U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

# ExploreIT: caminhadas e patuscadas

## Conceção e Análise de Algortimos

## Mieic 2º Ano

Turma 1 - Grupo 7

José David Rocha, up201806371
Telmo Botelho, up201806821
Tomás Mendes, up201806522

# Índice

# 1 Descrição do Tema

No âmbito da sua política de fortalecimento do espírito de equipa, uma empresa decidiu oferecer aos seus trabalhadores um fim de semana numa região rural, cheia de locais de interesse ligados por trilhos para a realização de caminhadas.

Os trilhos podem ter diferentes graus de dificuldade, sendo por vezes desaconselhados a pessoas pouco habituadas a estas andanças. Dada a dimensão do grupo, as pessoas poderiam, ao longo do dia, optar por efetuar diferentes conjuntos de trilhos, com a restrição de haver locais de encontro de todo o grupo, nomeadamente para o almoço (num restaurante rústico da zona, numa aldeia remota) e para o final do dia.

Neste trabalho, pretende-se implementar uma aplicação que, dado um conjunto de pontos de interesse, locais de confluência e trilhos, produza circuitos que possam ser percorridos pelos trabalhadores. Cada trilho tem um grau de dificuldade e uma duração estimada, função da distância e do declive do mesmo.

A dificuldade de alguns dos trilhos pode fazer com que certos pontos de interesse se tornem inacessíveis (em tempo útil), inviabilizando a sua utilização nos circuitos gerados. Assim, é necessário avaliar a conectividade do grafo, a fim de identificar pontos de interesse com pouca acessibilidade.

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

# 2 Formalização do Problema

## 2.1 Dados de Entrada

$G_i = (V_i, E_i)$ - grafo dirigido pesado, representando a zona rural, composto por:

- V - vértices (que representam pontos de interesse) com:

  Desc - Descrição do ponto de interesse

  Type - Ponto de Encontro ou de Interesse, NULL caso não seja nenhum dos anteriores

  Adj $\subseteq$ E - Conjunto de arestas que partem do vértice

- E - arestas (que representam vias de comunicação) com:

  W - peso da aresta (representa a distância entre os vértices que a delimitam)

  Id - identificador único da aresta

  Dest $\in V_i$ - vértice de destino

S $\in$ Vi - vértice inicial (entrada da região rural)
T $\in$ Vi - vértice final (ponto de encontro dos trabalhadores)
Duração Máxima - duração máxima para percorrer o trilho
Dificuldade Máxima - dificuldade máxima dos trilhos a percorrer, de modo a selecionar apenas os trilhos adequados consoante a experiência do trabalhador em questão

## 2.2 Dados de Saída

WeightF - peso total de todas as arestas percorridas (duração total)
P - sequência ordenada dos vértices que melhor se enquadram nas preferências do utilizador, maximizando o número de pontos de interesse possíveis de visitar tendo em conta a experiência do mesmo

## 2.3 Restrições

- Weight(E[i]) $\geq 0$, dado que representam distâncias
- $V_i \in \mathrm{V} \wedge V_i = P_0 \wedge \mathrm{type}(P_0) =$ "ponto de encontro", ou seja, o ponto inicial terá de ser o primeiro vértice presente na sequência de vértices ordenados relativos aos dados de saída e também terá que ser um ponto de encontro
- $V_f \in \mathrm{V} \wedge V_f = P_f \wedge \mathrm{type}(P_f)$, ou seja, o ponto final terá de ser o último vértice presente na sequência de vértices ordenados relativos aos dados de saída e também terá que ser um ponto de encontro
- type(V[i]) = "ponto de encontro" $\vee$ "ponto de interesse" $\vee$ NULL
- type($V_f$) = "ponto de encontro"
- R $\in$ V $\wedge$ type(R) = "ponto de encontro", uma vez que o local do restaurante os os trabalhadores irão almoçar terá que ser um ponto de encontro

Quando um vértice é catalogado como NULL significa que este representa um ponto genérico, não sendo, portanto, um ponto de encontro ou de interesse.

