À procura de estacionamento

Relatório de projeto

Concepção e Análise de Algoritmos

Mestrado Integrado em Engenharia Informática e Computação (MIEIC)

2ºAno 2ºSemestre - 2020/2021

Turma 7 - Grupo 1:

João Martins up201706978@edu.fe.up.pt

Nuno Castro up202003324@edu.fe.up.pt

Ricardo Cardoso up201604686@edu.fe.up.pt

Índice

1 – Introdução

1.1 Apresentação do projeto

Este projeto surge no âmbito da unidade curricular de Concepção e Análise de Algoritmos em que se pretende o processamento de um mapa para o cálculo de um caminho ideal conforme as necessidades do utilizador.

Foram definidos alguns requisitos iniciais para a realização deste projeto, sendo eles os seguintes:

- descrição do problema.
- identificação e formalização do problema.
- identificação de funcionalidades e casos de utilização a considerar para a implementação.
- formulação de uma proposta de solução e identificação dos principais algoritmos a considerar para a implementação desta mesma.

1.2 Descrição do projeto

O tema atribuído ao nosso grupo foi a procura de estacionamento numa determinada área e, neste contexto, o processamento do mapa e cálculo do caminho deverão contemplar o seguinte cenário de otimização:

- distância do ponto de origem até ao parque de estacionamento.
- garantir que o parque de estacionamento se encontra próximo do destino selecionado

Para este efeito, identificamos que a nossa solução para o problema deverá permitir que o utilizador define tanto o seu local de origem como o de destino.

2 – Formulação do problema

2.1 Dados de entrada

- V - Conjunto de vértices presentes no mapa que representam pontos de interesse para o serviço. Cada um destes vértices é caracterizado por:
    - Identificador (Único).
    - Coordenadas GPS (Localização geográfica do vértice).
    - Tipo do vértice (bool isCarPark).
- E - Conjunto de arestas que ligam os vértices que representam ruas que ligam pontos de interesse. Cada uma destas arestas é caracterizada por:
    - S - aresta inicial.
    - F - aresta final.
    - W - Peso da aresta que corresponde à utilidade da mesma para o serviço (comprimento da estrada).
- G(V, E) - Grafo dirigido pesado que contém a informação sobre os vários pontos de interesse do mapa (Vértices) e as estradas que os ligam (Arestas).
- AllAdj(Vi) - conjunto de arestas que entram e saem de Vi.
- Adj(Vi) - conjunto de arestas que saem de Vi.

2.2 Dados de saída

- Rc(i) - Array de conjunto de vértices ordenados da viagem de carro até ao estacionamento.

2.3 Restrições

- Todas as arestas têm de ter peso (distância) positivo (W(Ei) > 0).

2.4 Funções objetivo

Como já referido anteriormente, para este projeto pretende-se calcular um trajeto otimizado à distância percorrida desde a origem até ao parque de estacionamento. A função objetivo será:

Obj = Mín(Distância)

3 – Trabalho Implementado

3.1 Estratégia adotada

Para uma resolver o problema foi considerado que apenas existe um vértice de origem e um vértice de destino, não havendo paragens intermédias. É utilizado um algoritmo de pesquisa em largura para encontrar um dos parques de estacionamento mais próximos do destino e de seguida é utilizado o algoritmo de Dijkstra para calcular o caminho mais curto entre a origem e o parque de estacionamento.

A conectividade do grafo é analisada, retornando os componentes fortemente conexos identificados com uma cor aleatória (de forma a distinguir facilmente cada uma das componentes).

Ambas as funcionalidades funcionam em qualquer um dos grafos dirigidos presentes na página do moodle da unidade curricular.

3.2 Algoritmos

3.2.1 Dijkstra

Para este projeto, um algoritmo utilizado é o algoritmo de Dijkstra dado que este problema trata-se de calcular o caminho mais curto com um grafo pesado contendo apenas valores positivos.

Para este algoritmo, encontra-se as seguintes complexidades temporais e espaciais:

- Temporal: O((|V| + |E|) log V).
- Espacial: O(|V| + |E|).

