Trabalho Prático 1
Busca Ortogonal em Bares e Restaurantes de BH com KD-Tree

Alessandro Mesa Teppa - 2023028366
Arthur Guimarães Ferreira - 2023034781
Leonardo Romano Andrade - 2023075151

Universidade Federal de Minas Gerais (UFMG)
Departamento de Ciência da Computação (DCC)
Algoritmos II (DCC207 - 2025/1)

1 Introdução

A cidade de Belo Horizonte é amplamente conhecida no Brasil como a capital dos bares e restaurantes. Com mais de 10 mil estabelecimentos desse tipo, BH se torna um cenário ideal para a aplicação de algoritmos de geometria computacional na análise de dados geográficos.

Neste trabalho prático, nosso objetivo é implementar um sistema para consulta ortogonal desses estabelecimentos dentro de uma área retangular à escolha do usuário a partir de um mapa interativo.

A ferramenta, implementada em *Python* com a biblioteca *Dash Leaflet*, lê os dados dos bares e restaurantes a partir de um documento CSV e plota os pontos no mapa com base em sua latitude e longitude. Por serem muitos bares e restaurantes, os pontos plotados foram agrupados em *clusters*, que se ajustam dinamicamente à medida que o usuário se movimenta pelo mapa. Ao clicar em um *cluster* e em um ponto, as informações do estabelecimento aparecem como especificado no Enunciado do TP1, além da implementação extra que diz se aquele estabelecimento foi ou não participante do concurso Comida Di Buteco 2025 e uma descrição do prato concorrente. **Há também uma ferramenta extra no canto superior esquerdo que filtra quais foram os restaurantes participantes do CDB2025** para manter uma visualização clara de seus pontos.

O algoritmo de busca funciona a partir de uma KD-Tree de 2 dimensões, construída com os pontos listados no arquivo CSV. A árvore é construída ao iniciar o programa e, quando um retângulo é desenhado pelo usuário, uma busca por pontos dentro de seus limites é iniciada, e apenas aqueles que estão dentro são plotados. Ademais, uma tabela com informações de cada restaurante aparece abaixo do mapa. Quando não há retângulos desenhados, os dados que aparecem na tabela são dos estabelecimentos que aparecem dentro do mapa de acordo com o zoom do usuário. Quando há retângulos, apenas os dados dos pontos dentro dos limites são exibidos.

2 Criação da Tabela de Bares e Restaurantes de BH (CSV)

Conforme instruído pelo professor e pelo Enunciado do TP1, a tabela contendo as informações necessárias de todos os bares e restaurantes de BH foi montada a partir da tabela de Atividades Econômicas criada pela Prefeitura de Belo Horizonte, do dia 01/04/2025. Disponível em: https://dados.pbh.gov.br/dataset/atividades-economicas1.
O processo de tratamento desses dados foi dividido em duas partes principais, que são descritas a seguir.

2.1 Seleção dos Dados e Adequação das Coordenadas Geográficas

A tabela `20250401_atividade_economica.csv` é imensa e possui diversas informações que não precisamos para nosso programa de bares e restaurantes. Portanto, a primeira ação que tomamos foi reduzir drasticamente esse documento CSV para que ele incluísse apenas as seguintes categorias de estabelecimentos (colunas `DESCRICAO_CNAE_PRINCIPAL` e `CNAE`):

- `RESTAURANTES E SIMILARES`
- `BARES E OUTROS ESTABELECIMENTOS ESPECIALIZADOS EM SERVIR BEBIDAS, SEM ENTRETENIMENTO`
- `BARES E OUTROS ESTABELECIMENTOS ESPECIALIZADOS EM SERVIR BEBIDAS, COM ENTRETENIMENTO`

Dessa forma, foi possível reduzir o número de linhas de estabelecimentos no documento CSV de 534 mil linhas para 13 mil linhas. O número de colunas também foi reduzido, pois era necessário manter somente `ID_ATIV_ECON_ESTABELECIMENTO, DESCRICAO_CNAE_PRINCIPAL, DATA_INICIO_ATIVIDADE, IND_POSSUI_ALVARA, NOME_LOGRADOURO, NUMERO_IMOVEL, COMPLEMENTO, NOME_BAIRRO, NOME, NOME_FANTASIA, GEOMETRIA`.

