1. 调试成功程序

1.

题目：设元素值为整型的线性表L，分别采用顺序结构和链式结构存储，编写函数，用选择/冒泡排序算法实现线性表的表排序。

算法思想：见源代码

源代码：

#include<iostream>

**using** **namespace** std;

//线性表类创建

**class** SqList

{

**public**:

**int** \*v=**new** **int**[1000];

**int** length=0;

};

//线性表冒泡排序 从小到大 大数下沉 小数上浮

**void** SqListBubble\_Sort(SqList &L)

{

//只需比较length-1轮即可 排到最后最后一项自然确定

**for**(**int** i=0;i<L.length-1;i++)

{

//每次都从最上面开始比较

**for**(**int** j=0;j<L.length-1-i;j++)//只比较到倒数第二项 所以要减个1 已经比较过的不用比较 所以要减去一个i

{

//比较当前项和当前项的下一项

**if**(L.v[j]>L.v[j+1])

{

**int** temp=L.v[j];

L.v[j]=L.v[j+1];

L.v[j+1]=temp;

}

}

}

}

//线性表选择排序 从小到大

**void** SqListSelect\_Sort(SqList &L)

{

//只用比较length-1轮 比到最后最后一个的位置自然确定

**for**(**int** i=0;i<L.length-1;i++)

{

//每次开始比较前 先将最小值下标默认初始化为比较的第一位

**int** index=i;

//每次都从指定值后一位开始比较到最后

**for**(**int** j=i+1;j<=L.length-1;j++)

{

//每次找到符合情况后交换下标 以确保找到的是最小的值 最后再交换元素

**if**(L.v[j]<L.v[index])

{

**int** tempindex=j;

j=index;

index=tempindex;

}

}

//此时index中记录的是后续元素中的最小值 与i位交换即可

**int** temp=L.v[i];

L.v[i]=L.v[index];

L.v[index]=temp;

}

}

//线性表输出函数

**void** PrintSqList(SqList &L)

{

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

{

cout<<L.v[i]<<" ";

}

cout<<endl;

}

//线性表创建函数

**void** CreateSqList(SqList &L,**int** n)

{

**for**(**int** i=0;i<n;i++)

{

cin>>L.v[i];

L.length++;

}

}

//链表类创建

**class** LinkNode

{

**public**:

**int** data;

LinkNode \*next;

};

//创建链表函数

**void** CreateLinkList(LinkNode\* &L,**int** n)

{

LinkNode \*p,\*r;

r=L;

**for**(**int** i=0;i<n;i++)

{

p=**new** LinkNode;

cin>>p->data;

p->next=**NULL**;

r->next=p;

r=p;

}

}

//输出链表函数

**void** PrintLinkList(LinkNode\* &L)

{

LinkNode \*p=L->next;

**while**(p!=**NULL**)

{

cout<<p->data<<" ";

p=p->next;

}

cout<<endl;

}

//链表冒泡排序

**void** LinkListBubble\_Sort(LinkNode\* &L)

{

//统计链表中数据节点的个数

**int** count1=0;

LinkNode \*p=L->next;

**while**(p)

{

count1++;

p=p->next;

}

//进行冒泡排序

**for**(**int** i=0;i<count1-1;i++)

{

LinkNode \*p,\*q,\*r;

p=L;//需交换节点——前的前一位节点

q=L->next;//需交换节点——前

r=q->next;//需交换节点——后

//创建第二个计数器

**int** count2=0;

**while**(count2<count1-1-i)

{

**if**(q->data>r->data)

{

//节点

//1.q指向r的下一位

q->next=r->next;

//2.r指向q

r->next=q;

//3.纠正前一位指向至r

p->next=r;

//4.偏移 此时位置为p r q其中q的位置正确

p=r;

r=q->next;

count2++;

**continue**;

}

p=q;

q=r;

r=r->next;

count2++;

}

}

}

//链表选择排序 从小到大

**void** LinkListSelect\_Sort(LinkNode\* &L)

{

LinkNode \*p=L->next;

**int** count=0;

**while**(p)

{

count++;

p=p->next;

}

LinkNode \*lastq=L;

LinkNode \*q=L->next;

**for**(**int** i=0;i<count;i++)

{

LinkNode \*minp=q;

LinkNode \*lastminp=lastq;

LinkNode \*r=q->next;

LinkNode \*lastr=q;

**while**(r)

{

**if**((r->data)<(minp->data))

{

minp=r;

lastminp=lastr;

}

r=r->next;

lastr=lastr->next;

