U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Conceção e Análise de Algoritmos

Sistema de Evacuação

2MIEIC02 – Grupo 3 – Tema 3

Luís Borges **up201605859**@fe.up.pt

Patrícia Janeiro **up201605946**@fe.up.pt

Sandro Campos **up201605947**@fe.up.pt

Índice

1. Introdução

Na segunda parte do trabalho prático da unidade curricular de Conceção e Análise de Algoritmos foi-nos proposta uma implementação, a partir da primeira parte do trabalho prático de um sistema de linha de emergência, que permitisse a inserção do nome de uma rua de forma a obter uma rota de evacuação a partir desta.

Procedeu-se assim à implementação de algoritmos de pesquisa exata e aproximada de forma a procurar a localização, fornecida pelo utilizador, no grafo.

2. Descrição do Problema

2.1 Input

Construção de um grafo G = (V, E), em que:

- Vértices (V) – representam os vários pontos num mapa de estradas.
- Arestas (E) – representam as ligações entre os vários pontos do mapa, o nome da rua, a capacidade máxima e o tráfego atual da via.
- Nós de início e de destino.

2.2 Dados de entrada

Ficheiros com o tráfego a percorrer as estradas.

2.3 Output

O percurso que deve ser realizado de um ponto inicial para um ponto final, ou todos os percursos possíveis a partir de um ponto tendo em conta a existência de algum troço fechado, apresentado o percurso realizado no menor tempo possível.

2.4 Objetivo

Apresentar o percurso que deve ser realizado para evitar um troço fechado e chegar ao destino em segurança, no menor tempo possível, processando todo o tráfego. Neste caso, na inexistência de informação relativa à velocidade média de cada troço da autoestrada, assume-se que o caminho mais rápido a ser percorrido é aquele que corresponde ao caminho de menor distância. Apresentar todos os percursos disponíveis a partir de uma rua cujo nome é introduzido.

2.5 Restrições

Evitar um troço fechado e impedir que os veículos não se dirijam para troços lotados das autoestradas, uma vez que cada um destes possui uma capacidade limitada, isto é, um número máximo de automóveis que suporta.

3. *Algoritmos e estruturas de dados*

Com o objetivo de pesquisar o nome exato da rua, optou-se pela implementação do **algoritmo de Knuth-Morris-Pratt**. Este algoritmo efetua um pré-processamento do padrão em tempo **O(|P|)** e, ao contrário do **algoritmo baseado em autómato finito**, não chega a gerar explicitamente um autómato. Depois do pré-processamento, segue-se o processamento do texto em **O(|T|)**. Somando-se o tempo das duas etapas, totaliza-se uma *performance* **O(|T|+|P|)**, muito inferior ao tempo do **algoritmo naïve**, **O(|P|*|T|)** e do **algoritmo baseado em autómato finito** que depende do tamanho do alfabeto (∑), **O(|P|*|∑|)**. Por esta razão considerou-se tanto o **algoritmo naïve** como o **baseado em autómato finito** ineficiente. No **algoritmo de Knuth-Morris-Pratt** desloca-se o padrão para a direita de uma forma que permite continuar a comparação na mesma posição de texto, evitando comparações inúteis, sendo este deslocamento determinado pela **função prefixo** (π[q]).

