**（计算二项式系数）写一个迭代算法计算二项式系数：c(n,m)=c(n-1,m)+c(n-1,m-1)。**

Input

二个正整数，分别表示n和m(0<m<=n<35)。

Output

一个整数，表示二项式系数。

#include<iostream>

using namespace std;

int func(int n,int m);

int main()

{

int n,m;

cin>>n>>m;

cout<<func(n,m)<<endl;

return 0;

}

int func(int n,int m)

{

if(m==0||m==n) return 1;

else return func(n-1,m-1)+func(n-1,m);

}

**（字符查找）给定一个字符串S和一个字符x。编写递归程序实现查找功能。如果x在S中，输出其在S中出现的次数，否则输出Notpresent**

Input

第一行，一个字符串，表示S，串长不超过80；第二行一个字符，表示x。

Output

如果x在S中，输出一个整数，表示x在S中出现的次数，否则输出Notpresent。

#include<stdio.h>

int GetNum(char \*s,char x);

int main()

{

char s[100];

char x;

int num=0;

gets(s);

scanf("%c",&x);

num=GetNum(s,x);

if(num==0)

printf("Notpresent");

else

printf("%d",num);

return 0;

}

int GetNum(char \*s,char x)

{

int count=0;

if(\*s)

{

if((\*s)==x)

count++;

count+=GetNum(s+1,x);

return count;

}

else{

return count;

}

}

**（数的因子之和问题）编写一个递归程序求解下列问题：给定正整数n，确定n是否是它所有因子（不包括n）之和。**

Input

正整数n

Output

若n是它所有因子之和，输出Yes，否则输出No

#include <stdio.h>

int func(int n);

int main()

{

int n,sum;

scanf("%d",&n);

sum=func(n);

if(n==sum)

printf("Yes\n");

else

printf("No\n");

return 0;

}

int func(int n)

{

int i;

int sum=0;

for(i=2;i<n;i++)

{

if(n%i==0)

{

sum+=i;

}

}

return ++sum;

}

**（集合的幂集）设S是有n（n≤20）个元素的集合，S的幂集是S所有可能的子集组成的集合。例如，S={a,b,c},则S的幂集={()(c)(b)(bc)(a)(ac)(ab)(abc)}。写一个递归程序，以S为输入，输出S的幂集。**

Input

n（n≤20）的值以及S的n个元素

Output

S的幂集

#include <iostream>

#include <cstring>

using namespace std;

void func(int i,char \*temp);

char s[21];

int main()

{

int n;

char temp[21]={0};

cin>>n>>s;

func(0,temp);

return 0;

}

void func(int i,char \*temp)

{

char t[21];

strcpy(t,temp);

if(i==strlen(s))

{

cout<<"("<<temp<<")";

return ;

}

else

{

func(i+1,t);

strncat(t,(s+i),1);

func(i+1,t);

}

}

**（最小和数）设两个正整数的最大公约数是 G，最小公倍数是 L，那么它们的和最小是多少？请编程解决这个问题**。

Input

两个不大于 10000 的正整数 G 和 L，中间用单个空格隔开。数据保证 L是 G 的倍数。

Output

一个正整数，即最小的和

#include<stdio.h>

int main(){

int x,y,i,j,min=100000;

scanf("%d%d",&x,&y);

if(x==y){

printf("%d",x+y);

return 0;

}

for(i=x;i<=y;i=i+x)

{

for(j=x;j<=y;j+=x)

{

if(i==j||(i\*j/x!=y))

continue;

if(i+j<min)

min=i+j;

}

}

printf("%d",min);

return 0;

}

**（整数质因素分解）每一个大于1的整数，或者自己是素数，或者是素数的乘积。比如12=2\*2\*3，11=11。请编写程序验证这个定理。**

Input

一个整数n，1<x<=1000000。

Output

一个素数或者一组素数的乘积式。

#include<stdio.h>

int main()

{

int x,i,flag,j=0,count=0,str[50];

scanf("%d",&x);

for(i=2;i<x;i++)

{

if(x%i==0)

break;

}

if(i>=x)

flag=1;

else

flag=0;

if(flag==1)

printf("%d",x);

else{

while(x!=1){

for(i=2;i<=x;i++)

{

if(x%i==0){

str[j]=i;

j++;

count++;

x=x/i;

break;

}

}

}

}

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

{

if(i==count-1)

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

else

printf("%d\*",str[i]);

}

return 0;

}

**（多项式计算）请设计一个算法，对于给定的变量x和各系数的值，求多项式anxn + an-1xn-1 + … + a1x + a0的值。假设x的值和各系数的值均为整型。注意：不能使用系统pow函数。**

Input

第一行为n和x的值，第二行输入an,an-1,...,a0共n+1个系数的值。其中1≤n≤100000。

Output

此多项式的值。

（测试数据保证结果用整型能表示）

#include<stdio.h>

int s(int a[],int n,int x){

int cout=1;

int sum=0;

for(int i=x;cout<n+1;i=i\*x,cout++){

sum=sum+a[cout]\*i;

}

sum=sum+a[0];

return sum;

}

int main(){

int n,x;

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

int a[10000];

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

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

}

int res=s(a,n,x);

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

return 0;

}

**（有重复元素的排列问题）设R={r1,r2,...,rn}是要进行排列的n（<10）个元素，其中元素r1,r2,...,rn可能相同。请编写程序，计算R中所有元素的不同排列数。**

Input

一个序列，表示R中的n个元素。

Output

一个正整数，表示R中所有元素的不同排列数。

#include<stdio.h>

#include<iostream>

#include<algorithm>

#include<string.h>

using namespace std;

int func(int n);

void funb();

char str[10000];

int main(){

funb();

return 0;

}

int func(int n){

if(n==0)

return 0;

if(n==1)

return 1;

return n\*func(n-1);

}

void funb(){

cin>>str;

int ans=0;

int count[30]={0};

int n=strlen(str),i;

sort(str,str+n);

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

{

count[str[i]-96]++;

}

ans=func(n);

for(i=0;i<=26;i++){

if(count[i])

ans/=func(count[i]);

}

cout<<ans<<endl;

}

**（栈的问题）栈是一种重要的数据结构，其主要的操作包括入栈和出栈。请编写程序，对任意给定的n，输出1，2，…，n的所有出栈顺序。**

Input

一个正整数，表示n（1≤n≤10）的值。

Output

输出1，2，…，n的所有出栈顺序，每个数字后有一个空格

#include<iostream>

using namespace std;

int ams[100000][20];

int length,temp;

bool judge(int \*test);

void ifTrue(int \*stack,int start,int end);

int main(){

int stack[20],i,j;

cin>>length;

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

{

stack[i]=i+1;

}

ifTrue(stack,0,length-1);

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

{

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

cout<<ams[i][j]<<" ";

cout<<endl;

}

return 0;

}

bool judge(int \*test){

int i,j,k;

for (i=0;i<length-2;i++)

