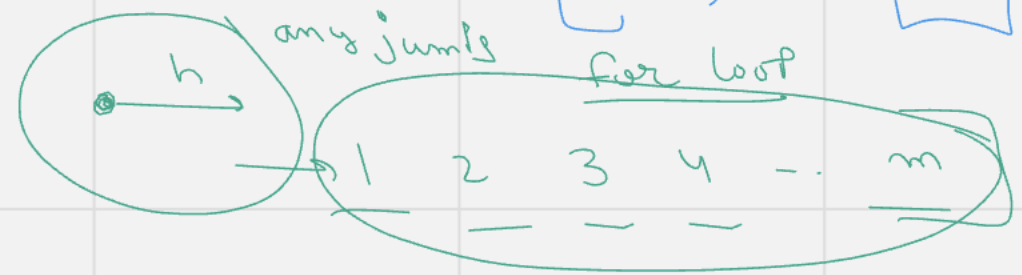
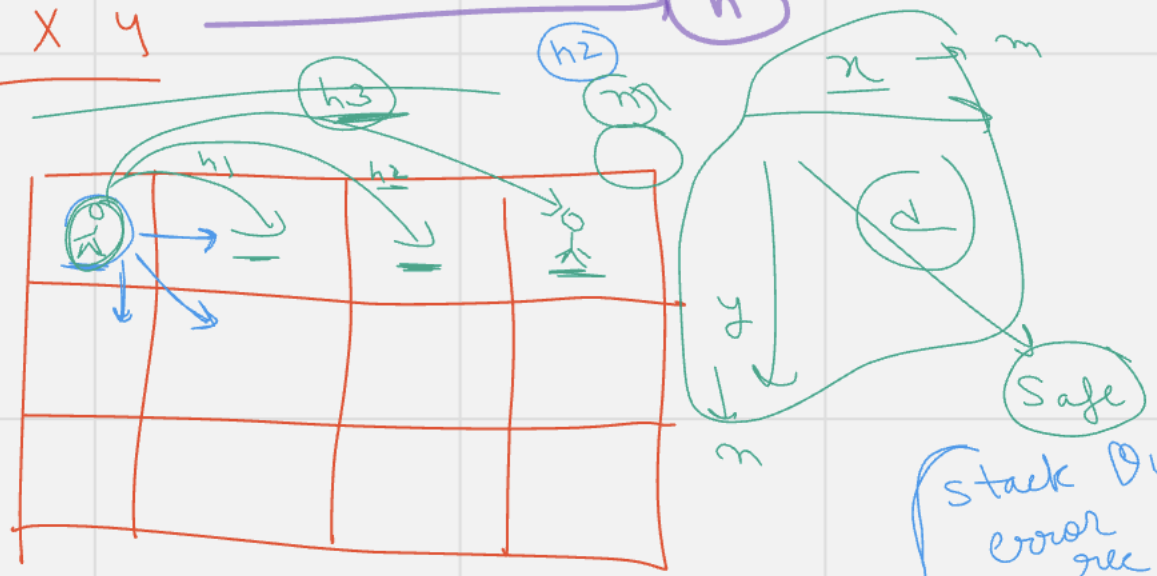
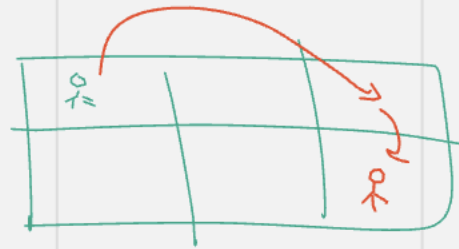


