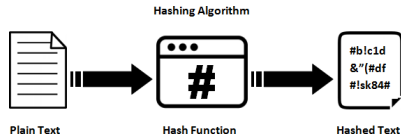


Algoritmo e Estrutura de Dados III

Tabelas Hash

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia



Roteiro

- 1 Objetivos
- 2 Motivação
- 3 Introdução
- 4 Definição
- 5 Projeto de Funções Hash
- 6 Tratamento de Colisões
- 7 Conclusões
- 8 Referências bibliográficas
- 9 Material Complementar

Table of Contents

- 1 Objetivos
- 2 Motivação
- 3 Introdução
- 4 Definição
- 5 Projeto de Funções Hash
- 6 Tratamento de Colisões
- 7 Conclusões
- 8 Referências bibliográficas
- 9 Material Complementar

Objetivos

Esta aula tem como objetivos:

- 1 Formalizar a definição de Tabelas Hash,
- 2 Apresentar os conceitos de Função Hash, Hashing Universal e tratamento de colisões.

Table of Contents

- 1 Objetivos
- 2 Motivação**
- 3 Introdução
- 4 Definição
- 5 Projeto de Funções Hash
- 6 Tratamento de Colisões
- 7 Conclusões
- 8 Referências bibliográficas
- 9 Material Complementar

Motivação

- Os métodos de pesquisa vistos até agora buscam informações armazenadas com base na comparação de suas chaves.
- Esses métodos utilizam listas ou árvores para organizar as informações.
 - ▶ Os algoritmos mais eficientes de busca, mostrados até o momento, demandam esforço computacional $O(\log n)$.
- Porém, em nenhuma dessas estruturas se obtém o acesso direto a alguma informação, a partir do conhecimento de sua chave.

Aplicações

- Suas aplicações incluem banco de dados, implementações das tabelas de símbolos dos compiladores, na programação de jogos para acessar rapidamente a posição para qual o personagem irá se mover e na implementação de um dicionário.
- Em redes de computadores, NAT, *Network Address Translation*, também conhecido como masquerading, é uma técnica que consiste em reescrever os endereços IP utilizando-se de uma tabela hash.

Table of Contents

- 1 Objetivos
- 2 Motivação
- 3 Introdução**
- 4 Definição
- 5 Projeto de Funções Hash
- 6 Tratamento de Colisões
- 7 Conclusões
- 8 Referências bibliográficas
- 9 Material Complementar

Introdução

- Uma Tabela Hash, também conhecida como tabela de dispersão ou tabela de espalhamento, é uma estrutura de dados especial, que associa chaves e valores.
- Seu objetivo é a partir de uma chave simples, fazer uma busca rápida e obter o valor desejado.

Introdução

- A Tabela Hash leva em conta o valor absoluto de cada chave, interpretado como um valor numérico.
- Através da aplicação de uma função conveniente, a chave é transformada em um endereço de uma tabela



Table of Contents

- 1 Objetivos
- 2 Motivação
- 3 Introdução
- 4 Definição**
- 5 Projeto de Funções Hash
- 6 Tratamento de Colisões
- 7 Conclusões
- 8 Referências bibliográficas
- 9 Material Complementar

Introdução

Princípio de Funcionamento

- Suponha que existam n elementos a serem armazenados em uma tabela T , sequencial e de tamanho m .
- As posições da tabela se situam no intervalo $[0, m - 1]$.
- Isto é, a tabela é particionada em m compartimentos, cada uma corresponde a um endereço, podendo armazenar m elementos.
- A forma mais simples de implementar uma **tabela hash** é utilizando **endereçamento direto**.

Endereçamento Direto

- Cada elemento é identificado por uma chave em \mathbb{N} ;
- Quando o universo de chaves $U = 0, 1, \dots, m - 1$ é pequeno, a tabela pode ser implementada diretamente como um vetor.
- Cada posição representa uma chave de U e armazena um elemento x ou um ponteiro para x .

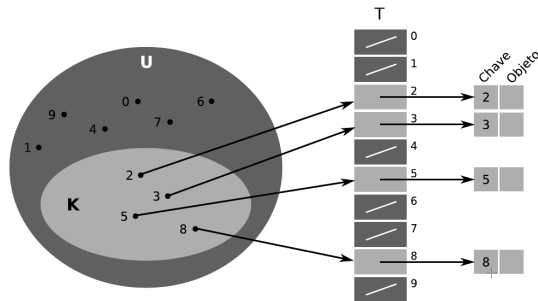


Figura: Endereçamento direto. Fonte: Cormen et al. (2012).

Endereçamento Direto

Operações

As operações disponíveis em Tabelas Hash são:

- Inserir-Endereçamento-Direto(T, x): inserir elemento x na tabela hash T ;
- Remover-Endereçamento-Direto(T, x): remover elemento x da tabela hash T ;
- Buscar-Endereçamento-Direto(T, k): retornar elemento com chave k na tabela hash T , quando $k \in T$.

