Exercise - Stack Problems

Section 1 - Stack Implementation Problems

1. 다음 코드를 완성하여 괄호가 올바르게 매칭되는지 검사하는 함수를 구현하세요.

```
class Stack:
    # ... 생략

def is_valid_brackets(s):
    stack = Stack()
    pairs = {'(': ')', '[': ']', '{': '}'}

    for char in s:
        if char in pairs:
            stack.push(char)
        elif char in pairs.values():
            # 여기를 완성하세요 (닫는 괄호 처리)

    return stack.is_empty()
```

입력예시

```
is_valid_brackets("()")
is_valid_brackets("([{}])")
is_valid_brackets("([)]")
is_valid_brackets("(((")
```

출력예시

```
True
True
False
False
```

> **힌트:** *여는 괄호는 스택에 push하고, 닫는 괄호는 스택에서 pop해서 매칭을 확인하세요.*

해설

```
def is_valid_brackets(s):
    stack = Stack()
    pairs = {'(': ')', '[': ']', '{': '}'}

    for char in s:
        if char in pairs:  # 여는 괄호
            stack.push(char)
        elif char in pairs.values():  # 닫는 괄호
            if stack.is_empty():
                return False
            if pairs[stack.pop()] != char:
                return False

    return stack.is_empty()
```

2. 다음 코드를 완성하여 유효한 경로 확인 및 스택 추가 부분을 구현하세요.

```
class Stack:
    # ... 생략

def solve_maze(maze):
    rows, cols = len(maze), len(maze[0])
    visited = [[False] * cols for _ in range(rows)]
    stack = Stack()

    start_path = [(0, 0)]
    stack.push(start_path)

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    while not stack.is_empty():
        path = stack.pop()
        row, col = path[-1]

        if visited[row][col]:
            continue

        visited[row][col] = True

        if row == rows - 1 and col == cols - 1:
            return path

        for dr, dc in directions:
            new_row, new_col = row + dr, col + dc

            # 여기를 완성하세요 (유효한 경로인지 확인하고 스택에 추가)

    return []
```

입력예시

```
maze = [[0, 1, 0, 0, 0], [0, 1, 0, 1, 0], [0, 0, 0, 1, 0], [1, 1, 0, 0, 0], [0, 0, 0, 1, 0]]
solve_maze(maze)
```

출력예시

```
경로 발견
```

> **힌트:** *경계 내에 있고, 벽이 아니며, 방문하지 않은 경로만 스택에 추가하세요.*

해설

```
def solve_maze(maze):
    rows, cols = len(maze), len(maze[0])
    visited = [[False] * cols for _ in range(rows)]
    stack = Stack()

    start_path = [(0, 0)]
    stack.push(start_path)

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    while not stack.is_empty():
        path = stack.pop()
        row, col = path[-1]

        if visited[row][col]:
            continue

        visited[row][col] = True

        if row == rows - 1 and col == cols - 1:
            return path

        for dr, dc in directions:
            new_row, new_col = row + dr, col + dc

            # 유효한 경로인지 확인하고 스택에 추가
            if (0 <= new_row < rows and 0 <= new_col < cols and
                maze[new_row][new_col] == 0 and not visited[new_row][new_col]):
                new_path = path + [(new_row, new_col)]
                stack.push(new_path)

    return []
```

3. 다음 코드를 완성하여 DFS 시작 전 스택을 초기화하는 부분을 구현하세요.

```
class Stack:
    # ... 생략

def dfs_iterative(graph, start):
    visited = set()
    path = []
    stack = Stack()

    # 여기를 완성하세요 (시작 노드 초기화)

    while not stack.is_empty():
        current = stack.pop()

        if current in visited:
            continue

        visited.add(current)
        path.append(current)

        for neighbor in reversed(graph[current]):
            if neighbor not in visited:
                stack.push(neighbor)

    return path
```

입력예시

```
graph = {0: [1, 3], 1: [0, 2, 4], 2: [1, 5], 3: [0, 4], 4: [1, 3, 5], 5: [2, 4]}
dfs_iterative(graph, 0)
```

