Universidade do Minho

Licenciatura em Engenharia Informática

Relatório de Inteligência Artificial

Grupo 31

Unidade Curricular: Inteligência Artificial [J305N3]

Coordenador: Professor Doutor Paulo Jorge Freitas Oliveira Novais

Nome: Carlos Filipe Almeida Dias

Número: 93185

Contacto: a93185@alunos.uminho.pt

Curso: Licenciatura em Engenharia Informática, Universidade do Minho

Nome: José Pedro Martins Magalhães

Número: 93273

Contacto: a93273@alunos.uminho.pt

Curso: Licenciatura em Engenharia Informática, Universidade do Minho

Nome: Francisco Reis Izquierdo

Número: a93241

Contacto: a93241@alunos.uminho.pt

Curso: : Licenciatura em Engenharia Informática, Universidade do Minho

Nome: Duarte Augusto Rodrigues Lucas

Número: a89526

Contacto: a89526@alunos.uminho.pt

Curso: : Licenciatura em Engenharia Informática

Ano Letivo 2021/2022

Conteúdo

1 Introdução

No âmbito da disciplina de Inteligência Artificial, foi nos colocado o problema de desenvolver um sistema capaz de demonstrar as funcionalidades subjacentes à utilização da linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento e construção de mecanismos de raciocínio para a resolução de um dado problema. Este problema assenta na vertente ecológica imposta pela empresa Green Distribution, no qual a mesma tem como objetivo privilegiar sempre o meio de entrega mais ecológico.

Com isto e dado o objetivo de resolver o problema supramencionado, implementámos e construímos estruturas, desde a base até ao objetivo.

2 Desenvolvimento

2.1 Predicados Base:

- ***Cliente***:

Constituído por: Id, Nome.

```
cliente(Id, Nome).
```

- ***Encomenda***:

Constituído por: Id, Peso, Volume, Transporte, IdCliente, IdEstafeta, Preço, Prazo.

```
encomenda(Id, Peso, Volume, Transporte, IdCliente, IdEstafeta, Preco,
Prazo).
```

- ***Entregas***:

Constituído por: IdEntrega, Rua, Freguesia, Data, IdEncomenda, Classificação, Tempo, Distancia.

```
entrega(IdEntrega, Rua, Freguesia, Data, IdEncomenda, Classificacao, Tempo, Distancia).
```

- ***Estafeta***:

Constituído por: Id, Nome, Classificação, TotalEntregas.

```
estafeta(Id, Nome, Classificacao, TotalEntregas).
```

- ***Transporte***:

Constituído por: Transporte, Peso, Velocidade.

```
transporte(Transporte, Peso, Velocidade).
```

- ***Data***:

Constituído por: Dia, Mês, Ano.

```
data(Dia, Mes, Ano).
```

Assim conseguimos construir a nossa base de conhecimento.

2.2 Base de Conhecimento

Baseando-nos nos predicados de base em cima mencionados, produzimos a seguinte base de conhecimento:

- ***Cliente***:

```
cliente( 1, tropa).
cliente( 2, bino).
cliente( 3, chico).
cliente( 4, paulo).
cliente( 5, andre).
cliente( 6, maria).
cliente( 7, jose).
cliente( 8, ãntonio).
cliente( 9, ana).
cliente( 10, raimundo).
cliente( 11, francisco).
cliente( 12, paulo).
cliente( 13, manuel).
cliente( 14, lucas).
cliente( 15, luis).
cliente( 16, francisca).
cliente( 17, daniel).
cliente( 18, gabriel).
cliente( 19, bea).
cliente( 20, antonio).
```

