

Code Template for ACM-ICPC

jasison @ PolyU

May 30, 2016

Contents

1	Graph	1
1.1	Dinic	1
1.2	KM	2
1.3	Mixed Euler Circuit	3
2	Tree	4
2.1	Divide And Conquer Tree	4
2.2	Link Tree	7
2.3	Segment Tree	9
2.4	Splay Tree	13
2.5	Treap	15
3	Geometry	17
3.1	Basic Struct and Algorithm	17
3.2	Polygon Area	17
3.3	Half Plane Intersection	18
4	Math	18
4.1	China Remainder Theory	18
4.2	China Remainder Theory	19
4.3	China Remainder Theory	20
4.4	China Remainder Theory	20
4.5	China Remainder Theory	20
4.6	China Remainder Theory	21
4.7	China Remainder Theory	21
4.8	China Remainder Theory	21
4.9	China Remainder Theory	21
4.10	China Remainder Theory	21
4.11	China Remainder Theory	21
4.12	China Remainder Theory	22
4.13	China Remainder Theory	22
4.14	China Remainder Theory	22
4.15	China Remainder Theory	22
4.16	China Remainder Theory	22
4.17	China Remainder Theory	23
4.18	China Remainder Theory	23
4.19	China Remainder Theory	23
4.20	China Remainder Theory	23
4.21	China Remainder Theory	23
4.22	China Remainder Theory	23
4.23	China Remainder Theory	24
4.24	China Remainder Theory	24
4.25	China Remainder Theory	24
4.26	China Remainder Theory	24
4.27	China Remainder Theory	24
4.28	China Remainder Theory	24
4.29	China Remainder Theory	25
4.30	China Remainder Theory	25
4.31	China Remainder Theory	25
4.32	China Remainder Theory	25
4.33	China Remainder Theory	25
4.34	China Remainder Theory	26
4.35	China Remainder Theory	27

4.36	China Remainder Theory	27
4.37	China Remainder Theory	27
4.38	China Remainder Theory	28
4.39	China Remainder Theory	28
4.40	China Remainder Theory	28
4.41	China Remainder Theory	29
5	Others	30
5.1	Exact Cover	30
5.2	Matrix Fast Power	32
5.3	Polynomial	32

1 Graph

1.1 Dinic

```
const int inf = 0x3f3f3f3f;
const int N = 205;
const int M = 1205;
struct Edge{
    int v,f,nxt;
};
struct Dinic{
    int src,sink;
    int g[N],en;
    Edge e[M*2];
    int level[N];
    void _addEdge(int u,int v,int f){
        e[en].v=v;
        e[en].f=f;
        e[en].nxt=g[u];
        g[u]=en++;
    }
    void addEdge(int u,int v,int f){
        _addEdge(u,v,f);
        _addEdge(v,u,0);
    }
    void init(){
        en=0;
        memset(g,-1,sizeof(g));
    }
    int q[N],front,rear;
    bool bfs(){
        memset(level,0,sizeof(level));
        level[src]=1;
        front=0; rear=1;
        q[0]=src;
        while(front<rear){
            int u=q[front++];
            if(u==sink)return 1;
            for(int i=g[u];i!=-1;i=e[i].nxt){
                int v=e[i].v,f=e[i].f;
                if(!level[v]&&f){
                    level[v]=level[u]+1;
                    q[rear++]=v;
                }
            }
        }
        return 0;
    }
    int dfs(int u,int delta){
        if(u==sink || delta==0)return delta;
        int ret=0;
        for(int i=g[u];i!=-1;i=e[i].nxt){
            int v=e[i].v, f=e[i].f;
            if(level[v]==level[u]+1&&f){
                int minf=min(delta-ret,f);
                f=dfs(v,minf);
                e[i].f-=f;
                e[i^1].f+=f;
            }
        }
        return ret;
    }
};
```

```

        delta-=f;
        ret+=f;
        if(ret==delta)return ret;
    }
}
return ret;
}
int maxflow(int _src,int _sink){
    src=_src;
    sink=_sink;
    int ret=0;
    while(bfs())ret+=dfs(src,inf);
    return ret;
}
}dicnic_solver;

```

1.2 KM

```

// eg: soj 1013
const int N = 105;
const int inf = 1000000000;
struct KM{
    int w[N][N],x[N],y[N];
    int px[N],py[N],sy[N],sk[N],pr[N];
    int lx,ly,n;
    void adjust(int v){
        sy[v]=py[v];
        if(px[sy[v]]!=-2)adjust(px[sy[v]]);
    }
    int solve(int _n,int _w[][N]){
        n=_n;
        memcpy(w,_w,sizeof(w));
        return km();
    }
    bool find(int v){
        for(int i=0;i<n;++i)if(py[i]==-1){
            if(sk[i]>x[v]+y[i]-w[v][i]){
                sk[i]=x[v]+y[i]-w[v][i];
                pr[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;
    }
    int km(){
        int i,j,m;
        for(i=0;i<n;++i){
            sy[i]=-1;

```

```

        y[i]=0;
    }
    for(i=0;i<n;++i){
        x[i]=0;
        for(j=0;j<n;++j){
            x[i]=max(x[i],w[i][j]);
        }
    }
    bool f;
    for(i=0;i<n;++i){
        for(j=0;j<n;++j){
            px[j]=py[j]=-1;
            sk[j]=inf;
        }
        px[i]=-2;
        if(find(i))continue;
        f=0;
        while(!f){
            m=inf;
            for(j=0;j<n;++j)if(py[j]==-1)m=min(m,sk[j]);
            for(j=0;j<n;++j){
                if(px[j]==-1)x[j]-=m;
                if(py[j]==-1)y[j]+=m;else sk[j]-=m;
            }
            for(j=0;j<n;++j)if(py[j]==-1&&!sk[j]){
                py[j]=pr[j];
                if(sy[j]==-1){
                    adjust(j);
                    f=1;
                    break;
                }
                px[sy[j]]=j;
                if(find(sy[j])){
                    f=1;
                    break;
                }
            }
        }
    }
    int ans=0;
    for(i=0;i<n;++i)ans+=w[sy[i]][i];
    return ans;
}
}km_solver;

```

1.3 Mixed Euler Circuit

```

// eg: soj 1066
const int N = 205;
int degree[N],n;
void init(){
    dicnic_solver.init();
    int m,a,b,c;
    scanf("%d%d",&n,&m);
    memset(degree,0,sizeof(degree));
    while(m--){
        // c=0,a<->b; c=1,a->b

