Resolução do Puzzle Skyscraper usando Programação em Lógica com Restrições

Pedro Santos e Pedro Ferreira
FEUP-PLOG, Turma 3MIEIC2, Grupo Skyscraper_2

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

Resumo. Este artigo acompanha o código desenvolvido para a unidade curricular de PLOG que encontra a solução de um jogo de 'Skyscraper' de tamanho KxK recorrendo a Programação em Lógica com Restrições.

Keywords/Palavras-Chave: Prolog, Sicstus,Skyscraper, Constraints, Restrições,Feup

1 Introdução

Este trabalho realizado no âmbito da unidade curricular de Programação em Lógica tem como finalidade desenvolver em Programação em Lógica com Restrições uma abordagem de resolução de problemas de decisão/otimização.

Sendo dado aos alunos a hipótese de escolher o puzzle/problema que pretendiam abordar de uma lista de possibilidades, o nosso grupo escolheu o puzzle 'Skyscraper'.

Neste artigo descrevemos detalhadamente o puzzle 'Skyscraper'; a abordagem do grupo para implementar uma resolução capaz de resolver qualquer tabuleiro deste puzzle; estatísticas de resolução de puzzles com diferentes complexidades e por fim a conclusão do projeto realizado.

2 Descrição do Problema

O 'Skyscraper' é um puzzle de colocação de edifícios numa grelha de KxK com algumas pistas aplicadas a linhas e colunas. O objetivo é colocar um edifico em cada célula, com uma altura compreendida entre 1 e K (tamanho do puzzle), de forma a que não hajam dois edifícios em cada linha/coluna com a mesma altura (formando, portanto, um quadrado latino). Para além desta restrição temos ainda que o número de edifícios visíveis, vistos a partir da direção de cada pista deve ser igual ao valor da pista, note que edifícios mais altos bloqueiam a visão de edifícios mais baixos colocados por trás destes.

No exemplo em baixo temos um 'Skyscraper' puzzle de 5x5 com 5 colunas e 5 linhas onde é necessário colocar edifício de altura 1 até edifícios de altura 5 em cada linha/coluna seguindo as regras descritas em cima.

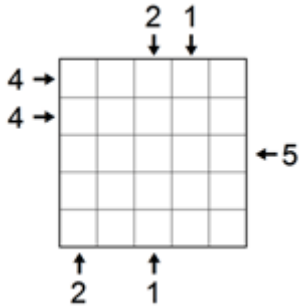


Fig. 1. Exemplo de um estado inicial de um puzzle de ‘Skyscraper’ com 5 linhas e 5 colunas

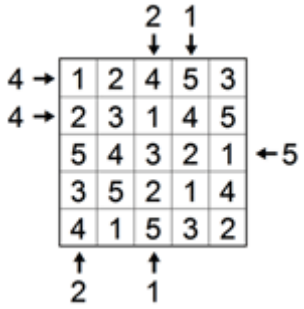


Fig. 2. O mesmo puzzle resolvido de acordo com todas as restrições definidas em cima.

3 Abordagem

O primeiro passo do grupo na abordagem foi tentar perceber como modelar o puzzle como um problema de restrições.

Fazendo um *brainstorming* de quais deveriam ser as variáveis de decisão a utilizar no predicado de labeling, bem como quais as restrições adequadas ao ‘Skyscraper’ a utilizar.

3.1 Variáveis de decisão

Decidimos representar o tabuleiro como uma lista de listas em que cada célula representa a altura do edifício aí colocado.

A solução pretendida para este puzzle é o próprio tabuleiro de jogo completamente preenchido e respeitando todas restrições e pistas.

Neste sentido, a única variável de decisão que a nossa abordagem necessita é a variável *Skyscraper* que é uma lista de listas que representa o tabuleiro de jogo.

3.2 Restrições

A resolução do problema pode ser resumida em duas restrições:

1. O tabuleiro é um quadrado latino.
2. Respeitar o número de edifícios visualizados indicado pelas pistas em cada linha/coluna.