- ***Encomenda***:

```
encomenda(1,4,10,bicicleta,1,1,100,1).
encomenda(2,5,12,bicicleta,2,2,60,1).
encomenda(3,3,14,bicicleta,3,2,24,3).
encomenda(4,2,9,bicicleta,4,3,45,4).
encomenda(5,1,11,bicicleta,5,3,60,5).
encomenda(6,4,16,bicicleta,2,1,37,2).
encomenda(7,10,20,mota,7,1,22,1).
encomenda(8,13,23,mota,3,2,49,6).
encomenda(9,16,26,mota,9,1,75,6).
encomenda(10,20,30,mota,10,5,23,4).
encomenda(11,11,22,mota,5,6,33,2).
encomenda(12,8,13,mota,12,3,88,5).
encomenda(13,55,40,carro,13,8,73,4).
encomenda(14,61,34,carro,14,7,90,1).
encomenda(15,42,40,carro,15,2,40,1).
encomenda(16,85,60,carro,15,1,34,6).
encomenda(17,73,55,carro,17,1,58,6).
encomenda(18,35,25,carro,18,4,71,4).
encomenda(19,47,40,carro,19,4,99,2).
encomenda(20,2,9,bicicleta,20,5,81,4).
```

- ***Entregas***:

```
entrega(1, "Rua Travessa do Falcão", "Canelas", data(18, 11, 2021), 1, 3, 2,20).
entrega(2, "Rua Gomes Ferreira", "Santo Tirso", data(7, 5, 2021), 2, 5, 1,10).
entrega(3, "Rua da Universidade", "Braga", data(18, 10, 2021), 3, 1, 5,5).
entrega(4, "Rua Caminho Velho dos Barreiros", "Madeira", data(7, 4, 2021), 4, 5, 1,15).
entrega(5, "Rua 25 Abril", "Agrela", data(10, 11, 2021), 5, 3, 2,12).
entrega(6, "Rua Comes Damião", "Lisboa", data(17, 3, 2021), 6, 2, 3,16).
entrega(7, "Rua do Velho", "Vila Nova de Famalicão", data(8, 1, 2021), 7, 2, 3,24).
entrega(8, "Rua Nova", "Guimaraes", data(8, 5, 2021), 8, 4, 6,5).
entrega(9, "Rua dos Caes", "Aves", data(28, 7, 2021), 9, 3, 3,7).
entrega(10, "Rua Agua Nova", "Agua Longa", data(17, 5, 2021), 10, 3, 1,12).
entrega(11, "Rua Escura", "Porto", data(18, 11, 2021), 11, 3, 2,2).
entrega(12, "Rua Gomes Ferreira", "Santo Tirso", data(3, 5, 2021), 12, 4, 2,1).
entrega(13, "Rua Gomes Ferreira", "Santo Tirso", data(23, 11, 2021), 13, 4, 3,16).
entrega(14, "Rua 25 Abril", "Agrela", data(6, 2, 2021), 14, 5, 1,17).
entrega(15, "Rua 25 Abril", "Agrela", data(9, 2, 2021), 15, 5, 1,27).
entrega(16, "Rua Comes Damião", "Lisboa", data(1, 1, 2021), 16, 2, 3,12).
entrega(17, "Rua Travessa do Falcão", "Canelas", data(9, 11, 2021), 17, 3, 3,21).
entrega(18, "Rua Comes Damião", "Lisboa", data(7, 12, 2021), 18, 1, 2,12).
entrega(19, "Rua Travessa do Falcão", "Canelas", data(1, 12, 2021), 19, 1, 2,20).
entrega(20, "Rua da Universidade", "Braga", data(26, 8, 2021), 20, 5, 1,14).
```

- ***Estafeta***:

```
estafeta( 1, duarte, 5.0, 20).
estafeta( 2, esquerdo, 0.0, 15).
estafeta( 3, pedro, 0.1, 10).
estafeta( 4, madeira, 0.2, 20).
estafeta( 5, ana, 3.4, 5).
estafeta( 6, maria, 4.1, 30).
estafeta( 7, antonio, 4.3, 27).
estafeta( 8, lucas, 4.7, 13).
estafeta( 9, margarida, 1.2, 16).
estafeta( 10, patricia, 1.5, 19).
estafeta( 11, filipa, 2.8, 32).
estafeta( 12, francisco, 3.8, 35).
estafeta( 13, vasco, 4.6, 17).
estafeta( 14, goncalo, 4.1, 10).
estafeta( 15, andre, 5.0, 20).
estafeta( 16, grimaldo, 3.7, 29).
estafeta( 17, rafa, 4.5, 19).
estafeta( 18, guedes, 1.3, 31).
estafeta( 19, felix, 1.2, 10).
estafeta( 20, silva, 4.7, 35).
```

2.3 Funções Auxiliares

- ***Predicado entrega_prazo***: IdEntrega

Esta função verifica se a entrega foi realizada no prazo estabelecido pelo cliente.

