Lecture :- Backtracking-1

Agenda

- Rat in a maze
- Permutations
- Subsets.

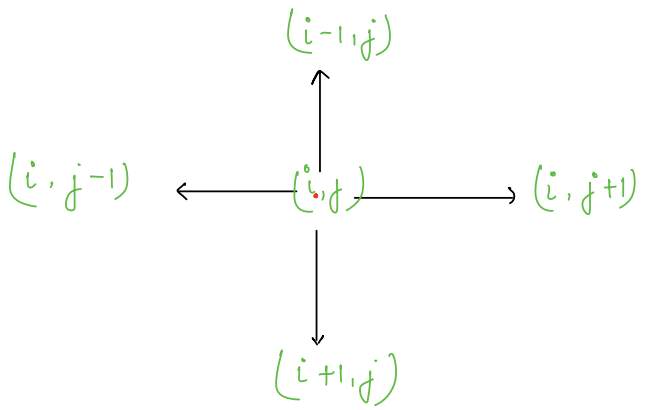
Qu1 **Rat in a maze**

Check if we can reach from start to end.

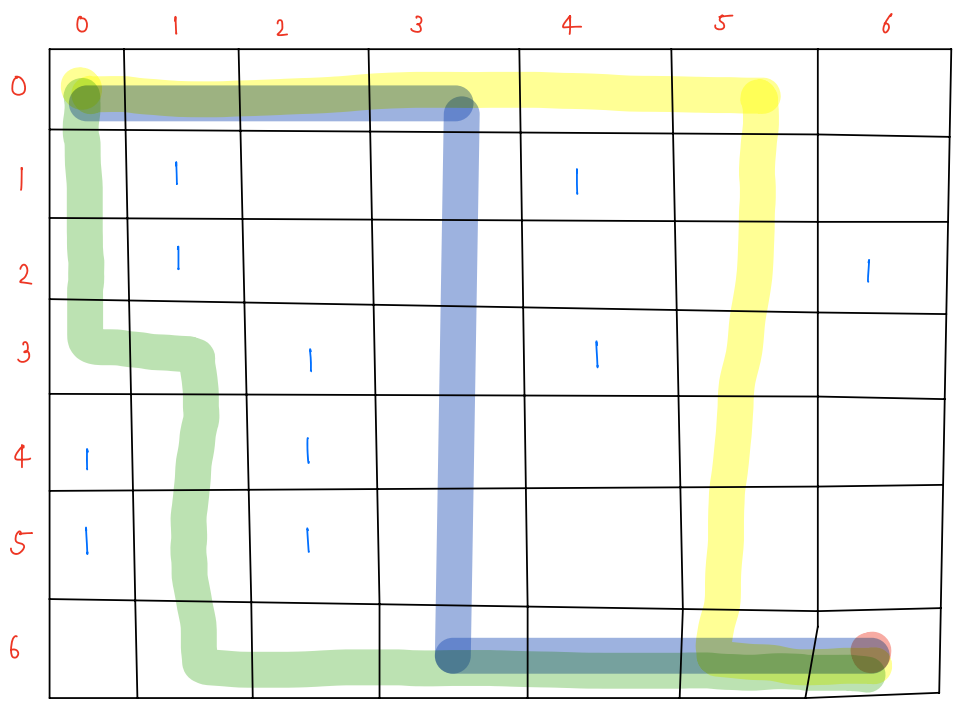
0 → empty

1 → blocked.

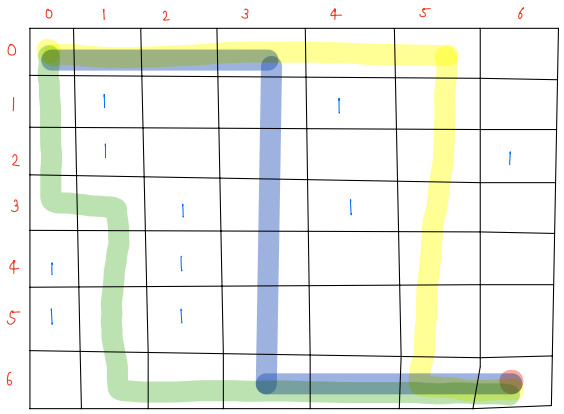
Moving directions:

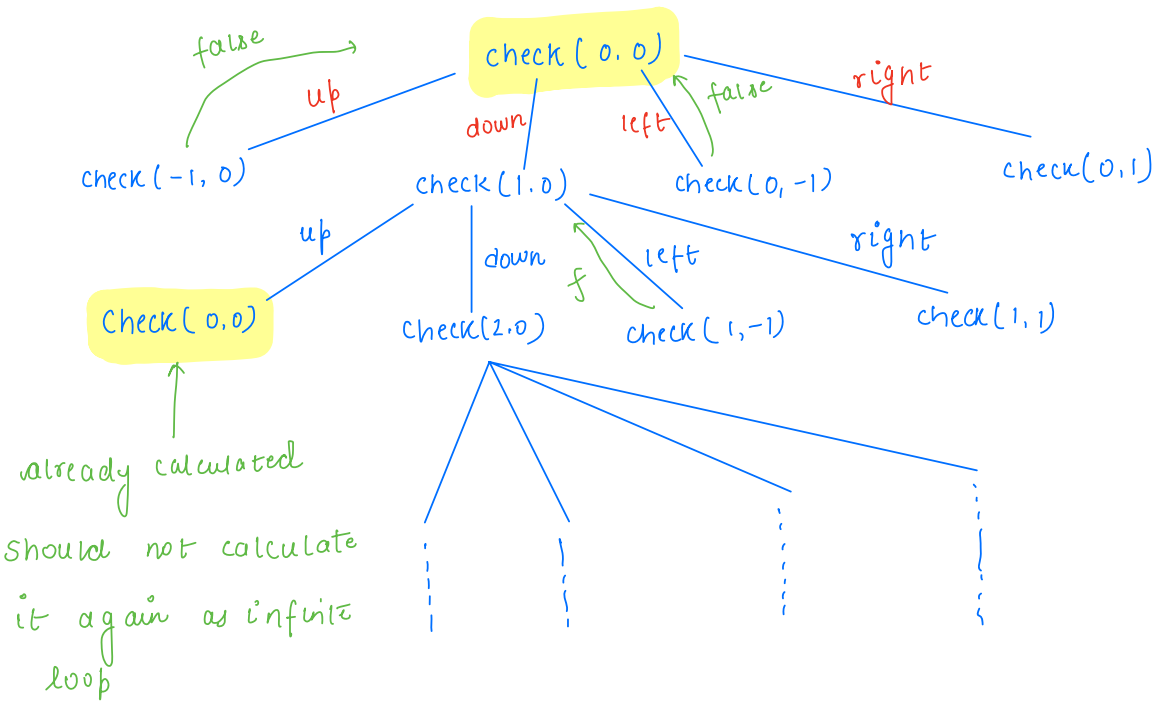


Example:



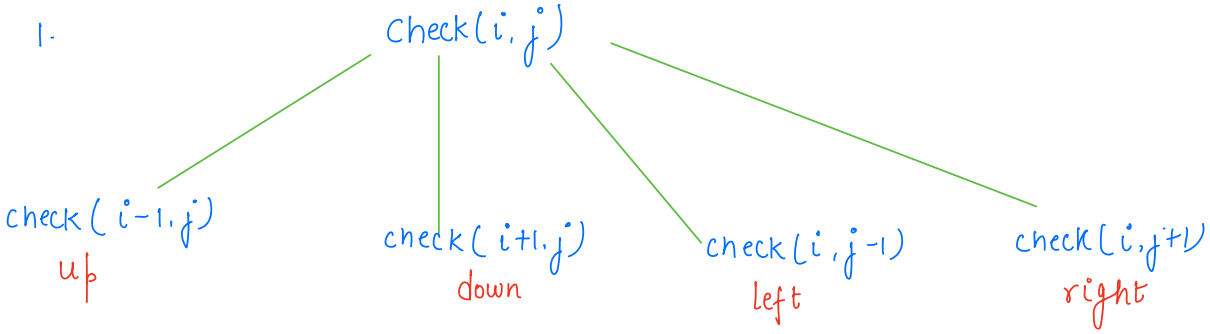
Idea:


0 1 2 3 4 5 6
0
1
2
3
4
5
6


check (0, 0)
false
up
down
left
false
right
check (-1, 0)
check (1, 0)
check (0, -1)
check (0, 1)
up
down
left
right
Check (0, 0)
Check (2, 0)
check (1, -1)
check (1, 1)
already calculated
Should not calculate it again as infinite loop

Observation

1. Check(i, j)



ans = up || down || left || right

2. Mark which cell is visited.

Code:

```
boolean check( mat[][], i, j, visited[][]) {
        if ( i == n-1  &&  j == m-1) {
            return true;
        }
        if( i<0  ||  j<0  ||  i>=n  ||  j>=m  ||
            mat[i][j] =1  ||  visited[i][j] == true) {
            return false ;
        }
        visited[i][j] = true;
        up =  check( mat, i-1, j , visited;
        down =  check( mat, i+1, j , visited;
        left =  check( mat, i, j-1, visited;
        right =  check( mat, i, j+1, visited;
        return  up || down || left || right;
}

boolean  check( mat[][]) {
        n = mat.length;
        m = mat[0].length;
        boolean[][] visited = new boolean[n][m];
        return  check( mat, 0, 0, visited);
}
```

* Can you solve this problem without using visited[] ?

0 → empty

1 → blocked

2 → visited.

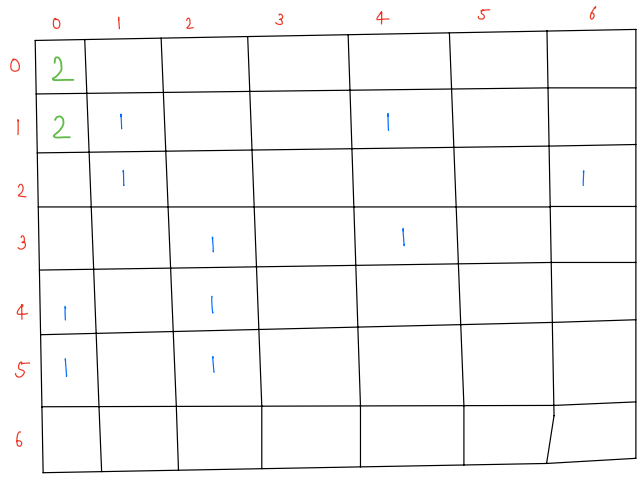
```
boolean check( mat[][], i, j ) {
        if( i == n-1 && j == m-1) {
            return true;
        }
        if( i<0 || j<0 || i>=n || j>=m ||
            mat[i][j] =1 || mat[i][j] == 2 ) {
            return false ;
        }
        mat[i][j] = 2;
        up = check( mat, i-1, j) ;
        down = check( mat, i+1, j) ;
        left = check( mat, i, j-1) ;
        right = check( mat, i, j+1) ;
        return up || down || left || right;
}
```

