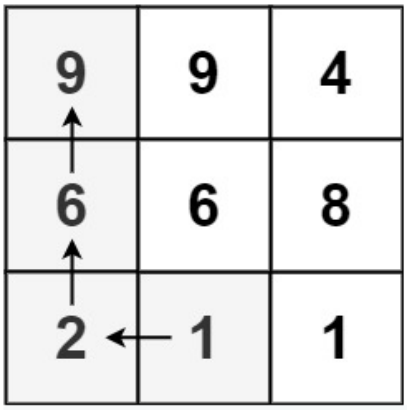
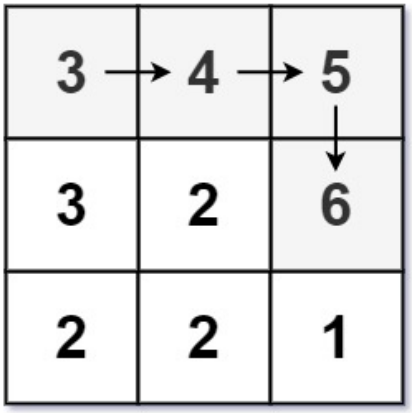
Longest_Increasing_Path_In_A_Matrix

Example 1:




```
Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]
Output: 4
Explanation: The longest increasing path is [1, 2, 6, 9].
```

Example 2:




```
Input: matrix = [[3,4,5],[3,2,6],[2,2,1]]
Output: 4
Explanation: The longest increasing path is [3, 4, 5, 6]. Moving diagonally is not allowed.
```

접근 방식

1. 연산 상 특정 배열 칸에 여러번 방문하게 됨.
2. 방문 했을 때, 최대 길이의 대한 결과를 가지고 있으면, 추가적인 연산 불필요.
3. Dp를 통한 Memoization 을 해놓고, DFS 를 통해 최대 길이 갱신

Matrix 상에서 나보다 큰 값이면 탐색을 실시하여서,

이미 dp가 계산된 값이면 리턴받고, 계산이 안된곳이면 끝까지 탐색 후에 종료시점에 1씩

역연산으로 + 해주면서 시작지점까지의 거리를 갱신한다.

```
Dp              Matrix

0  0  0          9  9  4

0  0  0          6  6  8

0  0  0          2  1  1

(0,1)          (0,2)       (0,3)
1  0  0        1  1  0     1  1  2

0  0  0        0  0  0     0  0  1

0  0  0        0  0  0     0  0  0


(1,1)           (1,2)       (1,3)

1  1  2       1  1  2       1  1  2

2  0  1       2  2  1       2  2  1

0  0  0       0  0  0       0  0  0


(2,1)           (2,2)       (2,3)

1  1  2       1  1  2       1  1  2

2  2  1       2  2  1       2  2  1

3  0  0       3  4  0       3  4  2
```

코드

```
class Solution {

  static int[] dx = new int[] { -1, 1, 0, 0 };
  static int[] dy = new int[] {  0, 0, -1, 1};
  static int row, col;
  static int[][] dp;

  private static int dfs(int r, int c, int[][] matrix) {

    if(dp[r][c] != 0) {
      return dp[r][c];
    }
    int answer = 0;

    for(int i = 0; i < 4; i++) {
      int nr = r + dx[i];
      int nc = c + dy[i];

      if(nr >= 0 && nr < row && nc >= 0 && nc < col && matrix[nr][nc] > matrix[r][c]) {
        answer = Math.max(answer, dfs(nr,nc,matrix));
      }
```

```
    }
    dp[r][c] = answer + 1;

    return dp[r][c];
  }
    public static int longestIncreasingPath(int[][] matrix) {

      int answer = 0;
      row = matrix.length;
      col = matrix[0].length;

      if(row == 0) {
        return 0;
      }

      dp = new int[row][col];

      for(int i = 0; i < row; i++) {
        for(int j = 0; j < col; j++) {
          answer = Math.max(answer, dfs(i,j, matrix));
        }
      }

      return answer;
    }
}
```