Dando o idEntrega verifica em que tempo a encomenda foi realizada.

```
% Extensão do predicado entrega_prazo:  IdEntrega -> {V,F}

entrega_prazo(IdEntrega) :-
    entrega(IdEntrega, _, _, _, IdEncomenda, _, Tempo, _),
    encomenda(IdEncomenda, _, _, _, _, _, _, Prazo),
    Prazo @>= Tempo.
```

- ***Predicado Penalização***: IdEntrega

Esta função é responsável por aplicar uma penalização ao estafeta que não fez uma certa encomenda no prazo estabelecido, ficando responsável por menos uma entrega.

```
% Extensão do predicado penalizacao:  IdEntrega -> {V,F}

penalizacao(IdEntrega) :- not(entrega_prazo(IdEntrega)), entrega(IdEntrega, _, _, _, X, _, _,_), encomenda(X,_,_,_,_,Y,_,_), estafeta(Y,_,_,_,G), estafeta(
Y,_,_,_,Novo), Novo is G-1.
```

Dado um Id de uma entrega, a função verifica se a entrega foi realizada no tempo estabelecido ou não, recorrendo à função auxiliar "entrega_prazo", caso esta tenha sido entregue então o estafeta não leva penalização, caso contrário a função vai ao estafeta responsável pela entrega e atribui-lhe uma penalização, ficando responsável por menos 1 entrega.

- ***Predicado Entre Datas:***

O predicado entre_datas, dado três datas, verifica se a última se encontra no intervalo das duas primeiras datas.

```
entre_datas(data(D0, M0, A0), data(D10, M10, A10), data(D1, M1, A1)) :-
    (maior_ou_igual(A1, A0), menor_ou_igual(A1, A10));
    (maior_ou_igual(A1, A0), menor_ou_igual(A1, A10), maior_ou_igual(M1, M0), menor_ou_igual(M1, M10));
    (maior_ou_igual(A1, A0), menor_ou_igual(A1, A10), maior_ou_igual(M1, M0),
     menor_ou_igual(M1, M10), maior_ou_igual(D1, D0), menor_ou_igual(D1, D10)).

maior_ou_igual(N1, N2) :- N1 @>= N2.
menor_ou_igual(N1, N2) :- N1 @=< N2.
```

- ***Predicado getAllClients:***

A Função getAllClients devolve uma lista com todos os clientes e todas as respetivas informações.

```
% Predicado que devolve a lista de todos os clientes
% Extensão do predicado: getAllClients: Lista -> {V, F}

getAllClients(Lista):- findall((Id, Nome), cliente(Id, Nome), Lista).
```

- ***Predicado getALLEncomendas:***

A Função getALLEncomendas devolve uma lista com todas as encomendas e as respetivas informações.

```
% Predicado que devolve a lista de todas as encomendas
% Extensão do predicado getAllEncomendas: Lista -> {V, F}

getAllEncomendas(Lista):- findall((Id, Peso, Volume, Transporte, IdCliente, IdEstafeta, Preco, Prazo), encomenda(Id, Peso, Volume, Transporte, IdCliente, IdEstafeta, Preco,
Prazo), Lista).
```

- ***Predicado getALLEntregas:***

O predicado getALLEntregas percorre todas as Entregas e devolve uma lista com todas as Entregas e as respectivas informações.

```
% Predicado que devolve a lista de todas as entregas
% Extensão do predicado getAllEntregas: Rua, Freguesia, Data, IdEncomenda, Classificacao, Tempo, Distancia -> {V, F}

getAllEntregas(Lista):- findall((IdEntrega, Rua, Freguesia, Data, IdEncomenda, Classificacao, Tempo, Distancia), entrega(IdEntrega, Rua, Freguesia, Data, IdEncomenda,
Classificacao, Tempo, Distancia), Lista).
```

