Snakes and Ladders - QOTD 24 Jan 23

Leetcode Link : Click

Time : $O(n * n)$

> in worst case we will have to visit all the cells of the matrix

Space : O(n*n)

> 2 d matrix 'isVisited' to store boolen if the cell is visited or not

Approach / Steps

```
/* ✔️⭐Approach - 1 (using BFS)

        explanation :-

                ->//fun.2 :

                    step 1 : use formula

                        row = (n-1) - (data-1) / n;
                        col = (data-1)%n;
                        if(row%2 == n%2) col = n-1-col;


                ->// main function

                step 1 : create a matrix isVisited[n][n], to not visit the already
visited elements again (initialize all the elements as false initially)
                step 2 : now create a q, and push 1 into it (because initially we are
at 1 ) (also mark 1 as visited into the visited array[note thet 1 will be at board[n-
1][0] place])
                step 3 : initialize steps = 0
                step 4 : now traverse this q while it is not empty
                    step 5: find the curr size of the q and run a loop from 0 to curr
size,note that this below loop is added coz we should increment the steps once all
posiblites to where we can move for 1 step are visited.
                        step 6 : pop the front element
                        step 7 : if front element is already the final box, then
return steps
                        step 8 : now run a loop from i = 1 to 6
                            step 9 : create a int 'newData ' and add this front ele
with i and store it in newData
                            step 10 : if the newData is greater then the matrix last
number, then in that case break the loop, coz we have explored the possiblities from 1
to 6
                            step 11 : call fun.2 to and find the coordinates of this
new Data in input matrix (board)
                            step 12 : if the current cell we are on is visited, then
continue to next cell, else visit it
                            step 13 : if current cell is not a snake or ladder then
store the sum into the q
                            step 14 : if the current cell is a ladder or snake, then
push the destination value of ladder / snake into queue and not the current cell value
                    step 15 : increment steps by 1 once all the possible next steps
have been visited for 1 front element eg. incrment 1 when after cell = 1,
(2,3,4,5,6,7) have also been visited and no answer was found
                step 16 : return -1 when all loops are complete, and we still did not
reached the final cell


*/
```

