Algoritmos e Programação – TAD0102

Prof. Dr. Josenalde Barbosa de Oliveira

josenalde@eaj.ufrn.br

Aulas: 2T345, 4T345 – 6 CRDS – 90h

Atendimento: e-mail, discord

https://github.com/josenalde/algorithms-programming

Algoritmos e Programação

Objetivos

(a) discente associará o pensamento algorítmico na proposta de soluções desde problemas do dia-a-dia que exigem tomada de decisão, estendendo para problemas lógico-matemáticos e de negócios, mediante estruturas de controle de fluxo como sequencial, condicional e repetição. Através do estudo de sintaxe e semântica de linguagem de programação, codificará soluções e planejará testes a partir de pseudocódigos e fluxogramas, compreendendo as traduções e equivalências entre as formas de representação, atentando as boas práticas de programação.

Conteúdo programático

Modelagem de problemas para solução computacional; Representação e manipulação de dados; Estruturas de controle de fluxo; Dados heterogêneos; Manipulação de Arquivos; Bibliotecas e noções de Orientação a Objetos

Algoritmos e Programação

Ementa – 90h

Introdução a algoritmos; Conceitos de memória, variáveis e constantes; Tipos básicos de dados; Operadores aritméticos, relacionais e lógicos; Comandos básicos de entrada, saída e atribuição; Conceito de bloco de comandos (escopo); Estruturas de controle de fluxo (condicionais e repetição); Estruturas de Dados Homogêneas (vetores e matrizes); Estruturas de Dados Heterogêneas (registros); Funções; Variáveis locais e globais; Passagem de parâmetros por valor e por referência; Ponteiros; Interpretação de mensagens de erro, depuração, noções de tratamento de erro.

Metodologia

Apresentações de slides com estímulo à interação e relatos de experiências, Uso de vídeos contextuais e para além (curiosidades), Resolução de exercícios de competições de programação

Avaliação aprendizagem

Projetos coletivos e individuais, Resolução de exercícios com participação em sala de aula, simulação competição de programação, avaliações escritas

Algoritmos e Programação

Avaliações

A divulgar

Referências

PRATA, S. C++ Primer Plus. 6th ed. NJ: Addison-Wesley, 2012

MIZRAHI, V.V. Treinamento em linguagem c++, 2. ed. PH: São Paulo, 2006.

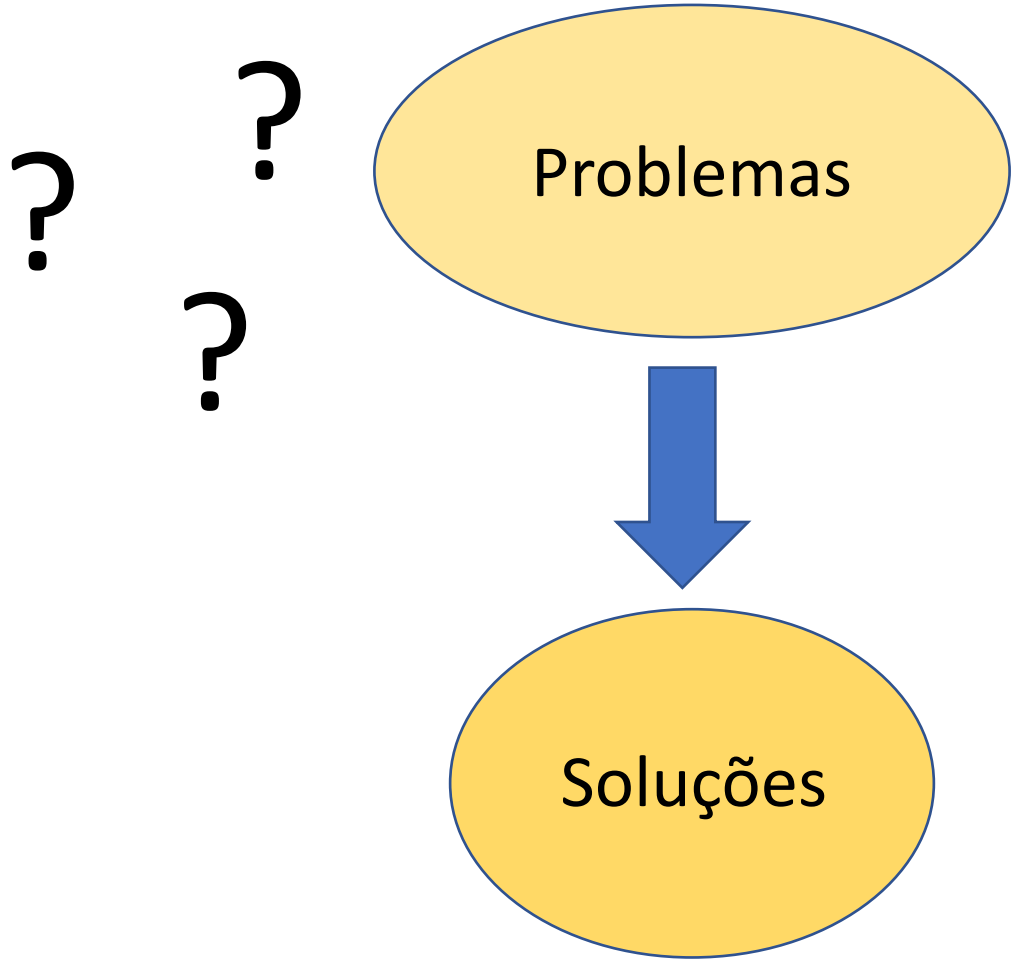
NEPS.ACADEMY – WEBSITE (dicas, macetes, muitos exercícios)

