```
//并查集  时间复杂度O(n)
/*并查集一般用于对动态连通性的判断，主要应用于判断两个元素是否同集合，
 *是否连通，间接好友的判断。判断图是否连通，是否有环。
 *并查集分为带权和不带权
 */

//不带权并查集，合并时将序号小的作为fa，大多数情况直接套用
void init(vector<int>&fa, int n) {
    for (int i=0; i<n; ++i)
        fa[i]=i;
}

int findFather(vector<int>&fa, int r) {
    if (fa[r]!=r)
        fa[r]=findFather(fa, fa[r]);
    return fa[r];
}

void Union (vector<int>&fa, int u, int v) {
    int ufa=findFather(fa, u);
    int vfa=findFather(fa, v);
    if (ufa<vfa)
        fa[v]=fa[vfa]=ufa;
    else
        fa[u]=fa[ufa]=vfa;
}

/*带权并查集，合并时要处理秩!
  //摘自经典题目 POJ 1182 食物链
  本题用rank[x]记录x与x的最远的祖先的关系。 这里定义rank[x]=0表示x与
x的祖先是同类。rank[x]==1表示x吃x的祖先。rank[x]==2表示x的祖先吃x; 这
样定义后就与题目中输入数据的D联系起来, (D-1)就可以表示x与y的关系。这
样就可以用向量的形式去推关系的公式了。我们用f(x,father[x])表示rank[x]的值;
*/
int fa[50005]= {0};
int rank[50005]= {0};
int n;

void initial() {
    for(int i=1; i<=n; i++) {
        fa[i]=i;
        rank[i]=0;
    }
}
```

```
int getfather(int x) {
    if(x==fa[x])  return x;
    int oldfa = fa[x];
    fa[x]=getfather(fa[x]);
    rank[x]=(rank[x]+rank[oldfa])%3;  //用向量的形式很快就可以看出来
    return fa[x];
}

void unionset(int r,int x,int y) {
    int fx,fy;
    fx=getfather(x);
    fy=getfather(y);
    if(fx==fy) return;
    fa[fx]=fy;
    rank[fx]=(rank[y]+r-rank[x]+3)%3;
    // 这里同样可以用向量来推公式。另外需要注意的是，这里只更新了fx的rank值，而
fx的儿子的rank值都没有更新会不会有问题。
    //其实不碍事，由于我们每次输入一组数据我们都对x和y进行了getfather的操作
（x>n || y>n ……)的除外。
    //在执行getfather的操作时，在回溯的过程中就会把fx的儿子的rank值都更新了。
    return ;
}

int istrue(int d,int x,int y) {
    int fx,fy,r;
    if(x>n || y>n || ((x==y)&&(d==2)) )
        return 0;
    fx=getfather(x);
    fy=getfather(y);
    if(fx!=fy)  return 1;
    else {
        if(rank[x]==((d-1)+rank[y])%3) return 1;
        // 这个公式可以用向量来推：如果（ f(x,y) + f(y,father[y]) ）% 3 ==
f(x,father[x]) 则是正确的，否则是错的。
        //这个形式可以用向量来表示，就是判断这个向量加法对不对   x--->y + y--->
fx(fy) 是否等于  x--->fx(fy)
        else  return 0;
    }
}
```

```
//树状数组  时间复杂度O(ElogE)
/*树状数组用于查询任意两位间元素和，每次只能修改一个元素的值，代码简洁
 *一般情况下树状数组能解决的问题线段树都能解决，反之不行。
 */
const int N = 500005;
struct Node {
    int val;
    int pos;
};
Node node[N];
int reflect[N], n;
bool cmp(const Node& a, const Node& b) {
    return a.val < b.val;
}

//完全功能模板
//注意c中元素位置从1开始
int c[N];
int lowbit(int x) {
    return x & (-x);
}
void update(int x,int add) {    //一维
    while(x<=n) {   //n为元素个数 ，与MAXN不同，x为位置
        a[x]+=add;
        x+=lowbit(x);
    }
}
int getsum(int x) {
    int sum = 0;
    while (x > 0) {
        sum += c[x];
        x -= lowbit(x);
    }
    return sum;
}

int main() {
    for (int i = 1; i <= n; ++i) c[i] = 0;   //初始化树状数组
    sort(node + 1, node + n + 1, cmp);   //排序
    for (int i = 1; i <= n; ++i) reflect[node[i].pos] = i;   //离散化
    for (int i = 1; i <= n; ++i) {  update(reflect[i],1);
        ans += i - getsum(reflect[i]);  //反面思考，总个数-小于等于的元素个数=
比他大的个数
    }   printf("%lld\n", ans);return 0;}
```

```
void modify(int x,int y,int data) { //二维
    for(int i=x; i<MAXN; i+=lowbit(i))
        for(int j=y; j<MAXN; j+=lowbit(j))
            a[i][j]+=data;
}
int get_sum(int x,int y) {
    int res=0;
    for(int i=x; i>0; i-=lowbit(i))
        for(int j=y; j>0; j-=lowbit(j))
            res+=a[i][j];
    return res;
}
```

```
// Dijkstra，路径的花费不能有负   时间复杂度 O(ElogE)
/*注意具体题目变化，一般需要多加考虑限制条件和其它变量值
    摘自 ZOJ3794 贪心驾驶员
 */

const int maxn = 1500;
const int INF = 0x3f3f3f3f;

