回溯、DFS

0046 全排列

链接：https://leetcode-cn.com/problems/permutations/

标签：回溯

精选题解

- 回溯算法入门级详解 + 练习（持续更新）
  - https://leetcode-cn.com/problems/permutations/solution/hui-su-suan-fa-python-dai-ma-java-dai-ma-by-liweiw/
- 官方题解
  - https://leetcode-cn.com/problems/permutations/solution/quan-pai-lie-by-leetcode-solution-2/
- ※ **C++ 回溯法/交换法/stl 简洁易懂的全排列**
  - https://leetcode-cn.com/problems/permutations/solution/c-hui-su-fa-jiao-huan-fa-stl-jian-ji-yi-dong-by-sm/
- 精选代码
  - https://leetcode-cn.com/problems/permutations/solution/quan-pai-lie-by-leetcode-solution-2/532710/

复杂度

- 时间复杂度：O(n*n!)。回溯复杂度 O(n!)；每次新的生成数组需要复制 n 个元素。
- 空间复杂度：SO(n)。长度为 n 的标记数组；递归时深度最大为 n。

代码 1：标记数组

0046 全排列.cpp

https://leetcode-cn.com/submissions/detail/127476452/

```
1  /*
2  * 【46】C++ 回溯法/交换法/stl 简洁易懂的全排列
3      * https://leetcode-cn.com/problems/permutations/solution/c-hui-su-fa-jiao-huan-fa-stl-jian-ji-yi-dong-by-sm/
4  */
5  class Solution {
6  public:
7      vector<vector<int>> res;
8
```

0046 全排列.cpp

```
9       void backtrack(vector<int> &nums, vector<int> &current, vector<bool>
    &flags) {
10          if (current.size() == flags.size()) {
11              res.push_back(current);
12          } else {
13              for (int i=0; i<nums.size(); ++i) {
14                  if (not flags[i]) { // nums[i] not in current
15                      current.push_back(nums[i]);
16                      flags[i] = true;
17                      backtrack(nums, current, flags);
18                      current.pop_back();
19                      flags[i] = false;
20                  }
21              }
22          }
23      }
24
25      vector<vector<int>> permute(vector<int>& nums) {
26          if (nums.empty()) {
27              return {};
28          }
29          vector<bool> flags(nums.size(), false); // true: in current; false:
    not in current
30          vector<int> current;
31          backtrack(nums, current, flags);
32          return res;
33      }
34  };
35
```

代码 2：交换元素

0046 全排列 - swap.cpp

https://leetcode-cn.com/submissions/detail/127482585/

```
1   /*
2   * 【46】C++ 回溯法/交换法/stl 简洁易懂的全排列 - 全排列 - 力扣（LeetCode）
3       * https://leetcode-cn.com/problems/permutations/solution/c-hui-su-fa-jiao-
    huan-fa-stl-jian-ji-yi-dong-by-sm/
4   * 全排列 - 全排列 - 力扣（LeetCode）
5       * https://leetcode-cn.com/problems/permutations/solution/quan-pai-lie-by-
    leetcode-solution-2/
6   */
```

0046 全排列 - swap.cpp

```cpp
class Solution {
public:
    vector<vector<int>> res;

    void backtrack(vector<int> &nums, int start, int end) {
        if (start == end) {
            res.push_back(nums);
        } else {
            for (int i=start; i<=end; ++i) {
                swap(nums[i], nums[start]);
                backtrack(nums, start+1, end);
                swap(nums[i], nums[start]);
            }
        }
    }

    vector<vector<int>> permute(vector<int>& nums) {
        if (nums.empty()) {
            return {};
        } else {
            backtrack(nums, 0, nums.size()-1);
            return res;
        }
    }
};

```

0047 全排列 II

题目：https://leetcode-cn.com/problems/permutations-ii/

标签：回溯

精选题解

- 官方题解
  - https://leetcode-cn.com/problems/permutations-ii/solution/quan-pai-lie-ii-by-leetcode-solution/

关键思路

定义一个标记数组 visited 来标记已经填过的数。若 visited[i] 为 true，表示第 i 个数已经使用了；若 visited[i] 为 false，表示第 i 个数尚未使用。

