RECURSION - WITH - ARRAYLISTS

Get Subsequence

abc →
- ___
- __c
- _b_
- _bc
- a__
- a_c
- ab_
- abc

This is subsequence of abc

$\therefore n \rightarrow 2^n$

abc →
- a
- ab
- abc
- b
- bc
- c

This is sub-string

$\therefore n \rightarrow \frac{n(n+1)}{2}$

Hume input me getss(abc) → isko ek arraylist bhar kr deni jisme saare subsequence padey ho!

Expectation

getss(abc) =
- ___
- __c
- _b_
- _bc
- a__
- a_c
- ab_
- abc

Faith

getss(bc) =
- __
- _c
- b_
- bc

Expectation meet Faith

getss(abc) = getss(bc) + a getss(bc)

{ ___, __c, _b_, _bc } { a__, a_c, ab_, abc }

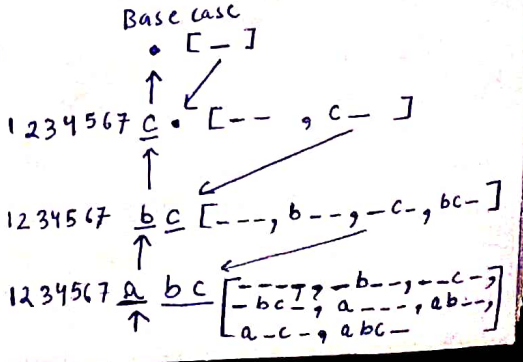
blank string kitne subsequence? → 1 → konsi → blank itself → why? → $2^0 = 1$

```
ps v m(S[] a){
Scanner s = new Scanner(System.in);
String str = s.next();
ArrayList<String> res = gss(str);
System.out.println(res);
}

p s ArrayList<String> gss(String str){
 if(str.length() == 0) { //Base Case
   ArrayList<String> bres = new ArrayList<>();
   bres.add("");
   return bres;
 }
 char ch = str.charAt(0); (1)
 String ros = str.substring(1); (2)
 ArrayList<String> rres = gss(ros); (3)
 ArrayList<String> mres = new ArrayList<>(); (4)
 for(String val : rres){ (5)
    mres.add("" + val);
 }
 for(String val: rres){ (6)
    mres.add(ch + val);
 }
   return mres; (7)
} }
```

bc [--, -c, b-, bc]

↑

abc [---, --c, -b-, -bc, a--, a-c, abc, abc]



While Going Up

• [_] (Base Case)

① ② ③ c •

① ② ③ b c

① ② ③ a b c

While Going Down

• [_]

① ② ③ ④ ⑤ ⑥ ⑦ c • [--, c-]

① ② ③ ④ ⑤ ⑥ ⑦ b c [---, -c-, b--, bc-]

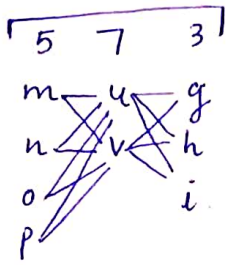
① ② ③ ④ ⑤ ⑥ ⑦ a b c [----, --c-, -b--, -bc-, a---, a-c-, ab--, abc-]

∴ FINAL OUTPUT = ["", "c", "b", "bc", "a", "ac", "ab", "abc"]

- TIME COMPLEXITY : O(n)
- SPACE COMPLEXITY : O(1)

Get Kpc

1 → abc
2 → def
3 → ghi
4 → jkl
5 → mnop
6 → qrst
7 → uv
8 → wx
9 → yz
0 → . ;



[mug, muh, mui,
nug, nuh, nui,
oug, ouh, oui,
pug, puh, pui,
mvg, mvh, mvi,
nvg, nvh, nvi,
ovg, ovh, ovi,
pvg, pvh, pvi]

73 [ug uh ui
vg vh vi] 6 words

5 7 3 → m, n, o, p

[mug, muh, mui, mvg, mvh,
mvi, nug, nuh, nui, nvg, nvh,
nvi, oug, ouh, oui, ovg, ovh,
ovi, pug, puh, pui, pvg, pvh, pvi] 24 words

Expectation

5 7 3 ⇒ [ ]
↓
24 words milengy
↓
why ?

∴ 5 → mnop ∴ 4
7 → uv ∴ 2
3 → ghi ∴ 3
] 4×2×3 = 24

Faith

73 = [ ]
↓
6 words milenay
↓
why

7 → ∴ 2
3 → ∴ 3
] 2×3 = 6

```
public static void main (String [] args){
Scanner s = new Scanner (System.in);
String str = s.next();
ArrayList<String> words = getKPC (str);
System.out.println (words);
}
```

to declare globally

```
static String [] codes = {". ;", "abc", "def", "ghi",
"jkl", "mno", "pqrs", "tu", "vwx",
"yz"};
```

```
Public static ArrayList<String> getKPC(String str) {
  if (str.length() == 0){
  ArrayList<String> bres = new ArrayList<>();
  bres.add("");
  return bres;
  }
    // 678
char ch = str.charAt(0); // 6 — (1)
String ros = str.substring(1); // 78 — (2)  → (3)
ArrayList<String> rres = getKPC(ros); // 6 words of 78
ArrayList<String> mres = new ArrayList<>(); // 24 words of 678
                                  → (4)
String codeforch = codes[ch - '0'];
for (int i = 0; i < codeforch.length(); i++){
  char chcode = codeforch.charAt(i);
  for(String val : rres) {
    mres.add(chcode + val);
  }
}
return mres;
}
}
```

(5)

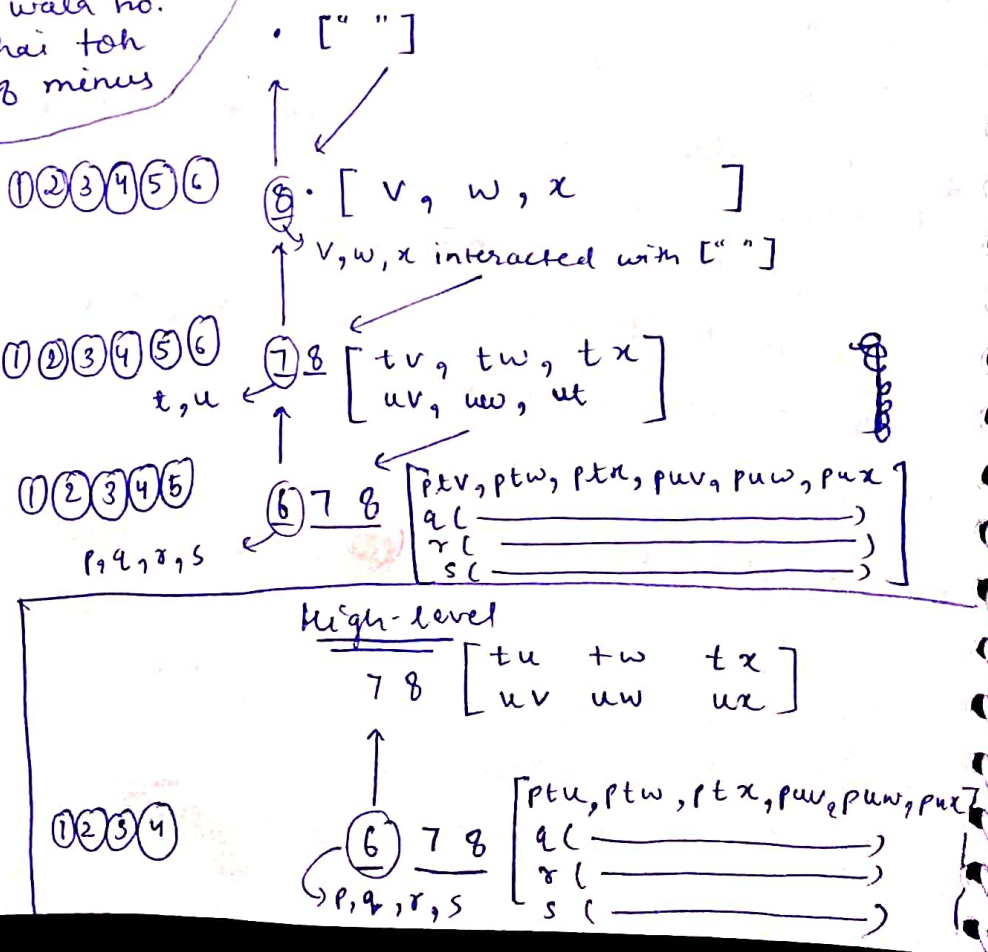
Base case : Agar koi key nahi press hui toh ek word pada hoga vo word blank hoga! → $2^0 = 1$

