PCC104 - Projeto e Análise de Algoritmos

Marco Antonio M. Carvalho

Departamento de Computação
Instituto de Ciências Exatas e Biológicas
Universidade Federal de Ouro Preto

Conteúdo

Fonte

Este material é baseado nos livros

- T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.
- S. Halim. *Competitive Programming*. 3rd Edition, 2013.
- Ian Parberry and William Gasarch. *Problems on Algorithms*. Second Edition, 2002.
- Ian Parberry *Lecture Notes on Algorithm Analysis and Complexity Theory*. Fourth Edition, 2001.

Licença

Introdução

Quase todos algoritmos vistos até aqui possuem tempo polinomial: em entradas de tamanho n, o tempo de execução no pior caso é $O(n^k)$, para alguma constante k.

Podemos pensar, intuitivamente, que todos os problemas podem ser resolvidos em tempo polinomial.

Entretanto, a resposta é não. Há problemas que não podem ser resolvidos pelo computador, e outros que podem, porém, não em tempo $O(n^k)$, para alguma constante k.

Geralmente, designamos os problemas resolvíveis em tempo polinomial como *tratáveis*, ou *fáceis*.

Por outro lado, designamos os problemas resolvíveis apenas em tempo supra polinomial como *intratáveis*, ou *difíceis*.

Introdução

Um problema é considerado indecidível caso não seja possível criar um algoritmo qualquer para sua solução, *e. g.*, o Problema da Parada:

"*Dadas uma descrição de um programa e uma entrada finita, decida se o programa termina de rodar ou rodará indefinidamente, dada essa entrada.*"

Um problema é considerado intratável caso não haja algoritmo em tempo polinomial que o resolva deterministicamente.

Veremos alguns exemplos mais à frente.

Introdução

Em geral, consideramos que os problemas resolvíveis em tempo polinomial são tratáveis, mas por razões filosóficas, e não matemáticas:

- Embora seja razoável considerar intratável um problema que exige tempo $O(n^{100})$, um número muito pequeno de problemas práticos exigem tempo de ordem tão alta;
- Um problema que pode ser resolvido em tempo polinomial em um modelo computacional pode também ser resolvido em tempo polinomial, para muitos modelos razoáveis de computação;
- A classe de problemas resolvíveis em tempo polinomial tem propriedades de fechamento interessantes, já que polinômios são fechados por adição, multiplicação e composição.

Introdução

Uma classe importante de problemas, denominada NP-completo, possui status desconhecido.

Não se desenvolveu nenhum algoritmo de tempo polinomial para nenhum problema desta classe, porém, não se provou também a impossibilidade de tal algoritmo existir – eis a famosa questão P vs. NP.

Vários problemas NP-completos são interessantes porque se parecem muito com problemas fáceis:

- Caminho mais curto vs. Caminho mais longo;
- Ciclo Euleriano vs. Ciclo Hamiltoniano;
- 2-SAT vs. 3-SAT.

Computabilidade e Teoria da Complexidade Computacional

A Computabilidade e a Teoria da Complexidade Computacional estudam os **limites** da computação:

- Quais problemas jamais poderão ser resolvidos por um computador, independente da sua velocidade ou memória?
- Quais problemas podem ser resolvidos por um computador, mas requerem um período tão extenso de tempo para completar a ponto de tornar a solução impraticável?
- Em que situações pode ser mais difícil resolver um problema do que verificar cada uma das soluções manualmente?

Recapitulando...

Problema de Decisão

Tipo de problema computacional em que a resposta para cada instância é **sim** ou **não**:

"Dadas uma lista de cidades e as distâncias entre todas, há uma rota que visite todas as cidades e retorne à cidade original com distância total menor do que 500 km?"

Problema de Otimização

Tipo de problema computacional em que é necessário determinar a **melhor** solução possível entre todas as soluções viáveis.

"Dadas uma lista de cidades e as distâncias entre todas, determine a menor rota que visite todas as cidades e retorne à cidade original."

Problemas de Decisão e Otimização

Geralmente, os problemas de decisão não são mais difíceis que os problemas de otimização.

Se pudermos caracterizar o problema de decisão como difícil, também estaremos caracterizando o problema de otimização relacionado como difícil.

NP-Completude

A teoria da NP-Completude, por conveniência, se aplica a problemas de decisão.

A razão pela qual nos concentraremos nos problemas de decisão é a propriedade natural de codificação por linguagens formais.

Codificações

Para um programa de computador resolver um problema abstrato, temos de representar as instâncias de um problema de modo que o programa entenda.

Uma codificação consiste em um mapeamento de objetos abstratos para cadeias, por exemplo, binárias.

Dizemos que um algoritmo **resolve** um problem em tempo $O(T(n))$ se, dada uma instância i de comprimento $n = |i|$, o algoritmo gerar a solução em tempo máximo $T(n)$.

Utilizando o conceito de codificação, podemos criar uma codificação mais geral e independente para problemas, denominadas linguagens formais.

Linguagens e Problemas de Decisão

Como mencionado, o foco em problemas de decisão permite utilizar os benefícios da teoria das linguagens formais.

Os conceitos são os mesmos da linguística, que define uma linguagem formal como um conjunto de cadeias de caracteres com um conjunto de regras específicas.

