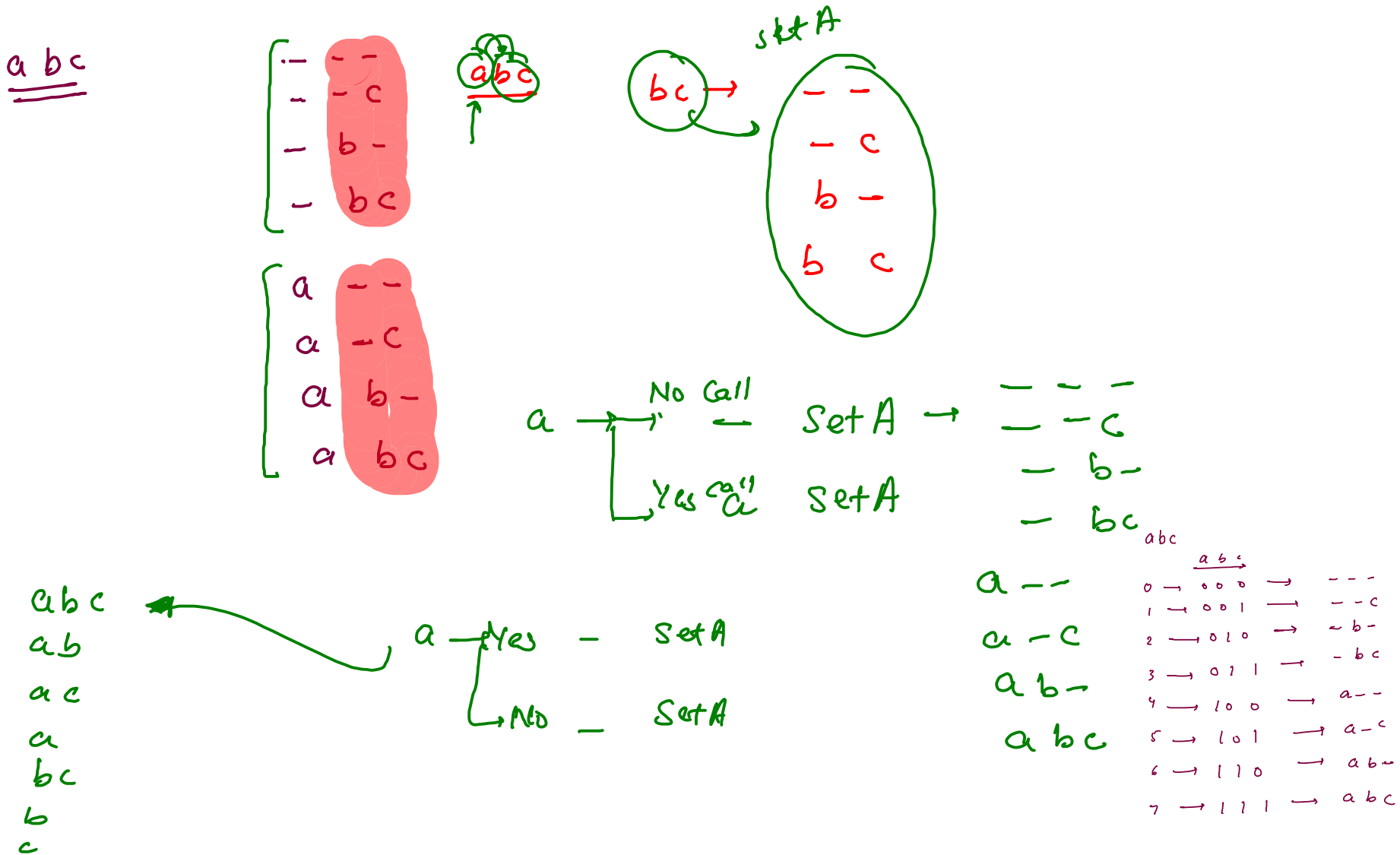

a b c
- -
- c
- b -
- b c
a b c
a - -
a - c
a b -
a b c
b c
skt A
- -
- c
b -
b c
a
No Call
Set A
Yes "a"
Set A
- - -
- - c
- b -
- b c
abc
a b c
0 → 0 0 0 → - - -
1 → 0 0 1 → - - c
2 → 0 1 0 → - b -
3 → 0 1 1 → - b c
4 → 1 0 0 → a - -
5 → 1 0 1 → a - c
6 → 1 1 0 → a b -
7 → 1 1 1 → a b c
a - -
a - c
a b -
a b c
a
Yes
-
Set A
No
-
Set A
a b c
a b
a c
a
b c
b
c

