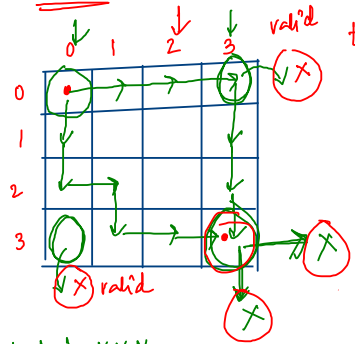
★ Maze paths - 1 :

(dr, dc) ⇒ no. of cols ⇒ dc + 1
(0,0) → (3,3) ⇒ no. of rows ⇒ dr + 1



top left → bottom right right → horizontal (h)
print all the ways down → vertical (v)

op: h h h v v v
v v h v h h

1. faith

⇒ printAllpaths (sr, sc, dr, dc, path)

→ sr, sc, dr, dc, It will print all the paths from (sr, sc) → (dr, dc)

2. logic

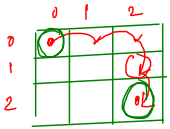
(0,0) (0,1)
(r, c) → (r, c+1)
↓
(r+1, c)
(1, 0)

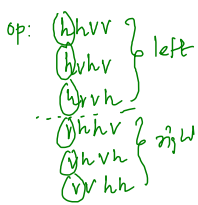
printAllpaths (sr, sc+1, dr, dc, path + "h");
printAllpaths (sr+1, sc, dr, dc, path + "v");

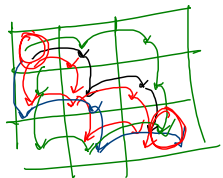
sr, sc, dr, dc

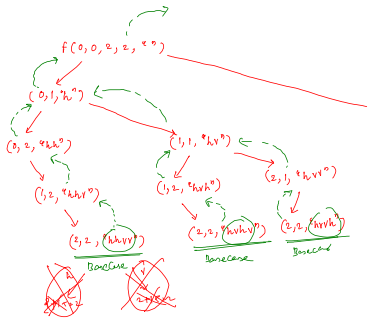
path ⇒ current path until sr, sc
(how did you reach sr, sc from (0,0))

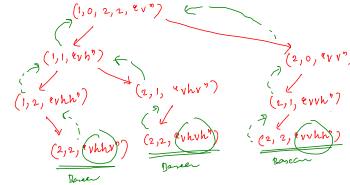
```
function printMazePaths(sr, sc, dr, dc, path) {
  if (sr == dr && sc == dc) {
    console.log(path);
    return;
  }

  if (sc + 1 <= dc) {
    printMazePaths(sr, sc + 1, dr, dc, path + "h");
  }
  if (sr + 1 <= dr) {
    printMazePaths(sr + 1, sc, dr, dc, path + "v");
  }
}
```



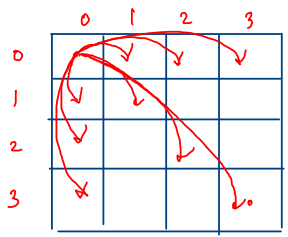






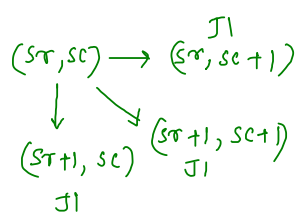


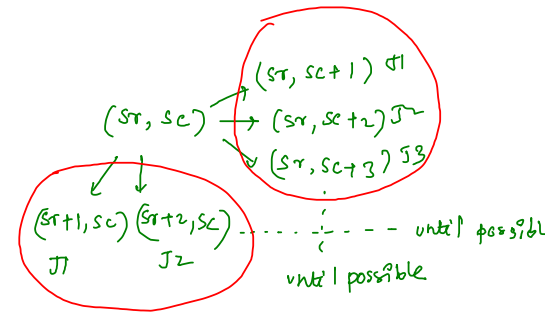
* Maze paths - 2 : (with Jumps)



(sr, sc) → (dr, dc)

(0, 0) → (3, 3)





