数据结构作业一：

**1454.顺序表的插入**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

先读入数据建立顺序表va，表中的数据元素递增有序，再读入一个整数x，将它插入到顺序表中的适当位置，并保持顺序表的有序性。

输入

先输入一个小于100的正整数n，紧接着读入n个从小到大的正整数，建立顺序表，最后再读入一个正整数x，把它插入到顺序表中的适当位置，并保持顺序表的有序性。

输出

按顺序输出整个顺序表，输出的每个元素占一行。

输入样例

4  
2 100 3000 3890  
200

输出样例

2  
100  
200  
3000  
3890

提示

输出语句为：printf("%d\n");

**1455.顺序表的逆置**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

读入一个顺序表，实现顺序表的就地逆置。

输入

先输入一个小于100的正整数n，再从小到大的输入n个正整数，建立一个顺序表，然后实现顺序表的就地逆置。

输出

按顺序输出逆置后的顺序表的所有元素，每个元素占一行。

输入样例

3  
300 3000  50000

输出样例

50000  
3000  
300

#include<stdio.h>

#include<stdlib.h>

typedef int ElemType;

typedef struct tagLNode {

ElemType data;

struct tagLNode \*next;

}LNode, \*LinkList;

void CreateLinkF(LinkList \*L, int n){

LinkList s;

\*L=(LinkList)malloc(sizeof(LNode));

(\*L)->next=NULL;

for(;n>0;n--){

s=(LinkList)malloc(sizeof(LNode));

scanf("%d",&s->data);

s->next=(\*L)->next;

(\*L)->next=s;

}

}

void ListTraverse(LinkList L, void(\*visit)(ElemType\*)){

LinkList p=L->next;

while(p!=NULL) {

visit(&(p->data));

p=p->next;

}

}

void DestroyList(LinkList \*L){

LinkList q, p=\*L;

while(p!=NULL){

q=p->next;

free(p);

p=q;

}

\*L=NULL;

}

void visit(ElemType \*ep){

printf("%d\n", \*ep);

}

int main()

{

int n;

LinkList L;

scanf("%d", &n);

CreateLinkF(&L, n);

ListTraverse(L, visit);

DestroyList(&L);

return 0;

}

**1456.顺序表元素的比较和删除**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

已知a、b和c三个递增有序的线性表，现在要求对a做如下操作：删除其中既即在b中出现又在c中出现的元素（注意同一表中的元素有可能重复）。

输入

先在第一行输入3个小于100的正整数m，n，l分别表示顺序表a，b，c的长度，然后在第二行输入m个递增的正整数，第三行输入n个递增的正整数，第四行输入l个递增的正整数，分别为这三个线性表的元素，

输出

按顺序输出删除元素后的线性表a的值，每个元素占一行。

输入样例

3 8 5  
23 47 88  
2 5 7 9 47 47 88 99  
10 20 30 40 47

输出样例

23  
88

数据结构作业二：

**1458.单链表的就地逆置**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

读入数据构造一个单链表，实现单链表的就地逆置。

输入

先输入一个小于100的正整数n，再从小到大的输入n个正整数，建立一个单链表，然后实现单链表的就地逆置。

输出

按顺序输出逆置后的单链表的所有元素，每个元素占一行。

输入样例

3  
300 3000  50000

输出样例

50000  
3000  
300

**1459.归并两个递增序列链表为一个递减有序链表**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

假设有两个按元素值递增有序排列的线性表a和b，均以单链表作为存储结构，请编程实现将表a和表b归并成一个按元素值递减有序排列的线性表c（注意：非严格递减，也就是说本题中的数据有可能相等），并要求利用原表的结点空间构造c表。

输入

第一行先输入两个小于100的正整数m，n，第二行从小到大的输入m个整数，第三行从小到大的输入n个整数。

输出

归并这两个序列为一个递减的序列c，用链表存储，之后输出按顺序输出链表c的值，每个数占一行。

输入样例

4 3  
2 6 6 10  
3 10 50

输出样例

50  
10  
10  
6  
6  
3  
2

#include<stdio.h>

#include<stdlib.h>

#define LIST\_INIT\_SIZE 100

#define LIST\_INCREMENT 10

#define LIST\_SIZE 100

typedef int ElemType;

typedef struct tagList{

ElemType \*base;

int length;

int size;

} List;

void visit(ElemType \*ep)

{

printf("%d\n",\*ep);

}

void ListTraverse(List L,void(\*visit)(ElemType \*)){

int i;

ElemType \*p=L.base;

for(i=1;i<=L.length;i++) visit(p++);}

void input(ElemType \*ep)

{

scanf("%d",ep);

}

void CreateList( List \*L,int n)

{

ElemType s,\*p;

p=L->base=(ElemType\*)malloc(LIST\_SIZE\*sizeof(ElemType));

L->length=n;

L->size=LIST\_INIT\_SIZE;

for (;n>0;n--){

scanf("%d",&s);

\*p++=s;

}

}

int ListInsert(List \*L,int i,ElemType e)

{

ElemType \*base,\*p,\*q;

if(i<1||i>L->length+1)return 0;

if(L->length>=L->size){

L->size+=LIST\_INCREMENT;

base=(ElemType \*)realloc(L->base,L->size\*sizeof(ElemType));

L->base=base;

}

q=L->base+i-1;

for(p=L->base+L->length-1;p>=q;--p)

\*(p+1)=\*p;

\*q=e;

++(L->length);

return 1;

}

void InitList( List \*L)

{

L->base=(ElemType\*)malloc(LIST\_SIZE\*sizeof(ElemType));

L->length=0;

L->size=LIST\_INIT\_SIZE;

}

int GetElem(List L,int i,ElemType \*e)

{

if (i<1||i>L.length)return 0;

if(e!=NULL) \*e=\*(L.base+i-1);

return 1;

}

int ListLength(List L)

{

return L.length;

}

void MergeList(List La,List Lb,List \*Lc)

{

int i=ListLength(La),j=ListLength(Lb),k=0;

ElemType a,b;

InitList(Lc);

while (i>=1&&j>=1){

GetElem(La,i,&a),GetElem(Lb,j,&b);

if(a>=b){

ListInsert(Lc,++k,a);i--;

}

else{

ListInsert(Lc,++k,b);j--;}

}

while (i>=1){

for(;i>0;i--){

GetElem(La,i,&a),ListInsert(Lc,++k,a);}

}

while (j>=1){

for(;j>0;j--){

GetElem(Lb,i,&b),ListInsert(Lc,++k,b);}

}

}

int main()

{

int m,n;

List La,Lb,Lc;

scanf("%d%d",&n,&m);

CreateList(&La,n);

CreateList(&Lb,m);

MergeList( La, Lb, &Lc);

ListTraverse(Lc, &visit);

return 0;

}

**1460.链表元素的比较和删除**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

已知a、b和c三个递增有序的链表，现在要求对a做如下操作：删除其中既即在b中出现又在c中出现的元素（注意同一表中的元素有可能重复）。

输入

先在第一行输入3个小于100的正整数m，n，l分别表示链表a，b，c的长度，然后在第二行输入m个递增的正整数，第三行输入n个递增的正整数，第四行输入l个递增的正整数，分别为这三个链表的元素，

输出

按顺序输出删除元素后的链表a的值，每个元素占一行。

输入样例

3 8 5  
23 47 88  
2 5 7 9 47 47 88 99  
10 20 30 40 47

输出样例

23  
88

数据结构作业三：

**1473.括号匹配**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

假设一个算术表达式中可以包含三种括号：圆括号"("和")",方括号"["和"]"和花括号"{"和"}"，且这三种括号可按任意的次序嵌套使用，如：...[...(...)...{...{...}...[...]...}...]...，编写程序判别给定表达式中所含括号是否正确配对出现。

输入

输入一个由这几种括号组成的字符串，字符串长度不大于100。

输出

若正确则输出"Yes"，否则输出"No"。注意字母的大小写，结尾带回车。

输入样例

{()[]}{}

输出样例

Yes

#include<stdio.h>

#include<stdlib.h>

#define M 100

#define N 10

typedef char ElemType;

typedef struct tagStack{

ElemType \*base;

ElemType \*top;

int size;

} Stack;

void InStack(Stack \*S){

S->base=(ElemType \*)malloc(M\*sizeof(ElemType));

S->top=S->base;

S->size=M;

}

void shutStack(Stack \*S){

free(S->base);

S->base=S->top=NULL;

S->size=0;

}

int Push(Stack \*S, ElemType e){

if(S->top-S->base >= S->size){

S->base=(ElemType \*)realloc(S->base,S->size\*sizeof(ElemType));

S->top=S->base+S->size;

S->size+=N;

}

\*(S->top)++=e;

return 1;

}

int Pop(Stack \*S, ElemType \*e){

if(S->top==S->base||e==NULL) return 0;

\*e=\*(--S->top);

return 1;

}

int GetTop(Stack \*S, ElemType \*e){

if(S->top>S->base && e!=NULL){

\*e=\*(S->top-1);

return 1;

}

return 0;

}

int StackEmpty(Stack \*S){

return S->top==S->base;

}

int main()

{

char s[100], e, a;

int i, cnt=0;

Stack S;

InStack(&S);

while((s[cnt]=getchar())!='\n')

cnt++;

