T1

贪心从左往右找到第一个 1，然后接着找到第一个 2，第一个 3，以此类推，时间复杂度为 $O(n)$。

T2

对于前30%的数据，暴力搜索以每一个平台为终点的最大长度再输出最大值即可。

考虑使用动态规划，$dp[i][j]$表示走到第i行第j列的平台时能经过的最大长度。

注意到状态只能从低处到高处单向转移，可推出状态转移方程：

```
    dp[i][j]=max(dp[i-1][j],dp[i+1][j],dp[i][j+1],dp[i][j-1])
```

加上只能从更低处转移的限制条件即可。

时间复杂度为$O(mnlogmn)$。

改用记忆化搜索可以使时间复杂度降为$O(mn)$。

```
#include<bits/stdc++.h>
using namespace std;
int ma=0;
int ind=0;
struct rua {
    int x;
    int y;
    int zhi;
} a[10000005];
bool cmp(rua a,rua b) {
    return a.zhi>b.zhi;
}
int main() {
    int r,c;
    cin>>r>>c;
    int dp[c+5][r+5];
    int mk[c+5][r+5];
    for(int i=0;i<r+5;i++){
        for(int j=0;j<c+5;j++){
            dp[j][i]=0;
            mk[j][i]=0;
        }
    }
    for(int i=1; i<=r; i++) {
        for(int j=1; j<=c; j++) {
            ind++;
            cin>>a[ind].zhi;
            a[ind].x=j;
            a[ind].y=i;
            mk[j][i]=a[ind].zhi;
        }
    }
    sort(a+1,a+r*c+1,cmp);
    for(int i=1; i<=r*c; i++) {
```

```
        if(mk[a[i].x+1][a[i].y]>a[i].zhi) dp[a[i].x][a[i].y]=max(dp[a[i].x+1][a[i].y],dp[a[i].x][a[i].y]);
        if(mk[a[i].x-1][a[i].y]>a[i].zhi) dp[a[i].x][a[i].y]=max(dp[a[i].x-1][a[i].y],dp[a[i].x][a[i].y]);
        if(mk[a[i].x][a[i].y+1]>a[i].zhi) dp[a[i].x][a[i].y]=max(dp[a[i].x][a[i].y+1],dp[a[i].x][a[i].y]);
        if(mk[a[i].x][a[i].y-1]>a[i].zhi) dp[a[i].x][a[i].y]=max(dp[a[i].x][a[i].y-1],dp[a[i].x][a[i].y]);
        dp[a[i].x][a[i].y]+=1;
        ma=max(ma,dp[a[i].x][a[i].y]);
    }
    cout<<ma;
    return 0;
}
```

T3

- 30%： 对于每轮，每个人O(n)的查询离自己**最远**的**未被淘汰**的人，总复杂度$O(n^3)$
- 50%： 先排序，发现每次只会把票投给最两端的的人之一，只需找到每个人投哪个即可，比30%减少了$O(n)$查询
- 100%： 会存在一个分界线，分界线左侧的人投给最最右边的，右侧反之。 那么每轮投票时，二分这个分界线，然后判断淘汰哪侧人即可

```
#include <iostream>
#include <algorithm>
#include <cstdio>
#include <cstring>
using namespace std;
#define in inline
const int _ = 1e6 + 23;
#define get getchar()

int read()
{
    int x = 1, t = 0;
    char ch = get;
    while ((ch < '0' || ch > '9') && ch != '-')
        ch = get;
    if (ch == '-')
        x = -1, ch = get;
    while (ch <= '9' && ch >= '0')
        t = t * 10 + ch - '0', ch = get;
    return t * x;
}

struct yzx
{
    int x, id;
} a[1000100];
int n;

in int cmp(yzx a, yzx b)
{
    return a.x < b.x;
```

```
}

in int check(int u, int l, int r)
{
    if (a[u].x - a[l].x > a[r].x - a[u].x)
        return 0;
    return 1;
}

int main()
{
    cin >> n;
    for (int i = 1; i <= n; ++i)

    {
        a[i].x = read();
        a[i].id = i;
    }
    sort(a + 1, a + n + 1, cmp);
    int L = 1, R = n;
    while (R - L > 1)
    {
        int l = L, r = R - 1, ans = l;
        while (l <= r)
        {
            int mid = l + r >> 1;
            if (check(mid, L, R))
                ans = mid, l = mid + 1;
            else
                r = mid - 1;
        }
        if (ans - L + 1 >= R - ans)
            R--;
        else
            L++;
    }
    cout << a[L].id << endl;
}
```

