Análise de estruturas de indexação para consultas por similaridade com condições adicionais

Rodrigo Mimura Shimomura[1], **Daniel dos Santos Kaster**[1]

[1]Departamento de Computação – Universidade Estadual de Londrina (UEL)
Caixa Postal 10.011 – CEP 86057-970 – Londrina – PR – Brasil

shimomura.rodrigo@uel.br, dskaster@uel.br

Abstract. *With the accelerated development of technologies and means of communication, there is a need to store, manage and recover not only traditional data such as numbers and texts, but also complex information such as photos, videos, genetic sequences and geographic coordinates, for example. Database management systems, originally designed for storing simple data, face difficulties in standardization and retrieval of these types of complex information, stored in binary format. To overcome this obstacle, similarity search emerges as an effective approach, optimizing queries and returning relevant results. This research aims to carry out a comparative analysis of different indexing structures for complex data, evaluating performance metrics such as access to indexes and data files, average query time with a special focus on searching for the k-nearest neighbors, aiming to identify the most suitable situations for the application of each proposed solution.*

Resumo. *Com o desenvolvimento acelerado de tecnologias e meios de comunicação, surge a necessidade de armazenar, gerenciar e recuperar não apenas dados tradicionais como números e textos, mas também informações complexas como fotos, vídeos, sequências genéticas e coordenadas geográficas, por exemplo. Os sistemas de gerenciamento de banco de dados, originalmente projetados para dados simples, enfrentam dificuldades na padronização e recuperação desses tipos de informações complexas, armazenados em formato binário. Para superar esse obstáculo, a busca por similaridade emerge como uma abordagem eficaz, otimizando consultas e retornando resultados relevantes. Esta pesquisa objetiva a realização de uma análise comparativa de diferentes estruturas de indexação para dados complexos, avaliando métricas de desempenho como acessos a arquivos de índices e dados, tempo médio de consulta com foco especial na busca dos k-vizinhos mais próximos, visando a identificação das situações mais adequadas para a aplicação de cada solução proposta.*

1. Introdução

Nos últimos anos, o mundo vem testemunhando a produção e troca massiva de dados digitais devido ao acelerado desenvolvimento dos meios de comunicação e das tecnologias [9]. Nesse cenário, a produção de dados abrange não somente os dados tradicionais como números e textos, mas também dados complexos que são informações não representáveis por apenas um dado simples, como representações visuais, registro de vídeos, coordenadas geográficas, séries temporais e sequências de DNA, por exemplo. À medida

que a sociedade está imersa com esse fluxo em profusão de informações, a necessidade de armazenar, gerenciar e recuperar tanto os dados tradicionais/simples quanto os dados complexos se torna cada vez mais indispensável [2][8].

Os sistemas de gerenciamento de banco de dados (SGBDs) foram projetados com foco no armazenamento e recuperação de informações de natureza simples, como valores numéricos e blocos de texto. Diante do exposto, torna-se evidente a existência de desafios na padronização e recuperação de dados complexos, uma vez que esses são armazenados no formato *BLOBs* (*Binary Large Objects*) [5], o qual guarda as informações em strings binárias de baixo nível. O problema de realizar a recuperação de dados complexos está no fato de que raramente consultas envolvendo igualdade são realizadas com esse tipo de informação, o que pode levar as buscas a se tornarem ineficientes em bancos de dados com abundância de dados.

Diante do exposto, o modo mais utilizado para recuperar dados complexos é por meio de buscas por similaridade, em virtude de otimizar os processos que o SGBD terá que realizar visando retornar os resultados das consultas [4]. Para realizar as buscas, são extraídas características relevantes ao dado, guardando-as em *feature vectors* (vetores de características) e posteriormente definindo um critério de similaridade para cada tabela do banco de dados. Por exemplo, a criação dos vetores de características de eletrocardiogramas (ECG) utiliza os coeficientes do método DWT (*Discrete Wavelet Transform*) (após aplicados para remoção de ruídos e melhoramento da qualidade dos exames) [13]. Após ter criado os vetores de características, o último passo seria definir o critério de similaridade, e posteriormente a utilização de uma função de distância para os cálculos de (dis)similaridade.

