Resolução de um problema de decisão com restrições

Trid Puzzle

Autores:

Nuno Miguel Outeiro Pereira up201506265

Ricardo José Santos Pereira up201503716

Instituição de ensino:

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

Sumário

Este artigo/relatório tem como objetivo dar suporte ao trabalho realizado no âmbito da unidade curricular de Programação em Lógica, sendo que o programa desenvolvido é capaz de resolver qualquer Triad com N maior ou igual a 5 e menor ou igual a 34.

´

1 Introdução

O trabalho desenvolvido tem como elo comum, independentemente do tema escolhido, a resolução de um problema de decisão em Prolog usando restrições.

O nosso tema é o Trid, pelo que o desafio com que nos deparámos foi resolver uma pirâmide composta por nós com números de forma a que não existissem números repetidos em qualquer linha reta, independentemente da sua direção. Outro condicionante seria, o eventual, número entre 3 nós, que caso existisse, indiciaria a soma desses mesmos nós.

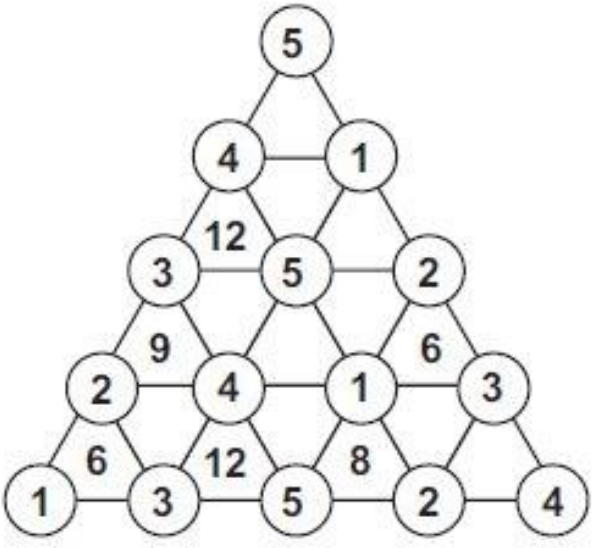
Ao longo deste artigo, será explicado a abordagem utilizada na implementação realizada para solucionar o problema apresentado, assim como explicitar a sua representação, avaliar a complexidade dos resultados e irá ser incluído no mesmo também uma conclusão.

2 Descrição do problema

O Trid é uma pirâmide com vários nós, em que cada um desses nós, no final da sua resolução, terá de possuir todos os nós preenchidos com números, em que cada linha reta, em qualquer das 3 direções, não possua o mesmo número 2 vezes.

Adicionalmente, no triângulo formado no interior de cada 3 nós, pode existir um número, que caso exista será igual a soma entre os 3 nós que circundam esse número.

A complexidade de resolução deste problema aumenta conforme o N também aumenta, sendo este N a altura da pirâmide, isto é, o número de camadas que a pirâmide tem.



3 Abordagem

3.1 Variáveis de decisão

A solução do puzzle é representada internamente por uma lista simples. Como tal, as variáveis de decisão constituem uma lista simples que contém os números de cada nó organizados da esquerda para a direita, e de, cima para baixo, com domínio de 1 a N.

3.2 Restrições

Para além da restrição imposta na nossa implementação para o N (número de camadas da pirâmide) ser compreendido entre 5 e 34, inclusive, foram impostas as demais restrições para que as regras de completamento fossem cumpridas de forma a bem solucionar o puzzle.

A chamada a esses predicados de restrições está localizada no predicado *trid_logic* como a seguir se confirma.

```
trid_logic(Layer, Sums, Numbers):-
    Size is div(Layer * (Layer + 1), 2),
    length(Numbers, Size),
    domain(Numbers, 1, Layer),
    distinct_horizontal(Layer, Numbers),
    distinct_vertical(Layer, Numbers),
    distinct_diagonal(Layer, Numbers),
    constraint_sums(Sums, Numbers),
    labeling([ffc], Numbers),
    !.
```

