Pesquisa em Memória Primária*

Última alteração: 10 de Outubro de 2006

*Transparências elaboradas por Fabiano C. Botelho, Leonardo Rocha, Leonardo Mata e Nivio Ziviani

Hashing Perfeito

- Se $h(x_i) = h(x_j)$ se e somente se $i = j$, então não há colisões, e a função de transformação é chamada de **função de transformação perfeita** ou função *hashing* perfeita(hp).
- Se o número de chaves N e o tamanho da tabela M são iguais ($\alpha = N/M = 1$), então temos uma **função de transformação perfeita mínima**.
- Se $x_i \leq x_j$ e $hp(x_i) \leq hp(x_j)$, então a ordem lexicográfica é preservada. Nesse caso, temos uma **função de transformação perfeita mínima com ordem preservada**.

Vantagens e Desvantagens de Uma Função de Transformação Perfeita

- Não há necessidade de armazenar a chave, pois o registro é localizado sempre a partir do resultado da função de transformação.
- Uma função de transformação perfeita é específica para um conjunto de chaves conhecido.
- A desvantagem no caso é o espaço ocupado para descrever a função de transformação hp.
- Entretanto, é possível obter um método com $M \approx 1,25N$, para valores grandes de N.

Algoritmo de Czech, Havas e Majewski

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

$$hp(x) = (g(h_1(x)) + g(h_2(x))) \bmod N,$$

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

Problema Resolvido Pelo Algoritmo

- Dado um grafo não direcionado $G = (V, A)$, onde $|V| = M$ e $|A| = N$, encontre uma função $g : V \rightarrow [0, N - 1]$, definida como $hp(a = (u, v) \in A) = (g(u) + g(v)) \bmod N$.
- 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.

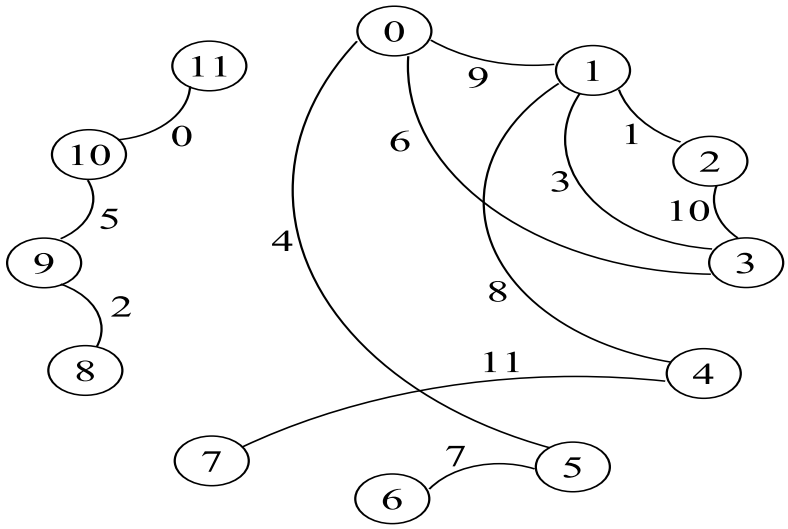
Exemplo

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

Chave x	$h_1(x)$	$h_2(x)$	$hp(x)$
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

