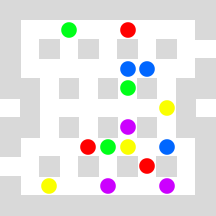
Bird Watching



Disciplina de Programação em Lógica, do 3º Ano (1º Semestre) do Mestrado Integrado em Eng. Informática e Computação da FEUP



Carlos Tiago da Rocha Babo

Felipe de Souza Schmitt

Helder Alexandre Moreira dos Santos

FEUP-PL, Turma 3MIEIC1, Grupo 106

**Resumo/Abstract.**

O trabalho foi realizado no âmbito da disciplina de Programação em Lógica na Faculdade de Engenharia da Universidade do Porto, com o objectivo de aplicar os conhecimentos adquiridos ao longo do semestre, mais propriamente programação de PROLOG com restrições de forma a encontrar uma solução para o puzzle Birdwatching.

Em primeiro lugar foi desenvolvida uma solução que tinha partes onde era aplicado prolog com restrições e outras sem, após verificar que os tempos de execução não eras satisfatórios decidimos desenvolver uma solução que possuisse apenas prolog com restrições.

Neste artigo vamos comparar as diferenças nos seus tempos de execução e avaliar qual das duas soluções seria a mais apropriada para este tipo de problemas.

Introdução

O objectivo principal para este trabalho foi aplicar os nossos conhecimentos adquiridos em Programação em Lógica. Na escolha do tema achamos que este problema era cativante e interessante, sendo estas as nossas principais motivações na resolução do trabalho.

Este artigo é constituido por três principais partes, em primeiro lugar é apresentado a descrição do problema a resolver. De seguida é demonstrado as abordagens utilizadas para a resolução desse mesmo problema e por fim uma analise dos resultados obtidos e a comparação entre abordagens diferentes.

Descrição do problema

O problema Birdwatching consiste num puzzle em que é necessário encontrar um caminho entre a entrada do tabuleiro e a sua saida em que passe por um e apenas um pássaro de cada cor, não podendo realizar o mesmo caminho mais do que uma vez.

Sendo o objectivo realizar uma solução utilizando a linguagem PROLOG com restrições.

Ficheiros de Dados

Escolhemos vários problemas em que consistiam em vários puzzles em que o objectivo era passar por um passáro de cada cor e também escolhemos puzzles em que o objectivo era passar não um, mas dois e três pássaros de cada cor.

O primeiro problema que utilizamos para encontrar a resolução através das restrições que definimos encontra se em anexo no ponto 1, junto com a sua solução.

O segundo problema é de mesma dificuldade que o primeiro, alterando a posição dos pássaros no tabuleiro, no ponto 2 do anexo.

O terceiro problema destaca-se dos outros pois neste caso é necessário encontrar um caminho desde a entrada à saida do tabuleiro que passe por apenas dois pássaros de cada cor e encontra-se no ponto 3 do anexo deste artigo.

O quarto problema e o mais complexo obriga o caminho a passar por três pássaros de cada cor, este caminho encontra se exemplificado no anexo, ponto 4.

Variáveis de Decisão

Neste trabalho existe uma variável de decisão, designada por “Caminho” que indica o caminho percorrido para chegar à saida do tabuleiro. Esta variável de decisão tem um dominio de 1 a CaminhoMax+1, em que a variável CaminhoMax é o numero máximo que o caminho pode ter e é calculada pela subtracção das casas a zero e das casas por onde o caminho não passa às 121 casas totais do tabuleiro, sendo no pior caso 60 o número máximo do caminho e 61 o número de casas por onde o caminho não passa.

Restrições

As restrições aplicadas para a solução deste trabalho foram em primeiro lugar as restrições das paredes do tabuleiro em que estão assinaladas com um 0 no tabuleiro, de seguida a restrição foi aplicar a conectividade 4, ou seja num caminho só era possivel dirigir se para cima, baixo, esquerda ou direita, a proxima restrição aplicada é definir as casas que não são do caminho como paredes.

As restrições rígidas aplicadas foram a casa de entrada, a casa adjacente à entrada, a casa de saida e a casa adjacente à saida.

Enquanto as restrições flexiveis aplicadas foram que as posições por onde é possivel passar num caminho podem assumir o valor da sua ordem caso pertençam ao caminho ou o valor da parede caso não o pertençam.

Função de Avaliação

A solução obtida é avaliada verificando o caminho percorrido com a resolução existente no site da bibliografia, assim como analisando o output e verificando que passa correctamente pelos Nbirds de cada cor, assim como a sua ordem é a correcta.

Estratégia de Pesquisa

Na implementação da etiquetagem (“labeling”) utilizamos uma estratégia que é incluir na sua chamada o min e o up, de forma a que a etiquetagem começasse pela variavél mais à esquerda com o dominio esquerdo mais pequeno, ou seja pela entrada do tabuleiro. Enquanto o up é para o dominio ser explorado em ordem ascendente , desta forma processando primeiro os caminhos mais pequenos, uma vez que a etiquetagem destes é mais rápida.

