ACM Template

zyeric from BIT

图论

割顶和桥

```
#include <vector>
#include <cstdio>
#include <cstring>

const int maxn = 111;

std::vector<int> g[maxn];
int pre[maxn];
int low[maxn];
bool iscut[maxn];
int dfs_time;

//边(u,v)是否是桥的判别条
件,low[v]>low[u].
int dfs(int fa, int u)
{
    int lowu = pre[u] = ++ dfs_time;
    int child = 0;
    for (int i = 0; i < g[u].size();
i ++) {
        int v = g[u][i];
        if (!pre[v]) {
            child ++;
            int lowv = dfs(u, v);//
利用孩子节点的low值进行更新
            lowu = std::min(lowv,
lowu);
            if (lowv >= pre[u]) {
                iscut[u] = true;
            }
        }
        else if (pre[v] < pre[u] &&
v != fa)//利用反向边更新low值,因为是
无向图,存储的是双向边,所以要对是否是父
亲节点进行特判
        {
            lowu = std::min(lowu,
pre[v]);//在重边的情况下判断桥的条件:
对每个点标记访问它的父向边,如果能通过另
一条重边访问到父节点,则将子节点的low值
更新为pre[u].
        }
    }
    //对于根节点是否是割顶的特判
    if (fa == -1 && child == 1) {
        iscut[u] = false;
    }
    low[u] = lowu;
    return lowu;
}

int main()
{
    int n, m;
    while (scanf("%d%d", &n, &m) ==
2) {
        int a, b;
        for (int i = 1; i <= n; i ++)
{
            g[i].clear();
        }
        for (int i = 0; i < m; i ++)
{
            scanf("%d%d", &a, &b);
            g[a].push_back(b);
            g[b].push_back(a);
        }
        memset(pre, 0,
sizeof(pre));
        memset(iscut, false,
sizeof(iscut));
        dfs_time = 0;
        dfs(-1, 1);
        for (int i = 1; i <= n; i ++)
{
            printf("%d ",
iscut[i]);
        }
        puts("");
        for (int i = 1; i <= n; i ++)
{
            printf("%d ", low[i]);
```

```
        }
    }
}
```

最短路 dij(堆优化)

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <cmath>
#include <queue>
using namespace std;

#define LL long long
const LL inf = 1LL << 60;

const int maxn = (int)1e5 + 10;

struct edge
{
    LL to,cost;
};

typedef pair<LL,int> P;//first是最短距离,second是顶点编号

vector<edge> G[maxn];
int V;
LL  d[maxn];

void dijkstra(int s)
{
    priority_queue<P >  que;

    //距离初始化
    for(int i=0;i<V;i++)
    {
        d[i]=inf;
    }

    d[s]=0;

    //访问数组
    static bool used[maxn];
    for(int i=0;i<V;i++)
    {
        used[i]=false;
    }

    //因为堆默认是最大堆,所以可以取相反数处理
    for(int i=0;i<V;i++)
    {

que.push(make_pair(-d[i],i));
    }

    while(!que.empty())
    {
        int v=que.top().second;
        que.pop();
        if(used[v])  continue;
        used[v]=true;
        int size=(int)G[v].size();
        for(int i=0;i<size;i++)
        {
            edge e =G[v][i];

if(d[e.to]>d[v]+e.cost)
            {

d[e.to]=d[v]+e.cost;

que.push(P(-d[e.to],e.to));
            }
        }
    }
}
```

dfs 生成欧拉序列

```
#include <vector>
#include <cstdio>
#include <cstring>

const int maxn = 111;
```

```
std::vector<int> g[maxn];
bool vis[maxn];
int len;
int euler[maxn<<2];


void dfs(int fa,int u)
{
    if (vis[u]) {
        return;
    }
    vis[u] = true;
    euler[len ++] = u;
    for (int i = 0; i < g[u].size();
i ++) {
        if(g[u][i] != fa)
        {
            dfs(u, g[u][i]);
            euler[len ++] = u;
        }
    }
}

int main()
{
    int n;
    while (scanf("%d", &n) == 1) {
        int a, b;
        for (int i = 0; i + 1 < n; i
++) {
            scanf("%d%d", &a, &b);
            g[a].push_back(b);
            g[b].push_back(a);
        }
        memset(vis, false,
sizeof(vis));
        len = 0;
        dfs(0, 1);
        for (int i = 0; i < len; i ++)
{
            printf("%d ",
euler[i]);
        }
        puts("");
    }
}
```

spfa

```
//spfa,当一个点进入队列大于n次,则存在
负环
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <vector>
#include <queue>
using namespace std;

const int maxn = 111;
const int inf = 0xfffffff;

struct edge
{
    int to, cost;
    edge(int to,int cost)
    {
        this -> to = to;
        this -> cost = cost;
    }
};

vector<edge>   g[maxn];
bool vis[maxn];
int d[maxn];
int n;

//双端队列优化spfa.

void spfa(int s)
{
    fill(d+1,d+n+1,inf);
    memset(vis, false,
sizeof(vis));
    d[1]=0;
    vis[1]=true;
    deque<int> q;
    q.push_back(1);
```

```
    while(!q.empty())
    {
        int v = q.front();
        q.pop_front();
        vis[v]=false;
        for(int
i=0;i<g[v].size();i++)
        {
            int to = g[v][i].to;
            int cost = g[v][i].cost;
            if(d[v]+cost<d[to])
            {
                d[to]=d[v]+cost;
                if(!vis[to])
                {
                    if(!q.empty())
                    {

if(d[to]>d[q.front()])
                        {

q.push_back(to);
                        }
                        else
                        {

q.push_front(to);
                        }
                    }
                    else
                    {

q.push_back(to);
                    }
                    vis[to]=true;
                }
            }
        }
    }
}
```

max_flow Dinic

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <cmath>
#include <queue>
using namespace std;

//max_flow, Dinic algorithm

const int maxn = 0xfff;//点的个数
const int inf = 0xffffff;

struct edge
{
    int to,cap;
    int rev;//rev表示反向边的位置
    edge(int to,int cap,int rev)
    {
        this->to = to;
        this->cap = cap;
        this->rev = rev;
    }
};

//每次使用前将vector清空
vector <edge> g[maxn];
int level[maxn];
int iter[maxn];

void add_edge(int from,int to,int
cap)
{
        g[from].push_back(edge(to
,cap,(int)g[to].size()));
        g[to].push_back(edge(from
,0,(int)g[from].size()-1));
}

//每次bfs预处理出分层图,在分层图上dfs
void bfs(int s)
{
        memset(level,-1,sizeof(le
vel));
        queue<int> que;
        level[s]=0;
```

```
        que.push(s);
        while(!que.empty())
    {
            int v=que.front();
que.pop();
            for(int
i=0;i<g[v].size();i++)
        {
                edge &e=g[v][i];

if(e.cap>0&&level[e.to]<0)
           {

level[e.to]=level[v]+1;
                    que.push(e.to);
                }
            }
        }
}


int dfs(int v,int t,int f)
{
        if(v==t)  return f;
        for(int
&i=iter[v];i<g[v].size();i++)//每
一次dfs后修改弧标记,下次dfs后从标记的
弧开始做dfs
    {
            edge &e=g[v][i];

if(e.cap>0&&level[v]<level[e.to]
)
        {
                int
d=dfs(e.to,t,min(f,e.cap));
                if(d>0)
            {
                    e.cap-=d;

g[e.to][e.rev].cap+=d;
                    return d;
                }
            }
        }
        return 0;
}

int max_flow(int s,int t)
{
        int flow=0;
        for(;;)
    {
            bfs(s);
            if(level[t]<0)  return
flow;

memset(iter,0,sizeof(iter));
            int f;

while((f=dfs(s,t,inf))>0)
        {
            flow+=f;
        }
        }
}
```

二分图最大匹配

```
#include <cstdio>
#include <cstring>
#include <vector>
#include <cmath>
#include <queue>
using namespace std;

//二分图最大匹配
//匹配过程由dfs实现
const int maxn = 555;
vector <int> g[maxn];
bool vis[maxn];
//left 和 right数组存储匹配的信息
int left[maxn];
int right[maxn];

//单向边
void add_edge(int a,int b)
```

```
{
    g[a].push_back(b);
}

//match函数是dfs过程
bool match(int i)
{
    for(int
j=0;j<g[i].size();j++)
    {
        if(!vis[g[i][j]])
        {
            int tar=g[i][j];
            vis[tar]=true;

if(left[tar]==-1||match(left[tar
]))
            {
                left[tar]=i;
                right[i]=tar;
                return true;
            }
        }
    }
    return false;
}

void solve(int size)
{
    int ans=0;
    memset(left,-1,sizeof(left));

memset(right,-1,sizeof(right));
    //每次寻找增广路增加答案
    for(int i=1;i<=size;i++)
    {

memset(vis,false,sizeof(vis));
        if(match(i))  ans++;
    }
    printf("%d\n",ans);
}
```

