# 2022-2023王道计算机练习：上机+笔试

## 1.typedef使用

<https://blog.csdn.net/weixin_46263778/article/details/126150684>

#include <stdio.h>

typedef struct student

{

int num;

char name[30];

char sex[30];

}stu, \* Studet;

//定义一个结构，里面有三个主体，在main中进行结构赋值，之后输出

typedef int INT;

int main()

{

stu s = { 180,"张珊","男" };

Studet p;

int n = 100;

p = &s;

printf("%d,身高:%d\n", n, p->num);

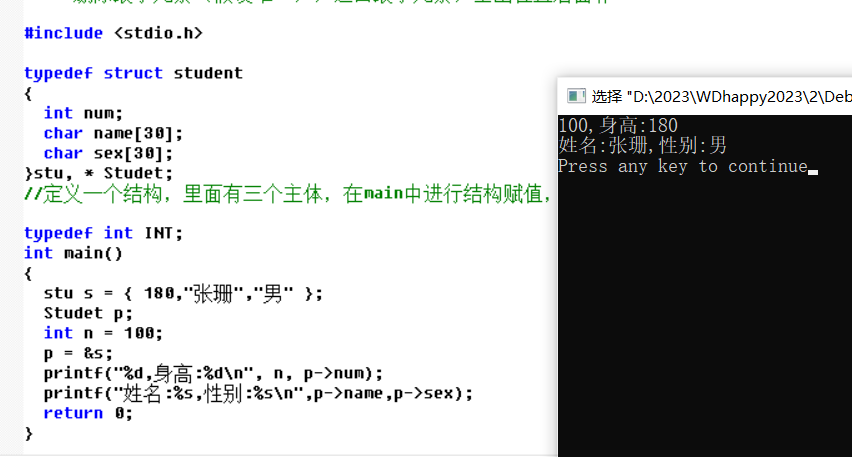
printf("姓名:%s,性别:%s\n",p->name,p->sex);

return 0;

}

2.

**\*是指针运算,&是取地址**



王道计算机练习：

## 2.顺序表的一系列操作

插入删除查找：

顺序表插入（静态）：

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#define MaxSize 50

#define InitSize 100

typedef int ElemType;

typedef struct {

ElemType data[MaxSize]; //假定顺序表的元素类型ElemType

int length; //顺序表的当前长度

}SqList; //顺序表的类型定义

//静态分配。动态分配

typedef struct {

ElemType \*data; //指定动态分配数组的指针

int length; //数组的最大容量和当前个数

}SeqList; //动态分配数组顺序表的类型定义

//c:malloc,c++:new

//插入操作 i位置(1<=i<L.length+1)

bool ListInsert(SqList &L,int i,ElemType e){

if(i<1||i>L.length+1) //判断i的插入位置是否合法

return false;

if(L.length>=MaxSize) //判断存储空间是否已满。满则不插

return false;

for(int j=L.length;j>=i;j--) //开始是length-1~i,插入数据是i(j)~length-1+1

L.data[j]=L.data[j-1]; //i

L.data[i-1]=e;//i位置插入

L.length++;

return true;

}

int main()

{

SqList L;

int i=0;

//ElemType s[]={1,2,3,4,5};

//L.data[0]=1; //r C2106: '=' : left operand must be l-value

L.length=0;

for(i=0;i<4;i++){

L.data[i]=i;//赋初值

L.length++;

}

//L.data=(ElemType\*)malloc(ElemType)\*InitSize;//动态分配

int flag;

flag =ListInsert(L,2,6);//i代表插入位置是从1开始，不要和数组下标混淆

//printf("%d\n\n",flag);

//printf("%d\n\n",L.length);

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

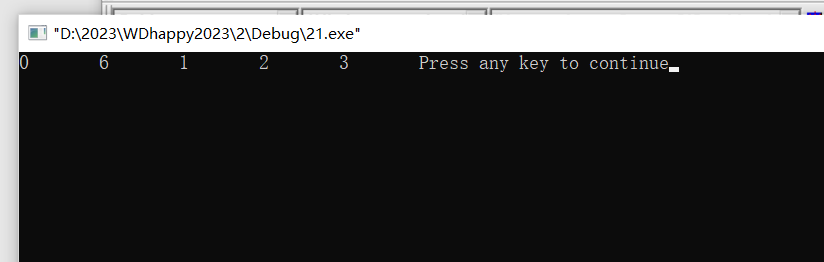
printf("%d\t",L.data[i]);

}

return 0;

