Universidade Federal de Lavras

# Trabalho Prático Final

GCC 218 - Algoritmos em Grafos

Prof. Mayron Cesar O. Moreira

Integrantes:

Gabriel Moraes de Oliveira
Guilherme dos Santos Cordeiro

# Link do repositório no GitHub

https://github.com/gabriel-moraes03/Projeto-Grafos.git

# Introdução

O projeto busca encontrar a melhor solução para o problema das paradas de ônibus, que pode ser visto como uma variação do Problema do Caixeiro Viajante. No problema original, o objetivo é minimizar a distância total do trajeto. Já no problema em questão, além de garantir que todos os vértices sejam percorridos formando um ciclo hamiltoniano, busca-se minimizar a distância entre dois vértices consecutivos.

# Formulação

A formulação do projeto teve início com a procura por uma estrutura de dados adequada para o armazenamento dos pontos e modelagem da solução. Uma ideia inicial que se manteve durante todo o desenvolvimento foi o uso de um registro do tipo “ponto”, que contém um inteiro “id”, que armazena o identificador do ponto, dois números de ponto flutuante com precisão dupla para armazenar a latitude e a longitude de um ponto, além de um booleano para indicar se determinado ponto já foi visitado ou não.

Em um primeiro momento, a estrutura de dados utilizada foi a KD-Tree, devido à sua capacidade de se trabalhar com múltiplas dimensões. Como cada ponto tem uma latitude e longitude, essa abordagem parecia promissora.

No entanto, após testes iniciais, o código baseado na KD-Tree se mostrou pouco eficaz. Apesar de uma busca por um vizinho mais próximo extremamente veloz, os resultados obtidos estavam muito acima dos valores de referência. Além disso, a equipe possuía pouca afinidade com essa estrutura, o que dificultava seu aprimoramento. Diante desses fatores, optou-se por construir uma nova solução baseada em outras estruturas de dados.

Em uma segunda tentativa, a resolução do problema foi baseada no algoritmo de Inserção Mais Próximo, visto em sala de aula durante os estudos sobre grafos hamiltonianos. A solução final foi baseada nesse algoritmo pois, o Vizinho Mais Próximo por considerar somente um vértice como comparação a cada iteração pode levar a resultados globais ruins, problema também compartilhado pelo algoritmo Guloso. Já como o Inserção Mais Próximo considera não

só o ponto atual, mas a inserção de novo caminho na melhor posição do ciclo, ou seja, naquela posição que vão gerar as menores arestas, ele se mostrava mais promissor.

Quanto à estrutura de dados escolhida, chegou-se a conclusão que utilizar simples vetores para armazenar os pontos e, posteriormente, vetores de tuplas para armazenar o caminho percorrido e a distância entre os pontos já seria suficiente para a resolução do problema.

# Descrição da solução

- Primeiramente, foi criado um registro que armazena as informações do ponto, sendo elas seu id, latitude, longitude e se já foi visitado.

- Em segundo lugar, temos duas funções para encontrar a distância entre dois pontos. Nos primeiros testes, esse detalhe passou despercebido: a existência de dois tipos de coordenadas para os pontos, geográficas (GEO) e euclidianas (EUC_2D). Identificado o tipo a partir da leitura das instâncias, temos uma função que recebe dois pontos como parâmetros e retorna a distância entre eles a partir da fórmula de Haversine (para o caso de pontos geográficos), e uma função que retorna a distância de dois pontos, também passados como parâmetros, porém, a distância cartesiana desses dois pontos.

- Em seguida, temos a construção da matriz de distâncias. Essa função recebe como parâmetro um vetor dinâmico de pontos, que armazena as informações das paradas de ônibus, além de um string denominada “tipo_de_coordenada”, que contém a informação se os pontos estão com sua posição descrita por coordenadas geográficas ou cartesianas, sendo utilizada em uma condicional que define qual função para encontrar a distância entre dois pontos será utilizada. A matriz armazena a distância entre todos os pontos do grafo. Apesar do gasto extra de memória, reduziu em aproximadamente ⅓ o tempo computacional do algoritmo, por evitar o cálculo de distâncias que já foram calculadas. Desse modo, impactou positivamente no projeto.

