

DAY
5
DSA

Recursion in ArrayList continued..

1. Date :- 27 October

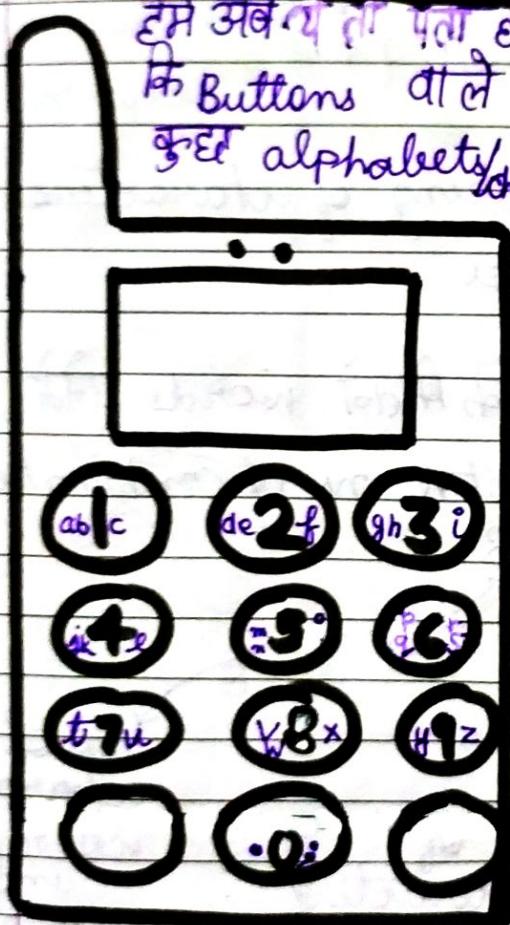
2. Day :- Wednesday

3. Agenda :- 1. Get KPC → CW

2. Get Maze Path → CW

3. Get Maze Path with multiple paths → HW
Keypad combination

Get KPC



0 →	• ;
1 →	abc
2 →	def
3 →	ghi
4 →	jkl
5 →	mno
6 →	pqr
7 →	tuv
8 →	wxz
9 →	yz

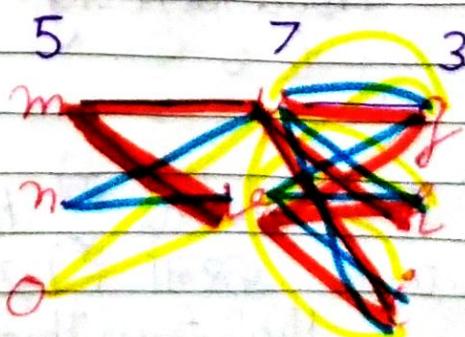
इसे एक String जैसा होगा उस String में Integers होंगे, eg → "573"

