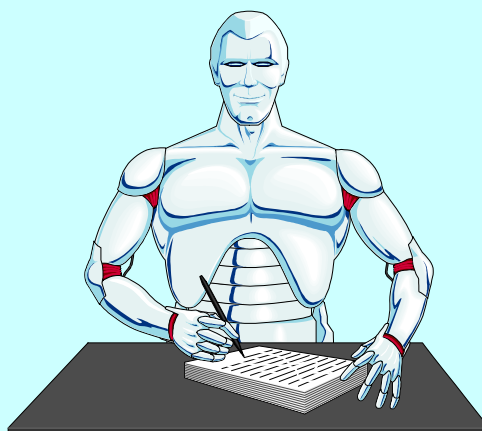




# **INTELIGÊNCIA ARTIFICIAL NO ENSINO MÉDIO**

Construção de computadores que se comportam como humanos.



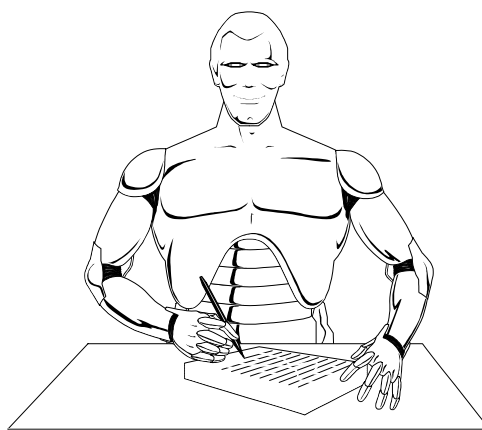
**Prof. Dr. Pedro Luiz Aparecido Malagutti**

**UFSCar  
2002**



# **INTELIGÊNCIA ARTIFICIAL NO ENSINO MÉDIO**

Construção de computadores que se comportam como humanos.



**Prof. Dr. Pedro Luiz Aparecido Malagutti**

**UFSCar  
2002**

**BIENAL DA SBM UFMG**  
**14 a 18 de outubro de 2002**

PEDRO LUIZ APARECIDO MALAGUTTI  
Departamento de Matemática da Universidade Federal de São Carlos  
Rodovia Washington Luiz, Km 235  
13.565-905 SÃO CARLOS - SP  
Fone/Fax 0 XX 16 260 8218  
E-mail: [malagutti@dm.ufscar.br](mailto:malagutti@dm.ufscar.br)

**Título:** INTELIGÊNCIA ARTIFICIAL NO ENSINO MÉDIO – Construção de computadores que se comportam como humanos.

**Tipo:** Minicurso (oficinas pedagógicas).

**Público-Alvo:** Professores e estudantes do Ensino Médio.

**Pré-requisitos:** Não há.

**Materiais:** Por se tratar de uma atividade desenvolvida na forma de oficinas pedagógicas, sugerimos o trabalho em equipes de 4 alunos cada. Serão necessários: uma boa quantidade de papel, disquetes, tesoura, cola, canetas coloridas, canudos de refrigerante, cartolina de várias cores, fios elétricos maleáveis e finos, pilhas pequenas e lâmpadas de 1,5 V. Um computador com configuração standard também será necessário para as atividades com *softwares*.



## A MÁQUINA

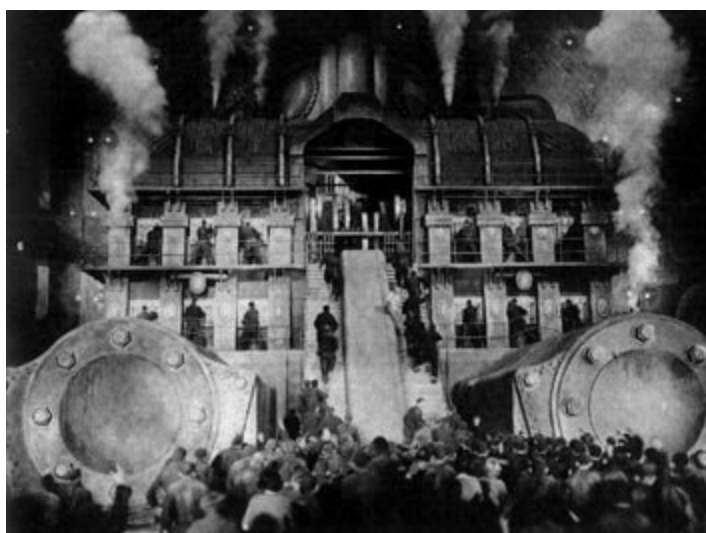
*Rainer M. Rilke*

A máquina é a ameaça a todo o conseguido  
Quando ousa estar no espírito e não na obediência.  
Para que não brilhe, esplêndida e hesitante, a mão,  
talha a rija pedra para o edifício atrevido.

Nem uma vez lhe escapamos, pois nunca se atrasa.  
Luzente de óleo, na muda fábrica está bem.  
E resoluta, ela ordena, ela cria, ela arrasa.  
É a vida – que julga entender como ninguém.

Mas a existência ainda nos fascina, tantos  
são os lugares onde nasce. Um jogo de puras  
forças que se toca de joelhos, com espanto.

Palavras roçam ainda o inefável, asas...  
E, de pedra palpitante, a sempre nova música  
ergue no espaço inapto a sua divina casa.



Agradecimento:

Agradeço a colaboração dos colegas do Departamento de Matemática da UFSCar, pelos muitos anos de cooperação, companheirismo e aprendizagem mútua.

À minha esposa Thomyris e às minhas filhas Lygia e Elisa

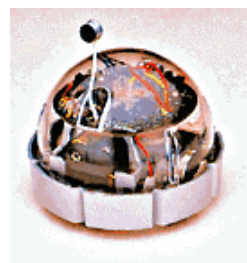




# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b> <i>Uma pequena viagem pelo reino da Lógica e da Computação</i>	<b>1</b>
<b>2</b>	<b>LÓGICA</b> <i>Dedução e Indução. O impossível e o contraditório</i>	<b>5</b>
<b>3</b>	<b>MÁQUINAS QUE APRENDEM</b> <i>Máquinas que jogam, aprendem a jogar e ganham de humanos</i>	<b>15</b>
<b>4</b>	<b>MÁQUINAS DE CALCULAR DO ARCO DA VELHA</b> <i>Como calcular sem eletricidade</i>	<b>31</b>
<b>5</b>	<b>CONVERSANDO COM MÁQUINAS</b> <i>Como entender a linguagem de uma máquina. A caixa preta de um disco voador</i>	<b>61</b>
<b>6</b>	<b>AUTÔMATOS</b> <i>Modelos inteligentes fora do tempo e do espaço</i>	<b>67</b>
<b>7</b>	<b>COMO UMA MÁQUINA PODE SE LEMBRAR</b> <i>Porque um computador nunca esquece</i>	<b>77</b>
<b>8</b>	<b>GRAMÁTICA COMBINA COM MATEMÁTICA?</b> <i>Organizando as linguagens que são entendidas por máquinas</i>	<b>89</b>
<b>9</b>	<b>O MAIS PODEROSO DE TODOS OS COMPUTADORES</b> <i>Software Visual Turing. Projetos para a construção de computadores</i>	<b>95</b>
<b>10</b>	<b>A VIDA SEXUAL DAS MÁQUINAS</b> <i>Auto-reprodução. O software Winlife</i>	<b>115</b>
<b>11</b>	<b>O QUE UMA MÁQUINA FAZ MAS NÃO DEVERIA FAZER</b> <i>Complexidade computacional. O problema P versus NP</i>	<b>121</b>
<b>12</b>	<b>O QUE AS MÁQUINAS NUNCA FARÃO</b> <i>Impossibilidades lógicas</i>	<b>127</b>
<b>13</b>	<b>REFERÊNCIAS BIBLIOGRÁFICAS OU NÃO</b>	<b>139</b>





# 1. INTRODUÇÃO

## *Uma pequena viagem pelo reino da Lógica e da Computação*

Será que no futuro, talvez não muito distante, as máquinas irão dominar o mundo? Será possível construir um cérebro eletrônico que realize todas as tarefas intelectuais humanas? Acredite, estas preocupações ocupam a mente de muitos de nossos alunos, seja pela influência da mídia, seja pela rápida disseminação da Informática. Até que ponto eles estão prevendo as conseqüências científicas das novas tecnologias ou até que ponto estão apenas alimentando apenas fantasias da ficção científica?

A tecnologia realmente tem fascinado nossos alunos a ponto de muitos adorarem o trabalho com tarefas complicadas de computação e paradoxalmente odiarem a Matemática, sem se aperceberem da estreita união que há entre as duas áreas. Além disso, o uso indiscriminado de jogos eletrônicos e *softwares*, em vez de aproximar o conhecimento científico do cotidiano, tem relegado a um segundo plano as tarefas e projetos que poderiam ser desenvolvidos pelos alunos “com suas próprias mãos”, retirando assim do processo de ensino e de aprendizagem a sedução da descoberta e o prazer de conhecer o mundo pelos olhos da razão.

A aprendizagem de uma linguagem de programação (comunicação homem-máquina), que permita ao estudante entender o funcionamento de certos aplicativos e programas, também merece atenção. Existem muitas linguagens de programação e o domínio de todas parece impossível. Entretanto todas elas têm um substrato comum baseado em idéias matemáticas simples (estrutura lógica e recursividade, por exemplo). Nosso objetivo neste minicurso também é o de motivar os alunos à compreensão destas idéias simples, porém fundamentais. Com isto esperamos estar incentivando a iniciativa de se produzir novos

conhecimentos, que vão além dos que se apresentam nos pacotes computacionais fechados.

O objetivo desta proposta de minicurso (na verdade uma série de oficinas pedagógicas) é o de aproveitar esta curiosidade natural dos alunos para trabalhar uma questão artificial: a Inteligência Artificial. O objetivo não é o de adentrar tecnicamente nesta intrincada área do conhecimento, mas utilizar as questões ligadas à inovação tecnológica para ensinar Matemática, particularmente Lógica Matemática. Por isto esta proposta está mais ligada à Lógica do que à Informática (no sentido de utilização e domínio de *softwares*). Trata-se de uma proposta de ensino *sobre* computadores e seu funcionamento lógico, para esclarecer o papel do ensino *com* computadores.

Como linha de trabalho, será apresentada no minicurso uma proposta envolvendo temas interdisciplinares que entrelaçam várias disciplinas do Ensino Médio, tendo a Lógica Matemática como seu fio condutor. Apesar da Lógica Matemática permear todas as ciências exatas e da natureza, não é muito comum no ensino tradicional dedicar atenção a esta área, nem mesmo estabelecer conexões entre a Lógica e as outras áreas do conhecimento.

De acordo com os Parâmetros Curriculares Nacionais, a escolha de temas transversais que integrem as diferentes áreas do saber devem estar presentes em todas as atividades didáticas do Ensino Médio e Fundamental. Assim como as outras ciências (tais como a Física, a Química e Computação), a Matemática tem uma linguagem própria que, de certo modo, impede o diálogo com outras áreas, tornando difícil a utilização de suas tecnologias para um olhar mais interdisciplinar do mundo.

Queremos com essa oficina pedagógica motivar e satisfazer o aluno na sua curiosidade frente as novas tecnologias ligadas à Inteligência Artificial, apresentando exemplos simples de máquinas que podem ser programadas e realizam tarefas surpreendentes. São projetos que podem ser trabalhados em várias fases da educação escolar no Ensino Médio e que procuram despertar no aluno o gosto pelo estudo da Matemática e da Computação, explicitando o papel fundamental da Lógica Matemática no desenvolvimento da tecnologia e suas perspectivas futuras.

Nas oficinas, pretendemos trabalhar com a seguinte questão: quais as características dos seres humanos que as máquinas podem possuir? Serão apresentados e confeccionados alguns kits experimentais contendo atividades sobre máquinas que aprendem, comunicam-se, memorizam e até mesmo reproduzem-se. Os kits constituem-se numa espécie de Projeto Frankstein a respeito da criação de uma máquina pensante. As atividades que serão desenvolvidas podem ser agrupadas de acordo com os seguintes tópicos:

- A Lógica Matemática e o surgimento de paradoxos. O que é impossível e o que é contraditório.

- Máquinas que jogam, aprendem a jogar e ganham de humanos. Computadores feitos com caixas de fósforos.
- Máquinas que calculam sem eletricidade e sem mistério.
- Linguagens formais e comunicação entre humanos e máquinas. A caixa preta de um disco voador.
- Autômatos. Computadores do tempo do Flash Gordon (já ouviu falar dele?)
- Como uma máquina pode se lembrar. Porque um computador nunca esquece.
- Gramática combina com Matemática? (Matemática e Português combinam incrivelmente bem).
- O mais poderoso de todos os computadores. Software VisualTuring.
- A vida sexual das máquinas. Auto-reprodução. Software Winlife.
- O que uma máquina faz, mas não deveria fazer: o problema  $P \neq NP$ . Depois que algo está feito até uma máquina acredita e confere.
- O que as máquinas nunca farão. O problema da parada e as impossibilidades lógicas.

Todos os experimentos são confeccionados com materiais simples (papel, caixas de fósforos, lâmpadas e pilhas, planilhas eletrônicas, etc...) e não exigem conhecimentos profundos de eletrônica ou de computação.

Acreditamos que para o entendimento de uma dada tecnologia é necessário conhecermos primeiramente o contexto histórico dentro do qual foram incubadas as descobertas científicas. Com esse intuito, o enfoque que daremos será direcionado primeiramente para o estudo da matemática dos computadores. Não daremos prioridade ao computador como ferramenta de trabalho, nem ao seu uso em atividades específicas, embora acreditemos que essas sejam tarefas importantes, colocadas como um desafio para os educadores. O que pretendemos é fazer uma pequena viagem pelo universo da computação, tentando revelar o grande potencial que ainda nos espera e as limitações inerentes ao uso de máquinas “inteligentes”.

Grande parte do trabalho a respeito da estrutura teórica de um computador foi realizada na década de 30 do século passado, muito antes da criação das máquinas tais quais as conhecemos hoje em dia. Este trabalho foi desenvolvido por lógicos (Turing, Church e Post entre outros) e permite uma avaliação teórica e crítica dos computadores, constituindo ainda uma ferramenta pedagógica atual para o seu entendimento e correta utilização. Esta proposta de trabalho tentará captar o exato momento em que a lógica se funde com a computação.

Em outras palavras, queremos desmistificar o computador, estudando sua essência. E, sua essência é a Lógica Matemática. Este tema, desde a sua criação, relaciona-se com uma das mais fascinantes teorias já desenvolvidas, a Inteligência Artificial, e possui conexões com a Sociologia, Psicologia, com todas as ciências exatas e naturais e, por si mesmo, deve provocar mudanças em nosso fazer pedagógico. Deste ponto de vista, a interface da Lógica/Computação permite conhecer as limitações e impossibilidades das máquinas. Nos interessamos aqui

pelas impossibilidades matemáticas pois somente a Matemática é capaz de tratar com o impossível. As impossibilidades impostas pela Lógica não são técnicas e não há esperanças de transpô-las no futuro, embora acreditamos também que não hajam obstáculos intransponíveis para o poder criador da mente humana.

Iniciaremos com uma pequena apresentação sobre lógica elementar, seus cativantes paradoxos e uma abordagem histórica enfatizando as bases científicas e os avanços tecnológicos na interface lógica/informática.

Apresentaremos então os autômatos, que nada mais são do que modelos teóricos de computadores. Serão desenvolvidas também as bases das linguagens formais, isto é, linguagens que podem ser entendidas por máquinas. Daremos destaque especial às máquinas de Turing, que descrevem teoricamente o funcionamento dos atuais computadores e parecem captar plenamente o sentido do termo “computação”. Pode-se mostrar que as máquinas com acesso aleatório de memória (RAM) são, em tese, equivalentes às máquinas de Turing. Existe um software que simula o funcionamento de uma máquina de Turing, muito simples de ser programado, chamado VisualTuring; com ele entenderemos como o criador pode ser engolido pela criatura!

Faremos muitas referências à importante área de pesquisa chamada Inteligência Artificial e as tentativas de se criar um cérebro totalmente eletrônico. Serão descritas algumas experiências simples, mas importantes do ponto de vista didático, a respeito de como as máquinas aprendem e até de como elas podem se reproduzir. Para isto serão apresentados *softwares* específicos e mostraremos que até mesmo o Excel do aplicativo Windows pode ser utilizado para a reprodução de máquinas. O objetivo é entender como as máquinas realizam tarefas, apresentando uma maneira criativa de se construir concretamente aparatos “pensantes”, feitos com materiais comuns e que podem realizar atividades cognitivas e evolutivas.

Estas experiências foram realizadas dentro do Projeto Pró-Ciências (CAPES/FAPESP) com professores das escolas públicas da região central do estado de São Paulo. Todas atividades propostas foram realizadas com a participação de alunos nas escolas públicas e posteriormente os resultados foram relatados e avaliados pelo grupo de professores/pesquisadores em reuniões promovidas dentro da vigência do projeto. O sucesso obtido nesta jornada serviu de motivação para a apresentação desta proposta.

Pretendemos com isto, enfim, levantar algumas questões relevantes sobre o uso de computadores no ensino, através de uma proposta de implementação das bases científicas ligadas aos computadores e sua aplicabilidade no desenvolvimento do raciocínio lógico dentro do Ensino Médio. Queremos enfim provar que os humanos devem vencer as máquinas!

$$\alpha \Rightarrow \beta$$

## 2. LÓGICA

*Dedução e Indução. O impossível e o contraditório.*

**Resumo:** Trata-se de um pequeno texto onde se apresentam as principais idéias da lógica elementar, distinguindo-se os raciocínios dedutivos dos indutivos. Como fundamentação serão apresentados os surgimentos de paradoxos lógicos. Os paradoxos são excitantes, cativam e motivam o avanço das teorias científicas. Os paradoxos que serão apresentados incluem o paradoxo do mentiroso, o paradoxo de Grelling, do barbeiro de Russel e algumas fontes curiosas de paradoxos em outras áreas do conhecimento.

### A INDUÇÃO E A DEDUÇÃO

Como surgem os estudos científicos? Certamente muito do que conhecemos surgiu da necessidade, da interação do mundo físico com o nosso intelecto. As conclusões baseadas nas experiências são, na maioria dos casos, generalizações de um certo número repetido de observações a respeito de alguma hipótese ou fenômeno. Por exemplo, observando os meses do ano, muitos agricultores planejam como será o plantio de sua lavoura prevendo que nos meses de verão ocorrerão mais chuvas. Do mesmo modo, quando um químico faz certas misturas de elementos em seu laboratório repetidas vezes, as conclusões permitem prever certos comportamentos futuros. Esta maneira de raciocinar recebe o nome de raciocínio indutivo.

Devemos notar que o raciocínio indutivo permite apenas conclusões prováveis a partir de certas afirmações iniciais, chamadas de premissas ou hipóteses. Nos argumentos indutivos se todas as premissas forem verdadeiras, a conclusão provavelmente será verdadeira e as conclusões encerram informações que não estavam totalmente contidas nas hipóteses.

Exemplo de um raciocínio indutivo:

Premissa: Todos os ratos observados no laboratório tinham sangue em suas veias.

Conclusão: Todos os ratos têm sangue em seu aparelho circulatório.

Já que indutivamente só podemos chegar a conclusões prováveis, como saber se um raciocínio indutivo é correto? Isto é, como saber se ele é falso ou verdadeiro? Naturalmente isto vai depender das relações entre as conclusões e as hipóteses, mas existem algumas situações onde o trabalho com argumentações dedutivas é bem entendido. Vejamos algumas delas:

**Analogia:** Os biocientistas em geral fazem testes de remédios em animais para depois testá-los em seres humanos. Com isto espera-se prever as reações de um determinado medicamento no corpo de um homem e testar hipóteses. Evidentemente a certeza de tal raciocínio dependerá do grau de semelhança entre as afirmações testadas e do número de vezes que se repetiu a experiência.

**Enumeração:** Trata-se de um raciocínio indutivo baseado na contagem. Por exemplo, retirando uma amostra de um saco de arroz, observa-se que aproximadamente 80% dos grãos são do tipo extra-fino. Conclui-se então que o saco de arroz é do tipo extra-fino. A validade deste argumento depende muito da quantidade da amostra e os métodos estatísticos são sua base de sustentação. As prévias eleitorais são outro exemplo deste tipo de raciocínio indutivo.

**Método da autoridade:** Frequentemente se faz uso da opinião de uma autoridade para se concluir indutivamente um argumento. Por exemplo, na disputa de um jogo de futebol, decidir se um jogador estava ou não em impedimento. Este tipo de argumento tem a seguinte forma: uma hipótese é feita e a autoridade é digna de confiança para decidir questões sobre ela. A autoridade sustenta que a afirmação expressa na hipótese é verdadeira. Conclui-se que a hipótese é verdadeira. Este raciocínio indutivo claramente gera refutações pois a autoridade pode não entender do assunto, não ter competência ou as evidências podem ser falhas.

**Método da contra-autoridade:** Se uma pessoa não é digna de confiança para afirmar uma certa sentença e faz isto, concluímos que ela provavelmente é falsa (basta pensar nos políticos).

Passemos agora a considerar os raciocínios dedutivos. Este tipo de raciocínio é o preferido dos matemáticos. As conclusões são obtidas, a partir das premissas, usando-se o raciocínio lógico e, uma vez encontradas, as conclusões são incontestáveis. Para sermos mais precisos, um argumento lógico dedutivo é sempre formado por três partes: as hipóteses (ou premissas), as conclusões (ou a tese) e a inferência (ou seja o processo pelo qual passamos das hipóteses à tese).

A lógica dedutiva pode ser ilustrada com o seguinte exemplo:

1. Todo homem é mortal.
  2. Fernando Henrique é um homem.
- Conclusão: Fernando Henrique é mortal.



Observemos que se aceitarmos as hipóteses 1 e 2, somos forçados ou compelidos a aceitar a conclusão. É importante salientar que o raciocínio dedutivo não trata da verdade dos fatos, mas sim de sua validade; pode muito bem ocorrer das premissas serem todas falsas, da conclusão ser falsa e mesmo assim o raciocínio dedutivo ser correto. Vejamos um exemplo:



Exemplo:

1. Todos os planetas são quadrados

2. A Terra é um planeta

Conclusão: A Terra é quadrada.

Este raciocínio está, de fato, correto.

Existem também raciocínios dedutivos que não são válidos, embora todas as premissas e a conclusão sejam verdadeiras. Neste caso as premissas não sustentam a conclusão:

Exemplo:

1. Todo ser humano é um mamífero

2. Toda criança é um mamífero

Conclusão: Toda criança é um ser humano.

Este raciocínio dedutivo não é válido.

Entretanto, nos raciocínios dedutivos, se as premissas forem verdadeiras e o raciocínio for válido, então fatalmente a conclusão também será verdadeira pois, de certo modo, ela já estava contida nas premissas. Assim, em raciocínios dedutivos válidos, de algo verdadeiro nunca se pode concluir algo falso. Por outro lado, de hipóteses falsas pode-se concluir tanto coisas verdadeiras como falsas e tudo isto por meio de um raciocínio dedutivo absolutamente correto:

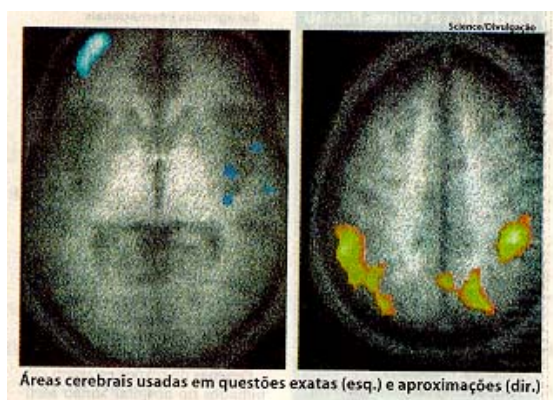
Exemplo:

Admita que  $1 = 2$  e que  $3 = 2$ . Então, somando os membros das igualdades, concluiremos que  $4 = 1 + 3 = 2 + 2 = 4$ .

Porque então distinguir os raciocínios indutivos e dedutivos? Enquanto que o pensamento dedutivo leva a conclusões inquestionáveis, porém já contidas nas hipóteses, o raciocínio indutivo leva a conclusões prováveis, porém mais gerais do que o conteúdo das hipóteses. O entrelaçamento dos argumentos indutivos e dedutivos é a base do método científico.

Existe uma tênue linha que separa os raciocínios indutivos dos dedutivos, se é que eles podem de fato ser separados. Com relação a isto, observe a figura a seguir que apareceu no jornal A Folha de São Paulo, no artigo intitulado "Pesquisa mapeia a Matemática do Cérebro". São retratos de um cérebro em

operação; os cálculos exatos (mais ligados às deduções formais) estão relacionados com a linguagem (áreas azuis) e os cálculos aproximados (mais ligados à indução) estão relacionados com a percepção visual e espacial (áreas verdes).



Vamos apresentar agora dois exemplos que mostram como é interessante e difícil separar o pensamento indutivo do dedutivo.

## A DEDUÇÃO FORMAL PODE CONTRARIAR NOSSA PERCEPÇÃO DE MUNDO

Considere a seguinte argumentação dedutiva:

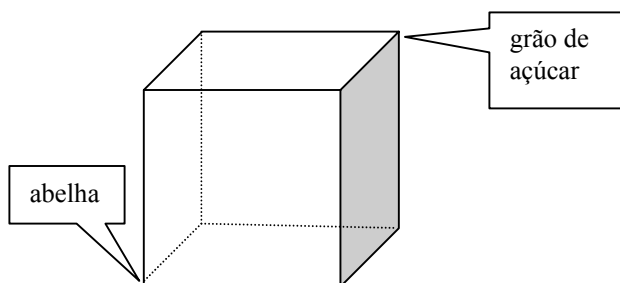


1. Tudo o que é raro é caro
  2. Um carro bom e barato é raro
- Conclusão: Um carro bom e barato é caro.

Pergunta-se: este raciocínio é correto? Note a contradição indutiva presente na conclusão.

## A INDUÇÃO PODE LEVAR A CONCLUSÕES DEDUTIVAS FALSAS

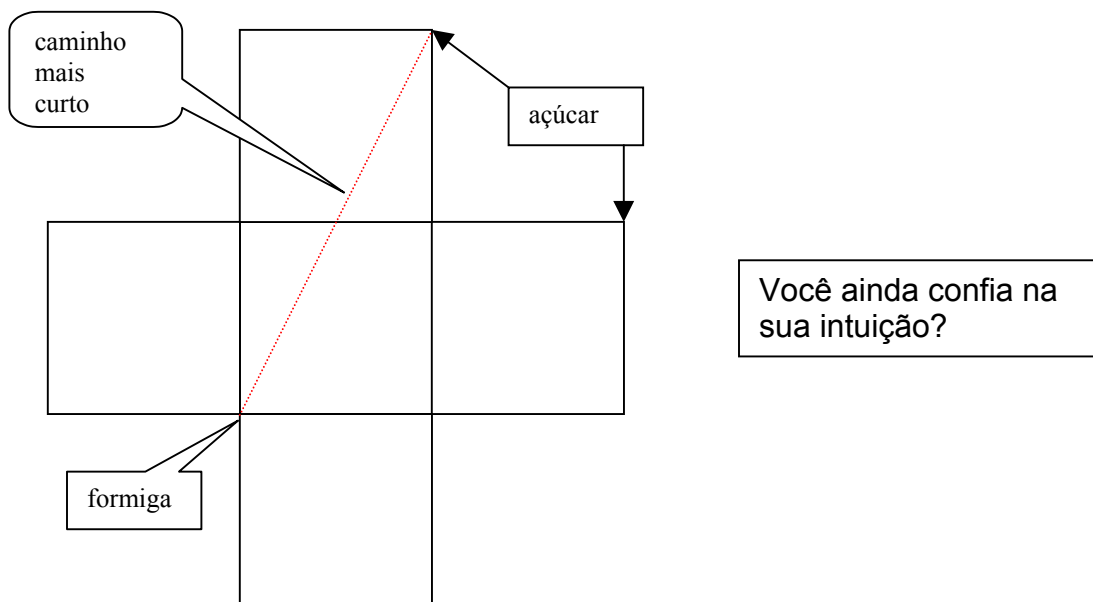
Como exemplo, considere uma sala com o formato de um cubo, na qual encontra-se em um canto uma pequena abelha e no canto diametralmente oposto um pequeno grão de açúcar, como ilustrado na figura abaixo:



Pergunta-se:

- a) Qual é o caminho mais curto para a abelha chegar ao grão de açúcar?
- b) Imagine que no lugar da abelha encontra-se uma formiga. Qual é o caminho mais curto para a formiga atingir o grão de açúcar?

A maioria das pessoas responde que a formiga deve caminhar pela diagonal da base do cubo e depois subir pela aresta vertical que contém o grão de açúcar. Outras soluções de mesma natureza também podem ser citadas, como por exemplo, subir pela aresta vertical e caminhar diagonalmente no teto. As respostas dadas de forma intuitiva estão incorretas. Para verificar o motivo, basta abrir o cubo como se fosse uma caixa de papelão:



## PARADOXOS

Em muitos momentos de nossa vida nos deparamos com situações contraditórias; elas parecem escapar ao senso comum, pois violam nossa intuição e aparentemente nos confundem.

A partir do momento em que a linguagem do ser humano cessou de ser somente um veículo de comunicação e tornou-se, ela mesma, um objeto do discurso, de discussões e portanto de estudos, surgiram os paradoxos.

Dentro do estudo da Lógica, os paradoxos aparecem com frequência, dando impulso ao estudo do funcionamento de nosso raciocínio, de nossas capacidades e limitações. Apresentaremos a seguir alguns paradoxos lógicos que cativam pela simplicidade e podem ser usados para despertar nos alunos o gosto pela aprendizagem da Matemática e suas conexões. Não discutiremos os desdobramentos científicos destas contradições no reino da Lógica, pois isto requer estudos profundos, em que os paradoxos deixam de ser construções que desafiam nossa intuição, passando a ser a ferramenta de trabalho do estudioso.

**Paradoxo do Mentiroso:** Alguém afirma: -- Estou mentindo. Isto é verdadeiro ou falso? Se for verdadeiro a pessoa diz a verdade e portanto mente. Se for falso, o que ela diz não é verdade e portanto ela não mente, isto é, ela fala a verdade.

**Paradoxo da exceção:** Todas as regras têm exceção. Esta é uma regra. Portanto ela deve ter exceção. Logo, deve existir uma regra sem exceção.

**Duplo Paradoxo:** Um professor de Direito fez um acordo com um aluno: o aluno não precisaria pagar pelas aulas dadas por ele se perdesse sua primeira causa, mas se ele ganhasse, deveria pagar pelas aulas assistidas. Terminado o curso, o aluno não aceitou nenhuma causa. Afim de cobrá-lo o professor processou-o. Em sua defesa, o aluno apresentou o seguinte argumento:

-- Ou ganho ou perco a causa; se ganhar não precisarei pagar porque afinal eu ganhei e se perder também não precisarei pagar pois, de acordo com o trato feito com o professor, ao perder a minha primeira causa, também não precisarei pagar.

A que o professor retrucou:

-- Ou ganho ou perco a causa; se ganhar o aluno deverá me pagar, posto que ganhei o embate; se perder ele também deverá me pagar, pois o nosso trato anterior impõe que ele me pague, pois venceu sua primeira causa.

Quem afinal tem razão?

**Paradoxo de Greling:** Um adjetivo é chamado de heterólogo se não possui a propriedade que ele mesmo expressa e é chamado de homólogo se possui. Por exemplo, polissílaba é uma palavra polissílaba e proparoxítona é uma palavra proparoxítona. Estes são exemplos de adjetivos homólogos. Todo adjetivo que não for homólogo deve ser heterólogo.

Heterólogo é um adjetivo. Como os adjetivos são divididos em duas classes (homólogos e heterólogos), a palavra heterólogo deve estar em uma delas. Se o adjetivo heterólogo for heterólogo, ele não possui a propriedade de ser heterólogo e portanto é homólogo. Se ele for homólogo ele possui a propriedade que ele mesmo expressa e portanto é heterólogo.

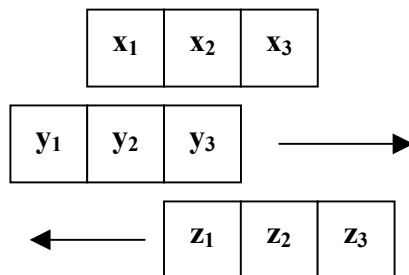
**Paradoxo do barbeiro:** Esta é uma versão folclórica do paradoxo de Russel. Em uma certa ilha existe um só barbeiro e este barbeia todos os homens e somente aqueles que não se barbeiam a si mesmos. Pergunta-se: o barbeiro barbeia a si mesmo ou não? Se o barbeiro se barbeia, entramos em contradição com o fato dele somente barbear aqueles que não barbeiam a si próprios. Se ele não se barbeia, então ele deve se barbear pois esta é a sua tarefa.

## Os paradoxos de Zenão:

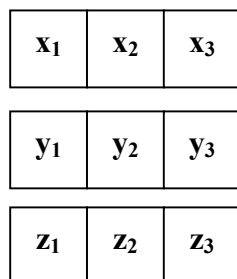
### 1. Aquiles e a tartaruga.

Aquiles, um famoso corredor grego de maratonas, deseja apostar corrida contra uma tartaruga. Inicialmente a tartaruga encontra-se 100 metros à frente do corredor. Mas, não importa quão veloz seja Aquiles e quão vagarosa seja a tartaruga, Aquiles nunca poderá alcançá-la. Iniciada a corrida, quando Aquiles se encontrar no ponto onde a tartaruga inicialmente estava, esta última avançou uma certa distância; quando Aquiles chegar ao novo ponto em que a tartaruga estava, ela avançou mais um pouco. Como este processo pode ser repetido indefinidamente e, em cada estágio, a tartaruga sempre se encontra um pouco à frente de Aquiles, ele nunca conseguirá ultrapassá-la, e o movimento torna-se impossível.

### 2. O espaço e o tempo não podem, por divisão, transformarem-se em quantidades indivisíveis. Admitamos que existisse uma unidade de tempo que não pudesse ser subdividida. Tomemos três pequenas fitas com o mesmo tamanho (de papel, por exemplo) divididas em pequenas casas, inicialmente dispostas como na figura:



A fita  $x_1, x_2, x_3$  ficará fixa. A fita  $y_1, y_2, y_3$  move-se para a direita de modo que cada  $y_i$  passa  $x_i$  no menor instante de tempo possível (indivisível). Simultaneamente, a fita  $z_1, z_2, z_3$  move-se para a esquerda de modo que cada  $z_i$  passa um  $x_i$  no mesmo instante de tempo. Deste modo, depois de passado um instante de tempo, as fitas terão a seguinte disposição:

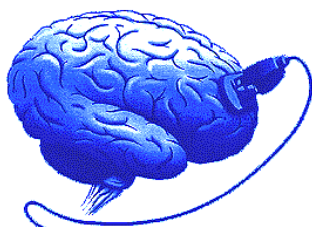


Observemos que  $z_1$  passou por dois  $y_i$ 's. Portanto o instante de tempo não pode ser o menor possível, pois existiria um instante de tempo ainda menor, aquele gasto quando  $z_1$  passou por um dos  $y_i$ 's.

Existem muitas outras contradições (aparentes ou não) que surgem nas ciências e que servem de mola propulsora para as teorias científicas. Os paradoxos da Física Quântica, a Teoria do Caos e principalmente os paradoxos que podem ser formulados dentro da Lógica Matemática são manifestações deste fato. Cada uma delas exige um estudo detalhado, que foge dos objetivos deste texto, mas que pode ser trilhada pelo aluno interessado.

Entretanto, vamos apresentar alguns aspectos folclóricos de algumas situações aparentemente contraditórias. Vejamos então algumas idéias conflitantes que merecem o prêmio Ignobel:

**Nas relações sociais:** Eu nunca ficaria sócio de um clube que me aceitasse como sócio (Grouxo Marx)



**Moto Perpétuo:** Como todos sabem, a energia mecânica pode ser transformada em energia elétrica com o auxílio de um dínamo e reciprocamente, a energia elétrica se converte em energia mecânica com o uso de um motor. Seria possível fabricar um dínamo que gerasse energia elétrica que pudesse ser usada para girar o próprio dínamo, como se fosse um motor? Tente ligar a tomada de um liquidificador no próprio aparelho.

**O mal existe no mundo:** Se Deus é onipotente, ele pode fazer absolutamente tudo. Então ele pode criar uma pedra tão pesada que Ele mesmo não pode carregar e portanto existe uma coisa que Ele não pode fazer (carregar a pedra). Logo Deus não é onipotente, contrariando nossa hipótese inicial.



**O dissolvente universal:** No tempo dos alquimistas, um inspirado iniciante na arte da magia científica anunciou que tinha descoberto o dissolvente universal, isto é, uma substância que podia dissolver absolutamente tudo. Esta substância nunca foi apresentada ao público pois ela não pôde ser guardada em nenhum recipiente.

**Paradoxo na propaganda:** Não leia este anúncio.

**Escolinha de Alfabetização:** Para se inscrever na Escola, basta escrever para o endereço ABC, número 0.

**Ruindows Millenium:** Na tela do computador aparece a mensagem: Problema de comunicação com o teclado, para continuar pressione qualquer tecla.

**Paradoxo lingüístico:** Esta frase contém três erros. Qual é o terceiro erro? (Observe o significado semântico da sentença).

**Sintática em guerra com a semântica:** A frase abaixo é verdadeira ou falsa?

**ESTA SENTENÇA CONTÉM SEIS PALAVRAS**

Claramente ela é falsa, pois contém apenas 5 palavras. Se ela é falsa então sua negação

**ESTA SENTENÇA NÃO CONTÉM SEIS PALAVRAS**

deve ser verdadeira. Mas alguma coisa deve estar errada pois a última frase contém, de fato, seis palavras.

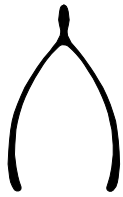
**Paradoxo estatístico:** 100% dos entrevistados disseram que as pesquisas são inúteis. Quer pesquisa mais inútil do que esta?

**Paradoxo na política:** Um político mentiu durante toda a sua vida e, estando na eminência de se aposentar, arrependeu-se disto e, como uma confissão, afirmou:

-- Eu sempre menti e este fato pode ser provado.

Seus adversários políticos, quiseram mostrar que ele realmente havia mentido a vida toda, mas tiveram que a seguinte dificuldade: se ele de fato sempre mentiu, então esta sua última declaração deveria também ser mentirosa, ou seja, ou ele não havia mentido a vida toda, ou se houvesse mentido, tal fato não poderia ser provado. Por outro lado seus amigos também tiveram dificuldade em mostrar que ele não havia mentido a vida toda: de fato sua última declaração deveria ser verdadeira, mas isto sendo uma confissão, acaba afirmando que de fato ele sempre mentiu. Observe que isto nos leva ao seguinte fato curioso: a afirmação "ele sempre mentiu" é uma frase que não pode ser provada, nem sua negação pode ser provada, mas ela é verdadeira, primeiro porque ele sempre mentiu, incluindo quando disse que a frase acima poderia ser provada.

