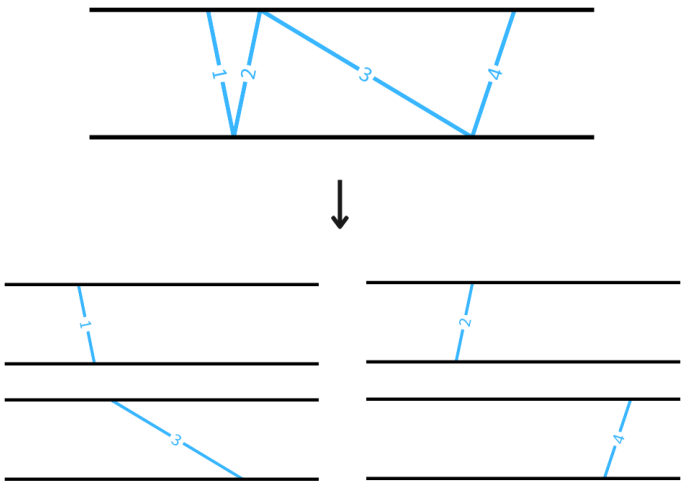
TOI-zero A2 editorial (เฉพาะบางข้อ)

ชร วาณิชยชลกิจ

April 12, 2025

1 A2-001 (beam)

ในข้อนี้เนื่องจากข้อจำกัดของ N, M ค่อนข้างน้อย (ไม่เกิน 300) เราเลยสามารถลองคิดไอเดียที่ค่อนข้างถึก และใช้ Runtime มาก ๆ ได้ โดยหนึ่งในไอเดียของข้อนี้คือการพิจารณาลำแสงทีละเส้นแยกจากกัน และหาคู่ลำแสงที่ตัดกันทั้งหมด รวมแล้วเราจะต้องพิจารณาลำแสงทั้งหมด $N \cdot M$ คู่



ในแต่ละคู่เส้นจะสามารถแบ่งได้เป็น 4 กรณีได้แก่

กรณีที่	ลำแสงที่ 1	ลำแสงที่ 2
1	บนไปล่าง	บนไปล่าง
2	ล่างไปบน	บนไปล่าง
3	บนไปล่าง	ล่างไปบน
4	ล่างไปบน	ล่างไปบน

โดยการตรวจสอบว่าเป็นกรณีไหน สามารถใช้สถานะความเป็น คู่-คี่ (parity) มาช่วยตรวจสอบได้ ซึ่งเราสามารถสังเกตได้อีกว่ากรณีที่ 1, 4 สามารถใช้วิธีคิดแบบเดียวกันได้ เช่นเดียวกับกรณีที่ 2, 3 และอาจมีกรณีที่ต้องดักเพิ่มเติม คือกรณีที่ลำแสงตัดกันที่จุดตัดพอดี

Implementation :

```
#include<stdio.h>
int a[505], b[505];
int cut1(int x1, int x2, int y1, int y2) {
   return (x1 < y1 && x2 > y2) || (x1 > y1 && y2 > x2);
}
int cut2(int x1, int x2, int y1, int y2) {
   return (x1 <= y1 && x2 > y1) || (x1 > y1 && y2 > x1);
}
int main() {
   int n, m;
   scanf("%d %d", &n, &m);
   for (int i = 1;i <= n;i++) scanf("%d", &a[i]);
   for (int i = 1;i <= m;i++) scanf("%d", &b[i]);
   int ans = 0;
   // meet at the event point
   for (int i = 0;i <= n;i++) for (int j = 0;j <= m;j++) {
      if (i % 2 == j % 2 && a[i] == b[j]) ans++;
   }
   for (int i = 0;i < n;i++) {
      for (int j = 0;j < m;j++) {
         if (i % 2 == j % 2 && cut1(a[i], a[i + 1], b[j], b[j + 1]))
            ans++; // from same side
         else if (i % 2 != j % 2 && cut2(a[i], a[i + 1], b[j], b[j + 1]))
            ans++; // from different side
      }
   }
   printf("%d\n", ans);
   return 0;
}
```

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

2 A2-002 (bigsquare)

ในข้อนี้เราต้องกังวลเรื่องของเวลาที่โปรแกรมใช้ (Runtime) เนื่องจาก N ที่มีค่าได้ถึง 100,000