Um ponto a ser levado em consideração na realização das buscas é que geralmente os dados complexos são armazenados em conjunto com dados tradicionais, os quais devem ser utilizados na filtragem dos resultados e operações de similaridade visando otimizar e responder consultas complexas [6]. Diante do exposto, os mecanismos de busca por similaridade mais frequentemente utilizados são a consulta por abrangência (*Range query - Rq*), consulta dos k vizinhos mais próximos (*k-Nearest Neighbor query - k-NNq*), consulta dos k vizinhos mais próximos reverso (*RkNN*) [10]. Entretanto, aplicações de SGBDs que possuem alguns desses algoritmos implementados, foram criadas e estruturadas de maneira muito específica para cada caso, fato que dificulta a reutilização e a manutenção dos códigos que muitas vezes são ineficazes e custosos.

As principais estruturas de indexação que possuem dados complexos em seus elementos são a *Slim-tree* [16], *cx-Sim* tree* (*Condition-eXtended Similarity tree*) com suas 4 variações (*cx-sim* Simple, cx-sim* Covering, cx-sim* Chained, cx-sim* Composite*) [15], *S2I (Spatial Inverted Index)* [14] e *BSlim* [15], cada uma com suas peculiaridades, vantagens e desvantagens para cada caso de uso. Cada um dos autores mencionados já procedeu à implementação e execução de experimentos relativos a cada uma das estruturas, entretanto, o fizeram de forma independente. Tal circunstância suscita o interesse na condução de uma análise comparativa entre todas as estruturas acima mencionadas, levando em consideração distintos cenários de busca e conjuntos de dados diversos.

Dessa maneira, o objetivo deste trabalho de conclusão de curso é realizar uma análise comparativa das estruturas de indexação de dados complexos, com ênfase na

avaliação das métricas de desempenho como acessos ao arquivo de índices e dados e tempo médio de consulta, principalmente para a busca dos *k*-vizinhos mais próximos. A proposta será elaborar uma maneira de comparar as estruturas, bem como evidenciar as melhores situações de uso para cada tipo de maneira de indexação.

A Seção 2 apresenta os conceitos, métodos, técnicas e definições, além da revisão do estado da arte necessários para o desenvolvimento do trabalho. Na seção 3 são descritos os objetivos a serem atingidos ao final deste trabalho de conclusão de curso. A seção 4 apresenta como os objetivos serão alcançados utilizando a fundamentação teórica e metodológica. A seção 5 exibe o cronograma de execução das atividades propostas na seção da metodologia. E por fim, a seção 6 enumera as contribuições esperadas do trabalho para o progresso e fortalecimento do conhecimento dos leitores.

2. Fundamentação Teórico-Metodológica e Estado da Arte

2.1. Busca por similaridade

Sequências temporais, imagens, vídeos, coordenadas geográficas, sequências de DNA, resultados de experimentos científicos e arquivos de texto extensos e compactados (.pdf) são chamados dados complexos devido à incapacidade de representá-los com apenas um tipo de dado simples como um número inteiro, cadeia de caracteres ou ponto flutuante [3]. A maioria dos domínios complexos não possuem relações de ordem total entre seus elementos, o que implica a incapacidade de utilizar operadores de comparação relacionais ($>, \geq, <, \leq$). Já operadores "=" e "$\neq$" não possuem utilidade, uma vez que não faz sentido procurar um áudio exatamente igual ao fornecido, salvo exceções quando o objetivo da consulta é verificar a existência do objeto de referência no banco de dados [4] [6].

A principal maneira de representar esses dados complexos para realizar comparações por similaridade consiste em 2 etapas primordiais. A primeira consiste em escolher um método para realizar a extração de características do dado em questão a fim de armazená-las em um vetor de características, por exemplo, a extração de informações relevantes de imagens consiste em 2 tipos de informação: características de baixo nível como cor, textura e forma e características de alto nível como objetos na imagem (utilizando os característica de baixo nível) [1]. A segunda etapa está na elaboração da função de distância entre elementos do conjunto, indicando a dissimilaridade entre dois vetores de características, portanto, a similaridade entre dois elementos é inversamente proporcional à sua distância [17].

