UNIVERSIDADE
DE ÉVORA

Projecto : Nonograma

Relatório de Programação Declarativa

Docente: Salvador Abreu

Trabalho realizado no âmbito
da disciplina de Programação
Declarativa por:
Pedro Louro nº40011
Miguel Carvalho nº41136

1 Introdução

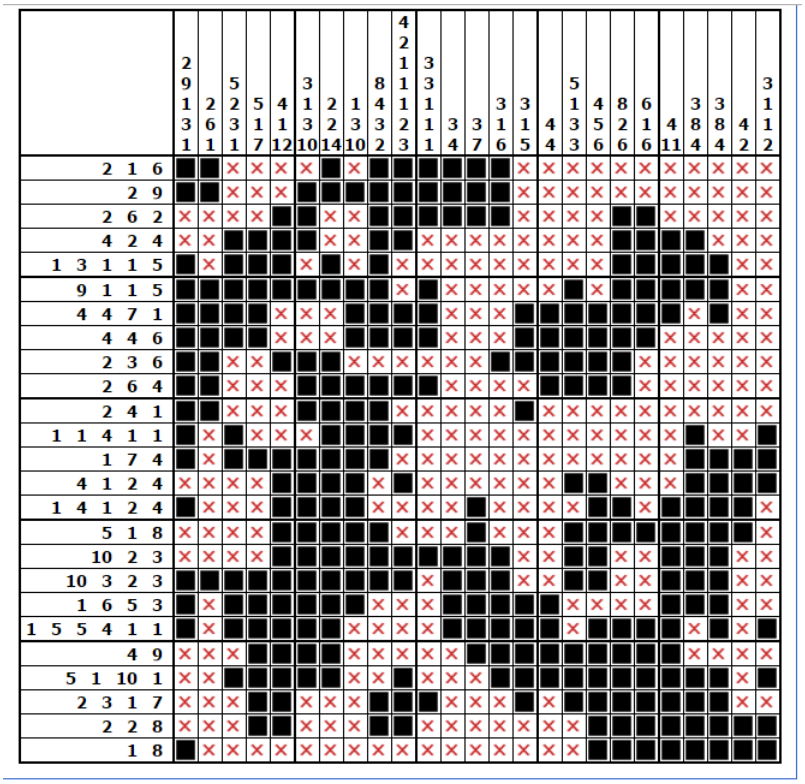
Este trabalho foi realizado a pedido do professor Salvador Abreu de Programação Declarativa, e tem como objetivo a criação de um programa para a resolução por parte dos alunos de um nonograma de acordo com a matéria e a linguagem lecionadas nas aulas.

Como foram lecionadas duas linguagens de programação durante o decorrer da disciplina, o professor deu a escolher aos alunos realizar o trabalho por Programação Lógica (Prolog) ou Programação Funcional (OCaml).

1.1 Descrição do Nonograma

O nonograma é um puzzle japonês, foi inventado por Tetsuya Nishio em 1987. O objectivo do puzzle é descobrir que células serão pintadas e quais ficarão vazias, numa grelha. Os números ao lado da grelha são "dicas" para a resolução do puzzle e vão ditar quantos quadrados consecutivos são preenchidos numa linha ou numa coluna.

Veja-se, por exemplo, a seguinte dica "5 3 2 1 5" significa que existe um conjunto de 5, de 3, de 2, de 1, de 5 quadrados preenchidos, necessariamente por essa ordem, com pelo menos um espaço vazio a separar os blocos. Para a resolução deste puzzle é de igual importância determinar que célula da grelha são preenchidas bem como as células que ficam vazias.



2 Descrição do Trabalho

Para a realização deste projecto, optámos por escolher Programação Funcional (OCaml).

2.1 Implementação

Num estado inicial chama-se a função *puzzle* para tratar do input dado. De seguida, é chamada a função que vai gerar todas as possibilidades de preencher cada linha do tabuleiro.

Imediatamente, atua o algoritmo por trás da *Simple Box*, algoritmo esse que devolve uma lista, com os blocos preenchidos através da sobreposição de matrizes, para cada linha.

Assim, as possibilidades de preenchimento de cada linha são geradas através dos números de espaços possíveis entre cada pista tendo em conta ser possível existir espaços entre o início da linha e da primeira pista. Depois de gerada, a possibilidade é comparada com a linha de blocos preenchidos que a aplicação do algoritmo *Simple Box* devolveu. Caso a possibilidade não tenha os mesmos blocos preenchidos que os blocos que são provenientes da aplicação do algoritmo *Simple Box* esta possibilidade é excluída.

Após o cálculo de todas as possibilidades para cada linha, é chamada a função *solve-puzzle* passando também como argumentos as pistas das linhas, das colunas e o número de colunas(dado pela dimensão da lista de pistas das linhas). Posteriormente, são testadas todas as permutações de cada linha, tentando encontrar a permutação que respeita as pistas dadas pelas linhas e pelas colunas.

