本节内容

串

朴素模式匹配算法

1

什么是模式匹配

主串：S='wangdao'

子串：'wang'、'ang'、'ao' ……

一定是主串中存在的才叫“子串”

模式串：'gda'、'bao'

想尝试在主串中找到的串，未必存在

串的模式匹配：在主串中找到与模式串相同的子串，并返回其所在位置。

Index(S,T)：定位操作。若主串S中存在与串T值相同的子串，则返回它在主串S中第一次出现的位置；否则函数值为0。

2

使用基本操作实现模式匹配

Index(S,T)：定位操作。若主串S中存在与串T值相同的子串，则返回它在主串S中第一次出现的位置；否则函数值为0。

S.ch="wangdao"
S.length=7

S

	w	a	n	g	d	a	o			7
0	1	2	3	4	5	6	7	8	9	

Length

```
int Index(SString S, SString T){
    int i=1, n=StrLength(S), m=StrLength(T);
    SString sub;     //用于暂存子串
    while(i<=n-m+1){
        SubString(sub,S,i,m);
        if(StrCompare(sub, T)!=0)  ++i;
        else return i;  //返回子串在主串中的位置
    }
    return 0;    //S中不存在与T相等的子串
}
```

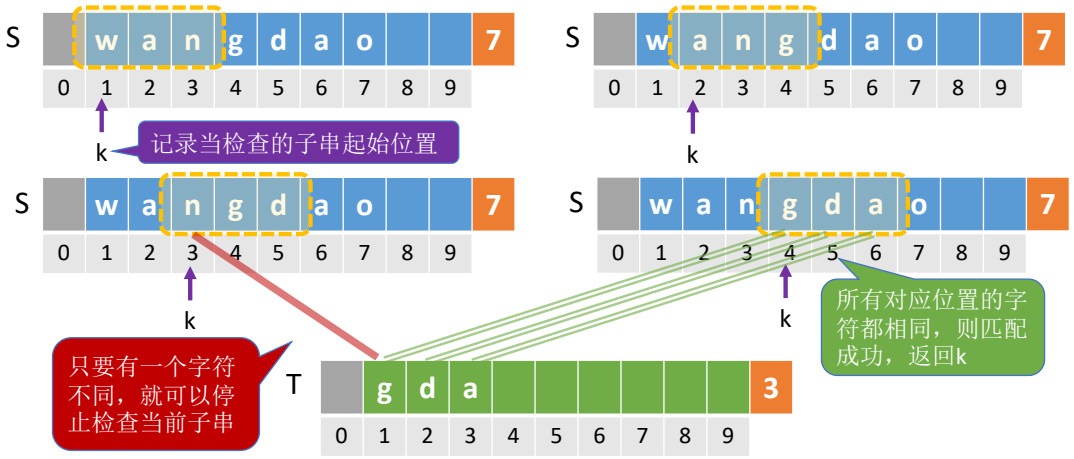
T

	g	d	a							3
0	1	2	3	4	5	6	7	8	9	

3

朴素模式匹配算法

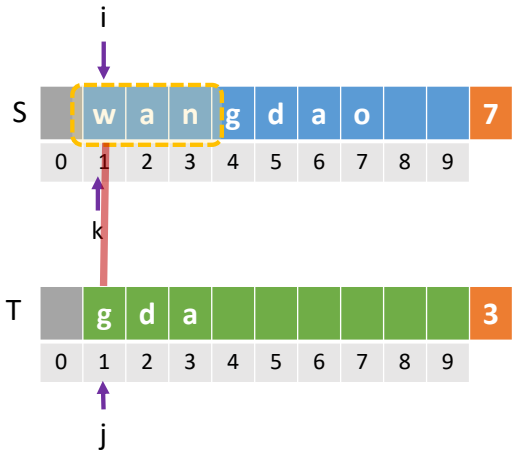
Index(S,T)：定位操作（模式匹配）。若主串S中存在与串T值相同的子串，则返回它在主串S中第一次出现的位置；否则函数值为0。



