回溯法

jask

09/29/2024

77 组合

[链接](https://leetcode.cn/problems/combinations/)

经典的组合数目题目，就是用递归 + 回溯，添加一个节点到 path 然后递归。

示例代码：

```
class Solution {
public:
vector<vector<int>> result;
vector<int> path;
void helper(int &n,int& k,int startIndex){
    if(path.size()==k){
        result.emplace_back(path);
        return;
    }
    for(int i=startIndex;i<=n-(k-path.size())+1;i++){
        path.emplace_back(i);
        helper(n,k,i+1);
        path.pop_back();
    }
}
    vector<vector<int>> combine(int n, int k) {
        helper(n,k,1);
        return result;
    }
};
```

组合总和

与上述基本相同，注意的是同一个可以被多次计算。

```
class Solution {
public:
vector<vector<int>> result;
vector<int> path;
    void backtracking(vector<int>& candidates,int target,int sum,int startIndex){
        if(sum>target){
            return;
        }
        if(sum==target){
            result.emplace_back(path);
        }
        for(int i=startIndex;i<candidates.size();i++){
            sum+=candidates[i];
            path.emplace_back(candidates[i]);
            backtracking(candidates,target,sum,i);
            sum-=candidates[i];
            path.pop_back();
        }
```

```
        return;
    }
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        backtracking(candidates,target,0,0);
        return result;
    }
};
```

优化版本

```
class Solution {
public:
vector<vector<int>> result;
vector<int> path;
    void backtracking(vector<int>& candidates,int target,int sum,int startIndex){
        if(sum==target){
            result.emplace_back(path);
        }
        for(int i=startIndex;i<candidates.size()&&sum+candidates[i]<=target;i++){
            sum+=candidates[i];
            path.emplace_back(candidates[i]);
            backtracking(candidates,target,sum,i);
            sum-=candidates[i];
            path.pop_back();
        }
        return;
    }
    vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
        ranges::sort(candidates.begin(),candidates.end());
        backtracking(candidates,target,0,0);
        return result;
    }
};
```

17 电话号码的字母组合

```
class Solution {
public:
vector<string> ans;
std::string path;
unordered_map<char,string> key_to_map{
    {'2',"abc"},
    {'3',"def"},
    {'4',"ghi"},
    {'5',"jkl"},
    {'6',"mno"},
    {'7',"pqrs"},
    {'8',"tuv"},
    {'9',"wxyz"}
};
void backtracking(string& digits,int startIndex){
    if(path.length()==digits.length()){
        ans.emplace_back(path);
        return;
    }
    auto map_=key_to_map[digits[startIndex]];
    for(int i=0;i<map_.length();i++){
        path.push_back(map_[i]);
        backtracking(digits,startIndex+1);
        path.pop_back();
    }
    return;
```

```
}
    vector<string> letterCombinations(string digits) {
        if(digits.length()==0){
            return vector<string>{};
        }
        backtracking(digits,0);
        return ans;
    }
};
```

216 组合总和

```
class Solution {
public:
vector<vector<int>> result;
vector<int> path;
    void backtracking(int targetSum,int k,int sum,int startIndex){
        if(path.size()==k){
            if(targetSum==sum){
                result.emplace_back(path);
            }
            return;
        }
        for(int i=startIndex;i<=9&&sum+i<=targetSum;i++){
            sum+=i;
            path.emplace_back(i);
            backtracking(targetSum,k,sum,i+1);
            path.pop_back();
            sum-=i;
        }
    }
    vector<vector<int>> combinationSum3(int k, int n) {
        backtracking(n,k,0,1);
        return result;
    }
};
```

