Universidade do Minho

Sistemas de Representação de Conhecimento e Raciocínio

Departamento de Informática

Trabalho Individual

João Nuno Abreu

Número:
A84802

5 de Junho de 2020

Conteúdo

1. Resumo

O presente relatório é constituído por uma breve descrição e demonstração do trabalho realizado no âmbito do Exercício individual da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio.

Assim sendo, ao longo deste documento será apresentada toda a base do conhecimento e predicados que a constituem que permitem garantir uma correta evolução da base de conhecimento.

2. Introdução

Este exercício teve como principal objectivo motivar-nos para a utilização da linguagem de programação em lógica PROLOG, no âmbito da representação de conhecimento e construção de mecanismos de raciocínio para a resolução de problemas.

Durante o exercício foi desenvolvido um sistema para caracterizar um sistema de transportes do concelho de Oeiras, pelo que nos foi fornecido um repositório de dados abertos, contendo informações relativas às paragens de autocarro de Oeiras, englobando várias características, tais como a sua localização, as carreiras que as utilizam e respetivas operadoras, entre outras.

Este sistema tinha como requisitos a implementação de diversas funcionalidades, apresentadas ao longo deste relatório, que permitem uma correta manipulação da base de conhecimento e a aplicação de uma série de queries sobre ela.

3. Implementação

Para uma melhor exposição do trabalho realizado, dividi o mesmo em **5** diferentes partes. A primeira parte e segunda parte são a **Base de Conhecimento sobre paragens** e a **Base de Conhecimento sobre adjacências**, onde descrevi os predicados que constituem a base de conhecimento do sistema de representação de conhecimento e raciocínio.

Em seguida, apresenta-se o **Parser** que controla a inserção de conhecimento, lendo, analisando e guardando os dados a partir dos ficheiros fornecidos.

Guardei todas os predicados auxiliares (e.g. calcular inverso de uma lista) num ficheiro à parte chamado **Auxiliares**, e por fim as **Queries** encontram-se todas definidas no ficheiro chamado main.

3.1 Pesquisas e seus tipos

A pesquisa é um processo de encontrar uma sequência de etapas necessárias para resolver qualquer problema. A diferença anterior entre pesquisa informada e desinformada é que a pesquisa informada fornece a orientação sobre onde e como encontrar a solução. Por outro lado, a pesquisa desinformada não fornece informações adicionais sobre o problema, excepto sua especificação. No entanto, entre técnicas de pesquisa informadas e não informadas, a pesquisa informada é mais eficiente e económica.

Base para comparação	Pesquisa Informada	Pesquisa desinformada
Basic	Usa conhecimento para encontrar as etapas para a solução.	Nenhum uso de conhecimento
Eficiência	Altamente eficiente, consome menos tempo e custo.	A eficiência é mediadora
Custo	Baixo	Comparativamente alto
atuação	Encontra a solução mais rapidamente	A velocidade é mais lenta que a pesquisa informada
Algoritmos	Pesquisa em profundidade, primeira pesquisa e primeira pesquisa de menor custo	Profundidade heurística em primeiro lugar e pesquisa em largura e pesquisa A *

3.1.1 Depth-First

O algoritmo de largura foi o que utilizei em grande maioria do trabalho. Um dos exemplos do uso deste algoritmo pode-se encontrar em baixo:

```
solve_df_1(Node, Ending, Solution) :- depthfirst1([], Node, Ending,Solution,[]).
```

```
depthfirst1(Path, Node, Node,(D,L),Lista) :- inverso([Node|Path],D),inverso(Lista,L).
depthfirst1(Path, Node, Ending,Sol,Lista) :-
Node \= Ending,
move(Node, Node1, _,Carreira),
\+ member(Node1, Path),
depthfirst1([Node|Path], Node1, Ending,Sol,[Carreira|Lista]).
```

3.1.2 Breadth-First

O algoritmo de profundidade foi utilizado em comparação com a query 1, cujo código está apresentado em baixo:

```
query1_bf(Node,Ending,L) :- findall(Solution,solve_bf_1(Node, Ending, Solution),L).

solve_bf_1(Start,Ending, Res) :- breadthfirst([[Start]],Ending, Res).
solve_df_1(_,_,([],[])).

breadthfirst([[Node|Path]|_],Node, D) :- inverso([Node|Path],D).
breadthfirst([[Node|Path]|Paths],Ending, Solution) :-
Node \= Ending,
extend([Node|Path], NewPaths),
append(Paths, NewPaths, Paths1),
breadthfirst(Paths1, Ending,Solution).

extend([Node|Path], NewPaths) :-
    findall([NewNode, Node|Path],
    (move(Node, NewNode,_,_),
    \+ member(NewNode,[Node|Path])),
    NewPaths).
extend(Path,[]).
```

