Tópicos em Recuperação de Informação[1]

Nivio Ziviani

[1]Conjunto de transparências elaborado por Nivio Ziviani, Patrícia Correia e Fabiano C. Botelho

Hashing Perfeito

Uma função de transformação (*hashing function*) transforma uma conjunto de n chaves x_j em um conjunto de valores inteiros no intervalo:

$$0 \leq h(x_j) \leq m - 1$$

com duplicações (ou colisões) permitidas.

Exemplo de função:

$$h(x) = x \bmod m$$

onde,

x: inteiro $\qquad m > \frac{n}{\alpha}, \qquad \alpha = \dfrac{\#\text{registros}}{\text{espaços disponíveis}}$

Ex.: $n = 100$ *chaves inteiras*

$h(x) = x \bmod 143$

$\alpha = 0.7$

$m = 143$

Hashing Perfeito

- Quando $h(x_i) = h(x_j)$ sse $i = j$

$$\Downarrow$$

Função Hashing Perfeita (PHF)

  - Neste caso não há colisões.

- Quando $m = n$ $(\alpha = 1.0)$

$$\Downarrow$$

Função Hashing Perfeita Mínima (MPHF)

  - 1 acesso.

  - Não há lugares vazios na tabela.

Hashing Perfeito

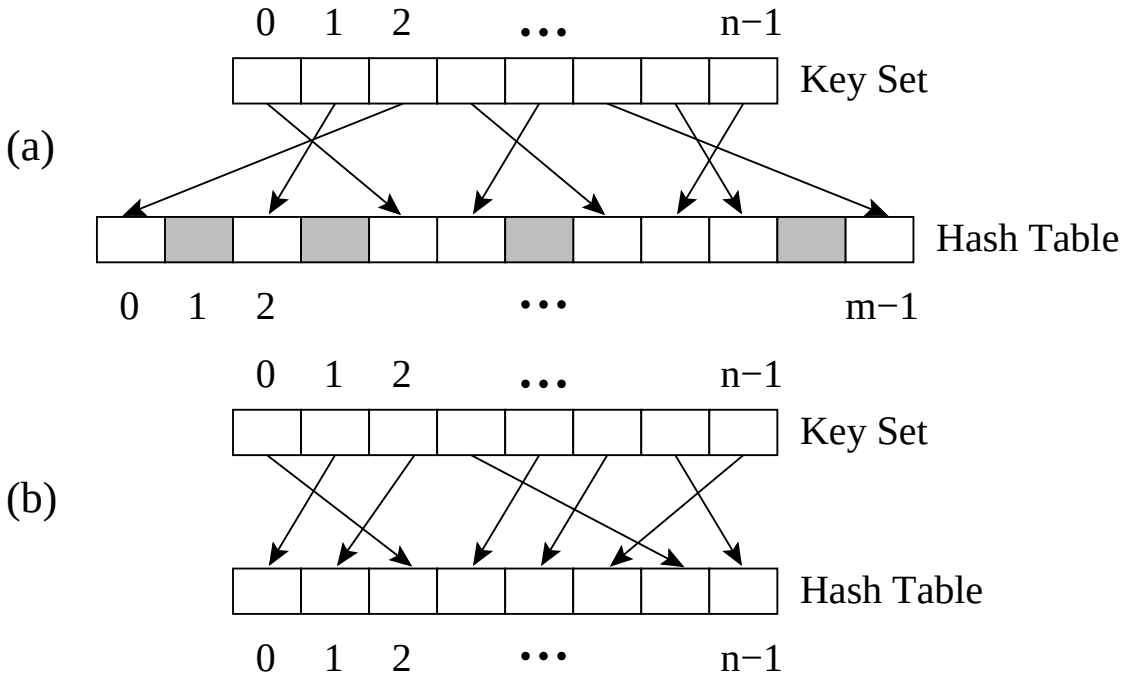


Figura 1: (a) Perfect hash function. (b) Minimal perfect hash function.

Hashing Perfeito

- Quando $x_i < x_j$ e $h(x_i) < h(x_j)$ a ordem é preservada.

$$\Downarrow$$

MPHF com Ordem Preservada (OPMPHF)

  - Chaves são localizadas em $O(1)$.
  - Não há espaço vazio.
  - Processamento na ordem lexicográfica.

Paradoxo do Aniversário

Quantas pessoas têm que ser colocadas juntas para que a probabilidade de que 2 pessoas façam aniversário no mesmo dia seja maior que 50%?

A probabilidade se inserir n itens consecutivos sem colisão é:

$$\frac{m-1}{m} \times \frac{m-2}{m} \times \ldots \times \frac{m-n+1}{m} = \prod_{i=1}^{n} \frac{m-i+1}{m} = \frac{m!}{(m-n)!m^n}$$

n	p
10	0.883
22	0.524
23	0.493
30	0.303

$p \approx \frac{n(n-1))}{730}$ para n pequeno

$n = 10 \Rightarrow p \approx 87.7\%$

Hashing Perfeito

Qual a vantagem da MPHF h ?

- Não há necessidade de armazenar os termos t do vocabulário para buscas com sucesso. Por exemplo, na indexação

- Para qualquer termo t basta armazenar na posição da tabela hash $h(t)$:
  - endereço do arquivo invertido.
  - f_t ($\#$ de documentos onde o termo t aparece.)

Desvantagem:

- Espaço ocupado para descrever a função h.

