2025 YCPC 題解

Contest Link (https://codeforces.com/gym/106059)

Problem A. Angle Problem **Medium Hard**

▼ 題意

平面上有 $n \leq 10^5$ 個點 $p_i = (x_i, y_i)$。每次查詢給一個區間 $[L, R]$ 與觀測點 $h = (X, Y)$，且保證 $Y > \max y_i$。要求最小視角 θ（以度為單位），使得從 h 出發、以寬度為 θ 的角錐可同時包含所有 $i \in [L, R]$ 的點。
共 $q \leq 10^5$ 查詢

題目就是要找 $[L, R]$ 中對於 h 視野最左與最右的兩個點，可以發現這兩個點一定會是 $[L, R]$ 所形成的**凸包**與 h 的切線上的切點。

有了以上的觀察，我們可以建線段樹，每節點保存該區間的凸包 (可以暴力建)，查詢時就對線段樹做區間查詢，對每個凸包用三分搜之類的找 h 的兩個切點，共 $O(\log n)$ 個凸包要找，總共會有 $O(2\log n)$ 個切點，答案就在這些切點中，接著枚舉這些切點即可找到最左及最右的點。

複雜度為 $O(n\log^2 n + q\log^2 n)$

Problem B. Binary Palindromes **Hard**

首先定義

- $dp[l][r]$: 區間 l, r 最後形成回文有幾種切法
- $dp0[l][r]$: 區間 l, r 最後形成回文 + 一個 0 有幾種切法
- $dp1[l][r]$: 區間 l, r 最後形成回文 + 一個 1 有幾種切法

轉移方法是從 $dp[l][r]$ 轉到 $dp0/dp1[l][r]$，再從 $dp0/dp1[l][r]$ 轉回 $dp[l][r]$

1. $dp[l][r]$ 轉到 $dp0/dp1[l][r]$：可以注意到如果要用推的話，$dp[l][r]$ 轉到 $dp0/dp1[l][j]$ 會是加到 $dp1$ for small j / 加到 $dp0$ for large j
所以你可以對每個 r 存說從 1 轉換到 0 的那個時刻，分別對每個 l 開一個變數代表 $\text{sum of } dp0[l][*]$ 以及 $\text{sum of } dp1[l][*]$，然後做區間 dp 的時候因為轉移順序的關係剛好就可以很自然地維護這兩個變數，撇除排序就可以均攤 $O(n^2)$ 做完
2. $dp0/dp1[l][r]$ 轉回 $dp[l][r]$：這裡就比較簡單，可以發現他是 $dp0[i][r]$ for 一段 i 的總和 + $dp1[j][r]$ for 一段 j 的總和，可以前綴和計算

