DAY 49

Recursion with Arraylist

1. Date → 25 October 2021.
2. Day → Monday

3. Agenda →
1. Get Subsequence = CW
2. Get KPC = CW
3. Get stairs Path = CW.
4. Get Maze Path
5. First Index
6. Last Index
7. All Indices of array

Done in day 46 HW

Ques-1 Get Subsequence

→ हमें एक String मिलेगी as input, हमें उस string के subsequences return करने हैं as output in a arraylist

→ हमने पहले question strings वाले Module में किया है लेकिन iteratively किया है;

→ हमें अब ये question Recursively करना है solve. तो पहले जानते हैं What is Subsequence?

A subsequence is a part derived from a given string which is obtained by deleting some or no elements of the given string without changing the order of the remaining elements.

For a String of n length, we have 2^n subsequences

eg → String = abc

Subsequences of String ⇒

Step 1) → Take a blank String

_ _ _

अब इसे भी लो (इस String को) और एक इसकी copy बनाओ जिसमें c को उसकी correction position पर रख दो

Step 2)

_ _ _ → Put last character at correct position

_ _ c → copied same & put c (last element) at its pos".

इसे लो और इसकी copy बनाओ एक (इन दोनों Strings को) उस copied में b को उसकी correct Position पर रखो

Step 3)

_ _ _ → Copy it

_ _ c → Put last second character at correct position

_ b _ → copied same as above & put b at its correct position

_ b c → Put last second character

Take these string & Copy all these string and put a at correct psn

Step 4

_ _ _
_ _ c
_ b _
_ b c.
a _ _
a _ c
a b _
a b c.

Copied as Previous Strings and have put a (last 3rd) character at its correct position

→ These are the Subsequences of abc, because.

Character in abc = 3

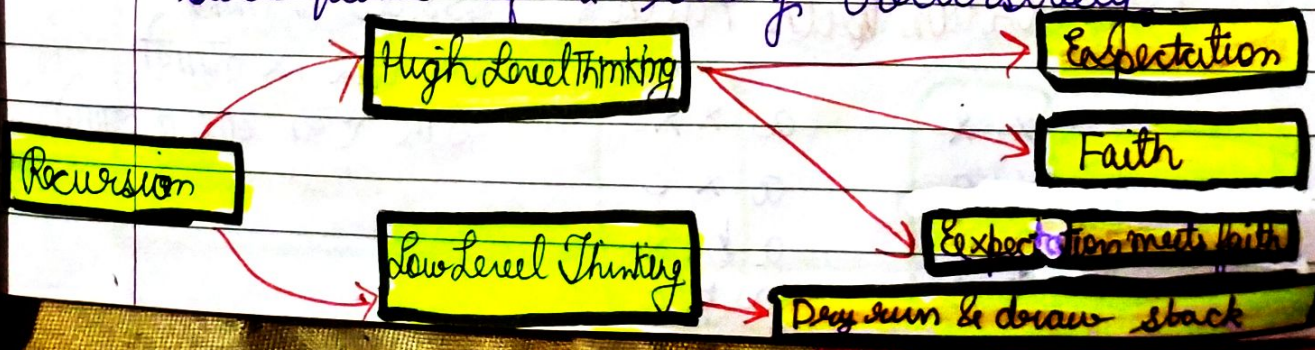
Subsequences of abc = $(2)^n = (2)^3 = 8$

& these are total 8 Subsequences.

Hence, we are done with finding all the subsequences of abc

* अब हमने ये समझलिया है कि subsequence क्या होते हैं।

Now we need to code to find all the subsequence of a String recursively



1. High Level Thinking

a) Expectation

gss(abc) = x x x
x x c
x b x
x b c
a x x
a x c
a b x
a b c

get subsequence.

Expectation में हम gss(abc) function देंगे वो 8 subsequence देगा।

x x x
x x c
x b x
x b c

a x x
a x c
a b x
a b c

b) Faith → faith भी Expectation का ही छोटा part होता है।

gss(bc) →
x x
x c
b x
b c

Same.

faith में हम output तो अपने आप ही मिलेगा अगर हमने bc pass किया तो।

c) Expectation meets Faith → gss(bc) के सामने एक बार x लगादो और एक बार a लगादो

x x x
x x c
x b x
x b c

a x x
a x c
a b x
a b c

हमने High Level Thinking करली है; so now, we can write the basic code.

Get 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();
  ArrayList<String> Subsequence = gss(str);
  System.out.println(Subsequence);
}
```

(gss → get subsequence)

Expectation // gss(abc) = [..., ..c, .b., .bc] + [a.., a.c, ab., abc]

Faith // gss(bc) = [.., .c, b., bc]

Expectation meets faith // = . + gss(bc) + a + gss(bc)

```
public static ArrayList<String> gss(String str)
{
  if (str.length() == 0)
  {
    ArrayList<String> bres = new ArrayList<>();
    String theSubsequence = "";
    bres.add(theSubsequence);
    return bres;
  }
  char ch = str.charAt(0);
  String ros = str.substring(1);
  ArrayList<String> rres = gss(ros);
  ArrayList<String> mres = new ArrayList<>();
  for (String rstr: rres)
  { mres.add("_" + rstr); }
  for (String rstr: rres)
  { mres.add(ch + rstr); }
  return mres;
}
```

Notes:
- gss → get subsequence
- bres → base result
- "" → Blank होता है subsequence
- This is the base case & it is explained in Low Level Thinking
- ch → first character of given String
- ros → rest of the string
- rres → recursion का result
- mres → my result
- for → ये foreach loop होता है: for each element of rres (recursion का result)

2. Low Level Thinking

↳ Dry run & draw stack.

Page No.
Date

```
Line 1 → char ch = str.charAt(0);          // abc → a
Line 2 → String ros = str.substring(1);    // abc → bc
Line 3 → ArrayList<String> rres = gss(ros); // gss(bc)
Line 4 → ArrayList<String> mres = new ArrayList<>();

Line 5 → for (String rstr : rres)
         {
           mres.add("" + rstr);
         }

// mstr = [--, -c, -b-, -bc]

Line 6 → for (String rstr : rres)
         {
           mres.add(ch + rstr);
         }

// mstr = [--, -c, -b-, -bc]
//      + [a--, a-c, ab-, abc]

Line 7 → return mstr;
```

String की 0th character

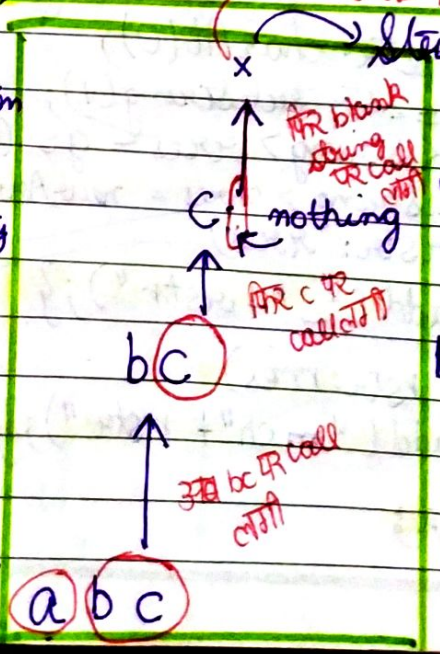
ros → rest of the String (0th character को हटाकर)

→ This is the Base Case.

String length is 0, अगर String में एक भी character न हो तो 1 subsequence होता है उस String का because No. of Subs = $(2)^{\text{No. of char}}$

So, no. of character = 0

No. of subsequence = 2^0 = 1



→ $x^0 = 1$ (Multiplication का identity 1 होता है)

→ $x^2 = x * x$ (Sum का identity 0 होता है)

→ $x^1 = x$

→ In how many ways can we select 0 apple out of 5 apple?

↳ 0 apple can be selected in 1 way i.e. Don't select any apple

1 2 3 right side call

अगर कोई String $x = [\,"\,"\,]$, मतलब blank string with 0 characters String है x, तो उसके subsequence होंगे 1

Page No. | Date

bcz no. of character = 0.

no. of subsequence $= (2)^{\text{no. of character}}$

$= (2)^0$

$= 1$

Time Complexity of Get Subsequence $O(n) \rightarrow$ Linear bcz a recursion call is made along with using for loop.
Space Complexity of Get Subsequence $\rightarrow O(1) \rightarrow$ No extra space required. ∴ constant.

अब इससे ये तो पता चलगया कि 1 Subsequence होगा लेकिन उस Subsequence में value क्या होगी?

→ [ ]

→ कोई भी value नहीं होगी

→ Blank Subsequence

तो Base Case Code में कैसे लिखेंगे?

```
if (str.length()==0)
{
    ArrayList<String> bres = new ArrayList<>();
    bres.add("");
    return bres;
}
```

base result

बिलकुल blank subsequence add करो arraylist में, even without a space

[""]

→ Arraylist का size 1 है अब लेकिन इस Arraylist में जो string है उसका size 0 है।

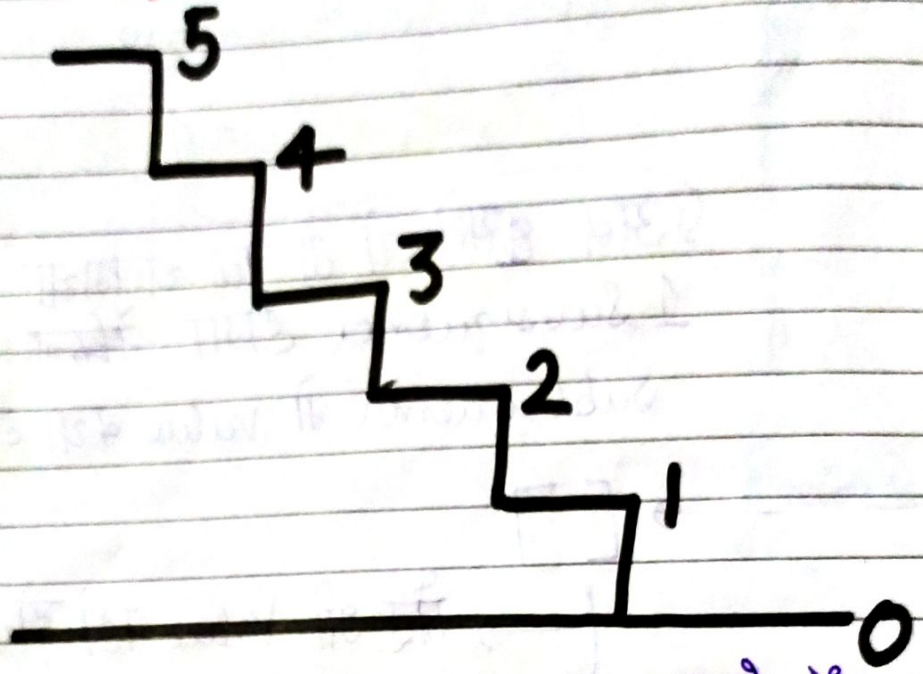
bres.get(0).length() = 0

Quiz - 2)

Get Stair Paths

Page No.
Date

↳ हमें एक Staircase (सीढ़ी) given है।

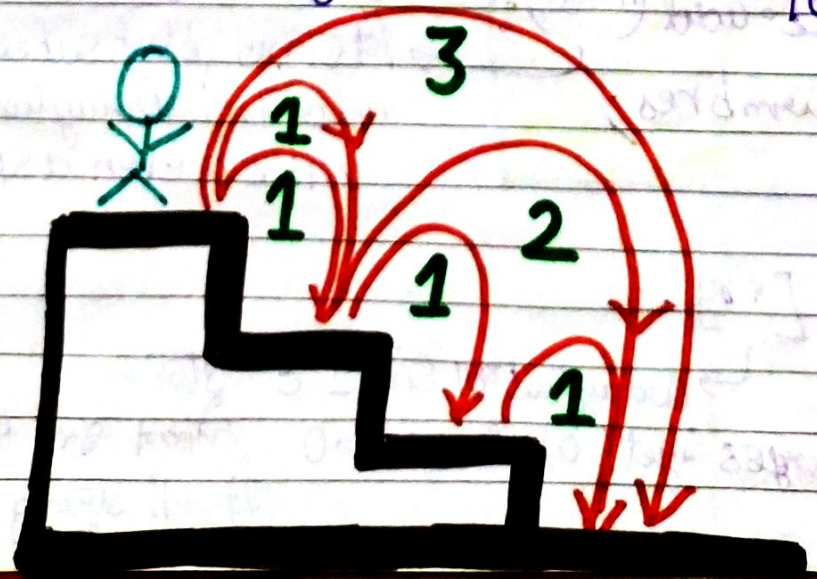


हम एक time पर एक कदम रख सकते हैं।
या दो कदम रख सकते हैं।
या तीन कदम रख सकते हैं।

एक ही बात है।

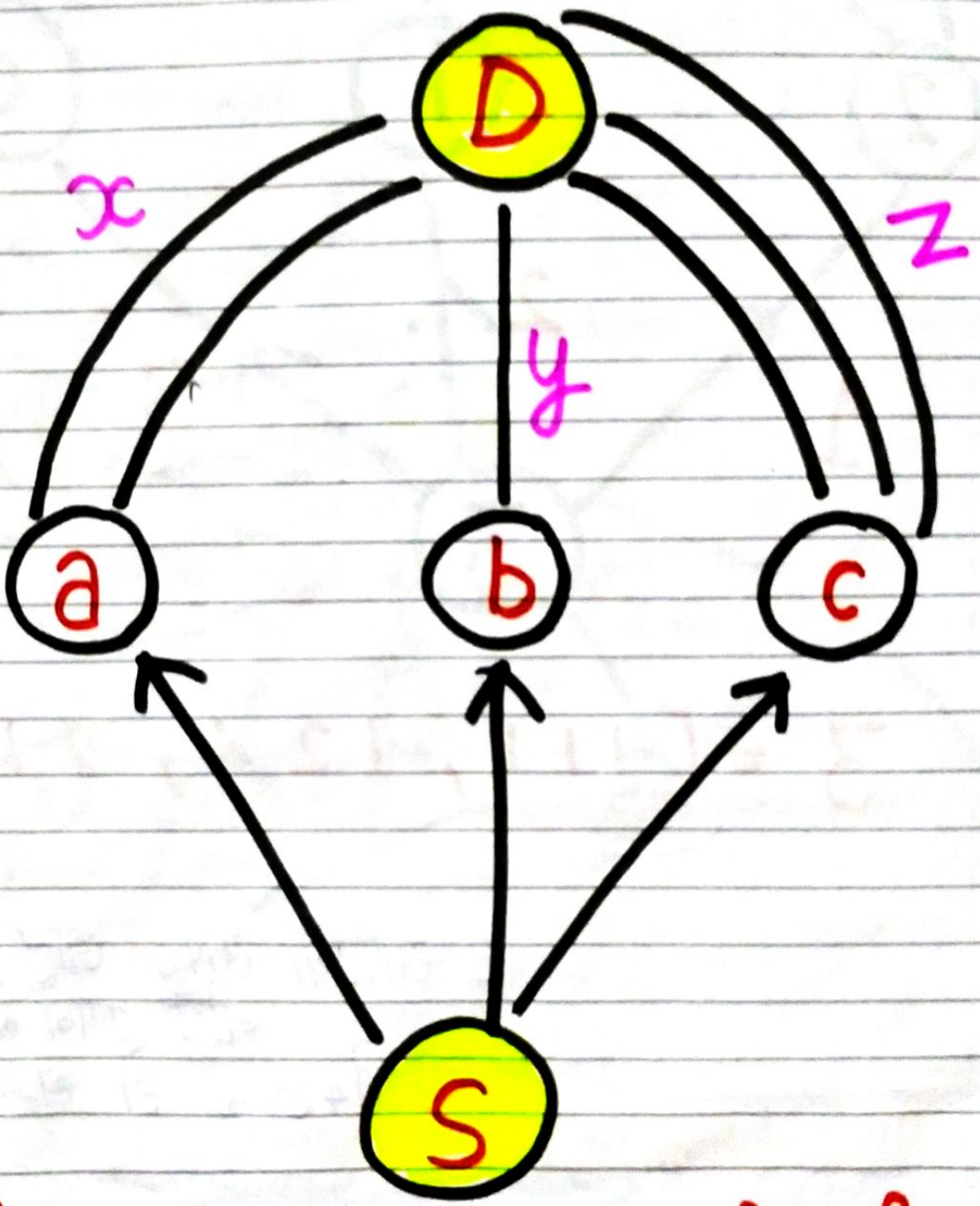
हमें 0 से 5 तक जाने के सारे तरीके बताने हैं।
0 से 5 तक पहुँचने के 16 तरीके हो सकते हैं।

या फिर 5 से 0 तक आने के सारे तरीके बताने हैं,
5 से 0 तक आने के भी 16 तरीके हो सकते हैं।

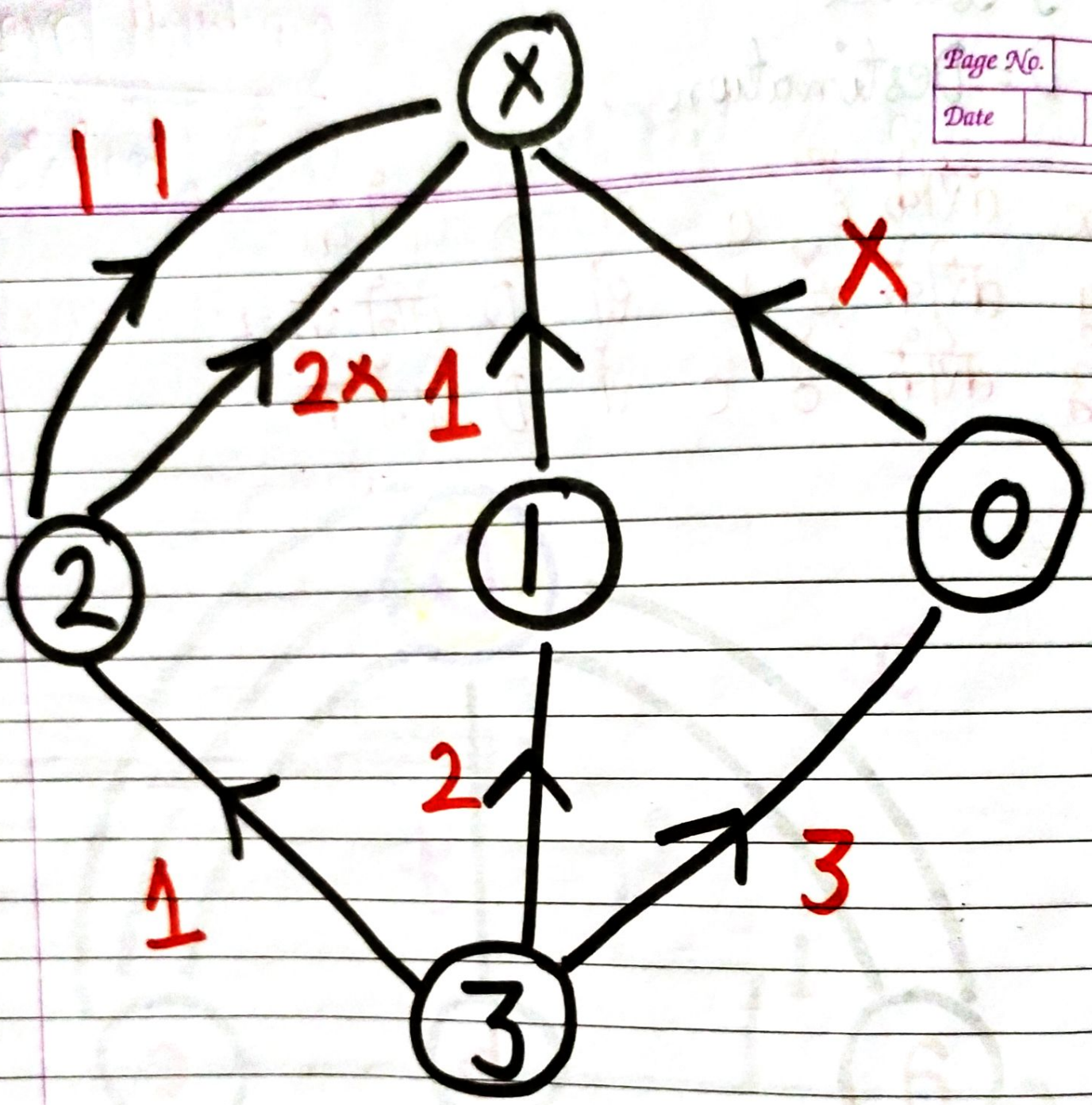


S = Source
