**Practice Question 1**

Knight can move 2 spaces in one direction and 1 space in the other.

Our graph will be an array of dimensions [8][8]. We want to calculate the minimum distance (dist[8][8]) to each square in the graph. Initialise dist[i][j] = INF for all i, j.

Each square is a node of the graph, hence there are 64 squares. To make edges – we make an array –

int arr[8][2] = { {2, 1}, {1, 2}, {2, -1}, {1, -2}, {-2, 1}, {-1, 2}, {-1, -2}, {-2, -1} };

1. Take input of every black piece. It can be stored in a struct, with variables name, x-position, y-position.
2. Input every white piece, make the dist[i][j] of that piece equal to -1.
3. Input the position of the knight. **Run BFS with this position.**

For every node {i, j}, its adjacent nodes will be {i+arr[k][0], j+arr[k][1]} for k = 0 to 7. The if-else statement for corner cases (where not all edges will give a valid node on the chessboard) will be –

int x = i + arr[k][0];

int y = j + arr[k][1];

if(x<0 || x>=8 || y<0 || y>=8) //skip

else //continue BFS

BFS will run as follows – If dist[x][y] is -1, then ignore it, don’t add that node to the queue. Else, dist[x][y] = min(dist[i][j]+1, dist[x][y]), and add that node to the queue.

Finally, to print the answer, we go back to our struct, iterate through it, and for every name in the struct, print dist[x-position][y-position].

Note –

* In this solution, it has been assumed that to capture a black piece, the knight may capture other black pieces first.
* If there were two knights, the solution would be similar, but, in the beginning, both the knight’s positions would have their dist[i][j] set to 0, and pushed into the queue.