

Problem F: Those who trespass against us

Manuel Wolz, Marvin Ewald, Tim Janiak

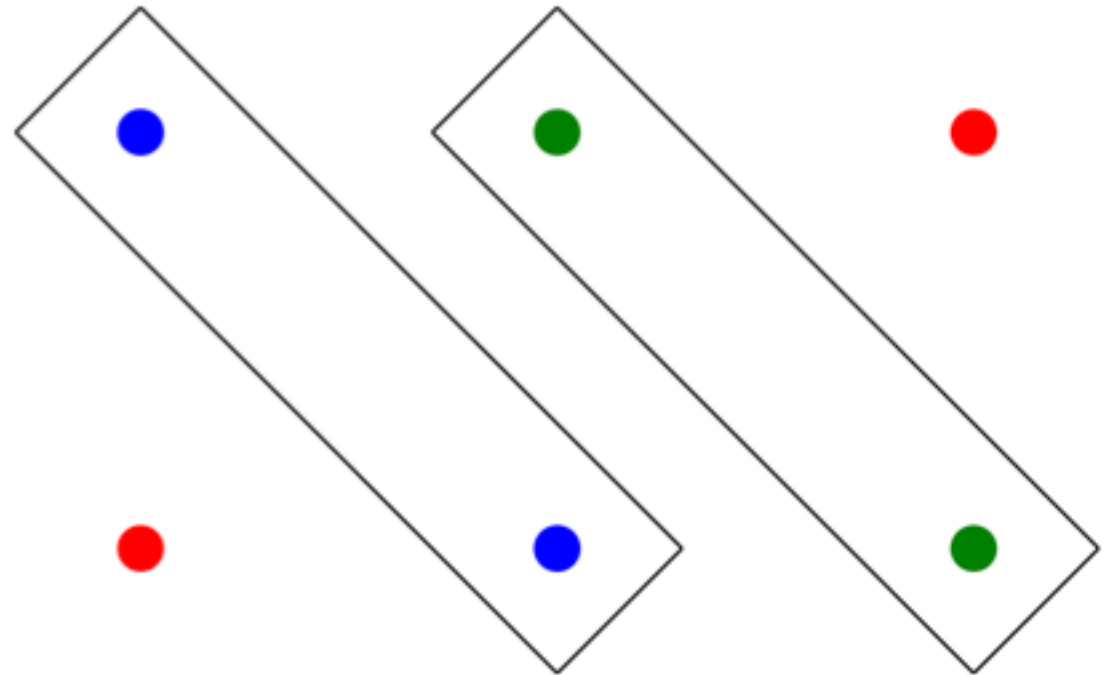
Description

Somewhere in the great North American plains live the tribes of chiefs *Blue Eagle*, *Red Beaver*, and *Green Serpent*. Their population is scattered over numerous villages all over the land and conflict arises whenever members of different tribes meet while traveling across the plains.

To put an end to the constant animosities the chiefs have decided that the **land should be divided between the tribes** so that they can avoid each other when moving between villages belonging to the same tribe. More precisely, they want to construct **two border fences** – thus dividing the land **into three regions** – such that two villages lie in the **same region** precisely when they belong to the **same tribe**.

Description

The villages are represented by points in the Euclidean plane that are colored blue, red or green, depending on the tribe, and the fences should be drawn in the form of two polygons. The polygons may not touch or intersect themselves or each other and none of the points may lie on their boundary.



Constraints

- Villages
 - At least one village per colour
 - In total 100 villages at most
 - On unique integer coordinates anywhere from -1000 to 1000

Constraints

- Villages
 - At least one village per colour
 - In total 100 villages at most
 - On unique integer coordinates anywhere from -1000 to 1000
- Fences
 - A maximum of 1000 vertices per polygon
 - Vertices on coordinates between -3000 and 3000 with up to five decimal places of precision

Input

- 1st line: Number of villages

6

0 0 2

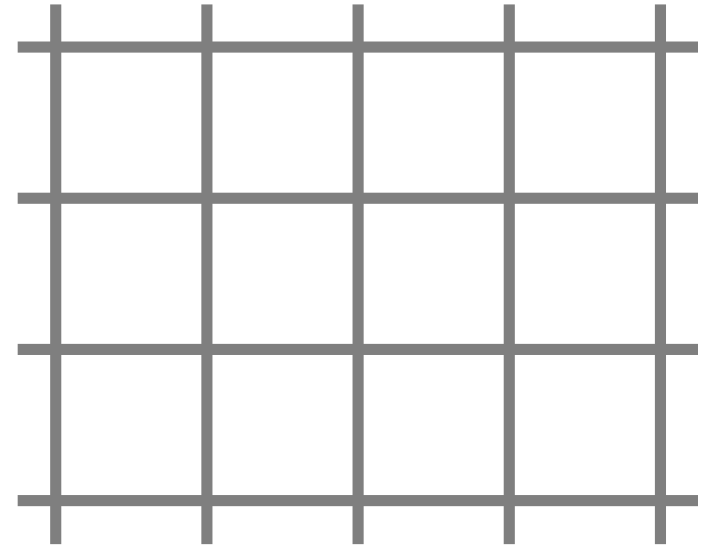
0 1 1

1 0 1

1 1 3

2 0 3

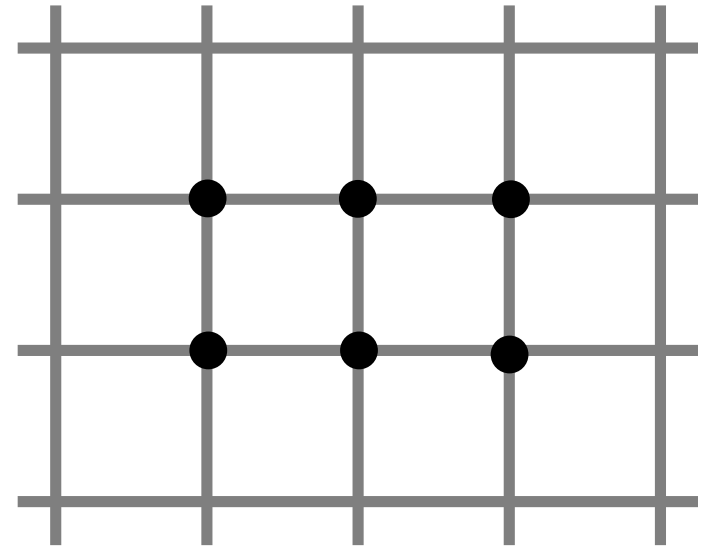
2 1 2



Input

- 1st line: Number of villages
- Subsequent lines correspond to one village each, containing:
 - x-coordinate
 - y-coordinate

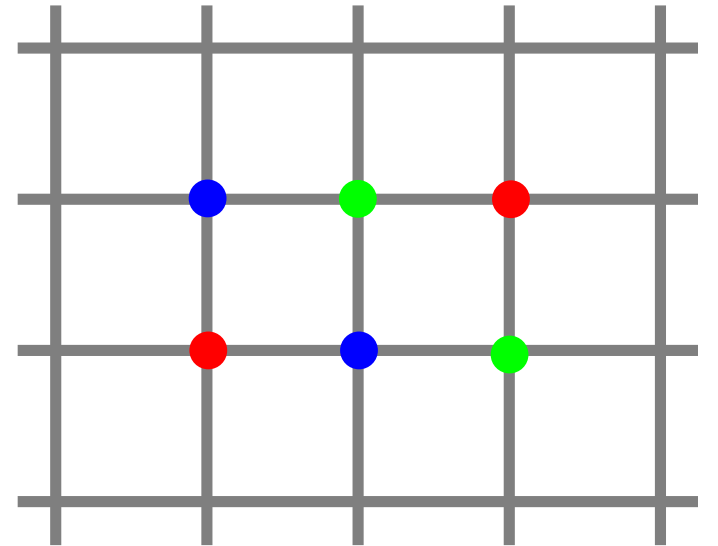
6		
0	0	2
0	1	1
1	0	1
1	1	3
2	0	3
2	1	2



Input

- 1st line: Number of villages
- Subsequent lines correspond to one village each, containing:
 - x-coordinate
 - y-coordinate
 - Colour (1: blue, 2: red, 3: green)

