Teoria da Computação

1 - Limites da Computação

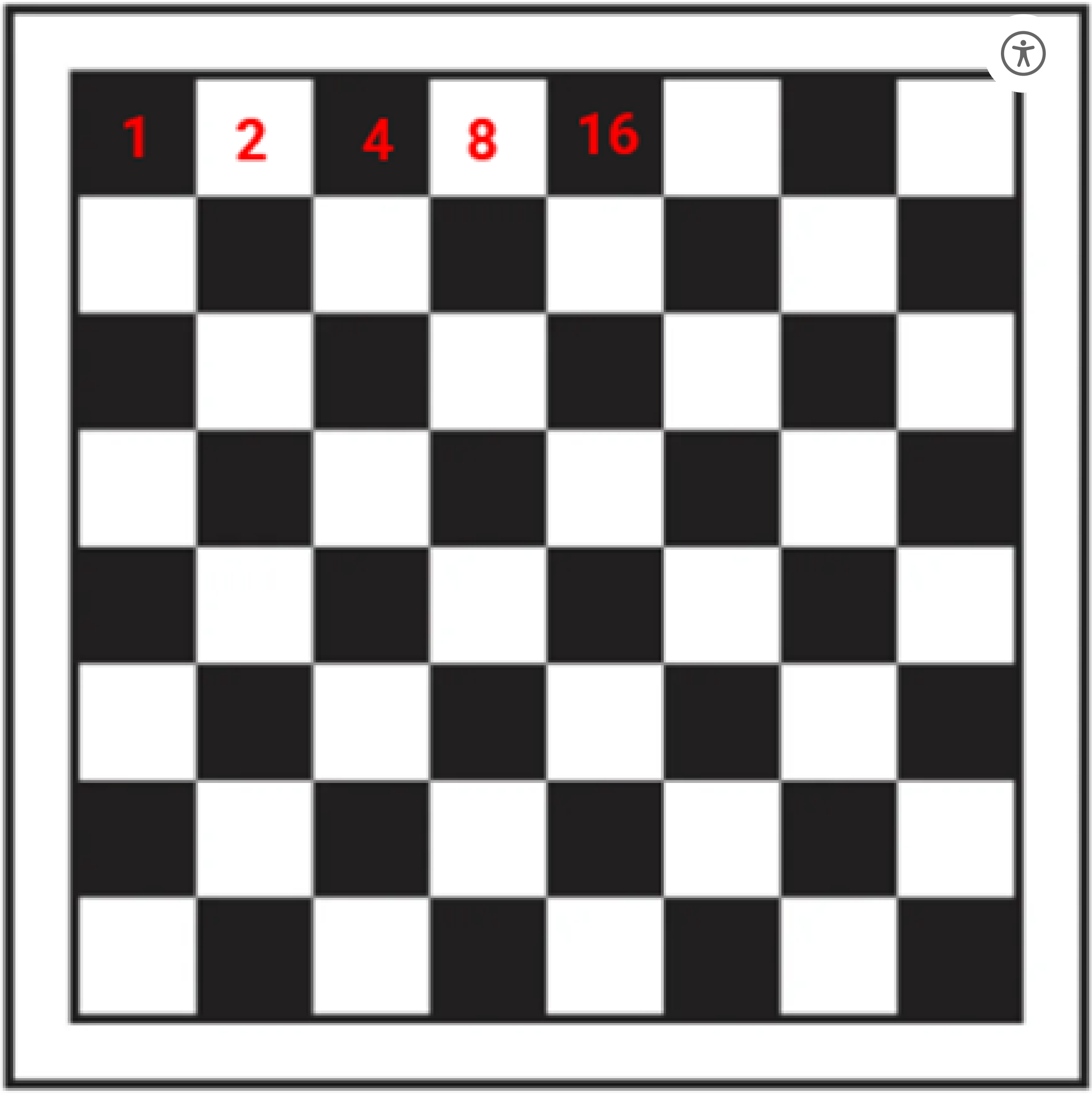
Computadores resolvem uma série de problemas e tarefas mas há problemas e tarefas que ele não consegue realizar.

