

Notas de Aula de Algoritmos e Programação de Computadores

FLÁVIO KEIDI MIYAZAWA

com a colaboração de

TOMASZ KOWALTOWSKI

Instituto de Computação - UNICAMP

Versão 2000.1

Estas notas de aula não devem ser usadas como única fonte de estudo. O aluno deve ler outros livros disponíveis na literatura.

Nenhuma parte destas notas pode ser reproduzida, qualquer que seja a forma ou o meio, sem a permissão dos autores.

Os autores concedem a permissão explícita para a utilização e reprodução deste material no contexto do ensino de disciplinas regulares dos cursos de graduação sob a responsabilidade do Instituto de Computação da UNICAMP.

© Copyright 2000

Instituto de Computação
UNICAMP
Caixa Postal 6176
13083-970 Campinas-SP
{fkm,tomasz}@ic.unicamp.br

8 Processamento de Cadeias de Caracteres

Em praticamente todos os padrões de codificação, a representação interna das letras está agrupada e ordenada. Este é um dos motivos dos computadores poderem fazer comparações com letras, como 'A' < 'B' rapidamente, internamente é comparada sua codificação interna. Além disso, a codificação das letras e números é em geral seqüencial. Um exemplo disto é a codificação ASCII (veja tabela 1).

Como vimos, uma cadeia de caracteres pode ser definida com o tipo **string**. Além disso, podemos fazer operações com os caracteres que estão nesta string (cadeia). Considere as seguintes declarações:

```
type TipoTexto = string[1000];  
var Texto : TipoTexto;
```

Com estas declarações, podemos fazer a seguinte atribuição para a variável *Texto*: `Texto := 'Algoritmo';` Uma função que devolve o comprimento da cadeia de caracteres é a função **length**. Após a atribuição acima, **length**(*Texto*) deve retornar o inteiro 9. Além disso, podemos trabalhar com cada caracter deste texto, como se *Texto* fosse um vetor de 9 posições, i.e., podemos acessar e atualizar um elemento da cadeia. Assim, após a atribuição da variável *Texto* feita acima, temos em *Texto*[1] a letra 'A', *Texto*[2] a letra 'l', *Texto*[3] a letra 'g', ..., *Texto*[9] a letra 'o'.

Observações:

1. Em Turbo Pascal há uma limitação do tamanho máximo de uma String, de 255 caracteres. A declaração de variáveis usando apenas a palavra reservada "String" denota o tipo string com 255 caracteres. No Turbo Pascal a passagem de tipos é feita de maneira rígida, de tal forma que uma variável declarada como String[255] não pode ser enviada como um parâmetro declarado apenas como o tipo string.
2. Em Delphi, "String" denota normalmente o tipo "long string" ou "AnsiString" cujo limite de comprimento máximo é 2 GB! Com a diretiva de compilação \$H-\$, "String" denota o tipo "String[255]" como em Turbo Pascal. Outra alternativa equivalente é usar o tipo pré-definido "ShortString".
3. Em Gnu Pascal não é possível declarar variáveis usando apenas a palavra "String", mas na passagem de parâmetros e ponteiros sim. Desta maneira poderemos ter variáveis declaradas como strings de tamanhos diferentes mas que são recebidas como parâmetros do tipo String (sem especificação do tamanho).
4. Em Extended Pascal também não é possível se usar apenas a palavra String para declarar variáveis, além disso o tamanho da string é declarado usando se parenteses em vez de colchetes. Por exemplo, a seguinte declaração apresenta uma string em Extended Pascal com 100 caracteres.
var nome:string(100);

Nos exemplos deste texto usaremos a seguinte sintaxe:

- Na declaração de variáveis e tipos usaremos colchetes com o devido tamanho da string.
- Na passagem de parâmetros daremos prioridade para a sintaxe usando a palavra string. No caso onde a rotina tem funcionalidade fixa para um determinado tipo usaremos o nome do tipo no lugar da palavra string.

