TABELAS HASH

Prof. Muriel Mazzetto

Estrutura de Dados

Pesquisa

- A busca de um dado consiste em procurar a informação **comparando valores de chaves.**

Pesquisa

- A busca de um dado consiste em procurar a informação **comparando valores de chaves.**
- Os algoritmos mais eficientes exigem a ordenação dos dados:
  - Melhor ordenação: O(N log(N))
  - Melhor busca: O(log(N))

Pesquisa

- A busca de um dado consiste em procurar a informação **comparando valores de chaves.**
- Os algoritmos mais eficientes exigem a ordenação dos dados:
  - Melhor ordenação: O(N log(N))
  - Melhor busca: O(log(N))
- Busca ideal é quando se acessa o elemento diretamente, sem comparação de chaves
  - custo O(1).

Tabela Hash

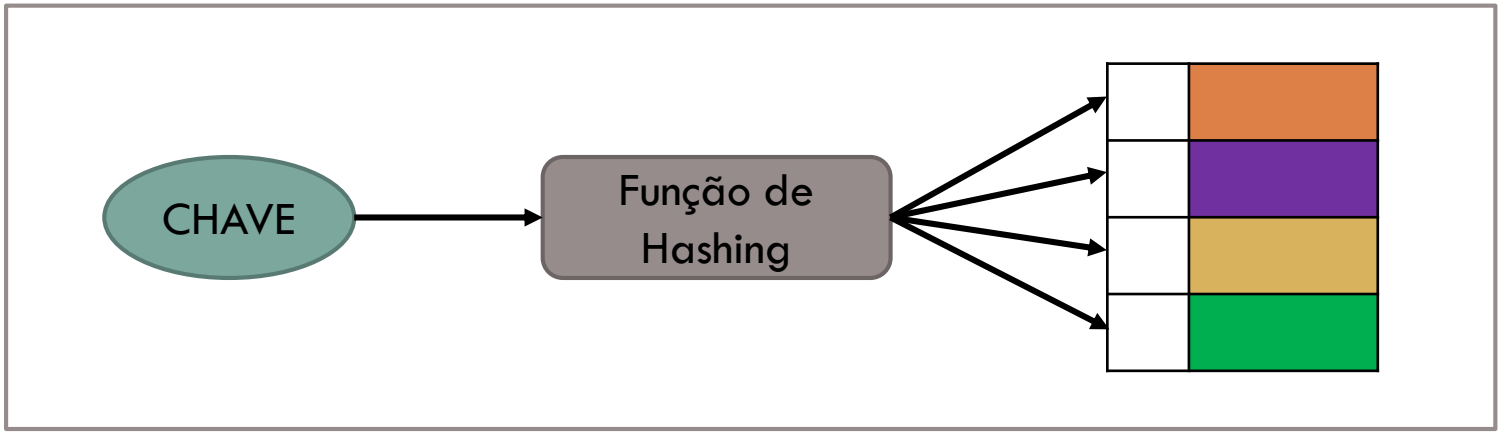
- Também conhecida como tabela de dispersão ou tabela de espalhamento.
- É uma estrutura de dados do tipo **dicionário**, que associa chaves de pesquisa aos valores.
- A partir de uma chave simples, faz uma busca rápida para obter o valor desejado.

Tabela Hash

- É uma generalização de um vetor.
- Utiliza uma **função de hashing** para determinar a posição de armazenamento dos elementos no vetor.
- Os elementos ficam dispersos de forma não ordenada no vetor.