Além disso, foi necessário realizar uma conversão nos dados das coordenadas na coluna `GEOMETRIA`. Notamos que essa coluna trazia as coordenadas dos pontos no formato de coordenadas projetadas (valores altos de UTM). Como o mapa no *Dash Leaflet* funciona apenas com o padrão EPSG:4326 (lon/lat), rodamos um script que percorre toda a tabela, detecta quais pontos não estão no intervalo de -180 a 180 / -90 a 90 e reprojeta esses valores de SIRGAS/UTM (EPSG:31983) para graus decimais. No final, cada geometria ficou no formato uniforme "POINT (lon lat)", garantindo que todos os marcadores apareçam corretamente no mapa.

2.2 Cruzamento com os Dados do Concurso Comida Di Buteco 2025

Para implementação da funcionalidade extra, que diz se determinado restaurante fez parte ou não do concurso CDB2025, foi necessário realizar um cruzamento de dados de forma manual. O site do Comida Di Buteco (https://comidadibuteco.com.br/) não disponibiliza os dados dos bares e restaurantes competidores em uma tabela e também não padroniza o nome dos concorrentes com base nos dados da Prefeitura. Portanto, qualquer tentativa de automatização seria muito difícil e o cruzamento dos dados manualmente dos 124 concorrentes de BH foi a melhor alternativa.

Já a parte de colocar a descrição dos pratos na tabela foi parcialmente automatizada, mas ainda assim exigiu alguns ajustes manuais.

Ao realizar essa tarefa, foi possível verificar diversas inconsistências entre os dados da Prefeitura e os dados do concurso, como: diferenças no nome dos restaurantes, diferenças de endereço, bares ou restaurantes classificados como 'lanchonete' ou 'sociedade empresária limitada', bares não encontrados etc.

No fim, foi possível cruzar os dados com 114 dos 124 concorrentes, em uma nova coluna definida como `CDB2025_Participante`, que recebe os valores 'Sim' e 'Não', além das colunas `PRATO` e `DESC_PRATO`, que contêm os dados sobre os pratos.

3 KD-Tree

A estrutura de dados principal do nosso programa de busca ortogonal é uma KD-Tree de 2 dimensões. Essa estrutura é muito eficiente para organizar pontos em um espaço multidimensional, que no nosso caso é um espaço 2D definido pela latitude e longitude dos estabelecimentos. Sua grande vantagem é a capacidade de responder a buscas por região de forma muito mais rápida que uma verificação exaustiva de todos os pontos.

A implementação consiste em três partes principais, que estão no módulo `KDTree.py`:

1. Definição do nó
2. Construção da árvore
3. Algoritmo de busca

3.1 Estrutura do Nó (KDNode)

Cada nó da árvore representa um ponto no mapa de BH e uma divisão no espaço geográfico, gerada a partir de sua latitude ou longitude. Em nosso programa, a classe KDNode armazena:

- `point`: uma lista no formato `[latitude, longitude]` extraída da coluna `GEOMETRIA` da tabela CSV.
- `index`: o índice numérico que o estabelecimento ocupa quando a tabela é lida pelo Pandas. Ele é a chave para, após a busca, recuperar todas as outras informações do estabelecimento (nome, endereço, etc.) de forma rápida.
- `left` e `right`: as referências para os nós filhos da esquerda e da direita.
- `axis`: o eixo de divisão (0 para latitude, 1 para longitude) que aquele nó usa para particionar o espaço.

3.2 Construção da Árvore (build_kdtree)

No momento em que o programa é iniciado, essa função é chamada e constrói toda a KD-Tree de acordo com as coordenadas da tabela de bares e restaurantes.

Primeiramente, na função `main`, uma lista de pontos é preparada, com os atributos `([lat, lon], idx)` para a construção dos nós. Na chamada da função, essa lista é passada como parâmetro e, em seguida, a função define qual será o `axis` daquele nó, ou seja, se seus filhos serão definidos pela latitude ou longitude.