}

在第二个位置插入6（位置2下标的1，在1 的位置插入，1之后元素向后移动）



对上面补充：（删除，查找第一个为e元素操作）

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

#define MaxSize 50

#define InitSize 100

typedef int ElemType;

typedef struct {

ElemType data[MaxSize]; //假定顺序表的元素类型ElemType

int length; //顺序表的当前长度

}SqList; //顺序表的类型定义

//静态分配。动态分配

typedef struct {

ElemType \*data; //指定动态分配数组的指针

int length; //数组的最大容量和当前个数

}SeqList; //动态分配数组顺序表的类型定义

//c:malloc,c++:new

//11 插入操作 i位置(1<=i<L.length+1)

bool ListInsert(SqList &L,int i,ElemType e){

if(i<1||i>L.length+1) //判断i的插入位置是否合法

return false;

if(L.length>=MaxSize) //判断存储空间是否已满。满则不插

return false;

for(int j=L.length;j>=i;j--) //开始是length-1~i,插入数据是i(j)~length-1+1

L.data[j]=L.data[j-1]; //i后元素后移 i==>

L.data[i-1]=e;//i位置插入

L.length++;

return true;

}

//22 删除操作 i位置(1<=i<L.length)（范围和插入不同）

bool ListDelete(SqList &L,int i,ElemType &e){

if(i<1||i>L.length)

return false;

e=L.data[i-1];//i位置删除，下标i-1+1后 即i后元素前移 i<==

for(int j=i;j<L.length;j++) //开始是length-1~i,插入数据是i(j)~length-1+1

L.data[j-1]=L.data[j]; //i

L.length--;

return true;

}

//顺序查找

int LocalElem(SqList L,ElemType e){

int i;

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

if(L.data[i]==e)//从前=》后查找第一个元素等于e的值

return i+1;

return 0;

}

int main()

{

SqList L;

int i=0,e=0;

int flag=0;

L.length=0;

for(i=0;i<10;i++){

L.data[i]=i;//赋初值

L.length++; //length不能丢

}

ListInsert(L,2,121);//i代表插入位置是从1开始，不要和数组下标混淆

ListInsert(L,3,2333);

ListDelete(L,3,e);//一次性操作

int local=LocalElem(L,5);//查找元素为5的位置（不是下标）

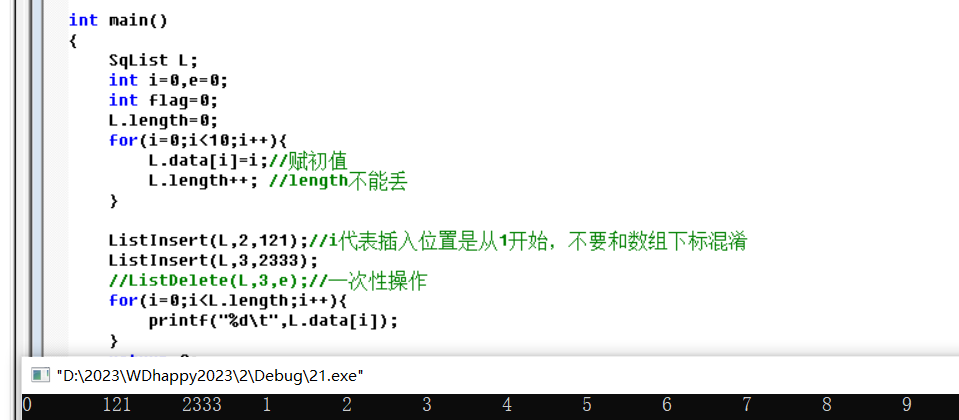
printf("%d\n",local);

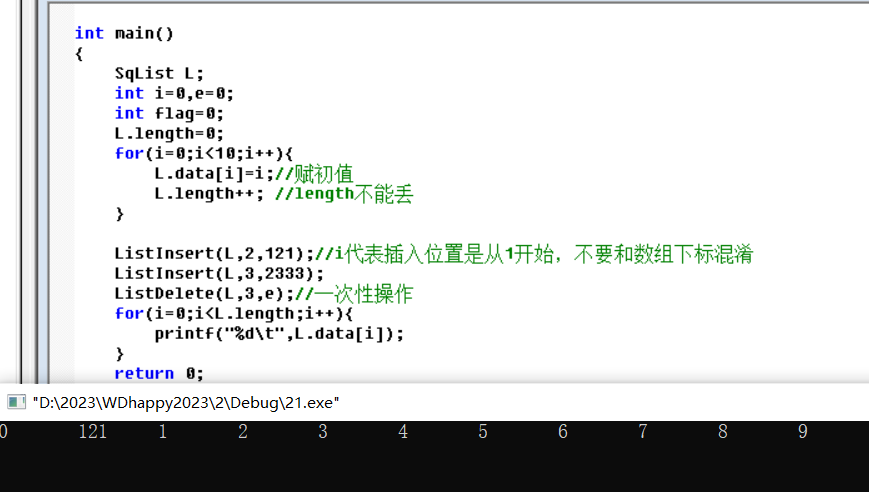
for(i=0;i<L.length;i++){

printf("%d\t",L.data[i]);

