Trabalho de Grafos

Integrantes: Elian Fernando Simões Costa | Clarisse Lacerda Pimentel

Introdução:

Uma empresa do ramo de transportes urbanos iniciou uma série de investimentos em ferramentas de apoio à decisão para a otimização das rotas de uma linha de ônibus. Essa linha possui n pontos aos quais o motorista deve realizar uma parada. Ao fim do trajeto, o motorista deve voltar ao ponto inicial. Cada ponto deve ser visitado exatamente uma vez no percurso.
Em geral, pretende-se a minimização do custo total do percurso. Todavia, a empresa resolveu inovar e para seu modelo de negócio, o mais importante é que a distância máxima entre dois pontos quaisquer do percurso seja minimizada.

Formulação:

O problema se trata de um grafo euleriano, não-direcionado (pois representa ruas) e ponderado. Os vértices são os pontos que o ônibus deve parar e as arestas são as ruas. O grafo remete ao problema do caixeiro viajante. A estrutura de dados a ser utilizada deve ser uma matriz de adjacência pois permite um acesso rápido ao índice (vértice) e isso é benéfico pois diminui o tempo da execução do algoritmo que resolve o problema do caixeiro viajante (NP-Difícil).

Solução:

A primeira coisa que devemos fazer é garantir que o grafo seja euleriano (é conexo e todos os vértices tem grau par, depois aplicar o algoritmo ou heurística mais adequada ao problema, para a primeira questão, todos os pontos são calculados entre si, ou seja, o grafo inicial é completo, portanto, existe um caminho euleriano. No caso do algoritmo de resolução TSP, devido a forma que o ocorre a escolha de quais vértices devem ser ligados, Vizinho Mais Próximo não é uma boa heurística pois pode aumentar o peso da maior aresta, especialmente para completar o caminho euleriano. Inserção Mais Próxima é uma boa ideia, começando com um ciclo de dois ou três vértices (ideal) e incluindo o vértice mais próximo do ciclo a solução, diminuindo o peso da maior aresta pois garante um emparelhamento melhor com o grafo, porém costuma ser mais

custoso, mesmo assim, é a melhor opção, especialmente pela complexidade de fácil implementação.

A ideia é receber os valores pelo terminal, a forma mais fácil de fazer isso, sem se preocupar em ficar alocando vetor dinamicamente é utilizando um vector, ou seja, uma lista, armazenando os pontos x e y através de uma struct, também armazenamos o id – como terá ordenação com heap, o id pode não ser preservado só com a ordem de inserção de dados em uma lista - depois calculamos a distância euclidiana entre cada ponto e salvamos em uma matriz, tornando o grafo completo. Em seguida aplicamos o algoritmo de Inserção Mais Próxima, para isso, primeiramente criamos uma lista de vértices visitados, para garantir que quando um vértice for incluído ao ciclo, durante o processamento, não seja visto novamente. Depois aplicamos uma procura pelo maior triangulo para iniciar o ciclo – isso permite que o algoritmo produza resultados mais satisfatórios, pois a busca inicial já será otimizada pela localidade do ciclo encontrado nessa função - e configuramos a heap: criamos uma struct chamada inserção, onde armazena o ponto, posição e custo do vértice a ser adicionado, também sobrecarregamos o método "<" para que o heap funcione como um minheap. Após isso, começamos o funcionamento do algoritmo, onde ele irá buscar um vértice que não foi visto, quando isso ocorre, ele calcula o custo de inserção desse vértice entre dois pontos do ciclo, pelo seguinte cálculo: `ciclo[(j+1)%ciclo.size()].` Cada inserção é colocada na minheap e o custo atual entre os dois pontos é atualizado com a inserção do terceiro. Depois ocorre a inserção de fato no ciclo, onde, através da inserção dinâmica onde devido a minheap, a inserção ocorre na melhor posição estimada mais promissora, não havendo necessidade de sempre recalcular para cada ciclo, para isso é utilizado um while e, enquanto n (quantidade de vértices) for maior que o tamanho do ciclo, é verificado se o vertice retirado da minheap já foi visitado – essa verificação ocorre novamente por problemas em implementações passadas que não consegui identificar – caso não, é inserido no ciclo, com a posição extraída anteriormente + 1, marca como visitado e calcula a maior aresta fazendo um max() do ponto atual inserido e da maior aresta. Depois desse processo, a minheap é atualizada pois precisa atualizar os novos custos, através de um for, percorre todos os pontos não visitados e calcula o custo de inseri-los na posição entre o ponto extraído no processo anterior e o próximo ponto (ponto anterior + 1) e insere esse novo ponto na minheap. Por último, cria-se uma lista para armazenar o caminho final usando os IDs dos pontos, ao invés de seus índices, copiando o ID dos pontos do ciclo na lista e fechando o ciclo adicionando o primeiro ponto de volta ao final do caminho, retorna o caminho. Depois disso o caminho é gravado e o valor da maior aresta é dado no terminal.

