



Inês Silva, up201806385@fe.up.pt

Mariana Truta, up201806543@fe.up.pt

Rita Peixoto, up201806257@fe.up.pt

Grupo 8, Turma 3

**Conceção e Análise de Algoritmos**

**2ºAno MIEIC**

24 de abril de 2020

TourMateApp: rotas turísticas urbanas adaptáveis (tema 3)

**Índice**

[Introdução 2](#_Toc37969161)

[Descrição do tema 2](#_Toc37969162)

[Dados de entrada 3](#_Toc37969163)

[Dados de saída 3](#_Toc37969164)

[Restrições 4](#_Toc37969165)

[Função objetivo 4](#_Toc37969166)

[Perspetiva de solução 5](#_Toc37969167)

[Técnicas de implementação 5](#_Toc37969168)

[Algoritmos 7](#_Toc37969169)

[Casos de utilização e funcionalidades 10](#_Toc37969170)

[Conclusão 11](#_Toc37969171)

[Bibliografia 12](#_Toc37969172)

# Introdução

# Descrição do tema

Neste projeto, a intenção é implementar o aplicativo TourMateApp, que permite a construção de itinerários turísticos adaptáveis ​​às preferências e disponibilidade do usuário.

A aplicação mantém uma lista de pontos turísticos de interesses (POI) e sugere itinerários que incluem as atrações mais adequadas ao perfil do usuário, num caminho que possa ser realizado no tempo indicado, de uma origem a um destino final, indicado pelo usuário.

As recomendações maximizam o número de atrações turísticas de acordo com o perfil e as preferências do usuário, que também pode selecionar circuitos a pé, de carro ou de transporte público.

Para além disto, é necessário ter em atenção se as áreas pelas quais os caminhos passam se encontram no presente momento inacessíveis.

  Esta aplicação irá utilizar mapas reais extraídos do OpenStreetMaps ([www.openstreetmap.org](http://www.openstreetmap.org)) e coordenadas geográficas de alguns pontos de interesse turístico.

<https://moodle.up.pt/pluginfile.php/115863/mod_label/intro/1_CAL1516_2MIEIC06_B.pdf>

<https://github.com/literallysofia/feupcal/blob/master/Documentation/Part_1/Relatorio.pdf>

<https://github.com/AndreFCruz/feup-cal/blob/master/Report%20-%20Final.pdf>

<https://github.com/xRuiAlves/FEUP-CAL-Proj/blob/master/Report.pdf?fbclid=IwAR1ev1R9dOEd8ARSjexbM5PwP7w1FH4-mu7CONFZuC8MG9N5fedASNRmtj0>

[x](https://github.com/diogosmac/feup-cal/blob/master/Entrega_1/CAL1819_2MIEIC04_D_8.pdf)

# 

# Dados de entrada

**G** **(N, E)** - grafo dirigido pesado, representando o mapa da cidade em questão. Constituído por:

**N** - Conjunto de vértices que representam os pontos. Cada ponto é caracterizado por:

id – identificador único do vértice

type - NULL se não for um ponto de interesse

**E** - Conjunto de arestas que ligam os vértices entre si, que representam os caminhos entre pontos. Caracterizadas por:

weight - tempo entre vértices

id – identificador único da aresta

dest ∊ N- vértice destino da aresta

**POIs ∊ N** - conjunto de todos os pontos de interesse

**Preferências do usuário** - tempo máximo para realizar o caminho meio de locomoção selecionado, pontos turísticos de interesse, cidade em que se encontra, ponto inicial do percurso (ponto onde o usuário se encontra, Ni) e ponto final do percurso (ponto de destino do usuário, Nf), etc.

# Dados de saída

**WeightF**- peso total de todas as arestas percorridas no percurso (tempo de duração)

**P** - sequência ordenada dos vértices que representam o melhor caminho entre Ni e Nf, passando pelo maior número de pontos de interesse possíveis de visitar dentro das especificações selecionada, sem ordem especificada.

# 

# Restrições

* Para todos os vértices:

type(N[i]) = “information” v “hotel” v “attraction” v “viewpoint” v “guest\_house” v “picnic\_site” v “artwork” v “camp\_site” v “museum” v “\*” v NULL

type (Ni) = “information” v “hotel” v “attraction” v “viewpoint” v “guest\_house” v “picnic\_site” v “artwork” v “camp\_site” v “museum” v “\*” v NULL

type (Nf) = “information” v “hotel” v “attraction” v “viewpoint” v “guest\_house” v “picnic\_site” v “artwork” v “camp\_site” v “museum” v “\*” v NULL

O type do vértice ser NULL significa que é um vértice genérico e, portanto, não representa um ponto de interesse.

* Para todas as arestas

weight(E[i]) ≥ 0, uma vez que representam distâncias/tempos

Tal como WeightF≥0.