要解决重复问题，只需保证在填第 i 个数时，重复数字只被填入一次。方法：**对原数组排序，保证相同数字都相邻，然后每次填入的数一定是这个数所在重复数集合中「从左往右第一个未被填过的数字」，即如下的判断条件：**

```
if (i > 0 && nums[i] == nums[i-1] && !visited[i-1]) {
    continue;
}
```

假如排完序后的完整数组 nums 中有三个连续的数，那么一定只有如下 4 种状态：[×, ×, ×]，[√, ×, ×]，[√, √, ×]，[√, √, √]。（√ 表示已在生成的数组中，× 表示未在生成的数组中。）

复杂度

详见官方题解。

- 时间复杂度：O(n*n!)。回溯复杂度 O(n!)；每次新的生成数组需要复制 n 个元素。
- 空间复杂度：SO(n)。长度为 n 的标记数组；递归时深度最大为 n。

代码

0047 全排列 II.cpp

https://leetcode-cn.com/problems/permutations-ii/submissions/

```
/*
* 全排列 II - 全排列 II - 力扣（LeetCode）
    * https://leetcode-cn.com/problems/permutations-ii/solution/quan-pai-lie-ii-by-leetcode-solution/
*/
class Solution {
    vector<int> visited;
public:
    void backtrack(vector<int> &nums, vector<vector<int>> &res, int idx, vector<int> &current) {
        if (idx==nums.size()) {
            res.emplace_back(current);
            // * C++ STL vector 添加元素（push_back()和 emplace_back()）详解
            //   * http://c.biancheng.net/view/6826.html
            // push_back() 向容器尾部添加元素时，首先会创建这个元素，然后再将这个元素拷贝或者移动到容器中（如果是拷贝的话，事后会自行销毁先前创建的这个元素）；
            // 而 emplace_back() 在实现时，则是直接在容器尾部创建这个元素，省去了拷贝或移动元素的过程。
```

0047 全排列 II.cpp

```
15              return ;
16          }
17
18          for (int i=0; i<nums.size(); ++i) {
19              // 哪些情况不取当前的元素：
20              // 1. 已经访问过/在当前路径数组中
21              // 2. 和前一个数相等，且前一个数未被填过（表明该数不是第一个未填的数，故仍然跳过）
22              //    反过来理解，如果前一个相等的数已经被填过，那么此时就可以插入这后一个相等的数了，
23              //    因为我们在上一层嵌套中，已经保证前一个数当时是第一个未被填过的数了
24              //    此时意味着我们在当前路径数组中存在多个相等的数了
25              if (visited[i] || (i>0 && nums[i]==nums[i-1] && !visited[i-1]))
26                  continue;
27              current.emplace_back(nums[i]);
28              visited[i] = true;
29              backtrack(nums, res, idx+1, current);
30              visited[i] = false;
31              current.pop_back();
32          }
33      }
34
35      vector<vector<int>> permuteUnique(vector<int>& nums) {
36          vector<vector<int>> res;
37          vector<int> current;
38          visited.resize(nums.size());
39          sort(nums.begin(), nums.end());
40          backtrack(nums, res, 0, current);
41          return res;
42      }
43  };
44
```

0077 组合

链接：https://leetcode-cn.com/problems/combinations/

标签：回溯

精选题解

- ※ 官方题解 - 组合
  - https://leetcode-cn.com/problems/combinations/solution/zu-he-by-leetcode-solution/
- 回溯算法 + 剪枝（Java） - 组合
  - https://leetcode-cn.com/problems/combinations/solution/hui-su-suan-fa-jian-zhi-python-dai-ma-java-dai-ma-/

关键思路

合理剪枝：剩余个数是否足够；个数正好则加入并返回。

```
if (temp.size() + (n - cur + 1) < k)
    return ;
```

选取当前数，需要考虑入栈出栈；不选取，则跳到下一个。

```
// choose cur
temp.push_back(cur);
dfs(cur+1, n, k);
temp.pop_back();

// do not choose cur
dfs(cur+1, n, k);
```

复杂度

- 时间复杂度：O($C_n^k \times k$)。其中C_n^k表示从 n 个数中取出 k 个数的组合数目，k 表示每次需要复制 k 个数。
- 空间复杂度：O(n+k)=O(n)。递归最大层数 n；临时数组空间 k。