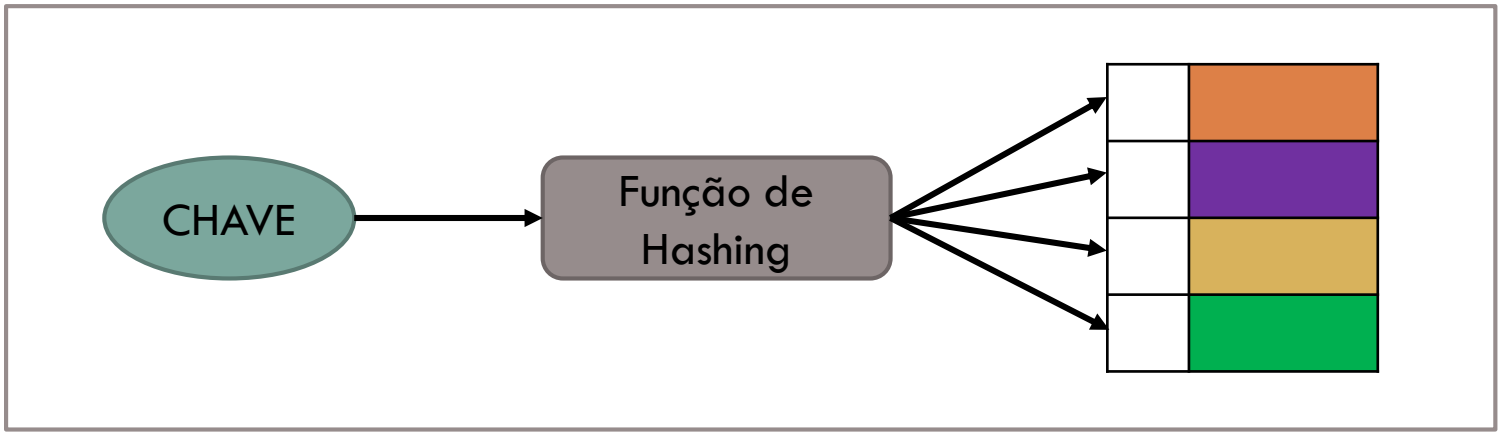
Tabela Hash

- É uma generalização de um vetor.
- Utiliza uma **função de hashing** para determinar a posição de armazenamento dos elementos no vetor.
- Os elementos ficam dispersos de forma não ordenada no vetor.



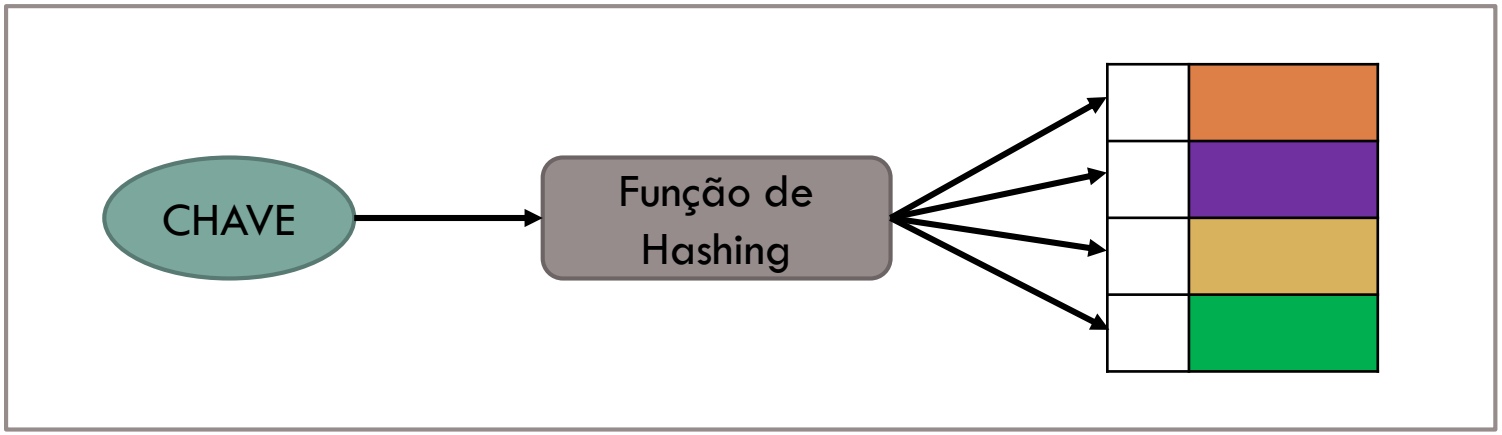
Função de Hashing

- É responsável por gerar um índice a partir de uma chave.