O tabuleiro é um quadrado latino:

```
61  %Create the board and apply the latin square constraints
62  applyLatinSquareConstraints(Skyscraper,K):-
63    rows(K, Skyscraper),
64    transpose(Skyscraper, SkyscraperT),
65    rows(K, SkyscraperT).
```

Após a inicialização do tabuleiro *Skyscraper*, de ordem K, o predicado ***applyLatinSquareConstraints/2*** aplica ao tabuleiro a restrição dos quadrados latinos (uma matriz de ordem K preenchida de K diferentes símbolos de tal maneira que estes ocorrem no máximo uma vez em cada linha ou coluna).

```
112 % checks if every row is of length N and contains each integer from 1 to N exactly once
113 rows(_, []).
114 rows(N, [Head|Tail]):-
115     length(Head, N),
116     domain(Head, 1, N),
117     all_distinct(Head),
118     rows(N, Tail).
```

Este predicado é auxiliado por ***rows/2*** que percorre a ainda lista Skyscraper (que neste momento representa as linhas do tabuleiro) de tamanho K e a transforma numa matriz K*K (K linhas com K colunas).

Em ***rows/2,***com o auxílio de ***domain/3*** cada elemento só vai ter valores no domínio de 1 a K.Fazemos uso do predicado ***all_distinct/1***, que nos garante que para a lista a ser verificada, os elementos ocorrem no máximo uma vez. Numa primeira chamada a restrição é aplicada às linhas do tabuleiro.

De seguida em ***applyLatinSquareConstraints/2*** aplicamos um ***transpose/2*** a Skyscraper e chamando de novo ***rows/2,*** garantimos a restrição em cada coluna.

Respeitar o número de edifícios visualizados indicado pelas pistas nas linhas e colunas:

Para aplicar esta restrição recorremos ao predicado ***visible/3***.Este predicado recebe uma lista de listas *Visible*, que contem as pistas que indicam a quantidade de edifícios visualizados a partir de uma determinada direção(norte,este,sul,oeste) para uma determinada linha ou coluna. Este predicado percorre todas as pistas que vão restringir a solução do puzzle.

```
% visnum(L, K): the number of left-visible numbers in L is K.
visnum([], 0).
visnum([L|Ls], K):-
    visnum(Ls, K, L).
visnum([], 1, _).
visnum([Head|Tail],K, Greatest):-
    (Head #> Greatest) #<=> B,
    B #=> Greatest1 #= Head,
    #\B #=> Greatest1 #= Greatest,
    K #= K1+B,
    visnum(Tail, K1, Greatest1).
```

Com auxílio do predicado v***isnum/2***, que recebe uma linha ou coluna, vai então aplicar a restrição do número de edifícios visualizados. Neste predicado fazemos uso de uma variável booleana B. Usamos a restrição aritmética #<=> para inicializar o valor de B. Caso o elemento Head(valor à cabeça da lista) seja maior que Greateast(maior valor da lista) toma o valor true(1) senão toma o valor false(0). De seguida, dependendo do valor de B, assumimos, ou não, um novo valor para o maior valor da lista. Este predicado vai então organizar uma sequência de números, de maneira a respeitar a pista anteriormente recebida.

No final,um outro predicado chamado ***flattenList/2*** retira qualquer lista de dentro de outras listas,garantindo que todos os valores da matriz são colocados numa única e última lista,para ser usada com argumento do predicado ***labeling/2.***

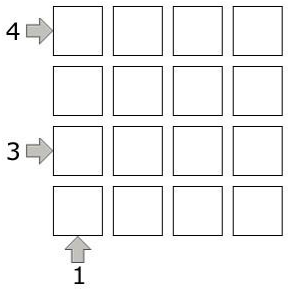
3.4 Estratégia de Pesquisa

A estratégia de etiquetagem implementada, nomeadamente no que diz respeito à ordenação de variáveis e valores no Sicstus Prolog foi o ffc.

Das variáveis com os domínios mais pequenos, a parte mais à esquerda que participa na maioria das restrições é etiquetada primeiro.

