Hashing

Letícia Rodrigues Bueno

UFABC

Hashing (Tabelas de Dispersão): Introdução

- *hash*:

Hashing (Tabelas de Dispersão): Introdução

- *hash*:

Hashing (Tabelas de Dispersão): Introdução

- *hash*:
  1. fazer picadinho (de carne e vegetais);

Hashing (Tabelas de Dispersão): Introdução

- *hash*:
  1. fazer picadinho (de carne e vegetais);
  2. fazer bagunça;

Hashing (Tabelas de Dispersão): Introdução

- *hash*:
  1. fazer picadinho (de carne e vegetais);
  2. fazer bagunça;
- Através de uma função $h(x)$, chave x é transformada em um endereço de uma tabela;

Hashing (Tabelas de Dispersão): Introdução

- *hash*:
  1. fazer picadinho (de carne e vegetais);
  2. fazer bagunça;
- Através de uma função $h(x)$, chave x é transformada em um endereço de uma tabela;
- **Objetivo**: atingir diretamente o local onde chave está;

Hashing (Tabelas de Dispersão): Introdução

- *hash*:
  1. fazer picadinho (de carne e vegetais);
  2. fazer bagunça;
- Através de uma função $h(x)$, chave x é transformada em um endereço de uma tabela;
- **Objetivo**: atingir diretamente o local onde chave está;
- Complexidade média por operação: $O(1)$;

Hashing (Tabelas de Dispersão): Introdução

- *hash*:
  1. fazer picadinho (de carne e vegetais);
  2. fazer bagunça;
- Através de uma função $h(x)$, chave x é transformada em um endereço de uma tabela;
- **Objetivo**: atingir diretamente o local onde chave está;
- Complexidade média por operação: $O(1)$;
- Complexidade pior caso: $O(n)$;

Hashing (Tabelas de Dispersão): Introdução

- *hash*:
  1. fazer picadinho (de carne e vegetais);
  2. fazer bagunça;
- Através de uma função $h(x)$, chave x é transformada em um endereço de uma tabela;
- **Objetivo**: atingir diretamente o local onde chave está;
- Complexidade média por operação: $O(1)$;
- Complexidade pior caso: $O(n)$;
- **Analogia:** distribuição de correspondência em escaninhos rotulados;

Hashing: Definição

Hashing: Definição

- n chaves armazenadas em tabela T sequencial de dimensão m no intervalo $[0, m-1]$;

Hashing: Definição

- n chaves armazenadas em tabela T sequencial de dimensão m no intervalo $[0, m-1]$;
- Duas etapas:

Hashing: Definição

- n chaves armazenadas em tabela T sequencial de dimensão m no intervalo $[0, m-1]$;
- Duas etapas:
  1. Calcular valor da **função de transformação** ou **função hashing**;

Hashing: Definição

- n chaves armazenadas em tabela T sequencial de dimensão m no intervalo $[0, m-1]$;
- Duas etapas:
  1. Calcular valor da **função de transformação** ou **função hashing**;
  2. Lidar com **colisões**;

Hashing: transformação da chave

Transformações sobre chaves são aritméticas, portanto:

Hashing: transformação da chave

Transformações sobre chaves são aritméticas, portanto:

Primeiro passo: transformar chaves não-numéricas em números:

Hashing: transformação da chave

Transformações sobre chaves são aritméticas, portanto:

Primeiro passo: transformar chaves não-numéricas em números:

$$\sum_{i=0}^{n-1} chave[i] \times p[i]$$

Hashing: transformação da chave

Transformações sobre chaves são aritméticas, portanto:

Primeiro passo: transformar chaves não-numéricas em números:

$$\sum_{i=0}^{n-1} chave[i] \times p[i]$$

onde $chave[i]$ está em ASCII ou Unicode e $p[i]$ é um peso.

Hashing: transformação da chave

Em C/C++:

Hashing: transformação da chave

Em C/C++:

```
1  void geraPesos(){
2       for (int i = 0; i < n; i++)
3           pesos[i] = rand() % m + 1;
4  }
```

Hashing: transformação da chave

Em C/C++:

```
1  void geraPesos(){
2       for (int i = 0; i < n; i++)
3           pesos[i] = rand() % m + 1;
4  }
```

Objetivo: chaves diferentes com mesmos caracteres terão pesos diferentes e levarão a funções hashing diferentes.

Hashing: transformação da chave

Em C/C++:

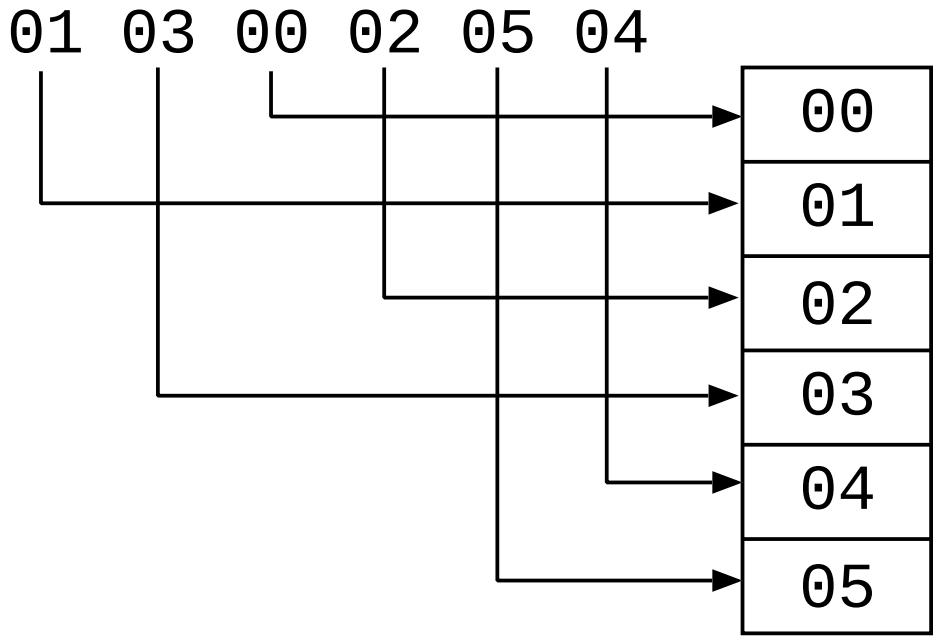
Hashing: transformação da chave

Em C/C++:

```
1  int h(char chave[ ]){
2        int soma = 0;
3        for (int i = 0; i < n; i++)
4              soma += (unsigned int)chave[i] * pesos[i];
5        return soma % m;
6  }
```

