Day 52 DSA

Recursion with Arraylist cont..

Date :- 28 October, 21
Day :- Thursday
Agenda :- 1. Get Maze Paths With Jump
2. Print Subsequence
3. Print KPC
4. Print Stair Path
5. Print Maze Path

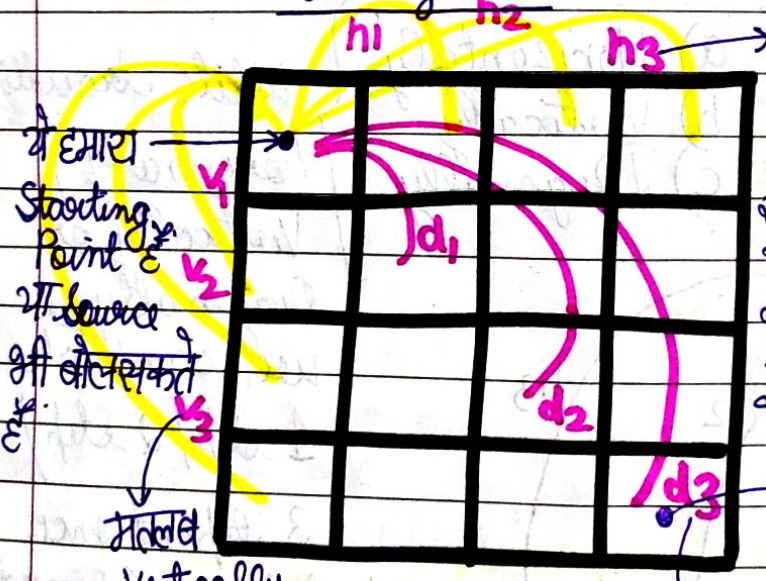
This question on of Recursion with ArrayList Module

These questions are from Recursion On The WayUp Module

Ques:1.

Get Maze Paths with JUMPS

ये Question Get Maze Path का थोड़ा बड़ा हुआ रूप है। → हम Maz. में या grid में या horizontal या vertical ही travel कर पाते है!



ये हमारा Starting Point है या Source भी बोल सकते है.

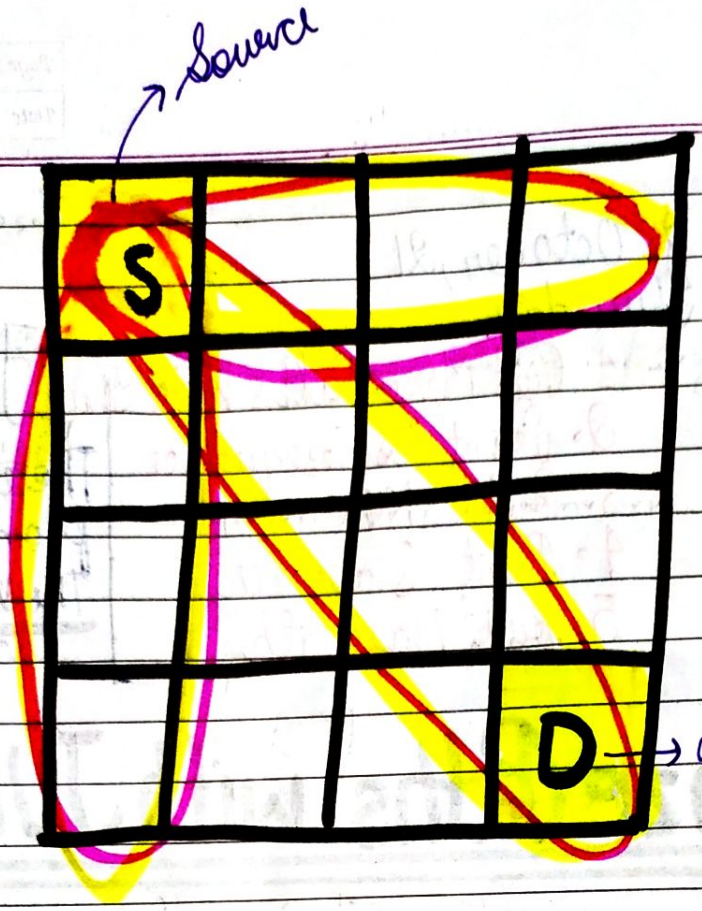
मतलब horizontally 3 Step भी चल सकते है 2 Step भी 1 Step भी क्योंकि max columns 4 है।

मतलब vertically 3 Step भी चल सकते है 2 Step भी या 1 भी

मतलब diagonally 3/2/1 step चल सकते है।

ये हमारा end point है या Destination Point भी बोल सकते है।

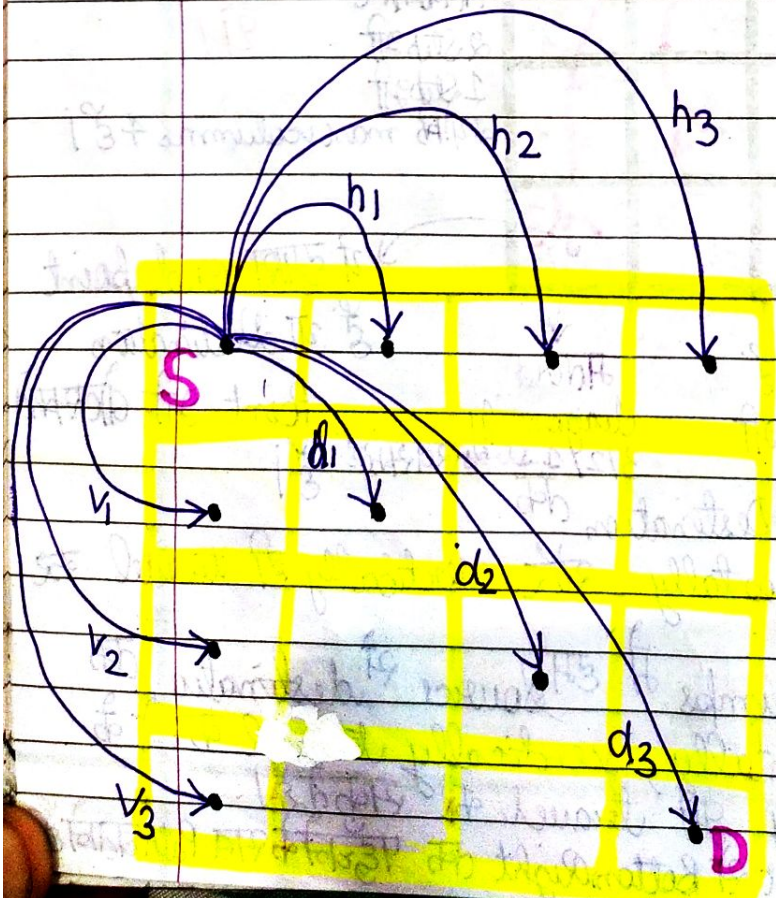
* Get Maze Path में हम Source से Destination तक पहुँचने के लिए के वल horizontally और vertically ही travel कर पाते है।
* Get Maze Path with Jumps में हम Source से destination तक पहुँचने के लिए Horizontally, verdically travel करने के साथ साथ diagonally भी travel कर सकते है।
* अब Solution में हमें Top Left से Bottom Right तक पहुँचने के सब Path बताने है.