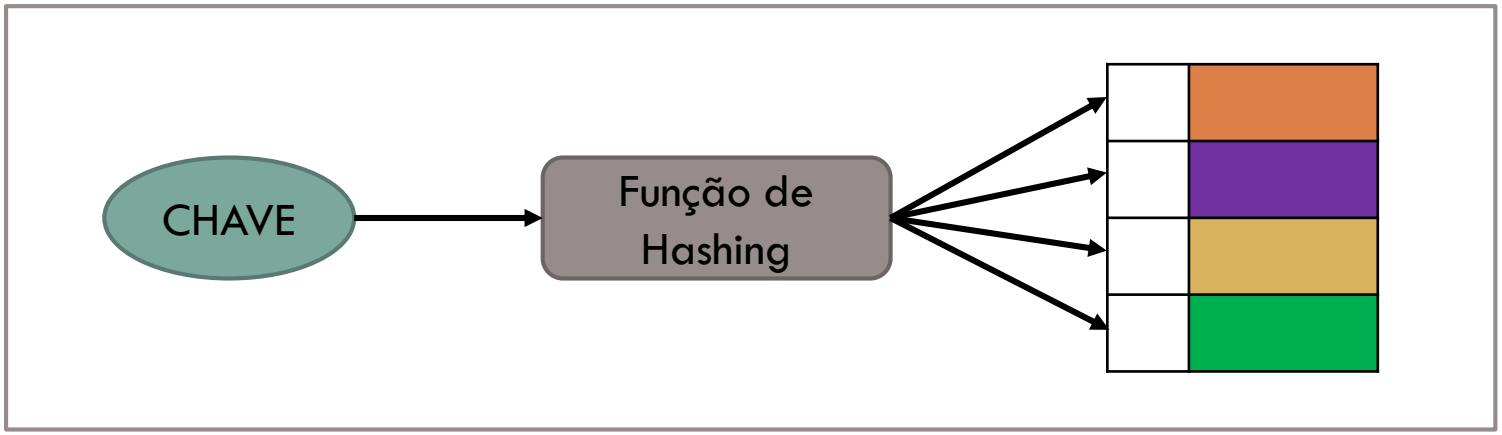
Função de Hashing

- É responsável por gerar um índice a partir de uma chave.
  - É **ideal** não gerar índice igual para chaves diferentes.
  - Evitar **colisões**, ou saber tratá-las.