▼ 程式碼

```cpp
#include <algorithm>
#include <iostream>
#include <vector>
using namespace std;

template <typename T> struct M {
  constexpr static T MOD = 998'244'353;
  T v;
  M(T x = 0) {
    v = (-MOD <= x && x < MOD) ? x : x % MOD;
    if (v < 0)
      v += MOD;
  }
  explicit operator T() const { return v; }
  bool operator==(const M &b) const { return v == b.v; }
  bool operator!=(const M &b) const { return v != b.v; }
  M operator-() { return M(-v); }
  M operator+(M b) { return M(v + b.v); }
  M operator-(M b) { return M(v - b.v); }
  M operator*(M b) { return M((long long)v * b.v % MOD); }
  M operator/(M b) { return *this * (b ^ (MOD - 2)); }
  friend M operator^(M a, int b) {
    M ans(1);
    for (; b; b >>= 1, a *= a)
      if (b & 1)
        ans *= a;
    return ans;
  }
  friend M &operator+=(M &a, M b) { return a = a + b; }
  friend M &operator-=(M &a, M b) { return a = a - b; }
  friend M &operator*=(M &a, M b) { return a = a * b; }
  friend M &operator/=(M &a, M b) { return a = a / b; }
};
using Mod = M<int>;

int main() {
  ios::sync_with_stdio(false);
  cin.tie(0);
  int t;
  cin >> t;
  while (t--) {
    int n, m;
    cin >> n >> m;
    string s, t;
    cin >> s >> t;
    vector<vector<Mod>> dp(n, vector<Mod>(n, Mod(0)));
    vector<vector<Mod>> dp0(n, vector<Mod>(n, Mod(0))),
        dp1(n, vector<Mod>(n, Mod(0)));
    vector<Mod> sum0(n), sum1(n);

    vector<int> p(n);
    for (int i = 0; i < n; i++) {
      for (int j = 0; j <= n; j++) {
        if (i + j >= n || j >= m || s[i + j] != t[j]) {
          p[i] = j;
          break;
        }
      }
```

```
    }
    vector<vector<pair<int, int>>> modify(n);
    for (int l = 0; l < n; l++) {
      for (int r = l; r + 1 < n; r++) {
        modify[l].push_back({r + 1 + p[r + 1], r});
      }
      sort(modify[l].begin(), modify[l].end());
      reverse(modify[l].begin(), modify[l].end());
    }

    for (int len = 1; len <= n; len++) {
      for (int l = 0; l + len - 1 < n; l++) {
        int r = l + len - 1;
        int mismatch = l + p[l];
        dp[l][r] = 1;
        if (mismatch <= r)
          dp0[l][r] = 1;
        else
          dp1[l][r] = 1;

        while (!modify[l].empty() && modify[l].back().first <= r) {
          auto [_, idx] = modify[l].back();
          sum1[l] -= dp[l][idx];
          sum0[l] += dp[l][idx];
          modify[l].pop_back();
        }

        dp0[l][r] += sum0[l];
        dp1[l][r] += sum1[l];
        if (l < r) {
          dp0[l][r] += dp0[l + 1][r];
          dp1[l][r] += dp1[l + 1][r];
          dp[l][r] += dp1[l + 1][r];
        }
        if (mismatch + 1 < n)
          dp[l][r] -= dp1[mismatch + 1][r];
        if (mismatch + 1 <= r)
          dp[l][r] += dp0[mismatch + 1][r];
        sum1[l] += dp[l][r];
      }
    }
    cout << dp[0][n - 1].v << '\n';
  }
}
```

整體時間複雜度瓶頸是排序的 $O(n^2 \log n)$，不含排序是 $O(n^2)$

另外還有暴力一點的解法，一樣存 $dp, dp0, dp1$，轉移方法一樣是從 $dp[l][r]$ 轉到 $dp0/dp1[l][r]$，再從 $dp0/dp1[l][r]$ 轉回 $dp[l][r]$

1. $dp[l][r]$ 轉到 $dp0/dp1[l][r]$，一樣是先注意到，$dp[l][r]$ 轉到 $dp0/dp1[l][j]$ 會是加到 $dp1$ for small j / 加到 $dp0$ for large j，這會對應到 $dp0$ 與 $dp1$ 中的某個 column 的區間和

2. $dp0/dp1[l][r]$ 轉回 $dp[l][r]$: 這邊也是 $dp0[i][r]$ for 一段 i 的總和 + $dp1[j][r]$ for 一段 j 的總和，會對應到 $dp0, dp1$ 中的某個 row 的區間和

所以可以用二維的資料結構去維護 $dp0, dp1$ (像是線段樹套BIT)，整體時間複雜度為 $O(n^2 \log^2 n)$

Problem C. Chess Pieces **Medium**

▼ 題意

給定三個棋子在二維平面上的初始位置 $(x_1, y_1), (x_2, y_2), (x_3, y_3)$，以及目標位置 $(x'_1, y'_1), (x'_2, y'_2), (x'_3, y'_3)$。
操作規則：選擇任一棋子移動到新位置，但該棋子與另外兩個棋子所形成的角度必須保持不變。問是否可以經過任意次操作，同時將三個棋子移動到目標位置。

分成兩個 case，起始位置共線/非共線

1. 初始三點不共線：首先可以觀察發現，移動的點能在外接圓弧上移動(圓周角公式)，也可以跳到以另外兩個不動點作為對稱軸的另一邊，雖然外接圓心可能會變動，但外接圓半徑一定不會改變，所以就檢查起始三點與終點三點的外接圓半徑是否一樣。

