2. Algoritmos

Prof. Renato Tinós

Local: Depto. de Computação e Matemática (FFCLRP/USP)

Principais Tópicos

2. Algoritmos

2.1. Introdução
2.2. Pseudo-código
2.3. Fluxograma
2.4. Projeto

2.1. Introdução

- **Computadores são ferramentas poderosas**
  - Podem armazenar e processar uma enorme quantidade de informação a uma velocidade impressionante
  - No entanto, são fortemente dependentes
    - » necessitam que digam o que eles devem fazer e como devem solucionar os problemas

2.1. Introdução

- **Algoritmo é a descrição de um conjunto de comandos que, quando obedecidos, resultam em uma sucessão finita de ações**
  - Geralmente, se destina a especificar a solução para a resolução de um problema através de uma seqüência ordenada, finita e sem ambigüidades de ações
  - Diferentemente do programa, que é escrito através de uma linguagem de computação específica, o algoritmo é escrito em linguagem natural
    - » Portanto, é menos formal

2.1. Introdução

- **Exemplos de algoritmos podem ser encontrados facilmente na vida quotidiana**
  - Instruções para o uso de eletrodomésticos
  - Receitas em culinária
  - Instruções para preenchimento de um formulário
  - Descrição do conjunto de passos para a solução de um problema matemático
  - etc...

2.1. Introdução

- **Exemplo de algoritmo: receita culinária**

1. Coloque o macarrão em uma panela com água quente
2. Coloque a manteiga e o azeite em uma frigideira
3. Frite em fogo brando as tiras de carne até que estejam bem passadas
4. Junte as vagens e deixe tostar
5. Se as vagens estiverem tostadas, desligue a frigideira
6. Se o macarrão estiver cozido, desligue a panela
7. Junte o macarrão e o conteúdo da frigideira
8. Sirva enquanto estiver quente

2.1. Introdução

- **Um algoritmo correto deve possuir 3 qualidades**

  1. Cada passo do algoritmo deve ser uma instrução que possa ser realizada de maneira inequívoca
  2. A ordem dos passos deve ser precisamente determinada
  3. O algoritmo deve ter fim

2.1. Introdução

- **Um algoritmo é considerado completo se**
  - os seus comandos puderem ser entendidos de maneira inequívoca
  - sua seqüência puder ser executada de maneira inequívoca
- **Caso contrário, os comandos devem ser desdobrados em novos comandos, processo que é chamado de refinamento do comando inicial**

2.1. Introdução

- **Considere, por exemplo, um algoritmo que escreve os termos da seqüência de Fibonacci inferiores a um número *L***
  - a seqüência de Fibonacci é definida como tendo os dois primeiros termos iguais a 1 e cada termo seguinte igual à soma dos dois termos imediatamente anteriores

Algoritmo Fibonacci
Início
escreva os termos de Fibonacci inferiores a L
Fim

2.1. Introdução

- **Observe que o comando principal do algoritmo Fibonacci é muito vago**
- **Assim, um refinamento faz-se necessário**

Algoritmo Fibonacci
Início
    receba o valor de L
    processe os dois primeiros termos
    processe os termos restantes
Fim

2.1. Introdução

- **Um algoritmo é formado por**
  - **Comandos**
    - » Definem as ações executadas
  - **Estruturas de Controle**
    - » Definem se os comandos devem ou não ser executados
    - » Definem a ordem em que os comandos devem ser executados
    - » Definem se os comandos devem ser repetidos
- **No algoritmo anterior, vigora a mais simples estrutura de controle: a estrutura seqüencial**
  - Especifica que os comandos devem ser executados seqüencialmente

2.1. Introdução

- **Já que os comandos do algoritmo Fibonacci são ainda vagos, pode-se desdobrá-los e representá-los de modo mais refinado**
- **As duas últimas linhas de comando do algoritmo Fibonacci serão representadas a seguir por refinamentos**

