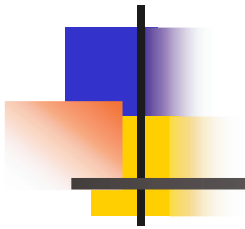
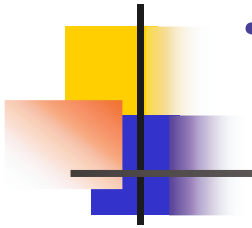


Tabelas de Dispersão

Algoritmos e Estuturas de Dados

Inverno 2006

Cátia Vaz



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).

Tabelas de Dispersão - Exemplo

Exemplo com função de dispersão **`hash(x)=x mod 10`** e inserção na cauda:

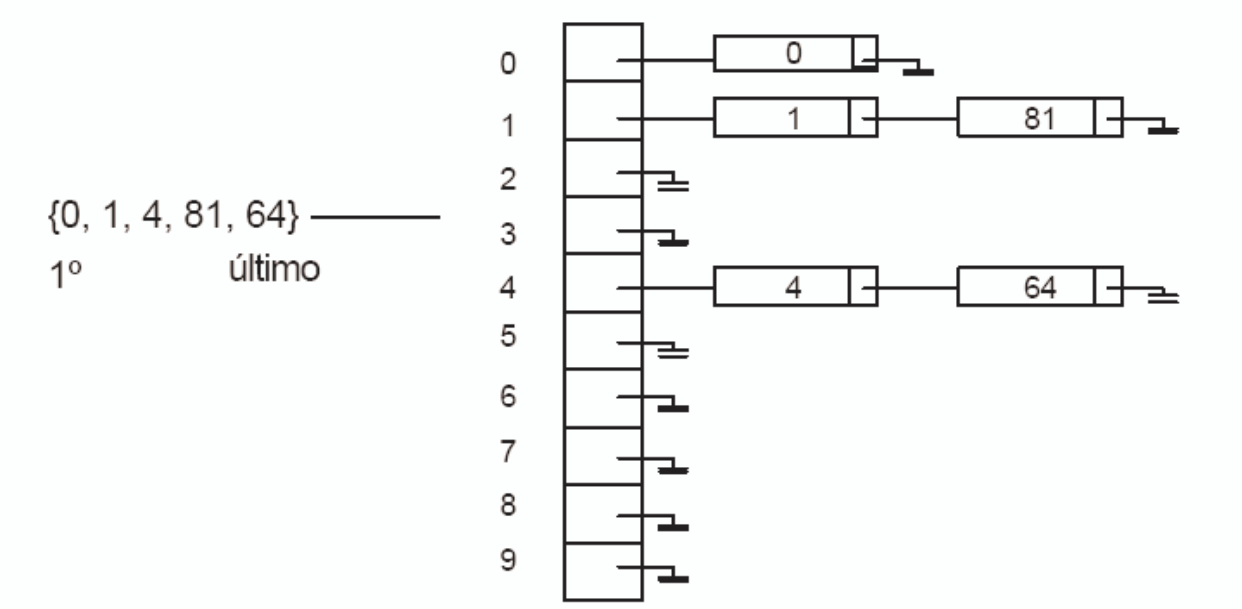
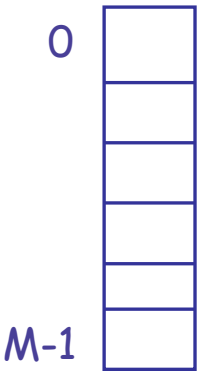
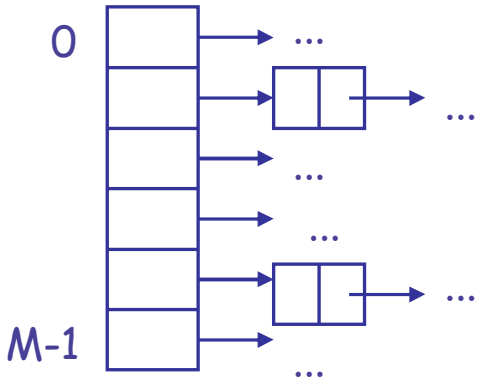


