Crack 'n' Code

Sponsored by TAURUS

Pre POSN1

Editorial

Written by

Crack 'n' Code Problem Writer Team

โจทย์การแข่งขัน

ข้อ	ID	ชื่อโจทย์	Time Limit	Memory Limit
A1	shibuyajihen	อุบัติการณ์ชิบูย่า	0.5 second	32 megabytes
A2	triangle builder	สร้างสามเหลี่ยม	1 second	256 megabytes
A3	enigma	เครื่องเข้ารหัส	1 second	256 megabytes
B1	magic	มายากล	2 seconds	256 megabytes
B2	merge	รวมเป็นหนึ่ง	0.25 second	32 megabytes
C	body builder	นักกล้าม	1.5 seconds	256 megabytes

อุบัติการณ์ชิบูย่า (shibuyajihen) (100 คะแนน)

0.5 seconds, 32 megabytes

ผู้แต่ง: Leomotors

เนื้อหาที่ใช้: Observation

วิเคราะห์โจทย์

จากโจทย์ข้อนี้ที่ค่อนข้างยาว และเนื้อโจทย์มีแต่น้ำ เราสามารถสรุปได้ว่า สิ่งที่โจทย์ต้องการคือให้พิมพ์ตัว N ที่มีลักษณะดังนี้

$N = 3$ มี 5 บรรทัด ความกว้าง 5

```
#   #
##  #
# # #
#  ##
#   #
```

$N = 4$ มี 7 บรรทัด ความกว้าง 7

```
#     #
##    #
# #   #
#  #  #
#   # #
#    ##
#     #
```

วิธีที่ 1

จะสังเกตได้ว่า ตัว N ขนาด N จะมี $2N-1$ บรรทัด และความกว้างแต่ละบรรทัด (L) เป็น $2N-1$ โดยแบ่งเป็นบรรทัดแรกและบรรทัดสุดท้ายที่มี # เพียงสองตัว และ บรรทัดที่ 2 จนถึง $2N-2$ ที่มี # 3 ตัว

สำหรับบรรทัดแรกและบรรทัดสุดท้าย เราจะพิมพ์ # สองตัว โดยมีพื้นที่ว่างคั่น $L-2 = 2N-3$

สำหรับบรรทัดตรงกลาง เราจะต้องพิมพ์ # 3 ตัว โดยมีที่ว่าง s_1 และ s_2 โดยจากการสังเกต $s_1 = i-2$ เมื่อ i เป็นเลขบรรทัด $s_2 = L-s_1-3 = 2N-1-(i-2)-3 = 2N-2-i$

Time Complexity: $\mathcal{O}(N^2)$

วิธีที่ 2

เราสามารถมองตัว N เป็นเหมือนตารางรูปสี่เหลี่ยมจัตุรัส ขนาด $L \times L$ ได้ โดยจะมี # ในกรณีดังนี้

- $c = 1$ แถวซ้ายสุด
- $c = L$ แถวขวาสุด
- $r = c$ อยู่ในแนวทแยง

ทำการ for loop ไล่ตั้งแต่ $(r, c) = (1, 1)$ จนถึง $(r, c) = (L, L)$ แล้วตรวจสอบเงื่อนไขด้านบน หากตรงตามข้อใดข้อหนึ่งให้พิมพ์ # แต่หากไม่ ให้พิมพ์ space bar

Time Complexity: $\mathcal{O}(N^2)$

Solution Code

วิธีที่ 1

```c
#include <stdio.h>

// Print '#'
void s() {
    printf("#");
}

// Print '\n'
void endl() {
    printf("\n");
}

// Print space length of n
void t(int n) {
    for (int i = 0; i < n; i++) {
        printf(" ");
    }
}

int main(void) {
    int n;
    scanf("%d", &n);

    // First line
    s();
    t(2 * n - 3);
    s();
    endl();

    // Line 2 to 2N - 2
    for (int i = 2; i <= 2 * n - 2; i++) {
        s();
        t(i - 2);
        s();
        t(2 * n - 2 - i);
        s();
        endl();
    }

    // Last Line (2N - 1)
    s();
    t(2 * n - 3);
    s();
    endl();
}
```

วิธีที่ 2

```c
#include <stdio.h>

int main() {
    int n;
    scanf("%d", &n);
    n = 2*n - 1;
```

```
    for (int r = 1; r <= n; r++) {
        for (int c = 1; c <= n; c++) {
            if (c == 1 || c == n || r == c) {
                printf("#");
            } else {
                printf(" ");
            }
        }
        printf("\n");
    }
}
```

สร้างสร้ามเหลี่ยม (trianglebuilder) (100 คะแนน)

1 seconds, 256 megabytes

ผู้แต่ง: kunzaZa183

เนื้อหาที่ใช้: Brute Force, Geometry, Combinatorics

Official Solution

เนื่องจากค่าของ $N, M \leq 10$ ทำให้เราสามารถวน for loop 6 ชั้น เพื่อไล่จุดยอดสามเหลี่ยมทุกรูปที่เป็นไปได้แล้วใช้สูตรที่โจทย์ให้ในการหาพื้นที่สามเหลี่ยม

เพราะว่าโจทย์ให้หาค่าของ $2X$ ทำให้เราไม่จำเป็นต้องหารค่าของสามเหลี่ยมด้วย 2 จึงทำให้สามารถคำนวณพื้นที่สามเหลี่ยมโดยใช้ $|x_1y_2 + x_2y_3 + x_3y_1 - x_1y_3 - x_2y_1 - x_3y_2|$ ได้เพื่อหลีกเลี่ยงการคำนวณทศนิยม

เพื่อป้องกันการนับสามเหลี่ยมรูปเดิมซ้ำ เราจำเป็นต้องหารคำตอบที่ได้ด้วย 6 เพราะว่าจุดยอดสามจุดสามารถสับเปลี่ยนได้ $3! = 6$ แบบ

Time Complexity: $\mathcal{O}(N^3M^3)$

Solution Code

```c
#include "stdio.h"
#include "math.h"

int main()
{
    int n, m;
    scanf("%d %d",&n,&m);
    int total = 0;
    for (int i = 0; i <= n; i++)
        for (int j = 0; j <= m; j++)
            for (int k = 0; k <= n; k++)
                for (int l = 0; l <= m; l++)
                    for (int x = 0; x <= n; x++)
                        for (int y = 0; y <= m; y++)
                        {
                            // coords: (i, j),(k, l), (x, y)
                            int area = 0;
                            area += i * l + k * y + x * j;
                            area -= i * y + k * j + x * l;
                            total += abs(area);
                        }
    total /= 6;
    printf("%d",total);
}
```

เครื่องเข้ารหัส (enigma) (100 คะแนน)

1 seconds, 256 megabytes

ผู้แต่ง: neonaht

เนื้อหาที่ใช้: String, Math

Subtask 1 (30 คะแนน) $C_i = x$

ในปัญหาย่อยนี้ค่าของ C_i มีค่าเท่ากันหมด ดังนั้นในการ $shift$ แต่ละครั้งตัวอักขระที่เหมือนกันตั้งแต่เริ่มแรกก็จะยังคงเหมือนกันอยู่ เช่น A -> B และ A -> B สังเกตว่าตัวอักขระเปลี่ยนแต่ทั้งสองยังเป็นตัวอักขระเดียวกันอยู่ ส่วนตัวที่ต่างกันก็จะยังคงต่างกันเหมือนเดิม จึงสามารถนับจำนวนตัวอักขระหนึ่งที่ซ้ำกันมากที่สุดได้เลยโดยไม่ต้องทำการ $shift$ แม้แต่ครั้งเดียว

```
int count[26] = {};
int result = 0; // Answer

for(int i=0; K[i]!='\0'; i++) {
    count[K[i] - 'A']++;
    if(count[K[i] - 'A'] > result) {
        result = count[K[i] - 'A'];
    }
}
```

Time Complexity: $\mathcal{O}(N)$

Official Solution

สำหรับปัญหานี้สามารถใช้ในการแก้ **Subtask 2, 3** ได้เช่นกัน เนื่องจากปัญหานี้ค่าของ C_i อาจมีได้หลายค่า เราจึงต้องหาวิธีคำนวณจำนวนครั้งของการ $shift$ ที่เหมาะสม เพราะเราไม่สามารถ $shift$ ไปเรื่อยๆเป็น ∞ รอบได้เนื่องจากปัญหาด้านเวลาการรันโปรแกรม ฉะนั้นจึงจำเป็นต้องทดสมการ