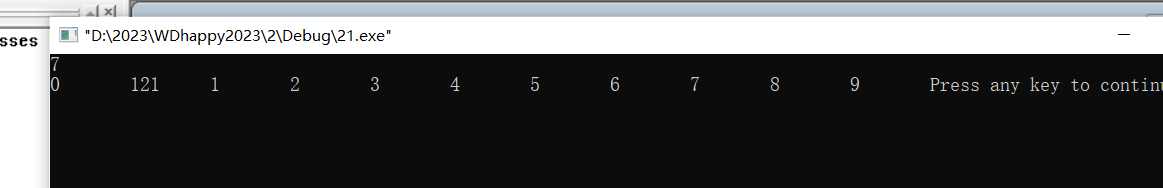
}

return 0;

}



查找第一个元素等于5的值：



## 3.顺序表练习题

/\*王道计算机\*/

//

//

/\*\*/

//

/\*\*/

//

/\*1.从数据库删除最小值元素（假设唯一）并返回被删除元素，空出位置由后位补。若顺序表为空，显示错误信息并退出\*/

//算法思想：搜索整个顺序表，查找最小值元素并记住其位置，搜索结束后用最后一个元素填补空出的原最小元素的位置

//代码：

bool Del\_Min(sqList &L,ElemType &value){

if(L.length==0) //判断长度，长度为0即空， 返回false

return false;

value=L.data[0]; //假定第一个元素最小，之后比较进行交换

int pos=0; //用于标记删除的元素位置i

for(int i=1;i<L.length;i++){

if(L.data[i]<value){ //查找比第一个元素小的，不断进行交换

value=L.data[i]; //存在更小的进行交换

pos=i; //记住位置

}

}

L.data[pos]=L.data[L.length-1];//将位置由最后一个元素补充，因为下标是0 开始，所以最后一个元素的下标是length-1

L.length--;//删除最小元素（唯一的意思是假定一个），相应长度需要减少

return true;

}

//2.设置一个高效算法，将顺序表L的所有元素逆置，要求算法的空间复杂度为1

/\*算法思想：扫描顺序表L的前半部分，对于元素L.data[i](0 <= i < L.length/2),将其与后半部分的对应元素L.data[L.length-i-1]进行交换\*/

//代码：

void Reserve(Sqlist &L){

Elemtype temp;

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

temp=L.data[i]; //把左元素赋值给temp，防止后面赋值覆盖前面数据导致丢失

L.data[i]=L.data[L.length-i-1]; //左=右

L.data[L.length-i-1]=temp; //右=temp(先前保存的左元素)

}

}

//3.对于长度为n的顺序表L，编写时间复杂度0（n），空间复杂度0（1）算法，删除表中所有值为x的值

//算法思想：k记录不等于x的值，修改L长度

//解法一：

void del\_x\_1(Sqlist &L,Elemtype x){

int k=0,i;

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

if(L.data[i]!=x){

L.data[k]=L.data[i]; //遍历寻找不等于x的值返回给原数组，下标变化

k++;

}

L.length=k;

}

}

//解法二：

void del\_x\_2(Sqlist &L,Elemtype x){

int k=0,i=0;

while(i<L.length){

if(L.data[i]==x)

k++;

else

L.data[i-k]=L.data[i]; //前移k个位置 7-6类似

i++;

}

L.length=L.length-k;

}

//4.(有序顺序表)删除s-t值之间元素（连续）(s<t)，若不合理或顺序表为空，显示错误信息

void Del\_s\_t2(Sqlist &L,Elemtype s,Elemtype t){

int i,j;

if(s>=t||L.length=0)

return false;

for(i=0;i<L.length&&L.data[i]<s;i++);//寻找第一个大于s的元素 下标s<i

if(i>=L.length)

return false;

for(j=i;i<L.length&&L.data[j]<=t;j++);////寻找第一个大于t的元素 下标j>t

for(;j<L.length;j++)

L.data[i]=L.data[j]; //前移，填补被删除元素(开始：s-t，前移i《j) s 【i(值大于等于s的第一个元素)……t】 j

}

//5.(顺序表)删除s-t值之间元素（连续）(s<t)(包含s和t)，若不合理或顺序表为空，显示错误信息

void Del\_s\_t(Sqlist &L,Elemtype s,Elemtype t){

int i,k=0;

if(s>=t||L.length=0)

return false;

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

if(L.data[i]>=s&&L.data[i]<=t)//寻找s-t元素（包含便捷）

k++;

else

L.data[i-k]=L.data[i]; //s(<=i)……t(<=)j

}

L.length-=k;

return true;

