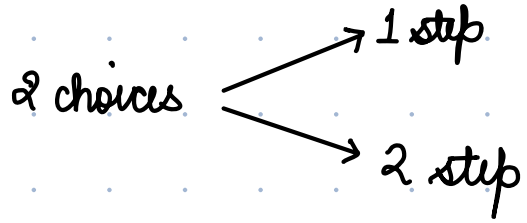
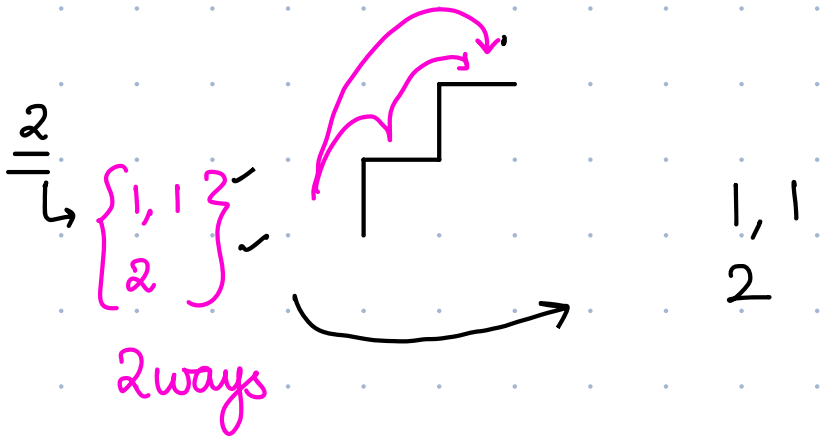
Today's Agenda:

Print paths in staircase
Print path in matrix
Shortest path

Q. Print paths in staircase



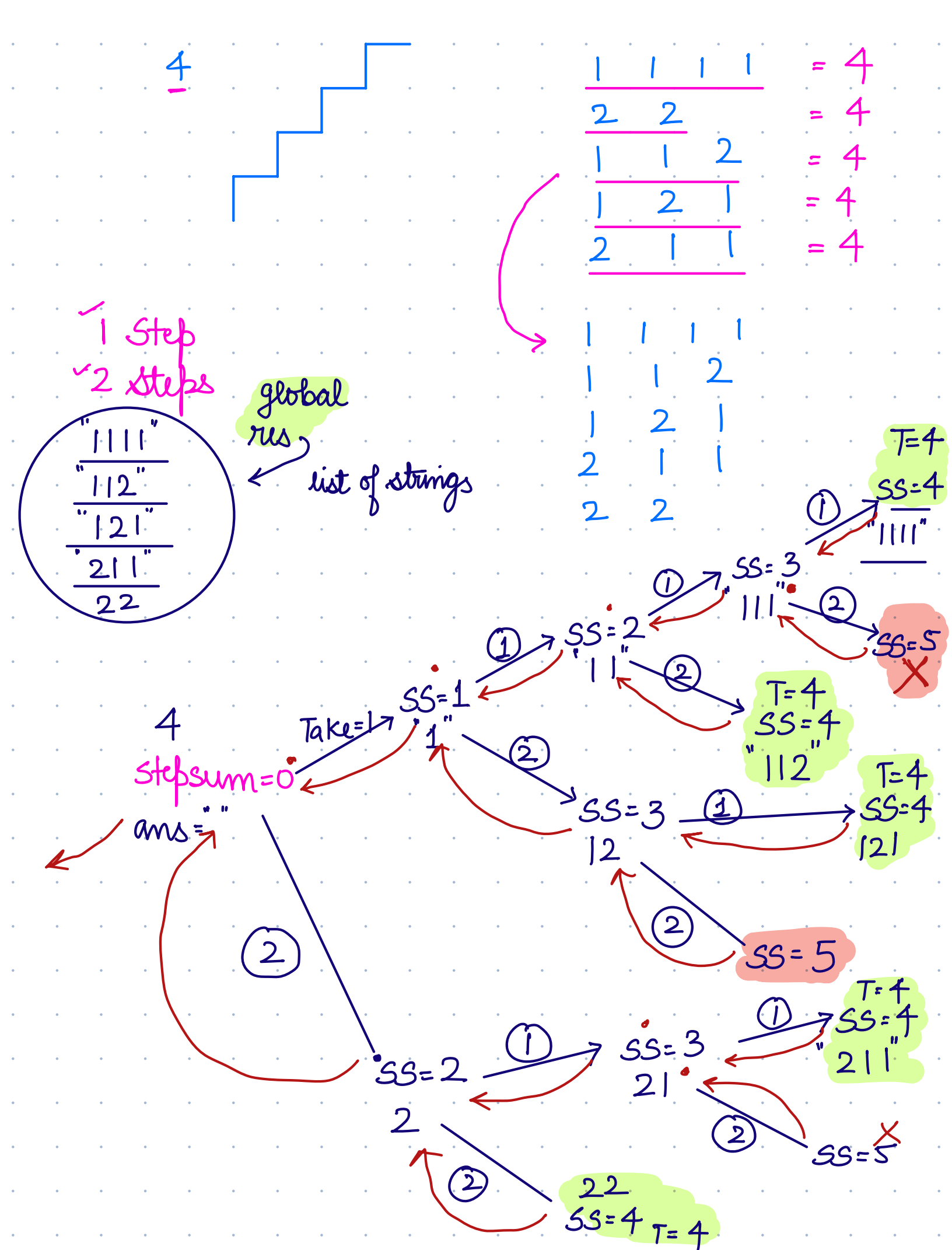


you need to return all distinct ways to climb to the top in lexicographical order

dictionary order

aero
aeroplane
apple
bow

aero
aeroplane
apple
bow


4
1 1 1 1 = 4
2 2 = 4
1 1 2 = 4
1 2 1 = 4
2 1 1 = 4
1 1 1 1
1 1 2
1 2 1
2 1 1
2 2
1 Step
2 steps
"1111"
"112"
"121"
"211"
22
global res
list of strings
4
stepsum=0
ans=""
Take=1
Take=2
SS=1
"1"
SS=2
"11"
SS=3
"111"
SS=4
"1111"
T=4
SS=5
X
T=4
SS=4
"112"
SS=3
12
T=4
SS=4
121
SS=5
SS=2
2
SS=3
21
T=4
SS=4
"211"
SS=5
X
22
SS=4
T=4

Code :

Parameters : int stepsum
int target
String ans

---

global variable : List of strings res

Recursive calls → 2 → 1 step
2 step

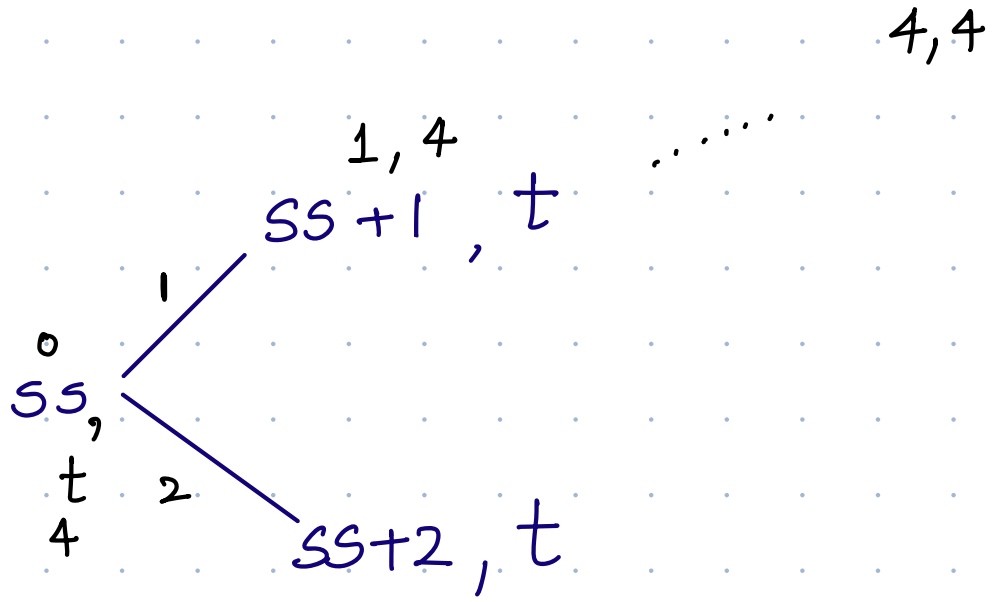
Return type = void

Code:

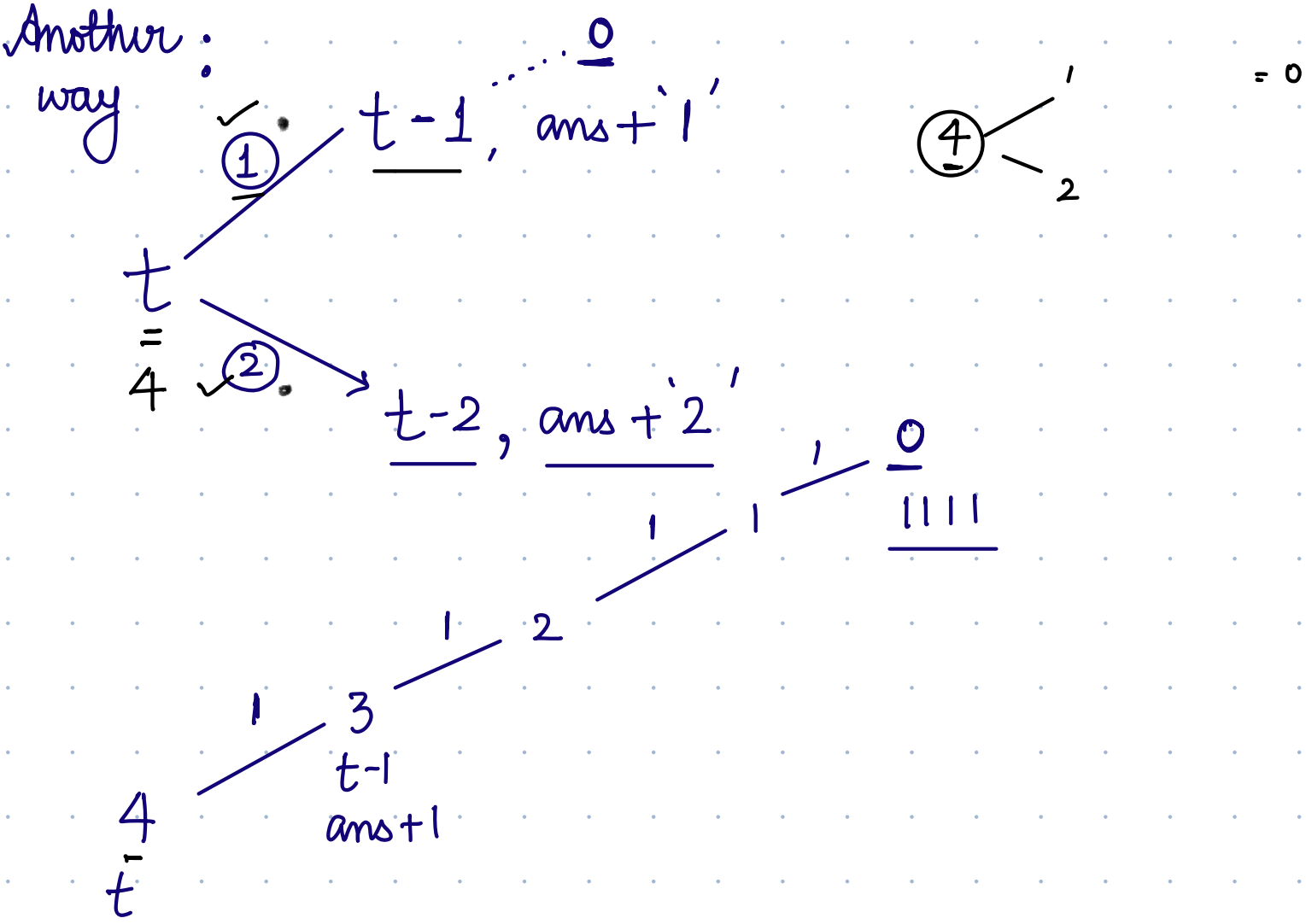
```
List of string  res

void  Allpaths ( int stepsum, int target,
                 string ans) {
    if( stepsum == target ) {
        res.add (ans)
        return;
    }
    Allpaths ( stepsum+1, target , ans + "1");
    if( stepsum+2 <= target) {
        Allpaths ( stepsum+2, target , ans + "2");
    }
}
```

```
main() {
    Print res list
}
```


