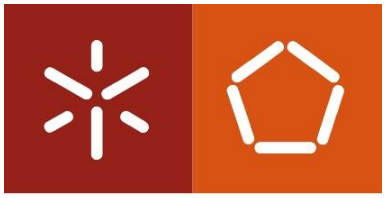
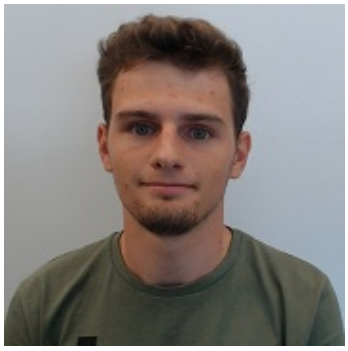


Universidade do Minho
Licenciatura em Engenharia Informática

Inteligência Artificial

Primeira fase - Instrumento de Avaliação
Programação em lógica

Grupo 50



Bohdan Malanka
a93300

Diogo Rebelo
a93278

Henrique Alvelos
a93316

Teresa Fortes
a93250

9 de fevereiro de 2024

Conteúdo

1 Introdução

No âmbito da Unidade Curricular de Inteligência Artificial, realizamos este trabalho prático de modo a aprofundar os conhecimentos relativamente a um sistema de representação de conhecimento e raciocínio, com capacidade para caracterizar um universo de discurso na área da logística de distribuição de encomendas.

Naturalmente, um dos objetivos é a utilização mais aprofundada da linguagem de programação em lógica, *Prolog*, também lecionada durante as aulas práticas. Neste contexto, perante o enunciado colocado, pretende-se inevitavelmente e transversalmente promover a resolução de problemas, através de mecanismos de raciocínio relevantes.

2 Resumo

A Ecologia é um tema relevante e que, cada vez mais, marca o seu lugar nos dias de hoje. Com efeito, o trabalho em questão prende-se com o desenvolvimento de uma Base de Conhecimento alusiva a um sistema de gestão de distribuição de encomendas em contexto sustentável. Assim sendo, uma associação, *Green Distribution*, visa transportar um conjunto de encomendas efetuadas por um ou mais clientes, numa certa região incentivando à utilização, preferencialmente, de meios de transporte mais sustentáveis.
Então, determinados estafetas, responsáveis pela distribuição da encomenda, são recompensados proporcionalmente à sua taxa ecológica, ou valor ecológico global. Então, é de extrema relevância a implementação das várias funcionalidades solicitadas no respetivo enunciado, de modo a ver concretizado com sucesso a boa gestão da empresa.

Igualmente importante, nesta secção, é referir que, ao longo do desenvolvimento de todo o trabalho (mais propriamente, a sua vertente mais prática: o código em si), foram surgindo algumas dúvidas em relação à melhor forma de representação da cada entidade e o modo de obtenção de cada resultado das queries. Contudo, socorremo-nos de várias funções já definidas pela própria linguagem para uma resolução bem conseguida e que nos pareceu mais eficaz.

Naturalmente, surge detalhada a explicação de cada modo de resolução das queries e a forma como se chegou ao resultado apresentado.

3 Código

3.1 Base de Conhecimento

Antes de começar a realizar as funcionalidades mínimas, tentamos perceber a forma como declarar conhecimento. Como referido anteriormente, a modo como a BC está organizada é importante não só em termos de compreensão do respetivo código, como também influencia a dificuldade de implementação de cada funcionalidade. Assim sendo, selecionou-se a forma de organização da informação que facilitaria a posterior implementação. Seguem-se, então, as diferentes estruturas:

3.1.1 Transporte: `transporte(tipo, valorEco)`

Transporte é o meio usado pelo estafeta para se deslocar até ao destino da encomenda.

- Só tem 3 opções: bicicleta, moto e carro;
- Para diferenciar o valor económico, adicionou-se outro elemento. A bicicleta, moto e carro têm valor 3,2,1, respetivamente.

3.1.2 Estafeta: `estafeta(idEstafeta, sumEcologico, sumAvaliacao,nEntregas)`

O Estafeta é o funcionário da empresa que entrega encomendas.

- Para podermos diferenciar de estafeta para estafeta, é necessário um parâmetro relacionado com o seu nome, neste caso *ID*;
- Para responder à *query* 1, foi necessário usar um parâmetro que funciona como acumulador, *sumEcologico*. Este valor acumula por cada vez que o estafeta usar algum tipo de transporte;
- De modo a responder à *query* 6 foi necessário usar outro parâmetro que também funciona como acumulador, e este corresponde à soma de todas as avaliações dos clientes aos quais entregou encomendas;
- Obviamente que, para fazer a média de valores, é necessário outro valor que corresponde ao número de entregas que o estafeta fez.