A forma escolhida para representar a pirâmide simplificou um pouco a implementação das restrições essenciais para o correto solucionamento do puzzle. Abaixo, segue em anexo a implementação das restrições acima mencionadas.

```
%% Logic

distinct_horizontal(Layer, Numbers):-
    distinct_horizontal(Layer, 1, Numbers).
distinct_horizontal(Layer, I, Numbers):-
    horizontal_numbers(Numbers, I, Horizontal, NumbersNext),
    all_distinct(Horizontal),
    (I == Layer;
    INext is I + 1,
    distinct_horizontal(Layer, INext, NumbersNext)).

distinct_vertical(Layer, Numbers):-
    distinct_vertical(Layer, Layer, Numbers).
distinct_vertical(Layer, I, Numbers):-
    vertical_numbers(Numbers, I, Vertical, NumbersNext),
    all_distinct(Vertical),
    (I == 1;
    INext is I - 1,
    distinct_vertical(Layer, INext, NumbersNext)).

distinct_diagonal(Layer, Numbers):-
    distinct_diagonal(Layer, Layer, Numbers).
distinct_diagonal(Layer, I, Numbers):-
    diagonal_numbers(Numbers, I, Diagonal, NumbersNext),
    all_distinct(Diagonal),
    (I == 1;
    INext is I - 1,
    distinct_diagonal(Layer, INext, NumbersNext)).

constraint_sums([], _).
constraint_sums([sum(V1, V2, V3, S)|Sums], Numbers):-
    element(V1, Numbers, N1),
    element(V2, Numbers, N2),
    element(V3, Numbers, N3),
    S #= N1 + N2 + N3,
    constraint_sums(Sums, Numbers).
```

3.3 Estrutura de pesquisa

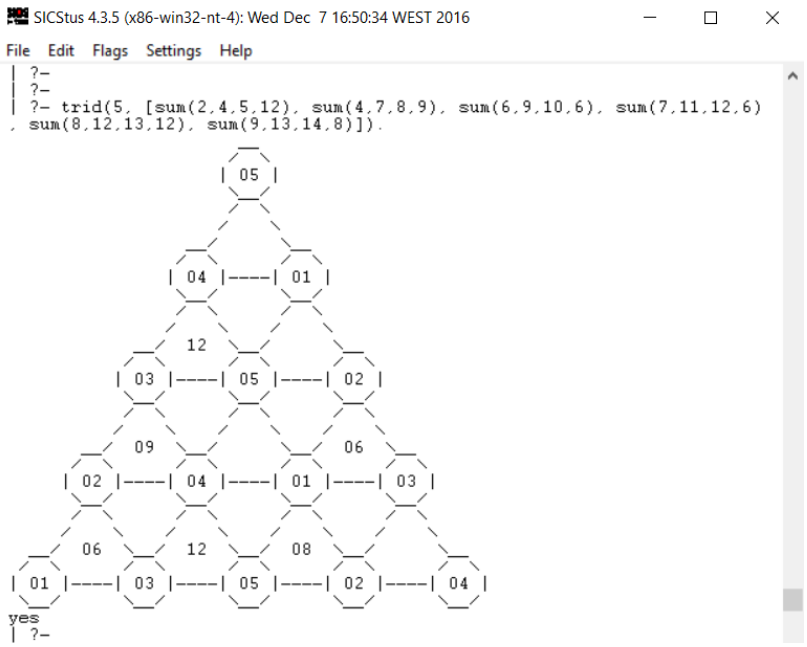
A estrutura de labeling utilizada, de forma a tornar a pesquisa mais eficiente, foi a fcc (first-fail-constraint), ou seja, as variáveis são etiquetadas por ordem de domínios crescentes e número de restrições decrescentes. Sendo que, neste puzzle, todas as variáveis possuem o mesmo domínio, desta forma, o único critério de organização das mesmas é o número de restrições.

4 Visualização da solução

Neste projeto, optámos por representar a pirâmide com a solução obtida após ser inserido o comando que pede pela resolução do problema, sendo que assim, para qualquer chamada ao predicado *trid*, será apresentada a solução usando alguns carateres simples de forma a imitar a pirâmide em cima representada por círculos para os nós.

