

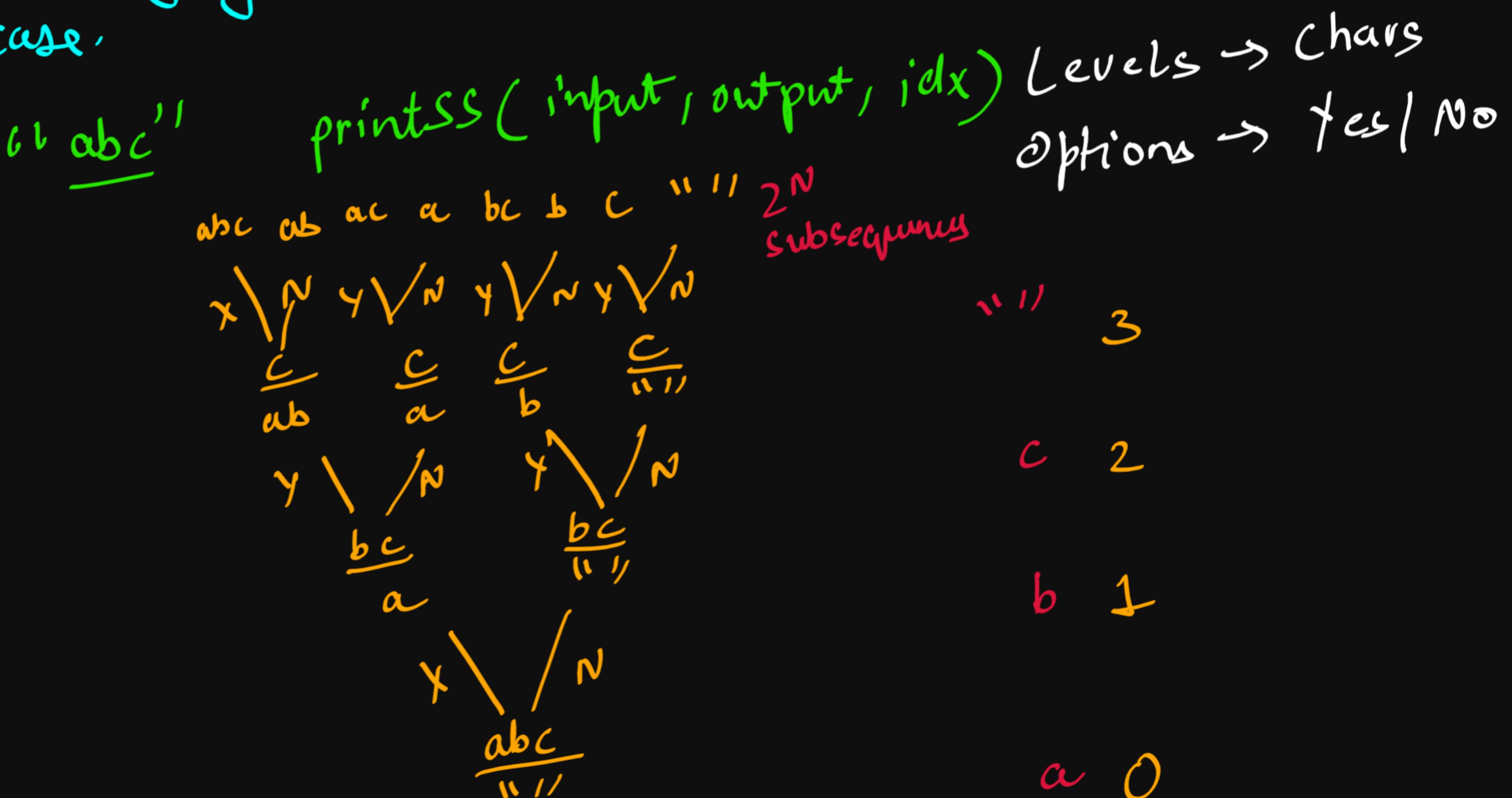
Recursion on the Way Up (Print Variation)

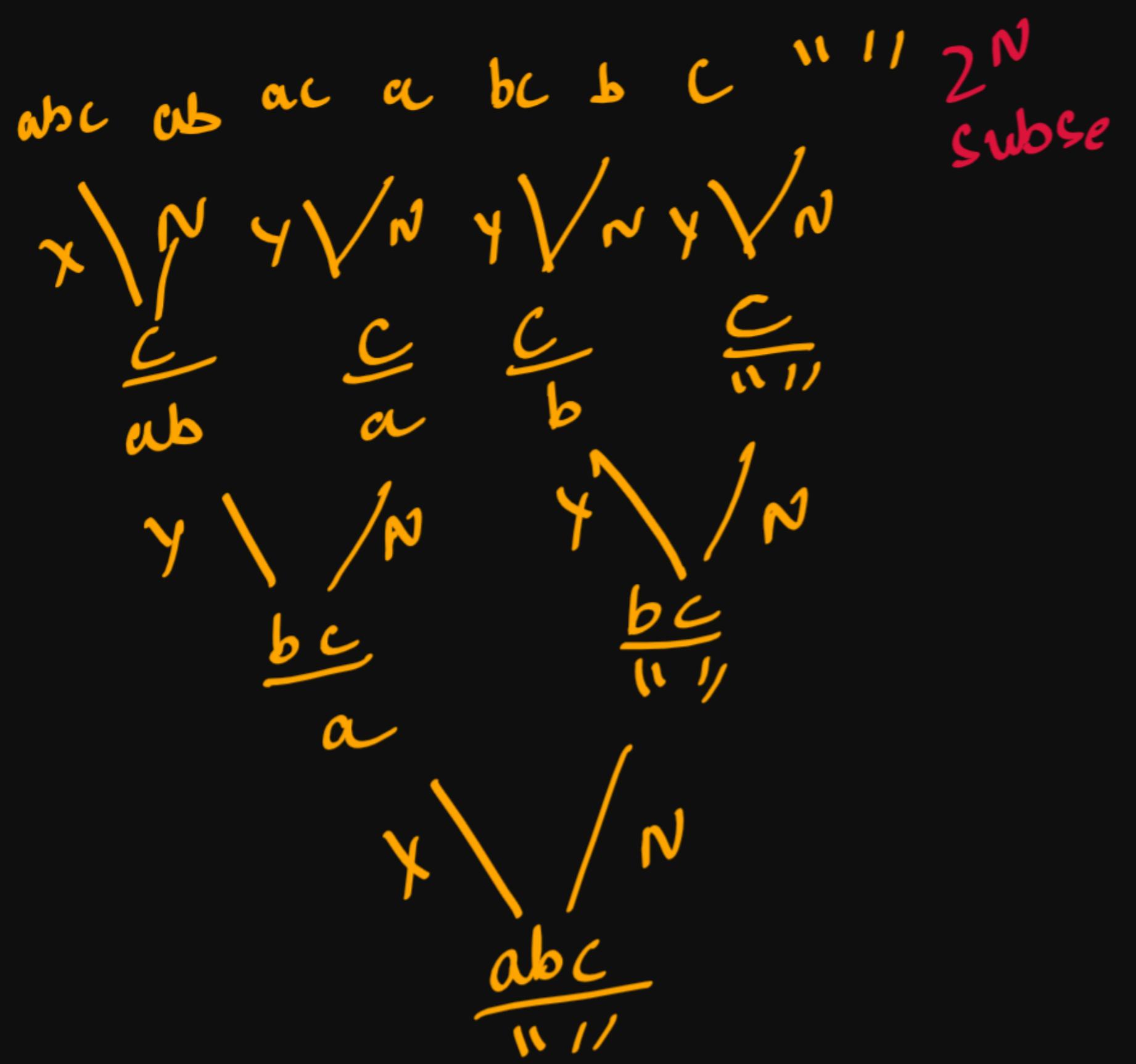
① Print Subsequence

$\{a, b, c\} \xrightarrow{\text{Subsets}} \emptyset, \{a\}, \{b\}, \{c\}, \{a, b\}, \{a, c\}, \{b, c\}$
 $"abc" \xrightarrow{} "", "a", "b", "c", "ab", "ac", "bc", "abc"$

→ In print category, biggest ans/ highest width will lie in the base case.

