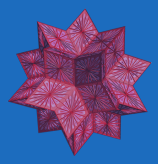
Projeto Computacional
1.ª Parte

RELATÓRIO

MATEMÁTICA EXPERIMENTAL



Trabalho realizado por:
Afonso Ribeiro, ist1102763
Diogo Rodrigues, ist1113787
Pedro Mendes, ist1109994

Com o apoio dos professores:
Pedro Lima
André Crispim

1. Número de Trigg

a)

- `HasDistinctDig[n_Integer] /; IntegerLength[n] <= 4 :=`
  `If[IntegerLength[n] == 4,`
  `Length[Union[IntegerDigits[n]]] > 1,`
  `True]`

A função **FourDifDig** recebe como argumento um número inteiro constituído por quatro algarismos, no máximo. Se o operador atribuir à função um número inteiro constituído exatamente por quatro dígitos, com primeiro dígito não nulo (IntegerLength[n]==4), a função converte o número fornecido para uma lista dos seus dígitos decimais (IntegerDigits[n]), na qual foram eliminados todos os algarismos duplicados utilizando o comando Union. A função avalia se o número dado é constituído por algarismos todos iguais, vendo se o comprimento (Length) da lista criada é maior do que 1. Se este for maior que 1, o número não é formado por quatro dígitos iguais. Se o número introduzido for constituído por menos de quatro dígitos com primeiro dígito não nulo, a função avalia esse número com True (de seguida vamos querer adicionar zeros à esquerda até o número ter 4 dígitos. Assim, esse número nunca terá 4 dígitos iguais.

- `FilteredList = Select[Range[9999], HasDistinctDig];`

A função `FilteredList` seleciona da lista de todos os números inteiros de 1 a 9999 aqueles que não têm dígitos todos iguais.

- `DigList[n_Integer]:= IntegerDigits[n,10,4]`

A função `DigList` recebe como argumento um número inteiro. Ela converte o número dado pelo operador para uma lista dos seus dígitos na base decimal, garantindo que o número tem um comprimento fixo. Ou seja, se a representação do número introduzido tiver menos de quatro dígitos, o mesmo é preenchido com zeros à esquerda até ser constituído por quatro dígitos.

- `N4 = Map[DigList, FilteredList];`

A função `N4` aplica a função `DigList` a todos os elementos de `FilteredList`, devolvendo uma lista de sublistas, cada uma constituída pelos dígitos dos números pertencentes a `N4`.

- `Length[N4]`

O comando `Length[N4]` devolve o comprimento da lista `N4`, ou seja, indica quantos números constituem o conjunto `N4`.

b)

- `alfa[n_Integer]:= Sort[IntegerDigits[n,10,4]][[4]]`
- `beta[n_Integer]:= Sort[IntegerDigits[n,10,4]][[3]]`
- `gamma[n_Integer]:= Sort[IntegerDigits[n,10,4]][[2]]`
- `delta[n_Integer]:= Sort[IntegerDigits[n,10,4]][[1]]`

Cada uma destas funções recebe como argumento um número inteiro. Cada uma delas devolve o i-ésimo algarismo da lista dos dígitos do número introduzido organizada por ordem crescente.

- `T[n_Integer]:= FromDigits[beta[n], alfa[n], delta[n], gamma[n]] - FromDigits[Reverse[beta[n], alfa[n], delta[n], gamma[n]]]`

A função `T` recebe como argumento um número inteiro. Ela efetua a diferença entre dois números inteiros construídos a partir de uma lista ordenada de forma diferente. No primeiro número, os algarismos que ocupam as posições 1 e 3 trocam, entre si, de posição, e o mesmo acontece com os algarismos que ocupam as posições 2 e 4. No segundo número, a ordem dos algarismos é invertida em relação ao primeiro.

- `T[1023]`
- `T[1234]`
- `T[5626]`

Os comandos `T[1023]`, `T[1234]` e `T[5626]` calculam a imagem dos números 1023, 1234 e 5626, respetivamente, pela função `T`.

c)

- `FindFixedPoint[n_Integer]:= NestWhile[T, n, Function[x, T[x]!=x]]`

A função `FindFixedPoint` recebe como argumento um número inteiro. Partindo do número introduzido, a função `T` é iterada sucessivas vezes até que a condição `T[x]!=x` seja falsa, ou seja, até encontrar o seu ponto fixo. A função `FindFixedPoint` devolve o ponto fixo do número introduzido.