code :-

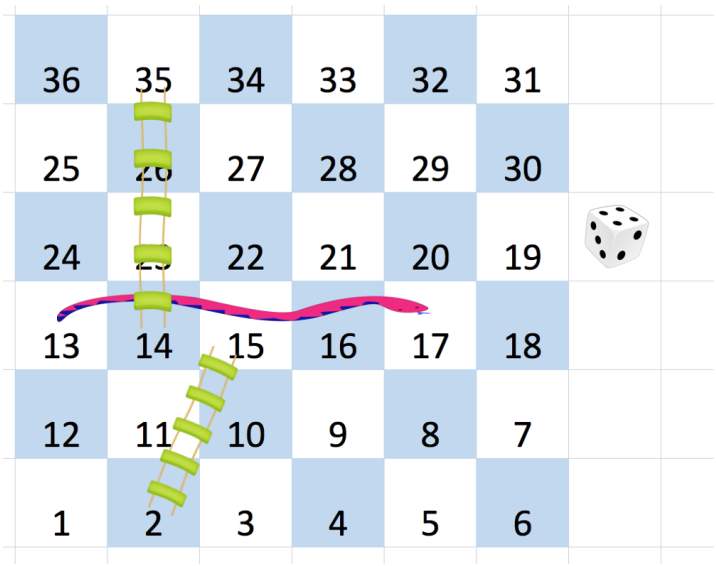
```
private://fun.2 :  function to return row and col coordinates of any 'data' ,for any matrix of size 'n'
    void findCordinates(int n, int data, int &row, int &col){

        row = (n-1) - (data-1) / n;

        col = (data-1)%n;

        if(row%2 == n%2) col = n-1-col;
    }

public:
    // Main function
    int snakesAndLadders(vector<vector<int>>& board) {

        int n = board[0].size() ; //n x n matrix

        // step 1 : create a matrix isVisited[n][n], to not visit the already visited elements again (initialize all the elements as false initially)
        vector<vector<bool>> isVisited(n, vector<bool>(n,false));

        // step 2 : now create a q, and push 1 into it (because initially we are at 1 ) (also mark 1 as visited into the visited array[note thet 1 will be at board[n-1][0] place])
        queue<int> q;
        q.push(1);
        isVisited[n-1][0] = true;

        // step 3 : initialize steps = 0
        int steps = 0;

        // step 4 : now traverse this q while it is not empty
        while(!q.empty()){

            //step 5: find the curr size of the q and run a loop from 0 to curr size,note that this below loop is added coz we should increment the steps once all posiblites to where we can move for 1 step are visited.
            int size = q.size();
            for(int j = 0; j < size; j++){

                // step 6 : pop the front element
                int frontEle = q.front();
                q.pop();

                // step 6 : if front element is already the final box, then return steps
                if(frontEle == n*n) return steps;

                // step 7 : now run a loop from i = 1 to 6
                for(int i = 1; i <= 6; i++){
```

```
                    // step 8 : create a int 'newData ' and add this front ele with i
and store it in newData
                    int newData = frontEle + i;

                    // step 9 : if the newData is greater then the matrix last number,
then in that case break the loop, coz we have explored the possiblities from 1 to 6
                    if(newData > n*n) break;

                    // step 10 : call fun.2 to and find the coordinates of this new
Data in input matrix (board)
                    int row, col;
                    findCordinates(n,newData,row,col);

                    // step 11 : if the current cell we are on is visited, then
continue to next cell, else visit it
                    if(isVisited[row][col]) continue;
                    isVisited[row][col] = true;


                    // step 12 : if current cell is not a snake or ladder then store
the sum into the q
                    if(board[row][col] == -1) q.push(newData);

                    // step 13 : if the current cell is a ladder or snake, then push
the destination value of ladder / snake into queue and not the current cell value
                    else
                        q.push(board[row][col]);

                }

            }
            // step 14 : increment steps by 1 once all the possible next steps have
been visited for 1 front element eg. incrment 1 when after cell = 1, (2,3,4,5,6,7)
have also been visited and no answer was found
            steps++;

        }

        // step 15 : return -1 when all loops are complete, and we still did not
reached the final cell
        return -1;
    }
```

