The code library of SOWHAT

Version 1.2

Last update at 2011.10

Zhengzhou University

Collated by zxy , dwl , cw

Content

1 头文件 && 宏

```
#include <set>
#include <map>
#include <queue>
#include <stack>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <limits.h>
#include <string.h>
#include <string>
#include <algorithm>
#define MID(x,y) ( ( x + y ) >> 1 )
#define L(x) ( x << 1 )
#define R(x) ( x << 1 | 1 )
#define FOR(i,s,t) for(int i=(s); i<(t); i++)
#define BUG puts("here!!!")
#define STOP system("pause")

using namespace std;

int main()
{

    return 0;
}
```

2 数论

```
typedef long long LL;// 可以根据题目要求调整 LL 的类型
```

2.1 常用公式

```
    n!=(n/e)*n * sqrt(2*pi*n)
```

2.2 欧几里得算法

```
LL gcd(LL a,LL b) {
    return b==0?a:gcd(b,a%b);
}
```

2.3 快速欧几里得

```
LL kgcd( LL a,LL b ){
    if(!a)return b;
    if(!b)return a;
    if( !(a&1) && !(b&1) ) return kgcd(a/2,b/2)<<1;
    else if( !(b&1) ) return kgcd( a,b>>1 );
    else if( !(a&1) ) return kgcd( a>>1,b );
    else return kgcd( abs(a-b),min(a,b) );
}
```

```
//计算机中的负数模(a%n)与数论中的不一样，需要转换
LL rightmod(LL a,LL n){
    return (n+a%n)%n;
}
```

2.4 扩展欧几里得

```
LL ext_euclid(LL a,LL b,LL &x,LL &y){
    LL t,d;
    if( b == 0 ){
        x = 1;
        y = 0;
        return a;
    }
    d = ext_euclid(b,a%b,x,y);
    t = x;
    x = y;
    y = t - (a/b)*y;
    return d;
}
```

2.5 输出模线性方程 ax=b(modn)的最小解（所有解）

```
LL modf(LL a,LL b,LL n){
    LL x,y;
    LL d = ext_euclid(a,n,x,y);
    if( b%d ) cout << "FOREVER\n";
    else
    {
        x = rightmod(x*b/d,n);
        //for(i=0;i<d;i++) print x+i(n/d)所有解
        return rightmod(x,n/d);
    }return 0;
}
```

2.6 中国剩余定理

a 数组表示 ai，n 表示互质的 ni 数组，k 为个数

```
LL CRT(LL *a,LL *n,LL k){
    LL i,N = 1,ans=0;
    LL t;
    for( i = 0; i < k; i++ )
        N *= n[i];
    for( i = 0; i < k; i++ )
    {
        t = N/n[i]*modf(N/n[i],1,n[i]);
        ans = (ans + a[i]*t)%N;
    }
    return ans;
}
```

2.7 快速幂模

求 b=a^k%M

```
LL fastmod(LL a,LL k,LL M){
    LL b = 1;
```

```
    while( k ){
        if( k&1 )
            b = a*b%M;
        a = (a%M)*(a%M)%M;
        k /= 2;
    }
    return b;
}
```

2.8 欧拉函数

```
LL euler(LL n ){
    LL i,m = (int)sqrt( n + 0.5 ),ans = n;
    for( i = 2; i <= m; i++ )
        if( n%i == 0 )
        {
            ans = ans/i*(i-1);
            while( n%i == 0 ) n /= i;
        }
    if( n > 1 ) ans = ans/n*(n-1);
    return ans;
}
```

2.9 质因子分解

count 表示质因子个数,d[i][0]和 d[i][1]分别表示第 i 个质因子和个数

```
LL d[100][2];
LL divprime(LL n){
    LL i,count=0,k=n;
    for( i = 2; i*i <= k; i++ )
        if( n%i == 0 ){
            d[count][0] = i;
            d[count][1] = 0;
            while( n%i == 0 )
                n/=i,d[count][1]++;
            count++;
        }
    if( n > 1 )
        d[count][0] = n,d[count][1] = 1,count++;
    return count;
}

//搜索质因子的各种组合，即找到 n 所有约数
//调用方式 finddiv(0,1,n,count)
//count 为因子个数,变量 ji 保存的是每次搜到的约数
void finddiv( LL ceng,LL ji,LL n,LL count ){
    if( ceng == count )
    {
        cout<<ji<<endl;
        return;
    }
    LL i,m;
    for( i = 0,m = 1; i <= d[ceng][1] ; i++ )
        finddiv( ceng+1,ji*m,n,count),m *= d[ceng][0];
}
```

2.10 素数线性筛法+质因子个数和

```
const int SULEN = 1000000;
bool flag[SULEN+1];//是否为素数
int prime[1000001];
int sum[SULEN+1];//n 的所有质因子的和
void xianxingshai(void)
{
    int count,i,j;
    fill( flag,flag+SULEN,true );
    fill( sum ,sum +SULEN, 0 );
    flag[0] = flag[1] = false;
    for( count = 0, i = 2; i <= SULEN; i++ )
    {
        if( flag[i] )
        {
            prime[count++] = i;sum[i] = i;
        }
        for( j = 0; j < count && i*prime[j] <= SULEN; j++ )
        {
            flag[ i*prime[j] ] = false;
            if( i%prime[j] == 0 ){
                sum[ i*prime[j] ] = sum[i];
                break;
            }
            else
                sum[ i*prime[j] ] = sum[ i ] + prime[j];
        }
    }
}
//nlogn 素数筛法
int isprime[1000000];
void nlognprime( int n ){
    fill( isprime,isprime+1000000,1 );
    isprime[0] = isprime[1] = 0;
    int i,j,m= (int)sqrt( n+0.5);
    for( i= 2; i <= m; i++ )
        if( isprime[i] )
        for( j = i*i; j <= n; j+=i )
            isprime[j] = 0;
}
```

2.11 **miller rabin** 素数测试

```
LL witness( LL a,LL n ){
    LL u = n-1,t = 0;
    while( !(u&1) ) {u>>=1;t++;}
    int x1,x0 = fastmod(a,u,n);
    for( int i = 1; i <= t; i++ ){
        x1 = x0*x0%n;
        if( x1 == 1 && x0 != 1 && x0 != n-1 )
            return 1;
        x0 = x1;
    }
    if( x0 != 1 ) return 1;
    else return 0;
}
LL miller_rabin(LL n ,LL s ){//s 为测试次数
    if( n==2 ) return 1;
```

```
    if( n%2==0 || n < 2 ) return 0;
    int j,a;
    for( j = 1; j <= s; j++ ){
        a = rand()*(n-1)/RAND_MAX + 1;
        if( witness(a,n) ) return 0;
    }
    return 1;
}


//n 的阶乘中素因子 a 的幂
int aofn( int a,int n ){
    int count = 0;
    while( n ){
    n/=a;
    count += n;
}
return count;
}
```

2.12 （待补充）快速斐波那契数

3 图论—最短路径

3.1 Dijkstra

```
/*
***********************************************
最短路：dijkstra (邻接矩阵表示)
n 表示 点数，MAX 是 n 的上界+10
***********************************************
*/

int map[MAX][MAX];
int Dijkstra(int from,int to,int n)  // DIJ + 邻接矩阵
{
    int dis[MAX];
    bool used[MAX];
    memset(used,false,sizeof(used));
    for(int i=1; i<=n; i++)
        dis[i] = INT_MAX;
    dis[from] = 0;
    used[from] = true;
    int now = from;
    for(int i=1; i<n; i++)
    {
        for(int k=1; k<=n; k++)
            if( map[now][k] && dis[k] > dis[now] + map[now][k] )
                dis[k] = dis[now] + map[now][k];

        int min = INT_MAX;
        for(int k=1; k<=n; k++)
            if( !used[k] && dis[k] < min )
                min = dis[now = k];
        used[now] = true;
```

```
    }
    return dis[to];
}

/*
**********************************************
最短路：dijkstra (邻接表表示)
n 表示 点数，MAX 是 n 的上界+10
**********************************************
*/

typedef struct NODE{
    int to,len;
    NODE *next;
}NODE;
NODE *p[MAX],node[MAX*MAX];

void Add(int from,int to,int len)
{
    node[cou].next = p[from];
    node[cou].to = to;
    node[cou].len = len;
    p[from] = &node[cou++];
}

int Dijkstra_List(int from,int to,int n)
{
    int dis[MAX];
    bool used[MAX];
    memset(used,false,sizeof(used));
    for(int i=1; i<=n; i++)
        dis[i] = INT_MAX;
    dis[from] = 0;
    used[from] = true;
    int now = from;
    for(int i=1; i<=n; i++)
    {
        NODE *head = p[now];
        while( head != NULL )
        {
            int len = head->len;
            int v = head->to;
            if( dis[v] > dis[now] + len )
                dis[v] = dis[now] + len;
            head = head->next;

        }
        int min = INT_MAX;
        for(int k=1; k<=n; k++)
            if( !used[k] && dis[k] < min )
                min = dis[now = k];
        used[now] = true;
    }
    return dis[to];
}

/*
**********************************************
最短路：dijkstra + priority(邻接矩阵表示)
记得加头文件!!!!! <queue>
```

```
n 表示 点数，MAX 是 n 的上界+10
************************************************
*/

int map[MAX][MAX];
typedef struct PRI{
    int u,len;
}RPI;
priority_queue<PRI> Q;
bool operator<(PRI a,PRI b)
{
    return a.len > b.len;        // 大于号!!!!
}

int Dijkstra_priority(int from,int to,int n)// 出发点，终止点，总节点
{
    while( !Q.empty() ) Q.pop(); //记得清空
    PRI dis[MAX];
    bool used[MAX];
    memset(used,false,sizeof(used));
    for(int i=1; i<=n; i++)
    {
        dis[i].len = INT_MAX;
        dis[i].u = i;
    }
    dis[from].len = 0;
    used[from] = 1;
    int now = from;
    for(int i=1; i<n; i++)
    {
        for(int k=1; k<=n; k++)
            if( map[now][k] && !used[k] && dis[k].len > dis[now].len +
map[now][k] )
            {
                dis[k].len = dis[now].len + map[now][k];
                Q.push(dis[k]);
            }
        now = Q.top().u;
        Q.pop();
        used[now] = 1;
    }
    return dis[to].len;
}

/*
************************************************
最短路：dijkstra + priority(邻接矩阵表示)
记得加头文件!!!!! <queue>
n 表示 点数，MAX 是 n 的上界+10
************************************************
*/

int map[MAX][MAX],dis[MAX];
struct cmp{
    bool operator()(int a,int b)
    {
        return dis[a] > dis[b];
    }
};
int Dijkstra_priority(int from,int to,int n)// 出发点，终止点，总节点
```

```
{
    priority_queue(int,vector<int>,cmp) q;
    bool used[MAX];
    memset(used,false,sizeof(used));
    for(int i=0; i<n; i++)
        dis[i] = INT_MAX;
    dis[from] = 0;
    while( !q.empty() )
    {
        int now = q.top(); q.pop();
        if( used[now] ) continue;
        used[now] = true;
        for(int k=0; k<n; k++)
            if( map[now][k] && dis[k].len > dis[now].len + map[now][k] )
            {
                dis[k] = dis[now] + map[now][k];
                q.push(k);
            }
    }
    return dis[to].len;
}


/*
************************************************
最短路：dijkstra + priority(邻接表表示)
记得加头文件!!!!! <queue>
n 表示 点数，MAX 是 n 的上界+10
************************************************
*/

typedef struct NODE{
    int to,len;
    NODE *next;
}NODE;
NODE *p[MAX],node[MAX*MAX];
int dis[MAX];
struct cmp{
    bool operator()(int a,int b)
    {
        return dis[a] > dis[b];
    }
};
void Add(int from,int to,int len)
{
    node[cou].next = p[from];
    node[cou].to = to;
    node[cou].len = len;
    p[from] = &node[cou++];
}
typedef pair<int,int> pii;

int Dijkstra_List_priority(int from,int to,int n) // 出发点，终止点，总节
点
{
    priority_queue<pii,vector<pii>,greater<pii> > q;
    int dis[MAX];
    bool used[MAX];
    memset(used,false,sizeof(used));
    for(int i=1; i<=n; i++)
```

```
        dis[i] = INT_MAX;
    dis[from] = 0;
    q.push(make_pair(0, from));
    while( !q.empty() )
    {
        int now = q.top().second; q.pop();
        if( used[now] ) continue;
        used[now] = true;
        NODE *head = p[now];
        while( head != NULL )
        {
            int v = head->to;
            int len = head->len;
            if( dis[v] > dis[now] + len )
            {
                dis[v] = dis[now] + len;
                q.push(make_pair(dis[v], v));
            }
            head = head->next;
        }
    }
    return dis[to];
}

/*
**************************************************
最短路：dijkstra + heap
n 表示 点数，MAX 是 n 的上界+10
**************************************************
*/

typedef struct NODE{
    int to,len;
    NODE *next;
}NODE;
NODE *p[MAX],node[MAX*MAX];

void Add(int from,int to,int len)
{
    node[cou].next = p[from];
    node[cou].to = to;
    node[cou].len = len;
    p[from] = &node[cou++];
}

const int MAX = 1010;
class HEAP{               // DIJ + HEAP
    public:
        typedef struct PRIH{
            int u;
            int len;
        }PRIH;
        int heap_size;
        int pre[MAX*2];
        PRIH a[MAX*2];
        void Change(int x,int y)
        {
            pre[a[x].u] = y;
            pre[a[y].u] = x;
            swap(a[x],a[y]);
```

```
}
void Heap(int i)
{
    int r,l,larg;
    while(1)
    {
        l = (i<<1);
        r = l+1;
        if( l <= heap_size && a[l].len < a[i].len )
            larg = l;
        else
            larg = i;
        if( r <= heap_size && a[r].len < a[larg].len )
            larg = r;
        if( larg != i )
        {
            Change(i,larg);
            i = larg;
        }
        else
            break;
    }
}
void init(int s,int n)       // s 是 出发点，n 是点数
{
    for(int i=0; i<MAX*2; i++)
    {
        pre[i] = i;
        a[i].u = i;
        a[i].len = INT_MAX;
    }
    a[s].len = 0;
    heap_size = n;
    Heap(1);
}
void Update(int i)
{
    int parent;
    while( i > 1 )
    {
        parent = (i>>1);
        if( a[i].len < a[parent].len )
        {
            Change(i,parent);
            i = parent;
        }
        else
            break;
    }
}
void Modify(int x,int key)       // 更新点 x 的长为 key 的路径
{
    x = pre[x];
    if( a[x].len <= key )
        return ;
    a[x].len = key;
    Update(x);
}
void pop()            // 删除最小值
{
```

```
            Change(1,heap_size);
            heap_size--;
            Heap(1);
        }
        int Gettop()      // 取堆中最小值
        {
            return a[1].u;
        }
};

HEAP heap;
int Dijkstra_Heap(int from,int to,int n)
{
    heap.init(from,n);
    int now = from;
    for(int i=1; i<=n; i++)
    {
        now = heap.Gettop();
        int rulen = heap.a[1].len;
        heap.pop();
        NODE *head = p[now];
        while( head != NULL )
        {
            int v = head->to;
            int len = head->len;
            int rv = heap.pre[v];
            if( heap.a[rv].len > rulen + len )
                heap.Modify(v,rulen+len);
            head = head->next;
        }

    }
    int rto = heap.pre[to];
    return heap.a[rto].len;
}
```

3.2 SPFA

```
/*
***********************************************
最短路：SPFA + 邻接矩阵
n 表示 点数，MAX 是 n 的上界+10
***********************************************
*/
int map[MAX][MAX];
queue<int> q;
int SPFA(int from,int to,int n)
{
    while( !q.empty() ) q.pop();
    int dis[MAX];
    int cnt[MAX];
    bool inq[MAX];
    memset(cnt,0,sizeof(cnt));
    memset(inq,false,sizeof(inq));
    for(int i=1; i<=n; i++)
        dis[i] = INT_MAX;
    dis[from] = 0;
    q.push(from);
    inq[from] = true;
    cnt[from]++;
```

```
    int neg = 0;
    while( !q.empty() )
    {
        type now = q.front();
        q.pop();
        inq[now] = false;
        for(int i=1; i<=n; i++)
            if( map[now][i] && dis[i] > dis[now] + map[now][i] )
            {
                dis[i] = dis[now] + map[now][i];
                if( !inq[i] )
                {
                    inq[i] = true;
                    q.push(i);
                    ++cnt[i];
                    if( cnt[i] > n ) // 判断是否存在负环
                    {
                        neg = 1;
                        break;
                    }
                }
            }
        if( neg )
            break;
    }
    if( neg )
    {
        cout << "SPFA say : Exist negative circle !!!!!" << endl;
        return -1;
    }
    return dis[to];
}

/*
**************************************************
最短路：SPFA + 邻接表
n 表示 点数，MAX 是 n 的上界+10
**************************************************
*/
typedef struct NODE{
    int to,len;
    NODE *next;
}NODE;
NODE *p[MAX],node[MAX*MAX];

void Add(int from,int to,int len)
{
    node[cou].next = p[from];
    node[cou].to = to;
    node[cou].len = len;
    p[from] = &node[cou++];
}

int SPFA_List(int from,int to,int n)
{
    while( !q.empty() ) q.pop();
    int dis[MAX],cnt[MAX],neg = 0;
    bool inq[MAX];
    for(int i=1; i<=n; i++)
        dis[i] = INT_MAX;
```

```
    memset(cnt,0,sizeof(cnt));
    memset(inq,false,sizeof(inq));
    dis[from] = 0;
    q.push(from);
    inq[from] = cnt[from] = 1;
    while( !q.empty() )
    {
        int now = q.front();
        q.pop();
        inq[now] = false;
        NODE *head = p[now];
        while( head != NULL )
        {
            int v = head->to;
            int len = head->len;
            if( dis[v] > dis[now] + len )
            {
                dis[v] = dis[now] + len;
                if( !inq[v] )
                {
                    inq[v] = true;
                    q.push(v);
                    cnt[v]++;
                    if( cnt[v] > n )
                    {
                        neg = 1;
                        break;
                    }
                }
            }
            head = head->next;
        }
        if( neg )
            break;
    }
    if( neg )
    {
        cout << "SPFA_List say : Exist negative circle !!!!!" << endl;
        return -1;
    }
    return dis[to];
}
```

3.3 **Floyd**

```
/*
************************************************
最短路：Floyd 邻接矩阵
n 表示 点数，MAX 是 n 的上界+10
这个做法是 map 初始化为 INT_MAX 了
************************************************
*/

int map[max][max];
int Floyd(int from,int to,int n)
{
    for(int i=1; i<=n; i++)
        for(int k=1; k<=n; k++)
            for(int j=1; j<=n; j++)
                if( map[k][i] != INT_MAX && map[i][j] != INT_MAX && map[k][i]
```

```
+ map[i][j] < map[k][j] )
                    map[k][j] = map[k][i] + map[i][j];
    return map[from][to];
}
```

Floyd 记录路径

```
#define INF (1 << 29)

#define N 100
int g[N + 1][N + 1];
int d[N + 1][N + 1];
int n;
int path[N + 1][N + 1];
int path2[N + 1][N + 1];
int begin, end, mid;
int ans;

static void floyd_warshall(void);
static void output(int a, int b);

int
main(void)
{
    int m;
    int i, j, w;

    while (scanf("%d", &n), n != -1) {

        scanf("%d", &m);

        for (i = 1; i <= n; ++i)
        for (j = 1; j <= n; ++j)
            g[i][j] = INF;

        while (m--) {
            scanf("%d%d%d", &i, &j, &w);
            if (g[i][j] > w)
                g[i][j] = g[j][i] = w;
        }

        memset(path, -1, sizeof(path));
        floyd_warshall();

        if (ans == INF)
            printf("No solution.\n");
        else {
            output(begin, end);
            printf("%d %d\n", end, mid);
        }
    }

    return 0;
}

static void
floyd_warshall(void)
{
    int i, j, k;

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

```
    for (j = 1; j <= n; ++j)
        d[i][j] = g[i][j];

    ans = INF;
    for (k = 1; k <= n; ++k) {
        for (i = 1; i <= k - 1; ++i)
        for (j = i + 1; j <= k - 1; ++j) {
            if (ans > d[i][j] + g[i][k] + g[k][j]) { /* 引入 k 这个点构成环 */
                ans = d[i][j] + g[i][k] + g[k][j];
                begin = i, end = j, mid = k; /* i，j 需要 output 输出，k 单独
输出 */
                memcpy(path2, path, sizeof(path)); /* 保存当前最优的路径 */
            }
        }

        for (i = 1; i <= n; ++i)
        for (j = 1; j <= n; ++j) {
            if (d[i][j] > d[i][k] + d[k][j]) {
                d[i][j] = d[i][k] + d[k][j];
                path[i][j] = k;
            }
        }
    }
}

static void
output(int a, int b)
{
    if (path2[a][b] == -1) {
        printf("%d ", a);
        return;
    }

    output(a, path2[a][b]);
    output(path2[a][b], b);
}
```

3.4 **Bellman-Ford**

```
/*
*****************************************************
最短路：Bellman-ford
n 表示 点数，MAX 是 n 的上界+10
*****************************************************
*/
void BAdd(int from,int to,int len)// bellman
{
    e[coue].u = from;
    e[coue].v = to;
    e[coue++].w = len;
}

int Bellman(int from,int to,int n)
{
    int dis[MAX];
    bool used[MAX];
    int neg = 0;
    memset(used,false,sizeof(used));
    for(int i=1; i<=n; i++)
```

```
        dis[i] = INT_MAX;
    dis[from] = 0;
    for(int i=1; i<n; i++)
    {
        for(int k=0; k<coue; k++)
        {
            int v = e[k].v;
            int u = e[k].u;
            int len = e[k].w;
            if( dis[u] != INT_MAX && dis[v] > dis[u] + len )
                dis[v] = dis[u] + len;
        }
    }
    for(int i=1; i<n && !neg; i++)// 判断是否是负环
        for(int k=0; k<coue && !neg; k++)
            if( dis[e[k].u] != INT_MAX && dis[e[k].v] > dis[e[k].u] + e[k].w )
                neg = 1;
    if( neg )
    {
        cout << "BellmanFord say : Exist negative circle !!!!!" << endl;
        return -1;
    }
    return dis[to];
}
```

4 图论—最小生成树

4.1 Prim

```
/*
实例化: Prime< 100 , 10000 > Net;
初始化: Net.init( 30 );//点的数量
加边，Net.add( from, to, w );//directional..
求W(mst); w = Net.prim();
*/
class Edge {
public:
    int to,w;
    Edge *next;
    void add( int t, int ww,Edge *&b){
        to = t;w = ww;next=b;b=this;
    }
};
template<int Vsize, int Esize>
class Prim{
private:
    int countt,N;
    Edge *biao[ Vsize ],A[ 3*Esize ];
    int cnt[ Vsize ];// how many nodes refresh it
    int d[ Vsize ],flag[ Vsize ];
public:
    void init( int n ){
        countt = 0;N = n;
        for( int i = 0; i <= N; i++ )
            biao[i] = NULL;
    }
```

```
    void add( int from,int to,int w ){
        A[ countt++ ].add( to, w, biao[from] );
    }

    int prim( void ){
        int i,j,minn,sum = 0,now = 1;
        memset( flag, 0, sizeof(flag) );
        memset( cnt, 0, sizeof(cnt) );
        fill( d, d+Vsize, INF );
        d[ now ] = 0;
        flag[ now ] = 1;
        for( i = 1; i < N; i++ ){
            for(Edge *p = biao[now]; p!=NULL; p = p->next )
                if( !flag[p->to] ){
                    if( d[p->to] > p->w ){
                        d[p->to] = p->w;
                        cnt[p->to] = 1;
                    }
                    else if( d[p->to] == p->w ){
                        cnt[p->to] ++;
                    }
                }
            minn = INF;
            for( j = 1; j <= N; j++ )
                if( !flag[j] && d[j] < minn )
                    minn = d[ now = j ];
            flag[ now ] = 1;
            //if( cnt[now] > 1 ) return -1;//返回-1 说明 mst 不唯一
            sum += d[now];
        }
        return sum;
    }
};

/*
**************************************************
最小生成树 Prim 算法 ,n 为点的个数，起始点从 1 开始
**************************************************
*/
int map[MAX][MAX],n;
int Prim( int map[][], int n )
{
    int used[MAX],dis[MAX],i,j,sum = 0,now,min;

    memset(used,0,sizeof(used));
    fill(dis,dis+MAX,INT_MAX);

    now = 1; dis[now] = 0; used[now] = 1;
    for(i=1; i<n; i++)
    {
        for(j=1; j<=n; j++)
            if( !used[j] && dis[j] > map[now][j] )
                dis[j] = map[now][j];
        min = INT_MAX;
        for(j=1; j<=n; j++)
            if( !used[j] && dis[j] < min )
                min = dis[now = j];
        used[now] = 1;
        sum += min;
    }
```

