一、搜索回溯分治

1、固定模式

递归回溯法算法框架[一]

```
procedure Search(k:integer);
begin
   for i:=1 to 算符种数 Do
     if  满足条件 then
        begin
           保存结果
           if  到目的地 then 输出解
             else Search(k+1);
           恢复：保存结果之前的状态{回溯一步}
        end;
end;
```

递归回溯法算法框架[二]

```
procedure Search(k:integer);
begin
  if  到目的地 then 输出解
    else
       for i:=1 to 算符种数 Do
          if  满足条件 then
            begin
               保存结果
               Search(k+1,参数表);
            end;
end;
```

（例 1）八皇后

八皇后问题：要在国际象棋棋盘中放八个皇后，使任意两个皇后都不能互相吃。（提示：皇后能吃同一行、同一列、同一对角线的任意棋子。）

放置第 i 个(行)皇后的算法为：

```
    procedure Search(i);
begin
```

```
      for  第 i 个皇后的位置=1 to 8 do；            //在本行的 8 列中去试
          if  本行本列允许放置皇后  then
        begin
          放置第 i 个皇后；
                  对放置皇后的位置进行标记；
            if i=8 then  输出                     //已经放完个皇后
               else Search(i+1)；                 //放置第 i+1 个皇后
          对放置皇后的位置释放标记，尝试下一个位置是否可行；
        end；
end；
```

```
#include<iostream>
#include<cmath>
using namespace std;

int a[9];
bool hang[9];        //行标记
bool lie[9];         //列标记
bool xie1[17];       //     \标记
bool xie2[17];       //    /标记
int total;

void print()
{
    ++total;
    printf("<%d>:",total);
    int i=0;
    for(i=1;i<9;++i)
    {
        printf("%d ",a[i]);
    }
    printf("\n");
}

void search(int t)
{
    int i=0;
    for(i=1;i<9;++i)                            //搜索所有的可能性
    {
        if(hang[t]&&lie[i]&&xie1[(t-i)+8]&&xie2[t+i])
        {
            a[t]=i;
            hang[t]=lie[i]=xie1[(t-i)+8]=xie2[t+i]=false;      //改变标记
            if(t==8)                                          //得到一种答案
```

```
                {
                        print();
                }
                Else                                        //没有得到，则继续搜索
                        search(t+1);
                hang[t]=lie[ i]=xie1[(t-i)+8]=xie2[t+i]=true;//把标记改回来，继续搜
        }
    }
}

int main()
{
        memset(hang,true,sizeof(hang));
        memset(lie,true,sizeof(lie));
        memset(xie1,true,sizeof(xie1));
        memset(xie2,true,sizeof(xie2));
        total=0;
        search(1);
        printf("total:%d\n",total);
}
```

（例 2）素数环

素数环:从 1 到 20 这 20 个数摆成一个环，要求相邻的两个数的和是一个素数。

【算法分析】

非常明显，这是一道回溯的题目。从 1 开始，每个空位有 20 种可能，只要填进去的数合法：与前面的数不相同；与左边相邻的数的和是一个素数。第 20 个数还要判断和第 1 个数的和是否素数。

【算法流程】

1、数据初始化；　　2、递归填数：判断第 J 种可能是否合法；

A、如果合法：填数；判断是否到达目标（20 个已填完）：是，打印结果；不是，递归填下一个；

B、如果不合法：选择下一种可能；

```
#include<iostream>
#include<cmath>
using namespace std;

int a[21];
bool b[21];
int total;

void print()
{
        ++total;
```

```
    int i=0;
    for(i=1;i<21;++i)
        printf("%d ",a[i]);
    printf("\n");
    char ch;
    ch=cin.get();

}

bool pd(int a,int b)
{
    int c=a+b;
    double sqrtm=sqrt(c*1.0);
    int i=0;
    for(i=2;i<=sqrtm;i++)
        if(c%i==0)
            return false;
    return true;
}


void search(int t)
{
    int i=0;
    for(i=1;i<=20;++i)              //搜索所有的可能性
    {
        if(pd(a[t-1],i)&&b[i])
        {
            a[t]=i;
            b[i]=false;                         //改变标记
            if(t==20)
            {
                if(pd(a[20],a[1]))   //搜到一种答案
                    print();
            }
            Else                              //没有搜到，继续搜索
                search(t+1);
            b[i]=true;                     //把标记改回来
        }
    }
}

int main()
{
        memset(b,true,sizeof(b));
        total=0;
        search(1);
        cout<<"total:"<<total<<endl;
}
```

2、一堆小棍儿找四个边

/*首先
错误的算法:贪心算法,首先从大到小排序,然后根据一定得条件,从大到小选择.
组成一条条边.可是发现这个方法是 wrong answer,原因:假如有一条边还
需要 8,有 7,5,3 可以提供选择,可是根据从大到小来选择,选择 7,剩下的 5 和 3 都无法使它满足,
于是出现错误.
正确算法：深度搜索＋剪枝＋排序

这个题目写出搜索还是很容易的，不容易的是怎么去剪枝，特别是下面的第三点剪枝，如果不剪枝的话每次都是 Time Limit Exceeded.剪枝剪了两个小时,看了有些心烦,
不过还是搞定了..给自己鼓励一下...
解题报告:
首先按木棍的长度从大到小排序,这样做方便下面的剪枝 2 和 3 用到,采用深度搜索..

剪枝:
1. 最长的木棍不可以超过边长.
2. 木棍的总长应该是 4 的倍数
3. 拼凑方法总是向小根拼凑,即如果是 8,3,5,为了避免多余的搜索,把大数放在前面,
即这个组合只能是 8,5,3.例如:假设 8,5,3 不行,那么 8,3,5 也不可以是一个组合.
*/

```
  #include<iostream>
  #include<algorithm>
  using namespace std;
  int sticknumber,bian;
  bool flag;
  int sticklength[25];
  bool used[25];
  void check(int s,int bs,int next)
                    //从 next 根木棍的位置开始拼凑长度,s:当前长度,bs:已经拼凑的木棍数
  {
       int ns,nbs,i;
       if(flag==true)
          return;
     for(i=next;i<sticknumber;i++)      //从 next 开始搜索.
  {
        ns=s;
        nbs=bs;
        if(used[i]==true)
            continue;
        else
          {
            if(ns+sticklength[i]>bian)
            {
                 continue;
            }
            ns+=sticklength[i];
            used[i]=true;
            if(ns==bian)
            {
                ns=0;
                nbs++;
                if(nbs==3)
                {            //找到了三边,另一边也就不用找了.
                    flag=true;
                }
            }
            if(flag==true)
                break;
            if(ns==0) //如果要重新凑边,要从新开始搜索.即从最大的没用的木棍开始
                if(ns<bian)
                    check(ns,nbs,0);
            if(ns!=0)    //如果不是重新开始凑边,要向小木棍搜索,即满足凑成的木棍
唯一性(剪枝 3)
                if(ns<bian)
                    check(ns,nbs,i);
            if(flag==true)
                break;
            used[i]=false;
        }
  }
}

bool cmp(int a,int b)
{
  return a>b;
}

int main()
```

```
{
    //freopen("in.txt","r",stdin);
    int testnumber;
        cin>>testnumber;
    int i;
    while(testnumber--)
    {
            cin>>sticknumber;
            bian=0;
            memset(used,0,sticknumber*sizeof(bool));
            for(i=0;i<sticknumber;i++)
            {
                    cin>>sticklength[i];
                    bian+=sticklength[i];
            }
            sort(sticklength,sticklength+sticknumber,cmp);

            flag=true;
            if((bian%4)!=0)//不是 4 的倍数(剪枝 2)
            {
                    flag=false;
            }
                bian=bian/4;

                if(sticklength[0]>bian)//第一根木棍的长度大于边长(剪枝 1)
            {
                    flag=false;
            }
            if(flag==false)
            {
                    cout<<"no"<<endl;
                    continue;
            }

                flag=false;
            check(0,0,0);
                if(flag==true)
                          cout<<"yes"<<endl;
                  else
                      cout<<"no"<<endl;
    }
    return 0;
}
```

3、求逆序对个数 hdu2689Sort it

Problem Description

You want to processe a sequence of n distinct integers by swapping two adjacent sequence elements until the sequence is sorted in ascending order. Then how many times it need.
For example, 1 2 3 5 4, we only need one operation : swap 5 and 4.

Input

The input consists of a number of test cases. Each case consists of two lines: the first line contains a positive integer n (n <= 1000); the next line contains a permutation of the n integers from 1 to n.

Output

For each case, output the minimum times need to sort it in ascending order on a single line.

Sample Input

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

Sample Output

```
0
6
```

```
/*
HDOJ 2689    求逆序对  归并排序    分治
*/

#include<iostream>
using namespace std;

int a[1100];
int ans;


void merge(int s,int mid,int t)//从大到小排序
{
       int b[1100];
       int i,j,k;
/*     cout<<"merge "<<s<<"    "<<mid<<"    "<<t<<endl;
```

```
        for(i=s;i<=t;i++)
                cout<<a[i]<<" ";
        cout<<endl;
*/      i=s,j=mid+1,k=0;
        while(i<=mid&&j<=t)
        {
                if(a[i]>a[j])
                {
                        b[k++]=a[i];
//                      cout<<"a["<<i<<"]:"<<a[i]<<"     a["<<j<<"]:"<<a[j]<<endl;
//                      cout<<"ans:"<<ans<<"->"<<ans+(t-j+1)<<endl;
                        ans+=(t-j+1);
                        i++;
                }
                else
                {
                        b[k++]=a[j];
                        j++;
                }
        }
        for(j=mid;j>=i;j--)
        {
                a[j+t-mid]=a[j];
        }
        k--;
        while(k>=0)
        {
                a[s+k]=b[k--];
        }
/*      for(i=s;i<=t;i++)
                cout<<a[i]<<" ";
        cout<<endl<<endl;
*/
}

void mergesort(int s,int t)
{
        if(s>=t)
                return;
        int mid=(s+t)/2;
        mergesort(s,mid);
        mergesort(mid+1,t);
        merge(s,mid,t);
}
```

```
int main()
{
        int n;
        while(scanf("%d",&n)!=EOF)
        {
                int i,j,k;
                ans=0;
                for(i=1;i<=n;i++)
                        scanf("%d",&a[i]);
                mergesort(1,n);
                printf("%d\n",ans);
        }
        return 0;
}
```

二、计算几何

1、两个凸多边形相交的面积(difficult)

Description

在平面上有两给定的凸多边形，若两凸多边形相交，则它们的交集也是一个凸多边形。若两凸多边形不相交，指的是两凸多边形相离或仅限于边界点与边上相交，则相交面积为 0。如图所示： 你的任务是编程给出交集多边形的面积。

Input

第一行为一整数 M，表示第一个凸多边形的边数，以后 M 行分别给出了 M 个顶点的坐标；接着，给出第二个凸多边形的边数 N，以后 N 行分别给出了 N 个顶点的坐标。

两给定的凸多边形按顺时针方向依次给出多边形每个顶点的坐标。

Output

只一个数据即交集面积，保留两位小数点。

Sample Input

```
4
0 0
0 1
1 1
1 0
4
-0.5 -0.5
-0.5 0.5
```

```
0.5 0.5
0.5 -0.5
Sample Output
0.25
```