6		
0	0	2
0	1	1
1	0	1
1	1	3
2	0	3
2	1	2



Output

- 1st line: Number of vertices in one polygon

4

-0.3 1.0

1.0 -0.3

1.3 0.0

0.0 1.3

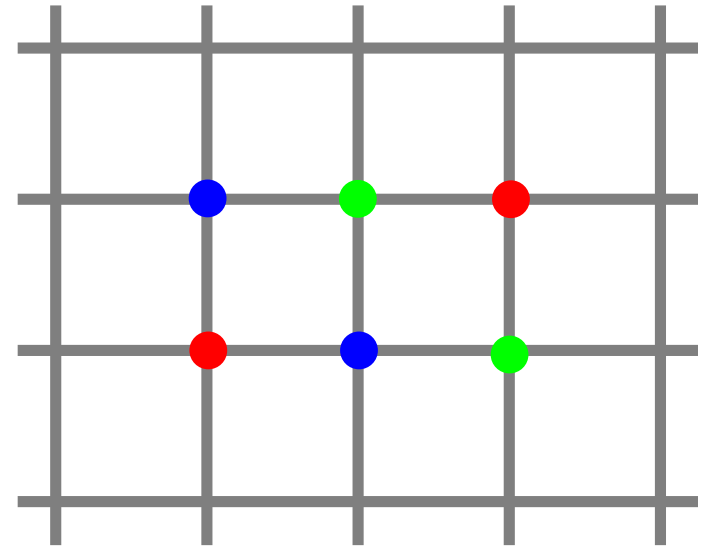
4

0.7 1.0

2.0 -0.3

2.3 0.0

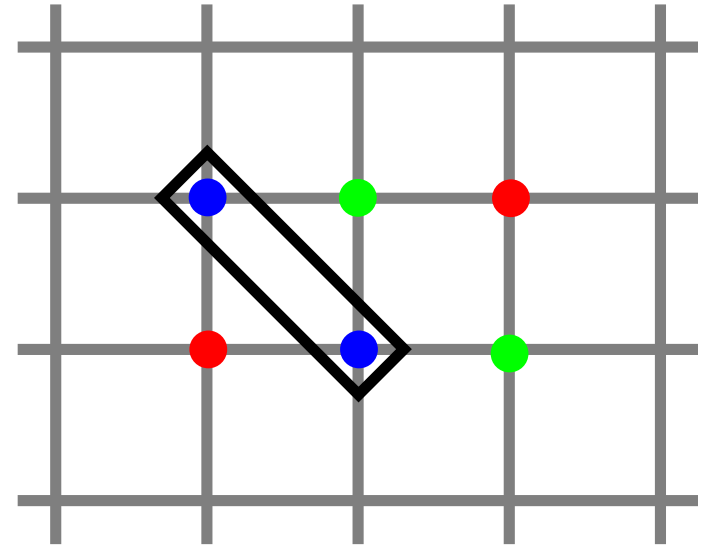
1.0 1.3



Output

- 1st line: Number of vertices in one polygon
- Subsequent lines correspond to the vertices in clockwise or counter-clockwise order, containing:
 - x-coordinate
 - y-coordinate

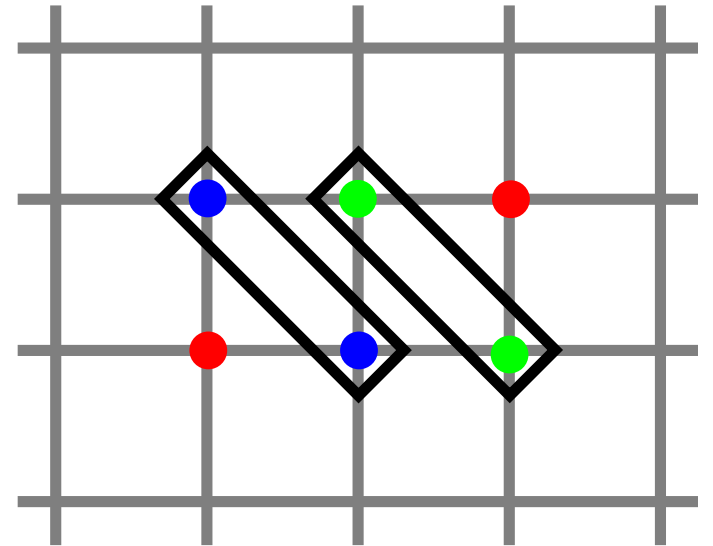
```
4
-0.3 1.0
1.0 -0.3
1.3 0.0
0.0 1.3
4
0.7 1.0
2.0 -0.3
2.3 0.0
1.0 1.3
```



Output

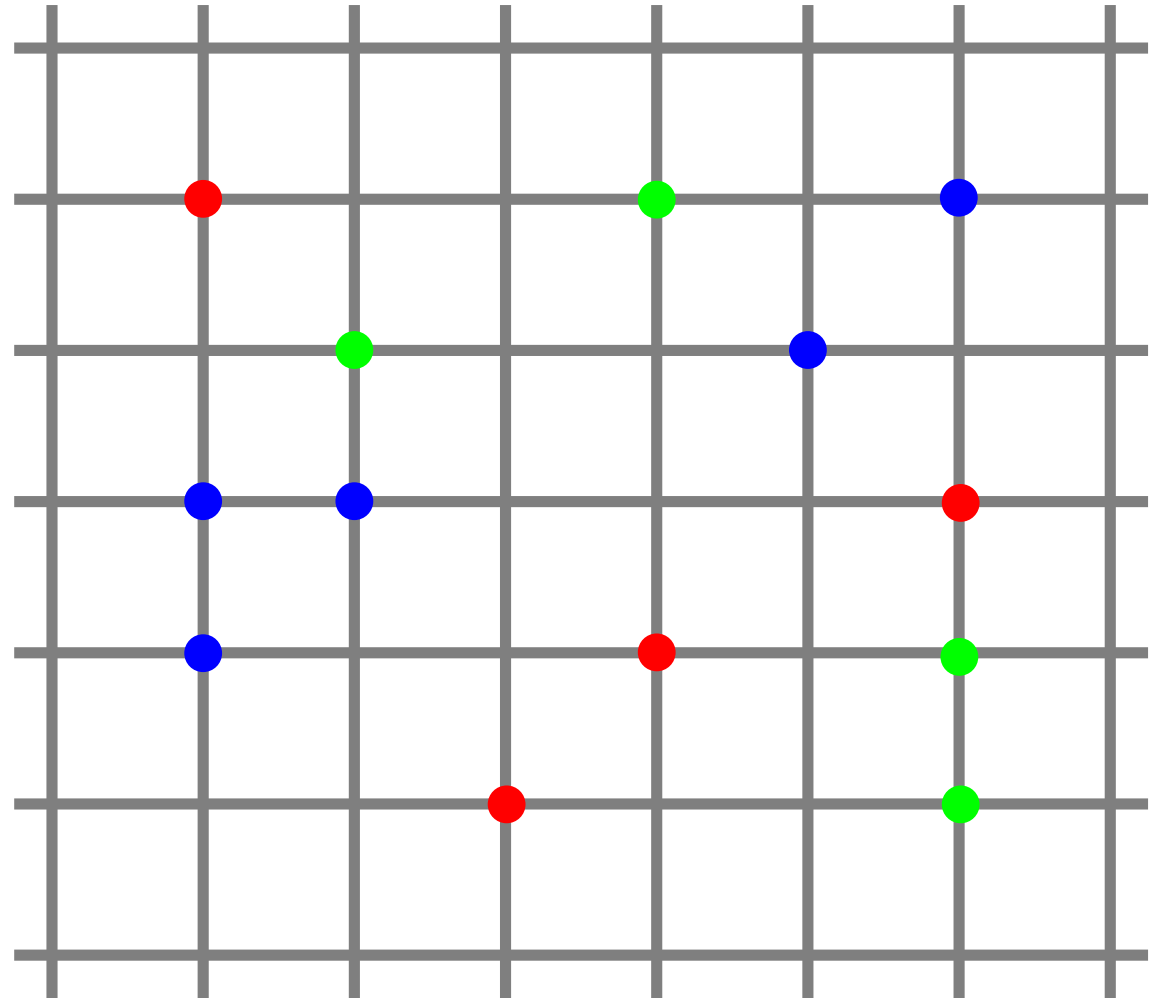
- 1st line: Number of vertices in one polygon
- Subsequent lines correspond to the vertices in clockwise or counter-clockwise order, containing:
 - x-coordinate
 - y-coordinate
- Repeat for the second polygon

```
4
-0.3 1.0
1.0 -0.3
1.3 0.0
0.0 1.3
4
0.7 1.0
2.0 -0.3
2.3 0.0
1.0 1.3
```



