


Galil-Seiferas, Optimal Mismatch, Two-Way, String Matching on Ordered Alphabets



Integrantes da Equipe

- Gonzalo Ivan dos Santos Portales;
- Gabriel Sena San Martin;
- Victor Yan Pereira e Lima.

Galil-Seiferas

Definição

- É um algoritmo de correspondência de string com espaço constante e tempo linear;
- Criado por Zvi Galil e Joel Seiferas;
- Seu principal caso de uso é quando não é possível ordenar alfabeticamente;
- Complexidade: $O(|x| + |y|)$, sendo:
 - $x \Rightarrow$ o padrão a ser encontrado;
 - $y \Rightarrow$ a string onde a busca será realizada.

Sobre o Algoritmo

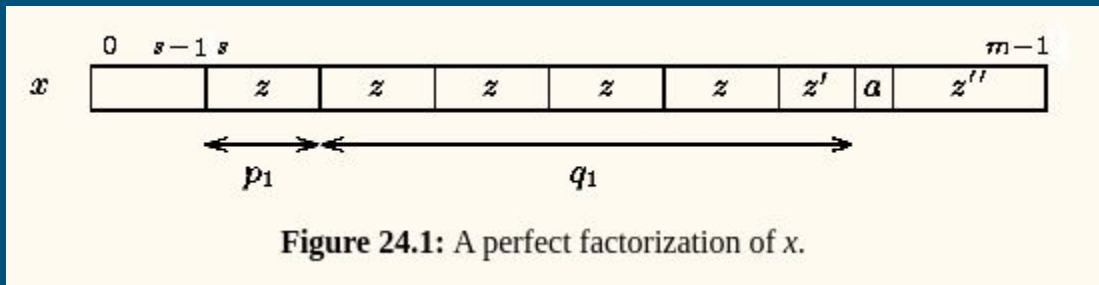
- O algoritmo é constituído por duas fases: pré-processamento (funções newP1, newP2 e parse) e busca (função search);
- O objetivo da parte de pré-processamento é achar uma fatoração perfeita uv de x onde $u = x[0 .. s-1]$ e $v = x[s .. m-1]$;
- A função newP1 tem a finalidade de achar o período de prefixo mais curto de $x[s .. m-1]$.

Sobre o Algoritmo

- A função `newP2` é responsável por achar o segundo período de prefixo mais curto de $x[s .. m-1]$;
- A função `parse` é responsável por incrementar a variável global `s`;
- Dessa forma, antes de chamar a função `search`, temos que:
 - $x[s .. m-1]$ tem, no máximo, um período do prefixo;
 - se $x[s .. m-1]$ tem um período do prefixo, então seu tamanho é $p1$;
 - $x[s .. s+p1+q1-1]$ tem o período mais curto de tamanho $p1$;
 - $x[s .. s+p1+q1]$ não tem período de tamanho $p1$.

Sobre o Algoritmo

O padrão x está na forma $x[0 \dots s-1]x[s \dots m-1]$ onde $x[s \dots m-1]$ está na forma $z|z'az''$, com $|z| = p_1$, z' é prefixo de z , a não é prefixo de z e $|z|z'| = p_1 + q_1$



Sobre o Algoritmo

- Dessa forma, ao pesquisar por $x[s \dots m-1]$ dentro de y :
 - se $x[s \dots s+p_1+q_1-1]$ foi correspondido, uma mudança de comprimento p_1 pode ser realizada e as comparações são retomadas a $x[s+q_1]$;
 - caso contrário, se uma não-correspondência ocorrer em $x[s+q]$ sendo q diferente de p_1+q_1 , então uma mudança de tamanho $q/k+1$ deve ser realizada e as comparações são retomadas com $x[0]$.

Vantagens

- Utiliza espaço constante em sua implementação;
- Realiza ambos os limites assintóticos simultaneamente;
- Elimina completamente a necessidade de uma função “falha” prevista nos algoritmos de tempo linear de Knuth, Morris and Pratt;
- Pode ser implementado como uma sub-rotina Fortran ou mesmo como um autômato finito de 6 cabeças.