▼ 證明外接圓半徑不變

考慮三角形 ABC，設邊 $a = |BC|$，對應角為 $\angle A$。
由正弦定理可得：

$$\frac{a}{\sin A} = 2R$$

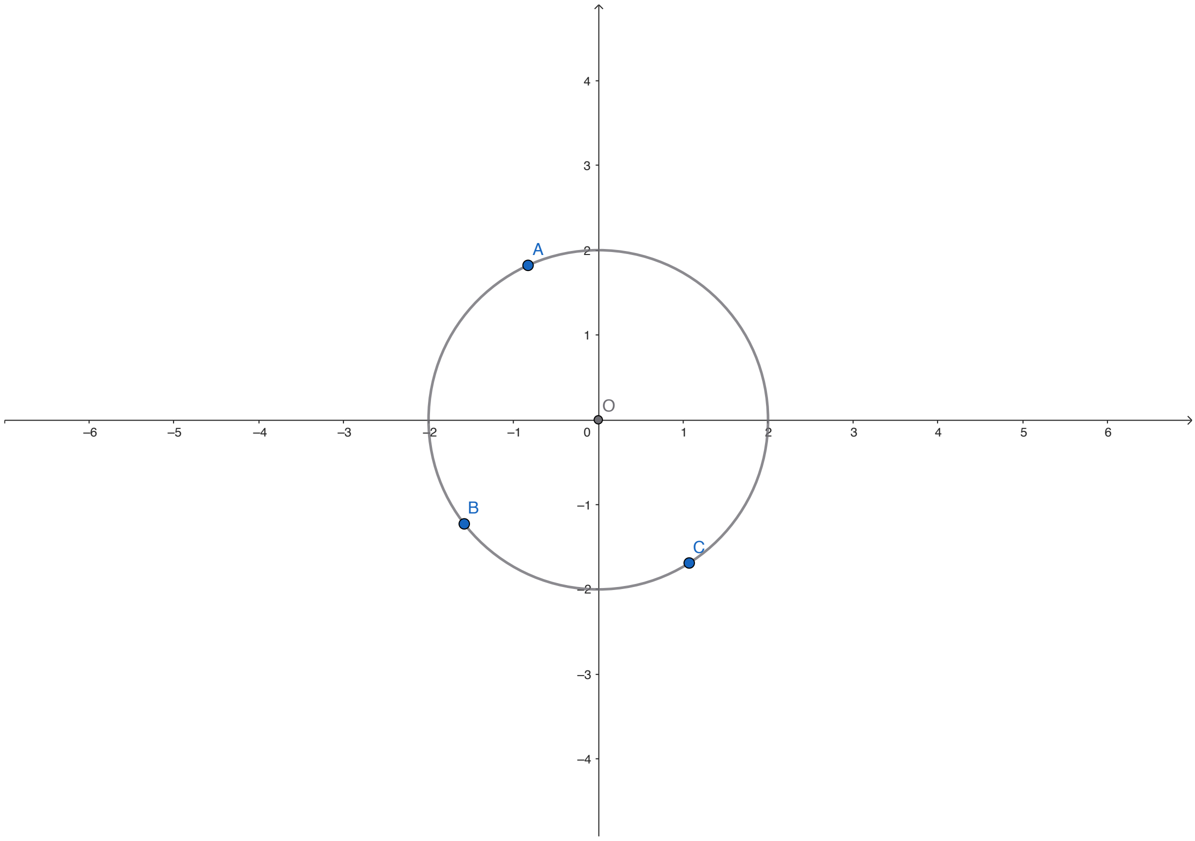
因此外接圓半徑 R 可寫為：

$$R = \frac{a}{2 \sin A}$$