- ***Predicado getALLEstafetas:***

O predicado getALLEstafetas devolve uma lista com todos os estafetas e as respectivas informações.

```
% Predicado que devolve a lista de todos os estafetas
% Extensão do predicado getAllEstafetas: Lista -> {V, F}

getAllEstafetas(Lista):- findall((Id, Nome, Classificacao, TotalEntregas), estafeta(Id, Nome, Classificacao, TotalEntregas), Lista).
```

2.4 Predicado de Evolução:

2.4.1 Predicado de evolução de Cliente:

```
% Predicado que insere um cliente
% Extensão do predicado new_cliente: Nome -> {V, F}

client_count(21).
new_cliente(Nome) :- client_count(C), asserta(cliente(C, Nome)), retract(client_count(C)), asserta(client_count(C + 1)).
```

A Função newcliente, dando um nome de um cliente, este insere um novo cliente na base de conhecimento com o respetivo nome e um id superior aos existentes.

Nota Evolução: Os predicados da nossa base de conhecimento não são editáveis, ou seja, mesmo que adicionemos um predicado com informações iguais a um predicado existente este terá um identificador diferente. Mantemos o controlo sob este identificador mantendo apenas um invariante que guarda a informação do próximo id a ser utilizado (id_Count), fazemos isto atualizando o id_Count sempre que criamos um novo predicado, ou seja, utilizamos o valor do id_Count como o novo id, apagamos o id_Count e criamos um novo id_Count com o valor incrementado do antigo.

2.4.2 Predicado de evolução de Encomenda

A Função new_encomenda, dando um volume, IdCliente, IdEstafeta, Preço e um Prazo, este cria uma nova encomenda com essas informações e um id superior aos existentes.

```
% Predicado que insere uma encomenda
% Extensão do predicado new_encomenda: Volume, Transporte, IdCliente, IdEstafeta, Preco, Prazo -> {V, F}

encomenda_count(21).
new_encomenda(Volume, Transporte, IdCliente, IdEstafeta, Preco, Prazo) :- encomenda_count(C),
    asserta(encomenda(C, Volume, Transporte, IdCliente, IdEstafeta, Preco, Prazo)), retract(encomenda_count(C)), asserta(encomenda_count(C + 1)).
```

2.4.3 Predicado de evolução de Entrega:

O predicado de evolução new_entrega, dando a rua, freguesia, data, IdEncomenda, Classificação, Tempo e uma Distância, cria uma entrega, com as respectivas informações e um id superior aos existentes.

```
% Predicado que insere uma entrega
% Extensão do predicado new_entrega: Rua, Freguesia, Data, IdEncomenda, Classificacao, Tempo, Distancia -> {V, F}

entrega_count(21).
new_entrega(Rua, Freguesia, Data, IdEncomenda, Classificacao, Tempo, Distancia) :- entrega_count(C),
    asserta(entrega(C, Rua, Freguesia, IdEncomenda, Classificacao)), retract(entrega_count(C)), asserta(entrega_count(C + 1)).
```

2.4.4 <u>Predicado de evolução de Estafeta:</u>

O predicado de evolução new_estafeta, dando um nome do estafeta, este cria um novo estafeta com um id superior aos existentes.

```
% Predicado que insere um estafeta
% Extensão do predicado new_estafeta: Nome -> {V, F}

estafeta_count(21).
new_estafeta(Nome) :- estafeta_count(C), asserta(estafeta(C, Nome, 0, 0)), retract(estafeta_count(C)), asserta(estafeta_count(C + 1)).
```

2.5 Predicado de Involução:

2.5.1 Predicado de involução de Cliente:

A Função Deletecliente, dando o Id do Cliente tenta fazer a remoção do cliente caso ele exista.

