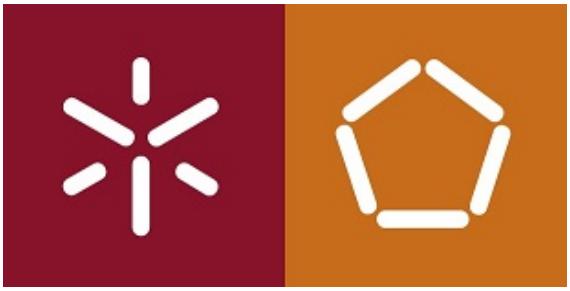

Universidade do Minho
Departamento de Informática

Inteligência Artificial
Grupo 5
Fase II

2 de dezembro, 2021


Alexandre Flores
(a93220)


Mariana Rodrigues
(a93294)


Matilde Bravo
(a93246)


Pedro Alves (a93272)

Conteúdo

Capítulo 1

Introdução

O presente relatório descreve a segunda fase do trabalho prático realizado no âmbito da Unidade Curricular de Inteligência Artificial.

Serão apresentados e devidamente justificados os predicados que foram criados para a realização de *queries* e resolução de problemas de pesquisa. Serão também demonstradas as funcionalidades extra implementadas.

1.1 Descrição do Trabalho

Esta segunda fase do projeto sobre a companhia de entregas *Green Distribution* consiste no desenvolvimento de funcionalidades de resolução de problemas de pesquisa em grafos, colocando em prática o conhecimento adquirido nas aulas sobre algoritmos de pesquisa. Estas funcionalidades serão posteriormente utilizadas para o cálculo de circuitos de entrega de encomendas. Apesar de ser dada a oportunidade de escolher a linguagem de programação a usar, escolhemos usar novamente *Prolog*.

Através dos predicados desenvolvidos na linguagem construímos uma série de ferramentas que nos permitem encontrar e comparar "circuitos"de entrega, isto é, caminhos que estafetas percorrem para entregar encomendas. Foram desenvolvidos diversos predicados de forma a implementarmos vários algoritmos de pesquisa em grafos, conforme pedido no enunciado do trabalho.

Tal como na primeira fase do projeto, a base de conhecimento pode ser manipulada dinamicamente no **swiprolog**, através das funcionalidades de introdução e remoção de conhecimento desenvolvidas. Estas funcionalidades são suportadas por verificações de regras de validade para a base de conhecimento (os invariantes, também descritos neste relatório).

1.2 Formulação do problema

A obtenção de um circuito de entregas ótimo dada uma lista de entregas a realizar pode ser representado por um problema pesquisa em grafo, cujos nodos representam as moradas de entrega de encomendas e o centro de distribuição. Como damos o custo de entregar uma encomenda como 0, tanto em tempo como em distância e tempo, podemos simplificar o problema afirmando que se o estafeta passa por um nodo onde tem uma encomenda por realizar essa encomenda é dada como entregue. Assim, o problema passa por encontrar o caminho de menor custo possível que passe por todas as moradas das entregas de encomendas.

1.2.1 Estado inicial e final

Tanto o estado inicial como o estado final do problema encontram-se no nodo da Rua da Universidade, no centro de distribuição da *Green Distribution*. Podemos também afirmar que no estado inicial todas as encomendas ainda estão por entregar, e no estado final todas têm de estar entregues, no entanto ao nível de implementação do problema em código estamos apenas preocupados em encontrar um circuito ótimo que passe pelos locais de entrega.

1.2.2 Operadores

O principal operador deste problema consiste em viajar da rua A para a rua B, para quaisquer ruas A e B ligadas por uma aresta.

A pré-condição deste operador deverá obrigar a que o estado atual descreva uma rua A que está ligada por uma aresta à rua B.

O custo deste operador depende do indicador de produtividade de comparação escolhido. Cada aresta tem dois valores de peso: O seu peso em custo monetário e o seu peso em distância do trajeto. Através desta informação e da velocidade do veículo de transporte utilizado podemos calcular um custo que nos permite fazer pesquisas de acordo com o custo monetário da viagem, distância percorrida e tempo de viagem.

Capítulo 2

Implementação

O presente trabalho foi dividido em vários ficheiros por forma a obter uma melhor organização.

- **base_dados.pl**

  Contém os predicados de factos acerca da nossa base de conhecimento (informações de clientes, meios de transporte, etc).

- **funcoes_auxiliares.pl**

  Contém predicados gerais que são utilizados ao longo do projeto, auxiliando o desenvolvimento de predicados de uso mais específico.

- **identificacoes.pl**

  Contém predicados que permitem encontrar conhecimento a partir de identificadores associados (por exemplo, encontrar o predicado relativo a uma rua pelo seu nome).

- **procura.pl**

  Contém os predicados quem compõe as *queries* pedidas no enunciado da primeira fase do projeto.

- **trabalho.pl**

  O ficheiro principal, que inclui todos os outros ficheiros do projeto. É responsável também tanto pelos predicados relacionados com a adição e remoção de conhecimento, como por exemplo os invariantes.

- **writeStructs.pl**

  Contém predicados responsáveis por escrever, numa dada stream, o conhecimento atual do programa. Desta forma, é possível construir uma de base de conhecimento persistente guardando num ficheiro o seu estado atual.

```
:- include('base_dados.pl').
:- include('funcoes_auxiliares.pl').
:- include('writeStructs.pl').
:- include('queries.pl').
:- include('procura.pl').
```

2.1 Base de conhecimento

Para uma melhor representação do sistema de conhecimento para este companhia de entregas, começamos primeiramente por estabelecer o tipo de conhecimento proposto criando predicados que o representam.

Finalmente, concluímos que iríamos precisar de definir:

- **freguesia**

  Para representar todas as freguesias conhecidas pelo sistema, bem como informações relevantes às mesmas.

- **rua**

  Para associar as ruas por nós conhecidas às suas freguesias, bem como indicar a sua posição absoluta em coordenadas para estimar distâncias em pesquisas informadas.

- **aresta**

  Para representar associações existentes entre localizações por forma a podermos realizar pesquisas de circuitos em grafos.

- **transporte**

  Para representar todos os transportes que a *Green Distribution* possui, bem como as suas especificações.

- **estafeta**

  Para representar todos os trabalhadores da *Green Distribution* e os seus detalhes.

- **centroDistribuicao**

  Para indicar a morada de um centro de distribuição *Green Distribution*.

- **ranking**

  Para poder penalizar um estafeta caso este não cumprir o prazo determinado para a entrega de uma dada encomenda. Este *rank* representa a confiança atribuída a um estafeta, podendo este vir a diminuir com penalizações.

- **cliente**

  Para representar todos os clientes da *Green Distribution*, bem como as suas informações pessoais.

- **encomenda**

  Para representar todas as encomendas realizadas pelos clientes, quer estas já tenham ou não sido entregues.

- **serviço**

  Para representar um serviço, ou seja, toda a informação associada com o transporte e entrega de encomendas por um dado estafeta num dado circuito.

2.1.1 Freguesia

```
% freguesia: Nome -> {V,F}
```

A **freguesia** é identificada através de um **nome**. Tendo esta fase do trabalho o objetivo de estudar a pesquisa em grafos, foi removida a informação de custo e tempo de visita à freguesia presente na primeira fase do trabalho, tendo este de ser calculado a partir da localização inicial do trajeto a percorrer.

```
freguesia('Sao Vitor').
freguesia('Nogueiró').
freguesia('Merelim').
freguesia('Frossos').
freguesia('Caldelas').
freguesia('Briteiros').
freguesia('Priscos').
freguesia('Adaúfe').
freguesia('Maximinos').
freguesia('Esporões').
freguesia('Lamas').
freguesia('Braga').
```

2.1.2 Rua

```
% rua: Nome, Nome da freguesia,Coordenadas -> {V,F}
```

A **rua** é caracterizada por um **nome**, por uma dada **freguesia** à qual pertence e pelas suas coordenadas.

```
rua('Rua do Taxa','Sao Vitor',coordenada(1,4)).
rua('Avenida Dom João II','Nogueiró',coordenada(1,1)).
rua('Rua de São Bento','Merelim',coordenada(3,3)).
rua('Rua Doutor José Alves Correia Da Silva'
    ,'Frossos',coordenada(4,0)).
rua('Rua do Moinho','Caldelas',coordenada(3,5)).
rua('Rua Dr. Lindoso','Briteiros',coordenada(6,4)).
rua('Rua do Coucão','Priscos',coordenada(5,1)).
rua('Rua da Mota','Adaúfe',coordenada(7,2)).
rua('Rua Joãozinho Azeredo','Maximinos',coordenada(6,0)).
rua('Rua de Santa Marta','Esporões',coordenada(7,5)).
rua('Rua do Sol','Lamas',coordenada(3,1)).
rua('Rua da Universidade', 'Braga',coordenada(0,2)).
```

2.1.3 Morada

```
% morada: Nome da rua, Nome da freguesia -> {V,F}

morada(R,F):- rua(R,F), freguesia(F,_,_).
```

A **morada** foi construída de forma a posteriormente associarmos uma **morada** a um **cliente**.