最小费用路模板

```
const int N = 1010;//点

const int M = 2 * 10010;//边

const int inf = 1000000000;

struct Node{//边，点 f 到点 t，流量为 c，费用为 w

    int f, t, c, w;

}e[M];

int next1[M], point[N], dis[N], q[N], pre[N], ne;//ne 为已添加的边数，next，point 为邻接表,dis 为花费，pre 为父亲节点

bool u[N];

void init(){

    memset(point, -1, sizeof(point));

    ne = 0;

}

void add_edge(int f, int t, int d1, int d2, int w){//f 到 t 的一条边，流量为 d1,反向流量 d2,花费 w,反向边花费-w（可以反悔）

    e[ne].f = f, e[ne].t = t, e[ne].c = d1, e[ne].w = w;

    next1[ne] = point[f], point[f] = ne++;

    e[ne].f = t, e[ne].t = f, e[ne].c = d2, e[ne].w = -w;

    next1[ne] = point[t], point[t] = ne++;

}
```

```
bool spfa(int s, int t, int n){
    int i, tmp, l, r;
    memset(pre, -1, sizeof(pre));
    for(i = 0; i < n; ++i)
        dis[i] = inf;
    dis[s] = 0;
    q[0] = s;
    l = 0, r = 1;
    u[s] = true;
    while(l != r) {
        tmp = q[l];
        l = (l + 1) % (n + 1);
        u[tmp] = false;
        for(i = point[tmp]; i != -1; i = next1[i]) {
            if(e[i].c && dis[e[i].t] > dis[tmp] + e[i].w) {
                dis[e[i].t] = dis[tmp] + e[i].w;
                pre[e[i].t] = i;
                if(!u[e[i].t]) {
                    u[e[i].t] = true;
                    q[r] = e[i].t;
                    r = (r + 1) % (n + 1);
                }
            }
        }
    }
    if(pre[t] == -1)
        return false;
    return true;
}

void MCMF(int s, int t, int n, int &flow, int &cost){//起点 s，终点 t，点数 n，最大流 flow，最小花费 cost
    int tmp, arg;
    flow = cost = 0;
    while(spfa(s, t, n)) {
        arg = inf, tmp = t;
        while(tmp != s) {
            arg = min(arg, e[pre[tmp]].c);
            tmp = e[pre[tmp]].f;
        }
        tmp = t;
        while(tmp != s) {
            e[pre[tmp]].c -= arg;
```

```
            e[pre[tmp] ^ 1].c += arg;

            tmp = e[pre[tmp]].f;

        }

        flow += arg;

        cost += arg * dis[t];

    }

}
```

//建图前运行 init()

//节点下标从 0 开始

//加边时运行 add_edge(a,b,c,0,d)表示加一条 a 到 b 的流量为 c 花费为 d 的边（注意花费为单位流量花费）

//特别注意双向边，运行 add_edge(a,b,c,0,d),add_edge(b,a,c,0,d)较好，不要只运行一次 add_edge(a,b,c,c,d),费用会不对。

//求解时代入 MCMF(s,t,n,v1,v2)，表示起点为 s，终点为 t，点数为 n 的图中，最大流为 v1，最大花费为 v2

BCC(点双联通分量分解)

```
#include <vector>
#include <cstdio>
#include <cstring>
#include <stack>

const int maxn = 111;

struct edge
{
    int from, to;
};

//边双联通分量的实现方法:先探查出桥,并标记,第二次dfs不经过桥
std::vector<int> g[maxn],
bcc[maxn];
std::stack<edge> S;//利用栈存储点双联通分量的边.
int pre[maxn];
int low[maxn];
int bccno[maxn];
bool iscut[maxn];
int dfs_time, bcc_cnt;


//边(u,v)是否是桥的判别条件,low[v]>low[u].
int dfs(int fa, int u)
{
    int lowu = pre[u] = ++ dfs_time;
    int child = 0;
    for (int i = 0; i < g[u].size(); i ++) {
        int v = g[u][i];
        edge e = (edge){u, v};
        if (!pre[v]) {
            S.push(e);
            child ++;
            int lowv = dfs(u, v);//利用孩子节点的low值进行更新
            lowu = std::min(lowv, lowu);
            if (lowv >= pre[u]) {
                iscut[u] = true;
                bcc_cnt ++;

bcc[bcc_cnt].clear();
                for (; ; ) {
                    edge x = S.top();
                    S.pop();
                    if (bccno[x.from] != bcc_cnt) {

bcc[bcc_cnt].push_back(x.from);
```

```
                    bccno[x.from]
= bcc_cnt;
                }
                if
(bccno[x.to] != bcc_cnt) {

bcc[bcc_cnt].push_back(x.to);
                    bccno[x.to] =
bcc_cnt;
                }
                if (x.from == u &&
x.to == v) {
                    break;
                }
            }
        }
    }
    else if (pre[v] < pre[u] &&
v != fa)//利用反向边更新low值,因为是无向图,存储的是双向边,所以要对是否是父亲节点进行特判
    {
        S.push(e);
        lowu = std::min(lowu,
pre[v]);//在重边的情况下判断桥的条件:对每个点标记访问它的父向边,如果能通过另一条重边访问到父节点,则将子节点的low值更新为pre[u].
    }
  }
  //对于根节点是否是割顶的特判
  if (fa == -1 && child == 1) {
    iscut[u] = false;
  }
  low[u] = lowu;
  return lowu;
}

int main()
{
  int n, m;
  while (scanf("%d%d", &n, &m) ==
2) {
    int a, b;
    for (int i = 1; i <= n; i ++)
{
      g[i].clear();
    }
    for (int i = 0; i < m; i ++)
{
      scanf("%d%d", &a, &b);
      g[a].push_back(b);
      g[b].push_back(a);
    }
    memset(pre, 0,
sizeof(pre));
    memset(iscut, false,
sizeof(iscut));
    dfs_time = bcc_cnt = 0;
    for (int i = 1; i <= n; i ++)
{
      if (!pre[i]) {
        dfs(-1, i);
      }
    }
    for (int i = 1; i <= n; i ++)
{
      printf("%d ",
bccno[i]);
    }
    puts("");
  }
}
```

SCC(强联通分量分解)

```
#include <vector>
#include <cstdio>
#include <cstring>
#include <stack>

const int maxn = 111;

std::vector<int>  g[maxn];
std::stack<int>    S;
int pre[maxn];
int lowlink[maxn];
```

```
int sccno[maxn];//从1开始标号
int dfs_time, scc_cnt;

void dfs(int u)
{
   lowlink[u] = pre[u] = ++
dfs_time;
   S.push(u);
   for (int i = 0; i < g[u].size();
i ++) {
       int v = g[u][i];
       if (!pre[v]) {
          dfs(v);
          lowlink[u] =
std::min(lowlink[u], lowlink[v]);
       }
       else if (!sccno[v])
       {
          lowlink[u] =
std::min(lowlink[u], pre[v]);
       }
   }
   if (lowlink[u] == pre[u]) {
       scc_cnt ++;
       for (; ; ) {
          int x = S.top(); S.pop();
          sccno[x] = scc_cnt;
          if(x == u)  break;
       }
   }
}

int main()
{
   int n, m;
   while (scanf("%d%d", &n, &m) ==
2) {
       dfs_time = 0;
       scc_cnt = 0;
       for (int i = 1; i <= n; i ++)
{
          g[i].clear();
       }
       int a, b;
       while (m --) {
          scanf("%d%d", &a, &b);
          g[a].push_back(b);
       }
       memset(pre, 0,
sizeof(pre));
       memset(lowlink, 0,
sizeof(lowlink));
       memset(sccno, 0,
sizeof(sccno));
       for (int i = 1; i <= n ; i ++)
{
          if (!sccno[i]) {
             dfs(i);
          }
       }
       for (int i = 1; i <= n; i ++)
{
          printf("%d ",
sccno[i]);
       }
       puts("");
   }
   return 0;
}
```

2-sat

```
#include <iostream>
#include <cstdio>
#include <cstring>

#define MAXV 200010
#define MAXE 2000010
#define T(x) ((x) << 1)
#define F(x) (((x) << 1) | 1)

using namespace std;

struct _2SAT {
   struct Edge {
       int ed, next;
   } edgex[MAXE], edgey[MAXE];
```

```
    int headx[MAXV], heady[MAXV], n,
nv, ne;
    int sid[MAXV], vs[MAXV], scnt,
idx;
    bool vis[MAXV];

    void init(int _n) {
        memset(headx, -1,
sizeof(headx));
        memset(heady, -1,
sizeof(heady));
        n = _n; nv = (n << 1); ne =
0;
    }

    void addEdge(int a, int b) {
        edgex[ne].ed = b;
edgex[ne].next = headx[a];
        edgey[ne].ed = a;
edgey[ne].next = heady[b];
        headx[a] = heady[b] = ne++;
    }

    void addOR(int a, int b) {
        addEdge(a ^ 1, b); addEdge(b
^ 1, a);
    }

    void addAND(int a, int b) {
        addEdge(a ^ 1, a); addEdge(b
^ 1, b);
    }

    void addXOR(int a, int b) {
        addOR(a, b); addOR(a ^ 1, b
^ 1);
    }

    void dfsx(int x) {
        int i;
        vis[x] = true;
        for (i = headx[x]; ~i; i =
edgex[i].next) {
            if (!vis[edgex[i].ed])
dfsx(edgex[i].ed);
        }
        vs[--idx] = x;
    }

    void dfsy(int x, int k) {
        int i;
        vis[x] = true; sid[x] = k;
        for (i = heady[x]; ~i; i =
edgey[i].next) {
            if (!vis[edgey[i].ed])
dfsy(edgey[i].ed, k);
        }
    }

    bool solve() {
        int i;
        memset(vis, 0,
sizeof(vis));
        for (idx = nv, i = 0; i < nv;
++i) {
            if (!vis[i]) dfsx(i);
        }
        memset(vis, 0,
sizeof(vis));
        for (scnt = i = 0; i < nv; ++i)
{
            if (!vis[vs[i]])
dfsy(vs[i], scnt++);
        }
        for (i = 0; i < n; ++i) {
            if (sid[T(i)] ==
sid[F(i)]) return false;
        }
        return true;
    }

    bool getAns(int k) const {
        return sid[T(k)] >
sid[F(k)];
    }
} sat;
```