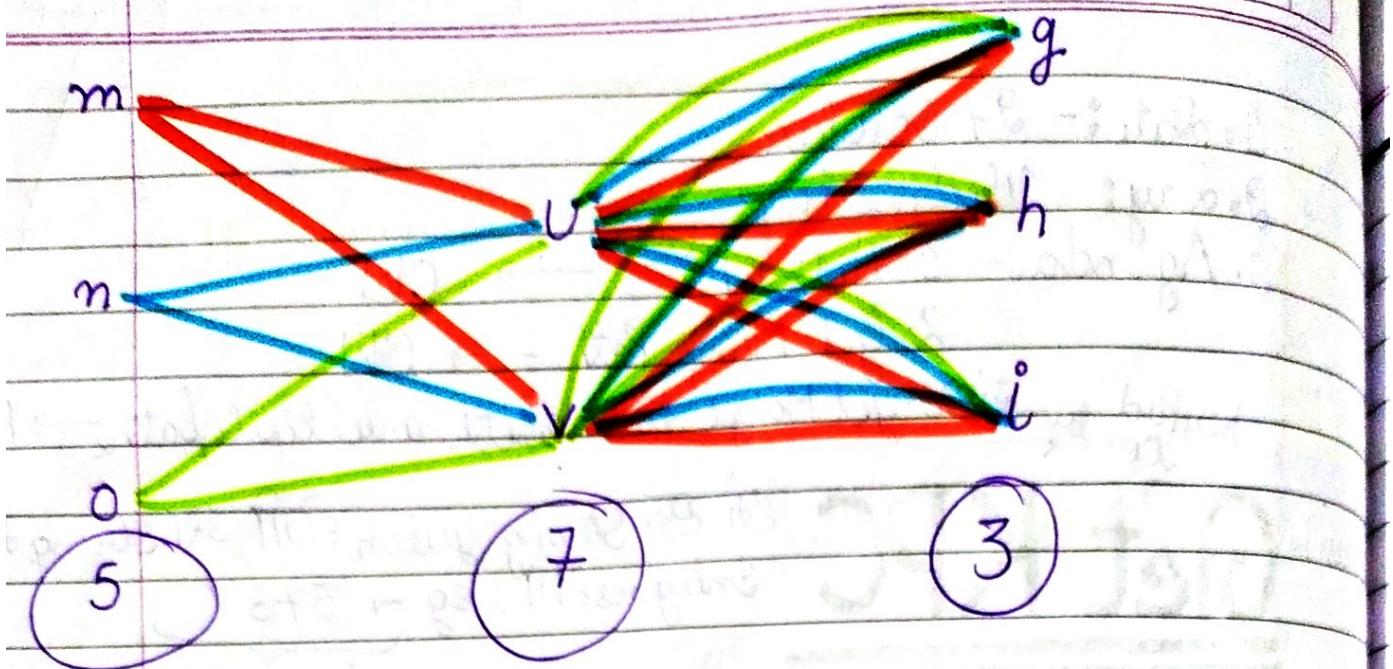
It's a string consisting of integers

कि Buttons पर के Phone में हर Integer के corresponding कुहाँ alphabets भी लिखे होते हैं character

eg → 5 के corresponding m
7 के corresponding tu
3 के corresponding ghi

तो हमें अब से सब 5, 7, 3 के corresponding जैसी characters हैं उनको map करके जो words मिलेंगी, जो सब words को एक ArrayList में add करके, as output print करेंगी हैं तथा arrayList को।





This is how mapping of characters should take place.

अब हमें ये क्या पता चलेगा कि कितने words की

Map करने के बाद So, total no. of combinations

$$\rightarrow 3 * 2 * 3$$

no. of characters corresponding to 5.

no. of characters corresponding to 7

no. of characters corresponding to 3.

हमें सभी words को Arraylist में add करके return करना है। हमें ये question recursively

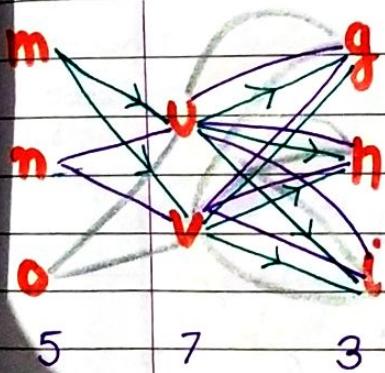
Recursion → High Level Thinking

→ Low Level Thinking

- Expectation
- Faith
- Expectation meets faith
- Day run & draw stack

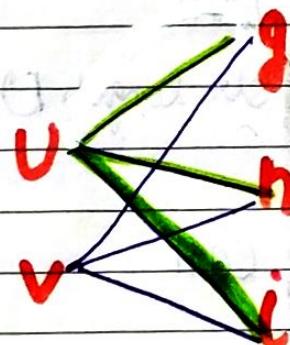
EXPECTATION

हम expect करते हैं कि अमर हमें 573 string मिली है जो Input तो हमें output में 5, 7, 3 के corresponding characters को map करके जो शी words मिले, हमें वो output में मिलेंगे ArrayList में stored as string.



FAITH

हम faith/विश्वास करते हैं कि अगर हमारा code हमें 573 string के लिए output देसकता है तो 573 की Substring 73 के लिए वो output जरूर देगा ही क्योंकि 73 के output में जितने भी words मिले, उन सब words को concatenate करेंगे।

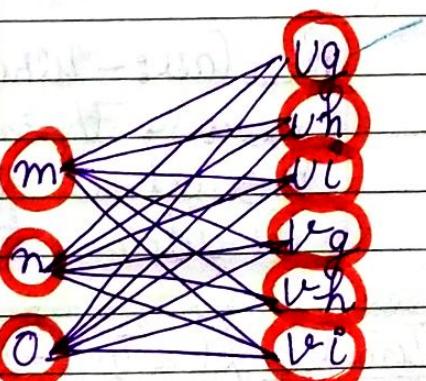


EXPECTATION MEETS FAITH

हम desired output print करने के लिए 573 string के लिए, हमें 5 index के corresponding alphabets m/n/o को (73 के output में जितने भी words मिले, उन सब words को concatenate करेंगे।

जैसे add करना है और वो output में words मिलेंगे, वो सब ArrayList में add करेंगे और return करेंगे।

So, we have to add each of the character m/n/o to each & every output of "73".



Output = ["mug", "muh", "mui", "mvg", "mvh", "mvi", "nug", "nh", "nui", "nvg", "nvh", "nvi", "oug", "oh", "oi", "ovg", "ovh", "ovi"]

So, we are done with High Level Thinking

Low Level Thinking → It is done to find the

base case.

Eg → 367

→ Dry run & draw Stack.

Page No.	
Date	

["]

Empty String in ArrayList

ghi
jklrs
stu

t + []
GetKPC()

u + []
GetKPC()

str.length = 0 = Base Case

7
Call

Call

str.length > 1

p + []
GetKPC(7)

q + []
GetKPC(7)

r + []
GetKPC(7)

s + []
GetKPC(7)

6 7

Call

str.length

g + []
getKPC(67)

h + []
getKPC(67)

i + []
getKPC(67)

str.length = 3

Base Case:- When the string length becomes zero, then a new ArrayList base result is created. A blank string is added to it & it is returned.

Time Complexity :- $O(n)$ → The time complexity is linear because a recursion call is made along with using "for loop".

Space Complexity :- $O(1)$ → Bcz no extra space required. However, if we include space used in recursive stack, then space complexity is $O(n)$.

GetKPC <code>

Page No.	
Date	

```
→ import java.util.*;  
→ import java.io.*;  
→ public class Main  
{  
    public static void main (String args)  
    {  
        Scanner scan = new Scanner (System.in);  
        String str = scan.nextLine();  
        ArrayList<String> words = getKPC (str);  
        System.out.println (words);  
    }  
}
```

we have
created a
global
array

of
String called
Codes that
store the
key to character elements, here
key is index & characters are character of string at that index

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

```
public static ArrayList<String> getKPC  
(String str)
```

```
{  
    if (str.length() == 0)
```

Base
Case
if
String
length
becomes
zero

```
    {  
        ArrayList<String> bres = new ArrayList<>();  
        bres.add (" ");  
        return bres;  
    }
```

eg 653

मेरी

653 का

किया

जो है

53 पर

recursion

call करता

है।

जो भी

faith है

first character
string का

\rightarrow char ch = str.charAt(0);
 \rightarrow string ros = str.substring(1);

rest of the string

\rightarrow ArrayList<String> mres = getKPC(ros);

result of rest of the string

\rightarrow ArrayList<String> mres = new ArrayList<>()

ASCII of '0'

48

1 = 49

2 = 50

3 = 51

& so on

To convert char to integer, always subtract from it.

my result

String code = codes[ch - 48];

first character

ch मे stored

गया string का

अभी अब String/character की form मे है eg - "653" से

दर्शाएं "6" निकला तो वे अभी वी character ए integer नहीं हैं, हमें codes array मे 6 index पर वी String है जो निकला है ए code मे, ले हमें codes[6] pass करना होगा। अब इसके पास 6

integer नहीं "6" character है, so we convert "6" character to 6, we can subtract "0" character from "6" character bcz ascii of "6" is 54, ascii of "0" is 48,

