T1 喷泉

根据初中数学知识，可以知道，一个定点到圆上点的最大（小）距离等于其到圆心的距离加（减）半径的长度，而一个定点到一条线段的最大距离显然是到线段两端之一，最小距离是垂线段长度（题目保证垂足在线段上）。

于是输出圆心到线段距离减半径，到两端点的最大距离加半径即可。

别溢出了。

具体证明问初中数学老师，剩下的就是解析几何计算了。

```
#include <bits/stdc++.h>
using namespace std;
signed main(){
    #define int long long
    int t, x1, y1, x2, y2, x3, y3, r;
    cin >> t;
    while(t--){
        cin >> x1 >> y1 >> x2 >> y2 >> x3 >> y3 >> r;
        int A=y1-y2,B=x2-x1;
        int C=-(A*x1+B*y1);
        long double minm=abs(A*x3+B*y3+C)/sqrt((long double)A*A+B*B)-r;
        long double maxm=sqrt((long double)(x3-x1)*(x3-x1)+(y3-y1)*(y3-y1));
        maxm=max(maxm,sqrt((long double)(x3-x2)*(x3-x2)+(y3-y2)*(y3-y2)))+r;
        // printf("%.2lf %.2lf\n",(double)minm,(double)maxm);
        printf("%.2Lf %.2Lf\n",minm,maxm);
    }
    return 0;
}
```

T2 红绿灯

其实本题暴力就大致能通过了，只不过有一些大优化。

题目简述：维护一个长度为 n 的序列，初始为 1 至 n。之后有 m 次操作，每次输入一个整数 a，将序列中每一个元素 x 变成 $\lceil \frac{x}{a} \rceil \times a$，问最终序列。

可以发现在一系列操作之后整个序列中有很多相同的元素，那么我们去重一下，并且记录一下每一个数字出现在序列中的哪些位置，显然是一个区间。

这样，我们记录的数字个数就大大减少，可以拿到约 $70pts$。

可以发现在 $2,3$ 循环的数据中跑的很慢，而我们输出序列发现在之后的操作中，序列甚至一点都没变。

这是因为序列中的每一个元素**都整除了** a，所以根本不会变。

这样我们可以在每一次操作后，求一下整个序列所有元素的 gcd，一旦操作的 a 是 gcd 的约数，直接跳过不用修改。

由于本题数据较为随机，所以可以通过。

```
#include<bits/stdc++.h>
#define int long long
```

```cpp
using namespace std;
int b[300001][2];
int a[300001][2],n,m;
int gcd(int a,int b){
    return b?gcd(b,a%b):a;
}
signed main(){
    cin >> n >> m;
    a[0][0]=n;
    for(int i=1;i<=n;i++)a[i][0]=b[i][0]=i;
    int G=1, x;
    int sum=0;
    for(int i=1;i<=m;i++){
        cin >> x;
        if(G%x==0)continue;
        int now=i&1,las=now^1;
        sum+=a[0][las];
        a[0][now]=0;
        for(int j=1;j<=a[0][las];j++)a[j][las]=((a[j][las]-1)/x+1)*x;
        for(int j=1;j<=a[0][las];j++){
            if(j==1||a[j][las]!=a[j-1][las])
                a[++a[0][now]][now]=a[j][las];
            b[a[0][now]][now]=b[j][las];
        }
        G=a[1][now];
        for(int j=1;j<=a[0][now];j++)G=gcd(G,a[j][now]);
    }
    sum+=a[0][m&1];
    int j=0;
    for(int i=1;i<=n;i++){
        while(b[j][m&1]<i)j++;
        cout << a[j][m&1] << " ";
    }
}
```

