【第十五课】深搜与广搜——深搜系列题目

1.看回放时，请把鼠标往后拖30分钟，因为提前30分钟 在调试电脑推流，不好意思啊。

2.下周一的刷题课，挪到下周二了哦！

130. 被围绕的区域

1、题目要把岛屿沉了，变成海水。把 X 看作海水，把 O
看作陆地，被海水包围的区域就是岛屿。没被海水包围的陆地，与边界有连通，不是岛屿。

2、判断是否为岛屿,凡是与边界有联系的 O，标记为 NO，表示非岛屿。

3、最后非岛屿被标记为 NO，最后将它们恢复为 O，其余的 O，变成X。

```
const solve = (board) => {
  const m = board.length;
  if (m == 0) return;          // [] 情况的特判
  const n = board[0].length;
  const dfs = (i, j) => {  // 用深搜完成标记的过程
    if (i < 0 || i == m || j < 0 || j == n) return; // 越界了
    if (board[i][j] == 'O') { // 遇到O，染为NO
      board[i][j] = 'NO';
      dfs(i + 1, j);           // 对四个方向的邻居进行dfs
      dfs(i - 1, j);
      dfs(i, j + 1);
      dfs(i, j - 1);
    }
  };
  for (let i = 0; i < m; i++) {
    for (let j = 0; j < n; j++) {
      if (i == 0 || i == m - 1 || j == 0 || j == n - 1) {
        if (board[i][j] == 'O') dfs(i, j); // 从最外层的O，开始DFS
      }
    }
  }
  for (let i = 0; i < m; i++) {
    for (let j = 0; j < n; j++) {
      if (board[i][j] === 'NO') board[i][j] = 'O';     // 恢复为大O
      else if (board[i][j] === 'O') board[i][j] = 'X'; // 小O变为X
    }
  }
};
```

494. 目标和

1、数组 nums的每个元素都可以添加符号 + 或 -，因此每个元素有 2种添加符号的方法，n*n 个数共有2的n次幂种添加符号的方法。

2、当 n个元素都添加符号之后，即得到一种表达式，如果表达式的结果等于目标数target，则该表达式即为符合要求的表达式。

3、可以使用回溯的方法遍历所有的表达式，回溯过程中维护一个计数器count，当遇到一种表达式的结果等于目标数 target 时，将 count 的值加 1。

4、遍历完所有的表达式之后，即可得到结果等于目标数 *target* 的表达式的数目。

```
var findTargetSumWays = function(nums, target) {
    let count = 0;
    const backtrack = (nums, target, index, sum) => {
        if (index === nums.length) {
            if (sum === target) {
                count++;
            }
        } else {
            backtrack(nums, target, index + 1, sum + nums[index]);
            backtrack(nums, target, index + 1, sum - nums[index]);
        }
    }

    backtrack(nums, target, 0, 0);
    return count;
};
```

