78. Subsets

Created	@April 4, 2021 10:34 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/subsets/
Importance	
Tag	Backtrack DFS NEET
Video	https://www.youtube.com/watch?v=rtFHxiQAICA&list=PLH8TFsY0qnE2R9kf_9vahNY6pG9601z_4&index=59

Given an integer array `nums` of **unique** elements, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

```
Input: nums = [1,2,3]
Output: [[],[1],[2],[1,2],[3],[1,3],[2,3],[1,2,3]]
```

Example 2:

```
Input: nums = [0]
Output: [[],[0]]
```

Constraints:

- `1 <= nums.length <= 10`
- `-10 <= nums[i] <= 10`
- All the numbers of `nums` are **unique**.

Solution 1

Backtrack

Ref: 16 九章算法班2020版 subsets-version-1_1.mp4

```
# 九章算法
class Solution:
    def subsets(self, nums: List[int]) -> List[List[int]]:
        results = []
        if not nums:
            return results

        # 本题有没有sort都可以
        # 如果要求subset必须是non-descending，那就必须先排序
        nums.sort()
        self.dfs(nums, 0, [], results)
        return results

    def dfs(self, nums, index, subset, results):
        if index == len(nums):
            results.append(list(subset))
            return

        subset.append(nums[index])
        self.dfs(nums, index + 1, subset, results)

        subset.pop()
        self.dfs(nums, index + 1, subset, results)
```

90. Subsets II

Created	@April 5, 2021 1:59 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/subsets-ii/
Importance	
Tag	Backtrack Recursion
Video	https://www.youtube.com/watch?v=rtFHxiQAICA&list=PLH8TFsY0qnE2R9kf_9vahNY6pG9601z_4&index=59

Given an integer array `nums` that may contain duplicates, return *all possible subsets (the power set)*.

The solution set **must not** contain duplicate subsets. Return the solution in **any order**.

Example 1:

```
Input: nums = [1,2,2]
Output: [[],[1],[1,2],[1,2,2],[2],[2,2]]
```

Example 2:

```
Input: nums = [0]
Output: [[],[0]]
```

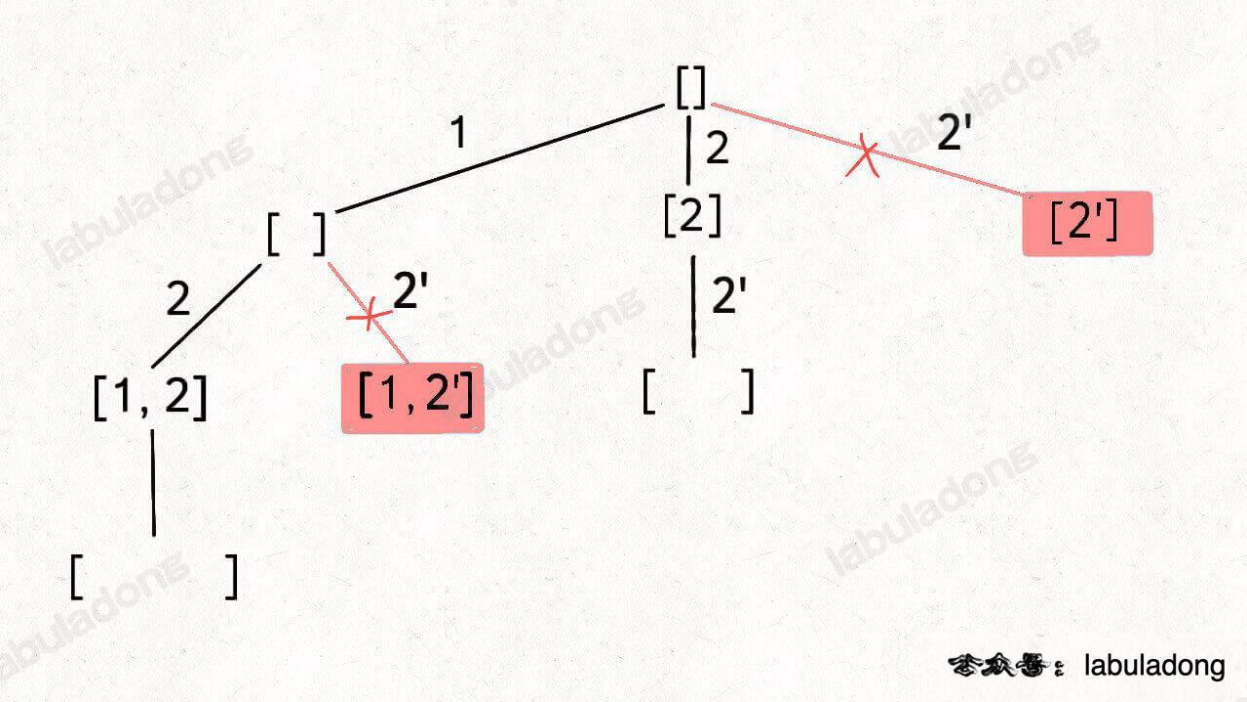
Constraints:

- `1 <= nums.length <= 10`
- `10 <= nums[i] <= 10`

Solution

best solution;

所以我们需要进行剪枝，如果一个节点有多条值相同的树枝相邻，则只遍历第一条，剩下的都剪掉，不要去遍历：



体现在代码上，需要先进行排序，让相同的元素靠在一起，如果发现 `nums[i] == nums[i-1]`，则跳过

```
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        results = []
        if not nums:
            return results

        nums.sort()
        self.dfs(nums, 0, [], results)
        return results

    def dfs(self, nums, startIndex, subset, results):
        results.append(list(subset))
        for i in range(startIndex, len(nums)):
            if i > startIndex and nums[i] == nums[i - 1]:
                continue
            subset.append(nums[i])
            self.dfs(nums, i + 1, subset, results)
            subset.pop()
```

```
List<List<Integer>> res = new LinkedList<>();
LinkedList<Integer> track = new LinkedList<>();

public List<List<Integer>> subsetsWithDup(int[] nums) {
    // 先排序，让相同的元素靠在一起
    Arrays.sort(nums);
    backtrack(nums, 0);
    return res;
}

void backtrack(int[] nums, int start) {
    // 前序位置，每个节点的值都是一个子集
    res.add(new LinkedList<>(track));

    for (int i = start; i < nums.length; i++) {
        // 剪枝逻辑，值相同的相邻树枝，只遍历第一条
        if (i > start && nums[i] == nums[i - 1]) {
            continue;
        }
        track.addLast(nums[i]);
        backtrack(nums, i + 1);
        track.removeLast();
    }
}
```

better than the second one

```
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        results = []
        if not nums or len(nums) == 0:
            return results

        nums.sort()
        visited = [False] * len(nums)
        self.dfs(nums, 0, [], results, visited)
        return results

    def dfs(self, nums, startIndex, subset, results, visited):
        results.append(list(subset))
        for i in range(startIndex, len(nums)):
            if i != 0 and nums[i] == nums[i - 1] and visited[i - 1] == False:
                continue
            subset.append(nums[i])
            visited[i] = True
            self.dfs(nums, i + 1, subset, results, visited)
            visited[i] = False
            subset.pop()
```

Ref: 16 九章算法班2020版 subsets-ii_1

```
class Solution:
    def subsetsWithDup(self, nums: List[int]) -> List[List[int]]:
        results = []
        if not nums or len(nums) == 0:
            return results

        nums.sort()
        self.dfs(nums, 0, [], results)
        return results

    def dfs(self, nums, startIndex, subset, results):
        results.append(list(subset))
        for i in range(startIndex, len(nums)):
            if i != 0 and nums[i] == nums[i - 1] and i > startIndex:
                continue
            subset.append(nums[i])
            self.dfs(nums, i + 1, subset, results)
            subset.pop()
```

77. Combinations

Created	@July 9, 2021 12:18 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/combinations/
Importance	*****
Tag	Backtrack
Video	

Given two integers `n` and `k`, return *all possible combinations of* `k` *numbers out of the range* `[1, n]`.