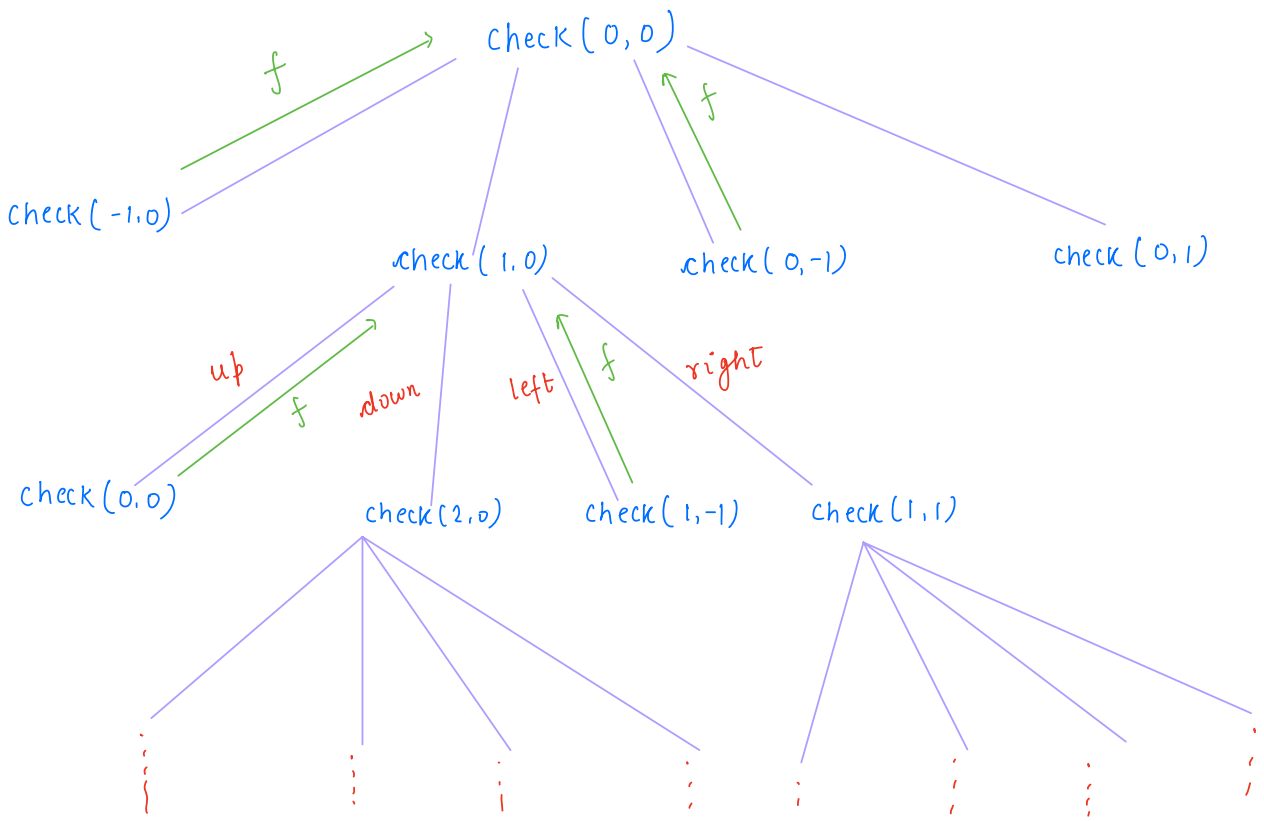
```
boolean check( mat[][]) {
    n = mat.length;
    m = mat[0].length;
    return check( mat, 0, 0);
}
```

TC: O(n*m)

SC: stack space of rec (H/W)

final dry run:





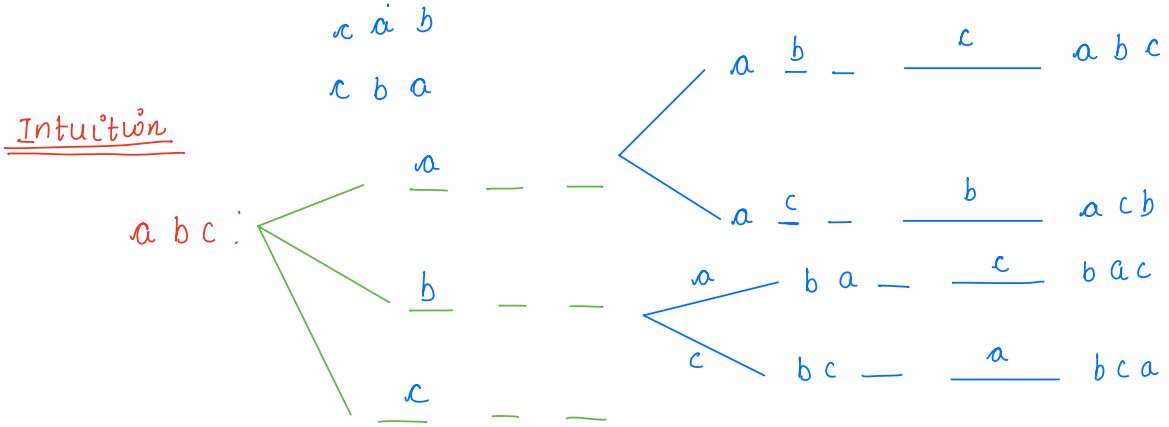
Qu Given a string with all distinct elements. Print all permutations.