数学

矩阵快速幂

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

#define MAX 5
#define MOD 1000000007
#define LL long long

LL a[MAX][MAX], b[MAX][MAX],
c[MAX][MAX], buff[MAX][MAX];
LL vec[MAX], tmp[MAX];

void matCpy(LL a[MAX][MAX], LL
b[MAX][MAX], int n) {
    int i, j;
    for (i = 0; i < n; ++i) {
        for (j = 0; j < n; ++j) {
            a[i][j] = b[i][j];
        }
    }
}

void norm(LL a[MAX][MAX], int n) {
    int i, j;
    for (i = 0; i < n; ++i) {
        for (j = 0; j < n; ++j) {
            a[i][j] = (i == j);
        }
    }
}

void matMul(const LL a[MAX][MAX],
const LL b[MAX][MAX], LL
c[MAX][MAX], int n, LL MOD) {
    int i, j, k;
    for (i = 0; i < n; i ++) {
        for (j = 0; j < n; j ++) {
            c[i][j] = 0;
        }
    }
    for (i = 0; i < n; i ++) {
        for (j = 0; j < n; j ++) {
            if (a[i][j]) {
                for (k = 0; k < n; k
++) {
                    c[i][k] = c[i][k]
+ a[i][j] * b[j][k] % MOD;
                }
            }
        }
    }
    for (i = 0; i < n; i ++) {
        for (j = 0;j < n; j ++) {
            c[i][j] %= MOD;
        }
    }
}

//函数接口:c = a ^ b, a 的大小为 n.
void matPow(LL a[MAX][MAX], LL b,
LL c[MAX][MAX], int n, LL MOD) {
    for (norm(c, n); b; b >>= 1) {
        if (b & 1) {
            matMul(c, a, buff, n,
MOD);
            matCpy(c, buff, n);
        }
        matMul(a, a, buff, n, MOD);
        matCpy(a, buff, n);
    }
}

void matMulVec(LL a[MAX][MAX], LL
vec[MAX], int n)
{
    memset(tmp, 0, sizeof(tmp));
    for (int i = 0; i < n; i ++) {
        for (int j = 0; j < n; j ++)
{
            tmp[i] = tem[j] + a[i][j]
* vec[j] % MOD;
        }
```

```
    }
    for (int i = 0; i < n; i ++) {
        vec[i] = tmp[i] % MOD;
    }
}
```

pell 方程

```
import java.math.*;
import java.io.*;
import java.util.*;

public class Main {
    final static BigInteger ZERO =
BigInteger.ZERO;
    final static BigInteger ONE =
BigInteger.ONE;
    final static BigInteger TWO =
BigInteger.valueOf(2);
    final static BigInteger FOUR =
BigInteger.valueOf(4);
    final static BigInteger SIX =
BigInteger.valueOf(6);

    static class Pell {
        BigInteger x, y;
        boolean solve(int D) {
            int         s         =
(int)Math.sqrt((double)D);
            if (s * s == D) return
false;
            BigInteger     N     =
BigInteger.valueOf(D);
            BigInteger   sqrtD   =
BigInteger.valueOf(s);
            BigInteger c = sqrtD, q
= N.subtract(c.pow(2));
            BigInteger     a     =
c.add(sqrtD).divide(q);
            int step = 0;
            BigInteger    X[]    =
{BigInteger.ONE, sqrtD};
            BigInteger    Y[]    =
{BigInteger.ZERO,
BigInteger.ONE};
            while (true) {
                X[step]           =
a.multiply(X[step                 ^
1]).add(X[step]);
                Y[step]           =
a.multiply(Y[step                 ^
1]).add(Y[step]);
                c                 =
a.multiply(q).subtract(c);
                q                 =
N.subtract(c.pow(2)).divide(q);
                a                 =
c.add(sqrtD).divide(q);
                step ^= 1;
                if (c.equals(sqrtD)
&&  q.equals(BigInteger.ONE)  &&
step == 1) {
                    x = X[0]; y = Y[0];
                    return true;
                }
            }
        }
    }
    public static void main(String
args[]) throws Exception{
        Scanner    cin    =    new
Scanner(System.in);
        int p, A;
        while        ((p        =
cin.nextInt()) != 0) {
            A = cin.nextInt();
            Pell equa = new Pell();
            boolean flag = false;
            if (p == 3) {
                if (equa.solve(8 * A)
&&
equa.x.subtract(ONE).mod(TWO).eq
uals(ZERO)) {
                    equa.x        =
equa.x.subtract(ONE).divide(TWO)
;
                    flag = true;
```

```
            }
        }
        else if (p == 5) {
            if (equa.solve(24 * A)
&&
equa.x.add(ONE).mod(SIX).equals(
ZERO)) {
                equa.x         =
equa.x.add(ONE).divide(SIX);
                flag = true;
            }
        }
        else if (p == 6) {
            if (equa.solve(8 * A)
&&
equa.x.add(ONE).mod(FOUR).equals
(ZERO)) {
                equa.x         =
equa.x.add(ONE).divide(FOUR);
                flag = true;
            }
        }
        if              (!flag)
System.out.println("Impossible!"
);
        else
System.out.println(equa.x + " " +
equa.y);
      }
   }
}
```

数值积分

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

//数值积分,simpson公式
double f(double x)
{
    return x;
}
//a,b为积分上下限,n为划分份数
double simpson(double a,double
b,int n)
{
    double h = (b-a)/n;
    double ans= f(a)+f(b);
    for(int i=1;i<n;i+=2)
ans+=4.*f(a+i*h);
    for(int i=2;i<n;i+=2)
ans+=2.*f(a+i*h);
    return ans*h/3.;
}
```

gauss 消元

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

#define maxn 0xf

//线性方程组求解
//n个变元,n个方程
//A是增广矩阵,即A[i][n]是第i个方程右
边的常数bi
//运行结束后A[i][n]是第i个未知数的值

typedef double
Martrix[maxn][maxn];

void gauss_elimination(Martrix
A,int n)
{
    int i,j,k,r;
    //将系数矩阵化为上三角矩阵
    //每一次处理一列,为了减少精度损失,
每一次按列最大的元素进行消元
    for(i=0;i<n;i++)
    {
        r=i;
        //找出该列系数最大的行
        for(j=i+1;j<n;j++)
```

```
if(fabs(A[j][i])>fabs(A[r][i]))
r=j;
       //如果第i行不是第i个元素最大的
行,则将它与系数最大的行进行交换
       if(r!=i)  for(j=0;j<=n;j++)
swap(A[r][j],A[i][j]);
       //为了保证精度,从后到前进行消
元
       for(j=n;j>=i;j--)
       {
          for(k=i+1;k<n;k++)

A[k][j]-=A[k][i]/A[i][i]*A[i][j]
;
       }
    }
    //回代过程
    for(i=n-1;i>=0;i--)
    {
       for(j=i+1;j<n;j++)

A[i][n]-=A[j][n]*A[i][j];
       A[i][n]/=A[i][i];
    }
}



//gauss_jordan
//注意如何判定解是否存在
//正常情况下 xi=A[i][n]/A[i][i]
void gauss_jordan(Martrix A,int n)
{
    int i,j,k,r;
    for(i=0;i<n;i++)
    {
       r=i;
       for(j=i+1;j<n;j++)

if(fabs(A[j][i])>fabs(A[r][i]))
r=j;
       if(fabs(A[r][i])<eps)
continue;
       if(r!=i)  for(j=0;j<=n;j++)
swap(A[r][j],A[i][j]);
       for(k=0;k<n;k++)  if(k!=i)
          for(j=n;j>=i;j--)
A[k][j]-=A[k][i]/A[i][i]*A[i][j]
;
    }
}
```

mobius 函数

```
//mobius函数可以分块处理

int mu[100010];

