Departamento de Matemática e Engenharias

Emulação da Máquina URM no *Mathematica*

José Laurindo de Góis Nóbrega Sobrinho

UMa, 13 de Novembro de 2003

Motivação:

Na cadeira de Paradigmas da Programação (1ºano,1º Semestre) os alunos familiarizaram-se com o sistema *Mathematica*.

Aprenderam a definir funções por *abstracção funcional* e por *atribuição paramétrica*.

Praticaram os paradigmas da programação *recursiva*, *imperativa*, *funcional* e por *reescrita*.

Contrariamente ao que seria desejável acaba sempre por não haver tempo para desenvolver aplicações mais complexas...

Motivação:

Estes programas de aplicação podem no entanto ser introduzidos (em fases posteriores do curso) no âmbito de outras cadeiras.

É o que acontece no caso da *Teoria da Computabilidade e Complexidade* (2ºano,1ºSemestre).

Em anos anteriores foram facultados aos alunos alguns programas, em linguagem *Mathematica*, capazes de resolver questões específicas referentes à matéria leccionada:

- **emulador da máquina URM**
- codificador de programas URM
- descodificador de programas URM

Motivação:

Seria proveitoso para os alunos se fossem eles próprios a construir os referidos programas:

- Interiorizar de forma mais marcante a matéria da disciplina de *Teoria da Computabilidade e Complexidade*.

- Alargar a prática de programação em *Mathematica*.

emulador da máquina URM → **2ª aula Teórico-Prática**

codificador de programas URM
descodificador de programas URM
} **13ª aula Teórico-Prática**

Plano das aulas Teórico-Práticas

1 Construção de programas URM.

2 Construção do emulador da máquina URM (no *Mathematica*).

3 Mais exemplos da construção de programas URM.

4 Demonstração de que certos predicados são decidíveis.
Programas URM com "subprogramas".

5 e 6 Composição e recursão, soma limitada, produto limitado e minimização limitada.

7, 8 e 9 Programação na máquina de Turing.

10 Método da diagonal e da redução.

11 Teorema s-m-n e teorema de Rice.

12 Predicados parcialmente decidíveis. Conjuntos recursivos e enumeráveis.

13 Construção do codificador e descodificador de programas URM.

Plano da Aula

1	Enunciar o problema, apresentar e descrever o plano (sumário) da aula	10m
2	Revisão ou introdução de algumas das funções *Mathematica* a utilizar	30m
3	Elaboração do programa *executaURM*	45m
4	Teste do programa com alguns exemplos	35m

Problema:

Definir uma função *Mathematica*, de nome *executaURM*, que recebendo um programa URM, p, e uma sequência de n números naturais $x_1,...,x_n$, retorna o resultado de aplicar a $x_1,...,x_n$ a função n-ária calculada pelo programa p.

Para evitar a possibilidade de não terminação do cálculo deverá ainda ser indicado, juntamente com os argumentos anteriores, um inteiro positivo i. Assim se não for obtido um resultado num número de passos inferior a i, o programa deverá parar retornando a mensagem: *" O programa não parou ao fim de " i "passos.*".

Exemplo:

```
executaURM [maior,7,8,100]
```

maior é uma variável *Mathematica* onde está guardado um programa URM capaz de calcular a função:

$$f(x,y)=\begin{cases} x, & x \geq y \\ y, & x < y \end{cases}$$

7 e 8 são os argumentos da função (x e y).

100 é o número de passos (i) findos os quais o programa deve parar (se ainda não o tiver feito).

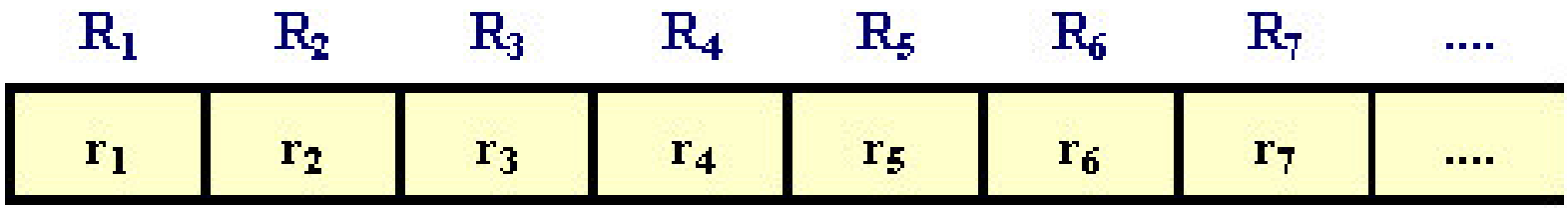
Sumário

- *Revisão ou introdução de algumas das funções Mathematica a utilizar.*
- *Elaboração do programa* ***executaURM.***
- *Teste do programa com alguns exemplos.*

URM - **U**nlimited **R**egister **M**achine

Existe um número infinito de registos designados por R_1, R_2, R_3, ...

Cada registo contém um número natural. O número contido no registo R_n é normalmente designado por r_n.



Instruções reconhecidas pela máquina URM

Nome da Instrução	**Instrução**	**Resultado**
Zero	Z(n)	$r_n \rightarrow 0$
Sucessor	S(n)	$r_n \rightarrow r_n+1$
Transferência	T(m,n)	$r_n \rightarrow r_m$
Salto	J(m,n,q)	Se $r_m=r_n$ então o programa continua na instrução nºq, caso contrário continua na instrução seguinte.

Funcionamento da máquina URM

Um programa URM é constituído por uma sequência de instruções: $I_1, I_2, I_3,\ldots,I_s$.

A execução do programa começa sempre pela instrução I_1.

Se a instrução I_k não for uma instrução de salto então será executada a seguir a instrução I_{k+1}. Se I_k for da forma $J(m,n,q)$ então a instrução a executar a seguir será I_q se $r_m=r_n$ ou I_{k+1} no caso contrário.

A execução de um programa, $I_1, I_2, I_3,\ldots,I_s$, termina quando o índice da instrução a executar a seguir for superior a ***s*** (ou seja quando a instrução a executar a seguir não existir). Se isso nunca acontecer então o programa nunca termina.