A seguir apresentamos algumas rotinas para manipulação de strings.

<i>Função</i>	<i>Resultado</i>
Length (<i>s</i>)	Retorna a quantidade de caracteres da string <i>s</i>
SetLength (<i>s</i> , <i>n</i>)	Redefine a quantidade de caracteres em uso como <i>n</i>
Pos (<i>s</i> ₁ , <i>s</i> ₂)	Retorna a posição da cadeia <i>s</i> ₁ na cadeia <i>s</i> ₂ (0 se não ocorre)
Concat (<i>s</i> ₁ , <i>s</i> ₂ , ...)	Retorna a concatenação de <i>s</i> ₁ , <i>s</i> ₂ , ... (equivalente a <i>s</i> ₁ + <i>s</i> ₂ + ...)
Copy (<i>s</i> , <i>p</i> , <i>n</i>)	Retorna a cadeia formada pelos <i>n</i> caracteres a partir da posição <i>p</i>
Insert (<i>s</i> ₁ , <i>s</i> ₂ , <i>p</i>)	Insere <i>s</i> ₁ em <i>s</i> ₂ a partir da posição <i>p</i>
Delete (<i>s</i> , <i>p</i> , <i>n</i>)	Remove os <i>n</i> caracteres a partir da posição <i>p</i>

Exemplo 8.1 *Descreva uma implementação em pascal da função Pos.*

```
function Pos(var s1,s2: String): Integer;  
var n1,n2,d,i,j : Integer; achou, cont: Boolean;  
begin  
  n1 := Length(s1); n2 := Length(s2); d := n2-n1+1;  
  achou := False;  
  i :=1;  
  while (not achou) and (i<=d) do  
  begin  
    j := 1; cont := True;  
    while cont and (j<=n1) do  
    begin  
      cont := (s1[j]=s2[i+j-1]);  
      inc(j)  
    end;  
    achou := cont;  
    inc(i);  
  end;  
  if achou then Pos := i-1  
  else Pos := 0  
end; {Pos}
```

Exemplo 8.2 *Descreva uma implementação em pascal da função Insert.*

```
function Insert(var s1,s2: String; p: Integer);  
{Insera os caracteres de s1 que couberem em s2}  
const ComprMax = 255;  
var n1,n2,d,i : Integer;  
begin  
  n1 := Length(s1); n2 := Length(s2);  
  if (n1+n2)>ComprMax  
  then d := ComprMax-n2  
  else d := n1;  
  i := 1;  
  for i:= n2 downto n2-d+1 do  
    s2[i+d] := s[i];  
  for i:=1 to d do  
    s2[p+i-1] := s1[i];  
  SetLength(s2,n2+d)  
end; {Insera}
```

Exemplo 8.3 *Uma cadeia de caracteres é dita ser palíndrome se a seqüência dos caracteres da cadeia da esquerda para a direita é igual a seqüência de caracteres da direita para a esquerda. Por exemplo: As seguintes cadeias de caracteres são palíndromes: ABC12321CBA, ACCA, XYZ6.6ZYX.*

Faça uma função que retorna verdadeiro se a cadeia de caracteres enviada como parâmetro é palíndrome.

```

function Palindrome(var s: String): Boolean;
var i,m,n : Integer; p: Boolean;
begin
  p := True;   n := Length(s);
  m := n div 2;   i := 1;
  while p and (i<=m) do begin
    p := (s[i]=s[n-i+1]);
    inc(i);
  end;
  Palindrome := p
end; {Palindrome}

```

8.1 Letras com Acêntos

Em muitas situações temos cadeias de caracteres seguindo uma codificação para acentuação das letras (eg. textos escritos na língua portuguesa). As codificações de acentuação não são consideradas na codificação ASCII e na maioria das vezes não seguem uma padronização envolvendo diferentes plataformas (i.e., uma letra acentuada em um ambiente Unix pode ter código diferente da mesma letra acentuada em ambiente MS-DOS).

