DATA STRUCTURES AND ALGORITHMS – 2

TERM PROJECT

BATCH: B

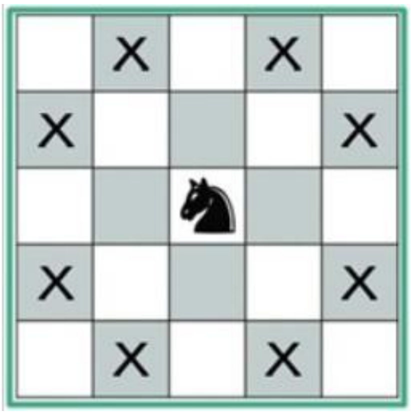
GROUP NO: 2

GROUP MEMBERS:

- CHARISHMA CB.EN.U4AIE21169
- CHANDANA CB.EN.U4AIE21118
- JAIDEV CB.EN.U4AIE21117
- PRANISH CB.EN.U4AIE21137

KNIGHT’S TOUR

We are aware of the knight's unique ability to hop in chess. It may move two squares horizontally and one square vertically, or two squares vertically and one square horizontally, such that the overall movement resembles the letter "L" in English. An empty chessboard and a knight starting from any point on the board are presented in this issue. Our objective is to determine whether the knight can travel to every square on the board.



This problem can have multiple solutions; we have solved it using breadth first search and depth first search algorithm .

Depth First search algorithm:

```
import java.util.Scanner;

public class dfs_approach {
    static int N = 8;
    static int[][] chessBoard = new int[N][N];
    static int[] xMoves = {2, 1, -1, -2, -2, -1, 1, 2};
    static int[] yMoves = {1, 2, 2, 1, -1, -2, -2, -1};

    static boolean isSafe(int x, int y) {
        return (x >= 0 && x < N && y >= 0 && y < N && chessBoard[x][y] == -1);
    }

    static boolean findTour(int x, int y, int moveNumber) {
        if (moveNumber == N * N) {
            return true;
        }

        for (int i = 0; i < N; i++) {
            int newX = x + xMoves[i];
            int newY = y + yMoves[i];
            if (isSafe(newX, newY)) {
                chessBoard[newX][newY] = moveNumber;
                if (findTour(newX, newY, moveNumber + 1)) {
                    return true;
                } else {
                    chessBoard[newX][newY] = -1;
                }
            }
        }

        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the starting coordinates of the knight:");
        int x = sc.nextInt();
        int y = sc.nextInt();
        sc.close();

        for (int i = 0; i < N; i++) {
            for (int j = 0; j < N; j++) {
                chessBoard[i][j] = -1;
            }
        }

        chessBoard[x][y] = 0;

        if (findTour(x, y, 1)) {
            for (int i = 0; i < N; i++) {
                for (int j = 0; j < N; j++) {
                    System.out.print(chessBoard[i][j] + " ");
                }
                System.out.println();
            }
        } else {
            System.out.println("No solution found.");
        }
    }
}
```

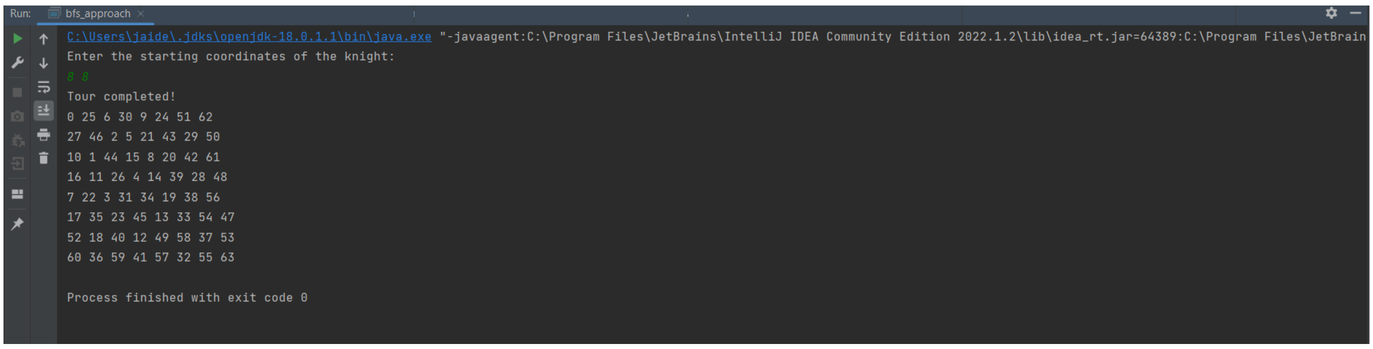
EXPLANATION:

1. First we have imported Scanner class from the java.util package which is used to get input from the user.
2. The next line defines the class of the program, which is dfs_approach.
3. The next few lines define some static variables that will be used throughout the program.
    - N: an integer representing the size of the chess board, set to 8.
    - chessBoard: a two-dimensional array of integers that will represent the chess board and keep track of the knight's tour.
    - xMoves: an array of integers representing the x-coordinate moves that the knight can make.
    - yMoves: an array of integers representing the y-coordinate moves that the knight can make.
4. The next method, isSafe, takes in two integers as its parameters (x, y) and checks if the coordinates are on the board and if it has not been visited before.
5. The findTour method is a recursive function that takes in three integers as its parameters (x, y, moveNumber). This method is where the main logic of the program is.

- It first checks if the moveNumber is equal to N * N, which means that the knight has visited every square on the chess board, returns true.
- It then uses a for loop that iterates N times. For each iteration, it calculates the new x and y coordinates by adding the corresponding xMoves and yMoves values.
- It then checks if the new coordinates are safe, if so it sets the chessBoard cell to the current moveNumber, and then calls findTour recursively with the new coordinates and the next moveNumber.
- If findTour returns true, it means that a solution has been found and it returns true. If it returns false, it means that the current path didn't work and it backtracks by setting the chessBoard cell back to -1 and continues the iteration.
- If none of the recursive calls return true, it means that a solution couldn't be found and it returns false.

6. The main method is the entry point of the program. It creates a new Scanner object to get input from the user, asking for the starting coordinates of the knight.
    - It then initializes the chessBoard with -1.
    - It sets the starting coordinates to 0 on the chessBoard.
    - It then calls findTour with the starting coordinates and the first move number.
    - If findTour returns true, it prints the chessBoard, otherwise it prints "No solution found."

OUTPUT:



```
C:\Users\jaide\.jdks\openjdk-18.0.1.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.1.2\lib\idea_rt.jar=64389:C:\Program Files\JetBrain
Enter the starting coordinates of the knight:
8 8
Tour completed!
0 25 6 30 9 24 51 62
27 46 2 5 21 43 29 50
10 1 44 15 8 20 42 61
16 11 26 4 14 39 28 48
7 22 3 31 34 19 38 56
17 35 23 45 13 33 54 47
52 18 40 12 49 58 37 53
60 36 59 41 57 32 55 63

Process finished with exit code 0
```

Breadth First algoritm:

```java
import java.util.ArrayDeque;
import java.util.Queue;
import java.util.Scanner;

public class bfs_approach {
    private static final int[] ROW_MOVES = {2, 1, -1, -2, -2, -1, 1, 2};
    private static final int[] COL_MOVES = {1, 2, 2, 1, -1, -2, -2, -1};

    private static boolean isValidMove(int[][] chessBoard, int row, int col, int
numRows, int numCols) {
        return row >= 0 && row < numRows && col >= 0 && col < numCols &&
chessBoard[row][col] == -1;
    }

    public static boolean knightsTour(int[][] chessBoard, int numRows, int numCols,
int startRow, int startCol, int moveNumber){
        Queue<Integer> rowQueue = new ArrayDeque<>();
        Queue<Integer> colQueue = new ArrayDeque<>();

        rowQueue.add(startRow);
        colQueue.add(startCol);

        chessBoard[startRow][startCol] = moveNumber;

        while (!rowQueue.isEmpty()) {
            int row = rowQueue.poll();
            int col = colQueue.poll();

            if (moveNumber == numRows * numCols - 1) {
                // we have completed a tour
                return true;
            }

            for (int i = 0; i < ROW_MOVES.length; i++) {
                int newRow = row + ROW_MOVES[i];
                int newCol = col + COL_MOVES[i];
                if (isValidMove(chessBoard, newRow, newCol, numRows, numCols)) {
                    chessBoard[newRow][newCol] = ++moveNumber;
                    rowQueue.add(newRow);
                    colQueue.add(newCol);
                }
            }
        }

        return false;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the starting coordinates of the knight:");
        int numRows = sc.nextInt();
        int numCols = sc.nextInt();
        int[][] chessBoard = new int[numRows][numCols];

        for (int i = 0; i < numRows; i++) {
            for (int j = 0; j < numCols; j++) {
                chessBoard[i][j] = -1;
            }
        }

        int startRow = 0;
        int startCol = 0;

        boolean tourCompleted = knightsTour(chessBoard, numRows, numCols, startRow,
startCol, 0);
```