4,4
1,4
ss+1, t
1
0
ss, t
4
2
ss+2, t

Another way:


t-1, ans+'1'
0
1
t
=
4
2
t-2, ans+'2'
0
1111
1
1
2
3
t-1
ans+1
4
t

4
1
2
= 0

Ques 2 You are given a matrix A × B

You need to print all paths to go from

Top left (0,0) to bottom Right (A-1, B-1)

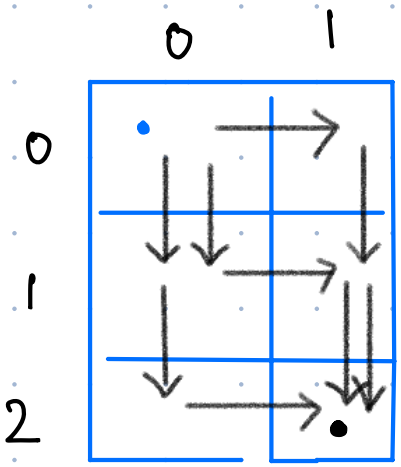
2 ways

- Down ("D")
- Right ("R")

D or R

D ← small

- Print all paths in lexicographical order.



A = 3, B = 2

↓ ↓

2 1

0,0 to 2,1

{ D D R

D R D

R D D }

Choices/Recursive Calls = 2

i, j → R(i, j+1)

