

# Título do Documento

Seu Nome

January 4, 2025

Teste

# 1. INTRODUÇÃO À INDEXAÇÃO

## 1.1 O que é indexação?

Um índice é uma estrutura de dados que melhora a velocidade das operações de query de dados em uma tabela, ao custo de escritas adicionais e espaço de armazenamento para manter a estrutura do índice. Índices permitem localizar dados rapidamente sem precisar buscar sequencialmente em cada linha de uma tabela.

A maioria dos softwares de banco de dados inclui tecnologia de indexação que permite buscas em tempo sub-linear para melhorar o desempenho, já que a busca linear é ineficiente para grandes bancos de dados.

[1]

## 1.2 Implementações de índices

### 1.2.1 B Tree

Uma **B-tree** é uma estrutura de dados em árvore auto-balanceada que mantém dados ordenados e permite buscas, acessos sequenciais, inserções e deleções em tempo logarítmico. A B-tree generaliza a árvore binária de busca, permitindo nós com mais de dois filhos.

É amplamente utilizada em sistemas de arquivos e bancos de dados. É uma estrutura que se beneficia da leitura e escrita em bloco, levando vantagem em um aspecto historicamente relevante, uma vez que o número de operações de I/O (em discos magnéticos) era igualmente relevante para o desempenho quanto o número de operações de comparação.

Foi inventada por Rudolf Bayer e Edward M. McCreight em 1972 [2] (o B não foi explicado por eles).

*What Rudy (Bayer) likes to say is, the more you think about what the B in B-Tree means, the better you understand B-Trees!*

Os principais algoritmos associados a B-trees são: busca (algorithm 1) e inserção (algorithm 2) (existem variações para a operação de deleção).

São necessárias duas funções auxiliares para a inserção: SPLITCHILD, que divide um nó cheio em dois, e INSERTNONFULL, que insere uma chave em um nó não cheio.

*Bulk loading?*

---

**Algorithm 1** Algoritmo de busca na B Tree, assumindo que a chave  $k$  é o valor a ser buscado e  $x$  é o nó onde a busca começa.

---

```

1: procedure BTREESEARCH( $x, k$ )
2:    $i \leftarrow 0$ 
3:   while  $i < x.n$  and  $k > x.key[i]$  do
4:      $i \leftarrow i + 1$ 
5:   end while
6:   if  $i < x.n$  and  $k = x.key[i]$  then
7:     return  $x$ 
8:   end if
9:   if  $x.leaf$  then
10:    return None
11:  end if
12:  return BTREESEARCH( $x.child[i], k$ )
13: end procedure

```

---



---

**Algorithm 2** Algoritmo de inserção na B Tree, assumindo que a chave  $k$  é o valor a ser inserido.

---

```

1: procedure BTREEINSERT( $T, k$ )
2:    $r \leftarrow T.root$ 
3:   if  $r.n = 2(T.d) - 1$  then
4:      $s \leftarrow$  new Node
5:      $T.root \leftarrow s$ 
6:      $s.child[1] \leftarrow r$ 
7:     SPLITCHILD( $s, 1$ )
8:     INSERTNONFULL( $s, k$ )
9:   else
10:    INSERTNONFULL( $r, k$ )
11:  end if
12: end procedure

```

---

[2], [3], [4]

### 1.2.2 B+ Tree

Uma B+ tree pode ser vista como uma B-tree onde cada nó contém apenas chaves (não pares chave-valor), com um nível adicional de folhas ligadas na parte inferior.

O principal valor de uma B+ tree está no armazenamento de dados para recuperação eficiente em um contexto de armazenamento orientado a blocos, como sistemas de arquivos. Diferente das árvores binárias de busca, as B+ trees têm um fanout muito alto (número de ponteiros para nós filhos em um nó, tipicamente na ordem de 100 ou mais), o que reduz o número de operações de I/O necessárias para encontrar um elemento na árvore.