Uma tarefa comum que temos de fazer, quando consideramos acêntos, é a remoção destes do texto. A seguir apresentamos duas aplicações onde precisamos fazer um pré-processamento para remover os acentos:

1. *Envio de mensagens pela internet por correio eletrônico.*

A internet é uma rede de computadores que não necessariamente seguem a mesma configuração. Os textos como as mensagens enviadas por correio eletrônico são enviadas considerando a configuração ASCII. Por outro lado, letras acentuadas em um computador podem não ser consideradas realmente letras em outros computadores. Assim, estes podem ser interpretados como códigos de ação totalmente imprevisível. Uma solução para isso é eliminar os acentos, ou trocá-los por uma seqüência de letras ASCII que representem a letra acentuada.

2. *Ordenação de cadeias de caracteres.*

Para comparar duas cadeias de caracteres, a maioria dos programas considera as representações binárias caracter a caracter. Considerando a codificação ASCII, a ordenação de letras em ASCII se torna fácil, uma vez que temos apenas que comparar letra a letra nesta codificação. Por outro lado, teremos uma ordenação errada caso tenhamos acentos.

Por exemplo, considere as palavras *macro*, *macaco* e *maço*. Se a codificação de 'ç' vier antes de 'c', então teremos a seguinte ordem para estas palavras: *maço*, *macaco* e *macro*, que não é a ordem que desejamos. Por outro lado, se a codificação de 'c' vier antes de 'ç', então teremos a ordem: *macaco*, *macro* e *maço*, que também não é a desejada. Assim, uma solução para isto é comparar as palavras trocando as letras acentuadas pelas correspondentes letras sem acento.

Exemplo 8.4 *O seguinte programa contém uma função que dado uma letra, retorna a letra sem acentuação.*

Obs.: Esta implementação é dependente da codificação das letras usada no programa fonte.

```

program acentos;
function SemAcento(caracter : char):char;
begin
  case caracter of
    'á' : SemAcento:= 'a';      'Á' : SemAcento:= 'A';      'é' : SemAcento:= 'e';
    'É' : SemAcento:= 'E';      'Í' : SemAcento:= 'i';      'Í' : SemAcento:= 'I';
    'ó' : SemAcento:= 'o';      'Ó' : SemAcento:= 'O';      'ú' : SemAcento:= 'u';
    'Ú' : SemAcento:= 'U';      'à' : SemAcento:= 'a';      'À' : SemAcento:= 'A';
    'â' : SemAcento:= 'a';      'Â' : SemAcento:= 'A';      'ê' : SemAcento:= 'e';
    'Ê' : SemAcento:= 'E';      'ô' : SemAcento:= 'o';      'Ô' : SemAcento:= 'O';
    'ã' : SemAcento:= 'a';      'Ã' : SemAcento:= 'A';      'õ' : SemAcento:= 'o';
    'Õ' : SemAcento:= 'O';      'ü' : SemAcento:= 'u';      'Ü' : SemAcento:= 'U';
    'ç' : SemAcento:= 'c';      'Ç' : SemAcento:= 'C';
  else SemAcento:=caracter;
  end; { case }
end;
var c : char;
begin
  c:= 'á'; writeln('A letra ',c,' sem acento fica ',SemAcento(c));
  c:= 'é'; writeln('A letra ',c,' sem acento fica ',SemAcento(c));
  c:= 'ã'; writeln('A letra ',c,' sem acento fica ',SemAcento(c));
  c:= 'õ'; writeln('A letra ',c,' sem acento fica ',SemAcento(c));
end.

```

Exercício 8.1 *Faça uma função que tem como parâmetro uma cadeia de caracteres e retorna a mesma cadeia sem acentos.*

8.2 Transformação entre Maiúsculas e Minúsculas

Uma maneira de se transformar uma cadeia de caracteres trocando cada letra por sua correspondente em maiúscula é percorrer toda a cadeia e para cada caracter colocar uma seqüência de testes, um para cada letra minúscula, e trocá-la pela correspondente letra maiúscula. Um algoritmo deste tipo iria requerer uma estrutura com pelo menos 25 condições, tornando o programa longo e lento.