Agar hum character waley no. ko asli wala no. bana chaty hai toh usme se 48 minus krna hoga

6 7 8
p t v
q u w
x
r
s

'6' ≠ 6 → no. ka 6
↓
character
6

'a' → 97 '3' = 51
'0' → 48 '4' = 52
'1' → 49 '5' = 53
'2' → 50 '6' = 54



- Agar hum character wale number ko real wale number me convert krna chaty hai toh character wale number se 48 minus krna hoga!

$\therefore$ '6' $\neq$ 6

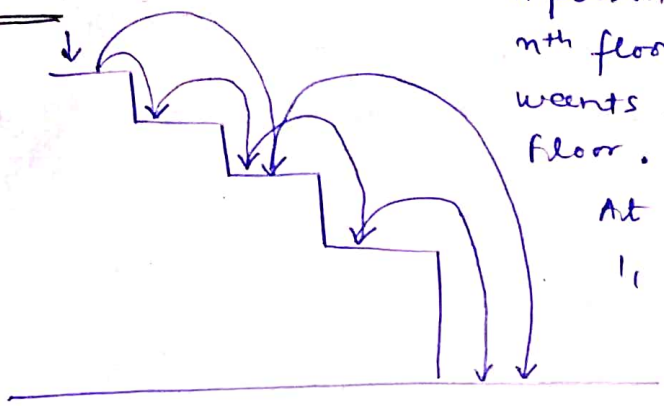
$\therefore$ But,

'6' - '0' = 6

54 - 48

- Time Complexity : O(n)
- Space Complexity : O(1)

Het Stair Path



A person is standing on the n^{th} floor of staircase and wants to come to the 0^{th} floor.

At a time he can go 1, 2, or 3 step down

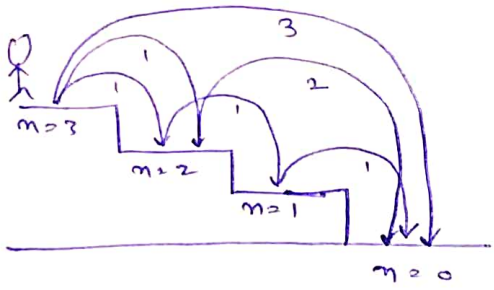
↓

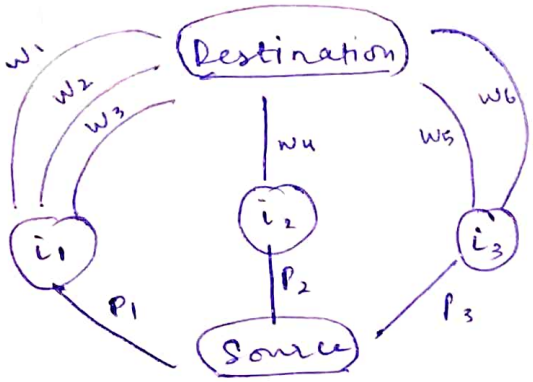
Hume saare path print krne hai

For example

$n = 3 \longrightarrow$ paths

111
12
21
3

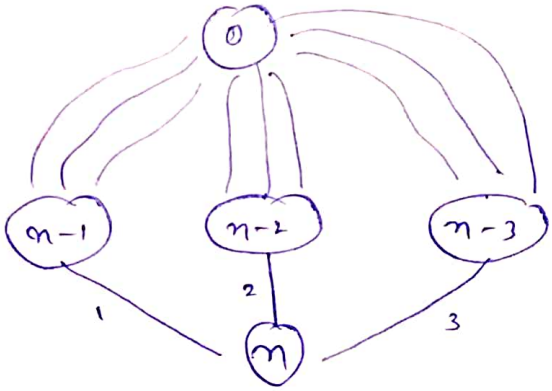




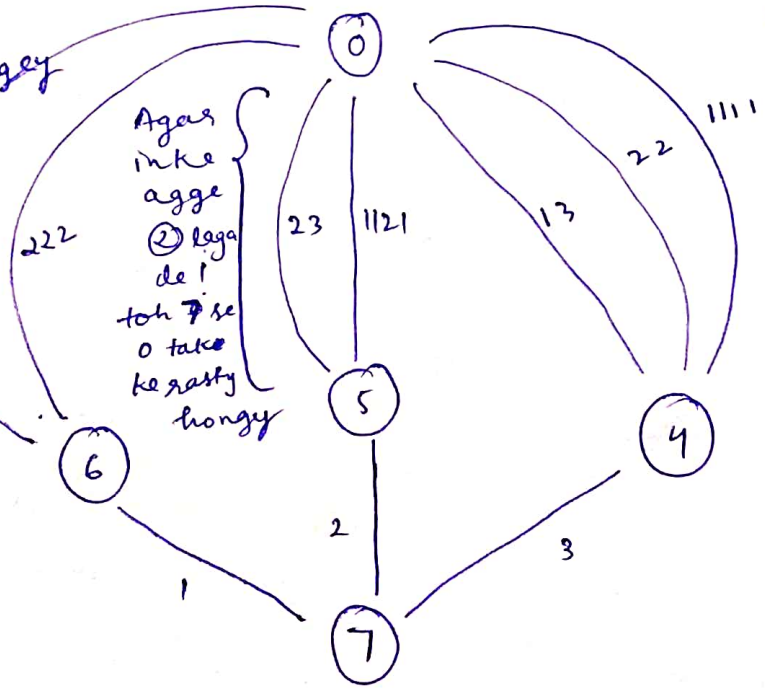
$P_1, P_2, P_3 \rightarrow$ path

$w_1, w_2, w_3, w_4, w_5, w_6 \rightarrow$ ways

$i_1, i_2, i_3 \rightarrow$ intermediate



Agar inke agey ① laga dia jaye 33 toh 7 se 0 tak ke raasty honge!