## 2.4 Função Objetivo

A função objetivo deverá retornar uma sequência ordenada de vértices que representam um circuito ótimo, consoante as preferências de duração e dificuldade máximas do utilizador. Com efeito, a solução ótima maximiza o número de pontos de interesse que o utilizador poderá visitar tendo em conta as restrições relativas à duração máxima e experiência do mesmo. Logo, a solução ótima passa pela minimização da seguinte função:

$$f = \sum_{e \in P} w(e)$$

# 3 Perspetiva de Solução

## 3.1 Técnicas de Implementação

A resolução deste problemas passará pela implementação de um conjunto de estratégias.

## 3.2 Pré-Processamento do Grafo

Numa fase inicial, serão selecionados apenas trilhos cuja dificuldade seja adequada à experiência do utilizador, de modo a selecionar quais os pontos passíveis de serem visitados e quais os pontos inacessíveis.

## 3.3 Determinação dos trilhos possíveis de acordo com as limitações de duração máxima do utilizador

Após a filtragem inicial dos pontos que podem ser visitados, pretende-se agora determinar quais os conjuntos de trilhos que podem ser percorridos pelo utilizador entre a origem do grafo e ponto de encontro, maximizando o número de pontos de interesse visitados e o número de trilhos percorridos.

## 3.4 Maximização dos pontos de interesse visitados

Finalmente, a solução ótima passará por escolher, do conjunto obtido, o conjunto que possui maior número de trilhos e pontos de interesse numa duração total menor.

# 4 Casos de Utilização e Funcionalidades

Neste projeto, temos como objetivo implementar uma interface com vários menus, sendo que pretendemos que os utilizadores consigam fazer as opções que condizem com os seus interesses, no que toca ao trilho que tencionam percorrer.

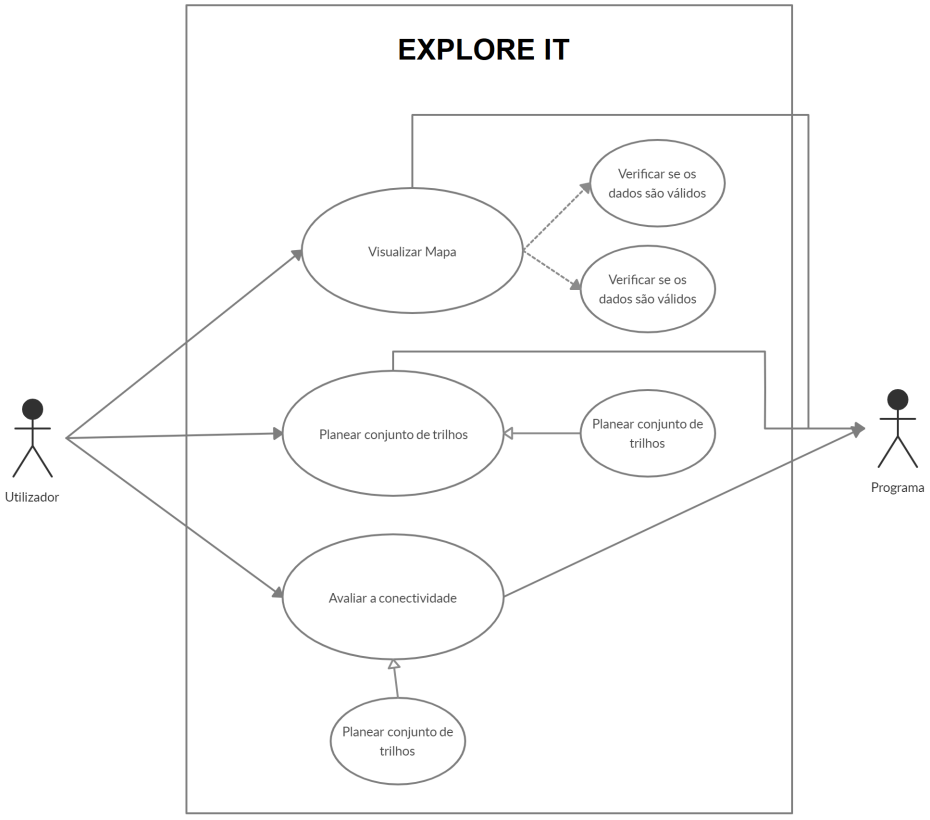
Assim, será permitida a visualização de um mapa e a navegação entre diferentes locais. Neste sentido, o utilizador poderá delimitar o mapa da zona que possivelmente irá percorrer através das restrições que colocar. No que diz respeito a este último tópico, poderá ainda filtrar a sua pesquisa com, por exemplo, a sua experiência em caminhadas rurais, pontos de interesse (enumerando por ordem decrescente), tempo dispensado no trilho, pontos de partida e de chegada, entre outras alternativas.