Entretanto, é possível obter métodos nos quais o espaço ocupado pelas PHFs e MPHFs é aproximadamente 1.95 e 2.6 bits/chave, respectivamente.

Limite Inferior de Espaço para MPHFs

Número de funções hash n_h que mapeiam n elementos em n entradas de uma tabela hash:

$$n_h = n^n$$

Número de funções hash perfeitas mínimas n_{mp} que mapeiam n elementos em n entradas de uma tabela hash:

$$n_{mp} = A_n^n = \frac{n!}{(n-n)!} = n!$$

A probabilidade de se obter uma MPHF é:

$$Pr_{mp} = \frac{n_{mp}}{n_h} = \frac{n!}{n^n} = e^{\log n! - n \log n} \approx e^{-n}$$

Aproximação de Stirling usada: $\log n! \approx n \log n - n$

Número mínimo de funções hash geradas para se obter uma MPHF: $1/Pr_{mp} = e^n$

Portanto, uma MPHF requer pelo menos $(\log e)\, n \approx 1.4427n$ bits de espaço, aproximadamente, isto é, $\Omega(n)$ bits.

Hashing Perfeito

Algoritmo de Czech, Havas e Majewski

- Czech, Havas e Majewski (1992, 1997) propõem um método elegante baseado em **grafos randômicos** para obter uma OPMPHF.
- A função de transformação é do tipo:

$$h(t) = (g(h_1(t)) + g(h_2(t))) \bmod n,$$

na qual $h_1(t)$ e $h_2(t)$ são duas funções não perfeitas, t é a chave de busca, e g um arranjo especial que mapeia números no intervalo $0 \ldots m-1$ para o intervalo $0 \ldots n-1$.

Hashing Perfeito

Funções não perfeitas:

$$h_j(t) = (\sum_{i=1}^{|t|} t[i] \times p_j[i] \ ) \bmod m, \ \ j = 1, 2$$

ou

$$h_j(t) = (\sum_{i=1}^{|t|} T_j[i, t[i]] \ ) \bmod m, \ \ j = 1, 2$$

onde:

$t[i]$: i-ésimo caractere do termo t na base

$$36 = \begin{cases} 0 - 9 & \text{digitos 0 - 9} \\ 10 - 35 & \text{caracteres a - z} \end{cases}$$

$| \ t \ |$: número de caracteres do termo t

Hashing Perfeito

Problema Resolvido Pelo Algoritmo

- Dado um grafo não direcionado $G = (V, A)$, onde $|V| = m$ e $|A| = n$, encontre uma função $g : V \rightarrow [0, n-1]$, que transforme $h(a = (u, v) \in A) = (g(u) + g(v)) \bmod n$ em uma bijeção.
- Em outras palavras, estamos procurando uma atribuição de valores aos vértices de G tal que a soma dos valores associados aos vértices de cada aresta tomado módulo n é um número único no intervalo $[0, n-1]$.
- A questão principal é como obter uma função g adequada. A abordagem mostrada a seguir é baseada em grafos e hipergrafos randômicos.

Hashing Perfeito

Exemplo

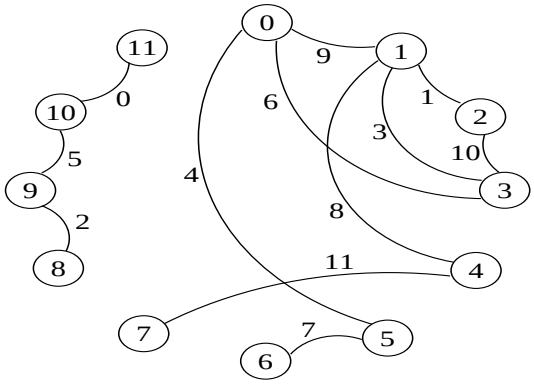
- **Chaves**: 12 meses do ano abreviados para os três primeiros caracteres.
- **Objetivo**: obter uma função de transformação perfeita h de tal forma que o i-ésimo mês é mantido na $(i-1)$-ésima posição da tabela $hash$:

Termo t	$h_1(t)$	$h_2(t)$	$h(t)$
jan	10	11	0
fev	1	2	1
mar	8	9	2
abr	1	3	3
mai	0	5	4
jun	10	9	5
jul	0	3	6
ago	5	6	7
set	4	1	8
out	0	1	9
nov	3	2	10
dez	4	7	11

Hashing Perfeito

Grafo Randômico gerado

- O problema de obter a função g é equivalente a encontrar um grafo não direcionado contendo m vértices e n arestas.



- Os vértices são rotulados com valores no intervalo $0 \ldots m-1$
- As arestas são definidas por $(h_1(t), h_2(t))$ para cada um dos n termos t.
- Cada termo corresponde a uma aresta que é rotulada com o valor desejado para a função h perfeita.
- Os valores das duas funções $h_1(t)$ e $h_2(t)$ definem os vértices sobre os quais a aresta é incidente.

Hashing Perfeito

Obtenção da Função g a Partir do Grafo

- **Passo importante**:
  conseguir um arranjo g de vértices para inteiros no intervalo $0 \ldots n-1$ tal que, para cada aresta $(h_1(t), h_2(t))$, o valor de $h(t) = g(h_1(t)) + g(h_2(t))) \bmod n$ seja igual ao rótulo da aresta.

Hashing Perfeito

Algoritmo para obter g:

1. **Para cada** $v \in V$ **faça**
   $g[v] \leftarrow desconhecido$.
2. **Para cada** $v \in V$ **faça**
   Se $g[v] = desconhecido$ **então**
   $Rotule(v, 0)$.

