Faculdade de Engenharia da Universidade do Porto

Projeto Concepção e Análise de Algoritmos

Meat Wagons: transporte de prisioneiros

Ano letivo 2019/2020

Turma 7 Grupo 2

José Miguel Afonso Mações - up201806622@fe.up.pt

João Castro Pinto - up201806667@fe.up.pt

Afonso Maria Rebordão Caiado de Sousa - up201806789@fe.up.pt

Índice

1. Introdução **3**

2. Descrição sucinta do problema **4**
2.1 Considerando que só temos uma camioneta 4
2.2 Considerando as características das carrinhas 4
2.3 Considerando que temos transportes especializados 5

3. Formalização do problema **6**
3.1 Dados de entrada 6
3.2 Dados de saída 6
3.3 Restrições 7
3.2.1 Sobre os dados de entrada 7
3.2.2 Sobre os dados de saída 7
3.4 Função objetivo 7

4. Descrição da Solução **9**
4.1 Estrutura de Dados 9
4.1.1 Representação do Grafo (Mapa) 9
4.1.2 Gestor da empresa 10
4.2 Algoritmos 10
4.2.1 Análise de Conectividade 10
4.2.2 Cálculo de menor distância 11
4.2.3 Cálculo de caminho entre dois vértices 12
4.2.4 Cálculo de um caminho a passar por mais de dois vértices 12

5. Casos de Utilização **13**
5.1 Trajetos Regulares 13
5.2 Situações Especiais 13
5.2.1 Trajeto Regular, especificando uma ou mais estradas 13
5.2.2 Serviço extraordinário 13
5.2.3 Manutenção de um veículo 13

6 . Conclusão **14**

7. Referências Bibliográficas **15**

1. Introdução

Este trabalho é realizado no âmbito da cadeira de Concepção e Análise de Algoritmos, do segundo ano do Mestrado Integrado em Engenharia Informática e Computação. Realizando este trabalho, que incide maioritariamente sobre a matéria de grafos e algoritmos em grafos, estamos a treinar a nossa compreensão não só sobre esta matéria, mas também de outras práticas que vão ser utilizadas como a programação dinâmica, algoritmos gananciosos, de retrocesso e de divisão e conquista.

2. Descrição sucinta do problema

O transporte de prisioneiros é feito usando carrinhas e camionetas adaptadas para o serviço (informalmente chamado de “meat wagons”). Estes veículos têm a necessidade de serem altamente resistentes para garantir que os prisioneiros não fogem.

Um destes sistemas de transportes de prisioneiros pretende optimizar o percurso feito pelos veículos de forma a recolher e entregar os prisioneiros nos pontos de interesse de forma mais eficaz.

Este problema pode ser decomposto em três fases:

2.1 Considerando que só temos uma camioneta

Na fase inicial, vamos desprezar a capacidade da camioneta e com isso calculamos o percurso mais curto entre todos os pontos de recolha e entrega de prisioneiros. A carrinha começa e termina as suas viagens numa garagem, onde a empresa é sediada. Considera-se que todos os estabelecimentos que estão envolvidos com a atividade da empresa são pontos de recolha e de entrega de passageiros.

Regra geral, é mais importante a recolha de um prisioneiro que uma entrega. Isto porque, não queremos que um prisioneiro esteja demasiado tempo numa esquadra ou num tribunal, por exemplo.

Considera-se que enquanto a carrinha está em viagem, a empresa pode receber outro pedido de transporte de um ou mais prisioneiros, e que o trajeto terá que ser recalculado com base nas tarefas que tem a fazer.

Repare-se que é necessário que todos os pontos por onde a carrinha passa pertençam ao mesmo componente fortemente conexo do grafo. Se esta condição não se verificar não é possível calcular um trajeto entre todos os pontos em que a carrinha tem de passar. Portanto, sem analisar a conectividade do grafo pode acontecer que tentemos calcular um trajeto que seja impossível de calcular.

Repare-se que a ligação entre entre dois vértices do grafo pode ser alterada devido a obras na estrada.

2.2 Considerando as características das carrinhas

Numa segunda fase, passa-se a trabalhar com múltiplos veículos, tendo em conta as suas diferentes características. Assim, passa-se a ter que se dividir o trabalho dos diferentes veículos, tendo em conta a sua capacidade, ou seja, ter que verificar se uma carrinha tem espaço para os prisioneiros ou se será necessária uma nova carrinha. Mesmo que

uma carrinha não esteja cheia, pode fazer sentido enviar outra carrinha porque a zona onde irá operar pode ser muito diferente. Quando é recebido um pedido, é atribuído à carrinha que fizer com que a distância total percorrida por todas as carrinhas seja mínima.