& as a result we will get 6
 $[6 - 48] = 6$ so now we can get string at codes[6]

Meeting Expectations using faith

Page No.	
Date	

```
for (int i=0; i<node.length(); i++)  
{  
    character at index i in code string  
    char chi = node.charAt(i);  
  
    for (String vrstr : vres)  
    {  
        for each String resultString  
        of result of rest of string  
        mres.add(chi + vrstr);  
        }  
        character i in code string  
        for myresult string + result  
        add. ↑.  
  
    }  
    return mres;  
}  
} → myresult
```

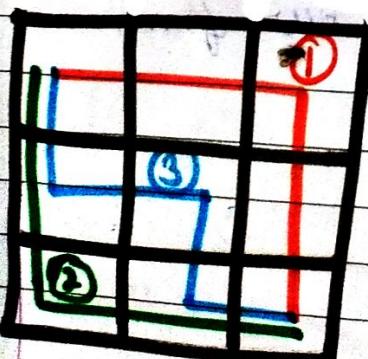
→ We apply a for loop for every keyboard combination of "53" which is stored in vres and then add each character corresponding to 5 to every output of '53'. This is eventually added in our final ArrayList mres.

Get Maze Path

- हमें एक 2-D Maze^{given} के लिए की dimension
- हम n, m (no. of rows, columns) given के Maze/ grid के form में
- हम उस 2-D Maze की grid की Top Left corner पर रहे [1, 1] cell of this maze
- हम अब इस Top left corner से Maze की Bottom Right corner पर पहुंचा है।
- हमें ये Path moves करने में लगानी steps लगती है जो हम 1 time पर 1 step ले पाते हैं और वस horizontal या vertical direction में ही steps ले सकते हैं हम 1 या diagonally भी step नहीं ले सकते।
- हमें ये सब paths print करते हैं जिनके द्वा start point से end point तक पहुंचे पर following the above mentioned rules.