Agar inke agge ② laga de! toh 7 se 0 take ke raasty honge

Agar in saare sample raasto ke aage ③ laga dia jaye, toh yeh (7) se leke ⓪ tak ke raasty hojayenge!

expectation :

(n) se (0) tak ke rasty chaiye

[n, 0] paths

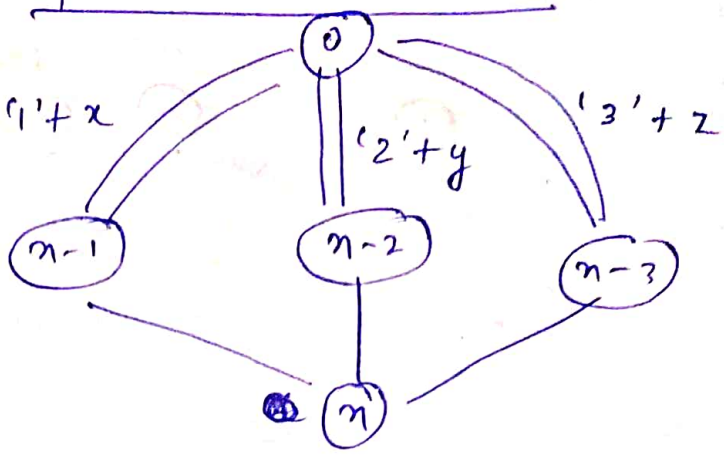
faith

(n-1) se 0,]
(n-2) se 0,]
(n-3) se 0;]

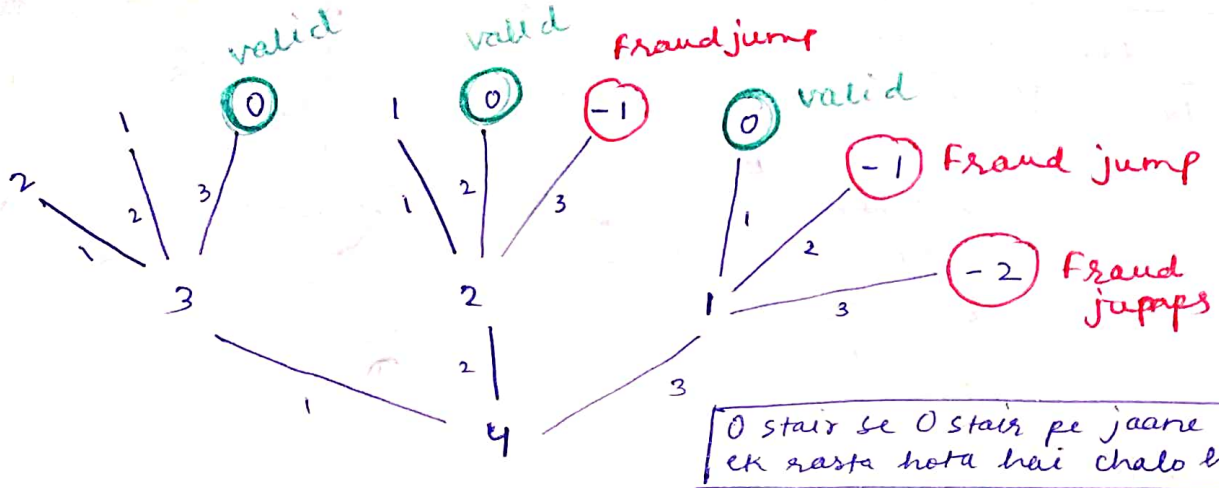
inn sabke rasty chaiye !

[n-1, 0] paths
[n-2, 0] paths
[n-3, 0] paths

Expectation meets faith




```
psvm(S[]a){
Scanner s = new Scanner(System.in);
int n = s.nextInt();
ArrayList<String> paths = getStairPaths(n);
System.out.println(paths);
}

ps ArrayList<String> getStairsPaths(int n){
if(n==0) {
ArrayList<String> bres = new ArrayList<>();
bres.add("");
return bres;
} else if (n<0) {
Arraylist<String> bres = new Arraylist<>();
return bres;
}
Arraylist<String> paths1 = getStairPaths(n-1); (1)
_____ paths2 = _____(n-2); (2)
_____ paths3 = _____(n-3); (3)
Arraylist<String> paths = new Arraylist<>(); (4)
for(String path1: paths1) {
paths.add("1" + path1); } (5)
}
For(String path: paths2){
paths.add("2" + path); } (6)
}
For(String path: paths3){
paths.add("3" + path); } (7)
}return paths; (8)
}
}
```

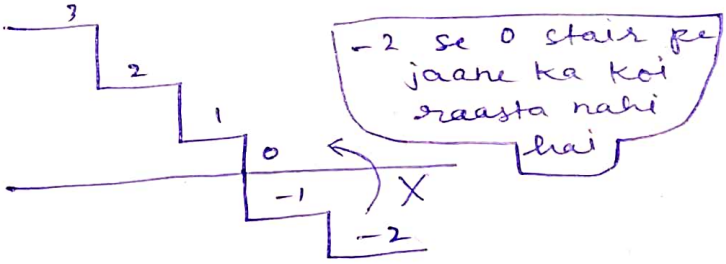


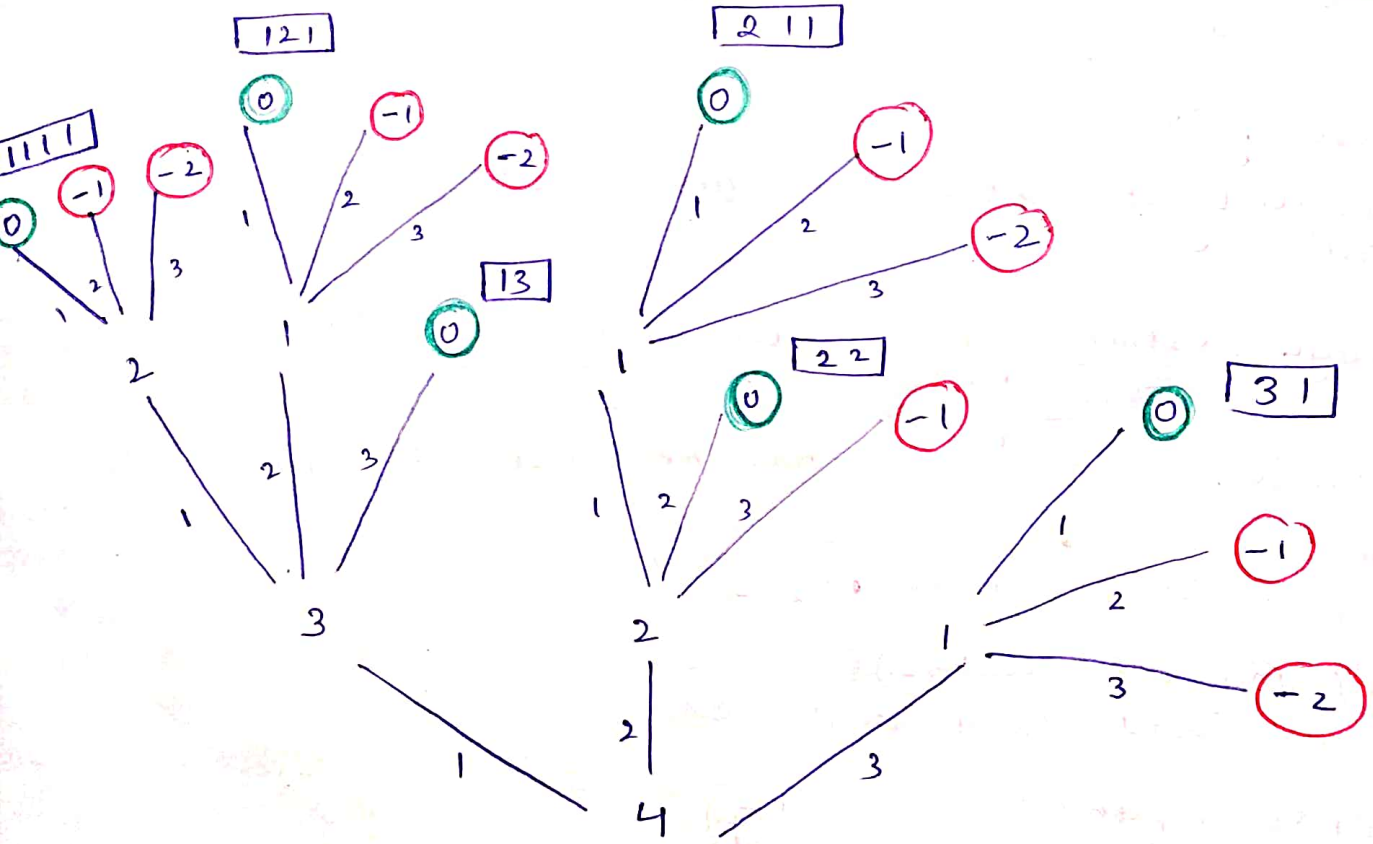
0 stair se 0 stair pe jaane ka ek rasta hota hai chalo ki mat

