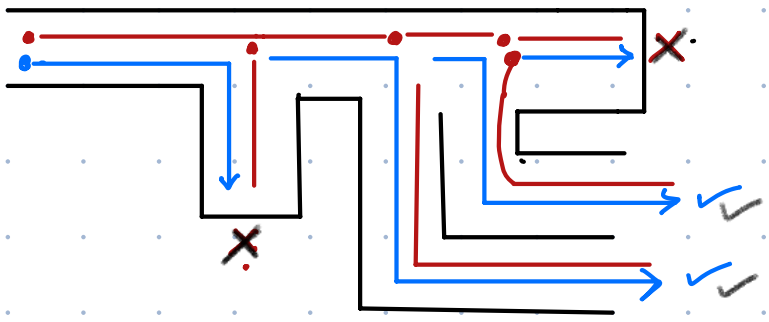
Backtracking: generating all solutions using recursion.

- all paths
- all permutations
- all subsets

Maze: all ways to exit maze



Idea: while generating all possible solutions —

Choose path

1. Path – blocked — comeback ✗
2. Path – solution — comeback ✓

Q1 Print all valid parenthesis ⟶

↳ ()

N = 2

2 O, 2 C

( ) ( )

(( ))

N = 3

5 { ((( ))) 3O, 3C
(() ()) 3O, 3C
(( )) ( ) 3O, 3C
( ) (( ))
( ) ( ) ( )