Ambiente

Visual Studio Code (VS Code) com extensões C/C++ IntelliSense, Code Runner

Instalar COMPILADOR, MinGW: https://osdn.net/projects/mingw/releases/ e apontar em c/c++ edit configurations (JSON), o compilerPath: C:/MinGW/bin/g++.exe

Motivação



Analisar Problemas **E**

Analisar Sistemas E

Desenvolver Soluções **E**

Desenvolver Sistemas

Ter em mente que...

- Podem existir inúmeras soluções para um mesmo problema:
  - melhor? Mais otimizada, eficaz? Menos recursos, tempo de execução e resposta, menos linhas de código a manter...!
- Em termos computacionais, procuramos explorar as que necessitem do menor número possível de “passos” ou instruções (linhas de código) E com menor tempo de execução E consumo de recursos;
- O estudante de computação e, particularmente, analista e desenvolvedor de sistemas, é um solucionador em potencial dos mais variados problemas (áreas etc.);
- Portanto, raciocínio lógico-matemático, encadeamento de ideias, enfim, LÓGICA, são competências a estimular, desenvolver, consolidar ou mesmo despertar, descobrir.

Exemplo: compreender e estruturar solução

- Você está numa margem de um rio, com três animais: uma galinha, um cachorro e uma raposa. Somente pode atravessar com um animal por vez e nunca deixar a raposa e o cachorro sozinhos nem a raposa e a galinha. Descreva uma formar de conseguir atravessar os três animais, obedecendo a essas condições.

Exemplo: compreender e estruturar solução

- Dados do problema:
  - Variáveis: galinha (G), cachorro (C), raposa (R)
  - Restrições: 1) **C e R**, 2) **R e G**, 3) **um por vez, bidirecional**
    - Das restrições infere-se que **C e G podem estar sozinhos**
    - **Passos para solução:**
      1. Levar R, deixando C e G sozinhos
      2. Levar G, deixando C sozinho. Trazer R, deixando G sozinho
      3. Levar C, deixando R sozinho.
      4. Levar R