Note: All characters are lowercase

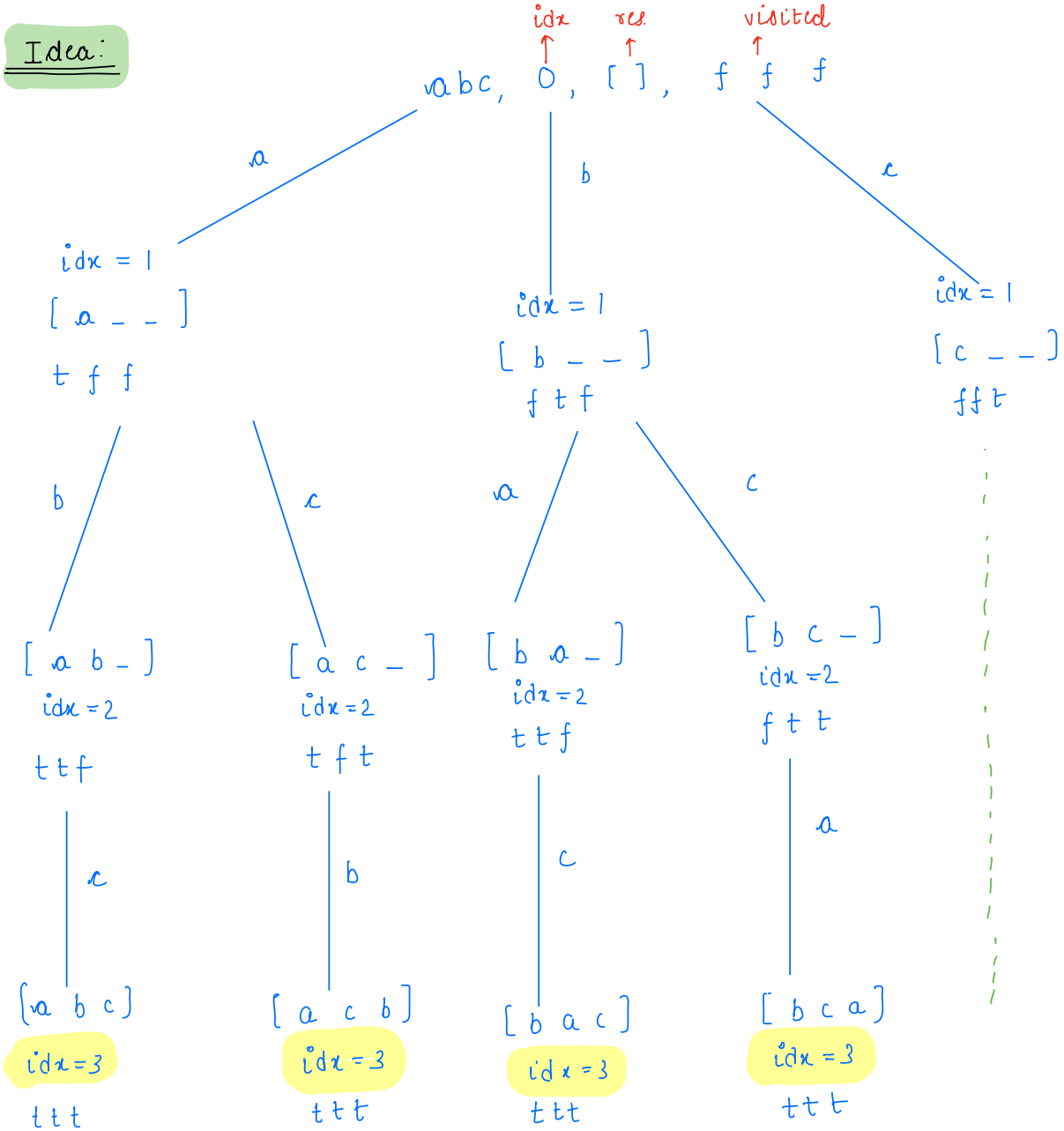
Example: abc : 3! permutations.