做这个的时候感觉很怪，就是添加两条线段求交点的时候，两条线段添加的顺序不一样，得到的交点就不一样，看了很久，还是不知道。算了，既然逆时针和顺时针都对了，记着就行了。

```
#include<cstdio>
#include<cmath>
using namespace std;

struct DOT
{
        double x,y;
}a[301],b[301],L,now,p;

int n,m;
int ANS=0;

inline double X(DOT p1,DOT p2,DOT p0)
{
        return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

inline bool Cross(DOT A,DOT B,DOT C,DOT D) //叉积
{
        return ( (X(C,D,A)*X(C,D,B)<0)&&(X(A,B,C)*X(A,B,D)<0) );
}

inline DOT Intersection(DOT A,DOT B,DOT C,DOT D) //添加交点
{
        double c1,c2,tmp;
        DOT New;
        c1=abs(X(C,D,A));
        c2=abs(X(C,D,B));
        New.x=(A.x*c2+B.x*c1)/(c1+c2);
        New.y=(A.y*c2+B.y*c1)/(c1+c2);
        return New;
}

inline void Cut()
{
      int i,j,k;
      int N=0;
```

```
	a[n+1]=a[1];
	for( i=2 ;i<=n+1 ;i++ )
		if( X(a[i-1],now,L)<0 )//这里面的大于，小于都是视逆时针或顺时针而定，这
题是顺时针
		{
			if( X(a[i],now,L)>0 ) b[++N]=Intersection(a[i-1],a[i],now,L); //假如改成
Intersection(now,L,a[i-1],a[i]); 就错了，不知道为什么诶
		}
		else
		{
			b[++N]=a[i-1];
			if( X(a[i],now,L)<0 ) b[++N]=Intersection(a[i-1],a[i],now,L);
		}
	for( i=1 ;i<=N ;i++ ) a[i]=b[i];
	n=N;
}

inline void init()
{
	int i;
	scanf("%d",&n);
	for( i=1 ;i<=n ;i++ ) scanf("%lf%lf",&a[i].x,&a[i].y);
}

inline void doit(){
	int i,j,k;
	scanf("%d",&m);
	scanf("%lf%lf",&L.x,&L.y);
	p=L;
	for( i=1 ;i<m ;i++ )
	{

		scanf("%lf%lf",&now.x,&now.y);
		Cut();
		L=now;
	}
	now=p;
	Cut();
}

inline void print(){
	int i,j;
	double k=0;
	a[n+1]=a[1];
```

```
        for( i=1 ;i<=n ;i++ )
                k+=a[i].x*a[i+1].y-a[i+1].x*a[i].y;
        printf("%.2lf\n",abs(k)/2);
}

int main()
{
        //freopen("in.txt","r",stdin);
        init();
        doit();
        print();
        return 0;
}
```

2、极坐标的运用

问题描述，平面上有700个点，每个点在直角坐标系中的坐标已给出。如果从中任意找四个点组成凸四边形。问能组成多少凸四边形。

解法，如果用暴力，肯定超时。可以先选一个点做中心点，其它点按极坐标排序，然后遍历一遍其他点，如果能组成三角形。则可以和中心点组成凸四边形。

```
#include<iostream>
#include<cstdio>
#include<cmath>
#include<algorithm>
using namespace std;
const double pi=3.1415926535;
struct Point
{
        double x,y;//注意用double    如果用int会超时
};
Point a[1000];
double t[2100];
double dis(Point a, Point b)
{
        return sqrt(0.0+(a.x-b.x)*(a.x-b.x) + (a.y-b.y)*(a.y-b.y));
}
__int64 C(int n,int k)
{
        if(n<k) return 0;
        if(k==2)
                return (__int64)n*(n-1)/2;
        if(k==3)
                return (__int64)n*(n-1)*(n-2)/6;
        if(k==4)
        return (__int64)n*(n-1)*(n-2)*(n-3)/24;
}
int main()
{
    int ci;
    scanf("%d",&ci);
    while(ci--)
    {
        int n;scanf("%d",&n);
        for(int i=0;i<n;i++)
            scanf("%lf%lf",&a[i].x,&a[i].y);//注意用double   如果用int会超时
        __int64 cnt=C(n,4);//四边形总个数
        for(int i=0;i<n;i++)
        {
            int pl=0;//极角排序
            for(int j=0;j<n;j++)
            {
                if(i==j) continue;
                t[pl++]=acos((a[j].x-a[i].x)/dis(a[i],a[j]));//acos   important
                if(a[j].y<a[i].y) t[pl-1]=2*pi-t[pl-1];//点在下方    important
            }
            sort(t,t+pl);//极角排序
            for(int i=0;i<pl;i++)  t[i+pl]=t[i]+2*pi;//important
            __int64 res=C(n-1,3);//三角形总个数
            for(int i=0,j=1;i<pl&&j<2*pl;i++)
            {
                while(t[j]-t[i]<pi) j++;
                if(j-1-i>=2) res-=C(j-1-i,2);//C(j-1-i,2)  点在三角形外面的情况
            }
            cnt-=res;//res   点在三角形里面的情况
        }
        printf("%I64d\n",cnt);
    }
    return 0;
}/*如果四个点不能组成凸四边形，则必然是其中三个点组成一个三角形，另一个点在该三角形内部。于是我们可以O(n)枚举一个点作为内部中心，试图从其他的点里选出三个来，组成三角形把它包围住，看看有多少种可能的选择。继续观察发现，如果三个点不能圈住中心点，则必然是存在一条通过中心点的直线，使得这三点都在直线的同侧。于是我们可以把所有点（除了中心点）按极角排序，然后线性转圈扫描一下就可以统计出来了。总的复杂度是O(n^2*logn)*/
```

3、判断是否为凸多边形

4、判断点是否在（任意）多边形内

5、判断线段是否在多边形内

6、多边形重心

```
#include <stdlib.h>
#include <math.h>
#define MAXN 1000
#define offset 10000
#define eps 1e-8
#define zero(x) (((x)>0?(x):-(x))<eps)
#define _sign(x) ((x)>eps?1:((x)<-eps?2:0))
struct point{double x,y;};
struct line{point a,b;};

double xmult(point p1,point p2,point p0){
        return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
}

//判定凸多边形,顶点按顺时针或逆时针给出,允许相邻边共线
int is_convex(int n,point* p){
        int i,s[3]={1,1,1};
        for (i=0;i<n&&s[1]|s[2];i++)
                s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
        return s[1]|s[2];
}

//判定凸多边形,顶点按顺时针或逆时针给出,不允许相邻边共线
int is_convex_v2(int n,point* p){
        int i,s[3]={1,1,1};
        for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
                s[_sign(xmult(p[(i+1)%n],p[(i+2)%n],p[i]))]=0;
        return s[0]&&s[1]|s[2];
}

//判点在凸多边形内,顶点按顺时针或逆时针给出,在多边形边上返回0
int inside_convex_v2(point q,int n,point* p){
        int i,s[3]={1,1,1};
```

```
	for (i=0;i<n&&s[0]&&s[1]|s[2];i++)
		s[_sign(xmult(p[(i+1)%n],q,p[i]))]=0;
	return s[0]&&s[1]|s[2];
}

//判点在任意多边形内,顶点按顺时针或逆时针给出
//on_edge表示点在多边形边上时的返回值,offset为多边形坐标上限
int inside_polygon(point  q,int n,point* p,int on_edge=1){
	point q2;
	int i=0,count;
	while (i<n)
		for (count=i=0,q2.x=rand()+offset,q2.y=rand()+offset;i<n;i++)
			if
(zero(xmult(q,p[i],p[(i+1)%n]))&&(p[i].x-q.x)*(p[(i+1)%n].x-q.x)<eps&&(p[i].y-q.y)*(p[(i+
1)%n].y-q.y)<eps)
				return on_edge;
			else if (zero(xmult(q,q2,p[i])))
				break;
			else if
(xmult(q,p[i],q2)*xmult(q,p[(i+1)%n],q2)<-eps&&xmult(p[i],q,p[(i+1)%n])*xmult(p[i],q2,p[
(i+1)%n])<-eps)
				count++;
	return count&1;
}

inline  int opposite_side(point  p1,point p2,point l1,point  l2){
	return xmult(l1,p1,l2)*xmult(l1,p2,l2)<-eps;
}

inline  int dot_online_in(point  p,point l1,point  l2){
	return zero(xmult(p,l1,l2))&&(l1.x-p.x)*(l2.x-p.x)<eps&&(l1.y-p.y)*(l2.y-p.y)<eps;
}

//判线段在任意多边形内,顶点按顺时针或逆时针给出,与边界相交返回1
int inside_polygon(point  l1,point l2,int  n,point* p){
	point t[MAXN],tt;
	int i,j,k=0;
	if (!inside_polygon(l1,n,p)||!inside_polygon(l2,n,p))
		return 0;
	for (i=0;i<n;i++)
		if (opposite_side(l1,l2,p[i],p[(i+1)%n])&&opposite_side(p[i],p[(i+1)%n],l1,l2))
			return 0;
		else if (dot_online_in(l1,p[i],p[(i+1)%n]))
			t[k++]=l1;
		else if (dot_online_in(l2,p[i],p[(i+1)%n]))
			t[k++]=l2;
		else if (dot_online_in(p[i],l1,l2))
			t[k++]=p[i];
	for (i=0;i<k;i++)
		for (j=i+1;j<k;j++){
			tt.x=(t[i].x+t[j].x)/2;
			tt.y=(t[i].y+t[j].y)/2;
			if (!inside_polygon(tt,n,p))
				return 0;
		}
	return 1;
}

point intersection(line u,line v){
	point ret=u.a;
	double t=((u.a.x-v.a.x)*(v.a.y-v.b.y)-(u.a.y-v.a.y)*(v.a.x-v.b.x))
			/((u.a.x-u.b.x)*(v.a.y-v.b.y)-(u.a.y-u.b.y)*(v.a.x-v.b.x));
	ret.x+=(u.b.x-u.a.x)*t;
	ret.y+=(u.b.y-u.a.y)*t;
	return ret;
}

point barycenter(point a,point b,point c){
	line u,v;
	u.a.x=(a.x+b.x)/2;
	u.a.y=(a.y+b.y)/2;
	u.b=c;
	v.a.x=(a.x+c.x)/2;
	v.a.y=(a.y+c.y)/2;
	v.b=b;
	return intersection(u,v);
}

//多边形重心
point barycenter(int n,point* p){
	point ret,t;
	double t1=0,t2;
	int i;
	ret.x=ret.y=0;
	for (i=1;i<n-1;i++)
		if (fabs(t2=xmult(p[0],p[i],p[i+1]))>eps){
			t=barycenter(p[0],p[i],p[i+1]);
			ret.x+=t.x*t2;
```

```
                ret.y+=t.y*t2;
                t1+=t2;
            }
        if (fabs(t1)>eps)
            ret.x/=t1,ret.y/=t1;
        return ret;
    }
```

7、点到线段的距离

```
#include<iostream>
#include<cmath>
using namespace std;
#define Min(x,y) (x<y?x:y)
#define Max(x,y) (x>y?x:y)

struct Point
{
    double x,y;
};

inline double Dis(Point a,Point b)
{
    return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));
}

double MinDistance(Point p1,Point p2,Point q)
{
    int flag=1;
    double k;
    Point s;//q点到线段所在直线的垂足
    if(p1.x==p2.x)
    {
        s.x=p1.x;
        s.y=q.y;
        flag=0;
    }
    if(p1.y==p2.y)
    {
        s.x=q.y;
        s.y=p1.y;
        flag=0;
    }
    if(flag)
```

```
        {
               k=(p2.y-p1.y)/(p2.x-p1.x);
               s.x=(k*k*p1.x+k*(q.y-p1.y)+q.x)/(k*k+1);//两个方程联立得出垂足
               s.y=k*(s.x-p1.x)+p1.y;
        }
        if(Min(p1.x,p2.y)<=s.x && s.x<=Max(p1.x,p2.x))
                   return Dis(q,s);
        else
                 return Min(Dis(q,p1),Dis(q,p2));
}

int main()
{
        freopen("in.txt","r",stdin);
        Point p1,p2,q;
        while(cin>>p1.x>>p1.y>>p2.x>>p2.y>>q.x>>q.y)
        {
               cout<<MinDistance(p1,p2,q)<<endl;
        }
        return 0;
}
```