遥远的她 (distance)

30 分的做法当然直接 $O(n^4)$ 暴力枚举，简直是出题人良心的馈赠。

将平面绕原点旋转45度，并缩放2倍。然后，原本在 (X, Y) 的点移动到 $(X+Y, X-Y)$。

令每个点 P_i 在此变换后的坐标为 (x_i, y_i)。则有 $x_i = X_i + Y_i$ 和 $y_i = X_i - Y_i$。

接下来，我们考虑 $\mathrm{dist}(A, B)$ 的定义是如何变化的。

在原始定义中，兔子可以从 (X, Y) 跳到 $(X+1, Y+1)$, $(X+1, Y-1)$, $(X-1, Y+1)$, 和 $(X-1, Y-1)$；

因此，在变换后，它可以从 $(X+Y, X-Y)$ 跳到 $(X+Y+2, X-Y)$, $(X+Y, X-Y+2)$, $(X+Y, X-Y-2)$, 和 $(X+Y-2, X-Y)$。

将 $x = X+Y$ 和 $y = X-Y$ 替换后，它可以从 (x, y) 跳到 $(x+2, y)$, $(x, y+2)$, $(x, y-2)$, 和 $(x-2, y)$。

从 A 到 B 所需的最小跳跃次数是 $\text{dist}(A, B)$ 的定义（若无法到达，则 $\text{dist}(A, B) = 0$）。

接下来，我们考虑变换后的问题。

即，设 $P_i = (x_i, y_i)$，定义 $\text{dist}(A, B)$，并考虑如下的和：

$\sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \text{dist}(P_i, P_j)$，使用上述定义的 $\text{dist}(A, B)$。

显然，这会得到与原始问题相同的答案。

我们进一步考虑 $A = (x_1, y_1)$ 和 $B = (x_2, y_2)$ 之间的 $\text{dist}(A, B)$。

如果 $x_1 \not\equiv x_2 \pmod 2$ 或 $y_1 \not\equiv y_2 \pmod 2$，兔子无法从 A 到达 B，所以 $\text{dist}(A, B) = 0$。

否则，它正好等于曼哈顿距离的一半，即 $\frac{1}{2}(|x_1 - x_2| + |y_1 - y_2|)$。

注意到 $x_i = y_i + 2Y_i \equiv y_i \pmod 2$ 对所有 i 成立，这些 N 个点可以分为两组：x_i 和 y_i 都是偶数，或者 x_i 和 y_i 都是奇数。

对于属于不同组的两个点 A 和 B，$\text{dist}(A, B) = 0$；对于属于同一组的两个不同点，$\text{dist}(A, B) = \frac{1}{2}(|x_1 - x_2| + |y_1 - y_2|)$。

设 $E = \{E_1, E_2, \ldots, E_{|E|}\}$ 是一组点，其中 x_i 和 y_i 都是偶数，那么

$\sum_{i=1}^{|E|-1} \sum_{j=i+1}^{|E|} \text{dist}(P_{E_i}, P_{E_j}) = \frac{1}{2} \sum_{i=1}^{|E|-1} \sum_{j=i+1}^{|E|} |x_{E_j} - x_{E_i}| + \frac{1}{2} \sum_{i=1}^{|E|-1} \sum_{j=i+1}^{|E|} |y_{E_j} - y_{E_i}|$.

我们可以找到如下和：

$\sum_{i=1}^{|E|-1} \sum_{j=i+1}^{|E|} |x_{E_j} - x_{E_i}| = \sum_{i=1}^{|E|-1} \sum_{j=i+1}^{|E|} (x'_j - x'_i) = \sum_{i=1}^{|E|} (|E| + 1 - 2i) x'_i$,

其中 $(x'_1, x'_2, \ldots, x'_{|E|})$ 是对 $(x_{E_1}, x_{E_2}, \ldots, x_{E_{|E|}})$ 进行升序排序后的序列。

应用相同的讨论到 y 坐标，可以找到 $\sum_{i=1}^{|E|-1} \sum_{j=i+1}^{|E|} \text{dist}(P_{E_i}, P_{E_j})$；同样的讨论适用于 x_i 和 y_i 都是奇数的组，最终可以找到最终的和。

对两组的 x 和 y 坐标进行排序的时间复杂度为 $O(N \log N)$，其余计算的时间复杂度为 $O(N)$，所以这个问题可以在 $O(N \log N)$ 的时间内解。