Grafo Randômico gerado

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



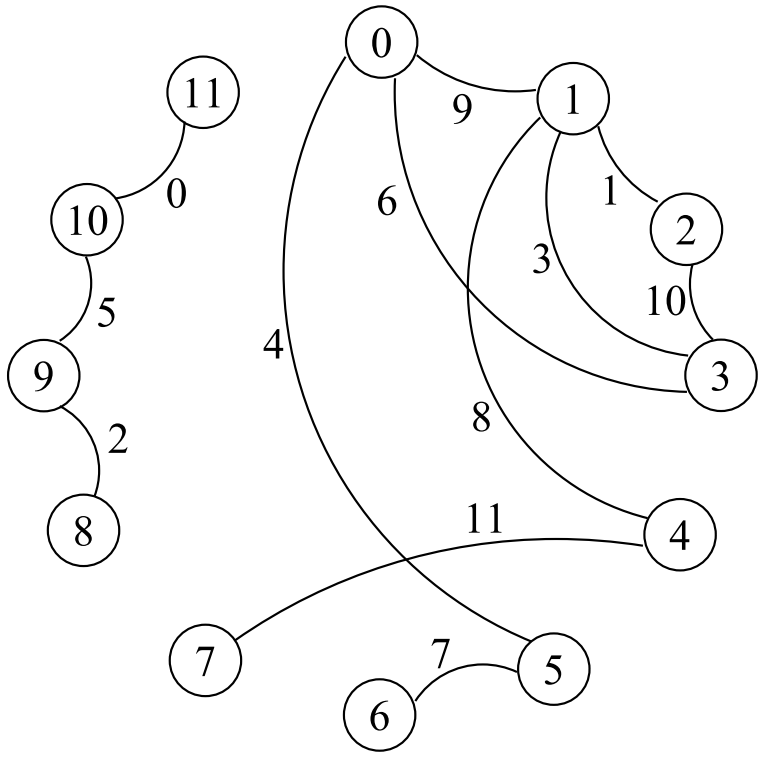
- Os vértices são rotulados com valores no intervalo $0 \ldots M-1$
- As arestas definidas por $(h_1(x), h_2(x))$ para cada uma das N chaves x.
- Cada chave corresponde a uma aresta que é rotulada com o valor desejado para a função hp perfeita.
- Os valores das duas funções $h_1(x)$ e $h_2(x)$ definem os vértices sobre os quais a aresta é incidente.

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 $(h_1(x), h_2(x))$, o valor de $hp(x) = g(h_1(x)) + g(h_2(x))) \bmod N$ seja igual ao rótulo da aresta.

- **Algoritmo:**
  1. Qualquer vértice não processado é escolhido e feito $g[v] = 0$.
  2. As arestas que saem do vértice v são seguidas e o valor $g(u)$ do vértice u destino é rotulado com o valor da diferença entre o valor da aresta (v, u) e $g(v)$, tomado $\bmod N$.
  3. Procura-se o próximo componente conectado ainda não visitado e os mesmos passos descritos acima são repetidos.

Seguindo o Algoritmo para Obter g no Exemplo dos 12 Meses do Ano



	Chave x	$h_1(x)$	$h_2(x)$	$hp(x)$		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

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).

Primeiro Refinamento do Procedimento para Atribuir Valores ao Arranjo g

```
boolean rotuleDe (int v, int c, Grafo G, int g[]) {
  boolean grafoRotulavel = true;
  if (g[v] != Indefinido) if (g[v] != c)
                    grafoRotulavel = false;
  else {
    g[v] = c;
    for (u ∈ G.listaAdjacentes (v))
      rotuleDe (u, (G.aresta (v,u) – g[v]) % N, g);
  }
  return grafoRotulavel;
}
boolean atribuig (Grafo G, int g[]) {
  boolean grafoRotulavel = true;
  for (int v = 0; v < M; v++) g[v] = Indefinido;
  for (int v = 0; v < M; v++)
    if (g[v] == Indefinido)
        grafoRotulavel = rotuleDe (v, 0, G, g);
  return grafoRotulavel;
}
```

Algoritmo para Obter a Função de Transformação Perfeita

```
void obtemHashingPerfeito () {
  Ler conjunto de N chaves;
  Escolha um valor para M;
  do {
    Gera os pesos p1[i] e p2[i] para
                        0 ≤ i ≤ maxTamChave − 1;
    Gera o grafo G = (V, A);
    grafoRotulavel = atribuig (G, g);
  } while (!grafoRotulavel);
  Retorna p1 , p2 e g;
}
```