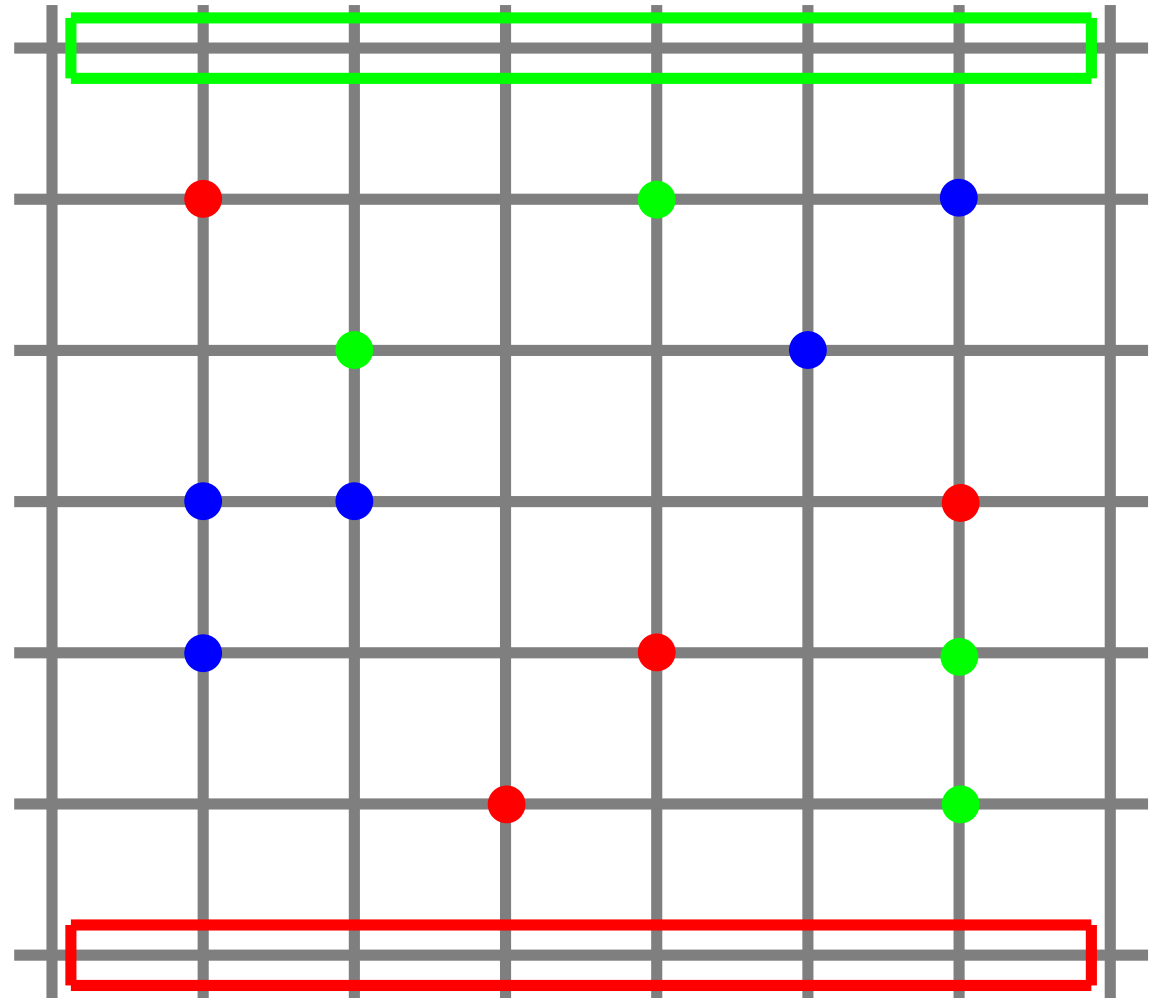
Idea

- Select the two colours with fewest villages



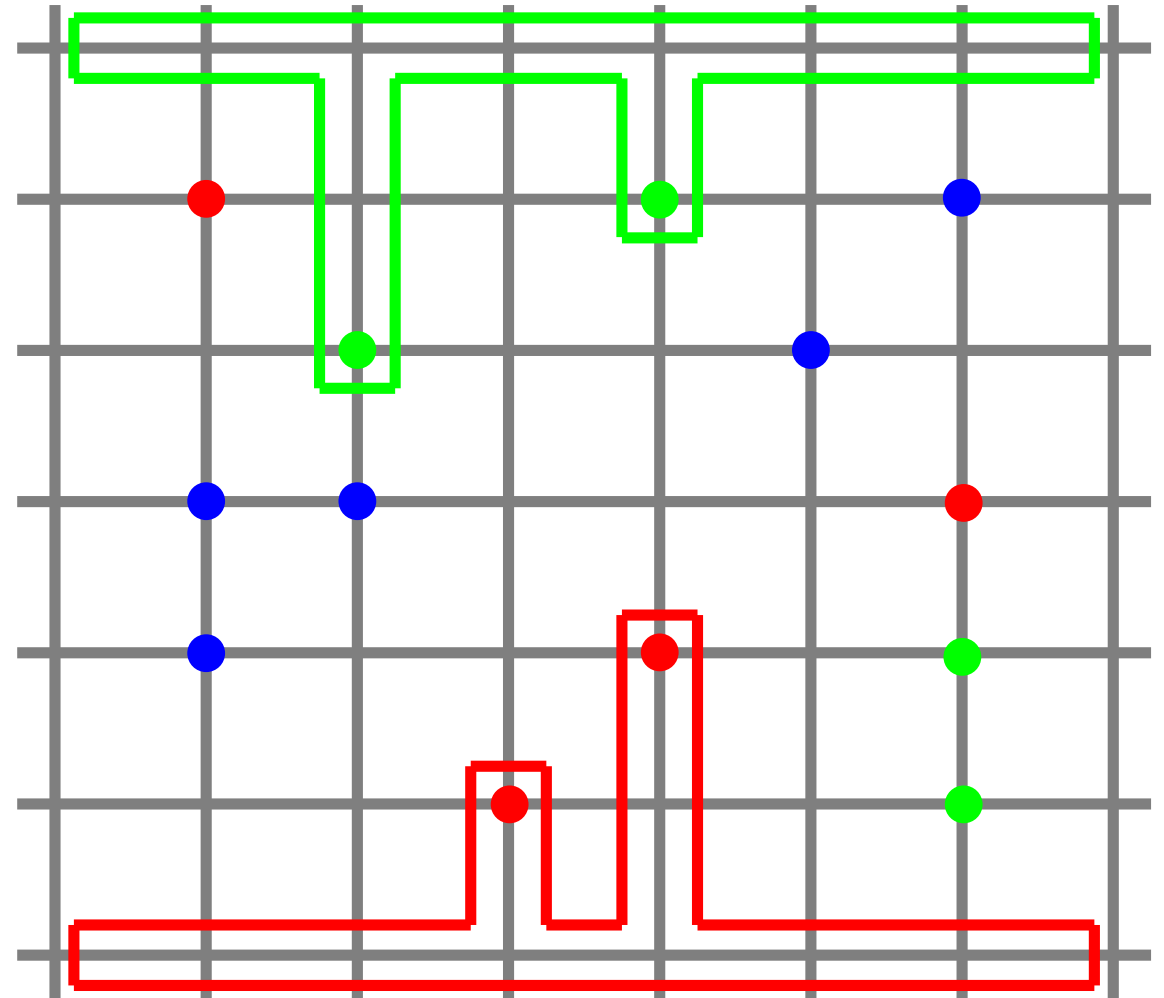
Idea

- Select the two colours with fewest villages
- Start polygons on opposite sides of the area with villages



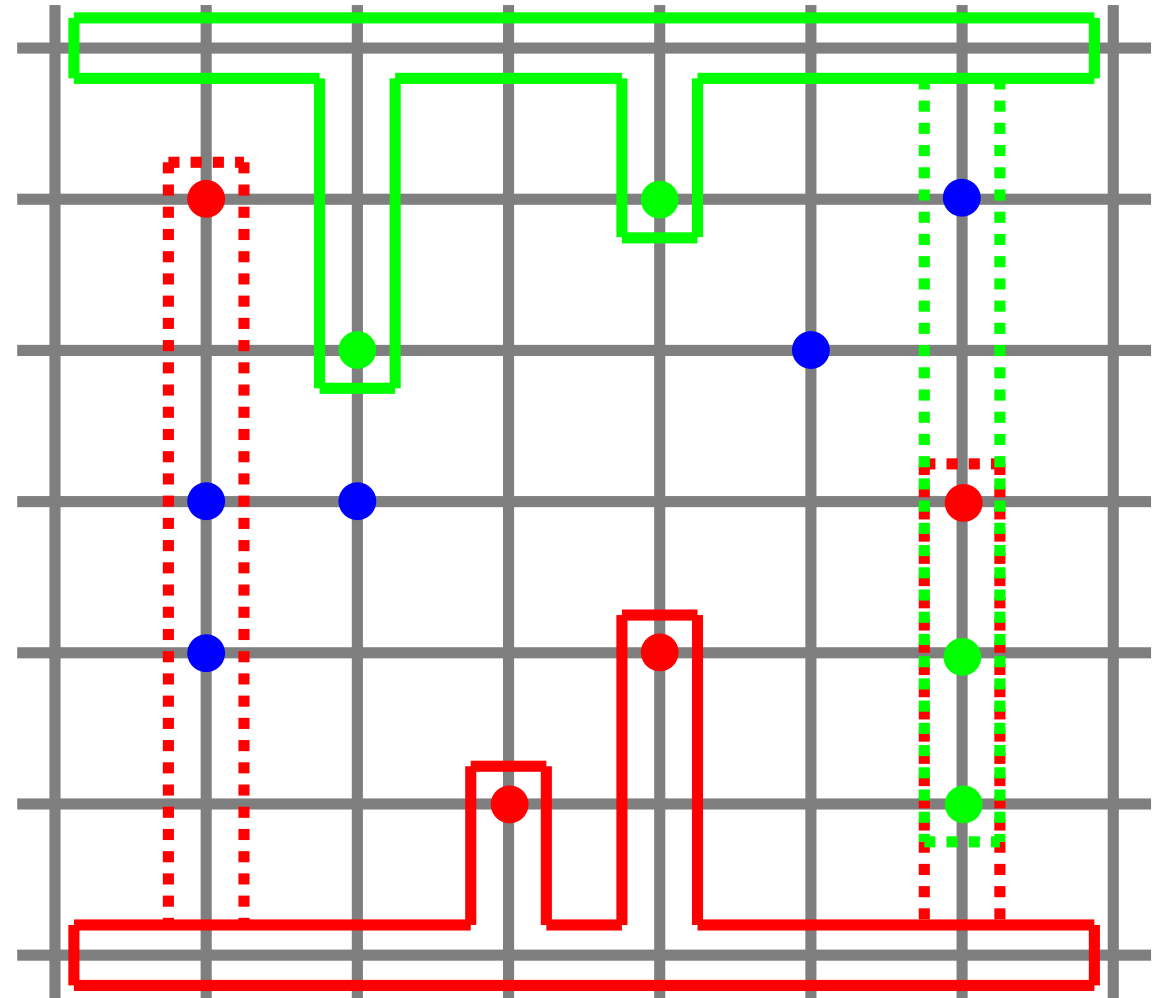
Idea

- Select the two colours with fewest villages
- Start polygons on opposite sides of the area with villages
- Connect the villages
 - Easy for some