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

if(i==cnt){

if(StackEmpty(&S))

printf("Yes\n");

else

printf("No\n");

}

else if(s[i] == '{' || s[i] == '(' || s[i] == '[')

Push(&S, s[i]);

else if(s[i] == '}' || s[i] == ')' || s[i] == ']'){

GetTop(&S, &e);

if(StackEmpty(&S)){

printf("No\n");

break;

}

else if(s[i] - e < 3 && s[i] - e >-3)

Pop(&S, &a);

else {

printf("No\n");

break;

}

}

}

shutStack(&S);

return 0;

}

**1474.四则元算表达式转换为逆波兰式**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

假设表达式由单字母整数类型变量和双目四则运算符构成（"+","-","\*","/"）,写一个程序，将一个通常书写形式且书写正确的表达式转换为逆波兰式。

输入

输入一个完整的正确的四则运算表达式，长度不超过100（其中仅含有："("、")"、"a"、"b"和"c"五种字符）。

输出

输出这个表达式的逆波兰表达式。注意结尾带回车。

输入样例

输入样例一：(a+b)  
输入样例二：((a+b)\*(a-c))  
输入样例三：((a\*(b+c))-(a/c))

输出样例

输出样例一：ab+  
输出样例二：ab+ac-\*  
输出样例三：abc+\*ac/-

/\*假设表达式由单字母整数类型变量和双目四则运算符构成

（"+","-","\*","/"）,写一个程序，将一个通常书写形式

且书写正确的表达式转换为逆波兰式。\*/

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define STACK\_INIT\_SIZE 100

#define STACK\_INCREMENT 10

typedef char ElemType;

typedef struct tagStack{

ElemType \*base;

ElemType \*top;

int stacksize;

}Stack;

void InitStack(Stack \*S);

void DestoryStack(Stack \*S);

int StackEmpty(Stack S);

int GetTop(Stack S,ElemType \*e);

int Push(Stack \*S,ElemType e);

int Pop(Stack \*S,ElemType \*e);

char Precede(char left,char right)

{

int isp;//in stack priority

int icp;//incoming priority

switch(left){

case '\*':case '/': isp=3; break;

case '+':case '-': isp=2; break;

case '#': isp=1;break;

case '(':

if(right == ')') return '=';

else if(right == '#') return -1;

else return '<';

break;

case ')':

if(right == '(') return -1;

else return '>';

break;

default: ;

}

switch(right){

case '\*':case '/': icp=3; break;

case '+':case '-': icp=2; break;

case '#':icp=1;break;

case '(': return '<';

case ')': return '>';

default: ;

}

if(isp<icp) return '<';

else return '>';

}

int is\_optr(char c)

{

if(c == '+' || c == '-' || c == '\*' || c == '/'

|| c == '(' || c == ')' || c == '#') return 1;

return 0;

}

int main()

{

Stack OPTR;//operator

char c,tope;

char str[100];

int i=0;

InitStack(&OPTR);

Push(&OPTR,'#');

gets(str);

strcat(str,"#");

GetTop(OPTR,&tope);

c=str[i++];

while ( (c!='#') || tope != '#')

{

if (!is\_optr(c)) {

putchar(c);

c=str[i++];

}else{

switch(Precede(tope,c)){

case '<':

Push(&OPTR,c);

c=str[i++];

break;

case '=':

Pop(&OPTR,&tope);

c=str[i++];

break;

case '>':

Pop(&OPTR,&tope);

//重点

putchar(tope);

break;

}

}

GetTop(OPTR,&tope);

}

putchar('\n');

DestoryStack(&OPTR);

return 0;

}

void InitStack(Stack \*S)

{

S->base= (ElemType \*)malloc(STACK\_INIT\_SIZE\*sizeof(ElemType));

S->top=S->base;

S->stacksize=STACK\_INIT\_SIZE;

}

void DestoryStack(Stack \*S)

{

free(S->base);

S->base = S->top = NULL;

S->stacksize=0;

}

int StackEmpty(Stack S)

{

return S.top==S.base;

}

int GetTop(Stack S,ElemType \*e)

{

if(S.top>S.base && e!=NULL){

\*e = \*(S.top-1);

return 1;

}

return 0;

}

int Push(Stack \*S,ElemType e)

{

ElemType \*nptr;

if(S->top - S->base >= S->stacksize){

nptr=(ElemType \*)realloc(S->base ,

(S->stacksize + STACK\_INCREMENT)\*sizeof(ElemType));

if(!nptr) return -1;

else S->base=nptr;

S->top = S->base + S->stacksize;

S->stacksize += STACK\_INCREMENT;

}

\*(S->top++) = e;

return 0;

}

int Pop(Stack \*S,ElemType \*e)

{

if(S->top == S->base || e==NULL) return -1;

\*e=\*(--(S->top));

return 0;

}

**1475.rear和length表示的循环队列**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

假设将循环队列定义为：以与变量rear和length分别指示循环队列中队尾元素的位置和内含元素的个数。试给出此循环队列的队满条件，并写出相应的入队列和出队列的算法（在出队列的算法中要返回队头元素）。

输入

先输入一个不大于100的正整数n（输入数据个数）和m（循环队列数组的大小），再输入n个整数，其中输入0表示出队。

输出

要求用循环队列实现，队满时如有入队则输出"No"并丢掉此元素，出队时输出这个元素。注意：这种表示方法队列中可以存储m个元素！

输入样例

10 5  
2 6 0 56 89 34 2322 25 0 10

输出样例

2  
No  
6

//(1)

#include<stdio.h>

#include<stdlib.h>

int M;

typedef int ElemType;

typedef struct tagQueue{

ElemType \*base;

int front;

int rear;

}Queue;

void Inqueue(Queue \*Q){

Q->base=(ElemType \*)malloc(M\*sizeof(ElemType));

Q->front=Q->rear=0;

}

int QueueEmpty(Queue \*Q){

return Q->front==Q->rear;

}

int DQueue(Queue \*Q, ElemType \*e)

{

if(Q->front==Q->rear) return 0;

Q->front=(Q->front+1)% M;

if(e!=NULL) \*e=Q->base[Q->front];

return 1;

}

int EQueue(Queue \*Q, ElemType e)

{

if((Q->rear+1)% M==Q->front) return 0;

Q->rear=(Q->rear+1)% M;

Q->base[Q->rear]=e;

return 1;

}

int QueueL(Queue \*Q){

return Q->rear-Q->front;

}

void shutqueue(Queue \*Q)

{

if(Q->base) free(Q->base);

Q->base=NULL;

Q->front=Q->rear=0;

}

int main()

{

Queue Q;

int a[100], n, m, e, i;

scanf("%d%d", &n, &m);

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

scanf("%d", &a[i]);

M=m+1;

Inqueue(&Q);

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

if(a[i]==0){

DQueue(&Q, &e);

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

}

else if((Q.rear+1)% M==Q.front){

printf("No\n");

continue;

}

else EQueue(&Q, a[i]);

}

shutqueue(&Q);

return 0;

}

//(2)

/\*循环队列\*/

#include<stdio.h>

#include<stdlib.h>

#define MAX\_QSIZE 100

typedef int ElemType;

typedef struct tagQueue{

ElemType \*base;

int front;

int rear;

}Queue;

void InitQueue(Queue \*Q);

void DestoryQueue(Queue \*Q);

void ClearQueue(Queue \*Q);

int QueueEmpty(Queue Q);

int QueueLength(Queue Q);

int GetHead(Queue Q,ElemType \*e);

int EnQueue(Queue \*Q,ElemType e);

int DeQueue(Queue \*Q,ElemType \*e);

int main()

{

return 0;

}

void InitQueue(Queue \*Q)

{

Q->base= (ElemType \*)malloc(MAX\_QSIZE\*sizeof(ElemType));

Q->front = Q->rear = 0;

}

void DestoryQueue(Queue \*Q)

{

if(Q->base) free(Q->base);

Q->base = NULL;

Q->front = Q->rear = 0;

}

void ClearQueue(Queue \*Q)

{

Q->front = Q->rear = 0;

}

int QueueEmpty(Queue Q)

{

return Q.front == Q.rear;

}

int QueueLength(Queue Q)

{

return Q.rear - Q.front;

}

int GetHead(Queue Q,ElemType \*e)

{

if(Q.front==Q.rear) return 0;

if(e!=NULL) \*e=Q.base[Q.front];

return 1;

}

int EnQueue(Queue \*Q,ElemType e)

{

if((Q->rear+1)%MAX\_QSIZE == Q->front) return 0;//full

Q->rear = (Q->rear+1)%MAX\_QSIZE;

Q->base[Q->rear]=e;

return 1;

}

int DeQueue(Queue \*Q,ElemType \*e)

{

if(Q->front == Q->rear) return 0;

Q->front = (Q->front+1)%MAX\_QSIZE;

if(e!=NULL) {

\*e=Q->base[Q->front];

return 1;

}

return 0;

}

**1487.k阶斐波那契数列**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

已知K阶斐波那契数列定义为：  
f0 = 0,  f1 = 0, … , fk-2 = 0, fk-1 = 1;  
fn = fn-1 + fn-2 + … + fn-k , n = k , k + 1, …  
利用循环队列编写求k阶斐波那契数列中前n+1项（f0, f1, f2,…, fn）的算法，要求满足：fn<=max,并且fn+1>max,其中max为某个约定的常数。（注意：本题所用循环队列的容量仅为k，则在算法执行结束时，留在队列中的元素应是所求k阶斐波那契序列中的最后k项fn-k-1 , fn-k, …, fn）。