void init(){
       for(int i =
1;i<=100000;i++){
          int t = (i==1?1:0);
          int d = t-mu[i];
          mu[i] = d;
          for(int j =
i+i;j<=100000;j+=i)
             mu[j]+=d;
       }
}
```

Number Theory

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <map>
#include <vector>
const int MOD = 1e9 + 7;

//注意题目数据范围,能用int用int,LL有
时会超时
#define LL long long
using namespace std;

//线性素数筛法
```

```
const long N = 200000;
long prime[N] = {0},num_prime = 0;
int isNotPrime[N] = {1, 1};
void produce_prime()
{
    for(long i = 2 ; i < N ; i ++)
    {
            if(! isNotPrime[i])
                prime[num_prime
++]=i;
            for(long j = 0 ; j <
num_prime && i * prime[j] <  N ; j
++)
       {

            isNotPrime[i * prime[j]]
= 1;
                if( !(i %
prime[j] ) )
                    break;
            }
        }
}

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

//线性初始化逆元表
int fac[N], inv[N], rv[N];
void init() {
    fac[0] = inv[0] = fac[1] = inv[1]
= rv[1] = 1;
    for (int i=2; i<MOD; i++) {
        rv[i] = (LL)rv[MOD % i] *
(MOD - MOD / i) % MOD;

fac[i]=((LL)fac[i-1]*i)%MOD;
        inv[i] = (LL)inv[i - 1] *
rv[i] % MOD;
    }
}
```

```
//lucas定理
LL lucas(LL n, LL m, LL mod, int
inv[], int fac[]) {
    if (n == 0 && m == 0) return 1;
    LL a = n % mod, b = m % mod;
    if (b > a) return 0;
    return (((((lucas(n / mod, m /
mod, mod, inv, fac) * fac[a]) % mod)
* inv[b]) % mod) * inv[a - b]) % mod;
}

//拓展gcd，求解ax+by=d,d=gcd(a,b),
且|x|+|y|最小
void extend_gcd(LL a,LL b,LL &d,LL
&x,LL &y)
{
    if(!b)
        {
            d=a;
            x=1;
            y=0;
        }else{

        extend_gcd(b,a%b,d,y,x);
            y-=x*(a/b);
        }
}

//当可能有溢出情况出现时使用
LL mul_mod(LL a,LL b,LL c)
{
    LL ret=0,tmp=a%c;
    while(b)
    {
        if(b&0x1)
                if((ret+=tmp)>=c)
                    ret-=c;
        if((tmp<<=1)>=c)
                tmp-=c;
        b>>=1;
    }
    return ret;
}
```

```
//快速幂
LL pow_mod(LL a,LL n,LL mod)
{
    LL ret = 1;
    LL tmp = a % mod;
    while (n)
    {
        if (n & 1)
        {
            ret = ret * tmp % mod;
        }
        tmp = tmp * tmp % mod;
        n >>= 1;
    }
    return ret;
}

//求单个数的欧拉函数值
LL euler_phi(LL n)
{
    LL m, ans;

        m=(int)sqrt(n+0.5) + 1;
        ans = n;
    for(int 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;
}

//求多个数的欧拉函数值
LL phi[N];
void phi_table(LL n)
{
    for(int i=2;i<=n;i++)
           phi[i]=0;
    phi[1]=1;
    for(int i=2;i<=n;i++)
        {
        if(!phi[i])
            {
            for(int j=i;j<=n;j+=i)
                {
                if(!phi[j])
                        phi[j]=j;

phi[j]=phi[j]/i*(i-1);
            }
        }
    }
}

//求逆元
LL inv(LL a,LL n)
{
    LL d,x,y;
    extend_gcd(a, n, d, x, y);
    return d == 1? (x+n)%n:-1;
}

//求单个线性同余方程ax=b(mod n)
LL linear_mod(LL a,LL b,LL n)
{
    LL d, tem;
    d=gcd(a,b);
    if(b%d==0)
        {
        a=a/d;
            b=b/d;
            n=n/d;
        tem=inv(a, n);
        return tem*b%n;
    }
    return -1;
}

//CRT，n个方程：x=a[i](mod m[i])
0<=i<n
LL CRT(int n,int *a,int *m)
{
    LL M=1,d,y,x=0;

    for(int i=0;i<n;i++)  M*=m[i];
```

```
    for(int i=0;i<n;i++){
        LL w=M/m[i];
        extend_gcd(m[i],w,d,d,y);
        x=(x+y*w*a[i])%M;
    }
    return (x+M)%M;
}

//可以不互素的一次同余方程组求解
LL extend_crt(int n,int *a,int *m){
    LL a1,m1,a2,m2;
    LL flag=0;
    LL d,tem,K;

    a1=a[0];
        m1=m[0];
    for(int i=1;i<n;i++)
        {
        a2=a[i-1];
            m2=m[i-1];
        d=gcd(m1,m2);
        if((a2-a1)%d==0)
            {
            tem=inv(m1/d, m2/d);

K=(tem*(a2-a1)/d)%(m2/d);

a1=(K*m1+a1)%(m1*m2/d);
            m1=m1*m2/d;
        }else{
            flag=1;
        }
        if(flag==1)
                return -1;
    }
    return (a1+m1)%m1;//注意返回值是0的情况，这时候根据题目要判断是否加m1
}

//离散对数 baby-step,giant-steap
LL log_mod(LL a,LL b,LL n)
{
    LL m,v,e=1,i;
        map <LL,LL> x;

    m=(int)sqrt(n+0.5);
    v=inv(pow_mod(a, m, n),n);
    x[1]=0;
    for(i=1;i<m;i++)
        {
        e=mul_mod(e, a, n);
        if(!x.count(e))
                x[e]=i;
    }
    for(i=0;i<m;i++)
        {
        if(x.count(b))
                return  i*m+x[b];
        b=mul_mod(b, v, n);
    }
    return -1;
}

//生成较小的组合数
#define MAX 0xfff
LL cb[MAX][MAX];
//先初始化cb数组都为-1
LL calc(int n, int k)
{
    if (!k || n == k)
            return 1;
    if (~cb[n][k])
            return cb[n][k];
    return cb[n][k] = (calc(n - 1, k - 1) + calc(n - 1, k)) % MOD;
}

//第一类stirling数 讲n个元素分为k组，每组中圆排列的组数
LL stirling[MAX][MAX];
LL calstirling(int n,int k)
{
    if(k>n||(n>0&&k==0))
        return 0;
    if(n==k)
        return 1;
    if(~stirling[n][k])
        return stirling[n][k];
```

```
    return
stirling[n][k]=(calstirling(n-1,
k-1)+(n-1)*calstirling(n-1,k))%M
OD;
}

//分解质因数
void factor(int n,int a[MAX],int
b[MAX],int &tot)
{
    int temp,i,now;

    temp=(int)(sqrt(n)+1);
    tot=0;
    now=n;
    for(i=2;i<=temp;++i)
    {
        if(now%i==0)
        {
            a[++tot]=i;
            b[tot]=0;
            while(now%i==0)
            {
                ++b[tot];
                now/=i;
            }
        }
    }
    if(now!=1){
        a[++tot]=now;
        b[tot]=1;
    }
}

//求原根,返回最小原根
vector<LL> a;

bool g_test(LL g,LL p)
{
    for(LL i=0;i<a.size();i++)

if(pow_mod(g,(p-1)/a[i],p)==1)
            return 0;
    return 1;
}

LL primitive_root(LL p)
{
    LL tmp=p-1;
    for(LL i=2;i<=tmp/i;i++)
        if(tmp%i==0)
            {
            a.push_back(i);
            while(tmp%i==0)
                tmp/=i;
        }
    if(tmp!=1)
        {
        a.push_back(tmp);
    }
    LL g=1;
    while(true)
        {
        if(g_test(g,p))
            return g;
        ++g;
    }
}
```

Polynomial

```
#include <iostream>
#include <cstdio>
#include <cstring>
#include <algorithm>

using namespace std;

#define MAX 0xfff
#define MOD 1000000007
#define LL long long

struct Polynomial
{
    LL v[MAX];
    int size;
};
```

```
void print(Polynomial a)
{
    printf("%d\n",a.size);
    for(int i=a.size-1;i>=0;i--)
printf("%lld ",a.v[i]);
    puts("");
}

void init_Poly(Polynomial a,int
n,int x)
{
    a.size=n;
    for(int i=0;i<n;i++)
a.v[i]=x;
}

Polynomial make_mul(Polynomial
a,Polynomial b)
{
    Polynomial c;
    c.size=a.size+b.size-1;
    int i,j;
    for(i=0;i<c.size;i++)
c.v[i]=0;
    for(i=0;i<a.size;i++)
    {
        if(!a.v[i])  continue;
        for(j=0;j<b.size;j++)
        {

c.v[i+j]+=a.v[i]*b.v[j];

c.v[i+j]=(c.v[i+j]+MOD)%MOD;
        }
    }
    return c;
}

Polynomial make_mod(Polynomial
a,Polynomial b)
{
    int i,j,n=b.size-1;
    for(i=a.size-1;i>=n;i--)
    {
        if(!a.v[i])  continue;
        for(j=1;j<=n;j++)
        {

a.v[i-j]-=a.v[i]*b.v[n-j];
            a.v[i-j]%=MOD;
            if(a.v[i-j]<0)
a.v[i-j]=(a.v[i-j]+MOD)%MOD;
        }
    }
    a.size=min(a.size,n);
    return a;
}

Polynomial make_pow(LL
n,Polynomial a,Polynomial b)
{
    Polynomial c;
    Polynomial tem=a;
    c.size=1;
    c.v[0]=1;
    for(;n;n>>=1)
    {
        if(n%2==1)
        {
            Polynomial
d=make_mul(c,tem);
            c=make_mod(d,b);
        }

tem=make_mod(make_mul(tem,tem),b
);
    }
    return c;
}

int main(int argc, const char *
argv[])
{
    return 0;
}
```

高次多项式求根

```
const double EPS = 1e-12;
const double INF = 1e+18;

inline int sign(double x)
{
    return x < -EPS ? -1 : x > EPS;
}

inline double get(const
vector<double>&coef, double x)
{
    double e = 1, s = 0;
    for (int i = 0; i < coef.size();
i ++) {
        s += coef[i] * e;
        e *= x;
    }
    return s;
}

double find(const
vector<double>&coef, int n, double
lo, double hi)
{
    double sign_lo, sign_hi;
    if ((sign_lo = sign(get(coef,
lo))) == 0) {
        return lo;
    }
    if ((sign_hi = sign(get(coef,
hi))) == 0) {
        return hi;
    }
    if (sign_lo * sign_hi > 0) {
        return INF;
    }
    for (int step = 0; step < 100 &&
hi - lo > EPS; step ++) {
        double mid = (lo + hi) * .5;
        int sign_mid =
sign(get(coef, mid));
        if (sign_mid == 0) {
            return mid;
        }
        if (sign_lo * sign_mid < 0)
{
            hi = mid;
        }
        else
        {
            lo = mid;
        }
    }
    return (lo + hi) * .5;
}

vector<double>
equation(vector<double> coef, int
n)
{
    vector<double> ret;
    if (n == 1) {
        if (sign(coef[1])) {
            ret.push_back(-coef[0]
/ coef[1]);
            return ret;
        }
    }
    vector<double> dcoef(n);
    for (int i = 0; i < n; i ++) {
        dcoef[i] = coef[i + 1] * (i
+ 1);
    }
    vector<double> droot =
equation(dcoef, n - 1);
    droot.insert(droot.begin(),
-INF);
    droot.push_back(INF);
    for (int i = 0; i + 1 <
droot.size(); i ++) {
        double tmp = find(coef, n,
droot[i], droot[i + 1]);
        if (tmp < INF) {
            ret.push_back(tmp);
        }
    }
```

```
    return ret;
}
```