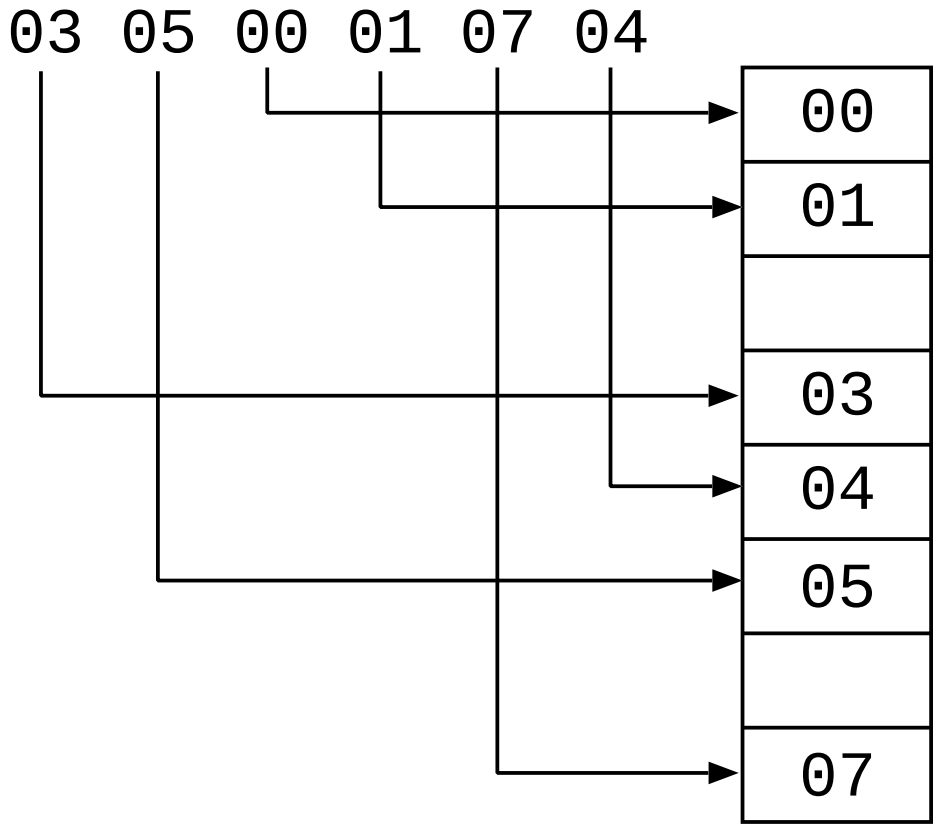
Hashing: Acesso Direto

Chave *x* é armazenada no compartimento *x*:



Hashing: Acesso Direto

Chave x é armazenada no compartimento x:



Pode ser utilizado quando $n = m$ ou $n < m$ mas $m - n$ é pequeno.

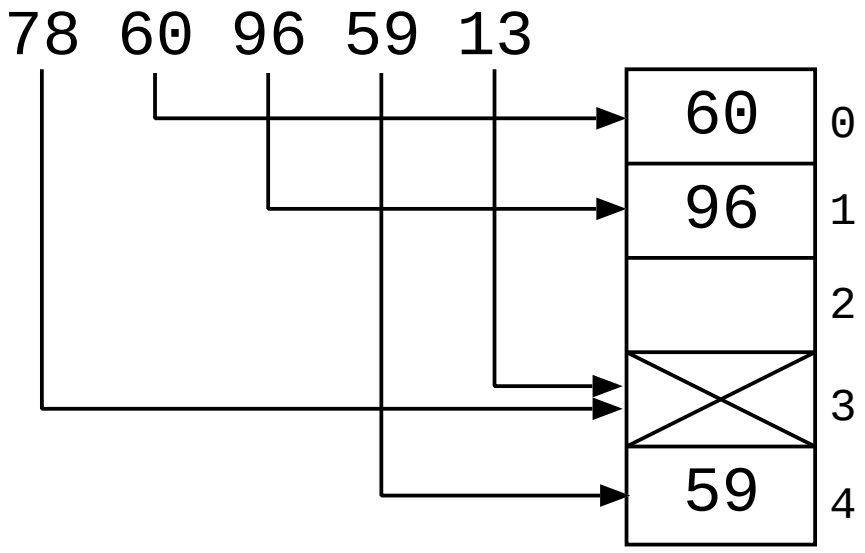
Hashing: Colisões

Hashing: Colisões

Para chaves $x \neq y$, obtemos $h(x) = h(y)$:

Hashing: Colisões

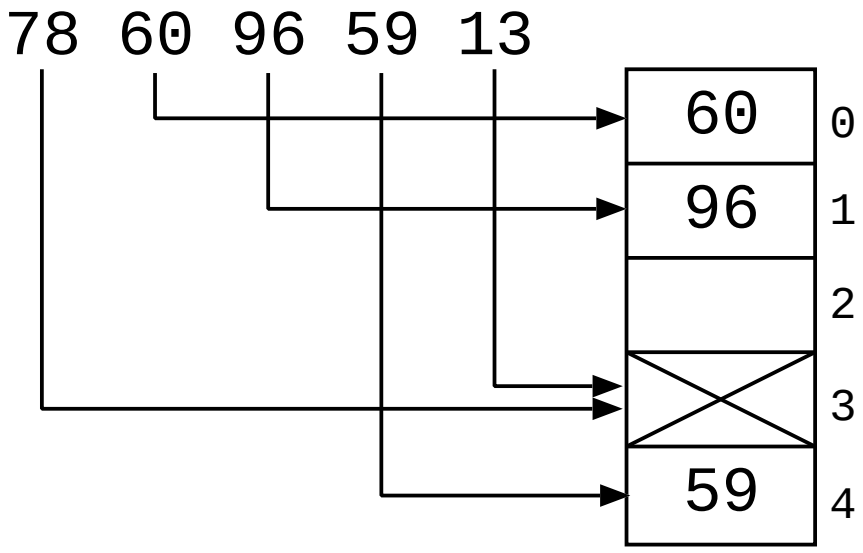
Para chaves $x \neq y$, obtemos $h(x) = h(y)$:



h(x)=x mod 5

Hashing: Colisões

Para chaves $x \neq y$, obtemos $h(x) = h(y)$:



h(x)=x mod 5

Chaves x e y são **sinônimas**.

Hashing: Função de Dispersão

Objetivos:

Hashing: Função de Dispersão

Objetivos:

- produzir número baixo de colisões;

Hashing: Função de Dispersão

Objetivos:

- produzir número baixo de colisões;
- ser facilmente computável: assume-se tempo $O(1)$;

Hashing: Função de Dispersão

Objetivos:

- produzir número baixo de colisões;
- ser facilmente computável: assume-se tempo $O(1)$;
- ser uniforme;

Hashing: Função de Dispersão

Objetivos:

- produzir número baixo de colisões;
- ser facilmente computável: assume-se tempo $O(1)$;
- ser uniforme;

Função uniforme: mesma probabilidade para todos compartimentos, ou seja, $\frac{1}{m}$ de $h(x)$ acontecer.