Num = inp
Den = out





→ This is the expected combination also
 " " 3
 c 2
 b 1
 a 0

So, basically, apply the
 Yes call first & then the
 No call.

```
public static void main(String[] args) throws Exception {
    Scanner scn = new Scanner(System.in);
    String str = scn.next();
    printSS(0,str,"");
}

public static void printSS(int idx,String str, String ans) {

    if(idx == str.length()) {
        System.out.println(ans);
        return;
    }

    printSS(idx + 1,str,ans + str.charAt(idx)); //yes call
    printSS(idx + 1,str,ans); //no call
}
```

Print Kpc

```

0 -> .;
1 -> abc
2 -> def
3 -> ghi
4 -> jkl
5 -> mno
6 -> pqrs
7 -> tu
8 -> vwx
9 -> yz
    
```

"179" → input string

Remembering the getchar

levels: one digit at a time

options: chars corresponding to each digit

Expectation: printKpc(0, "179", "") → all Kpc's

↑ ↑ ↑
idx input output

Base case:

if (idx <= str.length()) faith: printKpc(1, "179", "a") } "79" Kpc's will
 Sys.out.println();
 printKpc(1, "179", "b") } be appended to
 printKpc(1, "179", "c") this

Input = "179"

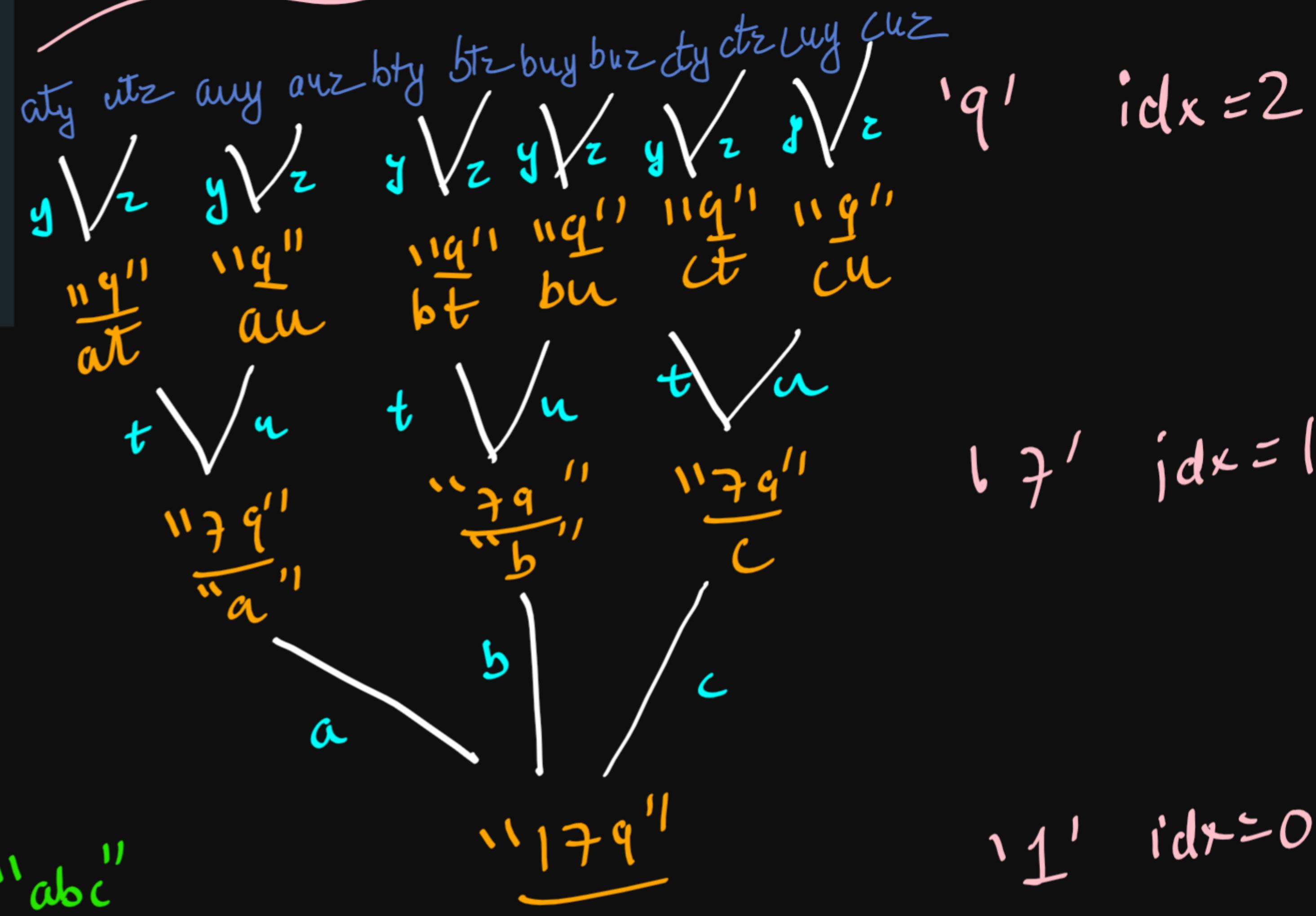
```
public static void printKPC(int idx, String input, String output) {  
    if(idx == input.length()) {  
        System.out.println(output);  
        return;  
    }  
  
    String str = keys[input.charAt(idx) - '0'];  
    for(int i=0; i<str.length(); i++) {  
        printKPC(idx + 1, input, output + str.charAt(i));  
    }  
}
```

0 -> .;
1 -> abc
2 -> def
3 -> ghi
4 -> jkl
5 -> mno
6 -> pqrs
7 -> tu
8 -> vwx
9 -> yz