2.1. Introdução

Refinamento processe os dois primeiros termos
Início
    atribua o valor 1 ao primeiro termo
    se ele for menor que L **então**
        escreva-o
    fim se
    atribua o valor 1 ao segundo termo
    se ele for menor que L **então**
        escreva-o
    fim se
Fim

2.1. Introdução

- **Nota-se, neste refinamento, a presença de outra estrutura de controle: a estrutura condicional**

```
se condição então
        comandos
fim se
```

- **A estrutura condicional especifica a execução ou a não-execução de um comando, ou conjunto de comandos**

2.1. Introdução

Refinamento processe os termos restantes

```
Início
        repita
                calcule novo termo somando os dois anteriores
                se novo termo for maior ou igual a L então
                        interrompa
                fim se
                escreva novo termo
        fim repita
Fim
```

2.1. Introdução

- **Existe, neste refinamento, uma outra estrutura de controle: a estrutura de repetição**
- **Na estrutura de repetição, os comandos e as estruturas de controle devem ser executadas repetidamente até que uma condição seja satisfeita**

2.1. Introdução

Algoritmo Fibonacci
Início
receba o valor de L
atribua o valor 1 ao primeiro termo
se ele for menor que L **então**
escreva-o
fim se
atribua o valor 1 ao segundo termo
se ele for menor que L **então**
escreva-o
fim se
repita
calcule novo termo somando os dois anteriores
se novo termo for maior ou igual a L **então**
interrompa
fim se
escreva novo termo
fim repita
Fim

2.1. Introdução

- **Problemas computacionais complexos requerem o desenvolvimento de algoritmos grandes e/ou complexos**
  - Algoritmos grandes e/ou complexos podem ser difíceis de serem compreendidos e analisados
- **Algoritmos Estruturados**
  - As técnicas de desenvolvimento estruturado de algoritmos surgiram para sistematizar e auxiliar o desenvolvimento de algoritmos para a resolução de problemas computacionais grandes e complexos

2.1. Introdução

- **Os objetivos do desenvolvimento estruturado são**
  - Facilitar o desenvolvimento de algoritmos
  - Facilitar o seu entendimento por seres-humanos
  - Antecipar a comprovação de sua correção
  - Facilitar sua manutenção e modificação
  - Permitir o seu desenvolvimento simultâneo por uma equipe de pessoas

2.1. Introdução

- **Para isso, é necessário que**

  a) Os algoritmos sejam desenvolvidos por refinamentos sucessivos

  b) Os sucessivos refinamentos sejam módulos que delimitam poucas funções

  c) Os módulos usem um número limitado de diferentes comandos e de diferentes estruturas de controle

2.2. Pseudo-código

- **Existem diversas técnicas para a representação de algoritmos**
  - Pseudo-código
  - Fluxograma
  - Etc...

2.2. Pseudo-código

- **Pseudo-Código**
  - Descrição em alto nível que combina
    - Linguagem natural
    - Estruturas de linguagem de programação
  - Escrito para ser interpretado por um ser-humano, e não por um computador
  - Mais compacto que o código na linguagem de programação
  - Facilita a análise dos algoritmos

2.2. Pseudo-código

```
Algoritmo Maximo_Elemento
Início
        declare A[10], maximo, i
        leia (A[1])
        maximo ← A[1]
        para i ← 2 até 10 faça
                leia (A[i])
                se A[i] > maximo então
                        maximo ← A[i]
                fim se
        fim para
        escreva (maximo)
Fim
```

2.2. Pseudo-código

- **Variáveis e constantes**
  - Variável: valor pode ser modificado durante a execução do algoritmo
  - Constante: valor não muda durante a execução do algoritmo
  - Variáveis e constantes precisam ser declaradas
- **Tipos de dados mais utilizados**
  - Numéricos
    - » Inteiros
      - Exemplos: 9 , 5 , 0 , -3
    - » Reais
      - Exemplos: 3.14 , 0.5 , -10.12334
  - Literais ou caracteres
    - » Exemplos: “a” , “c” , “aluno” , “numero_1”

2.2. Pseudo-código

- **Estruturas de linguagem de programação utilizadas**
  - **Expressões**
    - » Atribuição: ←
    - » Relação de igualdade: =
  - **Comandos de entrada e saída:** leia(...), imprima(...)
    - » Importante: a leitura e escrita devem ser feitas para cada variável ou elemento de vetor
  - **Declarações de Rotinas:** Algoritmo (par1, ...)
  - **Chamadas de sub-rotinas:** subrotina (par1, ...)
  - **Retorno de sub-rotinas:** retorne... (em inglês: *return*...)