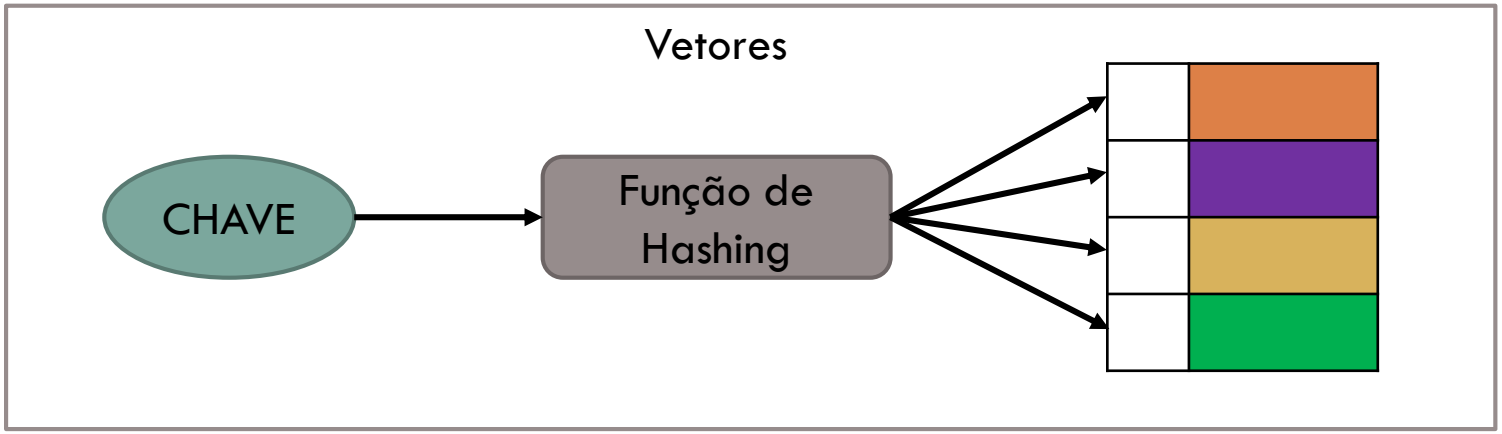


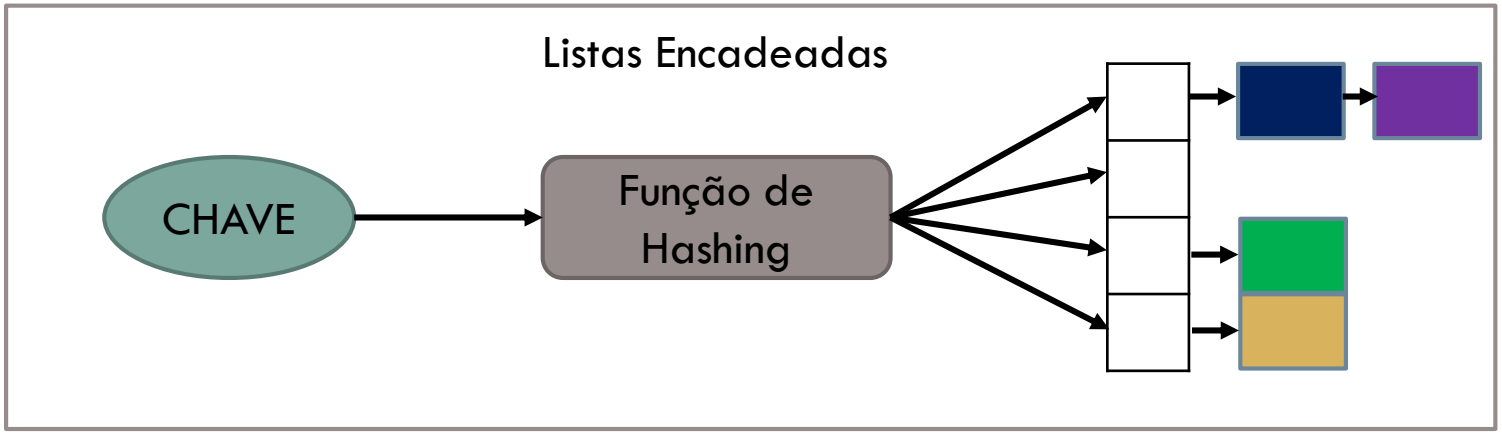
Função de Hashing

- É responsável por gerar um índice a partir de uma chave.
  - É **ideal** não gerar índice igual para chaves diferentes.
  - Evitar **colisões**, ou saber tratá-las.
- Convertem valores de tamanho variável em índices de tamanho fixo.



Função de Hashing





Função de Hashing

- O bom desempenho depende diretamente da escolha da função de hashing.
  - Deve distribuir os dados de maneira mais uniforme possível.
  - Minimizar a quantidade de colisões.

Função de Hashing

- **Mau Exemplo** de hashing:
  - h(C) = C%100.
  - C: chave de entrada.
  - h: índice gerado pelo hashing.

CHAVE (C)	ÍNDICE (h)
123456	56
7531	31
3677756	56

  - Gera colisão para muitos valores.

Função de Hashing

- **Bom Exemplo** de hashing:
  - h(C) = C%M.
  - C: chave de entrada.
  - M: tamanho do vetor (número primo).
  - h: índice gerado pelo hashing.

Função de Hashing

- **Bom Exemplo** de hashing:
  - h(C) = C%M.
  - C: chave de entrada.
  - M: tamanho do vetor (número primo).
  - h: índice gerado pelo hashing.

