

Mestrado Integrado em Engenharia Informática e Computação

Conceção e Análise de Algoritmos

Central de atendimento de urgências

Turma 4 / Grupo 9

Carlos Miguel da Silva de Freitas (201504749) - up201504749@fe.up.pt

Luís Noites Martins (201503344) - up201503344@fe.up.pt

Rui Emanuel Cabral de Almeida Quaresma (201503005) - up201503005@fe.up.pt

21 de Maio de 2017

Índice

Introdução	3
Identificação do Problema	3
Descrição do Problema	4
Input.....	4
Output.....	4
Dados de entrada	5
Objetivo.....	5
Formalização do Problema	6
Input.....	6
Output.....	6
Objetivo.....	6
Solução implementada	7
Algoritmos.....	7
Funcionamento do programa.....	11
Análise da complexidade	13
Avaliação analítica da complexidade temporal	13
Avaliação analítica da complexidade espacial.....	14
Avaliação empírica da complexidade temporal.....	15
Lista de casos de utilização.....	16
Principais dificuldades.....	17
Soluções encontradas.....	17
Contribuição dos membros do grupo	18
Conclusão	19

Introdução

Este trabalho foi realizado no âmbito da Unidade Curricular de Conceção e Análise de Algoritmos do 2º ano do MIEIC.

O objetivo é recorrer aos algoritmos de cálculo de semelhança de strings, abordados nas aulas, para encontrar uma solução para um tema que nos foi proposto.

Neste relatório é descrito o problema, feita a sua formalização, bem como apresentada a solução implementada. São ainda elencadas as principais dificuldades sentidas na elaboração do trabalho, bem como as soluções encontradas para as ultrapassar.

No final, apresenta-se uma breve conclusão, que inclui uma apreciação sobre a aprendizagem decorrente deste trabalho.

Identificação do Problema

Numa situação de emergência em que há necessidade de verificar de imediato quais os veículos que se encontram numa determinada localização, existe uma dificuldade acrescida, pelo facto de, com frequência, ocorrerem erros na introdução dos dados da localização pelo utilizador.

Perante este problema, pretende-se obter uma solução que permita, em tempo útil, e mediante os dados fornecidos pelo utilizador, determinar com precisão qual a localização em causa.

Descrição do Problema

Pretende-se identificar os veículos existentes num determinado local escolhido pelo utilizador. Para tal, este escreve o nome da freguesia e da rua pretendida (string), informação que é posteriormente lida e analisada.

Input

Grafo, $G = (V, A)$, de cruzamentos e ruas em que:

- V (vértices): representam pontos de interceção de duas ruas, ou seja, cruzamentos;
- A (arestas): ligações entre dois cruzamentos (distâncias entre dois vértices);

O utilizador terá que inserir uma string, com a qual se deve verificar a semelhança com as freguesias existentes.

Terá também de introduzir uma nova string, mas desta vez para identificar a rua que pretende.

Output

Ao longo do programa vão sendo listadas as freguesias e ruas disponíveis, para que o utilizador possa saber quais pode escolher.

No fim da execução serão apresentados os veículos e respetivos nós onde se encontram, considerando a freguesia e rua escolhidas.

Dados de entrada

- Seis ficheiros que contêm os nós (cruzamentos) do mapa, com os ids, coordenadas x e y dos mesmos:
 - Um ficheiro que contém os nós onde não está localizado nenhum veículo ou hospital. Contém os ids, o x e o y dos nós;
 - Um ficheiro que contém os nós onde estão localizadas as ambulâncias;
 - Um ficheiro que contém os nós onde estão localizados os carros dos bombeiros;
 - Um ficheiro que contém os nós onde estão localizados os carros da PSP;
 - Um ficheiro que contém os nós onde estão localizados os hospitais.
 - Um ficheiro que contém as freguesias, com os respetivos ids, nomes e ids das ruas e nós que cada uma contém.
- Um ficheiro que contém as ruas, ou seja, conjuntos de arestas que podem ser unidirecionais ou bidirecionais, com os ids e o nome das mesmas, a indicação sobre se são ou não bidirecionais e ainda os ids dos nós que as compõem.

Objetivo

Providenciar ao utilizador informação sobre quais os veículos de emergência existentes na rua solicitada, tendo em conta a freguesia escolhida.