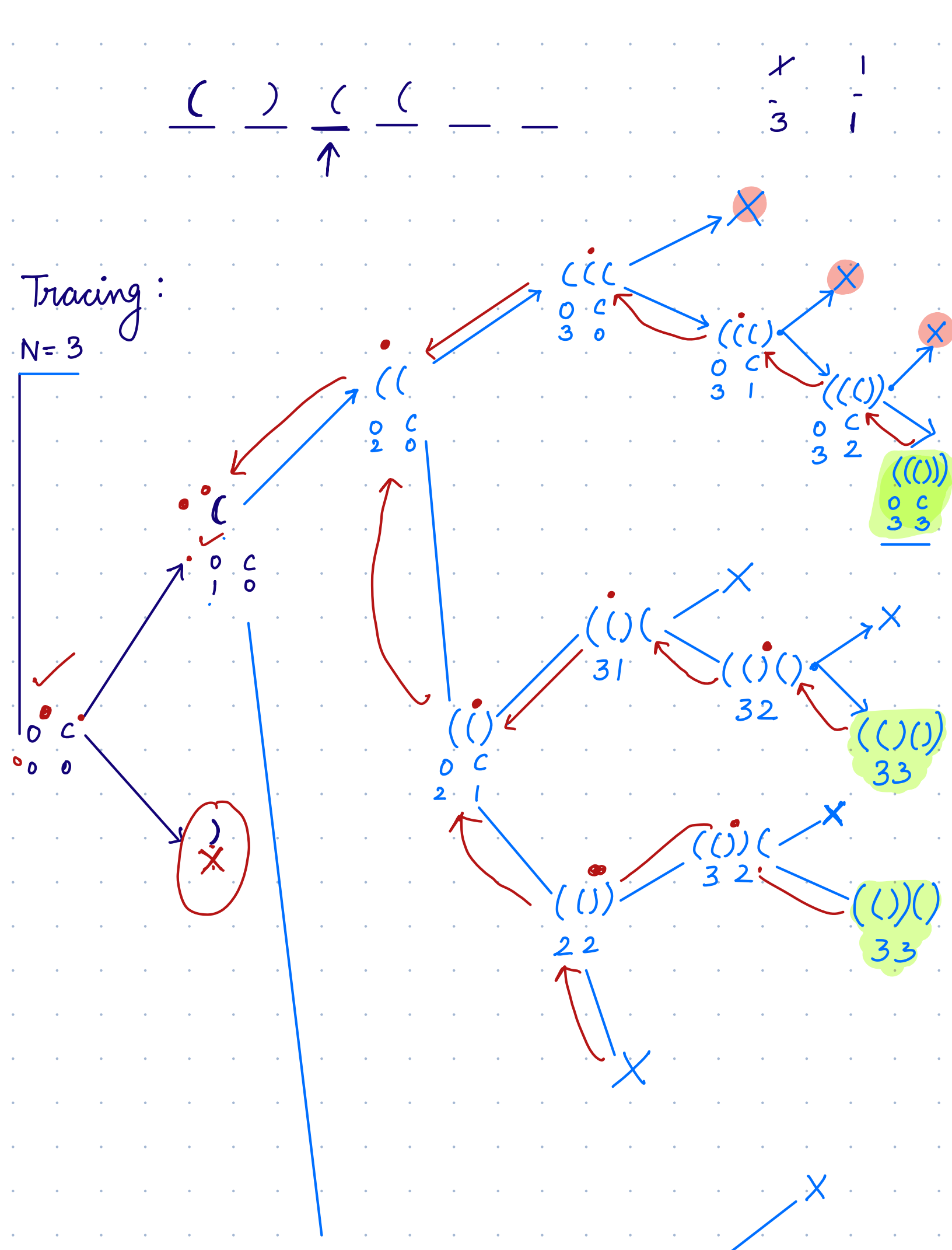
N = 3 ⟶ 3 O, 3 C

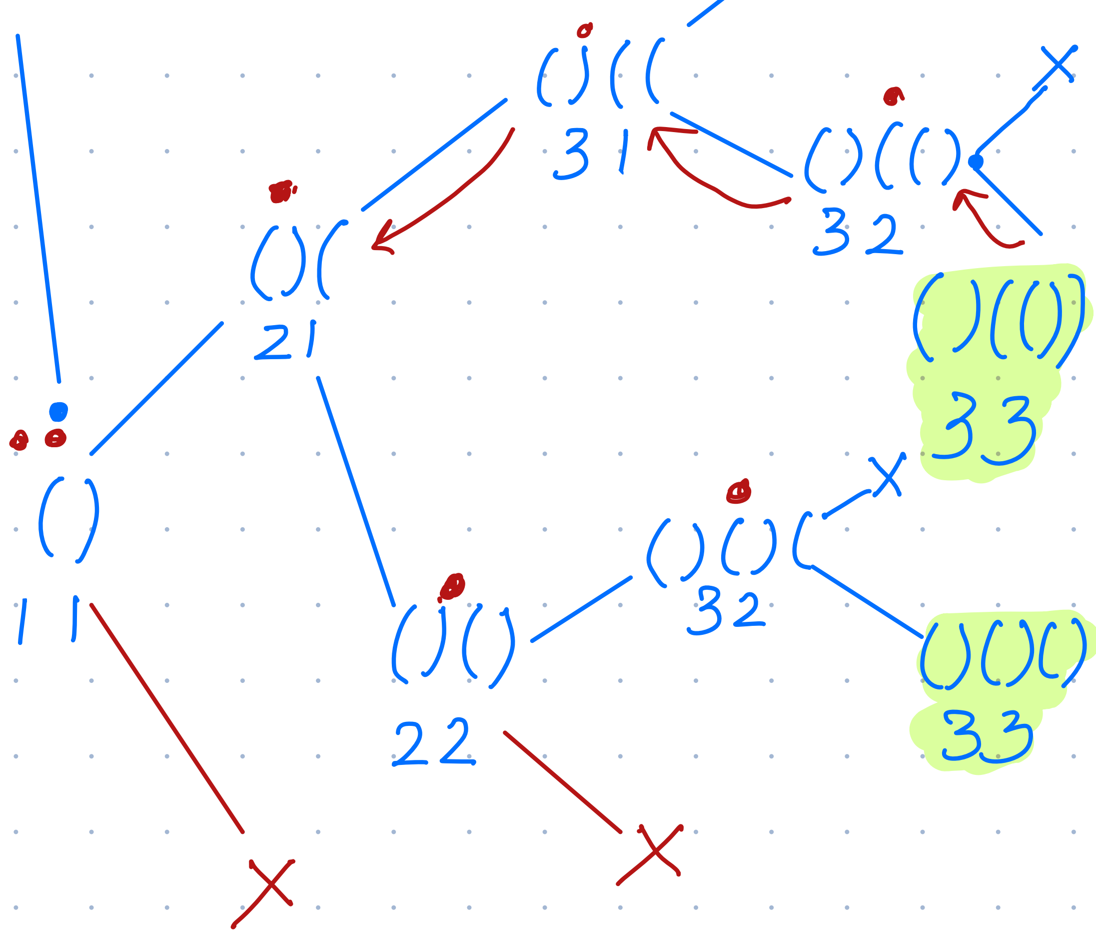
O	C
~~0~~	~~0~~
~~1~~	1
~~2~~	
3	

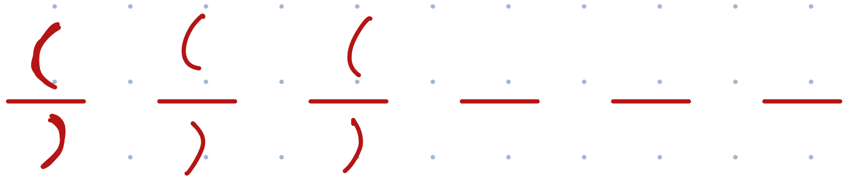
( ) (( ))

C < O ← for closing

O < N ← for opening brackets


( ) ((_ _
x 1
3 1
Tracing :
N = 3
0 0 0
0 C
1 0
)
0 C
2 0
(
0 C
2 1
((
0 C
3 0
(()
0 C
3 1
((())
0 C
3 2
(((()))
0 C
3 3
X
()(
31
(()()
32
(()())
33
(())
22
(())(
32
(())()
33


()
11
()(
21
()()
22
()((
31
()(()
32
()()(
32
()((()))
33
()()()
33



- Recursive calls → 2 → (and)
- Parameters →

String res
int o
int c
int N

```
void printAll(String res, int o,
              int c, int N) {
    if (res.length() == 2*N) {print(res) return;}

    if(o < N)
        printAll(res + "(", o+1, c, N);
    if(c < o)
        printAll(res + ")", o, c+1, N);
    return; // good practice
}
```

Ques2. Subsets & subsequences

subarray = continuous part of array
single element, complete array

subsequence =

L or P

0	1	2	3	4	5
7	2	6	9	10	8

arr[] = { 7 2 6 9 10 8 }

2 9 10

1) Not necessarily continuous
2) i → L or P.