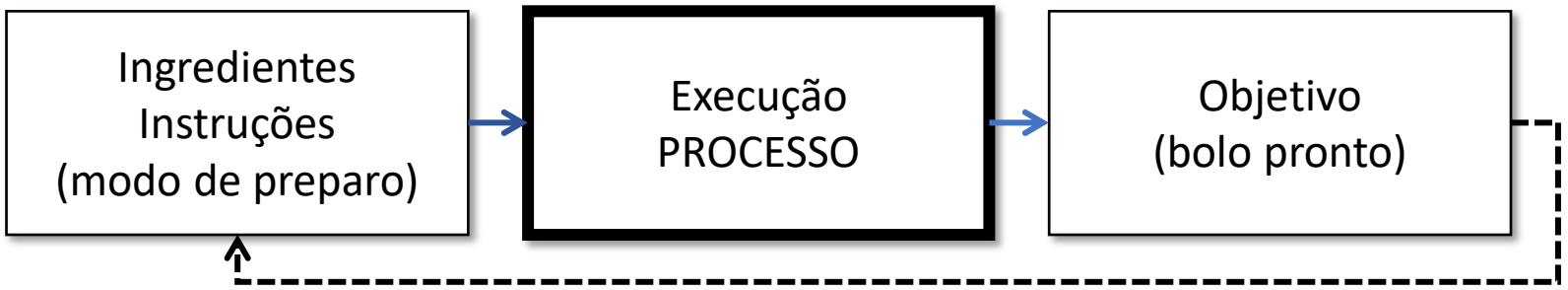
Inferido!

OBS: *percebemos que são realizadas operações sobre as variáveis (levar, trazer) e que as mesmas possuem "estados", ou seja, estão de um lado ou do outro do rio.*

Existem outras soluções?

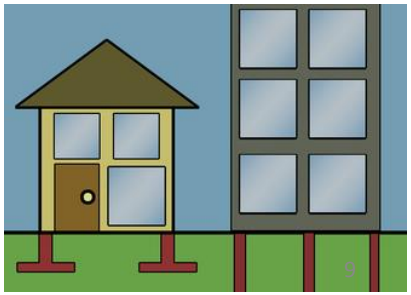
Surge então o conceito de algoritmo

- Um **algoritmo** representa um conjunto de **instruções** organizadas de **modo lógico** para a **solução** de um problema (descrição geral). Neste sentido, uma receita de bolo é um exemplo de algoritmo.



Pode haver realimentação (feedback)

- Ao raciocinar, empregamos algoritmos! Eles estão em nosso dia-a-dia!
- Será neste curso nossa BASE, nosso alicerce



Exemplo 1 - identificar estruturas de fluxo

SEQUENCIA, CONDIÇÃO

```
Algoritmo para fazer um bolo simples
1 – pegar os ingredientes;
2 – se (roupa branca) então
          colocar avental;
3 – se (tiver batedeira) então
          bater os ingredientes na batedeira;
    senão
          bater os ingredientes à mão;
4 – colocar a massa na forma;
5 – colocar a forma no forno;
6 – aguardar o tempo necessário;
7 – retirar o bolo;
Fim
```

Exemplo 2

SEQUENCIA, CONDIÇÃO

```
Algoritmo para trocar lâmpadas
1 – se (lâmpada estiver fora de alcance)
        pegar a escada;
2 – pegar a lâmpada;
3 – se (lâmpada estiver quente)
            pegar pano;
4 – tirar lâmpada queimada;
5 – colocar lâmpada boa;
Fim
```

Exemplo 3

SEQUENCIA, CONDIÇÃO, REPETIÇÃO

```
Algoritmo para descascar batatas
1 – pegar faca, bacia e batatas;
2 – colocar água na bacia;
3 – enquanto (houver batatas)
        descascar batatas;
Fim
```

Exemplo 4

```
Algoritmo para fazer uma prova
1 – ler a prova;
2 – pegar a caneta;
3 – enquanto ((houver questão em branco) E (tempo não terminou)) faça/repita
      se (souber a questão)
            resolvê-la;
      senão
            pular para outra;
4 – entregar a prova;
Fim
```