Em Teoria da Computação estudamos, entre outros temas, os limites da Computação, procurando identificar se pode ser resolvido por uma máquina e a que preço, não só em termos de tempo de processamento mas em número de operações computacionais.

Problemas que tem solução computacional equivalente a funções matemáticas exponenciais costumam ser muito complicados ou mesmo irrealizáveis.

Funções exponenciais crescem muito rápido.

Veja o que aconteceria se fôssemos preencher um tabuleiro de xadrez seguindo a ideia de que a cada casa, teríamos o dobro de elementos da casa anterior:



Cada cssa contém 2^n elementos. Se fôssemos representar grãos de areia, não demoraria ao término de preencher o tabuleiro, teríamos mais grãos de areia do que a quantidade total que existe no planeta Terra! Assim soluções exponenciais para certos problemas são muito difíceis de serem tratadas mesmo por um computador

O problema do Caixeiro Viajante: um clássico da Computação

Um vendedor precisa visitar um certo número de cidades e retornar para casa ao fim do dia. Para elevar seus ganhos, precisa percorrer o itinerário mais curto entre essas cidades. Sabe-se qual a distância entre cada uma das cidades.

Exemplo:

Percorrer as cidades A, B, C, D e depois retornar à cidade de origem A.

Distâncias parciais

	A	B	C	D
A	0	5	6	4
B	5	0	5	5
C	4	6	0	5
D	8	5	3	0

Rota
A – B – C – D – A
A – B – D – C – A
A – C – B – D – A
A – D – C – B – A
A – D – B – C – A

Distância Total	
5 + 5 + 5 + 8 = 23	
5 + 5 + 3 + 4 = 17	**Melhor Rota**
6 + 6 + 5 + 8 = 25	
6 + 5 + 5 + 5 = 21	
4 + 3 + 6 + 5 = 18	
4 + 5 + 5 + 4 = 18	

A velocidade para se encontrar o menor percurso entre um conjunto de cidades depende do computador usado e do número de cidades!

Para **3** cidades existem **3!** combinações possíveis, isto é, **6** combinações para serem testadas.

Para **10** cidades existem **10!** combinações, mais de **3 milhões** de combinações !!

2 - Crescimento Assintótico

Problemas semelhantes a esse são de extrema importância para a indústria, considere, por exemplo, as várias etapas envolvidas na montagem de um motor e que devem ser efetuadas em uma certa ordem - esse é conhecido como **Problema do Sequenciamento de Tarefas**.