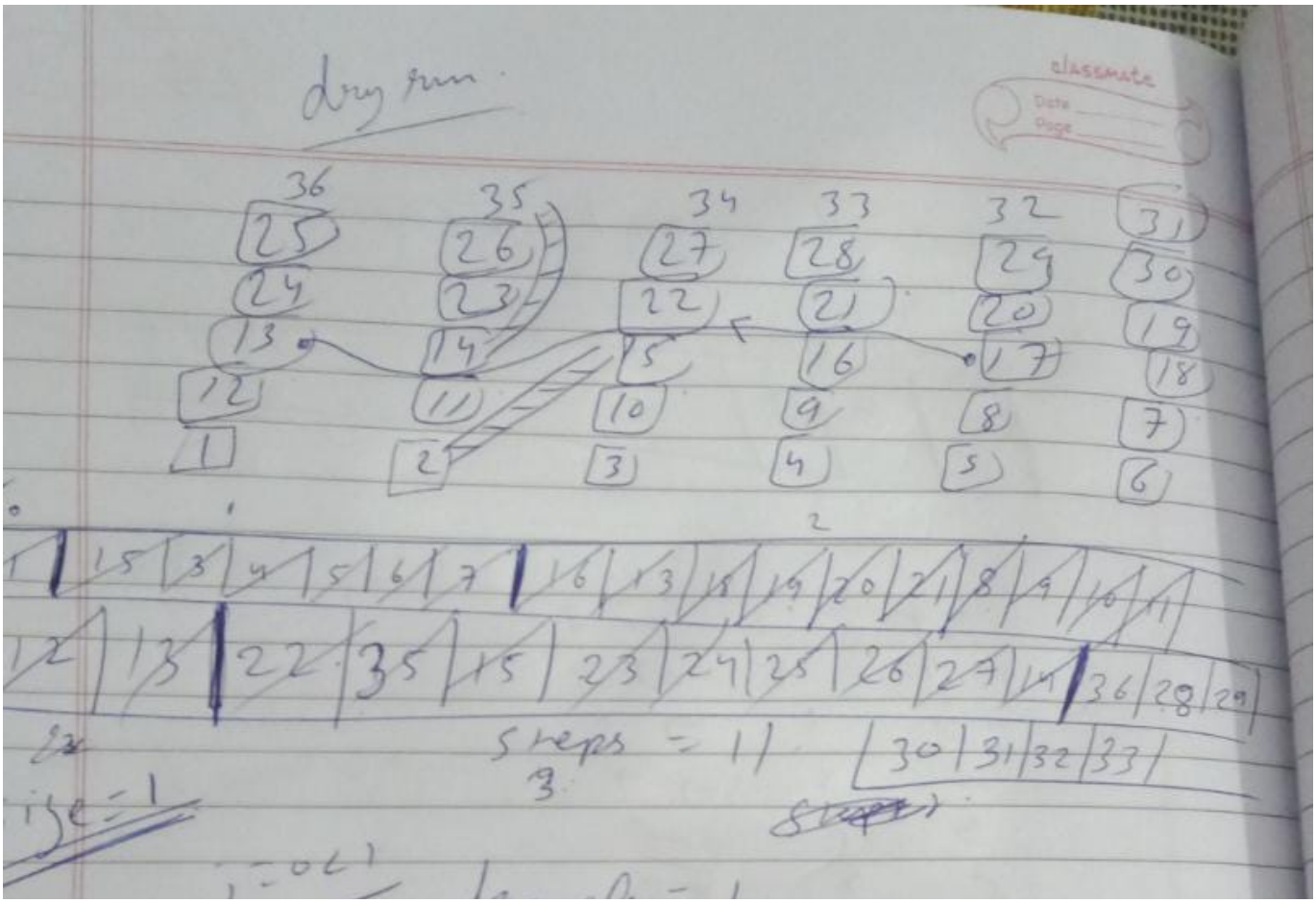
dry run for below test case :-

Input :-

```
board =
[[-1,-1,-1,-1,-1,-1],
 [-1,-1,-1,-1,-1,-1],
 [-1,-1,-1,-1,-1,-1],
 [-1,35,-1,-1,13,-1],
 [-1,-1,-1,-1,-1,-1],
 [-1,15,-1,-1,-1,-1]]
```