4 Visualização da Solução

O programa permite resolver puzzles *Skyscraper* sendo para isso preciso passar-lhe obrigatoriamente um *BoardSize* (Tamanho do lado do quadrado) e uma lista de pistas *Visible* (número de edifícios visíveis quando olhados de determinada direção) no formato [Visible, Direction, RowOrColumn] em que Visible é o numero de edifícios visíveis, Direction pode tomar os valores 'n','e','s','w', que representam *north ,east, south, west,* e indicam de que lado se encontra essa pista e RowOrColumn que representa em que posição da direção considerada a pista se encontra. Por exemplo [3,n,2] significa que são visíveis 3 edifícios na segunda coluna de cima para baixo e [4,e,1] significa que são visíveis 4 edifícios na primeira linha da direita para a esquerda. Pode também ser passada uma lista de listas *Given* (Números já colocados no tabuleiro inicialmente) no formato [Number, Row, Column].




```
run:-
  % Size of the Board side.
  BoardSize=4,
  % List of Given numbers already on the board (Number,Row,Column)
  Given=[],
  % List of Visible buildings (Visible,Direction,RowOrColumn)
  Visible=[[1,s,1],[4,w,1],[3,w,3]],
  statistics(runtime, [T0|_]),
  skyscraper(BoardSize,Given,Visible,Result),
  statistics(runtime, [T1|_]),
  T is T1 - T0,nl,
  printBoardRowValues(Result),
  write('Puzzle solved in: '),write(T),write('ms').

%Solves a skyscraper puzzle size KxK
 skyscraper(K, Given, Visible, Skyscraper):-
  length(Skyscraper, K),
  applyLatinSquareConstraints(Skyscraper,K),
  given(Given,K,Skyscraper),
  visible(Visible, K, Skyscraper),
  flattenList(Skyscraper, SkyscraperFlatten),
  labeling([ffc], SkyscraperFlatten).
```

Fig. 3. Exemplo de Skyscraper 4x4 Difícil e forma como são passados os valores no programa

Para a visualização da solução final recorremos ao predicado *printBoardRowValues(+List)* que escreve no ecrã as sublistas (linhas), separadas por um *new line.* obtendo assim o formato 'normal' do tabuleiro.

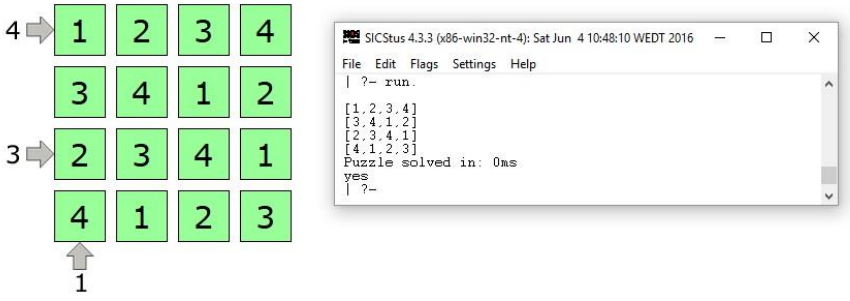


Fig. 4. Visualização (à direita) do resultado de executar o puzzle representado na fig.3

5 Resultados

É possível verificar os resultados da execução do programa recorrendo ao predicado *fd_statistics*. O tempo que o Sicstus Prolog demora a retornar uma solução depende da complexidade do puzzle a ser resolvido. No nosso projeto, podendo ser descrito como um problema de decisão, o tempo necessário para criar uma solução é muito baixo, especialmente para os puzzles mais facéis. As maiores diferenças são ao nível de prunings, created constraints, backtracks e outras estatísticas devolvidas pelo predicado fd_statistics, que serão tanto maiores quando maior a complexidade do puzzle.

Puzzle	Tempo de Execução(ms)	Backtracks	Restrições
4x4 Fácil	0ms	1	352
4x4 Difícil	0ms	2	79
5x5 Fácil	0ms	15	564
5x5 Difícil	23ms	638	272
6x6 Fácil	8ms	52	820
6x6 Difícil	41ms	1099	409
7x7	16ms	44	1175
8x8	16ms	78	1596
9x9	50ms	570	1006

Tabela 1. Tabela de resultados para alguns puzzles atribuídos pelo utilizador.