Linguagens

Para um conjunto finito de símbolos (ou alfabeto) Σ, denotamos por Σ^* o conjunto de todas as cadeias finitas de símbolos de Σ.

Por exemplo, para $\Sigma = \{0, 1\}$, temos que Σ^* é composto por $\emptyset$, 0, 1, 00, 11, e todas as cadeias finitas de 0s e 1s.

Um subconjunto L de Σ^* é chamado de linguagem sobre o alfabeto Σ.

Aceitação e Rejeição

Dizemos que um algoritmo A aceita uma cadeia $x \in \Sigma^*$ se dada a entrada x, a saída do algoritmo é $A(x) = 1$.

O mesmo algoritmo A rejeita uma cadeia x se $A(x) = 0$.

A linguagem aceita por um algoritmo A é o conjunto de cadeias $L = \{x \in \{0,1\}^* : A(x) = 1\}$.

Decisão

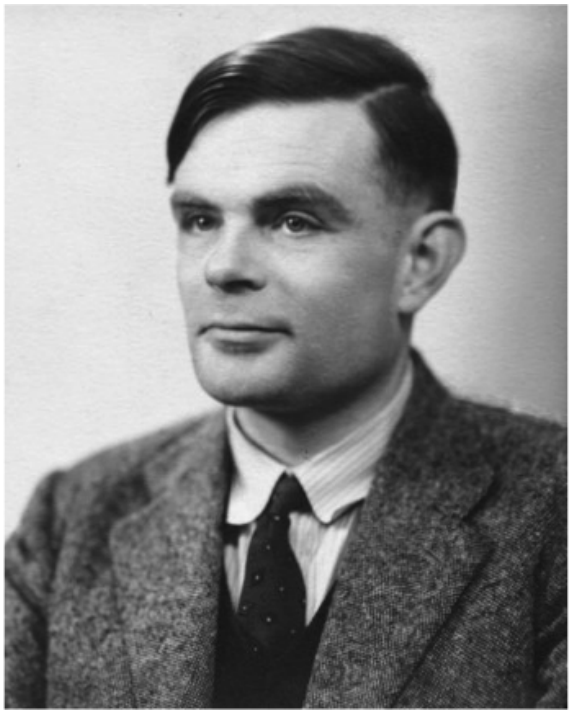
Ainda que a linguagem L seja aceita por um algoritmo A, o algoritmo não necessariamente rejeitará uma cadeia $x \notin L$ dada como entrada.

Por exemplo, o algoritmo pode simplesmente entrar em loop infinito.

Uma linguagem L é decidida por um algoritmo A se toda cadeia em L é aceita por A e toda cadeia não pertencente a L é rejeitada por A.

Uma linguagem L é aceita em tempo polinomial por um algoritmo A se for aceita por A propriamente e se houver uma constante k tal que, para qualquer cadeia $x \in L$ de comprimento n, A aceita x em tempo $O(n^k)$.

Uma linguagem L é decidida em tempo polinomial por um algoritmo A se houver uma constante k tal que, para qualquer cadeia $x \in \Sigma^*$ de comprimento n, A decide **corretamente** se $x \in L$ em tempo $O(n^k)$.



Alan Mathison Turing

- Matemático, lógico e criptoanalista inglês;
- Participação importante na II Guerra Mundial;
- **Pai da Ciência da Computação**
  - Dedicou a vida à teoria da computabilidade;
  - Formalizou os conceitos de algoritmo e computabilidade;
  - Aos 24 anos (1936), criou a Máquina de Turing;
  - Parte de sua vida foi retratada no filme *Breaking the Code* (1996) e no filme *The Imitation Game* (2014).
- Hoje o **Prêmio Turing** equivale ao Nobel da Computação.

Introdução

A noção de algoritmos pode ser formalizada utilizando uma máquina universal chamada **Máquina de Turing Determinística**:

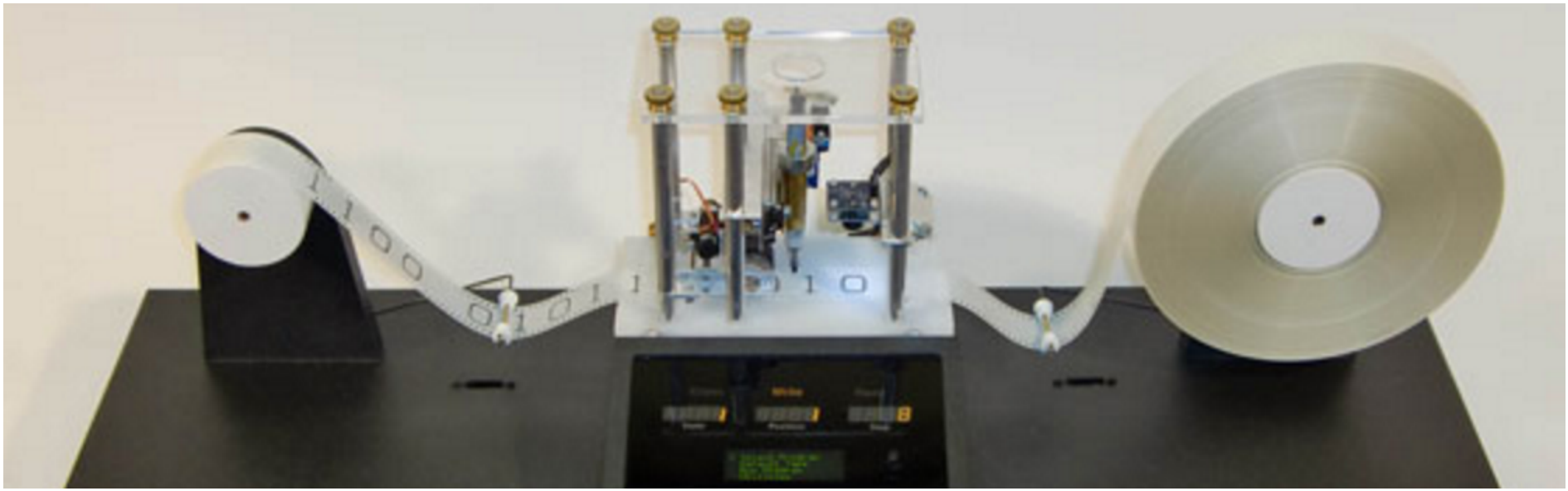
- Funciona como computadores elementares, emulando a parte lógica;
- Foi idealizada muitos anos antes dos computadores digitais;
- O determinismo vem do fato de podermos predizer seu comportamento;
- Opera sobre linguagens.

Definição

Uma máquina de Turing é composta por:

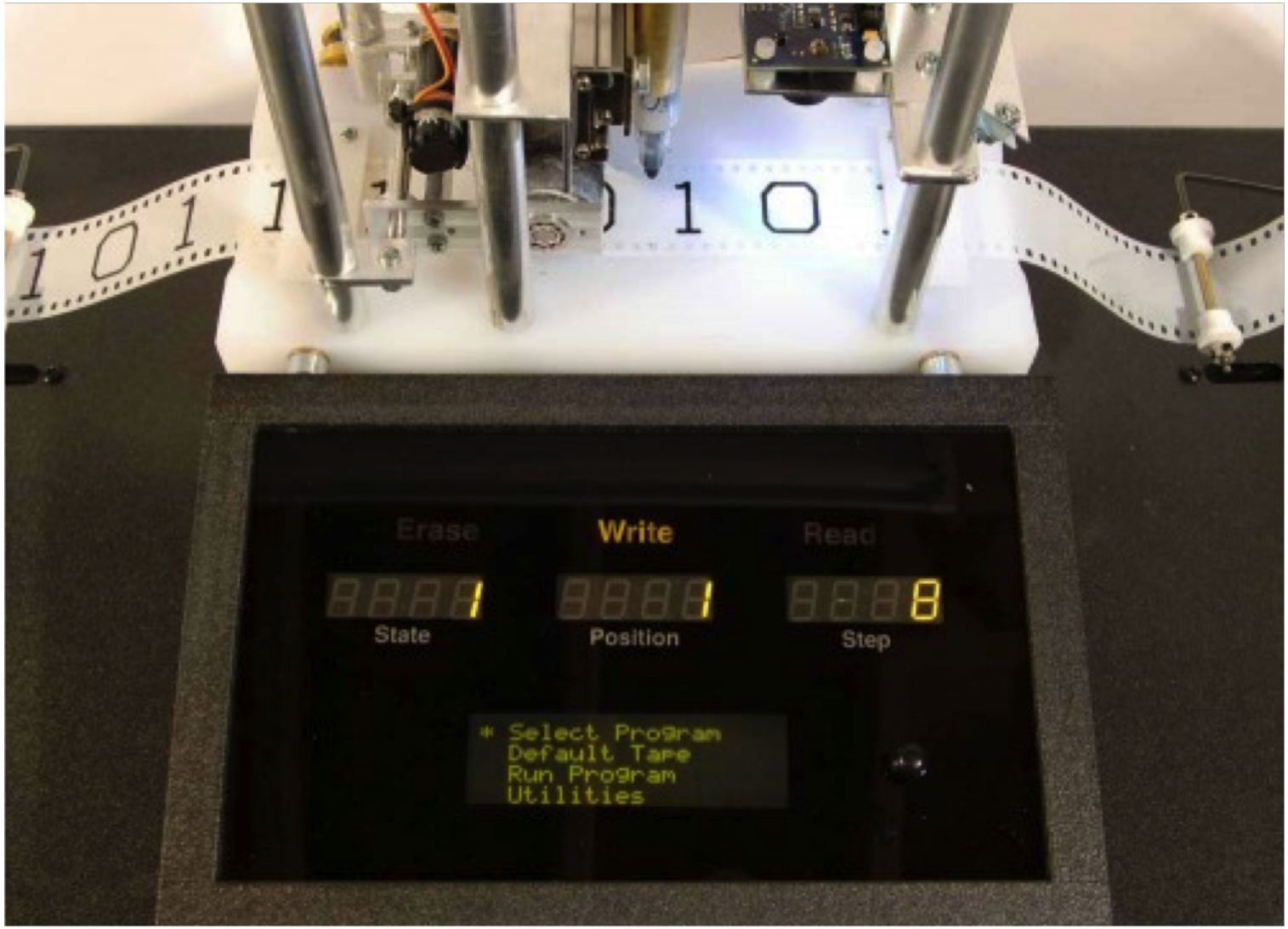
- Uma **fita** infinita nos dois sentidos, dividida em células contíguas com os dados de entrada, um símbolo por célula;
- Um **cabeçote** de leitura e escrita na fita;
- Um **registrador de estados**, que indica o estado atual da máquina;
- Uma **função de transição**, que dados o símbolo lido na fita e o estado atual, indica o que deve ser escrito, em qual direção o cabeçote deve se mover e qual será seu novo estado.