Endereçamento Direto

Operações

Algoritmo 1: Inserir-Endereçamento-Direto

Entrada: Tabela Hash T , Elemento x

- 1 **início**
 - 2 $T[key[x]] \leftarrow x$
-

Algoritmo 2: Remover-Endereçamento-Direto

Entrada: Tabela Hash T , Elemento x

- 1 **início**
 - 2 $T[key[x]] \leftarrow NULL$
-

Algoritmo 3: Buscar-Endereçamento-Direto

Entrada: Tabela Hash T , Chave k

- 1 **início**
 - 2 **retorna** $T[k]$
-

Endereçamento Direto

Operações

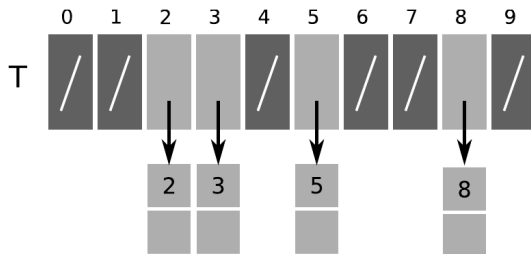


Figura: Endereçamento direto. Fonte: Cormen et al. (2012).

Endereçamento Direto

Análise

- As operações possuem complexidade temporal $O(1)$ (constante) no pior caso.
- No entanto, se o universo de chaves for muito grande ou esperso, torna-se inviável sua utilização.
- A solução é utilizar uma função hash h para mapear um elemento x à sua chave $k = h(x)$.

Tabelas Hash

- A tabela é implementada como um vetor de m posições em que cada posição armazena um subconjunto de U .

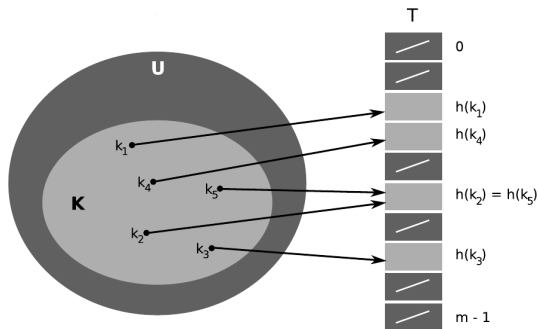


Figura: Função hash. Fonte: Cormen et al. (2012).

Tabelas Hash

Vantagens x Desvantagens

Vantagem

Se K é o conjunto das chaves armazenadas, a tabela requer espaço $\Theta(|K|)$ ao invés de $\Theta(|U|)$.

Desvantagens

Colisão: duas chaves podem ser mapeadas para a mesma posição! A busca na tabela requer $O(1)$ no caso médio, mas $O(n)$ no pior caso.

Tabelas Hash

Possíveis Problemas

Problema

O número de colisões não pode ser muito grande.

- Esse número depende de como a função de hash h espalha os elementos.
- **Solução:** escolher uma função h determinística, mas com saída aparentemente aleatória.

Table of Contents

- 1 Objetivos
- 2 Motivação
- 3 Introdução
- 4 Definição
- 5 Projeto de Funções Hash**
- 6 Tratamento de Colisões
- 7 Conclusões
- 8 Referências bibliográficas
- 9 Material Complementar

Projeto de Funções Hash

Problema

Funções de hash verdadeiramente aleatórias não podem ser implementadas com tempo constante.

- Precisamos de uma função que pareça aleatória, ou seja, mapeie um elemento para cada posição com probabilidade próxima de $\frac{1}{m}$.
- Isso depende da distribuição das entradas.

Projeto de Funções Hash

Exemplos:

- Se as entradas k são valores reais uniformemente distribuídos no intervalo $[0, 1)$, podemos usar $h(k) = \lfloor km \rfloor$;
- Se as entradas são identificadores de um programa, h deve diminuir a probabilidade de elementos parecidos como “pt” e “pts” colidirem.

Projeto de Funções Hash

Método da Divisão

Método da Divisão

A função h é definida como $h(k) = k \bmod m$.

A qualidade depende da escolha de m :

- Se $m = 2^p$, a função escolhe os bits menos significativos de k ;
- Se m é um número primo não muito próximo de uma potência de 2, h considera mais bits de k .

Projeto de Funções Hash

Método da Divisão

Exemplo

- Para armazenar $n = 2000$ elementos em uma tabela de hash, onde uma busca sem sucesso pode visitar até 3 elementos, m deve ser primo e próximo de $\frac{2000}{3}$.
 - ▶ Um bom valor para m é 701.
 - ▶ Um valor ruim é 500.

Projeto de Funções Hash

Método da Divisão

Exemplo

- Para armazenar $n = 2000$ elementos em uma tabela de hash , onde uma busca sem sucesso pode visitar até 3 elementos, m deve ser primo e próximo de $\frac{2000}{3}$.
 - ▶ Um bom valor para m é 701.
 - ▶ Um valor ruim é 500.