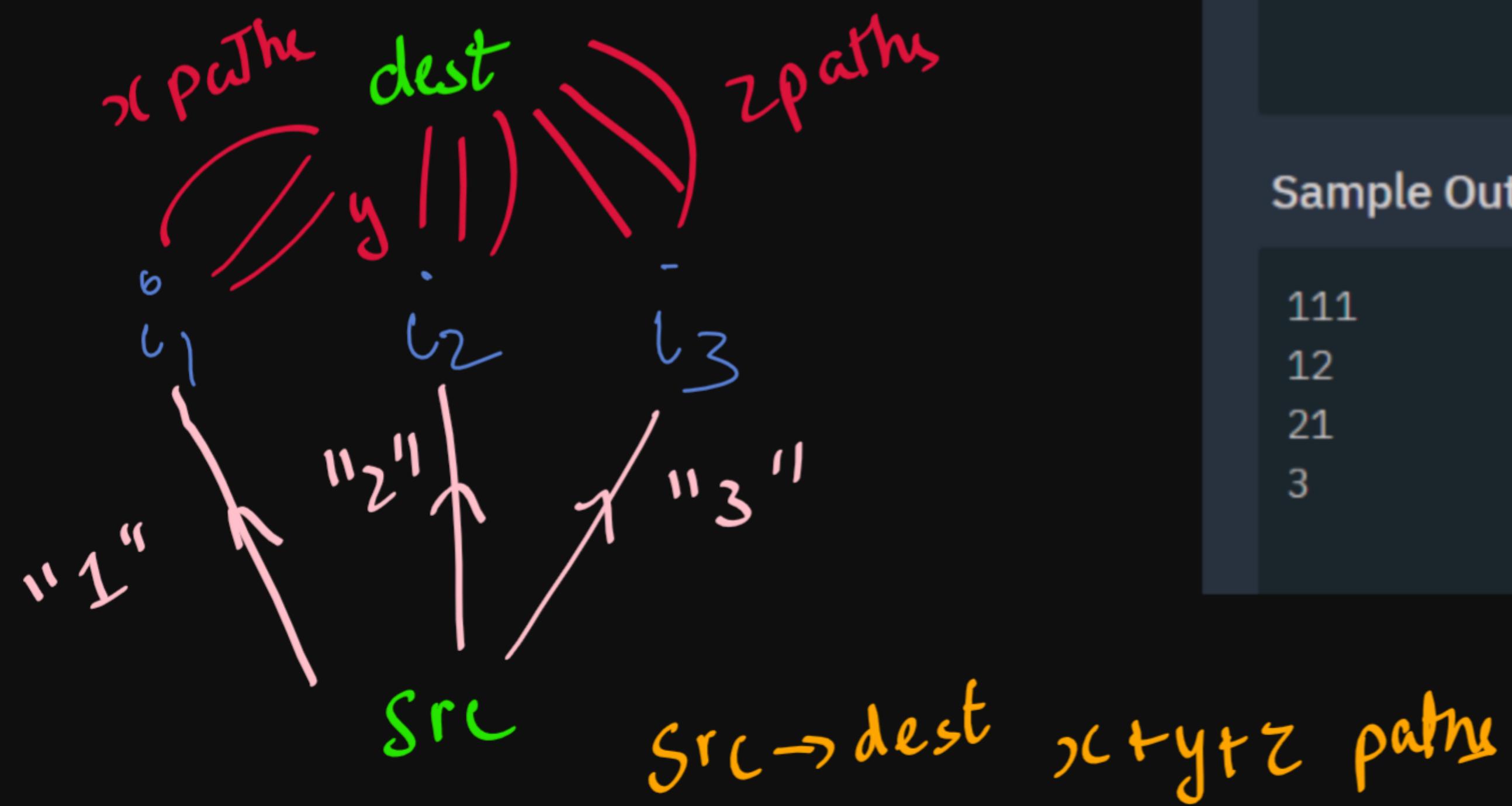
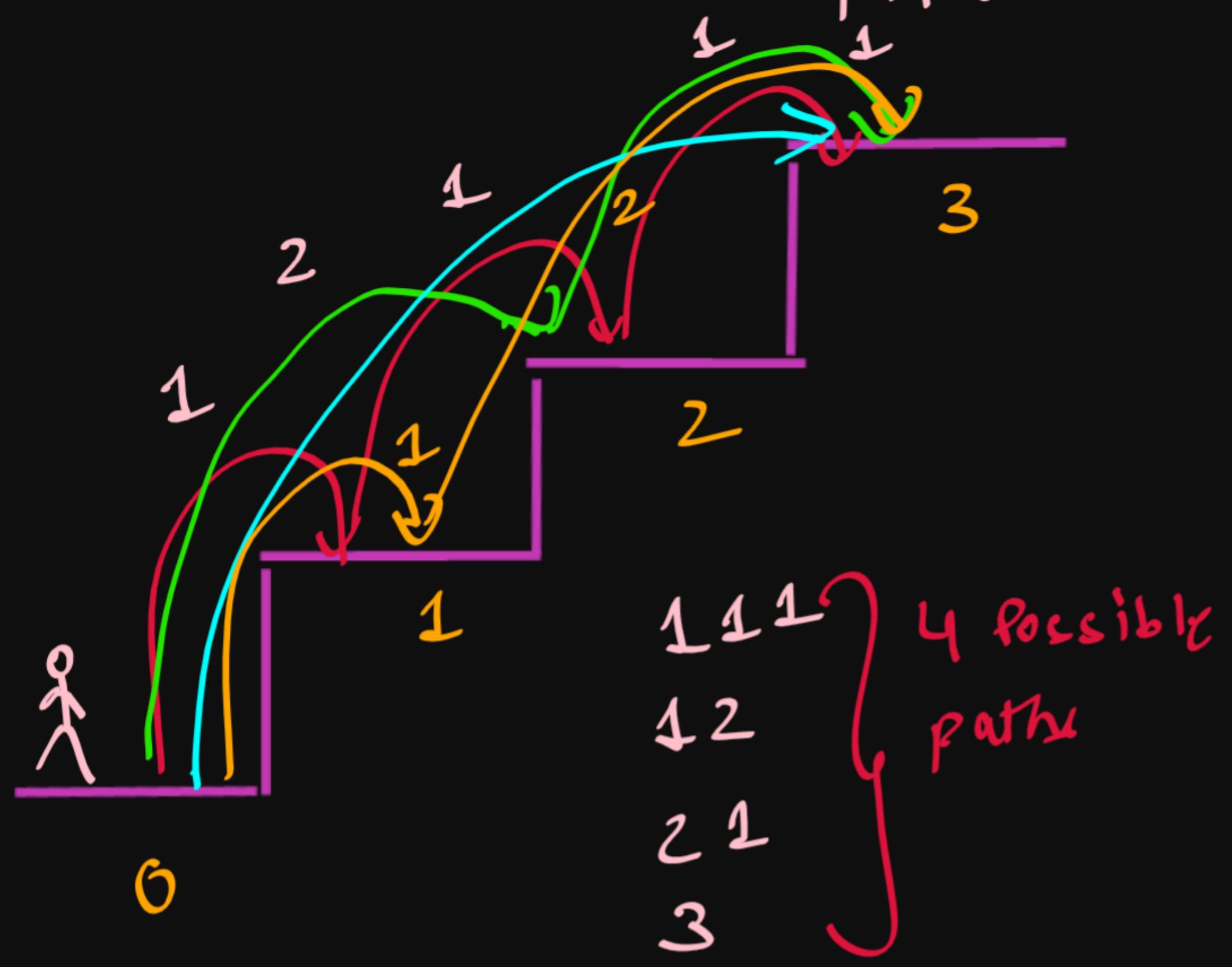
str = "abc"

final KPC's

(Base case has max width)



Print Stair Paths



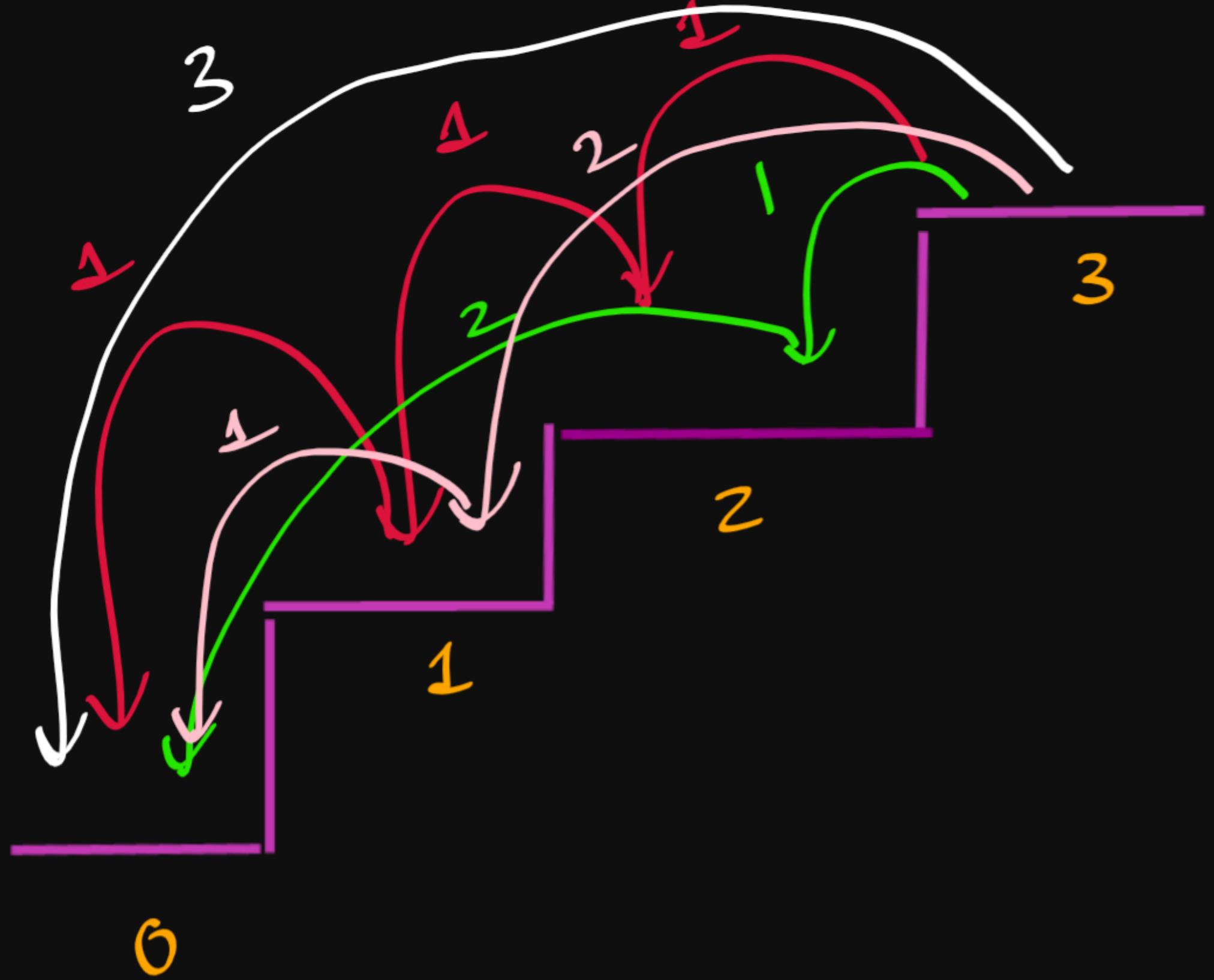
1. You are given a number n representing number of stairs in a staircase.
2. You are standing at the bottom of staircase. You are allowed to climb 1 step, 2 steps or 3 steps in one move.
3. Complete the body of `printStairPaths` function - without changing signature - to print the list of all paths that can be used to climb the staircase up.
Use sample input and output to take idea about output.

