U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Projeto de Conceção e Análise de Algoritmos (CAL)

MIEIC

EatExpress: Entrega de Comida

Trabalho realizado por:

Turma 4 - Grupo 7 **24/04/2020**

Mariana Oliveira Ramos up201806869@fe.up.pt
Paulo Jorge Salgado Marinho Ribeiro up201806505@fe.up.pt
Rita Sofia Araújo Sá Lopes da Silva up201806527@fe.up.pt

Índice

Introdução

No âmbito da unidade curricular Conceção e Análise de Algoritmos (CAL) do Mestrado Integrado em Engenharia Informática e Computação (MIEIC), será desenvolvida uma aplicação de entrega de comida: EatExpress.

Esta aplicação irá processar pontos de partida, recolha e chegada, marcados num mapa de acordo com a localização dos utilizadores, restaurantes bem como dos estafetas.

A aplicação será desenvolvida em C++, com recurso à API GraphViewer para se desenhar o grafo gerado para os diferentes trajetos. Os mapas utilizados serão extraídos da plataforma OpenStreetMaps.

Neste documento será apresentada uma breve descrição do problema bem como a sua formalização e uma proposta de implementação.

Descrição do Tema

A EatExpress é um sistema de entrega de comida entre os restaurantes registados na plataforma e os utilizadores da sua aplicação. Acedendo à aplicação o utilizador pode escolher o restaurante e o prato que deseja. A refeição escolhida é, de seguida, entregue na morada do cliente por um estafeta que utiliza o seu próprio meio de transporte da empresa para lá chegar (a pé, bicicleta, mota ou carro).

Por ser uma rede com vários restaurantes e vários clientes, existem vários percursos que o estafeta pode fazer para entregar a comida ao cliente. Sendo, portanto, necessário avaliar várias hipóteses de forma a otimizar a distância e o tempo de viagem de cada entrega. Coloca-se ainda a hipótese de um cliente ser inacessível a partir de qualquer restaurante, o que requer uma análise de conetividade entre restaurantes e os utilizadores.

Aquando da conclusão do projeto deverá ser possível, para uma dada lista de restaurantes e utilizadores, fazer a distribuição das entregas através dos estafetas de forma otimizada em termos de distância e tempo.

Por forma a ter uma implementação incremental podemos subdividir o problema em variantes com diferentes graus de complexidade:

1. **Apenas existe um estafeta que realiza um pedido de cada vez**

Numa primeira fase considera-se que a plataforma EatExpress tem disponível apenas um estafeta que terá de realizar uma entrega. Nesta situação, o problema reduz-se a encontrar o trajeto mais curto, com início na posição do estafeta quando é realizado o pedido (esta posição é a morada do cliente onde este realizou o último pedido ou a "Casa dos Estafetas"), que passe pelo restaurante indicado, e termine na morada do cliente que efetuou o pedido. Ao estafeta é lhe associado um meio de transporte com uma velocidade média.

2. Apenas um estafeta que pode realizar múltiplos pedidos em simultâneo (carga ilimitada)

Nesta segunda fase, semelhante ao caso anterior, vamos considerar apenas um estafeta que irá realizar múltiplos pedidos em simultâneo. Isto é, um estafeta pode ser encarregue de um certo número de pedidos que terá de atender numa só deslocação. Assim este irá realizar um trajeto que passe por todos os restaurantes e moradas de clientes dos pedidos selecionados, garantindo sempre que o restaurante do cliente respetivo é atingido primeiro que este.

3. Múltiplos estafetas a atender os pedidos (carga ilimitada)

Nesta terceira fase consideramos, ao contrário das fases anteriores, múltiplos estafetas que estão a trabalhar para a aplicação. Assim cada estafeta atende um ou vários pedidos

4. Múltiplos estafetas a atender os pedidos (carga limitada)

Nesta quarta fase vamos assumir o problema na íntegra, ou seja, a aplicação tem ao seu dispor vários estafetas cada um com o seu meio de transporte com capacidade limitada. É importante salientar que, ao ter disponível diferentes estafetas, não implica que que todos tenham que estar a circular. Cada estafeta atende um ou vários pedidos diferentes, realizando o mesmo caminho que na primeira ou segunda etapas.

