



Data Structures

字符串和多维数组 String & Matrices

2022年9月28日

学而不厌 诲人不倦



- ➡ **4.1 引言**
- ➡ **4.2 字符串**
- ➡ **4.3 多维数组**
- ➡ **4.4 矩阵的压缩存储**
- ➡ 4.5 扩展与提高
- ➡ 4.6 应用举例

4.1 引言

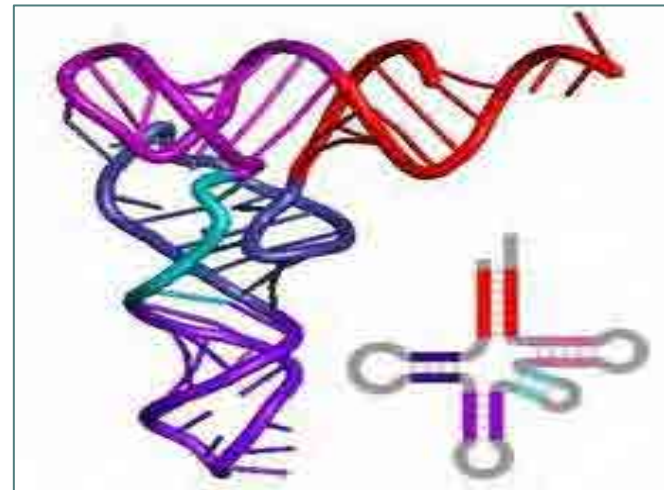
3.1 引言

字符串在逻辑上有什么特点？在操作上有什么特性？

随处可见的字符串

在非数值问题的处理过程中，数据常常以字符串的形式出现

- 📎 英文字母 → 英文单词，汉字 + 标点 → 文章
- 📎 C 语言字符集 → C 语言源程序
- 📎 DNA的碱基 = {A(腺嘌呤), G(鸟嘌呤), T(胸腺嘧啶), C(胞嘧啶)}
- 📎 RNA的碱基 = {A(腺嘌呤), G(鸟嘌呤), U尿嘧啶, C(胞嘧啶)}

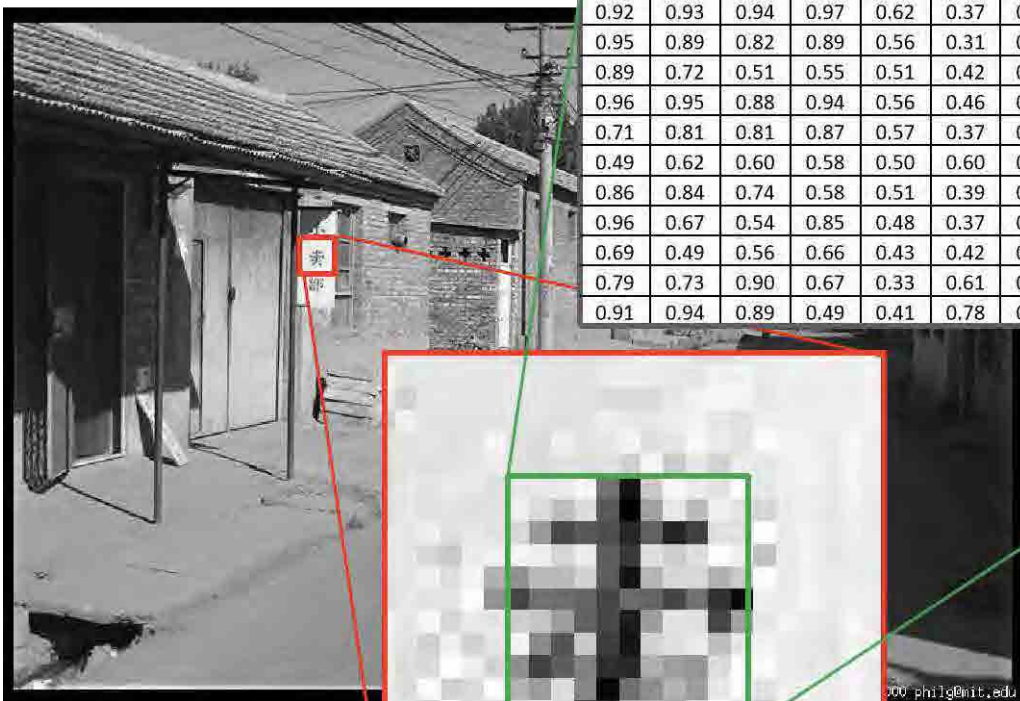


3.1 引言

数组/矩阵在逻辑上有什么特点？在操作上有什么特性？

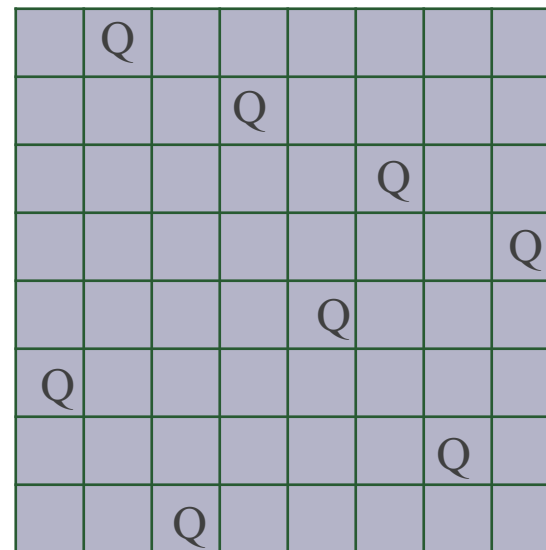
数组/矩阵

很多问题的表现形式是矩阵，很多科学问题的数据模型是矩阵



0.92	0.93	0.94	0.97	0.62	0.37	0.85	0.97	0.93	0.92	0.99
0.95	0.89	0.82	0.89	0.56	0.31	0.75	0.92	0.81	0.95	0.91
0.89	0.72	0.51	0.55	0.51	0.42	0.57	0.41	0.49	0.91	0.92
0.96	0.95	0.88	0.94	0.56	0.46	0.91	0.87	0.90	0.97	0.95
0.71	0.81	0.81	0.87	0.57	0.37	0.80	0.88	0.89	0.79	0.85
0.49	0.62	0.60	0.58	0.50	0.60	0.58	0.50	0.61	0.45	0.33
0.86	0.84	0.74	0.58	0.51	0.39	0.73	0.92	0.91	0.49	0.74
0.96	0.67	0.54	0.85	0.48	0.37	0.88	0.90	0.94	0.82	0.93
0.69	0.49	0.56	0.66	0.43	0.42	0.77	0.73	0.71	0.90	0.99
0.79	0.73	0.90	0.67	0.33	0.61	0.69	0.79	0.73	0.93	0.97
0.91	0.94	0.89	0.49	0.41	0.78	0.78	0.77	0.89	0.99	0.93

八皇后问题



- 一幅图像可以用**矩阵**表示；
- 每一个像素点对应矩阵中的一个元素。
- 一幅分辨率为1024*768的图像，对应的矩阵表示为： $I \in \mathbb{R}^{1024 \times 768}$

【问题】八皇后问题是数学家高斯于1850年提出的。问题是：在8×8的棋盘上摆放八个皇后，使其不能互相攻击，即任意两个皇后都不能处于同一行、同一列或同一斜线上。



4.2 字符串

4-2-1 字符串的逻辑结构



1. 字符串的定义

字符串常量
字符串变量 + 一系列字符串操作 = 一种新的数据类型——**串类型**

- (1) 字符串是由一个个字符组成的，需要对每个字符分别处理
- (2) 字符串需要先转化为数值后再进行处理

$$\begin{aligned}\text{ASC}(\text{'A'}) &= 65 \\ \text{ASC}(\text{'a'}) &= 97\end{aligned}$$