struct Edge {
    int from, to, dist;
};

struct HeapNode {
    int d, u;
    bool operator < (const HeapNode & rhs) const {
        return d > rhs.d;
    }
};

struct Dijkstra {
    int n, m;
    vector<Edge> edges;
    vector<int> G[maxn];
    bool done[maxn];    //标记
    int d[maxn];        //花费

void init(int n) {
    this -> n = n;
    for (int i = 0; i < n; i++)
        G[i].clear();
        edges.clear();
    }

void AddEdge(int from, int to, int dist) {
    edges.push_back((Edge) {from, to, dist});
    m = edges.size();
    G[from].push_back(m-1);
}
```

```
void dijkstra(int s) {
    priority_queue<HeapNode> Q;
    memset(d,0x3f,sizeof(d));
    memset(done, 0, sizeof(done));
    d[s] = 0;
    Q.push((HeapNode) {0, s});
    while (!Q.empty()) {
        HeapNode x = Q.top();
        Q.pop();
        int u = x.u;
        if (done[u]) continue;
        done[u] = true;
        for (int i = 0; i < (int)G[u].size(); i++) {
            Edge &e = edges[G[u][i]];
            if (d[e.to] > d[u] + e.dist//&& d[u] + e.dist <= c  //一定注意具体
题目限制 ) {
                d[e.to] = d[u] + e.dist;
                    //if (pp[e.to]) d[e.to] = 0;
                    //p[e.to] = G[u][i];    //路径
                Q.push((HeapNode) {d[e.to], e.to});
            }
        }
    }
} G, H;

int main() {
    H.init(n+1);
    G.init(n+1);
    H.AddEdge(u, v, w);
    G.AddEdge(v, u, w);
    H.dijkstra(1);
    G.dijkstra(n);
}
```

```
// spfa 算法求最短路径，允许负环
/*有两种路，一种走完这条路需要的时间是正的，另一种需要的时间是负的，问有没
 *有这样一条回路，走完整条回路后，需要的时间的和是负的(判负环)
 *判断每个点的入队次数，如果大于N（图中总的点数），就是有负环
 *摘自  POJ 3259  虫洞
 */

#include<iostream>
#include<vector>
#include<queue>
using namespace std;

#define INF 0x3f3f3f3f
struct node {
    int to;
    int len;
};

int F,N,M,W;
vector<vector<node> >graph;
//设置相应全局变量后，完全模板函数
bool spfa(int s) {
    vector<int>dis(N+1,INF);
    dis[s]=0;
    vector<int>count(N+1,0);
    count[s]=1;
    vector<bool>inque(N+1,0);
    queue<int>q;
    q.push(s);

    while(!q.empty()) {
        int t=q.front();
        q.pop();
        inque[t]=false;
        for (int i=0; i<graph[t].size(); ++i) {
        node &nd=graph[t][i];
        if(dis[nd.to]>dis[t]+nd.len) {
            dis[nd.to]=dis[t]+nd.len;
            if (!inque[nd.to]) {
                inque[nd.to]=true;
                q.push(nd.to);
                if(++count[nd.to]>N)
                    return false;
                }
            }
        }
    }
    return true;
}
```

```cpp
int main() {
    cin>>F;
    while(cin>>N>>M>>W) {
        graph.assign(N+1,vector<node>());

        int s,e,t;
        int i;
        for(i=0; i<M; ++i) {
            cin>>s>>e>>t;
            graph[s].push_back({e,t});
            graph[e].push_back({s,t});
        }
        for (i=0; i<W; ++i) {
            cin>>s>>e>>t;
            graph[s].push_back({e,-t});
        }
        if(spfa(1))
            cout<<"NO"<<endl;
        else
            cout<<"YES"<<endl;
    }
    return 0;
}
```

```
//次短路径    算法复杂度O（eloge）
/*次短路是的长度有可能和最短路一样长。
 *求次短路：Dijkstra的dist数组和vis数组再加一维，松弛的时候讨论
 *当前的路小于最短路，或者大于最短路但小于次短路这两种情况
 */

const int maxn = 1000 + 5;
const int INF = 0x3f3f3f3f;

struct Node {
    int v, c, flag; //节点、耗费、最次标记
    Node (int _v = 0, int _c = 0, int _flag = 0) : v(_v), c(_c), flag(_flag) {}
    bool operator < (const Node &rhs) const {
        return c > rhs.c;