```

```

        scanf("%d%d%d", &a, &b, &c);
        a--; b--;
        degree[a]--;
        degree[b]++;
        if(!c) dicnic_solver.addEdge(a, b, 1);
    }
}

bool work(){
    int ans=0;
    for(int i=0; i<n; ++i) if(degree[i]&1) return 0;
    for(int i=0; i<n; ++i){
        if(degree[i]<0){
            dicnic_solver.addEdge(n, i, -degree[i]/2);
            ans-=degree[i]/2;
        } else if(degree[i]>0){
            dicnic_solver.addEdge(i, n+1, degree[i]/2);
        }
    }
    return dicnic_solver.maxflow(n, n+1) >= ans;
}

void solve(){
    puts(work()? "possible": "impossible");
}

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

```

2 Tree

2.1 Divide And Conquer Tree

```

//hdu 4812 D Tree
#include <iostream>
#include <cstdio>
#include <cstring>
#include <vector>
#pragma comment(linker, "/STACK:102400000,102400000")
using namespace std;
const int maxn = 1e5 + 10;
const int md = 1e6 + 3;
int N, K;
vector<int> edge[maxn];
void add_edge(int from, int to) {
    edge[from].push_back(to);
}

void init() {
    for(int i = 1; i <= N; i++) edge[i].clear();
}

int vi[maxn];
int vis[maxn];

```

```

int root;
int mi;
int son[maxn];
int hash[md + 10];
int vers[md + 10];
int verc;
pair<int , int > ans;
int fastpow(int x,int y) {
    int ret = 1 ,mul = x;
    while(y) {
        if(y & 1 ) ret = 1LL * mul * ret % md;
        mul = 1LL * mul* mul % md;
        y >>= 1;
    }
    return ret;
}
int comm[md + 10];
void inv1() {
    for(int i = 0;i < md;i ++) {
        comm[i] = fastpow(i,md - 2);
    }
}
int inv(int t) {
    return comm[t];
}
void getroot(int t,int sz) {
    vis[t] = true;
    son[t] = 1;
    int mx = 0;
    for(int i = 0;i < edge[t].size();i ++) {
        int nxt = edge[t][i];
        if(!vis[nxt]) {
            getroot(nxt,sz);
            son[t] += son[nxt];
            mx = max(mx,son[nxt]);
        }
    }
    mx = max(mx,sz - son[t]);
    if(mx <= mi) {
        root = t;
        mi = mx;
    }
    vis[t] = false;
}
void dfs(int t,int mul,int ri) {
    vis[t] = true;
    //query
    mul =1LL * mul * vi[t] % md;
    if(1LL * mul * ri % md == K) {
        pair<int ,int > tmp = pair<int ,int > (min(root,t),max(root,t));
        if(tmp < ans) ans = tmp;
    }
    int q = 1LL* inv(1LL * mul * ri % md) * K % md;
    if(vers[q] == verc && hash[q] != 0 ) {
        pair<int ,int > tmp = pair<int ,int > (min(t,hash[q]),max(t,hash[q]));
        if(tmp < ans) ans = tmp;
    }
    son[t] = 1;
    for(int i = 0;i < edge[t].size();i ++) {

```



```

    int nxt = edge[t][i];
    if(!vis[nxt]) {
        dfs(nxt,mul,ri);
        son[t] += son[nxt];
    }
}
//set
if(vers[mul] != verc ) {
    vers[mul] = verc;
    hash[mul] = t;
}
hash[mul] = min(hash[mul],t);
vis[t] = false;
}
void work(int t,int sz) {
    mi = sz;
    getroot(t,sz);
    // dfs
    int rt = root;
    vis[rt] = true;
    verc ++;
    for(int i = 0;i < edge[root].size();i ++) {
        int nxt = edge[rt][i];
        if(!vis[nxt]) {
            dfs(nxt,1,vi[rt] % md);
        }
    }
    for(int i = 0;i < edge[rt].size();i ++) {
        int nxt = edge[rt][i];
        if(!vis[nxt]) {
            work(nxt,son[rt]);
        }
    }
}
int main() {
    inv1();
    verc = 0;
    while(scanf("%d%d",&N,&K) != EOF) {
        init();
        for(int i = 1;i <= N;i ++) {
            scanf("%d",&vi[i]);
        }
        for(int i = 0;i < N - 1;i ++) {
            int u,v;
            scanf("%d%d",&u,&v);
            add_edge(u,v);
            add_edge(v,u);
        }
        memset(vis,0,sizeof(vis));
        ans = pair<int ,int > (N+1,N+1);
        work(1,N);
        if(ans.first == N+1 && ans.second == N + 1) {
            puts("No solution");
        } else {
            printf("%d %d\n",ans.first,ans.second);
        }
    }
}

```

2.2 Link Tree

```
//HDU 3966
//operation1 path c1 to c2 plus k
//operation2 path c1 to c2 minus k
#include <iostream>
#include <cstdio>
#include <algorithm>
#include <vector>
#include <cstring>
#pragma comment(linker, "/STACK:1024000000,1024000000")
using namespace std;
#define lc (o<<1)
#define rc (o<<1|1)
int N,M,P;
const int maxn = 100010;
vector<int > edge[maxn];
int ai[maxn];
void add_edge(int from,int to) {
    edge[from].push_back(to);
}
void init() {
    for(int i = 1;i <= N;i ++) edge[i].clear();
}
int son[maxn]; // size of children
int fa[maxn];
int wn[maxn]; //index in segment
int wcnt;
int vis[maxn];
int dep[maxn]; // depth
int top[maxn]; // link fa
//Tree link
void dfs1(int t,int d) {
    vis[t] = true;
    dep[t] = d;
    son[t] = 1;
    for(int i = 0;i < edge[t].size();i ++) {
        int nxt = edge[t][i];
        if(!vis[nxt]) {
            fa[nxt] = t;
            dfs1(nxt,d + 1);
            son[t] += son[nxt];
        }
    }
    vis[t] = false;
}
void dfs2(int t) {
    vis[t] = true;
    wn[t] = wcnt ++;
    bool first = true;
    int index = -1;
    for(int i = 0;i < edge[t].size();i ++) {
        int nxt = edge[t][i];
        if(!vis[nxt]) {
            if(first) {
                first = false;
                index = nxt;
            }
        }
    }
}
```

```

        if(son[nxt] > son[index]) {
            index= nxt;
        }
    }
}
if(!first ) {
    top[index] = top[t];
    dfs2(index);
    for(int i = 0;i < edge[t].size();i ++) {
        int nxt = edge[t][i];
        if(!vis[nxt] && nxt != index) {
            top[nxt] = nxt;
            dfs2(nxt);
        }
    }
}
vis[t] = false;
}
//segment tree
int addv[maxn << 2];
void add(int o,int l,int r,int y1,int y2,int v) {
    if(y1 <= l && r <= y2) {
        addv[o] += v;
    } else {
        int m = (l + r) >> 1;
        if(y1 <= m) add(lc,l,m,y1,y2,v);
        if(m < y2) add(rc,m+1,r,y1,y2,v);
    }
}
void query(int o,int l,int r,int x,int & ans) {
    if(l == r && r == x) {
        ans += addv[o];
    } else {
        int m = (l + r ) >> 1;
        ans += addv[o];
        if(x <= m ) {
            query(lc,l,m,x,ans);
        } else {
            query(rc,m+1,r,x,ans);
        }
    }
}
void init_seg() {
    memset(addv,0,sizeof(addv));
}
char buff[5];
int main() {
    while(~scanf("%d%d%d",&N,&M,&P) ) {
        init();
        for(int i = 1;i <= N;i ++) {
            scanf("%d",&ai[i]);
        }
        for(int i = 0;i < M;i ++) {
            int u,v;
            scanf("%d%d",&u,&v);
            add_edge(u,v);
            add_edge(v,u);
        }
        dfs1(1,1);
    }
}

