Universidade do Minho

Relatório do Exercício Individual

Mestrado Integrado em Engenharia Informática

Sistemas de Representação de Conhecimento e Raciocínio

(2º Semestre/2019-2020)

Carlos Manuel Marques Afonso

(A82529)

Braga,

julho de 2020

Resumo

Com a realização deste exercício, pretende-se aprofundar a utilização da Programação em Lógica, usando a linguagem de programação PROLOG, no âmbito de métodos de Resolução de Problemas e no desenvolvimento de algoritmos de pesquisa.

No desenvolvimento das soluções, consideraram-se diferentes estratégias de pesquisa (não-informada e informada) e apresentou-se, posteriormente, uma tabela comparativa com as propriedades das estratégias utilizadas.

Conteúdo

Índice de Figuras

Introdução

Este exercício foi proposto pelos docentes da unidade curricular de Sistemas de Representação de Conhecimento e Raciocínio, e tem como principal objetivo o desenvolvimento de um sistema que permita importar os dados fornecidos, representados numa base de conhecimento.

No desenvolvimento das soluções, consideraram-se diferentes estratégias de pesquisa (não-informada e informada) e, por fim, de forma a comparar os diferentes algoritmos de pesquisa, foi elaborada uma tabela comparativa (com as propriedades das estratégias) com as que utilizou.

Descrição do Trabalho e Análise de Resultados

Para a resolução do enunciado proposto pelos docentes da unidade curricular, foi considerada a caracterização de conhecimento, dado na seguinte forma:

cidade: *Id, Cidade, Latitude, Longitude, Admin, Capital, Características -> { V, F }*
ligacao: *Cidade1, Cidade2 -> { V, F }*

A elaboração deste caso prático deverá, assim, permitir:

1. Calcular um trajeto entre duas cidades;
2. Selecionar apenas cidades, com uma determinada característica, para um determinado trajeto;
3. Excluir uma ou mais características de cidades para um percurso;
4. Identificar num determinado percurso qual a cidade com maior número de ligações;
5. Escolher o menor percurso (usando o critério do menor número de cidades percorridas);
6. Escolher o percurso mais rápido (usando o critério da distância);
7. Escolher um percurso que passe apenas por cidade "minor";
8. Escolher uma ou mais cidades intermédias por onde o percurso deverá obrigatoriamente passar.

Antes de ser implementado qualquer tipo de solução para estas queries, foi implementado um *parser*, de forma a que o *Dataset* fosse introduzido na base de conhecimento.

Após uma conversão do ficheiro *.xlsx* para *.csv*, foi implementado, em JAVA, um *parser*, com o objetivo de transpor os dados de ficheiro *.csv* para o ficheiro *.pl*.

É, também, necessário destacar o facto de que, por algum motivo, o *sicstus* apresentava erros de compilação aquando da existência de um ficheiro com acentos agudos em alguns casos excecionais, o que levou à eliminação daqueles que levantavam erros.

Quanto às descrições de cada cidade, estas foram acrescentadas posteriormente à base de conhecimento. No que diz respeito às ligações entre cidades, estas foram geradas de forma aleatória, com recurso a um *randomizer*.

Respostas às Queries

1. Calcular um trajeto entre duas cidades

Para esta *query*, foram implementados dois algoritmos. O algoritmo de procura primeiro em profundidade, e o de procura primeiro em largura [2].

Quanto à procura em largura, a solução apresentada passa por obter logo uma ligação a partir da Origem introduzida pelo utilizador. De seguida, o algoritmo pesquisa as ligações seguintes, de forma a que o Destino seja alcançado, sem que, no entanto, se passe por nodos previamente visitados.

No que diz respeito à procura em profundidade, a ideia é semelhante à da procura em largura, sendo que este abstém-se de tanta complexidade.

2. Selecionar apenas cidades, com uma determinada característica, para um determinado trajeto

A solução implementada para esta *query* é muito semelhante à da *query* anterior, uma vez que apenas foi acrescentado o predicado "temCaracteristica", que verifica se o nodo onde estamos tem a característica que o utilizador introduziu, tanto na procura em profundidade, como na procura em largura.

3. Excluir uma ou mais características de cidades para um percurso

Para esta *query*, foram implementadas soluções que recorrem à procura em largura e à procura em profundidade.

Não fugindo à similaridade das soluções anteriores, a única alteração está na introdução de um predicado, sendo que agora é verificado se uma cidade tem características que pertencem à lista de características introduzidas pelo utilizador e, caso tal seja verificável, essa cidade é ignorada.