Por fim, após ser recolhida a informação selecionada e filtrada pelo *user*, verificar-se-á se existe ou não um caminho possível. Em caso afirmativo, será apresentado o melhor caso, isto é, aquele que consiga passar pelo maior número de pontos de interesse escolhidos no menor tempo possível, tendo em conta também a experiência do grupo.

# 5 Principais casos de uso implementados

Um dos objetivos do programa foi implementar uma interface que facilitasse a interação com o utilizador. Com efeito, o utilizador poderá:

- **Planear trilhos** - o utilizador fornece a duração máxima do percurso (que varia entre 5 e 12 horas), bem como a dificuldade máxima dos trilhos a percorrer. Após esta fase inicial o programa irá calcular o melhor conjunto de trilhos possíveis para cada elemento do grupo.
- **Visualizar o mapa da região** - o utilizador pode visualizar o mapa da região.



# 6 Estruturas de Dados Utilizadas

Ao longo da execução foram utilizadas estruturas de dados, com o objetivo de implementar os algoritmos necessários para o bom funcionamento do programa. Com efeito, foram implementados vetores para armazenar os caminhos devolvidos conforme a duração e dificuldade máximas, bem como uma Mutable Priority Queue, fornecida durante as aulas práticas.

Os trilhos calculados são armazenados em vetores e depois divididos em 3 vetores secundários, conforme a sua dificuldade (fácil, média e difícil). Estes são depois atribuídos aos vários trabalhadores, maximizando a diversidade entre trilhos, de modo a que praticamente todos os trabalhadores tenham trilhos diferentes.

# 7 Algoritmos

## 7.1 Algoritmo de Pesquisa em Profundidade

Este algoritmo consiste em explorar todas as arestas relativas a partir de um vértice $v$, que corresponde ao último vértice encontrado.

Quando todas as arestas de $v$ forem exploradas, retorna para explorar arestas que saíram do vértice a partir do qual v foi descoberto. Se se mantiverem vértices por descobrir, um deles é selecionado, através de *backtracking*, como a nova fonte e o processo de pesquisa continua a partir daí.

Este processo é repetido até todos os vértices terem sido descobertos.

O facto da pesquisa em profundidade ser recursiva faz com que as chamadas recursivas guardem na stack |V| vértices (no pior caso). A sua complexidade espacial é, portanto, O(|V|).

```
G = (V, E)
Adj(v) = {w | (v, w) ∈ E} (∀ v ∈ V)

DFS(G):
1.  for each v ∈ V
2.      visited(v) ← false
3.  for each v ∈ V
4.      if not visited(v)
5.          DFS-VISIT(G, v)

DFS-VISIT(G, v):
1.  visited(v) ← true
2.  pre-process(v)
3.  for each w ∈ Adj(v)
4.      if not visited(w)
5.          DFS-VISIT(G, w)
6.  post-process(v)
```

## 7.2 Algoritmo de Dijkstra

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

Para tal, considera-se um conjunto S de menores caminhos, com um vértice inicial I. A cada passo do algoritmo percorrem-se as arestas adjacentes aos vértices pertencentes a S de modo a descobrir o vértice com menor distância relativa a I. Este vértice é, então, adicionado a S e repete-se o processo até que todos os vértices alcançáveis por I estejam em S.

É caracterizado como ganancioso pelo facto de que, em cada passo, procura maximizar o ganho imediato, ou seja, minimizar a distância entre o ponto inicial e o ponto final.