Código-fonte

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  char *x, *y;
6  int k, m, n, p, p1, p2, q, q1, q2, s;
7
8  /* Função utilizada para pesquisar o texto por x = [ 0, s ], [ s, |x| ] */
9  void search() {
10     while (p ≤ n - m) {
11         while (p + s + q < n && x[s + q] == y[p + s + q]) ++q;
12
13         if (q == m - s && memcmp(x, y + p, s + 1) == 0) printf("%d\n", p);
14
15         if (q == p1 + q1) {
16             p += p1;
17             q -= p1;
18         }
19         else {
20             p += (q/k + 1);
21             q = 0;
22         }
23     }
24 }
```

```
26 /* Função utilizada para incrementar a variável global s */
27 void parse() {
28     while (1) {
29         while (x[s + q1] == x[s + p1 + q1]) ++q1;
30
31         while (p1 + q1 ≥ k * p1) {
32             s += p1;
33             q1 -= p1;
34         }
35
36         p1 += (q1 / k + 1);
37         q1 = 0;
38
39         if (p1 ≥ p2) break;
40     }
41     newP1();
42 }
43 }
```

Código-fonte

```
45  /* Função utilizada para encontrar o segundo período de prefixo mais curto de [ s, |x| ]. */
46  void newP2() {
47      while (x[s + q2] == x[s + p2 + q2] && p2 + q2 < k * p2) ++q2;
48
49      if (p2 + q2 == k * p2) parse();
50      else if (s + p2 + q2 == m) search();
51      else {
52          if (q2 == p1 + q1) {
53              p2 += p1;
54              q2 -= p1;
55          }
56
57          else {
58              p2 += (q2 / k + 1);
59              q2 = 0;
60          }
61      }
62      newP2();
63  }
64 }
```

Código-fonte

```
66  /* Função utilizada para encontrar o período de prefixo mais curto de [ s, |x| ]. */
67  void newP1() {
68      while(x[s + q1] == x[s + p1 + q1]) ++q1;
69
70      if(p1 + q1 ≥ k * p1) {
71          p2 = q1;
72          q2 = 0;
73          newP2();
74      }
75      else {
76          if(s + p1 + q1 == m) search();
77          else {
78              p1 += (q1/k + 1);
79              q1 = 0;
80              newP1();
81          }
82      }
83  }
```

Código-fonte

```
85  /* Método inicial com a inicialização das variáveis globais e chamada do método newP1 */
86  void GS(char *argX, int argM, char *argY, int argN) {
87      x = argX;
88      m = argM;
89      y = argY;
90      n = argN;
91      k = 4;
92      p = q = s = q1 = p2 = q2 = 0;
93      p1 = 1;
94      newP1();
95  }
```

Questão do URI - 2651 - Link Bolado

```
7 char *x, *y;
8 int k, m, n, p, p1, p2, q, q1, q2, s;
9 int found;
10
11 void newP1();
12
13 /* Função utilizada para pesquisar o texto por x = [ 0, s ], [ s, |x| ] */
14 void search() {
15     while (p ≤ n - m) {
16         while (p + s + q < n && x[s + q] == y[p + s + q]) ++q;
17
18         if (q == m - s && memcmp(x, y + p, s + 1) == 0) found = 1;
19
20         if (q == p1 + q1) {
21             p += p1;
22             q -= p1;
23         }
24         else {
25             p += (q/k + 1);
26             q = 0;
27         }
28     }
29 }
```