Formalização do Problema

Input

$G(V, A)$,

V : cruzamentos

A : ligações entre cruzamentos (distancia da ligacao)

String para identificação da freguesia

String para identificação da rua

Output

Veiculos existentes na rua escolhida pelo utilizador

Objetivo

Min (distancia entre a string dada e o nome das ruas/freguesias):

$$distancia = \sum_{i=1}^n (\min(A_i B_j)),$$

A_i – cada palavra da string do utilizador,

B_j – cada uma das ruas/freguesias

Solução implementada

Algoritmos

Foram implementados dois algoritmos de pesquisa em strings, um de pesquisa exata e outro aproximada, tanto para a pesquisa pelo nome das freguesias como pelo nome das ruas. Para a pesquisa exata, foi utilizado o algoritmo de Knuth-Morris-Pratt (KMP).

O algoritmo de Knuth-Morris-Pratt tem como objetivo verificar se uma dada string (padrão) está contida numa outra string (texto).

O KMP começa por realizar um pré-processamento do padrão, que serve para que o mesmo carater seja verificado mais do que uma vez. Este cria um array auxiliar com tamanho igual a $|\text{padrão}|$ que irá conter em cada posição um prefixo para cada posição do padrão (determina se os prefixos do padrão são substrings deles mesmos).

```
COMPUTE-PREFIX-FUNCTION( $P$ )
1   $m \leftarrow \text{length}[P]$ 
2   $\pi[1] \leftarrow 0$ 
3   $k \leftarrow 0$ 
4  for  $q \leftarrow 2$  to  $m$ 
5      do while  $k > 0$  and  $P[k + 1] \neq P[q]$ 
6          do  $k \leftarrow \pi[k]$ 
7          if  $P[k + 1] = P[q]$ 
8              then  $k \leftarrow k + 1$ 
9           $\pi[q] \leftarrow k$ 
10 return  $\pi$ 
```

De seguida, o algoritmo começa a percorrer a string texto e a compará-la com a string padrão. Quando os caracteres não são iguais, a pesquisa inicia-se no carater seguinte, não sendo necessário voltar a comparar os caracteres que já tenham sido igualados.

```

KMP-MATCHER( $T, P$ )
1   $n \leftarrow \text{length}[T]$ 
2   $m \leftarrow \text{length}[P]$ 
3   $\pi \leftarrow \text{COMPUTE-PREFIX-FUNCTION}(P)$ 
4   $q \leftarrow 0$                                      ▷ Number of characters matched.
5  for  $i \leftarrow 1$  to  $n$                              ▷ Scan the text from left to right.
6      do while  $q > 0$  and  $P[q + 1] \neq T[i]$ 
7          do  $q \leftarrow \pi[q]$                      ▷ Next character does not match.
8          if  $P[q + 1] = T[i]$ 
9              then  $q \leftarrow q + 1$                ▷ Next character matches.
10         if  $q = m$                                   ▷ Is all of  $P$  matched?
11             then print "Pattern occurs with shift"  $i - m$ 
12              $q \leftarrow \pi[q]$                    ▷ Look for the next match.

```

Para a pesquisa aproximada foi utilizado o algoritmo lecionado nas aulas teóricas para descobrir a distância entre 2 strings. Uma vez que o nome de cada rua tem várias palavras e o utilizador também pode introduzir mais do que uma palavra, recorreu-se então à separação de strings antes da verificação da distância entre elas.

Assim, para cada palavra introduzida pelo utilizador, percorre-se todas as ruas existentes. Cada rua é separada nas diferentes palavras que a constituem. Para cada uma destas, calcula-se a distância entre esta e a palavra do utilizador que está a ser analisada. Se esta distância for inferior à distância mínima atribuída na relação entre a palavra do utilizador e a rua em análise, então esta distância mínima é atualizada. Desta maneira preenche-se um vetor em que a cada posição corresponde uma das palavras introduzidas pelo utilizador e, para cada palavra existe um multimap, no qual a chave é a distância mínima da rua à palavra do utilizador e o valor é o nome da rua.


```

for i=0 to |split_string_user_words|
  for int j=0 to |ruas_vector|
    for int k=0 to |ruas_vector[j]_words|
      if editDistance(split_user_words[i], ruas_vector[j]_words[k]) < distmin
        update distmin
      add to map ruas_vector[j] com dist = distmin
    add map to user_words_distance_vector

for i=0 to |ruas_vector|
  for j=0 to |user_words_distance_vector|
    string_total_distance += user_words_distance_vector[j][ruas_vector[i]]
  add to final_map string_total distance with value ruas_vector[i]
return final_map

```

Para calcular a distância entre cada par de palavras (sendo a string proveniente da rua considerada como string padrão e a string do utilizador a string a ser testada) foi então utilizado o algoritmo da pesquisa aproximada, no qual é preenchida uma matriz D de largura igual ao tamanho da string a analisar (T) e altura igual ao tamanho da string padrão (P).

