|  |
| --- |
| http://talkabit.org/assets/images/partners/feup.png |
| Supermercado ao Domicílio |
| Tema 6 – Parte 2 |
|  |
|  |
|  |

|  |
| --- |
|  |

Turma 2MIEIC01 - Grupo A

Tiago Lascasas dos Santos - up201503616

Leonardo Gomes Capozzi - up201503708

Ricardo Miguel Oliveira Rodrigues de Carvalho – up201503717

**19/05/2017**

Índice

[Descrição do Tema 2](#_Toc483155938)

[Descrição das Soluções 3](#_Toc483155939)

[1. Pesquisa Exata de *strings* 3](#_Toc483155940)

[2. Pesquisa Aproximada de *strings* 4](#_Toc483155941)

[Considerações sobre a implementação prática das soluções 5](#_Toc483155942)

[Diagrama de Classes 6](#_Toc483155943)

[Lista de Casos de Utilização 0](#_Toc483155944)

[Principais Dificuldades 2](#_Toc483155945)

[Esforço de cada elemento do grupo 2](#_Toc483155946)

[Conclusão 3](#_Toc483155947)

[Bibliografia e referências: 4](#_Toc483155948)

# Descrição do Tema

O tema deste projeto, “Supermercado ao Domicílio”, tem como objetivo a criação de um sistema que permita a uma cadeia de supermercados gerir as entregas ao domicílio de compras feitas pela internet.

Como continuação do projeto anterior, esta segunda parte do projeto acrescenta a funcionalidade de pesquisa por *strings*. Utilizando mapas semelhantes aos usados na primeira entrega, foi possível associar nomes às cadeias de supermercados e também às ruas e respetivo distrito.

Os nomes das ruas e supermercados foram então utilizados para pesquisas exatas e aproximadas, por forma a facilitar ao utilizador a localização de pontos de interesse através da morada. Neste projeto foi considerado que cada mercado se encontra no cruzamento de duas ruas sendo que a sua morada é dada pelas ruas adjecentes.

# Descrição das Soluções

A partir da segunda parte do enunciado, foram isolados dois problemas, pesquisa exata de *strings* e pesquisa aproximada de *strings*, os quais se encontram seguidamente detalhados.

## 1. Pesquisa Exata de *strings*

O algoritmo utilizado foi o algoritmo de Knuth-Morris-Pratt.

* **1ª parte** – Começa-se por declarar uma array de integers “t” de tamanho igual à string que estamos à procura, chamada “pattern”.
  + A seguir percorre-se a palavra inteira e e em cada posição de pattern escrevemos em “t” o índice a partir do qual se deve continuar a pesquisar caso não coincida a letra do texto com a letra do pattern.
* **2ª parte** – Percorre-se a palavra em paralelo com o texto verificando se cada letra é igual nos dois. Caso alguma letra falhe procura-se na array “t” o indice de “pattern” a partir do qual deveremos continuar a pesquisa. Se todas as letras da palavra coincidirem com as letras do texto quer dizer que esta foi encontrada, mas se for percorrido o texto todo sem fazer coincidir a última letra da palavra quer dizer que esta não existe no texto.

## 2. Pesquisa Aproximada de *strings*

Por forma a determinar qual o mercado ou rua com nome mais próximo a um padrão introduzido pelo utilizador, é utilizado um algoritmo de distância baseado na distância de Levenshtein (número de operações de deleção, inserção ou substituição necessárias para a partir de uma string obter outra). Após obter a distância do padrão a cada *string* num vetor dado (nomes de ruas ou mercados) as *strings* são associadas à respetiva distância e introduzidas numa fila de prioridades, que é devolvida pela função, ordenada de forma que a menor distância esteja no topo. Este processo pode, então, ser dividido em diversos pontos:

* **Input inicial** – texto, padrão, *caseSensitive*. Sendo;
  + texto – um vetor com as *strings* que se pretende comparar.
  + padrão – a *string* que se usa como referência para a comparação.
  + *caseSensitive* – valor booleano para definir se a comparação de carateres distinguirá, ou não, letras maiúsculas de minúsculas.
* **1ª parte** – é percorrido o vetor texto, sendo cada *string*, t, usada como input no algoritmo de distância, juntamente com o padrão, p, e *caseSensitive*.
* **2ª parte** – é corrido o algoritmo de distância, com complexidade espacial e complexidade temporal .
  + Inicialmente é criado um *array*, d, com o comprimento que é preenchido com valores sequenciais de 0 a .
  + De seguida, para todos os valores de i desde 1 até (inclusive) é atribuído a d[0] o valor de i e são percorridos os valores de j desde 1 até (inclusive)
    - São, então, comparados os conteúdos de t e p nos índices j-1 e i-1, respetivamente, tendo em conta se a comparação deve distinguir letras maiúsculas de minúsculas. O resultado desta comparação influencia um parâmetro, *substitutionCost*, que é 0 caso seja verdade e 1 caso contrário.
    - A partir daqui é feita a seguinte atribuição:
      * . Se:
        + , é realizada uma deleção
        + , é realizada uma inserção
        + , é realizada uma substituição, sendo last\_diagonal o valor de na iteração anterior.
  + No final, o resultado é o valor na última posição de d, isto é,

## Considerações sobre a implementação prática das soluções

/\*A implementação das soluções segue as descrições acima apresentadas, sendo que poderá ter sido acrescentada complexidade extra, principalmente para fins de output e apresentação de informação. Contudo, consideramos que o overhead extra introduzido nestes detalhes é desprezável quando comparado com as soluções em si.