Miller-Rabin&Pollard rho

```
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <cstring>
#include <iostream>
const int S=20;
using namespace std;

typedef long long LL;
#define maxn 10000

LL factor[maxn];
int tot;

LL muti_mod(LL a,LL b,LL c){   //返回(a*b) mod c,a,b,c<2^63
    a%=c;
    b%=c;
    LL ret=0;
    while (b){
        if (b&1){
            ret+=a;
            if (ret>=c) ret-=c;
        }
        a<<=1;
        if (a>=c) a-=c;
        b>>=1;
    }
    return ret;
}

LL pow_mod(LL x,LL n,LL mod){  //返回x^n mod c ,非递归版
    if (n==1) return x%mod;
    int bit[64],k=0;
    while (n){
        bit[k++]=n&1;
        n>>=1;
    }
    LL ret=1;
    for (k=k-1;k>=0;k--){
        ret=muti_mod(ret,ret,mod);
        if (bit[k]==1) ret=muti_mod(ret,x,mod);
    }
    return ret;
}

bool check(LL a,LL n,LL x,LL t){   //以a为基，n-1=x*2^t，检验n是不是合数
    LL ret=pow_mod(a,x,n),last=ret;
    for (int i=1;i<=t;i++){
        ret=muti_mod(ret,ret,n);
        if (ret==1 && last!=1 && last!=n-1) return 1;
        last=ret;
    }
    if (ret!=1) return 1;
    return 0;
}

bool Miller_Rabin(LL n){
    LL x=n-1,t=0;
    while ((x&1)==0) x>>=1,t++;
    bool flag=1;
    if (t>=1 && (x&1)==1){
        for (int k=0;k<S;k++){
            LL a=rand()%(n-1)+1;
            if (check(a,n,x,t)) {flag=1;break;}
            flag=0;
        }
    }
    if (!flag || n==2) return 0;
    return 1;
}

LL gcd(LL a,LL b){
    if (a==0) return 1;
    if (a<0) return gcd(-a,b);
```

```
    while (b){
        LL t=a%b; a=b; b=t;
    }
    return a;
}

LL Pollard_rho(LL x,LL c){
    LL i=1,x0=rand()%x,y=x0,k=2;
    while (1){
        i++;

x0=(muti_mod(x0,x0,x)+c)%x;
        LL d=gcd(y-x0,x);
        if (d!=1 && d!=x){
            return d;
        }
        if (y==x0) return x;
        if (i==k){
            y=x0;
            k+=k;
        }
    }
}

void findfac(LL n){          //递归
进行质因数分解N
    if (!Miller_Rabin(n)){
        factor[tot++] = n;
        return;
    }
    LL p=n;
    while (p>=n)
p=Pollard_rho(p,rand() % (n-1)
+1);
    findfac(p);
    findfac(n/p);
}

int main(){
    srand(time(NULL));
    int t;
    scanf("%d",&t);
    while (t--){
        LL n;
        scanf("%lld",&n);
        if (!Miller_Rabin(n))
printf("Prime\n");
        else{
            tot = 0;
            findfac(n);
            LL ans=n;
            for (int i = 0; i < tot;
i++)
                ans = min(ans,
factor[i]);
            printf("%lld\n",ans);
        }
    }
}
```

n 以内约数最多的数

```
#include "cstdio"
#define LL long long
int
prime[16]={1,2,3,5,7,11,13,17,19
,23,29,31,37,41,43,47};
LL n;
LL ans,cnt;
void dfs(LL base,LL pos,LL pre,LL
num){
    if(num>cnt){
        ans=base,cnt=num;
    }
    if(num==cnt&&base<ans)
ans=base;
    for(int i=1;i<=pre;i++){
        base*=prime[pos];
        if(base>n)  break;
        if(pos<15)
dfs(base,pos+1,i,num*(i+1));
    }
}
int main(){
    int t;
    scanf("%d",&t);
    while(t--){
        scanf("%lld",&n);
```

```
        ans=1,cnt=1;
        dfs(1,1,999,1);

printf("%lld %lld\n",ans,cnt);
    }
    return 0;
}
```

数据结构

树链剖分

```
//树链剖分模版
#include <cstdio>
#include <cstring>
#include <algorithm>
using namespace std;
const int max_v = 100005;

int size[max_v];//size[v]为以v为根的子树的个数
int depth[max_v];//depth[v]表示v的深度(根的深度为1)
int top[max_v];//top[v]表示v所在的重链的顶端节点标号
int fa[max_v];//fa[v]表示v的父亲节点
int son[max_v];//son[v]表示与v在同一条重链上的儿子节点标号
int w[max_v];//w[v]表示v与其父亲节点的连边在线段树中的位置
int first[max_v];//邻接表存图
//树链剖分使用log(n)的时间将一条路径分为若干条链

struct edge
{
    int to, next;
};

class segment_tree
{
private:
    int max_val[max_v << 2];
public:
    void init()
    {
        memset(max_val, 0, sizeof(max_val));
    }

    void update(int pos, int v, int l, int r, int rt)
    {
        if (l == r) {
            max_val[rt] = v;
            return;
        }
        int mid = (l + r) >> 1;
        if (pos <= mid) {
            update(pos, v, l, mid, rt << 1);
        }
        else
        {
            update(pos, v, mid + 1, r, rt << 1 | 1);
        }
        max_val[rt] = max(max_val[rt << 1], max_val[rt << 1 | 1]);
    }

    int query(int L, int R, int l, int r, int rt)
    {
        if (L <= l && r <= R) {
            return max_val[rt];
        }
        int mid = (l + r) >> 1;
        if (R <= mid) {
            return query(L, R, l, mid, rt << 1);
        }
        if (L > mid) {
```

```
            return query(L, R, mid +
1, r, rt << 1 | 1);
        }
        int res = query(L, R, l, mid,
rt << 1);
        res = max(query(L, R, mid +
1, r, rt << 1 | 1), res);
        return res;
    }
};

int edge_num, tot_w;
edge pool[max_v << 1];
int d[max_v][3];

void add_edge(int x, int y)
{
    pool[++edge_num].to = y;
    pool[edge_num].next =
first[x];
    first[x] = edge_num;
}

void dfs_1(int v)
{
    size[v] = 1, son[v] = 0;
    for (int i = first[v]; i > 0; i
= pool[i].next) {
        int u = pool[i].to;
        if (u != fa[v]) {
            fa[u] = v;
            depth[u] = depth[v] + 1;
            dfs_1(u);
            if (size[u] >
size[son[v]]) {
                son[v] = u;
            }
            size[v] += size[u];
        }
    }
}

void dfs_2(int v, int top_v)
{
    w[v] = ++tot_w, top[v] = top_v;
    if (son[v] != 0) {
        dfs_2(son[v], top_v);
    }
    for (int i = first[v]; i > 0; i
= pool[i].next) {
        int u = pool[i].to;
        if (u != fa[v] && u != son[v])
{
            dfs_2(u, u);
        }
    }
}

segment_tree tree;

int cal_path(int u, int v)
{
    int f1 = top[u], f2 = top[v];
    int ret = 0;
    while (f1 != f2) {
        if (depth[f1] < depth[f2]) {
            swap(f1, f2), swap(u,
v);
        }
        ret = max(ret,
tree.query(w[f1], w[u], 1, tot_w,
1));
        u = fa[f1], f1 = top[u];
    }
    //如果是按点剖分,此处不返回.
    if (u == v) {
        return ret;
    }
    if (depth[u] > depth[v]) {
        swap(u, v);
    }
    ret = max(ret,
tree.query(w[son[u]], w[v], 1,
tot_w, 1));
    return ret;
}

void init()
```

```
{
    int n;
    scanf("%d", &n);
    int a, b, c;
    int root = (n + 1) / 2;
    memset(size, 0, sizeof(size));
    memset(first, 0,
sizeof(first));
    tree.init();
    edge_num = tot_w = 0;
    fa[root] = depth[root] = 0;
    for (int i = 1; i < n; i ++) {
        scanf("%d%d%d", &a, &b,
&c);
        add_edge(a, b), add_edge(b,
a);
        d[i][0] = a, d[i][1] = b,
d[i][2] = c;
    }
    dfs_1(root);
    dfs_2(root, root);
    for (int i = 1; i < n; i ++) {
        a = d[i][0], b = d[i][1], c
= d[i][2];
        if (depth[a] < depth[b]) {
            swap(a, b);
        }
        tree.update(w[a], c, 1,
tot_w, 1);
    }
}

void solve()
{
    init();
    char op[10];
    while (1) {
        scanf("%s", op);
        if (op[0] == 'D') {
            break;
        }
        int x, y;
        scanf("%d%d", &x, &y);
        if (op[0] == 'C') {
            int a = d[x][0], b =
d[x][1];
            if (depth[a] < depth[b])
{
                swap(a, b);
            }
            tree.update(w[a], y, 1,
tot_w, 1);
        }
        else
        {
            printf("%d\n",
cal_path(x, y));
        }
    }
}

int main()
{
    int T;
    scanf("%d", &T);
    while (T--) {
        solve();
    }
    return 0;
}
```

ST 表

```
const int MAXN = 100005;
const int MAX_LOG = 17;

int pre_log[MAXN];
int st[MAXN][MAX_LOG];
int n;

void init()
{
    pre_log[1] = 0;
    for (int i = 2; i <= n; i ++) {
        pre_log[i] = pre_log[(i >>
1)] + 1;
    }
    for (int i = n; i >= 1; i --) {
```

```
       for (int j = 0; i + (1 << (j
+ 1)) - 1 <= n; j ++) {
           st[i][j + 1] =
gcd(st[i][j], st[i + (1 << j)][j]);
       }
   }
}

int query(int l, int r)
{
   int t = pre_log[r - l + 1];
   return gcd(st[l][t], st[r - (1
<< t) + 1][t]);
}
```