}

//6 有序顺序表(值相同的元素连续)中删除重复元素,剩余均不同(可以调试看)

bool Delete\_Same(SqList &L){

int i,j;

if(L.length==0)

return false;

for(i=0,j=1;j<L.length;j++)

if(L.data[i]!=L.data[j])//第一个元素是不变，后一个和前一个对比，如果不同，插入进去i++，j++向后判断，if相同，i不变，j向后找不同插入进去

L.data[++i]=L.data[j];//相当于把相同的第二个元素覆盖 -----(简单来说：L.data[0]确定，后面比较不同L.data[j]插入)

L.length=i+1;

return true;

}

//如果本题是无序表 o（n），用散列表

//7 两有序顺序表合并新有序表 c=a并b(不同的)（重要）,返回新

bool Merge(SeqList A,SeqList B,SeqList &C){

int i,j=0,k=0;

if(A.length+B.length>C.MaxSize)

return false;

while(i<A.length&&j<B.length){

if(A.data[i]<B.data[j])

C.data[k++]=A.data[i++];//先把A.data[i]插入，然后i++再进行后面比较

else

C.data[k++]=B.data[j++];

}

while(i<A.length) //最后全部比较完之后还会剩余一个没有匹配完.进行判断

C.data[k++]=A.data[i++];

while(j<A.length)

C.data[k++]=B.data[j++];

C.length=k;

return true;

}

//8 一维数组存放两个线性表，a（1-m），b（1-n）下标，两个顺序表位置互换，b放在a之前

//本题王道：逆置

typedef int DataType;

void Reserve(DataType A[],int left,int right,int arraySize){

//a1 a2……an,b1,b2……bn

//one:bn …… b2,b1,an……a2,a1（全部逆置）

//two:b1,b2……bn,a1,a2……an

Elemtype temp;

if(left>right||right>=arraySize)

int mid=(right+left)/2;

for(int i=0;i<mid-left;i++){

DataType temp=A[left+i]; //把左元素赋值给temp，防止后面赋值覆盖前面数据导致丢失

A[left+i]=L.data[right-i]; //左=右

A[right-i]=temp; //右=temp(先前保存的左元素)

}

}

void Exchange(DataType A[],int m,int n,int arraySize){

Reserve(A,0,m+n-1,arraySize);//全部

Reserve(A,0,n-1,arraySize);//前半部分

Reserve(A,n,m+n-1,arraySize);//后半部分

}

//9 线性表a1-an,递增有序且顺序存储于计算机，最少时间查找数值为x值，找到与后继元素互换，找不到插入到表中，表仍然递增有序

//可以顺序、折半=》折半时间少

void SearchExchangeInsert(ElemType A[],ElemType x){

int low=0,high=n-1,mid;

while(low<high){ //找到A[mid]==x 》mid

mid=(low+high)/2;

if(A[mid]==x)

break;

else if(A[mid]<x)

low=mid+1;

else

high=mid+1;

}

if(A[mid]==x&&mid!=n-1){ //mid!=high 假设最后一个元素等于x，不存在后继元素交换,提前剔除这个情况

t=A[mid];

A[mid]=A[mid+1];

A[mid+1]=t;

}

if(low>high){ //查找失败，这个x插入到表中仍然递增有序

for(i=n-1;i>high;i--) //n-1是查找失败的位置,n-1后位置后移，插入这个x（当i=high结束）

A[i+1]=A[i];

A[i+1]=x;

}

}

//10.一维数组R，R中循环左移p(0<p<n) x0 x1……x(n-1) =》 xp x(p+1) …x(n-1) x1 x0 … x(p-1)

/\*\*\*

(1)给出算法的基本设计思想

将这个问题视为数组ab》ba(a前p个元素，b代表余下的n-p个元素,)

a逆置a^-1\*b b逆置a^-1\*b^-1 最后真个a^-1\*b^-1逆置 (a^-1\*b^-1)^-1=ba

设Reverse函数执行逆置操作 对abcdefgh左循环移动3（p=3）位置过程如下：

Reverse(0,p-1);cba defgh

Reverse(p,n-1);cba hgfed

Reverse(0,n-1);defghabc

(2)根据设计思想，代码，关键之处给注释

(3)时空复杂度

Revesve时间复杂度分别是O(p/2) O((n-p)/2) O(n/2)=》时间复杂度：O(n) 空间复杂度O(1)

\*\*/

void Revesve(int R[],int from,int to){

int i,temp;

for(int i=0;i<(to-from+1)/2;i++){

temp=L.data[from+i]; //把左元素赋值给temp，防止后面赋值覆盖前面数据导致丢失

L.data[from+i]=L.data[to-i]; //左=右 (可以理解为双向奔赴)

L.data[to-i]=temp; //右=temp(先前保存的左元素)