2 6 10 8
1 2 4 5

3.) Order of index should be maintained

6 2 7 10 ✗

↓

7 2 6 10 Correct order of index

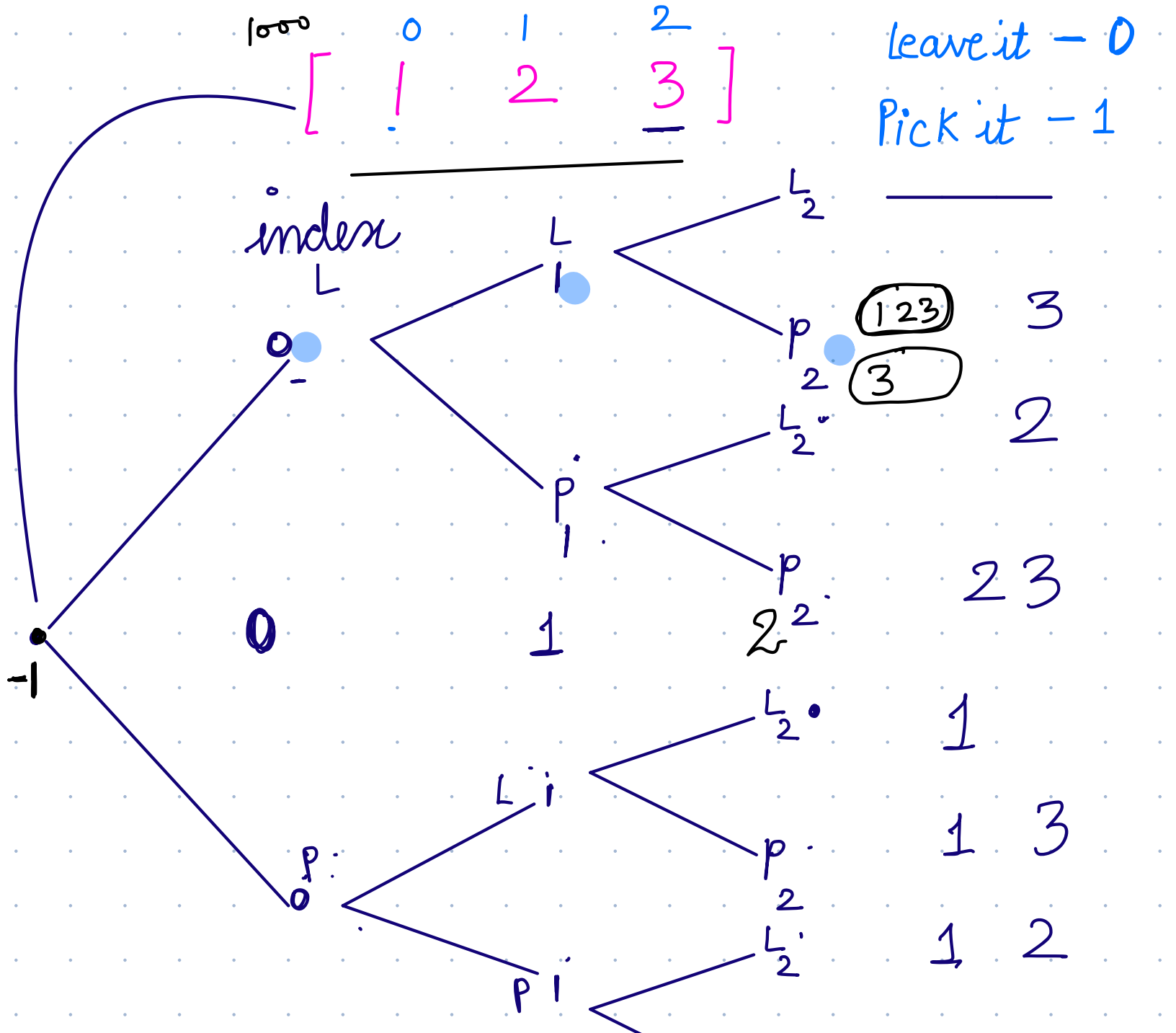
7 2 6 9 8 is it subsequence? YES

7 6 9 10 Yes

Subsets → same as subsequences not need to maintain order

7, 6, 9, 10

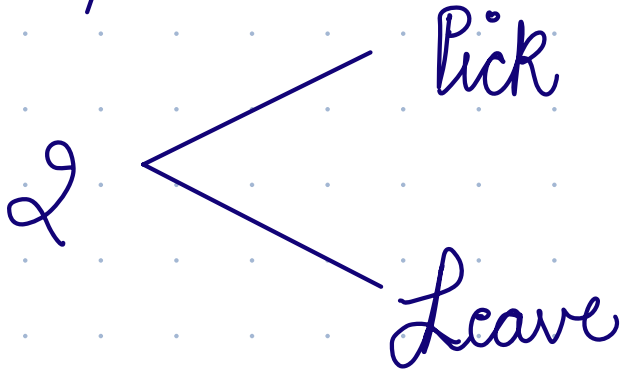
7 9 6 10



Parameters —

int index
list ans
int [] arr

Recursion / Recursive calls —



Code:

```
void subsequences(list ans, int[] A, int index) {
    if (index == A.length) {
        print(ans)
        return;
    }
    //choice 1: Leave it
    subsequences(ans, A, index+1);

    // choice 2: Pick it
    ans.add(A[index])
    subsequences(ans, A, index+1);
    ans.remove(A[index])
}
```