输入

输入表示阶数的k（2<=  k  <= 100）以及表示某个常数的max（0 <= max <= 100000）。

输出

输出满足条件的项n（n从0开始计数），占一行；   
以及第n项的值，占一行；  
输出一个回车符。

输入样例

4 10000

输出样例

17  
5536

数据结构作业四：

**1485.稀疏矩阵的加法(实现C=A+B)**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

输入两个稀疏矩阵，输出它们相加的结果。

输入

第一行输入四个正整数，分别是两个矩阵的行m、列n、第一个矩阵的非零元素的个数t1和第二个矩阵的非零元素的个数t2，接下来的t1+t2行是三元组，分别是第一个矩阵的数据和第二个矩阵的数据, 三元组的第一个元素表示行,第二个元素表示列,第三个元素是该元素的值。

输出

输出相加后的矩阵三元组。

输入样例

3 4 3 2  
1 1 1  
1 3 1  
2 2 2  
1 2 1  
2 2 3

输出样例

1 1 1  
1 2 1  
1 3 1  
2 2 5

**1493.稀疏矩阵的加法(用十字链表实现A=A+B)**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

输入两个稀疏矩阵A和B，用十字链表实现A=A+B，输出它们相加的结果。

输入

第一行输入四个正整数，分别是两个矩阵的行m、列n、第一个矩阵的非零元素的个数t1和第二个矩阵的非零元素的个数t2，接下来的t1+t2行是三元组，分别是第一个矩阵的数据和第二个矩阵的数据, 三元组的第一个元素表示行,第二个元素表示列,第三个元素是该元素的值。

输出

输出相加后的矩阵三元组。

输入样例

3 4 3 2  
1 1 1  
1 3 1  
2 2 2  
1 2 1  
2 2 3

输出样例

1 1 1  
1 2 1  
1 3 1  
2 2 5

### 1497.数组元素移动

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

试设计一个算法，将数组a中的元素a[0]至a[n-1]循环右移k位，并要求只用一个元素大小的附加存储，元素移动或交换次数为O(n)。

输入

先输入一个大于1且小于100的正整数n，再输入n个整数存到数组a中，最后输入一个小于n正整数k，

输出

循环移动k位后输出。

输入样例

5  
2 6 15 39 5  
3

输出样例

15  
39  
5  
2  
6

}

数组元素的移动：

#include<stdio.h>

int main()

{

int n, k, i, a[100];

scanf("%d", &n);

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

scanf("%d",&a[i]);

}

scanf("%d",&k);

for(i=n-k;i<n;i++)

printf("%d\n", a[i]);

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

printf("%d\n", a[i]);

return 0;

}

**1496.求广义表深度**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

按表头、表尾的分析方法重写求广义表深度的递归算法

输入

输入一串以'('开始,以')'结束的字符串,并且输入的左右括号必须匹配,如:(),(())..

输出

分别输出按表头、表尾分析方法求广义表深度的结果，每个结果占一行。

输入样例

((a,b,(c,(d,e),f)),g)

输出样例

4  
4

提示

注意：每个数据后面都有回车

广义表的深度

#include<stdio.h>

int main(){

int a=0, b, i=0, j, t=0, A[100];

char C[100];

while((C[i]=getchar())!='\n') i++;

for(j=1;j<i-1;j++){

if(C[j]=='(') a++;

else if(C[j]==')') {

A[t]=a;

a--;

t++;

}

}

for(j=0;j<t-1;j++){

if(A[j]>=A[j+1]){

b=A[j];

A[j]=A[j+1];

A[j+1]=b;

}

}

if(t>=1){

printf("%d\n", A[t-1]+1);

printf("%d\n", A[t-1]+1);

}

else{

printf("%d\n", 1);

printf("%d\n", 1);

}

return 0;

}

数据结构作业五：

**1498.计算二叉树中叶子节点的数目**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

先序建立一棵二叉树，采用二叉链表结构存储，并计算该二叉树中叶子节点的数目。

输入

输入一串先序遍历字符串，根据此字符串建立一棵二叉树（以指针方式存储）。  
如输入某二叉树的先序序列为ABC##DE#G##F###（其中#表示空格字符，空格字符代表空树）。

输出

输出叶子节点的个数（输出结果后换行）。

输入样例

ABC##DE#G##F###

输出样例

3

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MaxSize 1000

typedef char DataType;

struct Node{

DataType info;

struct Node \*lchild, \*rchild;

};

typedef Node Node, \*pNode;

struct Tree{

pNode root;

char str[MaxSize];

int pos;

};

typedef struct Tree Tree, \*pTree;

pNode initialize\_node(pTree pT)

{

if(pT->str[pT->pos] == '\0') return NULL;

if(pT->str[pT->pos] == '#') {pT->pos++; return NULL;}

pNode pN = (pNode)malloc(sizeof(struct Node));

pN->info = pT->str[pT->pos++];

pN->lchild = initialize\_node(pT);

pN->rchild = initialize\_node(pT);

return pN;

}

pTree initialize\_tree()

{

pTree pT = (pTree)malloc(sizeof(struct Tree));

scanf("%s", pT->str);

pT->pos = 0;

pT->root = initialize\_node(pT);

return pT;

}

int push(pNode pN){

if(!pN->lchild && !pN->rchild) return 1;

int res = 0;

if(pN->lchild) res += push(pN->lchild);

if(pN->rchild) res += push(pN->rchild);

return res;

}

int search(pTree pT){

if(pT->root == NULL) return 0;

return push(pT->root);

}

int main()

{

pTree pT = initialize\_tree();

printf("%d\n",search(pT));

return 0;

}

**1501.用括号法递归建立二叉树**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

如果用大写字母标识二叉树节点，则一棵二叉树可以用以下字母序列建立，#表示空节点。试写一个递归算法，有这种形式的字符序列，建立相应的二叉树的二叉链表存储结构，并按层次遍历输出。如一棵二叉树可表示为：A(B(#,D),C(E(#,F),#))

输入

按题中要求，用括号法输入二叉树序列，#表示空节点

输出

按层次遍历建立好的二叉树并输出

输入样例

A(B(#,D),C(E(#,F),#))

输出样例

ABCDEF

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define MAX 100

typedef char infotype;

typedef struct tagBTNode{

infotype info;

struct tagBTNode\* lchild;

struct tagBTNode\* rchild;

}BinTreeNode,\*btree;

typedef btree QType;

typedef struct tagQueue{

QType \*base;

int front;

int rear;

}Queue;

void intqueue(Queue \*Q)

{

Q->base= (QType \*)malloc(MAX\*sizeof(QType));

Q->front = Q->rear = 0;

}

int EnQueue(Queue \*Q,QType e)

{

if((Q->rear+1)%MAX== Q->front) return 0;

Q->rear = (Q->rear+1)%MAX;

Q->base[Q->rear]=e;

return 1;

}

int QueueEmpty(Queue Q)

{

return Q.front == Q.rear;

}

void DestoryQueue(Queue \*Q)

{

if(Q->base) free(Q->base);

Q->base = NULL;

Q->front = Q->rear = 0;

}

int DeQueue(Queue \*Q,QType \*e)

{

if(Q->front == Q->rear) return 0;

Q->front = (Q->front+1)%MAX;

if(e!=NULL) {

\*e=Q->base[Q->front];

return 1;

}

return 0;

}

void leoltraverse(btree T ) {

Queue qu;

btree p;

intqueue(&qu);

if ( !T ) {

return;

}

EnQueue ( &qu, T );

while ( !QueueEmpty( qu ) ) {

DeQueue ( &qu, &p );

printf ( "%c",p->info );

if ( p->lchild != NULL )

EnQueue ( &qu, p->lchild );

if ( p->rchild != NULL )

EnQueue ( &qu, p->rchild );

}

DestoryQueue(&qu);

}

void DistoryBinTree (btree &T )

{

if ( T != NULL )

{

DistoryBinTree (T->lchild);

DistoryBinTree (T->rchild);

free(T);

T=NULL;

}

}

void CreateBinTree\_bracket(btree &T, char\* str, int l, int r)

{

int cnt,i;

if(str[l] == '#'){

T=NULL;

return;

}

if ( !( T = (btree) malloc(sizeof(BinTreeNode)) ) ) exit(-1);

T->info = str[l];

if(l==r){

T->lchild = T->rchild = NULL;

return;

}

i=l+2;

for(cnt=0; cnt!=0 || str[i]!=',' ; ++i){

switch(str[i]){

case '(': ++cnt;break;

case ')': --cnt;break;

}

}

CreateBinTree\_bracket(T->lchild, str, l+2, i-1);

CreateBinTree\_bracket(T->rchild, str, i+1, r-1);

}

int main()

{

int l,r;

char str[100];

btree BT = NULL;

gets(str);

l=0;

r=strlen(str)-1;

CreateBinTree\_bracket(BT, str, l , r);

leoltraverse( BT );

printf("\n");

DistoryBinTree( BT );

return 0;