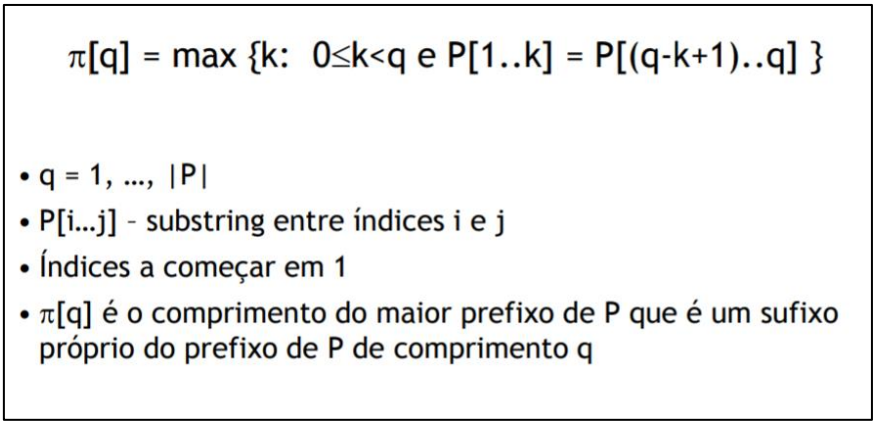
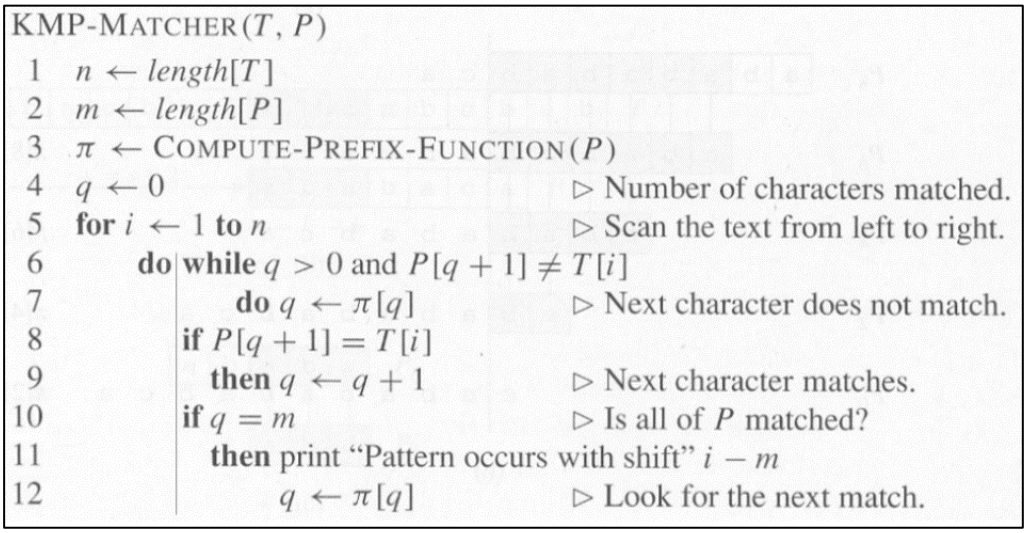
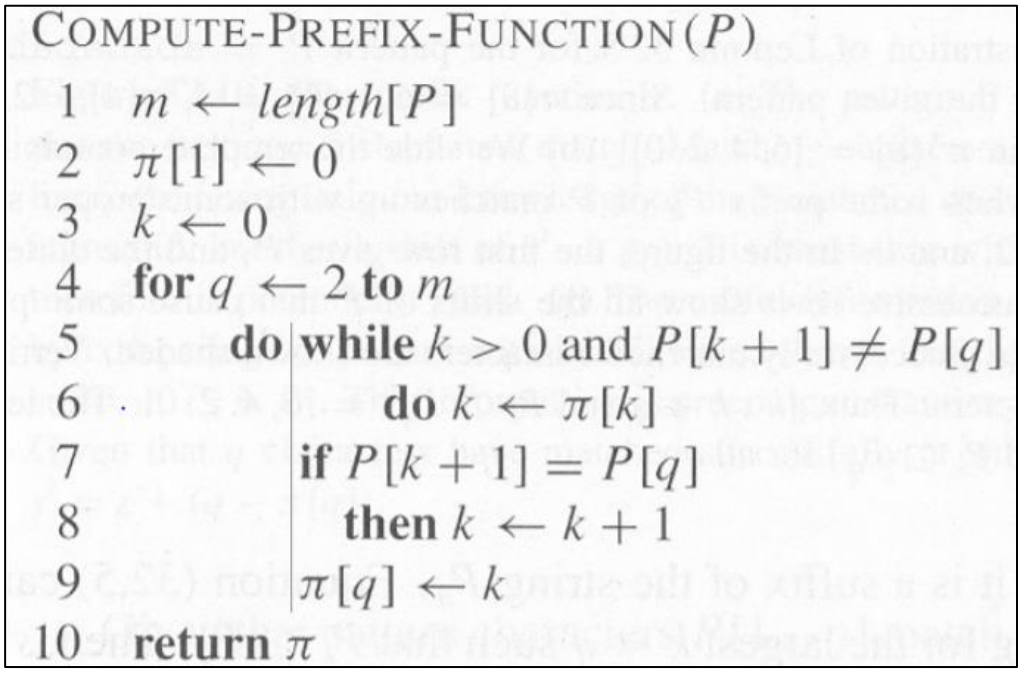