Projeto de Funções Hash

Método da Divisão

- Determine os resultados de $h(k) = k \bmod m$, utilizando os dois valores de $m = 500$ e $m = 701$, para as chaves

$$k = \{501, 601, 1000, 1500, 1101, 1501\}$$

Utilizando a função hash

$$h(k) = k \bmod m$$

Percurso

Busca em Profundidade

$$h(k) = k \bmod m$$

$$m = 701.$$

$$h(501) = 501$$

$$h(601) = 601$$

$$h(1000) = 299$$

$$h(1101) = 400$$

$$h(1500) = 98$$

$$h(1501) = 99$$

$$m = 500.$$

$$h(501) = 1$$

$$h(601) = 101$$

$$h(1000) = 0$$

$$h(1101) = 101$$

$$h(1500) = 0$$

$$h(1501) = 1$$

Projeto de Funções Hash

Método da Multiplicação

Método da Multiplicação

A função h é definida como $h(k) = \lfloor m(kc \bmod 1) \rfloor$.

Função h

$kc \bmod 1$ significa a parte fracionária de kc , que é $kc - \lfloor kc \rfloor$. Portanto,
 $h(k) = \lfloor m(kc - \lfloor kc \rfloor) \rfloor$.

- Neste caso, o valor de m não é crítico, mas a escolha de c depende das características da entrada.
- Mas, a escolha da constante c é importante.

Projeto de Funções Hash

Método da Multiplicação

Exemplo:

- Considere os valores $m = 1000$ e $c = 0,5$.
- Determine os resultados de $h(k)$ para

$$k = \{1100, 1101, 1102, 1103, 1104, 1105\}$$

Projeto de Funções Hash

Método da Multiplicação

$m = 1000$ e $c = 0,5$.

$$h(1100) = 0$$

$$h(1101) = 500$$

$$h(1102) = 0$$

$$h(1103) = 500$$

$$h(1104) = 0$$

$$h(1105) = 500$$

$$h(1106) = 0$$

$$h(1107) = 500$$

Colisão!

Projeto de Funções Hash

Método da Multiplicação

$m = 1000$ e $c = 0,5$.

$$h(1100) = 0$$

$$h(1101) = 500$$

$$h(1102) = 0$$

$$h(1103) = 500$$

$$h(1104) = 0$$

$$h(1105) = 500$$

$$h(1106) = 0$$

$$h(1107) = 500$$

Colisão!

Projeto de Funções Hash

Método da Multiplicação

- Alguns valores de c são melhores do que outros.
- Em especial, $c = \frac{\sqrt{5}-1}{2} \approx 0,6180339887\dots$. Ou seja, a razão áurea!
- Knuth (1997) mostrou por meio de experimentos que o uso da razão áurea apresenta bons resultados.

Projeto de Funções Hash

Mapeamento Universal

Problema

Heurísticas são determinísticas e podem ser manipuladas de forma indesejada. Um adversário pode escolher as chaves de entrada para que todas colidam.

- Uma estratégia que tenta minimizar o problema de colisões é o hashing universal.
- A idéia é escolher aleatoriamente (em tempo de execução) uma função hash a partir de um conjunto de funções cuidadosamente desenhado.

Projeto de Funções Hash

Mapeamento Universal

Definição

Uma classe \mathcal{H} de funções de hash é universal se o número de funções $h \in \mathcal{H}$ em que $h(k_i) = h(k_j)$ é $\frac{|\mathcal{H}|}{m}$.

Mapeamento Universal

Análise

- A colisão entre duas chaves k_i e k_j ocorre com probabilidade $\frac{1}{m}$, a mesma probabilidade de colisão se $h(k_i)$ e $h(k_j)$ fossem selecionados aleatoriamente em \mathcal{H} .
- O limite superior para o número esperado de colisões para cada chave k , baseando-se na escolha da função de hash é:

$$\sum_{i \in T, i \neq k} \frac{1}{m}.$$

- Se $k \notin T$, a lista $n_h(k)$ tem tamanho esperado $\frac{n}{m} = \alpha$;
- Se $k \in T$, a lista $n_h(k)$ tem tamanho esperado $1 + \frac{n-1}{m} < 1 + \alpha$.

Mapeamento Universal

Análise

Teorema

Com mapeamento universal e encadeamento, o tamanho esperado de cada lista n_i é no máximo $1 + \alpha$.

Corolário

Com mapeamento universal e encadeamento, uma tabela com m posições realiza qualquer sequência de n operações contendo $O(m)$ inserções em tempo esperado $\Theta(n)$.

Complexidade: as operações tomam tempo constante em média.

Mapeamento Universal

Projeto

Seja p um número primo tal que $p > |U|$. Denota-se por $\mathbb{Z}_p = 0, 1, \dots, p-1$ e $\mathbb{Z}_p^* = \mathbb{Z}_p - 0$.

Teorema

A classe $\mathcal{H}_{p,m}$ de funções $h_{a,b} : a \in \mathbb{Z}_p^*, b \in \mathbb{Z}_p$ é universal para $h_{a,b}(k) = ((a_k + b) \bmod p) \bmod m$.