Função de Dispersão: Método da divisão

Função de Dispersão: Método da divisão

Função: $h(x) = x \pmod{m}$

Função de Dispersão: Método da divisão

Função: $h(x) = x \pmod{m}$

Sobre a escolha de m:

Função de Dispersão: Método da divisão

Função: $h(x) = x \pmod{m}$
Sobre a escolha de m:

- **Se m par:**

Função de Dispersão: Método da divisão

Função: $h(x) = x \pmod{m}$

Sobre a escolha de m:

- **Se** m **par:** x par

Função de Dispersão: Método da divisão

Função: $h(x) = x \pmod{m}$

Sobre a escolha de m:

- **Se m par:** x par $\Rightarrow h(x)$ par;

Função de Dispersão: Método da divisão

Função: $h(x) = x \pmod{m}$

Sobre a escolha de m:

- **Se m par:** x par $\Rightarrow h(x)$ par;
- **Se m ímpar:**

Função de Dispersão: Método da divisão

Função: $h(x) = x \pmod{m}$

Sobre a escolha de m:

- **Se m par:** x par $\Rightarrow h(x)$ par;
- **Se m ímpar:** x ímpar

Função de Dispersão: Método da divisão

Função: $h(x) = x \pmod{m}$

Sobre a escolha de m:

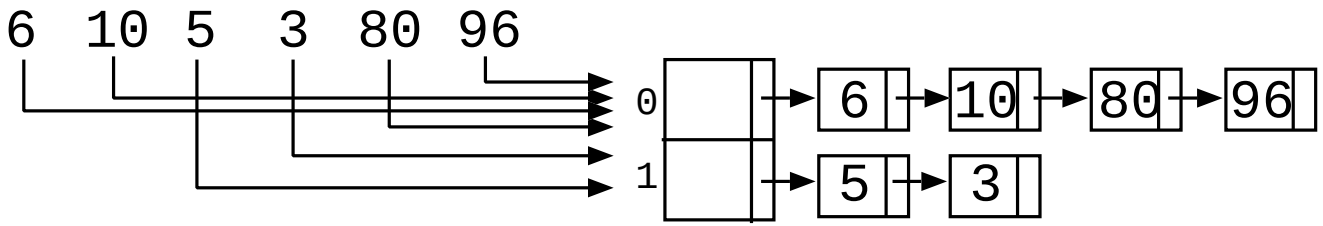
- **Se m par:** x par $\Rightarrow h(x)$ par;
- **Se m ímpar:** x ímpar $\Rightarrow h(x)$ ímpar;

Função de Dispersão: Método da divisão

Função: $h(x) = x \pmod{m}$

Sobre a escolha de m:

- **Se m par:** x par $\Rightarrow$ $h(x)$ par;
- **Se m ímpar:** x ímpar $\Rightarrow$ $h(x)$ ímpar;



h(x)=x mod 2

Função de Dispersão: alguns critérios para escolha de *m*

Função de Dispersão: alguns critérios para escolha de *m*

- *m* é número primo não próximo a potência de 2;

Função de Dispersão: alguns critérios para escolha de *m*

- *m* é número primo não próximo a potência de 2;
- *m* não possui divisores primos menores que 20;

Função de Dispersão: alguns critérios para escolha de m

- m é número primo não próximo a potência de 2;
- m não possui divisores primos menores que 20;
- se $m = 2^j$: $h(x)$ depende apenas últimos j dígitos de x;

Função de Dispersão: alguns critérios para escolha de m

- m é número primo não próximo a potência de 2;
- m não possui divisores primos menores que 20;
- se $m = 2^j$: $h(x)$ depende apenas últimos j dígitos de x;

Inteiro	Binário	resto divisão por $2^3 = 8$
0	00000	0
1	00001	1
2	00010	2
3	00011	3
4	00100	4
5	00101	5
6	00110	6
7	00111	7
8	01000	0
9	01001	1
10	01010	2
11	01011	3
12	01100	4
13	01101	5
14	01110	6
15	01111	7
16	10000	0

Função de Dispersão: alguns critérios para escolha de m

- m é número primo não próximo a potência de 2;
- m não possui divisores primos menores que 20;
- se $m = 2^j$: $h(x)$ depende apenas últimos j dígitos de x;

Inteiro	Binário	resto divisão por $2^3 = 8$
0	00**000**	0
1	00**001**	1
2	00**010**	2
3	00**011**	3
4	00**100**	4
5	00**101**	5
6	00**110**	6
7	00**111**	7
8	01**000**	0
9	01**001**	1
10	01**010**	2
11	01**011**	3
12	01**100**	4
13	01**101**	5
14	01**110**	6
15	01**111**	7
16	10**000**	0

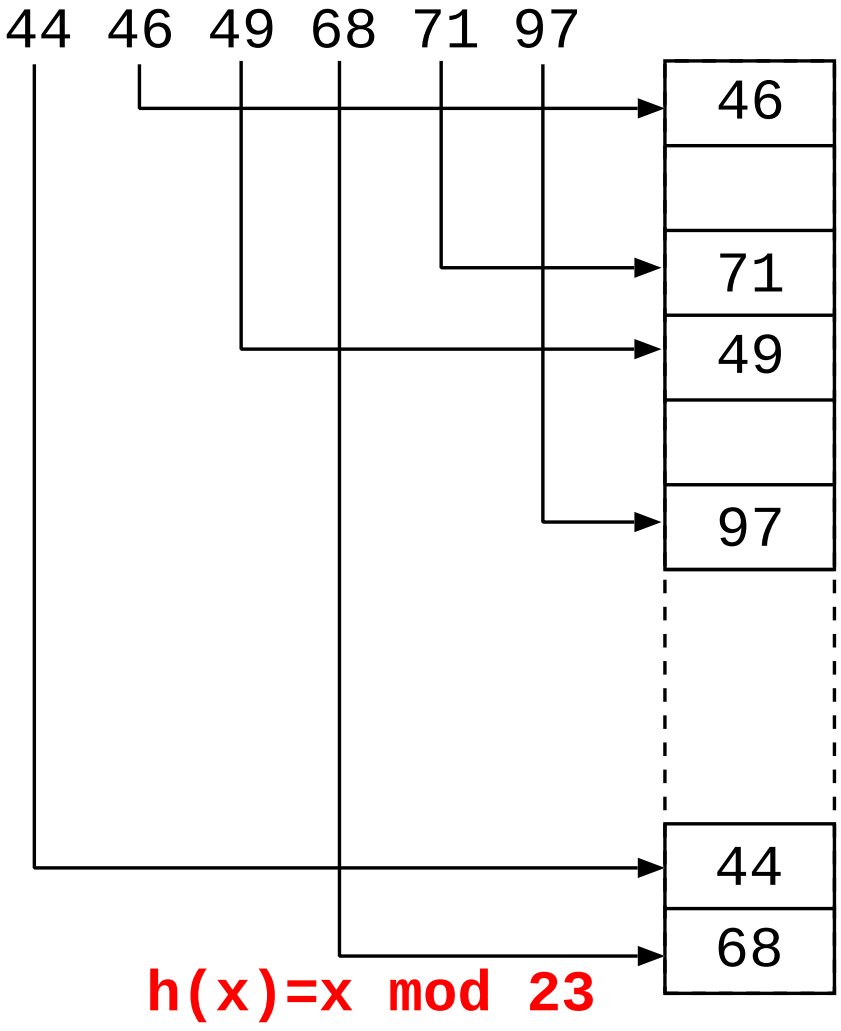
Função de Dispersão: alguns critérios para escolha de *m*

Função de Dispersão: alguns critérios para escolha de *m*

m primo:

Função de Dispersão: alguns critérios para escolha de *m*

m primo:



Hashing: Tratamento de Colisões

Hashing: Tratamento de Colisões

- Tratamento de colisões é procedimento crítico e deve ser realizado com cuidado.

Hashing: Tratamento de Colisões

- Tratamento de colisões é procedimento crítico e deve ser realizado com cuidado.
- **Fator de carga:** $\alpha = \frac{n}{m}$.

Hashing: Tratamento de Colisões

- Tratamento de colisões é procedimento crítico e deve ser realizado com cuidado.
- **Fator de carga:** $\alpha = \frac{n}{m}$.
- Quanto menor o fator de carga, menos colisões.

Hashing: Tratamento de Colisões

- Tratamento de colisões é procedimento crítico e deve ser realizado com cuidado.
- **Fator de carga:** $\alpha = \frac{n}{m}$.
- Quanto menor o fator de carga, menos colisões.
- Fator de carga pequeno não garante ausência de colisões.

Hashing: Tratamento de Colisões

- Tratamento de colisões é procedimento crítico e deve ser realizado com cuidado.
- **Fator de carga:** $\alpha = \frac{n}{m}$.
- 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.

Hashing: Tratamento de Colisões

- Tratamento de colisões é procedimento crítico e deve ser realizado com cuidado.
- **Fator de carga:** $\alpha = \frac{n}{m}$.
- 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: $\alpha = \frac{23}{365} = 0,063$.

Hashing: Tratamento de Colisões

- Tratamento de colisões é procedimento crítico e deve ser realizado com cuidado.
- **Fator de carga:** $\alpha = \frac{n}{m}$.
- 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: $\alpha = \frac{23}{365} = 0,063$.
- Mesmo assim, há 50% de chance de ocorrer pelo menos uma colisão.

Hashing: Tratamento de Colisões

Hashing: Tratamento de Colisões

Algumas estratégias:

Hashing: Tratamento de Colisões

Algumas estratégias:

1. Tratamento de Colisões por Encadeamento:

Hashing: Tratamento de Colisões

Algumas estratégias:

1. Tratamento de Colisões por Encadeamento:
   - **1.1** Exterior;

Hashing: Tratamento de Colisões

Algumas estratégias:

1. Tratamento de Colisões por Encadeamento:
   - **1.1** Exterior;
   - **1.2** Interior;

Hashing: Tratamento de Colisões

Algumas estratégias:

1. Tratamento de Colisões por Encadeamento:
   - **1.1** Exterior;
   - **1.2** Interior;
2. Tratamento de Colisões por Endereçamento Aberto:

Hashing: Tratamento de Colisões

Algumas estratégias:

1. Tratamento de Colisões por Encadeamento:
   - **1.1** Exterior;
   - **1.2** Interior;
2. Tratamento de Colisões por Endereçamento Aberto:
   - **2.1** Tentativa Linear;

Hashing: Tratamento de Colisões

Algumas estratégias:

1. Tratamento de Colisões por Encadeamento:
   - **1.1** Exterior;
   - **1.2** Interior;
2. Tratamento de Colisões por Endereçamento Aberto:
   - **2.1** Tentativa Linear;
   - **2.2** Tentativa Quadrática;

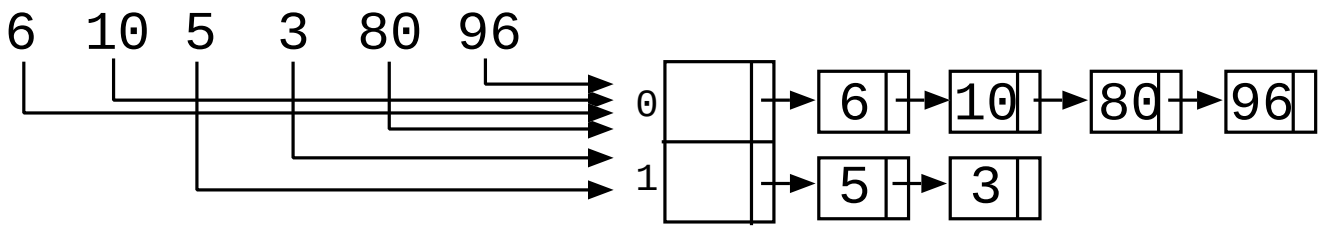
Tratamento de Colisões: Encadeamento Exterior

Tratamento de Colisões: Encadeamento Exterior

- Cada endereço-base mantém uma lista encadeada;

Tratamento de Colisões: Encadeamento Exterior

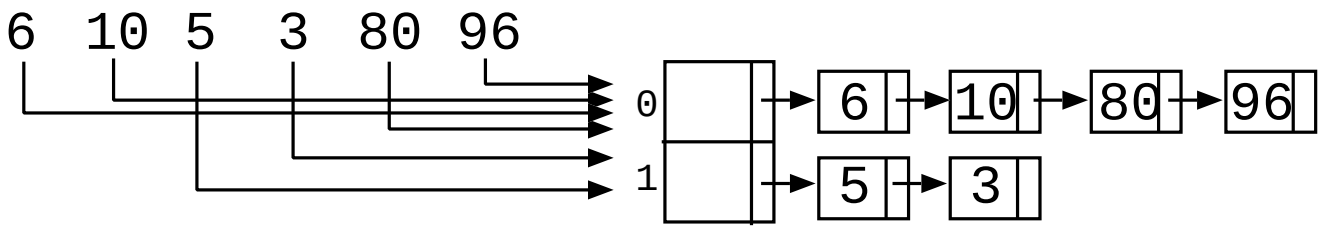
- Cada endereço-base mantém uma lista encadeada;



h(x)=x mod 2

Tratamento de Colisões: Encadeamento Exterior

- Cada endereço-base mantém uma lista encadeada;

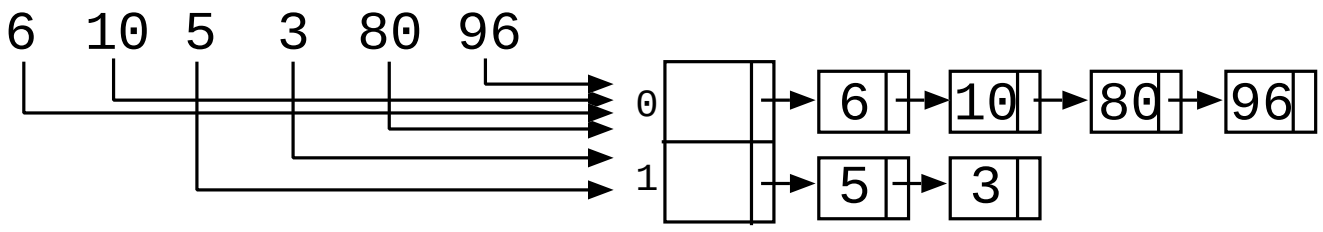


h(x)=x mod 2

- chaves inseridas no final da lista que tem que ser percorrida para verificar se chave já não existe;