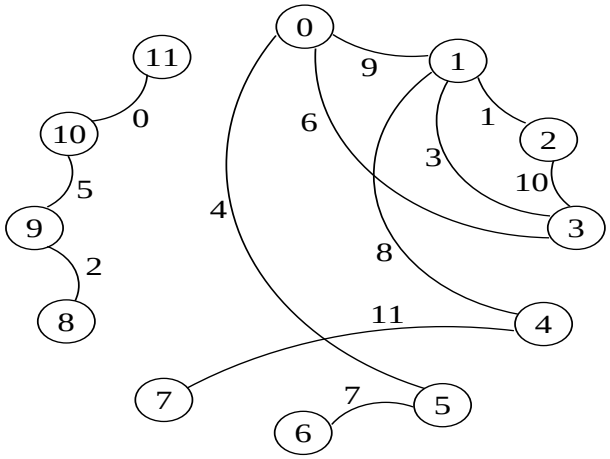
Onde a função $Rotule(v, c)$ é definida por:

1. **Se** $g[v] \neq desconhecido$ **então**
   Se $g[v] \neq c$ **então**
   Retorne *Falha: o grafo cíclico não pode ser rotulado.*
   Senão
   Retorne *Este vértice já foi visitado.*
2. $g[v] \leftarrow c$.
3. **Para cada** $u \in Adjacente(v)$ **faça**
   $Rotule(u, (h((v, u)) - g[v]) \bmod n)$.

Adjacente(v): vértices que compartilham um arco com v.
$h((v, u))$: rótulo associado com o arco (v, u).

Hashing Perfeito

Aplicando o Algoritmo no Exemplo



	Termo t	$h_1(t)$	$h_2(t)$	$h(t)$		$v:$	$g(v)$
	jan	10	11	0		0	0
	fev	1	2	1		1	9
	mar	8	9	2		2	4
	abr	1	3	3		3	6
	mai	0	5	4		4	11
(a)	jun	10	9	5	(b)	5	4
	jul	0	3	6		6	3
	ago	5	6	7		7	0
	set	4	1	8		8	0
	out	0	1	9		9	2
	nov	3	2	10		10	3
	dez	4	7	11		11	9

Hashing Perfeito

Geração da OPMPHF:

1. Escolha um valor para m.

2. Escolha os pesos $p_1[i]$ e $p_2[i]$ ou $T_1[i,j]$ e $T_2[i,j]$ para $1 \ll i \ll max_t \mid t \mid$ e $1 \leq j \leq |\Sigma|$, sendo Σ o alfabeto utilizado para compor cada termo t.

3. Gerar o grafo $G = (V, E)$ onde:
   $V = \{1, \ldots, m\}$ e
   $E = \{(h_1(t), h_2(t)) \mid t \in L\}$

4. Tente obter o mapeamento g.

5. Se houve falha, retornar ao passo 2.

6. Retornar $p_1[i]$, $p_2[i]$ (ou $T_1[i,j]$, $T_2[i,j]$) e g.

Hashing Perfeito

Problema

- **Quando o grafo contém ciclos**: o mapeamento a ser realizado pode rotular de novo um vértice já processado e que tenha recebido outro rótulo com valor diferente.
- Por exemplo, se a aresta $(5, 6)$, que é a aresta de rótulo 7, tivesse sido sorteada para a aresta $(8, 11)$, o algoritmo tentaria atribuir dois valores distintos para o valor de $g[11]$.
- Para enxergar isso, vimos que se $g[8] = 0$, então $g[11]$ deveria ser igual a 7, e não igual ao valor 9 obtido acima.
- Um grafo que permite a atribuição de dois valores de g para um mesmo vértice, não é válido.
- Grafos acíclicos não possuem este problema.
- Um caminho seguro para se ter sucesso é obter antes um grafo acíclico e depois realizar a atribuição de valores para o arranjo g. Czech, Havas e Majewski (1992).

Teoria de Grafos Randômicos

- Para $m < 2n$
  - Prob. de gerar grafo acíclico $\longrightarrow$ 0, quando n cresce.

- Para $m = cn$, sendo $c > 2$
  - Prob. de um grafo randômico de m vértices e n arestas ser acíclico é aproximadamente:

$$P_a = e^{1/c}\sqrt{\frac{c-2}{c}}$$

- O número esperado de grafos gerados até encontrar o primeiro acíclico é:

$$N_i = \frac{1}{e^{1/c}\sqrt{\frac{c-2}{c}}}$$

Ex.: para $m = 3n$

$N_i \approx 1.24$

- Problema: espaço $m = 3n$ é muito!

