字符串匹配算法

本章重点：

> 1、暴力匹配（BF）算法
>
> 2、KMP算法

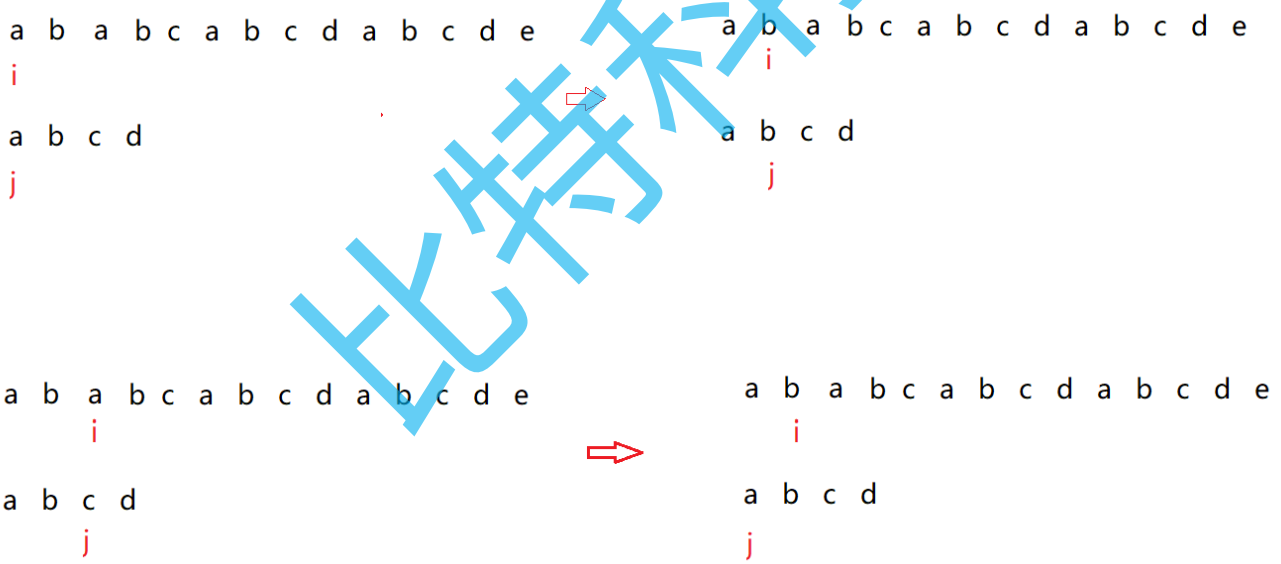
BF算法

BF算法，即**暴力(Brute Force)算法**，是普通的模式匹配算法，BF算法的思想就是将目标串S的第一个字符与模式串T的第一个字符进行匹配，若相等，则继续比较S的第二个字符和 T的第二个字符；若不相等，则比较S的第二个字符和T的第一个字符，依次比较下去，直到得出最后的匹配结果。BF算法是一种蛮力算法。

----这段话来自百度百科。

这段话晦涩难懂，需要例子支持。下面我们就通过例子来解释这个问题。

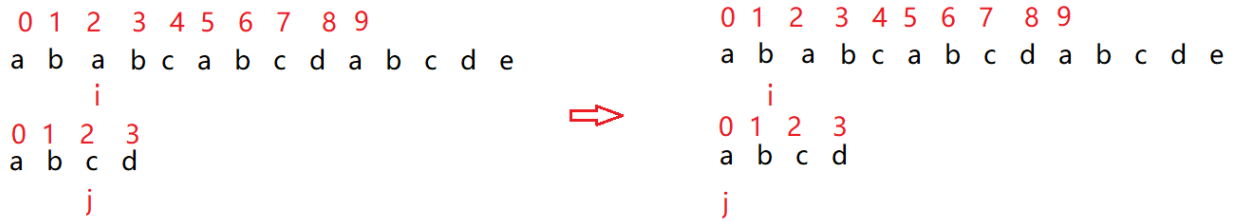
假定我们给出字符串 ”ababcabcdabcde”作为主串， 然后给出子串： ”abcd”,现在我们需要查找子串是否在主串中出现，出现返回主串中的第一个匹配的下标，失败返回-1；