Exemplo: Se $p = 17$ e $m = 16$, temos $h_{3,4}(8) = 5$. A classe tem $p(p-1)$ funções distintas. A universalidade segue as propriedades da redução módulo o número primo p .

Table of Contents

- 1 Objetivos
- 2 Motivação
- 3 Introdução
- 4 Definição
- 5 Projeto de Funções Hash
- 6 Tratamento de Colisões**
- 7 Conclusões
- 8 Referências bibliográficas
- 9 Material Complementar

Tratamento de Colisões

Problema

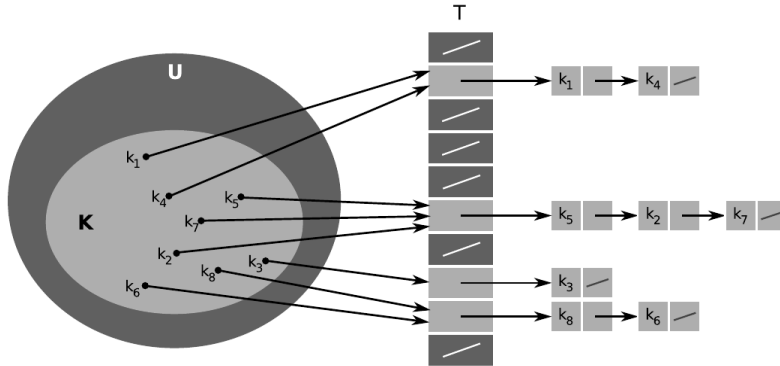
Como $|U| > m$, a escolha de h apenas minimiza o número de colisões.

- **Solução:** tratar as colisões restantes de forma algorítmica, aplicando:
 - ▶ **lista encadeada** ou
 - ▶ **endereçamento aberto.**

Tratamento de Colisão

Lista Encadeada

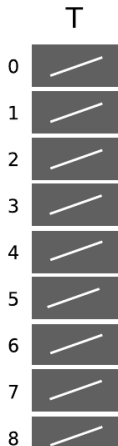
- Em uma tabela de hash encadeada, todos os elementos mapeados para uma mesma posição são armazenados em uma lista ligada.



Tratamento de Colisão

Lista Encadeada - Exemplo

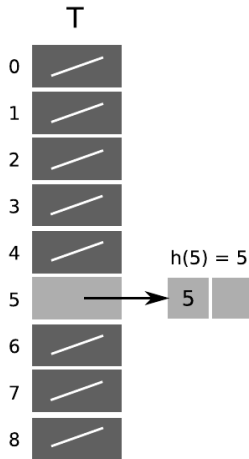
- Inserção das chaves 5, 28, 19, 15, 20, 33 em uma tabela com 9 posições, utilizando $h(k) = k \bmod 9$.



Tratamento de Colisão

Lista Encadeada - Exemplo

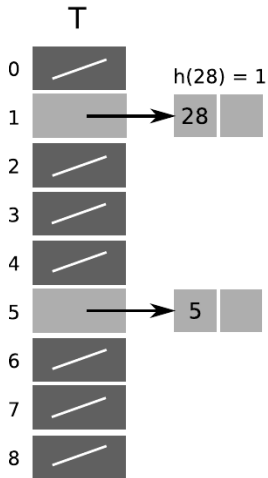
- Inserção da chave 5 para $h(k) = k \bmod 9$.



Tratamento de Colisão

Lista Encadeada - Exemplo

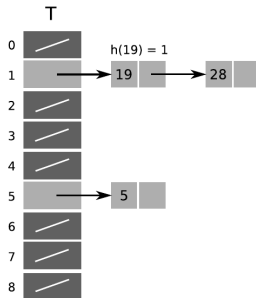
- Inserção da chave 28 para $h(k) = k \bmod 9$.



Tratamento de Colisão

Lista Encadeada - Exemplo

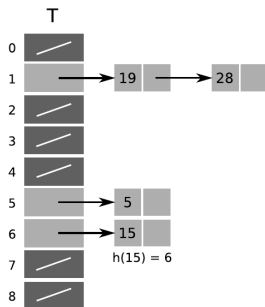
- Inserção da chave 19 para $h(k) = k \bmod 9$.



Tratamento de Colisão

Lista Encadeada - Exemplo

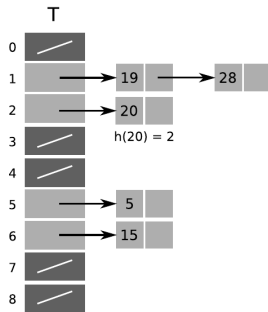
- Inserção da chave 15 para $h(k) = k \bmod 9$.



Tratamento de Colisão

Lista Encadeada - Exemplo

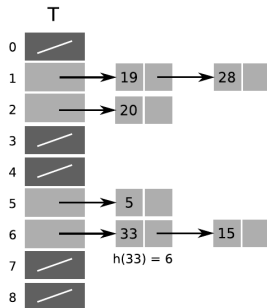
- Inserção da chave 20 para $h(k) = k \bmod 9$.



Tratamento de Colisão

Lista Encadeada - Exemplo

- Inserção da chave 33 para $h(k) = k \bmod 9$.