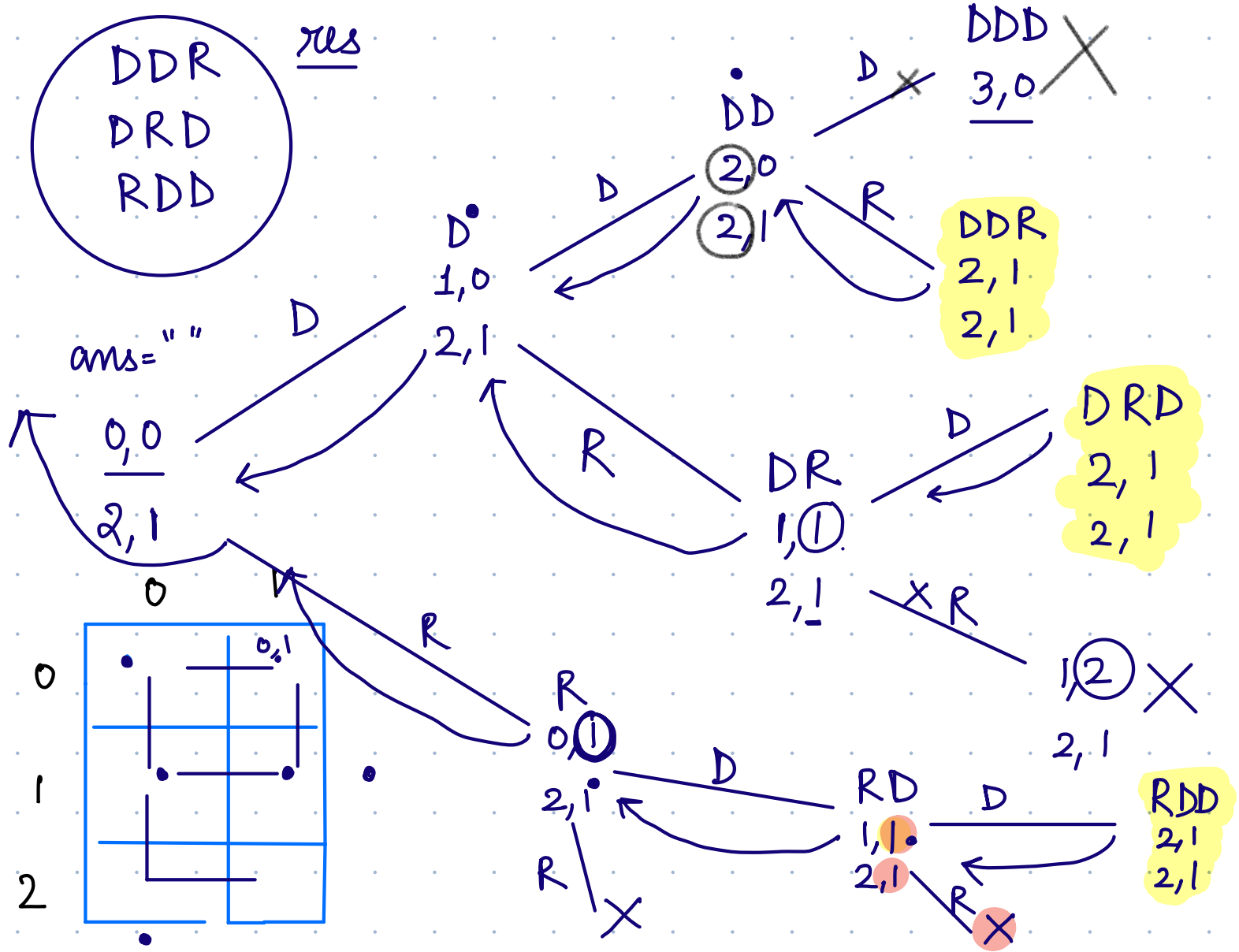
↓ D

(i+1, j)

1) Tracing → D R

2) Code

res: DDR, DRD, RDD



Parameters → i , j
ti , tj
String ans

List of strings res

Recursive calls → 2

Return type → void

Code:

List of strings res

```
void print_maze_path( int i, int j,
    int ti , int tj , String ans ) {
  if( i == ti && j == tj)
      res. add (ans) , return
```

```
if(i+1 <= ti)
    print_maze_path(i+1, j, ti, tj, ans+"D");

if(j+1 <= tj)
    print_maze_path(i, j+1, ti, tj, ans+"R");
}
```

Q3 given M x N matrix ⟶ 1 or 0

1 means a path
0 means a hurdle

find the shortest path between source cell to destination cell.

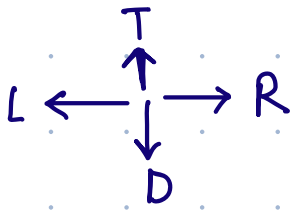
1)

	0	1	2	3
0	1	1	0	0
1	0	1	1	0
2	0	0	1	1
3	0	0	0	1

0,0 to 3,3

RDRDRD = 6 steps

We can move in any 4 directions



⇓

6 ans

2) 3 x 3

	0	1	2
0	1 →	1 →	1
1	1	0	1
2	1	1	1

0,0 to 0,2

RR

2 steps

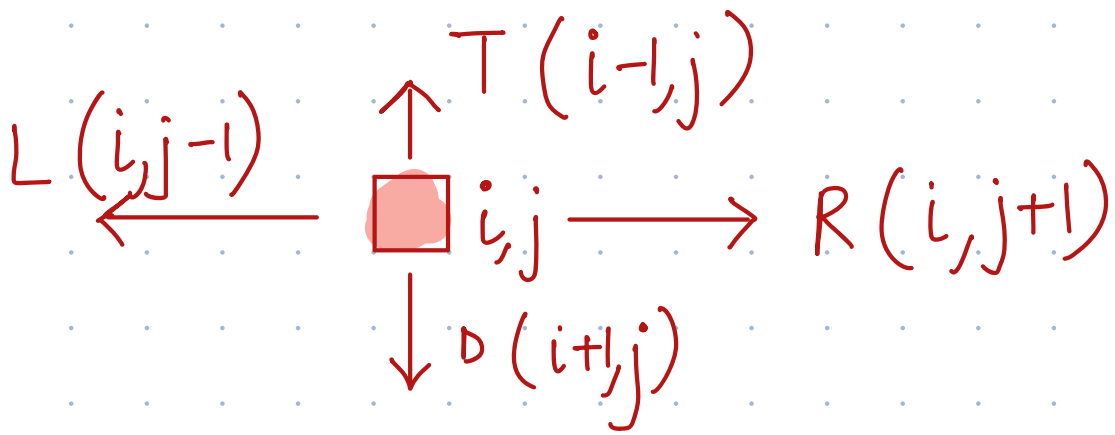
3)