Existem diversas famílias de funções de distâncias utilizadas na academia para cada estudo. Neste caso em particular de calcular de funções de dissimilaridade, as da família *Minkowski* são as mais utilizadas (equação 1):

$$L_p[(x_1, ..., x_n), (y_1, ..., y_n)] = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \tag{1}$$

Onde n é o tamanho do vetor de características e $1 \leq p \leq \infty$. Os valores mais frequentes de p são: $p = 0$ ou ∞ (distância Chebyshev), $p = 1$ (distância de Manhattan) e $p = 2$ (distância Euclidiana) [4].

2.2. Espaço métrico

O espaço métrico pode ser definido como o par $(\mathbb{S}, d)$, sendo $\mathbb{S}$ representa o domínio dos objetos e d representa a função de distância [17]. As propriedades da função $d : \mathbb{S}\times\mathbb{S} \rightarrow \mathbb{R}$ para $\forall x, y, z \in \mathbb{R}$ são tipicamente caracterizadas como:

- Não negatividade: $d(x, y) \geq 0$
- Simetria: $d(x, y) = d(y, x)$
- Identidade: $x = y \Leftrightarrow d(x, y) = 0$
- Desigualdade triangular: $d(x, z) \leq d(x, y) + d(y, z)$

2.3. Tipos de busca por similaridade

Seja o espaço métrico $(\mathbb{S}, d)$, com o conjunto de dados $S \subseteq \mathbb{S}$, um elemento de consulta $q \in \mathbb{S}$, as principais maneiras de realizar consultas baseadas em similaridade são pelos k-vizinhos mais próximos (*k-Nearest Neighbors queue* - $kNNq$) e buscas por abrangência (*Range queue* - *Rq*) [4]

2.3.1. *Range query - Rq*

A consulta $Rq(q, r)$ retorna para um grau de tolerância $r \in \mathbb{R}^+$ todos os objetos s que satisfazem a condição $d(s, q) \leq r$. Em notação de conjunto, temos:

$$Resultado = \{s_i \in S | d(q, s_i) \leq r\} \tag{2}$$

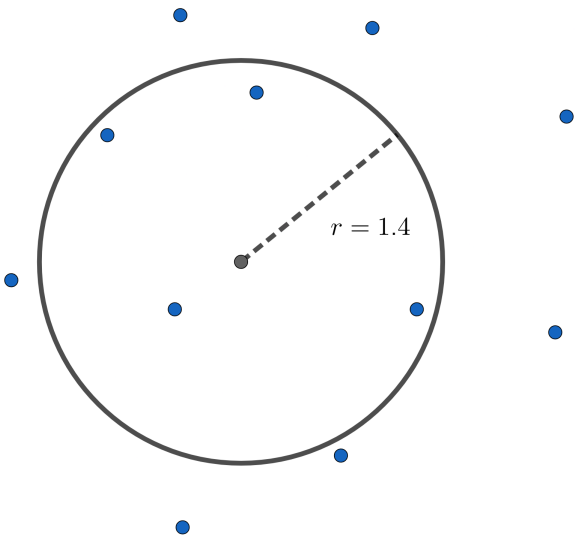

Figura 1. Exemplo da Rq com $r = 1.4$

2.3.2. *k-Nearest Neighbors query - k-NNq*

A consulta $k-NNq(q,k)$ retorna os k vizinhos mais próximos do elemento de referência q. Em notação de conjunto, temos:

$$Resultado = \{s_i \in S | \forall s_j \in S \setminus K, |K| = k, d(s_q, s_i) \leq d(s_q, s_j)\} \tag{3}$$

