2021/2022
Diogo Faustino Nº40968
João Rouxinol Nº44451
André Rato Nº45517

Implementação de Strimko

Introdução

No âmbito da UC de **Programação 3** foi proposta a implementação de um programa que fosse capaz de resolver puzzles do **Strimko**. O **Strimko** é um puzzle lógico criado em 2008 que consiste num tabuleiro de tamanho **NxN**, onde cada um dos espaços no tabuleiro ser preenchido por um número de **1 a N**, seguindo as seguintes restrições:

1. Cada **Linha** deve conter números diferentes.
2. Cada **Coluna** deve conter números diferentes.
3. Cada **Stream** deve conter números diferentes.

Uma **Stream** é um caminho no tabuleiro que pode ter várias direções, tornando assim o puzzle mais desafiante.

Foi pedida a implementação das seguintes componentes:

- Predicado **go/1** que tem como argumento o nome do ficheiro;

O programa foi desenvolvido em **SWI-PROLOG.**

Desenvolvimento e Solução Encontrada

De modo a conseguir encontrar uma solução para o problema pedido, foi primeiro comparado o puzzle **Strimko** ao puzzle **Sudoku**, sendo que a única diferença mais notória entre os dois é:

- No **Sudoku**, é feita uma verificação das linhas, colunas e sub tabuleiro (cada tabuleiro de sudoku **9x9** está dividido em 9 tabuleiros de **3x3**.
- No **Strimko** é feita uma verificação das linhas, colunas e **Streams.**

Após esta análise, é possível perceber que, ao encontrar uma solução para o **Sudoku** é possível adaptar a mesma para que esta resolva o puzzle **Strimko** (sendo apenas necessário chegar a uma forma de fazer a verificação das **Streams**).

A solução criada para resolver o problema consiste num conjunto de predicados que vão ser descritos de seguida.

Descrição da solução

A solução criada para resolver o problema consiste no seguinte:

1. Ao iniciar o programa, é aberto o ficheiro passado como argumento.
2. O ficheiro é processado e o programa lê o **N** (dimensão das linhas e colunas do tabuleiro **NxN**), a lista de **Streams** e de **Posições pré definidas**.
3. Depois o programa restringe as linhas no aspeto em que cada elemento de uma linha tem que ser diferente, bem como os das colunas (feito através do predicado **transpose/1** e **all_different/1**).
4. De seguida, são colocados os pontos **Predefinidos** no tabuleiro.

5. Após serem colocados os pontos, são verificadas as **Streams** e restringidas também, de modo a que dois elementos da mesma **Stream** não possam ter os mesmos elementos.
6. Finalmente, com o predicado **labelling/2** o tabuleiro é preenchido e a solução é imprimida.

Predicados utilizados

Na execução do programa foram criados os seguintes predicados:

escreve/1 - escreve uma linha com apenas um espaço no final do ficheiro (resolve problema descrito no final do relatório).
Recebe como argumento o nome ficheiro onde vai adicionar a linha.

without_last/1 - retira o ultimo elemento da lista, necessário na leitura das posições onde era gerada uma lista com um elemento vazio.
Recebe como argumentos a lista a retirar o elemento e a lista sem o ultimo elemento.

readN/4 - predicado responsável pela leitura do **N** (tamanho do tabuleiro) e tratamento do ficheiro, chama também os predicados responsáveis pela leitura das **Streams** do tabuleiro e das posições pré colocadas do mesmo.
Recebe como argumentos o nome do ficheiro a processar, **N**, o conjunto de **Streams** e **Posições** pré definidas.

lerStreams/3 - predicado responsável pela leitura das **Streams** do tabuleiro.
Recebe como argumentos a **Stream** da leitura do ficheiro onde vai operar, a lista que vai conter as **Streams** e o **N** (que é o numero de linhas/colunas do tabuleiro).

lerPosicoes/3 - predicado responsável pela leitura das **Posições** do tabuleiro.
Recebe como argumentos a **Stream** da leitura do ficheiro onde vai operar, a lista que vai conter as **Posições** e o **N** (que é o numero de linhas/colunas do tabuleiro).

lerLinha/3 - predicado responsável por ler e tratar cada uma das linhas do tabuleiro. Este predicado é também responsável por criar o tabuleiro de **Streams** (uma lista de listas de valores inteiros) e a lista de **Posições** pré definidas.
Recebe como argumentos a **Stream** da leitura do ficheiro onde vai operar, a lista onde vai o output da leitura, e o número de linhas a ler.