Subsequence

Expectation → getsubseq (abc) → { - , c, b, bc, a, ac, ab, abc }

faith → getsubseq (bc) → { - , c, b, bc } } Recursion

Merging of faith and Expectation →

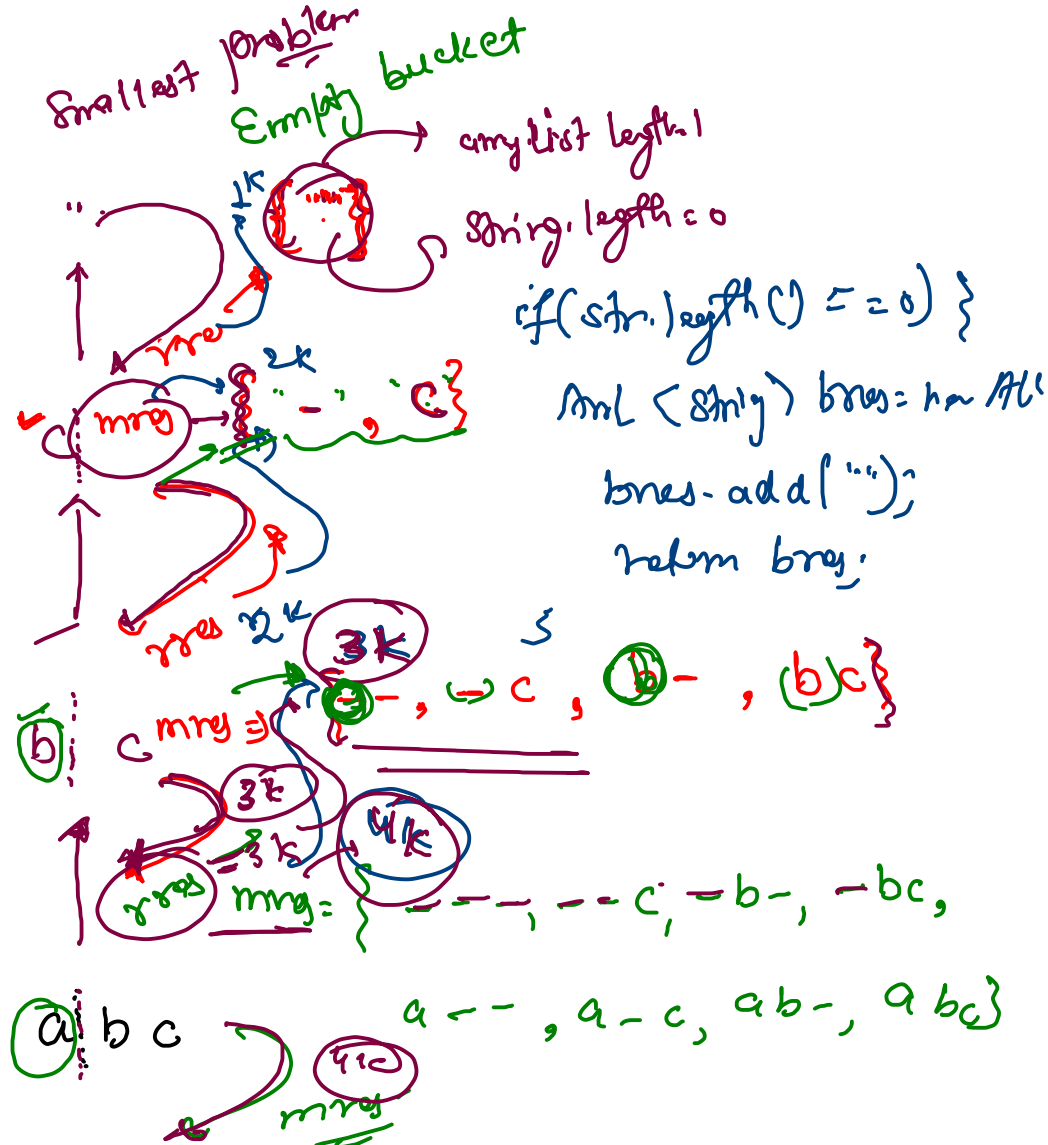
```
getsubseq ( abc ) →  char ch = a.
                     rres = getsubseq(bc) {-, c, b, bc}
                     mres → ( - rres) {-, c, b, bc
                            (ch rres)  a, ac, ab, abc}
                     return mres;
```