比特科技

只要在匹配的过程当中，匹配失败，那么：i回退到刚刚位置的下一个,j回退到0下标重新开始。

接上图匹配失败，回退



匹配成功的情况如下

对应C代码：

```c
/**
Author: 高博
Date: 2021-08-02 21:54
str:主串
sub:子串
*/
int BF(char *str,char *sub)
{
    assert(str != NULL && sub != NULL);
    if(str == NULL || sub == NULL)
    {
        return -1;
    }

    int i = 0;
    int j = 0;
    int strLen = strlen(str);
    int subLen = strlen(sub);
    while(i  < strLen && j < subLen)
    {
        if(str[i] == sub[j])
        {
            i++;
            j++;
        }
        else
        {
            //回退
            i = i-j+1;
            j = 0;
        }
    }
```

比特科技

```
    if(j >= subLen)
    {
        return i-j;
    }
    return -1;
}
int main()
{
    printf("%d\n",BF("ababcabcdabcde","abcd"));
    printf("%d\n",BF("ababcabcdabcde","abcde"));
    printf("%d\n",BF("ababcabcdabcde","abcdef"));
    return 0;
}
```

对应Java代码：

```
/**
 * Created by GAOBO
 * Description:
 * User: GAOBO
 * Date: 2021-08-02
 * Time: 22:05
 */
public class Test {

    public static int BF(String str,String sub) {
        if(str == null || sub == null) return -1;
        int strLen = str.length();
        int subLen = sub.length();
        int i = 0;
        int j = 0;
        while (i < strLen && j < subLen) {
            if(str.charAt(i) == sub.charAt(j)) {
                i++;
                j++;
            }else {
                i = i-j+1;
                j = 0;
            }
        }
        if(j >= subLen) {
            return i-j;
        }
        return -1;
    }

    public static void main(String[] args) {
        System.out.println(BF("ababcabcdabcde","abcd"));
        System.out.println(BF("ababcabcdabcde","abcde"));
        System.out.println(BF("ababcabcdabcde","abcdef"));
    }
}
```

比特科技

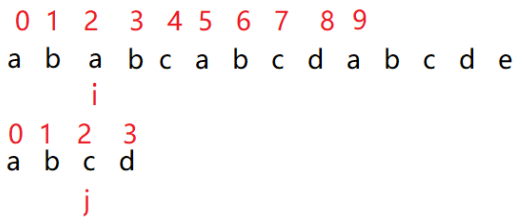
时间复杂度分析：最坏为O(m*n); m是主串长度，n是子串长度

KMP算法

KMP算法是一种改进的字符串匹配算法，由D.E.Knuth，J.H.Morris和V.R.Pratt提出的，因此人们称它为克努特—莫里斯—普拉特操作（简称KMP算法）。KMP算法的核心是利用匹配失败后的信息，尽量减少模式串与主串的匹配次数以达到快速匹配的目的。具体实现就是通过一个next()函数实现，函数本身包含了模式串的局部匹配信息。KMP算法的时间复杂度O(m+n) [1] 。 来自-------百度百科。

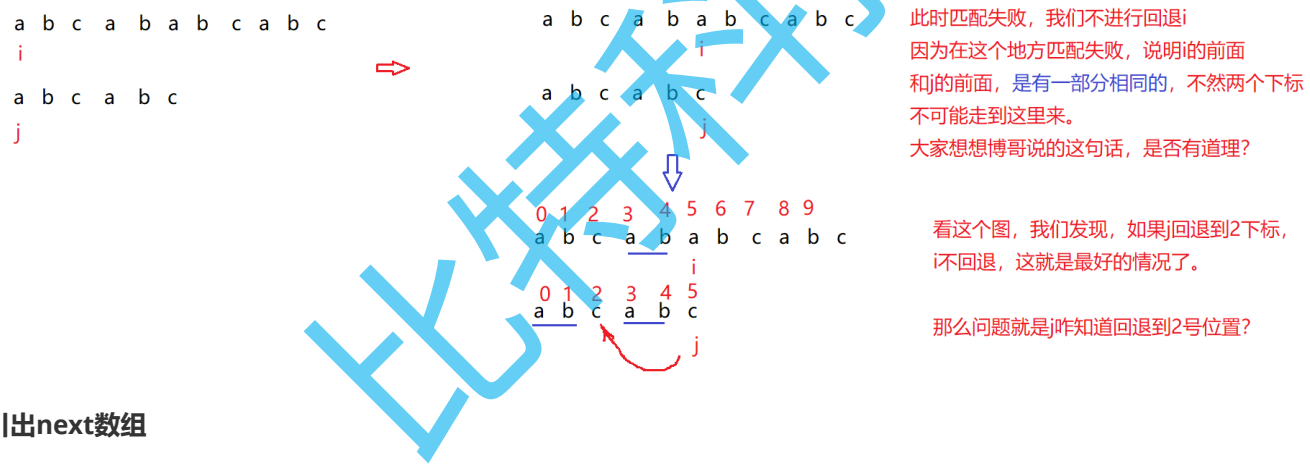
区别：KMP 和 BF 唯一不一样的地方在，我主串的 i 并不会回退，并且 j 也不会移动到 0 号位置

1 首先举例，为什么主串不回退?



