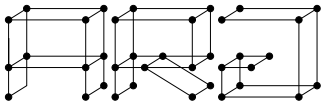
Rascunho de Notas de Estudo sobre

Hashing Perfeita

Leandro Zatesko Orientador: Prof. Dr. Jair Donadelli Jr.

http://www.inf.ufpr.br/{leandro,jair}



Algorithms Research Group Universidade Federal do Paraná

Mestrado em Ciência da Computação

Sumário

1 Introdução

1.1 Básico

Definição 1. Uma *hash* é uma função $h\colon U \to M$. $S \subseteq U$, $U = [0..u-1]$ finito, chamado de universo. $M = [0..m-1]$ finito. $|U| = u$, $|M| = m$, $|S| = n$. $x \in S$ é chamado de chave. Cada $t \in M$ é chamado de endereço. Uma tabela hash é uma área de armazenamento para as imagens dos elementos de S por h. Se $h(x) = h(y)$, dizemos que x e y são sinônimos e dizemos que ocorreu uma colisão. h é perfeita se é injetiva para os elementos de S. Se h é perfeita e $m = n$, h é mínima. Particionamos S em $\{S_0, \ldots, S_{m-1}\}$ onde

$$S_i = \{x\colon x \in S \text{ e } h(x) = i\}\,.$$

1.2 FKS

Teorema 2. *$m \geqslant n$. Então existe um $a \in [1..u-1]$ tal que, para $h\colon x \mapsto (ax \bmod u) \bmod m$,*

$$\sum_{i=0}^{m-1} \binom{|\, S_i \,|}{2} = \frac{n(n-1)}{m}.$$

Demonstração. $\binom{S_i}{2} = \{(x,y)\colon x,y \in S, x \neq y, h(x) = h(y) = i\}$. Se $x \neq y$ mas $h(x) = h(y) = i$, então

$$(ax \bmod u) \bmod m = (ay \bmod u) \bmod m.$$

Logo, $a(x-y) \bmod u \in X = \{m, 2m, 3m, \ldots, u-m, u-2m, u-3m, \ldots\}$. Portanto,

$$\bigcup_a \bigcup_i \binom{S_i}{2} \simeq \bigcup_{x,y \in S x \neq y} \{a\colon a(x-y) \bmod u \in X\},$$

e

$$\begin{aligned}\sum_a \sum_i \binom{|\, S_i \,|}{2} &= \sum_{\substack{x,y \in S \\ x \neq y}} |\, \{a\colon a(x-y) \bmod u \in X\} \,| \\ &= |\, \{a\colon \exists (x,y), x \neq y, a(x-y) \bmod u \in X\} \,|.\end{aligned}$$

Como para todo x, y, $x \neq y$, $|\, \{a\colon h(x) = h(y)\} \,| < \frac{2u}{m}$,

$$\sum_a \sum_i \binom{|\, S_i \,|}{2} < \binom{n}{2} \frac{2u}{m} = \frac{un(n-1)}{m}.$$

Logo, existe ao menos um a tal que $\sum_i \binom{|\, S_i \,|}{2} < \frac{n(n-1)}{m}$. ♦

Corolário 3. *Existe um a tal que, para $h\colon x \mapsto (ax \bmod u) \bmod n$, $\sum_i |\, S_i \,|^2 < 3n$.*

Demonstração. Tomemos $m = n$, então $\sum_i \binom{|\, S_i \,|}{2} < n-1$. Como

$$|\, S_i \,|^2 = 2\binom{|\, S_i \,|}{2} + |\, S_i \,|$$

e $\sum_i |\, S_i \,| = n$,

$$|\, S_i \,|^2 = 2(n-1) + n < 3n.$$

♦

Corolário 4. *Existe a tal que se $m = n(n-1)+1$, h é injetiva para S.*

Demonstração. Existe um a tal que

$$\sum |\, S_i \,|^2 < \frac{2n(n-1)}{n(n-1)+1} + n < n+2.$$

Como $|\, S_i \,| \in \mathbb{N}$ e como $|\, S_i \,| \leqslant 1$ — supondo existir um S_i tal que $|\, S_i \,| \geqslant 2$, temos que $\sum |\, S_i \,|^2 \not< n+2$. Logo, h injetiva para S. ♦

Definição 5 (Esquema FKS)**.** Função de *hashing* primária: $h\colon x \mapsto (ax \bmod u) \bmod n$, onde a é tal que $\sum |\, S_i \,|^2 < 3n$. A função primária fornece, para cada x, o índice i para achar a_i e c_i. As funções secundárias são as $h_1\colon x \mapsto (a_i x \bmod u) \bmod (|\, S_i \,| \, (|\, S_1 \,| - 1) + 1)$. $|\, S_i \,| \, (|\, S_1 \,| - 1) + 1$ é o c_i, que é o tamanho da tabela *hash* para h_i. As chaves

x são armazenadas na tabela D em $C_i + h_i(x)$, onde os $C_i = \sum_{j=0}^{i=1} c_j$ são também previamente armazenados numa tabela C, assim como os a_i em A e os c_i em c. A *hashing* é feita por $h_i \circ h$. Note que $(h_i \circ h)(x) = C_i + h_i(x)$. $h_i \circ h$ requer a, u, n, $A[0..n-1]$, $c[0..n-1]$, $C[0..n-1]$. Total de $3n + O(1)$ locações, ou $(3n + O(1)) \log u$ bits. Assim D também requer $3n \log u$ bits. Porém, gostaríamos de $\Omega(n + \log\log u)$ bits. Ademais, construir a representação de S pode ser $O(nu)$ no pior caso — para encontrar a, podemos ter de fazer uma busca exaustiva que custaria $O(nu)$: u possibilidades testado os n elementos de S cada.