- Em seguida, temos a implementação da função “insercao_mais_proxima”, responsável por construir um ciclo que visita todos os pontos com base na estratégia de inserção mais próxima. Inicialmente, o algoritmo seleciona um ponto de partida arbitrário, insere-o no ciclo e o marca como visitado. No código feito esse ponto de partida é o ponto inicial do arquivo de entrada, porém, como encontramos um ciclo, o ponto de partida não interfere no resultado final e pode ser arbitrário. Em seguida, busca o ponto mais próximo desse ponto inicial e o adiciona ao ciclo. A partir desse momento, para cada novo ponto a ser inserido, o algoritmo percorre os pontos ainda não visitados e identifica aquele que possui a menor distância para algum dos pontos já presentes no ciclo. Depois de escolhido o ponto, ele é inserido na posição que minimiza o custo da inserção, garantindo a

construção de menores arestas. Esse processo se repete até que todos os pontos sejam visitados, resultando em um ciclo fechado que passa por todas as paradas de ônibus consideradas no problema.

- Por último, a função principal, que realiza uma leitura pela entrada padrão (cin), identifica a quantidade de vértices e o tipo das entradas dos vértices, realiza a chamada das funções auxiliares e, por último, imprime a sequência de visitação dos vértices e a distância máxima percorrida entre dois pontos.

# Resultados obtidos

## Configurações do computador utilizado:

- Processador: Ryzen 5 3500u
- Memória Ram: 12gb
- Armazenamento Interno: 1tb ssd NVME M2 e 1tb HD
- Placa Gráfica Integrada: AMD Radeon Integrada RX Vega 8

| Instâncias | Solução Inicial | Solução Final | Desvio (em relação ao valor inicial) | Desvio (em relação ao valor ótimo) | Tempo Final (em segundos) |
|---|---|---|---|---|---|
| 1 | 13788,084 | 3843,22 | 72,14051 | 3,63 | 2,17 |
| 2 | 19233,366 | 1573,93 | 85,6043 | 1,291 | 6,47 |
| 3 | 13988,36 | 1457,99 | 89,57712 | 1,2202 | 9,68 |
| 4 | 12258,66 | 1100,71 | 91,021 | 2,85 | 19,78 |
| 5 | 15872,22 | 458,053 | 97,114 | 16,1075 | 7,72 |
| 6 | 16793,72 | 431,067 | 97,433 | -0,0155 | 8,70 |

| | | | | | |
|---|---|---|---|---|---|
| 7 | 14011,524 | 85,4283 | 99,3903 | 60,9916 | 6,81 |
| 8 | 11873,43 | 214,021 | 98,12748 | 19,5409 | 4,62 |
| 9 | 13863,95 | 22,8035 | 99,8355 | 12,2942 | 1,70 |
| 10 | 12960,618 | 230,016 | 98,225 | 2,9468 | 14,56 |

# Conclusão

Neste trabalho, foi realizada a modelagem do problema de rotas do transporte urbano de uma cidade utilizando conceitos de grafos, visando otimizar as rotas para que o mesmo passe apenas uma vez por cada ponto e volte ao local inicial. A escolha da estrutura de dados vetor dinâmico foi essencial para garantir que os pontos fossem registrados da melhor forma e minimizar o gasto de recursos da linha de ônibus.

A implementação do algoritmo de inserção mais próxima junto ao fato do programa calcular todas as distâncias para todos os caminhos permitiu que o tempo computacional fosse reduzido em 1/3.

Dessa forma, este estudo contribuiu para o aprendizado e a aplicação de técnicas de otimização e redução de custo computacional, reforçando sua importância na solução de problemas reais, especialmente no contexto de manutenção de rotas de veículos.

# Bibliografia


CPLUSPLUS.COM. *C++ Reference*. Disponível em: https://cplusplus.com/. Acesso em: 29 jan. 2025.

VON WANGENHEIM, A. Árvore k-D. Disponível em: <https://www.inf.ufsc.br/~aldo.vw/estruturas/k-d/arvore-kd.html>. Acesso em: 29 jan. 2025.

REINELT, G. *TSPLIB 95*. Disponível em: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. Acesso em: 29 jan. 2025.

MAPARADAR. *Tópico do fórum sobre [Cálculo da Distância entre Pontos]*. Disponível em: https://forum.maparadar.com/viewtopic.php?t=5181. Acesso em: 29 jan. 2025.