Após essa etapa, a lista de pontos é ordenada de acordo com o `axis` escolhido, o que permite a escolha do ponto do nó a ser retornado (que é a mediana da lista) e a chamada recursiva adequada para os nós da esquerda e da direita.

3.3 Algoritmo de Busca Ortogonal (range_search)

A função de busca, amplamente utilizada no programa, é simples e recebe como parâmetros: um nó, uma área de busca (retângulo) e uma lista a ser atualizada.

A primeira chamada da função ocorre sempre com o nó raiz e, a partir dela, ocorre a busca recursiva, que segue a seguinte lógica: se o nó for nulo, a recursão para naquele galho. Caso contrário, a função executa dois passos principais para cada nó visitado:

1. **Verificação do Ponto Atual:** o algoritmo primeiro verifica se o ponto (latitude e longitude) armazenado no próprio nó está dentro das fronteiras do retângulo de busca. Se estiver, seu índice é adicionado à lista de resultados para posterior exibição na tabela e no mapa.

2. **Decisão de Poda e Recursão:** em seguida, o algoritmo decide se precisa explorar as subárvores. Ele compara a linha de divisão do nó (seja ela uma linha de latitude ou longitude) com as bordas do retângulo de busca. A função só é chamada recursivamente para um lado

(esquerdo ou direito) se a região do mapa daquela subárvore tiver alguma interseção com o retângulo de busca. Isso evita, de forma extremamente eficiente, a verificação de pontos em áreas do mapa que não são de interesse.

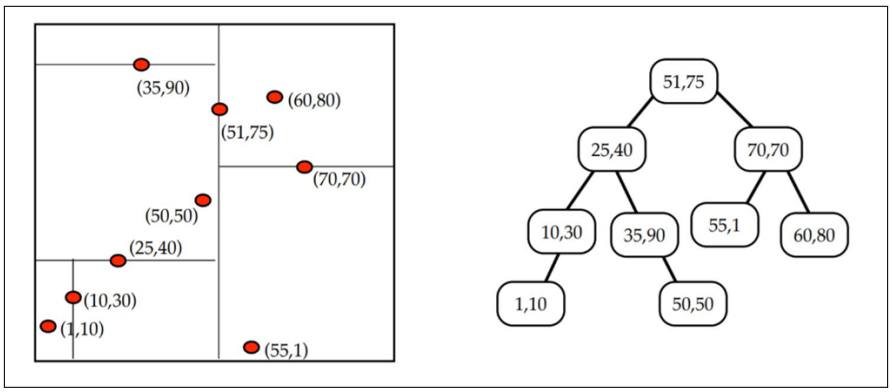


Figura 1: Exemplo de uma KD-Tree 2D

4 O Programa

O programa, criado em *Python* e com uso da biblioteca *Dash Leaflet*, é simples. Além das funções de `build_kdtree` e `range_search`, presentes no módulo explicado na seção 3, o algoritmo é dividido em mais quatro funções, que são explicadas a seguir.

4.1 Função Main

A `main` é a principal função do programa, que cria o app e o coloca para rodar no navegador web. Sua execução ocorre em duas etapas:

1. **Leitura de Dados e Construção da KD-Tree:** a primeira etapa é a preparação dos dados. O programa carrega o arquivo CSV com as informações dos estabelecimentos e o arquivo GeoJSON com o contorno geográfico de Belo Horizonte. Em seguida, as coordenadas de cada bar e restaurante são extraídas e utilizadas para construir, em memória, a árvore de busca. Essa estrutura de dados é o que permite a realização de buscas geográficas de forma eficiente.
2. **Configuração da Interface do Usuário (Layout):** a segunda etapa define todos os elementos visuais da aplicação. O layout é composto por um mapa interativo (`dash-leaflet`) que exibe o contorno da cidade e uma ferramenta que permite ao usuário desenhar retângulos de seleção. Abaixo do mapa, é inserida uma tabela (`dash_table.DataTable`), que é populada dinamicamente com as informações dos estabelecimentos selecionados.