40 组合总和 2

```
class Solution {
public:
vector<vector<int>> result;
vector<int> path;
void backtracking(vector<int>& candidates,int target,int targetSum,int startIndex,int n,vector<bool>& used){
    if(targetSum>target){
        return;
    }
    if(target==targetSum){
        result.emplace_back(path);
        return;
    }
    for(int i=startIndex;i<n;i++){
        if(i>0&&candidates[i]==candidates[i-1]&&used[i-1]==false){
            continue;
        }
        used[i]=true;
        targetSum+=candidates[i];
        path.push_back(candidates[i]);
        backtracking(candidates,target,targetSum,i+1,n,used);
        used[i]=false;
```

```
        targetSum-=candidates[i];
        path.pop_back();
    }
}
    vector<vector<int>> combinationSum2(vector<int>& candidates, int target) {
        sort(candidates.begin(),candidates.end());
        vector<bool> used(candidates.size(),false);
        backtracking(candidates,target,0,0,candidates.size(),used);
        return result;
    }
};
```

131 分割回文串

```
class Solution {
public:
    vector<vector<string>> result;
    vector<string> path;
    bool is_palindrome(const std::string& str,int start,int end){
        for(int i=start,j=end;i<j;i++,j--){
            if(str[i]!=str[j]){
                return false;
            }
        }
        return true;
    }
    void backtracking(const std::string& s,int startIndex){
        if(startIndex>=s.size()){
            result.emplace_back(path);
            return;
        }
        for(int i=startIndex;i<s.size();i++){
            if(is_palindrome(s, startIndex,i)){
                auto str=s.substr(startIndex,i-startIndex+1);
                path.emplace_back(str);
            }else{
                continue;
            }
            backtracking(s, i+1);
            path.pop_back();
        }
    }
    vector<vector<string>> partition(string s) {
        backtracking(s,0);
        return result;
    }
};
```

93 复原 IP 地址

```
class Solution {
public:
vector<string> result;
bool is_valid(string& s,int startIndex,int end){
    if(startIndex>end){
        return false;
    }
    if(s[startIndex]=='0'&&startIndex!=end){
        return false;
    }
```

```
    int num=0;
    for(int i=startIndex;i<=end;i++){
        if(s[i]>'9'||s[i]<'0'){
            return false;
        }
        num=num*10+(s[i]-'0');
        if(num>255){
            return false;
        }
    }
    return true;
}
void backtracking(string& s,int startIndex,int pointNum){
    if(pointNum==3){
        if(is_valid(s,startIndex,s.size()-1)){
            result.emplace_back(s);
        }
        return;
    }
    for(int i=startIndex;i<s.size();i++){
        if(is_valid(s,startIndex,i)){
            s.insert(s.begin()+i+1,'.');
            pointNum++;
            backtracking(s,i+2,pointNum);
            s.erase(s.begin()+i+1);
            pointNum--;
        }else{
            break;
        }
    }
}
    vector<string> restoreIpAddresses(string s) {

        backtracking(s,0,0);
        return result;
    }
};
```

78 子集

```
class Solution {
public:
    vector<vector<int>> result;
    vector<int> path;
    void backtracking(vector<int>& nums,int startIndex){
        result.emplace_back(path);
        if(startIndex>=nums.size()){
            return;
        }
        for(int i=startIndex;i<nums.size();i++){
            path.push_back(nums[i]);
            backtracking(nums,i+1);
            path.pop_back();
        }
    }
    vector<vector<int>> subsets(vector<int>& nums) {
        backtracking(nums, 0);
        return result;
    }
};
```

子集是收集树形结构中树的所有节点的结果。

而组合问题、分割问题是收集树形结构中叶子节点的结果。

90 子集 2

参考前面的组合总和 2，排序 + 判断数组

```
class Solution {
public:
vector<vector<int>> result;
vector<int> path;
void backtracking(vector<int>& nums,int startIndex,vector<bool> &used){
    result.emplace_back(path);
    for(int i=startIndex;i<nums.size();i++){
        if(i>0&&nums[i]==nums[i-1]&&used[i-1]==false){
            continue;
        }
        used[i]=true;
        path.push_back(nums[i]);
        backtracking(nums,i+1,used);
        path.pop_back();
        used[i]=false;
    }
}
    vector<vector<int>> subsetsWithDup(vector<int>& nums) {
        sort(nums.begin(),nums.end());
        vector<bool> used(nums.size());
        backtracking(nums,0,used);
        return result;
    }
};
```

