Notes **for** ACMICPC World Finals 2015

ACMICPC World Finals 2015 **参考资料**

Chinese Edition **中文版**

Beijing Jiao Tong University : **Hyacinth**

Coach	教　练
Hua HUANG	黄　华

Team member	队　员
YanBin KANG	康燕斌
Shu WANG	王　戍
ShiYing LUO	骆石英

目录


```
set nocompatible sta hls wrap ruler cindent nu nobackup noswapfile
autoindent ts=4 noet sts=4 sw=4
syntax on
autocmd FileType c,cpp nmap <F8> <ESC>:w <CR><ESC>:!g++ % -o %< <CR>
autocmd FileType c,cpp nmap <F9> :!time ./%< <./%<.in <CR>
autocmd FileType c,cpp nmap <F10> :!time ./%< <CR>
nmap <F2> :vs %<.in <CR>
```

KM

```
typedef double ValueType;//结点0~n-1
const int maxn=200; const ValueType MOD=(1e20);
ValueType x[maxn], y[maxn], w[maxn][maxn], slack[maxn];
int sy[maxn], px[maxn], py[maxn], par[maxn];
int pa[200][2],pb[200][2],n0,m0,na,nb;
char s[200][200], n;
void adjust(int v){ sy[v]=py[v]; if(px[sy[v]]!=-2) adjust(px[sy[v]]);}
bool find(int v){for (int i=0;i<n;i++)
    if (py[i]==-1){
        if (slack[i]>x[v]+y[i]-w[v][i]){
            slack[i]=x[v]+y[i]-w[v][i];par[i]=v;}
        if (x[v]+y[i]==w[v][i]){
            py[i]=v; if (sy[i]==-1){adjust(i);  return 1;}
            if (px[sy[i]]!=-1) continue; px[sy[i]]=i;
            if (find(sy[i])) return 1; } } return 0; }
ValueType km(){ int i,j; ValueType m;
    for (i=0;i<n;i++) sy[i]=-1,y[i]=0;
    for (i=0;i<n;i++) { x[i]=-MOD;
        for (j=0;j<n;j++) x[i]=max(x[i],w[i][j]); }
    bool flag;
    for (i=0;i<n;i++){
        for (j=0;j<n;j++) px[j]=py[j]=-1,slack[j]=MOD;
        px[i]=-2; if (find(i)) continue; flag=false;
        for (;!flag;){ m=MOD;
            for (j=0;j<n;j++) if (py[j]==-1) m=min(m,slack[j]);
            for (j=0;j<n;j++){if (px[j]!=-1) x[j]-=m;
                if (py[j]!=-1) y[j]+=m; else slack[j]-=m; }
            for (j=0;j<n;j++){
                if (py[j]==-1&&!slack[j]){ py[j]=par[j];
                    if (sy[j]==-1){adjust(j); flag=true; break; }
                    px[sy[j]]=j; if (find(sy[j])){flag=true;break;}
                } } } } ValueType ans=0;
    for (i=0;i<n;i++) ans+=w[sy[i]][i]; return ans; }
```

Hopcroft

```
int n, m, match = 0; queue<int> Q;
int mx[Maxn], my[Maxn], dx[Maxn], dy[Maxn], dis, visit[Maxn];
int ux[Maxn], uy[Maxn], px[Maxn], py[Maxn], pv[Maxn];
void adde(int u, int v) {}
bool searchPath() {
    int i, j, u, v; dis = MOD; for(i = 0; i < n; i++) dx[i] = -1;
    for(j= n; j< n + m;j++) dy[j]= -1;while(!Q.empty()) Q.pop();
    for(i = 0; i < n; i++) if(-1 == mx[i]) Q.push(i);
    while(!Q.empty()) {
        u = Q.front(); Q.pop(); if(dx[u] > dis) break;
        for(j = last[u]; j != -1; j = e[j].next) {
            v = e[j].v; if(-1 != dy[v]) continue; dy[v] = dx[u] + 1;
            if(-1 == my[v]) dis = dy[v];
            else {dx[my[v]] = dy[v] + 1; Q.push(my[v]);} } }
    return dis != MOD; }
bool dfs(int u){ int v;
    for(int j = last[u]; j != -1; j = e[j].next) {
        v = e[j].v; if(visit[v] || dx[u] + 1 != dy[v]) continue;
        if(dy[v] == dis && my[v] != -1) continue; visit[v] = true;
        if(-1 == my[v] || dfs(my[v])){my[v] = u; mx[u] = v;return true;} }
return false; }
int solve(){int i,j; match = 0;for(i = 0; i < n; i++) mx[i] = -1;
    for(j = n; j < n + m; j++) my[j] = -1;
    while(searchPath()){for(j = n;j < n + m;j++)visit[j] = false;
        for(int i= 0; i< n; i++)if(-1== mx[i] && dfs(i))match++;}
    return match; }
```

稳定婚姻匹配

```
//延迟认可算法(Gale-Shapley算法)
```

```
int mx[Maxn], my[Maxm], cur[Maxn], n, m; //x匹配的y, y匹配的x
//yorder表示在x眼中y的顺序,0~m-1为喜爱度递减的y的编号
//xorder表示在y眼中x的顺序,0~n-1为编号0~n-1的x的重要度,越重要, 值越小
int yorder[Maxn][Maxm], xorder[Maxm][Maxn]; queue<int> que;
void GaleShapley() { int i, j, v;
    for(i = 0; i <= n; i++) mx[i] = -1,cur[i] = 0;//初始化
    for(j = 0; j <= m; j++) my[j] = -1; while(!que.empty()) que.pop();
    for(i = 0; i < n; i++) que.push(i);//将x加入队列
    while(!que.empty()) {//x还有没找到朋友的
        i = que.front(); que.pop(); if(cur[i] >= m) continue;
        v = yorder[i][cur[i]++];
        if(my[v] == -1) {mx[i] = v; my[v] = i; } //y没有匹配
        else if(xorder[v][i] < xorder[v][my[v]]) {//i比之前的好
            mx[my[v]] = -1; que.push(my[v]); my[v] = i; mx[i] = v; }
        else que.push(i);//i比以前的差, i找下一个y
    }/*while*/  }/*func*/
```

一般图最大匹配

```
int n, head, tail, Start, Finish, Q[Maxn], mark[Maxn];
int InBlossom[Maxn], inqueue[Maxn];int match[Maxn];//表示哪个点匹配了哪个点
int father[Maxn];//这个是增广路径的father
int base[Maxn];//该点属于哪朵花 bool mp[Maxn][Maxn]; //邻接关系
void BlossomContract(int x,int y){memset(mark, false, sizeof(mark));
    memset(InBlossom, false, sizeof(InBlossom));
#define pre father[match[i]]
    int lca, i; for( i = x; i; i = pre) {i = base[i]; mark[i] = true; }
    for(i = y; i; i = pre) {i = base[i]; //寻找lca
        if(mark[i]) {lca = i; break; } }
    for(i = x; base[i] != lca; i = pre) {
        if(base[pre] != lca) father[pre] = match[i];
        //对于BFS树中的父边是匹配边的点, father向后跳
        InBlossom[base[i]] = true; InBlossom[base[match[i]]] = true; }
    for(i = y; base[i] != lca; i = pre) {
        if(base[pre] != lca) father[pre] = match[i]; // 同理
        InBlossom[base[i]] = true; InBlossom[base[match[i]]] = true; }
#undef pre
    if(base[x] != lca) father[x] = y; if(base[y] != lca) father[y] = x;
    for(i = 1; i <= n; i++) if(InBlossom[base[i]]) { base[i] = lca;
        if(!inqueue[i]) {Q[tail++] = i; inqueue[i] = true; } } }
void Change() { int x, y, z = Finish;
    while(z){y= father[z];x= match[y];match[y] = z;match[z] = y; z = x;} }
void FindAugmentPath() { int i; memset(father, 0, sizeof(father));
    memset(inqueue, false, sizeof(inqueue));
    for(i = 1; i <= n; i++) base[i] = i;
    head = tail = 0; Q[tail++] = Start; inqueue[Start] = 1;
    while(head < tail) { int x = Q[head++];
        for(int y = 1; y <= n; y++)
            if(mp[x][y] && base[x] != base[y] && match[x] != y) {
                if(Start == y || match[y] && father[match[y]])
                    BlossomContract(x, y);
                else if(!father[y]) { father[y] = x;
                   if(match[y]){Q[tail++]=match[y];inqueue[match[y]]= true;}

                    else {Finish = y; Change(); return; } } } } }
void Edmonds() { memset(match, 0, sizeof(match));
    for(Start=1;Start<= n;Start++)if(match[Start]== 0) FindAugmentPath();}
void output() { memset(mark, false, sizeof(mark));
    int i, cnt = 0;for(i = 1; i <= n; i++) if(match[i]) cnt++;
/*  printf("%d\n", cnt);          //输出匹配关系
    for(int i = 1; i <= n; i++) {
        if(!mark[i] && match[i]) { mark[i] = true;
            mark[match[i]] = true; printf("%d %d\n", i, match[i]);}} //*/
    if(cnt < n)printf("NO\n");else printf("YES\n"); }
```

LCA

```
//倍增算法加边之前使用initLCA()初始化数组
//调用solveLCA()初始化LCA, 调用getLCA(x,y)返回x和y的LCA
#define STEP 17
void initLCA() {}//初始化tot, last, depth, fa[][]
void dfsLCA(int u) {}//确定结点深度depth[]和fa[0][u]
int getLCA (int x, int y) { int i, dif = abs(depth[x] - depth[y]);
    if (depth[x] < depth[y]) swap(x, y);
    for (i = STEP - 1; i >= 0; i--) {
        if ((1 << i) & dif) { dif -= (1 << i); x = fa[i][x]; }}
    for (i = STEP - 1; i >= 0; i--) {
        if (fa[i][x] != fa[i][y]) {x = fa[i][x]; y = fa[i][y]; }}
    if (x == y) return x; else return fa[0][x]; }
void solveLCA(){int i, j, root = 1;for(i = 0; i <= n; i++) depth[i] = -1;
    fa[0][root] = root; depth[root] = 0; dfsLCA(root);
    for (i = 1; i < STEP; i++) for(j = 0; j <= n; j++)
            fa[i][j] = fa[i-1][fa[i-1][j]]; }
//Tarjan离线LCA, 利用struct Graph, e存放树边, q存放query
struct GRAPH {}e, q;
void tarjanLCA(int u) { int i, j, v, f; fa[u] = u; visit[u] = 1;
    for(j = e.last[u]; j != -1; j = e.adj[j].next) { v = e.adj[j].v;
        if(!visit[v]) { tarjanLCA(v); fa[v] = u; } }
    for(j = q.last[u]; j != -1; j = q.adj[j].next) { v = q.adj[j].v;
        if(visit[v]) {lca[q.adj[j].n] = f = getfa(v);
            ans[q.adj[j].n] = dist[v] + dist[u] - 2 * dist[f]; } } }
//LCA转RMQ
graph: 1-2, 1- 7, 2-3, 2-4, 4-5, 5-6, 7-8
step 1: dfs遍历树, 依次记录每次到达的点, 以及每个点的深度得到序列:
    结点访问顺序是: 1 2 3 2 4 5 4 6 4 2 1 7 8 7 1 //共2n-1个值
    结点对应深度是: 0 1 2 1 2 3 2 3 2 1 0 1 2 1 0
step 2: 利用ST计算任意区间最小深度的点的ID
step 3: 对于每次查询, 查询u第一次出现位置到v第一次出现位置区间的最小值
```