4

朴素模式匹配算法

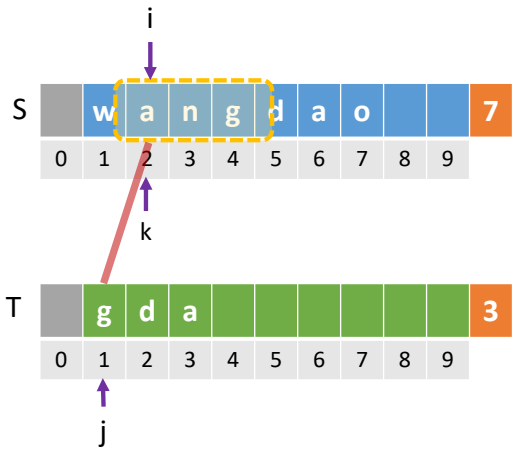
```
int Index(SString S,SString T){
    int k=1;
    int i=k, j=1;
    while(i<=S.length && j<=T.length){
        if(S.ch[i]==T.ch[j]){
            ++i;
            ++j;     //继续比较后继字符
        } else{
            k++;     //检查下一个子串
            i=k;
            j=1;
        }
    }
    if(j>T.length)
        return k;
    else
        return 0;
}
```



5

朴素模式匹配算法

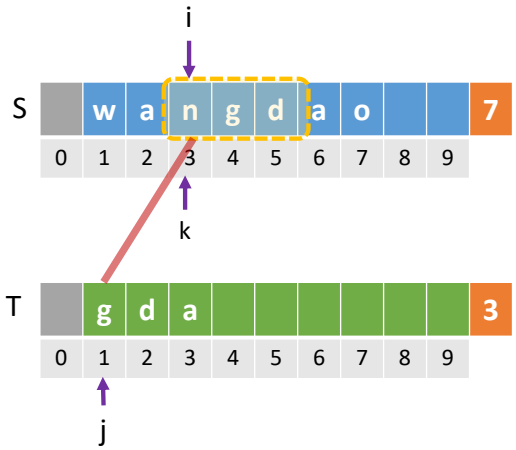
```
int Index(SString S,SString T){
    int k=1;
    int i=k, j=1;
    while(i<=S.length && j<=T.length){
        if(S.ch[i]==T.ch[j]){
            ++i;
            ++j;     //继续比较后继字符
        } else{
            k++;     //检查下一个子串
            i=k;
            j=1;
        }
    }
    if(j>T.length)
        return k;
    else
        return 0;
}
```



6

朴素模式匹配算法

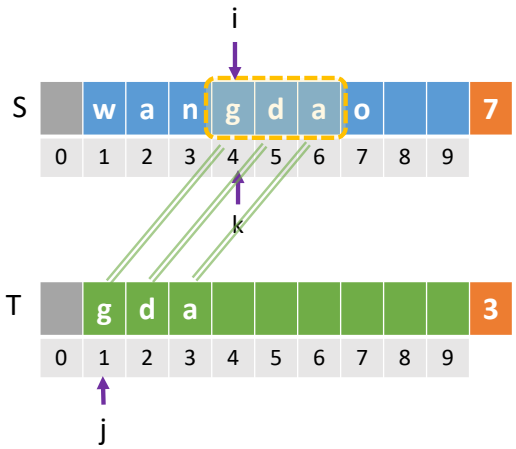
```
int Index(SString S,SString T){
    int k=1;
    int i=k, j=1;
    while(i<=S.length && j<=T.length){
        if(S.ch[i]==T.ch[j]){
            ++i;
            ++j;      //继续比较后继字符
        } else{
            k++;      //检查下一个子串
            i=k;
            j=1;
        }
    }
    if(j>T.length)
        return k;
    else
        return 0;
}
```



7

朴素模式匹配算法

```
int Index(SString S,SString T){
    int k=1;
    int i=k, j=1;
    while(i<=S.length && j<=T.length){
        if(S.ch[i]==T.ch[j]){
            ++i;
            ++j;      //继续比较后继字符
        } else{
            k++;      //检查下一个子串
            i=k;
            j=1;
        }
    }
    if(j>T.length)
        return k;
    else
        return 0;
}
```

