串的模式匹配

浙江大学 陈 越

浙江大学计算机科学与技术学院

什么是串

- 线性存储的一组数据（默认是字符）
- 特殊操作集
    - □ 求串的长度
    - □ 比较两串是否相等
    - □ 两串相接
    - □ 求子串
    - □ 插入子串
    - □ 匹配子串
    - □ 删除子串

目 标

给定一段文本，从中找出某个指定的关键字。

例如从一本 **Thomas Love Peacock** 写于十九世纪的小说《 **Headlong Hall** 》中找到那个最长的单词 ***osseocarnisanguineoviscericartilaginonervomedullary***

或者从古希腊喜剧《 **Assemblywomen** 》中找到一道菜的名字 ***Lopadotemachoselachogaleokranioleipsanodrimhypotrimmatosilphioparaomelitokatakechymenokichlepikossyphophattoperisteralektryonoptekephalliokigklopeleiolagoiosiraiobaphetraganopterygon***

目 标

给定一段文本：**string = s_0s_1 …… s_{n-1}**

给定一个模式：**pattern = p_0p_1 …… p_{m-1}**

求 **pattern** 在 **string** 中出现的位置

```
Position PatternMatch(char *string, char *pattern)
```

简单实现

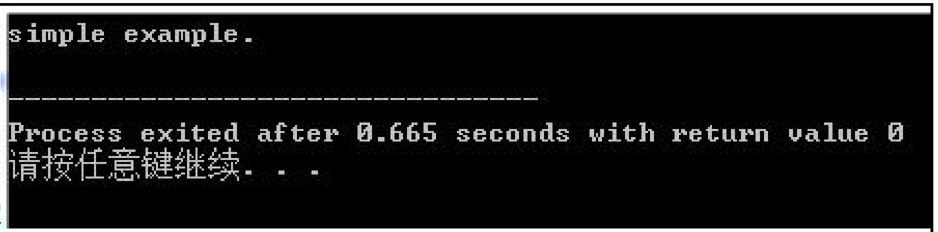
- 方法1：C的库函数 strstr

```
char *strstr(char *string, char *pattern)
```

```
#include <stdio.h>
#include <string.h>

typedef char* Position;

int main()
{   char string[] = "This is a simple example.";
    char pattern[] = "simple";
    Position p = strstr(string, pattern);
    printf("%s\n", p);
    return 0;
}
```



```
simple example.

--------------------------------
Process exited after 0.665 seconds with return value 0
请按任意键继续. . .
```

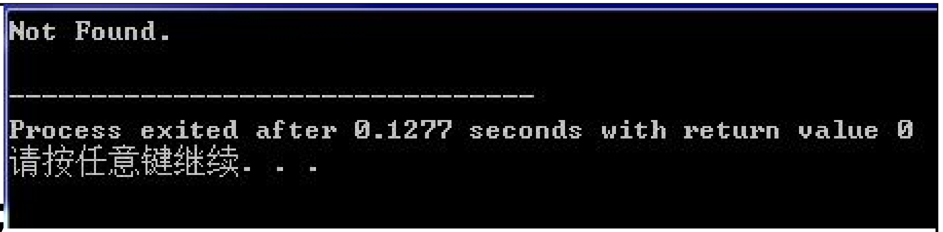
简单实现

- 方法1：C的库函数 strstr

```
char *strstr(char *string, char *pattern)
```

```
#include <stdio.h>
#include <string.h>

typedef char* Position;
#define NotFound NULL
int main()
{   char string[] = "This is a simple example.";
    char pattern[] = "sample";
    Position p = strstr(string, pattern);
    if ( p == NotFound ) printf("Not Found.\n");
    else printf("%s\n", p);
    return 0;
}
```



```
Not Found.

--------------------------------
Process exited after 0.1277 seconds with return value 0
请按任意键继续. . .
```