A seguinte chamada, como exemplo, causa a representação ilustrada no printscreen abaixo inserido.

```
trid(5, [sum(2, 4, 5, 12), sum(4, 7, 8, 9), sum(6, 9, 10, 6), sum(7, 11, 12, 6), sum(8, 12, 13, 12), sum(9, 13, 14, 8)]).
```



5 Resultados

Seguem aqui algumas medições efetuadas em milissegundos para complexidades de 5 a 11, isto é pirâmides com N (número de camadas) de 5 a 11 conforme vários tipos de pesquisa, sendo as mesmas, leftmost, ff e first-fail-constraint respetivamente.

leftmost							average
16	16	16	16	16	16	16	16
62	63	62	94	62	47	47	62,42857
109	125	109	141	63	140	78	109,2857
3766	3984	3844	3859	3719	4079	3937	3884
78	47	63	47	47	79	78	62,71429
2640	2562	2547	2656	2547	2515	2750	2602,429
45234	48453	46531	45297	44688	44265	44406	45553,43

ff							average
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
32	32	32	32	32	32	32	32
31	47	63	47	47	31	31	42,42857
46	63	78	94	78	94	31	69,14286
125	125	110	140	109	125	141	125
157	172	203	204	203	172	204	187,8571

ffc							average
16	16	16	16	16	16	16	16
16	16	16	16	16	16	16	16
32	16	15	16	15	31	32	22,42857
46	63	31	47	47	47	63	49,14286
78	63	62	78	94	94	93	80,28571
141	109	109	125	94	109	140	118,1429
187	172	203	172	172	219	203	189,7143

Após alguma análise de dados semelhante à acima inserida, compreendemos que a forma de obter a solução no menor tempo seria usando a ffc como estrutura de pesquisa, sendo que assim o fizemos.

6 Conclusões

Tendo sido o tema escolhido, pensamos em várias de formas de representar o puzzle atribuído, entre elas uma lista de listas, assim como uma matriz triangular, acabando pela escolha final, uma lista simples. A tradução de ideias para código não ofereceu grandes entraves e foi relativamente simples de solucionar.

Após a conclusão deste projeto, é, de facto, de destacar que concluímos que Prolog é uma linguagem bastante eficiente e com provas dadas na resolução de problemas de lógica.

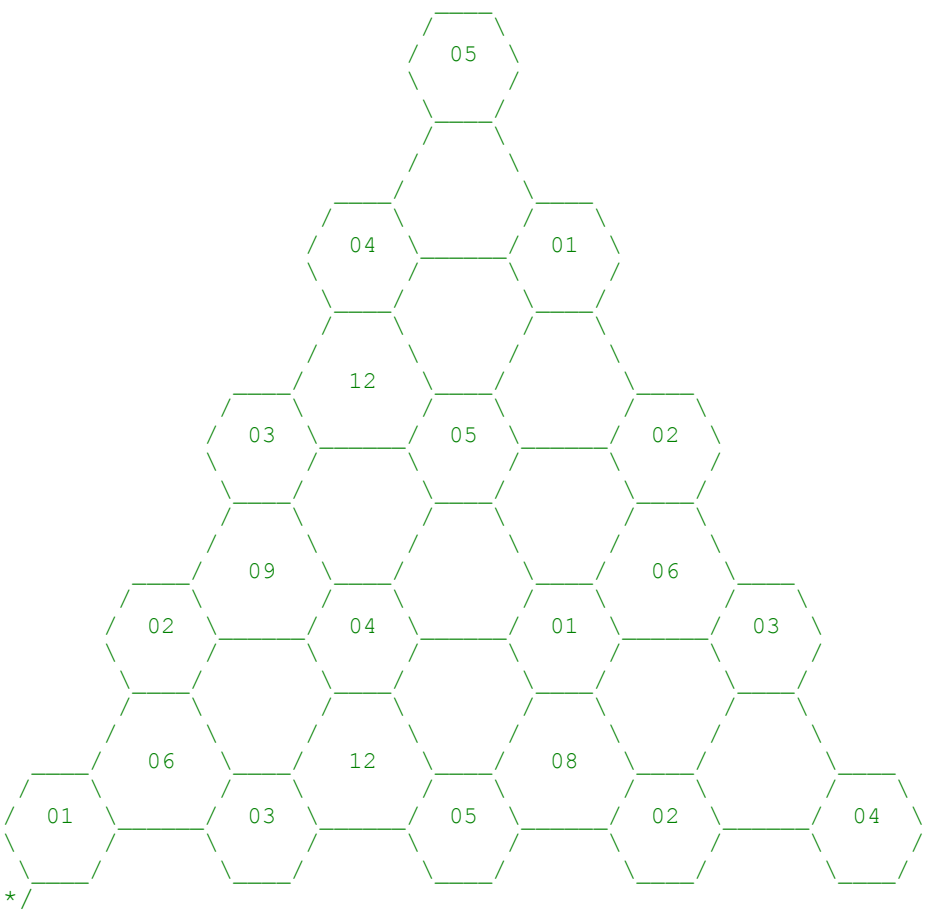
Consideramos também que a realização deste projeto foi bem sucedida visto que tudo foi implementado como previsto, sendo apenas o número de camadas máximo da pirâmide a solucionar restringido a 34, por questões de pertinência e de tempo de resolução.