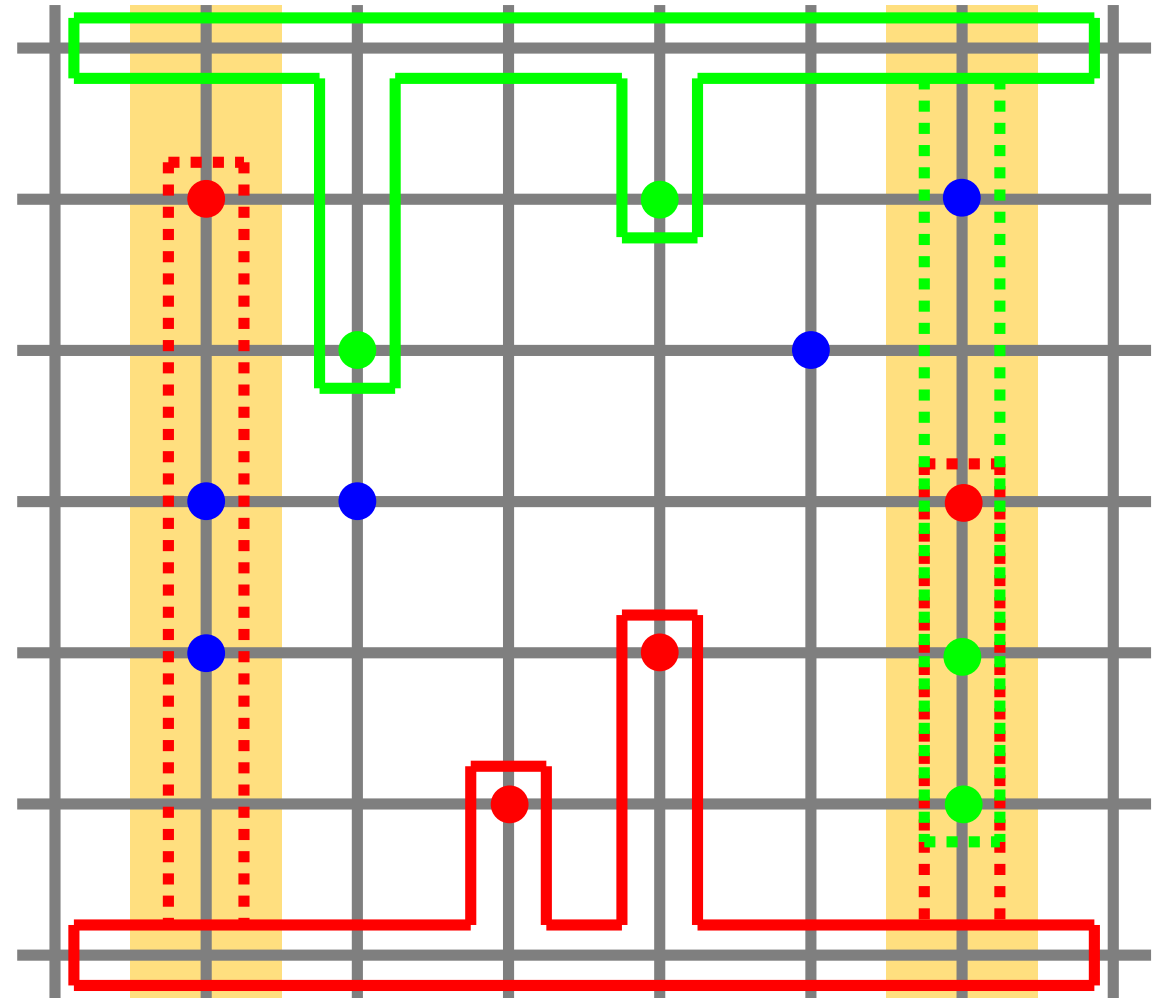
Idea

- Select the two colours with fewest villages
- Start polygons on opposite sides of the area with villages
- Connect the villages
 - Easy for some
 - Harder for others



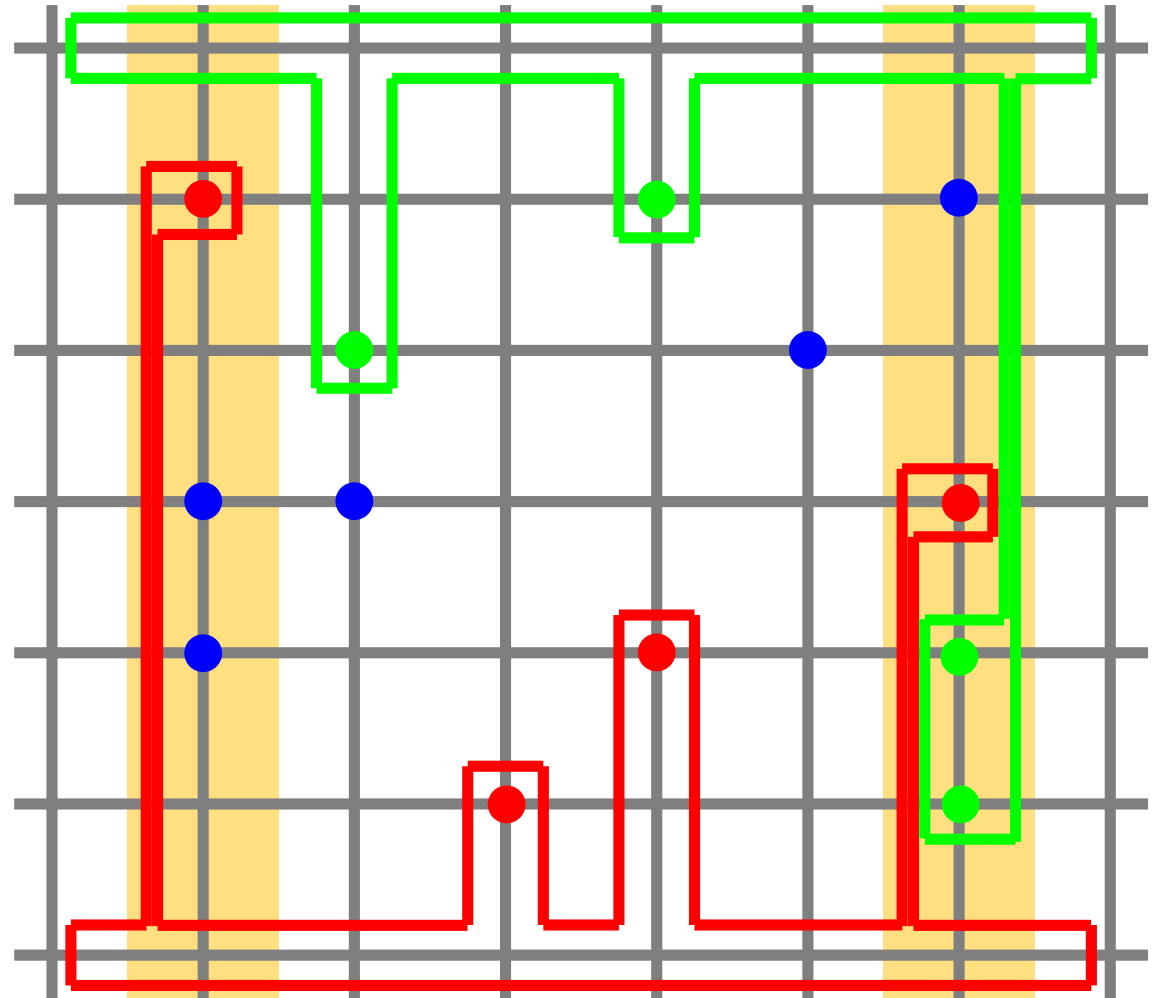
Idea

- Select the two colours with fewest villages
- Start polygons on opposite sides of the area with villages
- Connect the villages
 - Easy for some
 - Harder for others
 - use "wobble room" from -0.5 to +0.5 around the column



