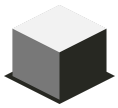
CCC beta #10: Welcome 2017

Editorial



A - Welcome to 2017 by Jumpwmk

สำหรับข้อนี้มีสิ่งที่ต้องพิจารณาสองอย่าง

1. การหาตัวที่ใกล้เคียงค่า 2017 มากที่สุด

ในการแก้นั้นให้ลองไล่ค่าระยะห่างจาก 2017 โดยให้เริ่มจาก 0 ไปเรื่อยๆจนกระทั่งเจอเลขที่ใกล้ที่สุด หรือมั่นใจว่าไม่เหลือเลขให้พิจารณา(ในกรณีนี้ให้ตอบ -1)

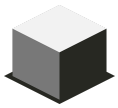
2. เลขนั้นต้องมีเพียงแค่ตัวเดียวเท่านั้น (สำหรับ Large)

ในส่วนนี้เราสามารถใช้ Array ในการนับค่าเลขแต่ละตัวเก็บไว้ แล้วพิจารณาเฉพาะเลขที่นับได้ครั้งเดียวเท่านั้น

Time Complexity: O(T); T คือระยะห่างที่มากที่สุดที่เป็นไปได้

Solution Code :

```
#include <stdio.h>

const int MX = 5010;

int count[ MX ];
int n;

int main( void ) {

	// read input
	scanf( "%d", &n );
	for ( int i = 0 ; i < n ; ++i ) {
		int val;
		scanf( "%d", &val );
		++count[ val ];
	}

	// iterate i from 0 to LIMIT ( in this case 2017 ) until a solution is found
	for ( int i = 0 ; i <= 2017 ; ++i ) {
		// this task requires checking 2017 - i before 2017 + i
		if ( count[ 2017 - i ] == 1 ) {
			printf( "%d\n", 2017 - i );
			return 0;
		}
		if ( count[ 2017 + i ] == 1 ) {
			printf( "%d\n", 2017 + i );
			return 0;
		}
	}

	// no solution found
	printf( "-1\n" );

	return 0;
}
```



B - Hacker by Piwattz

ในการแก้ปัญหาข้อนี้นั้นสิ่งที่ต้องทำคือ

1. ในแต่ละบริษัท ให้เรียงโปรแกรมเมอร์ตามค่าความสามารถจากน้อยไปมาก

ในช่วงนี้น่าจะมีปัญหามากที่สุดโดยหลักๆ มีสองส่วนที่ต้องทำคือ

- การจอง Array - ในส่วน Small สามารถจอง Array สองมิติโดยไม่มีปัญหาแต่ถ้ากรณี Large ต้องจอง Array ตามขนาดเอง (วิธีลองดูได้ใน solution code)
- การเรียงเลข - ถ้าจะแก้ Small ใช้ Sorting algorithm ใดก็ได้ในการแก้ แต่ถ้า Large ต้องใช้ Sorting algorithm ที่ running time เป็น O(N log N) อย่างเช่น Merge sort, Quick sort หรือ Heap sort (ใน solution code จะใช้ std::sort เป็น library ของ c++11 ซึ่งข้างในจะเป็น Quick sort)

2. ในการแข่งขันคนที่ลงแข่งรอบที่ i ก็คือคนที่อยู่ในลำดับที่ i ของทุกบริษัทเราก็หาคนที่มีความสามารถสูงสุดในนั้น

ในส่วนนี้สามารถคำนวณค่ามากที่สุด ตรงๆโดยต้องเก็บตำแหน่งไว้ด้วย

3. หาบริษัทที่มีโปรแกรมเมอร์ที่ถูกจ้างมากที่สุด

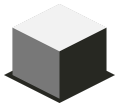
ในส่วนนี้ก็หาค่ามากที่สุดคล้ายกับส่วนที่ 2

Time Complexity: Small - O($NMlogM$)

Solution Code :

```
/* NOTE : even some part of this code is an actual c code, this code still must be
compiled as c++ code */
#include <stdio.h>
#include <malloc.h>
#include <algorithm>
#include <vector>

using namespace std;

const int N = 1000010;

int hire[ N ];
int n, m;

int main( void ) {

	// read input
	scanf( "%d %d", &n, &m );

	/* memory allocation */

	// c
	int **arr;
	arr = (int**)malloc( sizeof( int* ) * n );
	for ( int i = 0 ; i < n ; ++i ) {
		arr[i] = (int*)malloc( sizeof( int ) * m );
	}

	// c++
	// int **arr = new int*[n];
```

```
        // for ( int i = 0 ; i < n ; ++i ) {
        //      arr[i] = new int[m];
        // }

        // c++ with standard library
        // vector< vector< int > > arr( n, vector< int >( m, 0 ) );

        // read rest of input
        for ( int i = 0 ; i < n ; ++i ) {
                for ( int j = 0 ; j < m ; ++j ) {
                        scanf( "%d", &arr[i][j] );
                }
        }

        /* apply sorting algorithm */
        for ( int i = 0 ; i < n ; ++i ) {
                sort( arr[i], arr[i] + m );
                // if using vector
                // sort( arr[i].begin(), arr[i].end() );
        }

        // iterate from 0 to number of rounds
        for ( int j = 0 ; j < m ; ++j ) {

                // find max
                int mx = 0;
                for ( int i = 1 ; i < n ; ++i ) {
                        if ( arr[i][j] > arr[mx][j] ) {
                                mx = i;
                        }
                }

                ++hire[ mx ];
        }

        // NOTE : This code uses 0 based indexing. The answer will be increased by 1
before being displayed
        int ans = 0;
        for ( int i = 1 ; i < n ; ++i ) {
                if ( hire[i] > hire[ ans ] ) {
                        ans = i;
                }
        }

        printf( "%d\n", ans+1 );

        return 0;
}
```



C - Great Wall by Piwattz

ข้อนี้เริ่มที่จะไม่ตรงไปตรงมาแล้ว ในการที่จะแก้ต้องลองทำปัญหาให้ง่ายขึ้น
ถ้าเกิดปัญหามีการกำหนดค่าความสูงที่กำแพงทุกอันห้ามเกิน h เราสามารถแก้ข้อนั้นได้ง่ายขึ้น (เทคนิคนี้เป็นการเปลี่ยน optimization problem ให้กลายเป็น decision problem)

ถ้าเกิดความสูงของกำแพงต้องไม่เกิน h เวลาแก้เราสามารถใช้ Greedy algorithm ในการแก้ได้
นั้นคือลองดูทุกตำแหน่งจากซ้ายไปขวาถ้าความสูงของกำแพงนั้นสูงเกิน h
เราทำการลดความสูงตั้งแต่จุดไปเรื่อยๆนั้นกระทั่งความสูงจุดนั้นไม่เกิน h (การลดความสูงนั้นจะส่งผลต่อ W ตัวที่ติดกัน)
โดยเราสามารถบอกว่าสามารถทำให้ทุกจุดความสูงไม่เกิน h ได้ก็ต่อเมื่อจำนวนครั้งในการทำลายกำแพงไม่เกิน K

ในขั้นตอนการลดความสูงนั้นถ้ากรณี Small เราสามารถค่อยๆลดความสูงทีละจุดทั้ง W จุดได้ แต่ถ้ากรณี Large เราต้องทำให้เร็วขึ้นโดยเทคนิคที่ใช้คือ Partail sum (ส่วนใหญ่จะรู้จักในชื่อ Quick sum)
โดยการลดช่วงที่ติดกันนั้นทำได้ใน O(1)

โดยขั้นตอนสุดท้ายที่ต้องหาให้ได้คือค่า h ที่ต่ำที่สุดที่เราสามารถลดความสูงให้ทุกจุดสูงไม่ให้เกิน h ได้
ถ้าในกรณี Small การไล่ค่า h ตั้งแต่ 0 ขึ้นไปเรื่อยๆ แต่กรณี Large นั้นต้องสังเกตว่า

- ถ้า h เท่ากับความสูงที่สูงที่สุดนั้น เราสามารถลดความสูงทุกอันให้ไม่เกิน h นั้นได้(นั่นคือไม่ต้องทำอะไรเลย)
- ถ้า h เป็นคำตอบที่เราต้องการ เราไม่สามารถทำความสูงทุกจุดนั้นน้อยกว่าหรือเท่ากับ h' โดยที่ $h' < h$ ได้อย่างแน่นอน
- ถ้า h เป็นคำตอบที่เราต้องการ เราสามารถทำความสูงทุกจุดนั้นน้อยกว่าหรือเท่ากับ h' โดยที่ $h' > h$ ได้อย่างแน่นอน

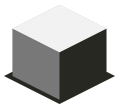
ถ้าเกิดเราเขียนค่าความจริงออกมาสำหรับทุกๆค่า h น่าตาที่มันออกมาจะเป็นประมาณ < 0, 0, …, 0, 1, 1, 1, …, 1 > แทนที่เราจะลองไล่ไปเรื่อยๆ เราจะใช้เทคนิค Binary Search เข้าช่วยได้ประหยัดเวลาได้มาก

Time Complexity: $O(N\log(max\ H_i))$

Solution Code :

```
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

const int N = 100010;

long long sum[ N ], H[ N ];
int n, w, k;

// return whether the input can be
```

```cpp
bool check( int h );

int main( void ) {

	// read input
	cin.sync_with_stdio( false );
	cin >> n >> w >> k;
	for ( int i = 0 ; i < n ; ++i ) {
		cin >> H[i];
	}

	// Small - NOTE: answer is h
	// int h = 0;
	// while ( !check(h) ) {
	//	++h;
	// }
	// cout << h << endl;

	// Large - NOTE: answer is lo
	int maxH = *max_element( H, H + n );
	int lo = 0, hi = maxH;
	while ( lo < hi ) {
		int mid = ( lo + hi ) / 2;
		if ( check( mid ) ) {
			hi = mid;
		} else {
			lo = mid + 1;
		}
	}
	cout << lo << endl;

	return 0;
}

bool check( int h ) {

	// reset sum
	fill( sum, sum + n, 0 );

	long long numWallBreaking = 0ll;
	for ( int i = 0 ; i < n ; ++i ) {

		// fix quick sum
		sum[i] += sum[i-1];

		// in case that the height is more than the limit
		if ( H[i] - sum[i] > h ) {

			// break walls H[i] - h times
			long long breakingRequired = H[i] - sum[i] - h;
			numWallBreaking += breakingRequired;

			// update quick sum
			sum[i] += breakingRequired;
			if ( i + w < n ) sum[ i + w ] -= breakingRequired;
		}
	}

	// it's valid if number of required wall breaking is not more than k
	return numWallBreaking <= k;
}
```



D - New Year Gift by Jumpwmk & PeaTT~

ข้อนี้โค้ดง่ายแต่ที่มาค่อนข้างวุ่นอยู่ ขอใช้นิยามสิ่งต่อไปนี้เพื่อความง่าย

- node คือ จุด, edge คือ เส้นเชื่อม
- edge ที่ตัดแล้ว tree แบ่งเป็นสองส่วนเรียกว่า bridge โจทย์คือเพิ่มเส้นเชื่อมให้ไม่มี bridge ลงเหลืออยู่
- leaf คือ node ที่ degree(จำนวนเส้นเชื่อมที่ต่อกับจุดนั้น) เท่ากับ 1
- cycle คือ set ของ edge ที่ต่อเป็นวงกลม(ทุก node มี degree เท่ากับ 2)

วิธีคือให้หยิบ root หนึ่งตัวที่ไม่ใช่ leaf ทำการ DFS (Depth-first search) ให้เก็บ leaves ทุกใบตามลำดับที่เจอเป็น list หนึ่งต้วให้เป็น L ความยาว l โดยเชื่อม L_i กับ $L_{i+floor(l/2)}$ (มีพิสูจน์หลัง solution code) โดยสำหรับ Small คำตอบคือ $ceil(l/2)$

Solution Code :

```
#include <iostream>
#include <vector>

using namespace std;

const int N = 210;

typedef pair< int, int > pii;

vector< int > order;
vector< pii > ans;

vector< int > adj[ N ];
int n, m;

// dfs find all leaves
void dfs( int u, int p ) {

	// if the node is a leaf
	if ( adj[u].size() == 1 ) {
		order.push_back( u );
	}

	for ( int v : adj[u] ) {
		if ( v != p ) {
			dfs( v, u );
		}
	}
}
int main( void )  {

	// read input
	cin >> n >> m;
	for ( int i = 0 ; i + 1 < n ; ++i ) {
		int u, v;
		cin >> u >> v;
		adj[u].push_back( v );
		adj[v].push_back( u );
	}
```

```
        if ( n == 2 ) {
                // check for special case; no root according to the definition
                ans = { { 1, 2 } };
        } else {

                // find root
                int r = 1;
                while ( adj[r].size() == 1 ) {
                        ++r;
                }

                // start DFS
                dfs( r, r );

                int len = ( order.size() + 1 ) / 2;
                // connect the ith leaf with (i+len/2)th leaf
                for ( int i = 0 ; i < len ; ++i ) {
                        ans.push_back( { order[i], order[ order.size() - len + i ] } );
                }
        }

        // report answer
        cout << ans.size() << "\n";
        if ( m == 2 ) {
                for ( pii e : ans ) {
                        cout << e.first << " " << e.second << "\n";
                }
        }

        return 0;
}
```

Proof of Correctness (ค่อยๆไปทีละข้อ ไม่ได้ยากขนาดนั้น)
Observation 1 การเชื่อม node u, v ทำให้ทุก edge ใน path จาก u ไป v ไม่เป็น bridge อย่างแน่นอน เนื่องจากการเชื่อม node u, v ทำให้เกิด cycle ใน edge เหล่านั้น(ทุก edge ใน cycle ไม่ใช่ bridge)

Observation 2 การเชื่อมที่ลด bridge ได้ดีที่สุดต้องเชื่อมจาก leaf ไป leaf เท่านั้น เนื่องจาก ถ้าไม่ได้เลือก leaf เราสามารถเลือก leaf ที่ทำให้ path นั้นยาวขึ้นและลด bridge ได้ดีขึ้น

Observation 3 ไม่สามารถหาคำตอบที่ดีกว่า optimal($ceil(l/2)$) ได้ เนื่องจากถ้าต่ำกว่านั้นจะมี leaf อย่างน้อยหนึ่งตัวที่ยังไม่มีเส้นเชื่อมเพิ่มโดยสามารถสรุปได้ว่าเส้นเชื่อมที่เชื่อมกับ leaf นั้น(มีเพียงเส้นเดียว)เป็น bridge อย่างแน่นอน

นิยามต่อไปนี้สำคัญต่อ Observation ที่เหลือ
ให้ $v_e \subset L$ โดยที่เป็น set ของทุก leaf ที่ path ตั้งแต่ root ถึง leaf นั้นผ่านedge e

Observation 4 ไม่มี v_e ที่เท่ากับ L (ลองสังเกตดู)

Observation 5 ถ้าพิจรณาตำแหน่งทุกตัวของ v_e สังเกตได้ว่าเป็นช่วงที่ติดกันใน L เนื่องจาก L เป็น list ที่เกิดจากการ DFS เราจึงแทน v_e ด้วย a_e กับ b_e แทนตำแหน่งใน L
NOTE: จาก Observation 4 สรุปได้ว่าไม่มีกรณีที่ $a_e = 1$ และ $b_e = l$ พร้อมกัน

Observation 6 ถ้าการเชื่อม leaf ที่ x^{th} ของ L กับ y^{th} ของ L ทำให้ edge e ไม่กลายเป็น bridge ก็ต่อเมื่อ x อยู่ระหว่าง a_e กับ b_e หรือ y อยู่ระหว่าง a_e กับ b_e อย่างใดอย่างหนึ่งเท่านั้น
สังเกตว่าถ้าเกิด leaf ตัวหนึ่งผ่าน e โดยจากการเดินจาก root และอีกตัวไม่ผ่าน e แสดงว่า path นั้นต้องผ่าน e อย่างแน่นอน

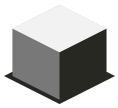
ณ จุดนี้สิ่งที่ต้องพิสูจน์คือการเชื่อมจุดนั้นทำให้ไม่เหลือ bridge
สังเกตว่าการจากการเชื่อมของเราคือเชื่อมตัวที่ 1, 2, 3, … กับ 1+l/2, 2+l/2, 3+l/2, … ตามลำดับ

Observation 7 ทุกๆ edge e ที่ a_e กับ b_e นั้น**ไม่ผ่าน**จุดกึ่งกลาง ของ L สรุปได้ว่า e นั้นไม่เป็น bridge อย่างแน่นอน เนื่องจาก ทุก leaf ระหว่าง a_e กับ b_e นั้นถูกเชื่อมกับ leaf อีกฝั่งของ L ถ้าเกิด edge e คลุมเพียงฝั่งเดียว leaf เหล่านั้นจะไม่ถูกคลุมโดย e สรุปได้ว่า e นั้นไม่เป็น bridge ตาม observation 6

Observation 8 ทุกๆ edge e ที่ a_e กับ b_e นั้น**ผ่าน**จุดกึ่งกลาง ของ L สรุปได้ว่า e นั้นไม่เป็น bridge อย่างแน่นอน ถ้าเกิด e นั้นผ่านจุดกึ่งกลางแสดงว่าต้องคลุม 2 leaf ตรงขอบแน่ๆ
ถ้าการที่ e เป็น bridge ได้นั้นแสดงว่า

- e ต้องคลุม leaf ตัวที่ 1 เพราะว่าตัวแรกของฝั่งขวานั้นถูกเชื่อมกับ leaf ตัวที่ 1 ถ้า e ไม่คลุมตัวนี้แสดงว่า e ไม่เป็น bridge ตาม observation 6
- e ต้องคลุม leaf ตัวสุดท้าย เพราะว่าตัวขอบฝั่งซ้ายนั้นถูกเชื่อมกับ leaf ตัวที่สุดท้าย ถ้า e ไม่คลุมตัวนี้แสดงว่า e ไม่เป็น bridge ตาม observation 6

แสดงว่า e ต้องคลุม leaf ตัวแรกกลับตัวสุดท้าย ซึ่งกลายเป็นว่าขัดแย้งกับ NOTE ใน Observation 5 ซึ่งทำให้สรุปได้ว่า e นั้นไม่ใช่ bridge อย่างแน่นอน



E - Stop the Conflict by phirasit

ข้อนี้เป็น Game Theory ที่ค่อนข้างแปลก

Small - ในกรณีนี้ ถ้าเกิดมีถนนจาก u ไป v ที่ไม่ว่า A กับ B ได้ไปก็ได้เงินเท่ากัน

Observation ตอนสุดท้าย A กับ B ได้จำนวนเมืองเท่ากัน เนื่องจากจำนวนเมืองเป็นคู่

สิ่งที่เราต้องทำคือทุกๆ ถนนให้เพิ่มถนนที่หน้าตาเหมือนกันเข้าไปกับทุกคู่ของเมืองเพราะไม่ว่า A กับ B เงินที่ได้จากถนนเส้นนี้จะต้องเท่ากันแน่นอน

โดยเพื่อไม่ให้เป็นการที่เติมถนนเยอะเกินไปคู่เมืองเดียวกันน่าจะเติมถนนไม่เกิน 1 เส้นเท่านั้นโดยรวมจะใช้ไม่เกิน 25000 เส้น

Large - อันนี้น่าสนใจมากๆ สังเกตจาก
หลักการที่ทุกคนจะเล่นให้แต่ที่มากกว่าอีกคนมากที่สุดเท่าที่เป็นไปได้(ไม่ใช่ให้เงินมากที่สุด)

ถ้าให้ M เป็นเงินที่ A ได้ลบเงินที่ B ได้ เกมจะกลายเป็น A ต้องให้ค่านี้มากสุด ส่วน B ต้องให้ค่านี้น้อยที่สุด

ตอนนี้เรามาพิจารณาเกมใหม่ ถนนหนึ่งเส้นนั้นเชื่อม u กับ v ด้วยค่า a_i กับ b_i ถ้าเราบอกว่าถ้า A เลือกสักเมืองจะได้เงิน $(a_i+b_i)/4 + (a_i-b_i)/4$ ส่วน B จะได้เงิน $(a_i+b_i)/4 - (a_i-b_i)/4$ ถนนจะไม่ได้เงินแล้วแต่จะเงินจากเมืองแทน โดยถ้า A เลือกเมืองที่มีค่ารวม x ให้เพิ่ม M ไป x และถ้า B เลือกเมืองที่มีค่ารวม x ให้ลด M ไป x

ข้อสังเกต ถ้า A ได้ทั้งเมืองเงินที่ A ได้นั้นค่า M เพิ่มขึ้นนั้นเท่ากับ a_i ถ้า B ได้ทั้งสองเมือง M จะลดเท่ากับ b_i ถ้า A กับ B ได้คนละเมืองค่า M ไม่เปลี่ยนแปลง สรุปได้ว่า M ของเกมใหม่นี้เท่ากับเกมเดิมเหมือนกันเลย

NOTE: พจน์หลัง $(a_i-b_i)/4$ ไม่ว่า A หรือ B จะได้ไปผลลัพธ์คือ M
ต้องเพิ่มขึ้นด้วยค่าเดียวกันเราสามารถคำนวณแยกเลย

ที่เหลือคือพจน์แรก $(a_i+b_i)/4$ ที่บวกเข้าทุก node strategy
ที่ทั้งสองคนเล่นคือเลือกเมืองที่ผลรวมค่าพจน์แรกสูงสุดเท่าที่เป็นไปได้ไปเรื่อยๆ

คราวนี้เราก็เรียงเมืองตามผลรวม $(a_i+b_i)/4$ แล้วเพิ่มมูลค่าเมืองในลำดับที่ 2, 4, 6, 8, … ไปให้มีค่าเท่ากับเมืองที่ 1, 3, 5, 7, … ตามลำดับโดยจะเพิ่มเพียง 100 เส้น

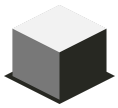
IMPOSSIBLE CASE

- ผลรวมค่า $(a_i-b_i)/4$ มากกว่า 0 การเพิ่มถนนของเราไม่เปลี่ยนค่านี้
- มีเมืองเป็นจำนวนคี่แล้วเมืองสุดท้ายในลำดับที่เรียงแล้วมีค่าไม่เท่ากับ 0

Solution Code :

```
#include <algorithm>
#include <iostream>
#include <iomanip>
#include <vector>

using namespace std;

const int N = 210;
const double EPS = 1e-6;

double score[ N ];
int idx[ N ];
int n, m, k;

bool cmpScore( int u, int v ) {
	return score[ u ] > score[ v ];
}

int main( void ) {

	double offset = 0.0;

	// read input
	cin >> n >> m >> k;
	for ( int i = 0 ; i < m ; ++i ) {

		int u, v;
		double a, b;
		cin >> u >> v >> a >> b;

		// update score
		score[ u ] += ( a + b ) / 4;
		score[ v ] += ( a + b ) / 4;

		offset += ( a - b );
	}

	for ( int i = 1 ; i <= n ; ++i ) {
		idx[ i ] = i;
	}

	// sort the indices into non-decreasing order
	sort( idx + 1, idx + n+1, cmpScore );

	// check for edge case
	if ( abs( offset ) > EPS or ( n % 2 == 1 and abs( score[ idx[ n ] ] ) > EPS ) ) {
		cout << -1 << endl;
		return 0;
	}

	vector< vector< double > > ans;
	int v = -1;

	for ( int u = 2 ; u <= n ; u += 2 ) {
		if ( score[ idx[u-1] ] > score[ idx[u] ] ) {
			if ( v == -1 ) {
				v = u;
			} else {
				double edge_weight = min( score[ idx[u-1] ] - score[ idx[u]
], score[ idx[v-1] ] - score[ idx[v] ] );
				ans.push_back( { (double) idx[u], (double) idx[v], 2 *
```

```
edge_weight } );
                                score[ idx[u] ] += edge_weight;
                                score[ idx[v] ] += edge_weight;
                                if ( abs( score[ idx[u-1] ] - score[ idx[u] ] ) > EPS ) {
                                        v = u;
                                } else if ( abs( score[ idx[v-1] ] - score[ idx[v] ] ) < EPS
) {
                                        v = -1;
                                }
                        }
                }
        }

        if ( v != -1 and score[ idx[v-1] ] > score[ idx[v] ] ) {
                double edge_weight = ( score[ idx[v-1] ] - score[ idx[v] ] );
                ans.push_back( { (double) idx[v], (double) idx[v], edge_weight } );
        }

        // output answer
        cout << setprecision(2) << fixed;
        cout << ans.size() << "\n";
        for ( vector<double>& data : ans ) {
                cout << (int)data[0] << " " << (int)data[1] << " " << data[2] << "\n";
        }

        return 0;
}
```



F - Flip the Cards by phirasit

Small - สามารถทำการ Brute Force O(NM) ได้ตรงๆ(ไม่ขอเจาะลึกวิธีนี้)

Large - ต้องลองพิจารณาไพ่ดีๆ ถ้าเกิดหน้าไพ่เป็น a, b ($w.l.o.g.\ a \le b$) ถ้าเกิดเลขที่คุณพูดเป็น x สิ่งที่เกิดขึ้นต้องเป็นหนึ่งในสามกรณีนี้

1. $x < a$ กรณีนี้ไม่ว่าหน้าไพ่เป็น a หรือ b ตอนสุดท้ายก็จะเป็นอย่างนั้นไม่เปลี่ยนแปลง
2. $a \le x < b$ กรณีนี้ไม่ว่าหน้าไพ่เป็นอย่างไร สุดท้ายก็จะกลายเป็น b
3. $b < x$ กรณีนี้ถ้าหน้าไพ่เป็น a สุดท้ายจะกลายเป็น b และถ้าสุดท้ายเป็น b สุดท้ายจะกลายเป็น a

แนวคิดข้อนี้นั่นคือถ้าหน้าไพ่เป็น $a,\ b$ ให้หา v_i ที่ i มากสุดที่ $min(a,b) \le v_i < max(a,b)$ สมมติให้เป็น k (โดยถ้าไม่มี k อยู่จริงก็ให้นับจำนวน v_j ที่ $max(a,b) \le v_j$ แล้วพลิกไพ่ไปเป็นจำนวนเท่ากับจำนวนนั้น) ในการแก้ส่วนนี้นั้นให้หา RMQ(Range Maximum Query) ในช่วง $min(a,\ b)$ กับ $max(a,b)$ ((ลองดูได้ใน solution code)

แต่ถ้ามี v_i ให้มองว่าไม่ว่าสิ่งที่พูดก่อน i จะเป็นอะไรหลังพูด v_i ไปหน้าไพ่นั้นจะเท่ากัน $max(a,b)$ เสมอ เพราะฉะนั้นก็ให้นับจำนวนตัวที่เหลือที่มากกว่า $max(a,b)$ แล้วพลิกหน้าไพ่ไปเท่านั้นครั้ง ในส่วนนี้เรารวมว่าไพ่แต่ละใบต้องถามช่วงในด้วยเลขเท่าไร(เป็นแนวคิดแบบ offline) แล้วใช้ Binary Indexed Tree (หรืออีกชื่อคือ Fenwick tree) โดยเราจะค่อยๆไล่ตำแหน่ง i จากมากไปน้อยในแต่ละรอบให้แก้ Fenwick tree ตำแหน่ง v_i แล้วก็หาคำตอบสำหรับไพ่ทุกใบที่ $i = k$ (ลองดูใน solution code ได้)

Solution Code :

```
#include <algorithm>
#include <iostream>
#include <map>
#include <set>
#include <vector>

using namespace std;

const int N = 50010;
const int LG = 20;

vector< vector< int > > query;
map< int, int > pos;
set< int > coor;

vector< int > front, back;
vector< int > flip;
int n, m;

// max RMQ
vector< vector< int > > RMQ;
void initRMQ( void );
int rmq( int, int ); // get max between two indices
```

```
// Fenwick Tree
vector< int > Fenwick;
void updateFenwick( int, int ); // update fenwick tree
int queryFenwick( int ); // query fenwick tree

int main( void ) {

	// read input
	cin >> n >> m;

	front.resize(n);
	back.resize(n);
	flip.resize(m);

	for ( int i = 0 ; i < n ; ++i ) {
		cin >> front[i] >> back[i];
	}
	for ( int i = 0 ; i < m ; ++i ) {
		cin >> flip[i];
	}

	// compress the coordinate
	for ( int i = 0 ; i < n ; ++i ) {
		coor.insert( front[i] );
		coor.insert( back[i] );
	}
	for ( int val : flip ) {
		coor.insert( val );
	}
	int cnt = 0;
	for ( int x : coor ) {
		pos[ x ] = cnt++;
	}

	// process max RMQ
	initRMQ();

	query.resize( m );
	for ( int i = 0 ; i < n ; ++i ) {
		auto card = minmax( { front[i], back[i] } );
		int idx = rmq( pos[ card.first ], pos[ card.second ] - 1 );
		if ( idx != -1 ) {
			front[i] = card.second;
			back[i] = card.first;
		}
		query[ max( idx, 0 ) ].push_back( i );
	}

	Fenwick.resize( coor.size() + 1, 0 );
	for ( int i = m-1 ; i >= 0 ; --i ) {
		updateFenwick( Fenwick.size() - pos[ flip[i] ], 1 );
		for ( int id : query[i] ) {
			auto card = minmax( { front[id], back[id] } );
			int tflip = queryFenwick( Fenwick.size() - pos[ card.second ] );
			if ( tflip % 2 == 1 ) {
				swap( front[id], back[id] );
			}
		}
	}

	// report answer
	for ( int i = 0 ; i < n ; ++i ) {
		cout << front[i] << "\n";
	}
```

```
	return 0;
}

void initRMQ( void ) {
	RMQ.resize( coor.size(), vector< int > ( LG, -1 ) );

	for ( int i = 0 ; i < m ; ++i ) {
		RMQ[ pos[ flip[i] ] ][ 0 ] = i;
	}

	for ( int j = 1 ; j < LG ; ++j ) {
		for ( int i = 0 ; i < coor.size() ; ++i ) {
			int nwi = i + ( 1 << (j-1) );
			RMQ[i][j] = max( RMQ[i][j-1], ( nwi < RMQ.size() ? RMQ[ nwi ][ j-1
] : -1 ) );
		}
	}
}
int rmq( int l, int r ) {
	if ( l > r ) return -1;
	int lg = log2( r - l + 1 );
	return max( RMQ[ l ][ lg ], RMQ[ r - ( 1 << lg ) + 1 ][ lg ] );
}

void updateFenwick( int idx, int val ) {
	while ( idx < Fenwick.size() ) {
		Fenwick[ idx ] += val;
		idx += idx & ( -idx );
	}
}
int queryFenwick( int idx ) {
	int ans = 0;
	while ( idx > 0 ) {
		ans += Fenwick[ idx ];
		idx -= idx & ( -idx );
	}
	return ans;
}
```