Bibliografia

1. http://rohanrao.blogspot.pt/2009/05/rules-of-trid.html

7 Anexos

```
:-use_module(library(lists)).
:-use_module(library(clpfd)).
:-dynamic
    sum/4.
/*
Example usage:
trid(5, [sum(2, 4, 5, 12), sum(4, 7, 8, 9), sum(6, 9, 10, 6),
sum(7, 11, 12, 6), sum(8, 12, 13, 12), sum(9, 13, 14, 8)]).
```




```
*/

%% Main

trid(Layer, Sums):-
    trid_check(Layer, Sums),
    trid_logic(Layer, Sums, Numbers),
    trid_display(Layer, Numbers),
```

```
    trid_clear.
trid(_Layer, _Sums):-
    trid_clear,
    fail.

trid_check(Layer, Sums):-
    Layer >= 5, %% Layer == 1 is trivial, Layer == 2 is
impossible, Layer == 3 is sums-trivial, and Layer == 4 is
impossible as well
    Layer =< 34,
    (length(Sums, 0);
    setof(Sum, get_sums(Sums, Sum), [MinSum|SumsRest]),
    MinSum >= 6,
    length(SumsRest, MaxSumIndex),
    nth1(MaxSumIndex, SumsRest, MaxSum),
    MaxSum =< Layer + (Layer - 1) + (Layer - 2)),
    set_sums(Layer, Sums),
    !.

trid_logic(Layer, Sums, Numbers):-
    Size is div(Layer * (Layer + 1), 2),
    length(Numbers, Size),
    domain(Numbers, 1, Layer),
    distinct_horizontal(Layer, Numbers),
    distinct_vertical(Layer, Numbers),
    distinct_diagonal(Layer, Numbers),
    constraint_sums(Sums, Numbers),
    labeling([ffc], Numbers),
    !.

trid_display(Layer, Numbers):-
    display_grid(Layer, Numbers).

trid_clear:-
    clear_sums.

%% Logic

distinct_horizontal(Layer, Numbers):-
    distinct_horizontal(Layer, 1, Numbers).
distinct_horizontal(Layer, I, Numbers):-
    horizontal_numbers(Numbers, I, Horizontal, Number-
sNext),
    all_distinct(Horizontal),
    (I == Layer;
    INext is I + 1,
    distinct_horizontal(Layer, INext, NumbersNext)).
```

```
distinct_vertical(Layer, Numbers):-
    distinct_vertical(Layer, Layer, Numbers).
distinct_vertical(Layer, I, Numbers):-
    vertical_numbers(Numbers, I, Vertical, NumbersNext),
    all_distinct(Vertical),
    (I == 1;
    INext is I - 1,
    distinct_vertical(Layer, INext, NumbersNext)).

distinct_diagonal(Layer, Numbers):-
    distinct_diagonal(Layer, Layer, Numbers).
distinct_diagonal(Layer, I, Numbers):-
    diagonal_numbers(Numbers, I, Diagonal, NumbersNext),
    all_distinct(Diagonal),
    (I == 1;
    INext is I - 1,
    distinct_diagonal(Layer, INext, NumbersNext)).

constraint_sums([], _).
constraint_sums([sum(V1, V2, V3, S)|Sums], Numbers):-
    element(V1, Numbers, N1),
    element(V2, Numbers, N2),
    element(V3, Numbers, N3),
    S #= N1 + N2 + N3,
    constraint_sums(Sums, Numbers).

%% Display

display_grid(Layer, [Number|Numbers]):-
    Y is 1,
    N0 is (Layer - Y) * 7,
    N1 is N0 + 1,
    N2 is N0 + 2,
    number_chars(Number, [NumberChar1|NumberChar2]),
    (length([NumberChar1|NumberChar2], 2),
    atom_chars(N, [NumberChar1|NumberChar2]);
    atom_chars(N, ['0', NumberChar1])),
    writeN(' ', N2), write('____'), nl,
    writeN(' ', N1), write('/    \\'), nl,
    writeN(' ', N0), write('/  '), write(N), write('
\\'), nl,