0,0 to 2,2

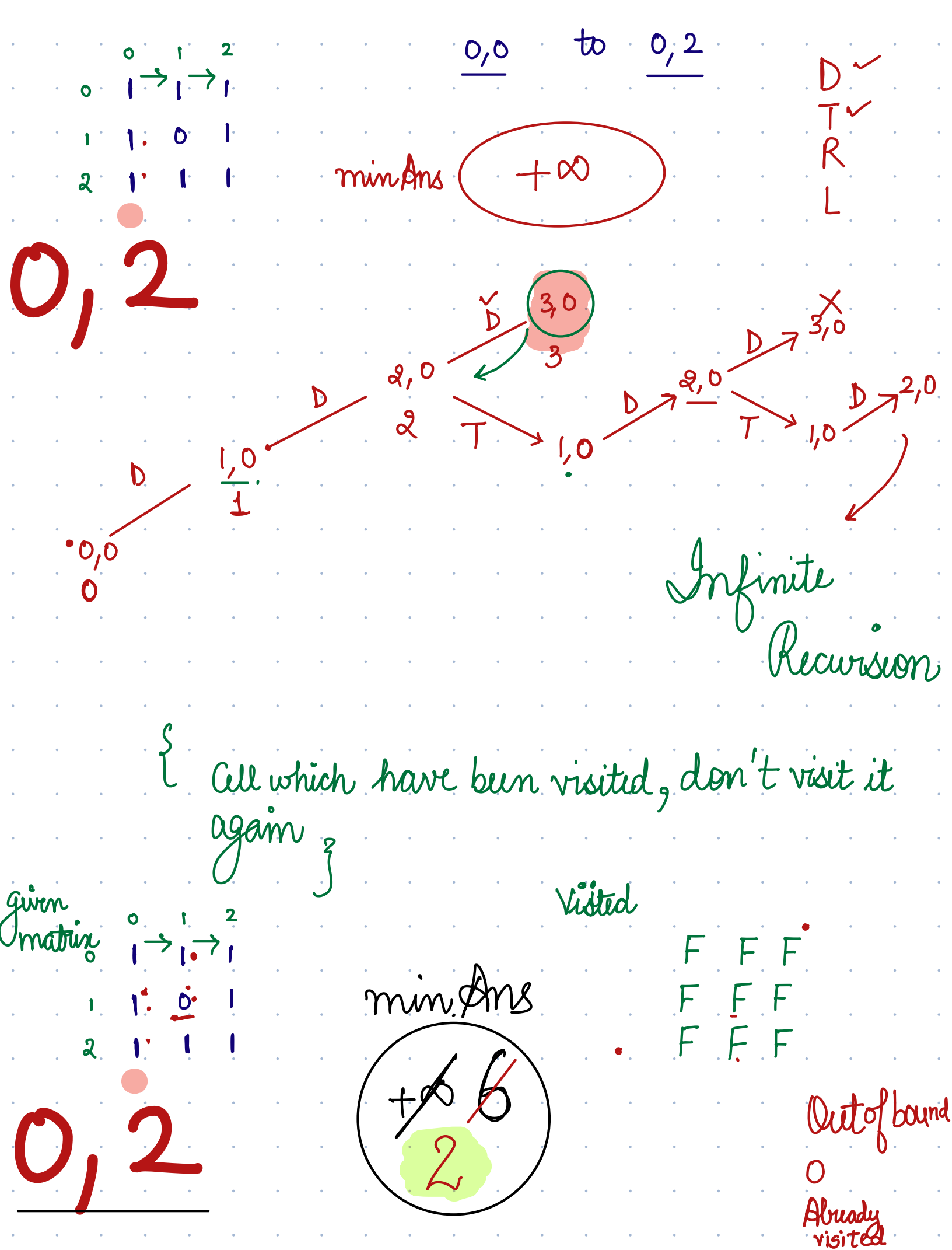
	0	1	2
0	1	0	1
1	1	0	1
2	1	0	1

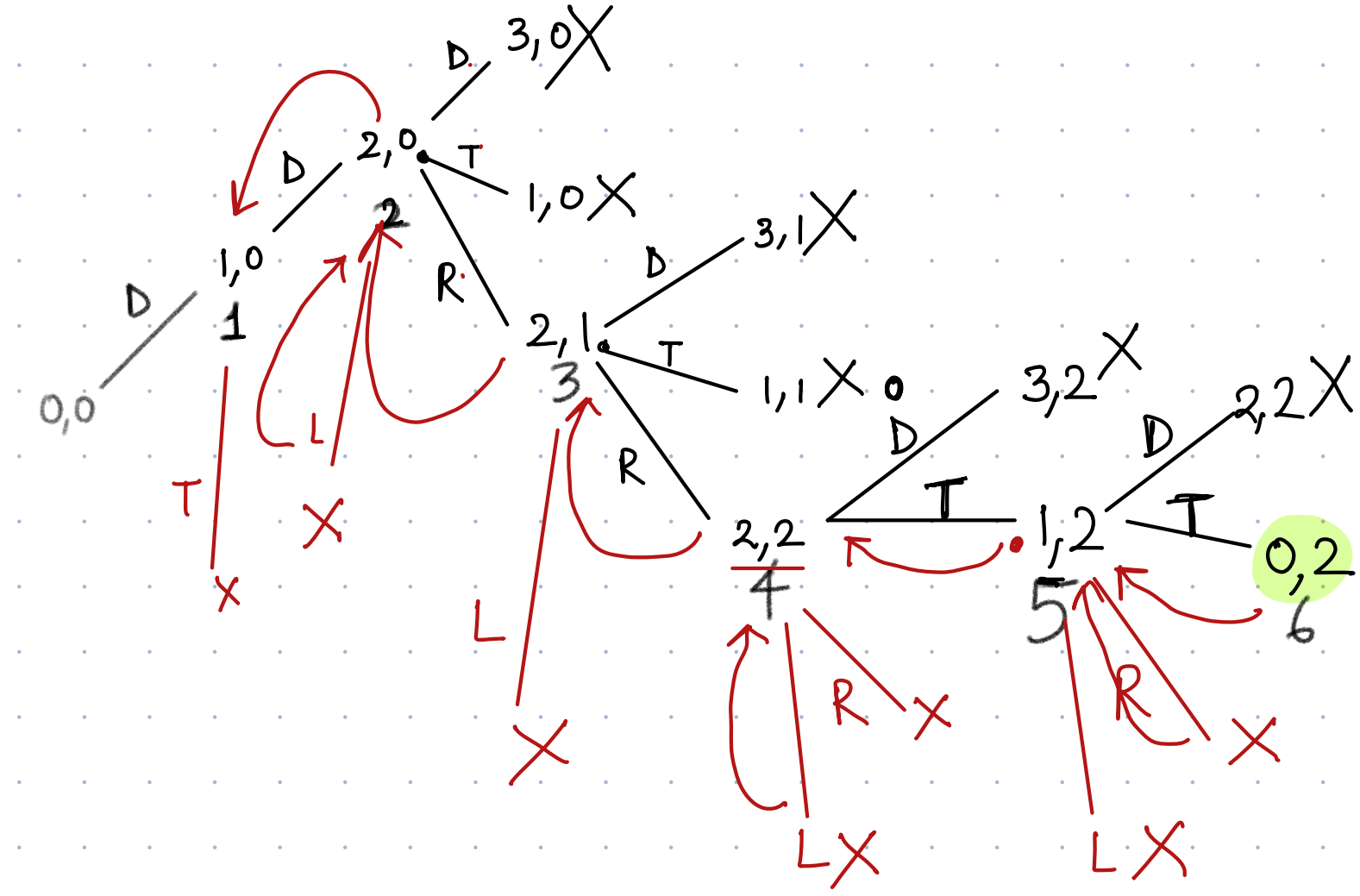
→ -1 ans

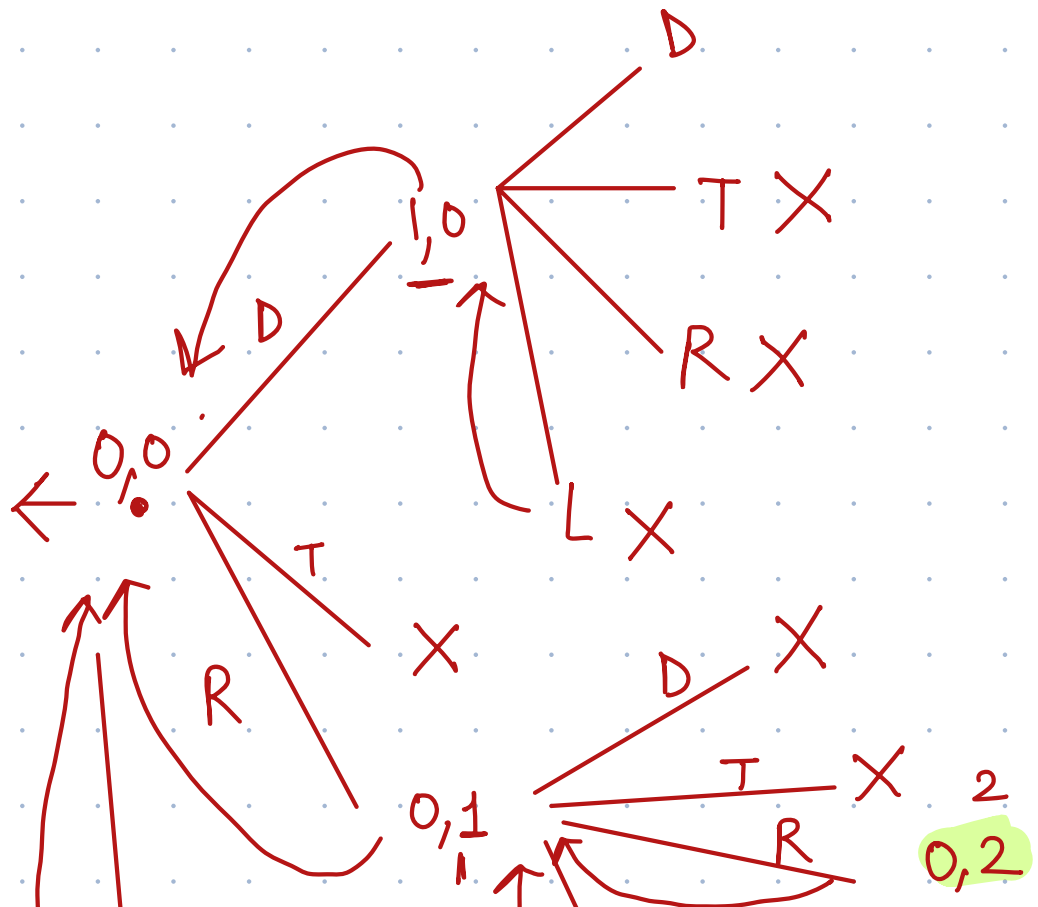
{No path exists}



Tracing


0,0 to 0,2
D ✓
T ✓
R
L
minAns +∞
0,2
3,0
3
2,0
2
1,0
1
0,0
0
Infinite Recursion
{ All which have been visited, don't visit it again }
given matrix
Visited
F F F
F F F
F F F
minAns
+∞ 6
2
0,2
Out of bound
0
Already visited


0,0
1,0
2,0
2,1
2,2
1,2
0,2
3,0
1,0
3,1
1,1
3,2
2,2
D
T
R
L
X
1
2
3
4
5
6


0,0
1,0
0,1
D
T
R
L
X
2

Out of bound

0

