South China University of Technology

SCUT_gugugu

TEMPLATE

Last build at April 19, 2019

Contents

1 Graph Theory

1.1 Shortest Path

1.1.1 Dijkstra

```
typedef pair<int, int> P;
struct Edge {
    int to, nxt;
    LL w;
}e[MAXM];
int head[MAXN], ecnt;
LL d[MAXN];
priority_queue<P, vector<P>, greater<P> > q;
inline void addEdge(int x, int y, LL w) {
    e[++ecnt] = (Edge) {y, head[x], w}; head[x] = ecnt;
}
void dijkstra(int st) {
    memset(d, 0x3f, sizeof(d));
    d[st] = 0;
    q.push(make_pair(0, st));
    while(!q.empty()) {
        P x = q.top(); q.pop();
        int u = x.second;
        for(int i = head[u], v; i; i = e[i].nxt) {
            v = e[i].to;
            if(d[v] > d[u] + e[i].w) {
                d[v] = d[u] + e[i].w;
                q.push(make_pair(d[v], v));
            }
        }
    }
}
```

1.1.2 SPFA

```
struct Edge {
    int to, nxt;
    LL w;
}e[MAXE];
int head[MAXN], ecnt;
LL d[MAXN];
bool exist[MAXN];
queue<int> q;
inline void addEdge(int x, int y, LL w) {
   e[++ecnt] = (Edge) {y, head[x], w}; head[x] = ecnt;
}
void SPFA(int st) {
    memset(d,0x3f,sizeof(d));
    d[st] = 0;
    q.push(st);
    exist[st] = 1;
    while(!q.empty()) {
        int u = q.front(); q.pop();
        exist[u] = 0;
        for(int i = head[u], v; i; i = e[i].nxt) {
            v = e[i].to;
            if(d[v] > d[u] + e[i].w) {
                d[v] = d[u] + e[i].w;
                //pre[v] = u;
                if(!exist[v]) {
                    q.push(v);
                    exist[v] = 1;
                }
            }
        }
    }
}
```

1.2 Network Flow

1.2.1 ISAP

```
namespace NWF {
    struct Edge{
        int to, nxt;LL f;
    }e[MAXM << 1];
    int S, T, tot;
    int ecnt, head[MAXN], cur[MAXN], pre[MAXN], num[MAXN], dis[MAXN];
    queue<int> q;
    void init(int _S, int _T, int _tot){
        ecnt = 1; S = _S; T = _T; tot = _tot;
        memset(num,  0, (tot + 1) * sizeof(int));
        memset(head, 0, (tot + 1) * sizeof(int));
    }
    inline void addEdge(int u, int v, LL f) {
        e[++ecnt] = (Edge) {v, head[u], f}; head[u] = ecnt;
        e[++ecnt] = (Edge) {u, head[v], 0}; head[v] = ecnt;
    }
    void bfs() {
        memset(dis, 0, (tot + 1) * sizeof(int));
        q.push(T);
        dis[T] = 1;
        while(!q.empty()) {
            int u = q.front(), v; q.pop();
            num[dis[u]]++;
            for(int i = cur[u] = head[u]; i; i = e[i].nxt) {
                if(!dis[v = e[i].to]) {
                    dis[v] = dis[u] + 1;
                    q.push(v);
                }
            }
        }
    }
    LL augment() {
        LL flow = INF;
        for(int i = S; i != T; i = e[cur[i]].to)
            flow = min(flow, e[cur[i]].f);
        for(int i = S; i != T; i = e[cur[i]].to) {
            e[cur[i]].f -= flow;
            e[cur[i] ^ 1].f += flow;
        }
        return flow;
    }
    LL isap() {
        bfs();
        int u = S, v;
        LL flow = 0;
        while(dis[S] <= tot) {
            if(u == T) {
                flow += augment();
                u = S;
            }
            bool fg = 0;
            for(int i = cur[u]; i; i = e[i].nxt) {
                if(e[i].f && dis[u] > dis[v = e[i].to]) {
                    pre[v] = u;
                    cur[u] = i;
                    u = v;
                    fg = 1;
                    break;
                }
            }
            if(fg) continue;
            if(!--num[dis[u]]) break;
            int maxDis = tot;
            for(int i = head[u]; i; i = e[i].nxt) {
                if(e[i].f && maxDis > dis[v = e[i].to]) {
                    maxDis = dis[v];
                    cur[u] = i;
                }
            }
            num[dis[u] = maxDis + 1]++;
            if(u != S) u = pre[u];
        }
        return flow;
    }
}
```

1.2.2 HLPP

```
namespace NWF{
    struct Edge{
        int to,nxt;LL f;
    }e[MAXM << 1];
    int S, T, tot;
    int ecnt, head[MAXN], dis[MAXN], num[MAXN];
    LL sumf[MAXN];
    queue<int> q;
    list<int> dep[MAXN];
    void init(int _S,int _T,int _tot){
        ecnt = 1;S = _S;T = _T;tot = _tot;
        memset(num,  0, (tot + 1) * sizeof(int));
        memset(head, 0, (tot + 1) * sizeof(int));
        memset(sumf, 0, (tot + 1) * sizeof(LL));
    }
    void addEdge(int u,int v,LL f){
        e[++ecnt] = (Edge) {v, head[u], f};head[u] = ecnt;
        e[++ecnt] = (Edge) {u, head[v], 0};head[v] = ecnt;
    }
    void bfs(){
        memset(dis, 0, (tot + 1) * sizeof(int));
        q.push(T); dis[T] = 1;
        while(!q.empty()){
            int u=q.front(), v; q.pop();
            for(int i = head[u]; i; i = e[i].nxt)
            if(!dis[v = e[i].to]){
                dis[v] = dis[u] + 1;
                q.push(v);
            }
        }
    }
    LL hlpp(){
        bfs();
        dis[S] = tot + 1;
        for(int i = 1;i <= tot; ++i)num[dis[i]]++;
        for(int i = tot + 1; ~i; --i)dep[i].clear();
        int maxd = dis[S];LL f;
        dep[maxd].push_back(S);sumf[S] = INF;
        for(;;){
            while(maxd && dep[maxd].empty())maxd--;
            if(!maxd)break;
            int u = dep[maxd].back(), v;dep[maxd].pop_back();
            int minDis = tot + 1;
            for(int i = head[u]; i;i = e[i].nxt)
            if(e[i].f){
                if(dis[u] > dis[v = e[i].to]){
                    f = min(sumf[u], e[i].f);
                    e[i].f -= f;e[i^1].f += f;
                    if(sumf[u] != INF) sumf[u] -= f;
                    if(sumf[v] != INF) sumf[v] += f;
                    if(v!=S && v!=T && sumf[v] == f){
                        maxd = max(maxd, dis[v]);
                        dep[dis[v]].push_back(v);
                    }
                    if(!sumf[u])break;
                }else minDis=min(minDis, dis[v] + 1);
            }
            if(sumf[u]){
                if(!--num[dis[u]]){
                    for(int i = dis[u];i <= maxd;++i){
                        while(!dep[i].empty()){
                            --num[i];
                            dis[dep[i].back()] = tot + 1;
                            dep[i].pop_back();
                        }
                    }
                    maxd = dis[u] - 1;dis[u] = tot + 1;
                }else{
                    dis[u] = minDis;
                    if(minDis > tot)continue;
                    num[minDis]++;
                    maxd = max(maxd, minDis);
                    dep[minDis].push_back(u);
                }
            }
        }
        return sumf[T];
    }
}
```

1.2.3 Dinic

```
namespace NWF {
    struct Edge {
        int to, nxt;LL f;
    } e[MAXM << 1];
    int S, T, tot;
    int ecnt, head[MAXN], cur[MAXN], dis[MAXN];
    queue<int> q;
    void init(int _S, int _T, int _tot){
        ecnt = 1; S = _S; T = _T; tot = _tot;
        memset(head, 0, (tot + 1) * sizeof(int));
    }
    void addEdge(int u, int v, LL f) {
        e[++ecnt] = (Edge) {v, head[u], f}; head[u] = ecnt;
        e[++ecnt] = (Edge) {u, head[v], 0}; head[v] = ecnt;
    }
    bool bfs() {
        memset(dis, 0, (tot + 1) * sizeof(int));
        q.push(S); dis[S] = 1;
        while (!q.empty()) {
            int u = q.front(), v; q.pop();
            for (int i = cur[u] = head[u]; i ; i = e[i].nxt) {
                if (e[i].f && !dis[v = e[i].to]) {
                    q.push(v);
                    dis[v] = dis[u] + 1;
                }
            }
        }
        return dis[T];
    }
    LL dfs(int u, LL maxf) {
        if (u == T) return maxf;
        LL sumf = maxf;
        for (int &i = cur[u]; i; i = e[i].nxt) {
            if (e[i].f && dis[e[i].to] > dis[u]) {
                LL tmpf = dfs(e[i].to, min(sumf, e[i].f));
                e[i].f -= tmpf; e[i ^ 1].f += tmpf;
                sumf -= tmpf;
                if (!sumf) return maxf;
            }
        }
        return maxf - sumf;
    }
    LL dinic() {
        LL ret = 0;
        while (bfs()) ret += dfs(S, INF);
        return ret;
    }
}
```

1.2.4 MCMF

```
namespace NWF{
    struct Edge {
        int to, nxt;LL f, c;
    } e[MAXM << 1];
    int S, T, tot;
    int ecnt, head[MAXN], cur[MAXN];LL dis[MAXN];
    bool exist[MAXN];
    queue<int> q;
    void init(int _S, int _T, int _tot){
        ecnt = 1; S = _S; T = _T; tot = _tot;
        memset(head, 0, (tot + 1) * sizeof(int));
    }
    void addEdge(int u, int v, LL f, LL c) {
        e[++ecnt] = (Edge) {v, head[u], f, c}; head[u] = ecnt;
        e[++ecnt] = (Edge) {u, head[v], 0,-c}; head[v] = ecnt;
    }
    bool spfa() {
        for(int i = 0;i <= tot; ++i){
            dis[i] = INF;exist[i] = cur[i] = 0;
        }
        q.push(S);dis[S] = 0;exist[S] = 1;
        while(!q.empty()) {
            int u = q.front(), v; q.pop();exist[u] = 0;
            for(int i = head[u]; i; i = e[i].nxt) {
                if(e[i].f && dis[v = e[i].to] > dis[u] + e[i].c) {
                    dis[v] = dis[u] + e[i].c;
                    cur[v] = i;
                    if(!exist[v]) {
                        q.push(v);
                        exist[v] = 1;
                    }
                }
            }
        }
        return dis[T] != INF;
    }
    LL mcmf() {
        LL cost = 0;
        while(spfa()) {
            LL flow = INF;
            for(int i = T; i != S; i = e[cur[i] ^ 1].to)
                flow = min(flow, e[cur[i]].f);
            for(int i = T; i != S; i = e[cur[i] ^ 1].to) {
                e[cur[i]].f -= flow;
                e[cur[i] ^ 1].f += flow;
            }
            cost += flow * dis[T];
        }
        return cost;
    }
}
```

1.3 Tree Related

1.3.1 Kruskal

```
namespace MST{
    struct Edge{
        int u,v; LL w;
        bool operator < (const Edge& x) const { return w < x.w; }
    }e[MAXM];
    int ecnt, fa[MAXN];
    void addEdge(int u, int v, LL w) {
        e[++ecnt] = (Edge){v, u, w}; headp[u] = ecnt;
    }
    int Find(int x) { return x == fa[x] ? x : fa[x] = Find(fa[x]); }
    LL kruskal(int n) {
        sort(e + 1, e + ecnt + 1);
        for(int i = 1; i <= n; i++) fa[i] = i;
        LL sum = 0;
        for (int i = 1; i <= ecnt; i++){
            int fu = Find(e[i].u), fv = Find(e[i].v);
            if(fu != fv){
                fa[fu] = fv;
                sum += e[i].w;
            }
        }
        return sum;
    }
}
```

