Other

输入输出挂

```
template <class T>
inline void In(T &ret) {
    char c;
    ret = 0;
    while ((c = getchar()) < '0' || c > '9');
    while (c >= '0' && c <= '9') {
        ret = ret * 10 + (c - '0'), c = getchar();
    }
}
void Out(LL a) {
    if (a >= 10) Out(a / 10);
    putchar(a % 10 + '0');
}


const int BUFFER_MAX_SIZE = 100000;
struct Quick_In {
    char buf[BUFFER_MAX_SIZE], *ps = buf, *pe = buf + 1;
    inline void InNext() {
        if (++ps == pe)
            pe = (ps = buf) + fread(buf, sizeof(char), sizeof(buf) / sizeof(char), stdin);
    }
    template<class T>
    inline bool operator()(T &number) {
        number = 0;
        T f = 1;
        bool vis_point = 0;
        if (ps == pe) return false; //EOF
        do {
            InNext();
            if ('-' == *ps) f = -1;
        } while (ps != pe && !isdigit(*ps));
        if (ps == pe) return false; //EOF
        do {
            if((*ps) == '.') vis_point = 1;
```

```
            else {
                number = number * 10 + *ps - 48;
                if(vis_point) f *= 0.1;
            }
            InNext();
        } while (ps != pe && (isdigit(*ps) || (*ps) == '.'));
        number *= f;
        return true;
    }
} In;
struct Quick_Out {
    char buf[BUFFER_MAX_SIZE], *ps = buf, *pe = buf + BUFFER_MAX_SIZE;
    char tmp[100];
    double dx[15] = {5e-1,5e-2,5e-3,5e-4,5e-5,5e-6,5e-7,5e-8,5e-9,5e-10,5e-11,5e-12,5e-13,5e-14,5e-15};
    inline void write() {
        fwrite(buf, sizeof(char), ps - buf, stdout);
        ps = buf;
    }
    inline void oc(char c) {
        *(ps++) = c;
        if (ps == pe) write();
    }
    inline void os(char *s) {
        for (int i = 0; s[i]; ++i) oc(s[i]);
    }
    template<class T>
    inline void oi(T x, char bc = '\n') {
        if (x < 0) oc('-'), x = -x;
        int len = 0;
        if (!x) tmp[len++] = '0';
        while (x) tmp[len++] = x % 10 + '0', x /= 10;
        while (len)  oc(tmp[--len]);
        oc(bc);
    }
    void of(double x, int fix = 8, char bc = '\n') {
        x += dx[fix];
        if (x < 0) oc('-'), x = -x;
        oi((LL)x, '.');
        x -= (LL)x;
        while (fix--) {
            x *= 10;
            oc('0' + (int)x);
            x -= (int)x;
        }
        oc(bc);
    }
    ~Quick_Out() {
        write();
    }
} Out;
```

预处理优化

// 除了高位数组优化,取模优化外的优化……
// xjb优化,似乎可以快一倍……暴力必备神器

```
#pragma GCC optimize("Ofast,unroll-loops,no-stack-protector,unsafe-math-optimizations")
#pragma GCC target("avx")
```

流水线作业的贪心

定义完成一个零件需要很多步，每一步只能在上一步完成后才能进行，并且每一步只能同时处理一个零件。也就是说，在第i步的当前零件完成后，扔进第$i+1$步的队列中，然后拿出第i步的队顶零件处理。求总加工时间。
定义零件A, B，当且仅当**考虑仅有这两个零件，先完成A后完成B的用时小于先完成B后完成A的用时**时$A，然后依此排序，按顺序贪心计算时间即可。

DP优化

注意DP方程的后效性

单调队列优化：
有些DP方程可以转化成

```
DP[i]=f[j]+x[i]
```

的形式，其中f[j]中保存了只与j相关的量。
这样的DP方程我们可以用单调队列进行优化

斜率优化：
对于这样的一类DP方程

```
f[i]=min{a[i]*x[j]+b[j]}
```

a[i]是和i有关的函数，x[j],b[j]是和j有关的函数或常数
我们可以把它改写成这个样子

```
-a[i]*x[j]+f[i]=b[j]
```

把-a[i]看做斜率，f[i]看做截距，每一个决策相当于平面上一个点，最优决策显然在平面点集的凸包上，要求f[i]的最小值，就相当于将一条斜率为-a[i]的直线不断向上平移，碰到的第一个点就是截距最小的点，也就是f[i]的最优决策，如果横坐标和斜率均单调，我们就可以维护一个单调队列（斜率是单调的，即最优解也是单调的（类似于决策单调）），队首指针不断往后走，每个点只会访问一次，复杂度为O(n)
如果均不单调，有时我们可以排序使得其中一个单调
或者维护一个单调的斜率，对每一个值在上面二分答案
另：如果方程是

```
f[i]=min{a[i]*x[j]+b[i]*y[j]}
```