* Ni ∊ N ∧ Ni = P0, isto é, o ponto inicial tem de ser o primeiro vértice na sequência ordenada de vértices que representa o trajeto ótimo
* Nf ∊ N ∧ Ni = Pf, isto é, o ponto final tem de ser o último vértice na sequência ordenada de vértices que representa o trajeto ótimo
* O meio de locomoção escolhido pelo usuário pode ser “a pé” v “carro” v “transportes públicos”.

# Função objetivo

A função objetivo deve retornar um itinerário ótimo que inclua as atrações mais adequadas ao perfil do usuário e que possa ser realizado no tempo indicado, de uma origem a um destino final, indicado pelo usuário.

A solução ótima maximiza o número de atrações que o usuário consegue visitar dentro das suas restrições de tempo e preferências, encontrando o caminho mais curto entre as atrações, permitindo obter o trajeto mais rápido.

Sendo assim, a função objetivo deve minimizar a função P:

# Perspetiva de solução

# Técnicas de implementação

A resolução deste problema passa por um conjunto de fases:

* Pré-processamento do grafo
* Pesquisa no grafo para verificar a possibilidade de chegar do ponto de origem ao de chegada.
* Criar função para encontrar o caminho mais curto entre dois pontos em grafo não dirigido
* Encontrar o máximo de caminhos mais curtos entre pontos de interesse cuja soma de tempo não exceda a especificada pelo utilizador
* Adicionar as restrições de preferências do utilizador e diferentes meios de transporte
* Utilizar grafos dirigidos para representar ruas de dois sentidos

**Pré-processamento do grafo:**

Nesta fase iremos remover informação desnecessária e redundante, começando por remover as arestas que representam vias inacessíveis ( devido a obras na via pública, entre outras razões); tendo em atenção o modo de locomoção do utilizador serão removidas as arestas que não podem ser utilizadas nesse caso ( por exemplo, vias onde não passam veículos, no caso do utilizador querer fazer o seu percurso de carro) e, também, serão removidas arestas redundantes, isto é, no caso de duas arestas terem a mesma origem, destino e sentido, uma delas será removida.

**Verificar se é possível a deslocação entre dois pontos:**

Inicialmente, após serem fornecidos os pontos de partida e de chegada, é necessário verificar se existe pelo menos um caminho possível entre esses pontos, não sendo considerados os pontos de interesse nem o tempo máximo de deslocação.

Nesta fase, não se pretende guardar o caminho encontrado nem o otimizar.

**Função que encontra o caminho mais curto entre dois pontos em grafo não dirigido:**

Como em geral, em problemas de trajetos, interessa não só encontrar um trajeto que vá de encontro a requisitado pelo utilizador, mas o melhor percurso possível, minimizando a distância percorrida e a duração.

Nesta iteração ainda não vão ser tidos em conta os pontos de interesse do usuário, focando apenas em otimizar o percurso possível entre dois pontos.

**Encontrar o máximo de caminhos mais curtos entre pontos de interesse cuja soma de tempo não exceda o especificado pelo utilizador:**

Agora para além de encontrar o melhor caminho, deseja-se também passar por um ou mais pontos de interesse (o máximo possível) do usuário.

Começando por passar apenas por um ponto de interesse e adicionando o máximo possível, com atenção às restrições de tempo impostas pelo utilizador, por uma ordem qualquer.

É importante avaliar a conectividade do grafo, uma vez que certas áreas podem encontrar-se inacessíveis

Em suma, qualquer caminho cujo tempo não exceda o especificado pelo utilizador e passe por pelo menos um ponto de interesse deve ser guardado.

**Adicionar as restrições de preferências do utilizador e diferentes meios de transporte**

**Utilizar grafos dirigidos para representar ruas de dois sentidos**

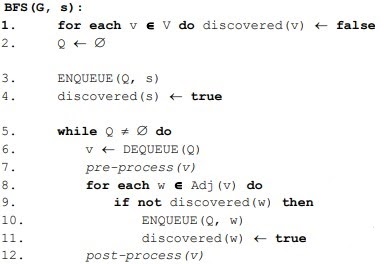
# Algoritmos

A análise da possibilidade de chegar da origem ao destino passa por uma pesquisa no grafo a partir do vértice de origem, a qual pode ser realizada utilizando o algoritmo de pesquisa em profundidade ou de pesquisa em largura.

**Algoritmo de pesquisa em largura**

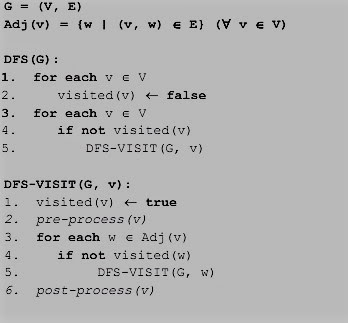
A pesquisa em largura é um dos métodos mais simples para a exploração de um grafo e é a base para muitos outros algoritmos, como o de Dijkstra.

Dado um vértice origem, explora-se sistematicamente as arestas do grafo, descobrindo todos os vértices alcançáveis a partir de s (vértices adjacentes), passando posteriormente à exploração do vértice seguinte, ou seja, encontra todos os vértices a uma distância k do vértice origem antes de descobrir qualquer vértice a uma distância k + 1. Para tal é utilizada uma fila, em que se processa o vértice da frente da fila e à qual são adicionados no fim os vértices descobertos através desse processamento.