Nota Involução: Sempre que eliminamos um predicado fazemo-lo através do id do predicado que queremos eliminar, este só é eliminado se existir um predicado com esse id.

```
% Predicado que apaga um cliente
% Extensão do predicado delete_cliente: IdCliente -> {V, F}

delete_cliente(IdCliente) :- retract(cliente(IdCliente, _)).
```

2.5.2 Predicado de involução de Encomenda:

O predicado DeleteEncomenda, dando o IdEncomenda, tenta fazer a remoção da encomenda caso ela exista.

```
% Predicado que apaga uma encomenda
% Extensão do predicado delete_encomenda: IdEncomenda -> {V, F}

delete_encomenda(IdEncomenda) :- retract(encomenda(IdEncomenda, _, _, _, _, _, _)).
```

2.5.3 Predicado de involução de Entrega:

O predicado de involução DeleteEntrega, dado o IdEntrega tenta fazer a remoção da Entrega caso ela exista.

```
% Predicado que apaga uma entrega
% Extensão do predicado delete_entrega: IdEntrega -> {V, F}

delete_entrega(IdEntrega) :- retract(entrega(IdEntrega, _, _, _, _, _, _, _)).
```

2.5.4 <u>Predicado de involução de Estafeta:</u>

O predicado de involução DeleteEstafeta, dando o IdEstafeta tenta fazer a remoção do Estafeta caso ele exista.

```
% Predicado que apaga um estafeta
% Extensão do predicado delete_estafeta: IdEstafeta -> {V, F}

delete_estafeta(IdEstafeta) :- retract(estafeta(IdEstafeta, _, _, _)).
```

2.6 Funcionalidades

2.6.1 Identificar o estafeta que utilizou mais vezes um meio de transporte mais ecológico;

O predicado estafeta_ecologico devolve o id do estafeta mais ecológico.

Para identificarmos qual o estafeta mais ecológico desenvolvemos um método que atribui pontuações de poluição aos três transportes (carro: 2, mota: 1, bicicleta: 0) dando pontuação zero à bicicleta uma vez que esta não polui, pontuação essa que é depois multiplicada pela distância percorrida o que nos permite obter uma pontuação do nível de poluição de cada estafeta, por fim devolvemos o estafeta com a pontuação mais baixa.

```
% Pergunta 1:

estafeta_ecologico(IdEstafeta) :- findall((IdEstafeta, Distancia, Trans),
    (entrega(_, _, _, _, IdEncomenda, _, _, Distancia), encomenda(IdEncomenda, _, _, Trans, _, IdEstafeta, _, _)),
    Lista),
    findEco(Lista, [], ResLista),
    sumRep(ResLista, [], ResResLista),
    smallest(ResResLista, (0, 100000000), (IdEstafeta, Score)).

smallest([], (TId, TM), (TId, TM)).
smallest([(Id, M) | T], (TId, TM), R) :- M @< TM, smallest(T, (Id, M), R).
smallest([(Id, M) | T], (TId, TM), R) :- M @>= TM, smallest(T, (TId, TM), R).

findEco([], N, N).
findEco([(IdEstafeta, Distancia, Trans) | T], N, R) :- findEco(T, [(IdEstafeta, M) | N], R), trans(Trans, TT),
    M is Distancia * TT.

sumRep([], N, N).
sumRep([(IdEstafeta, M) | T], N, R) :- contains(N, (IdEstafeta, M)), add(N, (IdEstafeta, M), [], NewN), sumRep(T, NewN, R).
sumRep([(IdEstafeta, M) | T], N, R) :- sumRep(T, [(IdEstafeta, M) | N], R).

contains([(Id, M) | T], (Id, NewM)).
contains([(Id, M) | T], (NewId, NewM)) :- contains(T, (NewId, NewM)).

add([], (Id, M), N, N).
add([(IdEstafeta, LM) | T], (IdEstafeta, M), N, R) :- add(T, (IdEstafeta, M), [(IdEstafeta, G) | N], R), G is LM + M.
add([(LId, LM) | T], (IdEstafeta, M), N, R) :- add(T, (IdEstafeta, M), [(LId, LM) | N], R).

trans(carro, 2).
trans(mota, 1).
trans(bicicleta, 0).
```