代码

0077 组合.cpp

https://leetcode-cn.com/submissions/detail/138357287/

```
class Solution {
public:
    vector<vector<int>> res;   // result 2d vector
    vector<int> temp;          // temp vector path

    void dfs(int cur, int n, int k) {
    // cur: current element index, choose or not
```

0077 组合.cpp

```
8           if (temp.size() + (n - cur + 1) < k)
9               return ;
10          if (temp.size() == k) {
11              res.push_back(temp);
12              return ;
13          }
14
15          // choose cur
16          temp.push_back(cur);
17          dfs(cur+1, n, k);
18          temp.pop_back();
19
20          // do not choose cur
21          dfs(cur+1, n, k);
22      }
23
24      vector<vector<int>> combine(int n, int k) {
25          dfs(1, n, k);
26          return res;
27      }
28  };
29
```

0078 子集

链接：https://leetcode-cn.com/problems/subsets/

标签：回溯，位运算

精选题解

- 官方题解 - 子集
  - https://leetcode-cn.com/problems/subsets/solution/zi-ji-by-leetcode-solution/

关键思路

每个位置有两种情况，选或者不选，所以类似“0077 组合”(p5)的思路。最后索引到达 n 就退出。

```
1   // choose nums[cur]
2   temp.push_back(nums[cur]);
3   dfs(cur+1, nums);
4   temp.pop_back();
```

```
5
6   // not choose nums[cur]
7   dfs(cur+1, nums);
```

复杂度

- 时间复杂度：O($n \times 2^n$)。一共2^n个子集，每个子集需要 O(n)的时间来构造。
- 空间复杂度：SO(n)。递归栈空间 O(n)；临时数组空间 O(n)。

代码

0078 子集.cpp

https://leetcode-cn.com/submissions/detail/138375047/

```cpp
class Solution {
public:
    vector<vector<int>> res;
    vector<int> temp;

    void dfs(int cur, vector<int> &nums) {
        if (cur == nums.size()) {
            res.push_back(temp);
            return ;
        }

        // choose nums[cur]
        temp.push_back(nums[cur]);
        dfs(cur+1, nums);
        temp.pop_back();

        // not choose nums[cur]
        dfs(cur+1, nums);
    }

    vector<vector<int>> subsets(vector<int>& nums) {
        dfs(0, nums);
        return res;
    }
};

```

0090 子集 II

链接：https://leetcode-cn.com/problems/subsets-ii/

标签：回溯

精选题解

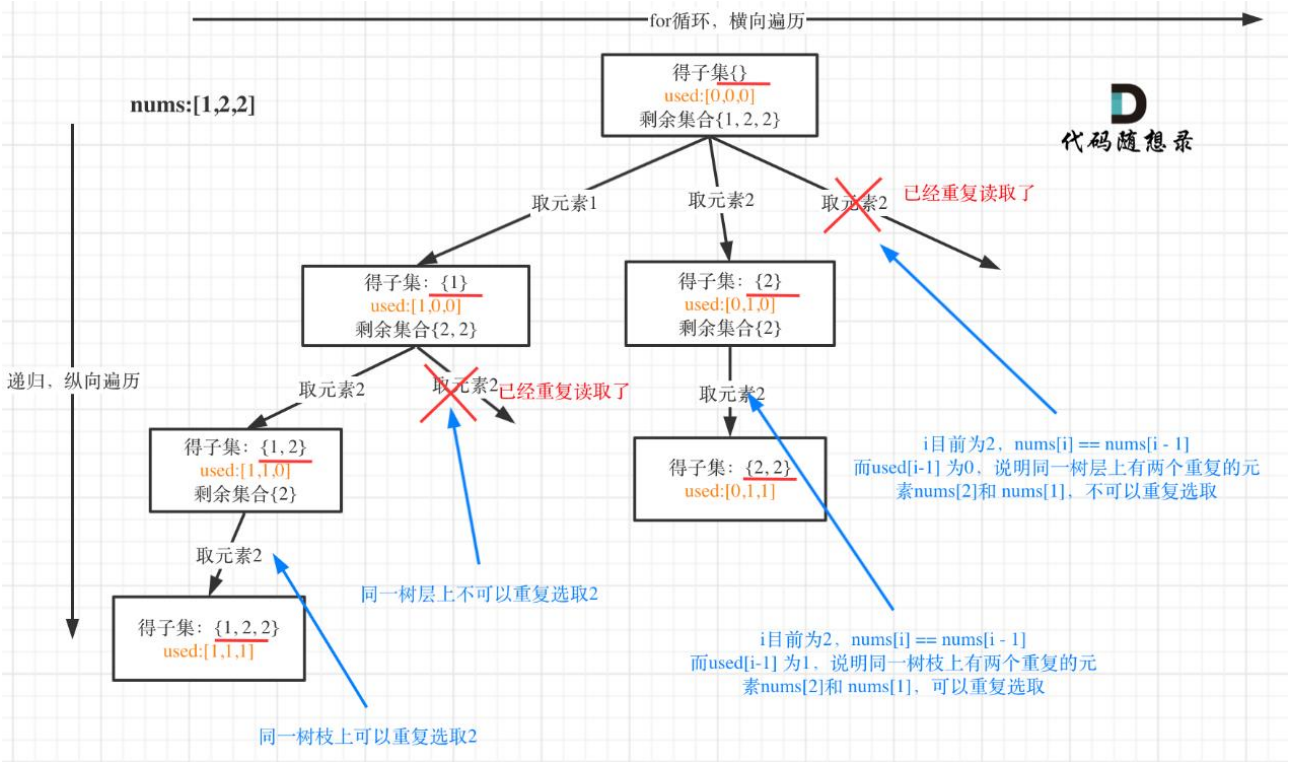
- 90. 子集 II:【彻底理解子集问题如何去重】详解 - 子集 II
  - https://leetcode-cn.com/problems/subsets-ii/solution/90-zi-ji-iiche-di-li-jie-zi-ji-wen-ti-ru-he-qu-zho/

