Fundação CECIERJ - Vice Presidência de Educação Superior a Distância
Curso de Tecnologia em Sistemas de Computação
Disciplina: Programação I
AD2 1^o **semestre de 2016.**

---

1 Questão Única

A busca de padrões de texto dentro de strings é um problema de importância intrínseca. Algoritmos para casamento exato de strings, onde uma string P occorre como uma substring da string T são bastante conhecidos. Infelizmente, a vida não é tão simples assim. Palavras em um texto ou num padrão de busca podem estar soletradas erradas, e portanto, similaridade exata não irá acontecer. Mudanças evolucionárias em sequências de genomas, ou no uso de uma linguagem, implicam que buscas devem ser executadas com padrões arcaicos em mente: "Thou shalt not kill" mudou ao longo do tempo para "You should not murder."

Como podemos procurar pela substring mais próxima para compensar por erros de grafia? Para lidar com o casamento inexato de strings, devemos definir primeiro uma função de custo, que informe o quão longe duas strings estão, ou seja, uma medida de distância entre pares de strings. Uma medida razoável deve refletir o número de movimentos que precisam ser executados para levar uma string na outra. Existem três tipos naturais de mudanças:

- Substituição - Troca um único caracter de um padrão P por um caracter no texto T, como, por exemplo, mudando "shot" por "spot."
- Inserção - Insere um único caracter no padrão P para auxiliar no casamento com o texto T, por exemplo, trocando "ago" para "agog."
- Deleção - Remove um único caracter do padrão P para auxiliar no casamento com o texto T, por exemplo, trocando "hour" por "our."

A correta colocação do problema da similaridade entre strings, requer o uso de uma função de custo para cada uma dessas operações de transformação de strings. Associando-se cada operação ao mesmo custo unitário (igual a 1), define-se uma distância entre duas strings.

O casamento aproximado de strings surge em várias aplicações importantes. Embora pareça ser um problema difícil, porque devemos decidir exatamente onde deletar e inserir (potencialmente) vários caracteres no padrão e no texto. Porém, pode-se pensar no problema reverso: qual informação seria necessária para tomar a decisão final? O que pode acontecer com o último caracter de cada string, durante o casamento? Considere o programa recursivo abaixo:

```
Program strMatching;

Type
  OperType = (UNDEFINED, MATCH, INSERT, DELETE);

cell = record
    /// cost of reaching this cell
    cost: integer;
    /// parent cell
    parent: OperType;
end;

var str1, str2, str3, str4: String;
    dist: Integer;
    /// dynamic programming table
    m: Array [0..MAXLEN, 0..MAXLEN] of cell;

/// Returns the cost of removing a character.
function indel (c: char): integer;
begin
    indel := 1;
end;

/// Returns the cost of two matching characters as 0, and 1 otherwise.
function char_match (c: char; d: char): Integer;
begin
    if ( c <> d ) then char_match := 1
    else char_match := 0;
end;

(** This program is absolutely correct - convince yourself.
 *  It also turns out to be impossibly slow. Running on my
 *  computer, the computation takes several seconds to compare
 *  two 11-character strings, and disappears into Never-Never Land
 *  on anything longer.
 *  Why is the algorithm so slow? It takes exponential time because
 *  it recomputes values again and again and again.
 *  At every position in the string, the recursion branches three ways,
 *  meaning it grows at a rate of at least 3^n - indeed, even faster
 *  since most of the calls reduce only one of the two indices,
 *  not both of them.
 *
 *  @param s original string
 *  @param t template string
 *  @param i position of s's last character.
 *  @param j position of t's last character.
 *)
```

```
function string_compare (s: String; t: String; i: integer; j: integer): intege
var
    k: OperType;
    opt: Array [OperType] of Integer;
    lowest_cost: integer;

begin
    // i is zero, then the cost is removing the remaining j characters
    if ( i = 0 ) then
    begin
        string_compare := j*indel(' ');
        exit
    end;
    // j is zero, then the cost is removing the remaining i characters
    if ( j = 0 ) then
    begin
        string_compare := i*indel(' ');
        exit
    end;

    opt[MATCH]  := string_compare(s,t,i-1,j-1) + char_match(s[i],t[j]);
    opt[INSERT] := string_compare(s,t,i,j-1)   + indel(t[j]);
    opt[DELETE] := string_compare(s,t,i-1,j)   + indel(s[i]);

    lowest_cost := opt[MATCH];
    for k := INSERT to DELETE do begin
        if ( opt[k] < lowest_cost ) then lowest_cost := opt[k]
    end;
    string_compare := lowest_cost;
end;

begin
    str1 := 'GAATTCAGTTA';
    str2 := 'GGATCGA';
    str3 := 'thou-shalt-not';
    str4 := 'you-should-not';
    dist := string_compare (str1, str2, length(str1), length(str2));
    writeln ( dist, ' moves' );
    \\dist := string_compare (str3, str4, length(str3), length(str4));
    \\writeln ( dist, ' moves' );
end.
```

1. O programa ”strMatching” acima, embora funcione, é extremamente lento. Sua tarefa é escrever um algoritmo, baseado em programação dinâmica, para executar o casamento inexato de duas strings quaisquer.

2. Usando o ”array m”, reconstrua as decisões tomadas, andando para trás a partir do último estado, e seguindo o ponteiro ”parent” para uma célula anterior.

   O processo é repetido até que se chegue a célula inicial. O campo ”parent” na posição m[i,j] indica quando a operação (i, j) foi um MATCH, INSERT, ou DELETE.

   Por exemplo, usando as strings ”thou-shalt-not” e ”you-should-not”, obtemos: $DSMMMMMISMSMMMM$

   - D(t)S(h->y)M(o)M(u)M(-)M(s)M(h)I(o)S(a->u)M(l)S(t->d)M(-)M(n)M(o)M(t)
   - deleta o primeiro ”t”,
   - troca o ”h”por ”y”,
   - casa os próximos cinco caracteres antes de
   - inserir um ”o”,
   - troca ”a”por ”u”,
   - casa ”l”, e
   - finalmente troca o ”t” por um ”d” e
   - casa os quatro últimos caracteres.

   Andando para trás, reconstói-se a solução na ordem inversa. No entanto, um uso esperto de recursão, pode produzir o resultado já invertido.

---

1. Solução

O programa recursivo fornecido, embora funcione, ou seja, retorne o custo da transformação de uma string na outra, possui um problema sério: o número de chamadas recursivas é proibitivo para strings relativamente pequenas.

Novamente, a programação dinâmica é a solução. O que deve ser feito é usar o array ”m”para guardar os resultados das chamadas recursivas anteriores. Senão, o programa recursivo não consegue terminar, se as strings tiverem tamanho maior do que 11.

A reconstrução das decisões tomadas pode ser conseguida apenas acessando o OperType armazenado em cada posição da matriz m. Com essa informação, as duas strings e dois índices inteiros, pode-se implementar um procedimento recursivo que imprimirá o tipo de operação e as letras envolvidas.

Para a interface pode-se utilizar um TMemo.