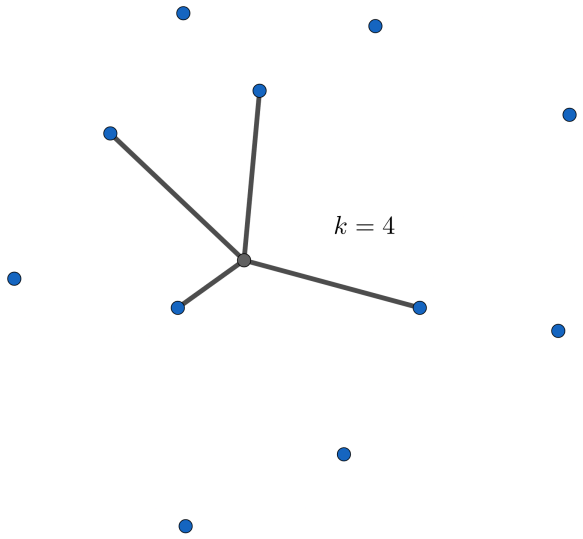

Figura 2. Exemplo de $kNNq$ com $k = 4$

2.4. Estruturas de indexação

2.4.1. *Slim-tree*

A *Slim tree* é uma árvore balanceada e dinâmica, na qual os elementos estão organizados de maneira hierárquica utilizando um elemento representante que cobre uma região considerando um raio máximo de cobertura. Nessa abordagem, os objetos são agrupados em páginas de disco de tamanho fixo correspondendo a nós da árvore. As distâncias de cada elemento em relação ao representante são calculadas durante a construção da árvore, diminuindo a necessidade de realizar cálculos de distância durante as buscas [16][6].

Existem 2 tipos de nós na *Slim tree* [6]:

- Nós folha: armazenam um vetor de tuplas de 3 elementos no formato:

$$< id_i, d(s_i, s_{rep}), s_i >$$

em ordem, temos o identificador do elemento, a distância em relação ao elemento representante e o elemento propriamente dito.

- Nós índice: armazenam um vetor de tuplas de 5 elementos no formato:

$$< s_i, r_i, d(s_i, s_{rep}), Ptr(T_{s_i}), \#Ent(Ptr(T_{s_i})) >$$

onde s_i é o elemento representante da subárvore apontada por $Ptr(T_{s_i})$, r_i é o raio da região circular coberta pelo nó (distância entre o representante e o elemento mais distante desse nó), $d(s_i, s_{rep})$ é a distância entre s_i e s_{rep}, por fim, $\#Ent(Ptr(T_{s_i}))$ é o número de entradas da subárvore apontada por $Ptr(T_{s_i})$.

A figura 3 mostra a representação visual da estrutura da *Slim-tree*. Os pontos em preto são os representantes, os demais elementos são os pontos cinza. Os nós folhas estão representandos nos círculos brancos e os nós índice estão representados em círculos cinza.

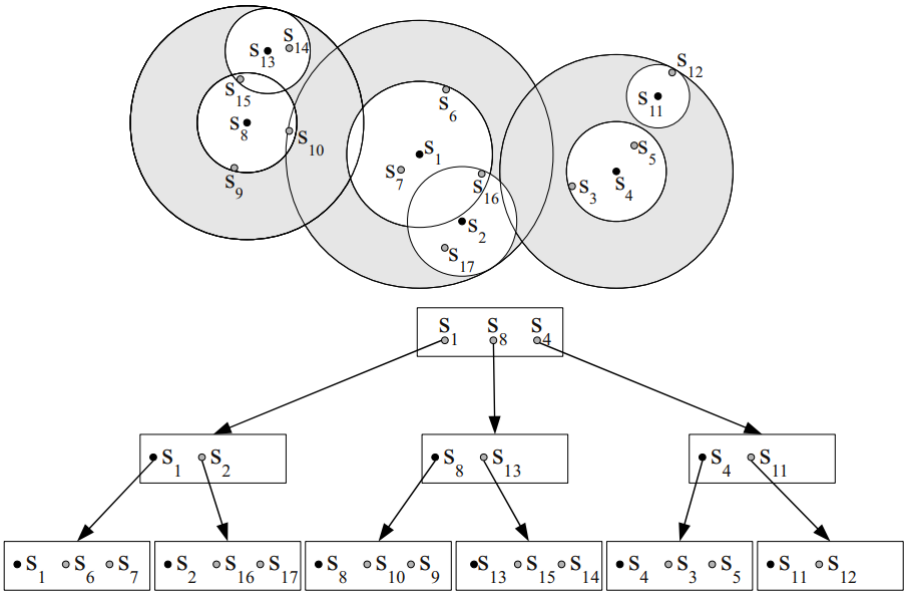

Figura 3. Representação da *Slim-tree*

2.4.2. *cx-Sim* Simple Tree*

Proposto por [15], a *cx-Sim* tree* (*Condition-eXtended Similarity tree*) é uma adição à *Slim-tree* apresentada por [16] que possibilita a introdução de atributos tradicionais ao índice. A principal vantagem da *cx-Sim* tree* é verificar as condições das consultas por similaridade diretamente no índice, antes mesmo de acessar o arquivo de dados do SGBD. Isso otimiza as consultas pelo fato das comparações de similaridade e informações tradicionais são feitas simultaneamente, reduzindo assim, o acesso ao arquivo de dados.

