Governo Federal

Ministério da Educação

departamento de Informática

Universidade Federal do Maranhão

A Universidade que Cresce com Inovação e Inclusão Social

Hashing

Estrutura de Dados II

Prof. João Dallyson

Email: Joao.dallyson@ufma.br

Colisões

- **A melhor solução é evitar as colisões!**
  - Contudo, $|U| > m$ existe uma chance muito grande de haver colisões!
  - Assim, temos que desenvolver/utilizar técnicas para tratar as colisões.

Colisões

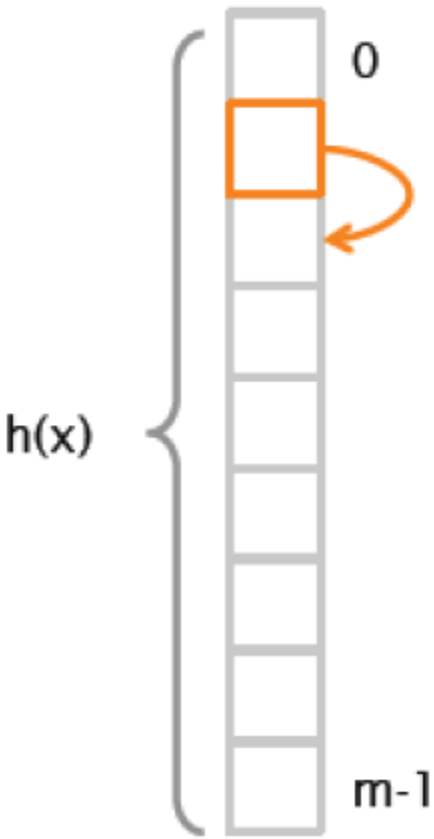
- Tratamento de colisões é procedimento crítico e deve ser realizado com cuidado.
  - Fator de carga: $\alpha = n/m$ *(n elementos / m posições)*
- Quanto menor o fator de carga, menos colisões.
- Fator de carga pequeno não garante ausência de colisões.
- Paradoxo do aniversário (Feller): em grupo com 23 ou mais pessoas juntas ao acaso, existe chance maior que 50% de 2 pessoas fazerem aniversário no mesmo dia.
- Em uma tabela hashing, isso significaria: $\alpha = 23/365 = 0,063$.
- Mesmo assim, há 50% de chance de ocorrer pelo menos uma colisão.

Tratamento de Colisões

- **Algumas Estratégias:**
  - Tratamento de Colisões por Endereçamento Aberto:
    - Tentativa Linear;
    - Tentativa Quadrática;
  - Tratamento de Colisões por Encadeamento:
    - Externo;
    - Interno;

Endereçamento Aberto – Hashing Linear

- Implementação



  - A função é calculada até que uma posição livre seja encontrada

```
void inserir(Chave ch, Table HashTable){
   inteiro i;
   i = h(ch); //calculo de endereço
   enquanto (HashTable[i] ocupado)
      i = (i + 1) mod n;
   HashTable[i] = ch;
}
```

  - A busca é feita de modo análogo

Endereçamento Aberto – Tentativa Quadrática

- **Tentativa Quadrática:**
  - Suponha endereço da chave x é h'(x), ou seja,
  - h'(x) = h(x, 0)
  - h(x, k) = (h(x, k − 1) + k) (mod m), 0 < k < m.
- **Exemplo:**

$$
\begin{array}{lllll}
h(x,0) & = 1 & & = & 1 \\
h(x,1) & = (h(x,0)+1) & (\text{mod } m) & = & 2 \\
h(x,2) & = (h(x,1)+2) & (\text{mod } m) & = & 4 \\
h(x,3) & = (h(x,2)+3) & (\text{mod } m) & = & 7 \\
h(x,4) & = (h(x,3)+4) & (\text{mod } m) & = & 11 \\
h(x,5) & = (h(x,4)+5) & (\text{mod } m) & = & 16 \\
\vdots & = & \vdots & = & \vdots
\end{array}
$$

Endereçamento Aberto – Tentativa Quadrática

- **Exemplo**

Índice	Situação	Chave
0	L	
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	L	

