Hashing

Estrutura de Dados II

Jairo Francisco de Souza

Funções Hashing

- ✓ Divisão
- ✓ Compressão de chaves alfanuméricas
- ✓ Multiplicação
- ✓ Enlaçamento
  - ✓ Deslocado
  - ✓ Limite
- ✓ Função Meio-Quadrado
- ✓ Extração
- ✓ Transformação da Raiz

Hashing - Divisão

- Maneira mais simples
- Usar módulo da divisão
- TSize = sizeof(table)
- ***h*(*k*) = *k* mod *TSize*, caso k seja um número**
- Melhor se Tsize é um número primo
  - Números primos tendem a distribuir os restos das divisões de maneira mais uniforme do que números não primos.
- Se TSize não é um número primo pode-se utilizar função:
  - ***h*(*k*) = (*k* mod *p*) mod *TSize*, onde p é um número primo maior que TSize**
- Divisores não primos podem trabalhar bem com a condição que não tenham fatores não primos maiores do que 20 (Lum *et al.*, 1971).

Hashing – Compressão de chaves alfanuméricas

- Utilizado quando as chaves são alfanuméricas
- Chaves são representadas utilizando sua representação interna

B	R	A	S	I	L
C2	D2	C1	D3	C9	CC
11000010	11010010	11000001	11010011	11001001	11001100

Representação hexadecimal e binária do código ASCII

- Muitas vezes surge a necessidade de comprimir uma chave, dado sua representação numérica excessivamente grande
  - Uma idéia útil é utilizar o operador XOR (ou exclusivo)

Hashing – Compressão de chaves alfanuméricas

- Utilização do XOR

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

- Comprimindo a chave BRASIL para um total de 16 bits:

```
(BR) 1100001011010010
(AS) 1100000111010011
(IL) 1100100111001100
xor  1100101011001101 = 51917(10)
```

- O cálculo final do endereço é feito a partir do valor comprimido, por exemplo, utilizando uma tabela com 521 entradas:

```
(51917 mod 521) + 1 = 338
```

Hashing – Compressão de chaves alfanuméricas

- Problema do método:
  - Chaves com permutações de grupos de letras/dígitos irão produzir colisões.
  - BRASIL, BRILAS, ASBRIL, ILBRAS irão produzir o mesmo endereço
  - Uma forma de contornar esse problema é executar uma operação de rotação de bits de cada grupo
    - Rotação de 1 bit no segundo grupo
    - Rotação de 2 bits no terceiro grupo
    - ...

Hashing - Multiplicação

- Este método utiliza uma propriedade do número conhecido como inverso da *relação áurea* ($ø^{-1}$), cujo valor é:

$$ø^{-1} = \frac{\sqrt{5} - 1}{2} = 0{,}61803399$$

- Se calcularmos K=$\lfloor 10(i\phi^{-1} - \lfloor i\phi^{-1} \rfloor) \rfloor$, 1<= i<= 20:

Há permutação perfeita nas 10 primeiras posições

i	k	i	k
1	**6**	11	**7**
2	**2**	12	**4**
3	**8**	13	**0**
4	**4**	14	**6**
5	**0**	15	**2**
6	**7**	16	**8**
7	**3**	17	**5**
8	**9**	18	**1**
9	**5**	19	**7**
10	**1**	20	**3**

Quase acontece o mesmo nas demais posições!

Hashing - Multiplicação

- Podemos generalizar, como função de endereçamento, o valor $\lfloor m(i\phi^{-1} - \lfloor i\phi^{-1} \rfloor) \rfloor$

- Considerando *m* o tamanho da tabela e *i* o valor da chave.
- A função produzirá endereços no intervalo [0, m-1]
- Problema:
  - Computação lenta para cálculo da chave
  - Donald Knuth apresentou uma solução alternativa e igualmente satisfatória

Hashing - Multiplicação

- Segundo Donald Knuth [1973], ao invés de utilizar $ø^{-1}$, ele propõe que seja usado um valor $\frac{A}{w} \cong ø^{-1}$ onde w equivale ao tamanho da palavra do computador e A um inteiro tal que A e w sejam primos entre si.
  - Por exemplo, considerando A = 19 e um computador de 32 bits, temos: $\frac{A}{w} = \frac{19}{32} = 0{,}59357$
  - O método deve ser aplicado em tabelas com 2^k espaços
  - Nesse exemplo, o cálculo do endereço “e” no qual deve ser instalada a chave de valor C numa tabela de 2^k registros é feito pela execução da seguinte seqüência de comandos:

```
e := C * A/w
e := e mod tamanho_tabela
```

Hashing - Enlaçamento

- Chave é dividida em diversas partes
- Partes são combinadas ou enlaçadas e frequentemente transformadas para produzir endereço alvo
- Tipos
  - Enlaçamento deslocado
  - Enlaçamento limite

