Crack 'N' Code X CodeBlitz

Pre-TOI19

Editorial

Written by

Crack 'N' Code Problem Writer Team

โจทย์การแข่งขัน

วันที่	ข้อที่	ID	ชื่อโจทย์	Time Limit	Memory Limit
1	1	wordbuilder	สร้างคำวิเศษณ์	1 second	128 megabytes
	2	busan	รถไฟไปปูซาน	1 second	512 megabytes
	3	mineral	แร่ล้ำค่า	2 seconds	64 megabytes
2	1	mangoes	สุดยอดสวนมะม่วงของเจแปน	0.5 seconds	128 megabytes
	2	oranges	สวนส้มคู่แข่งเจแปน	1 second	32 megabytes
	3	tourist	นักท่องเที่ยว	2 seconds	512 megabytes

สร้างคำวิเศษณ์ (wordbuilder) (100 คะแนน)

1.0 seconds, 128 megabytes

ผู้แต่ง: Icy, JO, M-W

เนื้อหาที่ใช้: Greedy Algorithm, Shortest Path

Subtask 1 (12 คะแนน) $K = 1$

สังเกตว่า มีตัวอักษรที่เราต้องเก็บเพียงแค่ตัวเดียว ซึ่งสามารถใช้วิธีการ BFS ธรรมดา เริ่มจากเมืองที่ 1 ไปหาทุกเมืองที่มีตัวอักษรตัวนั้น แล้วส่งออกระยะห่างที่น้อยที่สุด (อย่าลืมคูณ 2 เนื่องจากต้องคิดระยะทางทั้งขาไปและขากลับ)

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

Subtask 2 (14 คะแนน) เมืองทุกเมืองมีตัวอักษรเดียวกัน

เราสามารถทำการ BFS จากเมืองที่ 1 ไปยังทุก ๆ เมือง หากว่าสายอักขระความยาว K นั้น มีตัวอักษรอื่นนอกจากตัวอักษรที่แต่ละเมืองมีอยู่ หรือ $K > N$ ก็จะไม่สามารถทำได้ จึงตอบ -1 ไม่เช่นนั้นคำตอบจะเป็น ผลรวมของระยะห่างของแต่ละเมืองที่มีค่าน้อยที่สุด K ตัวแรก ซึ่งสามารถทำได้โดยการ sort ระยะห่างของแต่ละเมืองจากน้อยไปมาก แล้วนำระยะห่างของ K เมืองแรกมาบวกกัน

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

Subtask 3 (19 คะแนน) $N, K \leq 500$

ในแต่ละตัวอักษรในสายอักขระ เราสามารถทำการ BFS แล้วหาระยะทางที่น้อยที่สุดได้เหมือน subtask 1 พร้อมกับทำสัญลักษณ์ว่าได้ทำการเก็บตัวอักษรจากเมืองที่มีระยะทางน้อยที่สุดนั้นไปแล้ว เพื่อที่จะไม่นำไปคำนวณซ้ำในตัวอักษรถัด ๆ ไป

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

Official Solution

เราจะทำการ BFS เพื่อหาระยะห่างจากเมืองที่ 1 ไปยังทุก ๆ เมือง และเก็บเข้า queue ของตัวอักษรทั้ง 26 (สามารถดำเนินการได้ระหว่างทำ BFS) จากนั้นไล่ดูตัวอักษรในสายอักขระความยาว K ทีละตัว หาก queue ของตัวอักษรที่ต้องการนั้นว่าง แสดงว่าไม่สามารถสร้างสายอักขระได้ ให้ตอบ -1 มิเช่นนั้นให้บวกค่าแรกของ queue เข้าไปในคำตอบ แล้ว pop ค่านั้นทิ้ง **โปรดระวังเรื่องการใช้ long long ซึ่งจะทำให้ได้คะแนนเพียงแค่ subtask 4**

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

Solution Code

```cpp
#include <bits/stdc++.h>
using namespace std;
const int MAX_N = 2e5 + 5;
char S[MAX_N], T[MAX_N];
vector <int> graph[MAX_N];
int dist[MAX_N];
queue <int> alphas[26];

int main() {
    cin.tie(nullptr)->sync_with_stdio(false);
    int N, M, K;
    cin >> N >> M >> K;
    cin >> (S + 1);
    while (M--) {
        int u, v;
        cin >> u >> v;
        graph[u].push_back(v);
        graph[v].push_back(u);
    }
    memset(dist, -1, sizeof(dist));
    queue <int> q;
    q.push(1);
    dist[1] = 0;
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        alphas[S[u] - 'A'].push(dist[u]);
        for (auto v : graph[u]) {
            if (dist[v] == -1) {
                dist[v] = dist[u] + 2;
                q.push(v);
            }
        }
    }
    cin >> (T + 1);
    long long ans = 0;
    for (int i = 1; i <= K; i++) {
        if (alphas[T[i] - 'A'].empty()) {
            ans = -1;
            break;
        }
        ans += alphas[T[i] - 'A'].front();
        alphas[T[i] - 'A'].pop();
    }
    cout << ans;
    return 0;
}
```

รถไฟไปปูซาน (busan) (100 คะแนน)

1 second, 512 megabytes

ผู้แต่ง: neonaht

เนื้อหาที่ใช้: Dynamic Programming , Meet in the middle

Subtask 1 (11 คะแนน) $N, M \leq 300$ และ $S = 0$

สำหรับปัญหาย่อยนี้จะสังเกตได้ว่าสถานีไหนก็ตามที่มีคนอยู่ทั้งสองฝั่งเราจะเลือกจอดแล้วรับคนขึ้นรถไฟทั้งหมด แต่หากสถานีไหนมีซอมบี้อยู่ทั้งสองฝั่งเราจะเลือกข้ามสถานีนั้น ส่วนสถานีที่เหลือเราสามารถหาวิธีการรับคนให้ได้มากที่สุดโดยที่ประตูไม่พังได้ด้วยการแก้ปัญหาแบบ Dynamic Programming โดยใช้วิธีการ Knapsack 0/1 แต่เนื่องจากประตูมีสองข้าง จึงต้องเพิ่มมิติที่ 3 ให้กับ dp ของเรา โดยเราจะนิยามให้ $dp[i][j][k]$ คือ จำนวนคนที่รับได้มากที่สุดเมื่อไปถึงสถานีที่ i โดยประตูข้างซ้ายและขวาโดนโจมตีไปแล้ว j, k หน่วยตามลำดับ

- เมื่อฝั่งซ้ายเป็น คน และฝั่งขวาเป็น ซอมบี้

$$dp[i][j][k] = \max(dp[i-1][j][k], dp[i-1][j][k+R[i]] + L[i])$$

- เมื่อฝั่งซ้ายเป็น ซอมบี้ และฝั่งขวาเป็น คน

$$dp[i][j][k] = max(dp[i-1][j][k], dp[i-1][j+L[i]][k] + R[i])$$

- เมื่อทั้งสองฝั่งเป็นคน

$$dp[i][j][k] = dp[i-1][j][k] + L[i] + R[i]$$

- เมื่อทั้งสองฝั่งเป็นซอมบี้

$$dp[i][j][k] = dp[i-1][j][k]$$

เนื่องจากยังไม่มีการสลับประตูเกิดขึ้น $(S = 0)$ คำตอบของเราจึงเป็น $dp[N][M][M]$ ได้เลย