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

Intuition



Idea:


idx
res
visited
abc, 0, [ ], f f f
a
b
c
idx = 1
[a _ _]
t f f
idx = 1
[b _ _]
f t f
idx = 1
[c _ _]
f f t
b
c
a
c
[a b _]
idx = 2
t t f
[a c _]
idx = 2
t f t
[b a _]
idx = 2
t t f
[b c _]
idx = 2
f t t
c
b
c
a
[a b c]
idx = 3
t t t
[a c b]
idx = 3
t t t
[b a c]
idx = 3
t t t
[b c a]
idx = 3
t t t

Code:

```
void perm(str, ans[], idx, visited[]) {
    if (idx == str.length()) {
        print(ans);
        return;
    }
    for (i=0; i<str.length(); i++) {
        char ch = str.charAt(i);
        int j = ch - 97;
        if (visited[ j ] == false) {
            visited[ j ] = true;
            ans[idx] = ch;
            perm(str, ans, idx+1, visited);
            visited[ j ] = false;
        }
    }
}
```

Restoring the state of next recursive calls

TC: O(n!)

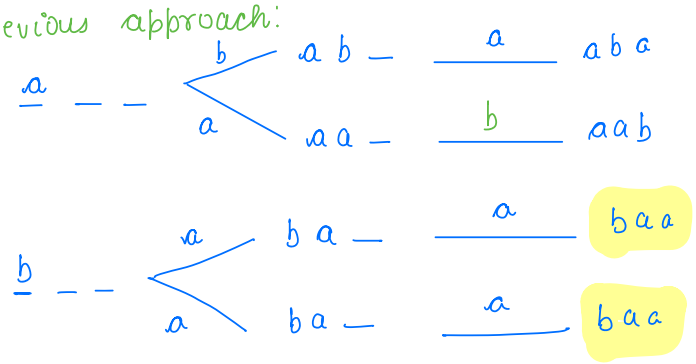
SC: O(n) — height of tree

Break: 8:15 – 8:26 AM

Qu. Print all unique permutations of string.

Ex: aba :

Previous approach:



aba: a b a

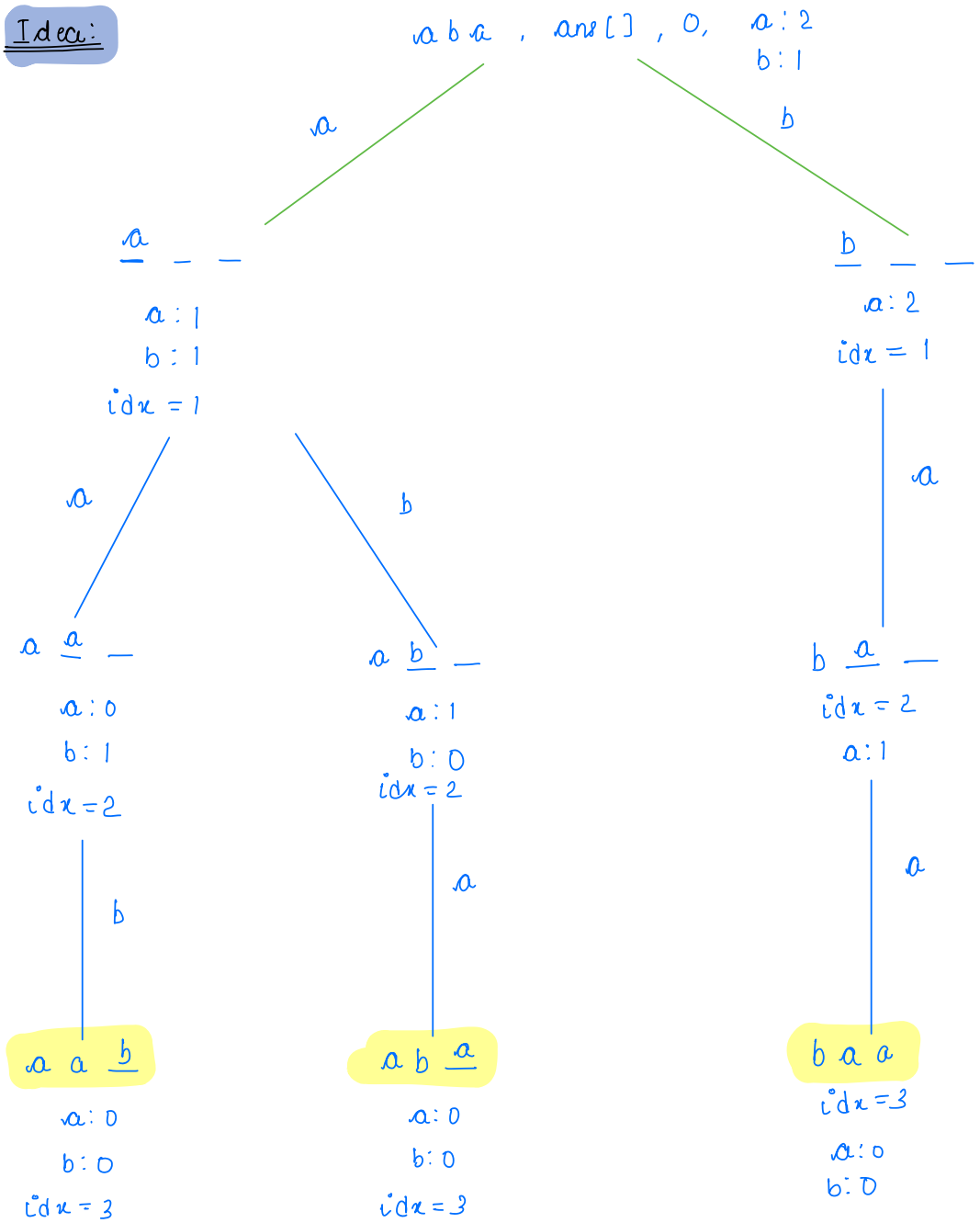
a a b

b a a

Unique perm = $\frac{n!}{(freq(a))! * (freq(b))!}$

aba : $\frac{3!}{2! * 1!} = \frac{6}{2} = 3$ Ans

Idea:


a b a , ans[] , 0, a : 2 b : 1
a
b
a _ _
a : 1
b : 1
idx = 1
b _ _
a : 2
idx = 1
a
a
b
a a _
a : 0
b : 1
idx = 2
a b _
a : 1
b : 0
idx = 2
b a _
idx = 2
a : 1
b
a
a
a a b
a : 0
b : 0
idx = 3
a b a
a : 0
b : 0
idx = 3
b a a
idx = 3
a : 0
b : 0

Code:

```
void perm(str, ans[], idx, freq[]) {
    if (idx == str.length()) {
        print(ans);
        return;
    }
    for (i=0; i<26; i++)        {
        if (freq[i] > 0) {
            freq[i]--;
            ans[idx] = (char) i + 'a';
            perm(str, ans, idx+1, freq);
            freq[i]++;
        }
    }
}
```

Restoring the state of next recursive calls

Qu: Print all subsets of an array.

Eg: A[] = [1 2 3] = 2^3 subsets

[ ]

1

2

3

1 2

1 3

2 3

1 2 3

Approach:

[1 2 3]

