UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Universidade Federal de Santa Catarina- UFSC
Departamento de Informática e Estatística (INE)
Ciência da Computação
INE5416 - Paradigmas de Programação

Leonardo Lima Appio (21101963)
Fillipi Mangrich Costa de Souza (21202110)
Arthur Scarpatto Rodrigues (21200068)

Análise do problema

Kojun é um quebra-cabeça lógico solucionável por meio da técnica de backtracking, que envolve a experimentação metódica de possibilidades. As regras do jogo exigem que cada célula seja preenchida com um número, garantindo que números iguais não sejam adjacentes. Adicionalmente, é proibido ter valores repetidos dentro de um mesmo grupo e cada grupo deve organizar seus números em ordem decrescente na direção vertical, posicionando o maior número no topo e o menor na base.

Solução e estratégia da implementação

Tabuleiro: A entrada do tabuleiro possui o seguinte formato. O primeiro valor de cada lista interna corresponde ao grupo e o segundo ao valor inicial de cada célula no jogo.

```
/* Definição do tabuleiro inicial com grupos e posições vazias*/
tabuleiro([[[1,2],[1,_],[2,_],[2,_],[2,1],[3,_]],
           [[4,_],[4,_],[4,_],[4 ,3],[4,_],[3,_]],
           [[5,_],[6,3],[6,_],[6,_],[4,5],[7,3]],
           [[5,_],[5,_],[5,_],[6,_],[7,_],[7,_]],
           [[8,_],[8,_],[10,3],[0,_],[0,4],[0,2]],
           [[9,_],[9,_],[10,_],[10,_],[0,_],[0,_]]]).
```

Grupos: Definição do tamanho de cada grupo. Os valores devem ser colocados manualmente pelo usuário.

```
/* Tamanho de cada grupo (grupo, tamanho) */
grupo_quantidade(0,5).
grupo_quantidade(1,2).
grupo_quantidade(2,3).
grupo_quantidade(3,2).
grupo_quantidade(4,6).
grupo_quantidade(5,4).
grupo_quantidade(6,4).
grupo_quantidade(7,3).
grupo_quantidade(8,2).
grupo_quantidade(9,2).
grupo_quantidade(10,3).
```

Regras:

Valores máximos no grupo: O maior valor que os valores de cada grupo podem assumir.

```
/* Define o valor máximo que os valores de cada grupo podem assumir, com base no tamanho do grupo. */
valor_maximo_grupo([R,X]) :- grupo_quantidade(R,T), X in 1..T.
```

Vizinhos diferentes: Todos vizinhos são diferentes (predicado executado para a matriz em linhas e a matriz em coluna).

```
/* Verifica se o vizinho a direita eh diferente, de forma recursiva */
vizinhos_diferentes([[_,_]]).
vizinhos_diferentes([[_,X1],[R2,X2]|T]) :-
    X1 #\= X2, append([[R2,X2]],T,L), vizinhos_diferentes(L).
```

Superior maior: Posições superiores em um mesmo grupo devem possuir valores maiores do que as posições inferiores.

```
/* Verifica se o valor acima de outro eh maior, se fizerem parte do mesmo grupo */
superior_maior([[_,_]]).
superior_maior([[R1,X1],[R2,X2]|T]) :-
    ((R1 #\= R2);
    (X1 #> X2)), append([[R2,X2]],T,L), superior_maior(L).
```

Todos os grupos diferentes: Todos os valores de um grupo devem ser diferentes.

```
/* Verifica se os membros de uma lista sao diferentes */
todos_diferentes_grupo([H]) :-
    all_distinct(H).
todos_diferentes_grupo([H|T]) :-
    all_distinct(H),
    todos_diferentes_grupo(T).
```

Vantagens e Desvantagens

Prolog é mais fácil de implementar, podemos reparar na quantidade de linhas que deu bem menor do que o trabalho feito em Haskell, por exemplo. Por ser uma linguagem declarativa, as regras e restrições do jogo foram definidas de forma clara e concisa, como a verificação de números repetidos em linhas e colunas, facilitando o desenvolvimento. O uso de “trace” auxiliou a executar o código de forma passo a passo, a fim de encontrar onde estavam os problemas no código e tentar corrigi-los. Entretanto, é pior no desempenho com grande volume de dados e menos eficiente que as soluções implementadas nos dois primeiros trabalhos.

Organização do grupo

O grupo se reuniu por meio do discord para a comunicação e entendimento do jogo. As funções foram sendo implementadas gradualmente pelos membros do grupo e avaliadas e corrigidas pelos outros, quando necessário.

Dificuldades e resolução

As maiores dificuldades foram encontrar erros no código e adaptar o código ao novo paradigma, uma vez que tem grande diferença com o Haskell. O código foi reestruturado para se adaptar ao paradigma lógico, contendo muitas diferenças em relação ao que foi feito no paradigma funcional, uma vez que o foco da implementação agora foram os predicados e fatos e não mais tentar manipular as estruturas de matrizes e listas com funções e retornos. Ainda que com um código inteiramente novo, conhecer o problema do Kojun e estratégias para sua resolução facilitou a implementação, guiando quanto a necessidade do que cada predicado deveria ser responsável. Por ser um paradigma completamente diferente, por vezes dificultou o desenvolvimento da solução em si, principalmente para entender o que estava sendo feito de errado em cada predicado.