DAY 53

DSA

Recursion on the Way up continued...

Date → 29 Oct, 21

Day → Friday

Agenda for Today :-

1. Print Maze Paths with Jumps
2. Print Permutations
3. Print Encodings

Ques-1

PRINT MAZE PATHS WITH JUMPS

We have done Get Maze Paths with Jumps question in Recursion with Arraylist module- तब हम paths को arraylist में store करते जाते थे और फिर उन्हें print करवा देते थे Arraylist को print करके। लेकिन ये करने से हमारी space complexity काफी बढ़ जाती थी तो this is the disadvantage of storing in Arraylist

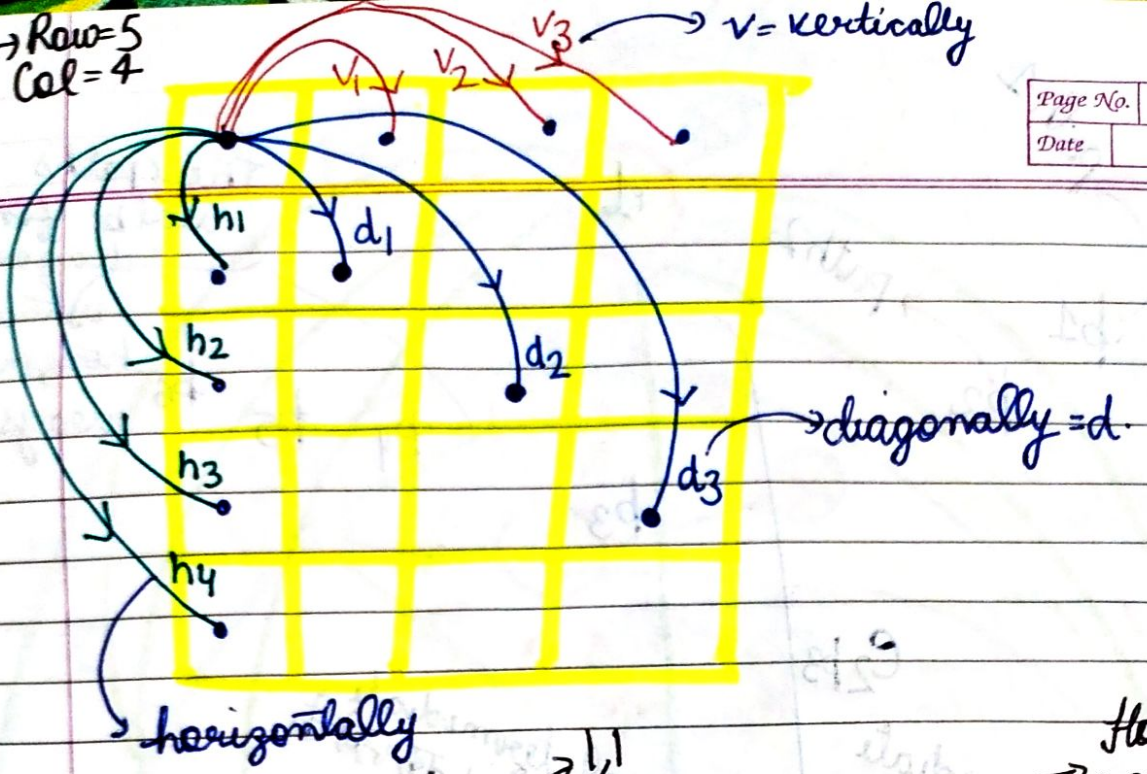
अब हम paths को store नहीं करेंगे Arraylist में सिर्फ print करा देंगे सब paths को।

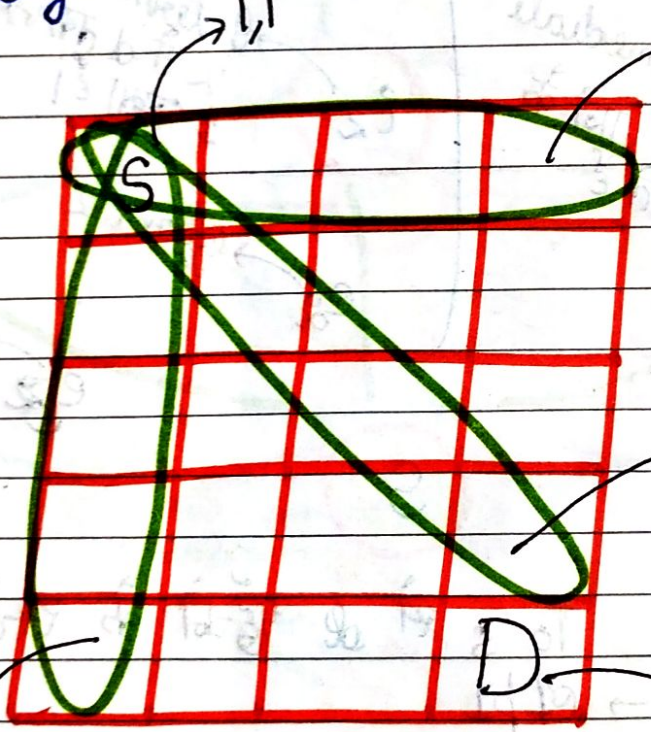
→ हमें no. of rows और no. of columns given होगी। हमें (1,1) से (n,m) पर पहुंचना है।

(n, m) → n: no. of row, m: no. of col

जब हमारा question केवल Get Maze Path होता था तब हम बस Horizontally और vertically ही travel कर पाते थे लेकिन अब हमें Get Maze Paths with Jumps question में हम vertically horizontally और diagonally भी travel कर सकते हैं।

Eg → Row=5
Col=4





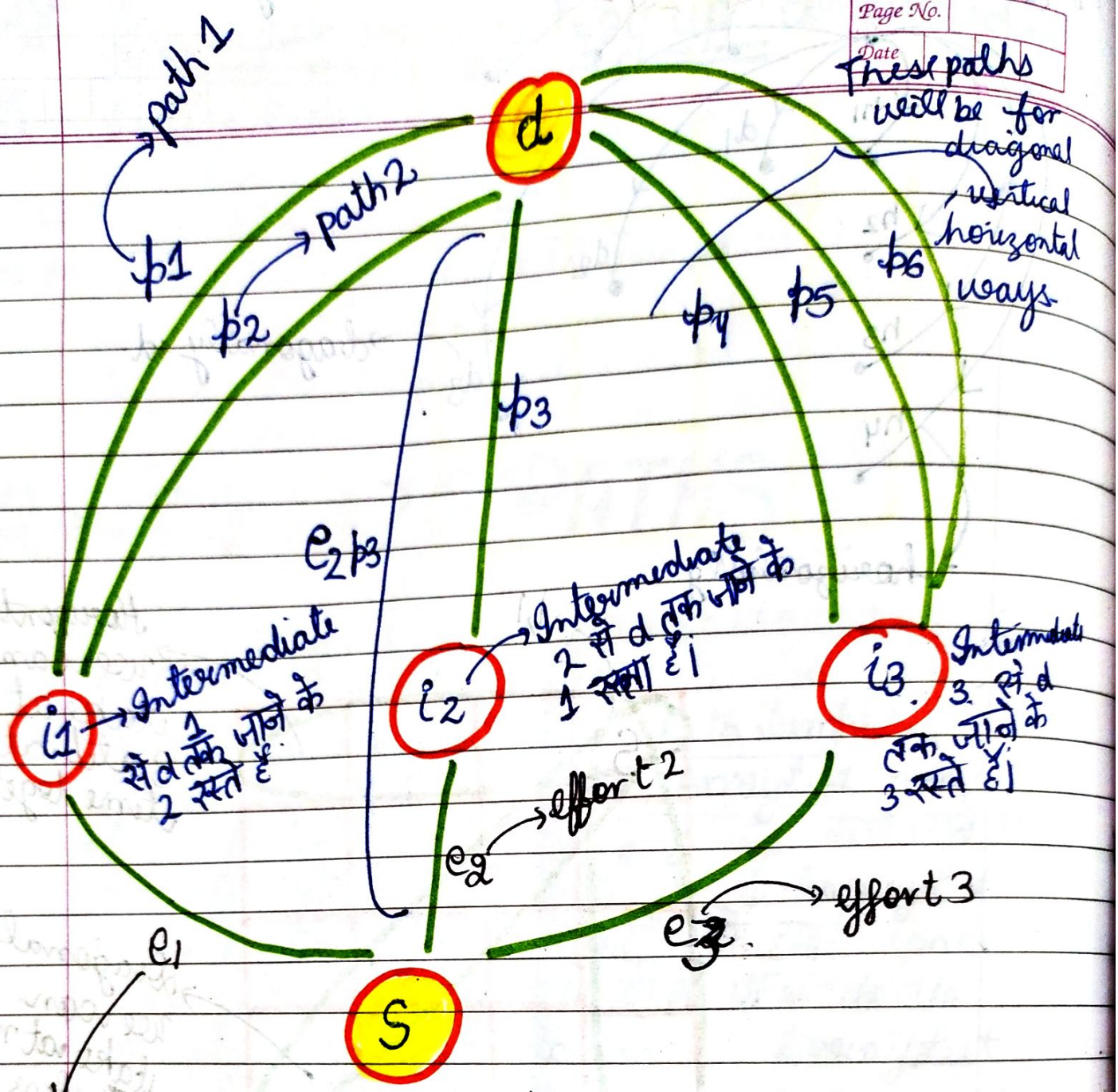
Horizontally, we can take at max 3 step at a time together

Diagonally, we can take at max 3 steps at a time together

Vertically, we can take at max 4 steps at a time together

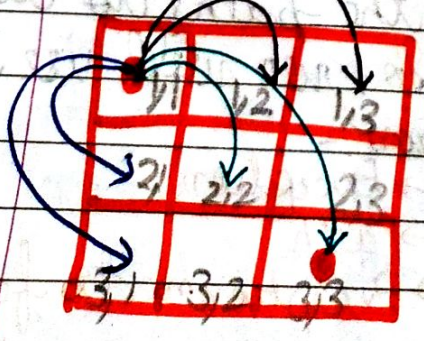
So, now we need to print all the paths to travel from source to Destination, so, we will use Recursion to solve this question :-

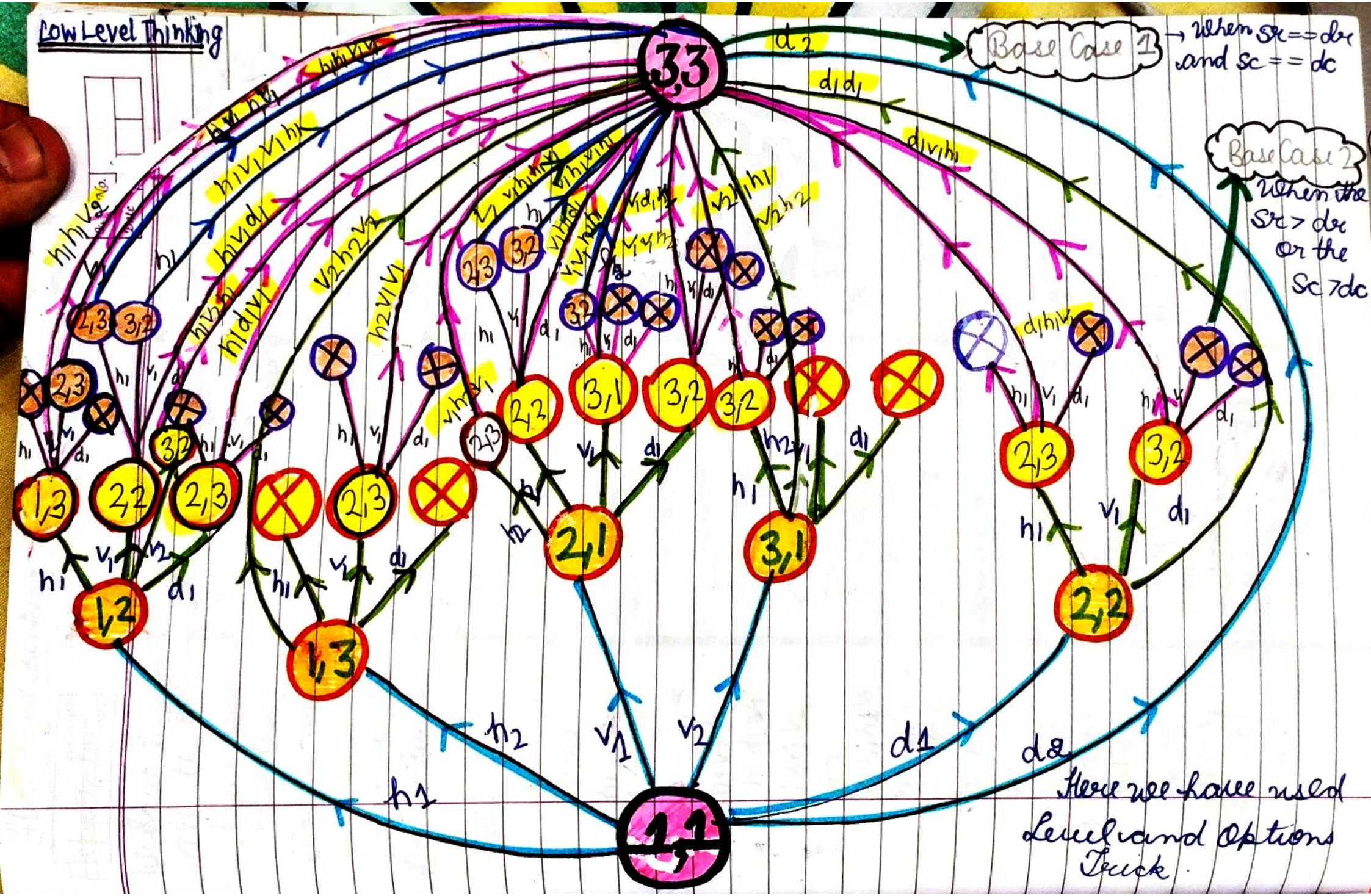
Recursion
- High Level Thinking
  - Exception
  - Faith
  - Exception meets faith
- Low Level Thinking → Dry run & draw stack to find the base case



To s से d पहुँचने के रास्ते हैं—

→ e1p1
→ e1p2
→ e2p3
→ e3p4
→ e3p5
→ e3p6




Low Level Thinking
Base Case 1
→ When Sr == dr and Sc == dc
Base Case 2
When the Sr > dr or the Sc > dc
Here we have used Level and Options Trick.

High Level Thinking

Expectation

We have to print all the paths from source to destination in the maze

Source → (1,1)
Destination → (3,3)

Output →

h1 h1 v1 v1
h1 h2 v2
h1 v1 h1 v1
h1 v1 v1 h1
h1 v1 d1
h1 v2 h1
h1 d1 v1
h2 v1 v1
h2 v2
v1 h1 h1 v1
v1 h1 v1 h1
v1 h1 d1
v1 h2 v1
v1 v1 h1 h1
v1 v1 h2
v1 d1 h1
v2 h1 h1
v2 h2
d1 h1 v1
d1 v1 h1
d1 d1
d2

Faith

In this case, we need to build faiths that recursive calls will give us the paths from an intermediate spot (eg {(2,1),(1,2),(2,2),(1,3),(3,1),(3,2) & etc..)

Faith 1 → h1
Source → (1,2)

h1 →
- h1 v1 v1
- h2 v2
- v1 h1 v1
- v1 v1 h1
- v1 d1
- v2 h1
- d1 v1

Faith 2 → h2
Source → (1,3)

h2 →
- v1 v1
- v2

Faith 3 → v1
Source → (2,1)

v1 →
- h1 h1 v1
- h1 v1 h1
- h1 d1
- h2 v1
- v1 h1 h1
- v1 h2
- d1 h1

Faith 4 → v2 (3,1)

v2 →
- h1 h1
- h2

Faith 5 → d1
Source → (2,2)

d1 →
- d1
- h1 v1
- v1 h1

Faith 6 → d2

d2 →
- d2

Expectation meets faith

We will get the paths from intermediate spots ie {(1,2),(1,3),(2,1),(3,1),(2,2),(3,3)}.

Before printing it, we will add to it, the path we took to reach to that intermediate spot.

h1 +
- h1 v1 v1 → h1 h1 v1 v1
- h2 v2 → h1 h2 v2
- v1 h1 v1 → h1 v1 h1 v1
- v1 v1 h1 → h1 v1 v1 h1
- v1 d1 → h1 v1 d1
- v2 h1 → h1 v2 h1
- d1 v1 → h1 d1 v1

h2 +
- v1 v1 → h2 v1 v1
- v2 → h2 v2

v1 +
- h1 h1 v1 → v1 h1 h1 v1
- h1 v1 h1 → v1 h1 v1 h1
- h1 d1 → v1 h1 d1
- h2 v1 → v1 h2 v1
- v1 h1 h1 → v1 v1 h1 h1
- v1 h2 → v1 v1 h2
- d1 h1 → v1 d1 h1

v2 +
- h1 h1 → v2 h1 h1
- h2 → v2 h2

d1 +
- d1 → d1 d1
- h1 v1 → d1 h1 v1
- v1 h1 → d1 v1 h1

d2 +
- d2

PRINT MAZE PATHS WITH JUMPS

<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,"");
```

Initially, the path so far is nothing so, an empty string will be passed.

```
//sr = source row
//dr = destination row
//sc = source column
//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(sr>dr || sc>dc)
    {
        return;
    }
```

Base Case 1

Base Case 2

When the source becomes equal to the destination

When the source row or column becomes greater than destination row or column

horizontal jump

```
for (int hjump = 1; hjump <= (dc - sc); hjump++)
{
    printMazePaths(sr, sc + hjump, dr, dc, psf + "h" + hjump);
}
```

vertical jump

```
for (int vjump = 1; vjump <= (dr - sr); vjump++)
{
    printMazePaths(sr + vjump, sc, dr, dc, psf + "v" + vjump);
}
```

diagonal jump

```
for (int djump = 1; djump <= (dr - sr) && djump <= (dc - sc); djump++)
{
    printMazePaths(sr + djump, sc + djump, dr, dc, psf + "d" + djump);
}
}
}
```

Ques-2.

PRINT PERMUTATIONS

We have done "Print Permutation of a String iteratively" question in the "String, StringBuilder & ArrayList" Module.

Now, we need to print all the permutations of a String, we need to not use an "ArrayList" to first store those permutations and then print instead we will see directly printing the permutation which is going to save the Time Complexity.

* So, we must know that we will see herein a String, let say → abc, and we have to print all the permutations of this String

String → "abc"

Permutations → abc
acb
bac
bca
cab
cba

These will be the permutations of 'abc'

Date

String Length = n

no. of Permutations = $!n$

* We need to print these Permutations Recursively
Earlier we have printed the Permutations Iteratively

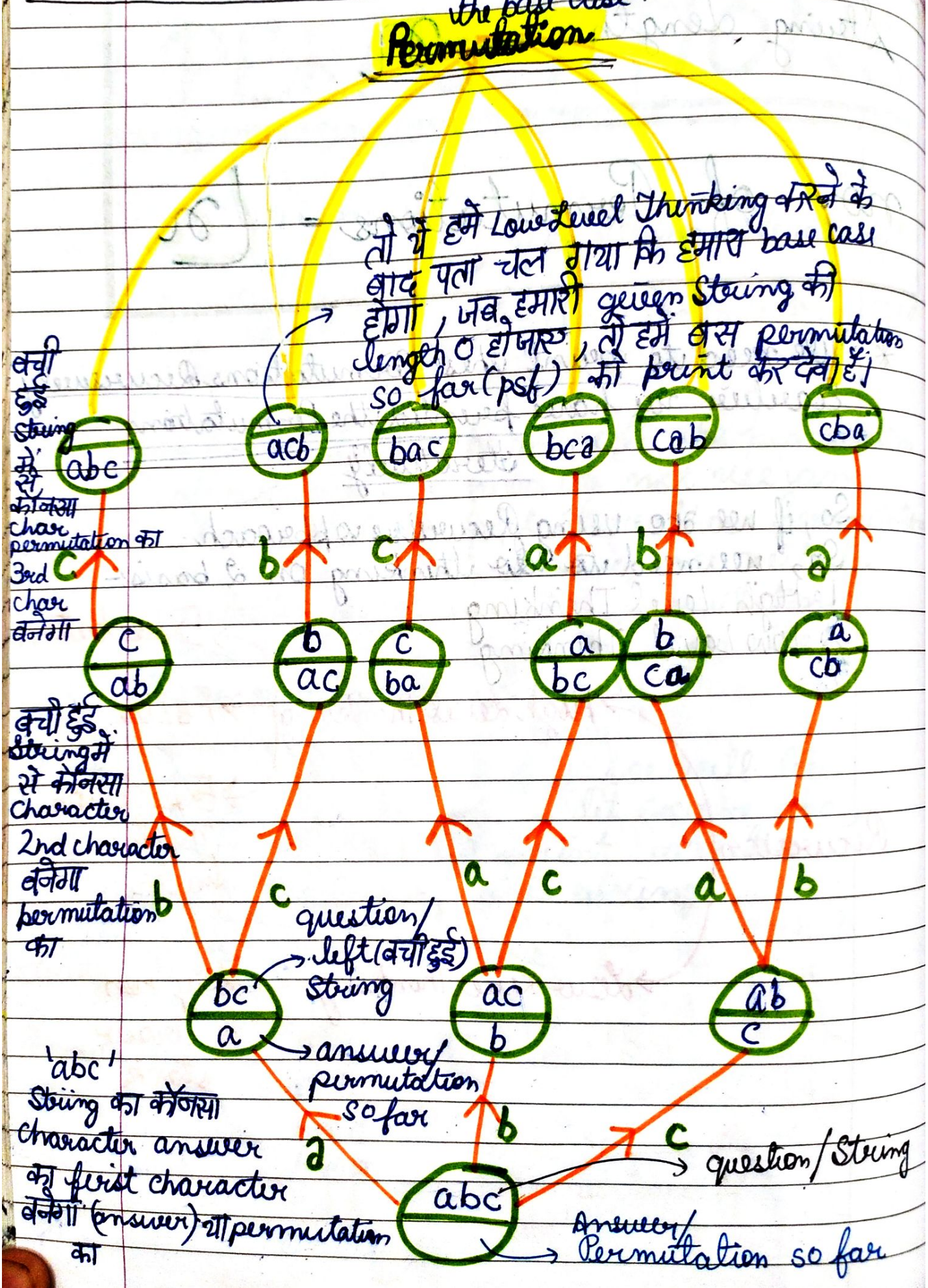
So, if we are using Recursive approach.
So, we need to do thinking on 2 basis:-
1. High Level Thinking
2. Low Level Thinking

Recursion
- High Level Thinking
  - Expectation
  - Faith
  - Expectation meets faith
- Low Level Thinking → Dry run & draw stack

abc → abc bac cab
acb bca cba

LOW LEVEL THINKING → Dry Run & draw Stack to find the base case

Permutation



HIGH LEVEL THINKING

1. **Expectation** → We expect that we will get the permutations of the given String 'abc'.
   ↳ abc, acb, bac, bca, cab, cba

2. **Faith** → हमे ये विश्वास रखना होगा कि अगर हमारा code हमे given String 'abc' की permutations provide कर सकता है तो हमे

String - asf = roq

- String → Given String
- asf → answer so far (so far मे string के char आएगे एक-एक बार सब की permutation मिलजाएगी, हम ये faith रखते है कि ये (ab, ac, ab) की सारी permutation हमे मिलजाएगी।
- roq → rest of question (remaining String) की provide करेगा।

bc	ac	ab
cb	ca	ba

3) **Expectations meets Faith** → हमे (bc, ac, ab) की permutation मिल जाएगी तो हमे a, b, c को accordingly asf मे add करते जाना है और जब String की length 0 हो जाए तब asf को print करना है।

a + permutations of bc	b + permutation of ac	c + perm. of ab
a + bc = abc	b + ac = bac	c + ab = cab
a + cb = acb	b + ca = bca	c + ba = cba

Print Permutations <code>

```
import java.util.*;
import java.io.*;
public class Main{
public class void main(String[] args)
  {
  Scanner scan = new Scanner(System.in);
  String str = scan.next();
  printPermutations(str, "");
  }
public static void printPermutations(String
        str, String asf)
{
                    answer so far
                    (permutation so far)
if(str.length()==0)
  {
  System.out.println(asf);
  return;
  }
for(int i=0; i<str.length(); i++)
  {
  char ch = str.charAt(i)
  String leftOfCh = str.substring(0, i);
  String rightOfCh=
        str.substring(i+1);
```

substring starts from 0 and will be left of ch

last index is not included in substrings

i+1 to last character will be the Right of Ch

Base Case

rest of question
rest of string (left of ch + right of ch)
character

→ String roq = leftOfCh + rightOfCh ;

This roq String will consist the left part of Ch & the right part of Ch which means it consist remaining part of String after extracting ch.

→ printPermutations (roq , ans + ch);

We have concatinated or addded the character ch to the string asf (answer/ permutation so far)

}
}
}

Ques 3

PRINT ENCODINGS

We need to print the encoding of a given string.

1 → a
2 → b
3 → c
4 → d
.
.
.
25 → y
26 → z

We need to print all of the possible strings which the given string can be encoded into.

eg → 123

↓

Given String can be encoded into following possible ways

"abc"
"aw"
"lc"

We need to solve this question Recursively, so we need to do 2 types of Thinking

- High Level Thinking
- Low Level Thinking

Recursion
- High Level Thinking → Faith → Expectation
  - → Expectation meets faith
- Low Level Thinking → Dry run and draw Stack

Expectation

हमें '123' की sab encoding print करनी है।

1 को हम उसके भी code करसकते है as 'a'
और 1 को 2 के साथ
12 बनाके भी code करसकते है as 'l'

से ही 2 को भी 2 → b
और
23 → w
बना सकते है।

हमे output में चाहिए
"abc"
"aw"
"lc"

Faith

हम यहाँ विश्वास रखते है कि अगर हमारा code हमें 123 के corresponding encodings दे सकता है तो a वो हमें 1 के लिए 'a' asf में add करने के बाद, 23 के लिए अगर Recursive call करेंगे तो उसके encodes भी provide करेगा।
b) और वो हमें 12 के लिए 'l' asf में add करने के बाद '3' के corresponding encodes करके provide करेगा।

```
a + {printEncodings(23);
l + {printEncodings(3);
```

Expectation meets faith

हमें (23) और (3) के encodes मिलजाएगे।
हमने जब call की थी तो (asf + a) और (asf + l) add करने के बाद ही की थी, This is our self work.

a + bc = abc
a + w = aw
l + c. = lc.

↳ These are the desired encoding that are to be printed.

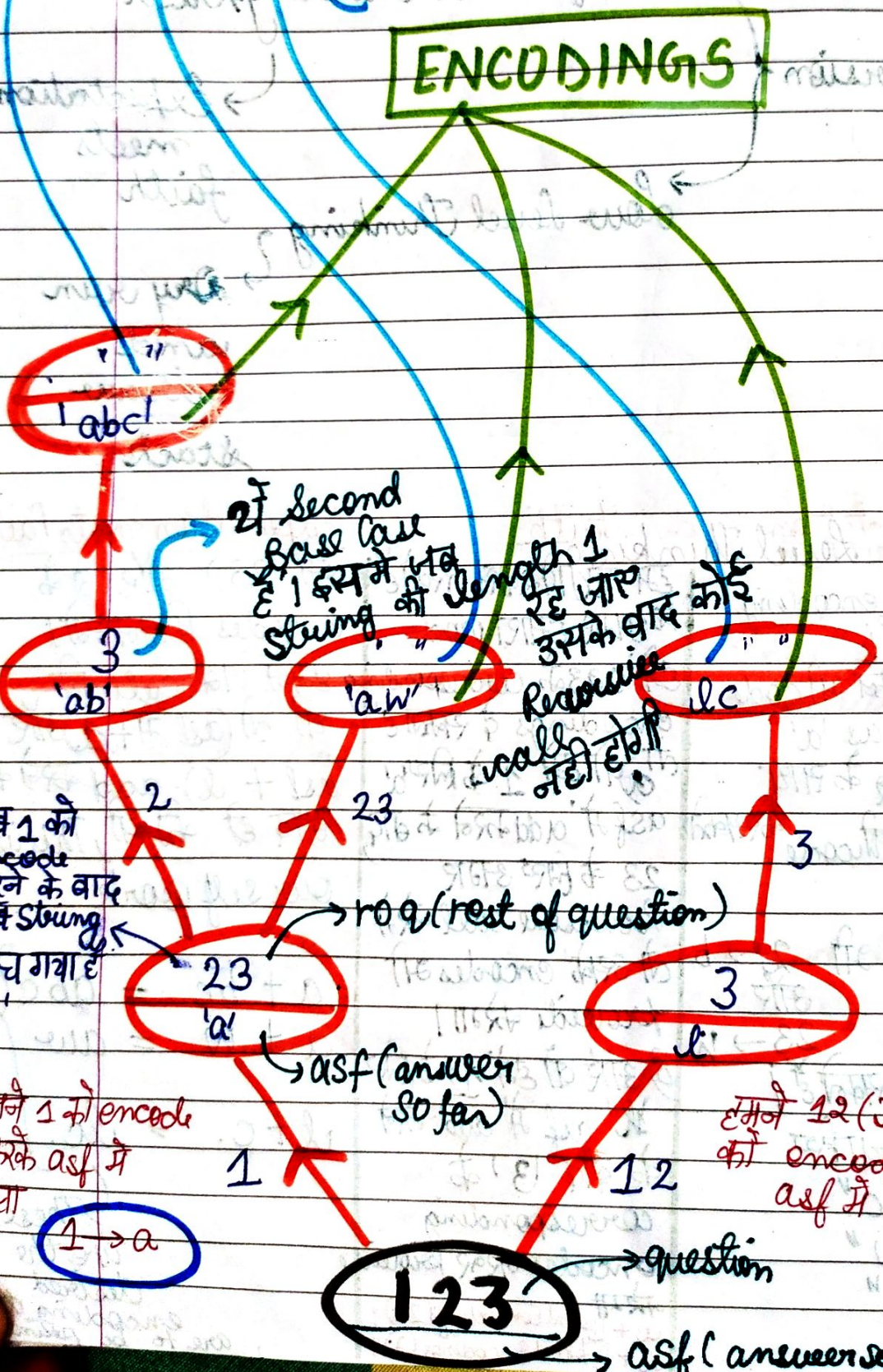
LOW LEVEL THINKING

→ Done to find the Base Case

Page No.
Date

Here we are using the "Level & options" Trick

So, This becomes our base case, when the string length becomes 0, then we just need to print the answer so far and return



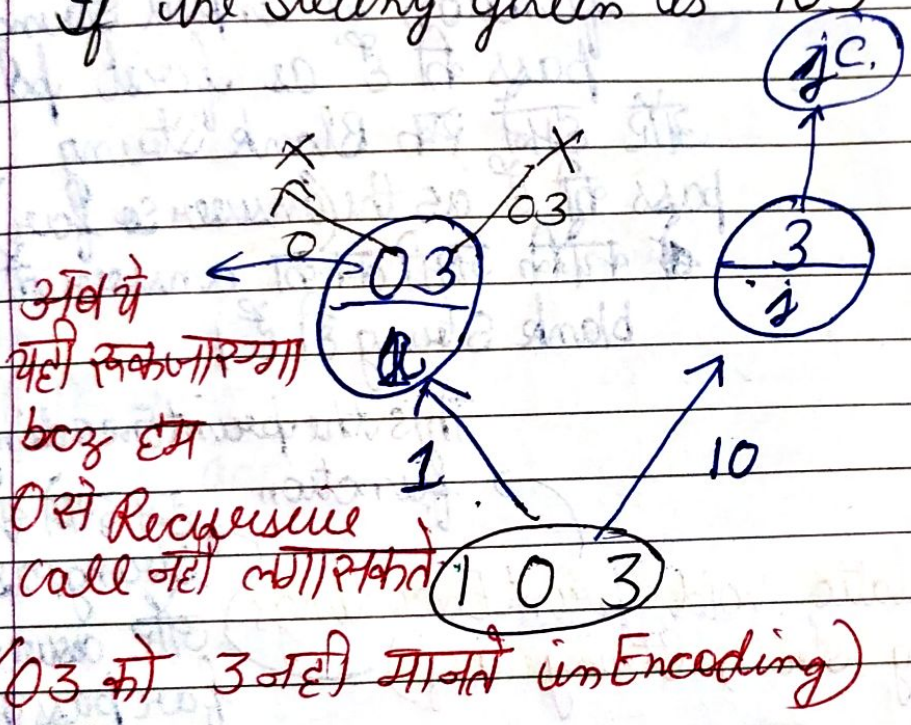
There is an exception in this case :-

eg 1 → If the String given is 013

↓

Here 0 doesn't denote anything
So this string can't be given as Input (where the String starts from 0)

eg 2) If the String given is 103

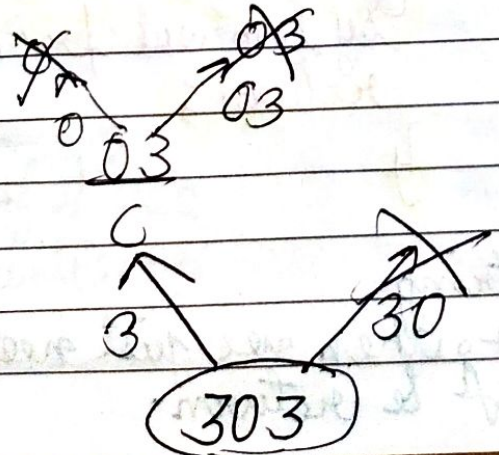


उसे ये नहीं करेंगे
bcz हम 0 से Recursive call नहीं लगा सकते

(03 को 3 नहीं मानते in Encoding)

eg 3) If the String given is 303.

So, No Encoding possible here



30 is greater than 26

PRINT ENCODINGS <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.next();
        printEncodings(str, "");
    }
    public static void printEncodings(String str, String asf)
    {
        if(str.length() == 0)
        {
            System.out.println(asf);
            return;
        }
    }
```