Q.

abc → 3!

a b c
a c b
b a c
b c a
c a b
c b a

} 6

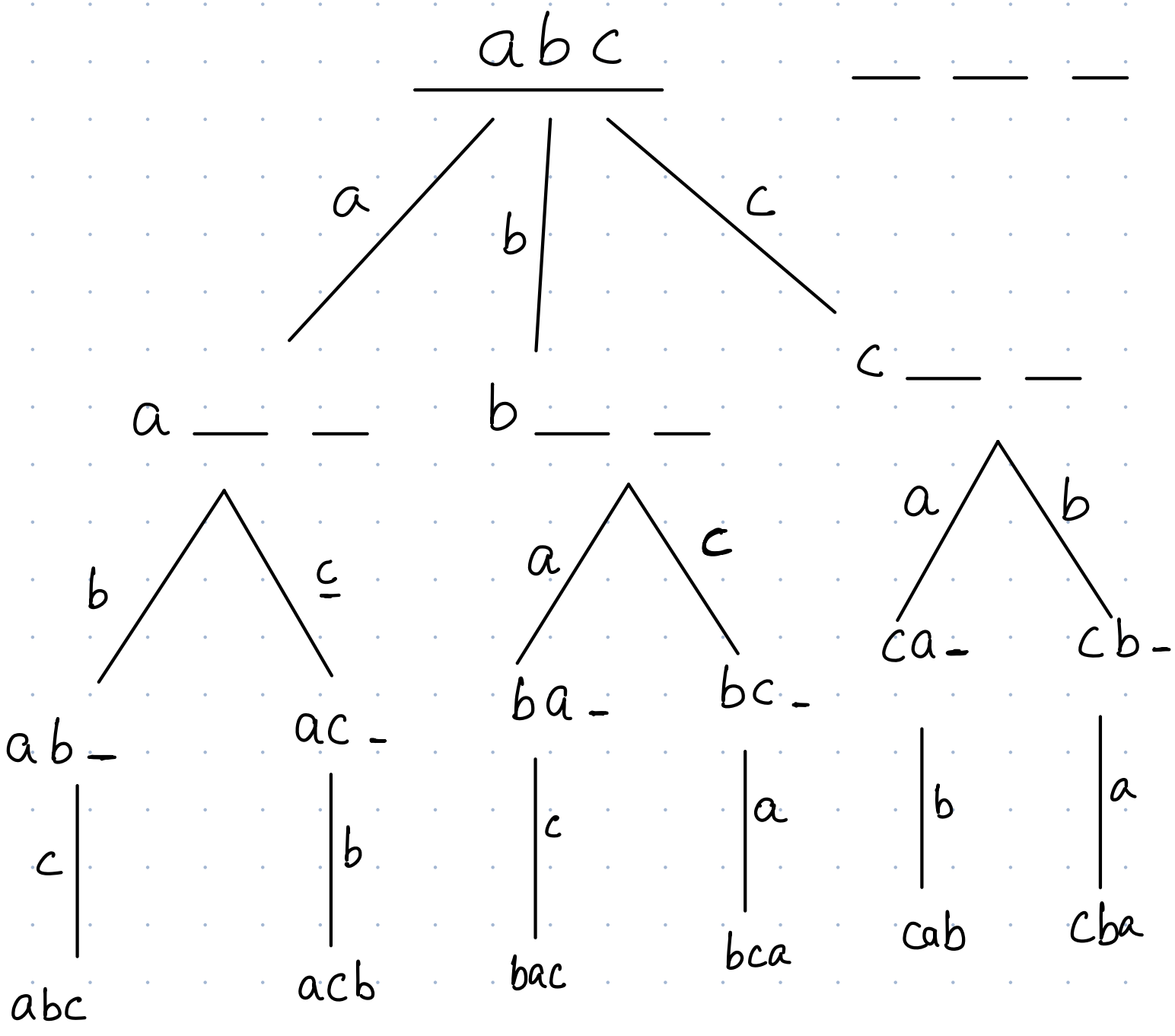
CureFit

