• 作业1：

1） 手动计算next数组以及KMP串匹配手动推导过程。

2）并且算一下做了多少次文本与模式的子串对齐，做了多少次字符比较?

3）如果是BF算法进行比较要对齐多少次?

4）KMP算法更快的原因是什么?

文本串S GCACTGCAGCACAGCAGCAGTACG

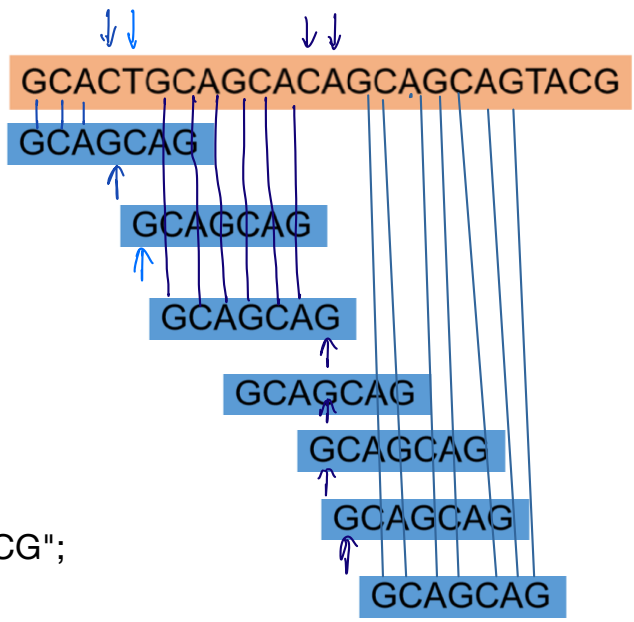
模式串T GCAGCAG

解：(1)

k	-1	0	-1	0	-1	0	1	2	3
j	0	1		2		3	4	5	6
T[j]	G	C		A		G	C	A	G
T[k]		G		G		G	C	A	G
next[j]	-1	0		0		0	1	2	3

next = [-1, 0, 0, 0, 1, 2, 3]

(2)&(3)




```
#include <stdio.h>

#include <string.h>

int Kmp (char* S, char* T,int* num,int* next);

int BF (char* S, char* T,int* num);

int main(void)

{

    char S[]="GCACTGCAGCACAGCAGCAGTACG";

    char T[]="GCAGCAG";

    int numkmp[2]={0,0};

    int numbf[2]={0,0};

    int next[7]={-1,0,0,0,1,2,3};

    Kmp(S,T,numkmp,next);

    BF(S,T,numbf);

    printf("KMP: 比较次数是%d ， 对齐次数是%d\n",numkmp[0],numkmp[1]);

    printf("BF: 比较次数是%d ， 对齐次数是%d",numbf[0],numbf[1]);

    return 0;
```

```
}
int Kmp (char* S, char* T,int* num,int* next){
    int i = 0;  int j = 0;
    int LenS = strlen(S);
    int LenT = strlen(T);
    while (i < LenS && j < LenT) {
      if (j == -1 || S[i] == T[j]) {
           i++;
           j++;
      }
      else{
            j = next[j];
            num[1]++;
       }
       num[0]++;
     }
     if (j == LenT)
          return i - j;
     else
          return -1;
}
int BF (char* S, char* T,int* num){
    int i = 0;  int j = 0;
    int LenS = strlen(S);
    int LenT = strlen(T);
    while (i < LenS && j < LenT) {
      if (S[i] == T [j]) {
           i++;
```

```
            j++;
        }
        else {
            i=i-j+1;
            j = 0 ;
            num[1]++;
        }
        num[0]++;
    }
    if (j == LenT)
        return i - j;
    else
        return -1;
}
```

```
KMP: 比较次数是27 ， 对齐次数是7
BF: 比较次数是32 ， 对齐次数是13
--------------------------------
Process exited after 0.1391 seconds with return value 0
请按任意键继续. . .
```

（4）KMP算法（Knuth-Morris-Pratt算法）相比BF算法（Brute-Force算法）更快的主要原因在于它有效地利用了模式串中的信息，通过构建next数组来避免在文本串中不必要的字符比较。以下是KMP算法相对于BF算法的优势和更快的原因：

- 避免不必要的字符比较：

  KMP算法根据已经匹配的前缀信息，能够在文本串中跳过多个字符，而不是每次只移动一个字符。这减少了字符比较的次数，特别是在模式串较长、文本串较长或包含大量不匹配的情况下。

- next数组的构建：

  KMP算法中的next数组通过预处理模式串来计算，该数组存储了模式串中每个位置的最长匹配前缀的长度。

当在文本串中发生不匹配时，KMP算法利用next数组中的信息来决定如何移动模式串，从而跳过已经匹配的部分，而不是重新从头开始比较。

- 前缀和后缀信息：

  KMP算法基于前缀和后缀信息，能够有效地避免在文本串中重复比较相同的子串，从而提高了匹配效率。

  BF算法每次只移动一个字符，而KMP算法能够充分利用前面已匹配的信息，减少了不必要的比较。