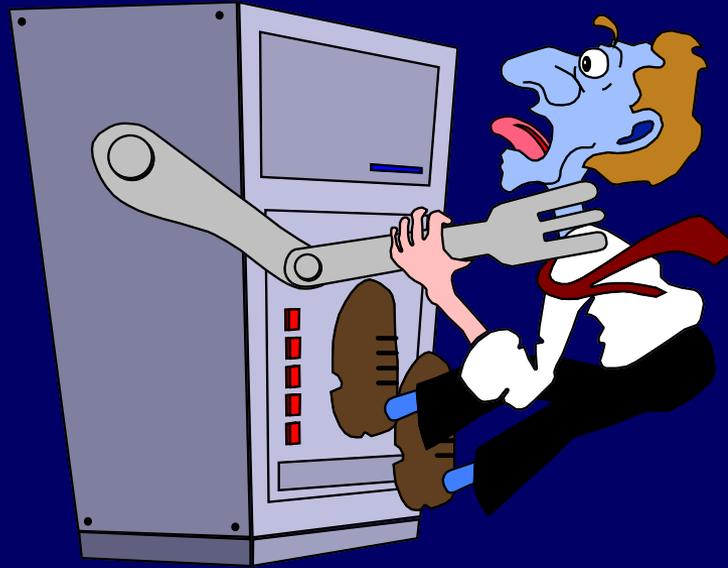


Máquinas de Turing 3



Máquinas de Turing com Múltiplas Fitas

Máquinas de Turing Não-determinísticas

A Tese/Hipótese de Church-Turing

Linguagens decidíveis por Máquinas de Turing (Recursivas)

Linguagens Aceitas/Reconhecidas por Máquinas de Turing
(Recursivamente Enumeráveis)

Usos de uma MT

- como reconhecedor de linguagens
(Visto)
- para calcular funções
- para processar problemas de decisão
(procedimento)

MT como um processador de funções inteiras

- Tradicionalmente, os inteiros são representados em unário
- O inteiro $i \geq 0$ é representado pela cadeia 0^i .
- Se a função tem k argumentos (i_1, i_2, \dots, i_k) então esses inteiros são colocados na fita separados por 1's como:

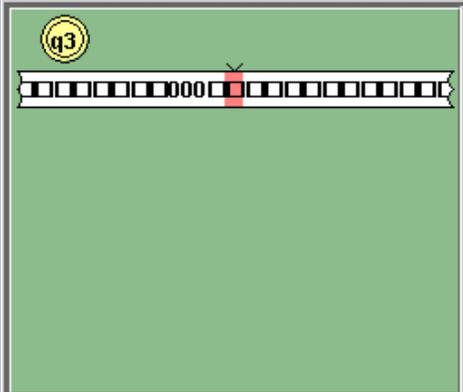
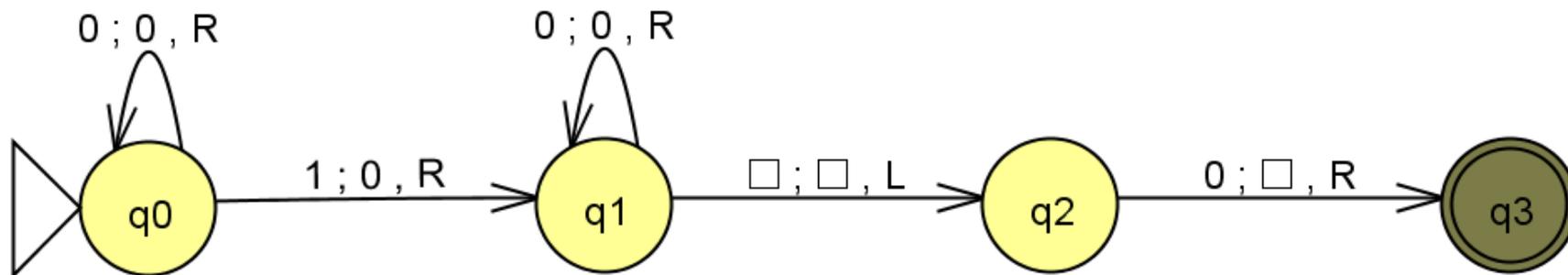
$0^{i_1} 1 0^{i_2} 1 \dots 1 0^{i_k}$

- O inverso também é possível.
- Se a máquina pára (não importa se num estado final) com a fita consistindo de 0^m para algum m então dizemos que $f(i_1, i_2, \dots, i_k) = m$, onde f é uma função de k argumentos computados por essa MT.

