). Considere $N \geq 1$ execuções do algoritmo de Morris sobre entradas de comprimento $m \geq 1$ (não precisam ser independentes), com contadores finais $X^{(1)}, \dots, X^{(N)}$, e seja $\eta \in (0, 1)$. Pelo Teorema 5.4 e pela desigualdade de Markov, para cada j ,

$$\mathbb{P}\left[2^{X^{(j)}} \geq \frac{N(m+1)}{\eta}\right] \leq \frac{\eta}{N}.$$

Pela cota da união, com probabilidade pelo menos $1 - \eta$ vale $X^{(j)} < \log_2 \frac{N(m+1)}{\eta}$ para todo j , e então cada contador cabe em

$$\log_2 \log_2 \frac{N(m+1)}{\eta} + O(1) = O\left(\log \log m + \log \log \frac{N}{\eta}\right)$$

bits. O espaço efetivamente usado é aleatório. Para ter memória fixa, o algoritmo reserva esse espaço e *aborta* se algum contador o exceder, o que ocorre com probabilidade no máximo η .

Teorema 5.5

Sejam $\varepsilon, \delta \in (0, 1)$ e $m \geq 1$, com $1/(\varepsilon\delta) \leq m^{O(1)}$. Existem algoritmos que (ε, δ) -aproximam m , com os seguintes usos de memória durante a leitura do fluxo:

a) (*média*) com $t = \lceil 1/(\varepsilon^2 \delta) \rceil$ contadores,

$$O\left(\frac{\log \log m}{\varepsilon^2 \delta}\right) \text{ bits};$$

b) (*mediana de médias*) com ts contadores, em que $t = \lceil 3/(2\varepsilon^2) \rceil$ e s é o menor ímpar com $s \geq 18 \ln(2/\delta)$,

$$O\left(\frac{1}{\varepsilon^2} \cdot \ln \frac{2}{\delta} \cdot \log \log m\right) \text{ bits}.$$

Demonstração. Seja N o número de contadores ($N = t$ em a), $N = ts$ em b)) e seja A a variante correspondente, sem limite de espaço. Pelas subseções anteriores, o evento E_1 de que a saída de A erra por εm ou mais tem probabilidade no máximo $\delta/2$.

Pela Observação 12 com $\eta = \delta/2$, o evento E_2 de que algum contador de A excede $\log_2 \frac{2N(m+1)}{\delta}$ tem probabilidade no máximo $\delta/2$. Seja A' o algoritmo que se comporta como A , mas reserva esse espaço para cada contador e aborta se o exceder. Fora de E_2 , as execuções de A e A' coincidem; logo A' erra apenas se $E_1 \cup E_2$ ocorre, com probabilidade no máximo δ .

O espaço de A' é N vezes $O(\log \log m + \log \log(N/\delta))$. Como $N \leq \text{poly}(1/(\varepsilon\delta))$ e $1/(\varepsilon\delta) \leq m^{O(1)}$, temos $\log \log(N/\delta) = O(\log \log m)$. Substituindo $N = O(\varepsilon^{-2}\delta^{-1})$ em a) e $N = O(\varepsilon^{-2} \ln(2/\delta))$ em b), obtemos os limites. $\square$

Observação 13. Os limites acima referem-se ao estado mantido *durante a leitura do fluxo*. As médias e a mediana são calculadas ao final, a partir dos contadores guardados. Isso usa $O(\log m)$ bits de memória de trabalho: as médias são comparadas pelas somas dos t valores (o que dispensa a divisão por t) e recalculadas sob demanda para achar o elemento de posto $(s+1)/2$, sem guardar as s médias. A própria saída, escrita em binário, já ocupa $\Theta(\log m)$ bits.

Comparando as duas variantes, a mediana troca o fator $1/\delta$ por $\ln(2/\delta)$ ao custo de um fator constante, o que a torna preferível quando δ é pequeno.

5.4 Exercícios

Exercício 34. Dado um conjunto de n números e um real positivo ε , uma ε -aproximação da mediana é um elemento cujo posto (posição considerando os mesmos elementos em ordem crescente) está no intervalo $[(1-\varepsilon)n/2, (1+\varepsilon)n/2]$. Projete um algoritmo aleatorizado para computar uma $1/2$ -aproximação da mediana de um vetor com n

elementos com tempo de execução $O(\log n \log \log n)$ e probabilidade de erro $2n^{-2}$ (dica: amostre e ordene).