1.3.2 Prim

```
namespace MST {
    struct Edge{
        int to,nxt; LL w;
    }e[MAXM];
    int ecnt, head[MAXN], vis[MAXN]; // pre[MAXN];
    LL dis[MAXN];
    void addEdge(int u, int v, LL w){
        e[++ecnt] = (Edge){v, head[u], w}; head[u] = ecnt;
        e[++ecnt] = (Edge){u, head[v], w}; head[v] = ecnt;
    }
    LL Prim(int n){
        for (int i = 1; i <= n; i++){
            //pre[i] = 0;
            vis[i] = 0;
            dis[i] = INF;
        }
        vis[1] = 1;
        LL sum = 0;
        for (int i = head[1]; i; i = e[i].nxt)
            dis[e[i].to] = min(dis[e[i].to],e[i].w);
        for (int j = 1; j < n; j++){
            int u; LL minDis = INF;
            for (int i = 1; i <= n; ++i)
                if (!vis[i] && dis[i] < minDis){
                    minDis = dis[i];
                    u = i;
                }
            if (minDis == INF) return -1;
            vis[u] = 1;
            sum += minDis;
            for (int i = head[u], v; i; i = e[i].nxt)
            if (!vis[v = e[i].to] && e[i].w < dis[v]){
                //pre[u] = v;
                dis[v] = e[i].w;
            }
        }
        return sum;
    }
}
```

1.3.3 Tree Divide and Conquer

```
struct Edge {
    int to, nxt, w;
}e[MAXM];
int head[MAXN], ecnt;
int sz[MAXN];
int d[MAXN], t[5], ans;
bool vis[MAXN];
inline void add_edge(int u, int v, int w) {
    e[++ecnt] = (Edge) {v, head[u], w}; head[u] = ecnt;
    e[++ecnt] = (Edge) {u, head[v], w}; head[v] = ecnt;
}
int getsz(int x, int fa) {
    sz[x] = 1;
    for(int i = head[x]; i; i = e[i].nxt) {
        int y = e[i].to;
        if(vis[y] || y == fa) continue;
        sz[x] += getsz(y, x);
    }
    return sz[x];
}
int getrt(int x) {
    int tot = getsz(x, 0) >> 1;
    while(1) {
        int u = -1;
        for(int i = head[x]; i; i = e[i].nxt) {
            int y = e[i].to;
            if(vis[y] || sz[y] > sz[x]) continue;
            if(u == -1 || sz[y] > sz[u]) u = y;
        }
        if(~u && sz[u] > tot) x = u;
        else break;
    }
    return x;
}
void getdep(int x, int fa) {
    t[d[x]]++;
    for(int i = head[x]; i; i = e[i].nxt) {
        int y = e[i].to;
        if(vis[y] || y == fa) continue;
        d[y] = (d[x] + e[i].w) % 3;
        getdep(y, x);
    }
}
int cal(int x, int v) {
    t[0] = t[1] = t[2] = 0;
    d[x] = v % 3;
    getdep(x, 0);
    return t[0] * t[0] + t[1] * t[2] * 2;
}
void solve(int x) {
    vis[x] = 1;
    ans += cal(x, 0);
    for(int i = head[x]; i; i = e[i].nxt) {
        int y = e[i].to;
        if(vis[y]) continue;
        ans -= cal(y, e[i].w);
        solve(getrt(y));
    }
}
int main() {
    solve(getrt(1));
}
```

1.4 LCA

1.4.1 Tree Decomposition LCA

```
int sz[MAXN], dep[MAXN], top[MAXN], fa[MAXN], son[MAXN], num[MAXN], totw;
struct Edge {
    int to, nxt;
}e[MAXN << 1];
int head[MAXN], ecnt;
inline void add_edge(int x, int y) {
    e[++ecnt] = (Edge) {y, head[x]}; head[x] = ecnt;
}
void dfs1(int x) {
    sz[x] = 1; son[x] = 0;
    for(int i = head[x]; i; i = e[i].nxt) {
        int v = e[i].to;
        if(v == fa[x]) continue;
        fa[v] = x;
        dep[v] = dep[x] + 1;
        dfs1(v);
        sz[x] += sz[v];
        if(sz[v] > sz[son[x]]) son[x] = v;
    }
}
void dfs2(int x) {
    B[num[x]] = A[x];
    if(son[x]) {
        top[son[x]] = top[x];
        num[son[x]] = ++totw;
        dfs2(son[x]);
    }
    for(int i = head[x]; i; i = e[i].nxt) {
        int v = e[i].to;
        if(v == fa[x] || v == son[x]) continue;
        top[v] = v;
        num[v] = ++totw;
        dfs2(v);
    }
}
int lca(int u, int v) {
    if(u == v) return u;
    while(top[u] != top[v]) {
        if(dep[top[u]] > dep[top[v]]) swap(u, v);
        v = fa[top[v]];
    }
    if(dep[u] > dep[v]) swap(u, v);
    return u;
}
inline void init() {
    memset(head, 0, sizeof(head)); ecnt = 0;
    fa[1] = 0; dep[1] = 1; top[1] = 1; num[1] = 1; totw = 1;
}
inline void pre() {
    dfs1(1); dfs2(1);
}
```

1.4.2 Tarjan LCA

```
vector< pair<int,int> > G[MAXN],ask[MAXN];
```

```
int fa[MAXN], ans[MAXN], vis[MAXN] ,dis[MAXN];
int Find(int x){
    return x == fa[x] ? x : fa[x] = Find(fa[x]);
}
void init(int n){
    memset(ans, 0,sizeof ans);
    memset(vis, 0,sizeof vis);
    for(int i = 0; i <= n; i++){
        G[i].clear();
        ask[i].clear();
    }
}
void LCA(int u){
    int v;
    fa[u] = u;
    vis[u] = true;
    for(auto it : ask[u])
        if(vis[v = it.first])
            ans[it.second] = dis[u] + dis[v] - 2 * dis[Find(it.first)];
    for(auto it : G[u])
    if(!vis[v = it.first]){
        dis[v] = dis[u] + it.second;
        LCA(v);
        fa[v] = u;
    }
}
```

1.5 Tarjan

1.5.1 SCC

```
namespace SCC{
    vector<int> G[MAXN];
    int dfs_clock, scc_cn, dfn[MAXN], low[MAXN], sccno[MAXN];
    stack<int> S;
    void addEdge(int u, int v) {
        G[u].push_back(v);
    }
    void tarjan(int u) {
        dfn[u] = low[u] = ++dfs_clock;
        S.push(u);
        for(auto v : G[u]) {
            if(!dfn[v]) {
                tarjan(v);
                low[u] = min(low[u], low[v]);
            }else if(!sccno[v]) {
                low[u] = min(low[u], dfn[v]);
            }
        }
        if(dfn[u] == low[u]) {
            scc_cnt++;
            for(;;) {
                int v = S.top(); S.pop();
                sccno[v] = scc_cnt;
                if(v == u) break;
            }
        }
    }
    void findSCC(int n) {
        for(int i = 1; i <= n; i++)
            if(!dfn[i]) tarjan(i);
    }
    void init(int n){
        dfs_clock = scc_cnt = 0;
        for(int i = 0;i <= n;++i){
            dfn[i] = low[i] = sccno[i] = 0;
            G[i].clear();
        }
    }
}
```

1.5.2 BCC

```
namespace BCC{
    struct Edge {
        int to, nxt;
    }e[MAXM << 1];
    int ecnt, head[MAXN];
    int dfs_clock, dfn[MAXN], low[MAXN];

    int is_vertex[MAXN], vbcc_cnt, vbccno[MAXN];
    vector<int> vbcc[MAXN];
    stack<int> vS;

    int ebcc_cnt, ebccno[MAXN];
    stack<int> eS;

    inline void addEdge(int u, int v) {
        e[++ecnt] = (Edge) {v, head[u]}; head[u] = ecnt;
        e[++ecnt] = (Edge) {u, head[v]}; head[v] = ecnt;
    }
    inline void init(int n) {
        ecnt = 1;
        dfs_clock = 0;
        vbcc_cnt = 0;
        ebcc_cnt = 0;
        for(int i = 1; i <= n; ++i){
            head[i] = dfn[i] = low[i] = 0;
            is_vertex[i] = 0;
            vbccno[i] = 0;
            ebccno[i] = 0;
        }
        while(!vS.empty()) vS.pop();
    }
    //root's edge = -1;
    void tarjan(int u, int edge) {
        dfn[u] = low[u] = ++dfs_clock;
        int ch = 0;
        vS.push(u);
        eS.push(u);
        for(int i = head[u], v; i; i = e[i].nxt) {
            if(!dfn[v = e[i].to]) {
                tarjan(v, i ^ 1);
                low[u] = min(low[u], low[v]);
                if(low[v] >= dfn[u]) {
                    ++ch;
                    if(edge > 0 || ch > 1) is_vertex[u] = 1;
                    vbcc[++vbcc_cnt].clear();
                    vbcc[vbcc_cnt].push_back(u);
                    for(int x;;){
                        x = vS.top();vS.pop();
                        vbcc[vbcc_cnt].push_back(x);
                        vbccno[x] = vbcc_cnt;
                        if(x == v)break;
                    }
                }
                if(low[v] > dfn[u]) {
                // i && i ^ 1 is bridge
                }
            }
            else if(dfn[v] < dfn[u] && i != edge)
                low[u] = min(low[u], dfn[v]);
        }
        if(dfn[u] == low[u]) {
            ebcc_cnt++;
            for(int v;;) {
                v = eS.top(); eS.pop();
                ebccno[v] = ebcc_cnt;
                if(v == u) break;
            }
        }
    }
    void findBCC(int n){
        for(int i = 1; i <= n; i++)
            if(!dfn[i]) tarjan(i, -1);

        //findBridge
        for(int u = 1; u <= n; u++) {
            for(int i = head[u], v; i; i = e[i].nxt)
            if(ebccno[u] != ebccno[v = e[i].to]) {
                //is bridge
            }
        }
    }
}
```

