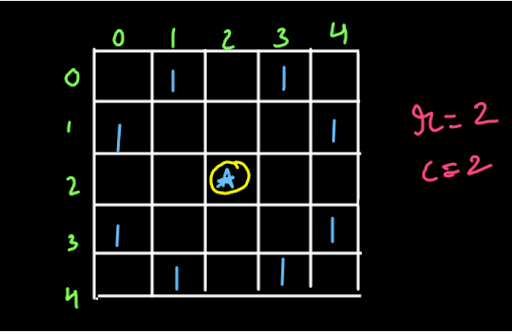
**1. Problem Discussion:**

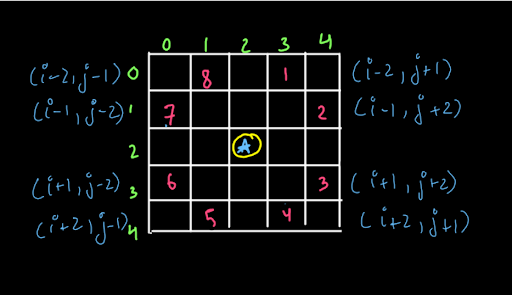
Consider a chess board of size n x n. Let us take n=5. So, let's say we are given a chess board of size 5x5. We are also given the row and column values which will be the starting point of a knight, i.e. a knight is initially kept at the specified row and column.

****

As we can see above, we have a chess board of size 5x5. We are also given the initial point/position where the knight is kept. In this case, it is r=2 and c=2. We have marked all the possible moves of the knight by 1. This means that all the boxes to which a knight can go from the initial spot are marked as 1. Our problem is to print all possible paths which the knight should follow in order to traverse the complete chess board i.e. visit all the boxes of the chess board. The condition is that the knight should visit all the boxes only once, i.e. it cannot go to any box again once it has visited the same. We recommend you try to solve the problem on your own first and then go to the solution.

Hint and Advice before you start with the problem:

Have a look at the diagram given below:

****

We have already shown you the moves that a knight can take. We want you to explore the moves in the order as shown above. This diagram shows that you have to first visit the (i-2,j+1) box, then the (i-1,j+2) box and so on. Obviously, we are not visiting all the boxes in one go, we are just exploring them. If you explore them in the order shown above, your answer will match our output and the online judge will be able to judge your solution. Now you may hit your brain hard and start brainstorming.

**2. Approach:**

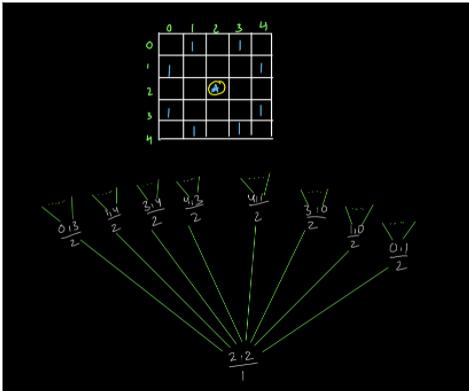
High Level Thinking:

1. Expectation:

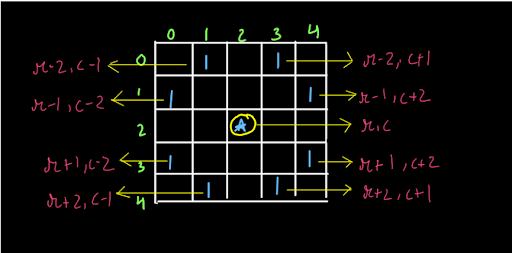
We expect that we will get all possible paths of a knight on the chess board, if we provide the size of the chessboard and the row and column we are currently present in. For instance, in fig-1, we expect that we will get all possible paths on the chess board of size 5x5 if we start from row=2 and column=2.

2. Faith and Relation:

We will use our levels and options method here. Let us just state the faith that we have straight forward. We have faith that recursion will be able to traverse and print the path of the entire chessboard traversal, visiting one block only once from the initial move. What is the initial move here? Have a look at the diagram given below:

****

We are given the initial position of the knight on the 5x5 chessboard as 2,2. Now from r=2 and c=2, we can move to 8 possible blocks in the chess board. These are the options that we have on the next level in the recursion tree. These are the initial moves. Actually, we will choose one move each at a time. So what we want to say is that our expectation is to print the path when we are at 2,2 and we have a faith that recursion can print the paths when we are at any one of the 8 blocks in the next level. This is just like all the source-destination problems that we have already solved where we expect to get from the source to the destination and recursion gives us the path from the initial spot to the destination. If we compare this with the source-destination problem, our source is the r=2 and c=2 (in the above example) i.e. the spot at which the knight is kept initially. Our destination is just to complete the entire board without visiting any block more than once. So, from source, we have 8 initial spots and we have a faith that recursion will print the paths from all these 8 spots. These 8 spots have the positions as shown in the figure below:

****

So, we will have to make 8 recursive calls in this function. We hope that you have clearly understood the high-level thinking process and have got a complete idea of what we are going to do. The process will look something like this: We will start from r and c given to us and the move number will be one. So, we will put 1 at chess[r][c]. This shows that our first move was at row=r and column=c. Then we will make the recursive calls and we will explore the 8 positions in the same order as shown in fig-2. After this, we will make chess[r][c] zero again as we are returning back from the recursion so we have to empty the visited boxes (this is a basic principle that we follow in backtracking). Now that we know the entire procedure (except the base cases), let us write the code for the same.