}

//此时minp保存了储存最小值节点的指针 只有当minp指向的值与q不同时才需要进行交换

**if**(q->data!=minp->data)

{

//不相邻时执行如下交换算法

**if**(minp!=q->next)

{

//1.删除q指向的节点

lastq->next=q->next;

//2.删除minp指向的节点

lastminp->next=minp->next;

//3.将minp插入q的位置

minp->next=lastq->next;

lastq->next=minp;

//4.将q插入minp的位置

q->next=lastminp->next;

lastminp->next=q;

}

//相邻时执行如下交换算法

**else**

{

q->next=minp->next;

minp->next=q;

lastq->next=minp;

}

}

//偏移 包含了交换后和未交换后的情况

q=minp->next;

lastq=minp;

}

}

**int** main()

{

SqList sql;

LinkNode \*ll;

ll=**new** LinkNode;

ll->next=**NULL**;

}

2.

题目：设线性表a、b，元素值为整型，且递减有序，编写函数，实现下列功能：对采用顺序结构和链式结构2种存储结构，要求在a的空间上构成一个新线性表c，其元素为a和b元素的并集，且表c中的元素值递减有序（互不相同）。

算法思想：依次对比两个线性表的元素。若A>B则A后移B不动，若A=B则AB均后移，若A<B则将B插入到A之前，然后比对的位置均向后偏移。最终若A先结束，则将B的剩余元素全部加入到A的末尾

源代码：

#include<iostream>

**using** **namespace** std;

**class** SqList

{

**public**:

**int** \*v=**new** **int**[1000];

**int** length=0;

};

**class** LinkNode

{

**public**:

**int** data;

LinkNode \*next;

};

**void** CreateSqList(SqList &L,**int** n)

{

**for**(**int** i=0;i<n;i++)

{

cin>>L.v[i];

L.length++;

}

}

**void** CreateLinkList(LinkNode\* &L,**int** n)

{

LinkNode \*r=L;

**for**(**int** i=0;i<n;i++)

{

LinkNode \*p=**new** LinkNode;

cin>>p->data;

p->next=**NULL**;

r->next=p;

r=p;

}

}

**void** PrintSqList(SqList &L)

{

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

{

cout<<L.v[i]<<" ";

}

cout<<endl;

}

**void** PrintLinkList(LinkNode\* &L)

{

LinkNode \*p=L->next;

**while**(p)

{

cout<<p->data<<" ";

p=p->next;

}

cout<<endl;

}

**void** SqListInsert(SqList &L,**int** index,**int** e)

{

**if**(index==L.length)

{

L.v[index]=e;

L.length++;

}

**else**

{

**for**(**int** i=L.length-1;i>=index;i--)

{

L.v[i+1]=L.v[i];

}

L.v[index]=e;

L.length++;

}

}

**void** SqListUnion(SqList &SA,SqList &SB)

{

**int** i=0;

**int** j=0;

//A小于B B插入A的该位置 而后A还是找原来那个数 B找下一个数

**while**(i<=SA.length-1&&j<=SB.length-1)

**if**(SA.v[i]<SB.v[j])

{

SqListInsert(SA, i, SB.v[j]);

i++;//移回原来的数

j++;

**continue**;

}

**else** **if**(SA.v[i]==SB.v[j])

{

i++;

j++;

**continue**;

}

**else**

{

i++;

**continue**;

}

**if**(i==SA.length)

{

**for**(;j<=SB.length-1;j++)

{

SqListInsert(SA, i, SB.v[i]);

i++;

}

}

}

**void** LinkListUnion(LinkNode\* &LA,LinkNode\* &LB)

{

LinkNode \*lastp=LA;

LinkNode \*lastq=LB;

LinkNode \*p=LA->next;

LinkNode \*q=LB->next;

**while**(p&&q)

{

**if**((p->data)<(q->data))

{

lastq->next=q->next;

q->next=p;

lastp->next=q;

//A B的寻找位均向后偏移一个

//此时p仍指向原来的数 只需将lastp向后偏移

lastp=q;

//q指向现在lastq的后面即为向后偏移

q=lastq->next;

}

**else** **if**((p->data)==(q->data))

{

lastp=p;

p=p->next;

lastq=q;

q=q->next;

}

**else**

{

lastp=p;

p=p->next;

}

}

**if**(!p)

{

lastp->next=q;

}

}

**int** main()

{

SqList SA,SB;

LinkNode \*LA=**new** LinkNode;

LA->next=**NULL**;

LinkNode \*LB=**new** LinkNode;

LB->next=**NULL**;

**return** 0;

}

3.

