首先声明一点，考试即将结束的那段时间，评测机压力比较大，所以如果有因此导致评测机性能下降而超时的提交，可以和dxy说明，允许重测。

以及由于水平不够，独立出原创题难度对dxy来说有点大，考虑到有些同学有OI基础，dxy尽可能在找一些对绝大部分同学都比较陌生的题目，同时也对题目进行一些变化。

T1 数字游戏

出题人背锅环节

这道题本来的目的是想送给大家100分，这一题相对于往年OJ期末题目来说难度下降很多。

考场上情况如下： 37人通过，9人拿到65分，4人拿到80分，1人75分，1人70分，1人25分，1人20分。算是比较符合预期

然后我看了一下65分的提交，有部分同学使用dp计算，每组样例重新计算dp数组，导致TLE或者其他问题，还有一部分同学是没注意到最后没有翻倍次数时要及时停止，距离正确非常接近。

对于70分和75分的同学，部分测试样例因为常数问题没有通过，这里放宽时间限制到两秒后重新评测，都获得了80分（如果有觉得自己是常数过大而没有通过评测可以联系dxy）

对于25分和20分的同学，BFS时没有剪枝，导致TLE

重测后的结果：44人通过，8人80分，2人65分，1人25分，1人20分。如果有因为重测导致自己分数降低的同学，请联系dxy

题目来源

https://leetcode.cn/problems/minimum-moves-to-reach-target-score/

题目分析

本题需要注意的地方是，当m为0时，只能进行一种操作，在这种情况下不需要逐次递增，直接计算剩余轮数即可

60%数据

可以采用BFS+剪枝的策略，将$\{当前数值，当前翻倍次数\}$作为状态，初始状态为{1,0}，BFS搜索到$\{x, m\}, x \leq n$，或$\{n, y\}, y \leq m$时更新答案，BFS轮数需要剪枝，当轮数大于当前答案时，后续无须继续搜索。

示例代码

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define endl '\n'


int bfs(ll tar, ll m) {
    queue<pair<ll, ll>>q;
    ll t = 0;
    ll ans = 1e10;
    q.push({ 1,0 });
    int mint[100];
    while (!q.empty()) {
        if (t > ans)break;
        int n = q.size();
        for (int i = 0; i < n; i++) {
            auto p = q.front(); q.pop();
            if (p.first > tar)continue;
            if (p.first == tar)ans = min(ans, t);
            else if (p.second == m) {
                ans = min(ans, t + (tar - p.first));
            }
```

```
            else {
                q.push({ p.first + 1,p.second });
                q.push({ p.first << 1,p.second+1 });
            }
        }
        t++;
    }
    return ans;
}

void solve() {
    int n, m;
    cin >> n >> m;
    cout << bfs(n, m) << endl;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    int t;
    cin >> t;
    while (t--) {
        solve();
    }
}
```

80%数据

可以采用dp的策略，dp[i][j]表示到达数字i,使用不超过j次倍增，所花费的最少轮次，状态转移方程如下

$$初始情况：dp[1][0]=0$$
$$转移方程：dp[i][j]=min(dp[i-1][j]+1,dp[i][j-1],dp[i>>1][j-1]+1)，i是偶数$$
$$dp[i][j]=min(dp[i-1][j]+1,dp[i][j-1]),i是奇数$$

示例代码

```
#include<bits/stdc++.h>
using namespace std;
#define endl '\n'

int dp[101][10100] = { 0 };
void init() {
    for (int i = 2; i <= 10000; i++) {
        dp[0][i] = dp[0][i-1] + 1;
    }
    for (int i = 1; i <= 100; i++) {
        for (int j = 1; j <= 10000; j++) {
            dp[i][j] = dp[i - 1][j];
            dp[i][j] = min(dp[i][j], dp[i][j - 1] + 1);
            if ((j & 1) == 0)dp[i][j] = min(dp[i][j], dp[i - 1][j >> 1] + 1);
        }
    }
}

void solve() {
    int n, m;
    cin >> n >> m;
    if (n > 10000)return;
    cout<<dp[m][n]<<endl;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
```