Aplicações: iDistance

[5]

## 1.3 Indexação Multidimensional

Exemplo de como estava sendo realizada a indexação multidimensional em multimedia(imagens)

Efficient and Effective Querying by Image Content

### 1.3.1 R-Tree

Usada para multidimensional

### 1.3.2 M-Tree

Usada para espaços métricos

### 1.3.3

Survey [6]

## BIBLIOGRAPHY

- [1] Wikipedia, *Database index*, 2024. [Online]. Available: [https://en.wikipedia.org/wiki/Database\\_index](https://en.wikipedia.org/wiki/Database_index).
- [2] R. Bayer and E. McCreight, “Organization and maintenance of large ordered indices,” in *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*, 1970, pp. 107–141.
- [3] Wikipedia, *B tree*, 2024. [Online]. Available: <https://en.wikipedia.org/wiki/B-tree>.
- [4] D. Comer, “Ubiquitous b-tree,” *ACM Computing Surveys (CSUR)*, vol. 11, no. 2, pp. 121–137, 1979.
- [5] Wikipedia, *B+ tree*, 2024. [Online]. Available: [https://en.wikipedia.org/wiki/B%2B\\_tree](https://en.wikipedia.org/wiki/B%2B_tree).
- [6] V. Gaede and O. Günther, “Multidimensional access methods,” *ACM Computing Surveys (CSUR)*, vol. 30, no. 2, pp. 170–231, 1998.

## 2. INDEXAÇÃO PARA FINGERPRINTS

### 2.1 Introdução

- Matching local baseado em minúcias
  - Abordagens antigas [1], [2]
  - Associa cada minúcia as suas vizinhas em estruturas invariantes a rotação e distâncias[3], [4]
  - Baseada em cilindros section 2.2
  - Outros métodos que incluem mais características: local orientation field, local frequency, ridge shapes

Estruturas locais de uma minúcia central podem ser baseadas em:

- *Vizinhos mais próximos*, que consideram as  $k$  minúcias mais próximas [3].

A vantagem dessa representação é com relação ao tamanho fixo, facilitando no procedimento de comparação.

- *Raio fixo*, que considera todas as minúcias dentro de um raio fixo, usada em [4].

A vantagem dessa representação é a tolerância com relação a ruído (minúcias extras ou faltantes).

### 2.2 Minutia Cylinder-Code

Baseada em [5], [6]

#### 2.2.1 Representação

Uma representação tridimensional de minúcias baseada em distâncias entre minúcias e ângulos relativos. A representação recebe o nome de Minutia Cylinder-Code (MCC) e tem como características principais: invariância de rotação, tamanho fixo e orientada a codificação binária.

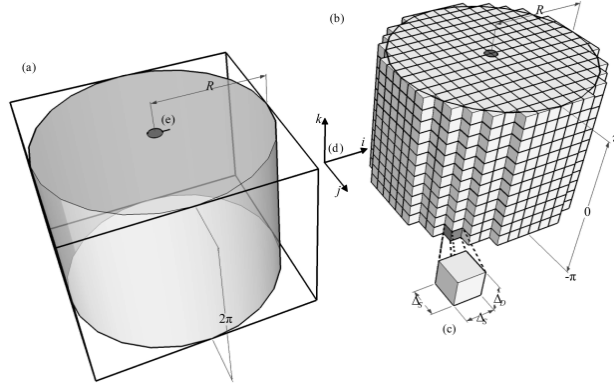


Fig. 2.1: Representação de um MCC.

O esquema de representação é um cilindro segmentado, como na fig. 2.1. O cilindro é um recorte de um cubo dividido em células, onde cada uma possui um valor indexado por  $C_m[i, j, k]$ .

O cálculo dos valores de  $C_m[i, j, k]$  é complicado, mas essencialmente envolve as seguintes ideias:

- Verifica se está em uma região válida: dentro do cilindro e dentro do *convex hull* da fingerprint.
- Calcula a contribuição de cada minúcia vizinha usando uma Gaussiana, fig. 2.2.
- Calcula a contribuição de cada minúcia usando a diferença entre a orientação.