Lista Encadeada

Operações

As operações disponíveis em Tabelas Hash são:

- Inserir-Hash-Encadeado(T , x): inserir elemento x no início da lista $T[h(\text{key}(x))]$;
- Remover-Hash-Encadeado(T , x): remover elemento x da lista $T[h(\text{key}(x))]$;
- Buscar-Hash-Encadeado(T , k): retornar elemento com chave k na lista $T[h(k)]$.

Lista Encadeada

Operações

Algoritmo 4: Inserir-Hash-Encadeado

Entrada: Tabela Hash T , Elemento x

1 início

```
2 | Inserir  $x$  no início da lista  $T[h(key[x])]$ 
```

Algoritmo 5: Remover-Hash-Encadeado

Entrada: Tabela Hash T , Elemento x

1 início

```
2 | Remover x da lista  $T[h(key[x])]$ 
```

Algoritmo 6: Buscar-Hash-Encadeado

Entrada: Tabela Hash T , Chave k

1 início

```

2 | Buscar por um elemento com chave  $k$  na lista  $T[h(k)]$ 

```

Lista Encadeada

Análise

- Ao se trabalhar com listas encadeadas, é usual efetuar-se a inserção de uma nova chave x no final da lista correspondente ao endereço $h(x)$.
- A ideia é que a lista será percorrida de qualquer maneira, para assegurar que x não pertence à mesma.
- Mas, caso chaves repetidas sejam aceitas, essa condição pode ser relaxada, e a chave inserida no início da lista.

Lista Encadeada

Análise

- Inserção: Complexidade Temporal constante ($O(1)$);
- Remoção: Complexidade Temporal constante ($O(1)$) utilizando lista duplamente ligada.
- Busca sem sucesso: depende do comprimento de $T[h(k)]$;
- Busca com sucesso: depende do número de elementos antes de x em $T[h(key[x])]$;

Lista Encadeada

Análise

No pior caso, todos os elementos são mapeados para a mesma posição e a busca custa $\Theta(n)$ mais o cálculo de h .

Mapeamento uniforme simples

No caso médio, podemos assumir que um elemento pode ser mapeado para qualquer posição igualmente e que dois elementos são mapeados independentemente:

$$Pr\{h(k_i) = h(k_j)\} = \frac{1}{m}.$$

Lista Encadeada

Análise

Definição

Em uma tabela de hash com m posições que armazena n elementos, o fator de carga α é definido como $\frac{n}{m}$.

Seja n_j o comprimento da lista $T[j]$. O valor esperado de n_j é α . Assume-se que para calcular $k = h(x)$ necessite de tempo constante $O(1)$.

Lista Encadeada

Análise

- **Busca Sem Sucesso:** examina-se toda a lista $T[k]$ com tamanho esperado α . A complexidade é $O(1 + \alpha)$.
- **Busca Com Sucesso:** examinam-se os elementos anteriores a x e o próprio x . Na média, examinam-se

$$1 + \frac{1}{n} \sum_{i=1}^n \sum_{j=i+1}^n \frac{1}{m} = O\left(1 + \frac{1}{n} \frac{1}{m} n^2\right) = O(1 + \alpha).$$

Lista Encadeada

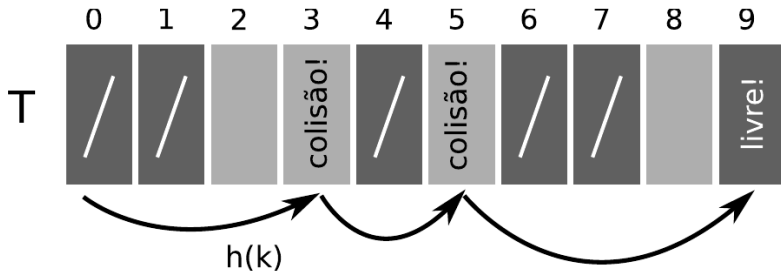
Análise

- Se o número de posições é proporcional ao número de elementos, ou $n = O(m)$, o fator de carga é $\alpha = O(1)$ e a busca toma tempo constante no caso médio;
- Todas as operações tomam tempo constante em média.

Tratamento de Colisão

Endereçamento Aberto

- Em uma tabela de hash com endereçamento aberto, todos os elementos são armazenados na tabela propriamente dita.
- O espaço gasto com encadeamento é economizado e a colisão é tratada com a busca de uma nova posição para inserção.
- Naturalmente, o fator de carga não pode exceder o valor 1.



Endereçamento Aberto

Operações

Durante a inserção, uma sequência de posições é testada até que uma posição livre seja encontrada. A função de hash é modificada para receber um argumento que armazena o número do teste.