```
    return sum;
}
```

4.2 Kruskal

```
/*
************************************************
最小生成树 Kruskal 算法 ,n 为点的个数,e 为边数
记得头文件!!! <algorithm> !!!!!!!
************************************************
*/
int pre[MAX];  // 记录父节点
typedef struct NODE
{
    int x,y,len;   // x，y 为边的两个顶点
}NODE;
NODE edge[LALA];  // 边的条数根据题意
int find(int x)
{
    while( x != pre[x] )
        x = pre[x];
    return x;
}
int cmp ( NODE a ,NODE b)
{
    return a.len < b.len;
}
int Kruskal()
{
    int i,sum = 0,a,b;
    sort(edge,edge+e,cmp);
    for(i=0; i<n; i++)
        pre[i] = i;
    for(i=0; i<e; i++)
    {
        a = find( edge[i].x );
        b = find( edge[i].y );
        if( a != b )
        {
            sum += edge[i].len;
            pre[b] = a;
        }
    }
    return sum;
}
```

5 图论—二分图匹配

5.1 匈牙利算法

```
/*
************************************************
二分图最大匹配 匈牙利算法 邻接矩阵
s 为数组起始点，n 为终止点
************************************************
*/
int map[MAX][MAX];
```

```
int used[MAX],mat[MAX];
void init()
{
    memset(map,0,sizeof(map));
}
int Augment(int s,int n,int x)
{
    int i;
    for(i=s; i<=n; i++)
        if( !used[i] && map[x][i] )
        {
            used[i] = 1;
            if( mat[i] == -1 || Augment(s,n,mat[i]) )
            {
                mat[i] = x;
                return 1;
            }
        }
    return 0;
}

int Hungary(int s,int n)
{
    int i,sum = 0;
    memset(mat,-1,sizeof(mat));
    for(i=s; i<=n; i++)
    {
        memset(used,0,sizeof(used));
        if( Augment(s,n,i) )
            sum++;
    }
    return sum;
}

/*
************************************************
二分图最大匹配 匈牙利算法 数组模拟邻接表
s 为数组起始点，n 为终止点,len[v]存以 v 为起点的个数
建图用 map[x][++len[x]] = y;
************************************************
*/
int map[MAX][MAX];
int used[MAX],mat[MAX],len[MAX];
void init()
{
    memset(len,0,sizeof(len));
    memset(map,0,sizeof(map));
}
int Augment(int x)
{
    int i,k;
    for(i=1; i<=len[x]; i++)
    {
        k = map[x][i];
        if( !used[k] )
        {
            used[k] = 1;
            if( mat[k] == -1 || Augment(mat[k]) )
            {
                mat[k] = x;
```

```
                return 1;
            }
        }
    }
    return 0;
}

int Hungary(int s,int n)
{
    memset(mat,-1,sizeof(mat));
    int i,sum = 0;
    for(i=s; i<=n; i++)
    {
        memset(used,0,sizeof(used));
        if( Augment(s,i) )
            sum++;
    }
    return sum;
}
```

5.2 KM

```
#define INF (1 << 29)

#define N 150
int g[N][N];
int n;

static int lx[N];
static int ly[N];
static char usedx[N];
static char usedy[N];
static int match[N];
static int slack[N];

static void km(void);

int
main(void)
{
    int i, j;
    int sum;

    scanf("%d", &n);

    for (i = 0; i < n; ++i)
    for (j = 0; j < n; ++j)
        scanf("%d", &g[i][j]);

    km();

    sum = 0;
    for (i = 0; i < n; ++i)
        sum += g[i][match[i]];

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

    return 0;
}
```

```
static int
find(int x)
{
    int y;
    int t;

    usedx[x] = 1;

    for (y = 0; y < n; ++y) {
        if (usedy[y])
            continue;
        t = lx[x] + ly[y] - g[x][y];
        if (t == 0) {
            usedy[y] = 1;
            if (match[y] == -1 || find(match[y])) {
                match[y] = x;
                return 1;
            }
        } else {
            if (slack[y] > t)
                slack[y] = t;
        }
    }

    return 0;
}

static void
km(void)
{
    int x, y;
    int i, j;
    int d;

    memset(match, -1, sizeof(match));
    memset(lx, 0, sizeof(lx));
    memset(ly, 0, sizeof(ly));

    for (i = 0; i < n; ++i)
    for (j = 0; j < n; ++j)
        if (g[i][j] > lx[i])
            lx[i] = g[i][j];

    for (x = 0; x < n; ++x) {
        for (i = 0; i < n; ++i)
            slack[i] = INF;
        while (1) {
            memset(usedx, 0, sizeof(usedx));
            memset(usedy, 0, sizeof(usedy));
            if (find(x))
                break;
            d = INF;
            for (y = 0; y < n; ++y) {
                if (!usedy[y] && d > slack[y])
                    d = slack[y];
            }
            for (i = 0; i < n; ++i) {
                if (usedx[i])
                    lx[i] -= d;
            }
```

```
            for (i = 0; i < n; ++i) {
                if (usedy[i])
                    ly[i] += d;
                else
                    slack[i] -= d;
            }
        }
    }
}

KM(copy)
const int MAX = 1024;

int n;               // X 的大小
int weight [MAX] [MAX];        // X 到 Y 的映射（权重）
int lx [MAX], ly [MAX];        // 标号
bool sx [MAX], sy [MAX];    // 是否被搜索过
int match [MAX];        // Y(i) 与 X(match [i]) 匹配

// 初始化权重
void init (int size);
// 从 X(u) 寻找增广道路，找到则返回 true
bool path (int u);
// 参数 maxsum 为 true ，返回最大权匹配，否则最小权匹配
int bestmatch (bool maxsum = true);

void init (int size)
{
    // 根据实际情况，添加代码以初始化
    n = size;
    for (int i = 0; i < n; i ++)
        for (int j = 0; j < n; j ++)
            scanf ("%d", &weight [i] [j]);
}


bool path (int u)
{
    sx [u] = true;
    for (int v = 0; v < n; v ++)
        if (!sy [v] && lx[u] + ly [v] == weight [u] [v])
            {
            sy [v] = true;
            if (match [v] == -1 || path (match [v]))
                {
                match [v] = u;
                return true;
                }
            }
    return false;
}

int bestmatch (bool maxsum)
{
    int i, j;
    if (!maxsum)
        {
        for (i = 0; i < n; i ++)
            for (j = 0; j < n; j ++)
                weight [i] [j] = -weight [i] [j];
```

```
        }

    // 初始化标号
    for (i = 0; i < n; i ++)
        {
        lx [i] = -0x1FFFFFFF;
        ly [i] = 0;
        for (j = 0; j < n; j ++)
            if (lx [i] < weight [i] [j])
                lx [i] = weight [i] [j];
        }

    memset (match, -1, sizeof (match));
    for (int u = 0; u < n; u ++)
        while (1)
            {
            memset (sx, 0, sizeof (sx));
            memset (sy, 0, sizeof (sy));
            if (path (u))
                break;

            // 修改标号
            int dx = 0x7FFFFFFF;
            for (i = 0; i < n; i ++)
                if (sx [i])
                    for (j = 0; j < n; j ++)
                        if(!sy [j])
                            dx = min (lx[i] + ly [j] - weight [i] [j], dx);
            for (i = 0; i < n; i ++)
                {
                if (sx [i])
                    lx [i] -= dx;
                if (sy [i])
                    ly [i] += dx;
                }
            }

    int sum = 0;
    for (i = 0; i < n; i ++)
        sum += weight [match [i]] [i];

    if (!maxsum)
        {
            sum = -sum;
            for (i = 0; i < n; i ++)
                for (j = 0; j < n; j ++)
                    weight [i] [j] = -weight [i] [j];         // 如果需要保
持 weight [ ] [ ] 原来的值，这里需要将其还原
        }
    return sum;
}
```

6 图论—网络流

6.1 最大流

6.1.1 EK

```
/*
************************************************
最大流—EK 算法 （记得初始化 cap） 邻接矩阵形式
************************************************
*/
int cap[MAX][MAX];        //建图的时候用 cap[from][to] += 流量
                          //cap 存的是流量
int EKarp(int s,int t)    // 起点，终点
{
    queue<int> Q;             // a 存每次增广的流量
    int flow[MAX][MAX],a[MAX],u,v,f,pre[MAX];
    f = 0;                // 增广的流量和
    memset(flow,0,sizeof(flow));
    while(1)
    {
        Q.push(s);
        memset(a,0,sizeof(a));
        a[s] = INT_MAX;
        while( !Q.empty() )
        {
            u = Q.front();
            Q.pop();
            for(v=1; v<=m; v++)
                if( !a[v] && cap[u][v] > flow[u][v] )
                {
                    Q.push(v);
                    a[v] = a[u] < cap[u][v] - flow[u][v] ? a[u] : cap[u][v]
- flow[u][v];
                    pre[v] = u;
                }
        }
        if( a[t] == 0 )
            break;
        for(u=t; u!=s; u=pre[u])
        {
            flow[pre[u]][u] += a[t];
            flow[u][pre[u]] -= a[t];
        }
        f += a[t];
    }
    return f;
}
```

6.1.2 Dinic && SAP

```
const long long  INF = 21000000000000560ll;
typedef long long Tp;
using namespace std;
class Arc{
public:
    int to;
```

```
    Tp w;
    Arc *next,*anti;
    Arc *ass( int tt,Tp ww,Arc* &b ){
        to = tt;w = ww;next = b;b = this;return this;
    }
};

template<class Tp,int Vsize, int Esize >
class Network{
private:
    Arc A[ 2*Esize+10 ],*biao[ Vsize ];
    int countt,d[ Vsize ],S,T,N;
    int bfs( void ){
        queue<int> q;
        fill( d,d+Vsize,-1 );
        d[ S ] = 0;
        q.push( S );
        while( !q.empty() ){
            int u = q.front();q.pop();
            for( Arc *p=biao[u]; p != NULL; p=p->next )
                if( d[p->to] == -1 && p->w > 0 )
                    d[p->to] = d[u]+1,q.push(p->to);
        }
        return d[T] != -1;
    }
    Tp dinic( int u,Tp sum ){
        Tp f,o;
        if( u == T ) return sum;
        o = sum;
        for( Arc *p=biao[u]; p != NULL && sum; p=p->next )
            if( d[p->to] == d[u]+1 && p->w > 0 ){
                f = dinic(p->to,min(p->w,sum));
                p->w -= f;p->anti->w += f;
                sum -= f;
            }
        return o-sum;
    }
public:
    Network( void ) {}
    Network( int n ) { init(n) ;}
    void init( int n ){
        countt = 0;N = n;
        for( int i = 0; i <= n; i++ )
            biao[i] = NULL;
    }
    void add( int from,int to,Tp w ){
        Arc *p1 = A[countt++].ass( to,w ,biao[from]);
        Arc *p2 = A[countt++].ass( from,0,biao[ to] );
        p1->anti = p2; p2->anti = p1;
    }
    Tp MaxFlowDinic( int s, int t ){
        S = s;T = t;
        Tp total = 0;
        while( bfs() )
            total += dinic( S,INF );
        return total;
    }
    Tp MaxFlowSap( int s, int t){
        S = s;T = t;
        int i,md;
```

```
        Tp now,total;
        Arc *p,*locate,*ge[ Vsize ],*di[ Vsize ],*path[ Vsize ];
        int dist[ Vsize ], cd[  Vsize ];
        int his[  Vsize ], pre[ Vsize ];
        bool flag;
        memset( dist,0, sizeof(dist) );
        memset( cd,  0, sizeof(cd)   );
        cd[0] = N;
        for( i = 0; i <= N ; i++ )
            di[i] = ge[i] = biao[i];
        for( total = 0, now = INF,i = S; dist[i] < N; ){
            his[i] = now;
            for( flag = false,p=di[i]; p!=NULL; p= p->next){
                if( p->w > 0 && dist[i] ==dist[p->to] + 1 ){
                    now = min( now,p->w );
                    pre[ p->to ] = i;
                    path[ p->to ] = p;
                    di[i] = p;
                    i = p->to;
                    if( i == T ){
                        for( total += now; i != S; i = pre[i] )
                            path[i]->w -= now, path[i]->anti->w += now;
                        now = INF;
                    }
                    flag = true;
                    break;
                }
            }
            if( !flag ){
                for( md = N-1,p=ge[i];p!= NULL;p=p->next )
                    if( p->w > 0 && dist[p->to] < md )
                        md = dist[p->to],locate = p;
                di[i] = locate;
                if( !(--cd[ dist[i] ] ) ) break;
                cd[ dist[i] = md+1 ] ++;
                if( i != S )
                    i = pre[i],now = his[i];
            }
        }
        return total;
    }
    int check( int u ){
        bool flag[Vsize];
        memset( flag, 0, sizeof(flag) );
        dfs( S,flag );
        return count( flag+1,flag+1+N,true);
    }
    void dfs( int u ,bool *flag){
        flag[u] = true;
        for( Arc *p=biao[u];p != NULL; p = p->next ){
            if(  p->w == 0 || flag[p->to] )
                continue;
            dfs( p->to ,flag);
        }
    }
};
//最大流题目构图:
//1.最大权闭合子图:
//  对于 DAG 图中的边 u->v，建立 u 到 v 容量为 INF 的弧
//  正权点 U 连接 S->U，容量为 Weight（U）
```

```
//  负权点 V 连接 V->T，容量为-Weight（V）
//  闭合图的权 = 正权点权值和 - 最大流
//  隐含关系，一旦有 u，必定有 v。。
//2.二分图最小点权覆盖：
//  隐含关系，对于每条边的两个点，至少一个是在覆盖中的
//  对于每个点 x 输入 X，连接 S 到 x，容量为点权
//  对于每个点 y 属于 Y，连接 y 到 T,容来为点权
//  对于每条边 u->v，连接一条容量无限的弧
//3.二分图最大点权独立集：
//  隐含关系，在独立集中没有两个点之间有边
//  最大点权独立集 = 总的点权 - 最小点权覆盖
//4.二分加最大流，二分边的容量等判断最大流是否有答案
```

6.2 最小费用最大流

```
const int maxn = 110;
const int inf = 1008610086;
using namespace std;

class Arc{
public:
    int to,w,cost;
    Arc *next,*anti;
    Arc *add( int tt,int ww,int cc ,Arc *&b){
        to = tt; w = ww; cost = cc;
        next = b;b = this;return this;
    }
};
class Network{
    public:

    Arc a[ 6000 ],*biao[maxn];
    int d[maxn],i,countt;

    Network() {}
    Network(int n){init(n);}

    void init(int n ){
        countt = 0;
        for( i = 0; i < n ;i++ )
            biao[i] = NULL;
    }

    void add( int from, int to, int w, int c ){
        Arc *p1 = a[ countt++ ].add( to, w, c, biao[from] );
        Arc *p2 = a[ countt++ ].add( from, 0, -c, biao[to]);
        p1->anti = p2;
        p2->anti = p1;
    }

    int minimum_cost_maxflow(int s,int t){
        queue<int> q;
        int i,u,cost=0,flow=0;
        while( 1 ){
            bool  inq[maxn];
            int  prep[maxn];
            int  alow[maxn];
            Arc *prea[maxn];
            fill( d, d+maxn, inf );
            memset( inq, 0, sizeof(inq) );
```

```
            alow[s] = inf; d[s] = 0;
            q.push( s );
            while( !q.empty() ){
                u = q.front();q.pop();
                inq[u] = false;
                for( Arc *p=biao[u];p!=NULL;p=p->next )
                    if( p->w && d[p->to] > d[u] + p->cost ){
                        d[p->to] = d[u] + p->cost;
                        prep[p->to] = u;
                        prea[p->to] = p;
                        alow[p->to] = min( alow[u], p->w );
                        if( !inq[p->to] ){inq[p->to]=true;q.push(p->to);}
                    }
            }
            //bellmanford is upper
            if( d[t] == inf ) break;

            for( u = t; u != s; u = prep[u] ){
                prea[u]->w -= alow[t];
                prea[u]->anti->w += alow[t];
            }
            cost += d[t]*alow[t];
            flow += alow[t];
        }
        return cost;
    }

    bool full( int s ){
        for( Arc *p = biao[s]; p != NULL; p = p->next )
            if( p->w ) return false;
        return true;
    }
};
/*
*********************************************************
最小费用最大流 数组模拟邻接表 点的起点为 1，共 n 个点
*****建图的时候，先初始化 init()，再调用
Add(from,to,c,pay); 如果是双向边还需建 Add(to,from,c,pay)
*********************************************************
*/
typedef struct NODE{
    int from,to,cap,cost;
    int next;
}NODE;
NODE node[LALA];          // 根据题意
int p[MAX];               // 相当于指针数组
int cou,n,m;
void init()
{
    memset(p,-1,sizeof(p));
    memset(node,'\0',sizeof(node));
    cou = 2;              // 初始化为偶数
}
void Add(int u,int v,int cap,int cost)
{
    node[cou].from = u;
    node[cou].to = v;
    node[cou].cap = cap;
    node[cou].cost = cost;
    node[cou].next = p[u];
```

```
    p[u] = cou++;

    node[cou].from = v;
    node[cou].to = u;
    node[cou].cap = 0;
    node[cou].cost = -cost;
    node[cou].next = p[v];
    p[v] = cou++;
}
int MincostMaxflow(int s,int t,int n )
{
    queue<int> q;
    int inq[MAX],pre[MAX],dis[MAX],re[MAX];
    int u,v,i,a,c = 0,ind,cost,cap;
    while(1)
    {
        memset(inq,0,sizeof(inq));
        fill(dis,dis+MAX,INT_MAX);
        dis[s] = 0;
        inq[s] = 1;
        pre[s] = s;
        q.push(s);
        while( !q.empty() )
        {
            u = q.front();
            q.pop();
            inq[u] = 0;
            ind = p[u];
            while( ind != -1 )
            {
                u = node[ind].from;
                v = node[ind].to;
                cost = node[ind].cost;
                cap = node[ind].cap;
                if( cap > 0 && dis[v] > dis[u] + cost )
                {
                    dis[v] = dis[u] + cost;
                    pre[v] = u;
                    re[v] = ind;
                    if( !inq[v] )
                    {
                        q.push(v);
                        inq[v] = 1;
                    }
                }
                ind = node[ind].next;
            }
        }
        if( dis[t] >= 0 )
            break;
        a = INT_MAX;
        for(u=t; u!=s; u=pre[u])
            if( node[re[u]].cap < a )
                a = node[re[u]].cap;
        for(u=t; u!=s; u=pre[u])
        {
            node[re[u]^1].cap += a;
            node[re[u]].cap -= a;
        }
        c += dis[t]*a;
```

```
    }
    return -c;
}
```

6.3 有上下界的最大流

```
const int inf = 2100000000;

class Arc{
public:
    int to,w;
    int l,r;
    Arc *next,*anti;
    Arc *add( int tt,int ll,int rr,Arc *&b ){
        to = tt; l = ll; r = rr;
        w = rr-ll;next = b;b = this; return this;
    }
};

template<int Vsize, int Esize>
class BoundNetwork{
public:
    int d[ Vsize ],S,T,bound,countt,in[ Vsize ],out[ Vsize ];
    Arc a[ Esize ],*biao[ Vsize ];

    void init( int n ){
        bound = n;//S = n - 2;T = n - 1;
        for( int i = 0; i < n; i++ )
            biao[i] = NULL;
        memset( in, 0, sizeof(in) );
        memset( out, 0, sizeof(out) );
        countt = 0;
    }

    void add( int from,int to, int l,int r ){
        Arc *p1 = a[ countt++ ].add( to, l, r, biao[from] );
        Arc *p2 = a[ countt++ ].add( from, 0, 0, biao[to] );
        p1->anti = p2;
        p2->anti = p1;
    }

    int bfs( void ){
        queue<int> q;
        fill( d,d+Vsize,-1 );
        d[ S ] = 0;
        q.push( S );
        while( !q.empty() ){
            int u = q.front();q.pop();
            for( Arc *p=biao[u]; p!=NULL; p=p->next )
                if( d[ p->to ] == -1 && p->w > 0 )
                    d[ p->to ] = d[u] + 1,q.push( p->to );
        }
        return d[ T ] != -1;
    }

    int dinic( int u ,int sum){
        int f,o = sum;
        if( u == T ) return sum;
        for( Arc *p = biao[u]; p!=NULL && sum; p=p->next )
            if( d[p->to] == d[u] + 1 && p->w > 0){
```

```
            f = dinic( p->to, min( p->w,sum) );
            p->w -= f;
            p->anti->w += f;
            sum -= f;
        }
    return o - sum;
}

int max_flow1( int s,int t ){
    S = s;T = t;
    int total = 0;
    while( bfs() )
        total += dinic( S, 2100000000 );
    return total;
}

int max_flow(int s,int t){
    S = s;T = t;
    int i,now_flow,total,md;
    Arc *ge[Vsize],*di[Vsize],*path[Vsize];
    int dist[Vsize],cd[Vsize],his[Vsize],pre[Vsize],n=bound;
    Arc *p,*locate;
    bool flag;
    memset( dist,0,sizeof(dist) );
    memset( cd,0,sizeof( cd) );
    cd[0] = n;
    for( i = 0; i < n ; i++ ) di[i] = ge[i] = biao[i];
    for( total = 0, now_flow = inf,i = S; dist[i] < n; ){
        his[i] = now_flow;
        for( flag = false,p=di[i]; p!=NULL; p= p->next){
            if( p->w > 0 && dist[i] ==dist[p->to] + 1 ){
                now_flow = min( now_flow,p->w );
                pre[ p->to ] = i;
                path[ p->to ] = p;
                di[i] = p;
                i = p->to;
                if( i == T ){
                    for( total += now_flow; i != S; i = pre[i] ){
                        path[i]->w -= now_flow;
                        path[i]->anti->w += now_flow;
                    }
                    now_flow = inf;
                }
                flag = true;
                break;
            }
        }
        if( !flag ){
            for( md = n-1,p=ge[i];p!= NULL;p=p->next )
                if( p->w > 0 && dist[p->to] < md )
                    md = dist[p->to],locate = p;
            di[i] = locate;
            if( !(--cd[ dist[i] ] ) ) break;
            cd[ dist[i] = md+1 ] ++;
            if( i != S )
                i = pre[i],now_flow = his[i];

        }
    }
    return total;
```

```
    }

    void construct( int specialS,int specialT ){
        for( int i = 0; i < bound - 2; i++ )
            for( Arc *p=biao[i]; p!=NULL; p = p->next ){
                in[ p->to ] += p->l;
                out[ i ] += p->l;
        }
        for( int i = 0; i < bound - 2; i++ ){
            add( specialS, i, 0, in[i]  );
            add( i, specialT, 0, out[i] );
        }
    }

    bool full( void ){
        for( Arc *p = biao[S]; p != NULL; p = p->next )
            if( p->w > 0 ) return false;
        return true;
    }
};
//示例程序 zoj 3229 Shoot the bullet ~
BoundNetwork< 1500, 1500*80 > Net;
int main(void){
    int n,m,from,to;
    int i,j,c,d,r,l;
    while( scanf("%d%d",&n,&m) != EOF ){
        Net.init(n+m+4);
        for( i = 0; i < m; i++ ){
            scanf("%d",&j);
            Net.add( n+i, n+m+1, j, inf );
        }
        queue<int> q;
        for( i = 0; i < n; i++ ){
            scanf("%d%d",&c,&d);
            Net.add( n+m, i, 0, d );
            while( c-- ){
                scanf("%d%d%d",&j,&r,&l);
                q.push( Net.countt );
                Net.add( i, n+j, r, l );
            }
        }
        j = Net.countt;
        Net.add( n+m+1, n+m, 0, inf );
        Net.construct(n+m+2,n+m+3);
        Net.max_flow(n+m+2,n+m+3);

        if( !Net.full() ){
            puts("-1\n");continue;
        }
        Net.a[j].w = Net.a[j+1].w = 0;
        Net.max_flow1(n+m,n+m+1);
        int total = 0;
        for( Arc *p = Net.biao[Net.S]; p != NULL; p = p->next )
            if( p->to >= 0 && p->to < n ) {
                total += p->r - p->w;
            }
        printf("%d\n",total);
        while( !q.empty() ){
            int u = q.front();q.pop();
            printf("%d\n",Net.a[u].r - Net.a[u].w );
```

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

6.4 混合图的欧拉回路

大致就是，先将无向边定向，就是比如 1<->3，可以定它的方向为 1->3，1 的出度++，3 的入度++即可。

读入的时候如果遇到无向边，把这条边加入待建的网络中，流量为 1。读入完后，然后用出度减入度得到 x，如果 x 为奇数，肯定不存在欧拉回路，如果没有奇数，就用最大流求解。

如果 x 大于 0，则建一条 s（源点）到当前点容量为 x/2 的边，如果 x 小于 0，建一条从当前点到 t（汇点）容量为|x/2|的边。

然后求最大流，如果是满流（即 s 出的流==t 入的流即可，s 指的是建图的时候连接 s 的边的容量和）就满足欧拉回路。