SEQUENCIA, CONDIÇÃO, REPETIÇÃO

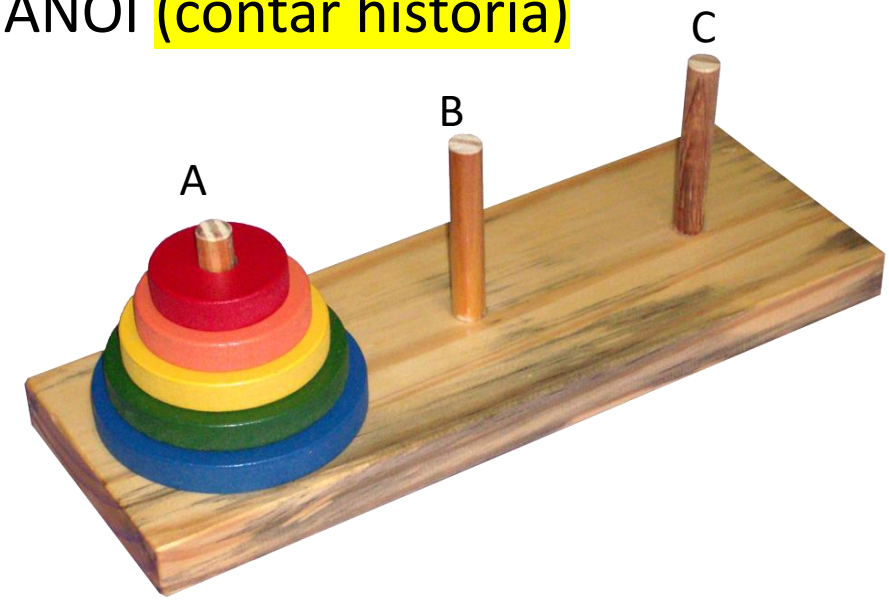
Então, podemos dizer

- que um **programa de computador** nada mais é do que um ou mais algoritmos **escritos** numa **linguagem** de programação (C, C++, Pascal, Fortran, Delphi, Java, Python, Ruby, Basic, PHP, Javascript, Dart, Kotlin entre outras);

- O mais importante de um programa é sua LÓGICA, o RACIOCÍNIO utilizado para resolver o problema, que é exatamente o **ALGORITMO**.

Vamos “brincar.”..

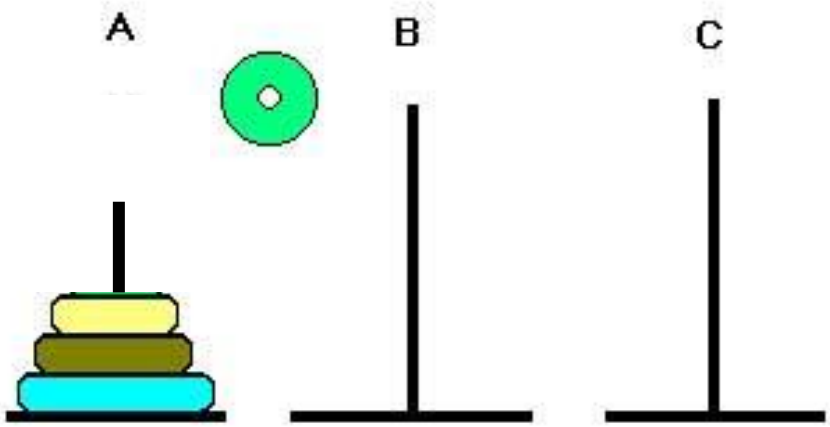
- Problema da TORRE DE HANOI (contar história)



- Proposição: inicialmente tem-se **três** hastes: A, B e C, e na haste A repousam **N** anéis de diâmetros diferentes, em **ordem decrescente** de diâmetro.

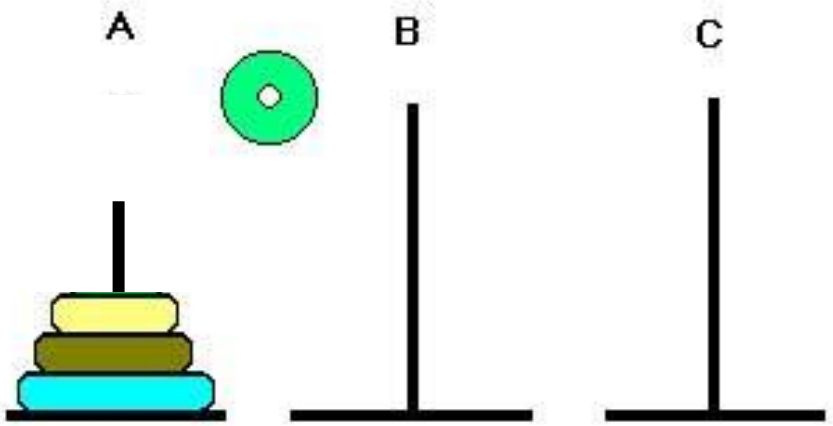
Torre de Hanoi

- O objetivo é transferir os três anéis da haste A para B, usando C se necessário. As regras do movimento são (restrições, condições):
  - Deve-se mover um único anel por vez;
  - Um anel de diâmetro maior nunca pode repousar sobre algum outro de diâmetro menor.