}

}

void Converse(int R[],int n,int p){

Reserve(R,0,p-1);//全部

Reserve(R,p,0,n-1);//前半部分

Reserve(R,0,n-1);//后半部分

}

//11 L升序S。处在L/2(向上取整)的数称为S的中位数 S1={11,13,15,17,19},中位数15，两个序列的中位数是包含所有元素的升序序列的中位数。

//例S2={2,4,6,8,20} S1和S2的中位数是11 =>两个等长升序序列A B，找出A,B的中位数

/\*\*

(1)算法思想

(2)代码及注释

(3)时空复杂度 O(log2n) 空间：O(1)

\*/

//算法基本设计思想：分别求A,B的中位数，设为a，b。过程如下：

/\*\*

one:a=b a或b即为所求中位数

two:a<b 设计A中较小和B中较大的一半，要求两次舍弃的长度相同

three:a>b 舍弃A中较大和B中较小的一半，要求两次舍弃的长度相同

在保留的两个升序序列中，重复1.2.3操作，直到两个序列中均只含一个元素为止，较小的元素即为所求中位数

\*/

int M\_Search(int A[],int B[],int n){

int s1=0,d1=n-1,m1,s2=0,d2=n-1,m2;

//分别表示A,B的首位数，末尾数，中位数 //s1 m1 d1,s2 m2 d2

while(s1!=d1||s2!=d2){

m1=(s1+d1)/2;//A中位数下标

m2=(s2+d2)/2;//B中位数下标

if(A[m1]==B[m2]) //if A B中位数相同，任意一个都可

return A[m1];

if(A[m1]<B[m2]){ //if A < B中位数,舍弃A中较小的，B 中较大的

if((s1+d1)%2==0){

s1=m1; //A的首位数=A的末尾数

d2=m2; //B的末尾数=B的中位数数 //m1 d1,s2 m2

}

else{

s1=m1+1;

d2=m2; //m1+1 d1,s2 m2

}

}

else{ //舍弃A中较大的和B中较小的

if((s1+d1)%2==0){

d1=m1;

s2=m2; //s1 d1 m2 d2

}

}

}

return A[s1]<B[s2]?A[s1]:B[s2]; //返回两者中的较小值 //s1,s2始终保存的都是中位数

}

//12 整数序列A=(a0,a1,……a^n-1),0<= a^i <n(0<=i<n)

//若存在ap1=ap2=……apm=x且m>n/2，（0<=pk<n,a<=k<=n），称x为A的主元素。

//例：A=(0,5,5,3,5,7,5,5) 5为主元素（5> 8/2） 又如A=(0,5,5,3,5,1,5,7) (4< 8/2)没有主元素。

/\*\*

假设A中的n个元素保存在一个一维数组中，找出A的主元素，存在输出，不存在输出-1

（1）给出算法的基本设计思想

（2）代码及注释

（3）时空复杂度 O(n) 空间：O(1)

\*/

/\*\*

算法思想:从前到后扫描数组元素，标记出一个可能成为主元素的值Num，确认其是否为主元素.分两步

(1)选取候选的主元素。从前到后扫描每个整数，将第一个遇到的整数Num存到c中，记录此Num出现次数，再次出现+1,否则减1；将下个遇到整数保存到c中，

重新计数为1，开始新一轮计数，从当前位置开始重复上述过程，直到扫描完全部数组元素

(2)判断c中元素是否是真正的主元素，再次扫描该数组，统计c中元素出现的次数，若大于n/2，则为主元素；否则不存在

\*/

int Majority(int A[],int n){

int i,c,count-1;//c用来保存主元素，count用来计数

c=A[0];

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

if(A[i]==c)

count++;

else

if(count>0) //处理不是候选主元素的情况

count--;

else{

c=A[i]; //更换候选主元素，重新计数

count=1;

}

if(count>0)

for(i=count=0;i<n;i++)

if(A[i]==c)

count++; //统计候选主元素的实际出现次数

if(count>n/2)

return c;//确认主元素

else

return -1;

}

//13 含n（n>=1）个整数的数组，找出数组中未出现的最小正整数，例如数组{-5,3,2,3}中未出现最小正整数1，数组{1,2,3}未出现最小正整数4

/\*\*

（1）算法的基本设计思想

要求在时间上尽可能高效，空间换时间。分配一个用于标记的数组B[n],用来记录A中是否出现了1~n的正整数，B[0]对应正整数1，B[n-1]对应正整数n

初始化B中全部为0.