}

**1502.建立二叉树的二叉链表**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

已知二叉树的中序序列和前序序列存放在两个一维数组中，尝试建立二叉树的二叉链表。检查是否建立成功的方法是输出该二叉树的后序序列。

输入

分别输入二叉树的中序序列和前序序列

输出

输出二叉树的后序序列

输入样例

ABCDEGF  
CBEGDFA

输出样例

C  
G  
E  
F  
D  
B  
A

提示

先建立二叉链表，在通过其输出后序序列

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MaxSize 1000

typedef char infotype;

struct Node{

infotype info;

struct Node \*lchild, \*rchild;

};

typedef Node Node, \*pNode;

struct Tree{

pNode root;

char pre[MaxSize];

char ino[MaxSize];

int pos, len;

};

typedef struct Tree Tree, \*pTree;

pNode intnode(pTree pT, int l, int r)

{

if(l > r) return NULL;

pNode pN = (pNode)malloc(sizeof(struct Node));

int i=l;

for(; i<=r; i++)

if(pT->ino[i] == pT->pre[pT->pos])

break;

if(i>r) i=r;

pN->info = pT->pre[pT->pos++];

pN->lchild = intnode(pT, l, i-1);

pN->rchild = intnode(pT, i+1, r);

return pN;

}

pTree inttree()

{

pTree pT = (pTree)malloc(sizeof(struct Tree));

scanf("%s", pT->pre);

scanf("%s", pT->ino);

pT->len = strlen(pT->pre);

pT->pos = 0;

pT->root = intnode(pT, 0, pT->len-1);

return pT;

}

void push(pNode pN){

if(pN == NULL) return;

push(pN->lchild);

push(pN->rchild);

printf("%c\n", pN->info);

}

void print(pTree pT)

{

push(pT->root);

}

int main()

{

pTree pT = inttree();

print(pT);

return 0;

}

**1503.输出以二叉树表示的算术表达式**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

编写一个程序，输出以二叉树表示的算术表达式，若表达式中有括号，则应在输出时加上

输入

按先序输入一行字符，其中#表示取消建立子树节点

输出

输出以二叉树表示的算术表达式

输入样例

\*+a(###b#)##c##

输出样例

(a+b)\*c

提示

注意，在本题中'('和')'也作为二叉树中的节点值，在程序编写过程中可以以课本中算法6.4为基础。

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define Max 1000

typedef char DataType;

struct Node{

DataType info;

struct Node \*lchild, \*rchild;

};

typedef Node Node, \*pNode;

struct Tree{

pNode root;

char str[Max];

int pos;

};

typedef struct Tree Tree, \*pTree;

pNode intnode(pTree pT)

{

if(pT->str[pT->pos] == '\0') return NULL;

if(pT->str[pT->pos] == '#'){ pT->pos++; return NULL;}

pNode pN = (pNode)malloc(sizeof(struct Node));

pN->info = pT->str[pT->pos++];

pN->lchild = intnode(pT);

pN->rchild = intnode(pT);

return pN;

}

pTree inttree()

{

pTree pT = (pTree)malloc(sizeof(struct Tree));

scanf("%s", pT->str);

pT->pos = 0;

pT->root = intnode(pT);

return pT;

}

void tui(pNode pN){

if(pN == NULL) return;

tui(pN->lchild);

printf("%c", pN->info);

tui(pN->rchild);

}

void show(pTree pT)

{

tui(pT->root);

printf("\n");

}

int main()

{

pTree pT = inttree();

show(pT);

return 0;

}

数据结构实验一：

**1037.合并有序数组(Merging sorted array)**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

给你两个有序且升序的数组，请你把它们合成一个升序数组并输出  
Give you two ordered ascending array, you put them into one ascending array and output.

输入

第一行为一个正整数n，n<=20 ；  
第二行为n个数字，这n个数字用空格隔开  
第三行为一个正整数m，m<=20 ；  
第四行为M个数字，这m个数字用空格隔开  
The first line is a positive integer n, n <= 20;  
The second line are n numbers separated by space  
The third is a positive integer m, m <= 20;  
The fourth line are m numbers separated by space

输出

输出合并后的数组，每个数字占一行，  
Output the combined array, each number per line,

输入样例

3  
1 3 7  
5  
2 4 6 8 10

输出样例

1  
2  
3  
4  
6  
7  
8  
10

#include<stdio.h>

#include<stdlib.h>

typedef int ElemType;

typedef struct tagLNode {

ElemType data;

struct tagLNode \*next;

}LNode, \*LinkList;

void CreateLinkR(LinkList \*L, int n){

LinkList p, s;

p=\*L=(LinkList)malloc(sizeof(LNode));

for(;n>0;n--){

s=(LinkList)malloc(sizeof(LNode));

scanf("%d",&s->data);

p->next=s,p=s;

}

p->next=NULL;

}

void InitList(LinkList \*L){

\*L=(LinkList)malloc(sizeof(LNode));

(\*L)->next=NULL;

}

void ListTraverse(LinkList \*L, void(\*visit)(ElemType\*)){

LinkList p=(\*L)->next;

while(p!=NULL) {

visit(&(p->data));

if(p->next!=NULL)

printf(" ");

else printf("\n");

p=p->next;

}

}

void visit(ElemType \*ep){

printf("%d\n", \*ep);

}

void MergeList(LinkList \*La, LinkList \*Lb, LinkList \*Lc){

LinkList pa, pb, pc;

pa=(\*La)->next;

pb=(\*Lb)->next;

\*Lc=\*La;

pc=\*La;

while(pa&&pb){

if(pa->data<=pb->data){

pc->next=pa;

pc=pa;

pa=pa->next;

}

else {

pc->next=pb;

pc=pb;

pb=pb->next;

}

}

pc->next=pa?pa:pb;

}

int main()

{

int m, n;

LinkList p, q, L;

scanf("%d", &m);

CreateLinkR(&p, m);

scanf("%d", &n);

CreateLinkR(&q, n);

MergeList(&p, &q, &L);

ListTraverse(&L, visit);

return 0;

}---

**1419.高精度计算PI值**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

限制使用双向链表作存储结构，请根据用户输入的一个整数（该整数表示精确到小数点后的位数，可能要求精确到小数点后500位），高精度计算PI值。可以利用反三角函数幂级展开式来进行计算：

输入

输入的一个正整数n

输出

输出PI的值，精确到小数点后n位，最后输出一个回车。

输入样例

5

输出样例

3.14159

数据结构实验二：

**1484.稀疏矩阵转置**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

输出稀疏矩阵的转置矩阵。（行列均不大于20）

输入

输入两个正整数n和m,为矩阵的行数和列数,然后输入矩阵三元组，（0 0 0）表示结束输入。

输出

转置后的矩阵。

输入样例

4 4  
1 1 1  
2 1 2  
3 2 3  
0 0 0

输出样例

1 1 1  
1 2 2  
2 3 3

矩阵的转置：

#include<stdio.h>

#define MAXSIZE 400

typedef int ElemType;

typedef struct{

int i, j;

ElemType e;

}Triple;

typedef struct{

Triple data[MAXSIZE+1];

int mu, nu, tu;

}TSMatrix;

int TransposeSMatrix(TSMatrix M, TSMatrix &T){

int p, q=1, col;

T.mu=M.nu;

T.nu=M.mu;

T.tu=M.tu;

if(T.tu){

for(col=1;col<=M.nu; ++col)

for(p=1;p<=M.tu; ++p)

if(M.data[p].j==col){

T.data[q].i=M.data[p].j;

T.data[q].j=M.data[p].i;

T.data[q].e=M.data[p].e;

++q;

}

}

return 1;

}

int main()

{

int n, m, i, a=0;

TSMatrix M, T;

scanf("%d%d", &n, &m);

for(i=1;;i++){

scanf("%d%d%d", &(M.data[i].i), &(M.data[i].j), &(M.data[i].e));

a++;

if(M.data[i].i==0 && M.data[i].j==0 && M.data[i].e==0) break;

}

a--;

M.tu=a;

M.mu=n;

M.nu=m;

TransposeSMatrix(M, T);

for(i=1;i<=M.tu;i++){

printf("%d %d %d\n", T.data[i].i, T.data[i].j, T.data[i].e);

}

return 0;

}

**1485.稀疏矩阵的加法(实现C=A+B)**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

输入两个稀疏矩阵，输出它们相加的结果。

输入

第一行输入四个正整数，分别是两个矩阵的行m、列n、第一个矩阵的非零元素的个数t1和第二个矩阵的非零元素的个数t2，接下来的t1+t2行是三元组，分别是第一个矩阵的数据和第二个矩阵的数据, 三元组的第一个元素表示行,第二个元素表示列,第三个元素是该元素的值。

输出

输出相加后的矩阵三元组。

输入样例

3 4 3 2  
1 1 1  
1 3 1  
2 2 2  
1 2 1  
2 2 3

输出样例

1 1 1  
1 2 1  
1 3 1  
2 2 5

稀疏矩阵的加法：

#include<stdio.h>

#define MAXSIZE 400

typedef int ElemType;

typedef struct{

int i, j;

ElemType e;

}Triple;

typedef struct{

Triple data[MAXSIZE+1];

int mu, nu, tu;

}TSMatrix;

int main()

{

int m, n, t1, t2, i, a, t, j;

