snow

考虑一开始共有 $\sum a_i$ 的积雪厚度，每次可以把两层在相邻位置的积雪合并，每层积雪只能与不超过两层且不在同一个结点上的积雪合并（即保证合并结果是一条树上最短路）。

记 f_x 表示在以 x 为根的子树中最多能合并多少次，g_x 表示在以 x 为根的子树中，保证合并次数最多的情况下最多有多少层积雪能够与其父亲合并。

考虑对于结点 u 进行转移。

首先，u 的每个儿子结点 v 有 g_v 层积雪能与 u 合并。容易发现，当 $g_v > a_u$ 时，多出来了 $g_v - a_u$ 次合并机会，显然，我们需要在统计时去掉这些次数。

类似的，u 中的 a_u 层积雪都至多能与 2 层积雪合并，即最多合并 $2a_u$ 次。记总合并次数 $G = \sum\limits_{v\in \text{son}_u} \min(a_u, g_v)$，则算上各个子树内部的合并次数可知

$$f_u = \sum_{v\in \text{son}_u} f_v + \min(2a_u, G).$$

考虑如何计算 g_u。对 G 的范围分类讨论：

- 若 $G \le a_u$，则显然 a_u 层积雪中有 G 层与儿子结点的积雪合并，但是所有 a_u 层积雪都能与父结点合并，故 $g_u = a_u$；
- 若 $G > 2a_u$，则为了尽可能多地合并，我们必须让 a_u 层积雪都与两层儿子结点的积雪合并，使得总共合并 $2a_u$ 次，而每层积雪都不能再与父结点合并，故 $g_u = 0$；
- 若 $a_u < G \le 2a_u$，则为了尽可能多地合并，我们必须让部分积雪与两层儿子结点的积雪合并，另外的积雪与一层儿子结点的积雪合并，则容易求得有 $G - a_u$ 层积雪与两层儿子结点的积雪合并，剩下 $a_u - (G - a_u) = 2a_u - G$ 层积雪与一层儿子结点的积雪合并，这些结点能与父结点的积雪合并，所以 $g_u = 2a_u - G$。

我们不必担心一层积雪会与同一个儿子结点的两层积雪合并，这是因为我们已经忽略了每个儿子结点提供的合并次数大于 a_u 的部分，我们一定能做到每层积雪合并的两层积雪不位于同一个儿子结点。

我们不必担心一个子树中的决策不取最优时整个子树的决策可以更优。因为每棵子树所选取的决策对应的 $f_v + g_v$ 值有一定上限，f_v 减小一定值将导致 g_v 增大相等的值，这不能使得整棵子树的决策更加优秀。

```
#include<bits/stdc++.h>
using namespace std;

int n,a[200001];
long long f[200001],g[200001];

struct Edge{
    int to,nxt;
}edge[400001];
int cntEdge,head[200001];
void addEdge(int u,int v){
    edge[++cntEdge]={v,head[u]},head[u]=cntEdge;
}

void dfs(int id,int father){
    long long tmp=0;
    for(int i=head[id];i;i=edge[i].nxt){
```

```
        if(edge[i].to==father)continue;
        dfs(edge[i].to,id);
        f[id]+=f[edge[i].to];
        tmp+=min(1ll*a[id],g[edge[i].to]);
    }
    f[id]+=min(2ll*a[id],tmp);
    if(tmp<=a[id])g[id]=a[id];
    else if(tmp<=a[id]*2)g[id]=a[id]-(tmp-a[id]);
    else g[id]=0;
}

long long answer;

int main(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
        scanf("%d",a+i),answer+=a[i];
    for(int i=1;i<n;i++){
        int u,v;
        scanf("%d%d",&u,&v);
        addEdge(u,v);
        addEdge(v,u);
    }
    dfs(1,0);
    printf("%lld\n",answer-f[1]);
    return 0;
}
```

plate

特殊情况

为了描述方便，我们称一个点所在的圆指的是其所在的任意一个 **不被其他圆包含** 的圆。

容易想到有一些特殊情况可以直接得出答案：

- 起点坐标等于终点坐标，直接输出 `YES`。
- 两个点有一个不在任意圆内，直接输出 `NO`。
- 不存在一个圆包含两个点且两个点到这个圆的圆心距离相等，直接输出 `YES`。

并且，我们可以忽略所有圆心相同的圆中，半径不是最大的那些圆。

空白环

容易发现，圆中有一些位置无论如何操作，都不能改变到圆心的位置，而这些位置在圆中总是呈现圆环的形状，我们定义这些位置组成的区域为 **空白环**。

形式化地，我们称一个圆 i 上距离半径 $(l,\ r)$ 的区域为空白环，当且仅当以下条件成立：

- 不存在一个圆 $j \neq i$，使得圆 j 中存在一个点到圆 i 的圆心距离在区间 $(l,\ r)$ 中；
- 存在一个圆 $j \neq i$ 包含一个点 P 到圆 i 的圆心距离为 l 或者 $l = 0$；
- 存在一个圆 $j \neq i$ 包含一个点 P 到圆 i 的圆心距离为 r 或者 $r = r_i$。

