Home Work

October 10, 2024

1 Home Work

1.1 logAnalyzer.py

[5]:
```
def analyze_log_file(log_file_path: str) -> dict:
    result = {
        'requests_by_ip': {},
        'most_requested_resource': None,
        'total_bytes': 0,
        'total_404_errors': 0,
    }
    with open(log_file_path, 'r') as log_file:
        lines = [line.strip().split() for line in log_file.readlines()]
    if len(lines) != 0:
        list_of_requests = []
        for line in lines:
            if len(line) != 5:
                continue
            if line[0] not in result['requests_by_ip']:
                result['requests_by_ip'][line[0]] = 1
            else:
                result['requests_by_ip'][line[0]] += 1
            list_of_requests.append(line[2])
            if line[3] == '404':
                result['total_404_errors'] += 1
            result['total_bytes'] += int(line[4])
        for resource in list_of_requests:
            try:
                if list_of_requests.count(result["most_requested_resource"]) <␣
↪list_of_requests.count(resource):
                    result["most_requested_resource"] = resource
            except KeyError:
                result["most_requested_resource"] = resource
        return result
    else:
        return result
```

```
log_file_path = '..\\HomeWork\\HW3\\server_log.txt'  # Replace with the path to␣
 ↪your log file
result = analyze_log_file(log_file_path)

# Print the analysis result
print("Log File Analysis Result:")
print(f"Requests by IP: {result['requests_by_ip']}")
print(f"Most Requested Resource: {result['most_requested_resource']}")
print(f"Total 404 Errors: {result['total_404_errors']}")
print(f"Total Bytes Transferred: {result['total_bytes']}")
```

```
Log File Analysis Result:
Requests by IP: {'192.168.1.1': 3, '192.168.1.2': 2}
Most Requested Resource: /index.html
Total 404 Errors: 2
Total Bytes Transferred: 2432
```

1.2 anagrams.py

[1]:
```
def group_anagrams(words: list) -> list:
    group_list = []
    for word in words:
        if set(list(word)) not in group_list:
            group_list.append(set(list(word)))

    result = []
    for group in group_list:
        temp = []
        for word in words:
            if set(list(word)) == group:
                temp.append(word)
        result.append(temp)
    return result

if __name__ == '__main__':
    words = ["eat", "tea", "tan", "ate", "nat", "bat"]
    print(group_anagrams(words))

    words = ["listen", "silent", "enlist", "hello", "world"]
    print(group_anagrams(words))
```

```
[['eat', 'tea', 'ate'], ['tan', 'nat'], ['bat']]
[['listen', 'silent', 'enlist'], ['hello'], ['world']]
```

1.3 maze.py

[6]:
```
from collections import deque

def maze_solver_with_teleport(maze: list, portals: dict):
    rows, cols = len(maze), len(maze[0])
    start = None
    end = None
    for r in range(rows):
        for c in range(cols):
            if maze[r][c] == 'S':
                start = (r, c)
            elif maze[r][c] == 'E':
                end = (r, c)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    queue = deque([(start, [start], 0)])
    visited = set()
    visited.add(start)
    while queue:
        (current, path, steps) = queue.popleft()
        if current == end:
            return steps, path
        for d in directions:
            next_r, next_c = current[0] + d[0], current[1] + d[1]

            if 0 <= next_r < rows and 0 <= next_c < cols:
                if maze[next_r][next_c] != '#' and (next_r, next_c) not in␣
 ↪visited:

                    visited.add((next_r, next_c))
                    next_steps = steps + 1
                    queue.append(((next_r, next_c), path + [(next_r, next_c)],␣
 ↪next_steps))

                    if maze[next_r][next_c] == 'P':
                        portal_destination = portals[(next_r, next_c)]
                        if portal_destination not in visited:
                            visited.add(portal_destination)
                            queue.append((portal_destination, path + [(next_r,␣
 ↪next_c), portal_destination], next_steps))

    return -1, []
    #
if __name__ == "__main__":
    # Example 1
    maze = [
        ['S', '.', '.', 'P'],
```

```
        ['#', '#', '.', '#'],
        ['P', '.', '.', 'E'],
        ['#', '#', '#', '#']
    ]

    portals = {
        (0, 3): (2, 0),  # Portal from (0, 3) to (2, 0)
        (2, 0): (0, 3)   # Portal from (2, 0) to (0, 3)
    }

    distance, path = maze_solver_with_teleport(maze, portals)
    print(f"Distance: {distance}, Path: {path}")
    # Output: Distance: 5, Path: [(0, 0), (1, 0), (1, 1), (1, 2), (2, 2), (2, 3)]


    # Example 2
    maze = [
        ['S', '.', '#', 'P', '#', 'P'],
        ['#', '.', '#', '.', '#', '.'],
        ['#', '.', 'P', '.', '.', 'E'],
        ['P', '#', '#', '#', '#', '#'],
        ['#', '.', '.', 'P', '.', '.']
    ]

    portals = {
        (0, 3): (3, 0),  # Portal from (0, 3) to (3, 0)
        (3, 0): (0, 3),  # Portal from (3, 0) to (0, 3)
        (0, 5): (2, 2),  # Portal from (0, 5) to (2, 2)
        (2, 2): (0, 5)   # Portal from (2, 2) to (0, 5)
    }

    distance, path = maze_solver_with_teleport(maze, portals)
    print(f"Distance: {distance}, Path: {path}")
    # Expected Output: Distance: 6, Path: [(0, 0), (0, 1), (1, 1), (2, 1), (2,␣
↪2), (0, 5), (1, 5), (2, 5)]
```

```
Distance: 5, Path: [(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3)]
Distance: 6, Path: [(0, 0), (0, 1), (1, 1), (2, 1), (2, 2), (0, 5), (1, 5), (2,
5)]
```