第5周小结

1 串的存储结构

❶ 串是一种特殊的线性表，是线性表的一个子集。

- 顺序串
- 链串

❷ 链串只能采用单链表吗？

- 不一定。需要根据需要情况而定。
- 如果需要从某个节点出发前后查找，可以采用双链表。
- 如果需要快速查找尾节点，可以采用循环双链表。

2 串的算法设计

❶ 串的基本算法设计

借鉴线性表的算法设计方法。

- 顺序串 ⇔ 顺序表
- 链　串 ⇔ 单链表

❷ 串的模式匹配算法设计

- BF算法
- KMP算法

- 为什么KMP算法平均性能更高？
- 是不是任何情况下KMP算法都好于BF算法？

假设串采用顺序结构存储。设计一个算法求串s中出现的第一个最长重复子串的下标和长度。

解：（*i*，*len*）记录当前重复子串，（*maxi*，*maxlen*）记录第一个最长重复子串。

i=0，*len*=1　　*maxi*=0，*maxlen*=0

i=1，*len*=2　　*maxi*=0，*maxlen*=1

i=2，*len*=1　　*maxi*=1，*maxlen*=2

i=3，*len*=3　　*maxi*=1，*maxlen*=2

maxi=3，*maxlen*=3

maxi=3，*maxlen*=3，即“*abc*”

算法如下：

```
void maxsubstr(SqString s，SqString &t)
{     int maxi=0，maxlen=0，len，i，j，k;
      i=0;
      while (i<s.length)     //从下标为i的字符开始
      {     j=i+1;           //从i的下一个位置开始找重复子串
```

```
    while  (j<s.length)
    {      if  (s.data[i]==s.data[j])  //找一个子串，其起始下标为i，长度为len
           {      len=1;
                  for (k=1;s.data[i+k]==s.data[j+k];k++)
                         len++;
                  if  (len>maxlen)        //将较大长度者赋给maxi与maxlen
                  {       maxi=i;
                         maxlen=len;
                  }
                  j+=len;
           }
           else j++;
    }
    i++;                                  //继续扫描第i字符之后的字符
}
```

```
    t.length=maxlen;                    //将最长重复子串赋给t
    for  (i=0;i<maxlen;i++)
        t.data[i]=s.data[maxi+i];
}
```

```
int main()
{     SqString s,t;
      StrAssign(s,"aababcabcd");
      printf("s:"); DispStr(s);
      maxsubstr(s,t);
      printf("t:");DispStr(t);
      return 1;
}
```