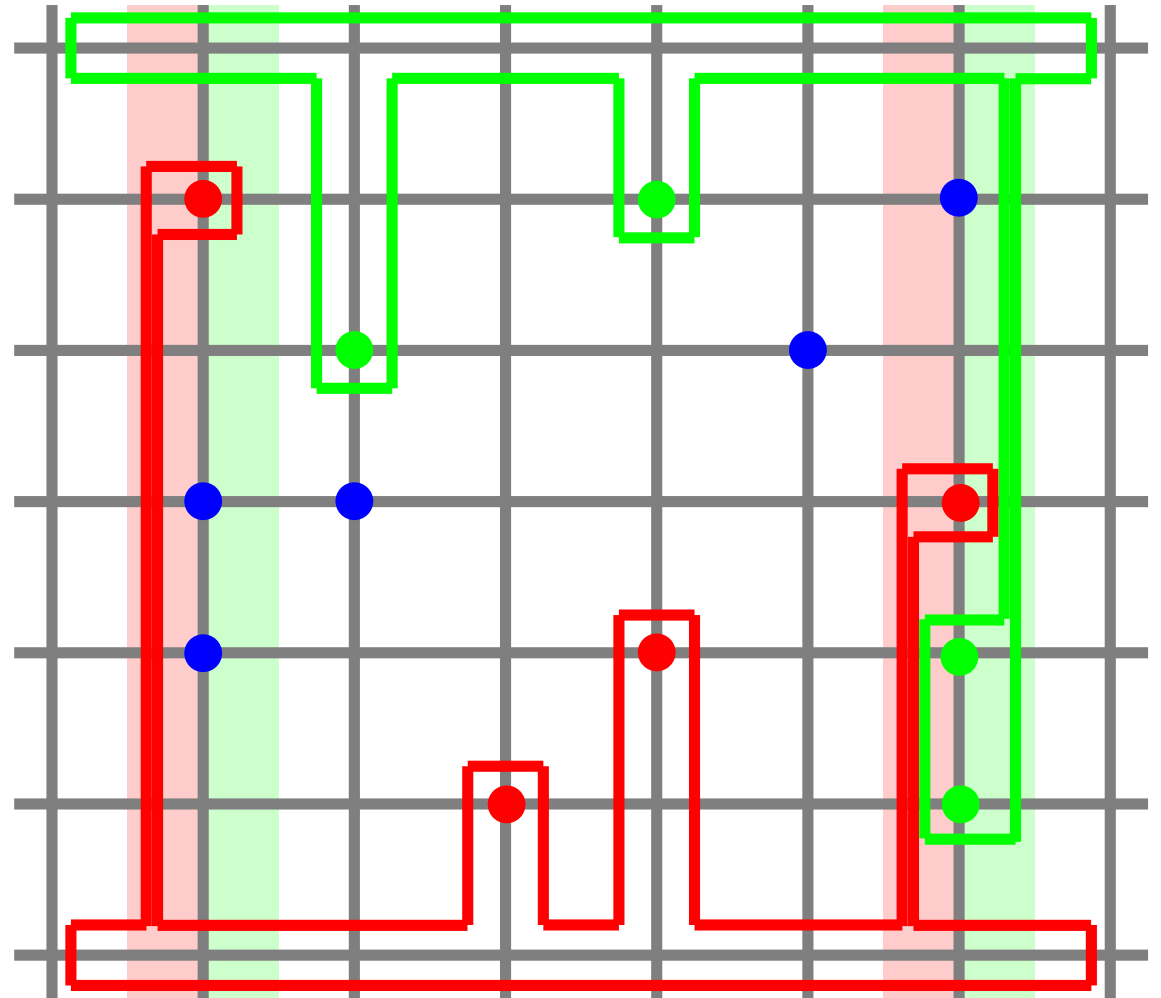
Idea

- Create connections which don't pass through other (potential) villages



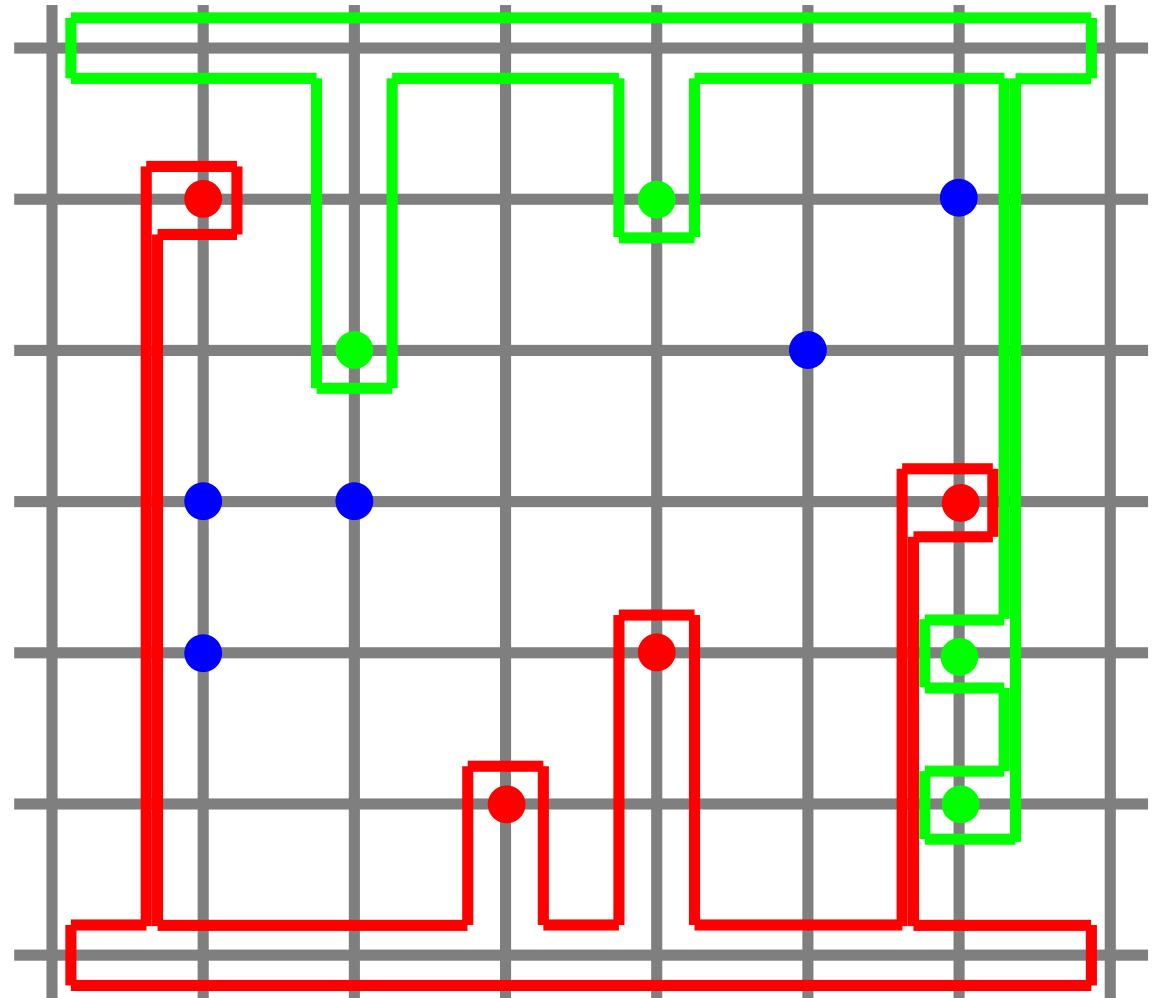
Idea

- Create connections which don't pass through other (potential) villages
- Use each half for one colour to prevent crossing



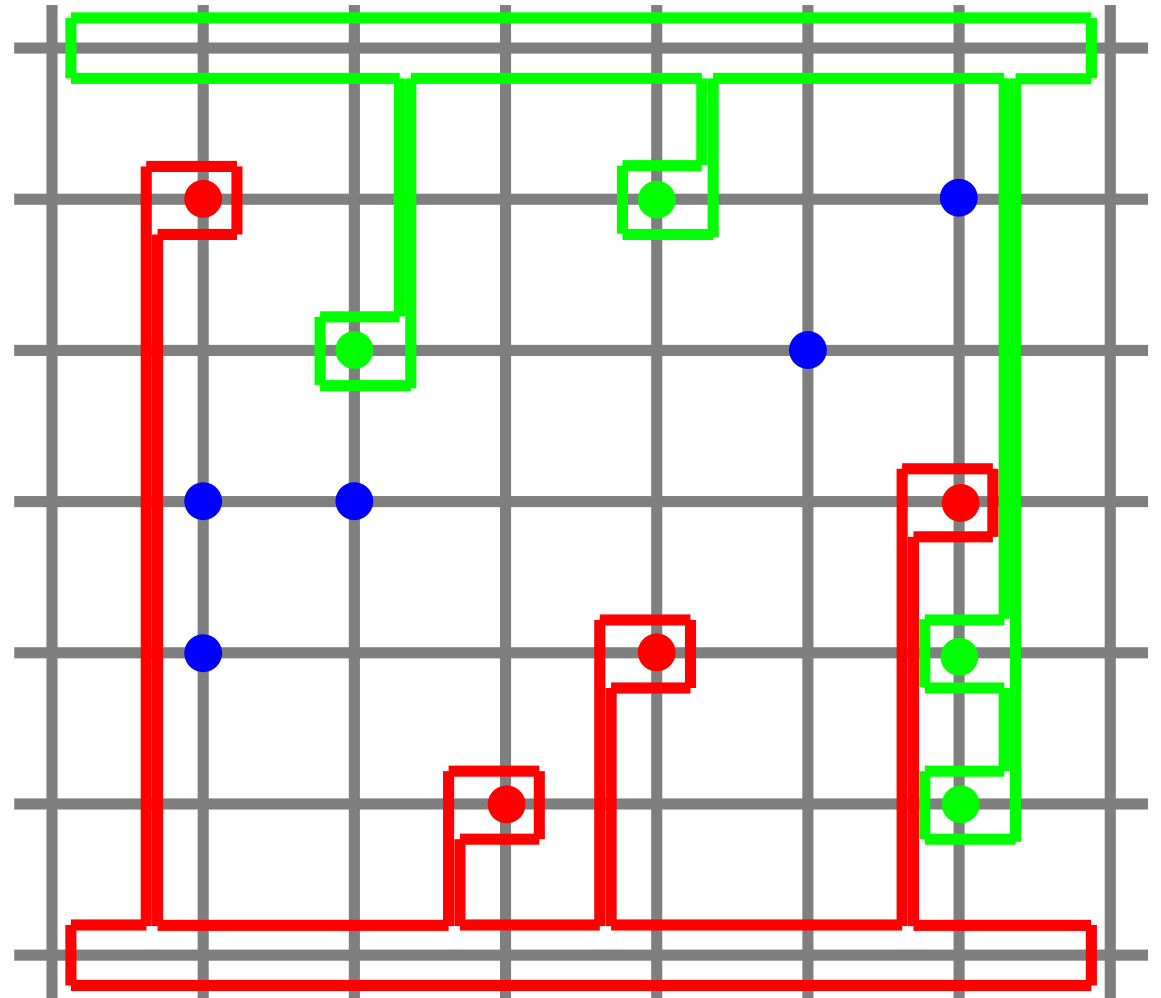
Idea

- Create connections which don't pass through other (potential) villages
- Use each half for one colour to prevent crossing
- More vertices but fewer special cases