→ No bucket is there

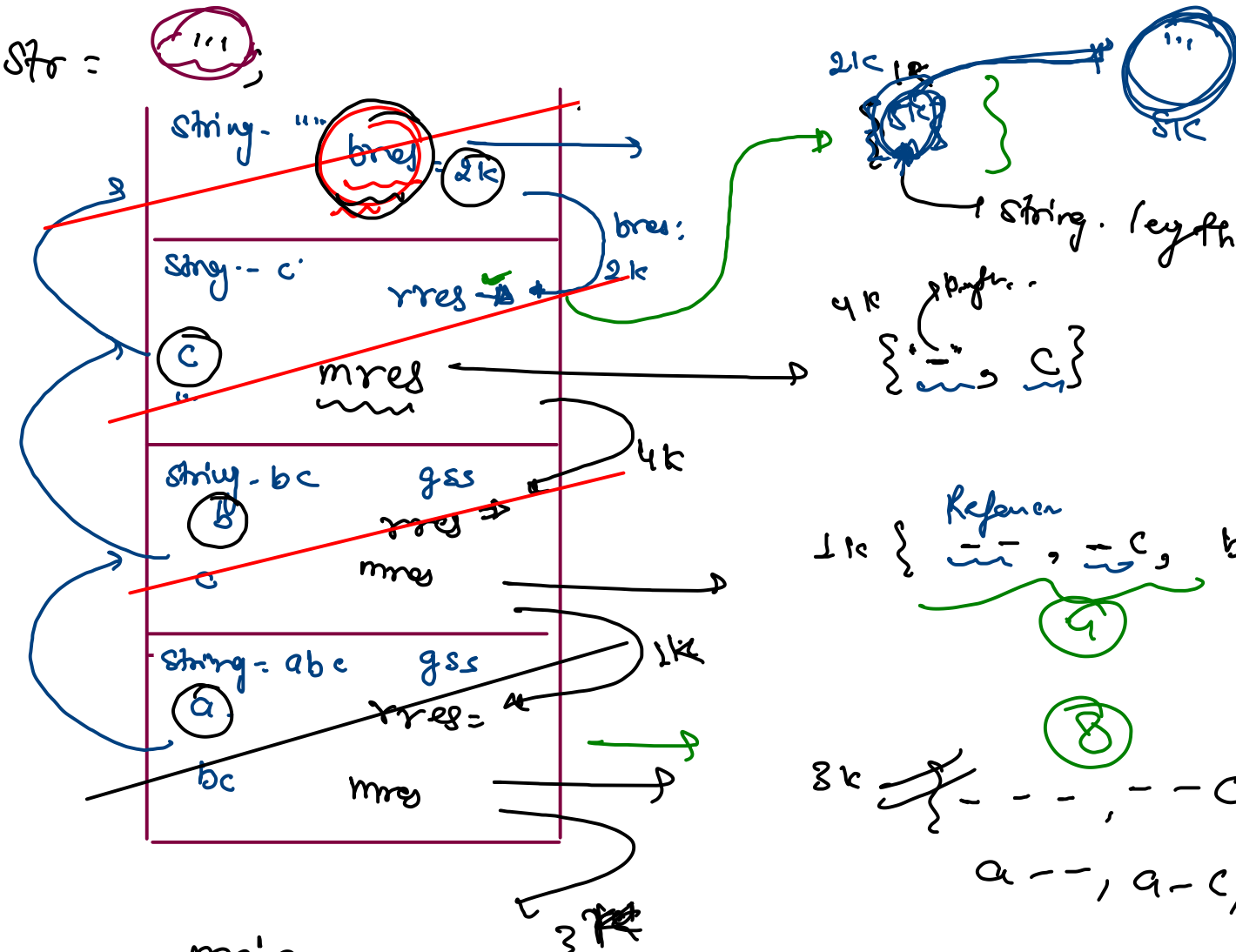
→ Empty bucket is Here

```java
public static ArrayList<String> getSubSeq(String str) {
    char ch = str.charAt(0);
    String roq = str.substring(1); // roq => rest of question

    ArrayList<String> rres = getSubSeq(roq);
    ArrayList<String> mres = new ArrayList<>();
    // add with -
    for(String s : rres) {
        mres.add("-" + s);
    }
    // add with char
    for(String s : rres) {
        mres.add(ch + s);
    }

    return mres;
}
```