- หากพิจารณาการ $shift$ ครั้งแรกเราจะได้ตัวอักขระใหม่เป็น $(K_i + 1 \times C_i) \mod 26$
- หากพิจารณาการ $shift$ รอบที่ n เราจะได้ตัวอักขระใหม่เป็น $(K_i + n \times C_i) \mod 26$

หากสังเกตดูจะพบว่าเมื่อทำการ $shift$ ไปทั้งหมด 26 รอบ เราจะได้ตัวอักขระใหม่เป็น $(K_i + 26 \times C_i) \mod 26$ ซึ่งเท่ากับ $(K_i + 27 \times C_i - C_i) \mod 26$ และเท่ากับ $(K_i + C_i - C_i)$ ซึ่งนั่นคือ K_i

ฉะนั้นปัญหานี้จึงจำเป็นต้อง $shift$ เพียงแค่ $26 - 1 = 25$ รอบเท่านั้น เนื่องจากรอบที่ 26 ข้อความจะวนกลับมาที่ค่าเริ่มต้นใหม่

Time Complexity: $\mathcal{O}(N)$

Solution Code

```c
#include <stdio.h>
#include <string.h>

int main() {
    char K[200007];
    int C[200007];

    gets(K);
    for(int i=0; K[i]!='\0'; i++) scanf("%d", &C[i]);

    int count[26] = {}; // Count number of characters
    int result = 0; // Answer

    for(int j=0; j<26; j++) {
        for(int i=0; i<26; i++) count[i] = 0; // Initial

        for(int i=0; K[i]!='\0'; i++) {
            count[(K[i] - 'A' + j*C[i])%26]++;
            int now = count[(K[i] - 'A' + j*C[i])%26];

            if(now > result) {
                result = now;
            }
        }
    }

    printf("%d", result);
    return 0;
}
```

มายากล (magic) (100 คะแนน)

2 seconds, 256 megabytes

ผู้แต่ง: Icy

เนื้อหาที่ใช้: Counting Sort, Two Pointers

Subtask 1 (30 คะแนน) $N \leq 200$

ในปัญหาย่อยนี้สามารถทำการ **Brute Force** ด้วยการลองทุกช่วงตั้งแต่ L ถึง R ทั้งหมดที่เป็นไปได้ โดยที่ $1 \leq L \leq R \leq N$

ระหว่างการพิจารณาช่วง L ถึง R ทุก ๆ ช่วงสามารถนับชนิดของคาถาที่ต่างกันในช่วงนี้ได้ใน $\mathcal{O}(N)$ ต่อการพิจารณาช่วง L ถึง R ทุกช่วง

ในการนับชนิดของคาถาในแต่ละช่วงสามารถสร้างตัวแปรชนิดอาเรย์มาเก็บข้อมูลได้ว่าในช่วง L ถึง R มีคาถา $A[i]$ หรือไม่สำหรับ $L \leq i \leq R$ โดยอาจศึกษาได้จากส่วนของโปรแกรมด้านล่าง

```
int have[1000010], count_distinct = 0;
void addNumber(int a) {
    if(have[a] == 0) {
        count_distinct += 1;
    }
    have[a] += 1;
}
```

จากส่วนของโปรแกรมนี้จะสังเกตได้ว่าจำนวนชนิดของคาถาที่ต่างกันจะถูกเก็บไว้ที่ตัวแปร $count\_distinct$

Time Complexity: $\mathcal{O}(Q \times N^3)$

Subtask 2 (30 คะแนน) $N \leq 1\,000$

ในปัญหาย่อยนี้สามารถลองพิจารณาทุก ๆ ช่วง L ถึง R ได้เหมือนกับ **Subtask 1** แต่ในการนับชนิดของเลขในแต่ละช่วงจะต้องทำได้ภายใน $\mathcal{O}(1)$ โดยสามารถสังเกตได้ว่าหลังจากการพิจารณาช่วง L ถึง R แล้ว เมื่อพิจารณาคาถาที่ $R+1$ ในช่วง L ถึง $R+1$ จะไม่ต้องนับใหม่ทั้งหมด นับเพียงแค่ส่วนที่เพิ่มเติมขึ้นมา (คาถาที่ $R+1$) เท่านั้น

