CODE:

from queue import Queue

class NQueens:

def \_init\_(self, size):

self.size = size

def solve\_dfs(self):

if self.size < 1:

return []

solutions = []

stack = [[]]

while stack:

solution = stack.pop()

if self.conflict(solution):

continue

row = len(solution)

if row == self.size:

solutions.append(solution)

continue

for col in range(self.size):

queen = (row, col)

queens = solution.copy()

queens.append(queen)

stack.append(queens)

return solutions

def solve\_bfs(self):

if self.size < 1:

return []

solutions = []

queue = Queue()

queue.put([])

while not queue.empty():

solution = queue.get()

if self.conflict(solution):

continue

row = len(solution)

if row == self.size:

solutions.append(solution)

continue

for col in range(self.size):

queen = (row, col)

queens = solution.copy()

queens.append(queen)

queue.put(queens)

return solutions

def conflict(self, queens):

for i in range(1, len(queens)):

for j in range(0, i):

a, b = queens[i]

c, d = queens[j]

if a == c or b == d or abs(a - c) == abs(b - d):

return True

return False

def print(self, queens):

for i in range(self.size):

print(' ---' \* self.size)

for j in range(self.size):

p = 'Q' if (i, j) in queens else ' '

print('| %s ' % p, end='')

print('|')

print(' ---' \* self.size)

def main():

print('.: N-Queens Problem :.')

size = int(input('Please enter the size of board: '))

print\_solutions = input('Do you want the solutions to be printed (Y/N): ').lower() == 'y'

n\_queens = NQueens(size)

dfs\_solutions = n\_queens.solve\_dfs()

bfs\_solutions = n\_queens.solve\_bfs()

if print\_solutions:

for i, solution in enumerate(dfs\_solutions):

print('DFS Solution %d:' % (i + 1))

n\_queens.print(solution)

for i, solution in enumerate(bfs\_solutions):

print('BFS Solution %d:' % (i + 1))

n\_queens.print(solution)

print('Total DFS solutions: %d' % len(dfs\_solutions))

print('Total BFS solutions: %d' % len(bfs\_solutions))

if \_\_name\_\_ == '\_\_main\_\_':

main()

CODE OUPUT:



