

Tabelas de Dispersão

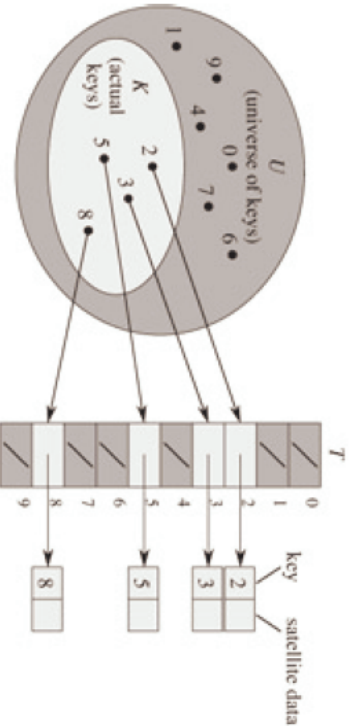
Algoritmos e Estruturas de Dados

Cátia Vaz

1

Tabelas de endereçamento directo

- Endereçamento directo é usado quando o universo de chaves é pequeno e todas as chaves são distintas:
 - Cada posição k , contém (referência) o elemento com chave k . Caso esse elemento não exista, $T[k]=\text{null}$.



Operations take $O(1)$ time each:

```
DIRECT-ADDRESS-SEARCH( $T, k$ )  
  return  $T[k]$   
DIRECT-ADDRESS-INSERT( $T, x$ )  
   $T[key[x]] \leftarrow x$   
DIRECT-ADDRESS-DELETE( $T, x$ )  
   $T[key[x]] \leftarrow \text{NIL}$ 
```

Cátia Vaz

2



Tabelas de endereçamento directo: desvantagens

- Se o universo U é muito grande, armazenar uma tabela T de tamanho $|U|$ é impraticável.
- O conjunto K de chaves que de facto foi armazenado pode ser tão pequeno relativamente à dimensão do universo U que a maior parte do espaço alocado para T está a ser desperdiçado.
- Alternativa: **tabelas de dispersão**.
 - Embora procurar por um elemento numa tabela de dispersão possa demorar o mesmo do que procurar um elemento numa lista ligada ($\Theta(N)$ no pior caso), sob certas condições, o tempo expectável para procurar por um elemento numa tabela de dispersão é $O(1)$.

Cátia Vaz

3



Tabelas de Dispersão

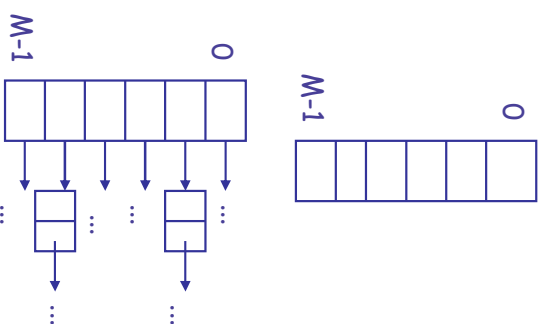
- As tabelas de dispersão (*hash tables*) são estruturas de dados que associam **chaves** a **valores**.
 - Cada chave é transformada pela função de *dispersão* num número que é utilizado para indexar num array para encontrar os valores associados aquela chave.
- As tabelas de dispersão utilizam:
 - tabela base de indexação;
 - funções de dispersão (*hash functions*);
 - algoritmos para resolução de colisões (se necessário).

Cátia Vaz

4

Tabela Base

- **Dispersão com índices livres:**
 - Quando o número de elementos pode ser estimado à partida em N
 - Tabela de M elementos ($M > N$)
- **Dispersão por separação em listas:**



Cátia Vaz

5

Função de Dispersão

- **Uma função de dispersão:**
 - Transforma a chave num inteiro $[0;M-1]$ (M tamanho do array);
 - Deve distribuir as chaves de forma uniforme e quase aleatória;
 - Deve ser rápida de calcular;
 - Diferentes funções devem ser usadas para diferentes tipos de dados.
- **Definição: Colisão** - ocorre quando a função de dispersão devolve o mesmo valor para chaves distintas
- **Definição: Função de dispersão ideal** - a probabilidade de ocorrer uma colisão para duas chaves distintas é $1/M$.