```
int pre[MAX],n;
int ind[MAX],outd[MAX];
int lev[MAX];
typedef struct MAP{
    int cap,to;
    int next;
}MAP;
MAP node[3000];
int head[MAX];
int cou;
void init()
{
    cou = 2;
    memset(node,'\0',sizeof(node));
    memset(head,-1,sizeof(head));
    memset(ind,0,sizeof(ind));
    memset(outd,0,sizeof(outd));
    for(int i=1; i<=n; i++)
        pre[i] = i;
}
int find(int x)
{
    while( x != pre[x] )
        x = pre[x];
    return x;
}
void Union(int x,int y)
{
    int a = find(x);
    int b = find(y);
    if( a == b )
        return ;
    int p = min(a,b);
    pre[a] = pre[b] = pre[x] = pre[y] = p;
}
int check()
```

```
{
    for(int i=1; i<=n; i++)
        if( find(i) != 1 )
            return 0;
    return 1;
}
void Add(int u,int v,int cap)
{
     node[cou].to = v;
     node[cou].cap = cap;
     node[cou].next = head[u];
     head[u] = cou++;

     node[cou].to = u;
     node[cou].cap = 0;
     node[cou].next = head[v];
     head[v] = cou++;
}
queue<int> q;
int BFS(int s,int t)
{
    int p,u,v,cap;
    memset(lev,-1,sizeof(lev));
    q.push(s);
    lev[s] = 0;
    while( !q.empty() )
    {
        u = q.front();
        q.pop();
        p = head[u];
        while( p != -1 )
        {
            v = node[p].to;
            cap = node[p].cap;
            if( cap > 0 && lev[v] == -1 )
            {
                lev[v] = lev[u] + 1;
                q.push(v);
            }
            p = node[p].next;
        }
    }
    return lev[t] != -1;
}
int Dinic(int k,int sum,int s,int t)
{
    int i,a,os;
    if( k == t )
        return sum;
    os = sum;
    int p = head[k];
    while( p != -1 && sum )
    {
        int to = node[p].to;
        int cap = node[p].cap;
        if( lev[to] == lev[k] + 1 && cap > 0 )
        {
            a = Dinic(to,min(sum,cap),s,t);
            node[p^1].cap += a;
            node[p].cap -= a;
```

```
            sum -= a;
        }
        p = node[p].next;
    }
    return os - sum;
}
int main()
{
    int m,from,to,s;
    int ncases;
    scanf("%d",&ncases);
    while( ncases-- )
    {
        scanf("%d%d",&n,&m);
        init();
        while( m-- )
        {
            scanf("%d%d%d",&from,&to,&s);
            if( from == to )
                continue;
            ind[to]++;
            outd[from]++;
            Union(from,to);
            if( s != 1 )
                Add(from,to,1);
        }
        if( !check() )
        {
            printf("impossible\n");
            continue;
        }
        int flag = 1;
        int sum = 0;
        for(int i=1; i<=n; i++)
        {
            outd[i] -= ind[i];
            if( outd[i] % 2 == 1 )
            {
                flag = 0;
                break;
            }
            outd[i] /= 2;
            if( outd[i] > 0 )
            {
                Add(0,i,outd[i]);
                sum += outd[i];
            }
            else
                Add(i,n+1,-outd[i]);
        }
        if( !flag )
        {
            printf("impossible\n");
            continue;
        }
        int ans = 0;
        while( BFS(0,n+1) )
            ans += Dinic(0,INT_MAX,0,n+1);
        if( ans == sum )
            printf("possible\n");
```

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

7 图论—连通性

7.1 强连通分量

7.1.1 Kosaraju

```
class Edge{
public:
    int to;
    Edge *next;
    void add( int tt, Edge *&b ){to=tt;next=b;b=this;}
};

template< int Esize, int Vsize >
class Kosarajo{
public:
    Edge *biao[ Vsize ],a[ 2*Esize+10 ],*fan[ Vsize ];
    int flag[ Vsize ],countt,N,M;
    int TIME,GROUP,label[ Vsize ];
    stack<int> S;
    void init(void){
        for(int i = 0; i < N ;i++  ){
            biao[i] = fan[i] = NULL;
            flag[i] = 0;
        }
        GROUP = countt = 0;
    }

    void dfs1( int u ){
        flag[u] = 1;
        for( Edge *p=biao[u];p!=NULL;p=p->next )
            if( !flag[p->to] ) dfs1(p->to);
        S.push( u );
    }

    void dfs2( int u,int group ){
        flag[u] = 1;
        for( Edge *p = fan[u]; p!=NULL; p=p->next )
            if( !flag[p->to] ) dfs2(p->to,group);
        label[u] = group;
    }

    int getscc(){
        for( int i = 0; i < N; i++ )
            if( !flag[i] ) dfs1(i);
        while( !S.empty() ){
            int u = S.top();S.pop();
            if( !flag[u] ) dfs2(u,GROUP++);
        }
        return GROUP;
```

```
    }
};
```

7.1.2 Tarjan

```
class Edge{
public:
    int to;
    Edge *next;
    void add( int tt, Edge *&b ){to=tt;next=b;b=this;}
};
template<int Vsize, int Esize>
class TarjanScc{
public:
    Edge *biao[ Vsize ],a[ Esize*2+10];
    int flag[ Vsize ],countt,N,M,d[ Vsize ];
    int TIME,GROUP,label[ Vsize ],in[ Vsize ],low[ Vsize ];
    stack<int> S;
    void init(void){
        for(int i = 0; i < N ;i++  )
            biao[i] = NULL,in[i]=flag[i]=0;
        TIME = GROUP = countt = 0;
    }
    void add( int from, int to ){
        a[ countt++ ].add( to, biao[from] );
    }
    void scc( int u ){
        low[u] = d[u] = TIME++;
        flag[u] = 1;
        S.push( u );
        in[u] = 1;
        for( Edge *p = biao[u]; p != NULL; p = p->next ){
            int v = p->to;
            if( !flag[v] ) {
                scc(v);
                low[u] = min( low[u], low[v] );
            }
            else if( d[v] < d[u] && in[v] == 1 )
                low[u] = min( low[u], d[v] );
        }

        if( low[u] == d[u] ){
            while( !S.empty() && d[S.top()] >= d[u] ){
                label[ S.top() ] = GROUP;
                in[ S.top() ] = 0;
                S.pop();
            }
            GROUP++;
        }
    }
};
```

7.2 双连通分量

7.2.1 Tarjan

```
const int MAXV=101;
struct edge{
    int to;
    struct edge *next;
```

```
    void add(int t,struct edge *&b){to=t;next=b;b=this;}
}*biao[ MAXV ],a[ 5000 ];//分别是临界表和节点仓库
//如果有一条边 from-to，那么添加边的语句为 a[ count++ ].add( to,biao[from] );

//************************************/
//用 tarjan 算法求桥，割点，双连通分量
//************************************/
int d[ MAXV ];//发现时间
int low[ MAXV ];//最早可达祖先的发现时间
int flag[ MAXV ];//0 白，1 灰，2 黑
int out[ MAXV ];// 当前点的不相交子树，i 点去掉所得到的连通图个数为
out[i]+1(ROOT 除外)
bool is[ MAXV ];//是否是割点
int N;//点的个数
int NN;//割点个数
int TIME;//时间戳
int ROOT;//每次 dfs 都是从 ROOT 开始的
void bcc( int u ,int father){
    d[u] = ++TIME;
    low[u] = d[u];
    flag[u] = 1;

    for(edge *p = biao[u]; p != NULL; p=p->next ){
        int v = p->to;
        if( flag[ v ] == 1 && father != v )
            low[u] = min( low[u], d[v]);
        else if( !flag[ v ] ){
            bcc(v,u);
            low[u] = min( low[u], low[v] );
            if( d[u] <= low[v] && u != ROOT )
                is[u] = true;
            if( d[u] <= low[v] ) out[u]++;
            if( d[u] < low[v] ) ;
            //条件成立，则此边是桥,否则，uv 在同一个双连通分量中
        }
    }

    if( u == ROOT && out[u] >= 2 ) is[u] =true;
    flag[u] = 2;
    if( is[u] ) NN++;
}
void init(void){
    fill(flag,flag+MAXV,0);
    fill(out,out+MAXV,0);
    fill( is,is+MAXV,0);
    for(int i = 0;i <= MAXV;i++)biao[i]=NULL;
    TIME = 0;
    ROOT = 1;
}
```

8 数据结构

8.1 （待补充）并查集

8.2 （待补充）AC 自动机

8.3 二叉查找树

```
typedef struct BST{
    int key;
    BST *lchild,*rchild,*parent; // 存左右孩子以及父节点
}BST;
BST *head,*p,node[100];
int cou;
void init()
{
    head = p = NULL;
    cou = 0 ;
    memset(node,'\0',sizeof(node));
}
void BuildBST(BST *&head,int x) // 创建 BST，即 BST 的插入~
{
    if( head == NULL )
    {
        node[cou].key = x;
        node[cou].parent = p;
        head = &node[cou++];
        return ;
    }
    p = head;
    if( head->key > x )
        BuildBST(head->lchild,x);
    else
        BuildBST(head->rchild,x);
}

void InorderTraver(BST *head)  // 中序遍历
{
    if( head == NULL )
        return ;
    InorderTraver( head->lchild );
    cout << head->key << ' ' ;
    InorderTraver( head->rchild );

}

BST* Search( BST *&head,int x )//在 BST 中查找 key 为 x 的值
{
    if( head->key == x )
    {
        p = head;
        return head;
    }
    if( head->key > x )
        Search(head->lchild,x);
```

```
    else
        Search(head->rchild,x);
}

BST* Minmum(BST* head) // 返回 BST 中最小值
{
    while( head->lchild != NULL )
        head = head->lchild;
    return head;
}

BST *Maxmum( BST* head ) // 返回 BST 中最大值
{
    while( head->rchild != NULL )
        head = head->rchild;
    return head;
}

BST* Successor(BST *head) // 返回节点 head 的后继节点
{
    if( head->rchild != NULL )
        return Minmum(head->rchild);
    BST* y = head->parent;
    while( y != NULL && head == y->rchild )
    {
        head = y;
        y = y->parent;
    }
    return y;
}

BST* Predecessor(BST *head) // 返回节点 head 的前驱节点
{
    if( head->lchild != NULL )
        return Maxmum(head->lchild);
    BST* y = head->parent;
    while( y != NULL && head == y->lchild )
    {
        head = y;
        y = y->parent;
    }
    return y;
}

void Delet(BST *z) //删除节点 z
{
    BST *x,*y;
    if( z->lchild == NULL || z->rchild == NULL )
        y = z;
    else
        y = Successor(z);
    if( y->lchild != NULL )
        x = y->lchild;
    else
        x = y->rchild;
    if( x != NULL )
        x->parent = y->parent;
    if( y->parent == NULL )
        head = x;
    else
```

```
        if( y == y->parent->lchild )
            y->parent->lchild = x;
        else
            y->parent->rchild = x;
    if( y != z )
        z->key = y->key;
}

int main()
{
    int x;
    while( cin >> x && x)
        BuildBST(head,x);
    InorderTraver(head); //中序遍历输出
    cout << endl;
    while( cin >> x )
    {
        BST *z;
        Search(head,x);
        z = p;
        Delet(z);
        if( head == NULL )
            break;
        InorderTraver(head); //删除一个 中序遍历一次
    }
return 0;
}
```

8.4 树状数组

```
/*
************************************************
树状数组
c ----> 树状数组
一维树状数组
************************************************
*/
int c[MAX][MAX];

int Lowbit(int x)
{
    return x & (-x);
}

void Updata(int x,int num)// num 可能都为 1，具体问题具体分析
{
    int i;
    for(i=x; i<MAX; i+=Lowbit(i))
        c[i] += num;   // 若 num 为 1，c[i][k]++
}

int Getsum(int x)//sum 的返回值的类型可能是 long long ，根据题意判断
{
    int sum = 0,i;
    for(i=x; i>0; i-=Lowbit(i))
        sum += c[i];
    return sum;
}

/*
```

```
************************************************
树状数组
c ----> 树状数组
二维树状数组
************************************************
*/
int c[MAX][MAX];

int Lowbit(int x)
{
    return x & (-x);
}

void Updata( int x,int y, int num)  // num 可能都为1，具体问题具体分析
{
    int i,k;
    for(i=x; i<MAX; i+=Lowbit(i))
        for(k=y; k<MAX; k+=Lowbit(k))
            c[i][k] += num;   // 若num 为1，c[i][k]++
}

int Getsum(int x,int y)//sum 的返回值的类型可能是 long long ，根据题意判断
{
    int i,k,sum = 0;
    for(i=x; i>0; i-=Lowbit(i))
        for(k=y; k>0; k-=Lowbit(k))
            sum += c[i][k];
    return sum;
}
```

8.5 线段树

8.5.1 注意事项

1、本人所写线段树最底层叶子均为[l,l+1)，所以建树的时候，一定要考虑端点问题。
一般比如题目要求为点树（即最底层为点的话），可以映射为 0 -> [0,1)。
线段树根节点统一从 1 开始编号。即,比如建树、更新，查询[x,y](x, y, >= 1)：

```
Build(1, x-1, y);
Updata(1, x-1, y);
Query(1, x-1, y);
```

2、注意题目要求，是用 long long（__int64）还是用 int。
3、假如区间总长度为 MAX,那么开的结点数为 MAX<<2。

8.5.2 线段树结点定义

```
/*************************************************************************
l       当前区间左端点的位置
r       当前区间右端点的位置
len()   求得当前区间的长度，即 r - l
mid()   (l + r)/2
in()    判断一个区间是否覆盖当前区间
lr()    给当前区间左右端点位置赋值
*************************************************************************/
struct Tnode{              // 一维线段树
    int l,r;
```

```
    int len() { return r - l;}
    int mid() { return MID(l,r);}
    bool in(int ll,int rr) { return l >= ll && r <= rr; }
    void lr(int ll,int rr){ l = ll; r = rr;}
};

/*************************************************************************
*********
内容同上
Tnode son  为子树的结点
*************************************************************************
********/
struct T2node             // 二维线段树（树套树）
{
    int l,r;
    Tnode son[MAXM<<2];
    int len() { return r - l;}
    int mid() { return MID(l,r);}
    bool in(int ll,int rr) { return l >= ll && r <= rr; }
    void lr(int ll,int rr){ l = ll; r = rr;}
};
```

8.5.3 扫描线定义

```
struct Sline              // 扫描线定义
{
    double x,y1,y2;
    int flag;
};
```

8.5.4 线段树一般模板

```
/*************************************************************************
                  一维线段树，例子是求区间乘积，Updata 修改的是点的值
                  根一律为 1 ，MAX 为区间总长度
*************************************************************************/
struct Tnode{             // 一维线段树
    int l,r,val;
    long long sum;
    int len() { return r - l;}
    int mid() { return MID(l,r);}
    bool in(int ll,int rr) { return l >= ll && r <= rr; }
    void lr(int ll,int rr){ l = ll; r = rr;}
};
Tnode node[MAX<<2];
int a[MAX];
void Updata_sum(int t)
{
    node[t].sum = ( node[L(t)].sum % MOD * node[R(t)].sum % MOD ) % MOD;
}
void Build(int t,int l,int r)
{
    node[t].lr(l,r);
    if( node[t].len() == 1 )
    {
        node[t].sum = node[t].val = a[l];
        return ;
    }
    int mid = MID(l,r);
```

```
    Build(L(t),l,mid);
    Build(R(t),mid,r);
    Updata_sum(t);
}

void Updata(int t,int l,int r,int xx)
{
    if( node[t].in(l,r) )
    {
        node[t].sum = node[t].val = val;
        return ;
    }
    if( node[t].len() == 1 ) return ;
    int mid = node[t].mid();
    if( l < mid ) Updata(L(t),l,r,val);
    if( r > mid ) Updata(R(t),l,r,val);
    Updata_sum(t);
}

int Query(int t,int l,int r)
{
    if( node[t].in(l,r) ) return node[t].sum;
    if( node[t].len() == 1 ) return 0;
    int mid = node[t].mid();
    long long ans = 1ll;
    if( l < mid ) ans *= Query(L(t),l,r);
    ans %= MOD;
    if( r > mid ) ans *= Query(R(t),l,r);
    ans %= MOD;
    return ans;
}

/*****************************************************************
                    二维线段树，例子是求区间最大值
*****************************************************************/
const int MAX = 100;          //第一维
const int MAXM = 1010;           //第二维
struct Tnode{
    int l,r,val;
    int len() { return r - l;}
    int mid() { return MID(l,r);}
    bool in(int ll,int rr) { return l >= ll && r <= rr; }
    void lr(int ll,int rr){ l = ll; r = rr;}
};
struct T2node
{
    int l,r;
    Tnode son[MAXM<<2];
    int len() { return r - l;}
    int mid() { return MID(l,r);}
    bool in(int ll,int rr) { return l >= ll && r <= rr; }
    void lr(int ll,int rr){ l = ll; r = rr;}

};
T2node node[MAX<<2];
void init()
{
    memset(node,0,sizeof(node));
}
void sub_build(int f,int t,int l,int r)
```

```
{
    node[f].son[t].lr(l,r); node[f].son[t].val = -inf;
    if( l == r - 1 ) return ;
    int mid = MID(l,r);
    sub_build(f,L(t),l,mid);
    sub_build(f,R(t),mid,r);
}

void Build(int t,int l1,int r1,int l2,int r2)
{
    node[t].lr(l1,r1);
    sub_build(t,1,l2,r2);            // 建立子树
    if( l1 == r1 - 1 ) return ;
    int mid = MID(l1,r1);
    Build(L(t),l1,mid,l2,r2);
    Build(R(t),mid,r1,l2,r2);
}

void sub_updata(int f,int t,int l,int r,int val)
{
    node[f].son[t].val = max(node[f].son[t].val,val);
    if( node[f].son[t].in(l,r) ) return ;
    if( node[f].son[t].len() == 1 ) return ;
    int mid = node[f].son[t].mid();
    if( l < mid ) sub_updata(f,L(t),l,r,val);
    if( r > mid ) sub_updata(f,R(t),l,r,val);
}

void Updata(int t,int l1,int r1,int l2,int r2,int val)
{
    sub_updata(t,1,l2,r2,val);              // 更新子树
    if( node[t].in(l1,r1) ) return ;
    if( node[t].len() == 1 ) return ;
    int mid = node[t].mid();
    if( l1 < mid ) Updata(L(t),l1,r1,l2,r2,val);
    if( r1 > mid ) Updata(R(t),l1,r1,l2,r2,val);
}

int sub_query(Tnode *node,int t,int l,int r)
{
    if( node[t].in(l,r) ) return node[t].val;
    if( node[t].len() == 1 ) return -inf;
    int mid = node[t].mid();
    int ans = -inf;
    if( l < mid )
        ans = max(ans,sub_query(node,L(t),l,r));
    if( r > mid )
        ans = max(ans,sub_query(node,R(t),l,r));
    return ans;
}

int Query(int t,int l1,int r1,int l2,int r2)
{
    if( node[t].in(l1,r1) )
        return sub_query(node[t].son,1,l2,r2);
    if( node[t].len() == 1 ) return -inf;
    int mid = node[t].mid();
    int ans = -inf;
    if( l1 < mid )
        ans = max(ans,Query(L(t),l1,r1,l2,r2));
```

```
    if( r1 > mid )
        ans = max(ans,Query(R(t),l1,r1,l2,r2));
    return ans;
}
```

8.5.5 矩形面积交

```
/*************************************************************************
       线段树求矩形交的面积（即每块区域被覆盖多于两次的面积）（hdu1255）
*************************************************************************/
const int MAX = 2010;
struct Tnode{int l,r,cover;double once,len;};
Tnode node[MAX<<2];
struct Sline{double x,y1,y2;int flag;};
Sline l[MAX];
double y[MAX];
void add_line(double x1,double y1,double x2,double y2,int &cnt)
{
    l[cnt].x = x1; l[cnt].y1 = y1; l[cnt].y2 = y2;
    l[cnt].flag = 1;
    y[cnt++] = y1;
    l[cnt].x = x2; l[cnt].y1 = y1; l[cnt].y2 = y2;
    l[cnt].flag = -1;
    y[cnt++] = y2;
}
bool cmp(Sline a,Sline b)
{
    if( a.x == b.x )
        return a.flag > b.flag;
    return a.x < b.x;
}
void Build(int t,int l,int r)
{
    node[t].l = l; node[t].r = r;
    node[t].cover = 0; node[t].once = node[t].len = 0.0;
    if( node[t].l == node[t].r - 1 ) return ;
    int mid = MID(l,r);
    Build(L(t),l,mid);
    Build(R(t),mid,r);
}
void Updata_len(int t)
{
    if( node[t].cover >= 2 )
    {
        node[t].once = 0;
        node[t].len = y[node[t].r] - y[node[t].l];
        return ;
    }
    if( node[t].cover == 1 )
    {
        if( node[t].l == node[t].r - 1 )
            node[t].len = 0;
        else
            node[t].len = node[R(t)].len + node[L(t)].len + node[R(t)].once
+ node[L(t)].once;
        node[t].once = y[node[t].r] - y[node[t].l] - node[t].len;
        return ;
    }
    if( node[t].cover == 0 )
    {
```

```
        if( node[t].l == node[t].r - 1 )
            node[t].len = node[t].once = 0;
        else
        {
            node[t].len = node[R(t)].len + node[L(t)].len;
            node[t].once = node[R(t)].once + node[L(t)].once;
        }
        return ;
    }
}
void Updata(int t, Sline p)
{
    if( y[node[t].l] >= p.y1 && y[node[t].r] <= p.y2 )
    {
        node[t].cover += p.flag;
        Updata_len(t);
        return ;
    }
    if( node[t].l == node[t].r - 1 ) return ;
    int mid = MID(node[t].l,node[t].r);
    if( p.y1 < y[mid] ) Updata(L(t),p);
    if( p.y2 > y[mid] ) Updata(R(t),p);
    Updata_len(t);
}
void solve(int n,int cnt)
{
    Build(1,0,cnt-1);
    double ans = 0.0;
    Updata(1,l[0]);
    for(int i=1; i<n; i++)
    {
        ans += node[1].len*(l[i].x - l[i-1].x);
        Updata(1,l[i]);
    }
    printf("%.2lf\n",ans);
}
int main()
{
    cnt = 0;
    add_line(x1,y1,x2,y2,cnt);
    sort(y,y+cnt);
    sort(l,l+cnt,cmp);
    int t = unique(y,y+cnt) - y;
    solve(cnt,t);

return 0;
}
```

8.5.6 矩形面积并

```
void Build(int t,int l,int r)
{
    node[t].l = l; node[t].r = r;
    node[t].cover = 0;
    if( l == r - 1 ) return ;
    int mid = MID(l,r);
    Build(R(t),mid,r);
    Build(L(t),l,mid);
}
```

```
void Updata_len(int t)
{
    if( node[t].cover > 0 )
        node[t].len = y[node[t].r] - y[node[t].l];
    else
        if( node[t].l == node[t].r - 1 )
            node[t].len = 0;
        else
            node[t].len = node[R(t)].len + node[L(t)].len;
}

void Updata(int t,Sline p)
{
    if( y[node[t].l] >= p.y1 && y[node[t].r] <= p.y2 )
    {
        node[t].cover += p.flag;
        Updata_len(t);
        return ;
    }
    if( node[t].l == node[t].r - 1 ) return ;
    int mid = MID(node[t].l,node[t].r);
    if( p.y1 < y[mid] ) Updata(L(t),p);
    if( p.y2 > y[mid] ) Updata(R(t),p);
    Updata_len(t);
}
long long solve(int n,int cnt)
{
    Build(1,0,cnt-1);
    Updata(1,l[0]);
    long long ans = 0ll;
    for(int i=1; i<n; i++)
    {
        ans += (l[i].x - l[i-1].x)*node[1].len;
        Updata(1,l[i]);
    }
    return ans;
}
```

8.5.7 线段树求矩形覆盖 K 次交面积（可求面积并）

```
/*******************************************************************
        线段树求矩形交面积（覆盖大于等于两次）
                矩形并面积（覆盖大于等于一次）
                矩形覆盖 K 次交面积（覆盖大于等于 N 次)
        注意：根据题意设定 slen 类型
        slen[i] 当前区间覆盖大于等于 i 次的长度
        cover   当前区间覆盖次数，cover == 0 不代表子区间没有被覆盖

*******************************************************************/
const int MAX = 60010;        // 矩形个数*2
const int K = 2;
typedef long long LL;

struct Sline{ int x,y1,y2,flag;};
struct Tnode{              // 一维线段树
    int l, r, cover, slen[K+1];      //大小至少为 K+1，需要用到 slen[K]
    int len() { return r - l;}
    int mid() { return MID(l,r);}
    bool in(int ll,int rr) { return l >= ll && r <= rr; }
    void lr(int ll,int rr){ l = ll; r = rr;}
```

```
};