**3. Algorithm:**

1• We will use the levels and options technique that we have used in many problems earlier as well. Since we are using the levels and options method, we have to decide what options we have at a particular level. A knight can move in 8 possible directions if those moves stay inside the chess board as shown in fig-1 and fig-2. 2• We will apply the recursive calls at each of these levels and the further exploration will be carried out by recursion. 3• We will also have to think about the base cases. Obviously, when we move out of the chess board then we have to return from that particular recursive call. Also, we have to return from the recursive call if we come across a box which has already been visited.

**4. Code:**

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void main(String[] args) throws Exception {

}

public static void printKnightsTour(int[][] chess, int r, int c, int move) {

//Code based on High Level Thinking

chess[r][c] = move; //Fill the chess[r][c] will the move number

//Apply recursive calls on the basis of faith

printKnightsTour(chess, r - 2, c + 1, move + 1);

printKnightsTour(chess, r - 1, c + 2, move + 1);

printKnightsTour(chess, r + 1, c + 2, move + 1);

printKnightsTour(chess, r + 2, c + 1, move + 1);

printKnightsTour(chess, r + 2, c - 1, move + 1);

printKnightsTour(chess, r + 1, c - 2, move + 1);

printKnightsTour(chess, r - 1, c - 2, move + 1);

printKnightsTour(chess, r - 2, c - 1, move + 1);

//Follow the Backtracking principle

chess[r][c] = 0;

}

public static void displayBoard(int[][] chess) {

for (int i = 0; i < chess.length; i++) {

for (int j = 0; j < chess[0].length; j++) {

System.out.print(chess[i][j] + " ");

}

System.out.println();

}

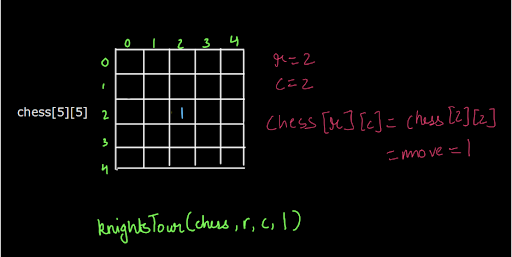
System.out.println();

}

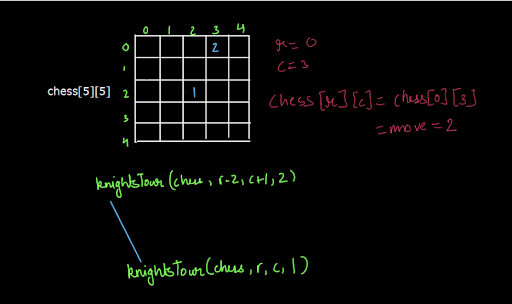
}

What is this code doing?

Let us say we have a 5 x 5 chessboard given to us and the initial row and column values are 2 and 2. Also, we start the counting of the moves from 1 i.e. keeping the knight initially at r=c=2 is also a move and it is move=1. Dear reader, we recommend you keep the code by your side to understand the working that we are explaining here. The first line of code executes i.e. chess[r][c]=move. This means that the matrix chess will now contain the value of move at r,c. Since the move value is one, we have chess[2][2]=1 as shown in the figure below:

****

Now we have 8 recursion calls after this. So, let us make the first recursion call i.e. we reach the second line of the code. Here, a call is made that is: knightsTour(chess,r-2,c+1,move+1). What happens in this call? Let us see. We enter this call and now the row value is r-2 and the column value is c+1. So, at chess[row][col], we insert the value of move and it is 2 now.

****

As you can see, a recursive call was made here. Now, we have executed the first line of code for this call. When we move to the second line, it is again a recursive call. Now if you think, we cannot make this recursive call as it calls for r-2 and c+1. Yes, we have the space available for columns but, we have reached the 0th row. We cannot subtract 2 from here. So, this is one of the base cases for recursion. If we generalize the above situation, we can say that whenever we have to make a move but for that move, we are moving out of the chess board, we have to return. So, the 4 conditions when we cross and go out of the chess board are: If row becomes negative i.e. r<0 If column becomes negative i.e. c<0 If row exceeds the chess board row size i.e. r>=chess.length If column exceeds the chess board column size i.e. c>=chess.length So, in these 4 cases we have to return from the recursion as we went out of the chess board. So, we executed the first recursive call for row=0 and column=3 (see fig-6) and we returned from there. Now when we have returned, we have another problem. We have reached the row=0 and column=3 again. But the call is made for move+3 and we have already visited this block as this was our second move. This means that we do not have to make a recursive call here. So, we have to also return whenever we get a block which is already visited. The condition will be as follows: If chess[r][c]>0, return So, this is also a base case of recursion. Now the last thing that we are left to do is to print the entire chess board. We will print the entire board when we reach the last move. How many moves will be there? We have to visit every block of the chessboard of size nxn. So, there will be nxn moves. So, when we reach nxn move, we will put the move value in the corresponding row and column (that we have been doing so far) and then we will print the board. So, the base cases or the exceptional cases that we handled are: If we reach out of the chess board, return If we reach an already visited block, return If we are at the last move, put the move in the chessboard and display the board.

**5. Analysis:**

Time Complexity:

The time complexity of this solution is O(8n^2) since there are n2 cells and for each we have a maximum of 8 possible moves.

Space Complexity:

The space complexity is O(1) without considering the recursion space and it is O(h) if we consider the recursion space, here h is the height of the stack (max height of the recursion stack).

With this, we have completed this problem.