经过观察，我们容易发现 **空白环** 有一些优美的性质：

- 若点 P 在一个空白环 $(i,\ l,\ r)$ 中，则 P 可能的行动范围为以圆 i 的圆心为圆心，P 到圆心的距离为半径作出的圆上。

- 类似地，P 不在任意空白环中，则 P 经过行动不能进入这个空白环或者穿越这个空白环。
- 对于圆 i 和区间 $[l,\ r]$ 满足 $0 \le l \le r \le r_i$ 且不存在空白环 $(i,\ l',\ r')$ 使得区间 $(l',\ r')$ 与区间 $[l,\ r]$ 相交，则对于任意点 P 与圆 i 圆心距离为 l，点 Q 与圆 i 圆心距离为 r，都可以找到操作方式从 P 移到 Q。

结论与证明

先给出结论：只要起点和终点不能经过 ≤ 1 次到达，且不被任意空白环分开，且所在圆连通，就能找到一种操作。

必要性显然，下证充分性。设起点 S 所在的圆是圆 i，终点 E 所在的圆是圆 j。若 $i = j$，又因为不被空白环分开，则由上文空白环的第三个性质可证。

因为圆连通又不被空白环分开，所以容易找出一条路径上所有点都在圆内但不在空白环内。

由于 i 和 j 是两个不同的圆，所以这条路径肯定经过两个圆的边界 P，Q。由空白环的第三个性质知我们能够找到 $S \to P$ 和 $Q \to E$ 的操作方案。下面考虑 $P \to Q$ 的操作方案。

我们只需要将路径转化为只经过圆边界的路径即可。我们考虑每次都找到一个圆中不在边界上的一段路径，根据空白环的性质三，可以找到另一条在边界上的路径与该路径等效。每次变换后，存在一段路径在圆内部的圆数量严格变小，可以用数学归纳法证明，一定存在这样的操作方案。

实现

首先判断上述所有特殊情况，再判断所在圆是否连通，可以用并查集实现。

再判断两个点是否被空白环分开。只需要枚举圆 i，再枚举圆 $j \ne i$ 就能求出所有不在空白环上的距离区间，将这些端点排序，就能求出所有空白环，并判断是否分开两个点（分开当且仅当一个点在空白环内层圆里，另一个点在空白环外层圆上或外部）。