Torre de Hanoi

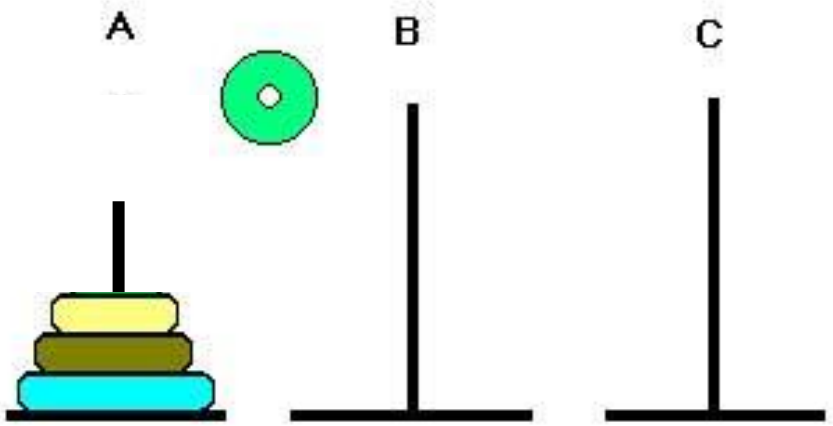
- Solução #1:
  - 1: A -> B
  - 2: A -> C
  - 3: B -> C
  - 4: A -> B
  - 5: C -> A
  - 6: C -> B
  - **7**: A -> B



Torre de Hanoi

- Solução #2:

  - 1: A -> C
  - 2: A -> B
  - 3: C -> B
  - 4: A -> C
  - 5: B -> C
  - 6: B -> A
  - 7: C -> A
  - 8: C -> B
  - 9: A -> C
  - 10: A -> B
  - **11**: C -> B



Portanto,

Podem existir várias soluções para o mesmo problema!

- Devemos nos esforçar (raciocinar) por buscar a “melhor” solução do ponto de vista computacional!

- No caso da TORRE DE HANOI, a solução que utilize o **MENOR NÚMERO DE MOVIMENTOS!**

- Imaginemos que cada movimento exige (gasta): tempo, energia, esforço (físico, mecânico,...)

Este é um tipo de problema determinístico quanto ao número mínimo de movimentos...

Número de movimentos = 2^n - 1

M(n) = 2^n - 1

onde n é o número de discos.

O PROBLEMA PODE SER GENERALIZADO e VISTO DE OUTRO MODO

Outra forma de pensar o problema é não estabelecer qual deve ser a haste destino e enxergar as hastes dispostas em círculo.

Olhar diferente...

Se n for ímpar, os anéis serão transferidos para a primeira haste após a haste de início, **no sentido horário;**
Se n for par, os anéis serão transferidos para a primeira haste após a haste de início, **no sentido anti-horário;**

Algoritmo GERAL para as Torres de Hanoi
Início
Repita
1. Mova o menor anel de sua haste atual para a próxima
2. Execute o único movimento possível com um anel que não seja o menor de todos.

Até que todos os discos tenham sido transferidos para outra haste.
Fim

Seria possível então programar um robô para jogar?

https://www.youtube.com/watch?v=SEMfUE5K35I

IDEIAS: ?

Modelagem de problemas – descrição computacional

Considere o seguinte problema:
Compraram-se 30 canetas iguais, que foram pagas com uma nota de R$ 100,00, obtendo-se R$ 67,00 como troco. Quanto custou cada caneta?

Como poderíamos raciocinar:
Se eu tinha R$ 100,00 e recebi como troco R$ 67,00, o custo do total de canetas é a diferença entre os R$ 100,00 que eu tinha e os R$ 67,00 do troco. Ora, isto vale R$ 33,00; portanto, esse valor foi o total pago pelas canetas. Para saber quanto custou cada caneta, basta dividir os R$ 33,00 por 30, resultando em R$ 1,10, ou seja, o preço de cada caneta.