Inserir chave 12

12%8=4

Índice	Situação	Chave
0	L	
1	L	
2	L	
3	L	
4	O	12
5	L	
6	L	
7	L	

Inserir chave 20

20%8=4

(4+1) mod 8=5

Índice	Situação	Chave
0	L	
1	L	
2	L	
3	L	
4	O	12
5	O	20
6	L	
7	L	

Inserir chave 28

28%8=4

(4+1) mod 8=5

(5+2) mod 8=7

Índice	Situação	Chave
0	L	
1	L	
2	L	
3	L	
4	O	12
5	O	20
6	L	
7	O	28

Inserir chave 36

36%8=4

(4+1) mod 8=5

(5+2) mod 8=7

(7+3) mod 8=2

Índice	Situação	Chave
0	L	
1	L	
2	O	36
3	L	
4	O	12
5	O	20
6	L	
7	O	28

Remover chave 12

12%8=4

Índice	Situação	Chave
0	L	
1	L	
2	O	36
3	L	
4	R	12
5	O	20
6	L	
7	O	28

Endereçamento Aberto – Hashing Duplo

- Também chamado de *re-hash*
- Em vez de incrementar a posição em 1, uma outra função de *hash* auxiliar é utilizada
    - $h(i,k) = (h_1(k) + i * h_2(k)) \bmod m$
- Restrição:
    - $h_2(k)$ primo em relação a m. Como?
        - m deve ser uma potência de 2
        - h_2 deve ser sempre ímpar
- Exemplo
    - h1(k) = k + 37; // usando número primo
    - h2(k) = k + 31; // se h1 é impar
    - h2(k) = (k + 31) + 1; // se h1 é par

Endereçamento Aberto – Hashing Duplo

- **Exemplo:**

h1 = 27 + 37 = 64

h2 = (27 + 31) + 1 = 59

h = (64 + 0 * 59) mod 8 = 0

27?

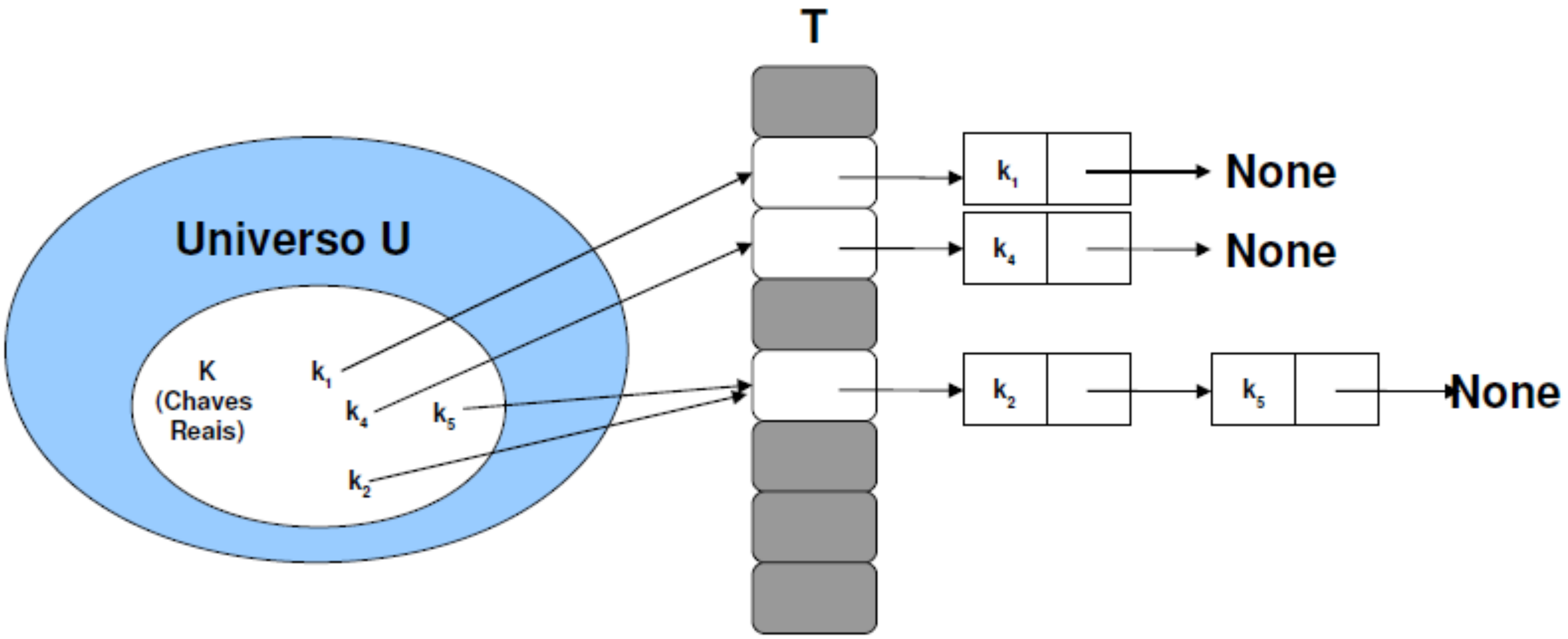
0	1	2	3	4	5	6	7
64			11	20			7

h = (64 + 1 * 59) mod 8 = 123 mod 8 = 3

h = (64 + 2 * 59) mod 8 = 182 mod 8 = 6

Resolução de Colisão por Encadeamento

- **Uma possível solução é colocar todos os elementos que efetuam o hash para a mesma posição como uma lista ligada**