You may return the answer in **any order**.

Example 1:

```
Input: n = 4, k = 2
Output:
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

Example 2:

```
Input: n = 1, k = 1
Output: [[1]]
```

Constraints:

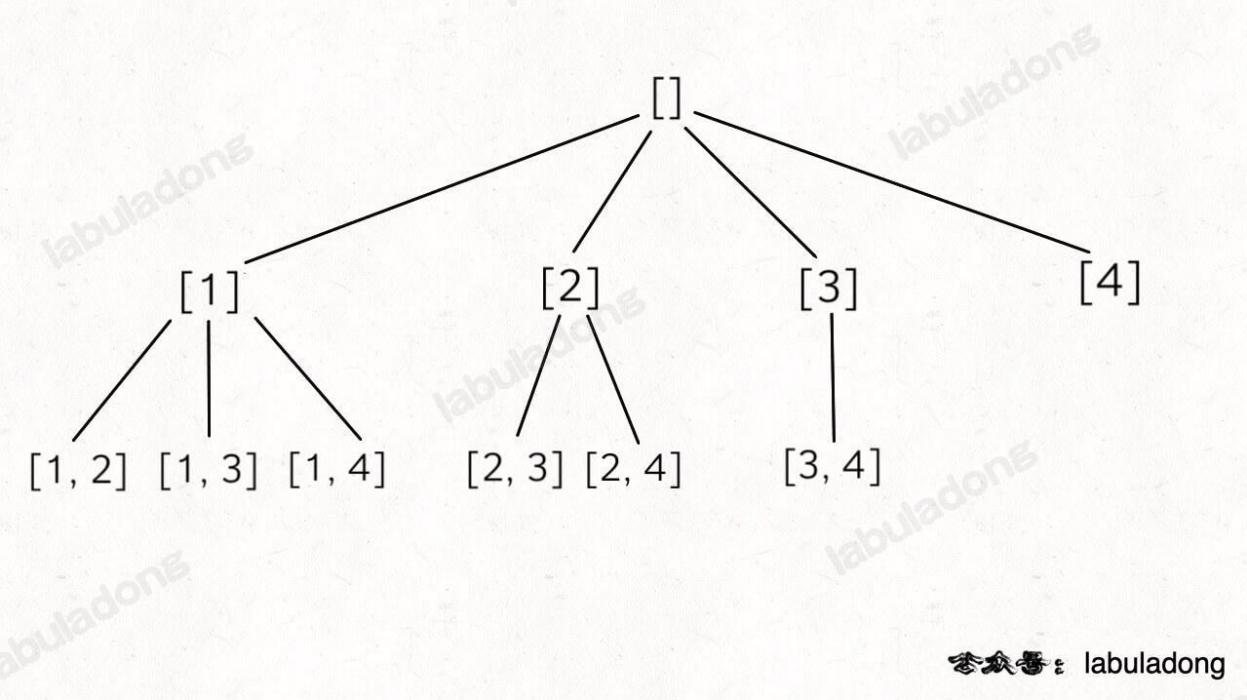
- `1 <= n <= 20`

- `1 <= k <= n`

Solution

PS：这道题在《算法小抄》 的第 293 页。

这也是典型的回溯算法，`k` 限制了树的高度，`n` 限制了树的宽度，继续套我们以前讲过的 回溯算法模板框架 就行了：




```
class Solution:
    def combine(self, n: int, k: int) -> List[List[int]]:
        res = []
        self.backtrack(n, 1, k, [], res)
        return res

    def backtrack(self, n, start_index, k, subset, res):
        # base case
        if k == len(subset):
            res.append(list(subset))
            return

        # backtrack
        for i in range(start_index, n + 1):
            subset.append(i)
```

```
            self.backtrack(n, i + 1, k, subset, res)
            subset.pop()
```

```
// labuladong
List<List<Integer>> res = new LinkedList<>();
// 记录回溯算法的递归路径
LinkedList<Integer> track = new LinkedList<>();

// 主函数
public List<List<Integer>> combine(int n, int k) {
    backtrack(1, n, k);
    return res;
}

void backtrack(int start, int n, int k) {
    // base case
    if (k == track.size()) {
        // 遍历到了第 k 层，收集当前节点的值
        res.add(new LinkedList<>(track));
        return;
    }

    // 回溯算法标准框架
    for (int i = start; i <= n; i++) {
        // 选择
        track.addLast(i);
        // 通过 start 参数控制树枝的遍历，避免产生重复的子集
        backtrack(i + 1, n, k);
        // 撤销选择
        track.removeLast();
    }
}
```

39. Combination Sum

Created	@July 17, 2020 3:04 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/combination-sum/
Importance	*****
Tag	Backtrack DFS NEET
Video	https://www.youtube.com/watch?v=lJ0qw4vr0_w&list=PL2rWx9cCzU85RX9NeRMVUV_kgI4YGKURD&index=3

Given an array of **distinct** integers `candidates` and a target integer `target`, return *a list of all **unique combinations** of* `candidates` *where the chosen numbers sum to* `target`. You may return the combinations in **any order**.

The **same** number may be chosen from `candidates` an **unlimited number of times**. Two combinations are unique if the frequency of at least one of the chosen numbers is different.

The test cases are generated such that the number of unique combinations that sum up to `target` is less than `150` combinations for the given input.

Example 1:

```
Input: candidates = [2,3,6,7], target = 7
Output: [[2,2,3],[7]]
Explanation:
2 and 3 are candidates, and 2 + 2 + 3 = 7. Note that 2 can be used multiple times.
7 is a candidate, and 7 = 7.
These are the only two combinations.
```

Example 2:

```
Input: candidates = [2,3,5], target = 8
Output: [[2,2,2,2],[2,3,3],[3,5]]
```

Example 3:

```
Input: candidates = [2], target = 1
Output: []
```

Constraints:

- `1 <= candidates.length <= 30`
- `2 <= candidates[i] <= 40`
- All elements of `candidates` are **distinct**.
- `1 <= target <= 500`

Solution

你需要先看前文 回溯算法详解 和 回溯算法团灭子集、排列、组合问题，然后看这道题就很简单了，无非是回溯算法的运用而已。

这道题的关键在于 `candidates` 中的元素可以复用多次，体现在代码中是下面这段：