eg. → अगर हमें एक 3×3 Maze given है।

row column



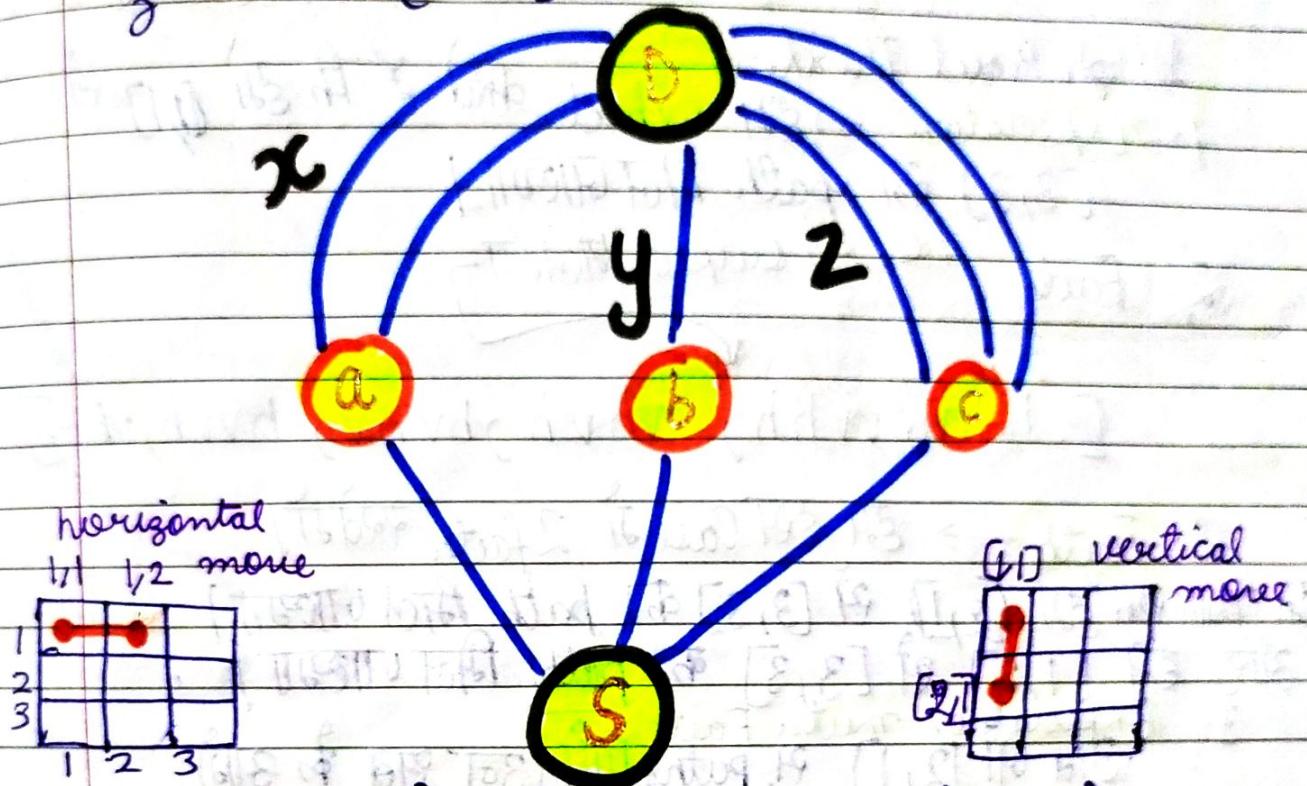
- | | |
|---|------|
| ① | hhvv |
| ② | vvhh |
| ③ | vvhv |
| ④ | hvhr |
| ⑤ | hvph |
| ⑥ | vhhv |

This question is based on same approach as we saw for Get stairs Path

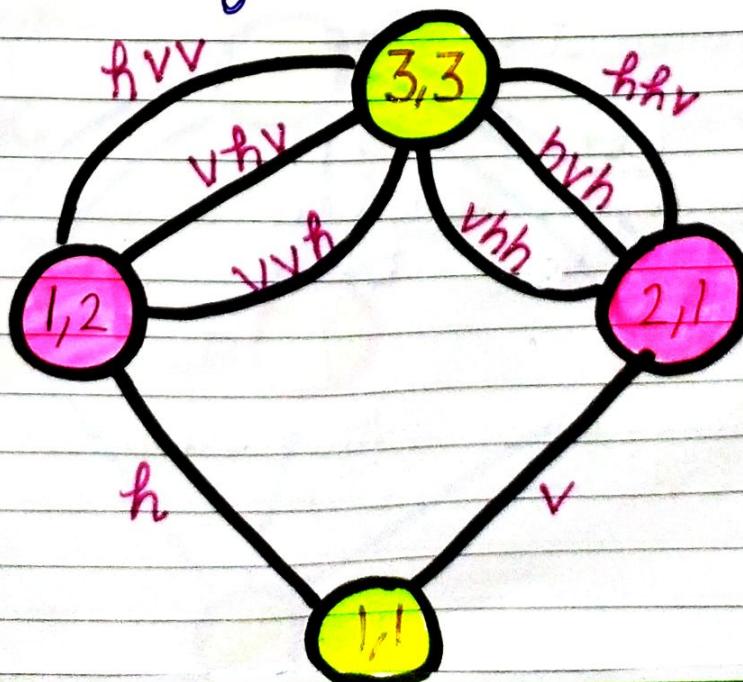
Page No.	
Date	

अगर S source और D destination हैं।

- x तरीके से a से D तक जाने के
- y तरीके से b से D तक जाने के
- z तरीके से c से D तक जाने के



इस starting में सबसे पहले $[1,1]$ cell से या तो $[2,1]$ cell पर जाएंगे vertical move करके, या किरण $[1,2]$ cell पर जाएंगे horizontal move करके।



तो ही पे problem recursively solve

करनी है।

Page No.