1.6 Cactus

1.6.1 Circle-Square Tree

```
#include <bits/stdc++.h>
using namespace std;
typedef pair<int, int> P;
const int MAXN = 2e4 + 5;
const int S = 15;
namespace Tree {
    struct Edge {
        int to, nxt, w;
    }e[MAXN << 1];
    int ecnt, head[MAXN];
    int rt, isrt[MAXN], fa[MAXN][S + 3];
    int sz[MAXN];
    inline void addEdge(int u, int v, int w) {
        e[++ecnt] = (Edge) {v, head[u], w}; head[u] = ecnt;
        fa[v][0] = u;
    }
}
int n, m, Q;
```

```
namespace BCC {
    struct Edge {
        int to, nxt, w;
    }e[MAXN << 1];
    int ecnt, head[MAXN];
    int dfs_clock, dfn[MAXN], low[MAXN];
    int is_vertex[MAXN], vbcc_cnt, vbccno[MAXN];
    vector<P> vbcc[MAXN];
    stack<P> vs;
    int tag[MAXN];
    inline void addEdge(int u, int v, int w) {
        e[++ecnt] = (Edge) {v, head[u], w}; head[u] = ecnt;
        e[++ecnt] = (Edge) {u, head[v], w}; head[v] = ecnt;
    }
    inline void init(int n) {
        ecnt = 1;
        dfs_clock = 0;
        vbcc_cnt = 0;
        for(int i = 0; i <= 2 * n; i++){
            head[i] = dfn[i] = low[i] = 0;
            vbccno[i] = 0;
            tag[i] = 0;
        }
        while(!vs.empty()) vs.pop();
    }
    //root's edge = -1;
    void tarjan(int u, int edge) {
        dfn[u] = low[u] = ++dfs_clock;
        vs.push(P(u, e[edge ^ 1].w));
        for(int i = head[u], v; i; i = e[i].nxt) {
            if(!dfn[v = e[i].to]) {
                tarjan(v, i ^ 1);
                low[u] = min(low[u], low[v]);
                if(low[v] >= dfn[u]) {
                    if(vs.top().first == v) {
                        Tree::addEdge(u, v, vs.top().second);
                        vs.pop();
                        continue;
                    }
                    vbcc[++vbcc_cnt].clear();
                    vbcc[vbcc_cnt].push_back(P(u, 0));
                    Tree::isrt[u] = 1;
                    int &sz = Tree::sz[n + vbcc_cnt];
                    tag[vs.top().first] = n + vbcc_cnt;
                    //Tree::addEdge(u, rt, 0);
                    for(P x;;) {
                        x = vs.top(); vs.pop();
                        sz += x.second;
                        //Tree::addEdge(rt, x.first, sz);
                        vbcc[vbcc_cnt].push_back(x);
                        vbccno[x.first] = vbcc_cnt;
                        if(x.first == v) break;
                    }
                }
            }
            else if(dfn[v] < dfn[u] && i != edge)
                low[u] = min(low[u], dfn[v]);
        }
        for(int i = head[u], v; i; i = e[i].nxt) {
            if(tag[v = e[i].to]) {
                int r = tag[v]; Tree::sz[r] += e[i].w;
                tag[v] = 0;
            }
        }
    }
    void findBCC(int n) {
        for(int i = 1; i <= n; i++)
            if(!dfn[i]) tarjan(i, -1);
    }
}
namespace Tree {
    int dis[MAXN], dep[MAXN], len[MAXN];
    inline void init(int n) {
        BCC::init(n);
        rt = n;
        ecnt = 1;
        for(int i = 0; i <= 2 * n; i++) {
            head[i] = 0;
            fa[i][0] = isrt[i] = dis[i] = dep[i]  = len[i] = 0;
        }
    }
    void dfs(int x) {
        for(int i = head[x], y; i; i = e[i].nxt) {
            if(!dep[y = e[i].to]) {
                dep[y] = dep[x] + 1;
                dis[y] = dis[x] + e[i].w;
                dfs(y);
            }
        }
    }
    void pre() {
        for(int k = 1; k <= BCC::vbcc_cnt; k++) {
            rt++;
            vector<P> &E = BCC::vbcc[k];
            addEdge(E[0].first, rt, 0);
            int cnt = 0;
            for(int i = E.size() - 1; i >= 1; i--) {
                cnt += E[i].second;
                len[E[i].first] = cnt;
                addEdge(rt, E[i].first, min(cnt, sz[rt] - cnt));
            }
        }
        for(int k = 1; k <= S; k++) {
            for(int i = 1; i <= rt; i++) {
                fa[i][k] = fa[fa[i][k - 1]][k - 1];
            }
        }
        dep[1] = 1;
        dfs(1);
    }
    int up(int x, int d) {
        for(int i = S; i >= 0; i--) {
            if(dep[fa[x][i]] >= d) x = fa[x][i];
        }
        return x;
    }
    int lca(int u, int v) {
        if(dep[u] > dep[v]) swap(u, v);
        v = up(v, dep[u]);
        if(u == v) return u;
        for(int i = S; i >= 0; i--) {
            if(fa[u][i] != fa[v][i]) {
                u = fa[u][i], v = fa[v][i];
            }
        }
        return fa[u][0];
    }
    int query(int u, int v) {
        int l = lca(u, v);
        if(l <= n) return dis[u] + dis[v] - 2 * dis[l];
        int x = up(u, dep[l] + 1), y = up(v, dep[l] + 1);
        int res = dis[u] - dis[x] + dis[v] - dis[y];
        int tmp = abs(len[x] - len[y]);
        return res + min(tmp, sz[l] - tmp);
    }
}

int main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.precision(6); cout << fixed;
    using namespace Tree;
    cin >> n >> m >> Q;
    init(n);
    for(int i = 1, u, v, w; i <= m; i++) {
        cin >> u >> v >> w;
        BCC::addEdge(u, v, w);
    }
    BCC::findBCC(n);
    pre();
    int u, v;
    while(Q--) {
        cin >> u >> v;
        cout << query(u, v) << endl;
    }
    return 0;
}
```

2 Data Structures

2.1 Basic Structures

2.1.1 RMQ

```
struct RMQ {
    int d[MAXN][S + 3];
    inline void init(int *a, int n) {
        for(int i = 0; i < n; i++) d[i][0] = a[i];
        for(int k = 1; (1 << k) < n; k++)
            for(int i = 0; i + (1 << k) - 1 < n; i++)
                d[i][k] = min(d[i][k - 1], d[i + (1 << (k - 1))][k - 1]);
    }
    inline int query(int l, int r) {
        if(l > r) swap(l, r);
        int k = 0;
        while((1 << (k + 1)) <= r - l + 1) k++;
        return min(d[l][k], d[r - (1 << k) + 1][k]);
    }
}rmq;
```

2.1.2 Divide Blocks

```
int belong[MAXN], l[MAXN], r[MAXN];
int sz, num;
void build(int n) {
    sz = sqrt(n);
    num = n / sz; if(n % sz) num++;
    for(int i = 1; i <= num; i++) {
        l[i] = (i - 1) * sz + 1;
        r[i] = i * sz;
    }
    r[num] = n;
    for(int i = 1; i <= n; i++) {
        belong[i] = (i - 1) / sz + 1;
    }
}
```

2.2 Tree Structures

2.2.1 Tree Decomposition

```
int sz[MAXN], dep[MAXN], top[MAXN], fa[MAXN], son[MAXN], num[MAXN], totw;
struct Edge {
    int to, nxt;
}e[MAXN << 1];
int head[MAXN], ecnt;
int n, m, Q;
#define Ls(x) (x << 1)
#define Rs(x) (x << 1 | 1)
struct Tree {
    int l, r, lazy;
    LL sum, mx;
}tree[MAXN << 2];
int A[MAXN], B[MAXN];
```

```
void push_up(int x) {
    tree[x].sum = tree[Ls(x)].sum + tree[Rs(x)].sum;
    tree[x].mx = max(tree[Ls(x)].mx, tree[Rs(x)].mx);
}
void push_down(int x) {
    if(tree[x].lazy) {
        tree[Ls(x)].sum += tree[x].lazy * (tree[Ls(x)].r - tree[Ls(x)].l + 1);
        tree[Rs(x)].sum += tree[x].lazy * (tree[Rs(x)].r - tree[Rs(x)].l + 1);
        tree[Ls(x)].mx += tree[x].lazy;
        tree[Rs(x)].mx += tree[x].lazy;
        tree[Ls(x)].lazy += tree[x].lazy;
        tree[Rs(x)].lazy += tree[x].lazy;
        tree[x].lazy = 0;
    }
}
void build(int x, int L, int R) {
    tree[x].lazy = 0;
    tree[x].l = L; tree[x].r = R;
    if(L == R) {
        tree[x].sum = B[L];
        tree[x].mx = B[L];
        return;
    }
    int mid = (L + R) >> 1;
    build(Ls(x), L, mid);
    build(Rs(x), mid + 1, R);
    push_up(x);
}
void update(int x, int L, int R, LL val) {
    if(tree[x].l >= L && tree[x].r <= R) {
        tree[x].lazy += val;
        tree[x].sum += val * (tree[x].r - tree[x].l + 1);
        tree[x].mx += val;
        return;
    }
    push_down(x);
    int mid = (tree[x].l + tree[x].r) >> 1;
    if(L <= mid) update(Ls(x), L, R, val);
    if(R > mid) update(Rs(x), L, R, val);
    push_up(x);
}
LL query(int x, int L, int R) {
    if(tree[x].l >= L && tree[x].r <= R)
        return tree[x].sum;
    push_down(x);
    int mid = (tree[x].l + tree[x].r) >> 1;
    LL res = 0;
    if(L <= mid) res += query(Ls(x), L, R);
    if(R > mid) res += query(Rs(x), L, R);
    return res;
}
LL query2(int x, int L, int R) {
    if(tree[x].l >= L && tree[x].r <= R)
        return tree[x].mx;
    push_down(x);
    int mid = (tree[x].l + tree[x].r) >> 1;
    LL res = -INF;
    if(L <= mid) res = max(res, query2(Ls(x), L, R));
    if(R > mid) res = max(res, query2(Rs(x), L, R));
    return res;
}
```

```
inline void add_edge(int x, int y) {
    e[++ecnt] = (Edge) {y, head[x]}; head[x] = ecnt;
}
void dfs1(int x) {
    sz[x] = 1; son[x] = 0;
    for(int i = head[x]; i; i = e[i].nxt) {
        int v = e[i].to;
        if(v == fa[x]) continue;
        fa[v] = x;
        dep[v] = dep[x] + 1;
        dfs1(v);
        sz[x] += sz[v];
        if(sz[v] > sz[son[x]]) son[x] = v;
    }
}
void dfs2(int x) {
    B[num[x]] = A[x];
    if(son[x]) {
        top[son[x]] = top[x];
        num[son[x]] = ++totw;
        dfs2(son[x]);
    }
    for(int i = head[x]; i; i = e[i].nxt) {
        int v = e[i].to;
        if(v == fa[x] || v == son[x]) continue;
        top[v] = v;
        num[v] = ++totw;
        dfs2(v);
    }
}
void up(int a, int b, int c) {
    int f1 = top[a], f2 = top[b];
    while(f1 != f2) {
        if(dep[f1] < dep[f2]) { swap(a, b); swap(f1, f2); }
        update(1, num[f1], num[a], c);
        a = fa[f1];
        f1 = top[a];
    }
    if(dep[a] > dep[b]) swap(a, b);
    update(1, num[a], num[b], c);
}
int qsum(int a, int b) {
    if(a == b) return query(1, num[a], num[a]);
    int f1 = top[a], f2 = top[b];
    int res = 0;
    while(f1 != f2) {
        if(dep[f1] < dep[f2]) { swap(a, b); swap(f1, f2); }
        res += query(1, num[f1], num[a]);
        a = fa[f1];
        f1 = top[a];
    }
    if(dep[a] > dep[b]) swap(a, b);
    res += query(1, num[a], num[b]);
    return res;
}
int qmax(int a, int b) {
    if(a == b) return query2(1, num[a], num[a]);
    int f1 = top[a], f2 = top[b];
    int res = -1000000000;
    while(f1 != f2) {
        if(dep[f1] < dep[f2]) { swap(a, b); swap(f1, f2); }
        res = max(res, query2(1, num[f1], num[a]));
        a = fa[f1];
        f1 = top[a];
    }
    if(dep[a] > dep[b]) swap(a, b);
    res = max(res, query2(1, num[a], num[b]));
    return res;
}
inline void init() {
    memset(head, 0, sizeof(head)); ecnt = 0;
    fa[1] = 0; dep[1] = 1; top[1] = 1; num[1] = 1; totw = 1;
}
inline void pre() {
    dfs1(1); dfs2(1); build(1, 1, totw);
}
```