Tratamento de Colisões: Encadeamento Exterior

- Cada endereço-base mantém uma lista encadeada;

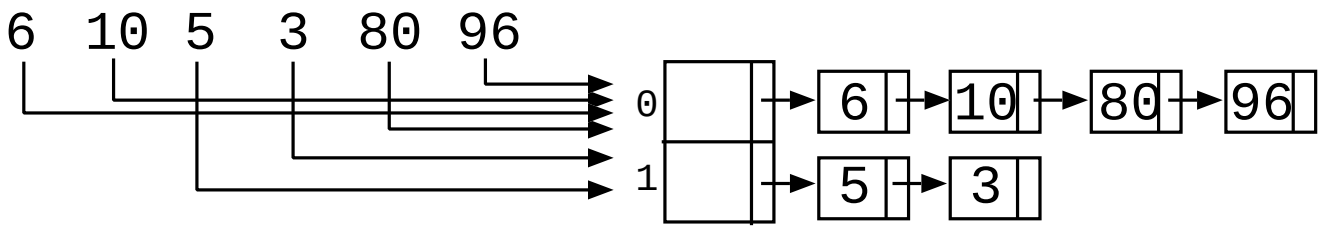


h(x)=x mod 2

- chaves inseridas no final da lista que tem que ser percorrida para verificar se chave já não existe;
- se função *hashing* é uniforme, busca tem tempo $O(1)$;

Tratamento de Colisões: Encadeamento Exterior

- Cada endereço-base mantém uma lista encadeada;



h(x)=x mod 2

- chaves inseridas no final da lista que tem que ser percorrida para verificar se chave já não existe;
- se função *hashing* é uniforme, busca tem tempo $O(1)$;
- lista encadeada pode ser implementada como árvore AVL;

Tratamento de Colisões: Encadeamento Interior

Tratamento de Colisões: Encadeamento Interior

- **Duas zonas na tabela com $m = p + s$:**

Tratamento de Colisões: Encadeamento Interior

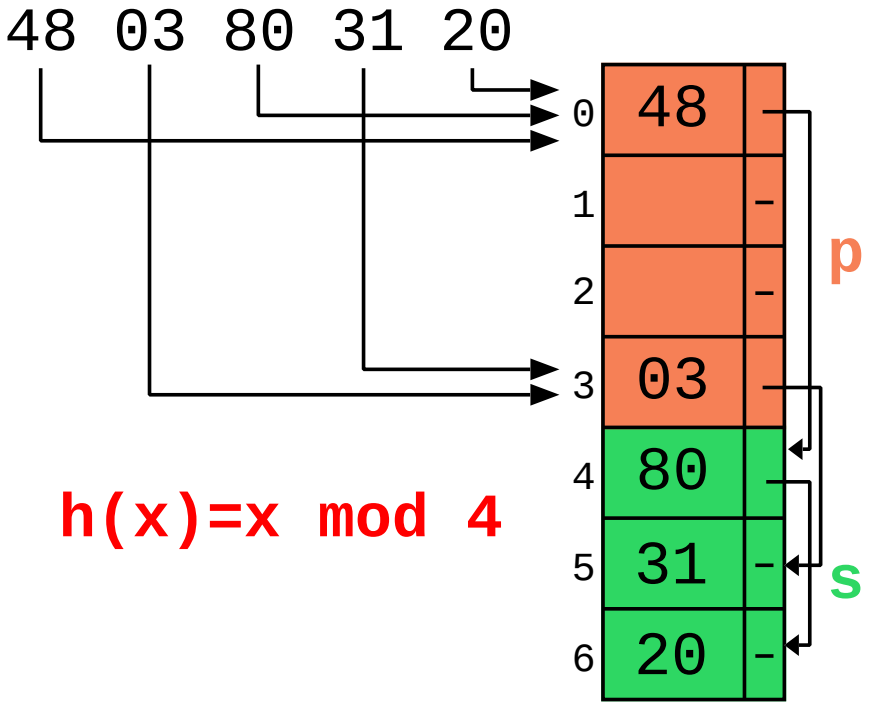
- **Duas zonas na tabela com $m = p + s$:**
  1. uma de endereços-base de tamanho p;

Tratamento de Colisões: 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;

Tratamento de Colisões: 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*;



Tratamento de Colisões: Encadeamento Interior

Tratamento de Colisões: Encadeamento Interior

Problemas:

Tratamento de Colisões: Encadeamento Interior

Problemas:

- **possibilidade de falso *overflow***: zona de sinônimos cheia e zona de endereços-base ainda com vazios;

Tratamento de Colisões: Encadeamento Interior

Problemas:

- **possibilidade de falso *overflow***: zona de sinônimos cheia e zona de endereços-base ainda com vazios;
- **Para evitar falso *overflow*:** aumentar zona de colisões;

Tratamento de Colisões: Encadeamento Interior

Problemas:

- **possibilidade de falso *overflow***: zona de sinônimos cheia e zona de endereços-base ainda com vazios;
- **Para evitar falso *overflow*:** aumentar zona de colisões;
- **Aumentar zona de colisões:** diminui eficiência da tabela;

Tratamento de Colisões: Encadeamento Interior

Problemas:

- **possibilidade de falso *overflow***: zona de sinônimos cheia e zona de endereços-base ainda com vazios;
- **Para evitar falso *overflow*:** aumentar zona de colisões;
- **Aumentar zona de colisões:** diminui eficiência da tabela;
- $p = 1$ e $s = m - 1 \Rightarrow$

Tratamento de Colisões: Encadeamento Interior

Problemas:

- **possibilidade de falso *overflow***: zona de sinônimos cheia e zona de endereços-base ainda com vazios;
- **Para evitar falso *overflow*:** aumentar zona de colisões;
- **Aumentar zona de colisões:** diminui eficiência da tabela;
- $p = 1$ e $s = m - 1 \Rightarrow$ **lista encadeada!!!**

Tratamento de Colisões: Encadeamento Interior

Opção: não diferenciar duas zonas da tabela

Tratamento de Colisões: Encadeamento Interior

Opção: não diferenciar duas zonas da tabela

- qualquer endereço pode ser de base ou de sinônimo;

Tratamento de Colisões: Encadeamento Interior

Opção: não diferenciar duas zonas da tabela

- qualquer endereço pode ser de base ou de sinônimo;
- **Problema:** colisões secundárias;