Resolvendo colisão por encadeamento

- **As operações de dicionário quando as colisões são resolvidas por encadeamento:**
  - CHAINED-HASH-INSERT(T,k)
    - Insere x no início da lista T[h(chave[x])]
  - CHAINED-HASH-SEARCH(T,k)
    - Procura por um elemento com a chave k na lista T[h(k)]
  - CHAINED-HASH-DELETE(T,k)
    - Elimina o elemento x da lista T[h(chave[x])]

Análise do Hash com Encadeamento

- **Qual a qualidade de execução do hash com encadeamento?**
  - Em particular, quanto tempo ele leva para localizar um elemento com uma determinada chave?
  - Para uma tabela hash com **m** posições e n elementos:
    - Fator de Carga α: n/m
      - Todas as análises serão feitas em termos de α!

Análise do Hash com encadeamento

- **Pior-caso:**
  - Todas as n chaves executam o hash na mesma posição!
    - É criada uma lista de comprimento n.
  - O tempo para a pesquisa é n mais o tempo de calcular a função hash.
    - Esta caso não é melhor que utilizar uma lista ligada para
      - representar todos os elementos

Análise do hash em encadeamento

- **Caso Médio:**
  - Irá depender como a função hash irá distribuir o conjunto de chaves entre as m posições.
  - Vamos supor que um elemento dado tem igual probabilidade de efetuar o hash para qualquer das m posições.
    - Hipótese de hash uniforme.
  - Denota-se
    - o comprimento da lista T[j] por nj, onde,

$$n = n_0 + n_1 + \cdots + n_{n-1}$$

- e

$$\overline{n_j} = E[n_j] = \frac{n}{m} \rightarrow Fator\ \ de\ Carga$$

Análise do hash com encadeamento

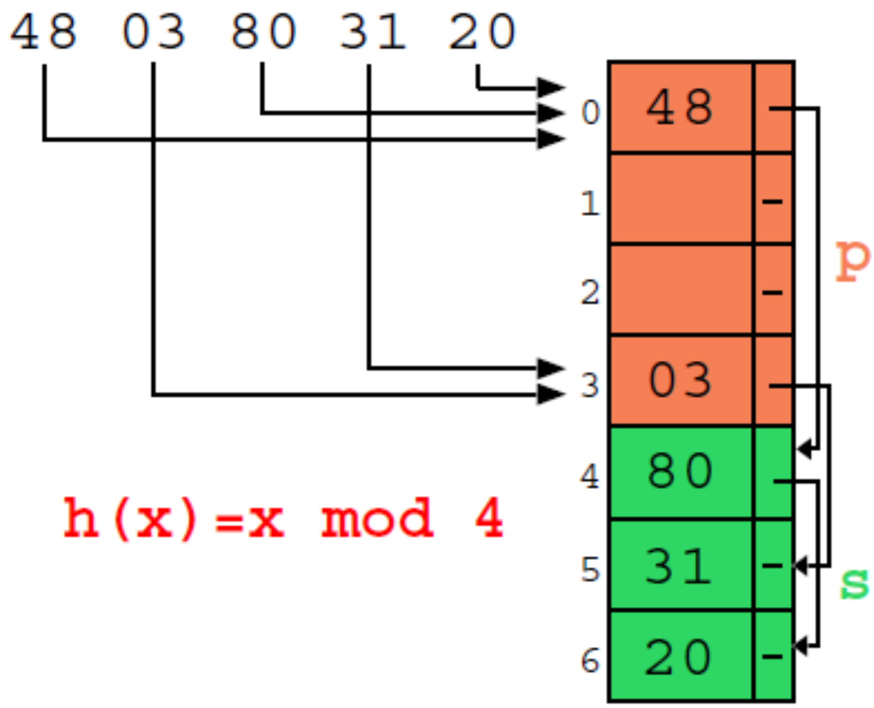
- **Supondo que o valor hash h(k) pode ser calculado em tempo O(1)**
  - Desta forma, o tempo necessário para encontrar um elemento k depende linearmente do comprimento das listas
  - Assim, é possível considerar duas situações:
    - Uma pesquisa não é bem-sucedida;
    - Uma pesquisa é bem-sucedida

Análise do Hash com encadeamento

- **Conclusão?**
  - Se o número de posições na tabela hash é no mínimo proporcional ao número de elementos na tabela:
    - $n = O(m)$, e consequentemente, $\alpha = n/m = O(m)/m = O(1)$
    - Portanto, a pesquisa demora um tempo constante em média
    - Todas as operações de dicionário podem ser admitidas em tempo $O(1)$ em média!