点双连通(重建图)

```
int dfn[Maxn], low[Maxn], iscut[Maxn], belong[Maxm], color[Maxm];
int lcnt[Maxm], nindex, ncnt, visit[Maxm], n, m, q;
stack<int> sta; set<PII> S;
void gao(int u) {}//遍历所有与u连通的点,标记visit[v]从-1变为1
void newAdde(int u, int v) {if(u > v) swap(u, v); PII ss = MP(u, v);
    if(S.find(ss) == S.ED) {S.insert(ss);e1.adde(u, v);e1.adde(v, u);}}
void Tarjan(int u,int from){int v, child = 0; dfn[u] = low[u] = ++nindex;
    for(int j = e.last[u]; j != -1; j = e.adj[j].next) {
        if(j == from) continue; v = e.adj[j].v;
        if(dfn[v] < dfn[u]) { sta.push(j);
            if(!dfn[v]){child++;Tarjan(v,j^ 1);low[u]= min(low[u],low[v]);
                if(low[v] >= dfn[u]) { ncnt++;
                    while(sta.top()!=j){belong[sta.top()/2]=ncnt;sta.pop();}
                    belong[j / 2] = ncnt; sta.pop(); iscut[u] = 1; }
            } else low[u] = min(low[u], dfn[v]); } }/*for*/
    if(from < 0 && child == 1)iscut[u] = -1; }//child
void buildGraph() {int i, j, u, v, x, y; e1.init(ncnt + 10); S.clear();
    for(j = 0; j < e.tot; j += 2) {u = e.adj[j].u; v = e.adj[j].v;
        if(iscut[u] == -1 && iscut[v] == -1) continue;
        else {if(iscut[u]!=-1){x=iscut[u];y = belong[j/2];newAdde(x,y);}
            if(iscut[v]!=-1){x=iscut[v];y=belong[j/2];newAdde(x,y);}}}
    for(i = 0; i <= ncnt; i++) visit[i] = -1;
    for(i = 1; i <= ncnt; i++) {
        if(visit[i] == -1) {visit[i] = 1;
            gao(i); e1.adde(0, i); e1.adde(i, 0);}}/*for*/ }/*func*/
void solve() {int i, j; memset(dfn, 0, sizeof(dfn)); ncnt = nindex = 0;
    /*memset low->0, iscur->-1, color->0, belong->-1*/
    for(i = 1; i <= n; i++) {
        if(!dfn[i]) {while(!sta.empty()) sta.pop();Tarjan(i, -1); } }
    for(i = 1; i <= n; i++) {
```

```
        if(iscut[i] == 1) {color[++ncnt] = 1;iscut[i] = ncnt; }
    }/*for*/ buildGraph(); }
```

dinic

```
bool bfs(int s, int t, int n) {}//注意对边.c!=0的判断
int dinic(int s, int t, int n) {int i, j, u, v; int maxflow = 0;
    while(bfs(s, t, n)) {for(i = 0; i < n; i++) cur[i] = last[i];
        u = s; top = 0;
        while(cur[s] != -1) {
            if(u == t) {int tp = MOD;//tp最小值,修改流量,修改top
                u = e[sta[top]].u; }
            else if(cur[u] != -1 && e[cur[u]].c > 0 && dist[u] + 1 ==
dist[e[cur[u]].v]) { sta[top++] = cur[u]; u = e[cur[u]].v; }
            else {while(u != s && cur[u] == -1)u = e[sta[--top]].u;
                cur[u] = e[cur[u]].next;   }
        }/*while(cur)*/  }/*while bfs*/ return maxflow; }
```

费用流

```
typedef int ValueType; deque<int>Q; const ValueTyep MOD=0x3f3f3f3f3f3f3fLL;
ValueType flow, cost, value, dist[Maxn];
int visit[Maxn], src, des;//注意全局变量 src,des 必须初始化
void adde(int u, int v, ValueType c, ValueType w) {}
ValueType Aug(int u, ValueType m) {
    if(u == des) { cost += value * m; flow += m; return m; }
    visit[u] = true; int j, v; ValueType l = m, c, w, del;
    for(j = last[u]; j != -1; j = e[j].next) {
        v = e[j].v; c = e[j].c; w = e[j].w;
        if(c && !w && !visit[v]) { del = Aug(v, l < c ? l : c);
            e[j].c -= del; e[j ^ 1].c += del; l -= del;if(!l) return m; } }
    return m - l; }
bool Modlabel(int src, int des, int n){int i, j, u, v; ValueType c, w, del;
    memset(dist, 0x3f, sizeof(dist[0])*(n + 3));
    while(!Q.empty()) Q.pop_back(); dist[src] = 0; Q.push_back(src);
    while(!Q.empty()) { u = Q.front(); Q.pop_front();
        for(j = last[u]; j != -1; j = e[j].next) {
            v = e[j].v; c = e[j].c; w = e[j].w;
            if(c && (del = dist[u] + w) < dist[v]) { dist[v] = del;
                if(Q.empty() || del <= dist[Q.front()]) Q.push_front(v);
                else Q.push_back(v);  } } }
    for(i = 0; i < n; i++) {
        for(j=last[i];j!=-1; j = e[j].next)e[j].w -= dist[e[j].v] - dist[i];}
    value += dist[des]; return dist[des] < MOD; }
void zkw(int src, int des, int n) { value = cost = flow = 0;
    while(Modlabel(src, des, n)){
        do{memset(visit,0,sizeof(visit[0])*(n+3));}while(Aug(src,MOD)); } }
```

无向图最小割

```
typedef int ValueType;//K连通块计数, 注意节点下标0~n-1
ValueType edge[Maxn][Maxn], g[Maxn][Maxn], minCut, maxi;
int n, m, k, S, T, top, sta[Maxn], comb[Maxn], node[Maxn];
vector<int> parta, partb, belong[Maxn];
ValueType Search (int n) {int i, j, u, vis[Maxn];
    ValueType wet[Maxn],minCut= 0,maxi;int temp= -1,top= 0;S= -1,T= -1;
    memset(vis, 0, sizeof(vis)); memset(wet, 0, sizeof(wet));
    for (i=0; i< n; i++) { maxi = -MOD;
        for (j = 0; j < n; j++) { u = node[j];
            if(!comb[u]&& !vis[u] && wet[u]> maxi){temp= u;maxi= wet[u];}}
        sta[top++] = temp;vis[temp] = true;if(i == n - 1) minCut = maxi;
        for (j = 0; j < n; j++) { u = node[j];
            if (!comb[u] && !vis[u]) wet[u] += edge[temp][u]; } }
    S = sta[top - 2]; T = sta[top - 1];
    for (i = 0; i < top; i++)  node[i] = sta[i]; return minCut; }
ValueType StoerWagner (vector<int> & li) {
    int i, j, k, n = li.SZ, u, v, used[Maxn];
    ValueType cur, ans = MOD; memset(comb, 0, sizeof(comb));
    for (i = 0; i < n; i++){node[i] = i;
belong[i].clear();belong[i].PB(i);}
```

```
    for (i = 1; i < n; i++) {k = n - i + 1; cur = Search(k);
        if (cur < ans) { ans = cur; for(j = 0; j < n; j++) used[j] = 0;
            for(j = 0; j < belong[T].SZ; j++) used[belong[T][j]] = 1; }
        for(j = 0; j < belong[T].SZ; j++) belong[S].PB(belong[T][j]);
        if (ans == 0) break; comb[T] = true;
        for (j = 0; j < n; j++) { if (j == S) continue;
            if (!comb[j]) {edge[S][j] += edge[T][j];
                edge[j][S] += edge[j][T]; } } }
    parta.clear(); partb.clear();
    for(j = 0; j < n; j++) {if(used[j]) parta.PB(li[j]);
        else partb.PB(li[j]); } return ans; }
int dfs(vector<int> &li) {int n = li.SZ, i, j;
for(i = 0; i < n; i++) for(j = 0; j < n; j++)
edge[i][j] = g[li[i]][li[j]];
    ValueType cur = StoerWagner(li); if(cur >= k) return 1;
    vector<int> a(parta), b(partb); return dfs(a) + dfs(b); }
```

<u>有向图最小生成树</u>

```
/* O(VE),根不固定,添加一个根节点与所有点连无穷大的边！
 * 如果求出比2*MOD大, 则不连通; 根和虚拟根相连的结点
 * 根据pre的信息能构造出这棵树! 注意结点必须从0~n-1*/
typedef int ValueType; ValueType inv[Maxn];
int visit[Maxn], pre[Maxn], belong[Maxn], ROOT;
ValueType dirtree(int n, int m, int root) {
    ValueType sum = 0; int i, j, k, u, v;
    while (1) {
        for (i = 0; i < n; i++) {
            inv[i] = MOD; pre[i] = -1; belong[i] = -1; visit[i] = -1; }
        inv[root] = 0; //除原点外,找每个点的最小入边
        for (i = 0; i < m; i++) {  u = e[i].u; v = e[i].v;
            if (u != v) {
                if (e[i].w < inv[v]) { inv[v] = e[i].w;  pre[v] = u;
                    if(u == root) ROOT = i; //记录根所在的边,输出根时利用ROOT-
m计算是原图哪个结点
                }/*if*/ }/*if*/ }/*for*/
        for (i = 0; i < n; i++) if (inv[i] == MOD) return -1; int num = 0;
        for (i = 0; i < n; i++) { //找圈,收缩圈
            if (visit[i] == -1) { j = i;
                for(j = i; j != -1 && visit[j] == -1 && j != root; j =
pre[j]) visit[j] = i;
                if (j != -1 && visit[j] == i) {
                    for (k = pre[j]; k != j; k = pre[k]) belong[k] = num;
                    belong[j] = num ++ ; } }  sum += inv[i]; }
        if (num == 0) return sum;
        for (i = 0; i < n; i++)if (belong[i] == -1) belong[i] = num++;
        for (i = 0; i < m; i++) { //重新构图
            e[i].w = e[i].w - inv[e[i].v]; e[i].v = belong[e[i].v];
            e[i].u = belong[e[i].u]; }
        n = num; root = belong[root]; } }
```

<u>最大团搜索算法</u>

```
Int g[][]为图的邻接矩阵. MC(V)表示点集V的最大团
令Si={vi, vi+1, ..., vn}, mc[i]表示MC(Si). 倒着算mc[i],那么显然MC(V)=mc[1]
此外有mc[i]=mc[i+1] or mc[i]=mc[i+1]+1
void init(){ int i, j;for (i=1; i<=n; ++i)
    for (j=1; j<=n; ++j) scanf("%d", &g[i][j]); }
void dfs(int size){int i, j, k;
    if (len[size]==0) {  if (size>ans) {  ans=size; found=true;} return;}
    for (k=0; k<len[size] && !found; ++k) {
        if (size+len[size]-k<=ans) break;
        i=list[size][k]; if (size+mc[i]<=ans) break;//第size个点选择点i
        for (j=k+1, len[size+1]=0; j<len[size]; ++j)
        if (g[i][list[size][j]])
list[size+1][len[size+1]++]=list[size][j];
        dfs(size+1);}}
```

```
void work(){ int i, j;    mc[n]=ans=1;
    for (i=n-1; i; --i) {found=false; len[1]=0;
        for (j=i+1; j<=n; ++j) if (g[i][j]) list[1][len[1]++]=j;
        dfs(1);  mc[i]=ans;}}
```

极大团的计数

bool g[][] 为图的邻接矩阵,图点的标号由1至n.

```
void dfs(int size){int i, j, k, t, cnt, best = 0;  bool bb;
    if (ne[size]==ce[size]){if (ce[size]==0) ++ans;return;}
    for (t=0, i=1; i<=ne[size]; ++i) {
        for (cnt=0, j=ne[size]+1; j<=ce[size]; ++j)
        if (!g[list[size][i]][list[size][j]]) ++cnt;
        if (t==0 || cnt<best) t=i, best=cnt; }
    if (t && best<=0) return;
    for (k=ne[size]+1; k<=ce[size]; ++k) {
        if (t>0){
            for (i=k; i<=ce[size]; ++i)
                if (!g[list[size][t]][list[size][i]]) break;
            swap(list[size][k], list[size][i]); }
        i=list[size][k]; ne[size+1]=ce[size+1]=0;
        for (j=1; j<k; ++j)if (g[i][list[size][j]])
            list[size+1][++ne[size+1]]=list[size][j];
        for (ce[size+1]=ne[size+1], j=k+1; j<=ce[size]; ++j)
        if (g[i][list[size][j]])
list[size+1][++ce[size+1]]=list[size][j];
        dfs(size+1); ++ne[size]; --best;
        for(j=k+1,cnt=0;j<=ce[size];++j)if(!g[i][list[size][j]]) ++cnt;
        if(t==0 || cnt<best) t=k, best=cnt;if(t && best<=0) break; } }
void work(){ int i;ne[0]=0; ce[0]=0;
    for (i=1; i<=n; ++i) list[0][++ce[0]]=i;ans=0; dfs(0);}
```

弦图的完美消除序列

最大势算法:简单的弦图判定,先求完美消除序列L,再利用L判断是否弦图

```
int adj[Maxn][Maxn], n, m, L[Maxn], cnt[Maxn], visit[Maxn], mpL[Maxn];
priority_queue<PII> que;
//利用MSC最大势算法求完美消除序列L,无合法序列返回false
int getList() { int i, j, k, u, v, w;
    for(i = 1; i <= n; i++) cnt[i] = 0, visit[i] = 0;
    while(!que.empty()) que.pop(); que.push(MP(0, 1)); k = n;
    while(!que.empty()) {u = que.top().BB; w = que.top().AA;  que.pop();
        if(w != cnt[u]) continue; visit[u] = 1; mpL[u] = k; L[k--] = u;
        for(v = 1; v <= n; v++) if(!visit[v] && adj[u][v]) {
            cnt[v]++; que.push(MP(cnt[v], v)); } }
    if(k < 1) return true; else return false; }
//利用完美消除序列判断是否弦图
int check() {int i, j, k, u, v, w;
    for(i = n - 1; i >= 1; i--) {u = L[i]; k = -1;
        for(j= i + 1; j <= n; j++){v = L[j];if(adj[u][v]){k = v; break;} }
        if(k != -1) for( j++; j <= n; j++) {
            v = L[j]; if(adj[u][v] && !adj[k][v]) return false; }
    } return true; }
```

1. 极大团：此团不是其他团的子集 2. 最大团：点数最多的团 -> 团数
3. 最小染色：用最少的颜色给点染色使相邻点颜色不同 -> 色数
4. 最大独立集：原图点集的子集，任意两点在原图中没有边相连
6. 最小团覆盖：用最少个数的团覆盖所有的点
   推论 -> 团数<=色数，最大独立集数<=最小团覆盖数
6.弦图：图中任意长度大于 3 的环都至少有 1 个弦
   推论 -> 弦图的每一个诱导子图一定是弦图,弦图的任一个诱导子图不同构于 Cn(n>3)
7.单纯点：记 N(v)为点 v 相邻点的集合，若 N(v)+{v}是一个团，则 v 为单纯点
   引理 -> 任何一个弦图都至少有一个单纯点,不是完全图的弦图至少有两个不相邻的单纯
8.弦图最多有 n 个极大团.
9.设 next(v) 表示 N(v)中最前的点.令 w*表示所有满足 A∈B 的 w 中最后的一个点.判断 v∪N(v)是否为极大团,只需判断是否存在一个 w,满足 Next(w)=v 且|N(v)| + 1≤|N(w)|即可.
10.完美消除序列：点的序列 v1,v2,..,vn，满足 vi 在{vi,vi+1,..,vn}中是单纯点
   定理 -> 一个无向图是弦图，当且仅当它有一个完美消除序列

构造算法 -> 令 cnt[i]为第 i 个点与多少个已标记的点相邻，初值全为零,每次选择一个 cnt[i]最大的结点并打上标记,标记顺序的逆序则为完美消除序列

判定算法 -> 对于每个 vi,其出边为 vi1,vi2,..,vik,然后判断 vi1 与 vi2,vi3,..,vik 是否都相邻,若存在不相邻的情况，则说明不是完美消除序列

11.弦图各类算法：最小染色：完美消除序列从后往前依次给每个点染色,给每个点染上可以染的最小的颜色.//团数=色数

最大独立集：完美消除序列从前往后能选就选.