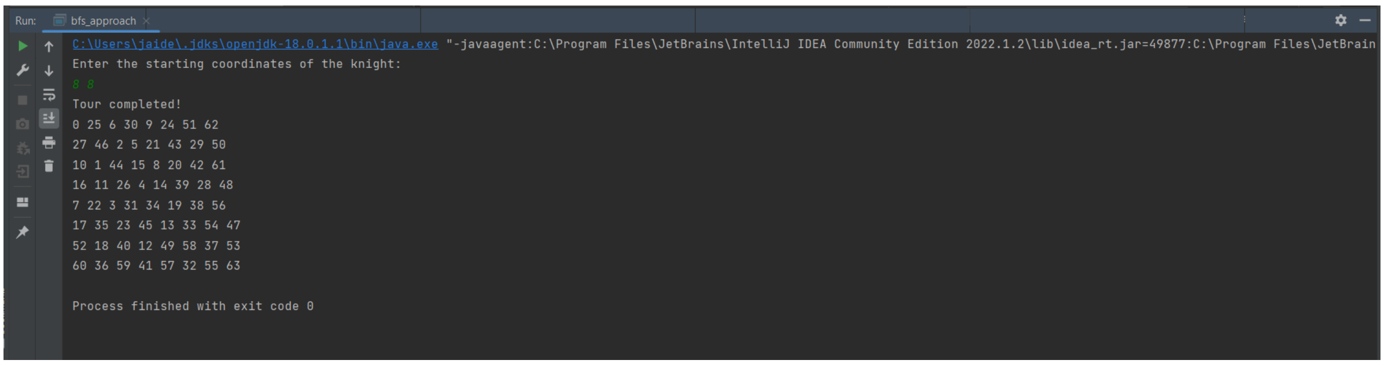
```java
        if (tourCompleted) {
            System.out.println("Tour completed!");
            for (int i = 0; i < numRows; i++) {
                for (int j = 0; j < numCols; j++) {
                    System.out.print(chessBoard[i][j] + " ");
                }
                System.out.println();
            }
        } else {
            System.out.println("Tour not completed.");
        }
    }
}
```

EXPLANATION:

1. In this code first we have imported ArrayDeque, Queue, and Scanner classes from the java.util package. ArrayDeque and Queue are used to implement the BFS algorithm, and the Scanner class is used to read input from the user.
2. The ROW_MOVES and COL_MOVES arrays are constant arrays that contain the possible moves of a knight in 2D chessboard.
3. The isValidMove() method checks whether the given move is valid or not. It takes a chessboard, row, col, numRows and numCols as input. It checks if the move is within the chessboard boundaries and if the square hasn't been visited yet, if both conditions are met it returns true, otherwise it returns false.
4. The knightsTour() method is the main method that is used to solve the Knight's Tour problem. It takes a chessboard, number of rows and columns, starting row and column and move number as input.
5. It initializes two queues, one for the rows and one for the columns, and adds the starting position to these queues. It also marks the starting position as visited by setting the corresponding element of the chessboard to the move number.
6. The method then enters a loop, which continues until the row queue is empty. In each iteration of the loop, it takes the next position from the queues, and checks if the knight has completed a tour. If it has, it returns true.
7. Otherwise, it generates all possible moves from the current position, and checks if each move is valid using the isValidMove() method. If a move is valid, it adds the move to the queues, increments the move number, and marks the square as visited by updating the corresponding element of the chessboard.
8. The main() method is the entry point of the program. It creates a chessboard of the specified size, and initializes all elements to -1.
9. It prompts the user to enter the starting coordinates of the knight.
10. Then it calls the knightsTour() method, passing the chessboard, number of rows and columns, starting position, and move number as input.
11. It checks the returned boolean value from the knightsTour() method, if it is true, it prints the chessboard, otherwise it prints "Tour not completed".

It's worth noting that the solution provided in this code is not guaranteed to find a solution for all starting positions, this is because the solution may get stuck in infinite loops, or it may reach a dead-end. To ensure that the code always finds a solution, you may want to add more sophisticated logic, such as backtracking.

OUTPUT:



```
C:\Users\jaide\.jdks\openjdk-18.0.1.1\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.1.2\lib\idea_rt.jar=49877:C:\Program Files\JetBrain
Enter the starting coordinates of the knight:
8 8
Tour completed!
0 25 6 30 9 24 51 62
27 46 2 5 21 43 29 50
10 1 44 15 8 20 42 61
16 11 26 4 14 39 28 48
7 22 3 31 34 19 38 56
17 35 23 45 13 33 54 47
52 18 40 12 49 58 37 53
60 36 59 41 57 32 55 63

Process finished with exit code 0
```

THANK YOU