Date

Recursion

→ High Level Thinking

Expectation

Faith

Expectation meets faith

→ Low Level Thinking → Deepen & draw stack.

1) High Level Thinking

1. Expectation → expect करो और भी हो (1,1)
- [3,3] का path मिल जाएगा।
i.e. दो output मिलेगा।

[hhvv, vvhh, vhvh, hvhv, hrvh, vrhv]

2. Faith → हम इस Case में 2 faith रखेगे।

हम किसी समें हमें [2,1] से [3,3] का path मिल जाएगा।

और हमें [1,2] से [3,3] का path मिल जाएगा।

3. Expectation meets, Faith

अब जो [2,1] से paths मिले, उन सब के आगे v add करे और result ArrayList में डालदो। और जो [1,2] से paths मिले, उन सब के आगे h add करे और result ArrayList में डालदो।

$$V + \begin{bmatrix} hvh \\ vhb \\ hhv \end{bmatrix} \quad h + \begin{bmatrix} hvv \\ vhv \\ vvh \end{bmatrix}$$

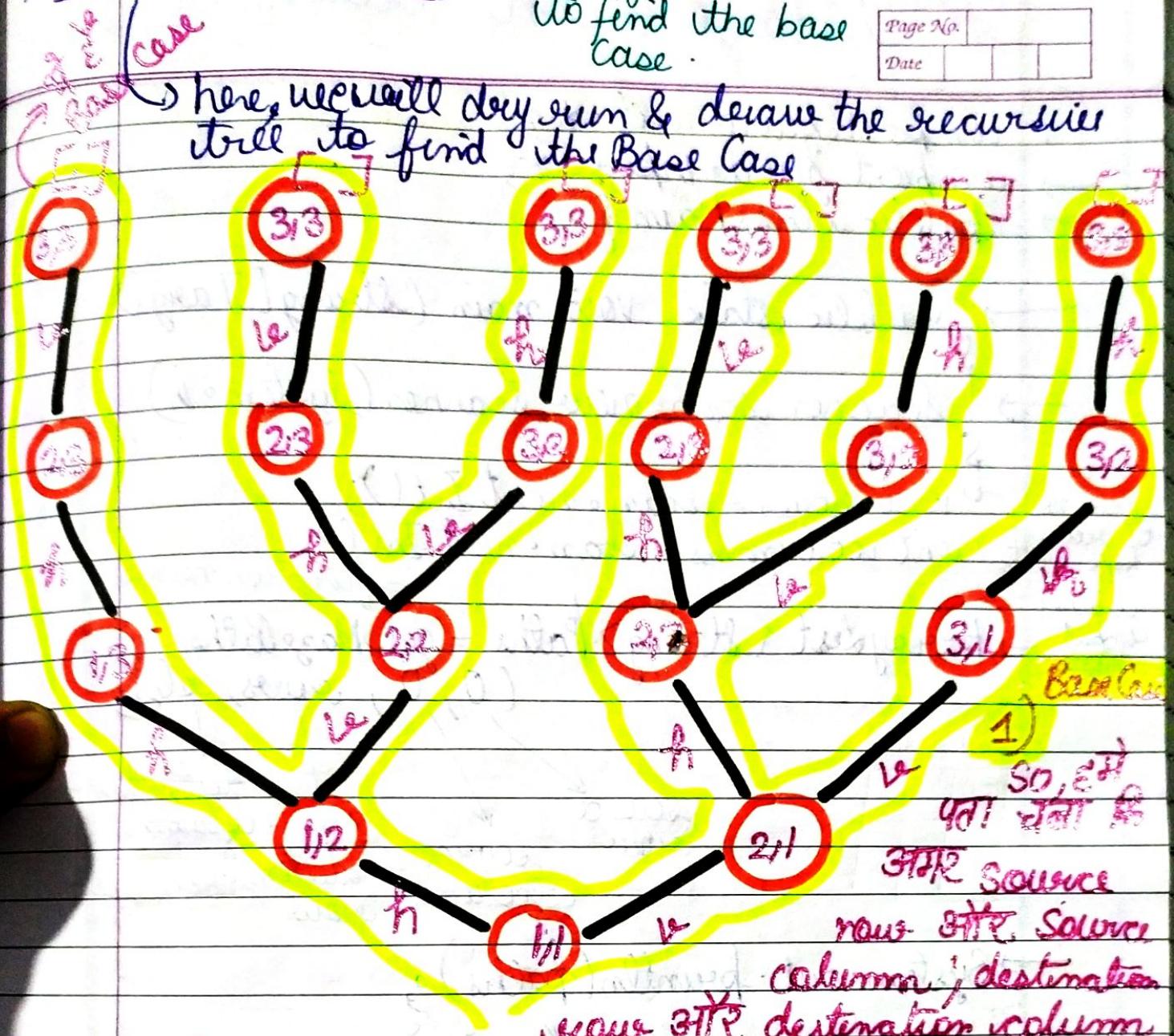
Paths from
[2,1]