เราจึงต้องหาวิธีการแก้ปัญหาข้อนี้ให้ไว ๆ ให้ได้

2.1 Subtask 1 (50 คะแนน)

เนื่องจากในปัญหาย่อยนี้มีข้อจำกัดคือ $N \leq 300$ เราจึงสามารถที่จะไล่จับคู่ทุก ๆ คู่จุดเพื่อหาขนาดเต็นท์ที่ใหญ่ที่สุดได้

โดยจุด (X_i, Y_i) และ (X_j, Y_j) จะจับคู่กันได้ ก็ต่อเมื่อ

$$X_i - X_j = Y_i - Y_j \text{ หรือ } X_i - X_j = Y_j - Y_i$$

2.2 Full Solution (100 คะแนน)

จาก Subtask 1 เราสามารถนำสมการมาจัดรูปให้เป็น

$$X_i - Y_i = X_j - Y_j \text{ หรือ } X_i + Y_i = X_j + Y_j$$

จากนั้นสำหรับแต่ละ i เราสามารถกำหนด $A_i = X_i - Y_i$ และ $B_i = X_i + Y_i$

โดยเราจะหาจุดที่มี A_i เท่ากัน หรือ B_i เท่ากัน แล้วเลือกคู่จุดที่มีค่า Y_i ห่างกันที่สุด (หรือ X_i ก็ได้)

สำหรับการหานั้นสามารถทำได้หลายวิธี เช่น การ Sort แล้วทำ Two-pointer technique หรือการใช้ map, unordered_map เนื่องด้วยค่าพิกัดจุดที่มากถึง 1,000,000,000

โดยใน solution นี้จะใช้วิธีแรก และจะมีการเขียน function ในการ sort เอง (Merge sort) ด้วย แต่เราสามารถใช้ built-in function ของ C, C++ ได้เช่นกัน ตัวอย่างเช่น qsort, std::sort

Implementation :

```
#include<stdio.h>
int x[100100], y[100100];
int a[100100], b[100100];
int ta[100100], tb[100100];
void merge_sort(int l, int r) {
    if (l == r) return;
    int mid = (l + r) / 2;
    merge_sort(l, mid), merge_sort(mid + 1, r);
    int pl = l, pr = mid + 1, pall = l;
    while (pl <= mid && pr <= r) {
        if (a[pl] < a[pr] || (a[pl] == a[pr] && b[pl] < b[pr])) {
            ta[pall] = a[pl], tb[pall] = b[pl];
            pall++, pl++;
        }
        else {
```

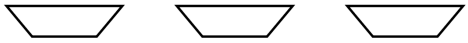
```
            ta[pall] = a[pr], tb[pall] = b[pr];
            pall++, pr++;
        }
    }
    while (pl <= mid) {
        ta[pall] = a[pl], tb[pall] = b[pl];
        pall++, pl++;
    }
    while (pr <= r) {
        ta[pall] = a[pr], tb[pall] = b[pr];
        pall++, pr++;
    }
    for (int i = l;i <= r;i++) {
        a[i] = ta[i], b[i] = tb[i];
    }
}
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1;i <= n;i++) {
        scanf("%d %d", &x[i], &y[i]);
    }
    for (int i = 1;i <= n;i++) {
        a[i] = x[i] + y[i];
        b[i] = y[i];
    }
    merge_sort(1, n);
    int mx = 0;
    for (int i = 0, j;i <= n;i = j) {
        for (j = i;j <= n && a[i] == a[j];j++);
        if (b[j - 1] - b[i] > mx) mx = b[j - 1] - b[i];
    }
    for (int i = 1;i <= n;i++) {
        a[i] = x[i] - y[i];
        b[i] = y[i];
    }
    merge_sort(1, n);
    for (int i = 0, j;i <= n;i = j) {
        for (j = i;j <= n && a[i] == a[j];j++);
        if (b[j - 1] - b[i] > mx) mx = b[j - 1] - b[i];
    }
    printf("%d\n", mx);
    return 0;
}
```

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

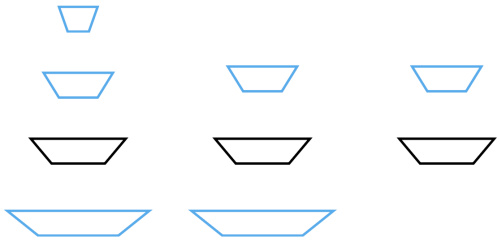
3 A2-004 (bowl)

