The first one : fly

题意：

给定一个的地图，每个地图由 $N \times M$ 个边长为 100 的小正方形组成，其中有 K 个特殊的小正方形可以直接从左下角到右上角，其他的正方形只能沿边界移动， 求从 $(1,1)$ 的左下角出发， 到 (N,M) 的右上角的最短路径长度。

分析：

首先， 本题可以排除暴搜的做法。($N \times M \leq 10^{10}$， 基本无法通过剪枝减少 3 个量级)
观察一下本题的特性：

> - 最长的路径不会超过 $(N+M) \times 100$
> - 减少路径的长度只能通过增加在特殊的正方形上移动的次数
> - 在不往回走的境况下(即向下或向上走)，经过的特殊的正方形越多，路径越短
> - $K \in [1,1000]$

由此， 本题实际上是求经过的特殊的正方形的最大个数。
我们这样考虑：

- 我们每次行走只向右走， 若遇到特殊的正方形在当前位置的上方，即到这个特殊正方形的右上角(相当于数学中控制变量的思想，仅让高度为变量)。
- 再考虑贪心的思想，在保证现在的高度大于上一个经过的特殊的正方形的情况下， 如果我们控制在竖直方向上的高度尽可能小，那么我们通过特殊的正方形的概率就越大。
- 再回忆一下 LIS 的实现方法， 你会惊喜的发现，这种思路与其如出一辙。

然后就没有然后了...

Code：

```
using namespace std;
const int N = 1e6 + 50;
int n, h[N], m, K, ans, len, pos;
struct node {
    int x, y;
    // 这里的排序方式注意一下， x的升序和y的降序排列
    // 防止每一行选了两次， 如果先选较小的y，那么较大的y可以借此为跳板，更新LIS的值
    bool operator <(const node &tmp) const {
        if(x == tmp.x) return y > tmp.y;
        return x < tmp.x;
    }
} a[N];
int main()
{
    ios_base::sync_with_stdio,cin.tie(0),cout.tie(0);
    cin >>n >>m >>K;
    for(int i = 1, x, y; i <= K; i ++) {
        cin >>a[i].x >>a[i].y;
    }
    sort(a + 1, a + 1 + K);
    // LIS 模板
    for(int i = 1; i <= K; i ++) {
        int pos = lower_bound(h + 1, h + 1 + len, a[i].y) - h;
        h[pos] = a[i].y;
        len = max(len, pos);
    }
    double ans = (n + m - 2 * len + len * sqrt(2)) * 100;
        // 四舍五入
```

```
    int tmp = ans, flag = 0;
    if(ans - tmp >= 0.5) flag = 1;
    cout <<(tmp + flag);
    return 0;
}
```