Para além disso, surge a questão do combustível. Sempre que a carrinha recebe um novo pedido de recolha, é necessário verificar se o número de quilômetros restantes após a realização desse pedido são superiores à distância entre o local de entrega e a garagem da empresa, onde é sempre abastecido o depósito. Isto obriga também a que carrinha esteja vazia quando ocorre o abastecimento. Surge assim a possibilidade de haver pedidos impossíveis, devido ao consumo de combustível ser demasiado alto e nenhuma carrinha o conseguir realizar. *Idem* para a capacidade das carrinhas.

2.3 Considerando que temos transportes especializados

Numa terceira fase, passam a existir carrinhas e camionetas especializadas. As especializações podem ser úteis na repetição de certos trajetos (por exemplo, tribunal - aeroporto, esquadra A, B, C e depois passar na prisão), otimizando a gestão dos prisioneiros.

Considera-se também nesta fase outros tipos de especialização de veículos, não pelo trajeto em si, mas sim pelas zonas percorridas pela carrinha durante o trajeto. No caso, ter por exemplo veículos de dimensão menor para zonas com densidade populacional maior. Para diferentes tipos de carrinhas, pode fazer sentido haver outras garagens, como por exemplo para carrinhas de menor capacidade que operem numa zona de maior densidade populacional.

Como consequência das especializações, esta solução minimiza o tempo de espera de alguns prisioneiros em trajetos recorrentes, ao ter veículos prontos em linhas fixas. Por outro lado, minimiza-se eventuais problemas rodoviários.

Nesta iteração, passa-se a considerar situações especiais (ver ponto 5.2).

3. Formalização do problema

3.1 Dados de entrada

Wagons - conjunto de carrinhas para transportar prisioneiros. Atributos das carrinhas (atributos com * não são considerados na primeira iteração):

- position - Vértice onde a carrinha se encontra;
- *cap - número de prisioneiros que pode transportar ao mesmo tempo;
- *fuel - quantidade de combustível disponível;
- *maxFuel - máxima quantidade de combustível;
- *consumption - consumo médio de combustível aos 100 km.

G = (V, E) - Grafo dirigido pesado, composto por:

- Conjunto de Vértices V:
  - localType - especificação do tipo do local que o vértice representa: pode ser uma garagem da empresa, um estabelecimento de gestão de prisioneiros, uma oficina ou um local diferente;
  - P - número de prisioneiros a ser recolhidos deste vértice;
  - Pmax - número máximo de prisioneiros que podem estar neste local;
  - adj(E) - arestas adjacentes ao vértice.
- Conjunto de Arestas E:
  - dist - peso da aresta, representa a distância entre dois pontos do mapa;
  - dest - aresta de destino.

POI $\in$ V - Pontos de interesse do trajeto da carrinha.

3.2 Dados de saída

Wagonsf - sequência das carrinhas a utilizar (na 1ª iteração apenas contém uma carrinha), para além dos dados de entrada também são associadas ao seguinte objeto:

- Course - lista de arestas a visitar. Com cada aresta guarda-se o número de prisioneiros que a carrinha transporta Pcur.

3.3 Restrições

3.2.1 Sobre os dados de entrada

- Wagons:
    - cap > 0;
    - fuel > 0 && fuel < maxFuel;
    - maxFuel > 0;
    - consumption > 0;
- Vértices:
    - localType tem 4 instâncias diferentes, cada uma representativa de um local diferente. Pode representar uma garagem da empresa, um estabelecimento relacionado com prisioneiros, uma oficina ou outros locais não especificados;
    - P >= 0;
    - Pmax >= 0.
- Arestas:
    - dist >= 0;
    - dest != NULL.
- | POI | >= 0.

3.2.2 Sobre os dados de saída

- |Wagonsf| <= |Wagons|;
- Em cada objeto Course, para todas as arestas, Pcur < cap;
- Em cada objeto Course, os seus primeiro e último elementos devem ser vértices que representam locais onde existam garagens;
- Em cada objeto Course, sempre que uma carrinha está numa garagem ou oficina, aplica-se: Pcur = 0.

