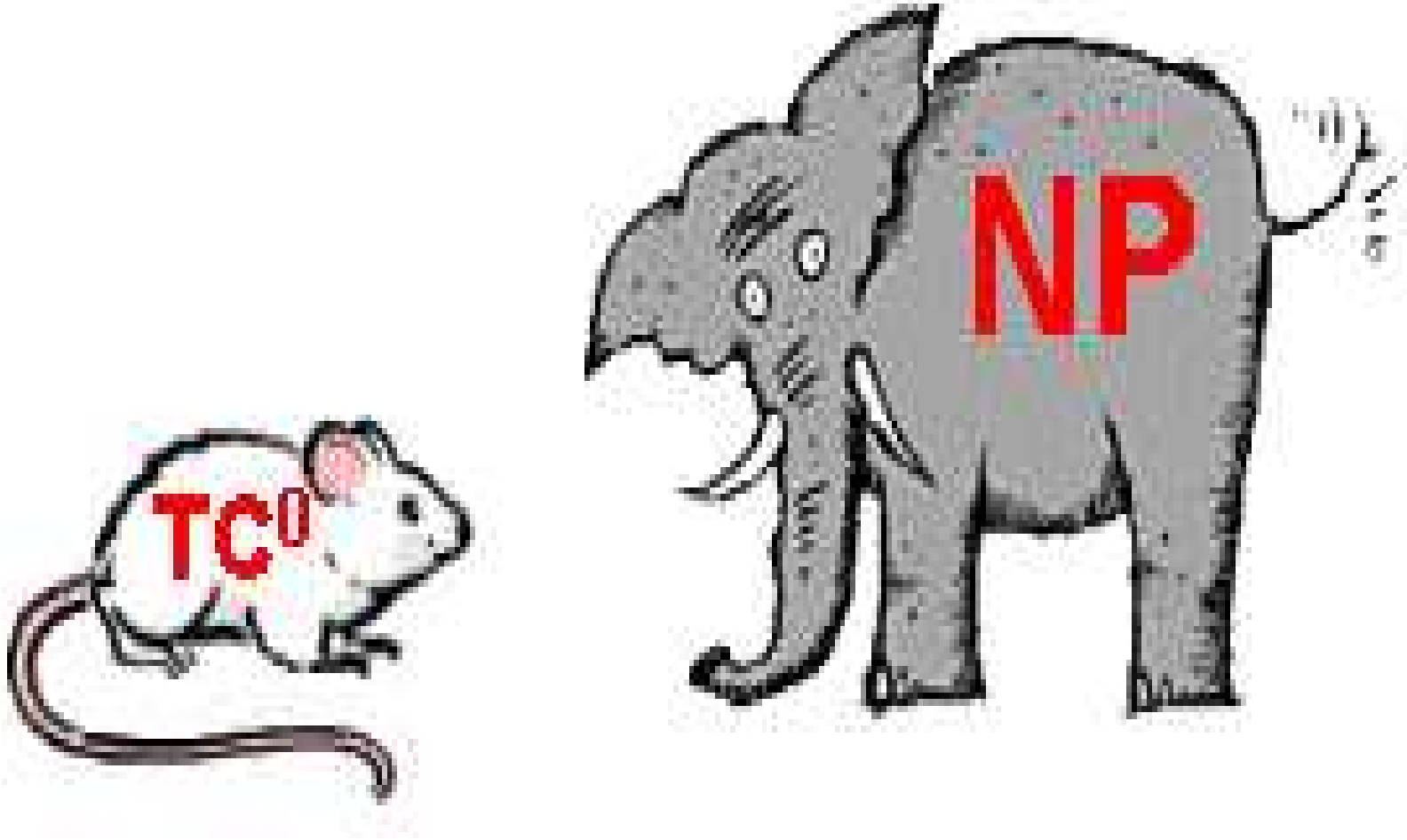
MAC5722

Complexidade Computacional

MAC5722 Complexidade Computacional



"Qual é o seu problema?"

http://qwiki.stanford.edu/wiki/Complexity_Zoo

Administração

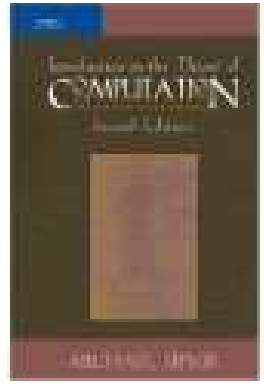
Professores: Zé Augusto e Coelho

Página da disciplina: `http://paca.ime.usp.br/`

- cadastro na paca
- código de inscrição: complexidade2010
- apresentação
- critério: provas, listas de exercícios, plágio...
- fórum

Texto principal

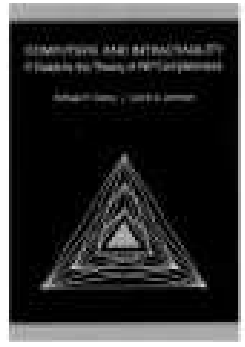
Si = Michael Sipser,
Introduction to the Theory of Computation



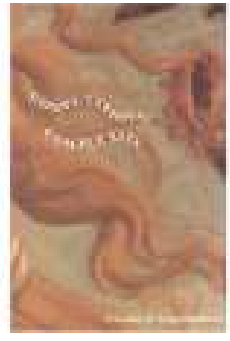
Si = Michael Sipser,
Introdução à Teoria da Computação
Tradução

Outros livros

GJ = Michael Randolph Garey e David Stifler Johnson,
Computers and Intractability: A Guide to the Theory of NP-completeness



Pa = Christos Harilaos Papadimitriou,
Computational complexity



Pausa para os nossos comerciais

Recepção dos alunos novos de mestrado e doutorado

Na sexta-feira, dia 12/3, às 12h00, haverá a usual recepção dos alunos de pós.

O evento ocorre na Anfiteatro Antonio Gilioli.
Haverá sanduíche de metro e refrigerantes financiados pelo departamento.

A organização está sendo tocada pelos representantes discentes da CCP Jihan e Alexandre.

Pré-requisitos

Prática em programação é bem-vinda

É recomendável alguma experiência em análise assintótica de algoritmos

- MAC0338 Análise de algoritmos
- MAC5711 Análise de algoritmos

ou algum conhecimento em modelos computacionais

- MAC0414 Linguagens formais e autômatos
- MAC4722 Linguagens, autômatos e computabilidade

AULA 0

Autômatos, Computabilidade e Complexidade

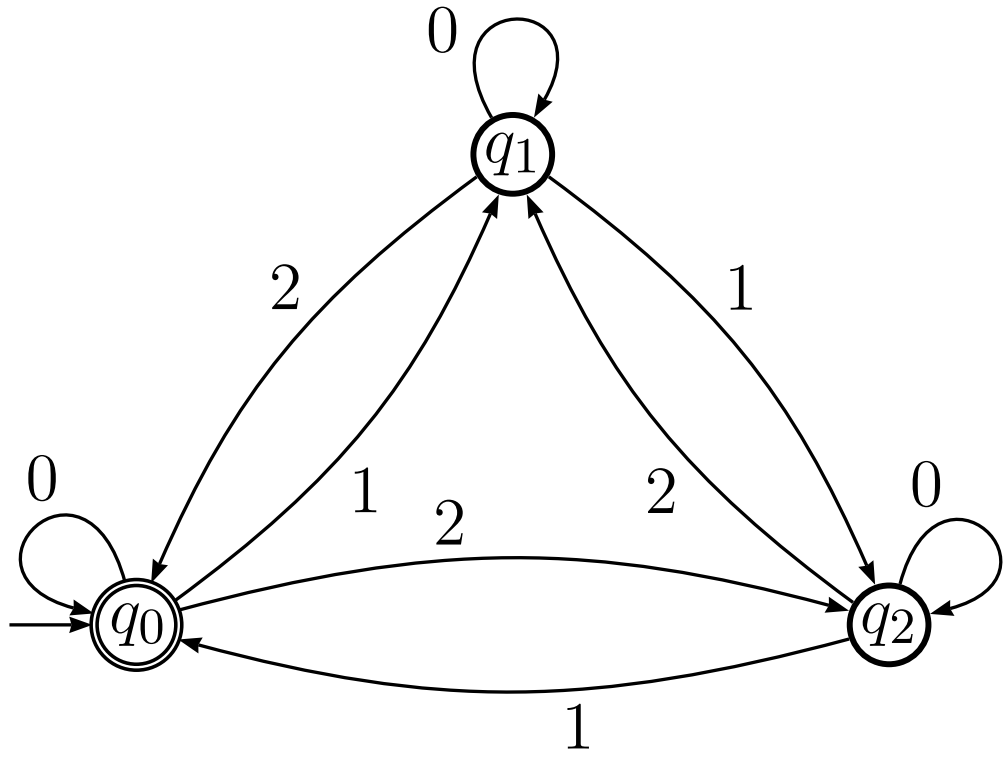
MS 0.1, GJ 1.1

Questão central

Quais são as capacidades e limitações fundamentais de computadores?

Autômatos

Modelo computacional usado em processamento de texto, compiladores . . .
FINITE-AUTOMATON-MATCHER e KMP-MATCHER (CLRS)



MAC4722 Linguagens, Autômatos e Computabilidade

Computabilidade

O que computadores podem fazer?

$10^{\underline{o}}$. problema de Hilbert (1900): projetar um algoritmo para decidir se um dado polinômio de coeficientes inteiros tem uma raiz inteira.

Exemplos:

- $10x^2y + 10yz + 5z$ tem raiz inteira $x = 3$, $y = 1$ e $z = -6$
- $x^2 - 5$ não tem raiz inteira.

Como definir *algoritmo*?

MAC4722 Linguagens, Autômatos e Computabilidade

Complexidade

Pergunta central:

O que faz alguns problemas computacionalmente difíceis e outros fáceis?

O que computadores podem fazer eficientemente?

Complexidade

Pergunta central:

O que faz alguns problemas computacionalmente difíceis e outros fáceis?

O que computadores podem fazer eficientemente?

Não se sabe a resposta . . .

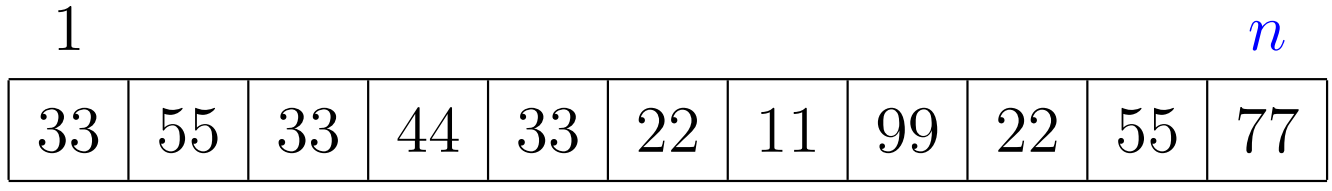
mas há esquemas para se classificar os problemas de acordo com sua dificuldade.

Ordenação

$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique crescente.

Entra:



Ordenação

$A[1\,..\,n]$ é crescente se $A[1] \leq \cdots \leq A[n]$.

Problema: Rearranjar um vetor $A[1\,..\,n]$ de modo que ele fique crescente.

Entra:

1										n
33	55	33	44	33	22	11	99	22	55	77

Sai:

1										n
11	22	22	33	33	33	44	55	55	77	99

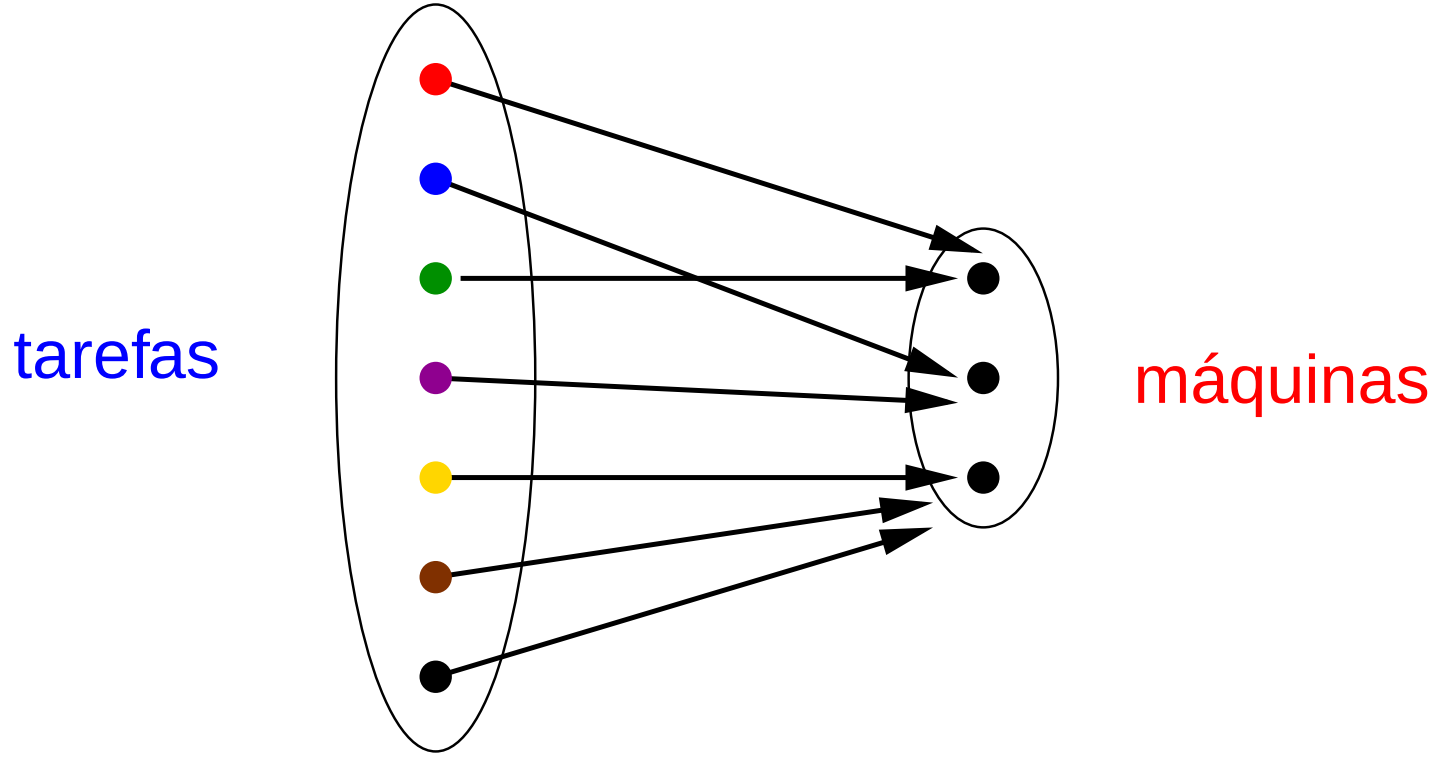
Escalonamento de máquinas idênticas

Dados: m máquinas

t tarefas

duração $d[i]$ da tarefa i $\quad (i = 1, \ldots, t)$

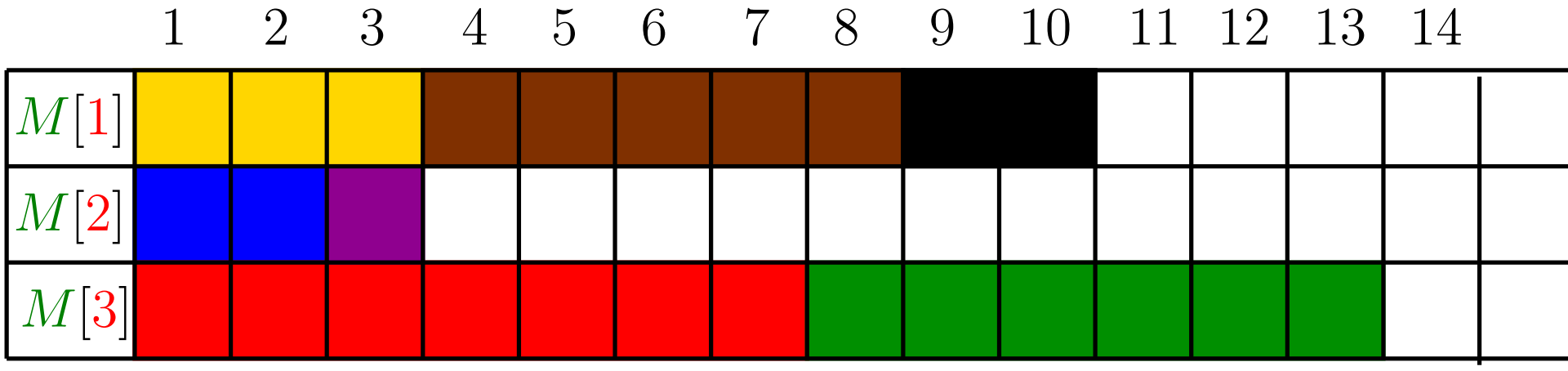
um **escalonamento** é uma partição $\{M[1], \ldots, M[m]\}$ de $\{1, \ldots, t\}$