Estruturas de dados e operações para obter a função *hash* perfeita

```
package cap5.fhpm;
import java.io.*;
import cap7.listaadj.arranjo.Grafo; // vide Programas do capítulo
7
public class FHPM {
  private int p1[], p2[]; // pesos de h1 e h2
  private int g[]; // função g
  private int N;    // número de chaves
  private int M;    // número de vértices
  private int maxTamChave, nGrafosGerados, nGrafosConsiderados;
  private final int Indefinido = -1;
  // Entram aqui os métodos privados das transparências 76, 103 e 104
  public FHPM (int maxTamChave, int n, float c) {
    this.N  = n; this.M  = (int)(c*this.N);
    this.maxTamChave = maxTamChave; this.g = new int[this.M];
  }
  public void obtemHashingPerfeito (String nomeArqEnt)
                                          throws Exception {
    BufferedReader arqEnt = new BufferedReader (
                            new FileReader (nomeArqEnt));
    String conjChaves[] = new String[this.N];
    this.nGrafosGerados = 0;
    this.nGrafosConsiderados = 0;  int i = 0;
// Continua na próxima transparência
```

Estruturas de dados e operações para obter a função *hash* perfeita

```
      while ((i < this.N) &&
             ((conjChaves[i] = arqEnt.readLine()) != null))i++;
      if (i != this.N)
        throw new Exception ("Erro: Arquivo de entrada possui"+
                             "menos que "+
                             this.N + " chaves");
      boolean grafoRotulavel = true;
      do {
        Grafo grafo = this.geraGrafo (conjChaves);
        grafoRotulavel = this.atribuig (grafo);
      }while (!grafoRotulavel);
      arqEnt.close ();
    }
    public int hp (String chave) {
      return (g[h (chave, p1)] + g[h (chave, p2)]) % N;
    }
    // Entram aqui os métodos públicos dos Programas 106 e 107
}
```

Gera um Grafo sem Arestas Repetidas e sem *Self-Loops*

```
private Grafo geraGrafo (String conjChaves[]) {
  Grafo grafo; boolean grafoValido;
  do {
    grafo = new Grafo (this.M, this.N); grafoValido = true;
    this.p1 = this.geraPesos (this.maxTamChave);
    this.p2 = this.geraPesos (this.maxTamChave);
    for (int i = 0; i < this.N; i++) {
      int v1 = this.h (conjChaves[i], this.p1);
      int v2 = this.h (conjChaves[i], this.p2);
      if ((v1 == v2) || grafo.existeAresta(v1, v2)) {
        grafoValido = false; grafo = null; break;
      } else {
        grafo.insereAresta (v1, v2, i);
        grafo.insereAresta (v2, v1, i);
      }
    }
    this.nGrafosGerados ++;
  } while (!grafoValido);
  return grafo;
}
```

Rotula Grafo e Atribui Valores para O Arranjo g

```
private boolean rotuleDe (int v, int c, Grafo grafo) {
  boolean grafoRotulavel = true;
  if (this.g[v] != Indefinido) {
    if (this.g[v] != c) {
      this.nGrafosConsiderados++; grafoRotulavel = false;
    }
  } else {
    this.g[v] = c;
    if (!grafo.listaAdjVazia (v)) {
      Grafo.Aresta adj = grafo.primeiroListaAdj (v);
      while (adj != null) {
        int u = adj.peso () – this.g[v];
        if (u < 0) u = u + this.N;
        grafoRotulavel = rotuleDe(adj.vertice2(),u,grafo);
        if (!grafoRotulavel) break; // sai do loop
        adj = grafo.proxAdj (v);
      }
    }
  }
  return grafoRotulavel;
}
```