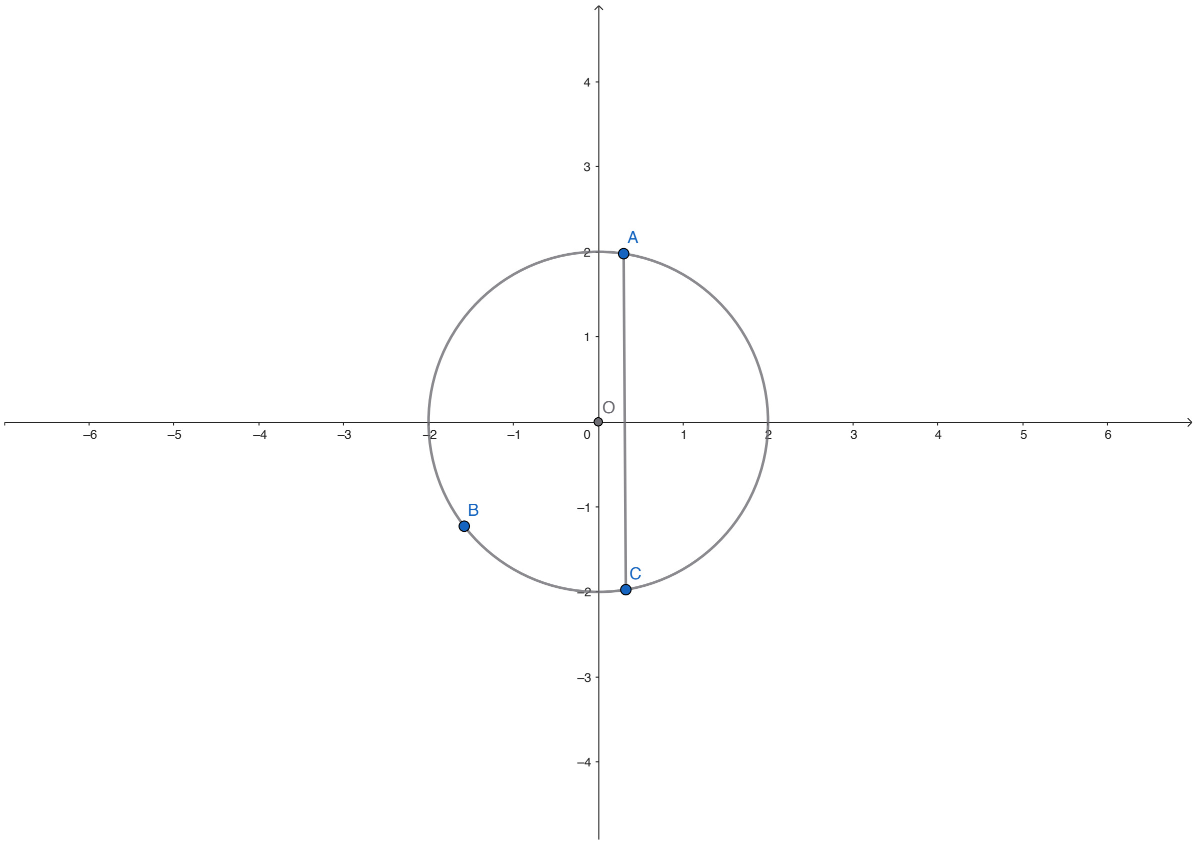
▼ 證明若兩外接圓半徑一樣，則答案一定是YES

我們可以想辦法將三個旗子的外接圓心，移動到目標位置的外接圓心，也就是讓外接圓心重疊，接著就可以在圓上移動到達三個目的點
可以透過以下的操作，使得圓心沿著 X 軸的方向平移一小段：