三、动态规划

<7>最小逼近问题(求出两数之比最接近某数/两数之和等于某数等等)
<8>方块消除游戏(某区间可以连续消去求最大效益)
<9>资源分配问题
<10>数字三角形问题
<11>漂亮的打印
<12>邮局问题与构造答案
<13>最高积木问题
<14>两段连续和最大
<15>2 次幂和问题
<16>N 个数的最大 M 段子段和
<17>交叉最大数问题
4.判定性问题的 dp(如判定整除、判定可达性等)
<1>模 K 问题的 dp
<2>特殊的模 K 问题，求最大(最小)模 K 的数
<3>变换数问题
5.单调性优化的动态规划
<1>1-SUM 问题
<2>2-SUM 问题
<3>序列划分问题(单调队列优化)
6.剖分问题(多边形剖分/石子合并/圆的剖分/乘积最大)
<1>凸多边形的三角剖分问题
<2>乘积最大问题
<3>多边形游戏(多边形边上是操作符,顶点有权值)
<4>石子合并(N^3/N^2/NLogN 各种优化)
7.贪心的动态规划
<1>最优装载问题
<2>部分背包问题
<3>乘船问题
<4>贪心策略
<5>双机调度问题 Johnson 算法
8.状态 dp
<1>牛仔射击问题(博弈类)
<2>哈密顿路径的状态 dp
<3>两支点天平平衡问题
<4>一个有向图的最接近二部图
9.树型 dp
<1>完美服务器问题(每个节点有 3 种状态)
<2>小胖守皇宫问题
<3>网络收费问题
<4>树中漫游问题
<5>树上的博弈
<6>树的最大独立集问题
<7>树的最大平衡值问题
<8>构造树的最小环

1、数塔 hdu2084

```
#include<iostream>
using namespace std;

int f[202][202];

int max(int a,int b)
{
	if(a>b)
		return a;
	else
		return b;
}
int main()
{
	int N=0;
	scanf("%d",&N);
	while(N--)
	{
		int depth=0;
		scanf("%d",&depth);
		int i=0,j=0;
		for(i=0;i<202;++i)
		{
			for(j=0;j<202;++j)
			{
				f[i][j]=-9999999;
			}
		}
		int a=0,b=0;
		int ans=0;
		int k=0;
		for(i=1;i<=depth;++i)
		{
			for(j=100-i+1,k=0;k<i;j+=2,k++)
				scanf("%d",&f[i][j]);
		}
		int temp1=0,temp2=0;
		for(i=depth-1;i>0;--i)
		{
			for(j=0;j<=200;++j)
			{
				if(j==0)
```

```
                {
                        temp1=max(f[i+1][j],f[i+1][j+1]);
                        f[i][j]+=temp1;
                }
                else if(j==200)
                {
                        temp2=max(f[i+1][j],f[i+1][j-1]);
                        f[i][j]+=temp2;
                }
                else
                {
                        temp1=max(f[i+1][j],f[i+1][j+1]);
                        temp2=max(f[i+1][j],f[i+1][j-1]);
                        f[i][j]+=max(temp1,temp2);
                }
            }
        }
        printf("%d\n",f[1][100]);
    }
}
```

2、最长不下降子序列

3、最长公共子序列　hdu1159

```
#include<iostream>
#include<string>
using namespace std;

int main()
{
    int f[500][500];
    string str1;
    string str2;
    int len1,len2;
    while(cin>>str1>>str2)
    {
        len1=str1.length();
        len2=str2.length();
        int i,j;
        for(i=0;i<=len1;i++)
        {
            f[0][i]=0;
        }
        for(i=0;i<=len2;i++)
        {
            f[i][0]=0;
        }
        for(i=1;i<=len1;i++)
        {
            for(j=1;j<=len2;j++)
            {
                if(str1[i-1]==str2[j-1])
                {
                    f[i][j]=f[i-1][j-1]+1;
                }
                else
                    f[i][j]=(f[i-1][j]>f[i][j-1])?f[i-1][j]:f[i][j-1];
            }
        }
        cout<<f[len1][len2]<<endl;
    }
}
```

4、最大连续子段和　hdu1231

```
#include<iostream>
using namespace std;

int f[10001];
int state[10001];

int main()
{
    int K=0;
    while(scanf("%d",&K)&&K)
    {
        int i,j;
        memset(f,0,sizeof(f));
        memset(state,0,sizeof(state));
        for(i=0;i<K;++i)
        {
            scanf("%d",&f[i]);
        }
        state[0]=f[0];
        for(i=1;i<K;++i)
        {
            if(state[i-1]>=0)
                    //如果到该节点时，状态和>=0,说明前面的序列可以为后面做
贡献
            {
                state[i]=state[i-1]+f[i];
            }
            else        //如果到该节点时，状态和<0,说明前面的序列不能为后面做
贡献
                state[i]=f[i];
        }
        int max=-999999999;
        int start=0,end=0;
        for(i=0;i<K;++i)    //寻找最大值，以及结束节点
        {
            if(max<state[i])
            {
                max=state[i];
                end=i;
            }
        }
        for(i=end;i>=0;--i) //寻找开始节点
        {
            if(state[i]<0)
            {
                start=i+1;
                break;
            }
        }
        if(max<0)       //如果和的最大值是负数，说明序列全为负数
            cout<<"0 "<<f[0]<<" "<<f[K-1]<<endl;
        else
            cout<<max<<" "<<f[start]<<" "<<f[end]<<endl;
    }
}
```

5、字符串对称处理

问题：要求对任意一个字符串，通过加入若干字符，是指对称。
如Ab3bd 插入两个字符后可以变成dAb3bAd或
Adb3bdA,但插入两个字符以下却无法完成对称性
处理。请求出需要插入的最少字符数

利用数字三角的思路
引入 Cost[i][j]表示

将串 S 从第 j 位开始长度为 i 的子串 sub 处理成对
称的最小插入字符数，
则按子串 sub 的长度从大到小依次进行递推求
解：

$$\text{Cost[i][j]}=\begin{cases}\text{cost[i-2][j+1]} & \text{若 st[j]=st[i+j-1]}\\ \text{min\{cost[i-1][j]+1,cost[i-1][j+1]+1\}} & \end{cases}$$

若 st[j]<>st[i+j-1]; 2=<i<=n,1=<j<=n-i+1
i=0 或 i=1 时 cost 取值均为 0。

6、邮局与村庄　poj1160 Post_Office

在一条高速公路边上有 V 个村庄，用一条坐标轴来描述这条公路，每个村庄的坐标各不相同，且都是整数。两个村庄间的距离用他们的坐标值差的绝对值表示。现在要在这些村庄中选出 P 个建立邮局，邮局建立在村庄里，每个村庄使用离他最近的那个邮局。求一种建设方案，使得所有村庄到各自所使用的邮局的距离总和最小。
输入：共两行：
第一行给出村庄数目 V 和邮局数目 P，
　1<=V<=300，1<=P<=30，P<=V;
第二行按递增顺序给出 V 个村庄的坐标 x1,x2,…xV，1<=xi<=10000
输出：　共两行：
第一行给出所有村庄到各自所使用的邮局的距离总和 S；
第二行按递增顺序给出 P 个邮局的坐标 x1,x2,…xP。

- 这是一道典型的动态规划的题目
- 阶段的划分标准是已安排的邮局数和覆盖的村庄数（所谓“某邮局覆盖的村庄”是指使用该邮局的村庄）。
- 现以 Cost[i,j]表示安排前 i 个邮局给前 j 个村庄使用，通过某种安排手段，使这 j 个村庄分别到这 i 个邮局中最近的一个的距离之和的所取到的最小值

- 先考虑最简单的情况：Cost[n，1]

－　设 n 个村庄的坐标值按递增顺序依次为 x1,x2,⋯xn，邮局坐标为 xr，考虑
|xi-xr|+|xn+1-i-xr|>=|xi-xn+1-i|.

- n=2k，|x1-xr|+|x2-xr|+⋯+|x2k-xr|
=|x1-xr|+|x2k-xr|+|x2-xr|+|x2k-1-xr|+… +|xk-xr|+|xk+1-xr|
>=|x1-x2k|+|x2-x2k-1|+⋯+|xk-xk+1|
- n=2k-1，|x1-xr|+|x2-xr|+⋯+|x2k-1-xr|
=|x1-xr|+|x2k-1-xr|+|x2-xr|+|x2k-2-xr|+…
+|xk-1-xr|+|xk+1-xr|+|xk-xr|
>=|x1-x2k-1|+|x2-x2k-2|+⋯+|xk-1-xk+1|

当 xr = xk 或时取到最小值。

$$Cost[n,1]=-\sum_{i=1}^{[n/2]}x_i+\sum_{i=[(n+3)/2]}^{n}x_i$$

- 综上，此时 xr = x[n/2]，式中[]表示取整。

- 现在考虑 Cost[n,m]，在前 n 个村庄中建立 m 个邮局，设最优情况下前 m-1 个邮局覆盖的范围是第 1~r 个村庄，第 m 个邮局覆盖的范围是第 r+1~n 个村庄，则这两部分邮局的设置都必须是最优的，因此

  Cost[n,m]=Cost[r,m-1]+D[r+1,n]

  其中 D[i,j]表示在第 i 个到第 j 个村庄中建立一个邮局得到的最短距离总和。
- 考虑到 Cost[n,m]仅与 Cost[i,m-1]相关，可以用两个一维数组存储
- D[a,b]可以利用关系式 d[a,a]=0, D[a,b]=D[a,b-1]+x[b]-x[(a+b)/2]预先计算存入一个数组。

```
//poj1160post office
#include<iostream>
using namespace std;

int dis[310][310],m[310][310],axis[310],v_num,p_num;

void DP()
{
    int i,j,k,mid;
    int sum,temp;
    for(i=1;i<v_num;i++)
    {
        for(j=i;j<=v_num;j++)
        {
            sum=0;
            temp=1;
            mid=(i+j)/2;
            for(k=i;k<mid;k++)
            {
                sum+=temp*axis[k];
                temp++;
            }
            temp=1;
             for(k=j;k>mid;k--)
            {
                sum+=temp*axis[k-1];
                temp++;
            }
            dis[i][j]=sum;
        }

    }

    for(i=1;i<=v_num;i++)      //用一个邮局
        m[i][1]=dis[1][i];
    for(i=2;i<=v_num;i++)      //用 i 个邮局
```

```
        {
                for(j=i;j<=v_num;j++) //前 j 个村庄用 i 个邮局
                {
                        temp=100000000;
                        for(k=i;k<j;k++)
                        {
                                if(temp>m[k][i-1]+dis[k+1][j])
                                        temp=m[k][i-1]+dis[k+1][j];
                        }
                        m[j][i]=temp;
                }
        }
        cout<<m[v_num][p_num]<<endl;
}

int main()
{
        int i,j,k;
        int temp1,temp2;
        while(scanf("%d %d",&v_num,&p_num)==2)
        {
                scanf("%d",&temp1);
                for(i=1;i<v_num;i++)
                {
                        scanf("%d",&temp2);
                        axis[i]=temp2-temp1;
                        temp1=temp2;
                }
                DP();
        }
        return 0;
}
```

7、箱子叠加　hdu1069Monkey and Banana

Problem Description

A group of researchers are designing an experiment to test the IQ of a monkey. They will hang a banana at the roof of a building, and at the mean time, provide the monkey with some blocks. If the monkey is clever enough, it shall be able to reach the banana by placing one block on the top another to build a tower and climb up to get its favorite food.

The researchers have n types of blocks, and an unlimited supply of blocks of each type. Each type-i block was a rectangular solid with linear dimensions (xi, yi, zi). A block could be reoriented so that any two of its three dimensions determined the dimensions of the base and the other dimension was the height.

They want to make sure that the tallest tower possible by stacking blocks can reach the roof. The problem is that, in building a tower, one block could only be placed on top of another block as long as the two base dimensions of the upper block were both strictly smaller than the corresponding base dimensions of the lower block because there has to be some space for the monkey to step on. This meant, for example, that blocks oriented to have equal-sized bases couldn't be stacked.

Your job is to write a program that determines the height of the tallest tower the monkey can build with a given set of blocks.

Input

The input file will contain one or more test cases. The first line of each test case contains an integer n,
representing the number of different blocks in the following data set. The maximum value for n is 30.
Each of the next n lines contains three integers representing the values xi, yi and zi.
Input is terminated by a value of zero (0) for n.

