Recursion on the Way Up (Print Variation)

① Print Subsequence

$\{a,b,c\} \xrightarrow{\text{subsets}} \phi, \{a\}, \{b\}, \{c\}, \{a,b\}\ \{a,c\}, \{a,b,c\}$

"abc" → "", "a", "b", "c", "ab", "ac", "abc"

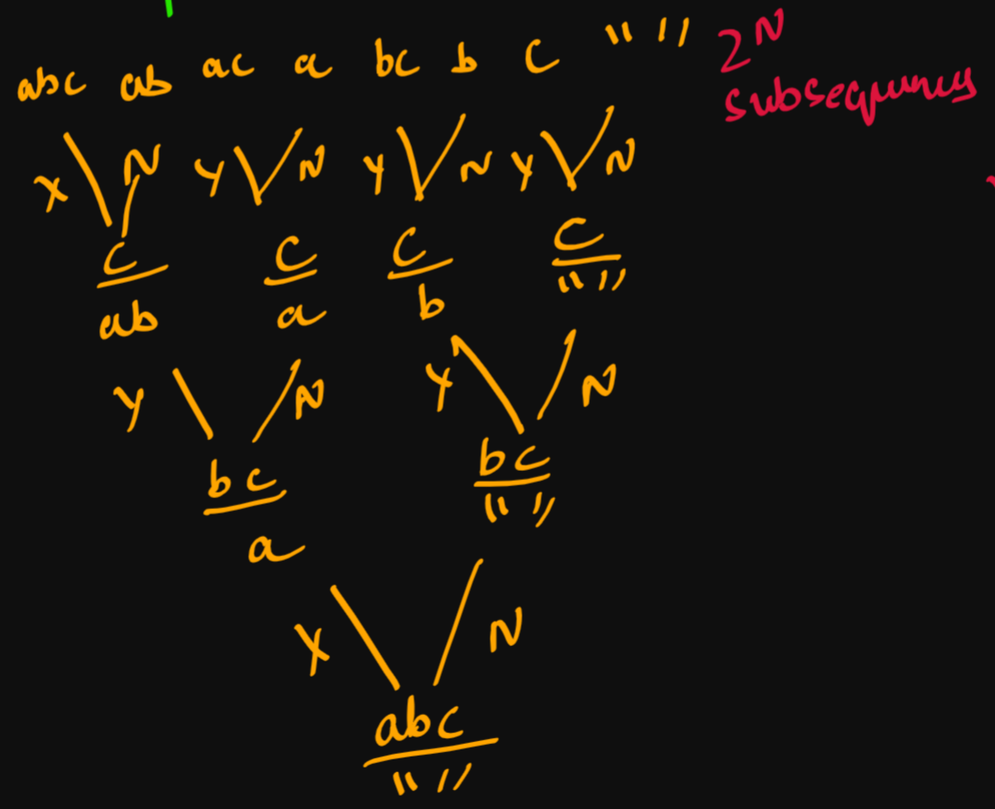
→ In print category, biggest ans / highest width will lie in the base case.

Num = inp
Res = Out

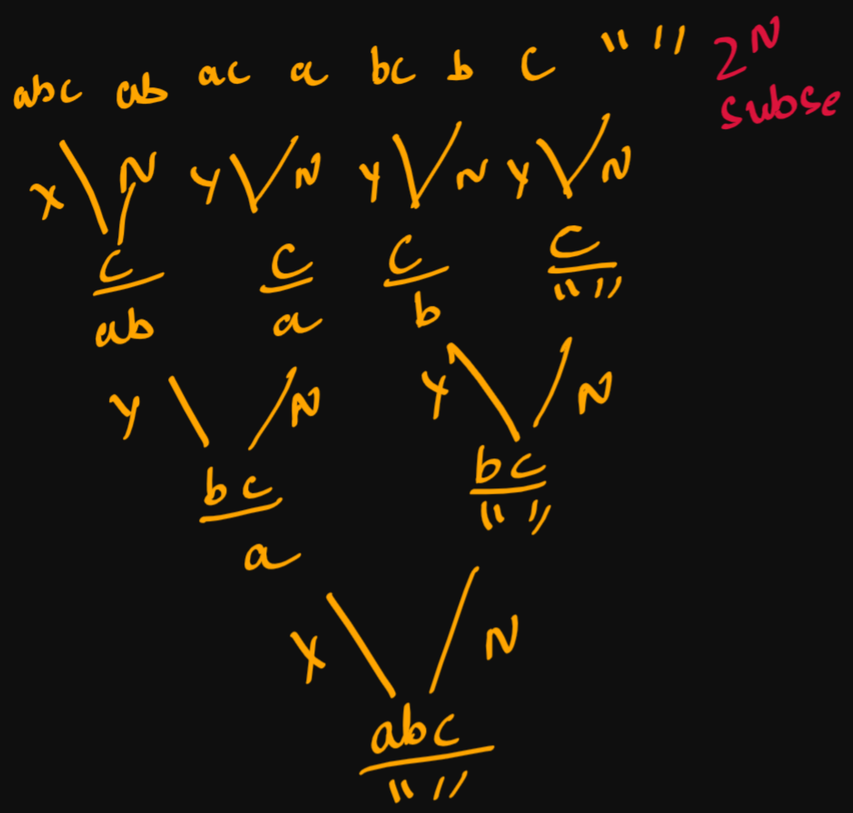
"abc"

