Problem A. Union of Squares

本题的关键性质是：有用的平面大小是 10^7 级别的。

对于只有 A 的情况。显然我们可以通过前缀和，计算出每个单位正方形是否被覆盖。

考虑所有正方形都能拆成若干个等腰直角三角形。也就是从顶点出发，往某个方向距离在 t 之内的都能覆盖，DP 每个位置能继续走多少步即可。

每个小方格根据左下角的格点来计算面积

Problem B. DNA Sequence

本题的关键在于理解题意，以及代码的实现。

暴力的做法就是直接维护类 DNA 序列。由于 **C** 操作的存在，元素个数可以达到指数级。

发现本质不同的氨基酸数量很少。对于每一个氨基酸我们都能用一对指针（复合）或者一个值（简单）来表示。于是我们可以维护一个氨基酸队列，每次对队头进行操作。唯一会新增氨基酸的操作是 **P**，我们只需要另外维护一个内存池即可。

事实上，通过指针达到节点的复用（本质不同的节点数很少）也是数据结构可持久化的常用方法。

Problem C. Special Book

先找性质

教授和学生都会按照自己到达的顺序读书，但是教授不会等别人，学生可以等后面人一起读

- 如果 A 先到，而 B 先看，一定可以把用时短的合并到用时长的人那里。

设计 DP

前 i 位教授，前 j 位学生，最后一位是 教授/学生，最早结束阅读的时间。枚举有多少位学生一起读，时间复杂度 $O(nm^2)$。

DP 优化

对于学生 A，如果存在学生 B 比 A 晚到同时阅读时间更长，那么 A 可以和 B 合并。

不考虑此类学生，那么学生的阅读时间就一定是递减的了。转移的时候只需要考虑是否和上一位学生一起读书即可（一定不会和教授的阅读时间重叠）。

具体来说

`dp[i][j][0]` 等于下述两个式子的较小值：

- max(`dp[i-1][j][0]`, professor[i].arrival) + professor[i].readingTime
- max(`dp[i-1][j][1]`, professor[i].arrival) +

professor[i].readingTime

`dp[i][j][1]` 等于下述两个式子的较小值:

- max(`dp[i][j-1][0]`, student[j].arrival) + student[j].readingTime
- max(`dp[i][j-1][1]`, student[j].arrival + student[j].readingTime)