Visualização da Solução

A visualização da solução em modo de texto é aplicada através dos predicados que se encontram no anexo A ponto 5.

O predicado printBirdsPos imprime apartir da lista dos pássaros que se encontram no caminho a sua posição e a sua ordem no caminho, desta forma ser mais fácil confirmar que a solução obtida é válida.

O segundo predicado da visualização da solução é o printSol que apartir da lista do caminho, dos pássaros e do tabuleiro que verifica se a cabeça da lista que está a processar se é um pássaro, se faz parte do caminho. Caso a cabeça da lista for um pássaro nessa posição é impresso um ‘B’ a simbolizar um pássaro que se encontra no caminho, caso faça parte do caminho é impresso um ‘\*’ para ser mais facil a visualização do caminho no tabuleiro, caso contrário é impresso o tabuleiro de origem.

Resultados

Em primeiro lugar foi desenvolvida uma resolução para o problema que era hibrida, pois incluia prolog sem e com restrições, calculando os caminhos a percorrer através de prolog sem restrições e posteriormente aplicar as restrições para que a solução apenas tivesse um pássaro de cada cor. Após o desenvolvimento desta solução foi verificada que a solução para os problemas em que o caminho teria que passar por apenas um pássaro eram resolvidos entre 850ms a 900ms. Enquanto na solução dos problemas que o caminho teria que passar obrigatóriamente por dois pássaros eram resolvidos em entre 3 minutos e 40 segundos a 3 minutos e 50 segundos.

Uma vez que para a resolução deste segundo problema consideramos que o seu tempo de execução era demasiado procuramos tentar abordar o problema de outra forma e criar uma resolução utilizando apenas prolog com restrições.

Após realizar uma solução para o problema utilizando apenas restrições conseguimos observar através dos resultados que para a resolução de um problema em que o caminho tem que passar por um pássaro o tempo de execução é acerca de 1900ms ~2segundos. Ou seja o cerca de o dobro da solução dada pela resolução hibrida. Mas no caso do problema em que o caminho tem que passar por dois pássaros de cada cor o tempo de execução é consideravelmente menor, sendo acerca de 14 segundos, diminuindo assim o tempo de execução para 6,3% do tempo inicial.

Conclusões e Perspectivas de Desenvolvimento

Uma vez que este trabalho deu nos a oportunidade de comparar duas soluções para o mesmo problema utilizando duas estratégias diferentes é possível tirar conclusões crediveis a diferenciar as duas soluções desenvolvidas.

Ao analisar os resultados podemos concluir que a solução que utiliza apenas prolog com restrições é mais eficaz globalmente do que a solução que é apenas parcialmente com restrições. Isto deve se ao facto de enquanto a solução hibrida encontra uma solução, testa se essa solução passa as restrições e caso falhe vai procurar uma nova solução e repete-se num ciclo até encontrar uma solução que passe as restrições, na solução que utiliza apenas restrições aplica todas as restrições e depois de aplicar as restrições todas, vai procurar uma solução válida, sendo desta forma mais rápida a procura.

Este projecto acabou por se tornar um grande desafio e uma boa forma de aprendizagem, uma vez que abordamos o mesmo problema de duas formas diferentes e fomos capazes de verificar a diferença entre as duas abordagens assim como comparar os seus tempos de execução e eficácias. Assim como também foi possível através deste projecto encontrar uma solução, verificar as suas limitações e posteriormente melhorar essa solução de forma a que o resultado seja mais eficaz. Este projecto também serviu para aplicarmos os nossos conhecimentos em prolog com o objectivo a resolver um problema de decisão.

Bibliografia

<http://www2.stetson.edu/~efriedma/birds/>

<http://www.fi.muni.cz/~hanka/sicstus/doc-3.11/pdf/sicstus.pdf>

Anexos

Anexo A

1. Problema 1

[0,0,0,0,0,0,0,0,0,0,0,

0,1,1,2,1,1,3,1,1,1,0,

0,1,0,1,0,1,0,1,0,1,1,

0,1,1,1,1,1,4,4,1,1,0,

0,0,1,0,1,0,2,0,1,0,0,

0,0,1,1,1,1,1,1,5,0,0,

0,0,1,0,1,0,6,0,1,0,0,

0,1,1,1,3,2,5,1,4,1,0,

1,1,0,1,0,1,0,3,0,1,0,

0,1,5,1,1,6,1,1,6,1,0,

0,0,0,0,0,0,0,0,0,0,0].