2.2. Pseudo-código

- **Estruturas de Decisão:** se...então...senão... (em inglês: *if...then...else...*)
- **Estruturas de Repetição:**
  - » enquanto...faça... (em inglês: while...do...)
  - » repita...até que... (em inglês: repeat...until...)
  - » para...faça... (em inglês: for...do...)
- **Indexação:** *A* [*i*]

2.3. Fluxograma

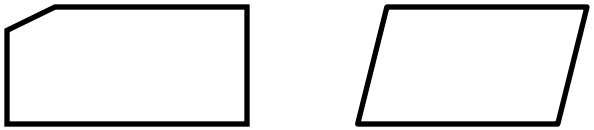
- **Fluxograma**
  - Descrição gráfica que utiliza símbolos pré-definidos
  - Facilita o entendimento da seqüência de passo do algoritmo
  - Pode tornar-se complexo demais quando o algoritmo é grande
  - Outro problema é que, por permitir uma ampla liberdade na representação, pode ocorrer a utilização incorreta das estruturas de controle

2.3. Fluxograma

- Indica o início e o fim do algoritmo

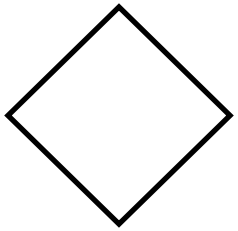
- Indica o sentido do fluxo de dados, sendo utilizado para conectar os demais símbolos