由于硬件结构的特点，决定了处理字符串数据时要比数值数据复杂的多。



1. 字符串的定义

📌 线性表（表）：具有相同类型的数据元素的有限序列



将元素类型限制为字符

📌 字符串（串）：零个或多个字符组成的有限序列

$(a_1, a_2, \dots, a_i, \dots, a_n)$



串名 — S = " $s_1 s_2 \dots s_n$ " — 定界符



2. 字符串的基本概念

- 串长：串中所包含的字符个数
- 空串：长度为 0 的串
- 子串：串中任意个连续的字符组成的子序列
主串：包含子串的串
子串的位置：子串的的第一个字符在主串中的序号

$S1 = "ab12cd"$

$S4 = "ab13"$

$S2 = "ab12"$

$S5 = ""$

$S3 = "ab12_"$

$S6 = "____"$



3. 字符串的比较

 字符串之间如何比较大小呢？ \Rightarrow 组成串的字符之间的比较

给定两个串： $X = "x_1x_2 \dots x_n"$ 和 $Y = "y_1y_2 \dots y_m"$ ，则：

1. 当 $n = m$ 且 $x_1 = y_1, \dots, x_n = y_m$ 时，称 $X = Y$
2. 当下列条件之一成立时，称 $X < Y$
 - (1) $n < m$ 且 $x_i = y_i (1 \leq i \leq n)$ ；
 - (2) 存在 $k \leq \min(m, n)$ ，使得 $x_i = y_i (1 \leq i \leq k-1)$ 且 $x_k < y_k$

$S1 = "ab12cd"$

$S2 = "ab12"$

$S3 = "ab12_"$

$S4 = "ab13"$

$S5 = ""$

$S6 = "____"$



4. 字符串的抽象数据类型定义

ADT String

DataModel

串中的数据元素仅由一个字符组成，相邻元素具有前驱和后继关系

Operation

StrAssign : 串赋值

StrLength : 求串S的长度

Strcat : 串连接

StrSub : 求子串

StrCmp : 串比较

StrIndex : 子串定位

StrInsert : 串插入

StrDelete : 串删除

endADT

 字符串的基本操作有什么特点？

 通常以**串整体**作为操作对象

 程序语言大都**实现**了串的基本操作



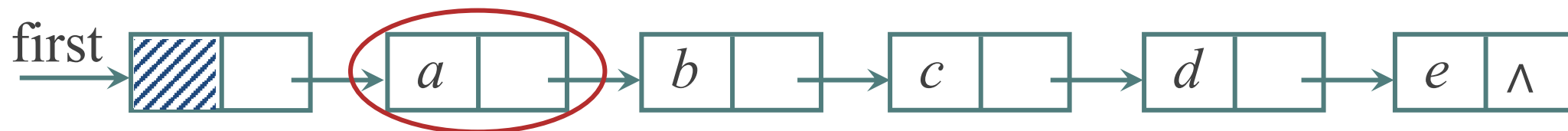
4.2 字符串

4-2-2 字符串的存储结构



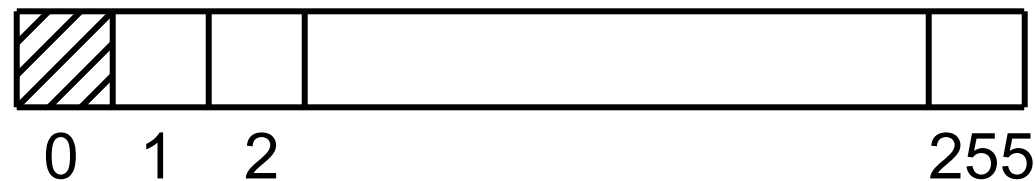
1. 串的存储

" a b c d e "



📎 指针的结构性开销降低空间性能

📌 **字符串通常采用顺序存储，即用数组存储**

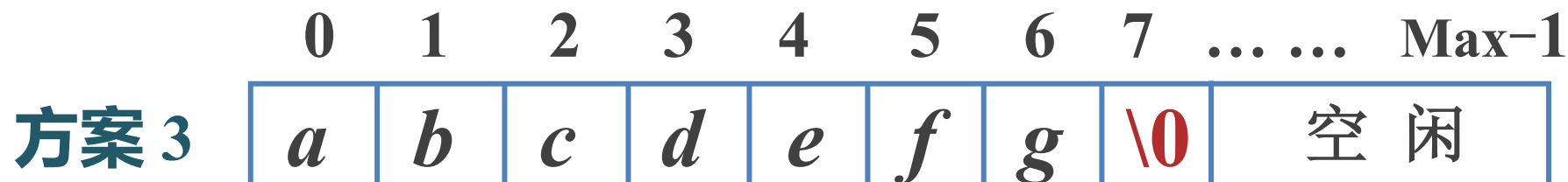
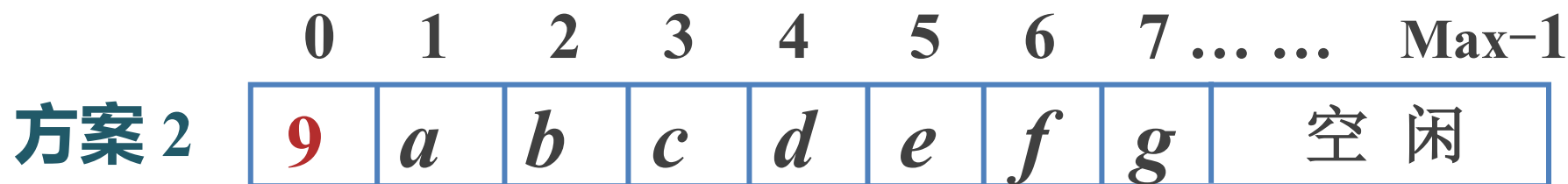
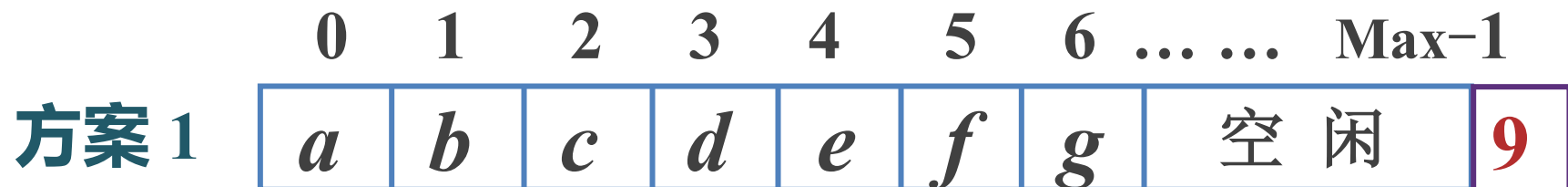