printSS(input, output, idx)

Levels → chars
Options → Yes / No



""	3
c	2
b	1
a	0



→ This is the expected combination also

"" 3

c 2

b 1

a 0

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

```java
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 gets qus

Levels: one digit at a time

Options: chars corresponding to each digit

Expectation: printKpc(0, "179", "") → all Kpc's of 179
(0 = idx, "179" = input, "" = output)

Base case:

if(idx == str.length())
  Sysо(ans);
  return;

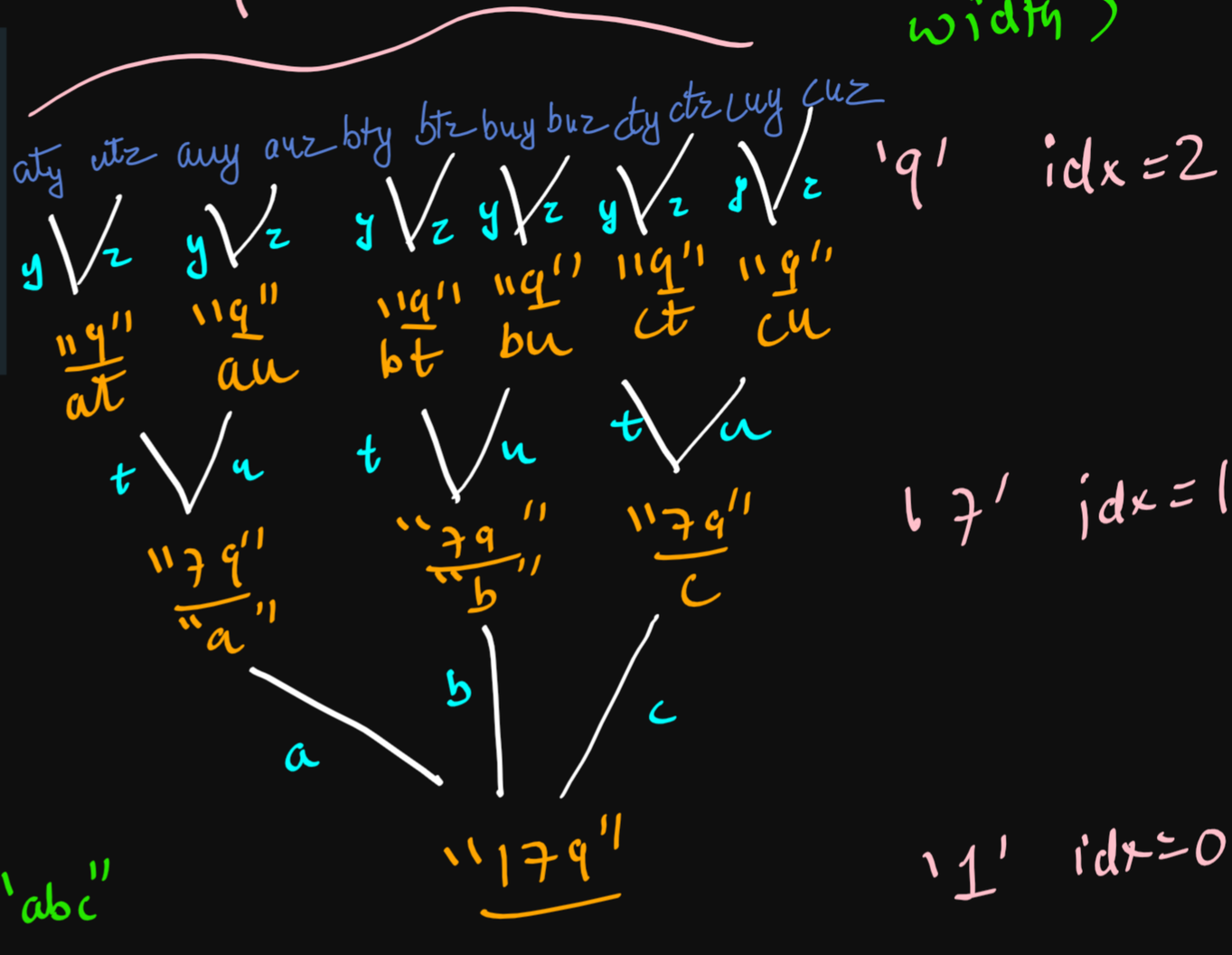
Faith: printKpc(1, "179", "a")
printKpc(1, "179", "b")
printKpc(1, "179", "c")

"79" Kpc's will be appended to this

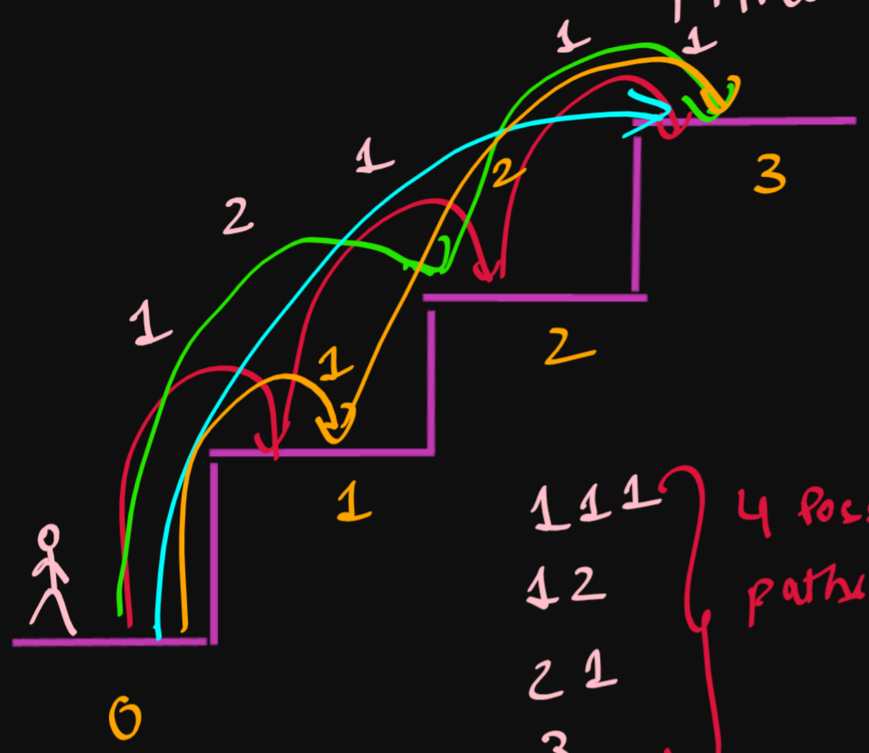
Input = "179"