### 2.2.2 Similaridade

A similaridade entre dois cilindros é obtida a partir do seguinte procedimento:

1. Lineariza o cilindro em um vetor, similar a operação de **reshape**. Por exemplo, o cilindro de uma minúcia  $a$ ,  $C_a[i, j, k]$ , é linearizado em  $\mathbf{c}_a$ .
2. Seleciona todas as entradas comparáveis desses vetores (células que são *válidas* em ambos) que dão origem aos vetores  $\tilde{\mathbf{v}}_a$  e  $\tilde{\mathbf{v}}_b$ .
3. Na implementação binária, é realizado um XOR bit a bit entre os vetores.

$$\gamma(\tilde{\mathbf{v}}_a, \tilde{\mathbf{v}}_b) = 1 - \frac{\|\tilde{\mathbf{v}}_a \oplus \tilde{\mathbf{v}}_b\|}{\|\tilde{\mathbf{v}}_a\| + \|\tilde{\mathbf{v}}_b\|} \quad (2.1)$$



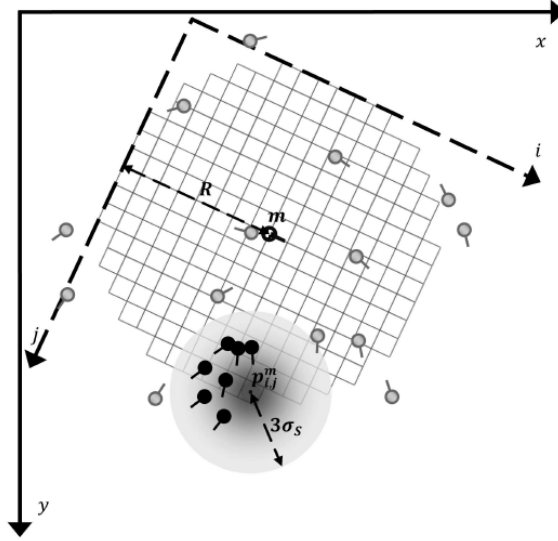


Fig. 2.2: Contribuição da distância de uma minúcia vizinha. Cores mais escuras refletem uma maior contribuição.

### 2.2.3 Indexação

Para a indexação de um MCC é empregado um esquema de *locality-sensitive hashing* (LSH).

O procedimento consiste em:

1. Seja  $\mathbf{v}_m$  o vetor obtido da linearização de um cilindro.
2. É feita a projeção de  $\mathbf{v}_m$  em um subespaço  $\mathbf{h}_m$ . A projeção é obtida simplesmente ao escolher um subconjunto dos índices  $H$  do vetor original.

$$\mathbf{h}_m = \mathbf{v}_m[H] \quad (2.2)$$

3. O conjunto  $H$  define uma função que mapeia um vetor binário  $\mathbf{v}_m$  em um número natural obtido ao interpretar o vetor binário  $\mathbf{h}_m$  como um número natural.

$$h_H : \{0, 1\}^n \rightarrow \mathbb{N} \quad (2.3)$$

4. São definidos  $\ell$  conjuntos  $H_1, H_2, \dots, H_\ell$ , cada um com uma função  $h_{H_i}$ .
5. O índice por sua vez é um conjunto de *hash tables*,  $\mathbb{H}_1, \mathbb{H}_2, \dots, \mathbb{H}_\ell$ , onde cada hash table tem os seus buckets definidos pela função  $h_{H_i}$ .