{

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

{

for (k=j+1;k<length;k++)

{

if ((test[j]<test[k])&&(test[k]<test[i]))

{

return false;

}

}

}

}

return true;

}

void ifTrue(int \*stack,int start,int end)

{

int test[20],i;

if(start==end){

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

test[i] = stack[i];

}

if (judge(test))

{

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

{

ams[temp][i]=test[i];

}

temp++;

}

}

else{

for(i=start;i<=end;i++){

int temp=stack[start];

stack[start]=stack[i];

stack[i]=temp;

ifTrue(stack,start+1,end);

temp=stack[start];

stack[start]=stack[i];

stack[i]=temp;

}

}

}

**（伸展树操作）请编程实现伸展树的Insert（插入）和Find（查找）算法。例如，依次插入关键字10、50、40、30、20和60，并以缩进的方式显示得到的伸展树（其中，60是根，它的左孩子是30，右孩子是空，--表示空指针；30的左孩子是20，右孩子是50……）**

Sample Input

6

10 50 40 30 20 60

40

Sample Output

40

30

20

10

--

--

--

--

60

50

--

--

--

#include<iostream>

using namespace std;

enum ResultCode{Underflow,Overflow,Success,Duplicate,Fail,NotPresent};

struct BTNode{

int element;

BTNode \*lChild,\*rChild;

};

void LRot(BTNode \*&p){

BTNode \*r=p->rChild;

p->rChild=r->lChild;

r->lChild=p;

p=r;

}

void RRot(BTNode \*&p){

BTNode \*r=p->lChild;

p->lChild=r->rChild;

r->rChild=p;

p=r;

}

ResultCode Insert(BTNode \*&p,int x)

{

BTNode \*r;

ResultCode result=Success;

if(p==NULL)

{

p=new BTNode;

p->element=x;

p->lChild=NULL;

p->rChild=NULL;

return result;

}

if(x==p->element)

{

result=Duplicate;

return result;

}

if(x<p->element)

{

r=p->lChild;

if(r==NULL)

{

r=new BTNode;

r->element=x;

r->lChild=NULL;

r->rChild=NULL;

r->rChild=p;

p=r;

return result;

}

if(x==r->element)

{

RRot(p);

result=Duplicate;

return result;

}

if(x<r->element)

{

result=Insert(r->lChild,x);

RRot(p);

}

else{

result=Insert(r->rChild,x);

LRot(r);

p->lChild=r;

}

RRot(p);

}

else{

r=p->rChild;

if(r==NULL)

{

r=new BTNode;

r->element=x;

r->lChild=NULL;

r->rChild=NULL;

r->lChild=p;p=r;

return result;

}

if(x==r->element)

{

result=Duplicate;

LRot(p);

return result;

}

if(x>r->element)

{

result=Insert(r->rChild,x);

LRot(p);

}

else

{

result=Insert(r->lChild,x);

RRot(r);

p->rChild=r;

}

LRot(p);

}

return result;

}

void ShowSPTree(BTNode \*p,int pos)

{

int i;

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

cout<<" ";

if(p!=NULL)

{

cout<<p->element<<endl;

ShowSPTree(p->lChild,pos+5);

ShowSPTree(p->rChild,pos+5);

}

else

{

cout<<"--"<<endl;

}

}

int main(){

int n,i,x,key;

BTNode \*root=NULL;

cin>>n;

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

{

cin>>x;

Insert(root,x);

}

cin>>key;

Insert(root,key);

ShowSPTree(root,0);

return 0;

}

***基本搜索和遍历方法***

**（关节点问题）对于一个给定的连通图G，采用深度优先搜索的方法，识别出G的所有关节点。**

Input

第一行输入结点个数n和边的个数m，下面m行输入各边。

Output

关节点个数

#include <iostream>

using namespace std;

struct ENode{

int adjVex;

ENode \*nextArc;

};

ENode \*\*a;

void DFS\_low(int u,int p);

void resultfunc(int n);

int numb=0,t=0,d[100],low[100],sum=0;

int parentt[100];

int main(){

int i,n,e,u,v;

ENode \*t;

cin>>n>>e;

a=new ENode\*[n];

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

a[i]=NULL;

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

cin>>u>>v;

t=new ENode;

t->adjVex=v;

t->nextArc=a[u];

a[u]=t;

t=new ENode;

t->adjVex=u;

t->nextArc=a[v];

a[v]=t;

}

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

d[i]=-1;

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

parentt[i]=-1;

DFS\_low(0,-1);

if(numb>=2)

sum++;

resultfunc(n);

cout<<sum<<endl;

return 0;

}

void DFS\_low(int u,int p){

if(p==-1)

numb++;

ENode \*w;

low[u]=d[u]=t++;

for(w=a[u];w;w=w->nextArc){

int v=w->adjVex;

if(d[v]==-1)

{

DFS\_low(v,u);

if(low[u]>low[v])

low[u]=low[v];

}

else if(v!=p&&low[u]>d[v])

low[u]=d[v];

else if(v==p)

parentt[u]=v;

}

}

void resultfunc(int n){

int i,j;

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

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

if(parentt[j]==i){

if(low[j]>=d[i]){

sum++;

break;

}

}

}

}

}

***分治法***

**（查找最大和次大元素）对于给定的含有n个不同元素的无序序列，请用分治法求这个序列中最大和次大的两个元素。（注：要求不能用排序）**

Input

第一行输入n的值 ，第二行输入n个元素的值。

Output

最大元素和次大元素

#include<iostream>

using namespace std;

int n;

int a[100000];

void MaxMinfunc(int i,int j,int &max,int &min);

int main()

{

int i;

int max,min;

cin>>n;

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

cin>>a[i];

}

MaxMinfunc(0,n-1,max,min);

cout<<max;

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

if(a[i]==max){

break;

}

}

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

a[i]=a[i+1];

}

n--;

MaxMinfunc(0,n-1,max,min);

cout<<" "<<max;

return 0;

}

void MaxMinfunc(int i,int j,int &max,int &min)

{

int maxr,minr,m;

if(i==j){

max=a[i];

min=a[i];

}

else if(i==j-1){

if(a[i]<a[j]){

max=a[j];

min=a[i];

}

else{

max=a[i];

min=a[j];

}

}

else{

m=(i+j)/2;

MaxMinfunc(i,m,max,min);

MaxMinfunc(m+1,j,maxr,minr);

if(max<maxr)

max=maxr;

if(min>minr)

min=minr;

}

}

**（三分搜索）三分搜索算法的做法是：在从小到大的序列中，先将待查元素x与n/3处的元素比较，然后将x与2n/3处的元素进行比较。比较的结果或者找到x，或者将搜索范围缩小到原来的n/3。**

Input

第一行，一个整数，表示序列元素的个数n；第二行，n个整数，表示序列的n个元素；第三行，一个整数，表示待查找的元素x。

Output

一个整数，如果查找成功，输出元素的下标，否则输出-1。

#include <cstdio>

int a[1000], n;

int ThSearch(int l,int r,int x);

int main()