Figura 1 Definição da função prefixo



```
KMP-MATCHER(T, P)
 1  n ← length[T]
 2  m ← length[P]
 3  π ← COMPUTE-PREFIX-FUNCTION(P)
 4  q ← 0                                  ▷ Number of characters matched.
 5  for i ← 1 to n                         ▷ Scan the text from left to right.
 6      do while q > 0 and P[q + 1] ≠ T[i]
 7           do q ← π[q]                   ▷ Next character does not match.
 8         if P[q + 1] = T[i]
 9           then q ← 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 ← π[q]                 ▷ Look for the next match.
```

Figura 2 Pseudo-código do algoritmo de Knuth-Morris-Pratt



```
COMPUTE-PREFIX-FUNCTION(P)
 1  m ← length[P]
 2  π[1] ← 0
 3  k ← 0
 4  for q ← 2 to m
 5      do while k > 0 and P[k + 1] ≠ P[q]
 6           do k ← π[k]
 7         if P[k + 1] = P[q]
 8           then k ← k + 1
 9         π[q] ← k
10  return π
```

Figura 3 Pseudo-código do algoritmo da função prefixo

Para a pesquisa aproximada do nome da rua, utilizou-se o algoritmo dado nas aulas da unidade curricular. Neste algoritmo a **distância de edição** entre P (*string* do padrão a procurar) e T (*string* das ruas) é o menor número de alterações necessárias para transformar T em P, recorrendo à substituição de um caracter por outro e à inserção ou remoção de um outro. Utilizou-se o algoritmo com otimização de espaço, com eficiência **O(|T|)**.

- $D[i,j] = EditDistance(P[1..i], T[1..j])$, $0 \le i \le |P|$, $0 \le j \le |T|$
- Condições fronteira:
  - $D[0, j] = j$, $D[i,0] = i$
- Caso recursivo ($i>0$ e $j>0$):
  - Se $P[i]=T[j]$, então $D[i, j] = D[i-1, j-1]$
  - Senão, escolhe-se a operação de edição que sai mais barata; isto é, $D[i,j]$ é o mínimo de:
    - $1 + D[i-1, j-1]$ (substituição de $T[j]$ por $P[i]$)
    - $1 + D[i-1, j]$ (inserção de $P[i]$ a seguir a $T[j]$)
    - $1 + D[i, j-1]$ (eliminação de $T[j]$)

Figura 4 Distância de edição entre duas strings

```
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,
                          D[j],
                          D[j-1])
       old = D[j]
       D[j] = new
  // finalização
  return D[|T|]
}
```

Tem D[i-1,j-1]

Ainda tem valor anterior D[i-1,j]

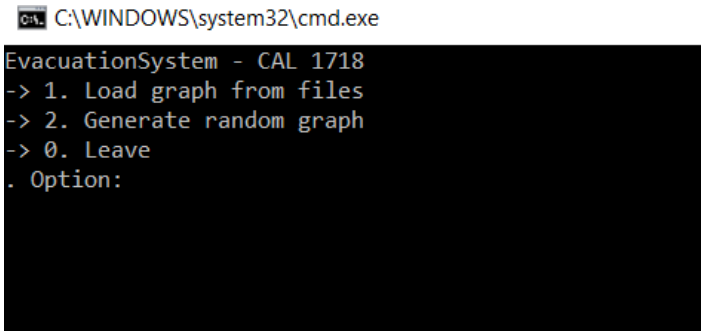
Já tem valor da iteração corrente, i.e., D[i, j-1]