关键思路

最重要的是理解 used[i-1] 的含义：

（1） true：同一**树枝**上选取过值相等的元素，可以重复选取

（2） false：同一**树层**上选取过值相等的元素，不可重复选取



类似“0047 全排列 II”，但有几点不同：

（1） 循环遍历的起点是 cur 而不是 0。因此，也就不需要判断 used[i] 是否为 true，因为这时肯定是 false。

（2） 可直接将 temp 加入 res，视为不选取 nums[cur]。

```
1   // not choose nums[cur]
2   res.push_back(temp);
3
4   // maybe choose nums[cur]
```

```
5   for (int i=cur; i<nums.size(); ++i) {
6       if (i>0 && nums[i]==nums[i-1] && !used[i-1])
7           continue;
8       temp.push_back(nums[i]);
9       used[i] = true;
10      backtrack(i+1, nums, used);
11      used[i] = false;
12      temp.pop_back();
13  }
```

事实上，还可以对上面的剪枝进行优化，不需要使用 used 数组，可以参考“0040 组合总和 II”的题解，重点是下面第 2 行的蓝色语句。代码 2 就是采用了该剪枝方法的优化解法。

```
1   for (int i=cur; i<nums.size(); ++i) {
2       if (i>cur && nums[i]==nums[i-1])
3           continue;
4       temp.push_back(nums[i]);
5       backtrack(i+1, nums);
6       temp.pop_back();
7   }
```

复杂度

时间复杂度：O($2^n \times n$)。子集最多有2^n个（元素均不重复）；构造每个子集需要 O(n)的时间。

空间复杂度：SO(n)。递归栈空间 O(n)；临时数组空间 O(n)。

代码 1

0090 子集 II
https://leetcode-cn.com/submissions/detail/138389158/

```
1   class Solution {
2   public:
3       vector<vector<int>> res;
4       vector<int> temp;
5
6       void backtrack(int cur, vector<int> &nums, vector<bool> &used) {
7           // not choose nums[cur]
8           res.push_back(temp);
9
10          // maybe choose nums[cur]
11          for (int i=cur; i<nums.size(); ++i) {
```

0090 子集 II

```
            if (i>0 && nums[i]==nums[i-1] && !used[i-1])
                continue;
            temp.push_back(nums[i]);
            used[i] = true;
            backtrack(i+1, nums, used);
            used[i] = false;
            temp.pop_back();
        }
    }

    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        vector<bool> used(nums.size(), false);
        sort(nums.begin(), nums.end());
        backtrack(0, nums, used);
        return res;
    }
};