Algoritmo 7: Inserir-Hash-Aberto

Entrada: Tabela Hash T , Elemento x

```
1 início
2    $i \leftarrow 0$ 
3   repita
4      $j \leftarrow h(k, i)$ 
5     se  $(T[j] = \text{NULL})$  então
6        $T[j] \leftarrow k$ 
7       retorna  $j$ 
8     senão
9        $i \leftarrow i + 1$ 
10  até  $(i=m)$ ;
11  Imprima ("Erro: Overflow")
```

Endereçamento Aberto

Operações

O algoritmo de busca percorre a mesma sequência examinada pelo algoritmo de inserção quando k foi inserido.

Algoritmo 8: Buscar-Hash-Aberto

Entrada: Tabela Hash T , Chave k

```
1 início
2    $i \leftarrow 0$ 
3   repita
4      $j \leftarrow h(k, i)$ 
5     se  $(T[j] = k)$  então
6       retorna  $j$ 
7     senão
8        $i \leftarrow i + 1$ 
9   até  $(T[j] = \text{NULL} \text{ ou } i = m)$ ;
10  retorna NULL
```

Endereçamento Aberto

Operações

- A **remoção** de elementos é difícil.
- Pode-se utilizar um valor especial para marcar elementos removidos.
- Mas o custo da busca deixa de depender do fator de carga.

Endereçamento Aberto

Busca Linear

A função de hash deve produzir como sequência de teste uma permutação de $0, 1, \dots, m - 1$.

Busca linear

Seja h uma função de hash auxiliar. A função h é definida como $h(k, i) = (h'(k) + i) \bmod m$.

Busca Linear

Vantagens x Desvantagens

Vantagem

Fácil implementação.

Desvantagem

É suscetível a **agrupamento primário**. Ou seja, são construídas sequências longas de posições ocupadas, o que degrada o desempenho da busca.

Busca Linear

Exemplo

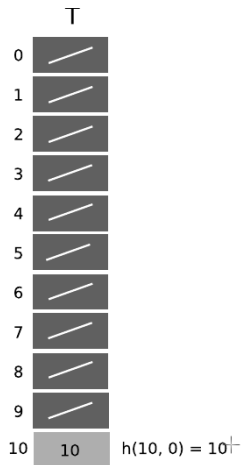
Inserção das chaves $\{10, 22, 31, 4, 15, 28, 59\}$ em uma tabela de tamanho 11 com teste linear e função $h(k, i) = (k + i) \bmod 11$.

T	
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Busca Linear

Exemplo

Inserção da chave 10 para $h(k, i) = (k + i) \bmod 11$.



Busca Linear

Exemplo

Inserção da chave 22 para $h(k, i) = (k + i) \bmod 11$.

T

0	22	$h(22, 0) = 0$
1		
2		
3		
4		
5		
6		
7		
8		
9		
10	10	$h(10, 0) = 10$

Busca Linear

Exemplo

Inserção da chave 31 para $h(k, i) = (k + i) \bmod 11$.

T	
0	22
1	
2	
3	
4	
5	
6	
7	
8	
9	31
10	10

$h(31, 0) = 9$

Busca Linear

Exemplo

Inserção da chave 4 para $h(k, i) = (k + i) \bmod 11$.

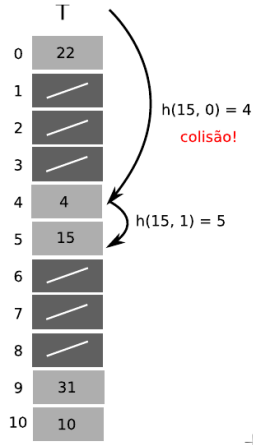
T

0	22	
1		
2		
3		
4	4	$h(4, 0) = 4$
5		
6		
7		
8		
9	31	
10	10	

Busca Linear

Exemplo

Inserção da chave 15 para $h(k, i) = (k + i) \bmod 11$.



+

Busca Linear

Exemplo

Inserção da chave 28 para $h(k, i) = (k + i) \bmod 11$.

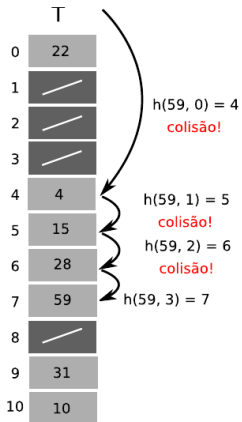
T

0	22	
1		
2		
3		
4	4	
5	15	
6	28	$h(28, 0) = 6$
7		
8		
9	31	
10	10	

Busca Linear

Exemplo

Inserção da chave 59 para $h(k, i) = (k + i) \bmod 11$.



Endereçamento Aberto

Busca Quadrática

Busca Quadrática

Seja h uma função de hash auxiliar, c_1 e c_2 constantes não-nulas. A função h é definida como $h(k, i) = (h'(k) + c_1i + c_2i^2) \bmod m$.

Busca Quadrática

Vantagens x Desvantagens

Vantagem

É imune a agrupamento primário.

Desvantagem

É suscetível a **agrupamento secundário**. Ou seja, as sequências de teste são idênticas para duas chaves k_i e k_j tais que $h'(k_i) = h1(k_j)$.

Busca Quadrática

Exemplo

Inserção das chaves $\{10, 22, 31, 4, 15, 28, 59\}$ em uma tabela de tamanho 11 com teste quadrático e $h(k, i) = (k + i + 3i^2) \bmod 11$.