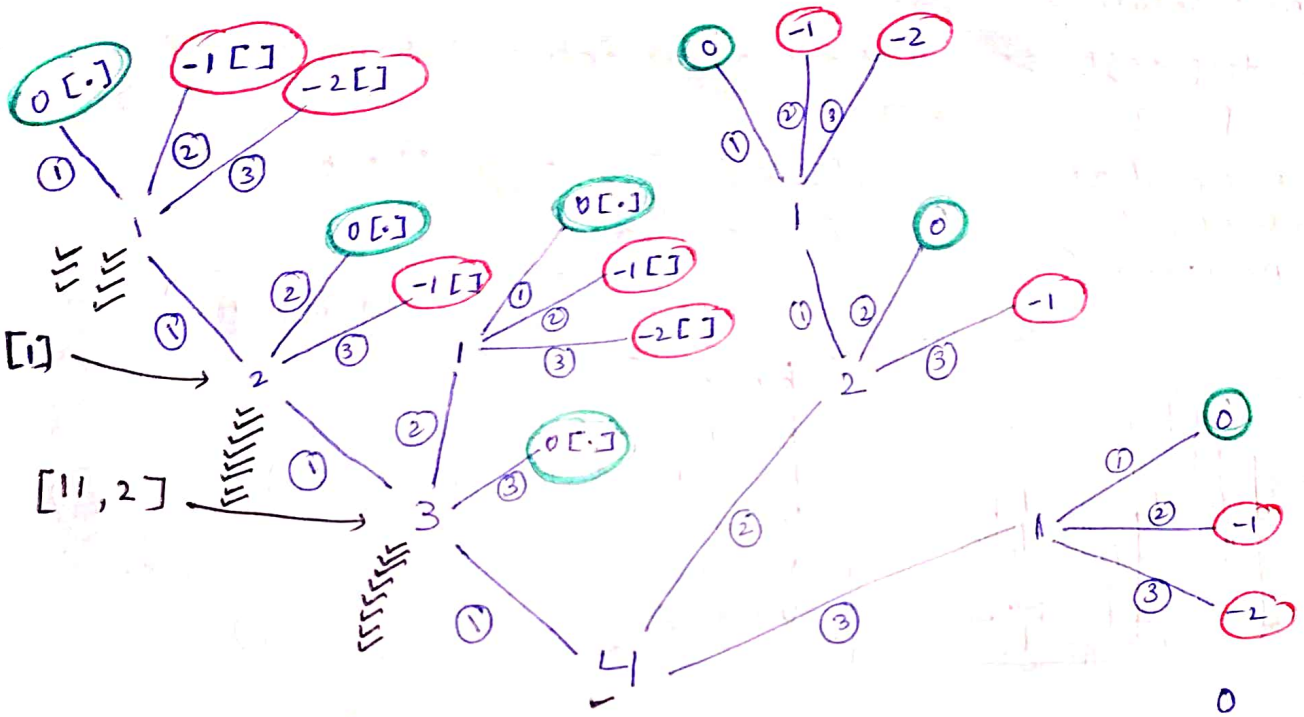
```
if (n == 0) {
Arraylist <string> bres = new Arraylist <>();
 bres.add("");
 return bres;
}
else if (n < 0) {
Arraylist <string> bres = new Arraylist <>();
 return bres;
}
```

