Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Curso de Graduação em Ciência da Computação

Trabalho Prático 3 - Estrutura de Dados
Estações de Recarga da BiUAIDi

Lucas Affonso Pires

Matrícula: 2023028420

Belo Horizonte, Agosto de 2024

1 Introdução

Como proposta do trabalho, foi selecionado a seguinte situação: Modernizar o aplicativo da fabricante de carros elétricos BiUAIDi, que identifica as 10 estações de recarga mais próximas ao usuário para que este possa reabastecer seu carro. Inicialmente o aplicativo é implementado de maneira bem rudimentar e pouco útil, não sendo capaz de adicionar e remover novas estações de reabastecimento, identificar quais estações estão ou não ocupadas e atualizar a posição do usuário. Assim, o novo programa implementado deve ser capaz de realizar as tarefas identificadas acima, além de utilizar o algoritmo conhecido como QuadTree, responsável por fazer com que o programa seja mais eficiente e tenha uma complexidade de execução menor que a do algoritmo anteriormente utilizado.

2 Método

O programa foi implementado visando unicamente resolver o problema acima, e, para isso, foram utilizadas algumas bibliotecas padrão do C++, como `<iostream>` para entrada e saída de dados, `<cmath>` para cálculos matemáticos, `<fstream>` para operações de leitura e escrita em arquivos, `<string>` para auxiliar na saída do programa e a `<iomanip>` para auxiliar na precisão das coordenadas. Como a base para a implementação do problema é o algoritmo QuadTree, ele será resumidamente explicado separadamente do resto do código, destacando sua importância.

- Algoritmo QuadTree:

  O algoritmo QuadTree é o algoritmo utilizado para melhorar a consulta do aplicativo verificando a distância do usuário para as estações mais próximas deles de maneira mais eficiente. A QuadTree é uma árvore que divide a região do espaço que estamos utilizando em quatro áreas e, por sua vez, cada área dessa pode ser divida recursivamente em até mais quatro de acordo com o interesse que temos na sub-região (nesse caso, se temos uma estação de recarga ou não no local). No aplicativo original, a distância era calculada utilizando vetores, que, primeirante, armazenavam em um vetor a distância do usuário para todos os pontos de recarga e então ordenavam-os em um segundo vetor para verificar quais são os mais próximos, já no aplicativo modernizado, a QuadTree rapidamente elimina grandes áreas em que as estações de recarga não estão próximas, evitando a comparação com todas as possíveis estações e somente verificando as mais próximas, melhorando bastante a eficiência do aplicativo. Quanto à complexidade, ela será detalhada de maneira melhor na seção de análise de complexidade, porém é claro que o uso da QuadTree é muito mais eficiente que o uso original de vetores.

Como método de implementação do restante do código, foram criados estruturas e funções simples, que obedecem o enunciado proposto para o trabalho. Abaixo, será comen-

tado o TAD (Tipo abstrato de dados) utilizado e cada uma das funções, structs e classes nele contido, de maneira breve, já que o próprio código possui comentários que auxiliam o entendimento.

- QuadTree:

  O TAD da QuadTree é responsável por todo a lógica do programa, controlando as operações de consulta, ativação e desativação; lendo os arquivos utilizados para gerar a saída do programa, além de, obviamente, conter todo o funcionamento da QuadTree para calcular a distância das estações mais próximas e retorná-las. Por questões de organização, todas as classes utilizadas foram declaradas dentro desse TAD e serão identificadas abaixo juntamente com sua utilizadade.

  - Classe QuadTree: Responsável pelo controle do algoritmo QuadTree, anteriormente explicado
  - Struct QuadTreeNode: Responsável por referenciar as estações.
  - Struct Estacao: Responsável pelos atributos existente nas estações de recarga, como localização, ID, status e etc.
  - Struct EstacaoDist: Responsável por atribuir a distância da estação ao usuário.
  - Classe EstacaoDistVetor: Responsável por armazenar os elementos de EstacaoDist.

  Quanto as funções serão destacadas a seguir:

  - QuadTreeNode(double x, double y, Estacao *estacao): Construtor que inicializa um nó da QuadTree com as coordenadas `x`, `y` e uma estação de recarga associada.
  - QuadTree(): Construtor que inicializa a QuadTree com a raiz como nula.
  - void inserir(double x, double y, Estacao *estacao): Insere uma nova estação na QuadTree, posicionando-a no quadrante correto com base em suas coordenadas.
  - Estacao* buscar_estacao_por_id(QuadTreeNode *no, const std::string &id): Busca uma estação na QuadTree utilizando seu identificador único, retornando um ponteiro para a estação encontrada.
  - void consulta_proximos(QuadTreeNode *no, double x, double y, EstacaoDistVetor &resultado): Consulta as estações mais próximas de um ponto dado, calculando a distância euclidiana e armazenando os resultados em um vetor de distâncias.
  - void destruir_quadtree(QuadTreeNode *no): Libera a memória alocada para a QuadTree, destruindo todos os nós da estrutura.
  - EstacaoDistVetor(): Construtor que inicializa o vetor dinâmico para armazenar as estações e suas respectivas distâncias.