Nota: A função FindFixedPoint está bem definida para qualquer número pertencente a N4 (todos os números pertencentes a N4 têm ponto fixo).

- `IterList[n_Integer]:= NestWhileList[T, n, Function[x, T[x]!=x]]`

A função `IterList` recebe como argumento um número inteiro. Partindo do número introduzido, a função `T` é iterada sucessivas vezes até que a condição `T[x]!=x` seja falsa. A função `IterList` devolve uma lista que contém os números obtidos em cada iteração.

- `FindFixedPoint[1023]`
- `IterList[1023]`
- `FindFixedPoint[1234]`
- `IterList[1234]`
- `FindFixedPoint[5626]`
- `IterList[5626]`

Os comandos `FindFixedPoint[1023]`, `FindFixedPoint[1234]` e `FindFixedPoint[5626]` devolvem todos o número 2538, ou seja, o ponto fixo de 1023, 1234 e 5626 é 2538. Os comandos `IterList[1023]` e `IterList[1234]` devolvem, respetivamente, as listas {1023, 1269, 4716, 2538} e {1234, 1269, 4716, 2538}. Já o comando `IterList[5626]` devolve uma lista diferente, resultante de um maior número de iterações: {5626, 1359, 2718, 5625, 360, 3537, 2358, 2538}.

- `DataFixedPoint = Transpose[{{1023,1234,5626}, Map[IterList,{1023,1234,5626}]}]; Grid[Prepend[DataFixedPoint,{Subscript[n,0], "Lista de Iterações Realizadas"}], Frame->All]`

Resultados obtidos sob a forma de tabela:

n_0	**Lista de Iterações Realizadas**
1023	$\{1023, 1269, 4716, 2538\}$
1234	$\{1234, 1269, 4716, 2538\}$
5626	$\{5626, 1359, 2718, 5625, 360, 3537, 2358, 2538\}$

d)

- `N4List:= Map[FromDigits, N4]`

A função `N4List` aplica a função `FromDigits` a todos os elementos que constituem `N4`, ou seja, converte cada uma das listas de dígitos de `N4` para o número correspondente.

- `Union[Map[FindFixedPoint, N4List]]`

O comando `Union[Map[FindFixedPoint, N4List]]` aplica a função `FindFixedPoint` a todos os elementos de `N4List`, ou seja, calcula o ponto fixo de cada número pertencente a `N4List`, devolvendo uma lista com todos os pontos fixos obtidos para cada número. Como a lista obtida era muito extensa, utilizou-se o comando `Union` para remover os pontos fixos repetidos, tendo-se obtido a lista $\{2538\}$. Logo, conclui-se que a iteração de Trigg leva ao ponto fixo de cada um dos 9990 números que constituem `N4`, e esse ponto fixo é igual para qualquer número pertencente a `N4`.

- `NumIterList:=Map[NumIter,N4List]`

A função `NumIterList` aplica a função `NumIter` a todos os elementos que constituem `N4List`, ou seja, calcula o número de iterações necessárias para chegar ao ponto fixo para cada um dos números que constituem `N4`, devolvendo o resultado sob a forma de lista.

- `Max[NumIterList]`

O comando `Max[NumIterList]` devolve o maior valor de `NumIterList`, ou seja, devolve o número máximo de iterações necessárias para chegar ao ponto fixo.

- `NumElementsIter[n_Integer]:= Length[Select[N4List,Function[x,NumIter[x]==n]]]`

Dado um número inteiro, a função `NumElementsIter` calcula a quantidade de elementos selecionados de `N4List` cujo o número de iterações necessárias para chegar ao ponto fixo é o número introduzido pelo operador, apresentando o resultado sob a forma de lista.

- `NumElementsIterList=Map[NumElementsIter,Range[Max[NumIterList]]]`

A função `NumElementsIterList` aplica a função `NumElementsIter` a todos os elementos da lista {1,2,3,4,5,6,7} (o número máximo de iterações é 7), ou seja, calcula para quantos elementos do conjunto `N4` o ponto fixo foi atingido a fim de *i* iterações.