Máquinas de Turing



Implementação de uma máquina de Turing.

Máquinas de Turing



Implementação de uma máquina de Turing.

Definição

Um programa para uma máquina de Turing especifica:

- Um conjunto finito Γ de símbolos da fita, incluindo os símbolos de Σ e o símbolo especial "em branco" b (ou $\square$);
- Um conjunto finito Q de estados, incluindo o estado inicial q_0 e estados finais q_s e q_n;
- Uma função de transição δ que, a partir do estado atual e o símbolo lido na fita, determina qual é o estado seguinte, qual símbolo deve ser escrito na fita e em qual direção o cabeçote deve se movimentar:

$$\delta : (Q - \{q_s, q_n\} \times \Gamma \rightarrow Q \times \Gamma \times \{-1, +1\})$$

Exemplo

Suponhamos $\delta(q, s) = (q', s', \Delta)$.

No estado q, ao ler o símbolo s, a máquina de Turing vai para o estado q', escreve s' no lugar de s e se movimenta para a direita ou esquerda, dependendo do valor de Δ.

Definição

Dada uma cadeia de símbolos de entrada, com início na célula número 1:

- A partir do estado inicial, a execução é passo-a-passo:
  - Se um estado final foi atingido, a computação terminou;
  - Caso contrário, existe algum símbolo a ser lido na fita;
  - Lido um símbolo, a função de transição informa, de acordo com o estado atual, o que deve ser escrito, em qual posição o cabeçote deve se mover e qual será seu novo estado.

Máquinas de Turing também podem ser representadas por diagramas de estados, em que todas as informações estão contidas.

q	0	1	b
q_0	(q_0, 0, +1)	(q_0, 1, +1)	(q_1, b, -1)
q_1	(q_2, b, -1)	(q_3, b, -1)	(q_n, b, -1)
q_2	(q_s, b, -1)	(q_n, b, -1)	(q_n, b, -1)
q_3	(q_n, b, -1)	(q_n, b, -1)	(q_n, b, -1)

Exemplo em que $\Gamma = \{0, 1, b\}$, $\Sigma = \{0, 1\}$ e $Q = \{q_0, q_1, q_2, q_3, q_s, q_n\}$.

Compute $x = 10100$ e $x = 111$.