Resultados:

Instancia	Solução Inicial	Solução Final	Desvio Percentual SF x SI (%)	Desvio Percentual SF x SO	Tempo Computacional - segundos
1	3986	210	-94.73		1
2	1289	3964	207.52		12.807
3	1476	3829	159.41		26.7
4	1133	1115	-1.58		54.709
5	546	535	-2.01		17.702
6	431	1641	280.74		22.744
7	219	2313	956.16		15.5
8	266	1823	585.33		10.091
9	52	236	353.84		2.923
10	237	1476	522.78		36.542

Configurações do computador utilizado:

Processador AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz

RAM instalada 8,00 GB (utilizável: 5,88 GB)

ID do dispositivo C1AEB81B-799C-4B9A-A5CB-6446BC7E2386

ID do Produto 00327-30932-03484-AAOEM

Tipo de sistema Sistema operacional de 64 bits, processador baseado em x64

Explicações gerais da tabela: Não foi possível compreender qual seria a solução ótima, utilizei os dados de referência informados no documento de instrução do trabalho.

Análise do resultado:

O resultado do código foi variável, contando com resultados menores que o esperado em alguns casos, como na instancia 4 e 5, e com um resultado extremamente melhor na primeira instancia, porém, com um grande aumento em outras. Isso foi causado por causa de um sacrifício, o uso de uma heurística gulosa, ou seja, o algoritmo implementado não compara diversos resultados para escolher uma melhor, isso diminui o tempo de execução do algoritmo em troca de um resultado não muito satisfatório. O resultado só não foi maior em todos os casos pois, a inserção dinâmica

e a escolha de um ciclo inicial bom podem já reduzir bastante o custo da maior aresta em alguns casos, nesse quesito, a maior aresta sempre é a do último vértice com o primeiro.

Bibliografia do Código:

Este código foi desenvolvido utilizando conceitos matemáticos e algoritmos computacionais para resolver o Problema do Caixeiro Viajante (TSP) de forma heurística. A seguir estão as principais referências teóricas e práticas que embasam o programa:

1. Teoria Matemática e Algoritmos Relacionados

Problema do Caixeiro Viajante (TSP):

- O TSP é um problema clássico da Teoria dos Grafos, estudado amplamente na área de Otimização Combinatória. Consiste em encontrar o menor ciclo que visite todos os vértices de um grafo exatamente uma vez.
- Referência:
    - *Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). The MIT Press.*
        - Capítulo sobre algoritmos de grafos e problemas de otimização.

2. Geometria Computacional

Distância Euclidiana:

- A fórmula para calcular a distância entre dois pontos no plano cartesiano é derivada diretamente da fórmula pitagórica: d(p1 ,p2)=sqrt((x2 −x1)^2+(y2 −y1)^2)
- Referência:
    - *Preparata, F. P., & Shamos, M. I. (1985). Computational Geometry: An Introduction. Springer.*
        - Discussão sobre cálculos geométricos em algoritmos computacionais.

Cálculo da Área do Triângulo:

- A fórmula para calcular a área de um triângulo dado por três pontos no plano é baseada no determinante da matriz formada pelas coordenadas dos pontos:
$$\text{Área} = \frac{1}{2} |x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)|$$
- Referência:

- *O'Rourke, J. (1998). Computational Geometry in C (2nd ed.). Cambridge University Press*.
    - Aplicações geométricas no cálculo da área de triângulos.

3. Heurísticas para o TSP

Algoritmos de Inserção:

- A heurística de **inserção mais próxima** e suas variações são amplamente usadas para gerar soluções aproximadas do TSP.
- O algoritmo insere dinamicamente os vértices em posições que minimizam o custo incremental do ciclo.
- Referência:
    - *Gutin, G., & Punnen, A. P. (Eds.). (2002). The Traveling Salesman Problem and Its Variations. Springer*.
        - Seção sobre heurísticas de inserção e sua eficiência em problemas práticos.