De maneira simplificada, a *cx-Sim* tree* trabalha com 2 camadas no índice. A primeira possui uma ou mais B^+*-trees* responsáveis por fazer a indexação dos atributos simples, em geral, é recomendável escolher o atributo com a maior seletividade para maximizar a poda de sub-árvores durante a busca. Já na segunda, existe uma floresta de métodos de acesso métricos que indexam os dados complexos, nesse caso de implementação foram utilizadas *Slim-trees*.

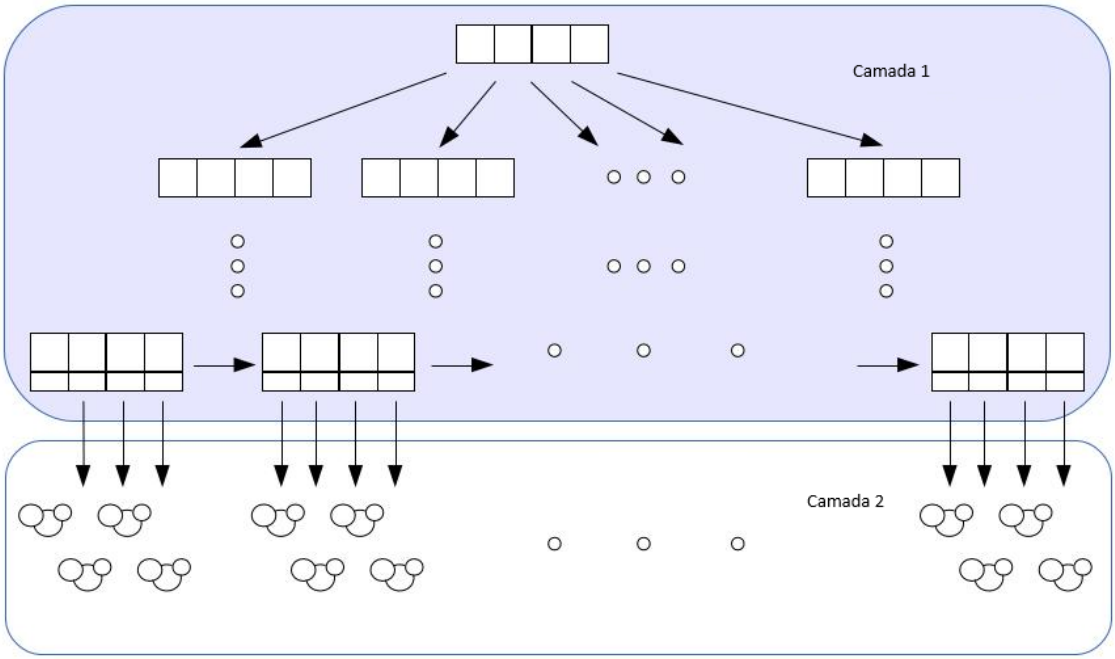

Figura 4. Estrutura da *cx-Sim* tree* - Adaptado de [15]

De maneira resumida, simulando a busca por similaridade utilizando $kNNq$ com condições nos atributos simples, considerando que na primeira camada foi utilizado o atributo com a maior seletividade na construção da *B*$^+$*-tree*:

1. A busca no índice inicia-se considerando o atributo A_1 na *B*$^+$*-tree* (camada 1).
2. Após encontrada a floresta correspondente ao conjunto que satisfaz a primeira condição, a busca $kNNq$ começa considerando os critérios de similaridade em relação ao elemento de consulta s_q, e funções de distâncias pré-definidas no espaço métrico (camada 2).
3. Tendo selecionado os elementos que atendam aos critérios da consulta $kNNq$, caso existam outros atributos envolvidos nas condições de busca, estes terão de ser verificados somente após a recuperação dos $rowIds$ no arquivo de dados.