2.2.2 Link-Cut Tree

```
namespace LCT {
    int fa[MAXN], rev[MAXN], tr[MAXN][2];
    int s[MAXN], val[MAXN];
    void push_up(int x) {
        int l = tr[x][0], r = tr[x][1];
        s[x] = s[l] + s[r] + val[x];
    }
    void Rev(int x) {
        rev[x] ^= 1; swap(tr[x][0], tr[x][1]);
    }
    void push_down(int x) {
        if(!rev[x]) return;
        int l = tr[x][0], r = tr[x][1];
        rev[x] = 0;
        if(l) Rev(l); if(r) Rev(r);
    }
    bool isroot(int x) {
        return tr[fa[x]][0] != x && tr[fa[x]][1] != x;
    }
    void pre(int x) {
        if(!isroot(x)) pre(fa[x]);
        push_down(x);
    }
    void rotate(int x) {
        int y = fa[x]; int z = fa[y];
        int l = tr[y][1] == x;
        int r = l ^ 1;
        if(!isroot(y)) tr[z][tr[z][1] == y] = x;
        fa[x] = z; fa[y] = x; fa[tr[x][r]] = y;
        tr[y][l] = tr[x][r]; tr[x][r] = y;
        push_up(y);
    }
    void splay(int x) {
        pre(x);
        int y, z;
        while(!isroot(x)) {
            y = fa[x]; z = fa[y];
            if(!isroot(y)) {
                if((tr[z][0] == y) == (tr[y][0] == x))rotate(y);
                else rotate(x);
            }
            rotate(x);
        }
        push_up(x);
    }
    void access(int x) {
        int y = 0;
        while(x) {
            splay(x); tr[x][1] = y;
            push_up(x);
            y = x; x = fa[x];
        }
    }
    void makeroot(int x) {
        access(x); splay(x); Rev(x);
    }
    void lnk(int x, int y) {
        makeroot(x); fa[x] = y;
    }
    void cut(int x, int y) {
        makeroot(x); access(y); splay(y);
        tr[y][0] = fa[x] = 0; push_up(y);
    }
    void update(int x, int y) {
        makeroot(x); val[x] = y; push_up(x);
    }
    int query(int x, int y) {
        makeroot(x); access(y); splay(y);
        return s[y];
    }
    bool check(int x, int y) {
        int tmp = y;
        makeroot(x); access(y); splay(x);
        while(!isroot(y)) y = fa[y];
        splay(tmp);
        return x == y;
    }
}
```

2.3 Sequence Structures

2.3.1 Segment Tree

```
#define Ls(x) (x << 1)
#define Rs(x) (x << 1 | 1)
struct Tree {
    int l, r, lazy;
    LL sum, mx;
}tree[MAXN << 2];
int A[MAXN];
void push_up(int x) {
    tree[x].sum = tree[Ls(x)].sum + tree[Rs(x)].sum;
    tree[x].mx = max(tree[Ls(x)].mx, tree[Rs(x)].mx);
}
void push_down(int x) {
    if(tree[x].lazy) {
        tree[Ls(x)].sum += tree[x].lazy * (tree[Ls(x)].r - tree[Ls(x)].l + 1);
        tree[Rs(x)].sum += tree[x].lazy * (tree[Rs(x)].r - tree[Rs(x)].l + 1);
        tree[Ls(x)].mx += tree[x].lazy;
        tree[Rs(x)].mx += tree[x].lazy;
        tree[Ls(x)].lazy += tree[x].lazy;
        tree[Rs(x)].lazy += tree[x].lazy;
        tree[x].lazy = 0;
    }
}
void build(int x, int L, int R) {
    tree[x].lazy = 0;
    tree[x].l = L; tree[x].r = R;
    if(L == R) {
        tree[x].sum = A[L];
        tree[x].mx = A[L];
        return;
    }
    int mid = (L + R) >> 1;
    build(Ls(x), L, mid);
    build(Rs(x), mid + 1, R);
    push_up(x);
}
void update(int x, int L, int R, LL val) {
    if(tree[x].l >= L && tree[x].r <= R) {
        tree[x].lazy += val;
        tree[x].sum += val * (tree[x].r - tree[x].l + 1);
        tree[x].mx += val;
        return;
    }
    push_down(x);
    int mid = (tree[x].l + tree[x].r) >> 1;
    if(L <= mid) update(Ls(x), L, R, val);
    if(R > mid) update(Rs(x), L, R, val);
    push_up(x);
}
LL query(int x, int L, int R) {
    if(tree[x].l >= L && tree[x].r <= R)
        return tree[x].sum;
    push_down(x);
    int mid = (tree[x].l + tree[x].r) >> 1;
    LL res = 0;
    if(L <= mid) res += query(Ls(x), L, R);
    if(R > mid) res += query(Rs(x), L, R);
    return res;
}
LL query2(int x, int L, int R) {
    if(tree[x].l >= L && tree[x].r <= R)
        return tree[x].mx;
    push_down(x);
    int mid = (tree[x].l + tree[x].r) >> 1;
    LL res = -INF;
    if(L <= mid) res = max(res, query2(Ls(x), L, R));
    if(R > mid) res = max(res, query2(Rs(x), L, R));
    return res;
}
```

2.3.2 Splay Tree

```
namespace splay{
    int n, m, sz, rt;
    int val[MAXN], id[MAXN];
    int tr[MAXN][2], size[MAXN], fa[MAXN], rev[MAXN], s[MAXN], lazy[MAXN];
    void push_up(int x) {
        int l = tr[x][0], r = tr[x][1];
        s[x] = max(val[x], max(s[l], s[r]));
        size[x] = size[l] + size[r] + 1;
    }
    void push_down(int x) {
        int l = tr[x][0], r = tr[x][1];
        if(lazy[x]) {
            if(l) {
                lazy[l] += lazy[x];
                s[l] += lazy[x];
                val[l] += lazy[x];
            }
            if(r) {
                lazy[r] += lazy[x];
                s[r] += lazy[x];
                val[r] += lazy[x];
            }
            lazy[x] = 0;
        }
        if(rev[x]) {
            rev[x] = 0;
            rev[l] ^= 1; rev[r] ^= 1;
            swap(tr[x][0], tr[x][1]);
        }
    }
    void rotate(int x, int &k) {
        int y = fa[x];
        int z = fa[y];
        int l, r;
        if(tr[y][0] == x) l = 0;
        else l = 1;
        r = l ^ 1;
        if(y == k) k = x;
        else {
            if(tr[z][0] == y) tr[z][0] = x;
            else tr[z][1] = x;
        }
        fa[x] = z; fa[y] = x; fa[tr[x][r]] = y;
        tr[y][l] = tr[x][r]; tr[x][r] = y;
        push_up(y); push_up(x);
    }
    void splay(int x, int &k) {
        int y, z;
        while(x != k) {
            y = fa[x];
            z = fa[y];
            if(y != k) {
                if((tr[y][0] == x) ^ (tr[z][0] == y)) rotate(x, k);
                else rotate(y, k);
            }
            rotate(x, k);
        }
    }
    int find(int x, int rank) {
        push_down(x);
        int l = tr[x][0], r = tr[x][1];
        if(size[l] + 1 == rank) return x;
        else if(size[l] >= rank) return find(l, rank);
        else return find(r, rank - size[l] - 1);
    }
    void update(int l, int r, int v) {
        int x = find(rt, l), y = find(rt, r + 2);
        splay(x, rt); splay(y, tr[x][1]);
        int z = tr[y][0];
        lazy[z] += v;
        val[z] += v;
        s[z] += v;
    }
    void reverse(int l, int r) {
        int x = find(rt, l), y = find(rt, r + 2);
        splay(x, rt); splay(y, tr[x][1]);
        int z = tr[y][0];
        rev[z] ^= 1;
    }
    void query(int l, int r) {
        int x = find(rt, l), y = find(rt, r + 2);
        splay(x, rt); splay(y, tr[x][1]);
        int z = tr[y][0];
        printf("%d\n", s[z]);
    }
    void build(int l, int r, int f) {
        if(l > r) return;
        int now = id[l], last = id[f];
        if(l == r) {
            fa[now] = last; size[now] = 1;
            if(l < f) tr[last][0] = now;
            else tr[last][1] = now;
            return;
        }
        int mid = (l + r) >> 1; now = id[mid];
        build(l, mid - 1, mid); build(mid + 1, r, mid);
        fa[now] = last;
        push_up(now);
        if(mid < f) tr[last][0] = now;
        else tr[last][1] = now;
    }
    void init() {
        s[0] = -INF;
        scanf("%d%d", &n, &m);
        for(int i = 1; i <= n + 2; i++) id[i] = ++sz;
        build(1, n + 2, 0); rt = (n + 3) >> 1;
    }
}
```

2.4 Persistent Data Structures

2.4.1 Chairman Tree

```
struct Node {
    int l, r;
    LL sum;
}t[MAXN * 40];
int cnt, n;
int rt[MAXN];
void update(int pre, int &x, int l, int r, int v) {
    x = ++cnt; t[x] = t[pre]; t[x].sum++;
    if(l == r) return;
    int mid = (l + r) >> 1;
    if(v <= mid) update(t[pre].l, t[x].l, l, mid, v);
    else update(t[pre].r, t[x].r, mid + 1, r, v);
}
int query(int x, int y, int l, int r, int v) {
    if(l == r) return l;
    int mid = (l + r) >> 1;
    int sum = t[t[y].l].sum - t[t[x].l].sum;
    if(sum >= v) return query(t[x].l, t[y].l, l, mid, v);
    else return query(t[x].r, t[y].r, mid + 1, r, v - sum);
}
```

2.4.2 Persistent Trie

```
//区间异或最值查询
const int N=5e4+10;
int t[N];
int ch[N*32][2],val[N*32];
int cnt;
void init(){
    mem(ch,0);
    mem(val,0);
    cnt=1;
}
int add(int root,int x){
    int newroot=cnt++,ret=newroot;
    for(int i=30;i>=0;i--){
        ch[newroot][0]=ch[root][0];
        ch[newroot][1]=ch[root][1];
        int now=(x>>i)&1;
        root=ch[root][now];
        ch[newroot][now]=cnt++;
        newroot=ch[newroot][now];
        val[newroot]=val[root]+1;
    }
    return ret;
}
int query(int lt,int rt,int x){
    int ans=0;
    for(int i=30;i>=0;i--){
        int now=(x>>i)&1;
        if(val[ch[rt][now^1]]-val[ch[lt][now^1]]){
            ans|=(1<<i);
            rt=ch[rt][now^1];
            lt=ch[lt][now^1];
            } else{
            rt=ch[rt][now];
            lt=ch[lt][now];
        }
    }
    return ans;
}
```

3 String

3.1 Basics

3.1.1 Hash

```cpp
const LL p1 = 201, p2 = 301, mod1 = 1200000319, mod2 = 2147483647;
struct Hash {
    LL a, b;
    void append(Hash pre, int v) {
        a = (pre.a * p1 + v) % mod1;
        b = (pre.b * p2 + v) % mod2;
    }
    void init(string S) {
        a = b = 0;
        for(int i = 0; i < S.size(); i++) append(*this, S[i]);
    }
    bool operator == (const Hash &x) const {
        return a == x.a && b == x.b;
    }
    bool operator < (const Hash &x) const {
        return a < x.a || (a == x.a && b < x.b);
    }
};
```

3.1.2 KMP && exKMP

```cpp
namespace KMP {
    int f[MAXN];
    void get_fail(string A) {
        f[0] = 0; f[1] = 0;
        for(int i = 1; i < A.size(); i++) {
            int j = f[i];
            while(j && A[i] != A[j]) j = f[j];
            f[i + 1] = A[i] == A[j] ? j + 1 : 0;
        }
    }

    void kmp(string A, string B) {
        get_fail(B);
        int j = 0;
        for(int i = 0; i < A.size(); i++) {
            while(j && B[j] != A[i]) j = f[j];
            if(B[j] == A[i]) j++;
            if(j == B.size()) {
                ans++;
                j = f[j];
            }
        }
    }
}
namespace exKMP {
    int nxt[MAXN], ext[MAXN];
    //ext[i]表示S以i开头的后缀与T的前缀相同的长度
    void get_nxt(string T) {
        int j = 0, mx = 0;
        int m = T.size();
        nxt[0] = m;
        for(int i = 1; i < m; i++) {
            if(i >= mx || i + nxt[i - j] >= mx) {
                if(i >= mx) mx = i;
                while(mx < m && T[mx] == T[mx - i]) mx++;
                nxt[i] = mx - i;
                j = i;
            }
            else nxt[i] = nxt[i - j];
        }
    }
    void exkmp(string S, string T) {
        int j = 0, mx = 0;
        get_nxt(T);
        int n = S.size(), m = T.size();
        for(int i = 0; i < n; i++) {
            if(i >= mx || i + nxt[i - j] >= mx) {
                if(i >= mx) mx = i;
                while(mx < n && mx - i < m && S[mx] == T[mx - i]) mx++;
                ext[i] = mx - i;
                j = i;
            }
            else ext[i] = nxt[i - j];
        }
    }
}
```

