回溯算法

回溯法

回溯法，一般可以解决如下几种问题：

- 组合问题：N个数里面按一定规则找出k个数的集合
- 切割问题：一个字符串按一定规则有几种切割方式
- 子集问题：一个N个数的集合里有多少符合条件的子集
- 排列问题：N个数按一定规则全排列，有几种排列方式
- 棋盘问题：N皇后，解数独等等

回溯法解决的问题都可以抽象为树形结构，因为回溯法解决的都是在集合中递归查找子集，**集合的大小就构成了树的宽度，递归的深度就构成了树的深度**。或者说，当for循环的层数写不出来的时候，就可以用回溯法！

递归就要有终止条件，所以必然是一棵高度有限的树（N叉树）。

```C#
void backtracking(参数) {
    if (终止条件) {
        存放结果;
        return;
    }

    for (选择：本层集合中元素（树中节点孩子的数量就是集合的大小）) {
        处理节点;
        backtracking(路径，选择列表); // 递归
        回溯，撤销处理结果
    }
}
```

回溯算法的关键：

- 回溯函数模板返回值以及参数，回溯算法中函数返回值一般为void
- 回溯函数终止条件
- 回溯搜索的遍历过程

第77题. 组合

力扣题目链接 中等

给定两个整数 `n` 和 `k`，返回范围 `[1, n]` 中所有可能的 `k` 个数的组合。你可以按 **任何顺序** 返回答案。

示例 1：

```
输入: n = 4, k = 2
输出:
[
  [2,4],
  [3,4],
  [2,3],
  [1,2],
  [1,3],
  [1,4],
]
```

示例 2:

```
输入: n = 1, k = 1
输出: [[1]]
```

提示:

- `1 <= n <= 20`
- `1 <= k <= n`

题解

暴力算法就是打k个for？但是发现没办法写啊，不知道k是多少，所以需要用回溯

用递归来实现for不知道的问题，等到个数够了，就开始回溯

三个步骤：

- 递归函数的返回值以及参数

在这里要定义两个全局变量，一个用来存放符合条件单一结果，一个用来存放符合条件结果的集合。

代码如下：

```C#
vector<vector<int>> result; // 存放符合条件结果的集合
vector<int> path; // 用来存放符合条件结果
```

- 回溯函数终止条件

回溯什么时候开始呢，数组的大小如果达到k，说明我们找到了一个子集大小为k的组合了，找到的这个path存的就是根节点到叶子节点的路径，也就是我们要的组合。

时用result二维数组，把path保存起来，并终止本层递归。

所以终止条件代码如下：

```C#
if (path.size() == k) {
    result.push_back(path);
    return;
}
```

- 单层搜索的过程

回溯法的搜索过程就是一个树型结构的遍历过程，用递归遍历k的同时，也要用for循环用来横向遍历n，递归的过程是纵向遍历。

for循环每次从startIndex开始遍历，然后用path保存取到的节点i。

代码如下：

```C#
for (int i = startIndex; i <= n; i++) { // 控制树的横向遍历
    path.push_back(i); // 处理节点
    backtracking(n, k, i + 1); // 递归：控制树的纵向遍历，注意下一层搜索要从i+1开始
    path.pop_back(); // 回溯，撤销处理的节点
}
```

可以看出backtracking（递归函数）通过不断调用自己一直往深处遍历，总会遇到叶子节点，遇到了叶子节点就要返回。

综上就可以组成完整的算法

```C#
class Solution {
public:
    vector<vector<int>> ans;
    vector<int> path;
    void find(int n,int k,int start){
        if(path.size()==k){
            ans.push_back(path);
            return;
        }
        for(int i=start;i<=n;i++){
            path.push_back(i);
            find(n,k,i+1);
            path.pop_back();
        }
        return;
    }
    vector<vector<int>> combine(int n, int k) {
        find(n,k,1);
        return ans;
    }
};
```

剪枝部分先忽略，等聪明了再看

216.组合总和III

力扣题目链接 中等 15min

找出所有相加之和为 `n` 的 `k` 个数的组合，且满足下列条件：

- 只使用数字1到9
- 每个数字 **最多使用一次**

返回 *所有可能的有效组合的列表* 。该列表不能包含相同的组合两次，组合可以以任何顺序返回。

示例 1:

```
输入: k = 3, n = 7
输出: [[1,2,4]]
解释:
1 + 2 + 4 = 7
没有其他符合的组合了。
```

示例 2:

```
输入: k = 3, n = 9
输出: [[1,2,6], [1,3,5], [2,3,4]]
解释:
1 + 2 + 6 = 9
1 + 3 + 5 = 9
2 + 3 + 4 = 9
没有其他符合的组合了。
```

示例 3:

```
输入: k = 4, n = 1
输出: []
解释: 不存在有效的组合。
在[1,9]范围内使用4个不同的数字，我们可以得到的最小和是1+2+3+4 = 10，因为10 > 1，没有有效的组合。
```

题解

明显和上面的组合是很类似的，遍历1-9，只要加一步判断是否等于要求再输入答案数组即可

C#

```
class Solution {
public:
    vector<vector<int>> ans;
    vector<int> path;//全局变量存答案
    void find(int k,int n,int start,int sum){//第一步确定参数
        if(path.size()==k){//第二步确定终止条件
            if(sum==n)ans.push_back(path);
            return;
        }
        if(sum>=n)return;//剪枝
        for(int i=start;i<=9;i++){//第四步层内遍历
            path.push_back(i);
            find(k,n,i+1,sum+i);//递归
            path.pop_back();//回溯
        }
        return;

    }
    vector<vector<int>> combinationSum3(int k, int n) {
        find(k,n,1,0);
```

```
        return ans;
    }
};
```

17.电话号码的字母组合

力扣题目链接 中等

给定一个仅包含数字 `2-9` 的字符串，返回所有它能表示的字母组合。答案可以按 **任意顺序** 返回。给出数字到字母的映射如下（与电话按键相同）。注意 1 不对应任何字母。

示例 1:

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

示例 2:

```
输入: digits = ""
输出: []
```

示例 3:

```
输入：digits = "2"
输出：["a","b","c"]
```

提示：

- `0 ⩽ digits.length ⩽ 4`
- `digits[i]` 是范围 `['2', '9']` 的一个数字。

题解

同样是不知道有几个digit，也就是不知道循环几次，所以使用回溯法

- 一开始以为一个数字对应三个字母，打算直接用ascall实现，后面发现7和9有四个，老老实实写对应表吧
- 对应表本来考虑从2开始，但是发现没必要，写上空的也行哇

C#

```
class Solution {
public:
    vector<string> ans;
    string letterMap[10]={//这样写就行了，不用加vector
        "",//0
        "",//1
        "abc",//2
        "def",//3
        "ghi",//4
        "jkl",//5
        "mno",//6
        "pqrs",//7
        "tuv",//8
        "wxyz"//9
    };
```

```
    string path;
    void find(string digits,int n)
    {
        if(path.size()==digits.size()){
            ans.push_back(path);
            return;
        }
        for(char x:letterMap[digits[n]-'0']){
            path.push_back(x);
            find(digits,n+1);
            path.pop_back();
        }
        return;
    }
    vector<string> letterCombinations(string digits) {
        if(digits=="")return ans;//要加一个判断是否为空，不然会返回一个0个元素的数组，和答案的要求不同捏
        find(digits,0);
        return ans;
    }
};
```