**Em uma aula de Português:** -- Professora, é verdade que preposição é uma coisa muito feia de se terminar a frase com?



**O osso da sorte:** Minha avó gostava de disputar comigo o osso da sorte, quando comíamos frango assado. Cada um fazia o seu desejo, e, segurando cada uma das extremidades, quebrávamos o osso. Quem ficava com o pedaço maior do osso tinha seu pedido atendido, a outra pessoa não. Uma certa vez eu ganhei a disputa e perguntei à minha avó o que ela havia desejado. Ela me disse que tinha desejado que eu ganhasse, o que de fato aconteceu. Minha avó perdeu ou ganhou? E se ela tivesse ficado com o pedaço de osso maior, teria perdido ou ganhado?

**Piaget e o cachorro:** Peça a uma criança na fase pré-operatória (com idade inferior a 5 anos), que ela responda à seguinte argumentação: Todo cachorro late. Eu lato. Então o que podemos concluir?







### 3. MÁQUINAS QUE APRENDEM

*Máquinas que jogam, aprendem a jogar e ganham de humanos.*

**Resumo:** Serão construídos dois computadores feitos com caixas de fósforos, que explicitam como uma máquina pode aprender. Eles são baseados na teoria da evolução de Darwin. O primeiro destes computadores joga um simplificado jogo de xadrez que paulatinamente começa a ganhar, quando joga com adversários humanos. O segundo computador joga o Jogo de NIM com palitos. Este é um jogo estratégico que tem interesse lógico e pedagógico, além de estabelecer conexões com o sistema binário de números.

#### UM COMPUTADOR FEITO COM CAIXAS DE FÓSFOROS

*Ele joga xadrez, aprende a jogar e começa a ganhar*

##### **Materiais:**

- 37 caixas de fósforos vazias
- Canudos de refrigerante de 4 cores diferentes (4 de cada cor: azul, vermelho, verde e amarelo)
- Configurações da máquina (anexas)
- 1 Tabuleiro (anexo)
- 3 fichas brancas e 3 fichas pretas (anexas)

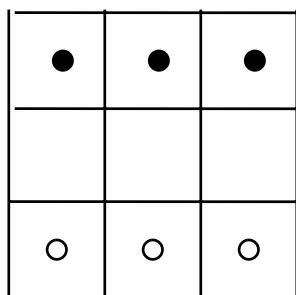
##### **Descrição:**

Este protótipo de computador aprende a jogar uma modalidade de jogo de xadrez contra adversários humanos. No início, ele perderá sempre, mas a cada jogo aprenderá como não perder até o ponto em que vencerá invariavelmente.

Esta máquina é concebida para jogar apenas um jogo simples que descreveremos a seguir:

O jogo é jogado em um tabuleiro com nove casas e seis fichas - três pretas e três brancas, como no desenho a seguir:

Avançam as pretas ↓

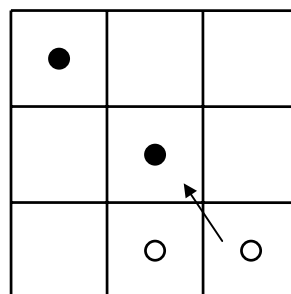
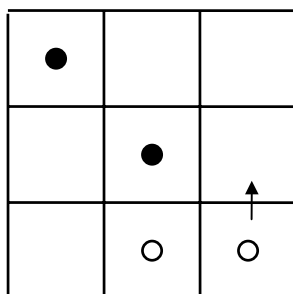


Tabuleiro mostrando a posição inicial

Avançam as brancas ↑



O jogo começa com as três pedras pretas em frente às três brancas, em lados opostos do tabuleiro. As pedras devem ser movidas como no jogo de xadrez, isto é, podem mover-se uma casa para a frente, para um quadrado vazio, ou uma casa na diagonal, para capturar uma peça da cor oposta, a qual é então removida do tabuleiro. Observe no desenho a seguir dois lances possíveis para as peças brancas durante um jogo:

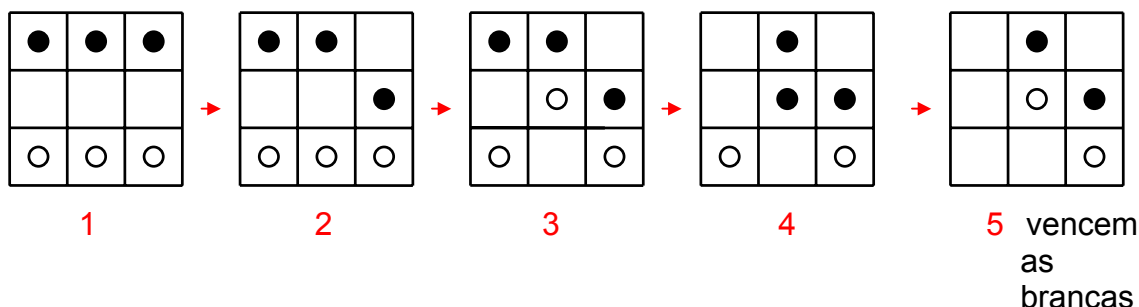


Um jogo é ganho em três situações:

- 1) quando uma peça atinge o lado oposto do tabuleiro.
- 2) quando todas as peças adversárias são capturadas.
- 3) quando todas as peças adversárias ficam numa posição que não podem mover-se.

A seguinte seqüência de esquemas mostra um jogo em que as pedras pretas avançam primeiro. Cada um dos desenhos a seguir mostra as posições das peças após a jogada anterior.

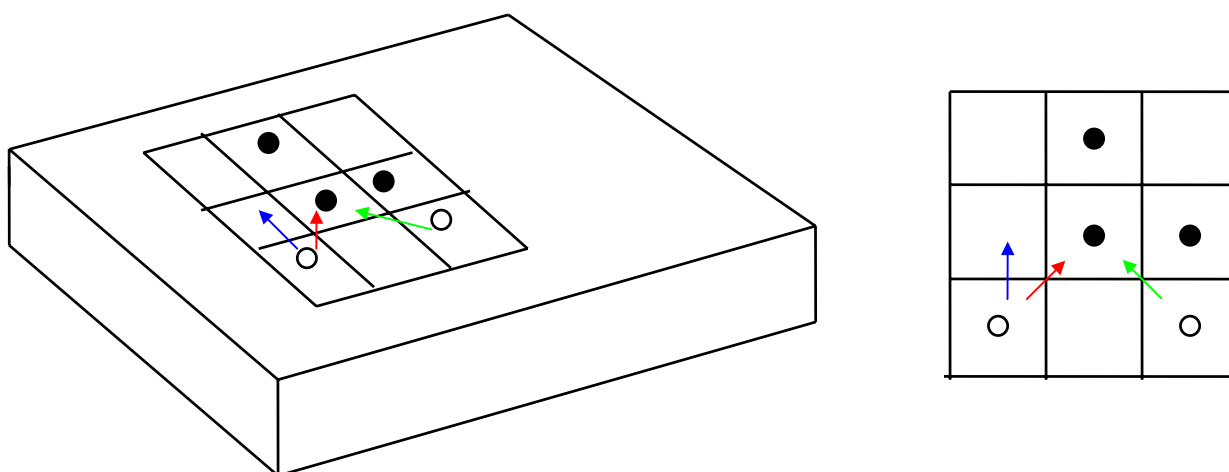
Saem as pretas



Este é um jogo muito simples e bastam alguns lances para decidir quem é o vencedor. Isto é necessário para manter pequenas as dimensões deste computador, mas nada impede que o jogo seja ampliado ou que apliquemos o mesmo princípio a outros jogos.

### As caixas de fósforos e como decidir um lance:

O computador que será confeccionado consiste de 37 caixas de fósforos, uma para cada posição que o jogo pode tomar. Apresentamos a seguir uma típica caixa de fósforos que representa uma posição do jogo. Estão desenhadas com setas coloridas todos os possíveis movimentos a partir das posições representadas. Estão representados três movimentos possíveis indicados pelas cores azul, vermelho e verde.



Dentro da caixa estão pedaços de canudinho da mesma cor que a das setas (no exemplo acima azul, vermelho e verde).

Quando chega a vez do computador jogar, o operador seleciona a caixa de fósforos apropriada, agita-a e, sem olhar, tira de dentro dela um pedaço de canudinho. A cor do canudinho indica o lance do computador. Inicialmente, o computador é um completo idiota, podendo fazer jogadas que serão perdedoras, pois em cada caixa é colocada exatamente um canudinho para cada lance possível. No início o computador faz muitas jogadas tolas, mas ele pode ser ensinado a jogar de modo eficiente por um sistema de recompensas e castigos.

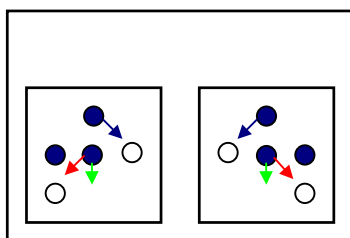
### **Como programar o computador para que ele aprenda a vencer:**

Quando o computador perde, como acontece quase que inevitavelmente no início, ele é castigado, retirando-se da caixa de fósforos correspondente o canudinho que representa o seu último lance, impedindo assim sua repetição. Quando o computador encontra uma caixa vazia, deve-se remover o canudinho que corresponde à última jogada efetuada e neste caso o computador perde, desistindo do jogo.

A medida que mais e mais jogos são jogados contra o computador, ele perde a capacidade de realizar maus lances, aumentando assim suas possibilidades de vitória.

Quando o computador ganha, ele recebe um incentivo, sendo-lhe dado um canudinho extra para cada uma das caixas de fósforos que produziram a vitória (assim toda a seqüência de lances que levou à vitória é premiada). Deste modo, a probabilidade de repetir os movimentos ganhadores aumenta a cada jogo.

Para diminuir o número de caixas de fósforos utilizada no jogo, colocamos em cada caixa uma posição e sua simétrica, como no exemplo:



Quando você tiver construído e ensinado o seu computador, faça um desafio com algum conhecido. Se ele também possuir tal aparato, é possível fazer com que os computadores se desafiem um ao outro.

Procure raciocinar sobre os mecanismos lógicos utilizados nesta construção, bem como sua aplicabilidade em outros jogos ou atividades educacionais.

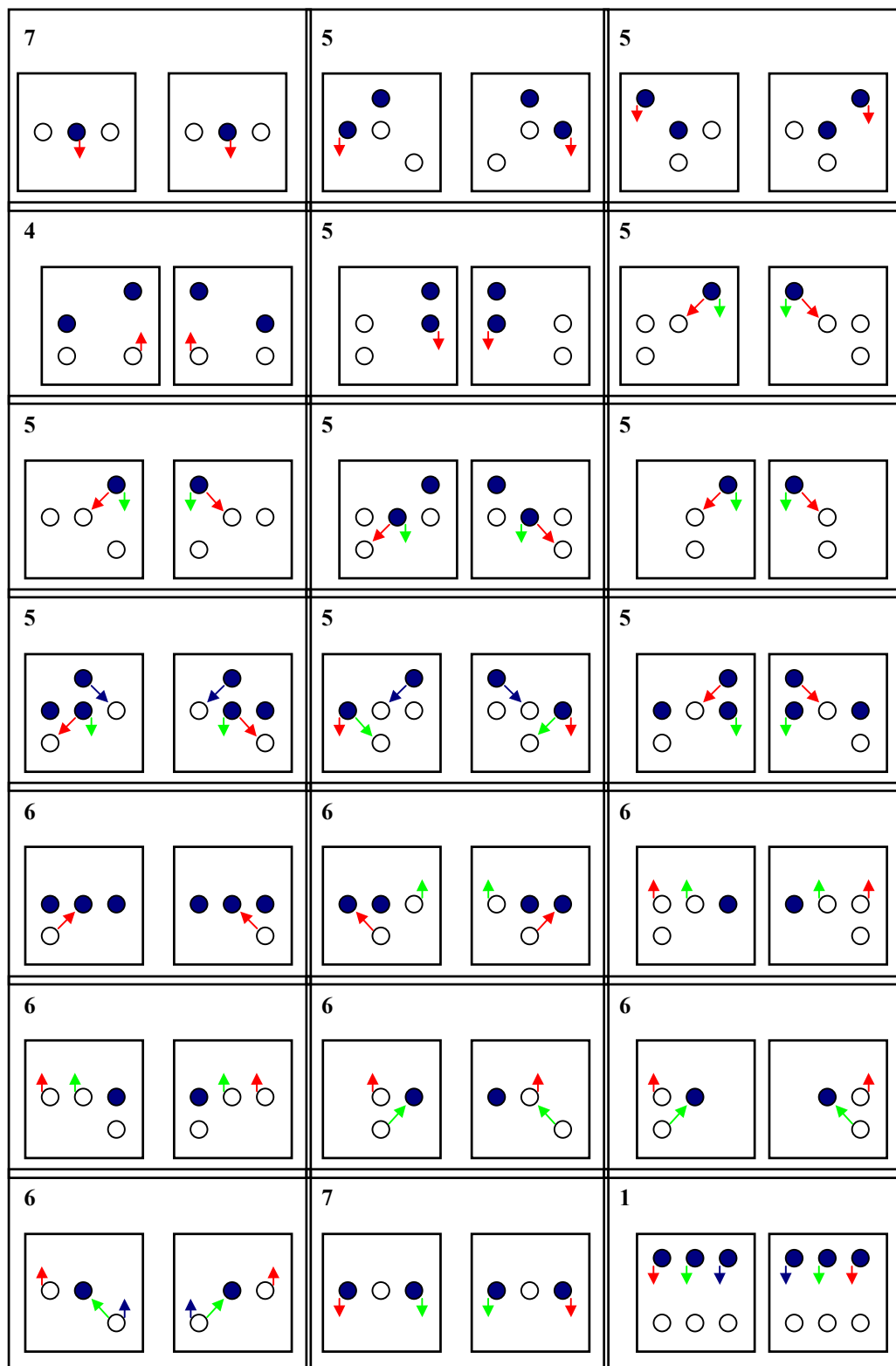


Jogue mais algumas vezes com seu computador e observe como ele devagarinho começa a ganhar. A “programação” da máquina segue drasticamente a teoria da evolução formulada por Charles Darwin e o experimento acima indica a possibilidade de se fabricar máquinas que geneticamente modificam-se para sobreviver. Esta linha de trabalho tem produzido experiências fascinantes, em que se criam várias formas de vida “in silico”, totalmente artificiais. Existem muitos sites na Internet dedicados a este assunto. Procure acessá-los e adentre-se no mundo artificial!

A maneira com que o computador anterior foi construído pode ser adaptado para se fabricar máquinas que jogam qualquer outro tipo de jogo, desde que não sejam jogos de azar. Veja também o computador Gabriela I, desenvolvido pela Fundação Brasileira para o Desenvolvimento do Ensino de Ciências. o jogo ainda é o mesmo, mas a dinâmica das jogadas é ligeiramente diferente.

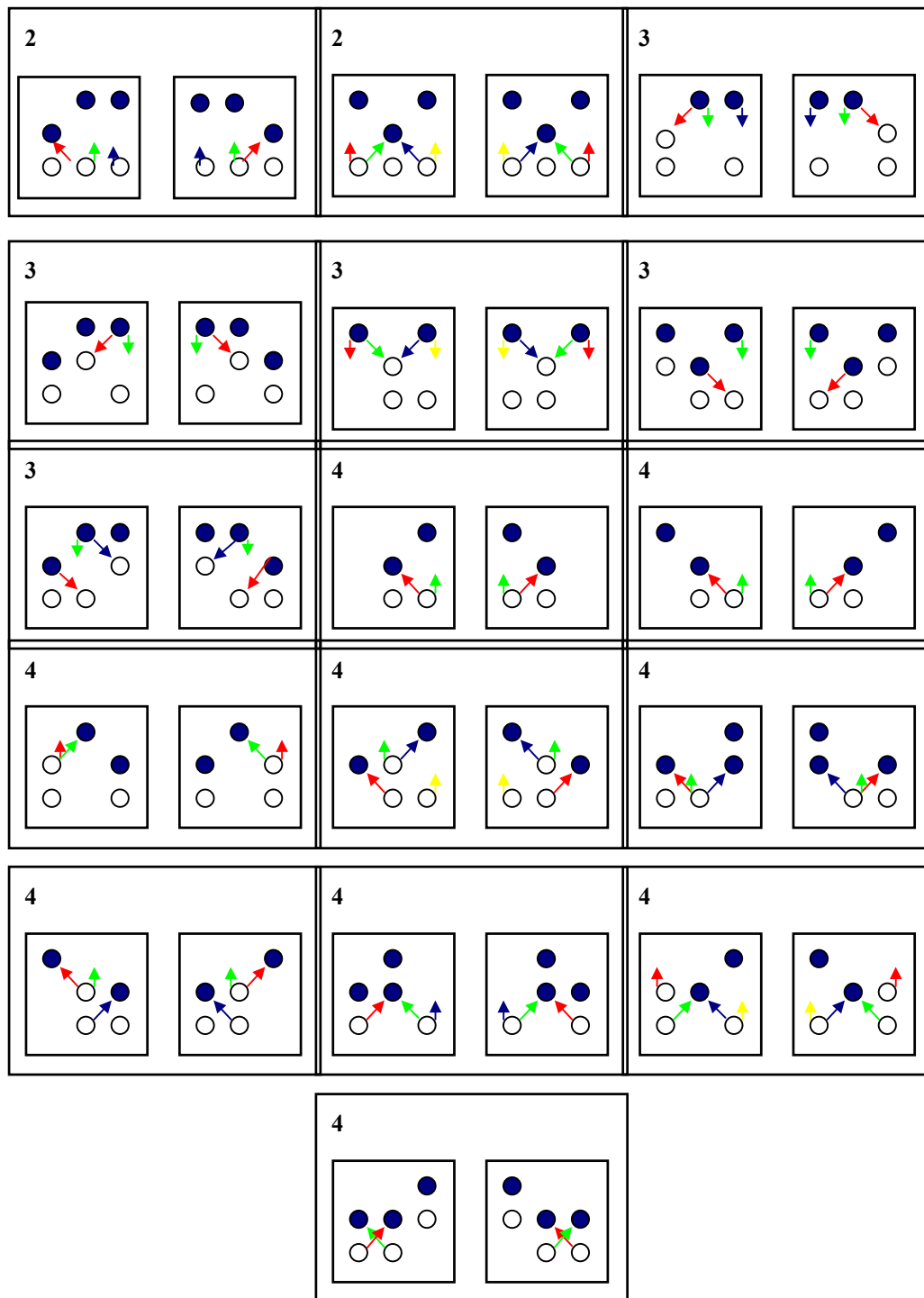
Nas páginas seguintes encontram-se impressos os materiais necessários para montar o computador que joga xadrez. A seguir, apresentaremos um novo jogo (NIM) que também será jogado com caixas de fósforos, seguindo os mesmos princípios já desenvolvidos, mas com estratégias mais desafiadoras.







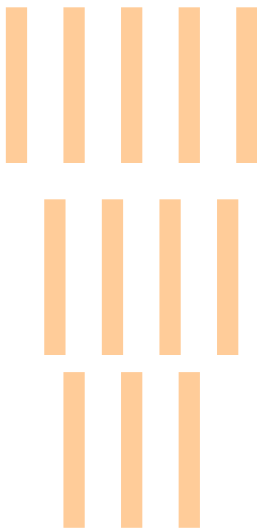






## JOGO DE PALITOS - NIM

O mais interessante dos jogos com palitos talvez seja o Jogo de Nim, que possui origem na China Antiga. Em geral ele é jogado com três fileiras de palitos, dispostos da seguinte maneira



O jogo é realizado com dois jogadores, digamos Alfa e Beta. Cada jogador deve, na sua vez, retirar a quantidade que desejar de palitos, mas de uma linha de cada vez. Não é permitido a remoção de palitos de duas linhas diferentes na mesma jogada. Quem ficar com o último palito perde.

Apesar de se ter regras simples, é muito difícil resolver este jogo, descrevendo explicitamente todas as estratégias vencedoras. Se nenhum dos dois jogadores conhece as estratégias que levam à vitória, então o jogador mais experiente tenderá a vencê-lo. Depois de umas poucas jogadas, algumas estratégias vencedoras ficarão logo claras. Por exemplo, se Alfa joga e deixa na mesa a configuração 1-2-3 ele, sabendo jogar, fatalmente vencerá. De fato, se a seguir Beta joga deixando 0-2-3, Alfa joga deixando 0-2-2. Beta joga novamente deixando 0-1-2 e Alfa finaliza com 0-1-0 ganhando o jogo, pois Beta ficou com o último palito.

**Jogo de Nim com duas fileiras:** Neste caso o jogo é muito mais simples. É fácil ver que, para vencer, basta igualar a quantidade de palitos nas duas filas. Alfa é forçado a fazer com que as pilhas fiquem iguais e Beta com que elas fiquem diferentes.

Algumas situações vencedoras: Voltemos ao jogo com três fileiras. Mesmo que o jogador não conheça a estratégia vencedora, ele ainda pode adotar certas práticas que levam à vitória. Nunca se pode deixar que o oponente deixe na mesa duas fileiras com o mesmo número de palitos e deve-se esperar a oportunidade de se obter a configuração 1-2-3, ou impedir que o adversário consiga isto. Conhecendo-se estas estratégias você é praticamente invencível, até que encontre um mestre no jogo, que conheça o sistema completo de situações que levam à vitória.

## O JOGO DE NIM E O SISTEMA BINÁRIO

Vamos analisar agora o jogo de Nim com três fileiras, colocando em cada fileira o número que quisermos de palitos (diferente de zero) a fim de iniciarmos o jogo. Para descrever as estratégias vencedoras, devemos escrever o número de palitos em cada linha na notação binária, um número embaixo do outro, alinhando o dígito das unidades, das “dezenas na base 2”, das “centenas na base 2”, etc. Exemplos:

### Caso 1)

81 palitos →   
 77 palitos →   
 28 palitos → 

Na base 2, temos:

1	0	1	0	0	0	1
1	0	0	1	1	0	1
		1	1	1	0	0

$$81 = 64 + 16 + 1 = 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 0 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$77 = 64 + 8 + 4 + 1 = 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

$$28 = 16 + 8 + 4 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$$

### Caso 2)

108 palitos →   
 88 palitos →   
 46 palitos → 

Na base 2, temos:

1	1	0	1	1	0	0
1	0	1	1	0	0	0
	1	0	1	1	1	0

$$108 = 64 + 32 + 8 + 4 = 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$$

$$88 = 64 + 16 + 8 = 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$$

$$46 = 32 + 8 + 4 + 2 = 1 \cdot 2^4 + 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$$

Depois de escrever os três números no sistema binário, nós somamos os dígitos da mesma coluna. Esta soma será chamada de dígito-soma. Se todos os dígitos-soma forem pares, diremos que estamos numa situação par. Se algum dígito-soma for ímpar, estaremos numa situação ímpar. Vejamos nos exemplos anteriores:

Caso 1)

1	0	1	0	0	0	1
1	0	0	1	1	0	1
		1	1	1	0	0
2	0	2	2	2	0	2

Linha dos dígitos-soma  
Neste caso estamos na situação par

Caso 2)

1	1	0	1	1	0	0
1	0	1	1	0	0	0
	1	0	1	1	1	0
2	2	1	3	2	1	0

Linha dos dígitos-soma  
Neste caso temos uma situação ímpar

Como tomar o último fósforo significa a derrota, jogar e deixar na mesa uma situação par leva à vitória, deixar a mesa numa situação ímpar leva à derrota, exceto quando a mesa ficar 1-1-1 ou 0-0-1. A regra para o jogo com somente duas fileiras está incluída aqui, considerando uma linha formada somente por zeros. Para vencer com duas fileiras, vimos que o número de palitos em cada uma delas deve se tornar igual, mas isto significa que os dígitos-soma são iguais pois aparecem repetidos. Estamos numa situação par e portanto deixar a mesa com duas fileiras contendo o mesmo número de palitos é uma estratégia vencedora.

Analise as seguintes afirmações:

1 A situação final que produz uma vitória é par ( 0 – 0 – 0 ) (quem recebe a mesa vazia ganha, pois o jogador ficou com o último palito)

2 Uma situação par não pode ser mudada para uma outra situação par em apenas uma jogada

3 Uma situação ímpar pode sempre ser mudada para uma situação par com apenas uma jogada.

Estas tarefas são simples, a grande dificuldade aqui está em estabelecer conexões entre este jogo de palitos e o sistema binário. Parece coisa da China! Até hoje não se sabe quem estabeleceu esta conexão, ela é belíssima.

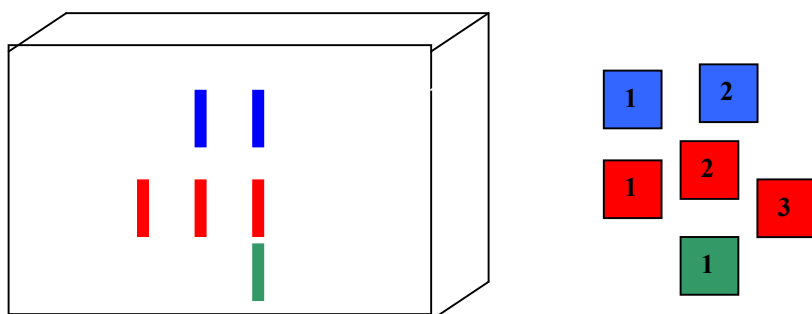
O jogo de Nim permite muitas variações, descreva uma delas. Descreva também outros jogos com palitos em que a estratégia matemática é mais importante que a sorte.

Para jogar o jogo de Nim no computador acesse [www.dm.ufscar.br](http://www.dm.ufscar.br) – hipertexto Pitágoras.

## COMPUTADOR QUE JOGA NIM

**Materiais:** 18 caixas de fósforos vazias e papel cartão grosso de três cores diferentes (azul, vermelho e verde).

A montagem deste jogo é semelhante ao do jogo de xadrez com peões descrito anteriormente. Devemos colocar do lado de fora das caixas de fósforos todas as jogadas possíveis, identificadas com diferentes cores (digamos azul, verde e vermelho). Dentro de cada caixinha deverão estar pequenos quadrados de papel cartão com as mesmas cores do lado externo e numeradas com números 1, 2 ou 3 que indicam quantos palitos daquela cor devem ser retirados. Por exemplo, dentro da caixa abaixo devem estar os cartões ao lado:

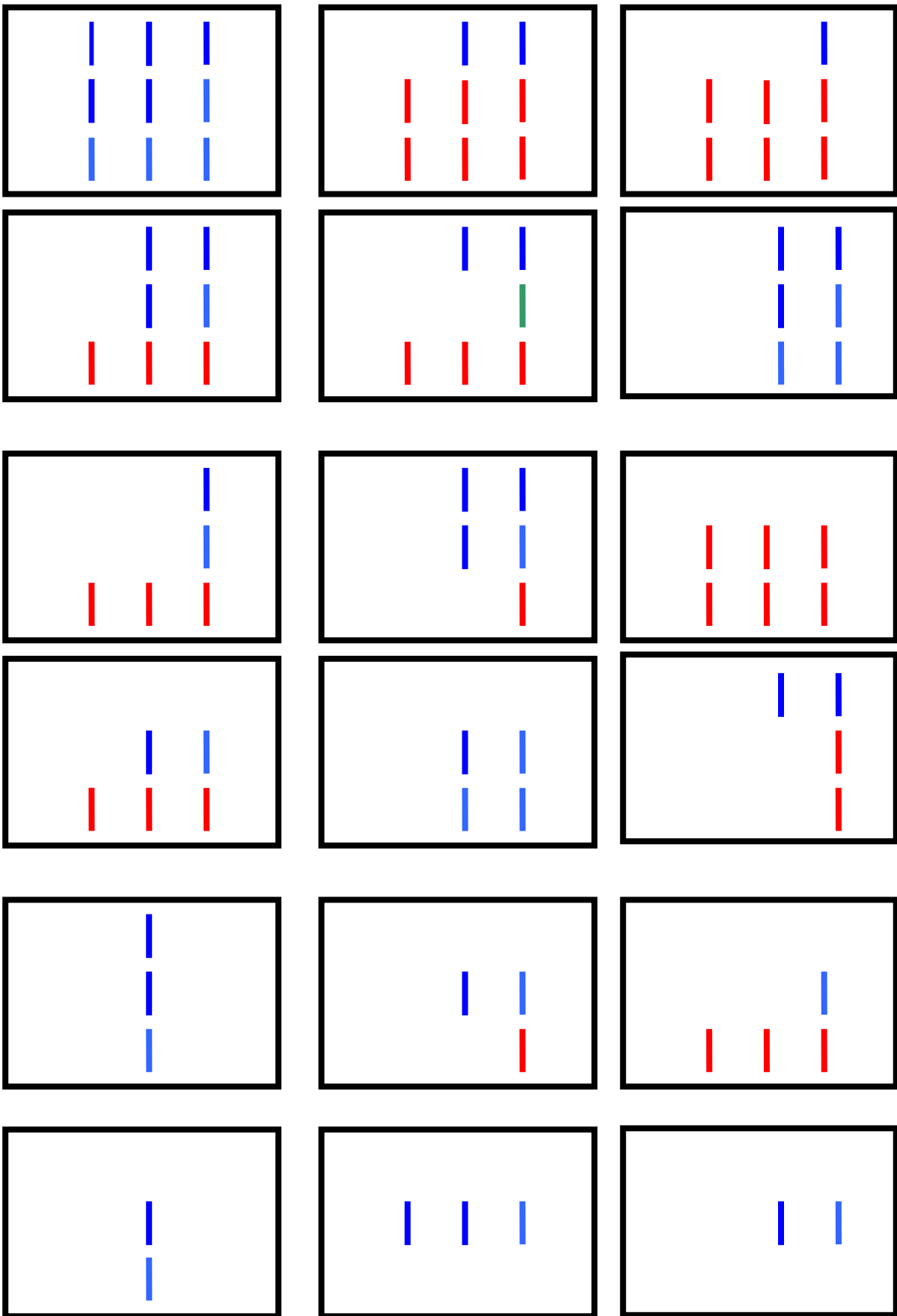


Para jogar, agitamos a caixa e, sem olhar escolhemos uma das seis pequenas fichas. O número e cor escolhidos nos indicam a jogada a ser feita. Por exemplo, se a ficha escolhida for



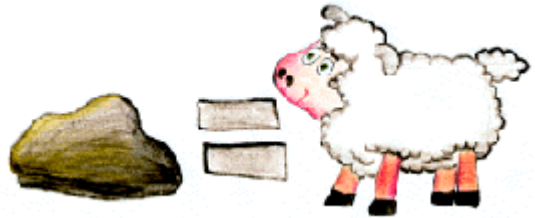
devemos retirar da primeira fileira os seus dois palitos. Caso o computador perca uma jogada, retiramos a última ficha que levou à jogada perdedora. Caso ele ganhe alimentamos toda a seqüência vencedora, desde a primeira jogada, para que ela “aprenda” a jogar e vença novamente.

Aqui estão as caixas necessárias para fabricar o computador que joga Nim:









## 4. MÁQUINAS DE CALCULAR DO ARCO DA VELHA

*Como calcular sem eletricidade*

**Resumo:** Serão construídas máquinas que realizam operações matemáticas: uma primeira que faz a soma binária usando água, uma segunda que realiza a soma de inteiros através de um aparato geométrico e uma terceira que usa uma lata para somar inteiros. A seguir serão apresentadas três máquinas que realizam multiplicações: réguas com espaçamento logaritmico, uma calculadora que usa as propriedades da parábola para cálculos e uma última baseada na idéia dos ossos (ou barras) de Napier. Esta seção encerra-se com uma pequena descrição das principais ferramentas computacionais para o cálculo, segundo o ponto de vista histórico.

### SOMANDO COM ÁGUA

É possível fazer uma calculadora que funcione com água. Um pequeno projeto encontra-se esboçado a seguir. A máquina consegue somar dois números na base 2, isto é ela consegue, quando convenientemente alimentada com as entradas corretas, efetuar as seguintes somas na base 2:

$$0 + 0 = 0$$

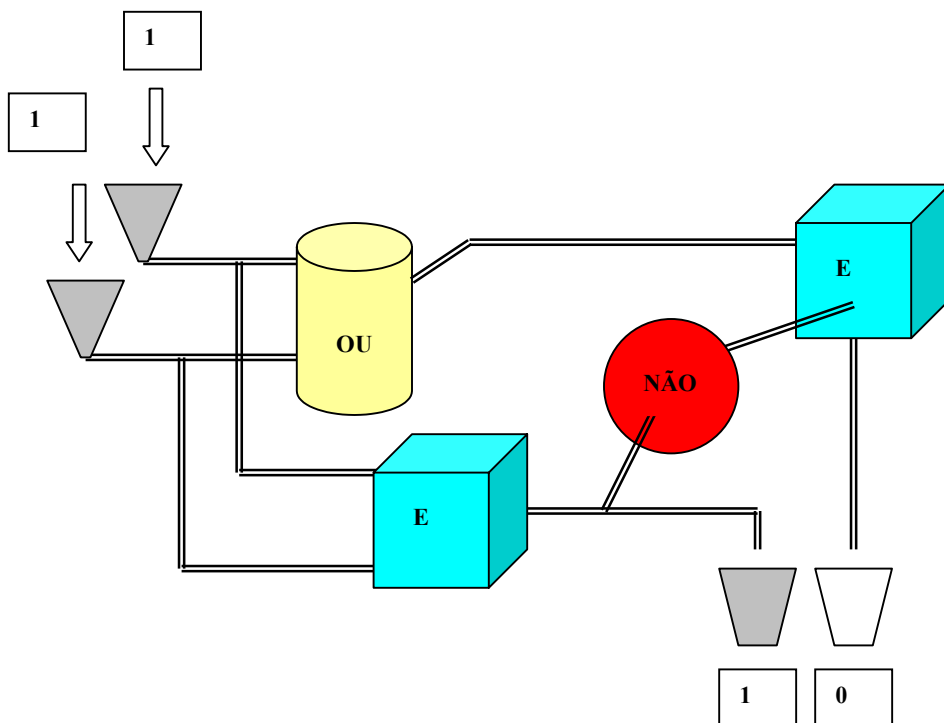
$$1 + 0 = 1$$

$$0 + 1 = 1 \text{ e}$$

$$1 + 1 = (10)_2$$

(lembre-se que  $(10)_2 = 1 \cdot 2^1 + 0 \cdot 2^0 = 2$ ). Deste modo ela consegue realizar todas as possíveis adições na base 2 de números que só contenham um dígito.

Nesta máquina as entradas são representadas pela adição de água nos funis, o número 1 é representado pela água suja (em cinza) e o número 0 pela água límpida (branco). Observe que entrando com os dados 1 e 1 obteremos como saída 10, que é 2 na base 2.



## MÁQUINA DE SOMAR DO FRED FLINTSTONE

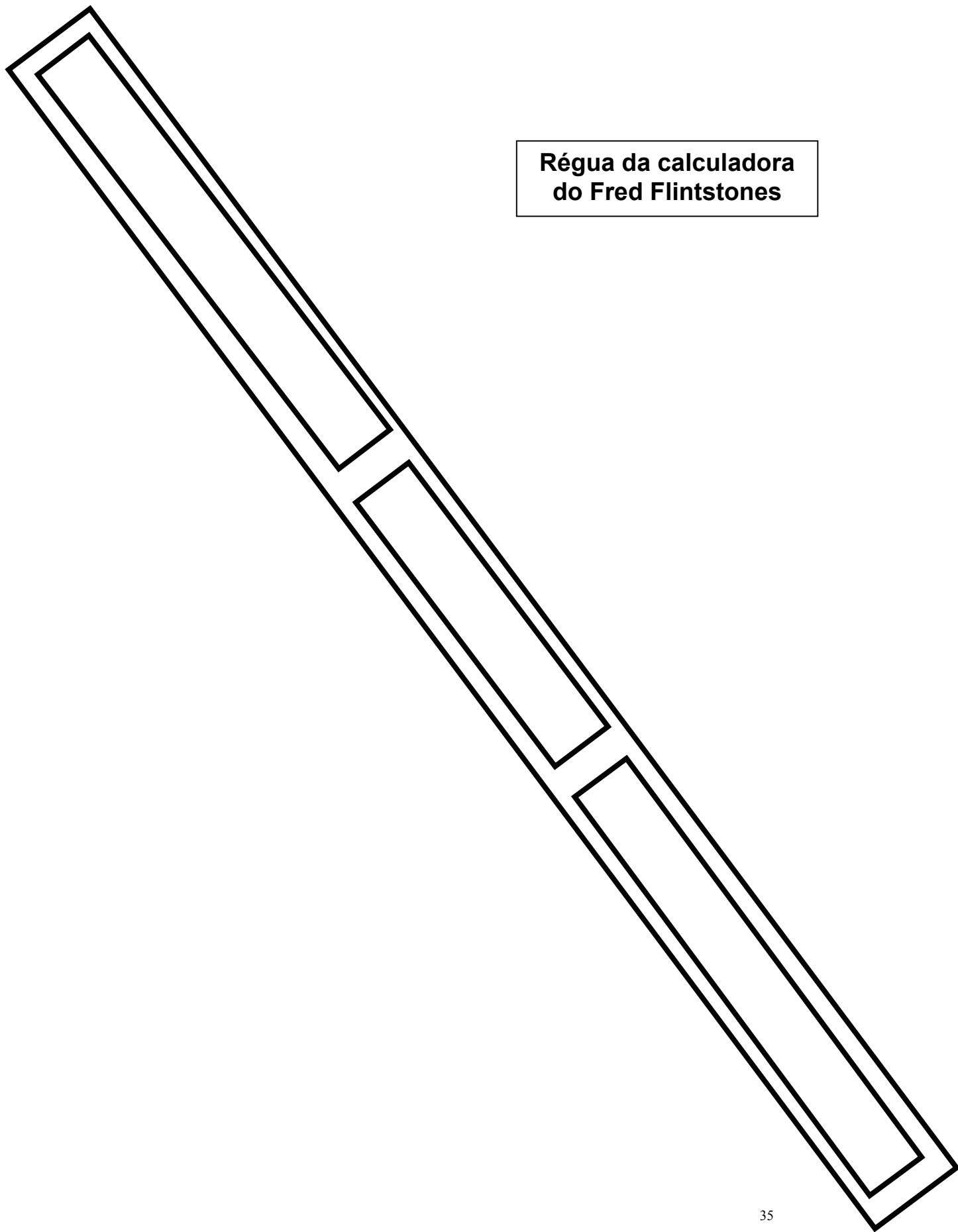
Observe a máquina abaixo constituída por três retas paralelas e uma transversal. Unindo-se dois números nas paralelas externas, teremos a leitura da soma desses números na paralela do meio. Ela também pode realizar a diferença de dois números. Você sabe como isto pode ser feito?