Nesta quarta versão do problema, o critério de escolha para um estafeta efetuar um pedido vai passar por selecionar aquele que, dentro dos que ainda tem capacidade de transporte disponível, se encontra mais perto do restaurante ao qual irá buscar o pedido.

Adicionalmente, em qualquer das versões do problema, é necessário antes de procurar definir qualquer percurso, a análise das características subjacentes aos dados. Por exemplo, a conetividade do grafo tem que ter sida em conta como já foi mencionado. Caso um destino de um cliente ou restaurante seja inacessível, esse pedido não deve fazer parte das possíveis rotas dos estafetas, e o utilizador deve ser notificado desta ocorrência. Estas situações serão analisadas num pré-processamento dos dados que será detalhado nos capítulos seguintes.

Formalização do Problema

A solução deste problema passa pela formalização do mesmo em grafos e resolução recorrendo a algoritmos com estas estruturas.

Dados de entrada

De modo a uniformizar a notação procedeu-se às seguintes definições:

G(V,E): grafo dirigido pesado com informação sobre a rede viária em causa. Composto por:

- **V**: conjunto de todos os nós de G, que representam pontos de interesse do mapa, como pontos de turismo, de cuidados de saúde, entre outros, sendo que os que nos interessam são os relacionados com gastronomia, nomeadamente restaurantes. Cada nó apresenta os seguintes atributos:
  - ➔ id - id do nó;
  - ➔ adj - conjunto contido em E, representa o conjunto de arestas (estradas que saem de um determinado nó);
- **E**: conjunto de todas as arestas de G, em que cada aresta representa o caminho entre dois pontos de interesse. Cada aresta apresenta os seguintes atributos:
  - ➔ orig - pertencente a V, representa o nó de partida da aresta;
  - ➔ dest - pertencente a V, representa o nó de chegada de E;
  - ➔ c - custo de percorrer a aresta (quer seja a distância, tempo, combustível, portagens, os algoritmos otimização para a propriedade correspondente);
  - ➔ unic - booleano que indica se uma rua é de sentido único ou não;

R: estrutura de todos os restaurantes inscritos na plataforma, tendo cada um:

- ➔ morada - morada do restaurante, pertencente a V;
- ➔ nome - nome do restaurante;
- ➔ descrição - breve descrição do restaurante e tipos de culinária;
- ➔ menu - conjunto dos pratos disponíveis em cada restaurante;

C: estrutura de todos os clientes registrados na EatExpress, tendo cada um:

- ➔ morada - morada do cliente, pertencente a V, para onde é enviado o pedido que este possa efetuar;
- ➔ nome - nome do cliente;
- ➔ nif - nif do cliente;

L: Estrutura com os diferentes pedidos, tendo cada um:

- ➔ C - cliente que efetuou o pedido;
- ➔ R - restaurante a que foi efetuado o pedido;
- ➔ order - conjunto de pratos pedidos;
- ➔ W - estafeta que vai efetuar o pedido;
- ➔ M - meio de transporte no qual o estafeta vai efetuar o pedido;

W: estrutura de todos os estafetas , tendo cada um:

- ➔ nome - nome do trabalhador na plataforma;
- ➔ nif - nif do trabalhador;
- ➔ pos - posição atual do estafeta;
- ➔ n_pedidos - número de pedidos efetuados por cada estafeta;
- ➔ d_total - número total de quilómetros efetuados por um estafeta;

M: conjunto de meios de transporte disponíveis na empresa, tendo cada um:

- ➔ vm - velocidade média do veículo;
- ➔ nome - nome do veículo;
- ➔ capacidade - capacidade total do veículo;

Dados de saída

Os dados de saída, isto é, dados obtidos a partir dos dados de entrada para uso no programa são:

Wf: estrutura de todos os estafetas, semelhante à estrutura inicial, tendo cada um:

- ➔ **P**: sequencia ordenada de nós (pertencentes a V) que constituem o percurso que define o caminho que o estafeta deve efetuar, sendo que parte da localização deste no início do pedido, passa pelo restaurante requisitado, e termina na morada do cliente que efetuou o pedido;
- ➔ **CP**: custo do percurso efetuado pelo estafeta;
- ➔ **T**: tempo que o estafeta leva a efetuar esse percurso no seu meio de transporte;

Funções Objetivo

O objetivo do programa dependerá da escolha do cliente relativa ao seu pedido, e pensamos em duas hipóteses de escolha:

- **Percurso mais barato** - O estafeta efetuará o percurso entre a sua posição inicial e a morada do cliente que efetuou o pedido, passando pelo restaurante requisitado, com menor custo monetário. (menor CP, isto é, min($\sum c$)).
- **Percurso menos demorado** - O estafeta efetuará o percurso entre a sua posição inicial e a morada do cliente que efetuou o pedido, passando pelo restaurante requisitado, com menor duração. (menor T, isto é, min($\sum t$), em que t depende da velocidade do meio de transporte utilizado, Vm, e da distância da aresta).

Assim a resolução dos vários problemas passará pela minimização da função f=C(nó início,nó fim), sendo C o critério escolhido.

A ter em atenção que selecionando a opção “percurso mais barato” e existindo vários percursos com custos mínimos iguais recorremos ao menos demorado dentro destes. O mesmo se reflete para a opção “percurso menos demorado”. Na possibilidade de existirem vários percursos com tempo mínimo igual optamos pelo mais barato.

Restrições

Nos dados de Entrada

∀v ∈ V, adj(e) ⊆ E, todas as arestas adjacentes a um nó fazem parte do conjunto de arestas do grafo;

∀e ∈ E, orig(e), dest(e) ⊆ V, para todas as arestas os seus nós de origem e destino pertencem ao conjunto de nós do grafo;

∀e ∈ E, c(e) > 0, não há ruas com custo 0 ou negativo, pois todas têm um certo comprimento ou tempo para atravessar;

∀r ∈ R, morada(r) ∈ V, todos os restaurantes tem a sua morada no conjunto de nós do grafo;

∀r ∈ R, menu(r).size > 0, todos os restaurantes tem pelo menos um prato no menu;

∀c ∈ C, morada(c) ∈ V, todos os clientes têm a sua morada no conjunto de nós do grafo;

∀l ∈ L, C(l) ∈ C, todos os pedidos correspondem a um cliente registrado na plataforma (pertencente ao conjunto de clientes C);

∀l ∈ L, R(l) ∈ R, todos os pedidos são feitos a um restaurante registrado na plataforma (pertencente ao conjunto de restaurantes R);

∀r ∈ R, order(r).size > 0, não é possível efetuar um pedido sem conjunto de pratos;

∀l ∈ L, W(l) ∈ W, todos os pedidos são feitos a um estafeta, trabalhador da plataforma (pertencente ao conjunto de estafetas W);

∀l ∈ L, M(l) ∈ M, todos os pedidos são efetuados num meio de transporte da EatExpress (pertencente ao conjunto de Meios de Transporte M);

∀w ∈ W, pos(w) ∈ V, as posições atuais dos diferentes estafetas correspondem a um nó do grafo.

∀m ∈ M, vm(m) > 0, não existem veículos com veículos com velocidades médias negativas;

O conjunto de de todos os pontos úteis, isto é, ponto de posição inicial do estafeta, morada do restaurante, morada do cliente, fazem todos parte de um mesmo componente fortemente conexo do grafo. Ou seja ∀v1,v2 ∈ Vn sendo P(v1,v2) a sequência ordenada de vértices do percurso que liga v1 a v2 então P(v1,v2)!={} e P(v2,v1)!={}. Existe sempre um caminho que ligue quaisquer dois pontos úteis um ao outro.