1.3 FKS um pouco melhorado

Corolário 6. *Para pelo menos metade dos valores $a \in [1..u-1]$, $\sum_{i=0}^{n-1} |S_i|^2 < 5n$.*

Demonstração. Como

$$\sum_{a=1}^{u-1} \sum_{i=0}^{m-1} \binom{|S_i|}{2} < \frac{un(n-1)}{m},$$

Se $m = n$,

$$\sum_{a=1}^{u-1} \sum_{i=0}^{m-1} \binom{|S_i|}{2} < u(n-1) = u(m-1).$$

Logo,

$$\sum_{a=1}^{u-1} \left(\sum_{i=0}^{m-1} |S_i|^2 - n \right) < 2u(n-1).$$

Já que, para todo a, $\sum_{i=0}^{m-1} |S_i|^2 - n \geqslant 0$, portanto só para no máximo metade dos a pode ocorrer

$$\sum_{i=0}^{m-1} |S_i|^2 - n \geqslant 2(2(n-1));$$

ou seja, só para no máximo metade dos a pode ocorrer de a soma exceder o dobro da média. Logo, para no mínimo metade dos a,

$$\sum_{i=0}^{m-1} |S_i|^2 - n < 4(n-1).$$

♦

Corolário 7. *Para no mínimo metade dos a, $h\colon x \mapsto (ax \bmod u) \bmod (2n(n-1)+1)$ opera injetivamente em S.*

Demonstração. $m = 2n(n-1)+1$,

$$\sum \left(\sum |S_i| - n \right) < \frac{2un(n-1)}{2n(n-1)+1} < u.$$

Logo, para no máximo metade dos a a soma pode exceder 2. Logo, para no mínimo metade dos a a soma é menor que 2. ♦

DEFINIÇÃO 8 (FKS melhorado)**.** Agora, a e a_i são escolhidos aleatoriamente até serem apropriados. Como a probabilidade de a ou de algum a_i tomado aleatoriamente ser apropriado é maior que $\frac{1}{2}$, o número esperado de sorteios para se encontrar um a ou um a_i apropriado é no máximo 2. Assim, para achar a e todos os a_i em $[0, n-1]$, gasta-se $O(n)$. Para a tabela D, precisamos de $\sum_{i=0}^{n-1}(2\,|\,S_i\,|\,(|\,S_i\,|-1)+1) = 2\sum|\,S_i\,|^2 - n = 9n\log u$ bits.

1.4 Mais um melhoramento

TEOREMA 9. *Existe um primo $q < n^2 \log u$ tal que $\zeta\colon x \mapsto x \bmod q$ é perfeita para S.*

DEMONSTRAÇÃO. $S = \{x_1, \ldots, x_n\}$. Note que $i \neq j$ implica $x_i \neq x_j$ e $x_i - x_j \neq 0$. Seja $t = \prod_{i<j}(x_i - x_j)\prod_i x_i$. É claro que $|t| \leqslant u^{\binom{n+1}{2}}$ e, portanto, $\log|t| \leqslant \binom{n+1}{2}\log u$. Do teorema dos números primos, para um x,

$$\log\Bigg(\prod_{\substack{q<x\\ q \text{ primo}}} q\Bigg) = x + o(x).$$

Supondo que q divide t para todo primo $q < n^2 \log u$, temos

$$n^2 \log u + o(n^2 \log u) < \binom{n+1}{2}\log u,$$

o que é um absurdo. Logo, existe um $q < n^2 \log u$ tal que q não divide t. Portanto, para esse q, ζ é perfeita. ♦

DEFINIÇÃO 10 (Melhoramento do esquema FKS)**.** Como o pior caso para representar S é $O(nu)$, se $u < n^2 \log u$, $O(nu) = O(n^2 \log u)$. Se $u \geqslant n^2 \log u$, computamos um primo q que dá ζ perfeita. Encontrar o primo é $O(nq) = O(n^3 \log u)$. Agora, não usamos mais x, mas $\zeta(x)$ no lugar. Trocamos o universo $[0, u-1]$ por $[0, q-1]$, para $q < n^2 \log u$. Representar S leva tempo de construção $O(n^3 \log u)$ deterministicamente. Dessa vez, a função *hash* é $h_i \circ h \circ \zeta$. O esquema, cujo espaço de representação queremos que seja $O(n \log n + \log\log u)$, é o seguinte:

1. $\rho\colon S \to [0..n^2-1]$, $\rho\colon x \mapsto (a_\rho x \bmod q) \bmod n^2$ tal que $a_\rho < q < n^2 \log u$ e $S' = \rho(S)$.
2. A *hashing* primária é $h\colon S' \to [0..n-1]$, $h\colon x \mapsto (ax \bmod p) \bmod n$, que conseguimos por causa do corolário 3. $p > n^2 - 1$, $a \in [1, p-1]$, $\sum|\,S_i\,|^2 < 3n$.
3. $h_i\colon S_i \to [0..c_i - 1]$, $c_i = |\,S_i\,|\,(|\,S_i\,|-1)+1$. $h_i\colon x \mapsto (a_i x \bmod p) \bmod c_i$, $a_i \in [1..p-1]$, do corolário 4. O esquema é $x \in S$ ir para $t_i = C_i + h_i(\rho(x))$ numa tabela de chaves de tamanho $3n\log u$ bits $P[0..3n-1]$, ou melhor, $3n\log n$ bits. t_i fornece o índice para a tabela $D[0..n-1]$, que requer só $n\log u$.

a_ρ e q requeremm $2\log n + \log\log u$ bits cada. Para a função h, a e p requerem $4\log n$ bits. Para as h_i, precisamos de tabelas $A[0..n-1]$ para os a_i, $c[0..n-1]$ para os c_i e $C[0..n-1]$ para os C_i: total $3n\log n$. P também precisa de $3n\log n$. O total é $6n\log n + 8\log n + 2\log\log u = O(n\log n + \log\log u)$, além dos $n\log u$ bits de D. O tempo e construção do esquema é $O(n^3 \log u)$, mas a busca é $O(1)$.