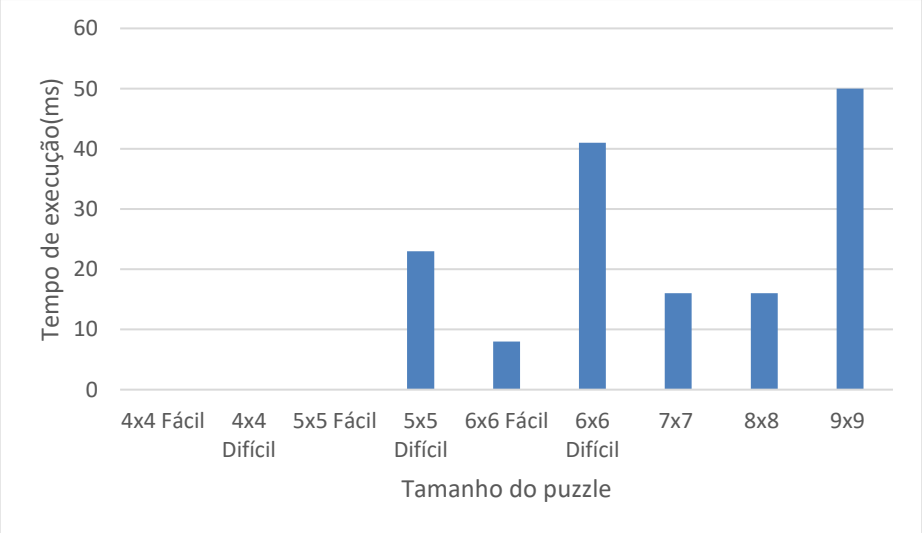


Fig 5. Gráfico com tempos de execução da solução para diferentes complexidades de puzzles Skyscraper.

6 Conclusão

Após a realização deste projeto, o grupo conclui que a linguagem *Prolog*, mais especificamente referindo-se aos seus módulos de resolução com restrições,é bastante ponderosa, permitindo a resolução de uma ampla variedade de questões de decisão e otimização. Depois de entendido o funcionamento por de trás de variáveis de decisão, formas de restringir o domínio de variáveis, a maneira como o labeling trabalha bem como aprender e enquadrar todo o potencial dos predicados existentes na biblioteca 'clpfd', tornou-se mais fácil a resolução do problema proposto e enriqueceu o nosso conhecimento deste paradigma de programação.

Bibliografia

1. Skyscraper techniques, http://www.conceptispuzzles.com/index.aspx?uri=puzzle/skyscrapers/techniques
2. Slides da cadeira relativos a Programação em Lógica com Restrições, https://moodle.up.pt/pluginfile.php/57188/mod_resource/content/7/PLR.pdf
3. Slides da cadeira relativos a PLR com SICStus, https://moodle.up.pt/pluginfile.php/57189/mod_resource/content/6/PLR_SICStus.pdf
4. Skyscraper puzzles and instructions, https://www.puzzlemix.com/Skyscraper
5. BrainBashers, https://www.brainbashers.com/skyscrapers.asp
6. SICStus Prolog User manual, https://sicstus.sics.se/sicstus/docs/latest4/pdf/sicstus.pdf
7. SWI-Prolog documentation, http://www.swi-prolog.org/
8. Stackoverflow, https://stackoverflow.com/questions/tagged/prolog

