Tópicos em Recuperação de Informação[1]

Nivio Ziviani

[1]Conjunto de transparências elaborado por Nivio Ziviani e Patrícia Correia

Hashing Perfeito

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

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

com duplicações (ou colisões) permitidas.

Exemplo de função:

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

onde,

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

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

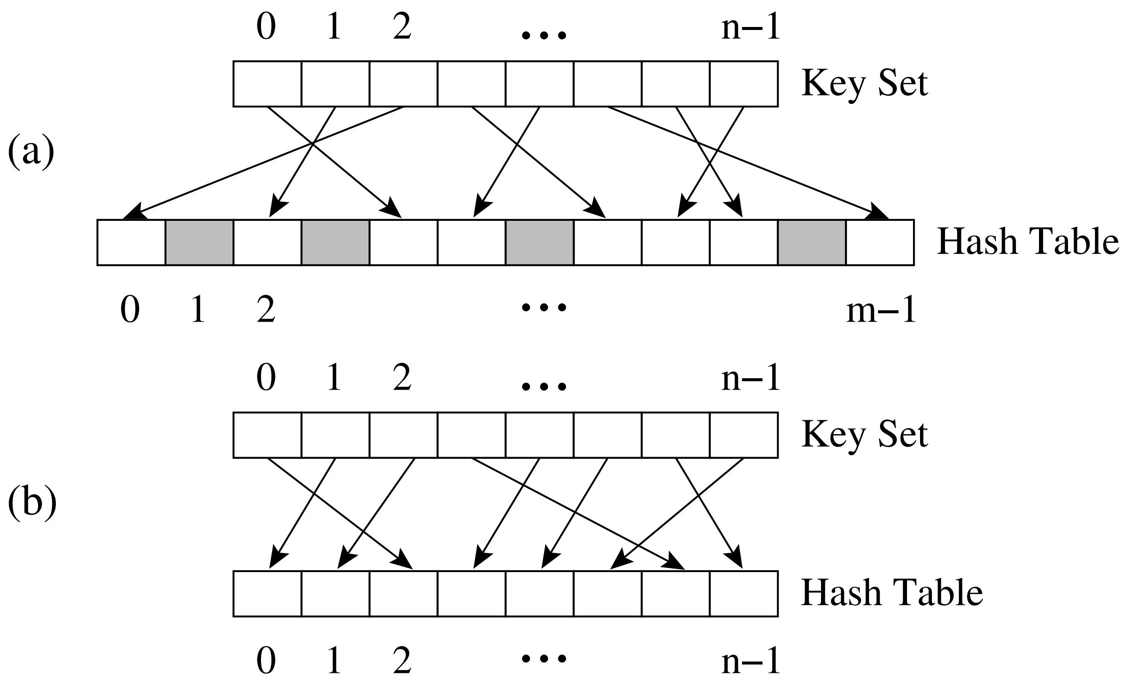


Figura 1: (a) Perfect hash function. (b) Minimal perfect hash function.

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 OPMPHF h ?

- Não há necessidade de armazenar os termos t do vocabulário.

- 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 um método com $m \cong 1.25n$.

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:

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

ou

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

onde:

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

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

$\mid t \mid$: 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

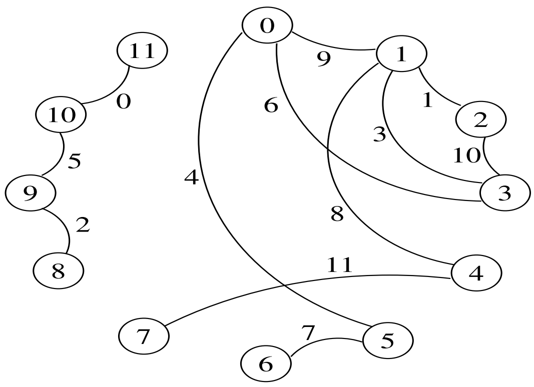
- **Chaves**: 12 meses do ano abreviados para os três primeiros caracteres.
- **Objetivo**: obter uma função de transformação perfeita h de tal forma que o i-ésimo mês é mantido na $(i-1)$-ésima posição da tabela *hash*:

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**
   $Rotule(v, 0)$.

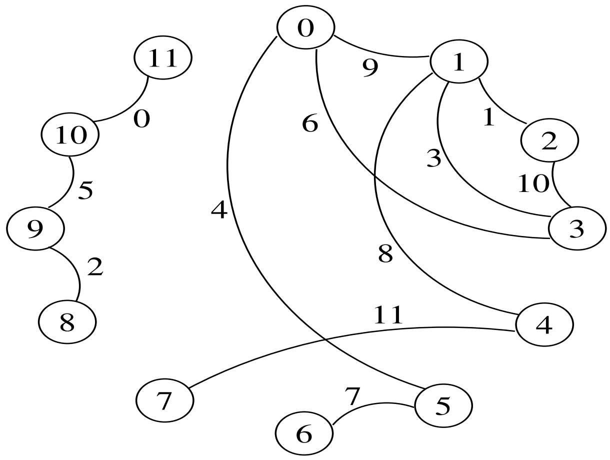
Onde a função $Rotule(v, c)$ é definida por:

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 $m \ll 2n$
  - Prob. de gerar grafo acíclico $\rightarrow$ 0, quando n cresce.

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

$$\sqrt{\frac{m-2m}{m}}$$

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

$$\sqrt{\frac{m}{m-2m}}$$

  Ex.: para $m = 3n$
  $\sqrt{3} \cong 1.7$

- 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).

- TREC com mais de 500.000: $<$ 1 minuto.

Hashing Perfeito x Hashing Linear

Um milhão de chaves do vocabulário do TodoBr

$\alpha = n/m$	Hashing Perfeito	Hashing Linear
1.00	0.89 s	141.90 s
0.90	-	1.78 s
0.80	-	1.10 s
0.70	-	0.88 s
0.60	-	0.76 s
0.50	-	0.70 s

Referências

[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.