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

Subtask 2 (15 คะแนน) $N, M \leq 300$

ปัญหาย่อยนี้แนวคิดยังคงเหมือนกับปัญหาย่อยแรก เพียงแต่เราต้องเพิ่มมิติที่สี่ให้กับ dp ตัวเดิมของเรา เนื่องจากสามารถเลือกสลับประตูได้ โดยจะนิยามให้ $dp[i][j][k][s]$ คือ จำนวนคนที่รับได้มากที่สุดเมื่อไปถึงสถานีที่ i โดยประตูข้างซ้ายและขวาโดนโจมตีไปแล้ว j, k หน่วย ตามลำดับ และสลับประตูไปแล้ว s ครั้ง

- เมื่อเกิดการสลับประตูขึ้น

$$dp[i][j][k][1] = \max(dp[i][j][k][1], dp[i][k][j][0])$$

เนื่องจากครั้งนี้อาจมีการสลับประตูเกิดขึ้น $(S = 1)$ คำตอบของเราจึงเป็น $\max(dp[N][M][M][0], dp[N][M][M][1])$

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

Subtask 3 (19 คะแนน) ฝั่งซ้ายของทุกสถานีจะไม่มีซอมบี้

ปัญหาย่อยนี้ทำให้เราสังเกตได้ว่าจริงๆแล้วเงื่อนไขการรับคนขึ้นรถไฟของประตูทั้งสองข้างไม่ได้มีความเกี่ยวข้องกันเลย สำหรับการเก็บคะแนนของปัญหาย่อยนี้ เราสามารถนิยามให้ $dp[i][j][s]$ คือ จำนวนคนที่รับได้มากที่สุดเมื่อไปถึงสถานีที่ i โดยประตูข้างขวาโดนโจมตีไปแล้ว j หน่วย และสลับประตูไปแล้ว s ครั้ง

ข้อสังเกต : ประตูฝั่งซ้ายจะไม่ถูกโจมตีเลยซักครั้งก่อนการสลับประตู ฉะนั้นการสลับประตูก็เปรียบเสมือนได้รับค่าความแข็งแรงของประตูกลับไปเป็น M อีกครั้ง เพราะประตูอีกข้างที่นำมาแทนที่ยังไม่เคยถูกโจมตี

- เมื่อฝั่งซ้ายเป็น คน และฝั่งขวาเป็น ซอมบี้

$$dp[i][j][s] = \max(dp[i-1][j][s], dp[i-1][j + R[i]][s] + L[i])$$

- เมื่อทั้งสองฝั่งเป็นคน

$$dp[i][j][s] = dp[i-1][j][s] + L[i] + R[i]$$

- เมื่อเกิดการสลับประตูขึ้น

$$dp[i][j][1] = \max(dp[i][j][1], dp[i][M][0])$$

คำตอบ : $\max(dp[N][M][0], dp[N][M][1])$

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

Subtask 4 (23 คะแนน) $S = 0$

เราสามารถนำข้อสังเกตของปัญหาย่อยก่อนหน้ามาใช้แก้ปัญหาย่อยนี้ได้ โดยจะทำการคิด Knapsack 0/1 แยกฝั่งแล้วค่อยนำคำตอบที่ได้จากฝั่งซ้ายและฝั่งขวามารวมกันตอนจบ โดยเราสามารถนิยามให้ $dp[i][j][0]$ คือ จำนวนคนที่รับได้มากที่สุดเมื่อไปถึงสถานีที่ i โดยประตูข้างขวาโดนโจมตีไปแล้ว j หน่วย และ $dp[i][j][1]$ คือ จำนวนคนที่รับได้มากที่สุดเมื่อไปถึงสถานีที่ i โดยประตูข้างซ้ายโดนโจมตีไปแล้ว j หน่วย

ข้อควรระวัง : อย่าลืม update ค่าของประตูให้ครบทั้งสองข้างครั้งเมื่อเดินทางไปถึงสถานีใดๆ

- เมื่อฝั่งซ้ายเป็น คน และฝั่งขวาเป็น ซอมบี้

$$dp[i][j][0] = \max(dp[i-1][j][0], dp[i-1][j+R[i]][0] + L[i])$$

$$dp[i][j][1] = dp[i-1][j][1]$$

- เมื่อฝั่งซ้ายเป็น ซอมบี้ และฝั่งขวาเป็น คน

$$dp[i][j][1] = \max(dp[i-1][j][1], dp[i-1][j+L[i]][1] + R[i])$$

$$dp[i][j][0] = dp[i-1][j][0]$$

- เมื่อทั้งสองฝั่งเป็นคน

$$dp[i][j][0] = dp[i-1][j][0] + L[i]$$

$$dp[i][j][1] = dp[i-1][j][1] + R[i]$$

- เมื่อทั้งสองฝั่งเป็นซอมบี้

$$dp[i][j][0] = dp[i-1][j][0]$$

$$dp[i][j][1] = dp[i-1][j][1]$$

คำตอบ : $dp[N][M][0] + dp[N][M][1]$

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

Official Solution

การแก้ปัญหาของข้อนี้สามารถนำวิธีการคิดของปัญหาย่อยที่ 4 มาใช้ได้เลย แต่เนื่องจากครั้งนี้อาจมีการสลับประตูเกิดขึ้นได้ $(S = 0, 1)$ ดังนั้นจึงต้องนำหลักการ Meet in the middle มาช่วยในการหาตำแหน่งของการสลับประตูที่ดีที่สุด แต่เนื่องจากเราไม่ทราบว่าการเดินทางโดยเริ่มจากสถานีใดๆไปยังเมืองปูซานนั้นจะรับคนได้มากที่สุดเท่าไหร่ จึงจำเป็นต้องมีการวิเคราะห์ปัญหาเพิ่ม

ข้อสังเกต #1 : เนื่องจากเราจำเป็นต้องแยกคิดประตูซ้ายและประตูขวาเพื่อให้โปรแกรมทำงานได้ทันเวลา ดังนั้นจึงไม่สามารถนำหลักการคิดของปัญหาย่อยที่ 2 มาใช้ได้

ข้อสังเกต #2 : เราต้องการทราบว่าการเดินทางโดยเริ่มจากสถานีใดๆไปยังเมืองปูซานนั้นจะรับคนได้มากที่สุดเท่าไหร่ เพื่อหาสถานีและเวลาที่เหมาะสมในการสลับประตู

ข้อสังเกต #3 : เราสามารถมองได้ว่าการเดินทางโดยเริ่มจากสถานีใดๆไปยังเมืองปูซานนั้น ไม่ต่างกับการเดินทางย้อนกลับจากเมืองปูซานไปยังสถานีต่างๆ

ดังนั้นการแก้ปัญหานี้จึงจำเป็นต้องทำ Knapsack 0/1 แยกฝั่ง และ แยกทิศด้วย โดยจะนิยามการทำ dp ดังนี้

$$dp[i][j][0] \Rightarrow Left - Forward$$

$$dp[i][j][1] \Rightarrow Right - Forward$$

$$dp[i][j][2] \Rightarrow Left - Backward$$

$$dp[i][j][3] \Rightarrow Right - Backward$$

จากนั้นเราจะนำค่าทั้งหมดที่ได้จากการ preprocessing มาใช้ในการหา Middle point โดยจะสังเกตได้ว่าเมื่อสลับประตู ค่าความแข็งแรงปัจจุบันของประตูซ้ายจะกลายเป็นค่าความแข็งแรงปัจจุบันของประตูขวาเมื่อเริ่มต้นที่สถานีถัดไป และในทำนองเดียวกันค่าความแข็งแรงปัจจุบันของประตูขวาจะกลายเป็นค่าความแข็งแรงปัจจุบันของประตูซ้ายเมื่อเริ่มต้นที่สถานีถัดไปเช่นกัน ฉะนั้นเมื่อแยกคิดตามประตู (ไม่ใช่ฝั่ง) เราจะได้จำนวนคนที่มากที่สุดที่สามารถรับได้โดยเกิดจากประตูแต่ละข้างเป็นดังนี้

- เมื่อต้องการสลับประตูที่สถานี i โดยประตูซ้ายโดนโจมตีไปแล้ว j หน่วย

$$LeftDoor = dp[i][j][1] + dp[i+1][M-j][2]$$

- เมื่อต้องการสลับประตูที่สถานี i โดยประตูขวาโดนโจมตีไปแล้ว j หน่วย

$$RightDoor = dp[i][j][0] + dp[i+1][M-j][3]$$

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

Solution Code

```cpp
#include "bits/stdc++.h"
using namespace std;

/* --- Official Solution --- */

const int SZ = 2e3+7;
const int MV = 3e3+7;

int L[SZ], R[SZ];
int dp[SZ][MV][4];

/*
0 --> Left Forward
1 --> Right Forward
2 --> Left Backward
3 --> Right Backward

+ --> People
- --> Zombies
*/

void knapsack(int N, int M, bool sym) {
    int l = sym ? 2:0;
    int r = sym ? 3:1;
    int c = sym ? 1:-1;
    int start = sym ? N:1;
    int end = sym ? 0:N+1;

    // Knapsack 0/1
    for(int i=start; i!=end; i-=c) {
        if(L[i]>=0 && R[i]<0) { // (+,-)
            for(int j=0; j<=M; j++) {
                dp[i][j][l] = max(dp[i+c][j][l], j+R[i]>=0 ? dp[i+c][j+R[i]][l] + L[i] : 0);
                dp[i][j][r] = dp[i+c][j][r];
            }
        }
        else if(R[i]>=0 && L[i]<0) { // (-,+)
            for(int j=0; j<=M; j++) {
                dp[i][j][r] = max(dp[i+c][j][r], j+L[i]>=0 ? dp[i+c][j+L[i]][r] + R[i] : 0);
                dp[i][j][l] = dp[i+c][j][l];
            }
        }
        else if(L[i]>=0 && R[i]>=0) { // (+,+)
            for(int j=0; j<=M; j++) {
                dp[i][j][l] = dp[i+c][j][l] + L[i];
                dp[i][j][r] = dp[i+c][j][r] + R[i];
            }
        }
        else { // (-,-)
            for(int j=0; j<=M; j++) {
                dp[i][j][l] = dp[i+c][j][l];
                dp[i][j][r] = dp[i+c][j][r];
            }
        }
    }
```

```
}

int main() {
    cin.tie(nullptr)->ios::sync_with_stdio(false);
    int N, M, S;
    cin >> N >> M >> S;
    for(int i=1; i<=N; i++) cin >> L[i];
    for(int i=1; i<=N; i++) cin >> R[i];

    knapsack(N, M, false); // Forward
    knapsack(N, M, true); // Backward

    int result = dp[N][M][0] + dp[N][M][1]; // default - not swap yet

    if(!S) {
        cout << result;
        return ;
    }

    // swap check - Meet in the middle
    for(int i=1; i<N; i++) {
        int best_left = 0;
        int best_right = 0;
        for(int j=0; j<=M ;j++) {
            best_left = max(best_left, dp[i][j][0] + dp[i+1][M-j][3]);
            best_right = max(best_right, dp[i][j][1] + dp[i+1][M-j][2]);
        }
        result = max(result, best_left + best_right);
    }
    cout << result;
    return 0;
}
```

แร่ล้ำค่า (mineral) (100 คะแนน)

2.0 seconds, 64 megabytes

ผู้แต่ง: spad1e, M-W

เนื้อหาที่ใช้: Binary Search, Divide & Conquer, Math

Subtask 1 (4 คะแนน) $A[i] = B[i]$

เนื่องจาก $A[i] = B[i]$ ไม่ว่าเราจะเพิ่มหรือลดช่วงอย่างไรจะได้ว่า $\sum_{i=L}^{R} A[i] = \sum_{i=L}^{R} B[i]$ เสมอ ดังนั้น $\frac{\sum_{i=L}^{R} A[i]}{\sum_{i=L}^{R} B[i]} = 1.000$

Time Complexity: $\mathcal{O}(1)$

Subtask 2 (9 คะแนน) $N \leq 1\,000$

ในปัญหาย่อยนี้สามารถใช้วิธีการ Brute Force ในการไล่ตรวจสอบทุกช่วงที่เป็นไปได้จากนั้นนำมา sort แล้วหาลำดับที่ K เพื่อตอบได้

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

Subtask 3 (7 คะแนน) $K = 1$

ในปัญหาย่อยนี้สิ่งหลักที่ต้องสังเกตคือจะมีแร่ i ที่ *คุ้มค่าที่สุด* นั่นคือแร่ที่มี $\frac{A[i]}{B[i]}$ มากที่สุด และหากเพิ่มแร่ชนิดอื่นเข้าไปจะทำให้มูลค่าลดลงเสมอ สมมติว่าเราต้องการเพิ่มแร่ j ให้ขุดพร้อมกับแร่ i เนื่องจากแร่ i *คุ้มค่าที่สุด* เราจะได้ว่า $\frac{A[i]}{B[i]} \geq \frac{A[j]}{B[j]}$ ซึ่งเมื่อย้ายข้างอสมการทำให้และบวก $A[i]B[i]$ ไปทั้งสองข้างจะได้ว่า $A[i]B[i] + A[i]B[j] \geq B[i]A[i] + B[i]A[j]$ เป็นจริง และได้ว่า $\frac{A[i]}{B[i]} \geq \frac{A[i] + A[j]}{B[i] + B[j]}$ เป็นจริงด้วย ดังนั้นเราสามารถไล่หาแร่ที่ *คุ้มค่าที่สุด* และตอบมูลค่าของแร่นั้นได้เลย

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

Subtask 4, 5 (12 + 11 คะแนน) $K \leq 1\,000$

จากข้อสังเกตในปัญหาย่อยที่ 3 จะได้ว่าค่าที่มากที่สุดลำดับที่ K จะมีแร่อยู่ไม่เกิน K ชนิด ดังนั้นเราสามารถไล่ Brute Force และ bound ให้ช่วงมีขนาดไม่เกิน K ได้ แต่ต้องมีการ optimize memory เพื่อไม่ให้ใช้ memory เยอะจนเกินไป เนื่องจากปัญหาต้องการค่าที่มากที่สุดลำดับที่ K จึงสามารถใช้ `std::priority_queue` ในการ maintain จำนวนที่มากที่สุด K อันดับแรกได้ และหากใน priority_queue มีสมาชิกมากกว่า K ตัวก็สามารถ pop สมาชิกที่มีค่าน้อยที่สุดออกได้

Time Complexity: $\mathcal{O}(NK \log(NK))$

Subtask 6 (20 คะแนน) $A[i]$ เท่ากันหมด, $B[i] \leq B[i+1]$

ในการแก้ปัญหานี้เราจะทำการ Binary Search บนคำตอบหรือก็คือมูลค่าที่มากที่สุดลำดับที่ K ให้มูลค่านั้นเป็น x หากเราเลือก L ใด ๆ จะได้ว่า $\frac{\sum_{i=L}^{R} A[i]}{\sum_{i=L}^{R} B[i]}$ จะเป็นลำดับลดลงตาม R ที่เพิ่มขึ้น ดังนั้นเราสามารถ binary search อีกครั้งเพื่อหาว่าหากเลือกค่า L แล้ว สามารถเลือกค่า R ได้มากที่สุดเท่าไหร่จึงจะไม่น้อยกว่า x จากนั้นบวกคำตอบของทุก ๆ ค่า L ว่าจำนวนช่วงที่เป็นไปได้มากกว่าหรือเท่ากับ K หรือไม่ หากมากกว่าก็สามารถให้ x เป็นคำตอบได้หรือหากน้อยกว่าก็ไม่สามารถให้ x เป็นคำตอบได้

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

Official Solution

เราจะทำการ Binary Search บนคำตอบของปัญหาข้อนี้ โดยเราจะพิจารณาว่า x สามารถเป็นคำตอบของปัญหานี้ได้ก็ต่อเมื่อมีช่วง $[L, R]$ ซึ่งเป็นค่าที่มากที่สุดลำดับที่ K บางช่วงที่สอดคล้องกับสมการ

$$\frac{\sum_{i=L}^{R} A[i]}{\sum_{i=L}^{R} B[i]} \geq x$$

และเนื่องจาก $\sum_{i=L}^{R} B[i] > 0$ ดังนั้นเราสามารถย้ายข้างอสมการและได้ว่า $\sum_{i=L}^{R} A[i] - x\sum_{i=L}^{R} B[i] \geq 0$ ซึ่งเมื่อแทนค่าในอสมการเป็น $X[i] = A[i] - xB[i]$ จะสามารถเขียนอสมการดังกล่าวใหม่ให้อยู่ในรูป $\sum_{i=L}^{R} X[i] \geq 0$ ได้ ดังนั้นเราลดรูปปัญหาให้อยู่ในรูปของการหาช่วง $[L, R]$ ที่เป็นช่วงมากสุดลำดับที่ K ของ $X[i]$ และตรวจสอบว่า $\sum_{i=L}^{R} X[i] \geq 0$ หรือไม่

ในการหาว่ามีช่วงดังกล่าวอยู่หรือไม่ เราจะทำการหาว่ามีช่วงที่มากกว่าหรือเท่ากับ 0 อยู่ทั้งหมดกี่ช่วง จากนั้นเทียบกับค่า K จะทำให้รู้ว่าช่วงที่มากที่สุดลำดับที่ K มากกว่าหรือน้อยกว่า 0 นั่นเอง โดยเมื่อเราแทนให้ $prefix\_X[i]$ แทน prefix sum ของ $X[i]$ แล้วช่วง $[L, R]$ จะมีค่ามากกว่าหรือเท่ากับ 0 ก็ต่อเมื่อ $prefix\_X[R] - prefix\_X[L-1] \geq 0$ และ $R \geq L$ เท่านั้นซึ่งปัญหานี้ก็จะตรงกับปัญหา classic อย่างการ count inversion ที่ใช้ divide and conquer ในลักษณะเดียวกันกับ merge sort algorithm ในการแก้ปัญหาได้ใน $\mathcal{O}(N \log N)$

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

Alternative Solution

อีกวิธีในการแก้ปัญหา count inversion สามารถใช้ Fenwick Tree หรือ Binary Indexed Tree มาช่วยในการแก้ไขปัญหานี้ได้ โดย Fenwick Tree เป็นเนื้อหาที่อยู่ในค่ายสสวท. ค่าย 1 แต่สามารถแก้ปัญหานี้ได้ใน Time Complexity $\mathcal{O}(N \log N)$ ซึ่งเท่ากับการเขียน merge sort

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

Solution Code

```cpp
#include <bits/stdc++.h>
using namespace std;
long long A[100001], B[100001], X[100001];
long long cnt_inv = 0;

void merge(int l, int r, int mid){
  vector<long long> L, R;
  for(int i = l; i <= mid; i++) L.push_back(X[i]);
  for(int i = mid + 1; i <= r; i++) R.push_back(X[i]);
  int lidx = 0, ridx = 0, idx = l;
  while(lidx < L.size() && ridx < R.size()){
    if(L[lidx] > R[ridx]) X[idx++] = L[lidx++];
    else{
      cnt_inv += (L.size() - lidx) * 1ll;
      X[idx++] = R[ridx++];
    }
  }
  while(ridx < R.size()) X[idx++] = R[ridx++];
  while(lidx < L.size()) X[idx++] = L[lidx++];
}

void count_inversion(int l, int r){
  if(l == r) return;
  int mid = (l + r) >> 1;
  count_inversion(l, mid);
  count_inversion(mid + 1, r);
  merge(l, r, mid);
}

int main(int argc, char* argv[]){
  long long N, K;
  scanf("%lld %lld", &N, &K);
  for(int i = 1; i <= N; i++){
    scanf("%lld", &A[i]);
    A[i] *= (1000 * 1ll);
  }
  for(int i = 1; i <= N; i++) scanf("%lld", &B[i]);
  long long L = 1, R = 1e7;
  while(L < R){
    long long mid = (L + R + 1) >> 1;
    X[0] = 0;
    for(int i = 1; i <= N; i++)
      X[i] = A[i] - (mid * B[i]) + X[i - 1];
    cnt_inv = 0;
    count_inversion(0, N);
    if(cnt_inv >= K) L = mid;
    else R = mid - 1;
  }
  double ans = double(L) / double(1000);
  printf("%.3lf", ans);
  return 0;
}
```

สุดยอดสวนมะม่วงของเจแปน (mangoes) (100 คะแนน)

0.5 second, 128 megabytes

ผู้แต่ง: CodeBlitz

เนื้อหาที่ใช้: Binary Search, Quick Sum

Subtask 1, 2 (15 + 16 คะแนน) $N, M \leq 50$ และ $R[i], C[i] \leq 500$

เราสามารถลองทุกค่า D ที่เป็นไปได้และหาค่า D ที่น้อยที่สุดที่สามารถรดน้ำต้นมะม่วงได้ครบตามเงื่อนไข และสำหรับแต่ละค่า D เราจะตรวจสอบ**ทุกคู่ของสปริงเกอร์และต้นมะม่วง**ว่าต้นมะม่วงอยู่ในช่วงที่สปริงเกอร์รดน้ำต้นไม้ถึงหรือไม่ ซึ่งจะใช้เวลา $\mathcal{O}(MN)$ สำหรับการตรวจสอบแต่ละค่า D

Time Complexity: $\mathcal{O}(NMD)$; $D = \max(R, C)$

Subtask 3 (18 คะแนน) $N, M \leq 1000$ และ $R[i], C[i] \leq 500$

สมมติให้ D_{ans} คือค่า D ที่เป็นระยะสั้นที่สุดที่สามารถรดน้ำต้นมะม่วงได้ครบตามเงื่อนไข(คำตอบที่ดีที่สุด) เราจะสังเกตได้ว่าถ้าหากเราลองค่า D ใดๆที่มีค่ามากกว่าค่า D_{ans} เราจะสามารถรดน้ำต้นมะม่วงได้ครบเสมอ และถ้าหากเราลองค่า D ใดๆที่น้อยกว่าค่า D_{ans} เราจะไม่สามารถรดน้ำต้นมะม่วงได้ครบตามเงื่อนไขเสมอ จากข้อสังเกตนี้ทำให้เราสามารถทำ Binary Search บนค่า D ได้ โดยการหาค่า D ที่น้อยที่สุดที่จะยังสามารถรดน้ำต้นมะม่วงได้ครบตามเงื่อนไข ซึ่งจะใช้เวลา $\mathcal{O}(\log D)$ และเราสามารถตรวจสอบว่าสามารถรดน้ำต้นมะม่วงได้ครบหรือไม่โดยใช้วิธีเดียวกับ subtask ที่ 2

Time Complexity: $\mathcal{O}(NM \log D)$; $D = \max(R, C)$

Official Solution

จากใน subtask ที่ 3 เรายังคง Binary Search หาค่า D เช่นเดิม แต่เราจำเป็นที่จะต้องหาวิธีที่จะตรวจสอบว่า "ต้นมะม่วงได้รับน้ำจากสปริงเกอร์เพียงพอหรือไม่" ให้เร็วมากยิ่งขึ้น ซึ่งหนึ่งในวิธีที่เป็นไปได้คือการใช้ Quicksum หรือ Sweepline 2D เพื่อใช้ในการหาว่าสำหรับแต่ละตำแหน่ง (r, c) ใดๆ จะมีสปริงเกอร์ทั้งหมดกี่อันที่รดน้ำถึง แล้วตรวจสอบว่าตำแหน่งของต้นไม้แต่ละต้นนั้นได้รับน้ำเพียงพอหรือไม่ ซึ่งแนวคิดนี้จะใช้เวลา $\mathcal{O}(RC \log D)$

อย่างไรก็ดีเราสามารถปรับปรุงแนวคิดให้ดียิ่งขึ้นได้จากการเปลี่ยนมุมมองจาก "สปริงเกอร์สามารถรดน้ำต้นไม้ได้ในช่วงระยะ D" เป็น "ในระยะ D รอบๆต้นไม้จะมีสปริงเกอร์อยู่กี่อัน" ทำให้สามารถทำ Quicksum 2D เก็บจำนวนสปริงเกอร์ไว้ล่วงหน้าและเรียกถามจำนวนสปริงเกอร์ทั้งหมดในช่วง (r_1, c_1) ถึง (r_2, c_2) ใดๆ ในเวลา $\mathcal{O}(1)$ หลังจากนั้นสำหรับแต่ละ D เราจะไล่ตรวจสอบต้นไม้แต่ละต้นว่าในระยะ D รอบๆต้นไม้นั้นมีสปริงเกอร์อยู่มากกว่าเท่ากับ $w[i]$ หรือไม่

Time Complexity: $\mathcal{O}(RC + N \log D)$; $D = \max(R, C)$

Solution Code

```cpp
#include <bits/stdc++.h>
using namespace std;

const int MAX_N = 1e5 + 1;
const int MAX_R = 5001;

int qs[MAX_R][MAX_R];
int Tr[MAX_N], Tc[MAX_N], W[MAX_N];

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

    for (int i = 1; i <= N; i++) scanf(" %d %d %d", &Tr[i], &Tc[i], &W[i]);
    for (int i = 1; i <= M; i++) {
        int Sr, Sc;
        scanf(" %d %d", &Sr, &Sc);

        qs[Sr][Sc]++;
    }

    for (int i = 1; i <= R; i++) {
        for (int j = 1; j <= C; j++) {
            qs[i][j] += qs[i][j - 1] + qs[i - 1][j] - qs[i - 1][j - 1];
        }
    }

    int l = 1, r = max(R, C);
    while (l < r) {
        int mid = (l + r) / 2;

        bool ok = true;
        for (int i = 1; i <= N; i++) {
            int min_i = max(1, Tr[i] - mid), max_i = min(R, Tr[i] + mid);
            int min_j = max(1, Tc[i] - mid), max_j = min(C, Tc[i] + mid);
            int sum_water = qs[max_i][max_j] - qs[min_i - 1][max_j]
                            - qs[max_i][min_j - 1] + qs[min_i - 1][min_j - 1];
            if (sum_water < W[i]) ok = false;
        }

        if (ok == true) r = mid;
        else l = mid + 1;
    }
    printf("%d", l);
    return 0;
}
```

สวนส้มคู่แข่งเจแปน (oranges) (100 คะแนน)

1.0 second, 32 megabytes

ผู้แต่ง: spad1e

เนื้อหาที่ใช้: Greedy Algorithm, Math

Subtask 1 (14 คะแนน) $x_1[i] = x_2[i]$ และ $y_1[i] = y_2[i]$

ในปัญหาย่อยนี้จะมีเพียงจุดเท่านั้น ดังนั้นหากเราสามารถตรวจสอบได้ว่าจุดใดบดบังกันก็สามารถลบจุดที่โดนบดบังออกได้เลย จุดหนึ่งจุดจะบดบังหรือโดนบดบังเป็นเส้นตรงที่ลากจาก $(0, 0)$ และ $(x[i], y[i])$ ดังนั้นหากมีเส้นตรงสองเส้นที่ความชันเท่ากันแล้ว จะมีหนึ่งในสองจุดที่ต้องนำออก ในการเก็บความชันของจุดก่อน ๆ สามารถใช้ data structure อย่าง std::set หรือ std::unordered_set เพื่อเพิ่มและตรวจสอบความชันที่เคยใส่แล้วได้โดยการนำ $gcd(x, y)$ ไปหารทั้ง y และ x ให้อยู่ในรูปของเศษส่วนอย่างต่ำก่อนใส่ใน set นั่นเอง

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

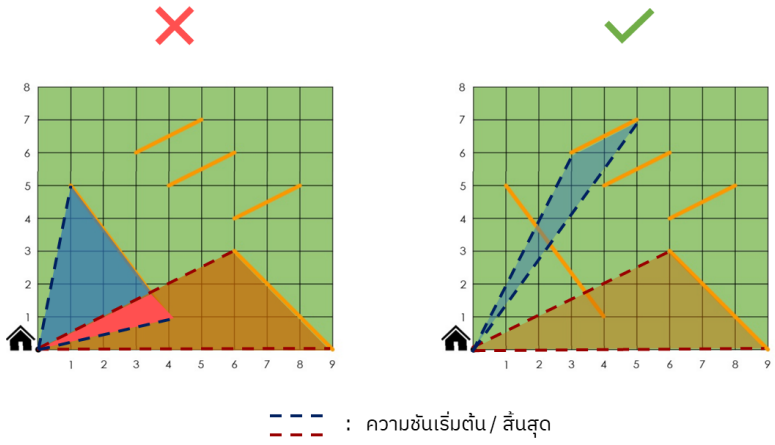
Subtask 2 (25 คะแนน) $y_1[i] = y_2[i] = 1$

เนื่องจาก $y_1[i] = y_2[i] = 1$ เราสามารถพิจารณาช่วงที่โดนบดบังในแกน $y = 1$ ได้ สังเกตว่าเส้นส้มหนึ่งเส้นจะตัดกับเส้นตรง $y = 1$ ที่จุด $x_1[i]$ และ $x_2[i]$ ซึ่งมองเป็นช่วงที่จะโดนบดบังได้ ดังนั้นหากเราหาได้ว่าต้องเลือกช่วงให้มีเส้นส้มได้มากที่สุดอย่างไร ก็จะสามารถตอบได้ว่าต้องถอนเส้นส้มออกอย่างน้อยที่สุดกี่เส้น ซึ่งปัญหานี้ก็สามารถแก้ไขได้ด้วยอัลกอริทึมคลาสสิกอย่าง Interval Scheduling[1] โดย Interval Scheduling เป็น greedy algorithm ที่จะทำการ sort ช่วงตามจุดจบและเลือกเส้นที่เลือกได้ทันที จากนั้นเมื่อพิจารณาช่วงถัด ๆ ไปก็จะพิจารณาว่าจุดเริ่มต้นซ้อนทับจุดสิ้นสุดของช่วงที่เคยเลือกไว้หรือไม่

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

Subtask 3 (26 คะแนน) $N \leq 20$

ข้อสังเกต #1: หากเลือกเส้นตรงหนึ่งเส้นจะมีช่วงที่ไม่สามารถเลือกได้เกิดขึ้น ซึ่งช่วงนั้นจะอยู่ระหว่างเส้นตรงที่ลากจาก $(0, 0)$ ไปยัง $(x_1[i], y_1[i])$ และ $(x_2[i], y_2[i])$ โดยเราสามารถมองเป็นช่วงของความชันได้ ในที่นี้เรียกค่าความชันที่น้อยกว่าว่า "ความชันเริ่มต้น" และเรียกค่าที่มากกว่าว่า "ความชันสิ้นสุด"



ปัญหาย่อยนี้สามารถแก้ได้ด้วยการทดลองทำตามเงื่อนไขโจทย์โดยตรงด้วยการเลือกสับเซตของเส้นส้มที่ต้องการเอาออกและตรวจสอบว่าเส้นส้มที่เหลืออยู่แต่ละเส้นตัดกันหรือไม่ (ใช้แนวคิดจากข้อสังเกต 1) เนื่องจากเรามีสับเซตที่เป็นไปได้ทั้งหมด 2^N สับเซต และการตรวจสอบว่าเส้นส้มเส้นหนึ่งตัดกับเส้นใด ๆ หรือไม่สามารถทำได้ใน $\mathcal{O}(N^2)$ ทำให้เราสามารถแก้ปัญหานี้ได้ใน $\mathcal{O}(N^2 2^N)$ ซึ่งเพียงพอสำหรับค่า $N \leq 20$

หมายเหตุ : ระวังช่วงที่มีความชันเริ่มต้นเท่ากับความชันสิ้นสุดของเส้นก่อนหน้า ไม่สามารถเลือกสองช่วงนั้น ๆ ได้

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

Subtask 4 (18 คะแนน) $N \leq 2\,000$

ในปัญหาย่อยนี้สามารถใช้ Dynamic Programming ในการแก้ได้ นิยามความชันเริ่มต้นและความชันสิ้นสุดเป็นค่าน้อยกว่าและค่ามากกว่าระหว่าง $y_1[i]/x_1[i]$ และ $y_2[i]/x_2[i]$ ตามลำดับ หากเราพิจารณาเส้นส้มจากค่าความชันเริ่มต้นที่เรียงจากน้อยไปมาก ในเส้นส้มลำดับที่ i จะได้สมการของอาเรย์ dp ว่า

$$dp[k] = \max(dp[k], dp[i] + 1) \quad \text{สำหรับทุก ๆ เส้นส้ม } k \text{ ที่ความชันเริ่มต้น} > \text{ความชันสิ้นสุดของ } i$$

ทำให้เราสามารถคำนวณ dp ได้ใน $\mathcal{O}(N)$ สำหรับแต่ละเส้นส้ม และส่งผลให้แก้ปัญหาทั้งหมดได้ใน $\mathcal{O}(N^2)$

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

Official Solution

หากเราสังเกตโครงสร้างของปัญหานี้ด้วยการใช้แนวคิดจากปัญหาย่อยที่ 2 และ 3 เราจะสามารถตรวจสอบได้ว่าเราเลือกเส้นส้มได้ทั้งหมดกี่เส้นด้วยอัลกอริทึมอย่าง Interval Scheduling[1] บนความชันความชันเริ่มต้นและความชันสิ้นสุดได้ และได้ว่าจำนวนเส้นส้มที่น้อยที่สุดที่ต้องถอนจะเท่ากับ N ลบด้วยจำนวนเส้นส้มที่มากที่สุดที่สามารถเลือกได้ ในปัญหานี้ได้ถูกออกแบบให้สามารถใช้ตัวแปรประเภท double เก็บข้อมูลได้ (การใช้ float จะทำให้คำตอบที่ผิด) อย่างไรก็ตามหากมีการเปรียบเทียบที่มีความแม่นยำมากกว่านี้ควรเก็บตัวแปรเป็น long long ของจำนวนเต็มและใช้วิธีการคูณไขว้เพื่อเทียบความชันในรูปเศษส่วนแทน โดยที่

$$\frac{A}{B} \leq \frac{C}{D} \iff AD \leq CB$$

เมื่อเรา assume ว่า $B, D > 0$

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

[1] ข้อมูลเพิ่มเติมของ interval scheduling สามารถศึกษาได้ที่ https://en.wikipedia.org/wiki/Interval_scheduling

Solution Code

```cpp
#include<bits/stdc++.h>
#define pii pair<int, int>
#define pII pair<pii, pii>
#define pll pair<long long, long long>
#define ll long long
#define ld long double
#define st first
#define nd second
#define pb push_back
#define INF DBL_MAX
#define sz(x) (int)x.size()
using namespace std;

vector<pair<double, double>> v;

void solve() {
    int N; cin >> N;
    int ans = 0;
    for (int i = 1; i <= N; ++i) {
        int x1, y1, x2, y2; cin >> x1 >> y1 >> x2 >> y2;
        double s1 = (x1==0?INF:(double)y1/x1);
        double s2 = (x2==0?INF:(double)y2/x2);
        if (s1 < s2) swap(s1, s2);
        v.pb({s1, s2});
    }
    sort(v.begin(), v.end());
    double prev = -1e18;
    for (int i = 0; i < N; ++i) {
        if (v[i].nd <= prev) ans++;
        else prev = v[i].st;
    }
    cout << ans << '\n';
}

int main() {
    ios_base::sync_with_stdio(0); cin.tie(NULL);
    int t = 1;
//    cin >> t;
    while (t--) {
        solve();
    }
}
```

นักท่องเที่ยว (tourist) (100 คะแนน)

1.5 seconds, 512 megabytes

ผู้แต่ง: M-W

เนื้อหาที่ใช้: Disjoint Set Union, Brute Force, Offline Query

Subtask 1 (9 คะแนน) $N, M, K, Q \leq 1\,000$, $L[i] \leq 1$, $R[i] \leq K$

ในปัญหาย่อยนี้ สังเกตว่า $L[i] = 1$ และ $R[i] = K$ ทำให้เราสามารถพิจารณาเส้นเชื่อมเป็นเพียงแค่ผ่านได้หรือผ่านไม่ได้เท่านั้น กล่าวคือเมื่อมีคำสั่งรูปแบบที่ 1 บนถนนใด ๆ ก็ตาม เราจะพิจารณาว่าถนนเส้นดังกล่าวไม่สามารถผ่านได้เลยสำหรับนักท่องเที่ยวทุก ๆ กลุ่ม

เนื่องจาก N, M, Q มีค่าน้อย จึงทำให้เราสามารถตรวจสอบการเดินทางหากันของเมืองสองเมืองได้โดยการ breadth-first search โดยตรง หากทั้งสองเมืองสามารถเดินทางหากันได้ให้เราตอบจำนวนกลุ่มนักท่องเที่ยวทั้งหมด และหากไม่ได้ให้ตอบ 0 ทั้งนี้จำนวนกลุ่มนักท่องเที่ยวทั้งหมดสามารถคำนวณได้หลายวิธี ในเบื้องต้นอาจใช้ `std::set` ในการนับจำนวนหมายเลขที่แตกต่างกันทั้งหมดก็ได้เช่นกัน

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

Subtask 2 (19 คะแนน) $N, M, K, Q \leq 25\,000$, $L[i] = 1$, $R[i] = K$

ข้อสังเกต #1 การที่เส้นเชื่อมค่อย ๆ ถูกปิดไม่ให้ผ่านก็เหมือนกับการเปิดเส้นเชื่อมให้ค่อย ๆ ผ่านได้บนการคำนวณแบบย้อนกลับบนคำสั่งทั้งหมด

การคำนวณในรูปแบบนี้สามารถช่วยให้แก้โจทย์ได้ง่ายขึ้น เพราะเราสามารถนำโครงสร้างข้อมูล Disjoint Set มาใช้ในการ Union สถานีเข้าด้วยกันได้ และการตรวจสอบการเดินทางหากันสามารถดูจาก component ของสถานีที่เชื่อมกันได้โดยตรง

ดังนั้น การแก้ปัญหาย่อยนี้สามารถทำได้โดยการเชื่อมทางรถไฟทั้งหมดที่ไม่ถูกปิดตลอด Q คำสั่ง และประมวลผลคำสั่งจากหลังมาหน้าด้วยการทยอยเชื่อมทางรถไฟและตอบคำถามด้วยการใช้ Disjoint Set Union ที่ได้กล่าวไว้ข้างต้น

อย่างไรก็ตาม คำสั่งปิดทางรถไฟอาจเกิดขึ้นหลายครั้งบนทางรถไฟเดียวกันได้ เราจึงต้องสร้างอาเรย์เพื่อเก็บค่าไว้ตรวจสอบสถานะการเปิดของทางรถไฟด้วย (ตัวอย่างเช่น ทางรถไฟหมายเลข 1 ถูกปิด 5 ครั้งตลอดทั้ง Q คำสั่ง เราต้องเก็บ $closed[1] = 5$ และทุกครั้งที่ผ่านคำสั่งปิดทางรถไฟหมายเลข 1 จากการประมวลผลย้อนกลับ เราจะค่อย ๆ ลดค่า $closed[1]$ ลงไปทีละหนึ่ง แล้วค่อยทำการเชื่อมทางรถไฟเมื่อ $closed[1] = 0$)

Time Complexity: $\mathcal{O}((M + Q) \log N)$

Subtask 3 (11 คะแนน) $N, M, K, Q \leq 25\,000$, $R[i] = K$, $S[i] = 1$, และคำสั่งรูปแบบที่ 1 ทั้งหมดมาก่อนคำสั่งรูปแบบที่ 2

สังเกตว่าเงื่อนไข $R[i] = K$ ทำให้เราทราบว่ากลุ่มนักท่องเที่ยวที่มีคนมากกว่าหรือเท่ากับ $L[i]$ จะไม่สามารถเดินทางผ่านทางรถไฟดังกล่าวได้เลย เนื่องจากว่าคำสั่งรูปแบบที่ 1 ทั้งหมดมาก่อนคำสั่งรูปแบบที่ 2 ทำให้เราสามารถประมวลผลคำสั่งบนทางรถไฟทั้งหมดก่อนหาคำตอบได้ ซึ่งเราจะสามารถ label น้ำหนักบนทางรถไฟที่ i เป็น $w[i] = \min(L[i]$ ของคำสั่งทั้งหมดบนทางรถไฟ $i)$ และ maximize ค่า $\min(w[i]$ บน path) ของทุก ๆ path จาก $S[i]$ ไป $E[i]$

นอกจากนี้ เนื่องจาก $S[i] = 1$ เราสามารถมองปัญหาอยู่ในรูปของ single source shortest path ได้ โดยใช้ Dijkstra's Algorithm โดยคำนวณ $distance[i] = \max(\min(distance[x], w(x, i)$ สำหรับทุก ๆ สถานี x ที่ติดกับ $i))$ เมื่อ $w(x, i)$ คือน้ำหนักทางรถไฟจาก x ไป i

Time Complexity: $\mathcal{O}(M \log N + Q)$

Subtask 5 (17 คะแนน) $N, M, K, Q \leq 25\,000$, $A[i] \leq 100$

ความน่าสนใจของปัญหาย่อยนี้คือเรามีค่า $A[i]$ ที่แตกต่างกันไม่เกิน 100 ค่า ซึ่งเป็นจำนวนที่เพียงพอต่อการคำนวณ brute force บนกลุ่มของนักท่องเที่ยวที่เป็นไปได้ทั้งหมด ในการแก้ปัญหาย่อยนี้จึงสามารถทำได้ด้วยการพิจารณานักท่องเที่ยวแต่ละกลุ่มเพื่อดูว่าคำสั่งการห้ามผ่านทางรถไฟครั้งใดมีผลต่อนักท่องเที่ยวกลุ่มดังกล่าวบ้าง (จำนวนนักท่องเที่ยวในกลุ่มดังกล่าวอยู่ระหว่าง $L[i]$ และ $R[i]$) และส่งผลให้เราสามารถแก้ปัญหาที่เหลือได้ในลักษณะเดียวกันกับปัญหาย่อยที่ 2

ทั้งนี้เราจะเก็บคำตอบไว้ในรูปของอาเรย์ โดยหากกลุ่มนักท่องเที่ยวบน $A[i]$ ค่าหนึ่งสามารถเดินทางบนคำสั่งรูปแบบที่ 2 ได้ เราจะบวกหนึ่งเข้าไปในช่องคำตอบของคำถามดังกล่าว แล้วจึงตอบคำถามทุกคำถามจากอาเรย์ดังกล่าวในตอนจบ

Time Complexity: $\mathcal{O}((M + Q) \times 100)$

Official Solution

ข้อสังเกต #2 กลุ่มนักท่องเที่ยวที่มีจำนวนคนเท่ากันจะมีคุณสมบัติในการคำนวณคำตอบเหมือนกันทุกประการ กล่าวคือสำหรับคำสั่งประเภทที่ 1 กลุ่มใด ๆ ที่มีจำนวนคนเท่ากันจะมีสถานะความสามารถการผ่าน / ไม่ผ่านทางรถไฟที่เหมือนกัน ส่งผลให้โครงสร้าง Disjoint Set มีลักษณะเหมือนกัน

ข้อสังเกต #3 หากพิจารณาจากข้อสังเกตรูปแบบที่ 2 แล้ว จะทราบว่ามีกลุ่มนักท่องเที่ยวที่มีจำนวนสมาชิกแตกต่างกันไม่เกิน $\sqrt{2K}$ กลุ่ม โดยเราสามารถพิสูจน์เบื้องต้นจากการพิจารณา worst case scenario ซึ่งก็คือการที่เรามีสมาชิกในกลุ่มนักท่องเที่ยวของแต่ละกลุ่มแตกต่างกันทั้งหมด

หากเราทราบว่ามีนักท่องเที่ยวที่มีสมาชิกแตกต่างกันทั้งหมด X กลุ่ม กลุ่มที่ 1 มีสมาชิก 1 คน, กลุ่มที่ 2 มีสมาชิก 2 คน, ... , จนถึงกลุ่มที่

X มี X คน จะได้ว่าจำนวนนักท่องเที่ยวทั้งหมดคือ

$$1 + 2 + \ldots + X = \frac{X(X+1)}{2}$$

แต่เราทราบว่าจำนวนนักท่องเที่ยว K ในโจทย์มีค่าไม่เกิน $100\,000$ จึงได้ว่า

$$\frac{X(X+1)}{2} \approx \frac{X^2}{2} \leq K \iff X \leq \sqrt{2K}$$

ดังนั้น หากเราจำแนกกลุ่มนักท่องเที่ยวตามจำนวนสมาชิกและคำนวณในลักษณะเดียวกับปัญหาย่อยที่ 5 จะรับประกันว่าโปรแกรมของเราจะวนลูปการคำนวณคำถามทั้งหมดไม่เกิน $\sqrt{2K}$ อย่างแน่นอน

ทั้งนี้ ปัญหาย่อยที่ 6 มีไว้สำหรับโค้ดคำตอบบางกรณีที่ทำถูกวิธีแล้วแต่ใช้เวลาในการรันนานกว่าปกติ ซึ่งสามารถแก้ไขได้ด้วยการลองสลับ index i และ j ในการเรียกอาเรย์สองมิติ (หากมี)

Time Complexity: $\mathcal{O}((M+Q)\log N \times \sqrt{K})$

หมายเหตุ Solution code ของข้อนี้มีวิธีการ implement ที่แตกต่างจากคำอธิบายเล็กน้อย โดยเปลี่ยนจากการวนทุกคำสั่ง $\sqrt{K}$ รอบ เป็นการวนดูนักท่องเที่ยวทุกกลุ่มสำหรับแต่ละคำสั่งแทน ซึ่งให้ผลลัพธ์และ Time complexity เดียวกัน โดยเราจำเป็นต้องทำ **coordinate compression** บนจำนวนสมาชิกของกลุ่มนักท่องเที่ยวเพื่อให้สามารถใช้อาเรย์ขนาดไม่เกิน $N\sqrt{K}$ ได้

Solution Code

```cpp
#include <bits/stdc++.h>
using namespace std;
int tour_group[100100], szidx[100100], gcnt[100100];
int pa[100100][320], blocked[100100][320];
vector<pair<int, int>> mark[100100], edge;
vector<int> allsz;
bitset<100100> szchk;

int findpa(int i, int a){
  return pa[a][i] == a ? a : pa[a][i] = findpa(i, pa[a][i]);
}

void union_range(int p, int l, int r){
  // Union edge #p for all group from index l to r
  int i = lower_bound(allsz.begin(), allsz.end(), l) - allsz.begin();
  for(; i < allsz.size() && allsz[i] <= r; i++){
    blocked[p][i]--;
    if(blocked[p][i]) continue;
    int gsz = allsz[i];

    auto [u, v] = edge[p];
    int pa_u = findpa(i, u), pa_v = findpa(i, v);
    if(pa_u != pa_v) pa[pa_u][i] = pa_v;
  }
}

int main(int argc, char* argv[]){
  int N, M, K, Q;
  scanf("%d %d %d %d", &N, &M, &K, &Q);
  // Collect how many tour groups and how many people,
  // Guaranteed to be lesser than sqrt(K)
  for(int i = 0, A; i < K; i++){
    scanf("%d", &A); tour_group[A]++;
  }
  for(int i = 1; i <= 100000; i++){
    if(!szchk[tour_group[i]]){
      szchk[tour_group[i]] = 1;
      allsz.push_back(tour_group[i]);
    }
    if(tour_group[i]) gcnt[tour_group[i]]++;
  }
  sort(allsz.begin(), allsz.end());
  // Construct initial graph
  edge.push_back({0, 0});
  for(int i = 1, u, v; i <= M; i++){
    scanf("%d %d", &u, &v); edge.push_back({u, v});
  }
  // Collect query
  vector<array<int, 4>> query;
  for(int i = 1; i <= Q; i++){
    int q; scanf("%d", &q);
    if(q == 1){
      int p, l, r; scanf("%d %d %d", &p, &l, &r);
      query.push_back({q, p, l, r});
      mark[p].push_back({l, r});
```

```
    }
    else{
      int s, e; scanf("%d %d", &s, &e);
      query.push_back({q, s, e, 0});
    }
  }
  // Add blocked layers for blocked edges
  for(int i = 1; i <= M; i++){
    for(auto [l, r] : mark[i]){
      int lidx = lower_bound(allsz.begin(), allsz.end(), l) - allsz.begin();
      int ridx = upper_bound(allsz.begin(), allsz.end(), r) - allsz.begin();
      blocked[i][lidx] += 1;
      blocked[i][ridx] -= 1;
    }
  }
  // Init DSU
  for(int i = 1; i <= N; i++)
    for(int j = 1; j < allsz.size(); j++) pa[i][j] = i;

  for(int i = 1; i <= M; i++){
    for(int j = 1; j < allsz.size(); j++){
      blocked[i][j] += blocked[i][j - 1];
      // Union all edges that can travel by default
      if(!blocked[i][j]){
        auto [u, v] = edge[i];
        pa[findpa(j, u)][j] = findpa(j, v);
      }
    }
  }
  // Start answering query from back to front
  stack<int> ans;
  for(int i = query.size() - 1; i >= 0; i--){
    int q = query[i][0];
    if(q == 1){
      int p = query[i][1], l = query[i][2], r = query[i][3];
      union_range(p, l, r);
    }
    else{
      int s = query[i][1], e = query[i][2], total_cnt = 0;
      for(int j = 1; j < allsz.size(); j++){
        if(findpa(j, s) == findpa(j, e))
          total_cnt += gcnt[allsz[j]];
      }
      ans.push(total_cnt);
    }
  }
  // Print all answers
  while(!ans.empty()){
    printf("%d\n", ans.top()); ans.pop();
  }
  return 0;
}
```