Uma maneira mais eficiente de se implementar tal procedimento, é considerar a representação interna de cada caracter. Considerando que internamente cada letra é representada em um byte, e a representação das letras é seqüencial a idéia é mudar apenas aqueles caracteres que estiverem no intervalo [*a*,...*z*]. Quando ocorrer um caracter *c* neste intervalo, obtemos o valor inteiro em ASCII de *c* e seu deslocamento *d* a partir do caracter '*a*'. Em seguida, reatribuímos o caracter que estiver na posição de '*A*' mais o deslocamento *d*.

```

program ProgramMaiusculas;
type TipoTexto = string[1000];
procedure Maiuscula(var texto : TipoTexto);
var i,tam : integer;
begin
  tam := length(texto);
  for i:=1 to tam do
    if ('a' <= texto[i] and (texto[i] <= 'z')) then
      texto[i] := chr( ord('A') + ord(texto[i]) - ord('a') );
end;
var Cadeia : TipoTexto;
begin
  write('Entre com um texto: ');
  readln(cadeia);
  Maiuscula(cadeia);
  writeln('O texto em maiúsculo é :',cadeia);
end.

```

Exercício 8.2 *Faça um programa análogo ao apresentado acima, mas para transformar uma cadeia em minúsculas.*

8.3 Casamento de Padrões

Nesta seção vamos construir uma função para que dados duas cadeia de caracteres, uma chamada *texto*, e a outra chamada *padrão*, verifica se existe uma ocorrência de *padrão* no *texto*. Caso ocorra uma ocorrência, a função retorna a posição no *texto* onde ocorre o *padrão*, caso contrário, a função retorna 0.

Uma idéia simples de implementação é ir percorrendo todas as posições possíveis de *texto*, de se começar o *padrão* (i.e., posições $1, \dots, \text{length}(\text{texto}) - \text{length}(\text{padrao}) + 1$). Para cada uma destas posições, há uma outra estrutura de repetição que verifica se o padrão está começando naquela posição. A função para assim que encontrar o primeiro padrão, ou até que todas as possibilidades tenham sido testadas. O seguinte programa implementa esta idéia.

```

program ProgramaBuscaPadrao;

type TipoString    = string[1000];
    TipoPadrao      = string[50];

{Retorna a posição do índice onde começa o padrao, 0 caso não exista. }
function BuscaPadrao(var texto : TipoString; var Padrao:TipoPadrao): integer;
var
    i,j,TamTexto,TamPadrao : integer;
    achou,SubsequenciaIguar : boolean;
begin
    TamTexto := length(texto);
    TamPadrao := length(padrao);
    i := 0;
    achou := false;
    while (not achou) and (i<=TamTexto-TamPadrao) do begin
        i:=i+1;
        j:=1;
        SubSequenciaIguar := true;
        while (j<=TamPadrao) and (SubsequenciaIguar=true) do
            if (Padrao[j]=Texto[i+j-1]) then j:=j+1
            else SubSequenciaIguar := false;
        achou := SubSequenciaIguar;
    end;
    if (achou) then BuscaPadrao := i
    else BuscaPadrao :=0;
end; { BuscaPadrao }

var texto : TipoString;
    padrao : TipoPadrao;
    pos    : integer;
begin
    write('Entre com um texto: ');
    readln(texto);
    write('Entre com um padrao: ');
    readln(padrao);
    pos := BuscaPadrao(texto,padrao);
    if (pos=0) then writeln('Padrao não encontrado.')
    else writeln('Padrao encontrado na posição ',pos);
end.

```

Obs.: Existem outros algoritmos para fazer busca de padrões que são computacionalmente mais eficientes, como o algoritmo de Knuth, Morris e Pratt e o algoritmo de Boyer e Moore.

