Input

|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |  |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | {{3, | 0, | 6, | 5, | 0, | 8, | 4, | 0, | 0}, |  |
| 1 | {5, | 2, | 0, | 0, | 0, | 0, | 0, | 0, | 0}, | 0 |
| 2 | {0, | 8, | 7, | 0, | 0, | 0, | 0, | 3, | 1}, |  |
| 3 | {0, | 0, | 3, | 0, | 1, | 0, | 0, | 8, | 0}, |  |
| 4 | {9, | 0, | 0, | 8, | 6, | 3, | 0, | 0, | 5}, | 1 |
| 5 | {0, | 5, | 0, | 0, | 9, | 0, | 6, | 0, | 0}, |  |
| 6 | {1, | 3, | 0, | 0, | 0, | 0, | 2, | 5, | 0}, |  |
| 7 | {0, | 0, | 0, | 0, | 0, | 0, | 0, | 7, | 4}, | 2 |
| 8 | {0, | 0, | 5, | 2, | 0, | 6, | 3, | 0, | 0}} |  |
|  | 0 |  |  | 1 |  |  | 2 |  |  |  |

# CanWePlaceNumber()

$r=4, c=7$

$R = \frac{r}{3}$ $\quad C = \frac{c}{3}$

$(i = R*3 ; i < (R*3)+3 , i++)$

$(j = C*3; j<(C*3)+3; j++)$

for iterating on any big cell if you know R, C value the row of the cell starts from $R \times 3$ and column of big cell starts from $C \times 3$

$f(r, c)$ = canweplaceNumber(r, c, k)

↳ grid[r][c] → k  $\underline{\underline{k \in [1-9]}}$

$f(r, c+1)$

grid[r][c] = ''

this function starts filling the grid as Sudoku from $\underline{\underline{r, c}}$ cell

```
if (c == 9) {
    f(r+1, 0)
}
```

→ filling Sudoku can be complex i.e why we will recursively try to solve the problem.

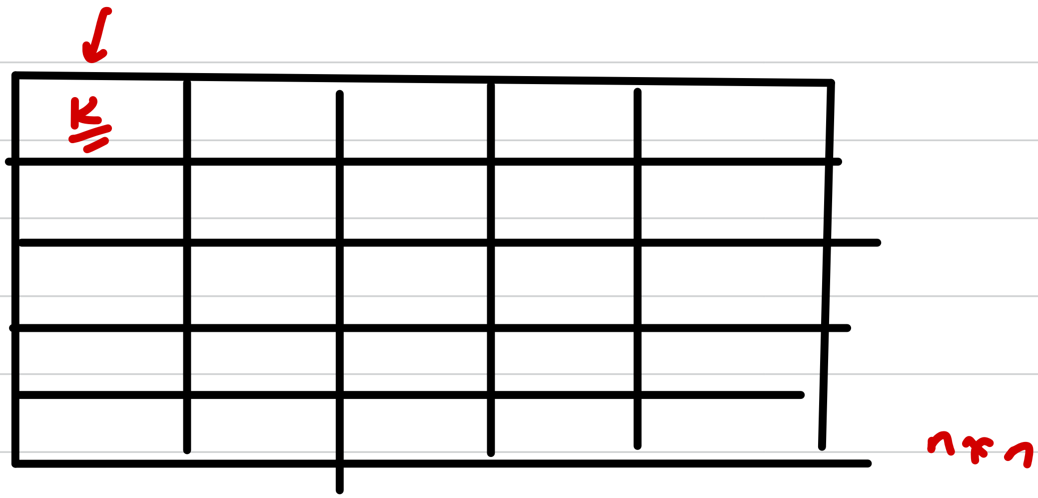
Self work → at any cell $(r, c)$ if we can safely put a number $k$, put it.

Recursive Intuition → if some one can fill the rest of the grid for us

While filling the grid if we exhaust a column, move to the next row, and if we have exhausted all rows we got the ans.

