Normal Recursion →

* Bottom up Approach → First occurrence first
* Top down Approach → Last ocerence
* calls flow → All Indices { use both Approach to solve this problem
* Area in Recursive code
* tower of Hanoi { Different type of merging faith with Expectation

Problem with array: →

1. size of array is freezed.
2. Addition & Removal is difficult in array.

How to avoid use of Array → String → Collection of character

→ Prefer arraylist

demo of arraylist

- → add
- → remove (index)
- → size
- → Is Empty
- → get
- → set
- * → add At

motive of demo

- creation
- print → syso, character wise (loop, for each)
- addition in string, contenation '+'] memory
- get → character wise
- size → .size ✗
- substring

demo → code

size: Java → str.length(), str.size ✗, arr.length (data member); C++ → str.size() ✓, str.length() ✓

1. str.substring (Beginning Index, Ending Index)
   - → B. Index is Include in Result
   - → E. Index is Exclue. 0 ≤ E ≤ arr.length
   - 0 ≤ B. Index ≤ arr.length
2. str.substring (Beginning Index)
   - → It give the substring from beg. to End.

Get Subsequence →

0 to $2^n - 1$

	Binary (a b c)	Req. output (2 bits)	String → a b c (normal mapping subseq.)
0 →	0 0 0	—	—
1 →	0 0 1	a	c
2 →	0 1 0	b	b
3 →	0 1 1	ab	bc
4 →	1 0 0	c	a
5 →	1 0 1	ac	ac
6 →	1 1 0	bc	ab
7 →	1 1 1	abc	abc

String.length() = n

no. of subseq → 2^n

String → a b c

Expectation → return all subseq in AL

bc → subseq

—	—
b	—
—	c
b	c

abc

a	— —
a	b —
a	— c
a	b c

—	— —
—	b —
—	— c
—	b c

abc → find subseq of res(bc)

subseq of abc → — + res(bc)

at [res(bc)]

Faith. *find subseq of Rest of string {except 0th one}

→ res → { } → subseq bc

Merging → subseq(abc) →

faith

sub(bc)

append (—)

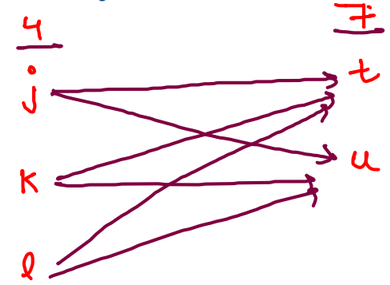
append (a)

subseq (abc)

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

keypad phone

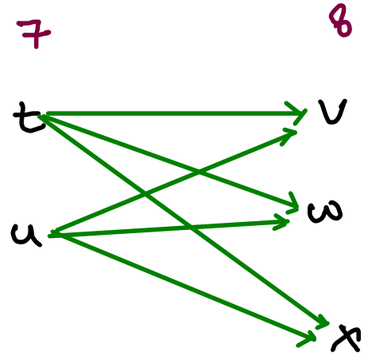
msg type → 47



possible message

jt
ju
kt
ku
lt
lu

78



possible message

tv
tw
tx
uv
uw
ux

Sample Output

[tv, tw, tx, uv, uw, ux]

Expectation, → getkpc(78)

length=2

getkpc(8) → [v, w, x]

t

faith

Merging → 7 → t

want to use as number availability → char-

numeric

0 → 9

"4 9 8"

'0' → 48

→ 4 as number ⇒ Binary store in comp.

→ 'a' as character ⇒ Binary ?? Every character (97) ASCII code

'8' - '0' → 8

'8' - 48 = 56

Arithmetic operation work on num

'4' → as character deal → 4 as number

→ as a number → (52)

'0' → 48

'4' - '0' = 4

52 - 48

char ch = charAt(0) = '4'

nums → 4 → [tv, tw, tx, uv, uw, ux]

return nums.

487

	ASCII
0 →	48
1 →	49
2 →	50
3 →	51
4 →	52
:	:
9	

Steps:

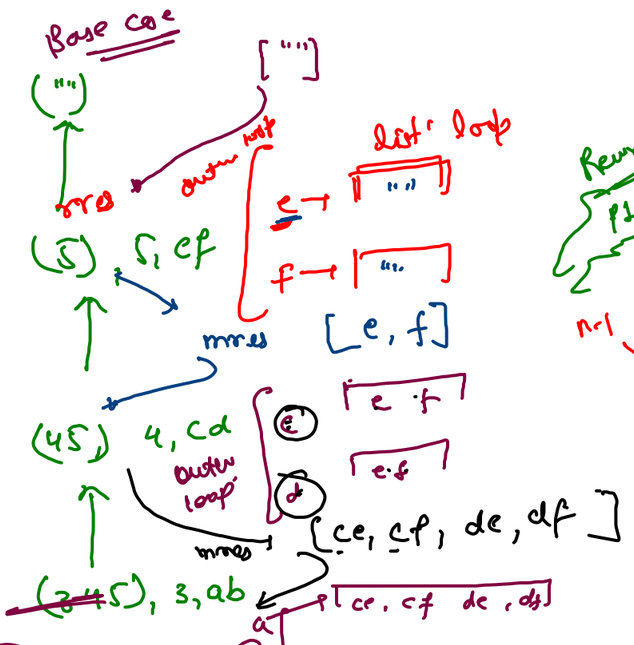
1. code 9ndx → 4
2. Rest of number → 87
3. code of code 9ndx → map 9 (4)
4. get Recursive answer
5. prepare my answer

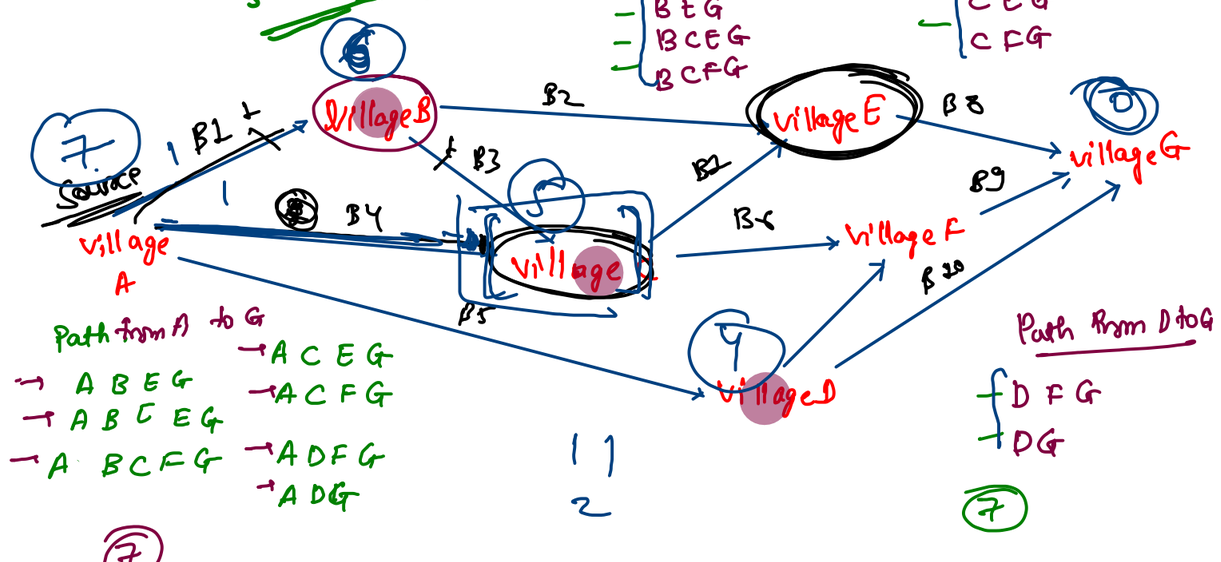
```java
public static ArrayList<String> getKPC(String num) {
    if(num.length() == 0) {
        ArrayList<String> bres = new ArrayList<>();
        bres.add("");
        return bres;
    }

    int codeIndx = num.charAt(0) - '0';
    String ron = num.substring(1);
    String code = codes[codeIndx];

    ArrayList<String> rres = getKPC(ron);
    ArrayList<String> mres = new ArrayList<>();
    for(int i = 0; i < code.length(); i++) {
        char ch = code.charAt(i);
        for(String str : rres) {
            mres.add(ch + str);
        }
    }
    return mres;
}
```

