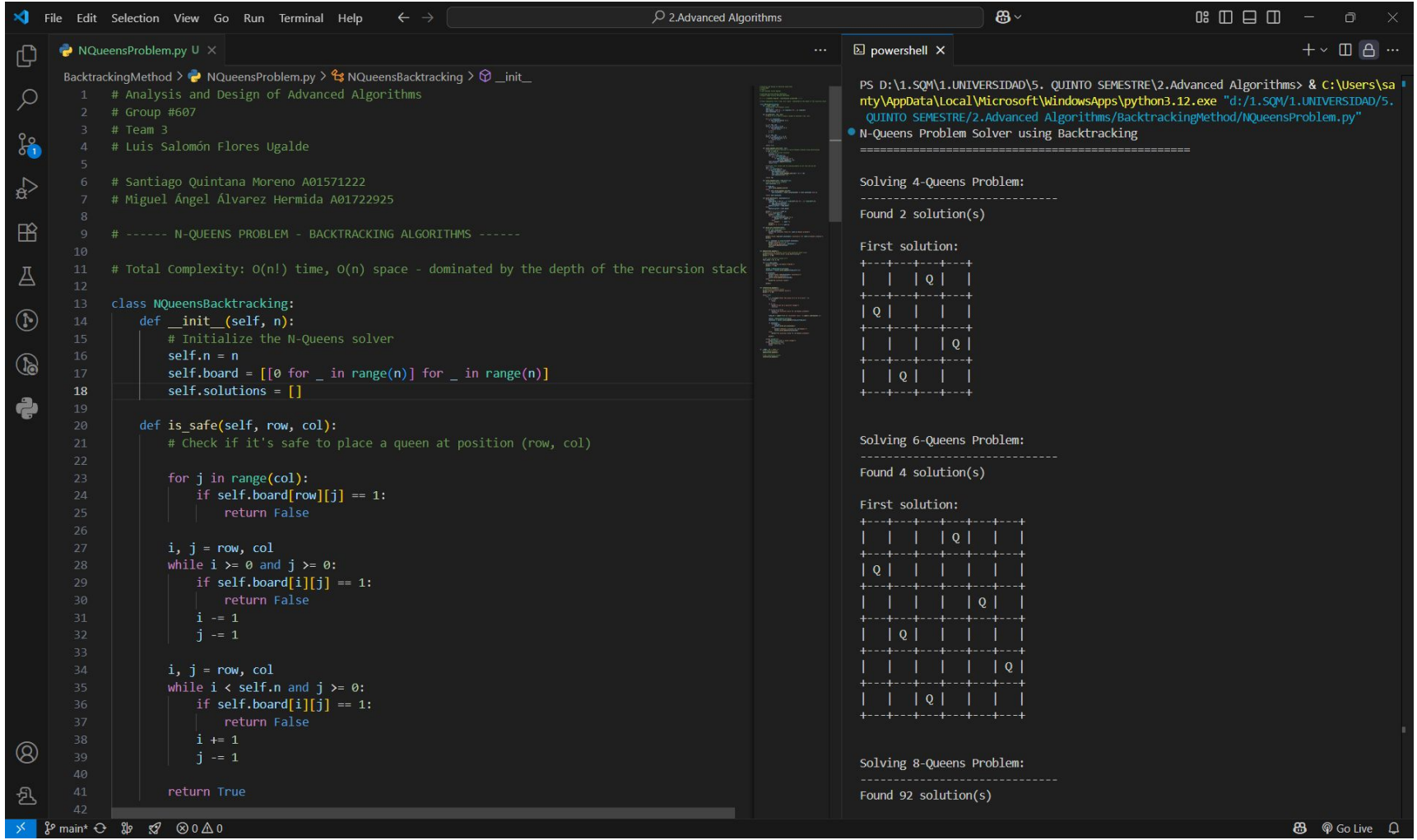
Tecnológico de Monterrey

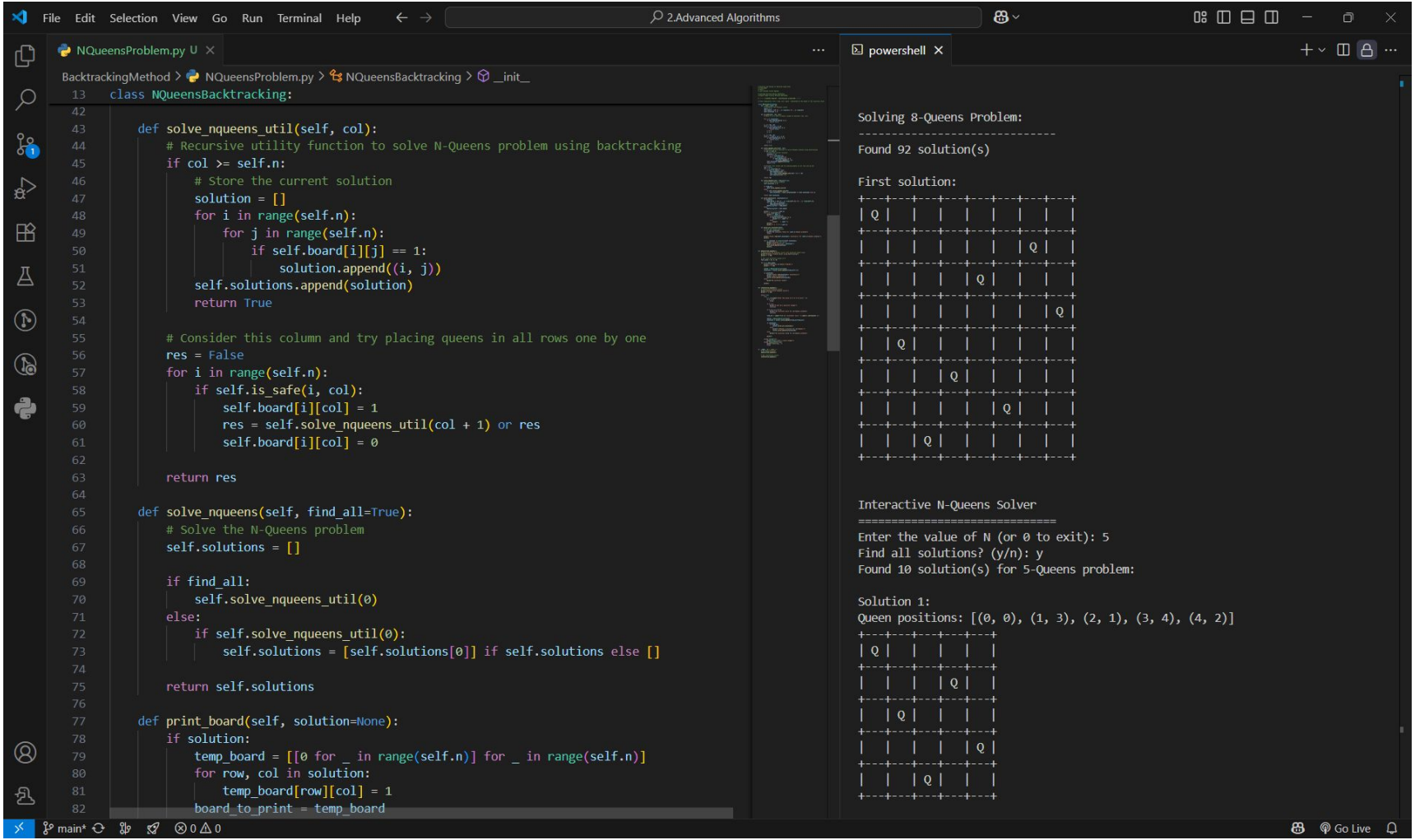
Tecnológico de Monterrey - Campus Monterrey
School of Engineering and Sciences
Engineering in Computational Technologies
Analysis and Design of Advanced Algorithms