O complexidade temporal do algoritmo de Dijkstra é O((|V|+|E|) * log |V|).

O número de extrações ou inserções na fila será, no pior caso, |V|, sendo cada uma destas operações realizada em tempo logarítmico no tamanho da fila. Assim, o tempo de execução será O(|V| * log |V|).

Relativamente à reordenação de vértices, esta é também realizada em tempo logarítmico, resultando num tempo de execução de O(|E| * log |V|).

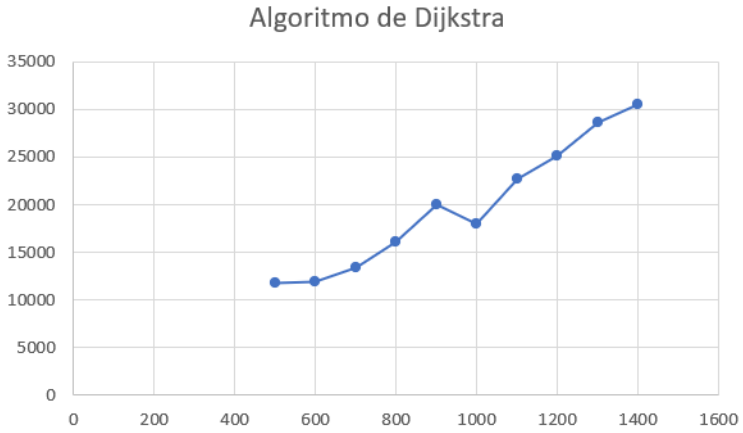
Somando os dois resultados acima, obtemos a complexidade temporal do algoritmo.

```
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)))
```

# 8 Análise Temporal Empírica

Para calcular a análise temporal empírica do programa recorreu-se à implementação de uma classe Timer. Com efeito, esta irá calcular os segundos decorridos durante o processo de geração de trilhos.

Verifica-se que o tempo tende a aumentar consoante maiores as restrições, ou seja, maior número de trabalhadores, menor duração e maior variedade na experiência dos trabalhadores. Contudo, esta raramente ultrapassa os 30 segundos, demorando, em média cerca de 20-30 segundos.

