U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

18 de Maio de 2017

BIKE SHARING: SISTEMA DE PARTILHA DE BICICLETAS

Conceção e Análise de Algoritmos

Turma 3, Grupo G
Margarida Xavier Viterbo
up201403205@fe.up.pt

Ângelo Daniel Pereira Mendes Moura
up201303828@fe.up.pt

Duarte Pinto Valente
up201504327@fe.up.pt

Índice

1. Descrição do tema

Atualmente, os grandes centros urbanos estão em processo de adaptação para *Smart Cities*. Pelo que se procuram soluções de mobilidade, de custo reduzido e baixo impacto ambiental, que satisfaçam as necessidades da população das várias metrópoles.

Uma destas soluções passa pela criação de sistemas de partilha de bicicletas. Para tal, são criados pontos de partilha espalhados pela cidade, nos quais é possível alugar uma bicicleta (através de pagamento eletrónico) que poderá ser devolvida em qualquer outro ponto de partilha, desde que o mesmo não esteja lotado.

O presente trabalho consiste na implementação um sistema capaz de informar o utente do ponto de partilha mais próximo (bem como o caminho desde o sítio onde está o utente até a esse mesmo ponto de partilha), de acordo com o critério de preferência da pessoa.

Os critérios abrangem 6 possibilidades diferentes. As primeiras 3 são relativas à distância a percorrer para deixar o bicicleta, em que o utilizador pode pedir o caminho mais curto no plano, o caminho mais curto em termos de elevações do terreno e o caminho mais curto tendo em conta a distância e a topografia em simultâneo. As restantes possibilidades de pesquisa são relativas a incentivos a devoluções efetuadas em lugares mais afastados do centro da cidade ou de difícil acesso (sítios elevados). Quem optar por deixar a bicicleta nestes sítios será recompensado com descontos, de modo a evitar a escassez de veículos em tais locais. O utilizador pode, no entanto, escolher usufruir desconto por optar por um local mais longe, mais elevado ou pela junção dos dois, usufruindo do maior desconto possível.

A pesquisa das ruas pode ser feita de duas formas, pesquisa exata e pesquisa aproximada. Na pesquisa exata é procurado no grafo uma rua que contenha o *input* do utilizador na totalidade (este tipo de pesquisa é *case sensitive*). Quanto à pesquisa aproximada, são procuradas ruas que contenham o *input* do utilizador mas possibilitando ter letras a mais, trocas de letras e letras apagadas (esta pesquisa não é *case sensitive*).

2. Formalização do problema

2.1. Dados de entrada

Como dados de entrada no programa são utilizados mapas que representam localizações reais, obtidos no OpenStreetMaps (OSM - www.openstreetmaps.org). Os ficheiros exportados a partir do OSM estão escritos em XML, pelo que se usa um *parser* para transformar essa informação em texto, de forma a facilitar a leitura dos dados.

O *parser* origina três ficheiros de texto, cada um contendo informações necessárias para a elaboração de um grafo que representa a área geográfica extraída, segundo a seguinte tabela.

Ficheiro	*Informação do grafo (estrutura)*	*Equivalência real*
FicheiroA.txt	Nós (id, latitude, longitude)	Locais
FicheiroB.txt	Arestas (id, nome, sentido)	Estradas
FicheiroC.txt	Conexões (idAresta, idNó1, idNó2)	Conexões entre estradas

O grafo G obtido pode ser declarado da seguinte forma: $G = (N, A)$. N é o conjunto dos pontos geográficos relevantes ao mapa e A é o conjunto das secções de estrada que interligam esses pontos.

2.2. Restrições e principais dificuldades

No programa são usados mapas que representam uma área geográfica relativamente pequena, pois o grafo torna-se exponencialmente mais complexo quanto maior for o mapa usado.

Assim, uma limitação desta aplicação é que nem sempre será encontrado um caminho que siga todas as condições estabelecidas. Isto acontece porque a rota que cumpriria essas restrições está fora da área coberta pelo programa.

Algo que foi mais complexo elaborar foi a o cálculo de pesos de arestas personalizado a cada tipo de critério de escolha do caminho. Foi também especialmente trabalhoso achar a forma mais correta de implementar a pesquisa aproximada.