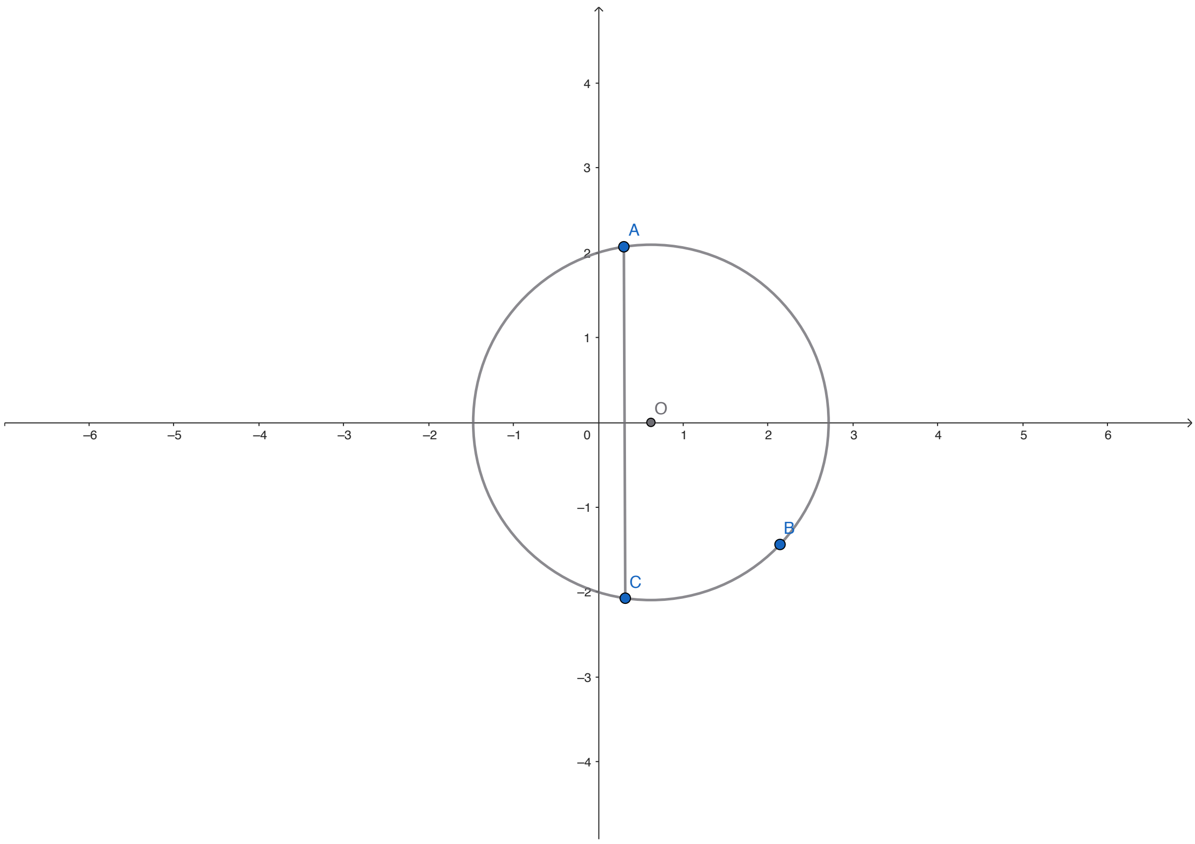
1. 一開始任意三點



2. 調整其中兩點線段垂直於 X 軸，並與圓心有一點點距離



3. 將剩下那點跳至以兩點線段為對稱軸的另一邊



藉由這個方法，可以將圓心沿 X 軸的方向平移任意 $\le R$ 的大小。且 Y 軸同理
所以透過多次操作將圓心移動至平面的任意一個點上

2. 初始三點共線：則根據規則，三點都只能在起始線上移動，且外側的兩點可以移動到另一邊的外側(例如：prq 三點可以變成 qpr 或 rqp)，特判一些線上的 case，和判斷 permutation 的 case 即可。

Problem D. Data Transmission **Easy Medium**

▼ 題意

樹上給多筆查詢 (a, b, c, d)，求兩條路徑 $a \rightsquigarrow b$ 與 $c \rightsquigarrow d$ 的節點聯集大小。

此題有許多不同的解法，以下只列出幾個

- 暴力枚舉四個點的組合所形成的 lca ，if-else 暴力判 path 的重疊情況，直接算出答案
- 將兩條 path 的長度加總，並扣掉重疊的部分：可以發現重疊的部分也是一條 path，而重疊的起點與終點和最高點會是某兩個點的 lca，所以也暴力枚舉四個點的組合所形成的 lca，去看哪些點同時在 $a \rightsquigarrow b$ 與 $c \rightsquigarrow d$ 上
- 用 Heavy-Light Decomposition，轉成序列問題 (區間的 Union 長度)