Exemplo 1

$m = 3 \quad t = 7$

$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2

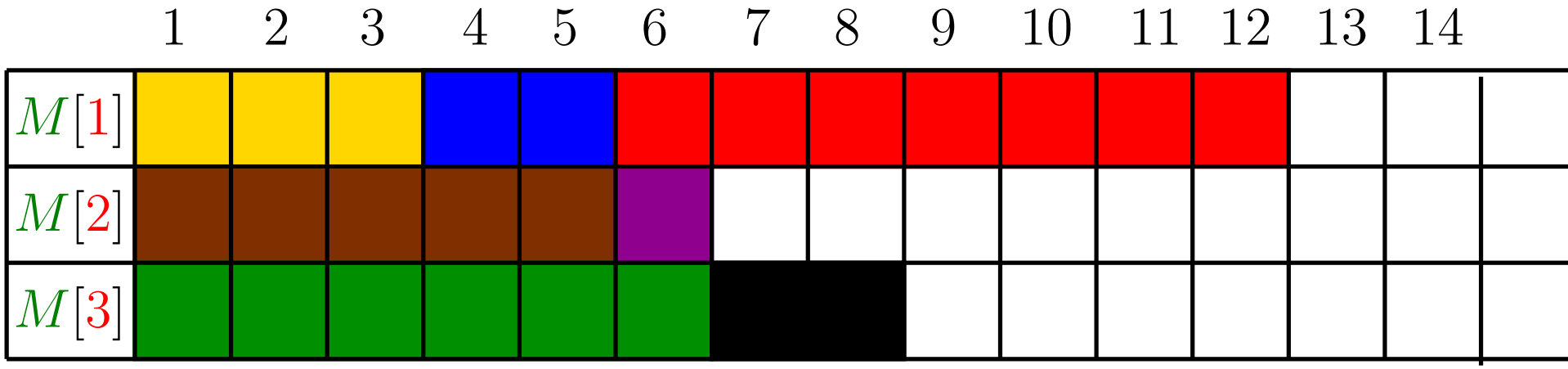


$\{\{1, 4, 7\}, \{2, 5\}, \{3, 6\}\} \Rightarrow$ Tempo de conclusão $= 13$

Exemplo 2

$m = 3 \quad t = 7$

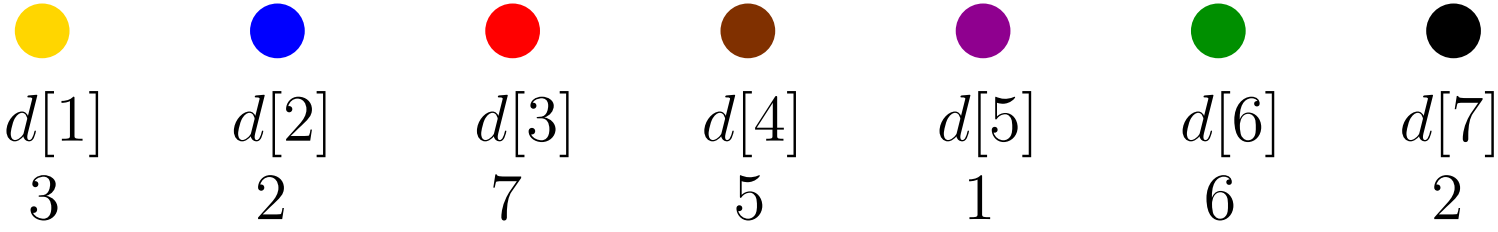
$d[1]$	$d[2]$	$d[3]$	$d[4]$	$d[5]$	$d[6]$	$d[7]$
3	2	7	5	1	6	2

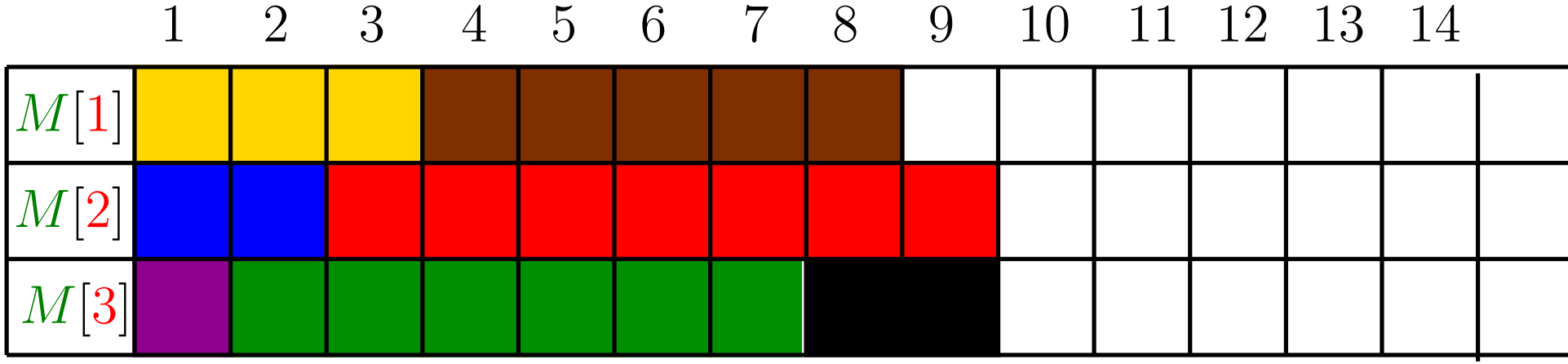


$\{\{1, 2, 3\}, \{4, 5\}, \{6, 7\}\} \Rightarrow$ Tempo de conclusão $= 12$

Problema

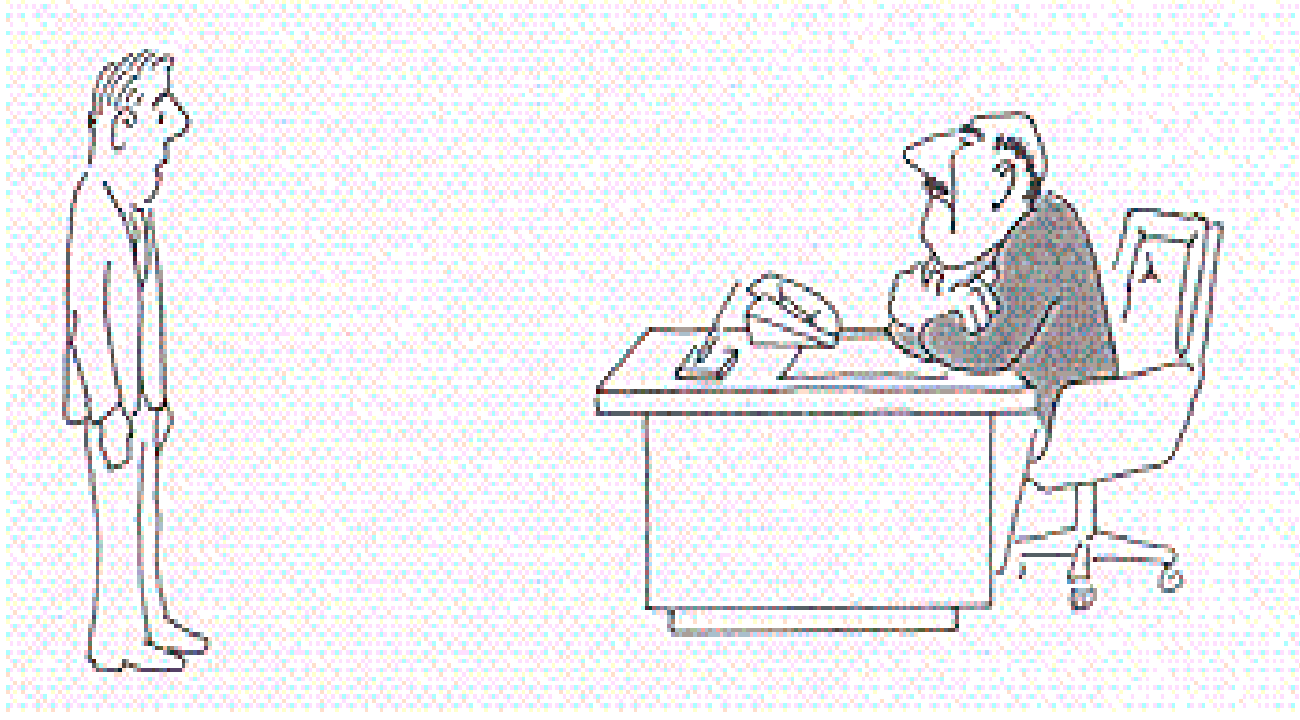
Encontrar um escalonamento com tempo de conclusão mínimo.





$\{\{1,4\},\{2,3\},\{5,6,7\}\} \Rightarrow$ Tempo de conclusão $= 9$

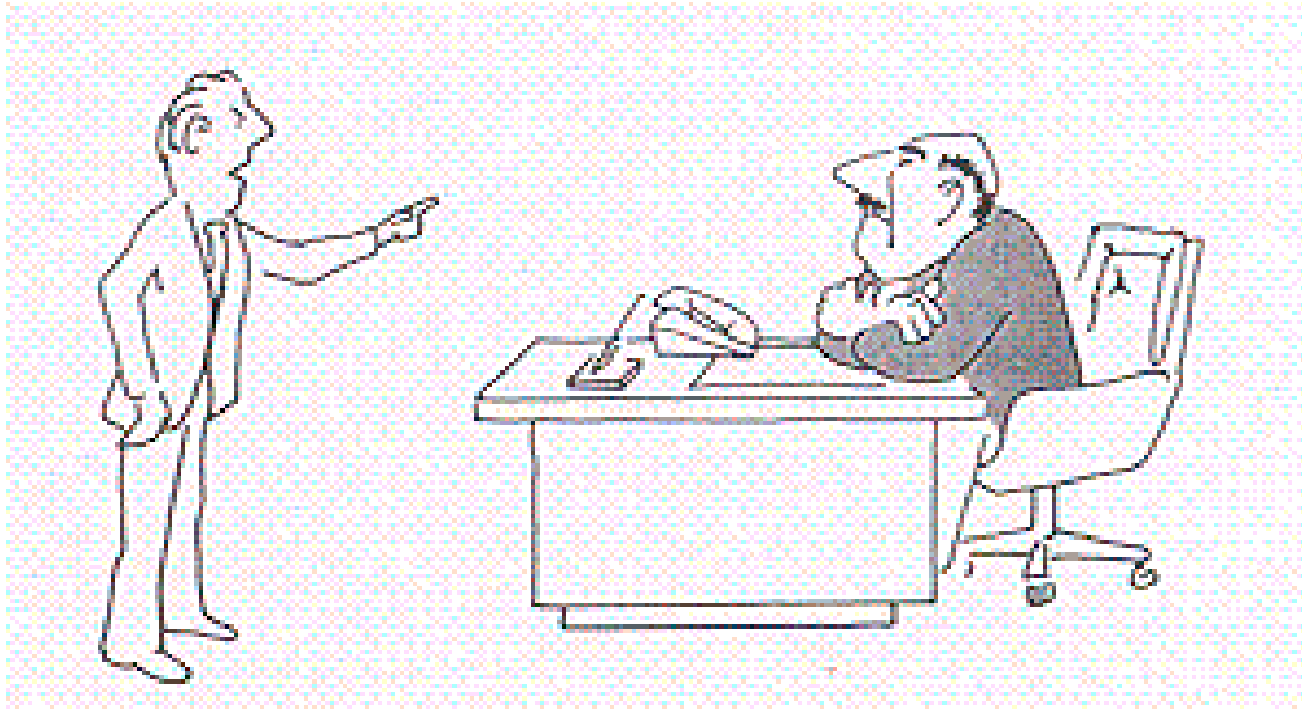
Orientador



“I can’t find an efficient algorithm, I guess I’m just too dumb.”

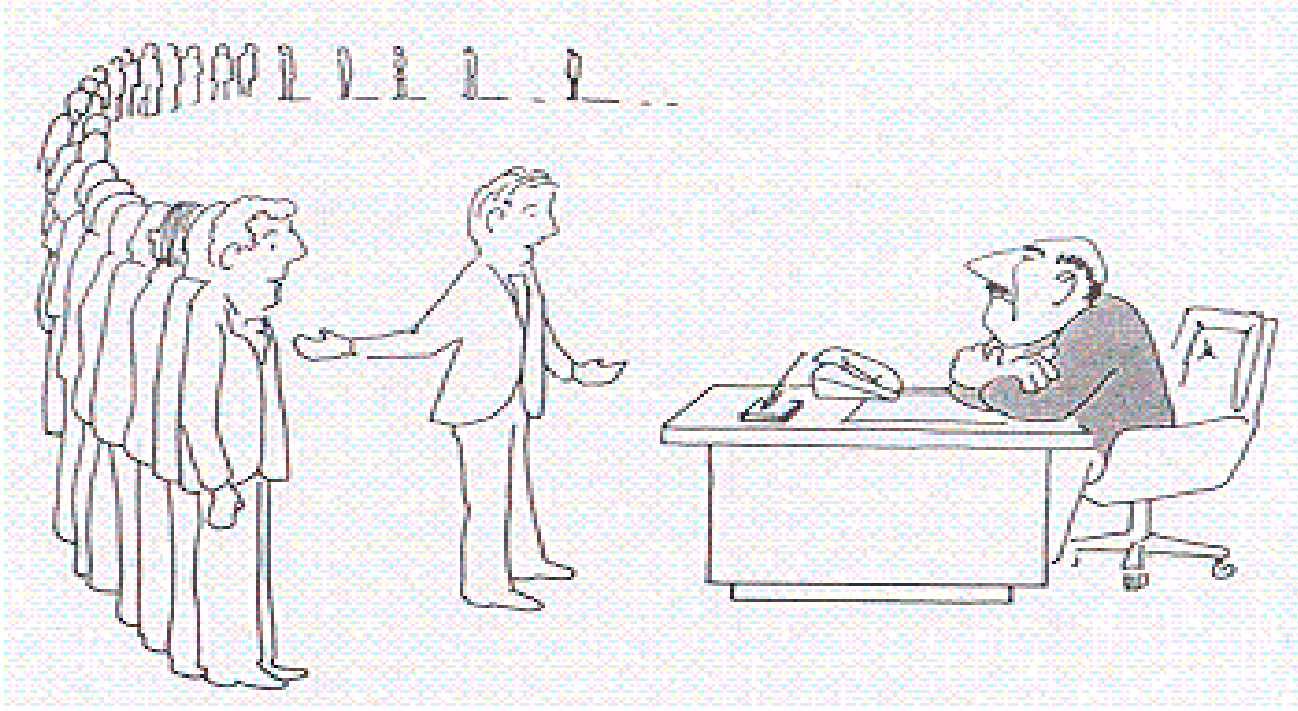
GJ 1.1

Orientador



“I can’t find an efficient algorithm, because no such algorithm is possible.”

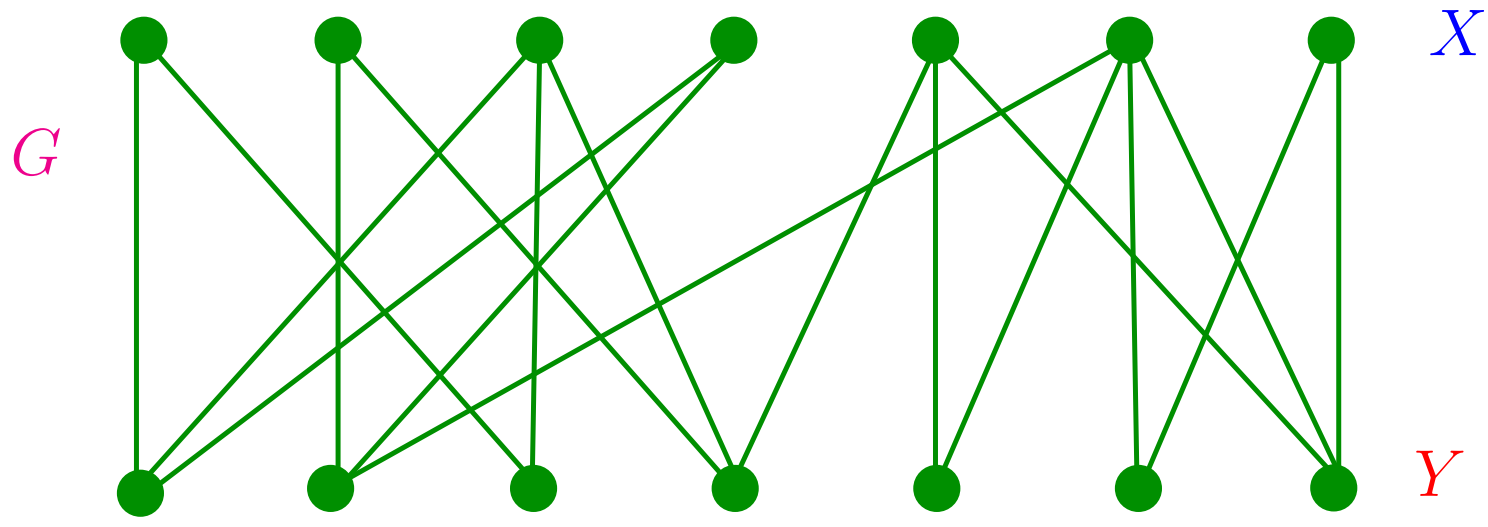
Orientador



“I can’t find an efficient algorithm, but neither can all these famous people.”

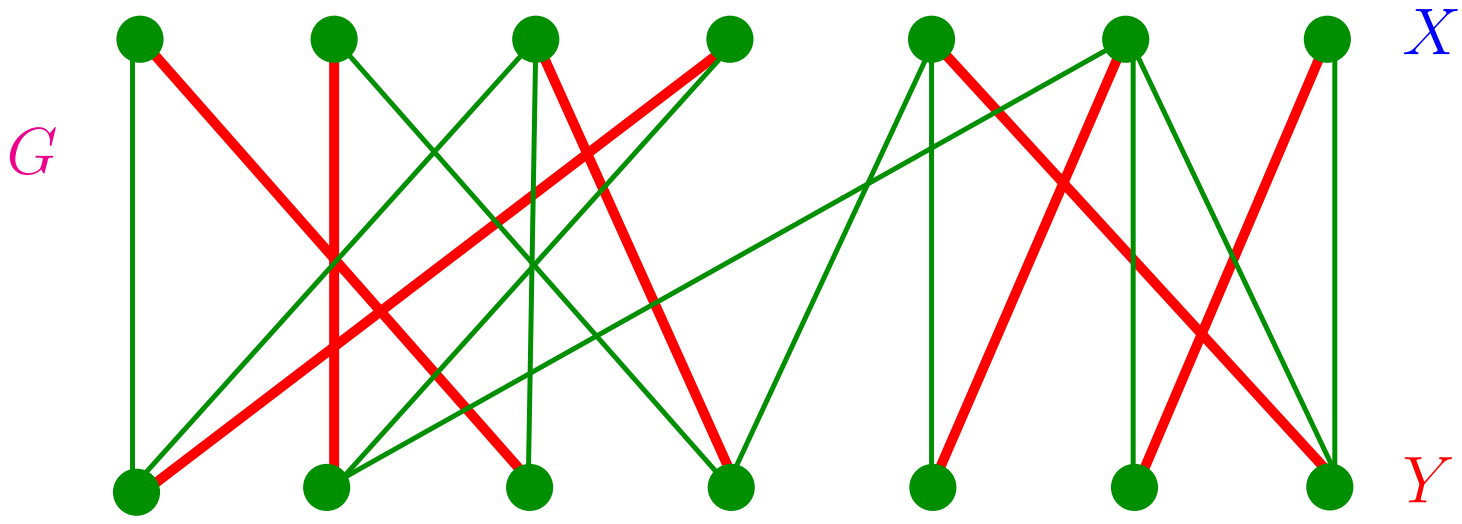
Emparelhamentos

Problema: Dado um grafo bipartido encontrar um emparelhamento perfeito.



