Apostila de

Algoritmo Estruturado

Prof. José Maria Rodrigues Santos Júnior
(prof. na Universidade Tiradentes)

INTRODUÇÃO

A automatização de tarefas é um aspecto marcante da sociedade moderna. O aperfeiçoamento tecnológico alcançado, com respeito a isto, teve como elementos fundamentais a análise e a obtenção de descrições da execução de tarefas em termos de ações simples o suficiente, tal que pudessem ser automatizadas por uma máquina especialmente desenvolvida para este fim, O COMPUTADOR.

Em ciência da computação houve um processo de desenvolvimento simultâneo e interativo de máquinas (hardware) e dos elementos que gerenciam a execução automática (software) de uma dada tarefa. E essa descrição da execução de uma tarefa, como considerada acima, é chamada **algoritmo**.

O objetivo desse curso é a Lógica de Programação dando uma base teórica e prática, suficientemente boa, para que, o aluno domine os algoritmos e esteja habilitado a aprender uma linguagem de programação. Será mostrado também um grupo de algoritmos clássicos para tarefas cotidianas, tais como : ordenação e pesquisa.

DEFINIÇÃO DE ALGORITMO

- "O conceito central da programação e da Ciência da Computação é o conceito de algoritmos, isto é, programar é basicamente construir algoritmos."
- É a descrição, de forma lógica, dos passos a serem executados no cumprimento de determinada tarefa.
- "O algoritmo pode ser usado como uma ferramenta genérica para representar a solução de tarefas independente do desejo de automatizá-las, mas em geral está associado ao processamento eletrônico de dados, onde representa o rascunho para programas (Software)."
- "Serve como modelo para programas, pois sua linguagem é intermediária à linguagem humana e às linguagens de programação, sendo então, uma boa ferramenta na validação da lógica de tarefas a serem automatizadas."
- "Um algoritmo é uma receita para um processo computacional e consiste de uma série de operações primitivas, interconectadas devidamente, sobre um conjunto de objetos. Os objetos manipulados por essas receitas são as variáveis."
- O algoritmo pode ter vários níveis de abstrações de acordo com a necessidade de representar ou encapsular detalhes inerentes às linguagens de programação. Ex: Certamente um algoritmo feito com o objetivo de servir como modelo para uma linguagem de III geração é diferente daquele para uma linguagem de IV geração. Mas isso não impede que a ferramenta em si possa ser usada em ambos o caso.
- Como qualquer modelo, um algoritmo é uma abstração da realidade. A abstração é o processo de identificar as propriedades relevantes do fenômeno que esta sendo modelado. Usando o modelo abstrato, podemos nos centrar unicamente nas propriedades relevantes para nós, dependendo da finalidade da abstração, e ignorar as irrelevantes.
- É a forma pela qual descrevemos soluções de problemas do **nosso mundo**, afim de, serem implementadas utilizando os recursos do **mundo computacional**. Como este possuí severas limitações em relação ao nosso mundo, exige que, sejam impostas algumas regras básicas na forma de solucionar os problemas, para que, possamos utilizar os recursos de **hardware** e **software** disponíveis. Pois, os algoritmos, apesar de servirem para representar a solução de qualquer problema, no caso do Processamento de Dados, eles devem seguir as regras básicas de programação para que sejam compatíveis com as **linguagens de programação**.

LINGUAGEM DE DESCRIÇÃO DE ALGORITMO (LDA)

Para escrevermos algoritmos é preciso uma linguagem clara e que não deixe margem a ambigüidades, para isto, devemos definir uma sintaxe e uma semântica, de forma a permitir uma única interpretação das instruções num algoritmo.

Estrutura um Algoritmo

Algoritmo Nome_Do_Algoritmo
variáveis
Declaração das variáveis
Procedimentos
Declaração dos procedimentos
Funções
Declaração das funções
Início
Corpo do Algoritmo
Fim

Identificadores

Representam os nomes escolhidos para rotular as variáveis, procedimentos e funções, normalmente, obedecem as seguintes regras :

1. O primeiro caracter deve ser uma letra
2. Os nomes devem ser formados por caracteres pertencentes ao seguinte conjunto : {a,b,c,..z,A,B,C,...Z,0,1,2,...,9,_}
3. Os nomes escolhidos devem explicitar seu conteúdo.

