Conceitos Básicos

Procedimentos e Algorítmos
Programas e Linguagens de Programação
Tese de Church-Turing
Formas de Representação de Linguagens

Introdução

- 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).*

- Antes de definir o modelo da máquina de Turing, vamos trabalhar com a idéia intuitiva do que quer dizer computável, e para isso, introduzimos os conceitos de *procedimento* e de *algoritmo*.

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.

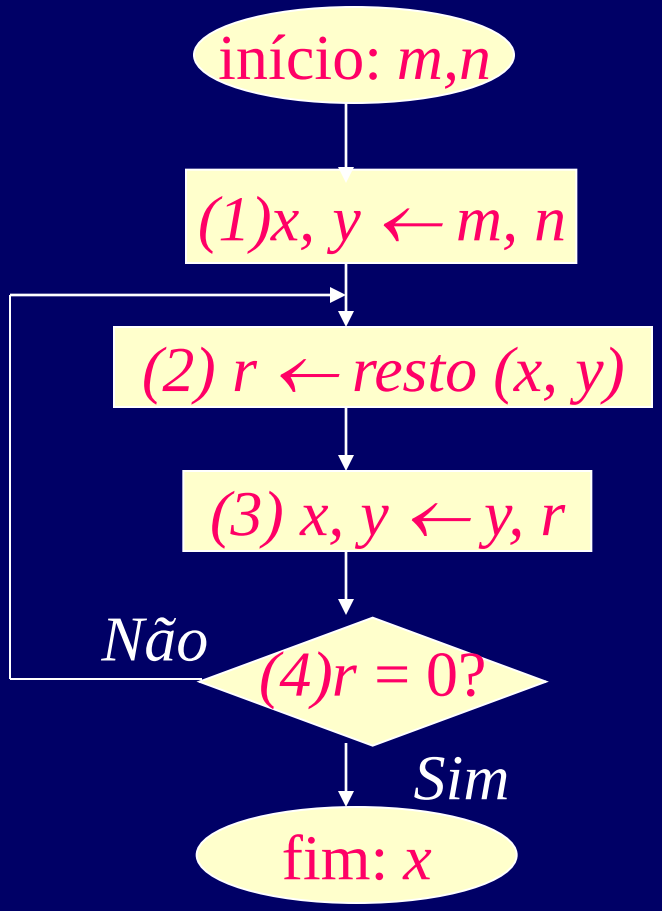
Formas de Representação de Procedimentos

- **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 >>>)