Exemplo 11. Seja $S_X = \{\texttt{JAN}, \texttt{FEB}, \texttt{MAR}, \texttt{APR}, \ldots, \texttt{DEC}\}$ e

$$S = \{217 = \texttt{'J'} + \texttt{'A'} + \texttt{'N'}, \ldots, 204 = \texttt{'D'} + \texttt{'E'} + \texttt{'C'}\}$$

(considerando a codificação ASCII). $\rho\colon x \mapsto (14x \bmod 167) \bmod 144$. $S \mapsto S' = \{14, \ldots, 120\}$. $h\colon x \mapsto (4x \bmod 149) \bmod 12$ particiona S' em $S_0, \ldots, S_{11}$. Encontramos os $h_i\colon x \mapsto (a_i x \bmod 149) \bmod c_i$. Note que $S_3 = S_6 = \emptyset$.

i	0	1	2	3	4	5	6	7	8	9	10	11
a_i	1	1	1		1	2		1	1	1	1	2
c_i	1	1	1		1	3		1	1	1	1	3

1.5 Schmidt e Siegel

Lema 12. *Sejam S_j, $j \in [t]$, subconjuntos disjuntos de U, $|U| = u$ e $t \leqslant u$. Então, existe um $a \in U$ para o qual qualquer $h_j\colon U \to M$, $h_j\colon x \mapsto (ax \bmod u) \bmod c_j$, $c_j = 2|S_i|(|S_i| - 1) + 1$, opera injetivamente em metade dos S_j.*

Demonstração. Dado um a, existe colisão entre x e y se $x \neq y$ mas $h_j(x) = h_j(y) = k$, para algum $k \in [0..c_j - 1]$, ou seja, se

$$(ax \bmod u) \bmod c_j = (ay \bmod u) \bmod c_j.$$

Portanto, $a(x-y) \bmod u \in X_1 = \{c_j, 2c_j, 3c_j, \ldots, u - c_j, u - 2c_j, u - 3c_j, \ldots\}$. Assim,

$$\begin{aligned}
&\sum_{a=1}^{u-1} \sum_{j=1}^{t} \sum_{k=0}^{c_j-1} |\{(x,y)\colon x, y \in S_j, x \neq y, h_j(x) = h_j(y) = k\}| \\
&= \sum_{j=1}^{t} \sum_{a=1}^{u-1} \sum_{k=0}^{c_j-1} |\{(x,y)\colon x, y \in S_j, x \neq y, h_j(x) = h_j(y) = k\}| \\
&= \sum_{j=1}^{t} \sum_{\substack{x,y \in S_j \\ x \neq y}} |\{a\colon a(x-y) \bmod u \in X_1\}| \\
&< \frac{2u\binom{|S_j|}{2}}{2|S_j|(|S_j| - 1) + 1} < \frac{ut}{2}.
\end{aligned}$$

Logo, para pelo menos um dos $u - 1$ possíveis a é preciso não haver colisões para menos $\frac{t}{2}$ conjuntos S_j. ♦

Definição 13 (Esquema FKS de Schmidt e Siegel)**.** Usa uma função *hash* composta $h_z \circ h \circ \rho$ como descrita anteriormente, mas:

(a) o espaço para cada *bucket* de colisão S_i é $c_i = 2|S_i|(|S_i| - 1) + 1$;

(b) são guardados apenas $\lfloor \log n \rfloor + 1$ multiplicadores a_z para a função *hashing* secundária — isso porque, do lema 12, existe um a_0 para o qual h_{i0} opera injetivamente para pelo menos metade dos S_i. Para a metade restante, existe um a_1 para o qual h_{i1} opera injetivamente para pelo menos metade dessa metade restante. E, assim por diante, até um a_z que serve cerca de $\frac{1}{2^{z+1}}$ dos S_i.

A grande sacada agora é como representar as tabelas A, c, C e P em espaço $O(n)$ e ainda manter a busca $O(1)$.

Para montar C contendo os $C_i = \sum_{j=0}^{i-1} c_j = \sum j = 0^{i-1}(2 \mid S_j^h \mid (\mid S_j^h \mid -1)+1)$, seja χ_j a *string* unária de c_j 1's que codificamm c_j. As *strings* χ_j são guardadas numa tabela T_0, com um 0 separando duas *strings* consecutivas. Como cada χ_j requer exatamente c_j bits, temos, pelo corolário 3, que o tamanho de T_0 é

$$\sum_{j=0}^{n-1}(\mid \chi_j \mid +1) = \sum_{j=0}^{n-1} c_j + n \leqslant 4n$$

bits. Assumamos então que T_0 é guardado como uma sequencia de palavras de tamanho $\lceil \log(4n) \rceil$ bits. Sabemos que extrair uma subsequencia de bits de uma palavra de tamanho $O(\log n)$ bits é algo que pode ser executado em tempo constante, assim como concatenar duas *strings* de tamanhos $O(\log n)$ bits. Logo, cada bit em T_0 é endereçável.

Dividimos as n *strings* codificadas em T_0 em $\left\lceil \frac{n}{\lceil \log n \rceil} \right\rceil$ grupos de $\lceil \log n \rceil$ strings cada, com a possível exceção do último grupo. Seja λ_i, $0 \leqslant \lambda_i < 4n$, o endereço (índice) do bit inicial da primeira *string* do i-ésimo grupo, ou seja, a *string* $\chi_{i\lceil \log n \rceil}$, sendo $i \in [0..\left\lceil \frac{n}{\lceil \log n \rceil} \right\rceil - 1$.