3.4 Função objetivo

O objetivo é a minimização da distância total percorrida pelas carrinhas de entrega e recolha de prisioneiros. Desta feita, apresentamos a função seguinte:

$$f = \sum_{wagons \in Wagonsf} \left(\sum_{e \in course} \text{dist}(e) \right)$$

Ao minimizar esta função estaríamos, numa situação real, a minimizar os custos da empresa na execução das suas tarefas.

4. Descrição da Solução

4.1 Estrutura de Dados

4.1.1 Representação do Grafo (Mapa)

Para a representação do Grafo em memória do programa há duas possibilidades a considerar:

1. Matriz de adjacências
2. Lista de adjacências

A matriz de adjacências aloca espaço para todas as potenciais arestas mesmo que algumas nunca vão existir. Assim, com esta opção haverá alocação de uma quantidade de memória proporcional ao quadrado do número de vértices do grafo [O(|V|^2)]. No entanto, esta representação tem a vantagem de o acesso, a adição e a remoção de uma aresta ser muito eficiente temporalmente [O(1)].

Por sua vez, a lista de adjacências apenas aloca memória para as arestas utilizadas. Portanto, a memória alocada por esta representação será algo proporcional ao número de vértices somado ao número de arestas [O(|V|+|E|)]. A desvantagem desta opção é que o acesso a uma aresta tem complexidade temporal linear [O(|V|)], mas mantém a eficiência de adição de uma aresta [O(1)]. A eficiência da operação de remoção de uma aresta também é afetada, passando para complexidade linear [O(|E|)].

Dada a natureza do problema, conhecendo o grafo que vamos inserir no nosso programa - o mapa de Portugal - como as ordens de grandeza do número de nós e do número de arestas são iguais e como são muito elevadas (o mapa de Portugal tem 1803214 nós e 1874296 arestas), a nossa prioridade vai ser “economizar” o máximo de memória possível.

Portanto, optamos pela segunda hipótese, a lista de adjacências, sacrificando parte da eficiência temporal por eficiência espacial. Note-se que nesta situação, a solução de utilizar uma matriz de adjacências para representar este grafo para um computador comum não seria viável, uma vez que teríamos de alocar espaço para 1803214*1803214 arestas. Isto resultaria em termos de alocar uma quantidade de memória na ordem dos terabytes, enquanto que pela lista de adjacências alocamos cerca de uma dezena de megabytes.

4.1.2 Gestor da empresa

No contexto do problema proposto, a aplicação será utilizada para obter os percursos que cada carrinha irá fazer. Para isso, será implementada uma interface, constituída por uma série de menus que permitirão uma navegação acessível pela aplicação. Através desta interface serão introduzidos os dados de entrada que permitirão o programa produzir os resultados esperados pelo utilizador.

Os dados da empresa vão ser guardados de uma forma orientada a objetos. Haverá um objeto central, o "sistema" que contém todos os outros dados, como por exemplo, as carrinhas. Por sua vez, as carrinhas e outros objetos mais específicos também vão conter os seus atributos.

4.2 Algoritmos

4.2.1 Análise de Conectividade

A conectividade do grafo assume extrema importância no contexto de qualquer problema que envolva determinação de caminhos ótimos, uma vez que a conectividade entre dois vértices determina se é possível chegar a um a partir do outro. Assim, a conectividade, em problemas que utilizam grafos para representar mapas de ruas, pode ser útil para simbolizar possíveis dificuldades reais de passagem entre dois pontos, como, por exemplo, obras na via pública. Assim, será necessário avaliar a conectividade do grafo para identificar possíveis locais com pouca acessibilidade, o que permitirá otimizar o cálculo de caminhos entre dois pontos.

Um grafo não dirigido diz-se conexo se uma pesquisa em profundidade (*depth-first search*) a começar em qualquer vértice visitar todos os vértices. Os vértices que, se removidos, tornam o grafo desconexo são apelidados de Pontos de Articulação.

No caso do problema em questão, o grafo é dirigido, pelo que a análise da conectividade passará por dividir o grafo em componentes fortemente conexos, componentes onde, de qualquer vértice, se pode chegar a qualquer outro. Para isso, fazer-se-á uma pesquisa em profundidade, que determinará múltiplas árvores de expansão (floresta de expansão), numerando também os vértices em pós-ordem (ordem inversa da numeração em pré-ordem). Depois, inverter-se-ão as arestas, resultando desta operação um novo grafo. Uma nova pesquisa em profundidade será feita sobre este novo grafo, começando pelo vértice de numeração mais alta ainda por visitar. As árvores de expansão resultantes indicarão os componentes fortemente conexos. Para este algoritmo resultar, o grafo não pode ser não conexo.