Esta análise é feita através da iteração da matriz e da análise dos valores já conhecidos. Assim, a primeira linha e primeira coluna são a distância de cada uma das palavras à string vazia e como tal estes valores irão de 0 até ao valor do comprimento de cada uma das strings.

Posteriormente, o algoritmo itera a matriz, começando em 1 (pois a linha e a coluna 0 já estão preenchidas), até ao tamanho de cada uma das strings. Para cada valor da matriz compara-se cada par de valores de P e T e avalia-se se estes são iguais ou não. Se forem, então é porque não existe nenhuma alteração de padrão e, portanto, o valor da matriz naquela posição é igual à que se encontra na posição diagonal desta mesma, na linha e coluna anteriores, ou seja, mantém o seu valor de distância. Se, por outro lado, estes forem diferentes, então o valor da matriz naquela posição é igual ao mínimo de entre as 3 posições que rodeiam a posição em consideração (à esquerda, em cima, e na diagonal) mais um, uma vez que é necessário considerar esta nova alteração.

Desta forma, no final do preenchimento da matriz, o valor que se encontra em $D[P.length][T.length]$ corresponde à distância mínima entre as 2 strings.

Pode-se, com tudo, fazer uma simplificação em termos espaciais a este algoritmo, armazenando apenas a informação referente à análise da string de texto, em relação a cada posição do padrão. Deste modo, quando se chegar ao fim da análise, o vetor terá os valores correspondentes a ultima linha da matriz. Começa-se então por inicializar o vetor D com as distâncias de cada posição de T à string vazia. Posteriormente itera-se a string padrão (começando na segunda posição já que a primeira seria para a distancia a stringa vazia) e utiliza-se uma variável old para saber qual é a distância mínima atual em que se encontra até aquele momento. É também utilizada também uma variável new que guardará o novo valor da posição do vetor e que irá atualizando a matriz. Assim chegando ao fim da análise, o ultimo elemento do vetor terá a distância mínima total do texto ao padrão.

```

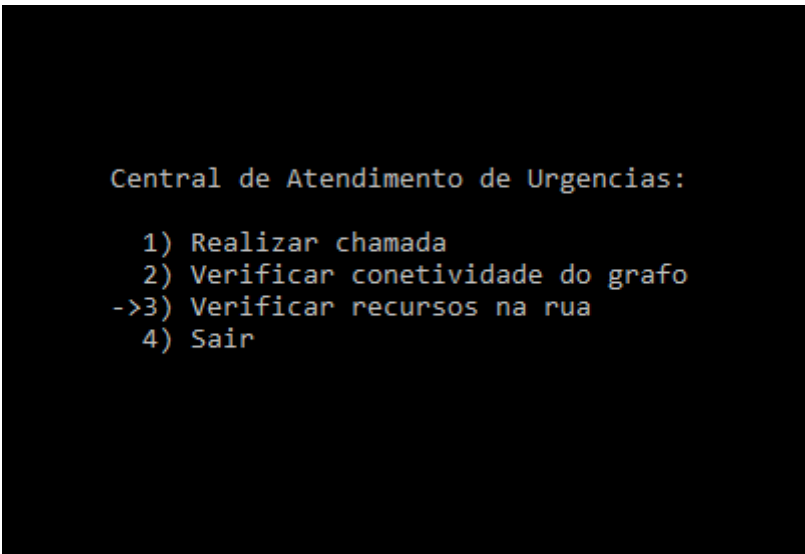
EditDistance(P,T) {
  // inicialização
  for j = 0 to |T| do D[j] = j // D[0,j]
  // recorrência
  for i = 1 to |P| do
    old = D[0] // guarda D[i-1,0]
    D[0] = i // inicializa D[i, 0]
    for j = 1 to |T| do
      if P[i] == T[j] then new = old
      else new = 1 + min(old,
        // Ainda tem valor anterior D[i-1,j]
        D[j],
        D[j-1])
      // Já tem valor da iteração corrente, i.e., D[i, j-1]
      old = D[j]
      D[j] = new
  // finalização
  return D[|T|]
}

```

Funcionamento do programa

Para implementar a solução para o problema em causa, foram utilizados os algoritmos acima referidos.