Sample Input

```
3
```

Sample Output

```
111  
12  
21  
3
```



```

public static void printStairPaths(int n, String path) {
    if(n < 0) {
        return;
    }

    if(n == 0) {
        System.out.println(path);
        return;
    }

    printStairPaths(n-1 ,path + "1");
    printStairPaths(n-2 ,path + "2");
    printStairPaths(n-3 ,path + "3");
}

```

```

public static void printStairPaths(int n, String path) {
    if(n == 0) {
        System.out.println(path);
        return;
    }

    if(n > 0) {
        printStairPaths(n-1 ,path + "1");
    }

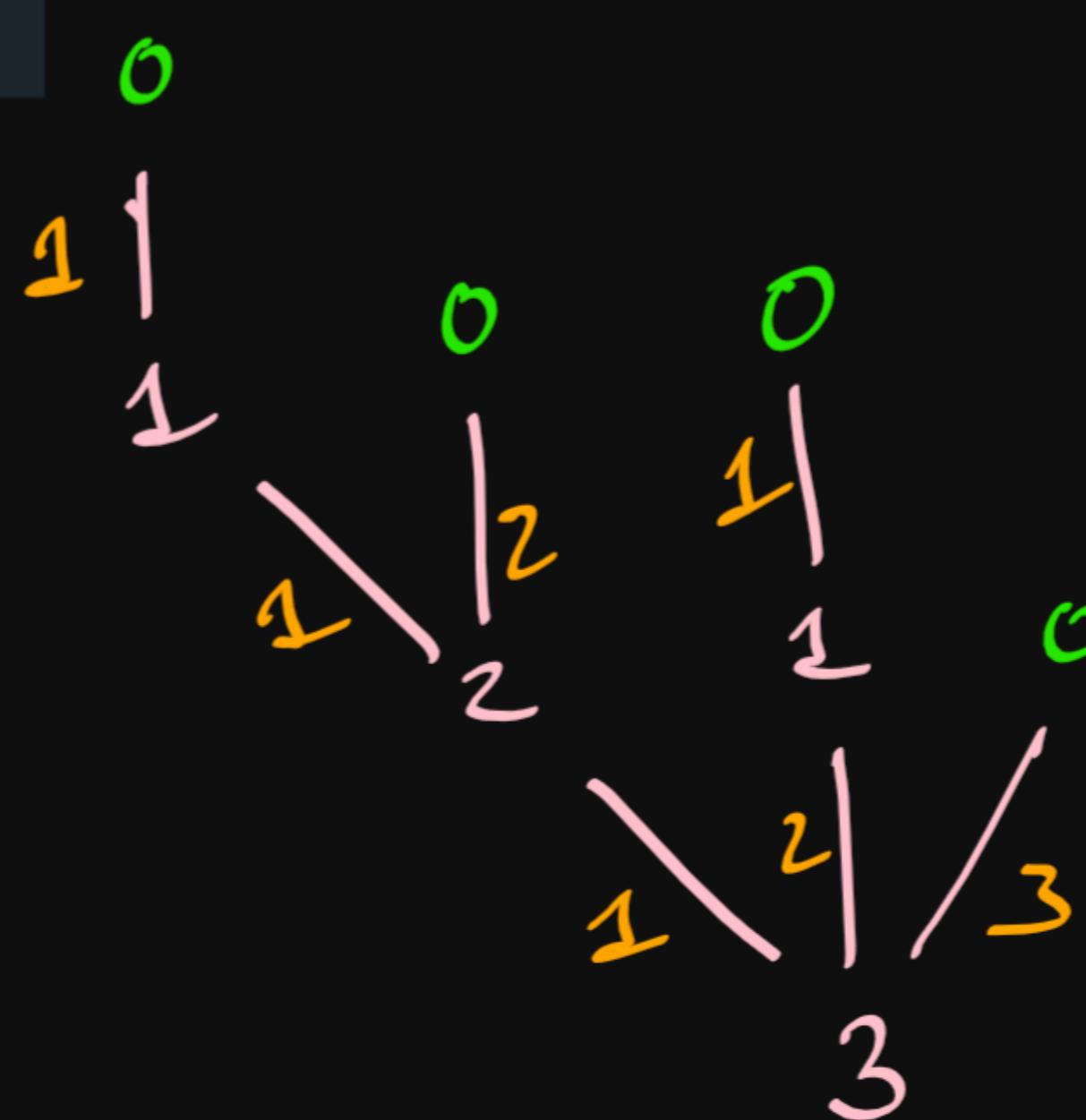
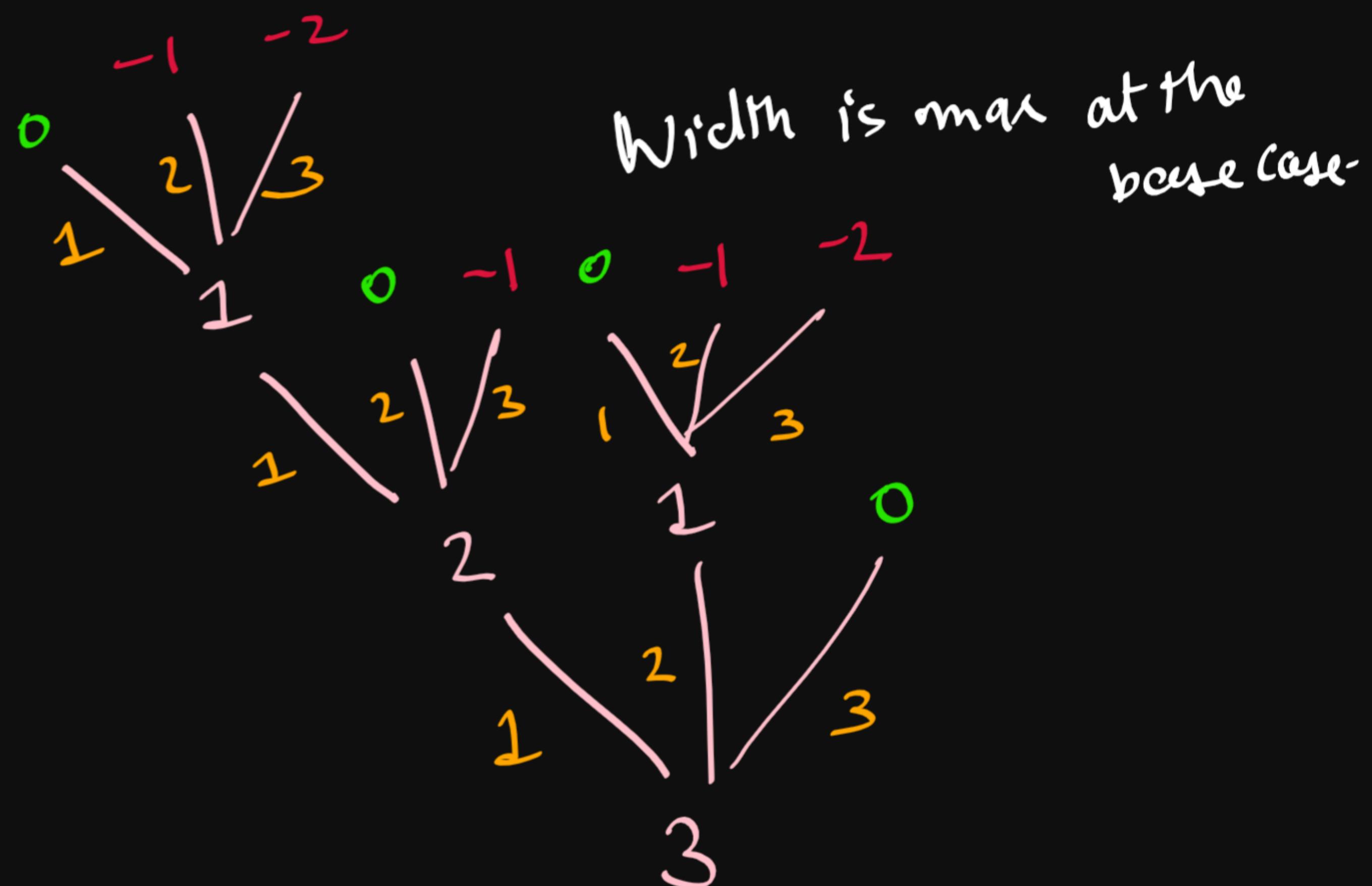
    if(n > 1) {
        printStairPaths(n-2 ,path + "2");
    }

    if(n > 2) {
        printStairPaths(n-3 ,path + "3");
    }
}