Figura 5 Pseudo-código da função de distância de edição

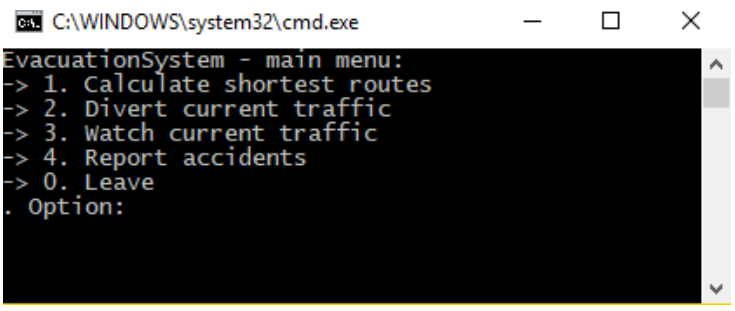
4. Casos de Utilização

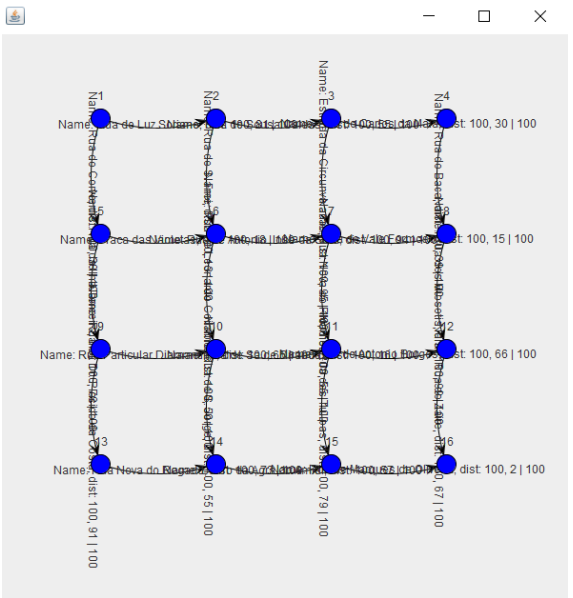
A interface do programa foi desenvolvida para a linha de comandos, sendo que todas as instruções são inseridas através desta. Todos os menus, incluindo inputs e outputs estão definidos na classe “Utils.cpp”.

Inicialmente, tal como na primeira parte do trabalho, opta-se por carregar um grafo já existente ou gerar um aleatoriamente.