注意浮点数精度处理。时间复杂度 $\Theta(n^2 \log n)$。

```
#include<bits/stdc++.h>
using namespace std;
const long double eps=1e-10;

int n;

struct Point{
    long double x,y;
    void read(){
        scanf("%Lf%Lf",&x,&y);
    }
}S,E;

struct Plate{
    int x,y,r;
    void read(){
        scanf("%d%d%d",&x,&y,&r);
    }
    Point getPoint(){
        return {(long double)x,(long double)y};
    }
    long double dis(Point A){
        return sqrt((A.x-x)*(A.x-x)+(A.y-y)*(A.y-y));
    }
    bool contain(Point A){
        return dis(A)<=r+eps;
```

```
    }
    bool isCross(Plate A){
        return 1ll*(x-A.x)*(x-A.x)+1ll*(y-A.y)*(y-A.y)<=1ll*(r+A.r)*(r+A.r)+eps;
    }
    bool contain(Plate A){
        return 1ll*(x-A.x)*(x-A.x)+1ll*(y-A.y)*(y-A.y)<=1ll*(r-A.r)*(r-A.r);
    }
}a[2001];

int fa[2001];
int find(int id){
    if(fa[id]==id)return id;
    else return fa[id]=find(fa[id]);
}
void solve(){
    scanf("%d",&n);
    for(int i=1;i<=n;i++)
        a[i].read();
    S.read(),E.read();
    if(abs(S.x-E.x)<eps&&abs(S.y-E.y)<eps)
        { puts("YES"); return; }
    for(int i=1;i<=n;i++)
        if(abs(a[i].dis(S)-a[i].dis(E))<eps
            &&a[i].contain(S))
            { puts("YES"); return; }
    {
        bool flag=false,flag2=false;
        for(int i=1;i<=n;i++){
            if(a[i].contain(S))flag=true;
            if(a[i].contain(E))flag2=true;
        }
        if(!flag||!flag2){ puts("NO"); return; }
    }
    int Sin=0,Ein=0;
    for(int i=1;i<=n;i++){
        fa[i]=i;
        if(a[i].contain(S)&&a[i].r>a[Sin].r)Sin=i;
        if(a[i].contain(E)&&a[i].r>a[Ein].r)Ein=i;
    }
    for(int i=1;i<=n;i++)
        for(int j=i+1;j<=n;j++){
            if(find(i)==find(j))continue;
            if(a[i].isCross(a[j]))fa[find(i)]=find(j);
        }
    if(find(Sin)!=find(Ein))
        { puts("NO"); return; }
    for(int i=1;i<=n;i++){
        vector<pair<long double,int> > c;
        for(int j=1;j<=n;j++){
            if(a[i].x==a[j].x&&a[i].y==a[j].y)continue;
            if(!a[i].isCross(a[j]))continue;
            long double l=a[i].dis(a[j].getPoint())-a[j].r,
                        r=a[i].dis(a[j].getPoint())+a[j].r;
            l=max(l,(long double)0);
            r=min(r,(long double)a[i].r);
            c.push_back({l,0});
            c.push_back({r,1});
        }
```

```
            c.push_back({(long double)a[i].r,0});
            sort(c.begin(),c.end());
            long double lstR=0; int coverd=0;
            for(auto j :c){
                if(j.second==0){
                    if(coverd==0&&j.first-lstR>eps){
                        if(lstR<=a[i].dis(S)+eps&&a[i].dis(S)<=j.first+eps)
                            { puts("NO"); return; }
                        if(lstR<=a[i].dis(E)+eps&&a[i].dis(E)<=j.first+eps)
                            { puts("NO"); return; }
                        if(a[i].dis(S)<=lstR+eps&&j.first<=a[i].dis(E)+eps)
                            { puts("NO"); return; }
                        if(a[i].dis(E)<=lstR+eps&&j.first<=a[i].dis(S)+eps)
                            { puts("NO"); return; }
                    }
                    coverd++;
                }
                else coverd--,lstR=j.first;
            }
        }
        puts("YES");
    }

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

board

当且仅当 n 为偶数时，能够找到合法的方案。

构造

使用数学归纳法，$n=2$ 时，结论平凡。考虑从 $n=k$ 到 $n=k+2$。

先把前 k 项和最后两项变得相同。再按照如下操作

$$
\begin{aligned}
& \underbrace{x,\ x,\ \cdots,\ x}_{k},\ y,\ y \\
\to\ & x,\ x,\ xy,\ x,\ x,\ \cdots,\ x,\ xy,\ y \\
\to\ & x,\ x,\ xy,\ x^2y,\ x,\ x,\ \cdots,\ x,\ x^2y,\ y \\
\to\ & x,\ x,\ xy,\ x^2y,\ x^3y,\ \cdots,\ x^{k-2}y,\ x^{k-2}y,\ y \\
\to\ & x,\ x,\ xy,\ x^2y,\ x^3y,\ \cdots,\ x^{k-2}y,\ x^{k-2}y^2,\ x^{k-2}y^2
\end{aligned}
$$

接着，对于任意 $1\le 2i\le k+2$，将第 i 个数与第 $k+3-i$ 个数进行一次操作。此时所有数都变为 $x^{k-1}y^2$。

证明

下证 n 为奇数时，一定不存在符合题意的构造。

取第 i 个数为第 i 大的素数（防止若干个数之积等于另外若干个数之积，且能保证两数操作后比原数大）。

只需要证明每次操作后，数列中 **最大值** 的数量保持为偶数。

- 若操作后出现新的最大值，则必定由两个数操作而得，最大值有 2 个；
- 若操作后没有出现新的最大值，则要么两个数操作后都变为最大值，最大值数量增加 2，要么操作后没有达到最大值，最大值数量不变。

综上所述，最大值数量总是保持偶数，而 n 为奇数，故一定不存在符合题意的构造。

```
#include<bits/stdc++.h>
using namespace std;

int main(){
    int T; scanf("%d",&T);
    while(T--){
        int n; scanf("%d",&n);
        if(n&1){
            printf("0\n");
            continue;
        }
        printf("1\n");
        int total=1;
        for(int i=4;i<=n;i+=2)
            total+=2+(i-4)+(i>>1);
        printf("%d\n",total);
        printf("1 2\n");
        for(int i=4;i<=n;i+=2){
            printf("%d %d\n",i-1,i);
            for(int j=3;j<=i-2;j++)
                printf("%d %d\n",j,i-1);
            printf("%d %d\n",i-1,i);
            for(int j=1;(j<<1)<=i;j++)
                printf("%d %d\n",j,i+1-j);
        }
    }
    return 0;
}
```

treasure

注意到 $k \leq 16$，容易想到状态压缩动态规划。

考虑在路径上必定经过一些晶石或结界（下称为特殊结点），而在每两个特殊结点之间的路径是朴素的（因为不经过特殊结点），我们可以预处理出任意两个结点之间不经过特殊结点的最短路径。

这时就可以状态压缩了，可以求出手上拥有集合 s 的钥匙时两个特殊结点间的距离。

对任意两个特殊结点都能求出它们之间的最短距离（可以经过特殊结点）。

对任意两个结点，都枚举第一个经过和最后一个经过的特殊结点，即可求出它们的最短路。