# Calculadora do Fred Flintstones

	7	
3	6	3
	5	
2	4	2
	3	
1	2	1
	1	
0	0	0
	-1	
-1	-2	-1
	-3	
-2	-4	-2
	-5	
-3	-6	-3
	-7	
1ª Parcela	Resultado	2ª Parcela



**Régua da calculadora  
do Fred Flintstones**

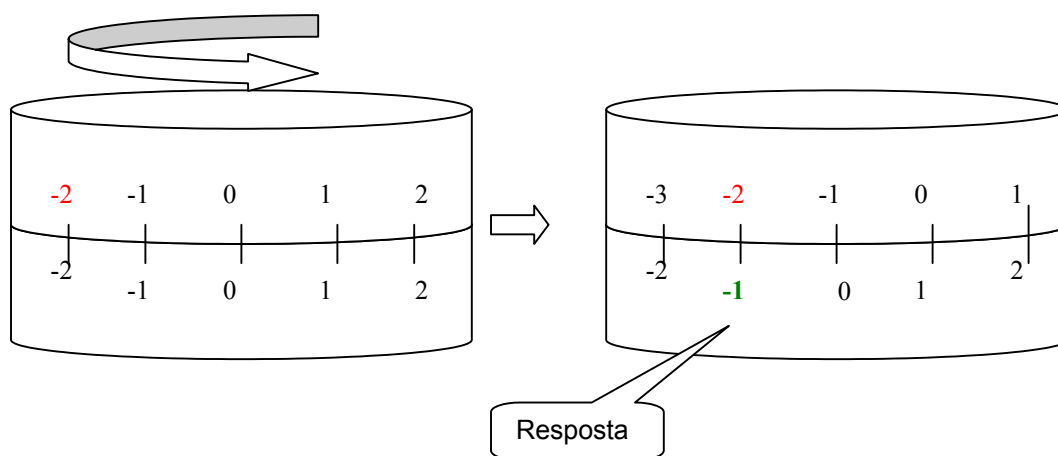




## A LATA DE CALCULAR:

Faça duas réguas de papel numeradas com números inteiros que tenham o comprimento da circunferência de uma pequena lata com tampa. Cole as réguas de papel em torno da lata e de sua tampa. A lata ficará fixa e a tampa girará.

Para realizar uma operação, digamos  $(-2) + 1$ , ajustamos os zeros das duas réguas e olhamos na régua da tampa o primeiro fator  $(-2)$ , giramos a tampa no sentido anti-horário uma unidade (que é o segundo fator). O local onde o número 2 da tampa foi parar será o resultado da operação:



## MULTIPLICANDO COM RÉGUAS:

A próxima máquina que veremos é uma sofisticação da máquina de somar do Fred Flintstone, substituindo a escala linear pela escala logarítmica. Do mesmo modo que a anterior, podemos efetuar multiplicações (e divisões), ajustando os números que queremos multiplicar nas colunas externas e lendo o resultado de sua multiplicação na coluna do meio.

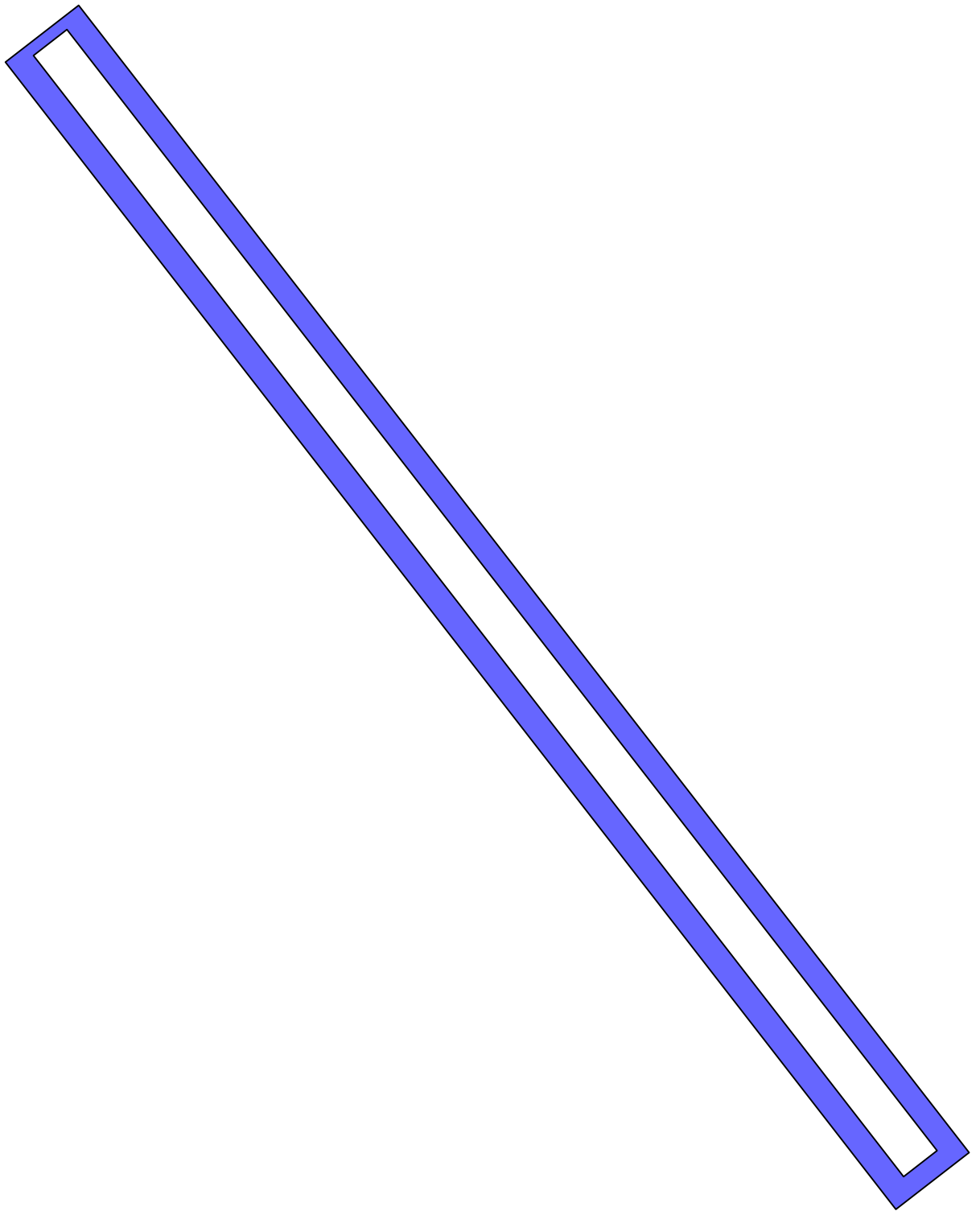




## MÁQUINA DE MULTIPLICAR



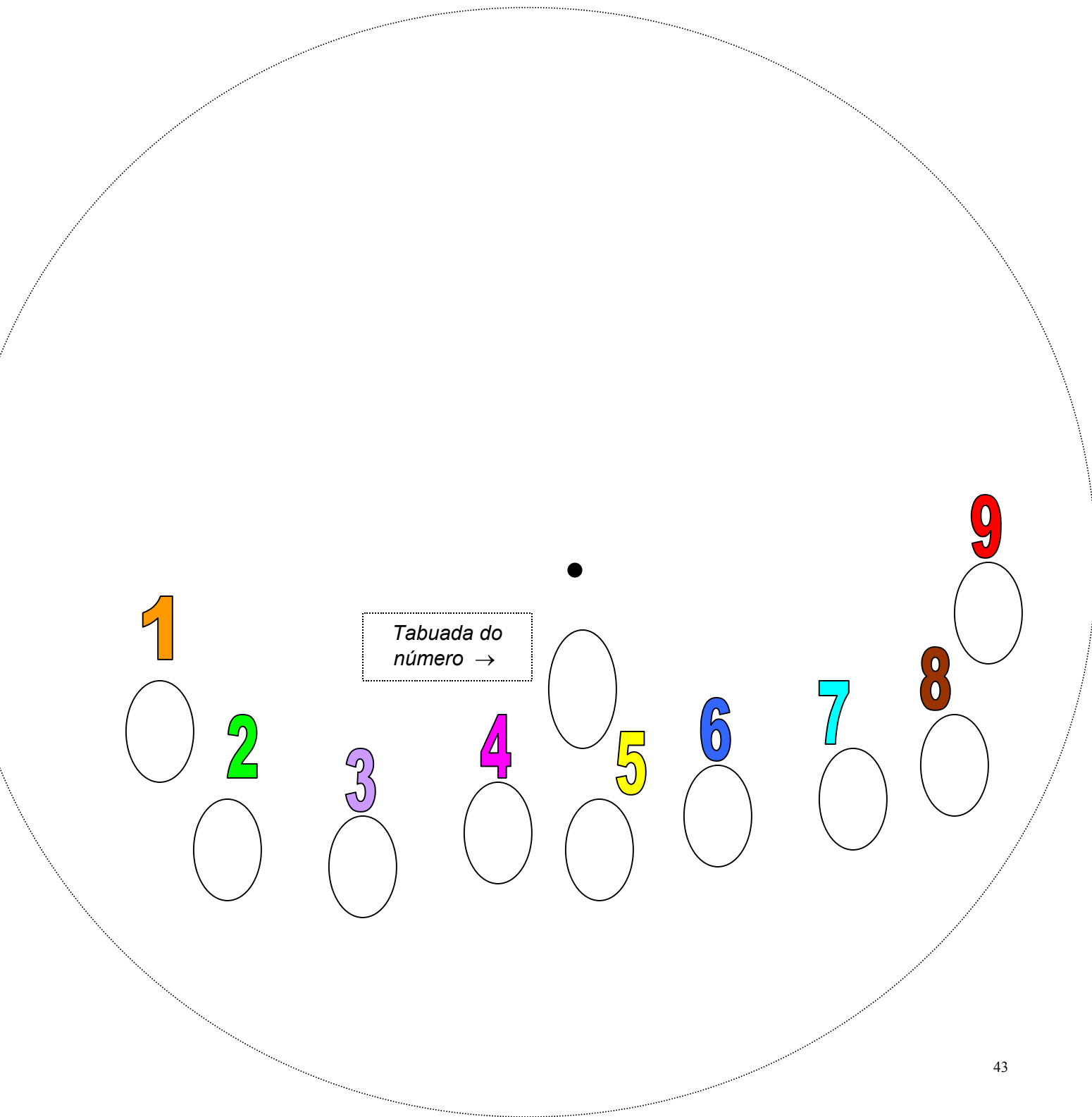




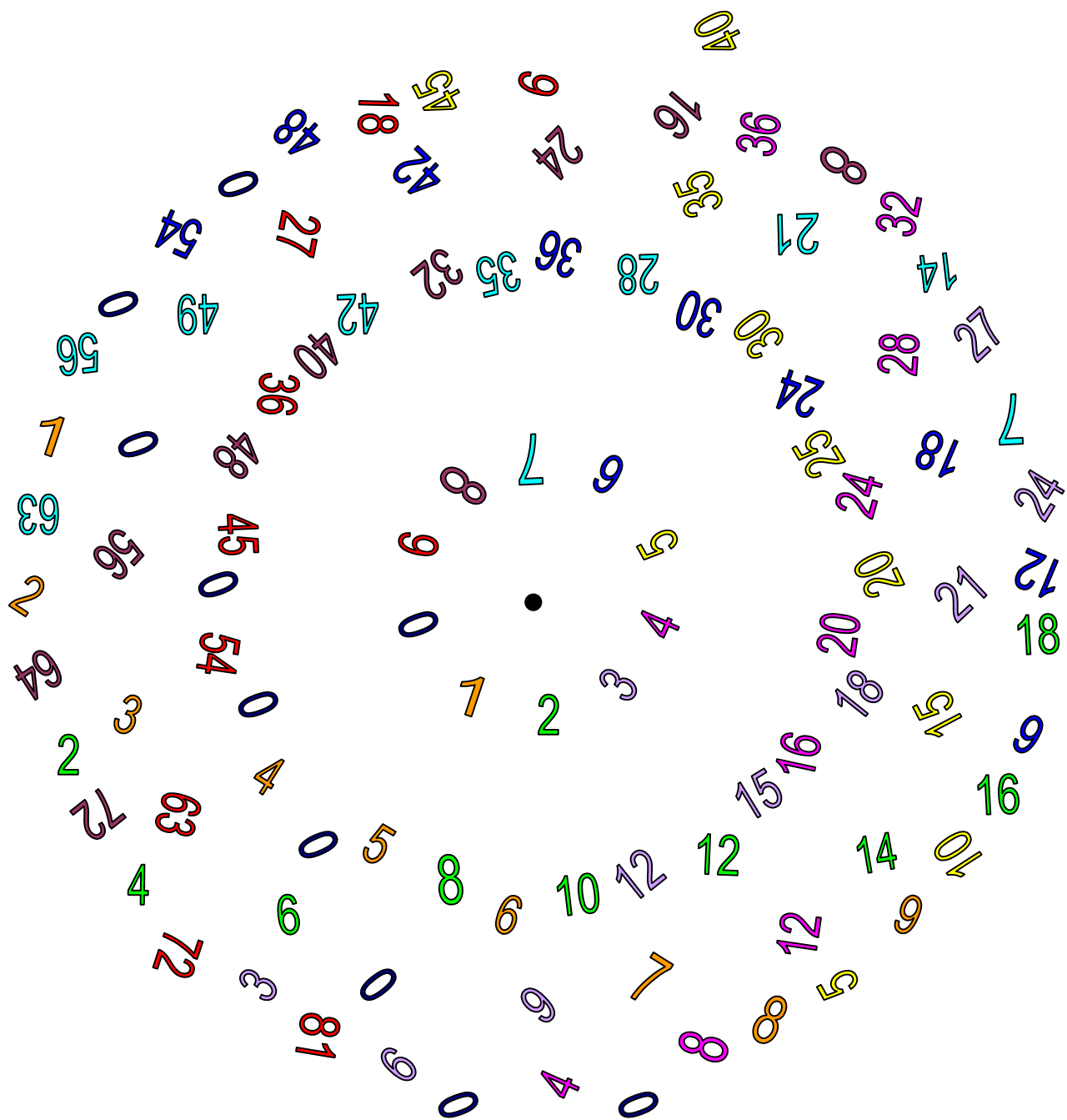


## MÁQUINA DE MULTIPLICAR DO BARNEY RUBBLE

Podemos efetuar o produto de dois números sobrepondo e rotacionando os discos abaixo





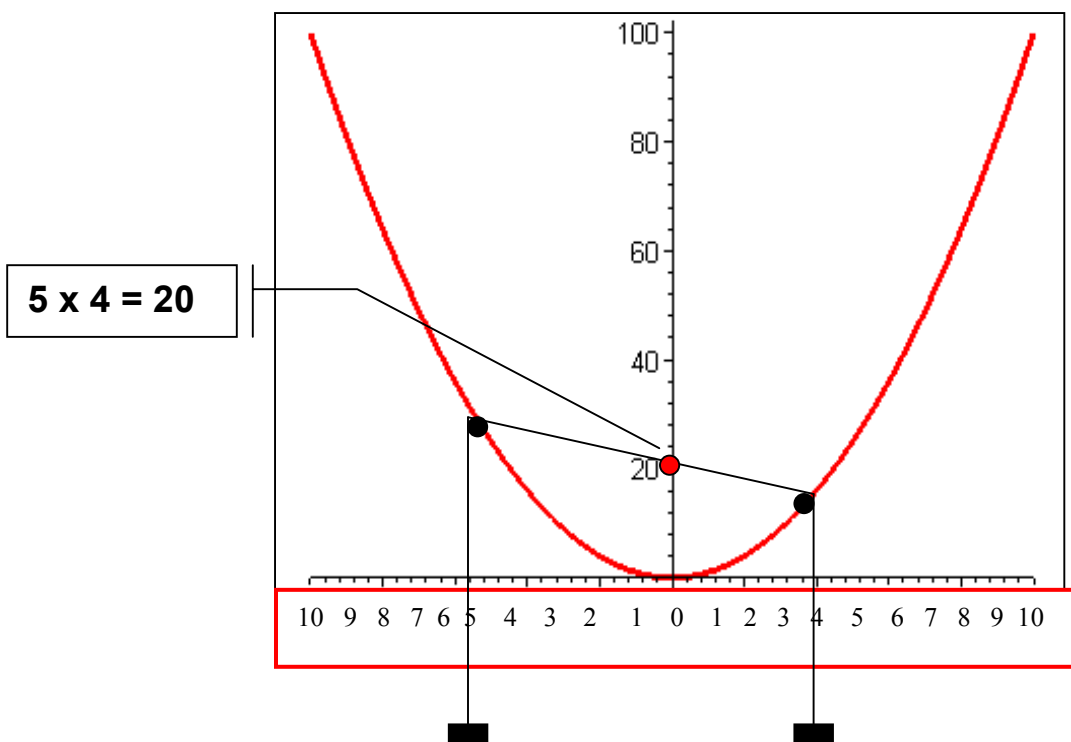






## A CALCULADORA PARABÓLICA

Na figura abaixo, encontra-se desenhada a parábola  $y = x^2$ . Usando-se dois alfinetes e uma linha com dois pesos presos nas extremidades é possível calcular o produto de dois números reais. Duvida?

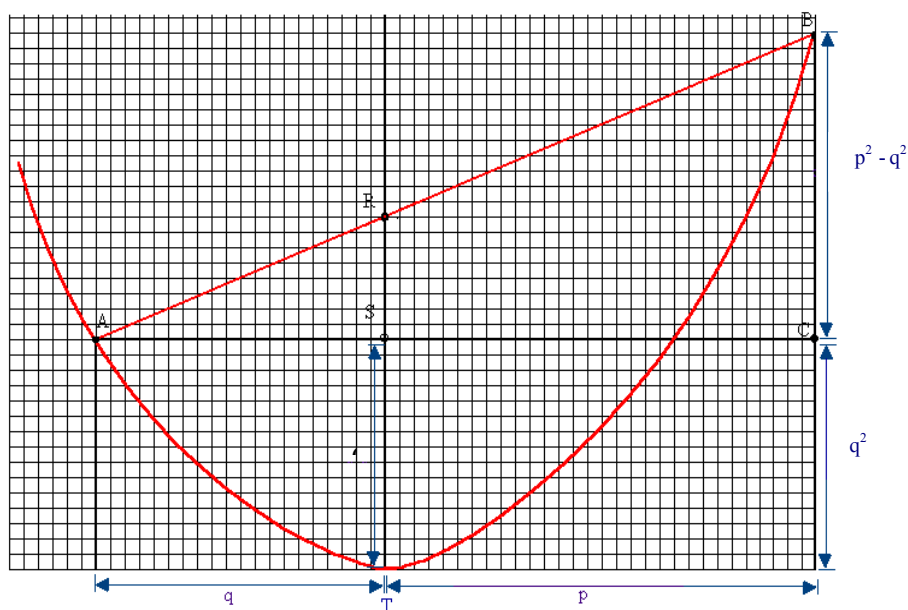


Observe no exemplo acima que  $5 \times 4 = 20$  é a leitura que se vê no eixo y. Os dois pequenos retângulos negros são pesos que fazem a linha ficar esticada. Sempre que realizarmos esta experiência, obteremos o produto (lido no eixo dos y's – no caso 20) de dois números dados no eixo dos x's, onde a linha encontra o eixo dos x (no caso 5 e 4). Vejamos o motivo de tal fato.

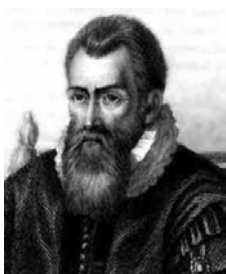
Para entender o funcionamento da calculadora parabólica, tomemos dois pontos p e q sobre o eixo horizontal (veja a próxima figura). Queremos obter o produto de p por q. Sobre o gráfico da parábola  $y = x^2$ , marcamos os pares ordenados A e B. Unindo os pontos A e B com uma reta, obteremos o ponto R sobre o eixo y, cujo valor da ordenada é precisamente p.q. Observe na figura os triângulos ARS e ABC. Eles têm os mesmos ângulos e portanto são semelhantes. Logo

$$\frac{RS}{BC} = \frac{AS}{AC} \quad \text{ou} \quad RS = \frac{AS \cdot BC}{AC} = \frac{q(p^2 - q^2)}{p + q} = q \cdot (p - q)$$

Assim  $RT = RS + ST = q.(p-q) + q^2 = p.q$ . Isto mostra que o nosso procedimento sempre produz o produto de dois números reais dados.



## PELAS BARRAS DE NAPIER



Uma versão simples dos “ossos de Napier” será apresentada abaixo. Trata-se de uma máquina que permite efetuar multiplicações de uma maneira muito mais simples que o algoritmo moderno que aprendemos na escola e portanto, com menos possibilidade de erros. Devemos reproduzir em uma cartolina a seguinte tabela:

	0	1	2	3	4	5	6	7	8	9	
1	$\frac{\quad}{\quad}$	$\frac{\quad}{1}$	$\frac{\quad}{2}$	$\frac{\quad}{3}$	$\frac{\quad}{4}$	$\frac{\quad}{5}$	$\frac{\quad}{6}$	$\frac{\quad}{7}$	$\frac{\quad}{8}$	$\frac{\quad}{9}$	1
2	$\frac{\quad}{\quad}$	$\frac{\quad}{2}$	$\frac{\quad}{4}$	$\frac{\quad}{6}$	$\frac{\quad}{8}$	$\frac{1}{0}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{6}$	$\frac{1}{8}$	2
3	$\frac{\quad}{\quad}$	$\frac{\quad}{3}$	$\frac{\quad}{6}$	$\frac{\quad}{9}$	$\frac{1}{2}$	$\frac{1}{5}$	$\frac{1}{8}$	$\frac{2}{1}$	$\frac{2}{4}$	$\frac{2}{7}$	3
4	$\frac{\quad}{\quad}$	$\frac{\quad}{4}$	$\frac{\quad}{8}$	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{2}{0}$	$\frac{2}{4}$	$\frac{2}{8}$	$\frac{3}{2}$	$\frac{3}{6}$	4
5	$\frac{\quad}{\quad}$	$\frac{\quad}{5}$	$\frac{1}{0}$	$\frac{1}{5}$	$\frac{2}{0}$	$\frac{2}{5}$	$\frac{3}{0}$	$\frac{3}{5}$	$\frac{4}{0}$	$\frac{4}{5}$	5
6	$\frac{\quad}{\quad}$	$\frac{\quad}{6}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{2}{4}$	$\frac{3}{0}$	$\frac{3}{6}$	$\frac{4}{2}$	$\frac{4}{8}$	$\frac{5}{4}$	6
7	$\frac{\quad}{\quad}$	$\frac{\quad}{7}$	$\frac{1}{4}$	$\frac{2}{1}$	$\frac{2}{8}$	$\frac{3}{5}$	$\frac{4}{2}$	$\frac{4}{9}$	$\frac{5}{6}$	$\frac{6}{3}$	7
8	$\frac{\quad}{\quad}$	$\frac{\quad}{8}$	$\frac{1}{6}$	$\frac{2}{4}$	$\frac{3}{2}$	$\frac{4}{0}$	$\frac{4}{8}$	$\frac{5}{6}$	$\frac{6}{4}$	$\frac{7}{2}$	8
9	$\frac{\quad}{\quad}$	$\frac{\quad}{9}$	$\frac{1}{8}$	$\frac{2}{7}$	$\frac{3}{6}$	$\frac{4}{5}$	$\frac{5}{4}$	$\frac{6}{3}$	$\frac{7}{2}$	$\frac{8}{1}$	9



A seguir, recortamos as colunas internas, deixando a moldura externa em forma de U intacta. Um exemplo: para multiplicar 679 por 381, as faixas numeradas com 6, 7 e 9 são seleccionadas e colocadas na moldura, como na figura a seguir:

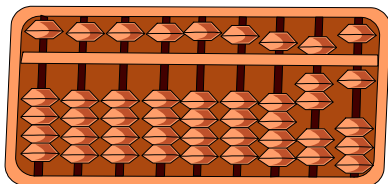
	<b>6</b>	<b>7</b>	<b>9</b>		<b>6</b>	<b>7</b>	<b>9</b>	
<b>1</b>	$\begin{array}{c} 6 \\ \hline 6 \end{array}$	$\begin{array}{c} 7 \\ \hline 7 \end{array}$	$\begin{array}{c} 9 \\ \hline 9 \end{array}$	(a)	$\begin{array}{c} 6 \\ \hline 6 \end{array}$	$\begin{array}{c} 7 \\ \hline 7 \end{array}$	$\begin{array}{c} 9 \\ \hline 9 \end{array}$	<b>x 1</b>
<b>2</b>	$\begin{array}{c} 12 \\ \hline 2 \end{array}$	$\begin{array}{c} 14 \\ \hline 4 \end{array}$	$\begin{array}{c} 18 \\ \hline 8 \end{array}$		6	7	9	
<b>3</b>	$\begin{array}{c} 18 \\ \hline 8 \end{array}$	$\begin{array}{c} 21 \\ \hline 1 \end{array}$	$\begin{array}{c} 27 \\ \hline 7 \end{array}$	(c)	$\begin{array}{c} 18 \\ \hline 8 \end{array}$	$\begin{array}{c} 21 \\ \hline 1 \end{array}$	$\begin{array}{c} 27 \\ \hline 7 \end{array}$	<b>x 300</b>
<b>4</b>	$\begin{array}{c} 24 \\ \hline 4 \end{array}$	$\begin{array}{c} 28 \\ \hline 8 \end{array}$	$\begin{array}{c} 36 \\ \hline 6 \end{array}$		2	0	3	7
<b>5</b>	$\begin{array}{c} 30 \\ \hline 0 \end{array}$	$\begin{array}{c} 35 \\ \hline 5 \end{array}$	$\begin{array}{c} 45 \\ \hline 5 \end{array}$					
<b>6</b>	$\begin{array}{c} 36 \\ \hline 6 \end{array}$	$\begin{array}{c} 42 \\ \hline 2 \end{array}$	$\begin{array}{c} 54 \\ \hline 4 \end{array}$					
<b>7</b>	$\begin{array}{c} 42 \\ \hline 2 \end{array}$	$\begin{array}{c} 49 \\ \hline 9 \end{array}$	$\begin{array}{c} 63 \\ \hline 3 \end{array}$					
<b>8</b>	$\begin{array}{c} 48 \\ \hline 8 \end{array}$	$\begin{array}{c} 56 \\ \hline 6 \end{array}$	$\begin{array}{c} 72 \\ \hline 2 \end{array}$	(b)	$\begin{array}{c} 48 \\ \hline 8 \end{array}$	$\begin{array}{c} 56 \\ \hline 6 \end{array}$	$\begin{array}{c} 72 \\ \hline 2 \end{array}$	<b>x 80</b>
<b>9</b>	$\begin{array}{c} 54 \\ \hline 4 \end{array}$	$\begin{array}{c} 63 \\ \hline 3 \end{array}$	$\begin{array}{c} 81 \\ \hline 1 \end{array}$		5	4	3	2

Como  $381 = 300 + 80 + 1$ , a primeira tarefa que faremos é multiplicar 679 por 1, o que é feito na parte (a). Para multiplicar 80 por 679, partimos para o estágio (b), observando apenas a linha correspondente na moldura ao número 8. No estágio b, efetuamos a soma diagonal dos números, começando com 2, o qual é registrado; a seguir somamos 7 e 6, colocado 3 e transportando 1 para a outra diagonal à esquerda. A próxima diagonal contém 5 e 8 que totalizam 13 que com o 1 da conta anterior perfaz 14. Registramos 4 e transportamos 1 para a outra diagonal mais à esquerda. A última operação é somar 4 com o 1 que veio da operação anterior, obtendo 5. O número obtido após o estágio (b) é 5432, que deve ser acrescido de 0 pois trata-se de multiplicação na classe das dezenas.

O último estágio, (c), é feito na terceira linha e trata-se da multiplicação na classe das centenas. O processo de soma diagonal é o mesmo: inicialmente registramos o número 7, a seguir  $2 + 1 = 3$  na segunda diagonal, na terceira  $8 + 2 = 10$ , registrando 0 e transportando 1 e finalmente 1 acrescido do 1 que veio do transporte, o que resulta 2. Acrescentamos 00, pois estamos na classe das centenas. Obteremos deste modo o subtotal 203700.

Adicionando os três subtotais, chegamos ao resultado: 258.699.

## A ORIGEM DOS COMPUTADORES



A verdadeira origem dos algoritmos e conseqüentemente dos computadores remonta o próprio surgimento do pensamento lógico. A História nos mostra que o desenvolvimento das culturas está ligado diretamente ao manuseio de técnicas de cálculo, principalmente as originárias do oriente (ábaco, algarismos indús, etc.).

Entretanto, nosso objetivo aqui é o de procurar a origem dos computadores vinculada ao desenvolvimento de técnicas que permitiram substituir o homem pela máquina. A invenção de máquinas que realizam tais tarefas está, por sua vez, ligada aos modos de produção da economia. Infelizmente a maioria das máquinas foi inventada para a dominação belicista e exploração do homem pelo homem. A primeira grande revolução do mundo moderno, conhecida como Revolução Industrial, exigiu o domínio da energia mecânica, principalmente a produzida pelo vapor. Isto, juntamente com a divisão do trabalho e a especialização, alavancaram a invenção de aparatos mecânicos que substituíram o trabalho humano pelo o de máquinas. Por volta 1600, com a expansão do mercantilismo, novas exigências técnicas e científicas brotaram, principalmente no setor de transporte e balística. O momento histórico propiciou o surgimento da teoria newtoniana na Física, integrando-a com a Matemática. Como conseqüência ocorreu uma racionalização do conhecimento, colocando-se regras rigorosas para o pensar (Descartes e Leibniz ).

Com o domínio cada vez mais intenso das técnicas de transformação da energia, cresceu assim a possibilidade da formalização e mesmo da mecanização das atividades intelectuais. Esta nova vertente, esta nova visão de mundo, foram expressas no trabalho dos principais filósofos da época que passamos muito brevemente a descrever:



Hobbes (1588-1679): combinação de unidades simples como método científico.

← Descartes (1596-1650): buscava a essência lógica através do método do raciocínio, inventor da Geometria Analítica.

Locke (1623-1704): introduziu métodos que permitiram a organização das idéias, as mais simples como construtoras das mais complexas.

Leibniz (1646-1716): matematização do conhecimento e formulação de uma linguagem simbólica para a Lógica. Inventor, junto com Newton do Cálculo Diferencial e integral.

Newton (1642-1727): fundamentou as bases da Ciência Moderna, introduzindo o Cálculo Diferencial e Integral para o estudo de fenômenos físicos.

Laplace (1724-1804): descrição da natureza levando em conta o determinismo.

Kant (1724-1804): alicerces do formalismo lógico, colocando a lógica como método científico. Criador de teorias formais, não interpretadas.

Boole (1815-1864): algebrização das regras do pensamento.

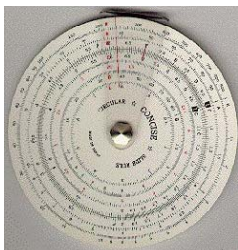
No final do século XIX, além da corrente clássica, surgiram novas linhas investigativas dentro da Lógica Matemática, dentre as quais destacam-se os trabalhos de Frege (1848-1925) (Fundamentos da Aritmética), de Bertrand Russel (1872-1970) (corrente logicista), de Brouwer (corrente construtivista) e de Hilbert (1862-1943) (corrente formalista). Russel conseguiu formular contradições dentro da teoria ingênua dos conjuntos e o trabalho construtivista de Brouwer abriu as portas para novas formulações da Lógica, admitindo-se a possibilidade de estudo de afirmações que não são nem verdadeiras, nem falsas.

No início do século XX, modificações radicais no pensamento foram introduzidas, principalmente relativos à universalização dos conceitos com experimentos e teorias não justificadas pela mecânica newtoniana. Houve a introdução de técnicas probabilísticas na descrição dos eventos e estudos de situações em que as interações entre o observador e o observável não podiam mais ser desprezadas.

Do ponto de vista técnico, as “máquinas pensantes” acompanharam paralelamente estas turbulências de idéias. Para entender melhor sua evolução, apresentaremos a seguir algumas invenções e datas significativas do ponto de vista histórico.

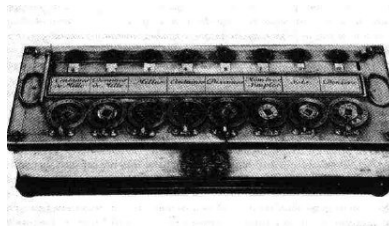
**Ábaco** – talvez o mais antigo dos computadores.

**Réguas de Cálculo** - um aparato mecânico, composto por duas régua, uma delas deslizante, marcadas com várias escalas numéricas, que permitem somar, subtrair, multiplicar, achar raízes e calcular logaritmos entre outras coisas.

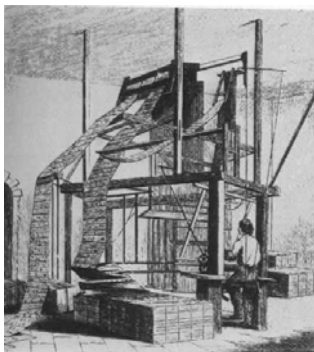
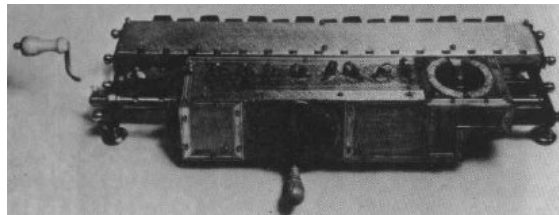


**1621 William Oughtred** - matemático inglês que inventou a primeira régua de cálculo circular. Ela é considerada o primeiro computador analógico.

**1642 Blaise Pascal** – construiu a primeira calculadora automática, feita com engrenagens. Ele construiu 50 para vender, mas os caixeiros e contadores recusaram-se a usá-las, com medo de perderem seus empregos!



**1673 Gottfried Leibniz** – inventou um novo tipo de calculadora, baseada num cilindro com dentes. Ela efetuava as quatro operações elementares.



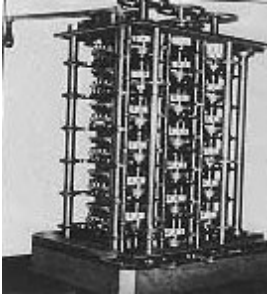
**1804 Joseph Jacquard** – um francês que inventou um sistema de cartões perfurados usados em tecelagem. Os cartões deixavam passar alguns elementos e bloqueava outros. O tear de Jacquard não se relaciona diretamente com computadores, mas a idéia de usar cartões perfurados foi adaptada mais tarde por Babbage como um método mecânico de colocar informações dentro do computador.

**1822 Charles Babbage** – completou a primeira máquina de diferenças com o apoio do governo britânico. Ela podia resolver algumas equações polinomiais. Era uma máquina muito sensível que quebrava freqüentemente, a ponto do primeiro ministro inglês da época declarar que o único propósito de sua construção era calcular a quantidade de dinheiro gasta para construí-la. Mais tarde, Babbage arquitetou um engenho analítico que deveria ser utilizado para realizar cálculos muito mais precisos, baseado em uma tecnologia inovadora para a época, que incluía 5 elementos cruciais presentes nos futuros computadores:

- um aparato de entrada (input)
- facilidades de estocagem para armazenar números durante o processamento



- um processador ou calculador numérico
- uma unidade de controle para tarefas diretas a serem realizadas
- um aparato de saída (output)



**1883 Augusta Ada** – uma matemática amadora e amiga íntima de Babbage que achava que o engenho analítico podia ser programado usando um conjunto simples de cartões para operações repetitivas. Foi a primeira vez que a programação de computadores foi sugerida. Ela é considerada a primeira programadora.

**1886 Herman Hollerith** - inventou uma máquina de tabulação que usava cartões perfurados para contar eletronicamente. Os cartões passavam entre cilindros de latão e, onde os cartões estavam perfurados ocorria contato e passagem de corrente elétrica, completando o circuito. Este aparelho foi construído para ser usado em censos populacionais. Tabulações manuais levariam mais de uma década. Em 1896 Hollerith fundou a Tabulating Machine Company e em 1924, após várias fusões, surgiu a International Business Machines (IBM).

**1936 Alan Turing** – introduziu um computador teórico digital hipotético, chamado de máquina de Turing. O desenvolvimento de sistemas computacionais foi impulsionado pela II Guerra Mundial, principalmente para decifrar códigos secretos e calcular trajetórias de mísseis. Turing trabalhou nestas tarefas para o governo inglês. Estudaremos com mais detalhes nos próximos capítulos alguns de seus trabalhos.

**1939 ABC** – o primeiro computador digital, inventado por John Astanasoff.

**1944 Mark I** – o primeiro computador para uso geral controlado por programas. Foi utilizado na Universidade de Harvard por 15 anos.



**1945 Primeiro 'Bug'** – durante o desenvolvimento do computador Mark II um relê dentro do computador falhou e os pesquisadores demoraram um mês para

descobrir um inseto morto dentro de seus contatos. Por isso o termo 'bug' é usado como sinônimo de algum problema dentro do computador.

**1945 John Von Neuman** – desenvolveu o conceito de programa armazenado. Sua idéia era estocar não somente os dados a serem processados na memória do computador, mas também as instruções usadas para processar os dados. Esta idéia é considerada por muitos como a mais importante de toda a Ciência.

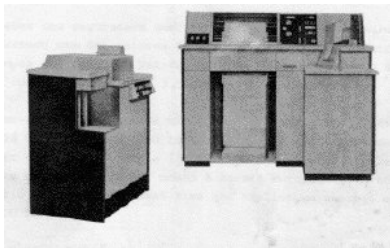


**1946 ENIAC** - era um computador enorme, com 18.000 válvulas. Seus tubos de vácuo produziam muito calor. Este sistema agora poderia aquecer uma grande cidade.

### **A primeira geração de Máquinas:**

A indústria dos computadores começou seu desenvolvimento a partir do ENIAC (Eletronic Numeric Integrator and Computer). Esta máquina foi arquitetada por Jonh Mauchly e J. Presper Eckert Jr. Na Universidade da Pensilvânia de 1942 a 1945. A máquina pesava 30 toneladas e ocupava um prédio inteiro. Em média, uma válvula queimava a cada 15 minutos. A programação exigia o acionamento de chaves e eram necessários 200 microsegundos para efetuar uma adição e 3 milissegundos para multiplicar por três. O ENIAC foi retirado de serviço em 1955.

A era da informática começou oficialmente em 14 de junho de 1951 quando o primeiro UNIVAC (Universal Automatic Computer) foi usado pelo órgão de recenseamento americano. Esta máquina, desenvolvida pela Eckert Mauchly Computer Corporation, foi o primeiro computador comercialmente disponível. A fábrica construtora foi fundada em 1947 e vendida para a Remington-Rand logo em seguida.



Em 1952 o UNIVAC previu a vitória para presidente dos Estados Unidos de Eisenhower, analisando apenas cinco por cento dos votos.

Um dos maiores computadores já construídos foi o UNIVAC 1105, manufaturado pela Remington-Rand para a Universidade da Carolina do Norte em 1959. Custou 2.450.000 dólares, continha 7.200 válvulas e pesava 35 toneladas. Tinha memória de 540.000 bytes, com aproximadamente 144.000 bytes para estocagem. As operações elementares eram realizadas em aproximadamente 44 microsegundos.

A IBM, por outro lado, já trabalhava com cartões perfurados para processar grande volume de dados, cerca de 40 anos antes da construção do ENIAC. A companhia hesitou em entrar no ramo dos computadores até a invenção do UNIVAC, que fez com que seus negócios comesçassem a diminuir. O primeiro computador produzido pela IBM foi o IBM 701, sob encomenda do governo norte-americano em 1953 e posteriormente substituído pelo IBM 650. A companhia liderou o mercado de computadores em 1956 com 76 computadores vendidos, ultrapassando a Remington-Rand.