We can travel

a) Horizontally
b) Vertically
c) Digonally

} While travelling Horizonally, Vertically or Diagonally, we can take 1 Step / 2 Step / 3 Steps at once.



Expectation:- We need all paths from Source to Destination (Horizontally, Vertically & Diagonally) taken 1 Step/2 Step/ 3 Steps / any no of Steps in Direction

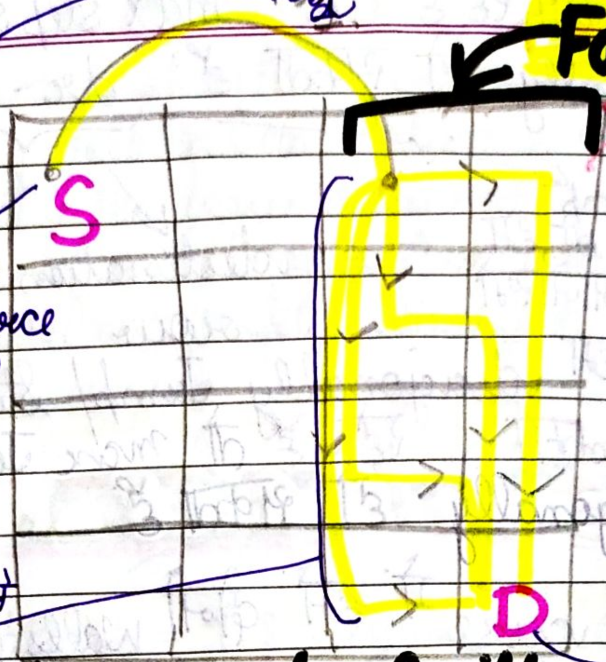
Expectation meet Faith

h2

Self Work

हमारा Self work होगा कि हम h2 लगाकर सभी paths के आगे जो हमे Recursively मिले हैं।

ये Source row, Source column है



Faith is that we will get paths from arr[0][2]

These 4 are the paths that can be there from arr[0,2]

हमने ये सब रस्ते recursion की help से मंगवालिये

→ हमें Dest row Dest Col तक जाना है

Expectation meets faith

→ So we need to add h2 in front of GetMazePaths (arr[0,2]

* हम Source Row & Source Column से Destination Row Destination Column तक जाने के लिए 3 तरीके से move कर सकते हैं :- Horizontally
Vertically.
Diagonally.

• और हमारा move size (Steps) हो सकता है 1, 2, 3, या any number तो Size के लिए हम loop लगाएंगे जिसमे Size 1 से (max length-1) horizontally / vertically / diagonally तक vary करेगा।

* अगर Step Size (horizontally) के loop चलरहा है तो columns vary होरहे हैं वहाँ तो maximum step हो सकता है dc − sc

dc − sc → Maximum Jump Size

dc → destination column

sc → source column

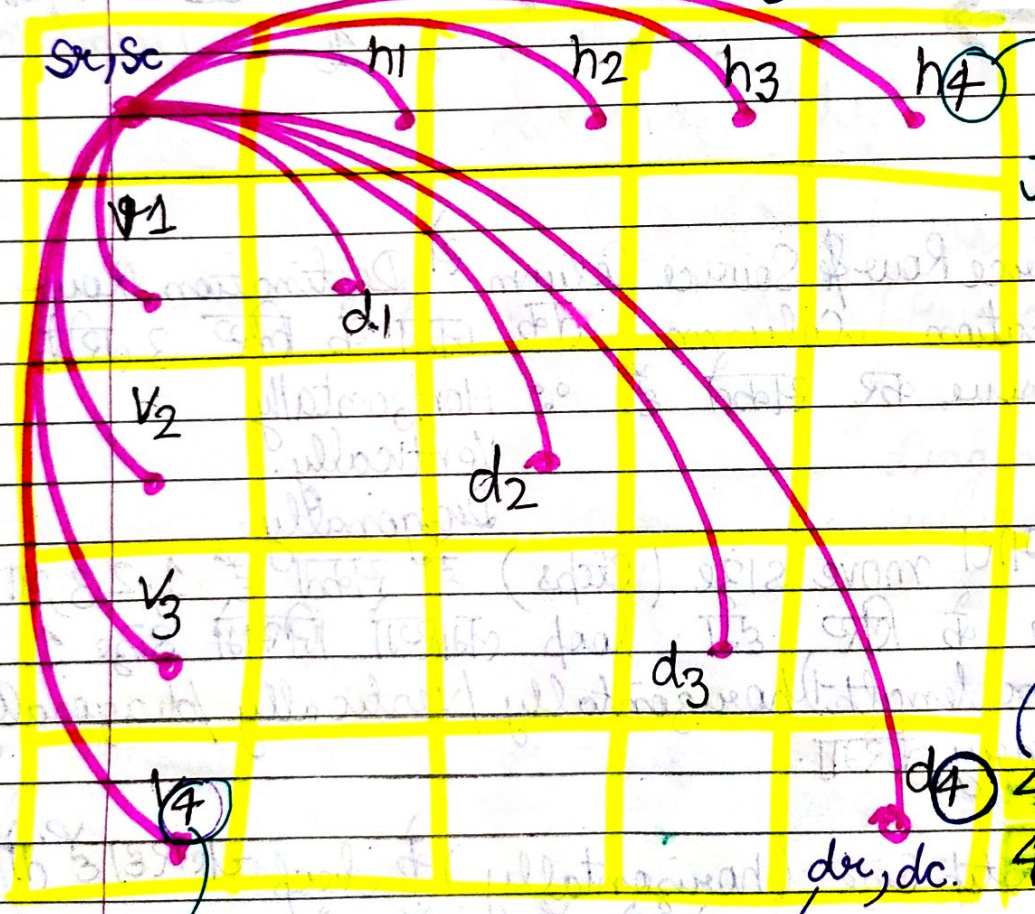
* अगर हम vertically Step Size के लिए loop लगा रहे हैं तो max Step Size / maxJump vertically हो सकता है dr - sr

इससे बड़ा Jump अगर हमने लिया तो हम array से बाहर आजाएंगे

destination row − Source row.

* अगर हम diagonally Jump/ Step Size के लिए loop लगा रहे हैं

तो इस case में हमें दोनों walls (row, column) को control करना पड़ता है [Rows भी और col भी]



Base Case

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

अगर dr और sc destination row और destination column के equal है तो it we have reached at our destination so Create a new ArrayList bres, add an empty String in it ↓ base result & return

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

for horizontal Jumps → horizontal Jumps हम 1 से (dc - sc) तक ही ले सकते है।

```
for(int hjump = 1; hjump <= dc - sc; hjump++)
{
  ArrayList<String> hpaths = getMazePaths(sr, sc + hjump, dr, dc);
```

sr → source row; sc + hjump → source column (0 + hJump) can be 0/1/2/3/4 ... dc - sc (horizontal jump); dr → destination row; dc → destination column.

We build faith, We will get all these paths recursively from this given Source row & source column.