Ademais, para lidar com o grande volume de dados de forma eficiente e garantir a fluidez da visualização, os marcadores dos estabelecimentos são agrupados em **clusters**. Essa técnica agrega pontos geograficamente próximos em um único círculo numerado. Conforme o usuário aumenta o zoom em uma região ou clica em um círculo, esses clusters se expandem, revelando os pontos individuais. Isso evita a sobrecarga visual e de processamento que ocorreria ao tentar renderizar milhares de marcadores simultaneamente, evitando que o programa trave. Obs: os clusters foram configurados com "$radius$" = 100, para gerar menor poluição visual, e, **ao clicar em um cluster, seus pontos são expandidos e podem ser consultados individualmente.**

4.2 Função de Callback (update_visible_markers)

Para fazer a plotagem de pontos de forma eficiente, sem que o programa trave devido ao grande número de estabelecimentos, a função `update_visible_markers` é usada pelo programa para plotar no mapa e na tabela somente os pontos de interesse, que são agrupados em clusters,

como descrito acima. Ela é acionada sempre que o usuário move o mapa, altera o zoom, ativa o filtro do CDB2025 ou desenha um retângulo, garantindo que os dados exibidos estejam sempre sincronizados com a interação do usuário. Seu funcionamento também pode ser descrito em duas etapas:

1. **Filtragem de Dados com a KD-Tree:** a primeira etapa, a mais crucial, é determinar quais estabelecimentos devem ser exibidos no mapa e na tabela. A função implementa uma lógica condicional para realizar a consulta ortogonal:

- **Caso 1:** se o usuário utilizou a ferramenta de desenho para criar uma ou mais áreas retangulares no mapa, a função extrai as coordenadas de cada um desses retângulos. Cada conjunto de coordenadas define uma caixa de busca (um `rect`) que é passada para a função `range_search`. Essa função consulta a KD-Tree pré-construída e retorna, de forma extremamente eficiente, os índices de todos os pontos contidos naquela área, que são plotados no mapa e têm suas informações exibidas na tabela.

- **Caso 2:** caso não haja nenhum retângulo desenhado, o sistema utiliza as fronteiras da área atualmente visível no mapa (`map.bounds`) como a região de busca. Da mesma forma que no cenário anterior, essas fronteiras definem um `rect` que é usado para consultar a KD-Tree, filtrando os estabelecimentos que aparecem na tela do usuário, tanto no mapa quanto na tabela.

- **Funcionalidade extra de filtragem CDB2025:** após determinar quais índices caem dentro da área de busca (seja pelo retângulo desenhado ou pelos limites visíveis), o callback verifica o valor de `CDB2025_Participante` para cada ponto. Se o usuário tiver ativado o filtro `cdb_filter` ("cdb"), o código descarta automaticamente todas as entradas cujo campo foi marcado como "Não", mantendo apenas aqueles estabelecimentos que participaram do Comida Di Buteco 2025. Dessa forma, o mapa e a tabela exibem só os pontos participantes quando o filtro está ligado, sem alterar a lógica de busca espacial e proporcionando uma visualização limpa dos restaurantes concorrentes e seus pratos.

Ao final desta etapa, o programa possui uma lista de índices (`filtered_indices`) que representa exatamente os pontos de interesse.

2. **Geração das Saídas para o Mapa e a Tabela:** com a lista de índices em mãos, a segunda etapa é preparar esses dados para serem visualizados. A função itera sobre cada índice filtrado e realiza o seguinte:

- **Criação dos Marcadores (Markers):** para cada estabelecimento, é criado um dicionário contendo suas coordenadas de latitude e longitude e um popup. Esse popup é uma pequena janela de informações que aparece quando o usuário clica em um pino no mapa, contendo o **nome, endereço, data de início das atividades, status do alvará e se o local participou do festival Comida di Buteco 2025**. Caso ele tenha participado, uma **descrição adicional sobre seu prato é exibida**.

- **Criação dos Dados da Tabela:** simultaneamente, é criado um outro dicionário com as mesmas informações, mas formatado especificamente para as colunas da tabela.

Ao final, a lista de marcadores é convertida para o formato GeoJSON e, junto com a lista de dados da tabela, é retornada pelo callback. O *Dash* atualiza o mapa e a tabela na interface do usuário com esses novos dados.