dry run :-



front = 1

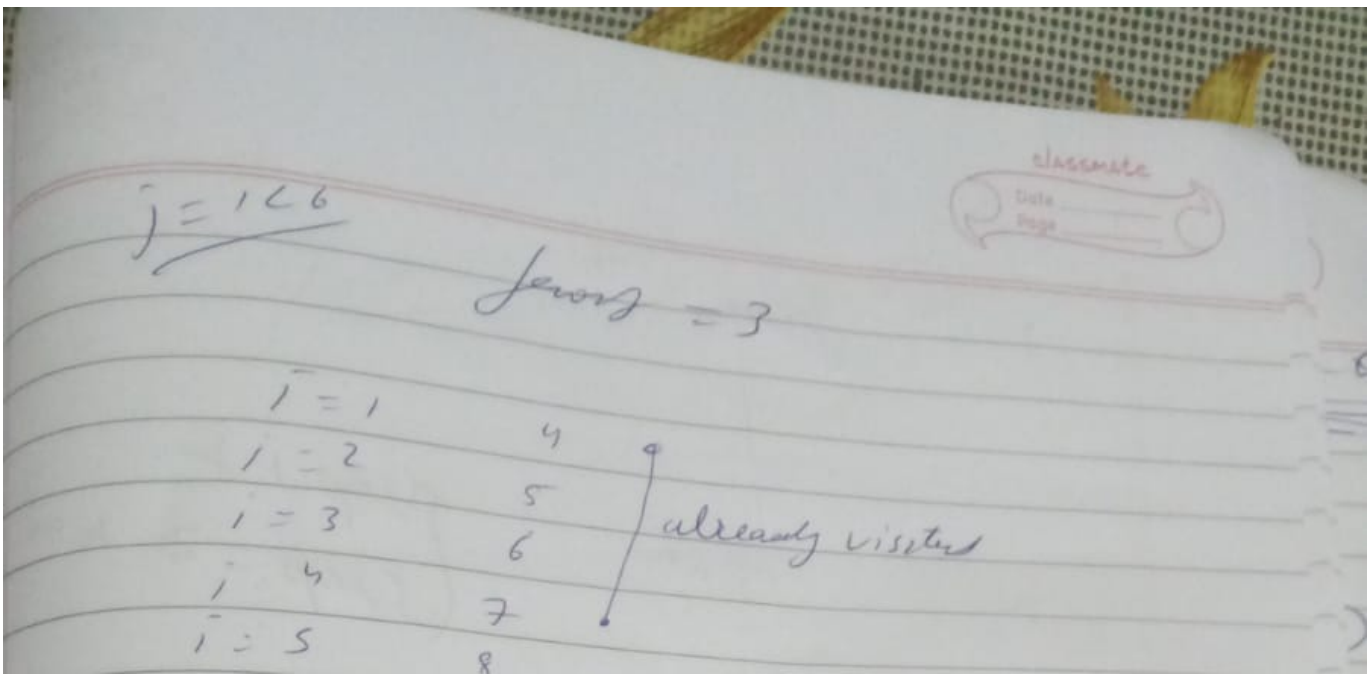
i = 1 2 => 15
2 3
3 4
4 5
5 6
i = 6 7

j = 1 < 1 ~~6~~ break steps + 1
(step now = 1)

~~size = 6~~

j = 0 < 6 front = 15

i = 1 16
2 17 => 13
3 18
4 19
5 20
6 21



j = 1 < 6

front = 3

i = 1 4
i = 2 5
i = 3 6
i = 4 7
i = 5 8

already visited

i = 6   9] new.

j < 2 < 6

front = 4

j = 1   5
2   6
3   7   already visited
4   8
5   9
6   10   new

j < 3 < 6

front = 5

j = 1   6
2   7   already visited
3   8
4   9
5   10
6   11. new.

j < 4 < 6   front 6

j < 5 < 6

12 new

$j = 5 < 6$ found = 7.

|

13 now

{step + 1
step = 2 now}

~~5~~ ~~e = 12~~

~~j = 0 < 12~~

found = 16

i = 1 17 ⇒ 13
18
19
20
21
22 now

~~j = 1 < 12~~ found = 13.

i = 1 14 ⇒ 35.
2 15
16
17 ⇒ 13.