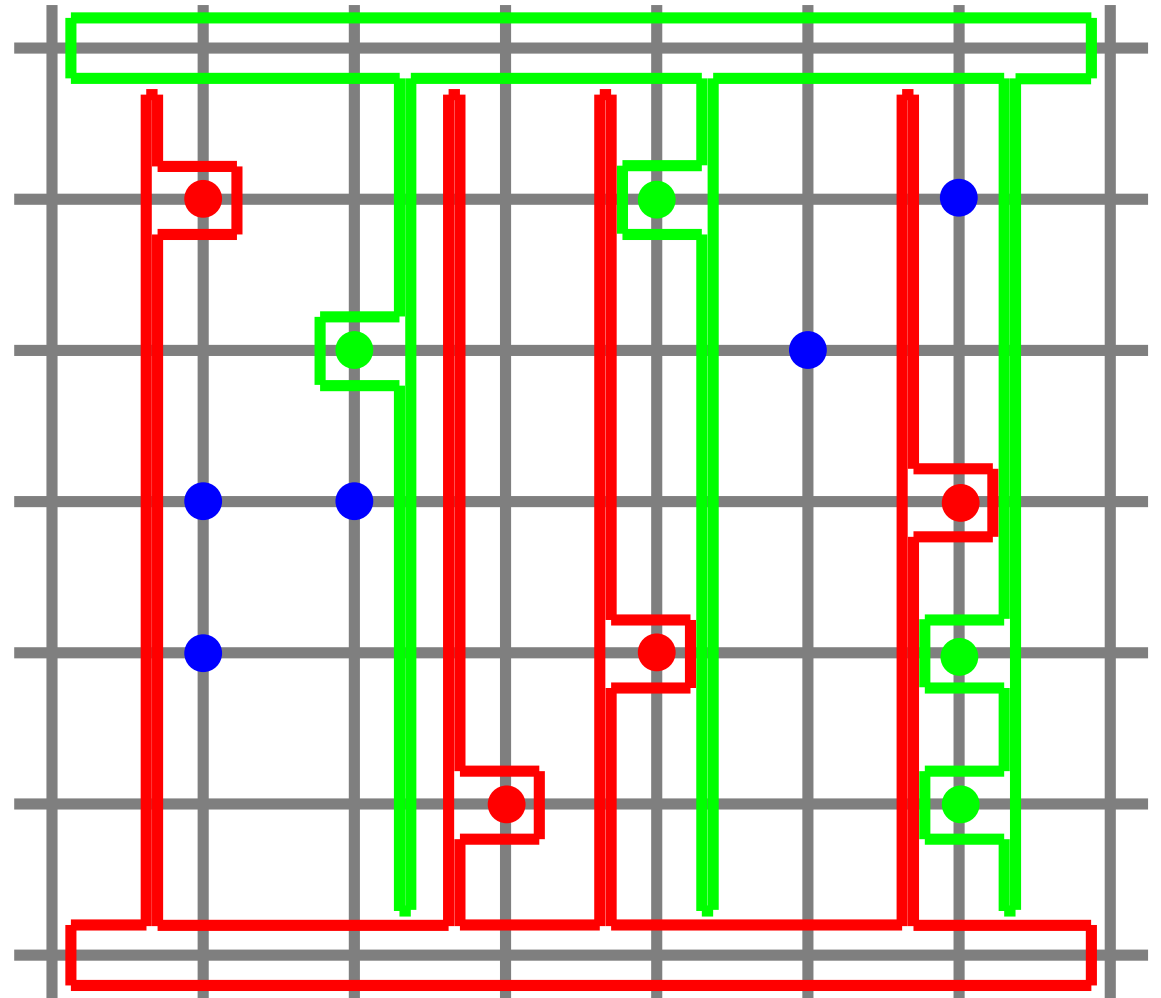
Idea

- Create connections which don't pass through other (potential) villages
- Use each half for one colour to prevent crossing
- More vertices but fewer special cases x2



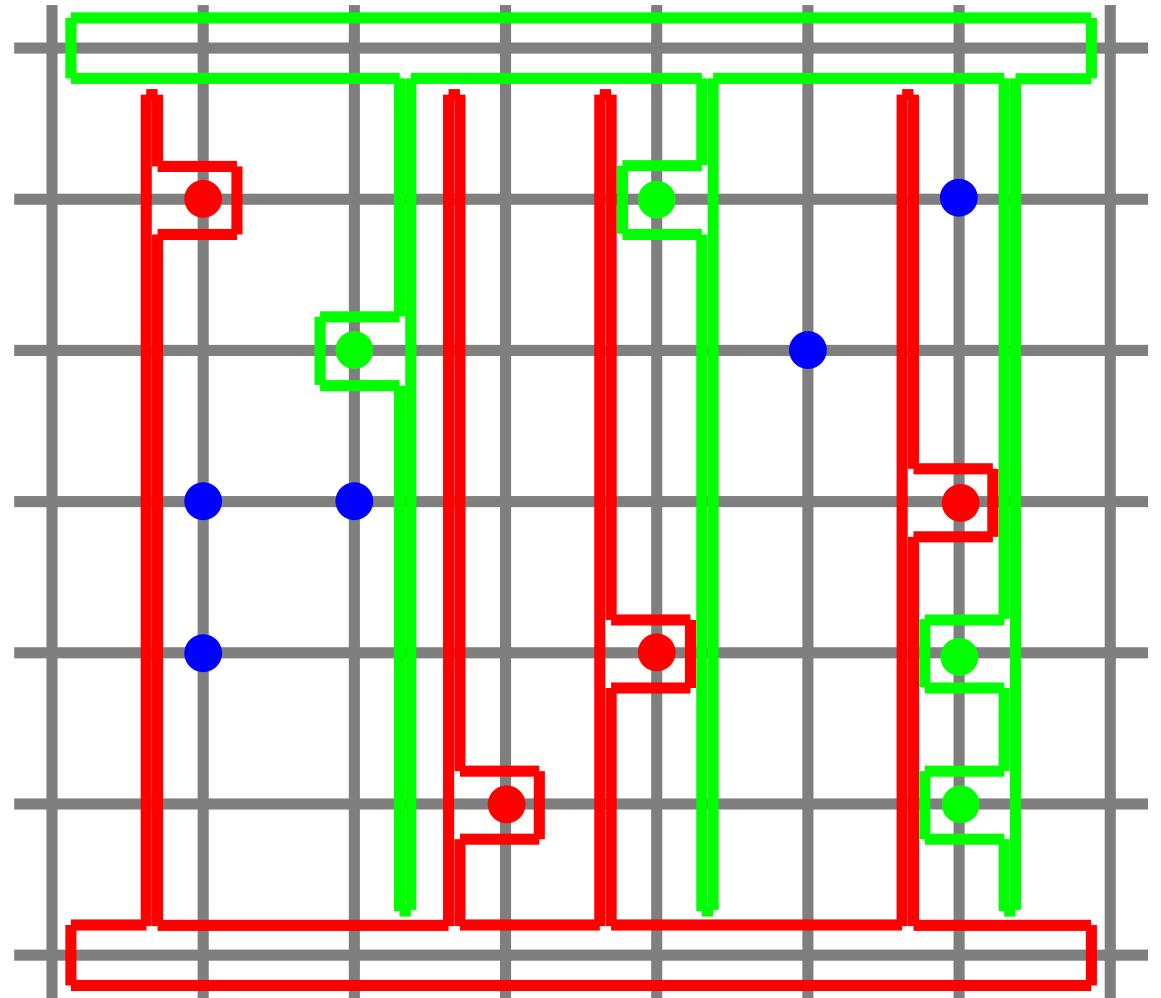
Idea

- Create connections which don't pass through other (potential) villages
- Use each half for one colour to prevent crossing
- More vertices but fewer special cases x3