Nos dados de Saída

$P \subseteq V$, todos os pontos de P tem que ser vértices do grafo;

Seja P0 o primeiro elemento de P, P0 = pos(W) inicial (morada do cliente que efetuou o último pedido, ou no caso de ainda não ter efetuado nenhum pedido, ponto de interesse denominado “Casa dos Estafetas”), pois todos os percursos partem da posição do estafeta;

Seja PF o último elemento de P, PF = morada(C), pois todos os percursos terminam na morada de um cliente;

$\forall i,j\ (P(i) \in P \wedge P(j) \in P \wedge (ii+1=j) \Rightarrow \exists e \in adj(P(i))$,dest(e)=P(j) , isto é, para quaisquer dois vértices de P consecutivos, são adjacentes (têm ligação entre eles);

$CP > 0$;

$T > 0$;

Perspetiva de Solução

Técnicas de Conceção

Interpretação do Problema

Após uma análise do problema, solidificamos a nossa visão deste, destacando-se os tópicos:

- Um pedido é realizado por apenas um cliente, e atendido por um único estafeta. É constituído pelo prato(s) que foram pedidos, e por dois vértices relativos ao restaurante que o cliente desejou. Numa fase mais avançada, o estafeta terá também associado um meio de transporte com uma capacidade limitada e poderá efetuar vários pedidos em simultâneo.
- Cada estafeta terá associado um vértice correspondente à sua localização atual. Caso não tenha atendido ainda nenhum pedido, este vértice será o vértice relativo a um ponto inicial da localização dos estafetas, que designaremos “Casa dos Estafetas”. Caso contrário, o vértice associado ao estafeta será o vértice correspondente à morada do cliente a quem o estafeta atendeu o seu último pedido.
- Esse vértice da localização do estafeta será o ponto de partida do percurso deste ao atender um pedido, que deverá passar pelo restaurante associado ao pedido, e terminar no vértice correspondente à morada do cliente que efetuou o pedido.

Características do Grafo e Pré-processamento

O grafo que a nossa aplicação irá abordar será um grafo dirigido, tendo em conta que é uma representação de uma rede viária, pelo que as estradas poderão ter apenas um sentido. Será também um grafo pesado, uma vez que o peso das arestas não será unitário, mas variará conforme a distância ou o custo destas, conforme a escolha do cliente no pedido.

O pré-processamento dos dados terá como objetivo encontrar sempre um percurso para o estafeta, de modo a que todos os pedidos tenham sucesso. Para isto, teremos de tornar o grafo fortemente conexo, de modo a garantir que haverá sempre um caminho que ligue um qualquer ponto A a um qualquer ponto B, e um caminho que ligue esse ponto B ao ponto A.

Uma forma de o conseguirmos seria realizar uma **Depth-First-Search** partindo do vértice inicial, e assim poderíamos remover quaisquer vértices não visitados nessa DFS, que

correspondem a pontos de interesse que nunca seriam alcançáveis a partir da “Casa dos Estafetas”, ponto inicial da localização dos estafetas. Depois disto, faríamos uma DFS aos outros vértices, e terminaríamos a procura caso a “Casa dos Estafetas” fosse alcançada. Caso não fosse alcançada, podíamos remover o vértice, sendo que este não teria retorno para o ponto inicial. Desta forma, seriam removidos todos os vértices que não teriam pelo menos uma ligação à “Casa dos Estafetas”, quer de ida, quer de volta, isto é, nos dois sentidos. Isto seria condição suficiente para que qualquer estafeta tenha liberdade máxima de movimento e para que qualquer pedido tivesse sucesso, ou seja, fosse sempre encontrado um percurso para este, uma vez que , no pior dos casos, esse percurso fosse a conjunção do percurso que leva o vértice A à “Casa dos Estafetas” e do percurso que parte da “Casa dos Estafetas” para o vértice B. Logo, haveria sempre ligação entre dois quaisquer vértices do grafo.