Ao ser encontrada a permutação que respeita o explicado acima, o nonograma estará resolvido e a permutação encontrada é imprimida.

2.2 Algoritmo *Simple Box*

Para encontrar uma solução para resolver um nonograma é possível ter uma abordagem matemática para preencher os blocos das linhas e das colunas. Assim sendo o processo é o seguinte:

- Primeiramente adiciona-se as pistas e acrescenta-se mais 1 por cada espaço entre as pistas;

- De seguida subtraímos ao total das linhas ou das colunas, o número que calculámos no passo anterior;
- Qualquer pista que seja maior que o número que se calculou no segundo passo, vai ter algumas das suas células pintadas;
- Para cada pista no terceiro passo subtrai-se o número do segundo passo que vai determinar que células dos blocos é que vão ser pintadas;
- Para preencher os blocos, assume-se que os blocos estão "empurrados" para o lado em que se começa a contar e começando a contar os blocos e preenche-se para trás as células apropriadas (o número que se calculou no passo 4). Isto pode ser feito quer da direita para a esquerda, quer da esquerda para a direita;
- Repete-se o quinto passo para todas as pistas identificadas no passo 3.

2.2.1 Exemplo de Algoritmo *Simple Box* (1)

Veja-se, por exemplo, a seguinte pista "6 2 3".

Seguindo os passos descritos acima, temos que:

Primeiramente, adicionar as pistas com os espaços. Assim sendo, (6+2+3)+(1+1)=13. Suponha-se que a linha era de 15 células. Subtrai-se o valor dado acima às células, 15-13=2.

Seguidamente a cada pista é se subtraído o número do passo anterior. Assim, (6-2=4) e (3-2=1), para o caso da pista "6" serão pintadas 4 das 6 células do bloco e para a pista "3" será pintada uma célula do bloco. Como para a pista 2, não seria pintada nenhuma célula porque (2-2=0), não se aplica neste passo.

Agora partindo de qualquer lado (esquerdo ou direito) preenche-se as células com os valores: *Da esquerda para a direita.* A nossa primeira pista é "6", assim andamos 6 células para a frente e pintamos apartir da 6ª células 4 células para trás.

Repete-se o processo para todas as células seguintes.

3 Resultados Obtidos

Testámos vários nonogramas de forma a perceber a eficiência do código desenvolvido.