- ~EstacaoDistVetor(): Destrutor que libera a memória alocada para o vetor dinâmico.
- void push_back(const EstacaoDist& value): Adiciona um novo elemento ao vetor, redimensionando-o automaticamente se necessário.
- EstacaoDist* comeco(): Retorna um ponteiro para o primeiro elemento do vetor.
- EstacaoDist* fim(): Retorna um ponteiro para o último elemento do vetor.
- std::size gettamanho() const: Retorna o número de elementos atualmente armazenados no vetor.
- EstacaoDist operator[](std::sizet index): Permite acessar elementos do vetor pelo índice.

3 Análise de Complexidade

A análise de complexidade do aplicativo foi relizada baseando-se no tempo e espaço utilizado por ele para cada tipo de implementação diferente (tanto a original quanto a modernizada). Adiante, serão analizadas as complexidades de tempo e espaço para ambas as versões do aplicativo:

- Aplicativo original:

  Analisaremos primeiro a complexidade de espaço. Para a implementação original, verificamos que a utilização de vetores para armazenar as estações de recarga tem complexidade $O(n)$, visto que todos os pontos de recarga n devem ser adicionados ao vetor.

  Analisando agora a complexidade de tempo. Verificamos que a utilização dos vetores faz com que tenhamos que comparar a distância do usuário com todas as estações de recarga n, além de ter que realizar uma ordenação após encontrar as distâncias para que tenhamos as mais próximas em $logn$, com isso, a complexidade temporal do aplicativo original é $O(nlogn)$.

- Aplicativo mordernizado com QuadTree:

  Analisaremos primeiro a complexidade de espaço. Perceba que para o algoritmo QuadTree, a complexidade espacial tem ordem idêntica à complexidade espacial do aplicativo que utiliza vetores, pois, apesar de poder ignorar áreas no cálculo da distância, todos os pontos de recarga ainda precisam ser armazenados . Desse modo, a complexidade espacial é $O(n)$, porém como temos que armazenar a estrutura responsável pela divisão do espaço no algoritmo, a complexidade espacial é ligeiramente maior que na implementação por vetores. Para situações onde a memória é extremamente limitada, talvez a implementação por vetores, apesar de pior, possa ser necessária.

Analisando agora a complexidade de tempo. Perceba que realizando as subdivisões do espaço e diminuindo o número de comparações da distância do usuário, o aplicativo utilizando a QuadTree é capaz de realizar $logn$ operações de comparação, transformando a complexidade temporal em $O(logn)$ para a maioria dos casos. Note que, se temos um caso extremo em que o espaço contém todos os pontos de interesse localizados muito próximos uns aos outros, a complexidade nesse pior caso pode chegar a se aproximar de $O(n)$, visto que todos os pontos próximos terão que ser verificados.

Partindo apenas da complexidade dada, vemos que a implementação por meio da QuadTree é definitivamente mais eficiente, porém, como destacado acima, em casos onde a memória é extremamente limitada, a ponto de ser necessário a utilização de um algoritmo muito simples, ou em que as estações estão todas próximas umas as outras, a implementação por vetores pode acabar se mostrando interessante (no caso em que estamos trabalhando, nenhuma dessas possibilidades ocorre, porém é importante destacar que não há uma hegemonia completa da utilização da QuadTree em relação aos vetores).

4 Estratégias de Robutez

Entre algumas das estratégias de robustez utilizadas estão: aquelas que evitam o uso desnecessário de memória e o uso desnecessário de tempo na execução, como o uso de construtores e destrutores; o encapsulamento de estruturas específicas para a execução do algoritmo, fazendo, assim, com que a resolução de possíveis problemas se torne mais simples e localizada; a utilização de condições de parada na execução dos algoritmos, garantindo que o programa finalize no momento correto; utilização de tratamentos de exceções, principalmente para a entrada de dados que não estão de acordo com o padrão fornecido no enunciado, utilizando verificações para caso os dados do arquivo base não estiverem corretos. De modo geral, as estratégias de robustez que foram escolhidas tem como foco garantir o bom funcionamento do programa, tanto evitando casos de borda, como melhorando o gerenciamento de tempo e espaço.

