算法基础

主讲人： 庄连生

Email: { *lszhuang@ustc.edu.cn* }

***Spring 2015*，USTC**

University of Science and Technology of China

字符串匹配

内容提要：

- BF算法（Brute Force）
- KMP算法（Knuth-Morris-Pratt)
- Horspool算法
- Sunday算法
- BM算法（Boyer-Moore）

概念约定

字符串匹配问题：

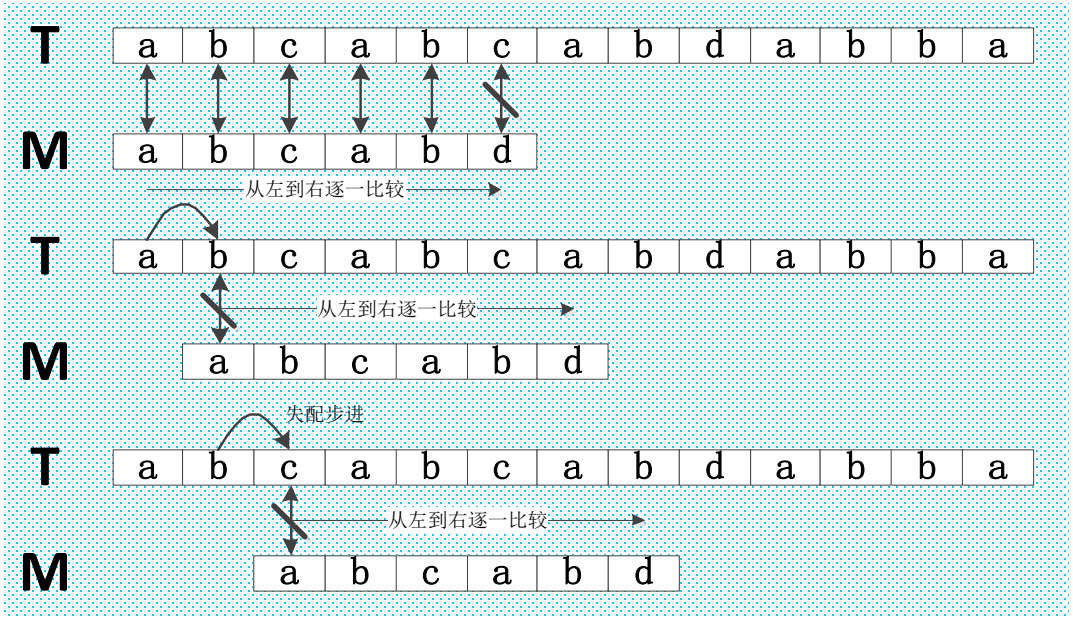
- 假设文本是一个长度为n的数组T[$1..n$]，模式是一个长度为$m <= n$的数组M[$1..m$]。进一步假设P和M都是属于有限字母表∑表中的字符。字符串匹配问题是在一段指定的文本T中，找出某指定模式M出现的所有有效位移的问题。

概念约定

- 文本串（T）：被检索的字符串，查找其中是否有某子字符串
- 模式串（M）：文本串中要检索出的子串
- i：文本串中当前参与比较的元素下标
- j：模式串中当前参与比较的元素下标
- 失配：与匹配相反，指T串与M串某位进行比较时出现不相等的情况。

暴力破解算法(BF)