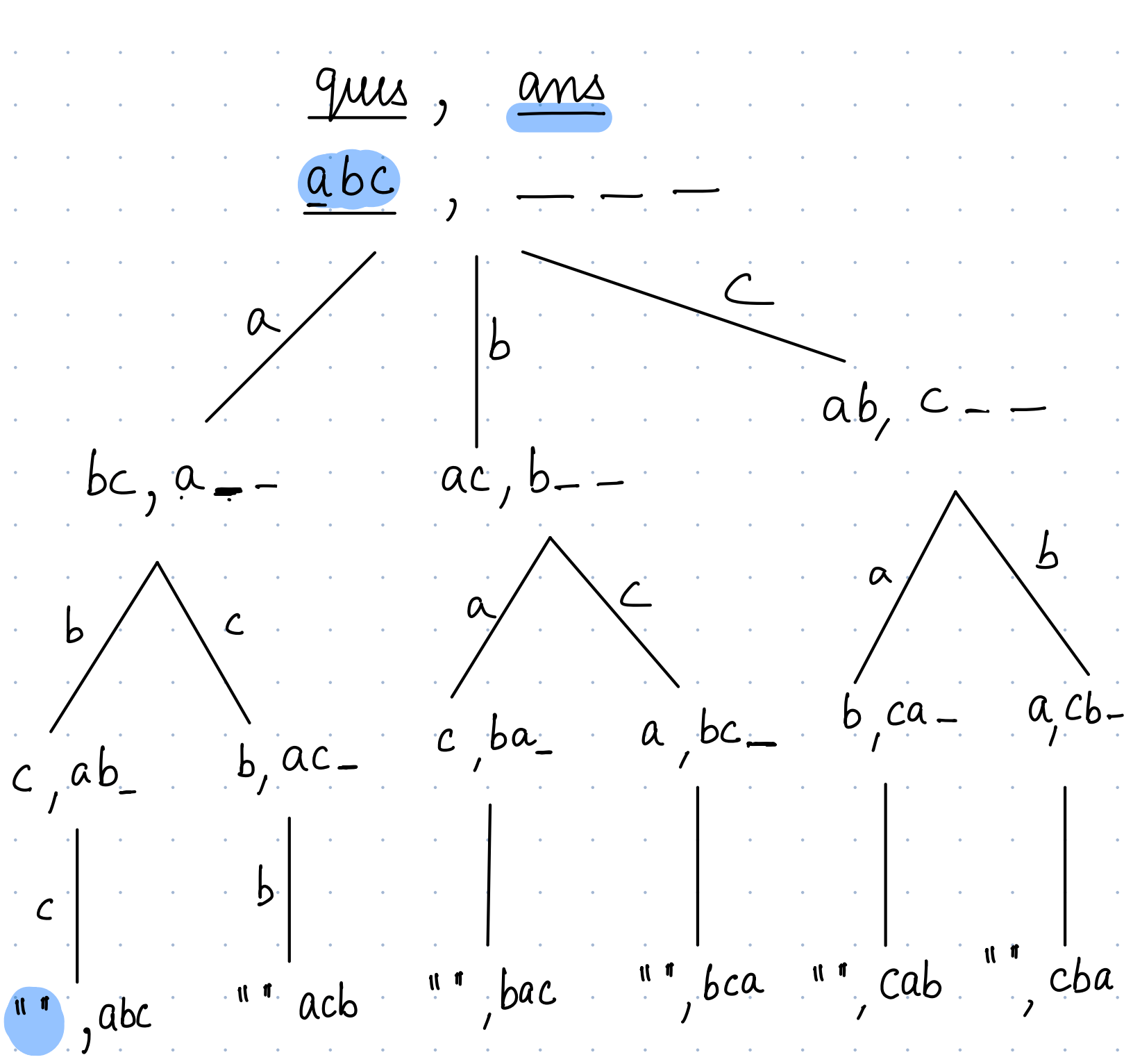
a = Pushups
b = Planks
c = Burpees

} all chars will be distinct

all possible permutations?