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

for each horizontal path that are there in the Arraylist horizontal paths (we get the ArrayList recursively) add, h & hjump in front of each hpath.

↓ horizontal.

for vertical jumps

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

vgives vertical paths recursively

for each vertical path in the ArrayList vpaths

In this case, we need to add 'V' & vjump in front of vpath.

ArrayList created to save all paths (answer)

for horizontal path & vertical path & diagonal paths all paths will be saved in the ArrayList named paths

for diagonal jumps

Page No.
Date

(diagonal jump)

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

diagonal paths

Rows और columns दोनों के लिए condition है यहाँ

In this case, "d" और djump को हम dpaths के आगे add करेंगे। ये path हमे recursively मिले हैं।

Expectation	Faith	Expectation meet faith
We want to print all possible paths from source to destination in a maze.	Recursion will give us the paths from a spot that is directly reachable from the source.	We will get the paths from an initial spot to the destination spot using recursion (The faith). Before printing it will add to it, the path we took to reach that initial point.

Get Maze Paths with Jump

<code>

```
import java.util.*;
import java.io.*;
public class Main{

public static void main (String[] args)
{
 Scanner scan= new Scanner (System.in);
 int n = scan.nextInt();
 int m = scan.nextInt();
 ArrayList<String> paths =
        getMazePaths( 1, 1, n, m);
```

Source Row — Source Column

It is not (n-1, m-1) Because the source row & source column is 1, & not 0.

```
System.out.println(paths);

}

  //sr=source row
  //sc = source column
  //dr = destination row
  //dc = destination column
public static ArrayList<String> getMazePaths
(int sr, int sc, int dr, int dc)
{
```
</code>

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

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

// horizontal jumps.
for(int hjump=1; hjump<=dc-sc; hjump++)
{ ArrayList<String> hpaths =
        getMazePaths(sr, sc+hjump, dr, dc);
  for(String hpath : hpaths)
  { paths.add("h"+hjump+hpath);
  }
}

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

Page No.
Date

Base Case

diagonal jumps

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

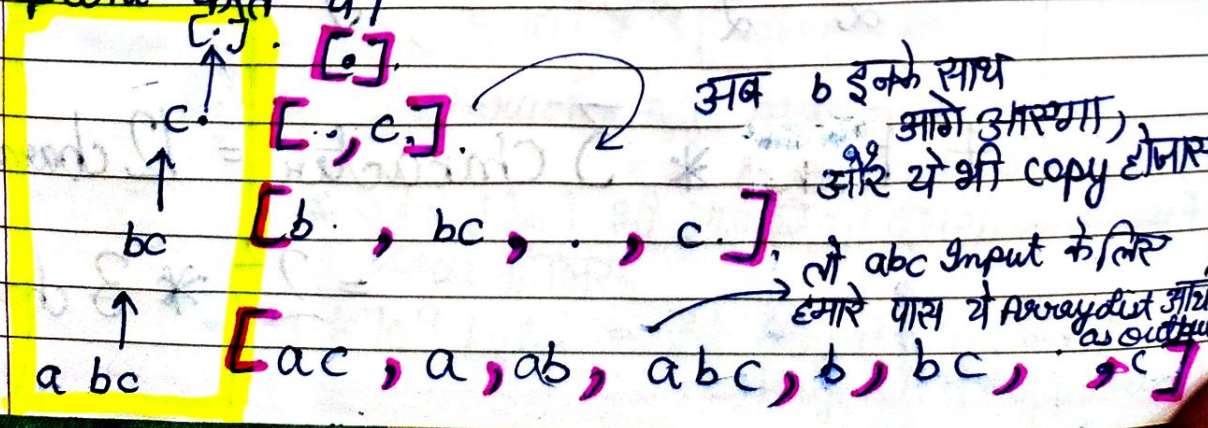
Recursion on the Way up

Page No. | Date

Questions :-
1. Print Subsequence
2. Print KPC
3. Print Stair Paths
4. Print Maze Paths With Jumps
5. Print Permutations
6. Print Encodings

Ques 1 Print Subsequence

हमने GetSubSequence कर रखा है, तो उसमें जो RecursionCall के answer आते थे। उन्हे हम Arraylist में store करके return करते थे। फिर उन्हें modify करके Answer Arraylist में save करते थे और उस Arraylist को print करते थे।



ये Total 2^3 Subsequence मिलें हैं.

Page No.
Date

abc → [" ", c, b, bc, a, ac, ab, abc]

3 → 2^3

↳ 3 Size की String के इसमें 2^3 Subsequences मिले

2 Size के Pair, 2^3 Subsequence में $\frac{2^3}{2}$ pairs बनेंगे हैं

[" ", c, b, bc, a, ac, ab, abc]

4 Pairs
- " " और abc को मिलाके Size की String की 3
- c और ab को मिलाके " " " " " " 3
- b और ac " " " " " " " 3
- bc और a " " " " " " " 3

$\frac{2^3}{2} = \frac{8}{2} = 4$ Pairs } हर Pair में 3 character

4 Pairs * 3 character = 12 character

$= 2^2 * 3$ character

n length की String → $2^{(n-1)}$ (n) characters

$n \longrightarrow \boxed{2^{n-1}\ (n)}$ characters

- 2^{n-1} → Total no. of characters in total subsequence
- n → Length of string is n
- Pairs of Subsequence is 2^{n-1}
- no of characters in each Pair

Important Fact

Ex: Length of String given = 31

→ Total no. of Characters in the ArrayList of Subsequence will be = $\boxed{2^{n-1}\ (n)}$

→ Total no. of Char = $(2)^{(30)} * (31)$.

$= (2^{10})^3 * 31$

$= (1024)^3 * 31$

↳ 1024 को अगर 1000 मानलो

$= (1000)^3 * 31$

$= (10^3)^3 * 31$

$= 10^9 * 31$ characters

Size of 1 character = 1 byte (atleast)

= 2 byte (In Java)

* अगर 1 byte ही माने किसी दूसरी language के code के लिए

$= 31 * 10^9$ bytes = 31 Gigabytes (GB)

↳ Huge RAM required

Date

तो अगर हम 31 Length की String के Subsequences को ArrayList में Store कराएंगे तो ये ArrayList का Size ही 31 GB होगा इतनी तो हमारे पास RAM भी नहीं होगी RAM भी 4GB / 8GB की होती है. तो इतनी बड़ी Arraylist को Store करवाना लगभग impossible है।

* तो इसलिए Better यही होगा कि जैसे ही हमें एक Subsequence मिले, उसे तभी print करादो instead of storing it in a Arraylist.