É também de notar que a distância física entre dois vértices (ou seja, o peso da aresta que os liga) foi calculada a partir das coordenadas geográficas de cada vértice a partir da fórmula de haversine.\*/

# Diagrama de Classes

# Lista de Casos de Utilização

Nesta parte do trabalho foi acrescentada uma opção no menu principal que engloba as funções abaixo descritas:

1. Pesquisa de ruas/mercados
   1. Pesquisa exata por rua
      * Funcionalidade que, com recurso ao algoritmo de pesquisa exata acima descrito, procura o nome de uma rua no grafo e, caso encontre, lista os mercados adjacentes (caso haja).
   2. Pesquisa exata por mercado
      * De forma semelhante à pesquisa por rua, procura entre os mercados e, se encontrar alguma correspondência, considerando que os mercados estão em esquinas de ruas, indica as ruas adjacentes a este.
   3. Pesquisa aproximada por rua
      * Pesquisa no grafo as ruas com nomes semelhantes ao dado listando-as por ordem decrescente de proximidade ao padrão dado, como descrito anteriormente.
   4. Pesquisa aproximada por mercado
      * Semelhantemente à pesquisa aproximada por rua, é efetuada uma pesquisa nos mercados, resultando na listagem destes por ordem decrescente de proximidade.

# Principais Dificuldades

/\*No decorrer do trabalho, deparámo-nos com dois problemas principais: o primeiro prendeu-se com o facto já mencionado de não ser possível usar uma Árvore de Expansão Mínima, o que nos levou a reavaliar grande parte do projeto e a adotar uma abordagem totalmente nova sobre o problema. O segundo problema deveu-se à API do GraphViewer facultada: sempre que se desenhava um novo grafo, havia resíduos do grafo anterior que se intrometiam no novo grafo, o que levava a resultados inconsistentes. Após inúmeras tentativas de solucionar o problema, resolveu-se criar um novo processo do GraphViewer por cada grafo visualizado (matando o processo anterior antes de criar um novo), o que, apesar de não ser a mais ideal das soluções, resolveu impecavelmente o problema \*/

# Esforço de cada elemento do grupo

/\*

Tiago Lascasas dos Santos – 40%

Leonardo Gomes Capozzi – 30%

Ricardo Miguel Oliveira Rodrigues de Carvalho – 30%\*/

# Conclusão

/\*Após a realização deste trabalho, concluímos que existem várias maneiras de abordar um mesmo problema, e que a decisão sobre qual a melhor maneira a adotar para o resolver da forma mais eficiente e apropriada pode ser tanto ou mais difícil do que a própria implementação da solução. No contexto concreto deste projeto, chegámos à conclusão que a solução apresentada para o último ponto (que é o cerne do trabalho) poderia ter sido realizada mais eficientemente se o grafo fosse não dirigido ou se tivéssemos a hipótese de usar algoritmos como os caminhos ou circuitos de Euler. A solução apresentada fornece uma boa solução para casos em que tanto o número de camiões como o número de clientes é alto, visto que quantos mais clientes houver, mais hipóteses há de um caminho passar por muitos clientes e que quantos mais camiões houver, mais eficientemente se consegue distribuir os vários caminhos entre eles, o que leva a um menor tempo de entrega.

Este projeto levou também a um melhor entendimento sobre os algoritmos em grafos estudados, nomeadamente o algoritmo de Dijkstra, visto que este está presente nas soluções de quase todos os subproblemas da premissa inicial. Foi também interessante usar pela primeira vez compilação condicional e a utilização de uma API de um programa a correr paralelamente ao nosso, como é o caso do GraphViewer. A utilização de dados reais sobre uma zona que conhecemos bem permitiu dar, também, uma melhor perspetiva e entendimento do contexto do trabalho.\*/

# Bibliografia e referências:

* Mark Allen Weiss, Data Structures and Algorithm Analysis in C++, 4th edition (Florida State University: Pearson, 2014), 386–399
* Slides disponibilizados no Moodle pelos docentes
* Fórmula de haversine: <http://www.movable-type.co.uk/scripts/latlong.html>
* Biblioteca ncurses: <http://www.invisible-island.net/ncurses/ncurses.html>