MT que soma dois números naturais

- Consider the addition function $a+b$ on non-negative integers
- Encode the input string on the tape as 0^a10^b
- The Turing machine will halt with 0^{a+b} on the tape

- Processo:
- Read 0's and rewrite them on the tape moving right until the 1 is read
- Replace the 1 with a 0 (SHIFT), and continue moving right without changing the tape until a blank is found
- Move left and replace the 0 with a blank

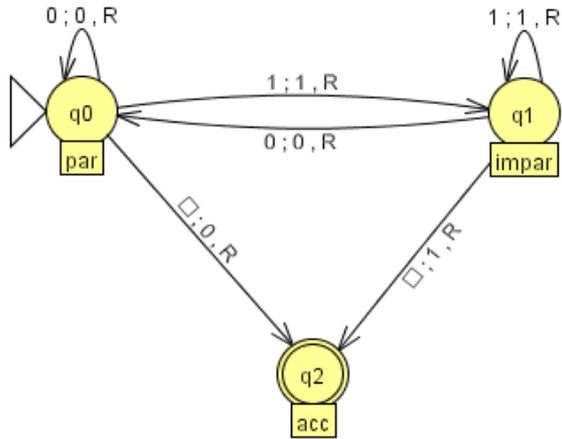


MT para processar problemas de decisão

- Quando usamos a MT para decidir (responder T/F) alguma propriedade, depois que a máquina parar, olhamos na fita a procura do resultado 0/1 após a cadeia de entrada.

MT que decide se um número binário é par ou ímpar

- Fica em q_0 enquanto par
- Fica em q_1 enquanto ímpar
- Se Par - pára e escreve 0
- Se Ímpar - pára e escreve 1



Accepting configuration found!

The diagram illustrates the sequence of configurations leading to an accepting configuration. Each configuration consists of a state and a tape. A red vertical line indicates the head's position.

- Configuration 1: State q_0 , tape $\dots 00000110000000$, head at the 11th cell.
- Configuration 2: State q_1 , tape $\dots 00000110000000$, head at the 12th cell.
- Configuration 3: State q_1 , tape $\dots 00000110000000$, head at the 13th cell.
- Configuration 4: State q_2 , tape $\dots 00000111000000$, head at the 14th cell.

Buttons:

Exercícios

- 1) Construir uma MT que decida se uma seqüência de parênteses é bem formada.
 - Escreve 0 se mal formada
 - Escreve 1 se bem formada
 - Dica: considere que a cadeia de parênteses é limitada por $<$ e $>$ para facilitar a checagem e separar a resposta, **mas não será um ALL, pois deve escrever 0/1 após as guardas.**
 - Idéia: Procurem por um $)$ e substitua por X e em seguida voltar a esquerda procurando o $($ mais próximo para substituir por X também.
- 2) Construir uma MT que dada uma cadeia w pertencente ao fecho de $\{0,1\}$ duplique w . Quando a máquina parar a fita deve conter $w\#w$ sendo que $\#$ indica fim de w .

Lembrem que:

- MT para calcular funções:
 - Se $f(n)$
 - não é definida para todo $n \rightarrow f(n)$ é **função parcial**;
 - se sempre é definida é **função total**
 - Se MT computa $f(n)$
 - para todo $n \rightarrow f(n)$ é **recursiva** (algoritmo que computa $f(n)$)
 - sempre que $f(n)$ é definida mas NÃO pára para aqueles n para os quais $f(n)$ não é definida $\rightarrow f(n)$ é **parcialmente recursiva (ou recursivamente enumerável)**
- MT para processar problemas de decisão:
 - Quando consideramos a MT como procedimento, se ela pára para toda a entrada dizemos que o procedimento é um algoritmo.

Hierarquia das Classes de Máquinas e Linguagens

L Recursivamente Enumeráveis/Máquinas de Turing que Reconhecem L

L Recursivas/Máquinas de Turing que Decidem L

L Livres de Contexto/Máquinas a Pilha não Determinísticas

L Livres de Contexto Determinísticas/
Máquinas a Pilha Determinísticas

L Regulares/Autômatos Finitos

Onde estão as LSC na figura anterior???