2.1.4 Aresta

A **aresta** contém informação relativa a duas moradas, o custo do trajeto direto e a distância real que as une.

```
%aresta: Freguesia , Freguesia, Custo, Distancia Real -> {V,F}
aresta(morada('Rua da Universidade', 'Braga'),
    morada('Rua do Taxa','Sao Vitor'),2,11).
aresta(morada('Rua da Universidade', 'Braga'),
    morada('Avenida Dom João II','Nogueiró'),10,12).
aresta(morada('Rua do Taxa','Sao Vitor'),
    morada('Rua do Moinho','Caldelas'),1,13).
aresta(morada('Rua do Taxa','Sao Vitor'),
    morada('Rua de São Bento','Merelim'),3,14).
aresta(morada('Avenida Dom João II','Nogueiró'),
    morada('Rua do Sol','Lamas'),3,13).
aresta(morada('Avenida Dom João II','Nogueiró'),
    morada('Rua Doutor José Alves Correia Da Silva',
    'Frossos'),3,10).
aresta(morada('Rua do Moinho','Caldelas'),
    morada('Rua Dr. Lindoso','Briteiros'),1,13).
aresta(morada('Rua de São Bento','Merelim'),
    morada('Rua Dr. Lindoso','Briteiros'),2,15).
aresta(morada('Rua de São Bento','Merelim'),
    morada('Rua do Coucão','Priscos'),2,12).
aresta(morada('Rua do Sol','Lamas'),
    morada('Rua do Coucão','Priscos'),5,13).
aresta(morada('Rua Doutor José Alves Correia Da Silva',
    'Frossos'),morada('Rua do Coucão','Priscos'),12,13).
aresta(morada('Rua Doutor José Alves Correia Da Silva',
    'Frossos'),morada('Rua Joãozinho Azeredo','Maximinos'),4,13).
aresta(morada('Rua Doutor José Alves Correia Da Silva',
    'Frossos'),morada('Rua da Mota','Adaúfe'),3,15).
aresta(morada('Rua Dr. Lindoso','Briteiros'),
    morada('Rua da Mota','Adaúfe'),5,12).
aresta(morada('Rua Dr. Lindoso','Briteiros'),
    morada('Rua de Santa Marta','Esporões'),5,10).
```

2.1.5 Centro de Distribuição

```
% centroDistribuicao: Morada -> {V,F}
```

O predicado do centro de distribuição necessita apenas da morada do mesmo.

```
% centroDistribuicao: Morada -> {V,F}
centroDistribuicao(morada('Rua da Universidade', 'Braga')).
```

Com a informação das moradas e dos caminhos (arestas) que as unem, podemos então construir o seguinte grafo para ajudar a visualizar o mapa das localizações da nossa base de conhecimento:

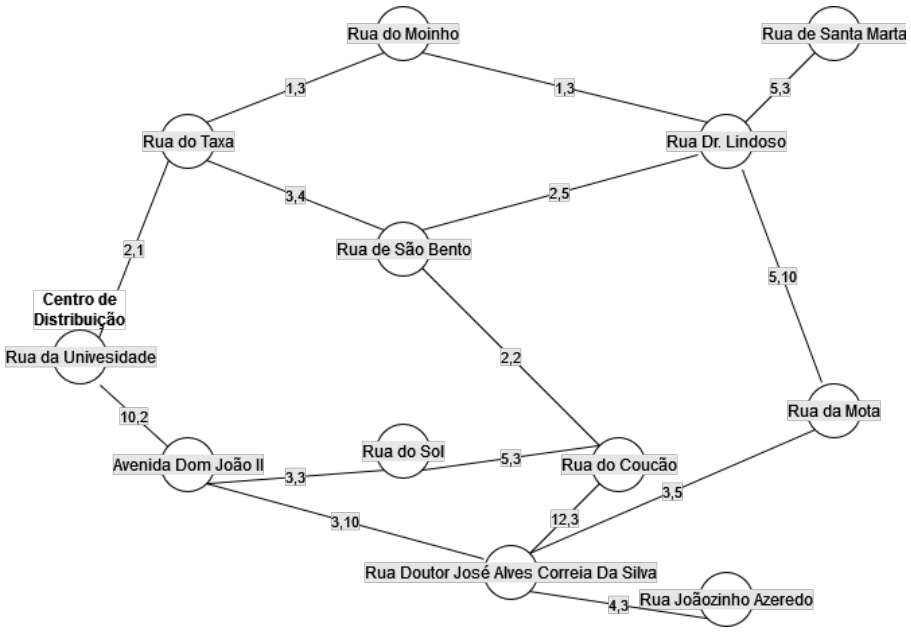

Figura 2.1: Grafo das ruas da base de conhecimento

O primeiro valor do peso da aresta é o custo monetário, enquanto que o segundo é o distância.

2.1.6 Transporte

```
% transporte: Id,Nome,Velocidade,Carga,
% Pontos_Ecologicos, MediaConsumo -> {V,F}
```

Cada **transporte** é identificado por um **ID** único, tendo também campos de **nome**, **velocidade média**, **carga máxima**, **pontos ecológicos** e média de custo de consumo.

```
transporte(1,'bicicleta',10,5,5,1).
transporte(2,'mota',35,20,-2,6).
transporte(3,'carro',25,100,-5,8).
transporte(4,'bicicleta elétrica',40,15,3, 2).
transporte(5,'bicicleta elétrica',40,15,3, 2).
transporte(6,'bicicleta elétrica',40,15,3, 2).
transporte(7,'carro',25,100,-5,8).
transporte(8,'carro',25,100,-5,8).
transporte(9,'carro',25,100,-5,8).
transporte(10,'carro',25,100,-5,8).
transporte(11,'mota',35,20,-2,4).
transporte(12,'mota',35,20,-2,4).
transporte(13,'mota',35,20,-2,4).
transporte(14,'mota',35,20,-2,4).
transporte(15,'bicicleta',10,5,5,1).
transporte(16,'bicicleta',10,5,5,1).
transporte(17,'bicicleta',10,5,5,1).
```

2.1.7 Estafeta

```
% estafeta: ID, Nome -> {V,F}
```

O **estafeta** é identificado por um **ID** único e tem ainda de um campo de **nome**.

```
estafeta(1,'Bernardo').
estafeta(2,'Sofia').
estafeta(3,'Costa').
estafeta(4,'Filipe').
estafeta(4,'Joana').
estafeta(5,'Filipa').
estafeta(6,'Pedro').
estafeta(7,'João').
estafeta(8,'Ricardo').
estafeta(8,'Catarina').
estafeta(9,'Mafalda').
estafeta(10,'Martim').
```

2.1.8 Ranking

```
% ranking: ID Estafeta, Classificação -> {V,F}
```

A um **ranking** é associado um dado ID de **estafeta** e a sua respetiva **classificação**.

```
ranking(1,5).
ranking(2,4.3).
ranking(3,3.2).
ranking(4,1.3).
ranking(5,2.6).
ranking(6,4.2).
ranking(7,4.6).
ranking(8,3.5).
ranking(9,4.9).
ranking(10,4.4).
```