Pseudocódigo

```
procedure dijkstra(G(V,E), s):
      for each v ∈ V do
            dist(v) ← ∞
            path(v) ← null
      dist(s) ← ∅
      Q ← ∅
      Insert(Q, (s, 0))
      while Q ≠ ∅ do
            v ← Extract-Min (Q)
            for each w ∈ Adj(v) do
                  if dist(w) > dist(v) + weight(v,w) then
                        dist(w) ← dist(v) + weight(v,w)
                        path(w) ← v
                        if w ∉ Q then
                              Insert(Q, (w, dist(w)))
                        else
                              Decrease-Key(Q, (w,dist(w)))
```

3.2.2 Pesquisa em Largura

De modo a encontrar o parque de estacionamento mais próximo do local de destino recorremos a uma pesquisa em largura a qual nos irá devolver o vértice correspondente a este mesmo.
Para este algoritmo, encontra-se as seguintes complexidades temporais e espaciais:

- Temporal: O(|V| + |E|).
- Espacial: O(|V| + |E|).

Pseudocódigo

```
procedure bfs(G, s):
      for each v ∈ V do discovered(v) ← false
      Q ← ∅
      ENQUEUE(Q, s)
      discovered(s)  true
      while Q ← do
            v ← DEQUEUE(Q)
            pre-process(v)
            for each w ∈ Adj(v) do
                  if not discovered(w) then
                        ENQUEUE(Q, w)
                        discovered(w) ← true
            post-process(v)
```

3.2.3 Conectividade

- Pesquisa em profundidade no grafo G determina floresta de expansão, numerando vértices em pós-ordem (ordem inversa de numeração em pré-ordem)
- Inverter todas as arestas de G (grafo resultante é Gr)
- Segunda pesquisa em profundidade, em Gr, começando sempre pelo vértice de numeração mais alta ainda não visitado
- Cada árvore obtida é um componente fortemente conexo, i.e., a partir de um qualquer dos nós pode chegar-se a todos os outros

Para este algoritmo, encontra-se as seguintes complexidades temporais e espaciais:

- Temporal: O(|V| + |E|).
- Espacial: O(|V| + |E|).