Onde a solução seria percorrer o seguinte caminho:

Solucao do problema:

0 0 0 0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 \* \* \* 0

0 1 0 1 0 1 0 \* 0 \* \*

0 1 1 1 1 1 1 B \* 1 0

0 0 1 0 1 0 1 0 \* 0 0

0 0 1 1 \* \* \* \* B 0 0

0 0 1 0 \* 0 1 0 1 0 0

0 \* \* \* B B 1 1 1 1 0

\* \* 0 \* 0 \* 0 1 0 1 0

0 1 1 \* \* B 1 1 1 1 0

0 0 0 0 0 0 0 0 0 0 0

Passaro - Ordem do caminho:

83-11 82-12 41-21 64-18 105-9

1. Problema 2

[0,0,0,0,0,0,0,0,0,0,0,

0,1,1,5,1,1,3,4,1,1,0,

0,1,0,1,0,1,0,1,0,1,1,

0,1,1,3,1,1,2,2,1,1,0,

0,0,1,0,5,0,1,0,1,0,0,

0,0,1,1,4,1,2,6,1,0,0,

0,0,6,0,1,0,1,0,1,0,0,

0,4,1,1,1,1,6,1,1,1,0,

1,1,0,1,0,1,0,1,0,1,0,

0,1,1,1,5,1,3,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0].

E a sua solução:

Solucao do problema:

0 0 0 0 0 0 0 0 0 0 0

0 1 1 B \* \* 1 1 1 1 0

0 1 0 \* 0 \* 0 1 0 \* \*

0 1 \* B 1 \* B 1 \* \* 0

0 0 \* 0 1 0 \* 0 \* 0 0

0 0 \* 1 B \* \* 1 \* 0 0

0 0 B 0 \* 0 1 0 \* 0 0

0 \* \* 1 \* \* \* \* \* 1 0

\* \* 0 1 0 1 0 1 0 1 0

0 1 1 1 1 1 1 1 1 1 0

0 0 0 0 0 0 0 0 0 0 0

Passaro - Ordem do caminho:

40-16 37-9 60-20 15-11 69-5

1. Problema 3

[0,0,0,0,0,0,0,0,0,0,0,

0,6,6,1,4,3,1,2,1,1,0,

0,1,0,4,0,4,0,1,0,1,1,

0,1,1,1,2,1,2,5,5,5,0,

0,0,1,0,1,0,1,0,1,0,0,

0,0,1,1,1,3,4,1,1,0,0,

0,0,1,0,6,0,1,0,1,0,0,

0,1,1,1,1,6,1,1,1,1,0,

1,1,0,1,0,3,0,1,0,1,0,

0,1,1,1,1,1,3,5,1,2,0,

0,0,0,0,0,0,0,0,0,0,0].

Sendo a sua solução o seguinte caminho:

Solucao do problema:

0 0 0 0 0 0 0 0 0 0 0

0 1 1 1 1 B \* B \* \* 0

0 1 0 1 0 B 0 1 0 \* \*

0 1 1 1 1 \* B B B 1 0

0 0 1 0 1 0 1 0 \* 0 0

0 0 \* \* \* 1 B \* \* 0 0

0 0 \* 0 B 0 \* 0 1 0 0

0 \* \* \* \* B \* 1 1 1 0

\* \* 0 \* 0 B 0 1 0 1 0

0 1 1 \* \* \* 1 1 1 1 0

0 0 0 0 0 0 0 0 0 0 0

Passaro - Ordem do caminho:

19-31 40-26 17-29 94-16 28-28 62-20 41-25 42-24 71-9 83-17

1. Problema 4

[0,0,0,0,0,0,0,0,0,0,0,

0,6,6,6,4,5,1,1,1,1,0,

0,1,0,5,0,6,0,2,0,1,1,

0,1,3,1,2,1,5,1,2,1,0,

0,0,4,0,1,0,2,0,4,0,0,

0,0,1,1,4,1,6,1,5,0,0,

0,0,1,0,1,0,1,0,1,0,0,

0,1,3,1,3,1,3,1,1,1,0,

1,1,0,1,0,1,0,1,0,1,0,

0,1,4,1,1,1,5,3,1,2,0,

0,0,0,0,0,0,0,0,0,0,0].

Solucao do problema:

0 0 0 0 0 0 0 0 0 0 0

0 B B B B B \* \* \* \* 0

0 \* 0 1 0 1 0 1 0 \* \*

0 \* B \* B \* B \* B 1 0

0 0 1 0 1 0 1 0 B 0 0

0 0 1 1 1 1 1 1 B 0 0

0 0 1 0 1 0 1 0 \* 0 0

0 1 1 1 1 \* B \* \* \* 0

\* \* 0 1 0 \* 0 \* 0 \* 0

0 \* B \* \* \* 1 B \* B 0

0 0 0 0 0 0 0 0 0 0 0

Passaro - Ordem do caminho:

38-26 42-22 109-15 36-28 84-10 107-13 16-34 53-21 102-4 17-35 40-24 64-20 13-31 14-32 15-33

1. Predicados correspondentes ao output da solução.

printSol(Caminho,\_, Pos,\_,\_,\_):-length(Caminho, Pos).

printSol(Caminho,SizeC, Pos, MaxSizeCaminho,Birds, Tab):-

sublist(Caminho, Part, Pos, SizeC, Next),

sublist(Tab, Part2, Pos, SizeC, Next),

printLinhaSol(Part, MaxSizeCaminho, Birds, Part2),nl,

length(Caminho, S),

Next2 is S-Next,

printSol(Caminho, SizeC, Next2, MaxSizeCaminho, Birds, Tab).

printLinhaSol([],\_,\_,\_).

printLinhaSol([H|T], H, Birds, [H2|T2]):-

write(H2),write(' '),

printLinhaSol(T, H, Birds, T2), !.

printLinhaSol([H|T], S, Birds, [\_|T2]):-

member(H, Birds),

write('B '),

printLinhaSol(T, S, Birds, T2), !.

printLinhaSol([\_|T], S, Birds,[\_|T2]):-

write('\* '),

printLinhaSol(T, S, Birds, T2), !.

Anexo B

Código Fonte:

% RESOLUCAO DE PUZZLES BIRDWATCHING COM RESTRICOES E COM UMA SOLUCAO HIBRIDA

% AUTORES: TIAGO BABO, HELDER MOREIRA E FELIPE SCHMITT

% IMPORTACAO DE MODULOS

:-use\_module(library(random)).

:-use\_module(library(clpfd)).

:-use\_module(library(lists)).

% ---UTILITARIOS---

% Verde = 2 = A

% Vermelho = 3 = B

% Azul = 4 = C

% Amarelo = 5 = D

% Roxo = 6 = E

% REMOVE UMA POSICAO DE UMA LISTA

remove\_at(X,[X|Xs],1,Xs).

remove\_at(X,[Y|Xs],K,[Y|Ys]) :- K > 1,

K1 is K - 1, remove\_at(X,Xs,K1,Ys).

% INSERE ELEMENTO NUMA CERTA POSICAO DE UMA LISTA

insert\_at(X,L,K,R) :- remove\_at(X,R,K,L).

% OBTER POSICAO DA LISTA CORRIDA, DADA UMA COLUNA E UMA LINHA

getPos(X,Y, Pos) :-

Pos is Y\*11+X.

% DADO UM TABULEIRO, RETIRA OS PASSAROS DA MESMA COR E AGRUPA-OS EM LISTAS SEPARADAS

parser([],[],[],[],[],[],\_).

parser([2|R],[Indice|RA],BOut,COut,DOut,EOut, Indice) :-

Indice1 is Indice +1,

parser(R,RA,BOut,COut,DOut,EOut, Indice1) ,!.

parser([3|R],AOut,[Indice|RB],COut,DOut,EOut, Indice) :-

Indice1 is Indice +1,

parser(R,AOut,RB,COut,DOut,EOut, Indice1) ,!.

parser([4|R],AOut,BOut,[Indice|RC],DOut,EOut, Indice) :-

Indice1 is Indice +1,

parser(R,AOut,BOut,RC,DOut,EOut, Indice1) ,!.

parser([5|R],AOut,BOut,COut,[Indice|RD],EOut, Indice) :-

Indice1 is Indice +1,

parser(R,AOut,BOut,COut,RD,EOut, Indice1) ,!.

parser([6|R],AOut,BOut,COut,DOut,[Indice|RE], Indice) :-

Indice1 is Indice +1,

parser(R,AOut,BOut,COut,DOut,RE, Indice1) ,!.

parser([\_|R],AOut,BOut,COut,DOut,EOut, Indice) :-

Indice1 is Indice +1,

parser(R,AOut,BOut,COut,DOut,EOut, Indice1) ,!.

% --- \UTILITARIOS ---

% TABULEIROS DE JOGO

tabuleiro(T) :-

T = [0,0,0,0,0,0,0,0,0,0,0,

0,1,1,2,1,1,3,1,1,1,0,

0,1,0,1,0,1,0,1,0,1,1,

0,1,1,1,1,1,4,4,1,1,0,

0,0,1,0,1,0,2,0,1,0,0,

0,0,1,1,1,1,1,1,5,0,0,

0,0,1,0,1,0,6,0,1,0,0,

0,1,1,1,3,2,5,1,4,1,0,

1,1,0,1,0,1,0,3,0,1,0,

0,1,5,1,1,6,1,1,6,1,0,

0,0,0,0,0,0,0,0,0,0,0].