{

int x,tempx,i;

scanf("%d",&n);

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

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

scanf("%d", &x);

tempx=ThSearch(0,n-1,x);

printf("%d",tempx);

return 0;

}

int ThSearch(int p,int r,int x)

{

int m1,m2;

if(p<=r)

{

m1=p+(r-p)/3;

m2=p+(r-p)\*2/3;

if(a[m2]<x)

return ThSearch(m2+1,r,x);

else if(a[m1]<x&&a[m2]>x)

return ThSearch(m1+1,m2-1,x);

else if(a[m1]>x)

return ThSearch(p,m1-1,x);

else if(a[m1]==x)

return m1;

else if(a[m2]==x)

return m2;

}

return -1;

}

**（快速排序）请用分治法实现快速排序算法，对n（0<n<=10^5）个元素按升序排序。**

Input

第一行一个正整数，表示n的值；第二行，有n个整数，表示待排序列。

Output

升序排列的n个整数，两个整数之间用一个空格分隔。

#include<iostream>

#include<algorithm>

using namespace std;

const int maxv=100010;

int l[maxv]={0};

int partition(int left,int right);

void Insertsort(int left,int right);

void Quicksort(int left,int right);

int main(){

int n,i;

cin>>n;

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

cin>>l[i];

l[n]=2147483647;

Quicksort(0,n-1);

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

cout<<l[i];

if(i!=n-1)cout<<" ";

}

cout<<endl;

return 0;

}

int partition(int left,int right){

int i=left,j,ran;

j=right+1;

ran= rand() % (right + 1 - left) + left;

swap(l[ran],l[left]);

do{

do i++;while(l[i]<l[left]);

do j--;while(l[j]>l[left]);

if(i<j)swap(l[i],l[j]);

}while(i<j);

swap(l[left],l[j]);

return j;

}

void Insertsort(int left,int right){

int n=right-left+1;

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

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

if(l[left+j]<l[left+j-1]){

swap(l[left+j],l[left+j-1]);

}

}

}

}

void Quicksort(int left,int right){

if(left<right){

int j = partition(left,right);

if(right-left<10){

Insertsort(left,right);

return;

}

Quicksort(left,j-1);

Quicksort(j+1,right);

}

}

**（第k小元素问题）用分治法编程解决在n(0<n<=1000)个数当中找第K小元素问题（注意：不能用排序）。**

Input

第一行输入n的值，第二行输入n个数，第三行输入K的值。

Output

n个数中的第K小元素。

#include<iostream>

using namespace std;

int L[100000], n;

int partition(int l,int r);

void selectFunc(int &x,int k);

int main()

{

int x,y;

cin>>n;

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

cin>>L[i];

cin>>y;

selectFunc(x,y);

cout<<x;

return 0;

}

int partition(int left,int right){

int i=left,temp;

int j=right+1;

do{

do i++; while(L[i]<L[left]);

do j--; while(L[j]>L[left]);

if(i<j){

int t=L[i];

L[i]=L[j];

L[j]=t;

}

}while(i<j);

temp=L[j];

L[j]=L[left];

L[left]=temp;

return j;

}

void selectFunc(int &x,int k){

int left,right;L[n]=0x7fffffff;

right=n;

left=0;

do{

int j=partition(left,right);

if(k==j+1){

x=L[j];

return ;

}

else if(k<j+1)

right=j;

else

left=j+1;

}while(1);

}

***贪心法***

**（部分背包问题）已知一个载重为M的背包和n(0<n<1000)件物品，第i件物品的重量为 wi，如果将第i件物品全部装入背包，将有收益pi，这里，wi>0，pi>0，1<=i<=n。所谓背包问题是指求一种最佳装载方案，使得收益最大。**

Input

第一行二个正整数，分别表示物品个数n和背包载重M，以下n行，每行三个整数，分别表示物品编号i，物品收益pi，物品重量wi。

Output

x1,x2,…,xn，0<=xi<=1，1<=i<=n，每个xi是第i件物品装入背包中的部分（小数位保留二位）。xi之间用一个空格分隔。

#include<iostream>

#include<algorithm>

#include<cstdio>

using namespace std;

int n,m;

struct Item{

int i;

int p;

int w;

double x;

};

int compareD(Item a,Item b);

int compareI(Item a,Item b);

int main()

{

int i,kp=0;

cin>>n>>m;

double u=m;

Item a[n];

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

cin>>a[i].i>>a[i].p>>a[i].w;

}

sort(a,a+n,compareD);

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

a[i].x=0;

}

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

if(a[i].w>u)

break;

a[i].x=1;

u=u-a[i].w;

}

if(i<n)

a[i].x=u/a[i].w;

sort(a,a+n,compareI);

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

if(kp==1){

cout<<" ";

}

else

kp=1;

printf("%.2f",a[i].x);

}

return 0;

}

int compareD(Item a,Item b)

{

return (double)a.p/a.w>(double)b.p/b.w;

}

int compareI(Item a,Item b)

{

return a.i<b.i;

}

**（带时限的作业排序问题）设有一个单机系统、无其它资源限制且每个作业运行相等时间，不妨假定每个作业运行1个单位时间。现有n(0<n<1000)个作业，每个作业都有一个截止期限di>0，di为整数，1<=i<=n。如果作业能够在截止期限之内完成，可获得pi>0的收益。问题要求得到一种作业调度方案，该方案给出作业的一个子集和该作业子集的一种排列，使得若按照这种排列次序调度作业运行，该子集中的每个作业都能如期完成，并且能够获得最大收益。**

Input

第一行一个正整数，表示n的值，以下n行，每行三个整数，分别表示作业号i，收益pi，截止期限di。

Output

n个作业的一个最优子集。作业号从小到大输出，之间用一个空格分隔。

#include<iostream>

#include<stdlib.h>

using namespace std;

typedef struct J

{

int number;

int p;

int d;

}J;

int n;

J \*d;

J \*x;

void sort()

{

int i,j;

J temp;

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

{

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

{

if(d[j].p>=d[i].p)

{

temp=d[i];

d[i]=d[j];

d[j]=temp;

}

}

}

}

void JS()

{

int k=0,i,j,r;

J temp;

x[0]=d[0];

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

{

r=k;

while(r>=0&&x[r].d>d[j].d&&x[r].d>r+1)

r--;

if((r<0||x[r].d<=d[j].d)&&d[j].d>r+1)

{

for(i=k;i>=r+1;i--) {

x[i+1]=x[i];

}

x[r+1]=d[j];

k++;

}

}

for(i=0;i<k;i++) //将x数组按照作业号升序排列

{

for(j=i+1;j<=k;j++)

{

if(x[j].number<x[i].number)

{

temp=x[i];

x[i]=x[j];

x[j]=temp;

