U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

24 de abril de 2020
2º ano do MIEIC, 2º semestre
Concepção e Análise de Algoritmos

TourMateApp

rotas turísticas urbanas adaptáveis

Turma 5 Grupo 4

Jéssica Nascimento up201806723@fe.up.pt
Rafael Cristino up201806680@fe.up.pt
Xavier Pisco up201806134@fe.up.pt

Índice

Siglas e Acrónimos

- ➔ OSM: OpenStreetMap (openstreetmap.org)
- ➔ POI: Ponto de Interesse (*Point of Interest*)

Descrição do Problema

A TourMateApp pretende **gerar itinerários eficientes, construídos consoante os interesses e a disponibilidade temporal de cada utilizador, bem como o ponto de partida e o ponto de chegada, baseados num mapa fornecido**.

Estes itinerários são definidos por um ponto de partida e um ponto de chegada, e tanto um como o outro são especificados pelo utilizador:

- ➔ **Caso o utilizador especifique ponto de partida** (provavelmente a sua localização atual) **mas não especifique ponto de chegada**, o itinerário será caracterizado como sendo um itinerário primariamente turístico, o que significa que **a aplicação procurará gerar um itinerário que passe pelo número máximo de pontos turísticos, dentro do tempo especificado e dando prioridade aos interesses do utilizador**. Se não for possível visitar nenhum ponto turístico dentro da janela de tempo fornecida, a aplicação dará ao utilizador a opção de alterar o tempo disponível ou cancelar a operação.

- ➔ **Caso o utilizador especifique ponto de chegada e ponto de partida**, e tenha ativado a opção de preencher o tempo restante disponível (calculado pela subtração do tempo do percurso ao tempo disponível) por visitas a atrações turísticas, **a aplicação procurará preencher esse tempo disponível com visitas a pontos turísticos, mediante os interesses do utilizador e sem exceder o tempo disponível especificado**. Senão, gera apenas o caminho mais curto desde a origem ao destino.

Decomposição do problema

Vertentes em que se divide o problema, numa perspectiva computacional.

1 - Obtenção do grafo a partir do mapa

Várias alternativas:

- ➔ mapas provenientes do OSM (XML) sem processamento prévio
  - ◆ Utilização de um XML parser e utilização do resultado para criar os vértices e arestas, que são posteriormente adicionados ao grafo.
- ➔ mapas fornecidos pelos docentes da unidade curricular, processados
  - ◆ GridGraphs
  - ◆ PortugalMaps
  - ◆ TagExamples (PortugalMaps com tags provenientes do OSM)

2 - Cálculo e atribuição das distâncias às arestas do grafo

Cálculo com base na diferença entre as coordenadas dos vértices que são conectados pela aresta.

3 - Adição da relação dos POIs com os vértices do grafo

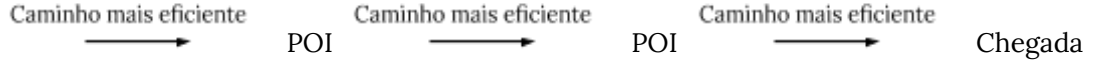
- ➔ Determinação do vértice que tem as coordenadas que mais se aproximam às coordenadas do ponto de interesse.
- ➔ Atribuição do objeto 'POI' ao vértice escolhido, e do vértice escolhido ao objeto 'POI'.

4 - Cálculo do itinerário mais eficiente desde a localização de partida à localização de chegada.

Ainda não tendo em conta as opções do utilizador, e assumindo a receção de ambas as localizações.

5 - Adição da capacidade de visitar POIs

Envolve o cálculo do caminho mais eficiente da localização de partida à localização do POI, e do ponto de interesse à localização de chegada.



6 - Adição da capacidade de visitar o número máximo de POIs, seguindo as preferências do utilizador (os seus interesses e tempo disponível)

Envolve a comparação dos caminhos resultantes de acrescentar variados POIs, sem ultrapassar o tempo disponível.

7 - Adição da capacidade de gerar um itinerário sem ser definido um local de chegada (puramente turístico)

Envolve o ponto anterior, mas sem a restrição do ponto de chegada.