2.6.2 Identificar que estafetas entregaram determinada(s) encomenda(s) a um determinado cliente;

Este predicado devolve a lista de estafetas que dada uma lista de ids de encomendas, verifica quais foram feitas pelo id do cliente dado.

```
% Predicado que devolve a lista dos ids dos estafetas que entregaram determinada(s) encomenda(s) a um determinado cliente
% Extensão do predicado getEstafetas: Lista1, Lista2, Resultado -> {V, F}


getEstafetas(IdCliente, [], []).
getEstafetas(IdCliente, [IdEncomenda|Tail1], [estafeta(IdEstafeta, Nome, Classificacao)|Tail2]):- encomenda(IdEncomenda, _, _, _, IdCliente, IdEstafeta, _,
_), estafeta(IdEstafeta, Nome, Classificacao), getEstafetas(IdCliente, Tail1, Tail2).
```

2.6.3 Identificar os clientes servidos por um determinado estafeta;

Este predicado devolve a lista de clientes servidos por um estafeta dado o seu id. Além disso, fazemos uso de um predicado auxiliar, elimina_repetidos que elimina da lista de clientes, clientes repetidos, isto é, o mesmo cliente ter sido servido mais que uma vez pelo estafeta em causa.

```
% Predicado que devolve a lista de clientes servidos por um dado estafeta
% Extensão do predicado clientes_servidos:  IdEstafeta, Resultado -> {V,F}

clientes_servidos(IdEstafeta, Clientes) :-
    findall(IdCliente, (entrega(_, _, _, _, X, _, _, _), encomenda(X, _, _, _, IdCliente, IdEstafeta, _, _)), C),
    elimina_repetidos(C, Clientes).


% Predicado que elimina elementos repetidos de uma lista
% Extensão do predicado elimina_repetidos: Lista, Resultado -> {V, F}

elimina_repetidos(L, R) :- elimina_repetidos_aux(L, [], R).
elimina_repetidos_aux([], L, L).
elimina_repetidos_aux([X | T], L, R) :- member(X, T), elimina_repetidos_aux(T, L, R).
elimina_repetidos_aux([X | T], L, R) :- elimina_repetidos_aux(T, [X|L], R).
```

2.6.4 Calcular o valor faturado pela Green Distribution num determinado dia;

Este predicado devolve o valor total faturado pela Green Distribution num dado dia. Além disso, fazemos uso do predicado auxiliar entregasDoDia, que devolve a lista de todas as entregas feitas no dado dia. Alado a este predicado auxiliar, usamos o predicado auxiliar somaPrecos que soma o valor de cada entrega feita, da lista de entregas supramencionada.

```
% Predicado que calcula o valor faturado pela Green Distribution num determinado dia
% Extensão do predicado entregasDoDia: Data, Resultado -> {V, F}

entregasDoDia(data(Dia, Mes, Ano), Total):- findall(IdEntrega, entrega(IdEntrega, _, _, data(Dia, Mes, Ano), _, _, _, _), Lista), somaPrecos(Lista, 0, Total).


% Predicado que dada uma lista de entregas entregas devolve o preço total das encomendas
% Extensão do predicado somaPrecos: Lista, Acumulador, Total -> {V, F}

somaPrecos([], Total, Total).
somaPrecos([IdEntrega|Tail], Acum, Total):- entrega(IdEntrega, _, _, _, IdEncomenda, _, _, _), encomenda(IdEncomenda, _, _, _, _, _, Preco, _), NovoAcum is
Acum + Preco, somaPrecos(Tail, NovoAcum, Total).
```

2.6.5 Identificar quais as zonas (e.g., rua ou freguesia) com maior volume de entregas por parte da Green Distribution;

Na resposta à query 5 decidimos abordar o problema de duas maneiras diferentes, para criàmos dois predicados que respondem à query de maneiras diferentes.

O predicado maior_volume_freg devolve uma lista de pares contendo o nome da freguesia e o número de entregas feitas nessa freguesia.

O predicado maior_volume_rua devolve uma lista de pares contendo o nome da rua e o número de entregas feitas nessa rua.