High Level Thinking

Expectation

We want all the subsequences for the provided String 'abc'

Faith

हम विश्वास रखते हैं कि अगर हम (a) first character को साथ रख के roq (rest of question) (bc) को printSubs को pass कर देंगे तो ये हमें bc के सारे subsequences provide कर देगा।

Expectation meets faith

→ printSS function को call किया गया bc (rest of question) (roq) पर, This will provide us all the subsequences with the character 'a' not added अब हमें Self work करना है. a को add करके

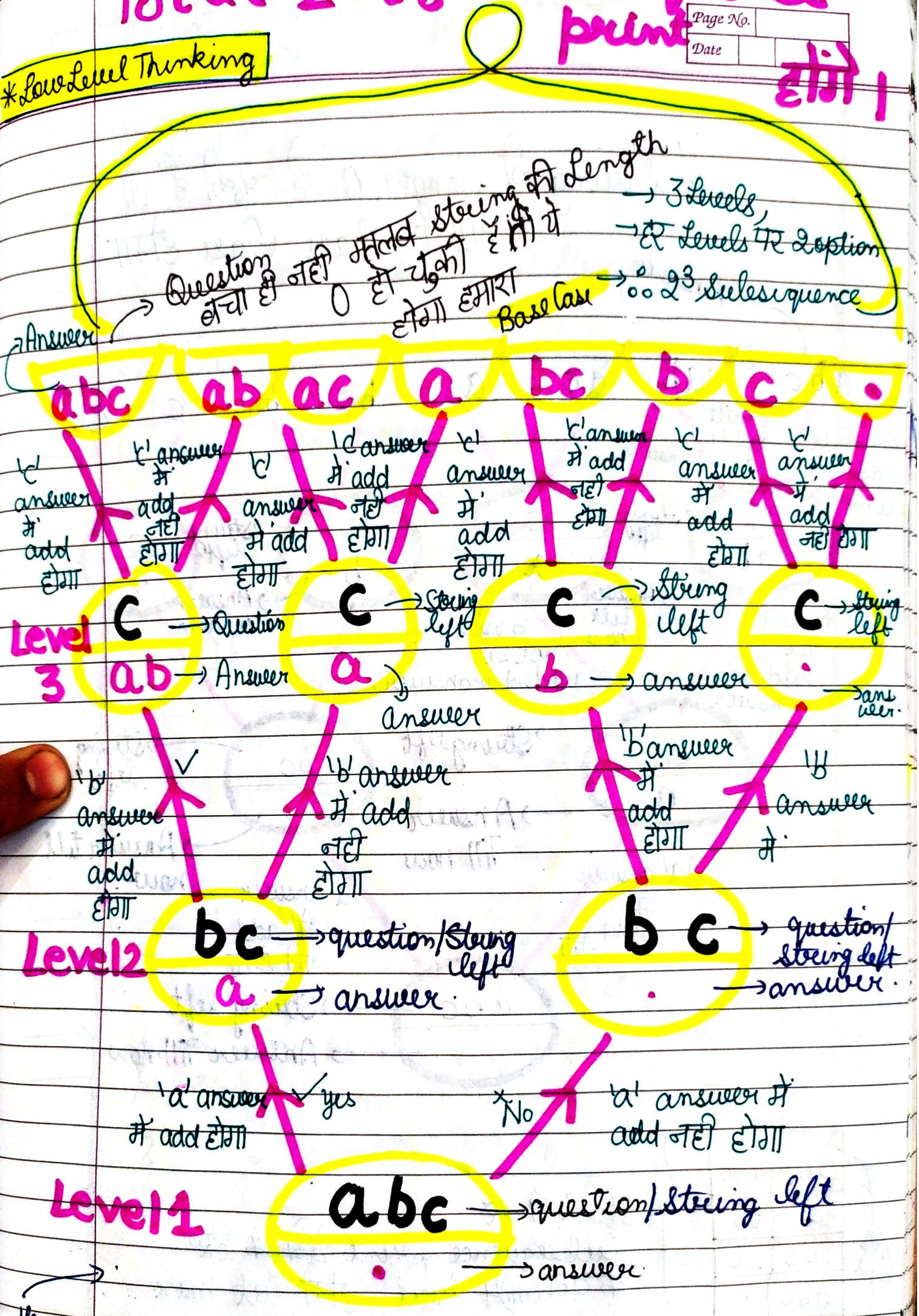
abc, ab, ac, a ← a यहाँ add हुआ

bc, b, c, . ← यहाँ a add नहीं हुआ

Total 8 Subsequence

Total $2^3 = 8$ Subsequence print होंगे।

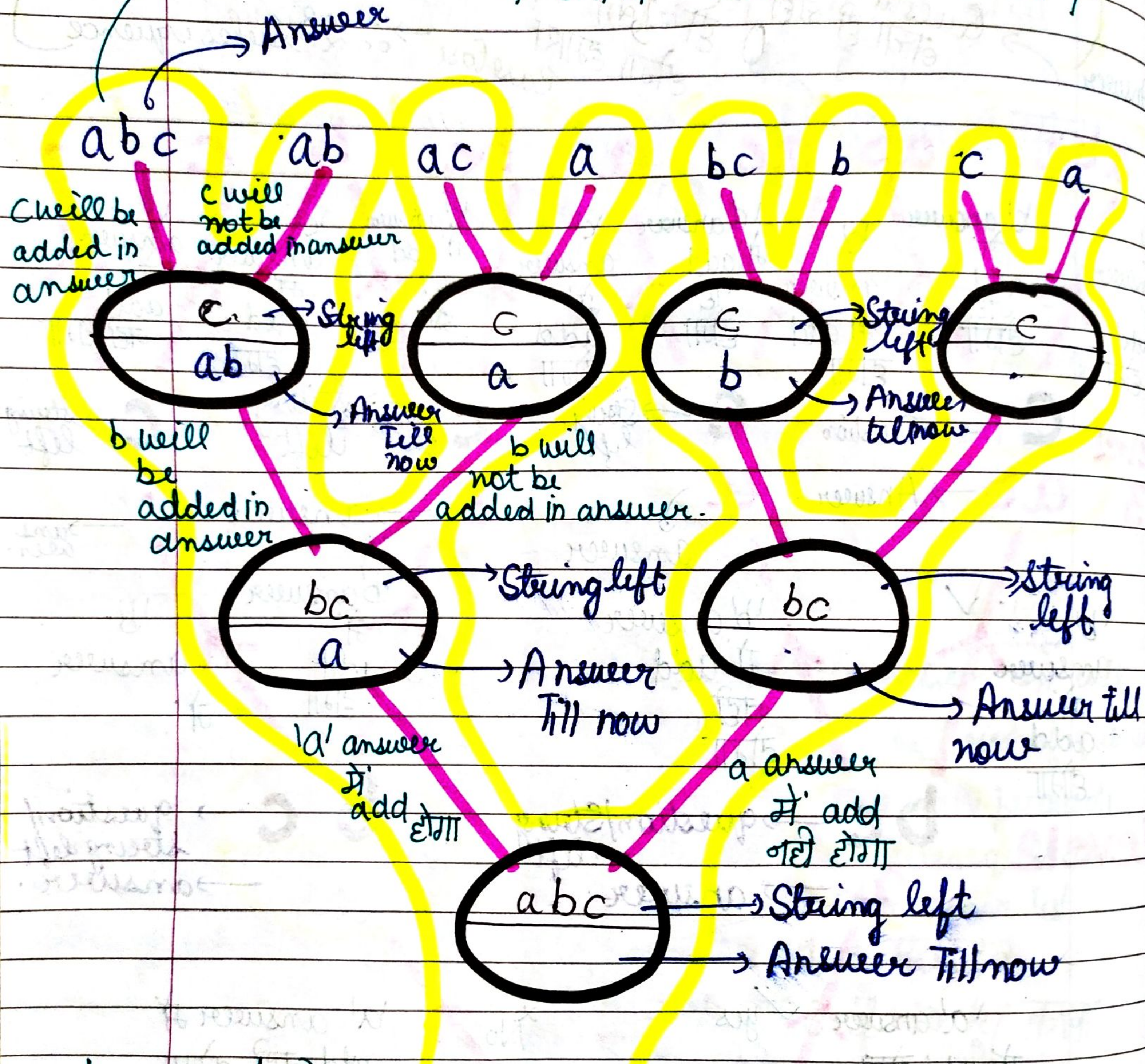
*Low Level Thinking



Here, we have used Level & Option Trick

3 Levels because 3 character in string

String की length 0 हो चुकी है तो ये हमारा Base Case होगा।



	abc
c	ab
bc	a
abc	.

n+1 level

String left

Stack में हर Subsequence print करने के लिए की तरह (n+1) level होंगे at max

Print Subsequence
<code>

```
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();      ] Input String लेली है।
  printSS(str, "");
     └→ print subsequence          Starting में answer blank pass किया है।
}
public static void printSS(String ques, String ans){
   if (ques.length ==0)
   {
      System.out.println(ans);        Base Case
      return;
   }
 char ch= ques.charAt(0);  → starting का character
 String roq = ques.substring(1);  → Starting character हटाकर जो बची String वही roq में store करदी

 printSS(roq, ans+ch);   → ans में character add करदिया
 printSS(roq, ans+"");   → answer में blank String add करदी मतलब starting character add नहीं किया
 }
}
```

Ques-2

Print KPC

Page No.
Date

KPC → Keypad Combinations

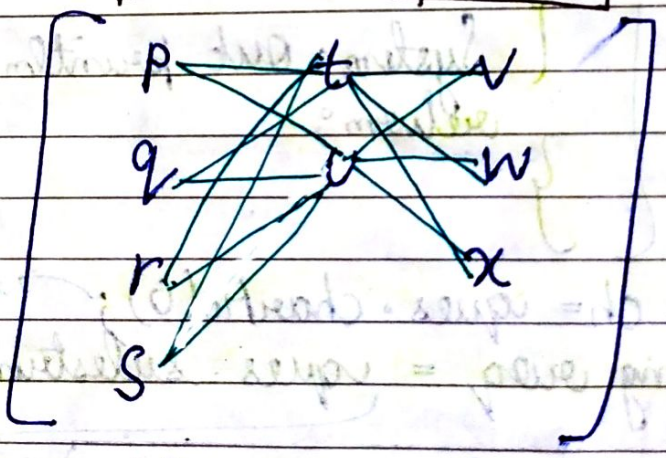
Get KPC question में हमने किया है। इसमें सब words को (Keypad Combination को) हमने एक Arraylist में store किया था और फिर उस Arraylist को print कराया था।

* Print KPC question में जब हमें एक keypad Combination मिलता है, हम तभी उसे print कर देंगे। हम Arraylist में Save नहीं करेंगे Because it increases so much Space Complexity.

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

Digits and alphabets corresponding to them.

Eg → | 6 | 7 | 8 |



We need to do mapping of character, get a keypad combination & print that keypad combination.

ptv
ptw
ptx

puv
puw
pux

qtv
qtw
qtx

quv
quw
qux

rtv
rtw
rtx

ruv
ruw
rux

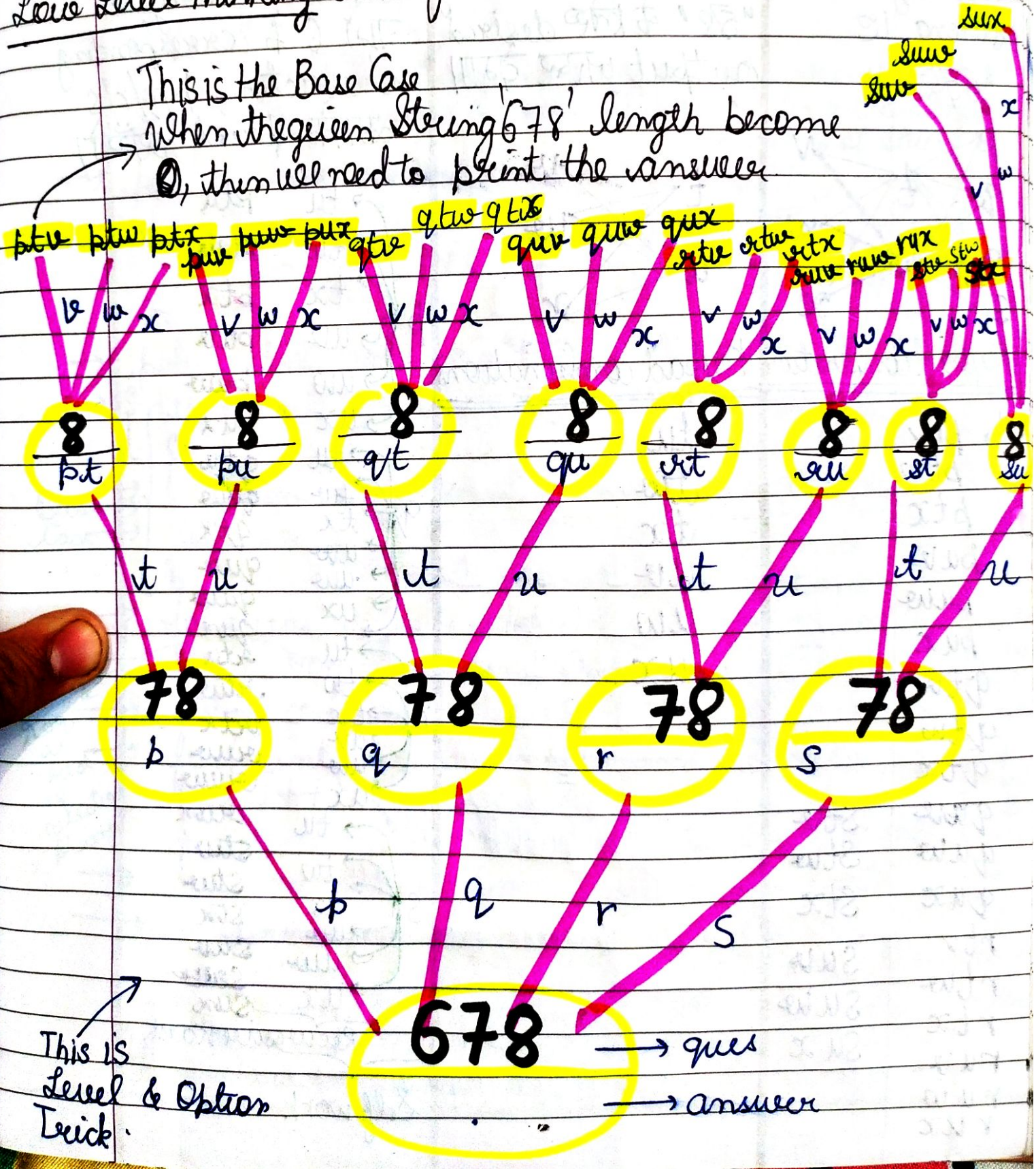
stv
stw
stx

suv
suw
sux

Keypad Combination

Love Level Thinking :- To find the Base Case

This is the Base Case
When the given String '678' length become 0, then we need to print the answer