3.1.3 AC Automaton

```
namespace AC {
    int ch[MAXN][sigma_size], last[MAXN];
    int val[MAXN], f[MAXN], sz;
    inline void init() { sz = 1; memset(ch[0], 0, sizeof(ch[0])); }
    inline int idx(char c) { return c - ’a’; }
    void insert(string s, int v) {
        int u = 0;
        for(int i = 0; i < s.size(); i++) {
            int c = idx(s[i]);
            if(!ch[u][c]) {
                memset(ch[sz], 0, sizeof(ch[sz]));
                val[sz] = 0;
                ch[u][c] = sz++;
            }
            u = ch[u][c];
        }
        val[u] = v;
    }
    void get_fail() {
        queue<int> q;
        f[0] = 0;
        for(int c = 0; c < sigma_size; c++) {
            int u = ch[0][c];
            if(u) { f[u] = 0; q.push(u); last[u] = 0; }
        }
        while(!q.empty()) {
            int r = q.front(); q.pop();
            for(int c = 0; c < sigma_size; c++) {
                int u = ch[r][c];
                if(!u) { ch[r][c] = ch[f[r]][c]; continue; }
                q.push(u);
                int v = f[r];
                while(v && !ch[v][c]) v = f[v];
                f[u] = ch[v][c];
                last[u] = val[f[u]] ? f[u] : last[f[u]];
            }
        }
    }
    inline void solve(int j) {
        if(j) {
            ans += val[j];
            solve(last[j]);
        }
    }
    void find(string T) {
        int j = 0;
        for(int i = 0; i < T.size(); i++) {
            int c = idx(T[i]);
            j = ch[j][c];
            if(val[j]) solve(j);
            else if(last[j]) solve(last[j]);
        }
    }
}
```

3.2 Suffix Related

3.2.1 Suffix Array

```
namespace SA {
    char s[MAXN];
    int sa[MAXN], rank[MAXN], height[MAXN];
    int t[MAXN], t2[MAXN], c[MAXN], n;
    void clear() { n = 0; memset(sa, 0, sizeof(sa)); }
    void build(int m) {
        int *x = t, *y = t2;
        for(int i = 0; i < m; i++) c[i] = 0;
        for(int i = 0; i < n; i++) c[x[i] = s[i]]++;
        for(int i = 1; i < m; i++) c[i] += c[i - 1];
        for(int i = n - 1; i >= 0; i--) sa[--c[x[i]]] = i;
        for(int k = 1; k <= n; k <<= 1) {
            int p = 0;
            for(int i = n - k; i < n; i++) y[p++] = i;
            for(int i = 0; i < n; i++) if(sa[i] >= k) y[p++] = sa[i] - k;
            for(int i = 0; i < m; i++) c[i] = 0;
            for(int i = 0; i < n; i++) c[x[y[i]]]++;
            for(int i = 1; i < m; i++) c[i] += c[i - 1];
            for(int i = n - 1; i >= 0; i--) sa[--c[x[y[i]]]] = y[i];
            swap(x, y);
            p = 1; x[sa[0]] = 0;
            for(int i = 1; i < n; i++)
                x[sa[i]] = y[sa[i - 1]] == y[sa[i]] && y[sa[i - 1] + k] == y[sa[i] + k]
    ? p - 1 : p++;
            if(p >= n) break;
            m = p;
        }
    }
    void buildHeight() {
        int k = 0;
        for(int i = 0; i < n; i++) rank[sa[i]] = i;
        for(int i = 0; i < n; i++) {
            if(k) k--;
            int j = sa[rank[i] - 1];
            while(s[i + k] == s[j + k]) k++;
            height[rank[i]] = k;
        }
    }
    void init() {
        n = strlen(s) + 1;
        build('z' + 1);
        buildHeight();
    }
}
```

3.2.2 Suffix Automaton

3.3 Palindrome Related

3.3.1 Manacher

```
namespace Palindrome {
    char s1[MAXN], s2[MAXN];
    int len1, len2, ans;
    int p[MAXN]; //p[i] - 1
    void init() {
        len1 = strlen(s1);
        s2[0] = '$';
        s2[1] = '#';
        for(int i = 0; i < len1; i++) {
            s2[2 * i + 2] = s1[i];
            s2[2 * i + 3] = '#';
        }
        len2 = len1 * 2 + 2;
        s2[len2] = '&';
    }
    void manacher() {
        int id = 0, mx = 0;
        for(int i = 1; i < len2; i++) {
            if(mx > i) p[i] = min(p[2 * id - i], mx - i);
            else p[i] = 1;
            while(s2[i + p[i]] == s2[i - p[i]]) p[i]++;
            if(i + p[i] > mx) {
                mx = i + p[i];
                id = i;
            }
        }
    }
}
```

3.3.2 Palindromic Tree

```
struct PalindromicTree {
    int len[MAXN]; //节点i表示的回文串的长度
    int fail[MAXN]; //fail指针
    int cnt[MAXN]; //节点i表示的本质不同的串的个数(调用count())
    int num[MAXN]; //以节点i表示的最长回文串的最右端点为回文串结尾的回文串个数
    int nxt[MAXN][sigma_size]; //编号为i的节点表示的回文串在两边添加字符c以后变成的回文串的编号
    int S[MAXN]; //存放添加的字符(S[0]=-1(or '#'))
    int last, p, n;
    int newnode(int l) {
        for(int i = 0; i < sigma_size; i++) nxt[p][i] = 0;
        cnt[p] = 0;
        num[p] = 0;
        len[p] = l;
        return p++;
    }
    inline void init() {
        p = 0;
        newnode(0);
        newnode(-1);
        S[0] = -1;
        n = 0;
        fail[0] = 1;
    }
    int getfail(int x) {
        while(S[n - len[x] - 1] != S[n]) x = fail[x];
        return x;
    }
    void insert(int c) {
        S[++n] = c;
        int cur = getfail(last);
        if(!nxt[cur][c]) {
            int now = newnode(len[cur] + 2);
            fail[now] = nxt[getfail(fail[cur])][c];
            nxt[cur][c] = now;
            num[now] = num[fail[now]] + 1;
        }
        last = nxt[cur][c];
        cnt[last]++;
    }
    void count() {
        for(int i = p - 1; i >= 0; i--) cnt[fail[i]] += cnt[i];
    }
};
```

4 Math

4.1 Algebra

4.1.1 FFT

```
const double pi = acos(-1.0);
const int MAXN = 300003;
struct comp {
    double x, y;
    comp operator + (const comp a) const { return (comp) {x + a.x, y + a.y}; }
    comp operator - (const comp a) const { return (comp) {x - a.x, y - a.y}; }
    comp operator * (const comp a) const { return (comp) {x * a.x - y * a.y, x * a.y + y
     * a.x}; }
};
int rev[MAXN], T;
comp tmp;
void fft(comp *a, int r) {
    if(r == -1) for(int i = 0; i < T; i++) A[i] = A[i] * A[i];
    for(int i = 0; i < T; i++) if(rev[i] > i) swap(a[rev[i]], a[i]);
    for(int i = 2, mid = 1; i <= T; mid = i, i <<= 1) {
        comp step = (comp) {cos(pi / mid), r * sin(pi / mid)};
        for(int j = 0; j < T; j += i) {
            comp cur = (comp) {1, 0};
            for(int k = j; k < j + mid; k++, cur = cur * step) {
                tmp = a[k + mid] * cur;
                a[k + mid] = a[k] - tmp;
                a[k] = a[k] + tmp;
            }
        }
    }
    if(r == -1) for(int i = 0; i < T; i++) a[i].y = (int)(a[i].y / T / 2 + 0.5);
}
int n, m;
comp A[MAXN];
void init() {
    for(T = 1; T <= n + m; T <<= 1);
    for(int i = 1; i < T; i++) {
        if(i & 1) rev[i] = (rev[i >> 1] >> 1) ^ (T >> 1);
        else rev[i] = rev[i >> 1] >> 1;
    }
}
```

4.1.2 NTT

```
const int MAXN = 300005, G = 3, mod = 998244353; //or (479LL<<21) + 1
int rev[MAXN], T;
LL qpow(LL x, LL y) {
    LL res = 1;
    while(y) {
        if(y & 1) res = res * x % mod;
        x = x * x % mod;
        y >>= 1;
    }
    return res;
}
void ntt(LL *a, int r) {
    if(r == -1) for(int i = 0; i < T; i++) A[i] = A[i] * B[i] % mod;
    for(int i = 0; i < T; i++) if(rev[i] > i) swap(a[rev[i]], a[i]);
    for(int i = 2, mid = 1; i <= T; mid = i, i <<= 1) {
        LL gn = qpow(G, (mod - 1) / i);
        if(r == -1) gn = qpow(gn, mod - 2);
        for(int j = 0; j < T; j += i) {
            LL cur = 1, tmp;
            for(int k = j; k < j + mid; k++, cur = cur * gn % mod) {
                tmp = a[k + mid] * cur % mod;
                a[k + mid] = ((a[k] - tmp) % mod + mod) % mod;
                a[k] = (a[k] + tmp) % mod;
            }
        }
    }
    if(r == -1) {
        LL inv = qpow(T, mod - 2);
        for(int i = 0; i < T; i++) a[i] = a[i] * inv % mod;
    }
}
int n, m;
LL A[MAXN], B[MAXN];
void init() {
    for(T = 1; T <= n + m; T <<= 1);
    for(int i = 0; i < T; i++) {
        if(i & 1) rev[i] = (rev[i >> 1] >> 1) ^ (T >> 1);
        else rev[i] = rev[i >> 1] >> 1;
    }
}
```

4.1.3 Linear Basis

```
int Gauss(int n, int m) {
    int num = 1;
    for(int x = 1; x <= n && x <= m; x++) {
        int t = 0;
        for(int j = x; j <= m; j++) if(g[j][x]) { t = j; break; }
        if(t) {
            swap(g[x], g[t]);
            for(int i = x + 1; i <= n; i++) {
                if(g[i][x]) {
                    for(int k = 1; k <= m; k++) g[i][k] ^= g[x][k];
                }
            }
            num++;
        }
    }
    return --num;
}
//long long
int Gauss() {
    int num = 1;
    for(int k = 61; k >= 0; k--) {
        int t = 0;
        for(int j = num; j <= cnt; j++) if((A[j] >> k) & 1) { t = j; break; }
        if(t) {
            swap(A[t], A[num]);
            for(int j = num + 1; j <= cnt; j++) if((A[j] >> k) & 1) A[j] ^= A[num];
            num++;
        }
    }
    return --num;
}
```

4.2 Math Theory

4.2.1 Inverse

```
//O(logn)求n的逆元
const int mod = 1e6 + 3;
int exgcd(int a, int b, int &x, int &y) {
    int d = a;
    if(b != 0) {
        d = exgcd(b, a % b, y, x);
        y -= (a / b) * x;
    }
    else {
        x = 1; y = 0;
    }
    return d;
}
int inverse(int a) {
    int x, y;
    exgcd(a, mod, x, y);
    return (x % mod + mod) % mod;
}
int inverse(int a) { return qpow(a, mod - 2); }
//O(n)求1~n的逆元
int inv[MAXN];
void init() {
    inv[0] = inv[1] = 1;
    for(int i = 2; i < MAXN; i++) inv[i] = (long long)(mod - mod / i) * inv[mod % i] % mod;
}
```