Hashing Perfeito

Solução para reduzir m:

- Não utilizar grafos tradicionais, mas sim **hipergrafos**, ou r-grafos, nos quais cada aresta conecta um número qualquer r de vértices.
- Para tanto, basta usar uma terceira função h_3 para gerar um trigrafo com arestas conectando três vértices, chamado de 3-grafo.
- Em outras palavras, cada aresta é uma tripla do tipo $(h_1(t), h_2(t), h_3(t))$, e a função de transformação é dada por:

$$h(t) = (g(h_1(t)) + g(h_2(t)) + g(h_3(t))) \bmod n.$$

Hashing Perfeito

Solução para reduzir m:

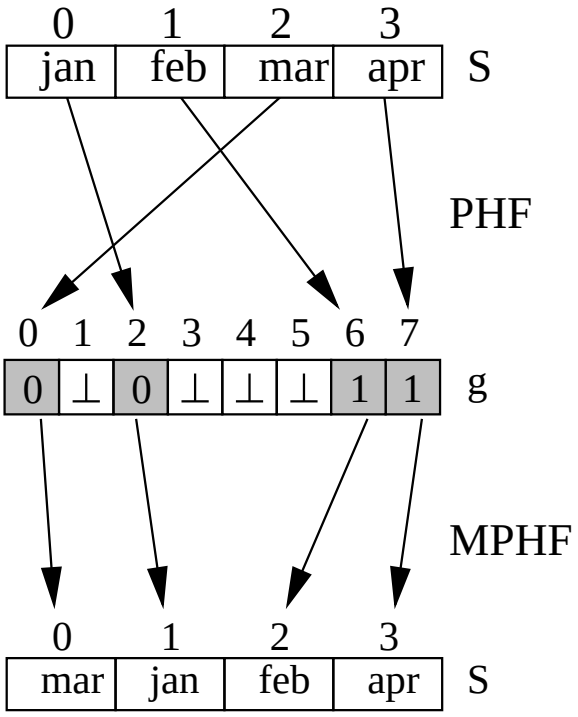
- Nesse caso, o valor de m pode ser próximo a $1.23n$.
- Logo, o uso de trigrafos reduz o custo de espaço da função de transformação perfeita, mas aumenta o tempo de acesso ao vocabulário.
- Além disso, o processo de rotulação não pode ser feito como descrito.
- Ciclos devem ser detectados previamente, utilizando a seguinte propriedade de r-grafos:

  Um r-grafo é **acíclico** se e somente se a remoção repetida de arestas contendo apenas vértices de grau 1 (isto é, vértices sobre os quais incide apenas uma aresta) elimina todas as arestas do grafo.
- Neste caso, para $m > 1.23n$ o número de tentativas para para gerar o grafo é O(1).

- Conjunto com $3,541,615$ URLs: $<$ 11 s e o espaço ocupado pela função foi $26.76n$ bits.

Hashing Perfeito

Algoritmo de Botelho, Pagh e Ziviani



mphf(jan) = rank(phf(jan)) = 1

mphf(feb) = rank(phf(feb)) = 2

mphf(mar) = rank(phf(mar)) = 0

mphf(apr) = rank(phf(apr)) = 3

- $\perp$ representa valores não assinalados
- g é uma função especial apresentada a seguir
- rank é uma função que conta quantas entradas estão assinaladas antes de uma determinada entranda i em g.

Hashing Perfeito

Construção de uma PHF em dois passos:

1. Mapeamento
   - Gerar um grafo randomico bipartido $G = G(h_0, h_1) = (V, E)$, onde $V = [0, m-1]$, $E = \{\{h_0(t), h_1(t) + \nu\}/x \in S\}$ e $\nu = m/2$.
   - Testar se o grafo não contém ciclos. Um grafo é acíclico se, e somente se, alguma sequência de remoções de arestas contendo vértices de grau 1 gera um grafo $G' = (V', E')$ onde $V' = V$ e $E = \emptyset$.
   - Caso o grafo contenha ciclos, gerar um novo par de funções (h_0, h_1) e reiniciar o Mapeamento.
2. Assinalamento: geração da função $g : V \rightarrow \{0, 1, \perp\}$, onde o valor $\perp = 2$ é usado para representar vértices não assinalados.

A PHF gerada tem a seguinte forma: phf$(t) = h_{\phi(t)}(t) + \phi(t)\nu$, onde $\phi(t) = (g(h_0(t)) + g(h_1(t) + \nu)) \bmod 2$.

Hashing Perfeito

Funções não perfeitas h_0 e h_1:

$$h_j(t) = (\sum_{i=1}^{|t|} t[i] \times p_j[i]) \bmod \nu, \ j = 0, 1$$

ou

$$h_j(t) = (\sum_{i=1}^{|t|} T_j[i, t[i]]) \bmod \nu, \ j = 0, 1$$

onde:

$t[i]$: i-ésimo caractere do termo t na base

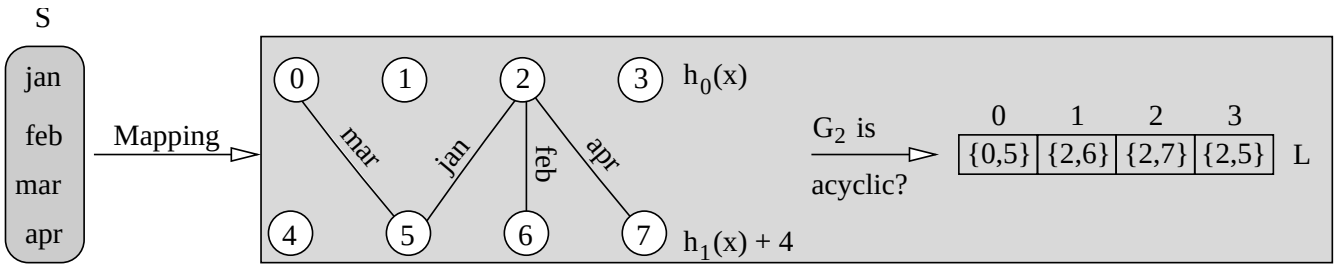
$$36 = \begin{cases} 0 - 9 & \text{digitos 0 - 9} \\ 10 - 35 & \text{caracteres a - z} \end{cases}$$

$| \ t \ |$: número de caracteres do termo t

Hashing Perfeito

Mapeamento

Exemplo:



Para testar se G contém ciclos em tempo $O(n)$, utiliza-se o seguinte algoritmo:

1. Percorra as arestas de G e armazene em uma fila Q toda aresta que tem pelo menos um vértice de grau 1.
2. Enquanto Q não for fazia, retire uma aresta de Q, remova-a de G, armazene-a em uma lista L, verifique se algum vértice v da aresta é agora de grau 1. Se for o caso, insira em Q a única aresta que contém v.