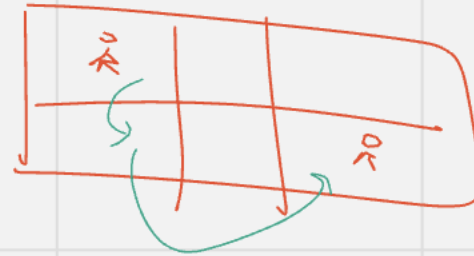
① skip

① option $\rightarrow h$
3 X 4





h 2 v 1
pm



v 1 h 2

```
public static void printMazePaths(int i, int j, int n, int m, String psf) {
    // safe
    if(j > m || i > n){
        return;
    }

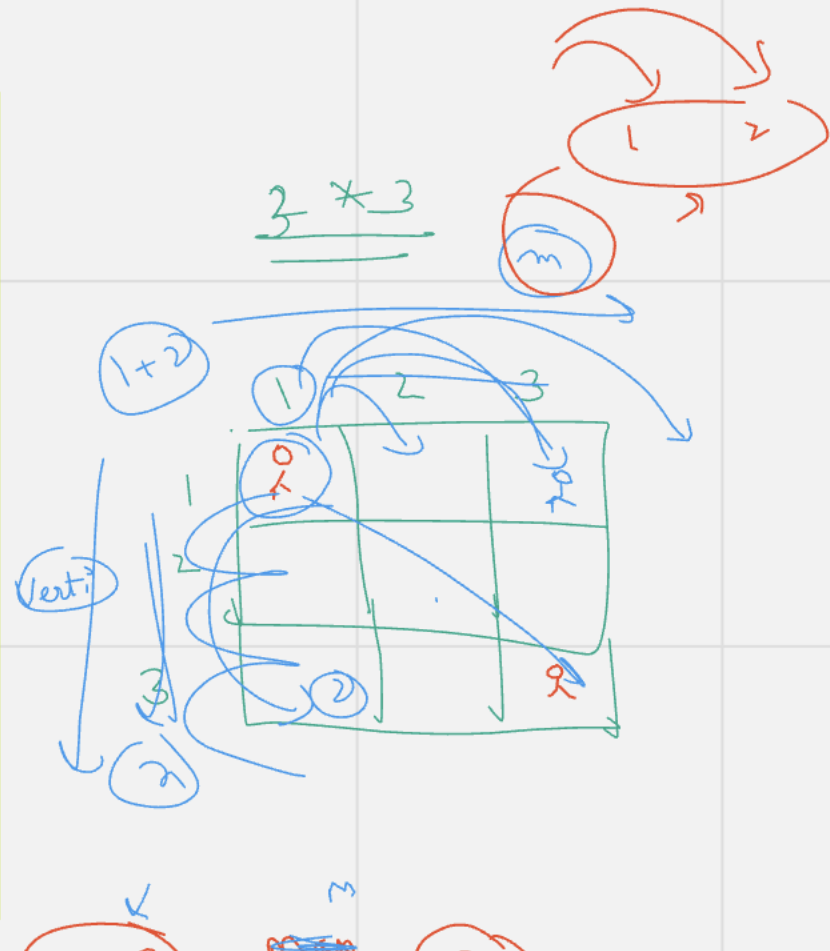
    // base case
    if(i == n && j == m) {
        System.out.println(psf);
        return;
    }

    for(int x = 1; x <= m; x++) {
        // hori
        printMazePaths(i, j + x, n, m, psf + "h" + x);
    }

    for(int y = 1; y <= n; y++) {
        // vert
        printMazePaths(i + y, j, n, m, psf + "v" + y);
    }

    for(int d = 1; d <= Math.min(n, m); d++) {
        // dia
        printMazePaths(i + d, j + d, n, m, psf + "d" + d);
    }
}
```

3 * 3



1 2 3
 1 (3 2)

9) single

Pair $10 \leq 20 \leq 26$

10 2 3

0 2 3

" 11 "