Tabela Base

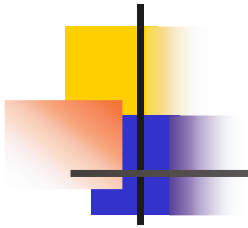
- **Dispersão com índices livres:**
  - Quando o número de elementos pode ser estimado à partida em N
  - Tabela de M elementos (M > N)
- **Dispersão por separação em listas:**
  - Quando o número de elementos a guardar (N) é desconhecido
  - Tabela de M listas de elementos (M < N)





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.



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.
    - **Nota:** Para reais num domínio pequeno
      - Escalar para a gama [0,1.0]
      - Multiplicar por 2^w, para obter um inteiro de w bits, **k**
      - Obter **`hash(k) = k mod M`**

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

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;}
```

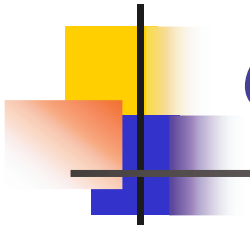
Resolução de Colisões

- Para uma tabela de tamanho M, quantas inserções podem ser feitas até à primeira colisão?
  - Para uma função de dispersão "aleatória" as primeiras colisões ocorrem ao fim de **sqrt(pi*M/2)**

M	sqrt(pi*M/2)
100	10
1021	40
10000	125

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

Algoritmos de resolução de colisões



- Algoritmos de resolução de colisões:
  - dispersão com índices livres, i.e., encadeamento interno:
    - Procura Linear (*Linear Probing*);
    - Dupla Dispersão (*Double Hashing*).
  - dispersão por separação em listas, i.e., encadeamento externo (*Separate Chaining*).

Resolução de Colisões - Procura Linear

- N < M : método com índices livres.
  - Dado que há sempre posições livres na tabela, procurar outra posição.
- Procura linear:
  - se a posição correspondente ao índice devolvido pela função de dispersão estiver ocupada, ir incrementando o índice até se encontrar uma posição livre.

0	70
1	14
2	72
3	77
4	
5	19
6	75

Ordem de inserção: 70, 72, 14, 77, 19, 75

```
hash(k) = k mod 7
```

Resolução de Colisões - Procura Linear

```
public class LPHashTable{
/*nesta classe supõe-se que N<M*/
   private Item[] st;
/*Item é um tipo classe; cada instância da classe Item tem um método
   key() que retorna a chave do objecto.*/
   private int N,M;
   public LPHashTable(int maxN){
        N=0; M=2*maxN; st=new Item[M];
   }
/*função de dispersão. O primeiro parâmetro é do tipo da chave do Item*/
   private int hash(Key x, int y) {...}/*Key é o tipo da chave*/
   public void insert(Item item){...}
   public Item search(Item item){...}
   public void remove(Item item){...}
   }
```

Cátia Vaz

Resolução de Colisões - Procura Linear

```
public void insert(Item item){
   int i=hash(item.key(),M); /*função de dispersão*/
   while(st[i]!=null)
        i=(i+1)%M;
   st[i]=item; N++;
   }
/*supõe-se que o método equals é um método estático implementado na
   classe Item*/
public Item search(Key k){
   int i=hash(k,M);
   while(st[i]!=null)
        if(Item.equals(item.key(),st[i].key()))
                return st[i];
        else i=(i+1)%M;
   return null;}
```

Cátia Vaz

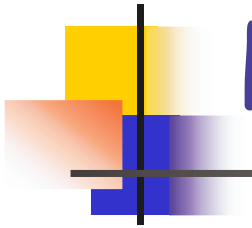
Resolução de Colisões - Procura Linear

- No processo de eliminação de um item, é necessário corrigir a posição dos items que:
  - foram inseridos após o item removido
  - as sua inserções tenham sido em posições mais a direita da posição do item removido, por esta já se encontrar ocupada.