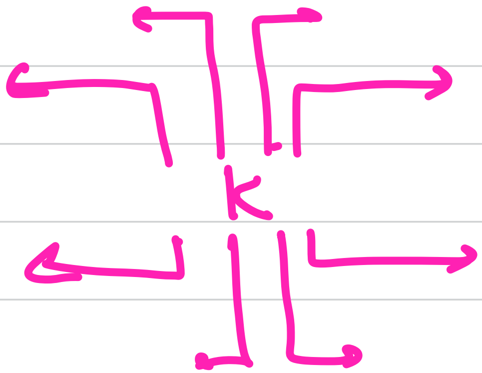
Input

```
{ {3, 1, 6, 5, 7, 8, 4, 9, 2},
  {5, 2, 9, 1, 3, 4, 7, 6, 8},
  {4, 8, 7, 6, 2, 9, 5, 3, 1},
  {2, 6, 3, 4, 1, 5, 9, 8, 7},
  {9, 7, 4, 8, 6, 3, 1, 2, 5},
  {8, 5, 1, 7, 9, 2, 6, 4, 3},
  {1, 3, 8, 9, 4, 7, 2, 5, 6},
  {6, 9, 2, 3, 5, 1, 8, 7, 4},
  {7, 4, 5, 2, 8, 6, 3, 1, 9} }
```




if while travelling we went outside the grid then we just return

Knight has to touch all the cells only once.




if the cell is inside the grid & unvisited then we go there.

$$f(i, j, c) = \begin{cases} f(i+1, j+2, c+1), \\ f(i+2, j+1, c+1), \\ f(i-1, j+2, c+1), \\ f(i-2, j+1, c+1), \\ f(i+1, j-2, c+1 \\ f(i+2, j-1, c+1), \\ f(i-1, j-2, c+1), \\ f(i-2, j-1, c+1) \end{cases}$$

this function does the knights tour from i, j at the $c^{th}$ step.

(max $n^2$ steps we can take)

j

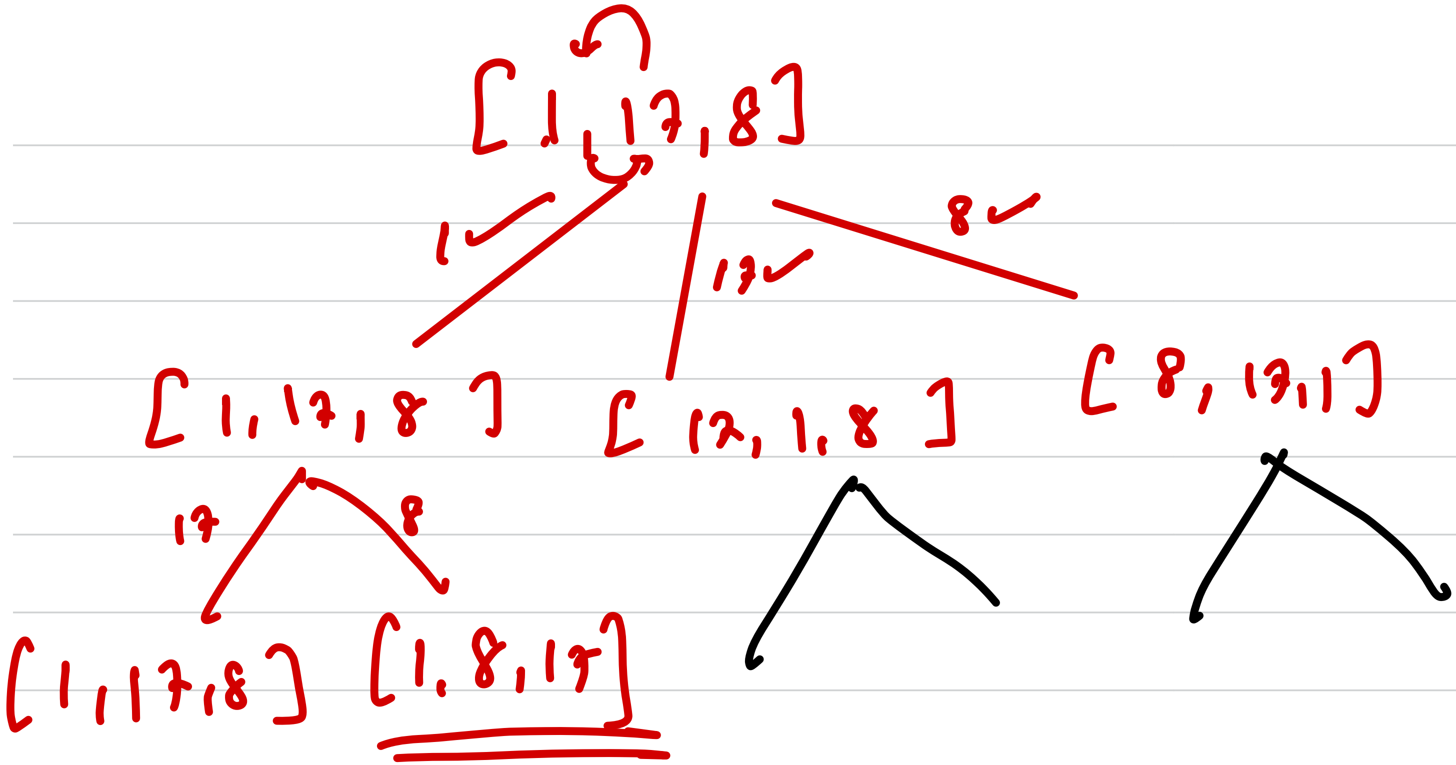
| | | | | |
|---|---|---|---|---|
| 0 | -1 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 2 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