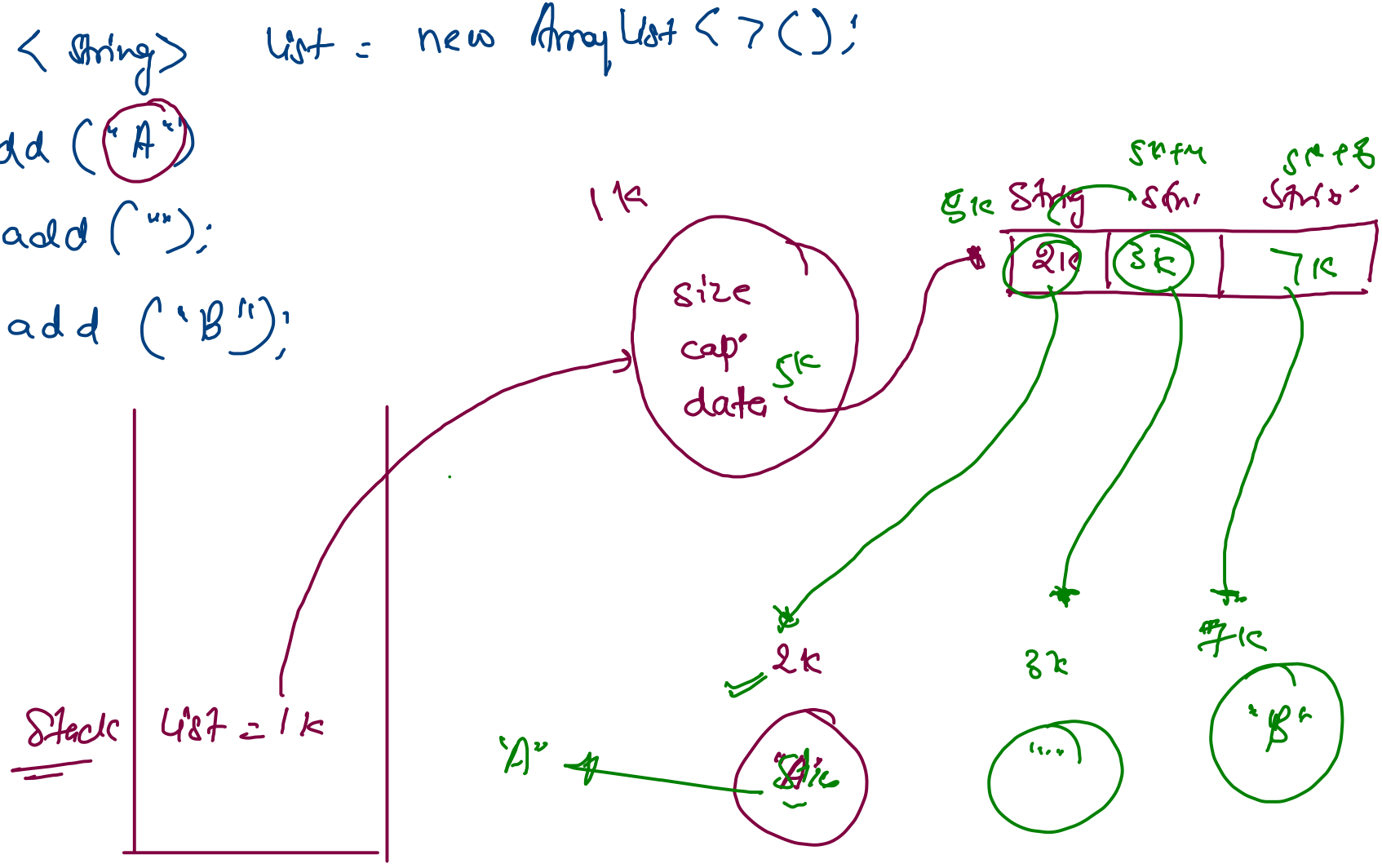
garbage Collector



String Str = "";


String- ""
brej
2k
brej:
String- c
rrej
2k
c
mrej
4k
String- bc
gss
rrej
b
c
mrej
String = abc
gss
a
rrej=
bc
mrej
1k
3k
main
rej =
2k
String. length = 0
4k
buffer
Reference
1k
8
8k
- - -, - - c, - b -, - bc,
a - -, a - c, a b -, abc

```
ArrayList<String> list = new ArrayList<>();
list.add("A");
list.add("");
list.add("B");
```



String str = scn.next();

Hello world

Str: Hello

Read string until first space

String str = scn.nextLine();

Read the content until "Enter" is not Encounter

"Hello world"

Str = "Hello world"

Kpc . key pad code

op1 op2 x op3

f f

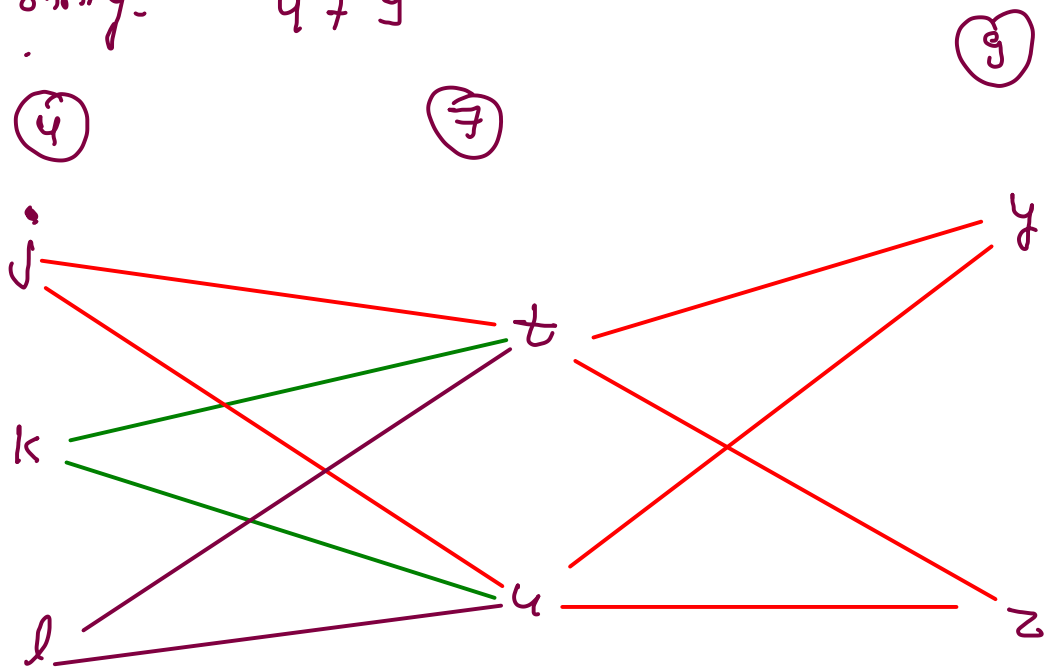
4 7 9

4 7 9

Mapping.