6. 18
19

j = 2<12

jvars = 18

i=1 19
20
21 / already visited.
22
23 nw
24 nw

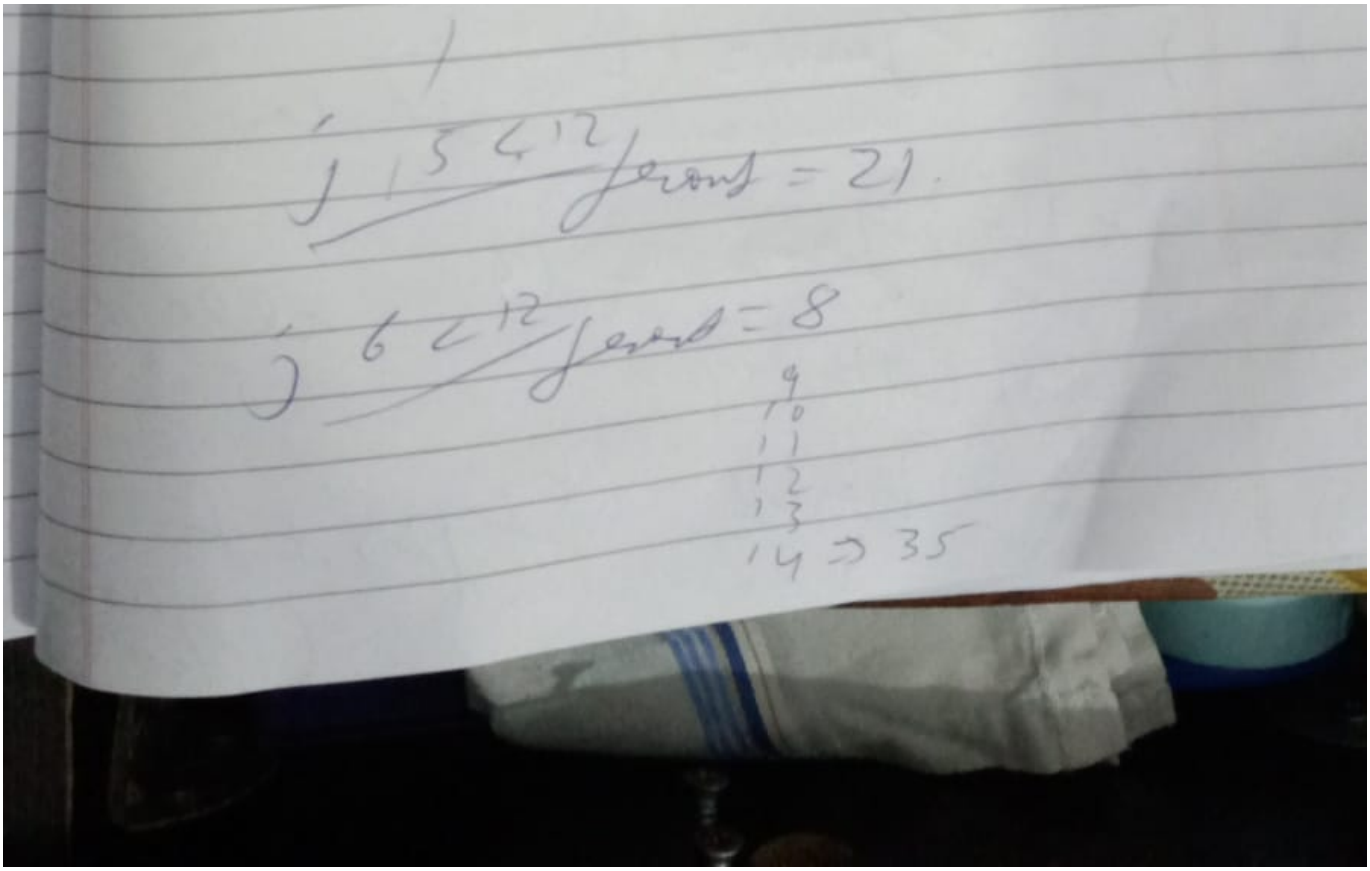
j = 3<12

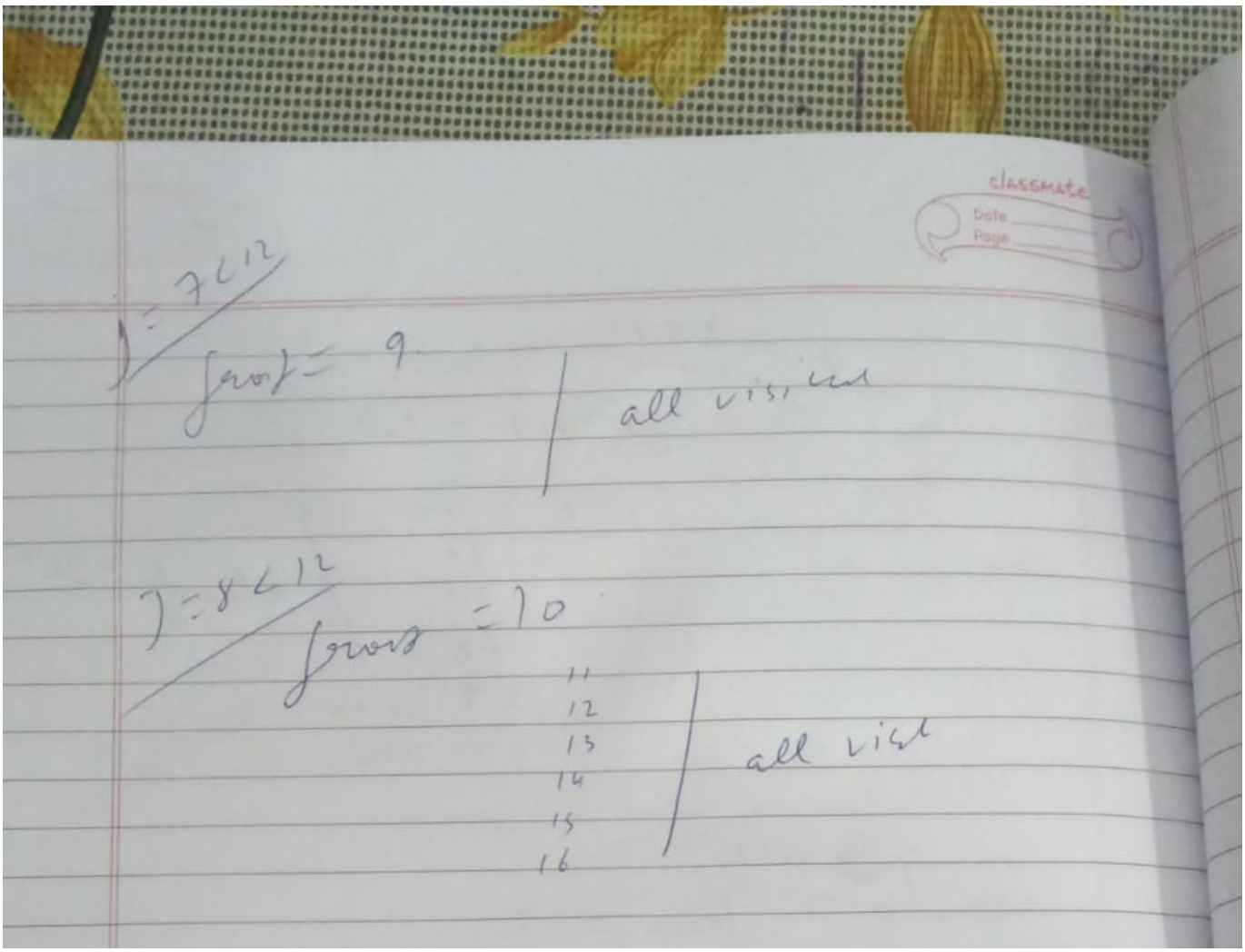
jvars = 19.

20
21
22
23
24
25

j = 4<12

jvars = 20


j = 5 < 12 / jrow = 21.
j = 6 < 12 / jrow = 8
9
10
11
12
13
14 ⇒ 35


classmate
Date
Page
j = 7 < 12 / jrow = 9
all visited
j = 8 < 12 / jrow = 10
11
12
13
14
15
16
all visit