Analogamente, construímos uma tabela T_1 contendo os λ_i, sendo $i \in [1..\left\lceil \frac{n}{\lceil \log n \rceil} \right\rceil]$, em binário, armazenados como palavras de $\lceil \log(4n) \rceil$ bits. Note que $\lambda_0 = 0$ não é armazenado. O tamanho da tabela T_1 é $\left\lceil \frac{n}{\lceil \log n \rceil} \right\rceil \lceil \log(4n) \rceil = O(n)$ bits. Notemos ainda que

$$\lambda_i = C_{i\lceil \log n \rceil} + i \lceil \log n \rceil .$$

Logo, podemos utilizar λ_i para obter $C_{i\lceil \log n \rceil}$. Assim como extrair uma subsequência de bits de uma palavra de tamanho $O(\log n)$ bits pode ser feito em tempo constante, acessar algumas (um número constante de) constantes de uma palavra de tamanho $O(\log n)$ também pode ser feito em tempo constante. Se $\lambda_{i+1} - \lambda_i \leqslant 2\lceil \log(4n) \rceil$, os endereços dos c_j intermediários, para $i\lceil \log n \rceil \leqslant j < (i+1)\lceil \log n \rceil$, que nos permitem computar os C_j correspondentes, podem ser codificados em tempo $O(1)$ através de acessos às tabelas T_0 e T_1. Se, entretanto, $\lambda_{i+1} - \lambda_i > \lceil \log(4n) \rceil^2$, usamos a tabela T_2.

A tabela T_2 armazena, começando no bit inicial pertinente a λ_i, $\lceil \log n \rceil - 1$ índices binários para os pontos iniciais dos χ_j em T_0, sendo $i\lceil \log n \rceil < j < (i+1)\lceil \log n \rceil$.

Agora, se $2\lceil \log(4n) \rceil < \lambda_{i+1} - \lambda_i \leqslant \lceil \log(4n) \rceil^2$, procedemos da seguinte maneira. Especialmente, cada grupo de *strings* é dividido em $\left\lceil \frac{\lceil \log n \rceil}{\lceil \log \log n \rceil} \right\rceil$ subgrupos, cada um com $\lceil \log \log n \rceil$ *strings*. Denote por $\eta_{i,j}$ o endereço de T_0 do ponto inicial de $\chi_{i\lceil \log n \rceil + j\lceil \log \log n \rceil}$, para $j \in [1..\left\lceil \frac{\lceil \log n \rceil}{\lceil \log \log n \rceil} \right\rceil - 1]$. Assim, $\lambda_i < \eta_{i,j} < \lambda_{i+1}$. Os *offsets* binários $\eta_{i,1} - \lambda_i$, $\eta_{i,2} - \lambda_i$ etc. são armazenados em T_2 como números de $2\lceil \log \log(4n) \rceil$ bits começando pelo local λ_i. Se $\eta_{i,j+1} - \eta_{i,j} \leqslant 2\lceil \log(4n) \rceil$, a informação para o c_k intermediário, $i\lceil \log n \rceil + j\lceil \log \log n \rceil < k < i\lceil \log n \rceil + (j+1)\lceil \log \log n \rceil$, pode ser facilmente decodificada da tabela T_0. Caso contrário, os *offsets* de tamanho $2\lceil \log \log(4n) \rceil$ de todos os c_k intermediários são armazenados numa tabela T_3 de $4n$ bits, começando com o ponto pertinente a $\eta_{i,j}$. Essa última codificação requer $(\lceil \log \log n \rceil - 1) \cdot 2\lceil \log \log(4n) \rceil \leqslant 2\lceil \log(4n) \rceil$ bits.

Exemplo 14. Para $S = \{2, 4, 5, 15, 18, 30\}$, $n = 6$, $u = 31$, codificamos a tabela C da seguinte maneira. Os valores $c_0 = 1$, $c_1 = 0$, $c_2 = 1$, $c_3 = 0$, $c_4 = 3$ e $c_5 = 3$ são armazenados em unário na tabela T_0 de tamanho $4n = 24$ bits. As *strings* de codificação são divididas em $\left\lceil \frac{n}{\lceil \log n \rceil} \right\rceil = \left\lceil \frac{6}{\lceil \log 6 \rceil} \right\rceil = 2$ grupos de $\lceil \log n \rceil = \lceil \log 6 \rceil = 3$ *strings* cada. A tabela T_i contém os índices λ_i, $i \in \{1, 2\}$, para os pontos iniciais das *strings* χ_3 e χ_6 em T_0. Já que tanto $\lambda_1 - \lambda_0$ quanto $\lambda_2 - \lambda_1$ são no máximo $2\lceil \log(24) \rceil = 10$, nenhum nível adicional é necessário.

É fácil ver que não é mais necessário guardar os valores c_i, pois eles podem ser encontrados em tempo constante através da representação de C como descrita acima.

As tabelas P e A podem ser codificadas de uma maneira similar. Em particular, os $a_0, a_1, \ldots, a_{\lfloor \log n \rfloor}$ (lembrar do lema 12) são guardados em um *array* de $\lfloor \log n \rfloor + 1$ palavras. Total: $O((\log n)^2)$ bits. A tabela A_0 contém n *strings* em unário que codificam os índices dos multiplicadores associados a cada *bucket* de colisão. A i-ésima sequência de bits codifica o inteiro $j_i \leqslant \lceil \log n \rceil$ se a_{j_i} é o multiplicador associado ao *bucket* S_i. O primeiro multiplicador codificado pela *string* 0 é usável para pelo menos metade dos *buckets*. O segundo multiplicador, codificado pela *string* 10 é usável para pelo menos $\frac{1}{4}$ dos *buckets* etc. Desse modo, a sequência de bits em A_0 contempla no máximo $\frac{n}{2}$ 0's, $\frac{n}{4}$ 10's etc. Total do tamanho da *string* é no máximo $2n$ bits. Logo, pode-se encontrar o multiplicador associado a cada *bucket* em tempo $O(1)$.