491 非递减子序列

描述：

给你一个整数数组 nums ，找出并返回所有该数组中不同的递增子序列，递增子序列中至少有两个元素。你可以按任意顺序返回答案。

数组中可能含有重复元素，如出现两个整数相等，也可以视作递增序列的一种特殊情况。

```
class Solution {
public:
vector<vector<int>> result;
vector<int> path;
void backtracking(vector<int>& nums,int startIndex){
    if(path.size()>1){
        result.emplace_back(path);
    }
    int uset[201]{};
    for(int i=startIndex;i<nums.size();i++){
        if((path.size()&&nums[i]<path.back())||uset[nums[i]+100]>0){
            continue;
        }
        path.push_back(nums[i]);
        uset[nums[i]+1]++;
        backtracking(nums,i+1);
        path.pop_back();
    }
}
    vector<vector<int>> findSubsequences(vector<int>& nums) {
        backtracking(nums,0);
```

```
        return result;
    }
};
```

46 全排列

要点在于，需要注意到重复元素

```
class Solution {
public:
vector<vector<int>> result;
vector<int> path;
void backtracking(vector<int>& nums,int startIndex,vector<bool>& used){
    if(path.size()==nums.size()){
        result.emplace_back(path);
        return;
    }
    for(int i=0;i<nums.size();i++){
        if(used[i]){
            continue;
        }
        used[i]=true;
        path.push_back(nums[i]);
        backtracking(nums,i+1,used);
        used[i]=false;
        path.pop_back();
    }

}
    vector<vector<int>> permute(vector<int>& nums) {
                vector<bool> used(nums.size());
        backtracking(nums, 0,used);

        return result;
    }
};
```

47 全排列 2

给定一个可包含重复数字的序列 `nums` ，按任意顺序返回所有不重复的全排列。

```
class Solution {
public:
vector<vector<int>> result;
vector<int> path;
void backtracking(vector<int>& nums,int startIndex,vector<bool>& used){
    if(path.size()==nums.size()){
        result.emplace_back(path);
        return;
    }
    for(int i=0;i<nums.size();i++){
        if(i>0&&nums[i]==nums[i-1]&&used[i-1]==false){
            continue;
        }
        if(used[i]){
            continue;
        }
        used[i]=true;
        path.push_back(nums[i]);
        backtracking(nums,i+1,used);
        path.pop_back();
```

```
        used[i]=false;
    }
}
    vector<vector<int>> permuteUnique(vector<int>& nums) {
        sort(nums.begin(),nums.end());
        vector<bool> used(nums.size());
        backtracking(nums, 0, used);
        return result;
    }
};
```

332 重新安排行程

描述:

给你一份航线列表 `tickets` ，其中 `tickets[i] = [fromi, toi]` 表示飞机出发和降落的机场地点。请你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。如果存在多种有效的行程，请你按字典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。

假定所有机票至少存在一种合理的行程。且所有的机票必须都用一次且只能用一次。

```
class Solution {
    //出发机场-> 到达机场-> 航班次数
    std::unordered_map<string,map<string,int>> targets;

public:
    bool backtracking(int ticketNum,vector<string>& result){
        if(result.size()==ticketNum+1){
            return true;
        }
        for(auto& target:targets[result.back()]){
            if(target.second>0){//说明有达到的路线
                result.push_back(target.first);
                target.second--;
                if(backtracking(ticketNum,result)){
                    return true;
                }
                result.pop_back();
                target.second++;
            }
        }
        return false;
    }
    vector<string> findItinerary(vector<vector<string>>& tickets) {
        vector<string> result;
        for(const auto& vec: tickets){
            targets[vec[0]][vec[1]]++;
        }
        result.push_back("JFK");
        backtracking(tickets.size(),result);
        return result;
    }
};
```

