Steps by Knight

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.

Note:
The initial and the target position coordinates of Knight have been given according to 1-base indexing.

Example 1:

Input:

N=6

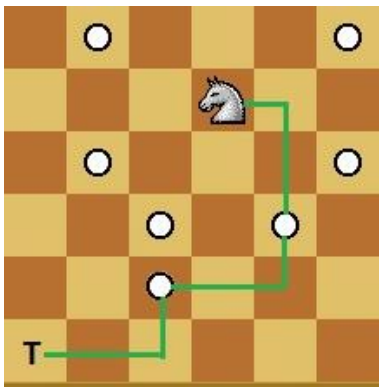
knightPos[ ] = {4, 5}

targetPos[ ] = {1, 1}

Output:

3

Explanation:



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 initial position of Knight (KnightPos), the target position of Knight (TargetPos), and the size of the chessboard (N) as input parameters and returns the minimum number of steps required by the knight to reach from its current position to the given target position or return -1 if its not possible.

Expected Time Complexity: $O(N^2)$.
Expected Auxiliary Space: $O(N^2)$.

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.
     struct Cell {
        int x, y, dist;
        Cell(int x, int y, int dist) : x(x), y(y), dist(dist)
{}
    };

    bool isValid(int x, int y, int N) {
    // Check if the position is within the bounds of the
chessboard
        return (x >= 1 && x <= N && y >= 1 && y <= N);
    }

    int minStepToReachTarget(vector<int>& KnightPos,
vector<int>& TargetPos, int N) {
    // Possible moves for a knight
        int dx[] = {-2, -2, -1, -1, 1, 1, 2, 2};
        int dy[] = {-1, 1, -2, 2, -2, 2, -1, 1};

        // Visited array to keep track of visited positions
    vector<vector<bool>> visited(N + 1, vector<bool>(N + 1,
false));
```

```
    // Queue for BFS, initialized with the starting position
of the knight
    queue<Cell> q;
    q.push(Cell(KnightPos[0], KnightPos[1], 0));

    // Mark the starting position as visited
    visited[KnightPos[0]][KnightPos[1]] = true;

    // BFS loop
    while (!q.empty()) {
        Cell current = q.front();
        q.pop();

        // If the current position is the target position,
return the distance (steps)
        if (current.x == TargetPos[0] && current.y ==
TargetPos[1]) {
            return current.dist;
        }

        // Explore all 8 possible moves
        for (int i = 0; i < 8; i++) {
            int newX = current.x + dx[i];
            int newY = current.y + dy[i];

            // If the new position is valid and not yet
visited, enqueue it
            if (isValid(newX, newY, N) &&
!visited[newX][newY]) {
                visited[newX][newY] = true;
                q.push(Cell(newX, newY, current.dist + 1));
            }
        }
    }

    // If the target is unreachable (should not happen for
valid input)
    return -1;
}
};
```