D = Destination

x तरीके हैं a से D जाने के
y तरीके हैं b से D जाने के
z तरीके हैं c से D जाने के

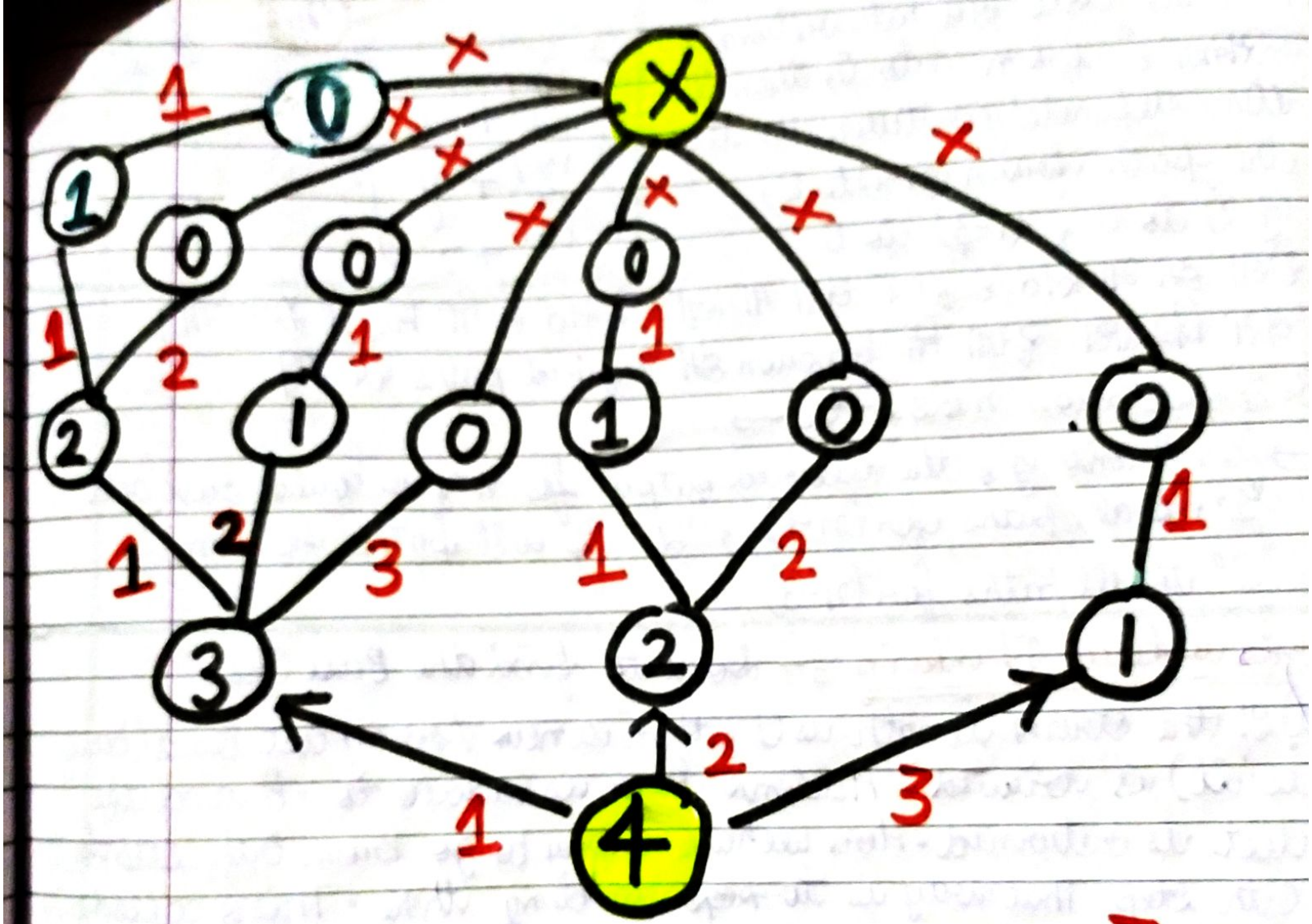


तो S से D तक जाने के
x + y + z तरीके होंगे।



$3 = [111, 12X, 21, 3X]$

3 सीढ़ियाँ ऊपर जाने के/
नीचे आने के
पे 4 Steps हो सकते हैं।



4 = [1111, 112, 121, 13, 211, 22, 31]

4 सीढ़ियाँ ऊपर जाने के / नीचे आने के थे 7 तरीके हो सकते हैं।

We need to solve this Problem Recursively

Page No.
Date

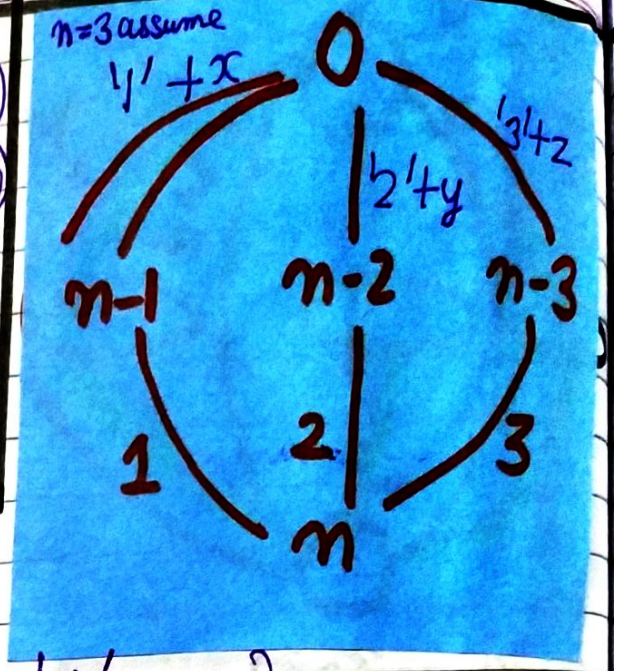
Recursion

- High Level Thinking
  - Expectation
  - Faith
  - Expectation meets faith
- Low Level Thinking
  - Dry run & draw Stack

1. High Level Thinking :-

Expectation	Faith
paths(n to 0).	F_1 = paths (n-1 to 0) F_2 = paths (n-2 to 0) F_3 = paths (n-3 to 0)

Expectation meets faith



1. Set Expectation → We expect that we will get all the paths from n to 0.