的话，可以将其变为

```
f[i]=min(a[i]/b[i]*x[j]+y[j])*b[i]
```

g[i,j]表示两点构成的斜率表达式
1，用一个单调队列来维护解集。
2，假设队列中从头到尾已经有元素a b c。那么当d要入队的时候，我们维护队列的上凸性质，即如果g[d,c]<g[c,b]，那么就将c点删除。直到找到g[d,x]>=g[x,y]为止，并将d点加入在该位置中。
3，求解时候，从队头开始，如果已有元素a b c，当i点要求解时，如果g[b,a]<sum[i]，那么说明b点比a点更优，a点可以排除，于是a出队。最后dp[i]=getDp(q[head])。

四边形不等式优化：
最有代价用$d[i,j]$表示 $d[i,j]=min\{d[i,k-1]+d[k+1,j]\}+w[i,j]$
(一维)$f[j]=min(f[i]+w[i,j])$
四边形不等式 $w[a,c]+w[b,d]<=w[b,c]+w[a,d](a 就称其满足凸四边形不等式
决策单调性 $w[i,j]<=w[i',j'](i,j$属于$i',j')$ 既 $i'<=i
//一般需证明 $w[i,j]+w[i+1,j+1]<=w[i+1,j]+w[i,j+1]$
//max时不等号取反……

于是有以下三个定理
定理一： 如果w同时满足四边形不等式 和 决策单调性 ,则d也满足四边形不等式
定理二：当定理一的条件满足时，让d[i,j]取最小值的k为K[i,j]，则K[i,j-1]<=K[i,j]<=K[i+1,j]
定理三：w为凸当且仅当w[i,j]+w[i+1,j+1]<=w[i+1,j]+w[i,j+1]
由定理三知 判断w是否为凸即判断 w[i,j+1]-w[i,j]的值随着i的增加是否递减
于是求K值的时候K[i,j]只和K[i+1,j] 和 K[i,j-1]有关，所以 可以以i-j递增为顺序递推各个状态值最终求得结果

常见方式：
s为决策区间
dp[i][j] = min{dp[i][k] + dp[k + 1][j] + w[i][j] }, i <= k <= j - 1
s[i][j - 1] <= s[i][j] <= s[i + 1][j] //s[i][i] = i;

dp[i][j] = min{dp[i - 1][k] + w[k + 1][j]}, i - 1 <= k <= j - 1
s[i - 1][j] <= s[i][j] <= s[i][j + 1] //s[i][n + 1] = n;

dp[i][j] = min{dp[k][j - 1] + w[k + 1][j]}, j - 1 <= k <= i - 1
s[i][j - 1] <= s[i][j] <= s[i + 1][j] //s[n + 1][i] = n;

动态维护凸壳

```
//使用多个时HullDynamic，调用指针维护
const LL is_query = - (1LL << 62); // 询问标记
struct Line {
    LL k, b; // kx + b
    mutable function<const Line*()> succ;
    bool operator < (const Line &rhs) const {
        if (rhs.b != is_query) return k < rhs.k;
        const Line* s = succ();
        if (!s) return 0;
        LL x = rhs.k;
        return 1.0L * b - s->b < 1.0L * (s->k - k) * x;
    }
};
struct HullDynamic : public multiset<Line> { // 上凸壳求最大值(维护下凸壳的话插入为insert(-k,-b),询问为-eval(x))
    bool bad(iterator y) {
        auto z = next(y);
        if (y == begin()) {
            if (z == end()) return 0;
            return y->k == z->k && y->b <= z->b;
        }
        auto x = prev(y);
        if (z == end()) return y->k == x->k && y->b <= x->b;
        return 1.0L * (x->b - y->b) * (z->k - y->k) >= 1.0L * (y->b - z->b) * (y->k - x->k);
    }
    void insert_line(LL k, LL b) {
        auto y = insert({k, b});
        y->succ = [=] { return next(y) == end() ? 0 : &*next(y); };
        if (bad(y)) { erase(y); return; }
        while (next(y) != end() && bad(next(y))) erase(next(y));
        while (y != begin() && bad(prev(y))) erase(prev(y));
    }
    LL eval(LL x) { // 询问x处的值
        Line u = {x, is_query};
        auto l = *lower_bound(u);
        return l.k * x + l.b;
    }
};
```