- As LSC são reconhecidas por MT com fita limitada, mas na Teoria da Computabilidade o foco está nas MT que decidem e nas MT que reconhecem.
- Esta é a razão de serem menos comentadas.
- As LSC são um subconjunto próprio das L Recursivas.

Máquinas e Linguagens incluindo as LSC

L Recursivamente Enumeráveis (tipo 0)/Máquinas de Turing que Reconhecem L

L Recursivas/Máquinas de Turing que Decidem L

L Sensíveis ao Contexto/Máquinas de Turing com Fita Limitada

L Livres de Contexto/ Autômatos a Pilha não Determinísticos

L Livres de Contexto Det/Autômatos a Pilha Determinísticos

L Regulares/AF

Modelos Equivalentes

- Máquinas de Turing Determinísticas (MTD) são extremamente poderosas. Nós podemos simular outros modelos com elas e vice-versa.
- As variações são equivalentes, isto é, reconhecem a mesma classe de linguagens (seja ela recursiva/re).
- Uma variação simples é a MT que tem a habilidade de permanecer parada após a leitura.
 - Como provamos que esta variação tem poder equivalente ao da MTD?
 - Lembrem que para mostrar que 2 modelos são equivalentes mostramos que simulamos um pelo outro.
- Vamos ver outro exemplo em que a simulação ocorre com somente uma perda **polinomial** de eficiência

Máquinas de Turing com múltiplas fitas

SIP 136-138



Inicialmente a entrada é escrita na primeira fita. As outras começam com branco.



As transições permitem acesso simultâneo a todas as fitas:

$$\delta: Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{L,R\}^k \quad \text{Ex: } \delta(q_i, a_1, \dots, a_k) = (q_j, b_1, b_2, \dots, b_k, L, R, L, R, \dots)_7$$

Robustez

- Máquinas de Turing com múltiplas fitas são polinomialmente equivalentes a máquinas com uma fita (que são casos especiais da primeira).
- Teo 3.8 (Sipser) Toda MT com múltiplas fitas tem uma MT equivalente com uma única fita.
 - A idéia é mostrar como simular M com k fitas em S com 1 fita.
 - S simula o efeito das k fitas armazenando suas informações em uma sua fita, separando-as com $\#$ e marca as posições de suas cabeças com símbolos marcados (a. e b., por exemplo que são acrescentados ao alfabeto da fita):
 $\#w_1.w_2\dots w_n\# \ .\# \ .\# \ \dots\#$
 - O que fazer quando a cabeça chega num $\#$?

Existem mais modelos Equivalentes?

Vamos ver um modelo computacional menos realista - MT Não-Determinísticas - que pode ser simulado com uma MT Determinística com uma perda **exponencial** de eficiência.