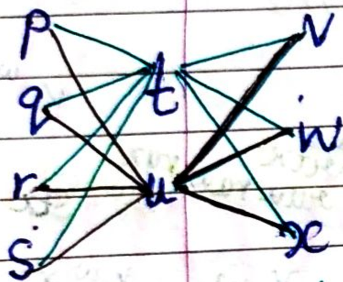


* High Level Thinking

Page No.
Date

Expectation

* We are supposed to print all the keypad combinations of a given string ie "678"

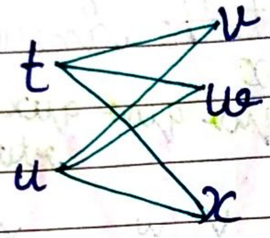


Keypad Combination

↳ ptv
ptw
ptx
puv
puw
pux
qtv
qtw
qtx
quv
quw
qux
rtv
rtw
rtx
ruv
ruw
rux
stv
stw
stx
suv
suw
sux

Faith

* हम विश्वास रखते है कि अगर print KPC function हमें "678" के लिए desired output दे सकता है तो print KPC function हमें '678' की substring "78" के लिए desired output भरकर देगा।



Keypad combinations

tv
tw
tx
uv
uw
ux.

Expectation meets faith:

हमने "78" पर recursive call की print KPC function को, तो भी हम Strings मिलेगी as keypad combinations of 78, हम उन Strings के आगे 6 के corresponding characters p/q/r/s लगाके print करेंगे।

p → tv, tw, tx, uv, uw, ux → ptv, ptw, ptx, puv, puw, pux

q → tv, tw, tx, uv, uw, ux → qtv, qtw, qtx, quv, quw, qux

r → tv, tw, tx, uv, uw, ux → rtv, rtw, rtx, ruv, ruw, rux

s → tv, tw, tx, uv, uw, ux → stv, stw, stx, suv, suw, sux

Recursive work

Self work

PRINT KPC <code>

```
import java.util.*;
import java.io.*;
public class Main{

public static void main (String[]args)
{
  Scanner scan= new Scanner(System.in);
  String str = scan.nextInt();
  printKPC(str, "");
}
```

This is a global array of String called codes which stores the key to character elements where key is index & characters are characters of String at that index.

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

public static void printKPC (String ques, String ans)
{ if(ques.length()==0)
  {
    System.out.println(ans);
    return;
  }
```

If the length of ques String become 0 then we print the answer String and return the function

Base Case

```
  char ch = ques.charAt(0);
  String roq = ques.substring(1);
  String codeforch= codes[ch-'0'];
  for(int i=0; i<codeforch.length(); i++)
  {
    char optionofchar = codeforch.charAt(i);
    printKPC(roq, ans+optionofchar);
  }
}
```

Separate the first element of ques String and store in ch.

Store rest part of ques string in roq

roq is rest of question String

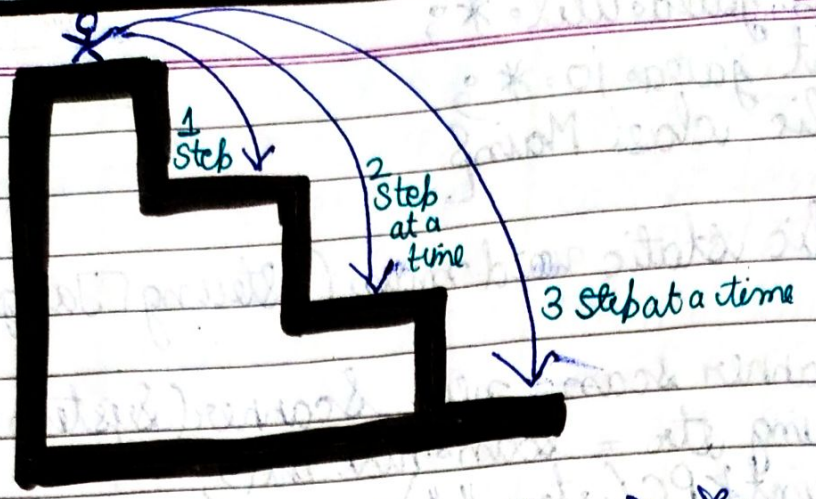
In coding "64 is not equal to 6 is 6 then from 0 to 3 for P,Q,R,S.

A loop from 0 to length()-1 of String to get index of each char of that String eg → if codeforch

The function KPC is recursively called on roq i.e 78 from 67 as the "ques" and "ans" as the sum of previous answer & character corresponding to optionofchar (P/Q/R/S)

Ques 3

