① Get Maze Paths
② Get Maze Path with Jumps (HW)
③ Print Subsequences
④ Print Keypad
★) Print Permutations.
⑤ Print Stair Paths
⑥ Print Maze Paths
⑦ Print Maze paths with jumps (HW)
⑧ Print Encodings.

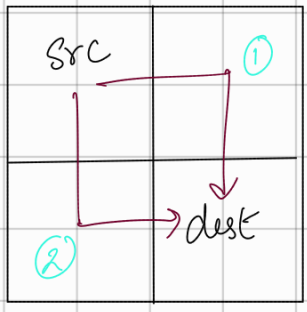
Get

Print

Next week → Print Permutations
→ Floodfill → Target Sum Subset
→ N-Queen → N Queen

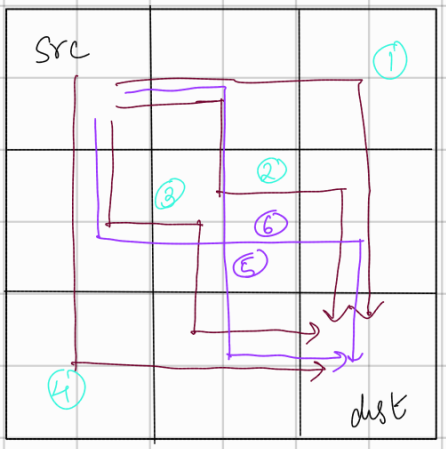
① Get maze Paths.

Print ways to reach destination (dr, dc) from source (sr, sc)



⟹ 2 ways.

① 'h, v'
② 'v, h'.