0 → Incorrect paths

0 → Correct path







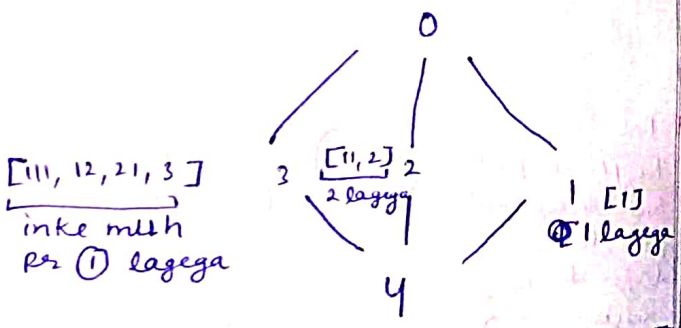
- Time complexity

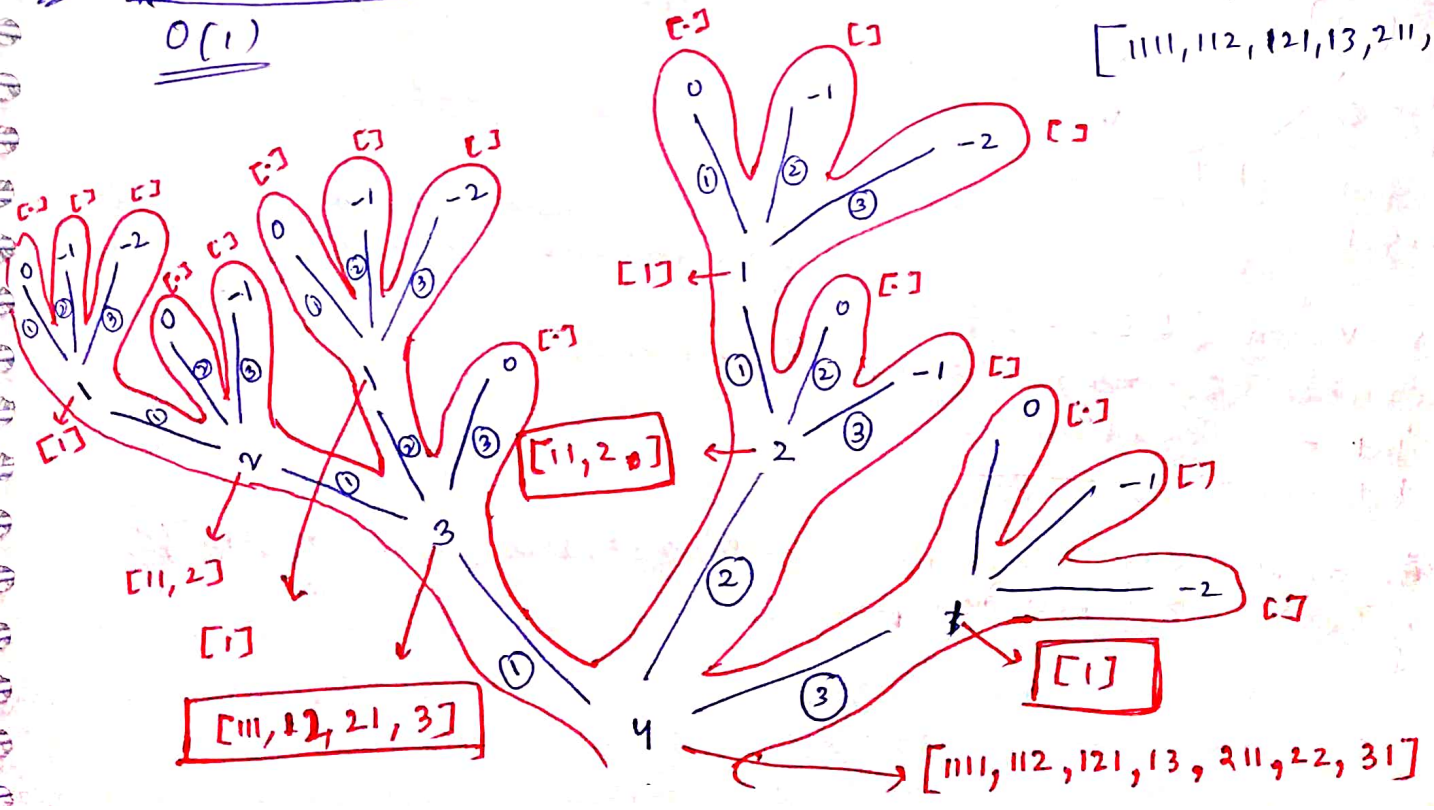
$O(3^n)$

↳ becoz of each state, 3 recursion calls are made

- Space Complexity

$O(1)$

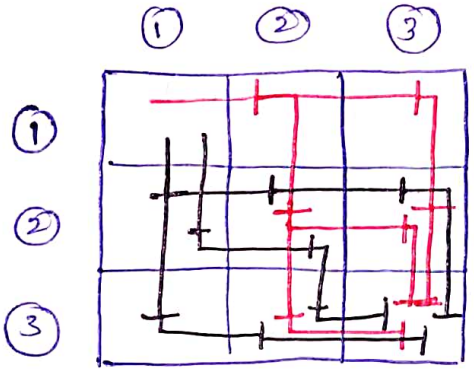




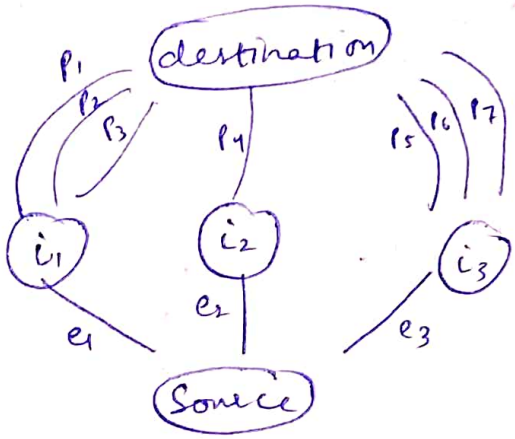
Get Maze Path

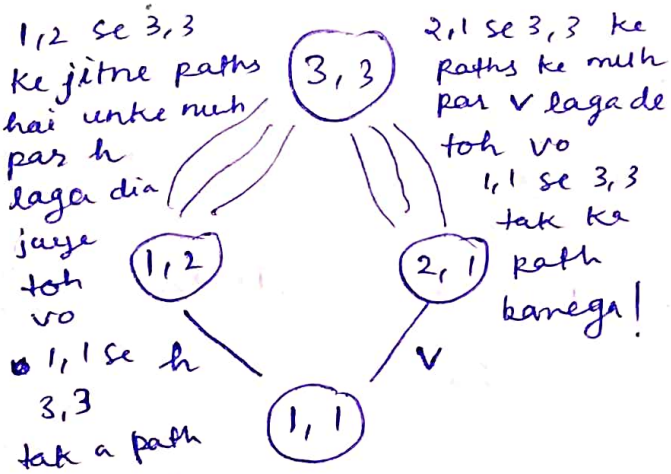
- Hume ek 2D maze dia hai
  - n → rows
  - m → columns
- Hum top left se bottom right corner tak jana hai
  - [1,1] → [n,m]
- Do tarike hai
  - ↳ Vertical
  - ↳ Horizontal

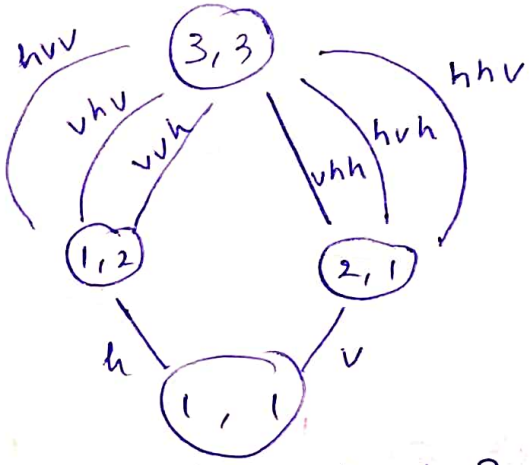
Hume saare paths print krne hai!



{ hhvv, hvhv, hvvh, vvhh, vhvh, vhhv }







{ hhvv, hvhv, hvvh, vvhh, vhvh, vhhv }

```
P s v m (s [] a) {
  Scanner s = new Scanner (System.in);
  int n = s.nextInt();
  int m = ____________;

  ArrayList <String> paths = getMazePaths (1, 1, n, m);
  System.out.println(paths)
}
```

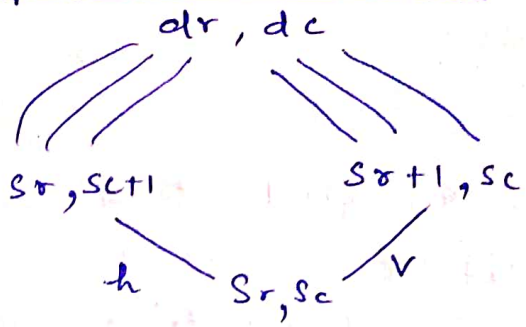
Expectation

Humne (1,1) se (n,m) tak ke saare paths chaiye

Faith

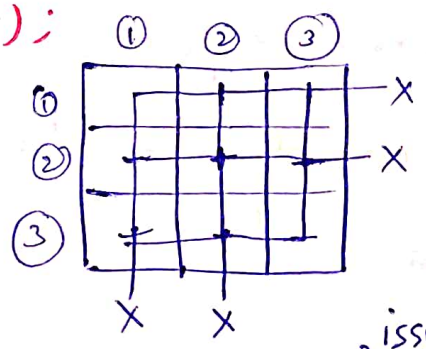
(1,2) → (n,m)
(2,1) → (m,n)
inke paths humne milengy humne Faith hai