}

}

}

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

cout<<x[i].number<<" ";

cout<<x[k].number<<endl;

}

int main()

{

int i;

cin>>n;

d=(J \*)malloc(n\*sizeof(J));

x=(J \*)malloc(n\*sizeof(J));

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

{

J job;

cin>>job.number;

cin>>job.p;

cin>>job.d;

d[i]=job;

}

sort();

JS();

free(d);

free(x);

return 0;

}

**（最小代价生成树）一个无向连通图的生成树是一个极小连通子图，它包括图中全部结点，并且有尽可能少的边。一棵生成树的代价是树中各条边上的代价之和。一个网络的各生成树中，具有最小代价的生成树称为该网络的最小代价生成树（minimum-cost spanning tree）。**

Input

第一行输入结点个数n(n<100)和边的个数m，以下m行输入各边的两个结点u、v及该边上的代价。

Output

如果有生成树，则输出最小生成树的代价；如果没有生成树，则输出"no spanning tree"。

#include<iostream>

#include <stdio.h>

#include <string.h>

#include <algorithm>

#define MAXN 100

#define MAXM 100

using namespace std;

struct Enode

{

int u, v, w; //边的两个顶点、权值

}Enodes[MAXM]; //边的数组

int parent[MAXN];

int n,m;

void UFset( );

int Find(int x);

bool cmp(Enode a,Enode b);

void Union(int R1,int R2);

void Kruskal( );

int main( )

{

int i,u,v,w;

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

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

{

scanf("%d%d%d",&u,&v,&w); //读入边的起点和终点

Enodes[i].u=u;Enodes[i].v=v;Enodes[i].w=w;

}

sort(Enodes,Enodes+m,cmp);

Kruskal();

return 0;

}

void UFset( ) //初始化

{

int i;

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

parent[i]=-1;

}

int Find(int x)

{

int s,t;

for(s=x;parent[s]>=0;s=parent[s]);

while(s!=x)

{

t=parent[x];

parent[x]=s;

x=t;

}

return s;

}

void Union(int R1,int R2)

{

int r1,r2,t;

r1=Find(R1);

r2=Find(R2);

t=parent[r1]+parent[r2];

if( parent[r1]>parent[r2] )

{

parent[r1]=r2;

parent[r2]=t;

}

else

{

parent[r2]=r1;

parent[r1]=t;

}

}

bool cmp(Enode a,Enode b)

{

return a.w<=b.w;

}

void Kruskal( )

{

int i,sum=0, num=0;

int u,v; //选用边的两个顶点

UFset( ); //并查集

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

{

u=Enodes[i].u;

v=Enodes[i].v;

if(Find(u)!=Find(v))

{

sum=sum+Enodes[i].w;

num++;

Union(u,v);

}

if(num>=n-1)

break;

}

if(num==n-1)

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

else

printf("no spanning tree");

}

**（乘船问题）有n(0<n<1000)个人，第i个人体重为wi(0<=i<n)，每艘船的最大载重量均为C，且最多只能乘两个人，试用最少的船装载所有人。**

Input

第一行，一个正整数，表示n的值；第二行是n个整数，表示n个人的体重；第三行，一个正整数，表示船的最大载重量C的值。

Output

一个整数，表示最少的船数。

#include<iostream>

using namespace std;

int main(){

int n,c,a[10000],ans=0,i,j;

cin>>n;

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

{

cin>>a[i];

}

cin>>c;

i=0,j=n-1;

while(i<j)

{

if(a[i]+a[j]<=c) {

ans++;

i++;

j--;

}

else {

ans++;

j--;

}

}

if(i==j)

cout<<ans+1;

else

cout<<ans;

return 0;

}

***动态规划法***

**（多段图问题）多段图G=(V,E)是一个带权有向图，它具有如下特性：图中的结点被划分成k>=2个互不相交的子集Vi，1<=i<=k。其中V1和Vk分别只有一个结点，V1包含源点（source）s，Vk包含汇点（sink）t。对所有边属于E，多段图要求若u属于Vi，则v属于Vi＋1，1<=i< k，每条边的权值为c(u,v)。从s到t的路径长度是这条路径上边的权值之和，多段图问题（multistage graph problem）是求从s到t的一条长度最短的路径。**

Input

第一行输入结点个数n和边的个数m，以下m行输入各有向边的两个结点u、v及该边上的代价。

Output

从s到t的最短的路径的长度。

#include<iostream>

#include<limits.h>

using namespace std;

struct Enode

{//邻接表

int adjVex;

int w;

Enode \*nextArc;

};

int n,m; //n为有向图的结点个数，m为边的个数

Enode \*\*a; //指向邻接表的指针

void ljtable()

{//建立有向图的邻接表

int u,v,i,weight;

Enode \*t;

cin>>n>>m;

a=new Enode\*[n];

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

a[i]=NULL;

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

{

cin>>u>>v>>weight;

t=new Enode;

t->adjVex=v;

t->nextArc=a[u];

t->w=weight;

a[u]=t;

}

}

int FMultiGraph()

{//多段图的向前递推算法

float c,\*cost=new float[n];

int v,j,q,\*d=new int[n];

Enode \*r;

cost[n-1]=0;

d[n-1]=-1;

for(j=n-2;j>=0;j--)

{

float min=(float)INT\_MAX;

for(r=a[j];r;r=r->nextArc)

{

v=r->adjVex;

if(r->w+cost[v]<min)

{

min=r->w+cost[v];

q=v;

}

}

cost[j]=min;

d[j]=q;

}

c=cost[0];

delete []cost;

delete []d;

return c;

}

int main()

{

ljtable();

cout<<FMultiGraph()<<endl;

return 0;

}

**（游船租金问题）长江游艇俱乐部在长江上设置了n个游艇出租站1，2，…，n。游客可在这些游艇出租站租用游艇，并在下游的任何一个游艇出租站归还游艇。游艇出租站i到游艇出租站j之间的租金为r(i,j),1 <= i< j <= n 。试设计一个算法，计算从游艇出租站1到游艇出租站n所需的最少租金。**

Input

第 1 行中有 1 个正整数 n（n<=200） ，表示有 n个游艇出租站。

接下来的n-1 行是r(i,j), r(i,j)为整数，1 ≤ i < j ≤ n。

Output

输出从游艇出租站1到游艇出租站n所需的最少租金。

#include<iostream>

using namespace std;

int main()

{

int best[200][200]={0}; //记录最优解

int n,i,j,r,k,temp;

cin>>n;

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

{

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

{

cin>>best[i][j];

}

}

for(r=2;r<=n;r++)

{

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

{

j=i+r-1;

for(k=i;k<=j;k++)

{//找 k使r(i,k)+r(k,j)最小

temp=best[i][k]+best[k][j];

if(temp<best[i][j])

{

best[i][j]=temp;

}

}

}

}

cout<<best[1][n]<<endl;

return 0;