Variáveis:

Unidades básicas de armazenamento das informações a nível de linguagens de programação. Os tipos de dados e variáveis utilizados dependem da finalidade dos algoritmos, mas, podemos definir alguns, pelo fato de serem largamente utilizados e implementados na maioria das linguagens, sendo estes:

INTEIRO : qualquer número inteiro, negativo, nulo ou positivo.
REAL : qualquer número real, negativo, nulo ou positivo.
CARACTER : qualquer conjunto de caracteres alfanuméricos.
LÓGICO : tipo especial de variável que armazena apenas os valores V e F, onde V representa VERDADE e F FALSO

Declaração de variáveis

Para que os programas manipulem valores, estes devem ser armazenados em variáveis e para isso, devemos declará-las de acordo com a sintaxe:
NomeVariável,... : **tipo**

Operações Básicas:

Na solução da grande maioria dos problemas é necessário que as variáveis tenham seus valores consultados ou alterados e, para isto, devemos definir um conjunto de **OPERADORES**, sendo eles:

- **OPERADOR DE ATRIBUIÇÃO:**

NomeDaVariavel ← Valor ou Expressão Atribuída.

- **OPERADORES ARITMÉTICOS:**

+ = Adição	Quociente = Quociente da divisão de inteiros
***** = Multiplicação	Resto = Resto da divisão de inteiros
- = Subtração ou inversor do sinal.	EXP(a,b) = Exponenciação a^b
/ = Divisão	

- **FUNÇÕES PRIMITIVAS:** SEN(x); COS(x); TG(x); ABS(x); INT(x); Raiz(x); PI( );

- **OPERADORES RELACIONAIS:**

São utilizados para relacionar variáveis ou expressões, resultando num valor lógico (Verdadeiro ou Falso), sendo eles:

= - igual	≠ - diferente
< - menor	> - maior
≤ - menor ou igual	≥ - maior ou igual

- **OPERADORES LÓGICOS:**

São utilizados para avaliar expressões lógicas, sendo eles:

e - e lógico ou conjunção.
ou - ou lógico ou disjunção.
não - negação.

PRIORIDADE DE OPERADORES:

Durante a execução de uma expressão que envolve vários operadores, é necessário a existência de prioridades, caso contrário poderemos obter valores que não representam o resultado esperado.
A maioria das linguagens de programação utiliza as seguintes prioridades de operadores :

1º - Efetuar operações embutidas em parênteses "mais internos"
2º - Efetuar Funções
3º - Efetuar multiplicação e/ou divisão
4º - Efetuar adição e/ou subtração
5º - Operadores Relacionais
6º - Operadores Lógicos

OBS: O programador tem plena liberdade para incluir novas variáveis, operadores ou funções para adaptar o algoritmo as suas necessidades, lembrando sempre, de que, estes devem ser compatíveis com a linguagem de programação a ser utilizada.

COMANDOS DE ENTRADA E SAÍDA :

No algoritmo é preciso representar a troca de informações que ocorrerá entre o mundo da máquina e o nosso mundo, para isso, devemos utilizar comandos de entrada e saída, sendo que, a nível de algoritmo esses comandos representam apenas a entrada e a saída da informação, independe do dispositivo utilizado (teclado, discos, impressora, monitor,...), mas, sabemos que nas linguagens de programação essa independência não existe, ou seja, nas linguagens de programação temos comandos específicos para cada tipo de unidade de Entrada/Saída.

Comando de Entrada de Dados
Leia(variável_1, variável_2,...)
Comando de Saída de Dados
Imprima(expressão_1, expressão_2,...)

COMANDOS DE CONTROLE DE FLUXO:

Para representar a solução de um problema devemos escrever o conjunto de passos a serem seguidos, sendo que, a maioria dos problemas exigem uma dinâmica na sua solução, impondo assim que os algoritmos executem conjunto de instruções de acordo com as possíveis situações encontradas no problema original. E de acordo com a **Programação Estruturada** os mecanismos utilizados para esse controle são : **Sequência**, **Seleção** e **Repetição.**

- **SEQUÊNCIA :** usada para executar comandos passo a passo, sabendo que todos eles serão executados na ordem de escrita, sem nenhum desvio. Uma sequência pode possuir um ou vários comandos, os quais devem ser delimitados pelos identificadores **Início** e **Fim**.

