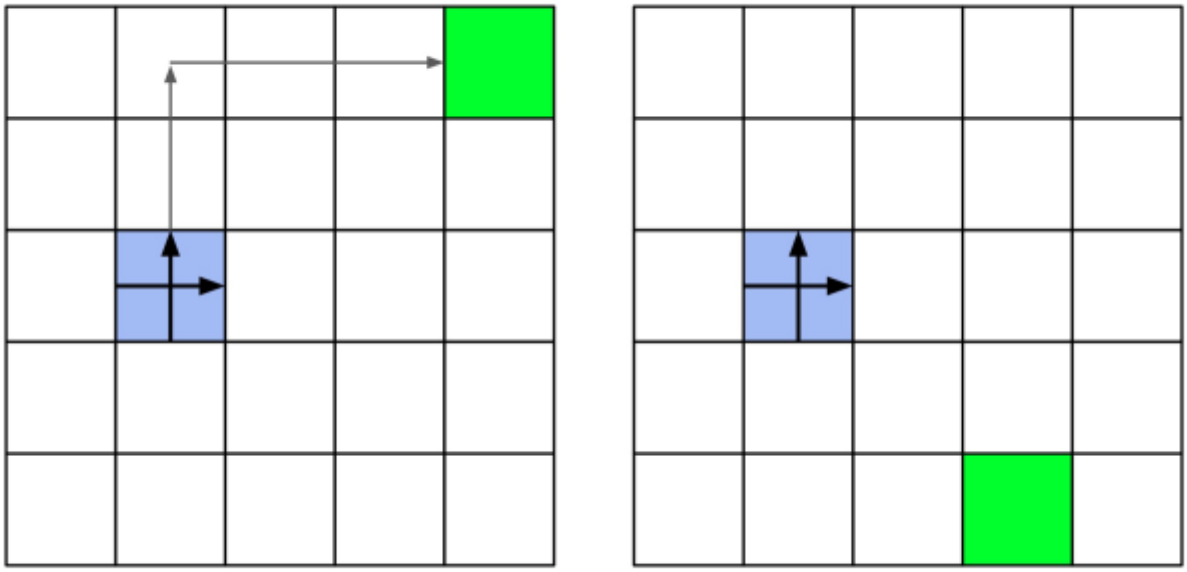
Access to Ant Food in a Grid World_____

Problem Statement

Consider a grid-world that is inhabited by an ant (BLUE). The ant can move only in two directions: UP or RIGHT. The ant has sensed the presence of a source of food somewhere in the grid. Your task is twofold:

- Determine if the ant can reach the food source.
- If it can, find out the number of steps it has to take.
  For example, in the grid-world on the left, the ant can reach the food source in five steps. On the right, it can't.



The grid-world is represented as a matrix of strings: 'B' stands for the initial position of the ant, 'W' stands for an empty cell and 'G' stands for the food source. For example, the grid-world on the left is represented as:
[W W W W G]
[W W W W W]
[W A W W W]
[W W W W W]
[W W W W W]

Write a function named is_reachable that accepts a
n×n matrix of strings named grid as argument. Return (True, steps) if the ant can reach the food source, where steps is the number of steps the ant needs to take. If it can't reach the food source, return (False, None)

Input Format

The input consists of an n × n grid-world represented as a matrix of characters. Each character represents the content of a cell in the grid. The grid contains three types of characters:

'B': Represents the initial position of the ant.
'G': Represents the location of the food source.
'W': Represents an empty cell.
The input grid is provided as a list of lists, where each inner list contains n characters representing the row of the grid.

[['W', 'W', 'W', 'W', 'W'],

```
    ['W', 'W', 'B', 'W', 'W'],
    ['W', 'W', 'W', 'W', 'W'],
    ['W', 'W', 'W', 'W', 'W'],
    ['G', 'W', 'W', 'W', 'W']
]
```

Output Format

The output consists of a tuple that provides information about the ant's ability to reach the food source and, if possible, the number of steps required. The output tuple contains two elements:

A Boolean value indicating whether the ant can reach the food source:

True if the ant can reach the food source.
False if the ant cannot reach the food source.

An integer value representing the number of steps required for the ant to reach the food source. This value is relevant only if the ant can reach the food source. If the ant cannot reach the food source, the second element of the tuple is 'None'

Example
(True, 5)

Solution

Solution in Python :

```
def is_reachable(grid):
    n = len(grid)
    for i in range(n):
        for j in range(n):
            if grid[i][j] == 'G':
                food = (i, j)
            elif grid[i][j] == 'B':
                ant = (i, j)
    if food[0] <= ant[0] and food[1] >= ant[1]:
        steps = (ant[0] - food[0]) + (food[1] - ant[1])
        return (True, steps)
    return (False, None)


grid = [
    ['W', 'W', 'W', 'W', 'W'],
    ['W', 'W', 'B', 'W', 'W'],
    ['W', 'W', 'W', 'W', 'W'],
    ['W', 'W', 'W', 'W', 'W'],
    ['G', 'W', 'W', 'W', 'W']
]

solution=is_rechable(grid)
print(solution)
```