2.3. Solução e função objetivo

Com o objetivo de adicionar funcionalidades e permitir melhor interação com o utilizador é mostrado no mapa (desenho do grafo) o centro da cidade e os pontos de partilha de bicicletas. Quando pesquisado um caminho esse é também visível no mapa (desenhado a verde) bem como o ponto de localização atual do utilizador. Para melhorar ainda mais o acesso a informação consta também no mapa o número de lugares disponíveis em cada *Sharing Point* e o id de cada local (nó do grafo).

Por outro lado com o objetivo de simular um cenário real, em que há zonas inacessíveis porque não se encontram em ruas ou as ruas não tem sentido de trânsito que permita lá chegar, são também representadas no mapa as ruas, com os respetivos nomes e sentidos.

2.4. Resultados esperados

Esta aplicação tem como funcionalidade auxiliar o utilizador na escolha do *Sharing Point* que mais se adequa às suas necessidades, bem como melhor percurso para lá chegar.

Assim, o utilizador pode selecionar um dos mapas disponíveis, acedendo depois a um menu de navegação. Aqui tem a opção de visualizar o mapa escolhido, pesquisar um ponto de partilha/caminho ou pesquisar um ponto de partilha dando duas ruas.

Em relação à pesquisa de um caminho, o utilizador seleciona o ponto do mapa onde se encontra (indicando o id desse ponto), podendo ainda escolher qual o seu critério de escolha de percurso. Após introduzidos estes dados é automaticamente mostrado o mapa com o melhor percurso assinalado.

Na pesquisa das ruas e identificação da existência ou não de ponto de partilha, o utilizador pode escolher um modo de pesquisa para cada uma das ruas, tendo disponíveis a pesquisa exata e a pesquisa aproximada. Após selecionadas as duas ruas o sistema devolve a informação indicando se as ruas se cruzam e, em caso afirmativo, indicando se o cruzamento entre as duas possui um ponto de partilha e quantos lugares disponíveis tem. No fim da pesquisa é ainda possível observar as estatísticas relativas ao tempo de execução e ao numero de execuções de cada algoritmo.

3. Divisão e estudo do problema

A abordagem utlizada foi baseada no objetivo final deste projeto: utilizar algoritmos de pesquisa em grafos.

Assim começou-se por elaborar o grafo e um algoritmo de pesquisa, testando-se os resultados. Depois foram criados *parsers* para os ficheiros de texto com a informação dos locais, definindo paralelamente a classe Local. Após o essencial feito foi concretizada a implementação do *GraphViewer* para mostrar o mapa e implementada a pesquisa exata e aproximada de *strings*. Por fim, foi criado um menu com todas as funcionalidades e interligando todos os elementos já construídos no trabalho.

4. Descrição da Solução

4.1. Algoritmo aplicado em termos de consulta

A estrutura do grafo que representa o mapa do *OpenStreetMaps* tende a ter maior profundidade que largura. Isto porque há um vértice para cada oscilação significativa da longitude ou da latitude ao longo de um determinada rua, mesmo que nela não hajam cruzamentos nem entroncamentos. Assim uma grande parte dos vértices tem apenas um vértice-filho, já que representam a continuação da mesma estrada e não a interseção com outras.

Para facilitar a procura de um local no grafo foi criado um vetor de apontadores para vértices onde é adicionado/removido um apontador de cada vez que e adicionado/removido um vértice do grafo. A complexidade temporal e espacial deste tipo de consultas é linear – O(n).

4.2. Aplicação do peso às arestas

A principal função do nosso programa é encontrar o menor percurso entre dois locais, sendo que este percurso nem sempre é literalmente o menor, mas sim o melhor de acordo com o critério escolhido pelo utilizador. No grafo importado a partir do *OpenStreetMaps*, as arestas não têm peso, pelo que a distância geográfica entre dois pontos não tinha influência no cálculo da trajetória.

Como mencionado antes, vão ser usadas fórmulas diferentes para o cálculo dos pesos consoante a preferência de quem procura um ponto de partilha para deixar a bicicleta.