扩栈

```
#pragma comment(linker, "/STACK:10240000,10240000")

//64-bit
    int size = 1 << 20;//256M
    char *p = (char *)malloc(size) + size;
    __asm__("movq %0, %%rsp\n" :: "r"(p));
//32-bit
    int size = 1 << 20;//256M
    char *p = (char *)malloc(size) + size;
    __asm__("movl %0, %%esp\n" :: "r"(p));
```

Java

格式模板

```
import java.io.*;
import java.util.*;
import java.math.*;

public class Main {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader cin = new InputReader(inputStream);
        PrintWriter cout = new PrintWriter(outputStream);
        int T = cin.nextInt();
        for (int i = 1; i <= T; i++) {
            Task ans = new Task();
            ans.solve(i,cin, cout);
        }
        cout.close();
    }
    static class Task {
        public void solve(int id,InputReader in, PrintWriter out) {

        }
    }

    static class InputReader {
        public BufferedReader reader;
        public StringTokenizer tokenizer;

        public InputReader(InputStream stream) {
            reader = new BufferedReader(new InputStreamReader(stream), 32768);
            tokenizer = null;
        }

        public String next() {
            while (tokenizer == null || !tokenizer.hasMoreTokens()) {
                try {
                    tokenizer = new StringTokenizer(reader.readLine());
                } catch (IOException e) {
                    throw new RuntimeException(e);
                }
            }
            return tokenizer.nextToken();
        }

        public int nextInt() {
            return Integer.parseInt(next());
        }
        public double nextDouble() {
            return Double.parseDouble(next());
        }
    }
}
```

凸多边形面积并

```
//需要头文件
import java.awt.geom.Area;
import java.awt.geom.Path2D;
import java.awt.geom.PathIterator;
```

```
    private static double calcArea(Area area) {
        double ret = 0;
        PathIterator it = area.getPathIterator(null);//获得该图形的边界
        double[] buffer = new double[6], last = new double[2], first = null;
        while (!it.isDone()) {
            switch (it.currentSegment(buffer)) {
                case PathIterator.SEG_MOVETO:
                case PathIterator.SEG_LINETO:
                    if (first == null) {
                        first = new double[2];
                        first[0] = buffer[0]; first[1] = buffer[1];
                    } else {
                        ret += last[0] * buffer[1] - last[1] * buffer[0];
                    }
                    last[0] = buffer[0]; last[1] = buffer[1];
                    break;
                case PathIterator.SEG_CLOSE:
                    ret += last[0] * first[1] - last[1] * first[0];
                    first = null;
                    break;
            }
            it.next();
        }
        return ret;
    }


    int n = in.nextInt();//多边形个数
    Area totalArea = new Area();
    double total = 0.0;
    for(int i = 0;i < n; i++) {
        Path2D.Double pd = new Path2D.Double();
        double a = in.nextDouble(),b = in.nextDouble();
        pd.moveTo(a,b);
        for(int j = 0;j < 3; j++) {//读入第i个多边形,这里是四边形
            a = in.nextDouble();
            b = in.nextDouble();
            pd.lineTo(a,b);
        }
        pd.closePath();
        Area tmpArea = new Area(pd);
        totalArea.add(tmpArea);//加入总图形
        total += calcArea(tmpArea);//获得第i个多边形的面积
    }
```

排序重载

```
//创建一个该类的List，例如下面类为Interger时
List<Integer> h = new ArrayList<Integer>();
//List 赋值
for(int i = 0;i < n; i++) h.add(i);
//排序重载 相等返回0，小于返回-1，大于返回1
Collections.sort(h,new Comparator<Integer>() {
    public int compare(Integer a,Integer b){
    if(相等) return 0;
    else if(小于) return -1;
    else return 1;
}
});
```

文件读写

```
public static void main(String[] args) throws FileNotFoundException {
        Scanner in = new Scanner(new File("xxx.in"));
        in.close();

        PrintWriter out = new PrintWriter(new File("xxx.out"));
        out.println(xxx);
        out.close();
}
```

保留位数小数输出

```
    public static String doudou(double value) {
         BigDecimal bd = new BigDecimal(value);
         bd = bd.setScale(6, RoundingMode.HALF_UP);//保留6位
         return bd.toString();
        }
```

JAVA类的函数

Integer类

a.bitCount() 返回a的二进制中1的个数
Integer.toBinaryString(x) 返回x的二进制形式的字符串
Integer.toHexString(x) 返回16进制形式的字符串
Integer.toOctalString(x) 返回8进制形式的字符串
Integer.toString() 数字转字符串
Integer.valueOf(s) 字符串s转为数字