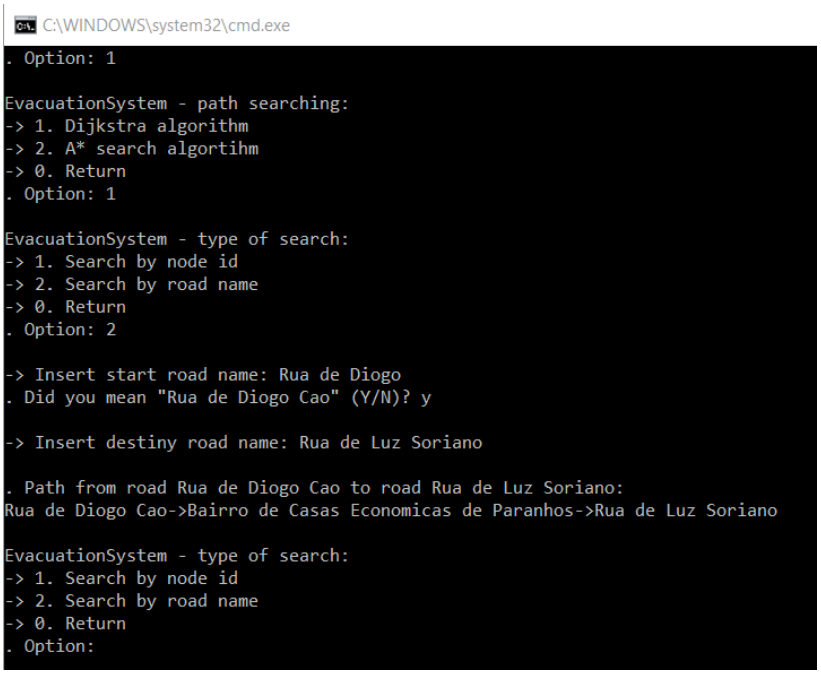


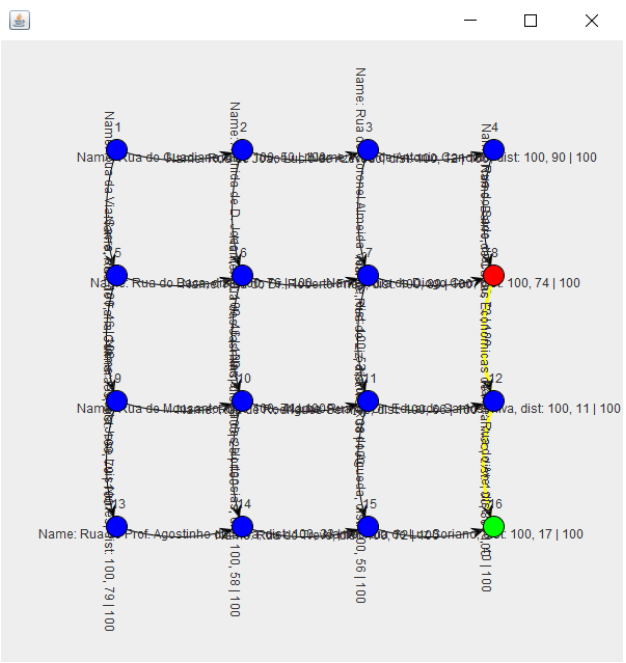
De seguida é possível selecionar uma das seguintes opções: calcular o caminho mais curto, divergir e observar o tráfego atual e reportar um novo acidente. É também apresentado o esquema do grafo, com o nome de cada rua, assim como a distância entre nós, o número de veículos atuais num troço e a sua capacidade máxima. De referir que os nomes atribuídos às arestas são selecionados aleatoriamente de um ficheiro (roads.txt).



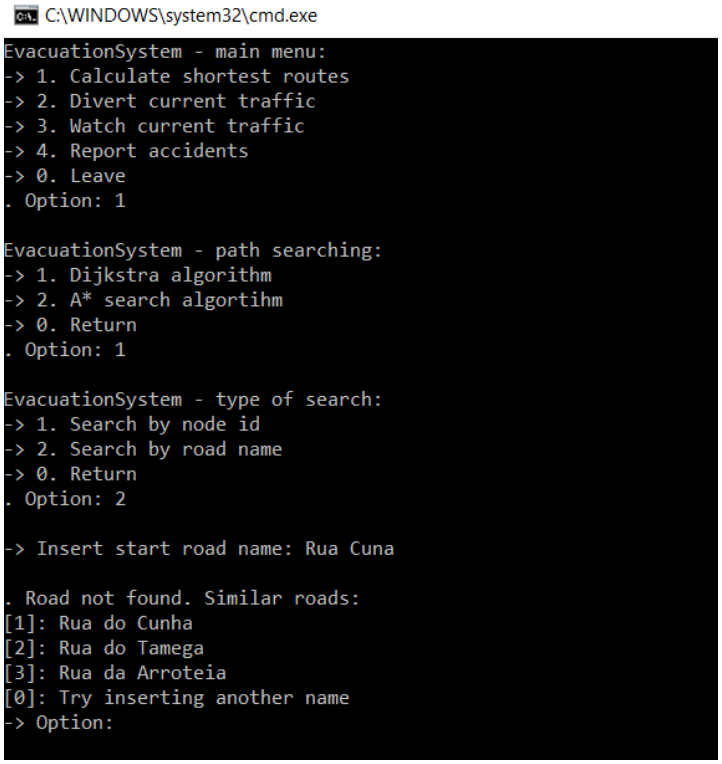


Todo os menus seguintes e as respetivas funcionalidades são semelhantes à primeira parte do trabalho prático. Por esta razão apresentam-se apenas as funcionalidades implementadas nesta parte do trabalho. Desta forma, é possível agora selecionar o nome da rua inicial e da rua de destino, ou os seus IDs respetivos. O percurso apresentado inicia-se no final da rua inicial (vértice a que é ligado) até ao início ou ao final da rua de destino. É possível pesquisar pelo nome exato da rua, ou pelo nome aproximado da mesma. Neste último caso é apresentada uma mensagem ao utilizador com os 3 nomes de ruas com maior grau de semelhança à introduzida. De seguida, o utilizador pode escolher uma destas ou reinserir outro nome para a rua.