最小团覆盖：设最大独立集为{p1 , p2 , …, pt},则{p1∪N(p1), …, pt∪N(pt)}为最小团覆盖. //最大独立集数 = 最小团覆盖数!!!

12.区间图：坐标轴上的一些区间看作点，任意两个交集非空的区间之间有边

定理：区间图一定是弦图 */

13.设第 i 个点在弦图的完美消除序列第 p(i)个.令 N(v) = {w | w 与 v 相邻且 p(w) > p(v)}弦图的极大团一定是 v∪N(v)的形式.

Manacher

```
//s为原串, str为插入$和#的串, 读入s后, 调用init(s, str, len),
//最后调用Manacher(str,p,len), 求解遍历p数组求最大值, 注意输出ans-1
最长回文子串对应原串T中的位置:l = (i - p[i])/2; r = (i + p[i])/2 - 2;
int len, p[Maxn]; char s[Maxn], str[Maxn];
void init(char s[], char str[], int& len) {
    int i, j, k; str[0] = '$'; str[1] = '#';
    for (i = 0; i < len; i++) { str[i * 2 + 2] = s[i];
        str[i * 2 + 3] = '#'; }
    len = len * 2 + 2; s[len] = 0; }
void Manacher (char str[], int p[], int len) {
    int i, mx = 0, id; for (i = len; i < Maxn; i++) str[i] = 0;
    for (i = 1; i < len; i++) {
        if ( mx > i ) p[i] = min ( p[2 * id - i], p[id] + id - i );
        else p[i] = 1;
        for (; str[i + p[i]] == str[i - p[i]]; p[i]++);
        if ( p[i] + i > mx ) {mx = p[i] + i; id = i;} } }
```

ExtKMP

```
char S[Maxn], T[Maxn]; int next[Maxn], B[Maxn];
void preExKmp(char T[], int LT, int next[]) {
    int i, ind = 0, k = 1; next[0] = LT;
    while(ind + 1 < LT && T[ind + 1] == T[ind]) ind++; next[1] = ind;
    for(i = 2; i < LT; i++) {
        if(i <= k + next[k] - 1 && next[i - k] + i < k + next[k])
            next[i] = next[i - k];
        else { ind = max(0, k + next[k] - i);
            while(ind + i < LT && T[ind + i] == T[ind]) ind++;
            next[i] = ind; k = i; } } }
void exKmp(char S[], int LS, char T[], int LT, int next[], int B[]) {
    int i, ind = 0, k = 0; preExKmp(T, LT, next);
    while(ind < LS && ind < LT && T[ind] == S[ind]) ind++; B[0] = ind;
    for(i = 1; i < LS; i++) { int p = k + B[k] - 1, L = next[i - k];
        if((i - 1) + L < p) B[i] = L; else { ind = max(0, p - i + 1);
            while(ind + i < LS && ind < LT && S[ind + i] == T[ind]) ind++;
            B[i] = ind; k = i; } } }
```

Aho-Corasick Automaton(部分代码)

```
void buildAC() { head = tail = 0; int i; node * p, * q;
    root->fail = root; que[tail++] = root;
    while(head < tail) { p = que[head++]; q = p->fail;
        for(i = 0; i < 10; i++) {
            if(p->next[i] != NULL) {
                if(p == root) p->next[i]->fail = root;
                else { p->next[i]->fail = q->next[i];
                    p->next[i]->is |= q->next[i]->is; }
                que[tail++] = p->next[i]; }
            else{if(p== root)p->next[i]= root;else p->next[i]= q->next[i];}
        }/*for*/ }/*while*/ }/*func*/
void query(char str[]) { node * p , * q; p = root;
    for(int i = 0, k; str[i]; i++) {k = str[i]-'0'; p = p->next[k];
        if(p->is) {q = p;while(q->is) {cnt[q->lab]++;q = q->fail;} } } }
```

SA

//论文模板，使用时注意num[]有效位为0~n-1，但是需要将num[n]=0，否则RE;另外，对于模板的处理将空串也处理了,作为rank最小的串，因此有效串为0~n共，n-1个，在调用da()函数时，需要调用da(num, n + 1, m); 对于sa[], rank[]和height[]数组都将空串考虑在内，作为rank最小的后缀！ //调用da(num, len+1, m);//m为字符个数略大

```
int len, num[Maxn], sa[Maxn], rank[Maxn], height[Maxn]; //num待处理的串
int wa[Maxn], wb[Maxn], wv[Maxn], wd[Maxn];
//sa[1~n]value(0~n-1); rank[0..n-1]value(1..n); height[2..n]
int cmp(int *r, int a, int b, int x) {
    return r[a] == r[b] && r[a + x] == r[b + x];}
void da(int *r, int n, int m) {//倍增 r为待匹配数组 n为总长度+1 m为字符范围
    int i, j, k, p, *x = wa, *y = wb, *t;for(i = 0; i < m; i++) wd[i] = 0;
    for(i = 0; i < n; i++) wd[x[i] = r[i]]++;
    for(i = 1; i < m; i++) wd[i] += wd[i - 1];
    for(i = n - 1; i >= 0; i--) sa[--wd[x[i]]] = i;
    for(j = 1, 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 < n; i++) wv[i] = x[y[i]];
        for(i = 0;i < m;i++) wd[i] = 0; for(i = 0; i < n; i++) wd[wv[i]]++;
        for(i = 1; i < m; i++) wd[i] += wd[i - 1];
        for(i = n - 1; i >= 0; i--) sa[--wd[wv[i]]] = y[i];
        for(t = x, x = y, y = t, p = 1, x[sa[0]] = 0, i = 1; i < n; i++)
            x[sa[i]] = cmp(y, sa[i - 1], sa[i], j) ? p - 1 : p++;    }
    for(i = 0, k = 0; i < n; i++) rank[sa[i]] = i;
    for(i = 0; i < n - 1; height[rank[i++]] = k)
        for(k ? k-- : 0, j = sa[rank[i] - 1]; r[i + k] == r[j + k]; k++); }
```

字符串的最小表示

```
int MinRep (char S[], int L) {int i = 0, j = 1, k = 0, t;
    while (i < L && j < L && k < L) { //找不到比它还小的或者完全匹配
        t = S[ (i + k) % L] - S[ (j + k) % L];
//t=s[(i+k) >= L ? i + k - L : i + k] - s[(j+k) >= L ? j + k - L : j+k];
        if (t == 0) k++;//相等的话,检测长度加1
        else {//大于的话,s[i]为首的肯定不是最小表示,最大表示就改<
            if(t > 0) i += k + 1; else j += k + 1; if(i == j) j++; k = 0;}
    } return min (i, j);}
```

DLX

```
struct DLX{
    struct Node{ Node *L, *R, *U, *D; int col, row;
    } *head, *row[Maxn], *col[Maxm], node[Maxn * Maxm];
    int colsum[Maxm], cnt;
    /* dancing link 精确覆盖问题 可以添加迭代加深优化：
     * 1）枚举深度h; * 2）若当前深度+predeep > h return false */
/*  int predeep(){bool vis[Maxm];int ret = 0;memset(vis, 0, sizeof(vis));
        for (Node *p = head->R; p != head; p = p->R)
            if (!vis[p->col]) { ret ++ ; vis[p->col] ++ ;
                for (Node *q = p->D; q != p; q = p->D)
                    for(Node *r = q->R;r != q; r = r->R)vis[r->col] = true;
            } return ret; }  //*/
    void init(int mat[][Maxm], int n, int m) {
        cnt = 0; head = &node[cnt ++ ];memset(colsum, 0, sizeof(colsum));
        for(int i = 1; i <= n; i ++ ) row[i] = &node[cnt ++ ];
        for(int j = 1; j <= m; j ++ ) col[j] = &node[cnt ++ ];
        head->D=row[1],row[1]->U = head;head->R = col[1],col[1]->L = head;
        head->U=row[n],row[n]->D = head;head->L = col[m],col[m]->R = head;
        head->row = head->col = 0;
        for(int i = 1; i <= n; i ++ ) { if (i != n) row[i]->D = row[i + 1];
            if(i != 1)row[i]->U= row[i - 1];row[i]->L= row[i]->R = row[i];
            row[i]->row = i, row[i]->col = 0;}
        for(int i = 1; i <= m; i ++ ) {if (i != m) col[i]->R = col[i + 1];
            if (i != 1) col[i]->L = col[i - 1];
            col[i]->U= col[i]->D= col[i];col[i]->col = i,col[i]->row = 0;}
        for(int i = n; i > 0; i -- ) for(int j = m; j > 0; j -- )
            if(mat[i][j]) { Node *p = &node[cnt ++ ];
```

```
                p->R = row[i]->R, row[i]->R->L = p;
                p->L = row[i], row[i]->R = p;
                p->D = col[j]->D, col[j]->D->U = p;
                p->U = col[j], col[j]->D = p;
                p->row = i; p->col = j; colsum[j]++; }/*for*/ }/*func*/
    void remove(Node *c) { c->L->R = c->R; c->R->L = c->L;
        for(Node *p = c->D; p != c; p = p->D)
            for(Node *q = p->R; q != p; q = q->R) {
                q->U->D = q->D;q->D->U = q->U;colsum[q->col] -- ;}}
    void resume(Node *c) {
        for(Node *p = c->U; p != c; p = p->U)
            for(Node *q = p->L; q != p; q = q->L) {
                q->U->D = q; q->D->U = q; colsum[q->col] ++ ;}
        col[c->col]->L->R = col[c->col]; col[c->col]->R->L = col[c->col]; }
    int dfs(int deep){if(head->R==head)return deep;Node *p, *q = head->R;
        for(p = head->R; p != head; p = p->R)
            if(colsum[p->col] < colsum[q->col]) q = p;
        remove(q);
        for(p = q->D; p != q; p = p->D) {
            for(Node* r = p->R; r != p; r = r->R)
                if (r->col != 0) remove (col[r->col]);
            /*可修改区域*/ans[deep] = p->row;/*------*/
            int sta = dfs (deep + 1); if(sta) return sta;
            for(Node* r = p->L; r != p; r = r->L)
                if(r->col != 0) resume (col[r->col]); }
        resume(q); return false; }
///*可重复覆盖
    void remove(Node *c) {
        for(Node * p = c->D; row[p->row] != row[c->row]; p = p->D)
            p->R->L = p->L; p->L->R = p->R;  }
    void resume(Node *c) {
        for(Node * p = c->U; row[p->row] != row[c->row]; p = p->U)
            p->L->R = p->R->L = p; }
    int dfs(int deep) { if(head->R == head) return deep <= K;
        if(deep + predeep() > K) return false;Node *p, *q = head->R, *r;
        for(p = head->R; p != head; p = p->R)
            if(colsum[p->col] < colsum[q->col]) q = p;
        for(p = q->D; p != q; p = p->D) { remove(p);
            for(r = p->R; r != p; r = r->R) if(r->col != 0) remove(r);
            /*可修改区域*/ans[deep] = p->row;/*------*/
            int sta = dfs(deep + 1); if(sta) return sta;
            for(r = p->L; r != p; r = r->L) if(r->col != 0) resume(r);
            resume(p); } return false; } //可重复覆盖*/
} dlx;
```

Tips-Lquartz

网络流拓展:

1.无源汇上下界可行流: 添加附加源汇S,T 对于某边 (u,v) 在新网络中连边S->v容量B[u,v],u->T容量B[u,v],u->v容量C[u,v]-B[u,v].最后,一样也是求一下新网络的最大流,判断从附加源点的边,是否都满流即可.求具体的解:根据最前面提出的强制转换方式,边(u,v)的最终解中的实际流量即为g[u,v]+B[u,v]

2.有源汇上下界可行流: 从汇点到源点连一条上限为INF,下限为0的边.按照1.无源汇的上下界可行流一样做即可.改成无源汇后,求的可行流是类似环的,流量即T->S边上的流量.这样做使S,T也流量平衡了.