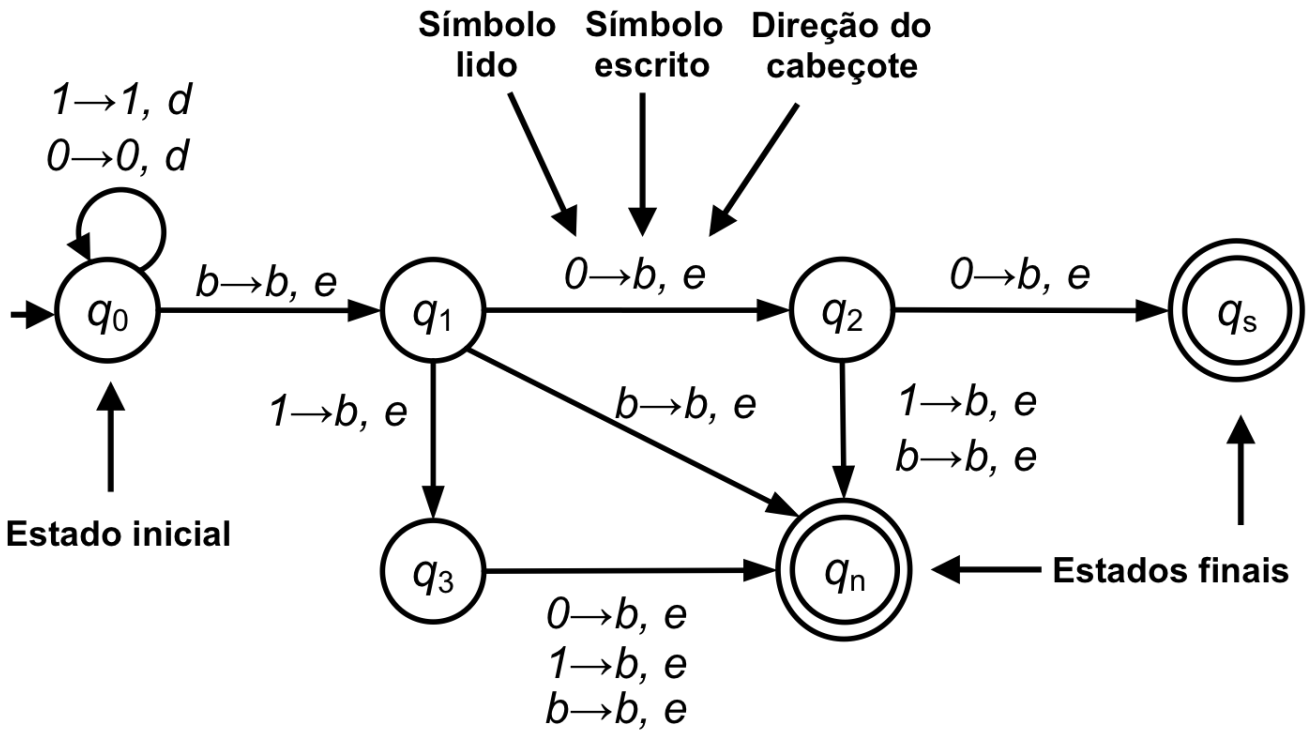


Diagrama de estados para a tabela de transições anterior.

q	0	1	b
q_0	$(q_0, 0, +1)$	$(q_0, 1, +1)$	$(q_1, b, -1)$
q_1	$(q_2, b, -1)$	$(q_3, b, -1)$	$(q_n, b, -1)$
q_2	$(q_s, b, -1)$	$(q_n, b, -1)$	$(q_n, b, -1)$
q_3	$(q_n, b, -1)$	$(q_n, b, -1)$	$(q_n, b, -1)$

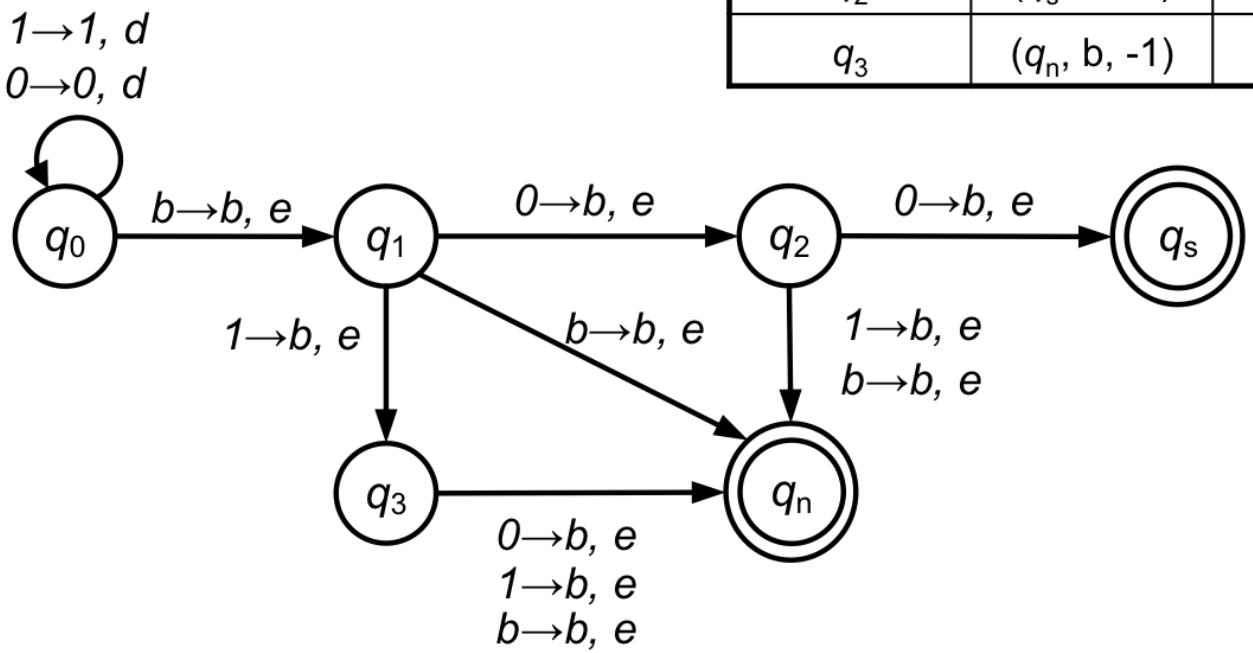


Diagrama de estados e tabela de transições.

Definição

Dizemos que um programa M com alfabeto Σ aceita x pertencente a Σ^* se e somente se a computação termina no estado terminal (estado q_s) quando a entrada é x.

A linguagem L_M reconhecida pelo programa M é definida por todas as cadeias de símbolos x pertencentes a Σ^* tal que M aceita x.