2.1.9 Cliente

```
% cliente: ID, Nome, Morada -> {V,F}
```

O **cliente** é identificado por um **ID** único, e tem também associado a si um **nome** e uma **morada**.

```
cliente(1,'Alexandra Epifânio',morada('Rua do Taxa','Sao Vitor')).
cliente(2,'Filipa Simão',morada('Rua do Coucão','Priscos')).
cliente(3,'Ana Reigada',morada('Rua de Santa Marta','Esporões')).
cliente(4,'Maria Dinis',morada('Rua Dr. Lindoso','Briteiros')).
cliente(5,'Mafalda Bravo',morada('Rua Dr. Lindoso','Briteiros')).
cliente(6,'Alexandre Rosas',morada('Rua da Mota','Adaúfe')).
```

2.1.10 Encomenda

```
% encomeda: ID, Id_Cliente, Peso, Volume, DiaPedido: D/M/Y/H/M,
%           Limite: D/H, -> {V,F}
```

A **encomenda** é identificada por um **ID** único, um ID de **cliente**, um **peso**, um **volume**, o **dia** em que o pedido foi efetuado e um **limite** de entrega.

```
encomenda(1, 1 ,7,  2,  10/11/2021/8/35, 0/2 ).
encomenda(2, 1 ,5,  10, 22/11/2021/23/45, 4/0 ).
encomenda(3, 3 ,2,  2,  23/11/2021/16/02, 1/0 ).
encomenda(4, 5 ,5,  2,  24/11/2021/9/03,  0/2 ).
encomenda(5, 6 ,10, 10, 03/12/2021/11/30, 4/0 ).
encomenda(6, 7 ,25,  2,  03/12/2021/17/0, 1/0 ).
encomenda(7, 3 ,20,  2,  03/12/2021/18/23, 2/0 ).
encomenda(8, 4 ,17,  2,  04/12/2021/17/0, 0/10 ).
encomenda(9, 2 ,80,  2,  10/12/2021/17/0, 0/8 ).
encomenda(10, 7 ,3,  2,  11/12/2021/17/0, 0/9 ).
```

2.1.11 Serviço

```
% servico: ID, Id_estafeta, Ids_encomendas, Id_transporte,
%          DiaEntrega: D/M/Y/H/M, Classificacao, Caminho,
%           CustoReal -> {V,F}
```

O **serviço** é identificado por um **ID** único, ID de um **estafeta**, uma lista de IDs de **encomendas**, ID de **transporte**, **dia** em que a encomenda foi entregue, uma **classificação**, o caminho percorrido pelo estafeta e o custo de percorrer esse caminho.

```
servico(1,2,[4],11,24/11/2021/20/43,3,[
    morada('Rua da Universidade','Braga'),
    morada('Rua do Taxa','Sao Vitor'),
    morada('Rua do Moinho','Caldelas'),
    morada('Rua Dr. Lindoso','Briteiros'),
    morada('Rua do Moinho','Caldelas'),
    morada('Rua do Taxa','Sao Vitor'),
    morada('Rua da Universidade','Braga')],25).
servico(2,3,[5,6,7],7,4/12/2021/9/0,5,
    [morada('Rua da Universidade','Braga'),
    morada('Rua do Taxa','Sao Vitor'),
    morada('Rua de São Bento','Merelim'),
    morada('Rua do Coucão','Priscos'),
    morada('Rua de São Bento','Merelim'),
    morada('Rua Dr. Lindoso','Briteiros'),
    morada('Rua da Mota','Adaúfe'),
    morada('Rua Dr. Lindoso','Briteiros'),
    morada('Rua de Santa Marta','Esporões'),
    morada('Rua Dr. Lindoso','Briteiros'),
    morada('Rua do Moinho','Caldelas'),
    morada('Rua do Taxa','Sao Vitor'),
    morada('Rua da Universidade','Braga')],60).
```

```
servico(3,1,[2,3],15,24/11/2021/10/30,4,
    [morada('Rua da Universidade','Braga'),
    morada('Rua do Taxa','Sao Vitor'),
    morada('Rua do Moinho','Caldelas'),
    morada('Rua Dr. Lindoso','Briteiros'),
    morada('Rua de Santa Marta','Esporões'),
    morada('Rua Dr. Lindoso','Briteiros'),
    morada('Rua do Moinho','Caldelas'),
    morada('Rua do Taxa','Sao Vitor'),
    morada('Rua da Universidade','Braga')],25).
```

2.2 Queries adicionais

Antes de demonstrarmos as estratégias de procura que utilizamos e como as implementamos em *Prolog*, é importante explicarmos as *queries* que desenvolvemos não só para auxiliar o desenvolvimento dos algoritmos de pesquisa como para fornecer mais funcionalidades ao programa. Estas *queries* estão definidas em predicados no ficheiro **queries.pl**, usando predicados auxiliares desse e de outros ficheiros.

2.2.1 Velocidade média em entrega

```
velocidadeMediaEntrega('bicicleta',Carga,R) :-
    velocidadeMediaByTransporteName('bicicleta',V),
    R is V - 0.7*Carga.

velocidadeMediaEntrega('bicicleta eletrica',Carga,R) :-
    velocidadeMediaByTransporteName('bicicleta eletrica',V),
    R is V - 0.7*Carga.

velocidadeMediaEntrega('carro',Carga,R) :-
    velocidadeMediaByTransporteName('carro',V),
    R is V - 0.1*Carga.

velocidadeMediaEntrega('mota',Carga,R) :-
    velocidadeMediaByTransporteName('mota',V),
    R is V - 0.5*Carga.
```

O predicado **velocidadeMediaEntrega** permite, dado um dos veículos válidos para o qual está programado, obter a velocidade média do mesmo tendo em conta a sua carga atual.

2.2.2 Custo do consumo total em distância

```
custoConsumoEntrega(IdVeiculo,Dist,R) :-
    mediaConsumoByIdTransporte(IdVeiculo,Media),
    Custo = 1.45, %% preco dos combustiveis
    R is (Dist * Media * Custo) / 100.
```

O predicado **custoConsumoEntrega** obtem o custo total do consumo de energia para uma dada distância.

2.2.3 Peso total em em função do caminho percorrido

```
calculaPesoTotalEmFuncaoDoCaminho( [], _, 0 ).

calculaPesoTotalEmFuncaoDoCaminho([ Morada/_ | T]
                                  , Caminho, R ) :-
    member(Morada,Caminho),
    calculaPesoTotalEmFuncaoDoCaminho(T,Caminho, R).

% Encomenda ainda nao foi entregue
calculaPesoTotalEmFuncaoDoCaminho([Morada/Peso | T]
                                  , Caminho, R ):-
    \+ member(Morada, Caminho),
    calculaPesoTotalEmFuncaoDoCaminho(T,Caminho, R1),
    R is Peso + R1.
```

Dada uma lista de pares **Morada/Peso** em que **Peso** representa o peso de uma encomenda a ser entregue numa dada **Morada**, e um **Caminho** já percorrido por um estafeta, este predicado permite obter o peso total das encomendas que ainda não foram entregues no fim do **Caminho**.

2.3 Algortimos de pesquisa

2.3.1 Pesquisa não informada

Neste projeto implementamos três algoritmos de pesquisa não informada:

- ***DFS*** (*Depth First Search*)
- ***BFS*** (*Breadth First Search*)
- ***IDS*** (*Iterative Deepening Search*)

Para utilizar estes algoritmos de pesquisa, utilizamos o predicado **searchNaoInformadaCaminhoIdaVolta**, ou apenas **searchNaoInformadaCaminho** caso o caminho de volta ao centro de distribuição não seja relevante. Estes são implementados da seguinte forma:

```
%searchNaoInformadaCaminhoIdaVolta(AlgoritmoDeProcura
%   , IdsDeEncomendasAEntregar, Caminho).
searchNaoInformadaCaminho(Procura, Enc, Caminho):-
    maplist(moradaByIdEncomenda, Enc, Encomendas),
    ((Procura == 'bfs', bfs_complex(Encomendas,Caminho));
    (Procura == 'dfs', dfs_complex(Encomendas,Caminho));
    (Procura == 'iterativa' , readNumber(X)
     , buscaIterativa_complex(Encomendas,X,Caminho))).

searchNaoInformadaCaminhoIdaVolta(Procura, Enc, Caminho):-
    maplist(moradaByIdEncomenda, Enc, Encomendas),
    ((Procura == 'bfs', bfs_complexIdaVolta(Encomendas,Caminho));
    (Procura == 'dfs', dfs_complexIdaVolta(Encomendas,Caminho));
    (Procura == 'iterativa' , readNumber(X)
     , buscaIterativa_complexIdaVolta(Encomendas,X,Caminho))).
```

Depth First Search

De forma a melhor explicar o predicado principal da pesquisa de ida e volta primeiro em profundidade, este pode ser dividido nos seguintes passos.