- `DataIter=Transpose[{Range[Max[NumIterList]],NumElementsIterList}];`
  `Grid[Prepend[DataIter,{"i","Número de Elementos"}], Frame->All]`

Resultados obtidos sob a forma de tabela:

i	Número de Elementos
1	383
2	576
3	2400
4	1272
5	1518
6	1656
7	2184

Nota: É de notar que a soma de elementos é 9989 (total de elementos de `N4`, exceto o próprio ponto fixo, cujo número de iterações é 0).

e)

- `ListD[n_Integer,p_List]:= Reverse[Sort[IntegerDigits[n,10,Length[p]]]]`

Dado um número inteiro e uma permutação sob a forma de lista, a função `ListD` ordena, por ordem decrescente, a lista dos dígitos, na base decimal, do número introduzido, preenchendo com zeros, à esquerda, até ser constituída pelo mesmo número de dígitos da permutação dada.

- `NumPermut[n_Integer,p_List]:=`
  `Map[Function[x,ListD[n,p][[p[[x]]]]],Range[Length[p]]]`

Dado um número inteiro e uma permutação representada sob a forma de lista, a função `NumPermut` devolve a `ListD` reordenada segundo a permutação inserida. Cada elemento `p[[i]]` indica a posição do dígito a ser selecionado da `ListD`.

- `TP[n_Integer,σ_List,δ_List]:=`
  `FromDigits[NumPermut[n,σ]] - FromDigits[NumPermut[n,δ]]`

Dado um número inteiro e duas permutações representadas sob a forma de lista, a função `TP` efetua a diferença entre o número construído a partir da `ListD` reordenada segundo a primeira permutação inserida e o número construído a partir da `ListD` reordenada por aplicação da segunda permutação.

f)

- `FindFixedPointPermut[n_Integer,σ_List,δ_List]:=`
  `NestWhile[Function[x,TP[x,σ,δ]],n,Function[x,TP[x,σ,δ]!=x]]`

A função `FindFixedPointPermut` recebe, como argumento, um número inteiro e duas permutações representadas sob a forma de lista. Partindo do número introduzido e estando fixas as permutações dadas, a função `TP` é iterada sucessivas vezes até que a condição `TP[x, ō3c3,ō3b4] != x` seja falsa, isto é, até encontrar o seu ponto fixo. A função `FindFixedPointPermut` devolve o ponto fixo do número introduzido segundo as permutações dadas pelo operador.

- `IterListPermut[n_Integer,σ_List,δ_List]:=`
  `NestWhileList[Function[x,TP[x,σ,δ]],n,Function[x,TP[x,σ,δ]!=x],1,20]`

A função `IterListPermut` recebe, como argumento, um número inteiro e duas permutações representadas sob a forma de lista. Partindo do número introduzido e encontrando-se fixas as permutações dadas, a função `TP` é iterada até a um máximo de 20 vezes. Se, entretanto, a condição `TP[x] != x` for falsa, ou seja, se for encontrado o ponto fixo do número, segundo o par de permutações dado, a função devolve a lista dos números resultantes das respetivas iterações até ponto fixo. Se ao fim das 20 iterações não for encontrado o ponto fixo, a função `IterListPermut` devolve uma lista que contém os números resultantes das 20 primeiras iterações, impedindo assim que a função entre em loop infinito, no caso de uma sequência de números se repetir ao longo da procura do ponto fixo.

Exemplos:

- `FindFixedPointPermut[1234,{1,2,3,4},{2,1,3,4}]`

O comando acima deveria devolver o ponto fixo de 1234 por aplicação das permutações {1,2,3,4} e {2,1,3,4}. No entanto, o comando, quando executado, permanece a correr infinitamente sem devolver nenhum resultado.

- `IterListPermut[1234,{1,2,3,4},{2,1,3,4}]`

O comando acima devolve uma lista que contém os primeiros vinte números resultantes das primeiras vinte iterações: {1234, 900, 8100, 6300, 2700, 4500, 900, 8100, 6300, 2700, 4500, 900, 8100, 6300, 2700, 4500, 900, 8100, 6300, 2700, 4500}. Por análise da sequência de números obtida, pode-se constatar que a sequência de números {900, 8100, 6300, 2700, 4500} se repete ao longo do ciclo. Desta forma, conclui-se que quando se executa o comando `FindFixedPointPermut` para o número 1234 e permutações {1,2,3,4} e {2,1,3,4}, entra, de facto, em loop infinito. Logo, conclui-se que para este número e permutações não existe ponto fixo.