```
void backtrack(int[] candidates, int start, int target, int sum) {
    // 回溯算法框架
    for (int i = start; i < candidates.length; i++) {
        // 选择 candidates[i]
        backtrack(candidates, i, target, sum);
        // 撤销选择 candidates[i]
    }
}
// 详细解析参见：
// https://labuladong.github.io/article/?qno=39
```

对比 回溯算法团灭子集、排列、组合问题 中不能重复使用元素的标准组合问题：

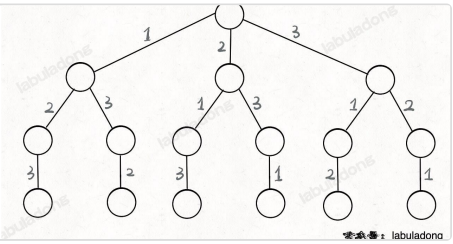
```
void backtrack(int[] candidates, int start, int target, int sum) {
    // 回溯算法框架
    for (int i = start; i < candidates.length; i++) {
        // 选择 candidates[i]
        backtrack(candidates, i + 1, target, sum);
        // 撤销选择 candidates[i]
    }
}
// 详细解析参见：
// https://labuladong.github.io/article/?qno=39
```

```
class Solution:
    def combinationSum(self, candidates: List[int], target: int) -> List[List[int]]:
        results = []
        self.dfs(candidates, target, 0, [], results)
        return results

    def dfs(self, candidates, target, start, combination, results):
        if target < 0:
            return

        if target == 0:
            return results.append(list(combination))

        for i in range(start, len(candidates)):
            combination.append(candidates[i])
            self.dfs(candidates, target-candidates[i], i, combination, results)
            combination.pop()
```

回溯算法解题套路框架

通知： 数据结构精品课 已更新到 V2.0，第 14 期打卡训练营开始报名。 读完本文，你不仅学会了算法套路，还可以顺便解决如下题目： ---- 这篇文章是很久之前的一篇 回溯算法详解 的进阶版，之前那篇不够清楚，就不必看

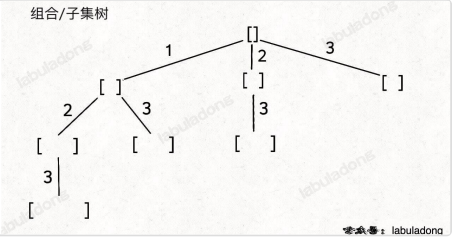
https://labuladong.github.io/algo/4/31/104/



回溯算法秒杀所有排列-组合-子集问题

通知： 数据结构精品课 已更新到 V2.0，第 14 期打卡训练营开始报名。 读完本文，你不仅学会了算法套路，还可以顺便解决如下题目： ---- 本文有视频版： 回溯算法秒杀所有排列/组合/子集问题 虽然排列、组合、子集系列问

https://labuladong.github.io/algo/4/31/106/



40. Combination Sum II

Created	@July 19, 2020 8:39 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/combination-sum-ii/
Importance	*****
Tag	Backtrack
Video	

Given a collection of candidate numbers (`candidates`) and a target number (`target`), find all unique combinations in `candidates` where the candidate numbers sum to `target`.

Each number in `candidates` may only be used **once** in the combination.

Note: The solution set must not contain duplicate combinations.

Example 1:

```
Input: candidates = [10,1,2,7,6,1,5], target = 8
Output:
[
[1,1,6],
[1,2,5],
[1,7],
[2,6]
]
```

Example 2:

```
Input: candidates = [2,5,2,1,2], target = 5
Output:
[
[1,2,2],
[5]
]
```

Constraints:

- `1 <= candidates.length <= 100`
- `1 <= candidates[i] <= 50`
- `1 <= target <= 30`

Solution

```python
class Solution:
    def combinationSum2(self, candidates: List[int], target: int) -> List[List[int]]:
        results = []
        subset = []

        if not candidates:
            return results

        candidates.sort()
        self.dfs(candidates, target, 0, subset, results)

        return results

    def dfs(self, candidates, target, index, subset, results):
        if target == 0:
            results.append(list(subset))
        elif target > 0:
            for i in range(index, len(candidates)):
                if i != index and candidates[i] == candidates[i - 1]:
                    continue
                subset.append(candidates[i])
                self.dfs(candidates, target - candidates[i], i + 1, subset, results)
                subset.pop()
```

```java
class Solution {
    public List<List<Integer>> combinationSum2(int[] candidates, int target) {
        List<List<Integer>> results = new ArrayList<List<Integer>>();
        List<Integer> curList = new ArrayList<Integer>();
        if (candidates == null) return results;
        Arrays.sort(candidates);
        helper(candidates, target, 0, curList, results);
        return results;
    }

    public void helper(int[] candidates, int target, int index, List<Integer> curList, List<List<Integer>> results) {
        if (target == 0) {
            results.add(new ArrayList<Integer>(curList));
        } else if (target > 0) {
            for (int i = index; i < candidates.length; i++) {
                if (i != index && candidates[i] == candidates[i-1]) continue;
                curList.add(candidates[i]);
                helper(candidates, target-candidates[i], i+1, curList, results);
                curList.remove(curList.size()-1);
            }
        }
    }
}
```

对比子集问题的解法，只要额外用一个 `trackSum` 变量记录回溯路径上的元素和，然后将 base case 改一改即可解决这道题：

```java
List<List<Integer>> res = new LinkedList<>();
// 记录回溯的路径
LinkedList<Integer> track = new LinkedList<>();
// 记录 track 中的元素之和
int trackSum = 0;
```

```
public List<List<Integer>> combinationSum2(int[] candidates, int target) {
    if (candidates.length == 0) {
        return res;
    }
    // 先排序，让相同的元素靠在一起
    Arrays.sort(candidates);
    backtrack(candidates, 0, target);
    return res;
}

// 回溯算法主函数
void backtrack(int[] nums, int start, int target) {
    // base case，达到目标和，找到符合条件的组合
    if (trackSum == target) {
        res.add(new LinkedList<>(track));
        return;
    }
    // base case，超过目标和，直接结束
    if (trackSum > target) {
        return;
    }

    // 回溯算法标准框架
    for (int i = start; i < nums.length; i++) {
        // 剪枝逻辑，值相同的树枝，只遍历第一条
        if (i > start && nums[i] == nums[i - 1]) {
            continue;
        }
        // 做选择
        track.add(nums[i]);
        trackSum += nums[i];
        // 递归遍历下一层回溯树
        backtrack(nums, i + 1, target);
        // 撤销选择
        track.removeLast();
        trackSum -= nums[i];
    }
}
```

46. Permutations

Created	@July 19, 2020 11:06 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/permutations/
Importance	*****
Tag	Backtrack
Video	

Given an array `nums` of distinct integers, return *all the possible permutations*. You can return the answer in **any order**.

Example 1:

```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

Example 2:

```
Input: nums = [0,1]
Output: [[0,1],[1,0]]
```

Example 3:

```
Input: nums = [1]
Output: [[1]]
```

Constraints:

- `1 <= nums.length <= 6`
- `10 <= nums[i] <= 10`

- All the integers of `nums` are **unique**.