PRINT STAIR PATHS



हम एक time पर 1 step ले सकते हैं
या 2 step ले सकते हैं
या 3 step ले सकते हैं

0 से 3 या 3 से 0 तक जाने के सब path बताने हैं।

4 paths 4 paths

Input → n = 3

Output →

1 1 1
1 2
2 1
3

High Level Thinking → Faith

Expectation

Expectation meets faith

Recursive Approach

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

Here, we will be using Level and Option Trick To reach n from 0.

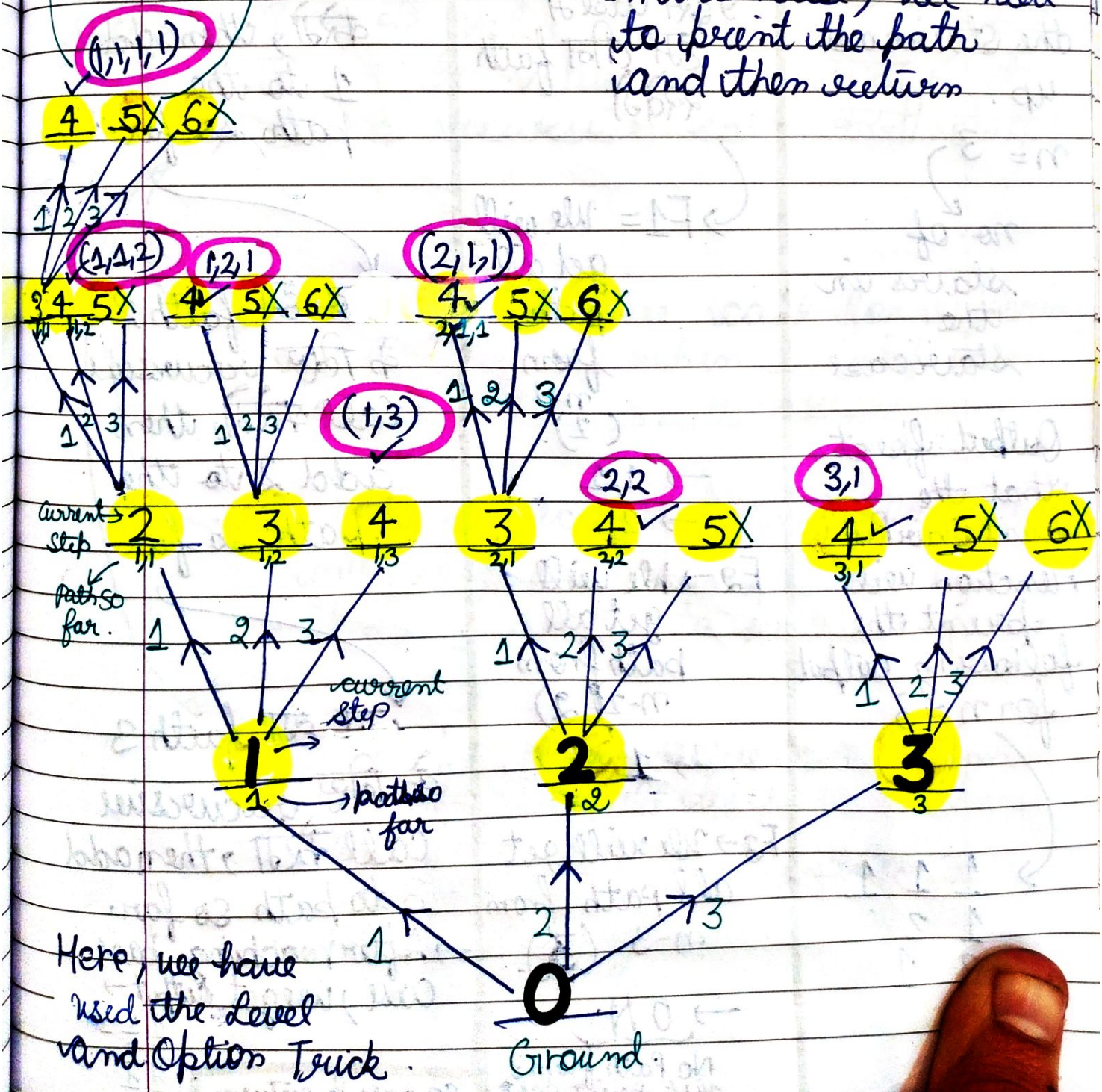
Page No. Date

* Low Level Thinking

So, These are the 2 base cases.

1) When n becomes less than 0.
(no. of steps taken are more than required steps) → In this case, just return

2) When n is equal to 0.
(no. of steps taken is equal to no. of steps required).
↳ In this case, we need to print the path and then return



Here, we have used the Level and Option Trick.

* High Level Thinking

Expectation

हमें तो सब paths print करने हैं that can be used to climb the staircase up.

n = 3
↓
no. of stairs in the staircase

We expect that the Print Stair Paths Function will print the following output for n = 3

1 1 1
1 2
2 1
3

Faith

हम विश्वास रखते हैं कि we will get the output for all 3 recursive calls.

इस case में हम तीन faith रखेंगे

F1 = We will get all path from n-1 (2).
→ 1,1
→ 2.

F2 → We will get all path from n-2 (2)
→ 1

F3 → We will get all path from n-3 (3).
→
No Path from 3 just print path so far & return

Expectation meets Faith

हमें Expectation से Faith meet करने के लिए

पहले हम Faith 1 के लिए recursive call करेंगे, then add 1 to the path so far.

फिर हम faith 2 के लिए recursive call करेंगे, then add 2 to the path so far

फिर हम faith 3 के लिए recursive call करेंगे, then add 3 to path so far.

→ After reaching base case, we get output →
1 1 1
1 2
2 1
3

Approach

Page No.
Date

Step 1) पहले ये check करो कि क्या n की value 0 से कम है? then we just need to return
first base case

Step 2) फिर हमें ये check करना है कि क्या n की value 0 के equal है? अगर n की value 0 के equal है, then just print path so far we have.
second base case

Step 3). अगर n की value base case नहीं है मतलब अगर n की value 0 से ज्यादा है तो, हमें 3 recursive call करनी होगी।

1) Make a recursive call for exploring all the paths when 1 step is taken, add 1 to path so far

2) Make a recursive call for exploring all the paths when 2 Steps are taken, add 2 to path so far.

3) Make a recursive call for exploring all the paths when 3 steps are taken, add 3 to path so far.

PRINT STAIR PATHS <code>

Date

