UNIVERSIDADE
DE ÉVORA

Relatório do 2º trabalho
Inteligência Artificial

Trabalho realizado por:

- Henrique Rosa, nº 51923

- Diogo Matos, nº 54466

24/04/2024

A)

Variáveis e representação dos estados:

É possível resolver este problema com nove variáveis, em que cada uma das variáveis representa uma posição no tabuleiro (o tabuleiro é nxn e n=3, logo, o tabuleiro tem 9 posições). Cada uma das variáveis tem o seu nome que é composto por um elemento pos(X,Y), em que X e Y seriam as posições do tabuleiro. Também iria ter o seu domínio (igual para todas as variáveis, sendo os números de 1 a 9, ou seja, os valores que cada posição no tabuleiro pode conter) e o seu valor (o valor que a posição que essa variável representa contém).As variáveis seguem este formato: **v(Nome, [Dominio] , Valor).**

O estado da nossa solução contem duas listas: LNI , que seria a lista com as variáveis ainda não instanciadas, e LI, a lista com as variáveis instanciadas, que já têm valor. Os estados seguem este formato: **e([LNI] , [LI]).**

Estado Inicial em prolog:

```
21
22  estado_inicial(e([  v(pos(1,1),[1,2,3,4,5,6,7,8,9],_),
23                      v(pos(2,1),[1,2,3,4,5,6,7,8,9],_),
24                      v(pos(3,1),[1,2,3,4,5,6,7,8,9],_),
25                      v(pos(1,2),[1,2,3,4,5,6,7,8,9],_),
26                      v(pos(2,2),[1,2,3,4,5,6,7,8,9],_),
27                      v(pos(3,2),[1,2,3,4,5,6,7,8,9],_),
28                      v(pos(1,3),[1,2,3,4,5,6,7,8,9],_),
29                      v(pos(2,3),[1,2,3,4,5,6,7,8,9],_),
30                      v(pos(3,3),[1,2,3,4,5,6,7,8,9],_)   ], [])).
31
```

Operador Sucessor em prolog:

```
16
17  sucessor(e([v(N,D,_)|R],E),e(R,[v(N,D,V)|E])):-
18      member(V,D).
19
```

Restrições:

As restrições para esta representação seriam:

- Nenhuma posição (variável) pode ter o mesmo valor que outra, i.e não pode haver dois números iguais;
- Nenhuma posição pode estar vazia no final;
- Todas as linhas, colunas e diagonais têm de ter o mesmo valor de soma das suas componentes.

Estas restrições estão implementadas em prolog no ficheiro magic.pl que segue com este relatório.

B) O problema foi resolvido executando a pesquisa backtracking conforme implementado no ficheiro magic.pl (a lógica é exatamente igual à pesquisa com backtracking dada nas aulas, mas com a adição da contagem de nós).

O código da pesquisa BackTracking usado é o seguinte:

```
88   pb:- estado_inicial(E0), back(E0,A), esc(A).
89
90   back(e([],A),A).
91   back(E,Sol):- sucessor(E,E1),
92       inc,
93       ve_restricoes(E1),
94       back(E1,Sol).
```

A primeira solução teve um total de 1069210 nós visitados, e para encontrar todas as soluções foi necessário visitar um total de 4543569 nós (o total de tempo de execução necessário para achar todas as soluções foi de dois minutos). Abaixo estão **todas** as soluções que o programa apresentou, cada uma delas identificada pelo número correspondente na sequência em que foram descobertas e apresentadas pelo programa - no total são 8, exatamente as esperadas para um quadrado mágico 3x3. Depois de encontrada a oitava solução para o problema, a execução mostra "no", significando isto que já não há mais soluções possíveis (exatamente o que era pretendido).

Soluções com pesquisa backtracking:

```
| ?- p.
8 | 3 | 4   (1)
--------
1 | 5 | 9
--------
6 | 7 | 2

nós visitados:  1068210

true ? ;
8 | 1 | 6   (2)
--------
3 | 5 | 7
--------
4 | 9 | 2

nós visitados:  1199691

true ? ;
6 | 7 | 2   (3)
--------
1 | 5 | 9
--------
8 | 3 | 4

nós visitados:  2092194

true ? ;
6 | 1 | 8   (4)
--------
7 | 5 | 3
--------
2 | 9 | 4

nós visitados:  2429802

true ? ;
4 | 9 | 2   (5)
--------
3 | 5 | 7
--------
8 | 1 | 6

nós visitados:  3181977

true ? ;
4 | 3 | 8   (6)
--------
9 | 5 | 1
--------
2 | 7 | 6

nós visitados:  3519585

true ? ;
2 | 9 | 4   (7)
--------
7 | 5 | 3
--------
6 | 1 | 8

nós visitados:  4412088

true ? ;
2 | 7 | 6   (8)
--------
9 | 5 | 1
--------
4 | 3 | 8

nós visitados:  4543569
```

C) O código do algoritmo modificado para realizar Forward Checking foi o seguinte:

```
 98  pbfc:-estado_inicial(E0), back_fc(E0,A),  esc(A).
 99
100  back_fc(e([],A),A).
101      back_fc(E,Sol):- sucessor(E,E1), inc, ve_restricoes(E1),
102      forCheck(E1,E2),
103      back_fc(E2,Sol).
104
105  forCheck(e(Lni,[v(N,D,V)|Li]), e(Lnii,[v(N,D,V)|Li])):-corta(V,Lni,Lnii).
106
107  corta(_,[],[]).
108  corta(V,[v(N,D,_)|Li], [v(N,D1,_)|Lii]):- delete(D,V,D1), corta(V,Li,Lii).
```

Este algoritmo chegou exatamente às mesmas soluções pela mesma ordem. O número de nós visitados para chegar à primeira solução foi de 187768 nós, e para todas as soluções foi necessário visitar um total de 798650 nós (tempo total de 24 segundos).