```
    init();
    int t;
    cin >> t;
    while (t--) {
        solve();
    }
}
```

100%数据

贪心策略，我们会有一个想法，我们应该尽可能地把翻倍放在后面使用，这个想法会成为我们做题的出发点。本题等价于从目标元素执行递减和折半两种操作，其中折半操作限制m次，最终到达1所需要的最少操作次数。在这种情况下，我们尽可能优先折半，在无法折半时执行递减操作，得到一种贪心策略，以下我将证明这种贪心策略的正确性。

假设最优的操作序列如下，

a_0次递增，翻倍，a_1次递增，翻倍，...，... 翻倍, a_k次递增 ($k \le m$)，我们可以证明$a_i \le 1, \forall i \ge 1$，且如果$a_0 > 1$，则$k = m$

假设$a_i > 1, i \ge 1$，则我们可以将$a_i = a_i - 2, a_{i-1} = a_{i-1} + 1$，可以证明，这样的操作序列得到的结果依然是target，而操作次数减少一次，也就是说 $a_i = 0/1, i \ge 1$

对于第二个假设，我们可以将$a_0 = a_0$为奇数 $?1:0$,$a_1 = a_0/2$（整数除法），后续的a_i都向后递推，则得到的新的操作序列的结果不变，操作次数减少$a_0/2 - 1$，操作次数不少于之前的操作序列，因此我们可以贪心地逆向构造操作序列。

构造方法如下：

对于当前的target，如果它是奇数，则执行递减(target--)，否则，执行折半操作(target/=2)（如果执行递减操作，至少要执行两次后才能折半，违背了上述的$a_i \le 1$的要求），重复此操作直到得到1，或者执行减半次数为m为止，如果执行了m次减半操作，则剩下需要target-1次操作即可。(80%的数据没过可能是这个点没有注意。)

示例代码

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define endl '\n'

void solve() {
    ll n, m;
    cin >> n >> m;
    ll cnt = 0;
    while (n!=1&&m>0) {
        if (m && (n % 2) == 0) {
            cnt++;
            m--;
            n /= 2;
        }
        else {
            cnt++;
            n -= 1;
        }
    }
    cout << cnt+n-1 << endl;
}

int main() {
    int t;
    cin >> t;
    while (t--) {
        solve();
    }
}
```

T2 寿司盛宴

出题人背锅环节

本题本来打算作为一道中等题放在OJ上，本来想着避免浮点数误差，所以概率的分数运算改成逆元计算，考场上有些同学在计算转移矩阵时由于逆元计算导致花费时间很长，相应的，在考试结束后，对相应代码进行重测，避免因为这个原因卡常。

关于逆元这个点，没想到有很多同学有疑问，这里提前透露以后要学的抽象代数的内容。

对于一个非空集合G和一个G上的代数运算"·"构成（即对于任意$a, b \in G, a \cdot b \in G$)的,如果他们满足以下几条性质，则称$(G，\cdot)$构成一个群

- 结合律：即对所有的$a, b, c \in G$，有$(a \cdot b) \cdot c = a \cdot (b \cdot c)$
- **单位元**：即G中存在元素e，满足$\forall a \in G, e \cdot a = a \cdot e = a$
- **逆元**：即G中每个元素a，都存在元素b，使得$a \cdot b = b \cdot a = e$

一个群的例子为整数域中的加法运算群$(Z, +)$，注意在这里不存在我们平时见到的“减法”操作，“减法“操作可以理解为加上减数的逆元，即$a - b = a + (-b)$。

本题中的逆元在Z_p^*（整数的模p单位群）上，这个群的集合是{0,1,2,...p-1}，运算操作为乘法（对p取模)，这个群中的单位元是1，由费马小定理

$a^{p-1} \equiv 1 (mod\ p)$我们可以知道，$a$的逆元为$a^{p-2}$，因此除$a$操作在这个群里是乘$a$的逆元$a^{p-2}$,也就是$\frac{1}{a} \equiv a^{p-2} (mod\ p)$

最后关于多组样例，`dxy`出多组样例的目的本来是为了防止样例数据过弱，导致一些预料之外的策略可以获得更多分数，在做这题数据时，发现最终结果的n可以提出来，dp得到的结果对任何n都适用，因此将最后几个数据点改成T>1，希望有同学发现这个点，然后多拿点分，没想到因为这个改动，可能同学本来想dp做，但是觉得dp单次时间复杂度太高而放弃这种做法。