```

代码 2：不使用 used 数组的剪枝

0090 子集 II -v2.cpp

https://leetcode-cn.com/submissions/detail/138841714/

```
class Solution {
public:
    vector<vector<int>> res;
    vector<int> temp;

    void backtrack(int cur, vector<int> &nums) {
        // not choose nums[cur]
        res.push_back(temp);

        // maybe choose nums[cur]
        for (int i=cur; i<nums.size(); ++i) {
            if (i>cur && nums[i]==nums[i-1])
                continue;
            temp.push_back(nums[i]);
            backtrack(i+1, nums);
            temp.pop_back();
        }
    }
```

0090 子集 II -v2.cpp

```
19
20      vector<vector<int>> subsetsWithDup(vector<int>& nums) {
21          sort(nums.begin(), nums.end());
22          backtrack(0, nums);
23          return res;
24      }
25  };
26
```

0079 单词搜索

链接：https://leetcode-cn.com/problems/word-search/

标签：回溯

精选题解

- ※ 官方题解 - 单词搜索
  - https://leetcode-cn.com/problems/word-search/solution/dan-ci-sou-suo-by-leetcode-solution/
- 在二维平面上使用回溯法（Python 代码、Java 代码） - 单词搜索
  - https://leetcode-cn.com/problems/word-search/solution/zai-er-wei-ping-mian-shang-shi-yong-hui-su-fa-pyth/

关键思路

- check(i,j,k,…) 表示是否存在一条从 board[i][j] 出发的路径与单词子集 word[k:]匹配。对 k=0，遍历所有的 i 和 j，即可得到二维网格中是否包含整个单词。
- 对不同方向的搜索，可以建立一个 vector<pair<int,int>> 表示四个方向。
- 终止条件：board[i][j] != word[k]，返回 false；board[i][j] == word[k] && k == word.length()-1，返回 true。
- 访问 board[i][j] 时将其置为 true，递归返回时不要忘记将其置为 false。

复杂度

详见官方题解。

- 时间复杂度：O($M \times N \times 3^L$)。M 表示 board 的行数，N 表示 board 的列数，L 表示字符串长度。需要进行$M \times N$次检查；每个后继字符至多有 3 个方向（除了第 2 个字符有 4 个方向），因此每次检查至多有3^L种分支；事实上由于提前返回和剪枝的存在，实际时间复杂度远低于这个理论上界。
- 空间复杂度：SO($M \times N$)。visited 数组的空间为$M \times N$；栈的深度至多为min $(L, M \times N)$。

代码

0079 单词搜索.cpp

https://leetcode-cn.com/submissions/detail/138405047/

```cpp
class Solution {
public:
    vector<pair<int,int>> directions{{0,1},{0,-1},{1,0},{-1,0}};
    // check(i,j,k,...): is exist a path starts from board[i][j] matches word[k:]
    bool check(int i, int j, int k, vector<vector<char>> &board, vector<vector<bool>> &visited, string &word) {
        if (board[i][j] != word[k])
            return false;
        else if (k==word.length()-1)
            return true;

        visited[i][j] = true;
        bool tmp_flag = false;
        for (const auto& d: directions) {
            int i_new = i + d.first;
            int j_new = j + d.second;
            if (i_new>=0 && i_new<board.size() && j_new>=0 && j_new<board[0].size()) {
                if (visited[i_new][j_new])
                    continue;
                tmp_flag = check(i_new, j_new, k+1, board, visited, word);
                if (tmp_flag) {
                    visited[i][j] = false;
                    return true;
                }
            }
        }
        visited[i][j] = false;
        return false;
    }

    bool exist(vector<vector<char>>& board, string word) {
        int row_num = board.size();
        if (row_num<=0)
            return false;
        int col_num = board[0].size();
        bool flag = false;
        vector<vector<bool>> visited(row_num, vector<bool>(col_num));
```

0079 单词搜索.cpp

```
37          for (int i=0; i<row_num; ++i) {
38              for (int j=0; j<col_num; ++j) {
39                  flag = check(i, j, 0, board, visited, word);
40                  if (flag)
41                      return true;
42              }
43          }
44          return false;
45      }
46  };
47
```