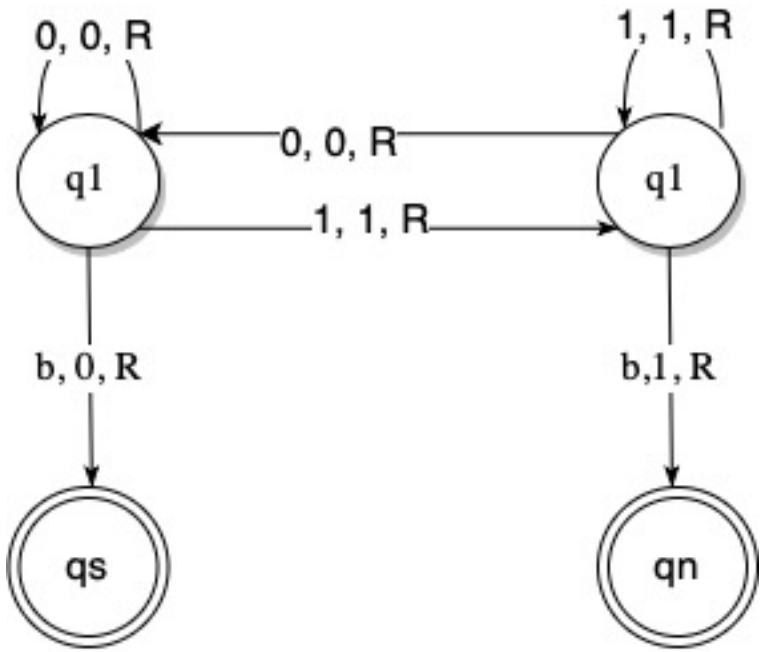
M não aceita todas as cadeias em Σ^*, apenas aquelas pertencentes a L_M – as demais ou param no estado q_n ou não param.

A correspondência entre aceitar uma linguagem e resolver problemas de decisão é direta.

Aplicações

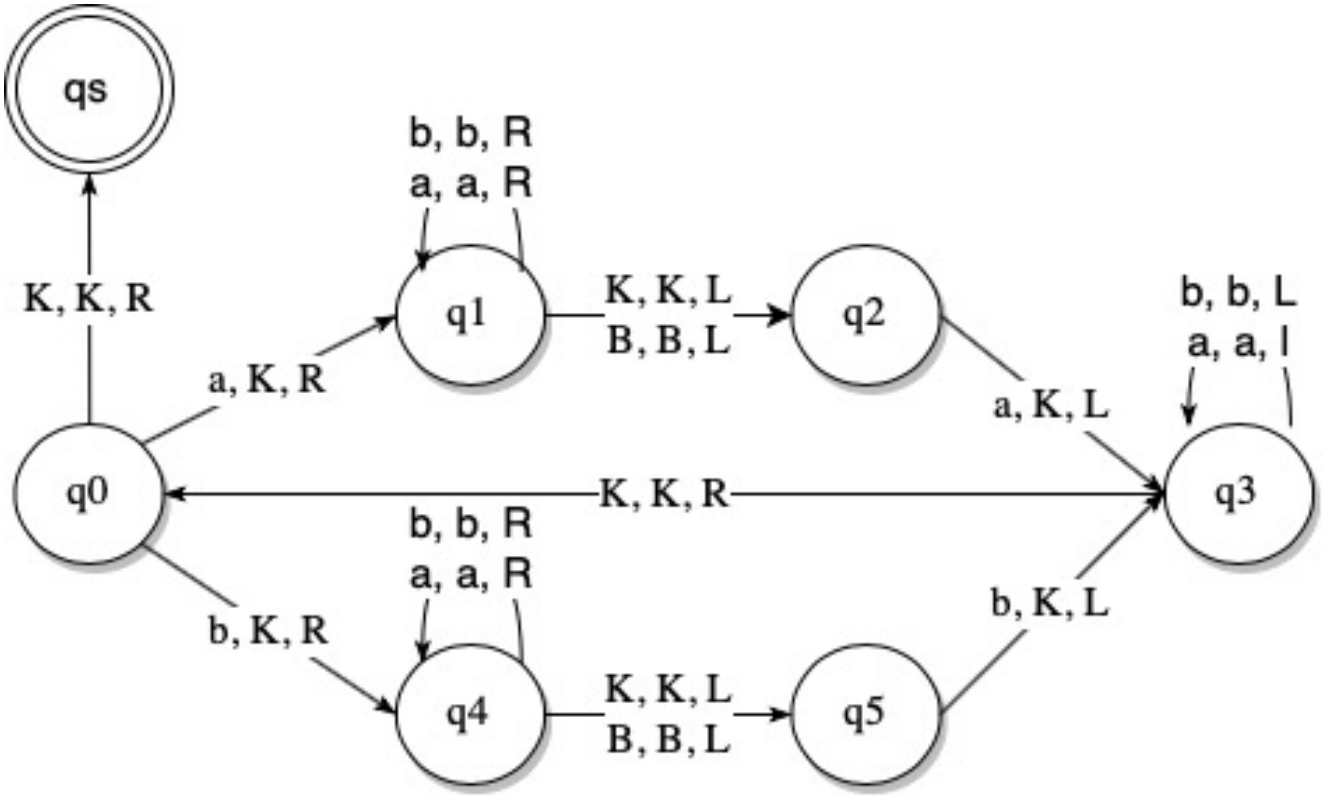
Uma máquina de Turing possui basicamente três aplicações gerais:

1. Reconhecer linguagens;
2. Calcular funções;
3. Processar problemas de decisão.



Máquina de Turing que decide se um número representado na base binária é par ou ímpar.

Máquinas de Turing



Máquina de Turing que reconhece $ww^r \mid w$ é uma palavra de $\{a, b\}^+$.