```
102 int main() {
103     string input; cin >> input;
104
105     string aux = "";
106
107     for(int i = 0; i < input.length(); i++) {
108         aux += tolower(input.at(i));
109     }
110
111     int n = aux.length();
112
113     char inputArray[n + 1];
114
115     strcpy(inputArray, aux.c_str());
116
117     GS("zelda", 5, inputArray, n);
118
119     cout << "Link " << (found ? "Bolado" : "Tranquilo") << endl;
120
121     return 0;
122 }
123
```

| | |
|------------|---|
| PROBLEMA: | 2651 - Link Bolado |
| RESPOSTA: | Accepted |
| LINGUAGEM: | C++17 (g++ 7.3.0, -std=c++17 -O2 -lm) [+0s] |
| TEMPO: | 0.000s |

Optimal Mismatch

Definição

- Autor: SUNDAY D.M., 1990.
- Um algoritmo de correspondência de string que compara os caracteres mais raros primeiro. Quando um caractere não corresponde, o próximo caractere no texto além da string de pesquisa determina onde a próxima correspondência possível começa.

Sobre o algoritmo

- Sunday projetou um algoritmo onde os caracteres do padrão são escaneados do menos frequente ao mais frequente. Fazendo isso, pode-se esperar que haja uma incompatibilidade na maioria das vezes e, assim, digitalizar todo o texto muito rapidamente.
- É preciso saber as frequências de cada um dos caracteres do alfabeto.
- Variante do algoritmo Quick Search;

Sobre o algoritmo

- A fase de pré-processamento do algoritmo Optimal Mismatch consiste em classificar os caracteres padrão em ordem decrescente de suas frequências.
- Em seguida, construir a função de deslocamento de *bad character* da Quick Search e uma função de deslocamento de *good suffix* adaptada à ordem de varredura dos caracteres do padrão.

Complexidade

- A fase de pré-processamento possui complexidade de $O(m^2 + \sigma)$ em tempo e $O(m + \sigma)$ em espaço;
- A fase de busca possui complexidade de $O(mn)$ em tempo.

Exemplo Gráfico

Fase de pré-processamento

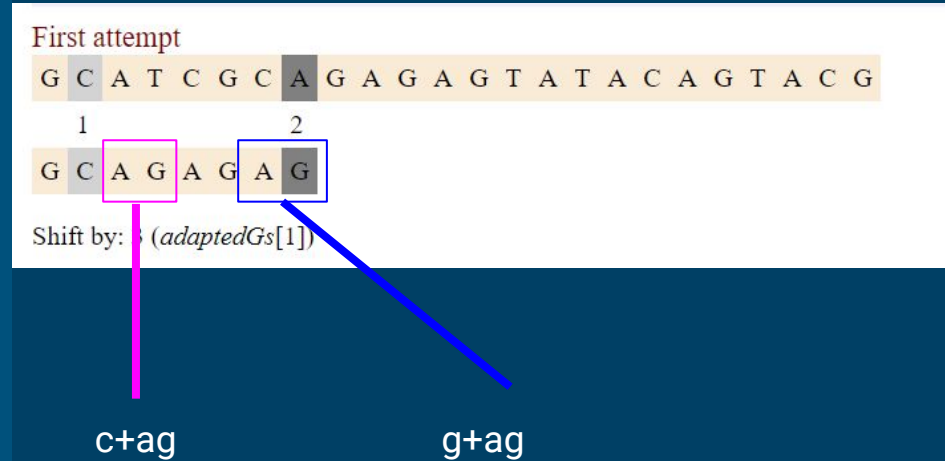
| <i>c</i> | A | C | G | T |
|----------------|---|---|---|---|
| <i>freq[c]</i> | 8 | 5 | 7 | 4 |
| <i>qsBc[c]</i> | 2 | 7 | 1 | 9 |

| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------------------|---|---|---|---|---|---|---|---|
| <i>x[i]</i> | G | C | A | G | A | G | A | G |
| <i>pat[i].loc</i> | 1 | 7 | 5 | 3 | 0 | 6 | 4 | 2 |
| <i>pat[i].c</i> | C | G | G | G | G | A | A | A |

| <i>i</i> | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---------------------|---|---|---|---|---|---|---|---|---|
| <i>adaptedGs[i]</i> | 1 | 3 | 4 | 2 | 7 | 7 | 7 | 7 | 7 |

Exemplo Gráfico

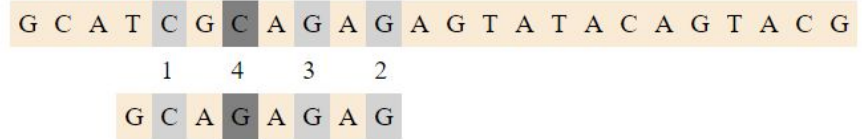
Fase de busca



Exemplo Gráfico

Fase de busca

Second attempt



Shift by: 2 ($qsBc[A] = adaptedGs[3]$)

Exemplo Gráfico

Fase de busca

First attempt

G C A T C G C A G A G A G T A T A C A G T A C G
1 2
G C A G A G A G

Shift by: 3 ($adaptedGs[1]$)

Second attempt

G C A T C G C A G A G A G T A T A C A G T A C G
1 4 3 2
G C A G A G A G

Shift by: 2 ($qsBc[A]=adaptedGs[3]$)

Third attempt

G C A T C G C A G A G A G T A T A C A G T A C G
5 1 8 4 7 3 6 2
G C A G A G A G

Shift by: 9 ($qsBc[T]$)

Fourth attempt

G C A T C G C A G A G A G T A T A C A G T A C G
1
G C A G A G A G

Shift by: 7 ($qsBc[C]$)

The Optimal Mismatch algorithm performs 15 character comparisons on the example.

Código-fonte