```
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



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.

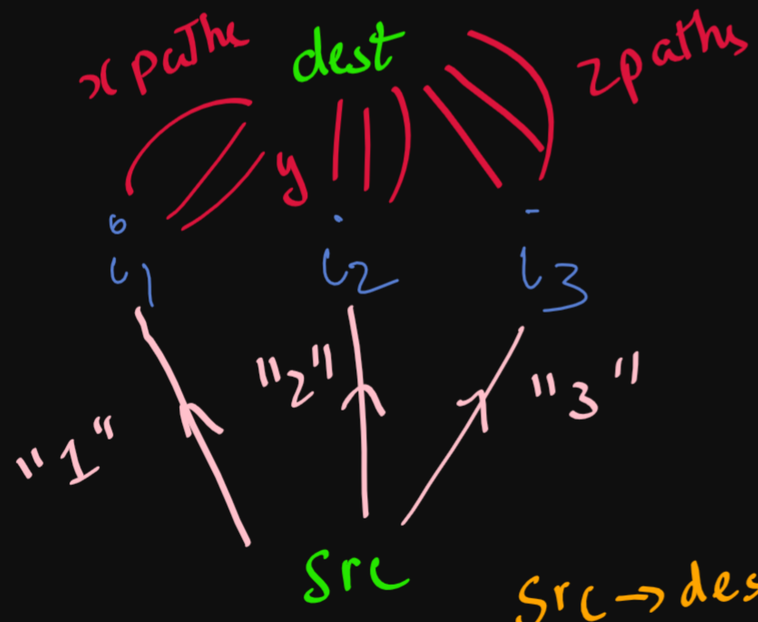
Use sample input and output to take idea about output.