```

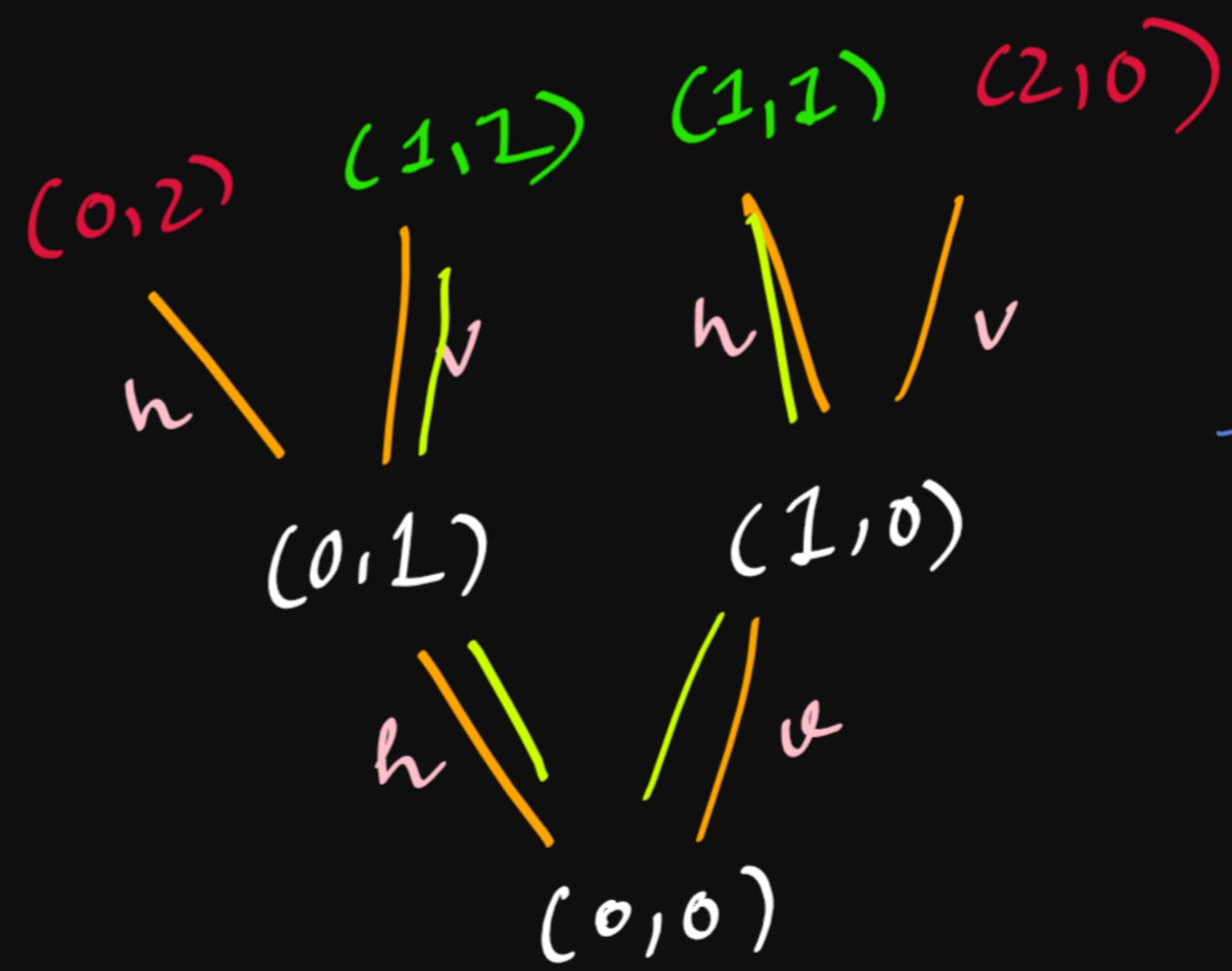
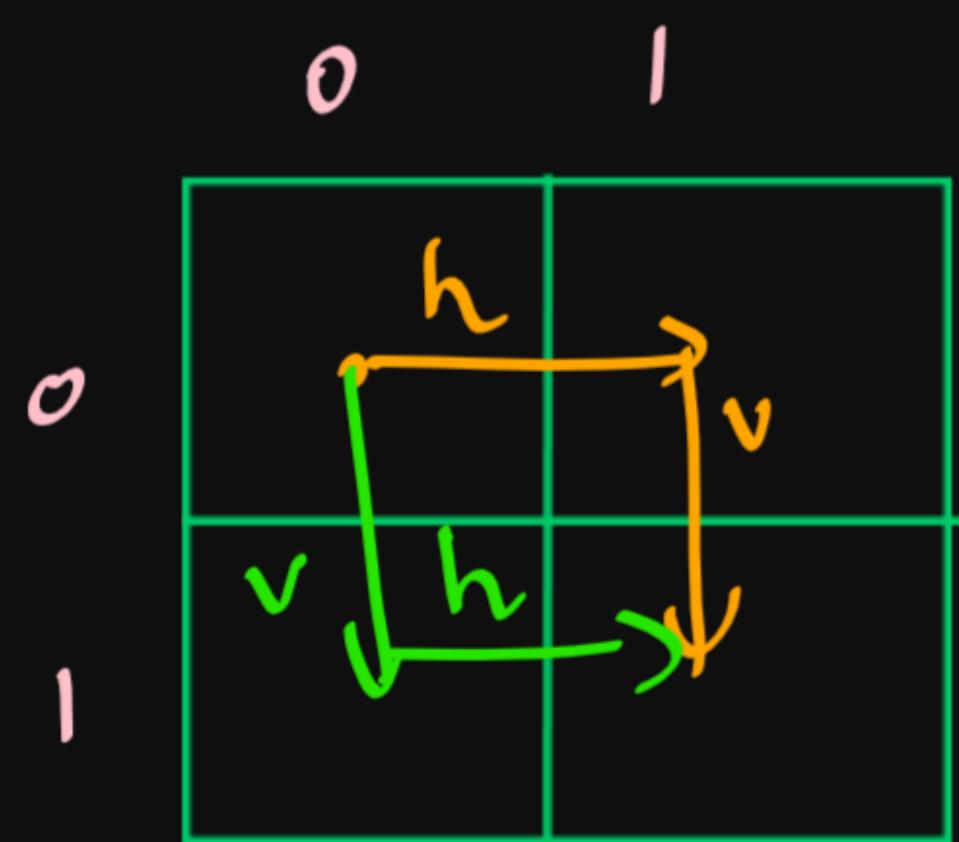
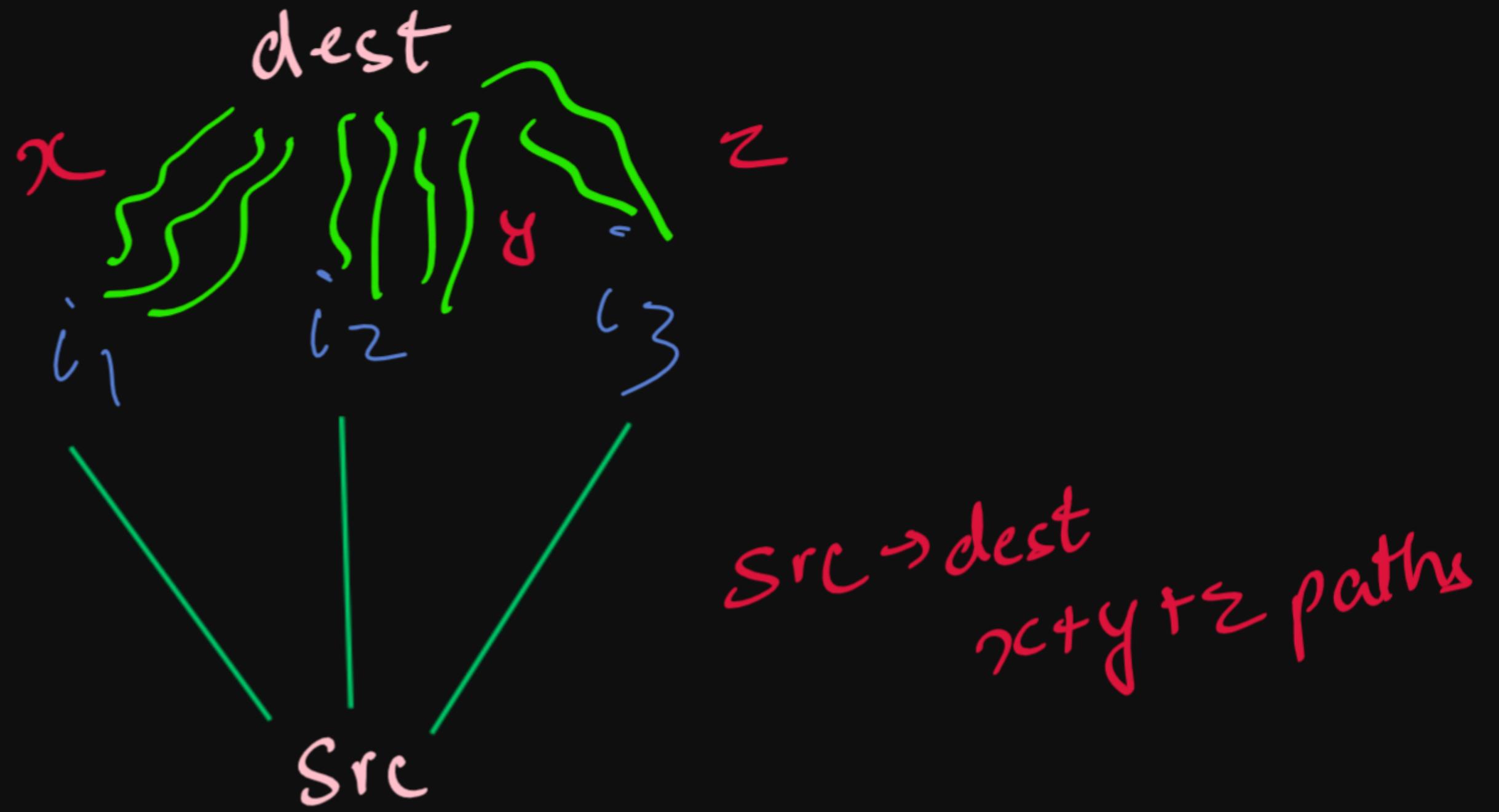
All paths which lead to a
-ve base case are valid