Time Complexity: $\mathcal{O}(Q \times N^2)$

Official Solution

ในปัญหานี้แทนที่เราจะนับจำนวนช่วงที่มีชนิดของคาถาที่ต่างกันตั้งแต่ K คาถาขึ้นไป เราจะใช้หลัก **"เพิ่มเข้า-ตัดออก"** ในการแก้ปัญหานี้แทน โดย จำนวนช่วงที่มีชนิดของคาถาที่ต่างกันตั้งแต่ K คาถาขึ้นไป = จำนวนช่วงทั้งหมดที่เป็นไปได้ - จำนวนช่วงที่มีชนิดของคาถาต่างกันน้อยกว่า K

จำนวนช่วงทั้งหมดที่เป็นไปได้ ($all\_segments$) ในอาเรย์ใด ๆ ที่มีขนาด N จะมีค่าเท่ากับ $\frac{N\times(N+1)}{2}$ เนื่องจาก ในการพิจารณาเมื่อ R ใด ๆ ที่ไม่เกิน N จะมี L ที่เป็นไปได้ทั้งหมด R ค่า

$$all\_segments = \sum_{i=1}^{N} i = \frac{N \times (N+1)}{2}$$

ข้อสังเกต ยิ่งขนาดของช่วง L ถึง R ลดลงจะทำให้มีชนิดของคาถาลดลงเสมอ

ในการนับจำนวนช่วงที่มีชนิดของคาถาต่างกันน้อยกว่า K สามารถนับได้ด้วยการใช้ **Two Pointers Technique** ในการแก้ปัญหานี้จะพิจารณาช่วง L ถึง i โดยที่ L เป็นจุดที่อยู่ซ้ายสุดของช่วงที่ถูกเงื่อนไข (จำนวนช่วงที่มีชนิดของคาถาต่างกันน้อยกว่า K) โดยในการเพิ่มคาถาที่ i เข้าไปอาจจะทำให้มีจำนวนชนิดของคาถาที่ต่างกันมากกว่าหรือเท่ากับ K ได้ ซึ่งไม่ตรงกับเงื่อนไขที่เรากำลังพิจารณาอยู่ก็จะเลื่อน L ไปเรื่อย ๆ จนตรงกับเงื่อนไขที่เรากำลังพิจารณาอยู่ โดยอาจสร้างฟังก์ชันคล้าย ๆ กับ ส่วนของโปรแกรมตัวอย่างจาก **Subtask 1**

```
void removeNumber(int a) {
    have[a] -= 1;
    if(have[a] == 0) {
        count_distinct -= 1;
    }
}
```

จากวิธีการแบบนี้จะทำให้ส่วนที่พิจารณาอยู่ตั้งแต่ L (หลังจากการเลื่อน) จนถึง i มีขนาดเท่ากับ $i - L + 1$ ซึ่งเป็นส่วนที่ไม่ต้องการ จึงสามารถลบออกจากวิธีทั้งหมดได้

Time Complexity: $\mathcal{O}(Q \times N)$

Solution Code

```
#include <stdio.h>

typedef long long ll;

int a[1000010], count[1000010], count_distinct;
ll n, k;

void addNumber(int num) {
    if(count[num] == 0) {
        count_distinct += 1;
    }
    count[num] += 1;
}

void removeNumber(int num) {
    count[num] -= 1;
    if(count[num] == 0) {
        count_distinct -= 1;
    }
}

void solve() {
    scanf("%lld %lld", &n, &k);
    for(int i=1; i<=n; ++i) {
        scanf("%d", &a[i]);
    }
    ll answer = (n * (n + 1ll)) / 2ll, not_in = 0ll;
    ll left_most = 1ll;
    for(ll i=1; i<=n; ++i) {
        addNumber(a[i]);
        while(count_distinct >= k) {
            removeNumber(a[left_most]);
            left_most += 1ll;
        }
        not_in += (i - left_most + 1ll);
    }
    printf("%lld", answer - not_in);
    return ;
}

signed main() {
    int q;
    scanf("%d", &q);
    while(q--) {
        // clean up
        count_distinct = 0;
        for(int i=0; i<=1000000; ++i) {
            count[i] = 0;
        }
        solve();
        printf("\n");
    }
    return 0;
}
```