T3 子集

注意到一个结论：

$$F_k(S)=\sum_{T\subseteq S}F_0(T)\cdot k^{|S|-|T|} \tag{1}$$

其中 $k>0$。

证明：$k=1$ 时显然成立。

若对任意 $k\le r$ 均有 (1) 成立，那么当 $k=r+1$ 时：

$$\begin{aligned}F_{r+1}(S)&=\sum_{T\subseteq S}F_r(T)=\sum_{T\subseteq S}\sum_{C\subseteq T}F_0(C)\cdot r^{|T|-|S|}\\&=\sum_{C\subseteq S}F_0(C)\cdot\sum_{i=0}^{|S|-|C|}\binom{|S|-|C|}{i}r^i\\&=\sum_{C\subseteq S}F_0(C)\cdot(r+1)^{|S|-|C|}\end{aligned}$$

因此 (1) 同样成立。从而，对任意正整数 k 均有 (1) 成立。

因此我们现在要做的就是：对每个和为 M 的子集 $T \subseteq S$，计算其贡献 $k^{|S|-|T|}$，最后加到一起。

可以考虑一个显然的 dp：设 $F(i,j,w)$ 表示前 i 个数中选了 j 个，其和为 w 的方案数。

枚举最后一个数选不选，不难得到转移方程

$$F(i,j,w)=F(i-1,j,w)+F(i-1,j-1,w-a_i)$$

那么答案就是

$$\sum_{j=0}^{n} F(n,j,M)\cdot k^{n-j}$$

时间复杂度为 $O(n^2M)$，可以获得 60 分。

考虑优化一下状态设计：设 $F(i,w)$ 表示 $\{a_1,a_2,\cdots,a_i\}$ 的所有和为 w 的子集的贡献之和。

同样考虑最后一个数选不选：

- 如果没有选 a_i，那么相当于式 (1) 中的 $|S|$ 加了 1，因此 k 的指数也会 $+1$；
- 反之，则 $|T|$ 也会 $+1$，那么原封不动加进来就行了。

因此，我们得到

$$F(i,w)=F(i-1,w)\cdot k+F(i-1,w-a_i)$$

时间复杂度 $O(nM)$，可以通过。

```
#include<bits/stdc++.h>
#define int long long
using namespace std;
const int MN=5005;
const int mod=1e9+7;
int a[MN],dp[MN][MN],n,M,k;
void solve(){
    memset(dp,0,sizeof(dp));
    cin >> n >> M >> k;
    for(int i=1;i<=n;i++) cin >> a[i];
    dp[0][0]=1;
    for(int i=1;i<=n;i++){
        for(int j=0;j<=M;j++){
            dp[i][j]=dp[i-1][j]*k%mod;
            if(j>=a[i])dp[i][j]=(dp[i][j]+dp[i-1][j-a[i]])%mod;
        }
    }
    cout<<dp[n][M]%mod<<endl;
}
signed main(){
    int tt;
    cin >> tt;
    while(tt--)solve();
}
```

T4 佛怒火莲

测试点1,2：考虑直接暴力选最多5个数字，dfs的复杂度是$C_n^5<10^8$的，所以直接dfs就行。

测试点3~6：对所有火焰按照b排序，用$dp_{i,sta}$表示考虑了前i个火焰，且我选了第i个火焰，当前选了sta这个颜色集合的火焰时，我的最大间隔是多少，转移是$O(n)$的。总复杂度是$O(n^2 \times 2^k)$。

测试点11~14：只需要选三种火焰，我们可以枚举中间的火焰是谁，然后，有一个显而易见的结论是，如果中间的火焰是i，那么选定的另外两朵火焰，一定在i左边最远的两种不同色火焰中选一朵，i右边最远的两种不同色火焰中选一朵，直接暴力预处理两遍即可，除了排序以外，复杂度是$O(n)$的。

测试点15~17：考虑优化3~6的dp，我们二分答案，然后只需要$check$可行性了就。$dp_{i,sta} = 0/1$表示我们考虑了前i个火焰，并且选了i，当前选的状态是sta的情况下，是否可行，这样子就有转移：

$dp_{i,sta} = \max(dp_{j,pre})$，其中$pre$是$sta$二进制下去掉第$a_i$位后的值，$j < i$且$b[i] - b[j] >= limit$，这里的$limit$是我们二分的答案。