Identificação e formalização do problema

Dados de entrada

- ➔ **M(V_M, E_M)** - Grafo do mapa da cidade em que o utilizador se encontra
  - ◆ **V_M** - Vértices correspondentes às interseções entre as ruas (arestas) que incluem:
    - **Localização** - Coordenadas da sua localização
    - **Adj** - Arestas adjacentes
    - **POI** - objeto relativamente ao POI, caso exista um POI nas coordenadas do vértice (ou muito próximo)
    - **Outros dados** - Fornecidos no mapa (tags, no caso de mapas provenientes do OSM)
  - ◆ **E_M** - Arestas correspondentes às ruas
    - **Destino** - Vértice correspondente ao final da aresta
    - **Distância** (weight) - Distância coberta pela aresta (do vértice inicial ao final)
    - **Outros dados** - Fornecidos no mapa (tags, no caso de mapas provenientes do OSM, por exemplo nome da rua)
- ➔ **Localização/Origem** - Vértice correspondente ao início do itinerário
- ➔ **Destino** - Vértice correspondente ao final do itinerário (opcional, como descrito na descrição do problema)
- ➔ **Tempo livre/Hora de chegada** - Tempo máximo da duração do itinerário
- ➔ **Pontos de interesse** - Lista dos pontos de interesse que presentes no mapa
- ➔ **Preferências** - Lista dos tipos de pontos de interesse que a pessoa mais gosta

Dados de saída

- ➔ **$M(V_M, E_M)$** - Grafo do mapa da cidade em que o utilizador se encontra
- ➔ **Melhor percurso** - Lista das arestas a percorrer para o itinerário
- ➔ **Pontos de interesse** - Lista de pontos de interesse por onde vai passar nesse itinerário
- ➔ **Tempo estimado do percurso** - Tempo estimado para o itinerário no total (incluindo o tempo médio de estadia nos pontos turísticos escolhidos)

Restrições

- ➔ A origem e o destino têm de existir (o destino pode ser calculado pelo programa caso não seja fornecido) e pertencer ao mapa utilizado
- ➔ O tempo livre tem de ser maior que 0
- ➔ Os pontos de interesse de um percurso têm de existir e pertencer ao mapa utilizado

Função objetivo

- ➔ O itinerário ter o maior número possível de pontos de interesse dos dados de entrada, respeitando as preferências do utilizador quando possível
- ➔ O tempo estimado para a conclusão do itinerário não ultrapassar o tempo disponível fornecido nos dados de entrada
- ➔ O ponto inicial e o ponto de chegada corresponderem aos fornecidos nos dados de entrada

Solução

A solução teorizada para o problema deste projeto envolve 2 algoritmos principais:

➔ **Dijkstra**

Este será utilizado para calcular o caminho mais curto desde um ponto de partida até um ponto de chegada (quer seja de um POI para outro POI ou do ponto de partida para um POI, etc), que nos ajudará a descobrir o tempo que o utilizador pode demorar a chegar lá.

Sendo necessário obter também o caminho mais curto entre o ponto de partida e um POI, deste até outro ponto de interesse ou chegada, este algoritmo também será usado para construir o nosso itinerário em conjunto com o algoritmo de retrocesso.

A vantagem de usar este algoritmo é que nos fornece o caminho mais curto para vários pontos tendo o mesmo ponto de partida. Isto ajuda-nos a verificar qual o ponto de interesse mais perto e o seu tempo de distância. Eventualmente, mediante os casos poderemos considerar a utilização do *algoritmo de Floyd–Warshall*.

```
DIJKSTRA(G, s):   // G=(V,E), s ∈ V
1.    for each v ∈ V do
2.       dist(v) ← ∞
3.       path(v) ← nil
4.    dist(s) ← 0
5.    Q ← ∅   // min-priority queue
6.    INSERT(Q, (s, 0)) // inserts s with key 0
7.    while Q ≠ ∅ do
8.       v ← EXTRACT-MIN(Q) // greedy
9.       for each w ∈ Adj(v) do
10.         if dist(w) > dist(v) + weight(v,w) then
11.            dist(w) ← dist(v)+ weight(v,w)
12.            path(w) ← v
13.            if w ∉ Q then  // old dist(w) was ∞
14.               INSERT(Q, (w, dist(w)))
15.            else
16.               DECREASE-KEY(Q, (w, dist(w)))
```

Tempo de execução:
O((|V|+|E|) * log |V|)