-1

grid[i][j] = c

f( _ , _ , c+1)

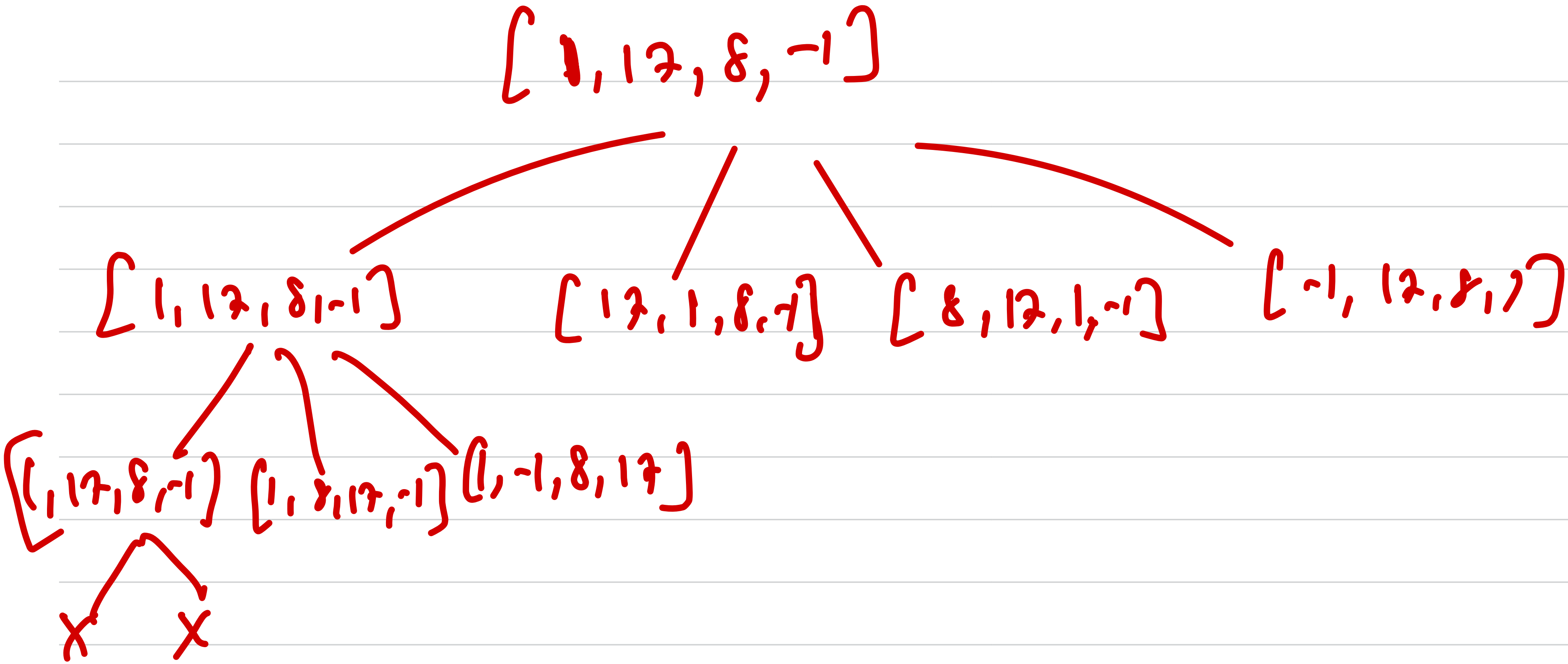
grid[i][j] = -1




→ generate perm

↳ follow and sq full arry ]

So while generating the permutation we can at the same time check if the adjacent pairs are perfect square. So if any one pair is also not a perfect sq., then we will move ahead with the perm, as some change later can help us to get sqful arr.


[1, 17, 8, -1]
[1, 17, 8, -1]
[17, 1, 8, -1]
[8, 17, 1, -1]
[-1, 17, 8, 1]
[1, 17, 8, -1]
[1, 8, 17, -1]
[1, -1, 8, 17]
X
X