3.1.3 Gulosa

O algoritmo da Gulosa foi utilizado na query 6. A heurística foi calculada com a distância em linha reta entre o nodo atual com o nodo destino, como se pode ver de seguida. O resto do algoritmo pode ser visto em Anexos.

```
estima(Nodo,Destino,Estima) :- distance(Nodo,Destino,Estima).

distance(N1,N2,D) :- lat(N1,Lat1),lon(N1,Long1),
                lat(N2,Lat2),lon(N2,Long2),
                D is sqrt((Long2-Long1)*(Long2-Long1) + (Lat2-Lat1)*Lat2-Lat1).

lat(N,Lat) :- paragem(N,Lat,_,_,_,_,_,_,_,_,_).
lon(N,Lon) :- paragem(N,_,Lon,_,_,_,_,_,_,_,_).
```

3.2 Queries

No trabalho que desenvolvi, maioria das queries foram desenvolvidas com o algoritmo de procura em profundidade. Para efeitos de comparação, a query 1 foi feita também com o algoritmo não informado de procura em largura e a query 6 com o algoritmo de procura informado gulosa.

4. Análise de Resultados

Podemos ver aqui as comparções de queries 1 e 6.

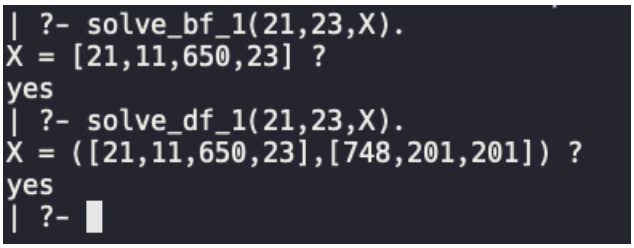

Figura 4.1: Query 1: depth e breadth

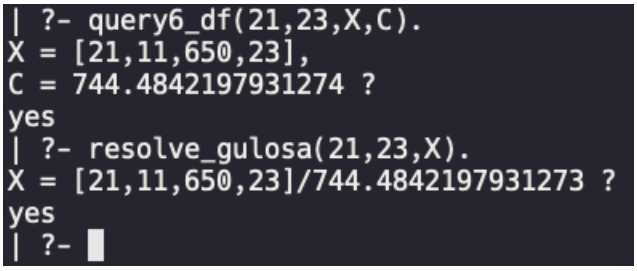

Figura 4.2: Query 6: depth e gulosa

5. Conclusões e Sugestões

No final deste exercício, foi desenvolvido com sucesso um sistema de transportes do concelho de Oeiras.

A boa preparação e o estudo prévio, quer de teoria sobre grafos quer do caso de estudo, possibilitaram que todo o processo decorresse dentro da normalidade, sem que tenham surgido problemas de dimensão considerável.

6. Anexos

```
estima(Nodo,Destino,Estima) :- distance(Nodo,Destino,Estima).

distance(N1,N2,D) :- lat(N1,Lat1),lon(N1,Long1),
                     lat(N2,Lat2),lon(N2,Long2),
                     D is sqrt((Long2-Long1)*(Long2-Long1) + (Lat2-Lat1)*Lat2-Lat1).

lat(N,Lat) :- paragem(N,Lat,_,_,_,_,_,_,_,_,_).
lon(N,Lon) :- paragem(N,_,Lon,_,_,_,_,_,_,_,_).

resolve_gulosa(Nodo,Ending,Caminho/Custo):- estima(Nodo,Ending,Estima),
                                    agulosa([[Nodo]/0/Estima],InvCaminho/Custo/_,Ending),
                                    inverso(InvCaminho,Caminho).

agulosa(Caminhos,Caminho,Ending):- obtem_melhor_g(Caminhos,Caminho),
                            Caminho = [Nodo|_]/_/_,Nodo is Ending.

agulosa(Caminhos,SolucaoCaminho,Ending):-obtem_melhor_g(Caminhos,MelhorCaminho),
                                  seleciona(MelhorCaminho,Caminhos,OutrosCaminhos),
                                  expande_gulosa(MelhorCaminho,ExpCaminhos,Ending),
                                  append(OutrosCaminhos,ExpCaminhos,NovoCaminhos),
                                  agulosa(NovoCaminhos,SolucaoCaminho,Ending).

obtem_melhor_g([Caminho],Caminho):- !.

obtem_melhor_g([Caminho1/Custo1/Est1,_/Custo2/Est2|Caminhos],MelhorCaminho):-
    Est1 =< Est2,
    !,
    obtem_melhor_g([Caminho1/Custo1/Est1|Caminhos],MelhorCaminho).


obtem_melhor_g([_|Caminhos],MelhorCaminho):- obtem_melhor_g(Caminhos,MelhorCaminho).

expande_gulosa(Caminho,ExpCaminhos,Ending):-
    findall(NovoCaminho,adjacente(Caminho,NovoCaminho,Ending),ExpCaminhos).

adjacente([Nodo|Caminho]/Custo/_, [ProxNodo,Nodo|Caminho]/NovoCusto/Est,Ending) :-
    move(Nodo, ProxNodo, PassoCusto,_),
    \+ member(ProxNodo, Caminho),
    NovoCusto is Custo + PassoCusto,
    estima(ProxNodo,Ending, Est).
```

7. Referências

Pesquisa Informada/Desinformada: https://pt.gadget-info.com/difference-between-informed