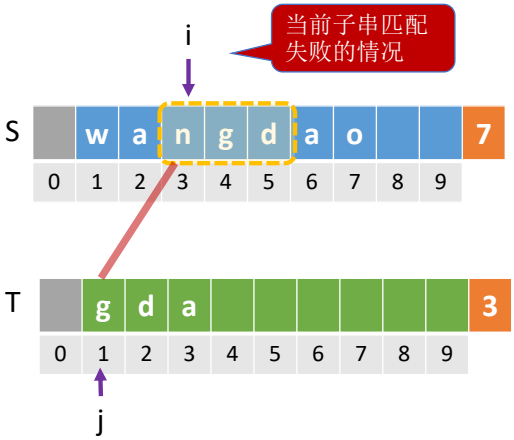


8

朴素模式匹配算法

课本的代码实现

```
int Index(SString S,SString T){
    int i=1,j=1;
    while(i<=S.length && j<=T.length){
        if(S.ch[i]==T.ch[j]){
            ++i; ++j;    //继续比较后继字符
        }
        else{
            i=i-j+2;
            j=1;         //指针后退重新开始匹配
        }
    }
    if(j>T.length)
        return i-T.length;
    else
        return 0;
}
```

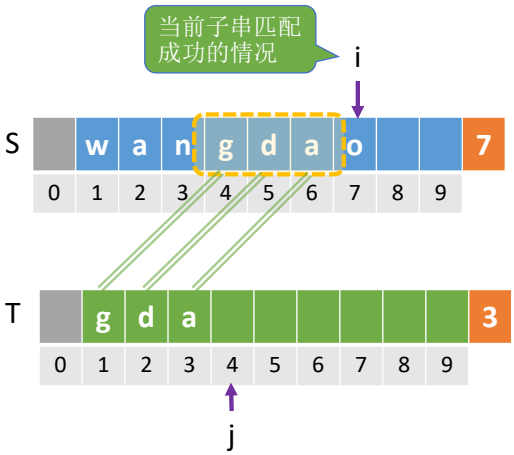


9

朴素模式匹配算法

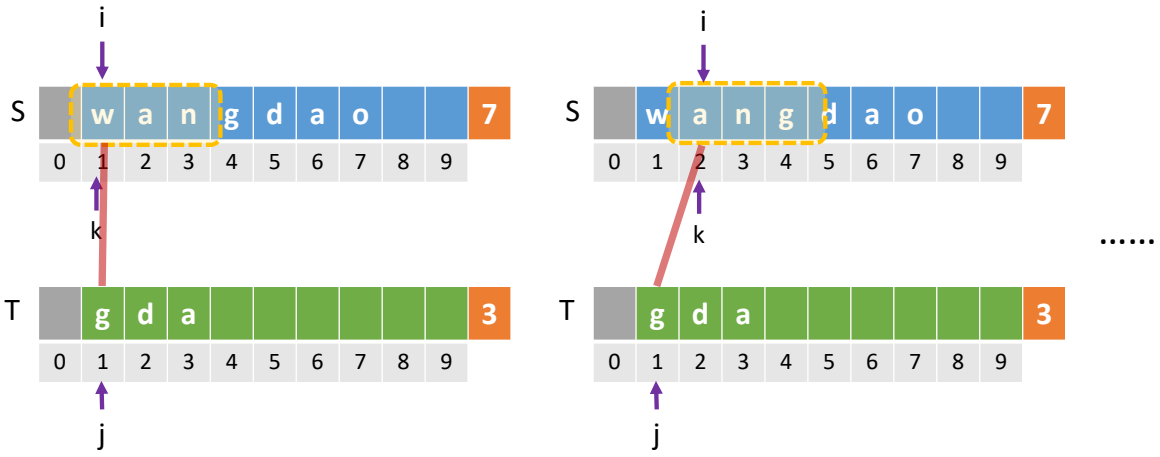
课本的代码实现

```
int Index(SString S,SString T){
    int i=1,j=1;
    while(i<=S.length && j<=T.length){
        if(S.ch[i]==T.ch[j]){
            ++i; ++j;    //继续比较后继字符
        }
        else{
            i=i-j+2;
            j=1;         //指针后退重新开始匹配
        }
    }
    if(j>T.length)
        return i-T.length;
    else
        return 0;
}
```



