

专业课程实验报告

课程名称：算法分析与设计

开课学期： 2021 至 2022 学年 第 1 学期

专业：软件工程 年级班级：19级3班

学生姓名：冯春霖 学号：222019321062074

实验教师：曹严元

计算机与信息科学学院 软件学院

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验项目名称 | | 回溯法 | | | |
| 实验时间 | | 2021 年 12 月 8 日 | 实验类型 | | □验证性 □设计性 □综合性 |
| 一、实验目的   1. 掌握回溯法的基本思想方法； 2. 了解适用于用回溯法求解的问题类型，并能设计相应回溯法算法； 3. 掌握回溯法算法复杂性分析方法，分析问题复杂性。   二、实验要求   1. 预习实验指导书及教材的有关内容，掌握回溯法的基本思想； 2. 严格按照实验内容进行实验，培养良好的算法设计和编程的习惯； 3. 认真听讲，服从安排，独立思考并完成实验。 | | | | | |
| 三、实验内容与设计（主要内容，操作步骤、算法描述或程序代码）  **对回溯算法的理解：**  回溯算法就是DFS算法的一种表现形式，本质上是对暴力穷举的优化。  解决回溯问题，实际上就是一个决策树的遍历过程，对于决策树，有三个方面的考量：1：路径，也就是已经做出的选择；2：选择列表，也就是当前可以做的选择；3：结束条件，也就是到达决策树的叶节点，无法再做选择的条件。  以N皇后问题为例，对于进行到第i层的N皇后问题(i < N)其路径就是[1, i - 1]层上所有已放置的皇后，选择列表即为该层的所有棋盘格的坐标，因为每个棋盘格均可用来放置皇后，结束条件为已遍历到N + 1层，超出棋盘范围。  **选做问题1. 实现n皇后问题的回溯算法并进行时空复杂度分析**  **1. 选择合适的数据结构来表示问题**  N皇后问题需要构造一个N \* N的棋盘来放置皇后，故需要一个二维字符数组来模拟棋盘。  **2. 确定出对问题的解空间树的深度优先搜索停止的条件（即开始回溯的条件）**  回溯的条件有：  a) 已经构造出可行的含有N个皇后的棋盘  b) 当前行所有位置(选择空间)均已尝试过  **3. 根据回溯法的基本原理，构造决策树如图（以4皇后为例）**    在此种优化情况下，搜索到不符合条件的棋盘位置时立即进行回溯剪枝以消除不必要的搜索，减少程序运行所需时间  **4. 根据N皇后的原理写出伪代码**    **5. 实现N皇后问题的代码(JavaScript语言)**  /\*\*   \* @param {number} n // 输入n为要求n皇后的棋盘大小   \* @return {string[][]} // 返回一个二维字符串数组，为n皇后的所有可行解   \*/  var NQueens = function (n) {      // 创建一个n \* n的二维数组作为棋盘，默认用 . 进行填充，证明该位置没有皇后      let board = new Array(n).fill(0).map(\_ => new Array(n).fill('.'))      // res储存所有的可行解      let res = []      // 回溯函数，参数row表示当前正在尝试填充棋盘的第row + 1行      const backtrack = row => {          // 参数row超过棋盘行数达到n时证明整个棋盘已经被n个完全填充，将现在的棋盘状态填入结果          if (row === n) {              // 创建一个当前棋盘的拷贝              let subRes = []              board.forEach(e => subRes.push(e.join('')))              // 将拷贝存入结果              res.push(Array.from(subRes))              // 满足终止条件，到达程序出口，回溯到上一层              return          }          // 遍历所在行的所有列，尝试使用皇后进行填充          for (let col = 0; col < n; col++) {              // 如果当前位置可以放皇后              if (isValid(row, col)) {                  // 在当前位置放一个皇后                  board[row][col] = 'Q'                  // 在下一行进行尝试，进入决策树的下一层                  backtrack(row + 1)                  // 移除当前位置的皇后，继续在本层的兄弟节点进行尝试                  board[row][col] = '.'              }          }          // 当前行的所有位置已全部尝试，回溯到上一层      }      // 判断当前位置是否能够放置皇后的函数，参数为尝试放置皇后的位置，能放置则返回true      const isValid = (row, col) => {          // 判断当前位置所在列是否有皇后          for (let i = 0; i < row; i++) {              if (board[i][col] === 'Q') return false          }          // 判断当前位置所在副对角线是否有皇后          for (let i = row - 1, j = col + 1; i >= 0 && j < n; i--, j++) {              if (board[i][j] === 'Q') return false          }          // 判断当前位置所在主对角线是否有皇后          for (let i = row - 1, j = col - 1; i >= 0 && j >= 0; i--, j--) {              if (board[i][j] === 'Q') return false          }          return true      }      // 程序主体，调用backtrack从第一行开始尝试在棋盘上放置皇后      backtrack(0)      // 返回所有的可能的放置方法      return res  }  **6. 时空复杂度分析**  时间复杂度：对于最朴素的暴力解法，在棋盘的N行N列上均有两种选择，放置或者不放置皇后，其时间复杂度为O(N ^ N)，使用回溯法解N皇后问题，需要遍历棋盘的所有位置，决策树共N层，每层的选择列表有N个，共N \* N个选择，对于每个选择，其时间复杂度为常数，在回溯过程中进行剪枝，每层仅保留可用的回溯选择，数学归纳法推知，带剪枝的回溯算法的时间复杂度为O(N!)  空间复杂度：为了构造抽象的N \* N棋盘，需要使用N \* N的二维数组，空间复杂度为O(N)，为了储存所有的可行解，需要一个数组来储存棋盘，由于N皇后解的数量无法进行公式推导且难以归纳，故在N的值较大时空间复杂度也较为巨大 | | | | | |
| 四、测试数据和执行结果 （在给定数据下，执行操作、算法和程序的结果，可使用数据、图表、截图等给出）  测试环境：VSCode + Node.js 14.14  N皇后问题的回溯解法  测试使用代码如下，即分别求6皇后问题和8皇后问题的所有可行解和解的个数  console.log(NQueens(6), NQueens(6).length)  console.log(NQueens(8), NQueens(8).length)  输出结果如下      可以看到该算法可以正确的得出结果，6皇后问题有4个解，8皇后问题有92个解 | | | | | |
| 五、实验结果分析及总结（对实验的结果是否达到预期进行分析，总结实验的收获和存在的问题等）  通过本次实验，我对回溯算法有了更深刻的了解，包括：  1. 掌握了了回溯的概念，对回溯算法的基础知识有了较好的了解，对回溯算法的决策树(解空间树)、路径、选择列表和结束条件等概念有了清晰的认识，对回溯算法与递归、深度优先搜索等算法的关系有了清晰的认识  2. 能够画出简单回溯算法的解空间树，对抽象的算法问题进行实际分析  3. 能够较好的运用回溯算法解决学习生活中遇到的实际问题  4. 对所学知识加以运用，使用JavaScript实现了使用回溯算法实现的求解N皇后问题的结果和可行解数量的代码 | | | | | |
| 教  师  评  阅 | 实验内容和设计（A-E）： | | |  | |
| 操作过程、算法或代码（A-E）： | | |  | |
| 实验结果（A-E）： | | |  | |
| 实验分析和总结（A-E）： | | |  | |
| 实验成绩（A-E）：  反馈评语： | | | | |