Determinismo e Não Determinismo

Definições

Em um algoritmo determinístico, podemos predizer o seu comportamento: para as mesmas entradas de um problema, teremos sempre as mesmas saídas.

Em um algoritmo não determinístico, existe um "**adivinho**" (ou "**oráculo**"): se existir uma solução, o algoritmo não determinístico simplesmente a "adivinha".

Algoritmos não determinísticos também podem ser polinomiais no tamanho da entrada, se a "adivinhação" levar tempo polinomial.

Máquinas de Turing Não Determinísticas

Definição

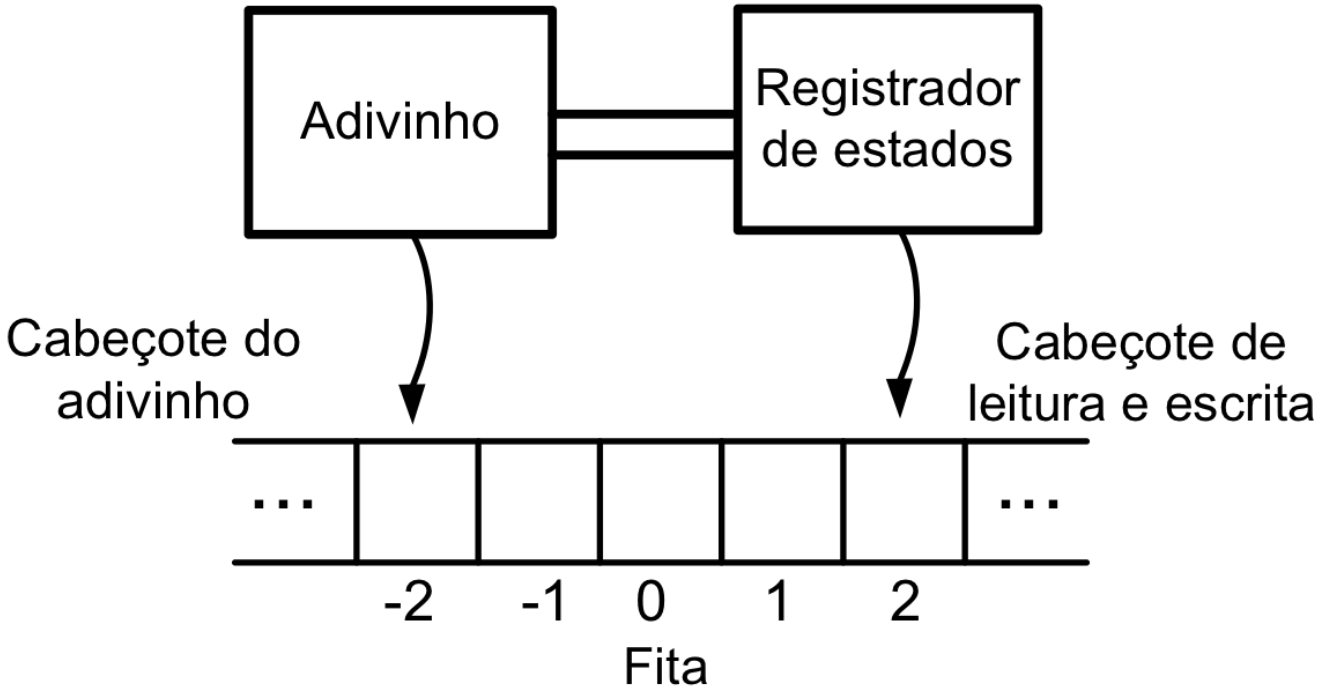
Máquinas de Turing Não Determinísticas podem ser vistas como as anteriores, com a adição de um módulo "adivinho".

Antes da execução normal, arbitrariamente, o adivinho escreve a solução na fita.

Posteriormente, o cabeçote executa normalmente, verificando a validade da solução e aceitando-a.

Os programas são especificados como antes, e o conceito de linguagem também é mantido.

Máquinas de Turing Não Determinísticas



Máquina de Turing não determinística.