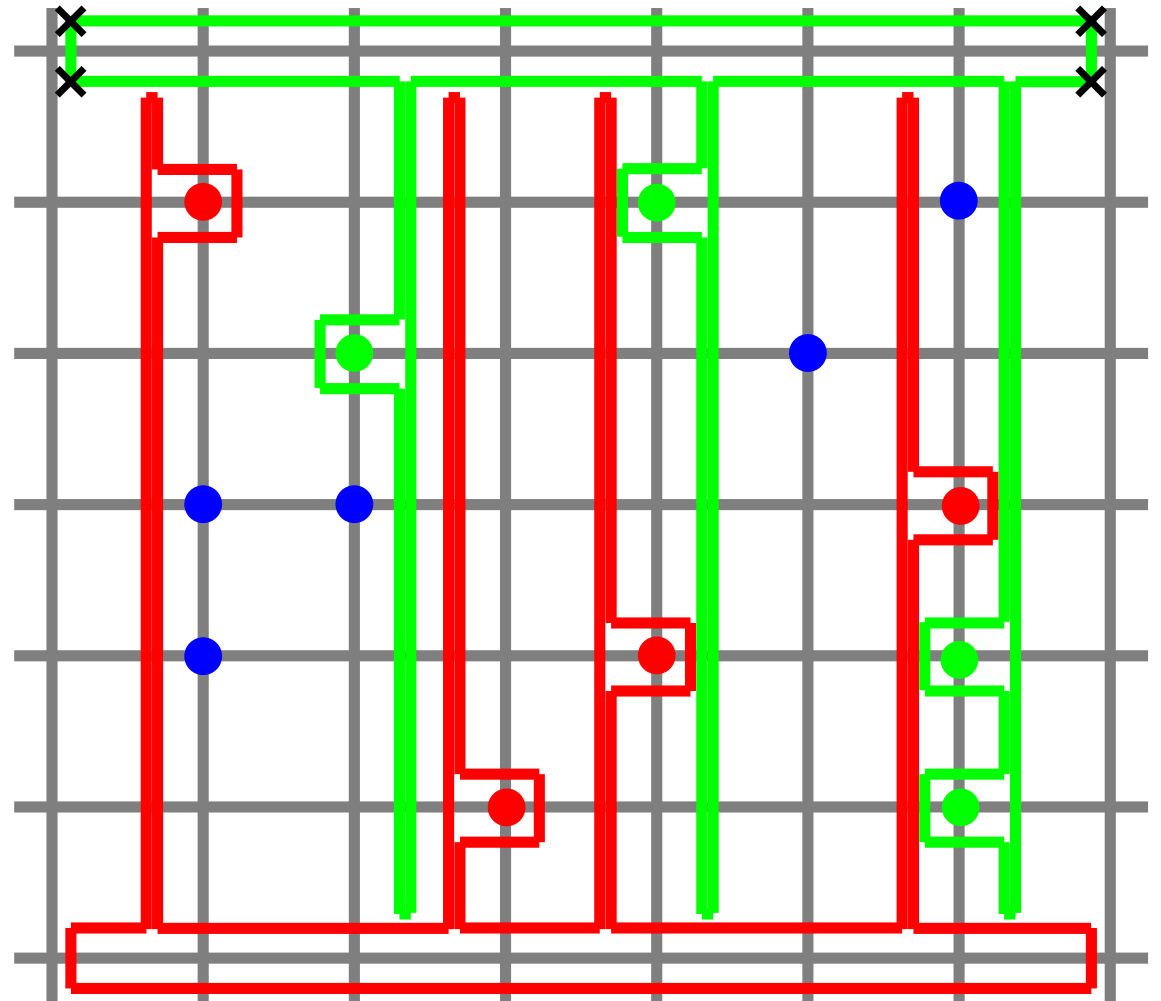
Are there too many vertices?

- Constraint: 1000 vertices per polygon



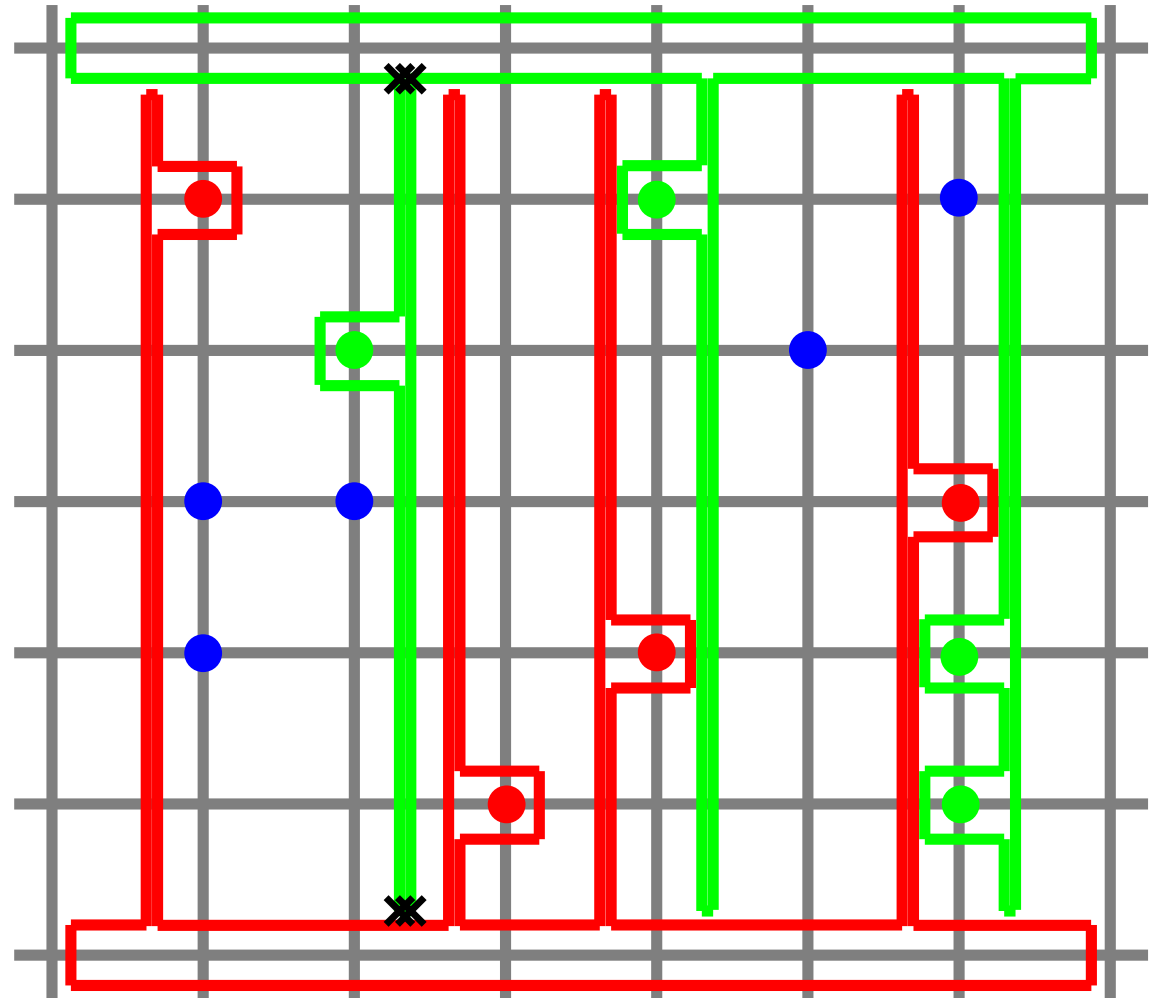
Are there too many vertices?

- Constraint: 1000 vertices per polygon
- 4 for the hub



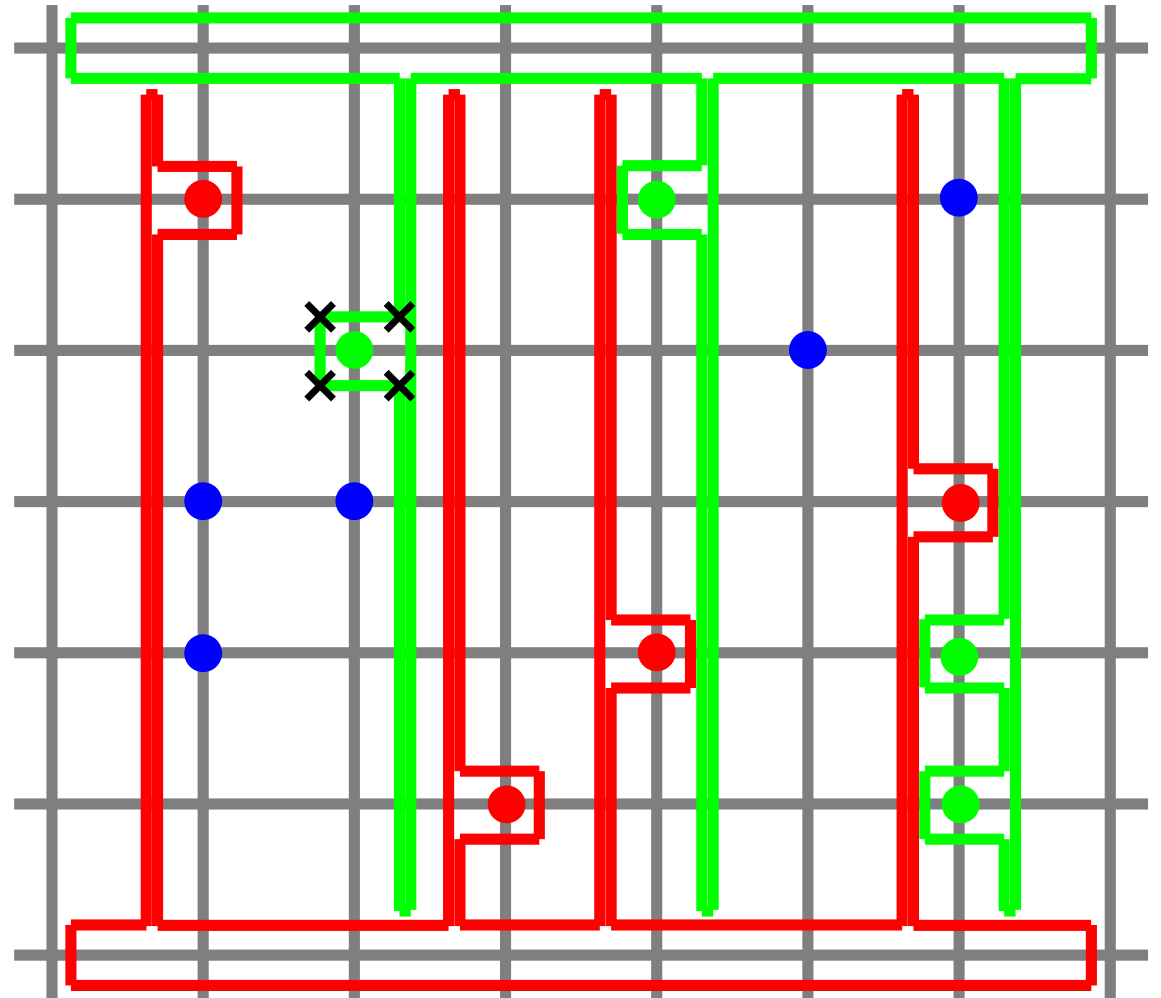
Are there too many vertices?