As próximas variações presentes nas seções 2.4.3, 2.4.4 e 2.4.5, também propostas em [15], alteram principalmente a localização dos atributos tradicionais ou o formato das camadas, objetivando reduzir o número de elementos a serem comparados nas buscas por similaridade, em consequência de podarem sub-árvores não promissoras usando atributos de alta seletividade.

2.4.3. *cx-Sim* Covering Tree*

A abordagem *covering* tem a ideia de mudar o posicionamento dos atributos $A_2 \ldots A_n$, visando **cobrir** todas as condições a serem verificadas durante a busca na *cx-Sim* tree*. Para realizar este objetivo, a variação *cx-Sim* Covering tree* opera de maneira semelhante na camada 1, utilizando um atributo tradicional A_1 com alta seletividade na *B*$^+$*-tree*, enquanto os outros atributos $A_2 \ldots A_n$ são armazenados dentro do vetor de características

presente na camada 2, **cobrindo** assim o restante das condições a serem avaliadas durante a busca por similaridade.

2.4.4. *cx-Sim* Chained Tree*

Levando em consideração a ideia de utilizar atributos de alta seletividade para otimizar o tempo de busca, a construção da *cx-Sim* Chained tree* altera principalmente o formato da camada 1. Isso acontece pelo fato da camada um possuir uma **sequência hierárquica** de B$^+$trees, onde cada uma indexa um atributo simples. A consequência deste modelo de construção da *cx-Sim* chained tree* é possibilitar a verificação das condições envolvendo atributos simples antes de realizar as buscas por similaridade.

2.4.5. *cx-Sim* Composite Tree*

Nesta última variação proposta por [15], a *cx-Sim* Composite tree* transfere todos os atributos simples para a camada 1, alterando a chave de busca na *B$^+$-tree* a qual se torna uma **combinação** de todos os atributos tradicionais $A_1 \dots A_n$. A principal vantagem da mudança do formato da chave na *B$^+$-tree*, reside no fato de que há uma redução do número de nós índice na estrutura. É de suma importância destacar que as comparações envolvendo a chave composta durante o processo de busca na estrutura da *cx-Sim* Composite tree* comparam atributo por atributo de maneira ordenada. Ou seja, considere 2 vetores de características X e Y, o processo de comparação testa $X(A_1)$ com $Y(A_1)$, caso $X(A_1) = Y(A_1)$, $X(A_2)$ com $Y(A_2)$ são comparados, e assim por diante.

2.4.6. *B-Slim tree*

Esta estrutura foi proposta para realizar a comparação da performace entre o formato de 2 camadas presente na *cx-Sim* Tree* 2.4.2 e a utilização de uma *Slim-tree* e uma *B$^+$-tree* de maneira autônoma. Chamado de *B-Slim*, esse algoritmo exige a existência de uma *Slim-tree* que indexe o atributo complexo para a realização da busca por similaridade, além de uma *B$^+$-tree* indexando o atributo simples utilizado na condição, sendo que as 2 estruturas são independentes [15]. Um ponto importante a ser levado em consideração é que durante a inserção de um novo dado neste modelo, haverá 2 inserções, uma na *B$^+$-tree* indexando o atributo tradicional, e outra na *Slim-tree* indexando o dado complexo.

O algoritmo opera a busca em 2 etapas, uma busca na *B$^+$-tree* e retornando os *rowIds* que satisfazem a condição do atributo simples e outra na *Slim-tree* executando a consulta por similaridade com condições. É importante destacar que elementos candidatos farão parte do conjunto resposta, desde que seus *rowIds* estejam presentes em ambos resultados obtidos nas etapas anteriores. A interseção entre os *rowIds* é feita **durante** a busca por similaridade, e não posterior a ela.

2.4.7. *S2I*

O *S2I (Spatial Inverted Index*) proposto por [14] foi criado visando otimizar a busca de palavras-chaves em um conjunto de dados, utilizando um índice invertido. Essa estrutura está no cerne de motores de busca em larga escala [12], e consiste em uma técnica de armazenamento de textos na qual para cada termo presente, existe uma lista de documentos onde é possível encontrá-lo [7]. Outra informação importante armazenada nessa estrutura, é a frequência da presença dos termos nos textos, o que possibilita acelerar as buscas, ao passo que é feito um ”rankeamento”das palavras mais comuns [7].