****

**Algoritmo de Pesquisa em Profundidade**

Este algoritmo consiste em explorar todas as arestas a partir do último vértice encontrado, sendo implementado de forma recursiva e utilizando o método de backtracking. Quando todas as arestas de um vértice forem exploradas, retorna e explora as restantes arestas do vértice que o antecedia. O algoritmo aprofunda a pesquisa até que encontre o vértice pretendido.



Pôr complexidade

**Algoritmo de Dijkstra**

O algoritmo de Dijkstra é um algoritmo ganancioso que tem como objetivo calcular o caminho mais curto entre dois vértices de um grafo dirigido pesado, sem arestas de peso negativo.

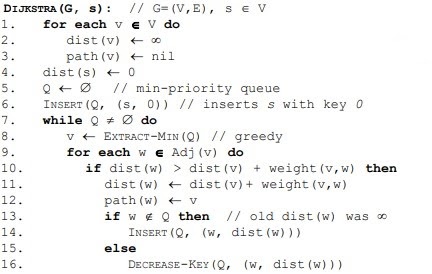
Este algoritmo é semelhante ao de pesquisa em largura, diferenciando-se no facto de utilizar uma fila de prioridade (alterável) como estrutura de dados auxiliar para guardar a ordem dos próximos vértices a pesquisar, na qual se prioriza os vértices cuja soma do peso das arestas do caminho origina uma distância mínima (ao invés de serem processados pela ordem em que foram descobertos, numa fila simples). É um algoritmo ganancioso pois em cada passo procura maximizar o ganho imediato, ou seja, minimizar a distância entre a origem e o destino.

Em cada vértice processado é guardada informação relativa ao vértice que o antecede no caminho e ao encontrar o vértice final na fila de prioridade percorre-se o caminho encontrado no sentido contrário, guardando-o para o retornar, até se retornar ao vértice inicial.

Para analisar a eficiência deste algoritmo, consideremos V o conjunto dos vértices do grafo e E o conjunto das arestas. Sendo que o número de extrações e inserções na fila de prioridades é |V| e cada uma destas operações pode ser realizada em tempo logarítmico no tamanho da fila, que é no máximo |V|, o tempo de execução das extrações e inserções na fila de prioridades é de O( |V| \* log |V|).

A operação de reordenação dos vértices na fila de prioridade é feita, no pior caso, uma vez por cada aresta, ou seja, |E| vezes, e pode ser realizada em tempo logarítmico no tamanho da fila, que no máximo é |V|, resultando numa complexidade O( |E| \* log |V|).

Assim, o tempo de execução do algoritmo de Dijkstra é O( (|V|+|E|) \* log |V|).



**Algoritmo A\***

Este algoritmo constitui uma otimização do algoritmo de Dijkstra que permite um melhoramento (speedup) moderado ao utilizar uma função heurística para orientar a busca do vértice de destino, não garantindo, no entanto, que o caminho encontrado seja o ótimo.

O algoritmo A\* apenas difere no de Dijkstra na ordenação dos vértices na fila de prioridade, dando prioridade aos vértices que se encontram mais próximos do destino ao somar à distância mínima conhecida entre o vértice atual e a origem, o valor da estimativa por baixo da distância mínima do próprio vértice ao vértice de chegada.

Podemos garantir que a solução encontrada será a ótima se, por exemplo, os pesos das arestas forem distâncias em km e a estimativa da distância do vértice atual ao destino for calculada usando a distância Euclidiana (em linha reta) entre os vértices.

# Casos de utilização e funcionalidades

Permitir ao utilizador preencher os seus dados e as suas preferências

* Meio de transporte
* Tempo disponível
* ponto de início (mostrar lista, o utilizador só tem de escolher uma opção)
* ponto de chegada
* pontos de interesse

Nota: era fofinho se colocássemos uma mensagem do tipo “caminho entre A-B possível; estamos a encontrar a melhor rota que satisfaça todos os seus interesses”

Mostrar as rotas disponíveis

Mostrar com o GraphViewer o caminho final (??) com o tempo de deslocação, ...

# Conclusão

Neste relatório, foi discutida uma solução possível para o problema proposto, analisando pormenorizadamente alguns algoritmos, apresentados nas aulas, que possam vir a ser utilizados bem como uma estratégia para o desenvolvimento do projeto.

Tendo o hábito de realizar relatórios durante ou após a implementação do código, a maior dificuldade encontrada foi conseguir organizar o processo de desenvolvimento sem de facto o implementar.

Toda a pesquisa necessária e a elaboração do relatório foram igualmente divididas pelos três membros do grupo, estando em constante comunicação e discussão. Desta forma, o esforço dedicado por cada elemento é de ⅓.

# Bibliografia

* Slides das aulas teóricas
* Wikipedia
* Relatórios fornecidos pelo docente