Tratamento de Colisões: Encadeamento Interior

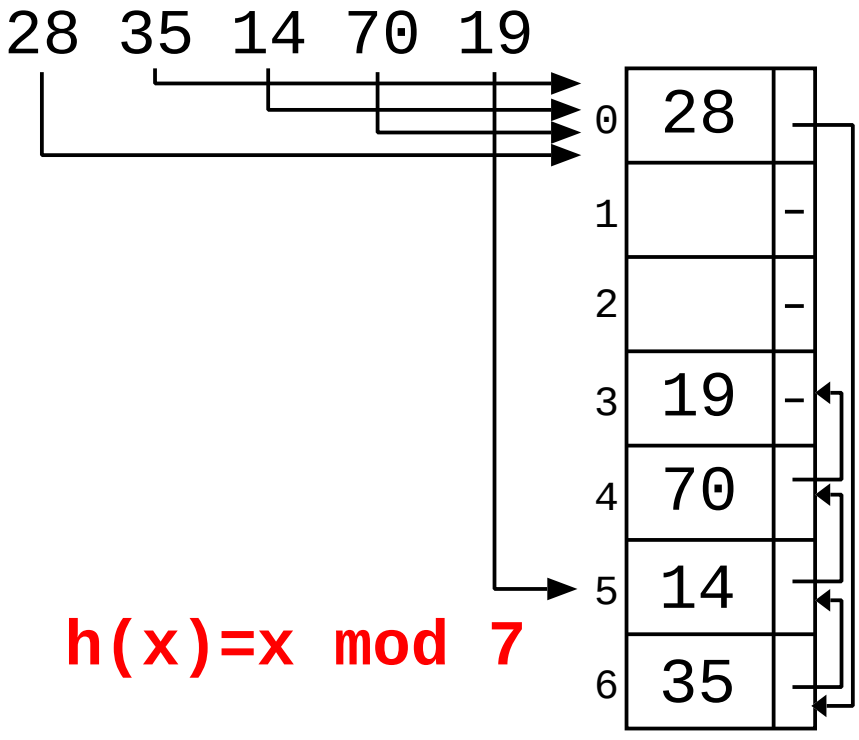
Opção: não diferenciar duas zonas da tabela

- qualquer endereço pode ser de base ou de sinônimo;
- **Problema:** colisões secundárias;
- **Fusão de listas diferentes:** diminuição de eficiência;

Tratamento de Colisões: Encadeamento Interior

Opção: não diferenciar duas zonas da tabela

- qualquer endereço pode ser de base ou de sinônimo;
- **Problema:** colisões secundárias;
- **Fusão de listas diferentes:** diminuição de eficiência;



Tratamento de Colisões: Endereçamento Aberto

Tratamento de Colisões: Endereçamento Aberto

- Idéia: armazenar chaves sinônimas também na tabela;

Tratamento de Colisões: Endereçamento Aberto

- Idéia: armazenar chaves sinônimas também na tabela;
- Sem uso de ponteiros;

Tratamento de Colisões: Endereçamento Aberto

- Idéia: armazenar chaves sinônimas também na tabela;
- Sem uso de ponteiros;
- Calcula próximo compartimento a ser examinado;

Tratamento de Colisões: Endereçamento Aberto

- Idéia: armazenar chaves sinônimas também na tabela;
- Sem uso de ponteiros;
- Calcula próximo compartimento a ser examinado;
- **Busca com sucesso:** termina quando encontra chave;

Tratamento de Colisões: Endereçamento Aberto

- Idéia: armazenar chaves sinônimas também na tabela;
- Sem uso de ponteiros;
- Calcula próximo compartimento a ser examinado;
- **Busca com sucesso:** termina quando encontra chave;
- **Busca sem sucesso:** termina quando encontra compartimento vazio ou exaure tabela;

Tratamento de Colisões: Endereçamento Aberto

- Idéia: armazenar chaves sinônimas também na tabela;
- Sem uso de ponteiros;
- Calcula próximo compartimento a ser examinado;
- **Busca com sucesso:** termina quando encontra chave;
- **Busca sem sucesso:** termina quando encontra compartimento vazio ou exaure tabela;
- **Função *hash*:** fornece m endereços-base $h(x, k)$ para chave x e $k = 0, \ldots, m - 1$ tentativas;

Tratamento de Colisões: Endereçamento Aberto

- Idéia: armazenar chaves sinônimas também na tabela;
- Sem uso de ponteiros;
- Calcula próximo compartimento a ser examinado;
- **Busca com sucesso:** termina quando encontra chave;
- **Busca sem sucesso:** termina quando encontra compartimento vazio ou exaure tabela;
- **Função *hash*:** fornece m endereços-base $h(x, k)$ para chave x e $k = 0, \ldots, m - 1$ tentativas;
- **Endereço-base:** $h(x, 0)$;

Tratamento de Colisões: Endereçamento Aberto

- Idéia: armazenar chaves sinônimas também na tabela;
- Sem uso de ponteiros;
- Calcula próximo compartimento a ser examinado;
- **Busca com sucesso:** termina quando encontra chave;
- **Busca sem sucesso:** termina quando encontra compartimento vazio ou exaure tabela;
- **Função *hash*:** fornece m endereços-base $h(x, k)$ para chave x e $k = 0, \ldots, m - 1$ tentativas;
- **Endereço-base:** $h(x, 0)$;
- **Se colisão:** $h(x, 0)$, $h(x, 1)$, $h(x, 2)$, $\ldots$, $h(x, m - 1)$;

Tratamento de Colisões: Endereçamento Aberto

Tratamento de Colisões: Endereçamento Aberto

Busca por endereçamento aberto em pseudo-código:

Tratamento de Colisões: Endereçamento Aberto

Busca por endereçamento aberto em pseudo-código:

```
1  buscaAberto(x, end, a):
2      a ← 3; k ← 0;
3      enquanto k < m faça
4          end ← h(x, k);
5          se (T[end].chave = x) então
6              a ← 1;
7              k ← m;
8          senão se (T[end].chave = null) então
9              a ← 2;
10             k ← m;
11         senão k ← k + 1;
```

onde $a = 1$ indica chave localizada, $a = 2$ ou 3 indica chave não localizada. Se $a = 2$, *end* indica posição livre

Endereçamento Aberto: Tentativa Linear

Endereçamento Aberto: Tentativa Linear

Tentativa Linear:

Endereçamento Aberto: Tentativa Linear

Tentativa Linear:

- Suponha endereço da chave x é $h'(x)$, ou seja, $h'(x) = h(x, 0)$
- $h(x, k) = (h'(x) + k) \pmod{m}, 0 \leq k \leq m - 1$.