TSMatrix P, Q, T;

scanf("%d%d%d%d", &m, &n, &t1, &t2);

for(i=1;i<=t1;i++)

scanf("%d%d%d", &(P.data[i].i), &(P.data[i].j), &(P.data[i].e));

for(i=1;i<=t2;i++)

scanf("%d%d%d", &(Q.data[i].i), &(Q.data[i].j), &(Q.data[i].e));

a=0;

t=0;

j=0;

for(i=1;i<=t1;i++){

for(;t<=t2;t++){

if(P.data[i].i==Q.data[t].i){

if(P.data[i].j==Q.data[t].j){

if((P.data[i].e+Q.data[t].e)!=0){

T.data[j].i=P.data[i].i;

T.data[j].j=P.data[i].j;

T.data[j].e=P.data[i].e+Q.data[t].e;

a++;

t++;

j++;

break;}

}

if(P.data[i].j>Q.data[t].j){

T.data[j].i=Q.data[t].i;

T.data[j].j=Q.data[t].j;

T.data[j].e=Q.data[t].e;

a++;

j++;

continue;

}

else{

T.data[j].i=P.data[i].i;

T.data[j].j=P.data[i].j;

T.data[j].e=P.data[i].e;

a++;

j++;

break;

}

}

if(P.data[i].i>Q.data[t].i){

T.data[j].i=Q.data[t].i;

T.data[j].j=Q.data[t].j;

T.data[j].e=Q.data[t].e;

a++;

j++;

continue;

}

else{

T.data[j].i=P.data[i].i;

T.data[j].j=P.data[i].j;

T.data[j].e=P.data[i].e;

a++;

j++;

break;

}

}

}

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

printf("%d %d %d\n", T.data[i].i, T.data[i].j, T.data[i].e);

return 0;

}

**1493.稀疏矩阵的加法(用十字链表实现A=A+B)**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

输入两个稀疏矩阵A和B，用十字链表实现A=A+B，输出它们相加的结果。

输入

第一行输入四个正整数，分别是两个矩阵的行m、列n、第一个矩阵的非零元素的个数t1和第二个矩阵的非零元素的个数t2，接下来的t1+t2行是三元组，分别是第一个矩阵的数据和第二个矩阵的数据, 三元组的第一个元素表示行,第二个元素表示列,第三个元素是该元素的值。

输出

输出相加后的矩阵三元组。

输入样例

3 4 3 2  
1 1 1  
1 3 1  
2 2 2  
1 2 1  
2 2 3

输出样例

1 1 1  
1 2 1  
1 3 1  
2 2 5

**1492.稀疏矩阵的乘法**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

计算两个稀疏矩阵的乘法

输入

首先输入第一个矩阵的行数和列数，再输入该矩阵的三元组形式，以0 0 0结束　　  
然后输入第二个矩阵的行数和列数，再输入该矩阵的三元组形式，以0 0 0结束

输出

两个矩阵相乘后得到的矩阵的三元组形式

输入样例

3 3　　  
1 1 1　　  
2 2 2　　  
2 3 4　　  
3 1 -4　　  
0 0 0　　  
3 3  
1 3 -2　　  
2 3 -5　　  
3 1 8　　  
3 2 -6　　  
0 0 0

输出样例

1 3 -2　　  
2 1 32　　  
2 2 -24　　  
2 3 -10　　  
3 3 8

稀疏矩阵的乘法：

#include<stdio.h>

#define MAXSIZE 1024

#define MAXRC 100

typedef int ElemType;

typedef struct{

int i, j;

ElemType e;

}Triple;

typedef struct{

Triple data[MAXSIZE+1];

int rops[MAXRC+1];

int mu, nu, tu;

}RLSMatrix;

int rpot[MAXRC+1];

int main()

{

int n1, m1, n2, m2, a=0, b=0, g=0, p, q, i, j, k, r, t;

RLSMatrix A, B, C;

int temp[1024];

int num[1024];

scanf("%d%d", &n1, &m1);

for(i=1;;i++){

scanf("%d%d%d", &(A.data[i].i), &(A.data[i].j), &(A.data[i].e));

a++;

if(A.data[i].i==0 && A.data[i].j==0 && A.data[i].e==0) break;

}

scanf("%d%d", &n2, &m2);

for(i=1;;i++){

scanf("%d%d%d", &(B.data[i].i), &(B.data[i].j), &(B.data[i].e));

b++;

if(B.data[i].i==0 && B.data[i].j==0 && B.data[i].e==0) break;

}

a--;

b--;

A.tu=a;

A.mu=n1;

A.nu=m1;

B.tu=b;

B.mu=n2;

B.nu=m2;

if(A.nu!=B.mu) return 0;

C.mu=A.mu;

C.nu=B.nu;

C.tu=0;

if(A.tu\*B.tu!=0){

for(i=1;i<=B.mu;i++)

num[i]=0;

for(k=1;k<=B.tu;k++){

i=B.data[k].i;

num[i]++;

}

rpot[1]=1;

for(i=2;i<=B.mu;i++)

rpot[i]=rpot[i-1]+num[i-1];

r=0;

p=1;

for(i=1;i<=A.mu;i++){

for(j=1;j<=B.nu;j++) temp[j]=0;

while(A.data[p].i==i){

k=A.data[p].j;

if(k<B.mu) t=rpot[k+1];

else t=B.tu+1;

for(q=rpot[k];q<t;q++){

j=B.data[q].j;

temp[j]+=A.data[p].e\*B.data[q].e;

}

p++;

}

for(j=1;j<=B.nu;j++)

if(temp[j]){

r++;

C.data[r].i=i;

C.data[r].j=j;

C.data[r].e=temp[j];

g++;

}

}

C.tu=r;

}

for(i=1;i<=g;i++){

printf("%d %d %d\n", C.data[i].i, C.data[i].j, C.data[i].e);

}

return 0;

}

数据结构实验三：

**1486.哈夫曼编/译码器**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

写一个哈夫曼码的编/译码系统，要求能对要传输的报文进行编码和解码。构造哈夫曼树时，权值小的放左子树，权值大的放右子树，编码时右子树编码为1，左子树编码为0.

输入

输入表示字符集大小为n（n <= 100）的正整数，以及n个字符和n个权值（正整数，值越大表示该字符出现的概率越大）；  
输入串长小于或等于100的目标报文。

输出

经过编码后的二进制码，占一行；  
以及对应解码后的报文，占一行；  
最后输出一个回车符。

输入样例

5 a b c d e 12 40 15 8 25  
bbbaddeccbbb

输出样例

00011111110111010110110000  
bbbaddeccbbb

提示

利用编码前缀性质。

数据结构实验四：

#include<stdio.h>

#define MAXVALUE 10000

typedef struct{

int weight;

int parent;

int lchild;

int rchild;

}HNodeType;

typedef struct{

int bit[100];

int start;

}HCodeType;

void HaffmanCode(HNodeType HFMTree[], HCodeType HuffCode[], int n){

HCodeType cd;

int i, j, c, p;

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

cd.start=n-1;

c=i;

p=HFMTree[c].parent;

while(p!=-1){

if(HFMTree[p].lchild==c)

cd.bit[cd.start]=0;

else cd.bit[cd.start]=1;

cd.start--;

c=p;

p=HFMTree[p].parent;

}

for(j=cd.start+1;j<n;j++)

HuffCode[i].bit[j]=cd.bit[j];

HuffCode[i].start=cd.start+1;

}

}

int main()

{

int n, i=0, j=0, t=1, m1, x1, m2, x2, a, b, c;

scanf("%d", &n);

HNodeType HFMTree[200];

HCodeType HuffCode[100];

int A[100];

char C[100], D[100];

for(;i<n;i++)

if((C[i]=getchar())!='\n'&&(C[i]=getchar())!=' ');

i=0;

while(i<n){

scanf("%d", &A[i]);

i++;

}

scanf("%c", &D[0]);

while((D[t]=getchar())!='\n') t++;

for(i=0;i<2\*n-1;i++){

HFMTree[i].weight=0;

HFMTree[i].parent=-1;

HFMTree[i].lchild=-1;

HFMTree[i].rchild=-1;

}

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

HFMTree[i].weight=A[i];

for(i=0;i<n-1;i++){

x1=MAXVALUE;

x2=MAXVALUE;

m1=0;

m2=0;

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

if(HFMTree[j].parent==-1 && HFMTree[j].weight<x1){

x2=x1;

m2=m1;

x1=HFMTree[j].weight;

m1=j;

}

else if(HFMTree[j].parent==-1 && HFMTree[j].weight<x2){

x2=HFMTree[j].weight;

m2=j;

}

}

HFMTree[m1].parent=n+i;

HFMTree[m2].parent=n+i;

HFMTree[n+i].weight=HFMTree[m1].weight+HFMTree[m2].weight;

HFMTree[n+i].lchild=m1;

HFMTree[m1].rchild=m2;

}

HaffmanCode(HFMTree, HuffCode, n);

for(a=0;a<t;a++){

for(b=0;b<n;b++){

if(D[a]==C[b]){

for(c=HuffCode[b].start;c<=n-1;c++){

printf("%d", HuffCode[b].bit[c]);

}

}

}

}

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

printf("%c", D[i]);

printf("\n");

return 0;