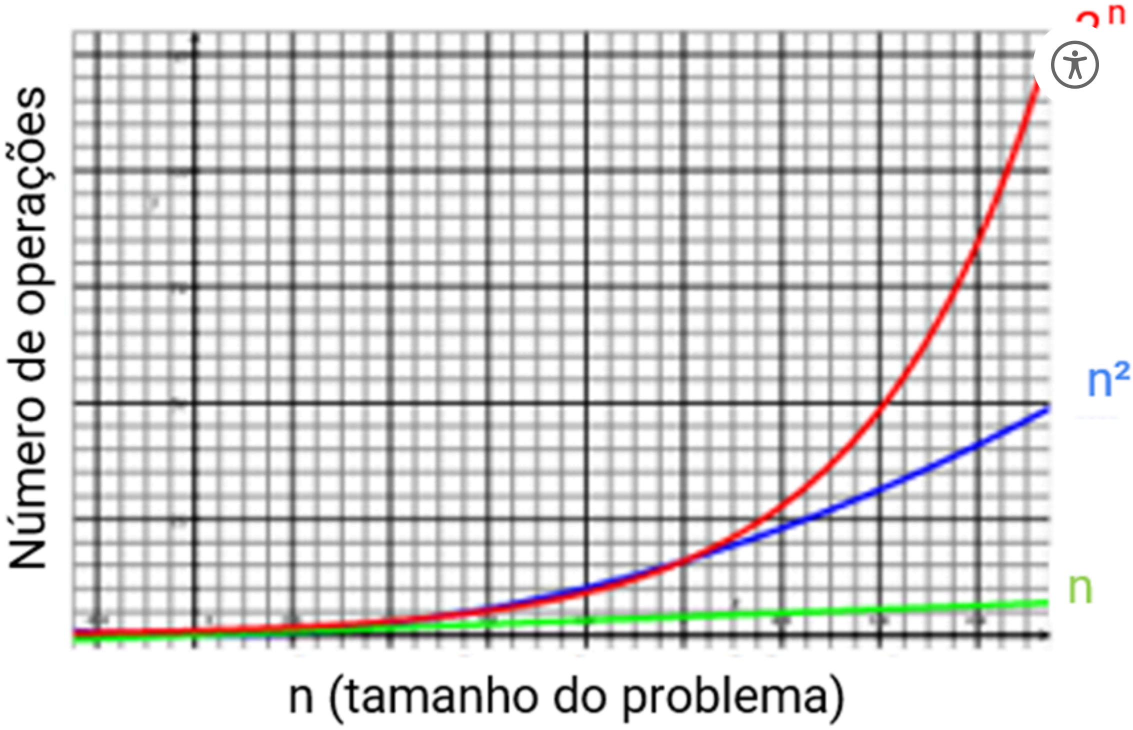
O crescimento da função é exponencial e extremamente trabalhoso mesmo para um computador, daí a importância da análise de complexidade de algoritmos: estudar meios para se conhecer melhor problemas e os limites de suas soluções.

Na análise de algoritmos, não interessa a velocidade do computador mas sim o número de passos computacionais envolvidos.

Soluções computacionais (algoritmos) que operam em velocidade logarítmica, linear ou polinomial são incomparavelmente menos custosos computacionalmente do que algoritmos que operam de forma exponencial. Veja a tabela abaixo, em que N representa o número de elementos de entrada envolvidos e o processamentos de cada instrução é um bilionésimo de segundo:

	Função tempo/ complexidade *F(N)*	Quantidade de dados (N) 10	20	30	40	50
Polinomial:	N	0,00001s	0,00002s	0,00003s	0,00004s	0,00005s
	N^2	0,0001s	0,0004s	0,0009s	0,0016s	0,0036s
	N^3	0,001s	0,008s	0,027s	0,064s	0,125s
Exponencial:	2^N	0,001s	1,0s	17,19s	12,7dias	35,7anos
	3^N	0,059s	58min	6,5anos	3.855séc.	200.000.000séc.

Olhando graficamente agora, repare que a partir de um certo tamanho **n** de entrada, a função exponencial cresce muito mais rápido:



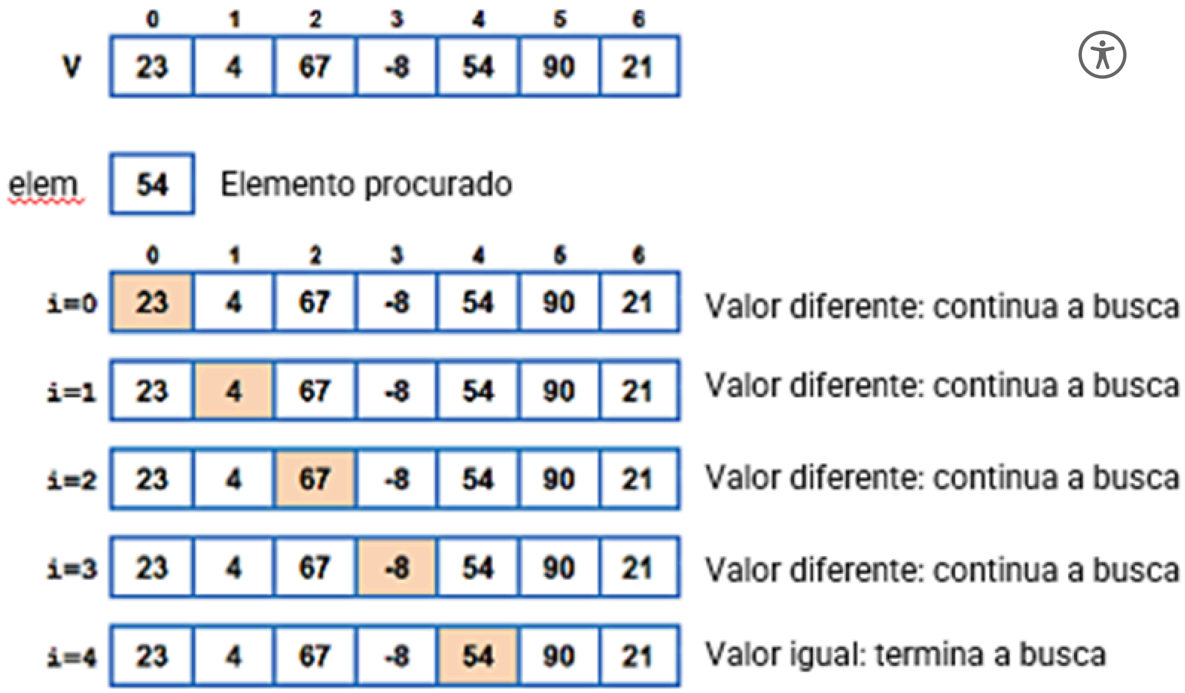
3 - Buscas e Ordenações

Se você pensar bem, quase tudo que fazemos em termos de computação é a busca de elementos em uma lista ou ordenação de elementos em uma lista!

Métodos de **Busca** e **Ordenação** são a essência de muitos problemas e soluções em Computação!

Busca sequencial ou linear

A busca sequencial ou linear é muito importante em estudos comparativos de análise de algoritmos. Dada uma lista de elementos, procura-se saber se um determinado elemento está na lista ou não. Leia atentamente a figura:



Percebe-se que a busca sequencial é um algoritmo pouco eficiente para encontrar um item em uma lista grande de elementos. No pior caso, todos os elementos devem ser visitados apenas para se descobrir que o elemento não estava na lista !

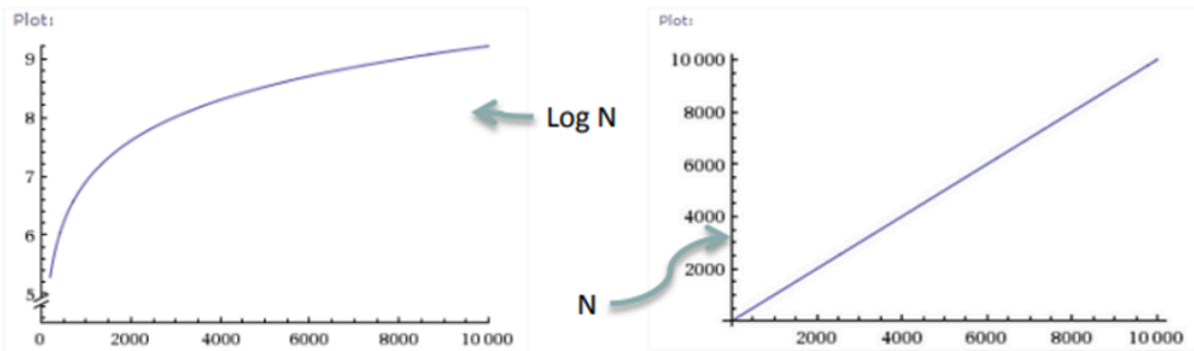
Busca binária

Neste caso, a lista está previamente ordenada. O método faz sucessivas divisões na lista comparando com o valor maior da primeira lista e descarta metade da lista até encontrar o elemento que está procurando.

