Solving the 5-Queens Problem Using Breadth-First Search (BFS)

Problem Description

The 5-Queens problem is a classical artificial intelligence problem. The objective is to place 5 queens on a 5×5 chessboard such that no two queens attack each other.

This means:
- No two queens are in the same row
- No two queens are in the same column
- No two queens are on the same diagonal

State Representation

Each state is represented as a list of integers:
- The index represents the column
- The value represents the row of the queen in that column

Example:
[0, 2, 4]

This means:
Queen at column 0, row 0
Queen at column 1, row 2
Queen at column 2, row 4

Only one queen is placed per column, which automatically avoids column conflicts.

Initial State

The initial state is an empty list:
[]

No queens are placed yet.

Goal State

A state is considered a goal state when:

- The length of the state is 5
- There are no conflicts between any two queens

Successor Function

From a given state, successor states are generated by:

- Placing a queen in the next column
- Trying all possible rows from 0 to 4
- Adding the queen only if the position is safe (no row or diagonal conflict)

Breadth-First Search (BFS)

Breadth-First Search explores the state space level by level.
It expands all states with the same depth before moving to deeper levels.

BFS guarantees finding the shallowest solution, but it requires a large amount of memory.

Python Code

```
from collections import deque

# Board size
N = 5

# Function to check if a queen can be safely placed in the given
row
def is_safe(state, row):
    col = len(state)  # Next column to place
    for c, r in enumerate(state):
        # Check row and diagonal conflicts
        if r == row or abs(r - row) == abs(c - col):
            return False
    return True

# Breadth-First Search function
def breadth_first_search():
    # Queue to store states
    queue = deque()
    queue.append([])  # Initial state (no queens)

    while queue:
        state = queue.popleft()

        # Goal test: 5 queens placed safely
        if len(state) == N:
            return state

        # Generate successors
        for row in range(N):
            if is_safe(state, row):
                new_state = state + [row]
                queue.append(new_state)

    return None

# Run BFS
solution = breadth_first_search()
print("Solution:", solution)
```

Example Output

Solution: [0, 2, 4, 1, 3]

Each number represents the row position of the queen in the corresponding column.

Conclusion

Breadth-First Search successfully solves the 5-Queens problem by exploring the state space level by level.
Although BFS guarantees finding a valid solution, it is memory-intensive compared to DFS and A* search.