    writeN(' ', N0), write('\\      /'), nl,
    writeN(' ', N1), write('\\____/'), nl,
    writeN(' ', N1), write('/    \\'), nl,
    writeN(' ', N0), write('/      \\'), nl,
    YNext is Y + 1,
    N0Next is N0 - 7,
    N1Next is N1 - 7,
```

```
    N2Next is N2 - 7,
    display_grid(Layer, YNext, N0Next, N1Next, N2Next,
Numbers).
display_grid(Layer, Y, N0, N1, N2, [Number|Numbers]):-
    number_chars(Number, [NumberChar1|NumberChar2]),
    (length([NumberChar1|NumberChar2], 2),
    atom_chars(N, [NumberChar1|NumberChar2]);
    atom_chars(N, ['0', NumberChar1])),
    writeN(' ', N2), write('____'), display_layer_1(1,
Y), nl,
    writeN(' ', N1), write('/    \\'), display_layer_2(1,
Y), nl,
    writeN(' ', N0), write('/  '), write(N), write('
\\'), display_layer_3(1, Y, Numbers, NumbersNext), nl,
    writeN(' ', N0), write('\\      /'), dis-
play_layer_4(1, Y), nl,
    writeN(' ', N1), write('\\____/'), display_layer_5(1,
Y), nl,
    (Y == Layer;
    writeN(' ', N1), write('/    \\'), display_layer_6(1,
Y), nl,
    writeN(' ', N0), write('/      \\'), dis-
play_layer_7(1, Y), nl,
    YNext is Y + 1,
    N0Next is N0 - 7,
    N1Next is N1 - 7,
    N2Next is N2 - 7,
    display_grid(Layer, YNext, N0Next, N1Next, N2Next,
NumbersNext)).

display_layer_1(Y, Y).
display_layer_1(X, Y):-
    XNext is X + 1,
    YPrev is Y - 1,
    vertexcoord(V1, [X, YPrev]),
    vertexcoord(V2, [X, Y]),
    vertexcoord(V3, [XNext, Y]),
    (sum(V1, V2, V3, Sum),
    number_chars(Sum, [SumChar1|SumChar2]),
    (length([SumChar1|SumChar2], 2),
    atom_chars(S, [SumChar1|SumChar2]);
    atom_chars(S, ['0', SumChar1]));
    S = '  '),
    write('/    '), write(S), write('    \\____'),
    display_layer_1(XNext, Y).

display_layer_2(Y, Y).
display_layer_2(X, Y):-
```

```
    write('        /    \\'),
    XNext is X + 1,
    display_layer_2(XNext, Y).

display_layer_3(Y, Y, Numbers, Numbers).
display_layer_3(X, Y, [Number|Numbers], NumbersNext):-
    number_chars(Number, [NumberChar1|NumberChar2]),
    (length([NumberChar1|NumberChar2], 2),
    atom_chars(N, [NumberChar1|NumberChar2]);
    atom_chars(N, ['0', NumberChar1])),
    write('_____/  '), write(N), write('  \\'),
    XNext is X + 1,
    display_layer_3(XNext, Y, Numbers, NumbersNext).

display_layer_4(Y, Y).
display_layer_4(X, Y):-
    write('      \\      /'),
    XNext is X + 1,
    display_layer_4(XNext, Y).

display_layer_5(Y, Y).
display_layer_5(X, Y):-
    write('       \\____/'),
    XNext is X + 1,
    display_layer_5(XNext, Y).

display_layer_6(Y, Y).
display_layer_6(X, Y):-
    XNext is X + 1,
    YNext is Y + 1,
    vertexcoord(V1, [X, Y]),
    vertexcoord(V2, [XNext, Y]),
    vertexcoord(V3, [XNext, YNext]),
    (sum(V1, V2, V3, Sum),
    number_chars(Sum, [SumChar1|SumChar2]),
    (length([SumChar1|SumChar2], 2),
    atom_chars(S, [SumChar1|SumChar2]);
    atom_chars(S, ['0', SumChar1]));
    S = '  '),
    write('   '), write(S), write('   /    \\'),
    display_layer_6(XNext, Y).