Tratamento colisão: Encadeamento Interior

- **Duas zonas na tabela com *m = p + s:***
  - 1. uma de endereços-base de tamanho *p;*
  - 2. uma de sinônimos de tamanho *s;*



Função Hash

- **O que faz uma função hash de boa qualidade?**
  - Satisfaz, aproximadamente, à hipótese do hash uniforme:
    - Cada chave tem igual probabilidade de efetuar o hash para qualquer das m posições, não importando o hash de qualquer outra chave.
  - Contudo, de forma geral, a distribuição de probabilidade das chaves não é conhecida
    - Impossibilitando a obtenção de um hash uniforme!
  - Ocasionalmente conhecemos a distribuição
    - Por exemplo: Chaves sao numeros reais aleatorios k, tal que $0 \leq k \leq 1$, entao: $h(k) = \lfloor km \rfloor$

Função Hash

- **Na prática utiliza-se:**
  - Heurísticas para definir as funções hash

- **De forma geral, as funções hash supoẽm que o universo de chaves é o conjunto de números naturais N.**
  - E a partir de uma chave qualquer realizam alguma operação para representá-la por um número inteiro!

Método de Divisão

- **Mapeia-se uma chave k para uma posição m tomando o resto da divisão inteira:**

$$\boldsymbol{h(k) = k \bmod m}$$

  - Por exemplo:
    - Se m=12 e k=100, então h(k)=4
- **Valores de m**
  - De forma geral evita-se potencias de 2
  - Uma boa escolha é um primo não muito próximo de uma potência de 2

Método de Divisão

- **Exemplo:**
  - Suponha uma tabela hash, com colisões resolvidas com encadeamento, para conter aproximadamente 2000 cadeias de caracteres
  - Suponha também: tolera-se uma pesquisa malsucedida com uma média de 3 elementos
  - Assim, m=701
    - Primo próximo de 2000/3 e distante de qualquer potencia de 2.

Método da Multiplicação

Este método opera em duas etapas:

- Primeira:
  - Multiplica-se a chave **k** por uma constante **A** (**0** < **A** < **1**) e extrai-se a parte fracionária de **kA**.
- Segunda:
  - Multiplica-se o valor encontrado por **m** e toma-se o piso.

Função Hash:

$$h(k) = \lfloor m(kA \bmod 1) \rfloor$$

Método da Multiplicação

- **Uma vantagem deste método é que o valor de m não é crítico!**
  - Pode-se escolher uma potência de 2
- **Quanto ao valor de A:**
  - Irá funcionar com qualquer valor de A. Contudo, funciona melhor com determinados valores que outros.
    - Por exemplo: um valor sugerido é

$$A \approx \left(\frac{\sqrt{5} - 1}{2}\right) = 0{,}6180339887$$

Método da multiplicação

$h(k) = \lfloor m(kA \bmod 1) \rfloor$

- **Exemplo: dado um m = 100, a tabela a seguir mostra as posições em que as chaves serão inseridas**

Chave	h(x)
1000	3
2000	6
3500	11
150	70
10	18
18	12

Chave	h(x)
25	45
32	77
136	5
357	63
211	40
896	75

- **Ex Chave 25: 25 * *0,6180339887 = 15,4508497175 . Fazendo 100 * 0,45 = 45***

Chaves não numéricas

- As chaves não numéricas devem ser transformadas em números:

$$\mathtt{H} = \sum_{i=1}^{n} \mathrm{Chave}[i] \times p[i],$$

  - n é o número de caracteres da chave.
  - Chave[i] corresponde à representação ASCII do i-ésimo caractere da chave.
  - p[i] é um inteiro de um conjunto de pesos gerados randomicamente para $1 \leq i \leq n$.

$$H = \sum_{i=1}^{n} Chave[i] \times 128^{n-i}$$

Chaves não numéricas

```
Indice h(TipoChave Chave, TipoPesos p) {
unsigned int Soma = 0;
int i;

   int comp = strlen(Chave);

   for (i = 0; i < comp; i++)
     Soma += (unsigned int) Chave[i] * p[i];

    return (Soma % M);
}
```

Conclusão

- Complexidade média por operação: *O(1);*
- Tabela *hash é estrutura de dados que não permite* armazenar elementos repetidos;
- Não permite recuperar elementos sequencialmente (ordenação);
- Não permite recuperar o elemento antecessor e sucessor;
- Para otimizar a função *hash é necessário conhecer a* natureza da chave a ser utilizada;
- No pior caso, a ordem das operações pode ser *O(n)* quando todos elementos inseridos colidem.