- `FindFixedPointPermut[1234,{1,2,3,4},{4,3,2,1}]`

Como o comando devolve o ponto fixo de 1234 por aplicação das permutações {1,2,3,4} e {4,3,2,1}, conclui-se que o ponto fixo para o número e permutações em questão é 6174.

- `IterListPermut[1234,{1,2,3,4},{4,3,2,1}]`

O comando acima devolve {1234,3087,8352,6174}, lista dos números resultantes das respetivas iterações, até encontrar o ponto fixo. Conclui-se, portanto, que a função itera 3 vezes até chegar ao ponto fixo.

- `FindFixedPointPermut[1234,{2,4,3,1},{3,2,4,1}]`

O comando acima devolve o número 0. Logo, conclui-se que o ponto fixo de 1234 por aplicação das permutações {2,4,3,1} e {3,2,4,1} é 0.

- `IterListPermut[1234,{2,4,3,1},{3,2,4,1}]`

Este comando devolve a lista {1234,810,900,0}. Logo, conclui-se que a função `IterListPermut` aplica a função `TP` 3 vezes até chegar ao ponto fixo.

2. Números Coprimos

a)

Para a ser congruente com p, o resto da divisão tem de ser 0. Os valores possíveis para o resto pertencem a $\{0, 1, 2, ..., p-1\}$, ou seja, este conjunto tem p elementos. Portanto, existe apenas um número que satisfaz a afirmação inicial: $a \equiv 0$ (mod p). Logo, a probabilidade de a ser congruente com p é $\frac{1}{p}$. Sendo que para b o método é o mesmo, a probabilidade para b também será $\frac{1}{p}$. Multiplicando as duas probabilidades, temos $\frac{1}{p^2}$. Portanto, a probabilidade de a e b serem coprimos é $1 - \frac{1}{p}$. Fazendo o produtório para o conjunto dos números primos, chegamos à fórmula.

b)

Nesta função usámos o `RandomInteger` para n pares de números de um intervalo [1, m]. A primeira parcela indica o intervalo e a segunda indica o número de pares. O output será uma lista contendo várias listas de 2 elementos cada.

c)

Para esta função usámos: a variável "coprimes" para contar os pares em que o máximo divisor comum entre os dois é 1; a "list", que está definida como sendo a lista da alínea anterior para os valores m e n; e o índice "i" para indicar a posição da lista em que estamos. A função Do é importante para percorrer a lista e em cada iteração usamos a função If para verificar se o máximo divisor comum do par é igual a 1 (função GCD). No final do ciclo, dividimos a variável "coprimes" por n, e assim obtemos a probabilidade de dois números serem coprimos. Utilizámos a função Module para definir a função.

d)

Nesta alínea começámos por calcular todas as probabilidades possíveis, para todos os graus de grandeza diferentes. Utilizámos a função N para os números aparecerem automaticamente na forma decimal em vez de fração. Na tabela "PmN" passámos simplesmente os valores das probabilidades para as suas respetivas posições. Na tabela "EmN" subtraímos o valor esperado pelos valores das probabilidades para observarmos a diferença entre os dois. Adicionalmente, usámos a função Abs para o erro ser sempre um valor positivo. De seguida, criámos listas que vamos colocar nas duas tabelas. Na construção das tabelas, utilizámos a função Grid e a função Transpose foi útil para juntar as listas. No final, utilizámos a função Column para podermos dar títulos às duas tabelas.

PmN

m/N	10^1	10^2	10^3	10^4	10^5	10^6
10^1	0.7	0.67	0.627	0.6311	0.62848	0.628823
10^2	0.6	0.59	0.593	0.6121	0.60808	0.608832
10^3	0.5	0.61	0.613	0.6063	0.61014	0.607992
10^4	0.7	0.45	0.614	0.6081	0.60827	0.607698
10^5	0.6	0.63	0.608	0.6054	0.60921	0.608198
10^6	0.5	0.63	0.625	0.5997	0.60844	0.608044

EmN

m/N	10^1	10^2	10^3	10^4	10^5	10^6
10^1	0.0920729	0.0620729	0.0190729	0.0231729	0.0205529	0.0208959
10^2	0.0079271	0.0179271	0.0149271	0.0041729	0.000152898	0.000904898
10^3	0.107927	0.0020729	0.0050729	0.0016271	0.0022129	0.0000648981
10^4	0.0920729	0.157927	0.0060729	0.000172898	0.000342898	0.000229102
10^5	0.0079271	0.0220729	0.0000728981	0.0025271	0.0012829	0.000270898
10^6	0.107927	0.0220729	0.0170729	0.0082271	0.000512898	0.000116898