Inicio
Comando_1
...
Comando_n
Fim

- **SELEÇÃO :** usada para tomar decisões, ou seja desviar a execução do algoritmo de acordo com uma condição, podendo ser simples ou composta.

Simples	**Composta**
Se (Expressão Lógica) **Então** Sequência_1	**Se** (Expressão Lógica) **Então** Sequência_1 **Senão** Sequência_2

- **REPETIÇÃO :** Serve para efetuar um conjunto de ações repetidas vezes. Existem três tipos básicos de repetições, sendo elas.

Enquanto (Expressão Lógica) **faça** Seqüência	O comando **Enquanto** analisa a *Expressão Lógica* e enquanto o seu resultado for, o valor lógico, *Verdade* a *Seqüência* é executada.
Para variável ← valor_inicial **até** valor_final **faça** Seqüência	O comando **Para** incrementa, a *variável*, a partir do *valor_inicial* de uma unidade, até que, esta atinja o *valor_final*. E para cada incremento a *Seqüência* é executada..
Repita Seqüência **Até** (Expressão Lógica)	O comando **Repita** executa a *Seqüência* até que o valor retornado pela *Expressão Lógica* seja *Verdadeiro*

TIPOS DE DADOS

Estruturas formadas por um conjunto de variáveis, permitindo modelar de forma mais natural os dados.

VETOR: estrutura formada por um conjunto unidimensional de dados de mesmo tipo (homogêneo) e possuindo número fixo de elementos (Estático). Na declaração dos vetores devemos informar o seu nome, seu tipo (inteiro, real, caracter, ...), e seu tamanho (número de elementos). Cada elemento do vetor é identificado por um índice (unidimensional), o qual indica a sua posição no vetor.

Declaração :
NomeDoVetor : **vetor**[nº de elementos] **de** Tipo do Vetor
Referência :
NomeDoVetor[índice]

MATRIZ: estrutura semelhante ao vetor, sendo que, pode possuir **n** dimensões. Desta forma para fazer referência aos elementos de uma matriz, precisaremos de tantos índices quanto for suas dimensões.

Declaração :
NomeDaMatriz : **matriz**[dimensões] **de** Tipo da Matriz
Referência :
NomeDaMatriz[índices]

REGISTRO: estrutura formada por um conjunto de variáveis, que podem possuir tipos diferentes (Heterogêneo), agrupadas em uma só unidade.

Declaração :
NomeDoRegistro : Registro
Declaração de Variáveis
FimRegistro
Referência :
NomeDoRegistro.NomeDaVariável
Obs: Podemos ainda definir um vetor formado por registros.

MODULARIZAÇÃO

A modularização consiste num método para facilitar a construção de grandes programas, através de sua divisão em pequenas etapas, que são : módulos, rotinas, sub-rotinas ou sub-programas. Permitindo o reaproveitamento de código, já que podemos utilizar um módulo quantas vezes for necessário, eliminando assim a necessidade de escrever o mesmo código em situações repetitivas.

Procedimentos - Um procedimento é um bloco de código precedido de um cabeçalho que contém o Nome do procedimento e seus parâmetros. Com isto, podemos fazer referência ao bloco de código de qualquer ponto do algoritmo através do seu *nome* e passando os seus *parâmetros*.

Declaração :
Procedimento NomeDoProcedimento [(*parâmetros*)]
Variáveis
Inicio
Comandos;
Fim;

Onde, *parâmetros* representam as variáveis que devem ser passadas ao procedimento. Os parâmetros podem ser de : ENTRADA (passado por valor) ou de ENTRADA/SAÍDA (passado por referência). Os parâmetros de ENTRADA não podem ser alterados pelo procedimento, para que isso seja possível o parâmetro deve ser de ENTRADA/SAÍDA Para indicar que um parâmetro é de ENTRADA/SAÍDA devemos colocar a palavra *VAR* antes da sua declaração.

Referência :
NomeDoProcedimento(variáveis)

OBS: As variáveis passadas aos procedimentos são associadas aos parâmetros do procedimento de acordo com a ordem das variáveis e da lista de parâmetros.