Comparando busca sequencial com busca binária

- Se a lista não está ordenada, a busca binária não se aplica.
- A busca sequencial é muito ineficiente para conjuntos grandes de valores.

Olhando para os gráficos de crescimento assintótico, notamos:

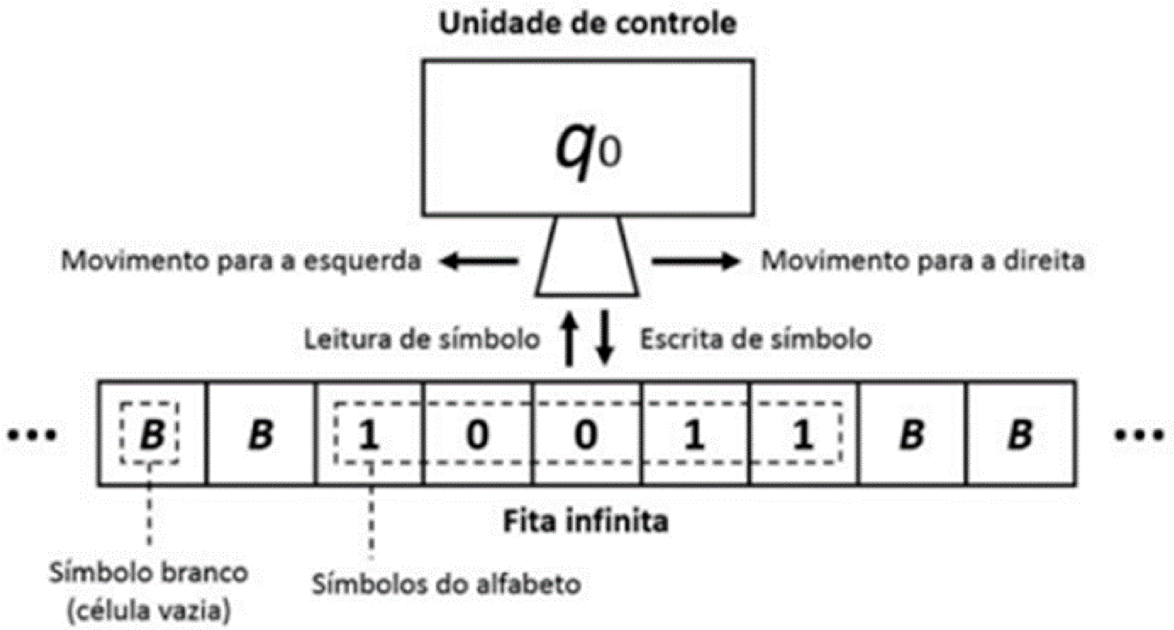


Costuma-se dizer que a busca binária tem complexidade n e que a busca binária tem complexidade log n, fazendo muito menos comparações a partir de um certo n de entrada. O gráfico da Busca binária cresce mais devagar, fazendo deste método mais eficiente do que a busca sequencial.

4 – Máquinas de Turing

É um dispositivo imaginário que formou a estrutura para fundamentar a ciência da computação moderna.

Em 1936 foi formalizado o termo algoritmo: um conjunto finito de instruções simples e precisas, que são descritas com um número finito de símbolos. “Qualquer processo aceito por nós homens como um algoritmo é precisamente o que uma máquina de Turing pode fazer” (Alonzo Church, matemático).



Exemplo de Máquina de Turing

As duas primeiras instruções (linhas 1 e 2) descrevem o que acontecerá no estado s0. Há duas possibilidades: na primeira, a máquina faz a leitura de um dígito ‘1’, movimentará a cabeça para a direita e permanecerá no estado s0. Na segunda, se for lido um dígito ‘0’ a máquina deixará o estado s0, entrará no estado s1 e escreverá o dígito ‘1’ nessa transição.