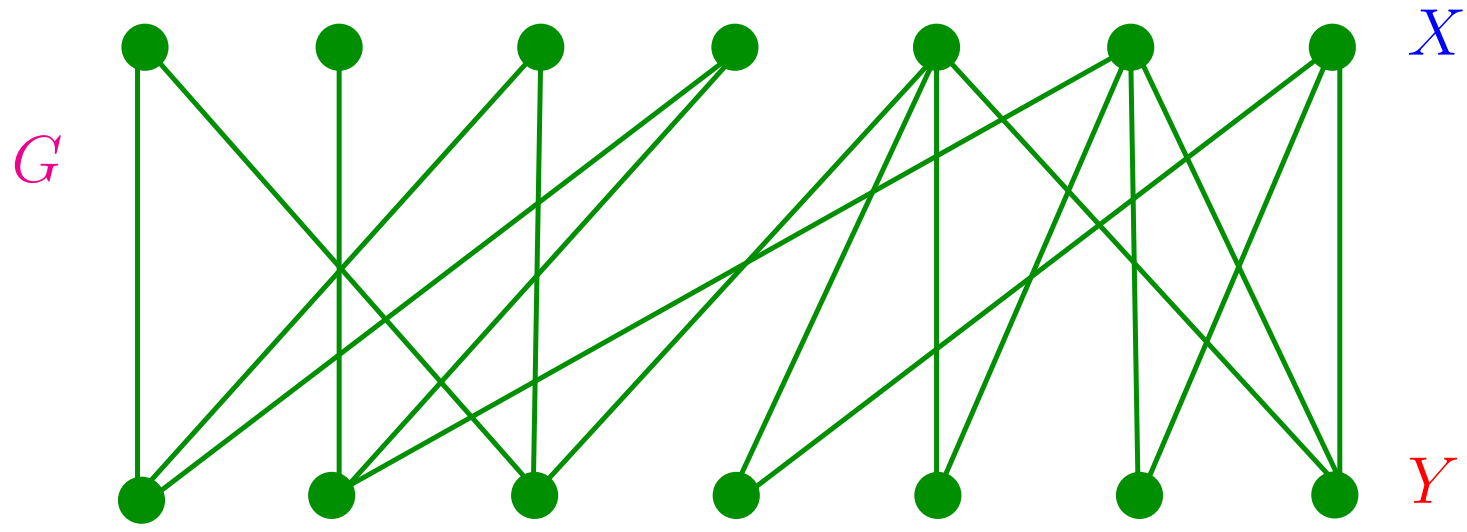
Emparelhamentos

Problema: Dado um grafo bipartido encontrar um emparelhamento perfeito.



Emparelhamentos

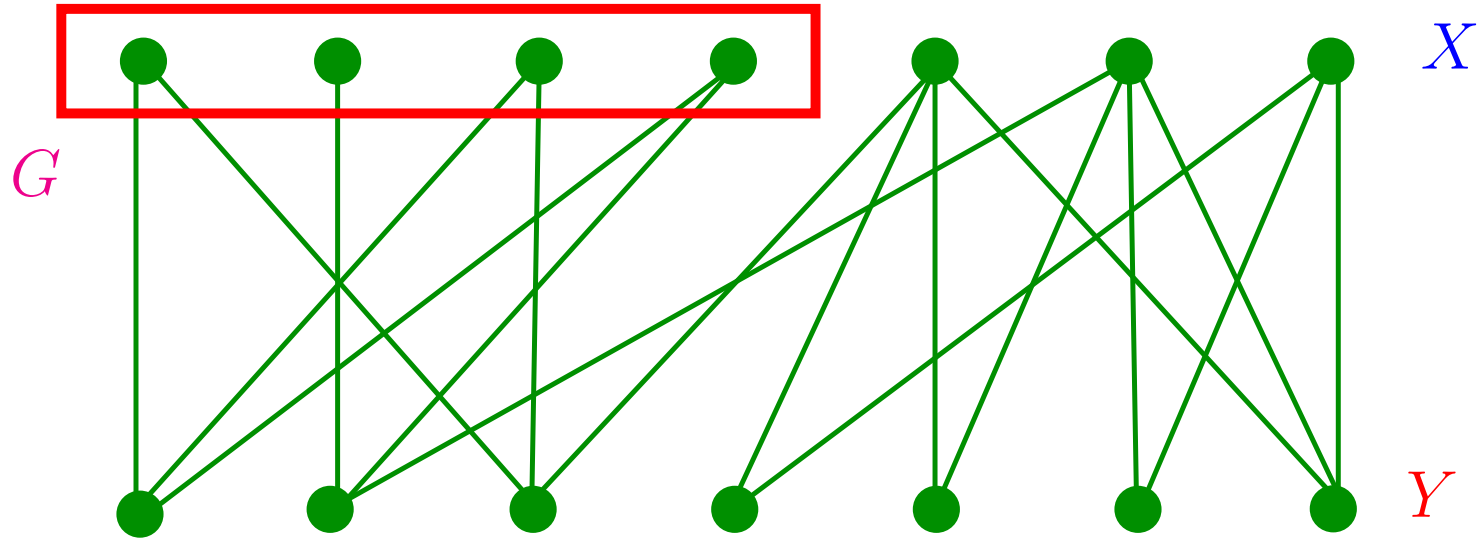
Problema: Dado um grafo bipartido encontrar um emparelhamento perfeito.



NÃO existe! Certificado?

Emparelhamentos

Problema: Dado um grafo bipartido encontrar um emparelhamento bipartido.



NÃO existe! Certificado: $S \subseteq X$ tal que $|S| > |\text{vizinhos}(S)|$.
Teorema de Hall: G tem um emparelhamento perfeito se e somente se

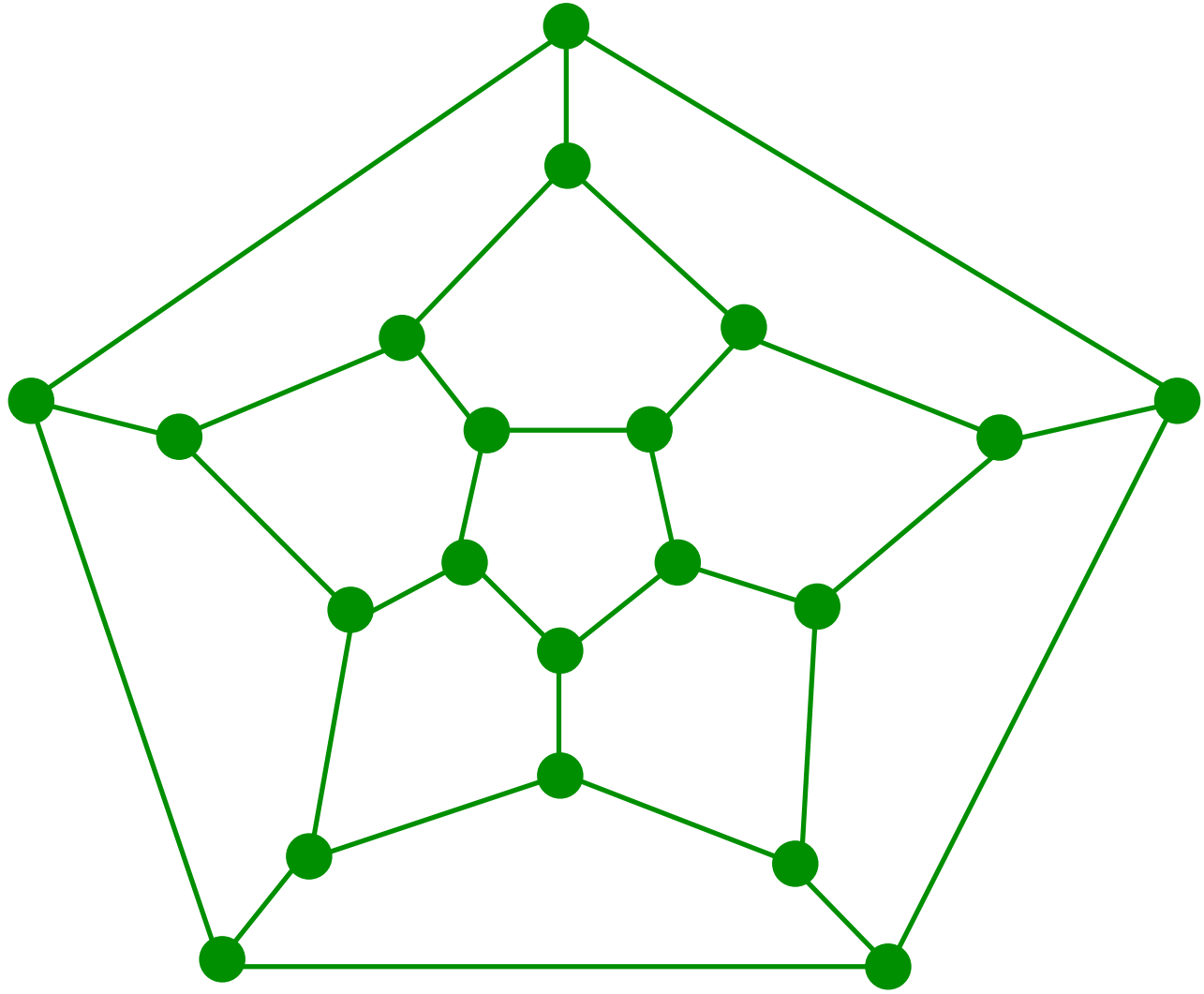
$$|S| \leq |\text{vizinhos}(S)|, \quad \text{para todo } S \subseteq X.$$

Arthur e Merlin

Grafos hamiltonianos

Problema: Dado um grafo encontrar um ciclo hamiltoniano.

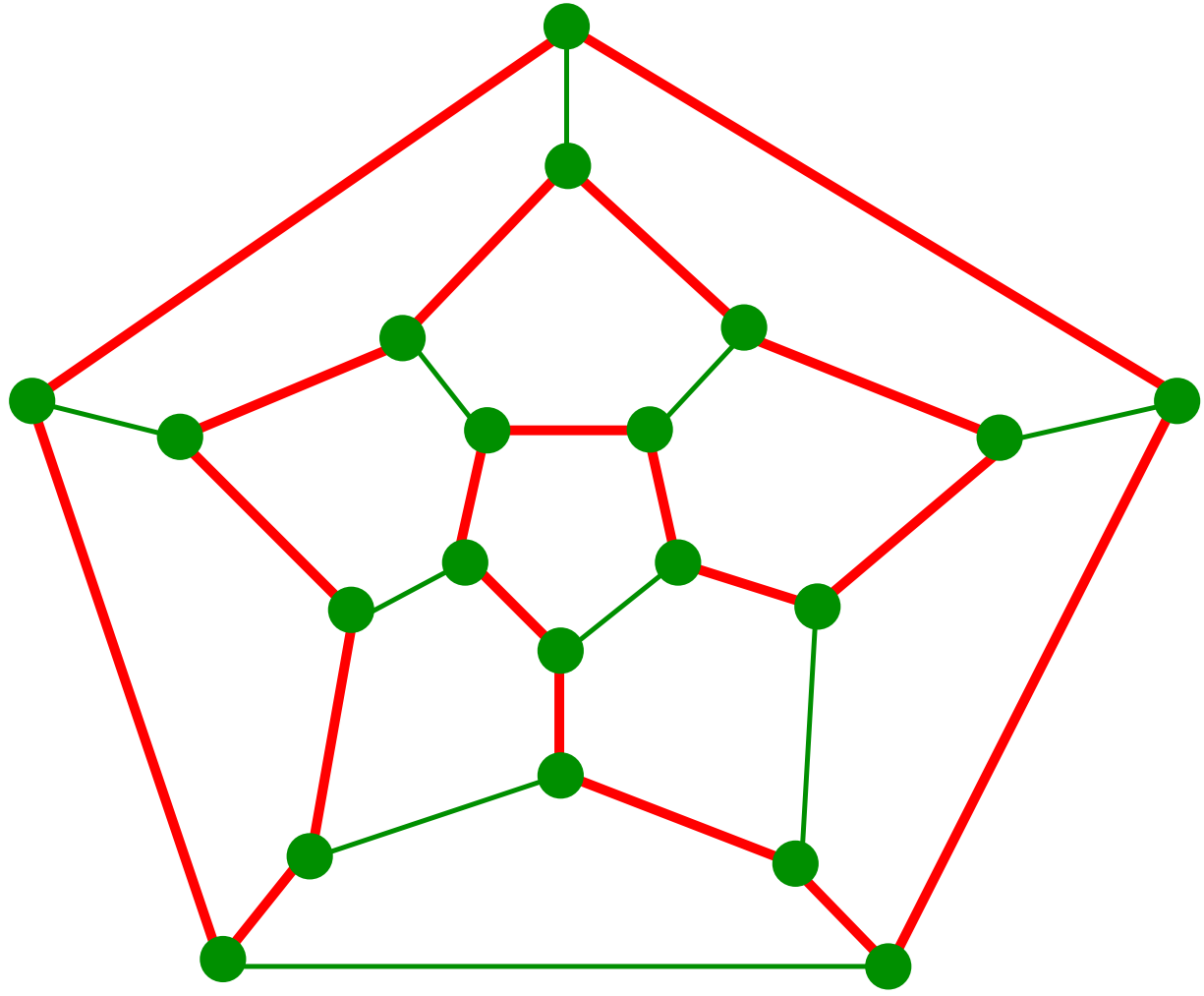
G



Grafos hamiltonianos

Problema: Dado um grafo encontrar um ciclo hamiltoniano.

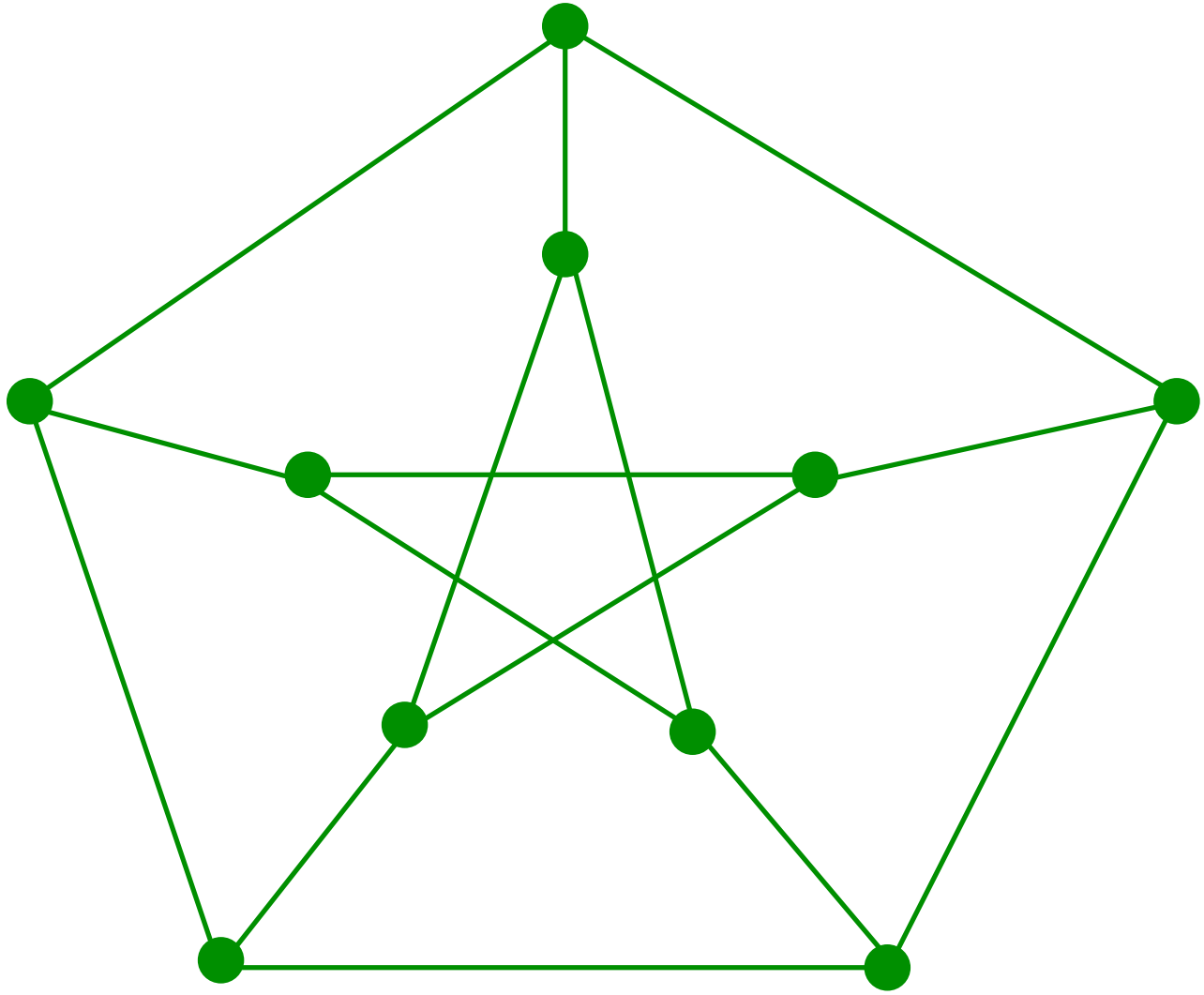
G



Grafos hamiltonianos

Problema: Dado um grafo encontrar um ciclo hamiltoniano.