3.有源汇的上下界最大流: 方法一:2.有源汇上下界可行流中,从汇点到源点的边改为连一条上限为INF,下限为x的边.因为显然x>ans即MIN(T->S)> MAX(S->T),会使求新网络的无源汇可行流无解的(S,T流量怎样都不能平衡)而x<=ans会有解.所以满足二分性质,二分x,最大的x使得新网络有解的即是所求答案原图最大流. 方法二:从汇点T到源点S连一条上限为INF,下限为0的边,变成无源汇的网络.照求无源汇可行流的方法(如1),建附加源点S'与汇点T',求一遍S'->T'的最大流.再把从汇点T到源点S的这条边拆掉.求一次从S到T的最大流即可.(关于S',T'的边好像可以不拆?)(一定满足流量平衡?)表示这方法我也没有怎么理解.

4.有源汇的上下界最小流

方法一:2.有源汇上下界可行流中,从汇点到源点的边改为连一条上限为x,下限为0的边.与3同理,二分上限,最小的x使新网络无源汇可行流有解,即是所求答案原图最小流. 方法二: 照

求无源汇可行流的方法(如1)，建附加源点S'与汇点T'，求一遍S'->T'的最大流. 但是注意这一遍不加汇点到源点的这条边，即不使之改为无源汇的网络去求解. 求完后，再加上那条汇点到源点上限INF的边. 因为这条边下限为0，所以S',T'无影响. 再直接求一遍S'->T'的最大流. 若S'出去的边全满流，T->S边上的流量即为答案原图最小流，否则无解.

混合欧拉回路判定:

给出混合图（有有向边,也有无向边）,判断是否存在欧拉回路: 首先是图中的无向边随意定一个方向,然后统计每个点的入度（indeg）和出度（outdeg）,如果存在点（indeg - outdeg）是奇数的话,一定不存在欧拉回路;否则就开始网络流构图：

1,对于有向边,舍弃; 对于无向边,就按照最开始指定的方向建权值为 1 的边;

2,对于入度小于出度的点,从源点连一条到它的边,权值为（outdeg - indeg）/2；出度小于入度的点,连一条它到汇点的权值为（indeg - outdeg）/2 的边;

构图完成,如果满流（求出的最大流值 == 和汇点所有连边的权值之和）,则存在欧拉回路.

树Hash判定树同构:

```
//初始时,给树的每一个节点赋一个随机的权值h[i]
int Hash (int j) { int sum = h[j + 5000];//这里的j是记录的节点度
    //这个巧妙的循环,把子节点的hash值都加给了父节点,作为父节点的hash值
    //由于树节点顺序不确定, 因此是子树hash值*根值的累加
    while (*p && *p++ == '0') {sum = (sum + h[j] * Hash (j + 1)) % MOD; }
    return (sum * sum) % 19001; }
```

RMQ(query):

```
int query(int l, int r) { //求[l, r]
    int k = kk[r - l + 1];//预处理 kk[i] = log2(i);
    return min(st[k][l], st[k][r - (1<<k) + 1]);}
```

斯坦纳树:

```
//dp[u][i]表示结点u已经和要连通的结点集合(2^k表示)i连通的最小花费
//初始化将k个点和n个点dp[u][1<<i]初始化为最短路, dp[u][0]=0, 加入队列
//利用spfa求出dp数组    //状态转移分三部分:
//  1. dp[u][su] 利用dp[u][sub] + dp[u][su^sub]更新, sub为su子集
//  2. dp[u][su] 更新相邻的dp[v][sv]
//  3. 将k中不属于su的点与u连接, 利用u, k的最短路
```

生成树相关的一些问题: By 猛犸也钻地 @ 2012.02.24

```
/* 度限制生成树 Q: 求一个最小生成树,其中V0连接的边不能超过K个或只能刚好K个
1. 去掉所有和V0连接的边,对每个连通分量求最小生成树
2. 如果除去点V0外共有T个连通分量,且T>K,无解
3. 于是现在有一个最小T度生成树,然后用dp[V]计算出该点到V0的路径上,权值最大的边是多少,再枚举和V0连接的没有使用过的边,找出一条边,使得用那条边替换已有的边,增加的权值最小,不停替换直到V0出度为K */
/* 次小生成树 Q: 求一个次小生成树,要求权值之和必须大于等于或严格大于其最小生成树
1. 求最小生成树
2. 找一个根然后dp,求出每个点往上走2^L能到达的祖先是谁,以及这段路径上的最大边和次大边(如果仅要求大于等于的话就不需要次大边)
3. 枚举没有使用过的边,利用上面得到的信息,在O(logN)时间内对每条边计算出其能够替换的已有的最大和次大边,然后找出最佳替换方式 */
/* 斯坦纳树 Q: 求一个包含指定的K个特殊点的最小生成树,其他点不一定在树中
1. 用dp[mask][x]记录树根在点x,mask所对应的特殊点集在树中的最小权值之和
2. 将dp[][]初始化为正无穷,只有dp[1<<i][Ai]被初始化为0,Ai为第i个特殊点
3. 先求出所有点对间最短路,然后枚举mask,依次做两种转移:
3.1. 枚举x和mask的子集sub,合并两棵子树
     dp[mask][x]=min(dp[mask][x],dp[sub][x]+dp[mask^sub][x])
3.2. 枚举x和y,计算结点从y移动到x的花费
     dp[mask][x]=min(dp[mask][x],dp[mask][y]+minDistance(y,x))
     在上面的转移中,也可以把这些点同时放到队列里,用spfa更新最短路 */
/* 生成树计数 Q: 给定一个无权的无向图G,求生成树的个数
1. 令矩阵D[][]为度数矩阵,其中D[i][i]为结点i的度,其他位置的值为0
2. 令矩阵A[][]为邻接矩阵,当结点i和j之间有x条边时,D[i][j]=D[j][i]=x
3. 令矩阵C=D-A,矩阵C'为矩阵C抽去第k行和第k列后的一个n-1阶的子矩阵
   其中k可以任意设定,构造完C'后,生成树的个数即为C'行列式的值 */
```

匹配问题结论:

6. 最大独立集 = 顶点数 - 最大匹配数(如果图G满足二分图条件,用二分图匹配来做)

7. 最小点覆盖 = 最大匹配数

8. 最小路径覆盖 = 顶点数 - 最大匹配数(最少不相交简单路径覆盖有向无环图G) (PS: 此处注意, 最小路径覆盖是针对有向图而言, 那么将一个点拆开成为i和i'建立二分图)

二维几何

```
int sign( double x ) {return x<-eps?-1:x>eps;}
struct point {double x, y;
    point( double _x=0, double _y=0 ) : x( _x ), y( _y ) {}
    point operator - ( point p ) {return point( x-p.x,y-p.y );}//+,*,/
    bool operator < ( const point &p ) const {
        return sign( x-p.x ) == 0?sign( y-p.y )<=0:sign( x-p.x ) <= 0;}
    double operator *( point p ) {return x*p.x+y*p.y;}//dot
    double operator ^( point p ) {return x*p.y-y*p.x;}//det
    double arc() {return atan2( y, x );}
    point rotate() {return point( -y, x );}
    point rotate( double arc )
    {return point( x*cos(arc)-y*sin(arc),x*sin(arc)+y*cos(arc) );}  };
bool isLL( point p1, point p2, point q1, point q2 ,point &is ) {
     double m=( q2-q1 )^( p1-q1 ),n=( q2-q1 )^( p2-q1 );
     if ( sign( n-m )==0 )return 0;
     is= ( p1*n-p2*m )/( n-m );    return 1;    }
double disLP(point p1,p2,q){
    return mabs( (p2-p1)^(q-p1) )/( (p2-p1).len() );}
double disSP(point p1,p2,q){
    if(sign((p2-p1)*(q-p1))<=0)return (q-p1).len();
    if(sign((p1-p2)*(q-p2))<=0)return (q-p2).len();
    return disLP(p1,p2,q);}
vector<point>tanCP(point c,double r,point p){//点圆切点
    vector<point>ret;
    double x=(p-c).len2() , d=x-r*r;
    if(sign(d)<0)return ret;if(d<0)d=0;
    point q1=(p-c)*(r*r/x);
    point q2=(p-c)*(-r*sqrt(d)/x).rotate();
    ret.PB(q1-q2);ret.PB(q1+q2);return ret;}
vector<pair<point,point> >tanCC(point c1,double r1,c2,r2){
    vector<pair<point,point> >ret;
    if(mabs(r1-r2)<eps){
        point dir=c2-c1;dir=dir*(r1/dir.len()).rotate();
        ret.PB(MP(c1+dir,c2+dir));ret.PB(MP(c1-dir,c2-dir));
    }else {point p=(c1*(-r2)+c2*r1)/(r1-r2);
        vector<point>A=tanCP(c1,r1,p),B=tanCP(c2,r2,p);
        for(int i=0;i<A.SZ&&i<B.SZ;i++)ret.PB(MP(A[i],B[i]));}
    point p=(c1*r2+c2*r1)/(r1+r2);
    vector<point>A=tanCP(c1,r1,p),B=tanCP(c2,r2,p);
    for(int i=0;i<A.SZ&&i<B.SZ;i++)ret.PB(MP(A[i],B[i]));
    return ret;}
```

~点在多边形内

```
bool Contain(const Point &curr) const { int i, res = 0; Point A, B;
     for (i = 0; i < n; i++) { A = list[i],B = list[(i + 1) % n];
         if (In_The_Seg(A, B, curr)) return 1;
         if (Sign(A.y - B.y) <= 0) swap(A, B);
         if (Sign(curr.y - A.y) > 0|| Sign(curr.y - B.y) <= 0) continue;
         res += Sign(Det(B - curr, A - curr)) > 0;}
     return res & 1;  }
```

~多圆面积

```
struct Tevent {point p;double ang;int add;
    Tevent(point q=point(0,0),double w=0,int e=0){p=q,ang=w,add=e;}
    bool operator <(const Tevent &a) const {return ang < a.ang; }
} eve[maxn * 2];
int E, cnt;
struct Tcir{point o;double r;};
void circleCrossCircle(Tcir &a, Tcir &b) {
    double l = (a.o - b.o).len2();
    double s = ((a.r - b.r) * (a.r + b.r) / l + 1) * .5;
    double t = sqrt(-(l - sqr(a.r - b.r))*(l - sqr(a.r + b.r))/(l*l*4.));
    point dir = b.o - a.o;  point Ndir = point(-dir.y, dir.x);
    point aa = a.o + dir * s + Ndir * t, bb = a.o + dir * s - Ndir * t;
    double A=atan2(aa.y-a.o.y, aa.x-a.o.x),B=atan2(bb.y-a.o.y,bb.x-a.o.x);
```

```
    eve[E++]=Tevent(bb, B, 1);eve[E++]=Tevent(aa,A,-1);if (B > A) cnt++;}
bool g[maxn][maxn], Overlap[maxn][maxn];
//必须去掉重复的圆 Overlap[i][j]:i包含j g[i][j]:i和j相交
double Area[maxn];Tcir c[maxn];int C;
int main() {
    for (int i = 0; i <= C; ++i) Area[i] = 0;
    for (int i = 0; i < C; ++i) {   E = 0,cnt = 1;
        for (int j = 0; j < C; ++j) if (j != i && Overlap[j][i]) cnt++;
        for (int j = 0; j < C; ++j) if (i != j && g[i][j])
                circleCrossCircle(c[i], c[j]);//cnt表示覆盖次数超过cnt
        if (E == 0) {Area[cnt] += PI * c[i].r * c[i].r;
        } else {    double counts = 0; sort(eve, eve + E); eve[E] = eve[0];
            for (int j = 0; j < E; ++j) {   cnt += eve[j].add;
                Area[cnt] += (eve[j].p^eve[j + 1].p) * .5;//det
                double theta = eve[j + 1].ang - eve[j].ang;
                if (theta < 0) theta += PI * 2.;
                Area[cnt]+=(theta-sin(theta))*c[i].r*c[i].r*.5;}}}}
```

~三角形心

```
circle( point a,point b,point c ) {    //外心
    double A,B,C,D,E,F;
    A=2*a.x-2*b.x;    B=2*a.y-2*b.y;    C=SQ( a.len() )-SQ( b.len() );
    D=2*a.x-2*c.x;    E=2*a.y-2*c.y;    F=SQ( a.len() )-SQ( c.len() );
    ct.x=( C*E-B*F )/( A*E-B*D );      ct.y=( A*F-C*D )/( A*E-B*D );
    r=( a-ct ).len();     }
```

内心: $\dfrac{a*\vec{A}+b*\vec{B}+c*\vec{C}}{a+b+c}$ 垂心: $3\vec{G}-2\vec{O}$ 旁心: $\dfrac{-a*\vec{A}+b*\vec{B}+c*\vec{C}}{-a+b+c}$