### **A Segunda geração de computadores (1958-1964)**

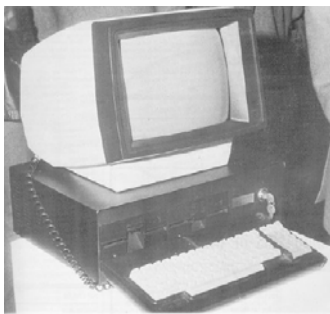


Com esses computadores iniciou-se o período da transferência tecnológica. As máquinas mais populares foram o IBM 7090, 7070 e 1410. Outros fabricantes surgiram, como, por exemplo, a Sperry-Rand, RCA, General Electric, Burroughs, Honeywell, NCR e Control Data. A característica mais comum destas máquinas foi a utilização de transistores ao invés de válvulas, reduzindo assim o tamanho e possibilitando a introdução de linguagens de alto nível e discos removíveis (1962).

### **A terceira geração de computadores (1964-1970)**

Foram os primeiros micro-computadores. O modelo mais conhecido foi o IBM 360. Já eram construídos com circuitos integrados. O uso de semicondutores de silício melhorou a performance e o poder computacional e reduziu tamanho e custo. Eram empregados em técnicas de processamento em grupo.

## A Quarta geração de computadores (1970 - ----)



A Quarta geração de computadores marca o início das máquinas baseadas em microprocessadores. Em 1969, Ted Hoff começou a trabalhar com a idéia de substituir todos os circuitos de um computador por um simples *chip*. O desenvolvimento desta idéia resultou no microcomputador atual. A Intel relutou em encampar o trabalho de Hoff pois os potenciais compradores não concebiam aplicações de tais aparatos. Hoje eles podem ser encontrados em todos os lugares, em carros, em escolas, em supermercados, etc.

Os primeiros microcomputadores reais foram: Altair, EMCII, Tandy TRS-80, Atari e Commodore. Em 1976, Steve Jobs e Steve Wozniak mudaram para sempre a indústria dos computadores quando venderam um Volkswagen e uma calculadora por 1.300 dólares e, com dinheiro, construíram o primeiro computador Apple. O trabalho foi feito em uma garagem. Foi o primeiro sistema importante com teclado e tela. Em 1983, a Apple já era uma grande empresa. O IBM PC foi anunciado em 1981, atingindo o topo de vendas por 18 meses e tornando-se um modelo para as outras máquinas. Hoje o IBM PC e o Macintosh da Apple estão conectados com estações de trabalho e outros poderosos computadores, de modo que as pesquisas e informações podem ser compartilhadas.

E os próximos computadores como serão? Computadores quânticos? Bio-computadores?

## COMPUTADORES ANALÓGICOS:

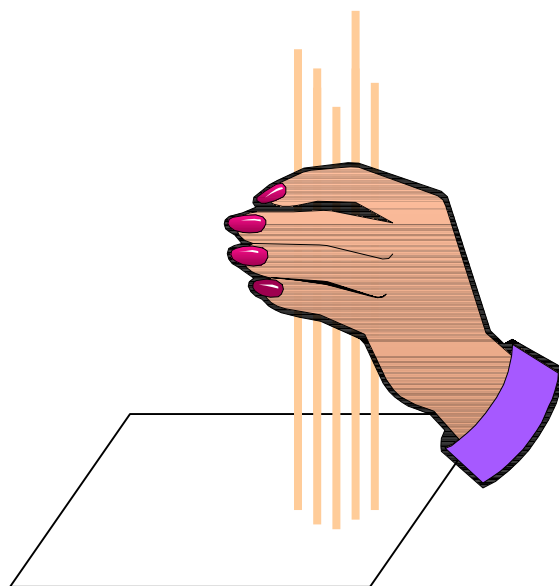
Há aproximadamente 50 anos, os computadores eram divididos em duas categorias: os analógicos e os digitais. Hoje não se fala mais em computadores analógicos, mas por muitos anos eles foram utilizados para a medição e cálculos servindo de instrumento para muitas áreas do conhecimento (medição do tempo, das marés, resolução de equações diferenciais, etc.). Os computadores analógicos permitiram que o homem pudesse explorar o espaço sideral, nos primórdios da Era Espacial.

Um modelo muito simples de um computador analógico é um relógio com ponteiros. No início da era dos computadores a velocidade de processamento dos computadores analógicos era muito superior aos dos computadores digitais. Esses computadores, como o próprio nome diz, funcionam por analogia, por comparação, adotando procedimentos contínuos. Já os computadores digitais (que são os disponíveis atualmente) funcionam com quantidades discretas que variam passo a passo (tais como os relógios com mostradores digitais).

Os computadores digitais mostram-se muito mais rápidos e precisos que os analógicos, embora alguns procedimentos sejam melhor realizados com computadores analógicos. Vamos exibir dois exemplos deste fato: um que usa varetas (parecidas com macarrão espaguete) para ordenar números naturais e outro que calcula instantaneamente a distância mais curta entre duas cidades com várias rotas de ligação (para isto usamos somente barbante).

### **Como ordenar números usando macarrão:**

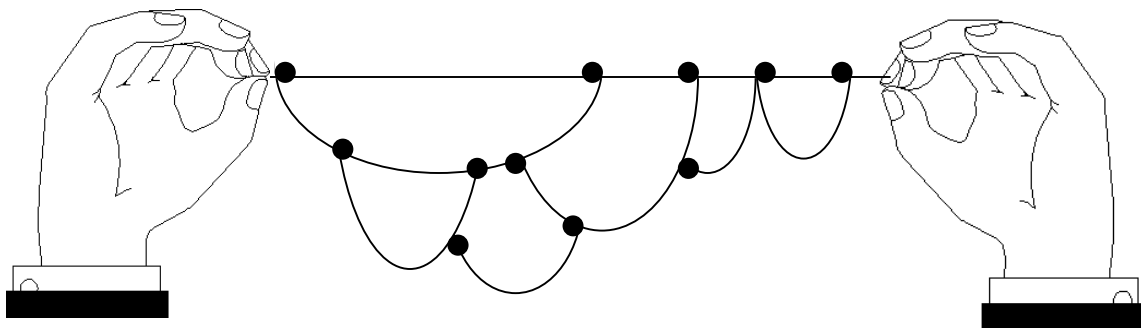
Para colocar em ordem decrescente uma dada seqüência de números naturais que estejam desordenados, tais como 7, 12, 4, 18, 11, 10, 21, 9, 13, podemos construir um algoritmo que compare cada um dos pares dentre os números dados, ou também construir um “computador analógico” feito macarrão do tipo espaguete. Basta cortar cada vareta de macarrão para que ele tenha o mesmo tamanho de um dos números que queremos ordenar (em centímetros). Agora, juntando todos as varetas em um feixe e apoiando-o sobre uma mesa, podemos escolher a vareta mais comprida para ser o maior dos números, depois a segunda mais comprida como o número imediatamente inferior ao maior e assim sucessivamente até chegarmos à vareta de menor tamanho. Com uma única operação enumeramos automaticamente todos os inteiros da lista.



### **Como economizar gasolina usando barbante:**

Ao planejar uma viagem entre duas cidades ligadas por muitas rodovias, podemos estar interessados em encontrar a rota de menor distância entre elas. Com um mapa à disposição, podemos encontrar todos os caminhos possíveis (passando por cidades intermediárias inclusive) e desta lista escolher aquela que

tem a menor distância. Se o número de opções for grande, poderemos ter que analisar um número enorme de possibilidades. Entretanto, se utilizarmos um fio de barbante atado com nós, podemos resolver facilmente este problema. Cada nó representa uma cidade e a distância entre dois nós representa uma estrada ligando-as. O comprimento do barbante entre dois nós representa (em centímetros) a distância da rodovia que une as duas cidades. Segurando os nós de origem e de destino e esticando a malha formada pelo barbante, obteremos imediatamente o caminho mais curto entre as duas cidades.





## 5. CONVERSANDO COM MÁQUINAS

*Como entender a linguagem de uma máquina. A caixa preta de um disco voador*

**Resumo:** Será apresentado e posteriormente construído um aparato eletrônico chamado de “caixa preta de um disco voador”. Ele é formado por uma pequena caixa com botões e uma lâmpada que “entende” uma linguagem simples previamente embutida no *hardware* da máquina. Esta linguagem só é entendida por marcianos e é tarefa dos humanos decifrar e compreender como são formadas as palavras em marcianês. O aparato é um autômato finito e linguagem entendida por ele motiva o uso de notação científica e de cálculos algébricos com potências. Este protótipo mostra o caminho para se construir outros aparatos que “conversam” com humanos.

### A CAIXA PRETA DE UM DISCO VOADOR

Uma nave extraterrestre sofreu uma avaria muito grave quando estava na órbita da Terra e caiu no solo provocando um grande estrondo e clarão. Não houve sobreviventes, todos os ET's a bordo morreram.

Dentre os destroços foi encontrada a caixa preta da astronave, que deve estar agora em suas mãos.

Uma conjectura feita pelos cientistas que analisaram a caixa misteriosa afirmava que ela continha todas as informações a respeito da linguagem falada pelos marcianos (com certeza eles vinham de Marte, pois eram verdinhos e tinham antenas).

A caixa consiste de três botões: um vermelho, um amarelo e um preto. Logo no início da inspeção os cientistas descobriram que o botão preto servia como uma chave de liga/desliga do aparato espacial. Os botões vermelho e amarelo deveriam significar duas letras do alfabeto marciano (em Marte os dicionários só têm duas seções: a seção das palavras que começam, digamos com **a** e a das palavras que começam com **b**).

Evidentemente nem todas as combinações de letras formam palavras. Mesmo em português, existem palavras legítimas da língua como rato e chapéu e combinações sem sentido como gapéu, fhc, acm, etc.

Como então fazer para descobrir, analisando a caixa preta, quais são as palavras da linguagem falada em Marte? Não devemos abrir a caixa ou quebrar o aparelho, pois a tecnologia marciana para a construção de equipamentos está muito desenvolvida para o nosso atual nível de compreensão.

Como desvendar os mistérios da língua marciana? Vejamos como isto pode ser feito:

Primeiro ligue o botão preto (RESET). Isto equivale a ligar o computador. Uma das seguintes coisas deve acontecer:

1a.) A lâmpada acende imediatamente (os marcianos esqueceram de desligar o monitor de vídeo da última vez que usaram o aparelho).

ou

2a.) A lâmpada não acende. Neste caso será necessário ligar o monitor, i. e., a lâmpada. Para isto basta acionar o botão vermelho.

Ótimo! Nosso computador marciano está pronto para o uso. Agora as teclas **a** e **b** podem ser usadas para saber se uma palavra faz sentido em marcianês ou não.

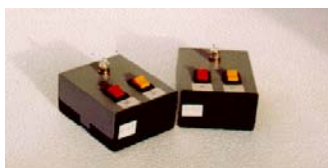
Exemplo1: digite **a**, depois **b**, depois **a**.

Feito isto observe que a lâmpada ficou acesa, mostrando que **aba** é uma legítima palavra da linguagem marciana.

Exemplo 2: digite **bab**.

A lâmpada apagou-se: esta palavra não pertence à linguagem de Marte.

Analisando a caixa preta você poderia dizer quais são todas as palavras que se encontram no dicionário do planeta Marte?

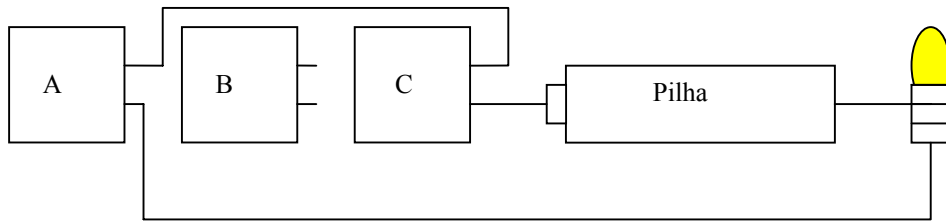


Escreva, usando a notação científica, todas as palavras da linguagem marciana.



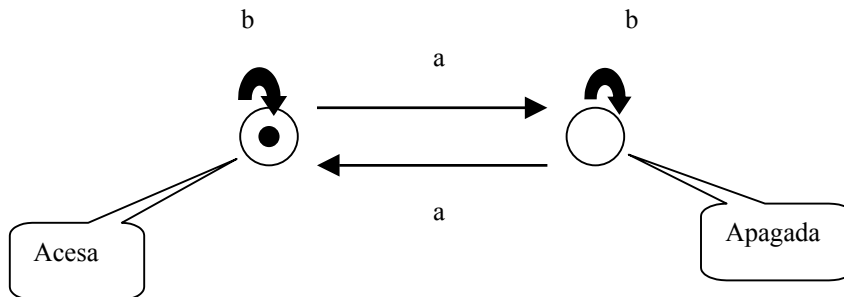
## ESQUEMAS DA CAIXA PRETA

### Circuito Elétrico:



A e B são teclas de digitação e C é a tecla “Reset”.

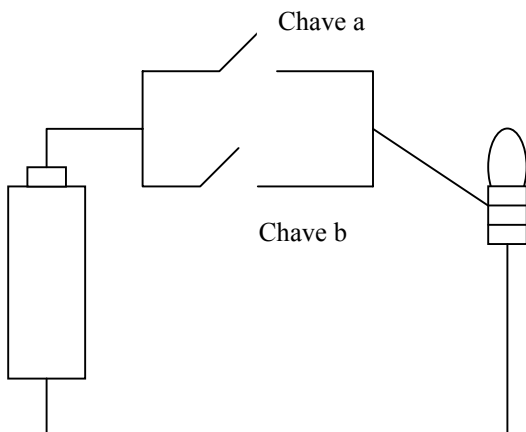
### Representação na forma de um grafo:



## OUTRAS MÁQUINAS E OUTRAS LINGUAGENS

### Máquina OU

Podemos, usando outros circuitos elétricos simples, construir outras máquinas que entendem linguagens mais sofisticadas do que a dos marcianos, apresentada acima. Por exemplo, considere o circuito ou:



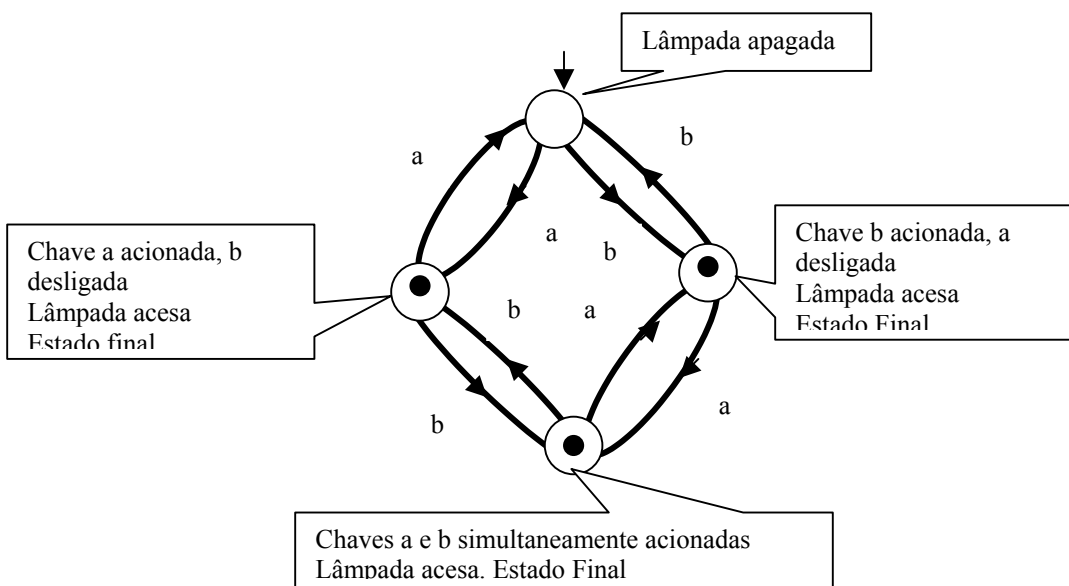
Manuseando os interruptores, podemos colocar as chaves a e b ambas na posição aberta, como na figura acima. Deste modo, inicialmente podemos supor que a lâmpada está apagada. Ao digitarmos a tecla a ou a tecla b ou ambas (a seguida de b ou b seguida de a), a lâmpada se acenderá.

Qual será a linguagem que esta máquina entende?. Para isto vamos fazer uma tabela, colocando de um lado as palavras aceitas pela máquina e por outro as palavras que ela não entende.

Aceitas	Não aceitas
a	aa
b	bb
aab	abab

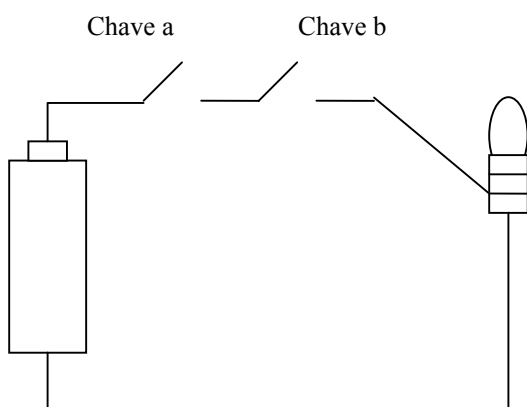
Complete a tabela e descubra quais são as palavras entendidas pela máquina. Lembre-se que antes de digitar uma seqüência de letras as chaves devem estar ambas desligadas (lâmpada apagada).

Observe o esquema de funcionamento da máquina:



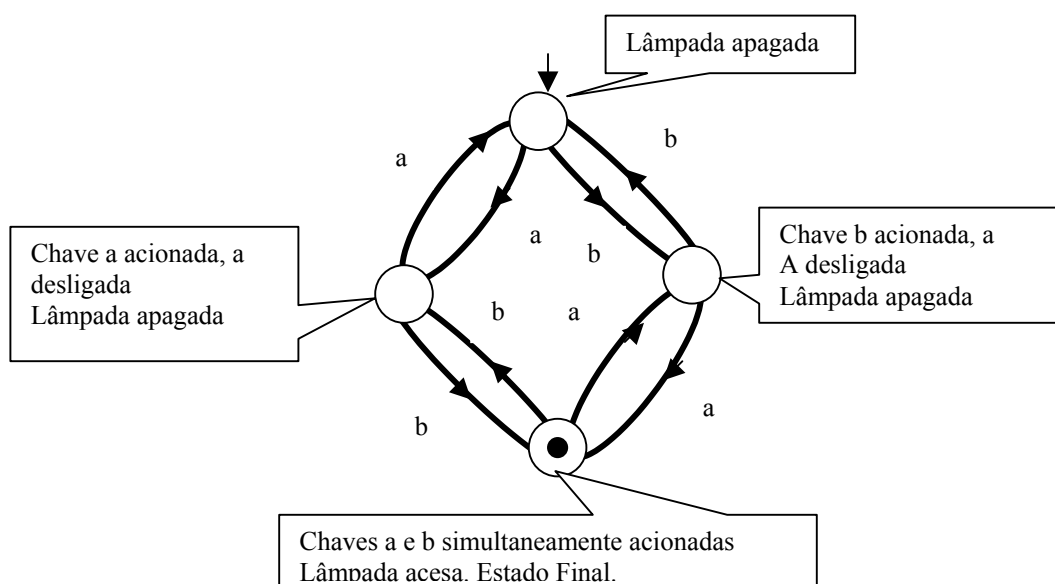
## Máquina E

Inicialmente as chaves devem ser colocadas, ambas abertas, de acordo com o esquema abaixo:



A lâmpada só acenderá se as chaves forem fechadas simultaneamente. Qual será a linguagem entendida por esta máquina? Complete a tabela e saberá:

Palavras aceitas	Palavras não aceitas
ab	a
ba	b
aaab	abab







## 6. AUTÔMATOS

*Modelos inteligentes fora do tempo e do espaço*

**Resumo:** Inicia-se aqui as correlações existentes entre linguagens formais e máquinas, na forma de um texto com exemplos simples, com o objetivo de se mostrar uma hierarquia das máquinas e das linguagens associadas a elas. O que se procura é a investigação das limitações das máquinas do tipo autômato finito (caixas pretas) quanto à sua compreensão de linguagens e a proximidade da linguagem entendida por máquinas com a linguagem natural, que evidentemente ocupa o topo da hierarquia das linguagens.

### LINGUAGENS

Nesta seção continuaremos nossos estudos sobre a comunicação entre homens e máquinas e entre máquinas e máquinas. Para isto, precisamos iniciar um breve estudo sobre linguagens. Os programas de computadores e as soluções de problemas matemáticos são na verdade sentenças construídas com um número finito de símbolos. Se quisermos compreender quais problemas podemos resolver, qual é a complexidade de tais problemas, devemos ter claro qual é a natureza e quais são as limitações destas linguagens, antes mesmo de buscarmos as desejadas soluções.

Esta é uma matéria de importância tanto em Ciências Exatas como em muitos ramos das Ciências Humanas, tais como a Psicologia e a Lingüística.

Quando estudamos fenômenos do mundo real através de modelos matemáticos, é muito comum que nos deparemos com tarefas complexas; por esse motivo convém selecionarmos quais os fatos essenciais para o nosso estudo e desprezarmos outras acontecimentos não tão significativos (por exemplo em

Física muitas vezes desprezamos o atrito, a variação da aceleração da gravidade, etc.).

Analogamente, quando estudamos linguagens, devemos ignorar, pelo menos inicialmente, assuntos como pronúncia, estilo, gramática e ficar próximos da “essência de toda linguagem”. Que essência é esta? Trataremos aqui quase que exclusivamente da essência sintática das linguagens; as questões semânticas são mais complicadas e merecem ser estudadas com muito mais profundidade.

Para nós uma linguagem será uma coleção de seqüências de símbolos de um dado alfabeto.

Def.: Um alfabeto  $A$  é um conjunto finito e não vazio.

Ex:  $\{\$, \{a,b,c,\dots,z\}, \{0,1\}, \{*, \#\}$  são alfabetos.

Def: Uma palavra de comprimento  $n$  sobre um alfabeto  $A$  é uma seqüência de  $n$  símbolos do alfabeto  $A$ .

Notações:  $p = a_1 \dots a_n$ ,  $a_i \in A$ . Se  $n = 0$ , a palavra vazia (silêncio), será denotada pelo símbolo  $\phi$ .

Ex:  $\$, \$ \$$ ,  $a b c$ ,  $0 0 1$  e  $* * \#$  são palavras de comprimento 3 nos alfabetos do exemplo anterior.

Def: O conjunto de todas as palavras sobre um alfabeto  $A$ , incluindo a palavra vazia  $\phi$ , é denotado por  $A^*$ .

Teorema: Se  $A$  é um alfabeto, então  $A^*$  é enumerável, isto é, podemos efetivamente listar todas as palavras de  $A^*$ .

De fato, as palavras podem ser contadas usando-se a ordem lexicográfica (do dicionário). Se  $A = \{a_1, \dots, a_n\}$ , podemos enumerar  $A^*$  do seguinte modo:  $\phi$ ,  $a_1$ ,  $a_2$ , ...,  $a_n$ ,  $a_1 a_1$ ,  $a_1 a_2$  e assim sucessivamente.

Def: Uma linguagem sobre o alfabeto  $A$  é um subconjunto  $L \subset A^*$ , isto é, um elemento do conjunto das partes de  $A$ .

Teorema: Existe uma quantidade não enumerável de linguagens.

A demonstração segue de um teorema clássico devido a Cantor (matemático que dedicou seu trabalho ao estudo da Teoria dos Conjuntos).

## DESCRIÇÃO DE LINGUAGENS

Existem muitas maneiras de se descrever uma linguagem, mas todas elas podem ser transcritas numa seqüência de sentenças feitas com símbolos da pessoa que a está descrevendo. Tal seqüência de sentenças pode ser

interpretada como uma palavra (talvez muito longa) sobre o alfabeto  $T$  do escriba,  $i$  é, como um elemento de  $T^*$ . Como  $T^*$  é enumerável, só existe uma quantidade enumerável possível de descrições de linguagens. Como existe uma quantidade não enumerável de linguagens, chegamos à conclusão de que nem todas as linguagens podem ser descritas.

Assim, como só podemos descrever algumas linguagens, devemos ter esperança que possamos descrever linguagens que são úteis. Este é um tópico fundamental em Ciência da Computação e em Linguística.

Como então compreender linguagens?

Por exemplo, como decidir se a frase “I belt much” é uma frase válida em inglês? Podemos proceder do seguinte modo:

- As letras do alfabeto empregado existem na língua inglesa
- As palavras listadas pertencem à linguagem inglesa (basta procurar num dicionário)
- As regras de formação e de concordância são aceitas na língua inglesa.

Isto tudo pode ser feito de uma vez só, se pudermos consultar uma pessoa fluente em Inglês, que nos diga se a expressão é aceitável ou não.

Passemos então ao estudo de linguagens que podem ser entendidas ou mesmo geradas por máquinas. Será que podemos construir um computador de modo que este aceite uma sentença se e somente se ela está em uma dada linguagem  $L$  e reciprocamente, dado um computador (funcionando!), podemos definir uma linguagem (a da máquina), constituída pelas palavras que são aceitas por ele?

Buscando somente a essência de tais procedimentos, nosso computador deve, no mínimo, “ler” as palavras de um alfabeto  $A$  e “aceitá-las” ou não, de acordo com certas regras que definem a linguagem.



Vamos imaginar como isto pode ser feito: Sente-se em frente a um computador. Existe um teclado repleto de símbolos e a coleção dos símbolos do teclado será nosso alfabeto  $A$ .

Ligando o computador observamos que ele adquire uma certa configuração (dados na memória, tela do monitor de vídeo, etc.). Este é o estado inicial da máquina que denotaremos por  $q_0$ . Os estados de uma máquina variam com o tempo e com os dados. Vamos assumir que a máquina possua um número finito  $E$  de estados. Quando a máquina é ligada e está preparada para receber o primeiro dado (primeiro *input*: pode ser o apertar de uma tecla, por exemplo), dizemos que a máquina está partindo de seu estado inicial.

Se datilogarmos um símbolo de  $A$ , podemos mudar o estado (modificando os dados na memória ou na tela), dependendo do que foi digitado e do estado anterior a esta tarefa.

Esta mudança ocorre de uma maneira determinística, isto é, para todo estado  $q \in E$  e símbolo  $p \in A$ , o par  $(q,p)$  determina univocamente o novo estado, o qual será denotado por  $\delta(q,p)$ . Se  $(q,p)$  determinar estados diferentes em diferentes tempos, é sinal que a máquina está com defeito ou algum mecanismo não determinístico deve estar agindo.

Para que o computador aceite ou reconheça linguagens, vamos selecionar um subconjunto  $F$  dos estados  $E$  tal que quando a máquina se encontrar no estado final  $F$ , podemos afirmar que ela aceitou o dado (a palavra).

Neste contexto, sem necessitarmos de eletricidade, discos e manuais, podemos definir um computador puramente em termos da teoria dos conjuntos:

**Def:** Um autômato finito  $M$  é um conjunto de 5 objetos:  $E, A, \delta, q_0$  e  $F$ , onde

- $E$  é o conjunto não vazio de estados
- $A$  é o alfabeto (também um conjunto não vazio)
- $\delta$  é uma função de  $E \times A$  em  $E$ , chamada função de transição
- $q_0 \in E$  é o estado inicial
- $F$  é um subconjunto de  $E$  que contém todos os estados finais de  $M$

**Exemplo 1:** Considere o autômato que lê o alfabeto  $A=\{a,b\}$  e tem somente dois estados  $E = \{ q_0, q_1 \}$ . Vamos supor que  $F=\{ q_0 \}$ , isto é,  $q_0$  é o estado inicial e também o estado final.

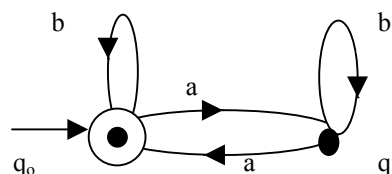
Vamos descrever a função de transição, usando as seguintes condições:

1. Lendo  $a$ , a máquina sempre muda de estado, ou seja,  $\forall q \in E, \delta(q,a) \neq q$ .
2. Lendo  $b$ , a máquina não muda de estado, ou seja,  $\forall q \in E, \delta(q,b) = q$ .

Resumidamente,

M	a	b
$q_0$	$q_1$	$q_0$
$q_1$	$q_0$	$q_1$

A dinâmica da máquina fica mais evidente se a representarmos por um grafo:



O círculo em  $q_0$  representa o estado final.



**Exemplo 2 (Mulher de Fases):** Ana acorda pela manhã e decide logo cedo como vai ser o seu dia. Se o dia amanhecer ensolarado, ela decide ir às compras, se amanhecer chuvoso ela decide arrumar a casa. Ela fica feliz se a casa ficar toda limpa (no caso dela ficar em casa), ou se ela comprar um par de sapatos novos (no caso de ir às compras). Se nenhuma dessas possibilidades ocorrer, ela fica triste. Nem todos os acontecimentos entretanto mudam o humor de Ana; por exemplo, se está arrumando a casa, ela não fica nem alegre nem triste por não ter realizado atividades ligadas às compras. E, uma vez que o humor de Ana fica estabelecido, ele não muda mais durante todo o dia, ou seja, uma vez feliz, sempre feliz, uma vez triste, sempre triste. Evidentemente Ana quer ser feliz, e portanto sua meta final é atingir a felicidade cotidiana. Podemos modelar o comportamento de Ana através de um autômato finito? Bem, tentaremos.

Sejam:

$E = \{\text{acordar, ir às compras, arrumar a casa, feliz, triste}\}$

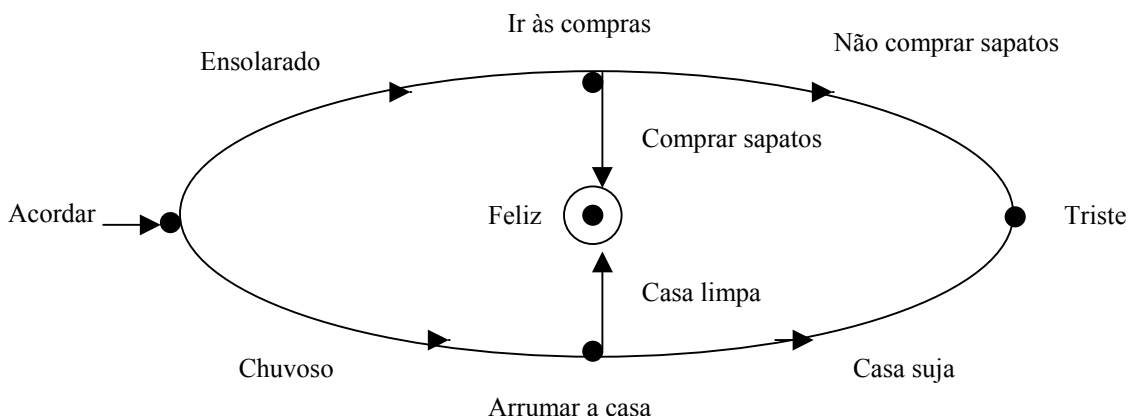
$A = \{\text{ensolarado, chuvoso, comprar sapatos, não comprar sapatos, casa limpa, casa suja}\}$

$q_0 = \text{acordar}$

$F = \{\text{feliz}\}$

Causas →	ensolarado	chuvoso	Comprar sapatos	Não comprar sapatos	Casa limpa	Casa suja
Estados ↓						
acordar	Ir às compras	Arrumar a casa	acordar	acordar	acordar	acordar
Ir às compras	Ir às compras	Ir às compras	feliz	triste	Ir às compras	Ir às compras
Arrumar a casa	Arrumar a casa	Arrumar a casa	Arrumar a casa	Arrumar a casa	feliz	triste
feliz	feliz	feliz	feliz	feliz	feliz	feliz
triste	triste	triste	triste	triste	triste	triste

O diagrama de estados neste caso é:



Voltemos ao estudo de linguagens e máquinas. Até agora os autômatos só são capazes de ler palavras com um único símbolo, precisamos fazer com que ele leia palavras com mais símbolos, ou seja, precisamos fazer com que ele atue sobre elementos de  $A^*$ , ao invés de operar apenas sobre elementos de  $A$ . Isto será feito estendendo a função  $\delta$  para  $E \times A^* \rightarrow E$ . Se  $p \in A^*$  é uma palavra,  $p = p_1 \dots p_n$ , como devemos ensinar  $M$  a ler  $p$  quando estamos num estado  $q$ ? Ora, da mesma maneira que acontece quando digitamos uma palavra no computador, digitando uma letra de cada vez.

Primeiro  $M$  lê  $p_1$  que a coloca no estado  $\delta(q, p_1)$ , então lê  $p_2$  que a põe no estado  $\delta(\delta(q, p_1), p_2)$ . Este resultado nós denotamos por  $\delta(q, p_1 p_2)$ . Em geral, tendo definido  $\delta(q, p_1 p_2 \dots p_k)$ , podemos definir

$$\delta(q, p_1 p_2 \dots p_k p_{k+1}) = \delta(\delta(q, p_1 p_2 \dots p_k), p_{k+1})$$

Se definirmos ainda  $\delta(q, \phi) = q$ ,  $\forall q \in E$  (ou seja  $M$  não muda de estado se nenhum *input* for entrado), obtemos a extensão  $\delta : E \times A^* \rightarrow E$  e assim  $M$  lê qualquer palavra de  $A^*$ . Já que  $M$  pode ler qualquer palavra de  $A^*$  ele pode aceitar ou não algumas delas. Mais precisamente,

Def: Um autômato finito  $M = (E, A, \delta, q_0, F)$  aceita uma palavra  $p \in A^*$  se  $\delta(q_0, p)$  pertence a  $F$ , isto é,  $M$  aceita  $p$  se  $M$ , começando de seu estado inicial chegar a um de seus estados finais lendo  $p$ .

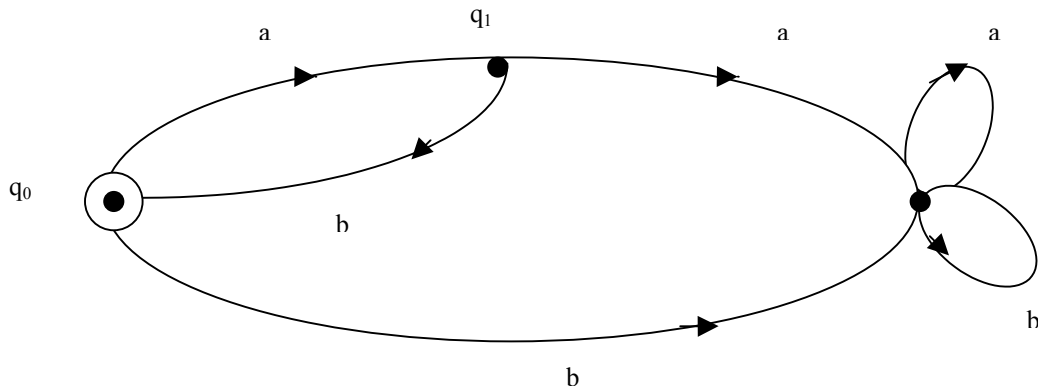
Def: A linguagem aceita por uma máquina  $M$  é  $L(M) = \{ p \in A^* / M \text{ aceita } p \}$ .

Exemplos:

1. No exemplo 1 acima  $L(M)$  é o conjunto de todas as palavras sobre  $\{a, b\}$  que têm um número par de  $a$ 's (incluindo as palavras com nenhum  $a$ ).
2. Considere o autômato Mulher de Fases. Quais são as "palavras" aceitas por Ana? São precisamente aquelas que deixam Ana feliz. Veja os roteiros possíveis: ela acorda, o sol está brilhando no céu, ela vai às compras e adquire um novo par de sapatos ou então ela desperta, o dia está chuvoso, ela arruma a casa e a casa fica limpa.
3. Alguns autômatos são muito idiotas, por exemplo:

$M = (E, A, \delta, q_0, F)$ . Se  $F = E$  então  $L(M) = A^*$ . Se  $F = \phi$ ,  $L(M) = \phi$ .

4. Seja  $A = \{ a, b \}$  e  $M$  o autômato com o seguinte diagrama de fase:



O estado  $q_0$  é tanto o estado inicial, como o estado final.

Neste caso  $L(M)$  consiste da palavra  $\phi$  e as palavras de comprimento par com símbolos alternados  $a$  e  $b$ , começando com  $a$  e terminando com  $b$ . Observe o ciclo que se fecha entre  $q_0$  e  $q_1$ . Assim  $abab\dots ab \in L(M)$ .

Note que em todo diagrama completo de estado, cada vértice tem exatamente uma seta emanando para cada símbolo de alfabeto. Se nenhuma seta marcada com  $p$  emana de  $q$ , então  $\delta(p,q)$  não está bem definida. Como  $\delta$  é função, não pode ocorrer de duas setas emanarem de  $q$  com a mesma palavra marcada  $p$ .

Def: Uma linguagem  $L$  é regular se existir um autômato finito  $M$  tal que  $L(M) = L$ .

Exemplos:

- Todas as linguagens vistas anteriormente nos exemplos são regulares.
- Toda linguagem finita é regular.

Existem entretanto linguagens que não são regulares. Para verificar isto precisamos do Lema do Bombeamento, que veremos a seguir. Antes porém, precisamos de uma pequena definição:

Def: Se  $A$  é um alfabeto e  $x$  e  $y$  são palavras de  $A^*$ , a concatenação de  $x$  e  $y$  é a palavra que se obtém escrevendo-se primeiro  $x$ , depois  $y$ .

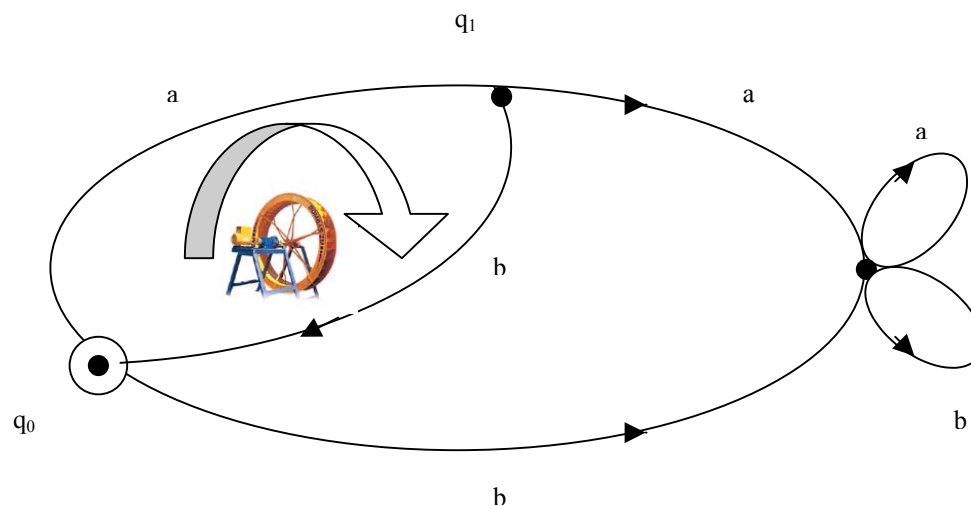
Ex: Se  $x=\text{gato}$ ,  $y=\text{rato}$ ,  $xy=\text{gatorato}$ ,  $x^2y=\text{gatogatorato}$ .



**Lema do Bombeamento:** Se  $A$  é um alfabeto e  $L \subset A^*$  é uma linguagem regular infinita, então existem palavras  $x$ ,  $y$  e  $z$  em  $A^*$ ,  $y \neq \emptyset$ , tal que  $xy^n z \in L$ , qualquer que seja o número natural  $n \in \mathbf{N}$ .