矩形面积并

```
#include "cstdio"
#include "algorithm"
using namespace std;
#define lson l,m,rt<<1
#define rson m+1,r,rt<<1|1
const int maxn=2222;
struct edge
{
   double a,b,h;
   int s;
};
edge e[maxn];
bool cmp(edge &p,edge &q){return
p.h<q.h;}
double sum[maxn<<2];
int cnt[maxn<<2];
double x[maxn];
void pushup(int l,int r,int rt)
{
   if(cnt[rt]!=0)
sum[rt]=x[r+1]-x[l];
   else  if(l==r)  sum[rt]=0;
   else
sum[rt]=sum[rt<<1]+sum[rt<<1|1];
}
void update(int L,int R,int c,int
l,int r,int rt)
{
   if(L<=l&&R>=r)
   {
       cnt[rt]+=c;
       pushup(l,r,rt);
       return;
   }
   int m=(l+r)>>1;
   if(L<=m)  update(L,R,c,lson);
   if(R>m)  update(L,R,c,rson);
   pushup(l,r,rt);
}
int bst(double v,double x[],int n)
{
   int l=0,r=n-1,m;
   while(l<=r)
   {
       m=(l+r)>>1;
       if(x[m]==v)  return m;
       if(x[m]<v)  l=m+1;
       else  r=m-1;
   }
   return -1;
}
int main()
{
   int n;
   int kase=1;
   double a,b,c,d;
   while(scanf("%d",&n)!=EOF&&n)
   {
       int m=0;
       double ans=0;
       for(int i=0;i<n;i++)
       {

scanf("%lf%lf%lf%lf",&a,&b,&c,&d
);//矩形左下角和右上角
           x[m]=a;

e[m].a=a,e[m].b=c,e[m].h=b,e[m++
].s=1;//add the up edge
           x[m]=c;
```

```
e[m].a=a,e[m].b=c,e[m].h=d,e[m++
].s=-1;//add the down edge
       }
       sort(x,x+m);//离散化x坐标,按
x坐标建树
       sort(e,e+m,cmp);
       int k=1;
       int L,R;
       for(int i=1;i<m;i++)
       {
          if(x[i]!=x[i-1])
x[k++]=x[i];
       }
       memset(cnt,0,sizeof(cnt));
       memset(sum,0,sizeof(sum));
       for(int i=0;i<m-1;i++)
       {
          L=bst(e[i].a,x,k);
          R=bst(e[i].b,x,k)-1;
          if(R>=L)
update(L,R,e[i].s,0,k-1,1);

ans+=sum[1]*(e[i+1].h-e[i].h);

//printf("%.2lf %.2lf\n",sum[1],
ans);
       }
       printf("Test case
#%d\nTotal explored
area: %.2lf\n\n",kase++,ans);
   }
   return 0;
}
```

圆的扫描线

```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<math.h>
#include<set>
#include<iostream>
#include<algorithm>
using namespace std;
const int maxn = 51111;
const double eps = 1e-8;
inline int dcmp(double x) {return
(x > eps) - (x < -eps);}
inline double Sqr(double x) {return
x * x;}
int LineNow, ltp, n, cnt[maxn];
struct Cir//圆
{
   int x;
   int y;
   int r;
}c[maxn];
struct Line//从左向右扫描节点
{
   int id;
   bool in;
   void Read(int id_, bool in_){id
= id_, in = in_;}
   inline int
GetSite()const{return c[id].x +
(in ? -c[id].r : c[id].r);}
   bool operator<(const Line
&b)const{return GetSite() <
b.GetSite();}
}l[maxn << 1];
struct Node//从上至下排序节点
{
   int id;
   bool up;
   Node(){}
   Node(int id_, bool up_){id = id_,
up = up_;}
   inline double GetSite()const
   {return c[id].y +
sqrt(Sqr(c[id].r) - Sqr(LineNow -
c[id].x)) * (up ? 1 : -1);}
   bool operator<(const Node
&b)const
   {
      double y1 = GetSite();
      double y2 = b.GetSite();
```

```
        return dcmp(y1 - y2) ? y1 > y2 : up > b.up;
    }
};
set<Node> s;//set<>用红黑树实现,在每次插入新节点时并不是重新进行排序,而是自根开始,运用比较规则进行排序
set<Node>::iterator iti, itn;
void ReadData(int n)
{
    int i;
    for(ltp = i = 0; i < n; ++ i)
    {
        scanf("%d%d%d", &c[i].x, &c[i].y, &c[i].r);
        l[ltp ++].Read(i, true);//圆的左端点
        l[ltp ++].Read(i, false);//圆的右端点
    }
}
int MakeAns()
{
    int i, ans = 0;
    sort(l, l + ltp);//每个圆的端点按照从左到右的顺序排序
    s.clear();
    for(i = 0; i < ltp; ++ i)
    {
        LineNow = l[i].GetSite();
        if(!l[i].in)
        {
            s.erase(Node(l[i].id, true));
            s.erase(Node(l[i].id, false));
        }
        else
        {
            iti = itn = s.insert(Node(l[i].id, true)).first;
            //此时iti和itn指向插入节点的位置
            itn ++;
            if(iti == s.begin() || itn == s.end()) cnt[l[i].id] = 1;//最外层的圆
            else
            {
                iti --;
                if((*iti).id == (*itn).id) cnt[l[i].id] = cnt[(*iti).id] + 1;//相交的两个交点在同一个圆上
                else cnt[l[i].id] = max(cnt[(*iti).id], cnt[(*itn).id]);//相交的两个交点在不同的圆上
            }
            ans = max(ans, cnt[l[i].id]);
            s.insert(Node(l[i].id, false));
        }
    }
    return ans;
}
int main()
{
    while(scanf("%d", &n) != EOF)
    {
        ReadData(n);
        printf("%d\n", MakeAns());
    }
    return 0;
}
```

BIT

```
//BIT
//1-D
const int maxn = 5e4 + 10;
int Tree[maxn],size;
inline int lowbit(int x){
    return (x&-x);
}
void add(int x,int value){
```

```
    for(int
i=x;i<=size;i+=lowbit(i))
        Tree[i]+=value;
}
int get(int x){
    int sum=0;
    for(int i=x;i;i-=lowbit(i))
        sum+=Tree[i];
    return sum;
}

//2-D
int mul_tree[maxn][maxn];
void add(int x,int y,int value)
{
    for(int i=x; i<=maxn;
i+=lowbit(i))
        for(int j=y; j<=maxn;
j+=lowbit(j))
            mul_tree[i][j]+=value;
}
int mul_get(int x,int y)
{
    int sum=0;
    for(int i=x; i; i-=lowbit(i))
        for(int j=y; j;
j-=lowbit(j))
            sum+=mul_tree[i][j];
    return sum;
}
```

线段树维护斜率优化凸包

```
#include <iostream>
#include <cstdio>
#include <vector>
#include <cstring>
#include <map>
#include <set>
#include <cmath>
#include <algorithm>

using namespace std;

int nextInt() {
    char c;
    int x = 0, p = 1;
    do {
        c = getchar();
    } while (c <= 32);
    if (c == '-') {
        p = -1;
        c = getchar();
    }
    while (c >= '0' && c <= '9') {
        x = x * 10 + c - '0';
        c = getchar();
    }
    return x * p;
}

const int maxn = 100500;

long double intersect(int k, int b,
int kk, int bb) {
    return (long double)(b - bb) /
(kk - k);
}

struct ConvexHull {
    int * k, * b;
    int len;
    ConvexHull() : k(0), b(0),
len(0) {}
    void addLine(int kk, int bb) {
        if (len == 1 && k[len - 1] ==
kk) {
            bb = min(b[len - 1], bb);
            len = 0;
        }
        if (len <= 1) {
            k[len] = kk;
            b[len] = bb;
            len++;
            return;
        }
```

```
        while (len >= 2 && ((k[len -
1] == kk && b[len - 1] > bb) || (kk !=
k[len - 1] && intersect(k[len - 2],
b[len - 2], k[len - 1], b[len -
1]) >= intersect(k[len - 1], b[len
- 1], kk, bb)))) len--;
        while (len >= 1 && k[len - 1]
== kk && b[len - 1] > bb) len--;
        if (len >= 1 && k[len - 1] ==
kk && b[len - 1] <= bb) return;
        k[len] = kk;
        b[len] = bb;
        len++;
    }
    int get(int idx, int x) {
        return k[idx] * x + b[idx];
    }
    bool f(int idx, int x) {
        return get(idx, x) >=
get(idx + 1, x);
    }
    int getMin(int x) {
        int l = -1, r = len - 1;
        while (r - l > 1) {
            int mid = (l + r) >> 1;
            if (f(mid, x)) l = mid;
            else r = mid;
        }
        return get(r, x);
    }
};

int n, q, a[maxn], s[maxn];
ConvexHull t[maxn * 4];

void mergeCHs(ConvexHull & res,
ConvexHull & a, ConvexHull & b) {
    res.len = 0;
    res.k = new int[a.len + b.len];
    res.b = new int[a.len + b.len];
    int l = 0, r = 0;
    while (l + r != a.len + b.len)
        if (l == a.len) {
            res.addLine(b.k[r],
b.b[r]);
            r++;
        }
        else if (r == b.len) {
            res.addLine(a.k[l],
a.b[l]);
            l++;
        }
        else if (a.k[l] > b.k[r]) {
            res.addLine(a.k[l],
a.b[l]);
            l++;
        }
        else {
            res.addLine(b.k[r],
b.b[r]);
            r++;
        }
}

void build(int v, int tl, int tr)
{
    if (tl == tr) {
        t[v].k = new int[1];
        t[v].b = new int[1];
        t[v].k[0] = a[tl];
        t[v].b[0] = a[tl] * tl -
s[tl];
        t[v].len = 1;
        return;
    }
    int tm = (tl + tr) >> 1;
    build((v << 1) + 1, tl, tm);
    build((v << 1) + 2, tm + 1, tr);
    mergeCHs(t[v], t[(v << 1) + 1],
t[(v << 1) + 2]);
}

int treeQuery(int v, int tl, int tr,
int l, int r, int x) {
    if (tl == l && tr == r) {
        return t[v].getMin(x);
    }
```

```
    int tm = (tl + tr) >> 1;
    if (r <= tm)
        return treeQuery((v << 1) +
1, tl, tm, l, r, x);
    else if (l > tm)
        return treeQuery((v << 1) +
2, tm + 1, tr, l, r, x);
    else
        return min(treeQuery((v <<
1) + 1, tl, tm, l, tm, x),
                   treeQuery((v << 1)
+ 2, tm + 1, tr, tm + 1, r, x));
}

int query(int x, int y) {
    int l = y - x + 1;
    int r = y;
    return treeQuery(0, 1, n, l, r,
x - y) + s[y];
}

int main()
{
    //freopen("input.txt", "r",
stdin);
    //freopen("output.txt", "w",
stdout);

    n = nextInt();

    for (int i = 1; i <= n; i++)
        a[i] = nextInt();

    for (int i = 1; i <= n; i++)
        s[i] = s[i - 1] + a[i];

    build(0, 1, n);

    q = nextInt();

    while (q--) {
        int x = nextInt();
        int y = nextInt();
        printf("%d\n", query(x,
y));
    }

    return 0;
}
```