8.4 Criptografia por Substituições - Cifra de César

Criptografia é a ciência e o estudo de escrita secreta. Um *sistema criptográfico* é um método secreto de escrita pelo qual um texto legível é transformado em texto cifrado. O processo de transformação é conhecido como *ciframento* e a transformação inversa é conhecida como *deciframento*.

A idéia aqui não é a de apresentar métodos seguros de criptografia, mas sim de trabalhar mais com as cadeias de caracteres.

Provavelmente os primeiros métodos de criptografia usavam métodos de substituição. Dado dois alfabetos \mathcal{A} e \mathcal{A}' , uma função de criptografia por substituição troca um caracter de um texto \mathcal{A} por outra em \mathcal{A}' . Naturalmente deve haver uma função inversa para que o receptor da mensagem criptografada possa recuperar o texto original.

Cifra de César

Um caso particular do método de substituição é o seguinte. Considere um alfabeto $\mathcal{A} = (c_0, c_1, \dots, c_{n-1})$ com n símbolos. Considere um inteiro k , $0 \leq k \leq n - 1$ (chamado de chave). Uma função de criptografia $f_k : \mathcal{A} \rightarrow \mathcal{A}$ e sua inversa f_k^{-1} (para decifrar) são dadas a seguir:

$$f_k(c_i) = c_{(i+k) \bmod n} \quad f_k^{-1}(c_i) = c_{(i-k+n) \bmod n}$$

Assim, se o alfabeto é $\mathcal{A} = (A, B, C, D, E, F, G, H, \dots, S, T, U, V, W, X, Y, Z)$, e $k = 3$ a função transforma A em D , B em E , C em F , \dots . I.e.,

A	B	C	D	E	F	G	H	...	V	W	X	Y	Z
↓	↓	↓	↓	↓	↓	↓	↓	...	↓	↓	↓	↓	↓
D	E	F	G	H	I	J	K	...	Y	Z	A	B	C

Este tipo de criptografia também é chamado de cifra de César, porque Júlio César usou com $k = 3$. A seguir apresentamos um programa que criptografa um texto, com o alfabeto das letras minúsculas, usando este tipo de criptografia, transformando apenas as letras e deixando os demais símbolos intactos.

```
program progcifrasesar;
type tipostring = string[100];
procedure criptocesar(var texto,cifra : tipostring; k:integer);
var i,tam : integer;
begin
  cifra := ' ';
  tam := length(texto);
  for i:=1 to tam do begin
    if (texto[i]>='a') and (texto[i]<='z') then
      cifra := cifra + chr((ord(texto[i]) - ord('a') + k) mod 26 + ord('a'))
    else cifra := cifra + texto[i];
  end;
end; { criptocesar }
var Texto,Cripto : tipostring;
begin
  write('Entre com uma cadeia de caracteres: ');
  readln(texto);
  criptocesar(texto,cripto,3);
  writeln('A texto criptografado é: ',cripto);
end.
```

Exercício 8.3 Faça um procedimento para decriptografar tendo como dados de entrada a cadeia de caracteres criptografada pelo programa acima e o valor usado de k .

Os métodos de substituição são rápidos, mas pouco seguros. Em geral, cada língua possui frequências diferentes para as ocorrências de cada letra.

Exemplo 8.5 Em um texto em inglês, temos em geral as seguintes frequências esperadas para as letras:

$A = 8,0\%$	$J = 0,5\%$	$S = 6,0\%$
$B = 1,5\%$	$K = 0,5\%$	$T = 9,0\%$
$C = 3,0\%$	$L = 3,5\%$	$U = 3,0\%$
$D = 4,0\%$	$M = 3,0\%$	$V = 1,0\%$
$E = 13,0\%$	$N = 7,0\%$	$W = 1,5\%$
$F = 2,0\%$	$O = 8,0\%$	$X = 0,5\%$
$G = 1,5\%$	$P = 2,0\%$	$Y = 2,0\%$
$H = 6,0\%$	$Q = 0,2\%$	$Z = 0,2\%$
$I = 6,5\%$	$R = 6,5\%$	

Assim, para decriptografar um texto, onde o original é em inglês, calculamos as frequências dos símbolos no texto criptografado e em seguida relacionamos com as frequências acima. Isto permite que possamos relacionar os símbolos com os caracteres originais mais prováveis.