- Positive Base Case
- Negative Base Case



There is no call for
-ve base case

Print Maze Paths



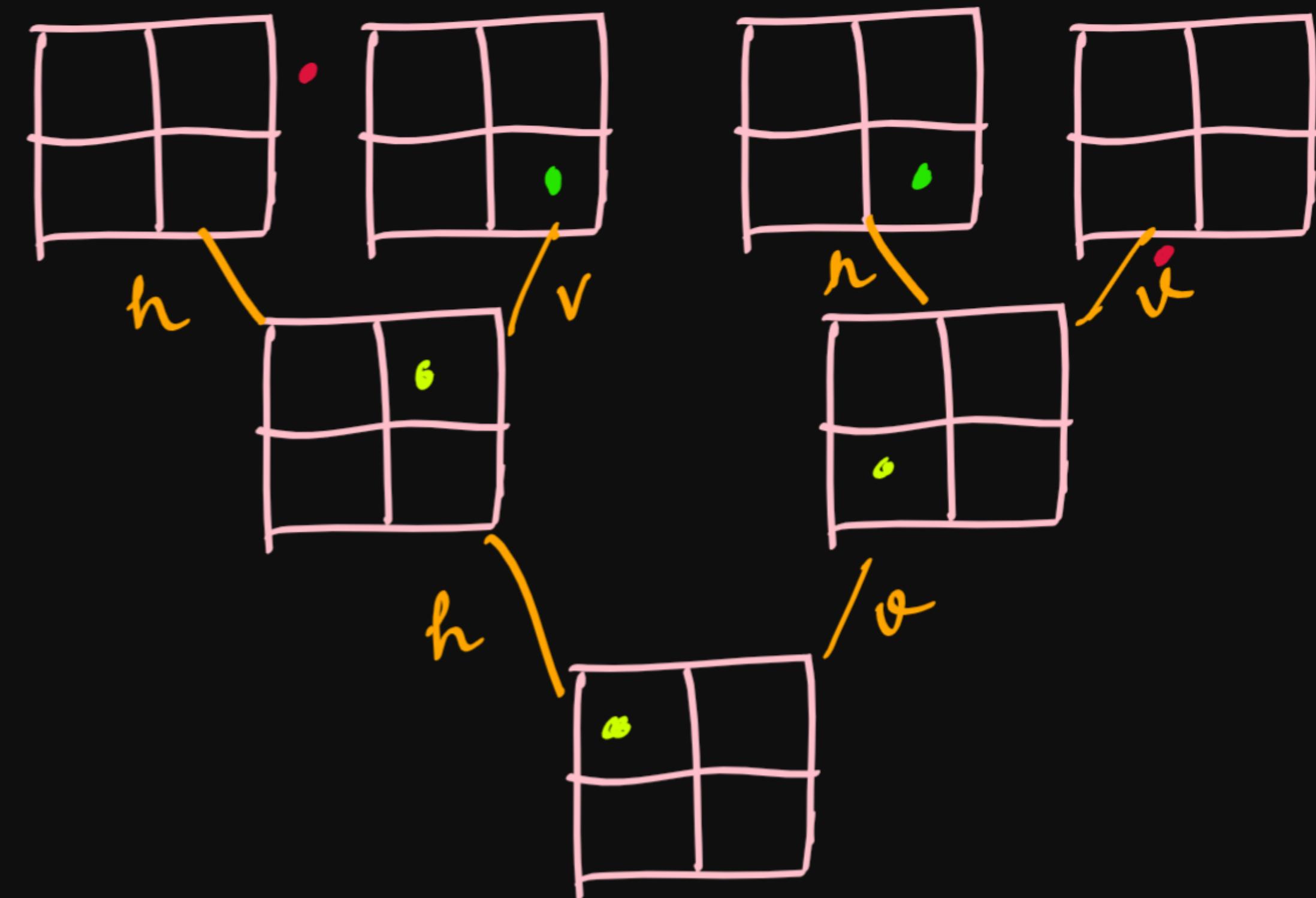
Without taking care of
- we base case calls -

Without smart call

```
// sr - source row
// sc - source column
// dr - destination row
// dc - destination column
public static void printMazePaths(int sr, int sc, int dr, int dc, String psf) {
    if(sr > dr || sc > dc) {
        return;
    }

    if(sr == dr && sc == dc) {
        System.out.println(psf);
        return;
    }

    printMazePaths(sr, sc + 1, dr, dc, psf + "h");
    printMazePaths(sr + 1, sc, dr, dc, psf + "v");
}
```