}

**（矩阵连乘问题）给定n（0<n<100）个矩阵{A0,A1,…,An-1}， 其中Ai，i=0,…,n-1的维数为pi\*pi+1，并且Ai与Ai+1是可乘的。考察这n个矩阵的连乘积A0A1…An-1，由于矩阵乘法满足结合律，所以计算矩阵的连乘可有许多不同的计算次序。矩阵连乘问题是确定计算矩阵连乘积的计算次序，使得按照这一次序计算矩阵连乘积，需要的“数乘”次数最少。**

Input

第一行输入n的值，第二行输入n个矩阵的维数pi（i=0，…，n）。

Output

最少计算量。

#include<stdio.h>

int p[100],m[100][100],s[100][100],n;

void MatrixChain()

{

int i,j;

int k,t,r;

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

m[i][i]=0;

for(r=2;r<=n;r++)

{

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

{

j=i+r-1;

m[i][j]=m[i+1][j]+p[i]\*p[i+1]\*p[j+1];

s[i][j]=i;

for(k=i+1;k<j;k++)

{

t=m[i][k]+m[k+1][j]+p[i]\*p[k+1]\*p[j+1];

if(t<m[i][j])

{

m[i][j]=t;

s[i][j]=k;

}

}

}

}

printf("%d\n",m[0][n-1]);

}

int main()

{

int i;

scanf("%d",&n);

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

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

MatrixChain();

return 0;

}

**（最长公共子序列）给定2个序列X={x1,x2,…,xm}和Y={y1,y2,…,yn}，其中0<m、n<1000。找出X和Y的最长公共子序列。**

Input

第一行输入序列X，第二行输入序列Y。

Output

X和Y的最长公共子序列的长度。

#include<stdio.h>

int c[100][100],s[100][100];

int m=0,n=0;

char x[100],y[100];

int LCSLength()

{

int i,j;

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

c[i][0]=0;

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

c[0][i]=0;

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

{

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

{

if(x[i-1]==y[j-1])

{

c[i][j]=c[i-1][j-1]+1;

s[i][j]=1;

}

else if(c[i-1][j]>=c[i][j-1])

{

c[i][j]=c[i-1][j];

s[i][j]=2;

}

else

{

c[i][j]=c[i][j-1];

s[i][j]=3;

}

}

}

return c[m][n];

}

int main()

{

int i;

scanf("%s",x);

scanf("%s",y);

for(i=0;x[i]!='\0';i++);

m=i;

for(i=0;y[i]!='\0';i++);

n=i;

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

return 0;

}

**（0/1背包问题）已知一个载重为M的背包和n件物品，物品编号从0到n-1。第i件物品的重量为 wi，若将第i种物品装入背包将获益pi，这里，wi>0，pi>0，0<=i<n。所谓0/1背包问题是指在物品不能分割，只能整件装入背包或不装入的情况下，求一种最佳装载方案使得总收益最大**。

Input

第 1 行中有 2 个正整数 n（n<=50）和M ，表示有 n件物品，背包载重为M(m<=100)。然后输入n个物品的重量，最后输入n个物品的收益值。

Output

最佳装载方案的总收益

#include<stdio.h>

int max(int a,int b)

{

return a>b?a:b;

}

int main()

{

int m,n,i,j,f[100][100],p[100],w[100];

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

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

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

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

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

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

f[0][i]=0;

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

f[i][0]=0;

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

{

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

{

if(j>=w[i-1])

f[i][j]=max(f[i-1][j-w[i-1]]+p[i-1],f[i-1][j]);

else

f[i][j]=f[i-1][j];

}

}

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

return 0;

}

**（最大上升子序列）计信学院门前摆了一排盆栽的绿植，高度都不一样，共有n（0<n<=100）盆。五一节到了，为了美化，现在要搬走其中的n-m盆，使得剩下的m盆绿植相对位置不变，但高度是递增的，而且留下的绿植盆数最多。**

Input

第一行一个整数，表示n的值；第二行是n个整数，分别表示每盆绿植的高度。

Output

符合条件的最多的绿植盆数。

#include<iostream>

using namespace std;

int main()

{

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

int num[200];//每个数的上升序列长度编号

int hmax=0;

cin>>n;

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

cin>>a[i];

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

{

num[i]=1;

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

{

if(a[j]<a[i]&&num[j]+1>num[i])//两者同时满足就是最大子序列编号

num[i]=num[j]+1;

}

}

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

if(hmax<num[i])

hmax=num[i];

cout<<hmax<<endl;

return 0;

}

***回溯法***

**（n-皇后）问题要求在一个n\*n的棋盘上放置n个皇后，使得它们彼此不受“攻击”。观察表明n-皇后问题的解存在垂直对偶性，请修改教材算法NQeens，令x[0]=1，2，……，[n/2]，使得只求其中不对称的那些解。**

Input

n(4<=n<=32)的值。

Output

不对称的那些解。

#include<iostream>

using namespace std;

int absFunc(int n)

{

if(n<0)

return -n;

else

return n;

}

bool Place(int k,int i,int \*x)

{

int j;

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

if((x[j]==i)||(absFunc(x[j]-i)==absFunc(j-k)))

return false;

return true;

}

void NQueens(int k,int n,int \*x)

{

int i,j;

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

{

if(Place(k,i,x))

{

x[k]=i;

if(k==n-1)

{

if(x[0]<n/2)

{

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

cout<<x[j]<<" ";

cout<<x[j]<<endl;

}

}

else

NQueens(k+1,n,x);

}

}

}

int main()

{

int n;

cin>>n;

int \*x=new int[n];

NQueens(0,n,x);

delete []x;

return 0;

}

**（子集和数问题）已知n(0<n<=20)个不同正整数wi，0<=i<=n-1，的集合，求该集合的所有满足条件的子集，使得每个子集中的正整数之和等于另一个给定的正整数M。**

Input

第一行输入n和M的值，第二行输入n个不同的正整数wi（i=0，…，n）。

Output

如果有答案，则输出所有满足条件的子集（用固定长度n-元组xi表示，xi之间用一个空格分隔，xi=0或1，i=0，…，n）。如果没有答案，则输出“no solution!”