O *S2I* mapeia cada uma das palavras-chave em 2 estruturas dependendo do número de ocorrências. Termos mais frequentes são armazenados em *aR-Trees* (*Aggregated R-tree*)[11], enquanto os menos frequentes são colocados em blocos [14]. A tabela 1 demonstra a organização feita no *S2I*.

Os componentes presentes no *S2I* são 3:

- Vocabulário: Armazena para cada termo o número de objetos/textos que possuem o termo, uma flag indicando qual o tipo de estrutura que está armazenando o termo (bloco ou *aR-Tree*) e um ponteiro para a localização da estrutura
- Blocos: Cada bloco é um conjunto de objetos, onde cada objeto possui um identificador único, a localização (geográfica, por exemplo) e o impacto do termo na descrição do objeto.
- Árvores: *aR-trees* de cada termo se assemelham com as *R-trees*, onde cada nó intermediário guarda o MBR (*Minimum Bounding Rectangle*) de cada um dos nós filhos. O que diferencia as *aR-trees* das *R-trees* reside no fato que os nós da *aR-tree* armazenam também valores agregados não espaciais (não complexos).

Termo	ID	Freq	Tipo armazenamento	Ponteiro	Armazenamento
Automóvel	ID1	5	Árvore	0x3A1F5678	aR^{id_1}
Volante	ID2	3	Bloco	0x8EBC2A95	$< p_1, p_3, p_5 >$
Rodas	ID3	7	Árvore	0x17D49FBE	aR^{id_3}
Capacete	ID4	2	Bloco	0xA0EF834D	$< p_2, p_4 >$

Tabela 1. Representação visual do *S2I*

3. Objetivos

O presente trabalho de conclusão de curso tem como propósito a exploração e condução de uma análise comparativa das estruturas de indexação de dados complexos de diversas naturezas, tais como imagens e coordenadas geográficas, sob condições específicas. Adicionalmente, incluem-se entre os objetivos a modificação dos códigos das referidas estruturas, de modo a assegurar a execução dos experimentos de maneira imparcial, bem com a criação de um ambiente para a realização de testes com *scripts* e diferentes tipos de *datasets*.

4. Procedimentos metodológicos/Métodos e técnicas

Pretende-se iniciar o estudo por meio de uma revisão bibliográfica das estruturas utilizadas para indexação de dados complexos, a fim de realizar buscas por similaridade. Após seleção das técnicas encontradas, será feito um estudo particular e implementação com modificações de cada método escolhido, visando uma análise objetiva de diferenciação dos métodos de indexação.

Posteriormente a esta etapa, será criado um ambiente dedicado à realização dos testes. Este ambiente compreenderá a seleção e preparação e conjunto de dados com distintas tipologias, assim como o desenvolvimento de *scripts* para a condução dos testes. Essa sequência de etapas possibilitará, por fim, a análise dos resultados obtidos, além de elucidar as melhores situações de uso das estruturas escolhidas.

5. Cronograma de Execução

Atividades:

1. Revisão bibliográfica e escolha dos algoritmos a serem comparados;
2. Estudo e implementação dos algoritmos selecionados;
3. Modificações nos códigos fonte visando à aquisição de métricas;
4. Criação do ambiente de testes;
5. Execução de testes com diferentes tipos de dados complexos;
6. Análises dos resultados obtidos;
7. Escrita da versão preliminar do TCC;
8. Escrita da versão para a banca examinadora do TCC;

Tabela 2. Cronograma de Execução

	ago	set	out	nov	dez	jan	feb	mar	abr	mai
Atividade 1	•									
Atividade 2	•	•	•	•						
Atividade 3		•	•	•						
Atividade 4			•	•	•					
Atividade 5				•	•	•	•	•		
Atividade 6						•	•	•	•	
Atividade 7		•	•	•	•					
Atividade 8					•	•	•	•	•	•

6. Contribuições e/ou Resultados esperados

Ao final do trabalho, os seguintes pontos são esperados:

- Aprofundar-se no entendimento do funcionamento das estruturas de indexação para consultas por similaridade com condições.
- Elucidar e demonstrar de forma simples e concisa maneiras de realizar comparações entre as principais estruturas utilizadas atualmente.
- Auxiliar na escolha das melhores estruturas para cada caso de uso, levando em consideração o tipo de dado e número de condições envolvidas no processo de busca por similaridade.
- Elaboração de um ambiente de testes completo para a realização de comparações com outras estruturas de indexação.