Funcionamento da máquina URM

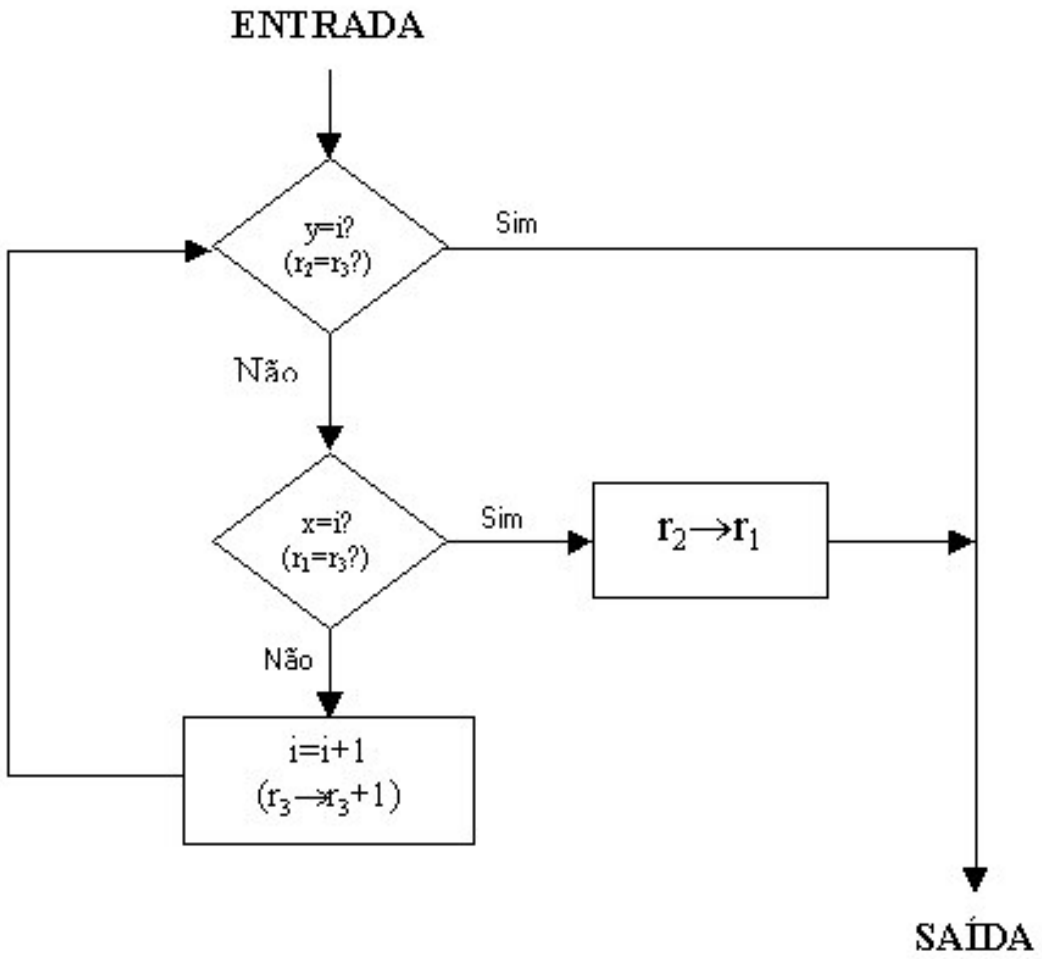
Para que a máquina URM efectue uma computação é necessário fornecer, para além de um programa, uma configuração inicial. Esta configuração consiste numa sequência de números naturais *a1, a2, a3,...* nos registos *R1, R2, R3, ...*

Quando a execução de um programa termina o conteúdo dos registos determinam a configuração final da máquina.

Se utilizarmos a máquina URM para calcular funções de $\mathbb{N}^n$ em $\mathbb{N}$ então os argumentos estão inicialmente nos registos $R_1,...,R_n$ guardando os restantes registos o valor zero. Por convenção o resultado deverá ser devolvido no registo $\boldsymbol{R_1}$.

O programa *maior*

I1 J(2,3,6)
I2 J(3,1,5)
I3 S(3)
I4 J(1,1,1)
I5 T(2,1)



R_1	R_2	R_3	R_4	R_5	R_6	R_7	....
x	y	i	0	0	0	0	....

Alguns preliminares *Mathematica*

1-Como representar as **instruções URM** no *Mathematica* ?

2-Como representar os **programas URM** no *Mathematica* ?

3-Como representar a **estrutura da máquina URM** no *Mathematica* ?

Expressões

Basicamente tudo o que se escreve em *Mathematica* é uma expressão.

x+y é uma expressão

A função ***FullForm*** diz-nos como é que uma dada expressão é armazenada internamente pelo *Mathematica*:

FullForm[x+y] $\longrightarrow$ **Plus[x,y]**

A função ***Head*** diz-nos qual o tipo de entidade que está à cabeça de uma dada expressão:

Head[x+y] $\longrightarrow$ **Plus**

Partes de uma expressão

A expressão *x*+*y* é composta por três partes: um operador (*Plus*) e dois argumentos (*x* e *y*). Para aceder às diferentes partes de uma expressão podemos utilizar o selector de componente *[[]]* ou recorrer à função *Part*:

(x+y)[[1]] ou Part[x+y,1] ⟶ **x**

(x+y)[[2]] ou Part[x+y,2] ⟶ **y**

(x+y)[[0]] ou Part[x+y,0] ⟶ **Plus**

Instruções URM no *Mathematica*

Podemos optar por representar no *Mathematica* as instruções URM por meio de expressões do tipo:

Z[n]

S[n]

T[m,n]

J[m,n,q]

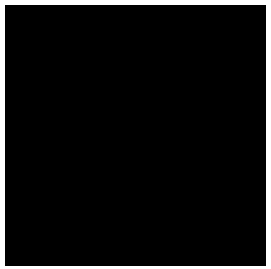
Instrucoes.nb

Moldes

O **molde universal**, representado por _, é verificado por qualquer expressão.