- **1-** É obtido o ponto de partida, ou seja, a morada do centro de distribuição
- **2-** É encontrado, pesquisando primeiro em profundidade, o caminho que passa por todas as moradas de entrega de encomendas
- **3-** É obtida a última morada a visitar no caminho obtido
- **4-** É encontrado, pesquisando primeiro em profundidade, um caminho que chegue ao centro de distribuição a partir da morada obtida no passo 3
- **5-** O caminho obtido no passo 4 é acrescentado ao caminho obtido no passo 2

```
dfs_complexIdaVolta(Dest, Caminho):-
    centroDistribuicao(X),          %(passo 1)
    dfsAux(X, Dest , [X], Cam),     %(passo 2)
    reverse(Cam,[Nodo|_]),          %(passo 3)
    dfs(Nodo,X,[ Nodo | CamVolta ]),%(passo 4)
    append(Cam,CamVolta,Caminho).   %(passo 5)

dfs_complex(Dest, S):-
    centroDistribuicao(X),          %(passo 1)
    dfsAux(X, Dest , [X], S).       %(passo 2)
```

Se o caminho de volta não for relevante, o predicado realizará apenas os primeiros 2 passos. Foram criados três de predicados de pesquisa. O **dfs** e **dfs2** (auxiliar ao **dfs**) encontram pesquisando primeiro em profundidade um caminho entre dois nodos dados. O **dfsAux** encontra pesquisando primeiro em profundidade um caminho que percorra todos os nodos fornecidos, começando de um dado nodo.

```
dfs(X, Y, S):-  dfs2(X, Y, [X], S).

dfs2(X, X, Cam, S):- reverse(Cam,S).

dfs2(X, Y, Cam, S):-
  adjacente(Novo,X,_,_),
  \+ member(Novo, Cam),
  dfs2(Novo, Y, [Novo|Cam], S).


dfsAux(_, Dest, Cam, S):-
    iguais(Dest,Cam,R),
    sort(R,R1),
    sort(Dest,Dest1),
    R1 = Dest1,
    reverse(Cam,S).

dfsAux(X, Y, Cam, S):-
  adjacente(Novo,X,_,_),
  \+ member(Novo, Cam),
  dfsAux(Novo, Y, [Novo|Cam], S).
```

De um modo geral, a nossa implementação do algoritmo ***DFS*** começa por encontrar um nodo adjacente que ainda não tenha sido percorrido (evitar ciclos). Posteriormente, ou encontramos

o caminho desejado se o novo nodo for o nodo de destino (ou se tivermos percorrido todos os nodos destino, no caso da **dfsAux**), ou o algoritmo repete, encontrando um outro nodo adjacente ao novo nodo encontrado previamente (daí a pesquisa ser primeiro em profundidade).

Breadth First Search

Este algoritmo foi implementado de uma forma praticamente idêntica ao de pesquisa primeiro em profundidade, sendo diferente apenas a ordem pela qual os nodos são visitados. Tal como a pesquisa primeiro em profundidade, podemos dividir o algoritmo principal de controlo da pesquisa em vários passos para o melhor demonstrar.

- **1-** É obtido o ponto de partida, ou seja, a morada do centro de distribuição
- **2-** É encontrado, pesquisando primeiro em largura, o caminho que passa por todas as moradas de entrega de encomendas
- **3-** É obtida a última morada a visitar no caminho encontrado
- **4-** É encontrado, pesquisando primeiro em largura, um caminho que chegue ao centro de distribuição a partir da morada obtida no passo 3
- **5-** O caminho obtido no passo 4 é acrescentado ao caminho obtido no passo 2

```
bfs_complexIdaVolta(Dest, Caminho):-
    centroDistribuicao(X),          %(passo 1)
    bfsAux(Dest,[[X]],Cam),         %(passo 2)
    reverse(Cam,[Nodo|_]),          %(passo 3)
    bfs(Nodo,X,[ Nodo | CamVolta ]),%(passo 4)
    append(Cam,CamVolta,Caminho).   %(passo 5)

bfs_complex(Dest, Cam):-
    centroDistribuicao(X),          %(passo 1)
    bfsAux(Dest,[[X]],Cam).         %(passo 2)
```

Se o caminho de volta não for relevante, o predicado realizará apenas os primeiros 2 passos. Foram criados três de predicados de pesquisa. O **bfs** e **bfs2** (auxiliar ao **bfs**) encontram pesquisando primeiro em largura um caminho entre dois nodos dados. O **bfsAux** encontra pesquisando primeiro em largura um caminho que percorra todos os nodos fornecidos, começando de um dado nodo.

```
bfs(Orig, Dest, Cam):- bfs2(Dest,[[Orig]],Cam).

bfs2(X,[[X|T]|_],Solucao)  :- reverse([X|T],Solucao).
bfs2(EstadoF,[EstadoA|Outros],Solucao) :-
    EstadoA = [Act|_],
    findall([X|EstadoA],
            (EstadoF\==Act,
             adjacente(Act,X,_,_),
             \+ member(X,EstadoA)),
            Novos),
    append(Outros,Novos,Todos),
    bfs2(EstadoF,Todos,Solucao).


bfsAux( [],[[X|T]|_],Solucao)  :-
    reverse([X|T],Solucao).
```

```
bfsAux( EstadoF,[[X|T]|Outros],Solucao)  :-
    member(X,EstadoF),
    delete(EstadoF,X,R),
    bfsAux(R,[[X|T]|Outros],Solucao).
bfsAux(EstadoF,[EstadoA|Outros],Solucao) :-
    EstadoA = [Act|_],
    findall([X|EstadoA],
            ( \+ member(Act,EstadoF),
             adjacente(Act,X,_,_),
             \+ member(X,EstadoA)),
            Novos),
    append(Outros,Novos,Todos),
    bfsAux(EstadoF,Todos,Solucao).
```

Esta implementação do algoritmo ***BFS*** mantém uma fila de caminhos a estudar que de forma a analisar primeiro todos os nodos que estão ao menor nível de profundidade possível acessíveis. Até ser encontrado um estado que atinja o nodo de destino (ou que tenha visitado todos os nodos destino no caso do predicado **bfsAux**), serão acrescentados à fila de caminhos a estudar todos os caminhos resultantes da visita a um nodo adjacente ao nodo atual do caminho a ser estudado (mantendo assim a ordem primeiro em largura).

Iterative Deepening Depth First Search

Ao utilizador este algoritmo de pesquisa, é pedido ao utilizador um limite para o número de iterações a realizar. As iterações do algoritmo são controladas através dos predicados **buscaIterativa_Control** e **buscaIterativa_complexControl**, sendo estes utilizados sempre que é realizada uma pesquisa neste algoritmo.

```
buscaIterativa_Control(_,_,Ls,L,[]):- Ls =:= (L+1) , !, fail.
buscaIterativa_Control(X,Dest,N,_,S):-
    buscaIterativaSimplesAux(X, Dest, [X], 1 , N, S).
buscaIterativa_Control(X,Dest,N,L,S):-
    Ns is N + 1,
    buscaIterativa_Control(X,Dest,Ns,L,S).

buscaIterativa_complexControl(_,_,Ls,L,[]):- Ls =:= (L+1) , !, fail.
buscaIterativa_complexControl(X,Dest,N,_,S):-
    buscaIterativaAux(X, Dest, [X], 1 , N, S).
buscaIterativa_complexControl(X,Dest,N,L,S):-
    Ns is N + 1,
    buscaIterativa_complexControl(X,Dest,Ns,L,S).
```

Mais uma vez, iremos dividir o predicado de controlo principal da pesquisa iterativa em passos para o melhor explicar.