Sample Input

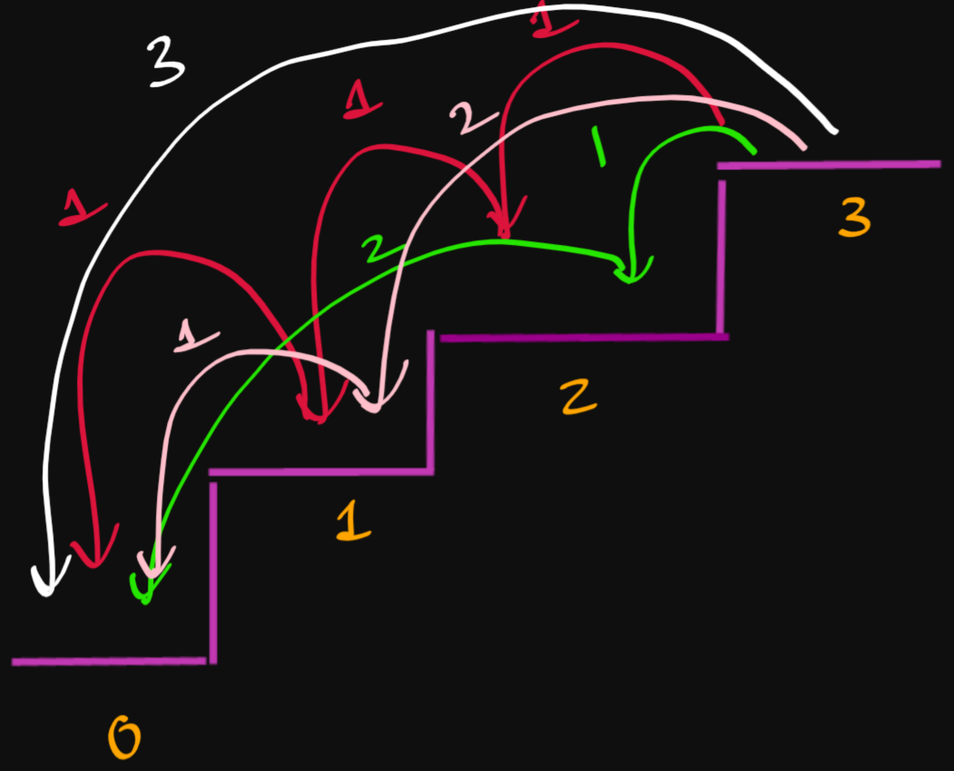
```
3
```

Sample Output

```
111
12
21
3
```



src → dest x+y+z paths




```java
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");
}
```