f[_]:={1,2,3}

g[x_Integer]:={x,x+1,x+2}



Moldes.nb

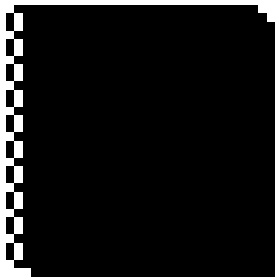
Moldes de sequências de expressões

__ molde satisfeito por uma sequência de uma ou mais expressões.

__h molde satisfeito por uma sequência de uma ou mais expressões, cada uma delas com a cabeça h.

___ molde satisfeito por uma sequência de zero ou mais expressões.

___h molde satisfeito por uma sequência de zero ou mais expressões, cada uma delas com a cabeça h.



Sequencias.nb

Programas URM no *Mathematica*

I1 J(2,3,6)
I2 J(3,1,5)
I3 S(3)
I4 J(1,1,1)
I5 T(2,1)

w = { J[2,3,6], J[3,1,5],S[3], J[1,1,1],T[2,1] }

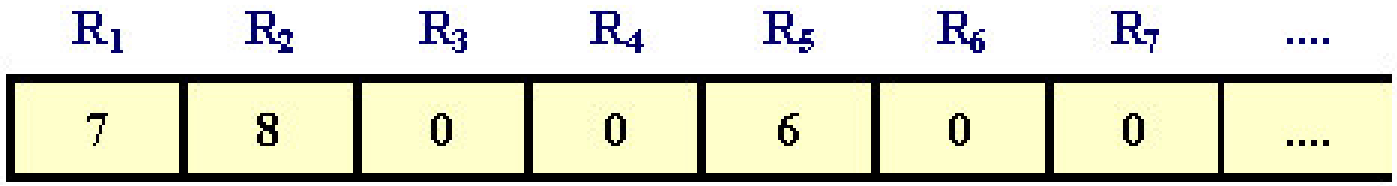
A primeira instrução a executar (I1) é dada por **w[[1]]**.

O número de instruções existentes no programa URM é dada por **Length[w].**

A execução do programa termina se o índice da instrução a executar de seguida for superior **a Length[w]**.

Estrutura da máquina URM no *Mathematica*

Hipótese A: Utilizar uma lista para armazenar os valores dos registos.

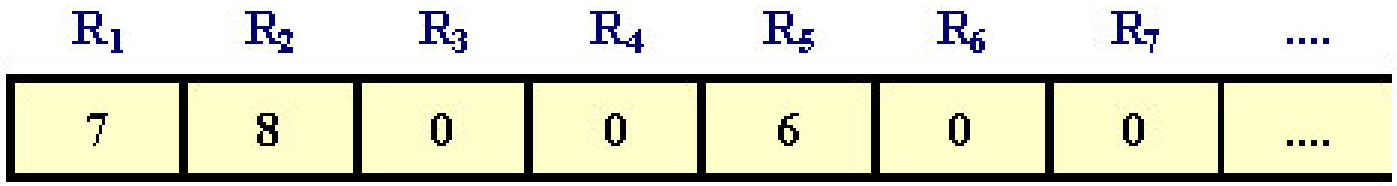



```
m={7,8,0,0,6,0,0,0,0,0}
```

Problema: As listas *Mathematica* guardam um número finito de elementos mas o número de registos existentes na máquina URM é infinito.

Estrutura da máquina URM no *Mathematica*

Hipótese B: Utilizar uma lista "compactada"



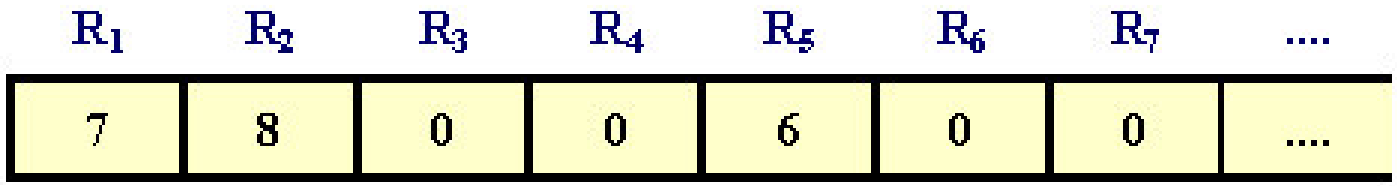

```
m={{1,7},{2,8},{5,6}}
```

São indicados apenas os registos cujo valor é diferente de zero

{nº registo, valor}

Estrutura da máquina URM no *Mathematica*

Hipótese C: Definir uma família de variáveis

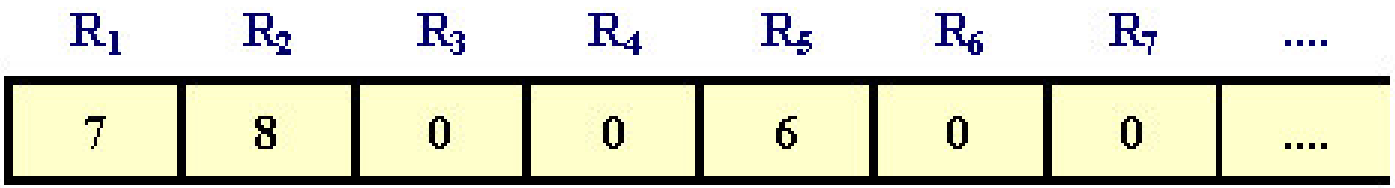



```
m[1]=5;
m[2]=8;
m[5]=6;
m[_]=0
```

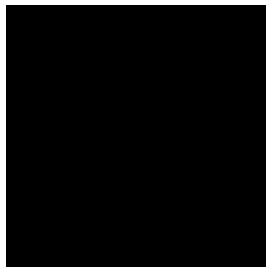
m[i] é o valor guardado no registo $\mathbf{R_i}$.

A regra **m[_]=0** iguala a zero todos os registos excepto aqueles para os quais se tenha definido uma regra mais específica.

Estrutura da máquina URM no *Mathematica*



Exemplo



Registos.nb

Execução do programa URM

Uma vez conhecida a cabeça da instrução URM a executar de seguida há que agir em conformidade. Estamos aqui perante um problema de *composição alternativa*.