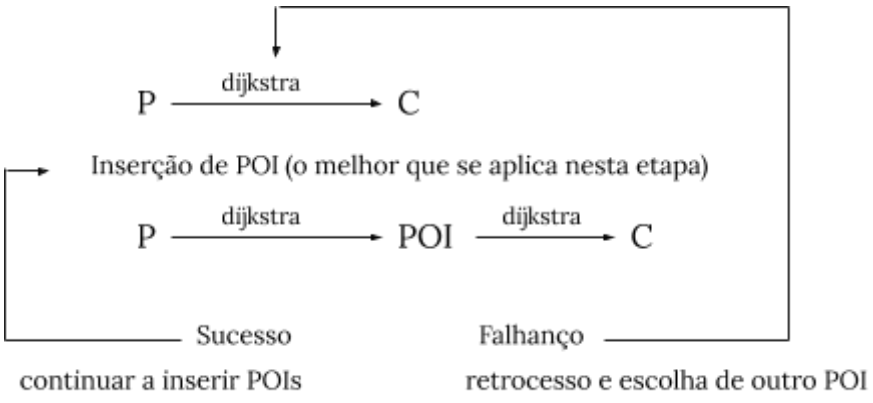
Algoritmo de Dijkstra (adaptado)[1]

➔ **Algoritmo baseado em backtracking**

Tal como no caso do labirinto ou do sudoku, os itinerários serão obtidos em grande parte por este algoritmo que utiliza a técnica de tentativa e erro para encontrar a saída, que neste caso é o melhor itinerário: com o tempo dentro do estipulado pelo utilizador, com as preferências dele e com o maior número de POIs. Em cada etapa é inserido um POI ao itinerário, conduzindo quer ao sucesso ou ao falhanço, caso em que se retrocede e se opta por uma alternativa (quando não houver alternativas considera-se completo).

Este algoritmo pode também ter uma *característica gananciosa*: em cada etapa, procura inserir o POI que se mostra ser a escolha óptima tendo em conta as circunstâncias atuais.

Como dito acima, este algoritmo será utilizado em conjunto com o Dijkstra para que seja possível encontrar o caminho mais curto entre cada POI e destes ao local de partida e de chegada. A utilização destes dois algoritmos em conjunto permite-nos utilizar o tempo que o utilizador demora a chegar a um local e o seu tempo de estadia no mesmo, o que contribui para comparar o itinerário com o tempo livre do utilizador.



Ao longo da implementação este algoritmo vai certamente sofrer alterações, mas esta é a nossa ideia inicial.

Casos de utilização e funcionalidades

1. O utilizador tem um intervalo de x tempo e pretende aproveitá-lo visitando alguns POIs na área, tendo de regressar ao local original.

Ponto de partida = ponto de chegada

2. O utilizador está a fazer escala entre dois voos e pretende conhecer a cidade em que se encontra, sem perder o voo.

Semelhante ao caso anterior

3. O utilizador tem um local onde pretende estar, mas tem mais tempo do que o necessário para lá chegar.

Ponto de partida = localização atual

Ponto de chegada = localização final

O utilizador visitará atrações turísticas a caminho do seu destino, sem ultrapassar o tempo disponível.

4. O utilizador está numa situação puramente turística e não pretende ter uma específica localização final.

Ponto de partida = localização atual

Ponto de chegada = null

O utilizador visitará o maior número de atrações turísticas sem ultrapassar o tempo disponível.

5. O utilizador pretende utilizar a aplicação simplesmente como um GPS.

A aplicação gerará o itinerário mais eficiente entre o ponto de partida e de chegada fornecidos.

6. O utilizador pretende escolher os pontos de interesse a visitar.

A aplicação gerará o itinerário mais eficiente entre o ponto de partida e os pontos de interesse, terminando no ponto de chegada ou no último ponto de interesse, case o ponto de chegada não seja especificado.

Todos os pontos anteriores que envolvem pontos de interesse têm em conta as definições escolhidas pelo utilizador, nomeadamente os seus interesses, que definem o tipo de atrações que ele se sente compelido a visitar.

Outros aspetos a considerar: receber coordenadas do utilizador, e determinar os vértices que mais se aproximam às mesmas, obtendo assim a sua localização; talvez também receber coordenadas dos destinos e pontos de interesse, etc.

Conclusão

O presente relatório é essencial para o apoio à criação e desenvolvimento do projeto.

Durante a sua realização tentamos prever quais seriam as necessidades de uma aplicação com as especificações fornecidas bem como o potencial a nível da sua utilização e das suas funcionalidades. Tentámos também teorizar uma possível solução para o problema proposto.

No entanto, ao passar à parte prática, estamos de espírito aberto para novas funcionalidades, perspectivas e soluções que nos ocorram, ao melhor nos familiarizarmos com o evoluir da aplicação e com a sua especificidade.

Em termos de esforço individual, cada um de nós esteve presente em toda a redação deste relatório (não se destacando algum elemento do grupo), bem como na idealização de cada uma das suas vertentes.

Referências

[1] “Algoritmos em Grafos: Caminho mais curto (Parte I)”, R. Rossetti, L. Ferreira, H. L. Cardoso, F. Andrade