# 河南工业大学信息学院物联网专业 2016 级数据结构作业

## 201616070320 物联网 1603 郭治洪

## （2017 年 11 月 13 日交）

第 1 章

分析以下程序段的时间复杂度，请说明分析的理由或原因。

⑴Sum1( int n )

```
{     int p=1, sum=0, m ;
for (m=1; m<=n; m++)
{    p*=m ; sum+=p ;    }
return (sum) ;
}
```

⑵Sum2( int n )

```
{     int sum=0, m, t ;
for (m=1; m<=n; m++)
{     p=1 ;
for (t=1; t<=m; t++)    p*=t ;
sum+=p ;
}
return (sum) ;
}
```

1.O(n) 一个递增循环从 1-n

2.O(n+m) 两个依次循环，一个 1-m，一个 1-n

第 2 章

1.在顺序表中插入和删除一个结点平均需要移动多少个结点?具体移动次数取决于哪两个因素?

由于是平均，所以各个结点的概率相同

(1/n)*(n-1)+(1/n)*(n-2)+...+(1/n)*1 = (1/n)*(1+2+...+(n-1)) = (n-1)/2

取决于要操作的**结点的位置**和**顺序表的长度**

2.设顺序表 L 是递增有序表，试写一算法，将 x 插入到 L 中并使 L 是递减有序表。

```
    void Mergelist_Sq(Sqlist LA,Sqlist LB,Sqlist &LC)
    {//已知顺序表 LA、LB，归并 LA、LB 得到顺序表 LC
    //时间复杂度 O(n)，n 为总元素数
        int *pa,*pb,*pc,*pa_last,*pb_last,i;
        pa=LA.elem; pb=LB.elem;//pa 为表 LA 的整数型数组地址，pb 表 LB 的整数型数组地址
        int *bak_pa=pa;int *bak_pb=pb;
       LC.listsize=LC.length=LA.length+LB.length;//表 LC 的长度和有效数据的长度为表 LA 和表 LC 的和
       pc=LC.elem=new int[LC.listsize * sizeof(ElemType)];//分配表 LC 的整数型数组能储存 LA 和 LB 中元素
       if (!LC.elem)
       {
           cout <<"不能分配内存"<<endl;
            exit(OVERFLOW);
       };
       pa_last=LA.elem+LA.length-1; //定义表 LA 的整形数组截至
       pb_last=LB.elem+LB.length-1;//定义表 LB 的整形数组截至
       while(pa<pa_last)//此循环会让表 LA 到达末尾元素
          pa++;
       while(pb<pb_last)//此循环会让表 LB 到达末尾元素
        pb++;
    //   cout<<*pa<<*pb<<endl;
       while(pa>=bak_pa&&pb>=bak_pb)
       {//此循环把小的元素放在 LC 前面
          if (*pa <= *pb) //如果表 LA 的元素小于表 LB 的元素
              *pc++ = *pb--;//赋值并且变成下一个元素
        else
              *pc++ = *pa--;
       }
       while (pa>=bak_pa) //不等于首个元素
              *pc++ = *pa--;
       while (pb>=bak_pb)
              *pc++ = *pb--;
        delete bak_pa,bak_pb;//删除为表 LA 和 LB 的分配空间
}
```

3.设 A 和 B 是两个按元素值递减有序的单链表，写一算法将 A 和 B 归并为按按元素值递增有序的单链表 C，试分析算法的时间复杂度。

时间复杂度 O(n)，n 为总元素数

```
void Mergelist_Sq(Sqlist LA,Sqlist LB,Sqlist &LC)
{//已知顺序表 LA、LB，归并 LA、LB 得到顺序表 LC
    int *pa,*pb,*pc,*pa_last,*pb_last,i;
    pa=LA.elem; pb=LB.elem;//pa 为表 LA 的整数型数组地址，pb 表 LB 的整数型数组地址
    int *bak_pa=pa;int *bak_pb=pb;
  LC.listsize=LC.length=LA.length+LB.length;//表 LC 的长度和有效数据的长度为表 LA 和表 LC 的和
  pc=LC.elem=new int[LC.listsize * sizeof(ElemType)];//分配表 LC 的整数型数组能储存 LA 和 LB 中元素
  if (!LC.elem)
  {
      cout <<"不能分配内存"<<endl;
       exit(OVERFLOW);
  };
  pa_last=LA.elem+LA.length-1; //定义表 LA 的整形数组截至
  pb_last=LB.elem+LB.length-1;//定义表 LB 的整形数组截至
  while(pa<pa_last)//此循环会让表 LA 到达末尾元素
     pa++;
  while(pb<pb_last)//此循环会让表 LA 到达末尾元素
    pb++;
//   cout<<*pa<<*pb<<endl;
  while(pa>=bak_pa&&pb>=bak_pb)
  {//此循环把小的元素放在 LC 前面
     if (*pa <= *pb) //如果表 LA 的元素小于表 LB 的元素
         *pc++ = *pa--;//赋值并且变成下一个元素
    else
         *pc++ = *pb--;
  }
  while (pa>=bak_pa) //不等于首个元素
         *pc++ = *pa--;
  while (pb>=bak_pb)
         *pc++ = *pb--;
    delete bak_pa,bak_pb;//删除为表 LA 和 LB 的分配空间
}
```