}

**1440.求赋权图中一个结点到所有结点的最短路径的长度**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

给一个赋权图（无向图），求0号结点到其余所有结点的最短路径的长度。

输入

先输入一个正整数n，然后输入赋权图的邻接矩阵（10000表示无穷大）

输出

按结点编号的顺序输出0号结点所有结点的最短路径的长度。

输入样例

6  
0 1 4 10000 10000 10000  
1 0 2 7 5 10000  
4 2 0 10000 1 10000  
10000 7 10000 0 3 2  
10000 5 1 3 0 6  
10000 10000 10000 2 6 0

输出样例

0 1 3 7 4 9

#include<stdio.h>//迪杰斯特拉算法

#define MaxVertexNum 100 //最大的顶点个数

typedef char VertexType;//顶点类型

typedef int EdgeType;//边的权值类型

typedef struct{

VertexType vexs[MaxVertexNum];//顶点表

EdgeType edges[MaxVertexNum][MaxVertexNum];//邻接矩阵

int n, e;//实际的顶点数和边数

}MGraph;

int ShortestPath(MGraph \*G, int P[], int D[], int N){//Dijkstra算法

int min, i, j, k;

int final[10]={1,0};//D[v]存放从0顶点到终点v的最短路径的长度；P[v]存放相应的最短路径终点的前驱点；INFINITY为邻接矩阵中的无穷大

for(i=0;i<N;i++){//final[N]用来表示集合S;final[j]=0表示顶点j未进入集合S，final[j]=1表示顶点j进入集合S。

D[i]=G->edges[0][i];//P[5]=3说明v5的最短路径最后一段是从v3到达的

P[i]=0;

}

D[0]=0;

final[0]=1;

P[0]=-1;

for(i=1;i<N;i++){

min=10000+1;//min=INFINITY+1;

for(k=0;k<N;k++)

if(final[k]==0 && D[k]<min){

j=k;

min=D[k];

}

final[j]=1;

for(k=0;k<N;k++)

if(final[k]==0 && (D[j]+G->edges[j][k]<D[k])){

D[k]=D[j]+G->edges[j][k];

P[k]=j;

}

}

/\* for(i=1;i<N;i++){//输出最短路径的长度和路径上的结点

printf("%f: %d", D[i], i);

pre=P[i];

while(pre>=0){

printf("<-%d", pre);

pre=P[pre];

}

printf("\n");

}\*/

return 1;

}

int main(){

MGraph G;

int n, i, j, A[10][10], B[10], F[10];

scanf("%d", &n);

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

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

scanf("%d", &A[i][j]);

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

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

G.edges[i][j]=A[i][j];

ShortestPath(&G, B, F, n);

for(i=0;i<n-1;i++)

printf("%d ", F[i]);

printf("%d", F[n-1]);

printf("\n");

return 0;

}

**1504.用迪杰斯特拉算法求赋权图中的最短路径**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

用迪杰斯特拉算法求一点到其余所有结点的最短路径。

输入

先输入一个小于100的正整数n，然后的n行输入图的邻接矩阵（10000表示无穷大，即两点之间没有边），最后输入两个0到n-1的整数表示两个点，

输出

用迪杰斯特拉算法求第一个点到其余所有结点的最短路径。  
并输出这两个点之间的最短路径（按顺序输出最短路径上的每一个点，每个数据占一行）。

输入样例

4  
0 2 10 10000  
2 0 7 3  
10 7 0 6  
1000 3 6 0  
0 2

输出样例

0  
1  
2

#include<stdio.h>//迪杰斯特拉算法

#define MaxVertexNum 100 //最大的顶点个数

typedef char VertexType; //顶点类型

typedef int EdgeType; //边的权值类型

typedef struct{

VertexType vexs[MaxVertexNum];//顶点表

EdgeType edges[MaxVertexNum][MaxVertexNum];//邻接矩阵

int n, e; //实际的顶点数和边数

}MGraph;

int ShortestPath1(MGraph \*G, int P[], int D[], int N, int e){//Dijkstra算法

int min, i, j, k;

int final[100]={1,0}; //D[v]存放从0顶点到终点v的最短路径的长度；P[v]存放相应的最短路径终点的前驱点；INFINITY为邻接矩阵中的无穷大

for(i=0;i<N;i++){ //final[N]用来表示集合S;final[j]=0表示顶点j未进入集合S，final[j]=1表示顶点j进入集合S。

D[i]=G->edges[e][i]; //P[5]=3说明v5的最短路径最后一段是从v3到达的

P[i]=0;

}

D[e]=0;

P[e]=-1;

final[0]=0;

final[e]=1;

for(i=1;i<N;i++){

min=10000+1; //min=INFINITY+1;

for(k=0;k<N;k++)

if(final[k]==0 && D[k]<min){

j=k;

min=D[k];

}

final[j]=1;

for(k=0;k<N;k++)

if(final[k]==0 && (D[j]+G->edges[j][k]<D[k])){

D[k]=D[j]+G->edges[j][k];

P[k]=j;

}

}

return 1;

}

int main(){

MGraph G;

int n, i, j, a, b, t=0, A[100][100], B[100], F[100], L[200];

scanf("%d", &n);

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

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

scanf("%d", &A[i][j]);

scanf("%d", &a);

j=0;

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

scanf("%d%d", &L[j], &L[j+1]);

j=j+2;

}

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

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

G.edges[i][j]=A[i][j];

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

ShortestPath1(&G, B, F, n, L[t]);

b=L[t+1];

printf("%d\n", F[b]);

t=t+2;

}

return 0;

}

#include<stdio.h>//迪杰斯特拉算法

#define MaxVertexNum 100 //最大的顶点个数

typedef char VertexType; //顶点类型

typedef int EdgeType; //边的权值类型

typedef struct{

VertexType vexs[MaxVertexNum];//顶点表

EdgeType edges[MaxVertexNum][MaxVertexNum];//邻接矩阵

int n, e; //实际的顶点数和边数

}MGraph;

int ShortestPath1(MGraph \*G, int P[], int D[], int N, int e){//Dijkstra算法

int min, i, j, k;

int final[100]={1,0}; //D[v]存放从0顶点到终点v的最短路径的长度；P[v]存放相应的最短路径终点的前驱点；INFINITY为邻接矩阵中的无穷大

for(i=0;i<N;i++){ //final[N]用来表示集合S;final[j]=0表示顶点j未进入集合S，final[j]=1表示顶点j进入集合S。

D[i]=G->edges[e][i]; //P[5]=3说明v5的最短路径最后一段是从v3到达的

P[i]=0;

}

D[e]=0;

P[e]=-1;

final[0]=0;

final[e]=1;

for(i=1;i<N;i++){

min=10000+1; //min=INFINITY+1;

for(k=0;k<N;k++)

if(final[k]==0 && D[k]<min){

j=k;

min=D[k];

}

final[j]=1;

for(k=0;k<N;k++)

if(final[k]==0 && (D[j]+G->edges[j][k]<D[k])){

D[k]=D[j]+G->edges[j][k];

P[k]=j;

}

}

return 1;

}

int main(){

MGraph G;

int n, i, j, a, b, t, A[100][100], B[100], F[100], D[100];

scanf("%d", &n);

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

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

scanf("%d", &A[i][j]);

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

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

G.edges[i][j]=A[i][j];

scanf("%d%d", &a, &b);

j=b;

B[b]=b;

i=0;

printf("%d", a);

while(B[b]!=a){

ShortestPath1(&G, B, F, n, a);

D[i]=B[b];

i++;

b=B[b];

}

for(t=i-1;t>=0;t--)

printf("%d\n", D[t]);

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

return 0;

}

**1505.用弗洛伊德算法求赋权图的两点间的最短路径的长度**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

用弗洛伊德算法求任意两点间的最短路径的长度，

输入

先输入一个小于100的正整数n，然后的n行输入图的邻接矩阵（10000表示无穷大，即两点之间没有边），之后再输入一个小于100的正整数m，最后的m行每行输入两个不同的0到n-1之间的整数表示两个点，

输出

用弗洛伊德算法求任意两点间的最短路径的长度，并输出这些两个点之间的最短路径的长度。

输入样例

4  
0 2 10 10000  
2 0 7 3  
10 7 0 6  
1000 3 6 0  
2  
0 2  
3 0

输出样例

9  
5

#include<stdio.h>//弗洛伊德算法

#define MaxVertexNum 30 //最大的顶点个数

typedef char VertexType; //顶点类型

typedef int EdgeType; //边的权值类型

typedef struct{

VertexType vexs[MaxVertexNum];//顶点表

EdgeType edges[MaxVertexNum][MaxVertexNum];//邻接矩阵

int n, e; //实际的顶点数和边数

}MGraph;

void shortestpath(MGraph G, int P[30][30][30], int D[30][30], int v, int w)

{

int i, u;

for(v=0;v<G.n;v++)

for(w=0;w<G.n;w++){

D[v][w]=G.edges[v][w];

for(u=0;u<G.n;u++) P[v][w][u]=0;

if(D[v][w]<10000){

P[v][w][v]=1;

P[v][w][w]=1;

}

}

for(u=0;u<G.n;u++)

for(v=0;v<G.n;v++)

for(w=0;w<G.n;w++)

if((D[v][u]+D[u][w])<D[v][w])