Hashing – Enlaçamento deslocado

- Enlaçamento deslocado
  - Chave é dividida em partes
  - Uma parte é colocada embaixo da outra
  - Chaves são processadas a seguir
- Exemplo: código pessoal 123-45-6789
  - Dividir em partes e colocar uma embaixo da outra
    - 123
    - 456
    - 789
  - Processamento:
    - adicionar partes
      - 123+456+789 = 1368
    - dividir em módulo TSize
      - Supondo TSize = 1000
      - 1368 mod 1000 = 368

Hashing – Enlaçamento limite

- Enlaçamento limite
  - Chave é dividida em partes
  - Cada parte predeterminada será colocada em ordem inversa
- Exemplo: código pessoal 123-45-6789
  - Dividir em partes e colocar uma embaixo da outra
    - 123
    - 456
    - 789
  - Processamento, 456 é considerado na ordem inversa
    - adicionar partes
      - 123+654+789 = 1566
    - dividir em módulo TSize
      - Supondo TSize = 1000
      - 1566 mod 1000 = 566

Hashing – Função Meio-Quadrado

- A chave é elevada ao quadrado e parte do resultado é usada como endereço.
- Exemplo: chave 3121
  - $3121^2 = 9740641$
  - Considerando TSize igual a 1000 e parte do meio para endereçamento
  - h(3121) = 406
    - 97406*41*
- Uma máscara binária pode ser utilizada para obter posição
  - Se tamanho da tabela é 1024 (em binário 10000000000)
  - 3121^2 em binário é igual a 1001010*0101000010*11000001
  - 0101000010 , que é igual a 322, pode ser extraído usando máscara e uma operação de deslocamento

Extração

- Parte da chave é usada para calcular o endereço
- Exemplo:
  - Código 123-45-6789
  - Pode-se utilizar
    - primeiros quatro dígitos: 1234
    - Últimos quatro dígitos: 6789
    - Dois primeiros dígitos combinados com os dois últimos: 1289
    - ...
- Parte escolhida deve produzir boa distribuição das chaves
- Parte descartada deve distinguir pouco as chaves
- Exemplo:
  - Universidade onde os primeiros 3 dígitos iguais a 999 indica estudante estrangeiro
    - Estes dígitos podem ser omitidos

Transformação da Raiz

- Mudar a base do número, por exemplo, de base 10 para base nonal e dividir em módulo TSize
- Exemplo:
  - 345 em decimal é igual a 423 em base nonal
  - Se TSize igual a 100
    - h(345) = 23
- Não evita colisões

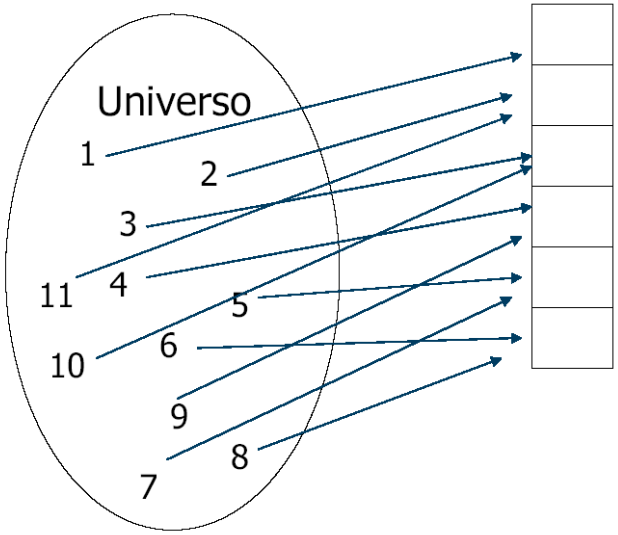
Hashing Universal

- Qualquer função de *hash* está sujeita ao problema de criar chaves iguais para elementos distintos.
- Um estratégia para minimizar as colisões é escolher aleatoriamente (em tempo de execução) uma função *hash* a partir do conjunto de funções cuidadosamente desenhado.
- Para *strings*, por exemplo, podemos assumir $x = (x_0, \ldots, x_k)$
- Deitzfelbinger *et al* (1992) trata a string *x* como os coeficientes de um polinômio módulo um primo.
- Em $x_i \in [u]$, seja $p \geq \max\{u, m\}$ um primo, definimos:

$$(x_0 \ldots x_k) = \left((\sum_{i=0}^{k} x_i \cdot a^i) \bmod p \right) \bmod m \quad \text{onde } a \in [p] \text{ é uniformemente randômico.}$$

Resolução de colisões

- Colisões podem ocorrer
  - Mais de uma chave atribuída para a mesma posição



  - Exemplo: hashing que considera a primeira letra de cada nome
    - Nomes que comecem com a mesma letra teriam conflitos
- Função hashing e tamanho da tabela pode ajudar a diminuir o número de colisões