**暨南大学研究生课程设计报告**

**学生姓名：覃俊仁 学号：202034511031**

**学院：网络空间安全学院 专业：网络空间安全**

**课程名称：《算法分析与设计》 课程设计名称：N皇后问题**

**类型: 验证-设计-综合 时间：2020年12月 20 日**

**课程设计内容：**

**皇后问题是指的棋盘要摆个皇后（以下称棋子），要求任何两个棋子不同行、不同列、也不在同一条斜线(两个棋子成)上。给定一个整数，返回皇后的摆法有多少种。**

**算法描述：**

1. **递归及动态规划：**

**可将问题转化为求函数，函数为前行的棋子位置已固定，第行到第行可以摆放棋子位置的个数。定义的取值为第行棋子位置，可分解为情况下的个子问题的可行解个数的和。可递归地定义为：**

1. **回溯法：**

**对于棋盘中的每行，自上而下开始按行尝试放置棋子，如果所有行都成功放置了一个棋子，则有可能是其中一个解。对于第行，判断第列放置棋子后是否会与前面行所放置的棋子冲突，，如果发生冲突，考虑行中棋子的下一个位置，如果不发生冲突，则将第行的棋子摆放到第列中，接着尝试摆放第行的棋子。当第行尝试了所有可能的j取值后，便回溯到行，对第行下一个可能的棋子的摆放位置进行尝试。**

**源程序：**

|  |
| --- |
| * **方案1：递归和动态规划**   **#include <iostream>**  **int same\_line(int quene[], int now) { //判断是否同列**  **int q = 0;**  **for (int i = 1; i < now; i++) {**  **if (quene[i] == quene[now]) {**  **q = 1; break;**  **}**  **}**  **return q;**  **}**  **int same\_xie(int quene[], int now) {//判断是否同一斜线**  **int q = 0;**  **for (int i = 1; i < now; i++) {**  **if (abs(i - now) == abs(quene[i] - quene[now])) {**  **q = 1; break;**  **}**  **}**  **return q;**  **}**  **int IS\_Correct(int quene[], int now) {**  **if (same\_line(quene, now))return 0;**  **else if (same\_xie(quene, now))return 0;**  **else return 1;**  **}**  **int queen(int quene[],int now, int n ) {**  **int q = 0;**  **if (!IS\_Correct(quene, now - 1)) {**  **return 0;**  **}**  **if (now > n) {**  **return 1;**  **}**  **for (int i = 1; i <= n; i++) {**  **quene[now] = i;**  **q += queen(quene,now + 1, n );**  **}**  **return q;**  **}**  **int NQueens(int n) {**  **int \*quene=(int \*)malloc((n+1)\*sizeof(int));**  **return queen(quene,1, n);**  **}**  **int main() {**  **std::cout << NQueens(8);**  **}**   * **方案2：回溯法**   **#include <iostream>**  **int same\_line(int quene[], int now) {//判断是否同列**  **int q = 0;**  **for (int i = 1; i < now; i++) {**  **if (quene[i] == quene[now]) {**  **q = 1; break;**  **}**  **}**  **return q;**  **}**  **int same\_xie(int quene[], int now) {//判断是否同一斜线**  **int q = 0;**  **for (int i = 1; i < now; i++) {**  **if (abs(i - now) == abs(quene[i] - quene[now])) {**  **q = 1; break;**  **}**  **}**  **return q;**  **}**  **int IS\_Correct(int quene[], int now) {**  **if (same\_line(quene, now))return 0;**  **else if (same\_xie(quene, now))return 0;**  **else return 1;**  **}**  **int NQueens(int n) {**  **int \*quene = (int \*)malloc((n + 1) \* sizeof(int));**  **for (int i = 0; i < n+1; i++){**  **quene[i] = 0;**  **}**  **int q = 0;**  **int now = 1;**  **while (now) {**  **if (now > n) {**  **++q;**  **--now;**  **continue;**  **}**  **++quene[now];**  **if (quene[now] > n) {**  **quene[now] = 0;**  **--now;**  **continue;**  **}**  **if (IS\_Correct(quene, now)) {**  **++now;**  **}**  **}**  **return q;}**  **int main() {**  **std::cout << NQueens(8);**  **}** |

**时间复杂度与空间复杂度的详细分析：**

**（1）方案1：递归和动态规划**

**A、时间复杂度：**

**我们在分解问题时将总体规模为的问题划分为个规模为的子问题。在将问题分解之前，对层进行循环判断以判断其是否合法。而在进行分解问题与合并解的时候，则进行一个大小为的循环，但其中所生成的个子问题由于进行了是否合法的判断从而起到了减枝的效果，所以实际上最多只有个规模为的子问题。因此有以下渐进时间复杂度：**