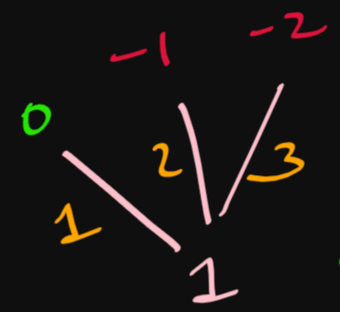
```java
public static void printStairPaths(int n, String path) {

    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

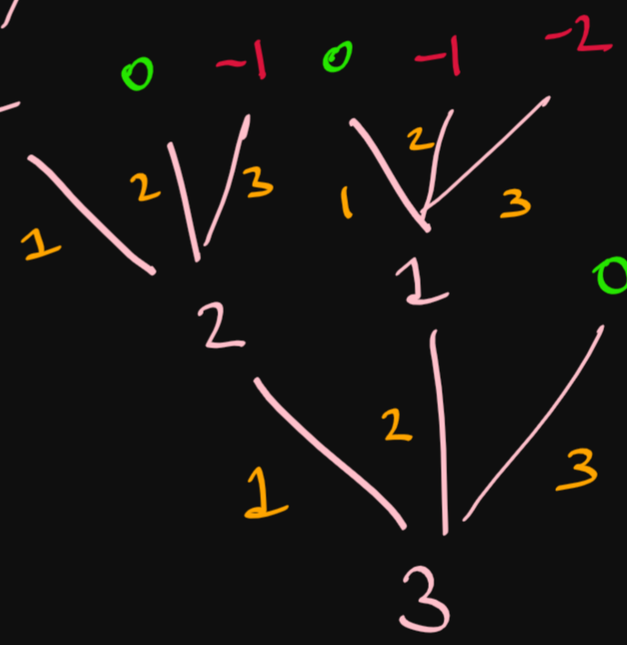
111 12

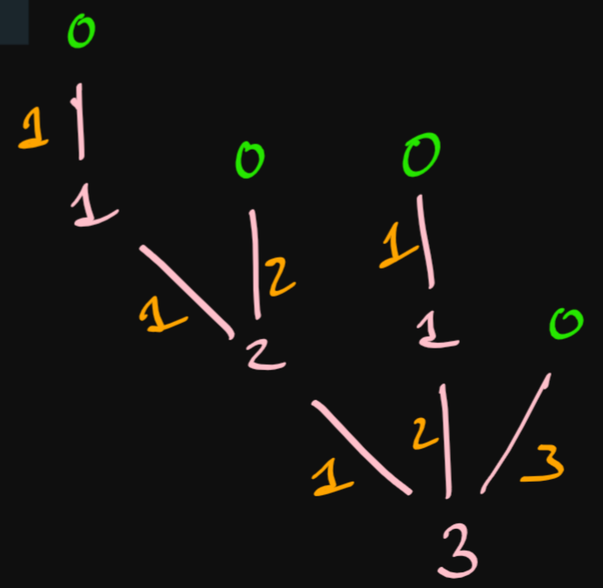
21 3

- Positive Base Case
- Negative Base Case



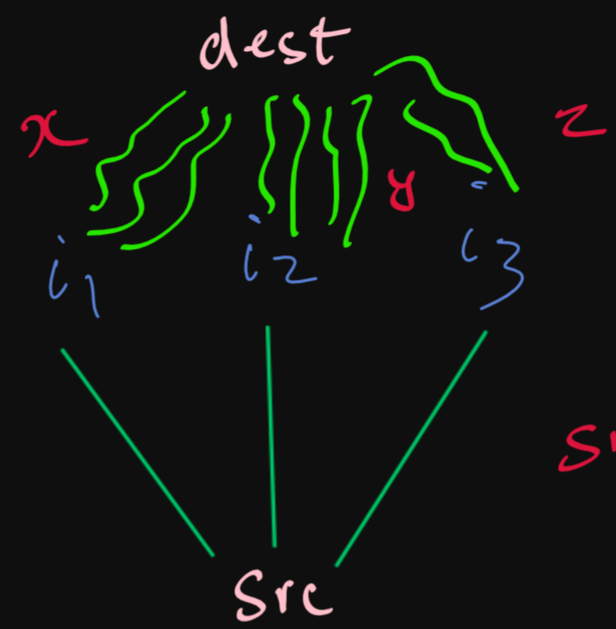