Hashing Perfeito

Assinalamento

- Constroi a função $g : V \rightarrow \{0, 1, \perp\}$.
- Objetivo: atribuir valores aos vértices de G de forma que cada aresta associada a uma chave em S seja representada unicamente por um de seus vértices.

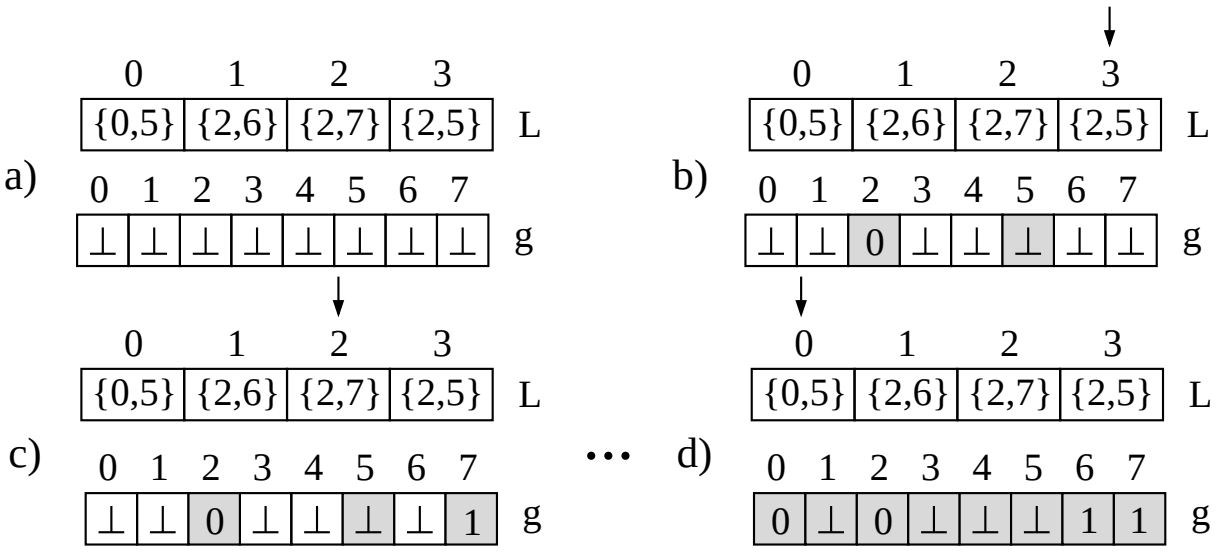
Algoritmo:

1. Percorra L na ordem reversa (fim para início). Por que?
2. Seja $e \in L$ a aresta que está sendo processada, a qual está associada a uma chave $x \in S$.
3. Divida e em dois subconjuntos A (vértices assinalados) e N_a (vértices não assinalados).
4. Seja u_{j_0} o primeiro vértice em N_a, onde j_0 é o índice de u na aresta e.
5. Faça $g(u_{j_0}) = (j_0 - \sum_{v \in A} g(v)) \bmod 2$. Isto assegurará que $\phi(x) = j_0$ e phf(x) retornará $h_{j_0}(x) + j_0\nu = u_{j_0}$ como resultado.
6. Marque os vértices de e como processados.

Hashing Perfeito

Assinalamento

Exemplo:



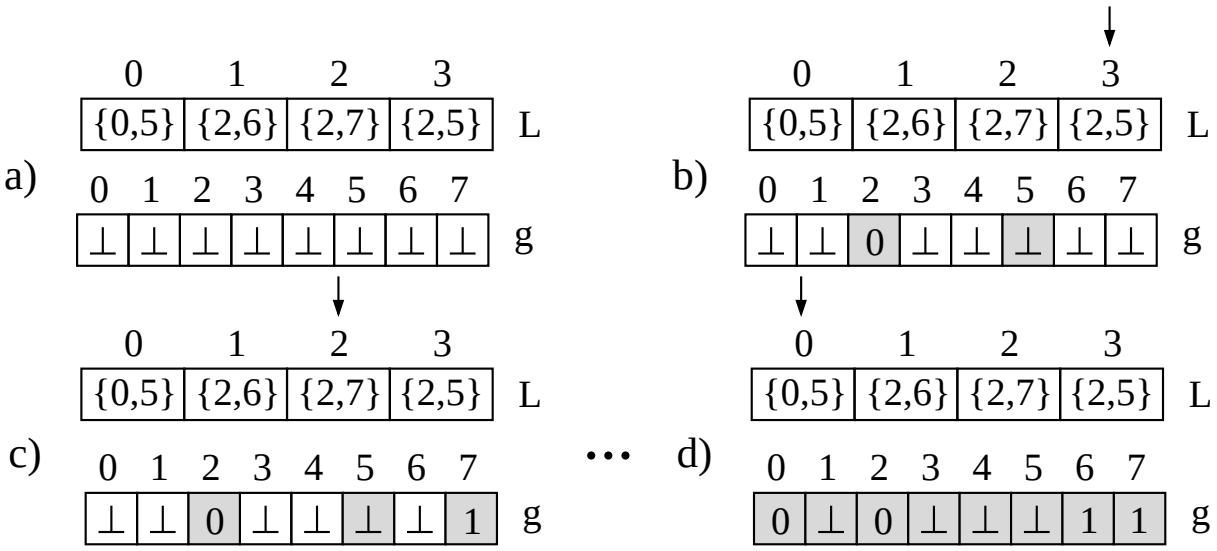
a) Estado inicial

b) Processamento da aresta em $L[3]$. $N_a = \{2, 5\}$ e $A = \emptyset$. Então, $\sum_{v \in A} g(v) = 0$, $j_0 = 0$, $u_{j_0} = u_0 = 2$, $g(2) = (0 - 0) \bmod 2 = 0$ e os vértices 2 e 5 são marcados como visitados.

Hashing Perfeito

Assinalamento

Exemplo:



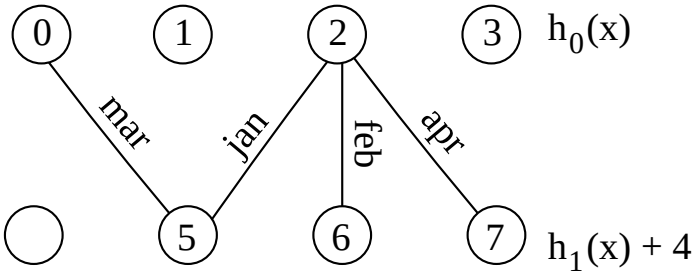
c) Processamento da aresta em $L[2]$. $N_a = \{7\}$ e $A = \{2\}$. $\sum_{v \in A} g(v) = g(2) = 0$, $j_0 = 1$, $u_{j_0} = u_1 = 7$, $g(7) = (1 - 0) \bmod 2 = 1$ e o vértice 7 é marcado como visitado.

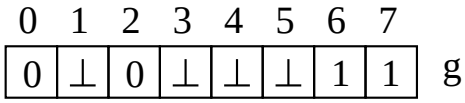
d) Processamento da aresta em $L[0]$. $N_a = \{0\}$ e $A = \{5\}$. $\sum_{v \in A} g(v) = g(5) = \perp = 2$, $j_0 = 0$, $u_{j_0} = u_0 = 0$, $g(0) = (0 - 2) \bmod 2 = 0$ e o vértice 0 é marcado como visitado.

Hashing Perfeito

Avaliação da PHF resultante

Exemplo:





$$\mathsf{phf}(t) = h_{\phi(t)}(t) + \phi(t)\nu,$$

onde,

$$\phi(t) = (g(h_0(t)) + g(h_1(t) + \nu)) \bmod 2.$$

Termo t	ν	$h_0(t)$	$h_1(t)+\nu$	$\phi(t)$	phf(t)
jan	4	2	5	0	2
feb	4	2	6	1	6
mar	4	0	5	0	0
apr	4	2	7	1	7

Hashing Perfeito

Construção da MPHF a partir da PHF:

1. Cada vértice do grafo assume somente um dentre dois estados: (i)**assinalado** e (ii)**não assinalado**
2. Existem exatamente n vértices assinalados.
3. Portanto, a seguinte função é uma MPHF:

$$\text{mphf}(x) = \text{rank}(\text{phf}(x)).$$

4. A função rank : $V \rightarrow [0, n-1]$ é definida por:

$$\text{rank}(x) = |\{y \in V \ / \ y < x \text{ and } g(y) \neq \perp\}|.$$

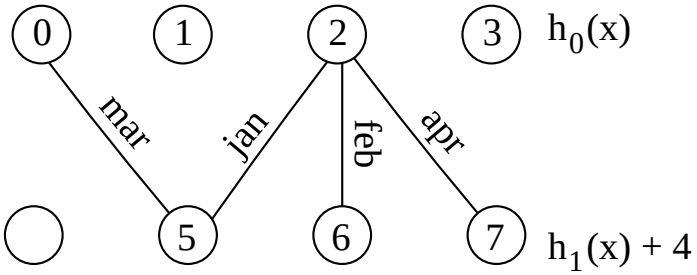
   e computa quantos vértices estão assinalados antes de um dado vértice x.

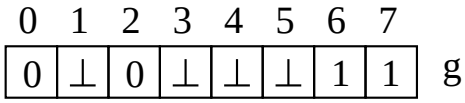
5. O objetivo desta etapa é computar as estruturas de dados que irão permitir a avaliação da função rank em tempo $O(1)$

Hashing Perfeito

Avaliação da MPHF resultante

Exemplo:





$$\mathsf{phf}(t) = h_{\phi(t)}(t) + \phi(t)\nu,$$

onde,

$$\phi(t) = (g(h_0(t)) + g(h_1(t) + \nu)) \bmod 2.$$

e,

$$\mathsf{mphf}(t) = \mathsf{rank}(\mathsf{phf}(t)).$$

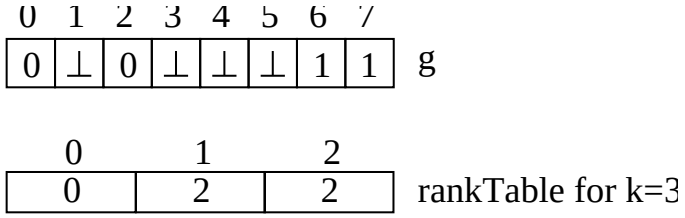
Termo t	ν	$h_0(t)$	$h_1(t)+\nu$	$\phi(t)$	phf(t)	mphf(t)
jan	4	2	5	0	2	1
feb	4	2	6	1	6	2
mar	4	0	5	0	0	0
apr	4	2	7	1	7	3

Hashing Perfeito

Como computar a função rank em tempo $O(1)$?