**由上可知时间复杂度为。**

1. **空间复杂度：**

**由于递归深度为并且存储空间为，因此空间复杂度为。**

**（2）方案2：回溯法**

**A、时间复杂度：**

**由代码可知，每一层的规模为，进行尝试的可能位置为，对层进行循环判断以判断其是否合法，如果合法，所进入的下一层规模变为，而在最坏情况下，所进入的下一层规模变为，下一层完成所有尝试后，回溯到本层继续进行下一个可能位置的尝试。所以有以下渐进时间复杂度：**

**由此可知时间复杂度为 。**

**B、空间复杂度：**

**由于使用了N的辅助存储空间，所以空间复杂度为**

**算法设计细节的具体分析、运行结果分析和截图 (不少于400字)：**

1. **递归及动态规划：**
2. **设计细节分析：**

**从算法描述中所给出的定义可知，正是所需求N皇后问题的解。在的递归代码中，依照所给出的递归定义，第一我们对前个棋子位置是否合法进行判断。由于上级递归中已经确定了前1到行棋子位置的摆放合法，因此我们仅需对第行是否与第1行到第行的棋子位置冲突进行判断。若前行中棋子位置已经发生了冲突，则该路径不合法，返回0。第二我们对前面放置的棋子个数进行判断，若前面放置的棋子个数为，该条路径即为N皇后问题的一个解，返回1。若不为则以该层规模n进行一次大小为n的循环，将原有问题拆分为n个子问题，为第行选定一个棋子摆放位置，进行下一层子递归的值，将所得结果进行累加便能求出的解的个数，并将该结果返回给上一层递归。**

1. **运行结果分析：**

**在**[**https://leetcode-cn.com/problems/n-queens-ii/**](https://leetcode-cn.com/problems/n-queens-ii/)**中代码提交进行OJ判断，通过了所有的测试用例（4~11），验证了代码的正确性，且本代码执行用时几乎为4ms。（结果如下图所示）**



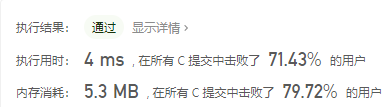
**递归及动态规划运行结果**

1. **回溯法：**
2. **设计细节分析：**

**首先动态定义一个大小为+1的数组来存储每行所尝试的棋子位置，通过自增方式对棋子位置进行遍历，并通过初始化算法将该数组中所有元素值设为0。接着定义指针now作为当前尝试摆放棋子位置的行数，摆放行数从1起始，因此当now=0时，即为对所有可能摆放进行了回溯，循环结束，因此循环条件即为now为真（now！=0）。在循环结构中，首先对指针now是否大于n进行判断，若为真，则已经对一组n个棋子进行摆放尝试，由此可得其中一组解，此时指针now回溯到上一状态，即now-1。并对此状态下一个可能摆放位置进行尝试，即将quene[now]自增（++queen[now]），这表示从余下第now行中列数进行尝试。若quene[now]>n，则第now行已经完成了对所有可能摆放位置的尝试，此时将quene[now]还原为0，以防止对之后不同序列1~n尝试时发生冲突情况，避免error，同时将指针now回溯到上一状态（now-1）。若，则对第now行第quene[now]列摆放棋子操作是否合法进行判断。因为在进行now行的尝试时，前面状态所给出棋子位置必定合法，因此只需要判断第now行的棋子位置是否与前面各行所摆放的棋子位置冲突，若为真，则进行下一状态的尝试，否则尝试当前状态的其余位置。**

1. **运行结果分析：**

**在**[**https://leetcode-cn.com/problems/n-queens-ii/**](https://leetcode-cn.com/problems/n-queens-ii/)**中代码提交进行OJ判断，通过了所有的测试用例（4~11），验证了代码的正确性，且本代码执行用时几乎为4ms。（结果如下图所示）**



**回溯法运行结果**

**个人总结 (不少于200字)：**

**在解决多段图问题算法的编写工作中，我的收获颇丰。首先增进了算法思想，对基础将n规模问题划分为n个子问题的算法思路有了更深的理解，这对于自身往后的工作学习上都给予了更大的帮助。其次对课堂上甘文生老师所讲的递归、动态规划和回溯法有了更深的理解，学到了新的知识的同时更获得了更多往后研究工作中算法编写的新思路。更开拓了往后算法编写时的视野，使得往后在进行实际算法编写或算法改进工作时能使算法获得更低的时间复杂度和空间复杂度，为往后算法改进工作及算法编写工作提供了帮助。最后，我同样也在这次project中熟悉了算法研究工作的基本步骤和核心思路，为将来的研究生学习提供了引导。**