1001

$n=k$的时候答案是m^n, 否则是m^{n-k}。

1002

因为是排列，所以一个平方数x只会有$x/2$个有贡献的点对。

$2n$以内一共根号个平方数，所以有贡献的点对一共是$O(n^{1.5})$个。

相当于一个二维数点的问题，单点修改区间查询。

用树状数组的话复杂度是$O(n^{1.5}logn+qlogn)$。理论上是不能过的，但是跑很快哈，卡不掉。

也可以分块，$O(1)$修改，$O(n^{0.5})$询问，复杂度是$O(n^{1.5}+qn^{0.5})$。

1003

如果每个向量的三维加起来都相等的话，三维总和是定值，第三维就可以被前两维表示了。

于是转化为二维平面上的问题。判断一个点是否在一个三角形内。

题目没有限制向量三个维度的和，但是可以通过除法或者最小公倍数等方法令他们相等。

1004

这题方法非常多，讲一种写起来最简单的

假如我们设'a', 'b', 'c'三种颜色的权值分别为x,y,z且$x+y+z=0(0\le x,y,z\le 10^5)$有解当前仅当$x=y=z$, 则原问题等价于求路径和为0

于是我们可以设$x=998244353,y=10^9+7,z=-x-y$

暴力验证它是满足上述条件的，于是问题变成路径和为0

变成点分治/dsu 经典题

时间复杂度: $O(nlog^2n)$

1005

首先我们可以求出以(i,j)为右下角的最大正方形的边长，这个用递推就可以解决。

随后我们需要对每个$l \le mx_{len}$的正方形都加上l,这个可以用二次差分-前缀和的方法来解决。

二次差分的时候可以对水平、垂直、斜线做差分，有多种实现方法。

1006

考虑修改一个点pos，那么只会影响跨越他的区间$[l,r](l \le pos\ and\ pos \le n)$

于是我们可以对于每个k($-300 \le k \le 300$), 维护有多少区间满足原本区间和$+k$是完全平方数

这个可以通过从左向右扫描pos, 每次只会变化$O(n)$个区间来维护

时间复杂度: $O(n^2sqrt(a_i))$

1007

赛时过的大部分代码是卡常/套数据

这个题std是常数很大且很麻烦的$nlog^2n$做法（事实它跑不过暴力）

贴一下赛时跑的最快的暴力代码(4.8s):

```
__attribute__((always_inline)) __m256i redc(__m256i a){
    __m256i b=_mm256_mul_epu32(a, _mm256_set1_epi32(998244351));
    __m256i c=_mm256_add_epi64(a, _mm256_mul_epu32(b, _mm256_set1_epi32(mod)));
    return _mm256_srli_epi64(c,32);
}
int n,m;
u64 p[3][2*N],f[N];
//p[w][j] j*3-w
void work(int i){
    int j=min(i,n),dt=i/3;
    u64* P=p[i%3]+N;
    u64 I[]={to(1),to(1),to(1),to(1)};
    for(;j>3;j-=4){
        Store(f+j-3,redc(
            _mm256_add_epi64(
                _mm256_mul_epu32(
                    Load(f+j-3),
                    Load(I)
                ),
                _mm256_mul_epu32(
                    Load(f+j-4),
                    Load(P+j-dt-3)
                )
            )
        ));
    }
    for(;j;--j)f[j]=redc(to(f[j])+f[j-1]*P[j-dt]);
}
int main(){
    scanf("%d%d",&n,&m),*f=to(1);
    FOR(w,0,2)FOR(i,-n,n)p[w][N+i]=to((mod+i*3-w)%mod);
    FOR(i,1,n+m){
        work(i);
        // FOR(j,0,n)cerr<<redc(f[j])<<' ';
        // cerr<<endl;
    }
    printf("%u",redc(f[n]));
    return 0;
}
```

std的做法:

考虑知道输了几场，那么最终的分数K已知

假如最后强制赢一场，此时最后的$sum[n+1]=K+2$确定

将赢的时候的分数写入sum数组，显然$sum[i+1]\le sum[i]+2$

题目等价于要满足
$sum[0]=0, sum[n+1]=K, sum[i+1]\le sum[i]+2$

考虑容斥最后一段，满足$sum[i+1]>sum[i]+2$，会发现是一段递增的序列

令f_i表示容斥的多项式(不考虑首尾限制)，g_i表示答案多项式(不考虑首尾限制)， p_i表示答案多项式(考虑首尾限制)