```

```

wcnt = 0;
top[1] = 1;
dfs2(1);
init_seg();
while(P --) {
    scanf("%s",buff);
    if(buff[0] == 'I' || buff[0] == 'D') {
        int c1,c2,k;
        scanf("%d%d%d",&c1,&c2,&k);
        if(buff[0] == 'D') k = - k;
        /// query path
        while(top[c1] != top[c2]) {
            int f1 = top[c1];
            int f2 = top[c2];
            if(dep[f1] < dep[f2]) {
                swap(f1,f2);
                swap(c1,c2);
            }
            add(1,0,N - 1,wn[f1],wn[c1],k);
            c1 = fa[f1];
        }
        if(dep[c1] < dep[c2]) {
            swap(c1,c2);
        }
        add(1,0,N - 1,wn[c2],wn[c1],k);
    } else if(buff[0] == 'Q') {
        int d;
        scanf("%d",&d);
        int ans = 0;
        query(1,0,N-1,wn[d],ans);
        ans += ai[d];
        printf("%d\n",ans);
    }
}
}
}

```

2.3 Segment Tree

```

//HDU 4578
//segment plus mul power sum
#include <cstdio>
#include <algorithm>
using namespace std;
#define lc (o<<1)
#define rc (o<<1|1)
const int maxn = 100010;
const int md = 10007;
int sumv1[maxn<<2], sumv2[maxn<<2], sumv3[maxn<<2];
int addv[maxn<<2], setv[maxn<<2], timv[maxn<<2];
void pushdown(int o) {
    if (setv[o] >= 0) {
        setv[lc] = setv[rc] = setv[o];
        addv[lc] = addv[rc] = 0;
        timv[lc] = timv[rc] = 1;
        setv[o] = -1;
    }
}

```

```

if (timv[o] != 1) {
    addv[lc] *= timv[o];
    addv[lc] %= md;
    addv[rc] *= timv[o];
    addv[rc] %= md;
    timv[lc] *= timv[o];
    timv[lc] %= md;
    timv[rc] *= timv[o];
    timv[rc] %= md;
    timv[o] = 1;
}
if (addv[o] > 0) {
    addv[lc] += addv[o];
    addv[lc] %= md;
    addv[rc] += addv[o];
    addv[rc] %= md;
    addv[o] = 0;
}
}
void maintain(int o, int l, int r) {
    if (l == r) {
        if (setv[o] != -1) {
            sumv1[o] = setv[o];
            setv[o] = -1;
        }
        if (timv[o] != 1) {
            sumv1[o] *= timv[o];
            timv[o] = 1;
            sumv1[o] %= md;
        }
        if (addv[o] > 0) {
            sumv1[o] += addv[o];
            sumv1[o] %= md;
            addv[o] = 0;
        }
        sumv2[o] = sumv1[o] * sumv1[o] % md;
        sumv3[o] = sumv1[o] * sumv2[o] % md;
    } else {
        sumv1[o] = (sumv1[lc] + sumv1[rc]) % md;
        sumv2[o] = (sumv2[lc] + sumv2[rc]) % md;
        sumv3[o] = (sumv3[lc] + sumv3[rc]) % md;
        if (setv[o] != -1) {
            sumv1[o] = setv[o] * (r - l + 1) % md;
            sumv2[o] = setv[o] * setv[o] % md * (r - l + 1) % md;
            sumv3[o] = setv[o] * setv[o] % md * setv[o] % md * (r - l + 1) % md;
        }
        if (timv[o] != 1) {
            sumv1[o] *= timv[o];
            sumv1[o] %= md;
            sumv2[o] *= timv[o] * timv[o] % md;
            sumv2[o] %= md;
            sumv3[o] *= timv[o] * timv[o] % md * timv[o] % md;
            sumv3[o] %= md;
        }
        if (addv[o] > 0) {
            int tmp1 = sumv1[o];
            sumv1[o] += addv[o] * (r - l + 1) % md;
            sumv1[o] %= md;
            int tmp2 = sumv2[o];

```

```

        int tmp3 = sumv3[o];
        sumv2[o] = (tmp2 + 2*tmp1%md * addv[o]%md + addv[o] * addv[o] %md* (r - l + 1)%md) % md;
        sumv3[o] = tmp3 + 3 * tmp2%md * addv[o] % md + 3 * tmp1 % md * addv[o]%md * addv[o] % md +
            addv[o] * addv[o] % md * addv[o] % md * (r - l + 1) %md;
        sumv3[o] %= md;
    }
}

void setq(int o,int l,int r,int y1,int y2,int v) {
    if (y1 <= l && r <= y2) {
        setv[o] = v;
        addv[o] = 0;
        timv[o] = 1;
    } else {
        pushdown(o);
        int m = (l + r) >> 1;
        if (y1 <= m) setq(lc,l,m,y1,y2,v);
        else maintain(lc,l,m);
        if (m < y2) setq(rc,m+1,r,y1,y2,v);
        else maintain(rc,m+1,r);
    }
    maintain(o,l,r);
}

void addq(int o,int l,int r,int y1,int y2,int v) {
    if (y1 <= l && r <= y2) {
        addv[o] += v;
        addv[o] %= md;
    } else {
        pushdown(o);
        int m = (l + r) >> 1;
        if (y1 <= m ) addq(lc,l,m,y1,y2,v);
        else maintain(lc,l,m);
        if (m < y2) addq(rc,m+1,r,y1,y2,v);
        else maintain(rc,m+1,r);
    }
    maintain(o,l,r);
}

void timq(int o,int l,int r,int y1,int y2,int v) {
    if (y1 <= l && r <= y2) {
        timv[o] *= v;
        timv[o] %= md;
        addv[o] *= v;
        addv[o] %= md;
    } else {
        pushdown(o);
        int m = (l + r) >> 1;
        if (y1 <= m) timq(lc,l,m,y1,y2,v);
        else maintain(lc,l,m);
        if (m < y2) timq(rc,m+1,r,y1,y2,v);
        else maintain(rc,m+1,r);
    }
    maintain(o,l,r);
}

int ans1, ans2, ans3;
void query(int o,int l,int r,int y1,int y2,int add,int ti) {
    if (setv[o] > 0) {
        add = ti * addv[o] % md + add;
        ti = ti * timv[o] % md;
        int len = min(r,y2) - max(y1,l) + 1;

