Trabalho Prático II - Algoritmos I

Lucas Braz Rossetti
2020041590

Departamento de Ciência da Computação (DCC)
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

lucasbraz@ufmg.br

Modelagem Computacional

Em resumo, o problema apresentado envolve determinar rotas de transporte de mercadorias entre todas as lojas da empresa varejista apresentada, sendo necessário que o custo final de transporte seja mínimo, dado que o custo de um trajeto entre duas lojas é variável e depende de sua distância: há três opções de locomoção possíveis, todas com custos distintos. Além disso, cada trajeto é percorrido apenas uma vez e em direção única, mas não é necessário que todas as lojas estejam conectadas no mesmo dia de transporte, isto é, rotas que imponham direções opostas de viagem para percorrer todas as lojas são permitidas, uma vez que essas viagens podem ser feitas em dias distintos. Também é válido ressaltar que, para cada loja, há um trajeto direto único entre quaisquer outras lojas.

Dado o problema apresentado, a representação em grafos de uma possível solução parece adequada, dada a natureza geométrica de problemas que envolvem rotas de transporte. Assim, assuma um grafo não-dirigido e valorado $G = (V, E)$ onde cada vértice representa uma das lojas e cada aresta (u, v) representa um trajeto direto entre as lojas dos vértices u, v e possui um peso associado com o valor da distância entre os dois estabelecimentos. Além disso, é seguro concluir que, pela observação de cada loja estar conectada às outras por trajetos únicos, o grafo G é completo, disso temos que, se $|V| = n$, então $|E| = n^2 - n$.

Assim, a partir de G, podemos calcular a rota de distância mínima usando algoritmos de grafos conhecidos. Nesse caso, tomando em consideração a discussão apresentada acima, é evidente que o problema se resume a encontrar uma **Árvore Geradora Mínima** , já que, pelo fato dos trajetos serem únicos, a rota final não deve possuir ciclos. Para isso, iremos usar uma versão quadrática do Algoritmo de Prim.

Mas ainda temos que o custo total depende do método de transporte escolhido, sendo que, dado um certo limite de distância K , trajetos que não ultrapassam esse valor são percorridos por motocicletas, que possuem um custo M por Km, enquanto trajetos com distâncias maiores que K são percorridos por caminhões, com custo C por Km rodado. Os dois meios de transporte apresentados não possuem limite de veículos, porém, o terceiro meio, viagem por drone, embora possua um custo de transporte *nulo,* é limitado pelo número de aparelhos disponíveis D. Assim, cada drone deve ser posicionado em uma loja, e um trajeto só pode ser percorrido pelos veículos aéreos se as duas lojas envolvidas possuem drones.

Então seja T uma árvore geradora mínima do grafo apresentado. Para determinar quais lojas devem receber os drones em prol da minimização do custo total, basta encontrar os trajetos com maiores distâncias e posicionar os drones nos vértices das lojas envolvidas, isto é, anular o custo das $D - 1$ arestas de maior peso de T.

Descrição da Implementação

O projeto foi dividido em duas headers , *“control.h”, “loja.h”* e dois arquivos fontes *“control.cpp”* e *“main.cpp”* , onde estão distribuídas duas classes: *Loja*, uma estrutura que representa os estabelecimentos e contém suas informações geográficas, e *Controller* , uma classe que realiza o tratamento dos dados, os organiza em uma estrutura de grafos (matriz de adjacência com os pesos nas entradas - distMatrix), e calcula uma árvore geradora mínima associada ao grafo de entrada.

A execução do programa passa por duas funções principais de uma instância da classe Controller: _recordData()_ , que recebe os dados de entrada e constrói a matriz de adjacência do grafo modelado; e, em seguida, _minCostTree()_, encarregada de construir a árvore geradora mínima usando uma versão adaptada do Algoritmo de Prim, além de calcular o custo total mínimo.