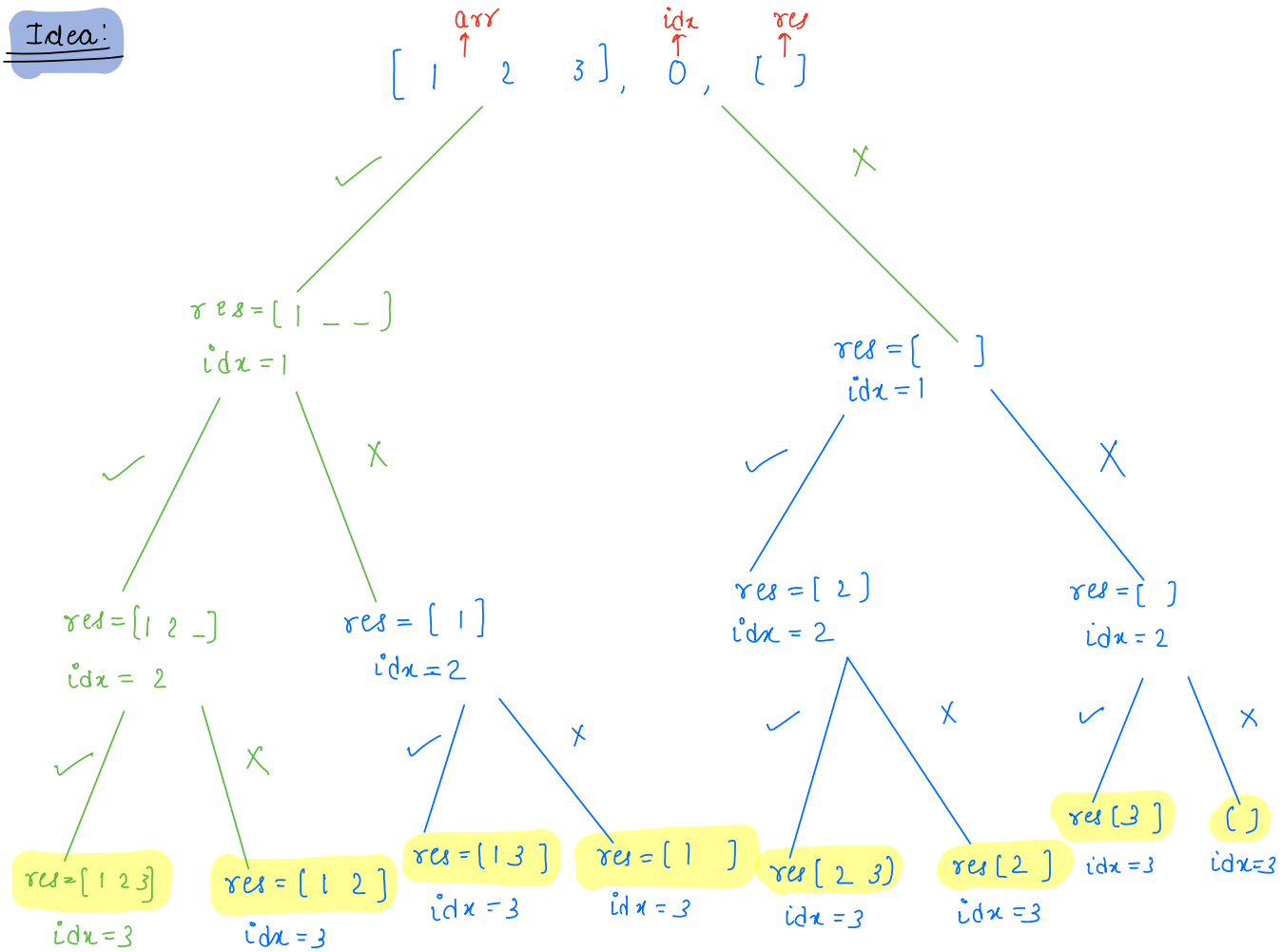
↑

1 can be part of some subsets — 1 | 1 2 | 1 3 | 1 2 3

1 cannot " " " " "

- [ ]
- 2
- 3
- 2 3

Idea:


arr
idx
res
[1 2 3], 0, [ ]
res = [1 _ _]
idx = 1
res = [ ]
idx = 1
res = [1 2 _]
idx = 2
res = [1]
idx = 2
res = [2]
idx = 2
res = [ ]
idx = 2
res = [1 2 3]
idx = 3
res = [1 2]
idx = 3
res = [1 3]
idx = 3
res = [1]
idx = 3
res [2 3)
idx = 3
res [2]
idx = 3
res [3]
idx = 3
[ ]
idx=3

Code:

```
void subsets(arr[], idx, List<Integer> res) {
    if (idx == arr.length) {
        print(res);
        return;
    }
    // Pick arr[idx]
    res.add(arr[idx]);
    subsets(arr, idx+1, res);
    res.remove(res.size()-1); ⟹ Restoring state

    // Don't pick arr[idx]
    subsets(arr, idx+1, res);
}
```

TC: $O(2^n)$

SC: $O(n)$

Thankyou