Exercício 8.4 Faça um programa que lê um número inteiro n , $0 \leq n \leq 1000000$ e escreve seu valor por extenso. Exemplos:

- 1 = um
- 19 = dezenove
- 20 = vinte
- 21 = vinte e um
- 100 = cem
- 111 = cento e onze
- 1000 = mil
- 2345 = dois mil trezentos e quarenta e cinco
- 9012 = nove mil e doze
- 12900 = doze mil e novecentos
- 100101 = cem mil cento e um
- 110600 = cento e dez mil e seiscentos
- 999999 = novecentos e noventa e nove mil novecentos e noventa e nove
- 1000000 = um milhão

Sugestão: Escreva funções ou procedimentos convenientes que tratam de números:

- até 9
- até 19
- até 99
- até 999
- até um milhão

Note que as funções para intervalos maiores podem usar as funções para intervalos menores. Pense com cuidado na lógica do uso da conjunção 'e'.

8.5 Exercícios

1. Um programador está implementando um processador de textos e quer adicionar uma facilidade para a composição de cartas, colocando um comando que com apenas os dados da data (dia, mês e ano), apresenta o seguinte cabeçalho:

<Dia da semana>, <dia> de <mês> de <ano>

Ex.: Colocando a data 01/01/1901, temos o seguinte cabeçalho:

Terça-feira, 1 de janeiro de 1901

Para implementar esta facilidade, ajude este programador construindo uma função que tem como parâmetros o *dia*, o *mês* e o *ano* e retorna uma string contendo o cabeçalho como no exemplo acima. Considere que as datas estão no intervalo de 01/01/1901 a 01/01/2099.

2. Pelo calendário gregoriano, intituído em 1582 pelo Papa Gregório XIII, os anos bisextos são aqueles cujo ano são divisíveis por 4, exceto os anos que são divisíveis por 100 e não por 400. Por exemplo, os anos 1600, 2000 são anos bisextos enquanto os anos 1700, 1800 e 1900 não são. Com isto, em mente, resolva o exercício anterior para considerar anos a partir de 1600.
Obs.: O dia 01/01/1600 é um sábado.
3. O programa apresentado na seção 8.4 restringe a criptografia ao alfabeto $\{a, \dots, z\}$. Faça um programa para criptografar e decriptografar considerando o alfabeto formado pelo tipo **byte**.
4. Faça um programa contendo funções com os seguintes cabeçalhos:

- **function** criptografa(str : string; k:integer):tipostring;
- **function** decriptografa(str : string; k:integer):tipostring;
- **function** maiuscula(str : string):tipostring;
- **function** minuscula(str : string):tipostring;
- **function** semacentos(str : string):tipostring;
- **function** cadeialivre(str : string):tipostring;

onde *tipostring* é definido como:

type tipostring=string[255];

As funções criptografa e decriptografa são para criptografar e decriptografar usando a cifra de César, com deslocamento k e o alfabeto das letras minúsculas. As funções maiúscula e minuscula são para retornar a função dada em maiúsculo e minúsculo. A função semacentos retorna a cadeia de caracteres str sem as letras dos acentos (ela preserva maiúsculas e minúsculas). A função cadeia livre transforma a cadeia de caracteres str em uma cadeia sem acentos e com todas as letras em minúsculo.

O programa deve ler uma cadeia de caracteres e uma opção para uma das seis operações acima. Caso a opção seja para criptografar ou decriptografar, o parâmetro k também deve ser lido. Após a execução de uma opção, a cadeia de caracteres resultante deve ser impressa.