Algoritmo:

1. Armazene explicitamente em uma rankTable o rank de toda $k-$ésima entrada de g, onde $k = \lfloor \log(|V|)/\epsilon \rfloor$, para qualquer $\epsilon > 0$. Exemplo para $k = 3$:



2. Para computar rank(i), sendo $i \in V$, procura-se o rank do maior índice pré-computado $j \leq i$, e adiciona o número de vértices assinalados da posição j até a $i-1$. Exemplos:

$$\begin{aligned} \text{rank}(6) &= \text{rankTable}[6/3] = 2 \\ \text{rank}(7) &= \text{rankTable}[7/3] + 1 = 3 \end{aligned}$$

3. Para se ter uma complexidade de tempo $O(1/\epsilon)$, utiliza-se uma tabela de *lookup* que permite contar o número de vértices assinalados em $\Omega(\log |V|)$ entradas de g em tempo constante.

Hashing Perfeito

Geração da MPHF:

1. Escolha um valor para m e faça $\nu = m/2$.
2. Escolha os pesos $p_0[i]$ e $p_1[i]$ ou $T_0[i,j]$ e $T_1[i,j]$ para $0 \leq i < \max_t \mid t \mid$ e $0 \leq j < |\Sigma|$, sendo Σ o alfabeto utilizado para compor cada termo t.
3. Gerar o grafo bipartido $G = (V, E)$ onde:
   $V = \{0, \ldots, m-1\}$ e
   $E = \{\{h_0(t), h_1(t) + \nu\} \mid t \in L\}$
4. Testar se G é acíclico. Caso não seja, retornar ao passo 2. Durate o teste é gerada uma lista L que armazena as arestas na ordem de remoção.
5. Gerar a função g percorrendo L do fim para o início.
6. Gerar as estruturas de dados rankTable e a tabela de *lookup* T para se computar a função $rank$ em tempo $O(1)$.
7. Retornar $p_0[i]$, $p_1[i]$ (ou $T_0[i,j]$, $T_1[i,j]$), g, rankTable e T.

Hashing Perfeito

Espaço de armazenamento da MPHFs:

1. A descrição da função é composta por:
   (a) $p_0[i]$, $p_1[i]$ (ou $T_0[i,j]$, $T_1[i,j]$)
   (b) g
   (c) rankTable.
2. As tabelas de valores na letra (a) são fixos a priori e não dependem de n. Assim, requerem $O(1)$ bits para serem armazenadas.
3. Os valores armazenados em g pertecem a $\{0,1,2\}$ e, portanto, podem ser codificados em 2 bits. Logo o espaço para g é $2|g| = 2|V| = 2m$ bits.
4. rankTable possui m/k entradas de $\log m$ bits. Como $k = \log(m)/\epsilon$, para $\epsilon > 0$. Então, o espaço para armazenar rankTable é ϵm bits.
5. Espaço total: $2m + \epsilon m + O(1)$ bits.

Hashing Perfeito

Espaço de armazenamento da tabela de *lookup* T:

1. Teoricamente:
   - T permite contar em temo $O(1)$ quantos vértices estão assinalados em $\log m$ bits.
   - Como cada entrada de g possui 2 bits, então, o maior valor retornado por T é $\frac{\log m}{2}$. para codificar o maior valor são necessários aproximadamente $log \log m - 1$ bits.
   - Como T possui m entradas e cada uma requer $O(\log \log m)$ bits, então, o espaço requerido por T é $O(m \log \log m)$ bits.
2. Implementação:
   - T permite contar em temo $O(1)$ quantos vértices estão assinalados em 16 bits.
   - O maior valor retornado por T é 8 que requer 4 bits para ser codificado. Por simplicidade utilizamos 8 bits.
   - Como T possui 2^{16} entradas e cada uma requer 8 bits, então, o espaço requerido por T é $2^{16} \times 2^3 = 2^{19}$ bits.

Hashing Perfeito

Problema

- **Quando o grafo contém ciclos**: isto impossibilitaria a associação de um único vértice a cada arestas. Grafos com um único ciclo são possíveis de serem tratados, embora o algoritmo fique um pouco mais complicado.
- Grafos acíclicos não possuem este problema.
- Um caminho seguro para se ter sucesso é obter antes um grafo acíclico e depois realizar a atribuição de valores para a função g.
- Esta idéia foi originalmente apresentada em Czech, Havas e Majewski (1992). Porém as MPHFs geradas por eles requerem $O(n \log n)$ bits para serem armazenadas, equanto que as apresentadas aqui requerem $O(n)$ bits poque $m = cn$, veja em seguida.

Teoria de Grafos Randômicos

- Para $m < 2n$
  - Prob. de gerar grafo acíclico $\longrightarrow$ 0, quando n cresce.