假设目前在2号位置匹配失败，就算回退到1位置，也是么必要的。1位置的字符b 和子串0位置的a，也不一样

2 j的回退位置



比特科技

引出next数组

KMP 的精髓就是 next 数组：也就是用 next[j] = k;来表示，不同的 j 来对应一个 K 值， 这个 K 就是你将来要移动的 j 要移动的位置。

而 K 的值是这样求的：

> 1、规则：找到匹配成功部分的两个相等的真子串（不包含本身），一个以下标 0 字符开始，另一个以 j-1 下标字符结尾。
>
> 2、不管什么数据 next[0] = -1;next[1] = 0;在这里，我们以下标来开始，而说到的第几个第几个是从 1 开始;

求next数组的练习：

> 练习 1： 举例对于"ababcabcdabcde"， 求其的 next 数组?
>
> -1 0 0 1 2 0 1 2 0 0 1 2 0 0
>
> 练习 2： 再对"abcabcabcabcdabcde",求其的 next 数组? "
>
> -1 0 0 0 1 2 3 4 5 6 7 8 9 0 1 2 3 0

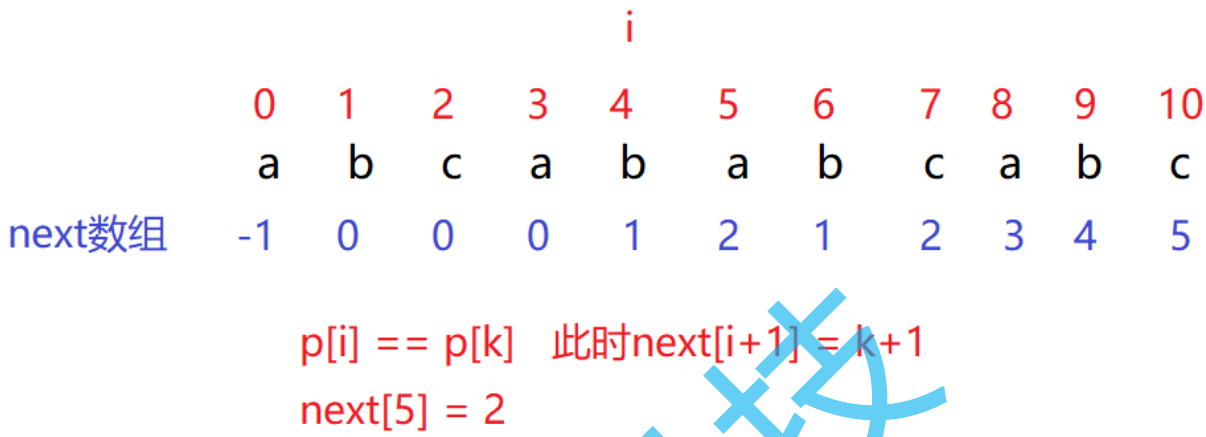
到这里大家对如何求next数组应该问题不大了，接下来的问题就是，已知next[i] = k;怎么求next[i+1] = ?

如果我们能够通过 next[i]的值,通过一系列转换得到 next[i+1]得值，那么我们就能够实现这部分。

那该怎么做呢?

首先假设： next[i] = k 成立，那么，就有这个式子成立： P0...Pk-1 = Px...Pi-1;得到： P0...Pk-1 = Pi-k..Pi-1;

到这一步：我们再假设如果 Pk = Pi;我们可以得到 P0...Pk = Pi-k..Pi;那这个就是 next[i+1] = k+1;



那么： Pk != Pi 呢?

看如下实例：

此时p[i] != p[k] 那么k要继续回退到0下标
也就是新的k_new = next[k];
进一步回退后p[0] == p[i] 呢么就否和next[i+1] = k_new+1;
也就是next[6] = 0+1 = 1