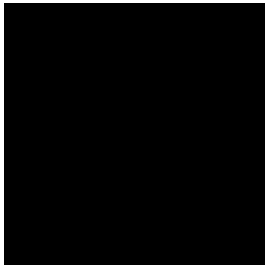
Hipótese A: Utilizar a estrutura *If*

```
If[Head[x] == Z, < executa Zero >,
  If[Head[x] == S , < executa Sucessor >,
    If[Head[x] == T, < executa Transferência >,
< executa Salto >]
    ]
  ]
```

Execução do programa URM

Hipótese B: Utilizar a estrutura *Switch*

```
Switch[Head[x],

  Z, < executa Zero >,
  S, < executa Sucessor >,
  T, < executa Transferência >,
  J, < executa Salto >

  ]
```

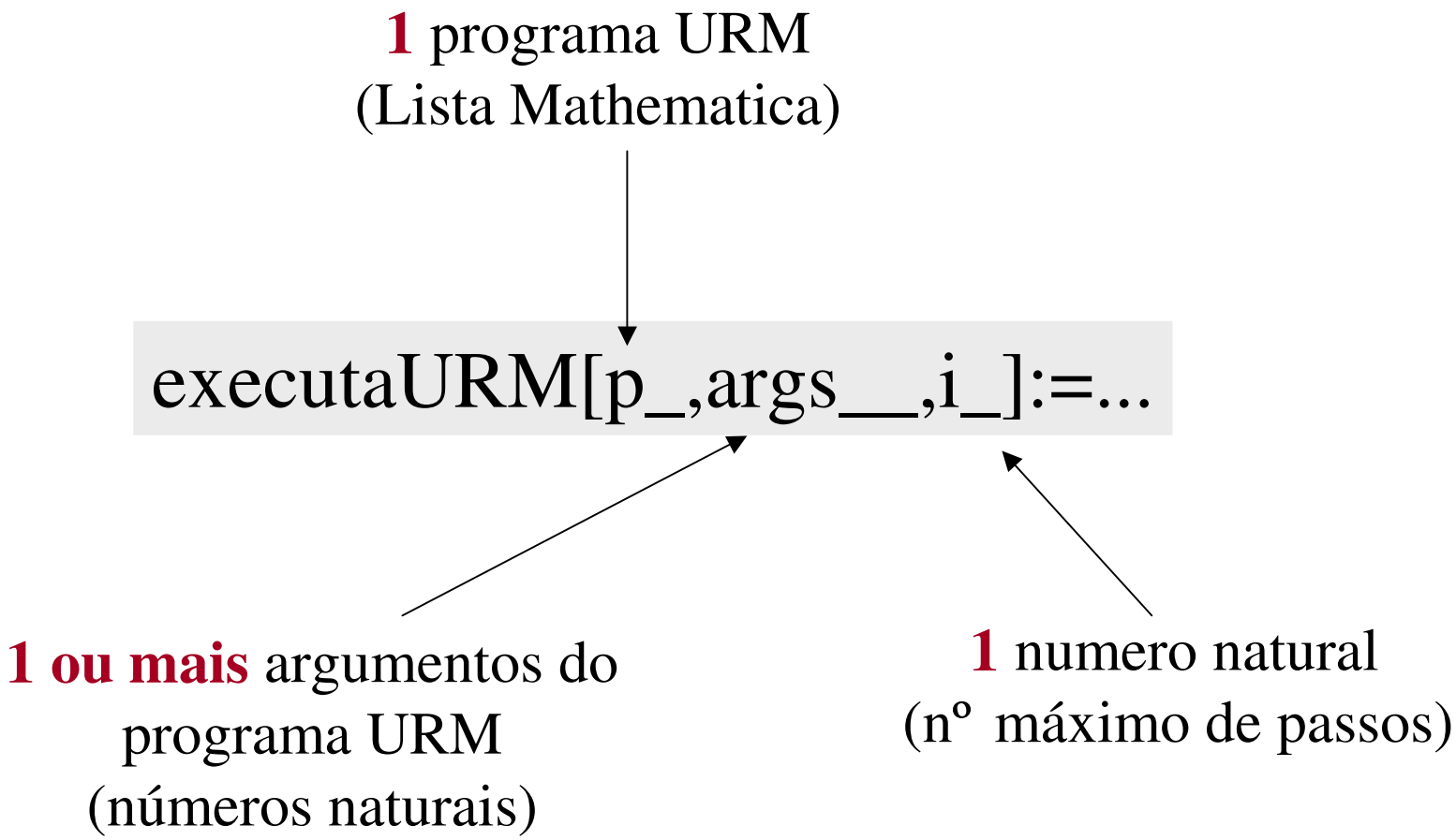


Switch.nb

Estrutura do Programa *executaURM*

executaURM[< Argumentos >] :=
(<Inicialização da máquina URM >;
< Execução do programa URM >;
< Devolução do resultado >;)

Argumentos do programa *executaURM*

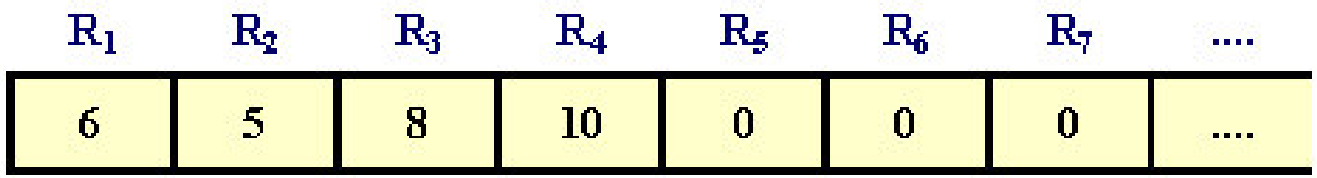


Inicialização da máquina URM

Supondo que foi feita a evocação:

executaURM[prog, 6,5,8,10,16]

devemos estabelecer a seguinte configuração inicial:



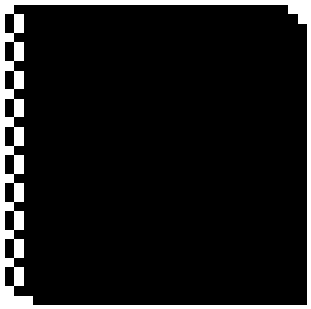
Inicialização da máquina URM

Os argumentos do programa URM são recebidos, pelo programa *Mathematica executaURM*, através de uma sequência, de um ou mais números naturais, que denotamos por *args*. Para transformar esta sequência numa lista:

```
argsList={args}
```

ou

```
argsList=List[args]
```



List.nb

Estrutura do ciclo While

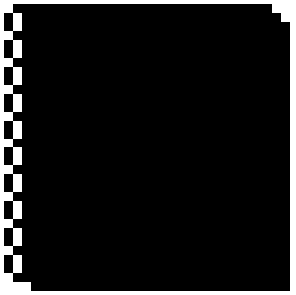
```
< Inicialização >

While [ < Guarda >,

        < Acção >;

        < Progresso >

        ]
```

Inicialização da máquina URM

argsList={6,5,8,10}

m[1]=6
m[2]=5
m[3]=8
m[4]=10
m[_]=0

m[i]=argsList[[i]]



Inicializa.nb

Inicialização da máquina URM

```
argsList = List[args];

n = Length[argsList];
r = 1;

While[r <= n,

m[r] = argsList[[r]];

r = r + 1

];

m[_] = 0;
```

Execução do programa URM

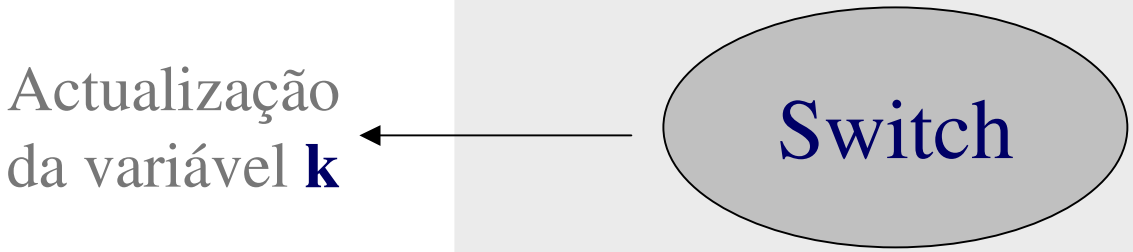
A execução pode ser controlada por um ciclo ***While***

k **-** número ou índice da instrução a executar de seguida. Se essa instrução não existir então o ciclo deverá terminar.

r **-** número de passos (iterações) executados até um dado instante. Se *r* exceder o valor do parâmetro *i* então o ciclo deve terminar.

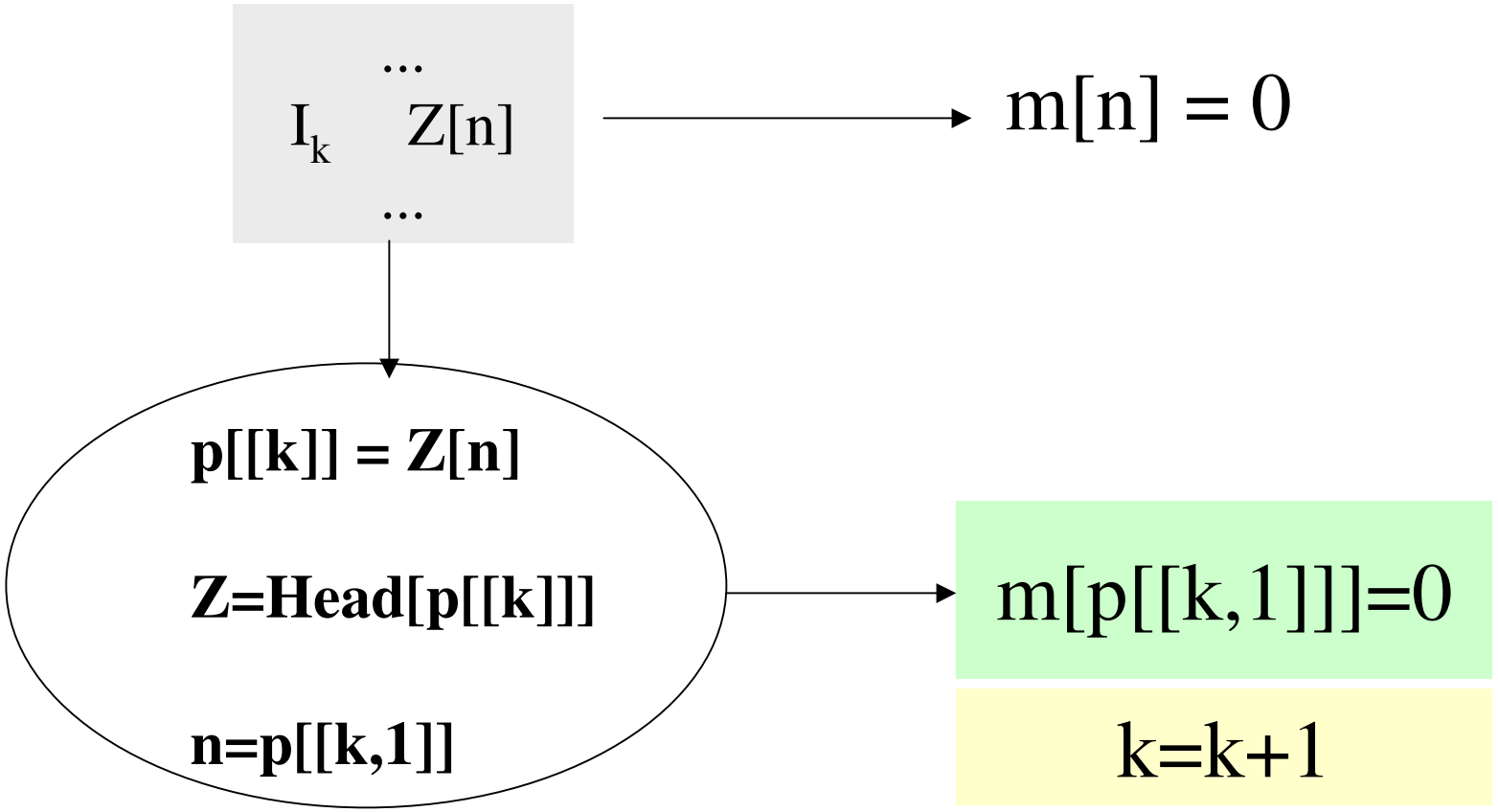
Execução do programa URM

```
s = Length[p];
k = 1;
r = 0;

While[k <= s && r <= i,
```



