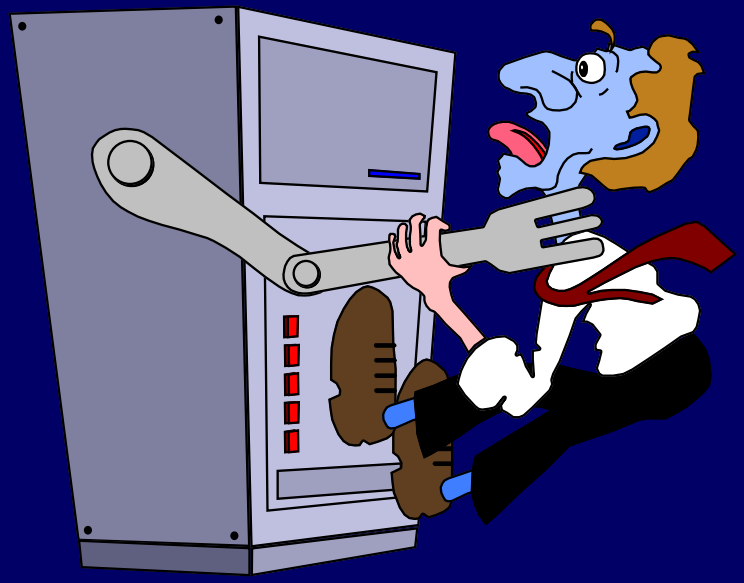
Máquinas de Turing



Técnicas de Extensão de MT

Permanecer parada após a leitura (STILL)

Máquinas de Turing com Múltiplas Fitas e Cabeças

Máquinas de Turing Não-deterministicas

A Tese/Hipótese de Church-Turing

Linguagens decidíveis por Máquinas de Turing (Recursivas)

Linguagens Reconhecidas por Máquinas de Turing (Recursivamente Enumeráveis)

Modelos Equivalentes

- A fim de demonstrar, empiricamente, o poder computacional das MT, existem técnicas de combinação e extensão das MT.
  - Dessa forma, o modelo de Turing vai ganhando recursos que facilitam a construção de MT para funções mais complexas.

- O ponto importante, no entanto, é que todas essas extensões comprovadamente **NÃO** alteram o conjunto de funções computáveis por MT.

- Isto é, o poder computacional das MT permanece inalterado com o acréscimo de todos esses recursos.

- Isso traz evidências para a Tese de Church-Turing: *uma função é computável se e somente se ela for MT-computável*

Variação: permanecer parada após leitura

- 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.

- Solução: caminhar para esquerda e depois direita ou vice-versa

- 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:

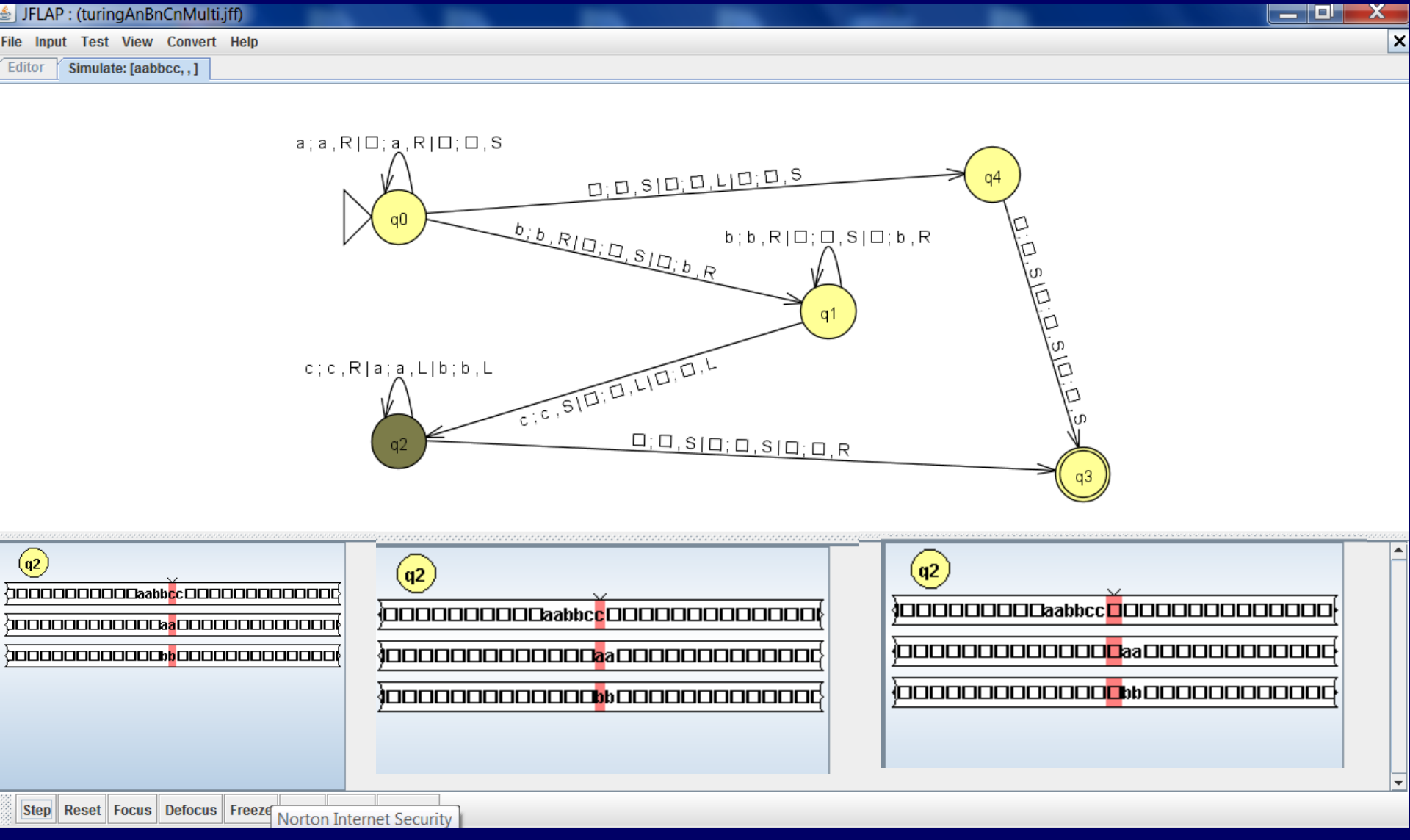
δ: Q X Γ^k -> Q X Γ^k X $\{L,R\}^k$ Ex: $\delta(qi,a1,\ldots,ak) = (qj,b1,b2,\ldots,bk,L,R,L,R\ldots)$

Exemplo

- $L = \{a^n b^n c^n \mid n >= 1\}$

- Vamos usar uma MT com 3 fitas:
  - Entrada na fita 1
  - Contagem dos a´s na fita 2
  - Contagem dos b´s na fita 3
  - Os c´s são contados se houver o mesmo número de a´s e b´s

Exemplo


JFLAP : (turingAnBnCnMulti.jff)
File Input Test View Convert Help
Editor
Simulate: [aabbcc, ,]
a ; a , R | □ ; a , R | □ ; □ , S
□ ; □ , S | □ ; □ , L | □ ; □ , S
b ; b , R | □ ; □ , S | □ ; b , R
b ; b , R | □ ; □ , S | □ ; b , R
c ; c , R | a ; a , L | b ; b , L
c ; c , S | □ ; □ , L | □ ; □ , L
□ ; □ , S | □ ; □ , S | □ ; □ , R
□ ; □ , S | □ ; □ , S | □ ; □ , S
q0
q1
q2
q3
q4
q2 aabbcc
q2 aa
q2 bb
q2 aabbcc
q2 aa
q2 bb
q2 aabbcc
q2 aa
q2 bb
Step
Reset
Focus
Defocus
Freeze
Norton Internet Security

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 acrescidos ao alfabeto da fita):

    #w1.w2…wn# .# .# ….#

  - 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.

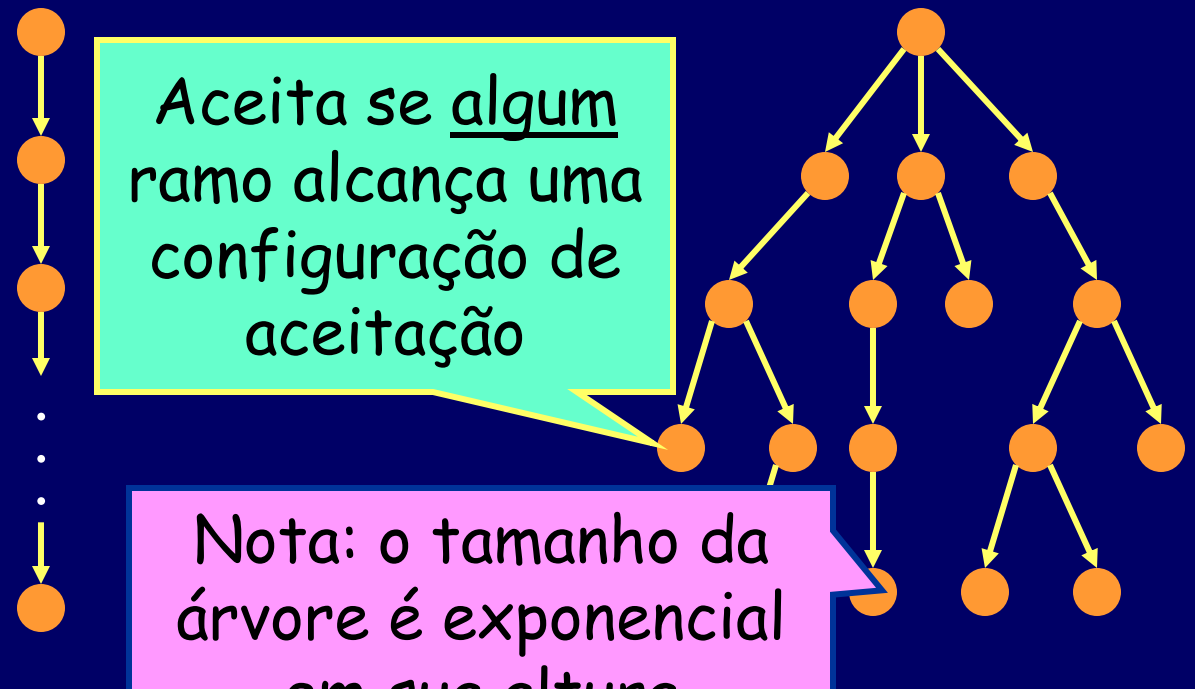
Computações

$\delta(q,X) = \{(q1,Y1,D1),\ldots,(qk,Yk,Dk)\}$

Computação determinística

Árvore de computação não-determinística

tempo



- A linguagem aceita por uma MTND é dada pelo conjunto de cadeias para as quais haja **ao menos uma sequência** de escolhas de movimentos que **leve do estado inicial a um um estado final**.
  - Ou seja, se por um dos caminhos possíveis, terminar em estado final, então a cadeia é aceita.

- É como se a MTND tentasse, em paralelo, reconhecer a cadeia, por diferentes caminhos.

MTND X MTD

- Embora, às vezes, seja mais fácil pensar num algoritmo não determinístico:

- **Teorema:** Se M_N é uma MTND, então existe uma MT determinística M_D tal que $L(M_N) = L(M_D)$.

- Ou seja, as MTND não aceitam nenhuma linguagem que não seja aceita por uma MTD.

Simulando uma MTND por uma MTD

- Uma MTD equivalente a uma MTND qualquer trabalharia da seguinte forma:
  - Se m é o número total de caminhos possíveis da MTND, então a MTD tentaria cada caminho c_i, $1 \leq i \leq m$, até que um deles terminasse num estado final; então a MTD também entraria num estado de aceitação.
  - Se m for finito e nenhum caminho levar a um estado final, a MTD rejeita a cadeia após verificar todos os candidatos.