Expectation meets Faith



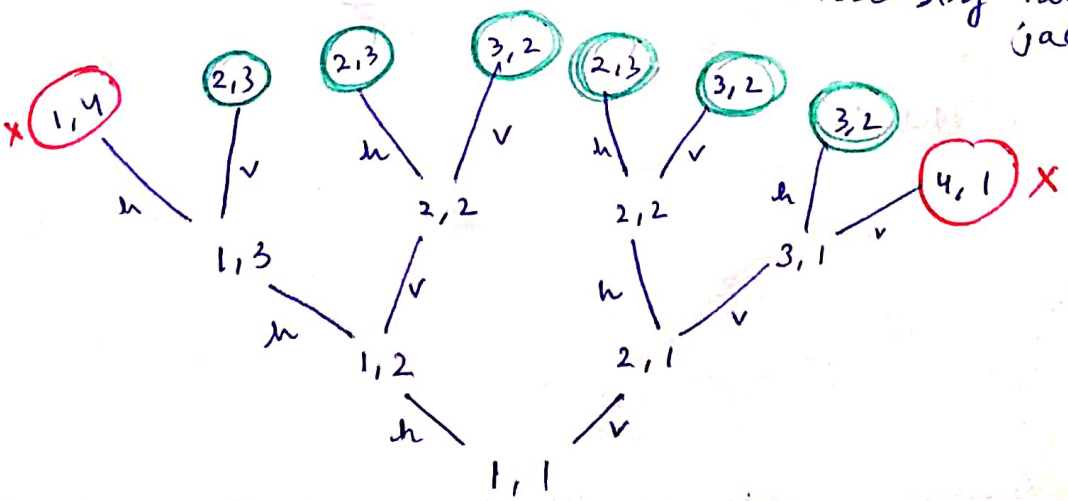

```
p s ArrayList<String> getMazePaths(int sr, int sc, int dr, int dc)
{
ArrayList<string> hpaths = getMazePaths(sr, sc+1, dr, dc);
_____ vpaths = _____(sr+1, sc, dr, dc);

ArrayList <string> paths = new ArrayList<>();
for (String hpath : hpaths){ //horizontal paths pe loop lagaya unke much per (h) lagaya!
    paths.add("h" + hpath);
}
for (String vpath : vpaths){ //v path humne vpaths se nikala aur hpaths pe loop laga kr unke much pas v lagadiya!
    paths.add("v" + vpath);
}
return paths;
}
```

hpath humne hpaths nikala aur



DRY RUN

We conclude,
Jo Right-most wall hai, usse bahar nahi jaa sakty! → issesirf vertical jaaty
Jo Bottom-most wall hai, usse bahar nahi ja sakty! → isse sirf horizontal jaaty!

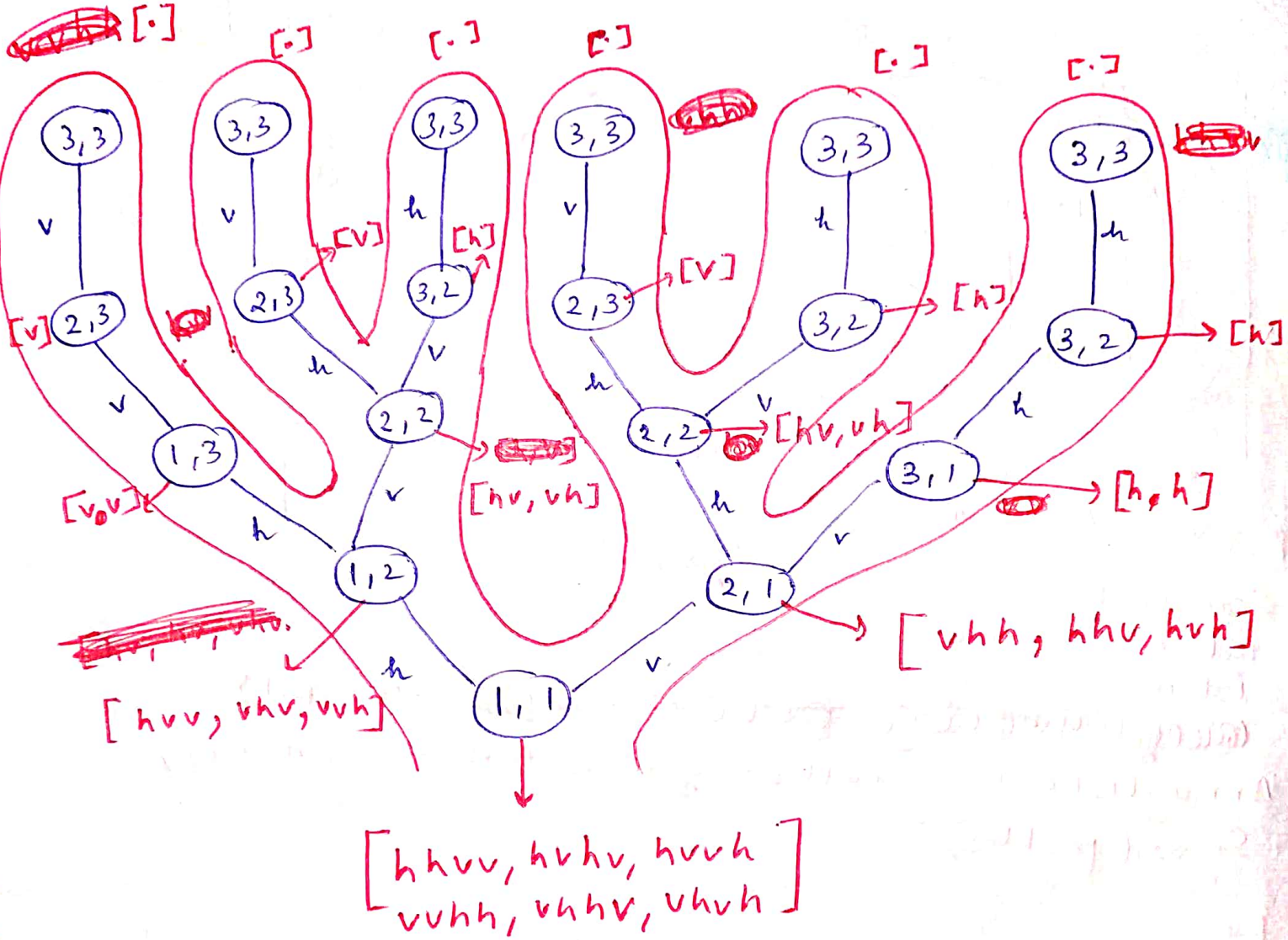


```
// sr = source row
// sc = source column
// dc = destination column
// dr = destination row

public ArrayList<String> getMazePaths(int sr, int sc, int dr, int dc) {
    if (sr == dr && sc == dc) {
        ArrayList<String> bres = new ArrayList<>();
        return bres;
    }

    ArrayList<String> hpaths = new ArrayList<>();
    ArrayList<String> vpaths = new ArrayList<>();

    if (sc < dc) {
        hpaths = getMazePaths(sr, sc + 1, dr, dc);
    }

    if (sr < dr) {
        vpaths = getMazePaths(sr + 1, sc, dr, dc);
    }

    ArrayList<String> paths = new ArrayList<>();

    for (String hpath : hpaths) {
        paths.add("h" + hpath);
    }

    for (String vpath : vpaths) {
        paths.add("v" + vpath);
    }

    return paths;
}
```

Base Case

To every path in hpaths, "h" is added.