10

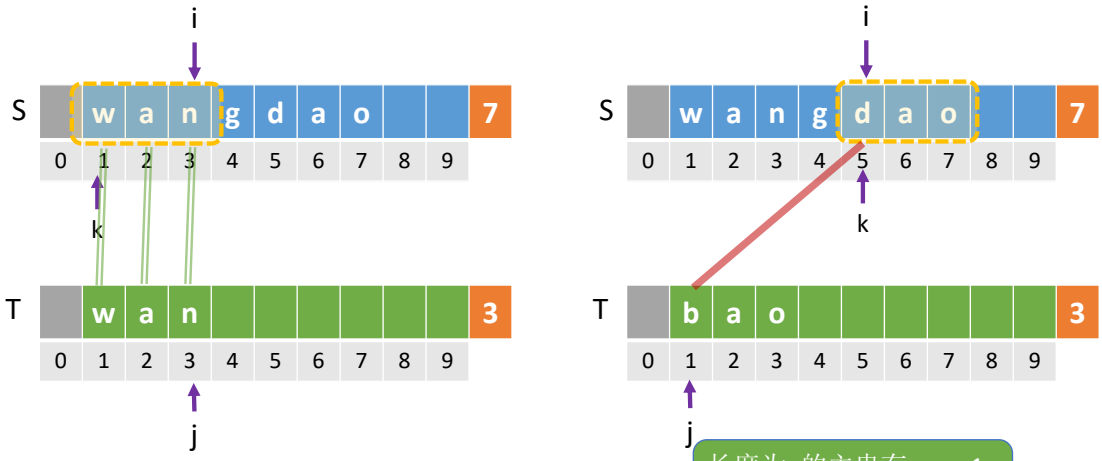
朴素模式匹配算法性能分析



较好的情况：每个子串第一个字符就与模式串不匹配

11

朴素模式匹配算法性能分析



若模式串长度为m，主串长度为n，则
匹配成功的最好时间复杂度：O(m)
匹配失败的最好时间复杂度：O(n-m+1) = O(n-m)≈O(n)

长度为n的主串有 n-m+1 个长度为m的子串

12

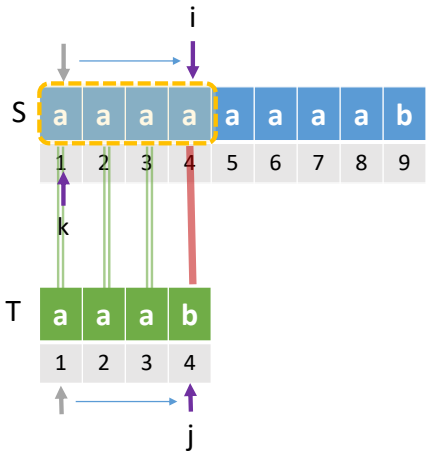
朴素模式匹配算法性能分析

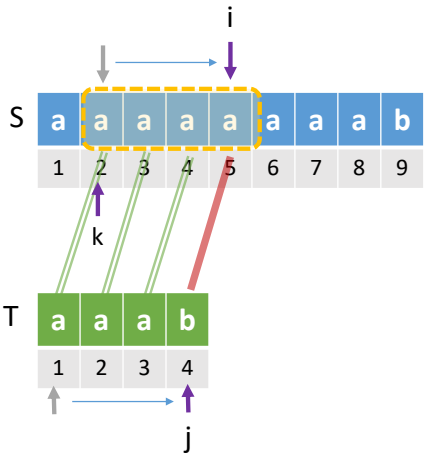
很多应用场景中，主串比模式串长很多，即 n>>m

O(n-m)≈O(n)

13

朴素模式匹配算法性能分析





.....

若模式串长度为m，主串长度为n，则直到匹配成功/匹配失败最多需要 (n-m+1)*m 次比较

最坏时间复杂度：O(nm)

14

知识回顾与重要考点

串的模式匹配：在主串中找到与模式串相同的子串，并返回其所在位置。

Tips：应该用你自己习惯的方式来记忆各种算法，而不是背课本

朴素模式匹配算法（简单模式匹配算法）思想：
将主串中与模式串长度相同的子串搞出来，挨个与模式串对比
当子串与模式串某个对应字符不匹配时，就立即放弃当前子串，转而检索下一个子串

若模式串长度为m，主串长度为n，则直到匹配成功/匹配失败最多需要 (n-m+1)*m 次比较
最坏时间复杂度：O(nm)
最坏情况：每个子串的前m-1个字符都和模式串匹配，只有第m个字符不匹配

比较好的情况：每个子串的第1个字符就与模式串不匹配

15