```

```

int tmp1 = setv[o] * len % md * ti % md;
int tmp2 = setv[o] * setv[o] % md * len % md * ti % md * ti % md;
int tmp3 = setv[o] * setv[o] % md * setv[o] % md * len % md * ti % md * ti % md * ti % md;
int _sum1 = tmp1 + add * len % md;
_sum1 %= md;
int _sum2 = (tmp2 + 2 * tmp1 * add % md + add * add % md * len % md) % md;
int _sum3 = (tmp3 + 3 * tmp2 * add % md + 3 * tmp1 * add % md * add % md + len * add % md *
    add % md * add % md) % md;
ans1 = (ans1 + _sum1) % md;
ans2 = (ans2 + _sum2) % md;
ans3 = (ans3 + _sum3) % md;
return ;
}
if (y1 <= 1 && r <= y2) {
    int tmp1 = sumv1[o] * ti % md;
    int tmp2 = sumv2[o] * ti % md * ti % md;
    int tmp3 = sumv3[o] * ti % md * ti % md * ti % md;
    int _sum = tmp1 + add * (r - 1 + 1) % md;
    int _sum2 = tmp2 + 2 * tmp1 * add % md + add * add % md * (r - 1 + 1) % md;
    int _sum3 = tmp3 + 3 * tmp2 % md * add % md + 3 * tmp1 % md * add % md * add % md + add * add
        % md * add % md * (r - 1 + 1) % md;
    _sum %= md;
    _sum2 %= md;
    _sum3 %= md;
    ans1 = (ans1 + _sum) % md;
    ans2 = (ans2 + _sum2) % md;
    ans3 = (ans3 + _sum3) % md;
} else {
    int m = (1 + r) >> 1;
    if (y1 <= m) query(lc, l, m, y1, y2, (ti * addv[o] % md + add) % md, ti * timv[o] % md);
    if (m < y2) query(rc, m + 1, r, y1, y2, (ti * addv[o] % md + add) % md, ti * timv[o] % md);
}
}
void init(int o, int l, int r) {
    setv[o] = -1;
    timv[o] = 1;
    addv[o] = 0;
    sumv1[o] = sumv2[o] = sumv3[o] = 0;
    if (l == r) {
    } else {
        int m = (1 + r) >> 1;
        init(lc, l, m);
        init(rc, m + 1, r);
    }
}
int main() {
    int N, M;
    while (scanf("%d%d", &N, &M) == 2 && N && M) {
        init(1, 1, N);
        while (M--) {
            int cmd, x, y, c;
            scanf("%d%d%d", &cmd, &x, &y, &c);
            if (cmd == 1) {
                c %= md;
                addq(1, 1, N, x, y, c);
            } else if (cmd == 2) {
                c %= md;
                timq(1, 1, N, x, y, c);
            }
        }
    }
}

```

```

    } else if(cmd == 3) {
        c %= md;
        setq(1,1,N,x,y,c);
    } else if(cmd == 4) {
        ans1 = ans2 = ans3 = 0;
        query(1,1,N,x,y,0,1);
        if(c == 1) {
            printf("%d\n",ans1);
        } else if(c == 2){
            printf("%d\n",ans2);
        } else if(c == 3) {
            printf("%d\n",ans3);
        }
    }
}
}
}
}

```

2.4 Splay Tree

```

#include <cstdio>
#include <iostream>
using namespace std;
struct Node {
    Node* ch[2];
    int v, s, flip;
    void maintain() {
        s = 1 + ch[0]->s + ch[1]->s;
    }
    void pushdown() {
        if (flip) {
            flip = 0;
            swap(ch[0], ch[1]);
            ch[0]->flip ^= 1;
            ch[1]->flip ^= 1;
        }
    }
    int cmp(int k) const {
        int d = k - ch[0]->s;
        if (d == 1) return -1;
        return d <= 0 ? 0 : 1;
    }
};
Node* null = new Node();
void rotate(Node* &o, int d) {
    Node* k = o->ch[d^1];
    o->ch[d^1] = k->ch[d];
    k->ch[d] = o;
    o->maintain();
    k->maintain();
    o = k;
}
void splay(Node* &o, int k) {
    o->pushdown();
    int d = o->cmp(k);
    if (d == 1) k -= o->ch[0]->s + 1;
    if (d != -1) {

```



```

    Node* p = o->ch[d];
    p->pushdown();
    int d2 = p->cmp(k);
    int k2 = (d2 == 0) ? k : k - p->ch[0]->s - 1;
    if (d2 != -1) {
        splay(p->ch[d2], k2);
        if (d == d2) {
            rotate(o, d^1);
        } else {
            rotate(o->ch[d], d);
        }
    }
    rotate(o, d^1);
}
}
Node* merge(Node* left, Node* right) { // make sure left != null
    splay(left, left->s);
    left->ch[1] = right;
    left->maintain();
    return left;
}
void split(Node* o, int k, Node* &left, Node* &right) { // make sure 1 <= k <= o->s
    splay(o, k);
    left = o;
    right = o->ch[1];
    o->ch[1] = null;
    left->maintain();
}
const int maxn = 300000 + 10;
struct SS {
    int n;
    Node seq[maxn];
    Node* root;
    Node* build(int sz) {
        if (!sz) return null;
        Node* L = build(sz/2);
        Node* o = &seq[++n];
        o->v = n-1;
        o->flip = 0;
        o->ch[0] = L;
        o->ch[1] = build(sz - sz/2 - 1);
        o->maintain();
        return o;
    }
    void init(int sz) {
        n = 0;
        null->s = null->flip = 0;
        root = build(sz);
    }
    void print(Node *o) {
        if (o != null) {
            o->pushdown();
            print(o->ch[0]);
            if (o->v) {
                if (o->v != 1) putchar(' ');
                printf("%d", o->v);
            }
            print(o->ch[1]);
        }
    }
}