tabuleiro\_2(T) :-

T = [0,0,0,0,0,0,0,0,0,0,0,

0,1,1,5,1,1,3,4,1,1,0,

0,1,0,1,0,1,0,1,0,1,1,

0,1,1,3,1,1,2,2,1,1,0,

0,0,1,0,5,0,1,0,1,0,0,

0,0,1,1,4,1,0,6,1,0,0,

0,0,6,0,1,0,1,0,1,0,0,

0,1,1,1,1,1,1,1,1,1,0,

1,1,0,1,0,1,0,1,0,1,0,

0,1,1,2,3,4,5,6,1,1,0,

0,0,0,0,0,0,0,0,0,0,0].

tabuleiroTwoBirds(T) :-

T = [0,0,0,0,0,0,0,0,0,0,0,

0,6,6,1,4,3,1,2,1,1,0,

0,1,0,4,0,4,0,1,0,1,1,

0,1,1,1,2,1,2,5,5,5,0,

0,0,1,0,1,0,1,0,1,0,0,

0,0,1,1,1,3,4,1,1,0,0,

0,0,1,0,6,0,1,0,1,0,0,

0,1,1,1,1,6,1,1,1,1,0,

1,1,0,1,0,3,0,1,0,1,0,

0,1,1,1,1,1,3,5,1,2,0,

0,0,0,0,0,0,0,0,0,0,0].

tabuleiroThreeBirds(T) :-

T = [0,0,0,0,0,0,0,0,0,0,0,

0,6,6,6,4,5,1,1,1,1,0,

0,1,0,5,0,6,0,2,0,1,1,

0,1,3,1,2,1,5,1,2,1,0,

0,0,4,0,1,0,2,0,4,0,0,

0,0,1,1,4,1,6,1,5,0,0,

0,0,1,0,1,0,1,0,1,0,0,

0,1,3,1,3,1,3,1,1,1,0,

1,1,0,1,0,1,0,1,0,1,0,

0,1,4,1,1,1,5,3,1,2,0,

0,0,0,0,0,0,0,0,0,0,0].

% MASCARA DO TABULEIRO

tabuleiroMask(T) :-

T = [0,0,0,0,0,0,0,0,0,0,0,

0,1,1,1,1,1,1,1,1,1,0,

0,1,0,1,0,1,0,1,0,1,1,

0,1,1,1,1,1,1,1,1,1,0,

0,0,1,0,1,0,1,0,1,0,0,

0,0,1,1,1,1,1,1,1,0,0,

0,0,1,0,1,0,1,0,1,0,0,

0,1,1,1,1,1,1,1,1,1,0,

1,1,0,1,0,1,0,1,0,1,0,

0,1,1,1,1,1,1,1,1,1,0,

0,0,0,0,0,0,0,0,0,0,0].

% SOLUCAO COM TOTALMENTE COM RESTRICOES

% COLOCA AS RESTRICOES NAS VARIAS PECAS DO TABULEIRO

processaCaminho2(\_,112,\_,\_,\_,\_,\_,\_):- !.

% SE A POSICAO CORRESPONDE A UMA PAREDE NO TABULEIRO, LOGO A ORDEM E' ZERO, E NAO E' NECESSARIO RESTRINGIR (FEITO ANTERIORMENTE)

processaCaminho2(Caminho, PosAct, Posicoes, PecasP, Tab, Fim, Inicio, SizeCaminho):-

element(PosAct, Tab, X),

X = 0,

PosAct2 is PosAct+1,

processaCaminho2(Caminho, PosAct2, Posicoes, PecasP, Tab, Fim, Inicio,SizeCaminho), !.

% PARA A POSICAO DE ENTRADA E SAIDA NAO E' NECESSARIO COLOCAR RESTRICOES

processaCaminho2(Caminho, PosAct, Posicoes, PecasP, Tab, Fim, Inicio, SizeCaminho):-

Inicio2 is Inicio+1,

Fim2 is Fim+1,

Restricoes = [Inicio,Fim, Inicio2, Fim2],

member(PosAct, Restricoes),

PosAct2 is PosAct+1,

processaCaminho2(Caminho, PosAct2, Posicoes, PecasP, Tab, Fim, Inicio, SizeCaminho), !.

% PARA PECAS QUE NAO SAO PAREDE, ENTRADA OU SAIDA

processaCaminho2(Caminho, PosAct, Posicoes, PecasP, Tab, Fim, Inicio, SizeCaminho):-

Redondeza = [A1,B1,C1,D1],

A is PosAct-1,

B is PosAct+1,

C is PosAct-11,

D is PosAct+11,

element(A, Caminho, A1),

element(B, Caminho, B1),

element(C, Caminho, C1),

element(D, Caminho, D1),

element(PosAct, Caminho, X),

element(\_, Redondeza, ValEntrada),

element(\_, Redondeza, ValSaida),

(X #> 0 #=> ValEntrada #= X-1 #/\ ValSaida #= X+1) #\/ X #= SizeCaminho,

isBird(PosAct, Caminho, Posicoes, PecasP),

PosAct2 is PosAct+1,

processaCaminho2(Caminho, PosAct2, Posicoes, PecasP, Tab, Fim, Inicio, SizeCaminho).

isBird(PosAct, Caminho, Posicoes, PecasP):-

element(PosBird, Posicoes, PosAct),

element(PosAct, Caminho, Val),

element(PosBird, PecasP, Val2),

Val #= Val2.

isBird(\_,\_,\_,\_).