รวมเป็นหนึ่ง (merge) (100 คะแนน)

0.25 seconds, 32 megabytes

ผู้แต่ง: Snowfall
เนื้อหาที่ใช้: Observation, Math

Subtask 1 (20 คะแนน) v_i มีค่าเท่ากันหมดทุกตัว , $C = D$ และ C เป็นจำนวนคี่

ในปัญหาย่อยนี้จะพบว่าการเลือกมา X จำนวนจากชุดเลขที่มีอยู่แล้วใส่ค่าผลลัพธ์กลับไป เปรียบเสมือนกับการที่ลบเลขทั้งหมด $X - 1$ จำนวนจากชุดเลขดังกล่าว ทำให้ปัญหาถูกลดรูปมาเป็น "หากนำเลขออกจากชุดเลขจำนวน $C - 1$ จำนวนและนำออกกี่ครั้งก็ได้ จะเหลือเลขเพียงตัวเดียว และเป็นจำนวนคี่หรือไม่" ซึ่งสามารถพิจารณาว่าสามารถเอาออกจนเหลือ 1 ตัวได้หรือไม่จากเงื่อนไข

$$N - 1 \equiv 0 \pmod{(C - 1)} \tag{1}$$

และ v_0 เป็นจำนวนคี่หรือไม่

Time Complexity: $\mathcal{O}(N)$

Subtask 2 (30 คะแนน) v_i เป็นจำนวนคี่และ $C = D = 2$

ในปัญหาย่อยนี้จะทำการพิจารณาเกี่ยวกับจำนวนคู่และจำนวนคี่และผลรวมของเลขทั้งหมดในชุดเลขเป็นหลัก โดยจะทำการแยกพิจารณาเคสย่อยในการให้เลขกลุ่ม A(v_A) และ B(v_B) ทั้งหมด 4 เคส

- **เคสที่ 1 :** v_A **เป็นจำนวนคู่** v_B **เป็นจำนวนคู่** ค่าของ $S_A - S_B$ จะเป็นจำนวนคู่
- **เคสที่ 2 :** v_A **เป็นจำนวนคี่** v_B **เป็นจำนวนคู่** ค่าของ $S_A - S_B$ จะเป็นจำนวนคี่
- **เคสที่ 3 :** v_A **เป็นจำนวนคู่** v_B **เป็นจำนวนคี่** ค่าของ $S_A - S_B$ จะเป็นจำนวนคี่
- **เคสที่ 4 :** v_A **เป็นจำนวนคี่** v_B **เป็นจำนวนคี่** ค่าของ $S_A - S_B$ จะเป็นจำนวนคู่

ซึ่งจากทั้ง 4 เคสจะพบว่าเมื่อเลือกเลขมา 2 จำนวนแล้วนำผลลัพธ์มาใส่ในชุดเลข และลบเลขที่เลือกออก เปรียบเสมือนกับการที่นำ v_A, v_B ออกไปและใส่ $S_A - S_B$ กลับเข้าไป ซึ่งจะทำให้ผลรวมของเลขในชุดใหม่เพิ่มขึ้น $-v_A - v_B + (S_A - S_B) = S_A - S_B - v_A - v_B$ ซึ่งจะพบว่า $S_A - S_B - v_A - v_B$ เป็นจำนวนคู่เสมอ ดังนั้นไม่ว่าจะเลือกจำนวนให้กับกลุ่ม A หรือ B อย่างไร ผลรวมของชุดเลขใหม่จะมีภาวะคู่หรือคี่ (parity) เหมือนกับผลรวมของชุดเลขเก่า (หากเป็นจำนวนคู่ก็จะเป็นจำนวนคู่เหมือนเดิม หากเป็นจำนวนคี่ก็จะเป็นจำนวนคี่เหมือนเดิม) ทำให้ปัญหาถูกลดรูปมาเป็น "ผลรวมของเลขทั้งหมดในตอนเริ่มต้นเป็นจำนวนคี่หรือไม่" ซึ่งสามารถพิจารณาว่า N เป็นจำนวนคี่หรือไม่เนื่องจาก v_i เป็นจำนวนคี่ทุกตัวอยู่แล้ว