- **1-** É obtido o ponto de partida, ou seja, o centro de distribuição
- **2-** É encontrado, pesquisando primeiro em profundidade com limite de profundidade, um caminho que passa por todas as moradas de entregas de encomendas
- **3-** É obtido o nodo da última morada a visitar na viagem de ida
- **4-** É encontrado, pesquisando primeiro em profundidade com limite de profundidade, um caminho que chegue ao centro de distribuição a partir do nodo obtido no passo 3
- **5-** O caminho obtido no passo 4 é acrescentado ao caminho obtido no passo 2

```
buscaIterativa_complex(Dest, L, S):-
    centroDistribuicao(X),                          %(passo 1)
    buscaIterativa_complexControl(X,Dest,1,L,S).    %(passo 2)

buscaIterativa_complexIdaVolta(Dest, L, Cam):-
    centroDistribuicao(X),                          %(passo 1)
    buscaIterativa_complex(Dest,L,S),               %(passo 2)
    len(S,N), N \==0,
    reverse(S,[Nodo|_]),                            %(passo 3)
    LNovo is L - N,
    buscaIterativa(Nodo, X,  LNovo, [_|CamVolta]),  %(passo 4)
    len(CamVolta,N1), N1\==0,
    append(S,CamVolta,Cam).                         %(passo 5)
buscaIterativa_complexIdaVolta(_, _, []).
```

Durante todo este processo, mantemos em conta o limite de profundidade da iteração. Se um caminho válido não for encontrado no limite de profundidade da iteração, o limite é aumentado e a procura começa novamente. Desenvolvemos dois predicados para implementar o algoritmo de pesquisa em si. O predicado **buscaIterativaSimplesAux** encontra o caminho de um nodo inicial a um nodo final, utilizando a pesquisa primeiro em profundidade com limite de profundidade. O predicado **buscaIterativaAux** encontra o caminho a partir de um nodo incial que passa por todos os nodos de uma dada lista de nodos, utilizando a pesquisa primeiro em profundidade com limite de profundidade.

```
buscaIterativaSimplesAux(_, _, _, Ls , L , []):- Ls =:= (L+1) , !,fail.
    buscaIterativaSimplesAux(X, X, Cam, _ , _ , S):- reverse(Cam,S).
    buscaIterativaSimplesAux(X, Y, Cam, N , L ,S):-
      adjacente(Novo,X,_,_),
      \+ member(Novo, Cam),
      N_ is N + 1,
      buscaIterativaSimplesAux(Novo, Y, [Novo|Cam], N_ ,L, S).


buscaIterativaAux(_, _, _, Ls , L , []):- Ls =:= (L+1) , !, fail.
buscaIterativaAux(_, Dest, Cam, _ , _ , S):-
    iguais(Dest,Cam,R),
    sort(Dest, DestSorted),
    sort(R, RSorted),
    RSorted = DestSorted,
    reverse(Cam,S).
buscaIterativaAux(X, Dest, Cam, N , L ,S):-
    adjacente(Novo,X,_,_),
    \+ member(Novo, Cam),
    N_ is N + 1,
    buscaIterativaAux(Novo, Dest, [Novo|Cam], N_ ,L, S).
```

Este algoritmo opera de uma forma praticamente idêntica ao algoritmo depth first, verificando apenas entre iterações se o limite de profundidade não foi ultrapassado para continuar a pesquisa.

2.3.2 Pesquisa informada

Neste projeto implementámos dois algoritmos de pesquisa informada:

- ***A****
- ***Gulosa***

Para utilizar estes algoritmos de pesquisa, utilizamos o predicado **searchInformadaCaminhoIdaVolta**, ou apenas **searchInformadaCaminho** caso o caminho de volta ao centro de distribuição não seja relevante. Estes são implementados da seguinte forma:

```
%%% PROCURA INFORMADA
%% Procura -> tipo de procura ("gulosa" ou "aestrela")
%% Funcao -> Custo poderá ser calculado através do "tempo", do "custo"
↪  (custo do gasoleo) ou da "distancia"
%%           Indicador de produtividade a ser usado
%% Enc -> lista de ids de encomendas
%% Transporte -> Id do transporte
%% Caminho -> resultado
%% Custo -> custo obtido relativamente à função usada.
searchInformadaCaminho(Procura, Funcao, Enc, Transporte,Caminho,Custo):-
    maplist(moradaAndPesoByIdEncomenda, Enc, Encomendas),
    resolve_procura_complex(Procura,Funcao, Encomendas
    ↪  ,Transporte,Caminho/Custo).

searchInformadaCaminhoIdaVolta(Procura, Funcao, Enc,
↪  Transporte,Caminho,Custo):-
    maplist(moradaAndPesoByIdEncomenda, Enc, Encomendas),
    resolve_procura_complex_idaVolta(Procura,Funcao, Encomendas
    ↪  ,Transporte,Caminho/Custo).
```

Previamente à resolução da procura em si, todos os ids de encomendas dados são convertidos para *tuples* (Morada na qual a encomenda será entregue e o seu peso).

De forma a melhor explicar o predicado principal destas pesquisas é de salientar:

- O que distingue estes dois algoritmos é o facto de que o **A*** não só tem atenção ao estima calcula ao longo de toda a pesquisa, ao contrário da **Gulosa**, como ao custo até lá calculado.
- **O modo como o estima é calculado**

  O estima será a distância, linha reta, de um dado nó até ao centro de distribuição, calculado através das suas coordenadas associadas.

```
%%% DISTANCIA com coordenadas
distancia(coordenada(X1,Y1), coordenada(X2,Y2), R):-
    Sum is ((X1 - X2)**2) + ((Y1 - Y2)**2),
    R is sqrt(Sum).
distancia(M1,M2,R):-
    coordenadaByMorada(M1,C1),
    coordenadaByMorada(M2,C2),
    distancia(C1,C2,R).

estima( A ,R):-
     coordenadaByMorada(A,M1),
     centroDistribuicao(X),
     coordenadaByMorada(X,M2),
     distancia(M1,M2,R).
```

- **A forma como o custo é calculado**

  Este encontra-se dependente do indicador de produtividade a ser utilizado.

  De modo a facilitar a implementação dos dois algoritmos, os predicados que encontram-se responsáveis por calcular o custo terão que ter *dynamic* 5 obrigatoriamente, sendo que serão por esta ordem respetivamente:

```
%% nome_predicado(IdTransporte, PesoTotalDasEncomendas,
↪  DistanciaDaAresta,CustoDaAresta, R_CustoObtido)
```

  Neste trabalho foram implementados três indicadores de produtividade:

  - **tempo de entrega**

```
custoTempo(Id,Peso,Dist,_,EsteCusto):-
    transporteById(Id,
    ↪  transporte(_,Veiculo,_,_,_,_)),
    velocidadeMediaEntrega(Veiculo,Peso,V),
    EsteCusto is Dist / V
.
```

  - **distância de entrega**

```
custoDistancia(_,_,Dist,_,Dist).
```

  - **custo de entrega**

```
custoConsumo(Id,_,Dist,CustoA,Custo):-
    custoConsumoEntrega(Id,Dist,CustoR),
    Custo is CustoA + CustoR.
```

  Com isto, passaremos ao algoritmo em si.

```
resolve_procura_complex_aux(_, _, [] , _, _,
↪  Caminho/Custo,[Goal|Caminho]/Custo):- goal(Goal).
resolve_procura_complex_aux(Procura, Nome, Encomendas , Final, Id,
↪  CamAux/Cus ,Caminho/Custo):-
    % Ir buscar o estima menor e fazer esse o nodo Final
    % ate percorrer todas as moradas das encomendas
    nodoMenorEstima(Encomendas, Final, Nodo/Estima ),
    ((Nome == tempo, Funcao = custoTempo) ;
     (Nome == distancia, Funcao = custoDistancia) ;
     (Nome == custo, Funcao = custoConsumo) ),
    procura(Procura,Funcao, Id, Encomendas, Final, [[Nodo]/0/Estima],
    ↪  [Final|InuCam]/CustoP/_),
    removeEncomendaLista(Encomendas, [Nodo|InuCam], EncAtualizadas),
    Cus2 is Cus + CustoP,
    append(CamAux,InuCam,Cam),
    resolve_procura_complex_aux(Procura, Nome, EncAtualizadas , Nodo, Id,
    ↪  Cam/Cus2 , Caminho/Custo)
.
```

Visto pretendermos ser capazes entregar múltiplas encomendas, inicialmente iremos determinar qual a encomenda que terá menor estima em relação ao centro de distribuição. Com isso, vamos realizar a procura do centro até essa mesma encomenda, obtendo um dado caminho.