Cátia Vaz

6

Função de Dispersão

- A função de dispersão mais usada é a **divisão**:
 - Chaves pequenas (representáveis por uma palavra de memória):
 - tratar as chaves como inteiros (k)
 - $h(k) = k \% M$
 - usar um número primo para M .

Cátia Vaz

7

Função de Dispersão

Chave k	$k \% 1000$	$k \% 1024$	$k \% 1021$
32699	699	955	027
15114	114	778	820
41502	502	542	662
30444	444	748	835
81699	699	803	019
30651	651	955	021
23670	670	118	187
12219	219	955	988
75745	745	993	191

Cátia Vaz

8

Função de Dispersão

- Chaves longas (String)
 - usar cada caracter como um inteiro (valor da representação ASCII)
 - $h(k) = k \% M$
 - dar um peso a cada caracter correspondente à sua posição na cadeia
 - usar um número primo para o tamanho da tabela

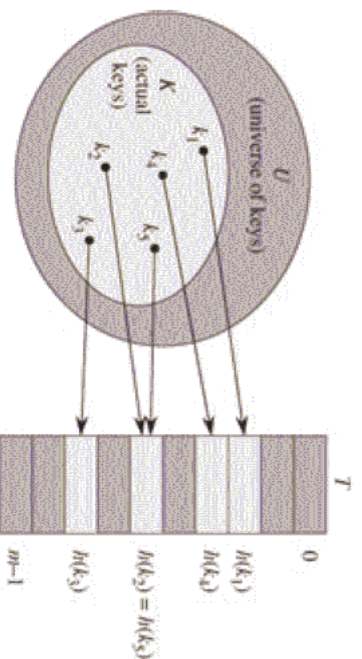
```
static int hash(String s, int M){
    int h=0; a=127;
    for(int i=0;i<s.length();i++){
        h=(a*h + s.charAt(i))%M;
    }
    return h;}
}
```

Cátia Vaz

9

Resolução de Colisões

- Para uma tabela de tamanho M , quantas inserções podem ser feitas até à primeira colisão?



- Os algoritmos de resolução de colisões depende do tipo de tabela base escolhido.

Cátia Vaz

10

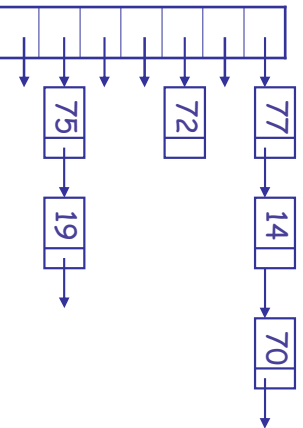
Resolução de colisões

- Algoritmos de resolução de colisões:
 - Dispersão com separação em listas, denominado encadeamento externo (*Separate Chaining*).
 - Dispersão com índices livres, denominado encadeamento interno (*Open Addressing*):
 - Procura Linear (*Linear Probing*);
 - Procura Quadrática (*Quadratic Probing*);
 - Dupla Dispersão (*Double Hashing*).

Cátia Vaz

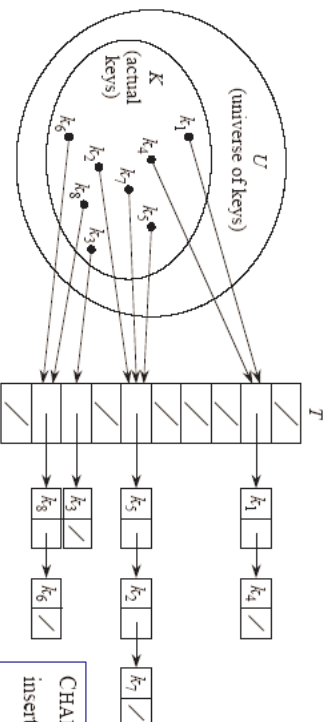
11

Resolução de Colisões: Encadeamento Externo (*Separate Chaining*)