2. Build Faith → हम faith रखते हैं कि हमारा code सब paths return करसकता है from n to 0, then it can definetely return us all the path from (n-1) to 0, (n-2) to 0, (n-3) to 0.

$$\begin{Bmatrix} '1'+x \\ '2'+y. \\ '3'+z \end{Bmatrix}$$

ऐसा हम मानलेते हैं, ये बस मानलो, ये मत सोचो कि, ये कैसे होगा। बस विश्वास रखो कि function हमें required paths देदेगा।

3. Expectation meds faith →
→ for printing the desired output for n, we could just add "1" to all paths for (n-1), "2" to all paths for (n-2), "3" to all paths for (n-3).

2. Low Level Thinking :- Done to find the Base Case

a) If the string length is 0, then a new Arraylist here (base result) is created. A blank path is added to it and the list is returned. Here we have 1 way to go from 0th step to 0th step. That way is to keep standing there. This is called the Positive Base Case.

b) When n<0, it means that there are no stairs left and the person will be going into the basement which isn't practically possible. Here, we don't add any path to the Arraylist. This is called the Negative Base Case.

Time Complexity :- $O(3^n)$ ← This time complexity is exponential because for each state, 3 recursion calls are made.

Space Complexity : $O(1)$

As no extra space is required. ∴ Space complexity is constant. If we include the space in recursive stack, then space complexity is $O(n)$.

Get Stair Path <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();
        ArrayList<String> paths = getStairPaths(n);
        System.out.println(paths);
    }

    public static ArrayList<String> getStairPath(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> pathfromnm1 = getStairPath(n-1);
        ArrayList<String> pathfromnm2 = getStairPath(n-2);
        ArrayList<String> pathfromnm3 = getStairPath(n-3);

        ArrayList<String> pathfromn = new ArrayList<>();
    }
```

n is no. of Stairs.

function name

return type is an arraylist of type String

base result

element is there but no value

वे होगा return

base result

no element.

वे होगा return

Hold path from n minus 1

These are Recursive calls

2 base case होगे इस case में

+ve Base Case

-ve Base Case

~~Hatch~~ for each ... String
in the ArrayList pathsfromnm1

```
for (String pathfromnm1 : pathsfromnm1)
{
    String pathfromn = 1 + pathfromnm1;
    pathsfromn.add(pathfromn);
}
for (String pathfromnm2 : pathsfromnm2)
{
    String pathfromn = 2 + pathfromnm2;
    pathsfromn.add(pathfromn);
}
for (String pathfromnm3 : pathsfromnm3)
{
    String pathfromn = 3 + pathfromnm2;
    pathsfromn.add(pathfromn);
}
return pathsfromn;
}
```

- TAMANNA VERMA