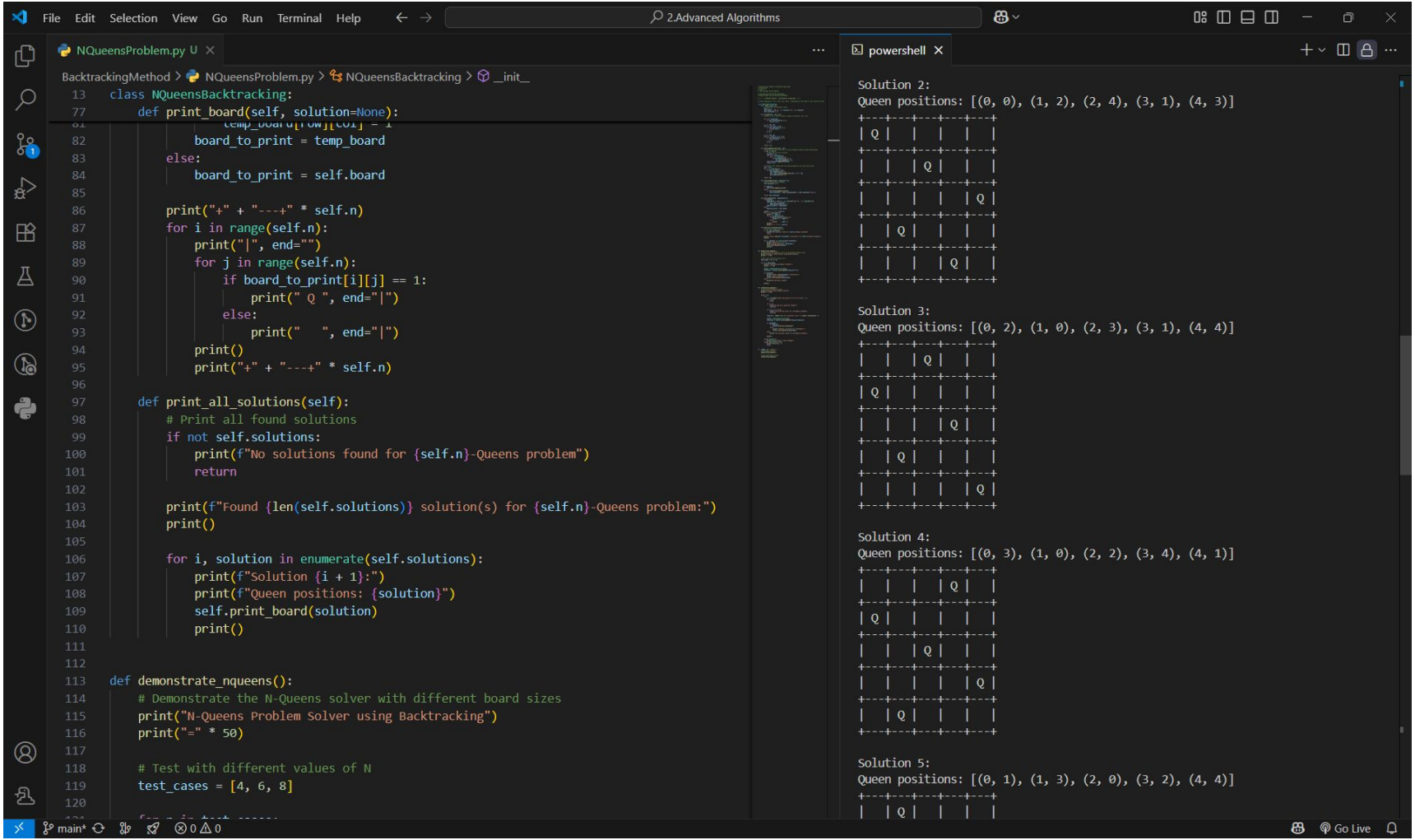
Homework 5:
N-Queens Problem using Backtracking Method