4. Identificar num determinado percurso qual a cidade com maior número de ligações

No que diz respeito a esta *query*, apenas foi implementado um algoritmo de **pesquisa em largura.**

A solução passa por fazer uma pesquisa igual à da *query* 1. Quando o resultado é obtido, o predicado "meteNrLigacoes" é aplicado à lista resultante, que irá obter uma lista com tuplos onde o primeiro elemento é a cidade, e o segundo é o número de ligações. Por fim, o predicado "máximo" é aplicado nesta lista de tuplos, obtendo a cidade com a maior número de ligações.

5. Escolher o menor percurso (usando o critério do menor número de cidades percorridas)

Para a realização desta *query*, foi utilizado o método de **procura em largura**.

Dadas duas cidades, uma origem e um destino, vão ser procuradas todas as ligações à cidade origem, verificando que estas não se encontram previamente na lista de nodos visitados nem de nodos em lista de espera. De seguida, calcula-se, para a lista resultante, o comprimento (conseguido através do predicado auxiliar "calculaCidades") entre cada nodo e o destino, através do método de procura em largura, obtendo uma lista de duplos (Ligada, #Cidades). Esta lista será depois ordenada por ordem crescente de número de cidades percorridas, ao qual será retirado o primeiro elemento. Irá ser aplicado este algoritmo recursivamente até chegar à cidade destino.

6. Escolher o percurso mais rápido (usando o critério da distância)

Dado que esta *query* requer a utilização de um critério de distância para calcular o percurso mais rápido entre duas cidades, foi necessária a utilização de um método de procura informada. Por este motivo, recorreu-se ao algoritmo **A***.

O primeiro passo para a implementação deste algoritmo está no cálculo do predicado "estima". Assim sendo, este predicado foi caracterizado pela distância Euclidiana entre duas cidades.

Seguindo a implementação deste algoritmo realizado nas aulas teórico-práticas, foi possível a adaptação do mesmo à situação em questão.

7. Escolher um percurso que passe apenas por cidade “minor”

Para esta *query*, foram implementadas três soluções distintas. A procura em largura, a procura em profundidade e o algoritmo A*.

A procura em largura e a procura em profundidade são semelhantes à resposta à *query* 1. No entanto, desta vez, à medida que se obtém uma ligação, verifica-se se a cidade que está nessa ligação tem a capital *minor*.

No que diz respeito ao algoritmo A*, adaptou-se o algoritmo inicial, adicionando a este um método de verificação se a cidade em questão tem a capital *minor*.

8. Escolher uma ou mais cidades intermédias por onde o percurso deverá obrigatoriamente passar

Por fim, para esta *query*, a solução implementada foi uma procura em largura, devido à sua simplicidade.

A realização desta *query* passa pela procura recursiva de trajetos possíveis até alcançar a cidade destino. Assim sendo, a partir de uma cidade origem, vai ser calculado um trajeto até à primeira cidade intermédia introduzida pelo utilizador. De seguida, será calculado um novo trajeto entre as cidades intermédias, até que é calculado um trajeto entre a última cidade intermédia e a cidade destino. A concatenação entre todos os trajetos calculados resulta no percurso desejado.

Conclusões e Sugestões

Após análise dos resultados obtidos, é visível uma clara distinção entre os diversos algoritmos implementados.

Algoritmo	Completo	Ótimo	Tempo
Procura em Largura	Sim	Às vezes	Bom
Procura em Profundidade	Não	Não	Bom
A*	Sim	Sim	Médio

No que diz respeito à **procura em profundidade**, esta evidenciou que os resultados obtidos não podem ser considerados ótimos. Para além disso, é um algoritmo que revela bons tempos de execução. No entanto, verifica-se um atraso significativo para trajetos de maiores dimensões.

Quanto à **procura em largura**, o conjunto de resultados mostrou serem bons resultados, obtendo, por vezes, resultados ótimos, e é, sem dúvida, o algoritmo mais rápido.

Por último, quanto ao algoritmo **A***, o conjunto de caminhos obtidos foi sempre ótimo. No entanto, devido à sua elevada complexidade, quando é pedido um trajeto de maiores dimensões, o tempo de execução aumenta, tornando-se num algoritmo lento.

De notar que, por vezes, alguns trajetos não têm resposta, uma vez que entra em ciclo infinito. Este problema deve-se ao facto de as ligações terem sido feitas "à mão", não garantindo, por vezes, saídas de um caminho, levando à criação de um ciclo.

Em suma, considero que fiz um bom trabalho, pois implementei vários algoritmos, oferecendo uma variedade de soluções, e o meu conhecimento de programação em Lógica, usando a linguagem de programação PROLOG, foi aprofundado, sendo esse o grande objetivo desta Unidade Curricular.