Todos os predicados supramencionados utilizam o predicado auxiliar more_freq que dada uma lista de entregas devolve uma lista de pares com o nome da rua/freguesia e o número de ocorrências.

```
% Predicado que devolve as zonas (Freguesias e Ruas) com maior volume de entregas
% Extensão do predicado maior_volume_order: Lista -> {V, F}

maior_volume_freg(L) :-
    findall(Freg, entrega(_, _, Freg, _, _, _, _, _), Lista),
    more_freq(Lista, [], L).


% Predicado que devolve as zonas (Ruas) com maior volume de entregas
% Extensão do predicado maior_volume_rua: Lista -> {V, F}

maior_volume_rua(L) :-
    findall(Rua, entrega(_, Rua, _, _, _, _, _, _), Lista),
    more_freq(Lista, [], L).


% Predicado que devolve uma lista de pares, no qual cada par é constituído por (Elemento, Quantidade)
% Extensão do predicado more_freq: Lista1, Lista2, Resultado -> {V, F}

more_freq([], L, L).
more_freq([H | T], L, R) :- contar(H, [H | T], N), delete(H, T, [], NT), more_freq(NT, [(H, N)|L], R).


% Predicado que conta a quantidade de vezes que um elemento aparece numa lista
% Extensão do predicado contar: Elemento, Lista, Quantidade -> {V, F}

contar(H, [], 0).
contar(H, [H | T], N) :- contar(H, T, G), N is G + 1.


% Predicado que remove todas aas ocorrências de um elemento de uma lista
% Extensão do predicado delete: Elemento, Lista, Resultado -> {V, F}
delete(H, [], N, N).
delete(H, [H | T], P, N) :- delete(H, T, P, N).
delete(H, [X | T], P, N) :- delete(H, T, [X | P], N).
```

2.6.6 Calcular a classificação média de satisfação de cliente para um determinado estafeta;

Este predicado devolve a média de satisfação, isto é, a média das classificações atribuídas pelos clientes a um estafeta dado o seu id. Para isto, o predicado faz uso do predicado auxiliar findall, começando por criar a lista de todas as classificações presentes nas entregas efetuadas pelo dado estafeta, e por fim somar todos os valores da lista e dividir este pelo comprimento da lista supramencionada.

```
% Predicado que devolve a classificação média de satisfação de cliente para um determinado estafeta
% Extensão do predicado classificacao_estafeta: IdEstafeta, Media -> {V, F}

classificacao_estafeta(IdEst,Med) :-  findall(Class, (entrega(_,_,_,_,X,Class, _, _),encomenda(_,_,_,_,_,IdEst,_, _)),T) , length(T,L) ,soma(T,S), Med is S/L .


% Predicado que devolve a soma dos elementos de uma lista
% Extensão do predicado soma: Lista, Total -> {V, F}

soma([],0).
soma([H|T],S):-soma(T,G),S is H+G.
```

2.6.7 Identificar o número total de entregas pelos diferentes meios de transporte, num determinado intervalo de tempo;

Este predicado, dado um intervalo de datas, devolve o número de entregas efetuadas usando os vários meios de transporte. Para isto, usa o predicado auxiliar entre_datas, criando uma lista para cada meio de transporte com todas as entregas efetuadas usando o mesmo. Por fim, devolve o comprimento de cada uma destas listas.

```
% Predicado que devolve o número total de entregas por cada meio de transporte num intervalo de tempo
% Extensão do predicado entregas_por_transporte_intervalo: DataInicio, DataFim, Resultado1, Resultado2, Resultado3 -> {V, F}

entregas_por_transporte_intervalo(Data0, Data10, Bicicleta, Mota, Carro) :-
    findall(IdEntregaBic, (entrega(IdEntregaBic, _, _, Data1, X, _, _, _),
        encomenda(X, _, _, bicicleta, _, _, _, _), entre_datas(Data0, Data10, Data1)), ListaBic),
    length(ListaBic, Bicicleta),

    findall(IdEntregaMot, (entrega(IdEntregaMot, _, _, Data2, X, _, _, _),
        encomenda(X, _, _, mota, _, _, _, _), entre_datas(Data0, Data10, Data2)), ListaMot),
    length(ListaMot, Mota),

    findall(IdEntregaCar, (entrega(IdEntregaCar, _, _, Data3, X, _, _, _),
        encomenda(X, _, _, carro, _, _, _, _), entre_datas(Data0, Data10, Data3)), ListaCar),
    length(ListaCar, Carro).
```