G



NÃO existe! Certificado? Hmmm . . .

Composto

Problema: Dado um número inteiros positivo n, encontrar números inteiros $k > 1$ e $m > 1$ tais que

$$n = k \times m.$$

Exemplo:

- $n = 44003111$, $k = 4651$ $m = 9461$
- $n = 1544824079$, $k = 37493$ $m = 41203$ fazem o serviço
- $n = 27644437$, não existem n e m

Solução café-com-leite

Considere o seguinte algoritmo que recebe um número natural n e devolve 0 se n é primo ou um fator não-trivial k de n.

```
FATOR (n)
1    para k ← 2 até n − 1 faça
2         se resto(n/k) = 0
3              devolva k
4    devolva 0
```

O consumo de tempo do algoritmo FATOR no pior caso é proporcional a n.

Solução café-com-leite

Considere o seguinte algoritmo que recebe um número natural n e devolve 0 se n é primo ou um fator não-trivial k de n.

```
FATOR (n)
1   para k ← 2 até n − 1 faça
2       se resto(n/k) = 0
3           devolva k
4   devolva 0
```

O consumo de tempo do algoritmo FATOR no pior caso é proporcional a n.

Não é considerado eficiente.

Que bagunça!

Vimos exemplo de problemas:

- não podem ser resolvidos por computador (Hilbert)
- sabemos resolver *eficientemente* (ordenação)
- não sabemos resolver *eficientemente* (escalonamento)
- sabemos verificar a resposta *eficientemente* (emparelhamento)
- não sabemos verificar a resposta *eficientemente* (hamiltoniano)
- sabemos verificar a resposta *eficientemente*, mas não sabemos encontrar a resposta *eficientemente* (composto)

Intratabilidade

A descoberta da *intratabilidade* de um problema é apenas o começo do trabalho.

Busca por *algoritmos eficientes* ganha baixa prioridade.

Podemos nos concentrar em objetivos menos pretenciosos. Encontrar algoritmos:

- *eficientes* para casos particulares;
- *freqüentemente* eficientes;
- *eficientes* que encontram *bons candidatos* a solução;
- . . .

Intratabilidade

A descoberta da *intratabilidade* de um problema é apenas o começo do trabalho.

Busca por *algoritmos eficientes* ganha baixa prioridade.

Podemos nos concentrar em objetivos menos pretenciosos. Encontrar algoritmos:

- *eficientes* para casos particulares;
- *freqüentemente* eficientes;
- *eficientes* que encontram *bons candidatos* a solução;
- . . .

Problema *intratáveis* são os preferidos pela galera de criptografia (sistema criptográfico RSA, composto, Pa 12).

MAC5722

MAC5722 Complexidade Computacional:

- é uma disciplina básica e tradicional em teoria da computação
- busca a compreensão das limitações do modelo computacional e da dificuldade inerente para resolver problemas computacionais algoritmicamente
- classifica problemas computacionais baseado na quantidade de recursos (tempo e espaço) necessários para resolvê-los e investiga a relação entre estas classes.

Principais tópicos

- máquinas de Turing
- variantes de máquinas de Turing
- definição de algoritmos
- tese de Church-Turing
- complexidade de tempo
- a classe P.
- as classes NP e coNP
- NP-completude
- problemas NP-completos
- complexidade de espaço
- teorema de Savitch
- classe PSPACE
- as classes L e NL
- NL-completude
- NL e coNL.

AULA 1

Cadeias e linguagens

MS 0.2

Cadeias

Para resolver um problema usando um computador é necessário descrever os dados do problema através de uma seqüência de símbolos retirados de algum **alfabeto**.

Este alfabeto pode ser, por exemplo, o conjunto de símbolos ASCII ou o conjunto $\{0, 1\}$.

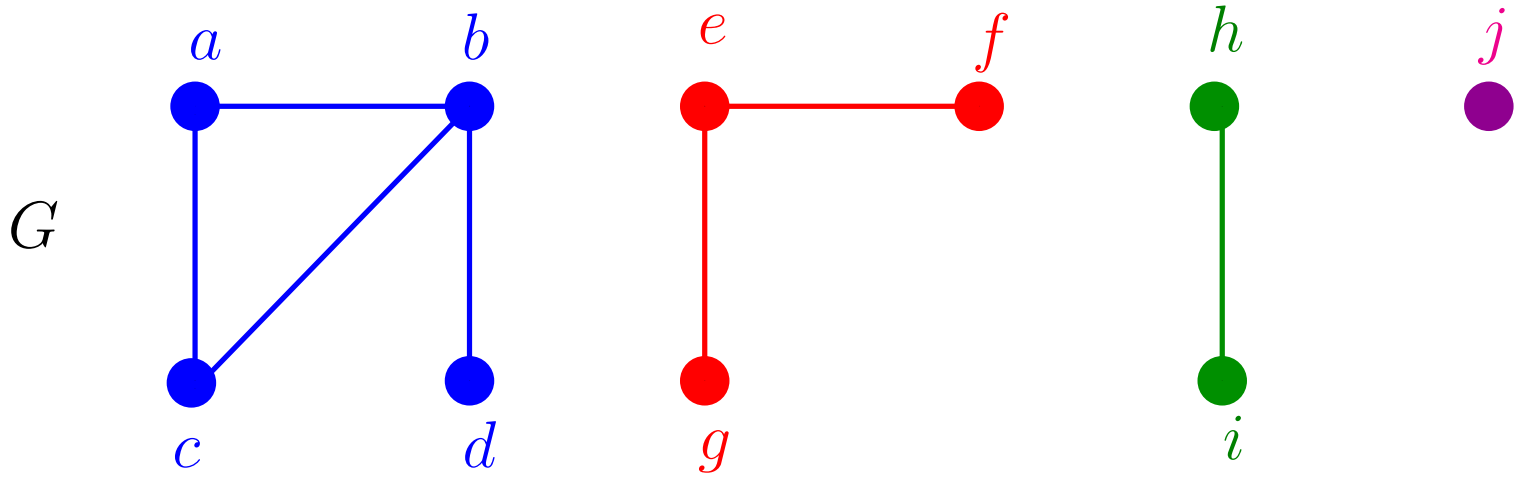
Qualquer seqüência dos elementos de um alfabeto é chamada de uma **cadeia**.

Não é difícil codificar objetos tais como racionais, vetores, matrizes, grafos e funções como cadeias.

O **comprimento** de uma cadeia w, denotado por $|w|$ é o número de símbolos usados em w, contando multiplicidades. O comprimento do racional '$123/567$' é 7.

Exemplo 1

Grafo



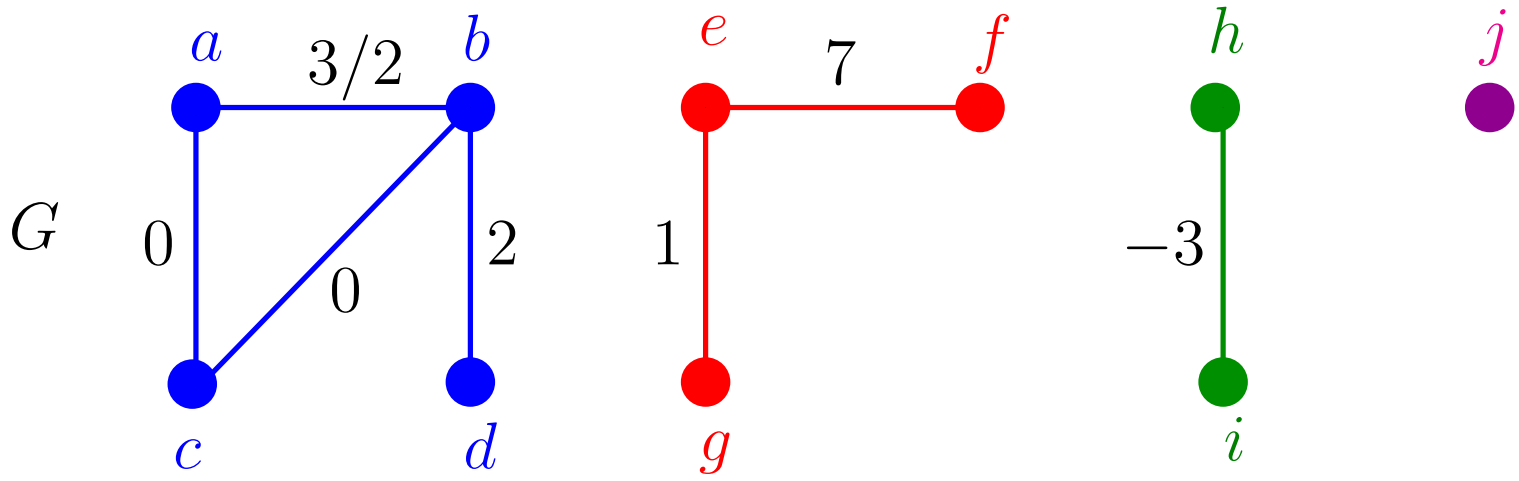
Cadeia:

$$(\{a, b, c, d, e, f, g, h, i, j\}, \{\{bd\}, \{eg\}, \{ac\}, \{hi\}, \{ab\}, \{ef\}, \{bc\}\}$$

Comprimento da cadeia: 59

Exemplo 2

Função



Cadeia:

$$((\{bd\}, 2), (\{eg\}, 1), (\{ac\}, 0), (\{hi\}, -3), (\{ab\}, 3/2), (\{ef\}, 7), (\{bc\}, 0))$$

Comprimento da cadeia: 67

Alfabeto, símbolos e cadeias

Um **alfabeto** é conjunto finito não vazio.

Os elementos de um alfabeto são chamados de **símbolos**.

Exemplos:

- $\Sigma_1 = \{0, 1\}$
- $\Sigma_2 = \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$

Alfabeto, símbolos e cadeias

Um **alfabeto** é conjunto finito não vazio.

Os elementos de um alfabeto são chamados de **símbolos**.

Exemplos:

- $\Sigma_1 = \{0, 1\}$
- $\Sigma_2 = \{\mathtt{a}, \mathtt{b}, \ldots, \mathtt{z}\}$

Uma **cadeia** sobre um alfabeto é uma sequência finita de símbolos do alfabeto.

Exemplos:

- $0\,1\,0\,0\,1$ é uma cadeia sobre Σ_1
- $\mathtt{abracadabra}$ é uma cadeia sobre Σ_2.

Comprimento de cadeias

O **comprimento** de uma cadeia w sobre um alfabeto Σ, denotado por $|w|$, é o número de símbolos em w, contando multiplicidades.

Exemplos:

- $0\,1\,0\,0\,1$ tem comprimento 5
- `abracadabra` tem comprimento 11

A **cadeia vazia** é denotada por ε e tem comprimento **zero**.

Se w tem comprimento n escrevemos

$$w = w_1 w_2 \ldots w_n,$$

onde cada w_i está em Σ.

Concatenação de cadeias

A **concatenação** das cadeias x e y é a cadeia xy.

Exemplo: se $x = \texttt{abra}$ e $y = \texttt{cadabra}$, então

$$xy = \texttt{abracadabra}$$

Para k é um inteiro, denotamos x^k a concatenação de x com ele mesmo $k - 1$ vezes.

Exemplo: se $x = \texttt{abra}$, então

$$x^4 = \texttt{abraabraabraabra}$$

Subcadeias

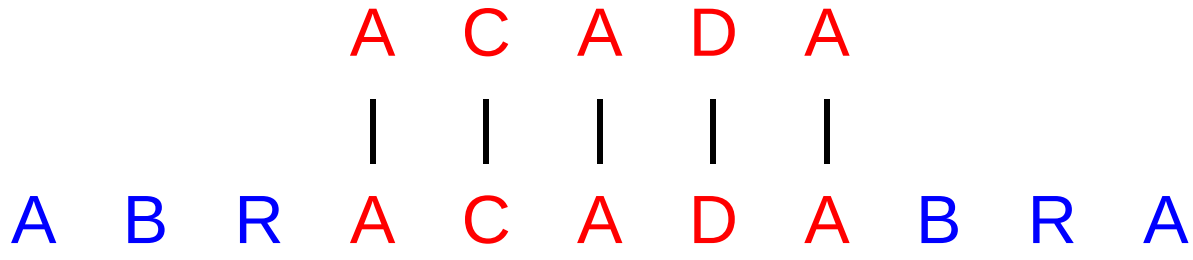
$z_1 \ldots z_k$ é **subcadeia** de $x_1 \ldots x_m$
se existe um índice i tal que

$$z_1 = x_i \quad \ldots \quad z_k = x_{i+k-1}$$

EXEMPLOS:

$5\,9\,2\,7$ é subseqüência de $9\,5\,5\,9\,2\,7\,3\,7$

A C A D A é subcadeia de A B R A C A D A B R A



Reverso e ordem lexicográfica

Se $w = w_1\, w_2\, \ldots\, w_n$ o **reverso** de w, denotado por w^R, é

$$w_n\, w_{n-1}\, \ldots\, w_1.$$

A **ordem lexicográfica** de cadeias é a mesma do dicionário, exceto que cadeias menores precedem cadeias maiores.

Exemplo: a ordem lexicográfica das cadeias sobre $\{0, 1\}$ é

$$\varepsilon \quad 0 \quad 1 \quad 00 \quad 01 \quad 10 \quad 11 \quad 000 \quad \ldots$$

Linguagem

Uma **linguagem** é um conjunto de cadeias sobre um alfabeto.

Exemplos:

- conjunto das cadeias que codificam grafos
- conjunto das cadeias que codificam grafos bipartidos que possuem um emparelhamento perfeito
- conjunto das cadeias que codificam grafos hamiltonianos
- conjunto de cadeias da forma $w\#w^R$ para alguma cadeia w sobre Σ
- ...

Operações

Se L e L' são linguagens então

União: $L \cup L' = \{x : x \in L$ ou $x \in L'\}$.

Concatenação: $L \circ L' = \{xy : x \in L$ e $y \in L'\}$.

Estrela: $L^* = \{x_1 x_2 \ldots x_k : k \geq 0$ e cada $x_i \in L\}$.

Note que, para qualquer L, $\varepsilon \in L^*$.

Σ^* = conjunto de todas as cadeias sobre Σ

Máquinas de Turing

MS 3.1

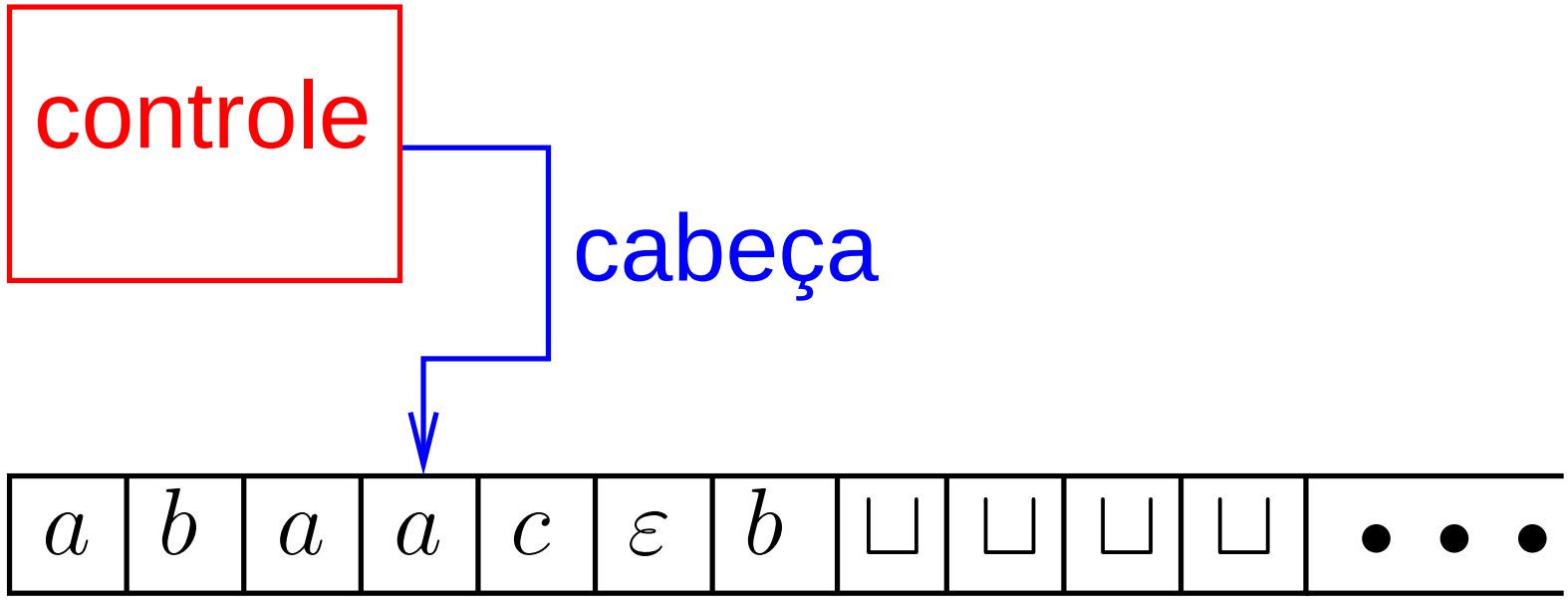
Modelo de computação

É uma descrição abstrata e conceitual de um computador que será usado para executar um algoritmo.

Um modelo de computação especifica as **operações elementares** um algoritmo pode executar e o critério empregado para medir a quantidade de tempo que cada operação consome.

No **critério uniforme** supõe-se que cada operação elementar consome uma quantidade de tempo constante.