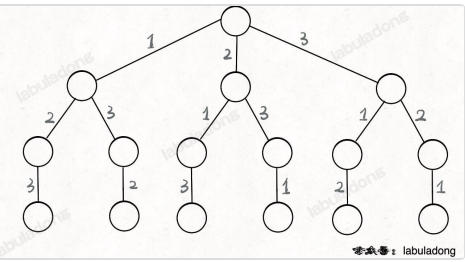
Solution

```
class Solution:
    def permute(self, nums: List[int]) -> List[List[int]]:
        if not nums:
            return []

        results = []
        visited = [False] * len(nums)
        self.dfs(nums, [], visited, results)
        return results

    def dfs(self, nums, subset, visited, results):
        if len(nums) == len(subset):
            results.append(list(subset))
            return

        for i in range(len(nums)):
            if visited[i]:
                continue
            subset.append(nums[i])
            visited[i] = True
            self.dfs(nums, subset, visited, results)
            visited[i] = False
            subset.pop()
```

回溯算法解题套路框架

通知： 数据结构精品课 已更新到 V2.0，第 14 期打卡训练营开始报名。 读完本文，你不仅学会了算法套路，还可以顺便解决如下题目： ---- 这篇文章是很久之前的一篇 回溯算法详解 的进阶版，之前

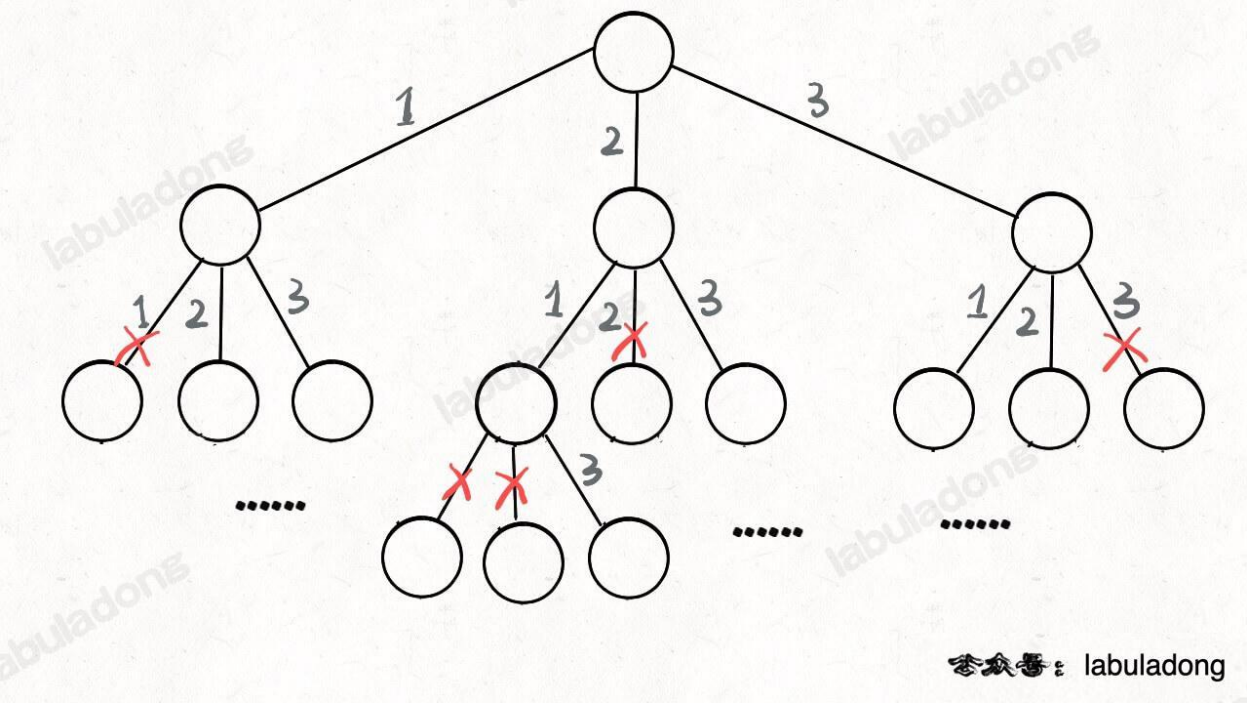
https://labuladong.github.io/algo/4/31/104/




```
List<List<Integer>> res = new LinkedList<>();

/* 主函数，输入一组不重复的数字，返回它们的全排列 */
List<List<Integer>> permute(int[] nums) {
    // 记录「路径」
    LinkedList<Integer> track = new LinkedList<>();
    // 「路径」中的元素会被标记为 true，避免重复使用
    boolean[] used = new boolean[nums.length];

    backtrack(nums, track, used);
```

```
    return res;
}

// 路径：记录在 track 中
// 选择列表：nums 中不存在于 track 的那些元素（used[i] 为 false）
// 结束条件：nums 中的元素全都在 track 中出现
void backtrack(int[] nums, LinkedList<Integer> track, boolean[] used) {
    // 触发结束条件
    if (track.size() == nums.length) {
        res.add(new LinkedList(track));
        return;
    }

    for (int i = 0; i < nums.length; i++) {
        // 排除不合法的选择
        if (used[i]) {
            // nums[i] 已经在 track 中，跳过
            continue;
        }
        // 做选择
        track.add(nums[i]);
        used[i] = true;
        // 进入下一层决策树
        backtrack(nums, track, used);
        // 取消选择
        track.removeLast();
        used[i] = false;
    }
}
```

why we need “used”?



但是必须说明的是，不管怎么优化，都符合回溯框架，而且时间复杂度都不可能低于 O(N!)，因为穷举整棵决策树是无法避免的。**这也是回溯算法的一个特点，不像动态规划存在重叠子问题可以优化，回溯算法就是纯暴力穷举，复杂度一般都很高**。

类似的题目：

- 51. N-Queens

47. Permutations II

Created	@July 20, 2020 12:01 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/permutations-ii/
Importance	*****
Tag	Backtrack
Video	

Given a collection of numbers, `nums`, that might contain duplicates, return *all possible unique permutations* ***in any order***.

Example 1:

```
Input: nums = [1,1,2]
Output:
[[1,1,2],
 [1,2,1],
 [2,1,1]]
```

Example 2:

```
Input: nums = [1,2,3]
Output: [[1,2,3],[1,3,2],[2,1,3],[2,3,1],[3,1,2],[3,2,1]]
```

Constraints:

- `1 <= nums.length <= 8`
- `10 <= nums[i] <= 10`

Solution

```python
class Solution:
    def permuteUnique(self, nums: List[int]) -> List[List[int]]:
        res = []
        if not nums:
            return res

        visited = [False] * len(nums)
        nums.sort()
        self.backtrack(nums, [], res, visited)
        return res

    def backtrack(self, nums, subset, res, visited):
        if len(subset) == len(nums):
            res.append(list(subset))
            return

        for i in range(len(nums)):
            if visited[i]:
                continue

            if i > 0 and nums[i] == nums[i - 1] and not visited[i - 1]:
                continue

            subset.append(nums[i])
            visited[i] = True

            self.backtrack(nums, subset, res, visited)

            visited[i] = False
            subset.pop()
```

```java
List<List<Integer>> res = new LinkedList<>();
LinkedList<Integer> track = new LinkedList<>();
boolean[] used;

public List<List<Integer>> permuteUnique(int[] nums) {
    // 先排序，让相同的元素靠在一起
    Arrays.sort(nums);
    used = new boolean[nums.length];
    backtrack(nums);
    return res;
}

void backtrack(int[] nums) {
    if (track.size() == nums.length) {
        res.add(new LinkedList(track));
        return;
    }

    for (int i = 0; i < nums.length; i++) {
```

```
        if (used[i]) {
            continue;
        }
        // 新添加的剪枝逻辑，固定相同的元素在排列中的相对位置
        if (i > 0 && nums[i] == nums[i - 1] && !used[i - 1]) {
            continue;
        }
        track.add(nums[i]);
        used[i] = true;
        backtrack(nums);
        track.removeLast();
        used[i] = false;
    }
}
```