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

string.



3 x 2 x 2.

= 12

jty
jtz
juy
juz

4

ktv
ktz
kuy
kuz

4

lty
ltz
luy
luz

4

String - "453"

char ch = Str.charAt(0);

ch : '4' ⟶ as character

4 ⟶ as Integer.

as Binary. ↓ Integer

} all possible Characters are mapped Integer

ASCII

int indx = Str.charAt(0) - '0';

'4' - '0'

get kpc

High level

Codes

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

Code

479

Expectation (479) → all possible mapping.

faith (79) → all possible mapping.

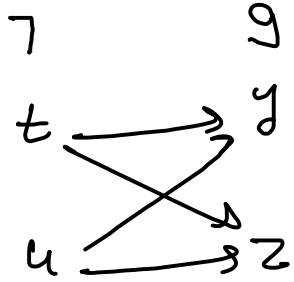
Merging - (479) → res + (79)

option (res)

4 → option

j → res {j ty j tz j uy j uz

k → res k ty k tz k uy k uz



79 → {ty, tz, uy uz}

l → res l ty l tz l uy l uz}

recursion.java > recursion > getKPC(String)

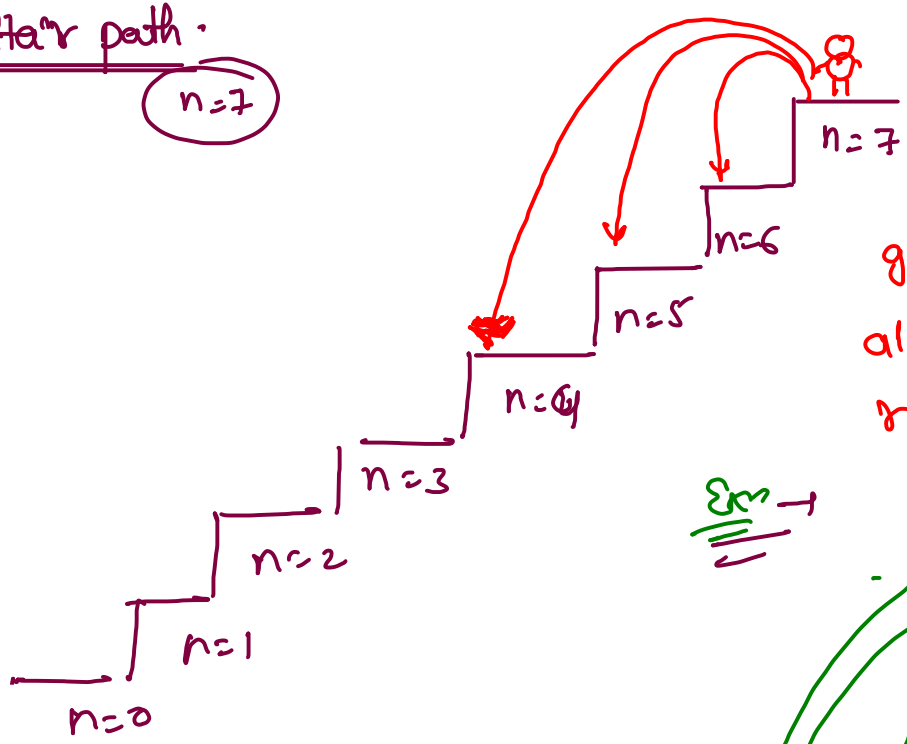
```java
import java.util.*;

public class recursion {

    public static ArrayList<String> getSubSeq(String str) {...

    public static String[] codes = { ".;", "abc", "def", "ghi", "jkl", "mno

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

        int indx = str.charAt(0) - '0';
        String roq = str.substring(1);
        String code = codes[indx];
        ArrayList<String> rres = getKPC(roq);

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

        for(int i = 0; i < code.length(); i++) {
            char ch = code.charAt(i);
            for(String s : rres) {
                mres.add(ch + s);
            }
        }

        return mres;
    }


    public static void ques() {
        String str = "abc";
```

output.txt

```
4

{ "ty", "tz", "uy", "uz"}

{ "jty", "jtz", "juy", "juz"

kty, ktz, kuy, kuz

lty ltz luy luz

}
```

Stair path.

n=7

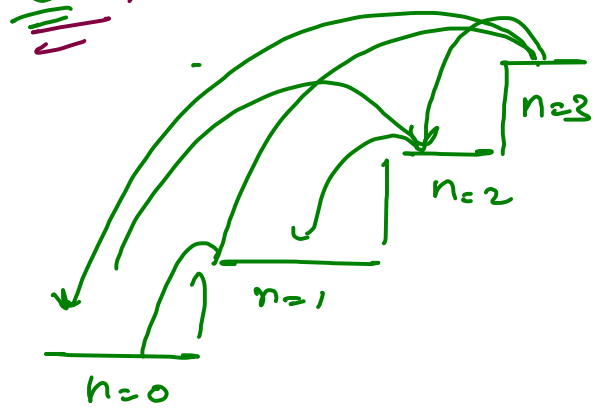


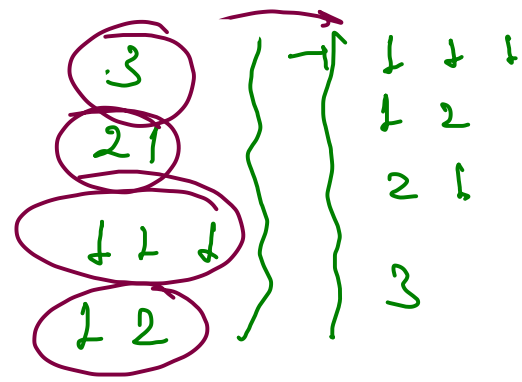
jump - (1) (2) (3)