Computações

$$\delta(q, X) = \{(q_1, Y_1, D_1), \dots, (q_k, Y_k, D_k)\}$$

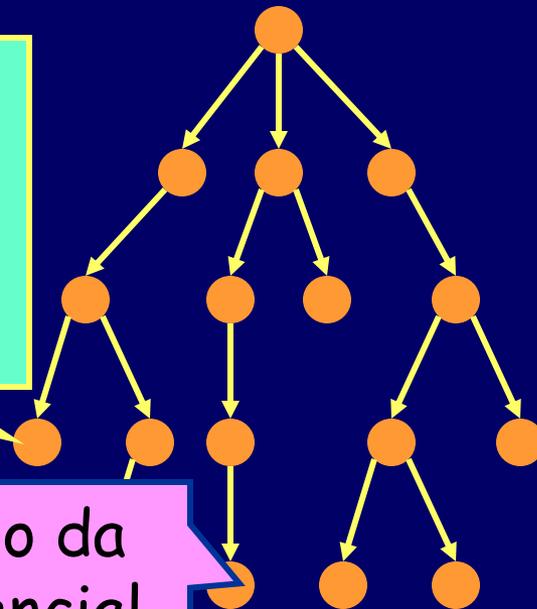
Computação
determinística

tempo



Aceita se algum ramo alcança uma configuração de aceitação

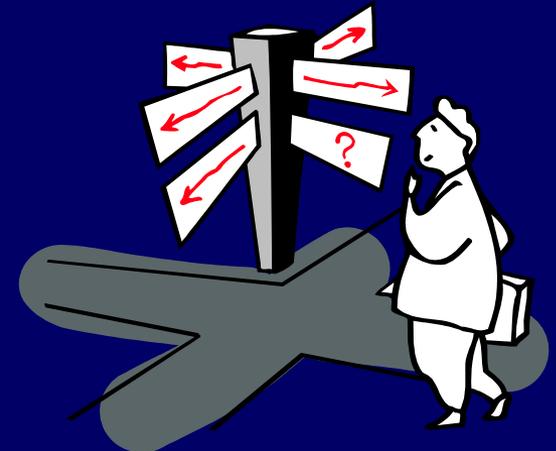
Árvore de computação
não-determinística



Nota: o tamanho da árvore é exponencial em sua altura

Descrição Alternativa

- Uma máquina não-determinística tem dois estágios:
 - um de **adivinhação** da resposta e
 - outro de **verificação**.
-
- É um modelo para capturar a noção de verificação em tempo polinomial (**NP**) e não um método para resolver problemas de decisão.



Exemplos

(1) Uma MT não-determinística que checa se 2 vértices são conectados em um grafo simplesmente "chuta" um caminho entre eles. Após isto a MT necessita somente verificar se o caminho é válido.



Lembram que vimos 2 problemas no início do curso:

- O problema da Mesa (CICLO-HAMILTONIANO)
- O problema do Pacote (CAMINHO-HAMILTONIANO)

(2) Para o problema de verificar se um grafo é hamiltoniano a MT "chuta" a lista de vértices no ciclo hamiltoniano.

Se o grafo for hamiltoniano o próprio ciclo oferece toda a informação para verificar este fato. Se o grafo não for nenhuma lista de vértices vai enganar o verificador que facilmente verifica se a lista é válida.

A Hipótese de Church-Turing

Noção Intuitiva
de algoritmo



Máquina de
Turing que
decide

Lembrem que alguns problemas são **parcialmente decidíveis**, isto é, existe uma máquina de Turing que os reconhece

- Estudar computação do ponto de vista teórico é sinônimo de caracterizar o que é ou não é computável.
- Para tanto, é preciso lançar mão de um **modelo matemático** que represente o que se entende por computação.
- Há diversos modelos (funções recursivas, algoritmos de Markov, etc.), mas iremos adotar um só: as Máquinas de Turing.

- Alonzo Church conjecturou que *todos os modelos razoáveis do processo de computação, definidos e por definir, são equivalentes (Tese de Church)*.
- Vamos primeiro trabalhar com a idéia **intuitiva** do que quer dizer computável, e para isso, introduzimos os conceitos de **procedimento e de algoritmo**. Depois voltaremos à *Tese de Church-Turing*.

Procedimentos e Algoritmos

Um procedimento (efetivo) é uma sequência **finita** de instruções, sendo uma **instrução** uma operação **claramente descrita**, que pode ser executada **mecanicamente**, (por um agente humano ou não) em **tempo finito**.

Esse conceito corresponde à noção intuitiva de "receita", "roteiro" ou "método".

Um exemplo clássico de procedimento foi inventado entre 400 e 300 D.C. pelo matemático grego Euclides para encontrar o **máximo divisor comum** entre 2 inteiros positivos.

Exemplo de Procedimento

Algoritmo de Euclides - Cálculo do máximo divisor comum (mdc) de dois inteiros positivos m e n .

- **Passo 1:** Adote como valores iniciais de x e y os valores m e n , respectivamente.
- **Passo 2:** Adote como valor de r o resto da divisão do valor de x pelo valor de y .
- **Passo 3:** Adote como o novo valor de x o valor de y , e como novo valor de y o valor de r .
- **Passo 4:** Se o valor de r é nulo, então o valor de x é o mdc procurado e o cálculo termina; caso contrário, volte a executar as instruções a partir do passo 2.

Esse exemplo ilustra as propriedades que vamos exigir de um procedimento:

- i) **Descrição Finita.** Utilizamos uma seqüência finita de palavras e símbolos para descrever o procedimento.
- ii) Todo procedimento **parte de um certo número de dados** pertencentes a conjuntos especificados de objetos (como m e n que são inteiros positivos), **e espera-se que produza um certo número de resultados** (como o valor final de x) que mantêm uma relação específica com os dados (função).
- iii) Supõe-se que exista um **agente computacional** - humano, mecânico, eletrônico, etc. - **que executa as instruções** do procedimento.

iv) **Cada instrução deve ser bem definida, não ambígua.** No exemplo, supõe-se que o agente saiba como calcular o resto da divisão inteira e haveria problemas se x e y pudessem ser inteiros quaisquer - a menos que definíssemos o que seria o resto de divisão para inteiros não positivos.

v) **As instruções devem ser efetivas,** isto é, devem ser tão simples que poderiam ser executadas, em princípio, por uma pessoa usando lápis e papel, **num espaço finito de tempo** (no exemplo, elas não o seriam caso x e y pudessem ser números reais quaisquer em representação decimal, possivelmente de comprimento infinito).