Time Complexity: $\mathcal{O}(N)$

Subtask 3 (10 คะแนน) $N \leq 3\,000$ และ $C = D$

ในปัญหาย่อยนี้จะต่อยอดมาจาก subtask 1 และ 2 เล็กน้อย จากเดิมที่จะทำการแยกเคสตามการเลือกจำนวนเพียง 1 จำนวนในแต่ละกลุ่ม จะเปลี่ยนเป็นพิจารณาค่า S_A และ S_B จากการเลือกโดยตรงแทน โดยจะแบ่งเป็นทั้งหมด 4 เคส

- **เคสที่ 1 :** S_A **เป็นจำนวนคู่** S_B **เป็นจำนวนคู่** ค่าของ $S_A - S_B$ จะเป็นจำนวนคู่
- **เคสที่ 2 :** S_A **เป็นจำนวนคี่** S_B **เป็นจำนวนคู่** ค่าของ $S_A - S_B$ จะเป็นจำนวนคี่
- **เคสที่ 3 :** S_A **เป็นจำนวนคู่** S_B **เป็นจำนวนคี่** ค่าของ $S_A - S_B$ จะเป็นจำนวนคี่
- **เคสที่ 4 :** S_A **เป็นจำนวนคี่** S_B **เป็นจำนวนคี่** ค่าของ $S_A - S_B$ จะเป็นจำนวนคู่

ซึ่งจากทั้ง 4 เคสจะพบว่าเมื่อเลือกเลขมา X จำนวนแล้วนำผลลัพธ์มาใส่ในชุดเลข และลบเลขที่เลือกออก เปรียบเสมือนกับการที่นำเลขที่กลุ่ม A และ B เลือกออกไป และใส่ $S_A - S_B$ กลับเข้ามา ซึ่งจะทำให้ผลรวมของเลขในชุดใหม่เพิ่มขึ้น $-S_A - S_B + (S_A - S_B) = -2S_B$ ซึ่งจะพบว่า $-2S_B$ เป็นจำนวนคู่เสมอ ดังนั้นไม่ว่าจะเลือกจำนวนให้กับกลุ่ม A หรือ B อย่างไร ผลรวมของชุดเลขใหม่จะมีภาวะคู่หรือคี่ (parity) เหมือนกับผลรวมของชุดเลขเก่า (หากเป็นจำนวนคู่ก็จะเป็นจำนวนคู่เหมือนเดิม หากเป็นจำนวนคี่ก็จะเป็นจำนวนคี่เหมือนเดิม) ทำให้ปัญหาถูกลดรูปมาเป็น "หากนำเลขออกจากชุดเลขจำนวน $C - 1$ จำนวนและนำออกกี่ครั้งก็ได้ จะเหลือเลขเพียงตัวเดียวได้หรือไม่ และผลรวมของเลขทั้งหมดในตอนเริ่มต้นเป็นจำนวนคี่" ซึ่งสามารถพิจารณาว่าสามารถเอาออกจนเหลือ 1 ตัวได้หรือไม่จากเงื่อนไข $(N-1) \equiv 0 \bmod (C-1)$

Time Complexity: $\mathcal{O}(N)$

Subtask 4 (10 คะแนน) $N \leq 3\,000$

ในปัญหาย่อยนี้จะต่อยอดมาจาก subtask 3 โดยที่จะทำทุกอย่างเหมือนกัน แต่จะพิจารณาเพิ่มว่าต้องเลือก $X = C$ จำนวนกี่ครั้งและ $X = D$ จำนวนกี่ครั้งเพื่อให้เหลือจำนวนเพียงแค่ 1 จำนวน ซึ่งจะได้เป็นสมการดังนี้

$$N - 1 = P(C-1) + Q(D-1)$$

เมื่อ P, Q เป็นจำนวนเต็มที่ไม่น้อยกว่า 0
ซึ่งจะสามารถลูปหาค่า P และ Q ได้โดยที่ $0 \leq P, Q \leq N$