Complexidade das MTND

- Seja m o número total de caminhos possíveis a serem verificados (repare que, a cada transição, pode haver um número arbitrário de possibilidades).

- Todos os caminhos são verificados simultaneamente.

- Como a MT para se (e quando) um dos caminhos resultar em sucesso, o tempo que terá gasto dependerá apenas de quanto o caminho de sucesso demorar. Ou seja, independe de m, dependendo apenas da entrada, n.

Complexidade das MTND X MTD

- Mas, se construirmos uma MTD no lugar da MTND, conforme explicado antes, o tempo gasto será dominado pelo número de caminhos que ela deverá verificar.

- Seja k o número máximo de movimentos possíveis a cada transição.
  - Assim, na primeira transição, gera-se k caminhos, na segunda, k novos para cada um anterior, ou seja, k^2 caminhos.
  - Após n transições, haverá $1 + k + k^2 + ... + k^n$ possibilidades.
  - Essa soma é no máximo nk^n, portanto, exponencial em n.

- Assim, embora as MTND sejam equivalentes às MTD, estas últimas (simuladas dessa forma) podem demorar exponencialmente mais tempo que a MTND.

Descrição Alternativa

- Uma máquina não-determinística tem dois estágios:
- um de advinhação 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 algorítmo ≡ 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.

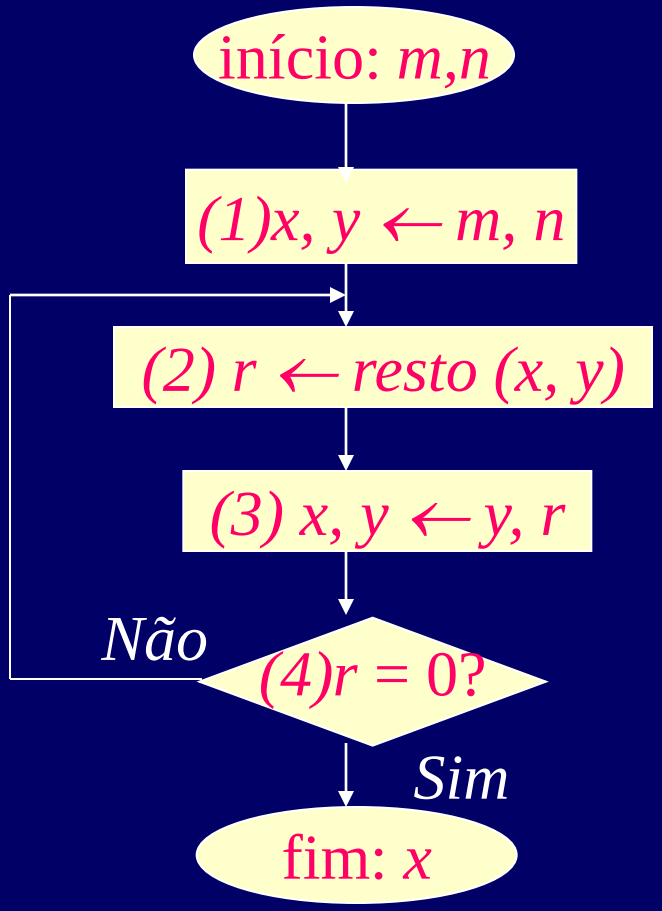
Formas de Representação de Procedimentos

- **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 >>>)