Exercício 35. Para dados ε e c positivos, projete um algoritmo aleatorizado para computar uma ε -aproximação da mediana de um vetor com n elementos com tempo de execução $O(\log n \log \log n)$ e probabilidade de erro n^{-c} .

Exercício 36. Sejam $X_i \sim \text{Bernoulli}(1/2)$, para $i \geq 1$, variáveis aleatórias independentes e $N = \min\{n \geq 1: X_{n+1} = 1\}$. Prove que $N + 1 \sim \text{Geom}(1/2)$ e que $\mathbb{E}N = 1$. Assuma que se $N = 0$ então $X_1 + \dots + X_N = 0$ e prove que

$$\mathbb{E} \sum_{i=1}^N X_i \neq \sum_{i=1}^{\mathbb{E}N} \mathbb{E}X_i. \quad (5.10)$$

Exercício 37. Verifique que a hipótese de convergência é necessária no Exercício 10, página 54. Para tal, considere o espaço de probabilidade $(\mathbb{N}, \mathbb{P})$ com $\mathbb{P}(n) = 2^{-n}$ e as variáveis aleatórias

$$X_n(j) := \begin{cases} 2^n, & \text{se } j = n, \\ -2^{n+1}, & \text{se } j = n+1, \\ 0, & \text{caso contrário.} \end{cases}$$

1. Mostre que $\mathbb{E}X_n = 0$ para todo $n \geq 1$.
2. Tome $X = \sum_{n \geq 1} X_n$. Determine $X(1), X(2), X(3), \dots$.
3. Usando o item anterior determine $\mathbb{E}X = \sum_{n \geq 1} X(n)\mathbb{P}(n)$.

Conclua que $\mathbb{E} \sum_n X_n \neq \sum_n \mathbb{E}X_n$.

Exercício 38. Prove que numa *skip list*,

$$\mathbb{P}[H > 100 \log_2(n)] \leq \frac{1}{n^{99}}.$$

Exercício 39. Verifique à mão que $\mathbb{E}2^{X_m} = 3,4$ e $\text{Var}(\hat{m}) = 1,3$ para $m = 2, 3$, e calcule $\mathbb{P}[X_2 = j]$ para $j = 1, 2$.

Exercício 40. (Morris com base $1 + \alpha$.) Considere a variante em que X sobe com probabilidade $(1 + \alpha)^{-X}$ e a saída é $\hat{m} = ((1 + \alpha)^X - 1)/\alpha$. Mostre que $\mathbb{E}\hat{m} = m$ e que $\text{Var}(\hat{m}) = \alpha m(m-1)/2$. Conclua que $\alpha = 2\varepsilon^2\delta$ já dá uma (ε, δ) -aproximação por Chebyshev, sem cópias paralelas.

Exercício 41. Suponha que temos uma fonte de bits aleatórias que responde 0 com probabilidade $p \in (0, 1)$ e responde 1 com probabilidade $1 - p$, independentemente. Escreva um algoritmo que usa essa fonte responda 0 ou 1 uniforme e independentemente. Calcule o tempo esperado de execução em função de p .

Exercício 42. Prove que se distribuimos ao acaso m bolas em n caixas com a condição que $m < (2/e)n \ln(n)$, então com alta probabilidade o maior número de bolas em uma caixa é

$$\frac{4 \ln(n)}{\ln\left(\frac{2n}{me} \ln(n)\right)}.$$

Exercício 43 (lei de grandes desvios para variáveis aleatórias binomiais). Sejam $X_i \sim \text{Bernoulli}(1/2)$ variáveis aleatórias independentes $1/2 < \alpha \leq 1$ um real. Seja $S_n = X_1 + \cdots + X_n$. Verifique que

$$\frac{1}{2^n} \frac{n!}{\lceil \alpha n \rceil! (n - \lceil \alpha n \rceil)!} \leq \mathbb{P}[S_n \geq \alpha n] \leq \frac{n+1}{2^n} \frac{n!}{\lceil \alpha n \rceil! (n - \lceil \alpha n \rceil)!}$$

e conclua, usando a aproximação de Stirling, que

$$\lim_{n \rightarrow \infty} \frac{1}{n} \mathbb{P}[S_n \geq \alpha n] = -I(\alpha)$$

onde $I(\alpha) = \alpha \ln \alpha + (1 - \alpha) \ln(1 - \alpha) + \ln 2$.