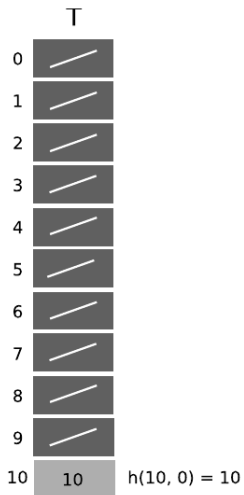
T

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Busca Quadrática

Exemplo

Inserção da chave 10 para $h(k, i) = (k + i + 3i^2) \bmod 11$.



Busca Quadrática

Exemplo

Inserção da chave 22 para $h(k, i) = (k + i + 3i^2) \bmod 11$.

T	
0	22 $h(22, 0) = 0$
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	10 $h(10, 0) = 10$

Busca Quadrática

Exemplo

Inserção da chave 31 para $h(k, i) = (k + i + 3i^2) \bmod 11$.

T

0	22
1	/
2	/
3	/
4	/
5	/
6	/
7	/
8	/
9	31
10	10

$h(31, 0) = 9$

Busca Quadrática

Exemplo

Inserção da chave 4 para $h(k, i) = (k + i + 3i^2) \bmod 11$.

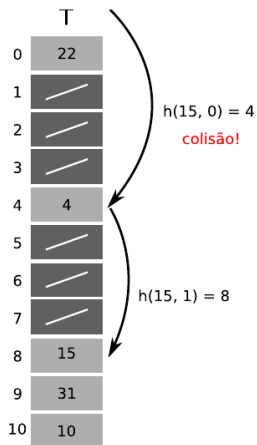
T	
0	22
1	
2	
3	
4	4
5	
6	
7	
8	
9	31
10	10

$h(4, 0) = 4$

Busca Quadrática

Exemplo

Inserção da chave 15 para $h(k, i) = (k + i + 3i^2) \bmod 11$.



Busca Quadrática

Exemplo

Inserção da chave 28 para $h(k, i) = (k + i + 3i^2) \bmod 11$.

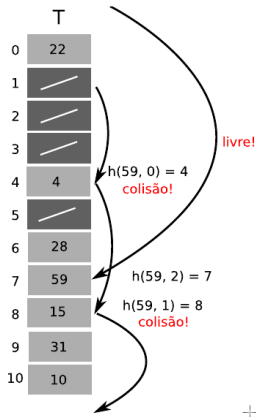
	+	T
0		22
1		/
2		/
3		/
4		4
5		/
6		28
7		/
8		15
9		31
10		10

$h(28, 0) = 6$

Busca Quadrática

Exemplo

Inserção da chave 59 para $h(k, i) = (k + i + 3i^2) \bmod 11$.



Endereçamento Aberto

Hash Duplo

Duplo Mapeamento

No **hash duplo**, a função h é definida como $h(k, i) = (h_1(k) + ih_2(k)) \bmod m$, para $1 \leq i \leq m - 1$.

- h_1 e h_2 são funções de hash auxiliares.
- O projeto e a implementação são mais difíceis que os métodos apresentados anteriormente.
- No entanto, não causa agrupamento do tipo produzido pelo teste linear ou pelo teste quadrático, e apresenta melhor desempenho na média.

Endereçamento Aberto

Hash Duplo

Duplo Mapeamento

No **hash duplo**, a função h é definida como $h(k, i) = (h_1(k) + ih_2(k)) \bmod m$, para $1 \leq i \leq m - 1$.

- Para varrer toda a tabela, é necessário que $h_2()$ e m sejam primos entre si, ou seja, o único divisor comum a eles é o número 1.
- Por exemplo, se m for potência de 2, basta definir $h_2()$ de forma a produzir números ímpares.
- Ou então, mais simples ainda, basta definir m como um número primo e projetar $h_2()$ de forma que ele sempre retorne um inteiro positivo menor que m .

Hash Duplo

Vantagens x Desvantagens

Vantagens

- O mapeamento duplo considera $\Theta(m^2)$ sequências de teste, já que cada par $(h_1(k), h_2(k))$ produz uma nova sequência.
- Teste linear ou quadrático apenas consideram $\Theta(m)$ sequencias.

Desvantagem

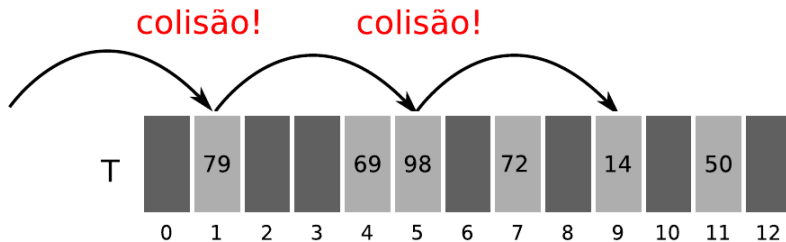
Projeto e implementação mais difícil.