Solution in Java:

```
class ReachabilityResult {
   boolean reachable;
   Integer steps;

   public ReachabilityResult(boolean reachable, Integer steps) {
      this.reachable = reachable;
      this.steps = steps;
   }
}

public class AntFoodReachability {

   public static ReachabilityResult isReachable(char[][] grid) {
      int n = grid.length;
      int antRow = -1, antCol = -1, foodRow = -1, foodCol = -1;

      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
            if (grid[i][j] == 'B') {
               antRow = i;
               antCol = j;
            } else if (grid[i][j] == 'G') {
               foodRow = i;
               foodCol = j;
            }
         }
      }

      // Check if ant can reach food
      if (foodRow <= antRow && foodCol >= antCol) {
         int steps = (antRow - foodRow) + (foodCol - antCol);
         return new ReachabilityResult(true, steps);
      }

      return new ReachabilityResult(false, null);
   }

   public static void main(String[] args) {
      char[][] grid = {
         {'W', 'W', 'W', 'W', 'W'},
         {'W', 'W', 'B', 'W', 'W'},
         {'W', 'W', 'W', 'W', 'W'},
         {'W', 'W', 'W', 'W', 'W'},
         {'G', 'W', 'W', 'W', 'W'}
      };

      ReachabilityResult result = isReachable(grid);

      if (result.reachable) {
         System.out.println("(True, " + result.steps + ")");
      } else {
         System.out.println("(False, None)");
      }
   }
}
```

Alternative solution using BFS:

```
import java.util.LinkedList;
import java.util.Queue;

public class AntFoodReachability {

    public static void main(String[] args) {
        char[][] grid = {
            {'W', 'W', 'W', 'W', 'G'},
            {'W', 'W', 'W', 'W', 'W'},
            {'W', 'W', 'W', 'W', 'W'},
            {'W', 'W', 'W', 'W', 'W'},
            {'W', 'W', 'B', 'W', 'W'}
        };

        ReachabilityResult result = isReachable(grid);

        if (result.reachable) {
            System.out.println("(True, " + result.steps + ")");
        } else {
            System.out.println("(False, None)");
        }
    }

    public static ReachabilityResult isReachable(char[][] grid) {
        if (grid == null || grid.length == 0 || grid[0].length == 0) {
            return new ReachabilityResult(false, null);
        }

        int n = grid.length;
        int m = grid[0].length;
        int[][] directions = {{-1, 0}, {1, 0}, {0, -1}, {0, 1}};

        int antX = -1;
        int antY = -1;

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == 'B') {
                    antX = i;
                    antY = j;
                    break;
                }
            }
        }

        Queue<Cell> queue = new LinkedList<>();
        queue.offer(new Cell(antX, antY, 0));

        boolean[][] visited = new boolean[n][m];
        visited[antX][antY] = true;

        while (!queue.isEmpty()) {
            Cell current = queue.poll();

            for (int[] dir : directions) {
                int newX = current.x + dir[0];
                int newY = current.y + dir[1];

                if (newX >= 0 && newX < n && newY >= 0 && newY < m && grid[newX][newY] == 'G') {
                    return new ReachabilityResult(true, current.steps + 1);
                }
```

```
            if (newX >= 0 && newX < n && newY >= 0 && newY < m && grid[newX][newY] == 'W'
&& !visited[newX][newY]) {
                queue.offer(new Cell(newX, newY, current.steps + 1));
                visited[newX][newY] = true;
            }
        }
    }

    return new ReachabilityResult(false, null);
  }

  static class ReachabilityResult {
    boolean reachable;
    Integer steps;

    public ReachabilityResult(boolean reachable, Integer steps) {
      this.reachable = reachable;
      this.steps = steps;
    }
  }

  static class Cell {
    int x;
    int y;
    int steps;

    public Cell(int x, int y, int steps) {
      this.x = x;
      this.y = y;
      this.steps = steps;
    }
  }
}
```

Test Cases

Test case ID	Input	Output	Weight	Comments
1	W,W,W,W,W,W,W W,W,W,W,W,W,W W,W,W,W,W,W,W W,W,W,W,W,W,W W,W,W,W,G,W,W W,W,W,W,W,W,W W,W,W,B,W,W,W	(True, 3)	30	Interesting
2	W,W,W,W,W W,W,W,W,W W,W,W,G,W W,W,W,W,W W,W,W,B,W	(True, 2)	20	Basic
3	W,W,W,W,W,W,W W,W,W,W,W,W,W W,W,W,W,W,W,W B,W,W,W,W,W,G W,W,W,W,W,W,W W,W,W,W,W,W,W W,W,W,W,W,W,W	(True, 6)	20	Basic

4	W,W,W,W,W,W,W,W,W,W W,W,W,W,W,W,W,W,W,W W,W,W,W,W,W,W,W,W,W W,W,W,W,W,W,W,W,W,W W,W,W,W,W,W,W,W,W,W W,W,W,W,W,W,W,W,W,W W,W,W,W,W,W,W,W,W,W W,W,W,W,W,W,W,W,B,W,W W,W,W,W,W,W,W,W,W,W W,W,W,W,W,W,W,G,W,W,W W,W,W,W,W,W,W,W,W,W	(False, None)	30	Interesting