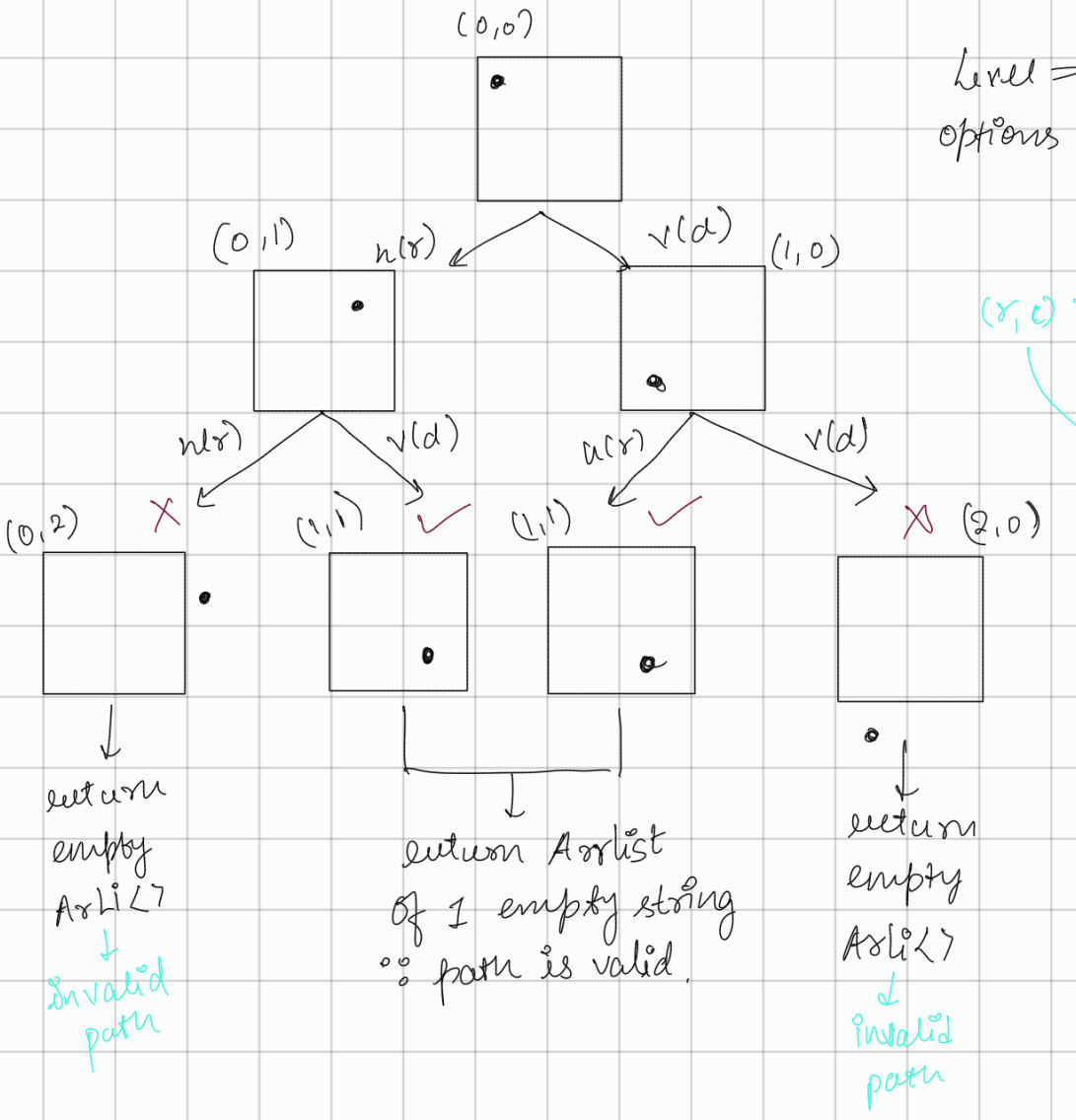


① 'h h v v',
② 'h v h v',
③ 'v h v h',
④ 'v v h h'
⑤ 'h v v h',
⑥ 'v h h v',

return ALL

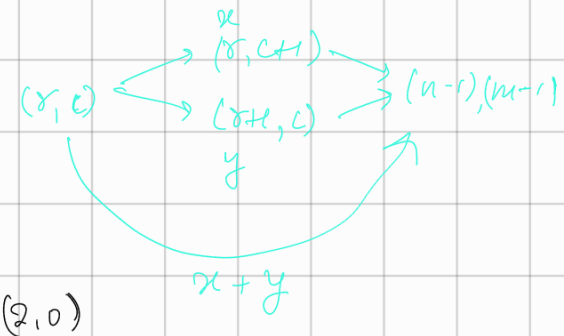
Expectation: returns ArrayList of Strings with paths from src to dest.

Faith: gMP(a, b, c, d) gives ArrayList of paths from (a,b) to (c,d) correctly



Level ⇒ cell

options ⇒ neighbours.

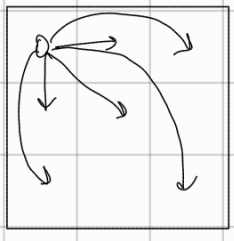


∴ CODE:

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

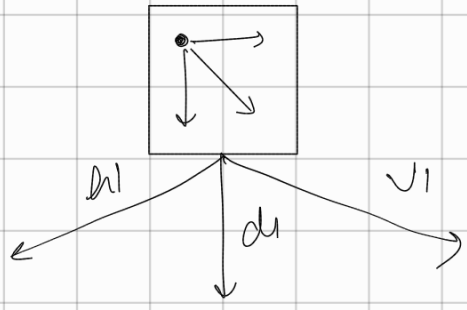
```
ArrayList<String> ans = new ArrayList<>();
ArrayList<String> rightPaths = getMazePaths(sr, sc+1, dr, dc);
for(String s: rightPaths)
    ans.add("h"+s);

ArrayList<String> downPaths = getMazePaths(sr+1, sc, dr, dc);
for(String s:downPaths)
    ans.add("v"+s);

return ans;
```

② Get maze Path with jumps



6 ways from each cell

Use Smart Calls.



Path lengths : Greedy → (n-1) + (m-1)

No. of Paths : DP. → CATALAN

∴ CODE:

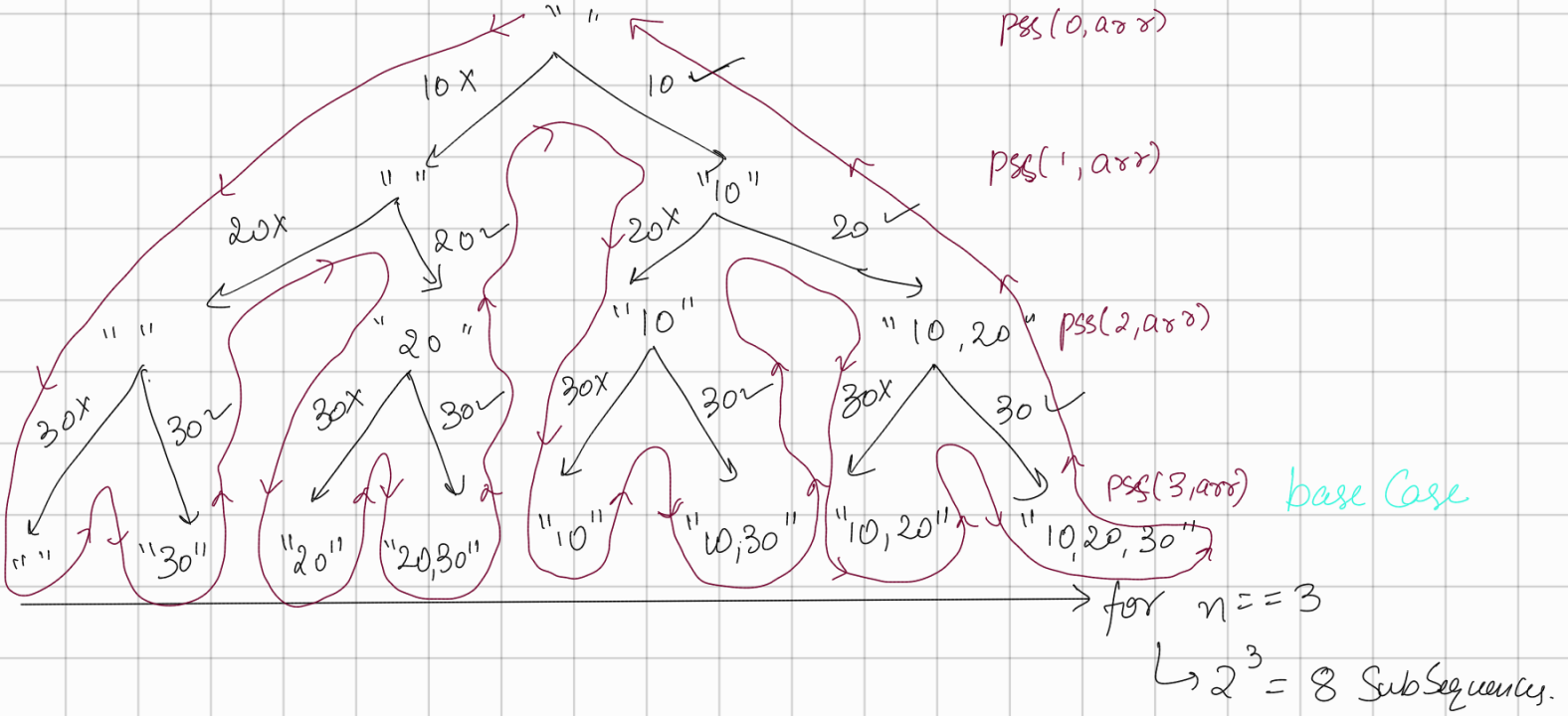
```
if(sr==dr && sc==dc){
    ArrayList<String> base = new ArrayList<>();
    base.add("");
    return base;
}

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

int jump=1;
while(sc+jump<=dc){
    ArrayList<String> rightPaths = getMazePaths(sr, sc+1, dr, dc);
    for(String s:rightPaths)
        ans.add("h"+jump+s);
    jump++;
}

jump=1
while(sr+jump<=dr){
    ArrayList<String> downPaths = getMazePaths(sr+jump, sc, dr, dc);
    for(String s:downPaths)
        ans.add("v"+jump+s);
    jump++;
}

jump=1
while(sr+jump<=dr && sc+jump<=dc){
    ArrayList<String> diagonalPaths = getMazePaths(sr+jump, sc+jump, dr, dc);
    for(String s:diagonalPaths)
        ans.add("d"+jump+s);
    jump++;
}

return ans;
```

③ Print Subsequences.

