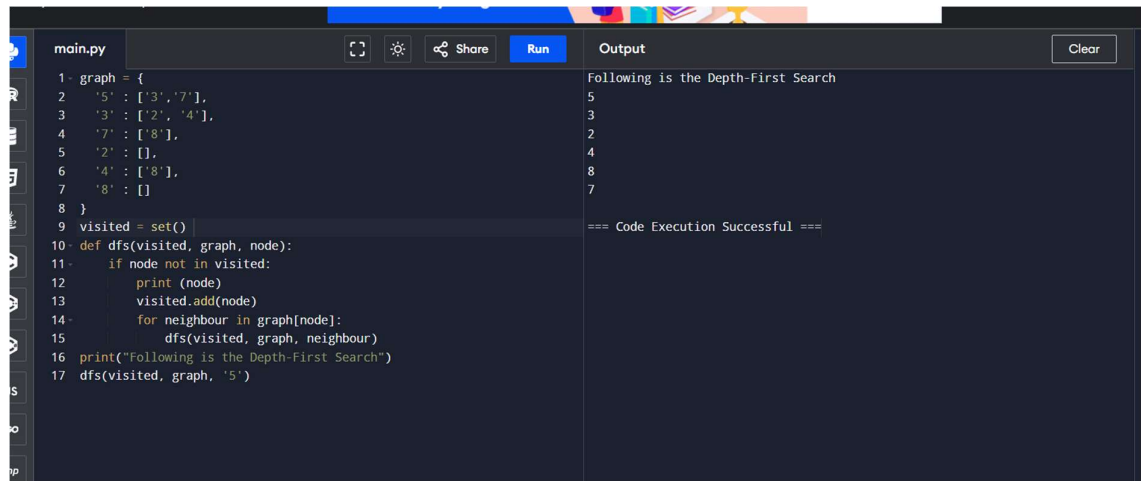


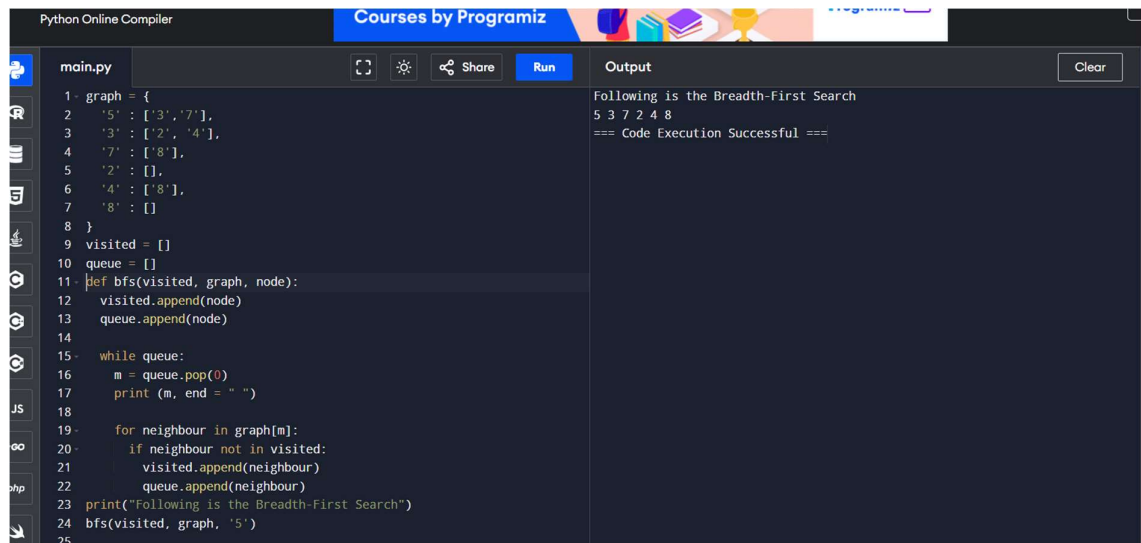
1. DFS
2. BFS
3. WATER JUG
4. CAP
5. 8 PUZZLES
6. 8 QUEENS
7. VACUUM CLEANER
8. MISSIONARIES CANNIBAL



The screenshot shows a Python Online Compiler interface. The code in the editor defines a graph and performs a Depth-First Search (DFS) starting from node '5'. The output displays the nodes visited in the order: 5, 3, 2, 4, 8, 7.

```
main.py 1 graph = {
2     '5': ['3', '7'],
3     '3': ['2', '4'],
4     '7': ['8'],
5     '2': [],
6     '4': ['8'],
7     '8': []
8 }
9 visited = set()
10 def dfs(visited, graph, node):
11     if node not in visited:
12         print (node)
13         visited.add(node)
14         for neighbour in graph[node]:
15             dfs(visited, graph, neighbour)
16 print("Following is the Depth-First Search")
17 dfs(visited, graph, '5')
```