Endereçamento Aberto: Tentativa Linear

Tentativa Linear:

- Suponha endereço da chave x é $h'(x)$, ou seja, $h'(x) = h(x, 0)$
- $h(x, k) = (h'(x) + k) \pmod{m}, 0 \leq k \leq m - 1$.

que é o mesmo que:

Endereçamento Aberto: Tentativa Linear

Tentativa Linear:

- Suponha endereço da chave x é $h'(x)$, ou seja, $h'(x) = h(x, 0)$
- $h(x, k) = (h'(x) + k) \pmod{m}, 0 \leq k \leq m - 1.$

que é o mesmo que:

- $h(x, k) = (h(x, k - 1) + 1) \pmod{m}, 0 \leq k \leq m - 1.$

Endereçamento Aberto: Tentativa Linear

Tentativa Linear:

- Suponha endereço da chave x é $h'(x)$, ou seja, $h'(x) = h(x, 0)$
- $h(x, k) = (h'(x) + k) \pmod{m}, 0 \leq k \leq m - 1.$

que é o mesmo que:

- $h(x, k) = (h(x, k - 1) + 1) \pmod{m}, 0 \leq k \leq m - 1.$

Exemplo:

Endereçamento Aberto: Tentativa Linear

Tentativa Linear:

- Suponha endereço da chave x é $h'(x)$, ou seja, $h'(x) = h(x, 0)$
- $h(x, k) = (h'(x) + k) \pmod{m}, 0 \leq k \leq m - 1.$

que é o mesmo que:

- $h(x, k) = (h(x, k-1) + 1) \pmod{m}, 0 \leq k \leq m - 1.$

Exemplo:

$$
\begin{array}{lll}
h(x,0) & = 1 & = 1 \\
h(x,1) & = (h(x,0) + 1) \pmod{m} & = 2 \\
h(x,2) & = (h(x,1) + 1) \pmod{m} & = 3 \\
h(x,3) & = (h(x,2) + 1) \pmod{m} & = 4 \\
\vdots & = \quad \vdots & = \quad \vdots
\end{array}
$$

Endereçamento Aberto: Tentativa Linear

Endereçamento Aberto: Tentativa Linear

Tentativa Linear:

Endereçamento Aberto: Tentativa Linear

Tentativa Linear:

- **Desvantagem:** agrupamento primário;

Endereçamento Aberto: Tentativa Linear

Tentativa Linear:

- **Desvantagem:** agrupamento primário;
- **Agrupamento primário:** longos trechos consecutivos de memória ocupados;

Endereçamento Aberto: Tentativa Linear

Tentativa Linear:

- **Desvantagem:** agrupamento primário;
- **Agrupamento primário:** longos trechos consecutivos de memória ocupados;
- quanto maior o agrupamento primário, maior probabilidade de aumentá-lo mais;

Endereçamento Aberto: Tentativa Quadrática

Endereçamento Aberto: Tentativa Quadrática

Tentativa Quadrática:

Endereçamento Aberto: Tentativa Quadrática

Tentativa Quadrática:

- Suponha endereço da chave x é $h'(x)$, ou seja, $h'(x) = h(x, 0)$
- $h(x, k) = (h(x, k-1) + k) \pmod{m}, 0 < k < m$.

Endereçamento Aberto: Tentativa Quadrática

Tentativa Quadrática:

- Suponha endereço da chave x é $h'(x)$, ou seja, $h'(x) = h(x, 0)$
- $h(x, k) = (h(x, k-1) + k) \pmod{m}, 0 < k < m$.

Exemplo:

Endereçamento Aberto: Tentativa Quadrática

Tentativa Quadrática:

- Suponha endereço da chave x é $h'(x)$, ou seja, $h'(x) = h(x, 0)$
- $h(x, k) = (h(x, k-1) + k) \pmod{m}, 0 < k < m.$

Exemplo:

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

Endereçamento Aberto: Tentativa Quadrática

Endereçamento Aberto: Tentativa Quadrática

Tentativa Quadrática:

Endereçamento Aberto: Tentativa Quadrática

Tentativa Quadrática:

- evita agrupamentos primários;

Endereçamento Aberto: Tentativa Quadrática

Tentativa Quadrática:

- evita agrupamentos primários;
- produz agrupamentos secundários;

Endereçamento Aberto: Tentativa Quadrática

Tentativa Quadrática:

- evita agrupamentos primários;
- produz agrupamentos secundários;
- degradação produzida pelos agrupamentos secundários ainda é menor que a produzida pelos agrupamentos primários;

Conclusão

Conclusão

- Complexidade média por operação: $O(1)$;

Conclusão

- Complexidade média por operação: $O(1)$;
- Tabela *hash* é estrutura de dados que não permite armazenar elementos repetidos;

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);

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;

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;

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

Aplicações

- **busca de elementos em base de dados:**

Aplicações

- **busca de elementos em base de dados:**
  - estruturas de dados em memória;

Aplicações

- **busca de elementos em base de dados:**
  - estruturas de dados em memória;
  - bancos de dados;

Aplicações

- **busca de elementos em base de dados:**
  - estruturas de dados em memória;
  - bancos de dados;
  - mecanismos de busca na Internet;

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:**

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*;

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;

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;

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;

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;

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;

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;
- **Exemplos de *hash* (criptográficos):** *MD*5 e família *SHA* (*SHA* – 2, *SHA* – 256 e *SHA* – 512);

Aplicações: mecanismos de busca na Internet

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;
2. *spiders* começam busca em servidores mais utilizados e páginas populares;

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;
2. *spiders* começam busca em servidores mais utilizados e páginas populares;
3. *spider* começa busca em site popular, indexa suas palavras (e onde encontrou) e segue por seus *links*;

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;
2. *spiders* começam busca em servidores mais utilizados e páginas populares;
3. *spider* começa busca em site popular, indexa suas palavras (e onde encontrou) e segue por seus *links*;
4. **atenção especial:** *meta-tags*, títulos, subtítulos;

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;
2. *spiders* começam busca em servidores mais utilizados e páginas populares;
3. *spider* começa busca em site popular, indexa suas palavras (e onde encontrou) e segue por seus *links*;
4. **atenção especial:** *meta-tags*, títulos, subtítulos;
5. **Google:** desconsidera artigos (“um”, “uma”, “o”, “a”);

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;
2. *spiders* começam busca em servidores mais utilizados e páginas populares;
3. *spider* começa busca em site popular, indexa suas palavras (e onde encontrou) e segue por seus *links*;
4. **atenção especial:** *meta-tags*, títulos, subtítulos;
5. **Google:** desconsidera artigos (“um”, “uma”, “o”, “a”);
6. **Construção do índice:** tabela *hash*;

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;
2. *spiders* começam busca em servidores mais utilizados e páginas populares;
3. *spider* começa busca em site popular, indexa suas palavras (e onde encontrou) e segue por seus *links*;
4. **atenção especial:** *meta-tags*, títulos, subtítulos;
5. **Google:** desconsidera artigos (“um”, “uma”, “o”, “a”);
6. **Construção do índice:** tabela *hash*;
7. **usa peso:** expressa número ocorrências de palavra, onde aparece (título, meta-tags, etc), se maiúscula, fonte, etc;

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;
2. *spiders* começam busca em servidores mais utilizados e páginas populares;
3. *spider* começa busca em site popular, indexa suas palavras (e onde encontrou) e segue por seus *links*;
4. **atenção especial:** *meta-tags*, títulos, subtítulos;
5. **Google:** desconsidera artigos (“um”, “uma”, “o”, “a”);
6. **Construção do índice:** tabela *hash*;
7. **usa peso:** expressa número ocorrências de palavra, onde aparece (título, meta-tags, etc), se maiúscula, fonte, etc;