Máquinas de Turing Não Determinísticas

Representação

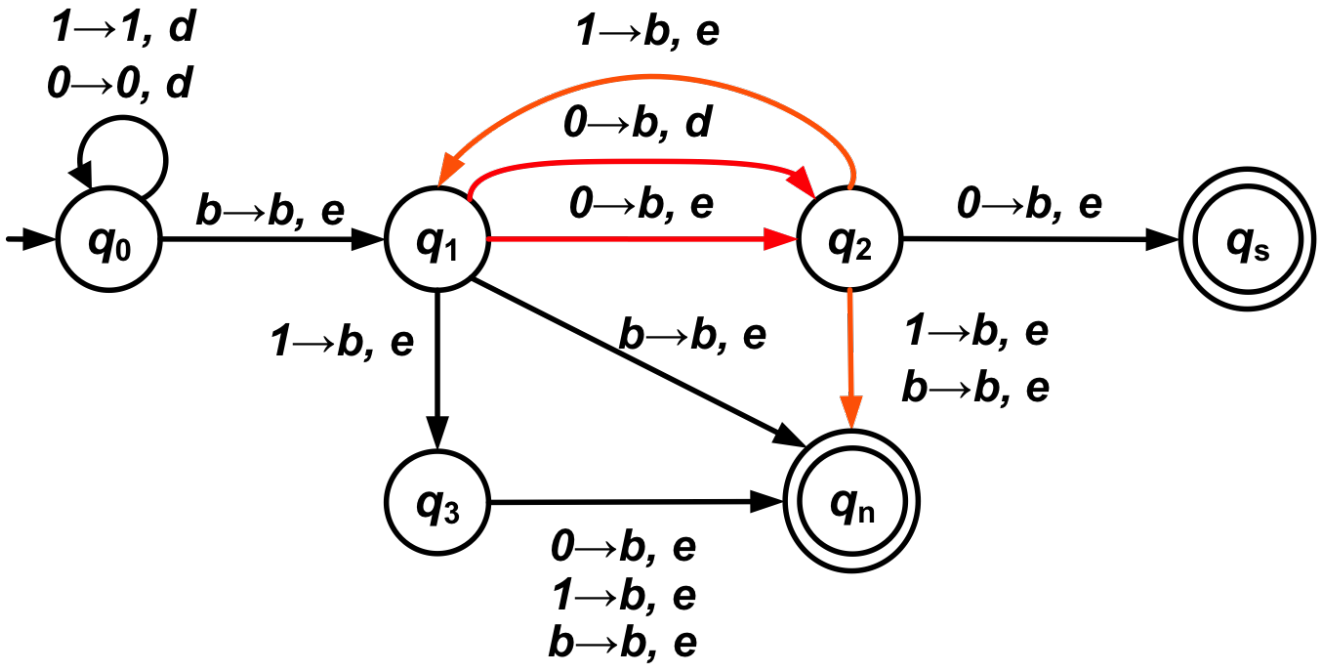
A representação por diagramas de estados de uma máquina de Turing não determinística é semelhante à anterior.

Porém, pode existir mais de uma transição para o mesmo símbolo a partir de um estado.

Nesse caso, o "adivinho" diz qual das transições para um determinado símbolo deve ser utilizada.

A condição de aceitação é a mesma.

Máquinas de Turing Não Determinísticas



Exemplo de Máquina de Turing não determinística.
Em detalhe, as múltiplas transições dado um mesmo símbolo.

Tempo Polinomial

Definição

Um problema é solucionável em tempo polinomial determinístico se existir uma máquina de Turing determinística que o solucione em tempo limitado por um polinômio em relação ao tamanho da entrada.

A máquina de Turing determinística equivale a um algoritmo determinístico.

Classe P

Definição

Consiste nos problemas que podem ser resolvidos deterministicamente em tempo polinomial no tamanho da entrada, ou seja, existem algoritmos de complexidade $O(n^k)$ para k constante que os resolvam.

P é uma referência a tempo determinístico polinomial.

Exemplos: pesquisa, ordenação, busca em grafos, menor caminho em grafos, fluxo máximo em grafos, detecção de árvores geradoras mínimas e classificação de arestas e vértices.

Classe NP

Definição

Consiste nos problemas que são verificáveis em tempo polinomial. Dada uma solução para o problema, podemos verificar se é uma solução válida em tempo polinomial.

NP é uma referência a Tempo Não Determinístico Polinomial.

Em geral, é mais difícil resolver um problema do que verificar uma dada solução. Isto leva alguns teóricos a acreditarem que há problemas em NP que não estão em P.

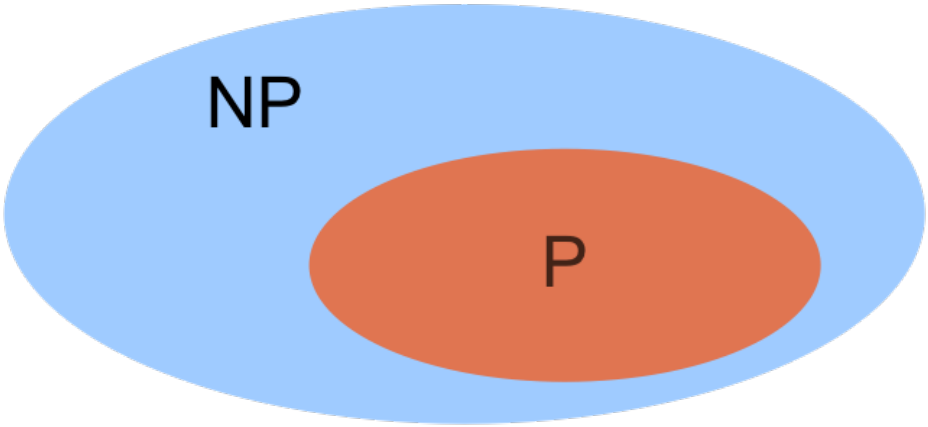
P vs. NP

É possível verificar uma solução para um problema da classe P em tempo polinomial, logo, qualquer problema pertencente a P pertence a NP.

A questão é, P é ou não um subconjunto próprio de NP? Mais diretamente, P=NP?

Talvez a razão mais forte para que se acredite que P≠NP seja a existência de problemas NP-Completos.

P e NP



Uma possível visão se P≠NP.

Dúvidas?