由于A中含有n个整数，可能返回的值为1~n+1 A中n个数刚好为1~n时返回n+1 （最小正整数为n+1）

当数组A中出现小于等于0或大于n的值会导致1~n中出现空余位置，返回结果在1~n之间 （最小正整数在1~n之间）

当数组A中出现了小于等于0的值，不采取操作 （小于0时不存在此范围内的最小正整数）

（2）代码和注释

（3）时空复杂度 O(n) 空间：O(n)

\*/

int findMissMin(int A[],int n){

int i,\*B; //标记数组

B=(int \*)malloc(sizeof(int)\*n);//分配空间

memset(B,0,sizeof(int)\*n);//赋初值为0

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

if(A[i]>0&&A[i]<=n) //若值在0-n范围内，标记数组B

B[A[i]-1]=1;

for(i=0;i<n;i++)//扫描数组B，找到目标值

if(B[i]==0) //初值都是0，1是被标记的=》找到等于0的值即为所求下标，例B[0]=0 》0+1=1 二、B[0]=B[1]=B[2]=1》B[3]=0;》i+1=4

break;

return i+1;//返回结果

}

//14 定义三元组(a,b,c)(a,b,c均为正数)的距离D=|a-b|+|b-c|+|c-a| 给定三个非空整数集合S1,S2,S3.按照升序分别存储在3个数组中。

//计算输出所有可能的三元组(a,b,c)(a,b,c分别属于S1,S2,S3)中的最小距离。

/\*\*

例如：S1={-1,0,9}，S2={-25,-10,10,11},S3={2,9,17,30,41}，则最小距离为2，相应的三元组{9,10,9}

\*/

/\*\*

(1)基本设计思想

结论：

one：a=b=c时，距离最小

two：其余情况。假设a<=b<=c 化数轴 a\_\_(L1)\_\_\_b\_\_\_(L2)\_\_\_c a\_\_\_(L3)\_\_\_c

L1=|a-b| L2=|b-c| L3=|c-a| D=|a-b|+|b-c|+|c-a|=L1+L2+L3=2L3

由上可知，决定D大小的关键是ac之间的距离

(2)代码和注释

(3)时空复杂度 O(n),O(1)

\*/

/\*\*

算法的基本思想：

1。使用Dmin记录所有已处理的三元组的最小距离，初值为一个足够大的整数。

2。集合S1,S2,S3分别保存在数组A,B,C中。数组的下标变量i=j=k=0，当i<|S1|，j<|S2|，k<|S3|时，（|S|表示集合S中的元素个数），循环执行下面操作

a.计算（A[i],B[j],C[k]）的距离D（计算D）

b.若D<Dmin,则Dmin=D（更新D）

c.将A[i],B[j],C[k]中的最小值的下标+1,（对照分析：最小值为a，最大值为c，c不变更新a，找到最小的距离D）

3。输出Dmin

\*/

#define INT\_MAX ox7fffffff //整个整数 0x7FFFFFFF 的二进制表示就是除了首位是 0，其余都是1,这是最大的整型数 int（因为第一位是符号位，0 表示他是正数）

int abs\_(int a){//计算绝对值

if(a<0)

return -a;

else

return a;

}

bool xls\_min(int a,int b,int c){//a是否是三个数中最小值

if(a<=b&&a<=c)

return true;

return false;

}

int findMinofTrip(int A[],int n,int B[],int m,int c[],int p){

//D\_min记录三元组中的最小距离，初赋值为INT\_MAX

int i=0,j=0,k=0,D\_min=INT\_MAX,D;

while(i<n&&j<m&&k<p&&D\_min>0){

D=abs\_(A[i]-B[j])+abs\_(B[j]-C[k])+abs\_(C[k]-A[i]);//计算D

if(D<D\_min)

D\_min=D; //获取更小值，更新D

if(xls\_min(A[i],B[j],C[k]))

i++; //更新a

else if(xls\_min(B[j],C[k],A[i]))

j++;

else

k++;

}

return D\_min;

}

//单链表：

/\*\*

\*/

typedef struct LNode{

ElemType data;

struct LNode \*next; //指针域

}LNode,\*LinkLst;

//头插法

LinkList List\_HeadInsert(LinkList &L){

LNode \*s,int x;

L=(LinkList)malloc(sizeof(LNode)); //创建头结点

L->next=NULL; //初始为空链表

scanf("%d",&x);

while(x!=9999){ //9999结束

s=(LNode\*)malloc(sizeof(LNode));//创建新结点

s->data=x;

s->next=L->next; // L s(x)

L->next=s; //先把新插入结点的（数据、指针域）处理好，再去处理L和s的关系，防止有歧义

scanf("%d",&x);

}

return L;

}

//尾插法建立单链表

LinkList List\_TailInsert(LinkList &L){ // L(r) s(值为x)（新插入的s变成新尾结点）

int x;

L=(LinkList)malloc(sizeof(LNode)); //创建头结点

LNode \*s,\*r=L;//r为表尾指针

scanf("%d",&x);

while(x!=9999){ //9999结束

s=(LNode\*)malloc(sizeof(LNode));//创建新结点

s->data=x;

r->next=s;

r=s; //r指向新的表尾结点

scanf("%d",&x);