- Para $m = cn$, sendo $c > 2$
  - Prob. de um grafo randômico de m vértices e n arestas ser acíclico é aproximadamente:

$$P_a = e^{1/c}\sqrt{\frac{c-2}{c}}$$

- O número esperado de grafos gerados até encontrar o primeiro acíclico é:

$$N_i = \frac{1}{e^{1/c}\sqrt{\frac{c-2}{c}}}$$

Ex.: para $m = 3n$

$N_i \approx 1.24$

- Problema: espaço $m = 3n$ é muito!

Hashing Perfeito

Solução para reduzir m:

- Não utilizar grafos tradicionais, mas sim **hipergrafos**, ou r-grafos, nos quais cada aresta conecta um número qualquer r de vértices.
- Para tanto, basta usar uma terceira função h_2 para gerar um trigrafo com arestas conectando três vértices, chamado de 3-grafo.
- Em outras palavras, cada aresta é uma tripla do tipo $\{h_0(t), h_1(t)+\nu, h_2(t)+2\nu\}$, e a função de transformação é dada por:

$$\mathsf{phf}(t) = h_{\phi(t)}(t) + \phi(t)\nu,$$

onde,

$$\phi(t) = (g(h_0(t)) + g(h_1(t) + \nu) + g(h_2(t) + 2\nu)) \bmod 3.$$

e,

$$\mathsf{mphf}(t) = \mathsf{rank}(\mathsf{phf}(t)).$$

Hashing Perfeito

Solução para reduzir m:

- Os valores de g ainda podem ser codificados em 2 bits, pois pertencem a $\{0, 1, 2, 3\}$. Assim, o Espaço total ainda é $(2 + \epsilon)m + O(1)$ bits
- Nesse caso, o valor de m pode ser próximo a $1.23n$.
- Logo, o uso de trigrafos reduz o custo de espaço da função de transformação perfeita, mas aumenta o tempo de acesso ao vocabulário.
- O algoritmo descrito funciona da mesma forma. Porém, é preciso passar agora o tamanho r das arestas para que a função $\phi(t)$ seja tomada módulo r.
- Neste caso, para $m \geq 1.23n$ o número de tentativas para para gerar o grafo é $O(1)$.
- Conjunto com $3,541,615$ URLs: $<$ 10 s e o espaço ocupado pela função foi $2.62n$ bits.

Hashing Perfeito

Pseudo código para gerar a MPHF

```
procedure Generate (S, r, g, rankTable)
    Mapping (S, G_r, L);
    Assigning (G_r, L, g);
    Ranking (g, rankTable);
```

Hashing Perfeito

Pseudo código para o passo de mapeamento

```
procedure Mapping (S , G_r , L)
    repeat
        E(G_r) = ∅;
        select randomly h_0, h_1, ..., h_{r-1} from H;
        for each x ∈ S do
            e = {h_0(x), ..., h_{r-1}(x) + (r − 1)ν};
            addEdge (G_r , e);
        L = isAcyclic(G_r);
    until L is not empty
```

Hashing Perfeito

Pseudo código para o passo de Assinalamento

```
procedure Assigning (G_r , L , g)
    for u = 0 to |V(G_r)| − 1 do
        visited[u] = false;
        g(u) = ⊥;
    for i = |L| − 1 to 0 do
        e = L[i]; sum = 0;
        for j = r − 1 to 0 do
            if (not visited[e[j]])
                visited[e[j]] = true;
                u = e[j];
                j_0 = j;
            else sum += g(e[j]);
        g(u) = (j_0 − sum) mod r;
```

Hashing Perfeito

Pseudo código para o passo de *Ranking*

```
procedure Ranking (g , rankTable)
    sum = 0;
    for i = 0 to |g| − 1 do
       if(i mod k == 0) rankTable[i/k] = sum;
       if(g(i) ≠ ⊥) sum++;
```

Hashing Perfeito

Pseudo código para avaliar a PHF

```
function phf (x, g, r)
    e = {h_0(x), ..., h_{r-1}(x) + (r - 1)ν};
    sum = 0;
    for i = 0 to r - 1 do sum += g(e[i]);
    return e[sum mod r];
```

Hashing Perfeito

Pseudo código para avaliar a MPHF:

```
u = phf(x , g , r );
j = u/k;
rank = rankTable[ j ];
for i = j * k to u − 1 step E do
    rank += T_r[g(i → i + E)];
return rank;
```

- T_r conta o número de vértices assinalados em $\mathcal{E}$ entradas de g.
- A notação $g(i \rightarrow j)$ representa os valores armazenados nas entradas de $g(i)$ até $g(j)$ para $i \leq j$.
- Se $j \geq |g|$ ou $(j - i + 1) < \mathcal{E}$, o valor $\perp$, o qual é usado para representar vértices não assinalados, é anexado para completar as entradas a serem consultadas em T_r.

Referências

[BPZ07] Botelho, F., Pagh, R., Ziviani, N. Simple and Efficient Minimal Perfect Hash Functions. Submitted to WADS 2007.

[CHM92] Czech, Z.J., Havas, G., Majewski, B.S. An Optimal Algorithm for Generating Minimal Perfect Hashing functions. Information Processing Letters, 43, 1992, 257-264.

[HMWC93] Havas, G., Majewski, B.S., Wormald, N.C., Czech, Z.J. Graphs, Hipergraphs and Hashing. Proceedings 19th International Workshops on Graph - Theoretic Concepts in Computer Science. Springer-Velasq LNCS, 1993.

[MWHC96] Majewski, B., Wormald, N., Havas, G., Czech, Z. A family of perfect hashing methods, The Computer Journal 39 (6) (1996) 547-554.