串

定长顺序存储

① typedef struct {
    char ch[MAXSIZE];
    int len;
} String;

② typedef unsigned char String[MAXSIZE+1];
    //0号位置放长度

堆分配存储

typedef struct {
    char *ch;
    int len;
} String;

块链连存储

typedef struct Node{
    char ch[SIZE];
    Node *next;
} Node;
typedef struct {
    Node *head, *tail;
    int len;
} String;

模式匹配算法（求子串位置）

①
```
int index(String S, String T, int pos){
    // 返回T在S的pos后的起始位置
    // String第1个字符不拆为1
    int i=pos, j=1;
    for(i<=S.len && j<=T.len){
        if(S.ch[i]==T.ch[j]){
            i++; j++;
        } else{
            i=1; i=i-j+2; // i=(i-j+1)+1 起始位置的后一个
        }
    }
    if(j>T.len) return i-T.len;
    else return 0;
}
```

最坏时间复杂度 O(nm)

② KMP 时间复杂度 $O(n+m)$

原理 右移位数 = 已匹配字符数 - 对应的部分匹配值

对应部分匹配值 = 已匹配部分最长相等前后缀长度

```
void get_next (string T, int next[]){
    int i=1, j=0;
    next[1]=0;
    while(i<T.len){
        if(j==0 || T.ch[i]==T.ch[j]){
            i++; j++;
            next[i]=j;
        }else{
            j=next[j];
        }
    }
}
int index (string S, String T, int pos){
    int i=pos, j=1;
    while( i<=S.len && j<=T.len){
        if(S.ch[i]==T.ch[j]||j==0){
            i++; j++;
        }else{
            j = next[j];
        }
    }
    if(j>T.len) return i-T.len;
    else return 0;
}
```

③ 改进KMP（用nextval代替next）

```
void get_nextval(String T, int nextval[]){
    int i=1, j=0;
    nextval[1]=0;
    while(i<T.len){
        if(j==0 || T.ch[i]==T.ch[j]){
            i++; j++;
            if(T.ch[i]!=T.ch[j])
                nextval[i]=j;
            else nextval[i]=nextval[j];
        } else {
            j=nextval[j];
        }
    }
}
```