goal

all possible path so that we can reach at ground (n=0) from Topfloor(n=7)

Ex →





High level Analysis

Expectation

stairpath (n:7) → all possible paths.

faith

- stairpath (6) → all possible path from 6 to 0
- stairpath (5) → " " " " 5 to 0
- stairpath (4) → " " " " 4 to 0

Merging of faith & Expectation

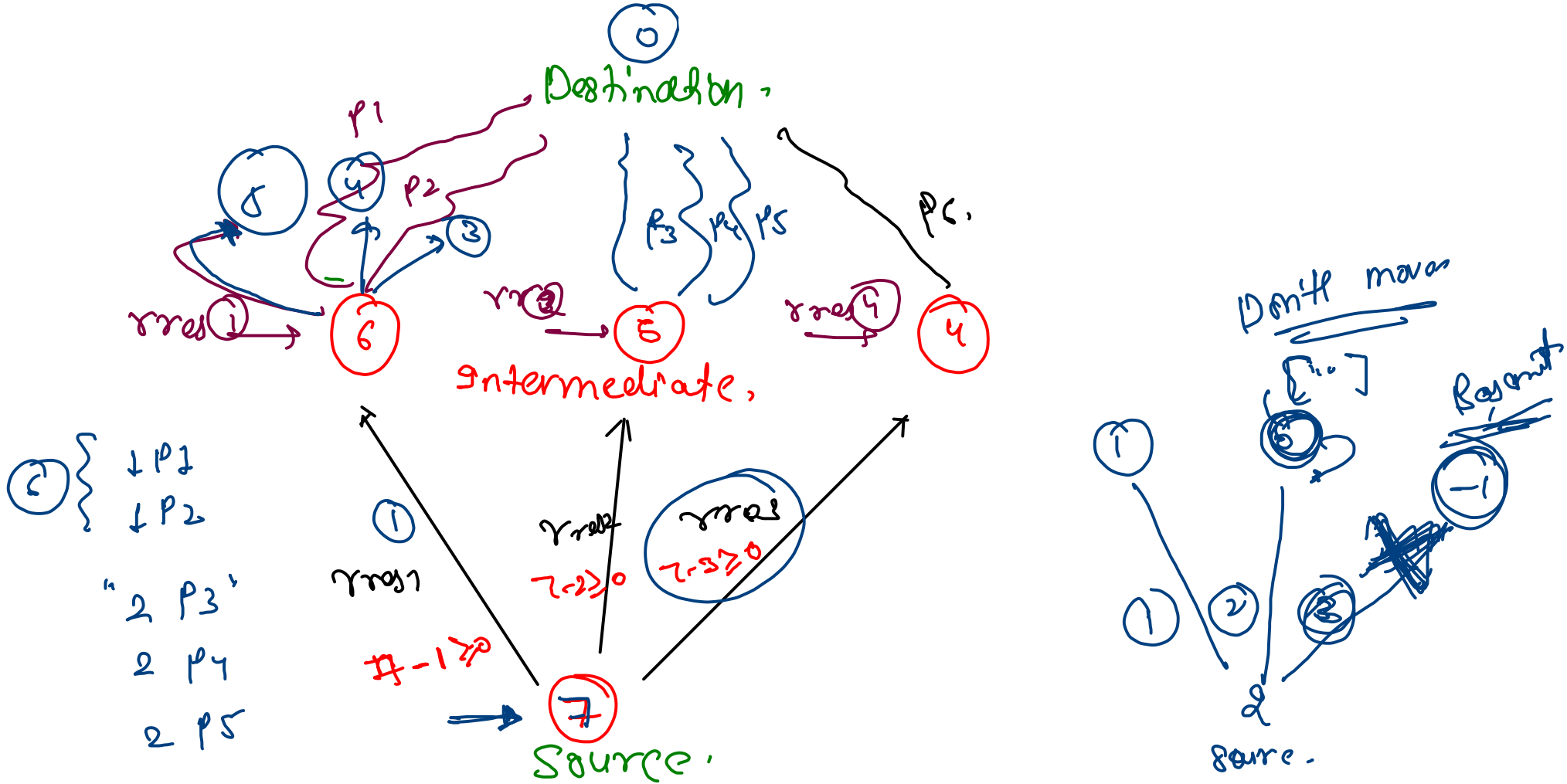
stairpath (7) →

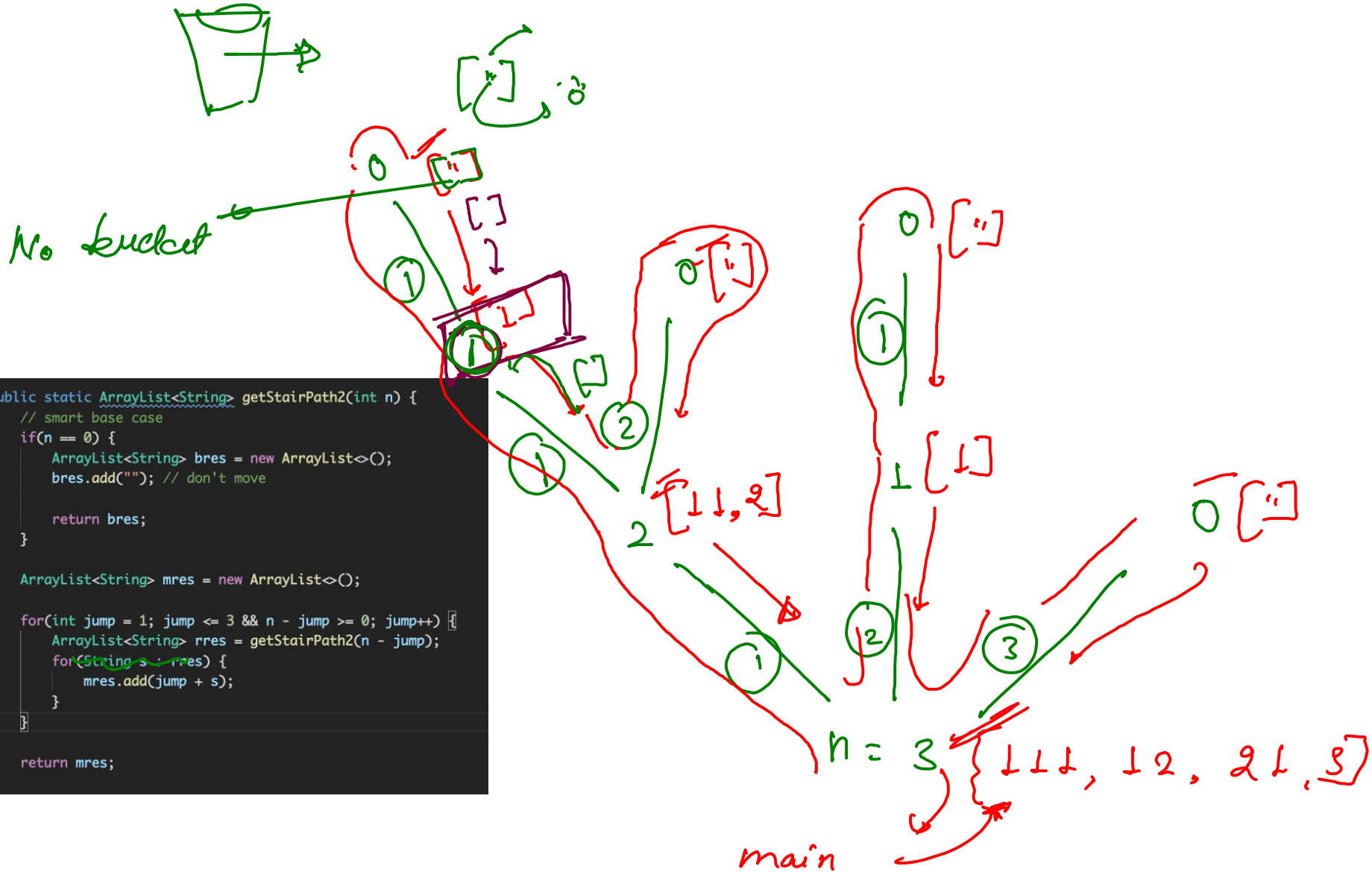
rres1 = stairpath (6)

rres2 = stairpath (5)

rres3 = stairpath (4)

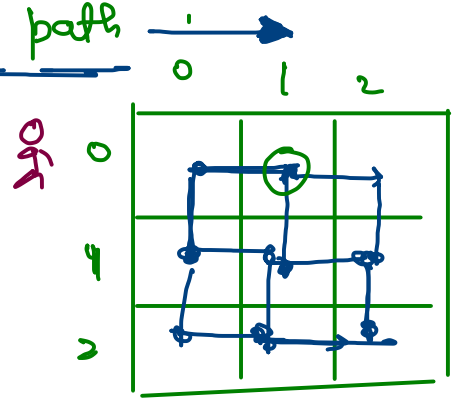
rres = {'1' + rres1, '2' + rres2, '3' + rres3}


