**N皇后实验报告**

1. 算法说明

import time

n = int(input("input number of queens:"))

g = [['.' for \_ in range(n)] for \_ in range(n)]

col = [False]\*n

dg = [False]\*(2\*n)

udg = [False]\*(2\*n)

N = 0

def dfs(u):

    global N  #声明 N 为全局变量

    if u == n:

        #打印当前的解

        for row in g:

            print(''.join(row))

        print()

        N += 1  #在找到一个有效解时增加计数

        return

    for y in range(n):

        #检查是否可以放置皇后

        if not col[y] and not dg[y-u+n] and not udg[y+u]:

            col[y] = True

            dg[y-u+n] = True

            udg[y+u] = True

            g[u][y] = 'Q'

            dfs(u+1)

            #回溯，重置状态

            col[y] = False

            dg[y-u+n] = False

            udg[y+u] = False

            g[u][y] = '.'

if n < 4:

    print("invalid input")

else:

    start\_time = time.time()  # 开始时间

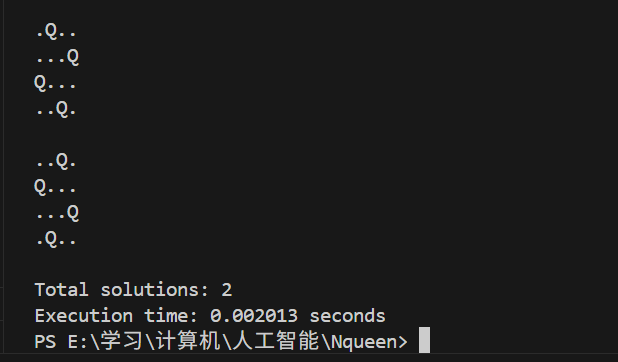
    dfs(0)

    print("Total solutions:", N)  #输出解的总数量

    end\_time = time.time()    # 结束时间

    print(f"Execution time: {end\_time - start\_time:.6f} seconds")

1. 实验结果



n=4

文本

AI 生成的内容可能不正确。

n=5

文本

AI 生成的内容可能不正确。

n=6

文本

AI 生成的内容可能不正确。

n=7

文本

AI 生成的内容可能不正确。

n=8

图表, 折线图

AI 生成的内容可能不正确。

时间增长曲线

分析：剪枝在很多情况下能大大减少无效的搜索分支，所以实际运行远远快于O(n!)，尤其在n比较小时（如n ≤ 12）效率很好。

但是理论复杂度还是不能突破O(n!) 这个上限，因为在最坏情况下，还是可能遍历所有有效排列。

1. 优化思路

1. 减少全局变量使用

现在用的是 global N，可以改成函数内部变量 count 用 nonlocal 管理，这样代码更整洁、更安全。