4.3 Função de Conversão de Coordenadas (parse_point)

A `parse_point` é uma função de pré-processamento, usada na parte de leitura de dados, essencial para preparar os dados geográficos para o uso na aplicação. Sua única responsabilidade é converter as coordenadas, que no arquivo de dados original vêm no formato de texto WKT (Well-Known Text) como, por exemplo, "POINT (-43.9386 -19.9191)", para um formato numérico que possa ser utilizado pela KD-Tree.

Primeiramente, ela recebe a string WKT como entrada. Então, ela remove os caracteres não numéricos (`"POINT ("` e `")"`) e a string resultante contém a latitude e a longitude do ponto, separadas por espaço. Por fim, os valores são convertidos para *float* e retornados em uma lista `[latitude, longitude]`, que é o padrão utilizado no programa.

4.4 Função de Abertura do Navegador (open_browser)

A função `open_browser` é uma pequena conveniência que melhora a experiência de quem executa o programa.

Sua única finalidade é abrir automaticamente o navegador web padrão do usuário e navegar para o endereço no localhost (`http://127.0.0.1:8050/`) onde a aplicação Dash está sendo executada.

5 Análise de Complexidade (KD-Tree)

O uso da KD-Tree no programa o torna muito mais eficiente do que seria caso utilizássemos a abordagem de força bruta para plotagem dinâmica dos pontos. Sem a KD-Tree, teríamos um custo de tempo e espaço para busca de $O(n)$, sendo n o número de pontos, e não seria necessária a construção de nenhuma estrutura adicional. Apesar do custo linear, para um grande número de pontos, como no nosso caso, essa abordagem se mostrou extremamente ineficiente com alguns testes: qualquer movimento no mapa gerava um grande delay para a plotagem de pontos.

A utilização da KD-Tree exige sua construção a partir da tabela de pontos com um custo maior que linear. No entanto, uma vez construída, ela é capaz de melhorar muito a fluidez do programa por conta de sua menor complexidade de busca para realizar a plotagem de pontos.

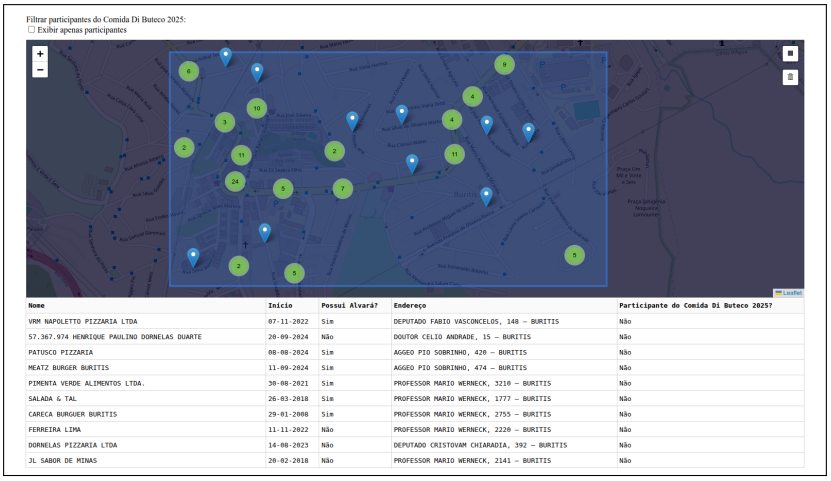
- **Construção da KD-Tree:** em cada nível de profundidade é feita uma ordenação dos n pontos para encontrar a mediana, o que custa $O(n \log n)$, e como a árvore balanceada tem profundidade $O(\log n)$, o tempo total de construção fica

$$O(n \log n) \times O(\log n) \;=\; O(n \log^2 n).$$

  A complexidade de espaço permanece $O(n)$, com adicional de $O(\log n)$ na pilha de recursão.