```

```

    }
} ss;
int n, m, a, b, c;
char op[10];
int main() {
    while (scanf("%d%d",&n,&m) == 2 && n != -1 && m != -1) {
        ss.init(n+1);
        Node *t1, *t2, *t3;
        while(m--){
            scanf("%s",op);
            if(op[0]=='C'){ // split [a,b], put it after c
                scanf("%d%d%d",&a,&b,&c);
                split(ss.root, b+1, t1, t2);
                split(t1, a, t1, t3);
                ss.root = merge(t1, t2);
                split(ss.root, c+1, t1, t2);
                ss.root = merge(merge(t1, t3), t2);
            } else { // flip [a,b]
                scanf("%d%d",&a,&b);
                split(ss.root, b+1, t1, t3);
                split(t1, a, t1, t2);
                t2->flip ^= 1;
                ss.root = merge(merge(t1, t2), t3);
            }
        }
        ss.print(ss.root);
        puts("\n");
    }
}

```

2.5 Treap

```

struct Node {
    Node *ch[2]; // 0-left 1-right
    int r, v, s; // rank, val, #node
    Node(int v): v(v) {
        ch[0] = ch[1] = NULL;
        r = rand();
        s = 1;
    }
    int cmp(int x) const {
        if (x == v) return -1;
        return x < v ? 0 : 1;
    }
    void maintain() { // maintain #node
        s = 1;
        if (ch[0] != NULL) s += ch[0]->s;
        if (ch[1] != NULL) s += ch[1]->s;
    }
};

void rotate(Node* &o, int d) {
    Node* k = o->ch[d^1];
    o->ch[d^1] = k->ch[d];
    k->ch[d] = o;
    o->maintain();
    k->maintain();
    o = k;
}

```

```

}
void insert(Node* &o, int x) {
    if (o == NULL) {
        o = new Node(x);
    } else {
        int d = o->cmp(x);
        if (d != -1) { // same ele won't be inserted
            insert(o->ch[d], x);
            if (o->ch[d]->r > o->r) rotate(o, d^1);
        }
    }
    o->maintain();
}

void remove(Node* &o, int x) {
    if (o == NULL) return ; // ele to be removed not exist
    int d = o->cmp(x);
    if (d == -1) {
        Node* ret = o;
        if (o->ch[0] != NULL && o->ch[1] != NULL) {
            int d2 = (o->ch[0]->r > o->ch[1]->r ? 1 : 0);
            rotate(o, d2);
            remove(o->ch[d2], x);
        } else {
            if (o->ch[0] == NULL) o = o->ch[1];
            else o = o->ch[0];
            delete ret;
        }
    } else {
        remove(o->ch[d], x);
    }
    if (o) o->maintain();
}

int find(Node* o, int x) {
    while (o != NULL) {
        int d = o->cmp(x);
        if (d == -1) return 1;
        else o = o->ch[d];
    }
    return 0;
}

int kth_big(Node* o, int k) {
    if (o == NULL || k <= 0 || k > o->s) return 0;
    int s = o->ch[1] == NULL ? 0 : o->ch[1]->s;
    if (k == s+1) return o->v;
    else if (k <= s) return kth_big(o->ch[1], k);
    else return kth_big(o->ch[0], k-s-1);
}

int kth_small(Node* o, int k) {
    if (o == NULL || k <= 0 || k > o->s) return 0;
    int s = o->ch[0] == NULL ? 0 : o->ch[0]->s;
    if (k == s) return o->v;
    else if (k < s) return kth_small(o->ch[0], k);
    else return kth_small(o->ch[1], k-s-1);
}

void merge(Node* &src, Node* &dest) {
    if (src == NULL) return ;
    merge(src->ch[0], dest);
    merge(src->ch[1], dest);
    insert(dest, src->v);
}

```

```

    delete src;
    src = NULL;
}
void clear(Node* &o) {
    if (o == NULL) return ;
    clear(o->ch[0]);
    clear(o->ch[1]);
    delete o;
    o = NULL;
}

```

3 Geometry

3.1 Basic Struct and Algorithm

```

struct Point {
    double x, y;
    Point(double x=0, double y=0):x(x),y(y){}
};

typedef Point Vector;

Vector operator + (const Vector &A, const Vector &B) { return Vector(A.x+B.x, A.y+B.y); }
Vector operator - (const Point &A, const Point &B) { return Vector(A.x-B.x, A.y-B.y); }
Vector operator * (const Vector &A, double p) { return Vector(A.x*p, A.y*p); }
double Dot(const Vector &A, const Vector &B) { return A.x*B.x + A.y*B.y; }
double Cross(const Vector &A, const Vector &B) { return A.x*B.y - A.y*B.x; }
double Length(const Vector &A) { return sqrt(Dot(A, A)); }
Vector Normal(const Vector &A) { double L = Length(A); return Vector(-A.y/L, A.x/L); }

struct Line {
    Point P;
    Vector v;
    double ang;
    Line() {}
    Line(Point P, Vector v):P(P),v(v){ ang = atan2(v.y, v.x); }
    bool operator < (const Line &L) const {
        return ang < L.ang;
    }
};

// if $p$ is on the left side of $L$
bool OnLeft(const Line &L, const Point &p) {
    return Cross(L.v, p-L.P) > 0;
}

// intersection of line $a$ and $b$
Point GetLineIntersection(const Line &a, const Line &b) {
    Vector u = a.P-b.P;
    double t = Cross(b.v, u) / Cross(a.v, b.v);
    return a.P+a.v*t;
}

```

3.2 Polygon Area

```
double PolygonArea(vector<Point> p) {
    int n = p.size();
    double area = 0;
    for(int i = 1; i < n-1; i++)
        area += Cross(p[i]-p[0], p[i+1]-p[0]);
    return area/2;
}
```

3.3 Half Plane Intersection

```
const double eps = 1e-6;
// intersection of areas (leftside of lines)
vector<Point> HalfplaneIntersection(vector<Line> L) {
    int n = L.size();
    sort(L.begin(), L.end());
    int first, last;
    vector<Point> p(n);
    vector<Line> q(n);
    vector<Point> ans;
    q[first=last=0] = L[0];
    for(int i = 1; i < n; i++) {
        while(first < last && !OnLeft(L[i], p[last-1])) last--;
        while(first < last && !OnLeft(L[i], p[first])) first++;
        q[++last] = L[i];
        if(fabs(Cross(q[last].v, q[last-1].v)) < eps) {
            last--;
            if(OnLeft(q[last], L[i].P)) q[last] = L[i];
        }
        if(first < last) p[last-1] = GetLineIntersection(q[last-1], q[last]);
    }
    while(first < last && !OnLeft(q[first], p[last-1])) last--;
    if(last - first <= 1) return ans;
    p[last] = GetLineIntersection(q[last], q[first]);
    for(int i = first; i <= last; i++) ans.push_back(p[i]);
    return ans;
}
```