Linhas 3 e 4 mostram o que acontecerá no estado s1, ou seja, se for lido o dígito ‘1’, a máquina movimentará a cabeça para a esquerda e permanecerá no estado s1. Se for lido o dígito ‘0’, a cabeça será movimentada para a direita e a máquina passará para o estado s2.

Como não há instruções definidas pelo algoritmo no estado s2, a máquina para a sua execução (condição de parada) ao atingir este estado.

1. $\langle s_0, 1, s_0, » \rangle$
2. $\langle s_0, 0, s_1, 1 \rangle$
3. $\langle s_1, 1, s_1, « \rangle$
4. $\langle s_1, 0, s_2, » \rangle$

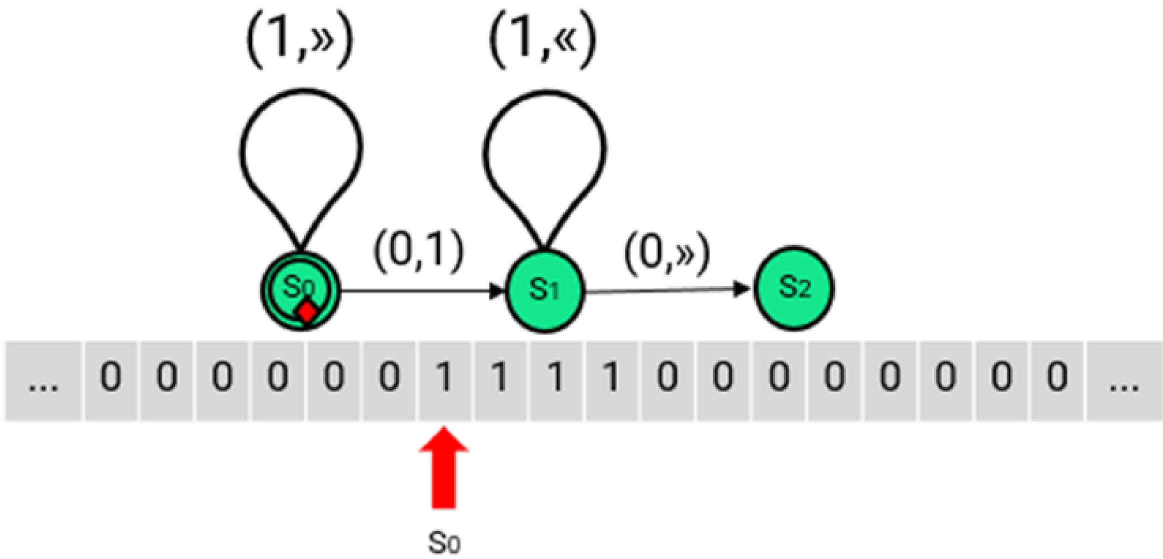
As duas primeiras instruções (linhas 1 e 2) descrevem o que acontecerá no estado s0. Há duas possibilidades: na primeira, a máquina faz a leitura de um dígito ‘1’, movimentará a cabeça para a direita e permanecerá no estado s0. Na segunda, se for lido um dígito ‘0’ a máquina deixará o estado s0, entrará no estado s1 e escreverá o dígito ‘1’ nessa transição.

Linhas 3 e 4 mostram o que acontecerá no estado s1, ou seja, se for lido o dígito ‘1’, a máquina movimentará a cabeça para a esquerda e permanecerá no estado s1. Se for lido o dígito ‘0’, a cabeça será movimentada para a direita e a máquina passará para o estado s2.

