

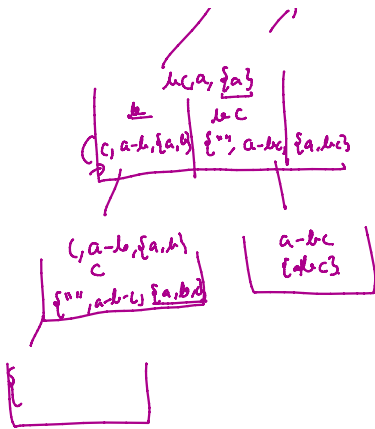
The diagram illustrates a recursive algorithm for string partitioning. The root node is "abc", which branches into "a", "ab", and "abc". Each branch leads to a node containing a list of characters and a recursive call to "add". The diagram is heavily annotated with red circles, arrows, and lines, indicating the flow of the algorithm and the state of the recursive calls. The final result is shown as a list of characters: "a, b, c, b, c, a, b, c".

`subtle(remain, bag + "-" + piece, copy);`
)

Diagram illustrating the recursive process of finding combinations that sum to 4 using the elements [1, 2, 3]:

- Initial state: `remain = 4, bag = "", copy = {}`
- Step 1: `subtle(3, "1", {1})` (using element 1)
- Step 2: `subtle(2, "12", {1, 2})` (using element 2)
- Step 3: `subtle(1, "123", {1, 2, 3})` (using element 3)
- Step 4: `subtle(0, "1234", {1, 2, 3, 4})` (base case reached)
- Step 5: `subtle(2, "13", {1, 3})` (backtrack, skip element 2)
- Step 6: `subtle(1, "134", {1, 3, 4})` (using element 4)
- Step 7: `subtle(0, "1345", {1, 3, 4, 5})` (base case reached)
- Step 8: `subtle(1, "23", {2, 3})` (backtrack, skip element 1)
- Step 9: `subtle(0, "234", {2, 3, 4})` (base case reached)
- Step 10: `subtle(1, "34", {3, 4})` (backtrack, skip element 1 and 2)
- Step 11: `subtle(0, "345", {3, 4, 5})` (base case reached)
- Step 12: `subtle(0, "45", {4, 5})` (base case reached)

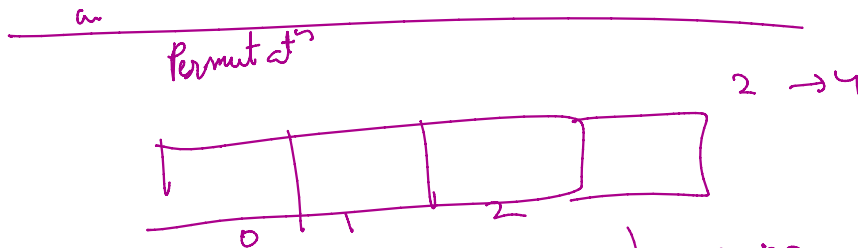
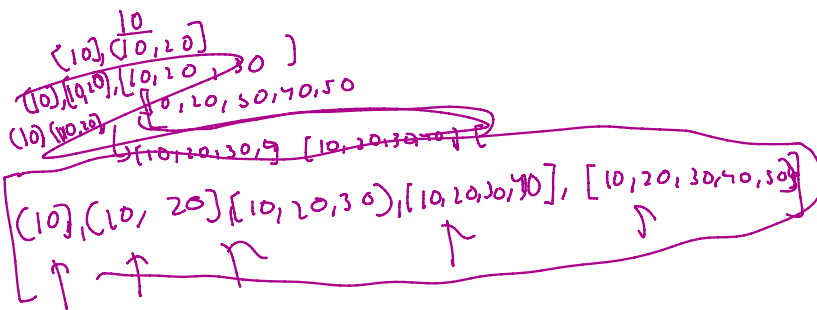
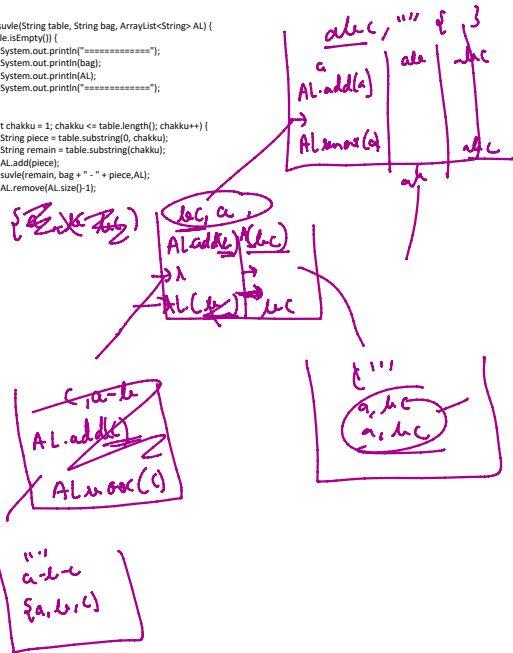
The final result is the list of combinations: `["1234", "1345", "234", "345", "45"]`.



```

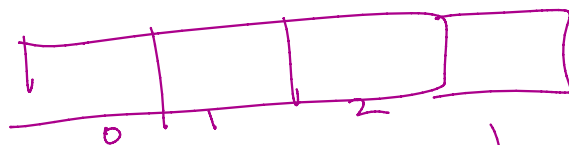
public static void solve(String table, String bag, ArrayList<String> AL) {
    if (table.isEmpty()) {
        System.out.println("*****");
        System.out.println(bag);
        System.out.println(AL);
        System.out.println("*****");
    }
    for (int chakku = 1; chakku <= table.length(); chakku++) {
        String piece = table.substring(0, chakku);
        String remain = table.substring(chakku);
        AL.add(piece);
        solve(remain, bag + "-" + piece, AL);
        AL.remove(AL.size() - 1);
    }
}