7. Espaço para assinaturas

Londrina, *15 de setembro de 2023*.

_____ _____

Aluno Orientador

Referências

[1] Sushila Aghav-Palwe and Dhirendra Mishra. Feature vector creation using hierarchical data structure for spatial domain image retrieval. *Procedia Computer Science*, 167:2458–2464, 2020.

[2] Fabio Gomes de ANDRADE et al. Sesdi: um arcabouço para a recuperação de dados geográficos em infraestruturas de dados espaciais. 2012.

[3] Maria Camila Nardini Barioni. Operações de consulta por similaridade em grandes bases de dados complexos. *São Carlos, SP: Universidade de São Paulo*, 2006.

[4] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM computing surveys (CSUR)*, 33(3):273–321, 2001.

[5] Tao Chen, Arif Khan, Markus Schneider, and Ganesh Viswanathan. iblob: Complex object management in databases through intelligent binary large objects. In *Objects and Databases: Third International Conference, ICOODB 2010, Frankfurt/Main, Germany, September 28-30, 2010. Proceedings 3*, pages 85–99. Springer, 2010.

[6] Daniel dos Santos Kaster. *Tratamento de condições especiais para busca por similaridade em bancos de dados complexos*. PhD thesis, Universidade de São Paulo, 2012.

[7] Ajit Kumar Mahapatra and Sitanath Biswas. Inverted indexes: Types and techniques. *International Journal of Computer Science Issues (IJCSI)*, 8(4):384, 2011.

[8] Joice Seleme Mota. Estendendo quadtress para suporte ao armazenamento e recuperação de dados espaço-temporais. 2000.

[9] Leonardo Fernandes Nascimento. A sociologia digital: um desafio para o século xxi. *Sociologias*, 18:216–241, 2016.

[10] Willian Dener de Oliveira. *Operação de busca exata aos K-vizinhos mais próximos reversos em espaços métricos*. PhD thesis, Universidade de São Paulo, 2010.

[11] Dimitris Papadias, Panos Kalnis, Jun Zhang, and Yufei Tao. Efficient olap operations in spatial data warehouses. In *Advances in Spatial and Temporal Databases: 7th International Symposium, SSTD 2001 Redondo Beach, CA, USA, July 12–15, 2001 Proceedings 7*, pages 443–459. Springer, 2001.

[12] Giulio Ermanno Pibiri and Rossano Venturini. Techniques for inverted index compression. *ACM Computing Surveys (CSUR)*, 53(6):1–36, 2020.

[13] Ayman Rabee and Imad Barhumi. Ecg signal classification using support vector machine based on wavelet multiresolution analysis. In *2012 11th International Conference*

on Information Science, Signal Processing and their Applications (ISSPA), pages 1319–1323. IEEE, 2012.

[14] Joao B Rocha-Junior, Orestis Gkorgkas, Simon Jonassen, and Kjetil Nørvåg. Efficient processing of top-k spatial keyword queries. In *Advances in Spatial and Temporal Databases: 12th International Symposium, SSTD 2011, Minneapolis, MN, USA, August 24-26, 2011, Proceedings 12*, pages 205–222. Springer, 2011.

[15] Leandro C Soares and Daniel S Kaster. cx-sim: A metric access method for similarity queries with additional conditions. *Journal of Information and Data Management*, 4(3):437–437, 2013.

[16] Caetano Traina Jr, Agma Traina, Bernhard Seeger, and Christos Faloutsos. Slim-trees: High performance metric trees minimizing overlap between nodes. In *International Conference on Extending Database Technology*, pages 51–65. Springer, 2000.

[17] Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity search: the metric space approach*, volume 32. Springer Science & Business Media, 2006.