Aplicações

- **busca de elementos em base de dados:**
  - estruturas de dados em memória;
  - bancos de dados;
  - mecanismos de busca na Internet;
- **verificação de integridade de dados grandes:**
  - 1. envio de dados com resultado da função hash;
  - 2. receptor calcula função hash sobre dados recebidos e obtém novo resultado;
  - 3. se resultados iguais, dados são iguais;
  - 4. se diferentes, novo download é feito;
    - Exemplo: download imagem de disco do Linux;
- **armazenamento de senhas com segurança:** somente resultado função hash é armazenado no servidor;

Exercício

(Poscomp 2002)

28 - Considere uma tabela de espalhamento (tabela de *hash*) com quatro posições numeradas $0, 1, 2$ e 3. Se a seqüência de quadrados perfeitos $1, 4, 9, \ldots, i^2, \ldots$ for armazenada nessa tabela segundo a função $f(x) = x \bmod 4$, como se dará a distribuição dos elementos pelas posições da tabela, à medida que o número de entradas cresce?

(a) Cada posição da tabela receberá aproximadamente o mesmo número de elementos

(b) Três posições da tabela receberão, cada uma, aproximadamente um terço dos elementos

(c) Uma única posição da tabela receberá todos os elementos, e as demais posições permanecerão vazias

(d) Todas as posições da tabela receberão elementos, mas as duas primeiras receberão, cada uma, o dobro das outras

(e) As duas primeiras posições da tabela receberão, cada uma, aproximadamente a metade dos elementos, e as demais posições permanecerão vazias

Exercício

2) Considere uma tabela com 11 posições, e seja a função de hash primária h'(k)=k mod m. Demostre a inserção das chaves 5, 28, 19, 15, 20, 33, 12, 17 e 10.

3) Insira os elementos [16, 23, 9, 34, 12, 56] na tabela hash de 11 posições utilizando função de hash primária h1(k)=k mod m e resolvendo colisões utilizando uma segunda função de hash h2(k) = 5 – (k mod 5).

Resposta Q 3

16, 23 9, 34, 12, 56

0		
1	23	← 34_0, 12_0, 56_0
2	34_1	←
3		56_3
4	12_1	
5	16	← 56_1
6	56_4	
7		
8		
9	9	← 56_2
10		

$16 \bmod 11 = 5$

$23 \bmod 11 = 1$

$9 \bmod 11 = 9$

$34 \bmod 11 = 1$

$H2(34) = 5 - (34 \bmod 5)$

$= 5 - 4 = 1$

$1 + (1 * 1) = 2$

$12 \bmod 11 = 1$

$H2(12) = 5 - (12 \bmod 5)$

$= 5 - 2 = 3$

$1 + (1 * 3) = 4$

$56 \bmod 11 = 1$

$H2(56) = 5 - (56 \bmod 5)$

$= 5 - 1 = 4$

$1 + (1 * 4) = 5$

$1 + (2 * 4) = 9$

$1 + (3 * 4) = 13 \bmod 11$

$= 2$

$1 + (4 * 4) = 17 \bmod 11$

$= 6$

Referências

Básica

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: Teoria e Prática. Editora Campus, 2002
- Ziviani, N. Projeto de Algoritmos Com Implementações em Pascal e C, Cengage Learning, 2004.

Complementar

- TENENBAUM, Aaron; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J. *Estruturas de dados usando C. São Paulo: Makron Books, 1995. ISBN: 9788534603485*
- ASCENCIO, Ana Fernanda Gomes; ARAUJO, Graziela Santos. Estruturas de Dados: Algoritmos, análise da complexidade e implementações em Java e C/C++. Pearson Prentice Hall, 2010
- DROZDEK, Adam. Adam Drozdek. Data Structures and Algorithms in Java. 2. Cengage Learning. 2004. 2. Cengage Learning. 2004
- GOODRICH, Michael T. Estruturas de dados e algoritmos em java. 4 ED. Porto Alegre: Bookman, 2007. 600.
- SKIENA, Steven S.. **The Algorithm Design Manual**. 2. Springer-Verlag. 2008
- Notas de aula: prof. Ítalo Cunha – UFMG. 2012.
- Notas de aula: prof. Rafael Fernandes – DAÍ/IFMA. 2014

Perguntas....