```



Permutatⁿ

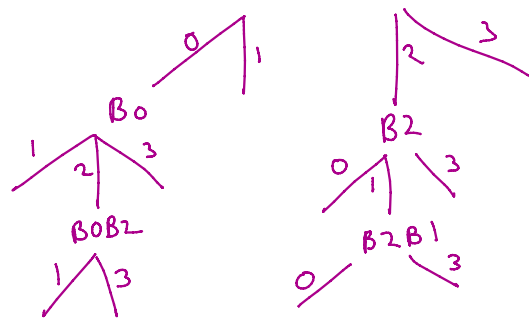
2 → 4



B0 B1	B1 B0	B2 B0	B3 B0
B0 B2	B1 B2	B2 B1	B3 B1
	B1 B3	B2 B3	B3 B2

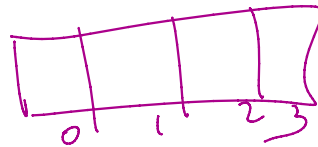
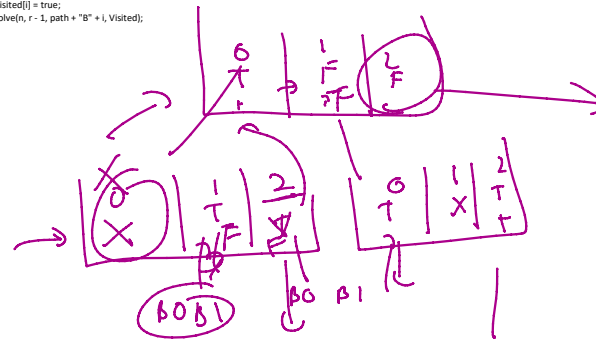
B0 B1	B1 B2	B2 B1	B3 B1
B0 B2	B1 B3	B2 B3	B3 B2
B0 B3			

3 1 4



```

public static void solve(int n, int r, String path, boolean[] Visited) {
    if (r == 0) {
        System.out.println(path);
        return;
    }
    for (int i = 0; i < n; i++) {
        // i the seat!
        if (Visited[i] == false) {
            Visited[i] = true;
            solve(n, r - 1, path + "B" + i, Visited);
        }
    }
}
  
```



0 1	1 2	2 3
0 2	1 3	
0 3		