Paths from
[1,2]

[vhvh, vvhh, vhhv, hhvv, hrhv, hrvh]

Low Level Thinking → Dry run & draw stack
to find the base case.

Page No.	
Date	



जब असे destination column के equal हो जाएं, तो उनके new ArrayList द्वारा दिये गये base result के लिए, उसमें एक blank string path add होता है और उसके बाद उसका last element वापस दिया जाता है।

[1] मालव 6,3 पर आरे पास move करने का तरीका क्या है?

Base Case 2) असे source row, destination row तो बड़ी ही जाएं या source column, destination column से बड़ा ही जाएं, तो उनके ArrayList द्वारा दिये गये base result के लिए, उसमें एक blank string add होती ही दी जाती है। उस position के move करने का कोई path ही नहीं हमारे पास।

Get Maze Path <code>

Page No. _____
Date _____

```
→ import java.util.*;  
→ import java.io.*;  
→ public class Main {  
    →     public static void main (String [] args)  
    →     {  
        →         Scanner scan = new Scanner (System.in);  
        →         int rows = scan.nextInt();  
        →         int columns = scan.nextInt();  
        →         ArrayList <String> Paths = getMazePaths  
        →             (0, 0, rows, columns);  
        →         System.out.println (Paths);  
    }
```

→ System.out.println (Paths);
 ↓
 output फूट अरेयलिस्ट
 में से कोने प्रिंट करेगा।

→ public static ArrayList <String>
 getMazePaths (int sr, int sc,
 int dr, int dc)

source row

destination row

source column

destination column

{ if ($sr > dc \text{ || } sc > dc$)

ArrayList<String> bres = new ArrayList<>();
return bres;

bres is an array list

else if ($sr == dc \text{ & } sc == dc$)

huge difference
if

{ ArrayList<String> bres = new ArrayList<>();
bres.add("");
return bres;

bres is empty
String
my horizontal paths

→ ArrayList<String> mypaths = new ArrayList<>()

→ ArrayList<String> myhpaths = getMazePaths

($sr, sc + 1, dc, dc$);

→ ArrayList<String> myvpaths = getMazePaths

($sr + 1, sc, dc, dc$);

my vertical paths

row

no col

col 1 Hatched

Vertically

column

no row

col 2 Hatched

horizontal

row 1 E

→ for (String myhpath : myhpaths)

for each string element in the

ArrayList myhpaths

mypaths.add("h " + myhpath);

for each String v in mypath
in the Arraylist mypaths

for (String mypath : mypaths)

{ mypaths.add(" " + v + mypath); }

}

return mypaths;

}

Time Complexity

$O(2^n) \rightarrow$ This time complexity
is exponential because
for each state, 2^2 recursion
calls are made.

Space Complexity

$O(1) \rightarrow$ No extra Space required.

But if we consider space used
by Recursion Stack, then Space
Complexity is $O(n)$.

— *Yamanna Verma*