O conceito de procedimento é primitivo independentemente de sua representação.

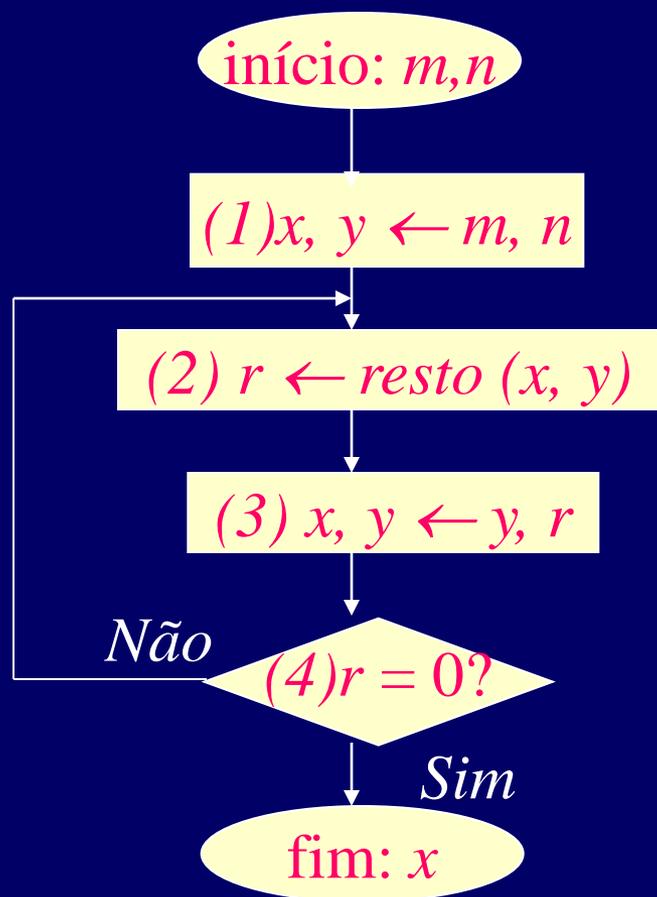
Formas de Representação de Procedimentos

- textual em língua natural
- diagrama de blocos
- pseudo-código

Exemplos sobre Término de Procedimentos

A seguir são apresentados alguns exemplos sobre a questão do término de procedimentos: como será visto, alguns procedimentos terminam quaisquer que sejam os valores dos dados de entrada e outros terminam apenas para alguns valores.

EXEMPLO 1 - Algoritmo de Euclides: Calcula o máximo divisor comum entre dois inteiros positivos m e n



Pergunta: Este procedimento termina quaisquer que sejam os valores dos dados de entrada?

Mostrar isto, neste exemplo, equivale a provar a seguinte proposição:

"Se no passo 2 do procedimento os valores de x e y são inteiros e positivos, então os passos 2, 3 e 4 serão executados apenas um número finito de vezes, com os cálculos terminando no passo 4".

Demonstração por **indução** sobre o valor de y :

se $y = 1$, então após o passo 2, $r = 0$. Portanto, os passos 2, 3 e 4 são executados uma única vez e o cálculo termina no passo 4.

- Suponhamos que a proposição é verdadeira para qualquer $x > 0$ e qualquer y , com $1 \leq y < k$, e demonstraremos que ela é verdadeira para $y = k$.

- Por definição do resto da divisão de inteiros positivos, teremos, se $y = k$, após a execução do passo 2, $0 \leq r < k$. Se $r = 0$, então a execução termina, numa única vez. Se $r > 0$, com a execução dos passos 3 e 4, (continua >>>)