display_layer_7(Y, Y).
display_layer_7(X, Y):-
    write('      /      \\'),
    XNext is X + 1,
    display_layer_7(XNext, Y).
```

```
%% Utilities

writeN(_, 0):-
    !.
writeN(C, N):-
    Next is N - 1,
    write(C),
    writeN(C, Next).

vertexcoord(V, C):-
    vertexcoord(V, 1, [1, 1], C),
    !.
vertexcoord(V, V, C, C).
vertexcoord(V, I, [X, Y], C):-
    INext is I + 1,
    (X =\= Y,
    XNext is X + 1,
    YNext is Y;
    XNext is 1,
    YNext is Y + 1),
    vertexcoord(V, INext, [XNext, YNext], C).

cell(V1, V2, V3):-
    vertexcoord(V1, [X0, Y0]),
    vertexcoord(V2, [X0v1, Y1v0]),
    vertexcoord(V3, [X1, Y1]),
    X1 =:= X0 + 1,
    Y1 =:= Y0 + 1,
    (X0v1 =:= X0, Y1v0 =:= Y1;
    X0v1 =:= X1, Y1v0 =:= Y0),
    !.

get_sums([sum(_, _, _, Sum)], Sum):-
    !.
get_sums([sum(_, _, _, Sum)|_], Sum).
get_sums([_|Sumslist], Sum):-
    get_sums(Sumslist, Sum).

set_sums(_, []).
set_sums(Layer, [sum(V1, V2, V3, S)|Sums]):-
    V1 >= 1,
    V3 =< div(Layer * (Layer + 1), 2),
    cell(V1, V2, V3),
    assertz(sum(V1, V2, V3, S)),
    !,
    set_sums(Layer, Sums).

clear_sums:-
```

```
    retractall(sum(_, _, _, _)).

horizontal_numbers(Numbers, 0, [], Numbers).
horizontal_numbers([HorizontalNumber|NextNumbers], I,
Horizontal, NumbersNext):-
    INext is I - 1,
    horizontal_numbers(NextNumbers, INext, Horizontal-
NumbersNext, NumbersNext),
    Horizontal = [HorizontalNumber|HorizontalNumber-
sNext].

vertical_numbers(Numbers, N, Vertical, NumbersNext):-
    vertical_numbers(Numbers, N, 1, Vertical, Number-
sNext).
vertical_numbers(Numbers, N, N, Vertical, NumbersNext):-
    horizontal_numbers(Numbers, N, [VerticalNumber|Num-
bersNext], _),
    Vertical = [VerticalNumber].
vertical_numbers(Numbers, N, I, Vertical, NumbersNext):-
    INext is I + 1,
    horizontal_numbers(Numbers, I, [VerticalNumber|Hori-
zontalNumbersNext], NextNumbers),
    vertical_numbers(NextNumbers, N, INext, VerticalNext,
VerticalNumbersNext),
    Vertical = [VerticalNumber|VerticalNext],
    append(HorizontalNumbersNext, VerticalNumbersNext,
NumbersNext).

diagonal_numbers(Numbers, N, Diagonal, NumbersNext):-
    diagonal_numbers(Numbers, N, 1, Diagonal, Number-
sNext).
diagonal_numbers(Numbers, N, N, Diagonal, NumbersNext):-
    horizontal_numbers(Numbers, N, HorizontalNumbers, _),
    reverse(HorizontalNumbers, [DiagonalNumber|Number-
sNextReverse]),
    reverse(NumbersNextReverse, NumbersNext),
    Diagonal = [DiagonalNumber].
diagonal_numbers(Numbers, N, I, Diagonal, NumbersNext):-
    INext is I + 1,
    horizontal_numbers(Numbers, I, HorizontalNumbers,
NextNumbers),
    reverse(HorizontalNumbers, [DiagonalNumber|Horizon-
talNumbersNextReverse]),
    reverse(HorizontalNumbersNextReverse, Horizontal-
NumbersNext),
    diagonal_numbers(NextNumbers, N, INext, DiagonalNext,
DiagonalNumbersNext),
    Diagonal = [DiagonalNumber|DiagonalNext],
```

```
      append(HorizontalNumbersNext,   DiagonalNumbersNext,
NumbersNext).
```