#include<iostream>

using namespace std;

void SumofSub(int s,int k,int r,int w[],int x[],int n,int m);

int a[100],count=0;

int main()

{

int m,n,i,j,t,s=0,r=0;

cin>>n>>m;

int \*w=new int[n];

int \*x=new int[n];

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

{

cin>>w[i];

a[i]=w[i];

x[i]=0;

r+=w[i];

}

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

{

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

{

if(w[j]>w[j+1])

{

t=w[j];

w[j]=w[j+1];

w[j+1]=t;

}

}

}

if(s+r>=m)

SumofSub(s,0,r,w,x,n,m);

else

cout<<"no solution!"<<endl;

if(count==0)

cout<<"no solution!"<<endl;

delete[]w;

delete[]x;

return 0;

}

void SumofSub(int s,int k,int r,int w[],int x[],int n,int m)

{

int i,j,l;

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

{

x[k]=i;

s+=x[k]\*w[k];

r-=w[k];

if(s==m)

{

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

{

if(w[j]==a[j])

{

cout<<x[j]<<" ";

}

else

{

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

{

if(w[l]==a[j])

{

cout<<x[l]<<" ";

}

}

}

}

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

{

if(w[l]==a[j])

{

cout<<x[l]<<endl;

count++;

}

}

}

else if(s+r>=m&&s+w[k+1]<=m)

{

SumofSub(s,k+1,r,w,x,n,m);

}

s-=x[k]\*w[k];

r=r+w[k];

}

}

***期中考试***

**（奇偶序列问题）输入n个正整数，彼此以空格分隔。重新排序以后输出(也按空格分隔)，要求:**

**1.先输出其中的奇数，并按从大到小排列；2.然后输出其中的偶数，并按从小到大排列。**

Input

每一行输入一个正整数n(<1000)，表示序列中整数的个数；第二行输入n个整数正整数，彼此以空格分隔。

Output

按照要求排序后输出，每个数后带一个空格分隔。

#include<stdio.h>

#include <algorithm>

using namespace std;

int main()

{

int a[1000],ji[1000],ou[1000],n,i,j,b,c;

scanf("%d",&n);

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

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

sort(a,a+n);

j=0;

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

{

if(a[i]%2!=0)

{

ji[j]=a[i];

j++;

}

}

b=j;

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

{

if(a[i]%2==0)

{

ou[j]=a[i];

j++;

}

}

c=j;

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

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

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

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

return 0;

}

**（加法序列问题）有两个整数n和m（0<n<m<35）。现在在n上可以加上1，也可以加上2，不断地这样加，最后使得n正好等于m。求共有多少种不同的加法序列。例如：n=1，m=4，则有n+1+1+1=m，n+1+2=m，n+2+1=m，所以此时有3种不同的加法序列。**

Input

二个正整数，分别表示n和m的值。

Output

不同的加法序列数。

#include<stdio.h>

int mss,ans=0;

void xl(int a);

int main()

{

int num;

scanf("%d%d",&num,&mss);

xl(num);

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

return 0;

}

void xl(int t){

if(t>mss)

return;

else if(t==mss){

ans++;

return;

}

else{

xl(t+1);

xl(t+2);

}

}

**（数字三角形）给定一个由n(1<n<100)行数字组成的三角形，如下图所示。试设计一个算法，计算出从三角形的顶部至底部的一条路径，使得该路径经过的数字总和最大。每一步只能从一个数走到下一层上和它最近的左边的数或者右边的数。**

7

3 8

8 1 0

2 7 4 4

4 5 2 6 5

Input

第一行输入n的值，下面n行输入数字三角形中的数字。

Output

最大数字总和及其路径（路径中每个数字后带一个空格，测试数据保证路径唯一）。

#include<iostream>

#include <algorithm>

using namespace std;

int main()

{

int b[200][200],c[200],d[200];

int i,j;

int r,a[200][200];

cin>>r;

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

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

cin>>a[i][j];

for(i=r;i>0;i--)

{

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

{

b[i][j]=max(b[i+1][j],b[i+1][j+1])+a[i][j];

if(b[i+1][j]>b[i+1][j+1])

c[i]=1;

else if(b[i+1][j]<=b[i+1][j+1])

c[i]=2+2-4;

}

}

cout<<b[1][1];

cout<<endl;

j=2+3-4;

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

{

if(c[i]==1)

{

d[i]=a[i][j];

}

if(c[i]!=1)

{

d[i]=a[i][j++];

}

cout<<d[i];

cout<<" ";

}

return 0;

}

**（元素包含问题）给定n个集合A1，A2，…，An，每个集合都由连续的正整数构成，即Ai={x|ai≤x≤bi}，这里，ai，x，bi都是正整数，i=1，2，…,n。设计一个算法求最小的集合S，使得对每个i=1，2，…，n，|S∩Ai|≠Ø，即每个Ai至少含有S中的一个数。**

Input

第一行一个整数，表示n（1<=n<=100）的值；接下来有n行，每行有二个整数，分别表示ai和bi的值。

Output

最小的集合S中元素的个数。

#include<iostream>

#include<algorithm>

using namespace std;

struct dd{

double e;

double o;

};

dd p[200];

int compare(dd a,dd b);

int main(){

int i,temp,k=2,num;

cin>>num;

for(i=0;i<num;i++)//sr

{

cin>>p[i].e;

cin>>p[i].o;

}

sort(p,p+num,compare);

temp=p[0].o;

k=k-1;

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

{

if(temp<p[i].e)

{

k++;

temp=p[i].o;

}

}

cout<<k;

cout<<endl;

return 0;

}

int compare(dd nd,dd m)

{

if(nd.o>m.o)

return 0;

else

return 1;

}

**已知图G=(V,E)是一个n个结点的连通图。连通图G的一个哈密顿环(Hamiltonlan cycle)是图G的一个回路，它经过图中每个结点，且只经过一次。编写程序判定一个连通图是否存在哈密顿环。**

0-----1

| /

| /

| /

2

Input

第一行二个整数n和e（1<n，e<50），表示一个连通图的顶点的个数和边的个数；下面共e行，每行有二个整数u和v，表示一条边的两个顶点。

Output

如果存在哈密顿环，输出“True”，否则，输出“False”

#include<iostream>

#include<cstdio>

using namespace std;

int a[10][10];

int n,count=0;

void NextValue(int k,int \*x){

do{

int j;

x[k]=(x[k]+1)%n;

if(!x[k]) return;

if(a[x[k-1]][x[k]]){

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

if(x[k]==x[j])

break;

if(j==k)

if((k<n-1)||(k==n-1)&&a[x[n-1]][x[0]])

return;

}

}while(1);

}

void Hamiltonian(int k,int \*x){

do{

NextValue(k,x);

if(!x[k]) break;

if(k==n-1){

count++;

}

else Hamiltonian(k+1,x);

}while(1);

}

int main(){

int m,h,g;

cin>>n>>m;

int i,j;

int \*x=new int[n];

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

x[i]=0;

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

a[i][j]=0;

}

}

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

cin>>h>>g;

a[h][g]=1;

a[g][h]=1;

}

Hamiltonian(1,x);

if(count==0) cout<<"False";

else cout<<"True";

return 0;

}

***分枝限界法***

**（15谜判定问题）在一个4\*4的方形棋盘上放置了15块编了号的牌，还剩下一个空格。问题要求对于给定的一种初始排列，通过一系列的合法移动，将给定的初始排列转换成目标排列。应当注意的是，并非所有可能的排列作为初始排列都能到达目标排列。编写程序对任意给定的排列，判断是否可由该排列经过一系列号牌移动到达目标排列。**