}

r->next=NULL;//尾指针置空

return L;

}

//按序号查找结点值

LNode \*GetElem(LinkList L,int i){

int j=1;//初始为1

LNode \*p=L->next; //第一个结点指针赋给p

if(i==0)

return L; //返回头结点

if(i<1)

return NULL;

while(p&&j<i){ //寻找结点在第一个值之后，遍历后边数据

p=p->next;

j++;

}

return p; //返回寻找到的第i个结点

}

//按值查找表结点

LNode \*LocateElem(LinkList L,ElemType e){

LNode \*p=L->next; //从第一个结点开始查找

while(p!=NULL&&p->data!=e)

p=p->next;

return p;

}

//插入结点操作 时间复杂度：O(n)

//……a（p） s(值为x)(位置i) b……

p=GetElem(L,i-1); //查找插入位置的前驱结点

s->next=p->next;

p->next=s;

//对某结点进行前插(转换为后插操作) 时间复杂度：O(1)

// s a(p) b 》a(p) s b (把p->data和s->data进行交换)

s->next=p->next;

p->next=s;

temp=p->data;

p->data=s->data;

s->data=temp;

//删除第i个结点

// (p)a (q)b(i) c 删除b

pGetElem(L,i-1);

q=p->next; //q指向被删除结点

p->next=q->next; //令\*q结点从链中“断开”

freeq(q); //释放结点的存储空间

/\*上述做法是从链表头结点顺序查找前驱结点，然后执行删除操作，时间复杂度O(n)

删除结点用后继结点操作实现，实质是将后继结点值赋予自身，然后删除后继结点，时间复杂度O(1).下例：删除\*p(转换成p值赋给q，q删除)

\*/

q=p->next;

p->data=p->next->data;

p->next=q->next;

freeq(q);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*双链表\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//结点类型：

typedef struct DNode{ //双链表结点类型

ElemType data;

struct DNode \*prior,\*next;

}DNode,\*DLinklist;

//双链表插入结点\*s

//p(a值) s(x值) c

s->next=p->next; //s后继 s>c 1

p->next->prior=s; //p后继前驱 s<c 2

s->prior=p; //s前驱 p<s

p->next=s; //p后继 p>s 4

//1,2必须在4之前

//删除

//p(a) q(b) c

p->next=q->next; //p>q

q->next->prior=p; //p<q

free(q);

//静态链表

#define MaxSize 50

typedef struct{

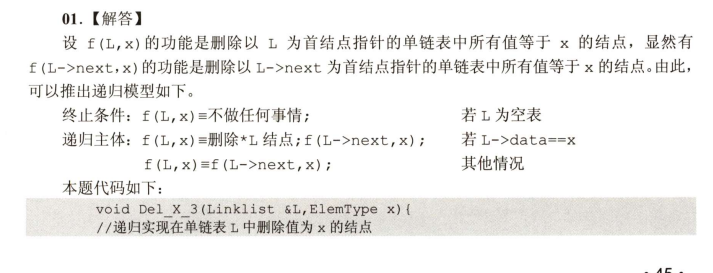
ElemType data;

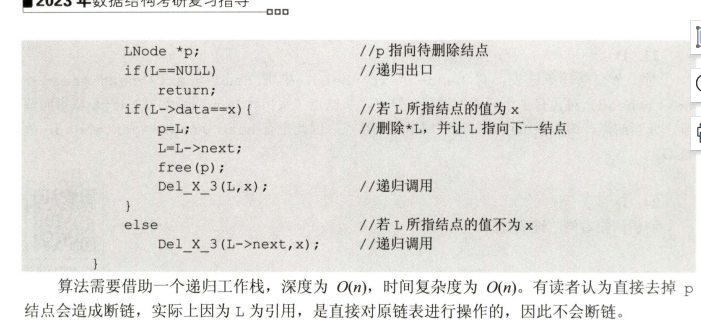
int next;//下个元素的数组下标

}SLinkList(MaxSize);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*王道第二章\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//01设计一个递归算法，删除不带头结点的单链表L中所有值为x的结点





//02 在带头结点的单链表L中，删除所有值为x的结点，并释放其空间，假设值为x的不唯一

//03 L带头结点的单链表，实现逆序输出每个结点值

//04 带头结点单链表L删除一个最小值结点高效算法（假设最小值是唯一的）

//05 带头结点单链表L就地逆置，“就地”辅助空间复杂度为O(1)