串实际长度在预定义长度内随意，超过预定义长度的串值则被舍去，称为“**截断**”。



1. 串的存储

用一组**地址连续**的存储单元存储串值的字符序列





4.2 字符串

4-2-3a 模式匹配BF算法

Brute Force 蛮力匹配/朴素匹配

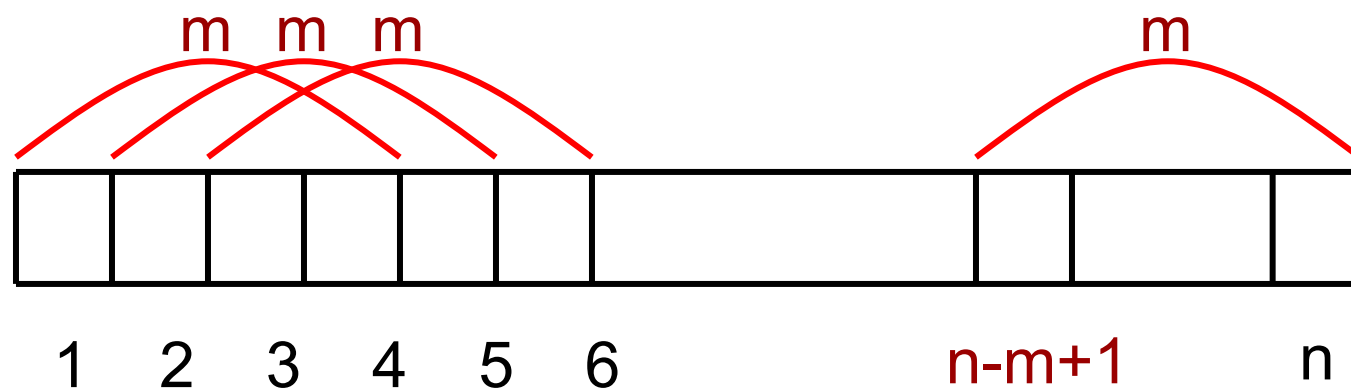


1. 模式匹配

- ✚ 模式匹配：在主串 S 中寻找子串 T 的过程， T 也称为模式
- ✚ 如果匹配成功，返回 T 在 S 中的位置；否则返回 0

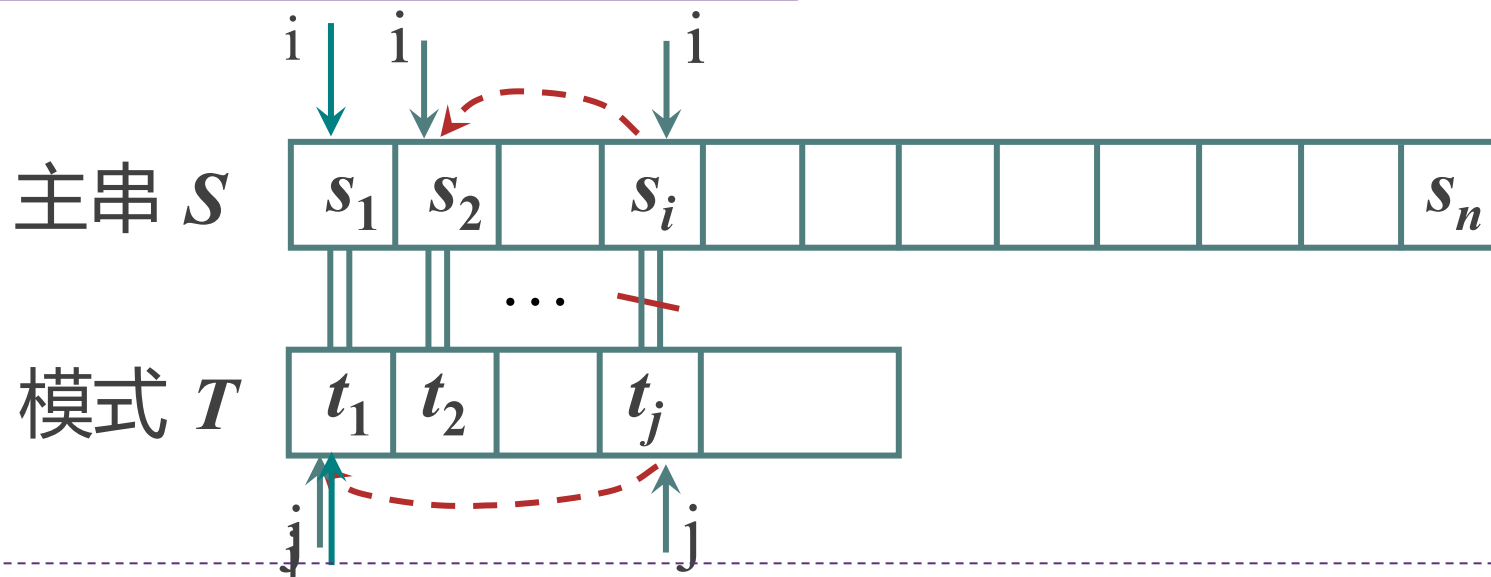
思想：

Index (String S , String T)
 n m





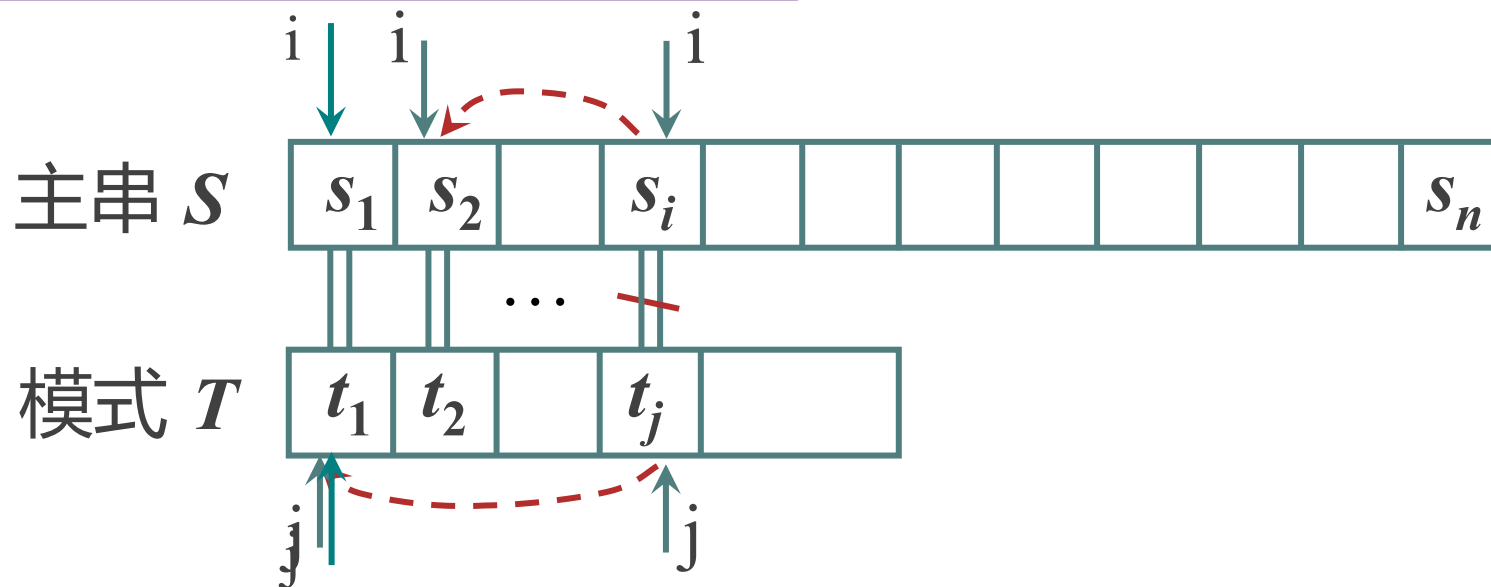
2. 模式匹配BF算法



1. 在串 S 和串 T 中设比较的起始下标 i 和 j ;
2. 循环直到 S 或 T 的所有字符均比较完
 - 2.1 如果 $S[i]$ 等于 $T[j]$, 继续比较 S 和 T 的下一个字符 ;
 - 2.2 否则 , 将 i 和 j 回溯 , 准备下一趟比较 ;
3. 如果 T 中所有字符均比较完 , 则返回匹配的起始比较下标 ; 否则返回 0 ;