Estes componentes serão fundamentais na escolha de caminhos, por permitirem associar cada vértice ao componente em que se encontra, sabendo que, se ambos o vértice de partida e de chegada pertencerem ao mesmo componente, é obrigatoriamente possível chegar dum ao outro.

4.2.2 Cálculo de menor distância

Para tal, temos vários algoritmos a considerar, como por exemplo o Algoritmo de Dijkstra ou o algoritmo de Bellman-Ford. No nosso caso, ao estarmos a lidar com um grafo dirigido pesado (ver ponto 3), portanto vamos utilizar o algoritmo de Dijkstra.

O algoritmo de Dijkstra é aconselhado no nosso caso para calcular a menor distância entre dois pontos de interesse *s* e *t*, dando como terminado o algoritmo quando o próximo nó a processar é o nó *t* (explicação mais detalhada a seguir).

Este algoritmo começa então por marcar o ponto de partida como tendo distância 0, e todos os outros vértices do grafo (interseções no caso prático de uma rede viária) como tendo distância ∞ (infinita). Processa-se então de seguida os vértices com distância mínima e repetir o método sucessivamente. Neste caso, as arestas poderão representar as vias da estrada. Pode ser necessário rever a distância de um vértice já alcançado, que não tenha ainda sido processado. Ao termos um grafo dirigido pesado, a distância obtém-se somando o peso das arestas.

A cada iteração, novos vértices são processados, atualizando a distância mínima calculada (ou não) até alguns dos vértices que ainda não foram processados.

O algoritmo de Dijkstra é um algoritmo ganancioso, minimizando a distância a cada passo. O seu tempo de execução é $O((|V| + |E|) * log|V|)$, sendo | V | o número de vértices e | E | o número de arestas.

Uma otimização a este algoritmo é utilizar o Dijkstra bidirecional. Em relação ao algoritmo regular de Dijkstra, o algoritmo de pesquisa bidirecional efetua um número significativamente inferior de passos.

A pesquisa bidirecional consiste em realizar o algoritmo de Dijkstra no sentido de *s* para *t* (ver ponto 4.2.2), assim como no sentido de *t* para *s*, simultaneamente. A pesquisa unidirecional, até chegar do ao vértice t, irá ter pesquisado pelo vértice t numa área cerca **π***(d/2)^2, sendo d a distância entre o s e t. Por sua vez, a pesquisa bidirecional vai pesquisar uma área cerca de 2***π***(d/4)^2 até encontrar o vértice pretendido. Como a área pesquisada pelo Dijkstra bidirecional é metade da área pesquisada pelo Dijkstra unidirecional, estamos aproximadamente a duplicar a eficiência do algoritmo.

Com esta otimização, em vez de terminarmos a pesquisa quando o próximo vértice a

processar for o ponto de chegada *t*, o algoritmo termina quando processa um vértice *x* que já foi processado na outra direção.

4.2.3 Cálculo de caminho entre dois vértices

Para este algoritmo utilizamos o Dijkstra bidireccional descrito no ponto anterior. A única diferença é que para além de guardarmos a distância de um vértice a todos os outros, também guardamos os vértices que compõem os trajetos de um vértice a todos os outros. Quando o algoritmo termina, concatena-se os caminho do primeiro Dijkstra com o caminho invertido do segundo Dijkstra. Daí resulta o caminho entre dois vértices.

4.2.4 Cálculo de um caminho a passar por mais de dois vértices

Nos casos em que temos de calcular um caminho que passa por mais do que dois pontos de interesse, é necessário repetir o algoritmo de cálculo de um trajeto entre dois vértices apenas (ver ponto 4.2.3). Isto pode acontecer, por exemplo, quando uma carrinha é requisitada para recolher um prisioneiro do local A e entregá-lo no local B. Assumindo que inicialmente a carrinha está numa das garagens, o trajeto a calcular será: garagem -> local A -> local B -> garagem.