Anexo 1 – Código Fonte

```
:- use_module(library(clpfd)).
:- use_module(library(lists)).

%---------------------------------------------------------%
%-----------------------SKYSCRAPER------------------------%
%---------------------------------------------------------%

%teste run
run:-
  % Size of the Board side.
  BoardSize=4,
  % List of Given numbers already on the board (Num-
ber,Row,Column)
  Given=[],
  % List of Visible buildings (Visible,Direction,RowOr-
Column)
  Visi-
ble=[[2,n,1],[3,n,2],[1,n,3],[3,n,4],[2,e,1],[3,e,2],[1,e
,3],[2,e,4],[2,s,1],[1,s,2],[4,s,3],[2,s,4],[2,w,1],[1,w,
2],[3,w,3],[2,w,4]],
  statistics(runtime, [T0|_]),
  skyscraper(BoardSize,Given,Visible,Result),
  statistics(runtime, [T1|_]),
  T is T1 - T0,nl,
  printBoardRowValues(Result),
  write('Puzzle solved in: '),write(T),write('ms').

%Solves a skyscraper puzzle size KxK
 skyscraper(K, Given, Visible, Skyscraper):-
  length(Skyscraper, K),
  applyLatinSquareConstraints(Skyscraper,K),
  given(Given,K,Skyscraper),
  visible(Visible, K, Skyscraper),
 flattenList(Skyscraper, SkyscraperFlatten),
  labeling([ffc], SkyscraperFlatten).


%Create the board and apply the latin square constraints
applyLatinSquareConstraints(Skyscraper,K):-
  rows(K, Skyscraper),
  transpose(Skyscraper, SkyscraperT),
  rows(K, SkyscraperT).
```

```
%Insert the given numbers on the board
given([], _, _).
given([Given|Rest], K, Skyscraper):-
  nth0(0,Given,N),
  nth0(1,Given,R),
  nth0(2,Given,C),
  /*length(Row, K),*/
  nth1(R, Skyscraper, Row),
  nth1(C, Row, N),
  given(Rest, K, Skyscraper).

%Apply the visibility contraints(Clues)
visible([], _,  _).
visible([Visible|Rest], K, Skyscraper):-
  nth0(0,Visible,V),
  nth0(1,Visible,Dir),
  nth0(2,Visible,RC),
  ( Dir=n -> columnToList(RC, Skyscraper, List)
  ; Dir=e -> nth1(RC, Skyscraper, List1), reverse(List1,
List)
  ; Dir=s -> columnToList(RC, Skyscraper, List1), re-
verse(List1, List)
  ; Dir=w, nth1(RC, Skyscraper, List)),
  /*length(List, K), domain(List,1, K), all_dis-
tinct(List),*/
  visnum(List, V),
  visible(Rest, K, Skyscraper).

% visnum(L, K): the number of left-visible numbers in L
is K.
visnum([], 0).
visnum([L|Ls], K):-
  visnum(Ls, K, L).
visnum([], 1, _).
visnum([Head|Tail],K, Greatest):-
  (Head #> Greatest) #<=> B,
  B #=> Greatest1 #= Head,
  #\B #=> Greatest1 #= Greatest,
  K #= K1+B,
  visnum(Tail, K1, Greatest1).

% given a list of lists, returns a list containing the
cth element of every sublist
columnToList(_, [], _).
```

```
columnToList(C, [Row|Rest], [X|List]):-
  /*length(Rest, L), length(List, L),*/
  nth1(C, Row, X),
  columnToList(C, Rest, List).

% checks if every row is of length N and contains each
integer from 1 to N exactly once
rows(_, []).
rows(N, [Head|Tail]):-
  length(Head, N),
  domain(Head, 1, N),
  all_distinct(Head),
  rows(N, Tail).

%-------------------------------------------------------%
%---------------SKYSCRAPER UTILITIES--------------------%
%-------------------------------------------------------%

flattenList([], []).
flattenList([H|T], NewList):-
        flattenList(T, Prev),
        pushElementsToList(H, Prev, NewList).

pushElementsToList([], R, R).
pushElementsToList([H|T], Prev, [H|NewList]):-
        pushElementsToList(T, Prev, NewList).

% Print Board
printBoardRowValues([]).
printBoardRowValues([Head | Tail]):-
  write(Head), nl,
  printBoardRowValues(Tail).
```