4 Math

4.1 China Remainder Theory

```
// china remainder theory, no matter whether gcd(m[i],m[j])=1
LL CRT(const vector<LL>&m, const vector<LL> &b){
    bool flag = false;
    LL x, y, i, d, result, a1, m1, a2, m2, Size = m.size();
    m1 = m[0], a1 = b[0];
    for(int i = 1; i < Size; i++){
        m2 = m[i], a2 = b[i];
        d = exgcd(m1, m2, x, y);
        if ((a2 - a1) % d != 0) flag = true;
        result = (mul_mod(x, (a2 - a1) / d, m2) % m2 + m2) % m2;
        LL tmp = m1;
        m1 = m1 / d * m2;
    }
}
```

```

    a1 = (a1 + mul_mod(tmp, result, m1)) % m1;
    a1 = (a1 % m1 + m1) % m1;
}
if (flag) return -1;
else return a1;
}

```

4.2 China Remainder Theory

```

// eg: poj 3471
const int maxn = 10000000;
const int maxp = 700000; // about maxn/log(maxn)
struct Factor{ // factor as p^num
    int p, num;
};
struct DeComposer {
    DeComposer() { gen_primes(); }
    bool vis[maxn+5];
    int pn, prime[maxp];
    void sieve() {
        int m = (int)sqrt(maxn+0.5);
        memset(vis,0,sizeof(vis));
        for(int i=2;i<=m;++i)if(!vis[i])
            for(int j=i*i;j<=maxn;j+=i)vis[j]=1;
    }
    void gen_primes() {
        sieve();
        pn = 0;
        for (int i = 2; i <= maxn; ++ i) {
            if (!vis[i]) prime[pn++] = i;
        }
    }
    int fcn;
    Factor fc[64]; // x = p1^a1 * p2^a2 * ...
    int fn, factor[maxp]; // all y satisfy y|x
    void decompose2(int x,int d){
        if(d==fcn){
            factor[fn++] = x;
        } else {
            for(int i = 0; i <= fc[d].num; ++ i) {
                decompose2(x, d+1);
                x /= fc[d].p;
            }
        }
    }
}
void decompose1(int x) {
    fcn = 0;
    for(int i = 0; i < pn && prime[i] * prime[i] <= x; ++ i) if (x % prime[i] == 0) {
        fc[fcn].p = prime[i];
        fc[fcn].num = 0;
        while(x % prime[i] == 0) {
            fc[fcn].num ++;
            x /= prime[i];
        }
        fcn ++;
    }
    if (x > 1) {

```

```

        fc[fcn].p = x;
        fc[fcn].num = 1;
        fcn ++;
    }
}
void decompose(int x){
    decompose1(x);
    fn = 0;
    decompose2(1,0);
}
} dc_solver;

```

4.3 China Remainder Theory

```

// #x that x<=n && gcd(x,n)==1
int euler_phi(int n) {
    int m = (int)sqrt(n+0.5);
    int 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;
}
int phi[maxn];
void phi_table(int 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);
            }
        }
        phi[i] += phi[i-1];
    }
}
}

```

4.4 China Remainder Theory

```

// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}

```

4.5 China Remainder Theory

```

// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
}

```

```
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }  
}
```

4.6 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum  
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){  
    if (a) { x = 0; y = 1; return a; }  
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }  
}
```

4.7 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum  
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){  
    if (a) { x = 0; y = 1; return a; }  
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }  
}
```

4.8 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum  
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){  
    if (a) { x = 0; y = 1; return a; }  
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }  
}
```

4.9 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum  
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){  
    if (a) { x = 0; y = 1; return a; }  
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }  
}
```

4.10 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum  
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){  
    if (a) { x = 0; y = 1; return a; }  
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }  
}
```

4.11 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.12 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.13 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.14 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.15 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.16 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.17 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.18 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.19 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.20 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.21 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.22 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.23 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.24 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.25 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.26 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.27 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.28 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.29 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.30 China Remainder Theory

```
// a * x + b * y = d, |x| + |y| get the minimum
LL exgcd(LL a, LL b, LL &d, LL &x, LL &y){
    if (a) { x = 0; y = 1; return a; }
    else { exgcd(b, a%b, d, y, x); y -= x*(a/b); }
}
```

4.31 China Remainder Theory

```
LL inv1(LL a, LL n) { // a-1 under n
    LL d, x, y;
    gcd(a,n,d,x,y);
    return d == 1 ? (x+n)%n : -1;
}
LL inv2(LL a, LL p) { // in case that p is a prime
    return pow_mod(a, p-2, p);
}
```

4.32 China Remainder Theory

```
// ax = b (mod n)
// let d = gcd(a,n), use exgcd to solve ax + ny = d
// if b|d, then there are #ans=d, otherwise, no solution
vector<LL> line_mod(LL a, LL b, LL n) {
    LL x, y;
    exgcd(a,n,x,y);
    vector<LL>ans;
    ans.clear();
    if(b%d==0){
        x%=n; x+=n; x%=n;
        ans.push_back(x*(b/d)%(n/d));
        for(LL i=1;i<d;++i){
            ans.push_back((ans[0]+i*n/d)%n);
        }
    }
    return ans;
}
```

4.33 China Remainder Theory

```
// eg: hdu 2815
```

```

// d*a^(x-c) = b (mod n), make sure that (a,n) = 1 and (d,n) = 1
map<LL,LL>f;
LL log_mod(LL a, LL b, LL n, LL c, LL d) {
    LL m, v, e=1, i, x, y, dd;
    m = ceil( sqrt(n + 0.5) );
    f.clear();
    f[1] = m;
    for(i = 1; i < m; ++ i) {
        e = e*a%n;
        if (!f[e]) f[e] = i;
    }
    e = (e*a)%n;
    for (i = 0; i < m; ++ i) {
        exgcd(d,n,dd,x,y);
        x = (x*b%n + n) % n;
        if (f[x]) {
            LL num = f[x];
            return c + i*m + (num==m ? 0 : num);
        }
        d = (d*e) % n;
    }
    return -1;
}
// a^x = b (mod n), no restriction
LL log_mod(LL a, LL b, LL n) {
    b%=n;
    LL c = 0, d = 1, t;
    while((t=__gcd(a,n))!=1){
        if(b%t) return -1;
        c++;
        n/=t;
        b/=t;
        d=d*a/t%n;
        if(d==b)return c;
    }
    return log_mod(a,b,n,c,d);
}

```

4.34 China Remainder Theory

```

// C(n,m) % p, make sure p is prime, p <= 10^5
// n = n[k] * p^k + n[k-1] * p^(k-1) + .. + n[0]
// m = m[k] * p^k + m[k-1] * p^(k-1) + .. + m[0]
// then, C(n,m) = C(n[k],m[k])*C(n[k-1],m[k-1])*...*C(n[0],m[0]) (mod p)
// C(n,m) = C(n%p, m%p) * C(n/p, m/p) (mod p)
// eg: hdu3037
LL Lucas(LL n, LL m, LL p) {
    LL ret = 1;
    while(n && m) {
        LL np = n%p, mp = m%p;
        if(np < mp) return 0;
        ret = ret * factorial(np) % p * reverse(factorial(mp), p) % p * reverse(factorial(np-mp), p) %
            p;
        n /= p;
        m /= p;
    }
    return ret;
}

```