यहाँ String input ली

यहाँ printEncodings function में Input String pass की & as first parameter और यहाँ पर Blank String pass की & as the answer so far क्योंकि अभी तक का answer तो blank String ही है।

This the printEncoding function

इस इसमें Input String pass करो और answer so far pass करो।

यह first Base case है जहाँ String length 0 होती है

So, this is the Base case we have i.e When the length of String becomes Zero, then we just need to print the asf & return.

ये second Base Case है, जब string की length 1 रह जाए हमें उसके बाद कोई भी Recursive call नहीं करनी हमने बस जो 1 character बचा है उसको encode करके asf में add करना है और asf को print करा देना है और फिर return करना है।

```
else if (str.length()==1)
{
  char ch=str.charAt(0);
  if(ch=='0')
  {
    return;
  }
  else
  {
    int chv=ch-'0';
    char code=(char)('a'+ch-1);
    asf=asf+code;
    System.out.println(asf);
    return;
  }
}
```

अगर वो एक character 0 है तो हमें कुछ नहीं करना बस return करदो यहाँ से

लेकिन अगर वो char '0' नहीं है तो उस character का code निकालो कि alphabet कौन सा आएगा

eg → 97 2 1

97+2-1=98

character में typecast करना जरूरी है bcz char integer में add होने के बाद integer बन जाती है।

और फिर उस code (मतलब alphabet को) asf (answer so far) में add करदो, फिर उस asf को print करदो और then we return.

else { मतलब जब दोनों Base case वाली conditions true ना हो तब हम इस else में enter करेंगे। which means हमारी given string की length 1 से बड़ी है तो अब हमें 2 case handle करने होंगे यहाँ

1) जब हम पहला एक character को encode करके asf में add करेंगे और बची हुई string को (roq) को printEncodings function को recursively call करेंगे।

Case 1 → जब हम first character को encode करेंगे.

अब हमें ये भी check करना पड़ेगा कि कहीं उस String का first character '0' तो नहीं, और अगर first character '0' मिला, तो हमें बस return करना है। और अगर first character '0' नहीं हुआ, then we just need to encode that character to the corresponding alphabet and then add that alphabet to asf and do the recursive call for rest of the String/question

character
first character

```
char ch = str.charAt(0);
String roq = str.substring(1);
if(ch=='0')
{
    return;
}
else
{
    int chv = ch - '0';
    char code = (char)('a' + chv - 1);
    printEncodings(roq, asf + code);
}
```

rest of the question

return करदो बस

character की value integer में convert करने के लिए

यहाँ से हमें character के corresponding जो alphabet है वो मिल जाएगा।

Case 2 → जब हम first 2 characters को एक साथ encode करेंगे

1) हमें सबसे पहले ये देखना होगा कि दो 2 characters (eg → '24') → 26 से बड़े नहीं होने चाहिए bcz 1 to 26 corresponds to a-z

2) फिर हमें ये check करना होगा कि उन दो character में से first character 0 के equal है तो हम बस return करना है।

3) और अगर first character 0 नहीं है तो इस character String जिसमे पहले 2 character है (eg → '24') इसको पहले Integer में convert करो Integer.parseInt use करके, और फिर इस Integer (24) के corresponding alphabet (x) को asf में add करो, फिर rest of the question (or eg) के लिए printEncodings को recursively call करदो।

```
if (str.charAt(0) != '0')
{
    String first2Ch =
        str.substring(0,2);
    String rem12 = str.substring
        (2);
    int ch12 = Integer.parseInt(first2Ch);
```

→ हम check कर रहे है अगर String के first 2 char 03 जैसे हुए तो हमे 03 को हम 3 नहीं मानेंगे। तो हम कुछ भी नहीं करना ऐसी condition में तो हम if block में enter ही नहीं करेंगे।

→ ये first 2 characters में first character 0 नहीं होगा

→ इसमे rest of the String आगई है।

→ converting the String (eg → '12' to 12 int) → character 12 value

```
if (ch12v <= 26)
{
                    → code/alphabets
char encodedFirst2 = (char) ('a' + ch12v - 1);
printEncodings( ... 12, asf + encodedFirst2);
```

} → for current if condition

} → for last if condition

} → for else block

} → for ~~function~~ printEncodings

} → for classMain