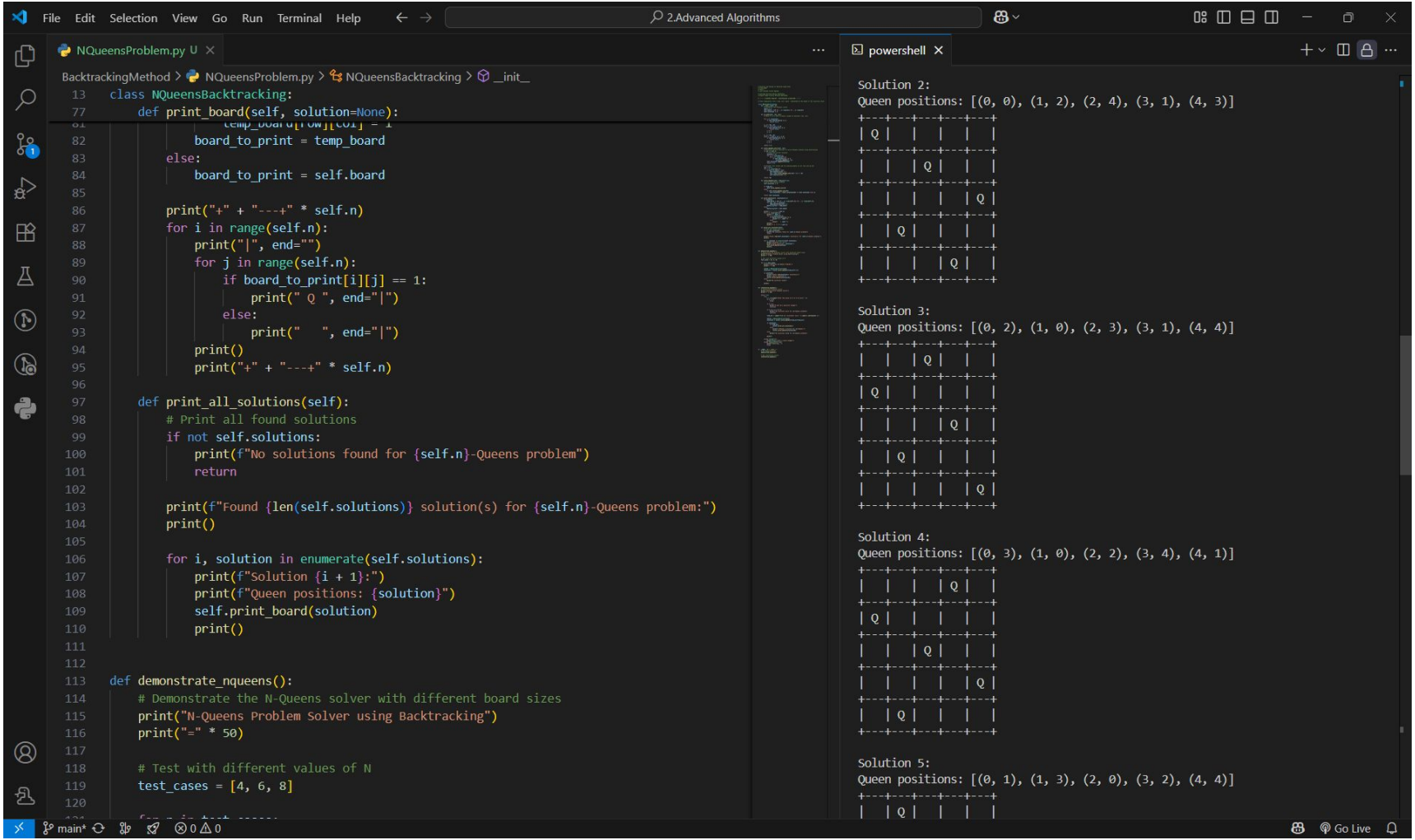
Group: 607
Team #3
Luis Salomón Flores Ugalde