Máquinas de Turing



fita de leitura e escrita

Componentes:

1. controle
2. cabeça
3. fita

Sobre os componentes

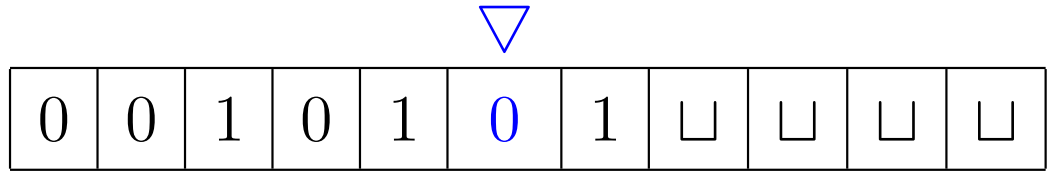
Componentes de uma máquina de Turing:

Controle: possui um conjunto finito de estados, dentres eles há 3 estados especiais: *inicial*, *aceitação* e *rejeição*;

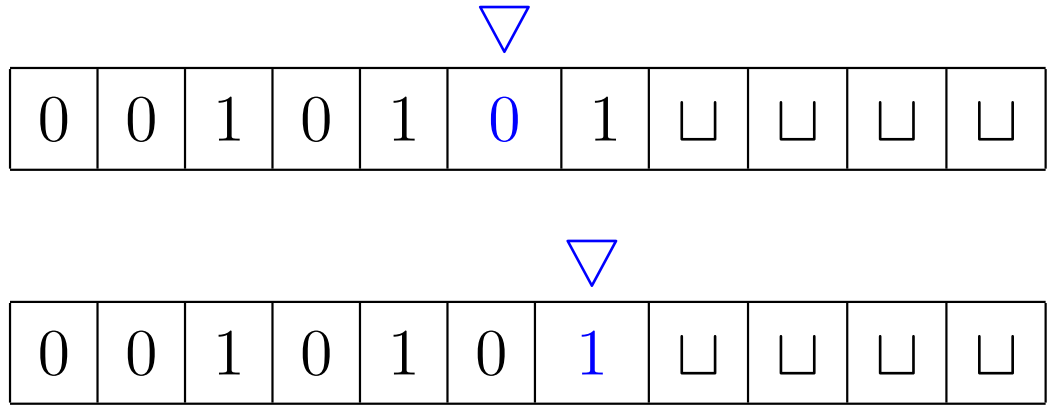
Cabeça: pode ler o símbolo na posição sobre a qual está, escrever um símbolo nessa posição e mover uma posição para a direita ou esquerda;

Fita: infinita à direita, possui brancos $\sqcup$ nas posições não utilizadas.

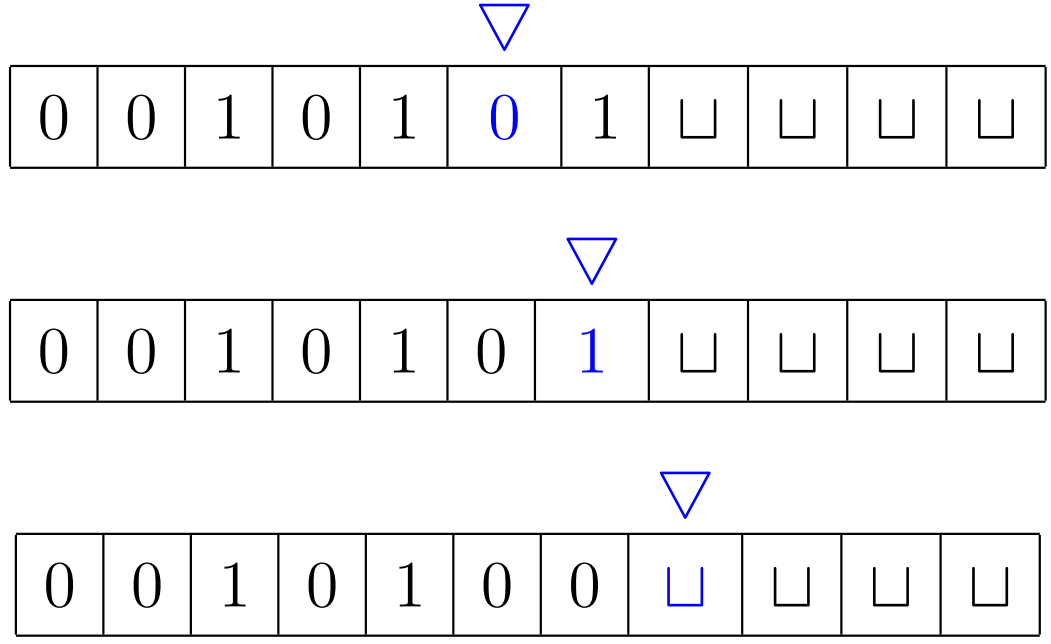
Exemplo de computação



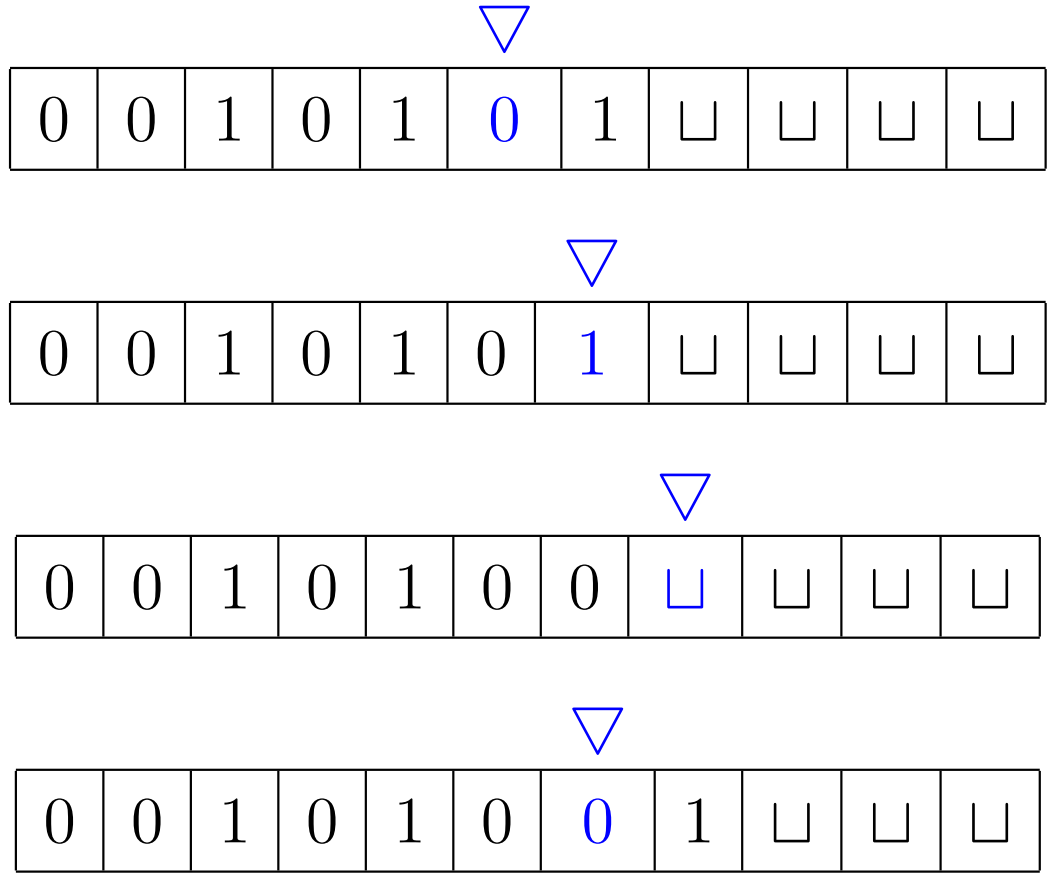
Exemplo de computação



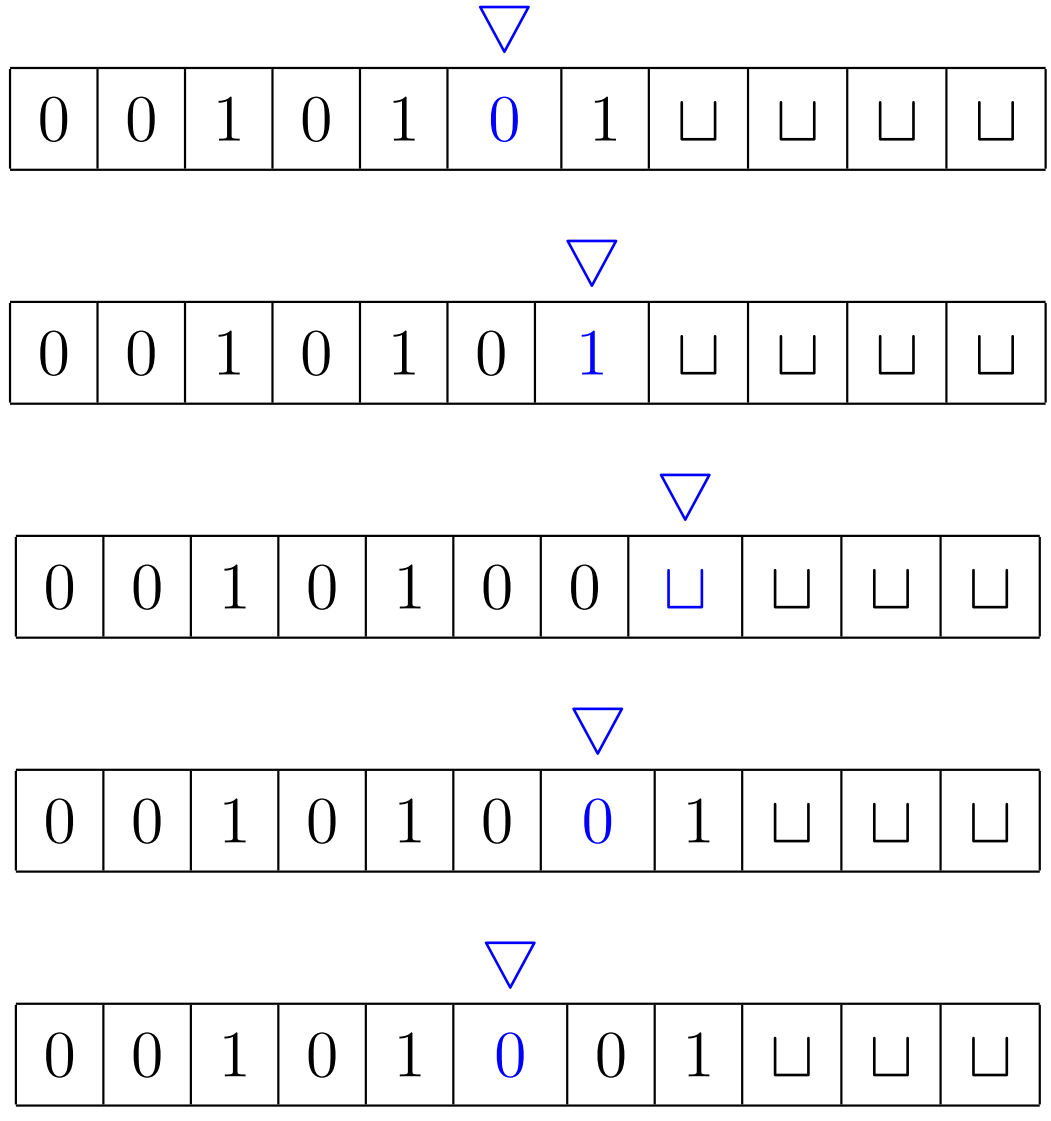
Exemplo de computação



Exemplo de computação



Exemplo de computação



Ainda sobre os componentes

Inicialmente:

Controle: está em um estado inicial;

Cabeça: está sobre a primeira posição da fita;

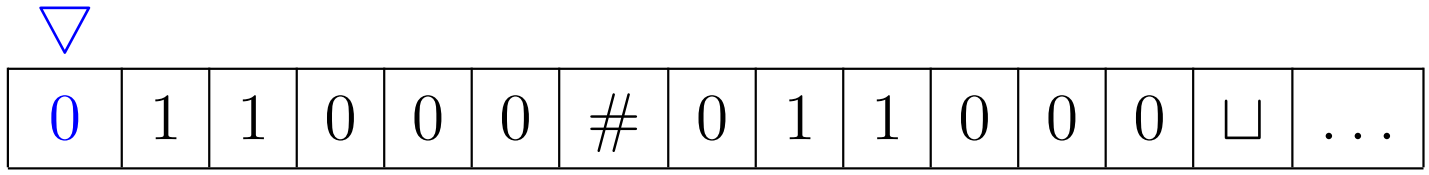
Fita: infinita contém a cadeia representando a entrada do problema nas primeiras posições, as demais posições contém brancos $\sqcup$.

Exemplo de máquina de Turing

Descrição alto nível de uma máquina de Turing que decide se uma dada cadeia w está na linguagem

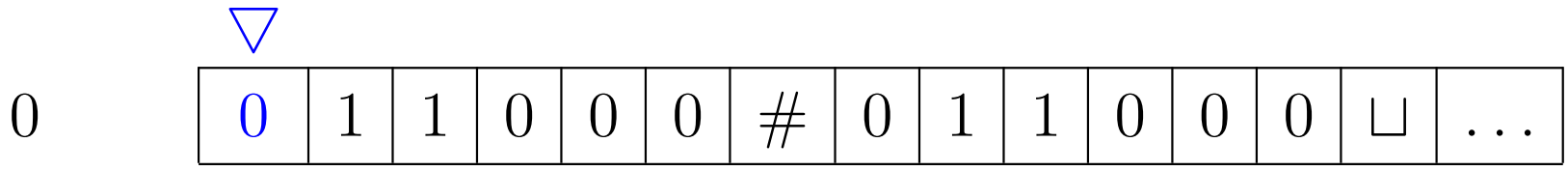
$$\{x\#x : x \in \{0,1\}^*\}.$$

$M_1 =$ "Com entrada w:



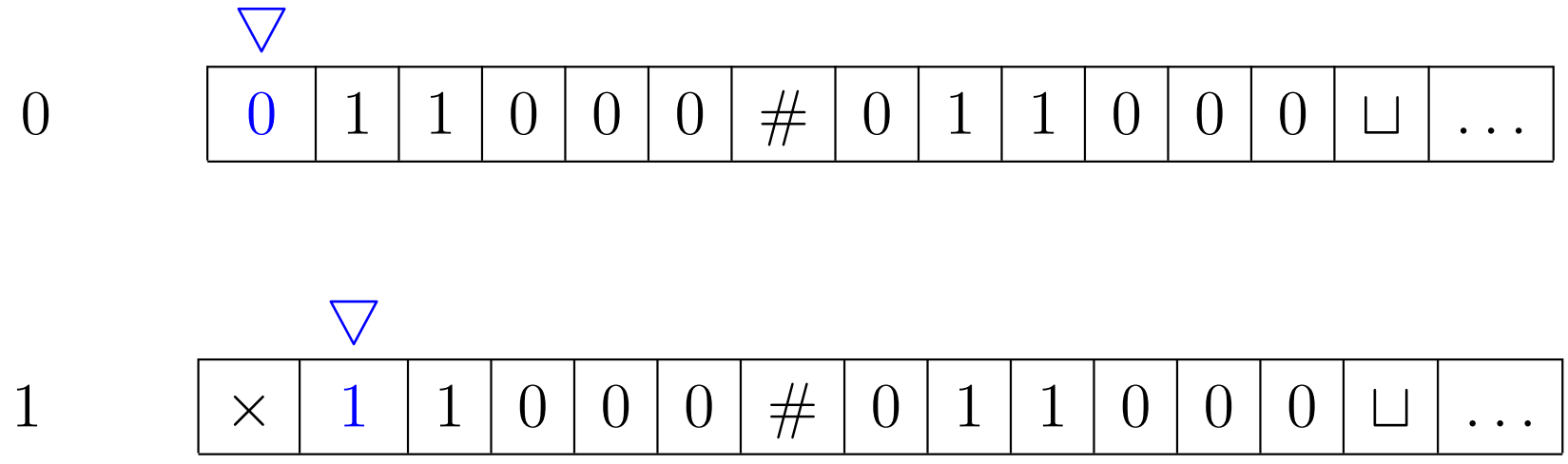
Simulação de M_1

passo



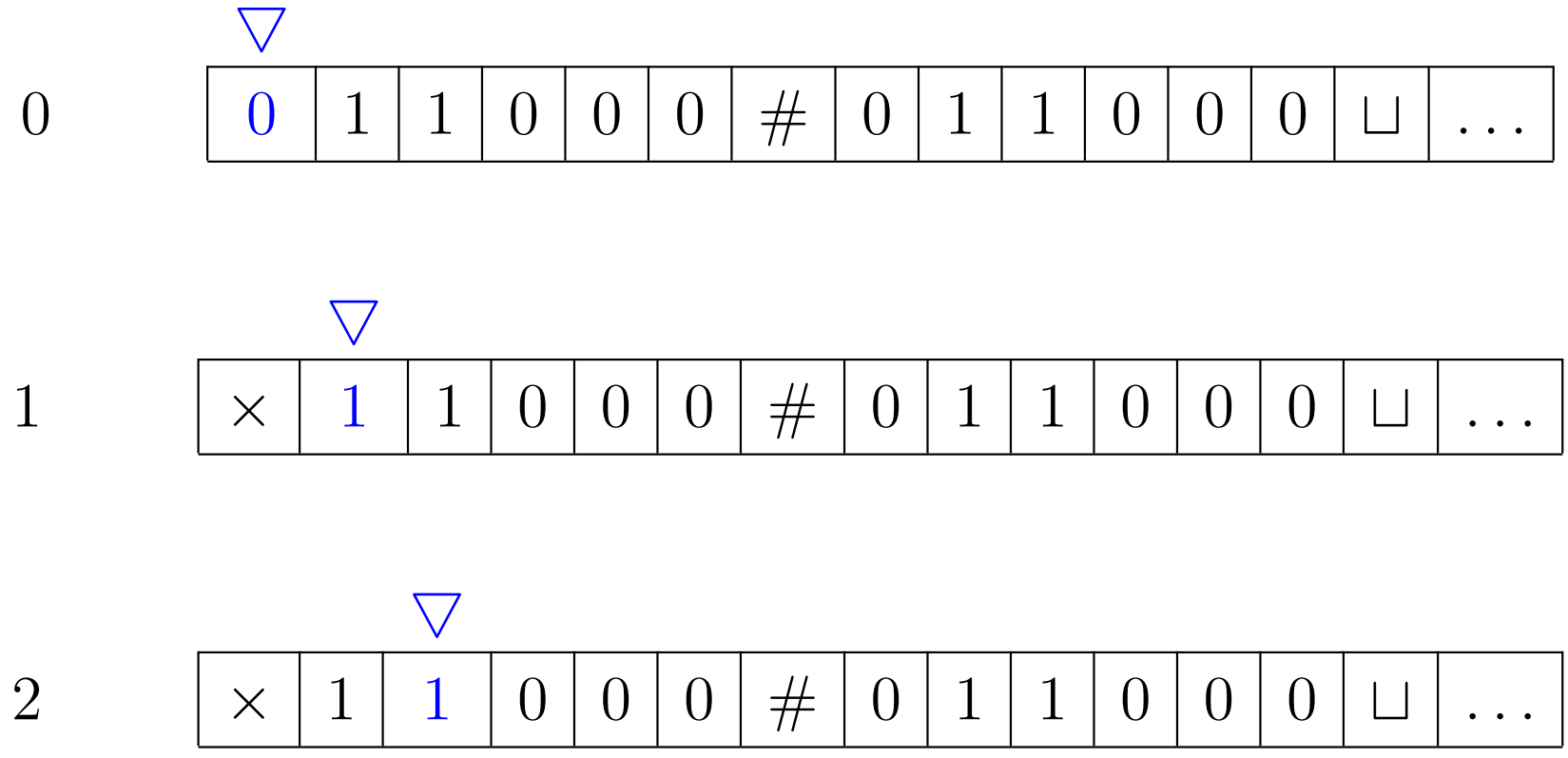
Simulação de M_1

passo



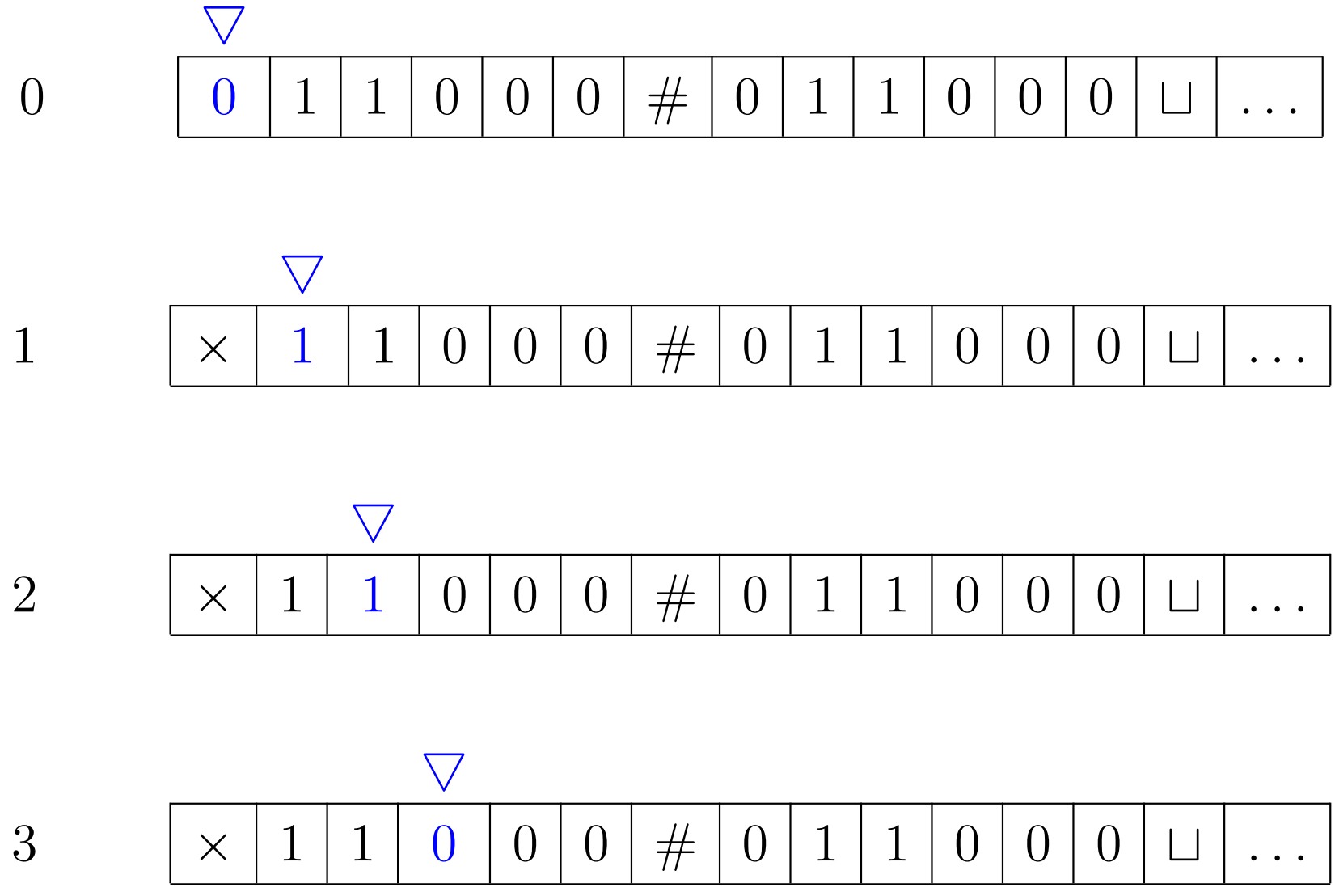
Simulação de M_1

passo



Simulação de M_1

passo

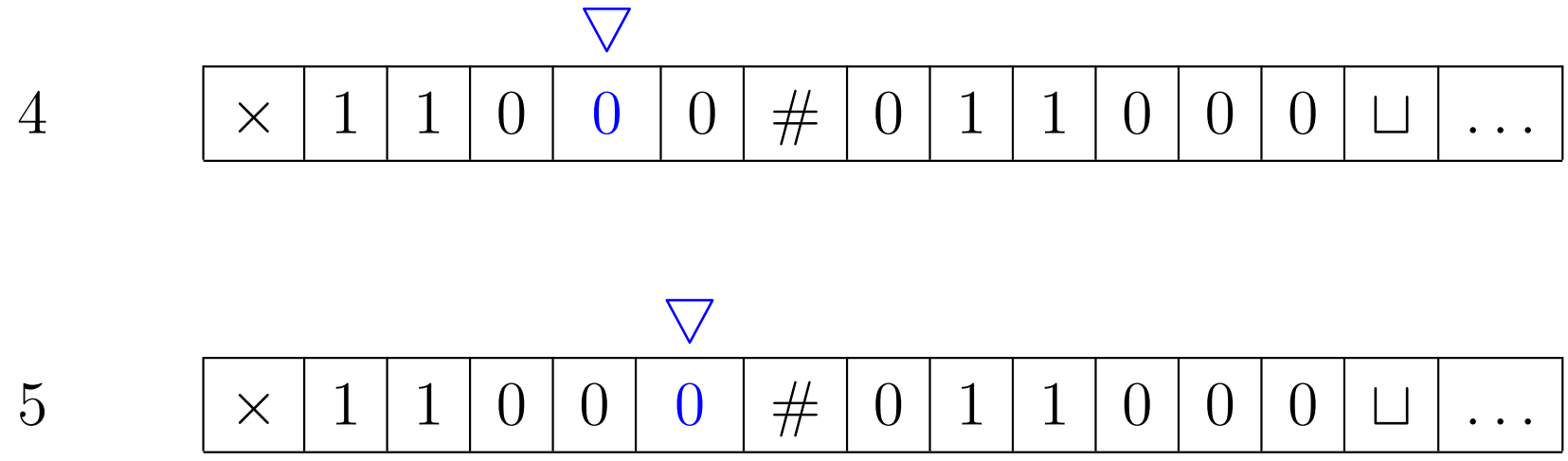


Simulação de M_1

passo

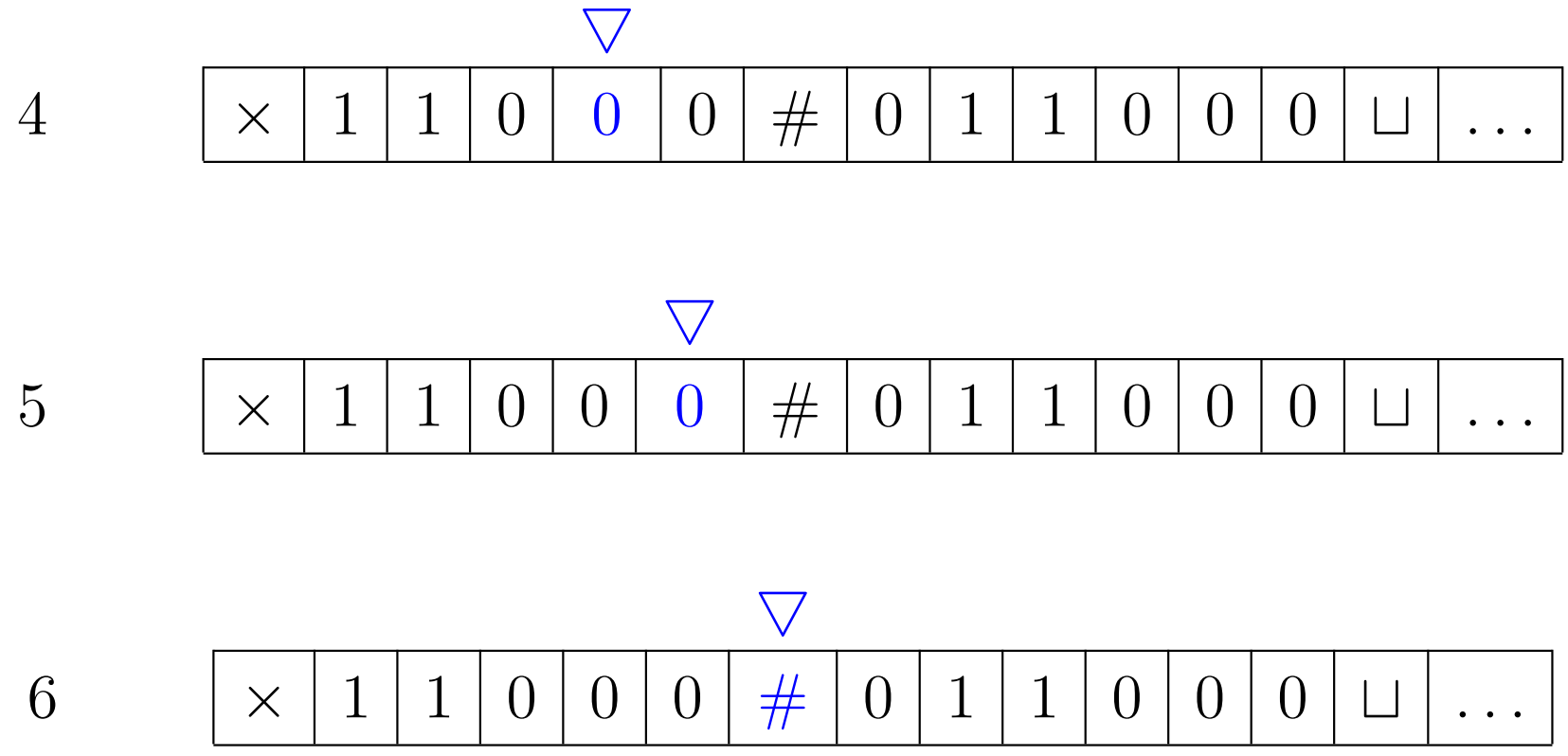
Simulação de M_1

passo



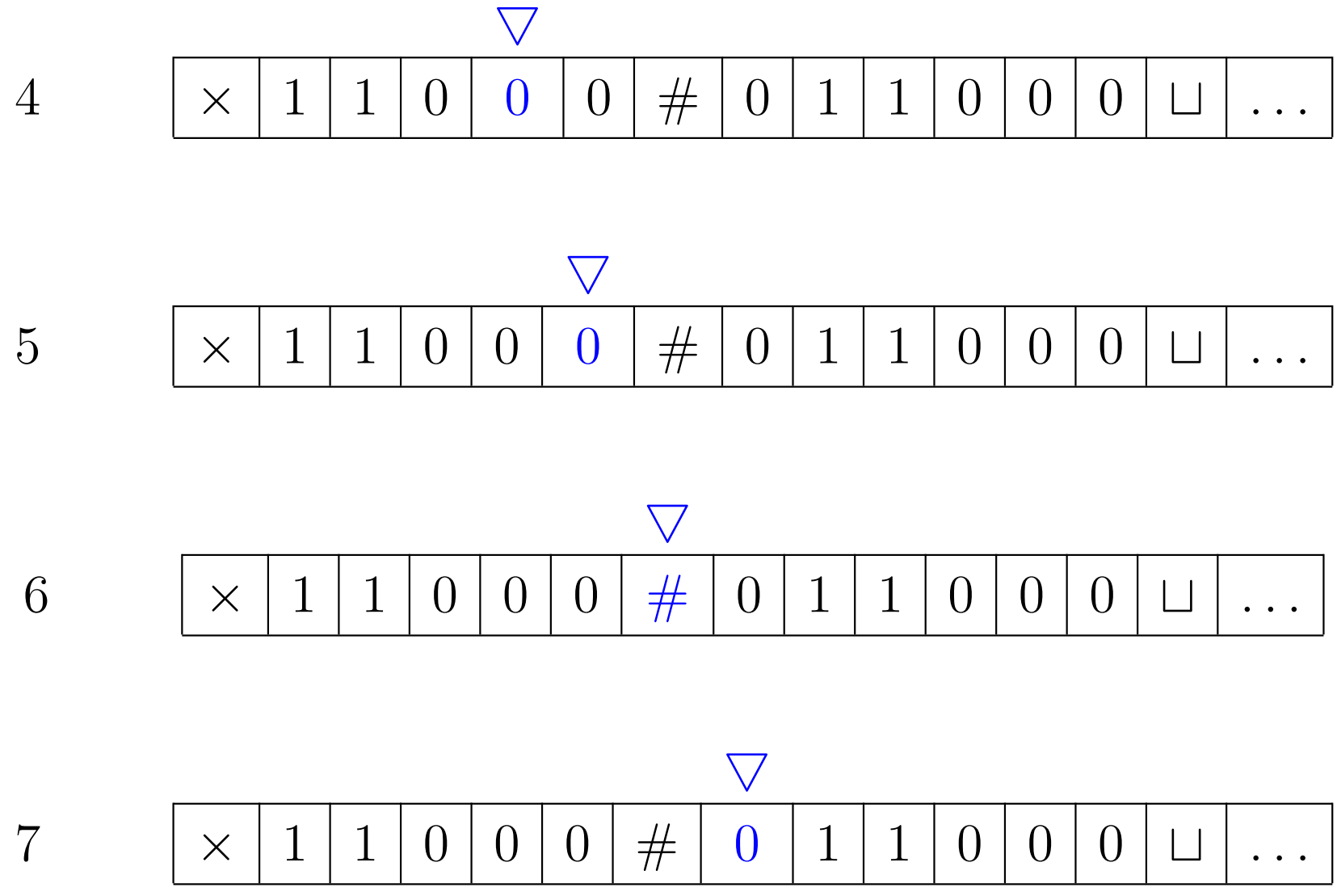
Simulação de M_1

passo



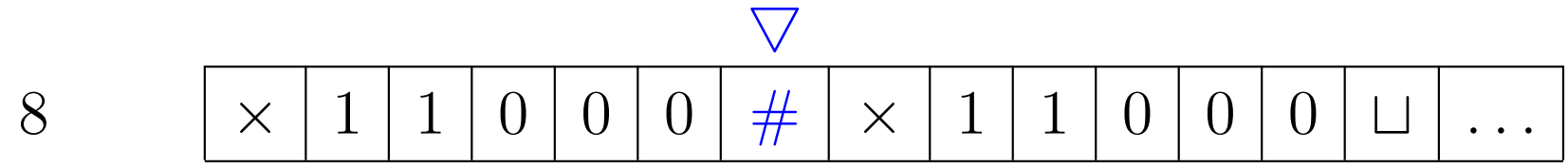
Simulação de M_1

passo



Simulação de M_1

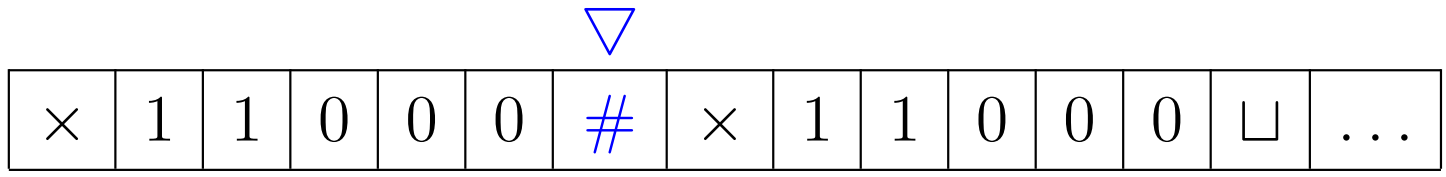
passo



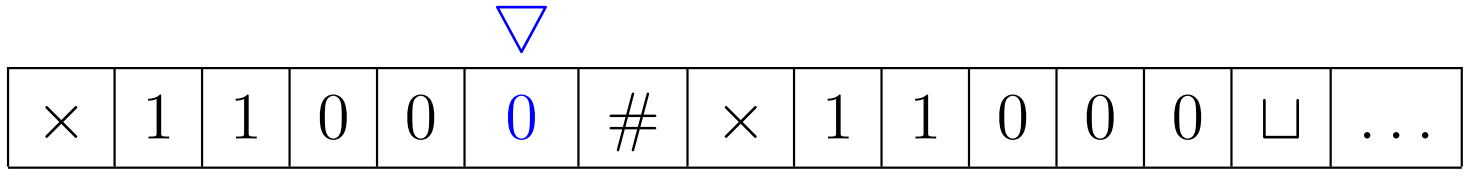
Simulação de M_1

passo

8



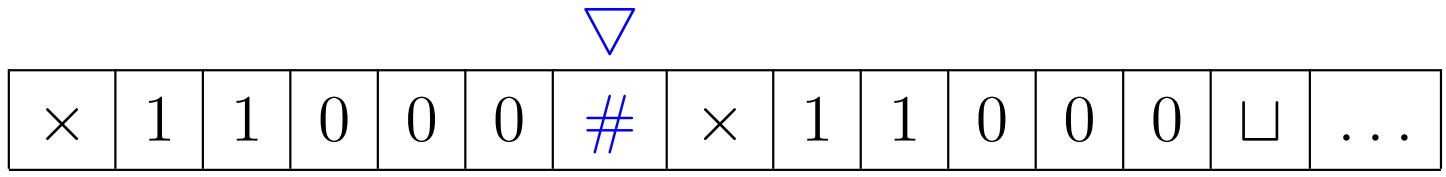
9



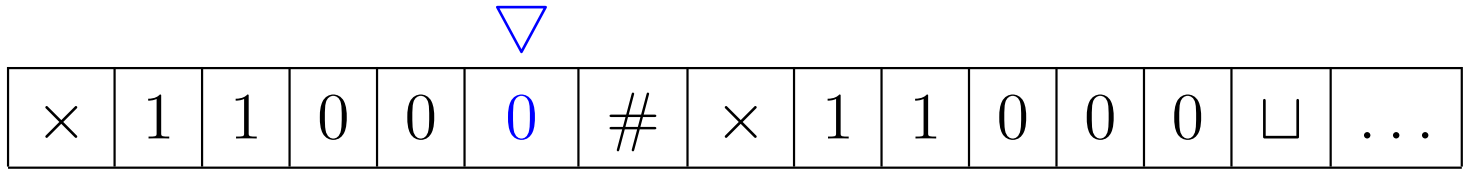
Simulação de M_1

passo

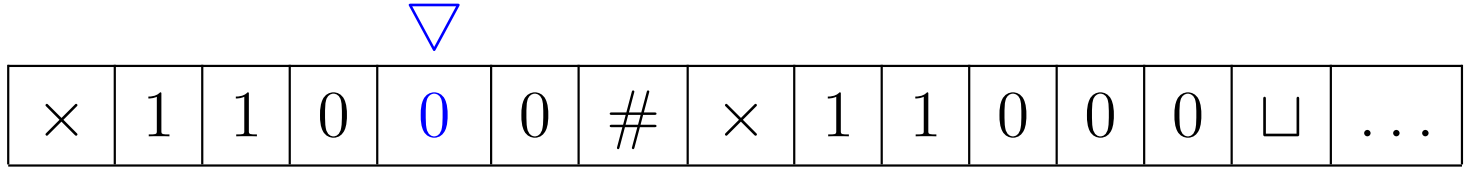
8



9

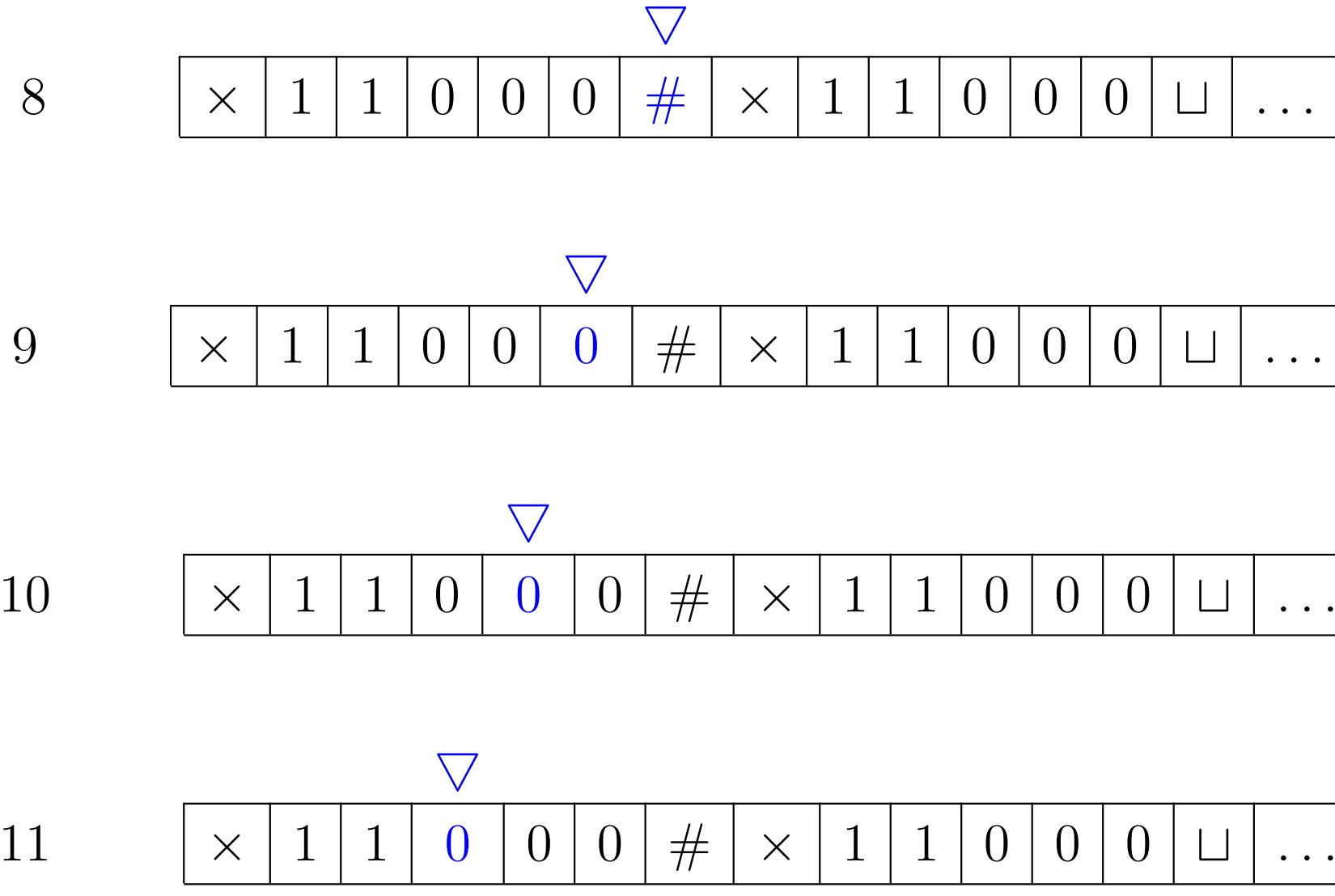


10



Simulação de M_1

passo



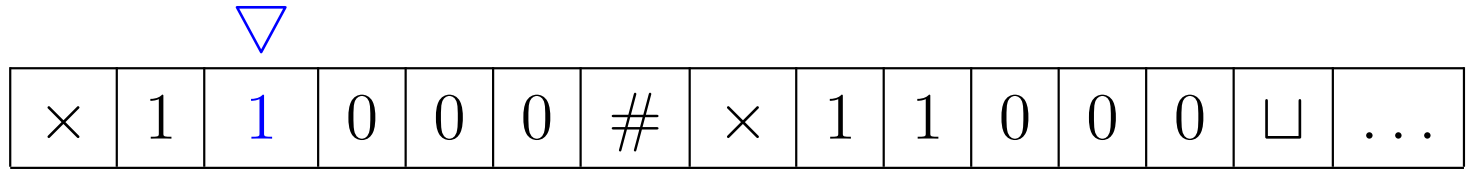
Simulação de M_1

passo

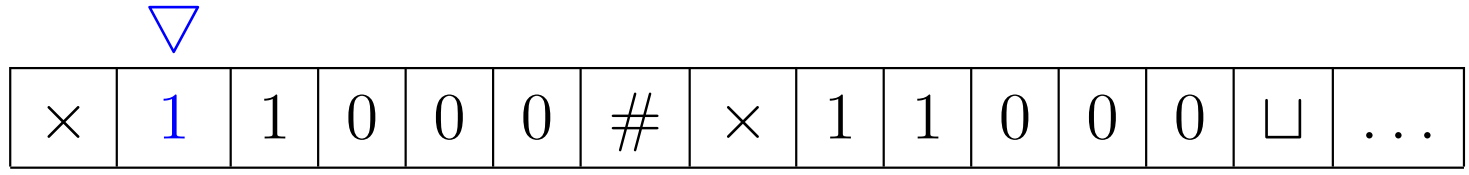
Simulação de M_1

passo

12



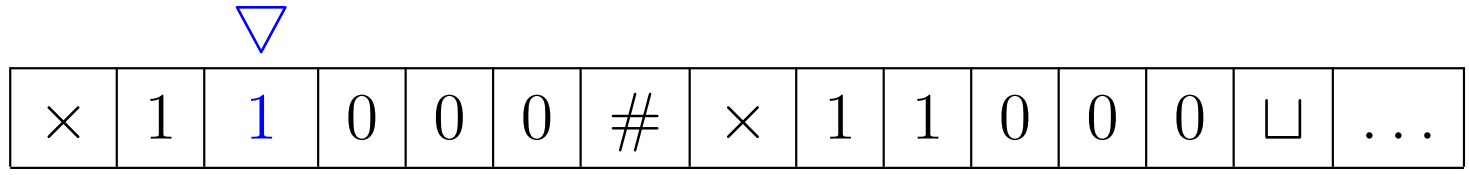