Anexo 2 – Visualização dos exemplos utilizados para teste

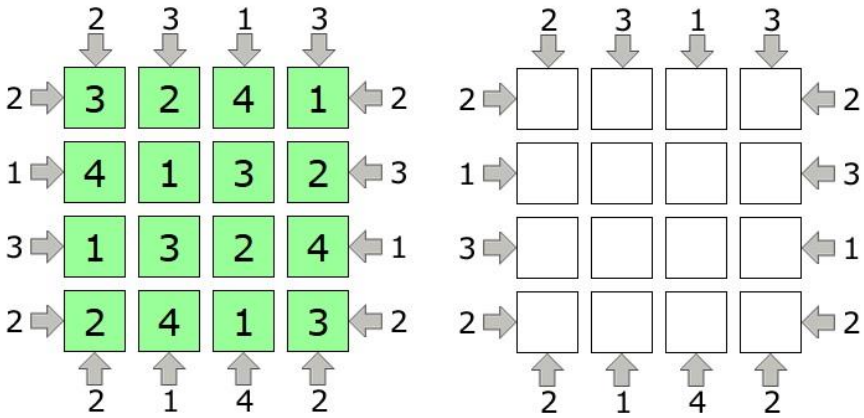


Fig. 6. 4x4 Fácil

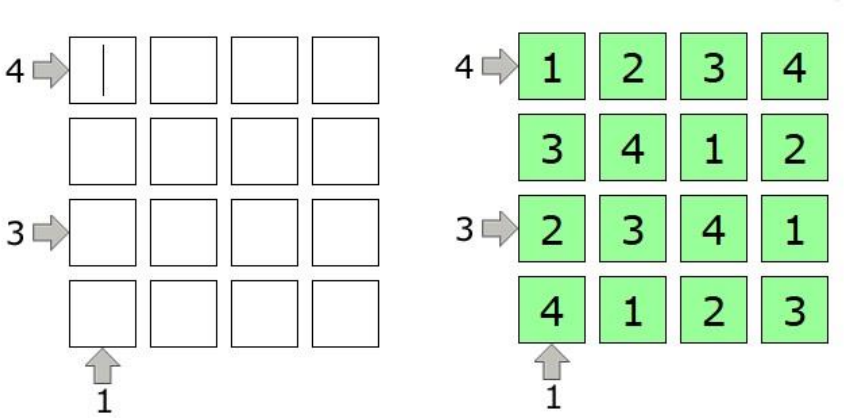


Fig. 7. 4x4 Dificil

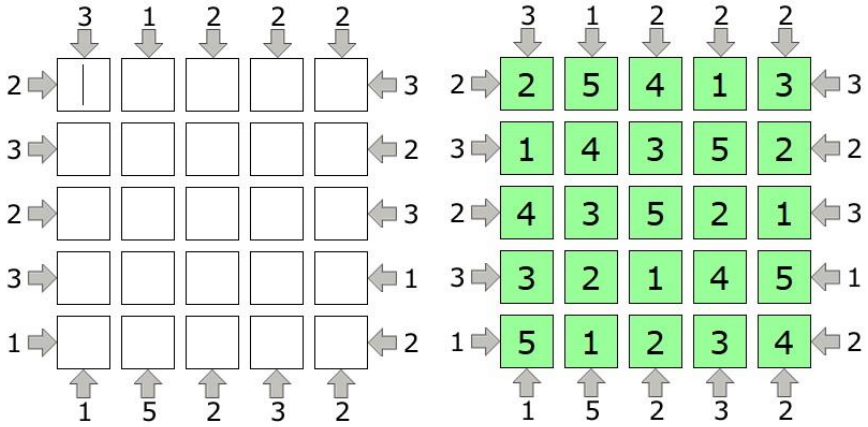