473. 火柴拼正方形

深度优先搜索加回溯；

1、将给定的火柴进行分组，将他们分成四组，每一根火柴恰好属于其中一组。

2、每一组火柴的长度之和都相同，等于所有火柴长度之和的四分之一。

3、用深度优先搜索将所有的分组情况都枚举出来，对于每一种情况，判断一下是否满足刚才的两种情况。

4、依次对每一根火柴进行搜索，当搜索出第 I 根火柴的时候，我们可以把它放到四组中的任意一种。

5、对于每一根火柴的放置方法，我破门继续对第 i + 1跟火柴进行递归搜索。

6、当我们搜索完全部的N根火柴后，在判断每一组火柴的长度之和是否都相同。

```
/**
 * @param {number[]} nums
 * @return {boolean}
 */
var makesquare = function (nums) {
    if (nums === null || nums.length == 0) return false;
    // 判断数组是否存在且长度不为0
    let allLen = 0;
```

```
    for (let item of nums) {
        allLen += item;     //计算总长度
    }
    if (allLen % 4 !== 0) return false;
    // 判断总长度是否可以刚好围成正方形
    const sideLen = allLen / 4;
    // 计算边长
    nums.sort((a, b) => b - a);
    // 可以将火柴数组从大到小排序，方便之后优化
    let sideList = new Array(4);
    for (let i = 0; i < sideList.length; i++) {
        sideList[i] = 0;
    }
    // 定义边长数组sideList并赋初值
    const dfs = (index) => {
        if (index == nums.length) {
            // 结束条件，当index === nums.length时，判断四条边长是否相等；
            return (
                sideList[0] === sideList[1] &&
                sideList[1] === sideList[2] &&
                sideList[2] === sideList[3]
            );
        }
        const targetLen = nums[index];
        // 当前正处理的火柴；
        if (targetLen > sideLen) {
            return false;
        }
        // 因为刚才对火柴进行了排序，所以如果有火柴targetLen大于我们所计算的边
长sideLen，则返回
        for (let i = 0; i < 4; i++) {
            if (sideList[i] + targetLen <= sideLen) {
                sideList[i] += targetLen;
                if (dfs(index + 1)) {
                    return true;
                }
                sideList[i] -= targetLen;
                // 这一步是回溯，先加上，再去递归判断下一步，如果false，则减去
            }
        }
        return false;
        // 最后一定记得返回结束
    };
```

```
    return dfs(0);
};
```

39. 组合总和

1、如果我们将整个搜索过程用一个树来表达，
每次的搜索都会延伸出两个分叉，直到递归的终止条件，这样我们就能不重复且不遗漏地找到所有可行解：

2、搜索回溯的过程一定存在一些优秀的剪枝方法来使得程序运行得更快

```
var combinationSum = function(candidates, target) {
    const ans = [];
    const dfs = (target, combine, idx) => {
        if (idx === candidates.length) {
            return;
        }
        if (target === 0) {
            ans.push(combine);
            return;
        }
        // 直接跳过
        dfs(target, combine, idx + 1);
        // 选择当前数
        if (target - candidates[idx] >= 0) {
            dfs(target - candidates[idx], [...combine, candidates[idx]], idx);
        }
    }

    dfs(target, [], 0);
    return ans;
};
```

51. N 皇后

皇后彼此不相互攻击，就是让皇后横直斜走，两个皇后不能在同一行，同一列，以及同一条斜线上。

1、每一行，选出一个格子，置为Q，一行行地往下选择，第一行有四种选择

2、在选择下一行的皇后的时候，为了避免列的冲突，有三种选择：

3、继续选下去，可能会遇到对角线的冲突，继续选下去没有意义，得出不合法的解，需要回溯。

```
 const solveNQueens = (n) => {
  const board = new Array(n);
  for (let i = 0; i < n; i++) {  // 棋盘的初始化
    board[i] = new Array(n).fill('.');
  }
```

```
  const cols = new Set();  // 列集，记录出现过皇后的列
  const diag1 = new Set(); // 正对角线集
  const diag2 = new Set(); // 反对角线集
  const res = [];

  const helper = (row) => {
    if (row == n) {
      const stringsBoard = board.slice();
      for (let i = 0; i < n; i++) {
        stringsBoard[i] = stringsBoard[i].join('');
      }
      res.push(stringsBoard);
      return;
    }
    for (let col = 0; col < n; col++) {
      // 如果当前点的所在的列，所在的对角线都没有皇后，即可选择，否则，跳过
      if (!cols.has(col) && !diag1.has(row + col) && !diag2.has(row - col)) {
        board[row][col] = 'Q';  // 放置皇后
        cols.add(col);          // 记录放了皇后的列
        diag1.add(row + col);   // 记录放了皇后的正对角线
        diag2.add(row - col);   // 记录放了皇后的负对角线
        helper(row + 1);
        board[row][col] = '.';  // 撤销该点的皇后
        cols.delete(col);       // 对应的记录也删一下
        diag1.delete(row + col);
        diag2.delete(row - col);
      }
    }
  };
  helper(0);
  return res;
};
```