**Problem Name:** Queens that can attack the king

**Topics:** Array, Matrix, Simulation

**Companies:** Amazon, Microsoft, Media.net

**Level:** Medium

**Language:** C++

**Problem Statement:** On an **8x8** chessboard, there can be multiple Black Queens and one White King.

Given an array of integer coordinates queens that represents the positions of the Black Queens, and a pair of coordinates king that represent the position of the White King, return the coordinates of all the queens (in any order) that can attack the King.

**Input Format:**

First line of the input contains integer n (no of coordinates of queens)

Second line contain 2n space separated integer values of (Xi,Yi) coordinates of queen.

Third line contain integer value (X,Y) coordinates of King.

Ex:

6

0 1 1 0 4 0 0 4 3 3 2 4

0 0

**Output Format:** Print 2D vector representing coordinates of queen targeting king. Ex for above input

Output would be:

0 1 1 0 3 3

**Constraints:**

* 1 <= queens.length <= 63
* queens[i].length == 2
* 0 <= queens[i][j] < 8
* king.length == 2
* 0 <= king[0], king[1] < 8
* At most one piece is allowed in a cell.

**Examples:**

**Input:** queens = [[0,1],[1,0],[4,0],[0,4],[3,3],[2,4]], king = [0,0]

**Output:** [[0,1],[1,0],[3,3]]

**Explanation:**

The queen at [0,1] can attack the king cause they're in the same row.

The queen at [1,0] can attack the king cause they're in the same column.

The queen at [3,3] can attack the king cause they're in the same diagonal.

The queen at [0,4] can't attack the king cause it's blocked by the queen at [0,1].

The queen at [4,0] can't attack the king cause it's blocked by the queen at [1,0].

The queen at [2,4] can't attack the king cause it's not in the same row/column/diagonal as the king.

**Brute force Solution:**

**Explanation:**

**Code:**

#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> intervalIntersection(vector<vector<int>>&l1, vector<vector<int>>&l2) {

    vector<vector<int>>v;

    for(int i=0;i<l1.size();i++){

        for(int j=0;j<l2.size();j++){

            if(l1[i][1]<l2[j][0])

                break;

            int x=max(l1[i][0],l2[j][0]);

            int y=min(l1[i][1],l2[j][1]);

            if(x<=y)

                v.push\_back({x,y});

        }

    }

    return v;

}

int main() {

    int m, n;

    cin>>n;

    vector<vector<int>> first;

    for(int i=0; i<n; i++){

        vector<int> temp(2);

        for(int j=0; j<2; j++){

            cin>>temp[j];

        }

        first.push\_back(temp);

    }

    cin>>m;

    vector<vector<int>> second;

    for(int i=0; i<m; i++){

        vector<int> temp(2);

        for(int j=0; j<2; j++){

            cin>>temp[j];

        }

        second.push\_back(temp);

    }

    vector<vector<int>> result;

    result = intervalIntersection(first, second);

    for(int i=0; i< result.size(); i++){

        for(int j=0; j<2; j++){

            cout<<result[i][j]<<" ";

        }

    }

    return 0;

}

**Time Complexity**: O(m\*n)

**Space Complexity:** O(1)

**Optimized Solution:**

**Explanation:**

Trace in all directions from the king until you hit a queen or go off the board. Since the board is limited to 8 x 8, we can use a Boolean matrix to lookup queen positions; we could use a hash map for a larger board.

**Code:**

#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> intervalIntersection(vector<vector<int>>& A, vector<vector<int>>& B) {

    vector<vector<int>> v;

    int i=0,j=0;

    while(i<A.size() && j<B.size()){

        int l=max(A[i][0], B[j][0]);

        int u=min(A[i][1], B[j][1]);

        if(l<=u)

            v.push\_back({l,u});

        if(A[i][1] < B[j][1])

            i++;

        else

            j++;

    }

    return v;

}

int main() {

    int m, n;

    cin>>n;

    vector<vector<int>> first;

    for(int i=0; i<n; i++){

        vector<int> temp(2);

        for(int j=0; j<2; j++){

            cin>>temp[j];

        }

        first.push\_back(temp);

    }

    cin>>m;

    vector<vector<int>> second;

    for(int i=0; i<m; i++){

        vector<int> temp(2);

        for(int j=0; j<2; j++){

            cin>>temp[j];

        }

        second.push\_back(temp);

    }

    vector<vector<int>> result;

    result = intervalIntersection(first, second);

    for(int i=0; i< result.size(); i++){

        for(int j=0; j<2; j++){

            cout<<result[i][j]<<" ";

        }

    }

    return 0;

}

**Time Complexity**: O(N)

**Space Complexity:** O(1)