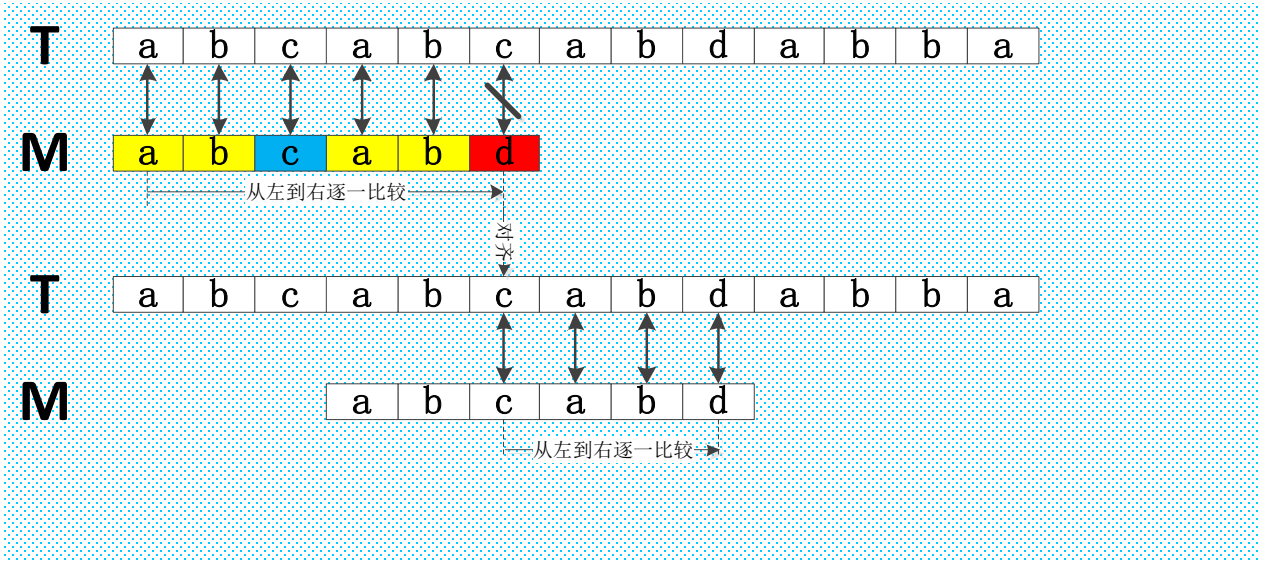
- **原理：**首先将文本串和模式串左对齐，然后从左向右一个一个进行比较，如果不成功则模式串向右移动一个单位。



- **优点：**简单易懂、易于实现
- **缺点：**时间复杂度为O(length(T) * length(M))

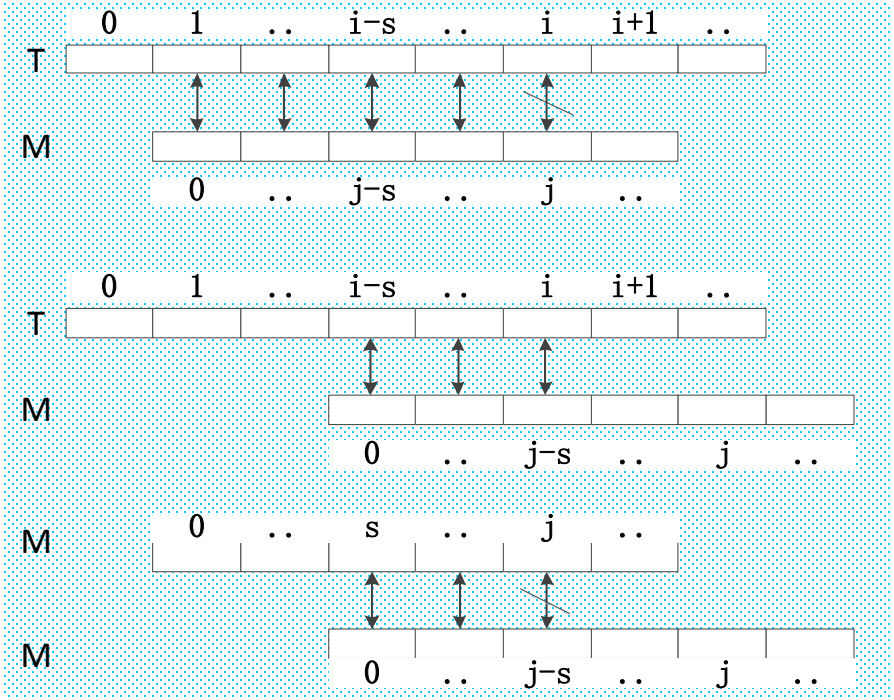
KMP算法

- KMP算法是一个经典的字符串匹配算法。
- **主要思想：**每当一趟匹配过程中出现失配时，不需回溯i，而是利用已经得到的“部分匹配”的结果将模式串向右滑动尽可能远的距离后继续进行比较。



KMP算法

算法原理：



失配后，M串要向右移动。如果移动s(j>=s>0)位后，为正确匹配结果，那么有M[j-1]=M[j-s-1],M[j-2]=M[j-s-2],…… ， M[s] = M[0],且M[j]!=M[j-s]。以上是正确匹配的必要条件。