carregaFicheiro/3 - predicado responsável por chamar as funções que operam sobre o ficheiro.
Recebe como argumentos o nome do ficheiro a abrir, a lista das **Streams** do tabuleiro e a lista de **Posições** pré definidas.

imprimeMatriz/3 - predicado responsável por imprimir o tabuleiro no input pedido no enunciado do trabalho prático, após este estar resolvido.
Recebe como argumentos a lista que representa o tabuleiro resolvido, o **N** de linhas e o **N** de colunas.

colocaNumero/2 - predicado responsável por colocar os valores pré definidos no tabuleiro.
Recebe como argumentos a lista que representa o tabuleiro e a lista que representa um valor a colocar (isto é, uma lista com a configuração **Linha, Coluna, Valor**).

procurarPorIndice/4 - predicado responsável por procurar no tabuleiro todos os valores com o índice passado no argumento.
Recebe como argumento a lista de índices, a o tabuleiro, e duas listas, sendo que a segunda vai ser a que vai conter os valores que devem ser todos diferentes (para respeitar a configuração das **Streams**.

streams/3 - predicado responsável por fazer a verificação de que todos os valores da mesma **Stream** são diferentes.
Recebe como argumentos a lista que representa o tabuleiro e a lista que representa as **Streams**.

strimko/1 - predicado responsável pela criação do tabuleiro, chama também todos os predicados auxiliares necessários à resolução do puzzle, bem como a impressão do mesmo no final.
Recebe como argumento o nome do ficheiro que contem a configuração a resolver.

go/1 - predicado usado para iniciar a resolução do puzzle.
Recebe como argumento o nome do ficheiro que contem a configuração a resolver.

Foram também utilizados os seguintes predicados/constraints presentes no **SWI-prolog:**

- **append/1**
- **write/1**
- **writeln/1**
- **open/2**
- **read_line_to_codes/2**
- **atom_codes/2**
- **atom_number/2**
- **maplist/3**
- **atomic_list_concat/3**
- **nth1/3**
- **element/3**
- **length/2**
- **findall/4**
- **between/3**
- **all_different/1**
- **appent/2**
- **transpose/2**
- **maplist/2**
- **flatten/2**
- **numlist/3**
- **labeling/2**

Funcionamento do programa

O funcionamento do programa é muito simples e consiste na realização dos seguintes passos:

1. Criar um ficheiro de texto de nome arbitrário (como por exemplo o ficheiro **board.txt**);
2. Inserir uma configuração de tabuleiro válida, isto é:
    a. Na primeira linha, N, que será o número de linhas do tabuleiro;
    b. Nas próximas N linhas, a configuração das Streams do tabuleiro

c. Nas seguintes linhas, um número arbitrário de posições pré definidas com o formato "LINHA COLUNA NÚMERO.

3. Compilar o programa em SWI-prolog
4. Executar o predicado go('board.txt').

Resultados Obtidos

Para o output descrito no enunciado do problema:

```
4
1 2 2 4
2 1 4 2
3 4 1 3
4 3 3 1
2 2 3
2 3 2
3 3 1
```

Foi obtido o seguinte resultado:

```
?- go('board.txt').
4 2 3 1
1 3 2 4
2 4 1 3
3 1 4 2
true.
```

Conclusões Finais e Problemas Encontrados

Com a realização deste trabalho foi possível concluir os seguintes pontos:

- **SWI-Prolog** permite (através de **labelling**, **all_different** e **definição de domínios**) resolver problemas desta natureza com muita facilidade.
- A **manipulação de listas** e pesquisa recursiva tornam o desenho de programas desta natureza muito mais interessantes.

Foram encontrados os seguintes problemas durante a criação do algoritmo:

- Puzzles grandes tornam-se um pouco mais lento (cerca de 2 segundos para 7x7 e 6 segundos para 9x9), isto acontece devido à utilização mais extensiva do predicado maplist.
- Problemas na leitura do ficheiro que contem o tabuleiro, o que fazia com que a ultima linha nunca fosse escrita, o que foi resolvido com a implementação do predicado **escreve/1**, que escreve uma linha vazia no final do ficheiro que contem o tabuleiro, bem como um problema que causava a inserção de um elemento vazio na lista de Posições pré definidas, resolvido com a implementação do predicado without_last/1