第 3 章

1. 设有一个栈，元素进栈的次序为 a, b, c。问经过栈操作后可以得到哪些输出序列？

a 进 a 出 b 进 b 出 c 进 c 出 abc

a 进 b 进 c 进 c 出 b 出 a 出 cba

a 进 b 进 b 出 a 出 c 进 c 出 bac

a 进 a 出 b 进 c 进 c 出 b 出 acb

a 进 b 进 b 出 c 进 c 出 a 出 bca

2.利用栈的基本操作，写一个返回栈 S 中结点个数的算法 int StackSize(SeqStack S) ，并说明 S 为何不作为指针参数的算法?

```
int StackSize(SeqStack s)
{
    int i=1;
    while(S.top!=S.base)
    {//出栈并判断栈是否为空
        --S.top;
        ++i;
    }
    return i;
 }
```

因为我们只需要元素出栈，并且判断栈是否为空，并不需要改变栈的值和对栈中的元素进行操作。

3.假设 Q[0,5]是一个循环队列，初始状态为 front=rear=0，请画出做完下列操作后队列的头尾指针的状态变化情况，若不能入队，请指出其元素，并说明理由。

d, e, b, g, h 入队

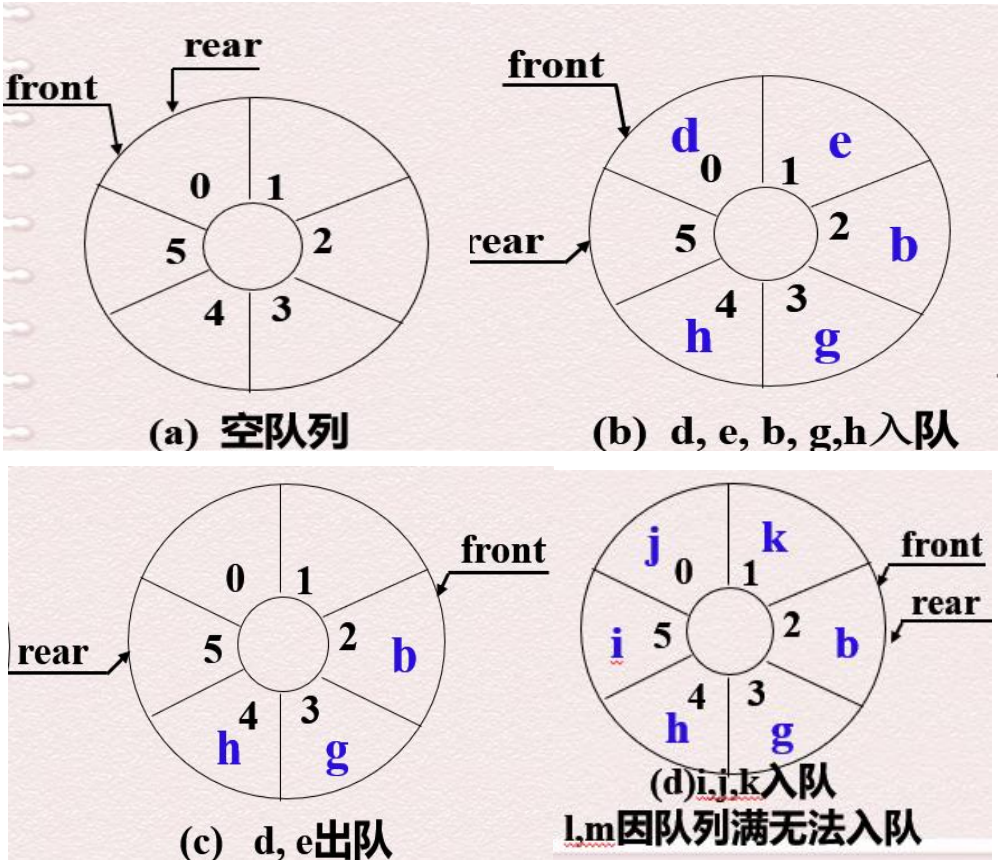
d, e 出队

i , j , k , l , m 入队

b 出队

n, o, p, q, r 入队

答案：图解



第 4 章

1. 若 x 和 y 是两个采用单链表结构存储的串，每块一个字符，写一算法比较这两个字符串是否相等。

```
int compare_string(StringList s,StringList t)
{//传过来是两个链表头节点的地址
 int pos=0;
 s=s->next;//变成首节点地址
 t=t->next;
 for(;s->next!=NULL&&t->next!=NULL;s=s->next,t=t->next)
 {//进行循环遍历，如果发现两个链表某个结点元素不同，则改变信号变量并且退出循环
      if(s->data!=t->data)
      {
           pos=1;
           break;
      }
 }
 if(pos==0)//如果两个字符串相等
      return 1;
 if(pos==1)//如果不相等
      return -1;
}
```

2. 写一算法 void StrRelace(char *T, char *P, char *S)，将 T 中第一次出现的与 P 相等的子串替换为 S，串 S 和 P 的长度不一定相等，并分析时间复杂度。

n 是 T 串长度，m 是 P 串长度，q 是 s 串长度

**要求 s<=q**

O(m+n+q+(m-n))=O(2m+q)

m+n 为 KMP 算法找到替换的位置

q 为复制 S 串

m-n 为复制剩下的部分

代码在下页。

```
//注：以下代码参考算法导论中伪代码
int *next(char *P)
{//计算 P 的部分匹配表
    int *num=(int *)new int[strlen(P) * sizeof(int)];
    if(!num)
    {
        cout<<"内存分配失败"<<endl;
        exit (-1);
    }
    num[0]=0;
    int k=0;
    for(int q=1;P[q]!='\0';q++)
    {//从第 2 位到字符串截止
        while(k>0&&(P[k]!=P[q]))//如果 k 大于 0 且 k+1 不等 k 位