36. Valid Sudoku

Created	@July 19, 2020 9:37 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/valid-sudoku/
Importance	***
Tag	Array&Sorting NEET
Video	https://www.youtube.com/watch?v=TjFXEUCMqI8&t=42s

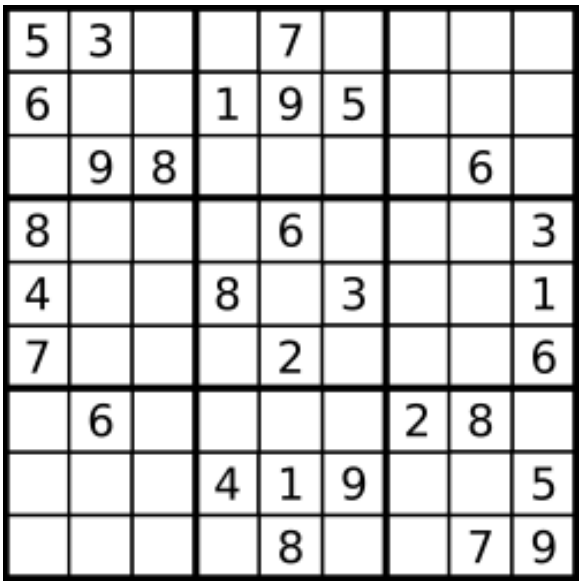
Determine if a `9 x 9` Sudoku board is valid. Only the filled cells need to be validated **according to the following rules**:

1. Each row must contain the digits `1-9` without repetition.
2. Each column must contain the digits `1-9` without repetition.
3. Each of the nine `3 x 3` sub-boxes of the grid must contain the digits `1-9` without repetition.

Note:

- A Sudoku board (partially filled) could be valid but is not necessarily solvable.
- Only the filled cells need to be validated according to the mentioned rules.

Example 1:




```
Input: board =
[["5","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]
,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".","2",".",".",".","6"]
,[".","6",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]]
Output: true
```

Example 2:

```
Input: board =
[["8","3",".",".","7",".",".",".","."]
,["6",".",".","1","9","5",".",".","."]
,[".","9","8",".",".",".",".","6","."]
,["8",".",".",".","6",".",".",".","3"]
,["4",".",".","8",".","3",".",".","1"]
,["7",".",".",".","2",".",".",".","6"]
,[".","6",".",".",".",".","2","8","."]
,[".",".",".","4","1","9",".",".","5"]
,[".",".",".",".","8",".",".","7","9"]]
Output: false
Explanation: Same as Example 1, except with the5 in the top left corner being modified to
8. Since there are two 8's in the top left 3x3 sub-box, it is invalid.
```

Constraints:

- `board.length == 9`
- `board[i].length == 9`
- `board[i][j]` is a digit `1-9` or `'.'`.

Solution

方法：一次遍历

有效的数独满足以下三个条件：

- 同一个数字在每一行只能出现一次；
- 同一个数字在每一列只能出现一次；
- 同一个数字在每一个小九宫格只能出现一次。

可以使用哈希表记录每一行、每一列和每一个小九宫格中，每个数字出现的次数。只需要遍历数独一次，在遍历的过程中更新哈希表中的计数，并判断是否满足有效的数独的条件即可。

对于数独的第 i 行第 j 列的单元格，其中 $0 \leq i, j < 9$，该单元格所在的行下标和列下标分别为 i 和 j，该单元格所在的小九宫格的行数和列数分别为 $\left\lfloor \frac{i}{3} \right\rfloor$ 和 $\left\lfloor \frac{j}{3} \right\rfloor$，其中 $0 \leq \left\lfloor \frac{i}{3} \right\rfloor, \left\lfloor \frac{j}{3} \right\rfloor < 3$。

由于数独中的数字范围是 1 到 9，因此可以使用数组代替哈希表进行计数。

具体做法是，创建二维数组 $rows$ 和 $columns$ 分别记录数独的每一行和每一列中的每个数字的出现次数，创建三维数组 $subboxes$ 记录数独的每一个小九宫格中的每个数字的出现次数，其中 $rows[i][index]$、$columns[j][index]$ 和 $subboxes\left[\left\lfloor \frac{i}{3} \right\rfloor\right]\left[\left\lfloor \frac{j}{3} \right\rfloor\right][index]$ 分别表示数独的第 i 行第 j 列的单元格所在的行、列和小九宫格中，数字 $index+1$ 出现的次数，其中 $0 \leq index < 9$，对应的数字 $index+1$ 满足 $1 \leq index+1 \leq 9$。

如果 $board[i][j]$ 填入了数字 n，则将 $rows[i][n-1]$、$columns[j][n-1]$ 和 $subboxes\left[\left\lfloor \frac{i}{3} \right\rfloor\right]\left[\left\lfloor \frac{j}{3} \right\rfloor\right][n-1]$ 各加 1。如果更新后的计数大于 1，则不符合有效的数独的条件，返回 false。

如果遍历结束之后没有出现计数大于 1 的情况，则符合有效的数独的条件，返回 true。

```
class Solution:
    def isValidSudoku(self, board: List[List[str]]) -> bool:
        cols = collections.defaultdict(set)
        rows = collections.defaultdict(set)
        squares = collections.defaultdict(set)  # key = (r // 3, c // 3)
```

```
        for r in range(9):
            for c in range(9):
                if board[r][c] == ".":
                    continue
                if (
                    board[r][c] in rows[r]
                    or board[r][c] in cols[c]
                    or board[r][c] in squares[(r // 3, c // 3)]
                ):
                    return False
                cols[c].add(board[r][c])
                rows[r].add(board[r][c])
                squares[(r // 3, c // 3)].add(board[r][c])

        return True
```

复杂度分析

- 时间复杂度：$O(1)$。数独共有 81 个单元格，只需要对每个单元格遍历一次即可。
- 空间复杂度：$O(1)$。由于数独的大小固定，因此哈希表的空间也是固定的。

https://www.lintcode.com/problem/389/solution/57196

37. Sudoku Solver

Created	@November 27, 2022 3:10 PM
Difficulty	Hard
LC Url	https://leetcode.com/problems/sudoku-solver/
Importance	****
Tag	Backtrack DFS
Video	https://www.lintcode.com/problem/802/solution/16753