String类

s.charAt(a) 返回下标为a的字符
s.compareTo(ss) 比较字典序，返回整数，0相等;大于0表示s比ss大;小于0表示s比ss小
s.compareToIgnoreCase(ss) 不区分大小写比较字典序
s.concat(ss) 返回 将ss连接到s的结尾后的新字符串
s.endsWith(ss) 判断ss是不是s的后缀
s.startsWith(ss) 判断ss是不是s的前缀
s.equals(ss) 判断s与ss是否相等
s.equalsIgnoreCase(ss) 不区分大小写判断s与ss是否相等
s.length() 返回s的长度
s.replace(a,b) 返回一个字符串，把s中所有的a字符替换为b
s.substring(a,b) 返回s的下标从a到b - 1的子串
s.toLowerCase() 返回将s中所有字符都替换成小写后的字符串
s.toUpperCase() 返回将s中所有字符都替换成大写后的字符串

StringBuffer类，性能比String类高

StringBuffer s = new StringBuffer(u); 构造方式，u可以为空，string，int，double，boolean，StringBuffer各种
s.append(u) 在s的结尾添加u，u可以为string，int，double，boolean，StringBuffer各种
s.insert(a,u) 在下标为a处插入u，u可以为string，int，double，boolean各种
s.charAt(a) 返回下标为a的字符
s.delete(a,b) 删除s中下标为a到b-1的字符
s.deleteCharAt(a) 删除s中下标为a的字符
s.length() 返回s的长度
s.reverse() 反转s
s.serCharAt(a,u) 将下标为a的字符置为字符u
s.setLength(a) 将s的长度置为a，即下标大于a的部分删去，不足补空格
s.substring(a,b) 返回s的下标从a到b - 1的子串
s.toString() 返回把s转换为String类

Random类，随机数 java.util.Random

Random u = new Random(); 构造一个以系统时间为种子的类
u.nextInt() 返回一个int范围(存在负数)内的伪随机数
u.nextInt(a) 返回一个[0,a)的伪随机数
u.nextDouble() 返回一个[0,1)之间的随机浮点数

BigInteger类

常数: BigInteger.ZERO 0; BigInteger.ONE 1; BigInteger.TEN 10
a.abs() 返回a的绝对值
a.and(b) 返回a & b
a.not() 返回 ~a
a.or(b) 返回a | b
a.xor(b) 返回 a ^ b
a.andNot(b) 返回a & (~b)
a.bitCount() 返回a的二进制中1的个数
a.bitLength() 返回a的二进制位数
a.setBit(n) 把a在二进制下第n位(从0开始)置1
a.clearBit() 把a在二进制下第n位(从0开始)置0
a.flipBit(n) 把a在二进制下第n位(从0开始)取反
a.testBit(n) 返回a在二进制下第n位是否为1
a.shiftLeft(n) 返回 a<<n a.shiftRight(n) 返回 a>>n
a.getLowestSetBit() 返回a在二进制下最低的为1的位(从0开始)
a.compareTo(b) 比较a和b的大小，返回0相等，正a比b大，负a比b小
a.add(b) 返回a+b
a.divide(b) 返回a/b
a.mod(b) 返回a%b
a.modInverse(b) 返回a模b下的逆元
a.modPow(b,m) 返回a^b%m
a.pow(b) 返回a^b
a.subtract(b) 返回 a - b
a.multiply(b) 返回a * b
a.doubleValue() 返回a转换成double后的值
a.intValue() 返回a转换成int后的值
a.longValue() 返回a转换成long后的值
a.equals(b) 判断a和b是否相等
a.max(b) 返回max(a,b)
a.min(b) 返回min(a,b)
a.gcd(b) 返回gcd(a,b)
a.hashCode() 返回a的hash值
a.isProbablePrime(int c) 如果a是质数的概率大于1-1/(2c)，返回true;否则返回false(c越大越精确)
a.nextProbablePrime() 返回第一个大于a的可能的质数
a.toString() 返回a转换成的字符串
BigInteger.valueOf(long a) 把a转换成大数