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$	$i_9$	$i_{10}$	$i_{11}$	$i_{12}$	$i_{13}$	$i_{14}$	$i_{15}$	$f_{H_1}$	$f_{H_2}$	$f_{H_3}$
$v_1$	0	1	1	1	0	0	1	0	1	0	0	0	0	1	0	7	4	0
$v_2$	0	0	1	0	1	0	1	0	0	0	0	1	0	0	1	2	5	3
$v_3$	1	0	1	0	1	0	0	0	0	1	0	0	0	1	0	0	6	6
$v_4$	0	0	0	0	0	1	1	0	1	0	1	1	1	0	0	3	1	0
$v_5$	1	1	1	0	1	0	0	0	1	0	0	1	0	0	0	5	5	6
$v_6$	1	0	1	0	0	1	0	0	0	1	0	0	0	1	1	0	6	5
$v_7$	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	3	3	0
$v_8$	1	0	0	0	0	1	1	1	0	1	0	0	1	0	0	2	2	4
$v_9$	0	1	0	1	1	0	0	0	1	1	0	0	0	0	1	5	2	3
$v_{10}$	0	1	1	0	0	0	1	1	1	0	1	0	0	0	1	7	4	1

$H_1$	$H_2$	$H_3$
$0 = 000_2 \{v_3, v_6\}$	$0 = 000_2 \emptyset$	$0 = 000_2 \{v_1, v_4, v_7\}$
$1 = 001_2 \emptyset$	$1 = 001_2 \{v_4\}$	$1 = 001_2 \{v_{10}\}$
$2 = 010_2 \{v_2, v_8\}$	$2 = 010_2 \{v_8, v_9\}$	$2 = 010_2 \emptyset$
$3 = 011_2 \{v_4, v_7\}$	$3 = 011_2 \{v_7\}$	$3 = 011_2 \{v_2, v_9\}$
$4 = 100_2 \emptyset$	$4 = 100_2 \{v_1, v_{10}\}$	$4 = 100_2 \{v_8\}$
$5 = 101_2 \{v_5, v_9\}$	$5 = 101_2 \{v_2, v_5\}$	$5 = 101_2 \{v_6\}$
$6 = 110_2 \emptyset$	$6 = 110_2 \{v_3, v_6\}$	$6 = 110_2 \{v_3, v_5\}$
$7 = 111_2 \{v_1, v_{10}\}$	$7 = 111_2 \emptyset$	$7 = 111_2 \emptyset$

	$i_1$	$i_2$	$i_3$	$i_4$	$i_5$	$i_6$	$i_7$	$i_8$	$i_9$	$i_{10}$	$i_{11}$	$i_{12}$	$i_{13}$	$i_{14}$	$i_{15}$	$f_{H_1}$	$f_{H_2}$	$f_{H_3}$
$v_S$	1	1	1	0	0	0	1	0	1	0	1	0	0	1	0	7	4	4

Fig. 2.3: Ilustração do procedimento de indexação usando LSH de um MCC.

6. A indexação segue fazendo a consulta do vetor desejado em cada hash table, retornando os candidatos.
7. Por fim, os candidatos são ranqueados usando a *distância de Hamming* entre os vetores.

O procedimento enumerado acima é ilustrado na fig. 2.3.

## BIBLIOGRAPHY

- [1] A. K. Hrechak and J. A. McHugh, “Automated fingerprint recognition using structural matching,” *Pattern recognition*, vol. 23, no. 8, pp. 893–904, 1990.
- [2] A. J. Willis and L. Myers, “A cost-effective fingerprint recognition system for use with low-quality prints and damaged fingertips,” *Pattern recognition*, vol. 34, no. 2, pp. 255–270, 2001.
- [3] X. Jiang and W.-Y. Yau, “Fingerprint minutiae matching based on the local and global structures,” in *Proceedings 15th international conference on pattern recognition. ICPR-2000*, IEEE, vol. 2, 2000, pp. 1038–1041.
- [4] N. K. Ratha, R. M. Bolle, V. D. Pandit, and V. Vaish, “Robust fingerprint authentication using local structural similarity,” in *Proceedings Fifth IEEE Workshop on Applications of Computer Vision*, IEEE, 2000, pp. 29–34.
- [5] R. Cappelli, M. Ferrara, and D. Maltoni, “Minutia cylinder-code: A new representation and matching technique for fingerprint recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 12, pp. 2128–2141, 2010.
- [6] R. Cappelli, M. Ferrara, and D. Maltoni, “Fingerprint indexing based on minutia cylinder-code,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 33, no. 5, pp. 1051–1057, 2010.