Output

For each test case, print one line containing the case number (they are numbered sequentially starting from 1) and the height of the tallest possible tower in the format "Case case: maximum height = height".

Sample Input

```
1
10 20 30
2
6 8 10
5 5 5
7
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
7 7 7
5
```

```
31 41 59
26 53 58
97 93 23
84 62 64
33 83 27
0
```

Sample Output

```
Case 1: maximum height = 40
Case 2: maximum height = 21
Case 3: maximum height = 28
Case 4: maximum height = 342
```

```
#include<iostream>
#include<algorithm>
using namespace std;

struct Node
{
    int x;
    int y;
    int z;
};

Node arr[91];

int f[91];

bool cmp(Node a,Node b)
{
    if(a.x!=b.x)
            return a.x>b.x;
    else if(a.y!=b.y)
            return a.y>b.y;
    else
            return a.z>b.z;
}

int main()
{
    int T=1;
    int N=0;
    int num[3];
int k;
while(scanf("%d",&N) && N)
{
        int i,j;
        k=0;
        memset(f,0,sizeof(f));
        for(i=0;i<N;++i)
        {
                scanf("%d%d%d",&num[0],&num[1],&num[2]);
                sort(num+0,num+3);
                arr[k].x=num[0],arr[k].y=num[1],arr[k].z=num[2];k++;
                arr[k].x=num[0],arr[k].y=num[2],arr[k].z=num[1];k++;
                arr[k].x=num[1],arr[k].y=num[2],arr[k].z=num[0];k++;
        }
        sort(arr,arr+k,cmp);
        /*for(i=0;i<k;++i)
        {
                cout<<arr[i].x<<" "<<arr[i].y<<" "<<arr[i].z<<endl;
        }
        cout<<endl<<endl;*/
        f[0]=arr[0].z;
        int max=0;
        for(i=1;i<k;++i)
        {
                max=0;
                for(j=0;j<i;++j)
                {
                        if( (arr[i].x<arr[j].x && arr[i].y<arr[j].y))
                        {
                                if(max<f[j])
                                        max=f[j];
                        }

                }
                f[i]=max+arr[i].z;
        }
        max=0;
        for(i=0;i<k;++i)
        {
                if(max<f[i])
                        max=f[i];
        }
        printf("Case %d: maximum height = %d\n",T,max);
        T++;
        }
    }
```

8、矩阵最大和　hdu1081To The Max

Problem Description

Given a two-dimensional array of positive and negative integers, a sub-rectangle is any contiguous sub-array of size 1 x 1 or greater located within the whole array. The sum of a rectangle is the sum of all the elements in that rectangle. In this problem the sub-rectangle with the largest sum is referred to as the maximal sub-rectangle.

As an example, the maximal sub-rectangle of the array:

0 -2 -7 0
9 2 -6 2
-4 1 -4 1
-1 8 0 -2

is in the lower left corner:

9 2
-4 1
-1 8

and has a sum of 15.

Input

The input consists of an N x N array of integers. The input begins with a single positive integer N on a line by itself, indicating the size of the square two-dimensional array. This is followed by N 2 integers separated by whitespace (spaces and newlines). These are the N 2 integers of the array, presented in row-major order. That is, all numbers in the first row, left to right, then all numbers in the second row, left to right, etc. N may be as large as 100. The numbers in the array will be in the range [-127,127].

Output

Output the sum of the maximal sub-rectangle.

Sample Input

```
4
0 -2 -7 0 9 2 -6 2
-4 1 -4 1 -1
```

```
8 0 -2
```

Sample Output

```
15
```

//对于二维矩阵，动态规划不好解，因此可以转化为一维的。

```
#include<iostream>
using namespace std;

int arr[101][101];
int state[101];

int main()
{
	int N;
	while(scanf("%d",&N)!=EOF)
	{
		int i,j,k,h;
		int temp;
		memset(arr,0,sizeof(arr));
		memset(state,0,sizeof(state));
		for(i=1;i<=N;++i)
		{
			for(j=1;j<=N;++j)
			{
				scanf("%d",&temp);
				arr[i][j]=arr[i][j-1]+temp;
			}
		}
		int maxsum=0;
		for(i=1;i<=N;++i)
		{
			for(j=i;j<=N;++j)
			{
				for(k=1;k<=N;++k)
				{
					if(state[k-1] >= 0)
						state[k]=state[k-1]+arr[k][j]-arr[k][i-1];
					else
						state[k]=arr[k][j]-arr[k][i-1];
					if(maxsum<state[k])
						maxsum=state[k];
				}
```

```
                }
            }
            cout<<maxsum<<endl;
        }
}
```

9、括号匹配/矩阵连乘　poj1141Brackets Sequence

括号匹配和矩阵连乘思路一样。都是先把一个当做一个状态，再将相邻的两个当做一个状态，再将相邻的三个当做一个状态……最后把全部当做一个状态。得到结果。矩阵连乘不需要输出具体的结合方法，比括号匹配简单一步。

Description

Let us define a regular brackets sequence in the following way:

1. Empty sequence is a regular sequence.
2. If S is a regular sequence, then (S) and [S] are both regular sequences.
3. If A and B are regular sequences, then AB is a regular sequence.

For example, all of the following sequences of characters are regular brackets sequences:

(), [], (()), ([]), ()[], ()[()]

And all of the following character sequences are not:

(, [,),)(, ([)], ([(]

Some sequence of characters '(', ')', '[', and ']' is given. You are to find the shortest possible regular brackets sequence, that contains the given character sequence as a subsequence. Here, a string a1 a2 ... an is called a subsequence of the string b1 b2 ... bm, if there exist such indices 1 = i1 < i2 < ... < in = m, that aj = bij for all 1 = j = n.

Input

The input file contains at most 100 brackets (characters '(', ')', '[' and ']') that are situated on a single line without any other characters among them.

Output

Write to the output file a single line that contains some regular brackets sequence that has the minimal possible length and contains the given sequence as a subsequence.

Sample Input

```
([(]
```

Sample Output

```
()[()]
```

```
#include<iostream>
#include<string>
using namespace std;
string    s;
int d[110][110],p[110][110];
int len;

void dp()
{
        int i,j,k,m;
        for(i=0;i<=100;i++)
                d[i][i]=1;
        for(k=1;k<len;k++)
        {
                for(i=0;i<len-k;i++)
                {
                        j=i+k;
                        if( (s[i]=='(' && s[j]==')') || (s[i]=='[' && s[j]==']') )
                        {
                                d[i][j]=d[i+1][j-1];
                                p[i][j]=-1;
                        }
                        else
                                d[i][j]=999999999;
                        for(m=i;m<j;m++)
                        {
                                if(d[i][j]>d[i][m]+d[m+1][j])
                                {
                                        d[i][j]=d[i][m]+d[m+1][j];
                                        p[i][j]=m;
                                }
                }

            }
        }
}




void output(int i,int j)
{
        if(i>j)
                return ;
        else if(i==j)
        {
                switch(s[i])
                {
                        case '(':
                        case ')':cout<<"()";break;
                        case '[':
                        case ']':cout<<"[]";break;
                }
        }
        else if(p[i][j]==-1)
        {
                cout<<s[i];
                output(i+1,j-1);
                cout<<s[j];
        }
        else
        {
                output(i,p[i][j]);
                output(p[i][j]+1,j);
        }
}

int main()
{
        while(getline(cin,s))
        {
                memset(d,0,sizeof(d));
                memset(p,-1,sizeof(p));
                len=s.length();
                dp();
        output(0,len-1);
        cout<<endl;
    }
    return 0;
}
```

10、凸多边形的最优三角剖分

四、背包问题

1、01 背包 hdu2602 Bone Collector

Problem Description

Many years ago , in Teddy’s hometown there was a man who was called “Bone Collector”. This man like to collect varies of bones , such as dog’s , cow’s , also he went to the grave …

The bone collector had a big bag with a volume of V ,and along his trip of collecting there are a lot of bones , obviously , different bone has different value and different volume, now given the each bone’s value along his trip , can you calculate out the maximum of the total value the bone collector can get ?

Input

The first line contain a integer T , the number of cases.

Followed by T cases , each case three lines , the first line contain two integer N , V, (N <= 1000 , V <= 1000)representing the number of bones and the volume of his bag. And the second line contain N integers representing the value of each bone. The third line contain N integers representing the volume of each bone.

Output

One integer per line representing the maximum of the total value (this number will be less than 2^{31}).

Sample Input

```
1
5 10
1 2 3 4 5
5 4 3 2 1
```

Sample Output

```
14
```

```
#include<iostream>
using namespace std;

struct Node
{
    int value;
    int volume;
};

Node arr[1002];
int f[1002];
int main()
{
    int N,V;
    int T=0;scanf("%d",&T);
    while(T--)
    {
        scanf("%d%d",&N,&V);
        int i,j;
        for(i=0;i<1001;++i)
            f[i]=0;
        for(i=1;i<=N;++i)
        {
            scanf("%d",&arr[i].value);
        }
        for(i=1;i<=N;++i)
        {
            scanf("%d",&arr[i].volume);
        }
        for(i=1;i<=N;++i)
        {
            for(j=V;j>=arr[i].volume;--j)
            {
                if(f[j-arr[i].volume]+arr[i].value>f[j])
                    f[j]=f[j-arr[i].volume]+arr[i].value;
            }
        }
        cout<<f[V]<<endl;
    }
}
```

2、完全背包（恰好装满）hdu1114Piggy-Bank

Input

The input consists of T test cases. The number of them (T) is given on the first line of the input file. Each test case begins with a line containing two integers E and F. They indicate the weight of an empty pig and of the pig filled with coins. Both weights are given in grams. No pig will weigh more than 10 kg, that means 1 <= E <= F <= 10000. On the second line of each test case, there is an integer number N (1 <= N <= 500) that gives the number of various coins used in the given currency. Following this are exactly N lines, each specifying one coin type. These lines contain two integers each, Pand W (1

<= P <= 50000, 1 <= W <=10000). P is the value of the coin in monetary units, W is it's weight in grams.

Output

Print exactly one line of output for each test case. The line must contain the sentence "The minimum amount of money in the piggy-bank is X." where X is the minimum amount of money that can be achieved using coins with the given total weight. If the weight cannot be reached exactly, print a line "This is impossible.".

Sample Input

```
3
10 110
2
1 1
30 50
10 110
2
1 1
50 30
1 6
2
10 3
20 4
```

Sample Output

```
The minimum amount of money in the piggy-bank is 60.
The minimum amount of money in the piggy-bank is 100.
This is impossible.
```

//f[0]初始化为 0，其他 f 取值为最大值。
//循环时，f[j]=min(f[j],f[j-arr[i].haofei]+arr[i].jiazhi);

```
#include<iostream>
#include<cmath>
using namespace std;
#define Max 999999999
struct Node
{
	int pvalue;
	int weight;
};
Node arr[501];
int f[10001];
int main()
{
	int T=0;
	scanf("%d",&T);
	while(T--)
	{
		int E=0,F=0;
		scanf("%d%d",&E,&F);
		int W=F-E;
		int i,j;
		for(i=1,f[0]=0;i<10001;++i)
			f[i]=Max;
		int n=0;
		scanf("%d",&n);
		for(i=1;i<=n;++i)
			scanf("%d%d",&arr[i].pvalue,&arr[i].weight);
		for(i=1;i<=n;++i)
		{
			for(j=arr[i].weight;j<=W;++j)
			{
					f[j]=min(f[j],f[j-arr[i].weight]+arr[i].pvalue);
			}
		}
		if(f[W]<Max)
			printf("The minimum amount of money in the piggy-bank
is %d.\n",f[W]);
		else
			printf("This is impossible.\n");
	}
}
```

3、多重背包 poj1276 Cash Machine

Description

A Bank plans to install a machine for cash withdrawal. The machine is able to deliver appropriate @ bills for a requested cash amount. The machine uses exactly N distinct bill denominations, say Dk, k=1,N, and for each denomination Dk the machine has a supply of nk bills. For example,
N=3, n1=10, D1=100, n2=4, D2=50, n3=5, D3=10

means the machine has a supply of 10 bills of @100 each, 4 bills of @50 each, and 5 bills of @10 each.