Matematicamente (**x** o custo de cada caneta, então quantoGastei=**30x**. Como quantoGastei + troco = R$ 100,00, tem-se):

30x + 67 = 100
30x = 100 – 67
30x = 33
X = 33/30
X = 1,1

Observe que o resultado é número REAL

Modelagem de problemas – descrição computacional

Problema GERAL: **Compraram-se N canetas iguais, que foram pagas com uma nota de Z reais, obtendo-se Y reais como troco. Quanto custou cada caneta?**

É NECESSÁRIO PENSAR SOBRE AS RESTRIÇÕES DO PROBLEMA (tratamentos, o que é necessário "tratar"):

-O que acontecerá se alguém tentar comprar 0 canetas?
-Ou -3 canetas? Faz sentido?
-Suponha que N = 10; Z = 10; Y = 15. Cada caneta será **R$ - 0,50.**

OU SEJA, temos as restrições (que devem estar codificadas)

-O valor pago pelas canetas seja sempre maior que o troco recebido;
-Que o valor pago e a quantidade de canetas seja sempre maior que zero;
-Que o troco seja maior ou igual a zero.

LOGO, as restrições são: Z > Y, N > 0, Z > 0 e Y >= 0

Modelagem de problemas – descrição computacional

Algoritmo GERAL e CORRETO para o problema das canetas:

```
Algoritmo custoCaneta
Início
        Leia(N, Y, Z)
        Se (Z > Y E N > 0 E Y >= 0 E Z > 0) Então
                totalCanetas ← (Z – Y) / N
                Exiba(totalCanetas).
        Senão
                Exiba(“Erro: os valores são inconsistentes!”)
        Fim_Se
Fim
```