BigDecimal类

a.abs() 返回a的绝对值
a.add(b) 返回a+b
a.compareTo(b) 比较a和b的大小，返回0相等，正a比b大，负a比b小
a.divide(b) 返回a/b
a.divide(b,100,BigDecimal.ROUND_HALF_UP) a/b 保留100位小数
a.multiply(b) 返回a * b
a.subtract(b) 返回 a - b
a.hashCode() 返回a的hash值
a.doubleValue() 返回a转换成double后的值
a.intValue() 返回a转换成int后的值
a.longValue() 返回a转换成long后的值
a.max(b) 返回max(a,b)
a.min(b) 返回min(a,b)
a.movePointLeft(n) 返回a的小数点左移n位后的数
a.movePointRight(n) 返回a的小数点右移n位后的数
a.pow(int b) 返回a^b
a.precision() 返回a的精度
a.toBigInteger()

BigDecimal.valueOf(a) 把a转换成高精度浮点
a.toPlainString() 返回不带指数的字符串
a.toString() 返回字符串，可能是指数形式
a.stripTrailingZeros() 去除a中小数点后末尾的0(前导0转换时自动去除)

List类

List<Integer> f = new ArrayList<Integer>();
f.add(x) 在f的末尾添加x
f.add(i,x) 在下标为i的位置插入x
f.clear() 清空f
f.contains(x) 判断f中是否存在x
f.get(i) 返回f中下标为i的元素
f.isEmpty() 判断f是否为空
f.remove(i) 移除f中下标为i的数
f.set(i,x) 把下标为i的元素替换为x
f.size() 返回f的大小
f.toArray() 转换为数组

Set && Map && HashSet && HashMap

(set独有)add(),toArray()
clear(),contains(),isEmpty(),remove(),size()
(map独有) get(key),put(kay,value),containsKey(),containsValue()

Arrays类

Arrays.binarySearch(f,x) 在f中二分x的下标，没有返回负数
Arrays.binarySearch(f,l,r,x) 在f中下标为[l,r)的区间二分x在f的下标，没有返回负数
Arrays.copyOf(f,n) 返回f中前n个元素组成的数组，不足补0
Arrays.copyOfRange(f,l,r) 返回f中[l,r)区间的元素
Arrays.fill(f,l,r,x) 把f中[l,r)区间的元素置为x
Arrays.fill(f,x) 把f所有元素置为x
Arrays.sort(f) 把f排序
Arrays.sort(f,l,r) 把f中[l,r)区间的元素排序
Arrays.toString(f) 返回f转换成的字符串

Scanner类

.hasNext() 还有输入返回true
.hasNextBigInteger()
.hasNextBigDecimal()
.hasNextInt()
.hasNextLine() 是否还有一行
.next() 读取字符串
.nextLine() 读取一行

```
题意: p(x) = \sum x^(p[i]) , q(x) = sum x^B[i](B[i]很小)
令 p(x) ^ n 的系数为a[i], \sum q(x)^i 的系数为b[i]
求\sum a[i]b[i]

p(x) = \sum x^(-p[i])
q(x) = 1 - \sum x^B[i]
求p(x) ^ n % q(x) 的 x^0的系数

int n,m,p[11],B[111];
int ta[505];
void multi(int a[],int b[],int c[]){
    memset(ta,0,sizeof(ta));
    for(int i=0;i<=250;i++){
        if(!a[i])continue;
        for(int j=0;j<=250;j++){
            ta[i+j]=(ta[i+j]+1ll*a[i]*b[j])%mod;
        }
```

```
    }
    for(int i=500;i>250;i--){
        if(!ta[i])continue;
        for(int j=1;j<=m;j++)ta[i-B[j]]=(ta[i-B[j]]+ta[i])%mod;
        ta[i]=0;
    }
    for(int i=0;i<=250;i++)c[i]=ta[i];
}
int tb[555];
void qpow(int a[],ll k){
    memset(tb,0,sizeof(tb));
    tb[0]=1;
    while(k){
        if(k&1)multi(tb,a,tb);
        multi(a,a,a);
        k>>=1;
    }
    for(int i=0;i<=250;i++)a[i]=tb[i];
}
ll K;
int a[555],aa[555];
int main(){
    cin>>n>>m>>K;
    for(int i=1;i<=n;i++)cin>>p[i];
    for(int i=1;i<=m;i++)cin>>B[i];
    for(int i=1;i<=n;i++){
        memset(a,0,sizeof(a));
        a[1]=1;
        qpow(a,p[i]);
        for(int j=0;j<=250;j++)aa[j]=(aa[j]+a[j])%mod;
    }
    qpow(aa,K);
    for(int i=250;i>0;i--){
        for(int j=1;j<=m;j++)if(B[j]<=i)aa[i-B[j]]=(aa[i-B[j]]+aa[i])%mod;
        aa[i]=0;
    }
    cout<<aa[0]<<endl;
    return 0;
}
```