4.2.2 Lucas

```
//mod很小可以预处理逆元的情况
void init() {
    fac[0] = 1;
    for(int i = 1; i < mod; i++) fac[i] = (long long)fac[i - 1] * i % mod;
    inv[0] = inv[1] = 1;
    for(int i = 2; i < mod; i++) inv[i] = (long long)(mod - mod / i) * inv[mod % i] % mod;
    for(int i = 1; i < mod; i++) inv[i] = (long long)inv[i] * inv[i - 1] % mod;
}
int C(int a, int b) {
    if(b > a) return 0;
    if(a < mod) return (long long)fac[a] * inv[b] % mod * inv[a - b] % mod;
    return (long long)C(a / mod, b / mod) * C(a % mod, b % mod) % mod;
}
//mod过大不能预处理逆元的情况
LL qpow(LL x, LL y) {
    LL res = 1;
    while(y) {
        if(y & 1) res = res * x % mod;
        x = x * x % mod;
        y >>= 1;
    }
    return res;
}
LL C(LL a, LL b) {
    if(b > a) return 0;
    if(b > a - b) b = a - b;
    LL s1 = 1, s2 = 1;
    for(LL i = 0; i < b; i++) {
        s1 = s1 * (a - i) % mod;
        s2 = s2 * (i + 1) % mod;
    }
    return s1 * qpow(s2, mod - 2) % mod;
}
LL lucas(LL a, LL b) {
    if(a < mod) return C(a, b);
    return lucas(a / mod, b / mod) * C(a % mod, b % mod);
}
```

4.2.3 CRT && exCRT

```
namespace CRT {
    LL m[MAXN], a[MAXN]; //x_i = a[i] (mod m[i])
    LL exgcd(LL _a, LL _b, LL &x, LL &y) {
        if(!_b) {
            x = 1; y = 0;
            return _a;
        }
        LL d = exgcd(_b, _a % _b, y, x);
        y -= (_a / _b) * x;
        return d;
    }
    LL crt(int n) {
        LL M = 1, tmp, res = 0, x, y;
        for(int i = 1; i <= n; i++) M *= m[i];
        for(int i = 1; i <= n; i++) {
            tmp = M / m[i];
            exgcd(tmp, m[i], x, y);
            x = (x + m[i]) % m[i];
            res = (a[i] * x % M * tmp % M + res) % M;
        }
        return res;
    }
}
namespace EXCRT {
    LL m[MAXN], a[MAXN];
    LL exgcd(LL _a, LL _b, LL &x, LL &y) {
        if(!_b) {
            x = 1; y = 0;
            return _a;
        }
        LL d = exgcd(_b, _a % _b, y, x);
        y -= (_a / _b) * x;
        return d;
    }
    LL excrt(int n) {
        LL M = m[1], A = a[1], x, y, d, tmp;
        for(int i = 2; i <= n; i++) {
            d = exgcd(M, m[i], x, y);
            if((A - a[i]) % d) return -1; //No solution
            tmp = M / d; M *= m[i] / d;
            y = (A - a[i]) / d % M * y % M;
            y = (y + tmp) % tmp;
            A = (m[i] % M * y % M + a[i]) % M;
            A = (A + M) % M;
        }
        return A;
    }
}
```

4.2.4 BSGS

```
const int MOD = 76543;
int hs[MOD + 5], head[MOD + 5], nxt[MOD + 5], id[MOD + 5], ecnt;
void insert(int x, int y) {
    int k = x % MOD;
    hs[ecnt] = x, id[ecnt] = y, nxt[ecnt] = head[k], head[k] = ecnt++;
}
int find(int x) {
    int k = x % MOD;
    for(int i = head[k]; i; i = nxt[i])
        if(hs[i] == x)
            return id[i];
    return -1;
}
int BSGS(int a, int b, int c){
    memset(head, 0, sizeof head); ecnt = 1;
    if(b == 1) return 0;
    int m = sqrt(c * 1.0), j;
    LL x = 1, p = 1;
    for(int i = 0; i < m; i++, p = p * a % c)
        insert(p * b % c, i);
    for(LL i = m; ;i += m){
        if((j = find(x = x * p % c)) != -1) return i - j;
        if(i > c) break;
    }
    return -1;
}
```

4.2.5 Miller-Rabin && PollardRho

```
LL ksc(LL a,LL n,LL mod){
    LL ret=0;
    for(;n;n>>=1){
        if(n&1){ret+=a;if(ret>=mod)ret-=mod;}
        a<<=1;if(a>=mod)a-=mod;
    }
    return ret;
}
LL ksm(LL a,LL n,LL mod){
    LL ret = 1;
    for(;n;n>>=1){
        if(n&1)ret=ksc(ret,a,mod);
        a=ksc(a,a,mod);
    }
    return ret;
}
int millerRabin(LL n){
    if(n<2 || (n!=2 && !(n&1)))return 0;
    LL d=n-1;for(;!(d&1);d>>=1);
    for(int i=0;i<20;++i){
        LL a=rand()%(n-1)+1;
        LL t=d,m=ksm(a,d,n);
        for(;t!=n-1 && m!=1 && m!=n-1;m=ksc(m,m,n),t<<=1);
        if(m!=n-1 && !(t&1)) return 0;
    }
    return 1;
}
LL cnt,fact[100];
LL gcd(LL a,LL b){return !b?a:gcd(b,a%b);}
LL pollardRho(LL n, int a){
    LL x=rand()%n,y=x,d=1,k=0,i=1;
    while(d==1){
        ++k;
        x=ksc(x,x,n)+a;if(x>=n)x-=n;
        d=gcd(x>y?x-y:y-x,n);
        if(k==i){y=x;i<<=1;}
    }
    if(d==n)return pollardRho(n,a+1);
    return d;
}
void findfac(LL n){
    if(millerRabin(n)){fact[++cnt]=n;return;}
    LL p=pollardRho(n,rand()%(n-1)+1);
    findfac(p);
    findfac(n/p);
}
```

4.2.6 $\Phi(n)$

```
int phi(int x) {
    int res = x;
    for(int i = 2; i * i <= x; i++) {
        if(x % i == 0) {
            res = res / i * (i - 1);
            while(x % i == 0) x /= i;
        }
    }
    if(x > 1) res = res / x * (x - 1);
    return res;
}
```

4.2.7 Euler Sieve

```
int prime[MAXN], cnt, phi[MAXN], mu[MAXN];
bool isp[MAXN];

int min_pow[MAXN];   //最小质因子最高次幂
int min_sum[MAXN];   //1+p+p^2+...+p^k
int div_sum[MAXN];   //约数和

int min_index[MAXN]; //最小质因子的指数
int div_num[MAXN];   //约数个数
void Euler(int n) {
    mu[1] = phi[1] = div_num[1] = div_sum[1] = 1;
    for(int i = 2; i <= n; i++) {
        if(!isp[i]) {
            prime[++cnt] = min_pow[i] = i;
            phi[i] = i - 1;
            mu[i] = -1;
            min_index[i] = 1; div_num[i] = 2;
            div_sum[i] = min_sum[i] = i + 1;
        }
        for(int j = 1; j <= cnt && i * prime[j] <= n; j++) {
            isp[i * prime[j]] = 1;
            if(i % prime[j] == 0) {
                phi[i * prime[j]] = phi[i] * prime[j];
                mu[i * prime[j]] = 0;

                min_index[i * prime[j]] = min_index[i] + 1;
                div_num[i * prime[j]] = div_num[i] / (min_index[i] + 1) * (min_index[i * prime[j]] + 1);

                min_sum[i * prime[j]] = min_sum[i] + min_pow[i] * prime[j];
                div_sum[i * prime[j]] = div_sum[i] / min_sum[i] * min_sum[i * prime[j]];
                min_pow[i * prime[j]] = min_pow[i] * prime[j];
                break;
            }
            phi[i * prime[j]] = phi[i] * (prime[j] - 1);
            mu[i * prime[j]] = -mu[i];

            div_num[i * prime[j]] = div_num[i] << 1;
            min_index[i * prime[j]] = 1;

            div_sum[i * prime[j]] = div_sum[i] * (prime[j] + 1);
            min_pow[i * prime[j]] = prime[j];
            min_sum[i * prime[j]] = prime[j] + 1;
        }
    }
}
```

4.2.8 Möbius Inversion

$$\sum_i^n \sum_j^m lcm(i,j)(mod\ p)$$

```
int mu[MAXN], prime[MAXN], sum[MAXN], cnt;
bool isp[MAXN];
void getmu(int n) {
    mu[1] = 1;
    for(int i = 2; i <= n; i++) {
        if(!isp[i]) {
            mu[i] = -1;
            prime[++cnt] = i;
        }
        for(int j = 1; j <= cnt && i * prime[j] <= n; j++) {
            isp[i * prime[j]] = 1;
            if(i % prime[j] == 0) {
                mu[i * prime[j]] = 0;
                break;
            }
            mu[i * prime[j]] = -mu[i];
        }
    }
}
ll n, m, ans;
ll query(ll x, ll y) { return (x * (x + 1) / 2 % mod) * (y * (y + 1) / 2 % mod) % mod; }
ll F(ll x, ll y) {
    ll res = 0, last;
    for(ll i = 1; i <= min(x, y); i = last + 1) {
        last = min(x / (x / i), y / (y / i));
        res = (res + (sum[last] - sum[i - 1]) * query(x / i, y / i) % mod) % mod;
    }
    return res;
}
int main() {
    cin>>n>>m;
    getmu(min(n, m));
    for(ll i = 1; i <= min(n, m); i++) sum[i] = (sum[i - 1] + (i * i * mu[i]) % mod) % mod;
    ll last;
    for(ll d = 1; d <= min(n, m); d = last + 1) {
        last = min(n / (n / d), m / (m / d));
        ans = (ans + (last - d + 1) * (d + last) / 2 % mod * F(n / d, m / d) % mod) % mod;
    }
    ans = (ans + mod) % mod;
    cout<<ans<<endl;
    return 0;
}
```

5 Geometry

5.1 Commonly Definition and Functions

5.1.1 Const and Functions

```
namespace CG{
    #define Point Vector
    const double pi=acos(-1.0);
    const double inf=1e100;
    const double eps=1e-9;
    template <typename T> inline T Abs(T x){return x>0?x:-x;}
    template <typename T> inline bool operator == (T x,T y){return Abs(x-y)<eps;}
    int sgn(double x){
        if (Abs(x)<eps) return 0;
        if (x>0) return 1;
        else return -1;
    }
}
```

5.1.2 Point Definition

```
namespace CG{
    struct Point{
        double x,y;
        Point(double x=0,double y=0):x(x),y(y){}
    };
    Vector operator + (const Vector a,const Vector b){return Vector(a.x+b.x,a.y+b.y);}
    Vector operator - (const Vector a,const Vector b){return Vector(a.x-b.x,a.y-b.y);}
    Vector operator * (const Vector a,const double k){return Vector(a.x*k,a.y*k);}
    Vector operator / (const Vector a,const double k){return Vector(a.x/k,a.y/k);}
    bool operator < (const Vector a,const Vector b) {return a.x==b.x?a.y<b.y:a.x<b.x;}
    bool operator == (const Vector a,const Vector b) {return a.x==b.x && a.y==b.y;}
    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 Norm(const Vector a){return sqrt(Dot(a,a));}
    double Angle(const Vector a,const Vector b){return acos(Dot(a,b)/Norm(a)/Norm(b));}
    Vector Rotate(const Vector a,const double theta){return Vector(a.x*cos(theta)-a.y*sin(theta),a.x*sin(theta)+a.y*cos(theta));}
    bool ToLeftTest(const Vector a,const Vector b){return Cross(a,b)<0;}
    double DisPP(const Vector a,const Vector b){return sqrt((a.x-b.x)*(a.x-b.x)+(a.y-b.y)*(a.y-b.y));}
}
```