```
#include <stdio.h>
#include <string.h>
#define ASIZE 100000
#define XSIZE 100000

typedef struct patternScanOrder {
    int loc;
    char c;
} pattern;

int freq[ASIZE];

// Constrói a tabela de troca de bad character
void preQsBc(char *x, int m, int qsBc[]) {
    int i;

    for (i = 0; i < ASIZE; ++i) qsBc[i] = m + 1;
    for (i = 0; i < m; ++i) qsBc[x[i]] = m - i;
}
```

```
/* Constrói a tabela de troca de bons sufixos
   a partir de uma string ordenada. */
void preAdaptedGs(char *x, int m,
                  int adaptedGs[], pattern *pat) {
    int lshift, i, ploc;

    adaptedGs[0] = lshift = 1;
    for (ploc = 1; ploc ≤ m; ++ploc) {
        lshift = matchShift(x, m, ploc, lshift, pat);
        adaptedGs[ploc] = lshift;
    }
    for (ploc = 0; ploc ≤ m; ++ploc) {
        lshift = adaptedGs[ploc];
        while (lshift < m) {
            i = pat[ploc].loc - lshift;
            if (i < 0 || pat[ploc].c ≠ x[i]) break;
            ++lshift;
            lshift = matchShift(x, m, ploc, lshift, pat);
        }
        adaptedGs[ploc] = lshift;
    }
}
```


Código-fonte

```
/* Constrói um pattern ordenado
   a partir de uma string. */
void orderPattern(char *x, int m,
                  int (*pcmp)(), pattern *pat) {
    int i;

    for (i = 0; i ≤ m; ++i) {
        pat[i].loc = i;
        pat[i].c = x[i];
    }
    qsort(pat, m, sizeof(pattern), pcmp);
}

/* função de comparação de
   patterns do Optimal Mismatch. */
int optimalPcmp(pattern *pat1, pattern *pat2) {
    float fx;

    fx = freq[pat1→c] - freq[pat2→c];
    return (fx ?
           (fx > 0 ? 1 : -1) :
           (pat2→loc - pat1→loc));
}
```

```
/* Encontrar o próximo deslocamento para a esquerda
   para os primeiros elementos do padrão ploc após
   uma troca atual(shift) ou lshift */
int matchShift(char *x, int m, int ploc,
               int lshift, pattern *pat) {
    int i, j;

    for (; lshift < m; ++lshift) {
        i = ploc;
        while (--i ≥ 0) {
            if ((j = (pat[i].loc - lshift)) < 0) continue;
            if (pat[i].c ≠ x[j]) break;
        }
        if (i < 0) break;
    }
    return (lshift);
}
```

Código-fonte

```
/* Algoritmo de correspondência de strings Optimal Mismatch. */
void OM(char *x, int m, char *y, int n) {
    int i, j, adaptedGs[XSIZE], qsBc[ASIZE];
    pattern pat[XSIZE];
    /* Preprocessing */
    orderPattern(x, m, optimalPcmp, pat);
    preQsBc(x, m, qsBc);
    preAdaptedGs(x, m, adaptedGs, pat);
    /* Searching */
    j = 0;
    while (j ≤ n - m) {
        i = 0;
        while (i < m && pat[i].c == y[j + pat[i].loc]) ++i;
        if (i ≥ m) printf("%d\n", j);
        int max = adaptedGs[i] > qsBc[y[j + m]] ? adaptedGs[i] : qsBc[y[j + m]];
        j += max;
    }
}
```

Questão do URI - 3300

Números Má Sorte Recarregados

Um número número 3 é de má sorte se contém um 1 seguido por um 3 entre seus dígitos. Por exemplo, o número 341329 é de má sorte, enquanto o número 26771 não é.

Dado um inteiro N , seu programa terá que determinar se N é azarado ou não.

Entrada

A entrada consiste em um número positivo N ($0 \leq N \leq 10^{100}$).

Saída

Imprima a mensagem "N es de Mala Suerte" se N é de má sorte, caso contrário imprima "N NO es de Mala Suerte".

Questão do URI - 3300

- Solução