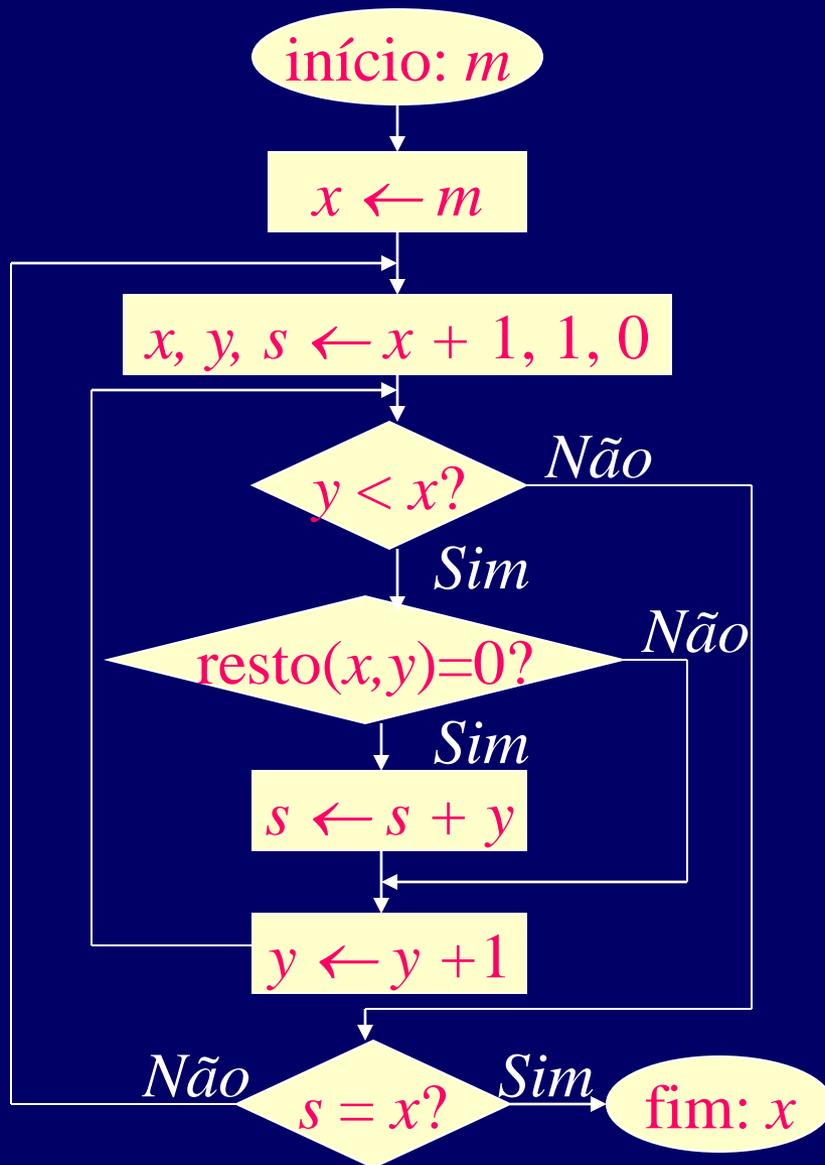
teremos $x = k > 0$ e $y = r$ com $0 < r < k$, e a execução volta ao passo 2. Por hipótese de indução, os passos 2, 3 e 4 serão executados um número finito p de vezes, com os cálculos terminando no passo 4. Ao todo teremos, então, $p+1$ execuções para $y = k$. Notemos, ainda, que os valores iniciais $x = m$ e $y = n$ resultantes da execução do passo 1 satisfazem as condições da proposição acima (i.e. inteiros positivos).

• Conclui-se, então, que o **Algoritmo de Euclides termina para quaisquer inteiros positivos m e n .**

EXEMPLO 2: Procedimento para determinar o menor número perfeito que é maior do que um inteiro positivo m dado (k é perfeito se for igual à soma de todos os seus divisores, exceto o próprio k).

Em outras palavras, dado m , deseja-se obter o primeiro número perfeito maior do que m .

Idéia adotada: a partir de $x = m + 1$, de um em um, verifica-se se x é número perfeito, calculando-se e somando-se todos seus divisores.



Pergunta: Este procedimento sempre termina?

Resposta: Apenas para certos valores de m .

Por exemplo, se $m = 4$, então ele pára com $x = 6$, pois $5 \neq 1$ e $6 = 1 + 2 + 3$.