複雜度為 $O(q \log n)$ 或 $O(q \log^2 n)$

Problem E. Echoes on the Endless Line **Easy**

▼ 題意

給定陣列 A（敵人位置），對每個守望者位置 b，和固定距離區間 $(L, R]$，問有多少 a_i 落在距離區間內以及這些距離之和。

可以對 a_i 排序，接著變成經典的資料查詢二分搜與區間和問題。

Problem F. Forbidden Spell Sequence **Easy Medium**

▼ 題意

有 m 條規則，每條規則給定一些字母，代表字串不能同時出現這些字母。問有幾種長度為 n 的合法字串 (只能使用 `a` 到 `g` 的字母)，答案 mod $10^9 + 7$。

首先把字母集合 轉成 bitmask。

解法大概有兩種

- 透過排列組合與排容，對每個 bitmask 算出在使用這樣的字母集合下，有幾種長度為 n 的合法字串，接著對合法 bitmask 加起來即可
- 用矩陣快速冪：定義 $dp_{i,S}$ 為有幾種長度為 i 、使用字母集合 S 的字串，接著考慮每個字元 c，將答案轉移給 $dp_{i+1,S'}$ 其中 $S' = S \cup c$，可以發現變成線性遞迴 ($Adp_i = dp_{i+1}$，其中 A 是轉移矩陣，dp_i 是向量)，用矩陣快速冪加速就可以

第一種方法就算暴力一點寫也可以通過，第二種方法的時間複雜度為 $O(2^{21} \cdot \log n)$，剛好可以壓線過時限

Problem G. Graph Orientation **Medium**

▼ 題意

給定一個**二分圖**，每個點有權重 w_i。
我們要將所有邊定向，形成一個有向圖。設

- $T =$ 出度為 0 的點集合。
- 對每個點 i，定義 $d_i =$ 到 T 中任一點的最短有向路徑長度，若不存在則 $d_i = 10^9$。

代價定義為 $\text{cost} = \sum_i d_i \cdot w_i$
請定向邊，使 cost 最小，並輸出最小值及一組對應的邊方向。

首先可以發現，在最佳解中，所有 $d_i \in \{0, 1\}$。

▼ 上面這句話的證明

若最大的 $d_i \geq 2$ (假設點 i 擁有最大 d_i)，我們可以透過改變 i 的出邊方向，在不改變其他點的 d_j 的情況下 (因為 d_i 最大)，將 i 變為出度為 0 的點 ($d_i = 0$)，讓解的 cost 更好。
所以我們可以不斷的做以上操作，讓 $\max d_i \leq 1$

定義 $T =$ 出度為 0 的點集合。

我們可以改寫 cost $= \sum_i w_i - \sum_{i \in T} w_i$ ($cost$ 變成所有點的權重和扣掉 T 的權重和)，問題轉化為要決定 T，使得 T 的權重和最大化

可以發現若 u, v 間有邊，最多只能讓其中一個點當 T 中的點，所以其實 T 是一個**獨立集**，問題變成經典的在二分圖上找帶權重最大獨立集，可以轉成帶權重最小點覆蓋，並用最小割 (min-cut) 去計算。

Problem H. Huge Subsets **Medium**

▼ 題意

給定一個長度為 $n \leq 10^5$ 的正整數陣列 $a_1, a_2, \ldots, a_n$。
對於每個 $1 \leq k \leq n$，定義多重集合 S_k 為從陣列中選取恰好 k 個元素的所有子集的總和集合。
要求計算每個 S_k 的 gcd。

這題有許多作法，以下為一種比較好證明的作法

1. 先算出所有元素和第一個元素的差的 gcd，也就是 $d = \gcd(a_2 - a_1, a_3 - a_1, \cdots, a_n - a_1)$
2. 對每個 $1 \leq k < n$，答案就是 $\gcd(\text{pref}[k], d)$，其中 $\text{pref}[k]$ 為 $a_1 + a_2 + \cdots + a_k$
3. 對 $k = n$，答案就是整體總和 $\sum a_i$。

▼ 證明 2. 正確性

首先觀察到 d 其實就是所有元素差的 gcd (因為 $\gcd(x, y) = \gcd(x - y, x)$)，也就是 $\gcd_{i<j}(a_i - a_j)$