Fig. 8. 5x5 Fácil

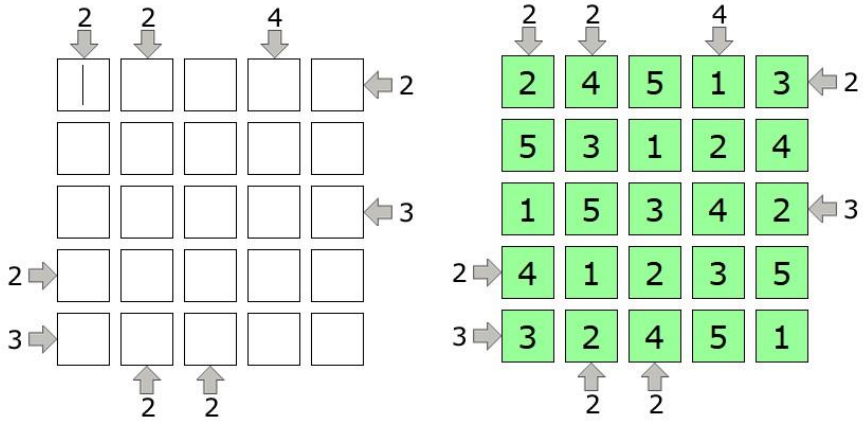


Fig. 9. 5x5 Dificil

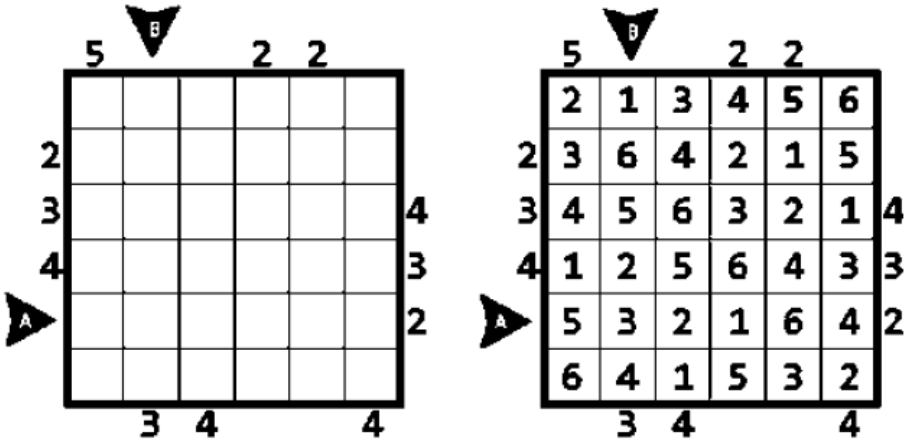


Fig. 10. 6x6 Enunciado

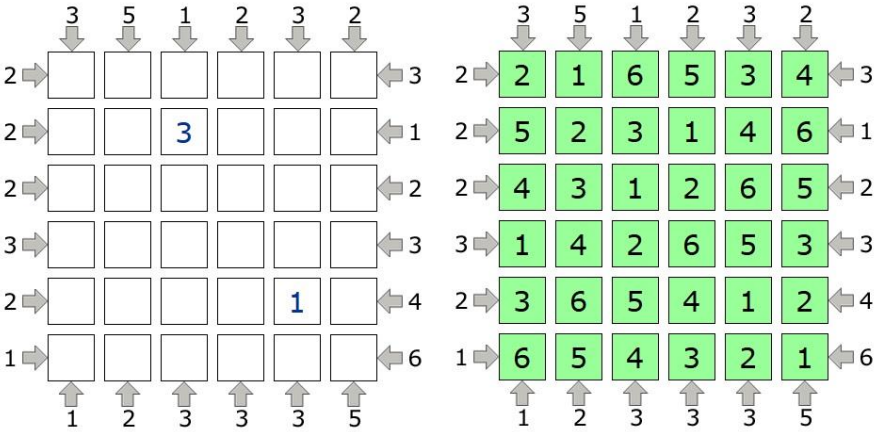


Fig. 11. 6x6 Fácil