Tnode  node[MAX<<2];
Sline  l[MAX];
int    y[MAX], cnty, cnt;

void add_line(int x1,int y1,int x2,int y2,int &cnt)
{
    l[cnt].x = x1; l[cnt].y1 = y1; l[cnt].y2 = y2;
    l[cnt].flag = 1;
    y[cnt++] = y1;
    l[cnt].x = x2; l[cnt].y1 = y1; l[cnt].y2 = y2;
    l[cnt].flag = -1;
    y[cnt++] = y2;
}

void Build(int t,int l,int r)
{
    node[t].lr(l,r);
    node[t].cover = 0;
    FOR(i, 0, K+1)
        node[t].slen[i] = 0;
    if( node[t].len() == 1 )
        return ;
    int mid = MID(l,r);
    Build(L(t),l,mid);
    Build(R(t),mid,r);
}

void Updata_len(int t)
{
    int cc = node[t].cover;
    FOR(i, 0, K+1)
    {
        if( cc >= i )
            node[t].slen[i] = y[node[t].r] - y[node[t].l];
        else
            if( node[t].len() == 1 )
                node[t].slen[i] = 0;
            else
                node[t].slen[i]      =      node[L(t)].slen[i-cc]      +
node[R(t)].slen[i-cc];
    }
}
void Updata(int t, Sline p)
{
    if( p.y1 <= y[node[t].l] && p.y2 >= y[node[t].r] )
    {
        node[t].cover += p.flag;
        Updata_len(t);
        return ;
    }
    if( node[t].len() == 1 ) return ;

    int mid = node[t].mid();
    if( p.y1 < y[mid] )      Updata(L(t), p);
    if( p.y2 > y[mid] )   Updata(R(t), p);

    Updata_len(t);
}
```

```
bool cmp(Sline a, Sline b)
{
    if( a.x == b.x )
        return a.flag > b.flag;
    return a.x < b.x;
}

LL solve(int n)
{
    LL ans = 0;
    sort(y, y+n);
    cnty = unique(y, y+n) - y;
    sort(l, l+n, cmp);

    Build(1, 0, cnty-1);

    Updata(1, l[0]);
    FOR(i, 1, n)
    {
        ans += node[1].slen[K] * 1ll * (l[i].x - l[i-1].x);
        Updata(1, l[i]);
    }
    return ans;
}
int main()
{
    int ind = 1, ncases, n, x1, y1, x2, y2;

    scanf("%d", &ncases);

    while( ncases-- )
    {
        cnt = 0;
        scanf("%d", &n);

        FOR(i, 0, n)
        {
            scanf("%d%d%d%d", &x1, &y1, &x2, &y2);
            x2++; y2++;
            add_line(x1, y1, x2, y2, cnt);
        }

        if( K > n )
        {
            printf("Case %d: 0\n", ind++);
            continue;
        }

        LL ans = solve(cnt);
        printf("Case %d: %lld\n", ind++, ans);
    }

return 0;
}
```

8.5.8 周长轮廓并

```
/*******************************************************************
    线段树矩形周长并 poj 1177 Picture
*******************************************************************/
```

```
const int MAX = 5010;
struct Tnode{ int l,r,num,len,cover;bool lb,rb;};
struct Sline{ int x,y1,y2,flag;};
Tnode node[MAX<<2];
Sline l[MAX<<1];
int y[MAX<<1];

void add_line(int x1,int y1,int x2,int y2,int &cnt)
{
    l[cnt].x = x1; l[cnt].y1 = y1; l[cnt].y2 = y2; l[cnt].flag = 1;
    y[cnt++] = y1;
    l[cnt].x = x2; l[cnt].y1 = y1; l[cnt].y2 = y2; l[cnt].flag = -1;
    y[cnt++] = y2;
}
void init()
{
    memset(node,0,sizeof(node));
}
void Build(int t,int l,int r)
{
    node[t].l = l; node[t].r = r;
    node[t].num = 0;
    if( l == r - 1 ) return ;
    int mid = MID(l,r);
    Build(R(t),mid,r);
    Build(L(t),l,mid);
}
void Updata_len(int t)
{
    if( node[t].cover > 0 )
    {
        node[t].num = node[t].lb = node[t].rb = 1;
        node[t].len = y[node[t].r] - y[node[t].l];
    }
    else
        if( node[t].l == node[t].r - 1 )
            node[t].rb = node[t].lb = node[t].len = node[t].num = 0;
        else
        {
            node[t].rb = node[R(t)].rb; node[t].lb = node[L(t)].lb;
            node[t].len = node[L(t)].len + node[R(t)].len;
            node[t].num    =    node[L(t)].num    +    node[R(t)].num    -
node[R(t)].lb*node[L(t)].rb;
        }       //两线段重合的话，得减一下。。
}
void Updata(int t,Sline p)
{
    if( y[node[t].l] >= p.y1 && y[node[t].r] <= p.y2 )
    {
        node[t].cover += p.flag;
        Updata_len(t);
        return ;
    }
    if( node[t].l == node[t].r - 1 ) return ;
    int mid = MID(node[t].l ,node[t].r);
    if( p.y1 < y[mid] ) Updata(L(t),p);
    if( p.y2 > y[mid] ) Updata(R(t),p);
    Updata_len(t);
}
int solve(int n,int cnt,Sline *l)
```

```
{
    init();
    Build(1,0,cnt-1);
    int ans = 0,last = 0,lines = 0;
    for(int i=0; i<n; i++)
    {
        Updata(1,l[i]);
        if( i >= 1 )
            ans += 2 * lines * (l[i].x - l[i-1].x);//计算平行于 x 轴的长度
        ans += abs(node[1].len - last);        // 计算平行于 y 轴的长度
        last = node[1].len;
        lines = node[1].num;
    }
    return ans;
}

bool cmp(Sline a,Sline b)
{
    if( a.x == b.x ) return a.flag > b.flag;
    return a.x < b.x;
}
int main()
{
    int n,x1,x2,y1,y2;

    while( ~scanf("%d",&n) )
    {
        if( n == 0 )
        {
            printf("0\n");
            continue;
        }
        int cnt = 0;
        for(int i=0; i<n; i++)
        {
            scanf("%d%d%d%d",&x1,&y1,&x2,&y2);
            add_line(x1,y1,x2,y2,cnt);
        }
        sort(y,y+cnt);
        sort(l,l+cnt,cmp);
        int t = cnt;
        t = unique(y,y+cnt) - y;
        int ans = solve(cnt,t,l);
        printf("%d\n",ans);
    }

return 0;
}
```

8.5.9 求区间连续递增最长子序列

```
/*******************************************************************
线段树求区间连续递增最长子序列（区间查询，区间统一加上某值）
UESTC 1425
lval        从左端起连续的递增序列（最左端为此序列的最大值）
rval        从右端起连续的递增序列（最右端为此序列的最小值）
max         当前区间连续递增序列的最大值
add         区间整体加的值
ll          当前区间左端点的值
```

```
rr          当前区间右端点的值
*************************************************************************/
int a[MAX];
struct Tnode{              // 一维线段树
    int l,r,lval, rval, max, add, ll, rr;
    int len() { return r - l;}
    int mid() { return MID(l,r);}
    bool in(int ll,int rr) { return l >= ll && r <= rr; }
    void lr(int ll,int rr){ l = ll; r = rr;}
};
Tnode node[MAX<<2];
void Updata_val(int t)  //左右区间连接的时候更新
{
    node[t].ll = node[L(t)].ll;
    node[t].rr = node[R(t)].rr;
    node[t].lval = node[L(t)].lval;
    if( node[L(t)].lval == node[L(t)].len()
           && node[R(t)].ll > node[L(t)].rr )
        node[t].lval += node[R(t)].lval;

    node[t].rval = node[R(t)].rval;
    if( node[R(t)].rval == node[R(t)].len()
           && node[R(t)].ll > node[L(t)].rr )
        node[t].rval += node[L(t)].rval;

    int mval = 0;
    if( node[R(t)].ll > node[L(t)].rr )
        mval = node[L(t)].rval + node[R(t)].lval;

    node[t].max = max(mval, max(node[L(t)].max, node[R(t)].max));
    node[t].max = max(node[t].max, max(node[t].lval, node[t].rval));
}
void Build(int t,int l,int r)
{
    node[t].add = node[t].lval = node[t].rval = node[t].max = 0;
    node[t].lr(l,r);
    if( node[t].len() == 1 )
    {
        node[t].lval = node[t].rval = node[t].max = 1;
        node[t].ll = node[t].rr = a[node[t].l];
        return ;
    }
    int mid = MID(l,r);
    Build(L(t),l,mid);
    Build(R(t),mid,r);
    Updata_val(t);
}

void Updata_llrr(int t, int add) // 更新区间左右端点的值
{
    node[t].ll += add;
    node[t].rr += add;
}

void Push_down(int t) // 向下推进 add 值
{
    if( node[t].len() == 1 )
    {
        node[t].add = 0;
        return ;
```

```
    }
    if( node[t].add )
    {
        node[L(t)].add += node[t].add;
        Updata_llrr( L(t), node[t].add );

        node[R(t)].add += node[t].add;
        Updata_llrr( R(t), node[t].add);

        node[t].add = 0;
    }
}

void Updata(int t,int l,int r,int val)//更新的是 add 值
{
    if( node[t].in(l,r) )
    {
        node[t].add += val;
        Updata_llrr(t, val);
        Push_down(t);
        return ;
    }
    if( node[t].len() == 1 ) return ;
    Push_down(t);
    int mid = node[t].mid();
    if( l < mid ) Updata(L(t),l,r,val);
    if( r > mid ) Updata(R(t),l,r,val);
    Updata_val(t);
}

int Query(int t,int l,int r)
{
    if( node[t].in(l,r) )
        return node[t].max;
    if( node[t].len() == 1 ) return 0;
    Push_down(t);
    int mid = node[t].mid();
    int ans = 0;
    if( r <= mid )
        ans = max(ans, Query(L(t),l,r));
    else
        if( l >= mid )
            ans = max(ans, Query(R(t),l,r));
        else
        {  //因为当前区间可能不完全包括连续值，所以需要以下操作
            ans = max(ans, Query(L(t),l,r));
            ans = max(ans, Query(R(t),l,r));
            int ll, rr;
            if( node[L(t)].r - node[L(t)].rval > l )
                ll = node[L(t)].rval;
            else
                ll = node[L(t)].r - l;
            // 找当前区间右边最大连续值

            if( node[R(t)].l + node[R(t)].lval > r )
                rr = r - node[R(t)].l;
            else
                rr = node[R(t)].lval;
            // 找当前区间左边最大连续值
            ans = max(ans, ll);
```

```
            ans = max(ans, rr);
            if( node[R(t)].ll > node[L(t)].rr )
            {
                int mval = rr + ll;
                ans = max(ans, mval);
            }
        }
    Updata_val(t);
    return ans;
}
```

8.5.10 线段树求区间连续为 N 空白的左端点

```
/*****************************************************************
线段树求区间连续为 N 空白的左端点
Codeforces Beta Round #43 D. Parking Lot
*****************************************************************/
struct Tnode{              // 一维线段树
    int l,r, lval, rval, mval, max, cover;
    int len() { return r - l;}
    int mid() { return MID(l,r);}
    bool in(int ll,int rr) { return l >= ll && r <= rr; }
    void lr(int ll,int rr){ l = ll; r = rr;}
};
Tnode node[MAX<<2];

void Build(int t,int l,int r)
{
    node[t].lr(l,r);
    node[t].lval = node[t].rval = node[t].mval = node[t].max = r - l;
    if( node[t].len() == 1 ) return ;
    int mid = MID(l,r);
    Build(L(t),l,mid);
    Build(R(t),mid,r);
}

void Updata_val(int t)
{
    node[t].lval = node[L(t)].lval;
    if( node[L(t)].lval == node[L(t)].len() )
        node[t].lval += node[R(t)].lval;

    node[t].rval = node[R(t)].rval;
    if( node[R(t)].rval == node[R(t)].len() )
        node[t].rval += node[L(t)].rval;

    node[t].mval = node[L(t)].rval + node[R(t)].lval;

    node[t].max = max(node[L(t)].max, node[R(t)].max);
    node[t].max  =  max(node[t].max,  max(node[t].lval,  max(node[t].rval,
node[t].mval)));
}
void Updata(int t,int l,int r,int val)
{
    if( node[t].in(l,r) )
    {
        node[t].lval = node[t].rval = node[t].mval = node[t].max = (val ?
0 : node[t].len());
        return ;
    }
    if( node[t].len() == 1 ) return ;
    int mid = node[t].mid();
```

```
    if( l < mid ) Updata(L(t),l,r,val);
    if( r > mid ) Updata(R(t),l,r,val);
    Updata_val(t);
}

void Query(int t, int len, int &pos)
{
    if( node[t].max == node[t].len() && node[t].max >= len )
    {
        pos = node[t].l;
        return ;
    }
    if( node[t].max < len ) return ;
    if( node[t].len() == 1 ) return ;
    if( node[L(t)].max >= len )
        Query(L(t), len, pos);
    else
        if( node[t].mval >= len )
        {
            pos = node[L(t)].r - node[L(t)].rval;
            return ;
        }
        else
            if( node[R(t)].max >= len )
                Query(R(t), len, pos);
}
struct NODE
{
    int len, pos;
};
NODE a[110];

int main()
{
    int l, b, f, n, cnt, pos, op, len;

    while( ~scanf("%d%d%d", &l, &b, &f) )
    {
        scanf("%d", &n);
        Build(1, -b, l + f);
        FOR(i, 1, n+1)
        {
            scanf("%d%d", &op, &len);
            if( op == 1 )
            {
                a[i].len = len;
                pos = -1000;
                Query(1, len+b+f, pos);
                if( pos == -1000 )
                {
                    puts("-1");
                    a[i].pos = -1;
                }
                else
                {
                    printf("%d\n", pos+b);
                    a[i].pos = pos + b;
                }
                if( pos != -1000 )
                    Updata(1, pos + b, pos + b + len, 1);
            }
            else
            {
```

```
                pos = a[len].pos;
                Updata(1, pos, pos + a[len].len, 0);
            }
        }
    }

return 0;
}
```

8.5.11 线段树区间加法乘法

```
/*************************************************************************
线段树区间加法乘法
BZOJ 1798: [Ahoi2009]Seq 维护序列 seq

Change operations:
1 a b c 区间[a , b]的每个元素都乘以 c
2 a b c 区间[a , b]的每个元素都加上 c

Output operations:
3 a b 输出区间[a , b]的和 mod d
*************************************************************************/
const int MAX = 100010;
struct Tnode{               // 一维线段树
    int l,r;
    long long sum,mul,add;
    bool ind;
    int len() { return r - l;}
    int mid() { return MID(l,r);}
    bool in(int ll,int rr) { return l >= ll && r <= rr; }
    void lr(int ll,int rr){ l = ll; r = rr;}
};
Tnode node[MAX<<2];
int a[MAX],d;
void init()
{
    memset(node,0,sizeof(node));
}
void Build(int t,int l,int r)
{
    node[t].lr(l,r);
    node[t].mul = 1;
    node[t].add = 0;
    if( node[t].len() == 1 )
    {
        node[t].sum = a[node[t].l];
        return ;
    }
    int mid = MID(l, r);
    Build(L(t), l, mid);
    Build(R(t), mid, r);
    node[t].sum = ( node[L(t)].sum + node[R(t)].sum ) % d;
}
void Mul(long long &sum,long long mul)
{
    sum *= mul; sum %= d;
}
void Add(long long &sum,long long add)
{
    sum += add; sum %= d;
}
void Updata_sub(int t,long long mul,long long add)
{
```

```
    Mul(node[t].sum, mul);
    Add(node[t].sum, add*node[t].len());
    Mul(node[t].add, mul);
    Mul(node[t].mul, mul);
    Add(node[t].add, add);
}
void Pushdown(int t)
{
    if( node[t].mul == 1 && node[t].add == 0 ) return ;
    Updata_sub(L(t), node[t].mul, node[t].add);
    Updata_sub(R(t), node[t].mul, node[t].add);
    node[t].mul = 1; node[t].add = 0;
}
void Updata(int t,int l,int r,int op,long long val)
{
    if( op == 0 && val == 1 || op == 1 && val == 0 ) return ;
    if( node[t].in(l,r) )
    {
        if( op == 0 )
        {
            Mul(node[t].sum, val);
            Mul(node[t].mul, val);
            Mul(node[t].add, val);
        }
        else
        {
            Add(node[t].sum, node[t].len()*val);
            Add(node[t].add, val);
        }
        return ;
    }
    if( node[t].len() == 1 ) return ;
    Pushdown(t);
    int mid = node[t].mid();
    if( l < mid ) Updata(L(t),l,r,op,val);
    if( r > mid ) Updata(R(t),l,r,op,val);
    node[t].sum = ( node[L(t)].sum + node[R(t)].sum ) % d;
}

long long Query(int t,int l,int r)
{
    if( node[t].in(l,r) ) return node[t].sum;
    if( node[t].len() == 1 ) return 0;
    Pushdown(t);
    int mid = node[t].mid();
    long long ans = 0ll;
    if( l < mid ) ans += Query(L(t),l,r);
    ans %= d;
    if( r > mid ) ans += Query(R(t),l,r);
    ans %= d;
    return ans;
}

int main()
{
    int n,m,x,y,z,ind;

    while( ~scanf("%d%d",&n,&d) )
    {
        init();
        for(int i=0; i<n; i++)
            scanf("%d",&a[i]);
```

```
        Build(1,0,n);
        scanf("%d",&m);
        while( m-- )
        {
            scanf("%d",&ind);
            if( ind == 1 )
            {
                scanf("%d%d%d",&x,&y,&z);
                Updata(1,x-1,y,0,z);
            }
            if( ind == 2 )
            {
                scanf("%d%d%d",&x,&y,&z);
                Updata(1,x-1,y,1,z);
            }
            if( ind == 3 )
            {
                scanf("%d%d",&x,&y);
                long long ans = Query(1,x-1,y);
                printf("%lld\n",ans);
            }
        }
    }

return 0;
}
```

8.5.12 线段树区间异或，覆盖，最长序列

```
/*******************************************************************
线段树区间异或，覆盖，最长序列。hdu 3397 Sequence operation
Change operations:
0 a b change all characters into '0's in [a , b]
1 a b change all characters into '1's in [a , b]
2 a b change all '0's into '1's and change all '1's into '0's in [a, b]
Output operations:
3 a b output the number of '1's in [a, b]
4 a b output the length of the longest continuous '1' string in [a , b]
*******************************************************************/
const int MAX = 100010;
struct Tnode{
    int l,r,lb,rb,lw,rw,sb,sw,sum;bool t;short cover;
    int mid() { return MID(l,r);}
    int len() { return r - l;}
    bool in(int ll,int rr) { return l >= ll && r <= rr; }
    void lr(int ll,int rr){ l = ll; r = rr;}
};
Tnode node[MAX*3];
bool aa[MAX];
void init()
{
    memset(aa,false,sizeof(aa));
    memset(node,0,sizeof(node));
}
void Updata_len(int t)
{
    node[t].sum = node[t].len() - node[t].sum;
}
void Swap(int t)
{
    swap(node[t].rw,node[t].rb);
    swap(node[t].lw,node[t].lb);
    swap(node[t].sw,node[t].sb);
```

```
}
void Updata_wb(int t,int v1,int v2)
{
    node[t].rb = node[t].lb = node[t].sb = v1;
    node[t].rw = node[t].lw = node[t].sw = v2;
    node[t].sum = v1;
}
void Updata_cover(int t)
{
    if( node[t].cover == 1 )
    {
        Updata_wb(L(t), 0, node[L(t)].len());
        Updata_wb(R(t), 0, node[R(t)].len());
    }
    else
    {
        Updata_wb(L(t), node[L(t)].len(), 0);
        Updata_wb(R(t), node[R(t)].len(), 0);
    }
}
void Pushdown_len(int t)
{
    if( node[t].cover > 0 && node[t].len() != 1 )
    {
        node[R(t)].cover = node[L(t)].cover = node[t].cover;
        Updata_cover(t);
        node[L(t)].t = node[R(t)].t = 0;
    }
    node[t].cover = 0;
}
void Updata_sum(int t)
{
    node[t].lw = node[L(t)].lw + ( node[L(t)].lw == node[L(t)].len() ?
node[R(t)].lw : 0 );
    node[t].rw = node[R(t)].rw + ( node[R(t)].rw == node[R(t)].len() ?
node[L(t)].rw : 0 );
    node[t].sw = max(node[R(t)].sw, max(node[L(t)].sw, node[L(t)].rw +
node[R(t)].lw));
    node[t].lb = node[L(t)].lb + ( node[L(t)].lb == node[L(t)].len() ?
node[R(t)].lb : 0 );
    node[t].rb = node[R(t)].rb + ( node[R(t)].rb == node[R(t)].len() ?
node[L(t)].rb : 0 );
    node[t].sb = max(node[R(t)].sb, max(node[L(t)].sb, node[L(t)].rb +
node[R(t)].lb));
    node[t].sum = node[L(t)].sum + node[R(t)].sum;
}
void Pushdown_xor(int t)
{
    if( node[t].len() == 1 )
    {
        node[t].t = 0;
        return ;
    }
    if( node[t].t )
    {
        node[R(t)].t = !node[R(t)].t;
        node[L(t)].t = !node[L(t)].t;
        Swap(R(t)); Swap(L(t));
        Updata_len(R(t)); Updata_len(L(t));
        node[t].t = !node[t].t;
    }
}
void Build(int t,int l,int r)
```

```
{
    node[t].lr(l,r);
    if( node[t].len() == 1 )
    {
        node[t].sum = node[t].lb = node[t].rb = node[t].sb = aa[l];
        node[t].lw = node[t].rw = node[t].sw = 1 - aa[l];
        return ;
    }
    int mid = MID(l,r);
    Build(L(t),l,mid);
    Build(R(t),mid,r);
    Updata_sum(t);
}
void Updata(int t,int l,int r,int val)
{
    Pushdown_len(t);
    Pushdown_xor(t);
    if( node[t].in(l,r) )
    {
        if( val != 3 )
        {
            node[t].cover = val;
            if( node[t].cover == 1 )
                Updata_wb(t, 0, node[t].len());
            else
                Updata_wb(t, node[t].len(), 0);
            return ;
        }
        else
        {
            node[t].t = !node[t].t;
            Swap(t);
            Updata_len(t);
            return ;
        }
    }
    if( node[t].len() == 1 ) return ;
    int mid = MID(node[t].l,node[t].r);
    if( l < mid )  Updata(L(t), l, r, val);
    if( r > mid )  Updata(R(t), l, r, val);
    Updata_sum(t);
}

int Query_nor(int t,int l,int r)
{
    Pushdown_len(t);
    Pushdown_xor(t);
    if( node[t].in(l,r) ) return node[t].sum;
    if( node[t].len() == 1 ) return 0;
    int mid = node[t].mid();
    int ans = 0;
    if( l < mid )  ans += Query_nor(L(t),l,r);
    if( r > mid )  ans += Query_nor(R(t),l,r);
    Updata_sum(t);
    return ans;
}

int Query_xor(int t,int l,int r)
{
    Pushdown_len(t);
    Pushdown_xor(t);
    if( node[t].in(l,r) ) return node[t].sb;
    if( node[t].len() == 1 ) return 0;
```

```
    int mid = node[t].mid();
    int ans = 0;
    if( l >= mid )
        ans = max(ans,Query_xor(R(t),l,r));
    else
        if( r <= mid )
            ans = max(ans,Query_xor(L(t),l,r));
        else
        {
            ans = max(ans, Query_xor(L(t), l, mid));
            ans = max(ans, Query_xor(R(t), mid, r));
            int a = ( node[L(t)].rb <= mid - l ? node[L(t)].rb : mid - l);
            int b = ( node[R(t)].lb <= r - mid ? node[R(t)].lb : r - mid);
            ans = max(ans, a+b);
        }
    Updata_sum(t);
    return ans;
}

int main()
{
    int n,m,ncases,a,b,ind;

    scanf("%d",&ncases);

    while( ncases-- )
    {
        init();
        scanf("%d%d",&n,&m);

        for(int i=0; i<n; i++)
        {
            scanf("%d",&a);
            if( a ) aa[i] = 1;                                  //数列的初始值
        }
        Build(1,0,n);
        while( m-- )
        {
            scanf("%d%d%d",&ind,&a,&b);

            if( ind == 0 ) Updata(1,a,b+1,1);
            if( ind == 1 ) Updata(1,a,b+1,2);
            if( ind == 2 ) Updata(1,a,b+1,3);
            if( ind == 3 )
            {
                int ans = Query_nor(1,a,b+1);
                printf("%d\n",ans);
            }
            if( ind == 4 )
            {
                int ans = Query_xor(1,a,b+1);
                printf("%d\n",ans);
            }
        }
    }
return 0;
}
```

8.6 划分树

```
/*****************************************************************
                        划分树求区间第 K 数
*****************************************************************/
```