- Escolher um tamanho apropriado para o tamanho do vetor, dentro das potências de 2.
- Adotar o número primo logo abaixo da potência escolhida como o tamanho de M.

Função de Hashing

- **Bom Exemplo** de hashing:
  - h(C) = C%M.
  - C: chave de entrada.
  - M: tamanho do vetor (número primo).
  - h: índice gerado pelo hashing.
- Escolher um tamanho apropriado par do vetor, dentro das potências de 2.
- Adotar o número primo logo abaixo escolhida como o tamanho de M.

k	2^k	M
7	128	127
8	256	251
9	512	509
10	1024	1021
11	2048	2039
12	4096	4093
13	8192	8191
14	16384	16381
15	32768	32749
16	65536	65521
17	131072	131071
18	262144	262139

Função de Hashing

- **Bom Exemplo** de hashing:
  - h(C) = C%M.
  - C: chave de entrada.
  - M: tamanho do vetor (número primo).
  - h:

```
int hash(int v, int M)
{
    return v%M;
}
```

- Esc… par… do vetor, dentro das potências de 2.
- Adotar o número primo logo abaixo escolhida como o tamanho de M.

k	2^k	M
7	128	127
8	256	251
9	512	509
10	1024	1021
11	2048	2039
12	4096	4093
13	8192	8191
14	16384	16381
15	32768	32749
16	65536	65521
17	131072	131071
18	262144	262139

Função de Hashing

- Dados **não numéricos** devem ser convertidos em um valor inteiro.
- Método de Horner:

```
int hash(char* v, int M)
{
    int i, h = v[0];
    for (i = 1; v[i] != '\0'; i++)
        h = (h * 251 + v[i]) % M;
    return h;
}
```

Função de Hashing

- **Colisões:** ocorre quando a função de hashing calcula o mesmo índice para chaves diferentes.
  - Mais dados do que o tamanho do vetor.
  - Má escolha da função de hashing.
  - Principal problema de criptografia de senhas.

- Soluções comuns:
  - Endereçamento aberto
  - Encadeamento

Função de Hashing

- **Colisões:** Correção por Endereçamento Aberto:
  - Os dados estão armazenados diretamente no vetor de hash. Exige um bom dimensionamento.
  - Acesso aos dados de maneira mais rápida.
  - Formas de tratamento:
    - Tentativa linear.
    - Tentativa quadrática.
    - Dispersão dupla.

Função de Hashing

- **Colisões:** Correção por <u>Endereçamento Aberto:</u>
  - Tentativa linear: utiliza uma segunda função para recalcular a dispersão.
    - $rh(pos,\ tentativas) = (pos + tentativas)\%tam$

Função de Hashing

- **Colisões:** Correção por <u>Endereçamento Aberto:</u>
  - Tentativa quadrática: tenta diminuir o número de colisões gerados pela tentativa linear.
    - $rh(pos,\ tentativas) = (pos + tentativas^2)\%tam$

Função de Hashing

- **Colisões:** Correção por <u>Endereçamento Aberto:</u>
  - Dispersão dupla: duas funções de hashing associadas.
    - $rh(ch,\ tentativas) = (h2(ch) + tentativas)\%tam$
    - $h2(ch) = 1 + ch\%(tam - 1)$

Função de Hashing

- **Colisões:** Correção por <u>Endereçamento Aberto:</u>
  - Dispersão dupla: duas funções de hashing associadas.
    - $rh(ch,\ tentativas) = (h2(ch) + tentativas)\%tam$
    - $h2(ch) = 1 + ch\%(tam - 1)$
  - As funções de hashing devem gerar valores diferentes.
  - Método mais utilizado para tratar colisões.
  - A seleção da função vai depender do domínio do problema.