1. 設 X、Y 為任兩個大小為 k 的子集，令 $S(X) = \sum_{i \in X} a_i$。則
$$S(X) - S(Y) = (a_{x_1} - a_{y_1}) + (a_{x_2} - a_{y_2}) \cdots$$
其中 $x_i \in X - Y$，$y_i \in Y - X$，而且 x_i 與 y_i 個數相同，所以差的部分可以寫成以上的形式，因為每個 $a_i - a_j$ 會是 d 的倍數，所以 $S(X) - S(Y)$ 也會是 d 的倍數
2. 因此所有大小為 k 的子集，都會在模 d 下同餘，若取任一代表 s（任一個大小為 k 的集合之和），則集合中所有元素都形如 $s + m \cdot d$。所以集合的 gcd 至少為 $\gcd(s, d)$。
3. 在這邊就簡略說明為何答案最大也是 $\gcd(s, d)$，可以用反證法去證。若實際的答案更大，則一開始找的 d 應該會更大，矛盾。

因此對任意 $1 \leq k < n$，答案恰好為 $\gcd($ 任一個 $\mathrm{S(X)}, d)$。實作時可取輸入順序中的前 k 個元素作為代表。

Problem I. Ice Sliding **Medium Hard**

▼ 題意

給定 $n \times m \leq 10^5$ 的格子地圖，包含空地與格子，從起點開始，一開始可以選上下左右其中一個方向開始滑動，撞牆或邊界後**立即右轉 90 度**再滑，重複直到到達終點。給起點與終點，求最少經過多少格子到達終點 (若重複經過相同的格子，算多格)，或不可達，輸出 -1)。

把 (cell,dir) 視為一個點(狀態)。每個點有唯一的下一個點，所以可以建出 out degree 為一的 functional graph。問題可轉為在 functional graph 上求起點到終點經過邊數，但開始方向可任選，且到達終點的方向有四種可能，所以要枚舉 16 組起點與終點對。

至於 functional graph 上求起點到終點經過邊數可以參考這題 (https://cses.fi/problemset/task/1160)。

Problem J. Jigsaw of Perfect Squares **Easy Medium**

▼ 題意

能否重排長度為 $n \leq 1000$ 的陣列 A 為 $b_1 \dots b_n$，使得對每個 $b_i + i$ 為完全平方數

把問題轉為 bipartite matching：左側第 i 個點對應 a_i，右側第 j 個點對應位置 j。若 $a_i + j$ 為完全平方數，則 i 與 j 建邊。檢查是否存在完美匹配即可 (最大匹配 = n)。

Problem K. Karl’s Dormitory Allocation **water**

▼ 題意

按題面公式計算 P, Q, R。

照題意實作即可。

Problem L. Lantern Festival **water**

▼ 題意

給一串 0/1 字元，輸出 1 的個數。

照題意實作即可。

Problem M. Median Replacement **Medium**

▼ 題意

首先定義一個操作：對於長度 m 的陣列 b，隨機選一位置，把它換成剩下 $m-1$ 個數字中的中位數。
給定長度為 $n \leq 2 \cdot 10^5$ 的陣列，並對陣列的所有前綴計算變為全相同值的所需期望步數。

假設現在要計算 $a_1 ... a_n$ 的期望回合數，令第 $\lfloor \frac{n}{2} \rfloor$ 小的數為小中位數，第 $\lfloor \frac{n}{2} \rfloor + 1$ 小的數為大中位數。

可以觀察發現，最終陣列只可能會全部變為小中位數或大中位數，取決於第一次操作的結果 (只會是小或大中位數)，後續操作的結果會全部與第一次的結果相同，所以我們只要分這兩個 case，各自算出對應的機率與期望回合，帶權重取平均即可。

再知道陣列最終會全部變成 x 後，假設還有 c 個數字還未變成 x，我們可以發現期望回合就是 $\frac{n}{c} + \frac{n}{c-1} + \frac{n}{c-2} + \cdots + \frac{n}{1}$ (可以參考 Coupon collector's problem)。

所以只要算出每個前綴的大小中位數(用 `pbds` 或 `priority queue` 或其他資料結構)，並且預先算好 $\frac{1}{1}, \frac{1}{2}, \frac{1}{3} \cdots$ 的前綴和 (算期望回合數時乘上 i)。