简单实现

- 方法1：C的库函数 strstr

```
char *strstr(char *string, char *pattern)
```

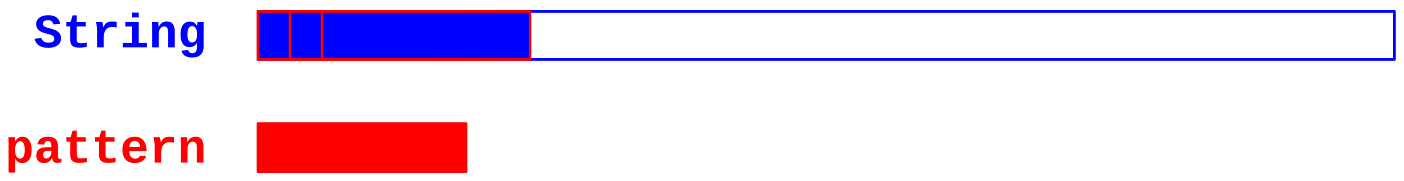
string = “$\overbrace{\texttt{aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa}}^{\mathbf{n}}$”

pattern =   “$\underbrace{\texttt{aab}}_{\mathbf{m}}$”

$$T = O(\mathrm{n}\cdot\mathrm{m})$$

简单改进

- 方法2：从末尾开始比



String

pattern

```
 string = “aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”
pattern = “aab”
```

$$T = O(n)$$

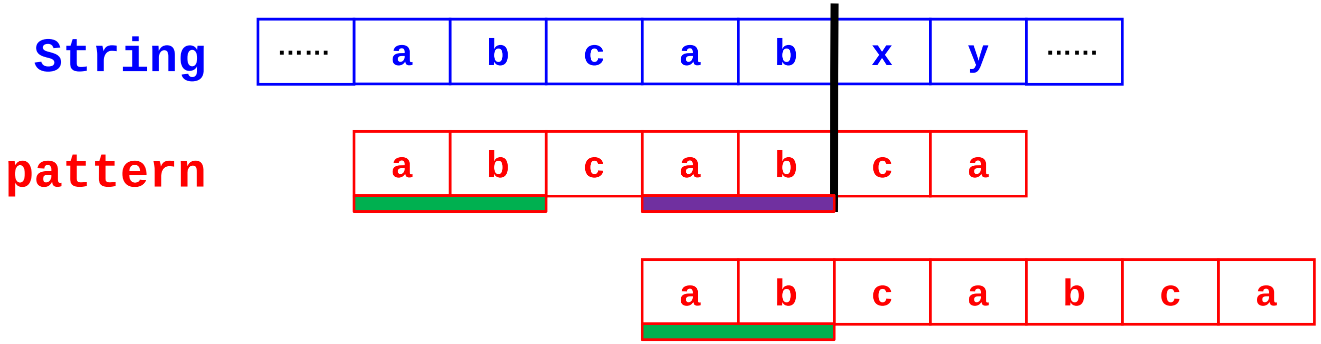
```
 string = “aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”
pattern = “baa”
```

🙄

大师改进

- 方法3：KMP（Knuth、Morris、Pratt）算法

$$T = O(\mathrm{n+m})$$



$$match(j) = \begin{cases} \text{满足 } p_0 \cdots p_i = p_{j-i} \cdots p_j \text{ 的}\underline{\text{最大}}\, i\,(< j) \\ -1 \quad \text{如果这样的 } i \text{ 不存在} \end{cases}$$