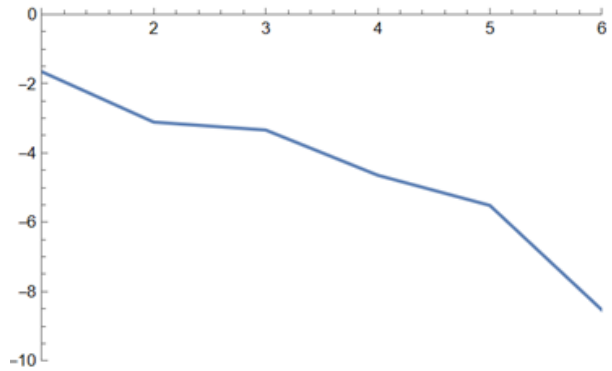
Podemos concluir a partir da tabela EmN que o erro absoluto vai diminuindo de forma consistente à medida que usamos mais números significativos, ou seja, à medida que aumentamos a precisão. Deste modo, é seguro afirmar que as tabelas confirmam a fórmula (1).

Notas:

- O código demora cerca de 10 segundos a correr.
- Sendo os valores dependem de probabilidades, estes variam sempre que corremos o código.
- Os valores para o erro não estão 100% alinhados com o que deveria acontecer (por exemplo, o erro de alguns casos comparado com outros é menor, apesar de os graus de grandeza serem menores), mas considerando que este método é aleatório também é de esperar que haja ligeiras inconsistências. Contudo, analisando de um modo geral, os valores estão perto do esperado (à medida que nos aproximamos do canto inferior direito da tabela, o valor do erro vai diminuindo).

e)

Para esta pergunta começámos por criar a função "Pm", que é simplesmente a fórmula no enunciado, e calculámos para os diferentes valores de m pedidos. De seguida, determinámos os erros para cada m (tal como no exercício anterior) e criámos a lista "Em" com todos estes valores. Criámos ainda uma lista "k", que vai ser útil para os valores das abcissas do gráfico, sendo que estes valores correspondem aos graus de grandeza. A lista "variables" serve para associar as coordenadas x ("k") e y ("Log[10, Em]") de cada ponto usando a função Transpose. Por fim, usámos a função ListPlot para construir o gráfico (com a lista anterior), ligámos os pontos com linhas para facilitar a visualização e indicámos os intervalos mais aconselháveis para a observação do mesmo.



Sendo que o erro tende para 0, então o logaritmo do erro também vai tender para $-\infty$, que é exatamente o que acontece no gráfico. Logo, este gráfico confirma a validade da fórmula (2).

3. Formiga de Langton

a)

A função LangtonsAnt é definida com três argumentos: um inteiro `n` tal que a matriz que representa a grelha tem dimensão $(2n+1)\times(2n+1)$; uma string `d`, que pode tomar os valores "Cima", "Baixo", "Esquerda"ou "Direita", definindo a direção inicial da formiga; e um inteiro `kmax`, correspondente ao número máximo de passos permitido. A função é definida utilizando o comando `Module`, que permite a declaração de variáveis locais.

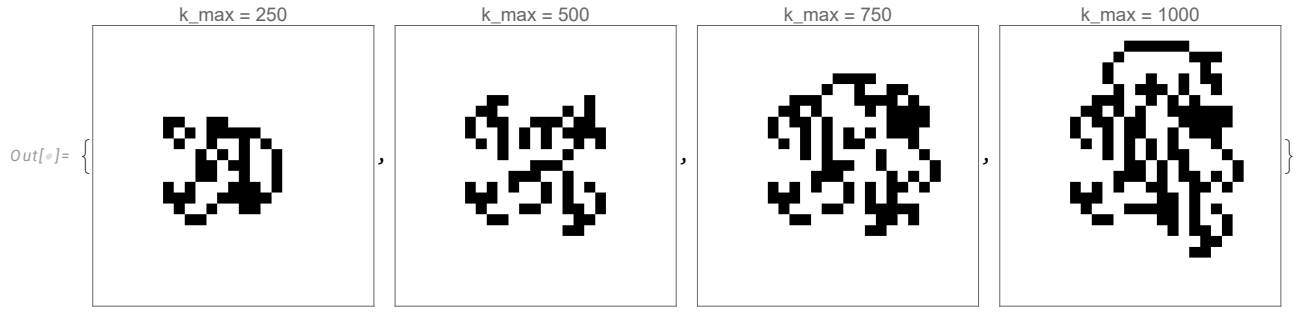
Inicialmente, definem-se duas funções internas, `rotateClockWise` e `rotateCounterclockWise`, que, dado um vetor, retornam o resultado de lhe aplicar uma rotação de 90º no respetivo sentido.