```
}
```

4.35 China Remainder Theory

```
// prime test
bool Witness(LL n, LL a) {
    LL m = n-1, j = 0;
    while(!(m&1)) m >>= 1, j ++;
    LL ans = pow_mod(a, m, n);
    while (j --) {
        LL tmp = mul_mod(ans, ans, n);
        if (tmp == 1 && ans != 1 && ans != n-1) return 1;
        ans = tmp;
    }
    return ans != 1;
}
bool Miller_Rabin(LL n) {
    if (n < 2) return 0;
    if (n == 2) return 1;
    if (!(n&1)) return 0;
    for (int i = 0; i < max_test; ++ i) {
        ll a = rand() % (n-2) + 2;
        if (Witness(n,a)) return 0;
    }
    return 1;
}
```

4.36 China Remainder Theory

```
// x*y % n
LL mul_mod(LL x, LL y, LL n) {
    LL T = floor(sqrt(n) + 0.5);
    LL t = T * T - n;
    LL a = x / T, b = x % T;
    LL c = y / T, d = y % T;
    LL e = a * c / T, f = a * c % T;
    LL v = ((a*d + b*c) % n + e*t) % n;
    LL g = v / T, h = v % T;
    LL ret = (((f+g)*t % n + b*d) % n + h*T) % n;
    return (ret % n + n) % n;
}
```

4.37 China Remainder Theory

```
// get a factor of n in log(n)
LL Pollard_Rho(LL n, LL c=1) {
    LL i=1, k=2, x=rand()%(n-1)+1, y=x, d;
    while(1) {
        i++;
        x = (mul_mod(x,x,n)+c)%n;
        d=__gcd(n,y-x);
        if(d>1 && d<n) return d;
        if(y==x) return n;
    }
}
```

```

        if(i==k){
            k<<=1;
            y=x;
        }
    }
}

```

4.38 China Remainder Theory

```

// a^x % n
LL pow_mod(LL a, LL x, LL n) {
    LL ret = 1, mul = a;
    while (x) {
        if (x&1) ret = mul_mod(ret, mul, n);
        mul = mul_mod(mul, mul, n);
        x >>= 1;
    }
    return ret;
}

```

4.39 China Remainder Theory

```

// x^n = a (mod p), make sure that p is prime
// let g be a primitive root of p, x = g^y, a = g^m
// use log_mod to get m, g^(yn) = g^m (mod p)
// thus yn = m (mod p-1), use exgcd to solve and get back
vector<int> power_mod(int a, int n, int p) {
    int g = primitive_root(p);
    LL m = log_mod(g, a, p);
    vector<int> ret;
    if(a==0){
        ret.push_back(0);
        return ;
    }
    if(m==-1)return ret;
    LL A=n,B=p-1,C=m,x,y;
    LL d = exgcd(A,B,x,y);
    if(C%d!=0)return ret;
    x=x*(C/d)%B;
    LL delta=B/d;
    for(int i=0;i<d;++i){
        x=((x+delta)%B+B)%B;
        ret.push_back((int)pow_mod(g,x,p));
    }
    sort(ret.begin(),ret.end());
    ret.erase(unique(ret.begin(),ret.end()), ret.end());
    return ret;
}

```

4.40 China Remainder Theory

```

// eg: SGU 511
struct PR {

```

```

// make sure that p is prime
// if p = 2, solve the prob. without PR
int divs[N+5];
int primitive_root(const int p) {
    if (p == 2) return 1;
    int cnt = 0, m = p-1;
    for (int i = 2; i*i <= m; ++ i) if (m%i == 0) {
        divs[cnt++] = i;
        if (i*i < m) divs[cnt++] = m/i;
    }
    int r = 2, j = 0;
    while (1) {
        for (j = 0; j < cnt; ++ j) {
            if (fastpow(r, divs[j], p) == 1) break;
        }
        if (j >= cnt) return r;
        r ++;
    }
    return -1;
}
} pr_solver;

```

4.41 China Remainder Theory

```

// x*x = a (mod n), make sure that n is prime
// be careful there is a single sol. when n = 2
// otherwise, x and n-x are both okay
// eg: ural 1132
LL modsqr(LL a, LL n) {
    LL b, k, i, x;
    if (n == 2) return a % n;
    if (pow_mod(a, (n-1)/2, n) == 1) {
        if (n%4 == 3) {
            x = pow_mod(a, (n+1)/4, n);
        } else {
            for (b=1; pow_mod(b, (n-1)/2, n) == 1; b ++);
            i = (n-1)/2;
            k = 0;
            do {
                i/=2;
                k/=2;
                if ((pow_mod(a, i, n) * pow_mod(b, k, n) + 1) % n == 0) {
                    k += (n-1)/2;
                }
            } while (i%2 == 0);
            x = (pow_mod(a, (i+1)/2, n) * pow_mod(b, k/2, n)) % n;
        }
        if (x*2 > n) x = n-x;
        return x;
    }
    return -1;
}

```

5 Others

5.1 Exact Cover

```
// la 2659
#include <cstdio>
#include <vector>
using namespace std;
const int MROW = 16*16*16 + 5;
const int MCOL = 16*16*4 + 5;
const int NODE = 16*16*16*4 + 5;
struct DLX {
    int n, sz;
    int S[MCOL];
    int row[NODE], col[NODE];
    int ansd, ans[MROW];
    int L[NODE], R[NODE], U[NODE], D[NODE];
    void init(int n) {
        this->n = n;
        for (int i = 0; i <= n; ++ i) {
            U[i] = D[i] = i;
            L[i] = i-1; R[i] = i+1;
            S[i] = 0;
        }
        R[n] = 0; L[0] = n;
        sz = n+1;
    }
    void addRow(int r, const vector<int> &columns) {
        int first = sz;
        for (int i = 0; i < columns.size(); ++ i) {
            int c = columns[i];
            L[sz] = sz-1; R[sz] = sz+1;
            D[sz] = c; U[sz] = U[c];
            D[U[c]] = sz; U[c] = sz;
            row[sz] = r; col[sz] = c;
            S[c] ++; sz ++;
        }
        R[sz-1] = first; L[first] = sz-1;
    }
#define FOR(i,A,s) for(int i=A[s];i!=s;i=A[i])
    void remove(int c) {
        L[R[c]] = L[c]; R[L[c]] = R[c];
        FOR(i,D,c)
            FOR(j,R,i) { U[D[j]] = U[j]; D[U[j]] = D[j]; -- S[col[j]]; }
    }
    void restore(int c) {
        FOR(i,U,c)
            FOR(j,L,i) { ++S[col[j]]; U[D[j]]=j; D[U[j]]=j; }
        L[R[c]] = c; R[L[c]] = c;
    }
    bool dfs(int d) {
        if (R[0] == 0) {
            ansd = d;
            return 1;
        }
        int c = R[0];
        FOR(i,R,0) if(S[i]<S[c]) c=i;
        remove(c);
    }
};
```