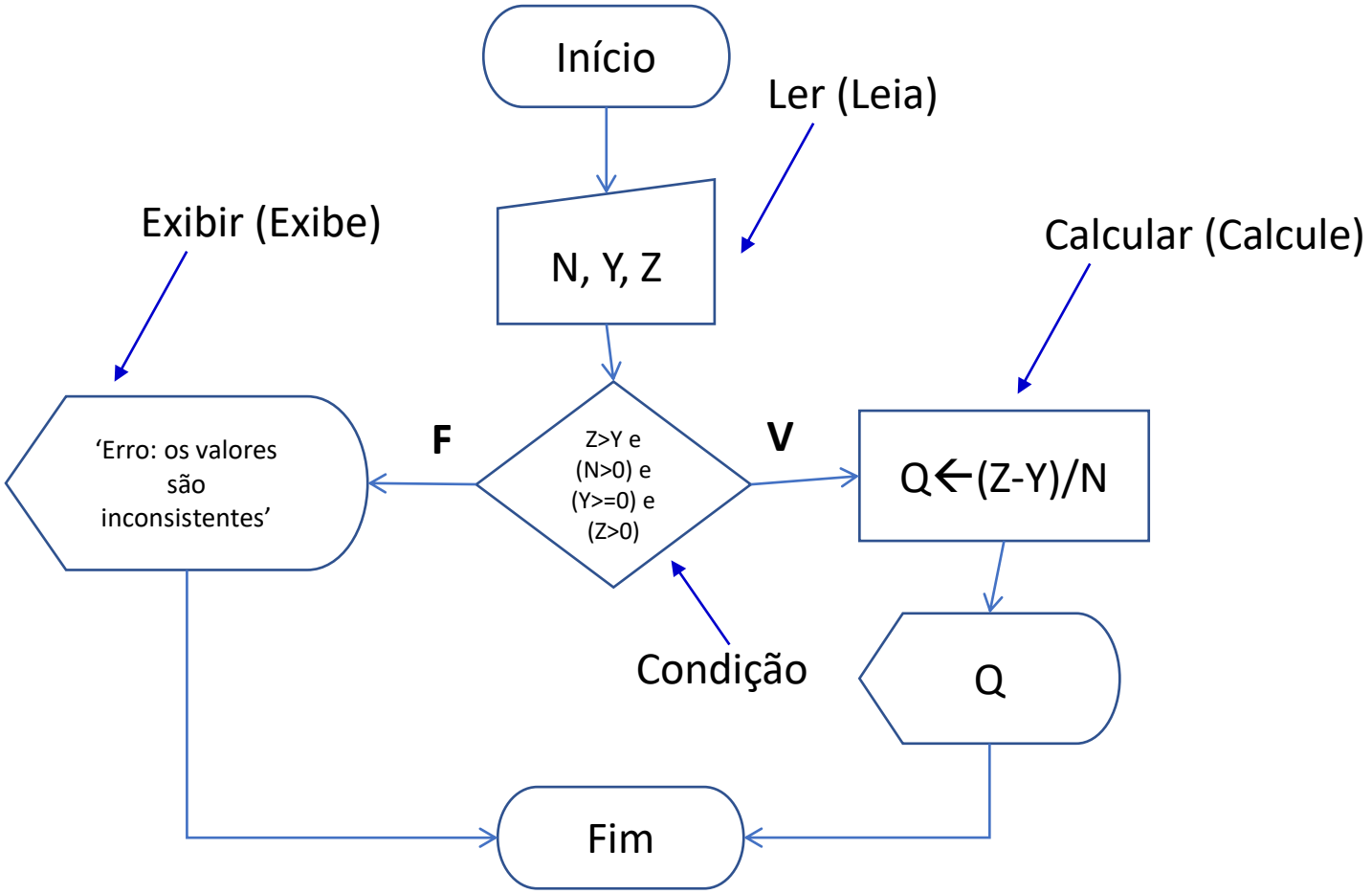
Estrutura GERAL de um algoritmo

D L T C E

Declarar – Ler – Testar – Calcular - Exibir

Modelagem de problemas – descrição computacional

Fluxograma para resolver o problema das canetas:



Modelagem de problemas – descrição computacional

Portanto, não **DECORAMOS** soluções, mas buscamos o **ENTENDIMENTO** de como foi obtida uma solução e usamos esta **EXPERIÊNCIA** adaptando-as a outras situações, por **ANALOGIA**, **GENERALIZAÇÃO** ou **ESPECIALIZAÇÃO**.

DICAS:

1. Ao se deparar com um problema novo, tente entendê-lo. Para auxiliar, pense no seguinte:
- o que se deve descobrir ou calcular? Qual é o objetivo **(saída)?**
- quais são os **dados de entrada** disponíveis? São suficientes?
- quais **as restrições** para resolver o problema?
- se possível, modele o problema de forma matemática.

Modelagem de problemas – descrição computacional

2. Crie um plano com a solução:

-Consulte sua memória e verifique se você já resolveu algum problema similar;

- Verifique se é necessário introduzir algum elemento novo no problema, como um problema auxiliar; (INFERÊNCIA)

- Se o problema for muito complicado, tente quebrá-lo em partes menores e solucionar estas partes; (TOP-DOWN)

- É possível enxergar o problema de outra forma, de modo que seu entendimento se torne mais simples? **(TROCA, OBI)**

Modelagem de problemas – descrição computacional

3. Formalize a solução:

-Crie um algoritmo informal com passos que resolvam o problema;

- Verifique se cada passo desse algoritmo está correto;

- Escreva um algoritmo formalizado por meio de um fluxograma ou outra técnica de representação.

Modelagem de problemas – descrição computacional

4. Exame dos resultados:

-Teste o algoritmo com diversos dados de entrada e verifique os resultados (teste de mesa);

- Se o algoritmo não gerou resultado algum, o problema está na sua **SINTAXE** e nos comandos utilizados. Volte e tente encontrar o erro;

- Se o algoritmo gerou resultados, estes estão corretos? Analise sua consistência (**SEMÂNTICA**).

- Se não estão corretos, alguma condição, operação ou ordem das operações está incorreta. Volte e tente encontrar o erro.

Modelagem de problemas – descrição computacional

5. Otimização da solução:

-É possível melhorar o algoritmo?

-É possível reduzir o número de passos ou dados?

-É possível conseguir uma solução ótima?

Encaremos os problemas de COMPUTAÇÃO como verdadeiros projetos de Engenharia (de software). Os programas de computador são ferramentas empregadas para auxiliar as pessoas.

USABILIDADE – EXPERIÊNCIA DO USUÁRIO – INTERFACE HUMANO-COMPUTADOR