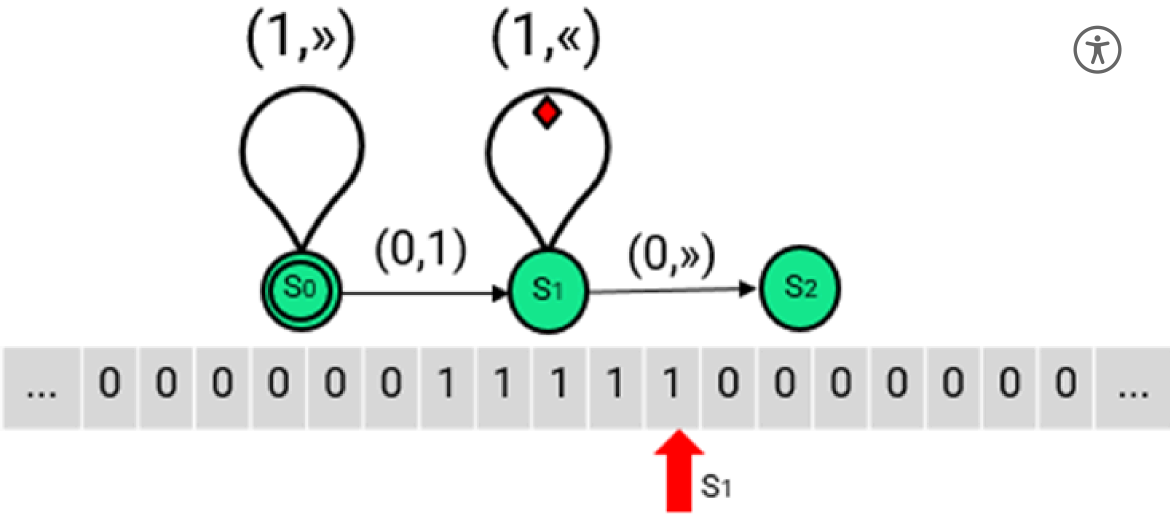
Como não há instruções definidas pelo algoritmo no estado s2, a máquina para a sua execução (condição de parada) ao atingir este estado.

Acompanhe na figura:

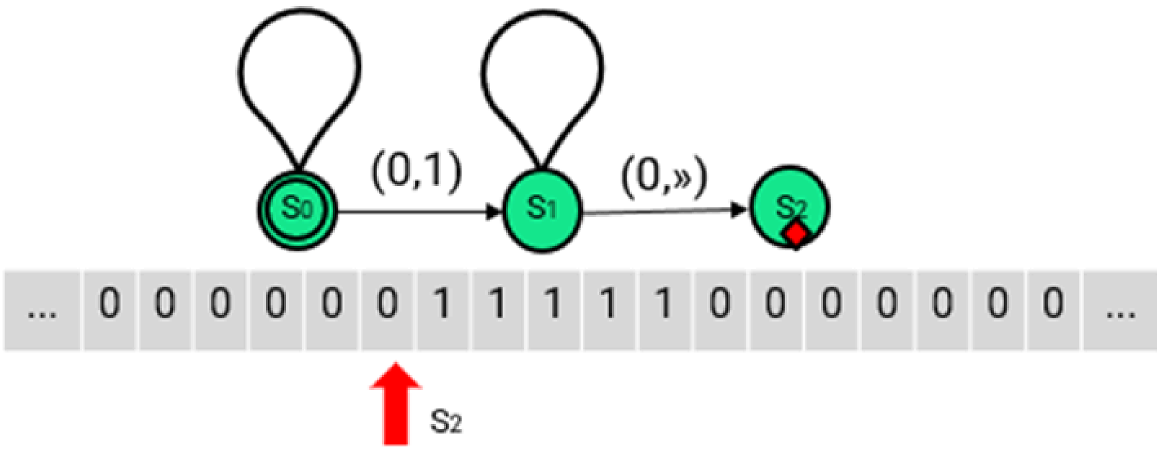
O leitor lê 1 no estado SO. A máquina move a fita para a direita e fica no estado SO. Isso acontece sucessivamente.



O leitor lê 0 ainda no estado SO. Nesse momento a máquina escreve 1 na fita e muda o estado para S1. A seguir lê 1 e está no estado S1. A máquina desloca a fita para a esquerda e o processo se repete.



Agora a leitora lê 0 enquanto está no estado S1. Muda para o estado S2 e desloca a fita. Como não há o que fazer no estado S2 a máquina pára.



