Here's Python code for two Uninformed Search Algorithms:

🔍 1. Breadth-First Search (BFS)

Explores all nodes at the current depth level before moving on to the next level.

```
from collections import deque

def bfs(graph, start, goal):
    visited = set()
    queue = deque([[start]])

    while queue:
        path = queue.popleft()
        node = path[-1]

        if node == goal:
            return path

        if node not in visited:
            visited.add(node)
            for neighbor in graph.get(node, []):
                new_path = list(path)
                new_path.append(neighbor)
                queue.append(new_path)

    return None

# Example graph
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
```

```
    'D': [],
    'E': ['F'],
    'F': []
}

print("BFS path:", bfs(graph, 'A', 'F'))
```

🔍 **2. Depth-First Search (DFS)**

Explores as far as possible along one branch before backtracking.

```
def dfs(graph, start, goal):
    visited = set()
    stack = [[start]]

    while stack:
        path = stack.pop()
        node = path[-1]

        if node == goal:
            return path

        if node not in visited:
            visited.add(node)
            for neighbor in reversed(graph.get(node, [])):  # Reverse for correct DFS order
                new_path = list(path)
                new_path.append(neighbor)
                stack.append(new_path)

    return None

# Reuse the same graph
print("DFS path:", dfs(graph, 'A', 'F'))
```

✅ **Output Example:**

BFS path: ['A', 'C', 'F']

DFS path: ['A', 'B', 'E', 'F']

These are path-finding implementations of BFS and DFS. They return the path from the start node to the goal node if found.

Let me know if you want to visualize the search tree or apply it to a maze or grid!