2. 模式匹配BF算法

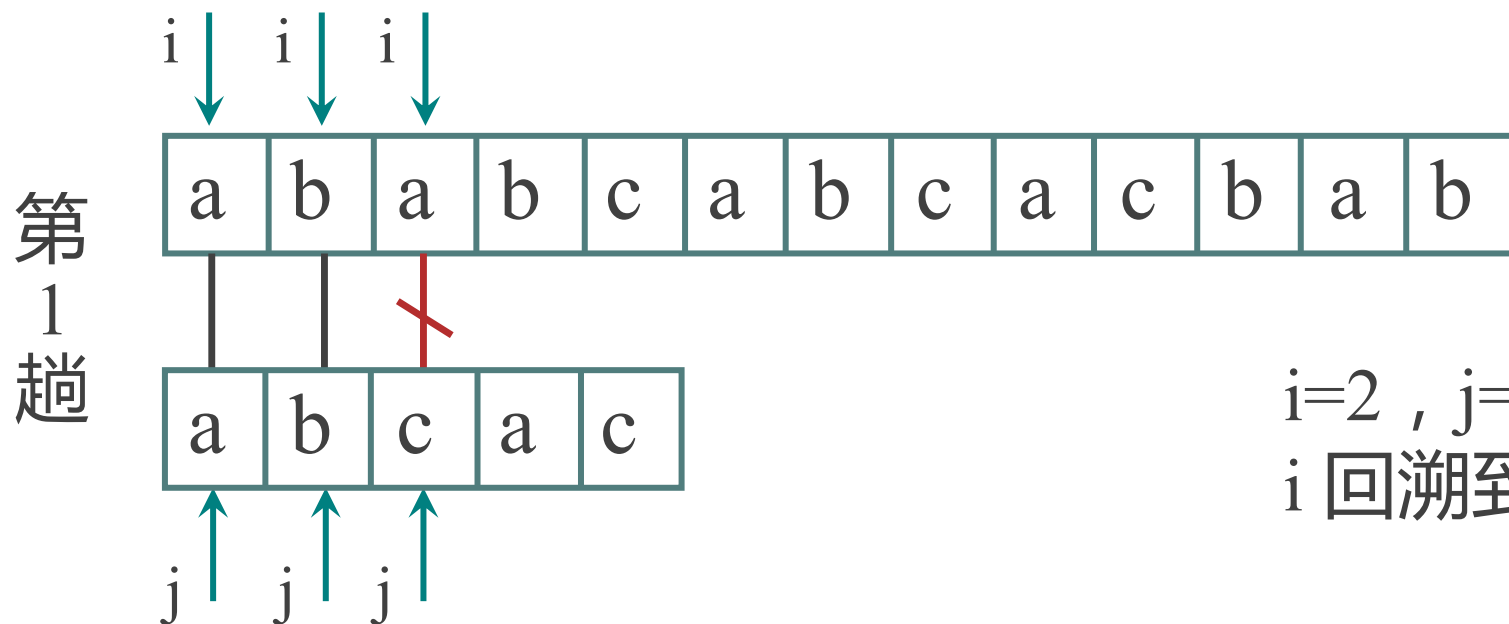


1. 在串 S 和串 T 中设比较的起始下标 i 和 j ;
2. 循环直到 S 或 T 的所有字符均比较完
 - 2.1 如果 $S[i]$ 等于 $T[j]$, 继续比较 S 和 T 的下一个字符 ;
 - 2.2 否则 , 将 i 和 j 回溯 , 准备下一趟比较 ;
3. 如果 T 中所有字符均比较完 , 则返回匹配的起始比较下标 ; 否则返回 0 ;