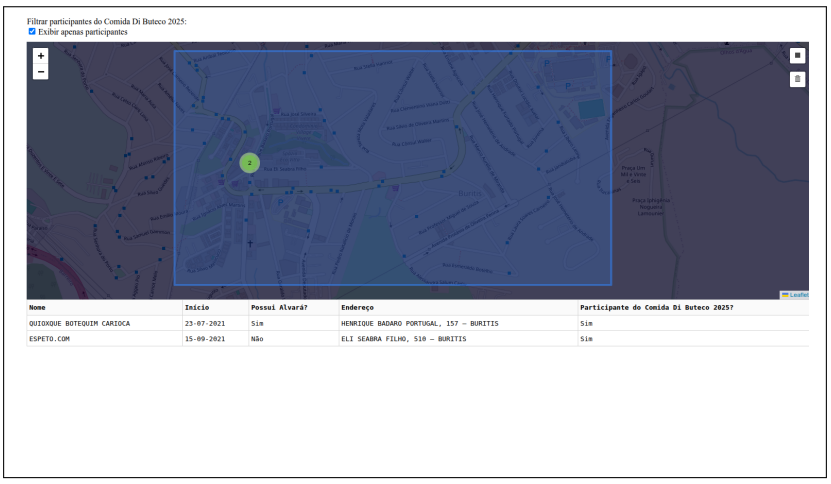
- **Busca**: em uma árvore balanceada, como em nosso programa, a busca na árvore é muito eficiente. A cada passo, ela descarta aproximadamente metade dos pontos, levando a um tempo de execução logarítmico $O(logn)$. A cada nível que a busca percorre, ela compara as bordas do retângulo de busca com a linha de divisão (seja de latitude ou longitude) do nó atual. Se o retângulo de busca estiver totalmente contido em um dos lados da linha, a árvore ignora completamente a subárvore do outro lado, descartando um grande número de pontos de uma só vez. A busca só continua em uma subárvore se a região do retângulo se sobrepuser à região representada por aquela subárvore.

6 Exemplos



Nome	Início	Possui Alvará?	Endereço	Participante do Comida Di Buteco 2025?
VRM NAPOLETTO PIZZARIA LTDA	07-11-2022	Sim	DEPUTADO FABIO VASCONCELOS, 148 – BURITIS	Não
57.367.974 HENRIQUE PAULINO DORNELAS DUARTE	20-09-2024	Não	DOUTOR CELIO ANDRADE, 15 – BURITIS	Não
PATUSCO PIZZARIA	08-08-2024	Sim	AGGEO PIO SOBRINHO, 420 – BURITIS	Não
MEATZ BURGER BURITIS	11-09-2024	Sim	AGGEO PIO SOBRINHO, 474 – BURITIS	Não
PIMENTA VERDE ALIMENTOS LTDA.	30-08-2021	Sim	PROFESSOR MARIO WERNECK, 3210 – BURITIS	Não
SALADA & TAL	26-03-2018	Sim	PROFESSOR MARIO WERNECK, 1777 – BURITIS	Não
CARECA BURGUER BURITIS	29-01-2008	Sim	PROFESSOR MARIO WERNECK, 2755 – BURITIS	Não
FERREIRA LIMA	11-11-2022	Não	PROFESSOR MARIO WERNECK, 2220 – BURITIS	Não
DORNELAS PIZZARIA LTDA	14-08-2023	Não	DEPUTADO CRISTOVAM CHIARADIA, 392 – BURITIS	Não
JL SABOR DE MINAS	20-02-2018	Não	PROFESSOR MARIO WERNECK, 2141 – BURITIS	Não

Figura 2: área de seleção aplicada em área no bairro Buritis, com filtro CDB2025 desativado.



Nome	Início	Possui Alvará?	Endereço	Participante do Comida Di Buteco 2025?
QUIOXQUE BOTEQUIM CARIOCA	23-07-2021	Sim	HENRIQUE BADARO PORTUGAL, 157 – BURITIS	Sim
ESPETO.COM	15-09-2021	Não	ELI SEABRA FILHO, 510 – BURITIS	Sim

Figura 3: mesma área da Figura 2, dessa vez com o filtro CDB2025 ativado.

Figura 4: popup com dados do restaurante Bom Sabor, incluindo seu prato concorrente no Comida Di Buteco 2025.