Finalmente, a complexidade do espaço do esquema de Schmidt e Siegel é $O(n + \log\log u)$. O esquema requer $O(n)$ bits para codificar os *offsets* C_i, a tabela P e os multiplicadores a_j, além de $O(2\log n + \log\log u)$ bits para armazenar os parâmetros a_ρ e q da função de pré-processamento. Assim, temos o seguinte teorema.

Teorema 15 (Schmidt e Siegel)**.** [1] *Para um conjunto S de n elementos de um universo $U = \{0, \ldots, u-1\}$, existe uma função hash perfeita de tempo constante com complexidade de espaço $\Theta(n + \log\log u)$.*

2 Hashing perfeita probabilística

2.1 O melhoramento do FKS de Dietzfelbinger

Definição 16 (Dietzfelbinger)**.** Seja $\mathscr{H}^d = \left\{h_a \colon a = (a_0, a_1 \ldots, a_d) \in U^{d+1}\right\}$ em que

$$h_a \colon x \mapsto \left(\sum_{i=0}^{d} a_i x^i\right) \bmod u.$$

Seja $\zeta \colon U \to M$, $\zeta \colon x \mapsto x \bmod m$. $\mathscr{H}_m^d = \left\{\zeta \circ g \colon g \in \mathscr{H}^d\right\}$. Para todo $h \in \mathscr{H}_m^d$ definimos $S_i^h = h^{-1}(i) \cap S$, $i \in M$, e $B_k^h = \sum_{i \in M} \binom{|S_i^h|}{k}$.

[1] Nota: a prova deste teorema não está escrita explicitamente, embora muito dela tenha sido feito nos comentários anteriores. Depois eu pretendo arrumar isso.

Teorema 17. [2] *Para um $S \subseteq U$ fixo e um $h \in \mathscr{H}_m^d$ aleatório, $d \geqslant 3$, $B_2^h < \frac{3|S|^2}{m}$ com probabilidade $1 - O(\left(\frac{|S|^2}{m}\right)^{-\epsilon})$ para algum $\epsilon > 0$.*

O teorema acima é usado para mostrar como o FKS pode ser rodado em tempo linear com probabilidade alta.

1. Tome uma h aleatoriamente de $\mathscr{H}_m^3$, com $m = n = |S|$. Do teorema 17, h é boa com probabilidade $1 - O(\left(\frac{|S|^2}{m}\right)^{-\epsilon}) = 1 - O(n^{-1})$. Portanto, este primeiro passo termina em tempo $O(n)$ com probabilidade $1 - O(n^{-1})$.

2. Para cada *bucket* S_i^h tome aleatoriamente uma função h_i de $\mathscr{H}_{2|S_i|^2}^{1,1}$. Seja t_i a variável aleatória que representa o número de tentativas necessárias para ser enncontrada uma h_i apropriada. Do corolário 7, $\mathbb{P}[t_i = 1] \geqslant \frac{1}{2}$ e, consequentemente, $\mathbb{E}(t_i) \leqslant 2$ e $\mathrm{Var}(t_i) \leqslant 2$. Se T_i é o número de operação necessárias para encontrar h_i apropriada, então $T_i = O(t_i\,|\,S_i\,|)$, $\mathbb{E}(T_i) = O(|\,S_i\,|)$ e $\mathrm{Var}(T_i) = O(|\,S_i\,|^2)$. Tomemos $T = \sum_{i \in M} T_i$. Como as escolhas h_i são independentes, $E(T) = \sum_{i \in M} O(|\,S_i\,|) = O(n)$ — como queríamos — e $\mathrm{Var}(T) = \sum_{i \in M} O(|\,S_i\,|^2) = O(n)$. Portanto, da desigualdade de Chebyshev segue que este segundo passo também termina com $T = O(n)$ com probabilidade $1 - O(n^{-1})$.

2.2 Dietzfelbinger e Meyer auf der Heide

Definição 18. Sendo $r, m \geqslant 1$, $d_1, d_2 \geqslant 1$, $f \in \mathscr{H}_r^{d_1}$, $g \in \mathscr{H}_m^{d_2}$, $a_1, \ldots, a_r \in [m]$ e $h = h(f, g, a_1, \ldots, a_r)$ definida como, para $x \in U$,

$$h(x) = (g(x) + a_{f(x)}) \bmod m,$$

definimos

$$\mathscr{R}(r, m, d_1, d_2) = \{h \colon U \to [m] \mid h = h(f, g, a_1, \ldots, a_r)\}.$$

Convencionamos que $h(f, g, a_1, \ldots, a_r)$ e $h(f', g', a_1', \ldots, a_r')$ são diferentes quando $(f, g, a_1, \ldots, a_r)$ e $(f', g', a_1', \ldots, a_r')$ são diferentes, mesmo que sejam extensionalmente iguais $h(f, g, a_1, \ldots, a_r)$ e $h(f', g', a_1', \ldots, a_r')$.

Se $m = n$, $r = n^{1-\delta}$ para algum $\delta > 0$, $\mathscr{R}(r, m, d_1, d_2)$ pode ser explicada como a seguir. A função $f \in \mathscr{H}_r^{d_1}$ particiona S em r *buckets* $S_i^f = f^{-1}(i) \cap S$, $0 \leqslant i < r$. Entretanto, ao invés de, como no esquema FKS, mapear as chaves do *bucket* S_i^f em um único espaço de tamanho $2|S_i^f|^2$, aplicaremos uma segunda função *hash*: $g(x) + a_i$, com todos os *buckets* compartilhando o alcance comum $[1, m]$.

Teorema 19 (Dietzfelbinger e Meyer auf der Heide)**.** [3] *Tomando um $0 < \delta < 1$ fixo e $r = n^{1-\delta}$ e escolhendo aleatoriamente $f \in \mathscr{H}_r^d$,*

$$\mathbb{P}[|S_i^f| \leqslant 2n^\delta \text{ para todo } i] \geqslant 1 - O(n^{1-\delta-\frac{\delta d}{2}}).$$

[2] Nota: a prova deste teorema não é apresentada no artigo, apenas referenciada. Eu pretendo, no futuro, buscá-la nas referências e acompanhá-la. Mas, ainda não o fiz.

[3] Nota: a prova deste teorema não é apresentada no artigo, apenas referenciada. Eu pretendo, no futuro, buscá-la nas referências e acompanhá-la. Mas, ainda não o fiz.

Isso significa que com $r = n^{1-\delta}$ e $d_1 > \frac{2(1-\delta)}{\delta}$ temos alta probabilidade de todos os *buckets* terem tamanho próximo da média, que é n^{δ}. Para um d_2 suficientemente grande e $m = O(n)$, a segunda função *hash* G mapeia cada $x \in S_i^f$ para um único endereço, novamente com alta probabilidade.

2.3 FKS com grafos aleatórios

Podemos enxergar o método FKS como um mapeamento de S para uma floresta de estrelas de núcleos "pequenos". Uma estrela é uma árvore onde todos os vértices, com exceção do núcleo, têm grau 1. A função primária h mapeia cada x no núcleo de uma estrela do grafo (a floresta de estrelas). A função secundária h_i mapeia o núcleo para um de seus vizinhos.

Problema 20 (Problema da associação perfeita (Czech))**.** *Dado um $G = (V, E)$, $|V| = \nu$ e $|E| = \mu$, encontrar uma função $g\colon V \to [0..\mu - 1]$ tal que $h\colon E \to [0..\mu - 1]$ definida como*

$$h(e = \{a, b\}) = (g(a) + g(b)) \bmod \mu$$

é uma bijeção.

Para um grafo acíclico, a solução para esse problema é um procedimento simples em tempo ótimo:

1. Distribua os $h(e) = 0, \ldots, \mu - 1$ para as arestas e em qualquer ordem.
2. Para cada componente conexa de G escolha um vértice v e atribua $g(v) = 0$ e:
   (a) Faça uma busca em profundida no grafo.
   (b) Para cada vértice b alcançado por um vértice a (tal que o valor associado a $e = \{a, b\}$ é $h(e)$), atribua $g(b) = (h(e) - g(a)) \bmod \mu$.

Agora, para encontrar a f que associa os x às estrelas, procedemos do seguinte modo (chamado de "fase de mapeamento"):

1. Escolha duas funções *hash* independentes aleatoriamente, f_1 e f_2, de domínio U e contradomínio $[0..\nu - 1]$.
2. Para cada $x \in S$, se $f_1(x) = f_2(x)$, modifique $f_2(x)$ acrescentando um número aleatório em $[1..\nu - 1]$ (isso para evitar *loops*);
3. S agora está correspondido com um grafo

   $$G = (V = [0..\nu], E = \{\{f_1(x), f_2(x)\} : x \in S\}).$$

   Então, testa se G é acíclico. Senão, joga fora f_1 e f_2 e pega outras até ter G acíclico.

Observe que sortear as f_1 e f_2 é rápido. Testar se G é acíclico também é rápido. Ademais, a quantidade de grafos acíclicos é grande no espaço amostral. Se f_1 e f_2 são computáveis em tempo $O(1)$, gerar o grafo G correspondente é $O(\mu)$ (busca) e verificar se ele é acíclico é $O(\nu + \mu)$.

Estratégia: precisamos ter $\mu = n = |S|$ e $\nu = O(\mu)$. Verificar essas duas condições leva tempo similar. Porém, para $\nu = c\mu$, para alguma constante c, a probabilidade um grafo aleatório ser acíclico continua grande o suficiente? Nessas novas condições, os grafos acíclicos dominam o espaço de todos os grafos aleatórios?

TEOREMA 21 (Havas)**.** *Seja G um grafo aleatório com ν vértices e μ arestas obtido escolhendo aleatória e independentemente as μ arestas com repetição. Então, se $\nu = c\mu$ para $c > 2$, a probabilidade de G ser acíclico, quando $\nu \to \infty$, é*

$$p = \mathrm{e}^{\frac{1}{c}} \sqrt{\frac{c-2}{c}}.$$

DEMONSTRAÇÃO. Erdös e Rényi mostraram[4] que a probabilidade de um grafo aleatório não ter ciclos, à medida que $\nu = c\mu$ tende ao infinito, é

$$\mathrm{e}^{\frac{1}{c}+\frac{1}{c^2}} \sqrt{\frac{c-2}{c}}.$$

Porém, em nosso caso, os grafos gerados no passo de mapeamento podem ter arestas múltiplas (não *loops*). A probabilidade procurada p, então, é a probabilidade calculada por Erdös e Rényi vezes a probabilidade de não haverem arestas múltiplas. Sabemos que a j-ésima aresta é única com probabilidade

$$\frac{\binom{\nu}{2} - j + 1}{\binom{\nu}{2}}.$$

Logo, a probabilidade de todas as arestas serem únicas é

$$\prod_{j=1}^{\mu} \frac{\binom{\nu}{2} - j + 1}{\binom{\nu}{2}} = \mathrm{e}^{-\frac{-1}{c^2}+o(1)}$$

Finalmente, $p = \left(\mathrm{e}^{\frac{1}{c}+\frac{1}{c^2}} \sqrt{\frac{c-2}{c}}\right)\left(\mathrm{e}^{-\frac{-1}{c^2}+o(1)}\right) = \mathrm{e}^{\frac{1}{c}} \sqrt{\frac{c-2}{c}}$. ♦

O algoritmo é realmente bem eficiente. Para uma constante c tão pequena quanto $2,09$, por exemplo, o número esperado de execuções (de tentativas para G) é menor (estritamente) que 3.