- Cada posição da tabela tem uma referência para uma lista
 - Colisões são resolvidas adicionando o elemento ao início da lista
 - Remoções são resolvidas removendo o elemento da lista
- Ordem de inserção:
70, 72, 14, 77, 19, 75
- $$\text{hash}(k) = k \bmod 7$$
- 

12

Resolução de colisões: Encadeamento Externo (*Separate Chaining*)



hash function

$$h : U \rightarrow \{0, 1, \dots, m - 1\}$$

```
CHAINED-HASH-INSERT(T, x)
insert x at the head of list T[h(key[x])]

• running time is  $O(1)$ .

CHAINED-HASH-SEARCH(T, k)
search for an element with key k in list T[h(k)]

• running time proportional to the length of the list

CHAINED-HASH-DELETE(T, x)
delete x from the list T[h(key[x])]

• running time proportional to the length of the list
```

13

Resolução de colisões: Encadeamento Externo (*Separate Chaining*)

```
public interface HashFunction<E> {
    int hashCode(E e);
    boolean equals(E e1, E e2);}
```

```
/*Função Hash por omissão*/
public class HashDefaultFunction implements HashFunction<Object> {

    public static HashFunction<Object> getDefaultInstance(){
        if(instance == null) instance = new HashDefaultFunction();
        return (HashFunction<Object>)instance;
    }

    private static HashFunction<Object> instance = null;

    public int hashCode(Object o){ return o.hashCode(); }

    private HashDefaultFunction(){}
}
```

Resolução de colisões:

Encadeamento Externo (*Separate Chaining*)

```
public class ChainingHashTable<E> {
    static class Node<T>{
        T value;
        Node<T> next;
        Node(T v){ value = v;}
    }

    Node<E>[] v;
    int m;
    HashFunction<E> ho;

    public ChainingHashTable(int len){
        this(len,(HashFunction<E>)HashDefaultFunction.getDefaultInstance());}

    public ChainingHashTable(int len, HashFunction<E> ho){
        v = (Node<E>[]) new Node[len];
        m = len;
        this.ho = ho; } //(...)}
```

Cátia Vaz 15

Resolução de colisões:

Encadeamento Externo (*Separate Chaining*)

```
public class ChainingHashTable<E> {
    //(...)
    private final int index(E e){
        int h=ho.hashCode(e)% m;
        return (h<0)?h+m:h;}

    public final void insert(E e){
        int i = index(e);
        Node<E> n = new Node<E>(e);
        n.next = v[i];
        v[i] = n;}

    //(...)
}

public class ChainingHashTable<E> {
    //(...)
    public final E search(E key){
        int i = index(key);
        Node<E> curr = v[i];
        while(curr != null){
            if(ho.equals(key,curr.value))
                return curr.value;
            curr = curr.next;
        }
        return null;
    }

    //(...)
}
```

Cátia Vaz 16

Resolução de colisões:

Encadeamento Externo (*Separate Chaining*)

```
public class ChainingHashTable<E> {  
    //(...)   
    public final boolean delete(E e){  
        int i = index(e);  
        Node<E> curr = v[i];  
        Node<E> prev = null;  
        while(curr != null){  
            if(!ho.equals(curr.value,e)){  
                if(prev == null){ v[i] = v[i].next;}  
                else{ prev.next = curr.next; }  
                return true;  
            }  
            prev = curr; curr = curr.next;  
        }  
        return false;  
    }  
    //(...)}  
}
```

Cátia Vaz 17

Encadeamento Externo (*Separate Chaining*): análise

- **Pior Caso do Encadeamento Externo:** todas as N chaves têm o mesmo hash, criando uma lista de comprimento N . $\Theta(N)$.
- **Caso Médio.** A performance da dispersão depende de como a função de dispersão distribui as chaves a serem armazenadas pelas M posições.
 - Para o análise do caso médio assume-se o princípio de **dispersão uniforme simples**. O algoritmo de procura é:
 - $\Theta(1 + N/M)$.
 - se $N=O(M)$ então $O(1 + N/M)=O(1)$