a b c


abc
a
b
c
a__ __
b__ __
c__ __
b
c
a
c
a
b
ab-
ac-
ba-
bc-
ca-
cb-
c
b
c
a
b
a
abc
acb
bac
bca
cab
cba




```
if (ques.length() == 0) {
    print (ans)
    return ;
}
```

Parameters :

String ques

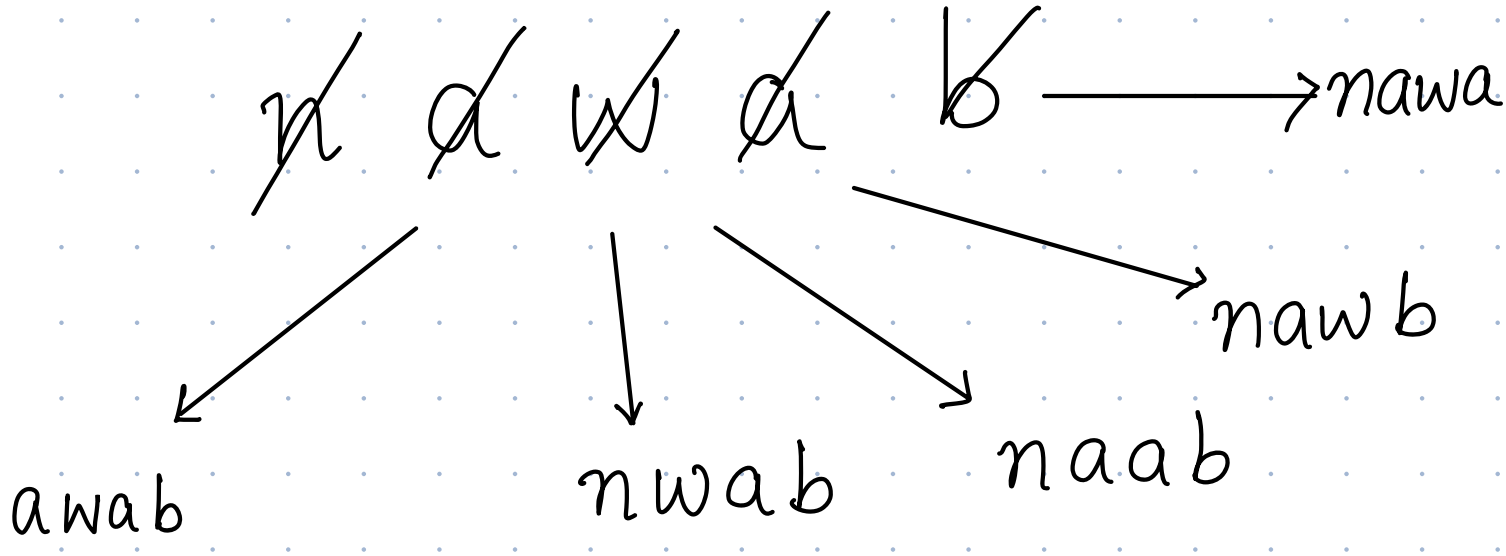
String ans

Recursive calls —

chars of ques string

↓

length of string



substring(A, B) ⟶ A to B-1

i

substring(0, i) + substring(i+1, N)