比特科技

```
void GetNext(int *next,const char *sub)
{

    int lensub = strlen(sub);
    next[0] = -1;
    next[1] = 0;
```

```
    int i = 2;//下一项
    int k = 0;//前一项的K
    while(i < lensub)//next数组还没有遍历完
    {
        if((k == -1) || sub[k] == sub[i-1])//
        {
            next[i] = k+1;
            i++;
            k++;//k = k+1???//下一个K的值新的K值
        }
        else
        {
            k = next[k];
        }
    }
}



int KMP(const char *s,const char *sub,int pos)
{
    int i = pos;
    int j = 0;

    int lens = strlen(s);
    int lensub = strlen(sub);

    int *next = (int *)malloc(lensub*sizeof(int));//和子串一样长
    assert(next != NULL);

    GetNext(next,sub);



    while(i < lens && j < lensub)
    {
        if((j == -1) || (s[i] == sub[j]))
        {
            i++;
            j++;
        }
        else
        {
            j = next[j];
        }
    }
    free(next);
    if(j >= lensub)
    {
        return i-j;
    }
    else
    {
```

比特科技

```
        return -1;
    }
}


int main()
{
    char *str = "ababcabcdabcde";
    char *sub = "abcd";
    printf("%d\n",KMP(str,sub,0));
    return 0;
}
```

Java代码：

```
public static void getNext(int[] next, String sub){
    next[0] = -1;
    next[1] = 0;
    int i = 2;//下一项
    int k = 0;//前一项的K
    while(i < sub.length()){//next数组还没有遍历完
        if((k == -1) || sub.charAt(k) == sub.charAt(i-1)) {
            next[i] = k+1;
            i++;
            k++;
        }else{
            k = next[k];
        }
    }
}

public static int KMP(String s,String sub,int pos) {
    int i = pos;
    int j = 0;
    int lens = s.length();
    int lensub = sub.length();

    int[] next= new int[sub.length()];

    getNext(next,sub);

    while(i < lens && j < lensub){
        if((j == -1) || (s.charAt(i) == sub.charAt(j))){
            i++;
            j++;
        }else{
            j = next[j];
        }
    }
    if(j >= lensub) {
        return i-j;
    }else {
        return -1;
```

比特科技

```
    }
}


public static void main(String[] args) {
    System.out.println(KMP("ababcabcdabcde","abcd",0));
    System.out.println(KMP("ababcabcdabcde","abcde",0));
    System.out.println(KMP("ababcabcdabcde","abcdef",0));
}
```

next数组的优化

next 数组的优化，即如何得到 nextval 数组：有如下串： aaaaaaaab,他的 next 数组是-1,0,1,2,3,4,5,6,7.而修正后的数组 nextval 是:
-1, -1, -1, -1, -1, -1, -1, -1, 7。为什么出现修正后的数组，假设在 5 号处失败了，那退一步还是a，还是相等，接着退还是 a。

练习：模式串 t='abcaabbcabcaabdab' ，该模式串的 next 数组的值为 (D) ， nextval 数组的值为 (F) 。

A. 0 1 1 1 2 2 1 1 1 2 3 4 5 6 7 1 2

B. 0 1 1 1 1 2 1 2 1 1 2 3 4 5 6 1 1 2

C. 0 1 1 1 0 0 1 3 1 0 1 1 0 0 7 0 1

D. 0 1 1 1 2 2 3 1 1 2 3 4 5 6 7 1 2

E. 0 1 1 0 0 1 1 1 0 1 1 0 0 1 7 0 1

F. 0 1 1 0 2 1 3 1 0 1 1 0 2 1 7 0 1

	a	b	c	a	a	b	b	c	a	b	c	a	a	b	d	a	b
next	-1	0	0	0	1	1	2	0	0	1	2	3	4	5	6	0	1
nextval	-1	0	0	-1	1	0	2	0	-1	0	0	-1	1	0	6	-1	0

A. 0 1 1 1 2 2 1 1 1 2 3 4 5 6 7 1 2　　B. 0 1 1 1 2 1 2 1 1 2 3 4 5 6 1 1 2

C. 0 1 1 1 0 0 1 3 1 0 1 1 0 0 7 0 1　　D. 0 1 1 1 2 2 3 1 1 2 3 4 5 6 7 1 2

E. 0 1 1 0 0 1 1 1 0 1 1 0 0 1 7 0 1　　F. 0 1 1 0 2 1 3 1 0 1 1 0 2 1 7 0 1

a	a	a	a	a	a	a	a	b
-1	0	1	2	3	4	5	6	7
-1	-1	-1	-1	-1	-1	-1	-1	7

nextval数组的求法很简单，如果当前回退的位置，正好是和当前字符一样，那么就写那个字符的nextval值。不一样就写自己的。
如上面例子中：4下标的a,应该回退当1号位置，1号位置不是a。所以，nextval值，就是当前的next值。
比如5下标的b，应该回退到1位置，正好1位置也是b.那么此时的nextval值，就是1下标b的nextval值。

比特科技