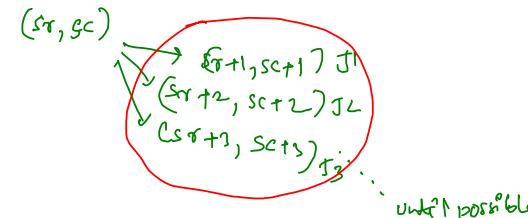
op: h3 v3

d3

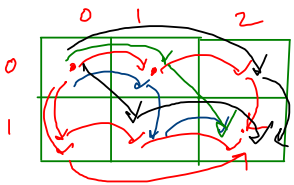
d1 d1 d1

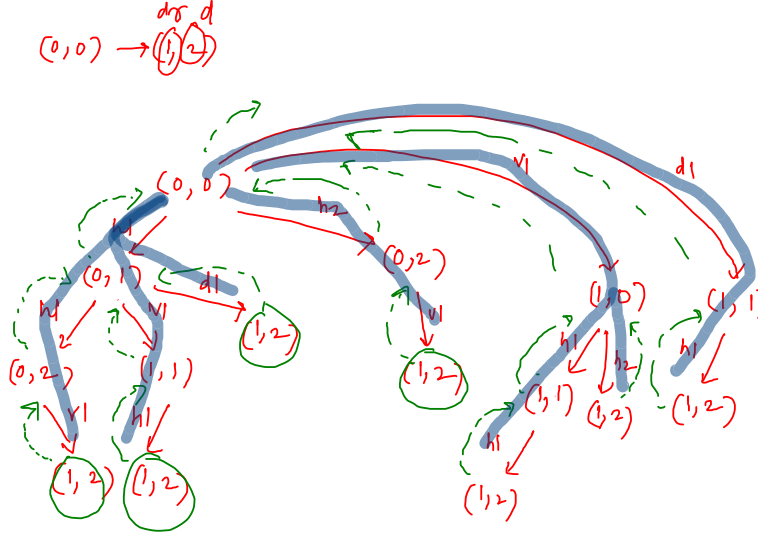
h1 h1 h1 v1 v1 v1

h2 h1 v2 v1



```
function printMazeJumps(sr, sc, dr, dc, path) {
  if (sr == dr && sc == dc) {
    console.log(path);
    return;
  }

  // I can go right by any no.of jumps
  for (let jump = 1; sc + jump <= dc; jump++) {
    printMazeJumps(sr, sc + jump, dr, dc, path + "h" + jump);
  }

  // I can go down by any no.of jumps
  for (let jump = 1; sr + jump <= dr; jump++) {
    printMazeJumps(sr + jump, sc, dr, dc, path + "v" + jump);
  }

  // I can go diagonally by any no.of jumps
  for (let jump = 1; sr + jump <= dr && sc + jump <= dc; jump++) {
    printMazeJumps(sr + jump, sc + jump, dr, dc, path + "d" + jump);
  }
}
```





★ permutations :

→ we have seen Combinations / subsets / subsequences (consider, not consider)

ip: "abc"

op:
abc
acb
bac
bca
cab
cba

_ _ _
↓ ↓ ↓
3 2 1

total ways = 3 × 2 × 1
= 3!
= 6

n! permutations

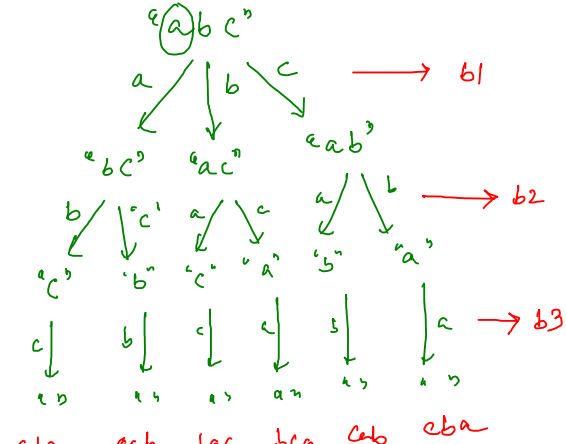
1. permute ("abc")

⇒ It will print all permutations of "abc"