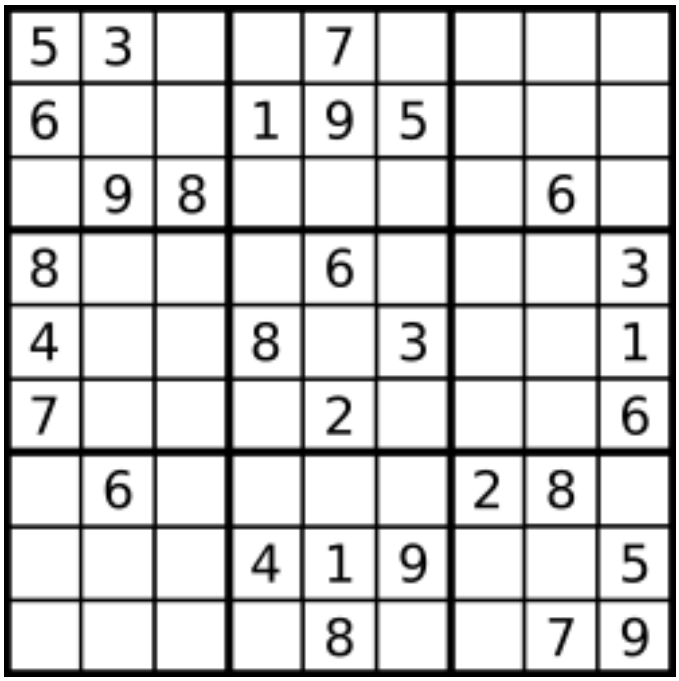
Write a program to solve a Sudoku puzzle by filling the empty cells.

A sudoku solution must satisfy **all of the following rules**:

1. Each of the digits `1-9` must occur exactly once in each row.
2. Each of the digits `1-9` must occur exactly once in each column.
3. Each of the digits `1-9` must occur exactly once in each of the 9 `3x3` sub-boxes of the grid.

The `'.'` character indicates empty cells.

Example 1:



```
Input: board = [["5","3",".",".","7",".",".",".","."],
["6",".",".","1","9","5",".",".","."],[".","9","8",".",".",".",".","6","."],
["8",".",".",".","6",".",".",".","3"],["4",".",".","8",".","3",".",".","1"],
["7",".",".",".","2",".",".",".","6"],[".","6",".",".",".",".","2","8","."],
[".",".",".","4","1","9",".",".","5"],[".",".",".",".","8",".",".","7","9"]]
Output: [["5","3","4","6","7","8","9","1","2"],["6","7","2","1","9","5","3","4","8"],
["1","9","8","3","4","2","5","6","7"],["8","5","9","7","6","1","4","2","3"],
["4","2","6","8","5","3","7","9","1"],["7","1","3","9","2","4","8","5","6"],
["9","6","1","5","3","7","2","8","4"],["2","8","7","4","1","9","6","3","5"],
["3","4","5","2","8","6","1","7","9"]]
Explanation: The input board is shown above and the only valid solution is shown below:
```

Constraints:

- `board.length == 9`
- `board[i].length == 9`
- `board[i][j]` is a digit or `'.'`.
- It is **guaranteed** that the input board has only one solution.

Solution

- 对于这道题目来说，我们要明确几点
  - 目标(base case)：填满每个格子。
  - 选择列表(choice list)：每个空的格子可以填入`1-9`这九个数字。
  - 约束条件(constrain)：`1-9`在每一行、每一列和每个3x3的大格中都只能出现一次。
  - 选择路径(path)：由于我们是inplace操作，`board`就存储了已经做过的选择。
- 我们在这里设置`backtrack`函数的返回类型为`bool`，这样一来，当程序找到可行解后就能终止递归。

```
class Solution:
    def solveSudoku(self, board: List[List[str]]) -> None:
```

```
        """
        Do not return anything, modify board in-place instead.
        """
        used = self.initial_used(board)
        self.dfs(board, 0, used)

    def initial_used(self, board):
        """
        Initialize flags for rows, columns, and boxes.
        """
        used = {
            'row': [set() for _ in range(9)],
            'col': [set() for _ in range(9)],
            'box': [set() for _ in range(9)],
        }

        for r in range(9):
            for c in range(9):
                cur = board[r][c]
                if cur == '.':
                    continue
                used['row'][r].add(cur)
                used['col'][c].add(cur)
                used['box'][r // 3 * 3 + c // 3].add(cur)
        return used

    def dfs(self, board, index, used):
        # return True after reaching the last element
        if index == 81:
            return True

        # find the global row and colmun index
        r, c = index // 9, index % 9
        if board[r][c] != '.':
            return self.dfs(board, index + 1, used)

        # try for different numbers
        for val in range(1, 10):
            # Note if the value is an int or string
            val = str(val)
            if not self.is_valid(r, c, val, used):
                continue

            # backtrack
            board[r][c] = val
            used['row'][r].add(val)
            used['col'][c].add(val)
            used['box'][r // 3 * 3 + c // 3].add(val)

            if self.dfs(board, index + 1, used):
                return True

            used['box'][r // 3 * 3 + c // 3].remove(val)
            used['col'][c].remove(val)
```

```
                used['row'][r].remove(val)
                board[r][c] = '.'

        return False

    def is_valid(sefl, r, c, val, used):
        """
        Check whether the val can be put in the board
        """
        if val in used['row'][r]:
            return False
        if val in used['col'][c]:
            return False
        if val in used['box'][r // 3 * 3 + c // 3]:
            return False
        return True
```

Ref: 37 算法班2020 37.4 数独问题的暴力搜索解法_1

51. N-Queens

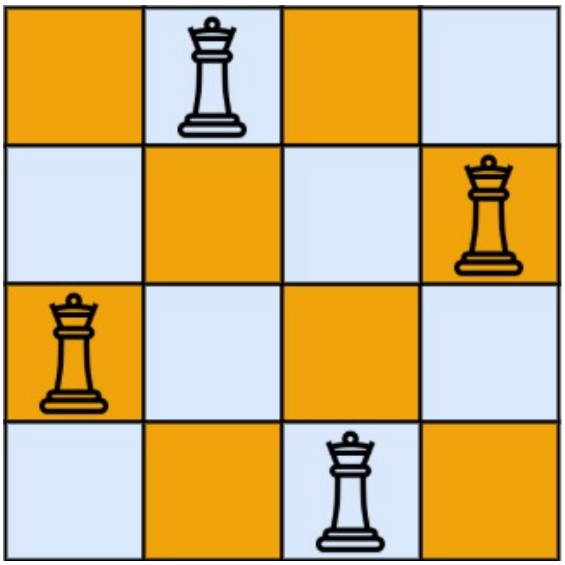
Created	@November 27, 2022 11:21 AM
Difficulty	Hard
LC Url	https://leetcode.com/problems/n-queens/
Importance	
Tag	Backtrack
Video	

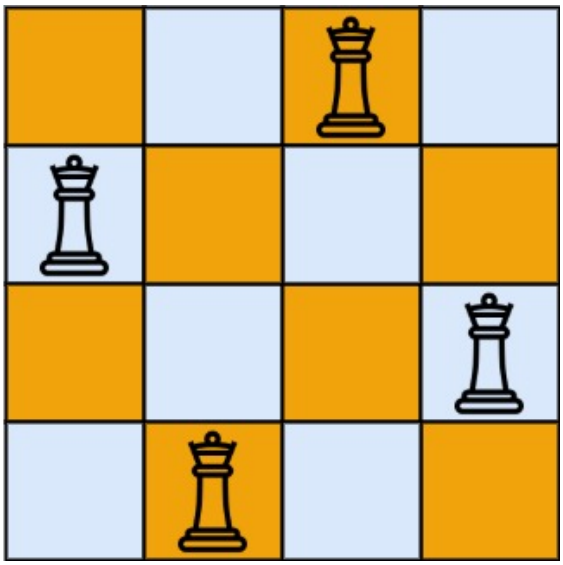
The **n-queens** puzzle is the problem of placing `n` queens on an `n x n` chessboard such that no two queens attack each other.