题目：输入正整数n、m（m<n），设有n个人坐成一圈，从第1个人开始循环报数，报到m的人出列，然后再从下一个人开始报数，报到m的人又出列，如此重复，直到所有的人都出列为止。要求用链式结构和顺序结构实现，按出列的先后顺序输出每个人的信息。

算法思想：对顺序表，将已出列的人的数据记为0；对链表，采用循环链表结构较为方便

源代码：

#include<iostream>

**using** **namespace** std;

**class** SqList

{

**public**:

SqList(**int** n)

{

v=**new** **int**[n+1];

**for**(**int** i=0;i<n;i++)

{

v[i]=i+1;

length=n;

}

}

**int** \*v;

**int** length;

};

**class** LinkNode

{

**public**:

**int** data;

LinkNode \*next;

};

//index为被删除值的下标 e用以接受被删除值

**void** DeleteSqListElem(SqList &L,**int** index,**int** &e)

{

**if**(index==L.length-1)

{

e=L.v[index];

L.length--;

}

**else**

{

e=L.v[index];

**for**(**int** i=index+1;i<=L.length-1;i++)

{

L.v[i-1]=L.v[i];

}

L.length--;

}

}

**void** PrintSqList(SqList &L)

{

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

{

cout<<L.v[i]<<" ";

}

cout<<endl;

}

//换一种思路 由于一直换下标太麻烦 不如将拿出来的人的信息赋值成0

**void** SqListCount(SqList &L,**int** m,**int** n)

{

**int** i=0;//遍历下标

**int** count=0;

**int** outnum=0;//用于记录已经出局的人数 当作退出条件

**while**(1)

{

**if**(L.v[i])

{

count++;

**if**(count%m==0)

{

cout<<L.v[i]<<" ";

outnum++;

L.v[i]=0;

}

}

**if**(outnum==n)

{

**break**;

}

i=(i+1)%n;

}

}

**void** CreateLinkList(LinkNode\* &head,LinkNode\* &tail,**int** n)

{

LinkNode \*first=head;//记录第一个节点的位置

**for**(**int** i=1;i<=n-1;i++)

{

LinkNode \*p=**new** LinkNode;

p->data=i+1;

p->next=**NULL**;

head->next=p;

head=p;

}

//尾插结束后构成循环链表 另尾节点指向首元节点对本题来说较为方便

tail=head;

head->next=first;

head=head->next;//令head重新指向第一个元素

}

**void** LinkListCount(LinkNode\* &head,LinkNode\* &tail,**int** m,**int** n)

{

**int** outnum=0;//记录出局人数

**int** count=0;

**while**(1)

{

count++;

**if**(count%m==0)

{

cout<<head->data<<" ";

outnum++;

LinkNode \*p=head;

head=head->next;

tail->next=head;

**delete** p;

**continue**;

}

**if**(outnum==n)

{

**break**;

}

head=head->next;

tail=tail->next;

}

}

**int** main()

{

SqList L(7);

LinkNode \*head=**new** LinkNode;

head->data=1;

head->next=**NULL**;

LinkNode \*tail;

**return** 0;

}

4.

题目：新学期伊始，适逢顿顿书城有购书满x元包邮的活动，小P同学欣然前往准备买些参考书。

一番浏览后，小P初步筛选出n本书加入购物车中，其中第i本（1≤i≤n）的价格为ai元。

考虑到预算有限，在最终付款前小P决定再从购物车中删去几本书（也可以不删），使得剩余图书的价格总和m在满足包邮条件（m≥x）的前提下最小。

试帮助小P计算，最终选购哪些书可以在凑够x元包邮的前提下花费最小？

输入格式：  
从标准输入读入数据。

输入的第一行包含空格分隔的两个正整数n和x，分别表示购物车中图书数量和包邮条件。

接下来输入n行，其中第i行（1≤i≤n）仅包含一个正整数ai，表示购物车中第i本书的价格。输入数据保证n本书的价格总和不小于x。

输出格式：  
输出到标准输出。

仅输出一个正整数，表示在满足包邮条件下的最小花费。

样例1输入：

4 100

20

90

60

60

样例1输出：  
110  
样例1解释：  
购买前两本书（20+90）即可包邮且花费最小。

样例2输入：

3 30

15

40

30

样例2输出：  
30  
样例2解释：  
仅购买第三本书恰好可以满足包邮条件。

样例3输入：

2 90

50

50

样例3输出：  
100  
样例3解释：  
必须全部购买才能包邮。

算法思想：采用DFS算法遍历出从第一件商品开始所有可能的加和情况，同时运用前缀和，将遍历过程中出现的所有之后全加也不可能超过包邮价的情况删除。从而剪枝降低时间复杂度