ในข้อนี้เราต้องพิจารณาเงื่อนไขของโจทย์คือชามด้านบนต้องมีขนาดเล็กกว่าชามด้านล่าง จากนั้นเราสามารถสังเกตได้ หากมีชามที่มีขนาดเท่ากันหลายใบ มันจะไม่สามารถอยู่กองเดียวกันได้อย่างแน่นอน

ดังนั้นเราต้องใช้จำนวนกองชาม ไม่น้อยไปกว่าชามขนาดที่ปรากฏบ่อยที่สุดอย่างแน่นอน ซึ่งหากเราลองพิจารณาเฉพาะชามที่มีขนาดดังกล่าวจะได้กองจานี่มีหน้าตาดังนี้



เนื่องจากเรารู้ว่าจากขนาดที่เหลือจะไม่ปรากฏบ่อยไปกว่านี้อีกเลยเราเลยสามารถนำชามขนาดที่เหลือมาวางเรียงดังภาพ



และสามารถจัดเป็นกองชามที่ตามเงื่อนไขโจทย์ได้นั่นเอง

ดังนั้นข้อนี้สามารถทำได้โดยการใช้ Array ขนาด 300 (ขนาดชามสูงสุด) เพื่อทำ Counting sort และหาจำนวนของขนาดชามที่ปรากฏบ่อยครั้งที่สุดได้เลย

Implementation :

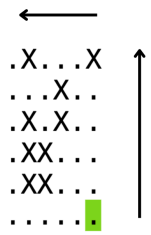
```c
#include<stdio.h>
int cnt[303];
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1;i <= n;i++) {
        int x; scanf("%d", &x);
        cnt[x]++;
    }
    int mx = 0;
    for (int i = 1;i <= 300;i++) {
        if (cnt[i] > mx) mx = cnt[i];
    }
    printf("%d\n", mx);
    return 0;
}
```

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

4 A2-006 (broken)

ในข้อนี้สามารถทำได้หลากหลายวิธี ไม่ว่าจะใช้ Graph Traversal เช่น DFS, BFS และยังสามารถใช้ Dynamic Programming ในการทำได้อีกด้วย โดยในที่นี้จะเฉลยเป็นวิธี Dynamic Programming

แทนที่เราจะมองว่าสามารถเริ่มจากจุดไหนได้บ้าง เราสามารถลองมองจุดจบเป็นจุดเริ่มแล้วถามว่าจบจุดไหนได้บ้างแทน



Dynamic Programming ในข้อนี้คือ $dp_{i,j}$ โดยนิยามว่า

$$dp_{i,j} = \begin{cases} 1 & \text{ถ้าสามารถเดินมายังช่อง } (i,j) \text{จากจุดเริ่มได้} \\ 0 & \text{ถ้าไม่สามารถเดินมายังช่อง } (i,j) \text{จากจุดเริ่มได้} \end{cases}$$

การที่เราจะเดินมายังช่อง (i,j) ได้เราจะต้องมาจากอีก 2 ช่องได้แก่

1. $(i+1,j)$
2. $(i,j+1)$

ทำให้ได้สมการ DP เป็น

$$dp_{i,j} = \begin{cases} 1 & (i,j) \text{ เป็นช่องว่าง และ } (dp_{i+1,j}=1 \text{ หรือ } dp_{i,j+1}=1) \\ 0 & \text{กรณีอื่น ๆ} \end{cases}$$

โดยลำดับการคำนวณจะเป็นตาม psuedo-code ดังนี้

```
for(int i=n;i>=1;i--){
    for(int j=n;j>=1;j--){
        // calculate dp[i][j]
    }
}
```

และมี Base case คือ $dp_{n,n}=1$ เพราะเป็นจุดเริ่มต้น และโจทย์รับประกันว่าเป็นช่องว่างแน่นอน

Implementation :

```c
#include<stdio.h>
char b[33][33];
int dp[33][33];
int main() {
    int n;
    scanf("%d", &n);
    for (int i = 1;i <= n;i++) {
        scanf("%s", b[i] + 1);
    }
    int ans = 0;
    dp[n][n] = 1;
    for (int i = n;i >= 1;i--) {
        for (int j = n;j >= 1;j--) {
            if (b[i][j] == 'X') continue;
            if (dp[i + 1][j] || dp[i][j + 1]) dp[i][j] = 1;
            if (dp[i][j]) {
                ans++;
            }
        }
    }
    printf("%d\n", ans);
    return 0;
}
```

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