Call cash the requested amount of cash the machine should deliver and write a program that computes the maximum amount of cash less than or equal to cash that can be effectively delivered according to the available bill supply of the machine.

Notes:
@ is the symbol of the currency delivered by the machine. For instance, @ may stand for dollar, euro, pound etc.

Input

The program input is from standard input. Each data set in the input stands for a particular transaction and has the format:

cash N n1 D1 n2 D2 ... nN DN

where 0 <= cash <= 100000 is the amount of cash requested, 0 <=N <= 10 is the number of bill denominations and 0 <= nk <= 1000 is the number of available bills for the Dk denomination, 1 <= Dk <= 1000, k=1,N. White spaces can occur freely between the numbers in the input. The input data are correct.

Output

For each set of data the program prints the result to the standard output on a separate line as shown in the examples below.

Sample Input

```
735 3  4 125  6 5  3 350
633 4  500 30  6 100  1 5  0 1
735 0
0 3  10 100  10 50  10 10
```

Sample Output

```
735
630
0
0
```

Hint

The first data set designates a transaction where the amount of cash requested is @735. The machine contains 3 bill denominations: 4 bills of @125, 6 bills of @5, and 3 bills of @350. The machine can deliver the exact amount of requested cash.

In the second case the bill supply of the machine does not fit the exact amount of cash requested. The maximum cash that can be delivered is @630. Notice that there can be several possibilities to combine the bills in the machine for matching the delivered cash.

In the third case the machine is empty and no cash is delivered. In the fourth case the amount of cash requested is @0 and, therefore, the machine delivers no cash.

```
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

int cash,N,bag[100001],n,d;
using namespace std;

int main(){
        //freopen("input","r",stdin);
        while(scanf("%d%d",&cash,&N)!=EOF){
                memset(bag,0,sizeof(bag));//清 0
                while(N--){//循环输入 N 个物品
                        scanf("%d%d",&n,&d);
                        if(n*d>=cash)//完全背包
                          for(int v=d;v<=cash;++v)
                                bag[v]=max(bag[v],bag[v-d]+d);
                        else{//多重背包（包括 01 的情况）
                                int num,cost,value;//因为花费和价
值相同，其实 cost,value

//只要一个就行了
                                while(n){//二进制处理
                                        num=(n+1)/2;
                                        n-=num;
                                        cost=value=d*num;
                                        for(int v=cash;v>=cost;--v)
bag[v]=max(bag[v],bag[v-cost]+value);
                                    }
                              }
                        }
                        printf("%d\n",bag[cash]);
                  }
}
```

4、二维背包 hdu2159FATE

Problem Description

最近 xhd 正在玩一款叫做 FATE 的游戏，为了得到极品装备，xhd 在不停的杀怪做任务。久而久之 xhd 开始对杀怪产生的厌恶感，但又不得不通过杀怪来升完这最后一级。现在的问题是，xhd 升掉最后一级还需 n 的经验值，xhd 还留有 m 的忍耐度，每杀一个怪 xhd 会得到相应的经验，并减掉相应的忍耐度。当忍耐度降到 0 或者 0 以下时，xhd 就不会玩这游戏。xhd 还说了他最多只杀 s 只怪。请问他能升掉这最后一级吗？

Input

输入数据有多组，对于每组数据第一行输入 n，m，k，s(0 < n,m,k,s < 100)三个正整数。分别表示还需的经验值，保留的忍耐度，怪的种数和最多的杀怪数。接下来输入 k 行数据。每行数据输入两个正整数 a，b(0 < a,b < 20)；分别表示杀掉一只这种怪 xhd 会得到的经验值和会减掉的忍耐度。(每种怪都有无数个)

Output

输出升完这级还能保留的最大忍耐度，如果无法升完这级输出-1。

Sample Input

```
10 10 1 10
1 1
10 10 1 9
1 1
9 10 2 10
1 1
2 2
```

Sample Output

```
0
-1
1
```

hdoj2159

```
/*Description:二维背包。为了求解方便，将忍耐度放在第一维，杀怪数
 放在第二维*/

#include<iostream>
```

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int n,m,k,s,a,b,bag[100][100];//bag 存放最大经验值
using namespace std;

int main(){
    //freopen("input","r",stdin);
    while(scanf("%d%d%d%d",&n,&m,&k,&s)!=EOF){
        memset(bag,0,sizeof(bag));
        while(k--){
            scanf("%d%d",&a,&b);
            for(int i=b;i<=m;++i)//第一维忍耐度
              for(int j=1;j<=s;++j)//第二维杀怪数
                bag[i][j]=max(bag[i][j],bag[i-b][j-1]+a);
        }
        int ans=-1;
        for(int i=1;ans==-1&&i<=m;++i)
          for(int j=1;j<=s;++j)
            if(bag[i][j]>=n)
              ans=i;//当找到一个大于 n 的经验值时，跳出循环， 由于是
                    //从小到大搜索的，这时的忍耐度花费最小
        if(ans!=-1)
          ans=m-ans;
        printf("%d\n",ans);
    }
}
```

5、依赖背包 hdu1561 The more, The Better

Problem Description

ACboy 很喜欢玩一种战略游戏，在一个地图上，有 N 座城堡，每座城堡都有一定的宝物，在每次游戏中 ACboy 允许攻克 M 个城堡并获得里面的宝物。但由于地理位置原因，有些城堡不能直接攻克，要攻克这些城堡必须先攻克其他某一个特定的城堡。你能帮 ACboy 算出要获得尽量多的宝物应该攻克哪 M 个城堡吗？

Input

每个测试实例首先包括 2 个整数，N,M.(1 <= M <= N <= 200);在接下来的 N 行里，每行包括 2 个整数，a,b. 在第 i 行，a 代表要攻克第 i 个城堡必须先攻克第 a 个城堡，如果 a = 0 则代表可以直接攻克第 i 个城堡。b 代表第 i 个城堡的宝物数量, b >= 0。当 N = 0, M = 0 输入结束。

Output

对于每个测试实例，输出一个整数，代表 ACboy 攻克 M 个城堡所获得的最多宝物的数量。

Sample Input

```
3 2
0 1
0 2
0 3
7 4
2 2
0 1
0 4
2 1
7 1
7 6
2 2
0 0
```

Sample Output

```
5
13
```

```
#include<iostream>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int
f[201][201],N,M,a,b,cnt,E[201],next[201],first[201],val[201],root[201
],root_cnt,bag[201];
using namespace std;

void deal(int i){//处理第 i 个物品
    int p=first[i];
    while(p!=-1){//有儿子，递归处理子结点
       deal(E[p]);
       p=next[p];
    }
    p=first[i];
    if(p==-1)//没有儿子，没有物品依赖它
      f[i][1]=val[i];
    else{//有物品依赖它
       int son=E[p];//第一个儿子
       //下面两句是将第一个儿子生成的一组物品赋给第 i 个物品，它们
       //的下标是相差 1 的，因为它的后代是依赖于它的，后代生成的物
       //品的存在是以它的存在为前提的，差的那个 1 就是第 i 个物品占
       //去的那个花费，当然也可以另开空间，等把所有的生成物品计算
       //好后再赋值给 f[i][v]。
       for(int v=1;v<=M;++v)
         f[i][v]=f[son][v-1];
```

```
            p=next[p];
            //下面的这个循环相当于求一个分组背包。一个儿子是一组，每个
            //儿子生成的物品是一组中的物品。但与分组背包还有不同，就是
            //这里的 f[i][v]表示将容量为 v 的背包恰好装满时的最大价值，这
            //样就要对一些情况作出处理，以防止未装满的情况发生。另外，
            //如果把这些处理去掉，就成了一般的分组背包的样子。这样也是
            //正确的，但时间效率低一些。
            while(p!=-1){//对于它的每个儿子生成的物品组都进行计算
                son=E[p];//儿子
                p=next[p];
                for(int v=M;v>=1;--v){//倒序进行
                    for(int po=1;po<=v;++po){
                        //当相加的两个物品有一个为 0 时，说明那个物品是不存
                        //的，不处理，如果也处理的话就会产生背包装不满的
                        //情况
                        if((po!=1&&f[i][po]==0)||(po!=v&&f[son][v-po]==0))
                          continue;
                        else
                          f[i][v]=max(f[i][v],f[i][po]+f[son][v-po]);
                    }
                }
            }
            //把第 i 个物品算上去
            f[i][1]=val[i];//这个情况要分出来讨论，因为 f[i][1]是 0
            for(int v=2;v<=M;++v)
              if(f[i][v]!=0)//等于 0 是背包装不满的标志，或说这个物品不存在
                f[i][v]+=val[i];
        }

}

int main(){
    //freopen("input","r",stdin);
    while(scanf("%d%d",&N,&M)!=EOF&&N&&M){
        root_cnt=cnt=0;//root_cnt 是不主件的个数，cnt 是存储子结点时的标记
        memset(f,0,sizeof(f));//f[i][v]是第 i 个物品及其附件产生的一组物品
        memset(first,-1,sizeof(first));//first[i]指 i 的第一个孩子的位置
        for(int i=1;i<=N;++i){
            scanf("%d%d",&a,&b);
            val[i]=b;
            if(a==0)
              root[root_cnt++]=i;
            else{
                //存放边
                next[cnt]=first[a];
                first[a]=cnt;
                E[cnt]=i;
                cnt++;
            }
        }
        for(int i=0;i<root_cnt;++i)//对每个主件都进行处理，得到 root_cnt
                                    //组物品
          deal(root[i]);
        //处理之后，就成了分组背包的问题，下面解决这个分组背包
        memset(bag,0,sizeof(bag));
        for(int i=0;i<root_cnt;++i){
            int j=root[i];//第 j 种物品
            for(int v=M;v;--v)
                for(int p=0;p<=v;++p){
                    if(f[j][p]==0)continue;
                    bag[v]=max(bag[v],bag[v-p]+f[j][p]);
                }
        }
        printf("%d\n",bag[M]);
    }
    return 0;
}
```

五、线段树

1、更新节点,区间求和　　hdu1166 敌兵布阵

Input

第一行一个整数 T，表示有 T 组数据。
每组数据第一行一个正整数 N（N<=50000）,表示敌人有 N 个工兵营地，接下来有 N 个正整数,第 i 个正整数 ai 代表第 i 个工兵营地里开始时有 ai 个人（1<=ai<=50）。
接下来每行有一条命令，命令有 4 种形式：
(1) Add i j,i 和 j 为正整数,表示第 i 个营地增加 j 个人（j 不超过 30）
(2)Sub i j ,i 和 j 为正整数,表示第 i 个营地减少 j 个人（j 不超过 30）;
(3)Query i j ,i 和 j 为正整数,i<=j，表示询问第 i 到第 j 个营地的总人数;
(4)End 表示结束，这条命令在每组数据最后出现;
每组数据最多有 40000 条命令

Output

对第 i 组数据,首先输出“Case i:”和回车,
对于每个 Query 询问，输出一个整数并回车,表示询问的段中的总人数,这个数最多不超过 1000000。

Sample Input

```
1
10
1 2 3 4 5 6 7 8 9 10
Query 1 3
Add 3 6
Query 2 7
Sub 10 2
Add 6 3
Query 3 10
End
```

Sample Output

```
Case 1:
6
33
59
```

```
#include<iostream>
using namespace std;
#define LC(x) ((x)<<1)
#define RC(x) ((x)<<1|1)

struct Seg_tree
{
    int left;
    int right;
    int num;
    int calmid()
    {
        return (left+right)/2;
    }
}tree[135000];
int num[50001];

int build(int left,int right,int idx)
{
    tree[idx].left=left;
    tree[idx].right=right;
    if(left==right)
    {
        return tree[idx].num=num[left];
    }
    int mid = (left+right)/2;
    return tree[idx].num=build(left,mid,LC(idx))+build(mid+1,right,RC(idx));
}
```