Visto querermos garantir, não só a entrega de uma encomenda, mas sim a de várias, iremos verificar se no caminho obtido mais alguma das outras encomendas pertencente à nossa lista foi entregue, removendo-a.

Até todas as encomendas serem entregues, iremos repetir o mesmo processo, tendo em atenção que nas próximas iterações, a próxima encomenda a ser entregue será escolhida, tendo em atenção ao menor estima em relação, não ao centro de distribuição, mas ao último nó do caminho obtido anteriormente. O caminho resultante será a junção de todos os caminhos obtidos ao longo de todo o processo.

```
procura(Procura,_,_,_,Nodo,Caminhos,Caminho):-
    obtem_caminho(Procura,Caminhos,Caminho),
    Caminho=[Nodo|_]/_/_.
% Resolve procura
% Procura -> é o nome do tipo de procura que queremos, pode ser gulosa ou a
↪  aestrela
% Funcao -> Nome do indice de produtividade a ser usado
% Id -> Id do transporte a ser usado
% Encomendas -> Encomendas a serem entregues
% Final -> Nodo inicial do caminho
% Caminhos -> auxiliar para conseguir obter a solucao
% SCaminho -> Solucao
procura(Procura, Funcao,Id,Encomendas,Final,Caminhos,SCaminho):-
    obtem_caminho(Procura,Caminhos, MelhorCaminho),
    seleciona(MelhorCaminho, Caminhos, OutrosCam),
    expande(Funcao, Id, Encomendas,Final,MelhorCaminho, ExpCam),
    append(OutrosCam, ExpCam, NCam),
    %% remover encomendas q ja foram entregues
    removeEncomendaLista(Encomendas, NCam, EncAtualizadas),
    procura(Procura,Funcao, Id, EncAtualizadas, Final, NCam, SCaminho).
```

A procura de um caminho de um dado dado nó até outro é obtido através de vários passos:

- Obtenção de o melhor caminho até então já encontrado.

```
obtem_caminho(_,[Caminho], Caminho) :- !.
obtem_caminho('aestrela',[ Caminho1/Custo1/Estima1,
↪  _/Custo2/Estima2|Caminhos], MCam) :-
        Custo1+Estima1 =< Custo2+Estima2, !,

        ↪  obtem_caminho('aestrela',[Caminho1/Custo1/Estima1|Caminhos],
        ↪  MCam).
obtem_caminho('gulosa',[ Caminho1/Custo1/Estima1,
↪  _/_/Estima2|Caminhos], MCam) :-
        Estima1 =< Estima2, !,

        ↪  obtem_caminho('gulosa',[Caminho1/Custo1/Estima1|Caminhos],
        ↪  MCam).
obtem_caminho(A,[_|Caminhos], MCam) :-
↪  obtem_caminho(A,Caminhos, MCam).
```

  Como se pode constatar, a obtenção desse melhor caminho terá em conta o tipo de algoritmo a ser usado. No caso da **Gulosa** iremos ter atenção apenas o estima, já no caso da **A*** teremos atenção tanto ao estima como a custo.

- Remoção do melhor caminho obtido da nossa lista de caminhos

```
    seleciona(E, [E|Xs], Xs).
    seleciona(E, [X|Xs], [X|Ys]) :-
        seleciona(E, Xs, Ys).
```

- Obtenção de todos os caminhos adjacentes que conseguimos obter, através do melhor caminho encontrado.

```
%Dá todos os caminhos adjacentes ao NovoCaminho
expande(_,_,_,Nodo,[[Nodo|Caminho]/Custo/Est | T],
↪  [[Nodo|Caminho]/Custo/Est | T]).

expande(Funcao, Id, Encomendas,_,Caminho, ExpCam) :-
    findall(NovoCaminho, adjacenteAux(Funcao, Id, Encomendas,
    ↪  Caminho, NovoCaminho), ExpCam).

adjacenteAux(Funcao, Id, Encomendas, [Nodo|Caminho]/Custo/_,
↪  [ProxNodo, Nodo| Caminho]/NovoC/Est) :-
    adjacente(Nodo,ProxNodo,CustoA,Distancia),
    \+ member(ProxNodo, Caminho),
    calculaPesoTotalEmFuncaoDoCaminho(Encomendas, [Nodo| Caminho],
    ↪  Peso),
    call(Funcao,Id,Peso,Distancia,CustoA,EsteCusto),
    estima(Nodo,Est),
    NovoC is Custo+EsteCusto.
```

O custo será obtido através da chamada do nome do predicado inicialmente dado com os seus respetivos argumentos.

Este processo será repetido até ser obtido o caminho pretendido.

2.3.3 Resultados

Utilizando os predicados de teste **testNaoInformada** e **testInformada**, medimos diversas estatísticas de execução através das quais podemos comparar e avaliar os nossos algoritmos de pesquisa. Entre estas estatísticas, é importante realçar que o uso de memória foi medido comparando o uso de memória do programa antes da execução da pesquisa ao uso de memória no final da pesquisa, podendo a gestão de memória automática do Prolog afetar estes resultados. Para tentar garantir resultados fiáveis, o programa foi reiniciado entre execuções de pesquisas. Os testes foram realizados todos com as mesmas 3 encomendas, IDs 6, 7 e 8 e o mesmo ID de transporte (11).

Pesquisa não informada

Estratégia	Tempo (ms)	Espaço (KB)	Dist/Custo/ Tempo	Encontrou a melhor solução?	Observações
DFS	4.318	23.7	157/49.1/19.4	Nao	
BFS	9.836	26.7	122/43.1/15.1	Não	Como as arestas do nosso grafo não têm todas o mesmo peso, esta estratégia não é suficiente para encontrar o caminho ótimo.
IDS	1.637	5.1	122/43.1/15.1	Não	O custo de memória e tempo de execução deste algoritmo de procura são relativamente menores.

Pesquisa informada

- **Tempo**

Estratégia	Tempo (ms)	Espaço (KB)	Tempo	Encontrou a melhor solução?	Observações
Gulosa	2.403	9.1	17.7	Nao	
A*	2.04	9.1	14.5	Sim	Por não sobreestimar o o valor do tempo gasto entre arestas, esta pesquisa consegue encontrar o melhor caminho.

- **Distância**

Estratégia	Tempo (ms)	Espaço (KB)	Distância	Encontrou a melhor solução?	Observações
Gulosa	1.902	6,8	146	Nao	
A*	0.934	6,9	119	Sim	Por não sobreestimar o o valor da distância entre nodos, esta pesquisa encontra o melhor caminho.

- **Custo**

Estratégia	Tempo (ms)	Espaço (KB)	Custo	Encontrou a melhor solução?	Observações
Gulosa	2.427	8.3	36.5	Nao	
A*	1.9	8.3	32	Sim	Por não sobreestimar o o valor do custo monetário entre arestas, esta pesquisa consegue encontrar o melhor caminho.

Observações

Consideramos, dados os tempos de execução arbitrariamente pequenos, que a pesquisa ***A**** é a pesquisa ótima a usar neste contexto. Se o contexto de uso de aplicação justificasse um grafo muito mais complexo (entregas internacionais, por exemplo), poderíamos observar resultados menos objetivos em relação à escolha de um algoritmo de pesquisa ótimo.