% GARANTE QUE O CAMINHO PASSA POR UM PASSARO DE CADA

garanteUm([]).

garanteUm([A,B,C|P]):-

Ordens = [A,B,C],

count(61, Ordens, #=, X),

X #= 2,

garanteUm(P).

% GARANTE QUE O CAMINHO PASSA POR DOIS PASSAROS DE CADA

garanteDois([], \_).

garanteDois([A,B,C,D|P], SizeCaminho):-

Ordens = [A,B,C,D],

count(SizeCaminho, Ordens, #=, X),

X #= 2,

garanteDois(P, SizeCaminho).

garanteXBirds(Birds,\_,\_,\_,Pos):-length(Birds, Pos), !.

garanteXBirds(Birds, SizeCaminho, XBirds, NBirds, Pos):-

sublist(Birds, Part, Pos, NBirds, Next),

W is NBirds-XBirds,

count(SizeCaminho, Part, #=, X),

X #= W,

length(Birds, S),

Next2 is S-Next,

garanteXBirds(Birds, SizeCaminho, XBirds, NBirds, Next2).

% RESTRINGE AS CASAS PAREDE AO VALOR ZERO

casasZero([],[], \_).

casasZero([H2|T2], [H|T], SizeCaminho):-

H #=0 #=> H2 #=SizeCaminho,

casasZero(T2, T, SizeCaminho).

% ROTINA RESPONSAVEL POR ENCONTRAR O CAMINHO E FAZER LABELING

xBirdsComRestricoes(Caminho, Birds, Tabuleiro, SizeC, XBirds):-

write('Tabuleiro a processar: '),

nl,

printTabuleiro(Tabuleiro, 1, SizeC),

NBirds is XBirds+2,

tabuleiroMask(T),

parser(Tabuleiro,AOut,BOut,COut,DOut,EOut,1),

append(AOut, BOut, P),

append(P, COut, P2),

append(P2, DOut, P3),

append(P3, EOut, P4),

length(Tabuleiro, SizeTab),

length(Caminho, SizeTab),

length(P4, SizeBirds),

length(Birds, SizeBirds),

SizeCaminho is (SizeC)\*6,

domain(Caminho, 1, SizeCaminho),

domain(Birds, 1, SizeCaminho),

Entrada is SizeC\*SizeC-3\*SizeC+1,

Entrada2 is Entrada+1,

element(Entrada, Caminho, 1),

element(Entrada2, Caminho, 2),

count(SizeCaminho, Caminho, #=, Sz),

Sz2 #= SizeTab-Sz #/\ Sz3 #= Sz2-1,

Saida is SizeC\*3,

Saida2 is Saida-1,

element(Saida2, Caminho, Sz3),

element(Saida, Caminho, Sz2),

casasZero(Caminho, T, SizeCaminho),

garanteXBirds(Birds, SizeCaminho, XBirds, NBirds, 0),

processaCaminho2(Caminho, 13, P4, Birds, T, 33, 89, SizeCaminho),

append(Birds, Caminho, List),

write('Labeling...'),nl,

!,

labeling([min,up], List),

write('Solucao do problema:'),nl,

printSol(Caminho, SizeC, 0, SizeCaminho, Birds, T),

write('Passaro - Ordem do caminho: '),nl,

printBirdsPos(Birds, P4, SizeCaminho).

printBirdsPos([],\_,\_).

printBirdsPos([H|T], [H2|T2], Max):-

H \= Max,

write(H2-H), write(' '),

printBirdsPos(T, T2, Max), !.

printBirdsPos([\_|T], [\_|T2], Max):-

printBirdsPos(T, T2, Max).

printSol(Caminho, \_, Pos, \_, \_,\_):-length(Caminho, Pos).

printSol(Caminho, SizeC, Pos, MaxSizeCaminho, Birds, Tab):-

sublist(Caminho, Part, Pos, SizeC, Next),

sublist(Tab, Part2, Pos, SizeC, Next),

printLinhaSol(Part, MaxSizeCaminho, Birds, Part2),nl,

length(Caminho, S),

Next2 is S-Next,

printSol(Caminho, SizeC, Next2, MaxSizeCaminho, Birds, Tab).

printLinhaSol([], \_, \_, \_).

printLinhaSol([H|T], H, Birds, [H2|T2]):-

write(H2),write(' '),

printLinhaSol(T, H, Birds, T2), !.

