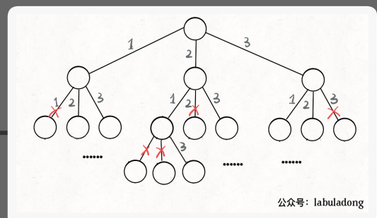
回溯算法

```
result = []
def backtrack(路径, 选择列表):
    if 满足结束条件:
        result.add(路径)
        return

    for 选择 in 选择列表:
        做选择
        backtrack(路径, 选择列表)
        撤销选择
```

一、全排列问题 46. 全排列

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        def backtrack(nums=nums,track=[]):
            if len(track) == len(nums):
                res.append(track[:])
            for i in range(0,len(nums)):
                if nums[i] in track:
                    continue
                track.append(nums[i])
                backtrack(nums,track)
                track.pop()

        res = []
        backtrack()
        return res
```



Subtopic

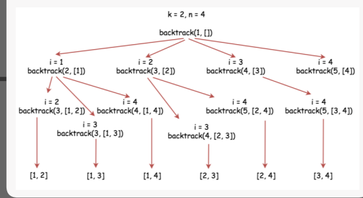
一、子集 78. Subsets

```
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        def dfs(nums, index, path, res):
            res.append(path)
            for i in range(index, len(nums)):
                dfs(nums, i+1, path+[nums[i]], res)

        res = []
        dfs(sorted(nums), 0, [], res)
        return res
```

二、组合 77. 组合

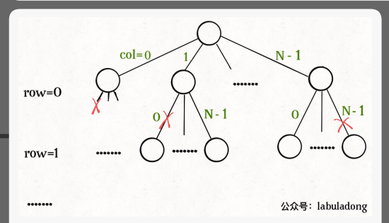
```
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        def backtrack(first = 1, curr = []):
            # if the combination is done
            if len(curr) == k:
                output.append(curr[:])
            for i in range(first, n + 1):
                # add i into the current combination
                curr.append(i)
                # use next integers to complete the combination
                backtrack(i + 1, curr)
                # backtrack
                curr.pop()

        output = []
        backtrack()
        return output
```

Subtopic



二、N 皇后问题

```
vector<vector<string>> res;

/* 输入棋盘边长 n，返回所有合法的放置 */
vector<vector<string>> solveNQueens(int n) {
    // '.' 表示空，'Q' 表示皇后，初始化空棋盘。
    vector<string> board(n, string(n, '.'));
    backtrack(board, 0);
    return res;
}

// 路径：board 中小于 row 的那些行都已经成功放置了皇后
// 选择列表：第 row 行的所有列都是放置皇后的选择
// 结束条件：row 超过 board 的最后一行
void backtrack(vector<string>& board, int row) {
    // 触发结束条件
    if (row == board.size()) {
        res.push_back(board);
        return;
    }

    int n = board[row].size();
    for (int col = 0; col < n; col++) {
        // 排除不合法选择
        if (!isValid(board, row, col))
            continue;
        // 做选择
        board[row][col] = 'Q';
        // 进入下一行决策
        backtrack(board, row + 1);
        // 撤销选择
        board[row][col] = '.';
    }
}
```



Subtopic

22. 括号生成

```
class Solution:
    def generateParenthesis(self, n: int) -> List[str]:
        ans = []
        def backtrack(S, left, right):
            if len(S) == 2 * n:
                ans.append(''.join(S))
                return
            if left < n:
                S.append('(')
                backtrack(S, left+1, right)
                S.pop()
            if right < left:
                S.append(')')
                backtrack(S, left, right+1)
                S.pop()

        backtrack([], 0, 0)
        return ans
```