- Cálculos e atribuições de valores



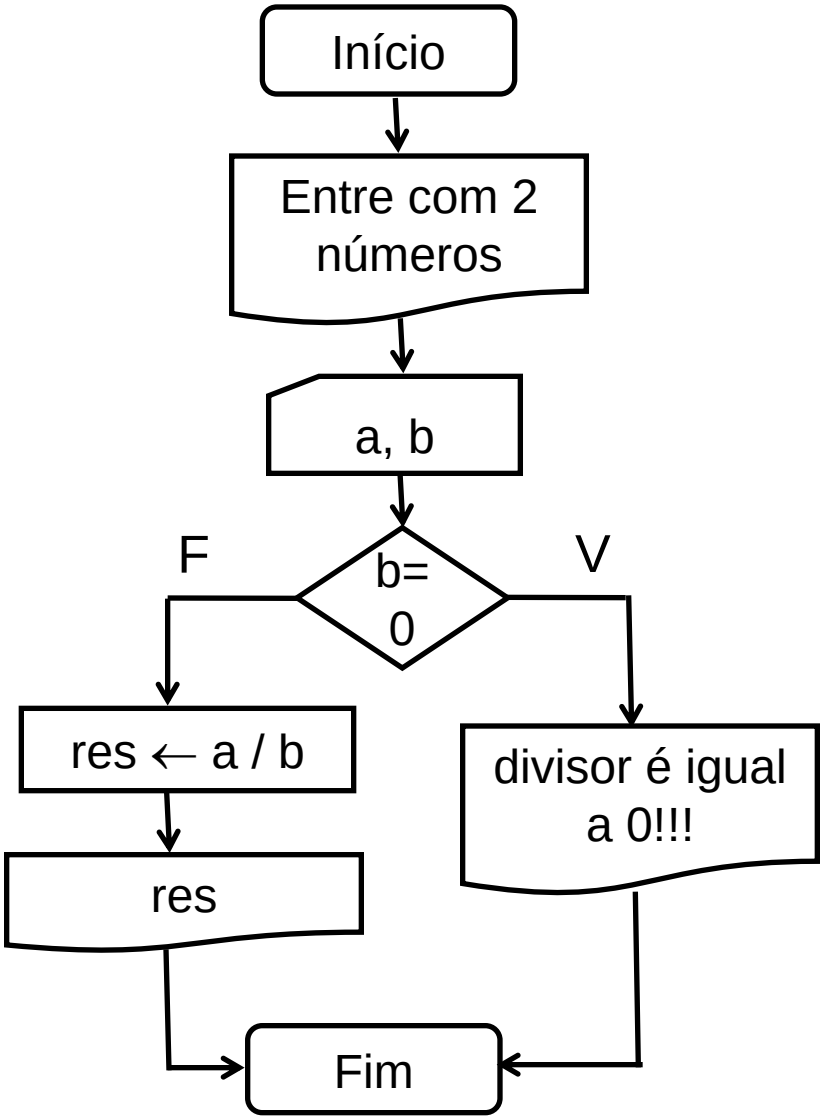
- Entrada de dados

- Saída de dados



- Tomada de decisão, com possibilidade de desvios

2.3. Fluxograma




```
Algoritmo Divisao
Início
        declare a, b, res
        escreva("Entre com 2 números:")
        leia (a,b)
        se b = 0  então
            escreva("divisor é igual a 0!!!")
        senão
            res  ← a / b
            escreva("O resultado e´:",res)
        fim se
Fim
```

2.4. Projeto

- **Construção de um algoritmo**
  - Começamos com uma afirmação genérica da solução do problema e prosseguimos até o algoritmo final, aumentando sistematicamente o nível de detalhamento

Como saber quando devemos parar o processo de detalhamento?

  - Isso depende do agente que irá executar o algoritmo
  - Além disso, deve-se lembrar que os computadores têm um conjunto muito limitado de instruções
    - » algoritmo deve ser expresso nos termos dessas instruções

2.4. Projeto

- **Metodologia**

Passo 1: estudar cuidadosamente a especificação do problema até o final

Passo 2: se ainda não entender o problema depois de estudá-lô cuidadosasamente, pergunte à pessoa (cliente, gerente, professor,…) responsável pela proposição do problema

Passo 3: levantar e analisar todas as saídas exigidas na especificação do problema

Passo 4: levantar e analisar todas as entradas citadas na especificação do problema

2.4. Projeto

Passo 5: Formular um esboço geral do algoritmo, não se concentrando em detalhes (*revisar mentalmente*)

Passo 6: verificar se é necessário gerar valores internamente e levantar as variáveis necessárias e os valores iniciais de cada uma (*indicar o propósito*)

Passo 7: levantar e analisar todas as transformações necessárias para, dadas as entradas e valores gerados internamente, produzir as saídas especificadas (*mapeamento entrada → saída*)

2.4. Projeto

Passo 8: testar cada passo do algoritmo, verificando se as transformações intermediárias executadas estão conduzindo aos objetivos desejados (*utilizar, sempre que possível, valores de teste que permitam prever os resultados*)

Passo 9: Retomar os passos individuais e prosseguir com o detalhamento (*verificar se os novos passos executam a função original*)

Passo 10: fazer uma reavaliação geral, elaborando o algoritmo através da integração das partes

2.4. Projeto

Mostre os algoritmos, nos exercícios a seguir, utilizando pseudo-código e fluxograma.

Exercício 2.1. Escreva um algoritmo que, dados os dois catetos, calcule a hipotenusa de um triângulo retângulo.

Exercício 2.2. Escreva um algoritmo que verifique se um número *M* é divisível por um número *N*.

Exercício 2.3. Escreva um algoritmo que mostre a situação de um aluno (aprovado ou reprovado) dadas as notas de duas provas. O aluno é considerado aprovado se a média for maior ou igual a 7.

2.4. Projeto

Exercício 2.4. Desenvolva um algoritmo que faça o seguinte somatório:

$$s = \sum_{i=1}^{L} i$$