```
int main() {
    string source;
    cin >> source;
    string pattern = "13";
    OM(pattern, pattern.length(),
        source, source.length());
    return 0;
}
```

SUBMISSÃO # 23684038

| | |
|------------|---|
| PROBLEMA: | 3300 - Números Má Sorte Recarregados |
| RESPOSTA: | Accepted |
| LINGUAGEM: | C++17 (g++ 7.3.0, -std=c++17 -O2 -lm) [+0s] |
| TEMPO: | 0.000s |
| TAMANHO: | 3,29 KB |
| MEMÓRIA: | - |
| CODE GOLF: | 0 caracteres (+0 que a mediana) |
| SUBMISSÃO: | 20/07/2021 22:32:36 |

```
void OM(string x, int m, string y, int n) {
    int i, j, adaptedGs[XSIZE], qsBc[ASIZE];
    pattern pat[XSIZE];

    /* Preprocessing */
    orderPattern(x, m, optimalPcmp, pat);
    preQsBc(x, m, qsBc);
    preAdaptedGs(x, m, adaptedGs, pat);

    /* Searching */
    j = 0;
    bool found = false;
    while (j ≤ n - m) {
        i = 0;
        while (i < m && pat[i].c == y[j + pat[i].loc])
            ++i;
        if (i ≥ m)
            found = true;

        j += max(adaptedGs[i], qsBc[y[j + m]]);
    }
    if (found)
        cout << y << " es de Mala Suerte" << endl;
    else
        cout << y << " NO es de Mala Suerte" << endl;
}
```

Two-Way

Definição

- Autores: Maxime Crochemore e Dominique Perrin(1991).
- Um algoritmo de correspondência de string que particiona o padrão x em dois, esquerda x_L e direita x_R para otimizar a busca.
- Em seguida, compara x_R da esquerda para a direita. Se o padrão casar, compara x_L da direita para a esquerda.

Definição

- A fase de pré-processamento do algoritmo consiste em escolher uma boa fatorização $x_L x_R$.
- Requer um alfabeto ordenado;

Conceitos

- **Fatorização:** uma string é considerada fatorada quando é dividida em duas metades. Suponha que uma string x seja dividida em duas partes (u, v) , então (u, v) é chamada de fatoraçoão de x .
- **Período:** um período p para uma string x é definido como um valor tal que para qualquer inteiro $0 < i \leq |x| - p$, $x[i] = x[i + p]$. Em outras palavras, " p é um período de x se duas letras de x na distância p sempre coincidem". O período mínimo de x é um número inteiro positivo denotado como $p(x)$.

Conceitos

- Uma **repetição** w em (u, v) é uma substring de x em que:
 - w é um sufixo de u ou u é um sufixo de w ;
 - w é um prefixo de v ou v é um prefixo de w ;
 - Em outras palavras, w ocorre em ambos os lados do corte com um possível overflow em qualquer um dos lados. Cada fatoração (u, v) de x tem pelo menos uma repetição. Pode-se ver facilmente que $1 \leq r(u, v) \leq |x|$

Conceitos

- Um **período local** is the length of a repetition in (u, v) . O menor período local em (u, v) é denotado como $r(u, v)$. Para qualquer fatoração, $0 < r(u, v) \leq |x|$
- Uma fatoração (u, v) de x tal que $r(u, v) = p(x)$ é chamada de **fatoração crítica** de x
- O algoritmo Two Way escolhe a fatoração crítica (x_L, x_R) tal que $|x_L| < p(x)$ e $|x_L|$ é mínimo.

Complexidade

- A fase de pré-processamento possui complexidade de $O(m)$ em tempo
- Complexidade linear em espaço
- A fase de busca possui complexidade de $O(n)$ em tempo.
- Realiza $2n-m$ comparações de caracteres no pior caso.

Caso de uso

- Ele é o algoritmo selecionado da glibc(biblioteca padrão do C do projeto GNU) e musl(biblioteca padrão C destinada a sistemas operacionais baseados no kernel Linux) para as famílias de funções de substring memmem e strstr.

Exemplo Gráfico

Fase de pré-processamento

| | | | | | | | | | |
|-----------------------------------|---|---|---|---|---|---|---|---|---|
| x | | G | C | A | G | A | G | A | G |
| local period | 1 | 3 | 7 | 7 | 2 | 2 | 2 | 2 | 1 |
| $x_\ell = GC, \quad x_r = AGAGAG$ | | | | | | | | | |

Exemplo Gráfico

Fase de busca

First attempt

G C A T C G C A G A G A G T A T A C A G T A C G
1 2

G C A G A G A G

Shift by: 2

Second attempt

G C A T C G C A G A G A G T A T A C A G T A C G
1

G C A G A G A G

Shift by: 1

Third attempt

G C A T C G C A G A G A G T A T A C A G T A C G
1

G C A G A G A G

Shift by: 1

Fourth attempt

G C A T C G C A G A G A G T A T A C A G T A C G
1

G C A G A G A G

Shift by: 1

Exemplo Gráfico

Fase de busca

Fifth attempt

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
| | | | | | 7 | 8 | 1 | 2 | 3 | 4 | 5 | 6 | | | | | | | | | | | |
| | | | | | G | C | A | G | A | G | A | G | | | | | | | | | | | |

Shift by: 7

Sixth attempt

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
| | | | | | | | | | | | | | | 1 | 2 | | | | | | | | |
| | | | | | | | | | | | | | | G | C | A | G | A | G | A | G | | |

Shift by: 2

Seventh attempt

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
| | | | | | | | | | | | | | | | | | 1 | 2 | | | | | |
| | | | | | | | | | | | | | | | | | G | C | A | G | A | G | |

Shift by: 2

Eighth attempt

| | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| G | C | A | T | C | G | C | A | G | A | G | A | G | T | A | T | A | C | A | G | T | A | C | G |
| | | | | | | | | | | | | | | | | | | | 1 | 2 | 3 | | |
| | | | | | | | | | | | | | | | | | | | G | C | A | G | A |

Shift by: 3

The Two Way algorithm performs 20 character comparisons on the example.

Código- fonte

