回溯算法问题

算法框架，需要考虑的三个问题：

1. 路径：已经做出的选择；
2. 选择列表：当前可以做的选择；
3. 结束条件：到达决策树底层，无法再做的选择条件。

回溯算法框架：

```
std::vector<int> res;
void backtrack(路径, 选择列表){
  if 满足条件 {
    res.push_back(路径);
    return ;
  }

  for 选择 in 选择列表 {
    做选择
    backtrack(路径,  选择列表)
    撤销选择
  }
}
```

多叉树遍历框架：

```
void traverse(TreeNode root){
  for(TreeNode treenode : root) {
    // 前序遍历操作
    traverse(child);
    // 后序遍历操作
  }
}
```

核心处理代码框架：

```
for 选择 : 选择列表 {
  // 做选择
  将该选择从选择列表移除
  路径.push_back(选择)
  backtrack(路径, 选择列表)
  // 撤销选择
  路径.pop_back(选择)
  将该选择再加入选择列表
}
```

一般情况下选择有以下开始方式：

> 从0开始到末尾，排除已被选择的元素；

全排列

```
// track 为路径
// result 最终结果集
void backtrack(std::vector<int> &nums, std::vector<int> &track,  int len,
std::vector<std::vector<int>> &result) {
  // 终止条件
  if(track.size() == len){
    result.push_back(track);
    return;
  }

  for(int i=0;i<len;i++){
    // 已经存在，继续循环
    if(find(track.begin(), track.end(), numd[i]) != track.end()){
      continue;
    }
    // 选择
    track.push_back(nums[i]);
    // 进入下一层决策树
    backtrack(nums, track, len, result);
    // 撤销选择
    track.pop_back();
  }

}
```

N皇后问题

```
std::vector<std::vector<std::string>> res;

/* 输入棋盘边长 n，返回所有合法的放置 */
std::vector<std::vector<std::string>> solveQueens(int n){
  // '.' 表示空，'Q' 表示皇后，初始化空棋盘。
  std::vector<std::string> board(n, string(n, '.'));
  backtrack(board, 0);
  return res;
}

// 路径：board 中小于 row 的那些行都已经成功放置了皇后
// 选择列表：第 row 行的所有列都是放置皇后的选择
// 结束条件：row 超过 board 的最后一行
void backtrack(vector<string>& board, int row) {
  // 触发结束条件
  if(row == board.size()){
    res.push_back(board);
```

```
      return ;
    }

    int col = board[row].size();
    for(int i=0;i<col;i++){
      if(!isValid(board, row, col)){
        continue;
      }
      board[row][i] = 'Q';
      backtrace(board, row+1);
      boared[row][i] = '.';
    }
  }

  bool isValid(std::vector<std::string>> &board, int row, int col) {
    int n = board.size();
    // 检查该列是否已经放置Q
    for(int i =0;i<n;i++){
      if('Q' == board[i][col]) {
        return false;
      }
    }

    // 检查右上方
    for(int i = row-1,j=col+1;i>=0&&j<n;i--,j++){
      if('Q' == board[i][j]) {
        return false;
      }
    }
    // 检查左上方
    for(int i=row-1,j=col-1;i>=0&&j>=0;i--,j--;){
      if('Q' == board[i][j]) {
        return false;
      }
    }

    return  true;
  }
```