To every path in vpaths, "v" is added.



Time Complexity: $O(2^n)$

↳ Because of each state, 2 recursion called.

Space Complexity:

$O(1)$

⑤ Get Maze Path With Jumps

Is baar hum, ek step me

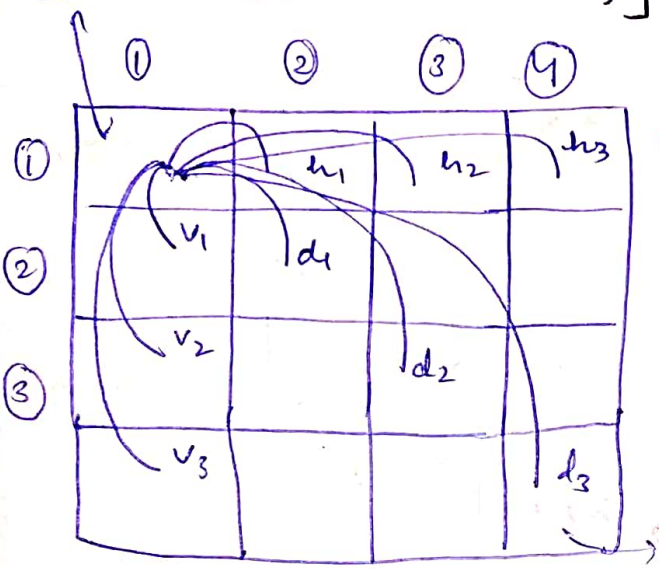
Source

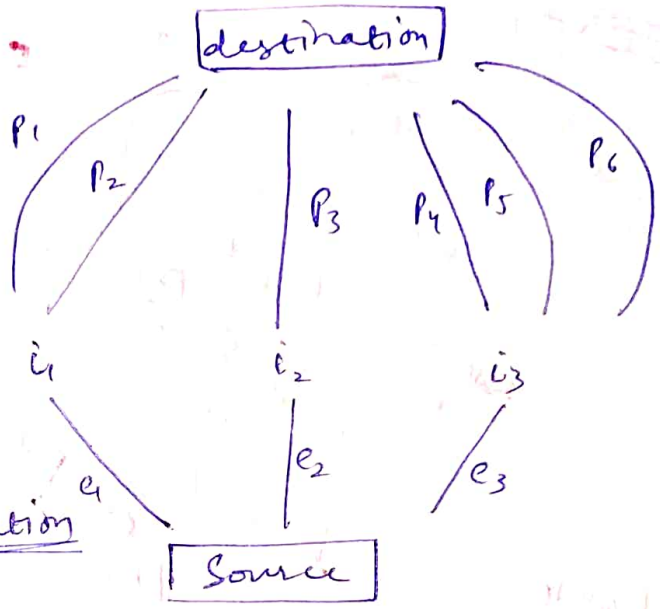
h_1, h_2, h_3] horizontal steps

v_1, v_2, v_3] vertical steps

d_1, d_2, d_3] diagonal steps

Le Sakty hai





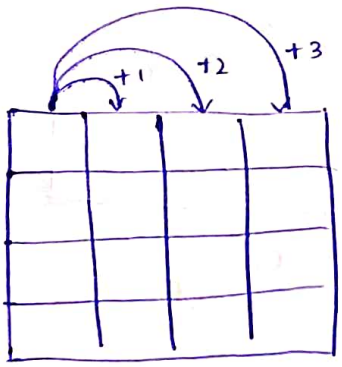

```
p s v m (S[] a) {
Scanner s = new Scanner(System.in);
int n = s.nextInt();
int m = ____________;
~~getMazePath(1,1,n,m);~~
ArrayList<String> paths = getMazePaths(1,1,n,m);
Syso(paths);
}
// sr = source row
// sc = source column
// dc = destination column
// dr = destination row
~~ArrayList<String> getMazePaths(int sr, int sc, int dr, int dc);~~
p s ArrayList<String> getMazePaths(int sr, int sc, int dr, int dc){
// HORIZONTAL MOVE
for(int ms=1; m <= dc-sc; ms++)
{ ArrayList<String> hpaths = getMazePaths(sr, sc+ms, dr, dc);
  for(String hpath: hpaths){
      paths.add("h"+ms+hpath);
  }
}
```

// Vertical Move

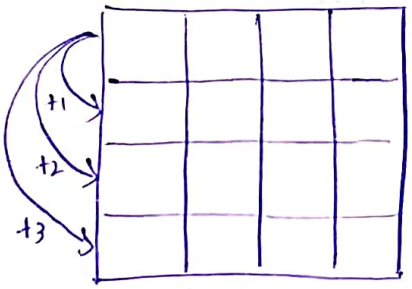
```
for (int ms=1; ms<=dr-sr; ms++){
ArrayList<String> vpaths = getMazePaths(sr+ms, sc, dr, dc);
   for (String vpath : vpaths) {
      paths.add("v"+ ms + vpath);
   }
}
```

// Diagonal Move

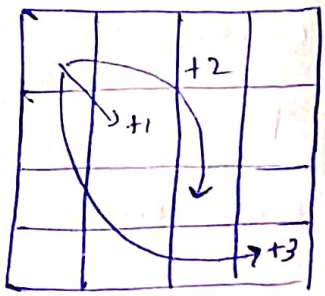
```
for (int ms=1; ms<=dr-sr && m<=dc-sc; ms++){
ArrayList<String> dpaths = getMazePaths(sr+ms, sc+ms, dr, dc);
for (String dpath : dpaths) {
   paths.add("d"+ms + dpath);
}
}
return paths;
}
```



Max size of = dc - sc
Jump along a row



Max size of = dr - dc
Jump along a column



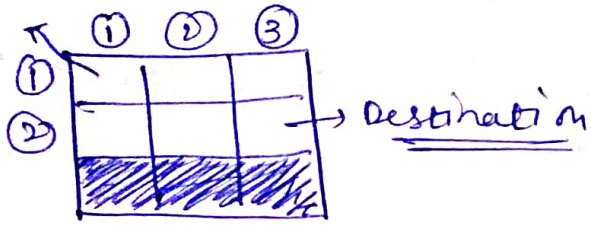
Max size of Jump along row and columns
{ ms<=dr-sr && ms<=dc-sc }

BASE CASE

```
if (sr==dr && sc==dc){
Arraylist<String> bres = new Arraylist<>();
 bres.add("");
 return bres;
}
else if(sr>dr || sc>dr){
Arraylist<String> bres = new Arraylist<>();
 return bres;
}
```