Width is max at the base case.



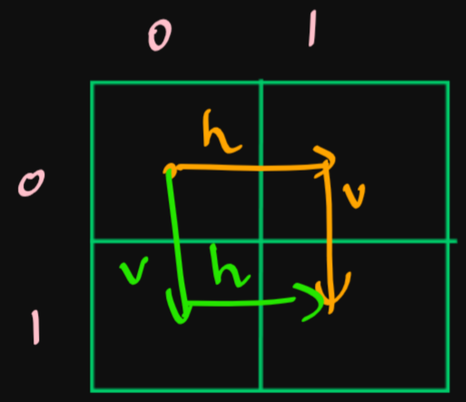


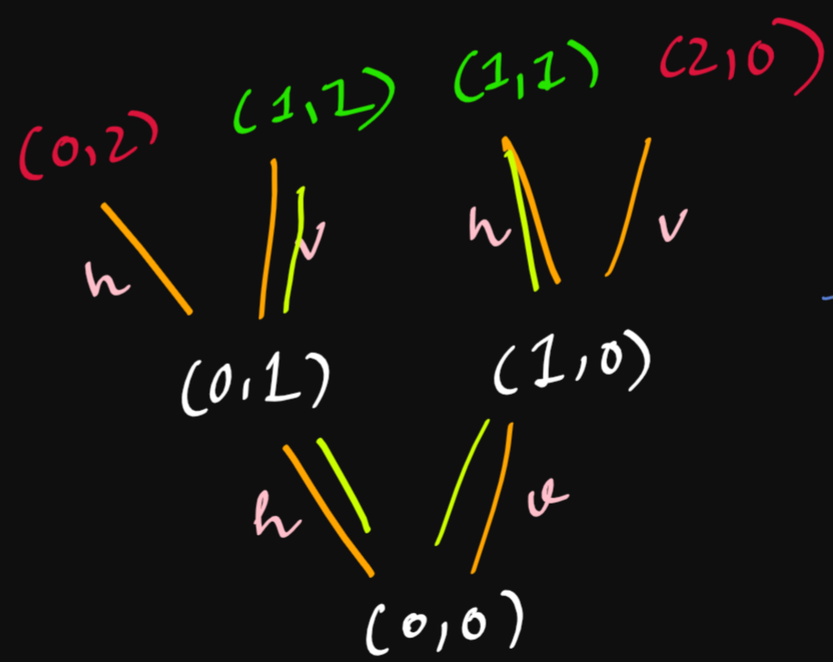
There is no call for -ve base case

Print Maze Paths



src → dest
x+y+z paths

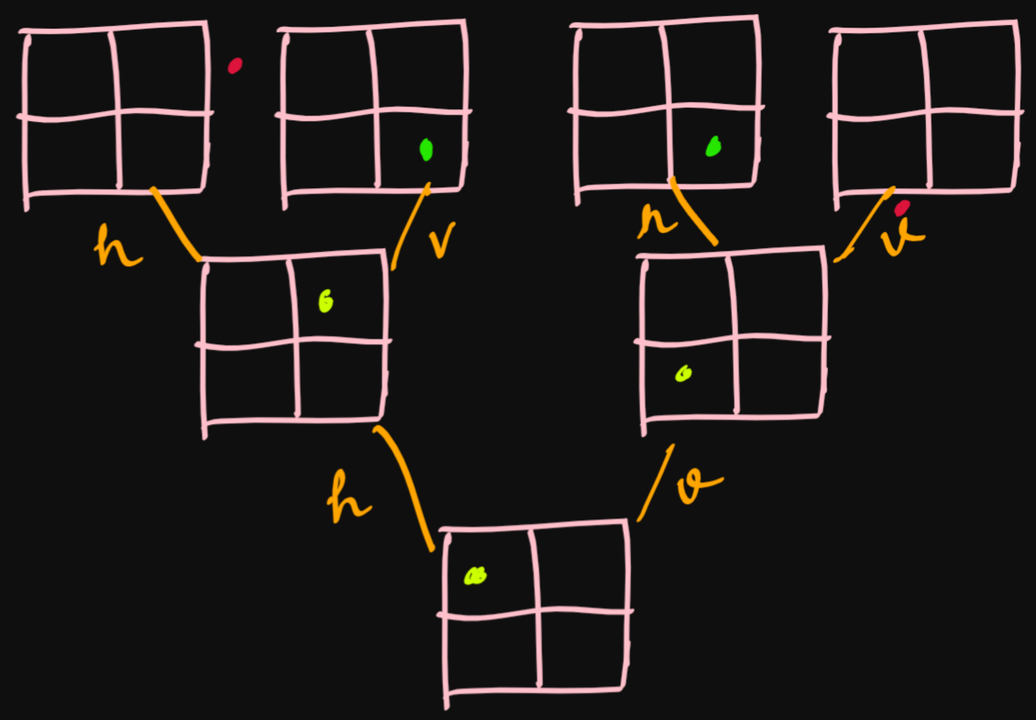




Without taking care of
- ve base case calls.

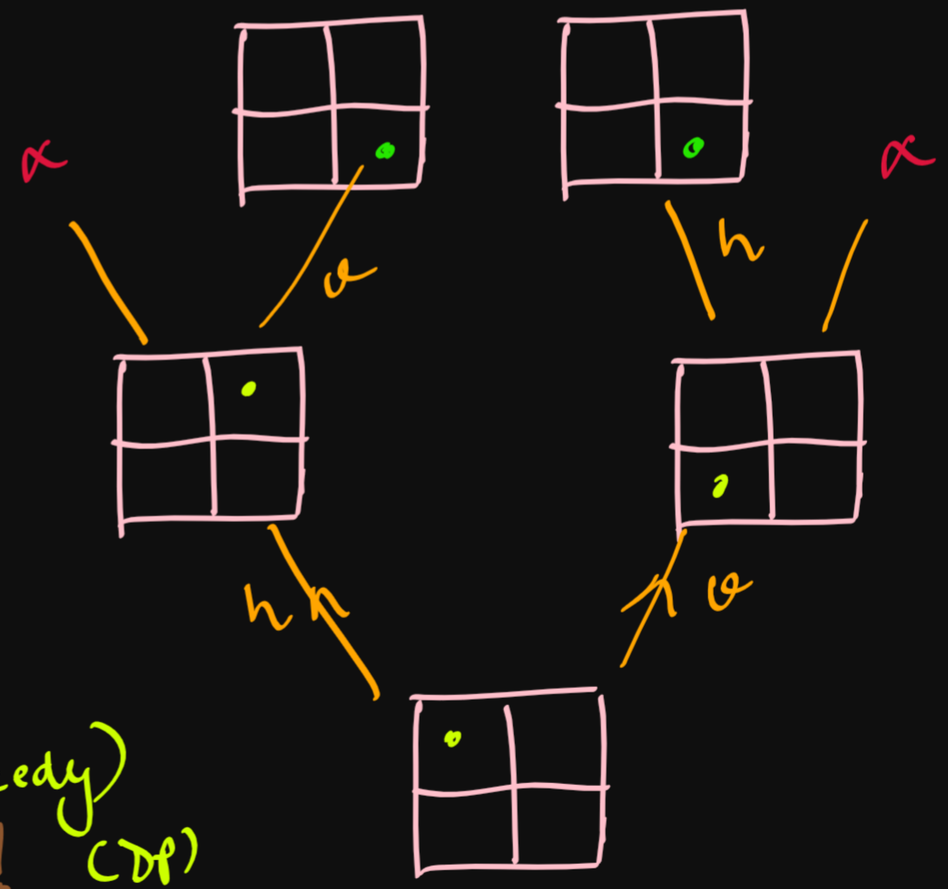
Without smart call