2.3.4 Funcionalidades adicionais da primeira fase

Por forma a manter este relatório curto e relevante ao enunciado da segunda fase deste trabalho, omitimos informação relativa a funcionalidades extra que implementamos na primeira fase do projeto. Estas funcionalidades, entre as coisas a obtenção de dados a partir de IDs, adição de conhecimento utilizando invariantes para validação e escrita do conhecimento para ficheiros, permanecem na implementação da segunda fase do projeto.

As suas explicações poderão ser consultadas no relatório da primeira fase do projeto, sendo que quaisquer alterações realizadas foram por motivos de compatibilidade com novos tipos de conhecimento.

2.4 Interface Gráfica

Por forma a providenciar uma *User Experience* cómoda e de fácil utilização, foi criada uma interface em *Python*. Esta permite realizar diversas interações com a base de conhecimento, o que inclui a execução de queries, bem como remoção e adição de conhecimento. Todas as as queries pedidas, tanto na Fase 1 como na Fase 2, estão disponíveis, assim como outras que considerámos relevantes. Os resultados são apresentados no que consideramos ser um formato inteligível e agradável de utilizar.

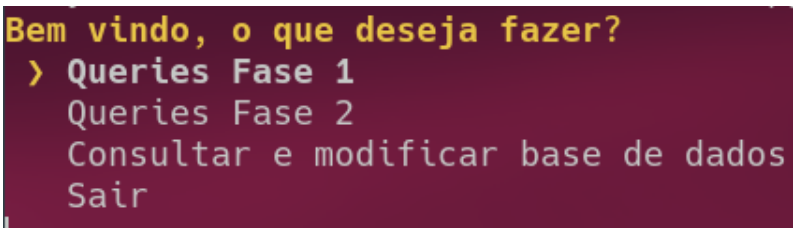

Figura 2.2: Menu Principal

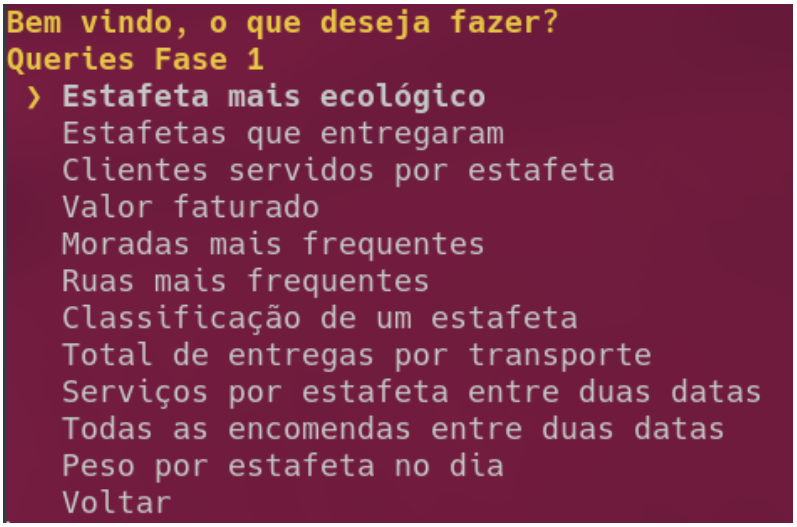

Figura 2.3: Menu da fase 1

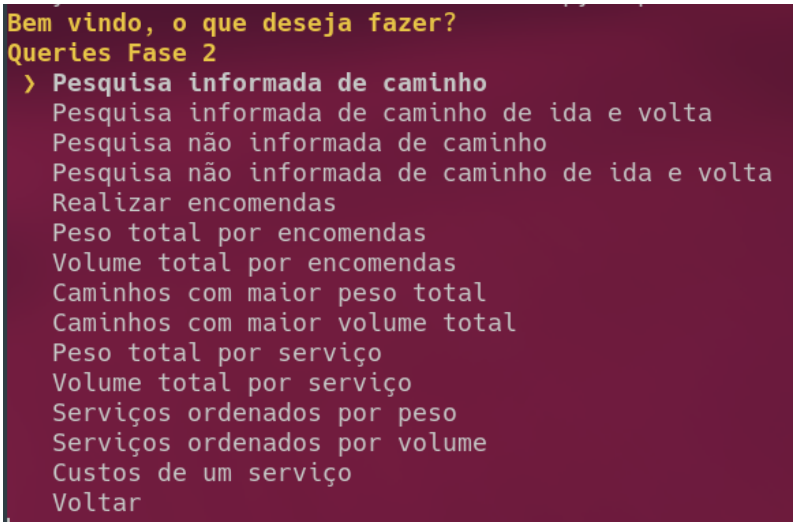

Figura 2.4: Menu da fase 2

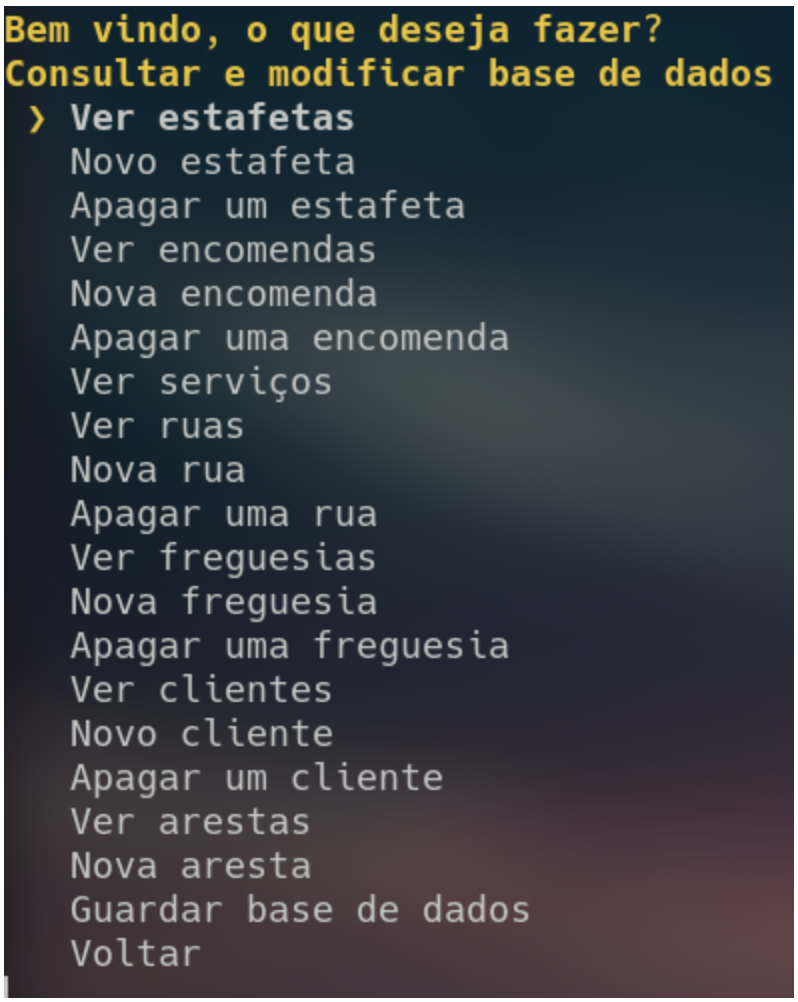

Figura 2.5: Menu para consultar e modificar a base de conhecimento

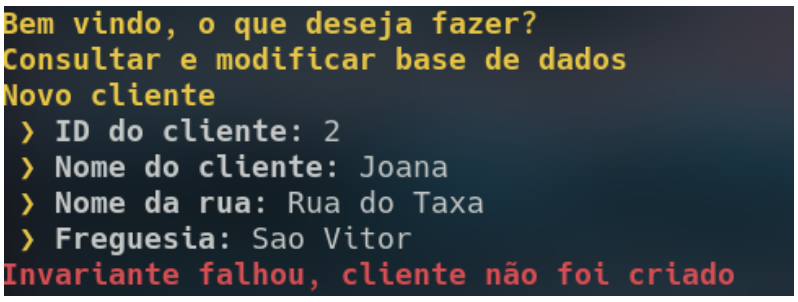

Figura 2.6: Adição de Conhecimento (Erro porque o id já existe)