2. 模式匹配BF算法

例：主串 $S = \text{"ababcabcacbab"}$ ，模式 $T = \text{"abcac"}$

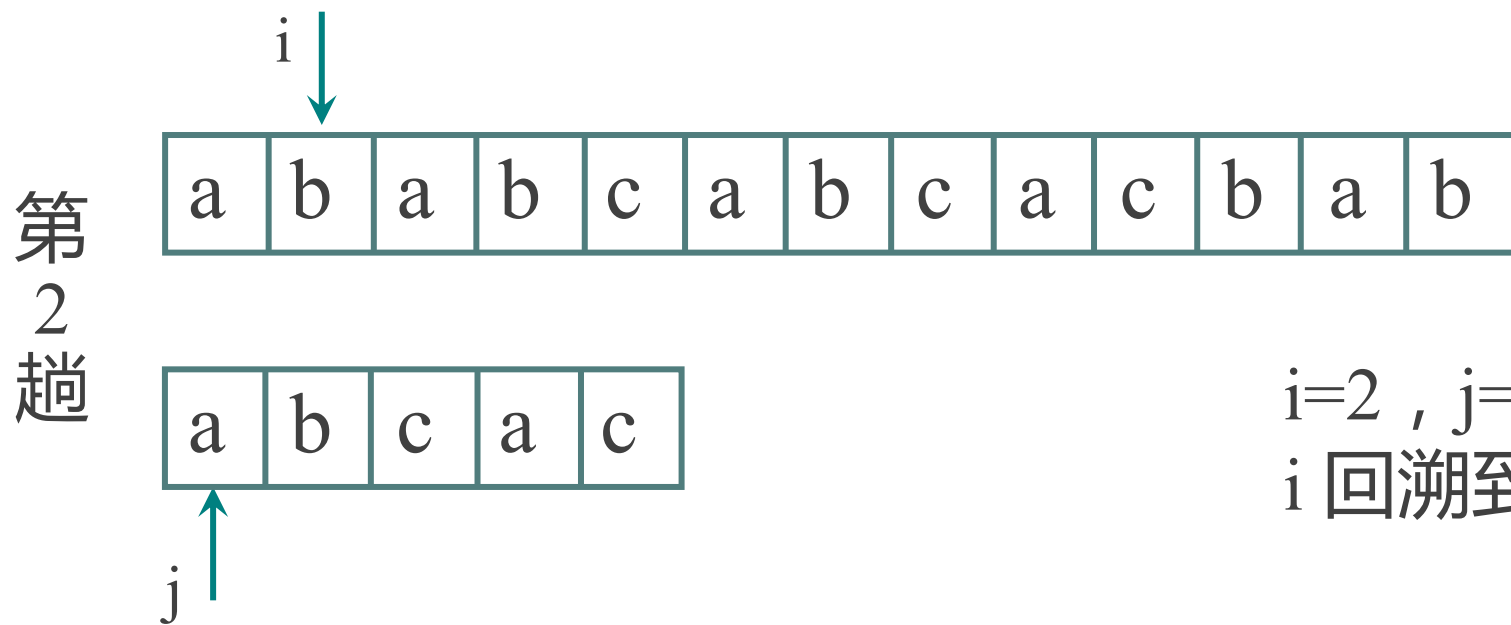


$i=2, j=2$ 失败；
 i 回溯到 1， j 回溯到 0



2. 模式匹配BF算法

例：主串 $S = \text{"ababcabcacbab"}$ ，模式 $T = \text{"abcac"}$

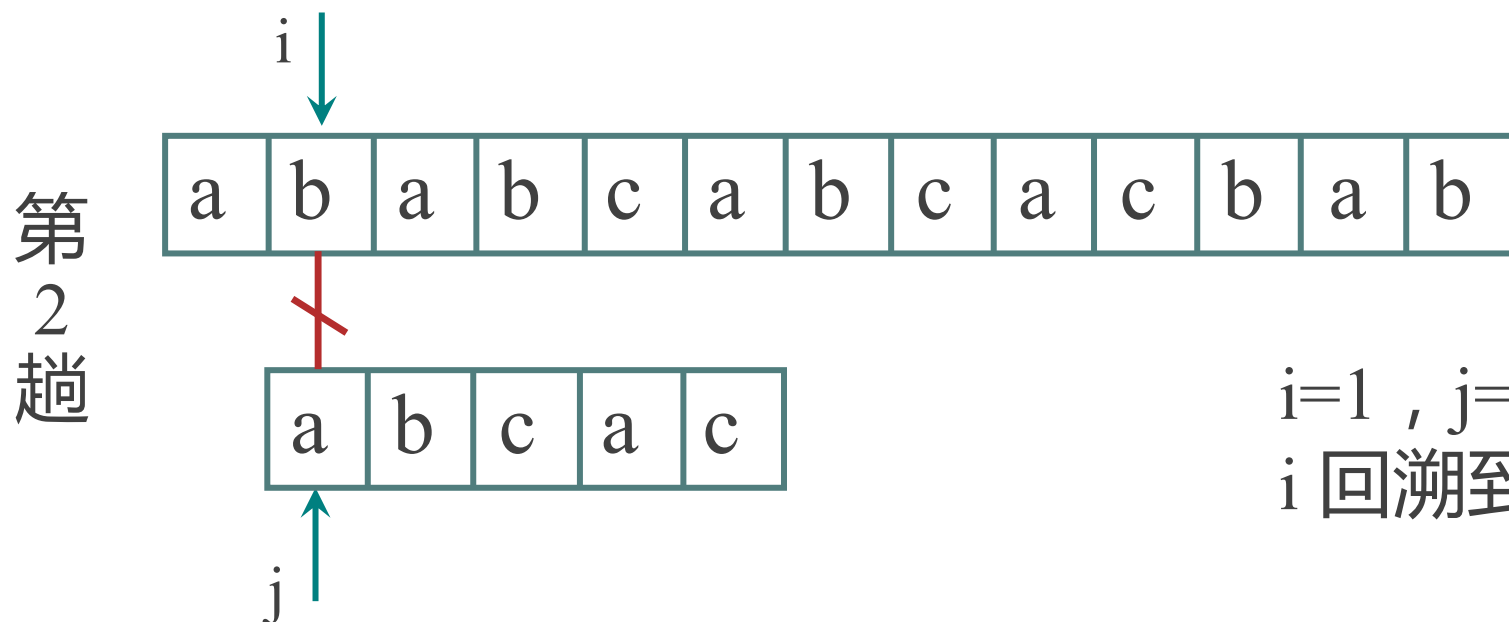


$i=2, j=2$ 失败；
 i 回溯到 1， j 回溯到 0



2. 模式匹配BF算法

例：主串 $S = \text{"ababcabcacbab"}$ ，模式 $T = \text{"abcac"}$



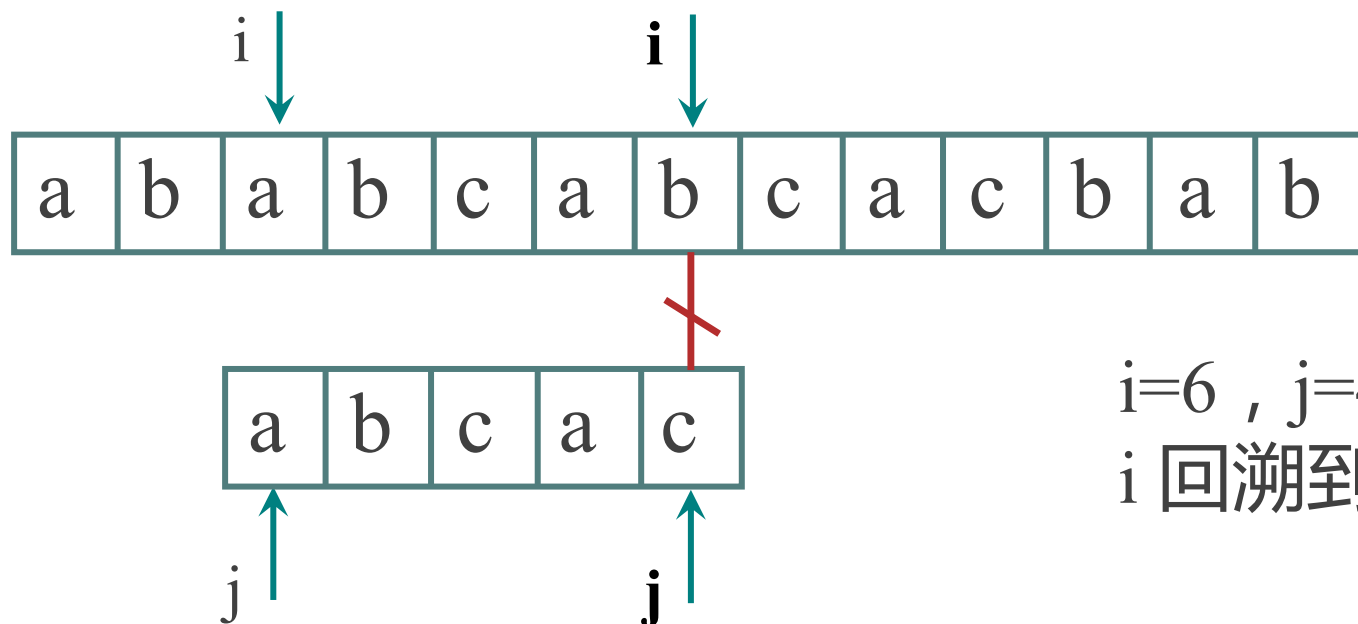
$i=1, j=0$ 失败
 i 回溯到 2, j 回溯到 0



2. 模式匹配BF算法

例：主串 $S = \text{"ababcabcacbab"}$ ，模式 $T = \text{"abcac"}$

第
3
趟



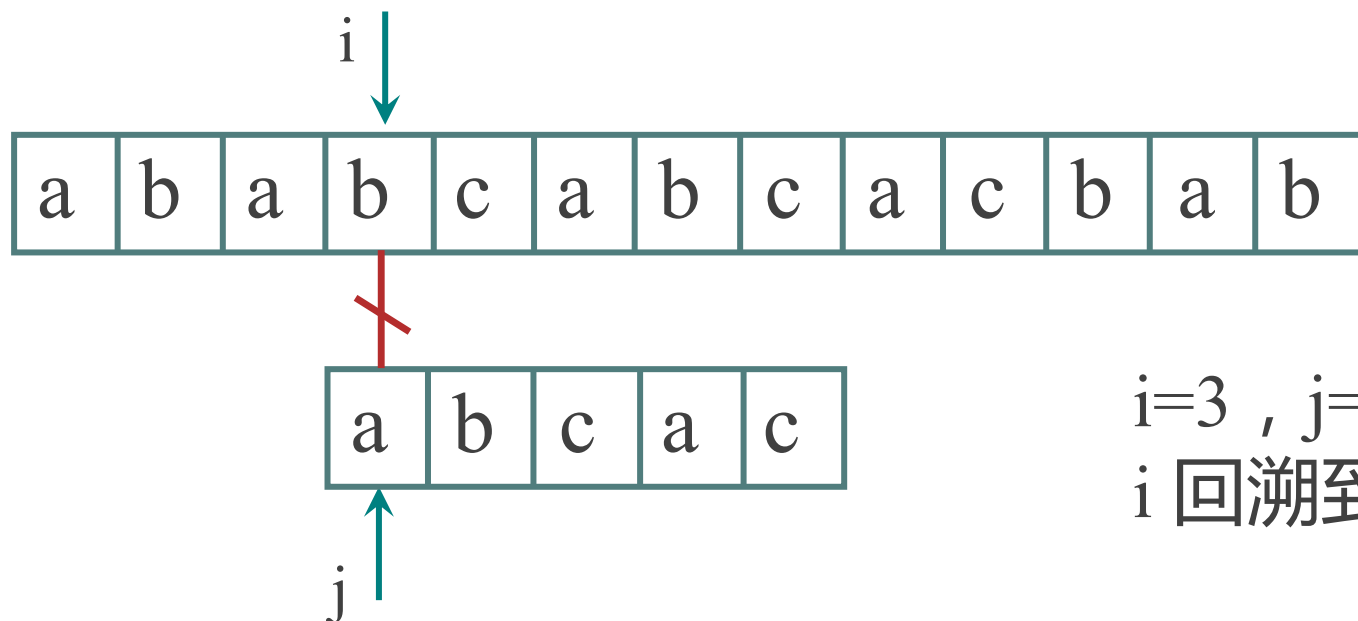
$i=6, j=4$ 失败
 i 回溯到 3, j 回溯到 0



2. 模式匹配BF算法

例：主串 $S = \text{"ababcabcacbab"}$ ，模式 $T = \text{"abcac"}$

第
4
趟

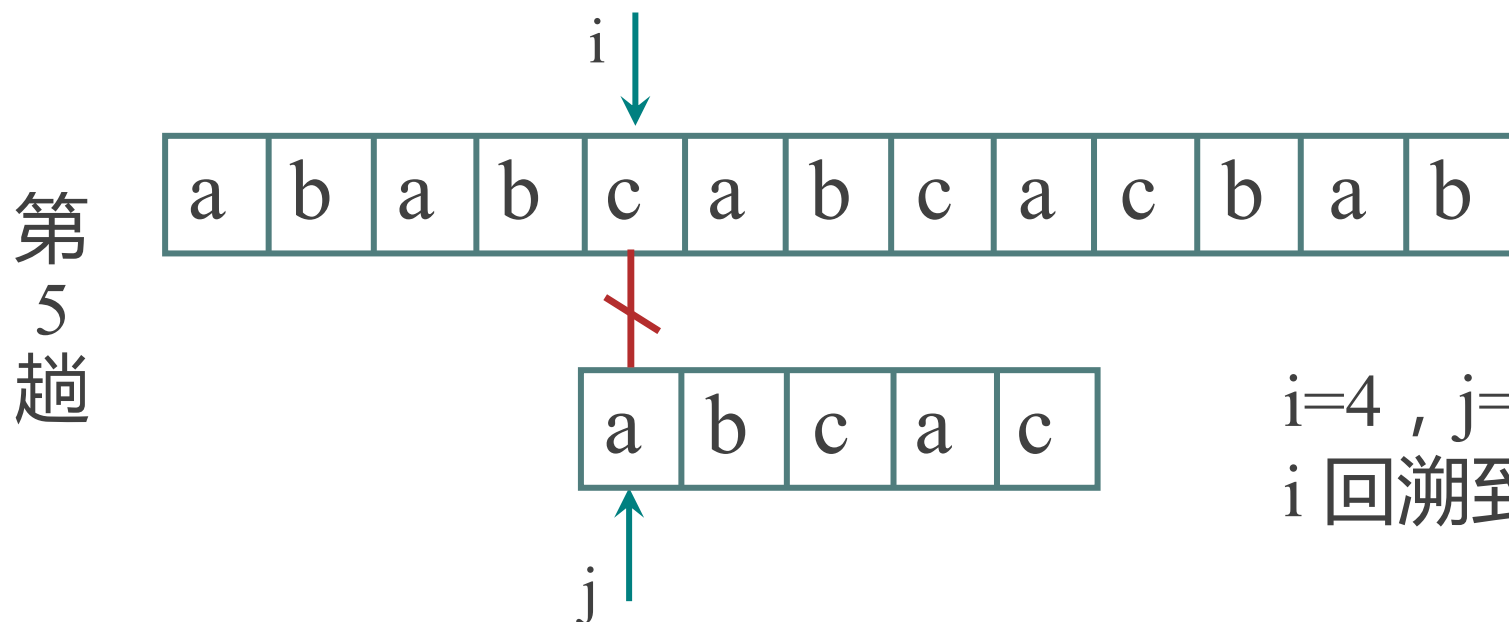


$i=3, j=0$ 失败
 i 回溯到 4, j 回溯到 0



2. 模式匹配BF算法

例：主串 $S = \text{"ababcabcacbab"}$ ，模式 $T = \text{"abcac"}$



$i=4, j=0$ 失败
 i 回溯到 5, j 回溯到 0



2. 模式匹配BF算法

```
int BF(char S[ ], char T[ ])
{
    int i = 0, j = 0;
    while (S[i] != '\0' && T[j] != '\0')
    {
        if (S[i] == T[j]) {
            i++; j++;
        }
        else {
            i = i - j + 1; j = 0;