Demonstração: Como  $L$  é regular, existe um autômato finito  $M$  tal que  $L = L(M)$ . Vamos supor que  $M$  tem  $k$  estados. Como  $L$  é infinito, existe uma palavra  $p$  pertencente a  $L$  de comprimento maior do que  $k$ . Como  $M$  lê os símbolos de  $p$  e ele anda de estado em estado e devido a  $p$  ter mais símbolos do que  $M$  de estados, deve existir um estado  $q$  no qual  $M$  entra duas vezes. Assim, existem palavras  $x, y, z$ ,  $y \neq \emptyset$ , tal que  $p = xyz$  ( $x, y, z$  são as sílabas de  $p$ ) de modo que  $M$  está no estado  $q$  antes e depois de ler  $y$ . Então  $M$  aceita a palavra mais longa  $xyyz$ , pois, após ler a sílaba do meio  $y$ , a máquina está no estado  $q$ , podendo ler novamente  $y$  e então passar a ler  $z$  e entrar num estado final de aceitação. Segue por indução que  $M$  também aceita  $xy^n z$ ,  $n \in \mathbf{N}$ .

Observe que a inclusão de um novo  $y$  na palavra  $xyz$  é feita, via diagrama de estados percorrendo-se um ciclo fechado 2 vezes ao invés de apenas uma vez.



Ex: Considere a linguagem  $L \subset \{a,b\}^*$  constituindo de todas as palavras formadas por uma seqüência de  $a$ 's seguidas de uma seqüência de  $b$ 's de mesmo tamanho.

$$L = \{a^n b^n / n > 0\}$$

A linguagem  $L$  não é regular pois, se fosse, pelo lema do bombeamento, existiriam palavras  $x, y, z$  em  $\{a, b\}^*$ , com  $y \neq \emptyset$ , tal que  $\forall n \in \mathbf{N}$ ,  $xy^n z \in L$ . Mas  $y$  não pode ser uma seqüência de  $a$ 's, pois, quando  $n$  é grande, a palavra  $xy^n z$  teria mais  $a$ 's do que  $b$ 's pois  $x$  e  $z$  têm um número limitado de  $a$ 's ou de  $b$ 's.

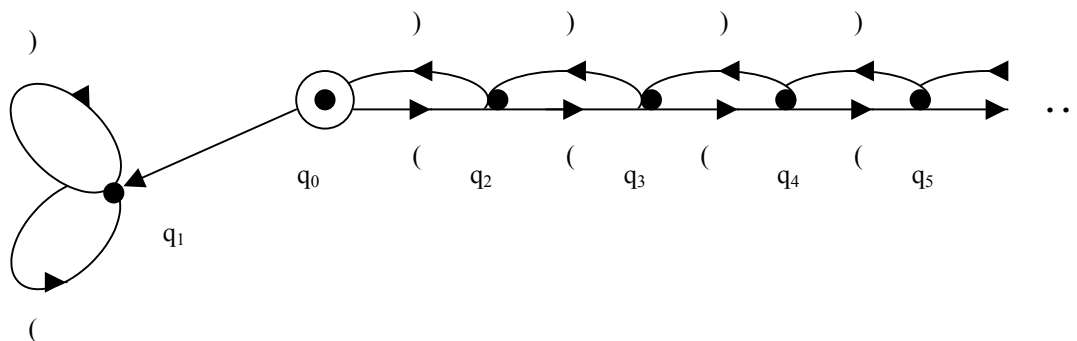
Analogamente,  $y$  não pode ser formado apenas por  $b$ 's. Finalmente, se  $y$  tivesse  $a$ 's e  $b$ 's então a palavra  $xy^2z$  não teria a forma  $a^n b^n$  e também não pertenceria a  $L$ . Assim  $L$  não é regular.

Este exemplo mostra que nenhum autômato finito tem memória suficiente para contar e guardar um número arbitrariamente grande de  $a$ 's para que ele possa conferir se uma palavra inicialmente formada por  $a$ 's é seguida do mesmo número de  $b$ 's (deficiência de *hardware*). Para sanar esta deficiência um novo autômato deve ser criado. Os **autômatos de pilha** conseguem aceitar a linguagem do exemplo anterior. Grosseiramente falando, este autômato tem uma memória expandível modelada de modo similar ao empilhamento que fazemos com pratos vazios.

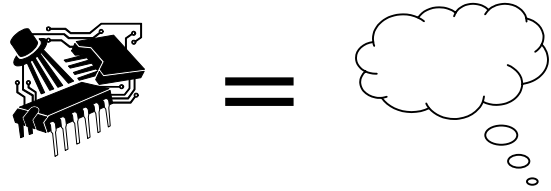
Ex: (parêntesis que se abrem e se fecham)

Os parêntesis usados em expressões algébricas constituem uma interessante linguagem. A todo parêntesis que é aberto "(", deve corresponder outro que o fecha ")". Assim  $( a + b ) )$  não é uma expressão aceitável, mas  $( a + b )$  é.

Seja  $L$  a linguagem formada por seqüências de parêntesis aceitáveis de acordo com a regra acima.  $L$  não é uma linguagem regular, mas é aceita por um autômato de pilha e também pelo autômato infinito:







## 7. COMO UMA MÁQUINA PODE SE LEMBRAR

*Porque um computador nunca esquece*

**Resumo:** Para linguagens com propriedades que requeiram a compreensão de regras que devem ser guardadas na memória, será apresentado e construído uma espécie de hardware mais sofisticado que um autômato finito. A experiência será feita adicionando-se ao autômato um tipo rudimentar de memória, chamada memória de pilha. Como exemplo, construiremos uma máquina que entende a linguagem dos palíndromos (que são palavras ou frases que resultam ser as mesmas quando lidas de trás para frente) e esclarecem o papel fundamental da memória na compreensão de linguagens. A seguir, através de circuitos lógicos simples, será apresentado com detalhes como funciona fisicamente a memória de um computador. Será estudado o funcionamento de um “flip-flop” para precisar a idéia de como armazenar um *bit* de informação. Para isto não serão necessários profundos conhecimentos de eletrônica, apenas os fundamentos do cálculo lógico proposicional (com conectivos e, ou e a negação).

### MEMÓRIA

Já vimos que uma linguagem é definida a partir de um alfabeto  $A$  (que nada mais é do que um conjunto não vazio de símbolos), formando-se o conjunto  $A^*$  de todas as combinações arbitrárias de símbolos e, dentro de  $A^*$ , selecionamos o subconjunto das palavras  $L$  que contém todas as palavras da linguagem.

Vimos também que é possível a construção de máquinas (isto é, computadores) que reconhecem se uma palavra pertence ou não a uma dada linguagem. Estas máquinas são aceitadoras ou reconhecedoras de linguagens. Elas também podem ser usadas para gerar linguagens.

Vimos como exemplo a caixa preta de um disco voador que reconhecia e gerava todas as palavras sobre o alfabeto  $A=\{a,b\}$ , formadas por um número par de a's e qualquer número de b's. Esta máquina é um protótipo de um autômato finito e a linguagem reconhecida por ela é regular, sendo bastante simples e previsível.

No entanto, nem toda linguagem pode ser reconhecida por um autômato finito. Um exemplo de uma tal linguagem é a formada por palavras palíndromas marcadas. O que seria isto? Uma palavra é dita palíndroma se podemos lê-la tanto da esquerda para a direita como da direita para a esquerda. Por exemplo, a palavra ovo é palíndroma (curiosidade: as frases 'a grama é amarga' e 'Socorram-me subi no ônibus em Marrocos' são palíndromas).

Sobre o alfabeto  $A=\{a,b,c\}$ , seja  $p$  uma palavra formada apenas pelas letras  $a$  e  $b$  e  $p'$  a palavra  $p$  escrita de trás para frente. Então  $L=\{pcp', p \text{ pertencente a } \{a,b\}^*\}$  não é uma linguagem regular, isto é, não pode ser reconhecida por um autômato finito. Este fato é consequência do lema do bombeamento, visto anteriormente.

Para reconhecer a linguagem das palavras palíndromas, chegamos a um ponto crucial da teoria das máquinas que aceitam linguagens, que é a memória. A maioria das pessoas entende a palavra memória com dois significados: um significado mais humano, ligado a lembranças do passado, à infância, e um significado mais técnico, mais recente, ligado à memória dos computadores. Este segundo pensamento é, para a maioria das pessoas, um pouco obscuro, e não entendemos de modo claro como uma máquina possa ter habilidades humanas.

Veremos que não há mistérios para entender a 'memória' de uma máquina (já a memória humana só mesmo Freud).

Um bom indicativo de como funciona a memória de uma máquina é o jogo de xadrez com caixas de fósforos, que estudamos no Capítulo 3. A cada jogada, percebemos que as informações anteriores ficam guardadas dentro das caixas de fósforos, impedindo que jogadas erradas sejam feitas novamente e consequentemente levando, mais e mais, a jogadas vencedoras.

Voltemos ao nosso estudo de linguagens. Para arquitetar uma máquina que reconheça palavras palíndromas, o uso de autômatos finitos não pode ser feito devido a este não ter memória suficiente para checar se uma palavra  $p$  de comprimento grande seja repetida na ordem inversa  $p'$ . É necessário armazenar a palavra  $p$ , para testar se depois de lido o símbolo intermediário  $c$ , aparece a mesma palavra na ordem inversa. Para isto devemos sofisticar o 'hardware'.



## Autômatos de Pilha

Um autômato de pilha é uma máquina que tem uma memória expandível modelada de modo análogo ao que fazemos com pratos empilhados. Vamos descrever como ele é, como funciona e o que é a sua memória, através de um exemplo de uma máquina que reconhece palavras palíndromas. As linguagens reconhecidas por autômatos de pilha são chamadas linguagens livres de contexto. O porquê deste nome ficará claro posteriormente.

Um autômato de pilha é um autômato finito munido de um dispositivo onde se pode empilhar discos um sobre os outros (a pilha). Um símbolo só pode ser adicionado à pilha no topo e só pode ser removido se estiver no topo da pilha. Quando um símbolo é adicionado ao topo da pilha, o símbolo previamente no topo torna-se o segundo a partir do topo, o segundo torna-se terceiro, etc. Quando um símbolo é removido do topo da pilha, o segundo símbolo prévio torna-se o primeiro, o terceiro torna-se o segundo e assim sucessivamente. Não é necessário que visualizemos a pilha inteira (ela pode estar contida num tubo, como por exemplo nos suportes para copinhos de café), é necessário apenas que conheçamos, a cada passo, quem está no topo da pilha.

Para os nossos propósitos, faremos a hipótese que a pilha pode se tornar arbitrariamente longa, de modo que podemos sempre adicionar tantos símbolos quanto desejarmos.

Vamos mostrar agora como usar uma pilha de pratos, acoplado a um controle finito para reconhecer a linguagem das palavras palíndromas. Faremos uso de um controle finito com dois estados:  $q_1$  e  $q_2$ , com três entradas (teclas) **a**, **b**, **c** e uma pilha de pratos com as cores azul, verde e vermelho.

1. A máquina começa com um prato vermelho na pilha e com o controle finito no estado  $q_1$ .
2. Se a tecla digitada for **a** e o equipamento está no estado  $q_1$ , um prato azul é colocado sobre a pilha. Se a tecla digitada for **b**, e o equipamento está no estado  $q_1$  e um prato verde é colocado no topo da pilha. Em ambos os casos a máquina continuará no estado  $q_1$ .
3. Se digitarmos o símbolo **c** e o equipamento estiver no estado  $q_1$ , ele mudará para o estágio  $q_2$  sem adicionar ou remover pratos.
4. Se a tecla digitada for **a** e o equipamento estiver no estado  $q_2$  com um prato azul no topo da pilha, este prato deve ser removido. Se a tecla digitada for **b** e o equipamento estiver no estado  $q_2$  com um prato verde no topo da pilha, este prato deve ser removido. Em ambos os caso o controle finito permanecerá no estado  $q_2$ .
5. Se o equipamento estiver no estado  $q_2$  e um prato vermelho estiver no topo da pilha, este prato deve ser removido sem esperar a próxima entrada de dados.

6. Em qualquer outra situação, o equipamento não pode realizar nenhum movimento.

**Importante:** Diremos que o equipamento aceita uma seqüência de símbolos, se, ao terminar de processar o último símbolo, a pilha de pratos estiver completamente vazia. Quando isto ocorrer, nenhum movimento adicional é permitido.

O equipamento opera basicamente do seguinte modo: no estado  $q_1$  ele adiciona pratos fazendo uma imagem do que está sendo digitado, colocando um prato azul no topo da pilha, cada vez que o símbolo digitado for **a** e um prato verde cada vez que o símbolo digitado for **b**. Quando **c** é digitado, o equipamento muda para o estado  $q_2$  (retirar pratos). A seguir, o restante da palavra é comparado com a pilha de pratos, removendo um prato azul do topo da pilha cada vez que digitarmos o símbolo **a** e um prato verde cada vez que digitarmos um símbolo **b**.

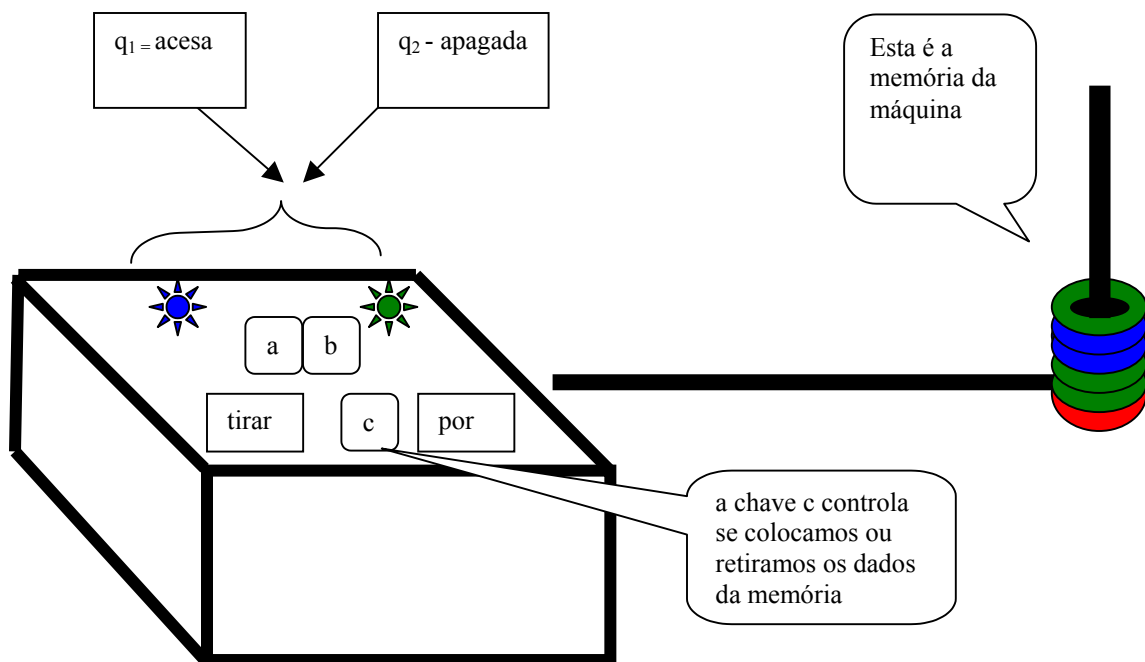
Pode acontecer do prato que está no topo da pilha ser da cor errada e nenhum movimento adicional é possível; neste caso a palavra não será aceita.. Se todos os pratos combinarem com o que foi digitado, com certeza o prato vermelho do fim da pilha será exposto, ele será imediatamente removido e o equipamento aceitará a palavra.

A pilha se tornará vazia somente no caso em que a palavra digitada após o aparecimento de **c** for igual à palavra digitada antes de **c**, mas escrita de trás para frente. Assim, este autômato aceita todas as palíndromas e somente elas.

O autômato de pilha descrito aqui pode ser fabricado utilizando-se 4 lâmpadas e duas pequenas chaves duplas de liga/desliga. Sua memória pode ser confeccionada com uma pilha de discos coloridos de cartolina. Manuseie-a e descubra o seu funcionamento!

## ESQUEMA

Prato do topo	Estado	a	b	c
Azul	q <sub>1</sub>	Adicione um prato azul e fique no estado q <sub>1</sub>	Adicione um prato verde e fique no estado q <sub>1</sub>	Passe para o estado q <sub>2</sub>
	q <sub>2</sub>	Remova o prato do topo e fique no estado q <sub>2</sub>	-----	-----
Verde	q <sub>1</sub>	Adicione um prato azul e fique no estado q <sub>1</sub>	Adicione um prato verde e fique no estado q <sub>1</sub>	Vá para o estado q <sub>2</sub>
	q <sub>2</sub>	-----	Remova o prato do topo e fique no estado q <sub>2</sub>	-----
Vermelho	q <sub>1</sub>	Adicione um prato azul e fique no estado q <sub>1</sub>	Adicione um prato verde e fique no estado q <sub>1</sub>	Vá para o estado q <sub>2</sub>
	q <sub>2</sub>	Sem esperar a próxima entrada, remova o prato do topo	Sem esperar a próxima entrada, remova o prato do topo	Sem esperar a próxima entrada, remova o prato do topo



## MEMÓRIA ELETRÔNICA

Vimos que um computador deve ter um mecanismo de memória para reconhecer linguagens mais elaboradas que as linguagens regulares. O aparelho que estudamos anteriormente tem uma memória formada por uma pilha onde se colocam ou se retiram discos, dependendo da tarefa que estamos realizando. Este procedimento é puramente mecânico e está de certa forma distante do funcionamento de nossos atuais computadores. O que faremos a seguir é estudar algumas noções ligadas a circuitos elétricos, que podem desempenhar a função de guardar algum dado, isto é, tentaremos entender como um circuito elétrico pode se lembrar.

Quando estudamos lógica elementar, nos deparamos com a questão de juntar sentenças a fim de formar novas afirmações a partir de outras já conhecidas. Um exemplo simples: Pedro usa óculos **e** Mário é elegante. Esta sentença tem um valor-verdade que pode ser verdadeiro ou falso, dependendo da validade das sentenças que a compõem. Se de fato Pedro usar óculos e Mário for elegante, nossa frase será obviamente verdadeira. Se, ao invés disso, Pedro não usar óculos ou Mário não for elegante, ou ambos, a frase será falsa. Isto pode ser resumido na seguinte tabela:

A = Pedro usa óculos

B = Mário é elegante

A	B	A e B
V	V	V
V	F	F
F	V	F
F	F	F

V significa verdadeiro e F falso.

Do mesmo modo podemos juntar sentenças usando o conectivo **ou**. Do ponto de vista matemático o **ou** não é exclusivo, assim a frase A ou B é verdadeira, se A for verdadeira ou B for verdadeira ou ambas forem verdadeiras. Sua tabela-verdade é:

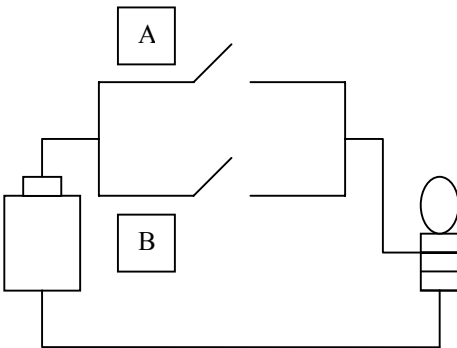
A	B	A ou B
V	V	V
V	F	V
F	V	V
F	F	F

Além dos conectivos **e** e **ou**, usamos muito freqüentemente a negação **não**, cuja tabela-verdade é:

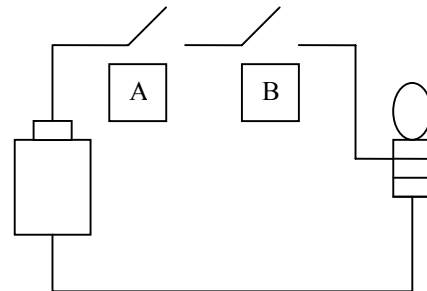
A	não A
V	F
F	V

Pois bem, estes três conectivos lógicos podem ser simulados por circuitos elétricos, o que passaremos a descrever a seguir. A passagem de corrente no circuito significa que sentença é verdadeira e a ausência de corrente significa que ela é falsa. No circuito **e** ambas as chaves precisam estar ligadas para que a lâmpada acenda, indicando um fato verdadeiro. Já no circuito **ou**, basta ligar uma das chaves para que a lâmpada acenda.

Ex 1: Circuito **ou**:

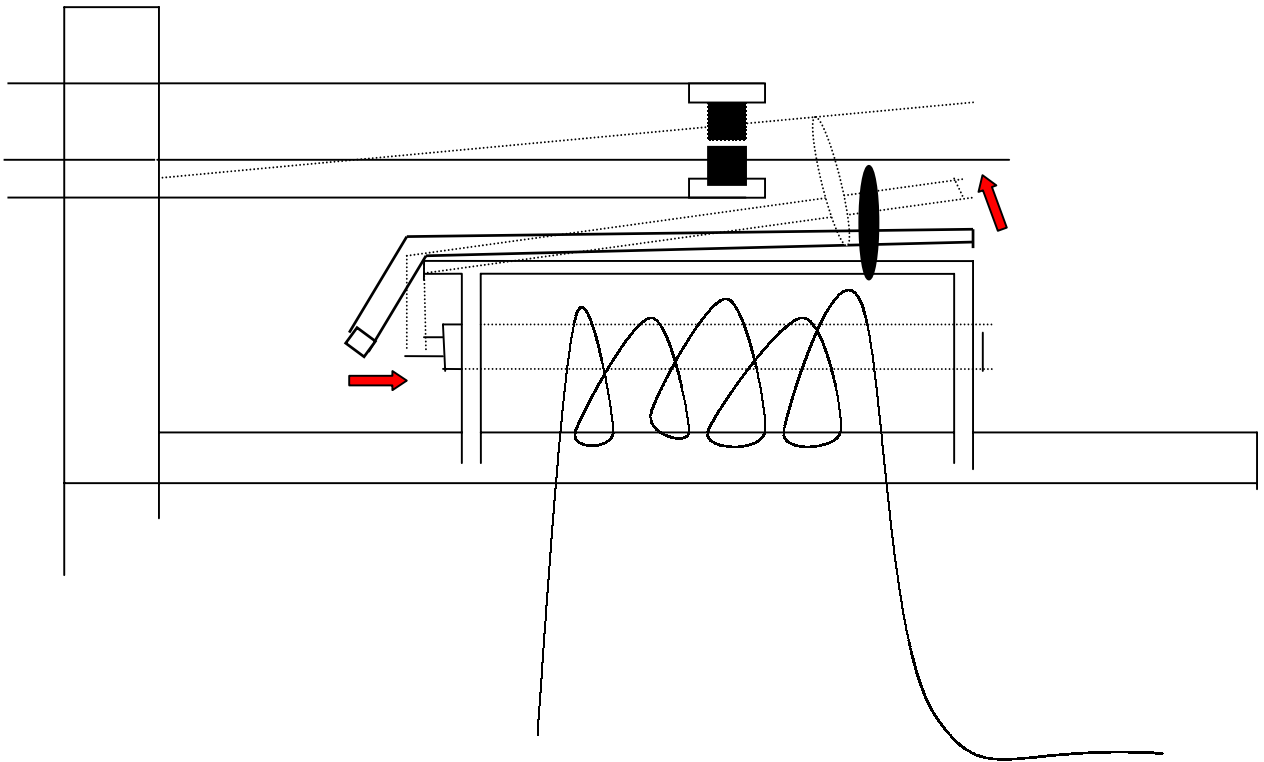


Ex 2: Circuito **e**:

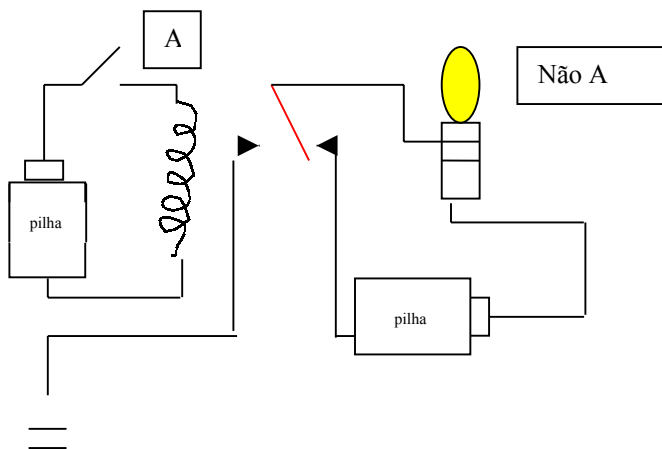


Como será que podemos fazer um circuito dizer **não**? Quando passar corrente, queremos que a lâmpada se apague e quando faltar corrente queremos que ela se acenda. Isto parece impossível, não é mesmo? Existe entretanto um circuito elétrico que fará justamente esta tarefa. Seu nome é relê, e passaremos a descrevê-lo sucintamente a seguir.

## Modelo de um Relê



Quando passa corrente pela espiral, a bobina fica imantada, a alavanca levanta e faz o contato. Esquemáticamente, um relê pode ser usado para representar o **não** lógico:

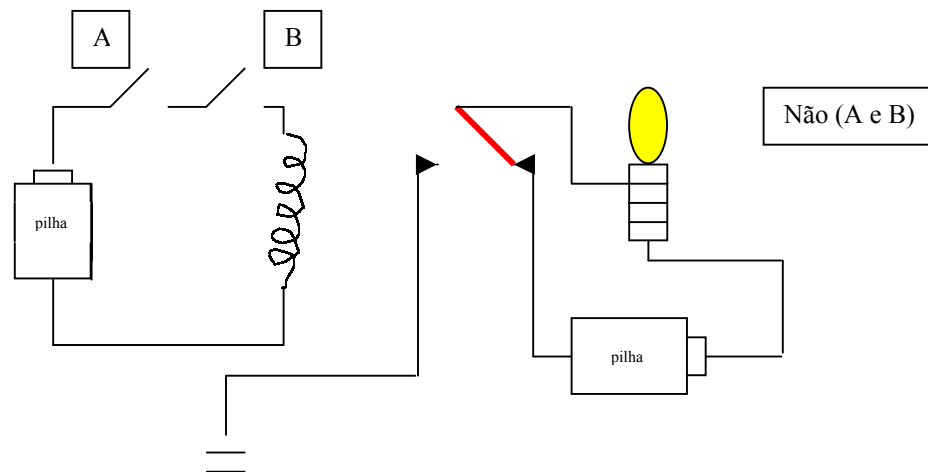


Observe que a chave A está desligada (indicando que A é falsa), mas a lâmpada está acesa (indicando que não A é verdadeira)! Quando ligamos a chave em A, A passa a ser verdadeira e o relê entra em funcionamento. A chave

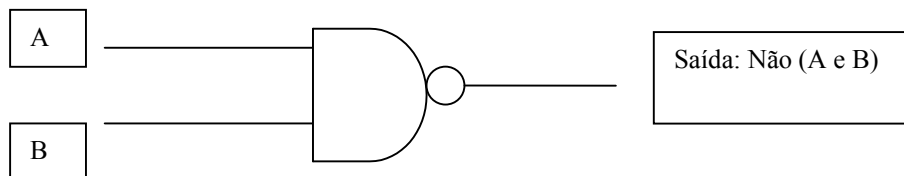
móvel vermelha se desloca, quebrando o contato que faz a lâmpada acender. Novamente neste caso temos a negação: A é verdadeira e não-A falsa.

O circuito abaixo é uma combinação do **não** com o **e**, e sua tabela-verdade é a seguinte:

A	B	Não (A e B)
V	V	F
V	F	V
F	V	V
F	F	V

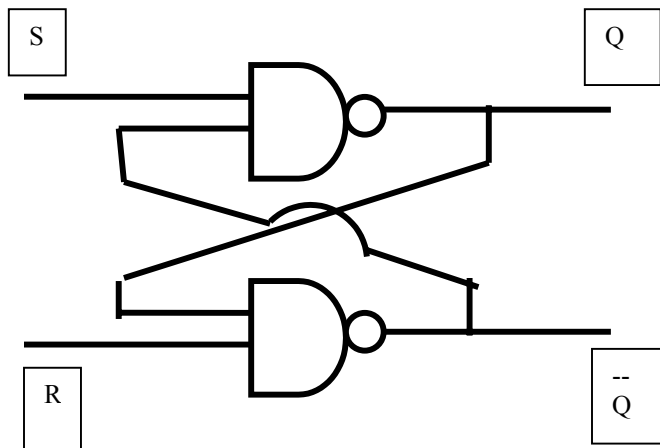


Como este desenho é de certa forma complicado para reproduzi-lo a toda hora, vamos representá-lo pelo seguinte esquema:



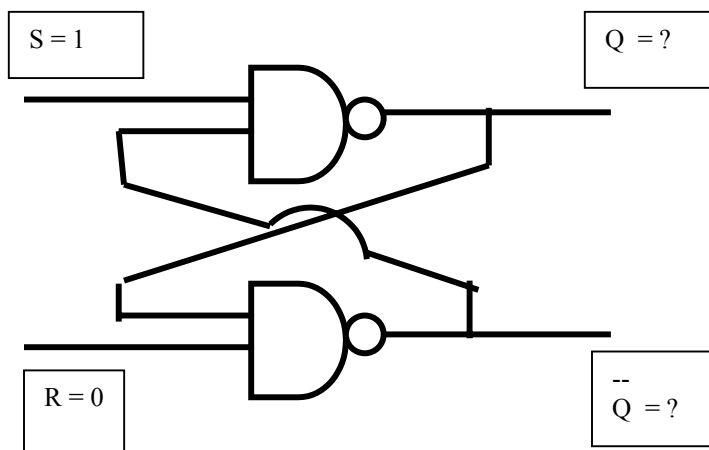
Se amarrarmos dois circuitos **não-E**, um engolindo o rabo do outro como no esquema a seguir, obteremos um **FLIP- FLOP**. Veremos que o Flip-Flop tem memória! É uma memória um tanto quanto idiota, mas ele se lembra de algo e este algo é suficiente para entendermos como armazenar dados num computador e entender a sua lógica.

### Modelo de um FLIP-FLOP



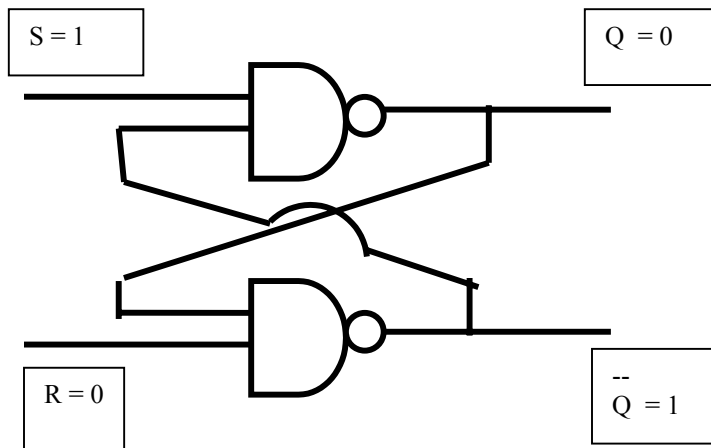
Os circuitos elétricos vistos anteriormente só mantinham as saídas enquanto se mantinham as entradas. Eles não eram capazes de armazenar ou se lembrar de dados se desligássemos as entradas. Um computador projetado com estes circuitos, quando desligado, perderia todo o trabalho realizado. Já com os Flip-Flops, isto não acontece pois eles podem manter uma saída indefinidamente, não se importando se modificarmos suas entradas. Vejamos como isto pode ser feito. O número 1 indicará passagem de corrente elétrica e o número 0 a ausência de corrente elétrica.

CASO 1:  $S = 1$  e  $R = 0$ . Como serão as saídas  $Q$  e  $Q^{-}$  ?

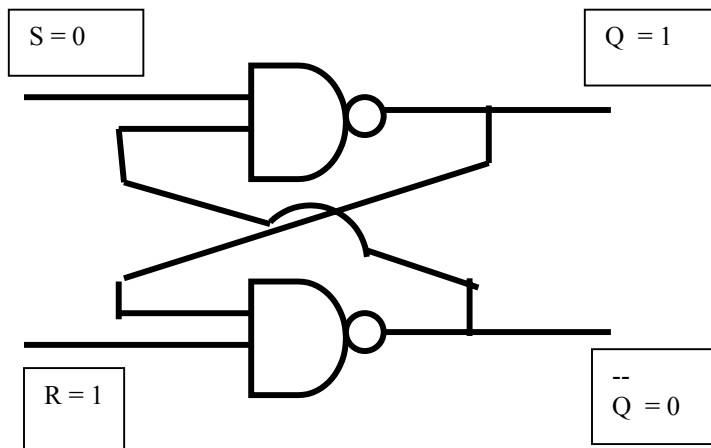




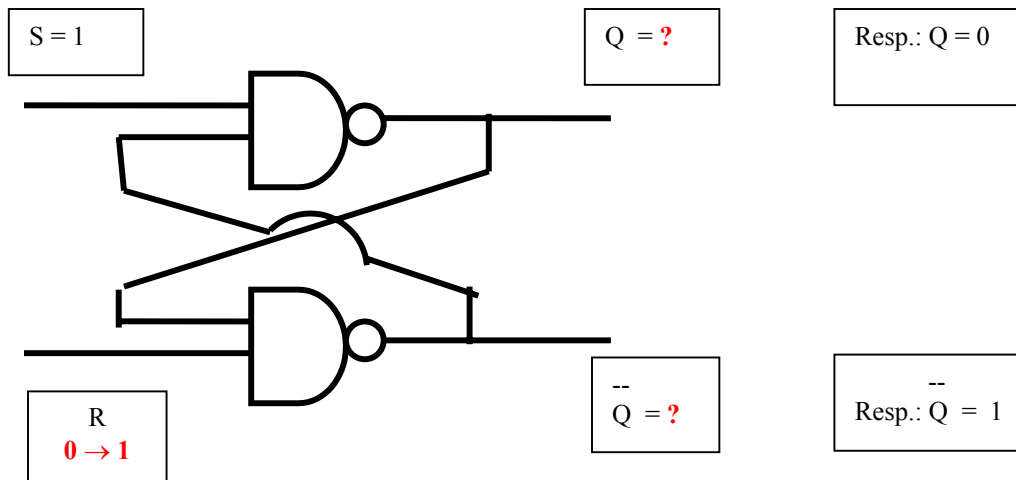
É fácil ver que  $Q^- = 1$ , pois o segundo não-e do flip-flop tem uma entrada 0 e isto é o suficiente para que a saída também seja 1. Se realimentarmos o primeiro não-e com este valor de  $Q^-$ , obteremos que  $Q = 0$ :



CASO 2:  $S = 0$  e  $R = 1$ . Este caso é muito parecido com o anterior e a resposta é a seguinte:

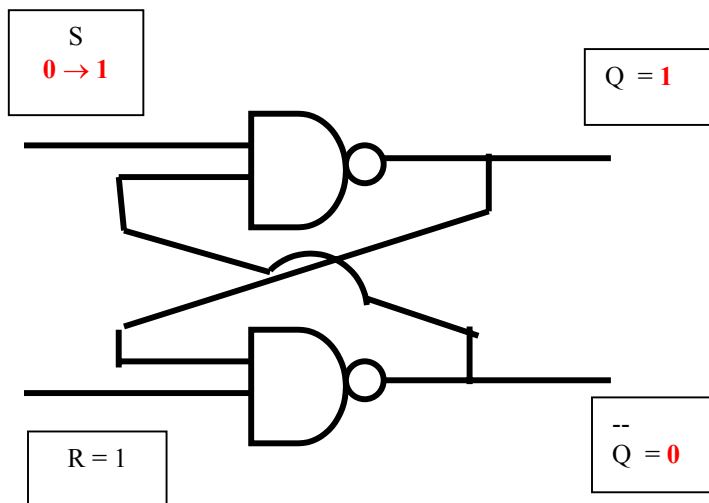


CASO 3: Vamos mudar as entradas. Suponhamos que inicialmente o flip-flop esteja com as entradas  $S = 1$  e  $R = 0$ , como no caso 1. Repentinamente mudamos a chave Q, alterando-a de 0 para 1. O que acontecerá com as saídas do flip-flop? Observe a figura:



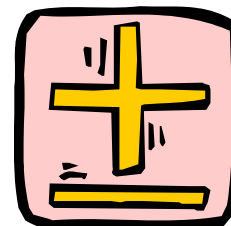
Neste caso, não acontece nada, isto é, as saídas não são afetadas pela mudança de 0 para 1 na entrada R. Esta é uma importante propriedade do flip-flop e ela ocorre porque as duas entradas da porta não-e inferior ficam 0 e 1 e portanto sua saída é ainda  $\bar{\bar{Q}} = 1$ ; realimentando este valor na primeira porta não-e, teremos 1 e 1 com *input* e portanto a saída Q ainda é 0.

CASO 4: Vamos mudar novamente as entradas. Suponhamos que inicialmente o flip-flop esteja com as entradas S = 0 e R = 1, como no caso 2. Mudamos agora a chave S de 0 para 1. O que acontecerá com as saídas do flip-flop? Observe a figura:



O mesmo argumento acima mostra que nada acontecerá com as saídas.

CONCLUSÃO: O flip-flop pode reter uma informação passada, já que as mesmas entradas S = 1 e R = 1 produzem saídas diferentes, dependendo das entradas anteriores, ou seja, os flip-flops têm memória!



## 8. GRAMÁTICA COMBINA COM MATEMÁTICA?

*Organizando as linguagens que são entendidas por máquinas*

**Resumo:** Neste tópico mostraremos uma conexão entre duas áreas aparentemente díspares: Português e Matemática. São apresentados novos enfoques no interrelacionamento entre linguagens e máquinas, dando atenção especial às linguagens geradas por máquinas, não somente as reconhecidas por elas. As regras sintáticas mais simples do Português serão analisadas afim de transportá-las para as linguagens associadas às máquinas.

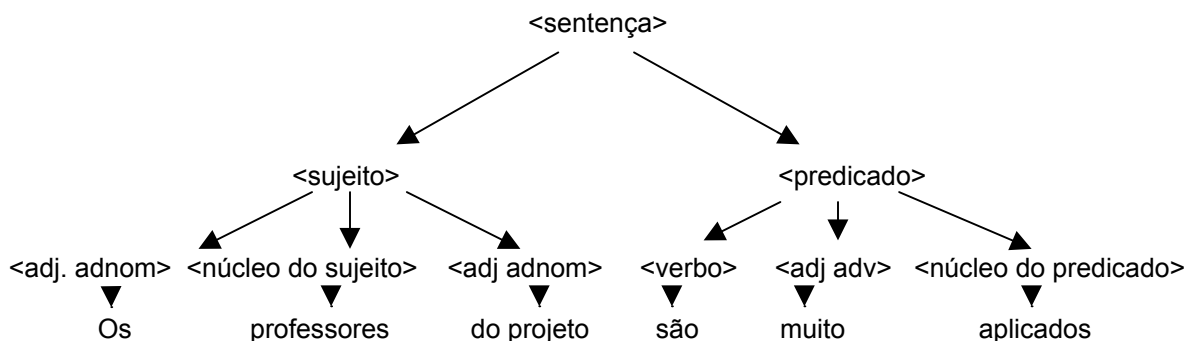
### GRAMÁTICA

A gramática é usualmente estudada como um ramo da Lingüística. O objetivo dos estudos gramaticais, pelo menos do ponto de vista inicial, é definir quando uma dada sentença é válida ou não na linguagem que está sendo usada e também fornecer descrições sobre as estruturas das sentenças.