```
const int MAX = 100010;
class Parti_tree{
public :
    class Tnode{                            // 我的一维线段树定义
    public :
        int l,r;
        int len() { return r - l;}
        int mid() { return MID(l,r);}
        bool in(int ll,int rr) { return l >= ll && r <= rr; }
        void lr(int ll,int rr){ l = ll; r = rr;}
    };
    Tnode node[MAX<<2];
    int num_left[20][MAX], seg[20][MAX],sa[MAX];//sa 数组存 sort 后的结果
                        //seg 数组存的是 d 层划分后的数字 （类似快排 Partation
（d-1）次后的结果）
                        //num_left 存的是 d 层在 i 之前(包括 i)小于 sa[mid] 的数
的数目
    void init()
    {
        memset(seg,0,sizeof(seg));
        memset(num_left,0,sizeof(num_left));
        memset(node,0,sizeof(node));
    }
    void build(int s,int t)
    {
        sort(sa+s,sa+t+s);
        Parti_build(1,s,t,1);
    }
    int query(int s,int t,int k)
    {
        return find_rank(1,s,t,1,k);
    }
    void Parti_build(int t,int l,int r,int d)
    {
        node[t].lr(l, r);
        if( node[t].len() == 0 ) return ;
        int mid = MID(l, r), lsame = mid - l + 1;
        for(int i=l; i<=r; i++)//首先确定分到每一侧的数的数目
            if( seg[d][i] < sa[mid] )//因为相同的元素可能被分到两侧
                lsame--;
        int lpos = l,rpos = mid+1;
        for(int i=l; i<=r; i++)
        {
            if( i == l )
                num_left[d][i] = 0;
            else
                num_left[d][i] = num_left[d][i-1];
            if( seg[d][i] < sa[mid] )
            {
                num_left[d][i]++;
                seg[d+1][lpos++] = seg[d][i];
            }
            if( seg[d][i] > sa[mid] )
                seg[d+1][rpos++] = seg[d][i];
            if( seg[d][i] == sa[mid] )
                if( lsame > 0 )   // 如果大于 0，说明左侧可以分和 sa[mid]相同
的数字
                {
                    lsame--;
                    num_left[d][i]++;
```

```
                seg[d+1][lpos++] = seg[d][i];
            }
            else          //反之，说明左侧数字已经分满了，就分到右边去
                seg[d+1][rpos++] = seg[d][i];
        }
        Parti_build(L(t), l, mid, d+1);
        Parti_build(R(t), mid+1, r, d+1);
    }
    int find_rank(int t,int l,int r,int d,int val)
    {
        if( node[t].len() == 0 ) return seg[d][l];
        int s,ss;             //s表示区间[l,r]有多少个小于sa[mid]的数被分到
左边
        if( l == node[t].l )
            ss = 0;
        else
            ss = num_left[d][l-1];
       s = num_left[d][r] - ss;//ss表示从当前区间的L到l-1有多少个小于sa[mid]
的数被分到左边
        if( s >= val )
            return  find_rank(L(t),  node[t].l+ss,  node[t].l+ss+s-1,  d+1,
val);
        else
        {
            int mid = node[t].mid();
            int bb = l - node[t].l - ss; //表示从当前区间 L 到 l-1 有多少个分
到右边
            int b = r - l + 1 - s;           //表示[l,r]有多少个分到右边
            return find_rank(R(t), mid+bb+1, mid+bb+b,d+1,val-s);
        }
    }
};

Parti_tree t;
int main()
{
    int n,m,x,y,k;

    while( ~scanf("%d%d",&n,&m) )
    {
        t.init();      //可以不要
        for(int i=1; i<=n; i++)
        {
            scanf("%d",&t.sa[i]);
            t.seg[1][i] = t.sa[i];
        }
        t.build(1,n);
        while( m-- )
        {
            scanf("%d%d%d",&x,&y,&k);
            int ans = t.query(x, y, k);
            printf("%d\n",ans);
        }
    }

return 0;
}

/************************************************************************
```

划分树求区间中位数

```
**********************************************************************/
const int MAX = 100010;
LL s[MAX];
class Parti_tree{
public :
    class Tnode{
    public :
        int l,r;
        int len() { return r - l;}
        int mid() { return MID(l,r);}
        bool in(int ll,int rr) { return l >= ll && r <= rr; }
        void lr(int ll,int rr){ l = ll; r = rr;}
    };
    Tnode node[MAX<<2];
    int num_left[20][MAX], seg[20][MAX],sa[MAX];
    LL less_mid, less_midt, less_num, sum_left[20][MAX];
    void init()           //less_mid 是区间内小于中位数的数的和 ，less_num 是
小于中位数的数量
    {                 // sum_left 存的是 d 层 i 之前小于中位数的和
        memset(num_left,0,sizeof(num_left));
        memset(node,0,sizeof(node));
    }
    void build(int s,int t)
    {
        sort(sa+s,sa+t+s);
        Parti_build(1,s,t,1);
    }
    int query(int s,int t,int k)
    {
        less_mid = less_midt = less_num = 0;
        return find_rank(1,s,t,1,k);
    }
    void Parti_build(int t,int l,int r,int d)
    {
        node[t].lr(l, r);
        if( node[t].len() == 0 ) return ;
        int mid = MID(l, r), lsame = mid - l + 1;
        for(int i=l; i<=r; i++)
            if( seg[d][i] < sa[mid] )
                lsame--;
        int lpos = l,rpos = mid+1;
        for(int i=l; i<=r; i++)
        {
            if( i == l )
                num_left[d][i] = sum_left[d][i] = 0;
            else
            {
                num_left[d][i] = num_left[d][i-1];
                sum_left[d][i] = sum_left[d][i-1];
            }
            if( seg[d][i] < sa[mid] )
            {
                num_left[d][i]++;
                seg[d+1][lpos++] = seg[d][i];
                sum_left[d][i] += seg[d][i];
            }
            if( seg[d][i] > sa[mid] )
                seg[d+1][rpos++] = seg[d][i];
            if( seg[d][i] == sa[mid] )
                if( lsame > 0 )
```

```
                {
                    lsame--;
                    num_left[d][i]++;
                    seg[d+1][lpos++] = seg[d][i];
                    sum_left[d][i] += seg[d][i];
                }
                else
                    seg[d+1][rpos++] = seg[d][i];
            }
            Parti_build(L(t), l, mid, d+1);
            Parti_build(R(t), mid+1, r, d+1);
        }
        int find_rank(int t,int l,int r,int d,int val)
        {
            if( node[t].len() == 0 ) return seg[d][l];
            int s,ss;
            if( l == node[t].l )
            {
                s = num_left[d][r];
                less_midt = sum_left[d][r];
                ss = 0;
            }
            else
            {
                s = num_left[d][r] - num_left[d][l-1];
                ss = num_left[d][l-1];
                less_midt = sum_left[d][r] - sum_left[d][l-1];
            }
            if( s >= val )
                return find_rank(L(t), node[t].l+ss, node[t].l+ss+s-1, d+1,
val);
            else
            {
                int mid = node[t].mid();
                int bb = l - node[t].l - ss;
                int b = r - l + 1 - s;
                less_mid += less_midt;
                less_num += s;
                return find_rank(R(t), mid+bb+1, mid+bb+b,d+1,val-s);
            }
        }
};

Parti_tree t;
int main()
{
    int n,m,x,y,k;
    int ncases,ind = 1;
    scanf("%d",&ncases);

    while( ncases-- )
    {
        scanf("%d",&n);
        t.init();
        s[0] = 0;
        for(int i=1; i<=n; i++)
        {
            scanf("%d",&t.sa[i]);
            s[i] = s[i-1] + t.sa[i];
            t.seg[1][i] = t.sa[i];
```

```
        }
        scanf("%d",&m);
        t.build(1,n);
        printf("Case #%d:\n",ind++);
        while( m-- )
        {
            scanf("%d%d",&x,&y);
            if( x == y )
            {
                printf("0\n");
                continue;
            }
            x++; y++;//这题是从 0 开始输入的。= =。。改成从 1 的
            int mid = (y-x)/2+ 1;
            LL ans = t.query(x, y, mid);
            LL sum = s[y] - s[x-1];

            LL suml = t.less_mid;
            LL sumr = sum - suml - ans;
            LL out = t.less_num*ans - suml + sumr - (y - x - t.less_num)*ans;
            printf("%I64d\n",out);
        }
        printf("\n");
    }

return 0;
}
```

8.7 归并树

```
/*************************************************************************
                          归并树求区间第 K 数 （效率没有划分树高）
*************************************************************************/
const int MAX = 100010;
class Merger_tree{
public :
    class Tnode{
    public :
        int l,r;
        int len() { return r - l;}
        int mid() { return MID(l,r);}
        bool in(int ll,int rr) { return l >= ll && r <= rr; }
        void lr(int ll,int rr){ l = ll; r = rr;}
    };
    Tnode node[MAX<<2];
    int seg[20][MAX],a[MAX],n;
    void init()
    {
        memset(seg,0,sizeof(seg));
        memset(node,0,sizeof(node));
    }
    void build(int s,int t){ n = t; Merger_build(1,s,t,1); }
    int query(int x,int y,int k) { return find_rank(n,x,y,k); };
    void Merger_build(int t,int l,int r,int deep)
    {
        node[t].lr(l, r);
        if( node[t].len() == 0 )
        {
            seg[deep][l] = a[l];
            return ;
```

```
        }
        int mid = MID(l, r);
        Merger_build(L(t), l, mid, deep+1);
        Merger_build(R(t), mid+1, r, deep+1);
        int k = l,i = l,j = mid+1;
        while( i <= mid && j <= r )
            if( seg[deep+1][i] < seg[deep+1][j] )
                seg[deep][k++] = seg[deep+1][i++];
            else
                seg[deep][k++] = seg[deep+1][j++];
        while( i <= mid )
            seg[deep][k++] = seg[deep+1][i++];

        while( j <= r )
            seg[deep][k++] = seg[deep+1][j++];

    }
    int find_k(int t,int l,int r,int deep,int val)
    {
        if( node[t].in(l,r) )
        {
            int ll = node[t].l, rr = node[t].r;
            while( ll < rr )
            {
                int mid = MID(ll, rr);
                if( seg[deep][mid] < val )
                    ll = mid + 1;
                else
                    rr = mid;
            }
            if( seg[deep][ll] <= val )
                return ll - node[t].l + 1;
            else
                return ll - node[t].l;
        }
        if( node[t].len() == 0 ) return 0;
        int ans = 0;
        int mid = node[t].mid();
        if( l <= mid ) ans += find_k(L(t), l, r, deep+1, val);
        if( r >= mid ) ans += find_k(R(t), l, r, deep+1, val);
        return ans;
    }
    int find_rank(int n,int x,int y,int k)
    {
        int l = 1,r = n;
        while( l < r )
        {
            int mid = MID(l, r);
            if( find_k(1, x, y, 1, seg[1][mid]) < k )
                l = mid + 1;
            else
                r = mid;
        }
        return seg[1][l];
    }
};

Merger_tree t;
int main()
{
```

```
    int n,m,x,y,k;
    while( ~scanf("%d%d",&n,&m) )
    {
        t.init();
        for(int i=1; i<=n; i++)
            scanf("%d",&t.a[i]);

        t.build(1,n);
        while( m-- )
        {
            scanf("%d%d%d",&x,&y,&k);
            int ans = t.query(x,y,k);
            printf("%d\n",ans);
        }
    }

return 0;
}
```

8.8 Treap

```
int sum;
template<class Tp>
class Treap_Node{
public:
    Tp value;
    int time,fix;
    Treap_Node *left,*right;

    Treap_Node() { puts("you forget something!"); }
    Treap_Node(Tp v,int f){
        left=right=NULL;
        value=v;fix=f;time=1;
    }
};

template<class Tp>
class Treap{
private:
    Treap_Node<Tp> *Root;
    void Left_Rotate( Treap_Node<Tp>* &H ){
        Treap_Node<Tp> *temp = H->right;
        H->right = temp->left;
        temp->left = H;
        H = temp;
    }
    void Right_Rotate( Treap_Node<Tp>* &H ){
        Treap_Node<Tp> *temp = H->left;
        H->left = temp->right;
        temp->right = H;
        H = temp;
    }
    void insert( Treap_Node<Tp>* &H, Tp v ){
        if( H == NULL )
            H = new Treap_Node<Tp>(v,rand());
        else if( v < H->value ){
            insert( H->left, v );
            if( H->left->fix < H->fix )
                Right_Rotate( H );
        }
```

```
        else if( v > H->value ){
            insert( H->right, v );
            if( H->right->fix < H->fix )
                Left_Rotate( H );
        }
        else H->time++;
    }
    bool del( Treap_Node<Tp>* &H, Tp v ){
        if( H == NULL ) return false;
        if( v == H->value ){
            if( H->time > 1 ) {H->time--;return true;}
            if( H->left == NULL || H->right == NULL ){
                Treap_Node<Tp> *t = H;
                H = (H->right==NULL)?H->left:H->right;
                delete t;return true;
            }
            else{
                if( H->left->fix < H->right->fix ){
                    Right_Rotate( H );
                    return del( H->right, v );
                }else{
                    Left_Rotate( H );
                    return del( H->left, v );
                }

            }
        }
        else if( v < H->value )
            return del( H->left, v );
        else
            return del( H->right, v );
    }
    Treap_Node<Tp>* find( Treap_Node<Tp>* H, Tp v ){
        if( H == NULL || v == H->value )
            return H;
        if( v < H->value ) return find( H->left,v );
        else return find( H->right, v );
    }
    void Destroy( Treap_Node<Tp>* H ){
        if( H == NULL ) return;
        Destroy( H->left );
        Destroy( H->right );
        delete H;
    }
    void out( Treap_Node<Tp> *H){
        if( H == NULL ) return;
        out( H->left );
        printf("%s %.4lf\n",H->value.c_str(),H->time*100.0/sum);
        out( H->right );
    }
    Treap_Node<Tp> *L(Treap_Node<Tp>* &H ){
        if( H == NULL ) return H;
        return H->left!=NULL?L(H->left):H;
    }
    Treap_Node<Tp> *R(Treap_Node<Tp>* &H ){
        if( H == NULL ) return H;
        return H->right!=NULL?R(H->right):H;
    }
public:
    Treap() { Root = NULL; }
```

```
    ~Treap() { Destroy( Root );}

    void Insert( Tp v ){ insert( Root, v ); }
    void Delete( Tp v ){    del( Root, v ); }
    Treap_Node<Tp> *Find( Tp v ){ return find(Root,v);}
    Treap_Node<Tp> *Min( void ){return L(Root);}
    Treap_Node<Tp> *Max( void ){return R(Root);}
    void print(void) { out( Root );}
};
int main(void){
    sum = 0;
    char a[100];
    Treap<string> T;
    while(gets(a) && a[0] != '\0'){
        string aa(a);
        T.Insert( aa );
        T.Delete( aa );
        T.Insert( aa );
        sum++;
    }
    T.print();
    return 0;
}
```

8.9 矩形切割

```
//////////////////////////////////////////////////////
//                  矩形切割 v2.0                   //
//////////////////Documentation//////////////////////
//                                                  //
// 1. class Rect:   矩形类，包括矩形坐标和颜色;     //
// 2. clear:        清空所有矩形;                   //
// 3. insert:       插入一个矩形;                   //
// 4. get_color:    获取各个颜色的面积统计;         //
// 5. total_area:   获取面积总和;                   //
// 6. color:        get_color 简洁版;               //
//                                                  //
//////////////////////////////////////////////////////
//            2008 CopyRight(c) by elf              //
//////////////////////////////////////////////////////
//            2011 revised by fookwood              //
//////////////////////////////////////////////////////

using namespace std;

// clr >= 0 为合法颜色, clr = -1 的方块为哨兵
template<class T>
class Rect {
public:
    T x1, y1, x2, y2;
    int clr;
    Rect() {}
    Rect( T a, T b, T c, T d, int clr ) :
        x1(a), y1(b), x2(c), y2(d), clr(clr) {}
    T area() const { return (y2-y1)*(x2-x1); }
};

const Rect<int> SENTINEL( 0, 0, 0, 0, -1 );

template<class T>
```

```
class RectCut {

    deque< Rect<T> >  Q;
    map<int, T>       M;

public:

    // 在队列中放置一个 sentinel
    void clear() { Q.assign( 1, SENTINEL ); M.clear(); }

    void insert( T a, T b, T c, T d, int clr ) {
        insert( Rect<T>( a, b, c, d, clr ) );
    }

    void insert( Rect<T> r ) {
        M[r.clr] += r.area();
        Q.push_back( r );
        while( Q.front().clr != -1 ) {
            Rect<T> z = Q.front();
            Q.pop_front();
            if( z.x1 >= r.x2 || z.x2 <= r.x1 ||
                z.y1 >= r.y2 || z.y2 <= r.y1 ) {
                Q.push_back( z );
                continue;
            }
            if( z.x1 < r.x1 ) {
                Q.push_back( Rect<T>( z.x1, z.y1, r.x1, z.y2, z.clr ) );
                z.x1 = r.x1;
            }
            if( z.x2 > r.x2 ) {
                Q.push_back( Rect<T>( r.x2, z.y1, z.x2, z.y2, z.clr ) );
                z.x2 = r.x2;
            }
            if( z.y1 < r.y1 ) {
                Q.push_back( Rect<T>( z.x1, z.y1, z.x2, r.y1, z.clr ) );
                z.y1 = r.y1;
            }
            if( z.y2 > r.y2 ) {
                Q.push_back( Rect<T>( z.x1, r.y2, z.x2, z.y2, z.clr ) );
                z.y2 = r.y2;
            }
            if( ( M[z.clr] -= z.area() ) == 0 ) M.erase( M.find( z.clr ) );
        }
        Q.push_back( Q.front() );
        Q.pop_front();
    }
    T color( int co ){
        return M[co];
    }
    vector<pair<int, T> > get_color() const {
        return  vector<pair<int, T> >( M.begin(), M.end() );
    }
    T total_area(){
        T sum = 0;
        vector< pair<int,T> > V = get_color();
        for( int i = 0; i < V.size(); i++ )
            sum += V[i].second;
        return sum;
    }
};
```

```
//////////////////////////////////////////////////
//            2008 CopyRight(c) by elf          //
//////////////////////////////////////////////////
//演示程序~zoj 1128~
double a, b, c, d;

int T = 1, N;
RectCut<double> RC;

int main() {
    while(cin >> N && N) {
        RC.clear();
        while(N--) {
            cin >> a >> b >> c >> d;
            RC.insert(a, b, c, d, 1);
        }
        printf("Test case #%d\n", T++);
        printf("Total explored area: %.2lf\n",
               RC.get_color().front().second);
        puts("");
    }
}
```

8.10 最近公共祖先

```
const int maxn = 50001;
using namespace std;
class Node{
public:
    int to,w;
    Node *next;
    void add( int tt,int ww,Node *&b ){
        to = tt;w= ww;next = b;b=this;
    }
};
int ab(int a,int b){return a>b?(a-b):(b-a);}
Node *biao[ maxn ],a[ 200010 ];
int TIME,countt,N;
int founder[ maxn ];
int timepoint[ maxn*2 ];
int timedeep [ maxn*2 ];
int opt[ maxn*2 ][ 20 ];
int flag[ maxn ];
int d[ maxn ],pre[maxn];
void get(int u,int deep){
    timedeep[ founder[u] = TIME++ ] = deep;
    timepoint[ TIME-1 ] = u;
    flag[u] = 1;
}
inline void dfs( int u,int deep ){

    timedeep[ founder[u] = TIME++ ] = deep;
    timepoint[ TIME-1 ] = u;
    flag[u] = 1;
    for( Node*p=biao[u]; p != NULL; p = p->next )
        if( !flag[p->to] ){
            d[p->to] = d[u] + p->w;
            dfs( p->to , deep +1 );
            timepoint[TIME++] = u;
```

```
            timedeep[TIME-1]= deep;
        }
}

int main(void){
    int i,from,m,to,w;
    int j,tt,countt;
    int start,end,u,v,k,s = 0;
    while( scanf("%d",&N) != EOF ){
        if( s++ ) cout << endl;
        for( i = 0; i < N; i++ )
            biao[i] = NULL;
        countt = 0;
        for( i = 1; i < N; i++ ){
            scanf("%d%d%d",&from,&to,&w);
            a[countt++].add( to,w,biao[from] );
            a[countt++].add( from,w,biao[to] );
        }

        for( i = 0; i < N; i++ )
            flag[i]  = 0;
        d[3*N/4] = 0;
        TIME = 0;
        dfs( 3*N/4,0 );

        for( i = 0; i < 2*N -1 ; i++ )
            opt[i][0] = i;

        for( j = 1; (1<<j) <= 2*N-1; j++ ){
            tt = 1<<j;
            for( i = 0; i + tt -1 < 2*N -1 ; i++ ){
                if( timedeep[ opt[i][j-1] ] <
                    timedeep[ opt[i+tt/2][j-1] ] )
                    opt[i][j] = opt[i][j-1];
                else
                    opt[i][j] = opt[i+tt/2][j-1];
            }
        }

        cin >> m;
        while( m-- ){

            scanf("%d%d",&u,&v);
            start = min( founder[u],founder[v] );
            end   = max( founder[u],founder[v] );
            k = (int)(log((double)(end-start+1))/log(2.0) );
            int p1 = opt[start][k];
            int p2 = opt[end-(1<<k)+1][k];
            int p;
            if( timedeep[p1] > timedeep[p2] )
                p = timepoint[ p2 ];
            else p = timepoint[ p1 ];
            cout << d[u]-d[p]+d[v]-d[p]<<endl;
        }
    }
    return 0;
}
```

9 计算几何

9.1 注意事项

1、可能输出-0.00000;

9.2 各种公式

//[Pick 定理]　设以整数点为顶点的多边形的面积为 S，多边形内部的整数点数为 N，多边形边界上的整数点数为 L，则
N + L/2 - 1 = S

//涉及乘法的时候 注意保证不越界 而且要保证精度

费马点
对于三角形：当三个角都小于 120 度，则费马点在三角形内部与任意两定点的连线构成的角都为 120 度；
//若存在一个角大于等于 120 度，则费马点为此角顶点。
//对于平面四边形：若为凸四边形，则费马点为两对角线交点；对于凹四边形，为其凹顶点。
//在不是凸四边形的情况下，只要枚举四个顶点即可。
//计算四面体体积，欧拉四面体公式
//a=OA，b=OB，c=OC，l=AB，m=BC，n=CA

```
double v_4mianti(double a,double b,double c,double l,double n,double m)
{
    return sqrt(4*a*a*b*b*c*c-a*a*(b*b+c*c-m*m)*(b*b+c*c-m*m)-

    b*b*(c*c+a*a-n*n)*(c*c+a*a-n*n)-c*c*(a*a+b*b-l*l)*(a*a+b*b-l*l)
               +(a*a+b*b-l*l)*(b*b+c*c-m*m)*(c*c+a*a-n*n))/12;
}
```

三角形：
1．半周长 P=(a+b+c)/2
2．面积 S=aHa/2=absin(C)/2=sqrt(P(P-a)(P-b)(P-c))
3．中线 Ma=sqrt(2(b^2+c^2)-a^2)/2=sqrt(b^2+c^2+2bccos(A))/2
4．角平分线 Ta=sqrt(bc((b+c)^2-a^2))/(b+c)=2bccos(A/2)/(b+c)
5．高线 Ha=bsin(C)=csin(B)=sqrt(b^2-((a^2+b^2-c^2)/(2a))^2)
6．内切圆半径 r=S/P=asin(B/2)sin(C/2)/sin((B+C)/2)
=4Rsin(A/2)sin(B/2)sin(C/2)=sqrt((P-a)(P-b)(P-c)/P)
=Ptan(A/2)tan(B/2)tan(C/2)
7．外接圆半径 R=abc/(4S)=a/(2sin(A))=b/(2sin(B))=c/(2sin(C))

四边形：
D1,D2 为对角线,M 对角线中点连线,A 为对角线夹角
1．a^2+b^2+c^2+d^2=D1^2+D2^2+4M^2
2．S=D1D2sin(A)/2
(以下对圆的内接四边形)
3．ac+bd=D1D2
4．S=sqrt((P-a)(P-b)(P-c)(P-d)),P 为半周长

正 n 边形：
R 为外接圆半径,r 为内切圆半径
1．中心角 A=2PI/n
2．内角 C=(n-2)PI/n
3．边长 a=2sqrt(R^2-r^2)=2Rsin(A/2)=2rtan(A/2)
4．面积 S=nar/2=nr^2tan(A/2)=nR^2sin(A)/2=na^2/(4tan(A/2))

圆:
1. 弧长 l=rA
2. 弦长 a=2sqrt(2hr-h^2)=2rsin(A/2)
3. 弓形高 h=r-sqrt(r^2-a^2/4)=r(1-cos(A/2))=atan(A/4)/2
4. 扇形面积 S1=rl/2=r^2A/2
5. 弓形面积 S2=(rl-a(r-h))/2=r^2(A-sin(A))/2

棱柱:
1. 体积 V=Ah,A 为底面积,h 为高
2. 侧面积 S=lp,l 为棱长,p 为直截面周长
3. 全面积 T=S+2A

棱锥:
1. 体积 V=Ah/3,A 为底面积,h 为高
(以下对正棱锥)
2. 侧面积 S=lp/2,l 为斜高,p 为底面周长
3. 全面积 T=S+A

棱台:
1. 体积 V=(A1+A2+sqrt(A1A2))h/3,A1.A2 为上下底面积,h 为高
(以下为正棱台)
2. 侧面积 S=(p1+p2)l/2,p1.p2 为上下底面周长,l 为斜高
3. 全面积 T=S+A1+A2

圆柱:
1. 侧面积 S=2PIrh
2. 全面积 T=2PIr(h+r)
3. 体积 V=PIr^2h

圆锥:
1. 母线 l=sqrt(h^2+r^2)
2. 侧面积 S=PIrl
3. 全面积 T=PIr(l+r)
4. 体积 V=PIr^2h/3

圆台:
1. 母线 l=sqrt(h^2+(r1-r2)^2)
2. 侧面积 S=PI(r1+r2)l
3. 全面积 T=PIr1(l+r1)+PIr2(l+r2)
4. 体积 V=PI(r1^2+r2^2+r1r2)h/3

球:
1. 全面积 T=4PIr^2
2. 体积 V=4PIr^3/3

球台:
1. 侧面积 S=2PIrh
2. 全面积 T=PI(2rh+r1^2+r2^2)
3. 体积 V=PIh(3(r1^2+r2^2)+h^2)/6

球扇形:
1. 全面积 T=PIr(2h+r0),h 为球冠高,r0 为球冠底面半径
2. 体积 V=2PIr^2h/3

Euler 的任意四面体体积公式（已知边长求体积）

$$V=\frac{\sqrt{\begin{vmatrix} 1 & 1 & 1 & 1 \\ x1 & x2 & x3 & x4 \\ y1 & y2 & y3 & y4 \\ z1 & z2 & z3 & z4 \end{vmatrix}}}{12}$$

已知 4 点坐标求体积（其中四个点的坐标分别为（x1,y1,z1),(x2,y2,z2),(x3,y3,z3),(x4,y4,z4)）

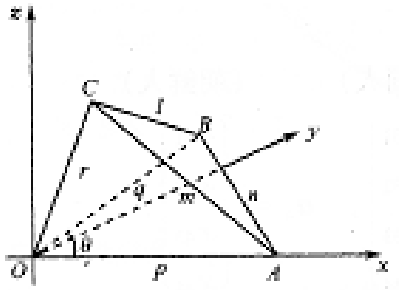

图 2.1 六条棱长已知的四面体

$$36V^2=\begin{vmatrix} p^2 & \frac{p^2+q^2-n^2}{2} & \frac{p^2+r^2-m^2}{2} \\ \frac{p^2+q^2-n^2}{2} & q^2 & \frac{q^2+r^2-l^2}{2} \\ \frac{p^2+r^2-m^2}{2} & \frac{q^2+r^2-l^2}{2} & r^2 \end{vmatrix}.$$

9.3 基本类型定义

