其实深搜和回溯也是相辅相成，毕竟都用递归

332.重新安排行程

题目地址：https://leetcode-cn.com/problems/reconstruct-itinerary/

给定一个机票的字符串二维数组 [from, to]，子数组中的两个成员分别表示飞机出发和降落的机场地点，对该行程进行重新规划排序。所有这些机票都属于一个从 JFK（肯尼迪国际机场）出发的先生，所以该行程必须从 JFK 开始。

提示：

- 如果存在多种有效的行程，请你按字符自然排序返回最小的行程组合。例如，行程 ["JFK", "LGA"] 与 ["JFK", "LGB"] 相比就更小，排序更靠前
- 所有的机场都用三个大写字母表示（机场代码）。
- 假定所有机票至少存在一种合理的行程。
- 所有的机票必须都用一次 且 只能用一次。

示例 1：

输入：[["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]

输出：["JFK", "MUC", "LHR", "SFO", "SJC"]

示例 2：

输入：[["JFK","SFO"],["JFK","ATL"],["SFO","ATL"],["ATL","JFK"],["ATL","SFO"]]

输出：["JFK","ATL","JFK","SFO","ATL","SFO"]

解释：另一种有效的行程是 ["JFK","SFO","ATL","JFK","ATL","SFO"]。但是它自然排序更大更靠后。

> 由于题目中说明所有的机票都用一次且只能用一次
> 则说明**在回溯算法中存在同一树枝去重的问题.**

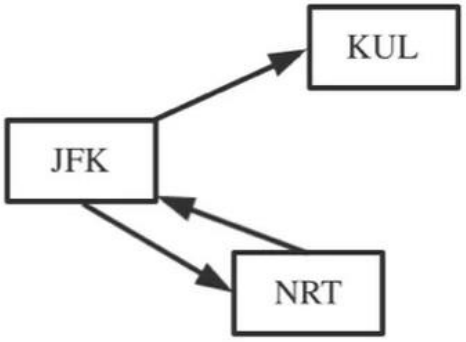
「这道题目有几个难点：」

1. 一个行程中，如果航班处理不好容易变成一个圈，成为死循环
2. 有多种解法，字母序靠前排在前面，让很多同学望而退步，如何该记录映射关系呢？
3. 使用回溯法（也可以说深搜） 的话，那么终止条件是什么呢?
4. 搜索的过程中，如何遍历一个机场所对应的所有机场。

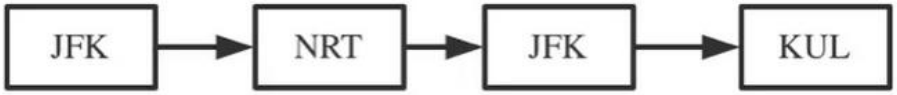
针对以上问题我来逐一解答！

如何理解死循环:

航班信息：[["JFK","KUL"],["JFK","NRT"],["NRT","JFK"]]



输出：["JFK","NRT","JFK","KUL"]



代码随想录

> 在解题过程中，出发机场与到达机场也会重复，如果**在解题过程中没有对集合元素处理好，就会死循环,所以要考虑到每次选择的一个机票的到达机场不能为之前去过的机场**

该记录映射关系

> 一个机场对应多个机场则可以用std::unordered_map，如果对应的机场之间要求有顺序则用std::map或者std::multiset。**(map和multimap的排序是key之间的排序)**
> 因此记录映射关系可以用**unordered_map<string,map<string,int>> targets;**
> **含义表示unordered_map<出发机场,map<到达机场,航班次数>> targets**
> **可以使用航班次数这个字段做相应的增减，来标记该机场是否使用过了**

回溯算法模板的三步骤：

递归函数参数：

```
unordered_map<string,map<string,int>> targets;
bool backtracking(int ticketNum,vector<string>&result);
```

「注意函数返回值我用的是bool！」

> 一般回溯算法都是用void此处用bool是**因为我们只需要找到一个行程**

如何初始化targets和result呢?

```
for(int i=0;i<tickets.size();i++){
 vector<string> tmp=tickets[i];
```

```
 targets[tmp[0]][tmp[1]]++;
}
result.push_back("JFK");
```