根据此必要条件，寻找符合条件的位移大小s。而满足这个必要条件的s可能有多个，那么只需选取其中最小s（最大重复串），这样就不会滤过正确匹配位移值。

当然，也可能正确匹配结果在T[i]之后，此时的位移值s>j，只需要选取位移j+1（s>=j+1）位，此时，也不会滤过正确匹配位移值。

KMP算法

- **关键问题**：如何计算跳转长度？也就是，前缀函数的确定。
- 前缀函数计算，参见课本P569页。
- 一个例子：

利用不匹配字符的前面那一段字符的最长前后缀来尽可能地跳过最大的距离

比如

模式串ababac 这个时候我们发现在c 处不匹配，然后我们看c 前面那串字符串的最大相等前后缀，然后再来移动

下面的两个都是模式串，没有写出来匹配串

原始位置 ababa c

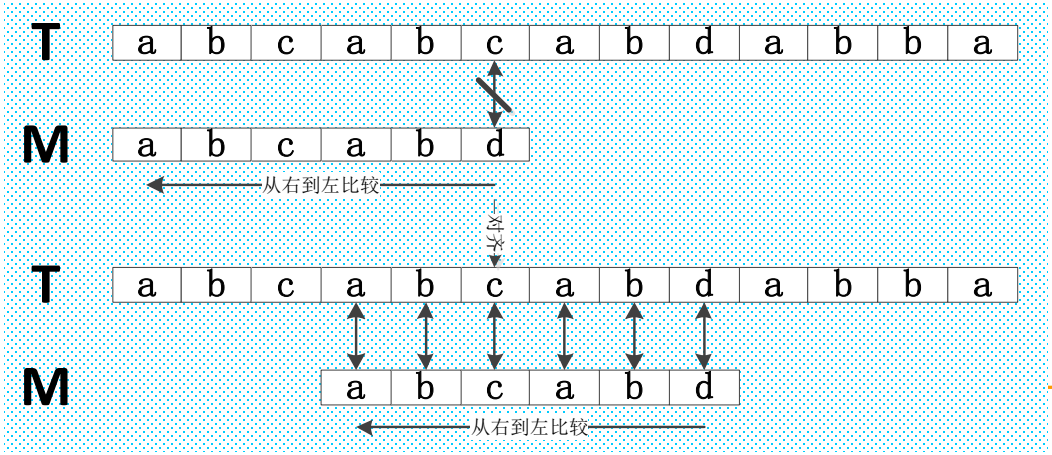
移动之后 aba bac

因为后缀是已经匹配了的，而前缀和后缀是相等的，所以直接把前缀移动到原来后缀处，再从原来的c 处，也就是现在的第二个b 处进行比较。 这就是KMP 。

Horspool算法

基本原理：

1. 首先将T串与M串左端对齐；
2. 从M串的末位从右到左比较字符；
3. 匹配就继续比较直到比较完成；
4. 失配的情况下，从M串失配的位置起，往左寻找与T串失配字符相等的字符，然后将M串向右移动相应的距离，转移到步骤2；找不到，则将M串左边与失配T[i]字符的下一位对齐，转移到步骤2。



Horspool算法

- **正确性分析：**失配发生时，M串要向右移动，T串的失配字符T[i]可能是正确结果中的一个元素也可能不是。
  - ✓ 如果不是，那么可以将M串的首位与T串中的 [i+1] 位对齐，继续进行比较；
  - ✓ 如果是，那么，自M串失配位置 [j] 开始向左的任意一位与M[i] 相等的字符c确定的比较位置都可能是完全匹配的，其产生的位移依次为*k1*，*k2*，……（*ki* 为 $j - \text{location}(c)$）。
- 综合这两种情况，选择位移最小值k1，这样就能保证不会错过正确匹配的情况。

Horspool算法

- **优点：**这个算法是字符串匹配算法中从右向左匹配的创始者，且其失配情况下不是+1位移，平均时间复杂度为O(length(T))。
- **缺点：**只是用了当前匹配位的特征，没有使用之前比较的历史数据，使得失配后M串位移没有最大化。
- 一个简单例子：

匹配串：abcbc sdxzcxx
模式串：cbcac

匹配串：abcbcsd xzcxx
模式串： cbcac

匹配串：abcbcsdxzcxx
模式串： cbcac

BM(Boyer-Moore)算法

- BM算法是在基于Horspool算法的一种改进算法，其广泛应用于实际的项目中。
- **算法原理**：匹配时，同Horspool算法；当失配时，计算M串失配字符的右边已经匹配了的字符串（称为“好后缀”）在M串中其他位置出现的位置，得到一个右移大小值（实际这个值是预先计算好了的），将这个值与Horspool失配时M串移动大小的值进行比较，并选择其中较大者。对M串移动后，继续进行比较。