{10, 20, 30, 40}

void printSS (0, arr)

Here recursion tree will be inverse, largest in basecase

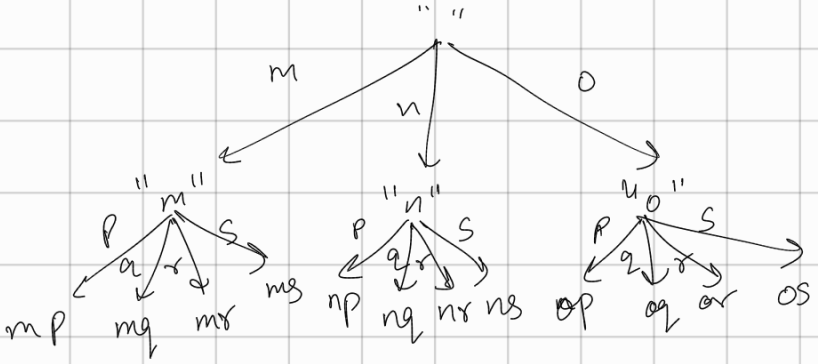


Depth first Search == Recursion

∴ CODE:

```
if (idx == input.length()) {                    // base case
    System.out.println(output);
    return;
}
char c = input.charAt(idx);
printSS(idx+1, input, output + c);   // Yes for c
printSS(idx+1, input, output);       // No for c.
```

④ Print KeyPad Combinations.



5 → "m, n, o"

6 → "p, q, r, s"

∴ CODE

```
if (idx == str.length()){
    System.out.println(asf);
    return;
}
char c = str.charAt(idx);
String s = dtoc[c-'0'];
for(int i=0; i<s.length; i++)        // Explore all chars at num
    printKPC(i+1, str, asf+s.charAt(i));
```

In Print problems, we update answers for each case and print these answers in base case.

To get ArrayList values from print functions with void return type

① We can declare List statically & add in base case

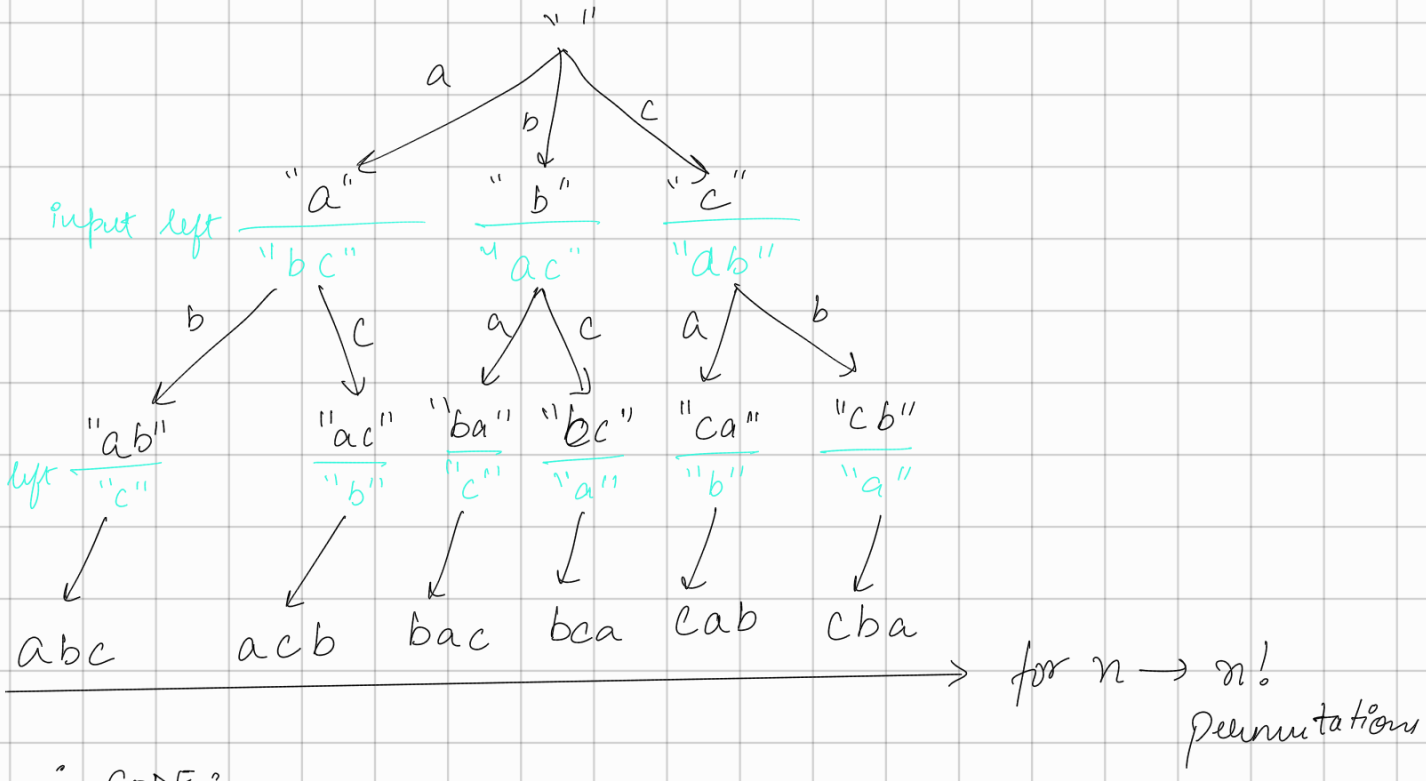
② We can declare List in main & pass in args. Update in base case.