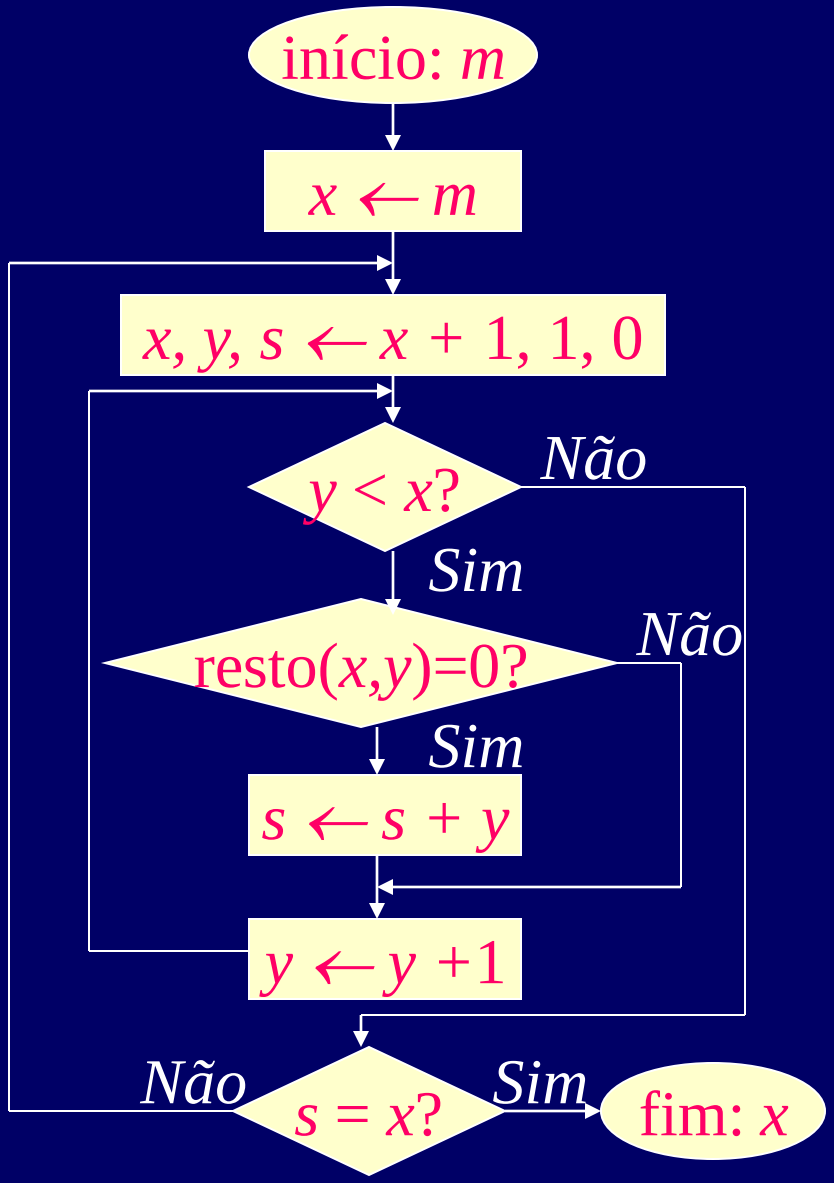
teremos $x = k > 0$ e $y = r$ com $0<r<k$, e a execução volta ao passo 2. Por hipótese de indução, os passos 2, 3 e 4 serão executados um número finito p de vezes, com os cálculos terminando no passo 4. Ao todo teremos, então, p+1 execuções para $y = k$. Notemos, ainda, que os valores iniciais $x = m$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

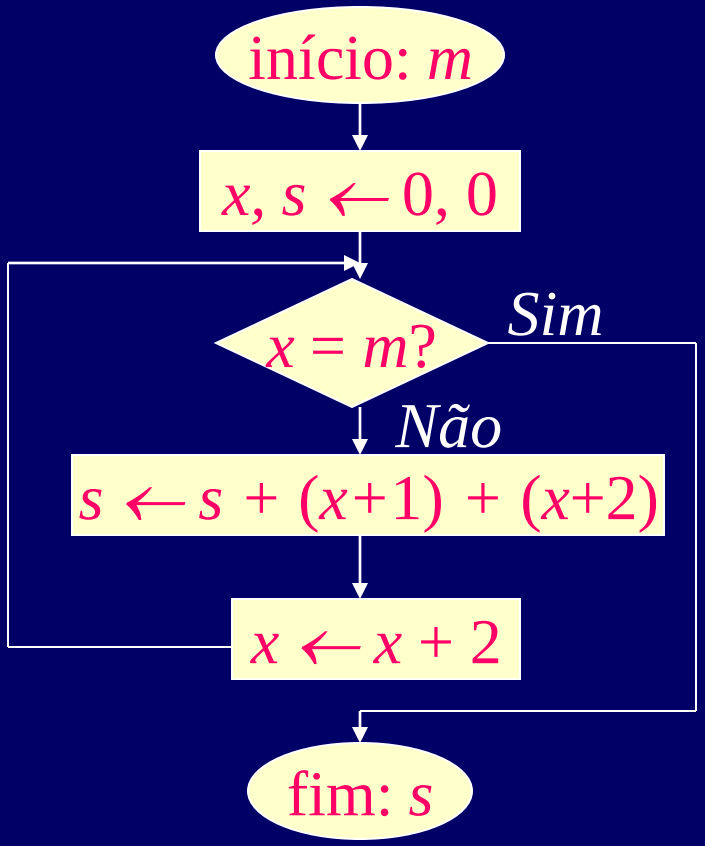
Se existirem infinitos números perfeitos, então a execução do procedimento termina para qualquer m; caso contrário, se K é o maior número perfeito, então o procedimento executa uma sequência infinita de cálculos para todo m ≥ K.

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.



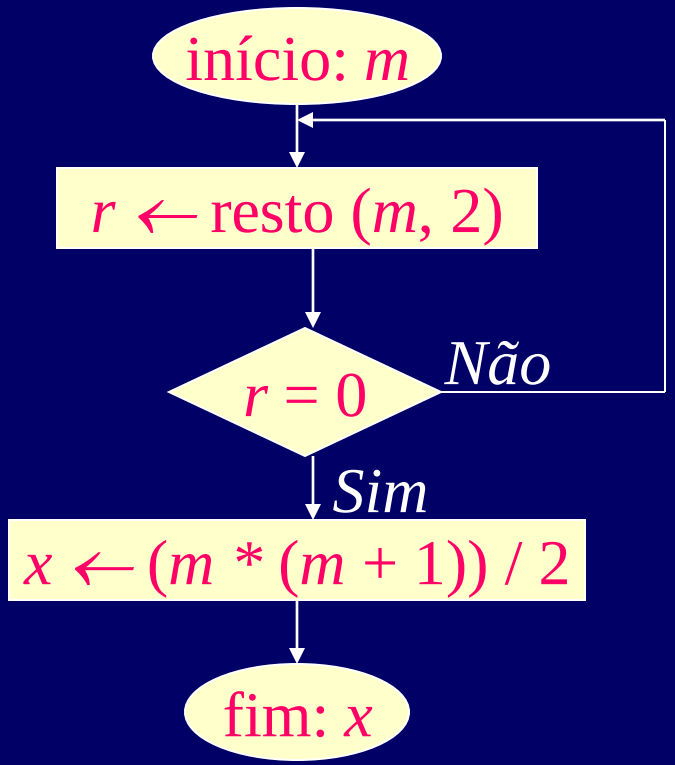
Pergunta: Este procedimento sempre pára?

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.

Voltaremos ao problema da parada de procedimentos mais para frente.

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 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. 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.

Formas de representação de linguagens

i) enumeração – linguagens finitas

ii) axiomático – regras de formação/geração de cadeias.
Veremos as gramáticas da hierarquia de Chomsky.

iii) operacional – máquina abstrata, também chamada de reconhecedor (ou regras de aceitação de cadeias).
Veremos autômato finito, autômato a pilha, MT.

iv) denotacional. Veremos as expressões regulares.

Definimos formalmente os conceitos de linguagem e gramática

Para as linguagens regulares vimos também o formalismo denotacional – expressões regulares e existem utilitários que as exploram (lex, grep).

Enumeração

Enumeração das cadeias de símbolos que formam as suas sentenças: todas as sentenças da linguagem aparecem explicitamente na enumeração, e a decisão acerca da pertinência ou não de uma cadeia à linguagem se faz por meio de uma busca da cadeia em questão no conjunto de sentenças da linguagem;

Exemplo: L = {a, b, ab, ba} (linguagem formada pelas cadeias a, b, ab e ba)

Leis de Formação

Através de um conjunto de leis de formação das cadeias (ao conjunto de leis de formação dá-se o nome de Gramática):

dada uma cadeia de símbolos, só é possível afirmar que tal cadeia pertence à linguagem se for possível, aplicando-se as leis de formação que compõem a gramática da linguagem, derivar (sintetizar) a cadeia em questão.

Ao processo de obtenção de uma sentença a partir da gramática dá-se o nome de derivação da sentença.

Regras de aceitação de cadeias

Através de regras de aceitação ou reconhecimento de cadeias (reconhecedor ou autômato):

para decidir se uma cadeia é uma sentença da linguagem, basta aplicar a ela as regras de aceitação, as quais deverão fornecer a decisão acerca da pertinência da cadeia à linguagem.