Funções - Uma função é semelhante a um procedimento, sendo que esta deve retornar, obrigatoriamente, um valor em seu nome, desta forma, é necessário declarar, no cabeçalho da função, qual o seu tipo.

Declaração :
Função NomeDaFunção [(parâmetros)] **:** tipo_da_função
Variáveis
Inicio
Comandos
NomeDaFunção ← (expressão de retorno)
Fim;
Referência :
NomeDaFunção(parâmetro)

ALGORÍTMOS DE PESQUISA

A capacidade de armazenar informações foi um passo decisivo na evolução da ciência da computação e para o nível generalizado de utilização do computador. Com isso, a capacidade de recuperar informações, para posterior processamento, assume papel de suma importância na utilização cotidiana do computador, existindo para isto inúmeros exemplos, como: recuperação de dados de dados de transações bancárias de um cliente através de um número de conta, no cadastro de cliente/operações de um banco. Portanto, algoritmos de pesquisa devem ser projetados de forma a garantir a confiabilidade e eficiência exigidas pela importância das aplicações existentes.

A pesquisa de dados pode ser efetuada tanto em unidades de memória secundárias (disco rígido, disquetes, fita), quanto na memória principal do computador.

PESQUISA SEQUENCIAL

O método mais simples de determinar a presença, ou não, de um elemento numa seqüência, é percorrê-la a partir do seu início, efetuando comparações, até que o elemento seja encontrado ou o fim da seqüência seja alcançado. Este método é chamado de pesquisa seqüencial.

Dados :

vetor de n elementos (n conhecido)
elemento a ser pesquisado no vetor

Resultado:

Se o elemento existe, mostra-se a sua posição ou o total de ocorrências deste no vetor.
Se o elemento não existe, mostra-se uma mensagem de falha.

As considerações que podem ser feitas sobre os dados de entrada (vetor), são do tipo: o vetor esta ou não ordenado; o elemento ocorre uma única vez (pesquisa única) ou repetidas vezes no vetor (pesquisa única). Isso acarreta os seguintes tipos de pesquisa:

a. Desordenada Única
b. Desordenação Múltipla
c. Ordenada Única
d. Ordenada Múltipla

Pesquisa Binária

O método de pesquisa seqüencial é fácil de escrever e é razoavelmente eficientes para seqüências com poucos elementos. Entretanto, para seqüências de tamanho considerável, que ocorrem na maioria das aplicações existentes, a utilização do método torna-se inviável. Uma estratégia interessante e eficiente é utilizada no método de pesquisa binária.

Descrição Geral do Método:

- Definir intervalo inicial (i, f) de busca
- Determinar a posição média do intervalo(m = (i+f) DIV 2)
- Comparar o elemento da posição média (v[m]) com o elemento E:
- Caso sejam iguais então terminou as pesquisa

- Caso contrário definir o novo intervalo de busca
- Aplicar sucessivamente o passo anterior até encontrar E ou não existir mais o intervalo de busca

São aspectos fundamentais do método:

- vetor de entrada tem que estar ordenado
- intervalo de busca inicial é (i,f) = (1,n)
- intervalo de busca, considerado a cada iteração, é definido do seguinte modo:

(i,m-1), se (E < v[m])
(m+1,f), se (E > v[m])
tendo a metade do tamanho do intervalo original

- O teste de repetição é (i <= f) e Não Achou

Dados :

vetor de n elementos (n conhecido)
elemento a ser pesquisado no vetor

Resultado

Se o elemento existe, mostra-se a sua posição ou o total de ocorrências deste no vetor.
Se o elemento não existe, mostra-se uma mensagem de falha

ALGORÍTMOS DE ORDENAÇÃO

Os problemas de ordenação são comuns tanto em aplicações comerciais quanto científicas. Entretanto, raro são os problemas que se resumem à pura ordenação de seqüências de elementos. Normalmente, os problemas de ordenação são inseridos em problemas de pesquisa, intercalação e atualização. Isto torna ainda mais importante o projeto e a construção de algoritmos eficientes e confiáveis para tratar o problema.