ques

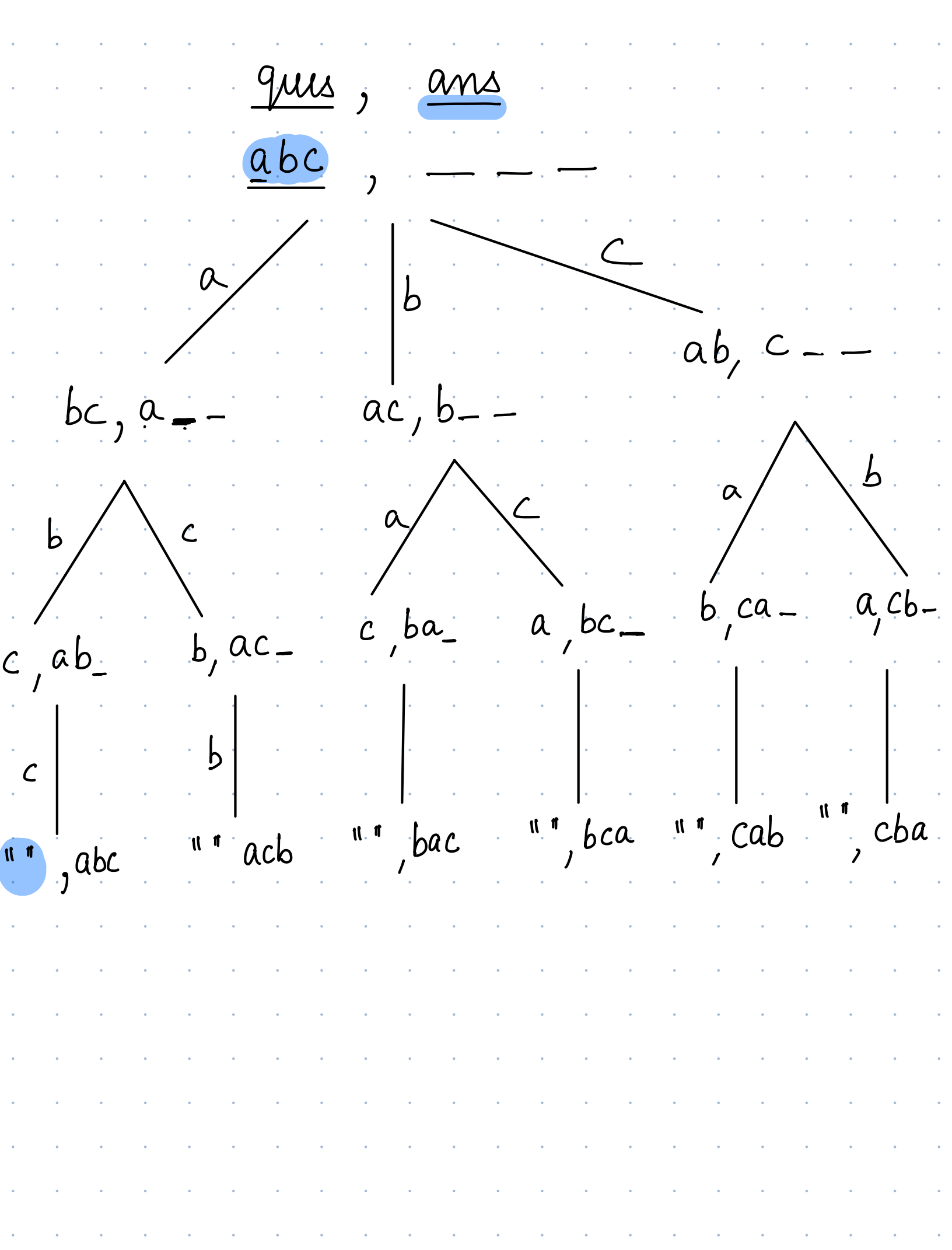
↓

ques.substring(0, i)
+ ques.substring(i+1, N)

ans + ques.charAt(i)

Code :

```
void permutation (String ques, String ans){
    if (ques.length() == 0) { print (ans), return;
    }
    for (int i=0; i < ques.length() i++){
        permutation (ques.substring (0, i) + ques.substring(i+1,N);
            ans+ques.charAt(i)) ;
    }
}
```


ques , ans
abc , ___
a
b
c
bc, a__
ac, b__
ab, c__
b
c
a
c
a
b
c, ab_
b, ac_
c, ba_
a, bc_
b, ca_
a, cb_
c
b
"", abc
"", acb
"", bac
"", bca
"", cab
"", cba