```
/* Cálculo do sufixo máximo para  $\leq$  */
int maxSuf(char *x, int m, int *p) {
    int ms, j, k;
    char a, b;

    ms = -1;
    j = 0;
    k = *p = 1;
    while (j + k < m) {
        a = x[j + k];
        b = x[ms + k];
        if (a < b) {
            j += k;
            k = 1;
            *p = j - ms;
        } else if (a == b)
            if (k != *p)
                ++k;
            else {
                j += *p;
                k = 1;
            }
        else { /* a > b */
            ms = j;
            j = ms + 1;
            k = *p = 1;
        }
    }
    return (ms);
}
```

```
/* Cálculo do sufixo máximo para  $\geq$  */
int maxSufTilde(char *x, int m, int *p) {
    int ms, j, k;
    char a, b;

    ms = -1;
    j = 0;
    k = *p = 1;
    while (j + k < m) {
        a = x[j + k];
        b = x[ms + k];
        if (a > b) {
            j += k;
            k = 1;
            *p = j - ms;
        } else if (a == b)
            if (k != *p)
                ++k;
            else {
                j += *p;
                k = 1;
            }
        else { /* a < b */
            ms = j;
            j = ms + 1;
            k = *p = 1;
        }
    }
    return (ms);
}
```


Código-fonte

```
/* Algoritmo de correspondência de strings Two Way. */  
void TW(char *x, int m, char *y, int n) {  
    int i, j, ell, memory, p, per, q;  
  
    /* Pré-processamento */  
    i = maxSuf(x, m, &p);  
    j = maxSufTilde(x, m, &q);  
    if (i > j) {  
        ell = i;  
        per = p;  
    } else {  
        ell = j;  
        per = q;  
    }  
  
    /* Buscando */  
    if (memcmp(x, x + per, ell + 1) == 0) {  
        j = 0;  
        memory = -1;  
        while (j ≤ n - m) {  
            int max = ell > memory ? ell : memory;  
            i = max + 1;  
            while (i < m && x[i] == y[i + j]) ++i;  
            if (i ≥ m) {  
                i = ell;  
                while (i > memory && x[i] == y[i + j]) --i;  
                if (i ≤ memory) printf("%d\n", j);  
                j += per;  
                memory = m - per - 1;  
            } else {  
                j += (i - ell);  
                memory = -1;  
            }  
        }  
    }  
}
```

Código-fonte