Uma outra forma seria recorrendo ao Algoritmo de **Floyd-Warshall**, que pode ser relevante para este pré-processamento do mapa de estradas. A descrição deste algoritmo será mais aprofundada na temática dos Principais Algoritmos mas, resumidamente, tem como objetivo encontrar as menores distâncias entre todos os pares de vértices num grafo dirigido pesado. Estas distâncias estariam armazenadas numa matriz de adjacências W, em que W[v1][v2] conteria o valor da menor distância percorrida desde o vértice v1 até ao v2, e W[v2][v1] conteria o valor da menor distância percorrida desde o vértice v2 até ao v1 (sentidos diferentes). Tendo isto em conta, após correr este algoritmo, poderiam ser removidos todos os vértices v1 tal que W[v0][v1] = INF (valor muito elevado em cada célula da matriz no momento anterior ao algoritmo, exceto nas células da matriz em que v1=v2, nas quais seria atribuído o valor 0), uma vez que estes correspondem aos vértices que nunca seriam alcançáveis partindo do vértice correspondente à “Casa dos Estafetas”. Da mesma forma, poderiam ser eliminados aqueles vértices em que W[v1][v0] = INF, isto é, que não teriam qualquer percurso que partisse destes e terminasse na “Casa dos Estafetas”. Posto isto, teriam sido removidos todos os vértices que não tivessem pelo menos uma ligação à “Casa dos Estafetas”, quer de ida, quer de volta, isto é, nos dois sentidos, pelo que podemos afirmar que o grafo é agora fortemente convexo. Como explicado em cima, isto seria condição suficiente para que qualquer estafeta tenha liberdade máxima de movimento e para que qualquer pedido tivesse sucesso, ou seja, fosse sempre encontrado um percurso para este. Este algoritmo é, no entanto, demorado, tendo em conta a sua complexidade temporal, explicada na temática dos Principais Algoritmos.

Para não tornar este processo tão demorado, poderíamos aplicar primeiro o algoritmo de DFS, descrito anteriormente, tendo como vértice origem a “Casa dos Estafetas”. Assim, poderíamos logo eliminar os vértices que não foram alcançados a partir da origem, e que, portanto, nunca seriam alcançáveis pelos estafetas. Desta forma, reduzimos o número de

vértices que o algoritmo de Floyd-Warshall tem de tratar, e não seria necessária a primeira verificação (W[v0][v1] = INF), pois estes vértices já foram removidos na DFS.

Esta abordagem irá ser, provavelmente, a que será usada na Parte 2 do Projeto, com o acrescento de melhorias que nos poderão surgir durante a sua implementação.

Atendimento de um Pedido

Associado a um pedido está o cliente que o realizou, o estafeta que o atenderá, o restaurante em que será feito o levantamento dos pratos pedidos. Numa fase posterior, em que haverão vários estafetas, a atribuição do estafeta que atenderá o pedido será feita com base num critério ainda a escolher, mas que poderá ser o número de pedidos atendidos (é escolhido o que tiver um menor número), ou a distância por eles percorrida, (é escolhido o que tiver menor distância), por exemplo.

Numa fase mais avançada, a empresa terá também associado um conjunto de meios de transportes, e a cada pedido será também associado um meio de transporte, que será utilizado pelo estafeta, e ao qual está associado uma certa velocidade média. A escolha do meio de transporte a atribuir ao pedido cairá sobre aquele que terá maior velocidade, de entre os disponíveis.

O Atendimento de um Pedido por um estafeta subdivide-se em dois problemas de cálculo do melhor caminho entre dois pontos: o primeiro consiste em calcular o melhor caminho entre o ponto de partida do estafeta e o restaurante especificado no pedido, e o segundo em calcular o melhor caminho entre esse restaurante e o vértice correspondente à morada do cliente que realizou o pedido. Para obter o percurso completo realizado pelo estafeta, basta fazer a junção desses dois caminhos.