Função de Hashing

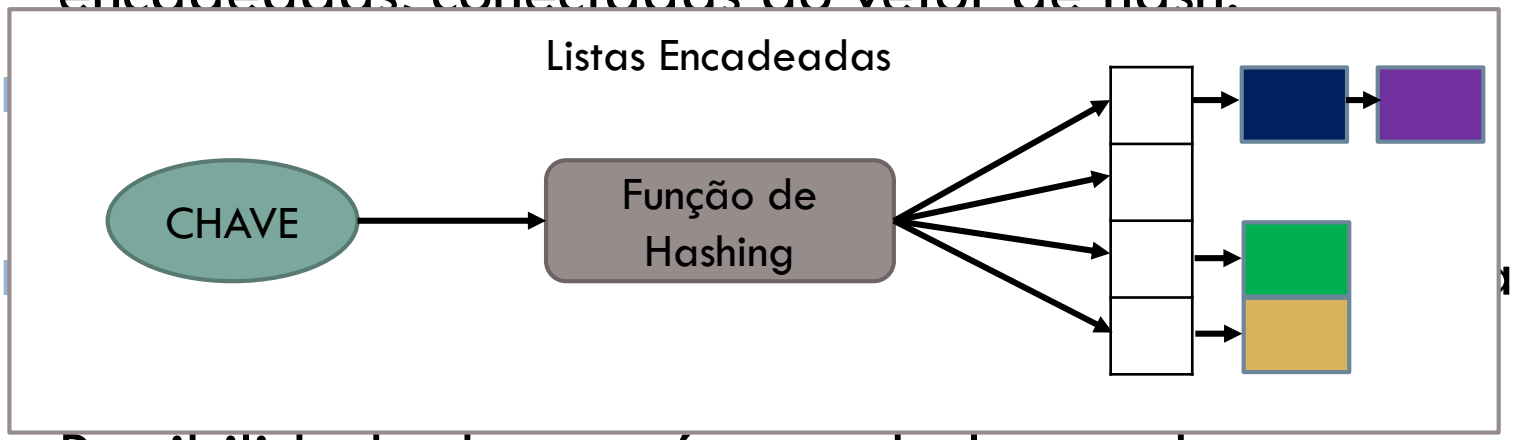
- **Colisões:** Correção por Encadeamento:
  - Os dados estão armazenados em estruturas encadeadas, conectadas ao vetor de hash.
  - Os dados conflitantes são encadeados dentro do mesmo índice de hash.

Função de Hashing

- **Colisões:** Correção por Encadeamento:
  - Os dados estão armazenados em estruturas encadeadas, conectadas ao vetor de hash.
  - Os dados conflitantes são encadeados dentro do mesmo índice de hash.
  - Se a lista de colisão for pequena, pode ser utilizada uma lista encadeada.
  - Possibilidade de usar árvores balanceadas para proteção de ataques DOS.

Função de Hashing

- **Colisões:** Correção por Encadeamento:
  - Os dados estão armazenados em estruturas encadeadas, conectadas ao vetor de hash.



  - Possibilidade de usar árvores balanceadas para proteção de ataques DOS.