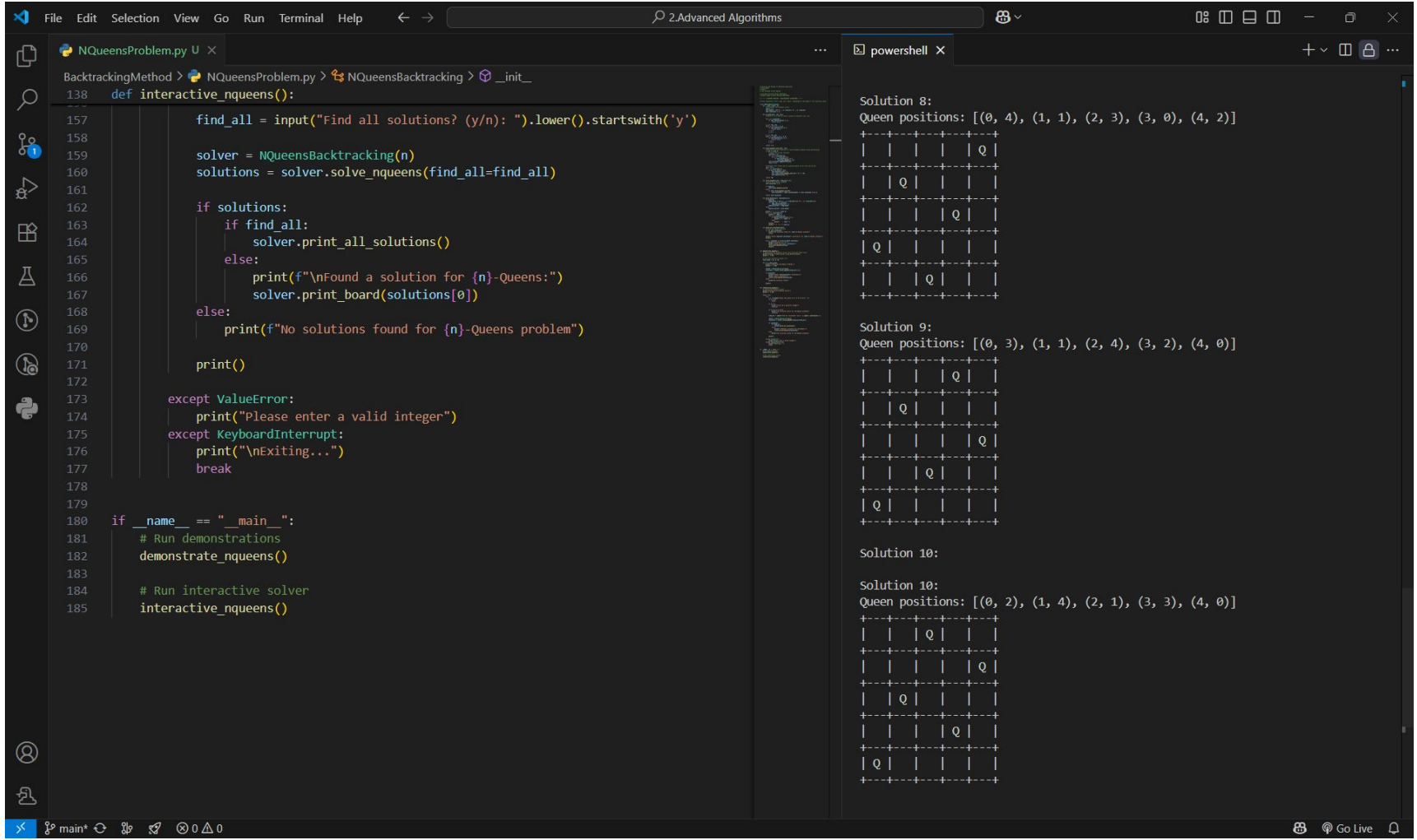
Santiago Quintana Moreno A01571222
Miguel Ángel Álvarez Hermida a01722925