1. <u>Caso em que o utilizador quer percorrer a menor distância plana:</u>

Seja a aresta A a que liga o nó $N1$ ao nó $N2$, o peso atribuído a essa aresta é:

$$W_A = \sqrt{(N2.latitude - N1.latitude)^2 + (N2.longitude - N1.longitude)^2}$$

Ou seja, a distância cartesiana entre os dois pontos em 2D, em que as coordenadas são a latitude e longitude.

2. <u>Caso em que o utilizador quer percorrer a menor distância vertical:</u>

Seja a aresta A a que liga o nó $N1$ ao nó $N2$, o peso atribuído a essa aresta é:

$$W_A = |N2.altura - N1.altura|$$

Ou seja, a diferença de alturas entre dois pontos.

3. <u>Caso em que o utilizador quer percorrer a menor distância total:</u>

Seja a aresta A a que liga o nó $N1$ ao nó $N2$, o peso atribuído a essa aresta é:

$$W_A = \sqrt{(N2.latitude - N1.latitude)^2 + (N2.longitude - N1.longitude)^2 + (N2.altura - N1.altura)^2}$$

Ou seja, a distância cartesiana entre os dois pontos em 3D, em que as coordenadas são a latitude e longitude.

4. <u>Caso em que o utilizador quer obter um desconto sacrificando apenas a distância percorrida em 2D:</u>

Seja a aresta A a que liga o nó $N1$ ao nó $N2$, o peso atribuído a essa aresta é:

$$W_A = 1 \div |N2.difDistânciaCentro - N1.difDistânciaCentro|$$

Ou seja, o inverso da diferença entre as distâncias de cada um dos vértices ao centro.

5. <u>Caso em que o utilizador quer obter um desconto sacrificando apenas a diferença de alturas:</u>

Seja a aresta A a que liga o nó $N1$ ao nó $N2$, o peso atribuído a essa aresta é:

$$W_A = 1 \div |N2.difAlturaCentro - N1.difAlturaCentro|$$

Ou seja, o inverso da diferença entre as alturas de cada um dos vértices ao centro.

6. <u>Caso em que o utilizador quer obter o maior desconto, não interessando o caminho que terá de percorrer:</u>

Seja a aresta A a que liga o nó $N1$ ao nó $N2$, o peso atribuído a essa aresta é:

$$W_A = 1 \div \sqrt{(N2.difDistânciaCentro - N1.difDistânciaCentro)^2 + (N2.difAlturaCentro - N1.difAlturaCentro)^2}$$

Ou seja, o inverso da distância cartesiana em 2D, em que as coordenadas são a diferença da distância e da altura dos locais ao centro.

4.3. Algoritmo para encontrar o caminho mais curto

Ao aplicar-se peso a cada aresta do grafo, torna-se necessário aplicar um algoritmo que obtenha o caminho mais curto entre dois vértices, tendo em conta esse peso.

O algoritmo de Dijkstra foi o que escolhemos para calcular esse caminho, uma vez que calcula o caminho mais curto entre dois vértices em grafos dirigidos e com pesos positivos e que incorpora no seu cálculo o peso das arestas entre os diferentes vértices.

Considera-se um conjunto S de caminhos mais pequenos, iniciado com um vértice inicial I. A cada passo do algoritmo procura-se nas adjacências dos vértices pertencentes a S aquele vértice com menor distância relativa a I e adiciona-o a S e, então, repetindo os passos até que todos os vértices alcançáveis por I estejam em S.

Este é um algoritmo ganancioso, com complexidade O(ELogV).

4.4. Algoritmo para encontrar o ponto de partilha mais próximo

Após ser corrido o algoritmo de Dijkstra para atribuir os valores de distâncias aos vértices é corrido o seguinte algoritmo, que compara as distâncias de todos os vértices e guarda o vértice com menor ou maior distância, conforme o critério do utilizador. Este algoritmo apresenta uma complexidade O(n).