Pseudocódigo

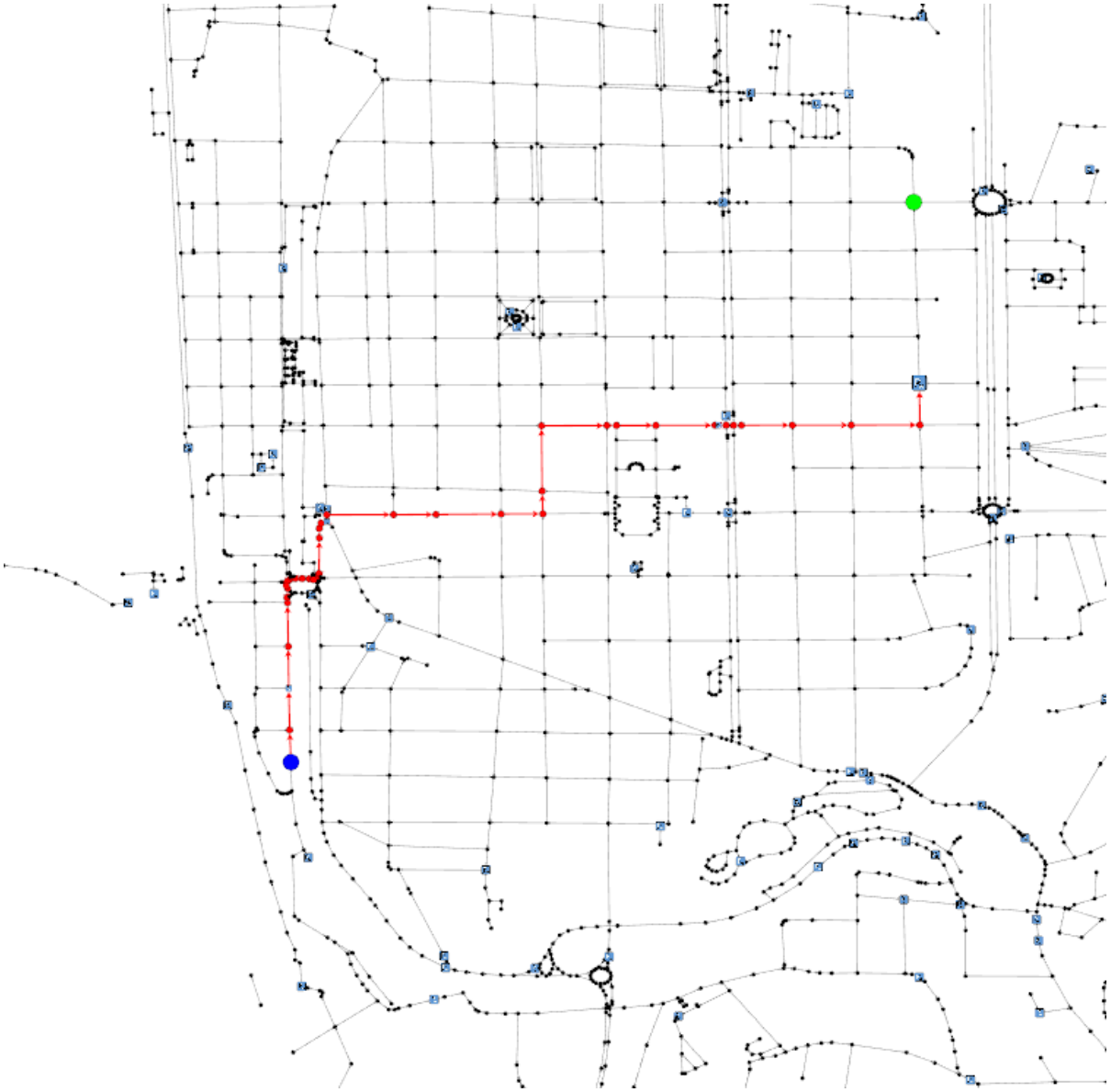
```
DFS(G):
      for each v ∈ V
            visited(v) ← false
      for each v ∈ V
            if not visited(v)
                  DFS-VISIT(G, v)


DFS-VISIT(G, v):
      visited(v) ← true
      pre-process(v)
      for each w ∈ Adj(v)
            if not visited(w)
                  DFS-VISIT(G, w)
      post-process(v)
```

3.3 Casos de utilização

- Escolher origem e destino.
- Testar a conectividade do mapa.

4 – Exemplos de execução

Exemplo “Encontrar parque de estacionamento”:

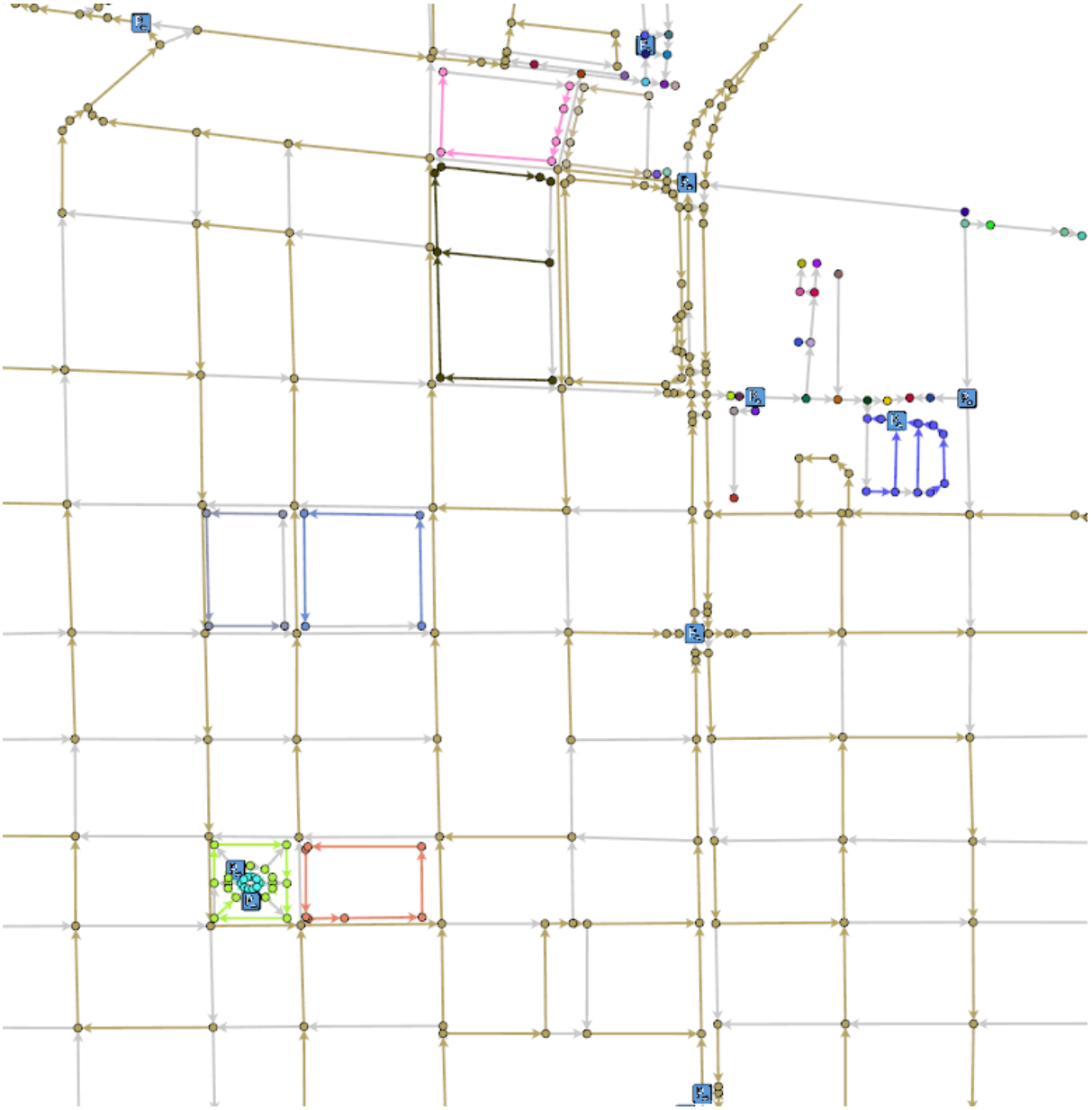


Ponto azul -> Origem

Ponto verde -> Destino

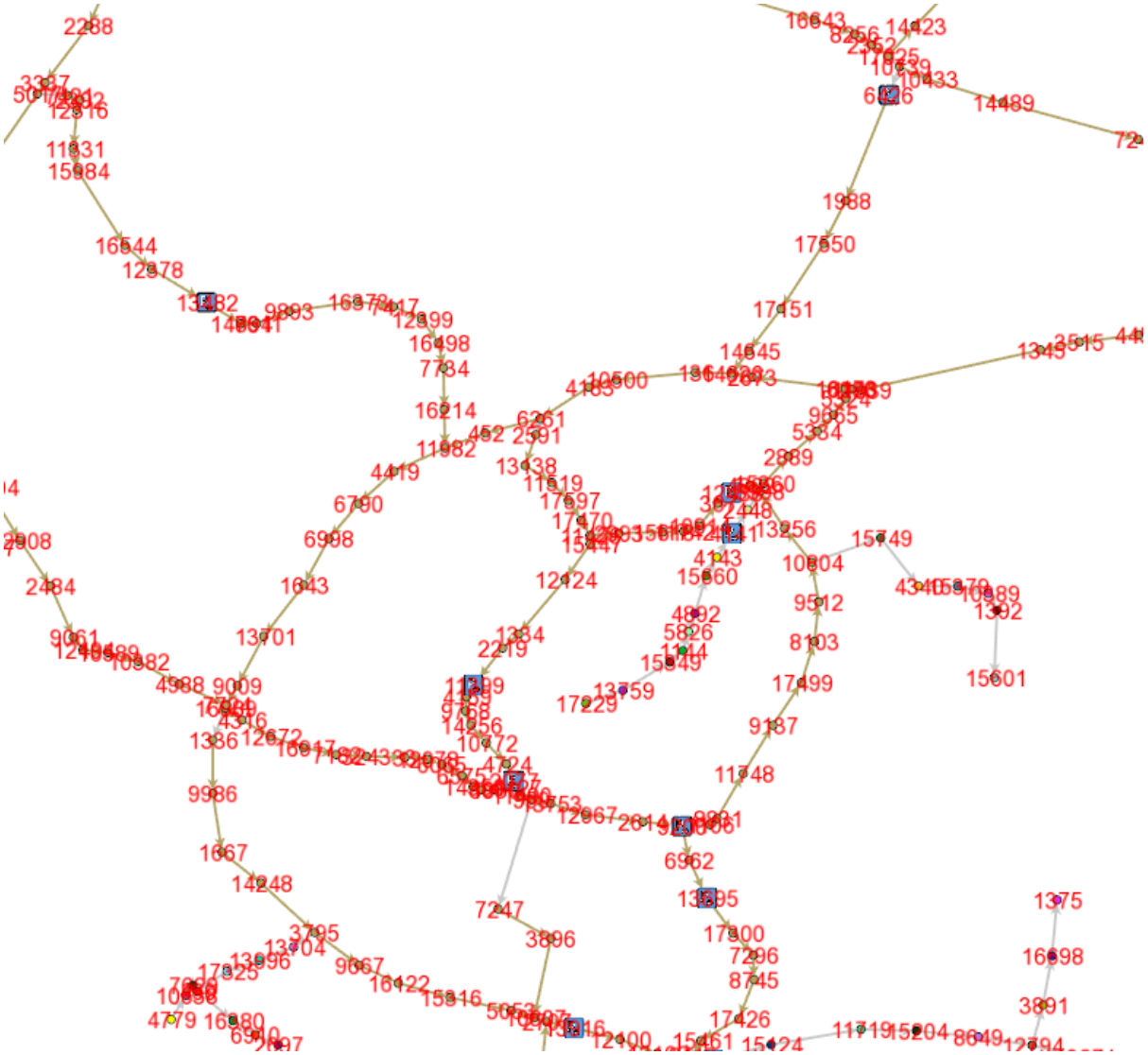
Quadrado azul -> Parque de estacionamento próximo do destino

Exemplo “Conectividade do mapa”:



Cada cor diferente representa um componente fortemente conexo do grafo (as cores são geradas de forma aleatória, em alguns casos pode haver repetição de cores).

Exemplo “Display de IDs de vértices”:



Para cada vértice é mostrado o seu respectivo ID.

5 – Conclusão

Consideramos que o método apresentado neste projeto é adequado à tarefa de procurar um estacionamento, sendo capaz de encontrar o caminho mais curto entre o ponto inicial e um dos parques de estacionamento mais perto do ponto final.

Em suma, no âmbito da unidade curricular Concepção e Análise de Algoritmos, atingimos todos os principais objetivos individuais e coletivos a atingir.

De forma a melhorar este trabalho poderíamos criar a possibilidade de o utilizador adicionar diversos pontos intermédios, calculando o percurso com a distância mínima entre cada um deles. Uma outra melhoria seria substituir a pesquisa do parque de estacionamento (que é feita com um algoritmo de pesquisa em largura) por um algoritmo ganancioso (por exemplo) que tenha em consideração a distância de cada uma das arestas.

5 – Bibliografia

- UC: Concepção e Análise de Algoritmos. Disponível em: <https://moodle.up.pt/course/view.php?id=1659>. Acesso em: 13 abril 2021.

6 – Esforço individual

João Martins - up201706978@edu.fe.up.pt

Esforço Dedicado: 1/3

Nuno Castro - up202003324@edu.fe.up.pt

Esforço Dedicado: 1/3

Ricardo Cardoso - up201604686@edu.fe.up.pt

Esforço Dedicado: 1/3

De uma forma generalizada todos os elementos participaram em todas as tarefas do projeto, pelo que não é fácil especificar o que cada elemento fez exatamente.