Principais Algoritmos

Algoritmo de Tratamento dos Dados

A preparação dos dados é realizada facilmente assumindo que já se tem numa estrutura C as moradas dos clientes que irão efetuar os diferentes pedidos e numa estrutura V os vértices do grafo, basta percorrer uma vez os clientes e adicioná-los ao respectivo vértice, atualizando a sua informação.

Algoritmo de Pesquisa em Profundidade

Um dos primeiros algoritmos que o programa executará será o Algoritmo de Pesquisa em Profundidade (Depth-First Search), isto se este for a opção escolhida para pré-processamento de modo a garantir que há sempre caminhos possíveis entre dois vértices, nos dois sentidos.

Resumidamente, este algoritmo procura percorrer todos os nós filhos do nó origem o mais profundo possível para só depois retroceder, pelo que a nossa pesquisa em profundidade segue uma política de visitar sempre os nós mais profundos primeiro. Abordaremos uma versão recursiva deste, uma vez que as arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda tenha arestas a sair dele.O nó origem pode ser qualquer nó do grafo, o que implica que esse nó será o nó inicial da busca.

No nosso caso, usaremos o nó correspondente à “Casa dos Estafetas” como origem, ou seja, a busca partirá deste nó. Há que ter em atenção que este algoritmo pode entrar em *loop* e nunca terminar a sua execução se for encontrado um ciclo. Para controlar isto, cada vértice tem como atributo um boleano, que indica se já foi ou não visitado durante a pesquisa, que começa a false em todos os vértices (exceto o nó origem), e se torna true à medida que exploramos cada nó. Assim, se tivermos alcançado um nó com esse boleano a true, saberemos que o grafo tem, pelo menos, um ciclo. No final do algoritmo, os vértices que não tiverem sido visitados pela pesquisa serão os que nunca poderão ser alcançados a partir do vértice origem. Neste contexto, estes vértices devem ser eliminados, tendo em conta que não poderão ser alcançados por nenhum estafeta, uma vez que estes partem, num ponto inicial, da “Casa dos Estafetas”, ponto utilizado como origem da pesquisa.

Como o algoritmo precisa, no pior caso, de percorrer todos os vértices e todas as arestas, a complexidade temporal é O(|V| + |A|). Porém, se escolhermos utilizar a DFS para tornar o grafo fortemente conexo, como descrito no Pré-Processamento, este algoritmo será realizado para todos os vértices do grafo, tendo como origem cada um deles, pelo que a complexidade temporal passará a ser O(|V| * (|V| + |A|)), no pior caso.

Algoritmo de Dijkstra

Uma das hipóteses que estamos a considerar implica recorrer ao **Algoritmo de Dijkstra** para obter o percurso que se adequa ao pedido**,** uma vez que encontra o caminho mais curto de um vértice para outro num grafo dirigido em tempo computacional - O([arestas + vértices] * log (vértices)).

Ora, o Algoritmo de Dijkstra é assim uma boa escolha, em que a distância será obtida pela soma dos pesos das arestas. Trata-se de um algoritmo ganancioso, uma vez que procura maximizar o ganho imediato (neste caso, minimizar o custo ou a distância) em cada passo.

Este algoritmo não é opção para grafos com pesos negativos, uma vez que não garante a exatidão da solução nessas situações, mas tendo em conta que este não é o caso, o algoritmo é perfeitamente aplicável. É uma boa opção se o grafo for esparso (|E|~|V|), como é o caso das redes viárias.

Claramente, teremos de adaptar o Algoritmo de forma a que funcione das maneiras descritas anteriormente, isto é, respeitando todas as restrições e tendo como resultado a Função Objetivo, escolhida pelo cliente.

Algoritmo de Floyd-Warshall

Caso seja escolhida a segunda abordagem explicada no Pré-Processamento, já teremos armazenados os dados da matriz relativos aos caminhos mais curtos entre todos os pares de vértices, pelo que estes poderiam ser aproveitados para resolver os problemas que o programa aborda.