Pseudo-código:

```
if(critério de pesquisa é menores distâncias){
    for(todos os vértices){
        if(é ponto de partilha && tem lugares livres && distância menor que min) {
            min = distância;
            id = id do vértice;
            vertice destino = vertice;
        }
    }
}

if(critério de pesquisa é descontos){
    for(todos os vértices){
        if(é ponto de partilha && tem lugares livres && distância maior que max) {
            max = distância;
            id = id do vértice;
            vertice destino = vertice;
        }
    }
```

4.5. Algoritmo para comparação exata de duas *strings*

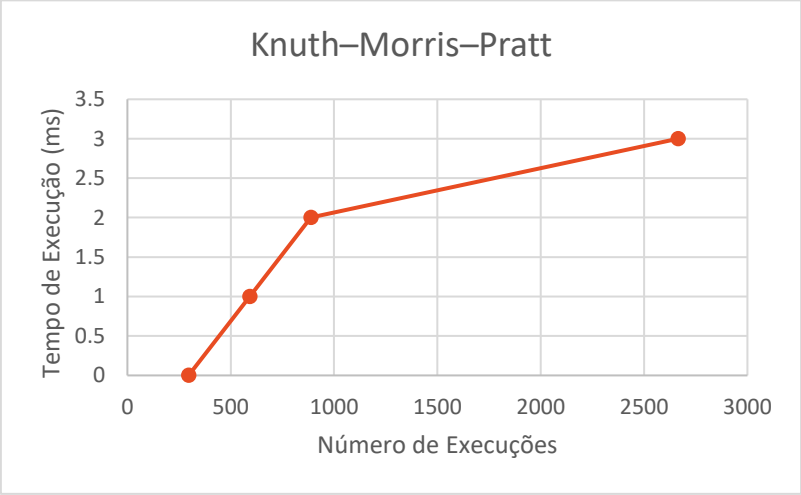
Para realizar a comparação de duas *strings* tendo em consideração a correspondência exata entre todos os caracteres da primeira na segunda (sendo que a segunda poderia ser maior mas teria de conter a expressão exata presente na primeira) foi utilizado o algoritmo de Knuth–Morris–Pratt.

Procura ocorrências de uma palavra dentro de um texto. Quando ocorre uma não-correspondência, a palavra já contém em si informação suficiente para determinar onde a próxima correspondência poderá começar, ultrapassando assim o reprocessamento de caracteres que já tinham correspondência.

Deu-se preferência a este relativamente a outras opções pois este algoritmo tem em conta a sua elevada eficiência: a abordagem de pré-processamento que este usa evita comparações triviais e evita reprocessamento de correspondências já encontradas. Além disso é um algoritmo com complexidade temporal linear, o que é bastante baixo.

- O(m) – para processar os valores da função de prefixos
- O(n) – para comparar o padrão fornecido com o texto
- Total: O(n + m) tempo de processamento

A analise do desempenho temporal pode ser observada no seguinte gráfico composto por tempos de processamento medidos na execução deste projeto:

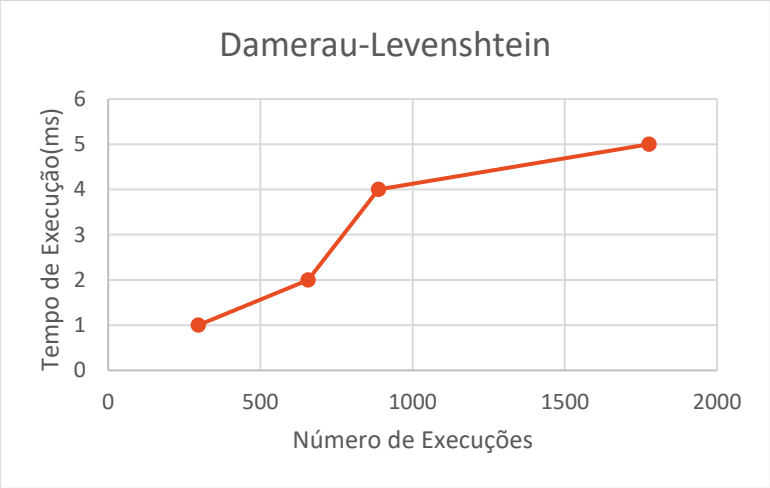


4.6. Algoritmo para obter uma medida de comparação entre duas *strings*

Para comparar a diferença entre duas *strings* de forma a poder ordena-las por grau de similaridade foi utilizado o algoritmo da distância de Damerau-Levenshtein.