判斷树上点 x 是否是点 y 的祖先

```
//利用 DFS 的遍歷順序,就可以輕鬆判斷一
點是不是另一點的祖先
bool adj[9][9];
int tin[9], tout[9];    // DFS進入
各點的時刻、離開各點的時刻
int t = 0;              // 現在時刻

void DFS(int x, int px) // px是x
的父親
{
    tin[x] = t++;

    for (int y=0; y<9; ++y)
        if (adj[x][y] && y != px)
            DFS(y, x);

    tout[x] = t++;
}

bool x_is_ancestor_of_y(int x, int
y)
{
    return tin[x] < tin[y] &&
tout[x] > tout[y];
}

void ancestor_descendant(int
root)
{
    t = 0;
    for (int i=0; i<9; ++i) tin[i]
= 0;
    DFS(root, root);

    int x, y;
```

```
    while (cin >> x >> y)
        if (x_is_ancestor_of_y(x,
y))
            cout << "x是y的祖先";
        else if
(x_is_ancestor_of_y(y, x))
            cout << "x是y的祖先";
        else
            cout << "xy不是祖孫關係";
}
```

字符串

SA

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <algorithm>
using namespace std;
//sa[]数组下标从0到len,其中sa[0]表示
排名最前的,即空串(最尾的串)
//height[]数组下标从1开始有意
义,height[i]表示sa[i]与sa[i-1]的
lcp
//使用方法见main函数
const int N = 100000 + 10;
char s[N];
int r[N], tx[N], ty[N], rs[N],
ranks[N], sa[N], height[N],
rmq[N][20]; //rs基数排序

bool cmp(int *r, int a, int b, int
len)
{
    return (r[a] == r[b]) && (r[a +
len] == r[b + len]);
}

void suffix(int n, int m) //n为长
度，最大值小于m
{
    int i, j, p, *x = tx, *y = ty,
*t;
    for(i = 0; i < m; ++i) rs[i] =
0;
    for(i = 0; i < n; ++i) { x[i] =
r[i]; ++rs[x[i]];}
    for(i = 1; i < m; ++i) rs[i] +=
rs[i - 1];
    for(i = n - 1; i >= 0; --i)
sa[--rs[x[i]]] = i;
    for(j = p = 1; p < n; j <<= 1,
m = p) {
        for(p = 0, i = n - j; i < n;
++i) y[p++] = i;
        for(i = 0; i < n; ++i)
{ if(sa[i] >= j) y[p++] = sa[i] -
j; }
        for(i = 0; i < m; ++i) rs[i]
= 0;
        for(i = 0; i < n; ++i)
++rs[x[y[i]]];
        for(i = 1; i < m; ++i) rs[i]
+= rs[i - 1];
        for(i = n - 1; i >= 0; --i)
sa[--rs[x[y[i]]]] = y[i];
        t = x, x = y, y = t;
        for(i = 1, p = 1, x[sa[0]] =
0; i < n; ++i) {
            if(cmp(y, sa[i - 1],
sa[i], j)) x[sa[i]] = p - 1;
            else x[sa[i]] = p++;
        }
    }
}

void calheight(int n)
{
    int i, j, k = 0;
    for(i = 1; i <= n; ++i)
        ranks[sa[i]] = i;
    for(i = 0; i < n; ++i) {
        if(k)
            --k;
```

```
        j = sa[ranks[i] - 1];
        while(r[i + k] == r[j + k])
            ++k;
        height[ranks[i]] = k;
    }
}

////////////////////////////////
////////////////////////////////
//////////rmq 求 lcp
int Log[N];
void initrmq(int n)
{
    int i, k;
    for(i = 2; i <= n; ++i)
        rmq[i][0] = height[i];
    for(k = 1; (1 << k) <= n; ++k)
{
        for(i = 2; i + (1 << k) - 1
<= n; ++i) {
            rmq[i][k] =
min(rmq[i][k - 1],
                        rmq[i + (1
<< (k - 1))][k - 1]);
        }
    }
}

void initlog()
{
    Log[0] = -1;
    for(int i=1;i<N;i++)

Log[i]=(i&(i-1))?Log[i-1]:Log[i-
1] + 1;
}

int lcp(int a, int b ,int n)//求a,b
的后缀的公用前缀长度，从0计
{
    if(a==b) return n-a;
    a = ranks[a], b = ranks[b];
    if(a > b)
        swap(a, b);
    ++a;
    int k = (int) Log[b - a + 1] /
Log[2];
    return min(rmq[a][k], rmq[b -
(1 << k) + 1][k]);
}

int main()
{
    initlog();
    while (scanf("%s", s) != EOF) {
        int len = (int)strlen(s);
        for (int i = 0; i < len; i ++)
{
            r[i] = s[i];
        }
        r[len] = 0;
        suffix(len + 1, 128);
        for (int i = 0; i <= len; i
++) {
            printf("%d ", sa[i]);
        }
        puts("");
        calheight(len);
        for (int i = 1; i <= len; i
++) {
            printf("%d ",
height[i]);
        }
        puts("");
    }
}
```

Trie 树

```
#include <cstdio>
#include <cstring>
#include <algorithm>

const int maxn = 100000 + 10;
const int C    = 26;

struct Trie
{
```

```
    Trie *ch[C];

    Trie()
    {
        memset(ch, 0, sizeof(ch));
    }

    void add(char *s)
    {
        if (*s == '\0') {
            return;
        }
        if (!ch[*s - 'a'])
        {
            ch[*s - 'a'] = new
Trie();
        }
        ch[*s - 'a'] -> add(s + 1);
    }

} *root;

char s[maxn];

int main()
{
    root = new Trie();
    scanf("%s", s);
    root -> add(s);
    return 0;
}
```

bkdr_hash

```
//bkdrhash
//mod = 4000037
unsigned int BKDRHash(string s)
{
    unsigned int seed = 131; // 31
131 1313 13131 131313 etc..
    unsigned int hash = 0;
    int len = (int)s.length();
    for (int i = 0; i < len; i ++)
{
        hash = hash * seed + s[i];
    }
    return (hash & 0x7FFFFFFF);
}
```

计算几何

普通算法

```
#include <algorithm>
#include <iostream>
#include <iomanip>
#include <complex>
#include <cstring>
#include <cstdlib>
#include <string>
#include <vector>
#include <cstdio>
#include <cmath>
#include <queue>
#include <stack>
#include <map>
#include <set>
#define cntbit __builtin_popcount
using namespace std;
//#pragma
comment(linker,"/STACK:102400000
,102400000")

const double EPS = 1e-8;
const int   maxn = 100000;
int sign(double x)
{
    if (x < - EPS) {
        return -1;
    }
    if (x > EPS) {
        return 1;
    }
    return 0;
}
```

```
struct point
{
    double x, y;
    point(){}
    point(double x, double y)
    {
        this -> x = x, this -> y = y;
    }
    point operator + (const point
&p)
    {
        return point(this -> x + p.x,
this -> y + p.y);
    }
    point operator - (const point
&p)
    {
        return point(this -> x - p.x,
this -> y - p.y);
    }
    double operator * (const point
&p)
    {
        return this -> x * p.x + this
-> y * p.y;
    }
    double dot(const point &p)
    {
        return this -> x * p.y - p.x
* this -> y;
    }
    double len()
    {
        return sqrt(this -> x * this
-> x + this -> y * this -> y);
    }
};


double multi (point a, point b,
point c)    //叉积判断点线关系

{
    double x1, y1, x2, y2;

    x1 = b.x - a.x;

    y1 = b.y - a.y;

    x2 = c.x - b.x;

    y2 = c.y - b.y;

    return x1*y2 - x2*y1;

}

bool judge_intersect (point a,
point b, point c, point d)    //判
断两线段是否相交

{

    if ((max (a.x, b.x) + EPS) >= min
(c.x, d.x) &&    //快速排斥试验

        (max (c.x, d.x) + EPS) >= min
(a.x, b.x) &&

        (max (a.y, b.y) + EPS) >= min
(c.y, d.y) &&

        (max (c.y, d.y) + EPS) >= min
(a.y, b.y) &&

        sign(multi (a, c, d)*multi
(b, c, d)) <= 0 &&    //跨立试验

        sign(multi (c, a, b)*multi
(d, a, b)) <= 0)

        return true;

    return false;