//06 带头结点L单链表，实现元素递增有序

//07 带头结点单链表所有元素数据值无序，删除表中介于给定两个值（作为函数参数给）之间的元素（若存在）

//08 给定两个单链表，找出两个链表的公共结点

//09 带头结点的单链表，head头指针，结点结构为（data，next），data为整型元素，next为指针，递增次序输出单链表中各结点的数据元素并

//释放结点所占的存储空间（不允许使用数组作为辅助空间）

//10 带头结点的单链表A分解为两个带头结点的单链表A,B，使得A表中含有原表中序号为奇数的元素，B表中为含偶数序号，且保持相对顺序不变

//11 C={a1,b1,a2,……an,bn}为线性表，采用带头结点的hc单链表存放，拆分为两个线性表，就地算法，使得A={a1,a2……an},B={b1,b2……bn}

//12 在递增有序的线性表中，有数值相同元素存在。存储方式为单链表，去掉数值相同的元素，使得表中无重复的元素

//例如{7,10,10,21,30,42,42,42,51,70}将变为{7,10,21,30,42,51,70}

//13 两个按元素递增次序排列的线性表，均以单链表形式存储。合并这两个为一个按元素递减次序排列的单链表，

//要求利用原来两个单链表的结点存放归并后的单链表

//14 设A,B是两个单链表（带头结点），其中元素递增有序，设计一个算法从A和B中的公共元素产生链表C，要求不破坏A,B结点

//15 A,B表示两个集合，其元素递增偶爱列，A和B的交集，存放到A集合中

//16 A=a1,a2……am，B=b1,b2……bn已经存入到两个单链表中，判断B是否是A的连续子序列

//17 判断带头结点的循环链表是否对称

//18 两个循环单链表，链表头指针分别为h1，h2，编写一个函数将链表h2链接到链表h1之后，要求链接后的链表仍然保持循环链表形式

//19 带头结点的循环单链表 结点值均为正整数，反复找出单链表中结点值最小的结点并输出，然后将该结点从中删除，知道单链表空为止，删除头结点

//20 设头指针为L的带表头的非循环双向链表，其每个结点中除有pred（前驱指针），data，next外，还有一个访问频度域freq。

//启用链表前，其值均初始化为0，每当在链表中进行一次Locate(L,x),令x结点的freq+1，并使此链表按访问频度递减顺序排列。

//最近访问的结点排在频度相同的结点之前，以便使频繁访问的结点总是靠近表头。返回找到的结点的地址，类型为指针型

//21 单链表有环，单链表最后一个结点的指针（通常为空）指向链表中的某个结点，判断单链表是否存在环

/\*\*

(1)算法基本思想

(2)代码、注释

(3)时空复杂度

\*/

//22 带头结点的单链表 data|link

//只有头指针list，不改变链表前提下，寻找链表中倒数第k个位置的结点（k为整数）。查找成功，输出data值，并返回1；否则返回0

/\*\*

(1)算法基本思想

(2)代码、注释

(3)时空复杂度

\*/

//23 带头结点的单链表保存单词，当两个单词有相同的后缀，可共享共同的后缀存储空间，例如:loading being (王道书p41)

//str1和str2分别指向两个单词所有单链表的头结点，结点结构 data|next 找到两个链表共同后缀的起始位置(例中的i)

/\*\*

(1)算法基本思想

(2)代码、注释

(3)时空复杂度

\*/

//24 单链表保存m个整数，结点结构[data][link],且|data|<=n(n为正整数)。对于链表中data绝对值相等的结点，仅保留第一次出现的结点而删除其他

//绝对值相等的结点。例如head如下：head->21->-15->-15->-7->15 删除之后：head->21->-15->-7

/\*\*

(1)算法基本思想

(2)单链表结点的数据类型定义

(3)代码、注释

(4)时空复杂度

\*/

//25 线性表L={a1,a2,a3……an}采用带头结点的单链表保存，定义：

typedef struct node{

int data;

struct node \*next;

}NODE;

/\*\*

要求空间复杂度O(1)且时间上尽可能高效的算法，重新排列L中的各结点，得到L'={a1,an,a2,a^n-1……}

\*/

/\*\*

(1)算法基本思想

(2)单链表结点的数据类型定义

(3)代码、注释

(4)时空复杂度

\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*3 栈，队列\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*4 串\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*5 树与二叉树\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//p135王道

// 05 已知一二叉树顺序结构存储，求编号分别为i和j的两个结点的最近的公共祖先结点的值

//非空二叉树的叶子结点个数=度为2节点数+1 => n0=n2+1(p131王道)

//证明:

/\*\*

设度为0,1,2的结点个数分别为n0,n1,n2,结点总数n=n0+n1+n2

再看二叉树的分支数，除根结点外，其余结点都有一个分支进入，设B为分支总数，则n=B+1,由于这些分支是由度为1或2结点射出的，所以由B=n1+2n2

于是得n0+n1+n2=n1+2n2+1,则n0=n2+1

\*/

//1 先序遍历（p139）

//2 后序遍历

//3 中序遍历

//4 递归和非递归

//5 层次遍历

//p149