基本解法（朴素DFS）

Java代码

```
class Solution {
    public int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
    public int rows, columns;

    public int longestIncreasingPath(int[][] matrix) {
        if (matrix.length == 0 || matrix[0].length == 0)
            return 0;
        rows = matrix.length;
        columns = matrix[0].length;
        int ans = 0;
        for (int i = 0; i < rows; ++i)
            for (int j = 0; j < columns; ++j)
                ans = Math.max(ans, dfs(matrix, i, j));
        return ans;
    }

    public int dfs(int[][] matrix, int row, int column) {
        int result = 1;
        for (int[] dir : dirs) {
            int newRow = row + dir[0], newColumn = column + dir[1];
            //如果邻居元素存在，且比当前元素大
            if (newRow >= 0 && newRow < rows && newColumn >= 0 && newColumn <
columns && matrix[newRow][newColumn] > matrix[row][column]) {
                result = Math.max(result, dfs(matrix, newRow, newColumn) + 1);
            }
        }
        return result;
    }
}
```

优化解法

Java代码（记忆化DFS）

```
class Solution {
    public int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
    public int rows, columns;
    public int[][] longestPath; //数组longestPath记录从各个节点出发的最长路径的长度
    public int longestIncreasingPath(int[][] matrix) {
        if (matrix.length == 0 || matrix[0].length == 0) {
            return 0;
        }
        rows = matrix.length;
        columns = matrix[0].length;
        longestPath = new int[rows][columns];
        int ans = 0;
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < columns; ++j) {
```

```
                ans = Math.max(ans, dfs(matrix, i, j));
            }
        }
        return ans;
    }

    public int dfs(int[][] matrix, int row, int column) {
        if (longestPath[row][column] != 0) return longestPath[row][column]; //该
节点已经被扩展过
        ++longestPath[row][column];
        for (int[] dir : dirs) {
            int newRow = row + dir[0], newColumn = column + dir[1];
            if (newRow >= 0 && newRow < rows && newColumn >= 0 && newColumn <
columns && matrix[newRow][newColumn] > matrix[row][column]) {
                longestPath[row][column] = Math.max(longestPath[row][column],
dfs(matrix, newRow, newColumn) + 1);
            }
        }
        return longestPath[row][column];
    }
}
```

Java代码（拓扑排序）

```
class Solution {
    public int[][] dirs = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
    public int rows, columns;
    public int longestIncreasingPath(int[][] matrix) {
        if (matrix.length == 0 || matrix[0].length == 0) return 0;
        rows = matrix.length;
        columns = matrix[0].length;
        int[][] indegrees = new int[rows][columns]; //统计每个节点的入度
        for (int i = 0; i < rows; ++i)
            for (int j = 0; j < columns; ++j)
                for (int[] dir : dirs) {
                    int newRow = i + dir[0], newColumn = j + dir[1];
                    if (newRow >= 0 && newRow < rows && newColumn >= 0 &&
newColumn < columns && matrix[newRow][newColumn] < matrix[i][j])  ++indegrees[i]
[j];
                }
        Queue<int[]> queue = new LinkedList<int[]>();
        for (int i = 0; i < rows; ++i)
            for (int j = 0; j < columns; ++j)
                //将入度为0的节点加入队列
                if (indegrees[i][j] == 0) queue.offer(new int[]{i, j});
        int ans = 0;
        while (!queue.isEmpty()) {
            ++ans;
            int size = queue.size();
            for (int i = 0; i < size; ++i) {
                int[] cell = queue.poll();
                int row = cell[0], column = cell[1];
                for (int[] dir : dirs) {
                    int newRow = row + dir[0], newColumn = column + dir[1];
                    if (newRow >= 0 && newRow < rows && newColumn >= 0 &&
newColumn < columns && matrix[newRow][newColumn] > matrix[row][column]) {
                        --indegrees[newRow][newColumn];
```

```
                        if (indegrees[newRow][newColumn] == 0)
                            queue.offer(new int[]{newRow, newColumn});
                    }
                }
            }
        }
        return ans;
    }
}
```

C++代码（记忆化DFS）

```
class Solution {
public:
    static constexpr int dirs[4][2] = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};
    int rows, columns;
    vector<vector<int>> longestPath;
    int longestIncreasingPath(vector< vector<int> > &matrix) {
        if (matrix.size() == 0 || matrix[0].size() == 0) {
            return 0;
        }
        rows = matrix.size();
        columns = matrix[0].size();
        longestPath = vector<vector<int>> (rows, vector<int>(columns));
        int ans = 0;
        for (int i = 0; i < rows; ++i) {
            for (int j = 0; j < columns; ++j) {
                ans = max(ans, dfs(matrix, i, j));
            }
        }
        return ans;
    }

    int dfs(vector< vector<int> > &matrix, int row, int column) {
        if (longestPath[row][column] != 0)
            return longestPath[row][column];
        longestPath[row][column] = 1;
        for (int i = 0; i < 4; ++i) {
            int newRow = row + dirs[i][0], newColumn = column + dirs[i][1];
            if (newRow >= 0 && newRow < rows && newColumn >= 0 && newColumn <
columns && matrix[newRow][newColumn] > matrix[row][column]) {
                longestPath[row][column] = max(longestPath[row][column],
dfs(matrix, newRow, newColumn) + 1);
            }
        }
        return longestPath[row][column];
    }
};
```

Python代码

```
class Solution:
    DIRS = [(-1, 0), (1, 0), (0, -1), (0, 1)]
    def longestIncreasingPath(self, matrix: List[List[int]]) -> int:
        if not matrix:
```

```
            return 0
        #缓存函数的返回值，如果重复调用，直接返回缓存中的值
        @lru_cache(None)
        def dfs(row: int, column: int) -> int:
            best = 1
            for dx, dy in Solution.DIRS:
                newRow, newColumn = row + dx, column + dy
                if 0 <= newRow < rows and 0 <= newColumn < columns and
matrix[newRow][newColumn] > matrix[row][column]:
                    best = max(best, dfs(newRow, newColumn) + 1)
            return best

        ans = 0
        rows, columns = len(matrix), len(matrix[0])
        for i in range(rows):
            for j in range(columns):
                ans = max(ans, dfs(i, j))
        return ans
```