0039 组合总和

链接：https://leetcode-cn.com/problems/combination-sum/

标签：回溯

精选题解

- 官方题解 - 组合总和
  - https://leetcode-cn.com/problems/combination-sum/solution/zu-he-zong-he-by-leetcode-solution/

关键思路

几个终止条件（必须保证先后顺序）：

（1） idx 表示当前指向数字的索引，索引到达最后；

（2） 和恰好为 target，此时需将 temp 加入结果数组；

（3） 后面的数都比剩余的 target 大，要利用此条件需要在主函数中将 candidates 排序。

```
1   if (idx==candidates.size())
2       return ;
3   if (target==0) {
4       res.push_back(temp);
5       return ;
6   }
7
8   // This part must be after the previous part
9   if (target - candidates[idx] < 0) {
10      return ;
```

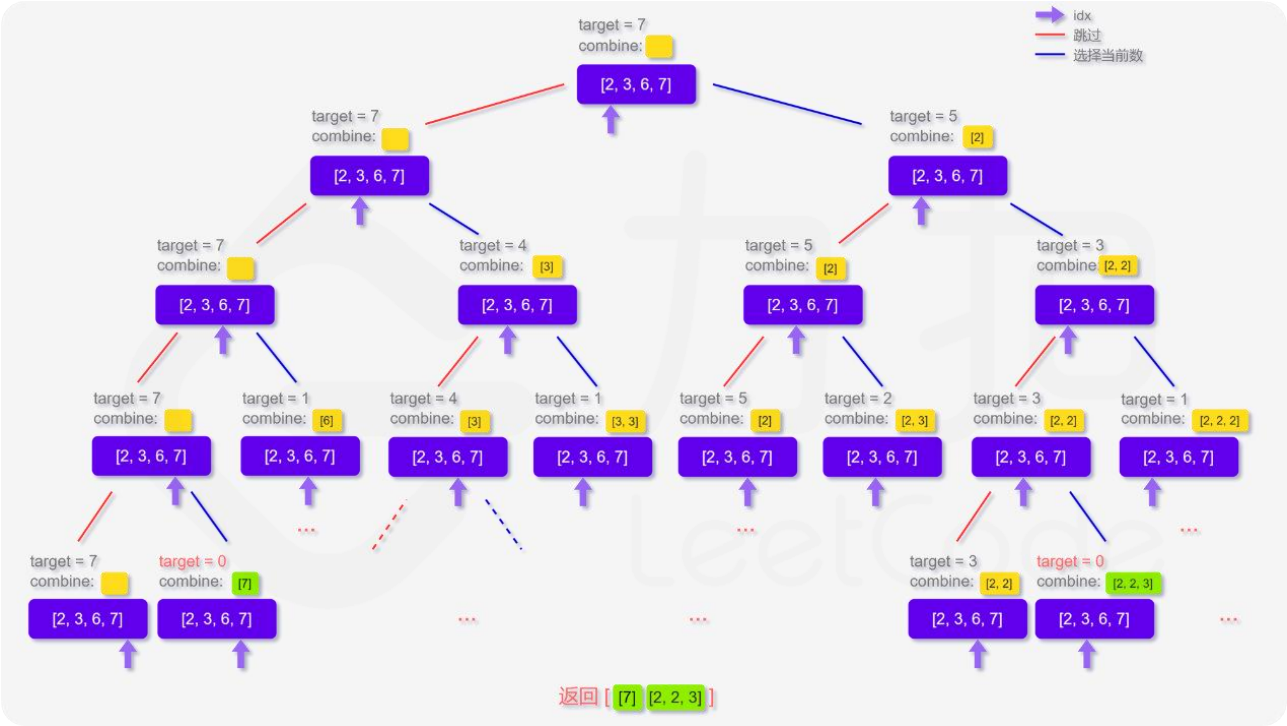
```
11  }
```

无非是取或不取当前数。区别有两点：

（1）是否出栈入栈（棕色部分）；

（2）dfs 的参数变化（红色部分）。

```
1   // do not choose candidates[idx]
2   dfs(candidates, target, idx+1);
3
4   // choose candidates[idx]
5   temp.push_back(candidates[idx]);
6   dfs(candidates, target-candidates[idx], idx);
7   temp.pop_back();
```