Uma das metas atuais do trabalho dos cientistas da computação com lingüistas é procurar uma gramática formal que possa descrever a linguagem natural, de modo que um computador possa “entender” as diversas línguas humanas. Isto será muito útil para traduções, para verificar se uma sentença está correta, bem formulada, etc. Este é um projeto bastante difícil de ser concretizado, devido à fluidez de nossa comunicação (entonações, gírias, sarcasmos, cinismos; como controlar a criatividade humana?).

Vamos estudar aqui algumas gramáticas que podem ser entendidas por computadores; por isso mesmo elas são simples e bem mais restritas que a linguagem humana.

Considere a sentença: Os professores do projeto são muito aplicados. Uma análise simples desta sentença permite construir o seguinte diagrama:



Este é um exemplo de uma frase correta gramaticalmente (e também semanticamente). Observe que a classe das sentenças, aqui denotada por <sentença>, “ recebe os valores” <sujeito> e <predicado>, que ainda são classes. Por sua vez, <sujeito> recebe os valores <adjunto adnominal> e <núcleo do sujeito>, que são novamente classes e estes últimos recebem os valores ‘Os’, ‘professores’ e ‘do projeto’, que não são mais classes mas sim palavras terminais ou finais de nossa análise.

É necessário colocar as classes entre os símbolos < > pois elas são palavras da nossa língua e queremos diferenciar entre a classe <verbo> e a palavra terminal ‘verbo’. Por exemplo, na frase: “O verbo conhecer termina em er”, usamos verbo como uma palavra terminal com significado completamente diferente da classe <verbo>.

O esquema exposto acima também pode ser usado para gerar sentenças, de modo natural: começamos com a classe inicial <sentença>, a seguir colocamos nesta classe as (sub) classes <sujeito> e <predicado>, posteriormente atribuímos a estas novas classes outras classes e assim sucessivamente até só termos palavras terminais.

Podemos formar uma quantidade enorme de sentenças por este processo. A maioria das frases obtidas pode não fazer sentido mas, do ponto de vista gramatical, elas são corretas num sentido amplo. As classes <sentença>, <sujeito>, etc. são chamadas variáveis e palavras em si (como professores), são chamadas simplesmente de terminais.

As relações existentes entre variáveis com outra variáveis (entre classes e subclasses) e entre variáveis e terminais são chamadas de produções.

Exemplos de produções:

<predicado>  $\longrightarrow$  <núcleo do predicado><adjunto adverbial>

<verbo>  $\longrightarrow$  são

Uma das variáveis deve ter um papel especial (no nosso caso <sentença>), pois, a partir dela as produções gerarão as sentenças da linguagem. Esta variável inicial será denotada por **I** (I de inicial).

**Definição:** Uma gramática  $G$  é um conjunto de 4 objetos:

$V$ : um conjunto de variáveis

$T$ : um conjunto de terminais

$P$ : um conjunto de produções

$I$ : variável inicial.

Vamos supor sempre que  $V$  e  $T$  não tenham elementos em comum, para não confundir variáveis com terminais.

O conjunto  $P$  das produções é formado por expressões da forma:

$\alpha \longrightarrow \beta$

com  $\alpha$  uma seqüência de símbolos no alfabeto  $V$ , diferente da palavra vazia, e  $\beta$  uma seqüência em  $V^*$  que é o conjunto das combinações arbitrárias de símbolos em  $(V \cup T)^*$ .

A linguagem gerada pela gramática  $G$  é o conjunto de todas as palavras terminais que podem ser obtidas, a partir da variável inicial  $I$ , usando-se sucessivas vezes as produções de  $G$ . Simbolicamente, se denotarmos por  $L(G)$  a linguagem gerada pela gramática  $G$ , teremos

$L(G) = \{ p / p \in T^* \text{ e } I \rightarrow \alpha_1 \rightarrow \alpha_2 \rightarrow \dots \rightarrow \alpha_n \rightarrow p \}$

Exemplo:  $G = \{V, T, P, I\}$ ,  $V = \{I\}$ ,  $T = \{0, 1\}$ ,  $P = \{I \rightarrow 0I1, I \rightarrow 01\}$

Neste caso  $I$  é a única variável, 0 e 1 são os únicos terminais. Só existem duas produções

$I \rightarrow 0I1$  e  $I \rightarrow 01$

As palavras obtidas são do tipo  $0^n 1^n$ , isto é, são formadas inicialmente por zeros seguidas de um mesmo número de uns. Basta aplicar sucessivas vezes as produções:

$I \rightarrow 0I1 \rightarrow 00I11 \rightarrow \dots \rightarrow 0^{n-1}I1^{n-1} \rightarrow 0^n 1^n$

## Algumas Gramáticas Especiais:

### 1) Gramáticas de Representação Finita

As linguagens que as máquinas entendem precisam ter representação finita, isto é, um número finito de produções. Como vimos anteriormente, nem todas as linguagens são assim. As máquinas que aceitam e reconhecem linguagens finitamente geradas são as Máquinas de Turing. As linguagens aceitas por estas máquinas são chamadas linguagens recursivamente enumeráveis. Dedicaremos o próximo capítulo ao entendimento de tais máquinas, suas linguagens e aplicações.

Antes porém apresentaremos algumas linguagens mais simples.

### 2) Gramáticas sensíveis ao contexto.

Seja  $G=\{V,T,P,I\}$  uma gramática. Se as produções em  $P$  forem do tipo  $\alpha \rightarrow \beta$ , com o comprimento da palavra  $\alpha$  menor ou igual ao da palavra  $\beta$ , dizemos que  $G$  é sensível ao contexto. As máquinas que aceitam estas linguagens recebem o nome de autômatos lineares limitados. São muito parecidos com as máquinas de Turing que estudaremos mais adiante.

Exemplo:  $G = \{ V, T, P, I \}$ ,  $V=\{I, B, C\}$ ,  $T=\{a,b,c\}$  e  $P$  é o conjunto de produções:

$I \rightarrow aIBC$ ,  $I \rightarrow aBC$ ,  $CB \rightarrow BC$ ,  $aB \rightarrow ab$ ,  $bB \rightarrow bb$ ,  $bC \rightarrow bc$ ,  $C \rightarrow c$

Observe que os comprimentos dos elementos à esquerda das setas é sempre menor ou igual aos da direita. Neste caso, a linguagem reconhecida por  $G$  é  $L(G) = \{ a^n b^n c^n, n = 0, 1, 2, \dots \}$ . Esta linguagem é sensível ao contexto.

Numa linguagem sensível ao contexto, as produções têm a forma:  $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ , isto é,  $A$  pode ser trocada por  $\beta$  somente quando  $A$  aparecer no contexto (entre)  $\alpha_1$  e  $\alpha_2$ .

Observe a analogia com a linguagem natural:

<adj. adnominal> <sujeito><verbo>  $\rightarrow$  Os alunos estudam

O sujeito, no contexto entre o adjunto adnominal e o verbo, produz uma frase aceitável no português usual. No entanto,

<adj. adnominal> <verbo><sujeito>  $\rightarrow$  Os estudam alunos

não é uma frase aceitável. Assim, a troca de uma variável por uma palavra terminal depende do contexto.

### 3) Linguagens livres de contexto

Uma gramática  $G = \{V, T, P, I\}$  cujas produções são todas da forma  $\alpha \rightarrow \beta$ , onde  $\alpha$  é uma variável simples e  $\beta$  é uma seqüência qualquer de símbolos, é chamada de gramática livre de contexto. Na linguagem gerada por esta gramática podemos trocar uma variável  $A$  por uma seqüência  $\beta$ , independentemente do contexto. As máquinas que aceitam esta linguagem são os autômatos de pilha.

Exemplo:  $G = \{V, T, P, I\}$ ,  $V = \{I, A, B\}$ ,  $T = \{a, b\}$  e  $P$  é o conjunto de produções:

$I \rightarrow aB, I \rightarrow bA, A \rightarrow a, A \rightarrow aI, A \rightarrow bAA, B \rightarrow b, b \rightarrow bI, B \rightarrow aBB.$

Observe que as variáveis que aparecem no primeiro membro de cada seta são simples. A linguagem aceita por esta gramática é livre de contexto e as palavras desta linguagem são formadas por um número igual de a's e b's.

### 4) Linguagens Regulares

São as linguagens mais simples, aquelas que são aceitas por autômatos finitos. Neste caso as únicas produções permitidas são da forma:

$A \rightarrow aB$  ou  $A \rightarrow a$ ,

onde  $A$  e  $B$  são variáveis e  $a$  é terminal.

Exemplo:  $G = \{V, T, P, I\}$ ,  $V = \{I, A, B\}$ ,  $T = \{a, b\}$  e  $P$  é o conjunto de produções:

$I \rightarrow aA, A \rightarrow bB, A \rightarrow \phi, B \rightarrow aA.$

A linguagem aceita é constituída por palavras começando com  $a$  seguido de um número qualquer de cópias de  $ba$  ( $\phi$  é a palavra vazia).

Apresentamos a seguir um resumo do que foi apresentado. Esta organização de máquinas e de linguagens é conhecida como hierarquia de Chomsky.

#### RESUMO

Tipo de Gramática	Tipo de Linguagem	Autômato que a reconhece
Regular	Regular	Autômato Finito
Livre de Contexto	Livre de Contexto	Autômato de Pilha
Sensíveis ao contexto	Algorítmica	Autômatos Lineares Limitados
Representação finita	Recursivamente Enumerável	Máquinas de Turing

## PROCEDIMENTOS E ALGORÍTMOS:

Um procedimento é uma seqüência finita de instruções que podem ser realizadas mecanicamente, tal como um programa de computador. Um procedimento que termina é chamado de algoritmo.

Um procedimento pode ser usado para reconhecer linguagens. Como isto é feito? Dada uma linguagem, se existir um procedimento que ao processar uma palavra responda “sim” se a palavra estiver na linguagem, dizemos que o procedimento reconhece a linguagem.

Uma linguagem que é reconhecida ou gerada por um procedimento chama-se linguagem recursivamente enumerável. As máquinas de Turing são aquelas que reconhecem e geram tais linguagens; pensa-se que elas devam ser os mecanismos mais sofisticados que permitam capturar plenamente o conceito de computabilidade. Vamos estudar a seguir a sua constituição, funcionamento e limitações.





## 9. O MAIS PODEROSO DE TODOS OS COMPUTADORES

*Software VisualTuring. Projetos para a construção de computadores*

**Resumo:** Como topo de nossa hierarquia de máquinas será apresentada a máquina de Turing através de duas de suas implementações concretas: utilizando-se bobinas de papel e o *software* VisualTuring. Com elas pretendemos iniciar a arte da programação, apresentando exemplos simples de como realizar operações matemáticas elementares, bem como os aspectos de geração e reconhecimento de linguagens dadas por procedimentos.

### MÁQUINAS DE TURING

Uma máquina de Turing é constituída por uma fita infinita dividida em quadrados. Cada quadrado ou está vazio ou contém um símbolo. A máquina pode ler, escrever e apagar símbolos. Em geral os símbolos são denotados por letras do alfabeto ou por números.

A máquina de Turing tem um programa, que nada mais é do que um conjunto de regras. Estas regras têm a seguinte forma: Se você está em tal e tal estado e lê tal e tal símbolo, então escreva tal e tal símbolo, mova para um quadrado em tal e tal direção e mude para tal e tal estado. Assim, toda regra fica determinada por 5 informações: estado atual, símbolo atual, símbolo a escrever, direção do movimento e novo estado.

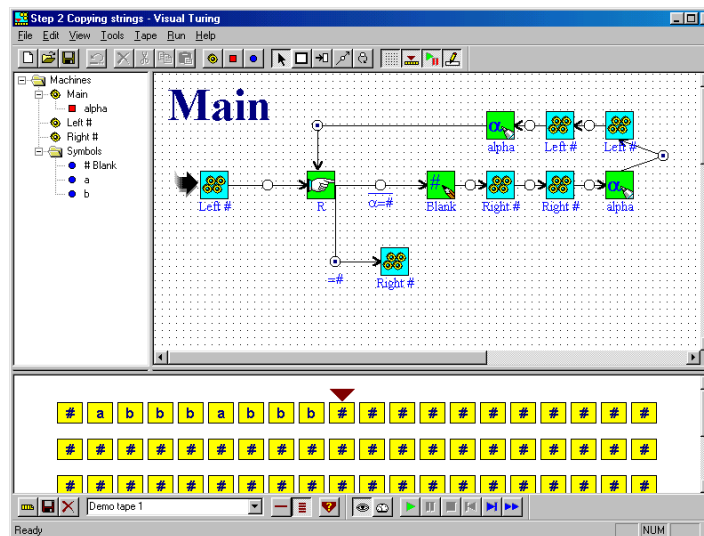
Uma implementação de uma máquina de Turing, feita com lápis e papel, será descrita a seguir (de acordo com o artigo de Newton da Costa em 28/11/1993 publicado na Folha de São Paulo).

Apesar da aparente simplicidade, estas máquinas podem, ao menos teoricamente, realizar qualquer tarefa efetuada por um computador moderno.

Muitos acreditam que elas possam descrever e simular o cérebro humano (vide artigo da Folha de São Paulo referido acima). Isto pode ser difícil de acreditar à primeira vista, mas uma olhada nos exemplos mostra que podemos realizar operações absolutamente não triviais com as máquinas de Turing. Vale a pena trabalhar com as implementações e tentar.

O software VisualTuring será de grande valia para esta tarefa. Com ele poderemos explorar muitos exemplos afim de entender a dinâmica das máquinas de Turing. Utilize o computador e mãos à obra!

### VisualTuring em funcionamento

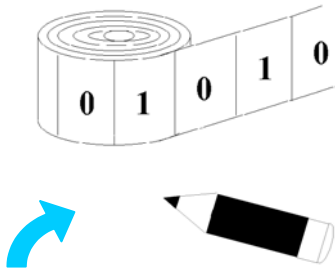


# MÁQUINA DE TURING

## UM SUPERCOMPUTADOR COM PAPEL E LÁPIS

A máquina de Turing é uma máquina hipotética, muito mais geral que os computadores que conhecemos. Ela foi imaginada para simular o funcionamento do cérebro resolvendo problemas de Matemática.

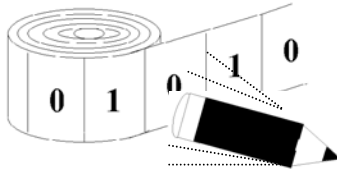
Todo computador é baseado em uma máquina de Turing. É constituída por uma fita de comprimento infinito, dividida em quadrados. Essa fita tem três funções, comparáveis a de um computador: “entrada” de dados para a “memória” e uma espécie de “programa”.



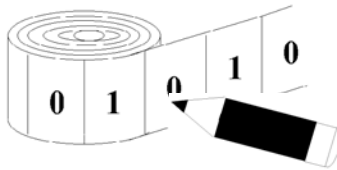
Além da fita, a máquina tem um lápis especial, capaz de ler, apagar e escrever.

A cada instante, o “lápis” lê a fita e, com base no que foi lido, a máquina pode:

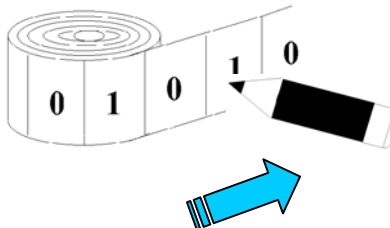
- 1** Apagar o que está escrito.



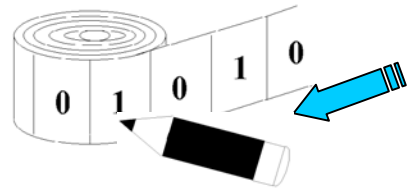
- 2** Escrever por cima, isto é, mudar o programa.



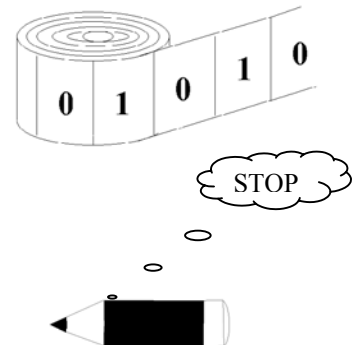
- 3** Mover o lápis para a direita para ler a outra casa.



- 4** Mover o lápis para a esquerda para ler a outra casa.



- 5** Mudar seu estado interno, parando ou não.



Veja o funcionamento dinâmico da Máquina de Turing no software VisualTuring.

## NOTAÇÕES PARA AS MÁQUINAS DE TURING

As máquinas de Turing nada mais são do que programas que instruem como trabalhar na fita. Para sermos organizados, podemos descrever uma dada máquina de três formas diferentes: através de uma tabela, de um grafo direcionado ou de uma coleção de quádruplas.

**Exemplo:** Máquina escreve três 1's na fita e pára.

Para exemplificar, suponhamos que a máquina possua três estados  $q_1$ ,  $q_2$ ,  $q_3$  e apenas dois símbolos 0: (que representa a casa vazia na fita) e 1 (representando a marca que faremos na fita). As letras E e D representarão o movimento para a esquerda ou para a direita respectivamente da cabeça de leitura/escrita da máquina.

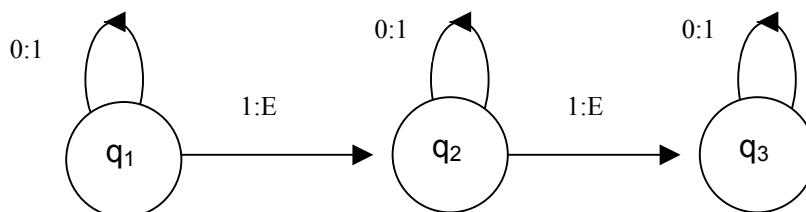
O funcionamento desta máquina é muito simples: a partir de uma fita vazia ela, no seu primeiro estado, escreve um número 1 e mantém-se neste estado. Ao analisar o símbolo 1 impresso, ela move-se para a esquerda e passa para o seu segundo estado, encontra uma casa vazia e escreve 1 nesta nova casa. Ainda no segundo estado, analisando o símbolo 1, ela move-se para a esquerda e passa para o terceiro estado. Encontrando a casa vazia ela escreve o número 1 e pára, pois não existem mais instruções a seguir. Desta maneira ela “carimba” três 1's consecutivos na fita, da direita para a esquerda.

Costumeiramente, usa-se uma das três descrições a seguir para o funcionamento da máquina:

TABELA:

	Símbolo analisado		
		0	1
Estado presente	$q_1$	1 $q_1$	E $q_2$
	$q_2$	1 $q_2$	E $q_3$
	$q_3$	1 $q_3$	

GRAFO DIRECIONADO:

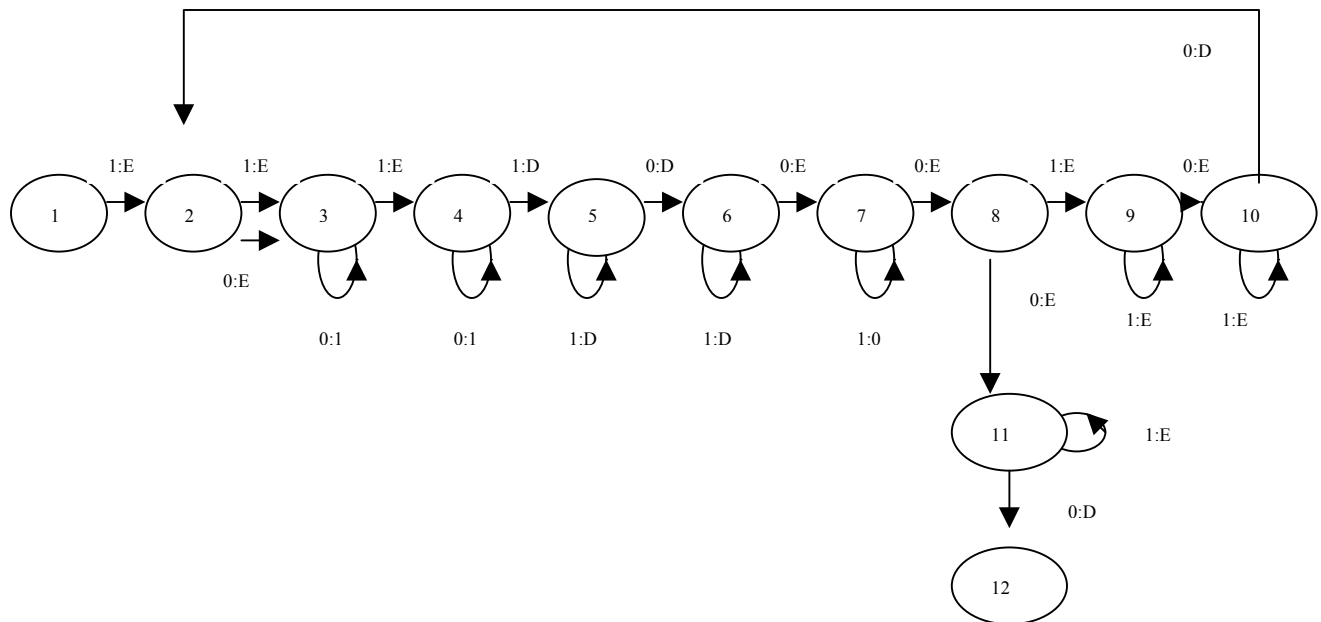


## COLEÇÃO DE QUÁDRUPLAS:

$q_1 0 1 q_1, q_1 1 E q_2, q_2 0 1 q_2, q_2 1 E q_3, q_1 0 1 q_3$

**Exemplo:** Máquina que dobra o número de 1's.

Esta máquina consegue dobrar o número de 1's originalmente impresso na fita. Para isto ela usa 11 estados e um décimo segundo estado que a faz parar.



É fácil ver que esta máquina duplica o número de 1's inicialmente marcados na fita. Vamos exemplificar, aplicando a máquina ao dado 11. No final, deveremos obter 1111. Em cada estágio, na linha superior temos o que se encontra escrito na fita e na linha inferior a posição da cabeça de leitura e o estado corrente em que a máquina se encontra:

11 1	011 2	0011 3	1011 3	01011 4	11011 4	11011 5	11011 5
11011 6	11011 6	110110 6	11011 7	11010 7	11010 8	11010 9	1 1 010 10
1 101 10	0 1101 10						

Do estado 10, retornamos ao estado 2 e começamos todo o processo novamente.

1101 2	01101 3	11101 3	011101 4	111101 5	111101 5	111101 5
111101 5	111101 6	1111010 6	111101 7	111100 7	111100 8	111 1 11
11 1 1 11	1 1 11 11	1 111 11	0 1111 11	1 111 12		

Estado de parada

Resultado da duplicação

Bastam 11 estados para duplicar qualquer número dado, o décimo segundo estado somente coloca a cabeça de leitura na primeira posição do número duplicado e indica que a máquina deve parar pois não tem mais instruções a seguir. Veja o funcionamento dinâmico desta máquina no *software* VisualTuring.

## PROJETOS PARA A CONSTRUÇÃO DE COMPUTADORES

Poderemos construir um computador com os conhecimentos adquiridos no Ensino Médio? É possível estruturar um projeto para, a partir de considerações teóricas apenas, construir uma máquina real que possa, mesmo em um contexto rudimentar, realizar operações tais quais um computador realiza?

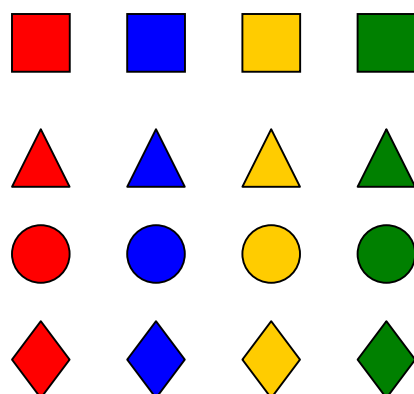
No texto a seguir indicamos alguns caminhos para se trabalhar nesta direção.

### PROGRAMANDO COM CORES E FORMAS:

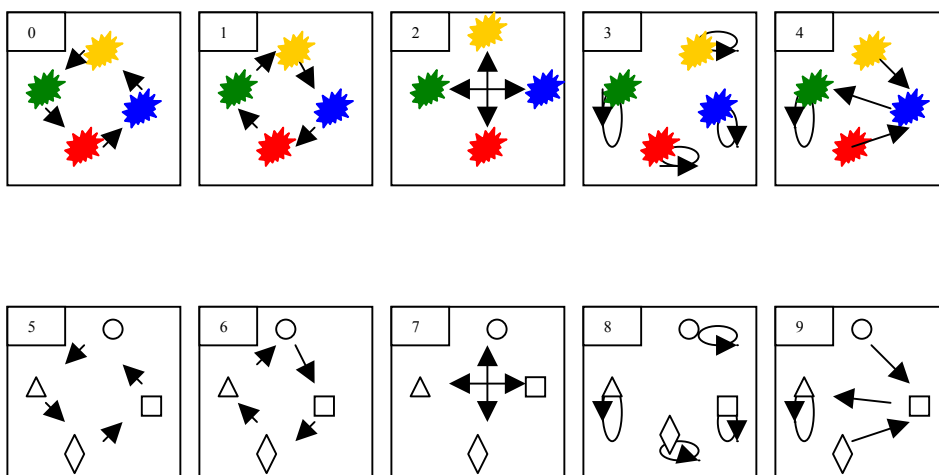
Este primeiro tipo de construção apareceu no livro *Pensamento Lógico e Tecnologia* (vide bibliografia). Para ilustrar as técnicas de programação que usaremos a seguir, vamos trabalhar com um pequeno tabuleiro, como o desenhado abaixo:

	1	2
1		
2	X	

Este tabuleiro pode ser pensado como uma matriz com quatro entradas ou endereços: 11, 12, 21 e 22. O primeiro número em cada par indica a linha e o segundo a coluna. Na figura acima o endereço assinalado com um X é 21. Vamos precisar também de 4 conjuntos de fichas de formas e cores diferentes, como os apresentados a seguir:




Estas pequenas fichas serão colocadas sobre a cartela e, seguindo uma série de instruções ganharão uma dinâmica que ilustram de modo bastante simples o funcionamento lógico dos computadores. As instruções estão dadas pelas 10 fichas abaixo:




As cinco primeiras transformações operam somente sobre a cor e as 5 últimas somente sobre a forma.

Vejamos como funciona a programação utilizando a cartela, as fichas e as instruções:

Exemplo: Suponha que inicialmente temos o seguinte dado inicial (*input*):





	1	2
1		
2		

(círculo verde no endereço 12). A transformação número 1 dada acima, transforma a cor verde em amarela, sem alterar sua forma. O resultado desta transformação é:

	1	2
1		
2		

Esta operação será denotado por 112, convencionando que o primeiro dígito (1) é a transformação que devemos aplicar e os últimos dois dígitos (12) é o endereço cujo conteúdo sofrerá a transformação.





Exemplo: A partir da configuração inicial:

	1	2
1		
2		





aplicar as instruções 711 e 721, isto é aplicar a transformação 7 na figura que está no endereço 11 e a seguir novamente a transformação 7 na figura que está



em 21. A instrução 7 altera as formas das figuras, transformando um círculo em um losango, um losango em um círculo, um triângulo em um quadrado e um quadrado em um triângulo. O resultado de 711 é





	1	2
1		
2		

E, compondo com a instrução 721, o resultado é:

	1	2
1		
2		

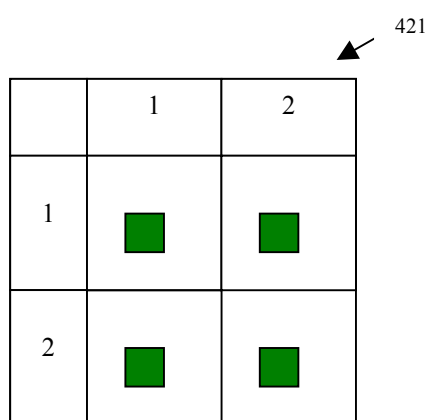
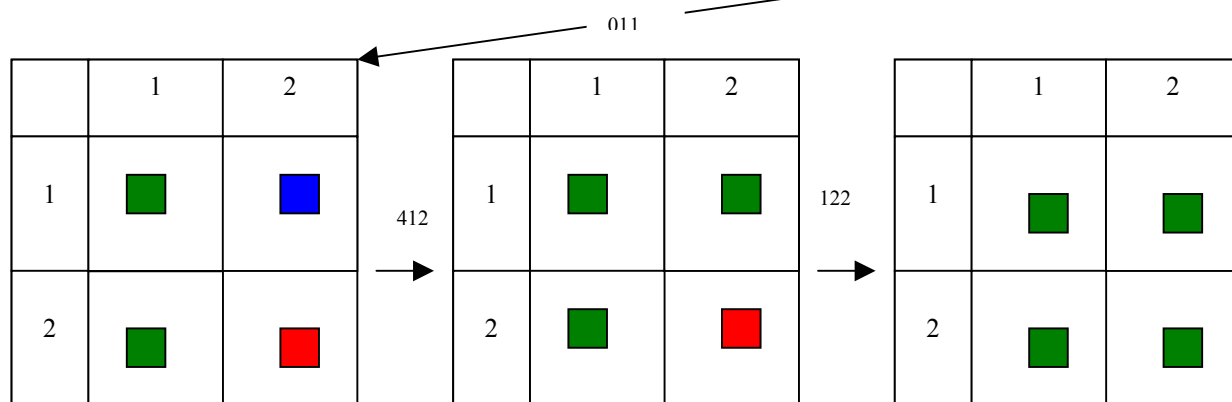
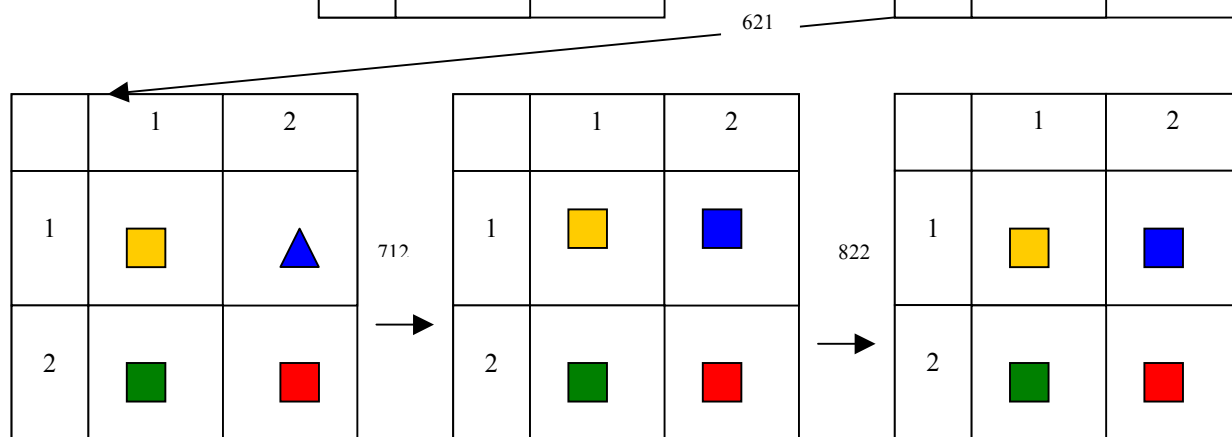
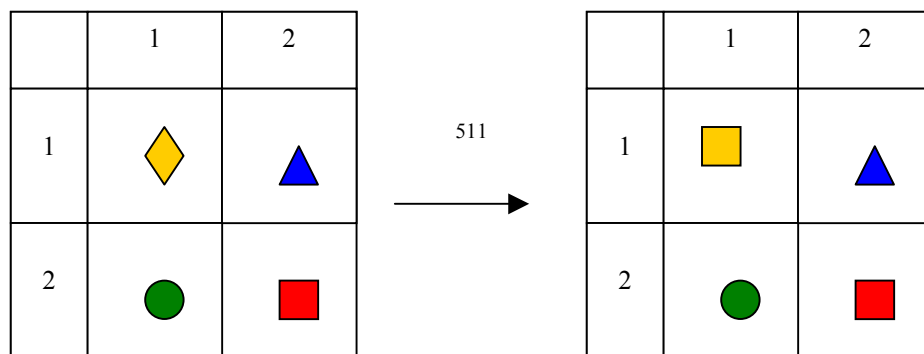
Vejamos mais um exemplo:

A partir da configuração inicial:

	1	2
1		
2		

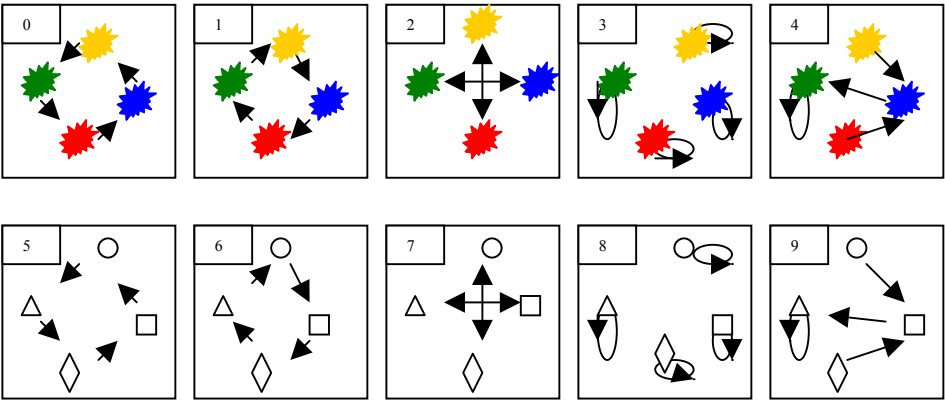
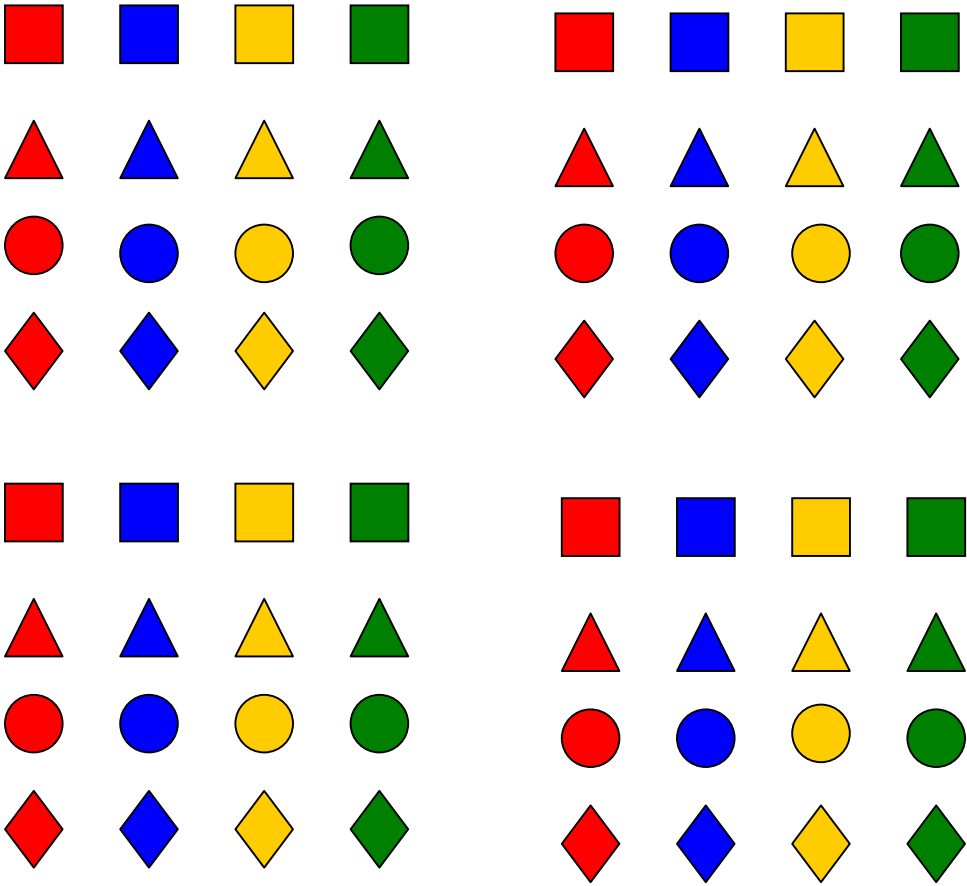
execute o “programa”:

1. 511
2. 621
3. 712
4. 822
5. 011
6. 412
7. 122
8. 421



Para mais exemplos e detalhes consulte P.L. T. – Pensamento Lógico e Tecnologia.

	1	2
1		
2		

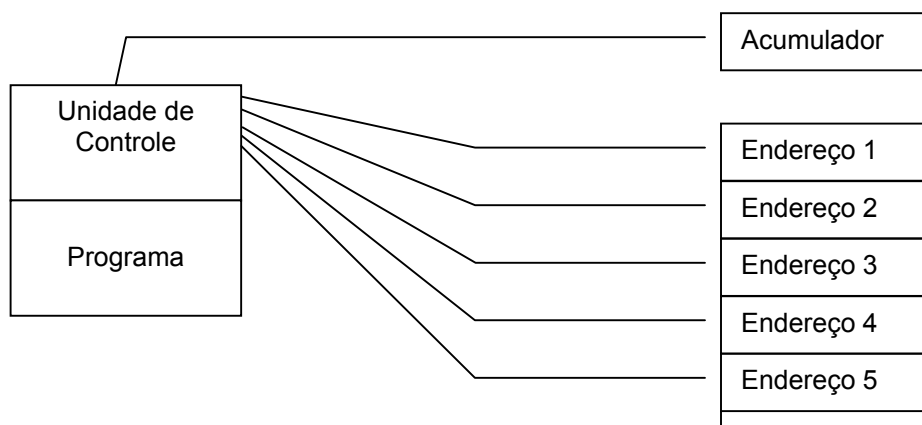




## MÁQUINAS COM ACESSO ALEATÓRIO - RAM

Acredita-se que as máquinas de Turing conseguem computar tudo o que pode ser computável (Tese de Church), entretanto elas são muito lentas pois necessitam acessar dados e informações de forma serial. Existe uma outra máquina que funciona com os mesmos princípios da máquina de Turing e que se assemelha aos computadores reais com os quais trabalhamos hoje em dia, sendo mais rápidos no processamento que as máquinas de Turing pois possuem acesso aleatório aos endereços de sua memória. Seu nome é máquina RAM.

Um modelo desta máquina encontra-se esboçado na figura abaixo:



Esta máquina se parece com uma rede telefônica que é controlada a partir de uma unidade central que tem acesso instantâneo a uma lista infinita de endereços e a um registro especial chamado de acumulador (uma espécie de depósito onde os dados são guardados temporariamente).

Para operar com esta máquina, devemos aprender como devemos estabelecer a comunicação com ela. Uma primeira tentativa é descrever uma série de comandos que a máquina entende e obedece (uma espécie de linguagem ASSEMBLER para os mais íntimos). Os comando básicos são os seguintes:

Nome resumido para se lembrar	Argumento	Significado
CDA	x	<b>Carregue Diretamente o Acumulador com o conteúdo do endereço X</b>
CIA	x	<b>Carregue Indiretamente o Acumulador com o conteúdo do endereço X</b>
EDA	x	<b>Estoque Diretamente o conteúdo do Acumulador no endereço X</b>

EIA	x	<b>Esto</b> que <b>I</b> ndiretamente o conteúdo do <b>A</b> cumulador no endereço X
ADA	x	<b>AD</b> icione o conteúdo do endereço X ao conteúdo do <b>A</b> cumulador
SUA	x	<b>SU</b> btraia o conteúdo do endereço X do conteúdo do <b>A</b> cumulador
PUL	x	<b>PUL</b> e para a instrução rotulada por X
PUZ	x	<b>PUL</b> e para a instrução rotulada por X se o acumulador contiver <b>Z</b> ero
PAR		<b>PAR</b> e

No acumulador e em cada endereço da memória, podemos estocar um número natural, não importa o quão grande ele seja. Vejamos o que cada um dos termos empregados acima significa:

- O conteúdo do endereço X é obviamente o inteiro que se encontra estocado no endereço X.
- Carregar indiretamente o conteúdo do endereço X no acumulador significa ir até o endereço citado, observar o seu conteúdo, ir imediatamente ao novo endereço dado pelo conteúdo observado e transferir o conteúdo deste novo endereço para o acumulador. Por exemplo, digamos que inicialmente no endereço 2 encontra-se o inteiro 5 e que no endereço 5 encontra-se estocado o inteiro 21, então o comando CIA(2) ordena o seguinte: vá até o endereço 2, veja qual é o inteiro estocado aí (é o 5), a seguir vá até o endereço 5, tome o inteiro estocado aí (21) e carregue-o no acumulador.
- O comando de pulo PUL(X) significa que a próxima instrução a ser executada é a instrução de número X.
- PUZ(X) significa que a próxima instrução a ser executada é a instrução de número X, desde que o conteúdo da memória seja zero. Se não for zero, continuamos a seguir as instruções na ordem natural em que elas aparecem.
- A máquina pode parar de dois modos: quando encontra o comando PAR ou quando uma instrução não executável aparecer (por exemplo quando se manda pular para a instrução 40 em um programa de apenas 30 instruções).

Esta máquina tem o mesmo poder computacional que a máquina de Turing, mas é mais fácil de programar, devido ao acesso aleatório da memória.

As linguagens que falam diretamente para a máquina o que elas devem fazer são conhecidas como linguagens ASSEMBLER ou linguagens de baixo nível ou ainda linguagens da máquina. Faremos duas abordagens deste tipo de linguagem de programação: a primeira usa os comandos descritos acima para detectar se dentro de uma dada lista de números naturais, existe pelo menos um par de

números repetidos e a segunda exemplifica o funcionamento de um “computador real”, pois já é realizado com um microprocessador.

## UM PROGRAMA QUE PROCURA E ACHA REPLICANTES:

Dada uma lista de números naturais, existe uma maneira simples de detectar se a lista contém um par de números repetidos: basta compará-los dois a dois e verificar tal fato. Existe ainda uma outra maneira mais rápida e simples de fazer isto; ela consiste em utilizar uma régua tão grande quanto o maior de todos os números dados e em colocar nessa régua, um a um, uma marca na posição correspondente ao número. Se na régua marcarmos a mesma posição duas vezes, a lista terá números repetidos e um dos números repetidos será o que for marcado duas vezes. Simples, não? Esta pequena tarefa pode ser descrita resumidamente da seguinte maneira (A descreve a seqüência de números dada e B a régua que será marcada com a marca 1 a cada leitura de um número de A):

### PROGRAMINHA:

**Para i variando de 1 até n**

**Se  $B(A(i)) \neq 0$**

**Então coloque A(i) como saída**

**Sair**

**Caso contrário, B(A(i)) recebe o valor 1**

Infelizmente nossa máquina RAM é rudimentar demais para entender um programa descrito desta maneira. Não podemos, neste contexto elementar, fazer uso de um compilador que transforme o nosso Programinha em uma série de instruções compreensíveis para o nosso computador. Precisamos escrever um programa mais rústico que seja entendido e possa ser executado pela máquina. Ele será apresentado a seguir. A máquina RAM pode ser pensada como uma lista de caixas cada uma guardando um certo número.

0		Acumulador
1	1	constante
2	0	Resposta
3	6	Índice i
4	9	Limite de A
5	0	Índice j
6	3	Primeiro número de A
7	4	Segundo número de A
8	2	Terceiro número
9	2	Quarto número de A
10	0	Primeiro índice de B
11	0	Segundo Índice de B
12	0	Terceiro índice de B
13	0	Quarto índice de B
14	0	Quinto índice de B

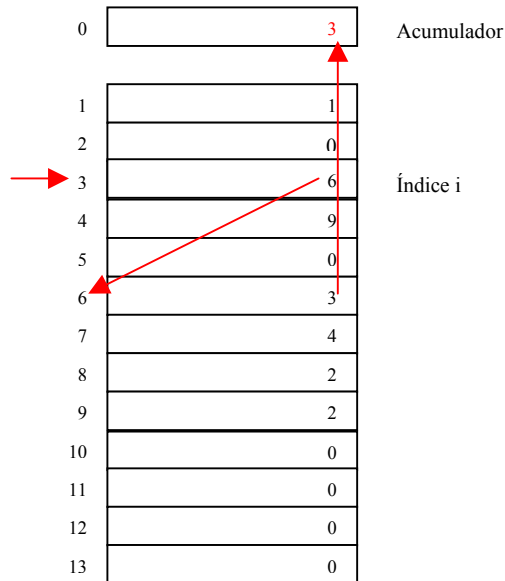
Como exemplo, inicialmente fornecemos a lista A formada pelos números 3, 4, 2 e 2. Queremos descobrir se nesta lista existe números repetidos (é claro que há). As primeiras posições da memória são reservadas para variáveis e constantes usadas pelo programa. As próximas quatro posições (6, 7, 8 e 9) contêm o conjunto A com os quatro inteiros que queremos saber se entre eles existem duplicatas ou não. Todas as outras posições da memória (do 10 para frente) formam a régua B que marcamos com o número 1 cada vez que o programa lê um novo número da lista A. Se encontrarmos duas destas marcas, então a lista A conterá números duplicados.

A primeira locação da memória contém a constante 1 que é usada para incrementar o índice i da lista A. O valor deste índice é estocado na posição 3. Um segundo índice j é estocado na locação 5; ele serve para acessar as componentes da lista B. O programa que analisaremos é o seguinte:

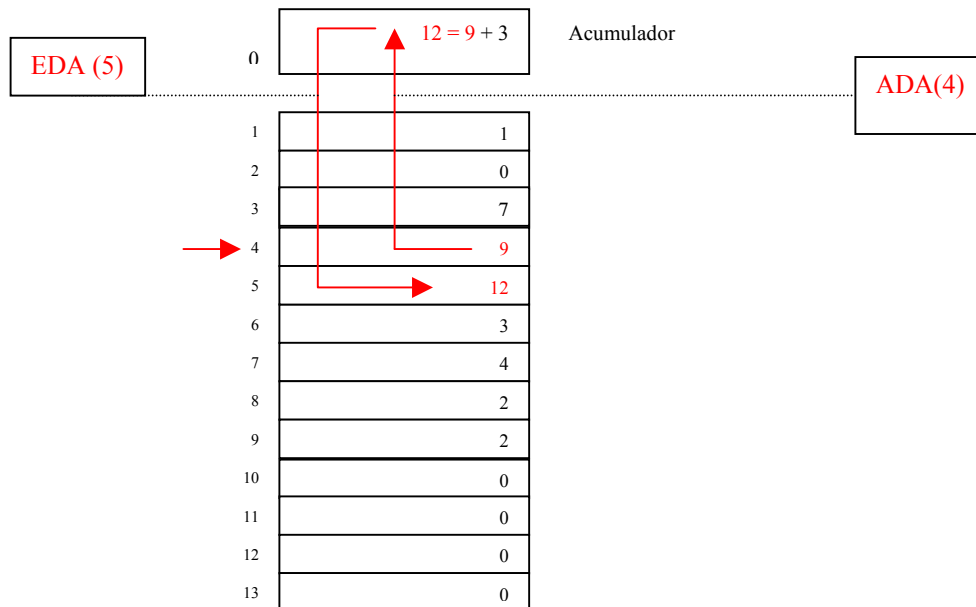
PROGRAMA RAM			
Linha	Comando	Variável	Comentário
1	CIA	3	Obtém a i-ésima entrada de A
2	ADA	4	Adiciona a quantidade para computar o índice j
3	EDA	5	Guarda no acumulador o índice j
4	CIA	5	Obtém a j-ésima entrada de B
5	PUZ	9	Se a entrada for 0 vai para a instrução 9
6	CDA	3	Se a entrada for 1, obtém o índice j
7	EDA	2	e estoca-o em 2
8	PAR		Pára a execução
9	CDA	1	Obtém a constante 1
10	EIA	5	e estoca-a em B
11	CDA	3	Obtém o índice j
12	SUA	4	Subtrai o limite
13	PUZ	8	Se i = limite, pára
14	CDA	3	Obtém o índice i (novamente)
15	ADA	1	Incrementa i
16	EDA	3	Estoca o novo valor i
17	PUL	1	Vai para a primeira instrução

A primeira instrução CIA(3) observa o conteúdo da locação 3, o número estocado neste endereço é 6 a seguir vai ao endereço 6, nele temos estocado o número 3 e é este 3 que é levado ao acumulador, pois a instrução CIA é indireta.





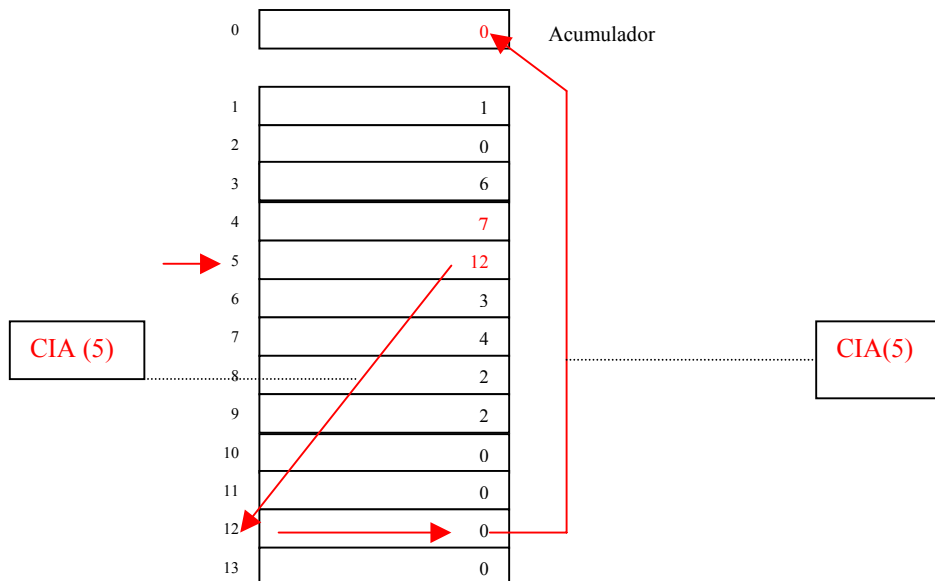
O segundo comando ADA(4) observa que na locação 4 está registrado o número 9. Este número delimita o tamanho do espaço da memória usado para se trabalhar com a lista A. O comando ADA adiciona 9 ao conteúdo do acumulador (que é 3), obtendo 12. A posição 12 é a terceira posição da lista B, pois B começa na posição 10 da memória. O terceiro comando EDA(5) leva o conteúdo do acumulador para o endereço de número 5.



Estes três primeiros comandos reconhecem que 3 é o primeiro inteiro da lista A e computa a localização da terceira posição de B (linha 12).

A quarta instrução do programa é um comando indireto - CIA (5), o qual verifica na quinta locação qual é o número registrado (12), vai até a posição 12

(terceira linha de B) e carrega o acumulador com o conteúdo encontrado aí (0). Sendo este número 0, a próxima instrução PUZ(9) pula para a instrução 9.



A instrução 9 (CDA(1)) carrega o acumulador com o conteúdo do endereço 1 (que é o número 1) e com o décimo comando (EIA(5)) o programa vai até a posição 5, lê o número ali registrado (12) prossegue até o endereço 12 e coloca neste local o conteúdo do acumulador (o número 1). Esta é a marca que queríamos fazer desde o começo: como o número 3 aparece em A, fazemos na posição 3 de B com uma pequena marca (1) para que no futuro possamos saber se um outro número 3 encontra-se na lista A. Se, mais tarde um outro número 3 aparecer na lista A, o acumulador deverá ter 1 registrado em seu conteúdo e o programa, quando encontrar o comando PUZ(9), deverá ir para a instrução 9, levar o valor corrente de i (na locação 3) para a locação 2 e finalmente parar.

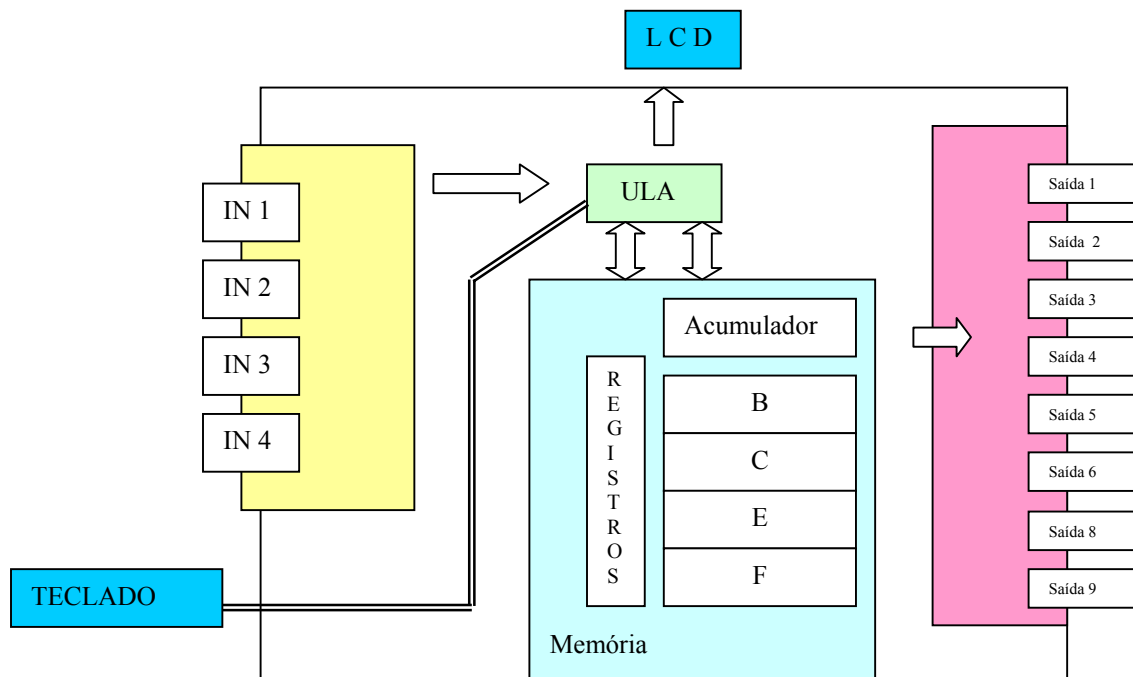
Deste modo a posição 2 da memória contém a resposta de nossa tarefa: se for 0, isto significa que nenhuma duplicata foi encontrada, caso contrário ele conterà a localização em A de um inteiro repetido.

O restante do programa (instruções de 11 até 16) faz o seguinte: recupera o índice i na localização 3, subtrai 9 (na localização 4) para decidir se o programa chegou ao fim de A. Senão, ele incrementa o índice i e estoca-o na posição 3 e volta ao início do programa para testar o próximo inteiro da sequência A.

## PROGRAMANDO EM ASSEMBLER NO MK-904

Um pequeno computador que pode ser programado em ASSEMBLER e que ilustra o funcionamento dos modernos computadores pode ser encontrado na Maleta de Experiências da MINIPA – Laboratório Eletrônico. O microprocessador empregado consiste de três seções: um acumulador, registradores, memória,

unidade lógica aritmética (ULA) e portas de entrada/saída (E/S), conforme o seguinte esquema:

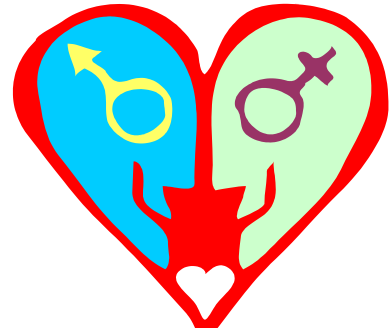


Instruções de Entrada/Saída	Transfere os dados de uma porta de entrada ou de saída para o acumulador	IN, OUT
Instruções de mover	Move os valores de dados de um registrador para outro	MOV
Instruções lógicas	Executa operações lógicas entre o acumulador e um registrador, entre o acumulador e um dado ou entre um registrador e um dado	AND, OR, XOR, NOT
Instruções aritméticas	Executa uma adição ou subtração entre o acumulador e um registrador ou entre um registrador e um dado	ADD, SUB
Instruções de deslocamento	Desloca o conteúdo do acumulador ou registrador para a direita ou esquerda	ROR, ROL
Instruções de incremento/decremento	Incrementa/decrementa 1 ao conteúdo do acumulador o registrador	INC, DEC
Instruções de saltar	Movimenta o fluxo do programa para o endereço do operando	JMP, JZ, JNZ, JC, JNC
Instrução de comparação	Compara o conteúdo do acumulador ou registrador com dados	CMP

Instruções sonoras	Inicia/termina a emissão de <i>beep</i>	BOM, BOF
Instruções de controle de tempo	Aguarda um período de tempo pelo valor do operando	TM1, TM2
Outras instruções	Interrompe o programa, aciona o “vai um”, define o formato de dados, etc.	STP, NOP, SEC, CLC, HEX

Com estas instruções é possível programar (usando diretamente a linguagem da máquina) o microprocessador, fazendo com que ele execute as tarefas que desejamos. O trabalho com a maleta de experiências propicia um bom entendimento do funcionamento de programas computacionais simples ligados à Inteligência Artificial e à Robótica.

Para fins didáticos, existem também descrições pormenorizadas para a construção de um Processador de Auxílio ao Ensino, elaboradas por Furuya em sua dissertação de Mestrado (ICMC-USP, 1984). Neste protótipo, o relógio (*clock*) acoplado a outras unidades do computador, desempenha um papel fundamental. A construção dos elementos básicos do computador, tais como circuitos-soma, contadores, etc. usando circuitos elétricos elementares também pode ser encontrada no texto Pensamento Lógico e Tecnologia (vide bibliografia).



## 10. A VIDA SEXUAL DAS MÁQUINAS

*Auto-reprodução. O software Winlife.*

**Resumo:** Será abordada a questão da reprodução de máquinas. Uma máquina ou um programa de computador pode gerar uma cópia perfeita de si mesmo? A resposta é afirmativa e serão apresentados dois exemplos ilustrativos de como isto pode ocorrer: o software Winlife que simula o Jogo da Vida inventado pelo matemático John Conway e experimentos de auto-reprodução no Excel. Serão descritos brevemente os avanços históricos nesta área a partir dos trabalhos pioneiros de Von Neumann sobre autômatos celulares que possuem o poder da auto-reprodução.

### MÁQUINAS QUE SE AUTO-REPRODUZEM

Em nosso estudo sobre máquinas projetadas para a realização de tarefas intelectuais, temos estabelecido um elo entre os trabalhos realizados por nossa mente e as tentativas de criação de inteligências artificiais. Nossas máquinas têm sido bastante simples, mas com elas já visualizamos possibilidade de:

- ✓ Entender como as máquinas aprendem e guardam informações (APRENDIZAGEM)
- ✓ Compreender como as máquinas podem entender linguagens (COMUNICAÇÃO)

Dando prosseguimento ao estudo do paralelismo entre inteligência humana x inteligência artificial, vamos colocar nossa atenção sobre a questão da REPRODUÇÃO e de como isto pode ser feito por máquinas.

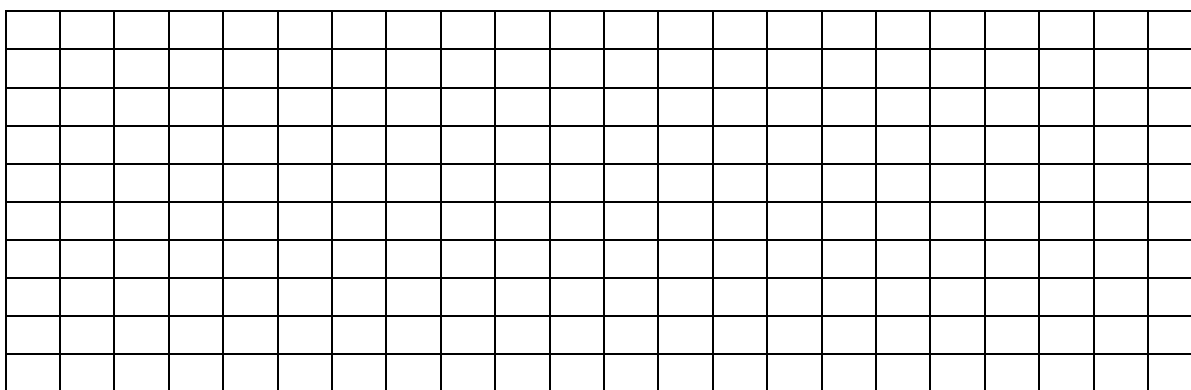
Com a origem dos computadores e a possibilidade, ao menos em tese, de se criar vida totalmente artificial, muitos pesquisadores de renome preocuparam-se com esta questão. Dentre eles destacamos John von Neumann e E. Codd. Algumas de suas idéias serão livremente abordadas aqui, através da introdução de um jogo chamado Jogo da Vida, que explicita alguns dos princípios desta importante área científica. Lembramos que o intuito é manter sempre os estudos dentro da esfera pedagógica do contexto do ensino médio, tentando aplicá-los como instrumento motivador da formação científica dos alunos.

Salientamos ainda que o estudo da aprendizagem, comunicação e reprodução não esgotam de modo algum a gama de enfoques necessários para o entendimento da vida artificial; nenhuma questão ligada à auto-consciência ou à metafísica foi ainda abordada; nossas máquinas atuais e mesmo as de um futuro próximo ainda estão muito distantes de realizarem as tarefas que podem ser realizadas por nossas mentes.

### O JOGO DA VIDA

Este jogo ilustra como uma população inicial de indivíduos, num contexto artificial e seguindo rigorosas normas de sobrevivência, pode evoluir, perpetuar-se ou extinguir-se. Ele é um exemplo de um autômato celular.

O ambiente onde vivem os indivíduos pode ser pensado como um quadriculado infinito no plano.

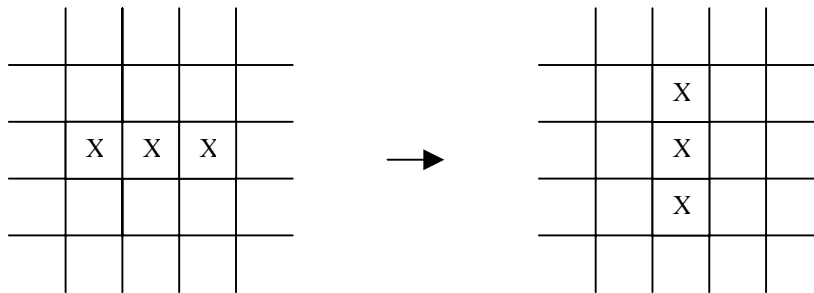


Uma célula pode assumir dois estados: “viva” ou “morta”. Células mortas devem ficar vazias e células vivas devem ser marcadas, com um X , com uma pedra ou com uma cor. Cada célula pode ter 8 vizinhos: 2 na horizontal, 2 na vertical e 4 nas diagonais.

Uma população inicial deve ser introduzida no tabuleiro. A cada instante de tempo, ou seja a cada geração, esta população inicial se modifica, com base no estado presente, mas também levando em conta os seus vizinhos (verticais, horizontais e diagonais). As regras de sobrevivência são as seguintes:

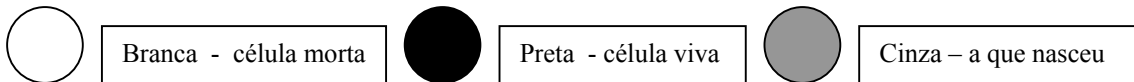
- 1) Se a célula estiver viva no instante  $t$ , ela permanecerá viva no instante seguinte  $t+1$ , desde que tenha 2 vizinhos (senão ela morre de solidão) e não mais do que 3 (se tiver 4 vizinhos ou mais ela morrerá devido à superpopulação).
- 2) Se a célula estiver morta no tempo  $t$ , ela continuará morta até que tenha exatamente 3 vizinhos vivos (um casal e um padre para fazer o casamento).

Exemplo:



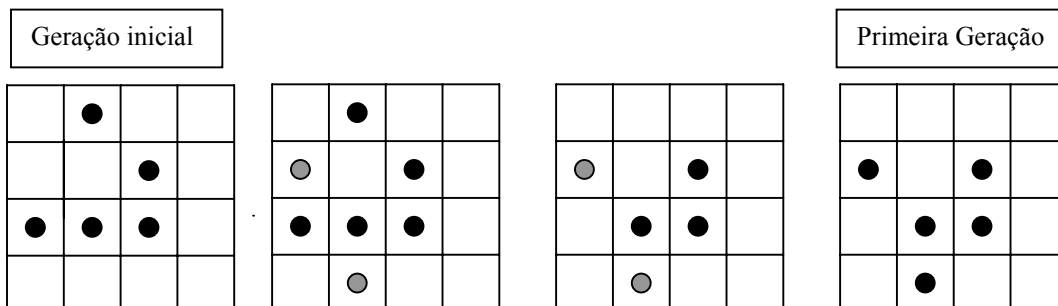
Observe: o X do meio continua vivo pois tem dois vizinhos vivos; entretanto os dois X's das pontas morrem porque só têm um vizinho vivo. Nas células acima e abaixo do X do meio, que estão vazias na primeira figura, nascem dois novos X's pois eles têm exatamente 3 vizinhos.

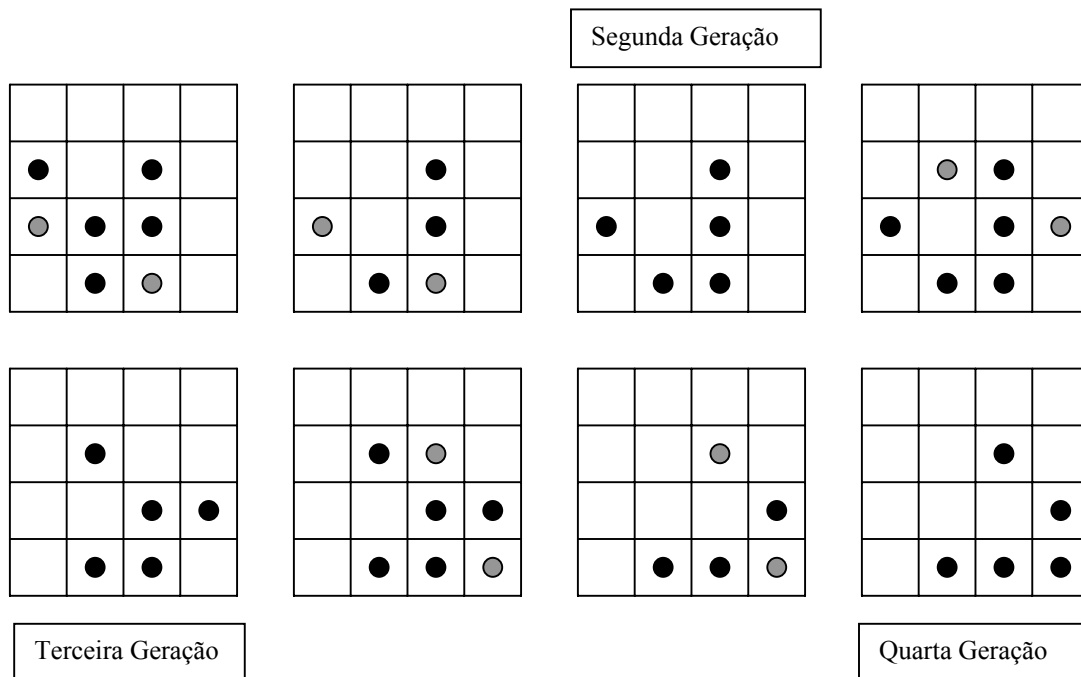
Para que o jogo fique organizado, podemos usar 3 tipos de fichas:



Primeiro colocamos a população inicial usando fichas pretas, a seguir colocamos as fichas cinzas nas células que nasceram, então removemos as células mortas e finalmente trocamos as fichas cinzas por pretas e iniciamos novamente todo o processo. Observe como este trabalho se assemelha com um programa de uma máquina de Turing.

Analisemos agora a seguinte evolução populacional:





Observe que a população inicial repete-se na última ilustração, mudando apenas de posição. Isto nos remete à questão da reprodução de máquinas: não seria possível, a partir de uma certa população inicial, obter duas cópias idênticas da mesma população? A partir desta duplicata poderíamos obter 4 cópias e assim sucessivamente chegando a um crescimento populacional que muito se assemelha à reprodução de certas formas microscópicas de vida?

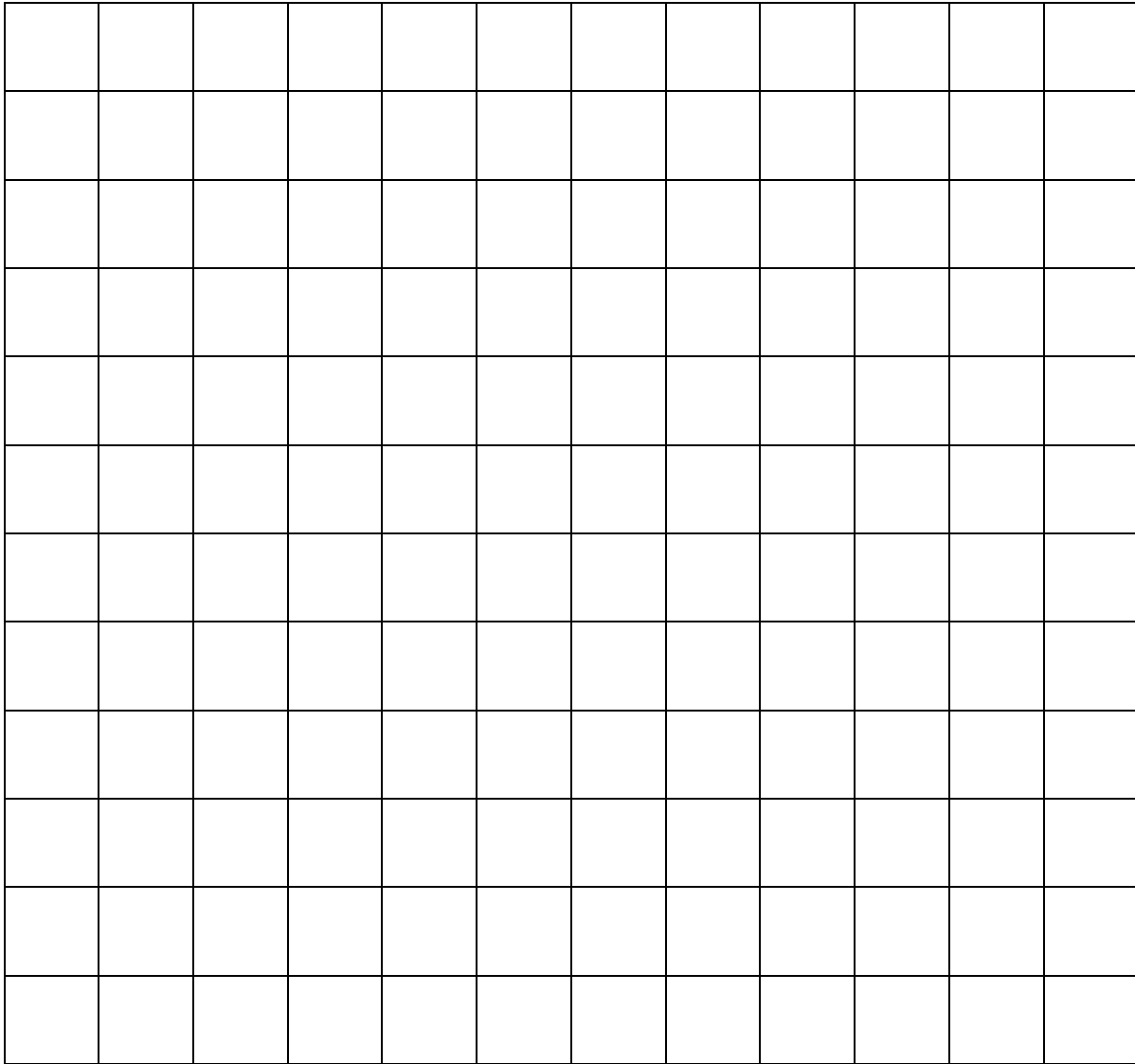
Por volta de 1950, o cientista John von Neumann construiu um modelo teórico de uma máquina que se auto-reproduzia. Suas idéias foram inspiradas no estudo do DNA de células vivas e a máquina arquitetada era um autômato celular com 29 estados que utilizava 200.000 células para se auto-reproduzir. Depois desta descoberta, muitos autômatos celulares auto-replicantes foram construídos teoricamente. Um dos trabalhos mais conhecidos nesta área deve-se a E. Codd que em meados dos anos 60 construiu uma máquina auto-reprodutiva com apenas 8 estados.

Existem vários simuladores do Jogo da Vida que podem ser implementados em computador. Destacamos dois: o WinLife, com muitos exemplos curiosos, e um pequeno programa desenvolvido no Excel, com cenas explícitas de sexo feito por uma máquina (mas sem cenas de sexo explícito). Eles mostram de maneira simples como podemos entender a auto-reprodução de máquinas. Acesse-os e divirta-se!



Foi noticiado recentemente que uma empresa japonesa construiu um robô que podia fazer uma cópia de si mesmo, desde que fosse colocado à sua frente as peças para a construção, como baterias, peças, fios, etc. Será isto uma ameaça à espécie humana? Bem, aí já é ficção científica e deixamos a cargo de sua imaginação.

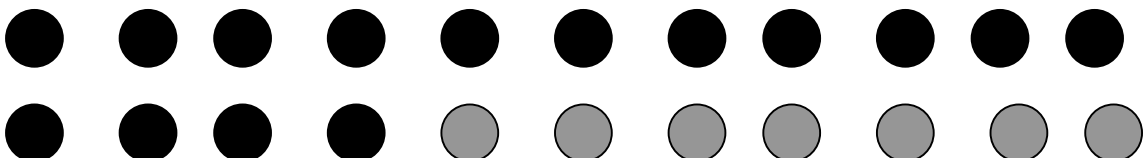




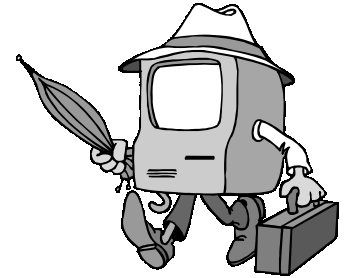
## O JOGO DA VIDA

Uma população inicial deve ser introduzida no tabuleiro. A cada instante de tempo, ou seja a cada geração, esta população inicial se modifica, com base no estado presente, mas também levando em conta os seus vizinhos (verticais, horizontais e diagonais). As regras de sobrevivência são as seguintes:

- 3) Se a célula estiver viva no instante  $t$ , ela permanecerá viva no instante seguinte  $t+1$ , desde que tenha 2 vizinhos (senão ela morre de solidão) e não mais do que 3 (se tiver 4 vizinhos ou mais ela morrerá devido à superpopulação).
- 4) Se a célula estiver morta no tempo  $t$ , ela continuará morta até que tenha exatamente 3 vizinhos vivos (um casal e um padre para fazer o casamento).







## 11. O QUE UMA MÁQUINA FAZ, MAS NÃO DEVERIA FAZER

*Complexidade computacional. O problema P versus NP.*

**Resumo:** Muitas tarefas que os computadores podem teoricamente realizar na prática não podem ser efetivamente levados a cabo, devido muitas vezes ao alto tempo de processamento ou ao uso de muita memória. Isto nos leva ao estudo da complexidade computacional, onde o problema  $P \neq NP$  desempenha um papel fundamental. Trata-se de um problema de simples compreensão, mas possivelmente de resolução muito difícil, ainda não encontrada. Ele foi incluído como um desafio para os cientistas do século XXI e sua solução está entre os “problemas de um milhão de dólares” ofertados pelo Clay Mathematics Institute. Será apresentado um texto sobre complexidade computacional com uma discussão sobre os problemas polinomiais e não determinísticos polinomiais. Serão indicados também endereços de páginas da Internet para que os alunos possam conhecer importantes problemas de nosso tempo.

### COMPLEXIDADE COMPUTACIONAL

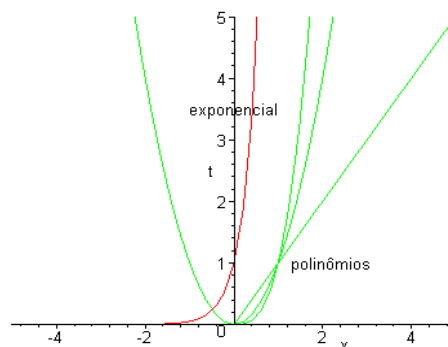
A Complexidade Computacional é um ramo da Matemática Computacional que estuda a eficiência dos algoritmos. Do ponto de vista prático, de nada adianta um algoritmo perfeito se sua implementação computacional demora uma centena de anos para ser processada. Mesmo tarefas relativamente simples, como o produto de dois números com muitos dígitos, pode demorar alguns minutos para serem concluídas em alguns computadores. Se considerarmos que alguns algoritmos necessitam multiplicar números muito grandes milhares de vezes esses alguns minutos podem se transformar em um tempo excessivamente longo.

Para medir a eficiência de um algoritmo frequentemente usamos o tempo teórico que o programa leva para encontrar uma resposta em função dos dados de

entrada. Este cálculo é feito associando-se uma unidade de tempo para cada operação básica que o procedimento executa. Se a dependência do tempo com relação aos dados de entrada for polinomial, o programa é considerado rápido. Se, entretanto, a dependência do tempo for exponencial o programa é considerado lento. Podemos também nos perguntar sobre os programas que estão entre estas duas classes.

Dado um polinômio  $p(x)$  sabemos que  $|p(x)|$  cresce para infinito com  $|x|$ . Esse crescimento, entretanto, é lento quando comparado com o crescimento de uma função exponencial  $a^x$ , onde  $a > 1$ . Este fato pode ser expresso pela fórmula

$$\lim_{x \rightarrow \infty} \frac{a^x}{|p(x)|} = \infty$$



## P versus NP

**Definição:** A classe de algoritmos P é formada pelos procedimentos para os quais existe um polinômio  $p(n)$  que limita o número de passos do processamento se este for iniciado com uma entrada de tamanho  $n$ .

O algoritmo da eliminação de Gauss, ou método do escalonamento, usado para resolver sistemas lineares, é um procedimento da classe P. De fato, para resolver um sistema linear  $n \times n$  são necessárias  $(4n^3 + 9n^2 - 7n)/6$  operações aritméticas, um custo aproximado de  $n^3$ . Por outro lado, o método de Cramer, também utilizado para resolver sistemas lineares, tem custo exponencial. Para um sistema linear  $n \times n$ , esse método dispense aproximadamente  $(n+1)!$  multiplicações. Para se ter uma idéia deste custo, se usarmos um computador que realiza 100 milhões de multiplicações por segundo, para resolver um sistema  $20 \times 20$ , seriam necessários 32.000 anos.