2. permute ("abc") ⇒ abc
acb

permute ("aac") ⇒ bac
bca

permute ("aab") ⇒ cab
cba



```
function permute(str, path) {
  // no ch in str to pick
  if (str.length == 0) {
    console.log(path);
    return;
  }

  // you can choose any ch from str in this blank
  // remaining str apart from ch will be the options for next blank(call)
  for (let i = 0; i < str.length; i++) {
    // exclude current ch so that options for next blank doesnt include current ch
    permute(str.slice(0, i) + str.slice(i + 1), path + str[i]);
  }
}
```

slice(0,0) + slice(1)

"" + "bc" ⇒ "bc"

slice(0,0) + slice(1)

"" + "c" ⇒ "c"

slice(0,0) + slice(1)

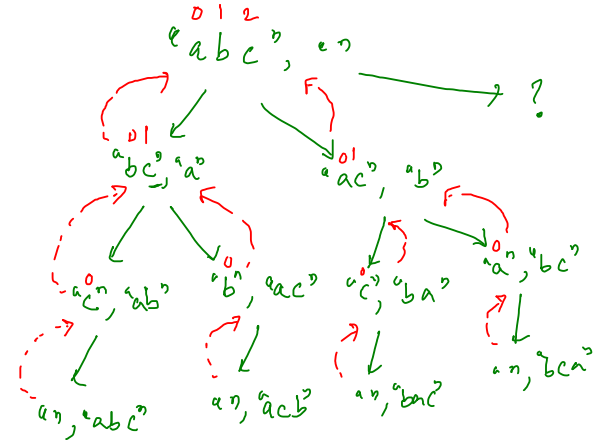
"" + "" ⇒ ""

slice(0,1) + slice(2)

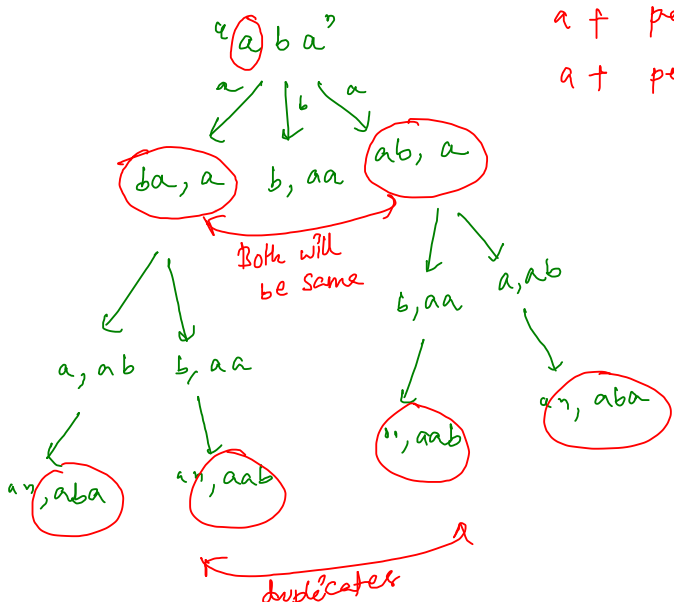
"b" + "" ⇒ "b"

slice(0,1) + slice(2)

"a" + "c" ⇒ "ac"



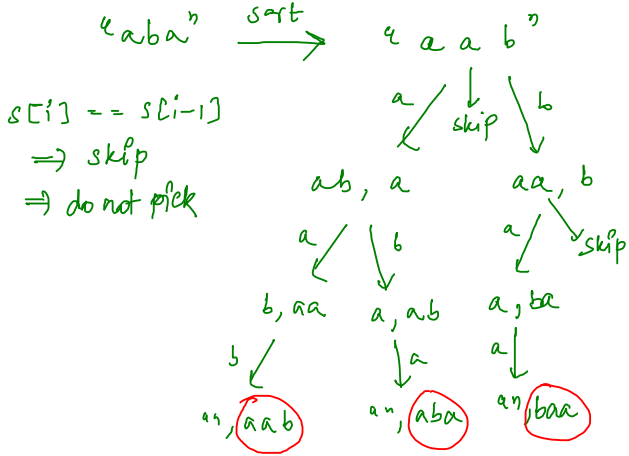
unique permutations (follow-up) :



a + permute ("aba") → duplicate

a + permute ("ab") → duplicate

→ we need to know some how that we already filled 'a' before. (next page)



IRS

doubts + revision

Sat – class

Sun – contest