Assim, para além do funcionamento já referido no relatório do primeiro trabalho, agora o utilizador tem também a opção de verificar quais os veículos existentes numa rua através da escolha da freguesia e nome da rua.



```
Central de Atendimento de Urgencias:  
1) Realizar chamada  
2) Verificar conetividade do grafo  
->3) Verificar recursos na rua  
4) Sair
```

Após a seleção da opção de verificação de ruas, o utilizador poderá seleccionar a freguesia que pretende, escrevendo o nome da mesma.

```
LISTA DE FREGUESIAS EXISTENTES

- Alvarenga      - Chave
- Escariz        - Mansores
- Moldes         - Rossas
- Arouca         - Soutelo
- Fermedo        - Varzea
- Espiunca

Por favor insira a freguesia pretendida:
```

Após este ponto, executa-se uma pesquisa da string escrita, primeiro por verificação exata e, caso esta não exista, por verificação aproximada, retornando os valores obtidos. Neste momento, caso existam várias opções, o utilizador selecionará a pretendida pelo seu número na listagem.

```
Possíveis freguesias parecidas com o que escreveu
1 - Chave
Escolha a freguesia (atraves do seu numero):
```

Após escolher a freguesia aparecerá uma lista com o nome das ruas existentes nesta freguesia, e o utilizador poderá neste momento introduzir o nome da rua que pretende. Serão novamente executados os algoritmos de pesquisa em strings e retornados os valores obtidos.

```
Possiveis ruas parecidas com o que escreveu  
1 - Rua Egas Moniz  
2 - Rua Padre Antonio Vieira  
  
Escolha a rua (atraves do seu numero):
```

Após isto são mostrados no ecrã os veículos existentes na zona seleccionada, mostrando para cada tipo de veículos os nós onde estes se encontram.

```
Escolha a rua (atraves do seu numero): 1  
  
Nao existe INEM nesta rua  
      BOMBEIROS: 5  
Nao existem Policias nesta rua
```

Análise da complexidade

Avaliação analítica da complexidade temporal

A complexidade temporal do algoritmo de pesquisa aproximada é
 $O([nr \text{ palavras introduzidas pelo utilizador}] * [nr \text{ de ruas a analisar}] * [nr \text{ de palavras de cada rua}] * [nr \text{ de letras de cada palavra da rua}] * [nr \text{ de letras de cada palavra da rua}])$

[*nr de letras de cada palavra do utilizador*]], tal como se pode verificar pela explicação encontrada na solução implementada

Já no caso do algoritmo KMP, a sua complexidade temporal do algoritmo é $O(|\text{texto}| + |\text{padrao}|)$. Neste caso em particular, existe uma complexidade

$O([\text{numero de ruas}] * [\text{nr de palavras do texto} - \text{nr de palavras do padrao} + 1] * [|\text{texto}| + |\text{padrao}|])$.

Avaliação analítica da complexidade espacial

A complexidade espacial do algoritmo de Knuth-Morris-Pratt é $O(|\text{texto}| + |\text{padrao}|)$.

Já para o algoritmo de verificação de strings aproximadas a complexidade espacial é $O(\text{nr de palavras escritas pelo utilizador} * (\text{nr de ruas analisadas} - \text{que pertencem a freguesia escolhida}) * (|\text{texto}|))$.

Avaliação empírica da complexidade temporal

Para analisar a complexidade temporal, medimos os tempos de execução do programa, usando diferentes dados de entrada, fazendo variar o número e tamanho das strings para uma mesma rua. Nesta avaliação não foi tida em conta a leitura da freguesia uma vez que como todas têm apenas uma palavra não seria muito relevante em termos de resultados obtidos.

Para os testes e respetivos resultados, a seguir apresentados, foi utilizada como referencia a Rua Fernando Pessoa Martins existente na freguesia de Rossas.

Tendo em conta que a freguesia se mantém e o número de ruas a analisar é também constante, então o único fator variante nesta análise foi o número de palavras, e respetivo número de letras, introduzidas pelo utilizador.

Foram então obtivemos os seguintes resultados:

Input utilizador	Pesquisa aproximada (nanosegundos)
erfnando	82400
erfnando pess	117200
erfnando pessoa	256300
erfnando pessoa mrat	1002400

Pode-se concluir, através da análise da tabela, que o tempo de pesquisa aproximada aumenta com o aumento do número de palavras e numero de letras de cada palavra. Note-se que entre a segunda e a terceira tentativas, mantendo-se o número de palavras e acrescentando apenas 2 letras, houve uma grande variação no tempo de pesquisa. Assim, apesar de o aumento de palavras levar a um tempo de cálculo superior, os resultados obtidos são mais corretos.

