《数据结构与算法》第一次作业

一、选择题

1. 现有两个带头结点的双向循环链表，其头结点分别为 M 和 N，现将头结点为 N 的链表接到头结点为 M 的链表尾部，L 为 N 链表的最后一个结点，则相应的指针操作为（ ）

A. L->next=M; M->prior=L; M->next->next=N; N->prior=M->next;

B. L->next=M; M->next->next=N; N->prior=M->next; M->prior=L;

C. L->next=M; M->prior->next=N; N->prior=M->prior; M->prior=L;

D. L->next=M; M->prior=L; M->prior->next=N; N->prior=M->prior;

【C】

2. 设栈 S 和队列 Q 的初始状态均为空，元素 abcdefg 依次进入栈 S。若每个元素出栈后立即进入队列 Q，且 7 个元素出队的顺序是 bdcfeag，则栈 S 的容量至少是（ ）。

A. 1　　B. 2　　C. 3　　D. 4

【C】

3. 假设 $a_1,a_2,\ldots,a_n$ 依次入栈（且 $a_1..a_n$ 是 1..n 的排列），出栈为 1,2,…,n。以下哪种说法正确 ？( )

A. 不存在 i<j<k，使得 $a_i<a_j<a_k$.

B. 不存在 i<j<k，使得 $a_k<a_j<a_i$.

C. 不存在 i<j<k，使得 $a_k<a_i<a_j$.

D. 不存在 i<j<k，使得 $a_j<a_i<a_k$.

【C】

4. 若用一个大小为 6 的数组来实现循环队列，且当前 rear 和 front 的值分别为 0 和 3，当从队列中删除一个元素，再加入两个元素后，rear，front 的值分别为多少？（ ）

A.1 和 5

B.2 和 4

C.4 和 2

D.5 和 1

【B】

5. 设串 s1=“ABCDEFG”，s2=“PQRST”函数 strconcat（s，t）返回 s 和 t 串的连接串，strsub（s，i，j）返回串 s 中从第 i 个字符开始的、由连续 j 个字符组成的子串。strlength(s)返回串 s 的长度。则 strconcat（strsub（s1，2，strlength（s2）），strsub（s1，strlength（s2），2））的结果串是（ ）。

A. BCDEF　　B. BCDEFG　　C. BCPQRST　　D. BCDEFEF

【D】

6. 如果主串和模式串的长度分别为 n 和 m，预处理模式串的 failure function 需要______时间。

A. O(n)　　B. O(m)　　C. O(n+m)　　D. O(nm)

【B】

二、简答题

1. 请用递推计算串“a b a a b a a b a b a”的 failure function π。

【答案】

【0　0　1　1　2　3　4　5　6　2　3】

2. 已知线性表中的元素以值递增有序排列，并以**单链表**做存储结构。试写一高效的算法，删除表中所有值**大于 a 且小于 b** 的元素（若存在），同时释放被删结点的空间，并分析时间复杂度（注意：a 和 b 是给定的两个参变量，它们的值可以和表中的元素相同，也可以不同）

【答案】

```
Status Algo_2_11(SqList *va, LElemType_Sq x)
{
    int i;
    LElemType_Sq *newbase;

    if(!(*va).length)
        return ERROR;

    if((*va).length==(*va).listsize)    //若存储空间已满，需开辟新空间
    {
        newbase = (LElemType_Sq*)realloc((*va).elem, ((*va).listsize+LISTINCREMENT)*sizeof
(LElemType_Sq));
        if(!newbase)
            exit(OVERFLOW);

        (*va).elem = newbase;
        (*va).listsize += LISTINCREMENT;
    }

    for(i=(*va).length; i>=1; i--)
    {
        if((*va).elem[i-1]>x)
            (*va).elem[i] = (*va).elem[i-1];
        else
            break;
    }

    (*va).elem[i] = x;
    (*va).length++;

    return OK;
}
```

```
/* 方法2 */
Status Algo_2_19_2(LinkList L, int mink, int maxk)
{
    LinkList p, pre, s;
```

```
    if(!L || !L->next)          //L不存在或为空表时，无法删除
        return ERROR;

    if(mink>=maxk)              //阙值设置错误
        return ERROR;

    pre = L;
    p = pre->next;              //p指向首结点


    while(p && p->data<=mink)   //下限
    {
        pre = p;
        p = p->next;
    }

    if(p)
    {
        while(p && p->data<maxk) //上限
        {
            s = p;
            pre->next = p->next;
            p = p->next;
            free(s);
        }
        return OK;
    }
}
```

时间复杂度分析：最坏的情况是全部扫描完也没找到适合的元素，故时间复杂度与链表长度有关，为 O(Length(L))。

3(Bonus 问题). 已知 Ackermann 函数定义如下:

$$\mathrm{Ack}(m,n)=\begin{cases} n+1 & \text{当}m=0\text{时} \\ \mathrm{Ack}(m-1,1) & \text{当}m\neq 0,n=0\text{时} \\ \mathrm{Ack}(m-1,\mathrm{Ack}(m,n-1)) & \text{当}m\neq 0,n\neq 0\text{时} \end{cases}$$

1) 写出计算 Ack(m,n)的递归算法，并根据此算法给出出 Ack(2,1)的计算过程。

2) 写出计算 Ack(m,n)的非递归算法。

【答案】

[算法描述]

```
int Ack(int m,n)
{ if (m==0) return(n+1);
  else if(m!=0&&n==0) return(Ack(m-1,1));
  else return(Ack(m-1,Ack(m,m-1));
}//算法结束
```

① Ack(2,1)的计算过程

Ack(2,1)= Ack(1,Ack(2,0))   //因 m<>0,n<>0 而得

　　　= Ack(1,Ack(1,1))   //因 m<>0,n=0 而得

　　　= Ack(1,Ack(0,Ack(1,0)))   // 因 m<>0,n<>0 而得

```
= Ack(1,Ack(0,Ack(0,1)))      // 因 m<>0,n=0 而得
= Ack(1,Ack(0,2))             // 因 m=0 而得
= Ack(1,3)                    // 因 m=0 而得
= Ack(0,Ack(1,2))             //因 m<>0,n<>0 而得
= Ack(0,Ack(0,Ack(1,1)))      //因 m<>0,n<>0 而得
= Ack(0,Ack(0,Ack(0,Ack(1,0)))) //因 m<>0,n<>0 而得
= Ack(0,Ack(0,Ack(0,Ack(0,1)))) //因 m<>0,n=0 而得
= Ack(0,Ack(0,Ack(0,2)))      //因 m=0 而得
= Ack(0,Ack(0,3))             //因 m=0 而得
= Ack(0,4)                    //因 n=0 而得
=5                            //因 n=0 而得
```

②

```
int Ackerman(int m, int n)
{  int akm[M][N];int i,j;
   for(j=0;j<N;j++) akm[0][j]=j+1;
   for(i=1;i<m;i++)
      {akm[i][0]=akm[i-1][1];
       for(j=1;j<N;j++)
           akm[i][j]=akm[i-1][akm[i][j-1]];
       }
 return(akm[m][n]);
} //算法结束
```