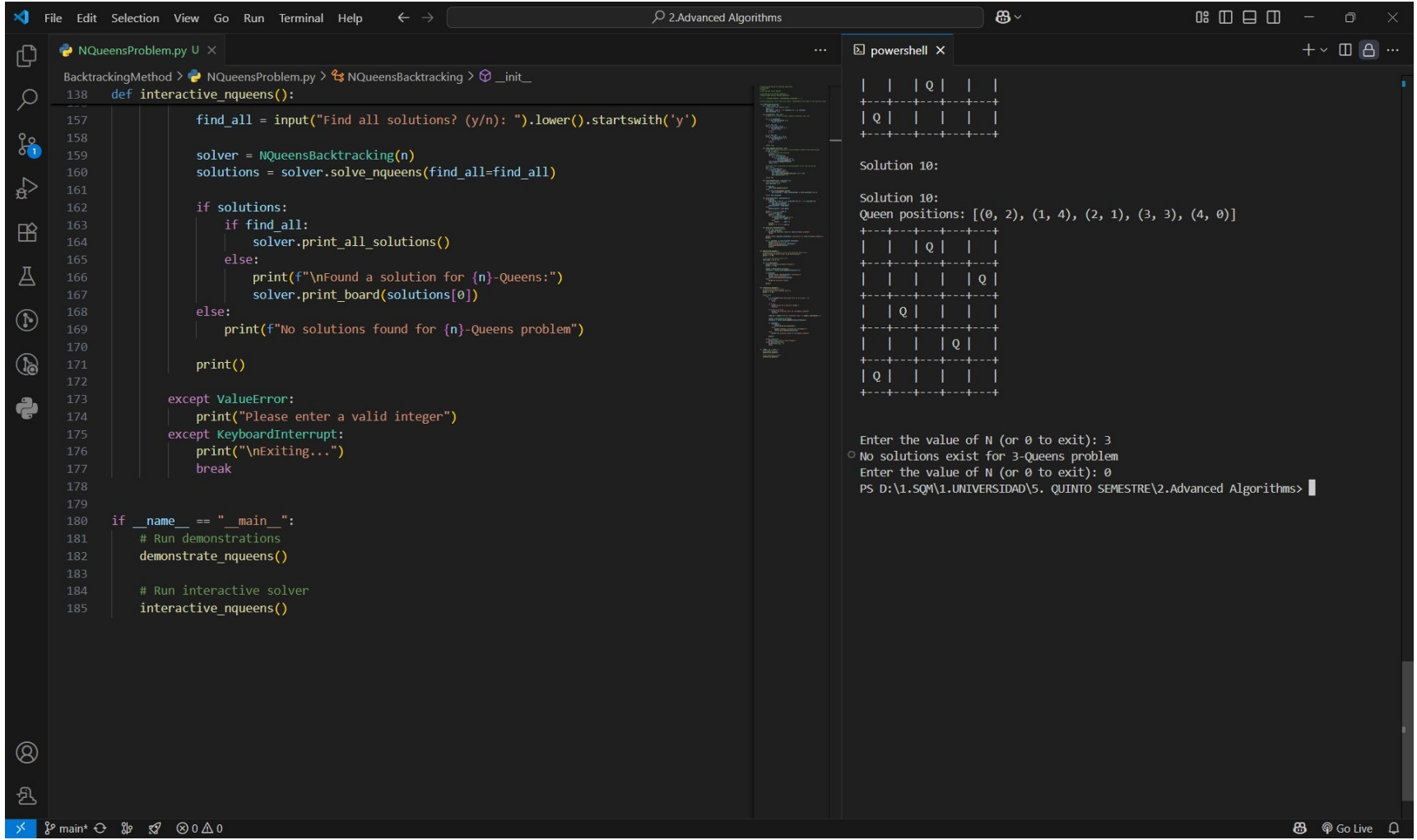

File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
NQueensProblem.py U
BacktrackingMethod > NQueensProblem.py > NQueensBacktracking > __init__
# Analysis and Design of Advanced Algorithms
# Group #607
# Team 3
# Luis Salomón Flores Ugalde
# Santiago Quintana Moreno A01571222
# Miguel Ángel Álvarez Hermida A01722925
# ------ N-QUEENS PROBLEM - BACKTRACKING ALGORITHMS ------
# Total Complexity: O(n!) time, O(n) space - dominated by the depth of the recursion stack
class NQueensBacktracking:
def __init__(self, n):
# Initialize the N-Queens solver
self.n = n
self.board = [[0 for _ in range(n)] for _ in range(n)]
self.solutions = []
def is_safe(self, row, col):
# Check if it's safe to place a queen at position (row, col)
for j in range(col):
if self.board[row][j] == 1:
return False
i, j = row, col
while i >= 0 and j >= 0:
if self.board[i][j] == 1:
return False
i -= 1
j -= 1
i, j = row, col
while i < self.n and j >= 0:
if self.board[i][j] == 1:
return False
i += 1
j -= 1
return True
powershell
PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms> & C:\Users\santy\AppData\Local\Microsoft\WindowsApps\python3.12.exe "d:/1.SQM/1.UNIVERSIDAD/5. QUINTO SEMESTRE/2.Advanced Algorithms/BacktrackingMethod/NQueensProblem.py"
N-Queens Problem Solver using Backtracking
Solving 4-Queens Problem:
Found 2 solution(s)
First solution:
Solving 6-Queens Problem:
Found 4 solution(s)
First solution:
Solving 8-Queens Problem:
Found 92 solution(s)
main*
Go Live