```
public void remove(Item item){
   int i=hash(item.key(),M); /*função de dispersão*/
   while(st[i]!=null)
        if(Item.equals(item.key(),st[i].key())) break;
        else i=(i+1)%M;
   if(st[i]==null) return;
   st[i]=null; N--;
   for(int j=i+1; st[j]!=null; j=(j+1)%M){
      x=st[j]; st[j]=null; insert(x); N--;}
}
```

Cátia Vaz

Resolução de Colisões - Procura Linear



- Desempenho:
  - os elementos tendem a ficar agrupados (*clusters*);
  - os agrupamentos grandes tendem a crescer ainda mais;
  - o tempo médio de procura tende a crescer para M à medida que a tabela enche;
  - operações na tabela de dispersão tornam-se demasiado lentas quando a tabela atinge 70% - 80% da sua capacidade.

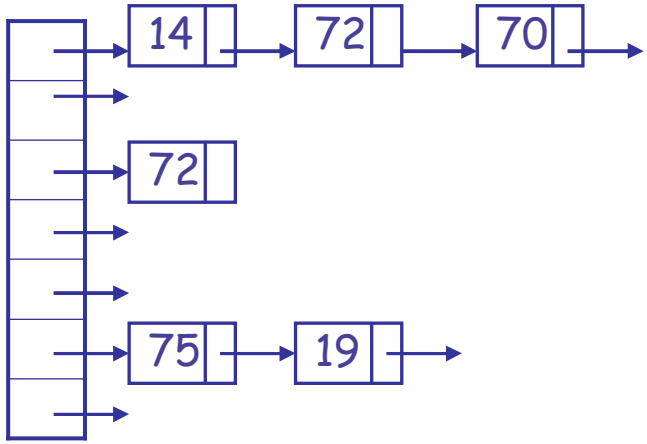
Resolução de Colisões - Procura Linear

- **Propriedade:** Quando as colisões são resolvidas com procura linear, o número de operações para encontrar um item numa tabela de dispersão de tamanho M que contém N=φM chaves (φ menor que 1) é aproximadamente:
  - com sucesso na procura – $\frac{1}{2}(1+ 1/(1- \varphi))$
  - com insucesso na procura - $\frac{1}{2}(1+ 1/(1- \varphi)^2)$

  Nota: esta estimativa **perde precisão** à medida que φ se aproxima de 1.

Resolução de Colisões – Encadeamento Externo

- Cada posição da tabela tem uma referência para uma lista
- Colisões são resolvidas juntando o elemento ao início da lista
- Remoções são resolvidas removendo o elemento da lista



Ordem de inserção:
70, 72, 14, 77, 19, 75

hash(k) = k mod 7

- Custo de inserção: O(1)
- Custo médio de procura (sem sucesso): O(N/M)
- Custo médio de procura (com sucesso): O(N/2M)

Cátia Vaz

Resolução de Colisões – Encadeamento Externo

```
public class EEHashTable{
   private Node[] heads;
   private int N,M;
   private class Node{
        Item item; Node next;
        Node(Item v){this.item=v;}
   }
   public EEHashTable(int maxN){
     N=0;M=maxN/5; heads=new Node[M];}
    /*função de dispersão. O primeiro parâmetro é do tipo da chave do
   Item*/
   private int hash(Key x, int y) {...}/*Key é o tipo da chave*/
   public void insert(Item item){...}
   public Item search(Key k){...}
   ...}
```

Cátia Vaz

Resolução de Colisões - Encadeamento Externo

```
public Item search(Key k){
   Node aux = heads[hash(k,M)];
   while(aux!=null){
     if(Item.equals(aux.item.key(),k)) return aux.item;
     aux=aux.next;}
   return null;
}
public void insert(Item item){
   int i= hash(item.key(),M);
   Node novo=new Node(item);
   novo.next=heads[i];
   heads[i]=novo;}
   }
```

Cátia Vaz