```
    } else {  
        int max = ell + 1 > m - ell - 1 ? ell + 1 : m - ell - 1;  
        per = max + 1;  
        j = 0;  
        while (j ≤ n - m) {  
            i = ell + 1;  
            while (i < m && x[i] == y[i + j]) ++i;  
            if (i ≥ m) {  
                i = ell;  
                while (i ≥ 0 && x[i] == y[i + j]) --i;  
                if (i < 0) printf("%d\n", j);  
                j += per;  
            } else  
                j += (i - ell);  
        }  
    }  
}
```

Questão do URI - 2356

Bactéria I

Entrada

Cada caso teste contém duas strings, D e S, cada qual em uma linha, e representam o DNA da bactéria e a sequência de código genético que leva a resistência. $1 \leq |D|, |S| \leq 100$. As strings são compostas apenas pelos caracteres: A, C, G, T.

Saída

Imprima uma linha por cada caso teste, contendo a string "Resistente" (sem aspas) caso a bactéria possua o código genético requerido em seu DNA, ou a string "Nao resistente" (sem aspas) caso contrário.

Questão do URI - 2356

Exemplo de Entrada

ACGTC
CGT
CCCT
AG

Exemplo de Saída

Resistente
Nao resistente

Questão do URI - 2356

- Solução

```
/* Buscando */
bool found = false;
if (memcmp(x.c_str(), x.c_str() + per, ell + 1) == 0) {
    j = 0;
    memory = -1;
    while (j ≤ n - m) {
        int max = ell > memory ? ell : memory;
        i = max + 1;
        while (i < m && x[i] == y[i + j]) ++i;
        if (i ≥ m) {
            i = ell;
            while (i > memory && x[i] == y[i + j]) --i;
            if (i ≤ memory) found = true;
            j += per;
            memory = m - per - 1;
        } else {
            j += (i - ell);
            memory = -1;
        }
    }
} else {
```

```
    } else {
        int max = ell + 1 > m - ell - 1 ? ell + 1 : m - ell - 1;
        per = max + 1;
        j = 0;
        while (j ≤ n - m) {
            i = ell + 1;
            while (i < m && x[i] == y[i + j]) ++i;
            if (i ≥ m) {
                i = ell;
                while (i ≥ 0 && x[i] == y[i + j]) --i;
                if (i < 0) found = true;
                j += per;
            } else {
                j += (i - ell);
            }
        }
    }
}

if (found)
    cout << "Resistente" << endl;
else
    cout << "Nao resistente" << endl;
```

Questão do URI - 2356

- Solução