File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
NQueensProblem.py U
BacktrackingMethod > NQueensProblem.py > NQueensBacktracking > __init__
13 class NQueensBacktracking:
42
43 def solve_nqueens_util(self, col):
44 # Recursive utility function to solve N-Queens problem using backtracking
45 if col >= self.n:
46 # Store the current solution
47 solution = []
48 for i in range(self.n):
49 for j in range(self.n):
50 if self.board[i][j] == 1:
51 solution.append((i, j))
52 self.solutions.append(solution)
53 return True
54
55 # Consider this column and try placing queens in all rows one by one
56 res = False
57 for i in range(self.n):
58 if self.is_safe(i, col):
59 self.board[i][col] = 1
60 res = self.solve_nqueens_util(col + 1) or res
61 self.board[i][col] = 0
62
63 return res
64
65 def solve_nqueens(self, find_all=True):
66 # Solve the N-Queens problem
67 self.solutions = []
68
69 if find_all:
70 self.solve_nqueens_util(0)
71 else:
72 if self.solve_nqueens_util(0):
73 self.solutions = [self.solutions[0]] if self.solutions else []
74
75 return self.solutions
76
77 def print_board(self, solution=None):
78 if solution:
79 temp_board = [[0 for _ in range(self.n)] for _ in range(self.n)]
80 for row, col in solution:
81 temp_board[row][col] = 1
82 board_to_print = temp_board
powershell
Solving 8-Queens Problem:
Found 92 solution(s)
First solution:
Interactive N-Queens Solver
Enter the value of N (or 0 to exit): 5
Find all solutions? (y/n): y
Found 10 solution(s) for 5-Queens problem:
Solution 1:
Queen positions: [(0, 0), (1, 3), (2, 1), (3, 4), (4, 2)]
main*
Go Live


File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
NQueensProblem.py U
BacktrackingMethod > NQueensProblem.py > NQueensBacktracking > __init__
class NQueensBacktracking:
def print_board(self, solution=None):
board_to_print = temp_board
else:
board_to_print = self.board
print("+" + "---+" * self.n)
for i in range(self.n):
print("|", end="")
for j in range(self.n):
if board_to_print[i][j] == 1:
print(" Q ", end="|")
else:
print("   ", end="|")
print()
print("+" + "---+" * self.n)
def print_all_solutions(self):
# Print all found solutions
if not self.solutions:
print(f"No solutions found for {self.n}-Queens problem")
return
print(f"Found {len(self.solutions)} solution(s) for {self.n}-Queens problem:")
print()
for i, solution in enumerate(self.solutions):
print(f"Solution {i + 1}:")
print(f"Queen positions: {solution}")
self.print_board(solution)
print()
def demonstrate_nqueens():
# Demonstrate the N-Queens solver with different board sizes
print("N-Queens Problem Solver using Backtracking")
print("=" * 50)
# Test with different values of N
test_cases = [4, 6, 8]
powershell
Solution 2:
Queen positions: [(0, 0), (1, 2), (2, 4), (3, 1), (4, 3)]
Q			
		Q	
				Q
	Q		
			Q
Solution 3:
Queen positions: [(0, 2), (1, 0), (2, 3), (3, 1), (4, 4)]
		Q	
Q			
			Q
	Q		
			Q
Solution 4:
Queen positions: [(0, 3), (1, 0), (2, 2), (3, 4), (4, 1)]
			Q
Q			
		Q	
				Q
	Q		
Solution 5:
Queen positions: [(0, 1), (1, 3), (2, 0), (3, 2), (4, 4)]
	Q		
main*
0 0
Go Live