~多边形与圆面积交

```
double r;    //O(0,0)
double area2( point pa,point pb ) {
    if ( pa.len()<pb.len() )swap( pa,pb ); if ( pb.len()<eps )return 0;
    double a,b,c,B,C,sinB,cosB,sinC,cosC,S,h,theta;
    a=pb.len(),b=pa.len(),c=( pb-pa ).len();
    cosB=pb*( pb-pa )/a/c;B=acos( cosB ); cosC=pa*pb/a/b;  C=acos( cosC );
    if ( a>r ) { S=( C/2 )*r*r;   h=a*b*sin( C )/c;
        if ( h<r&&B<PI/2 )S-=( acos( h/r )*r*r-h*sqrt( r*r-h*h ) );
    } else if ( b>r ) {  theta=PI-B-asin( sin( B )/r*a );
        S=.5*a*r*sin( theta )+( C-theta )*.5*r*r;
    } else S=.5*sin( C )*a*b;  return S;   }
double area() {  double S=0;
    for ( int i=0; i<n; i++ )
        S+=area2( info[i],info[i+1] )*sign( info[i]^info[i+1] );
    return fabs( S );     }
```

~二维凸包

```
vector<point> ConvexHull( vector<point> p ) {
    int n = p.size(), m = 0;
    vector<point> q; q.resize( n * 2 );   sort( p.begin(), p.end() );
    for ( int i = 0; i < n; i ++ ) {
        while ( m > 1 && sign( ( q[m - 1] - q[m - 2] ) ^ ( p[i] - q[m -
2] ) )<= 0 ) m -- ;   q[m ++ ] = p[i]; }
    for ( int i = n – 2, k = m ; i >= 0; i -- ) {
        while ( m > k && sign( ( q[m - 1] - q[m - 2] ) ^ ( p[i] - q[m -
2] ) )<= 0 ) m -- ;   q[m ++ ] = p[i]; }
    if ( n > 1 ) m -- ;   q.resize( m );   return q;    }
double ConvecDiameter( vector<point> p ) {
    int n = p.size(), j = 1;  double maxd = 0; p.push_back( p[0] );
    for ( int i = 0; i < n; i ++ ) {
        while ( ( ( p[i + 1] - p[i] ) ^ ( p[j + 1] - p[i] ) )> ( ( p[i + 1]
- p[i] ) ^ ( p[j] - p[i] ) ) )    j = ( j + 1 ) % n;
    cmax( maxd, max( p[i].dis( p[j] ), p[i + 1].dis( p[j + 1] ) ) );}
    return maxd;}
vector<point> convexCut( const vector<point>&ps, point q1, point q2 ) {
    vector<point> qs;    int n = ps.size();
    for (int i=0; i<n; ++i) { point p1=ps[i], p2=ps[( i + 1 ) % n];
        int d1 = sign( (q2-q1)^(p1-q1) ), d2 = sign( (q2-q1)^(p2-q1) );
```

```
        if ( d1 >= 0 ) qs.PB( p1 );
        if ( d1 * d2 < 0 ) {point is; int flag=isLL( p1, p2, q1, q2,is );
            if ( flag )qs.PB( is );   }    }   return qs;   }
bool in(point p1,p2,p3,p4,q){
    point o12=(p1-p2).rotate();
    point o23=(p2-p3).rotate();
    point o34=(p3-p4).rotate();
    return in(o12,o23,q-p2)||in(o23,o34,q-p3)
        ||in(o23,p3-p2,q-p2)&&in(p2-p3,o23,q-p3);}
bool in(point p1,p2,q){
    return sign(p1^q)>=0&&(p2^q)<=0;}
double disConvexP(vector<point>&ps,q){
    int n=ps.SZ,le=0,re=n;
    while(re-le>1){ int mid=(le+re)>>1;
        if(in(ps[le+n-1]%n,ps[le],ps[mid],ps[(mid+1)%n],q))
            re=mid;
        else le=mid;    }
    return disSP(ps[left],ps[right%n],q);   }
```

~半平面交

```
struct line {    point s,e;   double k;    //s->e left
    line() {}    line( point _s,point _e ):s( _s ),e( _e )
        {    k = atan2( e.y - s.y,e.x - s.x ); }
    bool operator <( const line &L )const {
        if ( sign( k-L.k ) )return k<L.k;
        return ( ( s-L.s )^( L.e-L.s ) )<0;    }
    point operator &( const line &b )const {
        point res;  isLL(s,e,b.s,b.e,res);   return res;  }   };
int HPI( line *L, int n, point *R ) {
    sort( L,L+n );   int tot = 1; line Q[n];
    for ( int i = 1; i < n; i++ )
        if ( sign( L[i].k - L[i-1].k )!=0 )   L[tot++] = L[i];
    int fi = 0, la = 1;  Q[0] = L[0] , Q[1] = L[1];
    for ( int i = 2; i < tot; i++ ) {
        if ( sign( ( Q[la].e-Q[la].s )^( Q[la-1].e-Q[la-1].s ) ) ==0 ||
             sign( ( Q[fi].e-Q[fi].s )^( Q[fi+1].e-Q[fi+1].s ) ) ==0 )
            return 0;
        point s=L[i].s,e=L[i].e;
        while ( fi<la && sign( ( ( Q[la]&Q[la-1] )-s )^( e-s ) )>0 ) la--;
        while ( fi<la && sign( ( ( Q[fi]&Q[fi+1] )-s )^( e-s ) )>0 ) fi++;
        Q[++la] = L[i]; }
    while ( fi<la && sign( ( ( Q[la]&Q[la-1] )-Q[fi].s )^( Q[fi].e-
Q[fi].s ) )>0 ) la--;
    if ( la <= fi + 1 )return 0;  int ret = 0;
    for ( int i = fi; i < la; i++ )R[ret++] = Q[i]&Q[i+1];
    if ( fi < la - 1 )R[ret++] = Q[fi]&Q[la];  return ret;  }
```

三维几何

```
{ret.x=y*s.z-z*s.y; ret.y=z*s.x-x*s.z; ret.z=x*s.y-y*s.x;} //det
struct sfl { spt p,o; sfl() {} sfl( spt _p,spt _o ):p( _p ),o( _o ) {}
    sfl( spt u,spt v,spt w ) {p=u,o=( ( v-u )^( w-u ) ).normal();}  };
double disLP( spt p1,spt p2,spt q ) {
    return fabs( ( ( p2-p1 )^( q-p1 ) ).len()/( ( p2-p1 ).len() ) );    }
double disLL( spt p1,spt p2,spt q1,spt q2 ) {
    spt p=q1-p1,u=p2-p1,v=q2-q1;  double d=( u*u )*( v*v )-SQ( u*v );
    if ( sign( d )==0 )return disLP( q1,q2,p1 );
    double s=( ( p*u )*( v*v )-( p*v )*( u*v ) )/d;
    return disLP( q1,q2,p1+u*s ); }
bool isFL( sfl f,spt q1,spt q2,spt &is ) {
    double a=f.o*( q2-f.p ),b=f.o*( q1-f.p );double d=a-b;
    if ( sign( d )==0 )return 0;  is=( q1*a-q2*b )/d;  return 1;    }
bool isFF( sfl a,sfl b,spt &is1,spt &is2 ) {
    spt e=a.o^b.o;   spt v=a.o^e;double d=b.o*v;
    if ( sign( d )==0 )return 0;  is1=a.p+v*( b.o*( b.p-a.p ) )/d;
    is2=is1+e;   return 1;    }
```

~三维凸包

```
spt s[MXN];  int mark[MXN][MXN],cnt,n;
struct Face {   int a,b,c;Face(int a=0,int b=0,int c=0):a(a),b(b),c(c) {}
    int &operator [](int k) { if(!k)return a;  return k==1?b:c; }   };
vector<Face>face;
void insert(int a,int b,int c) {face.PB(Face(a,b,c));}
double mix(spt a,spt b,spt c) {return a*(b^c);}
double volume(int a,int b,int c,int d) {
    return mix(s[b]-s[a],s[c]-s[a],s[d]-s[a]); }
void add(int v) {
    vector<Face>tmp; int a,b,c; cnt++;
    for(int i=0; i<face.SZ; i++) {a=face[i][0],b=face[i][1],c=face[i][2];
        if(sign(volume(v,a,b,c))<0)
mark[a][b]=mark[b][a]=mark[b][c]=mark[c][b]=mark[c][a]=mark[a][c]=cnt;
        else tmp.PB(face[i]); }
    face=tmp;
    for(int i=0; i<tmp.SZ; i++) { a=face[i][0],b=face[i][1],c=face[i][2];
        if(mark[a][b]==cnt)insert(b,a,v);
        if(mark[b][c]==cnt)insert(c,b,v);
        if(mark[c][a]==cnt)insert(a,c,v); }   }
int Find() {
    for(int i=2; i<n; i++) {  spt ndir=(s[0]-s[i])^(s[1]-s[i]);
        if(ndir==spt())continue;  swap(s[i],s[2]);
        for(int j=i+1; j<n; j++)  if(sign(volume(0,1,2,j))!=0) {
                swap(s[j],s[3]); insert(0,1,2);   insert(0,2,1);
                return 1;    }   }   return 0;    }
bool makeFace() {    sort(s,s+n); n=unique(s,s+n)-s;
    random_shuffle(s,s+n);    face.clear();
    int flag=Find(); if(!flag);//all points on same plane
    memset(mark,0,sizeof mark);   cnt=0;
    for(int i=3; i<n; i++)add(i); return 1;    }
```

```
//绕OS向量，OS视角逆时针旋转弧度A，旋转矩阵
mat Rotate( spt S, double A ){    S.normal();  mat ret;
    double Cos=cos(A),Sin=sin(A);
    ret[0][0]=x*x+(1-x*x)*Cos;    ret[0][1]=x*y*(1-Cos)-z*Sin;
    ret[0][2]=x*z*(1-Cos)+y*Sin;  ret[1][0]=y*x*(1-Cos)+z*Sin;
    ret[1][1]=y*y+(1-y*y)*Cos;    ret[1][2]=y*z*(1-Cos)-x*Sin;
    ret[2][0]=z*x*(1-Cos)-y*Sin;  ret[2][1]=z*y*(1-Cos)+x*Sin;
    ret[2][2]=z*z+(1-z*z)*Cos;    ret[3][3]=1; return ret;  }
```

KdTree

```
const int N = 100000+10;    int K,iCmp,nTid;
typedef vector<int> Obj;
struct Filter {  Obj L,R;
    bool ok( const Obj &o )const { for ( int i=0; i<K; i++ )
            if ( o[i]<L[i]||o[i]>R[i] )return 0;   return 1;   }    };
struct Node {    Obj u;  int c;  Node *ls,*rs; void update() { }    };
Node kd[N<<2],*root;
bool cmp( const Obj &a,const Obj &b ) {    return a[iCmp]<b[iCmp]; }
Node *newNode( const Obj &u,int c ) {
    Node &G=kd[nTid++];   G.u=u,G.c=c,G.ls=G.rs=0;  return &G;  }
Node *build( vector<Obj> &a,int l,int r,int c ) {
    if ( l>=r )return NULL;   int mid=( l+r )/2;  iCmp=c;
    nth_element( a.OP+l,a.OP+mid,a.OP+r,cmp ); Node *G=newNode( a[mid],c );
    G->ls=build( a,l,mid,( c+1 )%K );  G->rs=build( a,mid+1,r,( c+1 )%K );
    G->update();    return G;   }
void queryF( Node *p,const Filter &f ) {
    if ( !p )return;  int x=p->u[p->c];
    if ( f.ok( p->u ) ){}                  /*分支结点*/
    if ( x>=f.L[p->c] )queryF( p->ls,f );  /*左子树*/
    if ( x<=f.R[p->c] )queryF( p->rs,f );  /*右子树*/}
priority_queue<pair<double,Obj>  >Ans;
void queryO( Node *p,Obj &o,int m ) { if ( !p )return;
    pair<double,Obj> now( 0,p->u );
```

```
    for ( int i=0; i<K; i++ )now.AA+=1.0*SQ( p->u[i]-o[i] );
    int c=p->c,flag=0;  Node *x=p->ls,*y=p->rs;
    if ( o[c]>= p->u[c] )swap( x,y );
    queryO( x,o,m );  if ( Ans.SZ<m )Ans.push( now ),flag=1;
    else {   if ( now.AA<Ans.top().AA )Ans.pop(),Ans.push( now );
        if ( 1.0*SQ( o[c]-p->u[c] )<Ans.top().AA )flag=1;   }
    if ( flag )queryO( y,o,m ); }
vector<Obj>p;
void initKdTree() {  K=2,nTid=0; root=build( p,0,p.SZ,0 );   }
```