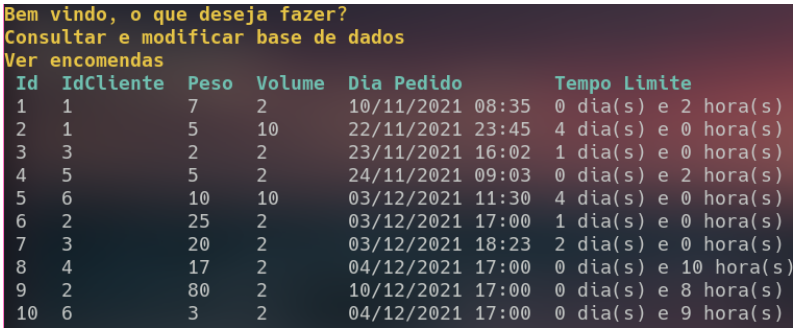

Figura 2.7: Consultar lista de encomendas

Figura 2.8: Consultar lista de estafetas

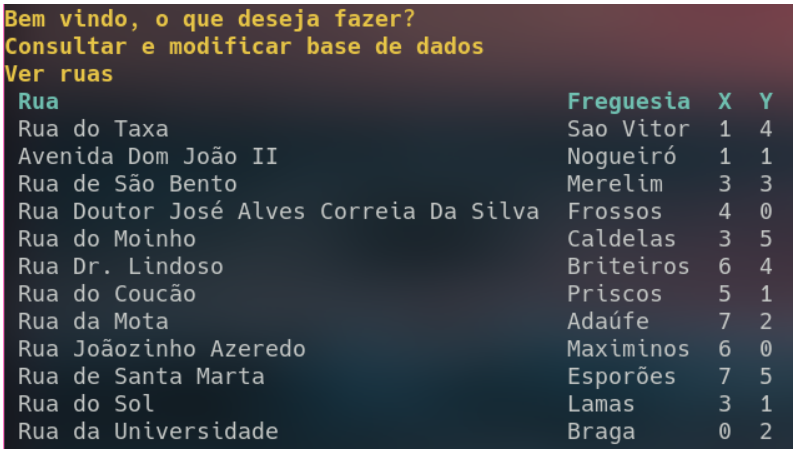

Figura 2.9: Listagem de ruas disponíveis na base de conhecimento

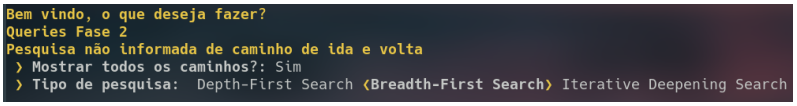

Figura 2.10: Escolha do algoritmo de pesquisa não informada a utilizar

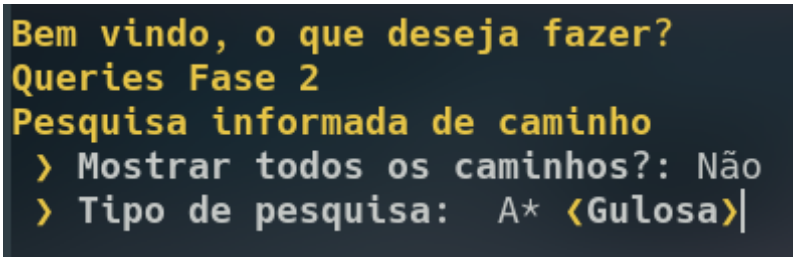

Figura 2.11: Escolha do algoritmo de pesquisa informada a utilizar

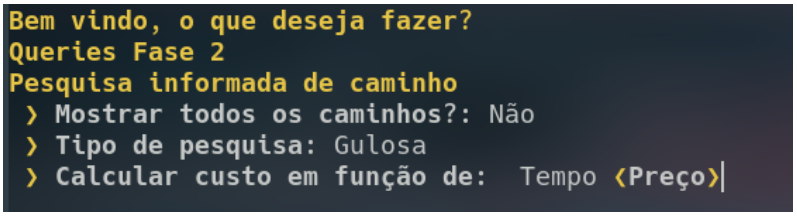

Figura 2.12: Escolha do fator de pesquisa a utilizar

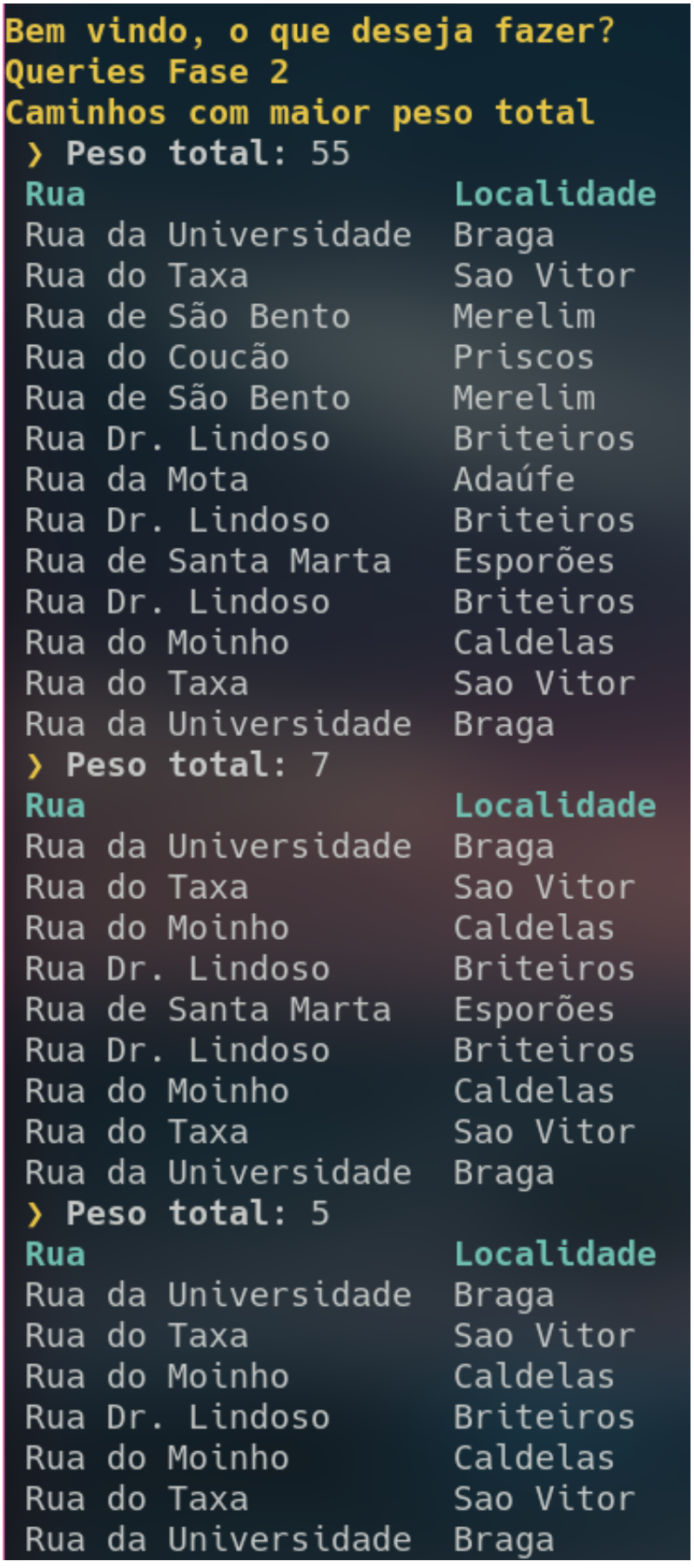

Figura 2.13: Listagem de caminhos com maior peso

Capítulo 3

Conclusão

Este projeto permitiu ao nosso grupo colocar em prática o conhecimento adquirido não só de programação lógica mas também sobre algoritmos de pesquisa. Ficamos globalmente satisfeitos com a nossa formulação do enunciado como um problema e a sua posterior resolução, bem como com a nossa implementação dos algoritmos lecionados nas aulas. As funcionalidades extra que adicionamos fornecem ao programa uma útil caixa de ferramentas que o aproxima de um contexto de uso mais realista.