```java
// 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");
}
```



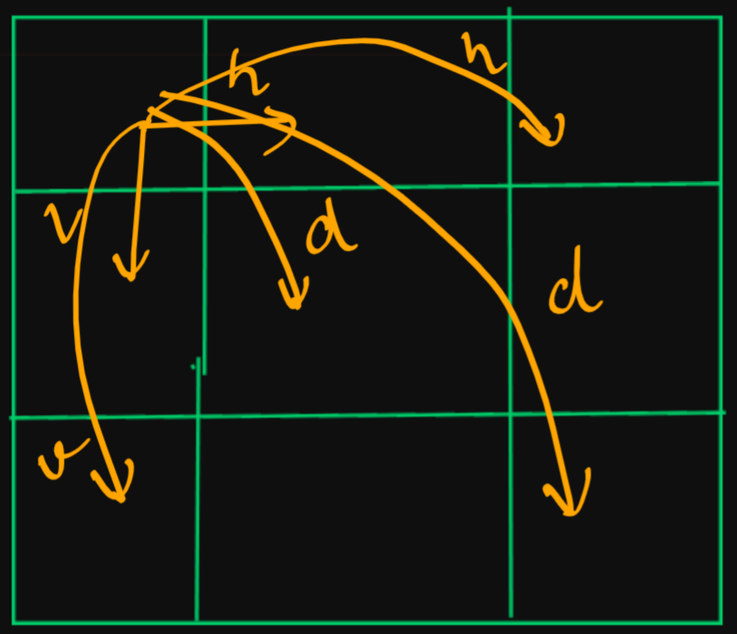
Smart Calls