Input

初始排列（空格用‘#’表示）。

Output

如果可由输入的初始排列经过一系列号牌移动到达目标排列，则输出“TRUE”；否则输出“FALSE”。

#include<iostream>

#include<string>

#include<cstdlib>

using namespace std;

int a[20];

string b[20];

int main()

{

int i,j;

int s=0;

for(i=1;i<17;i++)

{

cin>>b[i];

if(b[i]=="#")

{

b[i]="16";

s=s+i/4+1+i%4;

}

a[i]=atoi(b[i].c\_str());

}

for(i=1;i<17;i++)

{

for(j=i+1;j<=16;j++)

if(a[i]>a[j])

s++;

}

if(s%2)

cout<<"FALSE"<<endl;

else

cout<<"TRUE"<<endl;

return 0;

}

**（n皇后问题-分支限界法）在 n×n格的棋盘上放置彼此不受攻击的 n（3<n<11）个皇后。按照国际象棋的规则，皇后可以攻击与之处在同一行或同一列或同一斜线上的棋子。 n后问题等价于在 n×n格的棋盘上放置n个皇后，任何 2个皇后不放在同一行或同一列或同一斜线上。设计一个解 n 后问题的队列式分枝限界法，计算在 n×n个方格上放置彼此不受攻击的n个皇后的一个放置方案。**

Input

1 个正整数n。

Output

n个皇后的第一种放置方案（行和列号都从1开始，每个数字后带有一个空格）。

#include<iostream>

#include<queue>

#include<cmath>

#include<vector>

using namespace std;

struct Node{

int number;

vector<int>x;

};

int n;

int absfunc(int a);

bool Place(Node q,int n);

void Nqueen();

int main(){

cin>>n;

Nqueen();

return 0;

}

int absfunc(int a){

if(a<0) return -a;

return a;

}

bool Place(Node q,int n)

{

for(int j=0;j<n-1;j++)

if((absfunc(n-j-1)==absfunc(q.x[j]-q.x[n-1]))||(q.x[j]==q.x[n-1]))

return false;

return true;

}

void Nqueen()

{

int t=1;

Node Ew;

queue<Node>Q;

Node sign;

sign.number=-1;

Q.push(sign);

while(true){//检查所有的孩子节点

for(int k=1;k<=n;k++){

//把当前扩展节点的值赋给下一个节点

Node q;

q.number=t;

for(int i=0;i<t-1;i++) q.x.push\_back(Ew.x[i]);

q.x.push\_back(k);

if(Place(q,t))Q.push(q);

}

//取下一扩展节点赋值给Ew

Ew=Q.front();

Q.pop();

if(Ew.number==-1){

//同层节点尾部标记

t++;//进入下一层

Q.push(sign);//增加标记

//继续往下去下一个节点

Ew=Q.front();

Q.pop();

}

if(Ew.number==n){

for(int i=0;i<n;i++) cout<<Ew.x[i]<<" ";

break;

}

}

}

**（带时限的作业排序）问题可描述为：设有n个作业和一台处理机，每个作业所需的处理时间、要求的时限和收益可用三元组（pi, di, ti），0<=i<n表示，其中，作业i的所需时间为ti，如果作业i能够在时限di内完成，将可收益pi，求使得总收益最大的作业子集J。**

Input

第一行输入n（0<n<15）的值；第二行输入pi；第三行输入di；第四行输入ti （i=0，…，n-1且作业已经按时限非减次序排列）。

Output

xi（用固定长度n-元组xi表示，xi=0或1，i=0，…，n-1），每个xi后带有一个空格。

#include <iostream>

#include <queue>

using namespace std;

const double epsilon = 1e-6;

struct Node {

Node(Node\* par, int k)

{

parent = par;

j = k;

}

Node\* parent;

int j;

};

template<class T>

struct qNode {

qNode() {}

qNode(T p, T los, int sd, int k, Node\* pt)

{

prof = p; loss = los; d = sd; ptr = pt; j = k;

}

T prof, loss;

int j, d;

Node\* ptr;

};

template<class T>

class JS {

public:

JS(T\* prof, int\* de, int\* time, int size);

T JSFIFOBB();

void GenerateAns(int\* x, int& k);

private:

T\* p, total;

int\* t, \* d, n;

Node\* ans, \* root;

};

template<class T>

JS<T>::JS(T\* prof, int\* de, int\* time, int size) {

p = prof; d = de; t = time; n = size;

int i;

total = 0;

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

total += p[i];

}

}

template<class T>

void JS<T>::GenerateAns(int\* x, int& k) {

Node\* pt = ans;

while (pt->j > -1) {

x[pt->j] = 1;

pt = pt->parent;

}

}

template<class T>

T JS<T>::JSFIFOBB()

{

Node\* E, \* child;

queue<qNode<T> > q;

E = root = new Node(NULL, -1);

qNode<T> ep(0, 0, 0, -1, root), ec;

T U = total + epsilon;

while (1) {

T loss = ep.loss, prof = ep.prof; E = ep.ptr;

for (int j = ep.j + 1; j < n; j++) {

if (ep.d + t[j] <= d[j] && loss < U) {

child = new Node(E, j);

ec.prof = prof + p[j]; ec.d = ep.d + t[j];

ec.ptr = child; ec.loss = loss; ec.j = j;

q.push(ec);

T cost = total - ec.prof;

if (cost < U) {

U = cost; ans = child;

}

};

loss = loss + p[j];

}

do {

if (q.empty()) return total = U;

ep = q.front(); q.pop();

} while (ep.loss >= U);

}

}

int main()

{

int i;

int n;

cin >> n;

double\* p = new double[n];

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

cin >> p[i];

int\* d = new int[n];

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

cin >> d[i];

int\* t = new int[n];

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

cin >> t[i];

int\* x = new int[n], k = 0;

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

x[i] = 0;

JS<double> js(p, d, t, n);

js.JSFIFOBB();

js.GenerateAns(x, k);

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

cout << x[i] << " ";

cout << endl;

delete[] p;

delete[] d;

delete[] t;

delete[] x;

return 0;

}

**（运动员最佳匹配问题）羽毛球队有男女运动员各n人。给定2 个 n×n矩阵 P和 Q。P[i][j]是男运动员i和女运动员j配对组成混合双打的男运动员竞赛优势； Q[i][j]是女运动员i和男运动员 j配合的女运动员竞赛优势。由于技术配合和心理状态等各种因素影响，P[i][j]不一定等于 Q[j][i]。男运动员i和女运动员j配对组成混合双打的男女双方竞赛优势为P[i][j]\*Q[j][i]。 设计一个算法，计算男女运动员最佳配对法，使各组男女双方竞赛优势的总和达到最大。 设计一个优先队列式分支限界法，对于给定的男女运动员竞赛优势，计算男女运动员最佳配对法，使各组男女双方竞赛优势的总和达到最大。**