De seguida, inicializa-se a grelha como um `ConstantArray`, com o tamanho calculado por $2n+1$ e com todos os valores a zero. Para além disso, é inicializada a variável que representa a direção como o vetor (lista com 2 elementos) a partir da cadeia de caracteres que a função recebe como argumento.

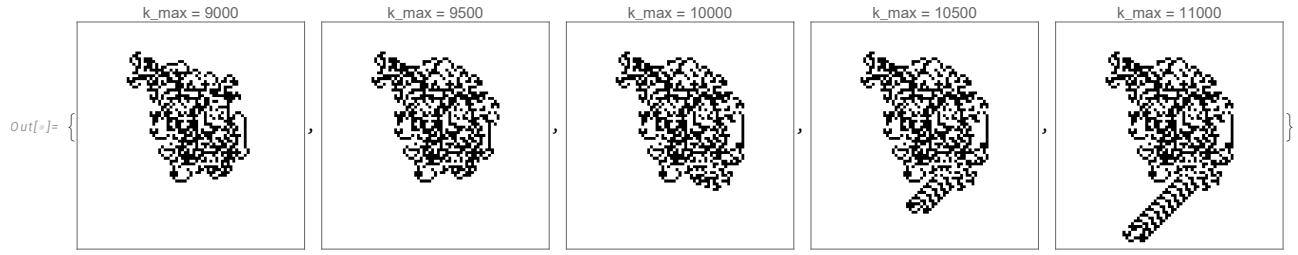
Finalmente, entra-se no while loop que fará a evolução dos vários estados da grelha. Identifica-se a cor da célula atual, para se alterar a direção de acordo com a regra correspondente, e, de seguida, atualiza-se a cor da célula atual e altera-se a posição para a célula seguinte. Este ciclo corre até que a formiga atinja a a fronteira da grelha ou se chegue ao número máximo de passos, `kmax`.

b)

A aplicação da função definida em a) para $k_{max} = \{250, 500, 750, 1000\}$, com valores $n = 12$ e $d = "Direita"$ (sendo o valor de n escolhido de forma a ter uma boa visualização e a não ocorrer o fim do ciclo por atingimento da fronteira), mostra os seguintes resultados:



A aplicação da função para $k_{max} = \{9000, 9500, 10000, 10500, 11000\}$, com valores $n = 38$ e $d = "Direita"$ (sendo n escolhido com o critério anterior), mostra os seguintes resultados:



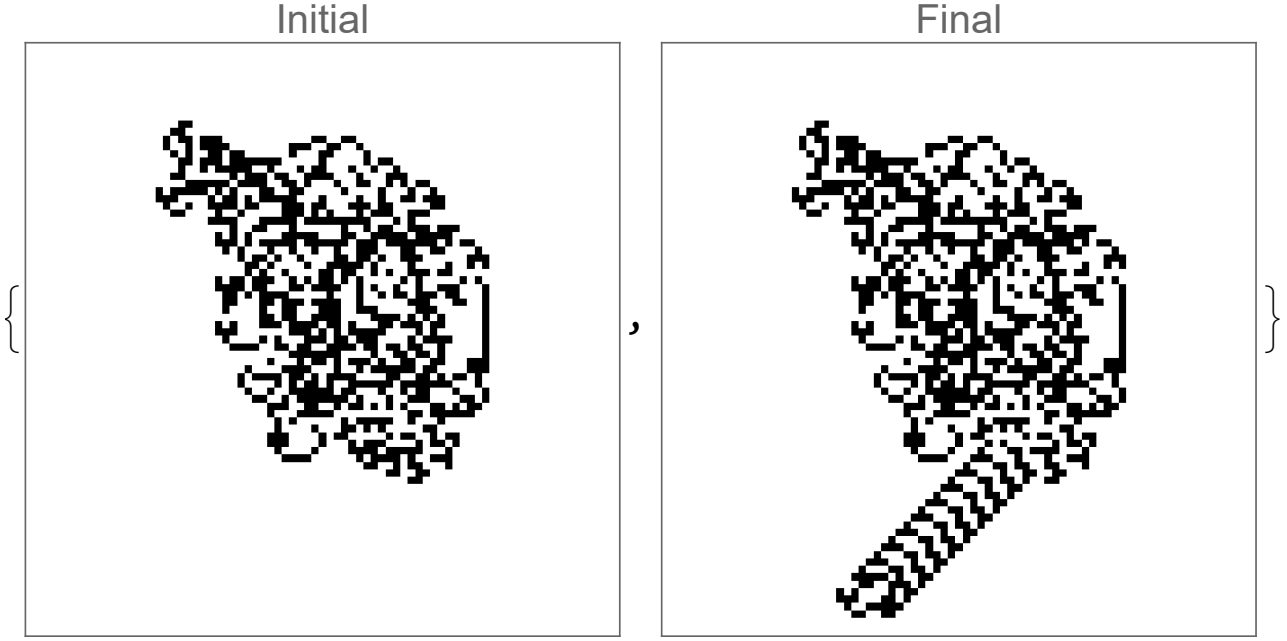
É possível verificar que, neste intervalo de valores de k_{max}, se chega a um ponto em que a formiga passa a descrever um padrão específico, fenómeno este que se analisará de seguida.

c)

Altera-se a função definida em a), de forma a que o seu output não seja apenas a grelha final, mas também a direção final. Isto deve-se ao facto de que o movimento com padrão específico é periódico a menos de translação, ou seja, para identificar e estudar o padrão, a variável de interesse é a direção final da formiga no fim do ciclo para cada valor de k_{max}.

Encontra-se o padrão utilizando-se os mesmos valores de $n = 38$ e $d = "Direita"$, e valores de k_{max} no intervalo de 10000 a 11000.

As grelhas resultantes dos valores de k_{max} de 10000 (inicial) e 11000 (final) são as seguintes:



Utilizando-se o comando `FindRepeat` para a lista das direções finais das chamadas à função para todos os valores de k_{max} entre o valor inicial e o final, obtém-se o seguinte padrão:

```
{{0,1},{1,0},{0,-1},{1,0},{0,-1},{1,0},{0,1},{-1,0},{0,-1},{-1,0},{0,-1},{-1,0},
{0,1},{1,0},{0,-1},{1,0},{0,1},{-1,0},{0,-1},{-1,0},{0,-1},{1,0},{0,-1},{1,0},
{0,1},{-1,0},{0,-1},{-1,0},{0,1},{1,0},{0,-1},{1,0},{0,-1},{1,0},{0,1},{-1,0},
{0,-1},{-1,0},{0,-1},{1,0},{0,1},{-1,0},{0,1},{1,0},{0,-1},{-1,0},{0,-1},{-1,0},
{0,-1},{1,0},{0,1},{-1,0},{0,1},{-1,0},{0,1},{1,0},{0,1},{-1,0},{0,1},{1,0},
{0,-1},{-1,0},{0,-1},{1,0},{0,-1},{-1,0},{0,-1},{1,0},{0,1},{-1,0},{0,1},{1,0},
{0,1},{-1,0},{0,1},{-1,0},{0,1},{1,0},{0,1},{1,0},{0,-1},{1,0},{0,-1},{1,0},
{0,-1},{-1,0},{0,-1},{1,0},{0,1},{-1,0},{0,1},{1,0},{0,1},{1,0},{0,-1},{1,0},
{0,1},{-1,0},{0,-1},{-1,0},{0,1},{-1,0},{0,-1},{1,0}}
```