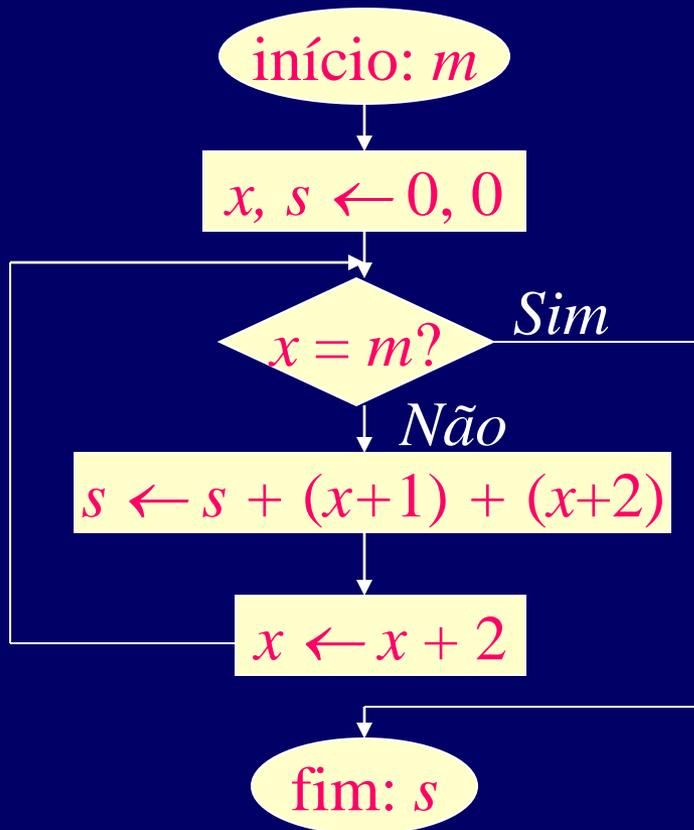
Porém, no caso geral, a resposta não é conhecida, pois a existência ou não de um número infinito de números perfeitos é um problema em aberto.

Se existirem infinitos números perfeitos, então a execução do procedimento termina para qualquer m ; caso contrário, se K é o maior número perfeito, então o procedimento executa uma sequência infinita de cálculos para todo $m \geq K$.

Conjecturas ainda não demonstradas

- O n -ésimo número perfeito tem n dígitos;
 - Todos os números perfeitos são pares;
 - Todos os números perfeitos terminam em 6 ou em 8, alternadamente;
- NP menores que 10.000 = 6, 28, 496, 8128 --

EXEMPLO 3: Cálculo de $s = \sum_0^m i$ com m inteiro positivo.



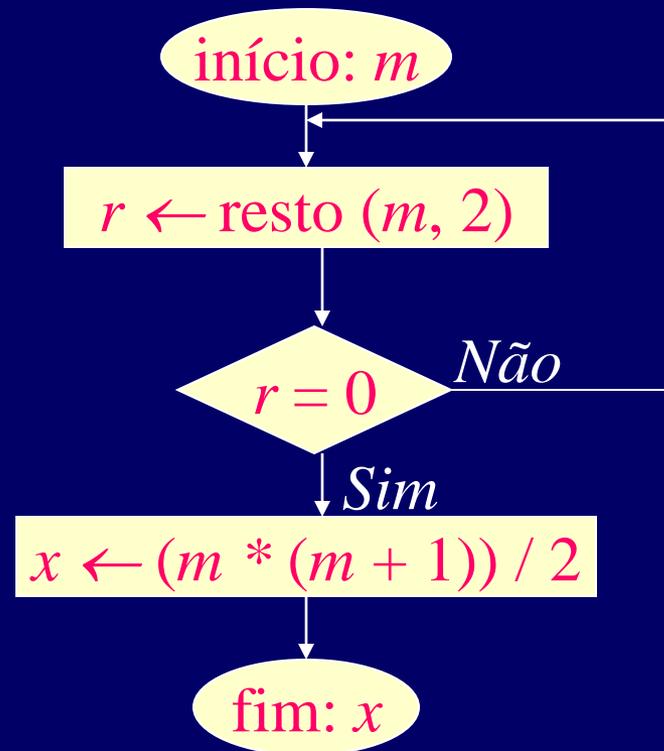
Pergunta: Este procedimento sempre pára?

Resposta: Não. Termina apenas para valores pares de m , pois os valores da sequência são $0, 2, 4, 6, \dots$. Para valores ímpares, a igualdade $x = m$ nunca é verdadeira e a execução do procedimento não termina.