pattern	a	b	c	a	b	c	a	c	a	b
j	0	1	2	3	4	5	6	7	8	9
match	-1	-1	-1	0	1	2	3	-1	0	1

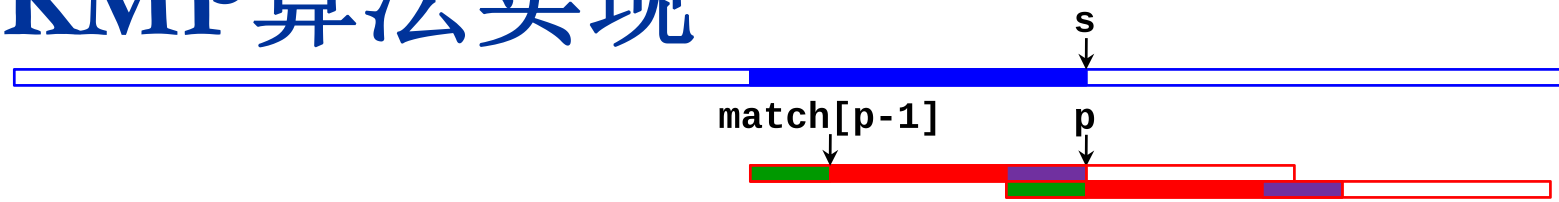
KMP算法实现

```
#include <stdio.h>
#include <string.h>

typedef int Position;
#define NotFound -1

int main()
{   char string[] = "This is a simple example.";
    char pattern[] = "simple";
    Position p = KMP(string, pattern);
    if ( p == NotFound ) printf("Not Found.\n");
    else printf("%s\n", string+p);
    return 0;
}
```

KMP算法实现



```
Position KMP( char *string, char *pattern )
{   int n = strlen(string);
    int m = strlen(pattern);
    int s, p, *match;

    match = (int *)malloc(sizeof(int) * m);
    BuildMatch(pattern, match);
    s = p = 0;
    while (s<n && p<m) {
        if (string[s]==pattern[p]) { s++; p++; }
        else if (p>0) p = match[p-1]+1;
        else s++;
    }
    return (p == m)? (s-m) : NotFound;
}
```

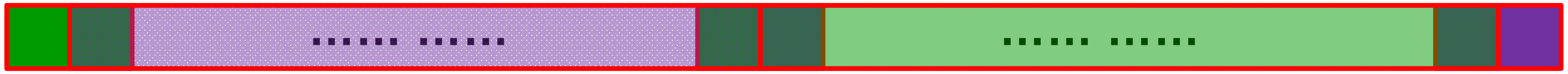
KMP算法实现

$$T = O(n+m+T_m)$$

```
Position KMP( char *string, char *pattern )
{   int n = strlen(string);     /* O(n) */
    int m = strlen(pattern);    /* O(m) */
    int s, p, *match;
    if ( n < m ) return NotFound;
    match = (int *)malloc(sizeof(int) * m);
    BuildMatch(pattern, match); /* Tm = O(?) */
    s = p = 0;
    while (s<n && p<m) { /* O(n) */
        if (string[s]==pattern[p]) { s++; p++; }
        else if (p>0) p = match[p-1]+1;
        else s++;
    }
    return (p == m)? (s-m) : NotFound;
}
```