复杂度

- 时间复杂度：O($n \times 2^n$)。n 为数组中的元素个数，此处为一个松上界，因为存在大量提前返回和剪枝，因此实际情况远小于该复杂度。
- 空间复杂度：SO(target/min(candidates))。递归层数和临时数组空间最多均为 target/min(canditates)。

代码

0039 组合总和.cpp
https://leetcode-cn.com/submissions/detail/138434940/
`1  class Solution {`

0039 组合总和.cpp

```cpp
public:
    vector<vector<int>> res;
    vector<int> temp;

    void dfs(vector<int> & candidates, int target, int idx) {
        if (idx==candidates.size())
            return ;
        if (target==0) {
            res.push_back(temp);
            return ;
        }

        // This part must be after the previous part
        if (target - candidates[idx] < 0) {
            return ;
        }

        // do not choose candidates[idx]
        dfs(candidates, target, idx+1);

        // choose candidates[idx]
        temp.push_back(candidates[idx]);
        dfs(candidates, target-candidates[idx], idx);
        temp.pop_back();
    }

    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        sort(candidates.begin(), candidates.end());
        dfs(candidates, target, 0);
        return res;
    }
};

```

0040 组合总和 II

链接：https://leetcode-cn.com/problems/combination-sum-ii/

标签：回溯

精选题解

- 回溯算法 + 剪枝（Java、Python）

- https://leetcode-cn.com/problems/combination-sum-ii/solution/hui-su-suan-fa-jian-zhi-python-dai-ma-java-dai-m-3/
- https://leetcode-cn.com/problems/combination-sum-ii/solution/hui-su-suan-fa-jian-zhi-python-dai-ma-java-dai-m-3/225211

关键思路

最重要的是保证选取的数不重复，和“0090 子集 II”中的思路类似，同一树层上不应有相同的数（[1,2] 和 [1,2] 不被允许），同一树枝上可以（[1,2,2] 允许）。
需要注意的是，在“0090 子集 II”的代码 1 中使用了 used 数组，判断条件多了!used[i]，因此代码 2 对这个剪枝方法进行了改进，采用了本题下面的做法。
我们发现，在递归中，**同一个 for 循环里面的数都是在同一树层中的**，因此在同一个 for 循环中每个数只能使用一次。
使用 candidates[i]==canditates[i-1] 的含义是，将所有相同的数都跳过。
在大多数情况下，上面这条都是够的，只有当 i==idx 时，有可能出现这两个相同的数是在同一树枝，而不是同一树层，因为 idx 是循环起点，所以 candidates[idx-1] 在本层递归的 for 循环，而 canditates[idx-1] 必然在上层递归的 for 循环。所以加了一句 i>idx 的判断条件，用于排除这个例外情况。

```
1   for (int i=idx; i<candidates.size(); ++i) {
2       if (target - candidates[idx] < 0)
3           break;
4       if (i>idx && candidates[i] == candidates[i-1])
5           continue;
6       temp.push_back(candidates[i]);
7       dfs(candidates, target-candidates[i], i+1);
8       temp.pop_back();
9   }
```

复杂度

- 时间复杂度：O($2^n \times n$)。n 为 candidates 数组长度。递归时每个数都有选或不选两种可能，故有 O(2^n)的可能；每次复制符合条件的数组则需要 O(n)的时间。当然，这里是一个宽松的上界，因为在实际递归中，有很多提前返回和剪枝，因此要远小于该复杂度上界。
- 空间复杂度：SO(n)。递归深度最多为 n；temp 数组最多 n 个数。

代码

0040 组合总和 II.cpp

https://leetcode-cn.com/submissions/detail/138838782/

```
1   class Solution {
2   public:
3       vector<vector<int>> res;
4       vector<int> temp;
5
6       void dfs(vector<int>& candidates, int target, int idx) {
7           if (target==0) {
8               res.push_back(temp);
9               return ;
10          }
11
12          for (int i=idx; i<candidates.size(); ++i) {
13              if (target - candidates[idx] < 0)
14                  break;
15              if (i>idx && candidates[i] == candidates[i-1])
16                  continue;
17              temp.push_back(candidates[i]);
18              dfs(candidates, target-candidates[i], i+1);
19              temp.pop_back();
20          }
21      }
22
23      vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
24          sort(candidates.begin(), candidates.end());
25          dfs(candidates, target, 0);
26          return res;
27      }
28  };
29
```