Função Hash Criptográfica

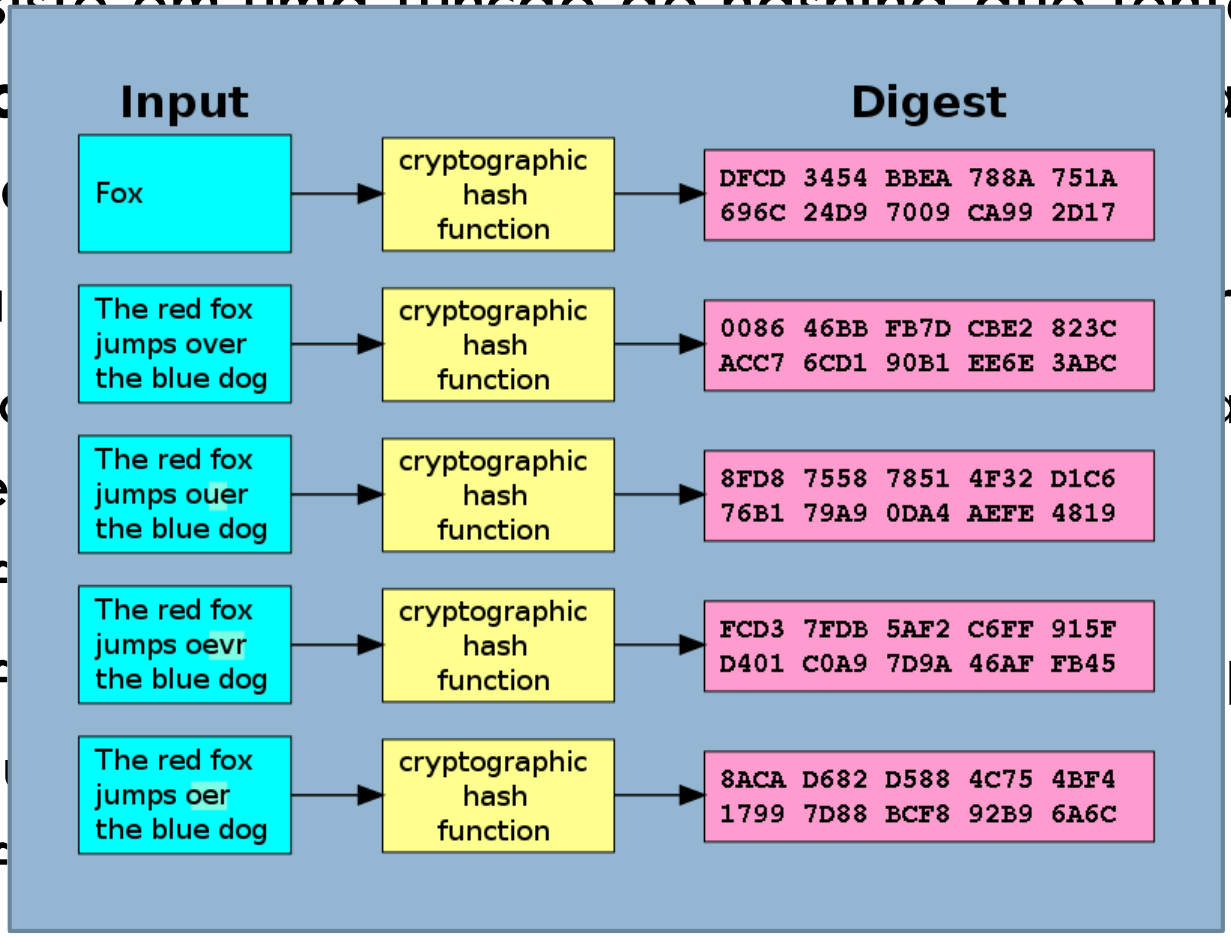
- Consiste em uma função de hashing que tenta **impossibilitar** a recriação do valor de entrada usando apenas o valor do hash.

Função Hash Criptográfica

- Consiste em uma função de hashing que tenta **impossibilitar** a recriação do valor de entrada usando apenas o valor do hash.
- Para as funções hash criptográficas deve ser:
  - Fácil de calcular computacionalmente para qualquer mensagem.
  - Difícil de gerar a mensagem a partir do hash.
  - Difícil de modificar a mensagem sem alterar o hash equivalente.
  - Difícil de gerar colisão.

Função Hash Criptográfica

- Consiste em uma função de hashing que tenta **impo** ada usan
- Para :
  - Fác alquer me
  - Dif
  - Dif hash eq
  - Dif



Função Hash Criptográfica

- Consiste em uma função de hashing que tenta **impo** ada usan
- Para :
  - Fá alquer me
  - Dif
  - Dif hash eq
  - Dif