// Continua na próxima transparência

Rotula Grafo e Atribui Valores para O Arranjo g

```
private boolean atribuig (Grafo grafo) {
  boolean grafoRotulavel = true;
  for (int v = 0; v < this.M; v++) this.g[v] = Indefinido;
  for (int v = 0; v < this.M; v++) {
    if (this.g[v] == Indefinido)
      grafoRotulavel = this.rotuleDe (v, 0, grafo);
    if (!grafoRotulavel) break;
  }
  return grafoRotulavel;
}
```

Método para salvar no disco a função de transformação perfeita

```
public void salvar (String nomeArqSaida) throws Exception {
  BufferedWriter arqSaida = new BufferedWriter (
                              new FileWriter (nomeArqSaida));
  arqSaida.write (this.N + " (N)\n");
  arqSaida.write (this.M + " (M)\n");
  arqSaida.write (this.maxTamChave + " (maxTamChave)\n");

  for (int i = 0; i < this.maxTamChave; i++)
    arqSaida.write (this.p1[i] + " ");
  arqSaida.write ("(p1)\n");

  for (int i = 0; i < this.maxTamChave; i++)
    arqSaida.write (this.p2[i] + " ");
  arqSaida.write ("(p2)\n");

  for (int i = 0; i < this.M; i++)
    arqSaida.write (this.g[i] + " ");
  arqSaida.write ("(g)\n");

  arqSaida.write ("No. grafos gerados por geraGrafo:" +
                   this.nGrafosGerados + "\n");
  arqSaida.write ("No. grafos considerados por atribuig:" +
                  (this.nGrafosConsiderados + 1) + "\n");
  arqSaida.close ();
}
```

Método para ler do disco a função de transformação perfeita

```
public void ler (String nomeArqFHPM) throws Exception {
  BufferedReader arqFHPM = new BufferedReader (
                           new FileReader (nomeArqFHPM));
  String temp = arqFHPM.readLine(), valor = temp.substring(0,
                                            temp.indexOf (" "));
  this.N = Integer.parseInt (valor);
  temp = arqFHPM.readLine(); valor = temp.substring(0,
                                     temp.indexOf (" "));
  this.M = Integer.parseInt (valor);
  temp = arqFHPM.readLine(); valor = temp.substring(0,
                                     temp.indexOf (" "));
  this.maxTamChave = Integer.parseInt (valor);
  temp = arqFHPM.readLine(); int inicio = 0;
  this.p1 = new int[this.maxTamChave];
  for (int i = 0; i < this.maxTamChave; i++) {
    int fim = temp.indexOf (' ', inicio);
    valor = temp.substring(inicio, fim);
    inicio =  fim + 1; this.p1[i] = Integer.parseInt (valor);
  }
  temp = arqFHPM.readLine(); inicio = 0;
  this.p2 = new int[this.maxTamChave];
  for (int i = 0; i < this.maxTamChave; i++) {
    int fim = temp.indexOf (' ', inicio);
    valor = temp.substring(inicio, fim);
    inicio =  fim + 1; this.p2[i] = Integer.parseInt (valor);
  }
  temp = arqFHPM.readLine(); inicio = 0;
  this.g = new int[this.M];
  for (int i = 0; i < this.M; i++) {
    int fim = temp.indexOf (' ', inicio); valor =
    temp.substring(inicio, fim);
    inicio =  fim + 1; this.g[i] = Integer.parseInt (valor);
  }
  arqFHPM.close();
}
```

Programa para gerar uma função de transformação perfeita

```
package cap5;
import java.io.*;
import cap5.fhpm.FHPM; // vide transparência 101
public class GeraFHPM {
  public static void main (String[] args) {
    BufferedReader in = new BufferedReader (
                        new InputStreamReader (System.in));
    try {
      System.out.print ("Numero de chaves:");
      int n = Integer.parseInt (in.readLine ());
      System.out.print ("Tamanho da maior chave:");
      int maxTamChave = Integer.parseInt (in.readLine ());
      System.out.print("Nome do arquivo com chaves a serem lidas:");
      String nomeArqEnt = in.readLine ();
      System.out.print ("Nome do arquivo para gravar a FHPM:");
      String nomeArqSaida = in.readLine ();
      FHPM fhpm = new FHPM (maxTamChave, n, 3);
      fhpm.obtemHashingPerfeito (nomeArqEnt);
      fhpm.salvar (nomeArqSaida);
    } catch (Exception e) {System.out.println (e.getMessage ());}
  }
}
```

