LAB-06 Parallel Celluar Algorithm and programs

```
import numpy as np
import matplotlib.pyplot as plt

def parallel_cellular_shortest_path(grid, start, end,
max_iterations=100):
    """
    Finds the shortest path in a grid using Parallel
Cellular Algorithm.
    Visualizes the grid and the shortest path.

    Parameters:
    - grid: 2D numpy array where 0 represents open
cell and 1 represents blocked cell.
    - start: Tuple (x, y) starting coordinates.
    - end: Tuple (x, y) ending coordinates.
    - max_iterations: Maximum number of iterations to
run the algorithm.

    Returns:
    - distance: 2D array with distance from start.
    - path: List of tuples representing the shortest
path from start to end.
    """
    rows, cols = grid.shape
    distance = np.full((rows, cols), np.inf)
    distance[start] = 0

    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]
# 4-directional
```

```
    for iteration in range(max_iterations):
        updated = False
        new_distance = distance.copy()
        for i in range(rows):
            for j in range(cols):
                if grid[i][j] == 1:
                    continue  # Skip blocked cells
                for d in directions:
                    ni, nj = i + d[0], j + d[1]
                    if 0 <= ni < rows and 0 <= nj <
cols:
                        if distance[ni][nj] + 1 <
new_distance[i][j]:
                            new_distance[i][j] =
distance[ni][nj] + 1
                            updated = True
        distance = new_distance
        if not updated:
            break  # No updates in this iteration

    path = []
    if distance[end] == np.inf:
        print("No path found.")
        visualize_grid(grid, start, end, path,
distance)
        return distance, path

    current = end
    path.append(current)
    while current != start:
```

```
        i, j = current
        for d in directions:
            ni, nj = i + d[0], j + d[1]
            if 0 <= ni < rows and 0 <= nj < cols:
                if distance[ni][nj] == distance[i][j]
- 1:
                    path.append((ni, nj))
                    current = (ni, nj)
                    break
    path.reverse()
    visualize_grid(grid, start, end, path, distance)
    return distance, path


def visualize_grid(grid, start, end, path, distance):
    """
    Visualizes the grid with the shortest path,
start, and end points.

    Parameters:
    - grid: 2D numpy array representing the grid.
    - start: Tuple (x, y) starting coordinates.
    - end: Tuple (x, y) ending coordinates.
    - path: List of tuples representing the shortest
path.
    - distance: 2D numpy array of distances from the
start cell.
    """
    rows, cols = grid.shape
    fig, ax = plt.subplots(figsize=(8, 8))
```

```
    # Plot the grid
    for i in range(rows):
        for j in range(cols):
            if grid[i, j] == 1:
                ax.add_patch(plt.Rectangle((j, rows -
i - 1), 1, 1, color="black"))  # Blocked cells
            else:
                ax.add_patch(plt.Rectangle((j, rows -
i - 1), 1, 1, edgecolor="gray", facecolor="white"))

    # Highlight the start and end points
    ax.add_patch(plt.Rectangle((start[1], rows -
start[0] - 1), 1, 1, color="green", label="Start"))
    ax.add_patch(plt.Rectangle((end[1], rows - end[0]
- 1), 1, 1, color="red", label="End"))

    # Plot the shortest path
    if path:
        for (x, y) in path:
            ax.add_patch(plt.Rectangle((y, rows - x -
1), 1, 1, color="blue", alpha=0.5))

    # Annotate distance values
    for i in range(rows):
        for j in range(cols):
            if np.isfinite(distance[i, j]):
                ax.text(j + 0.5, rows - i - 1 + 0.5,
f"{int(distance[i, j])}",
                        color="black", ha="center",
va="center", fontsize=8)
```

```
    ax.set_xlim(0, cols)
    ax.set_ylim(0, rows)
    ax.set_xticks(range(cols))
    ax.set_yticks(range(rows))
    ax.set_xticklabels([])
    ax.set_yticklabels([])
    ax.grid(True)
    ax.legend(loc="upper left")
    plt.gca().invert_yaxis()
    plt.show()


# Example Usage
if __name__ == "__main__":
    # 0: open cell, 1: blocked cell
    grid = np.array([
        [0, 0, 0, 0, 1],
        [1, 1, 0, 1, 0],
        [0, 0, 0, 0, 0],
        [0, 1, 1, 1, 0],
        [0, 0, 0, 0, 0]
    ])

    start = (0, 0)
    end = (4, 4)

    distance, path =
parallel_cellular_shortest_path(grid, start, end)
    print("Distance Grid:")
    print(distance)
    print("Shortest Path:")
```

```python
print(path)
```

OUTPUT

```
Distance Grid:
[[ 0.  1.  2.  3. inf]
 [inf inf  3. inf  7.]
 [ 6.  5.  4.  5.  6.]
 [ 7. inf inf inf  7.]
 [ 8.  9. 10.  9.  8.]]
Shortest Path:
[(0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (2, 4), (3, 4), (4, 4)]
```