```
void update(int id,int x,int idx)
{
        tree[idx].num+=x;
        if(tree[idx].right==tree[idx].left)
                return ;
        int mid= tree[idx].calmid();
        if(id<=mid)
                update(id,x,LC(idx));
        else
                update(id,x,RC(idx));
}

int query(int left,int right,int idx)
{
        if(left==tree[idx].left && tree[idx].right==right)
                return tree[idx].num;
        int mid=tree[idx].calmid();
        if(right <= mid)
        {
                return query(left,right,LC(idx));
        }
        else if(left > mid)
        {
                return query(left,right,RC(idx));
        }
        else
        {
                return query(left,mid,LC(idx))+query(mid+1,right,RC(idx));
        }
}

int main()
{
        int T;
        scanf("%d",&T);
        int cas=1;
        for(cas=1;cas<=T;++cas)
        {
                printf("Case %d:\n",cas);
                int n;
                char ch[9];
                int a,b;
                scanf("%d",&n);
                int i=0;
                for(i=1;i<=n;++i)
                {
                        scanf("%d",&num[i]);
                }

                build(1,n,1);
                while(scanf("%s",ch))
                {
                        if(ch[0]=='E')
                                break;
                        scanf("%d%d",&a,&b);
                        switch(ch[0])
                        {
                                case 'Q': printf("%d\n",query(a,b,1));break;
                                case 'A': update(a,b,1);break;
                                case 'S': update(a,-b,1);break;
                        }

                }
        }
        return 0;
}
```

2、更新节点,区间最值　 hdu1754I Hate It

Problem Description

很多学校流行一种比较的习惯。老师们很喜欢询问，从某某到某某当中，分数最高的是多少。

这让很多学生很反感。不管你喜不喜欢，现在需要你做的是，就是按照老师的要求，写一个程序，模拟老师的询问。当然，老师有时候需要更新某位同学的成绩。

Input

本题目包含多组测试，请处理到文件结束。

在每个测试的第一行，有两个正整数 N 和 M (0<N<=200000,0<M<5000)，分别代表学生的数目和操作的数目。

学生 ID 编号分别从 1 编到 N。

第二行包含 N 个整数，代表这 N 个学生的初始成绩，其中第 i 个数代表 ID 为 i 的学生的成绩。

接下来有 M 行。每一行有一个字符 C (只取'Q'或'U') ，和两个正整数 A，B。

当 C 为'Q'的时候，表示这是一条询问操作，它询问 ID 从 A 到 B(包括 A,B)的学生当中，成绩最高的是多少。

当 C 为'U'的时候，表示这是一条更新操作，要求把 ID 为 A 的学生的成绩更改为 B。

Output

对于每一次询问操作，在一行里面输出最高成绩。

Sample Input

```
5 6
1 2 3 4 5
Q 1 5
U 3 6
Q 3 4
Q 4 5
U 2 9
Q 1 5
```

Sample Output

```
5
6
5
9
```

Hint

Huge input,the C function scanf() will work better than cin

```
#include<iostream>
using namespace std;

#define LC(x) ((x)<<1)
#define RC(x) ((x)<<1|1)

struct Seg_tree
{
    int left;
    int right;
    int num;
    int calmid()
    {
        return (left+right)/2;
    }
}tree[530000];
int num[200001];

int build(int left,int right,int idx)
{
    tree[idx].left=left;
    tree[idx].right=right;
    if(left==right)
    {
        return tree[idx].num=num[left];
    }
    int mid=tree[idx].calmid();
    return tree[idx].num= max(build(left,mid,LC(idx)),build(mid+1,right,RC(idx)));
```

```
}

void update(int id,int num,int idx)
{
        if(tree[idx].left==tree[idx].right)
        {
                tree[idx].num=num;
                return;
        }
        int mid=tree[idx].calmid();

        if(id<=mid)
                update(id,num,LC(idx));
        else
        {
                update(id,num,RC(idx));
        }
        tree[idx].num = max(tree[LC(idx)].num,tree[RC(idx)].num);
}

int question(int left,int right,int idx)
{
        //cout<<left<<" "<<right<<" "<<tree[idx].left<<" "<<tree[idx].right<<endl;
        if(left==tree[idx].left && right==tree[idx].right)
                return tree[idx].num;
        int mid = tree[idx].calmid();

        if(right<=mid)
                return question(left,right,LC(idx));
        else if(mid<left)
                return question(left,right,RC(idx));
        else
                return max(question(left,mid,LC(idx)),question(mid+1,right,RC(idx)));
}
int main()
{
        int N,M;
        while((scanf("%d%d",&N,&M))!=EOF)
        {
                int i,j;
                for(i=1;i<=N;++i)
                        scanf("%d",&num[i]);

                build(1,N,1);
```

```
            char ch;
            int a,b;
            for(i=1;i<=M;++i)
            {
                scanf("%c",&ch);
                scanf("%c%d%d",&ch,&a,&b);

                if(ch=='Q')
                    printf("%d\n",question(a,b,1));
                else
                    update(a,b,1);
            }
        }
        return 0;
    }
```

3、成段更新,总区间求和　　hdu1698 Just a Hook

Problem Description

In the game of DotA, Pudge's meat hook is actually the most horrible thing for most of the heroes. The hook is made up of several consecutive metallic sticks which are of the same length.

Now Pudge wants to do some operations on the hook.
Let us number the consecutive metallic sticks of the hook from 1 to N. For each operation, Pudge can change the consecutive metallic sticks, numbered from X to Y, into cupreous sticks, silver sticks or golden sticks.
The total value of the hook is calculated as the sum of values of N metallic sticks. More precisely, the value for each kind of stick is calculated as follows:

For each cupreous stick, the value is 1.
For each silver stick, the value is 2.
For each golden stick, the value is 3.

Pudge wants to know the total value of the hook after performing the operations.
You may consider the original hook is made up of cupreous sticks.

Input

The input consists of several test cases. The first line of the input is the number of the cases. There are no more than 10 cases.
For each case, the first line contains an integer N, 1<=N<=100,000, which is the number of the sticks of Pudge's meat hook and the second line contains an integer Q, 0<=Q<=100,000, which is the number of the operations.

Next Q lines, each line contains three integers X, Y, 1<=X<=Y<=N, Z, 1<=Z<=3, which defines an operation: change the sticks numbered from X to Y into the metal kind Z, where Z=1 represents the cupreous kind, Z=2 represents the silver kind and Z=3 represents the golden kind.

Output

For each case, print a number in a line representing the total value of the hook after the operations. Use the format in the example.

Sample Input

```
1
10
2
1 5 2
5 9 3
```

Sample Output

```
Case 1: The total value of the hook is 24.
```

```
#include<iostream>
using namespace std;
#define LC(x) (x<<1)
#define RC(x) (x<<1|1)

int n;
struct Seg_tree
{
   int left;
   int right;
   int col;
   int calmid()
   {
         return (left+right)/2;
   }
}tree[300000];

void build(int left,int right,int idx)
{
   tree[idx].left=left;
   tree[idx].right=right;
   tree[idx].col=1;
   if(right==left)
         return;
   int mid = tree[idx].calmid();
```

```
    build(left,mid,LC(idx));
    build(mid+1,right,RC(idx));
}
void update(int left,int right,int col,int idx)
{
    if(left<=tree[idx].left && right>=tree[idx].right)
    {
            tree[idx].col=col;
            return ;
    }
    if(tree[idx].col != -1)
    {
            tree[LC(idx)].col = tree[RC(idx)].col =tree[idx].col;
            tree[idx].col= -1;
    }
    int mid=tree[idx].calmid();
    if(left <= mid)
            update(left,right,col,LC(idx));
    if(mid<right)
            update(left,right,col,RC(idx));
}

int query(int left,int right,int idx)
{
    int mid = tree[idx].calmid();
    if(left == tree[idx].left && right ==tree[idx].right)
    {
            if(tree[idx].col!=-1)
                    return (right-left+1)*tree[idx].col;
            else
                    return query(left,mid,LC(idx))+query(mid+1,right,RC(idx));

    }
    if(right<=mid)
            return query(left,right,LC(idx));
    else if(left>mid)
            return query(left,right,RC(idx));
    else
            return query(left,mid,LC(idx))+query(mid+1,right,RC(idx));
}
int main()
{
    int T;
    int cas;
    scanf("%d",&T);
    for(cas=1;cas<=T;++cas)
    {
        int m,n;
        scanf("%d",&n);
        build(1,n,1)
;
        scanf("%d",&m);
        int left,right,cal;
        while(m--)
        {
            scanf("%d%d%d",&left,&right,&cal);
            update(left,right,cal,1);
        }
        printf("Case %d: The total value of the hook is %d.\n",cas,query(1,n,1));
    }
    return 0;
}
```

4、更新节点,区间求和 hdu1394 Minimum Inversion Number

Problem Description

The inversion number of a given number sequence a1, a2, ..., an is the number of pairs (ai, aj) that satisfy i < j and ai > aj.

For a given sequence of numbers a1, a2, ..., an, if we move the first m >= 0 numbers to the end of the seqence, we will obtain another sequence. There are totally n such sequences as the following:

a1, a2, ..., an-1, an (where m = 0 - the initial seqence)

a2, a3, ..., an, a1 (where m = 1)

a3, a4, ..., an, a1, a2 (where m = 2)

...

an, a1, a2, ..., an-1 (where m = n-1)

You are asked to write a program to find the minimum inversion number out of the above sequences.

Input

The input consists of a number of test cases. Each case consists of two lines: the first line contains a positive integer n (n <= 5000); the next line contains a permutation of the n integers from 0 to n-1.

Output

For each case, output the minimum inversion number on a single line.

Sample Input

```
10
1 3 6 9 0 8 5 7 4 2
```

Sample Output

```
    16
    //(实现 nlog(n)的逆序数方法, 和树状数组比起来实在是有点鸡肋)
struct Seg_Tree {
    int left,right,val;
    int calmid() {
        return (left+right)/2;
    }
}tt[15000];
int val[5001];

void build(int left,int right,int idx) {
    tt[idx].left = left;
    tt[idx].right = right;
    tt[idx].val = 0;
    if(left == right) {
        return ;
    }
    int mid = tt[idx].calmid();
    build(left,mid,LL(idx));
    build(mid+1,right,RR(idx));
}

int query(int left,int right,int idx) {
    if(left == tt[idx].left && right == tt[idx].right) {
        return tt[idx].val;
    }
    int mid = tt[idx].calmid();
    if(right <= mid) {
        return query(left,right,LL(idx));
    } else if(mid < left) {
        return query(left,right,RR(idx));
    } else {
        return query(left,mid,LL(idx)) + query(mid+1,right,RR(idx));
    }
}

void update(int id,int idx) {
    tt[idx].val ++;
    if(tt[idx].left == tt[idx].right) {
        return ;
    }
    int mid = tt[idx].calmid();
    if(id <= mid) {
        update(id,LL(idx));
    } else {
            update(id,RR(idx));
    }
}

int main() {
    int n;
    while(scanf("%d",&n) == 1) {
            build(0,n-1,1);
            int sum = 0;
            FF(i,n) {
                    scanf("%d",&val[i]);
                    sum += query(val[i],n-1,1);
                    update(val[i],1);
            }
            int ret = sum;
            FF(i,n) {
                    sum = sum - val[i] + (n - val[i] - 1);
                    checkmin(ret,sum);
            }
            printf("%d\n",ret);
    }
    return 0;
}
```

5、成段更新,寻找空间(经典类型,求一块满足条件的最左边的空间) pku3667 Hotel

Description

The cows are journeying north to Thunder Bay in Canada to gain cultural enrichment and enjoy a vacation on the sunny shores of Lake Superior. Bessie, ever the competent travel agent, has named the Bullmoose Hotel on famed Cumberland Street as their vacation residence. This immense hotel has N ($1 \le N \le 50,000$) rooms all located on the same side of an extremely long hallway (all the better to see the lake, of course).

The cows and other visitors arrive in groups of size D_i ($1 \le D_i \le N$) and approach the front desk to check in. Each group i requests a set of D_i contiguous rooms from Canmuu, the moose staffing the counter. He assigns them some set of consecutive room numbers $r..r+D_i-1$ if they are available or, if no contiguous set of rooms is available, politely suggests alternate lodging. Canmuu always chooses the value of r to be the smallest possible.

Visitors also depart the hotel from groups of contiguous rooms. Checkout i has the parameters X_i and D_i which specify the vacating of rooms X_i ..X_i +D_i-1 ($1 \leq X_i \leq N\text{-}D_i+1$). Some (or all) of those rooms might be empty before the checkout.

Your job is to assist Canmuu by processing M ($1 \leq M < 50{,}000$) checkin/checkout requests. The hotel is initially unoccupied.

Input

* Line 1: Two space-separated integers: N and M
* Lines 2..M+1: Line i+1 contains request expressed as one of two possible formats: (a) Two space separated integers representing a check-in request: 1 and D_i (b) Three space-separated integers representing a check-out: 2, X_i, and D_i

Output

* Lines 1.....: For each check-in request, output a single line with a single integer r, the first room in the contiguous sequence of rooms to be occupied. If the request cannot be satisfied, output 0.

Sample Input