Input

第一行有1 个正整数n (1≤n≤20)。接下来的2n行，每行n个数。前n行是p，后n行是q。

Output

男女双方竞赛优势的总和的最大值。

#include<stdio.h>

#include<stdlib.h>

int P[100][100],Q[100][100],x[100],opt[100];

int tempV=0,maxV=0,n;

void func();

void traceback(int t);

int main(){

scanf("%d",&n);

int i,j;

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

x[i] = i;

}

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

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

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

}

}

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

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

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

}

}

traceback(1);

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

return 0;

}

void func(){

tempV=0;

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

tempV=tempV+P[i][x[i]]\*Q[x[i]][i];

}

if(tempV>maxV){

maxV= tempV;

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

opt[i] = x[i];

}

}

}

void traceback(int t){

int i,temp;

if(t>n){

func();

}

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

temp = x[i];

x[i] = x[t];

x[t] = temp;

traceback(t+1);

temp = x[i];

x[i] = x[t];

x[t] = temp;

}

}

***补充：贪心算法：***

**钱币找零问题**

**1、题目：指定币值和相应的数量，用最少的数量凑齐某金额。**

思路：利用贪心算法，我们优先选择面值大的钱币，以此类推，直到凑齐总金额。

算法实现：

public void greedy1(){

//面额

int[] values = { 1, 2, 5, 10, 20, 50, 100 };

//数量

int[] counts = { 3, 3, 2, 1, 1, 3, 3 };

//获取需要各种面值多少张

int[] result = getNumber1(446, values, counts);

System.out.println("各币值的数量："+Arrays.toString(result));

}

public int[] getNumber1(int sum , int[] values, int[] counts)

{

int[] result = new int[7];

int add=0; //当前凑的金额

for(int i=values.length-1;i>=0;i--)

{

int num = (sum-add)/values[i];

if(num>counts[i])

{

num=counts[i];

}

add=add+num\*values[i];

result[i]=num;

}

return result;

}

2、（**活动选择问题**）有n个需要在同一天使用同一个教室的活动a1,a2,…,an，教室同一时刻只能由一个活动使用。每个活动ai都有一个开始时间si和结束时间fi ，一旦被选择后，活动ai就占据半开时间区间[si,fi)。如果[si,fi]和[sj,fj]互不重叠，ai和aj两个活动就可以被安排在这一天，该问题就是要安排这些活动使得尽量多的活动能不冲突的举行。例如下图所示的活动集合S，其中各项活动按照结束时间单调递增排序。

public void greedy2(){

int [] st = {1,5,0,5,3,3,6,8,8,2,12};

int [] et = {4,9,6,7,8,5,10,12,11,13,14};

int num = getNumber2(st,et);

System.out.println("活动数量："+num);

}

public int getNumber2(int[] a , int[] b) //优先选择结束时间早的

{

int num=0;

int tempa=0;

int tempb=0;

int endTime=0;

int j=0;

for(int i=1;i<b.length;i++)//如果顺序混乱，则调整为结束时间从小到大的顺序,直接插入排序

{

tempb=b[i];

tempa=a[i];

for(j=i-1;j>=0&&tempb<b[j];j--)

{

b[j+1]=b[j];

a[j+1]=a[j];

if(j==0)

{

j--;

break;

}

}

b[j+1]=tempb;

a[j+1]=tempa;

}

System.out.println(Arrays.toString(a));

System.out.println(Arrays.toString(b));

num++;

endTime=b[0];

for(int k=1;k<b.length;k++)

{

if(a[k]>endTime)

{

num++;

endTime=b[k];

}

}

return num;

}

}

**动态规划法：走方格问题**

有一个矩阵map，它每个格子有一个权值。从左上角的格子开始每次只能向右或者向下走，最后到达右下角的位置，路径上所有的数字累加起来就是路径和，返回所有的路径中最小的路径和。

给定一个矩阵map及它的行数n和列数m，请返回最小路径和。保证行列数均小于等于100.

测试样例：

[[1,2,3],[1,1,1]],2,3

返回：4

import java.util.\*;

public class MinimumPath {

public int getMin(int[][] map, int n, int m) {

// write code here

int[][] dp = new int[n][m];

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

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

dp[i][0]+=map[j][0];

}

}

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

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

dp[0][i]+=map[0][j];

}

}

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

for(int j=1;j<m;j++){

dp[i][j] = min(dp[i][j-1]+map[i][j],dp[i-1][j]+map[i][j]);

}

}

return dp[n-1][m-1];

}

public int min(int a,int b){

if(a>b){

return b;

}else{

return a;

}

}

**回溯法：最大 k 乘积问题**：

设I是一个 n 位十进制整数。如果将 I 划分为 k 段，则可得到 k 个整数。这 k 个整数的乘积称为 I 的一个 k 乘积。试设计一个算法，对于给定的 I 和 k ，求出 I 的最大 k 乘积。

import java.util.Scanner;

public class 最大k乘积 {

private static long ans;

private static Scanner cin;

static{

cin = new Scanner(System.in);

}

static void work(int cur, int i, int k, long v){

//System.out.println("i = " + i + " cur = " + cur + " k = " + k);

if(i == k){

ans = Math.max(ans, v);

return;

}

if(cur == 0){

return;

}

int MOD = 1;

while(cur / MOD != 0){

work(cur % MOD, i + 1, k, v \* (cur / MOD));

MOD \*= 10;

}

}

public static void main(String[] args) {

int num, k;

System.out.println("请输入数字num和要分成的段数k: ");

while(cin.hasNext()){

num = cin.nextInt();

k = cin.nextInt();

ans = Long.MIN\_VALUE;

work(num, 0, k, 1L);

if(ans == Long.MIN\_VALUE){

System.out.println("整数" + num + "不能被分成" + k + "段");

}else{

System.out.println(num + "的最大" + k + "乘积是: " + ans);

}

System.out.println("请输入数字num和要分成的段数k: ");

}

}

}

分支限界法：最佳调度问题

最佳调度问题

【问题描述】

假设有n个任务由k个可并行工作的机器完成。完成任务i需要的时间为ti。试设计一个算法找出完成这n个任务的最佳调度，使得完成全部任务的时间最早。

【编程任务】

对任意给定的整数n和k，以及完成任务i需要的时间为ti，i=1~n。编程计算完成这n个任务的最佳调度。

【输入样例】

7 3

2 14 4 16 6 5 3

【输出样例】

17

#include<bits/stdc++.h>

using namespace std;

int n,k;

int x[100];//机器

int x1[100];//作业

int maxnum=1000000;

void task(int level)

{

if(level>n){

int temp=0;

for(int i=1;i<=k;i++){

if(x[i]>temp){

temp=x[i];

}

}

if(temp<maxnum){

maxnum=temp;

}

}

else{

for(int i=1;i<=k;i++){

x[i]+=x1[level];

task(level+1);

x[i]-=x1[level];

}

}

}

int main()

{

cin >> n;

cin >> k;

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

cin >>x1[i];

}

task(1);

cout << maxnum;

return 0;

}