printLinhaSol([H|T], S, Birds, [\_|T2]):-

member(H, Birds),

write('B '),

printLinhaSol(T, S, Birds, T2), !.

printLinhaSol([\_|T], S, Birds,[\_|T2]):-

write('\* '),

printLinhaSol(T, S, Birds, T2), !.

printTabuleiro([],\_,\_):- nl.

printTabuleiro([H|T], Num, Size):-

Size2 is Size+1,

Num == Size2,

nl,

printTabuleiro([H|T], 1, Size).

printTabuleiro([H|T], Num, Size):-

Num2 is Num+1,

write(H),

write(' '),

printTabuleiro(T,Num2, Size).

testeXBirds:-

%tabuleiro(Tabuleiro),

%tabuleiroTwoBirds(Tabuleiro),

tabuleiroThreeBirds(Tabuleiro),

write('Resolucao de puzzles de Birdwatching'),nl,

xBirdsComRestricoes(\_, \_, Tabuleiro, 11, 3).

% SOLUCAO HIBRIDA

oneBird(PosicoesEscolhidas) :-

tabuleiro(T),

Caminho = [A,B,C,D,E],

Posicoes = [P1,P2,P3,P4,P5],

PosicoesEscolhidas = [X1,X2,X3,X4,X5],

all\_distinct(PosicoesEscolhidas),

parser(T,AOut,BOut,COut,DOut,EOut,1),

domain(Caminho,1,5),

element(A,AOut,P1),

element(B,BOut,P2),

element(C,COut,P3),

element(D,DOut,P4),

element(E,EOut,P5),

member(X1, Posicoes),

existeCaminho(89, X1, T, T1),

member(X2, Posicoes),

existeCaminho(X1, X2, T1, T2),

member(X3, Posicoes),

existeCaminho(X2, X3, T2, T3),

member(X4, Posicoes),

existeCaminho(X3, X4, T3, T4),

member(X5, Posicoes),

existeCaminho(X4, X5, T4, T5),

existeCaminho(X5,33,T5,\_).

twoBirds(PosicoesEscolhidas) :-

tabuleiroTwoBirds(T),

Posicoes = [P1,P2,P3,P4,P5,P6,P7,P8,P9,P10],

PosicoesEscolhidas = [X1,X2,X3,X4,X5,X6,X7,X8,X9,X10],

all\_distinct(PosicoesEscolhidas),

all\_distinct(Posicoes),

parser(T,A,B,C,D,E,1),

element(\_,A,P1),

element(\_,A,P2),

element(\_,B,P3),

element(\_,B,P4),

element(\_,C,P5),

element(\_,C,P6),

element(\_,D,P7),

element(\_,D,P8),

element(\_,E,P9),

element(\_,E,P10),

member(X1, Posicoes),

existeCaminho(89, X1, T, T1),

member(X2, Posicoes),

existeCaminho(X1, X2, T1, T2),

member(X3, Posicoes),

existeCaminho(X2, X3, T2, T3),

member(X4, Posicoes),

existeCaminho(X3, X4, T3, T4),

member(X5, Posicoes),

existeCaminho(X4, X5, T4, T5),

member(X6, Posicoes),

existeCaminho(X5, X6, T5, T6),

member(X7, Posicoes),

existeCaminho(X6, X7, T6, T7),

member(X8, Posicoes),

existeCaminho(X7, X8, T7, T8),

member(X9, Posicoes),

existeCaminho(X8, X9, T8, T9),

member(X10, Posicoes),

existeCaminho(X9, X10, T9, T10),

existeCaminho(X10,33,T10,\_).

adjacente(Inicial,Final, Tabuleiro):- Inicial >= 0, Inicial < 121, Final is Inicial-1, nth1(Final, Tabuleiro,1).

adjacente(Inicial,Final, Tabuleiro):- Inicial >= 0, Inicial < 121, Final is Inicial+1, nth1(Final, Tabuleiro,1).

adjacente(Inicial,Final, Tabuleiro):- Inicial =< 110, Final is Inicial+11, nth1(Final, Tabuleiro,1) .

adjacente(Inicial,Final, Tabuleiro):- Inicial >= 11, Final is Inicial-11, nth1(Final, Tabuleiro,1).

adjacente2(Inicial,Final, Tabuleiro):- Inicial >= 0, Inicial < 121, Final is Inicial-1, \+ nth1(Final, Tabuleiro,0).

adjacente2(Inicial,Final, Tabuleiro):- Inicial >= 0, Inicial < 121, Final is Inicial+1, \+ nth1(Final, Tabuleiro,0).

adjacente2(Inicial,Final, Tabuleiro):- Inicial =< 110, Final is Inicial+11, \+ nth1(Final, Tabuleiro,0) .