Este algoritmo poderia ser substituído por uma execução repetida do algoritmo de Dijkstra, que teria complexidade temporal O(|V| * (|V| + |E|) * log|V|), no entanto, Floyd-Warshall seria uma melhor opção, tendo em conta que é melhor que o de Dijkstra se o grafo for denso ((|E|~|V|^2), e que, mesmo em grafos menos densos, pode ser melhor devido à sua simplicidade de código.

Baseia-se numa matriz de adjacências W, em que W[v1][v2] conteria o valor da menor distância percorrida desde o vértice v1 até ao v2, e W[v2][v1] conteria o valor da menor distância percorrida desde o vértice v2 até ao v1 (sentidos diferentes). Inicialmente, cada célula da matriz é iniciada com um valor muito elevado, INF, à exceção das células em que o número da linha é igual ao da coluna, em que essa distância toma o valor 0. À medida que se percorre os vértices, estas células vão sendo atualizadas com a menor

distância entre os vértices v1 e v2, pelo que no final da execução do algoritmo teremos completa a matriz com as distâncias mais curtas entre todos os pares de vértices. Serão também armazenados numa outra matriz, também bidimensional, os predecessores de cada célula (correspondente a uma aresta), de modo a poder construir, no final do algoritmo, o caminho mais curto entre dois quaisquer vértices.

A invariante do ciclo principal consiste, portanto, no facto de em cada iteração k desse ciclo, W[i][j] ter a distância mínima desde o vértice i a j, usando apenas vértices intermédios do conjunto {0, …, k}. A fórmula de recorrência do algoritmo pode ser traduzida da seguinte forma:

W[i][j] (iteração k) = mínimo(W[i][j] (iteração k-1) , W[i][k] (iteração k-1) + W[k][j] (iteração k-1))

Este algoritmo tem complexidade temporal O(V^3). Já a sua complexidade espacial é O(|V|^2), uma vez que tanto a matriz que vai armazenar os valores das distâncias mais curtas como a matriz que vai armazenar os predecessores são de duas dimensões.

Casos de utilização e funcionalidades

Temos como objetivo implementar soluções para os quatro níveis de problemas bem como a sua respetiva interface.

A aplicação EatExpress terá inicialmente um menu que permita escolher entre selecionar um mapa, operar sobre o mapa selecionado ou sair.

De seguida, num outro menu, seriam apresentadas as seguintes opções:

- ➔ Visualizar o mapa recorrendo ao GraphViewer;
- ➔ Adicionar ou remover restaurantes;
- ➔ Adicionar ou remover estafetas;
- ➔ Remover clientes;
- ➔ Testar a conetividade do grafo;
- ➔ Resolver as três variantes do problema

Nesta última opção encontrar-se-á um último menu para escolher entre os vários problemas que descrevemos acima e as diferentes implementações e soluções dos mesmos:

Apenas um estafeta a realizar uma única entrega:

Neste modo é pedido ao utilizador que efetue o login, caso já esteja registrado na EatExpress, ou registar-se com uma morada que será adicionada ao grafo caso contrário.

De seguida este pode escolher efetuar um pedido onde terá de escolher um restaurante e um conjunto de pratos. Serão ainda apresentadas duas opções entre as quais o cliente pode escolher: percurso mais barato ou percurso menos demorado. Por último ser-lhe-á mostrado o caminho mais curto/barato que parte da posição do único estafeta da plataforma (posição do último pedido efetuado por este ou Casa dos Estafetas) passa pelo restaurante indicado no pedido e termina na morada do cliente.

Um estafeta pode efetuar múltiplas entregas em simultâneo (carga ilimitada):

Neste modo podem ser efetuados múltiplos logins/registos como na primeira etapa, sendo acrescentados ao grafo múltiplas novas moradas de clientes. A quando de cada um desses logins os clientes serão apresentados com a opção de efetuar os respetivos pedidos. No fim, quando não existirem mais pedidos a serem efetuados, seleciona-se a opção que permitirá ver, como no primeiro caso, o percurso mais curto/barato que parte da posição do estafeta, passa pela lista de restaurantes indicados e termina na morada do último cliente atendido.