j = 9 < 12 / jevons = 11

12
13
14 = 35.
15
16
17 = 13

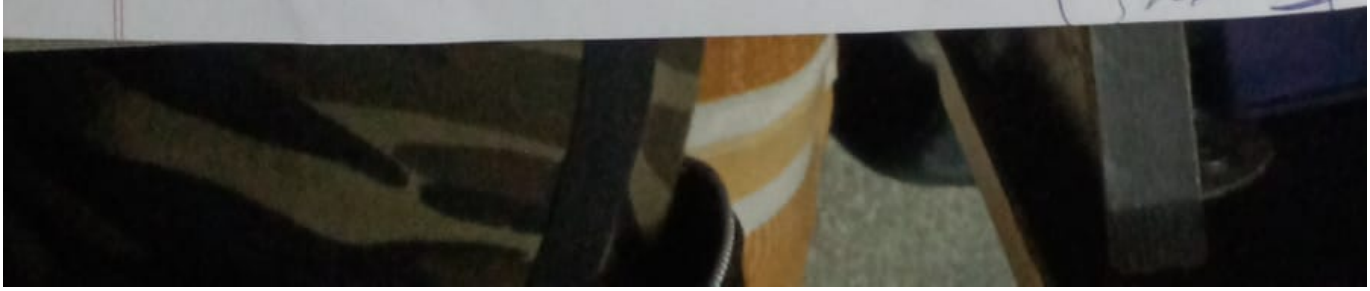
all visited

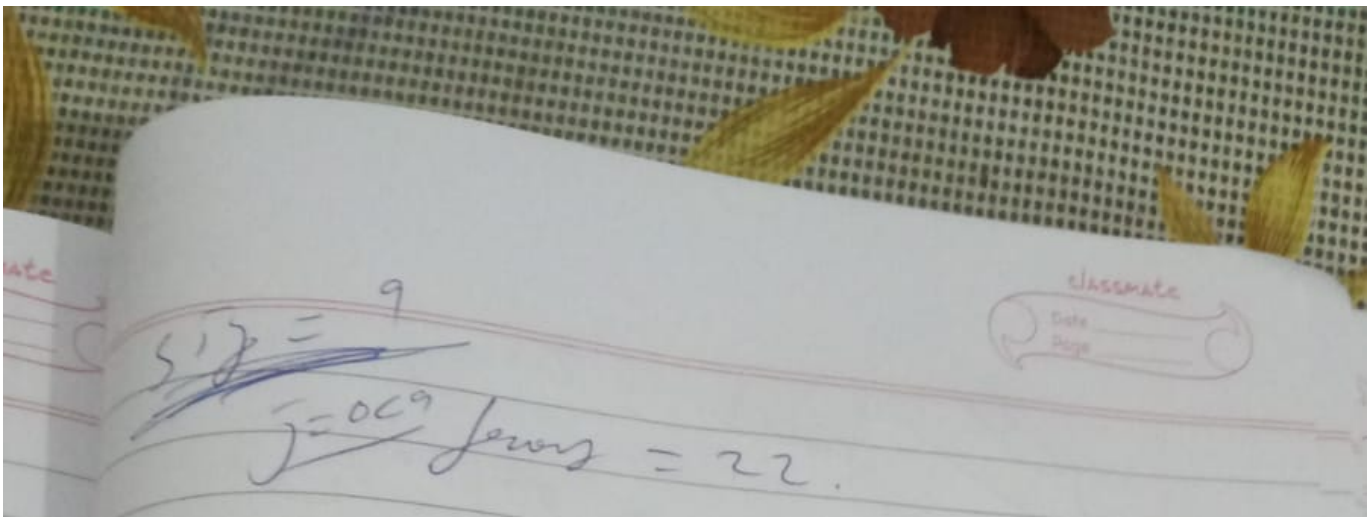
j = 10 < 12 / jevons = 12. all don

j = 11 < 12 / jevons = 13. all don

j = 12 < 12

bevels steps ++.





9

j = 0 < 9 / jevons = 22.

23
24
25
26
27
28 (new)

~~j = 1 < 9~~ front = 35

~~36~~

36 (new)
37 break (36 > 3*n)

break;

j = 2 < 9 front = 15

16
17
18
19
20
21 — all visited

j = 3 < 9 front = 23

24, 25, 26, 27, 28, 29 (new)

~~29 new~~

j = 4 < 9 front = 24

25, 26, 27, 28, 29, 30 (new)

j=5 < 9 / for = 25

26, 27, 28, 29, 30, 31
new

j=6 < 9 / for = 26

32 / new

j=7 < 9 / for = 27

33 / new

j=8 < 9 / for = 14

15, 16, 17, 18, 19, 20

all v... already

j = 9 < 9

break. steps ++.

