Grupo:

Gabriel da Silva Cardoso - 20100524

Julio Gonçalves Ramos - 19203165

Links:

Repositório Git com o código: https://github.com/Cardoz-0/Suguru-Solver/tree/main/Prolog

Apresentação: https://youtu.be/Zy75h_eb5sQ

SUGURU SOLVER EM PROLOG

O problema

Suguru é um quebra cabeça onde o tabuleiro é dividido em grupos ou regiões que devem ser preenchidos com números de acordo com as regras do jogo.

Existem duas regras para como os números podem ser postos no tabuleiro, que é onde a complexidade do puzzle é criada: 1. um número não pode ser posto nas oito casas em torno de um igual. 2. as regiões devem ser completadas obedecendo a regra acima, com números de 1 a N, onde N é o tamanho da região.

Solução

"Parte da dificuldade de realizar um algoritmo para o resolver suguru é que vários caminhos para a solução são sem saída e é preciso checar por esses casos e voltar o quanto for necessário para resolver o problema.

A solução elaborada para esse trabalho foi um tanto quanto diferente dos trabalhos passados. Ao invés de descrevermos um algoritmo para o backtracking da solução, nós precisamos descrever restrições para o backtracking que o próprio Prolog já realiza, como se fossem as regras do jogo.

Houveram inumeras dificuldades, detalhadas abaixo.

Entrada e saída

A entrada dos tabuleiros de jogo é feita diretamente no código. O armazenamento dos tabuleiros é feito manualmente pelo usuário, armazenados como matrizes (lista de listas) em variáveis a serem usadas no método chamado pelo main. Espaços vazios são representados pelo caracter _.

Também é necessário uma matriz para especificar as regiões do tabuleiro, também inserida pelo usuário. Cada região é representada por um grupo de elementos com o mesmo valor adjacentes um ao outro.

Também fizemos uma matriz de limites, que detalha qual o valor máximo que cada quadrado pode ter de acordo com o tamanho de sua região. Ela também deve ser inserida manualmente.

```
tabuleiro([[_,_,_,3,_,_,2,_],
        [4,_,_,_,_,_,_,_],
        [_,2,_,_,_,_,_,_],
        [_,1,5,_,_,1,5,_],
        [_,2,_,_,_,_,_,_],
        [_,_,_,_,4,_,_,4],
        [_,_,_,_,_,_,3,_,_],
        [_,5,_,_,_,5,_,_]]).


regiao([[0,0,0,3,4,4,5,5],
        [0,2,2,3,4,4,5,5],
        [2,2,3,3,4,6,6,5],
        [2,7,7,3,8,8,6,6],
        [10,10,7,7,8,9,9,6],
        [11,10,10,7,8,13,9,9],
        [11,11,10,12,8,13,13,9],
        [11,11,12,12,12,12,13,13]]).

limites([[4,4,4,5,5,5,5,5],
    [4,5,5,5,5,5,5,5],
    [5,5,5,5,5,5,5,5],
    [5,5,5,5,5,5,5,5],
    [5,5,5,5,5,5,5,5],
    [5,5,5,5,5,5,5,5],
    [5,5,5,5,5,5,5,5]]).
```

Para compilar basta rodar:

```
["suguru_comentado_final.pl"].
```

Para executar (A, B, C podem ser substituidos por qualquer nome de variável):

```
tabuleiro(A), regiao(B), limites(C), solucao(A,B,C).
```

Após ser executado o programa printa a matriz do tabuleiro resolvido:

```
[1,3,2,3,2,3,2,3]
[4,5,1,5,1,5,4,5]
[3,2,4,2,4,2,3,1]
[4,1,5,1,3,1,5,4]
[3,2,4,2,5,2,3,1]
[4,1,5,3,4,1,5,4]
[2,3,4,1,2,3,2,1]
[1,5,2,3,4,5,4,5]
```

Implementação

Foram utilizadas restrições para guiar o comportamento de backtracking do Prolog à solução adequada.

Passos da execução: - Transformar as matrizes (listas de listas) em uma só grande lista para cada; - Fazer uma lista com os índices de todas as regiões para checagem; - Checar se o valor testado está na região correta (checkIfInRegion); - Verificar resultados do teste anterior (validateNumber); - Checar se os números são todos diferentes; - Checar se o valor colocado na posição não excede o máximo possível especificado na matriz de limites (checkIfInDomain); - Procurar por repetições na matriz solução encontrada (checkIfDifferentLine e checkIfDifferentMatrix).

A última função (line) apresentada técnicamente checa números em blocos de 2x2 elementos, mas a função checkIfDiferentMatrix passa duas linhas da matriz de cada vez para ela, fazendo assim uma checagem 3x3 já que a linha do meio tem overlap.

Separação de tarefas

As tarefas do trabalho foram separadas entre os integrantes do grupo: Os dois integrantes ficaram a cargo de pesquisar sobre a implementação utilizando restrições e a sua implementação, e um deles montou o relatório e realizou as tarefas relacionadas a edição de vídeo.

Dificuldades

Tivemos bastante dificuldade com a linguagem Prolog, mais do que nos trabalhos passados. A forma de programar em restrições é bem diferente do que o que estamos acostumados.

Frequentemente encontramos erros de "argumentos não suficientemente instanciados", tendo que usar funções built in ou achar métodos menos convencionais.