EXEMPLO 22. $S = \{53, 59, 61, 67, 71, 73, 79, 83, 89, 97\}$ o conjunto de todos os primos maiores que 50 e menores que 100. Encontramos o acíclico G com as funções:

$$f_1(x) = ((34x + 11) \bmod 101) \bmod 21;$$
$$f_2(x) = ((29x + 18) \bmod 101) \bmod 21.$$

Sortear uma função dessas é sortear os coeficientes a_1 e a_0 no conjunto $[1..101 - 1]$. Assim, a correspondência entre S e G fica como na tabela abaixo:

53	59	61	67	71	73	79	83	89	97
$\{12, 19\}$	$\{12, 14\}$	$\{2, 7\}$	$\{4, 0\}$	$\{1, 15\}$	$\{14, 6\}$	$\{8, 3\}$	$\{5, 1\}$	$\{7, 11\}$	$\{3, 14\}$

[4]No futuro, também esta é uma demonstração que eu pretendo acompanhar.

Decidimos armazenar o i-ésimo primo na $(i-1)$-ésima posição. Ex: 83, o $8º$ primo, recebe endereço 7. Ou seja, $h(\{5,1\}) = 7$. No segundo passo, considerando uma componente conexa de cada vez, resolvemos o problema da associação perfeita valorando os g. Para checar se 59 é primo, computamos $f_1(59) = 14$ e $f_2(59) = 12$. Depois, $h(\{14,12\}) = 1$. Checando o endereço 1 na tabela, confirmamos que 59 é primo.

2.4 Métodos com hipergrafos

Problema 23 (Problema da associação perfeita para hipergrafos)**.** *Dado um r-hipergrafo $G = (V, E)$ em que $E \subseteq \binom{V}{r}$ e $|V| = \nu$ e $|E| = \mu$, encontrar uma função $g\colon V \to [0..\mu-1]$ tal que, para $h\colon E \to [0..\mu-1]$ definida como*

$$h(e = \{v_1, \ldots, v_r\}) = (g(v_1) + \cdots + g(v_r)) \bmod \mu$$

é uma bijeção.

Do mesmo modo que com 2-hipergrafos, somente r-hipergrafos acíclicos têm a garantia de ter uma solução em tempo linear para o problema da associação perfeita.

Definição 24. Um r-hipergrafo é acíclico se não contém subgrafo com grau mínimo 2.

Outra definição equivalente:

Definição 25. Um r-hipergrafo é acíclico se e só se há algum modo de retirar sucessivamente arestas contendo vértices de grau 1 e, no final, ficar com um grafo sem arestas.

Então, para verificarmos se um hipergrafo é acíclico, faremos o seguinte:

1. Inicialmente, marque todas as arestas como "não removidas".
2. Visite todos os vértices do grafo, cada vértice apenas uma vez:
   (a) Se v tem grau 1, remova do grafo a aresta e que levou a v.
   (b) Cheque se algum dos outros vértices da aresta removida possui grau 1. Se sim para algum vértice v', remova a única aresta e' para a qual v' pertence.
   (c) Repita isso recursivamente até que não sejam mais possíveis deleções de arestas. Para tanto, utilize uma pilha.
3. Se o grafo ainda contém arestas, então ele não é acíclico. Senão, é acíclio.

Teorema 26. *O teste apresentado acima leva tempo $O(r\mu + \nu)$.*

Demonstração. Seja o r-grafo $G = (V, E)$ e seja o 2-grafo bipartido $H = (V_1 \cup V_2, E')$ tal que $V_1 \cap V_2 = \emptyset$, $V_1 = V$, $V_2 = E$ e

$$E' = \{\{a, b\} : a \in V, b \in E, a \in b\}.$$

O teste para verificar se G é acíclico pode ser visto como um passeio em H. Cada vértice de V_1 é testado no mínimo uma vez. Uma vez que um vértice de V_2 é deletado, vamos para um outro vértice de H através de uma aresta $\{e, u\}$, onde $e \in V_2$ e $u \in V_1$ — e nunca mais $\{e, u\}$ vai ser usada. Assim, o número de testas executados sobre vértices de V_1 é no máximo $\sum_{v \in V_1} \mathrm{d}_H(v)$. Como acessamos vértices em V_2 apenas uma vez, todo o processo toma no máximo

$$|\, V_1 \,| + |\, V_2 \,| + \sum_{v \in V_1} \underset{H}{\mathrm{d}}(v) = \nu + \mu + 2\frac{\mu r}{2} = \nu + \mu(r+1).$$

♦

Procedimento para obter uma solução genérica para o problema da associação para hipergrafos acíclicos:

1. Atribua um único número $h(e) \in \{0, \ldots, \mu - 1\}$ para as arestas $e \in E$ em qualquer ordem, sem repetição.
2. Considere as arestas e na ordem inversa à ordem de deleção no teste do grafo acíclico:
   (a) Seja $\{v_1, \ldots, v_j\}$ o conjunto dos vértices de e que ainda não receberam g e que são exclusivos daquela aresta naquele momento (grau 1).
   (b) Atribua 0 para $g(v_2), \ldots, g(v_j)$.
   (c) Atribua
   $$g(v_1) = \left(h(e) - \sum_{i=2}^{r} g(v_i) \right) \bmod \mu.$$

Teorema 27. *G r-hipergrafo aleatório com $r \geqslant 1$. Se $\mu \leqslant \frac{\nu}{c_r}$, onde*

$$c_r = \begin{cases} \Omega(\mu) & \text{para } r = 1, \\ 2 + \epsilon, \epsilon > 0 & \text{para } r = 2, \\ r\left(\max\limits_{y>0} \left\{ \dfrac{y}{(1 - \mathrm{e}^{-y})^{r-1}} \right\} \right)^{-1} & \text{caso contrário.} \end{cases}$$

então o espaço amostral dos r-hipergrafos aleatórios se torna dominado de r-hipergrafos acíclicos.