```
10 6
1 3
1 3
1 3
1 3
2 5 5
1 6
```

Sample Output

```
1
4
7
0
```

5

```
struct Seg_Tree{
    int left,right;
    int cval,lval,rval;
    int st;
    int mid() {
            return (left+right)>>1;
    }
    void doit() {
            cval = lval = rval = st ? 0 : dis();
    }
    int dis() {
            return (right - left + 1);
    }
}tt[150000];

void build(int l,int r,int idx) {
    tt[idx].left = l;
    tt[idx].right = r;
    tt[idx].st = -1;
    tt[idx].cval = tt[idx].lval = tt[idx].rval = tt[idx].dis();
    if(l == r)  return ;
    int mid = tt[idx].mid();
    build(l,mid,LL(idx));
    build(mid+1,r,RR(idx));
}

void update(int l,int r,int st,int idx) {
    if(tt[idx].left >= l && r >= tt[idx].right) {
            tt[idx].st = st;
            tt[idx].doit();
            return ;
    }
    if(tt[idx].st != -1) {
            tt[LL(idx)].st = tt[RR(idx)].st = tt[idx].st;
            tt[LL(idx)].doit();
            tt[RR(idx)].doit();
            tt[idx].st = -1;
    }
    int mid = tt[idx].mid();
    if(l <= mid) update(l,r,st,LL(idx));
    if(mid < r)       update(l,r,st,RR(idx));

    tt[idx].cval          =          max(          tt[LL(idx)].rval          +          tt[RR(idx)].lval          ,
max(tt[LL(idx)].cval,tt[RR(idx)].cval) );
    tt[idx].lval = tt[LL(idx)].lval;
    tt[idx].rval = tt[RR(idx)].rval;
    if(tt[LL(idx)].cval == tt[LL(idx)].dis()) {
            tt[idx].lval += tt[RR(idx)].lval;
    }
    if(tt[RR(idx)].cval == tt[RR(idx)].dis()) {
            tt[idx].rval += tt[LL(idx)].rval;
    }
}

int query(int w,int idx) {
    if(tt[idx].left == tt[idx].right && w == 1) {
            return tt[idx].left;
    }
    if(tt[idx].st != -1) {
            tt[LL(idx)].st = tt[RR(idx)].st = tt[idx].st;
            tt[LL(idx)].doit();
            tt[RR(idx)].doit();
            tt[idx].st = -1;
    }
    if(tt[LL(idx)].cval >= w) {
            return query(w,LL(idx));
    } else if(tt[LL(idx)].rval + tt[RR(idx)].lval >= w) {
            return tt[LL(idx)].right - tt[LL(idx)].rval + 1;
    } else if(tt[RR(idx)].cval >= w){
            return query(w,RR(idx));
    } else {
            return 0;
    }
}

int main() {
    int n , m;
    while(scanf("%d%d",&n,&m) == 2) {
            build(1,n,1);
            while(m --) {
                    int com;
                    scanf("%d",&com);
                    if(com == 1) {
                            int a;
                            scanf("%d",&a);
                            int s = query(a,1);
                            printf("%d\n",s);
                    if(s) {
                        update(s,s+a-1,1,1);
                    }
                } else {
                    int a,b;
                    scanf("%d%d",&a,&b);
                    update(a,a+b-1,0,1);
                }
            }
        }
        return 0;
}
```

6、矩形面积并,扫描线法 hdu1542 Atlantis

```
#include<iostream>
#include<algorithm>
using namespace std;
#define LC(x) (x<<1)
#define RC(x) (x<<1|1)

struct Seg_tree
{
    int left,right;
    int cnt;
    double sum;
    int calmid()
    {
        return (left+right)>>1;
    }
}tree[6666];

struct Seg
{
    double high;
    double left,right;
    int side;
}s[2222];

bool cmp(Seg &a,Seg &b)
{
    return a.high<b.high;
}
```

```
double pos[2222];

void build(int l,int r,int idx)
{
    tree[idx].left = l;
    tree[idx].right=r;
    tree[idx].cnt=0;
    tree[idx].sum=0;
    if(l==r)
        return ;
    int mid = tree[idx].calmid();
    build(l,mid,LC(idx));
    build(mid+1,r,RC(idx));
}
void update(int idx)
{
    if(tree[idx].cnt)
    {
        tree[idx].sum = pos[tree[idx].right+1] - pos[tree[idx].left];
    }
    else if(tree[idx].left == tree[idx].right)
        tree[idx].sum=0;
    else
        tree[idx].sum = tree[LC(idx)].sum + tree[RC(idx)].sum;
}

void update(int l,int r,int st,int idx)
{
    if(l<=tree[idx].left && r>=tree[idx].right)
    {
        tree[idx].cnt+=st;
        update(idx);
        return;
    }
    int mid = tree[idx].calmid();
    if(l <= mid)
        update(l,r,st,LC(idx));
    if(mid<r)
        update(l,r,st,RC(idx));
    update(idx);
}

int Bin(double  x,int hi)
```

```
{
    int lo=0;
    while(lo <= hi)
    {
        int mid = (lo+hi)>>1;
        if(pos[mid]== x)
            return mid;
        if(pos[mid]<x)
            lo = mid +1;
        else
            hi = mid -1;
    }
    return -1;
}

int main()
{
    int n;
    int cas=1;
    while(scanf("%d",&n) == 1)
    {
        if(n==0)
            break;
        int m=0,i=0;
        for(i=0;i<n;++i)
        {
            double a,b,c,d;
            scanf("%lf%lf%lf%lf",&a,&b,&c,&d);
            pos[m]=s[m].left=s[m+1].left=a;
            s[m].high=b;
            pos[m+1]=s[m].right=s[m+1].right=c;
            s[m+1].high=d;
            s[m].side=1;
            s[m+1].side=-1;
            m+=2;
        }
        sort(s,s+m,cmp);
        sort(pos,pos+m);
        for(i=0;i<4;++i)
            cout<<s[i].left<<" "<<s[i].right<<" "<<s[i].high<<" "<<s[i].side<<endl;;
        cout<<endl;

        int k=0;
        for(i=0;i<m;++i)
```

```
            {
                if(i ==0 || pos[i] !=pos[i-1])
                {
                    pos[k++]=pos[i];
                }
            }
            for(i=0;i<k;++i)
                cout<<pos[i]<<" ";
            cout<<endl;

            build(0,k-1,1);

            double ret=0;
            for(i=0;i<m-1;++i)
            {
                int l=Bin(s[i].left,k-1);
                int r=Bin(s[i].right,k-1)-1;
                cout<<l<<" "<<r<<endl;
                if(l>r)
                {
                    ret+=tree[1].sum * (s[i+1].high-s[i].high);
                    continue;
                }
                update(l,r,s[i].side,1);
                for(int ij=1;ij<8;++ij)
                    cout<<tree[ij].sum<<" ";
                cout<<endl;
                ret+=tree[1].sum * (s[i+1].high-s[i].high);
                cout<<tree[1].sum * (s[i+1].high-s[i].high)<<endl;
            }
            printf("Test case #%d\n",cas++);
            printf("Total explored area: %.2lf\n\n",ret);
        }
    }
```

7. 成段更新,区间统计,位运算加速　pku2777 Count Color

Description

Chosen Problem Solving and Program design as an optional course, you are required to solve all kinds of problems. Here, we get a new problem.
There is a very long board with length L centimeter, L is a positive integer, so we can evenly divide the board into L segments, and they are labeled by 1,

2, ... L from left to right, each is 1 centimeter long. Now we have to color the board - one segment with only one color. We can do following two operations on the board:
1. "C A B C" Color the board from segment A to segment B with color C.
2. "P A B" Output the number of different colors painted between segment A and segment B (including).
In our daily life, we have very few words to describe a color (red, green, blue, yellow…), so you may assume that the total number of different colors T is very small. To make it simple, we express the names of colors as color 1, color 2, ... color T. At the beginning, the board was painted in color 1. Now the rest of problem is left to your.

Input

First line of input contains L (1 <= L <= 100000), T (1 <= T <= 30) and O (1 <= O <= 100000). Here O denotes the number of operations. Following O lines, each contains "C A B C" or "P A B" (here A, B, C are integers, and A may be larger than B) as an operation defined previously.

Output

Ouput results of the output operation in order, each line contains a number.

Sample Input

```
2 2 4
C 1 1 2
P 1 2
C 2 2 2
P 1 2
```

Sample Output

```
2
1
```

```
#include<iostream>
using namespace std;
#define MAXN 1000000

typedef struct
```

```
{
        int l;
        int r;
        int pl;
        int pr;
        int co;
}Node;
Node node[MAXN];

int n;

int simple(int  a)    //判断是否是单颜色，如果是单一的则返回
{
        if(a&(a-1))
                return 0;
        else
                return 1;
}

void build(int  l,int  r)
{
        int v;
        n++;
        v=n;
        node[v].l=l;
        node[v].r=r;
        node[v].co=1;
        if(l<r)
        {
                node[v].pl=n+1;
                build(l,(l+r)/2);
                node[v].pr=n+1;
                build((l+r)/2+1,r);
                return;
        }
}
void insert(int c,int d,int v,int co)
{
        if(c<=node[v].l && d>=node[v].r)
        {
                node[v].co=co;
                return ;
        }
        if(simple(node[v].co))
```

```
		node[node[v].pr].co=node[node[v].pl].co=node[v].co;
	if(c<=(node[v].l+node[v].r)/2)
		insert(c,d,node[v].pl,co);
	if(d>(node[v].l+node[v].r)/2)
		insert(c,d,node[v].pr,co);
	node[v].co=(node[node[v].pl].co)|(node[node[v].pr].co);
}

void swap(int *a,int *b)
{
	int t;
	t=*a;
	*a=*b;
	*b=t;
}

int    cal(int a)
{
	int count=0;
	while(a)
	{
		count+=a%2;
		a>>=1;
	}
	return count;
}

void f(int a,int b,int v,int *count)
{
	if(a<=node[v].l && b>=node[v].r || simple(node[v].co))
	{
		*count=(*count)|node[v].co;
		return ;
	}
	if(a<=(node[v].l+node[v].r)/2)
		f(a,b,node[v].pl,count);
	if(b>(node[v].l+node[v].r)/2)
		f(a,b,node[v].pr,count);
}

int main()
{
	int t,l,o,a,b,co,count;
	char ch[10];
	scanf("%d%d%d",&l,&t,&o);
	n=0;
	build(1,l);
	while(o--)
	{
		scanf("%s",ch);
		if(ch[0]=='C')
		{
			scanf("%d%d%d",&a,&b,&co);
			if(a>b)
				swap(&a,&b);
			insert(a,b,1,1<<(co-1));
		}
		else
		{
			scanf("%d%d",&a,&b);
			if(a>b)
				swap(&a,&b);
			count=0;
			f(a,b,1,&count);
			count=cal(count);
			printf("%d\n",count);
		}
	}
	return 0;
}
```

六、字典树