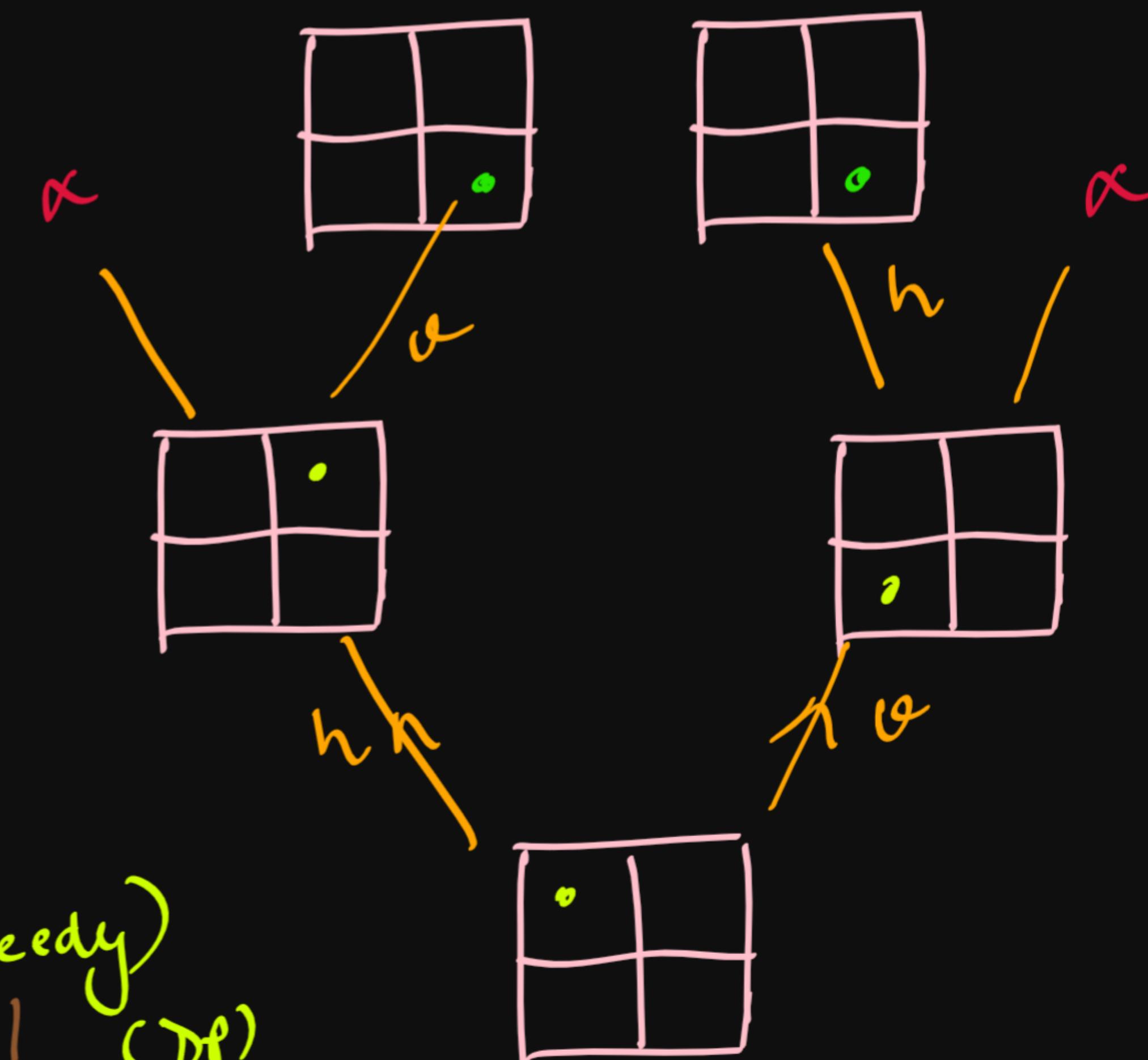


Smart calls

```
// sr - source row
// sc - source column
// dr - destination row
// dc - destination column
public static void printMazePaths(int sr, int sc, int dr, int dc, String psf) {
    if(sr == dr && sc == dc) {
        System.out.println(psf);
        return;
    }

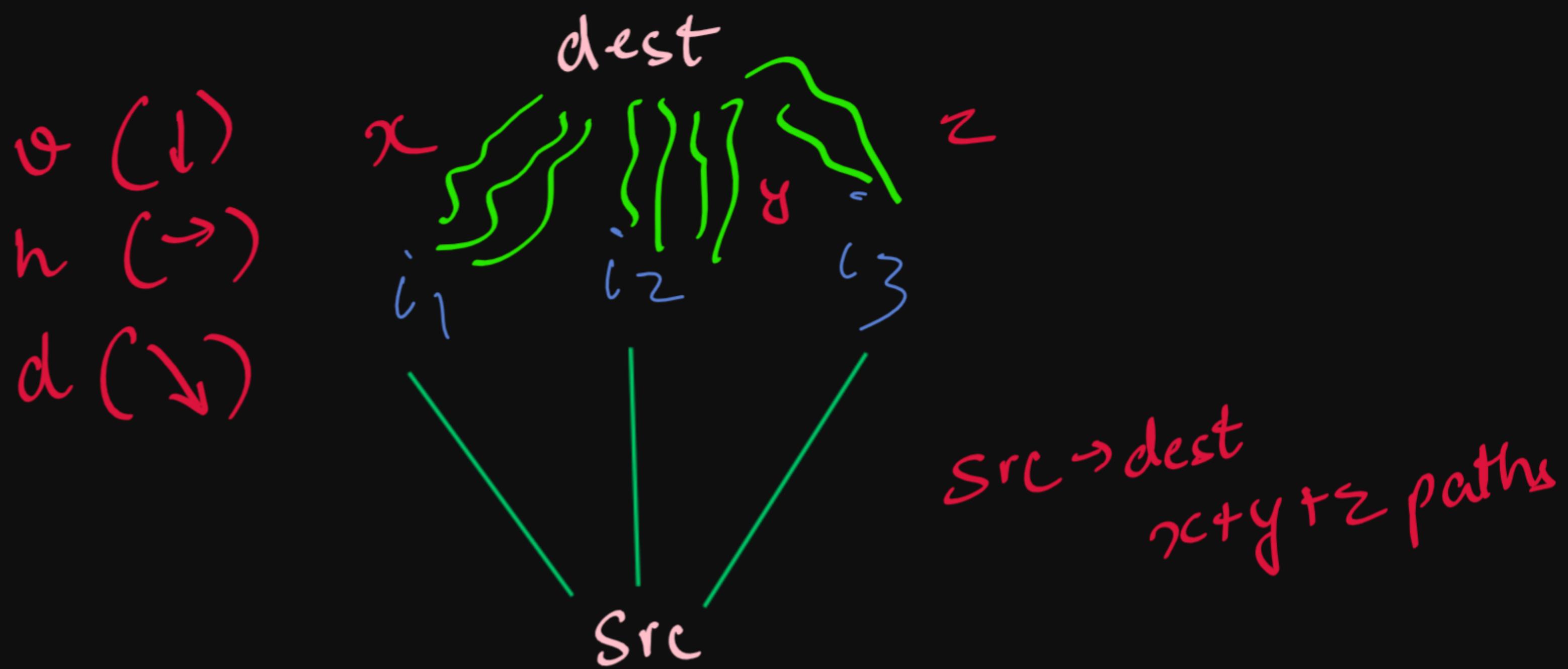
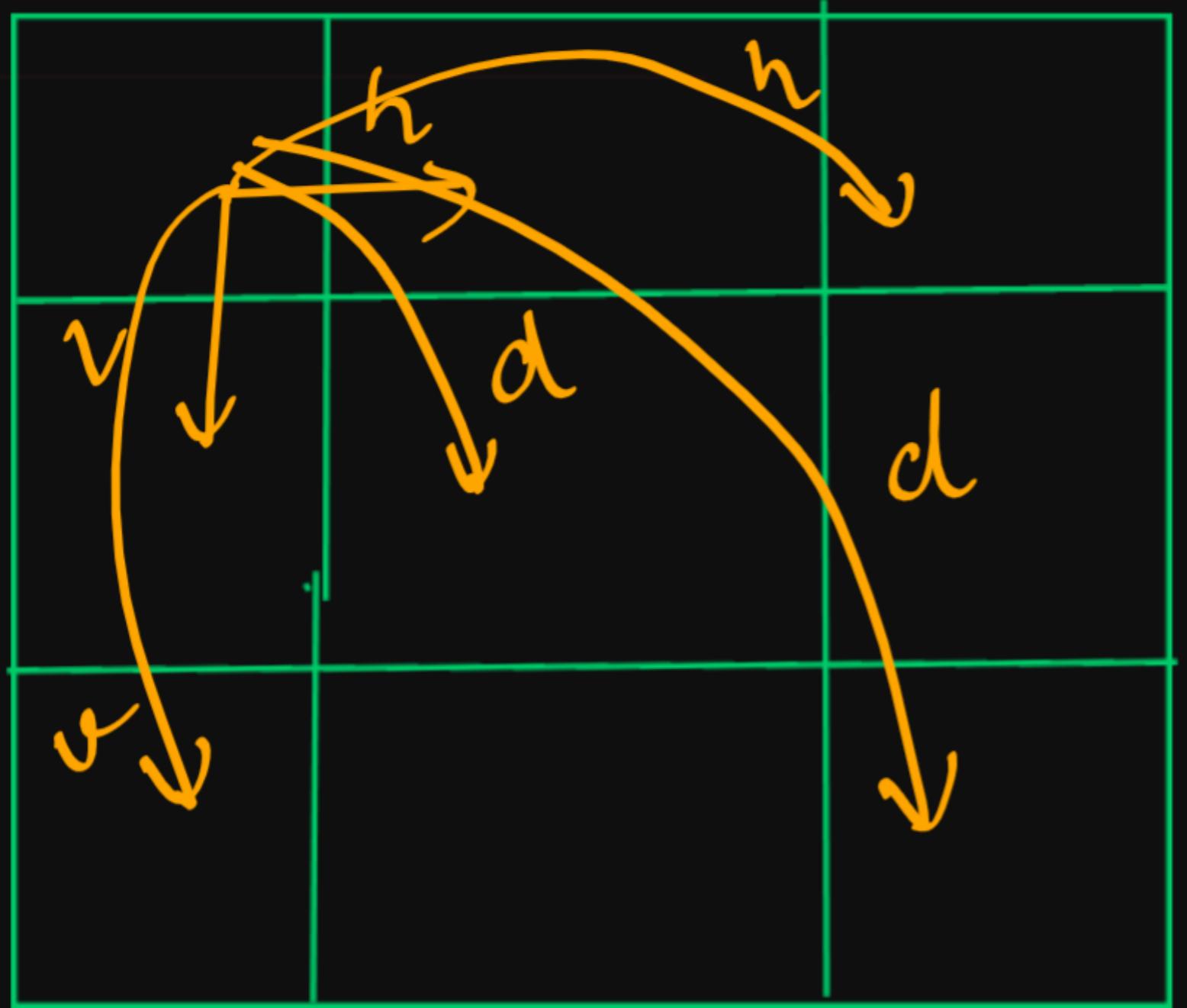
    if(sc < dc) {
        printMazePaths(sr, sc + 1, dr, dc, psf + "h");
    }

    if(sr < dr) {
        printMazePaths(sr + 1, sc, dr, dc, psf + "v");
    }
}
```



* Length of path = $rows - 1 + cols - 1$ (Greedy)
 ↳ No of paths = $\frac{(rows - 1 + cols - 1)!}{(rows - 1)! (cols - 1)!}$ (DP)

Print Maze Path With Jumps



+ve Base Case:

Destination Reached

-ve Base Case

Out of the Matrix

* We have to go to all the intermediates & apply recursion.