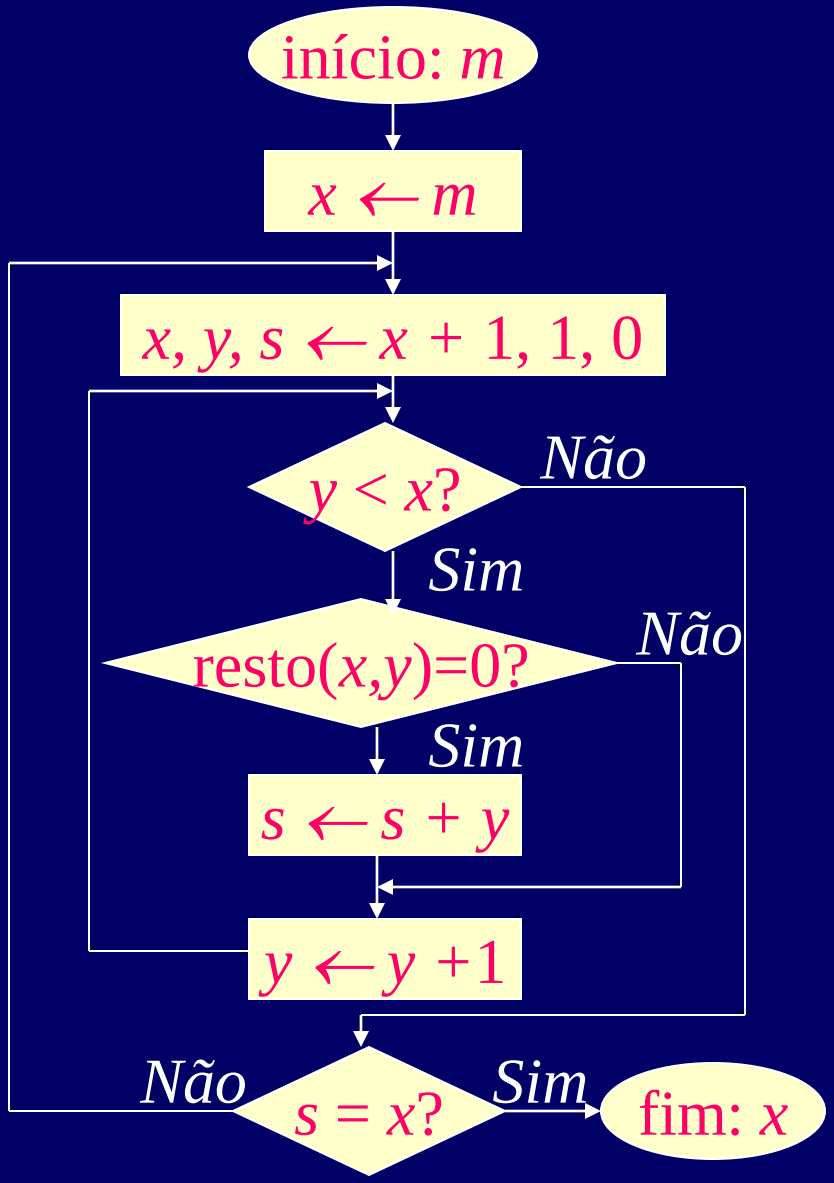
teremos x = k > 0 e y = r com 0<r<k, e a execução volta ao passo 2. Por hipótese de indução, os passos 2, 3 e 4 serão executados um número finito p de vezes, com os cálculos terminando no passo 4. Ao todo teremos, então, p+1 execuções para y = k. Notemos, ainda, que os valores iniciais x = m e y = n 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≠ 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.