一个具体例子：

http://www.ruanyifeng.com/blog/2013/05/boyer-moore_string_search_algorithm.html

BM算法

- BM算法的好后缀相关规定及计算方法：

① “好后缀”的位置以最后一个字符为准。假定"ABCDEF"的"EF"是好后缀，则它的位置以"F"为准，即5（从0开始计算）。

② 如果“好后缀”在搜索词中只出现一次，则它的上一次出现位置为-1。比如，"EF"在"ABCDEF"之中只出现一次，则它的上一次出现位置为-1（即未出现）。

③ 如果“好后缀”有多个，则除了最长的那个“好后缀”，其他“好后缀”的上一次出现位置必须在头部。比如，假定“BABCDAB”的“好后缀”是“DAB”、“AB”、“B”。这个规则也可以这样表达：如果最长的那个"好后缀"只出现一次，则可以把搜索词改写成如下形式进行位置计算"(DA)BABCDAB"，即虚拟加入最前面的"DA"。

BM算法

分为两步预处理，第一个是bad-character heuristics ， 也就是当出现错误匹配的时候，移位，基本上就是做的Horspool 那一套。

第二个就是good-suffix heuristics ， 当出现错误匹配的时候，我还要从不匹配点向左看啊，以前匹配的那段子字符串是不是在模式串本身中还有重复的啊，有重复的话，那么我就直接把重复的那段和匹配串中已经匹配的那一段对齐就是了。再比较

匹配串：abaccba bbazz

模式串：cbadcba

我们看到已经匹配好了cba ， 但是c-d 不匹配，这个时候我们发现既可以采用bad-character heuristics ， 也可以使用good-suffix heuristics(模式串： cba dcba) ， 在这种情况下，邪不压正。毅然投奔good 。移动得到