{

D[v][w]=D[v][u]+D[u][w];

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

P[v][w][i]=P[v][u][i]||P[u][w][i];

}

}

int main()

{

MGraph G;

int n, i, j, a, t=0, m, e;

int A[10][10], B[30][30][30], F[30][30], L[20];

scanf("%d", &n);

G.n=n;

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

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

scanf("%d", &A[i][j]);

scanf("%d", &a);

j=0;

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

scanf("%d%d", &L[j], &L[j+1]);

j=j+2;

}

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

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

G.edges[i][j]=A[i][j];

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

shortestpath(G, B, F, L[t], L[t+1]);

m=L[t];

e=L[t+1];

printf("%d\n", F[m][e]);

t=t+2;

}

return 0;

}

#include<stdio.h>//迪杰斯特拉算法

#define MaxVertexNum 100 //最大的顶点个数

typedef char VertexType; //顶点类型

typedef int EdgeType; //边的权值类型

typedef struct{

VertexType vexs[MaxVertexNum];//顶点表

EdgeType edges[MaxVertexNum][MaxVertexNum];//邻接矩阵

int n, e; //实际的顶点数和边数

}MGraph;

int ShortestPath1(MGraph \*G, int P[], int D[], int N, int e){//Dijkstra算法

int min, i, j, k;

int final[100]={1,0}; //D[v]存放从0顶点到终点v的最短路径的长度；P[v]存放相应的最短路径终点的前驱点；INFINITY为邻接矩阵中的无穷大

for(i=0;i<N;i++){ //final[N]用来表示集合S;final[j]=0表示顶点j未进入集合S，final[j]=1表示顶点j进入集合S。

D[i]=G->edges[e][i]; //P[5]=3说明v5的最短路径最后一段是从v3到达的

P[i]=0;

}

D[e]=0;

//final[e]=1;

P[e]=-1;

final[0]=0;

final[e]=1;

for(i=1;i<N;i++){

min=10000+1; //min=INFINITY+1;

for(k=0;k<N;k++)

if(final[k]==0 && D[k]<min){

j=k;

min=D[k];

}

final[j]=1;

for(k=0;k<N;k++)

if(final[k]==0 && (D[j]+G->edges[j][k]<D[k])){

D[k]=D[j]+G->edges[j][k];

P[k]=j;

}

}

return 1;

}

int main(){

MGraph G;

int n, i, j, a, b, t=0, A[100][100], B[100], F[100], L[200];

scanf("%d", &n);

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

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

scanf("%d", &A[i][j]);

scanf("%d", &a);

j=0;

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

scanf("%d%d", &L[j], &L[j+1]);

j=j+2;

}

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

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

G.edges[i][j]=A[i][j];

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

ShortestPath1(&G, B, F, n, L[t]);

b=L[t+1];

printf("%d\n", F[b]);

t=t+2;

}

return 0;

}

**1506.用弗洛伊德算法求赋权图的两点间的最短路径**

**时限：1000ms 内存限制：10000K  总时限：3000ms**

描述

用弗洛伊德算法求任意两点间的最短路径，

输入

先输入一个小于100的正整数n，然后的n行输入图的邻接矩阵（10000表示无穷大，即两点之间没有边），之后再输入一个小于100的正整数m，最后的m行每行输入两个不同的0到n-1之间的整数表示两个点，

输出

用弗洛伊德算法求任意两点间的最短路径，并输出这些两个点之间的最短路径。

输入样例

4  
0 2 10 10000  
2 0 7 3  
10 7 0 6  
1000 3 6 0  
2  
0 2  
3 0

输出样例

0  
1  
2  
3  
1  
0

#include<stdio.h>//迪杰斯特拉算法

#define MaxVertexNum 100 //最大的顶点个数

typedef char VertexType; //顶点类型

typedef int EdgeType; //边的权值类型

typedef struct{

VertexType vexs[MaxVertexNum];//顶点表

EdgeType edges[MaxVertexNum][MaxVertexNum];//邻接矩阵

int n, e; //实际的顶点数和边数

}MGraph;

int ShortestPath1(MGraph \*G, int P[], int D[], int N, int e){//Dijkstra算法

int min, i, j, k;

int final[100]={1,0}; //D[v]存放从0顶点到终点v的最短路径的长度；P[v]存放相应的最短路径终点的前驱点；INFINITY为邻接矩阵中的无穷大

for(i=0;i<N;i++){ //final[N]用来表示集合S;final[j]=0表示顶点j未进入集合S，final[j]=1表示顶点j进入集合S。

D[i]=G->edges[e][i]; //P[5]=3说明v5的最短路径最后一段是从v3到达的

P[i]=0;

}

D[e]=0;

//final[e]=1;

P[e]=-1;

final[0]=0;

final[e]=1;

for(i=1;i<N;i++){

min=10000+1; //min=INFINITY+1;

for(k=0;k<N;k++)

if(final[k]==0 && D[k]<min){

j=k;

min=D[k];

}

final[j]=1;

for(k=0;k<N;k++)

if(final[k]==0 && (D[j]+G->edges[j][k]<D[k])){

D[k]=D[j]+G->edges[j][k];

P[k]=j;

}

}

return 1;

}

int main(){

MGraph G;

int n, i, j, a, b, t=0, A[100][100], B[100], F[100], L[200];

scanf("%d", &n);

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

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

scanf("%d", &A[i][j]);

scanf("%d", &a);

j=0;

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

scanf("%d%d", &L[j], &L[j+1]);

j=j+2;

}

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

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

G.edges[i][j]=A[i][j];

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

ShortestPath1(&G, B, F, n, L[t]);

b=L[t+1];

printf("%d\n", F[b]);

t=t+2;

}

return 0;

}

数据结构作业六：

**Problem A**

**深度优先搜索顶点之间的路径（邻接表）**

**时限：1000ms 内存限制：10000K 总时限：3000ms**

描述：

    给出一个有向图（节点个数<=100），试利用邻接表方式存储，然后利用图的深度优先搜索，判断是否存在由顶点vi到顶点vj的路径（i != j）。注意：有多条路径需要判断，每个结果占一行，最后要输出一个回车符。

输入：

从1开始表示第一个节点。  
第一行输入: 有向图的边数n，测例的个数m。  
之后n行输入：用来描述边，如2 4表示存在一条由顶点2到4的边。  
之后是m行输入：用来给出要测的路径，如2 5表示是否存在由顶点2到顶点5的路径。

输出：

对于每个测试例子输出一个结果并占一行。  
输出 Y表示源点和终点之间没有路径  
输出 N表示源点和终点之间存在路径

输入样例：

3 2  
1 2  
2 3  
4 3  
1 4  
1 3

输出样例：

N  
Y

#include<stdio.h>//迪杰斯特拉算法

#define MaxVertexNum 100 //最大的顶点个数

typedef char VertexType; //顶点类型

typedef int EdgeType; //边的权值类型

typedef struct{

VertexType vexs[MaxVertexNum];//顶点表

EdgeType edges[MaxVertexNum][MaxVertexNum];//邻接矩阵

int n, e; //实际的顶点数和边数

}MGraph;

int ShortestPath1(MGraph \*G, int P[], int D[], int N, int e){//Dijkstra算法

int min, i, j, k;

int final[10]={1,0}; //D[v]存放从0顶点到终点v的最短路径的长度；P[v]存放相应的最短路径终点的前驱点；INFINITY为邻接矩阵中的无穷大

for(i=1;i<=N;i++){ //final[N]用来表示集合S;final[j]=0表示顶点j未进入集合S，final[j]=1表示顶点j进入集合S。

D[i]=G->edges[e][i]; //P[5]=3说明v5的最短路径最后一段是从v3到达的

P[i]=0;

}

D[e]=0;

P[e]=-1;

final[1]=0;

final[e]=1;

for(i=1;i<N;i++){

min=10000+1; //min=INFINITY+1;

for(k=1;k<=N;k++)

if(final[k]==0 && D[k]<min){

j=k;

min=D[k];

}

final[j]=1;

for(k=1;k<=N;k++)

if(final[k]==0 && (D[j]+G->edges[j][k]<D[k])){

D[k]=D[j]+G->edges[j][k];

P[k]=j;

}

}

return 1;

}

int main(){

MGraph G;

int n, m, i, j, t=0, A[100], B[100], C[100], D[100], F[100], E[100];

scanf("%d%d", &n, &m);

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

scanf("%d%d", &A[i], &B[i]);

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

scanf("%d%d", &C[i], &D[i]);

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

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

for(t=1;t<=n;t++){

if(i==A[t] && j==B[t]){

G.edges[i][j]=1;

break;

}

else if(t==n)

G.edges[i][j]=10000;

}

for(i=1;i<=m;i++){

ShortestPath1(&G, E, F, n+1, C[i]);

if(F[D[i]]<10000)

printf("Y\n");

else printf("N\n");

}

return 0;

}

### Problem B

### 广度优先搜索顶点之间的路径（邻接表）

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

  给出一个有向图（节点个数<=100），试利用邻接表方式存储，然后利用图的广度优先搜索，判断是否存在由顶点vi到顶点vj的路径（i不等于j）。有多条路径需要判断，每个结果占一行，最后要输出一个回车符.