O nosso objetivo é analisar os seguintes tipos de ordenação :

a. Selection Sort
b. Bubble Sort
c. Insertion Sort

a. Selection Sort

Este método é um dos mais simples e intuitivos dentre os métodos existentes. Sua estratégia básica é selecionar o menor elemento da seqüência considerada e colocá-lo no início da seqüência. Assim, dada uma seqüência de tamanho n, várias iterações são efetuadas, sendo que a cada vez que esta estratégia é aplicada, uma nova seqüência é gerada pela eliminação do menor elemento da seqüência original.

```
Procedure SelectionSort ( var vet : vetor; n : integer);
{ordenado crescente}
var
i, j, pmin : integer;
begin
for i← 1 to (n-1) do
begin
pmin ← i;
for j← (i+1) to n do
if vet[j] < vet[pmin]
then pmin ← j;
trocar (vet[i], vet[pmin] ) ;
end;
end;
```

b. Bubble Sort

A estratégia utilizada pelo BubbleSort consiste de comparações e trocas entre elementos consecutivos da seqüência, a fim de "empurrar" o maior elemento para a última posição. Assim, várias iterações são efetuadas e, para cada seqüência considerada, a aplicação da estratégia gera uma nova seqüência pela eliminação do maior elemento da seqüência original.

Além disto, uma variável de controle (lógica) é utilizada para registrar a ocorrência ou não de troca entre elementos da seqüência. Quando nenhuma troca é efetuada, tem-se que a seqüência considerada já estava ordenada. Esta particularidade determina, em alguns casos, um número menor de comparações que o método SelectionSort.

```
Procedure BubbleSort ( var vet : vetor ; n integer) ;
{ordem crescente}
var
i, limite : integer;
trocou : boolean;
begin
limite ← n;
repeat
trocou ← false;
for i← 1 to (limite - 1) do
begin
if vet[i] > vet [i+1] then
begin
trocar(vet[i], vet[i+1]);
trocou ← true;
end;
end;
limite ← limite - 1
until not trocou
end;
```

c. Insertion Sort

Este método baseia-se no seguinte processo de inserção controlada:

- Com o primeiro elemento da seqüência forma-se uma seqüência de tamanho 1, ordenada.

- Cada elemento restante da seqüência original é inserido na seqüência, de modo que esta permaneça ordenada. Isto é feito através de uma pesquisa na seqüência ordenada que determina a posição que o novo elemento deverá ser inserido.
- Quando um elemento é inserido a frente de outro, estes deverão ser deslocados de uma posição.

RECURSIVIDADE

Recursão é um método geral para resolver problemas reduzindo-os a problemas mais simples do mesmo tipo. A estrutura geral de uma solução recursiva de um problema é assim :
Resolva de forma recursiva um problema.

- Se o problema é trivial, faça o obvio (resolva-o)
- Simplifique o problema
- Resolva de forma recursiva (um problema mais simples)
- Combine (na medida do possível) a solução do(os) problemas mais simples em uma solução do problema original

Um subprograma recursivo chama a si próprio constantemente, cada vez em uma situação mais simples, até chegar ao caso trivial, quando pára. Devemos lembrar que recursividade deve ser utilizada na solução de problemas que tenham a natureza recursiva.

Exemplos :

a. Somatório de inteiros - Se n =1; Somatório = 1. Caso contrário Somatório = n + Somatório(n-1)
b. Fatorial - Se n=0 ou n=1 ; Fatorial = 1. Caso contrário Fatorial = n*Fatorial(n-1)
c. MDC - Se b divide a, então o MDC é b. Caso contrário, MDC(a,b) = MDC(b,a mod b)
d. N-ésimo termo da série de Finonacci . 1° e 2° = 1 e n-ésimo = (n-1)+(n-2)
e. Torre de hanói

Bibliografia

INTRODUÇÃO AO DESENVOLVIMENTO DE ALGORITMOS
WILSON SILVA PINTO

ALGORITMOS
JOSE AUGUSTO MANZANO - JAYR FIGUEIREDO OLIVEIRA

ALGORITMOS E ESTRUTURAS DE DADOS
NIKLAUS WIRTH

ALGORITMOS E ESTRUTURAS DE DADOS
ANGELO DE MOURA GUIMARAES - NEWTON A C LAGES

ALGORITMOS ESTRUTURADOS
H FARREL - C G BECKER - E C FARIA e MATOS, H F

PROJETO DE ALGORITMOS
NIVIO ZIVIANI