```
//hdu1251 统计难题
#include<iostream>
using namespace std;
#define Max 1000100
struct Node
{
	int count;
	bool is_word;
	Node * node[26];
	void init()
	{
		count=0;
		is_word = false;
```

```
        int i;
        for(i=0;i<26;i++)
            node[i]=NULL;
        return ;
    }
};
Node arr[Max+10];
Node *root;
char S[12];
int countnum;

int tire_search(const char * str)
{

    Node *temp=root;
    int i;
    int len=strlen(str);
    for(i=0;i<len;i++)
    {
        if( temp->node[str[i]-'a'] ==NULL )
            return 0;
        temp = temp->node[str[i]-'a'];
    }
    return temp->count;
}

void trie_insert(const char *str)
{
    if(root == NULL)
    {
        root =&arr[countnum];
        countnum++;
        //root = new Node;
        root->init();
    }
    Node * temp=root;
    int len=strlen(str);
    int i;
    for(i=0;i<len;i++)
    {
        int site = str[i]-'a';

        if(temp->node[site] == NULL)
        {
                if(countnum>Max)
                    temp->node[site] =new Node;
                else
                {
                    temp->node[site] =&arr[countnum];
                    countnum++;
                }
                temp=temp->node[site];
                temp->init();
                (temp->count)++;
            }
            else
            {
                temp = temp->node[site];
                (temp->count) ++;
            }
        }
        temp->is_word = true;
}

int main()
{
        int num=0;
        while(gets(S))
        {
            if(strlen(S)==0)
                break;
            trie_insert(S);
        }
        while(gets(S)!=0)
        {
            num=tire_search(S);
            printf("%d\n",num);
        }
        return 0;

}
```

七、图论

1、求强联通分支

```
#include<iostream>
using namespace std;
int n,m;//n为点的个数，m为边的个数
const int MAX=10002;
struct Node
{
    int to;
    Node* next;
    ~Node(){delete next;}
};
Node* map[MAX];
Node* reverse_map[MAX];
int f[MAX],id[MAX];//out记录连通分量的出度，id记点属于哪个连通
分量，f记录点深搜的完成时间
bool used[MAX],out[MAX];
int sccn,ftime;
void dfs1(int u)//第一次深搜形成f数组
{
    used[u]=true;
    for(Node *t=map[u];t;t=t->next)
        if(!used[t->to])
            dfs1(t->to);
    f[++ftime]=u;
}
void dfs2(int u)//按f递减的顺序对转置图进行深搜
{
    used[u]=true;
    id[u]=sccn;
    for(Node *t=reverse_map[u];t;t=t->next)
    {
        if(!used[t->to])
            dfs2(t->to);
        if(id[t->to]&&id[t->to]!=sccn)
            out[id[t->to]]++;
    }
}
void Kosaraju()
{
    memset(out,false,sizeof(out));
    memset(id,0,sizeof(id));
    int i;
    sccn=0,ftime=0;
    memset(used,false,sizeof(used));
    for(i=1;i<=n;i++)
        if(!used[i])
            dfs1(i);
    memset(used,false,sizeof(used));
    for(i=ftime;i>0;i--)
        if(!used[f[i]])
        {
            ++sccn;
            dfs2(f[i]);
        }
}
int main()
{
    while(scanf("%d%d",&n,&m)!=EOF)
    {
        memset(map,0,sizeof(map));
        memset(reverse_map,0,sizeof(reverse_map));
        int from,to,i;
        for(i=0;i<m;i++)
        {
            scanf("%d%d",&from,&to);
            Node* temp=new Node;
            temp->to=to;
            temp->next=map[from];
            map[from]=temp;

            Node* t=new Node;
            t->to=from;
            t->next=reverse_map[to];
            reverse_map[to]=t;
        }
        Kosaraju();
        int res=0,temp=-1;
        bool flag=true;
        for(i=1;i<=n;i++)
        {
            if(!out[id[i]])
            {
                res++;
                if(temp==-1)
                    temp=id[i];
                else if(temp!=id[i])
                    flag=false;
            }
        }
        if(flag)
            printf("%d\n",res);
        else
            printf("%d\n",0);
    }
}
```

求满足（v能到达的点w都能到达v）的点的集合，即求强连通分量，最后输出出度为零的点

```
#include<iostream>
#include<stack>
using namespace std;
const int MAX=5002;
int graph[MAX][MAX];//邻接表存储图
int pre[MAX],low[MAX],id[MAX];
bool out[MAX];//记录点所在的连通分量号
int dindex=0;//记时间戳用
int sccn=0,n,m;//记连通分量个数
stack<int>s;
void Tarjan(int u)
{
    int v,i;
    int minx=pre[u]=low[u]=dindex++;//为节点i设定次序编号和low初值
    s.push(u);//将节点压入栈中
    for(i=1;i<=graph[u][0];i++)//枚举每一条边
    {
        v=graph[u][i];
        if(pre[v]==-1)
            Tarjan(v);
        if(low[v]<minx)
            minx=low[v];
    }
    if(minx<low[u])
    {
        low[u]=minx;
```

```
            return;
        }
        sccn++;
        do
        {
            low[v=s.top()]=m;//将v退栈，为强连通分量中的一个顶点
            s.pop();
            id[v]=sccn;
        }while(v!=u);
    }
    void Component()
    {
        sccn=0,dindex=0;
        memset(pre,-1,sizeof(pre));
        for(int i=1;i<=m;i++)
            if(pre[i]==-1)
                Tarjan(i);
    }
    void dfs(int u)
    {
        pre[u]=1;
        for(int i=1;i<=graph[u][0];i++)
        {
            int v=graph[u][i];
            if(id[v]!=id[u])
                out[id[u]]=true;
            if(!pre[v])
                dfs(v);
        }
    }
    void DFS()
    {
        memset(out,false,sizeof(out));
        memset(pre,0,sizeof(pre));
        for(int i=1;i<=m;i++)
            if(!pre[i])
                dfs(i);
    }
    int main()
    {
        while(scanf("%d",&m)!=EOF&&m)//n表示边的个数，m表示点的个数
        {
            scanf("%d",&n);
            int i,from,to;
            for(i=1;i<=m;i++)//此处一定不能用memset赋值！！会超内存！
                graph[i][0]=0;
            for(i=0;i<n;i++)
            {
                scanf("%d%d",&from,&to);
                graph[from][++graph[from][0]]=to;
            }
            Component();
            DFS();
            for(i=1;i<=m;i++)
                if(!out[id[i]])
                    printf("%d ",i);
            printf("\n");
        }
        return 0;
    }
```

2、判断是否存在欧拉回路

```
#include<iostream>
#include<queue>
using namespace std;

bool map[210][210];
int degree[210];
bool mark[210];
int n,m;

bool Bfs(int k)
{
    queue<int> Q;
    Q.push(k);
    mark[k]=false;
    int temp;
    int i;
    int count=0;
    while(!Q.empty())
    {
        temp= Q.front();
        Q.pop();
        count++;
        for(i=1;i<210;i++)
        {
            if(map[temp][i] && mark[i]==true)
```

```
                {
                        Q.push(i);
                        mark[i]=false;
                }
            }
        }
        if(count==n)
            return true;
        else
            return false;
}

int main()
{
        int T;
        int i,j;
        int a,b;
        scanf("%d",&T);
        while(T--)
        {
                memset(degree,0,sizeof(degree));
                memset(map,false,sizeof(map));
                memset(mark,false,sizeof(mark));
                scanf("%d %d",&n,&m);
                for(i=1;i<=m;i++)
                {
                        scanf("%d %d",&a,&b);
                        map[a][b]=map[b][a]=true;
                        mark[a]=mark[b]=true;
                        degree[a]++;
                        degree[b]++;
                }
                if( !Bfs(1) )
                {
                        printf("no\n");
                        continue;
                }
                int count=0;
                for(i=1;i<=n;i++)
                {
                        if(degree[i]&1)
                                count++;
                }
                if(count>0)
                printf("no\n");

            else
                printf("yes\n");
        }
        return 0;
}
```

八、差分约束系统

Description

You are given n closed, integer intervals [ai, bi] and n integers c1, ..., cn.
Write a program that:
reads the number of intervals, their end points and integers c1, ..., cn from the standard input,
computes the minimal size of a set Z of integers which has at least ci common elements with interval [ai, bi], for each i=1,2,...,n,
writes the answer to the standard output.

Input

The first line of the input contains an integer n (1 <= n <= 50000) -- the number of intervals. The following n lines describe the intervals. The (i+1)-th line of the input contains three integers ai, bi and ci separated by single spaces and such that 0 <= ai <= bi <= 50000 and 1 <= ci <= bi - ai+1.

Output

The output contains exactly one integer equal to the minimal size of set Z sharing at least ci elements with interval [ai, bi], for each i=1,2,...,n.

Sample Input

```
5
3 7 3
8 10 3
6 8 1
```

```
1 3 1
10 11 1
```

Sample Output

```
6
```

1、题目类型：差分约束系统、SPFA 算法。

2、解题思路：（1）根据输入输入构图，并非简单的对输入进行构图，根据差分约束系统的理解，每个结束节点+1,而且第 i 个节点到第 i+1 个节点的权值为 0、第 i+1 个节点到第 i 个节点的权值为-1；（2）SPAF 算法寻找最短路径，是该路径的权值最大，初始状态下为-inf。

3、注意事项：构图是解决差分约束系统这类题的关键，根据择优的不同，合适的选择 SPFA 的判断条件。

4、实现方法：

```
#include<iostream>
#include<vector>
#include<algorithm>
#include<queue>
using namespace std;
#define inf -10000000

struct Edge
{
    int v,c;
};

vector<Edge> head[50010];
int n,MaxP;
int mindis[50010];

void Init()
{
    int i,a,b,c;
    Edge E;
    MaxP=-1;
    cin>>n;
    for(i=0;i<n;i++)
    {
        scanf("%d%d%d",&a,&b,&c);
        if(a>b)
            swap(a,b);
        E.v=b+1;
        E.c=c;
        head[a].push_back(E);
        if(MaxP<b)
            MaxP=b;
    }
    for(i=0;i<=MaxP;i++)
    {
        E.v=i;
        E.c=-1;
        head[i+1].push_back(E);
        E.v=i+1;
        E.c=0;
        head[i].push_back(E);
        mindis[i]=inf;
    }
    mindis[0]=0;
    mindis[MaxP+1]=inf;
}

void SPFA()
{
    queue<int> Q;
    int vis[50010]={0};
    Q.push(0);
    vis[0]=1;
    while(!Q.empty())
    {
        int u=Q.front();
        Q.pop();
        vis[u]=0;
        for(int i=0;i<head[u].size();i++)
        {
            if(mindis[head[u][i].v]<mindis[u]+head[u][i].c)
            {
                mindis[head[u][i].v]=mindis[u]+head[u][i].c;
                if(!vis[head[u][i].v])
                {
                    vis[head[u][i].v]=1;
                    Q.push(head[u][i].v);
                    }
                }
            }
        }
}

void Solve()
{
    SPFA();
    cout<<mindis[MaxP+1]<<endl;
}

int main()
{
    Init();
    Solve();
    return 0;
}
```

九、贪心　导弹拦截 hdu1257

```
#include<iostream>
using namespace std;
struct Node
{
    int high;
    bool flag;
};

Node f[1001];
int main()
{
    int n;
    while(cin>>n)
    {
        int i=0;
        for(i=0;i<n;++i)
        {
            cin>>f[i].high;
            f[i].flag=true;
        }
        int j=n;
        int count=0;
        while(j)                    //j 用于记录当前未拦截的导弹数目
        {
            int temp=999999999;
            count++;
            for(i=0;i<n;++i)
            {
                if(f[i].flag&&temp>f[i].high)
                             //如果 i 这颗导弹未拦截，并且它的高度小于前面的
高度
                {
                    temp=f[i].high;
                    f[i].flag=false;     //将 i 这个导弹拦截
                    j--;
                }
            }
        }
        cout<<count<<endl;
    }
}
```