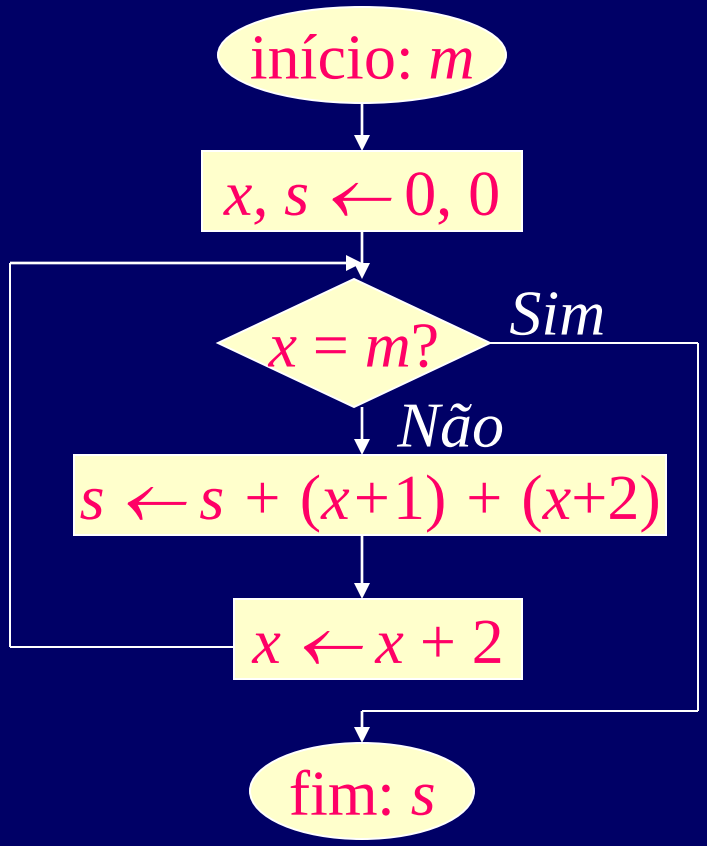
Se existirem infinitos números perfeitos, então a execução do procedimento termina para qualquer m; caso contrário, se K é o maior número perfeito, então o procedimento executa uma sequência infinita de cálculos para todo m ≥ K.

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.



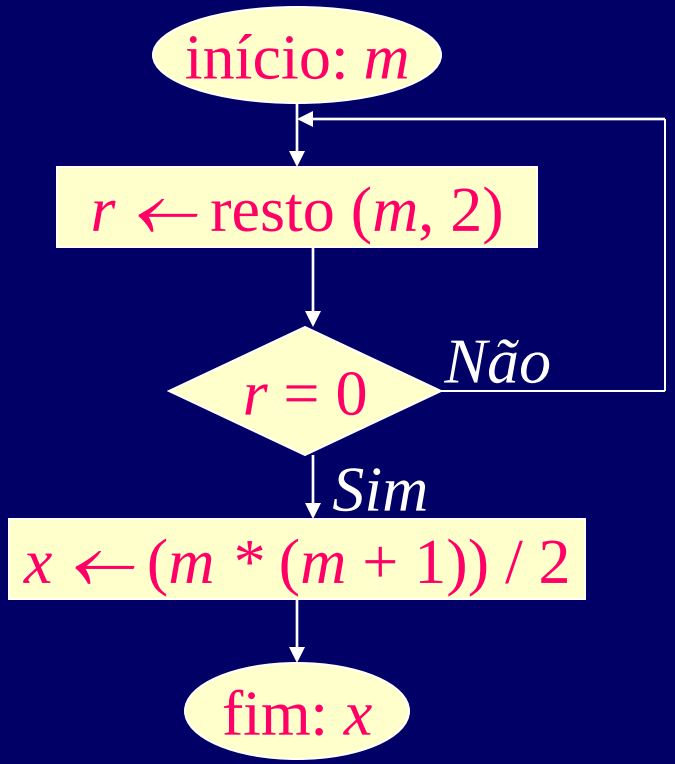
Pergunta: Este procedimento sempre pára?

Resposta: Não. Termina apenas para valores pares de m, pois os valores da sequência são 0, 2, 4, 6,... 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:

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

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;

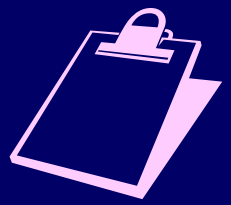
2) Nada podemos afirmar sobre o procedimento que determina o menor número perfeito maior do que um inteiro positivo *m* dado;

3) O procedimento que efetua o cálculo do somatório não é um algoritmo.

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

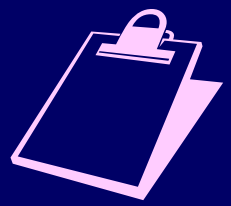
Caso contrário já saberíamos a resposta de várias conjecturas tais como a existência de um número infinito de números perfeitos e outras questões abertas da matemática.

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 Deterministicas que decidem são equivalentes a nossa noção intuitiva de algorítmos.

- Sabendo disto, podemos descrever usualmente algoritmos em pseudo-código ao invés de escrever em MT.