Os algoritmos NP não se referem a procedimentos não polinomiais (na verdade isto é uma conjectura, um problema que ainda não sabemos a resposta). A leitura correta para procedimentos NP é dizer que se referem a algoritmos "não-determinísticos polinomiais" no tempo. A classe NP será definida logo abaixo; antes vamos fazer uma breve abordagem histórica de suas origens.

No início dos anos sessenta do século passado foram encontrados muitos algoritmos que resistiam a uma simplificação polinomial, isto é, algoritmos que não admitiam procedimentos análogos na classe P. Nesta época Steve Cook (veja nas referências alguns pontos de partida para aprofundar os conhecimentos nesta

área) observou um fato simples e ao mesmo tempo surpreendente: se um problema pudesse ser resolvido em tempo polinomial, poderíamos também verificar se uma dada possível solução é correta em tempo polinomial (dizemos que o algoritmo pode ser certificado em tempo polinomial).

Um exemplo simples de como esta certificação pode ser feita é o problema de descobrir se um dado número é composto, ou seja, se ele não é primo. Suponha que queiramos descobrir se 4294967297 é um número composto. Não existe uma maneira eficiente (rápida) de fazer isto. De fato tal tarefa pode ser realizada pela utilização do crivo de Eratóstenes, testando os possíveis divisores do número, o que pode demandar um tempo excessivo de computação. Entretanto existe uma maneira sucinta de certificar que aquele número é composto: basta verificar que o produto de 6700417 por 641 é exatamente 4294967297. Assim, se pudermos achar uma certificação, podemos exibir efetivamente sua validade. Achá-la pode ser extremamente difícil, no entanto. A fatoração de 4294967297 foi encontrada por Leonard Euler em 1732, noventa e dois anos após Pierre de Fermat ter conjecturado erroneamente que tal número era primo.

Definição: A classe dos problemas NP é aquela para as quais podemos verificar, em tempo polinomial, se uma possível solução é correta.

Evidentemente  $P \subset NP$ . De fato, se um algoritmo pode ser executado em tempo polinomial e tivermos em mãos um possível candidato  $S$  à solução, é possível executar o programa, obter uma solução correta  $C$  e comparar  $C$  com  $S$  para certificar que  $S$  é de fato solução, tudo em tempo polinomial.

O problema P versus NP consiste na seguinte conjectura:

$$P = NP ?$$

A classe NP consiste dos algoritmos não-determinísticos polinomiais, e recebe este nome devido a uma formulação equivalente que não usa o conceito de certificação, mas o de decisão de linguagens. Infelizmente a demonstração desta equivalência é bastante técnica e foge dos objetivos destas notas (vide bibliografia). A categoria de linguagens a que nos referimos são as reconhecidas por computadores teóricos chamados de Máquinas de Turing.

Existe uma afirmação heurística, conhecida como Tese de Church, que diz que tudo o que pode ser programado através de um procedimento, pode também ser realizado por uma máquina de Turing.



Alan Mathison Turing (1912-1954) foi um brilhante matemático inglês, tendo trabalhado em várias áreas, como Lógica, Álgebra, Teoria dos Números, Computação, Inteligência Artificial, Morfogênese. Dedicou-se durante algum tempo à arquitetura de computadores. Em Lógica deu continuidade ao Teorema da Indecidibilidade de K. Gödel, demonstrando que não existe um algoritmo universal capaz de detectar proposições indecidíveis em um sistema axiomático. Para isso inventou e utilizou as máquinas de Turing. Durante a 2ª Guerra Mundial trabalhou para o governo britânico como decifrador de códigos, tendo desempenhado importante papel na criptoanálise do código alemão conhecido com "Enigma".

Existem máquinas de Turing determinísticas e não-determinísticas. As determinísticas são aquelas que quando estão em um certo estado, lendo um certo dado, podem se movimentar de um único modo rumo à próxima configuração; já as não-determinísticas podem se mover para diversas configurações, a partir do dado lido e da configuração interna corrente. Evidentemente as máquinas determinísticas formam uma subclasse das não-determinísticas.

Que ligação afinal existe entre máquinas de Turing e linguagens? Em termos breves, se  $L$  é uma linguagem sobre um alfabeto  $S$ , isto é, se  $L$  é um subconjunto finito de seqüências de letras de  $S$ , dizemos que uma máquina de Turing  $M$  aceita a linguagem  $L$  se para toda palavra construída com as letras de  $S$  colocada como entrada (*input*), após o processamento,  $M$  entra em um estado de aceitação (respondendo de algum modo sim) se a palavra pertencer à linguagem. A palavra é recusada por  $M$  se, após o processamento,  $M$  entrar num estado de rejeição (respondendo não de algum modo) ou se ela falhar em completar seu processo computacional. Neste contexto, o problema  $P$  versus  $NP$  assume a seguinte forma:

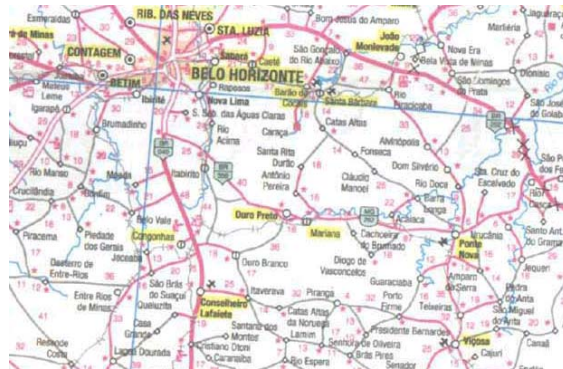
**É verdade que toda linguagem aceita em tempo polinomial por uma máquina de Turing não-determinística é também aceita, em tempo polinomial, por uma máquina determinística?**

Quais conseqüências a resolução positiva desta conjectura pode trazer? Destacamos apenas três: a existência de algoritmos úteis para uma série de problemas computacionais práticos nas indústrias; a destruição da segurança nas transações financeiras feitas eletronicamente; a quebra no sigilo de trocas de informações diplomáticas, Internet, etc.

A possível destruição dos códigos de segurança se deve ao fato de que a maioria dos algoritmos criptográficos são construídos sobre a hipótese da impossibilidade de uma criptoanálise em tempo polinomial.

Finalmente gostaríamos de destacar uma importante subclasse dos problemas NP, que são os problemas NP-completos. Leonid Levin e Steve Cook observaram que, dentre os problemas NP, existem alguns que são mais difíceis do que outros, no sentido de que, se pudermos resolver um desses problemas em tempo polinomial, então todos os problemas em NP também podem ser resolvidos em tempo polinomial. Assim a classe dos problemas NP-completos é o subconjunto dos "mais difíceis" problemas não-determinísticos polinomiais. Atualmente são conhecidos uma quantidade enorme de problemas NP-completos, mas o mais famoso deles talvez seja o **Problema do Caixeiro Viajante**, também designado por TSP (Traveling Salesman Problem):

Um viajante necessita visitar  $n$  cidades. As distâncias entre estas cidades são conhecidas. Começando inicialmente em uma cidade  $C_i$ , ele visita todas as outras cidades e retorna a  $C_i$ . Suponhamos que o orçamento disponível permita ao viajante se deslocar até uma distância  $k$ . Existe uma rota que passe por todas as cidades e tenha comprimento menor do que  $k$ ?



A solução deste problema, por ser ele NP-completo, pode levar à destruição de quase todos os sistemas de segurança eletrônicos do mundo!







## 12. O QUE AS MÁQUINAS NUNCA FARÃO

*Impossibilidades lógicas. O jogo axiomático.*

**Resumo:** Serão apresentadas algumas tarefas que nenhuma máquina consegue realizar, ou seja será tratada a questão da incomputabilidade. Será dada especial atenção ao problema da parada e, em um nível intuitivo, estabelecida sua relação com a decidibilidade de sistemas lógicos, mostrando que humanos devem afinal vencer as máquinas!

Embora existam tarefas em que os computadores substituem o homem com vantagens, existem outras inerentes à natureza humana que somente nós podemos realizar. Uma máquina pode ajudar o homem a demonstrar e a descobrir novos teoremas, mas não pode por si só demonstrar teoremas. O famoso teorema das quatro cores foi demonstrado com a ajuda de uma máquina. Este teorema afirma que, somente com 4 cores podemos colorir qualquer mapa, sem que países vizinhos tenham a mesma cor.

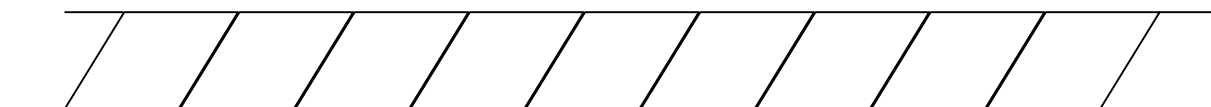
Se o computador pôde demonstrar este fato, não seria concebível tentar construir uma máquina que demonstrasse todos os teoremas, ou pelo menos todos os teoremas de uma dada teoria, tal como a Teoria dos Números?

A resposta é não. Vamos apresentar as razões deste fato em três abordagens. A primeira refere-se a uma máquina feita com carimbos, chama de máquina DIN-DIN. Ela exhibe, de modo intuitivo, a diferença entre o que é verdadeiro e o que é demonstrável. Os segundo e terceiro experimentos usam as máquinas de Turing descritas anteriormente e estabelece, em nível teórico, a apresentação de tarefas impossíveis de serem realizadas por máquinas computacionais tais como as conhecemos hoje em dia.

## A MÁQUINA DIN-DIN

É uma máquina muito simples, feita com uma fita de papel dividida em pequenos quadrados e quatro carimbos. Os carimbos são usados para imprimir na fita de papel um dos quatro símbolos: **D**, **I**, **N**, e **-**. Esta máquina serve para mostrar que as máquinas não podem tudo; ela exemplifica que é impossível construir uma máquina que demonstre todas as afirmações verdadeiras de uma dada teoria. Vejamos como é o seu funcionamento:

Usamos os carimbos para marcar as casas de uma fita de papel:



Podemos inicialmente produzir combinações aleatórias dos símbolos D, I, N, e -, mas só nos interessam certas combinações especiais destes símbolos, chamadas de sentenças. As sentenças serão combinações de símbolos dos seguinte tipos:

I - X ou ID - X ou NI - X ou NID - X

sendo que no lugar de X podemos colocar qualquer combinação dos quatro símbolos D, I, N, - que desejarmos. Este será o nosso sistema formal. Observe que X não faz parte do sistema formal, ele é apenas uma “meta-variável” utilizada na linguagem comum para designar uma seqüência de símbolos formada por D, I, N e -.

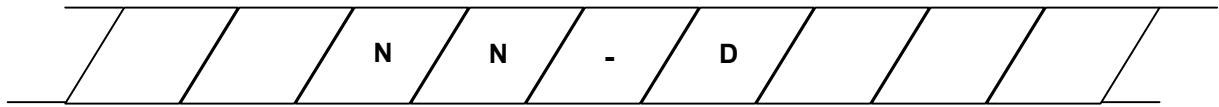
Vamos apresentar um modelo para o nosso sistema formal, conferindo um valor-verdade (verdadeiro ou falso) para as sentenças de nosso sistema formal, através das seguintes regras:

1. I significa “imprimível”
2. D significa “dobro”
3. N significa “não”
4. ND significa “dobro não”.

### Sentenças da linguagem:

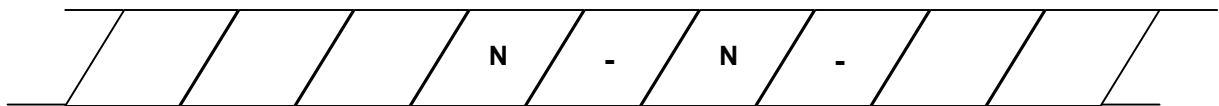
1. I – X significa que a seqüência de símbolos X aparece impressa na fita. Assim, I – X é verdadeira no caso em que X aparece impressa na fita e somente neste caso.

Observe a fita:



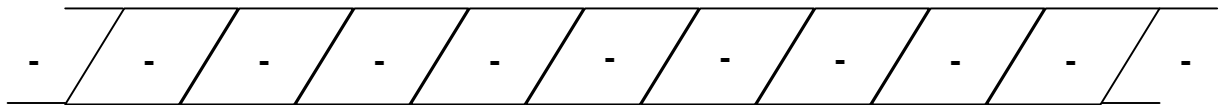
Neste caso dizemos que a sentença  $I - NN-D$  é verdadeira, pois de fato  $NN-D$  aparece impresso na fita.

2.  $ID - X$  significa que a seqüência de símbolos  $XX$  aparece impressa na fita. A sentença  $ID - X$  é verdadeira no caso, e somente no caso, da seqüência  $X$  aparecer escrita em dobro na fita.



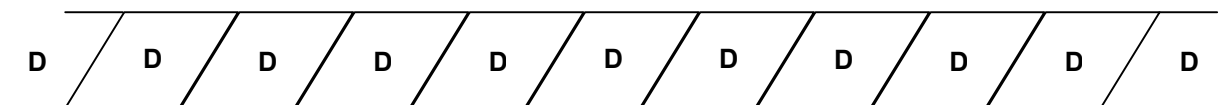
Observando a fita acima, podemos dizer que  $ID - N -$  é uma sentença verdadeira, pois  $N -$  aparece impressa duas vezes na fita.

3.  $NI - X$  significa que a seqüência de símbolos  $X$  nunca aparece impressa na fita. A sentença  $NI - X$  é verdadeira no caso, e somente no caso, de a seqüência  $X$  nunca aparecer impressa na fita.



Esta sentença é mais difícil de verificar, pois devemos percorrer a fita toda tentando encontrar a seqüência  $X$  impressa em algum lugar. Entretanto, se perguntarmos a alguém que carimbou os símbolos na fita se ele utilizou ou não a seqüência  $x$  em sua tarefa, podemos decidir isto sem a tarefa de vasculhar toda a fita. Na fita acima, todas as casas estão marcadas com o símbolo  $-$  e assim podemos dizer que  $NI - D$  por exemplo é verdadeira.

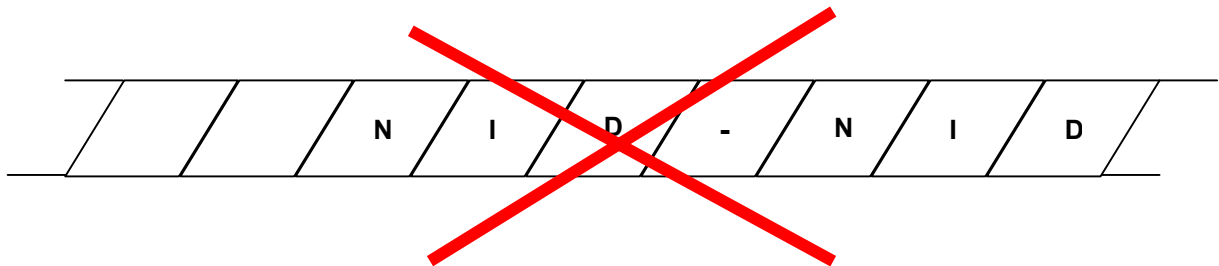
3.  $NID - X$  significa que a seqüência de símbolos  $XX$  nunca aparece impressa na fita. A sentença  $NID - X$  é verdadeira no caso, e somente no caso, da seqüência  $XX$  nunca aparecer impressa dobrada na fita.



Na fita acima, todas as casas estão marcadas com o símbolo D e assim podemos dizer que NID – N, por exemplo, é verdadeira. Já NID – D é falsa pois DD aparece impressa na fita.

**INDAGAÇÃO CRUCIAL:** A máquina DIN-DIN imprime todas as sentenças verdadeiras? (ou num nível mais abstrato: podemos fabricar uma máquina que responda efetivamente se uma sentença verdadeira é um teorema?)

A resposta é não. Para isto, suponhamos por hipótese que a máquina DIN-DIN não imprime sentenças falsas e consideremos a sentença NID-NID. Esta sentença é verdadeira ou não? O seu significado diz que o dobro da seqüência NID nunca aparece impressa dobrada na fita, isto é, a sentença NID-NID é verdadeira no caso e somente no caso dela mesma nunca aparecer impressa na fita.



Temos duas possibilidades:

- a) NID-NID é verdadeira. Então ela nunca aparecerá impressa na fita e portanto a máquina não consegue imprimir todas as sentenças verdadeiras.
- b) NID-NID é falsa. Neste caso é falso que ela não aparece impressa na fita, isto é, ela deve aparecer impressa na fita em algum lugar, mas então a máquina imprimiria sentenças falsas, o que foi descartado por hipótese.

Assim, inventar uma máquina que demonstre todas as verdades é uma tarefa sem esperança de se concretizar. Veja nas referências os trabalhos de Carnielli.

## O PROBLEMA DO CASTOR OCUPADO

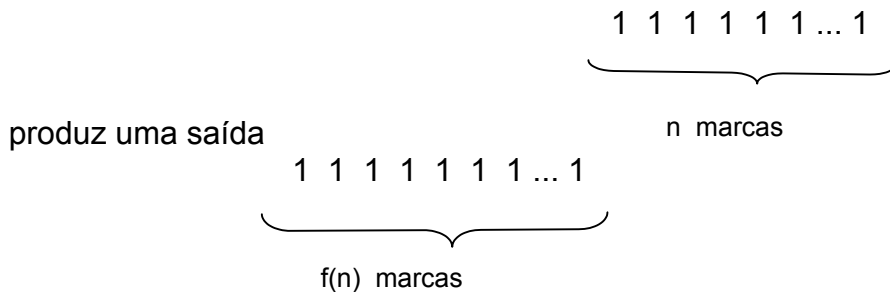


Os castores são animais conhecidos por seu trabalho de construir barreiras em rios, carregando varetas das margens, uma a uma em cada viagem. As máquinas de Turing nos lembram o trabalho dos castores, quando percorrem a fita, apagando, transportando e escrevendo 1's.

Vamos pensar sobre o problema de quão ocupada uma máquina de Turing pode ser.

Algumas máquinas de Turing nunca param e são de certo modo as mais ocupadas, mas não nos preocuparemos com elas. Dentre as que de fato param, existem algumas mais produtivas que outras. Queremos descobrir dentre estas qual é a mais produtiva de todas.

Uma máquina de Turing pode ser pensadas como uma função que a cada entrada de dados



As máquinas de Turing podem também produzir uma sequência de 1's a partir da fita vazia. O número de marcas que a máquina naturalmente produz, iniciando com sua fita inteiramente vazia, chama-se produtividade máquina. Deste modo, podemos definir

$p(n)$  = a produtividade da mais produtiva das máquinas de Turing com  $n$  estados.

Pensando no trabalho de um castor, o que queremos é “encontrar o castor mais trabalhador” que saiba realizar  $n$  tarefas básicas simples.

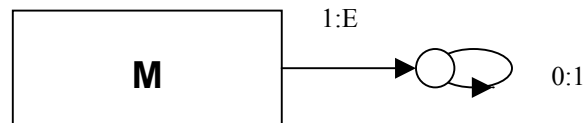
A função  $p(n)$  é bem definida que tem uma propriedade bastante interessante: não existe nenhuma máquina que possa computar todos os seus valores! Mais uma tarefa que os computadores nunca realizarão.

Vejamos uma justificativa intuitiva para tal fato:

- $p(1) = 1$ , pois a máquina que só tem um estado, aquele que escreve uma marca 1 na fita e pára, é a mais produtiva
- existe uma máquina com 11 estados que dobra o número de marcas em uma fita. Esta máquina foi apresentada anteriormente no capítulo que descreve as máquinas de Turing.
- existe uma máquina com 47 estados que produz cem números uns consecutivos e portanto  $p(47) \geq 100$ . Eis a justificativa deste fato: basta escrever 25 marcas na fita vazia (isto requer 25 estados) depois dobrar a quantidade de marcas, obtendo 50 marcas (usando a máquina anterior com 11

estados) e novamente dobrando as 50 marcas obtém-se as desejadas 100 marcas (usamos mais 11 estados). No total teremos:  $25 + 11 + 11 = 47$  estados.

- $p(n+1) > p(n)$ . De fato, tomemos  $M$  uma máquina de Turing que tenha  $n$  estados e seja a mais produtiva de todas as máquinas com  $n$  estados. Acoplamos a ela mais um estado como na figura:



Esta máquina produz  $p(n)+1$  números 1's consecutivos e tem  $n+1$  estados. Logo a mais produtiva de todas as máquinas com  $n+1$  estados deve gerar uma quantidade ainda maior de 1's, ou seja  $p(n+1) > p(n)$ . Logo  $n \rightarrow p(n)$  é uma função estritamente crescente.

- $p(n+11) \geq 2n$ . Para verificar isto, basta acoplar duas máquinas: uma primeira que escreve  $n$  1's na fita (gasta  $n$  estados) e outra que dobra o número de 1's originalmente escritos (gasta 11 estados). O resultado deste acoplamento é a produção de  $2n$  números 1's consecutivos e portanto a mais produtiva de todas as máquinas com  $n+11$  estados deve superar esta marca, isto é,  $p(n+11) \geq 2n$ .

### O Problema do Castor Ocupado é insolúvel via computadores:

Suponhamos que existisse uma máquina, que chamaremos de  $C$ , que computasse os valores da função  $p(n)$ , para todos os valores de  $n$ . Admitamos que a máquina  $C$  possua  $k$  estados. Veremos que isto nos levará a uma contradição.

Primeiramente, notemos que existe uma máquina com  $n+2k$  estados que produz  $p(p(n))$  números 1's consecutivos na fita: basta acoplar uma máquina que produz  $n$  1's (isto gasta  $n$  estados) e duas máquinas  $C$  (o que gasta  $2k$  estados). Logo a mais produtiva de todas as máquinas com  $n+2k$  estados deve superar esta marca de 1's, ou seja

$$p(n+2k) \geq p(p(n))$$

e como  $p$  é uma função crescente

$$n + 2k \geq p(n), \text{ qualquer que seja } n.$$

Com efeito, se  $n + 2k < p(n)$ , como  $p$  é estritamente crescente,  $p(n+2k) < p(p(n))$ , o que é uma contradição. Se trocarmos  $n$  por  $n+11$ , teremos

$$n + 11 + 2k \geq p(n + 11)$$

Mas, como vimos anteriormente,  $p(n+11) \geq 2n$  e assim

$$n + 11 + 2k \geq 2n \Rightarrow 11 + 2k \geq n, \text{ qualquer que seja } n$$

Substituindo  $n$  por  $12 + 2k$ , teremos  $11 + 2k \geq 12 + 2k$  e portanto  $11 \geq 12$ , o que não é verdade nem aqui nem na China! Logo, a função  $p(n)$  não pode ser computada por nenhuma máquina.

Uma curiosidade sobre a função  $p(n)$  é que ela cresce tão rápido que escapa ao poder computacional dos computadores (por exemplo, embora  $p(4)=13$ , já  $p(5) \geq 1915$  e  $p(6)$  é um número absurdamente grande ainda não conhecido).

## O PROBLEMA DA PARADA:

O problema da parada é um belo exemplo do que um computador não pode realizar, pois de certo modo ele deve decidir sobre o seu próprio funcionamento. É claro que a mente humana pode realizar tal tarefa sem se auto-destruir.

O problema da parada consiste inicialmente em colocar em uma fila infinita  $M_1, M_2, M_3, \dots$  todos os computadores que já foram ou que serão inventados. Estamos supondo que todos estes computadores são máquinas de Turing (isto é usando o que se conhece como a tese de Church). Quando alimentado com dados, cada um destes computadores pode parar, depois de processá-lo ou então entrar em um *loop* infinito, não parando nunca.

Gostaríamos de inventar um novo computador que respondesse a seguinte pergunta: Dada uma máquina  $M_i$  e um número  $n$ , a máquina  $M_i$  pára quando alimentada com o número  $n$ ? Mais precisamente, existe um computador  $P$  que, analisando a máquina  $M_i$  e o dado  $n$  consegue responder:

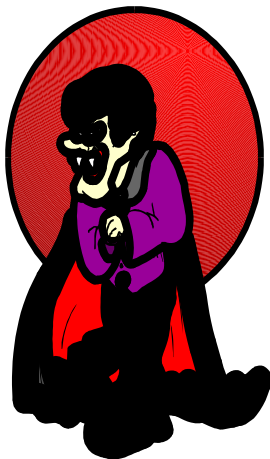
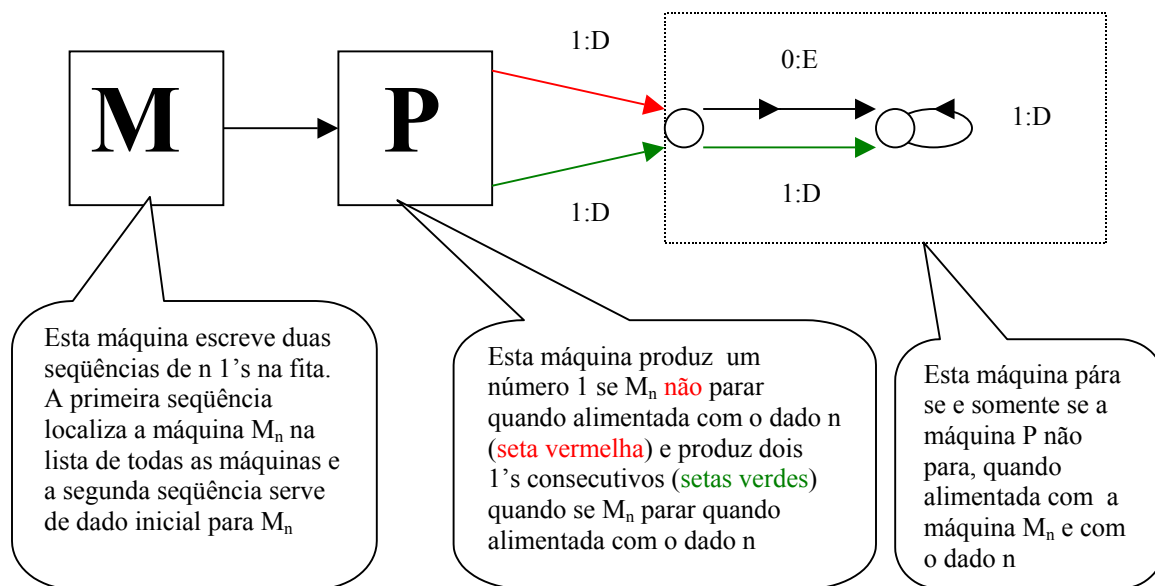
$P(M_i, n) = 1$  (dois uns seguidos) se a máquina  $M_i$  parar quando alimentada inicialmente com o dado  $n$

$P(M_i, n) = 0$  (um único 0) se a máquina  $M_i$  não parar quando alimentada inicialmente com o dado  $n$ .

Vamos mostrar que tal computador  $P$  não pode existir. A demonstração deste fato será dada na forma intuitiva explorando-se a auto-referência.

Suponhamos, por absurdo, que o computador P existisse. Todo número  $n$  pode ser escrito como uma seqüência consecutiva de  $n$  1's. É fácil construir uma máquina  $M$  que dobra o número de 1's: o primeiro destes  $n$  1's será usado para localizarmos a máquina pelo seu número na lista de todas as máquinas que fizemos acima e o segundo para servir de dado de entrada para essa mesma máquina.

Vamos acoplar três máquinas:  $M$ ,  $P$  e uma terceira máquina que pára quando alimentado por um único 1 e nunca pára quando alimentada por dois números 1's consecutivos. Esta nova máquina com os três acoplamentos tem o seguinte aspecto:



Vamos chamar esta nova máquina, formada pelo acoplamento das três máquinas acima de  $M_k$ . Esta é uma nova máquina com uma propriedade muito estranha: ela pára se e somente se a máquina  $M_n$  não pára quando alimentada com o dado  $n$  (isto é, com  $n$  1's consecutivos). Mas, como  $n$  é um número natural qualquer, podemos fazer o vampiro se alimentar do próprio sangue, tomando-se  $n = k$ . O que acontece então? Se  $M_k$  pára então ela mesma não para e se ela não pára, então ela pára. Uma tremenda contradição. Assim, a máquina  $P$  não pode existir o problema da parada é insolúvel.



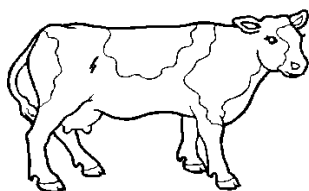
O problema da parada permite mostrar que certos sistemas lógicos são indecidíveis, resultando assim em importante ferramenta de estudo para a Lógica Matemática.

## IMPOSSIBILIDADES MATEMÁTICAS

Existem tarefas que são impossíveis de serem realizadas, tanto por homens, como por máquinas, pois elas trazem dificuldades intrínsecas que não podem ser superadas pelo método da dedução. A Matemática entretanto é, por excelência, a Ciência que pode cuidar destas causas impossíveis. Veremos dois exemplos que ilustram como demonstrar impossibilidades.

### O JOGO AXIOMÁTICO:

Uma teoria matemática formalizada é um exemplo de sistema axiomático. O que isto significa? O método axiomático teve origem nos trabalhos dos gregos antigos e é baseado em quatro componentes fundamentais: os termos e relações primitivas que devem ser admitidos sem uma definição explícita, os axiomas que são sentenças a respeito dos termos e relações primitivas (elas devem ser aceitas como os pontos iniciais da teoria) e finalmente os teoremas que são consequências lógicas dos axiomas, utilizando-se regras previamente estabelecidas. Vamos ilustrar como funciona um sistema axiomático, através de um jogo: o jogo do **MU**, que é uma adaptação das idéias do livro Gödel, Escher e Bach, escrito por Douglas Hofstadter.



Nosso jogo começa com apenas três letras do alfabeto: **M**, **I** e **U**. Eles serão nossos símbolos iniciais. Só poderemos formar palavras com estas três letras e seguindo regras rígidas que serão apresentadas a seguir. Antes porém, devemos partir de uma palavra inicial. Esta palavra será **MI**. Este será nosso único axioma. A partir dele e usando estritamente as regras que serão apresentadas, será possível formar novas palavras, seguindo o processo de produção já descrito no capítulo 8, a respeito de linguagens.

O jogo consiste em se procurar produzir a palavra **MU**, como um teorema, ou seja como resultado da aplicação das regras sucessivas vezes, começando-se com o axioma **MI**. Será possível realizar tal façanha?

As regras que podemos utilizar são as seguintes:

I. Se a última letra de uma palavra é **I**, podemos acrescentar **U** no final.

Ex: De **MI** podemos produzir **MIU**.

II. A partir de uma palavra do tipo **Mx**, em que **x** é uma seqüência qualquer das letras **M**, **I** e **U**, podemos produzir a palavra **Mxx**.

Ex: A partir de **MIU** podemos produzir **MIUIU**.

III. Substituir a ocorrência de **III** (três **I**'s consecutivos) por um único **U**.

Ex: A partir de **UMIIMU**, podemos formar **UMUMU**.

IV. Se ocorre **UU**, podemos simplesmente apagá-lo.

Ex: Iniciando-se com a palavra **MUUUIII** podemos produzir **MUIII**.

A sorte está lançada; a partir da palavra inicial **MI** e usando unicamente as quatro regras acima, podemos produzir a palavra **MU**? Encare esta tarefa como um quebra-cabeça e tente conseguir a produção de **MU**. Podemos utilizar algumas fichas de cartolina com as letras **M**, **I** e **U** e colocando na mesa a palavra inicial **MI**, ir agregando ou retirando novas letras, de acordo com a regra que estivermos empregando.

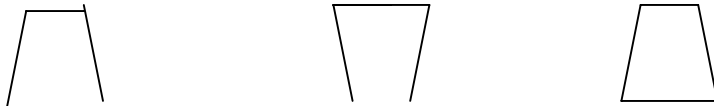
Se contarmos o número de letras **I**'s, veremos que ele nunca pode ser zero. Independentemente da regra aplicada, nunca poderemos chegar a não ter **I**'s nas palavras, se iniciarmos com o axioma **MI**. Na verdade nunca obteremos um número de **I**'s que seja um múltiplo de 3. De fato, as regras I e IV não alteram o número de **I**'s de uma palavra. A regra III diminui o número de **I**'s em três unidades e portanto ela não pode criar um número de **I**'s que seja um múltiplo de 3; ela só produz múltiplos de 3 **I**'s se originalmente já tivermos uma palavra constituída por um número de **I**'s que seja um múltiplo de 3. O mesmo acontece com a regra II, pois dobrando algo que não seja um múltiplo de 3, nunca conseguiremos um múltiplo de 3.

Como, começamos sempre com o axioma **MI** (que possui só uma letra **I**), as regras só produzem múltiplos de 3 **I**'s a partir de algo que já tenha um múltiplo de 3 **I**'s e 0 é um múltiplo de 3, chegamos à conclusão que nunca produziremos **MU**, dentro do sistema apresentado, mostrando assim a impossibilidade de se realizar tal tarefa.

## JOGANDO COM COPINHOS DE CAFÉ

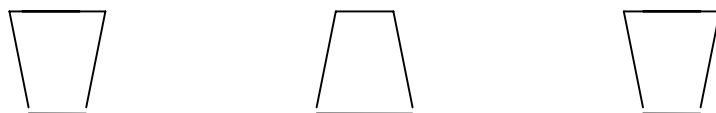
Questões que envolvem paridade servem muito freqüentemente para provar a impossibilidade de alguma tarefa. Para exemplificar um destes fenômenos, peça a alguém que resolva o seguinte desafio:

**DESAFIO** : Coloque três pequenos copos de café de acordo com a configuração abaixo:



Virando simultaneamente 2 desses copos, a pessoa deve deixá-los todos com a boca virada para cima. A solução é muito simples, basta virar o primeiro e o último dos copos. Peça à pessoa que dê uma segunda solução. Observe que isto pode ser feito virando-se por exemplo o primeiro e o segundo copos e a seguir o segundo juntamente com o terceiro.

Coloque agora os três copos na seguinte disposição:



Mais uma vez, virando-se simultaneamente 2 copos, solicite que a pessoa coloque todos os copos emborcados para cima. Depois de algumas tentativas, a tarefa vai se revelar impossível. Como sabemos disto? Associe ao copo com a boca para cima o número +1 e ao copo virado para baixo o valor -1. Da primeira vez que colocamos o desafio os copinhos tinham a configuração -1 +1 e -1.

Queremos que os três copos fiquem virados para cima, isto é obter a configuração +1 +1 e +1. Com devemos virar dois copos simultaneamente, isto é perfeitamente possível. Observe o que ocorre com o produto dos três números, antes e depois da virada:

$$(-1) \cdot (+1) \cdot (-1) = (+1) \cdot (+1) \cdot (+1) = 1$$

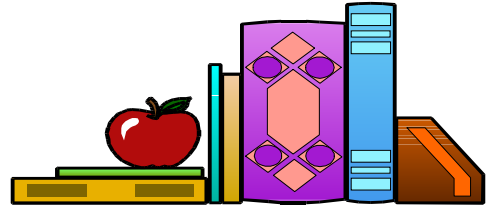
Agora, quando propomos pela segunda vez o desafio, a configuração inicial era +1 -1 e +1. O produto desses três números é -1. Este produto não se altera quando viramos simultaneamente dois dos copos pois se ambos tiverem o sinal +, depois da virada ficarão ambos com o sinal de - e se os sinais forem diferentes,

depois da virada mudarão ambos e ainda continuarão diferentes. Assim o problema é impossível.

Tente solucionar um problema parecido com mais copinhos. Por exemplo, veja a figura abaixo, virando-se de cada vez dois copos, podemos colocá-los todos com a boca para cima?



Qual é a estratégia para a solução?



### 13. REFERÊNCIAS BIBLIOGRÁFICAS OU NÃO

Adleman, L., Rivest, R. e Shamir, A *method for obtaining digital signature and public-Key cryptosystems*. Comm. ACM, 21(2), 1978.

Boolos, G. and Jeffrey, R. *Computability and Logic* , Cambridge University Press, 1974.

Carnielli, W e Rathjen – *Combinatória e Indemonstrabilidade ou o 13o. trabalho de Hércules*, Matemática Universitária 12, 1990, 23-41.

Cook, S., *The complexity of theorem-proving procedures*. New York, Proceedings of the 3rd Symposium on the Theory of Computing, ACM, 151-158.

Costa, N., *Máquina corre atrás do cérebro*. Folha de São Paulo, 28 de novembro de 1993.

Dewdney, A . K. – *The Turing Omnibus* Computer Science Press,, 1989.

Diverio, T Menezes, P. *Teoria da Computação, máquinas universais e computabilidade*, Editora Sagra Luzzatto, 1999.

D'Ottaviano, I. *Paradoxos auto-referenciais e as lógicas não clássicas heterodoxas*, Ciência e cultura, fevereiro de 1990.

Epstein, preparado por Saulo Almeida FUNBEC- *Computador Gabriela I – Coleção Mirim – Cientistas de amanhã*.

Furuya, N. *Processador de Auxílio ao Ensino* - Dissertação de Mestrado, ICMSC-USP, 1984.

Gardner, M *Aha! Gotcha* – W. H. Freeman and Company, 1981.

Guimarães/Lages – *Introdução à Ciência da Computação* – Livros Técnicos e Científicos Ed. Ltda.

Grimaldi, R. *Discrete and Combinatorial Mathematics*, Addison-Wesley Pub. Company, 1985.

Hopcroft, J. Ullman, J. *Formal Languages and their relation to automata*, Addison-Wesley Pub. Company, 1969.

Kleiner, I. *Paradox in Mathematics: History and Pedagogy*, HPM-Blumenal.

Larry Gonick *Introdução Ilustrada à Computação* publicado pela Harper & Row do Brasil em 1984.

Levin, L., *Universal Search Problems*, Probl. Pered. Inf. 9(3), 1973. Veja também Annals of the History of Computing, 6(4): 384-400, 1984.

Lewis, H. Papadimitriou, C. *Elements of the Theory of Computation*, Prentice Hall, 1981

Menezes, P. *Linguagens Formais e Autômatos*, Ed. Sagra Luzzato, 3a. Ed., 2000.

Moraes, L., Barbosa, L. Leite, N. e Malamud, S. *Pensamento Lógico e Tecnologia*, EDART – FUNBEC, 1977

Rose, A. *Computer Logic* Wiley InterScience, 1971

Smale, S., *Mathematical Problems for the next century*. MATHINT: The Mathematical Intelligencer, 20, 1998.

Sobrinho, J – *Aspectos da Tese de Church-Turing*, Matemática Universitária n. 6, 1987, 1-23.

Tenório, R. *Computadores de Papel: máquinas abstratas para um ensino concreto*, Ed. Cortez, 1991.

Uspensky, V. *A máquina de Post*, Ed. Mir, 1979

Para mais detalhes veja sobre o problema TSP consulte  
[http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB\\_home.html](http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html).

Para ver as biografias dos matemáticos que são citados consulte MacTutor History of Mathematics.

Para informações adicionais sobre Alan M. Turing consulte The Alan Turing Home Page.

Vídeos interessantes ligados à Inteligência Artificial:

*Solaris* de Andrei Tarkovsky

*I A* de Kubrik e Spielberg

*Blade Runner* de Ridley Scott