a 3
table

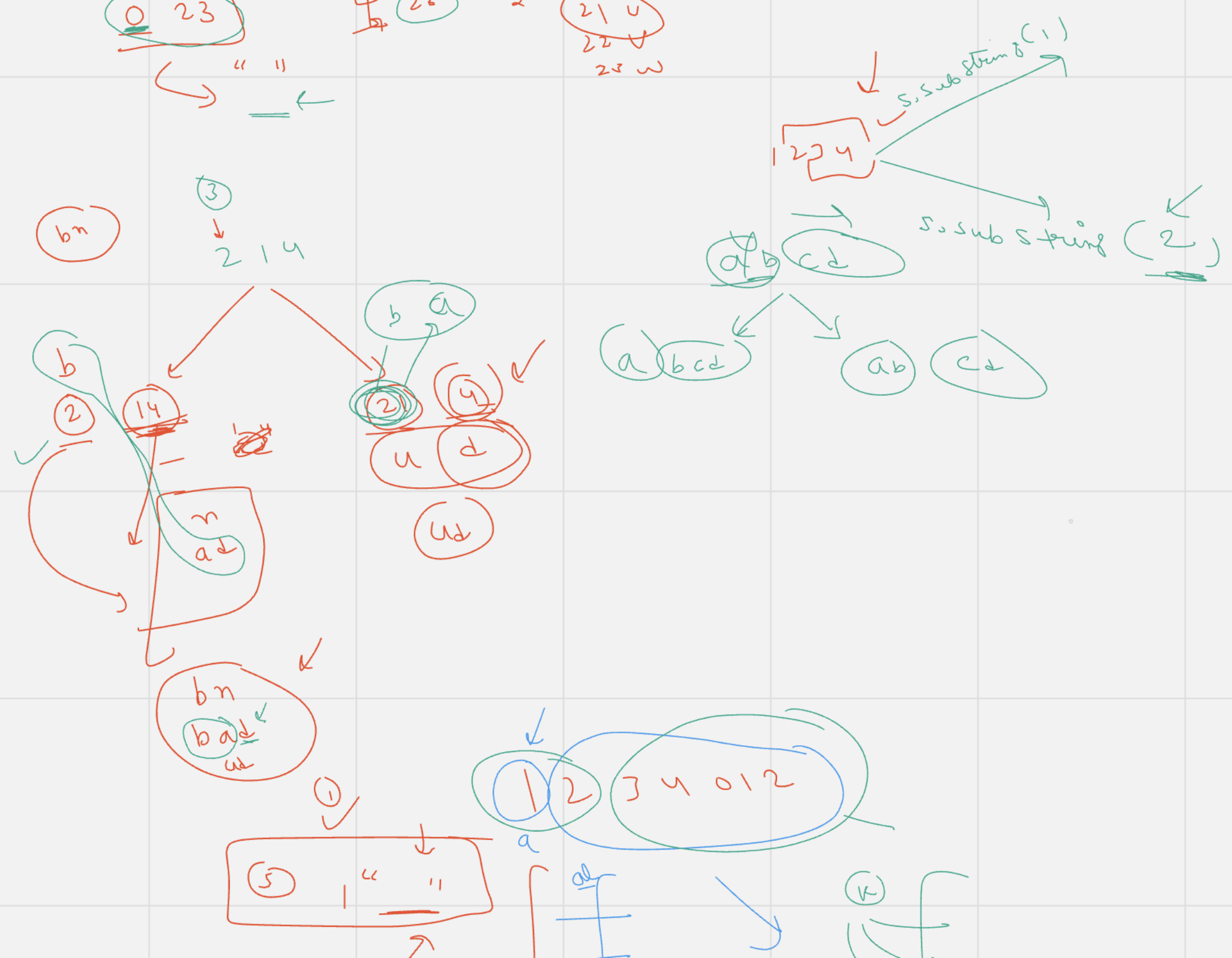
1 → a
 2 → b
 3 → c
 ...
 9
 10 → j
 11 → k
 12 → l
 13 → m
 14 → n
 15 → o
 16 → p
 17 → q
 18 → r
 19 → s
 20 → t
 21 → u
 22 → v
 23 → w
 24 → x
 25 → y
 26 → z

abc
 lc
 aw

P
 F
 R
 S
 T
 21 u
 22 v
 23 w

(1 2 3)
 a w
 1 2 3
 a b c
 (12) 3
 lc

S, SubString (1)



1 2 3

1

23

12

3

helper(s.substring(1), ans + (char)(s.charAt(0) - '0' + 96));

w

'1' - '0'