Time Complexity: $\mathcal{O}(N^2)$

Subtask 5 (10 คะแนน) $D = C + 1$

ในปัญหาย่อยนี้จะต่อยอดมาจาก subtask 4 เล็กน้อย โดยที่จะทำทุกอย่างเหมือนกัน แต่จะเปลี่ยนรูปสมการใหม่เป็น

$$N - 1 = P(C - 1) + QC \tag{2}$$

เมื่อ P, Q เป็นจำนวนเต็มที่ไม่น้อยกว่า 0 ซึ่งหากจัดรูปสมการจะได้เป็น

$$\frac{N - 1 + P}{C} - P = Q \tag{3}$$

ซึ่งจะสามารถลูปหาค่า P ได้โดยที่ $0 \leq P \leq N$ และจะได้ค่า Q จากสมการดังกล่าวโดยที่ Q ต้องสอดคล้องกับเงื่อนไขในสมการแรก โดยสามารถเช็คได้จาก $(N - 1 + P) \equiv 0 \bmod C$ และ $(N - 1 + P)/C - P >= 0$

Time Complexity: $\mathcal{O}(N)$

Official Solution

ในปัญหาฉบับเต็มนี้จะต่อยอดจาก subtask 5 โดยที่จะเขียนสมการดังนี้

$$N - 1 = P(C - 1) + Q(D - 1) \tag{4}$$

เมื่อ P, Q เป็นจำนวนเต็มที่ไม่น้อยกว่า 0 ซึ่งหากจัดรูปสมการจะได้เป็น

$$N - 1 - P(C - 1) = Q(D - 1) \tag{5}$$

$$\frac{N - 1 - P(C - 1)}{D - 1} = Q \tag{6}$$

ซึ่งจะสามารถลูปหาค่า P ได้โดยที่ $0 \leq P \leq N$ และจะได้ค่า Q จากสมการดังกล่าวโดยที่ Q ต้องสอดคล้องกับเงื่อนไขในสมการแรก โดยสามารถเช็คได้จาก $(N - 1 - P \times (C - 1)) \equiv 0 \bmod (D - 1)$ และ $(N - 1 - P \times (C - 1)) >= 0$

Time Complexity: $\mathcal{O}(N)$

Solution Code

```c
#include <stdio.h>
int main() {
    int n,c,d;
    scanf("%d %d %d",&n,&c,&d);
    int cur = 0;
    int parity = 0;
    for(int i = 0;i < n;i++) {
        scanf("%d",&cur);
        parity = (parity + cur) % 2;
    }
    if(parity == 0) {
        printf("No");
    }
    else {
        c--;
        d--;
        n--;
        int ck = 0;
        for(int i = 0;c * i <= n;i++) {
            if((n - c*i) % d == 0) {
                ck = 1;
                break;
            }
        }
        if(ck == 1) {
            printf("Yes");
        }
        else {
            printf("No");
        }
    }
    return 0;
}
```

นักกล้าม (Body Builder) (100 คะแนน)

1.5 seconds, 256 megabytes

ผู้แต่ง: JomnoiZ

เนื้อหาที่ใช้: Recursive, Brute Force, Preprocess

Subtask 1 (30 คะแนน) $N \leq 12, M \leq 1\,000$

วิธีที่ 1 : Recursive ข้อนี้เป็นการวัดความสามารถในการเขียนฟังก์ชัน Recursive พื้นฐาน ในการหารูปแบบ Combination ทั้งหมดที่เป็นไปได้ ซึ่งมีทั้งหมด $C(N, X)$ เมื่อ X แทนจำนวนเครื่องเล่นที่ถูกเลือกตามที่โจทย์ถาม โดยในระหว่างที่กำลังพิจารณาแต่ละ Combination ที่เป็นไปได้นั้น เราสามารถ Brute Force โดยพิจารณาทุก ๆ คู่ของเครื่องเล่นที่เป็นไปได้ แล้วเช็คว่ามีเครื่องเล่นใดเครื่องเล่นหนึ่งที่**ไร้ค่า**หรือไม่ ถ้ามีก็จะถือว่า Combination นั้น**ใช้ไม่ได้** ซึ่งสามารถเช็คความไร้ค่าได้ภายใน $\mathcal{O}(N^2)$ จากนั้นก็นับว่ามีกี่ Combination ที่ไม่มีเครื่องเล่นที่ไร้ค่าเลยสำหรับแต่ละคำถาม