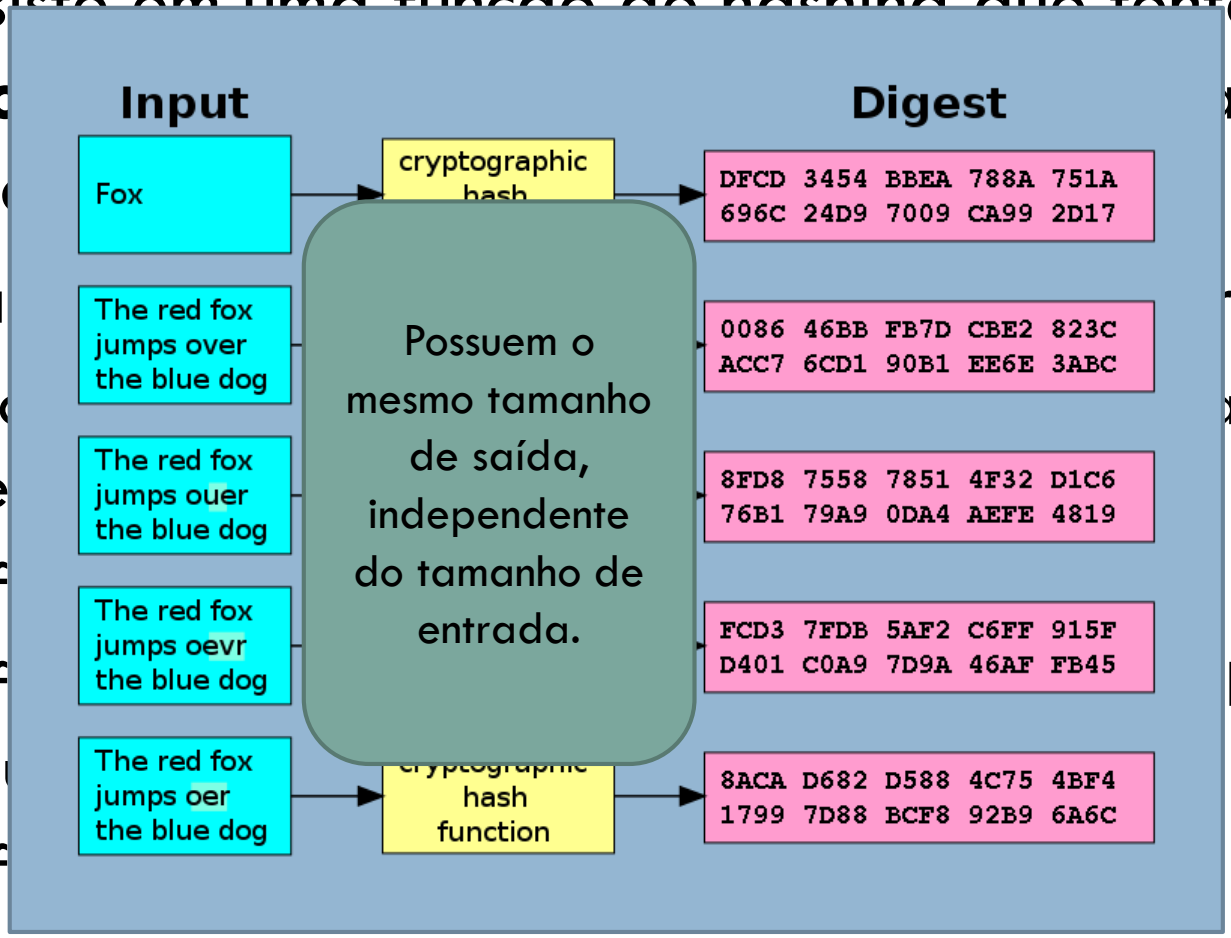


Tabela Hash

- Vantagens:
  - Alta eficiência na operação de busca.
  - Tempo de busca praticamente independente da quantidade de chaves armazenadas.
- Desvantagens:
  - Alto custo para acessar chaves de forma ordenada.
  - Pior caso quando a função de hashing gera muitas colisões (índices iguais para chaves diferentes).

Tabela Hash

- Aplicações:
  - Busca de elementos em base de dados.
  - Criptografia: MD4, MD5, família SHA, etc.
  - Tabela de símbolos dos compiladores.
  - Armazenamento de senhas (guardar o hash).
  - Verificação de integridade (checksum).