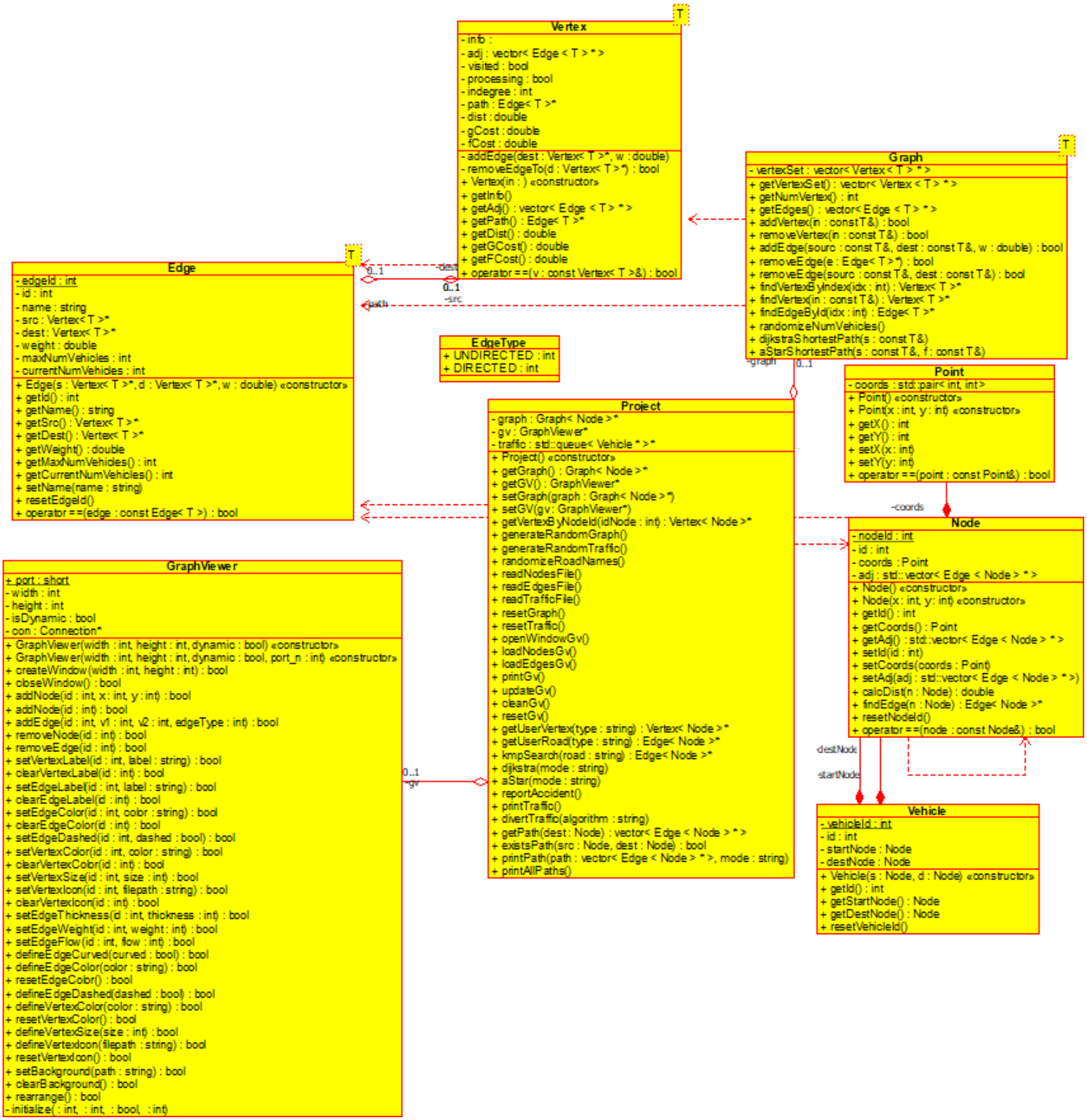




Caso o nome da rua não seja encontrado, é apresentada uma lista ao utilizador com os nomes de ruas similares.



5. *Diagrama de Classes*



6. Principais Dificuldades

Na realização da segunda parte do trabalho, e ao contrário da primeira parte, não encontrámos grandes dificuldades. Os algoritmos construídos eram, em geral, de fácil implementação.

7. Contribuição no Projeto

Consideramos que a divisão de tarefas foi a seguinte: Luís Borges - 40%, Patrícia Janeiro – 20% e Sandro Campos - 40%.

8. Conclusão

Com a realização deste projeto foi possível efetuar uma interligação entre os algoritmos de pesquisa do caminho mais curto em grafos, já implementados, com novos algoritmos e conceitos, relacionados com pesquisa exata e aproximada de *strings*, com as mais diversas aplicações no mundo tecnológico.