Portanto, vamos executar o algoritmo de cálculo de caminho entre dois vértices, apresentado no ponto acima (ver ponto 4.2.3), para cada “sub-trajeto” que seja necessário fazer. No exemplo dado acima, seria necessário executar o algoritmo 3 vezes, uma vez para o trajeto “garagem -> local A”, outra para “local A -> local B” e outra para “local B -> garagem”. Este exemplo apenas tem 3 pontos de interesse, mas poderia ter mais. Para um caminho com *n* pontos de interesse, é necessário calcular *n* caminhos que passam por dois vértices.

Assim, o cálculo entre um caminho a passar por mais de dois vértices vai depender do algoritmo para calcular o caminho entre dois vértices.

Note-se que as complexidades temporal e espacial são iguais às complexidades do cálculo de caminho entre dois vértices (ver ponto 4.2.3).

5. Casos de Utilização

5.1 Trajetos Regulares

A utilização mais comum da aplicação será a feita para obter o percurso ótimo para um serviço comum, ou seja, recolhas e entregas em certos locais do mapa. Normalmente, estes locais do mapa estão relacionadas com a gestão de prisioneiros.

Note-se que um local é representado por um vértice do grafo.

5.2 Situações Especiais

5.2.1 Trajeto Regular, especificando uma ou mais estradas

Uma situação de utilização extraordinária possível é o cálculo de um trajeto ótimo em que o utilizador especifica uma ou mais estradas que o trajeto tem de incluir ou de excluir. Repare-se que o que muda nesta situação é que em vez de trabalharmos apenas com "vértices de interesse", também consideramos "arestas de interesse".

Algoritmicamente, para a exclusão de uma aresta, retiramos a aresta temporariamente do grafo antes de executar o algoritmo, e voltamos a colocar a aresta no grafo após calcularmos o trajeto. Para a inclusão de uma aresta, basta considerar que os seus vértices de partida e destino são pontos de interesse, que o trajeto tem de conter.

Numa situação real, este cálculo especial pode ser interessante para obter um trajeto que evite portagens de auto estrada, por exemplo.

5.2.2 Serviço extraordinário

Um serviço especial é um serviço que pode ter como locais de recolha e entrega no mapa, vértices que não representam locais de gestão de prisioneiros.

Algoritmicamente, o cálculo do trajeto é feito normalmente.

Uma situação real em que este serviço pode ser útil, é um requerimento para transportar prisioneiros a fazer serviço comunitário.

5.2.3 Manutenção de um veículo

Também é possível que um veículo tenha a necessidade de ser levado a uma oficina para manutenção. Nesta situação, o veículo tem de ficar numa garagem até uma certa data prevista para quando a carrinha estará consertada.

Esta situação não requer alterações no cálculo do trajeto. A única diferença é que não adicionamos mais nenhum ponto de interesse a seguir à oficina.

Nota: É possível combinarem-se situações especiais

6 . Conclusão

O principal objetivo deste trabalho era desenvolver uma estratégia ótima para o transporte de prisioneiros com vários veículos entre diferentes pontos de interesse.

A nossa estratégia passava pela minimização do trajeto percorrido pelas carrinhas de transporte, que por consequente resulta numa minimização do dinheiro gasto.

Após estabelecermos os dados de entrada, saída, e as respectivas restrições, começámos por estabelecer uma solução, que passaria pela utilização de uma lista de adjacências, assim como a implementação do algoritmo bidirecional de Dijkstra.

De uma forma mais generalizada, foi uma primeira parte do projeto bem conseguida que nos deixou bem preparados para começar com o desenvolvimento de uma aplicação para a segunda parte.

O esforço foi bem distribuído pelos membros do grupo, tendo optado por dividir tarefas na maioria das vezes, exceto em alguns pontos em que consideramos que era necessária a atenção e opinião de todos em conjunto.

7. Referências Bibliográficas

- “Algoritmos em Grafos: Caminho mais curto (Parte I)”, R. Rossetti, L. Ferreira, H. L. Cardoso, F. Andrade. CAL, MIEIC, FEUP;
- “Algoritmos em Grafos: Caminho mais curto (Parte II)”, R. Rossetti, L. Ferreira, H. L. Cardoso, F. Andrade. CAL, MIEIC, FEUP;
- “Algoritmos em Grafos: Conectividade”, R. Rossetti, L. Ferreira, H. L. Cardoso, F. Andrade. CAL, MIEIC, FEUP;
- Fiset, W., 2019, *Algorithms Course - Graph Theory Tutorial from a Google Engineer*, vídeo, Youtube, visto 15 de Abril 2020, <https://youtu.be/09_LlHjoEiY>.