SAM

```
const int MXN = 100000 + 10, goSZ = 26;
inline int mhash(char c) {return c-'a';}
struct SAM {
    struct State {
        State *suf,*go[goSZ],*nxt;//Parent=suf
        int val,cnt,le;//le~val |Right|=cnt
        int ans;
        void clear() {
            val=cnt=le=0,suf=nxt=0;ans=0;
            memset(go,0,sizeof go);}
    }*root,*last;
    State pool[MXN<<1|1],*cur,*head[MXN|1];
    int L;
    void init() {
        L=0,cur=pool;cur->clear();root=last=cur++;}
    void extend(int w) {
        L++;
        State *p=last,*np=cur++;
        cur->clear();
        np->val=p->val+1,np->cnt=1;
        while(p&&!p->go[w])p->go[w]=np,p=p->suf;
        if(!p)np->suf=root;
        else {  State *q=p->go[w];
            if(p->val+1==q->val)np->suf=q;
            else {  State *nq=cur++;
                cur->clear();
                memcpy(nq->go,q->go,sizeof q->go);
                nq->val=p->val+1;
                nq->suf=q->suf;
                q->suf=nq;
                np->suf=nq;
                while(p&&p->go[w]==q)   p->go[w]=nq,p=p->suf;   }   }
        last=np;    }
    void topo() {
        for(int i=0; i<=L; i++)head[i]=0;
        for(State *p=pool; p!=cur; ++p) {
            p->nxt=head[p->val],head[p->val]=p;
            if(p->suf)p->le=p->suf->val+1;
            else p->le=1;   }
        for(int i=L; i>=0; --i)
            for(State *p=head[i]; p; p=p->nxt)
                if(p->suf)p->suf->cnt+=p->cnt;  }
} foo;
```

Millar & Rho

```
const int S=7;
LL cs[]={2,325,9375,28178,450775,9780504,1795265022};
LL mutiMod( LL a,LL b,LL c ) { //(a*b)%c in 2^63(a,b>0)}
LL powMod( LL x,LL n,LL mod ) { //(x^n)%mod in 2^63 }
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=powMod( a,x,n ),last=ret;
    for ( int i=1; i<=t; i++ ) {  ret=mutiMod( ret,ret,n );
        if ( ret==1&& last!=1&& last!=n-1 ) return 1;
        last=ret;  }     return ret!=1;  }
bool Miller_Rabin( LL n ) {
```

```
    LL x=n-1,t=0; bool flag=1;  while ( ( x&1 )==0 ) x>>=1,t++;
    if ( t>=1&& ( x&1 )==1 )for ( int k=0; k<S; k++ ) {
            LL a=cs[k]; if ( check( a,n,x,t ) ) {flag=1; break;} flag=0; }
    if ( !flag || n==2 ) return 0;  return 1;   }
vector<LL>factor;                 //clear
LL Pollard_rho( LL x,LL c ) { LL i=1,x0=rand()%x,y=x0,k=2;
    while ( i++ ) {
        x0=( mutiMod( x0,x0,x )+c )%x; LL d=__gcd( y>x0?y-x0:x0-y,x );
        if ( d!=1&& d!=x ) return d; if ( y==x0 ) return x;
        if ( i==k ) y=x0,k<<=1; }   }
void findfac( LL n ) {        //递归进行质因数分解 N
    if ( !Miller_Rabin( n ) ) { factor.PB( n ); return; }
    LL p=n;  while ( p>=n ) p=Pollard_rho( p,rand() % ( n-1 ) +1 );
    findfac( p );  findfac( n/p ); }
```

extGcd

```
LL extGcd (LL a, LL b, LL &x, LL &y) {LL ret = a;
    if (b) {ret = extGcd (b, a % b, y, x); y -= (a / b) * x;}
        else x = 1, y = 0;    return ret;  }
LL modInv (LL a, LL m) { LL x,y; extGcd(a,m,x,y); return (m+x%m)%m; }
//m 为质数 [费马小定理]a^(m-1)=1 mod m
```

阶乘模分解

```
int fact[MAX_P]; //预处理 n! mod p 的表 O(min(n,p))
int modFact (int n, int p, int &e) {// n!=a*p^e return a%p
    e = 0; if (!n) return 1; int res = modFact (n / p, p, e); e += n / p;
    if (n / p % 2) return res * (p - fact[n % p]) % p;
    return res * fact[n % p] % p; }
int eulerPhi (int n) {            // test: phi(846720)=193536
    int res = n; for (int i = 2; i * i <= n; i++)  //more fast with prime
        if (n % i == 0) {res=res/i*(i-1); while(n%i==0)n/=i; }
    if (n != 1) res = res / n * (n - 1); return res;   }
```

模同余方程

```
//a_i*x=b_i {%m_i}  m_i 可以不互质  //pair<b,m>  x=b {%m}
pair<LL,LL> linearMod( vector<LL>&A,vector<LL>&B,vector<LL>&M ) {LL x=0,m=1;
    for ( int i=0; i<A.SZ; i++ ) {
        LL a=A[i]*m,b=B[i]-A[i]*x,d=__gcd( M[i],a );
        if ( b%d )return MP( 0,-1 );
        LL t=b/d*modInv( a/d,M[i]/d )%( M[i]/d ); x+=m*t;m*=M[i]/d;x%=m;  }
    return MP( ( ( x%m )+m )%m,m );   }
```

离散对数 BSGS

```
int extBSGS( int A,int B,int C ) { //A^x==B mod C
    for (int i=0,tmp=1%C;i<100;i++,tmp=1LL*tmp*A%C)if (tmp==B)return i;
    int temp,d=0; LL D=1%C;
    while ( ( temp=__gcd( A,C ) )!=1 ) {if ( B%temp )return -1;
        C/=temp,B/=temp; d++;D=1LL*A/temp*D%C;    }
    int s=( int )ceil( sqrt( C+eps ) )+1; vector<PII>L;    LL G=1%C;
    for ( int i=0; i<s; i++ ) {   L.PB( MP( G,i ) );   G=G*A%C; }
    sort( L.OP,L.ED );
    for ( int i=0; i<=s; i++ ) {
        int tmp=modInv( D,C )*B%C;
        int id=lower_bound( L.OP,L.ED,MP( tmp,-1 ) )-L.OP;
        if ( id<L.SZ&&id>=0&&L[id].AA==tmp )  return i*s+L[id].BB+d;
        D=D*G%C; }   return -1;   }
```

线性筛

```
for(inv[1]=1,i=2;i<MXN;i++)inv[i]=inv[MOD%i]*(MOD-MOD/i)%MOD;//MODisPrime
int mu[N], p[N], pn; bool flag[N];    //true 为合数
void init(int n) {   pn = 0; mu[1] = 1;
    for(int i = 2; i <= n; i++) {
        if(!flag[i]) {p[pn ++ ] = i; mu[i] = -1; /*phi[i]=i-1;*/ }
        for(int j = 0; j < pn && i * p[j] <= n; j++) {
            flag[i * p [j]] = true;
            if(i % p[j] == 0) {
                mu[i*p[j]] = 0;/*phi[i*p[j]]=p[j]*phi[i];*/break;
```

```
            } else  mu[i*p[j]]=-mu[i]; /*phi[i*p[j]]=(p[j]-1)*phi[i];*/}}}
```

二次剩余

```
//call(b,0,a,(p+1)/2,p) return a sol of {x^2=a (mod p)}
//{p is odd prime}&&{a^[(p-1)/2]=1 mod p}&&{b^( (p-1)/2 )==-1 mod p}
LL call(LL b,LL c,LL a,LL x,LL p){
    if(x%2==0)return modPow(b,c/2,p)*modPow(a,x/2,p)%p;
    LL tp=modPow(b,c/2,p)*modPow(a,(x-1)/2,p)%p;
    if(tp==1)return call(b,c/2,a,(x+1)/2,p);
    return call(b,(c+p-1)/2,a,(x+1)/2,p); }
int pDiv2,P,a,b,c,Pb,d; /*a*x^2+b*x+c==0 (mod P) 求 0..P-1 的根 */
inline int calc(int x,int Time){
   if (!Time) return 1;  int tmp=calc(x,Time/2);
   tmp=(long long)tmp*tmp%P;
   if (Time&1) tmp=(long long)tmp*x%P;    return tmp;  }
inline int rev(int x){ if (!x) return 0;   return calc(x,P-2);}
inline void Compute(){
   while (1) { b=rand()%(P-2)+2;  if (calc(b,pDiv2)+1==P) return; } }
int main(){  srand(time(0)^312314);  int T;
   for (scanf("%d",&T);T;--T) {  scanf("%d%d%d%d",&a,&b,&c,&P);
      if (P==2)  {/*simple case*/
      }else  {  int delta=(long long)b*rev(a)*rev(2)%P;
         a=(long long)c*rev(a)%P-sqr( (long long)delta )%P;
         a%=P;a+=P;a%=P;  a=P-a;a%=P;  pDiv2=P/2;
         if (calc(a,pDiv2)+1==P) puts("0");
         else {int t=0,h=pDiv2;    while (!(h%2)) ++t,h/=2;
            int root=calc(a,h/2);
            if (t>0) {  Compute();  Pb=calc(b,h); }
            for (int i=1;i<=t;++i) {
               d=(long long)root*root*a%P;
               for (int j=1;j<=t-i;++j)  d=(long long)d*d%P;
               if (d+1==P)  root=(long long)root*Pb%P;
               Pb=(long long)Pb*Pb%P;   }
            root=(long long)a*root%P;
            int root1=P-root; root-=delta;
            root%=P; if (root<0) root+=P;
            root1-=delta; root1%=P;  if (root1<0) root1+=P;
            if (root>root1) { t=root;root=root1;root1=t;  }
            if (root==root1) printf("1 %d\n",root);
            else printf("2 %d %d\n",root,root1);
   }}}return 0;  }
```

Pell 方程求解

```
//求x^2-ny^2=1的最小正整数根,n不是完全平方数
p[1]=1;p[0]=0; q[1]=0;q[0]=1; a[2]=(int)(floor(sqrt(n)+1e-7));
g[1]=0;h[1]=1;
for (int i=2;i;++i) {
     g[i]=-g[i-1]+a[i]*h[i-1];     h[i]=(n-sqr(g[i]))/h[i-1];
     a[i+1]=(g[i]+a[2])/h[i];      p[i]=a[i]*p[i-1]+p[i-2];
     q[i]=a[i]*q[i-1]+q[i-2];      检查p[i],q[i]是否为解，如果是，则退出
}
```

FFT

```
const int MXN = 1<<20;
double ax[MXN],ay[MXN],bx[MXN],by[MXN],ansx[MXN],ansy[MXN];
int revv(int x,int mask) {    int ret=0;
   for(int i=0; i<mask; i++) {    ret<<=1; ret|=x&1; x>>=1; }
   return ret; }
void fft(double * rl, double * ig, int n, bool sign) {
   int d=0; while((1<<d) <n) ++d;
   for(int i=0; i<n; i++) {  int j=revv(i,d);
      if(i<j) swap(rl[i],rl[j]),swap(ig[i],ig[j]);    }
   for(int m=2; m<=n; m<<=1) {    int mh=m>>1;
      double _wr=cos(2*PI/m),_wi=sin(2*PI/m);if(sign) _wi*=-1.0;
      for(int i=0; i<n; i+=m) {  double wr=1,wi=0;
         for(int j=i; j<mh+i; j++) {int k=j+mh;
            double er=rl[k]*wr-ig[k]*wi , ei=rl[k]*wi+ig[k]*wr;
```

```
                double cr=rl[j],ci=ig[j];
                rl[j]+=er  ,ig[j]+=ei;  rl[k]=cr-er,ig[k]=ci-ei;
                double qr=wr*_wr-wi*_wi , qi=wr*_wi+wi*_wr;
                wr=qr,wi=qi;    }   }   }
    if(sign) for(int i=0; i<n; i++)    rl[i]/=n,ig[i]/=n;    }
int fftmultiply(int *a,int la,int *b,int lb,LL *ans) {
    int lans=max(la,lb),ln=0,i;    while((1<<ln) <lans) ++ln;   lans=2<<ln;
    for(i=0; i<lans; i++)ax[i]=i<la?a[i]:0,ay[i]=0; fft(ax,ay,lans,0);
    for(i=0; i<lans; i++)bx[i]=i<lb?b[i]:0,by[i]=0; fft(bx,by,lans,0);
    for(i=0; i<lans; i++) {
        ansx[i]=ax[i]*bx[i]-ay[i]*by[i];
        ansy[i]=ax[i]*by[i]+ay[i]*bx[i];    }   fft(ansx,ansy,lans,1);
    for(i=0; i<lans; i++)   ans[i]=ansx[i]+0.5; return lans; }
```

NTT 数论变换

```
const int p=786433,g=10; //g 是 p 的原根,p 为素数且 len|p-1&&len=2^?
LL pm(LL a,int n,int m=p) {}//a^n%p
int rb(int x,int m) {
    int r=0; for(;m>1;m>>=1,x>>=1)r=r<<1|x&1; return r;}
void ntt(int *a,int len){
    for(int i=0,j;i<len;++i)
        if(i<(j=rb(i,len)))swap(a[i],a[j]);
    for(int m=1; m < len ; m<<=1) {
        LL w=1;int w0 = pm(g, (p-1)/m>>1);
        for(int k = 0; k<len; k+=(m<<1), w=1)
            for(int j=0;j<m; ++j , w=w*w0 %p) {
                int t= w*a[k+j+m]%p;
                a[k+j+m] = (a[k+j]+p-t) %p;
                a[k+j] = (a[k+j]+t)%p; }    }    }
void conv(int *a, int *b, int *c, int len) {
    static int wa[N], wb[N];  rep (i, len) wa[i] = a[i], wb[i] = b[i];
    ntt(wa, len);ntt(wb, len);    int inv = pm(len, p - 2);
    rep (i, len) c[i] = wa[i] * (LL)wb[i] % p * inv % p;
    reverse(c + 1, c + len);  ntt(c, len); }
```