Low-level Thinking

Source

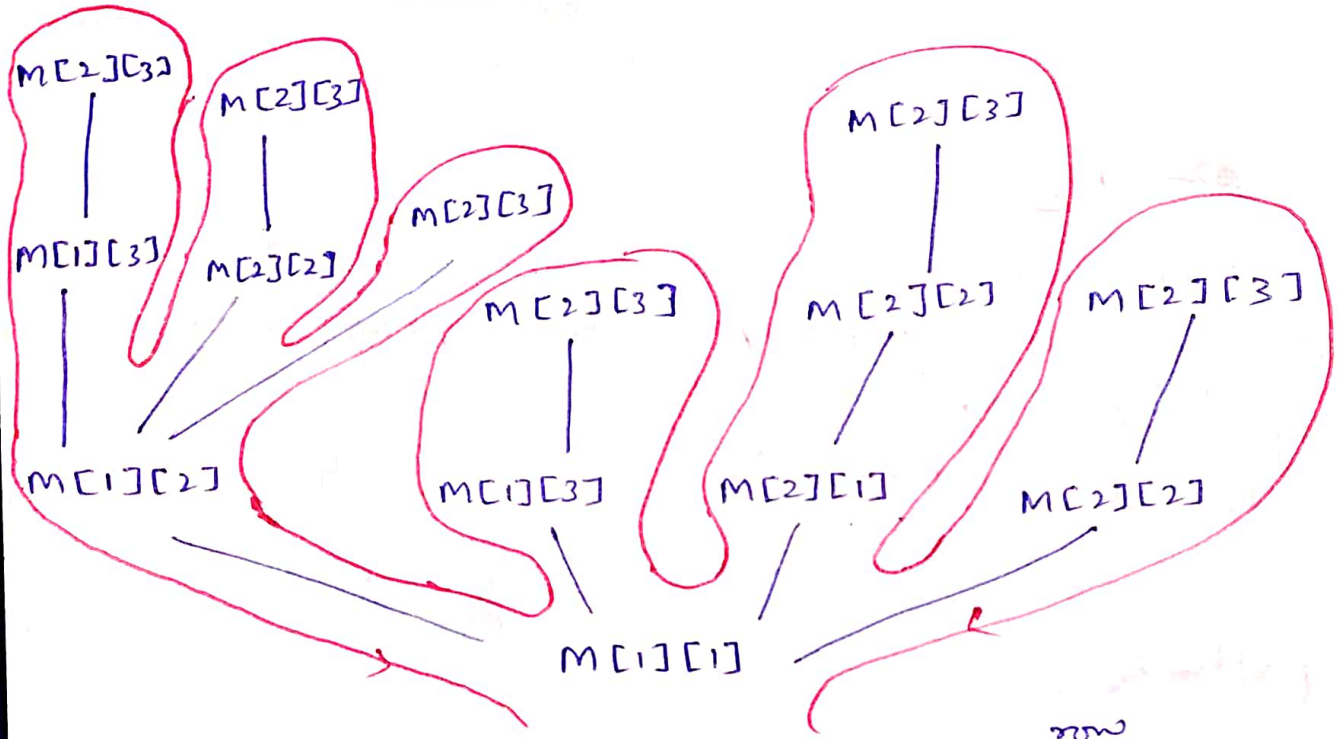


M [2] [3]

This is 2×3 matrix

∴

Enter Tree

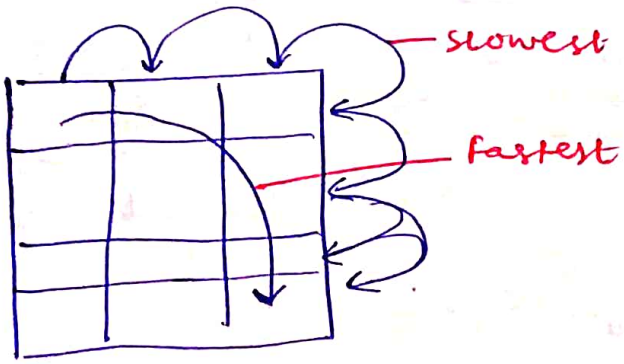


Time Complexity

TC depends on 3 recursion calls ← row, column, diagonal

$$T(n,m) = T(n,m-1) + T(n-1,m) + T(n-1,m-1)$$

- $T(n,m)$ → Total Time Complexity
- $T(n,m-1)$ → Row Call
- $T(n-1,m)$ → Column Call
- $T(n-1,m-1)$ → Diagonal Call



Fastest way → Longest Jump in diagonal

Slowest Way (Worst Case Scenario) : One by One step in rows and then one by one step in columns.

we assume that the no. of rows is greater than the no. of columns.

$T(n,m) = T(n,m-1) + T(n-1,m) + T(n-1,m-1) \quad \because (n > m)$

↳ maximum impact on Time complexity

$\therefore$ Replace $T(n,m-1)$, $T(n-1,m-1)$ → with $T(n-1,m)$

$\therefore$ we get,

$$T(n,m) \leq 3T(n-1,m)$$

TC : Time Complexity

- As we assumed the no. of rows is the largest.
- And traversal of columns depends upon the no. of rows.
- so, they will be responsible for the overall TC the maximum.
- If we replace TC function (other) with column TC function recursive call,
- Equal sign changes as column complexity is the largest so its thrice will always be greater than or equal to the overal complexity.

$\therefore$ Calculations

Since, $T(n,m) \leq 3T(n-1,m)$ (1)

$3\,T(n-1,m) \leq 3\,3T(n-2,m)$ (2)

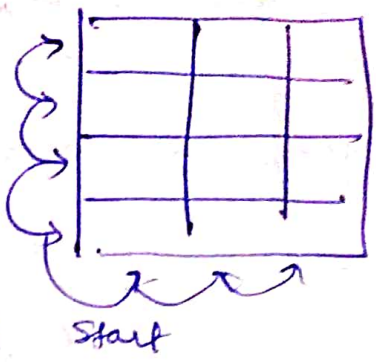
$3^2\,T(n-2,m) \leq 3^2\,3T(n-3,m)$ (3)

$\vdots \qquad \vdots \qquad \vdots$

$3^{n-1}\,T(1,m) \leq 3^{n-1}\,3T(0,m)$ (n)

$$T(n,m) \leq 3^n\,T(0,m)$$

= We have moved along one column.
Now, we will move along the row
∴ TC will depend upon $T(n, m-1)$



We know that,

$$T(n,m) = T(n,m-1) + T(n-1,m) + T(n-1,m-1)$$

Since $T(n-1,m)$ has executed now
TC will depend upon $T(n,m-1)$.

$$\therefore \boxed{T(n,m) \leq 3T(n,m-1)} \quad \text{---} \quad (a)$$

Since,
rows have been executed completely
∴ put $n=0$ in (a)

$$\therefore T(0,m) \leq 3T(0,m-1)$$

Now,

$$T(0,m) \leq 3T(0,m-1)$$
$$3T(0,m-1) \leq 3^2 T(0,m-2)$$
$$3^2 T(0,m-2) \leq 3^3 T(0,m-3)$$
$$\vdots$$
$$3^{m-1} T(0,1) \leq 3^m T(0,0)$$

$$T(0,m) \leq 3^m T(0,0)$$

Space Complexity
↳ $O(1)$

$$\therefore T(n,m) = 3^n T(0,m)$$
$$T(0,m) = 3^m T(0,0) \quad \text{---} \quad \{\because T(0,0) = 1\}$$

$$\therefore T(n,m) = 3^n 3^m = \underline{3^{n+m}}$$

Time Complexity ⇒ $O(3^{n+m})$