Referências

[1] Métodos de resolução de problemas e de procura

[2] Introductory notes on Prolog and AI Search http://www.cs.ukzn.ac.za/~hughm/ai/notes/prologsearch.pdf

Anexos

```
public static void main(String[] args) throws IOException {

    String csvFile = "C:\\Users\\f6car\\Desktop\\Trabalho SRCR\\cidades.csv
        ";
    BufferedReader br = null;
    String line = "";
    String cvsSplitBy = ",";
    FileWriter myWriter = new FileWriter("C:\\Users\\f6car\\Desktop\\
        Trabalho SRCR\\cidades.pl");

    try {

        br = new BufferedReader(new FileReader(csvFile));
        br.readLine();

        while ((line = br.readLine()) != null) {


            String[] country = line.split(cvsSplitBy);

            myWriter.write("cidade(" + country[0] + "," + '"' + country[1]
                + '"' + "," + country[2] + "," + country[3] + "," + '"' +
                country[4] + '"' + "," + '"' + country[5] + '"' + ")." +"\n
                ");

        }
        myWriter.close();

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        if (br != null) {
            try {
                br.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

}
```

Figura 1 - Parser

```
| ?- bfs('Braga','Fafe',R).
R = ['Braga','Portalegre','Montemor-o-Velho','Anadia','Alter do ChÃ£o','Seia','
SÃ£o Pedro do Sul','Vila Velha de RÃ³dÃ£o','Vale de Cambra','Odivelas','Fafe']
?
yes
```

Figura 2 - Query1 BFS

```
| ?- dfsCaracteristica('Ovar','Anadia','Sem Descricao',R).
R = ['Ovar','GolegÃ£','Mealhada','Moita','Odivelas','Vila Verde','Alenquer','Sa
nta Marta de PenaguiÃ£o','Mondim de Basto','Moura','Torres Vedras','Faro','Cast
ro Verde','Porto de MÃ³s','Odemira','SÃ£o Pedro do Sul','Seia','Alter do ChÃ£o'
,'Anadia'] ?
```

Figura 3 - Query2 DFS com característica "Sem Descrição"

```
| ?- bfsMaisCaracteristica('Braga','Porto',['Patrimonio Mundial','Capital do Surf'],R).
R = ['Braga','Portalegre','Montemor-o-Velho','Anadia','Alter do ChÃ£o','Seia','SÃ£o Pedro
 do Sul','Vila Velha de RÃ³dÃ£o','Vale de Cambra','Odivelas','Vila Verde','Alenquer','Alc
outim','GrÃ¢ndola','Arouca','Arraiolos','Miranda do Douro','Campo Maior','MesÃ£o Frio','P
orto'] ?
```

Figura 4 - Query3 BFS excluir características

```
| ?- bfsNumeroLigacoes('Vila Real','Moura',R).
R = ('Tavira',5) ?
yes
```

Figura 5 - Query 4 BFS maior número de ligações

```
| ?- bfsMenosCidades('Cartaxo','Vila Franca de Xira',R).
R = ['Cartaxo','Vouzela','Santa Comba DÃ£o','Guarda','Almada','Elvas','Seixal','Vila Nova
 de Gaia','Paredes de Coura','Espinho','Vila Franca de Xira'] ?
yes
```

Figura 6 - Query 5 BFS caminho com menor número de cidades percorridas

```
| ?- fastestEstrela('Braga','Porto', R).
R = ['Braga','Portalegre','Penacova','Vinhais','Alcanena','LourinhÃ£','Estarreja','Cartax
o','Rio Maior','Castelo de Vide','Arraiolos','Miranda do Douro','Campo Maior','MesÃ£o Fri
o','Porto']/2494.2707910442846 ?
```

Figura 7 - Query 6 A menor distância percorrida*

```
| ?- fastestEstrelaMinor('Paredes de Coura','Ovar',R).
R = ['Paredes de Coura','Vila Nova de Gaia','Odivelas','Moita','Mealhada','GolegÃ£','Ovar
']/842.7395721649326 ?
```

Figura 8 - Query 7 A cidades minor*

```
| ?- bfsCidades('Braga','Porto',['Ovar','Fafe'],R).
R = ['Braga','Portalegre','Montemor-o-Velho','Anadia','Alter do ChÃ£o','Seia','SÃ£o Pedro d
o Sul','Vila Velha de RÃ³dÃ£o','Abrantes','CovilhÃ£','Fornos de Algodres','ValenÃ§a','Ovar'
,'GolegÃ£','Mealhada','Moita','Odivelas','Fafe','Odivelas','Vila Verde','Alenquer','Alcouti
m','GrÃ¢ndola','Arouca','Arraiolos','Miranda do Douro','Campo Maior','MesÃ£o Frio','Porto']
 ?
```

Figura 9 - Query 8 BFS caminho que tem de passar obrigatoriamente por certas cidades