5.1.3 Line Definition

```
namespace CG{
    struct Line{
        Point p0,v,p1;
        double t,theta;
        Line(Point _p0=0,Point _v=0,double _t=1):p0(_p0),v(_v),t(_t){p1=p0+v*t; theta=atan2(v.y,v.x);}
        // Line(Point _p0=0,Point _v=0,double _t=1):p0(_p0),p1(_v){v=(p1-p0)/t; theta=atan2(v.y,v.x);}
    };
    bool operator < (const Line n,const Line m) {return n.theta<m.theta;}
    Point GetIntersection(const Line n,const Line m){return n.p0+n.v*Cross(m.v,(n.p0-m.p0))/Cross(n.v,m.v);}
    bool OnLine(const Vector a,const Line l){return Cross(l.p0-a,l.p1-a)==0;}
    bool OnSegment(const Point a,const Line l){return sgn(Cross(l.p0-a,l.p1-a))==0 && sgn(Dot(l.p0-a,l.p1-a))<0;}
    double DisPL(const Point a,const Line l){return Abs(Cross(l.p1-l.p0,a-l.p0)/Norm(l.p1-l.p0));}
    double DisPS(const Point a,const Line l){
        if (l.p0==l.p1) return Norm(a-l.p0);
        Vector v1=l.p1-l.p0,v2=a-l.p0,v3=a-l.p1;
        if (sgn(Dot(v1,v2))<0) return Norm(v2);
        if (sgn(Dot(v1,v3))>0) return Norm(v3);
        return DisPL(a,l);
    }
    Point GetProjection(const Point a,const Line l){
        Vector v=l.p1-l.p0;
        return l.p0+v*(Dot(v,a-l.p0)/Dot(v,v));
    }
    bool SegmentIntersection(const Line n,const Line m,bool p){
        double c1=Cross(n.p1-n.p0,m.p1-m.p0);
        double c2=Cross(n.p1-n.p0,m.p1-n.p0);
        double c3=Cross(m.p1-m.p0,n.p0-m.p0);
        double c4=Cross(m.p1-m.p0,n.p1-m.p0);
        if (p){
            if (!sgn(c1) || !sgn(c2) || !sgn(c3) || !sgn(c4)){
                return OnSegment(n.p0,m) || OnSegment(n.p1,m) || OnSegment(m.p0,n) || OnSegment(m.p0,m);

            }
        }
        return (sgn(c1)*sgn(c2)<0 && sgn(c3)*sgn(c4)<0);
    }
}
```

5.1.4 Get Area

```
namespace CG{
    double GetArea(Point *p,int n){
        double area=Cross(p[n],p[1]);
        for (int i=2;i<=n;i++) area+=0.5*Cross(p[i-1],p[i]);
        return Abs(area);
    }
}
```

5.1.5 Get Circumference

```
namespace CG{
    double GetCircumference(Point *p,int n){
        double Circumference=DisPP(p[n],p[1]);
        for (int i=2;i<=n;i++) Circumference+=DisPP(p[i-1],p[i]);
        return Circumference;
    }
}
```

5.2 Convex Hull

```
namespace CG{
    Point p[MAXN],s[MAXN];
    int ConvexHull(Point *p,int n){
        sort(p+1,p+1+n);
        int m=0;
        for (int i=1;i<=n;i++){
            for (;m>=2 && !ToLeftTest(s[m]-s[m-1],p[i]-s[m-1]);m--);
            s[++m]=p[i];
        }
        int k=m;
        for (int i=n-1;i;i--){
            for (;m>=k+1 && !ToLeftTest(s[m]-s[m-1],p[i]-s[m-1]);m--);
            s[++m]=p[i];
        }
        return m-1;
    }
}
```

5.3 Half Plane Intersection

```
namespace CG{
    void HalfPlaneIntersection(Line l[],int n){
        deque <Point> p;
        sort(l+1,l+1+n);
        deque <Line> q;
        q.push_back(l[1]);
        for (int i=2;i<=n;i++){
            for (;!p.empty() && !ToLeftTest(p.back()-l[i].p0,l[i].v);q.pop_back(),p.pop_back());
            for (;!p.empty() && !ToLeftTest(p.front()-l[i].p0,l[i].v);q.pop_front(),p.pop_front());
            if (sgn(Cross(l[i].v,q.back().v))==0)
                if (ToLeftTest(l[i].p0-q.back().p0),q.back().v){
                    q.pop_back();
                    if (!p.empty()) p.pop_back();
                }
            if (!q.empty()) p.push_back(GetIntersection(q.back(),l[i]));
            q.push_back(l[i]);
        }
        for (;!p.empty() && !ToLeftTest(p.back()-q.front().p0,q.front().v);q.pop_back(),p.pop_back());
        p.push_back(GetIntersection(q.back(),q.front()));
        double area=0.5*Cross(p.back(),p.front()); Point last=p.front();
        for (p.pop_front();!p.empty();last=p.front(),p.pop_front()) area+=0.5*Cross(last,p.front());
        printf("%.1f",Abs(area));
    }
}
```

5.4 Min Circle Cover

```
namespace CG{
    Point GetCircleCenter(const Point a,const Point b,const Point c){
        Point p=(a+b)/2.0,q=(a+c)/2.0;
        Vector v=Rotate(b-a,pi/2.0),w=Rotate(c-a,pi/2.0);
        if (sgn(Norm(Cross(v,w)))==0){
            if (sgn(Norm(a-b)+Norm(b-c)-Norm(a-c))==0) return (a+c)/2;
            if (sgn(Norm(b-a)+Norm(a-c)-Norm(b-c))==0) return (b+c)/2;
            if (sgn(Norm(a-c)+Norm(c-b)-Norm(a-b))==0) return (a+c)/2;
        }
        return GetIntersection(Line(p,v),Line(q,w));
    }
    void MinCircleCover(Point p[],int n){
        random_shuffle(p+1,p+1+n);
        Point c=p[1];
        double r=0;
        for (int i=2;i<=n;i++)
            if (sgn(Norm(c-p[i])-r)>0){
                c=p[i],r=0;
                for (int j=1;j<i;j++)
                    if (sgn(Norm(c-p[j])-r)>0){
                        c=(p[i]+p[j])/2.0;
                        r=Norm(c-p[i]);
                        for (int k=1;k<j;k++)
                            if (sgn(Norm(c-p[k])-r)>0){
                                c=GetCircleCenter(p[i],p[j],p[k]);
                                r=Norm(c-p[i]);
                            }
                    }
            }
        printf("%.10f\n%.10f %.10f",r,c.x,c.y);
    }
}
```

5.5 Circle Union Area

```
//k次覆盖
//圆并去重后s[0]
typedef pair<double, int> P;
const double pi = acos(-1.0);
const int MAXN = 10003;
P arc[MAXN << 1];
int acnt, cnt;
double s[1003];
bool del[1003];
void add(double st, double en) {
    if(st < -pi) {
        add(st + 2 * pi, pi);
        add(-pi, en);
        return;
    }
    if(en > pi) {
        add(st, pi);
        add(-pi, en - 2 * pi);
        return;
    }
    arc[++acnt] = P(st, 1);
    arc[++acnt] = P(en, -1);
}
double F(double x) {
    return (x - sin(x)) / 2;
}
struct Node {
    int x, y, r;
    Node(int _x = 0, int _y = 0, int _r = 0):x(_x), y(_y), r(_r) {}
    bool operator == (const Node& t) {
        return x == t.x && y == t.y && r == t.r;
    }
    inline void read() {
        scanf("%d%d%d", &x, &y, &r);
    }
}a[1003];
int main() {
    int n;
    scanf("%d", &n);
    for(int i = 1; i <= n; i++) a[i].read();
    /*
    //去重
    int nn = 0;
    for(int i = 1; i <= n; i++) {
        bool same = 0;
        for(int j = 1; j < i; j++) {
            if(a[i] == a[j]) {
                same = 1; break;
            }
        }
        if(!same) a[++nn] = a[i];
    }
    n = nn;
    //去包含
    for(int i = 1; i <= n; i++) {
        for(int j = 1; j <= n; j++) if(i != j) {
            if(hypot(a[i].x - a[j].x, a[i].y - a[j].y) < (double)(a[i].r - a[j].r)) del[j] = 1;
        }
    }
    nn = 0;
    for(int i = 1; i <= n; i++) if(!del[i]) {
        a[++nn] = a[i];
    }
    n = nn;
    */
    for(int i = 1; i <= n; i++) {
        acnt = 0;
        for(int j = 1; j <= n; j++) if(i != j) {
            int dis = (a[i].x - a[j].x) * (a[i].x - a[j].x) + (a[i].y - a[j].y) * (a[i].y - a[j].y);
            if(a[j].r > a[i].r && dis <= (a[j].r - a[i].r) * (a[j].r - a[i].r)) add(-pi, pi);
            else if(dis > (a[i].r - a[j].r) * (a[i].r - a[j].r) && dis < (a[i].r + a[j].r) * (a[i].r + a[j].r)){
                double c = sqrt(dis);
                double angle = acos((a[i].r * a[i].r + c * c - a[j].r * a[j].r) / (2 * a[i].r * c));
                double k = atan2(a[j].y - a[i].y, a[j].x - a[i].x);
                add(k - angle, k + angle);
            }
        }
        arc[++acnt] = P(pi, -1);
        sort(arc + 1, arc + acnt + 1);
        cnt = 0;
        double last = -pi;
        for(int j = 1; j <= acnt; j++) {
            s[cnt] += F(arc[j].first - last) * a[i].r * a[i].r; //扇形 - 三角形
            double xa = a[i].x + a[i].r * cos(last);
            double ya = a[i].y + a[i].r * sin(last);
            last = arc[j].first;
            double xb = a[i].x + a[i].r * cos(last);
            double yb = a[i].y + a[i].r * sin(last);
            s[cnt] += (xa * yb - xb * ya) / 2; //到圆心的三角形面积
            cnt += arc[j].second;
        }
    }
    //printf("%.3f\n", s[0]);
    for (int i = 0; i < n; i++) {
        printf("[%d] = %.3f\n", i + 1, s[i] - s[i + 1]);
    }
    return 0;
}
```

5.6 Simpson Integrate

```
double Simpson(double l,double r){
    return (r-l)*(F(l)+4*F((l+r)/2)+F(r))/6;
}
double Integrate(double l,double r,double S){
    double mid=(l+r)/2;
    double A=Simpson(l,mid);
    double B=Simpson(mid,r);
    if(A+B-S<eps)return S;
    return Integrate(l,mid,A)+Integrate(mid,r,B);
}
```

6 Others

6.1 Sample

6.1.1 vimrc

```
set nocompatible
source $VIMRUNTIME/vimrc_example.vim
source $VIMRUNTIME/mswin.vim
nunmap <c-v>
set cindent
set number
set mouse=a
set tabstop=4
set shiftwidth=4
set cursorline
set guifont=Consolas:h12
inoremap kj <esc>
inoremap jk <esc>
inoremap { {}<left>
syntax enable
func! Compile()
    exec "w"
    exec "! g++ % -o %< -Wall -Wextra -Wshadow -Wconversion --std=c++14 -O2"
    exec "! ./%<"
endfunc
func! Debug()
    exec "w"
    exec "! g++ % -o %< -g -Wall --std=c++14 && gdb %<"
endfunc
func! AddTitle()
    call append(0,"// Cease to struggle and you cease to live")
    call append(1,"#include <bits/stdc++.h>")
    call append(2,"using namespace std;")
    call append(3,"")
    call append(4,"int main() {")
    call append(5,"    ios::sync_with_stdio(0); cin.tie(0); cout.precision(6); cout << fixed;")
    call append(6,"")
    call append(7,"    return 0;")
    call append(8,"}")
endfunc
map <F9> :call Compile()<CR>
map <F5> :call Debug()<CR>
map <F8> :call AddTitle()<CR>
```