5 Análise experimental

Para a análise experimental iremos comparar os tempos de execução do aplicativo modernizado e do aplicativo original para regiões com diferentes distribuições das estações de recarga, além de discutir brevemente e localidade de referência de ambas as versões do aplicativo.

Vamos começar comparando as localidades de referência:

- Usando QuadTree:

  Localidade Espacial: Ao acessar um ponto em uma QuadTree, o algoritmo navega de uma região maior para sub-regiões menores até encontrar o ponto desejado. Isso

significa que, ao processar dados, o algoritmo pode acessar regiões de memória não contínuas, especialmente se os dados estiverem dispersos no espaço. Consequentemente, a localidade espacial pode ser comprometida, pois a estrutura da QuadTree dispersa os dados na memória de acordo com a região onde estão localizados os pontos de interesse.

Localidade Temporal: A localidade temporal pode ser limitada em uma QuadTree, pois, embora o acesso a um nó específico possa ser repetido, a dispersão dos dados no espaço (e, por extensão, na memória) significa que o algoritmo pode não reutilizar dados próximos com tanta frequência, especialmente conforme o ponto onde se encontra o usuário varia.

- Usando Vetores:

  Localidade Espacial: Vetores armazenam dados continuamente na memória, o que resulta em uma excelente localidade espacial. Ao iterar sobre os elementos de um vetor, o algoritmo acessa locais adjacentes na memória, aproveitando o cache de maneira eficiente. Isso geralmente resulta em melhor desempenho em termos de acesso à memória, especialmente quando comparado a estrutura da QuadTree.

  Localidade Temporal: A localidade temporal em vetores também pode ser boa, especialmente se o algoritmo faz múltiplas passagens pelos mesmos dados ou se reutiliza dados armazenados em posições próximas. A linearidade do vetor facilita o acesso repetido aos mesmos elementos, e como no aplicativo original, a posição que está o usuário não varia, a localidade de referência do aplicativo original é melhor que a do aplicativo modernizado.

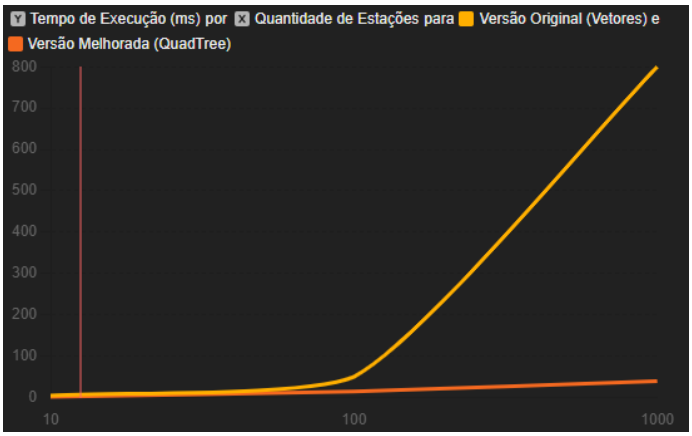

Figura 1: Comparativo das versões modernizada e original.

Por meio da análise da Figura 1, percebemos como o aplicativo modernizado é mais

eficiente na consulta conforme o número de estações totais da região aumentam, justamente por conta da complexidade ser menor e evitar comparações com todas as estações, enquanto o aplitivo original aumenta o número de comparações conforme o número de estações cresce.

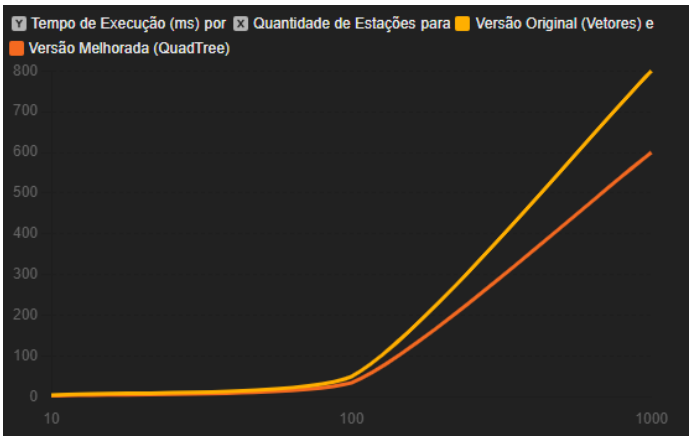

Figura 2: Comparativo das versões, porém com estações muito próximas.