Múltiplos estafetas a atender os pedidos (carga ilimitada):

Nesta opção o utilizador irá realizar os mesmos passos que no modo anterior no entanto estarão disponíveis diferentes estafetas com diferentes meios de transporte que os vão realizar. Deste modo será visualizado o percurso dos diferentes estafetas a atenderem os diferentes pedidos/conjunto de pedidos. Baseando-se em combinações de algoritmos utilizados nos casos anteriores.

Múltiplos estafetas a atender os pedidos (carga limitada):

Nesta opção o utilizador irá realizar os mesmos passos que no modo anterior no entanto estarão disponíveis diferentes estafetas com diferentes meios de transporte que os vão realizar. Deste modo será visualizado o percurso dos diferentes estafetas a atenderem os diferentes pedidos/conjunto de pedidos. Baseando-se em combinações de algoritmos utilizados nos casos anteriores.

Conclusão

Em suma, nesta primeira parte do projeto procuramos formalizar um sistema para gerar os percursos mais rápidos, entre os restaurantes e os clientes, para os estafetas da aplicação EatExpress. Optamos por dividir o problema em 4 iterações com graus de complexidade diferentes, de forma a ter uma implementação incremental. Para cada um desses subproblemas foi procurada uma solução que ainda irá ser colocada em prática e aprofundada na segunda parte do trabalho. Para além disso, apresentamos os casos de utilização e algumas funcionalidades que iremos implementar.

A proposta de trabalho continha um intuito educativo, que requeria da nossa parte que compreendêssemos e usássemos, não só novas estruturas como grafos, mas também diferentes algoritmos de pesquisa nos mesmo abordados nas aulas.

Pensamos ter sido bem-sucedidos na realização desta primeira parte do projeto, tendo cumprido todos os objetivos pedidos e acabando o projeto no prazo indicado. No entanto, ao longo da sua realização tivemos algumas dificuldades, uma vez que a sua escrita sem base empírica por trás, fazendo projeções para o futuro, revelou-se uma tarefa mais complicada do que antecipamos inicialmente (por exemplo, nas funcionalidades, onde estávamos a escrever sobre algo que não existe e ainda não temos garantia que serão possíveis de implementar).

Bibliografia

[1] Slides das aulas teóricas de Análise e Concepção de Algoritmos fornecidos ao longo do semestre

[2] Gustavo Pantuza, 2017. "Busca em Profundidade". Acedido a 10 de abril. https://blog.pantuza.com/artigos/busca-em-profundidade

[3] GeeksforGeeks, 2017. "Floyd Warshall Algorithm | DP-16". Acedido a 11 de abril. https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/

[4] UFSC, 2018. "Algoritmo de Dijkstra para cálculo do Caminho de Custo Mínimo". Acedido a 12 de abril. http://www.inf.ufsc.br/grafos/temas/custo-minimo/dijkstra.html

[5] Wikipedia, 2019. "Problema do caixeiro-viajante". Acedido a 19 de abril. https://pt.wikipedia.org/wiki/Problema_do_caixeiro-viajante

Duvidas:

Relatorio:

-> Adicionar o código dos Algoritmos?

Pratica:

1 -> possibilidade de um estafeta efetuar vários pedidos em simultâneo?

2 -> registo de novos clientes? novas moradas? novos nós? (é nos possivel editar os grafos iniciais que são fornecidos de modo a adicionar novas localizaçoes? seria possivel pedir ao utilizador que indicasse um dos nós para facilitar o processo?)

3 -> Temos que dar a possibilidade ao utilizador de usar diferentes algoritmos? ou esse não é o principal objetivo da aplicaçao?

4 -> Os estafetas têm que estar/não estar disponiveis? é suposto contabilizar o tempo?

5 -> Output vai ser efetuado num mapa?