BuildMatch 的实现

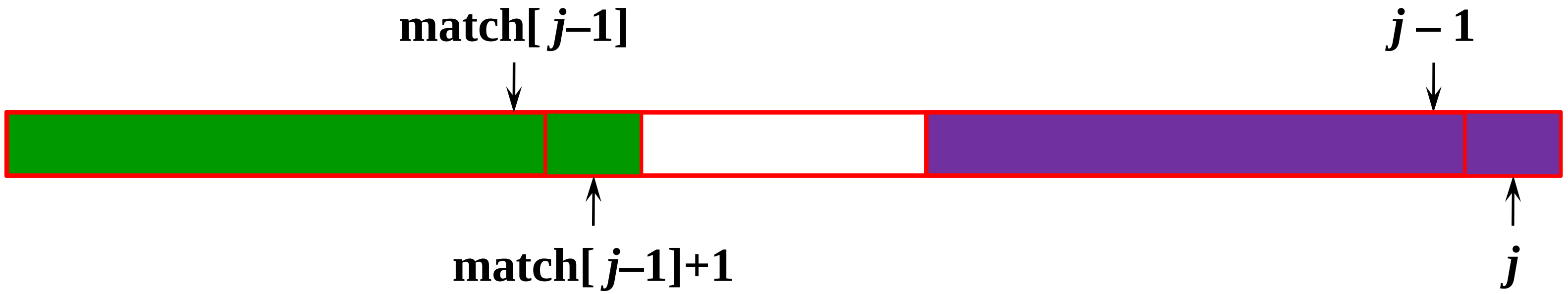
```
for ( j=0; j<m; j++ )
```



$$1 + 2 + \cdots + \frac{j+1}{2} + \cdots + j = O(j^2)$$

$$T_m = O(\mathrm{m}^3)$$ 👎

BuildMatch 的实现



match[*j*–1]

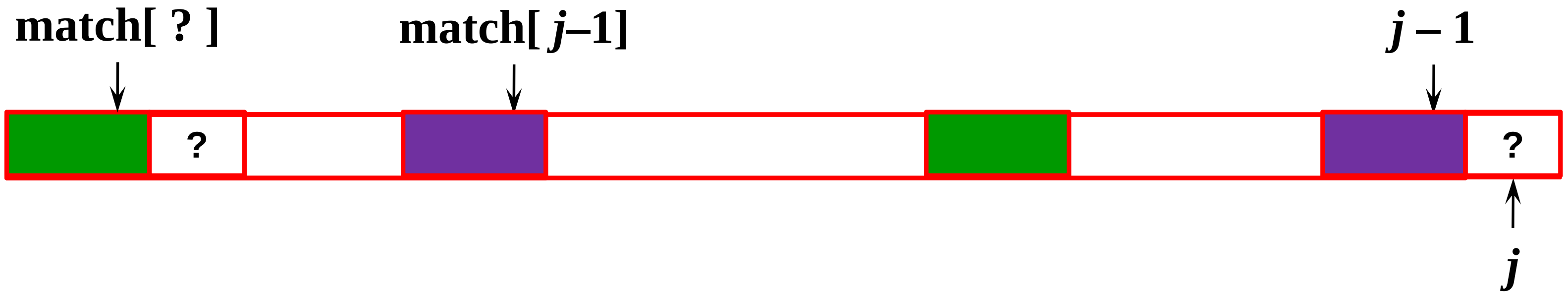
j – 1

match[*j*–1]+1

j

if (pattern[match [*j*–1]+1] == pattern[*j*])

match[*j*] = match[*j*–1] + 1



match[?]

match[*j*–1]

j – 1

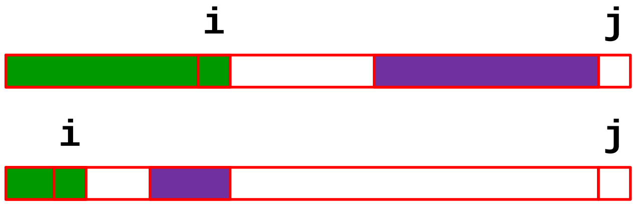
?

?

j

BuildMatch 的实现

```
void BuildMatch(char *pattern, int *match)
{   int i, j;
    int m = strlen(pattern);
    match[0] = -1;
    for (j=1; j<m; j++) {
        i = match[j-1];
        while ((i>=0) && (pattern[i+1]!=pattern[j]))
            i = match[i];
        if (pattern[i+1]==pattern[j])
            match[j] = i+1;
        else match[j] = -1;
    }
}
```



BuildMatch 的实现

```
void BuildMatch(char *pattern, int *match)
{   int i, j;
    int m = strlen(pattern);/* O(m) */
    match[0] = -1;
    for (j=1; j<m; j++) {  /* O(m) */
        i = match[j-1];
        while ((i>=0) && (pattern[i+1]!=pattern[j]))
            i = match[i];
        if (pattern[i+1]==pattern[j])
            match[j] = i+1;
        else match[j] = -1;
    }
}
```

$T_m = O(m^2)$?

i 回退的总次数不会超过 i 增加的总次数

$$T_m = O(m)$$