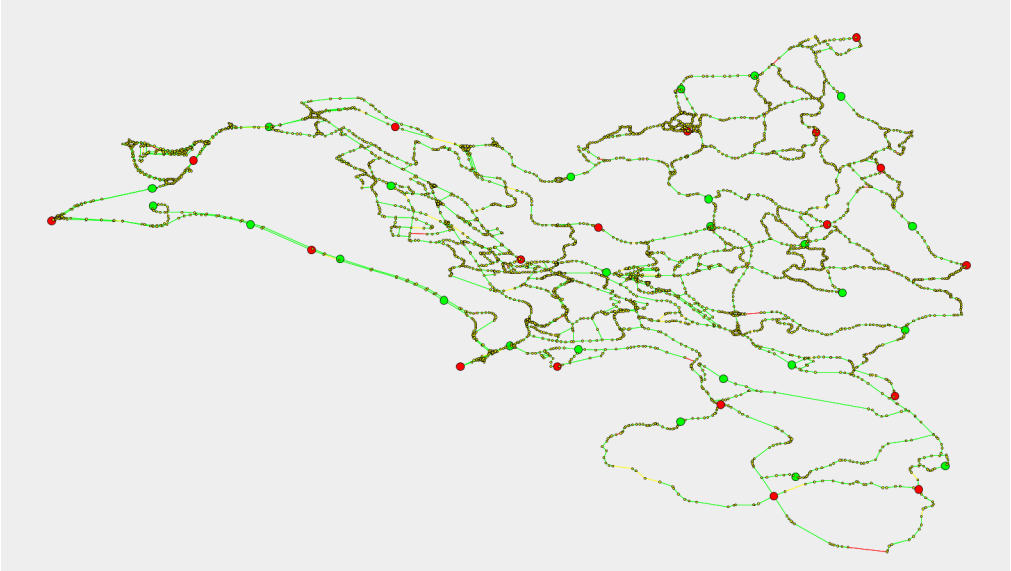


# 9 Conectividade dos grafos utilizados

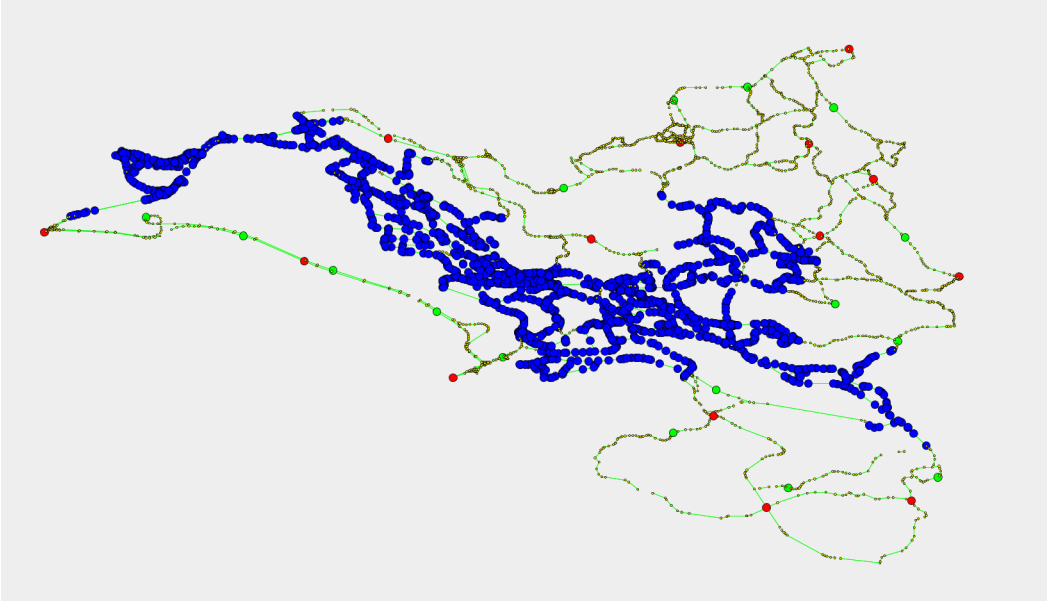
O grafo utilizado no decurso do programa representa uma zona rural de Penafiel.

Com vista a avaliar a conectividade do grafo, foi implementado o algoritmo de pesquisa em profundidade, de acordo com o que foi descrito nos slides das aulas teóricas.

Após fornecida a dificuldade máxima são calculados os pontos passíveis de serem percorridos a partir de um ponto inicial, também fornecido pelo utilizador.



Mapa da Região Rural



Trilhos com dificuldade máxima 2 passíveis de serem percorridos a partir do ponto 5

# 10 Conclusão

Ao longo deste trabalho, analisámos, numa fase inicial, o problema em questão e debatemos uma possível solução, assim como as diferentes etapas pelas quais deveríamos passar. Deste modo, após tratarmos da formalização do problema, apresentámos a nossa perspetiva de solução, tal como os algoritmos que considerámos relevantes para o desenvolvimento do projeto.

Tendo por hábito a realização de relatórios após o código estar implementado, o maior obstáculo que encontrámos foi pensarmos nos algoritmos que poderiam resolver o problema principal, uma vez que não estávamos com total certeza de quais seriam mais adequados.

Durante a implementação do projeto em si deparamo-nos com algumas dificuldades, especialmente no cálculo dos trilhos face às restrições impostas e que algoritmos utilizar para tal. Com efeito, foram feitas várias experiências com vista a determinar qual a melhor estratégia de solução.

Por último, no que toca à distribuição de tarefas e esforço do grupo, a percentagem de esforço deve ser divida em 40 por cento tanto para o David e para o Tomás e 20 para o Telmo.

# 11 Bibliografia

1. Slides usados nas aulas teóricas
2. Relatórios de anos passados fornecidos pelos docentes
3. Classe Timer - https://gist.github.com/mcleary/b0bf4fa88830ff7c882d
4. https://en.wikipedia.org/wiki/Dijkstra
5. https://en.wikipedia.org/wiki/Breadth-first_search