adjacente2(Inicial,Final, Tabuleiro):- Inicial >= 11, Final is Inicial-11, \+ nth1(Final, Tabuleiro,0).

existeCaminho(\_,Final,T, \_) :-

nth1(Final,T,0),!, fail.

existeCaminho(Inicial,Final, T, TNovo2) :-

adjacente2(Inicial,Final,T),

remove\_at(\_,T,Inicial,TNovo),

insert\_at(0,TNovo,Inicial, TNovo2).

existeCaminho(Inicial,Final,T, TRet) :-

adjacente(Inicial,P,T),

Pos is Inicial,

remove\_at(\_,T,Pos,TNovo),

insert\_at(0,TNovo,Pos, TNovo2),

existeCaminho(P,Final,TNovo2, TRet).

escolhePosicoes(Pecas,N,L,C):-

generatePuzzle(T, L, C),

length(Pecas,N),

findall(X-Y,verificaPeca(T, X,Y, 1),Sols),

escolhePosicoesAux(Pecas, Sols).

choose([], []).

choose(List, Elt) :-

length(List, Length),

random(0, Length, Index),

nth0(Index, List, Elt).

escolhePosicoesAux([],\_).

escolhePosicoesAux([H|T], PecasLivres):-

choose(PecasLivres, X),

member(X, [H]),

delete(PecasLivres, X, PecasLivresNovo),

escolhePosicoesAux(T, PecasLivresNovo).

verificaPeca(T,X,Y,Jogador) :- verificaPecaAux(T,X,Y,Jogador,1).

verificaPecaAux([T|\_],X,Y,Jogador,Y) :-

verificaPecaLinha(T,X,Jogador, 1).

verificaPecaAux([\_|R],X,Y,Jogador,Linha) :-

Linha2 is Linha+1,

verificaPecaAux(R,X,Y,Jogador,Linha2).

verificaPecaLinha([Jogador|\_], X, Jogador, X).

verificaPecaLinha([\_|R], X, Jogador, Coluna) :-

N1 is Coluna+1,

verificaPecaLinha(R, X, Jogador, N1).

limite([], 0).

limite([0|T], M):-

M > 0,

M1 is M-1,

limite(T, M1).

generatePuzzle([H|T], \_, M):-

limite(H, M),

generatePuzzle2(T, M, M, M).

linhaBranca([0|T], M):-

M1 is M-1,

linhaBranca2(T, M1).

linhaBranca2([0], 1).

linhaBranca2([1|T], M):-

M > 1,

M1 is M-1,

linhaBranca2(T, M1).

generateExit([1], 1).

generateExit([1|T], M):-

M > 1,

Div is M mod 2,

Div = 0,

M1 is M-1,

generateExit(T, M1).

generateExit([0|T], M):-

M > 1,

Div is M mod 2,

Div = 1,

M1 is M-1,

generateExit(T, M1).

generateEntrance([], 0).

generateEntrance([1|T], M):-

M > 0,

Div is M mod 2,

Div = 0,

M1 is M-1,

generateEntrance(T, M1).

generateEntrance([0|T], M):-

M > 0,

Div is M mod 2,

Div = 1,

M1 is M-1,

generateEntrance(T, M1).

generatePuzzle2([T], 1, M, \_):-limite(T, M).

generatePuzzle2([[1,1|H]|T], N, M, Norig):-

N =3,

M2 is M-2,

generateEntrance(H, M2),

N1 is N-1,

generatePuzzle2(T, N1, M, Norig).

generatePuzzle2([H|T], N, M, Norig):-

N =:= Norig-1,

generateExit(H, M),

N1 is N-1,

generatePuzzle2(T, N1, M, Norig).

generatePuzzle2([H|T], N, M, Norig):-

N > 1, N \== 3,N =\= Norig-1,

Div is N mod 2,

Div = 0,

linhaBranca(H, M),

N1 is N-1,

generatePuzzle2(T, N1, M, Norig).

generatePuzzle2([H|T], N, M, Norig):-

N > 1, N \== 3,N =\= Norig-1,

Div is N mod 2,

Div = 1,

generateLine(H, M),

N1 is N-1,

generatePuzzle2(T, N1, M, Norig).

generateLine([], 0).

generateLine([1|T], M):-

M > 0,

Div is M mod 2,

Div = 0,

M1 is M-1,

generateLine(T, M1).

generateLine([0|T], M):-

M > 0,

Div is M mod 2,

Div = 1,

M1 is M-1,

generateLine(T, M1).

concatenate([], L, L).

concatenate([X|L1], L2, [X|L3]) :-

concatenate(L1, L2, L3).

tabToList(T,Lista):-

tabToList2(T, Lista, \_).

tabToList2([], F, [F|\_]).

tabToList2([H|T], Lista, [H3|T3]):-

concatenate(H3, H, H4),

tabToList2(T, Lista, [H4|T3]).