A máquina de Turing Universal incorpora o princípio essencial do computador: uma máquina simples que poderá executar qualquer tarefa bem definida, desde que especificada com um programa apropriado.

5 – Autômatos finitos

Portas automáticas e elevadores

Um estado pode representar em qual estado o elevador está e as entradas podem ser os sinais recebidos dos botões.

Tal computador precisaria de poucos bits para guardar essa informação.

Dispositivos desse tipo requerem que se utilize a metodologia e a terminologia de autômatos finitos.

Modelos para computadores quando existe pouca disponibilidade de memória. Esses computadores estão no coração de vários dispositivos eletromecânicos (forno de micro-ondas, máquinas de lavar, portas automáticas, elevadores).

Lida com definições e propriedades de modelos matemáticos da computação.

Autômatos finitos são usados em processamento de textos, compiladores e projetos de hardware.

O modelo denominado gramática livre de contexto é utilizado em linguagens de programação e inteligência artificial

Autômatos finitos

É uma lista que contém cinco objetos (uma 5-upla):

- conjunto de estado
- alfabeto de entrada
- função de transição
- estado inicial

- estados de aceitação

Definições formais:

Alfabeto: qualquer conjunto finito não vazio constituído por caracteres.

Ex:

$\Sigma_1 = \{0, 1\}$

$\Sigma_2 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p\}$

Cadeia sobre um alfabeto:

Sequência finita de símbolos de um alfabeto.

Ex:

010011 é uma cadeia sobre Σ_1.

abracadabra é uma cadeia sobre Σ_2.

Um autômato de 3 estados

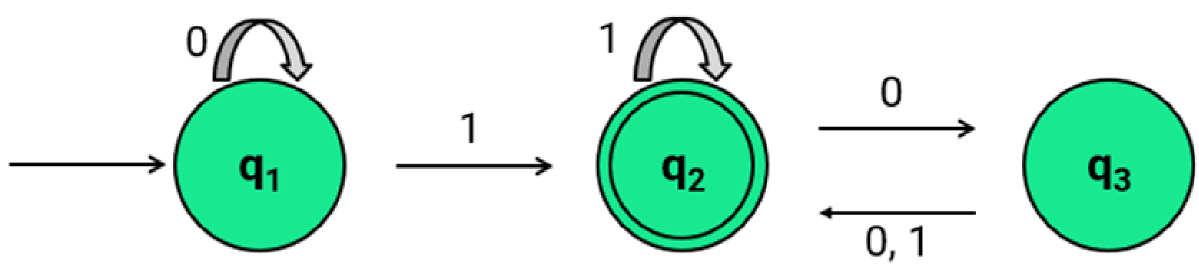


Diagrama de estado M1. O autômato tem 3 **estados**, q1, q2, q3. O estado inicial q1 é indicado pela seta apontando para ele a partir do nada. O **estado de aceitação q2, é aquele com círculo duplo**. As setas saindo de um estado para o outro são chamadas de **transições**.

Quando o autômato recebe uma cadeia de entrada tal como 1101, ele processa essa cadeia e produz uma saída. A saída será **aceita** se estiver no estado de aceitação ou **rejeitada**, caso contrário.

O que acontece para a entrada: 1101 ? ACEITA! Já 0011110? NÃO ACEITA!

Essa máquina aceita 1, 01, 11, 0101010101, entre infinitas outras.

Na verdade, **ela aceita qualquer cadeia que termine com o símbolo 1**, pois ela vai para o estado de aceitação sempre que lê o símbolo 1.

Aceita também cadeias que terminem com um número par de 0 's, seguindo o último 1.

Referência Bibliográfica

BROOKSHEAR, J.G. **Ciência da Computação: uma visão abrangente.** Porto Alegre: Bookman, 2013.

CORMEN, Thomas H. **Algoritmos**. Rio de Janeiro, Gen LTC, s/a.

Ir para exercício