Observando essa Figura 2, vemos que conforme as estações são colocadas muito próximas umas as outras, aumentando a densidade de onde a QuadTree deve verificar a proximidade, seu tempo de execução começa a se aproximar do aplicativo original com vetores, já que ele começa a verificar um número de estações próximos ao total n, mas, ainda assim, a versão modernizada é mais rápida, pois o aplicativo com vetores mantém a complexidade $O(nlogn)$ independentemente de como as estações são distribuídas no espaço.

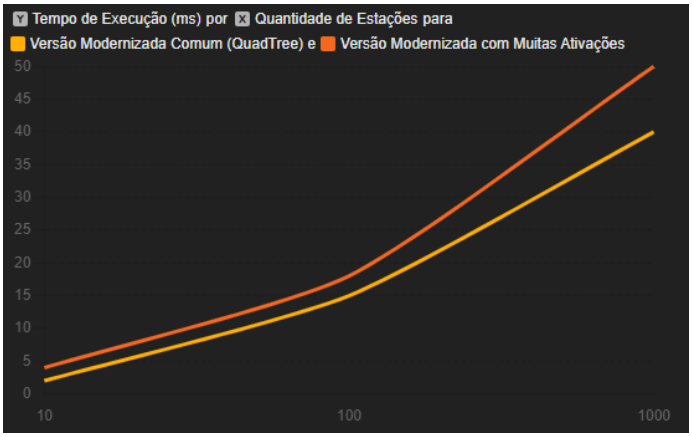

Figura 3: Aplicativo modernizado na consulta padrão e na consulta com muitas ativações.

Pela análise da Figura 3, vemos que conforme adicionamos muitas estações no espaço da região, o tempo de execução do aplicativo se torna mais lento, justamente porque precisa processar essas novas ativações para inserir as novas estações nas sub-regiões corretas, e, as vezes elas estão em locais mais próximos.

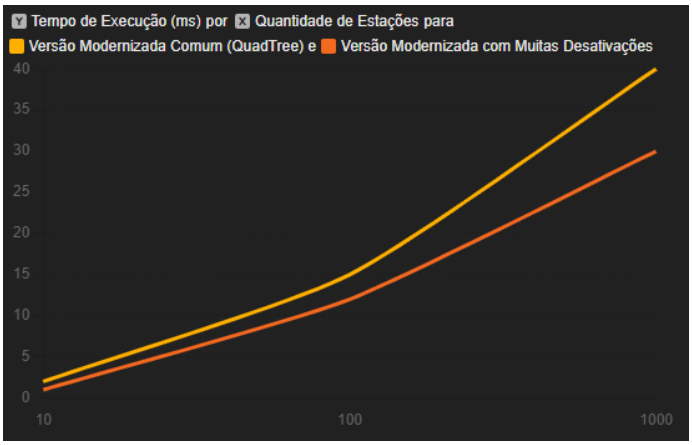

Figura 4: Aplicativo modernizado na consulta padrão e na consulta com muitas desativações.

Pela análise da Figura 4, vemos que conforme desativamos muitas estações no espaço da região, o tempo de execução do aplicativo se torna mais rápido, justamente porque ocasionalmente, uma estação que está mais próxima é desativada, diminuindo o tempo de consulta.

6 Conclusões

Revisando o que foi feito no trabalho, o aplicativo da BiUAIDi foi modernizado para que seja mais eficiente em suas consultas, além de executar novas tarefas de adição e remoção das estações de recarga, além de indicar se estão ou não ocupadas e atualizar a posição do usuário.

Implementado o aplicativo por meio da utilização da QuadTree podemos chegar a algumas conclusões. O uso da QuadTree, com toda certeza, melhora bastante a eficiência do aplicativo. Observando somente a complexidade temporal, vemos que o uso da QuadTree melhora por um fator de complexidade temporal n o aplicativo, porém, é necessário salientar que o uso da QuadTree nem sempre é melhor que o uso de vetores, já que em casos onde a memória é extremamente limitada ou o conjunto de regiões de interesse está muito próximo, a utilização de vetores pode ser interessante. Além disso, no quesito localidade de referência o uso de vetores se mostra melhor, justamente pela linearidade e facilidade

de predição dessa estrutura, diferentemente da QuadTree, onde os acessos não são lineares e podem variar bastante conforme os dados.

Concluindo, a modernização do aplicativo definitivamente mostrou uma melhora na experiência do usuário. Como observado na análise experimental, o tempo de execução diminuiu bastante em relação ao aplicativo original e também ampliou as possibilidades de verificação das estações mais próximas, tornando a consulta muito mais dinâmica.

7 Bibliografia

Chaimowicz, L. and Prates, R. (2020). Slides virtuais da disciplina de estruturas de dados.

`https://en.wikipedia.org/wiki/Quadtree`

`https://www.youtube.com/watch?v=OJxEcs0w_kE`