Combinação de

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;
2. *spiders* começam busca em servidores mais utilizados e páginas populares;
3. *spider* começa busca em site popular, indexa suas palavras (e onde encontrou) e segue por seus *links*;
4. **atenção especial:** *meta-tags*, títulos, subtítulos;
5. **Google:** desconsidera artigos (“um”, “uma”, “o”, “a”);
6. **Construção do índice:** tabela *hash*;
7. **usa peso:** expressa número ocorrências de palavra, onde aparece (título, meta-tags, etc), se maiúscula, fonte, etc;

Combinação de **indexação eficiente**

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;
2. *spiders* começam busca em servidores mais utilizados e páginas populares;
3. *spider* começa busca em site popular, indexa suas palavras (e onde encontrou) e segue por seus *links*;
4. **atenção especial:** *meta-tags*, títulos, subtítulos;
5. **Google:** desconsidera artigos (“um”, “uma”, “o”, “a”);
6. **Construção do índice:** tabela *hash*;
7. **usa peso:** expressa número ocorrências de palavra, onde aparece (título, meta-tags, etc), se maiúscula, fonte, etc;

Combinação de **indexação eficiente** +

Aplicações: mecanismos de busca na Internet

1. ***spiders***: constroem listas de palavras dos Web sites;
2. *spiders* começam busca em servidores mais utilizados e páginas populares;
3. *spider* começa busca em site popular, indexa suas palavras (e onde encontrou) e segue por seus *links*;
4. **atenção especial:** *meta-tags*, títulos, subtítulos;
5. **Google:** desconsidera artigos (“um”, “uma”, “o”, “a”);
6. **Construção do índice:** tabela *hash*;
7. **usa peso:** expressa número ocorrências de palavra, onde aparece (título, meta-tags, etc), se maiúscula, fonte, etc;

Combinação de **indexação eficiente** + **armazenagem efetiva**;

Exercícios

Exercícios

1. Considere as técnicas de busca sequencial, busca binária e busca baseada em *hashing*:

Exercícios

1. Considere as técnicas de busca sequencial, busca binária e busca baseada em *hashing*:
   - **1.1** Descreva as vantagens e desvantagens de cada uma dessas técnicas, indicando em que situações você usaria cada uma delas.

Exercícios

1. Considere as técnicas de busca sequencial, busca binária e busca baseada em *hashing*:
   - **1.1** Descreva as vantagens e desvantagens de cada uma dessas técnicas, indicando em que situações você usaria cada uma delas.
   - **1.2** Qual é a eficiência de utilização da memória (relação entre o espaço necessário para dados e o espaço total necessário) para cada método?

Exercícios

1. Considere as técnicas de busca sequencial, busca binária e busca baseada em *hashing*:
   - **1.1** Descreva as vantagens e desvantagens de cada uma dessas técnicas, indicando em que situações você usaria cada uma delas.
   - **1.2** Qual é a eficiência de utilização da memória (relação entre o espaço necessário para dados e o espaço total necessário) para cada método?
2. Quais as características de uma boa função *hash*?

Exercícios

1. Considere as técnicas de busca sequencial, busca binária e busca baseada em *hashing*:
   - **1.1** Descreva as vantagens e desvantagens de cada uma dessas técnicas, indicando em que situações você usaria cada uma delas.
   - **1.2** Qual é a eficiência de utilização da memória (relação entre o espaço necessário para dados e o espaço total necessário) para cada método?
2. Quais as características de uma boa função *hash*?
3. Em que situações a tabela *hash* deve ser utilizada?

Exercícios

Exercícios

4. Descreva dois mecanismos diferentes para resolver o problema de colisões de várias chaves em uma mesma posição da tabela, destacando as vantagens e desvantagens de cada método.

Exercícios

4. Descreva dois mecanismos diferentes para resolver o problema de colisões de várias chaves em uma mesma posição da tabela, destacando as vantagens e desvantagens de cada método.
5. Para um conjunto de *n* chaves *x* formada pelos *n* primeiros múltiplos de 7, quantas colisões são obtidas para as funções de dispersão abaixo?

Exercícios

4. Descreva dois mecanismos diferentes para resolver o problema de colisões de várias chaves em uma mesma posição da tabela, destacando as vantagens e desvantagens de cada método.
5. Para um conjunto de n chaves x formada pelos n primeiros múltiplos de 7, quantas colisões são obtidas para as funções de dispersão abaixo?
   - **5.1** $x \pmod 7$

Exercícios

4. Descreva dois mecanismos diferentes para resolver o problema de colisões de várias chaves em uma mesma posição da tabela, destacando as vantagens e desvantagens de cada método.

5. Para um conjunto de n chaves x formada pelos n primeiros múltiplos de 7, quantas colisões são obtidas para as funções de dispersão abaixo?

5.1 $x \pmod 7$

5.2 $x \pmod{14}$

Exercícios

4. Descreva dois mecanismos diferentes para resolver o problema de colisões de várias chaves em uma mesma posição da tabela, destacando as vantagens e desvantagens de cada método.

5. Para um conjunto de *n* chaves *x* formada pelos *n* primeiros múltiplos de 7, quantas colisões são obtidas para as funções de dispersão abaixo?

5.1 $x \pmod 7$

5.2 $x \pmod{14}$

5.3 $x \pmod 5$

Exercícios

4. Descreva dois mecanismos diferentes para resolver o problema de colisões de várias chaves em uma mesma posição da tabela, destacando as vantagens e desvantagens de cada método.
5. Para um conjunto de *n* chaves *x* formada pelos *n* primeiros múltiplos de 7, quantas colisões são obtidas para as funções de dispersão abaixo?
   - **5.1** $x \pmod 7$
   - **5.2** $x \pmod{14}$
   - **5.3** $x \pmod 5$
6. Descreva algoritmos de busca, inserção e remoção em uma tabela *hash* com tratamento de colisões por encadeamento exterior.

Exercícios

4. Descreva dois mecanismos diferentes para resolver o problema de colisões de várias chaves em uma mesma posição da tabela, destacando as vantagens e desvantagens de cada método.
5. Para um conjunto de *n* chaves *x* formada pelos *n* primeiros múltiplos de 7, quantas colisões são obtidas para as funções de dispersão abaixo?
   - **5.1** $x \pmod 7$
   - **5.2** $x \pmod{14}$
   - **5.3** $x \pmod 5$
6. Descreva algoritmos de busca, inserção e remoção em uma tabela *hash* com tratamento de colisões por encadeamento exterior.
7. Descreva algoritmos de busca e inserção em uma tabela *hash* com tratamento de colisões por encadeamento interior, supondo não-existência de exclusões.

Bibliografia

FRANKLIN, Curt. *“How Internet Search Engines Work”*. 27 September 2000. HowStuffWorks.com. $<$ *http* : $//$*computer*.*howstuffworks*.*com*$/$*internet*$/$*basics*$/$*search* $-$ *engine*.*htm* $>$ 14 October 2012.

SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

ZIVIANI, N. Projeto de Algoritmos: com implementações em Java e C++, 1a edição, Cengage Learning, 2009.

Perguntas?