单纯形

```
const int MVar = 444, MEqa = 444;
long double a[MEqa][MVar];int idx[MVar],nv,ne;
int nxt[MVar];//-a[0][0]=max ∑a[0][i]*x[i]
void show() {
    for ( int i=0; i<=ne; i++ ) {
        printf( "%d[%d]%3.5lf=\t",i,idx[i],a[i][0] );
        for ( int j=1; j<=nv; j++ )if(abs(a[i][j])>eps)printf( "%3.5lf*x[%d]
",a[i][j],j );
        printf( "\n" ); }  printf( "\n" ); }
void pivot( int e,int v ) {
    int i,j;    long double temp;   int tp=MVar-1;
    for ( j=nv; j>=0; j-- )nxt[j]=-1;
    for ( j=nv; j>=0; j-- )if ( abs( a[e][j] )>eps ) {nxt[tp]=j; tp=j;}
    temp=a[e][v];
    for ( tp=nxt[MVar-1]; tp!=-1; tp=nxt[tp] )a[e][tp]/=temp;
    for ( i=0; i<=ne; i++ )if ( abs( a[i][v] )>eps&&i!=e ) {
        temp=a[i][v];
        for ( tp=nxt[MVar-1]; tp!=-1; tp=nxt[tp] )
            a[i][tp]-=temp*a[e][tp];    }
    idx[e]=v;   }
int dualsolve() {   int i,j;    long double temp;
    for ( j=1; j<=nv; j++ )if ( a[0][j]<-eps )return 0;
    while ( 1 ) {   int l=0,r=0;    temp=-eps;
        for ( i=1; i<=ne; i++ )if ( a[i][0]<temp )temp=a[i][0],r=i;
        if ( !r )return 1;  temp=1e100;
        for ( j=1; j<=nv; j++ )if ( a[r][j]<-eps&&a[0][j]/a[r][j]<temp )
            temp=a[0][j]/a[r][j],l=j;
        if ( !l )return 0;  pivot( r,l );   }   }
int solve() {   int i,j;    long double temp;
    for ( i=1; i<=ne; i++ )for(k=0;k<=ne;k++)
```

```
        if (k!=i&& abs( a[k][idx[i]] )>eps ) {
            temp=a[k][idx[i]];
            for ( j=0; j<=nv; j++ )a[k][j]-=temp*a[i][j];   }
    int dual=0;
    for ( i=1; i<=ne; i++ )if ( a[i][0]<-eps )dual=1;
    if ( dual ) {   int dual=dualsolve();
        if ( !dual )return 0; /*no solution*/ }
    while ( 1 ) {   int l=0,r=0;    temp=1e100;
        for ( j=1; j<=nv; j++ )if ( a[0][j]>eps ) {l=j; break;}
        if ( !l )return 1;     //done
        for ( i=1; i<=ne; i++ )if ( a[i][l]>eps&&a[i][0]+eps<a[i][l]*temp )
            temp=a[i][0]/a[i][l],r=i;
        if ( !r )return -1;        //infinite
        pivot( r,l );   }   }
```

自适应 **simpsion**

```
long double simpson(long double a,long double b) {
    long double c=a+ (b-a) /2;    return (f(a)+4*f(c)+f(b)) * (b-a) /6; }
long double asr(long double a,long double b,long double eps,long double A){
    long double c=a+ (b-a) /2; long double L=simpson(a,c),R=simpson(c,b);
    if(fabs(L+R-A) <15*eps) return L+R+ (L+R-A) /15.;
    return asr(a,c,eps/2,L)+asr(c,b,eps/2,R); }
long double asr(long double a,long double b,long double eps) {
    return asr(a,b,eps,simpson(a,b)); }
```

高斯消元

```
double a[MXN][MXN],b[MXN];  int index[MXN];
int gauss_tpivot(int m,int n) { int i, j, k, row, col;  double maxp, t;
    for(i = 0; i < m; i++)index[i] = i;
    for(k = 0; k < n; k++) {
        for(maxp = 0, i = k; i < m; i++)for(j = k; j < n; j++)
            if(fabs(a[i][j]) > fabs(maxp))maxp = a[row = i][col = j];
        if(fabs(maxp) < eps)return 0;
        if(col != k) {swap(index[col],index[k]);
            for(i = 0; i < m; i++)swap(a[i][col],a[i][k]);}
        if(row != k) {swap(b[k],b[row]);
            for(j = k; j < n; j++)swap(a[k][j],a[row][j]);}
        for(j = k + 1; j < n; j++) {a[k][j] /= maxp;
            for(i = k + 1; i < m; i++)a[i][j] -= a[i][k] * a[k][j];}
        b[k] /= maxp;for(i = k + 1; i < m; i++)b[i] -= b[k] * a[i][k];}
    for(i = n - 1; i >= 0; i--)for(j = i + 1; j < n; j++)
    b[i] -= a[i][j] * b[j];
    for(k = 0; k < n; k++)a[0][index[k]] = b[k];
    for(k = 0; k < n; k++)b[k] = a[0][k]; return 1;}
```

划分数

```
LL dp[100010];// n 划分为 K 个自然数的和的方案数
void partition( int n ) { int i,j,r;
    for ( dp[0]=1,i=1; i<=n; i++ ) {  dp[i]=0;
        for ( j=1,r=1; i>=( 3*j*j-j )/2; j++,r*=-1 ) {
            dp[i]+=dp[i-( 3*j*j-j )/2]*r;
            if ( i>=( 3*j*j+j )/2 )dp[i]+=dp[i-( 3*j*j+j )/2]*r;
            dp[i]=dp[i]%MOD+MOD; }   }   }
int get( int n,int k ) {  //all parts are repeated less than k times.
    LL ret=dp[n];         //<==>all parts are less than k
    for ( int j=1,r=-1; n>=k*( 3*j*j-j )/2; j++,r*=-1 ) {
        ret+=dp[n-k*( 3*j*j-j )/2]*r;
        if ( n>=k*( 3*j*j+j )/2 ) ret+=dp[n-k*( 3*j*j+j )/2]*r;
        ret=ret%MOD+MOD; }   return ret%MOD; }
```

多项式拟合

```
typedef double VAL;// 传入 y=f(x)上的 n 个点，拟合出一元 n-1 次方程，返回各项系数
vector<VAL> interpolation(const VAL x[], const VAL y[], int n){
    vector<VAL> u(y,y+n),ret(n),sum(n);   ret[0]=u[0],sum[0]=1;
    for(int i=1;i<n;i++){
        for(int j=n-1;j>=i;j--) u[j]=(u[j]-u[j-1])/(x[j]-x[j-i]);
        for(int j=i;j;j--){
            sum[j]=-sum[j]*x[i-1]+sum[j-1];   ret[j]+=sum[j]*u[i]; }
```

```
        sum[0]=-sum[0]*x[i-1];    ret[0]+=sum[0]*u[i]; }
    return ret;  }
```

数位 dp

```
LL f[11][23][1<<11|1];    //initial with -1
int dig[23],ndig;         //ndig=max{i}:dig[i]!=0
int isTarget(int mask,int first,int aim){
    return __builtin_popcount(mask)==aim; }
int vary(int mask,int a){
    for(int i=a;i<10;i++)if(mask>>i&1) return mask^(1<<i)^(1<<a);
    return mask^(1<<a);  }
LL dfs(int id,int mask,int aim,int even=1,int first=1){
    //dfs(ndig,startMask,aim)
    if(id==-1)return isTarget(mask,first,aim);
    if(!even&&~f[aim][id][mask])return f[aim][id][mask];
    LL ret=0;
    if(even)ret+=dfs(id-1,vary(mask,dig[id]),aim,1,0);
    if(first)ret+=dfs(id-1,mask,aim,0,1);
    int u=even?dig[id]-1:9;
    for(int i=first?1:0;i<=u;i++)
        ret+=dfs(id-1,vary(mask,i),aim,0,0);
    return even?ret:f[aim][id][mask]=ret; }
LL solve(LL re,int aim){
    ndig=0;  while(re){dig[ndig++]=re%10;re/=10;}
    return dfs(--ndig,0,aim); }
```

Java 开根

```
public static BigInteger getsqrt(BigInteger n){
    if (n.compareTo(BigInteger.ZERO)<=0) return n;
    BigInteger x,xx,txx;     xx=x=BigInteger.ZERO;
    for (int t=n.bitLength()/2;t>=0;t--){
        txx=xx.add(x.shiftLeft(t+1)).add(BigInteger.ONE.shiftLeft(t+t));
        if (txx.compareTo(n)<=0){
            x=x.add(BigInteger.ONE.shiftLeft(t)); xx=txx;
        }   }return x;   }
```

直线下格点统计

```
LL count(LL n, LL a, LL b, LL m) {     //求∑_{k=0..n-1} ⌊(a+bk)/m⌋, a,b > 0
    if (b==0) return n * (a / m);
    if (a>=m) return n * (a / m) + count(n, a % m, b, m);
    if (b>=m) return (n-1) * n/2* (b/m) + count(n, a, b % m, m);
    return count((a + b * n) / m, (a + b * n) % m, m, b);   }
```

线性递推求最小

```
LL getmin(LL start,LL slope,LL cnt,LL mod) {
    //min{ (start+k*slope)%mod | 0<=k<=cnt }
    start%=mod; if(start+slope*cnt<mod)return start;
    if(start>=slope) {int use=(mod-1-start)/slope+1;
        return min(start,getmin(start+use*slope,slope,cnt-use,mod));}
    LL res=start,ns=slope-mod%slope,ncnt=(start+slope*cnt)/mod;
    return min(res,getmin(start,ns,ncnt,slope));   }
```

树的计数

有标号无根树 n^(n-2),有标号有根树 n^(n-1)

令 $S_{n,j} = \sum_{1\le i\le n/j} a_{n+1-ij} = S_{n-j,j} + a_{n+1-j}$

n+1 个结点的有根树的总数 $a_{n+1} = \frac{\sum_{1\le j\le n} j a_j S_{n,j}}{n}$

附：$a_1 = 1, a_2 = 1, a_3 = 2, a_4 = 4, a_5 = 9, a_6 = 20, a_9 = 286, a_{11} = 1842$

当 n 是奇数时，则有 $a_n - \sum_{1\le i\le n/2} a_i a_{n-i}$ 种不同的无根树。

当 n 是偶数时，则有这么多种不同的无根树。

$$a_n - \sum_{1\le i\le \frac{n}{2}} a_i a_{n-i} + \frac{1}{2} a_{n/2}(a_{n/2} + 1)$$

Stl 白科技

```
#include <ext/rope>
using namespace __gnu_cxx;
rope<int>L;//[]
L.insert(start,what);L.erase(start,len);  L.copy(start,len,int *);
his[i]=new rope<int>(*his[i-1]); //resistant
```

```
#include<ext/pb_ds/priority_queue.hpp>
__gnu_pbds::priority_queue<int>a,b;   a.join(b);
#include <ext/hash_map>
using namespace __gnu_cxx;
struct hashLL{
    inline unsigned operator()(long long a)const{return a*4423;}   };
hash_map<long long,int,hashLL>M;
```

Tips-Ronnoc

~BIT 二分

```
int find_Kth(int k){     //UESTC_Dagon
    int idx=0; for(int i=20;i>=0;i--){    //越界可能
        idx|=1<<i; if(idx<=n&&_[idx]<k)k-=_[idx];
        else idx^=1<<i; }    return idx-2;    }
```

~错排

一重 D[0]=1;D[1]=0;D[n]=(n-1)*(D[n-1]+D[n-2])

二重 U[n]=Σ(-1)^k*(2n)/(2n-k)*comb(2n-k,k)*(n-k)!

n+m 个数 m 错排:dp[0]=n!,dp[1]=n*n!,dp[i]=n*dp[i-1]+(i-1)*(dp[i-1]+dp[i-2]);/?