Este algoritmo dá uma unidade métrica de *strings* para medir a distância de edição entre duas *strings*. A distancia de Damerau-Levenshtein entre duas palavras é o numero mínimo de operações necessárias para transformar uma palavra noutra. Este algoritmo foi preferido ao algoritmo da distancia de Levenshtein pois inclui transposições de mais do que um caracter, para alem das operações de um só caracter (inserção, remoção e substituição).

A complexidade temporal deste algoritmo é O(n.m). A análise empírica desta complexidade pode ser observada no gráfico abaixo, onde se encontram valores resultantes da execução deste projeto.



4.7. Algoritmo para pesquisa exata de strings

A função elaborada para realizar a pesquisa exata de *strings* no mapa utiliza o algoritmo de Knuth–Morris–Pratt para determinar se existe alguma ocorrência da *string* com o nome da rua introduzida pelo utilizador em cada uma das ruas do mapa.

O funcionamento desta pesquisa é o seguinte:

```
for(todos os vértices){
    for(todas as ruas do vértice){
        if(existe pelo menos uma correspondencia){
            inserir e rua no vetor de resultados;
        }
    }
}
```

4.8. Algoritmo para pesquisa aproximada de strings

Para fazer a pesquisa aproximada de uma rua no mapa foi utilizado o algoritmo da distância de Damerau-Levenshtein para obter uma medida de similaridade entre a *string* introduzida pelo utilizador e as ruas do mapa. Esta função de pesquisa encontra se explicada abaixo:

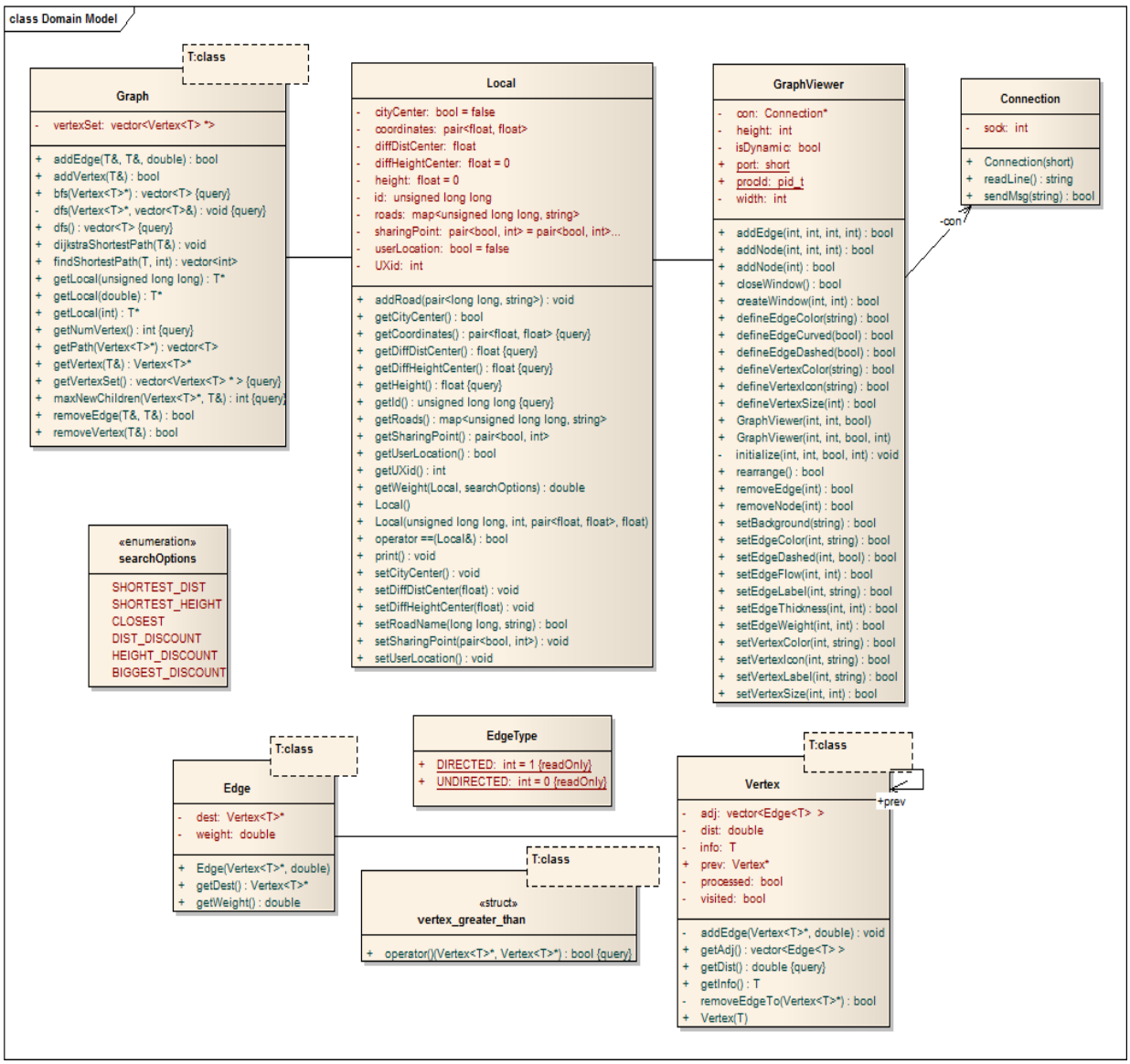
```
do{
    distancia_maxima++;
    for(todos os vértices){
        for(todas as ruas do vértice){
            distancia = damerau_levenshtein_distance(str_input, rua);
            distancia_final = distancia/numero de palavras da rua;
            if(distancia_final < distancia_maxima){
                inserir pair(rua, distancia_final) em matches;
            }
        }
    }
}while(tamanho mathes é 0);

while(matches nao está vazio){
    determinar elemento com distancia_final mais pequena;
    inserir elemento no vetor dos resultados;
    apagar elemento de matches;
}
```