13



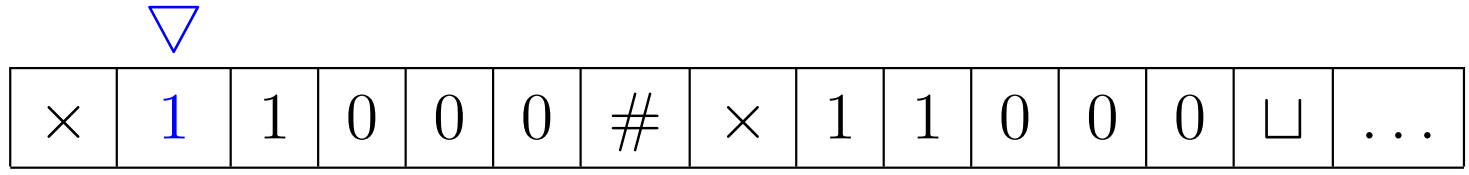
Simulação de M_1

passo

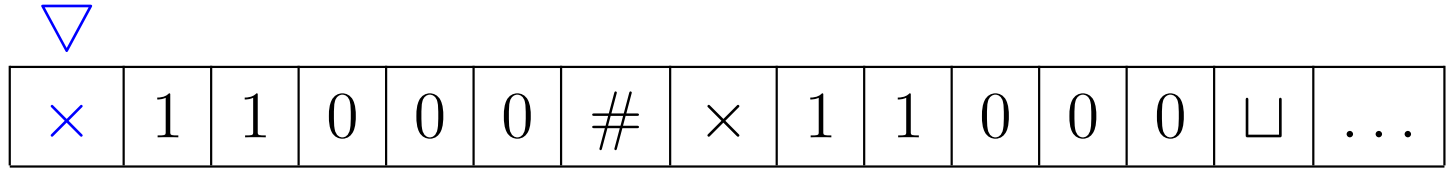
12



13

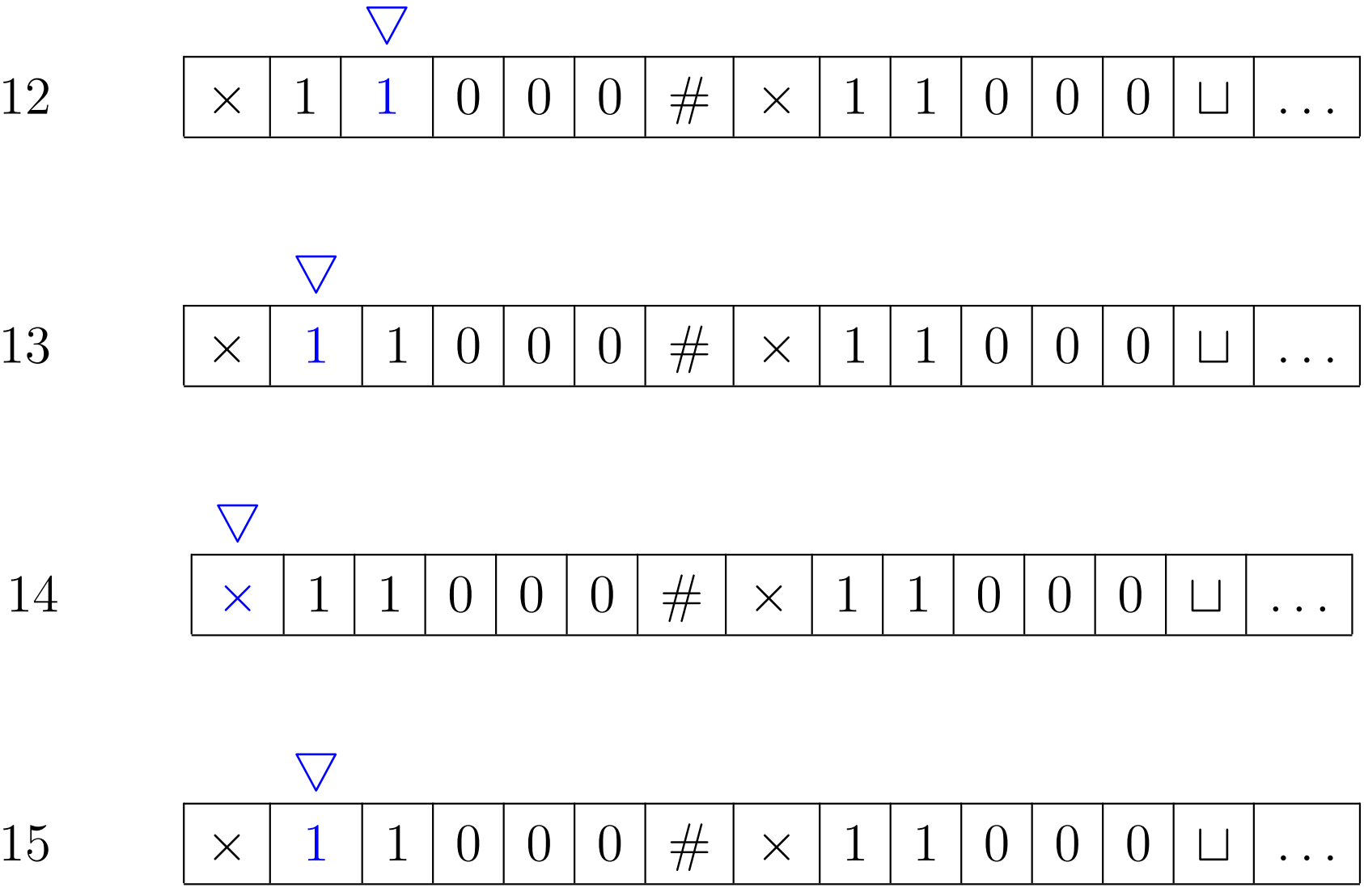


14



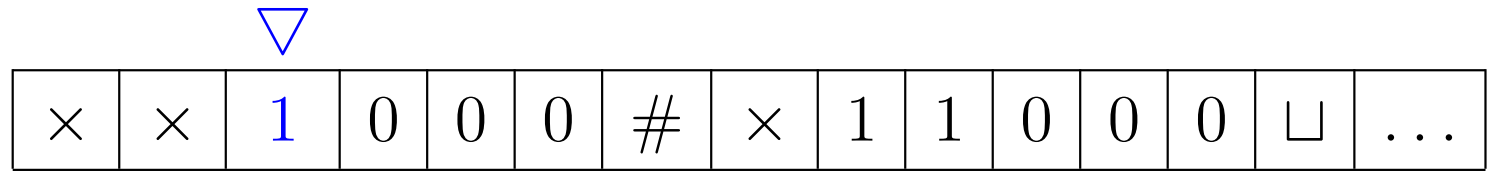
Simulação de M_1

passo



Simulação de M_1

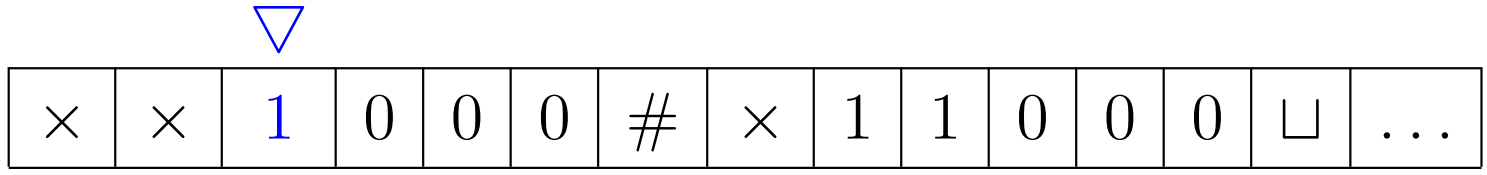
passo



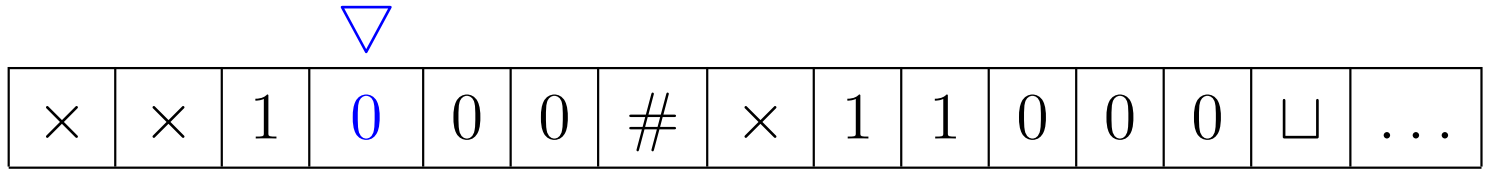
Simulação de M_1

passo

16

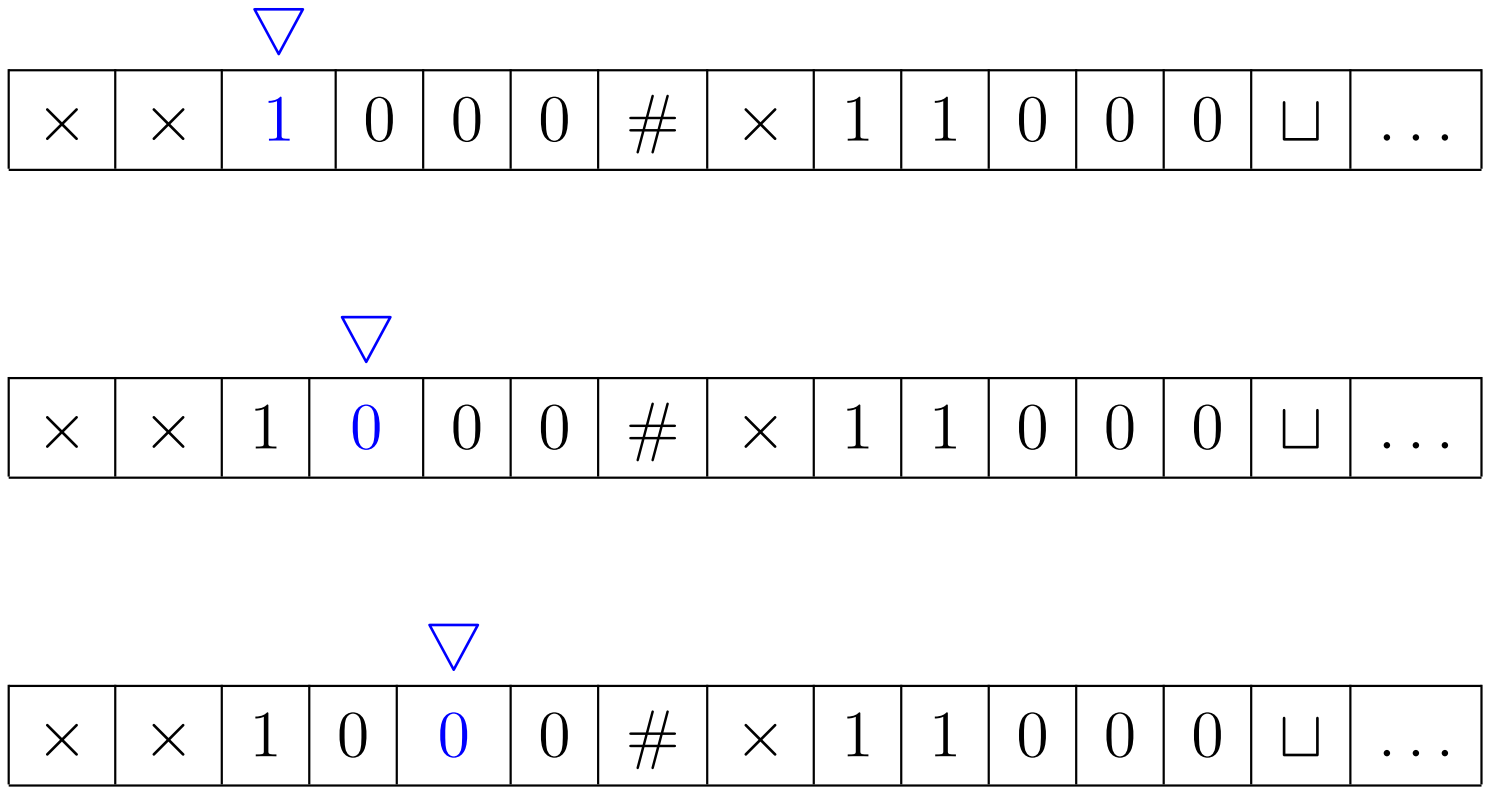


17



Simulação de M_1

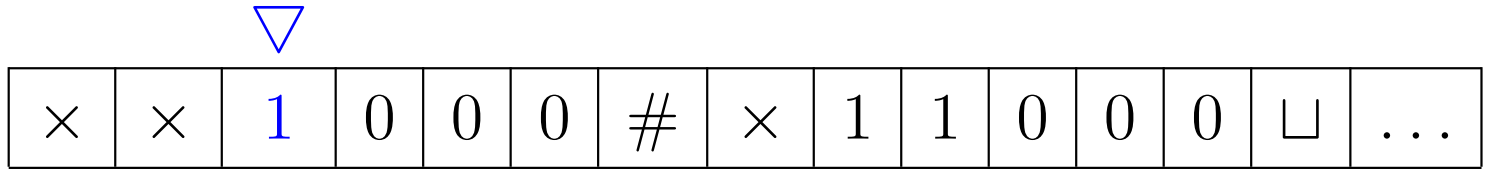
passo



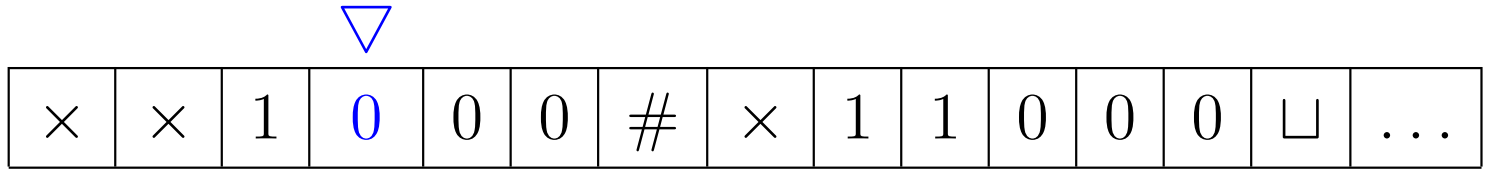
Simulação de M_1

passo

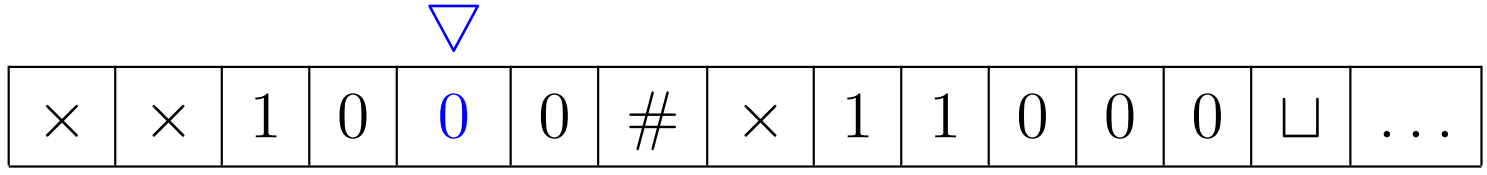
16



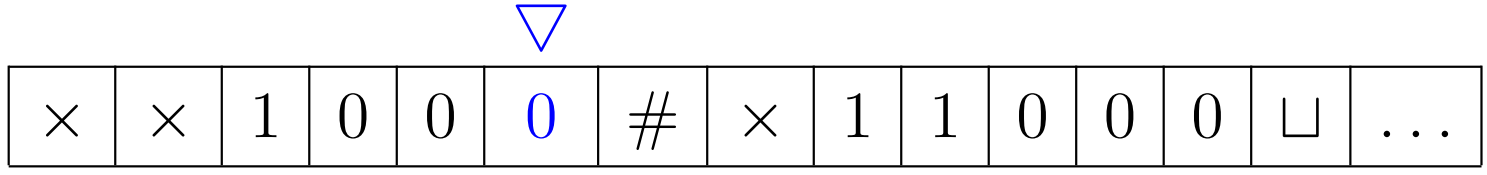
17



18

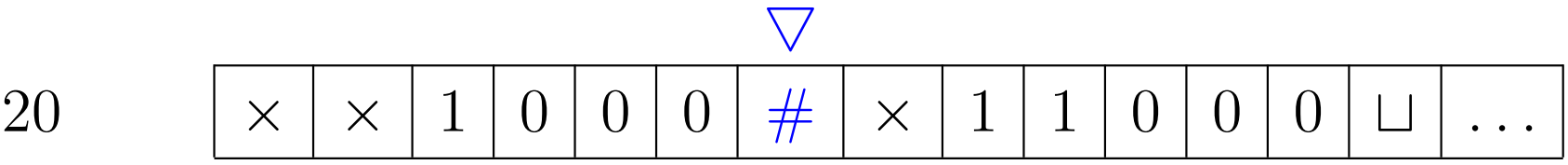


19



Simulação de M_1

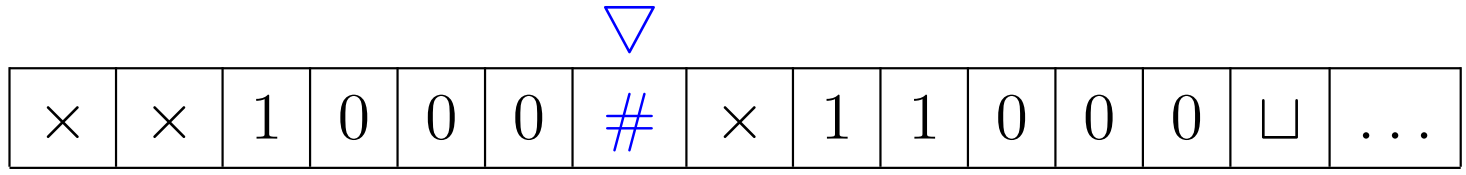
passo



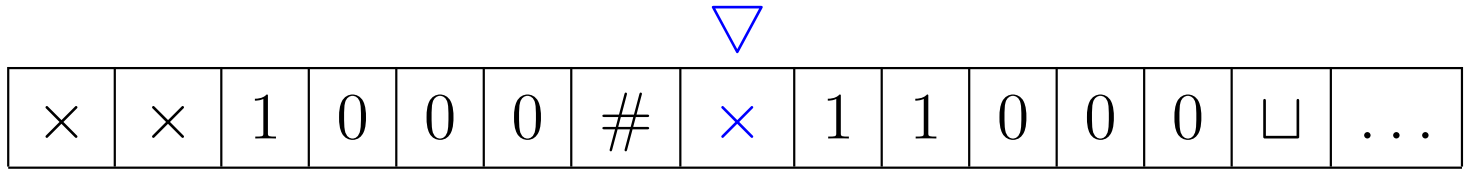
Simulação de M_1

passo

20



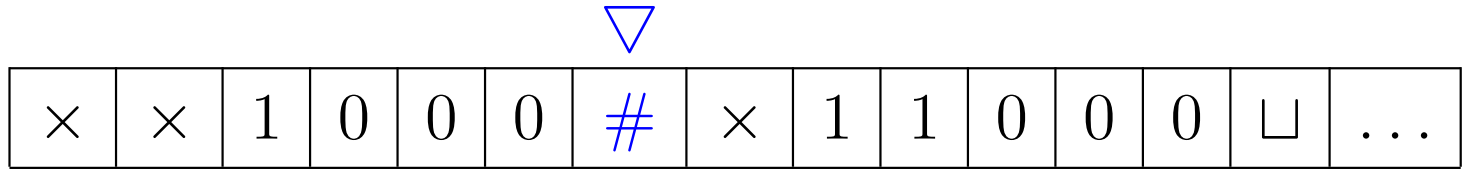
21



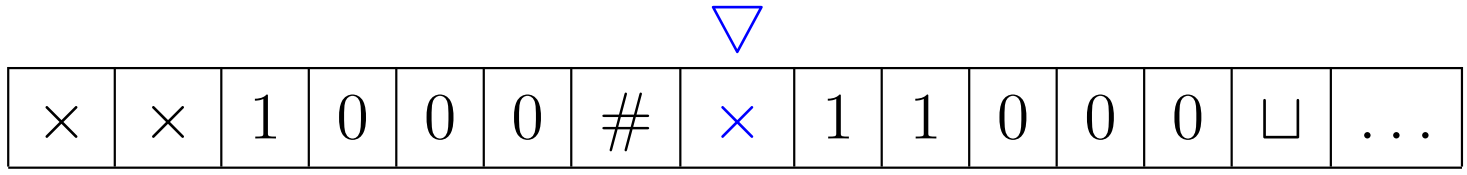
Simulação de M_1

passo

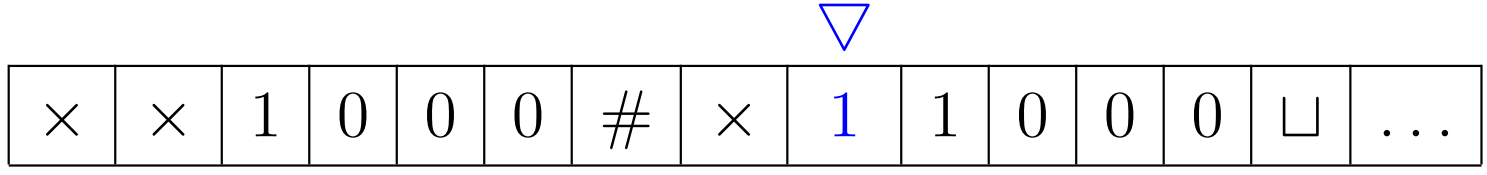
20



21