Output: Following is the Depth-First Search
5
3
2
4
8
7
=== Code Execution Successful ===



The screenshot shows a Python Online Compiler interface. The code in the editor defines a graph and performs a Breadth-First Search (BFS) starting from node '5'. The output displays the nodes visited in the order: 5, 3, 7, 2, 4, 8.

```
main.py 1 graph = {
2     '5': ['3', '7'],
3     '3': ['2', '4'],
4     '7': ['8'],
5     '2': [],
6     '4': ['8'],
7     '8': []
8 }
9 visited = []
10 queue = []
11 def bfs(visited, graph, node):
12     visited.append(node)
13     queue.append(node)
14
15 while queue:
16     m = queue.pop(0)
17     print (m, end = " ")
18
19     for neighbour in graph[m]:
20         if neighbour not in visited:
21             visited.append(neighbour)
22             queue.append(neighbour)
23 print("Following is the Breadth-First Search")
24 bfs(visited, graph, '5')
```

Output: Following is the Breadth-First Search
5 3 7 2 4 8
=== Code Execution Successful ===

main.py

Share

Run

Output

Clear

```
1 from collections import deque
2 def water_jug_problem(jug1_cap, jug2_cap, target):
3     visited = set()
4     queue = deque([(0, 0, [])])
5     while queue:
6         j1, j2, steps = queue.popleft()
7         if (j1, j2) in visited:
8             continue
9         visited.add((j1, j2))
10        if j1 == target:
11            return steps
12        actions = [
13            (jug1_cap, j2), (j1, jug2_cap),
14            (0, j2), (j1, 0),
15            (j1 - min(j1, jug2_cap - j2), j2 + min(j1, jug2_cap - j2)),
16            (j1 + min(j2, jug1_cap - j1), j2 - min(j2, jug1_cap - j1))
17        ]
18        for next_j1, next_j2 in actions:
19            if (next_j1, next_j2) not in visited:
20                queue.append((next_j1, next_j2, steps + [(j1, j2)]))
21    return None
22 result = water_jug_problem(4, 3, 2)
23 print(result)
24
```

[(0, 0), (4, 0), (1, 3), (1, 0), (0, 1), (4, 1)]
=== Code Execution Successful ===

Programiz

Python Online Compiler

Premium Coding Courses by Programiz

Programiz PRO

Pro

main.py

Share

Run

Output

Clear

```
1 from itertools import permutations
2 def cryptarithmic_solver():
3     letters = 'SENDMOREMONEY'
4     unique_letters = set(letters)
5     for perm in permutations(range(10), len(unique_letters)):
6         letter_to_digit = dict(zip(unique_letters, perm))
7         if letter_to_digit['S'] == 0 or letter_to_digit['M'] == 0:
8             continue
9         send = sum(letter_to_digit[char] * 10**(3 - i) for i, char in enumerate('SEND'))
10        more = sum(letter_to_digit[char] * 10**(3 - i) for i, char in enumerate('MORE'))
11        money = sum(letter_to_digit[char] * 10**(4 - i) for i, char in enumerate('MONEY'))
12        if send + more == money:
13            return letter_to_digit
14    return None
15 solution = cryptarithmic_solver()
16 if solution:
17     print("Solution found:", solution)
18 else:
19     print("No solution found.")
20
```

Solution found: {'M': 1, 'S': 9, 'E': 5, 'Y': 2, 'N': 6, 'D': 7, 'R': 8, 'O': 0}
=== Code Execution Successful ===

Premium Coding Courses by Programiz

Python Online Compiler

Run

Share

Settings

Fullscreen

main.py

```

1 import copy, heapq
2 n = 3
3 row, col = [1, 0, -1, 0], [0, -1, 0, 1]
4 def solve(initial, empty, final):
5     pq = [(sum(initial[i][j] != final[i][j] for i in range(n) for j in
6         range(n)), 0, initial, empty, [])]
7     while pq:
8         cost, level, mat, empty, path = heapq.heappop(pq)
9         if cost == 0:
10             for step in path + [mat]: print(*step, sep="\n")
11             return
12             for i in range(4):
13                 x, y = empty[0] + row[i], empty[1] + col[i]
14                 if 0 <= x < n and 0 <= y < n:
15                     new_mat = copy.deepcopy(mat)
16                     new_mat[empty[0]][empty[1]], new_mat[x][y] =
17                         new_mat[x][y], new_mat[empty[0]][empty[1]]
18                     heapq.heappush(pq, (sum(new_mat[i][j] != final[i][j]
19                         for i in range(n) for j in range(n)), level + 1,
20                         new_mat, (x, y), path + [mat]))
21
22 initial = [[1, 2, 3], [5, 6, 0], [7, 8, 4]]
23 final = [[1, 2, 3], [5, 8, 6], [0, 7, 4]]
24 solve(initial, (1, 2), final)
25