        }
    }
    if (T[j] == '\0') return (i - j + 1);
    else return 0;
}
```

```
int BF(char S[ ], char T[ ])
{
    int i = 0, j = 0, start = 0;
    while (S[i] != '\0' && T[j] != '\0')
    {
        if (S[i] == T[j]) {
            i++; j++;
        }
        else {
            start++; i = start; j = 0;
        }
    }
    if (T[j] == '\0') return start + 1;
    else return 0;
}
```



3. 模式匹配BF算法—性能分析

$$S = "s_1 s_2 \dots s_n" \quad T = "t_1 t_2 \dots t_m"$$

 在匹配成功的情况下，考虑两种极端情况：

 **最好情况**：不成功的匹配都发生在串 T 的第 1 个字符

例如： $S = "aaaaaaaaaaa**bcd**cccc"$

$T = "bcd"$

设匹配成功发生在 s_i 处，则：

$$\sum_{i=1}^{n-m+1} p_i \times (i-1+m) = \sum_{i=1}^{n-m+1} \frac{1}{n-m+1} \times (i-1+m) = \frac{(n+m)}{2} = O(n+m)$$



3. 模式匹配BF算法—性能分析

$$S = "s_1 s_2 \dots s_n" \quad T = "t_1 t_2 \dots t_m"$$

 在匹配成功的情况下，考虑两种极端情况：

 **最坏情况**：不成功的匹配都发生在串 T 的最后一个字符

例如： $S = "aaaaaaaaaa\textcolor{red}{aaab}cccc"$

$T = "aaab"$

设匹配成功发生在 s_i 处，则：

$$\sum_{i=1}^{n-m+1} p_i \times (i \times m) = \sum_{i=1}^{n-m+1} \frac{1}{n-m+1} \times (i \times m) = \frac{m(n-m+2)}{2} = O(m \times n)$$