0
Destination.
p1
p2
p3
p4
p5
pc.
1
2
3
4
5
6
7
intermediate.
Source.
2 p1
2 p2
2 p3
2 p4
2 p5
3 p6
Don't move
Source.




```java
public static ArrayList<String> getStairPath2(int n) {
    // smart base case
    if(n == 0) {
        ArrayList<String> bres = new ArrayList<>();
        bres.add(""); // don't move

        return bres;
    }

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

    for(int jump = 1; jump <= 3 && n - jump >= 0; jump++) {
        ArrayList<String> rres = getStairPath2(n - jump);
        for(String s : rres) {
            mres.add(jump + s);
        }
    }

    return mres;
}
```

Maze path



Horizontal toward Right

vertical toward down.

print all path so that we can reach from src to dst

src (0, 0) dst (m-1) (n-1)

Ex

H H V V

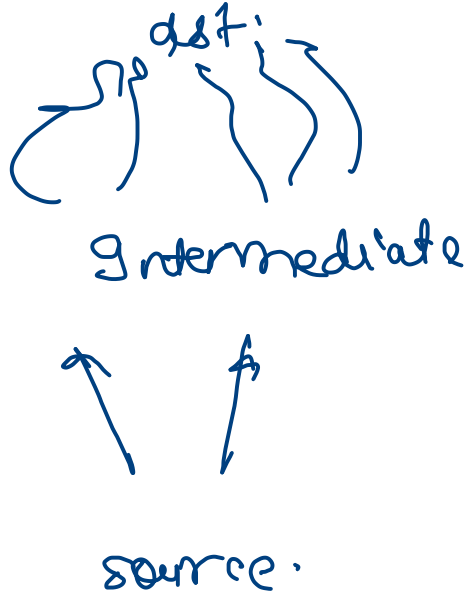
V V H H

H V H V

V H V H

⋮

all paths.



Expectation → solve(0, 0, 2, 2) → all paths from src to dst

faith → hres = solve(0, 1, 2, 2) → ..

vres = solve(1, 0, 2, 2) → ..

Merging → solve(0, 0, 2, 2) = hres = solve(0, 1, 2, 2)

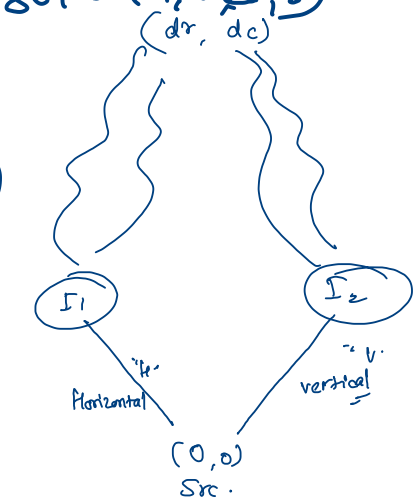
vres = solve(1, 0, 2, 2)

res =

("H" + hres)

("V" + vres)

return res

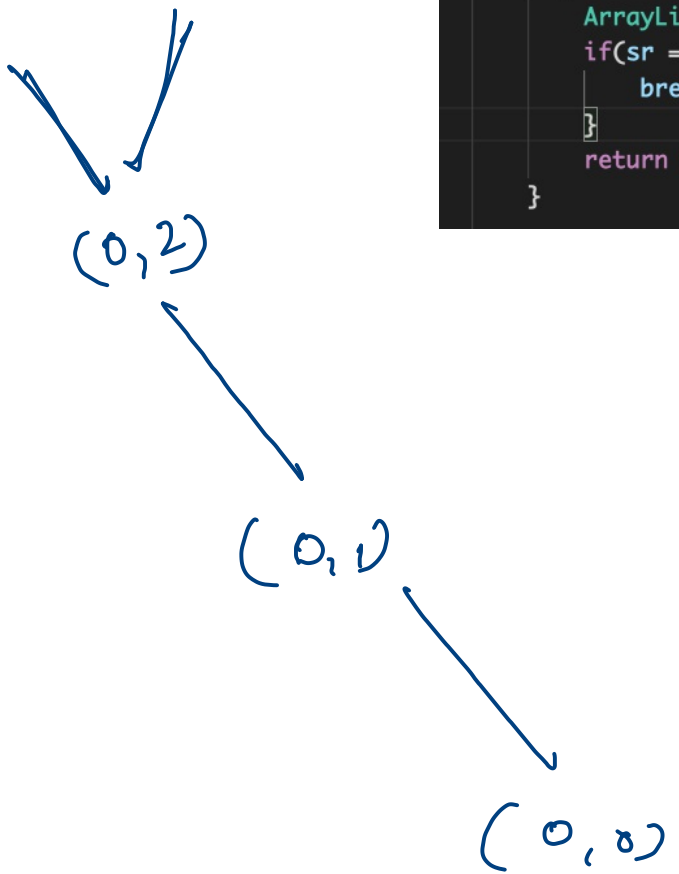


r, c → Hr → (r, c+1)

↓ V

(r+1, c)

```java
public static ArrayList<String> getMazePath(int sr, int sc,
    System.out.println(sr + " " + sc);
    if(sr >= dr || sc >= dc) {
        ArrayList<String> bres = new ArrayList<>();
        if(sr == dr && sc == dc) {
            bres.add("");
        }
        return bres;
    }
```



wrong base case