源代码：

//运用dfs剪枝和前缀和思想解决问题

//如果该节点及其后所有节点的和与当前和加上后仍小于x则剪枝

#include<iostream>

#include<vector>

#include<algorithm>

**using** **namespace** std;

**void** dfs(**int** k,**int** sum,**int** &ans,**int**\* first\_sum\_price,**int**\* price,**int** &x,**int** &n)

{

**if**(sum+first\_sum\_price[n]-first\_sum\_price[k-1]<x)

{

**return**;

}

**if**(sum>=x)

{

ans=min(ans,sum);

}

**if**(k>n)

{

**return**;

}

dfs(k+1,sum+price[k],ans,first\_sum\_price,price,x,n);//加上这个节点的可能性

dfs(k+1,sum,ans,first\_sum\_price,price,x,n);//不加上这个节点的可能性

}

**int** main()

{

**int** n,x;

cin>>n>>x;

**int** price[n+1];

**int** first\_sum\_price[n+1];//前缀和

first\_sum\_price[0]=0;

**for**(**int** i=1;i<=n;i++)

{

cin>>price[i];

first\_sum\_price[i]=first\_sum\_price[i-1]+price[i];

}

**int** ans=first\_sum\_price[n];//先把答案赋值成可能的最大值

dfs(1, 0, ans, first\_sum\_price, price, x, n);

cout<<ans<<endl;

}

5．

题目：小H和小W来到了一条街上，两人分开买菜，他们买菜的过程可以描述为，去店里买一些菜然后去旁边的一个广场把菜装上车，两人都要买n种菜，所以也都要装n次车。具体的，对于小H来说有n个不相交的时间段[a1,b1],[a2,b2]…[an,bn]在装车，对于小W来说有n个不相交的时间段[c1,d1],[c2,d2]…[cn,dn]在装车。其中，一个时间段[s, t]表示的是从时刻s到时刻t这段时间，时长为t-s。  
由于他们是好朋友，他们都在广场上装车的时候会聊天，他们想知道他们可以聊多长时间。

算法思想：遍历二者时间段，并创建两个另外的数组记录是否在某时间段内装车的状态量，然后比对状态量数组，两个同时为1时计数器自增

源代码：

#include<iostream>

#include<vector>

#include<algorithm>

**using** **namespace** std;

**class** Person

{

**public**:

vector<pair<**int**,**int**>> time;//用于记录每一段工作时间

vector<**bool**> t;//用于记录表示是否在装车的状态量

};

**void** PutOnTime(Person &p,**int** maxt)

{

p.t.resize(maxt,0);

**for**(vector<pair<**int**,**int**>>::iterator timeit=p.time.begin();timeit!=p.time.end();timeit++)

{

**for**(**int** i=(\*timeit).first;i<=(\*timeit).second-1;i++)

{

p.t[i]=1;

}

}

}

**int** main()

{

Person H;

Person W;

**int** n;//时间段的数量

cin>>n;

**int** maxh=0;

**int** maxw=0;

**int** maxt=0;

**int** sumtime=0;//用以记录总的聊天时长

**for**(**int** i=0;i<n;i++)

{

**int** a,b;

cin>>a>>b;

**if**(b>maxh)

{

maxh=b;

}

pair<**int**,**int**>p=make\_pair(a,b);

H.time.push\_back(p);

}

**for**(**int** i=0;i<n;i++)

{

**int** a,b;

cin>>a>>b;

**if**(b>maxw)

{

maxw=b;

}

pair<**int**,**int**>p=make\_pair(a,b);

W.time.push\_back(p);

}

maxt=max(maxh,maxw);

PutOnTime(H,maxt);

PutOnTime(W,maxt);

vector<**bool**>::iterator hit=H.t.begin();

hit++;

vector<**bool**>::iterator wit=W.t.begin();

wit++;

**while**(hit!=H.t.end()&&wit!=W.t.end())

{

**if**((\*hit)==1&&(\*wit)==1)

{

sumtime++;

}

hit++;

wit++;

}

cout<<sumtime;

**return** 0;

}

1. 体会
2. dfs可以用来得到一堆元素任意数量的组合情况
3. 灵活运用前缀和可以大幅减少比对数量的时间复杂度
4. 要灵活运用状态量数组来避免一些麻烦的操作