Max horizontal jumps: cols - 1

Max vertical jump: row - 1

Max diagonal jump: (stay inside both row & col)

```

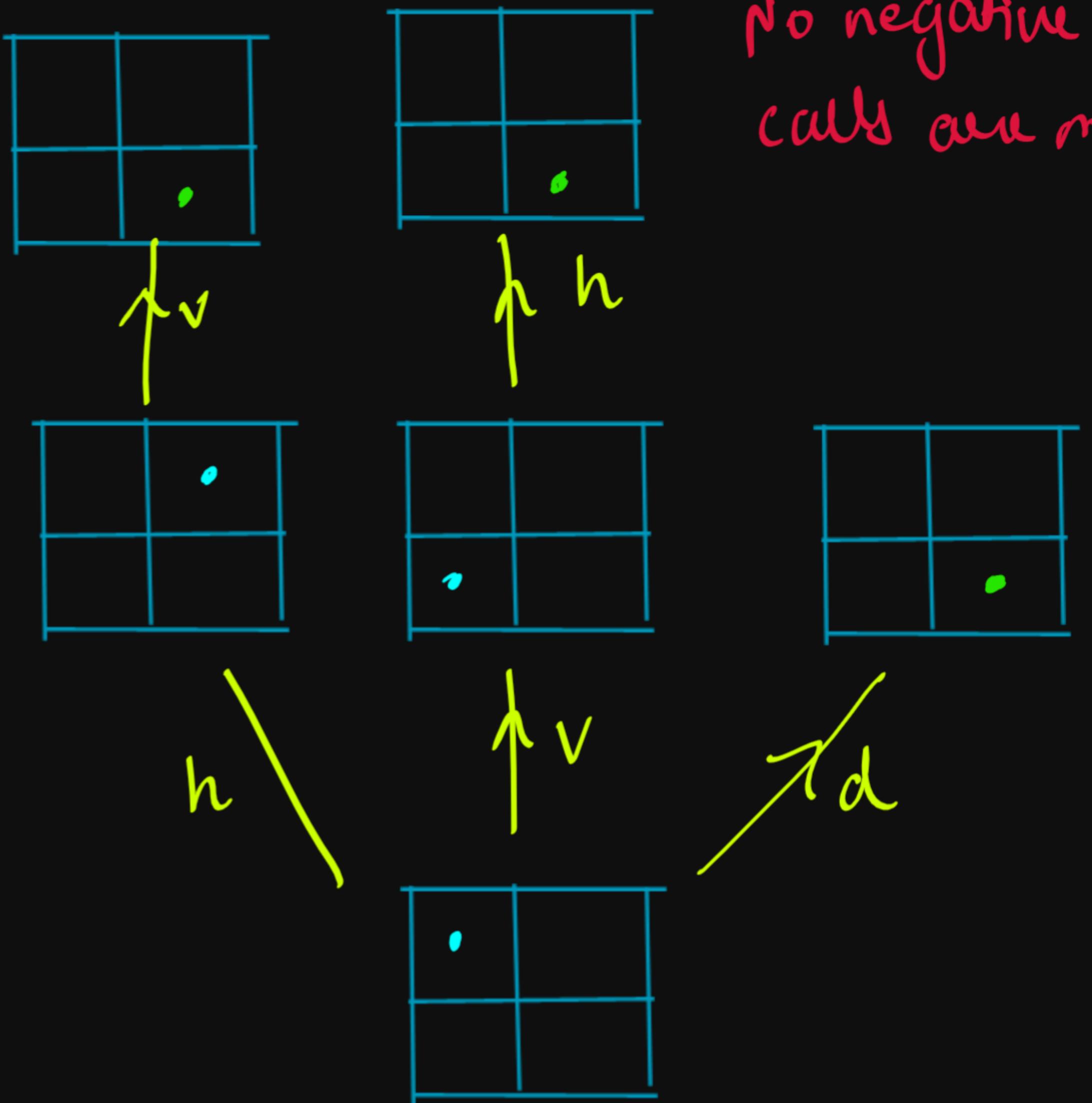
// sr - source row
// sc - source column
// dr - destination row
// dc - destination column
public static void printMazePaths(int sr, int sc, int dr, int dc, String psf) {
    if(sr == dr && sc == dc) {
        System.out.println(psf);
        return;
    }

    int jump = 1;
    while(sc + jump <= dc) {
        printMazePaths(sr, sc+jump, dr, dc, psf + "h" + jump);
        jump++;
    }

    jump = 1;
    while(sr + jump <= dr) {
        printMazePaths(sr+jump, sc, dr, dc, psf + "v" + jump);
        jump++;
    }

    jump=1;
    while(sr + jump <= dr && sc + jump <= dc) {
        printMazePaths(sr+jump, sc+jump, dr, dc, psf + "d" + jump);
        jump++;
    }
}

```



No negative bc
calls are made,

Print Encodings

Given a string of digits (never start with 0)

1 → a

2 → b

:

26 → z

Sample Test Case:

123

123
aw

123
lc

123
abc

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
$\frac{1}{a}$	$\frac{2}{b}$	$\frac{3}{c}$	$\frac{4}{d}$	$\frac{5}{e}$	$\frac{6}{f}$	$\frac{7}{g}$	$\frac{8}{h}$	$\frac{9}{i}$	$\frac{10}{j}$	$\frac{11}{k}$	$\frac{12}{l}$	$\frac{13}{m}$	$\frac{14}{n}$	$\frac{15}{o}$	$\frac{16}{p}$	$\frac{17}{q}$	$\frac{18}{r}$	$\frac{19}{s}$	$\frac{20}{t}$	$\frac{21}{u}$	$\frac{22}{v}$	$\frac{23}{w}$	$\frac{24}{x}$	$\frac{25}{y}$	$\frac{26}{z}$

Ex: 2114713

$\frac{2-b}{2-b} \sqrt{21-u}$

$\frac{b}{114713} \quad \frac{u}{14713}$

$\frac{1-a}{1-a} \sqrt{11-b} \quad \frac{1-a}{14-n}$

$\frac{ba}{14713} \quad \frac{bk}{4713} \quad \frac{aa}{4713} \quad \frac{an}{713}$

$\frac{1-u}{1-u} \sqrt{14-n} \quad \frac{1-u}{47x} \quad \frac{1-u}{47x} \quad \frac{713}{713}$

$\frac{baa}{4713} \quad \frac{ban}{713} \quad \frac{bed}{713} \quad \frac{aad}{713} \quad \frac{ang}{13}$

{ the main concept
of is covered in next
example
and so
on

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z

Ex: 610208

$6-\alpha$ ↘ $\downarrow 61-\alpha$

$\frac{f}{10208}$

10208

$\frac{1-\alpha}{10-\bar{j}}$

$\frac{fa}{fj}$

$\frac{0208}{0208}$ $\frac{208}{2-b}$ $\downarrow 20-t$

$\frac{fjb}{08}$ $\frac{fjt}{8}$

$\frac{0\alpha}{0\alpha}$ $\frac{18x}{18x}$ $\downarrow 8-h$

$\frac{(fjh)}{8}$

Only one encoding possible
for this case.

```

public static void printEncodings(int idx, String input, String output) {
    if(idx == input.length()) {
        //positive base case
        System.out.println(output);
        return;
    }

    int ch1 = input.charAt(idx) - '0';
    if(ch1 >= 1 && ch1 <= 9) {
        printEncodings(idx + 1, input, output + (char)(a' + ch1 - 1));
    }

    if(idx + 1 < input.length()) {
        int ch2 = (input.charAt(idx) - '0') * 10 + (input.charAt(idx+1) - '0');

        if(ch2 >= 10 && ch2 <= 26) {
            printEncodings(idx + 2, input, output + (char)(a' + ch2 - 1));
        }
    }
}

```