输入：

从1开始表示第一个节点。  
第一行输入: 有向图的边数n，测例的个数m。  
之后n行输入：用来描述边，如2 4表示存在一条由顶点2到4的边。  
之后是m行输入：用来给出要测的路径，如2 5表示是否存在由顶点2到顶点5的路径。

输出：

对于每个测试例子输出一个结果并占一行。  
输出 Y表示源点和终点之间没有路径  
输出 N表示源点和终点之间存在路径

输入样例：

3 2  
1 2  
2 3  
4 3  
1 4  
1 3

输出样例：

N  
Y

### Problem C

### 逆波兰表达式（有向无环图）

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

    一个四则运算算术表达式，只包含“（”，“）”，“+”，“-”，“\*”,“/”，括号可嵌套，利用有向无环图的邻接表达式存储，每个操作数原子都由一个小写字母表示。写一个算法输出其逆波兰表达式（又称后缀表达式）。

输入：

输入长度不超过100的四则运算表达式，中间没有空格,以’#’结尾。

输出：

输出表达式对应的逆波兰表达式

输入样例：

(a+b)\*c

输出样例：

ab+c\*

#include<stdio.h>

int main(){

int i=0, j=0, t=1, a=0, b=0, c=0, d=0, e=0;

char A[100], B[100], C[100];

while((A[i]=getchar())!='#') i++;

//for(j=0;j<i;j++){

if(A[j]=='('){

while(A[t]!=')'){

if(){

B[a]=A[t];

a++;

}

else{

C[b]=A[t];

b++;

}

t++;

}

for(){

printf("%c", );

}

for(){

}

}

else {

while(A[t]!='\*' && A[t]!='/'){

if(){

B[a]=A[t];

a++;

}

else{

C[b]=A[t];

b++;

}

}

for(){

printf("%c", );

}

for(){

}

### Problem D

### 最短路径的Dijkstra算法（邻接表）

#### 时限：1000ms 内存限制：10000K 总时限：3000ms

描述：

    以邻接表作为存储结构实现，求解从给定源点到给定结束点的最短路径。

输入：

从1开始表示第一个节点。  
第一行输入：顶点数n(2<=n<=100),边数m(2<=m<=100)  
第二行输入有向边：起始点s1，结束点 s2，边权值 w  
第三行输入：源点start，终点end

输出：

若存在路径，输出路径长度；  
若不存在，输出-1。

输入样例：

6 8  
1 6 100  
1 5 30  
1 3 10  
2 3 5  
3 4 50  
4 6 10  
5 4 20  
5 6 60  
1 6

输出样例：

60

#include<stdio.h>//迪杰斯特拉算法

#define MaxVertexNum 100 //最大的顶点个数

typedef char VertexType; //顶点类型

typedef int EdgeType; //边的权值类型

typedef struct{

VertexType vexs[MaxVertexNum];//顶点表

EdgeType edges[MaxVertexNum][MaxVertexNum];//邻接矩阵

int n, e; //实际的顶点数和边数

}MGraph;

int ShortestPath1(MGraph \*G, int P[], int D[], int N, int e){//Dijkstra算法

int min, i, j, k;

int final[100]={1,0}; //D[v]存放从0顶点到终点v的最短路径的长度；P[v]存放相应的最短路径终点的前驱点；INFINITY为邻接矩阵中的无穷大

for(i=1;i<=N;i++){ //final[N]用来表示集合S;final[j]=0表示顶点j未进入集合S，final[j]=1表示顶点j进入集合S。

D[i]=G->edges[e][i]; //P[5]=3说明v5的最短路径最后一段是从v3到达的

P[i]=0;

}

D[e]=0;

P[e]=-1;

final[1]=0;

final[e]=1;

for(i=1;i<N;i++){

min=10000+1; //min=INFINITY+1;

for(k=1;k<=N;k++)

if(final[k]==0 && D[k]<min){

j=k;

min=D[k];

}

final[j]=1;

for(k=1;k<=N;k++)

if(final[k]==0 && (D[j]+G->edges[j][k]<D[k])){

D[k]=D[j]+G->edges[j][k];

P[k]=j;

}

}

return 1;

}

int main(){

MGraph G;

int n, m, i, j, s1, s2, t=0, A[100], B[100], C[100], D[100], F[100];

scanf("%d%d", &n, &m);

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

scanf("%d%d%d", &A[i], &B[i], &C[i]);

scanf("%d%d", &s1, &s2);

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

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

for(t=1;t<=m;t++){

if(i==A[t] && j==B[t]){

G.edges[i][j]=C[t];

break;

}

else

G.edges[i][j]=10000;

}

ShortestPath1(&G, D, F, n, s1);

if(F[s2]<10000)

printf("%d\n", F[s2]);

else printf("%d\n", -1);

return 0;

}

typedef struct ANode{//弧的节点结构类型

int adjvex;

struct ANode \*nextarc;//指向下一条弧的指针

InfoType info;//该弧的相关信息

}ArcNode;

typedef struct Vnode{//领接表的头结点的类型

Vertex data;

ArcNode \*firstarc;//指向下一条弧

}VNode;

typedef VNode AdjList[MAXV];//AdjList是领接表类型

typedef struct{

AdjList adjlist;//领接表

int n, e;//顶点数， 边数

}ALGraph;

//shendu

void DFS(ALGraph \*G, int v){

ArcNode \*p;

visited[v]=1;//置已访问标记

printf("%d ", v);//输出被访问顶点的编号

p=G->adjlist[v].firstarc;//p指向顶点的第一条弧的弧头结点

while(p!=NULL){

if(visited[p->adjvex]==0)

DFS(G, p->adjvex);

p=p->nextarc;

}

}//输出结果为从v结点进行遍历直到遍历完所有结点

//guangdu

void BFS(ALGraph \*G, int v){

ArcNode \*p;

int queue[MAXV], front=0, rear=0;

int visited[MAXV];//定义存放结点的访问标志的数组

int w, i;

for(i=0;i<G->n;i++)//访问标志的数组的初始化

visited[i]=0;

printf("%2d", v);//输出访问节点的编号

visited[v]=1;//置已访问标记

rear=(rear+1)%MAXV;

queue[rear]=v;//v进队

while(front!=rear){

front=(front+1)%MAXV;

w=queue[front];

p=G->adjlist[w].firstarc;

while(p!=NULL){

if(visited[p->adjvex]==0)//若当前领接顶点未被访问

{

printf("%2d", p->adjvex);//访问相领结点

visited[p->adjvex]=1;//置该结点已被访问的标志

rear=(rear+1)%MAXV;//该顶点进队

queue[rear]=p->adjvex;

}

p=p->nextarc;

}

}

printf("\n");

}

#include<stdio.h>

#define MAXV 100

typedef int InfoType;

typedef char Vertex;

typedef struct ANode{//弧的节点结构类型

int adjvex;

struct ANode \*nextarc;//指向下一条弧的指针

InfoType info;//该弧的相关信息

}ArcNode;

typedef struct Vnode{//领接表的头结点的类型

Vertex data;

ArcNode \*firstarc;//指向下一条弧

}VNode;

typedef VNode AdjList[MAXV];//AdjList是领接表类型

typedef struct{

AdjList adjlist;//领接表

int n, e;//顶点数， 边数

}ALGraph;

int j=0;

//shendu

void DFS(ALGraph \*G, int v, int F[]){

ArcNode \*p;

int visited[100];

visited[v]=1;//置已访问标记

F[j]=v;

j++;

//printf("%d ", v);//输出被访问顶点的编号

p=G->adjlist[v].firstarc;//p指向顶点的第一条弧的弧头结点

while(p!=NULL){

if(visited[p->adjvex]==0)

DFS(G, p->adjvex, F);

p=p->nextarc;

}

}//输出结果为从v结点进行遍历直到遍历完所有结点

/\*//guangdu

void BFS(ALGraph \*G, int v){

ArcNode \*p;

int queue[MAXV], front=0, rear=0;

int visited[MAXV];//定义存放结点的访问标志的数组

int w, i;

for(i=0;i<G->n;i++)//访问标志的数组的初始化

visited[i]=0;

printf("%2d", v);//输出访问节点的编号

visited[v]=1;//置已访问标记

rear=(rear+1)%MAXV;

queue[rear]=v;//v进队

while(front!=rear){

front=(front+1)%MAXV;

w=queue[front];

p=G->adjlist[w].firstarc;

while(p!=NULL){

if(visited[p->adjvex]==0)//若当前领接顶点未被访问

{

printf("%2d", p->adjvex);//访问相领结点

visited[p->adjvex]=1;//置该结点已被访问的标志

rear=(rear+1)%MAXV;//该顶点进队

queue[rear]=p->adjvex;

}

p=p->nextarc;

}

}

printf("\n");

}\*/

int main(){

int n, m, i, t, A[100], B[100], C[100], D[100], E[100];

ALGraph \*G;

ArcNode \*p;

scanf("%d%d", &n, &m);

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

scanf("%d%d", &A[i], &B[i]);

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

scanf("%d%d", &C[i], &D[i]);

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

p=G->adjlist[A[i]].firstarc;

p->adjvex=B[i];

}

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

DFS(G, C[i], E);

for(t=0;t<100;t++)

if(E[t]==D[i]){

printf("Y");

break;

}

}

return 0;

}