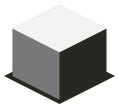
# CCC beta #9: Christmas

## Editorial



# Christmas Present by PalmPTSJ

สำหรับข้อนี้ ให้

- min1 คือตัวที่น้อยที่สุด
- min2 คือตัวที่น้อยที่สุดที่รองมาจาก min1

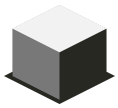
เราจะได้คำตอบคือตัวที่น้อยที่สุดของ

- min1(A) + min2(A) (ใช้ได้เมื่อ N > 1 เท่านั้น)
- min1(B) + min2(B) (ใช้ได้เมื่อ M > 1 เท่านั้น)
- min1(A) + min1(B) + 100

สำหรับการหา min เราสามารถ sort หรือจะไล่หาเก็บ min 2 ตัวก็ได้

Solution Code :

```
#include <bits/stdc++.h>
using namespace std;
int main() {
	int n,m;
	scanf("%d %d",&n,&m);
	int a[n];
	for(int i = 0;i < n;i++) scanf("%d",&a[i]);
	int b[m];
	for(int i = 0;i < m;i++) scanf("%d",&b[i]);

	sort(a,a+n);
	sort(b,b+m);

	int ans = 999999;
	if(n >= 2) ans = min(ans,a[0]+a[1]);
	if(m >= 2) ans = min(ans,b[0]+b[1]);
	ans = min(ans,a[0]+b[0]+100);

	printf("%d",ans);
}
```

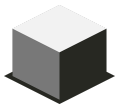


## Santa Claus Work by bT33

สำหรับข้อนี้เมื่อลองเขียนสมการเวียนเกิดออกมาจะได้เป็น $a_n = a_{n-1} + 3a_{n-1} = 4a_{n-1}$ เมื่อ $n > 1$และ $a_1 = 1$ แล้วเมื่อแก้สมการเวียนเกิดออกมาจะได้เป็น $a_n = 4^n$

Solution Code :

```
#include <iostream>

using namespace std;

int main()
{
	int n;
	cin >> n;
	cout << (1LL << (n + n));
	return 0;
}
```



# Xmas Beam by Peatt~

ก่อนอื่นต้องขอขอบคุณ phirasit สำหรับการทำ editorial ของข้อ Xmas Beam นี้

ตอนแรกเพื่อความง่ายเราจะเรียงดาวตามแกน X สมมุติให้เป็น (x1, y1), (x2, y2), (x3, y3), ..., (xN, yN) โดยที่ x1 <= x2 <= x3 <= ... <= xN ปัญหาที่ต้องทำก็เปลี่ยนไปเป็นให้เลือกช่วง a, b ที่ xb - xa น้อยที่สุดโดย max( ya, ya+1, ya+2, ..., yb ) - min( ya, ya+1, ya+2, ..., yb ) >= H

ในการแก้ปัญหาด้านบนให้เร็วเราจะใช้สิ่งที่เรียกว่า deque (double-ended queue: เป็น queue ที่ใส่หรือเอาออกได้ทั้งหัวและท้าย) กับเทคนิค sliding window (เป็นการเลื่อนขอบการค้นหา)

ในทุกๆ b ตั้งแต่ 1 ถึง N เราจะหาค่า a ที่มากที่สุดที่ max( ya, ya+1, ya+2, ..., yb ) - min( ya, ya+1, ya+2, ..., yb ) >= H เพราะว่าถ้า a มีค่ามาก ค่า xb - xa จะมีค่าน้อยลง (ข้อสังเกต ยิ่งค่า b เพิ่มขึ้นค่า a จะไม่มีทางลดลงอย่างแน่นอน (ลองพิสูจน์ดู) ) สิ่งที่เราทำคือไล่ตั้งแต่ b เท่ากับ 1 ไปถึง N ในระหว่างนั้นคอยเพิ่ม a จนกระทั่งได้ a ที่ max( ya, ya+1, ya+2, ..., yb ) - min( ya, ya+1, ya+2, ..., yb ) < H ระหว่างนั้นให้คำนวนค่า xb - xa โดยแสดงค่าที่น้อยที่สุดเป็นคำตอบ

ในการหาค่า max, min ให้รวดเร็ว ในส่วนนี้เราจะใช้ deque ช่วย
ในกรณีค่า max ตอนแรก deque จะว่างอยู่ โดยตอนที่เพิ่มค่า b ให้พิจรณาค่า yb กับท้าย deque(ถ้ามี) โดยถ้า yb นั้นมากกว่าตัวท้าย ให้นำตัวท้ายออกจนกระทั่งตัวท้ายมีค่ามากกว่า yb แล้วให้ใส่ค่า yb ไปที่ท้าย deque ในขณะเดียวกันตอนที่เพิ่มค่า a ให้ดูว่า ตัวแรกของ deque นั้นมี index มาก่อน a รึเปล่าถ้าใช่ให้เอาตัวนั้นออก ในที่สุดตัวแรกของ deque จะเป็น ค่า max ในช่วง a ถึง b
ส่วนค่า min จะทำคล้าย ๆ กัน (ลองดูใน solution code ก็ได้)

Running Time : O( N log N ) ต่อ test case

Solution Code :

```
#include <algorithm>
#include <iostream>
#include <deque>

using namespace std;

// define constants
const int N = 1e5 + 10;
const int inf = -1u / 2;

// **** these deques will store the indices of stars NOT the stars
deque< int > mxdeq, mndeq;

int idx[ N ];

int X[ N ], Y[ N ];
int q, n, H;
```

```
// return whether x-coordinate of star number id1 is less that id2's
int cmpX( int id1, int id2 ) {
	return X[ id1 ] < X[ id2 ];
}

int main( void ) {

	cin.sync_with_stdio( false );

	// read num of test cases
	cin >> q;
	while ( q-- ) {

		// read n, H, X, Y
		cin >> n >> H;
		for ( int i = 0 ; i < n ; ++i ) {
			cin >> X[i] >> Y[i];
		}

		for ( int i = 0 ; i < n ; ++i ) {
			idx[i] = i;
		}

		// sort the stars according to their x-coordinate
		sort( idx, idx + n, cmpX );
		// after this X[ idx[0] ] <= X[ idx[1] ] <= X[ idx[2] ] <= ... <= X[ idx[n-1] ]

		// clear both deques
		mxdeq = deque< int >();
		mndeq = deque< int >();

		// set ans to inf if ans is still inf by the end of process, it means the answer is -1
		int ans = inf;

		// set 'a' to 1 as a default
		int a = 0;

		// iterate b over 0 to n-1
		for ( int b = 0 ; b < n ; ++b ) {

			// insert star number idx[b] to the back of deques
			while ( !mxdeq.empty() and Y[ mxdeq.back() ] < Y[ idx[b] ] ) {
				// remove back of the deque
				mxdeq.pop_back();
			}
			mxdeq.push_back( idx[b] );

			// this is for min deque
			while ( !mndeq.empty() and Y[ mndeq.back() ] > Y[ idx[b] ] ) {
				// remove back of the deque
				mndeq.pop_back();
			}
			mndeq.push_back( idx[b] );

			// --finish insert step

			while ( a <= b and Y[ mxdeq.front() ] - Y[ mndeq.front() ] >= H ) {
				// if the subarray from a to b has max Y - min Y >= H

				// update answer
				ans = min( ans, X[ idx[b] ] - X[ idx[a] ] );

				// remove the front of each deque if its index is idx[a]
				if ( mxdeq.front() == idx[a] ) {
					mxdeq.pop_front();
```

```
                                }
                                if ( mndeq.front() == idx[a] ) {
                                        mndeq.pop_front();
                                }

                                // increase 'a'
                                ++a;
                        }
                }

                if ( ans == inf ) {
                        // if there is no valid answer
                        ans = -1;
                }

                cout << ans << endl;
        }


        return 0;
}
```

สำหรับโจทย์ข้อนี้ ให้คิดแยกทีละบิต โดยสังเกตจากตารางค่าความจริงจะพบว่า ที่บิตใด ๆ จะเป็น 1 ก็ต่อเมื่อ operand ทั้งคู่ต่างกัน

| A | B | A xor B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

ดังนั้น คำตอบของบิตใด ๆ จึงเป็นผลรวมของผลคูณของบิตที่ต่างกัน

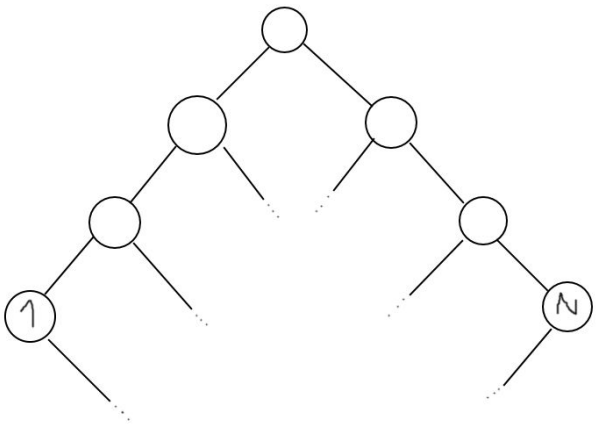
Time Comoplexity : O(N log A)

Solution Code :

```
#include <bits/stdc++.h>

using namespace std;
typedef long long ll;

ll a0[64], a1[64], b0[64], b1[64]; // count each bit

int main()
{
	// input A
	int n;
	scanf ("%d", &n);
	for (int i = 0; i < n; i++) {
		ll x;
		scanf ("%lld", &x);

		// count each bit of A
		for (int j = 0; j < 63; j++) {
			if (x & (1LL << j)) a1[j]++;
			else a0[j]++;
		}
	}

	// input B
	int m;
	scanf ("%d", &m);
	for (int i = 0; i < m; i++) {
		ll x;
		scanf("%lld", &x);

		// count each bit of B
```

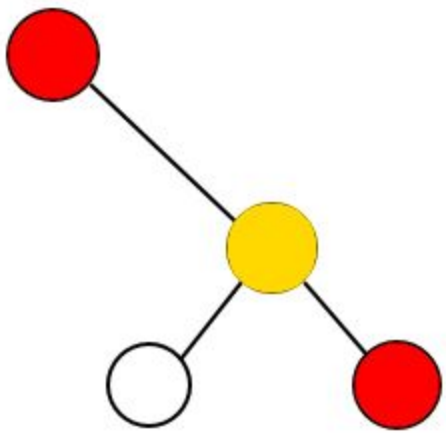
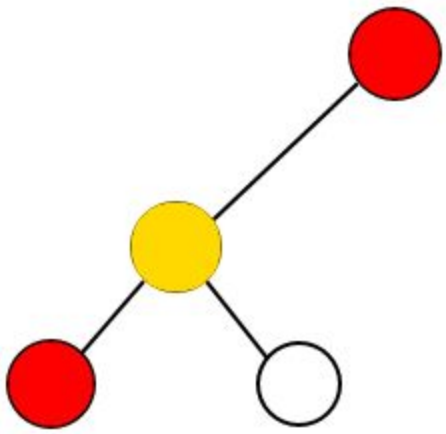
```
                for (int j = 0; j < 63; j++) {
                        if (x & (1LL << j)) b1[j]++;
                        else b0[j]++;
                }
        }

        // find the answer
        ll ans = 0;
        for (int i = 0; i < 63; i++) {
                ans += (1LL << i) * (a0[i] * b1[i] + a1[i] * b0[i]);
        }
        printf("%lld",ans);

        return 0;
}
```

เราสามารถรู้ได้ว่า root จะต้องเป็น node ที่อยู่บน path จาก node 1 ไป node N เท่านั้น




เราจะพิจารณา node ใน path ที่มี node จากนอก path เข้ามาต่อด้วย จะแบ่งได้เป็น 2 กรณีคือ

1. node ที่มาต่อมีค่าน้อยกว่า node ที่อยู่ใน path ที่โดนต่อ
2. node ที่มาต่อมีค่ามากกว่า node ที่อยู่ใน path ที่โดนต่อ

| | |
|---|---|
|  |  |
| กรณี 1. node ที่มาต่อมีค่าน้อยกว่า node ใน path<br><br>เราจะได้ว่า root ต้องอยู่ก่อนหน้า node สีเหลืองเท่านั้น | กรณี 2. node ที่มาต่อมีค่ามากกว่า node ใน path<br><br>เราจะได้ว่า root จะต้องอยู่หลังจาก node นี้เท่านั้น |

หลังจากไล่ตัด node ตามกรณีข้างบนแล้ว เราจะได้ node ที่เหลือที่สามารถเป็น root ได้อยู่จำนวนหนึ่ง node ที่สามารถเป็น root ได้นี้ เราสามารถหยิบมาแค่ 1 node แล้วลองดูว่าเป็น root ของ binary search tree

ได้หรือไม่ ถ้าเป็นได้ เราสามารถสรุปได้ทันทีว่า node ที่เหลือก็จะเป็น root ด้วย แต่ถ้าไม่ได้ เราสามารถสรุปได้ทันทีว่า node ที่เหลือจะไม่สามารถเป็น root ได้แน่นอน

Time Complexity :

- หา path จาก node 1 ไปยัง node N : O(N)
- ไล่ตัด node ที่ไม่ใช่ root : O(N)
- เช็คว่า node เป็น root ของ binary search tree : O(N)

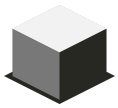
รวมแล้ว O(N)

Solution Code :

```
#include <bits/stdc++.h>
#define N 100005
using namespace std;
int n;
vector<int> adj[N];

void notBST(){
	printf("IMPOSSIBLE");
	exit(0);
}

vector<int> path;
bool fin=false;
bool inPath[N];
bool vst[N];
void dfs(int u){
	if(vst[u]) return;
	vst[u]=true;
	path.push_back(u);
	if(u==1){
		fin=true;
		return;
	}
	for(int v: adj[u]){
		dfs(v);
		if(fin) return;
	}
	path.pop_back();
}

vector<int> inord;
void dfsIn(int u,int p){
	int l=-1, r=-1;
	for(int v:adj[u]){
		if(v!=p){
			if(v<u){
				if(l==-1) l=v;
				else{
					notBST();
				}
			}
			else{
				if(r==-1) r=v;
				else{
					notBST();
				}
			}
		}
```

```
        }
        if(l!=-1) dfsIn(l,u);
        inord.push_back(u);
        if(r!=-1) dfsIn(r,u);
}

bool isAscending(vector<int> &v){
        for(int i=1;i<v.size();i++){
                if(v[i]<=v[i-1]) return false;
        }
        return true;
}

bool roots[N];
void test(int root){
        dfsIn(root,0);
        if(!isAscending(inord)){
                notBST();
        }
        for(int i=1;i<=n;i++){
                if(inPath[i]&&roots[i]) printf("%d ",i);
        }
        exit(0);
}

int main()
{
        scanf("%d",&n);
        for(int i=1;i<n;i++){
                int u,v; scanf("%d%d",&u,&v);
                adj[u].push_back(v);
                adj[v].push_back(u);
        }

        // check if binary tree
        if(adj[1].size()>2||adj[n].size()>2)
                notBST();
        for(int i=2;i<n;i++){
                if(adj[i].size()>3) notBST();
        }

        dfs(n); // find path from n to 1
        reverse(path.begin(),path.end()); // now is from 1 to n
        int m=path.size();
        if(!isAscending(path)){
                notBST();
        }

        //find possible roots
        for(int v:path) inPath[v]=true;
        memset(roots,true,sizeof roots);
        for(int i=0;i<m;i++){
                int u=path[i];
                for(int v:adj[u]){
                        if(!inPath[v]){
                                roots[u]=false;
                        }
                }
                if(!roots[u]){
                        int v;
                        for(int t:adj[u]){
                                if(!inPath[t]){
                                        v=t; break;
                                }
                        }
```

```
                        if(v<u&&i+1<m) roots[path[i+1]]=false;
                        else if(v>u&&i>0) roots[path[i-1]]=false;
                }
        }
        for(int i=m-1;i>=0;i--){
                int u=path[i];
                if(!roots[u]){
                        int v;
                        for(int t:adj[u]){
                                if(!inPath[t]){
                                        v=t; break;
                                }
                        }
                        if(v<u&&i+1<m) roots[path[i+1]]=false;
                        else if(v>u&&i>0) roots[path[i-1]]=false;
                }
        }


        for(int i=1;i<=n;i++){
                if(inPath[i]&&roots[i]) test(i);
        }
        notBST();
        return 0;
}
```



# New Year Party by phirasit

Observation: ถ้า u อยู่ที่โต๊ะที่ 1 ทุกคนที่ไม่เป็นเพื่อนกับ u ต้องอยู่ที่โต๊ะที่ 2

ถ้าให้ G เป็นกราฟความสัมพันธ์ของทุกคน G'เป็น complement graph ของ G
(นั้นคือกราฟที่มีเส้นเชื่อมที่ไม่มีใน G) ถ้าสามารถแบ่งคนให้อยู่ในสองโต๊ะได้ G' ต้องเป็น bipartite graph (กราฟที่สามารถแบ่งเป็นสองส่วนโดยที่ไม่มีเส้นเชื่อมในฝั่งเดียวกัน) สามารถตรวจสอบว่า G' เป็น bipartite graph ได้ด้วยการทำ DFS ออกจากจุดที่ยังไปไม่ถึง

ในส่วนของการแบ่งให้ขนาดโต๊ะใกล้เคียงกันมากที่สุด ให้พิจารณากราฟ G' โดย G' อาจจะไม่ใช่ connected graph โดยทุกครั้งที่เจอ component ( subgraph ที่ connected ) ใหม่จะมีการ DFS ตรวจว่าเป็น bipartite graph รึเปล่าซึ่งสามารถบอกได้ว่าถ้าเป็น bipartite graph
แต่ละฝั่งนั้นจะมีขนาดเท่าไรสิ่งที่เหลือคือบอกว่าฝั่งไหนมีควรจะอยู่โต๊ะที่ไหน

ถ้ากำหนดให้แต่ละ component ของกราฟอยู่ในรูป (ai, bi) เราสามารถเลือกว่าให้ ai คนหรือ bi คนอยู่ที่โต๊ะที่หนึ่งซึ่งเราสามารถทำ dynamic programming (แนวคิดแบบปัญหา knapsack)
ก็จะบอกได้ว่าโต๊ะที่ 1 สามารถใส่คนได้กี่คน สิ่งที่เหลือคือเลือกแบบที่คุ้มที่สุดเท่าที่เป็นไปได้

Running Time : $O(n^2)$ ต่อ test case

Solution Code :

```
#include <algorithm>
#include <cassert>
#include <iostream>

using namespace std;

const int N = 410;
const int N_LIMIT = 400;

int color[ N ];
bool knapsack[ N ];
bool flag;

bool adj[ N ][ N ];
int n, m;

// nCr for test case validation checking
inline int nC2( int n ) { return n * ( n + 1 ) / 2; }

// find the component, raise flag if it's not bipartite
pair< int, int > dfs( int u, int clr = 0 );

int main( void ) {

        cin.sync_with_stdio( false );

        // read num of test cases
```

```
int num_test;
cin >> num_test;
while ( num_test-- ) {

	// read num of nodes and num of edges
	cin >> n >> m;
	assert( 1 <= n and n <= N_LIMIT );
	assert( 0 <= m and m <= nC2( n ) );

	// reset adj, color, knapsack and flag
	for_each( adj + 1, adj + n + 1, []( bool *arr ) {
		fill( arr + 1, arr + n + 1, false );
	} );
	fill( color + 1, color + n + 1, -1 );
	fill( knapsack, knapsack + n + 1, false );
	flag = false;

	// read graph
	for ( int i = 0 ; i < m ; ++i ) {
		int u, v;
		cin >> u >> v;

		assert( 1 <= u and u <= n );
		assert( 1 <= v and v <= n );

		adj[ u ][ v ] = adj[ v ][ u ] = true;
	}

	// compute complement graph
	for ( int u = 1 ; u <= n ; ++u ) {
		for ( int v = 1 ; v <= n ; ++ v ) {
			if ( u != v ) {
				adj[ u ][ v ] = not adj[ u ][ v ];
			}
		}
	}

	// compute each component separately
	knapsack[ 0 ] = true;
	for ( int u = 1 ; u <= n ; ++u ) {
		if ( color[ u ] == -1 ) {
			// new componet found
			int sz1 = dfs( u ).first;

			// update knapsack with sz1
			for ( int i = n-sz1 ; i >= 0 ; --i ) {
				if ( knapsack[ i ] ) {
					knapsack[ i + sz1 ] = true;
				}
			}
		}

	}


	if ( flag ) { // flag is raised
		cout << -1 << endl;
	} else {
		// find the best configuration
		int ans = n;
		for ( int sz = 0 ; sz <= n ; ++sz ) {
			if ( knapsack[ sz ] ) {
				ans = min( ans, abs( ( sz ) - ( n - sz ) ) );
			}
		}
```

```
                        // report the answer
                        cout << ans << endl;
                }
        }

        return 0;
}

pair< int, int > dfs( int u, int clr ) {
        if ( color[ u ] != -1 ) {
                // if the node is visited
                if ( color[ u ] != clr ) {
                        // raise flag when the color is incorrect
                        flag = true;
                }
                return { 0, 0 };
        }

        // set color
        color[ u ] = clr;

        // continue searching
        pair< int, int > ans = { clr == 0, clr == 1 };
        for ( int v = 1 ; v <= n ; ++v ) {
                if ( adj[ u ][ v ] ) {
                        pair< int, int > result = dfs( v, not clr );
                        ans = { ans.first + result.first, ans.second + result.second };

                }
        }

        return ans;
}
```