3.1.3 Cliente: `cliente(idCliente, freguesia,rua, nPedidosEntrega)`

O Cliente é uma pessoa que aderiu aos serviços da empresa.

- Para podermos diferenciar de cliente para cliente, é necessário um parâmetro relacionado com o seu nome, neste caso *ID*;
- Para responder à *query* 5, foi necessário usar dois parâmetros cujos nomes são muito explicativos: freguesia e rua. Além disso, foi necessário outro constituinte de modo a verificar quantas entregas aquele cliente fez.

3.1.4 Encomenda: `encomenda(id, peso, volume, precobase)`

A encomenda é um pacote que vai ser entregue ao cliente pelo estafeta.

- Para podermos diferenciar de encomenda para encomenda, é necessário um parâmetro relacionado com o seu nome, neste caso *ID*.
- De modo a resolver a *query* 10, o peso é necessário.

3.1.5 Cliente: `cliente(idCliente, freguesia,rua, nPedidosEntrega)`

O Cliente é uma pessoa que aderiu aos serviços da empresa.

- Para podermos diferenciar de cliente para cliente, é necessário um parâmetro relacionado com o seu nome, neste caso *ID*;
- Para responder à *query* 5, foi necessário usar dois parâmetros cujos nomes são muito explicativos: freguesia e rua. Além disso, foi necessário outro constituinte de modo a verificar quantas entregas aquele cliente fez.

3.1.6 Entrega: `entrega(idEstafeta, idCliente, idEncomenda, dataE(A,M,D), dataR(A,M,D), transporte, preco, avaliacao)`

A entrega é uma espécie de relatório sobre os detalhes de um envio de uma encomenda.

- Para saber quem entregou a encomenda, é necessário saber o *ID* do estafeta;
- O envio teve que ser para alguém, daí ter o *ID* do cliente;
- Para verificar as encomendas não entregues, é necessário um parâmetro para a data de Envio;
- De modo a saber que a encomenda foi entregue, há um parâmetro para a data de receção. Caso não tenha sido entregue, o valor é dataR(0,0,0);
- O estafeta teve de usar um meio de transporte para se deslocar, logo também existe um argumento para o transporte;
- Para saber o lucro (*query* 4) da empresa num determinado dia, adicionamos um parâmetro para o preço;
- O cliente precisa de avaliar o trabalho do estafeta, daí ter um espaço para tal.

3.2 Queries

3.2.1 Query 1: Identificar o estafeta que utilizou mais vezes um meio de transporte mais ecológico

Assumimos que o estafeta mais ecológico seria aquele cujo valor da divisão entre o somatório do valor seu ecológico com o seu total de entregas fosse o maior. Apesar de o objetivo ser a obtenção do estafeta com o maior valor ecológico, perante a existência de estafetas com igual valor, poderíamos tomar duas decisões: manter esta espécie de "duplicados" ou não. Primeiramente, desenvolvemos o primeiro predicado, mas apercebemo-nos que o segundo seria mais correto. Assim sendo, construíram-se dois predicados diferentes:

obter_estafeta_mais_eco_sem_dups(Final)
Perante dois estafetas com o mesmo valor ecológico (e o maior da lista), na ordenação decrescente, há remoção destes duplicados. Neste caso, obtém-se como estafeta mais ecológico a primeira ocorrência na lista (o que surge à cabeça).
obter_estafeta_mais_eco_com_dups(Final)
Perante dois estafetas com o mesmo valor ecológico (e o maior da lista), na ordenção decrescente, não há remoção destes duplicados e a este valor ecológico máximo surge associado o nome de todos os estafetas com aquele respetivo valor.

3.2.2 *Query* 2: Identificar que estafetas entregaram determinada(s) encomenda(s) a um determinado cliente

O nosso pensamento foi pesquisar quem entregou determinada encomenda ao cliente e adicionar um par (*ID* estafeta, *ID* encomenda) à medida que pesquisava na sua base de conhecimento. Caso a encomenda não estivesse associada ao cliente, o programa simplesmente ignora.

3.2.3 *Query* 3: Identificar os clientes servidos por um determinado estafeta

O nosso raciocínio foi pegar em todas as entregas que o estafeta fez e entregar uma lista de *ID's* de clientes não repetidos.