```java
// 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");
    }

}
```

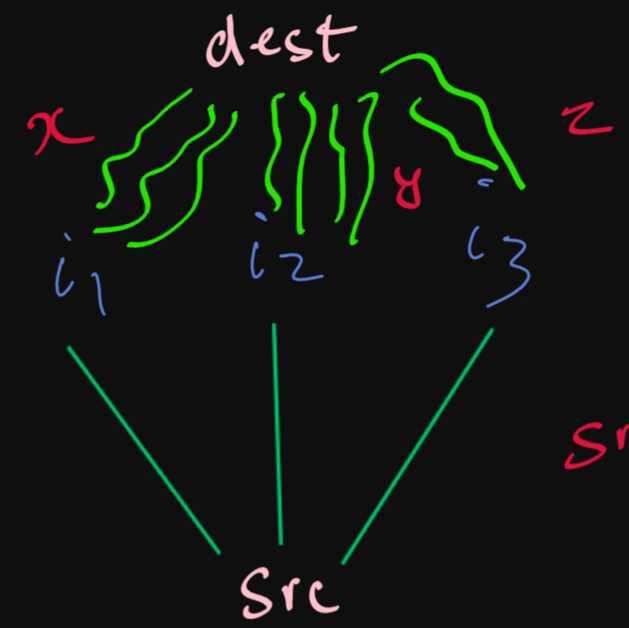


* No of paths = rows - 1 + cols - 1 (Greedy)
* Length of path = $\frac{(rows - 1 + cols - 1)!}{(rows-1)!\,(cols-1)!}$ (DP)

Print Maze Path With Jumps



v (↓)
h (→)
d (↘)



src → dest
x+y+z paths

+ve Base Case:

Destination Reached

-ve Base Case

Out of the Matrix

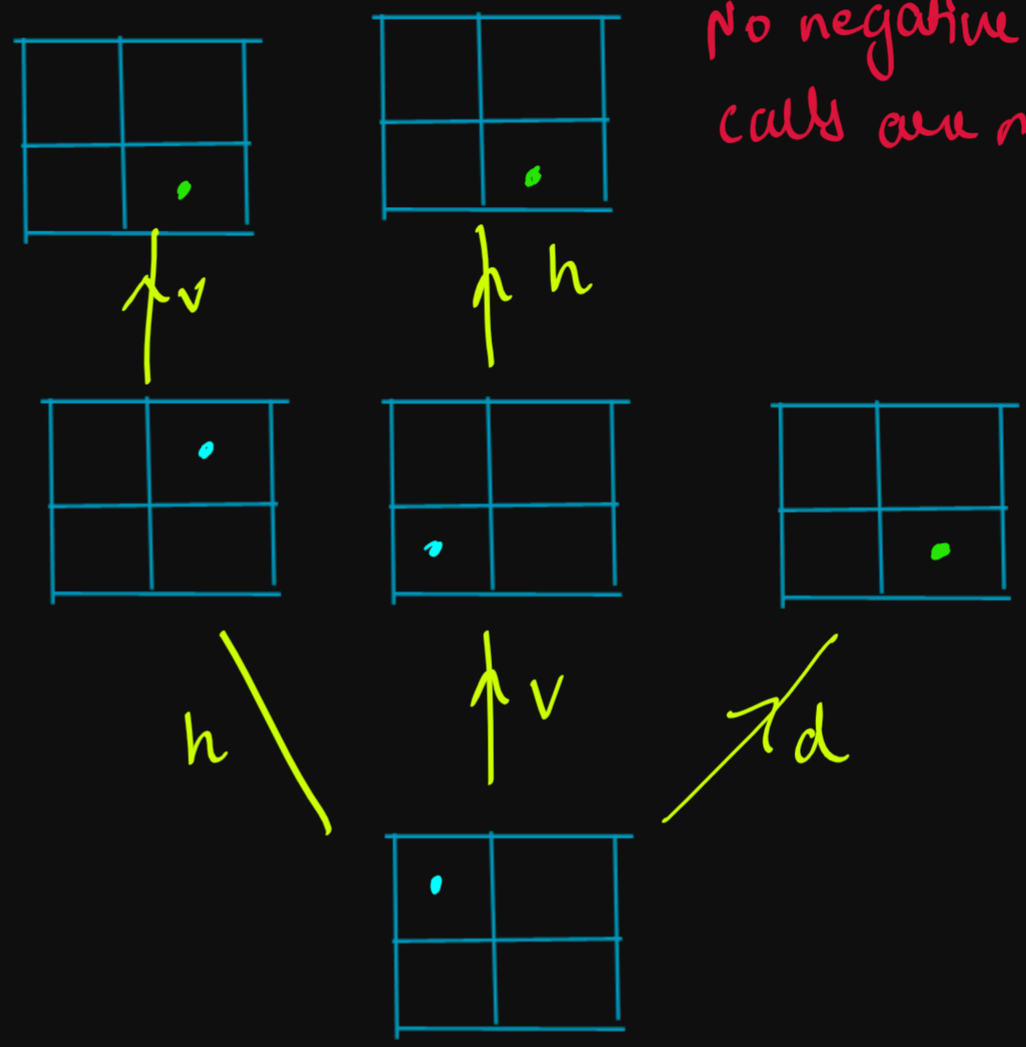
* We have to go to all the intermediates & apply recursion.

Max horizontal jumps: cols - 1

Max vertical jump: row - 1

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

```java
// 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++;
    }
}
```

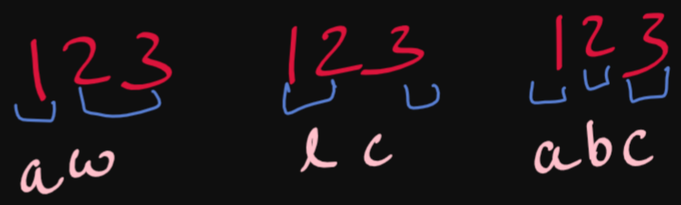


Print Encodings

Given a string of digits (never start with 0)

1 → a
2 → b
⋮
26 → z

Sample Test Case: 123



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: 2 11 4 7 13

2-b, 21-u

b — 11 4 7 13

u — 14 7 13

1-a, 11-k, 1-a, 14-n

ba — 14 7 13

bk — 4 7 13

aa — 4 7 13

an — 7 13

1-a, 14-n, 4-a, 4-d, 7-g, 7-x

baa — 4 7 13

ban — 7 13

bkd — 7 13

aad — 7 13

ang — 13

and so on

* the main concept of 0 is covered in next example

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-f ↙ ↘ 61-aq

f

10208

1-a ↙ ↘ 10-j

fa fj

0208 208

oa ↙ 102k 2-b ↙ ↘ 20-t

fjb fjt

08 8

oa ↙ 18x ↓ 8-h

fjth → only one encoding possible for this case.

```java
public static void printEncodings(int idx,String input,String output) {
    if(idx == input.length()) {
        //positve 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));
        }
    }
}
```