```java
/*
        Time complexity: O(N*N)
        Space complexity: O(N*N)
        where N is the size of cake
*/


public class Solution {

    static int[][] dir = { { 1, 0 }, { -1, 0 }, { 0, 1 }, { 0, -1 } };

        public static int dfs(String[] edge, int n) {
                boolean[][] visited = new boolean[n][n];

                int ans = 0;
                for (int i = 0; i < n; i++) {
                        for (int j = 0; j < n; j++) {

                                if (visited[i][j] == false && edge[i].charAt(j) == '1') {

                                        ans = Math.max(ans, __dfs(edge, visited, i, j, n));
                                }
                        }
                }

                return ans;
        }

        private static int __dfs(String[] edge, boolean[][] visited, int x, int y, int n) {

                visited[x][y] = true;
                int count = 1;

                for (int i = 0; i < 4; i++) {

                        int nex = x + dir[i][0];
                        int ney = y + dir[i][1];

                        if (valid(nex, ney, n) && edge[nex].charAt(ney) == '1' && visited[nex][ney] == false) {

                                count += __dfs(edge, visited, nex, ney, n);
                        }
                }
                return count;
        }

        private static boolean valid(int x, int y, int n) {
                return (x >= 0 && y >= 0 && x < n && y < n);
        }

}
```