Lista de casos de utilização

- Leitura dos dados de ficheiros representativos das componentes de um mapa;
- Identificação dos veículos existentes numa determinada rua de uma freguesia
- Visualização do mapa usando o GraphViewer;

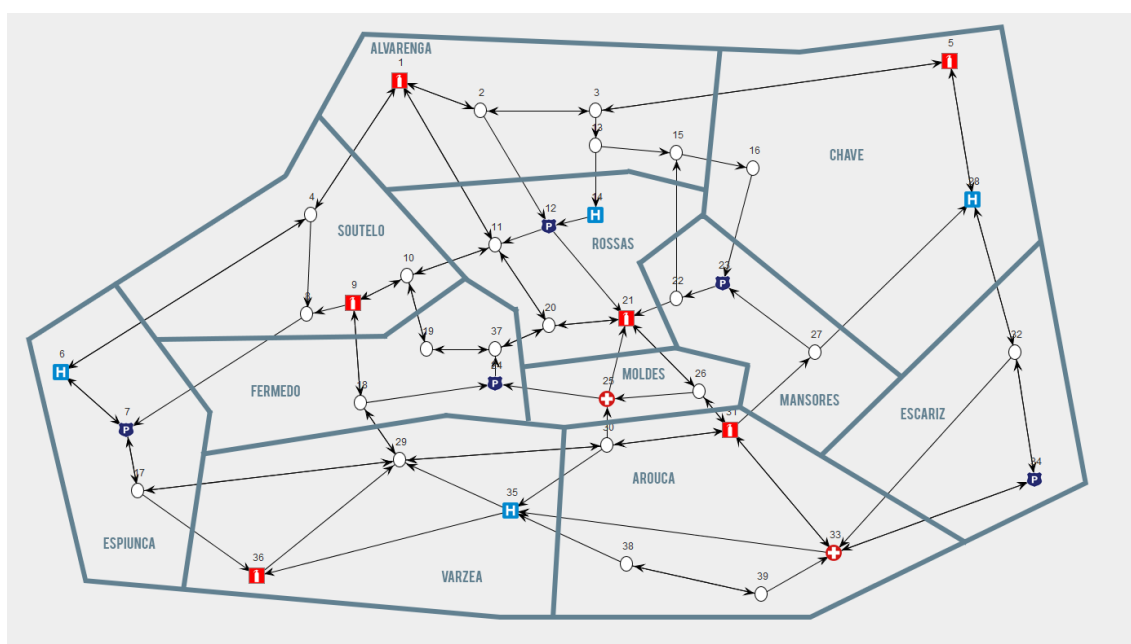
Principais dificuldades

No início do projeto, deparamo-nos com alguns entraves à sua execução, nomeadamente a dificuldade em saber qual a melhor maneira para que o algoritmo de semelhança de string por pesquisa aproximada retornasse os valores pretendidos.

Soluções encontradas

Como referido anteriormente, foram feitas algumas divisões das strings em palavras, para que as comparações fossem mais credíveis e assim se obtivessem resultados mais semelhantes ao pretendido.

Foi ainda feita uma divisão do mapa utilizado, em freguesias, sendo a divisão existente nos ficheiros aquela que se encontra na imagem abaixo.



Contribuição dos membros do grupo

Todos os membros contribuíram para a realização do projeto. A distribuição do trabalho foi a seguinte:

Carlos Freitas:

- Desenvolvimento da interface do utilizador (menus e display dos resultados).

Luís Martins:

- Implementação do algoritmo de pesquisa aproximada, incluindo a divisão de strings e possíveis condições de paragem
- Redação do relatório

Rui Quaresma:

- Criação e leitura dos ficheiros utilizados
- Implementação do algoritmo KMP
- Participação na redação do relatório

Conclusão

Perante o problema apresentado no início deste relatório, pretendia-se determinar quais os veículos disponíveis no local escolhido pelo utilizador, sendo esta escolha feita através da introdução de strings.

Foi feita a análise do input do utilizador com recurso a 2 algoritmos distintos, tendo sido obtidos resultados satisfatórios.

Conclui-se que através da verificação da diferença entre strings se consegue facilmente, e em tempo útil, obter resultados semelhantes aos pretendidos pelo utilizador. Verificou-se ainda que a análise por pesquisa exata é mais rápida. No entanto, esta nem sempre surte efeito, pelo que a pesquisa aproximada é um recurso fundamental para este tipo de casos.

O tema deste trabalho representa uma situação que, embora aqui seja analisada de forma simplificada, corresponde a uma situação real, pois a pesquisa e os resultados obtidos são algo importante para que se realize uma pesquisa rápida e correta mesmo que existam erros de input.