## The Problem

Consider a black and white image, where every pixel has a value of either **0** or **1**. Define a region of the image as a collection of pixels that all have the same value, and are interconnected. Specifically, for any two pixels in a region, there is a path between them only going up, down, left or right, and only going through other pixels with the same value.

You wish to see a border completely around every region in the image. You can choose certain regions to draw a border around; when you do, you draw a border around the entire region, including any internal “holes”. If two regions are adjacent, then you can supply the border between then by drawing the border around one, or the other, or both. What is the minimum number of regions you need to draw a border around in order to ensure that every region has a border?

Consider these examples:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | **0** | **0** |  | **0** | **0** | **0** |  | **0** | **1** | **0** |
| **1** | **1** | **1** |  | **0** | **1** | **0** |  | **1** | **0** | **1** |
| **0** | **0** | **0** |  | **0** | **0** | **0** |  | **0** | **1** | **0** |

In the first case, the minimum is three. Because they’re on the edges, there is no choice but to draw a border around all three.

In the second case, the minimum is one. Drawing a border around the **0** region puts a border around the **1** region.

Tn the third case, the answer is eight. Drawing a border around all of the regions on the edges puts a border around the center region.

## Algorithm

A solving program must first identify the regions. It must then count, and eliminate, the regions on the edges of the image. Then, they must build a graph, where each region is a node, and there’s an edge between nodes if their regions touch. Now, the problem is to cover all of the edges – it’s Vertex Cover, in a bipartite graph. Vertex Cover is NP-Complete in a general graph, but the limitations of a bipartite graph make it easier. Konig’s Theorem states that the size of the minimum vertex cover in a bipartite graph is equal to the maximum matches in the graph. So, then, any max matching algorithm can be used: Ford/Fulkerson, Edmunds/Karp, Dinic, Hopcroft/Karp, etc.

The problem lends itself to several simpler algorithms that don’t work (i.e. “cheese” solutions). First, teams may be tempted, after handling the edges, to simply count the number of ‘**0**’ regions, and the number of ‘**1**’ regions, and use the smaller number. This will generate a solution, but not a minimum solution. They may also try a greedy solution, choosing the nodes with the highest degree first. This also does not work.

## Difficulty

The difficulty of this problem is HIGH. It requires floodfill and max matching algorithms, and knowledge of Konig’s theorem. It also has several “cheese” solutions that don’t work, that can lure an unwary team.

|  |  |
| --- | --- |
| Case(s) | Description |
| 0000-0002 | Sample IO, taken from the problem statement |
| 1000 | Largest possible case, to break TLEs early |
| 1001 | Breaks the smallest-of-0s-and-1s “cheese” solution |
| 1002 | Breaks the greedy “cheese” solution |
| 1003-1007 | Various degenerate cases, where either n=1 or m=1 (or n=m=1) |
| 1008-1017 | 10 max size cases with varying densities |
| 1018-1027 | 10 totally random cases – random sizes and random densities |