[Already visited]

cell [i] [j]

cell [i+1] [j]

1

i+1 < no of rows

Should be un visited

Parameters —

visited [ ][ ] matrix

cell [ ][ ]

i, j

ti, tj

---

global variable = minAns = +∞

---

void

---

4 choices = 4 recursive calls

Code :

```
int minAns = +∞

void findShortestPath(boolean[][] visited,
int[][] cell, int i, int j, int ti, int tj, int myAns){

    if (i==ti && j==tj) {
        minAns = min(myAns, minAns);
        return;
    }

    if( invalid(visited, cell, i, j) == true) {
        return;
    }
```

```
visited[i][j] = true;

//D
findShortestPath(visited, cell, i+1, j, ti, tj
                 , myAns+1);

//T
findShortestPath(visited, cell, i-1, j, ti, tj
                 , myAns+1);
//R
findShortestPath(visited, cell, i, j+1, ti, tj
                 , myAns+1);
//L
findShortestPath(visited, cell, i, j-1, ti, tj
                 , myAns+1);

visited[i][j] = false

}
```

N → rows , M → cols

```
boolean invalid (visited, cell, i, j) {
```

3 cases

```
// visited[i][j] → true

// cell[i][j] == 0 → true

// i < 0 || i >= N || j < 0 || j >= M → true

return false
}
```