**第一次纸面作业**

无64 兰兴增 2016011082

1. 给以下程序加上必要的注释，指出其功能，并分析时间复杂度。

**(1) int sum(int n)** // 求1!+2!+……+n!

**{ int s=0**；

**for(int i=1**；**i<=n**；**i++)**

**{ int p=1**； // 重置循环变量p

**for(int j=1**；**j<=i**；**j++) p\*=j**； // 计算i!

**s+=p**； // 求和项加上i!

**}**

**return s**； // 时间复杂度为O(n\*(n+1)/2)

**}**

**(2) int fac(int n)** // 求1!+2!+……+n!

**{ int p=1,s=0**；

**for(int i=1**；**i<=n**；**i++)**

**{ p\*=i**； // 计算i!

**s+=p**； // 求和项加上i!

**}**

**return s**； // 时间复杂度为O(n)

**}**

2. 设数据元素的集合为D={a1,a2,a3,a4,a5,a6}，请分别画出与以下各关系R 对应的数据结构B=(D,R)的结构示意图，并指出它属于哪类结构。

(1) R={(a3,a4),(a4,a5),(a1,a2),(a2,a3),(a5,a6)}

(2) R={(a3,a2),(a2,a4),(a3,a1),(a2,a5),(a2,a6)}

(3) R={(ai+1,ai)︱i=5,4,3,2,1}

(4) R={(ai,aj)︱i>j}

(5) R={ }

(1)为线性结构

a6

a5

a4

a3

a2

a1

(2)为树结构

a3

a1

a2

a5

a6

a4

(3)为线性结构

a6

a5

a4

a3

a2

a1

(4)为图结构

a2

a1

a3

a4

a5

a6

(5)为集合结构

a6

a3

a1

a2

a5

a4

3. 分别写一个算法，对**带表头结点**和**不带表头结点**的情况，实现将两个有序的线性链表合并为一个有序链表，合并后使原链表为空。

1. 带表头

void mergeList(LinkList &HA,LinkList &HB,LinkList &HC){

/\*HA和HB为非递减有序线性链表，合并为非递减有序线性链表HC

\*HA.head是第一个结点

\*结点数据类型为

\* struct SNode{

\* int data;

\* SNode \*next;

\* SNode():data(0),next(NULL){}

\* };

\*/

SNode \* pHA = &HA.head, \* pHB = HB.head.next, \* pHC = &HC.head;

while (pHA != NULL && pHB != NULL) // 当链表非空或指针不指向表尾时

{

pHA = pHA->next;

while ((pHA != NULL && pHB != NULL)

{

if (pHB->data >= pHA->data)

{

pHC->next = pHA; // 若当前结点HB.data>=HA.data

pHC = pHC->next; // 则pHC->next指向pHA，pHC后移

break; // 之后pHA向后移，并判断是否为空

}

else

{

pHC->next = pHB; // 若当前结点HB.data<HA.data

pHC = pHC->next; // 则pHC->next指向pHB，pHC后移

pHB = pHB->next; // 之后pHB向后移，并判断是否为空

}

}

}

// 上述操作后pHA和pHB中必有一者为空，则将其中不为空者（若存在）

// 指向的后续data均按序合并到HC中即可。

// 若HA、HB均为空表，则pHA和pHB均为NULL，不进行操作。

if (pHA != NULL)

{

pHC->next = pHA;

}

if (pHB != NULL)

{

pHC->next = pHB;

}

// 将HA和HB置为空表

HA.head.next = NULL;

HB.head.next = NULL;

}

1. 不带表头

void mergeList(LinkList &HA,LinkList &HB,LinkList &HC){

SNode \* pHA = HA.head, \* pHB = HB.head, \* pHC = HC.head;

if (pHA == NULL || pHB == NULL) // 若两者中存在空表

{

if (pHA == NULL && pHB != NULL)

{

pHB = pHB->next;

pHC = pHB;

HB.head = NULL;

return;

}

else if (pHA != NULL && pHB == NULL)

{

pHA = pHA->next;

pHC = pHA;

HA.head = NULL;

return;

}

else

{

return;

}

}

// 若两者均非空表

pHB = pHB->next;

while (pHA != NULL && pHB != NULL) // 当指针不指向表尾时

{

pHA = pHA->next;

while (pHA != NULL && pHB != NULL)

{

if (pHB->data >= pHA->data) // 若当前结点HB.data>=HA.data

{

if (pHC == HC.head) // 若pHC为HC头指针

{

pHC = pHA;

}

else

{

pHC->next = pHA;

pHC = pHC->next;

}

break; // 之后pHA向后移，并判断是否为空

}

else // 若当前结点HB.data<HA.data

{

if (pHC == HC.head)

{

pHC = pHB;

}

else

{

pHC->next = pHB;

pHC = pHC->next;

}

pHB = pHB->next; // 之后pHB向后移，并判断是否为空

}

}

}

// 上述操作后pHA和pHB中必有一者为空，则将其中不为空者（若存在）

// 指向的后续data均按序合并到HC中即可。

if (pHA != NULL)

{

pHC->next = pHA;

}

if (pHB != NULL)

{

pHC->next = pHB;

}

HA.head = NULL;

HB.head = NULL;

}

4.分别实现逆转顺序表和逆转链表的算法，使表中最后一个元素成为第一个元素，倒数第二个元素成为第二个元素，以此类推。对于链表逆序算法，必须交换结点，而不是结点中的数据。

(1)

class SeqList{

public:

int length;

int \*list;

SeqList(int n){

length=n;

list=new int[n]; //分配一个足够大的空间

}

};

void inverseSeqList(SeqList &L){

int temp;

for (int i = 0; i < int(L.length/2); i++)

{

temp = L.list[i];

L.list[i] = L.list[L.length-i-1];

L.list[L.length-i-1] = temp;

}

}

(2)

void inverseLinkList(LinkList &HL){

//LinkList和SNode定义同上一题，链表不带表头

SNode \* pPrior = NULL, \*pNext;

while (HL.head != NULL)

{

pNext = HL.head->next; // 标记下一元素

HL.head->next = pPrior; // 将当前结点的next指针指向前驱元素

pPrior = HL.head; // 将当前结点标记为前驱元素

HL.head = pNext; // 当前结点后移

}

// 循环结束后HL.head为空，pPrior所指向的元素应为

// 新链表的第一个元素(即原链表最后一个元素)。

// 空表仍为空。

HL.head = pPrior;

}

5. 如果3个元素进栈顺序为X、Y、Z，试写出所有可能的出栈顺序（可能后来者进栈时，先来者已经出栈）。

如果3个元素进栈顺序为X、Y、Z，试写出所有可能的出栈顺序

出栈顺序可能为：

1. X、Y、Z

2. X、Z、Y

3. Y、X、Z

4. Y、Z、X

5. Z、Y、X

6. 给以下程序加上必要的**注释**，指出其**功能**，并分析**时间复杂度。**

**void writ(int n)** // 输出1~n

**{ if(n!=0)**

**{ writ(n-1);** // 递归调用（即cout<<n-1<<endl;）

**cout<<n<<endl;**

**return;**

**}**

**}**

入栈出栈均为n次操作，时间复杂度为O(n)，实际上操作了2n次。可以递推n次操作实现。