Given an integer `n`, return *all distinct solutions to the **n-queens puzzle***. You may return the answer in **any order**.

Each solution contains a distinct board configuration of the n-queens' placement, where `'Q'` and `'.'` both indicate a queen and an empty space, respectively.

Example 1:





```
Input: n = 4
Output: [[".Q..","...Q","Q...","..Q."],["..Q.","Q...","...Q",".Q.."]]
Explanation: There exist two distinct solutions to the 4-queens puzzle as shown above
```

Example 2:

```
Input: n = 1
Output: [["Q"]]
```

Constraints:

- `1 <= n <= 9`

Solution

37 Chapter_37._DFS经典题精讲

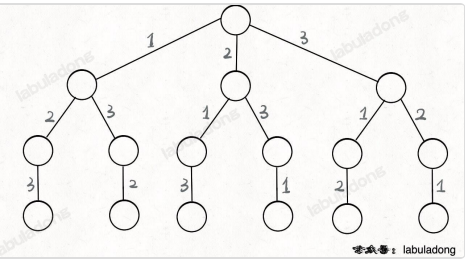
```
class Solution:
    def solveNQueens(self, n: int) -> List[List[str]]:
        results = []
        self.search(n, [], results)
        return results

    def search(self, n, cols, results):
        """
        n: board size
        cols: a list stores the column number for each row
        results: total combinations
        """
        row = len(cols)
        if row == n:
            results.append(self.draw_chessboard(cols))
            return

        for col in range(n):
            if not self.is_valid(cols, row, col):
                continue
            cols.append(col)
            self.search(n, cols, results)
            cols.pop()

    def draw_chessboard(self, cols):
        """
```

```
        画棋盘
        """
        n = len(cols)
        board = []
        for i in range(n):
            row = ['Q' if j == cols[i] else '.' for j in range(n)]
            board.append(''.join(row))
        return board

    def is_valid(self, cols, row, col):
        """
        cols: 已经放的位置
        row, col: 准备放的位置
        """
        for r, c in enumerate(cols):
            # 是否是同一列。注意不用考虑是否为同一行，因为每一行只放一次
            if c == col:
                return False
            # 对角线的考虑：左上角和右上角
            if r - c == row - col or r + c == row + col:
                return False
        return True
```

回溯算法解题套路框架

通知： 数据结构精品课 已更新到 V2.0，第 14 期打卡训练营开始报名。 读完本文，你不仅学会了算法套路，还可以顺便解决如下题目： ---- 这篇文章是很久之前的一篇 回溯算法详解 的进阶版，之前

https://labuladong.github.io/algo/4/31/104/



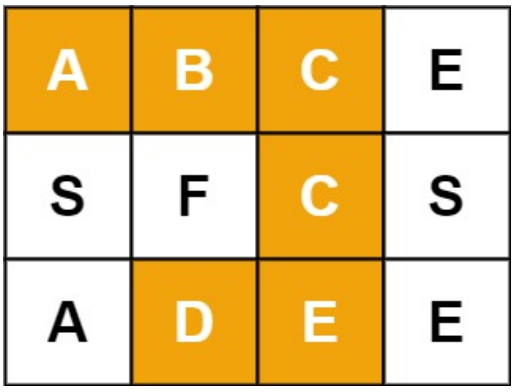
79. Word Search

Created	@July 10, 2021 6:03 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/word-search/
Importance	*****
Tag	Backtrack DFS
Video	https://www.youtube.com/watch?v=1zSg1WdmhIs

Given an `m x n` grid of characters `board` and a string `word`, return `true` *if* `word` *exists in the grid*.

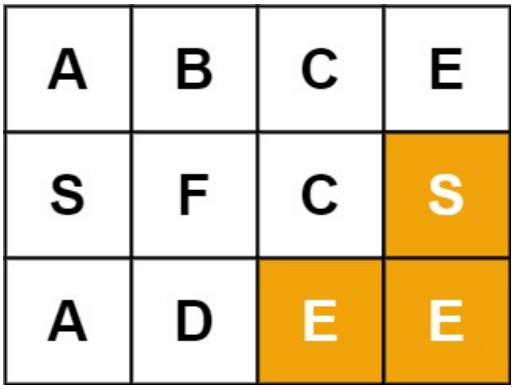
The word can be constructed from letters of sequentially adjacent cells, where adjacent cells are horizontally or vertically neighboring. The same letter cell may not be used more than once.

Example 1:



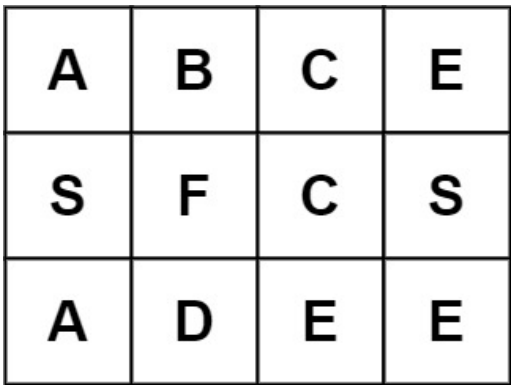

```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCCED"
Output: true
```

Example 2:




```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "SEE"
Output: true
```

Example 3:




```
Input: board = [["A","B","C","E"],["S","F","C","S"],["A","D","E","E"]], word = "ABCB"
Output: false
```

Constraints:

- `m == board.length`
- `n = board[i].length`
- `1 <= m, n <= 6`
- `1 <= word.length <= 15`
- `board` and `word` consists of only lowercase and uppercase English letters.

Solution

整体思路

使用深度优先搜索（DFS）和回溯的思想实现。关于判断元素是否使用过，我用了一个二维数组 `mark` 对使用过的元素做标记。

外层：遍历

首先遍历 `board` 的所有元素，先找到和 `word` 第一个字母相同的元素，然后进入递归流程。假设这个元素的坐标为 `(i, j)`，进入递归流程前，先记得把该元素打上**使用过**的标记：

```
mark[i][j] = 1
```

内层：递归

好了，打完标记了，现在我们进入了递归流程。递归流程主要做了这么几件事：

1. 从 `(i, j)` 出发，朝它的上下左右试探，看看它周边的这四个元素是否能匹配 `word` 的下一个字母

- 如果匹配到了：带着该元素继续进入下一个递归
- 如果都匹配不到：返回 `False`

2. 当 `word` 的所有字母都完成匹配后，整个流程返回 `True`

几个注意点

- 递归时元素的坐标是否超过边界
- 回溯标记 `mark[i][j] = 0` 以及 `return` 的时机

```
class Solution:

    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    def exist(self, board: List[List[str]], word: str) -> bool:
        m = len(board)
        if m == 0:
            return False

        n = len(board[0])
        visited = [[0 for _ in range(n)] for _ in range(m)]

        for i in range(m):
            for j in range(n):
                if board[i][j] == word[0]:
                    visited[i][j] = 1
                    if self.backtrack(i, j, visited, board, word[1:]) == True:
                        return True
                    else:
                        visited[i][j] = 0
        return False

    def backtrack(self, i, j, visited, board, word):
        if len(word) == 0:
            return True

        for direct in self.directions:
```

```
            cur_i = i + direct[0]
            cur_j = j + direct[1]

            if 0 <= cur_i < len(board) and 0 <= cur_j < len(board[0]) and board[cur_i][cur_j] == word[0]:
                if visited[cur_i][cur_j] == 1:
                    continue
                visited[cur_i][cur_j] = 1
                if self.backtrack(cur_i, cur_j, visited, board, word[1:]) == True:
                    return True
                else:
                    visited[cur_i][cur_j] = 0
        return False
```