4.9. Algoritmo para pesquisa de cruzamento

A função usada para determinar se existe um cruzamento entre as duas ruas selecionadas pelo utilizador percorre todos os vértices do grafo até encontrar um vértice que contenha as duas ruas.

```
for(todos os vértices){
    for(todas as ruas do vértice){
        if(rua == rua1){
            found1 = true;
        }
        if(rua == rua2){
            found2 = true;
        }
    }
    if(found1 e found2 são verdade){
        retornar este vertice;
    }
}
```

5. Diagrama de classes



6. Casos de Utilização

Segue-se uma listagem das diferentes funcionalidades do programa:

- Visualização do mapa

A representação visual do mapa é tratada por uma *GraphViewer* API, cujo projeto nos foi fornecido, que usa a *framework* JUNG (documentação disponível em http://jung.sourceforge.net/) .

- Procurar um ponto de partilha a partir de posição no mapa

Indicando ao sistema o ID do ponto onde o utilizador se encontra e o critério de pesquisa que quer utilizar o sistema mostra no mapa o caminho para chegar ao ponto de partilha que cumpre os requisitos impostos.

- Procurar um ponto de partilha no cruzamento de duas ruas

Indicando duas ruas as sistema e respetivo modo de pesquisa, o sistema indica a existência ou não de um ponto de partilha no cruzamento das mesmas e os lugares disponíveis.

7. Conclusão

O objetivo do trabalho era encontrar o melhor caminho para chegar a um ponto de partilha onde fosse possível deixar uma bicicleta alugada. Com a realização deste projeto foram cimentados os conceitos teóricos abordados nas aulas bem como estruturas de dados e algoritmos experimentados anteriormente.

As metas propostas foram alcançadas com sucesso e rigor, dentro do tempo previsto.

A organização e divisão de tarefas foi efetuada de forma pouco rígida, pelo que o projeto foi desenvolvido de acordo com o ritmo de trabalho e disponibilidade de cada elemento do grupo. Todos os elementos apresentaram empenho e dedicação no desenvolvimento das suas tarefas.