2.6.8 Identificar o número total de entregas pelos estafetas, num determinado intervalo de tempo;

Este predicado devolve o número total de entregas efetuadas por todos os estafetas, num intervalo de datas dado. Para isto, usa o predicado auxiliar todas_entregas, que dado um intervalo de datas devolve a lista com todas as entregas que se apresentam no intervalo de datas fornecido (fazendo também uso do já mencionado predicado auxiliar entre_datas). Por fim, devolve o comprimento da lista supramencionada.

```
% Predicado que calcula o número total de entregas pelos estafetas num determinado intervalo de tempo
% Extensão do predicado numero_total_entregas_intervalo: DataÍnicio, DataFim, Total -> {V, F}

numero_total_entregas_intervalo(DataInicio, DataFim, Total):- todas_entregas(DataInicio, DataFim, EntregasTotais), length(EntregasTotais, Total).


% Predicado que devolve todas as entregas feitas num intervalo de tempo
% Extensão do predicado todas_entregas: DataÍnicio, DataFim, Resultado -> {V, F}

todas_entregas(DataInicio, DataFim, Lista):- findall(X, (entrega(X, _,_,Data,_,_, _, _), entre_datas(DataInicio, DataFim, Data)), Lista).
```

2.6.9 Calcular o número de encomendas entregues e não entregues pela Green Distribution, num determinado período de tempo;

Este predicado devolve o número de encomendas entregues, mas também as não entregues, dado um intervalo de datas. Para o efeito, faz uso do predicado auxiliar findall que cria uma lista com todas as entregas e que são válidas no intervalo de datas fornecido, obtendo assim o número de encomendas entregues através do compirmento da mesma. Em contrapartida, compara todas as encomendas que não pertencem a esta lista (ou seja, encomendas não entregues) e verifica se estas pertencem ao intervalo de datas, criando para isto também uma lista e devolvendo o comprimento da mesma, obtendo assim o número de encomendas não entregues no intervalo de datas fornecido.

```
% Predicado que devolve o número de encomendas entregues (E) e não entregues (N) num determinado intervalo de tempo
% Extensão do predicado encomendas_tempo: DataÍnicio, DataFim, Total1, Total2 -> {V, F}

encomendas_tempo(Inicio,Fim, E, N) :-
    findall(IdEncomenda, (entrega(_,_,_,D,IdEncomenda,_, _, _),entre_datas(Inicio,Fim, D)),Lista),
    length(Lista, E),
    findall(IdEnc,(encomenda(IdEnc,_,_,_,_,_,_, _), not(member(IdEnc, Lista))), R),
    length(R, N).
```

2.6.10 Calcular o peso total transportado por estafeta num determinado dia;

Este predicado devolve o peso total transportado pelo estafeta, dado o seu id, num dado dia. Começa por criar uma lista com todos os pesos das entregas efetuadas pelo estafeta em causa no dado dia e por fim faz uso do predicado auxiliar sum que devolve o somatório de todos os elementos da lista supramencionada (cada elemento é o peso de uma entrega efetuada), obtendo assim o peso total transportado pelo estafeta, no dado dia.

```
% Predicado que calcula o peso total transportado por um estafeta num determinado dia
% Extensão do predicado peso_por_estafeta_em: Data, IdEstafeta, Total -> {V, F}

peso_por_estafeta_em(data(D, M, A), IdEstafeta, Total) :-
    findall(Peso, (entrega(_, _, _, _, X, _, _, _), encomenda(X, Peso, _, _, _, IdEstafeta, _, _)), Pesos),
    sum(Pesos, Total).

sum([], 0).
sum([H | T], N) :- sum(T, G), N is H + G.
```

3 Conclusão

O trabalho de Inteligência Artificial mostrou-se uma mais-valia não só como forma de aprender e exercitar o nosso conhecimento em relação à linguagem Prolog mas também como forma de aprender sobre a representação de conhecimento e raciocínio.

Importa também dizer que a realização do trabalho foi enriquecedora no que toca à consolidação de conceitos e conhecimentos aprendidos nesta disciplina e que serão uma ferramenta importante na nossa área de trabalho.