```

Output

Clear

```

[1, 2, 3]
[5, 6, 0]
[7, 8, 4]
[1, 2, 3]
[5, 0, 6]
[7, 8, 4]
[1, 2, 3]
[5, 8, 6]
[7, 0, 4]
[1, 2, 3]
[5, 8, 6]
[0, 7, 4]

=== Code Execution Successful ===

```

BLACK FRIDAY SALE

Get Programiz PRO for LIFE at 60% off! Claim My Discount

Sale ends in 00d : 23hrs : 00mins : 27s

Programiz

Python Online Compiler

Premium Coding Courses by Programiz

main.py

```

1 N = int(input())
2 board = [[0]*N for _ in range(N)]
3 def attack(i, j):
4     for k in range(0,N):
5         if board[i][k]==1 or board[k][j]==1:
6             return True
7     for k in range(0,N):
8         for l in range(0,N):
9             if (k+l==i+j) or (k-l==i-j):
10                if board[k][l]==1:
11                    return True
12     return False
13 def N_queens(n):
14     if n==0:
15         return True
16     for i in range(0,N):
17         for j in range(0,N):
18             if (not(attack(i,j))) and (board[i][j]!=1):
19                 board[i][j] = 1
20                 if N_queens(n-1)==True:
21                     return True
22                 board[i][j] = 0
23     return False
24 N_queens(N)
25 for i in board:

```

Output

Clear

```

Enter the number of queens
8
[1, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 1, 0]
[0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0]

=== Code Execution Successful ===

```

Premium Coding Courses by Programiz

Programiz PRO

Python Online Compiler

main.py

Share

Run

```

1 def vacuum_cleaner(world):
2     actions = []
3     for row in range(2):
4         for col in range(2):
5             if world[row][col] == "dirty":
6                 actions.append(f"Clean ({row}, {col})")
7                 world[row][col] = "clean"
8                 actions.append(f"Move to ({row}, {col})")
9     return actions
10 world = [{"dirty", "clean"}, {"dirty", "dirty"}]
11 print("Actions:", vacuum_cleaner(world))
12 print("Final World State:", world)
13

```

Output

Clear

```

Actions: ['Clean (0, 0)', 'Move to (0, 0)', 'Move to (0, 1)', 'Clean (1, 0)',
'Move to (1, 0)', 'Clean (1, 1)', 'Move to (1, 1)']
Final World State: [['clean', 'clean'], ['clean', 'clean']]

=== Code Execution Successful ===

```

Premium Coding Courses by Programiz

Programiz PRO

Python Online Compiler

main.py

Share

Run

```

1 from collections import deque
2 def is_valid(state):
3     m1, c1, b, m2, c2 = state
4     return (m1 == 0 or m1 >= c1) and (m2 == 0 or m2 >= c2)
5 def get_neighbors(state):
6     m1, c1, b, m2, c2 = state
7     moves = [(1, 0), (2, 0), (0, 1), (0, 2), (1, 1)]
8     new_b = 1 - b
9     return [(m1 - b*m, c1 - b*c, new_b, m2 + b*m, c2 + b*c)
10             for m, c in moves if is_valid((m1 - b*m, c1 - b*c, new_b,
11                                           m2 + b*m, c2 + b*c))]
12 def missionaries_cannibals():
13     start, goal = (3, 3, 1, 0, 0), (0, 0, 0, 3, 3)
14     queue, visited = deque([(start, [])]), set([start])
15     while queue:
16         (state, path) = queue.popleft()
17         if state == goal: return path + [state]
18         for neighbor in get_neighbors(state):
19             if neighbor not in visited:
20                 visited.add(neighbor)
21                 queue.append((neighbor, path + [state]))
22 solution = missionaries_cannibals()
23 print("Solution path:", solution)

```

Output

Clear

```

Solution path: [(3, 3, 1, 0, 0), (3, 1, 0, 0, 2), (3, 1, 1, 0, 2), (1, 1, 0,
2, 2), (1, 1, 1, 2, 2), (0, 0, 0, 3, 3)]

=== Code Execution Successful ===

```