这个dp显然可以维护前缀的最大值，来使得转移变成$O(1)$。

这样总复杂度$O(log(10^6) \times n \times 2^k)$，可以轻松通过。

测试点7~10：我们考虑把所有不同种类的b随机映射到$1 - k$这k种颜色去，然后去做上面的dp，那么，答案对应的k个火焰，有$\frac{k!}{k^k}$的概率恰好分配到了k种不同的颜色，也就是说，我们的dp即便在$k = 5$的情况下，也有$\frac{k!}{k^k} = 0.0384$的概率获得正确的结果。

那么我们就把随机分配颜色这个事情模拟200次，就可以有$1 - (1 - 0.0384)^{200} > 0.9996$的正确率了。

时间复杂度是$O(200 \times log(10^6) \times n \times 2^k)$。

> 考虑到能想到这一步的人，应该都会$O(n \times 2^k)$的dp，所以没留暴力dp的分数

测试点1~20：

我们考虑进一步优化我们的dp。

我们的dp本质上最多压了$2^5 = 32$个01，那么我们直接用一个$uint$把他存下来就可以了。

用dp_i表示考虑了前i个位置以后，所有二进制状态$00000 = 0$到$11111 = 31$分别行不行，我们把它存到$uint$的每一位上。

然后转移方程就是：

$$dp_i = dp_{i-1} | ((tp_j \& ok[col_i]) << (1 << col_i))$$

$ok[i]$表示对于颜色i，哪些二进制状态里面不含有第i种颜色。

此处比较绕，建议自己推一下细节，注意$1 << 31$是会爆int的。

时间复杂度变成$O(200 \times log(10^6) \times n)$。

> 实际上测试点11~14直接少随一些次数，直接暴力dp也是可以通过的。

> 实际上还可以暴力dp的时候，随机次数按数据范围来定，小数据随机200次，大数据就随机50~60次，大概率能拿到90及以上。（这个分数要看评测机的脸色）

```
#include <bits/stdc++.h>
#define ll long long
#define maxn 10005
using namespace std;
unsigned int dp[maxn];
unsigned int tmp;
pair<int,int> a[maxn];
int n,k,tp;
int col[maxn],best;
```

```
int ok[maxn];
void init()
{
    memset(ok,0,sizeof(ok));
    for (int i=0;i<k;i++)
    {
        for (int sta=0;sta<(1<<k);sta++) //for每一个二进制位
        {
            if ((sta&((unsigned int)1<<i))==0) //如果sta里面不包含二进制下第i位
                ok[i]|=(((unsigned int)1<<sta));
        }
    }
}

void add(int i) {tmp|=dp[i];}
int check(int lim)
{
    memset(dp,0,sizeof(dp));
    dp[0]=1; tmp=1;
    int pos=0;
    for (int i=1;i<=n;i++)
    {
        while (pos+1<i && a[i].first-a[pos+1].first>=lim) add(++pos);
        int se=col[a[i].second];

        dp[i]=dp[i-1]|((tmp&ok[se])<<((unsigned int)1<<se));
    }
    while (pos<n) add(++pos);
    return tmp>>(((unsigned int)1<<k)-1);
}

void work()
{
    for (int i=1;i<=n;i++) col[i]=rand()%k; //给个0~k-1之间的颜色
    int now=-1;
    for (int step=(1<<20);step>=1;step=step>>1)
    if (check(now+step)==1)
        now+=step;
    best=max(best,now);
    return;
}

int main()
{
    srand(time(0));
    int T;
    cin>>T;
    while (T--)
    {
        best=0;
        cin>>n>>k>>tp;
        init();
        for (int i=1;i<=n;i++) cin>>a[i].second>>a[i].first;
        sort(a+1,a+n+1);
        for (int turn=1;turn<=200;turn++) work();
```

```
        cout<<best<<endl;
    }
}
```