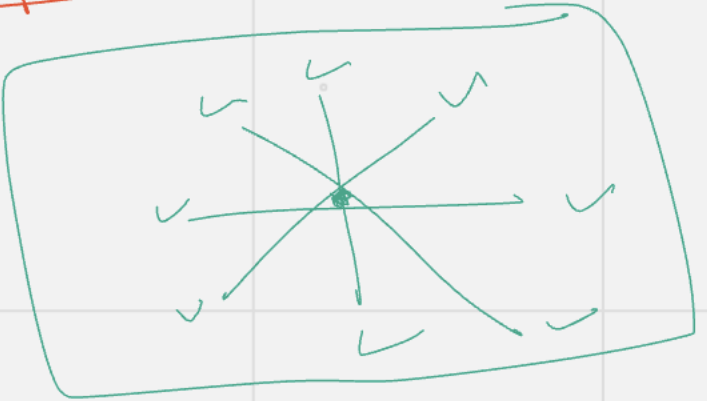
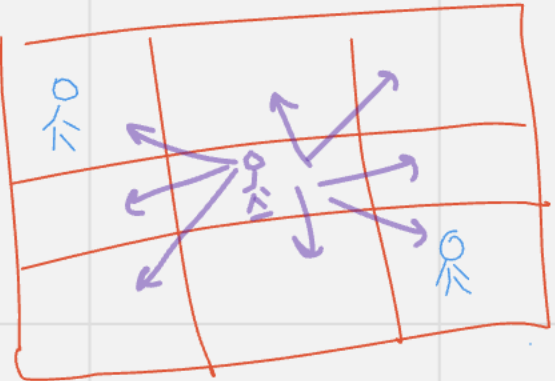
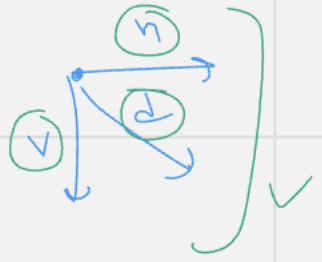
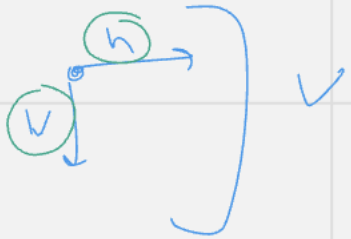
1 + 96