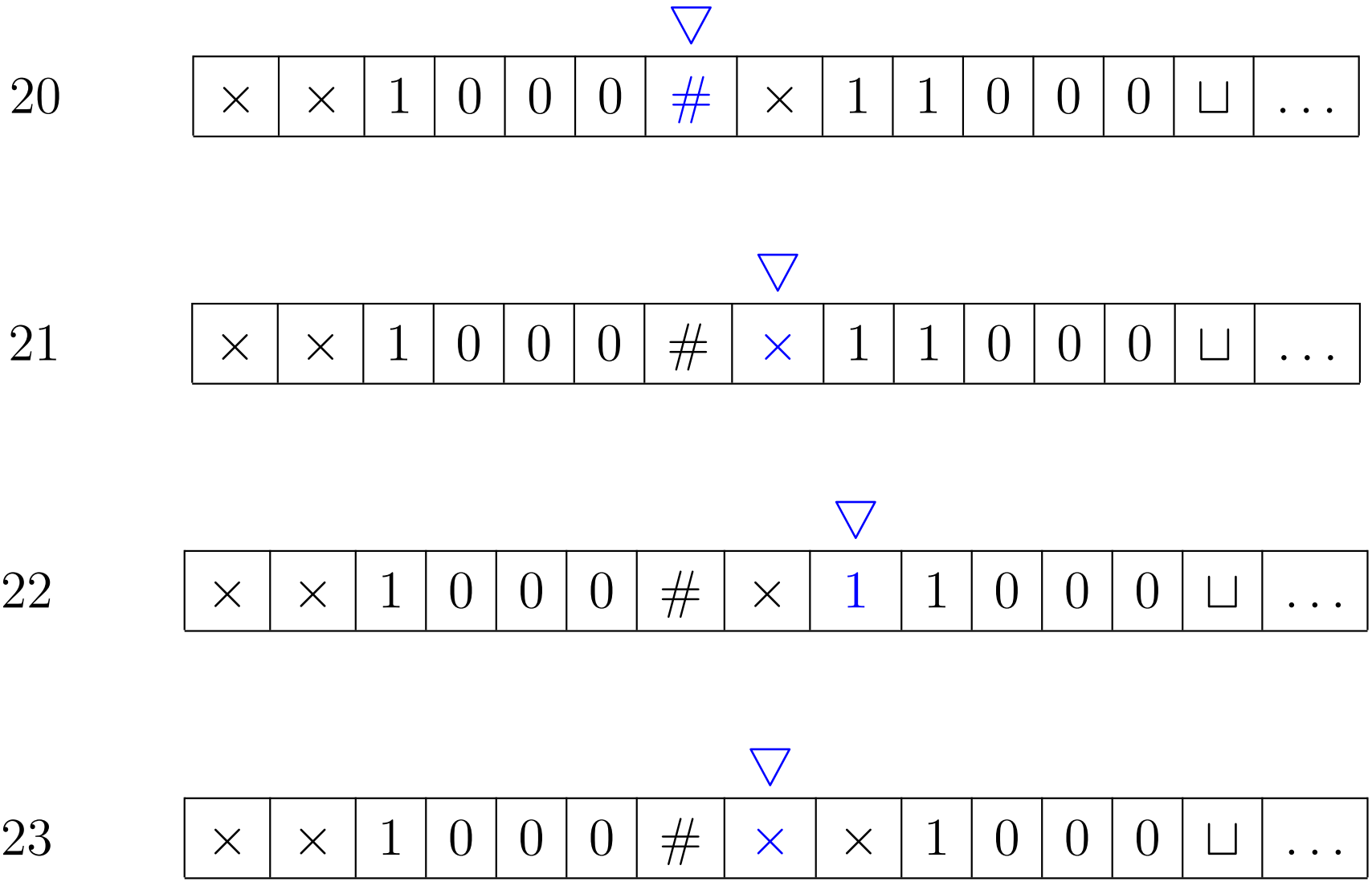


22



Simulação de M_1

passo



Simulação de M_1

passo	conteúdo da fita														
0	$\underline{0}$	1	1	0	0	0	#	0	1	1	0	0	0	$\sqcup$	...
1	×	$\underline{1}$	1	0	0	0	#	0	1	1	0	0	0	$\sqcup$	...
8	×	1	1	0	0	0	#	$\underline{\times}$	1	1	0	0	0	$\sqcup$	...
15	$\underline{\times}$	1	1	0	0	0	#	×	1	1	0	0	0	$\sqcup$	...
16	×	$\underline{\times}$	1	0	0	0	#	×	1	1	0	0	0	$\sqcup$	...
60	×	×	×	×	×	×	#	×	×	×	×	×	×	$\underline{\sqcup}$	...
														aceita	

Exemplo de máquina de Turing

Descrição alto nível de uma máquina de Turing que decide se uma dada cadeia w está na linguagem

$$\{x\#x : x \in \{0,1\}^*\}.$$

$M_1 =$ "Com entrada w:

1. Vá e volte na fita em torno de $\#$ verificando se posições correspondentes contém o mesmo símbolo. Caso negativo, ou no caso em que $\#$ não é encontrado, *rejeite*. Marque os símbolos já verificados.

2. quando todos os símbolos à esquerda de $\#$ foram verificados, examine se sobraram símbolos do lado direito de $\#$. Se sobrou algum símbolo, *rejeite*; senão, *aceite*."