```
import java.util.*;
import java.io.*;
public class Main{
public static void main(String[] args)
{
{ Scanner scan= new Scanner(System.in);
int n= scan.nextInt();
printStairPaths(n, "");
}
```

Initially the path so far is an empty string

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

Base Cases

```
if(n<0)
{
return;
}
if(n==0)
{
System.out.println(psf);
return;
}
```

No need to write it in double/single quotes

```
printStairPaths(n-1, psf+1);
```

String + Integer = String

```
printStairPaths(n-2, psf+2);
printStairPaths(n-3, psf+3);
}
}
```

Recc. 4

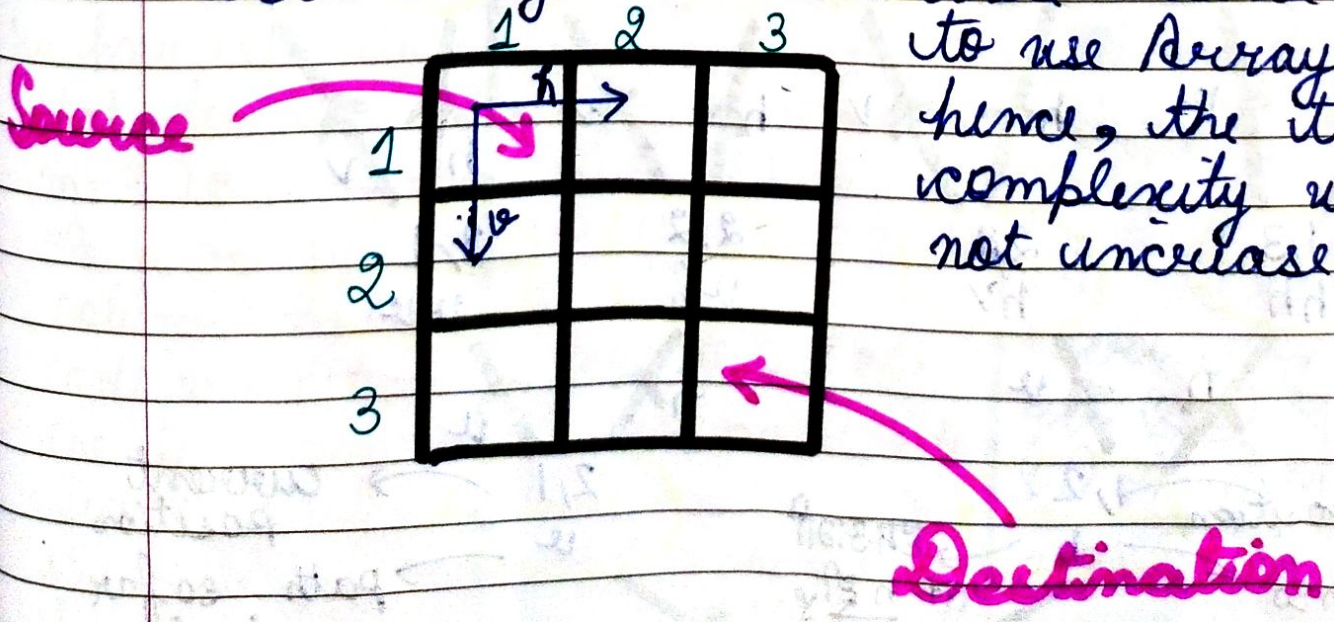
PRINT MAZE PATHS

Page No.
Date

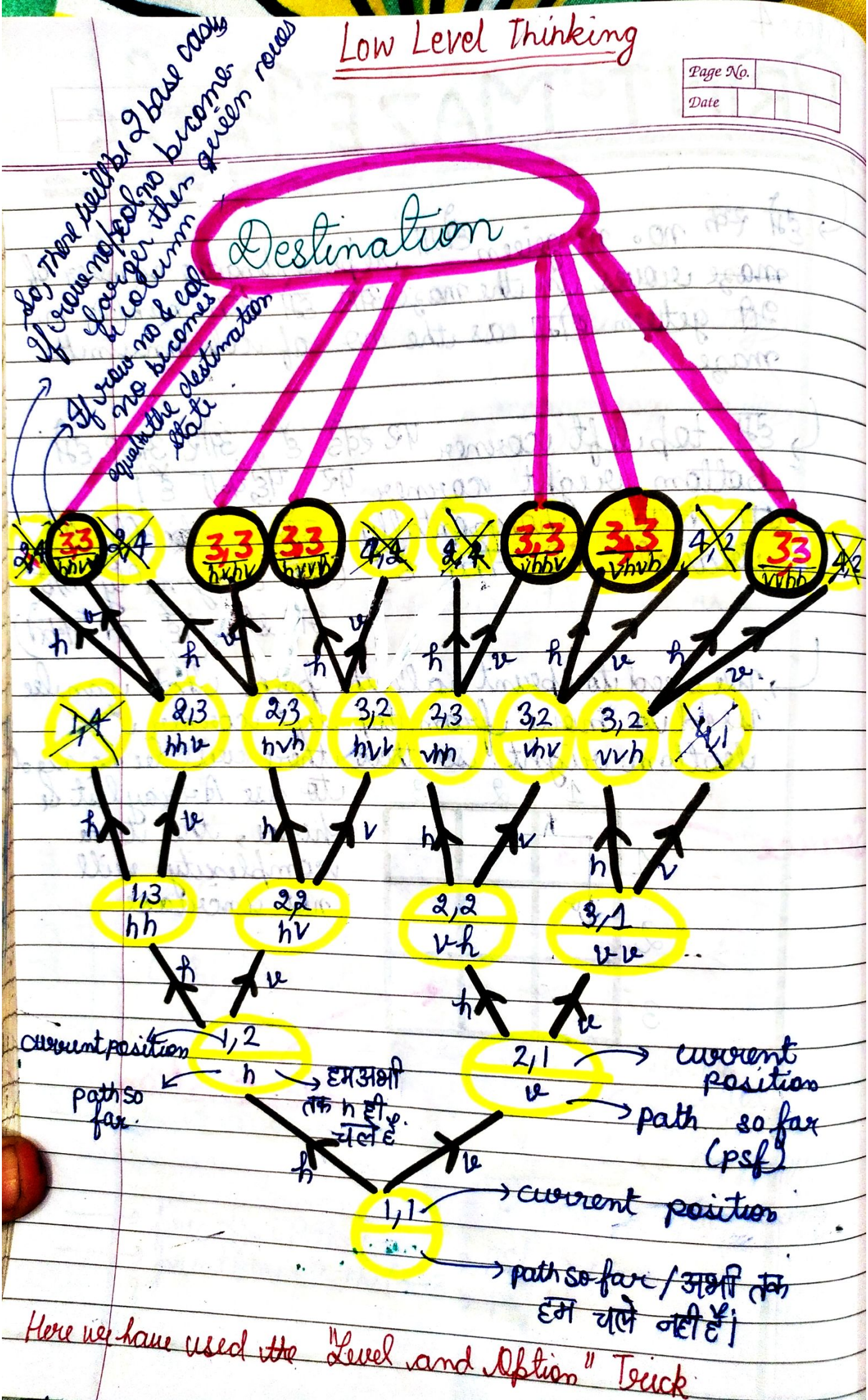
↳ हमें एक no. n given होगा। representing the number of rows in the maze. और हमें एक number m भी given होगा as the no. of columns in the maze.

↳ हम top left corner पर खड़े हैं, और अगर हमें Bottom right corner पर पहुँचना है। हम केवल horizontally और vertically ही travel कर सकते हैं। (हम diagonally कोई भी step नहीं ले सकते)

↳ We need to print all the paths that can be used to move from top left corner to bottom right. so that there will be no need to use Arraylist & hence, the time complexity will not increase.



Low Level Thinking



Here we have used the "Level and Option" Trick

So, we need to solve this problem recursively

Recursion
- High Level Thinking
  - Expectation
  - Faith
  - Expectation meets faith
- Low Level Thinking → Dry run and draw stack ↓ to find the base case.

High Level Thinking

Expectation

We have expectation that for n=3 m=3 i.e no. of row =3 & no. of column = 3, our code will return the following output if we start the (1,1) position, to reach to (3,3)

→
hhvv
hvhv
hvvh
vhhv
vhvh
vvhh

Faith

हम विश्वास रखते हैं कि अगर हमारा code हमें 1,1 से 3,3 तक का path provide कर सकता है तो वो हमें (1,2) से (3,3) और (2,1) से (3,3) तक का path जरूर provide करेगा। तो हमारे पास इस question में 2 Faith होंगे।

Faith 1 → printMazePaths(1, 2, 3, 3, psf)

Faith 2 → printMazePaths(2, 1, 3, 3, psf)

Output: hhv vhh hvh

Expectation meets faith

हम अगर horizontally move करके (1,2) पर पहुँचते हैं 1,1 से तो हमें psf में 'h' add करना होगा और अगर हम vertically move करके (2,1) पर पहुँचते हैं (1,1) से तो हमें psf में 'v' add करना होगा। ये हमारा Self work होगा। तो हमें Recursive calls करने के पहले psf update करना होगा और desired row column set करने होंगे।

h + hvv → hhvv
h + vhv → hvhv
h + vvh → hvvh
v + hhv → vhhv
v + hvh → vhvh
v + vhh → vvhh

So, after doing the Low Level Thinking, we got to know that here, we have 2 base case.

Page No.
Date

Base Case 1

When the source row become greater than the destination row or the source column become greater than the destination column, then, we need to stop and just return

```
if (sr > dr || sc > dc)
{
    return;
}
```

source row
source column
or
destination column
destination row

Base Case 2

When the destination is reached, it means if the sr == dr and sc == dc.

source row becomes equal destination row
source column become equal to destination column.

, then we need to print the path so far & return

PRINT MAZE PATHS <code>

```
import java.util.*;
import java.io.*;
public class Main{

  public static void main(String[]args)
  {
    Scanner scan = new Scanner(System.in);
    int n = scan.nextInt();
    int m = scan.nextInt();
    printMazePaths(1, 1, n, m, "");
  }

  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');
  }
}
```