Note-se que todo procedimento é um método para o cálculo de alguma função, **eventualmente não definida para certos argumentos**. O exemplo 3 corresponde à função h que pode ser descrita como:

$$h(m) = \begin{cases} \Sigma_0^m I & \text{se } m \text{ é par} \\ \text{não definida} & \text{se } m \text{ é ímpar} \end{cases}$$

Por outro lado, uma mesma função pode ser calculada por vários procedimentos distintos. É o caso do procedimento abaixo para o cálculo da mesma função $h(m)$:



Temos interesse especial nos procedimentos cuja execução termina para quaisquer valores dados.

Algoritmos

Def.: Algoritmos são procedimentos cuja execução termina para quaisquer valores dos dados de entrada.

- 1) O Algoritmo de Euclides é realmente um algoritmo;
- 2) Nada podemos afirmar sobre o procedimento que determina o menor número perfeito maior do que um inteiro positivo m dado;
- 3) O procedimento que efetua o cálculo do somatório não é um algoritmo.

Decidir se um procedimento é um algoritmo não é tarefa trivial!

Caso contrário já saberíamos a resposta de várias conjecturas tais como a existência de um número infinito de números perfeitos e outras questões abertas da matemática.

Programas e Linguagens de Programação

Vimos que um procedimento pode ser especificado por uma mistura de palavras e símbolos como fizemos para o algoritmo de Euclides, mas para ser executado em um computador usamos uma **linguagem de programação (LP)**.

Def: Uma **LP** é definida por um conjunto de símbolos sobre um *alfabeto* (**léxico**), e um conjunto de regras que especificam como compor esses símbolos (**sintaxe**) e quais são as ações associadas a elas (**semântica**).

Def: Um **programa** é uma sequência de símbolos de uma LP que representa um ou mais procedimentos.

As LP têm características variadas, dependendo de sua finalidade.

Há desde as muito simples, porém de grande interesse para a teoria da computação (ex. a **linguagem da Máquina de Turing**), como as de mais alto nível, usadas para propósito geral.

Toda LP é capaz de representar todos os procedimentos efetivos, isto é, é universal?

Tese (hipótese) de CHURCH-TURING

Qualquer procedimento pode ser representado em Linguagem de Turing (analogamente, pode ser computado por uma Máquina de Turing - MT).

Em primeiro lugar, *essa tese não pode ser demonstrada, devido à noção intuitiva de procedimento, por isto é mais adequado dizer que é uma hipótese.* Uma maneira de negar a tese é encontrar um procedimento que não pudesse demonstradamente ser computado por uma Máquina de Turing.

Isso não ocorreu, e devido ao grande número de dados experimentais, esta tese tem sido aceita pelos cientistas da computação.

Um outro fato notável que suporta a Tese de Church é que as várias tentativas independentes de formalizar o conceito de procedimento efetivo resultaram todos em formalismos demonstradamente **equivalentes** ao de Turing.

Em segundo lugar, para responder a pergunta acima, bastaria estabelecer a equivalência entre a LP dada e a Linguagem de Turing. Uma vez equivalentes, a universalidade da LP estaria garantida.

Embora tedioso, esse processo é perfeitamente possível:

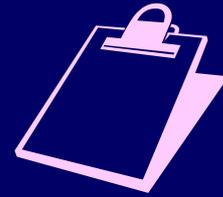
(a) construir MT que simulem o comportamento de programas em LP;

(b) construir programas em LP que executam a mesma função de um MT qualquer.

Isso é demonstrado, em geral, através do uso de uma linguagem de complexidade intermediária, LI, entre MT e LP, de tal forma que se estabelece a equivalência entre MT e LI, e entre LI e LP.

Na verdade, a exigência para esta equivalência é que a LP contenha um conjunto mínimo de operações primitivas, como soma, subtração, teste de zero e repetição. Como as LP contêm um conjunto muito mais abrangente, a parte (b) acima é bastante facilitada.

Sumário



- Apresentamos os dois modelos principais de computação: MT Determinística e MT Não-determinística.
- A MTD pode ser simulada por uma MTND com uma perda exponencial de eficiência.
- Por isto que dizemos que a classe NP é a classe de todos os problemas de decisão "solúveis" por algoritmos não-determinísticos de tempo polinomial.
- N = não-determinístico P = polinomial
- A verificação é em tempo polinomial pois **não** contamos o tempo de adivinhação, que será exponencial!

Sumário



- A Hipótese de Church-Turing: MT Determinísticas que decidem são equivalentes a nossa noção intuitiva de algoritmos.
- Sabendo disto, podemos descrever usualmente algoritmos em pseudo-código ao invés de escrever em MT.