~Picks 定理格点简单多边形面积 S,边上格点数 B,内部格点数 I :: S=B/2+I-1

~四面体 O-ABC 体积公式 a=AB, b=BC, c=CA, d=OC, e=OA, f=OB

(12V)^2 = a^2d^2(b^2 + c^2 + e^2 + f^2 - a^2 - d^2) + b^2e^2(c^2 + a^2 + f^2 + d^2 - b^2 - e^2) + c^2f^2(a^2 + b^2 + d^2 + e^2 -c^2 - f^2) - a^2b^2c^2 - a^2e^2f^2 - d^2b^2f^2 - d^2e^2c^2

~卡特兰数 n 个元素入栈的出栈顺序种数 Cat(n)

Cat(n)=Comb(2n,n)/(n+1)=Comb(2n,n)-Comb(2n,n+1)=Cat(n-1)*(4n-2)/(n+1)

~Bell 数 n 元素的集合划分数: Bell[n+1]=Σcomb(n,k)Bell[k], Bell[p+n]=B[n]+B[n+1] (mod p) {p 是质数}, Bell[n]=ΣStirling2[n,k]

~第一类 Stirling 数: 将 n 个物体排成 k 个非空循环排列的方法数

Str1[n,k]=(n-1)*Str1[n-1,k]+Str1[n-1,k-1]

~第二类 Stirling 数:将 n 个物体划分成 k 个非空的不可辨别的集合的方法数

Str2[n,k]=k*Str2[n-1,k]+Str2[n-1,k-1]

~欧拉数 n 全排列,恰有 m 个上升位置的排列数 E[n,k]=(k+1)E[n-1,k]+(n-k)E[n-1,k-1]

~容斥反演 g(A)=Σ{S⊆A}f(S) <==> f(A)=Σ{S⊆A}(-1)^(|A|-|S|)g(S)

~莫比乌斯反演 g(n)=Σ{d|n}f(d) <==> f(n)=Σ{d|n}miu[d]g(n/d)

g(x)=Σ{n=1,n<=x}f(x/n) <==> f(x)=Σ{n=1,n<=x}miu[n]g(x/n)

~二项式反演 an=Σ(-1)^k*C(n,k)bk <==> bn=Σ(-1)^k*C(n,k)ak

an=ΣC(n,k)bk <==> bn=Σ(-1)^(n-k)*C(n,k)ak

~Burnside 引理 $\text{ans} = \frac{(\sum \text{每种置换下的不变的元素个数})}{\text{置换群中置换的个数}}$

~Polya 定理 G 是集合 X 上的置换群,X 的每个元素可以染成 k 种颜色,不等价的着色数 P,nc(g)为置换 g 的循环个数`|G|*P=Σ{g∈G}k^nc(g)`

~三次方程求根公式 $x^3 + px + q = 0$

$$x_j = \omega^j \sqrt[3]{-\frac{q}{2} + \sqrt{\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3}} + \omega^{2j} \sqrt[3]{-\frac{q}{2} - \sqrt{\left(\frac{q}{2}\right)^2 + \left(\frac{p}{3}\right)^3}}$$

其中 j=0,1,2, $\omega = (-1 + i\sqrt{3})/2$

当求解$ax^3 + bx^2 + cx + d = 0$时, 令 $x = y - b/3a$ 做转化

~三角组合公式

$\sum_{k=1}^{n} k^4 = \frac{n(n+1)(2n+1)(3n^2+3n-1)}{30}$ $\sum_{k=1}^{n} k^5 = \frac{n^2(n+1)^2(2n^2+2n-1)}{12}$

$\sin(\alpha \pm \beta) = \sin\alpha \cos\beta \pm \cos\alpha \sin\beta$ $\cos(\alpha \pm \beta) = \cos\alpha \cos\beta \mp \sin\alpha \sin\beta$

$\tan(\alpha \pm \beta) = \frac{\tan(\alpha) \pm \tan(\beta)}{1 \mp \tan(\alpha)\tan(\beta)}$ $\tan(\alpha) \pm \tan(\beta) = \frac{\sin(\alpha \pm \beta)}{\cos(\alpha)\cos(\beta)}$

~Stirling 公式: n!≈sqrt(2nPI)*(n/e)^n

~枚举定大小集合

```
int sub = (1 << k) - 1;while (sub < 1 << n) {  //枚举大小为 k 的子集
    int x = sub & -sub, y = sub + x;
    sub = (sub & ~y) / x >> 1 | y;}
```

~牛顿迭代 f(x)某零点临近点 x_0, x_m=(x*f'[x]-f[x])/f'[x]

~二阶线性递推循环节 f(n)=a*f(n-1)+b*f(n-2),求 f(n)%p 的循环节

p = Πpi^ai 分别求 pi^ai 的循环节, 取最小公倍数

p mod px^ax 的循环节 = G(px) * px^(ax-1) , G(px) 就是 p mod px 的最小循环节

x^2=a*x+b ,delta=a*a+4*b 对于 G(px),如果 delta 是模 px 的二次剩余,G(px)是 px-1 的

因子，否则 G(px)是(px-1)*(px+1)的因子,矩阵快乘暴力判断

~四边形不等式

```
rep(r,1,n)rep(i,1,n-r){if(r==1)K[i][i+1]=i,dp;
        else rep(k,K[i][i+r-1],K[i+1][i+r])if(better)K[i][i+r]=k,dp;}
```

~拉格朗日插值 p_i(x)=∏{j!=i}(x-x_i)/(x_j-x_i);f(x)=∑y_i*p_i(x)

~Jacobi 迭代 AX=B:A=D-L-U,D(主对角线),L(下三角不含 D),U(上三角不含 D)

X=D^(-1)*((L+U)X+B)

~反素数(240@720720<=10^6),(1600@2095133040<2^31),

(6720@963761198400<=10^12),(26880@866421317361600<=10^15)

~欧拉定理 简单多面体顶点数 V、面数 F 及棱数 E :: V+F-E=2

~威尔逊定理 (p-1)!=-1 mod p

设正整数 n 的质因数分解为 n = ∏pi^ai,则 **x^2+y^2=n 有整数解**的充要条件是 n 中不存在形如 pi≡3(mod 4) &(and) 指数 ai 为奇数的质因数 pi

p 为奇素数，x^b = a(mod p),x 为 p 的 **b 次剩余**的必要充分条件为 若 x^（(p-1)/ (p-1 和 b 的最大公约数)） mod p=1.

~日期换算
```
int days(int y,int m,int d){if(m<3)y--,m+=12;
Return 365*y+y/4-y/100+y/400+(153*m+2)/5+d;}
```

~经纬度求球面最短距离
```
double Dist(double la1,lo1,la2,lo2,R){
    la1*=PI/180,lo1*=PI/180,la2*=PI/180,lo2*=PI/180;
    double x1=cos(la1)*sin(lo1),y1=cos(la1)*cos(lo1),z1=sin(la1);
    double x2=cos(la2)*sin(lo2),y2=cos(la2)*cos(lo2),z1=sin(la2);
    return R*acos(x1*x2+y1*y2+z1*z2);}
```

<u>Treap[version 2]</u>

```
bool random(double p) {
    return (double)rand() / RAND_MAX < p; }
struct Node;
typedef std::pair <Node*, Node*> Pair;
struct Node {
    int value, size;
    Node *left, *right;
    Node(int);
    Pair split(int);
    Node* update() {
        size = left->size + 1 + right->size;
        return this; }
    void print();
}*null;
Node::Node(int value) : value(value), size(1), left(null), right(null) {}
Node* merge(Node *p, Node *q) {
    if (p == null)return q;
    if (q == null)return p;
    if (random((double)p->size / (p->size + q->size))) {
        p->right = merge(p->right, q);
        return p->update(); }
    q->left = merge(p, q->left);
    return q->update(); }
Pair Node::split(int need) {
    if (this == null)return make_pair(null, null);
    if (left->size >= need) {
        Pair ret = left->split(need);
        left = null;
        this->update();
        return make_pair(ret.first, merge(ret.second, this)); }
    Pair ret = right->split(need - (left->size + 1));
    right = null;
    this->update();
    return make_pair(merge(this, ret.first), ret.second); }
int main() {
    null = new Node(-1);
    null->size = 0;
    null->left = null->right = null;
    return 0; }
```

<u>树链剖分</u>

```
int root[Maxn],No[Maxn],top[Maxn],slen[Maxn];
int parent[Maxn],lv[Maxn],depth[Maxn],size[Maxn];
int que[Maxn];
void bfsinit(int u) {}
void split(int u,int Top) {
    int i,j,v,Max=-1;
    top[u]=Top;
    for(j=last[u];j!=-1;j=edge[j].next){
        v=edge[j].v;
        if(v==parent[u])continue;
        if(Max==-1 || size[Max]<size[v])Max=v; }
    if(Max==-1){
```

```
        No[u]=depth[u]-depth[Top]+1;
        slen[u]=No[u];
        build(root[u]=++pcnt,1,slen[u]);
        refresh(root[u],1,slen[u],No[u],lv[u]);
        return; }
    split(Max,Top);
    No[u]=No[Max]-1;
    root[u]=root[Max];
    slen[u]=slen[Max];
    refresh(root[u],1,slen[u],No[u],lv[u]);
    for(j=last[u];j!=-1;j=edge[j].next){
        v=edge[j].v;
        if(v==parent[u] || v==Max)continue;
        split(v,v); }}
LL Query(int u,int v) {
    LL ret=0;
    while(1){
        if(top[u]==top[v]){
            if(No[u]>No[v])swap(u,v);
            if(No[u]<No[v])ret+=query(root[u],1,slen[u],No[u]+1,No[v]);
            return ret; }
        if(depth[top[u]]>depth[top[v]])swap(u,v);
        ret+=query(root[v],1,slen[v],1,No[v]);
        v=top[v];
        v=parent[v]; }}
```

<u>树分治[点分治]</u>

```
const int N=100011;
struct Entry{
    int root,son,depth;
    Entry(){}
    Entry(int a,int b,int c):root(a),son(b),depth(c){} };
vector<Entry>ent[N];
int que[N],parent[N],depth[N],size[N],balance[N],n;
bool visit[N];
int split(int root){
    int i,j,u,v,Max,ed=-1;
    que[++ed]=root;
    parent[root]=root;
    size[root]=1;
    for(i=0;i<=ed;i++){
        u=que[i];
        for(j=last[u];j!=-1;j=edge[j].next){
            v=edge[j].v;
            if(visit[v] || parent[u]==v)continue;
            parent[v]=u;
            size[v]=1;
            que[++ed]=v; }}
    for(i=ed;i>0;i--)size[parent[que[i]]]+=size[que[i]];
    for(i=0;i<=ed;i++)balance[que[i]]=size[root]-size[que[i]];
    for(i=ed;i>0;i--)cmax(balance[parent[que[i]]],size[que[i]]);
    for(i=0;i<=ed;i++)if(balance[root]>balance[que[i]])root=que[i];
    visit[root]=1;
    depth[root]=0;
    ent[root].PB(Entry(root,-1,0));
    for(int _=last[root];_!=-1;_=edge[_].next){
        v=edge[_].v;
        if(visit[v])continue;
        parent[v]=v;
        depth[v]=edge[_].c;
        ed=-1;
        que[++ed]=v;
        for(i=0;i<=ed;i++){
            u=que[i];
            for(j=last[u];j!=-1;j=edge[j].next){
                int vv=edge[j].v;
                if(parent[u]==vv || visit[vv])continue;
                depth[vv]=depth[u]+edge[j].c;
                parent[vv]=u;
                que[++ed]=vv; }}
        for(i=0;i<=ed;i++)ent[que[i]].PB(Entry(root,v,depth[que[i]]));
        split(v); }
    visit[root]=0;
    return root; }
```

<u>Link-Cut-Tree[SJTU]</u>

```
void rotate(int x) {
    int t = type[x];
    int y = parent[x];
    int z = children[x][1 ^ t];
    type[x] = type[y];
    parent[x] = parent[y];
    if (type[x] != 2)children[parent[x]][type[x]] = x;
    type[y] = 1 ^ t;
```

```
    parent[y] = x;
    children[x][1 ^ t] = y;
    if (z != 0) {
        type[z] = t;
        parent[z] = y; }
    children[y][t] = z;
    update(y); }
void splay(int x) {
    vector <int> stack(1, x);
    for (int i = x; type[i] != 2; i = parent[i])
        stack.push_back(parent[i]);
    while (!stack.empty()) {
        push(stack.back());
        stack.pop_back(); }
    while (type[x] != 2) {
        int y = parent[x];
        if (type[x] == type[y])rotate(y);
        else rotate(x);
        if (type[x] == 2)break;
        rotate(x); }
    update(x); }
void access(int x) {
    int z = 0;
    while (x != 0) {
        splay(x);
        type[children[x][1]] = 2;
        children[x][1] = z;
        type[z] = 1;
        update(x);
        z = x ;
        x = parent[x]; }}
```

Tips-Ryan

1) Anti-SG 先手必胜当且仅当：（1）游戏的 SG 函数不为 0 且游戏中某个单一游戏的 SG 函数大于 1；（2）游戏的 SG 函数为 0 且游戏中没有单一游戏的 SG 函数大于 1。
2) Every-SG(每个游戏同时同时进行)

$$\text{定义step}(v)=\begin{cases}0,\ v\text{ 为终止状态}\\ \max\big(step(u)\big)+1,\ SG(v)>0\text{ 且 }SG(u)=0\\ \min\big(\mathrm{step}(u)\big)+1,\ SG(v)=0\end{cases}$$

先手必胜当且仅当单一游戏中最大的 step 为奇数

3) STL 优先队列（堆）的运算符重载（推荐）

```
struct cmp {
    bool operator() (const int &a, const int &b){
      return a>b; }};
priority_queue<int, vector<int>, cmp> Q;
```

数表

n	log2_(n)	n!	C(n,n/2)	Bell[n]	划分数
2	1	2	2	2	2
3	1.58	6	3	5	3
4	2	24	6	15	5
5	2.32	120	10	52	7
6	2.58	720	20	203	11
7	2.81	5040	35	877	15
8	3	40320	70	4140	22
9	3.17	362880	126	21147	30
10	3.32	3628800	252	115975	42
11		39916800	462	678570	56
12		479001600	924	4213597	77
15			6435	1382958545	176
20			184756		627
25			5200300		1958
30			155117520		5604
40					37338
50					204226
70					4087968
100					190569292