Primeira solução:

```
| ?- pbfc.
8 | 3 | 4
---------
1 | 5 | 9
---------
6 | 7 | 2

nós visitados:  187768
```

Última solução:

```
true ? ;
2 | 7 | 6
---------
9 | 5 | 1
---------
4 | 3 | 8

nós visitados:  798650
```

D) O algoritmo de pesquisa backtracking com forward checking tem basicamente a função de retirar valores do domínio das variáveis ainda não instanciadas, valores esses que levam obrigatoriamente à falha (soluções que não respeitam as restrições).

Neste problema, como nenhuma posição do quadrado mágico pode ter um valor que já esteja noutra posição (não pode haver valores repetidos), os valores que o forward checking vai retirar dos domínios são os valores que já foram atribuídos às variáveis instanciadas, e esse valor vai ser retirado de todas as variáveis por instanciar, logo a lógica de dar prioridade às variáveis com menor domínio não iria fazer diferença, já que todas as variáveis por instanciar iriam ter exatamente o mesmo domínio.

Como tal, procedemos à lógica de alterar a ordem pela qual as variáveis são instanciadas, para que seja dada prioridade às variáveis com maior número de restrições. As posições com maior número de restrições são, por ordem:

- 4 restrições : A posição do meio (2,2):
- 3 restrições: as posições das diagonais [(1,1) , (3,3) , (3,1), (1,3)]:
- 2 restrições: todas as outras posições.

Para que isto seja feito, basta alterar a ordem da lista de variáveis não instanciadas do estado inicial, dado que, pelo predicado sucessor, as variáveis vão ser instanciadas pela ordem em que estão na lista. Esta foi a modificação feita:

```
98  pbfc:-estado_inicial(E0), order(E0,E0_), back_fc(E0_,A),  esc(A).
99
100 back_fc(e([],A),A).
101     back_fc(E,Sol):- sucessor(E,E1), inc, ve_restricoes(E1),
102     forCheck(E1,E2),
103     back_fc(E2,Sol).
104
105 forCheck(e(Lni,[v(N,D,V)|Li]), e(Lnii,[v(N,D,V)|Li])):-corta(V,Lni,Lnii).
106
107 corta(_,[],[]).
108 corta(V,[v(N,D,_)|Li], [v(N,D1,_)|Lii]):- delete(D,V,D1), corta(V,Li,Lii).
```

```
112 order(e(Lni,Li),e(LniO,Li)):-
113     member(v(pos(2,2),D,V),Lni),
114     List = [v(pos(2,2),D,V),
115             v(pos(1,3),D,V),
116             v(pos(3,1),D,V),
117             v(pos(1,1),D,V),
118             v(pos(3,3),D,V)],
119     remove_list(Lni,List,Aux),
120     append(List,Aux,LniO).
121
122
123 remove_list([], _, []).
124 remove_list([X|Tail], L2, Result):- member(X, L2), !, remove_list(Tail, L2, Result).
125 remove_list([X|Tail], L2, [X|Result]):- remove_list(Tail, L2, Result).
```

Ao contrário do que estávamos à espera, o número de nós visitados para encontrar a primeira solução aumentou, sendo necessário visitar 462706 nós em vez de apenas 187768. já o número de nós necessários para encontrar todas as soluções diminuiu significativamente, sendo esse número 523712 em vez de 798650

Primeira solução:

```
| ?- pbfc.
7 | 1 | 9
---------
3 | 6 | 4
---------
8 | 2 | 5

nós visitados:  462706
```

Última solução:

```
true ? ;
3 | 9 | 1
---------
7 | 4 | 6
---------
2 | 8 | 5

nós visitados:  523712
```

D) Realizamos a execução do predicado pbfc (backtracking com forward checking) num código modificado para comportar o problema do quadrado mágico com n=4* . Depois de um período de espera da mais de 20 minutos, a primeira solução ainda não tinha sido encontrada, pelo que achámos pertinente parar a execução por demorar demasiado tempo. Assumindo que a pesquisa com backtracking leva mais tempo e que quanto maior o n, mais tempo leva até à primeira solução, assumimos que não conseguimos resolver esta alínea pelo motivo de alta complexidade temporal. Incluímos na pasta do trabalho o ficheiro que usamos para este teste, chamado "**magic_4n.pl**" .

*16 variáveis com domínio de 1 até 16, e com a lógica dos predicados de verificação de restrições ligeiramente alterada para suportar um número de linhas e colunas maior, bem como posições diferentes nas diagonais.

Execução

O nosso código permite a sua execução, bastando compilar o ficheiro "**magic.pl**". Para executar a pesquisa com Backtracking, basta chamar o predicado " **pb.** ", já para executar a pesquisa backtracking com forward checking, basta chamar o predicado " **pbfc.** " - de notar que o predicado pbfc já tem as modificações de prioridade das posições com mais restrições.