a 7

a

1 → a

3x3



8 dir

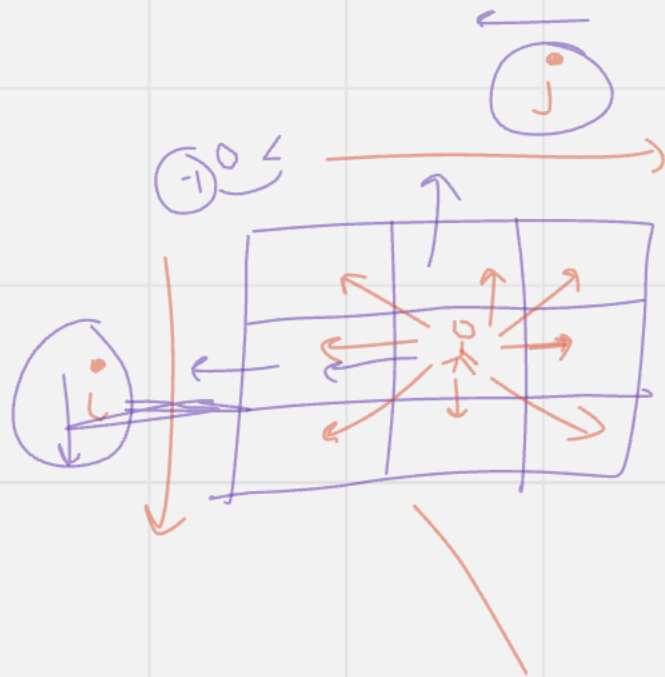


native
8 rec



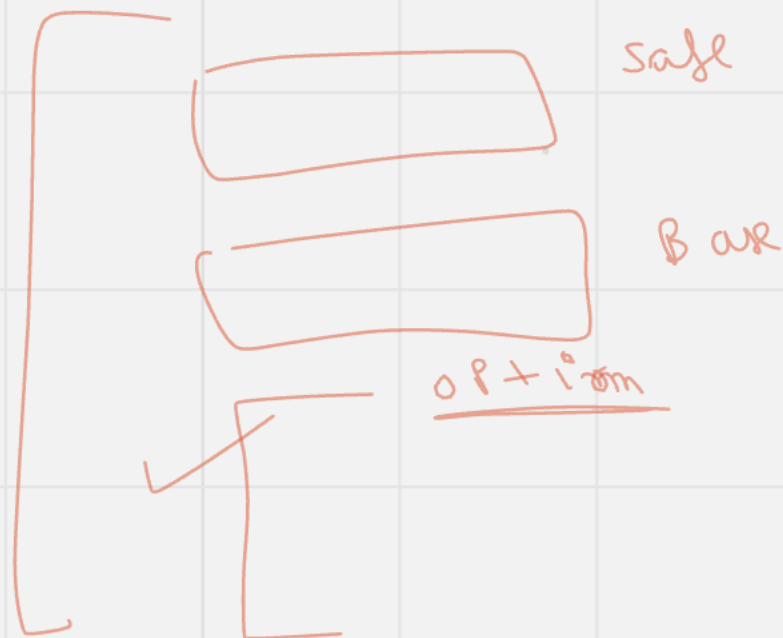
081

088



i [0, 1, 1, 1, 1, 0, -1, -1, -1]

j [1, 1, 0, -1, -1, -1, 0, 1]





↓↓↓
TLE



TLE

T	T	f
T	T	f
f	f	f

A 3x3 table with orange borders. The top two rows have 'T' in the first two columns, which are crossed out with blue ink. The bottom row has 'f' in all three columns. A blue circle is drawn around the top-left 'T', and a blue arrow points from it to the top-right 'T'. Another blue arrow points from the top-right 'T' to the middle-right 'f'. A blue 'X' is written to the left of the top-left 'T'.


```
static int iDir[] = {0, 1, 1, 1, 0, -1, -1, -1};
static int jDir[] = {1, 1, 0, -1, -1, -1, 0, 1};
```

```
static boolean isSafe(int i, int j, int n, int m) {
    // safe cond
    if(i > n || j > m || i < 0 || j < 0) {
        return false;
    }
    return true;
}
```

```
static int countAllPath(int i, int j, int n, int m, boolean vis[][])
```

```
{
    if(i == n && j == m) {
        return 1;
    }

    int ans = 0;
    for(int k = 0; k < 8; k++) {
        int newI = i + iDir[k];
        int newJ = j + jDir[k];
        if(isSafe(newI, newJ, n, m) && !vis[newI][newJ]) {
            vis[newI][newJ] = true;
            ans += countAllPath(newI, newJ, n, m, vis);
            vis[newI][newJ] = false;
        }
    }

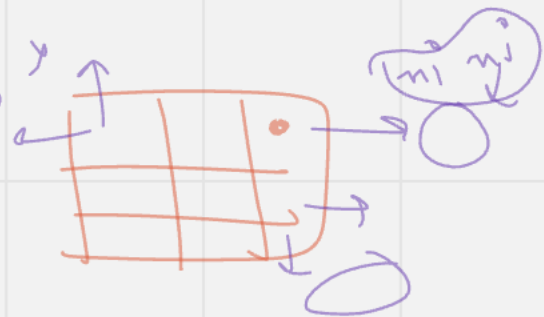
    return ans;
}
```

base

cannot

safe

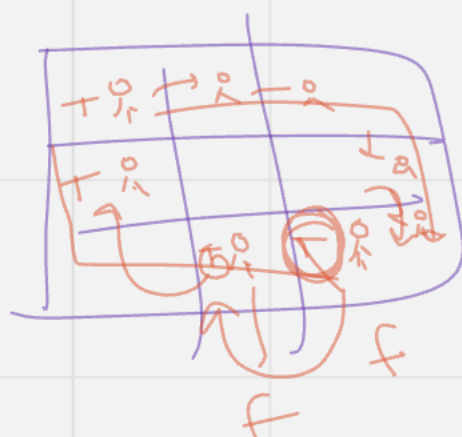
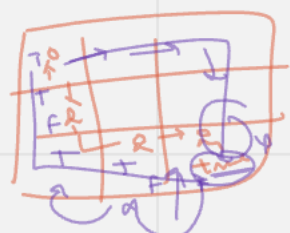
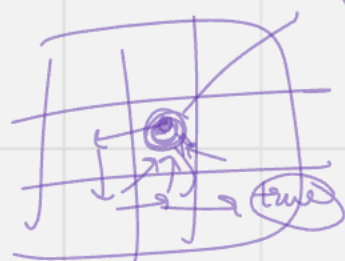
base



I
Safe ±
II
vis

! U
false

t



one more