The second one : puzzling

题意：

模拟拼图

分析：

数据范围 $1 \leq N,i,j \leq 5$，暴搜完全没有问题。

- 可以通过统计拼图中有意义的部分(1)总数的得到拼成之后的正方形的边长
- dfs 搜索每一个点，当前点已被覆盖之后在搜索下一个点，以保证每个正方形中的点均被覆盖
- 注意到数据中可能有 001 的情况，不能直接将拼图拼到地图上，要消除前导零的影响，可以直接枚举第一行的前导零的情况，毕竟，按照我的做法当枚举到 (x,y) 时，当前点之前的已完全被覆盖，只要考虑之后的即可
- !!! 数据第一个点会出现“空气”拼图

```
// 说实话, 本题的细节还是比较多的
using namespace std;
const int N = 100, Inf = 0x3f3f3f3f;
int n, vis[N][N], used[N], sum, m;
// 拼图个数， 当前状态， 是否被使用， '1'的总数， 计算得来的边长
struct node {
    int x, y;
    bool p[7][7];
}a[N]; // 拼图
bool _change(int x, int y, int id)
{
    // 尝试以 (x, y) 为 id 号拼图的左上角位置放拼图
    for(int i = x; i <= x + a[id].x - 1; i ++) {
        for(int j = y; j <= y + a[id].y - 1; j ++) {
        // 判断是否与其他拼图有重叠, 注意一定要判断当前拼图该位置是否为 '1'
            if(vis[i][j] && a[id].p[i - x + 1][j - y + 1] == 1) return false;
        // 是否出界
            if(a[id].p[i - x + 1][j - y + 1] == 1 && (i > m || j > m)) return false;
            if(a[id].p[i - x + 1][j - y + 1] == 1) vis[i][j] = id;
        }
    }
    return true;
}
// 当前拼到的位置， 已经拼好的拼图数目
void dfs(int x, int y, int step)
{
    if(step >= n + 1) {
    // 也可以判断 (x == m && y == m) , 更方便
        for(int i = 1; i <= m; i ++) {
            for(int j = 1; j <= m; j ++) {
                cout <<vis[i][j];
            }
            putchar('\n');
        }
        exit(0);
    }
```

```
    int tmp[N][N];
    memcpy(tmp, vis, sizeof(vis));
    if(vis[x][y]) {
        if(y == m) x ++, y = 1;
        else y ++;
        dfs(x, y, step);
    }
    else {
        for(int i = 1; i <= n; i ++) {
            if(used[i]) continue;
            for(int j = 1; j <= a[i].y; j ++) {
                // 枚举第一行的情况
                if(_change(x, y - j + 1, i) == 1) {
                    used[i] = 1;
                    // 当前位置被覆盖后再搜索下一个点
                    int k2 = y + (vis[x][y] != 0), k1 = x;
                    if(k2 > m) k1 ++, k2 = 1;

                    dfs(k1, k2, step + 1);
                    used[i] = 0;
                }
                // 注意回溯
                memcpy(vis, tmp, sizeof(vis));
                // 已经遇到一个'1',  结束枚举
                if(a[i].p[1][j] == 1) break;

            }
        }
    }
}
int main()
{
    cin >>n; int cnt = 0;
    for(int i = 1, x, y; i <= n; i ++) {
        cin >>x >>y;
        a[++cnt].x = x; a[cnt].y = y;
        char ch[15];
        bool flag = 1;
        for(int j = 1; j <= x; j ++) {
            cin >>(ch + 1);
            flag = 0;
            for(int k = 1; k <= y; k ++) {
                a[cnt].p[j][k] = ch[k] - '0';
                if(ch[k] == '1') sum ++, flag = 1;
            }
            // 判断是否有一行全为0
            a[cnt].x -= (flag == 0);
        }
        // 判断是否全为0
        if(a[cnt].x == 0) cnt --;
    }
    m = sqrt(sum); n = cnt;
    // 这里可以加一个特判: if(m * m != sum) {cout <<-1; return 0;}
    dfs(1, 1, 1);
    cout <<-1;
    return 0;
}
```

The third one : match

题意：

给定 n 只球队， 每支球队进行 n 天比赛， 每天比一场(或者轮空)。
现给定其中一只球队的比赛安排， 求出第 T 天的比赛情况。

分析：

本题实际上是一个模拟题，最简单的方法即是打开 样例 && 大样例， 分析一下即可...
给定第 d 只球队的比赛安排为 $abcdefg$

	a	b	c	d	e	f	g
1	d	c	b	**a**	g	f	e
2	e	d	c	**b**	a	g	f
3	f	e	d	**c**	b	a	g
4	g	f	e	**d**	c	b	a
5	a	g	f	**e**	d	c	b
6	b	a	g	**f**	e	d	c
7	c	b	a	**g**	f	e	d

洛谷

这是一种可行解， 也大概是唯一解。 这种排序方法应该比较容易理解吧...
求法: 通过观察可知， 第 T 天 a 的位置在 $(T+M-1)\%n$ (M 即为 d 的序号)。
让后可以发现从 a 向左推即为 $abcdefg$， 故而用循环解题即可。

Code :

```
using namespace std;
const int N = 1e5, Inf = 0x3f3f3f3f;
int n, M, T;
int a[N], ans[N];
int main()
{
    Read(n); Read(M); Read(T);
    for(int i = 1; i <= n; i ++) Read(a[i]);
    int k = (T + M - 1) % n, cnt = 0;
    for(int i = k + n; i >= k; i --) {
        ans[++cnt] = a[(i - 1) % n + 1];
    }
```

```
    for(int i = 1; i <= n; i ++) {
        print(ans[i]); putchar(' ');
    }
    return 0;
}
```

The last one : feather

分析：

数学的严格证明

题目

偷个懒吧...

Code：

```
using namespace std;
const int N = 2 * 1e5 + 50;
int a[N], maxn, sum, n;
int main()
{
    ios_base::sync_with_stdio,cin.tie(0),cout.tie(0);
    cin >>n;
    for(int i = 1; i <= n; i ++) cin >>a[i];
    a[n + 1] = a[1];
    for(int i = 1; i <= n; i ++) {
        maxn = max(maxn, a[i] + a[i + 1]);
        sum += a[i];
    }
    sum = (sum + (n/2) - 1) / (n/2);
    maxn = max(maxn, sum);
    cout <<maxn;
    return 0;
}
```