```
int main() {  
    string source;  
    cin >> source;  
    string pattern;  
    cin >> pattern;  
    while (!cin.eof()) {  
        TW(pattern, pattern.length(),  
            source, source.length());  
        cin >> source;  
        cin >> pattern;  
    }  
    return 0;  
}
```

SUBMISSÃO # 23684798

| | |
|------------|---|
| PROBLEMA: | 2356 - Bactéria I |
| RESPOSTA: | Accepted |
| LINGUAGEM: | C++17 (g++ 7.3.0, -std=c++17 -O2 -lm) [+0s] |
| TEMPO: | 0.000s |
| TAMANHO: | 2,57 KB |
| MEMÓRIA: | - |
| SUBMISSÃO: | 21/07/2021 00:32:53 |

String Matching on Ordered Alphabets



Definição

- É um algoritmo de correspondência de string com tempo linear e espaço constante que explora a ordem do alfabeto;
- A sua principal característica é que ele escaneia a string da esquerda para a direita.

Sobre o algoritmo

- O algoritmo calcula o prefixo máximo do prefixo correspondente ao padrão anexado do caractere incompatível do texto após cada tentativa;
- Ele evita computá-lo do zero após a mudança de comprimento ter sido executada.

Sobre o algoritmo

- Define-se tw^ew' a decomposição máxima do sufixo de uma palavra x se:
 - $v = w^ew'$ é o sufixo máximo de x de acordo com a ordem alfabética;
 - w é básico;
 - $e \geq 1$;
 - w' é sufixo adequado de w .
- Então temos $|t| < \text{per}/>(x)$.

Sobre o algoritmo

- Se tw^ew' é a decomposição de sufixo máxima de uma string x não-vazia, então as quatro propriedades são válidas:
 - Se t é sufixo de w então $\text{per}(x) = \text{per}(v)$;
 - $\text{per}(x) > |t|$;
 - Se $|t| \geq |w|$ então $\text{per}(x) > |v| = |x| - |t|$;
 - Se t não é sufixo de w e $|t| < |w|$ então $\text{per}(x) > \min(|v|, |tw^e|)$.

Sobre o algoritmo

- Se u for um sufixo de w então $\text{per}(x) == \text{per}(v) == |w|$;
-

Características

- Não possui uma fase de pré-processamento;
- A fase de busca pode ser realizada em tempo de execução $O(n)$ usando um espaço constante extra;
- No pior dos casos, o algoritmo faz não mais que $6n+5$ comparações de caracteres.

Código-fonte

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #define MAX(x, y) (((x) > (y)) ? (x) : (y))
6  #define MIN(x, y) (((x) < (y)) ? (x) : (y))
7
8  /* Função que calcula o próximo sufixo máximo */
9  void nextMaximalSuffix(char *x, int m,
10                        int *i, int *j, int *k, int *p) {
11      char a, b;
12
13      while (*j + *k < m) {
14          a = x[*i + *k];
15          b = x[*j + *k];
16          if (a == b)
17              if (*k == *p) {
18                  (*j) += *p;
19                  *k = 1;
20              }
21          else
22              ++(*k);
23          else if (a > b) {
24              (*j) += *k;
25              *k = 1;
26              *p = *j - *i;
27          }
28          else {
29              *i = *j;
30              ++(*j);
31              *k = *p = 1;
32          }
33      }
34 }
```

```
37  /* String matching on ordered alphabets algorithm. */
38  void SMOA(char *x, int m, char *y, int n) {
39      int i, ip, j, jp, k, p;
40
41      /* Searching */
42      ip = -1;
43      i = j = jp = 0;
44      k = p = 1;
45
46      while (j ≤ n - m) {
47          while (i + j < n && i < m && x[i] == y[i + j]) ++i;
48
49          if (i == 0) {
50              ++j;
51              ip = -1;
52              jp = 0;
53              k = p = 1;
54          }
55          else {
56              if (i ≥ m) printf("%d\n", j);
57
58              nextMaximalSuffix(y + j, i + 1, &ip, &jp, &k, &p);
59
60              if (ip < 0 || (ip < p && memcmp(y + j, y + j + p, ip + 1) == 0)) {
61                  j += p;
62                  i -= p;
63
64                  if (i < 0) i = 0;
65
66                  if (jp - ip > p) jp -= p;
67                  else {
68                      ip = -1;
69                      jp = 0;
70                      k = p = 1;
71                  }
72              }
73          }
74          j += (MAX(ip + 1, MIN(i - ip - 1, jp + 1)) + 1);
75          i = jp = 0;
76          ip = -1;
77          k = p = 1;
78      }
79  }
80
81 }
```

Questão do URI - 1241 - Encaixa ou Não II

```
57     }  
58     else {  
59         if (i ≥ m) found = 1;  
60  
61         nextMaximalSuffix(y + j, i+1, &ip, &jp, &k, &p);
```

| | |
|------------|---|
| PROBLEMA: | 1241 - Encaixa ou Não II |
| RESPOSTA: | Accepted |
| LINGUAGEM: | C++17 (g++ 7.3.0, -std=c++17 -O2 -lm) [+0s] |
| TEMPO: | 0.044s |
| TAMANHO: | 2,22 KB |
| MEMÓRIA: | - |
| SUBMISSÃO: | 21/07/2021 01:05:36 |

```
86 int main() {  
87     int n; cin >> n;  
88     string a, b;  
89  
90     while(n-- > 0) {  
91         cin >> a >> b;  
92  
93         int sizeA = a.length();  
94         int sizeB = b.length();  
95  
96         char aArray[sizeA + 1];  
97         char bArray[sizeB + 1];  
98  
99         strcpy(aArray, a.c_str());  
100        strcpy(bArray, b.c_str());  
101  
102        SMOA(bArray, sizeB + 1, aArray, sizeA + 1);  
103  
104        cout << (found ? "encaixa" : "nao encaixa") << endl;  
105  
106        found = 0;  
107    }  
108  
109    return 0;  
110 }
```

Referências

- **GALIL Z., SEIFERAS J.**, 1983, Time-space optimal string matching, *Journal of Computer and System Science* 26(3):280-294.
- <https://www-igm.univ-mlv.fr/~lecroq/string/node25.html>
- <https://www.geeksforgeeks.org/counting-sort/#:~:text=Counting%20sort%20is%20a%20sorting.object%20in%20the%20output%20sequence.>
- <https://www.programiz.com/dsa/counting-sort>
- <http://www.facom.ufu.br/~backes/gsi011/Aula06-Ordenacao.pdf>
- <http://www-igm.univ-mlv.fr/~lecroq/string/node27.html>