}
```

```
//判断点C是否在线段AB上
//true代表在,false代表不在
bool judge_in(point A, point B,
point C)
{
    if ((C.x - A.x) * (C.x - B.x) <=
0 && (C.y - A.y) * (C.y - B.y) <=
0) {
        point AC = C - A, BC = C - B;
        if (AC.dot(BC) == 0) {
            return true;
        }
        else
        {
            return false;
        }
    }
    return false;
}

//凸包graham法
int n;

bool cmp(const P &p,const P &q) {
    if (p.x == q.x)  return p.y<q.y;
    return p.x < q.x;
}

vector<P> convex_hull(P* ps) {
    sort(ps, ps + n, cmp);
    int k = 0;
    vector<P> qs(n*2);
    for (int i = 0; i < n; i ++){
        while ( k > 1 && (qs[k-1] -
qs[k-2]).det(ps[i] - qs[k-1]) <= 0)
k--;
        qs[k++] = ps[i];
    }
    for (int i = n-2, t = k; i >=
0;i--){

while(k>t&&(qs[k-1]-qs[k-2]).det
(ps[i]-qs[k-1])<=0)  k--;
        qs[k++]=ps[i];
    }
    qs.resize(k-1);
    return qs;
}

point a[maxn];
int   s[maxn];
//平面最近点对
bool cmpx(const point& a,const
point& b){
    return a.x<b.x;
}
bool cmpy(const point& a,const
point& b){
    return a.y<b.y;
}
double min_dis(point a[],int
s[],int l,int r){
    double ans=1e100;
    if(r-l<20){
        for(int q=l;q<r;q++)
            for(int w=q+1;w<r;w++)

ans=min(ans,sqrt((a[q].x-a[w].x)
*(a[q].x-a[w].x)+(a[q].y-a[w].y)
*(a[q].y-a[w].y)));
        return ans;
    }
    int tl,tr,m=(l+r)/2;

ans=min(min_dis(a,s,l,m),min_dis
(a,s,m,r));

for(tl=l;a[s[tl]].x<a[s[m]].x-an
s;tl++);

for(tr=r-1;a[s[tr]].x>a[s[m]].x+
ans;tr--);
    sort(a+s[tl],a+s[tr],cmpy);
    for(int q=tl;q<tr;q++)
        for(int
w=q+1;w<min(tr,q+6);w++)

ans=min(ans,sqrt((a[q].x-a[w].x)
```

```
*(a[q].x-a[w].x)+(a[q].y-a[w].y)
*(a[q].y-a[w].y)));
    sort(a+s[tl],a+s[tr],cmpx);
    return ans;
}
double Min_dis(point a[],int
s[],int n){
    for(int i=0;i<n;i++)  s[i]=i;
    sort(a,a+n,cmpx);
    return min_dis(a,s,0,n);
}

//空间最近点对
bool cmpz(const point& p,const
point& q){
    return p.z<q.z;
}
bool cmpy(const point& p,const
point& q){
    return p.y<q.y;
}
double cal(point p,point q){
    return
sqrt((p.x-q.x)*(p.x-q.x)+(p.y-q.
y)*(p.y-q.y)+(p.z-q.z)*(p.z-q.z)
);
}
double min_dis(point a[],int
s[],int l,int r){
    double ans=1e100;
    if(r-l<40){
        for(int q=l;q<r;q++)
            for(int w=q+1;w<r;w++)
ans=min(ans,cal(a[q],a[w]));
        return ans;
    }
    int tl,tr,m=(l+r)/2;

ans=min(min_dis(a,s,l,m),min_dis
(a,s,m,r));

for(tl=l;a[s[tl]].z<a[s[m]].z-an
s;tl++);

for(tr=r-1;a[s[tr]].z>a[s[m]].z+
ans;tr--);
    sort(a+s[tl],a+s[tr],cmpy);
    for(int q=tl;q<tr;q++)
        for(int
w=q+1;w<min(tr,q+16);w++)

ans=min(ans,cal(a[q],a[w]));
    sort(a+s[tl],a+s[tr],cmpz);
    return ans;
}
double Min_dis(point a[],int
s[],int n){
    for(int i=0;i<n;i++)  s[i]=i;
    sort(a,a+n,cmpz);
    return min_dis(a,s,0,n);
}
```

平面最小圆覆盖

```
#include "cstdio"
#include "cmath"
#include "iostream"
using namespace std;
#define EPS 1e-10
struct point{
    double x,y;
};
point save[111111];
int n;
void circle_centre3(point
p0,point p1,point p2,point &cp){
    double
a1=p1.x-p0.x,b1=p1.y-p0.y,c1=(a1
*a1+b1*b1)/2.;
    double
a2=p2.x-p0.x,b2=p2.y-p0.y,c2=(a2
*a2+b2*b2)/2.;
    double d=a1*b2-a2*b1;
    cp.x=p0.x+(c1*b2-c2*b1)/d;
    cp.y=p0.y+(a1*c2-a2*c1)/d;
}
```

```
void circle_centre2(point
p0,point p1,point &cp){
    cp.x=(p0.x+p1.x)/2.;
    cp.y=(p0.y+p1.y)/2.;
}
point centre;
double radius;
double dist(point p,point q){
    return
sqrt((p.x-q.x)*(p.x-q.x)+(p.y-q.
y)*(p.y-q.y));
}
void min_circle(){
    radius=0;
    centre=save[0];
    for(int i=1;i<n;i++)
if(dist(save[i],centre)+EPS>radi
us){
        centre=save[i],radius=0;
        for(int j=0;j<i;j++)
if(dist(save[j],centre)+EPS>radi
us){

circle_centre2(save[i],save[j],c
entre);

radius=dist(save[j],centre);
            for(int k=0;k<j;k++)
if(dist(save[k],centre)+EPS>radi
us){

circle_centre3(save[i],save[j],s
ave[k],centre);

radius=dist(save[k],centre);
            }
        }
    }
    printf("%.2lf\n",radius);
}
int main(){

while(scanf("%d",&n)!=EOF&&n){
        for(int i=0;i<n;i++)
scanf("%lf%lf",&save[i].x,&save[
i].y);

random_shuffle(save,save+n);
        min_circle();
    }
    return 0;
}
```

点是否在多边形内

极角排序后二分解决

FFT

```
#include <algorithm>
#include <cmath>
#include <cstdio>
#include <cstring>

struct Complex
{
    double a,b;
    Complex(double a=0.0,double
b=0.0)
    {
        this->a = a;
        this->b = b;
    }
    Complex operator +(const
Complex &o)
    {
        return
Complex(a+o.a,b+o.b);
    }
    Complex operator -(const
Complex &o)
    {
        return
Complex(a-o.a,b-o.b);
```

```
    }
    Complex operator *(const
Complex &o)
    {
        return
Complex(a*o.a-b*o.b,a*o.b+b*o.a)
;
    }
    Complex operator *(double v)
    {
        return Complex(a*v,b*v);
    }
    Complex operator /(double v)
    {
        return Complex(a/v,b/v);
    }
    double getreal()
    {
        return a;
    }
};

const double PI = acos(-1.);

void FFT(Complex* P, int n, int
oper)
{
    for (int i = 1, j = 0; i < n -
1; i++) {
        for (int s = n; j ^= s >>= 1,
~j & s;);
        if (i < j) {
            std::swap(P[i], P[j]);
        }
    }
    Complex unit_p0;
    for (int d = 0; (1 << d) < n; d++)
{
        int m = 1 << d, m2 = m * 2;
        double p0 = PI / m * oper;
        unit_p0 = Complex(cos(p0),
sin(p0));
        for (int i = 0; i < n; i +=
m2) {
            Complex unit = 1;
            for (int j = 0; j < m; j++)
{
                Complex &P1 = P[i + j
+ m], &P2 = P[i + j];
                Complex t = unit * P1;
                P1 = P2 - t;
                P2 = P2 + t;
                unit = unit *
unit_p0;
            }
        }
    }
}
```

输入输出加速器

```
template <class T>
inline bool scan_d(T &ret)
{
    char c;
    int sgn;
    if (c = getchar(), c == EOF)
        return 0; //EOF
    while (c != '-' && ( c < '0' ||
c > '9' ))
        c = getchar();
    sgn = (c == '-') ? -1 : 1;
    ret = (c == '-') ? 0 : (c - '0');
    while (c = getchar(), c >= '0'
&& c <= '9')
        ret = ret * 10 + (c - '0');
    ret *= sgn;
    return 1;
}
inline void out(int x)
{
    if (x > 9)
        out(x / 10);
    putchar(x % 10 + '0');
}

import java.io.*;
```

```
import java.math.*;
import java.util.*;

public class Main {
    public static BigInteger seven
= new BigInteger("7");
    public static BigInteger eight
= new BigInteger("8");
    public static BigInteger four =
new BigInteger("4");
    public static BigInteger one =
BigInteger.ONE;
    public static void main(String[]
args)                       throws
NumberFormatException,
IOException {
        BufferedReader  in  =  new
BufferedReader(new
InputStreamReader(System.in));
        PrintWriter   out   =   new
PrintWriter(System.out);

        int T;
        BigInteger n;

        T           =           new
Integer(in.readLine());
        for (int kase = 1; kase <= T;
kase++) {
            n         =         new
BigInteger(in.readLine());
            out.println("Case  #"  +
kase      +      ":      "      +
n.multiply(eight.multiply(n).sub
tract(seven)).add(one));
        }
        out.close();
    }
}
```

STL

去重:

```
sort(res.begin(),
res.end());
vector<int>::iterator iter =
unique(res.begin(),
res.end());
res.erase(iter, res.end());
```

set 遍历方法:

```
set<int> :: iterator it;
for (it = s.begin(); it !=
s.end(); it ++) {
    cout<<*it<<endl;
}
```