⑤ Print Permutations.

Permutations: ways/arrangements possible for given string

Eg: a b c → "a b c"
"a c b"
"b a c"
"b c a"
"c a b"
"c b a"

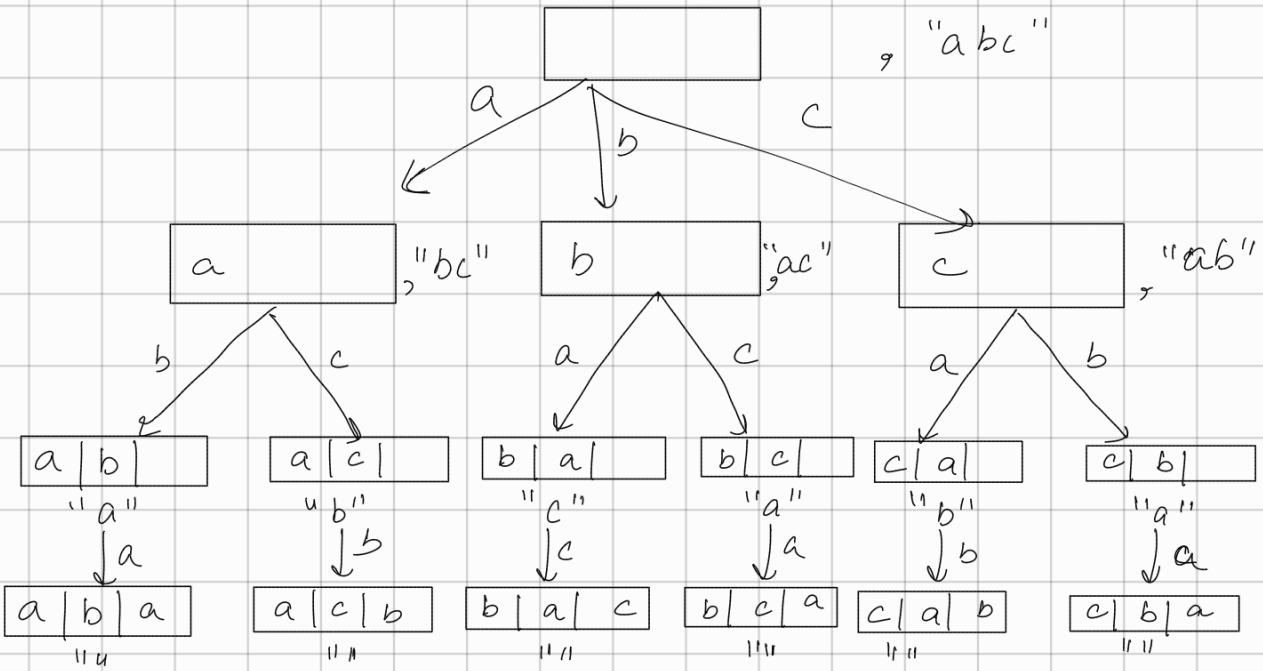
★ Box On Level Approach ★ for "a b c"



∴ CODE:

```
if (input.length == 0) {
    System.out.println(output);
    return;
}
for (int i=0; i<input.length(); i++) {
    String newInput = input.substring(0, i) + input.substring(i+1);
    printPermutations(newInput, output + input.charAt(i));
}
```

Using Boxes: for "a b c"



input string empty

∴ Base Case

Print Output String