File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
NQueensProblem.py U
BacktrackingMethod > NQueensProblem.py > NQueensBacktracking > __init__
class NQueensBacktracking:
def print_board(self, solution=None):
board_to_print = temp_board
else:
board_to_print = self.board
print("+" + "---+" * self.n)
for i in range(self.n):
print("|", end="")
for j in range(self.n):
if board_to_print[i][j] == 1:
print(" Q ", end="|")
else:
print("   ", end="|")
print()
print("+" + "---+" * self.n)
def print_all_solutions(self):
# Print all found solutions
if not self.solutions:
print(f"No solutions found for {self.n}-Queens problem")
return
print(f"Found {len(self.solutions)} solution(s) for {self.n}-Queens problem:")
print()
for i, solution in enumerate(self.solutions):
print(f"Solution {i + 1}:")
print(f"Queen positions: {solution}")
self.print_board(solution)
print()
def demonstrate_nqueens():
# Demonstrate the N-Queens solver with different board sizes
print("N-Queens Problem Solver using Backtracking")
print("=" * 50)
# Test with different values of N
test_cases = [4, 6, 8]
powershell
Solution 2:
Queen positions: [(0, 0), (1, 2), (2, 4), (3, 1), (4, 3)]
Q			
		Q	
				Q
	Q		
			Q
Solution 3:
Queen positions: [(0, 2), (1, 0), (2, 3), (3, 1), (4, 4)]
		Q	
Q			
			Q
	Q		
			Q
Solution 4:
Queen positions: [(0, 3), (1, 0), (2, 2), (3, 4), (4, 1)]
			Q
Q			
		Q	
				Q
	Q		
Solution 5:
Queen positions: [(0, 1), (1, 3), (2, 0), (3, 2), (4, 4)]
	Q		
main*
0 0
Go Live


File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
NQueensProblem.py U
BacktrackingMethod > NQueensProblem.py > NQueensBacktracking > __init__
def interactive_nqueens():
find_all = input("Find all solutions? (y/n): ").lower().startswith('y')
solver = NQueensBacktracking(n)
solutions = solver.solve_nqueens(find_all=find_all)
if solutions:
if find_all:
solver.print_all_solutions()
else:
print(f"\nFound a solution for {n}-Queens:")
solver.print_board(solutions[0])
else:
print(f"No solutions found for {n}-Queens problem")
print()
except ValueError:
print("Please enter a valid integer")
except KeyboardInterrupt:
print("\nExiting...")
break
if __name__ == "__main__":
# Run demonstrations
demonstrate_nqueens()
# Run interactive solver
interactive_nqueens()
powershell
Solution 8:
Queen positions: [(0, 4), (1, 1), (2, 3), (3, 0), (4, 2)]
Solution 9:
Queen positions: [(0, 3), (1, 1), (2, 4), (3, 2), (4, 0)]
Solution 10:
Solution 10:
Queen positions: [(0, 2), (1, 4), (2, 1), (3, 3), (4, 0)]
main*
Go Live


File Edit Selection View Go Run Terminal Help
2.Advanced Algorithms
NQueensProblem.py U
BacktrackingMethod > NQueensProblem.py > NQueensBacktracking > __init__
def interactive_nqueens():
find_all = input("Find all solutions? (y/n): ").lower().startswith('y')
solver = NQueensBacktracking(n)
solutions = solver.solve_nqueens(find_all=find_all)
if solutions:
if find_all:
solver.print_all_solutions()
else:
print(f"\nFound a solution for {n}-Queens:")
solver.print_board(solutions[0])
else:
print(f"No solutions found for {n}-Queens problem")
print()
except ValueError:
print("Please enter a valid integer")
except KeyboardInterrupt:
print("\nExiting...")
break
if __name__ == "__main__":
# Run demonstrations
demonstrate_nqueens()
# Run interactive solver
interactive_nqueens()
powershell
Solution 10:
Solution 10:
Queen positions: [(0, 2), (1, 4), (2, 1), (3, 3), (4, 0)]
Enter the value of N (or 0 to exit): 3
No solutions exist for 3-Queens problem
Enter the value of N (or 0 to exit): 0
PS D:\1.SQM\1.UNIVERSIDAD\5. QUINTO SEMESTRE\2.Advanced Algorithms>
main*
Go Live

https://colab.research.google.com/drive/19H-afaJew-yM_ErDwEqdLdnh9wRlXroM?usp=sharing

REFERENCES

N-Queens - LeetCode. (n.d.). LeetCode. https://leetcode.com/problems/n-queens/

El problema de las reinas N. (n.d.). Google for Developers. https://developers.google.com/optimization/cp/queens?hl=es-419

GeeksforGeeks. (2025, July 23). *Backtracking algorithm in Python*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/backtracking-algorithm-in-python/