4.2 字符串

4-2-3b 模式匹配KMP算法

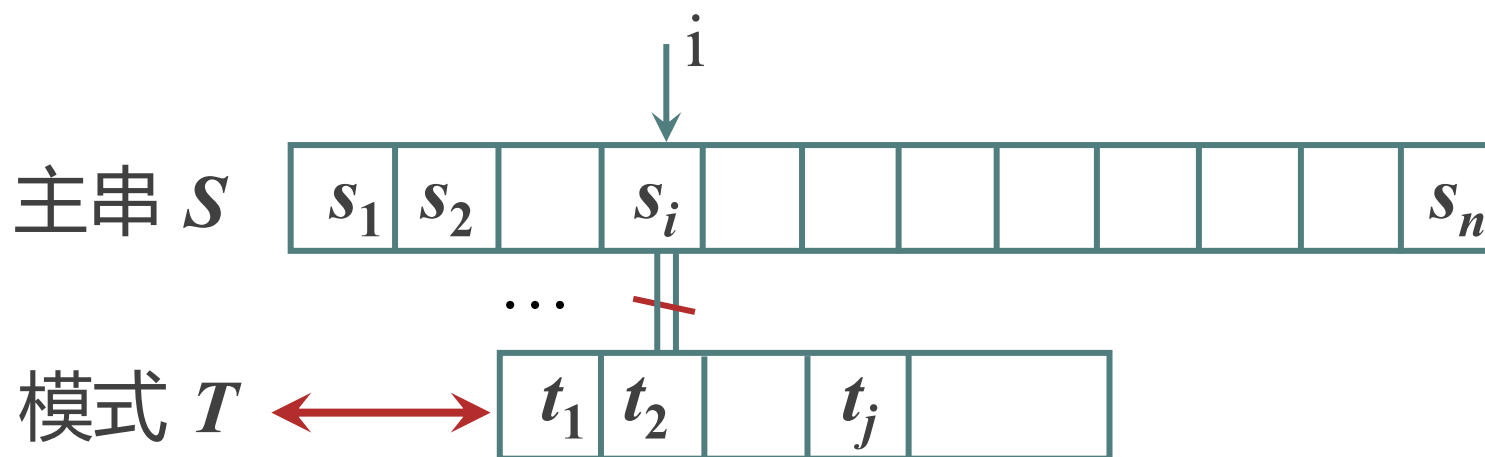
Knuth Morris Pratt



1. KMP算法

思想：每趟匹配过程中出现字符比较不等时，不回溯主指针 i ，利用已得到的“部分匹配”结果将模式向右滑动尽可能远的一段距离，继续进行比较。

 主串不回溯，模式就需要向右滑动一段距离





如何确定模式的滑动距离？



2. KMP算法—前缀/后缀集合

前缀集合

后缀集合

A		
AB	A	B
ABC	A, AB	C, BC
ABCA	A, AB, ABC	A, CA, BCA
ABCAB	A, AB, ABC, ABCA	B, AB, CAB, BCAB
ABCABF	A, AB, ABC, ABCA, ABCAB	F, BF, ABF, CABF, BCABF



3. KMP算法—next数组

✦ 设 $\text{next}[j]$ 表示在匹配过程中与 $T[j]$ 比较不相等时，下标 j 的回溯位置

前缀：开头的 k 个字符 后缀：后面的 k 个字符

$$\text{next}[j] = \begin{cases} -1 & j = 0 \\ \max\{k \mid 1 \leq k < j \text{ 且 } \overbrace{T[0] \dots T[k-1]}^{\text{前缀}} = \overbrace{T[j-k] \dots T[j-1]}^{\text{后缀}}\} & \text{集合非空} \\ 0 & \text{其它情况} \end{cases}$$

j	0	1	2	3	4
T[j]	a	b	a	b	c
next[j]	-1	0	0	1	2



KMP算法总结

1. 模式串的Next数组求解； 2. KMP匹配过程

✦ 设next[j]表示在匹配过程中与 T[j] 比较不相等时，下标 j 的回溯位置

前缀：开头的k个字符 后缀：后面的k个字符

$$\text{next}[j] = \begin{cases} -1 & j = 0 \\ \max\{k \mid 1 \leq k < j \text{ 且 } T[0] \dots T[k-1] = T[j-k] \dots T[j-1]\} & \text{集合非空} \\ 0 & \text{其它情况} \end{cases}$$

主串 $S = \text{"ababaababcb"}$ ，模式 $T = \text{"ababc"}$

j	0	1	2	3	4
T[j]	a	b	a	b	c
next[j]	-1	0	0	1	2

a b a b a a b a b c b

a b a b c



在KMP算法中求next数组的时间复杂度为 $O(m)$ ，在后面的匹配中因主串 S 的下标不减即不回溯，比较次数可记为 n ，所以KMP算法的平均时间复杂度为 $O(n+m)$ 。
最坏的时间复杂度为 $O(n \times m)$ 。



3. KMP算法—next数组

j	0	1	2	3	4	5	6	7
T[j]	a	b	a	a	b	c	a	c
next[j]	-1	0	0	1	1	2	0	1

```
void GetNext(SeqString t, int next[])
{
    int j,k;
    j=0;
    k=-1;
    next[0]=-1;
    while(j<t.len-1)
    {
        if (k==-1 || t.ch[j]==t.ch[k])
        {
            j++;
            k++;
            next[j]=k;
        }
        else k = next[k];
    }
}
```



4. KMP算法匹配过程

例题4-5 主串 $S = \text{"ababaababcb"}$, 模式 $T = \text{"ababc"}$

a	b	a	b	a	a	b	a	b	c	b
---	---	---	---	---	---	---	---	---	---	---

a	b	a	b	c
---	---	---	---	---



4. KMP算法匹配过程

注：模式串t并未向右移动，仅是为了效果，动图中进行了右移！

j	0	1	2	3	4	5
t[j]	A	A	A	A	B	A
next[j]	-1	0	1	2	3	0

从*i=0, j=0*开始匹配

0 1 2 3 4 5 6 7 8 9 ...
i
↓
s: A A A B A A A A A B A
t: A A A A B A
↑
j
0 1 2 3 4 5

算法 4.2.3

```
int KMPIndex(SeqString s, SeqString t)
{
    int next[MaxLen], i = 0, j = 0;
    GetNext(t, next);
    while(i < s.len && j < t.len)
    {
        if (j == -1 || s.ch[i] == t.ch[j])
        {
            i++;
            j++;
        }
        else j = next[j]; // i 不变, j 后退
    }
    if(j >= t.len)
        return (i - t.len); // 返回匹配模式串的首字符下标
    else
        return -1; // 返回不匹配标志
}
```



4. KMP算法匹配过程

例题 已知主串 $S = \text{“abaabaabacacaabaabcc”}$ ，模式 $T = \text{“abaabc”}$ ，采用KMP算法进行匹配，第一次出现失配 $s[i] \neq t[j]$ 时， $i=j=5$ ，则下次开始匹配时， i 和 j 的值分别是

- A. $i=1, j=0$ B. $i=5, j=0$ C. $i=5, j=2$. D. $i=6, j=2$

j	0	1	2	3	4	5
t[j]	a	b	a	a	b	c
next[j]	-1	0	0	1	1	2



5. KMP算法的时间复杂度

设主串 S 的长度为 n ，模式串 T 长度为 m ，在KMP算法中求 **next数组**的时间复杂度为 $O(m)$ ，在后面的匹配中因主串 S 的下标不减即不回溯，比较次数可记为 n ，所以**KMP算法的平均时间复杂度**为 $O(n+m)$ 。

