63010279 ณภัทร จิรารัตนกุลชัย

**N-queens Problem (Recursive VS Iterative)**

1) Source Code

1.1) Iterative

import time

class Board:

    def \_\_init\_\_(self, size):

        self.N = size

        self.queens = [] *# list of columns, where the index represents the row*

    def is\_queen\_safe(self, row, col):

        for r, c in enumerate(self.queens):

            if r == row or c == col or abs(row - r) == abs(col - c):

                return False

        return True

    def print\_the\_board(self):

        print ("solution:")

        for row in range(self.N):

            line = ['.'] \* self.N

            if row < len(self.queens):

                line[self.queens[row]] = 'Q'

            print(''.join(line))

    def solution(self):

        self.queens = []

        col = row = 0

        while True:

            while col < self.N and not self.is\_queen\_safe(row, col):

                col += 1

            if col < self.N:

                self.queens.append(col)

                if row + 1 >= self.N:

                    self.print\_the\_board()

                    self.queens.pop()

                    col = self.N

                else:

                    row += 1

                    col = 0

            if col >= self.N:

*# not possible to place a queen in this row anymore*

                if row == 0:

                    return *# all combinations were tried*

                col = self.queens.pop() + 1

                row -= 1

if \_\_name\_\_ == "\_\_main\_\_":

    q = Board(5)

    q.solution()

Credit : [algorithm - Avoid duplicates in N Queen iterative solutions (No Recursion Allowed) - Stack Overflow](https://stackoverflow.com/questions/42318343/avoid-duplicates-in-n-queen-iterative-solutions-no-recursion-allowed)

1.2) Recursive

import time

def is\_safe(board, x, y, c):

    for p in [board[i] for i in range(0, c)]:

        if p[0] == x or p[1] == y or x + y == p[0] + p[1] or x - y == p[0] - p[1]:

            return False

    return True

def nqueen\_nrec(n):

    num = 0

    c = 0

    step = [0 for x in range(0, n + 1)]

    board = [(x, x) for x in range(0, n)]

    while c != -1:

        if c == n:

            num += 1

            print(board)

            c -= 1

            step[c] += 1

        elif step[c] == n:

            c -= 1

            step[c] += 1

        elif is\_safe(board, step[c], c, c):

            board[c] = (step[c], c)

            c += 1

            step[c] = 0

        else:

            step[c] += 1

    print("Number of solution = {}".format(num))

if \_\_name\_\_ == "\_\_main\_\_":

    user\_input = int(input("Enter input : "))

    t0 = time.time()

    c = nqueen\_nrec(user\_input)

    t1 = time.time()

    print(f'Time(Iteration): {t1-t0:.8} seconds')

Credit : [algorithm - Avoid duplicates in N Queen iterative solutions (No Recursion Allowed) - Stack Overflow](https://stackoverflow.com/questions/42318343/avoid-duplicates-in-n-queen-iterative-solutions-no-recursion-allowed)

2) CPU Memory

CPU Memory

2.1) Iterative



2.2) Recursive



3 & 4) ผลการรัน และการจับเวลา & Input 4 - 12

|  |  |
| --- | --- |
| Iterative | Recursive |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

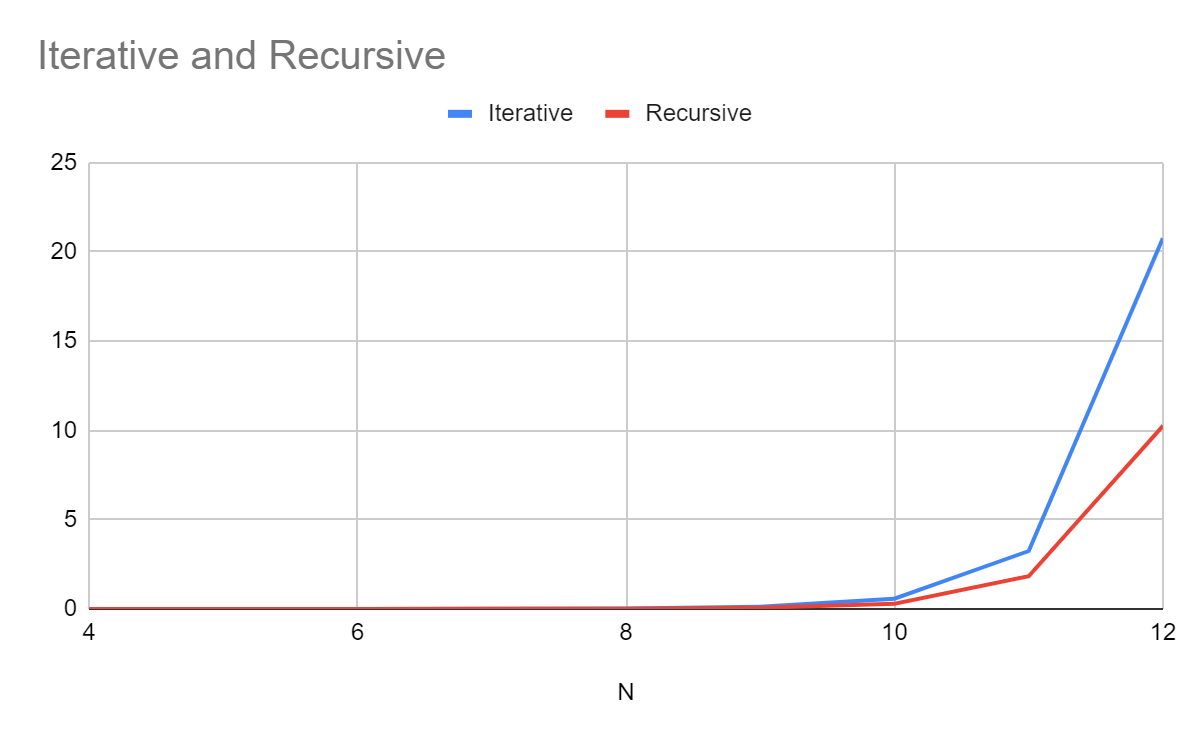
5) แหล่งอ้างอิง

Credit : [algorithm - Avoid duplicates in N Queen iterative solutions (No Recursion Allowed) - Stack Overflow](https://stackoverflow.com/questions/42318343/avoid-duplicates-in-n-queen-iterative-solutions-no-recursion-allowed)

6) ตารางบันทึกผล

|  |  |  |
| --- | --- | --- |
| N | Iterative | Recursive |
| 4 | 0 | 0 |
| 5 | 0.0010011196 | 0.0010015965 |
| 6 | 0.0009996891 | 0.0010006428 |
| 7 | 0.0070006847 | 0.00299716 |
| 8 | 0.02798748 | 0.012988806 |
| 9 | 0.11500311 | 0.063005924 |
| 10 | 0.5700407 | 0.28202057 |
| 11 | 3.242239 | 1.8301368 |
| 12 | 20.760544 | 10.249582 |

7) กราฟเปรียบเทียบเวลาในการรัน ทั้ง สอง อัลกอริทึม



8) การวิเคราะห์ผลลัพธ์ที่ได้

สรุปผลได้ว่าแบบRecursiveจะใช้เวลาประมวลผลและใช้Memoryมากกว่าแบบIterative

และRecursiveใช้ CPU มากกว่า(แต่ห่างกันไม่ค่อยมาก)