            k=num[k-1];//k 值等于这一位的部分匹配值
        if(P[k]==P[q])//如果当前位和 k+1 相等
            k++;//k+1
        num[q]=k;//当前位部分匹配值为 k 值
    }
//  for(int i=0;i<strlen(P);i++)
//      cout<<num[i]<<endl;
    return num;
 }

void StrRelace(char *T, char *P, char *S)
{
    int find=0;
    int *num=next(P);//调用计算 P 的部分匹配表
    int i,q;
    for(i=0,q=0;T[i]!='\0';i++)
    {//从 T 字符串第 0 位开始，到它结束
        while(q>0&&P[q]!=T[i])
            q=num[q];
        if(P[q]==T[i])
            q++;
        if(q==strlen(P))
        {
            find=1;
            break;
        }
    }
```

```
    i++;//替换结束位置的下一位
    if(find==1)
    {//找到
        if(strlen(S)>strlen(P))//如果要替换的字符串超过匹配的字符串长度，提示错误
        {
            cout<<"替换字符串超过匹配字符串的长度"<<endl;
//          新长度=字符串 T 的长度-字符串 P 的长度+字符串 S 的长度
//          char * new_T = (char *)new char[(strlen(T)-strlen(P)+strlen(S))*sizeof(char)];
//          if(!new_T)
//          {
//              cout<<"内存分配失败"<<endl;
//              exit (-1);
//          }
//          int s;
//          for(s=0;s<i-strlen(P);s++)
//              new_T[s]=[s];//复制替换前的字符串
//          for(int m=0;S[m]!='\0';m++,s++)
//              new_T[s]=S[m];//复制要替换的字符串
//          for(;T[i]!='\0';i++,s++)
//              new_T[s]=T[i];//复制剩下部分字符串
//          new_T[s]='\0';//最后加上截至字符
//          T=new_T;
        }
        else
        {
            int s=i-strlen(P);//替换开始位置
            for(int m=0;S[m]!='\0';m++,s++)
                T[s]=S[m];//替换成 S 串
            for(;T[i]!='\0';i++,s++)
                T[s]=T[i];//剩下部分前移
            T[s]='\0';//最后加上截至字符
        }
    }
    else
        cout<<"找不到无法完成替换"<<endl;
}
```