```

    FOR(i,D,c) {
        ans[d] = row[i];
        FOR(j,R,i) remove(col[j]);
        if(dfs(d+1)) return 1;
        FOR(j,L,i) restore(col[j]);
    }
    restore(c);
    return 0;
}
bool solve(vector<int>&v) {
    v.clear();
    if (!dfs(0)) return 0;
    for (int i = 0; i < ansd; ++ i) v.push_back(ans[i]);
    return 1;
}
} dlx;
char data[18][18];
bool input() {
    for (int i = 0; i < 16; ++ i) {
        if (scanf("%s",data[i]) == EOF) return 0;
    }
    return 1;
}
enum { SLOT=0, ROW, COL, BLOK };
int encode(int i, int j, int k) {
    return i*256 + j*16 + k + 1;
}
int block(int i, int j) {
    return 4*(i/4) + (j/4);
}
void decode(int x, int &a, int &b, int &c) {
    x --;
    c = x % 16; x /= 16;
    b = x % 16; x /= 16;
    a = x;
}
vector<int>columns;
void solve() {
    dlx.init(16*16*4);
    for (int i = 0; i < 16; ++ i) {
        for (int j = 0; j < 16; ++ j) {
            for (int k = 0; k < 16; ++ k) {
                if (data[i][j] == '-' || data[i][j] == k+'A') {
                    columns.clear();
                    columns.push_back(encode(SLOT, i, j));
                    columns.push_back(encode(ROW, i, k));
                    columns.push_back(encode(COL, j, k));
                    columns.push_back(encode(BLOK, block(i,j), k));
                    dlx.addRow(encode(i,j,k), columns);
                }
            }
        }
    }
    columns.clear();
    dlx.solve(columns);
    for (int i = 0; i < columns.size(); ++ i) {
        int r, c, v;
        decode(columns[i], r, c, v);
        data[r][c] = char('A' + v);
    }
}

```

```

    }
    for (int i = 0; i < 16; ++ i) {
        printf("%s\n", data[i]);
    }
}

int main() {
    int kcase = 0;
    while (input()) {
        if (kcase) puts("");
        kcase ++;
        solve();
    }
}

```

5.2 Matrix Fast Power

```

struct Matrix {
    int n, a[N][N];
    Matrix operator * (const Matrix &b) const {
        Matrix ret; ret.clear();
        ret.n = n;
        for (int i = 0; i < n; ++ i) {
            for (int k = 0; k < n; ++ k) if (a[i][k]) {
                for (int j = 0; j < n; ++ j) {
                    ret.a[i][j] += a[i][k] * b.a[k][j];
                    ret.a[i][j] %= mod;
                }
            }
        }
        return ret;
    }
    void clear() {
        memset(a, 0, sizeof(a));
    }
};

Matrix matrix_one(int n) {
    Matrix ret; ret.clear();
    ret.n = n;
    for (int i = 0; i < n; ++ i) {
        ret.a[i][i] = 1;
    }
    return ret;
}

Matrix matrix_pow(Matrix x, int n) {
    Matrix ret = matrix_one(x.n), mul = x;
    while (n) {
        if (n&1) ret = ret * mul;
        mul = mul * mul;
        n >>= 1;
    }
    return ret;
}

```

5.3 Polynomial

```

// eg: UVALive 4305
const int MAXN = 500;
const double EPS = 1e-10;
inline int sgn(const double &a) { return a > EPS ? 1 : (a < -EPS ? -1 : 0); }
struct Polynomial {
    double data[MAXN];
    int n;
    Polynomial() {}
    Polynomial(int _n) : n(_n) {
        memset(data, 0, sizeof(data));
    }
    Polynomial(double *_data, int _n) {
        memset(data, 0, sizeof(data));
        n = _n;
        for (int i = n; i >= 0; i--) data[i] = *_data[i];
    }
    Polynomial operator + (const Polynomial &a) {
        Polynomial c(max(n, a.n));
        for (int i = c.n; i >= 0; i--) c.data[i] = data[i] + a.data[i];
        while (sgn(c.data[c.n]) == 0 && c.n) c.n--;
        return c;
    }
    Polynomial operator - (const Polynomial &a) {
        Polynomial c(max(n, a.n));
        for (int i = c.n; i >= 0; i--) c.data[i] = data[i] - a.data[i];
        while (sgn(c.data[c.n]) == 0 && c.n) c.n--;
        return c;
    }
    Polynomial operator * (const Polynomial &a) {
        Polynomial c(n + a.n);
        for (int i = n; i >= 0; i--) for (int j = a.n; j >= 0; j--) c.data[i + j] += data[i] *
            a.data[j];
        return c;
    }
    Polynomial operator / (const Polynomial &a) {
        if (n < a.n) return *this;
        else {
            Polynomial c(n - a.n);
            for (int i = c.n; i >= 0; i--) c.data[i] = data[i + a.n];
            for (int i = c.n; i >= 0; i--) {
                c.data[i] /= a.data[a.n];
                for (int j = i - 1; a.n - i + j >= 0 && j >= 0; j--) c.data[j] -= c.data[i] * a.data[a.n -
                    i + j];
            }
            return c;
        }
    }
    Polynomial operator % (const Polynomial &a) {
        Polynomial c = *this - *this / a * a;
        while (sgn(c.data[c.n]) == 0 && c.n) c.n--;
        return c;
    }
    bool iszero() {
        return n == 0 && sgn(data[0]) == 0;
    }
    bool isconst() {
        return n > 0;
    }
}

```

```

Polynomial derivative() {
    Polynomial a(n - 1);
    for (int i = n - 1; i >= 0; i--) a.data[i] = data[i + 1] * (double)(i + 1);
    return a;
}

Polynomial integral() {
    Polynomial a(n + 1);
    for (int i = n + 1; i >= 1; i--) a.data[i] = data[i - 1] / (double)i;
    return a;
}

void show() {
    for (int i = n; i >= 0; i--) {
        printf("%.6f", data[i], i);
        if (i != 0) printf(" x");
        if (i != 1 && i != 0) printf(" ^ %d", i);
        if (i != 0) printf(" + ");
        else printf("\n");
    }
}

};

Polynomial gcd(Polynomial a, Polynomial b) {
    if (b.iszero()) return a;
    else return gcd(b, a % b);
}

```