力扣

https://leetcode.cn/problems/word-search/solution/shen-du-you-xian-sou-suo-yu-hui-su-xiang-jie-by-ja/

```
class Solution:
    def exist(self, board: List[List[str]], word: str) -> bool:
        for i in range(len(board)):
            for j in range(len(board[0])):
                if self.dfs(board, i, j, word, 0):
                    return True
        return False

    def dfs(self, board, i, j, word, wordIndex):
        if wordIndex == len(word):
            return True

        if i < 0 or i >= len(board) or j < 0 or j >= len(board[0]) or word[wordIndex] != board[i][j]:
            return False

        temp = board[i][j]
        board[i][j] = ''

        found = self.dfs(board, i+1, j, word, wordIndex+1) \
                or self.dfs(board, i-1, j, word, wordIndex+1) \
                or self.dfs(board, i, j+1, word, wordIndex+1) \
                or self.dfs(board, i, j-1, word, wordIndex+1)

        board[i][j] = temp

        return found
```

131. Palindrome Partitioning

Created	@September 21, 2022 3:40 PM
Difficulty	Medium
LC Url	https://leetcode.com/problems/palindrome-partitioning/
Importance	
Tag	Backtrack DFS
Video	https://www.youtube.com/watch?v=3jvWodd7ht0

Given a string `s`, partition `s` such that every substring of the partition is a **palindrome**. Return all possible palindrome partitioning of `s`.

A **palindrome** string is a string that reads the same backward as forward.

Example 1:

```
Input: s = "aab"
Output: [["a","a","b"],["aa","b"]]
```

Example 2:

```
Input: s = "a"
Output: [["a"]]
```

Constraints:

- `1 <= s.length <= 16`
- `s` contains only lowercase English letters.

Solution

Backtrack

```
class Solution:
    def partition(self, s: str) -> List[List[str]]:
        res = []
        part = []

        def dfs(i):
            if i >= len(s):
                res.append(part.copy())
                return

            for j in range(i, len(s)):
                if self.isPali(s, i, j):
                    part.append(s[i:j + 1])
                    dfs(j + 1)
                    part.pop()
        dfs(0)
        return res

    def isPali(self, s, left, right):
        while left < right:
            if s[left] != s[right]:
                return False
            left, right = left + 1, right - 1
        return True
```

复杂度分析

- 时间复杂度：$O(n \cdot 2^n)$，其中 n 是字符串 s 的长度，与方法一相同。
- 空间复杂度：$O(n^2)$，与方法一相同。

17. Letter Combinations of a Phone Number

Created	@July 15, 2020 10:24 AM
Difficulty	Medium
LC Url	https://leetcode.com/problems/letter-combinations-of-a-phone-number/
Importance	
Tag	Backtrack
Video	

Given a string containing digits from `2-9` inclusive, return all possible letter combinations that the number could represent. Return the answer in **any order**.

A mapping of digits to letters (just like on the telephone buttons) is given below. Note that 1 does not map to any letters.

Example 1:

```
Input: digits = "23"
Output: ["ad","ae","af","bd","be","bf","cd","ce","cf"]
```

Example 2:

```
Input: digits = ""
Output: []
```

Example 3:

```
Input: digits = "2"
Output: ["a","b","c"]
```

Constraints:

- `0 <= digits.length <= 4`
- `digits[i]` is a digit in the range `['2', '9']`.

Solution

方法一：回溯

首先使用哈希表存储每个数字对应的所有可能的字母，然后进行回溯操作。

回溯过程中维护一个字符串，表示已有的字母排列（如果未遍历完电话号码的所有数字，则已有的字母排列是不完整的）。该字符串初始为空。每次取电话号码的一位数字，从哈希表中获得该数字对应的所有可能的字母，并将其中的一个字母插入到已有的字母排列后面，然后继续处理电话号码的后一位数字，直到处理完电话号码中的所有数字，即得到一个完整的字母排列。然后进行回退操作，遍历其余的字母排列。

回溯算法用于寻找所有的可行解，如果发现一个解不可行，则会舍弃不可行的解。在这道题中，由于每个数字对应的每个字母都可能进入字母组合，因此不存在不可行的解，直接穷举所有的解即可。

```
class Solution:
    phoneMap = {
        "2": "abc",
        "3": "def",
        "4": "ghi",
        "5": "jkl",
        "6": "mno",
        "7": "pqrs",
        "8": "tuv",
        "9": "wxyz",
    }

    def letterCombinations(self, digits: str) -> List[str]:
        if not digits:
            return []

        subset = []
        res = []
        self.backtrack(digits, 0, subset, res)

        return res

    def backtrack(self, digits, index, subset, res):
        if index == len(digits):
            res.append(''.join(subset))
        else:
            digit = digits[index]
            for c in self.phoneMap[digit]:
                subset.append(c)
                self.backtrack(digits, index + 1, subset, res)
                subset.pop()
```

```
class Solution:
    def letterCombinations(self, digits: str) -> List[str]:
        if not digits:
            return []

        phoneMap = {
            "2": "abc",
            "3": "def",
            "4": "ghi",
            "5": "jkl",
            "6": "mno",
            "7": "pqrs",
            "8": "tuv",
            "9": "wxyz",
        }

        def backtrack(index):
            if index == len(digits):
                combinations.append(''.join(combination))
            else:
                digit = digits[index]
                for c in phoneMap[digit]:
                    combination.append(c)
                    backtrack(index + 1)
                    combination.pop()

        combination = []
        combinations = []
        backtrack(0)
```

```
        return combinations

        # 链接：https://leetcode.cn/problems/letter-combinations-of-a-phone-number/solution/dian-hua-hao-ma-de-zi-mu-zu-he-by-leetcode-solut
```

复杂度分析

- 时间复杂度：$O(3^m \times 4^n)$，其中 m 是输入中对应 3 个字母的数字个数（包括数字 2、3、4、5、6、8），n 是输入中对应 4 个字母的数字个数（包括数字 7、9），$m+n$ 是输入数字的总个数。当输入包含 m 个对应 3 个字母的数字和 n 个对应 4 个字母的数字时，不同的字母组合一共有 $3^m \times 4^n$ 种，需要遍历每一种字母组合。
- 空间复杂度：$O(m+n)$，其中 m 是输入中对应 3 个字母的数字个数，n 是输入中对应 4 个字母的数字个数，$m+n$ 是输入数字的总个数。除了返回值以外，空间复杂度主要取决于哈希表以及回溯过程中的递归调用层数，哈希表的大小与输入无关，可以看成常数，递归调用层数最大为 $m+n$。