(... = 4)

size = 7

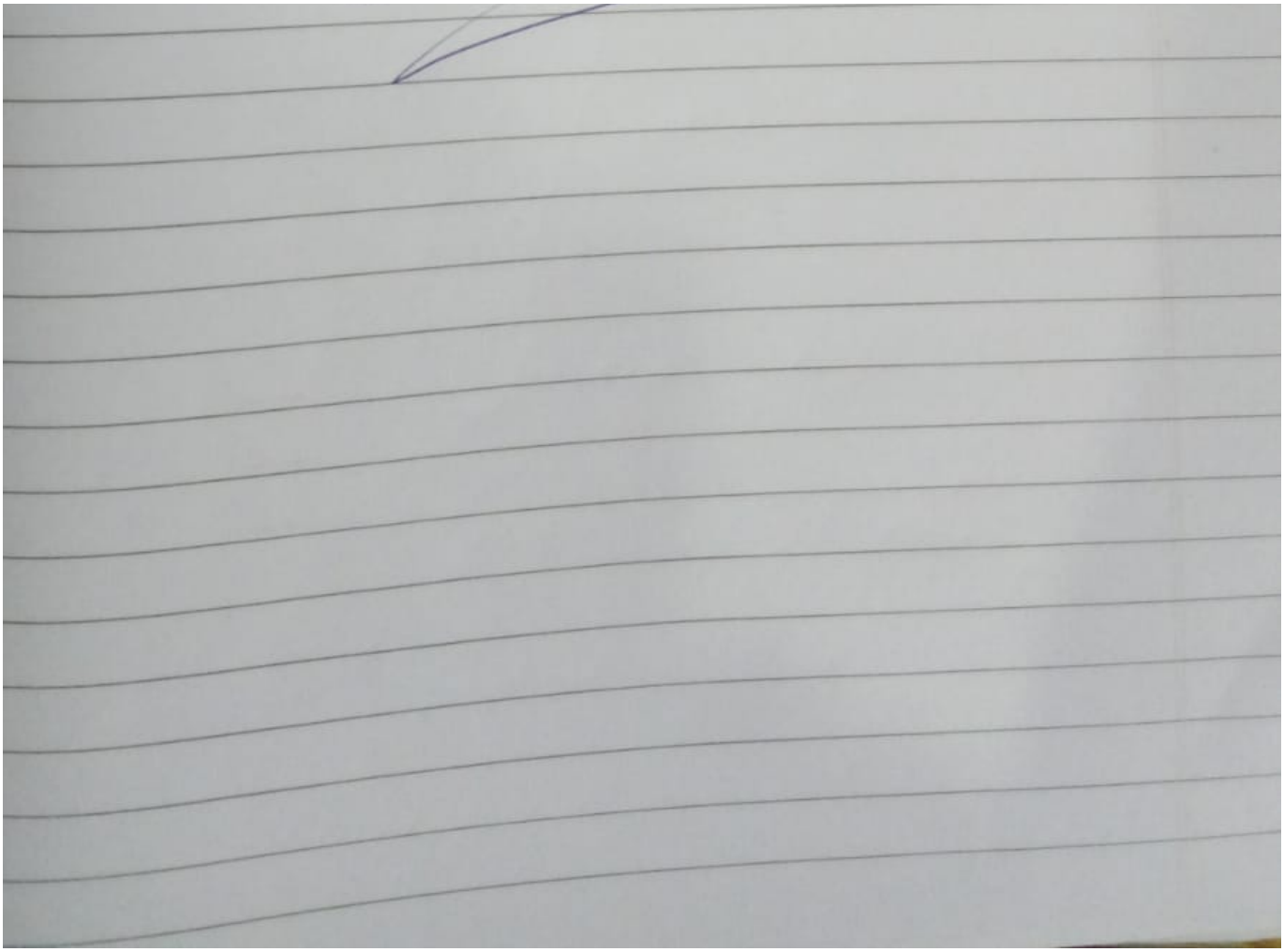
j = 0 < 7

front = 36

front == n * n (6 × 6 = 36)

return steps.

Ans = 4



END