출력예시

```
[0, 1, 2, 5, 4, 3]
```

> **힌트:** *DFS를 시작하기 위해 시작 노드를 스택에 push하세요.*

해설

```
def dfs_iterative(graph, start):
    visited = set()
    path = []
    stack = Stack()

    # 시작 노드를 스택에 추가
    stack.push(start)

    while not stack.is_empty():
        current = stack.pop()

        if current in visited:
            continue

        visited.add(current)
        path.append(current)

        for neighbor in reversed(graph[current]):
            if neighbor not in visited:
                stack.push(neighbor)

    return path
```

4. 다음 코드를 완성하여 스택을 이용한 DFS 탐색을 구현하세요.

```
class Stack:
    # ... 생략

def dfs_iterative(graph, start):
    visited = set()
    path = []
    stack = Stack()

    stack.push(start)

    while not stack.is_empty():
        current = stack.pop()

        if current in visited:
            continue

        visited.add(current)
        path.append(current)

        # 여기를 완성하세요 (이웃 노드들을 스택에 추가)

    return path
```

입력예시

```
graph = {0: [1, 3], 1: [0, 2, 4], 2: [1, 5], 3: [0, 4], 4: [1, 3, 5], 5: [2, 4]}
dfs_iterative(graph, 0)
```

출력예시

```
[0, 1, 2, 5, 4, 3]
```

> **힌트:** *이웃 노드들을 역순으로 스택에 추가하여 올바른 탐색 순서를 보장하세요.*

해설

```
def dfs_iterative(graph, start):
    visited = set()
    path = []
    stack = Stack()

    stack.push(start)

    while not stack.is_empty():
        current = stack.pop()

        if current in visited:
            continue

        visited.add(current)
        path.append(current)

        # 이웃 노드들을 스택에 추가 (역순으로)
        for neighbor in reversed(graph[current]):
            if neighbor not in visited:
                stack.push(neighbor)

    return path
```

5. 다음 코드를 완성하여 목표 지점 도달 확인 부분을 구현하세요.

```
class Stack:
    # ... 생략

def solve_maze(maze):
    rows, cols = len(maze), len(maze[0])
    visited = [[False] * cols for _ in range(rows)]
    stack = Stack()

    start_path = [(0, 0)]
    stack.push(start_path)

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    while not stack.is_empty():
        path = stack.pop()
        row, col = path[-1]

        if visited[row][col]:
            continue

        visited[row][col] = True

        # 여기를 완성하세요 (목표 지점 확인)

        for dr, dc in directions:
            new_row, new_col = row + dr, col + dc

            if (0 <= new_row < rows and 0 <= new_col < cols and
                maze[new_row][new_col] == 0 and not visited[new_row][new_col]):
                new_path = path + [(new_row, new_col)]
                stack.push(new_path)

    return []
```

입력예시

```
maze = [[0, 1, 0, 0, 0], [0, 1, 0, 1, 0], [0, 0, 0, 1, 0], [1, 1, 0, 0, 0], [0, 0, 0, 1, 0]]
solve_maze(maze)
```

출력예시

```
경로 발견
```

> **힌트:** *우측 하단 모서리에 도달했는지 확인하고 경로를 반환하세요.*

해설

```
def solve_maze(maze):
    rows, cols = len(maze), len(maze[0])
    visited = [[False] * cols for _ in range(rows)]
    stack = Stack()

    start_path = [(0, 0)]
    stack.push(start_path)

    directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

    while not stack.is_empty():
        path = stack.pop()
        row, col = path[-1]

        if visited[row][col]:
            continue

        visited[row][col] = True

        # 목표 지점 도달 확인
        if row == rows - 1 and col == cols - 1:
            return path

        for dr, dc in directions:
            new_row, new_col = row + dr, col + dc

            if (0 <= new_row < rows and 0 <= new_col < cols and
                maze[new_row][new_col] == 0 and not visited[new_row][new_col]):
                new_path = path + [(new_row, new_col)]
                stack.push(new_path)

    return []
```