    }
};

struct Edge {
    int v, cost;    //节点、耗费
    Edge (int _v = 0, int _cost = 0) : v(_v), cost(_cost) {}
};

vector<edge>E[maxn];
bool vis[maxn][2];  //0,1分别表示最短和次短
int dist[maxn][2];

void Dijkstra(int n, int s) {
    memset(vis, false, sizeof(vis));
    memset(dist,0x3f,sizeof (dist));

    priority_queue<Node>que;
    dist[s][0] = 0;
    que.push(Node(s, 0, 0));
    while (!que.empty()) {
        Node tep = que.top();
        que.pop();
        int u = tep.v;
        int flag = tep.flag;
        if (vis[u][flag])
            continue;
        vis[u][flag] = true;
        for (int i = 0; i < (int)E[u].size(); i++) {
            int v = E[u][i].v;
            int cost = E[u][i].cost;
```

```
            if (!vis[v][0] && dist[v][0] > dist[u][flag] + cost) {
                dist[v][1] = dist[v][0];    //最短
                dist[v][0] = dist[u][flag] + cost;
                que.push(Node(v, dist[v][0], 0));
                que.push(Node(v, dist[v][1], 1));
            } else if (!vis[v][1] && dist[v][1] > dist[u][flag] + cost) {
                dist[v][1] = dist[u][flag] + cost;  //次短
                que.push(Node(v, dist[v][1], 1));
            }
        }
    }
}

void addedge(int u, int v, int w) {
    E[u].push_back(Edge(v, w));
}

int main() {
    int n, m, v, w;
    while (scanf("%d", &n) != EOF) {
        for (int i = 0; i <= n; i++)
            E[i].clear();
        for (int u = 1; u <= n; u++) {
            scanf("%d", &m);
            for (int j = 0; j < m; j++) {
                scanf("%d%d", &v, &w);
                addedge(u, v, w);
            }
        }
        Dijkstra(n, 1);
        printf("%d\n", dist[n][1]);
    }
    return 0;
}
```

```
//最大公共祖先（LCA）   算法复杂度 O（nlogn）
/*给出一棵有边权的树，再加上一条边，Q组询问，求两点之间的最短距离缩短了多少
 */
const int maxn=100005;
const int maxe=100005;
const int maxdep=20;

struct edge {
    int to;
    int w;      //无权树时可省略
    int next;
} e[maxn<<1];

int head[maxn],tot;
int dis[maxn];  //无权树时可省略
int dep[maxn];
int fa[maxn][maxdep];

void init() {
    tot=0;
    memset(head,-1,sizeof (head));
}
void addedge(int u,int v,int w) {
    e[tot].to=v;
    e[tot].w=w; //可省略
    e[tot].next=head[u];
    head[u]=tot++;
}

void dfs(int u,int pre,int d) {
    dep[u]=d;
    fa[u][0]=pre;
    for (int i=1; i<maxdep; ++i)
        fa[u][i]=fa[fa[u][i-1]][i-1];
    for (int i=head[u]; i!=-1; i=e[i].next) {
        int v=e[i].to;
        if (v==pre)
            continue;
        dis[v]=dis[u]+e[i].w;   //可省略
        dfs(v,u,d+1);
    }
}
```

```
/*完全模板函数  返回u，v两个节点的 LCA
 */
int lca(int u,int v) {
    if (dep[u]>dep[v])
        swap(u,v);
    int hu=dep[u],hv=dep[v];
    int tu=u,tv=v;
            //找v节点向前第det祖先
    for (int det=hv-hu,i=0; det; det>>=1,i++)
        if(det&1)
            tv=fa[tv][i];
    if (tu==tv)
        return tu;
            //逐步逼近tv,tu的LCA
    for (int i=maxdep-1; i>=0; --i) {
        if (fa[tu][i]==fa[tv][i])
            continue;
        tu=fa[tu][i];
        tv=fa[tv][i];
    }
    return fa[tu][0];
}

int main() {
    init();addedge(u,v,w);addedge(v,u,w);
    dis[1]=0;
    dfs(1,1,0);
    L1=dis[a]+dis[b]2*dis[lca(a,b)];
    L2=dis[a]+dis[u]2*dis[lca(a,u)]+dis[b]+dis[v]-2*dis[lca(b,v)]+w;
    L3=dis[a]+dis[v]2*dis[lca(a,v)]+dis[b]+dis[u]-2*dis[lca(b,u)]+w;
    printf("%d\n",L1-min(L1,min(L2,L3)));
    return 0;
}
```