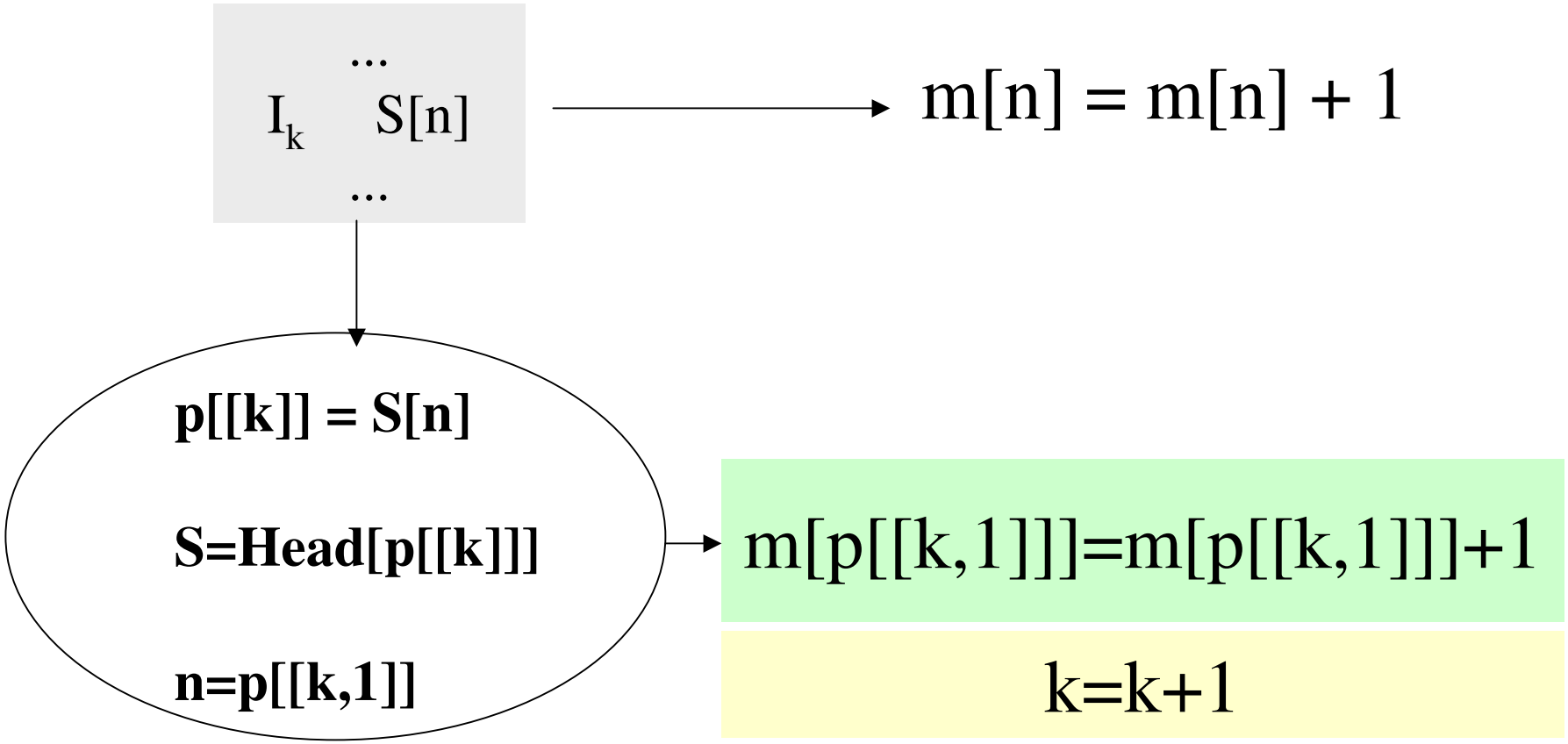

```
r = r + 1
];
```

< Executa Zero >

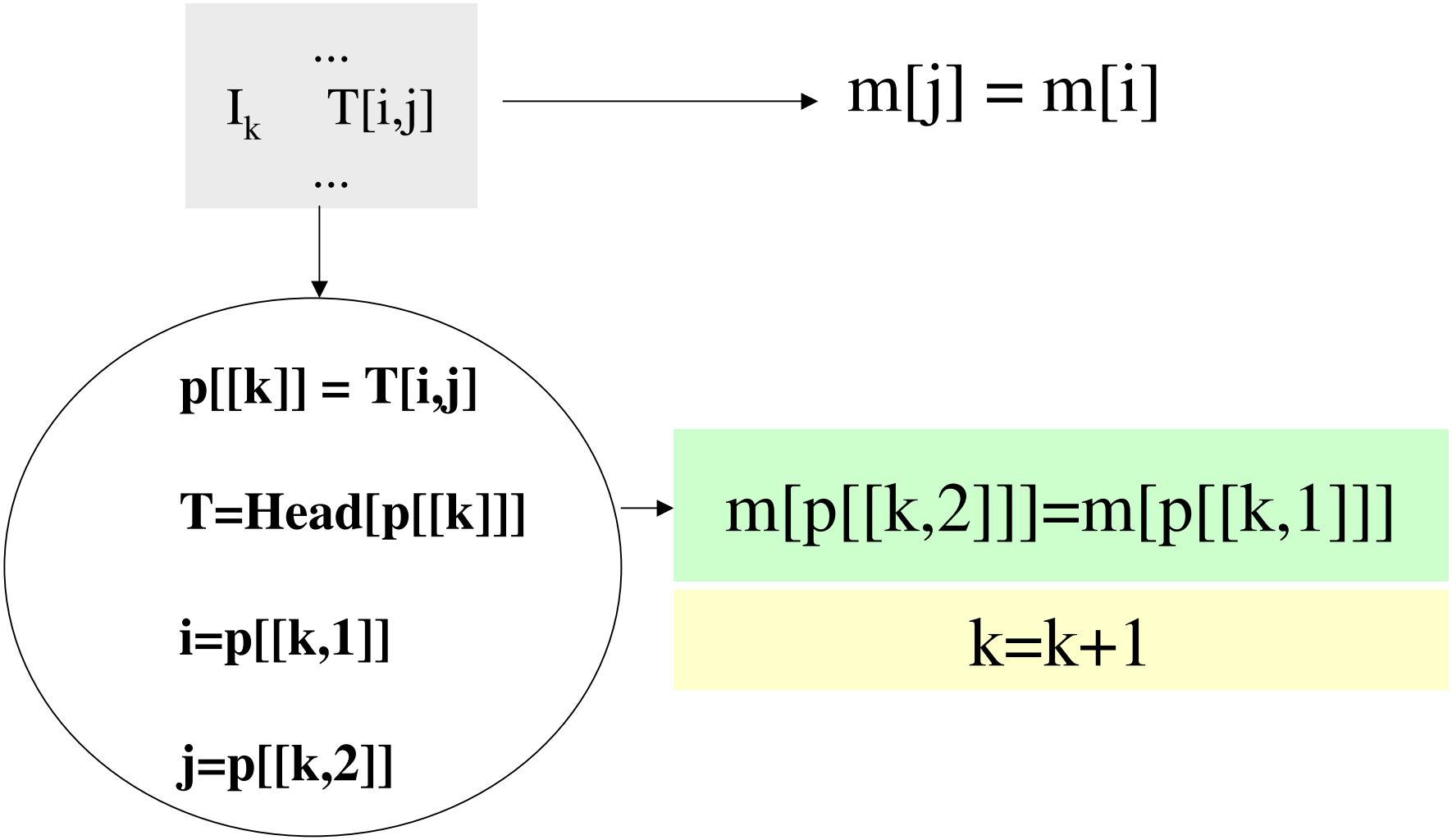


```
Switch[Head[p[[k]]],

  Z, m[p[[k,1]]]=0; k=k+1,
  S, < executa Sucessor >,
  T, < executa Transferência >,
  J, < executa Salto >

  ]
```