递归终止条件:

```
if(result.size()==ticketNum+1){
 return true;
}

```

单层逻辑:

```
unordered_map<string,map<string,int>>::iterator its=targets.find(result.back());
map<string,int> Map=its->second;
map<string,int>::iterator it=Map.begin();
while(it!=Map.end()){
 if(it->second>0){//说明该机票还剩余
 it->second--;
 result.push_back(it->first);
 if(backtracking(ticketNum,result)) return true;
 result.pop_back();
  it->second++;
  }
 }

```

解数独

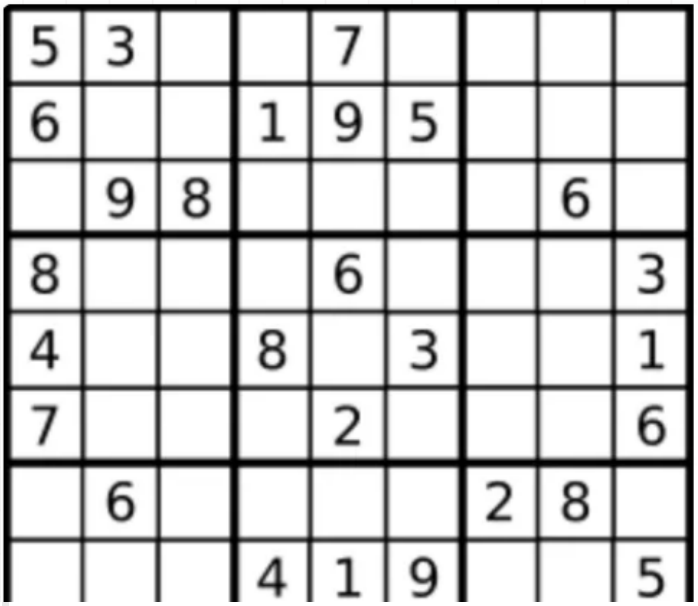
解数独,理解二维递归是关键

37. 解数独

题目地址：https://leetcode-cn.com/problems/sudoku-solver/

编写一个程序，通过填充空格来解决数独问题。

一个数独的解法需遵循如下规则：
数字 1-9 在每一行只能出现一次。
数字 1-9 在每一列只能出现一次。
数字 1-9 在每一个以粗实线分隔的 3x3 宫内只能出现一次。
空白格用 '.' 表示。