```
#include<bits/stdc++.h>
using namespace std;
void updateMin(int &x,int y){x=min(x,y);}

int n,m,k;
int dis_simple[401][401];
int dis_special[1<<16][33][33];
int dis_key[1<<16][17][17];
int dis[17][17];
```

```
int answer[401][401];

int door[17],key[17],requireKey[401];
bool isDoor[401],isKey[401];

bool checkKey(int status,int key){
    return (status|key)==status;
}

int main(){
    scanf("%d%d%d",&n,&m,&k);
    memset(dis_simple,0x3f,sizeof dis_simple);
    for(int i=1;i<=n;i++)
        dis_simple[i][i]=0;
    for(int i=1;i<=m;i++){
        int u,v,w;
        scanf("%d%d%d",&u,&v,&w);
        updateMin(dis_simple[u][v],w);
        updateMin(dis_simple[v][u],w);
    }
    for(int i=1;i<=k;i++){
        scanf("%d%d",door+i,key+i);
        isDoor[door[i]]=isKey[key[i]]=true;
        requireKey[door[i]]|=1<<i-1;
    }
    for(int k=1;k<=n;k++){
        if(isDoor[k])continue;
        for(int i=1;i<=n;i++)
            for(int j=1;j<=n;j++)
                updateMin(dis_simple[i][j],dis_simple[i][k]+dis_simple[k][j]);
    }

    memset(dis_special,0x3f,sizeof dis_special);
    for(int i=1;i<=k*2;i++){
        int x=i<=k?door[i]:key[i-k];
        for(int j=1;j<=k*2;j++){
            int y=j<=k?door[j]:key[j-k];
            dis_special[0][i][j]=dis_simple[x][y];
        }
    }
    for(int status=1;status<(1<<k);status++){
        int i=k; while(!(status&(1<<i-1)))i--;
        for(int x=1;x<=k*2;x++)
            for(int y=1;y<=k*2;y++)
                dis_special[status][x][y]=dis_special[status^(1<<i-1)][x][y];
        if(!checkKey(status,requireKey[door[i]]))continue;
        for(int x=1;x<=k*2;x++){
            if(dis_special[status][i][x]==0x3f3f3f3f)continue;
            for(int y=1;y<=k*2;y++)
                updateMin(dis_special[status][x][y],dis_special[status][x]
[i]+dis_special[status][i][y]);
        }
    }

    memset(dis_key,0x3f,sizeof dis_key);
    for(int i=1;i<=k;i++)
        if(checkKey(0,requireKey[key[i]]))
            dis_key[1<<i-1][i][i]=0;
```

```
    for(int status=1;status<(1<<k);status++){
        vector<int> own,notown;
        for(int i=1;i<=k;i++)
            if(status&(1<<i-1))own.push_back(i);
            else notown.push_back(i);
        for(int x :own)for(int y :own){
            if(dis_key[status][x][y]==0x3f3f3f3f)continue;
            for(int z :notown)if(checkKey(status,requireKey[key[z]]))
                updateMin(dis_key[status^(1<<z-1)][x][z],dis_key[status][x]
[y]+dis_special[status][k+y][k+z]);
        }
    }

    memset(dis,0x3f,sizeof dis);
    for(int x=1;x<=k;x++)
        for(int status=0;status<(1<<k);status++){
            if(!(status&(1<<x-1)))continue;
            for(int y=1;y<=k;y++){
                if(!checkKey(status,requireKey[door[y]]))continue;
                for(int i=1;i<=k;i++)if(status&(1<<i-1))
                    updateMin(dis[x][y],dis_key[status][x]
[i]+dis_special[status][i+k][y]);
            }
        }

    memset(answer,0x3f,sizeof answer);
    for(int x=1;x<=n;x++){
        if(isDoor[x])continue;
        for(int y=1;y<=n;y++){
            if(!isDoor[x]&&!isDoor[y])answer[x][y]=dis_simple[x][y];
            for(int u=1;u<=k;u++)for(int v=1;v<=k;v++)
                if(checkKey(0,requireKey[key[u]])&&(door[v]==y||!isDoor[y]))
                    updateMin(answer[x][y],(int)min(1ll*dis_simple[x]
[key[u]]+dis[u][v]+dis_simple[y][door[v]],1ll*0x3f3f3f3f));
        }
    }

    int q; scanf("%d",&q);
    while(q--){
        int s,t;
        scanf("%d%d",&s,&t);
        printf("%d\n",answer[s][t]==0x3f3f3f3f?-1:answer[s][t]);
    }
    return 0;
}
```