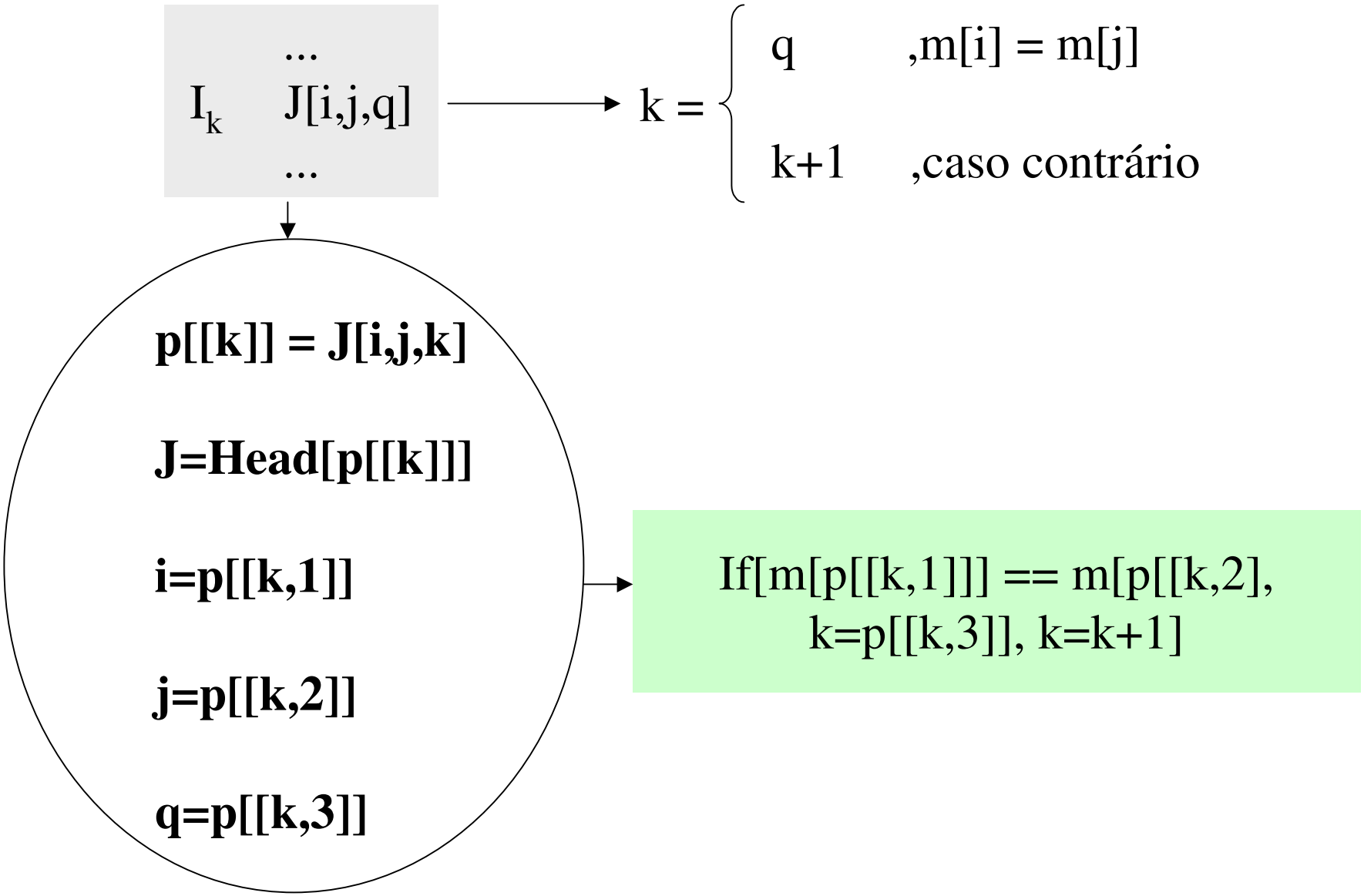
< Executa Sucessor >



< Executa Transferência >



< Executa Salto >



```
Switch[Head[p[[k]]],

        Z, m[p[[k,1]]] = 0; k = k + 1,

        S, m[p[[k,1]]] = m[p[[k,1]]] + 1; k = k + 1,

        T, m[p[[k,2]]] = m[p[[k,1]]]; k = k + 1,

        J, If[m[p[[k,1]]] == m[p[[k,2]]], k = p[[k,3]], k = k + 1]

 ];
```

Devolução do resultado

Resultado

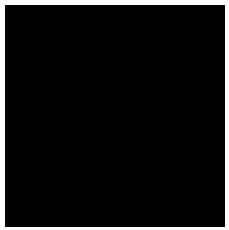
- m[1] se o programa foi executado até ao fim
- mensagem a dizer que o programa não terminou ao fim do número máximo de passos estabelecido no caso contrário.