Sobre a função minCostTree(), três vetores são criados para auxiliar no cálculo da árvore geradora mínima: **isInTree ,** vetor de valores booleanos tal que isInTree[i] = true se e somente se o vértice de índice i já está presente na MST; **pred** , vetor cuja entrada pred[i] possui o índice do nó parente do vértice de índice i na MST (naturalmente, pred[raiz] não existe) e, por último, **cost** , onde cost[i] possui o valor da aresta de menor peso que conecta o cutset atual da MST e o nó de índice i, no início, como nenhuma aresta é alcançável, todos os vértices recebem um custo infinito , exceto pelo nó inicial escolhido (no caso o de índice 0).

Em seguida, |V| - 1 vértices cujas arestas possuem custo mínimo em relação ao cutset atual são incluídos na árvore juntamente com as respectivas arestas, determinada pelo vetor de parentes. Para cada vértice adicionado, os nós que ainda não estão incluídos na MST são percorridos e suas estimativas de custo mínimo são atualizadas a partir da nova árvore, juntamente com os seus respectivos parentes.

Por fim, as arestas da MST são dadas pelas distâncias entre cada vértice e seu parente, o vetor da árvore é ordenado e seus primeiros D-1 elementos recebem o valor 0, uma vez que precisamos anular o custo das arestas percorridas por drones. Finalmente, as arestas da árvore são percorridas e, se a distância for maior que o valor K especificado, o custo da aresta é computado a partir da taxa de caminhões, caso contrário, o valor de referência será a taxa das motocicletas.

Análise de Complexidade

Como foi dito anteriormente, a execução do programa é resumida em duas funções da classe Controller: recordData() e minCostTree(), portanto a complexidade final será:

$$T(main()) = T(recordData()) + T(minCostTree())$$

1) $T(recordData())$:

Após os dados de entrada serem lidos, o tamanho do vetor com as lojas e a matriz de adjacência são redefinidos para |V|.

Em seguida, são feitas |V| iterações para armazenar os dados de cada loja s, e, para cada uma delas, o vetor referente à s na matriz de adjacência tem seu tamanho redefinido para |V| e a distância de todas as lojas a s é calculado.

Assim, a complexidade final da função é

$$T(recordData()) = 2 * O(|V|) + O(|V|) * [O(|V|) + O(|V|)] = O(|V|^2)$$

2) $T(minCostTree())$:

De acordo com a definição do Algoritmo de Prim, devemos adicionar a aresta de menor custo que cruza o cutset da árvore. Esse processo é repetido |V| - 1 vezes, porém o custo de encontrar a aresta de custo mínimo será linear no algoritmo usado:

Temos os custos lineares iniciais no número de vértices para preparar o vetor isInTree e cost, que armazena os custos mínimos atuais para se alcançar cada vértice a partir do nosso cutset (que no início é apenas o vértice de índice 0).

Assim, como dito anteriormente, iteramos |V| - 1 vezes incluindo os vértices na MST e, para cada iteração, iteramos por todos os vértices para encontrar o de menor custo e, em seguida, atualizamos as informações do vértice parente e custo mínimo de alcance para cada nó.

Depois, temos a árvore determinada pelo percorrimento das arestas dos vértices em conjunção com os parentes, o ordenamento das arestas da árvore (|V| -1) usando std::sort (complexidade O(n*log(n))) e ainda a eliminação do custo das arestas percorridas por drones.

Por fim, a árvore é percorrida mais uma vez enquanto os custos de cada meio de transporte são computados. Portanto, considerando que o número de drones é estritamente menor que o número de lojas, a complexidade final da função será:

$$T(minCostTree()) = O(|V|) + (|V| - 1) * (O(|V|) + O(|V|)) + \\ + O((|V| - 1) * log((|V| - 1))) + O((|V| - 1))$$

$$T(minCostTree()) = O(|V|) + O(|V|^2) + O(|V|log|V|) = O(|V|^2)$$

Finalmente, podemos dizer que a complexidade total do programa será

$$T(main()) = T(recordData()) + T(minCostTree())$$

$$T(main()) = O(V|^2) + O(V|^2) = O(V|^2)$$