3 → a b

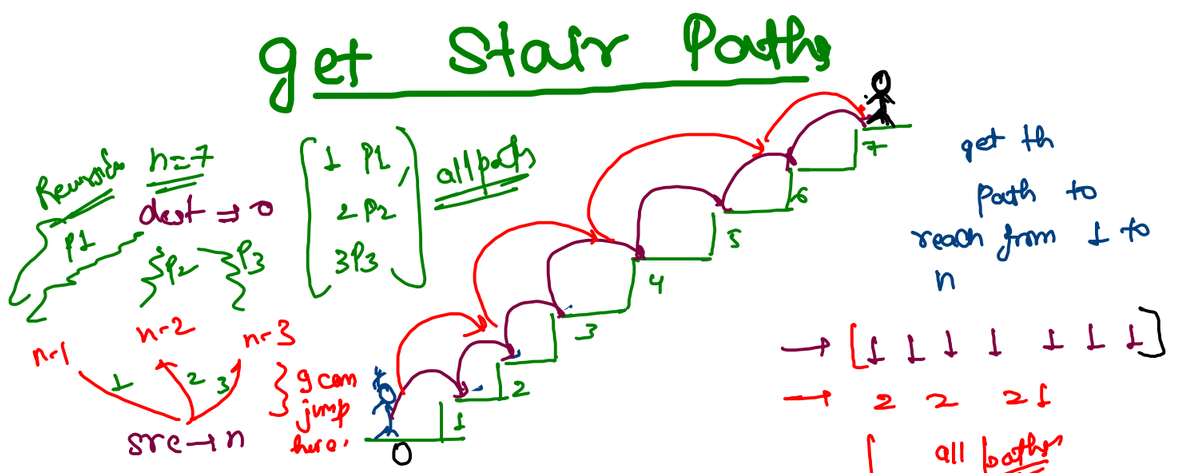
4 → c d

5 → e f





get Stair Paths



get the path to reach from 1 to n

→ [1 1 1 1 1 1 1]

→ 2 2 2 1

all path

return an arraylist

Expectation →

gsp(n) → return all path to to reach at 0 from 'n'

Faith →

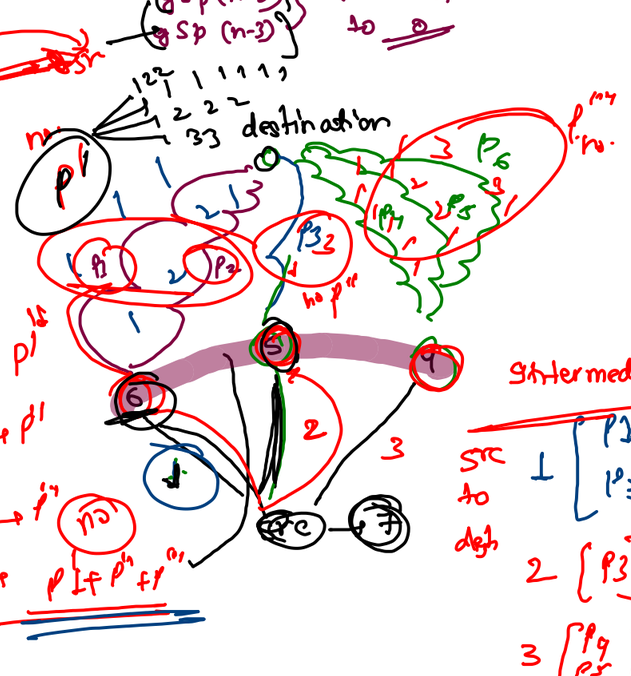
gsp(n-1)
gsp(n-2)
gsp(n-3)
} Return path from their respective start to 0

12 1 1 1 1
2 2 2
33

jump (3) path

}

Total path from (7)



Intermediate

6 → P1 + P2 → 2 path. = p^{12}

5 → P3 → 1 path → p^{1}

4 → P4 + P5 + P6 → 3 path → p^{13}

Total → 6 path → $p^{12} + p^{13} + p^{11}$

Intermediate

src to 4th

1 [P1 P2]

2 [P3]

3 [P4 P5 P6]

```java
public static ArrayList<String> getStairPaths(int n) {
    ...
    ArrayList<String> nm1Path = getStairPaths(n - 1);
    ArrayList<String> nm2Path = getStairPaths(n - 2);
    ArrayList<String> nm3Path = getStairPaths(n - 3);

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

    // make path for jump = 1
    for(String path : nm1Path) {
        mres.add("1" + path);
    }
    // make path for jump = 2
    for(String path : nm2Path) {
        mres.add("2" + path);
    }
    // make path for jump = 3
    for(String path : nm2Path) {
        mres.add("3" + path);
    }

    return mres;
```

Jumps which bring me at Basement are useless for me

Basement

No need to return path

4 to 0 → 4 paths.

7 to 4

```java
public static ArrayList<String> getStairPaths(int n) {
    ...
    ArrayList<String> nm1Path = getStairPaths(n - 1);
    ArrayList<String> nm2Path = getStairPaths(n - 2);
    ArrayList<String> nm3Path = getStairPaths(n - 3);

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

    // make path for jump = 1
    for(String path : nm1Path) {
        mres.add("1" + path);
    }
    // make path for jump = 2
    for(String path : nm2Path) {
        mres.add("2" + path);
    }
    // make path for jump = 3
    for(String path : nm3Path) {
        mres.add("3" + path);
    }

    return mres;
```

1 1 1 1
1 1 2
1 2 1
2 1 1
2 2
3 1

dist

Intermediate

Paths

7 src.

111