Uso de Estruturas de Dados (Heap):

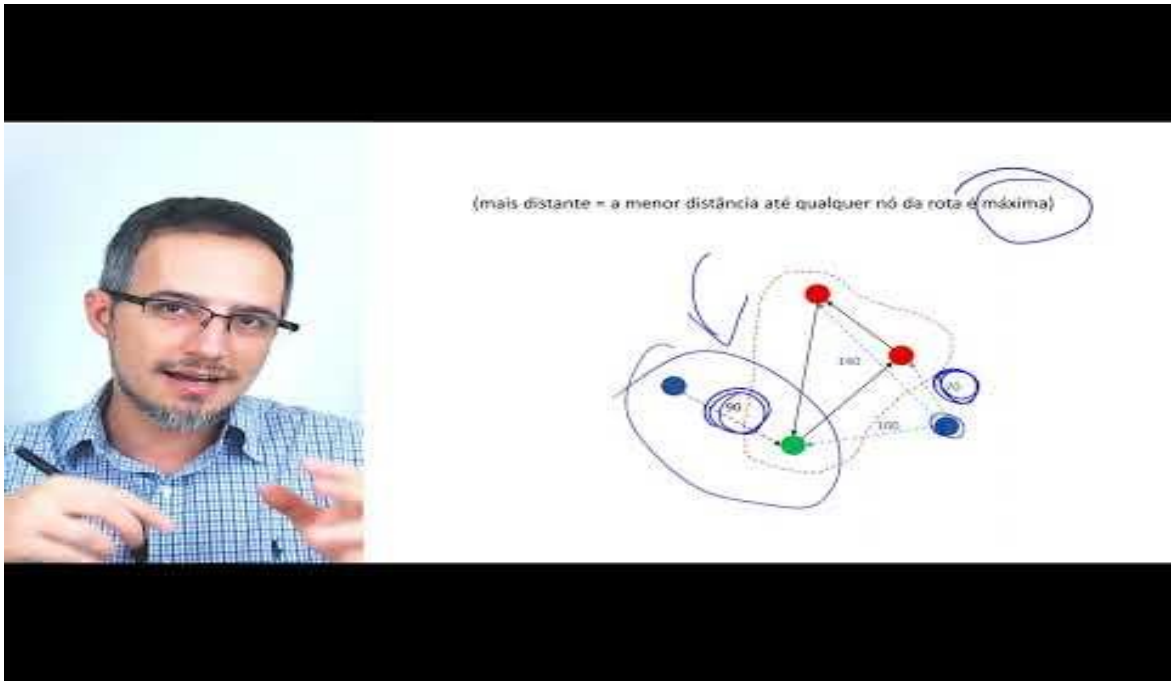
- A utilização de uma fila de prioridade (implementada como um heap) para organizar as inserções com base no menor custo reflete o uso eficiente de estruturas de dados em problemas de otimização.
- Referência:
    - *Sedgewick, R. (2011). Algorithms (4th ed.). Addison-Wesley*.
        - Discussão sobre heaps e filas de prioridade em algoritmos de otimização.

5. Contexto Histórico do Problema

- O TSP foi formulado no século XX e estudado em diferentes contextos, como logística, roteamento e bioinformática. A formulação inicial e as aplicações práticas são descritas em:
    - *Dantzig, G. B., Fulkerson, R., & Johnson, S. M. (1954). Solution of a Large-Scale Traveling-Salesman Problem. Journal of the Operations Research Society of America*.
        - Este artigo clássico introduz o TSP como um problema de otimização real e apresenta métodos matemáticos para resolvê-lo.

Alguns vídeos foram utilizados para a compreensão da heurística e formulação do algoritmo: 1 - https://www.youtube.com/watch?v=1k4fgcWPT7Y

2 - https://www.youtube.com/watch?v=YZ-2MD1M6PY



Conclusão:

O código não garante a solução ótima, mesmo assim, foi incrível a forma com que ele foi capaz de dar resultados tão distante um dos outros, contendo melhoras e pioras. O tempo de execução e a dificuldade de implementação foram a parte mais complexa de todo o trabalho, acredito que, caso fosse possível ter mais tempo para trabalhar com o código, poderia ocorrer algumas alterações que visariam melhorar os resultados,

possivelmente com alguma heurística combinatória com a ajuda de alguma alternativa de podar a busca, melhorando o tempo de execução que seria muito maior, de toda forma, foi uma ótima pesquisa prática para a compreensão de uma heurística que não estava acostumado. A conclusão do algoritmo utilizado é que não compensa o uso, visto que os prós de resultados não são tão melhores, e o tempo possivelmente não vai ser um bom pró para um resultado que pode chegar a ser 10x pior que o esperado.