考试中该题的得分情况如下：2个同学拿到满分，8个同学拿到80分，27个同学拿到60分，1个同学拿到15分

题目来源

https://atcoder.jp/contests/dp/tasks/dp_j

题目分析

60%数据

对于60%的数据，题目被转化成这样一个问题：一共有n种盲盒，每次购买盲盒会从n种里随机获得一种，问能集齐n种盲盒需要购买次数的期望，这里可以直接计算，当你手上有k种盲盒时，下一抽能获得新盲盒的概率为$\frac{n-k}{n}$,这是个几何概型，在这时能抽到下一种盲盒的期望次数为$\frac{n}{n-k}$，我们用$dp[i]$表示抽到i种盒子需要购买次数的期望，则$dp[i+1] = dp[i] + \frac{n}{n-i}$，利用这个递推式可以求出结果

80%的数据

将维度拓展到三维，我们用三元组(i, j, k)表示当前由i个盘子里剩下一个寿司，有j个盘子里剩下2个寿司，有k个盘子里剩下3个寿司的状态，用$dp[i][j][k]$表示从状态(i, j, k)到状态$(0, 0, 0)$所需投骰子的轮数，我们可以推出转移方程

$$dp[i][j][k] = 1 + \frac{n-i-j-k}{n} dp[i][j][k] + \frac{i}{n} dp[i-1][j][k] + \frac{j}{n} dp[i+1][j-1][k] + \frac{k}{n} dp[i][j+1][k-1]$$

经过整理移项后可以得到

$$dp[i][j][k] = \frac{n}{i+j+k} + \frac{i}{i+j+k} dp[i-1][j][k] + \frac{j}{i+j+k} dp[i+1][j-1][k] + \frac{k}{i+j+k} dp[i][j+1][k-1]$$

对于这个式子，还有一种理解方法

$$dp[i][j][k] = \text{选到非空盘子的期望次数} + P(\text{选到剩下一个寿司的盘子}|\text{选到有寿司的盘子}) * dp[i-1][j][k] + \ldots + \ldots$$

利用这个式子，可以推出T=1时的结果

100%的数据

对于上面那个式子，我们可以发现dp的每一项都有且只有一个关于n的因数n，因此我们可以设计一个和n无关的dp方法

$$dp'[i][j][k] = \frac{1}{i+j+k} + \frac{i}{i+j+k}dp'[i-1][j][k] + \frac{j}{i+j+k}dp'[i+1][j-1][k] + \frac{k}{i+j+k}dp'[i][j+1][k-1]$$

$$dp[i][j][k] = n * dp'[i][j][k]$$

示例代码

```
#include<bits/stdc++.h>
using namespace std;
typedef long long ll;
#define endl '\n'
const int mod = 998244353;

ll ksm(ll b, ll e) {
    ll r = 1;
    while (e) {
        if (e & 1)r = (r * b) % mod;
        b = (b * b) % mod;
        e >>= 1;
    }
    return r;
}

ll inv(ll x) {
    return ksm(x, mod - 2);
}

ll dp[302][302][302];//表示还剩3个有 i,还剩2个有 j，还剩1个有 k
ll invarr[902];
int cnt[4];

void init() {
    memset(dp, 0, sizeof(dp));
    for (int i = 1; i <= 300; i++) {
        invarr[i] = inv(i);
    }
    for (int i = 0; i <= 300; i++) {
        for (int j = 0; j <= 300; j++) {
            for (int k = 0; k <= 300; k++) {
                //E(i,j,k)=E(unsucc) + P(i)*E(i-1,j+1,k)+ P(j)*E(i,j-1,k+1) +P(k)*E(i,j,k-1)
                if (i + j + k == 0) continue;
                if (i + j + k > 300)continue;
                ll iv = invarr[i + j + k];
                dp[i][j][k] += iv;//选择一个非零层蛋糕的期望次数
                if (i > 0) dp[i][j][k] += (((i * dp[i - 1][j + 1][k]) % mod) * iv) % mod;//吃
掉一个三层蛋糕的概率为
                if (j > 0) dp[i][j][k] += (((j * dp[i][j - 1][k + 1]) % mod) * iv) % mod;//吃
掉一个两层蛋糕
                if (k > 0) dp[i][j][k] += (((k * dp[i][j][k - 1]) % mod) * iv) % mod;//吃掉上一
个单个蛋糕
                dp[i][j][k] %= mod;
            }
        }
```

```
    }
}

void solve() {
    int n;
    cin >> n;
    cnt[1] = cnt[2] = cnt[3] = 0;
    for (int i = 0; i < n; i++) {
        int x;
        cin >> x;
        cnt[x]++;
    }

    ll ans = (dp[cnt[3]][cnt[2]][cnt[1]] * n) % mod;
    cout << ans << endl;
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);
    init();
    int t;
    cin >> t;
    while (t--) {
        solve();
    }
}
```