```
//二维
struct point{ double x,y;};      //点
struct beeline{   double A,B,C;};   //直线 (直线方程)
struct line{ point a,b;};    //直线 （两点式）
struct segment{ point a,b;}; // 线段

//三维
struct point3 {
    double x, y, z;
    point3(){};
    point3(double xx, double yy, double zz) : x(xx), y(yy), z(zz) {}
};
struct plane{ point3 a, b, c;};
```

9.4 基础函数

```
// 大小比较
const double eps = 1e-6;
bool dy(double x,double y)   {   return x > y + eps;}  // x > y
bool xy(double x,double y)   {   return x < y - eps;}  // x < y
bool dyd(double x,double y)  {   return x > y - eps;}  // x >= y
bool xyd(double x,double y)  {   return x < y + eps;}  // x <= y
bool dd(double x,double y)   {   return fabs( x - y ) < eps;}  // x == y
double disp2p(point a,point b) //  a b 两点之间的距离
{
    return sqrt( ( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y ) );
}
// 叉积 (三点)
double crossProduct(point a,point b,point c)//向量 ac 在 ab 的方向 顺时针是
正
{
    return (c.x - a.x)*(b.y - a.y) - (b.x - a.x)*(c.y - a.y);
}
//叉积（三点坐标）
double  crossProduct(double  x0,double  y0,double  x1,double  y1,double
x2,double y2)
{
    return (x2 - x0)*(y1 - y0) - (x1 - x0)*(y2 - x0);
}
```

9.5 各种极角排序

```
/*********************************************************************
        极角排序，以 C 为顶点，按象限极角排序，相同角度，距离近的排前面
*********************************************************************/
int quad(point a)// 判断象限的函数，每个象限包括半个坐标轴
```

```
{
    if( dy(a.x,0) && xyd(a.y,0) ) return 1;
    if( xyd(a.x,0) && dy(a.y,0) ) return 2;
    if( xy(a.x,0) && xyd(a.y,0) ) return 3;
    if( dyd(a.x,0) && xy(a.y,0) ) return 4;
    return 0;      // 和原点重合 ，一般不会有这种数据的。。
}
bool cmp(point& a,point& b)
{
    point p1 = a,p2 = b;
    p1.x -= C.x; p1.y -= C.y;
    p2.x -= C.x; p2.y -= C.y;
    int l1,l2;
    l1 = quad(p1); l2 = quad(p2);
    if( l1 == l2 )
    {
        double c = crossProduct(C,a,b);
        return xy(c,0) || dd(c,0.0) && xy(fabs(a.x),fabs(b.x));
    }
    return l1 < l2;
}
/*********************************************************************
               极角排序，以 C 为顶点，按照向量排序，范围(-180°, +180°]
*********************************************************************/
bool cmp(point& a,point& b)
{
    double t1 = atan2(a.y - C.y, a.x - C.x);
    double t2 = atan2(b.y - C.y, b.x - C.x);
    if( dd(t1, t2) ) return xy(fabs(a.x),fabs(b.x));
    return xy(a.t, b.t);
}
```

9.6 点、直线、线段

9.6.1 两直线位置关系

```
/*********************************************************************
           线的位置关系：   平行 垂直 相交即不平行
*********************************************************************/
//判断直线是否平行，平行返回 1
bool parallel(point u1,point u2,point v1,point v2)
{
    return dd( (u1.x - u2.x)*(v1.y - v2.y) - (v1.x - v2.x)*(u1.y - u2.y) ,
0.0 );
}
bool parallel(line u,line v)
{
    return dd( (u.a.x - u.b.x)*(v.a.y - v.b.y) - (v.a.x - v.b.x)*(u.a.y -
u.b.y) , 0.0 );
}
//判断两直线是否垂直，垂直返回 1
bool perpendicular(point u1,point u2,point v1,point v2)
{
    return dd((u1.x - u2.x)*(v1.x - v2.x) + (u1.y - u2.y)*(v1.y - v2.y),0.0);
}
```

9.6.2 判断两线段是否相交

```
//判断线段 p1p2 与 p3p4 是否相交，包括端点与端点，端点与线段
const double eps = 1e-10;
```

```
struct point{  double x,y;       };
double crossProduct(point a, point b, point c )
bool onSegment(point a, point b, point c)
bool s2s_inst(point p1,point p2, point p3, point p4)
{
    double d1 = crossProduct(p3,p4,p1);
    double d2 = crossProduct(p3,p4,p2);
    double d3 = crossProduct(p1,p2,p3);
    double d4 = crossProduct(p1,p2,p4);
    if( xy(d1 * d2,0.0) && xy(d3 * d4,0.0) ) return true;
           //如果不判端点相交，则下面这句话不需要
    if(    dd(d1,0.0)    &&    onSegment(p3,p4,p1)    ||    dd(d2,0.0)    &&
onSegment(p3,p4,p2)
     ||    dd(d3,0.0)    &&    onSegment(p1,p2,p3)    ||    dd(d4,0.0)    &&
onSegment(p1,p2,p4) )
        return true;
    return false;
}
```

9.6.3 判断两直线是否相交

不平行即相交

9.6.4 判断直线和线段是否相交

```
//判断直线和线段是否相交，共线的不算相交
double crossProduct(point a,point b,point c)//向量 ac 在 ab 的方向
bool s2l_inst(point s1,point s2,point l1,point l2)//s 是线段，l 是直线
{   // xyd 包括端点在直线上。xy 是穿过
    return xyd(crossProduct(l1,l2,s1) * crossProduct(l1,l2,s2),0.0);
}
```

9.6.5 判断两向量位置

```
//判断向量 a1a2 与 b1b2 的位置
double rota_angle(point a1,point a2,point b1,point b2) //返回 b1b2 在 a1a2
的方向
{
    point t;
    t.x = b2.x - (b1.x - a1.x);
    t.y = b2.y - (b1.y - a1.y);
    return crossProduct(a1,a2,t);
}
```

9.6.6 求两直线交点

```
//求两直线的交点（先判断相交） ,注意可能得到-0.00000
point l2l_inst_p(point u1,point u2,point v1,point v2)
{
    point ans = u1;
    double t = ((u1.x - v1.x)*(v1.y - v2.y) - (u1.y - v1.y)*(v1.x - v2.x))/
               ((u1.x - u2.x)*(v1.y - v2.y) - (u1.y - u2.y)*(v1.x - v2.x));
    ans.x += (u2.x - u1.x)*t;
    ans.y += (u2.y - u1.y)*t;
    return ans;
}
point l2l_inst_p(line l1,line l2)
{
    point ans = l1.a;
    double  t  =  ((l1.a.x  -  l2.a.x)*(l2.a.y  -  l2.b.y)  -  (l1.a.y  -
l2.a.y)*(l2.a.x - l2.b.x))/
```

```
            ((l1.a.x - l1.b.x)*(l2.a.y - l2.b.y) - (l1.a.y -
l1.b.y)*(l2.a.x - l2.b.x));
    ans.x += (l1.b.x - l1.a.x)*t;
    ans.y += (l1.b.y - l1.a.y)*t;
    return ans;
}
```

9.6.7 求两线段交点

```
//求两线段的交点（先判断相交），然后按两直线求交点求得
point s2s_inst_p(point u1,point u2,point v1,point v2)
{
    point ans = u1;
    double t = ((u1.x - v1.x)*(v1.y - v2.y) - (u1.y - v1.y)*(v1.x - v2.x))/
               ((u1.x - u2.x)*(v1.y - v2.y) - (u1.y - u2.y)*(v1.x - v2.x));
    ans.x += (u2.x - u1.x)*t;
    ans.y += (u2.y - u1.y)*t;
    return ans;
}
```

9.6.8 判断点是否在线段上

```
//判断点 c 是否在线段 ab 上。
double crossProduct(point a,point b,point c)
bool onSegment(point a, point b, point c)
{
    double maxx = max(a.x,b.x);
    double maxy = max(a.y,b.y);
    double minx = min(a.x,b.x);
    double miny = min(a.y,b.y);
    if( dd(crossProduct(a,b,c),0.0) && dyd(c.x,minx) && xyd(c.x,maxx) &&
dyd(c.y,miny) && xyd(c.y,maxy) )
        return true;
    return false;
}
```

9.6.9 点到线的距离，点到线段的最小距离

```
/*******************************************************************
           点到线的距离，点到线段的最小距离
*******************************************************************/
//点到直线的距离
double disp2l(point a,point l1,point l2)
{
    return fabs( crossProduct(a,l1,l2) )/disp2p(l1,l2);
}
//点到线段的最短距离
double crossProduct(point a,point b,point c)
double disp2p(point a,point b)
double disp2seg(point p,point l1,point l2)
{
    point t = p;
    t.x += l1.y - l2.y;  t.y += l2.x - l1.x;
    if( dyd(crossProduct(l1,t,p)*crossProduct(l2,t,p),0.0) ) //包括点和线
段共线
        return xy(disp2p(p,l1),disp2p(p,l2)) ? disp2p(p,l1) : disp2p(p,l2);
    return fabs( crossProduct(p,l1,l2) )/disp2p(l1,l2);
}
```

9.6.10 判断点是否在线段上

```
/*********************************************************************
                判断点 c 是否在线段 ab 上。
*********************************************************************/
//判断点 c 是否在线段 ab 上。 包括端点，如果不包括的话 把 xyd -> xy
double crossProduct(point a,point b,point c)
bool onSegment(point a, point b, point c)
{
    if( dd(crossProduct(a,b,c),0.0) && dyd(c.x,min(a.x,b.x)) &&
        xyd(c.x,max(a.x,b.x))      &&      dyd(c.y,min(a.y,b.y))      &&
xyd(c.y,max(a.y,b.y)) )
        return true;
    return false;
}
```

9.6.11 求垂点（不是垂足）

```
// 求一个点，使得 ab 垂直于 l1l2
point foot_line(point a,point l1,point l2)  //ac 在 l1l2 的逆时针方向
{
    point c;
    c.x = a.x - l2.y + l1.y;
    c.y = a.y + l2.x - l1.x;
    return c;
}
```

9.6.12 通过两点求直线方程

```
// 通过两点求直线（Ax + By + C = 0）
struct beeline{       double A,B,C;  };
beeline makeline(point a,point b)
{
    beeline line;
    line.A = a.y - b.y;
    line.B = a.x - b.x;
    line.C = -line.A * a.x - line.B*a.y;
    return line;
}
```

9.6.13 向量的旋转

```
//向量的旋转 //底边线段 ab 绕 a 逆时针旋转角度 ang(弧度制)，原 Whirl 函数
point Rotate(double ang, point a, point b)
{
    b.x -= a.x; b.y -= a.y;
    point c;
    c.x = b.x * cos(ang) - b.y * sin(ang) + a.x;
    c.y = b.x * sin(ang) + b.y * cos(ang) + a.y;
    return c;
}
```

9.6.14 判断点是否在多边形内

```
//判断点是否在多边形内 凸凹均可
double crossProduct(point a,point b,point c)//向量 ac 在 ab 的方向
bool onSegment(point a, point b, point c)
bool s2s_inst(point p1,point p2, point p3, point p4)
// p 中按顺时针或者逆时针方向存多边形的点，n 为多边形的点数
bool point_inPolygon(point pot,point p[],int n)
```

```
{
    int count = 0;
    point a = pot,b;
    b.x = 1e20; b.y = pot.y;
    p[n] = p[0];
    for(int i=0; i<n; i++)
    {
        if( onSegment(p[i],p[i+1],pot) ) return true;
        if( !dd(p[i].y,p[i+1].y) )
        {
            int tmp = -1;
            if( onSegment(a,b,p[i]) )
                tmp = i;
            else
                if( onSegment(a,b,p[i+1]) )
                    tmp = i+1;
            if( tmp != -1 && dd(p[tmp].y,max(p[i].y,p[i+1].y)) ||
                tmp == -1 && s2s_inst(p[i],p[i+1],a,b) )
                count++;
        }
    }
    return count % 2 == 1;
}

//判断点 a 是否在凸多边形内，p 数组至少开[n + 2]个点
bool pin_convexh(point *p,int n,point a)
{
    p[n] = p[0]; p[n+1] = p[1];
    for(int i=0; i<n; i++)
        if( xy(crossProduct(p[i],p[i+1],a)*
            crossProduct(p[i+1],p[i+2],a),0.0) )
            return false;
    return true;
}
```

9.7 三角形

9.7.1 计算三角形面积，边长

```
/*****************************************************************
          计算三角形面积||边长，已知三高 || 三点 || 三边 || 三中线
*****************************************************************/

//计算三角形面积（已知三角形的三条高）
const double eps = 1e-8; // 精度要高
double area_triangle_h(double x,double y,double z)
{
    if( dd(x,0.0) || dd(y,0.0) || dd(z,0.0) ) return -1; // 构不成三角形
    double p =  (1/x +  1/y + 1/z)*(1/y + 1/z - 1/x)*(1/x + 1/z - 1/y)*(1/x
+ 1/y - 1/z);
    if( xyd(p,0.0) ) return -1; // 构不成三角形
    return 1/sqrt(p);
}
//计算三角形面积(三点)
double area_triangle(point a,point b,point c)
{
    return fabs( crossProduct(a,b,c) )/2.0;
}
//计算三角形面积（三边海伦公式）
```

```
double area_triangle(double a,double b,double c)
{
    double p = (a+b+c)/2.0;
    return sqrt(p*(p-a)*(p-b)*(p-c));
}
//计算三角形面积（三点，坐标）
double  area_triangle(double  x0,double  y0,double  x1,double  y1,double
x2,double y2)
{
    return fabs( crossProduct(x0,y0,x1,y1,x2,y2) )/2.0;
}
//计算三角形面积（已知三条中线长）
double area_midline(double x,double y,double z)
{
    return 4.0/3*area_triangle(x,y,z);
}
//公式 已知中线 x y z 求边长 a b c
void triangle_edge(double x,double y,double z,double &a,double &b,double
&c)
{
    a = 2.0/3.0*sqrt(2*x*x + 2*z*z - y*y);
    b = 2.0/3.0*sqrt(2*y*y + 2*z*z - x*x);
    c = 2.0/3.0*sqrt(2*x*x + 2*y*y - z*z);
}
```

9.7.2 计算三角形的外接圆半径，内切圆半径

```
double circumcir_r(double a,double b,double c)//已知三边求外接圆半径
{
    return a*b*c/(4*area_triangle(a,b,c));
}
double  circumcir_r(point ap,point bp,point cp)//已知三点求外接圆半径
{
    double a,b,c;
    a = disp2p(ap,bp); b = disp2p(bp,cp); c = disp2p(cp,ap);
    return a*b*c/sqrt((a+b+c)*(a+b-c)*(a+c-b)*(b+c-a));
}

double incir_r(double a,double b,double c)//已知三边求内切圆半径
{
    return 2*area_triangle(a,b,c)/(a+b+c);
}
double incir_r(point ap,point bp,point cp)//已知三点求内切圆半径
{
    double a,b,c;
    a = disp2p(ap,bp); b = disp2p(bp,cp); c = disp2p(cp,ap);
    return area_3p(ap,bp,cp)*2.0/(a+b+c);
}
```

9.7.3 各种心（外心，内心，垂心，重心，费马点）

```
//计算三角形重心
//到三角形三点距离的平方和最小的点
//三角形内到三边距离之积最大的点
point l2l_inst_p(point u1,point u2,point v1,point v2)
point barycenter(point a,point b,point c)
{
    point u1,u2,v1,v2;
    u1.x = (a.x + b.x)/2;
    u1.y = (a.y + b.y)/2;
```

```
    v1.x = (a.x + c.x)/2;
    v1.y = (a.y + c.y)/2;
    return l2l_inst_p(u1,c,v1,b);
}
//三角形的内心
point incenter(point a,point b,point c)
{
    line u,v;
    double m,n;
    u.a = a;
    m = atan2(b.y - a.y, b.x - a.x);
    n = atan2(c.y - a.y, c.x - a.x);
    u.b.x = u.a.x + cos((m+n)/2);
    u.b.y = u.a.y + sin((m+n)/2);
    v.a = b;
    m = atan2(a.y - b.y, a.x - b.x);
    n = atan2(c.y - b.y, c.x - b.x);
    v.b.x = v.a.x + cos((m+n)/2);
    v.b.y = v.a.y + sin((m+n)/2);
    return l2l_inst_p(u,v);
}
//计算三角形外心（外接圆圆心）
point l2l_inst_p(line u,line v)
point circumcenter(point a,point b,point c)
{
    point ua,ub,va,vb;
    ua.x = ( a.x + b.x )/2;
    ua.y = ( a.y + b.y )/2;
    ub.x = ua.x - a.y + b.y;//根据 垂直判断，两线段点积为0
    ub.y = ua.y + a.x - b.x;
    va.x = ( a.x + c.x )/2;
    va.y = ( a.y + c.y )/2;
    vb.x = va.x - a.y + c.y;
    vb.y = va.y + a.x - c.x;
    return l2l_inst_p(ua,ub,va,vb);
}
//这种不需要相交的那个函数 （外接圆圆心）
point circumcenter(point a,point b,point c)
{
    point ret;
    double a1 = b.x - a.x, b1 = b.y - a.y, c1 = (a1*a1 + b1*b1)/2;
    double a2 = c.x - a.x, b2 = c.y - a.y, c2 = (a2*a2 + b2*b2)/2;
    double d = a1 * b2 - a2 * b1;
    ret.x = a.x + (c1*b2 - c2*b1)/d;
    ret.y = a.y + (a1*c2 - a2*c1)/d;
 return ret;
}
//三角形的垂心（垂线的交点）
point perpencenter(point a,point b,point c)
{
    point ua,ub,va,vb;
    ua = c;
    ub.x = ua.x - a.y + b.y;
    ub.y = ua.y + a.x - b.x;
    va = b;
    vb.x = va.x - a.y + c.y;
    vb.y = va.y + a.x - c.x;
    return l2l_inst_p(ua,ub,va,vb);
}
//费马点
```

```
//到三角形三顶点距离之和最小的点
point fermentpoint(point a,point b,point c){
    point u,v;
    double
step=fabs(a.x)+fabs(a.y)+fabs(b.x)+fabs(b.y)+fabs(c.x)+fabs(c.y);
    int i,j,k;
    u.x=(a.x+b.x+c.x)/3;
    u.y=(a.y+b.y+c.y)/3;
    while (step>1e-10)
        for (k=0;k<10;step/=2,k++)
            for (i=-1;i<=1;i++)
                for (j=-1;j<=1;j++){
                    v.x=u.x+step*i;
                    v.y=u.y+step*j;
                    if
(distance(u,a)+distance(u,b)+distance(u,c)>distance(v,a)+distance(v,b)+
distance(v,c))
                        u=v;
                }
    return u;
}
```

9.8 圆

9.8.1 圆的位置关系

```
/********************************************************************
           圆的位置关系：判断 相交 相切 内含
                         求 交点 切点
           圆和矩形是否相交
********************************************************************/
//判断两圆是否相交 (不包括相切)
bool c2c_inst(point a,double r1,point b,double r2)
{
    if( xy(disp2p(a,b),r1+r2) && dy(disp2p(a,b),fabs(r1 - r2)) )
        return true;
    return false;
}
//求 直线与圆的交点（保证有交点）
// 也可计算 线段与圆的交点，然后求得交点判断是否在线段上
void l2c_inst_p(point c,double r,point l1,point l2,point &p1,point &p2)
{
    point p = c;
    double t;
    p.x += l1.y - l2.y;
    p.y += l2.x - l1.x;
    p = l2l_inst_p(p,c,l1,l2);
    t = sqrt(r*r - disp2p(p,c)*disp2p(p,c))/disp2p(l1,l2);
    p1.x = p.x + (l2.x - l1.x)*t;
    p1.y = p.y + (l2.y - l1.y)*t;
    p2.x = p.x - (l2.x - l1.x)*t;
    p2.y = p.y - (l2.y - l1.y)*t;
}
//求两圆交点（先判断相交）
//交得弧是弧 p1->p2
void c2c_inst_p(point c1,double r1,point c2,double r2,point &p1,point &p2)
{
    point u,v;
    double t;
```

```
    t = (1 + (r1*r1 - r2*r2)/disp2p(c1,c2)/disp2p(c1,c2))/2;
    u.x = c1.x + (c2.x - c1.x)*t;
    u.y = c1.y + (c2.y - c1.y)*t;
    v.x = u.x + c1.y - c2.y;
    v.y = u.y - c1.x + c2.x;
    l2c_inst_p(c1,r1,u,v,p1,p2);
}

//判断两圆是否相切 （外切和内切）
bool c2c_tangent(point a,double r1,point b,double r2)
{
    if( dd(disp2p(a,b),r1+r2) || dd(disp2p(a,b),fabs(r1-r2)) )
        return true;
    return false;
}
//求两圆切点（先判断相切，内外切交点不一样哎）
point c2c_tangent_p(point a,double r1,point b,double r2)
{
    point t;
    if( dd(disp2p(a,b),r1 + r2)  ) // 外切交点
    {
        t.x = (r1*b.x + r2*a.x)/(r1 + r2);
        t.y = (r1*b.y + r2*a.y)/(r1 + r2);
        return t;
    }
    t.x = (r1*b.x - r2*a.x)/(r1 - r2);
    t.y = (r1*b.y - r2*a.y)/(r1 - r2);
    return t;
}
//判断两圆是否内含（a 内含于 b，不包括相切）
bool c2c_ainb(point a,double r1,point b,double r2)
{
    return xy(disp2p(a,b),r2 - r1);  //a 在 b 中，如果是包括内切，用 xyd
}
```

9.8.2 两圆相交面积

```
double gongxing_area(double r1,double r2,double l) // 两圆半径以及连心线长
度
{
    double cosaa = (r1*r1 + l*l - r2*r2)/(2*r1*l);      // cos 值
    double fcosaa = acos(cosaa)*2;                  //夹角
    double Shu = fcosaa*r1*r1/2;                    // 扇形面积
    double Ssan = r1*r1*sin(fcosaa)/2;              // 三角形面积
    return (Shu - Ssan);                            //扇形面积减去三角形面
积
}
double c2c_inst_area(point a,double r1,point b,double r2)
{
    if( c2c_ainb(a,r1,b,r2) ) return pi*r1*r1;  // a 内含于 b
    if( c2c_ainb(b,r2,a,r1) ) return pi*r2*r2;  // b 内含于 a
    if( !c2c_inst(a,r1,b,r2) )   return 0;
    double l = disp2p(a,b);             // 如果单纯求相交面积，需判断相交，
从这儿到下面这段
    return gongxing_area(r1,r2,l) + gongxing_area(r2,r1,l);
}
```

9.8.3 判断圆和矩形是否相交