Programa para testar uma função de transformação perfeita

```
package cap5;
import java.io.*;
import cap5.fhpm.FHPM; // vide transparência 101
public class TestaFHPM {
  public static void main (String[] args) {
    BufferedReader in = new BufferedReader (
                        new InputStreamReader (System.in));
    try {
      System.out.print ("Nome do arquivo com a FHPM:");
      String nomeArqEnt = in.readLine ();
      FHPM fhpm = new FHPM (0, 0, 0);
      fhpm.ler (nomeArqEnt);
      System.out.print ("Chave:");String chave = in.readLine ();
      while (!chave.equals ("aaaaaa")) {
        System.out.println ("Indice: " + fhpm.hp (chave));
        System.out.print ("Chave:"); chave = in.readLine ();
      }
    } catch (Exception e) {System.out.println (e.getMessage ());}
  }
}
```

Análise

- **A questão crucial é:** quantas interações são necessárias para obter um grafo $G = (V, A)$ que seja rotulável?
- Para grafos arbitrários, é difícil achar uma solução para esse problema, isso se existir tal solução.
- Entretanto, para **grafos acíclicos**, a função g existe sempre e pode ser obtida facilmente.
- Assim, a resposta a esta questão depende do valor de M que é escolhido no primeiro passo do algoritmo.
- Quanto maior o valor de M, mais esparso é o grafo e, conseqüentemente, mais provável que ele seja acíclico.

Análise

- Segundo Czech, Havas e Majewski (1992), quando $M \leq 2N$ a probabilidade de gerar aleatoriamente um grafo acíclico tende para zero quando N cresce.
- Isto ocorre porque o grafo se torna denso, e o grande número de arestas pode levar à formação de ciclos.
- Por outro lado, quando $M > 2N$, a probabilidade de que um grafo randômico contendo M vértices e N arestas seja acíclico é aproximadamente

$$\sqrt{\frac{M-2N}{M}},$$

- E o número esperado de grafos gerados até que o primeiro acíclico seja obtido é:

$$\sqrt{\frac{M}{M-2N}}.$$

Análise

- Para $M = 3N$ o número esperado de iterações é $\sqrt{3}$, $\Rightarrow$ em média, aproximadamente 1,7 grafos serão testados antes que apareça um grafo acíclico.
- Logo, a complexidade de tempo para gerar a função de transformação é proporcional ao número de chaves a serem inseridas na tabela *hash*, desde que $M > 2N$.
- O grande inconveniente de usar $M = 3N$ é o espaço necessário para armazenar o arranjo g.
- Por outro lado, considerar $M < 2N$ pode implicar na necessidade de gerar muitos gráficos randômicos até que um grafo acíclico seja encontrado.

Outra Alternativa

- 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 $(h_1(x), h_2(x), h_3(x))$, e a função de transformação é dada por:

$$h(x) = (g(h_1(x)) + g(h_2(x)) + g(h_3(x))) \bmod N.$$

Outra Alternativa

- 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 dicionário.
- 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.

Experimentos

# Chaves	# Chamadas $geraGrafo$	# Chamadas $atribuig$	Tempo (s)
10	3586	1	0.130
20	20795	16	0.217
30	55482	24	0.390
40	52077	33	0.432
50	47828	19	0.462
60	27556	10	0.313
70	26265	17	0.351
80	161736	92	1.543
90	117014	106	1.228
100	43123	26	0.559