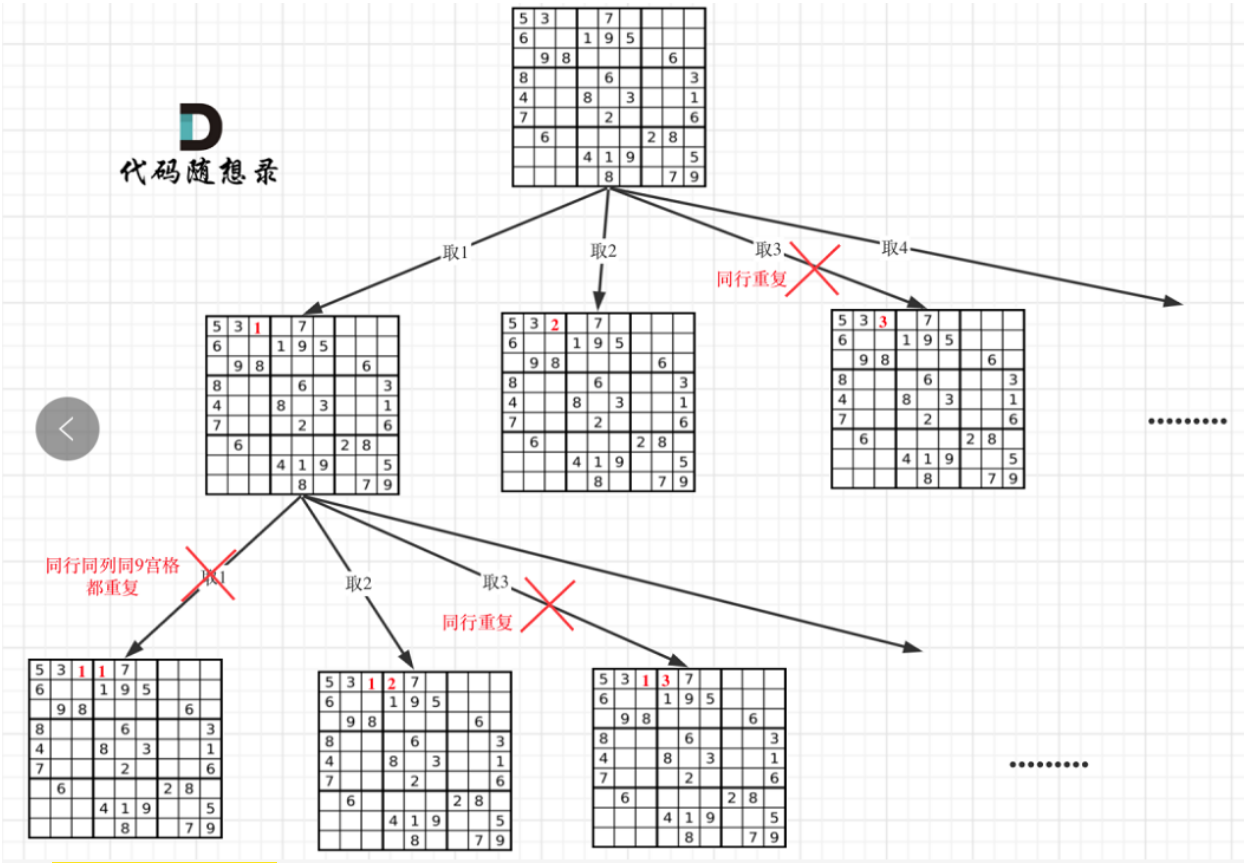


> **棋盘搜索问题**可以使用**回溯法暴力搜索，只不过我们要做的是二维递归**

解释为什么N皇后问题不是二维递归问题?

因为N皇后利用**一个for循环实现一个行的遍历**，而**通过递归实现一个列的遍历**，然后**一行一列确定换后的唯一位置.**

以下为解数独的过程：



递归函数以及参数

bool backtracking(vector<vector<char>>&board)

需要bool返回值类型，因为**一找到符合条件的就立刻返回**

递归终止条件

本题递归不用终止条件，解数独是要遍历整个树形结构寻找可能的叶子节点就返回

单层逻辑

```
bool backtracking(vector<vector<char>>& board){
 for(int i=0;i<board.size();i++){//遍历每一行
 for(int j=0;j<board[i].size();j++){//遍历每一列
 if(board[i][j]!='.')continue;
 for(int k=1;k<=9;k++){//当行与列确定
 if(isValid(board,k, i,j)){//如果在i行j列填入k是否合理
 board[i][j]='0'+k;
 if(backtracking(board)) return true;
 board[i][j]='.';
  }
  }
  return false;//当行与列确定从1到9都试过后，发现都不行那么说明不能找到该问题的解
  }
  }
  return true;//说明该棋盘的所有位置都有相应的数字填入
}

```

判断棋盘是否合法?

```
bool isValid(vector<vector<char>>& board,int k,int i,int j){
 //行
 for(int j_2=0;j_2<board[i].size();j_2++){
 if(board[i][j_2]==k+'.') return false;
 }
 //列
 for(int i_2=0;i_2<board.size();i_2++){
 //if(i_2==i) continue; 由于此时board在board[i][j]处还没有加入k
 if(board[i_2][j]==k+'.')return false;
  }
  //方格
  int row=i/3;
  int col=j/3;
  for(int i_2=row*3;i_2<=row*3+2;i_2++){
  for(int j_2=col*3;j_2<=col*3+2;j_2++){
  // if(i_2==i&&j_2==j)continue;
  if(board[i_2][j_2]==k+'.') return false;
  }
  }
  return true;
}
```