重点在于找到合适的描述字典序排序的容器。

51 N 皇后

按照国际象棋的规则，皇后可以攻击与之处在同一行或同一列或同一斜线上的棋子。

n 皇后问题研究的是如何将 n 个皇后放置在 n×n 的棋盘上，并且使皇后彼此之间不能相互攻击。

给你一个整数 n ，返回所有不同的 n 皇后问题的解决方案。

每一种解法包含一个不同的 n 皇后问题的棋子放置方案，该方案中 'Q' 和 '.' 分别代表了皇后和空位。

```
class Solution {
    vector<vector<string>> result;
    bool is_valid(int row,int col,vector<string>& chessboard,int n){
        int count=0;
        for(int i=0;i<row;i++){
            if(chessboard[i][col]=='Q'){return false;}
        }
        for(int i=row-1,j=col-1;i>=0&&j>=0;i--,j--){
            if(chessboard[i][j]=='Q') return false;
        }
        for(int i=row-1,j=col+1;i>=0&& j<n;i--,j++){
            if(chessboard[i][j]=='Q') return false;
        }
        return true;
    }
    void backtracking(int n,int row,vector<string>& chessboard){
        if(row==n){
            result.emplace_back(chessboard);
            return ;
        }
        for(int col=0;col<n;col++){
            if(is_valid(row,col,chessboard,n)){
                chessboard[row][col]='Q';//放置皇后
                backtracking(n,row+1,chessboard);
                chessboard[row][col]='.';//撤销皇后
            }
        }
    }
public:
    vector<vector<string>> solveNQueens(int n) {
        vector<string> chessboard(n,std::string(n,'.'));
        backtracking(n, 0,chessboard);
        return result;
    }
};
```

37 解数独

编写一个程序，通过填充空格来解决数独问题。

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。（请参考示例图）

数独部分空格内已填入了数字，空白格用 '.' 表示。

```
class Solution {
public:
bool backtracking(vector<vector<char>>& board){
    for(int i=0;i<board.size();i++){
        for(int j=0;j<board[0].size();j++){
            if(board[i][j]!='.')continue;
            for(char k='1';k<='9';k++){
                if(is_valid(i,j,k,board)){
                    board[i][j]=k;
                    if(backtracking(board)) return true;
```

```
                    board[i][j]='.';
                }
            }
            return false;
        }
    }
    return true;
}
bool is_valid(int row,int col,char val,vector<vector<char>>& board){
    for(int i=0;i<9;i++){
        if(board[row][i]==val){
            return false;
        }
    }
    for(int i=0;i<9;i++){
        if(board[i][col]==val){
            return false;
        }
    }
    int startRow=(row/3)*3;
    int startCol=(col/3)*3;
    for(int i=startRow;i<startRow+3;i++){
        for(int j=startCol;j<startCol+3;j++){
            if(board[i][j]==val){
                return false;
            }
        }
    }
    return true;
}
    void solveSudoku(vector<vector<char>>& board) {
        backtracking(board);
    }
};
```

980 不同路径 3

在二维网格 grid 上，有 4 种类型的方格：

1 表示起始方格。且只有一个起始方格。
2 表示结束方格，且只有一个结束方格。
0 表示我们可以走过的空方格。
-1 表示我们无法跨越的障碍。

返回在四个方向（上、下、左、右）上行走时，从起始方格到结束方格的不同路径的数目。

每一个无障碍方格都要通过一次，但是一条路径中不能重复通过同一个方格。

```
class Solution {
public:
    int dfs(int x,int y,int left,vector<vector<int>>& grid){
       if(x<0||y<0||x>=grid.size()||y>=grid[0].size()||grid[x][y]<0){
        return 0;
       }
        if(grid[x][y]==2){
            return left==0;
        }
        grid[x][y]=-1;
        int ans=dfs(x+1,y,left-1,grid)+dfs(x-1,y,left-1,grid)+dfs(x,y+1,left-1,grid)+dfs(x,y-1,left-1,grid);
        grid[x][y]=0;
        return ans;
    }
    int uniquePathsIII(vector<vector<int>>& grid) {
```

```
        int sx,sy,left;
        for(int i=0;i<grid.size();i++){
            for(int j=0;j<grid[0].size();j++){
                if(grid[i][j]==1){
                    sx=i;
                    sy=j;
                }
                if(grid[i][j]==0){
                    left++;
                }
            }
        }
        return dfs(sx,sy,left,grid);
    }
};
```