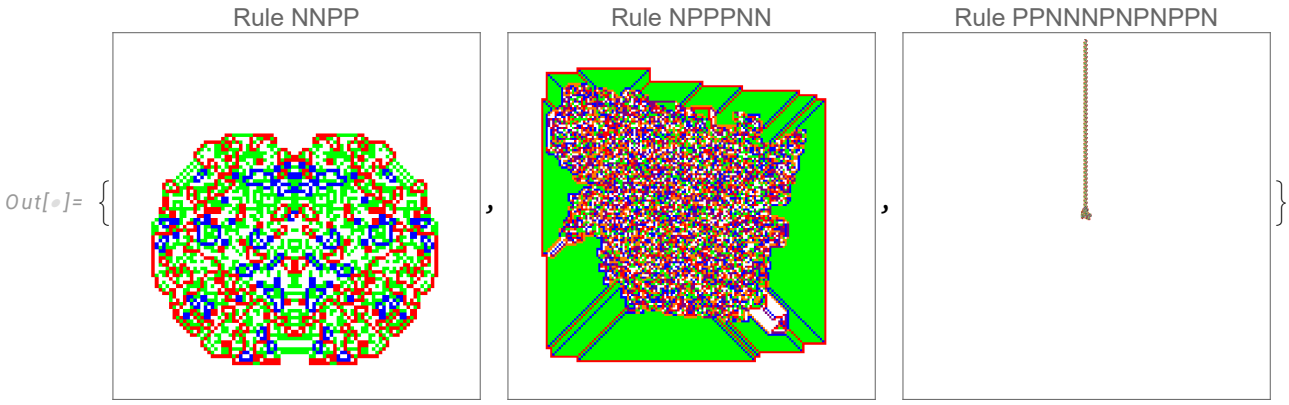
Através do comando `Length`, obtém-se o comprimento desta lista, concluindo-se que o período mínimo deste movimento periódico é de 104.

d)

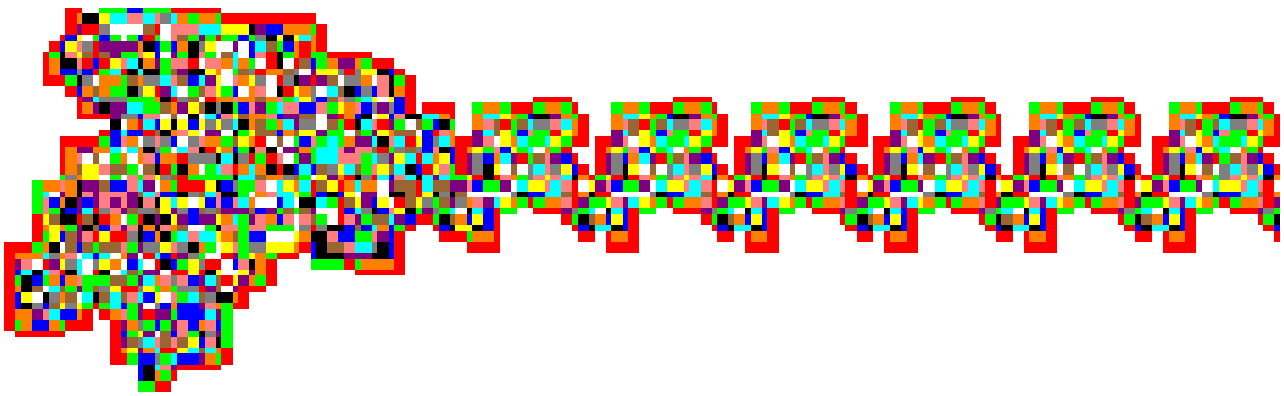
A função definida em a) é alterada, passando agora a ter mais um argumento, a cadeia de caracteres com as regras do sistema, que definem, para cada possível cor de uma célula, qual o sentido da rotação a aplicar na direção da formiga. É também a partir do comprimento desta string que se encontra o número de cores.

O ciclo da função é alterado. Este continua a alterar a direção de acordo com a regra correspondente à cor da célula atual e a atualizar a cor da célula atual, tal como anteriormente, mas o procedimento para o fazer é mais complexo, dado o maior número de cores. As cores são identificadas por inteiros a começar em 0, pelo que, para se obter a cor seguinte, se utiliza o comando `Mod`.

A aplicação da função alterada para $k_{max} = 500000$, com valores $n = \{50, 80, 700\}$ e $d = "Direita"$ (sendo, uma vez mais, os valores de n escolhidos de forma a não ocorrer o fim do ciclo por atingimento da fronteira), mostra os seguintes resultados:



Para a regra "PPNNNPNPNPPN", a visualização fica comprometida pelo formato vertical da trajetória da formiga, pelo que se apresenta de seguida uma ampliação no centro com rotação de 90º no sentido horário dessa imagem:



Verifica-se que, a partir de um certo ponto, a trajetória da formiga passa a descrever um movimento com padrão específico.