6.1.2 FastIO

```
namespace IO {
    const int MB = 1048576;
    const int RMAX = 16 * MB;
    const int WMAX = 16 * MB;
    #define getchar() *(rp++)
    #define putchar(x) (*(wp++) = (x))
    char rb[RMAX], *rp = rb,  wb[WMAX], *wp = wb;
    inline void init() {
        fread(rb, sizeof(char), RMAX, stdin);
    }
    template <class _T> inline void read(_T &_a) {
        _a = 0; register bool _f = 0; register int _c = getchar();
        while (_c < '0' || _c > '9') _f |= _c == '-', _c = getchar();
        while (_c >= '0' && _c <= '9') _a = _a * 10 + (_c ^ '0'), _c = getchar();
        _a = _f ? -_a : _a;
    }
    template <class _T> inline void write(_T _a) {
        static char buf[20], *top = buf;
        if (_a) {
            while (_a) {
                register _T tm = _a / 10;
                *(++top) = char(_a - tm * 10) | '0';
                _a = tm;
            }
            while (top != buf) putchar(*(top--));
        }
        else putchar('0');
    }
    void output() {
        fwrite(wb, sizeof(char), wp - wb, stdout);
    }
}
```

6.1.3 Java BigNum

```
import java.math.*;
import java.util.*;
public class Main{
    public static void main(String []args){
        Scanner in = new Scanner(System.in);
        while(in.hasNext()){} //EOF
        BigInteger zero = BigInteger.valueOf(0);
        BigInteger a = in.nextBigInteger();
        BigInteger b = in.nextBigInteger();
        BigInteger c = in.nextBigInteger();
        int d = in.nextInt();
        a.add(b);
        a.subtract(b);
        a.multiply(b);
        a.divide(b);
        a.mod(b);
        a.compareTo(b);
        a.negate();
        a.modInverse(b); //a^(-1)
        a.modPow(b,c); //a^b%c
        a.pow(d);
    }
}
```

6.2 Offline Algorithm

6.2.1 CDQ Divide and Conquer

```
struct Node {
    int x, y, z, ans;
    Node() {}
    Node(int _x, int _y, int _z):x(_x), y(_y), z(_z) {}
    bool operator < (const Node &b) const {
        if(y == b.y) {
            if(z == b.z) return x < b.x;
            return z < b.z;
        }
        return y < b.y;
    }
}A[MAXN], B[MAXN], C[MAXN];
int bit[MAXN];
void add(int k, int v) {
    for(; k <= m; k += k & -k) bit[k] = max(bit[k], v);
}
void clear(int k) {
    for(; k <= m; k += k & -k) bit[k] = 0;
}
int sum(int k) {
    int res = 0;
    for(; k; k -= k & -k) res = max(res, bit[k]);
    return res;
}
void solve(int l, int r) {
    if(l == r) {
        B[l] = A[l];
        return;
    }
    int mid = (l + r) >> 1;
    solve(l, mid);
    for(int i = mid + 1; i <= r; i++) B[i] = A[i];
    //sort(B + l, B + mid + 1);
    sort(B + mid + 1, B + r + 1);
    int L = l;
    for(int R = mid + 1; R <= r; R++) {
        while(L <= mid && B[L].y < B[R].y) add(B[L].z, B[L].ans), L++;
        A[B[R].x].ans = max(A[B[R].x].ans, sum(B[R].z - 1) + 1);
        B[R].ans = A[B[R].x].ans;
    }
    for(int i = l; i <= L; i++) clear(B[i].z);
    solve(mid + 1, r);
    L = l;
    int p = l, q = mid + 1;
    while(p <= mid || q <= r) {
        if(q > r || (p <= mid && B[p].y <= B[q].y)) C[L++] = B[p++];
        else C[L++] = B[q++];
    }
    for(int i = l; i <= r; i++) B[i] = C[i];
}
```

6.2.2 Mo’ s Algorithm

```
struct Node{
    int l, r, t, id;
    bool operator < (const Node& a) const {
        if(l /sz == a.l / sz) {
            if(r == a.r) return t < a.t;
            return r < a.r;
        }
        return l / sz < a.l / sz;
    }
}q[MAXN];
void solve() {
    while (t < q[i].t) addTime(t++, 1);
```

```cpp
    while (t > q[i].t) addTime(--t, -1);
    while(L < q[i].l) add(L++, -1);
    while(L > q[i].l) add(--L, 1);
    while(R < q[i].r) add(++R, 1);
    while(R > q[i].r) add(R--, -1);
}
```

6.2.3 Mo's Algorithm On Tree

```cpp
struct Edge {
    int to, nxt;
}e[MAXN << 1];
int head[MAXN], ecnt;
int stack[MAXN], top, belong[MAXN], cnt, sz;
struct Node {
    int l, r, id, ti;
    bool operator < (const Node &x) const {
        return belong[l] < belong[x.l] || (belong[l] == belong[x.l] && belong[r] < belong[x.r]) || (belong[l] == belong[x.l] && belong[r] == belong[x.r] && ti < x.ti);
    }
}q[MAXN];
struct Node2 {
    int l, r, ti;
}qq[MAXN];
int n, m, Q, Q0, Q1;
int V[MAXN], W[MAXN], C[MAXN];
int fa[MAXN][S + 3], dep[MAXN];
long long ans[MAXN], tans;
int vis[MAXN], cur[MAXN];
long long sum[MAXN];
int l, r, tm;
inline int read() {
    int x = 0; char ch = getchar(); bool fg = 0;
    while(ch < '0' || ch > '9') { if(ch == '-') fg = 1; ch = getchar(); }
    while(ch >= '0' && ch <= '9') { x = x * 10 + ch - '0'; ch = getchar(); }
    return fg ? -x : x;
}
inline void add_edge(int u, int v) {
    e[++ecnt] = (Edge) {v, head[u]}; head[u] = ecnt;
    e[++ecnt] = (Edge) {u, head[v]}; head[v] = ecnt;
}
void dfs(int u, int f) {
    fa[u][0] = f;
    dep[u] = dep[f] + 1;
    int bot = top;
    for(int i = head[u]; i; i = e[i].nxt) {
        int v = e[i].to;
        if(v == f) continue;
        dfs(v, u);
        if(top - bot >= sz) {
            cnt++;
            while(top != bot) belong[stack[top--]] = cnt;
        }
    }
    stack[++top] = u;
}
void G(int &u, int step) {
    for(int i = 0; i < S; i++) if((1 << i) & step) u = fa[u][i];
}
int lca(int u, int v) {
    if(dep[u] > dep[v]) swap(u, v);
    G(v, dep[v] - dep[u]);
    if(u == v) return u;
    for(int i = S; i >= 0; i--) if(fa[u][i] != fa[v][i]) {
        u = fa[u][i]; v = fa[v][i];
    }
    return fa[u][0];
}
inline void modify(int u) {
    tans -= V[C[u]] * sum[cur[C[u]]];
    cur[C[u]] += vis[u];
    vis[u] = -vis[u];
    tans += V[C[u]] * sum[cur[C[u]]];
}
inline void update(int u, int v) {
    if(u == v) return;
    if(dep[u] > dep[v]) swap(u, v);
    while(dep[v] > dep[u]) {
        modify(v);
        v = fa[v][0];
    }
    while(u != v) {
        modify(u); modify(v);
        u = fa[u][0]; v = fa[v][0];
    }
}
inline void upd(int t) {
    if(vis[qq[t].l] == -1) {
        modify(qq[t].l);
        swap(C[qq[t].l], qq[t].r);
        modify(qq[t].l);
    }
    else swap(C[qq[t].l], qq[t].r);
}
inline void moveto(int u, int v) {
    update(l, u); update(r, v);
    l = u; r = v;
}
int main() {
    n = read(); m = read(); Q = read();
    sz = (int)pow(n, 2.0 / 3.0);
    for(int i = 1; i <= m; i++) V[i] = read();
    for(int i = 1; i <= n; i++) W[i] = read();
    for(int i = 1, u, v; i < n; i++) {
        u = read(); v = read();
        add_edge(u, v);
    }
    for(int i = 1; i <= n; i++) {
        C[i] = read();
        vis[i] = 1;
        sum[i] = sum[i - 1] + W[i];
    }
    for(int i = 1, tp; i <= Q; i++) {
        tp = read();
        if(tp) {
            ++Q1;
            q[Q1].l = read(); q[Q1].r = read();
            q[Q1].id = Q1;
            q[Q1].ti = i;
        }
        else {
            ++Q0;
            qq[Q0].l = read(); qq[Q0].r = read();
            qq[Q0].ti = i;
        }
    }
    dfs(1, 0);
    while(top) belong[stack[top--]] = cnt;
    sort(q + 1, q + Q1 + 1);
    for(int k = 1; k <= S; k++) {
        for(int i = 1; i <= n; i++) {
            fa[i][k] = fa[fa[i][k - 1]][k - 1];
        }
    }
    for(int i = 1; i <= Q1; i++) {
        if(belong[q[i].l] > belong[q[i].r]) swap(q[i].l, q[i].r);
        moveto(q[i].l, q[i].r);
        int lc = lca(l, r);
        modify(lc);
        while(qq[tm + 1].ti < q[i].ti && tm < Q0) upd(++tm);
        while(qq[tm].ti > q[i].ti) upd(tm--);
        ans[q[i].id] = tans;
        modify(lc);
    }
    for(int i = 1; i <= Q1; i++) printf("%lld\n", ans[i]);
    return 0;
}
```

6.3 Randomized Algorithm

6.3.1 Simulated Annealing

```
void solve() {
    while(T > eps) {
        double alpha = ((rand() % 30001) / 15000.0) * pi;
        double theta = ((rand() % 10001) / 10000.0) * pi;
        tmp.x = cur.x + T * sin(theta) * cos(alpha);
        tmp.y = cur.y + T * sin(theta) * sin(alpha);
        tmp.z = cur.z + T * cos(theta);
        tmp.dis = cal(tmp);
        if(tmp.dis < cur.dis || (tmp.dis * 0.999 < cur.dis && (rand() & 7) == 7)) cur =
    tmp;
        //if(exp((cur.d - tmp.d) / T) > ((double)rand() / RAND_MAX)) cur = tmp;

        T *= 0.999;
    }
}
```

6.4 Other Method

6.4.1 Enumerate Subset

```
for(int i = 0; i < (1 << k); i++) {
    for(int j = i; ; --j &= i) {
        // work();
        if(j == 0) break;
    }
}
```

6.4.2 Enumerate $\lfloor\frac{n}{d}\rfloor\lfloor\frac{m}{d}\rfloor$

```
int cal(int n, int m) {
    if(n > m) swap(n, m);
    int res = 0, last;
    for(int i = 1; i <= n; i = last + 1) {
        last = min(n / (n / i), m / (m / i));
        res += (n / i) * (m / i) * (sum(last) - sum(i - 1));
    }
    return res;
}
```

6.5 Formula

6.5.1 Euler's theorem

$$a^x \equiv \left\{\begin{array}{ll} a^{b\%\phi(p)} & \gcd(a,p)=1 \\ a^b & \gcd(a,p)\neq 1, b<\phi(p) \\ a^{b\%\phi(p)+\phi(p)} & \gcd(a,p)\neq 1, b\geq\phi(p) \end{array}\right\} (mod\ p)$$

6.5.2 Dirichlet Convolution

$$(f\times g)(N)=\sum_{d|N} f(d)*g(\frac{N}{d})$$

$$\sum_{d|N}\phi(d)*\frac{N}{d}=\phi\times id$$

$$n=\sum_{d|n}\phi(d)$$

$$e(n)=\sum_{d|n}\mu(d)$$

$$id=\phi\times 1$$

$$e=\mu\times 1$$

6.5.3 Möbius Inversion Formula

$$id=\phi\times 1$$

$$\phi=id\times\mu$$

$$f=g\times 1$$

$$g=\mu\times f$$