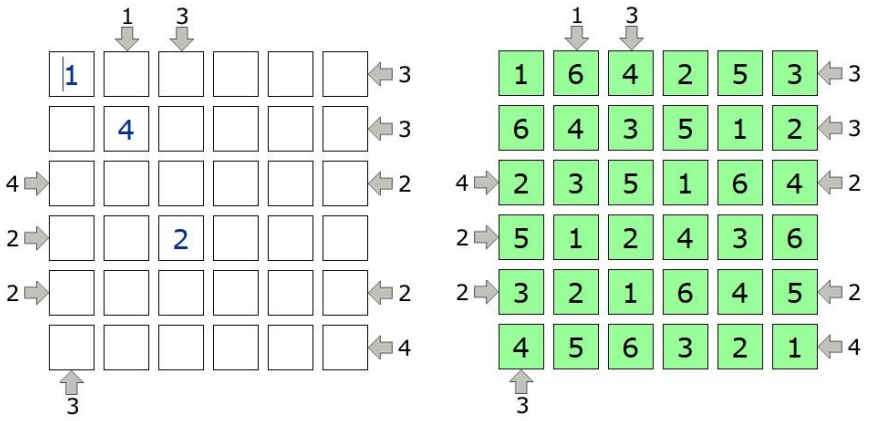


Fig. 12. 6x6 Dificil

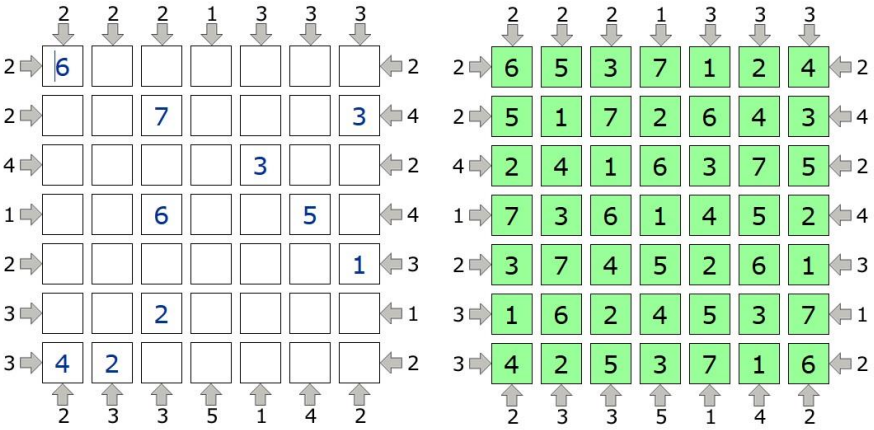


Fig. 13. 7x7

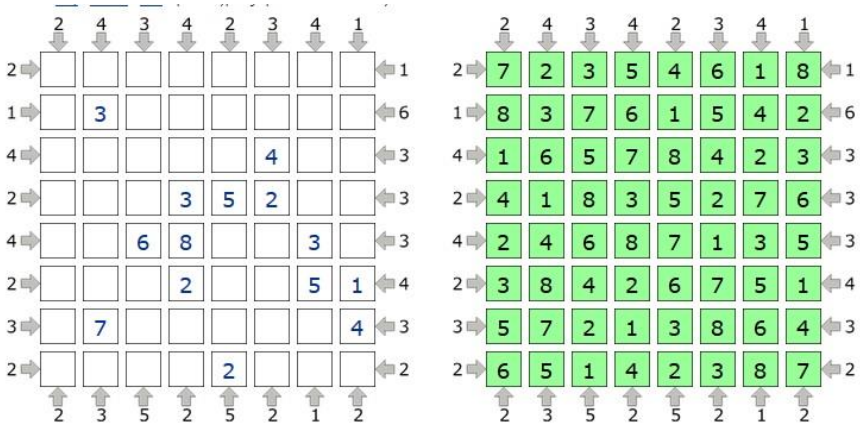


Fig. 14. 8x8

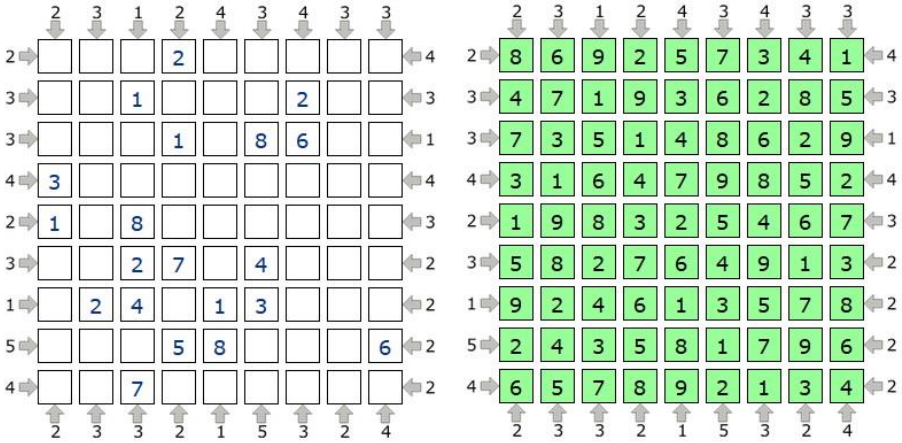


Fig. 15. 9x9