匹配串：abaccbabbaz z

模式串： cbadcba

可是，我们有时候也发现，已经匹配好的那一部分其实并没有再有重复了的啊。这个时候，我们发现已经匹配好的那串字符串有一部分在开头重新出现了，那么，赶快，对齐吧。

匹配串：abacccb bbazz

模式串：cbadccb

然后得到

匹配串：abacccbbbazz

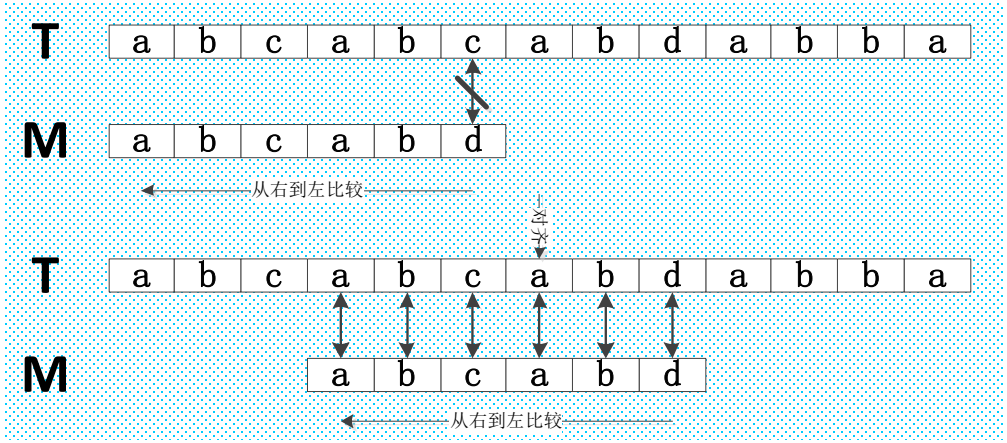
模式串： cbadccb

当两种Good-Suffix 出现的时候，取移动距离最大的那个。

Sunday算法

基本原理：

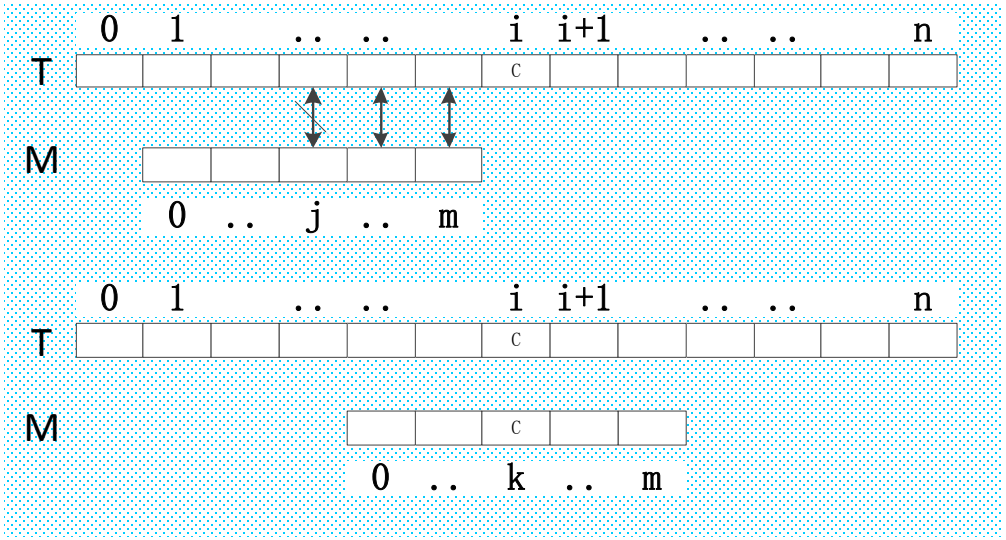
1. 首先将T串与M串左端对齐；
2. 从M串的末位从右到左比较字符；
3. 匹配就继续比较直到比较完成；
4. 失配的情况下，选中T串与M串右对齐端的下一个字符T[i]，从M串末位往左寻找与该字符相等的字符M[j]，找到后，T[i]与M[j]对齐，转移到步骤2；找不到，则M串左端与T[i]的下一个字符对齐，转移到步骤2。



Sunday算法

- **正确性**：设定T串与M串右对齐端的下一位序列为i，T[i] == C。
- T串与M串比较过程中失配时，M串至少要向右移一位继续进行比较，那么T[i]是正确匹配过程中的必经元素且T[i]可能是正确结果中的一个元素。
  - ✓ 如果不是，M串左端与T[i+1]对齐，继续进行比较；
  - ✓ 如果是，T[i]要与M串中为值为C的元素对齐进行比较。其产生的位移值有多个。
- 以上两种情况中，选取位移最小的进行对齐比较，保证一定不会滤过正确匹配位置。

Sunday算法



BF算法实现

```
int bf(const char *text,const char *mode)
{
        int text_length = strlen(text);
        int mode_length = strlen(mode);
        int i,j;
        for(i=0;i <= text_length-mode_length;i++)
        {
                for(j=0;j<mode_length;j++)
                        if (mode[j] != text[i+j])break;
                if(j==mode_length)return i;
        }
        return -1;
}
```

KMP算法实现

```
void get_next(const char *mode,int *nextval)
{
    int i = 0,j = -1,len = strlen(mode);
    nextval[i] = -1;
    while( i < len-1 )
    {
       if( j == -1 || mode[i] == mode[j] )
       {
            ++i,++j;
            if( mode[i] != mode[j] )
                nextval[i] = j;
            else
                nextval[i] = nextval[j];
        }
        else
            j = nextval[j];
     }
}
```

```
int kmp(const char *source,const char *mode)
{
    int source_length = strlen(source);
    int mode_length = strlen(mode);
    int i=0,j=0;
    int * pnext = new int[mode_length];
    get_next(mode,pnext);
    while( j <mode_length && i<source_length)
    {
       if (j==-1 || mode[j]==source[i])
          ++i ,++j;
       else  j = pnext[j];
    }
    if(j==mode_length)return i-mode_length;
    else return -1;
}
```

Horspool算法实现

```
int horspool(const char *source,const char *mode)
{
    int source_length = strlen(source);
    int mode_length = strlen(mode);
    int i=0,j=mode_length-1,k;
    while(i <= source_length-mode_length)
    {
        for(;j>=0;j--)
            if(mode[j] != source[i+j])
            {
                for(k=j-1;k>=0;k--)
                    if (mode[k]==source[i+j])break;
                i = i+j-k;
                j = mode_length - 1;
                break;
            }
        if(j<0)return i;
    }
    return -1;
}
```

Sunday算法实现

```
int sunday(const char *source,const char *mode)
{
    int source_length = strlen(source);
    int mode_length = strlen(mode);
    int i=0,j=mode_length-1,k;
    while(i <= source_length-mode_length)
    {
        for(;j>=0;j--)
            if(mode[j] != source[i+j])
            {
                for(k=mode_length-1;k>=0;k--)
                    if (mode[k]==source[i+mode_length])break;
                i = i+mode_length-k;
                j = mode_length - 1;
                break;
             }
        if(j<0)return i;
    }
    return -1;
}
```

University of Science and Technology of China

谢谢！

Q & A