Hash Duplo

Exemplo

Exemplo:

- Sejam $h_1(k) = k \bmod 13$ e $h_2(k) = 1 + (k \bmod 11)$.
- Então $h(14, 0) = 1$, $h(14, 1) = 5$ e $h(14, 2) = 9$.
 - ▶ $h(14, 0) = (h_1(14) + 0h_2(14)) \bmod 13 = 1$.
 - ▶ $h(14, 1) = (h_1(14) + 1h_2(14)) \bmod 13 = 1 + 1 \times 4 = 5$.
 - ▶ $h(14, 2) = (h_1(14) + 2h_2(14)) \bmod 13 = 1 + 2 \times 4 = 9$.



Hash Duplo

Análise

- O algoritmo de **pesquisa** (ou busca) percorre a mesma sequência de posições examinada pelo algoritmo de **inserção** quando a chave k foi inserida.
- Após a **remoção**, a posição não pode ser deixada como uma célula vazia, pois pode interferir nas buscas.
- A posição deve ser marcada de alguma maneira (com uma variável booleana, por exemplo) para que na busca possa-se saber que havia algo lá.

Endereçamento Aberto

Análise

Complexidade

- Considerando um mapeamento uniforme, o número médio de comparações em uma busca sem sucesso e na inserção é limitado por:

$$\sum_{i=1}^{\infty} \alpha^{i-1} = \frac{1}{1 - \alpha} = O(1).$$

sendo $\alpha = n/m$ o fator de carga da tabela.

- O aspecto negativo está relacionado com o pior caso, que é $O(n)$, se a função hash **não** conseguir espalhar os registros de forma razoável pelas entradas da tabela.

Endereçamento Aberto

Exercício

Escreva pseudocódigo para o algoritmo **Remover-Hash-Aberto** e modifique os algoritmos **Inserir-Hash-Aberto** e **Buscar-Hash-Aberto** de forma a levar em conta que elementos removidos da tabela são marcados com o valor especial *Deleted*.

Table of Contents

- 1 Objetivos
- 2 Motivação
- 3 Introdução
- 4 Definição
- 5 Projeto de Funções Hash
- 6 Tratamento de Colisões
- 7 Conclusões**
- 8 Referências bibliográficas
- 9 Material Complementar

Conclusões

- Considerando um mapeamento uniforme, cada operação toma tempo constante no caso médio.
 - ▶ Mas é raro conhecer a distribuição de probabilidade segundo a qual as chaves são obtidas.
- Na prática, podemos usar heurísticas de fácil implementação para criar uma função hash que provavelmente terá um bom desempenho.
- Para resolução de colisões, encadeamento é o método mais simples, mas gasta mais espaço.
- Endereçamento aberto tem implementação mais difícil ou que pode ser suscetível a efeitos de agrupamento.


Conclusões

- Vantagens:
 - ▶ Simplicidade de implementação.
 - ▶ Considerando K o conjunto de chaves armazenadas, a tabela requer espaço $\Theta(|K|)$ ao invés de $\Theta(|U|)$.
 - ▶ A busca na tabela requer $O(1)$ no caso médio.
- Desvantagens:
 - ▶ Colisão: Efeito que acontece quando duas chaves são mapeadas para a mesma posição na tabela.
 - ▶ A busca na tabela requer $O(|k|)$ no pior caso.

Table of Contents

- 1 Objetivos
- 2 Motivação
- 3 Introdução
- 4 Definição
- 5 Projeto de Funções Hash
- 6 Tratamento de Colisões
- 7 Conclusões
- 8 Referências bibliográficas**
- 9 Material Complementar

Referências bibliográficas

 CORMEN, T. H. et al. *Algoritmos: Teoria e Prática*. 3. ed. São Paulo: Campus, 2012. ISBN 978-0-262-03384-8.


 KNUTH, D. E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Third. Reading, Mass.: [s.n.], 1997. ISBN 0201896834 9780201896831.

Table of Contents

- 1 Objetivos
- 2 Motivação
- 3 Introdução
- 4 Definição
- 5 Projeto de Funções Hash
- 6 Tratamento de Colisões
- 7 Conclusões
- 8 Referências bibliográficas
- 9 Material Complementar**

Material Complementar

- Material Wikibooks
 - ▶ https://pt.wikibooks.org/wiki/Algoritmos/Estruturas_de_dados/Tabela_Hash
- Animação Método da Divisão - Tratamento de Colisão por lista encadeada.
 - ▶ <https://www.cs.usfca.edu/~galles/visualization/OpenHash.html>
- Animação Método da Divisão - Tratamento de Colisão por endereçamento aberto.
 - ▶ <https://www.cs.usfca.edu/~galles/visualization/ClosedHash.html>

Material Complementar

Youtube

- Estrutura de Dados Descomplicada - Tabelas Hash
- Prof. Marcos Kutova
- Estruturas de Dados - Conceitos de Tabela Hash (UNIVESP)
- Estruturas de Dados - Tabela Hash (implementação) (UNIVESP)

Algoritmo e Estrutura de Dados III

Tabelas Hash

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia

