**Steps by Knight**

**Medium**Accuracy: 50.11% Submissions: 31973 Points: 4

Given a square chessboard, the initial position of Knight and position of a target. Find out the minimum steps a Knight will take to reach the target position.

Square

Description automatically generated with medium confidence

**Note:**  
The initial and the target position co-ordinates of Knight have been given accoring to 1-base indexing.

**Example 1:**

**Input:**

N=6

knightPos[ ] = {4, 5}

targetPos[ ] = {1, 1}

**Output:**

3

**Explanation:**

Square

Description automatically generated with medium confidence

Knight takes 3 step to reach from

(4, 5) to (1, 1):

(4, 5) -> (5, 3) -> (3, 2) -> (1, 1).

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **minStepToReachTarget()** which takes the inital position of Knight (KnightPos), the target position of Knight (TargetPos) and the size of the chess board (N) as an input parameters and returns the minimum number of steps required by the knight to reach from its current position to the given target position.

**Expected Time Complexity:** O(N2).  
**Expected Auxiliary Space:** O(N2).

**Constraints:**  
1 <= N <= 1000  
1 <= Knight\_pos(X, Y), Targer\_pos(X, Y) <= N

class Solution

{

    public:

    //Function to find out minimum steps Knight needs to reach target position.

    int minStepToReachTarget(vector<int>&KnightPos,vector<int>&TargetPos,int N) {

        // Code here

        int x1=KnightPos[0]-1, y1=KnightPos[1]-1;

        int x2=TargetPos[0]-1, y2=TargetPos[1]-1;

        vector<vector<int>> sol(N, vector<int> (N, 0));

        queue<pair<int, int>> q;

        if (x1==x2 and y1==y2) return 0;

        q.push({x1, y1});

        while (!q.empty()) {

            pair<int, int> p=q.front();

            q.pop();

            int i=p.first, j=p.second;

            if (i==x2 and j==y2) return sol[i][j];

            if (i-1>=0 and j-2>=0 and sol[i-1][j-2]==0) {

                q.push({i-1, j-2});

                sol[i-1][j-2]=sol[i][j]+1;

            }

            if (i+1<N and j-2>=0 and sol[i+1][j-2]==0) {

                q.push({i+1, j-2});

                sol[i+1][j-2]=sol[i][j]+1;

            }

            if (i-2>=0 and j-1>=0 and sol[i-2][j-1]==0) {

                q.push({i-2, j-1});

                sol[i-2][j-1]=sol[i][j]+1;

            }

            if (i-2>=0 and j+1<N and sol[i-2][j+1]==0) {

                q.push({i-2, j+1});

                sol[i-2][j+1]=sol[i][j]+1;

            }

            if (i+2<N and j+1<N and sol[i+2][j+1]==0) {

                q.push({i+2, j+1});

                sol[i+2][j+1]=sol[i][j]+1;

            }

            if (i+2<N and j-1>=0 and sol[i+2][j-1]==0) {

                q.push({i+2, j-1});

                sol[i+2][j-1]=sol[i][j]+1;

            }

            if (i-1>=0 and j+2<N and sol[i-1][j+2]==0) {

                q.push({i-1, j+2});

                sol[i-1][j+2]=sol[i][j]+1;

            }

            if (i+1<N and j+2<N and sol[i+1][j+2]==0) {

                q.push({i+1, j+2});

                sol[i+1][j+2]=sol[i][j]+1;

            }

        }

        return sol[x2][y2];

    }

};