```
//判断圆和矩形是否相交 ,包括只交一个点
bool c2r_inst(point a,double r,point p[])
{
    bool flag = false;
    for(int i=0; i<4; i++)//如果严格相交，不要等号
        if( dyd(disp2p(a,p[i]),r) )
            flag = true;
    if( !flag ) return false;
    flag = false;
    for(int i=0; i<4; i++)
        if( xyd(disp2seg(a,p[i],p[(i+1)%4]),r) )
            flag = true;
    if( !flag ) return false;
    return true;
}
```

9.8.4 最小覆盖圆

```
/*******************************************************************
                        n 个点的最小覆盖圆 O(n)算法
*******************************************************************/
point circumcenter(point a,point b,point c)
void min_cover_circle(point p[],int n,point &c,double &r)
{
    random_shuffle(p,p+n);// #include <algorithm>
    c = p[0]; r = 0;
    for(int i=1; i<n; i++)
        if( dy(disp2p(p[i],c),r) )
        {
            c = p[i]; r = 0;
            for(int k=0; k<i; k++)
                if( dy(disp2p(p[k],c),r) )
                {
                    c.x = (p[i].x + p[k].x)/2;
                    c.y = (p[i].y + p[k].y)/2;
                    r = disp2p(p[k],c);
                    for(int j=0; j<k; j++)
                        if( dy(disp2p(p[j],c),r) )
                        {                          // 求外接圆圆心，三点必不共
线
                            c = circumcenter(p[i],p[k],p[j]);
                            r = disp2p(p[i],c);
                        }
                }
        }
}
```

9.8.5 扇形重心距圆心距离

```
// 扇形重心距圆心距离
//Xc = 2*R*sinA/3/A
//A 为圆心角的一半
double dis_z2c(double r, double angle)
{
    return 2 * r * sin(angle) / 3 / angle;
}
```

9.9 多边形

9.9.1 判断两个矩形是否相交

```
//判断两个矩形是否相交（即有公共面积），用左下角和右上角点表示矩形
bool r2r_inst(point a1,point a2,point b1,point b2)
{
    if( b1.x >= a2.x || b2.x <= a1.x || b1.y >= a2.y || b2.y <= a1.y )
        return false;
    return true;
}
```

9.9.2 多边形面积

```
// 计算多边形的面积，可以根据面积判断是哪种方向
//按照我写的这个，顺时针的话，没有加绝对值的面积为负
double area_polygon(point p[],int n)
{
    if( n < 3 ) return 0.0;
    double s = 0.0;
    for(int i=0; i<n; i++)
        s += p[(i+1)%n].y * p[i].x - p[(i+1)%n].x * p[i].y;
    return fabs(s)/2.0;
}
//改变时针方向
void change_wise(point p[],int n)
{
    for(int i=0; i<n/2; i++)
        swap(p[i],p[n-i-1]);
}
```

9.9.3 多边形的重心

```
//计算多边形的重心（凸凹均可）
double crossProduct(point a,point b,point c)//向量 ac 在 ab 的方向
point bary_center(point p[],int n)
{
    point ans,t;
    double area = 0.0,t2;
    ans.x = 0.0; ans.y = 0.0;
    for(int i=1; i<n-1; i++)
    {
        t2 = crossProduct(p[i],p[0],p[i+1])/2.0;
        ans.x += (p[0].x + p[i].x + p[i+1].x)*t2;
        ans.y += (p[0].y + p[i].y + p[i+1].y)*t2;
        area += t2;
    }
    ans.x /= (3*area);
    ans.y /= (3*area);
    return ans;
}
```

9.10 凸包相关

9.10.1 二维凸包

```
//判断一个多边形是否是凸包，判断线旋转方向即可
bool is_convexhull(point p[],int n)
```

```
{
    for(int i=0; i<n; i++)
        if( xy( crossProduct(p[i],p[(i+1)%n],p[(i+2)%n]) *
            crossProduct(p[(i+1)%n],p[(i+2)%n],p[(i+3)%n]),0.0) )
            return false;
    return true;
}

//凸包无序排序 s 为内点
point s;
bool cmp(point a,point b)
{
    double l1 = crossProduct(s,c[0],a);
    double l2 = crossProduct(s,c[0],b);
    if( dyd(l1,0.0) && dyd(l2,0.0) )
        return xy(crossProduct(s,a,b),0.0);
    if( dyd(l1,0.0) && xyd(l2,0.0) )
        return 0;
    if( xyd(l1,0.0) && xyd(l2,0.0) )
        return xy(crossProduct(s,a,b),0.0);
    return 1;
}
void sort_chull()
{
    s.x = (c[0].x + 2 * c[1].x + c[2].x)/4;  // S 求法：线的中点，俩中点再求
中点
    s.y = (c[0].y + 2 * c[1].y + c[2].y)/4;
    int tmp = 0;
    for(int i=1; i<n; i++)
        if(     xy(c[i].x,c[tmp].x)     ||     dd(c[i].x,c[tmp].x)     &&
xy(c[i].y,c[tmp].y) )
            tmp = i;
    swap(c[tmp],c[0]);
    sort(c+1,c+n,cmp);
}
/*********************************************************************
                                    求凸包
*********************************************************************/
double crossProduct(point a,point b,point c)
double disp2p(point a,point b)
bool cmp(point a,point b)  // 排序
{
    double len = crossProduct(c[0],a,b);
    if( dd(len,0.0) )
        return xy(disp2p(c[0],a),disp2p(c[0],b));
    return xy(len,0.0);
}
int stk[MAX];
int top;
//形成凸包条件：输入点个数大于等于三且至少有三个点不共线
//纯净凸包（即 stk 中没有三点共线）
void Graham(int n)
{
    int tmp = 0;
    for(int i=1; i<n; i++)
        if(     xy(c[i].x,c[tmp].x)     ||     dd(c[i].x,c[tmp].x)     &&
xy(c[i].y,c[tmp].y) )
            tmp = i;
    swap(c[0],c[tmp]);
    sort(c+1,c+n,cmp);
```

```
    stk[0] = 0; stk[1] = 1;
    top = 1;
    for(int i=2; i<n; i++)
    {
        while( xyd( crossProduct(c[stk[top]],c[stk[top-1]],c[i]), 0.0 ) &&
top >= 1 )
            top--;
        stk[++top] = i;
    }
}
void Graham(int n)
{
    int tmp = 0;
    for(int i=1; i<n; i++)
        if(     xy(c[i].x,c[tmp].x)     ||     dd(c[i].x,c[tmp].x)     &&
xy(c[i].y,c[tmp].y) )
            tmp = i;
    swap(c[0],c[tmp]);
    sort(c+1,c+n,cmp);
    stk[0] = c[0]; stk[1] = c[1];
    top = 1;
    for(int i=2; i<n; i++)
    {
        while( xyd( crossProduct(stk[top],stk[top-1],c[i]), 0.0 ) && top >=
1 )
            top--;
        stk[++top] = c[i];
    }
}
```

9.10.2 三维凸包

```
//三维凸包模板,可以求三维凸包表面积,体积,表面多边形数和表面三角形数
#define eps 1e-7
#define MAXV 310

//三维点
struct pt{
    double x, y, z;
    pt(){}
    pt(double _x, double _y, double _z): x(_x), y(_y), z(_z){}
    pt operator - (const pt p1){return pt(x - p1.x, y - p1.y, z - p1.z);}
    pt operator * (pt p){return pt(y*p.z-z*p.y, z*p.x-x*p.z, x*p.y-y*p.x);}
//叉乘
    double    operator    ^    (pt    p){return    x*p.x+y*p.y+z*p.z;}
//点乘
};

struct _3DCH{
    struct fac{
        int a, b, c;    //表示凸包一个面上三个点的编号
        bool ok;        //表示该面是否属于最终凸包中的面
    };

    int n;    //初始点数
    pt P[MAXV];    //初始点

    int cnt;    //凸包表面的三角形数
    fac F[MAXV*8]; //凸包表面的三角形
```

```
    int to[MAXV][MAXV];

    double vlen(pt a){return sqrt(a.x*a.x+a.y*a.y+a.z*a.z);}    //向量长度
    double area(pt a, pt b, pt c){return vlen((b-a)*(c-a));}    //三角形面
积*2
    double volume(pt a, pt b, pt c, pt d){return (b-a)*(c-a)^(d-a);}    //
四面体有向体积*6

    //正：点在面同向
    double ptof(pt &p, fac &f){
        pt m = P[f.b]-P[f.a], n = P[f.c]-P[f.a], t = p-P[f.a];
        return (m * n) ^ t;
    }

    void deal(int p, int a, int b){
        int f = to[a][b];
        fac add;
        if (F[f].ok){
            if (ptof(P[p], F[f]) > eps)
                dfs(p, f);
            else{
                add.a = b, add.b = a, add.c = p, add.ok = 1;
                to[p][b] = to[a][p] = to[b][a] = cnt;
                F[cnt++] = add;
            }
        }
    }

    void dfs(int p, int cur){
        F[cur].ok = 0;
        deal(p, F[cur].b, F[cur].a);
        deal(p, F[cur].c, F[cur].b);
        deal(p, F[cur].a, F[cur].c);
    }

    bool same(int s, int t){
        pt &a = P[F[s].a], &b = P[F[s].b], &c = P[F[s].c];
        return fabs(volume(a, b, c, P[F[t].a])) < eps && fabs(volume(a, b,
c, P[F[t].b])) < eps && fabs(volume(a, b, c, P[F[t].c])) < eps;
    }

    //构建三维凸包
    void construct(){
        cnt = 0;
        if (n < 4)
            return;

        /*********此段是为了保证前四个点不公面，若已保证，可去掉********/
        bool sb = 1;
        //使前两点不公点
        for (int i = 1; i < n; i++){
            if (vlen(P[0] - P[i]) > eps){
                swap(P[1], P[i]);
                sb = 0;
                break;
            }
        }
        if (sb)return;

        sb = 1;
```

```
    //使前三点不公线
    for (int i = 2; i < n; i++){
        if (vlen((P[0] - P[1]) * (P[1] - P[i])) > eps){
            swap(P[2], P[i]);
            sb = 0;
            break;
        }
    }
    if (sb)return;

    sb = 1;
    //使前四点不共面
    for (int i = 3; i < n; i++){
        if (fabs((P[0] - P[1]) * (P[1] - P[2]) ^ (P[0] - P[i])) > eps){
            swap(P[3], P[i]);
            sb = 0;
            break;
        }
    }
    if (sb)return;
    /*********此段是为了保证前四个点不公面********/


    fac add;
    for (int i = 0; i < 4; i++){
        add.a = (i+1)%4, add.b = (i+2)%4, add.c = (i+3)%4, add.ok = 1;
        if (ptof(P[i], add) > 0)
            swap(add.b, add.c);
        to[add.a][add.b] = to[add.b][add.c] = to[add.c][add.a] = cnt;
        F[cnt++] = add;
    }

    for (int i = 4; i < n; i++){
        for (int j = 0; j < cnt; j++){
            if (F[j].ok && ptof(P[i], F[j]) > eps){
                dfs(i, j);
                break;
            }
        }
    }
    int tmp = cnt;
    cnt = 0;
    for (int i = 0; i < tmp; i++){
        if (F[i].ok){
            F[cnt++] = F[i];
        }
    }
}

//表面积
double area(){
    double ret = 0.0;
    for (int i = 0; i < cnt; i++){
        ret += area(P[F[i].a], P[F[i].b], P[F[i].c]);
    }
    return ret / 2.0;
}

//体积
double volume(){
```

```
        pt O(0, 0, 0);
        double ret = 0.0;
        for (int i = 0; i < cnt; i++){
            ret += volume(O, P[F[i].a], P[F[i].b], P[F[i].c]);
        }
        return fabs(ret / 6.0);
    }

    //表面三角形数
    int facetCnt_tri(){
        return cnt;
    }

    //表面多边形数
    int facetCnt(){
        int ans = 0;
        for (int i = 0; i < cnt; i++){
            bool nb = 1;
            for (int j = 0; j < i; j++){
                if (same(i, j)){
                    nb = 0;
                    break;
                }
            }
            ans += nb;
        }
        return ans;
    }
};

_3DCH hull;    //内有大数组，不易放在函数内

int main()
{
    while (~scanf("%d", &hull.n)){
        for (int i = 0; i < hull.n; i++)
            scanf("%lf%lf%lf", &hull.P[i].x, &hull.P[i].y, &hull.P[i].z);
        hull.construct();
        printf("%d\n", hull.facetCnt());
    }
    return 0;
}
```

9.11 旋转卡壳

```
//旋转卡壳 先求凸包(纯净)，得出 stk 数组，n 为 top+1

// 返回最远点对距离
double crossProduct(Point a,Point b,Point c)//向量 ac 在 ab 的方向
double RC_maxdisp2p_inhull(int stk[],int n)
{
    stk[n] = stk[0];
    int q = 1;
    double ans = 0.0;
    for(int i=0; i<n; i++)
    {
        while( xy(fabs(crossProduct(c[stk[i]],c[stk[i+1]],c[stk[q]])),

    fabs(crossProduct(c[stk[i]],c[stk[i+1]],c[stk[(q+1)%n]]))) )
            q = (q+1)%n;
```

```
        ans = max(ans,disp2p(c[stk[i]],c[stk[q]]));
    }
    return ans;
}

//旋转卡壳求两个凸包之间最短距离
double crossProduct(point a,point b,point c)
double disp2p(point a,point b)
bool cmp(point a,point b)  // 凸包排序
void Graham(point c[],point stk[],int &top,int n) // 凸包
double disp2seg(point p,point l1,point l2)// 点到线段的最短距离
double rota_angle(point a1,point a2,point b1,point b2) //返回 b1b2 在 a1a2
的方向
double RC_mindish2h(point p[],point q[],int np,int nq) // 返回最短距离
{
    int sp = 0,sq = 0;
    for(int i=1; i<np; i++)
        if(     xy(p[i].y,p[sp].y)      ||      dd(p[sp].y,p[i].y)      &&
xy(p[i].x,p[sp].x) )
            sp = i;        //求得第一个凸包最左下角的点

    for(int i=1; i<nq; i++)
        if(     dy(q[i].y,q[sq].y)      ||      dd(q[sq].y,q[i].y)      &&
dy(q[i].x,q[sq].x) )
            sq = i;        //求得第二个凸包最右上角的点

    int tp = sp,tq = sq;
    double ans = disp2p(p[sp],q[sq]);
    do
    {
        double len = rota_angle(p[sp],p[(sp+1)%np],q[(sq+1)%nq],q[sq]);
        if( dd(len,0.0) )                // 卡壳正好和俩凸包的边重合
        {
            ans = min(ans,disp2seg(p[sp],q[sq],q[(sq+1)%nq]));
            ans = min(ans,disp2seg(p[(sp+1)%np],q[sq],q[(sq+1)%nq]));
            ans = min(ans,disp2seg(q[sq],p[sp],p[(sp+1)%np]));
            ans = min(ans,disp2seg(q[(sq+1)%nq],p[sp],p[(sp+1)%np]));
            sp++; sp %= np; sq++; sq %= nq;
        }
        else
            if( xy(len,0.0) )            // 卡壳和第一个凸包的边重合
            {
                ans = min(ans,disp2seg(q[sq],p[sp],p[(sp+1)%np]));
                sp++; sp %= np;
            }
            else
            {
                ans = min(ans,disp2seg(p[sp],q[sq],q[(sq+1)%nq]));// 卡壳
和第二个凸包的边重合
                sq++; sq %= nq;
            }
    }while( !(tp == sp && tq == sq) );
    return ans;
}

//求凸包最小面积外接矩形
double crossProduct(point a,point b,point c)//向量 ac 在 ab 的方向
double disp2p(point a,point b)
bool cmp(point a,point b)  // 凸包排序
double disp2seg(point a,point l1,point l2)
```

```
point foot_line(point a,point l1,point l2)  // 求一个点，使得 ab 垂直于 l1l2
double rota_angle(point a1,point a2,point b1,point b2) //判断向量 a1a2 与 b1b2 的位置
double Graham(int n)
double RC_minareaRectangle(point p[],int n) //返回面积
{
    int r[4];      // 0 == ymin, 1 == xmin, 2 == ymax ,3 == xmax;
    memset(r,0,sizeof(r));
    for(int i=0; i<n; i++)
    {
        if( xy(p[i].y,p[r[0]].y) )   r[0] = i;
        if( xy(p[i].x,p[r[1]].x) )   r[1] = i;
        if( dy(p[i].y,p[r[2]].y) )   r[2] = i;
        if( dy(p[i].x,p[r[3]].x) )   r[3] = i;
    }
    int tp = r[0];
    double area = inf;
    do
    {
        point t = foot_line(p[r[0]],p[r[0]],p[(r[0]+1)%n]);
        while( dy(rota_angle(t,p[r[0]],p[r[1]],p[(r[1]+1)%n]),0.0) )
            r[1]++, r[1] %= n;

        while( dy(rota_angle(p[r[0]],t,p[r[3]],p[(r[3]+1)%n]),0.0) )
            r[3]++, r[3] %= n;

        while( dy(disp2seg(p[(r[2]+1)%n],p[r[0]],p[(r[0]+1)%n]),
                disp2seg(p[r[2]],p[r[0]],p[(r[0]+1)%n])) )
            r[2]++, r[2] %= n;

        double a = disp2seg(p[r[2]],p[r[0]],p[(r[0]+1)%n]);
        t = foot_line(p[r[3]],p[r[0]],p[(r[0]+1)%n]);
        double b = disp2seg(p[r[1]],p[r[3]],t);
        area = min( area, a*b );
        r[0]++; r[0] %= n;
    }while( r[0] != tp );
    return area;
}
//卡壳求最大三角形面积，凸包预处理
double RC(point *s,int n) // 点集是凸包
{
    int p,q,r;
    p = 0; q = 1; r = 2;
    double area = area_triangle(s[p],s[q],s[r]);
    for(p=0; p<n; p++)
    {
        q = (p+1)%n; r = (p+2)%n;
        area = max(area,area_triangle(s[p],s[q],s[r]));
        while( xy(fabs(crossProduct(s[p],s[q],s[r])),
                fabs(crossProduct(s[p],s[q],s[(r+1)%n]))) && r != p )
        {
            area = max(area,area_triangle(s[p],s[q],s[(r+1)%n]));
            r = (r+1)%n;
        }
        if( r == p ) continue;
        int rr = (r+1)%n;
        while( q != rr && r != p )
        {
            area = area = max(area,area_triangle(s[p],s[q],s[r]));
            while( xy(fabs(crossProduct(s[p],s[q],s[r])),
```

```
                fabs(crossProduct(s[p],s[q],s[(r+1)%n]))) && r != p )
                    r = (r+1)%n;
            q = (q+1)%n;
        }
    }
    return area;
}
```

9.12 半平面交

```
// 线段 ab 向左方向推进 h 后得到 cd
void changepoint(point a,point b,point &c,point &d,double h)
{
    double len = disp2p(a,b);
    double dx = h / len * ( a.y - b.y );
    double dy = h / len * (-a.x + b.x );
    c.x = a.x + dx; c.y = a.y + dy;
    d.x = b.x + dx; d.y = b.y + dy;
}

//半平面交 N^2 算法，注意初始化有界平面（初始化成矩形，p[n]=p[0]）
//每增加一条直线 ab 切割  调用这个函数。最终切点存在 s 中，长度为 len。
void cut_hp(point a,point b,point *s,int &len)
{
    int tc = 0;
    point tp[MAX];
    for(int i=0; i<=len; i++)
        tp[i] = s[i];
    for(int i=0; i<len; i++)
    {
        if( xyd(crossProduct(a,b,tp[i]),0.0) )//右侧区域的话是 dyd
            s[tc++] = tp[i];
        if( xy(crossProduct(a,b,tp[i])*
                crossProduct(a,b,tp[i+1]),0.0) )
            s[tc++] = l2l_inst_p(a,b,tp[i],tp[i+1]);
    }
    s[tc] = s[0];
    len = tc;
}

// 半平面交 N*LOGN 注意初始化有界平面
// 半平面返回点集 s，点集点的个数 len
//ln 是 n 个半平面，用 line 的两个端点表示，一律考虑线段左边是半平面
//如果 ln 中有线段右边是半平面的，可以交换线段两点坐标
point l2l_inst_p(line l1,line l2)
bool parallel(line u,line v)
bool equal_ang(line a,line b)// 第一次 unique 的比较函数
{
    return dd(a.ang,b.ang);
}
bool cmphp(line a,line b) // 排序的比较函数
{
    if( dd(a.ang,b.ang) ) return xy(crossProduct(b.a,b.b,a.a),0.0);
    return xy(a.ang,b.ang);
}
bool equal_p(point a,point b)//第二次 unique 的比较函数
{
    return dd(a.x,b.x) && dd(a.y,b.y);
}
void makeline_hp(double x1,double y1,double x2,double y2,line &l)
```

```
{
    l.a.x = x1; l.a.y = y1; l.b.x = x2; l.b.y = y2;
    l.ang = atan2(y2 - y1,x2 - x1);
}
void makeline_hp(point a,point b,line &l) // 线段(向量 ab)左侧侧区域有效
{
    l.a = a; l.b = b;
    l.ang = atan2(b.y - a.y,b.x - a.x); // 如果是右侧区域，改成 a.y -
b.y,a.x - b.x
}
void inst_hp_nlogn(line *ln,int n,point *s,int &len)
{
    len = 0;
    sort(ln,ln+n,cmphp);
    n = unique(ln,ln+n,equal_ang) - ln;
    int bot = 0,top = 1;
    deq[0] = ln[0]; deq[1] = ln[1];
    for(int i=2; i<n; i++)
    {
        if(                parallel(deq[top],deq[top-1])                ||
parallel(deq[bot],deq[bot+1]) )
            return ;
        while( bot < top && dy(crossProduct(ln[i].a,ln[i].b,
            l2l_inst_p(deq[top],deq[top-1])),0.0) )
            top--;
        while( bot < top && dy(crossProduct(ln[i].a,ln[i].b,
            l2l_inst_p(deq[bot],deq[bot+1])),0.0) )
            bot++;
        deq[++top] = ln[i];
    }
    while( bot < top && dy(crossProduct(deq[bot].a,deq[bot].b,
        l2l_inst_p(deq[top],deq[top-1])),0.0) ) top--;
    while( bot < top && dy(crossProduct(deq[top].a,deq[top].b,
        l2l_inst_p(deq[bot],deq[bot+1])),0.0) ) bot++;
    if( top <= bot + 1 ) return ;

    for(int i=bot; i<top; i++)
        s[len++] = l2l_inst_p(deq[i],deq[i+1]);
    if( bot < top + 1 ) s[len++] = l2l_inst_p(deq[bot],deq[top]);
    len = unique(s,s+len,equal_p) - s;
}
```

9.13 整点相关

```
//计算多边形边的整点（包括顶点）
int gcd(int n,int m)
int intp_insegment(point a, point b)
int intp_edge(point p[],int n)
{
    int ans = n;
    for(int i=0; i<n; i++)
        ans += intp_insegment(p[i], p[(i+1)%n]);
    return ans;
}
// 求多边形内的整点个数，多边形顶点都是整点&& dx dy 不能为 0
double area_polygon(point p[],int n)
int intp_inpolygon(point p[],int n)
{
    double area = area_polygon(p,n);
    int pinedge = intp_edge(p,n);
```

```
    return (int)(area) - pinedge/2 + 1;
}
//计算在线段上的整点个数 （还是这个比较好）
int intp_insegment(point a, point b)
{
    int aa = abs(b.y - a.y), bb = abs(b.x - a.x);
    if(aa == 0 && bb == 0)    return 0;
    if(aa == 0)    return bb - 1;
    if(bb == 0)    return aa - 1;
    return gcd(aa, bb) - 1;
}
```

9.14 球面相关

```
//求大圆角度，返回大圆的圆心角，输入为弧度
double angle_3d(double lng1, double lat1, double lng2, double lat2)
{           //经度，纬度，经度，纬度
    return      acos(cos(lat1)*cos(lat2)*cos(lng1      -      lng2)      +
sin(lat1)*sin(lat2));
}
//求大圆劣弧长度，输入为弧度
double dis_3d(double lng1, double lat1, double lng2, double lat2)
{           //经度，纬度，经度，纬度
    double dlon = lng2 - lng1;
    double dlat = lat2 - lat1;
   double a = pow((sin(dlat/2)),2) + cos(lat1) * cos(lat2) * pow(sin(dlon/2),
2);
    double c = 2 * atan2(sqrt(a), sqrt(1-a));
    double d = r * c;
    return d;
}
//大地坐标转空间坐标系
//纬度， 经度 ，单位坐标， 输入为角度
    Point(double la, double lo) :
        x(cos(lo*PI/180) * cos(la*PI/180)),
        y(sin(lo*PI/180) * cos(la*PI/180)),
        z(sin(la*PI/180)) {}
```

9.15 模拟退火求多边形费马点