Sabemos que o programa terminou se no final for **k > s**

Instrução a executar de seguida

Número de instruções do programa

O programa *executaURM* completo



ExecutaURM.nb

```
executaURM[p_, args__, i_] :=
Module[{argsList, n, r, m, k, s},
(* ----CONFIGURAÇÃO INICIAL----------*)
argsList = List[args];
n = Length[argsList];
r = 1;
While[r <= n,
m[r] = argsList[[r]];
r = r + 1
];
m[_] = 0;
(* ----EXECUÇÃO DO PROGRAMA URM----------*)
s = Length[p];
k = 1;
r = 0;
While[k <= s && r <= i,
Switch[Head[p[[k]]],
Z, m[p[[k, 1]]] = 0; k = k + 1,
S, m[p[[k, 1]]] = m[p[[k, 1]]] + 1; k = k + 1,
T, m[p[[k, 2]]] = m[p[[k, 1]]]; k = k + 1,
J, If[m[p[[k, 1]]] == m[p[[k, 2]]], k = p[[k, 3]], k = k + 1]
];
r = r + 1
];
(* ----RESULTADO----------*)
If[k > s, m[1], Print["O programa não terminou ao fim de ", i, " passos"]]
];
```

Escolha do número máximo de passos a executar

O valor de ***i*** deve ser escolhido com algum cuidado, tendo em conta não só o programa URM em execução mas também a amplitude dos próprios argumentos.

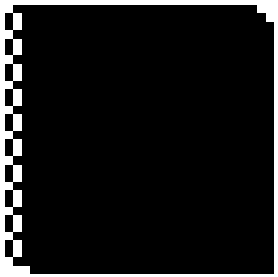
executaURM[maior,7,8,100] → **8**

executaURM[maior,260,150,100]

↓

O programa não terminou ao fim de 100 passos.

executaURM[maior,260,150,900] → **260**



Passos.nb

Exemplo 1:

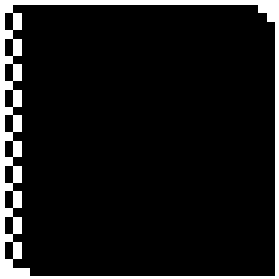
$$f : IN_o \rightarrow IN_o \quad \text{com} \quad f(x) = 5 \quad \forall x \in IN_o$$

Exemplo1.nb

I_1 Z(1)
I_2 S(1)
I_3 S(1)
I_4 S(1)
I_5 S(1)
I_6 S(1)

Exemplo 2:

$$f : IN_o \rightarrow IN_o \quad \text{com} \quad f(x) = x \text{ Div } 2$$

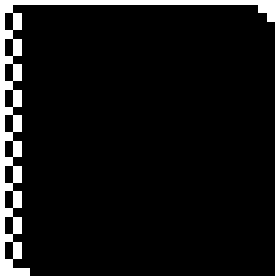


Exemplo2.nb

I_1 J(1,2,7)
I_2 S(2)
I_3 J(1,2,7)
I_4 S(3)
I_5 S(2)
I_6 J(1,1,1)
I_7 T(3,1)

Exemplo 3:

$$f : IN_o \rightarrow IN_o \quad \text{com} \quad f(x) = x \text{ Mod } 2$$

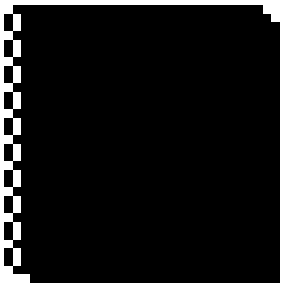


Exemplo3.nb

I_1 J(1,2,9)
I_2 S(2)
I_3 J(1,2,6)
I_4 S(2)
I_5 J(1,1,1)
I_6 Z(1)
I_7 S(1)
I_8 J(1,1,10)
I_9 Z(1)

Exemplo 4:

$$g : IN_o \to IN_o \text{ tal que } g(x) = \begin{cases} 1 & , \ x = 0 \\ x - 2 & , \ x \text{ par} \neq 0 \\ x + 2 & , \ x \text{ impar} \end{cases}$$



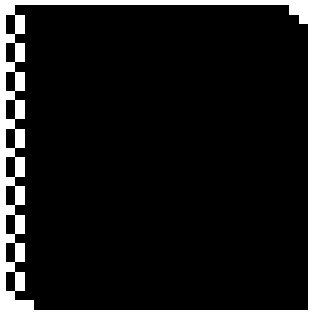
Exemplo4.nb

I_1 J(1,2,16)
I_2 S(2)
I_3 J(1,2,15)
I_4 S(2)
I_5 J(1,2,7)
I_6 J(1,1,2)
I_7 S(3)
I_8 S(3)

I_9 J(1,3,13)
I_{10} S(4)
I_{11} S(3)
I_{12} J(1,1,9)
I_{13} T(4,1)
I_{14} J(1,1,17)
I_{15} S(1)
I_{16} S(1)

Exemplo 5:

$$h : IN_o^3 \rightarrow IN_o \quad \text{tal que} \quad h(x, y, z) = x + y + z$$



Exemplo5.nb

I_1 J(2,4,5)
I_2 S(1)
I_3 S(4)
I_4 J(1,1,1)
I_5 Z(4)
I_6 J(3,4,10)
I_7 S(1)
I_8 S(4)
I_9 J(1,1,6)

Conclusão:

- Pretendeu-se com esta lição mostrar que é perfeitamente possível construir com os alunos, no decurso de uma aula prática, recorrendo ao *Mathematica*, o emulador da máquina URM.

- Ao frequentarem a aula os alunos acabam por interiorizar melhor a estrutura e programação da máquina URM.

- Relembrar aos alunos que o *Mathematica* é uma excelente ferramenta de trabalho.

- Aumentar a prática de programação.