Demonstração. Para $r = 2$ já provamos. Para $r = 1$, como f_1 é totalmente aleatório, a i-ésima chave é mapeada para um local único com probabilidade $\frac{\nu - i + 1}{\nu}$. Assim, a probabilidade de todas as μ chaves serem mapeadas para locais únicos é

$$p_1 = \nu^{-\mu} \prod_{i=1}^{\mu} (\nu - i + 1) \sim \mathrm{e}^{-\frac{\mu^2}{2\nu} - \frac{\mu^3}{6\nu^2}}.$$

Se $\nu = O(\mu^2)$, o expoente de e é $O(1)$, e, daí, p_1 se torna uma constante não nula.

Para $r \geqslant 3$, seja H um r-grafo escolhido aleatoriamente. $V_0 = V(H)$. Para $j \geqslant 0$, V_{j+1} é o conjunto dos vértices de V_j que têm grau no mínimo 2 em H_j, que é o grafo definido pela restrição de H a V_j. Esse processo termina quando $V_{j+1} = V_j = V_\infty$ para algum j. H tem um subgrafo de grau mínimo no mínimo 2

se e só se $V_\infty \neq \emptyset$. Suponha que H foi selecionado escolhendo-se aleatoriamente as arestas independentemente com probabilidade $\frac{\alpha(r-1)!}{\nu^{r-1}}$ para alguma constante α. Defina

$$p_{i+1} = (1 - \mathrm{e}^{-\alpha p_i})^{r-1}$$
$$q_{i+1} = (1 - \mathrm{e}^{-\alpha p_i})^{r}.$$

Pittel[5] mostrou que quase certamente $|V_j| \sim q_j \nu$ à medida que $\nu \to \infty$ com j fixo. Então, se $\lim_{j\to\infty} q_j = 0$ então para todo $\epsilon > 0$ temos quase certamente $|V_j| < \epsilon\nu$ para j suficientemente grande. Uma outra demonstração[6] mostra que, nesse caso, é quase certeza que não há subgrafo algum de no mínimo grau 2.

Quando $p_i = p_{i+1} = p$, $p = (1 - \mathrm{e}^{-\alpha p})^{r-1}$. Assim,

$$\alpha = \frac{\alpha p}{(1 - e^{-\alpha p})^{r-1}} = \frac{y}{(1 - e^{-y})^{r-1}}$$

para algum $y > 0$. O menor valor para essa fração é o menor α para o qual $\lim_{j\to\infty} q_j \neq 0$. ♦

Curiosidade (valores aproximados):

r	2	3	4	5
c_r	2.09	1.221793133	1.294867415	1.424947449

Note o valor mínimo (pico) para $r = 3$.

Clareando um pouco melhor as coisas: Na fase de mapeamento queremos corresponder S a um r-grafo $G = (V, E)$ onde $V = [0..\nu - 1]$,

$$E = \{\{f_1(x), f_2(x), \ldots, f_r(x)\} : x \in S\}$$

e $f_i \colon U \to [0..\nu - 1]$. $\mu = n = |S|$. $\nu = c_r\mu = c_r n$. O passo é repetido até que G seja acíclico. Daí, é executado o passo de mapeamento, que consiste em gerar uma *hash* perfeita mínima. Para tanto, associamos $h(e) = i - 1$ se x é a i-ésima chave de S, já que cada aresta $e = \{v_1, \ldots, v_r\}$ corresponde unicamente a alguma chave x, pois $f_i(x) = v_i, 1 \leqslant i \leqslant r$. Os valores para g são computados pelo passo de associação[7], que resolve o problema da associação para G.

Sortear verdadeiramente aleatoriamente todas as f_i pode não ser uma boa ideia. Ao invés disso, tomamos a classe $\mathscr{H}$ de funções *hash* tal que $\mathscr{H}$ é abundante com funções *hash* "de boa qualidade", que podem ser selecionadas rapidamente. Assim, ao invés de fixar uma função *hash* e usá-la para todas as possíveis entradas, selecionaremos um elemento aleatório de $\mathscr{H}$ antes de cada rodada. Assim, o tempo de execução médio sobre muitas rodadas é esperado ser perto de ótimo. Cada "rodada" é uma iteração da fase de mapeamento. Para[8] $S \subseteq \mathbb{Z}$, $f_i \in \mathscr{H}_v^d$ para qualquer d. Ou $f_i \in \mathscr{R}(r, s, d_1, d_2)$.

[5] Outra demonstração que no futuro eu vou acompanhar... :P

[6] Nem preciso falar nada... :P

[7] Preciso arranjar traduções boas para essas coisas, para eliminar algumas ambiguidades.

[8] O artigo ainda traz algumas considerações especiais sobre chaves que são cadeias de caracteres.

2.5 Modificações

Teorema 28. *G r-grafo acíclico, $|V| = \nu$, $|E| = \mu$, $h\colon E \to M = [0..m-1]$ não necessariamente injetiva mais. Então existe ao mesmos uma $g\colon V \to M$ tal que, para toda aresta $e = \{v_1, \ldots, v_r\}$,*

$$h(e) \equiv \left(\sum_{v_i \in e} g(v_i) \right) \mod m.$$

Note que, segundo esse teorema, não precisamos mais da restrição de $h(e_1) \neq h(e_2)$ para resolver o problema da associação perfeita. Assim, h pode ser encontrado através de g em tempo ótimo.

Depois do aqui o autor entre numa série de casos e exemplos específicos. Acho que seria exaustivo esboçá-los aqui. Um dos casos é a utilização de *hashing* para implementar conjuntos (com operações clássicas como união, interseção etc.) e operações sobre conjuntos. Para este último caso, ele enuncia o seguinte teorema, de fácil demonstração:

Teorema 29. *Sendo M um conjunto e $*$ uma operação binária sobre M, $*$ é apropriada para uma função hash $h\colon M^r \to M$, r inteiro positivo, se $(M, *)$ é um grupo.*

A seção 6 do capítulo é sobre aplicações avançadas de *hashing* e como cada técnica discutida no capítulo se aplicaria para cada caso. Uma interessante leitura.