```
const int MAX = 1010;
const double inf = 1e30;
const double eps = 1e-8;
const double pi = acos(-1.0);
const int N = 15; // 设定随机点的个数
const int L = 40; // 设定随机方向次数
bool dy(double x,double y)    {    return x > y + eps;}    // x > y
bool xy(double x,double y)    {    return x < y - eps;}    // x < y
bool dyd(double x,double y)    {     return x > y - eps;}    // x >= y
bool xyd(double x,double y)    {    return x < y + eps;}     // x <= y
bool dd(double x,double y)     {    return fabs( x - y ) < eps;}  // x ==
y
struct point { double x,y;
    point (double x,double y):x(x),y(y){}
    point ():x(0),y(0){}
};
double disp2p(point a,point b) //  a b 两点之间的距离
{
    return sqrt(( a.x - b.x ) * ( a.x - b.x ) + ( a.y - b.y ) * ( a.y - b.y ));
```

```
}
point p[MAX];
point rp[MAX];
double len[MAX];
double min_dis(point a,point *p,int n)
{
    double min = inf;
    for(int i=0; i<n; i++)
    {
        double len = disp2p(a,p[i]);
        if( xy(len,min) )
            min = len;
    }
    return min;
}
bool check(point a,double x,double y)
{
    return dyd(a.x,0.0) && dyd(a.y,0) && xyd(a.x,x) && xyd(a.y,y);
}
point Rand(double x,double y)
{
    point c;
    c.x = ( rand()%1000 + 1 ) / 1000.0 * x;
    c.y = ( rand()%1000 + 1 ) / 1000.0 * y;
    return c;
}
int main()
{
    int n,m,ncases;
    double x,y;
    srand(time(NULL));    // time.h!!
    scanf("%d",&ncases);

    while( ncases-- )
    {
        scanf("%lf%lf%d",&x,&y,&n);
        for(int i=0; i<n; i++)
            scanf("%lf%lf",&p[i].x,&p[i].y);
        for(int i=0; i<N; i++)
        {
            rp[i] = Rand(x,y);
            len[i] = min_dis(rp[i],p,n);
        }
        point st;
        double step = max(x,y)/2;
        while( step > 0.001 )
        {
            for(int k=0; k<N; k++)
            {
                st = rp[k];
                for(int i=0; i<L; i++)
                {
                    double ang = (rand()%1000+1)/1000.0*10*pi;
                    double xx = st.x + step*cos(ang);
                    double yy = st.y + step*sin(ang);
                    point t = point(xx,yy);
                    if( !check(t,x,y) ) continue;
                    double dis = min_dis(t,p,n);
                    if( dy(dis,len[k]) )
                    {
```

```
                    rp[k] = t;
                    len[k] = dis;
                }
            }
        }
        step *= 0.8;
    }
    int ind = 0;
    for(int i=1; i<N; i++)
        if( len[i] > len[ind] )
            ind = i;
    printf("The              safest              point              is
(%.1lf, %.1lf).\n",rp[ind].x,rp[ind].y);
    }

return 0;
}
```

10 DP

10.1 各种背包

```
////////////////1111111111
#define zero_one_pack(d, cost, weight, limit, i)              \
do {                                                          \
    for (i = limit; i >= cost; --i)                    \
        if (d[i] < d[i-cost] + weight)                 \
            d[i] = d[i-cost] + weight;              \
} while (0)

#define complete_pack(d, cost, weight, limit, i)              \
do {                                                          \
    for (i = cost; i <= limit; ++i)                    \
        if (d[i] < d[i-cost] + weight)                 \
            d[i] = d[i-cost] + weight;              \
} while (0)

#define multiple_pack(d, cost, weight, amount, limit, i, k)          \
do {                                                                 \
    if (cost * amount >= limit)                                \
        complete_pack(d, cost, weight, limit, i);          \
    else {                                                   \
        for (k = 1; k <= amount; k <<= 1) {             \
            zero_one_pack(d, k*cost, k*weight, limit, i);      \
            amount -= k;                                \
        }                                           \
        zero_one_pack(d, amount*cost, amount*weight, limit, i);       \
    }                                               \
} while (0)

struct node {
    int cost;
    int weight;
    int amount;
};

#define LIMIT 20
```

```
#define N 3

int
main(void)
{
    struct node obj[N];
    int d[LIMIT + 1];
    int i, j, k;

    printf("LIMIT: %d\n", LIMIT);

    srand(time(NULL));
    for (i = 0; i < N; ++i) {
        obj[i].cost = rand() % 10 + 1;
        obj[i].weight = rand() % 10 + 1;
        obj[i].amount = rand() % 3 + 1;
        printf("object %d: cost %d  weight %d  amount %d\n",
               i, obj[i].cost, obj[i].weight, obj[i].amount);
    }

    memset(d, 0, sizeof(d));
    for (i = 0; i < N; ++i)
        zero_one_pack(d, obj[i].cost, obj[i].weight, LIMIT, j);
    printf("zero one pack, max is %d\n", d[LIMIT]);

    memset(d, 0, sizeof(d));
    for (i = 0; i < N; ++i)
        complete_pack(d, obj[i].cost, obj[i].weight, LIMIT, j);
    printf("complete pack, max is %d\n", d[LIMIT]);

    memset(d, 0, sizeof(d));
    for (i = 0; i < N; ++i)
        multiple_pack(d, obj[i].cost, obj[i].weight, obj[i].amount, LIMIT,
j, k);
    printf("multiple pack, max is %d\n", d[LIMIT]);

    return 0;
}
//////////////////2222222222222
0-1 背包
//m 为容量，n 为东西个数
memset(bag,0,sizeof(bag));
for(int i=1; i<=n; i++)
    for(int k=m; k>=w[i]; k--)
        if( bag[k-w[i]]+ v[i] > bag[k] )
            bag[k] = bag[k-w[i]]+ v[i];
cout << bag[m] << endl;

完全背包
for(int i=1; i<=n; i++)
    for(int k=w[i]; k<=c; k++)
        {
            money[k] = max( money[k], money[k-w[i]]+w[i] );
        }
cout << money[c] << endl;

多重背包
count = 1;
memset(hp,0,sizeof(hp));
for(int i=1; i<=n; i++)
```

```
    cin >> mp[i] >> hurt[i] >> num[i];
// mp 为每个东西体积，hurt 为价值，num 为数目
for(int i=1; i<=n; i++)
{
    int sum = 0;
    for(int k=0; ; k++)
    {
        int x = (int)pow(2,k);   // <math.h>
        if( sum + x > num[i] )
            break;
        sum += x;
        h[count] = x*hurt[i];
        mmp[count] = x*mp[i];
        count++;
    }
    int x = num[i] - sum;
    if( x == 0 )
        continue;
    h[count] = x*hurt[i];
    mmp[count] = x*mp[i];
    count++;
}
for(int i=1; i<count; i++)
    for(int k=m; k>=mmp[i]; k--)
        if( hp[k] < hp[k-mmp[i]] + h[i] )
            hp[k] = hp[k-mmp[i]] + h[i];
cout << hp[m] << endl;
```

10.2 最长不下降子序列

```
int main(void)
{
    int n,geshu,i,j,nTmp,s=0;
    int a[1005];
    int flag[1005];
    scanf("%d",&n);
    while(n--)
    {
        scanf("%d",&geshu);
        for( i = 0 ; i < geshu ; i++ )
            scanf("%d",&a[i]);

        flag[0] = 1;
        for( i = 1 ; i < geshu ; i++ )//以下是关键代码
        {
            nTmp = 0;
            for( j = 0 ; j < i ; j++ )
                if( flag[j] > nTmp && a[i] >= a[j])//修改这里的符号
                    nTmp = flag[j];
            flag[i] = nTmp + 1;
        }

        nTmp = max_element(flag,flag+geshu);          //判断最长的子序列

        printf(s++?"\n%d\n":"%d\n",nTmp);
    }

    return 0;
}
//////22222222
```

最长上升子序列

```
for(int i=1; i<=n; i++)
    cin >> los[i];
max[1] = 1;
for(int i=2; i<=n; i++)
{
    int maxl = 0;
    for(int k=1; k<i; k++)
        if( los[i] > los[k] )
        {
            if( max[k] > maxl)
                maxl = max[k];
        }
    max[i] = maxl+1;
}
int outputmax = 0;
for(int i=1; i<=n; i++)
    if( max[i] > outputmax )
        outputmax = max[i];
```

10.3 最长公共子序列

```
//zoj 1953 最长公共子序列 带路径记录。。。before 数组为记录的前驱
//最后求出的 c 数组即为最长公共子序列（只是其中一种可能）
int main()
{
    char a[110],b[110],c[110],temp;
    int dp[110][110],before[110][110][2];
    int i,j,t,count;
    int lena,lenb;
    while( cin >> a+1 >> b+1 )
    {
        lena = strlen(a+1);
        lenb = strlen(b+1);
        memset( dp,0,sizeof(dp) );
        for( i = 1; i <= lena; i++ )
            for( j = 1; j <= lenb; j++ )
                if( a[i] == b[j] )
                {
                    dp[i][j] = dp[i-1][j-1] + 1;
                    before[i][j][0] = i - 1;
                    before[i][j][1] = j - 1;
                }
                else
                {
                    if( dp[i-1][j] > dp[i][j-1] ){
                        dp[i][j] = dp[i-1][j];
                        before[i][j][0] = i - 1;
                        before[i][j][1] = j;
                    }
                    else{
                        dp[i][j] = dp[i][j-1];
                        before[i][j][0] = i;
                        before[i][j][1] = j - 1;
                    }

                }
        i=lena;j=lenb;
        count = 0;
        while( i && j )
```

```
        {
            if( a[i] == b[j] )
                c[count++] = a[i];

            t = before[i][j][0];
            j = before[i][j][1];
            i = t;
        }
        c[count] = '\0';
        for( i = 0,count--; i < count; i++,count-- )
            temp=c[i],c[i]=c[count],c[count]=temp;
        for( i = j = 1,t = 0; c[t]!='\0' ;)
        {
            if( a[i] == c[t] && b[j] == c[t] )
                cout<<c[t],t++,i++,j++;
            else if( a[i] == c[t] )
                cout<<b[j++];
            else if( b[j] == c[t] )
                cout<<a[i++];
            else
                cout<<a[i++]<<b[j++];

        }
        cout << a+i << b+j << endl;
    }
    return 0;
}
//党姐的太长了 ><><><><
```

最长公共子序列

```
int maxlen[MAXLEN][MAXLEN];
int main(void)
{
    char str[MAXLEN],line[MAXLEN];
    int lens,lenl;
    while(cin >> str)
    {
        memset(maxlen,0,sizeof(maxlen));
        cin >> line;
        lens = strlen(str);
        lenl = strlen(line);
        for(int i=0; i<lens; i++)
            for(int k=0; k<lenl; k++)
                if( str[i] == line[k] )
                    maxlen[i+1][k+1] = maxlen[i][k] + 1;
                else
                    if( maxlen[i+1][k] > maxlen[i][k+1] )
                        maxlen[i+1][k+1] = maxlen[i+1][k];
                    else
                        maxlen[i+1][k+1] = maxlen[i][k+1];
        cout << maxlen[lens][lenl] << endl;
    }
return 0;
}
```

10.4 最大子段和

最大子串和

```
int Maxsum(int n, int *a)
{
```

```
    int sum = 0,b = 0;
    for(int i=0; i<n; i++)
    {
        if( b > 0 )
            b += a[i];
        else
            b = a[i];
        if( b > sum )
            sum = b;
    }
    return sum;
}
```

10.5 最大子矩阵和

```
int DP(int a[],int n)
{
    int i,f[101];
    int max = -200000000;
    for(i = 2 , f[1] = a[1] ; i <= n ; i++ )
    {
        if (f[i - 1] > 0)
            f[i] = f[i - 1] + a[i];
        else
            f[i] = a[i];

        if (f[i] > max)
            max = f[i];
    }

    return max;
}

int main(void)
{
    int n,i,he,j;
    int a[105][105],k;
    int sum[105],max;
    scanf("%d",&n);
    for( i = 1 ; i <= n ; i++ )
        for( j = 1 ; j <= n ; j++ )
            scanf("%d",&a[i][j]);

    max = -200000000;
    for( i = 1 ; i <= n ; i++ )
    {
        memset(sum,0,sizeof(sum));
        for(j = i ; j <= n ; j++ )
        {
            for( k = 1 ; k <= n ; k++ )
                sum[k] += a[j][k];
            he = DP(sum,n);
            if(he>max) max = he;
        }
    }

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

    return 0;
}
```

11 杂 7 杂 8

11.1 简单图判定

```
// 图序列判定，给定图的节点数还有每个点的度数，判断是否能构成简单图
//sum 是度的总和
const int MAX = 1010;
int deg[MAX];
bool is_simpleG(int n, int sum, int *deg)
{
    if( sum % 2 == 1 ) return false;
    sort(deg+1, deg+n+1, greater<int>());
    for(int i=1; i<=n; i++)
    {
        int s = 0;
        for(int k=1; k<=i; k++)
            s += deg[k];
        int s2 = 0;
        for(int k=i+1; k<=n; k++)
            s2 += min(i, deg[k]);
        if( s > i*(i-1) + s2 )
            return false;
    }
    return true;
}
```

11.2 KMP

```
#define MAX 1001
using namespace std;

int next[MAX];
char s[MAX],t[MAX];

void get_next(char *s)
{
    int len = strlen(s);
    int i = 0,j = -1;
    next[0] = -1;
    while( i < len )
        if( j == -1 || s[i] == s[j] )
        {
            i++, j++;
            next[i] = s[i] == s[j] ? next[j] : j;
        }
        else
            j = next[j];
}
int KMP(char *s,char *t)
{
    get_next(t);
    int lens = strlen(s);
    int lent = strlen(t);
    int i = 0,j = 0;
    while( i < lens && j < lent )
        if( j == -1 || s[i] == t[j] )
            i++, j++;
```

```
        else
            j = next[j];
    if( j >= lent )
        return i - lent;
    return -1;
}
int main()
{
    int n,pos;
    while( scanf("%s %s",&s,&t)!=EOF )
    {
        pos = KMP(s,t);
        if( pos == -1 )
            printf("Can't match\n");
        else
            printf("%d\n",pos);
    }
return 0;
}
```

11.3 等价表达式

```
int count1,count2;
int optr[100],opnd[100];
int h[7][7] = {{1,1,-1,-1,-1,1,1},{1,1,-1,-1,-1,1,1},{1,1,1,1,-1,1,1},

{1,1,1,1,-1,1,1},{-1,-1,-1,-1,-1,0,-2},{1,1,1,1,-2,1,1},{-1,-1,-1,-1,-1
,-2,0}};
void push1(int x)
{
    opnd[count1++] = x;
}
void push2(int x)
{
    optr[count2++] = x;
}
int empty2()
{
    if(count2 == 0)
        return 1;
    return 0;
}
int pop1()
{
    count1--;
    return opnd[count1];
}
char pop2()
{
    count2--;
    return optr[count2];
}
int computer(int x,int y, char ch)
{
    switch(ch)
    {
        case '+':return x+y;
        case '-':return x-y;
        case '*':return x*y;
        case '/':return x/y;
```

```
    }
}
char gettop()
{
    return optr[count2-1];
}
int number(char ch)
{
    switch(ch)
    {
        case '+':return 0;
        case '-':return 1;
        case '*':return 2;
        case '/':return 3;
        case '(':return 4;
        case ')':return 5;
        case '#':return 6;
    }
}
int record(char a,char b)
{
    return h[number(a)][number(b)];
}
int main(void)
{
    char c,fir[50];

    int x,y,temp,ch,length,i,plus,result1,result2;
    plus = 10;
    gets(fir);
    length = strlen(fir);
    fir[length] = '#';
    fir[length+1] = '\0';
    push2('#');
    i = 0;
           while( fir[i]!='#' || gettop()!='#')
           {
               c = fir[i];
               if( isdigit(c) || isalpha(c) )
               {
                   if(isalpha(c))
                       push1(c-'a'+plus);
                   else
                       push1(c-'0');
                   i++;
               }
               else
               {
                   if(c == '+' || c== '-'||c=='*' || c=='('||c==')'||c=='#')
                   {
                   switch( record(gettop(),c) )
                   {

                       case -1: push2(c); i++; break;
                       case 0 : pop2(); i++; break;
                       case 1 : ch = pop2();
                                y = pop1();
                                x = pop1();
                                push1(computer(x,y,ch));
                                break;
```

```
                }
                }
                else
                    i++;
            }
        }
        result1= opnd[count1-1];
    printf("%d\n",result1);
return 0;
}
```

11.4 字串种类数(copy)

```
#define MAXN 50000
#define MAXM 128          // 字母个数

int head[MAXN], succ[MAXN], succRank[MAXN];
int sa[MAXN], rank[MAXN];
int letter[MAXM];

void da(char* str, int len) {// 倍增算法
    memset(letter, 0, sizeof(letter));
    for (int i = 0; i < len; ++i)
        if (!letter[str[i]])
            letter[str[i]] = 1;
    int total = -1;
    for (int i = 0; i < MAXM; ++i)
        if (letter[i])
            letter[i] = ++total;
    memset(head, 255, sizeof(head));
    for (int i = len - 1; i >= 0; --i) { // 由于 head 头指针的性质，所以要倒序添加，下同
        rank[i] = letter[str[i]];
        succ[i] = head[rank[i]];
        head[rank[i]] = i;
    }
    int j = 0;
    for (int i = 0; i < len; ++i) {
        while (head[j] == -1) ++j;
        sa[i] = head[j];
        head[j] = succ[head[j]];
    }
    // 到此初始化排序完毕
    for (int k = 1; k < len; k <<= 1) {
        // 以下两个 for 对新一轮子串进行排序，由于第二关键字的排序在上一次排序中已完成
        // 故仅对第一关键字进行排序即可
        for (int i = len - 1; i >= 0; --i)      // 倒序添加
            if (sa[i] - k >= 0) {
                succRank[sa[i] - k] = rank[sa[i]];
                succ[sa[i] - k] = head[rank[sa[i] - k]];
                head[rank[sa[i] - k]] = sa[i] - k;
            }
        for (int i = len - 1; i >= len - k; --i) {  // 倒序添加
            succRank[i] = -1;                // 另外添加的尾字符的 rank 为 -1
            succ[i] = head[rank[i]];
            head[rank[i]] = i;
        }
        j = 0; total = -1;
```

```
        int preSuccRank = 0, preRank = 0;
        for (int i = 0; i < len; ++i) {
            while (head[j] == -1) ++j;
            sa[i] = head[j];
            if (i == 0 || preRank != rank[sa[i]] || preSuccRank != succRank[sa[i]]) {
                preRank = rank[sa[i]];
                rank[sa[i]] = ++total;    // 链表中保证不递减，所以可以这么做
            } else {
                preRank = rank[sa[i]];
                rank[sa[i]] = total;
            }
            preSuccRank = succRank[sa[i]];
            head[j] = succ[head[j]];
        }
    }
}

int height[MAXN];

void calcHeight(char* str, int len) {
    int i, j, k = 0;
    for (i = 0; i < len; ++i) {
        if (k != 0)              // h[i] >= h[i - 1] - 1
            --k;
        if (rank[i] == 0)
            continue;
        j = sa[rank[i] - 1];
        // 求 suffix(sa[rank[i] - 1]) 和 suffix(sa[rank[i]]) 的最大公共前缀
        while (str[i + k] == str[j + k])    // C 字符串以 '\0' 结尾
            ++k;
        height[rank[i]] = k;  // h[i] = k;
    }
}

char buf[MAXN + 1];
int len;

int main() {
    int t;
    scanf("%d", &t);
    for (int cas = 1; cas <= t; ++cas) {
        scanf("%s", buf);
        len = strlen(buf);
        da(buf, len);
        calcHeight(buf, len);
        long long ans = len - sa[0];
        for (int i = 1; i < len; ++i)
            ans += len - sa[i] - height[i];
        printf("%lld\n", ans);
    }
    return 0;
}
```

11.5 欧拉回路判断

先判连通

```
//有向图
int isoula()    //判断是否是欧拉路，包括欧拉回路和欧拉道路
{
```

```
    int i;
    int sum = 0,out[26];
    for(i=0; i<26; i++)
        if( ind[i] != outd[i] )
        {
            out[sum] = i;
            sum++;
        }
    if( sum == 0 )
    {
        i = 0;
        while( !used[i] )
            i++;
        beg = i;
        return 1;
    }
    if( sum == 2 )
    {
        int x = out[0],y = out[1];
        if( ind[x] - outd[x] == 1 && outd[y] - ind[y] == 1 )
        {
            beg = y;
            return 1;
        }
        if( ind[y] - outd[y] == 1 && outd[x] - ind[x] == 1 )
        {
            beg = x;
            return 1;
        }
    }
    return 0;
}
```

输出欧拉路

```
stack<int> s;
void output(int u)  // 用栈存输出编号，递归。。。
{
    MAP *head = alp[u];
    while( head != NULL )
    {
        if( head->flag == 0 )
        {
            head->flag = 1;
            output(head->v);
            s.push(head->to);
        }
        head = head->next;
    }
}
```

11.6 大数

```
const int maxn = 2100;
struct bign{
  int len, s[maxn];

  bign() {
    memset(s, 0, sizeof(s));
    len = 1;
  }
```

```
bign(int num) {
  *this = num;
}

bign(const char* num) {
  *this = num;
}

bign operator = (int num) {
  char s[maxn];
  sprintf(s, "%d", num);
  *this = s;
  return *this;
}

bign operator = (const char* num) {
  len = strlen(num);
  for(int i = 0; i < len; i++) s[i] = num[len-i-1] - '0';
  return *this;
}

string str() const {
  string res = "";
  for(int i = 0; i < len; i++) res = (char)(s[i] + '0') + res;
  if(res == "") res = "0";
  return res;
}

bign operator + (const bign& b) const{
  bign c;
  c.len = 0;
  for(int i = 0, g = 0; g || i < max(len, b.len); i++) {
    int x = g;
    if(i < len) x += s[i];
    if(i < b.len) x += b.s[i];
    c.s[c.len++] = x % 10;
    g = x / 10;
  }
  return c;
}

void clean() {
  while(len > 1 && !s[len-1]) len--;
}

bign operator * (const bign& b) {
  bign c; c.len = len + b.len;
  for(int i = 0; i < len; i++)
    for(int j = 0; j < b.len; j++)
      c.s[i+j] += s[i] * b.s[j];
  for(int i = 0; i < c.len-1; i++){
    c.s[i+1] += c.s[i] / 10;
    c.s[i] %= 10;
  }
  c.clean();
  return c;
}

bign operator - (const bign& b) {
  bign c; c.len = 0;
```

```
  for(int i = 0, g = 0; i < len; i++) {
    int x = s[i] - g;
    if(i < b.len) x -= b.s[i];
    if(x >= 0) g = 0;
    else {
      g = 1;
      x += 10;
    }
    c.s[c.len++] = x;
  }
  c.clean();
  return c;
}

bool operator < (const bign& b) const{
  if(len != b.len) return len < b.len;
  for(int i = len-1; i >= 0; i--)
    if(s[i] != b.s[i]) return s[i] < b.s[i];
  return false;
}

bool operator > (const bign& b) const{
  return b < *this;
}

bool operator <= (const bign& b) {return !(*this > b);}
bool operator >= (const bign& b) {return !(*this < b);}
bool operator == (const bign& b) {
  return !(b < *this) && !(*this < b);
}

bign operator += (const bign& b) {
  *this = *this + b;
  return *this;
}

bign operator << ( int& n )
{
  if( *this == 0 || n == 0 ) return *this;
  bign t = *this;
  int i = t.len-1;
  t.len += n;
  int *p = t.s;
  for( ; i >= 0; i-- )
      p[i+n]=p[i],p[i] = 0;
  for( i = 0; i < n; i++ )
      p[i] = 0;
  return t;
}

   bign operator / (bign b )
  {
  bign ans=0,big=*this,add,chu;
  bign one = 1;
  int t;
  while( big >= b )
  {
      t = big.len - b.len;
      chu = b << t;
      if( chu > big )
```

```
            chu = b << (--t);
        add = one<<t;
        while( chu <= big )
        {
            big = big -  chu;
            ans += add;
        }
    }
    return ans;
    }
};

istream& operator >> (istream &in, bign& x) {
  string s;
  in >> s;
  x = s.c_str();
  return in;
}

ostream& operator << (ostream &out, const bign& x) {
  out << x.str();
  return out;
}

int main() {
    bign a,b,two=2,one=1,zero=0;
    int n,i;

    cin >> n ;
    while( n-- )
    {
        cin >> a;
        b = a/two;
        if( a.s[0]%2==1 ) cout << b << endl;
        else
        {
            b = b-one;
            if( b.s[0]%2 == 0 ) b=b-one;
            cout << b << endl;
        }
    }
    return 0;
}
```