考虑怎么求f_i，比较显然的做法是可以对$sum[i]$的数值进行dp

定义$dp[i][j]$表示当前考虑到i，距离上一个选的数距离为j

现在需要的是优化转移，我们可以定义一个矩阵

$$M_i = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ i*x & 0 & 1 \end{bmatrix}$$

会发现我们现在只需要跑矩阵乘法，可以利用分治ntt求矩阵的乘积

当然这一部分你也可以选择直接分治ntt，并记录左右端点分别距离最近拿的数多远

$g(x)_n = \sum_{i=0}^{n-1}(-1)^{n-i-1}f_{n-i}*g_i$

令$h(x) = \sum_{i=0}^{n}(-1)^i f_i x^i$

可以得到$g(x) = \frac{1}{1-h(x)}$

注意到最终答案要求$sum[n+1] = K+2$，$sum[0] = 0$

所以我们对这两个再求出对应的$O_1(x), O_2(x)$

$p_1(x)_n = \sum_{i=0}^{n-1}(-1)^{n-i-1}f_{n-i}*O_1 i$

$p_2(x)_n = \sum_{i=0}^{n-1}(-1)^{n-i-1}p_{n-i}*O_2 i$

$p_2(x)$就是最终答案多项式

赛时还有部分$O(n)$的做法, 由于出题人太菜了并不知道是怎么做的

1008

一道和博弈其实没什么关系的dp题

定义$dp[i][j]$代表假设当前剩下的序列为$[i,j]$, 先手最优操作下能否赢/最后剩下的值是多少

转移枚举下一次操作的位置$k(i \le k \le j)$类似于区间dp

为了去掉转移的复杂度，我们发现可以对于两个端点分别开一个单调队列

时间复杂度: $O(n^2)$

1009

我们对原图建出$kruskal$最小生成树的重构树来，对应原题的题意，每个点能够加入的集合对应从叶子到根的一条链。

因此我们可以针对这样的结构建立网络流模型，我们先强制令所有点染白，然后我们连接$(i, i+n, 1, a_i - b_i)$来建立调整为黑点的选择。

对于每条边在重构树上对应的节点，在重构树上都有两个孩子节点对应着这条边合并的两个集合，我们连接从子节点到自身的$(u, v, inf, 0)$边(这个过程和建立重构树的过程是一样的)，再建立一个新点连接有上下界的边来限制流量(对应题目要求)。但是对于点权小于当前边权的情况，我们需要支持删除操作，例如我们当前在u这条边，要删除第i个点，我们可以连接$(u, i, 1, 0)$使其流量回流，最后跑一个有上下界的最小费用可行流即可。

1010

观察到对于当前手中的数字x,走一步就会变成$k_i x + b_i$,走两步就会变成$k_i k_j x + k_j b_i + b_j$,两次操作$(k_i, b_i), (k_j, b_j)$可以合成为$(k_i k_j, k_j b_i + b_j)$满足结合律的性质。

利用这个性质，我们可以用倍增数组递推地求出从某个点开始，走x步的操作，对于询问，二进制拆分求解即可。

我们只需要记录倍增数组，倍增后的k，倍增后的b，以减少空间。

1011

我们首先考虑对于一个字符串T，有哪些区间是包含它的。

我们找到在S串里T串出现的位置集合$[l_1, r_1], [l_2, r_2], [l_3, r_3]$

这样我们就可以计算总贡献为$(r_2 - r_1) * l_1 + (r_3 - r_2) * l_2 + ...$

于是我们可以把式子拆开来算，将表达式的每一个部分$l_i r_{i+1}, l_i r_i$分别用线段树维护,再对其做一个链上的前缀和。

最后我们用T串在S串上跑，会产生$|T|$个节点，然后用之前维护的链上前缀和计算即可。

时间复杂度为$O(nlogn + \sum_{i=1}^{q} |T_i|)$