วิธีที่ 2 : Bitmasks อีกหนึ่งวิธีที่ใช้ Time Complexity ในการรันเท่ากันนั่นก็คือ การใช้ Bitmask นั่นเอง โดยเราจะรันค่า bit ตั้งแต่ 0 ถึง $2^N - 1$ แทนแต่ละ Combination แล้วไล่เช็คค่า i ตั้งแต่ 0 ถึง $N - 1$ แทนเครื่องเล่นแต่ละเครื่อง โดยจะเช็คว่าเลข bit ปัจจุบันเมื่อนำมา AND กับ 2^i แล้วได้ค่าเท่ากับ 2^i หรือไม่ ถ้า**ใช่**เราจะถือว่าใน Combination นั้น ๆ**ใช้ได้** เราจะเลือกเครื่องเล่นที่ i มาใช้ในการพิจารณานั่นเอง จากนั้นเมื่อได้เครื่องเล่นครบ X เครื่องแล้วก็นำไปเช็คว่ามีเครื่องเล่นที่ไร้ค่าหรือไม่ แล้วนับจำนวน Combination ที่ไม่มีเครื่องเล่นที่ไร้ค่า

Time Complexity: $\mathcal{O}((2^N \times N^2) \times M)$

Subtask 2 (70 คะแนน) ไม่มีเงื่อนไขเพิ่มเติม

สังเกตว่าคำตอบที่เป็นไปได้จะมีแค่ N แบบเท่านั้น คือ เลือกเครื่องเล่นมา $1, 2, ..., N$ เครื่อง ดังนั้นเราสามารถเก็บคำตอบไว้ล่วงหน้าได้โดยการ Preprocess คำตอบเอาไว้ล่วงหน้าทั้ง 2^N วิธี แล้วก็แยกเก็บคำตอบตามจำนวนเครื่องเล่นที่ถูกเลือกของแต่ละ Combination จากนั้นเมื่อต้องการตอบแต่ละคำถามก็เพียงแค่นำค่าที่ Preprocess ไว้ก่อนหน้านี้มาตอบได้เลยภายใน $\mathcal{O}(1)$

Time Complexity: $\mathcal{O}((2^N \times N^2) + M)$

Solution Code : Recursive

```c
#include <stdio.h>

int N, sz;
int v[20], S[20], W[20];
int ans[20];

void solve(int cur) {
    int ok = 1;
    for (int i = 0; i < sz; i++) {
        for (int j = 0; j < sz; j++) {
            if (i != j && S[v[i]] >= S[v[j]] && W[v[i]] <= W[v[j]]) ok = 0;
        }
    }
    ans[sz] += ok;

    for (int nxt = cur + 1; nxt < N; nxt++) {
        v[sz++] = nxt;
        solve(nxt);
        sz--;
    }
}

int main() {
    int M;
    scanf(" %d %d", &N, &M);

    for (int i = 0; i < N; i++) scanf(" %d %d", &S[i], &W[i]);

    solve(-1);

    while (M--) {
        int X;
        scanf(" %d", &X);

        printf("%d\n", ans[X]);
    }
    return 0;
}
```

Solution Code : Bitmasks

```
#include <stdio.h>

int S[20], W[20];
int ans[20];

int main() {
    int N, M;
    scanf(" %d %d", &N, &M);

    for (int i = 0; i < N; i++) scanf(" %d %d", &S[i], &W[i]);

    for (int mask = 0; mask < (1<<N); mask++) {
        int v[20], k = 0;
        for (int i = 0; i < N; i++) {
            if (mask & (1<<i)) v[k++] = i;
        }

        int ok = 1;
        for (int i = 0; i < k; i++) {
            for (int j = 0; j < k; j++) {
                if (i != j && S[v[i]] >= S[v[j]] && W[v[i]] <= W[v[j]]) ok = 0;
            }
        }
        ans[k] += ok;
    }

    while (M--) {
        int X;
        scanf(" %d", &X);

        printf("%d\n", ans[X]);
    }
    return 0;
}
```