最坏的时间复杂度为 $O(n \times m)$ 。



6. KMP算法改进

例题4-5 主串 $S = \text{"aaabaaaab"}$, 模式 $T = \text{"aaaab"}$

a	a	a	b	a	a	a	a	b
---	---	---	---	---	---	---	---	---

a	a	a	a	b
---	---	---	---	---

j	0	1	2	3	4
t[j]	a	a	a	a	b
next[j]	-1	0	1	2	3



6. KMP算法改进—nextval数组

例题

主串 $S = \text{"aaabaaaab"}$, 模式 $T = \text{"aaaab"}$

j	0	1	2	3	4
t[j]	a	a	a	a	b
next[j]	-1	0	1	2	3
nextval[j]	-1	-1	-1	-1	3

```
void GetNextval(SeqString t, int nextval[])
{
    int j, k;
    j = 0;
    k = -1;
    nextval[0] = -1;
    while (j < t.len)
    {
        if (k == -1 || t.ch[j] == t.ch[k])
        {
            j++;
            k++;
            if (t.ch[j] != t.ch[k]) nextval[j] = k;
            else nextval[j] = nextval[k];
        }
        else k = nextval[k];
    }
}
```



6. KMP算法改进

例题

主串 $S = \text{"aaabaaaab"}$, 模式 $T = \text{"aaaab"}$

j	0	1	2	3	4
t[j]	a	a	a	a	b
next[j]	-1	0	1	2	3
nextval[j]	-1	-1	-1	-1	3

```
int Improved_KMPIndex(SeqString s, SeqString t)
{
    int nextval[MaxLen], i = 0, j = 0;
    GetNextval(t, nextval);
    while(i < s.len && j < t.len)
    {
        if (j == -1 || s.ch[i] == t.ch[j])
        {
            i++;
            j++;
        }
        else j = nextval[j]; //i 不变, j 后退
    }
    if(j >= t.len)
        return (i - t.len); //返回匹配模式串的首字符下标
    else
        return -1; //返回不匹配标志
}
```




7. KMP算法使用

```
struct SeqString
{
    char ch[MaxSize];
    int len;
};
```

```
int main()
{
    SeqString s, t;
    cout<<"input S:";
    cin>>s.ch;
    cout<<"S = "<<s.ch<<endl;
    string str = s.ch;
    s.len = str.length();
    cout<<s.len<<endl;

    cout<<"input T:";
    cin>>t.ch;
    cout<<"T = "<<t.ch<<endl;
    str = t.ch;
    t.len = str.length();
    cout<<t.len<<endl;

    int index = KMPIndex(s, t);
    cout<<"子串位置为: "<<index<<endl;
}
```



小结

1. 理解字符串的定义和存储结构
2. 掌握字符串的模式匹配的BF算法和KMP算法
3. 掌握KMP算法中next、nextval数组的计算方法及匹配过程

作业：设目标主串为S= “abcaabbcaaabababaabca” ，模式串为T= “babab”

- (1) 写出按BF算法对主串S进行模式匹配的过程;
- (2) 手工计算模式串T的next值和nextval值;
- (3) 写出利用求得的nextval数组，按KMP算法对主串S进行模式匹配的过程。



Thank You !

Q & A

4.2 字符串

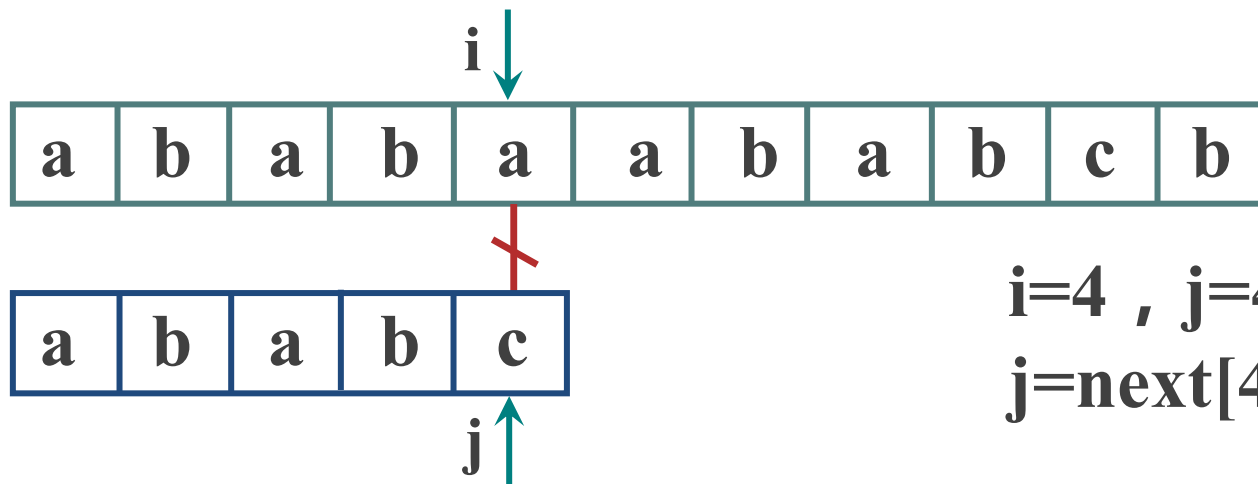
4-2-3b 模式匹配KMP算法



4. KMP算法匹配过程

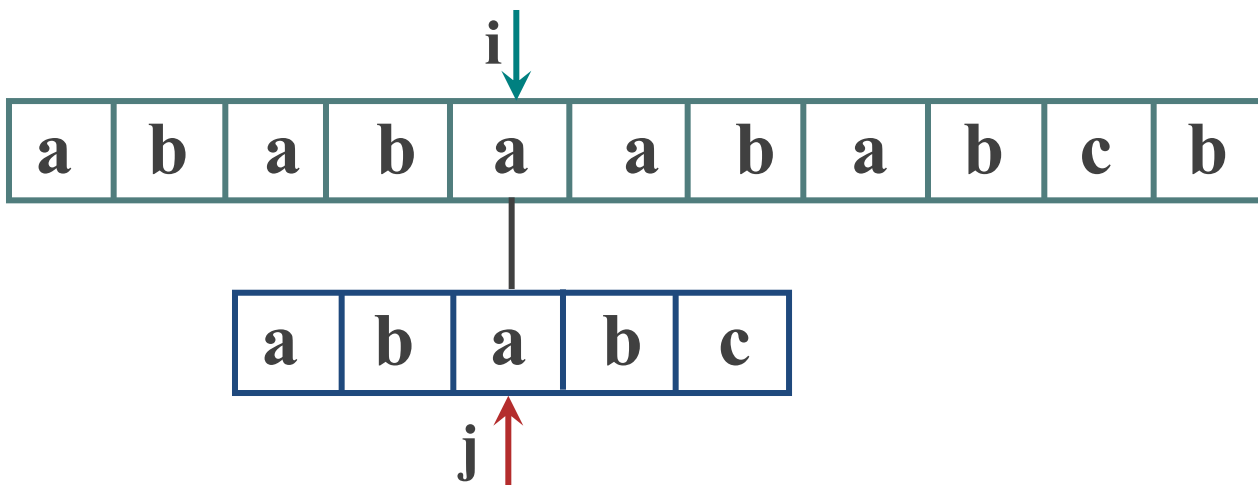
$\text{next}[j] = \{-1, 0, 0, 1, 2\}$

第1趟



$i=4$, $j=4$ 失败 ;
 $j=\text{next}[4]=2$

第2趟



4.2 字符串

4-2-3b 模式匹配KMP算法



4. KMP算法匹配过程

$\text{next}[j] = \{-1, 0, 0, 1, 2\}$

第2趟

a b a b a a b a b c b

i ↓



a b a b c

j ↑

$i=5, j=3$ 失败 ;
 $j=\text{next}[3]=1$

第3趟

a b a b a a b a b c b

i ↓

a b a b c

j ↑