3.2.4 *Query* 4: Calcular o valor faturado pela Green Distribution num determinado dia

Visto que a entrega tem um elemento relacionado com o preço, o algoritmo foi pegar nesses elementos cujas entregas foram no dia selecionado e somar tudo.

3.2.5 *Query* 5: Identificar quais as zonas (e.g., rua ou freguesia) com maior volume de entregas por parte da Green Distribution

Neste caso, o objetivo é para cada entrega, formar uma lista de pares (Freguesia-Número de Encomendas), ordenando-a de acordo com o segundo elemento do par. No nosso caso, a freguesia é a Key e o outro valor é o Value.

Então, agrupa-se os pares pela respetiva chave (e.g. `[(gualtar-2,gualtar-3,tenoes-4,...]`) e, depois, somam-se valores de encomendas da mesma freguesia (e.g. `[(gualtar-5,tenoes-4,...]`). Tendo-se esta lista, trocam-se em todos os pares a Key com o Value, ordenando-os por default utilizando o keysort/2 (que não remove duplicados!). O algoritmo é semelhante para as ruas.

obter_zonas_mais_povoadas(HotSortedFinal, 1)
Obtém as ruas com mais entregas registadas. O primeiro argumento é a respetiva lista, o segundo serve apenas para indicar que a ordenação se dá por rua.
obter_zonas_mais_povoadas(HotSortedFinal, 2)
Obtém as freguesias com mais entregas registadas. O primeiro argumento é a respetiva lista, o segundo serve apenas para indicar que a ordenação se dá por freguesia.

3.2.6 *Query* 6: Calcular a classificação média de satisfação de cliente para um determinado estafeta

Dado que o estafeta tem dois parâmetros relacionados com a média de satisfação (*sumAvaliacao* e *nEntregas*) basta dividir o primeiro valor pelo segundo.

3.2.7 *Query* 7: Identificar o número total de entregas pelos diferentes meios de transporte, num determinado intervalo de tempo

Para esta *query*, filtramos as entregas que se encontram no intervalo de tempo dado e retiramos a lista de transportes. Com a ajuda da função auxiliar ocorrências2Tuplas transformamos essa lista por uma com tuplas com os transportes e as suas ocorrencias.

3.2.8 *Query* 8: Identificar o número total de entregas pelos estafetas, num determinado intervalo de tempo

Presumimos que o número total de entregas pelos estafetas eram as com sucesso, então selecionamos, para uma lista, as que a data de receção estivesse compreendida no intervalo de tempo e calculámos o tamanho da lista.

3.2.9 *Query* 9: Calcular o número de encomendas entregues e não entregues pela *Green Distribution*, num determinado período de tempo

A respetiva *query* obtém o conjunto de encomendas entregues (E) e não entregues (NE), num determinado período de tempo introduzido pelo utilizador. Estas informações numéricas são mostradas na forma de um par (E,NE), visível no último parâmetro da query.

obter_entregas_no_periodo(AnoI,MesI,DiaI,AnoF,MesF,DiaF,(E,NE))
Para obter cada informação numérica usam-se dois predicados auxiliares que têm estrutura semelhante. O que auxilia no cálculo das encomendas entregues só verifica se para cada encomenda o envio se encontra no período recebido, agrupando as encomendas numa lista. Depois é obtido o tamanho da lista, tendo-se o número total. O predicado que auxilia no cálculo das encomendas não entregues só tem de verificar se a data de envia da encomenda é posterior à do utilizador e se a data de receção pelo cliente tem o formato (0,0,0).

3.2.10 *Query* 10: Calcular o peso total transportado por estafeta num determinado dia

Para obter o peso total transportado por um estafeta queríamos uma regra que recebe como parâmetros o ID desse estafeta, a data do dia e a resposta na variável Total. Assim, fazemos um *findall* das entregas desse dia recolhendo o ID das encomendas numa lista. Depois com a ajuda da função auxiliar **pesoTotalAux** transformamos essa lista por peso das respetivas encomendas e no final somamos os pesos todos.

4 Conclusão

Em relação à concretização dos objetivos propostos, o grupo verifica que estes se encontram bem estabelecidos e concretizados. Na construção e desenvolvimento do trabalho houve solidez e organização, permitindo posterior concretização. Em suma, consideramos que o trabalho foi bem conseguido, não só porque realizamos todas as *queries* solicitadas, como também adicionamos mais algumas funcionalidades, por exemplo,a atualizar parâmetros aos estafetas, clientes e transporte. Contudo sabemos que, no conhecimento positivo, quando fazemos expansão, ou seja, inserimos novos predicados, não atualizamos o resto dos predicados, porque eles estão relacionados entre si.