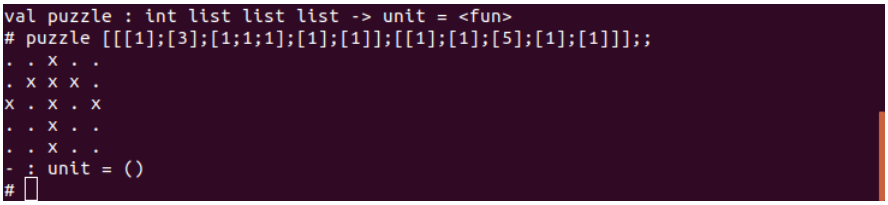


Figure 1: Seta

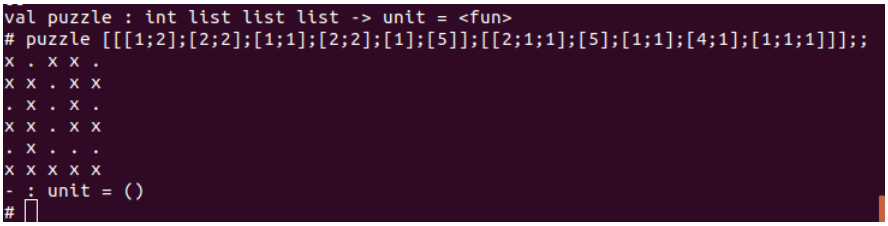


Figure 2: Exemplo 5x6

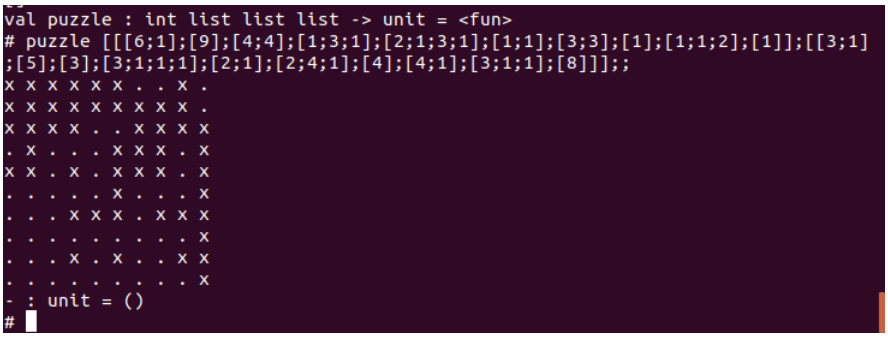


Figure 3: Exemplo 10x10

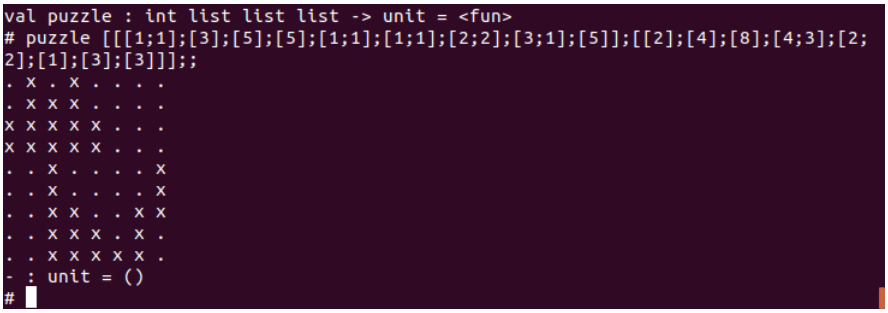


Figure 4: Gato

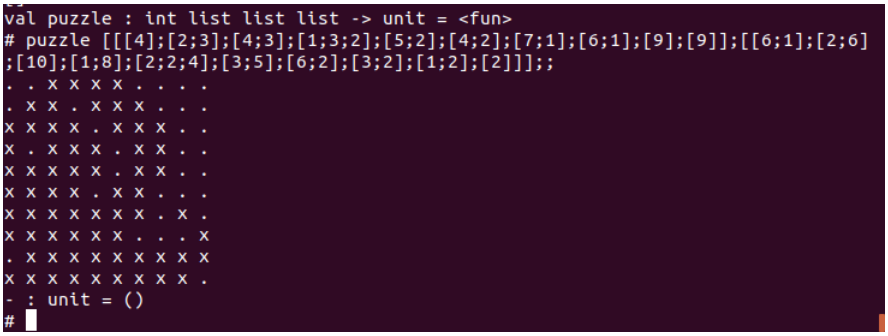


Figure 5: Caracol

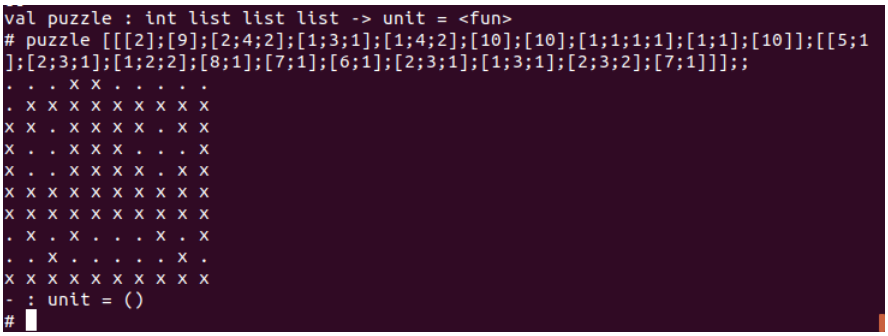


Figure 6: Ambulância

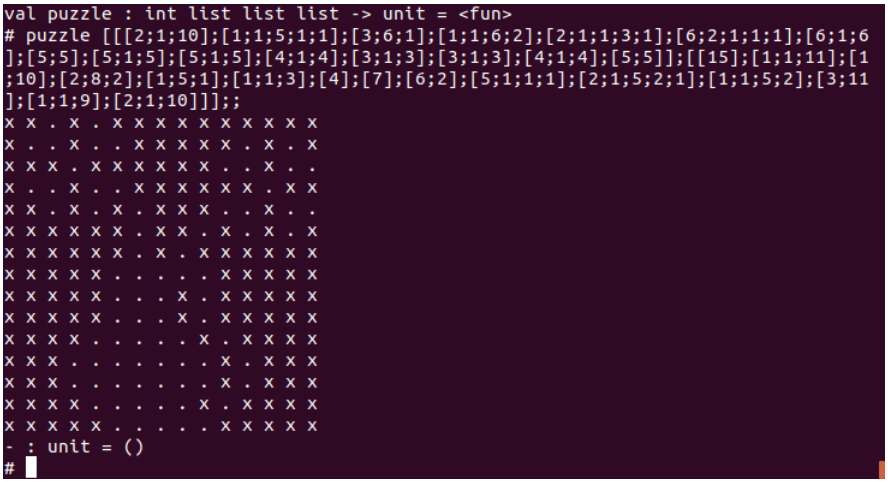


Figure 7: Flores

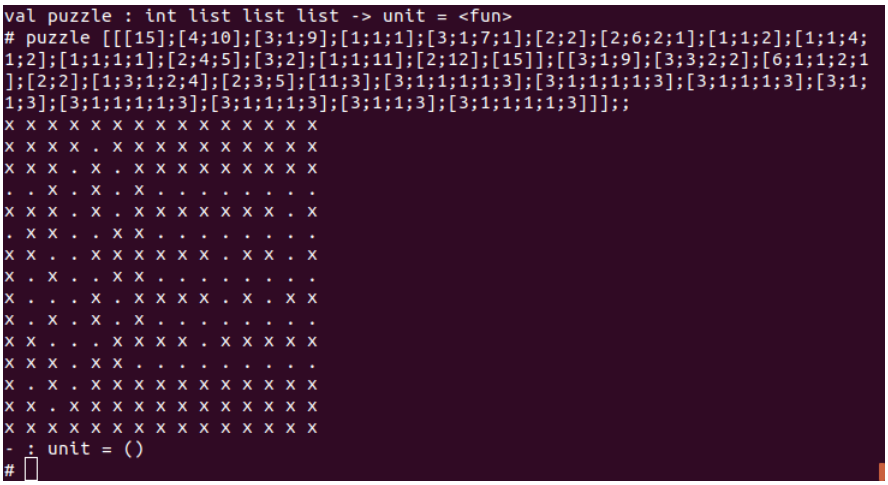


Figure 8: Pauta Musical

A partir destes exemplos conseguimos perceber que a rapidez com que um nonograma é resolvido depende do número de possibilidades de cada linha, mais concretamente, o número de permutações de possibilidades de cada linha que teêm de ser verificados até chegar à certa.

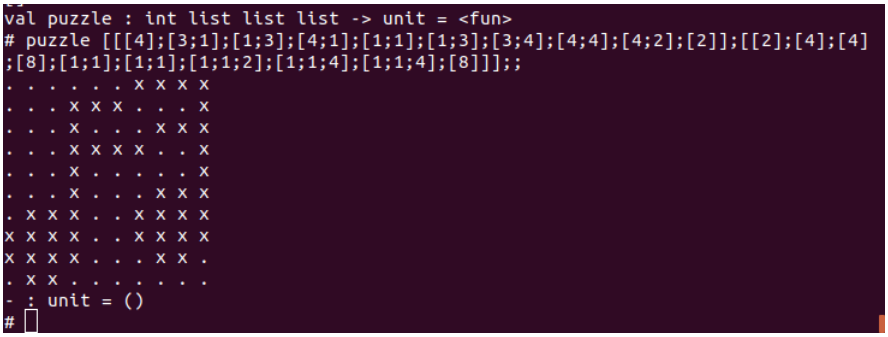


Figure 9: Nota Musical

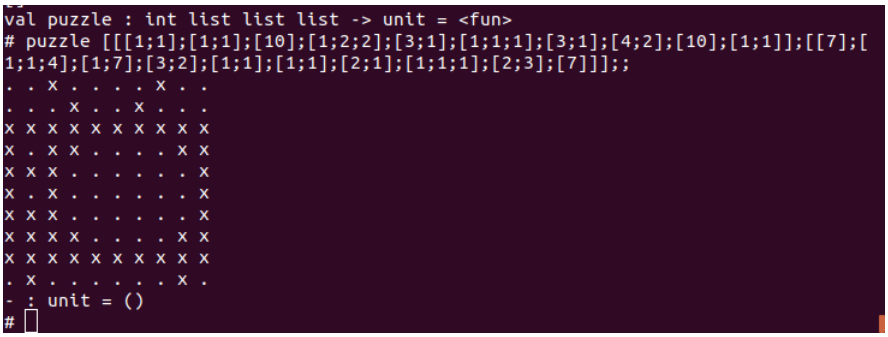


Figure 10: Televisão

Estes últimos dois exemplos, apesar de serem de dimensão menor a exemplos anteriores, precisam de mais tempo para serem resolvidos devido ao maior número de permutações de possibilidades de cada linha que teêm de ser verificadas.

4 Conclusão

A partir dos resultados obtidos, conseguimos perceber que a resolução de nonogramas será mais eficiente quanto menos permutações de possibilidades de cada linha sejam necessárias verificar até chegar à certa e, consequentemente, quantas menos possibilidades para cada linha exitirem.

Visto isto, a resulução pode ser otimizada encontrando mais estratégias de eliminação de possibilidades para uma dada linha para além da já implementada no código desenvolvido

Em suma, com este trabalho, achamos que conseguimos adquirir várias competências, quer sobre Programação Funcional (OCaml) quer sobre encontrar a solução para o resultado do nonograma, que certamente serão úteis no futuro. Com vista na nossa interpretação, acreditamos que conseguimos atingir os objetivos a que nos propusemos com sucesso.

References

[1] `https://en.wikipedia.org/wiki/Nonogram`