- Constraint: 1000 vertices per polygon
- 4 for the hub
- 4 per occupied column



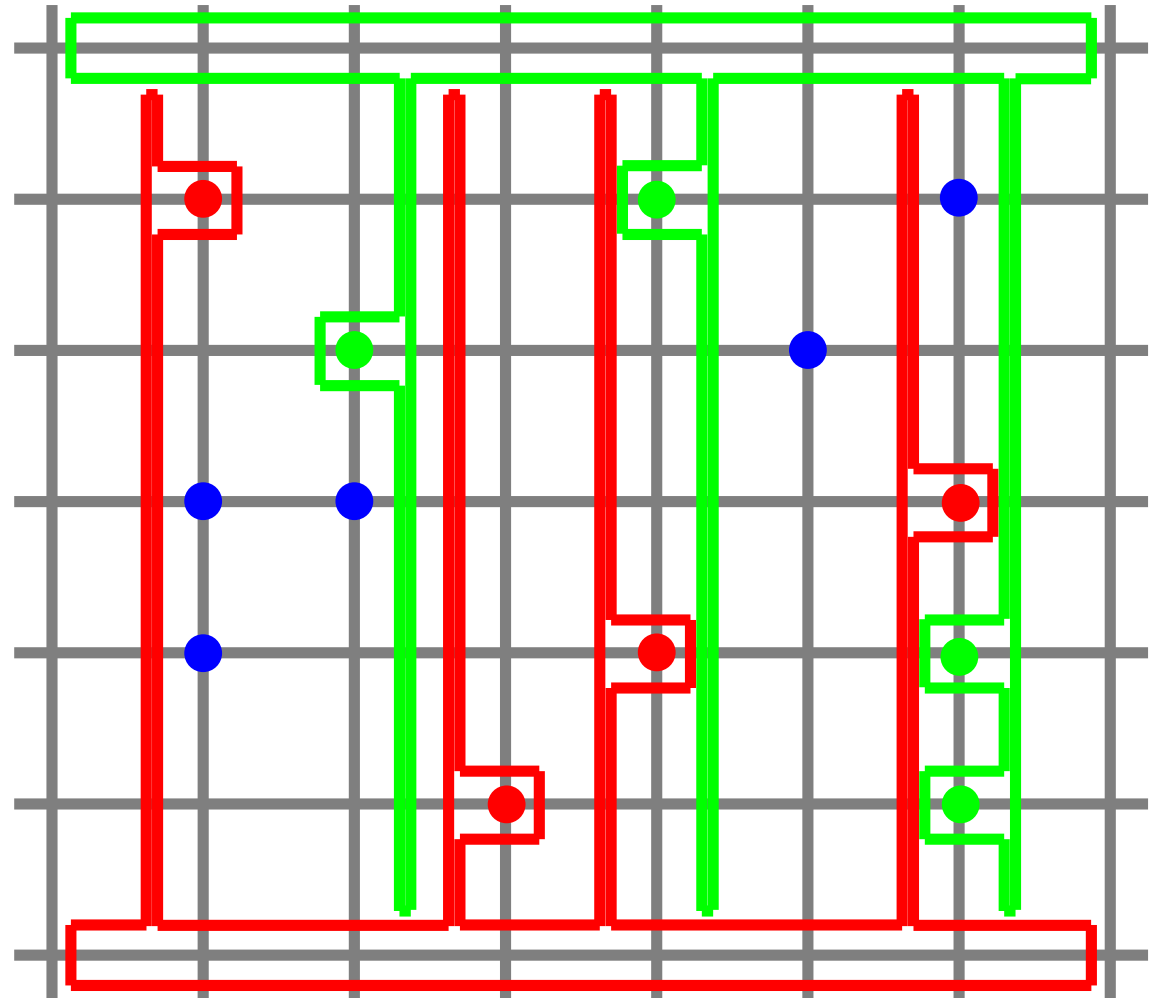
Are there too many vertices?

- Constraint: 1000 vertices per polygon
- 4 for the hub
- 4 per occupied column
- 4 per village



Are there too many vertices?

- Constraint: 1000 vertices per polygon
 - 4 for the hub
 - 4 per occupied column
 - 4 per village
-
- Fine, even with 100 villages, each in a different column:
 $4 + 400 + 400 = 804 < 1000$



Implementation

Input: 10

3 1 3

1 3 1

2 3 2

2 1 2

4 1 1

5 3 1

3 3 3

2 2 1

5 2 2

3 2 3

Implementation

- Group the villages by colour:
 $O(n)$

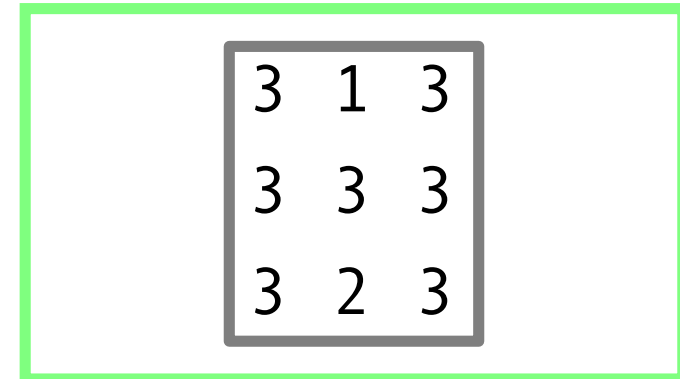
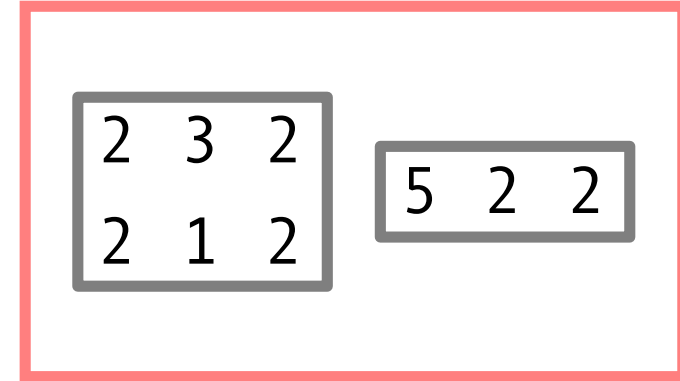
1	3	1
4	1	1
5	3	1
2	2	1

2	3	2
2	1	2
5	2	2

3	1	3
3	3	3
3	2	3

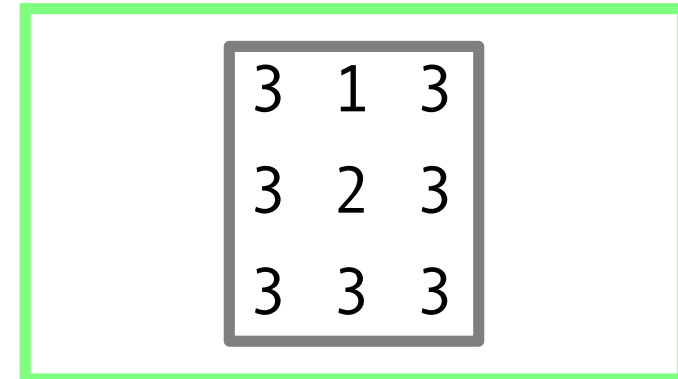
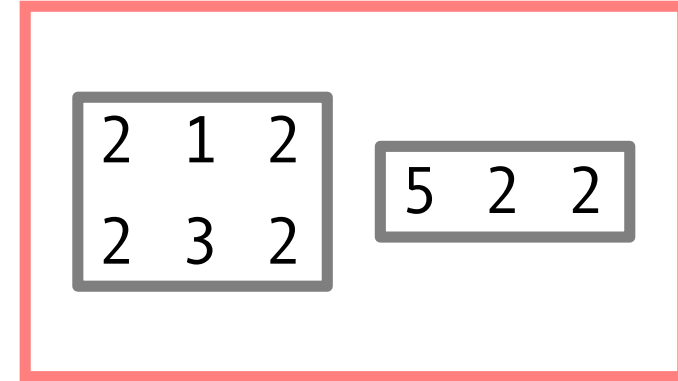
Implementation

- Group the villages by colour:
 $O(n)$
- Sort them by their x-coordinate:
 $O(n \log n)$



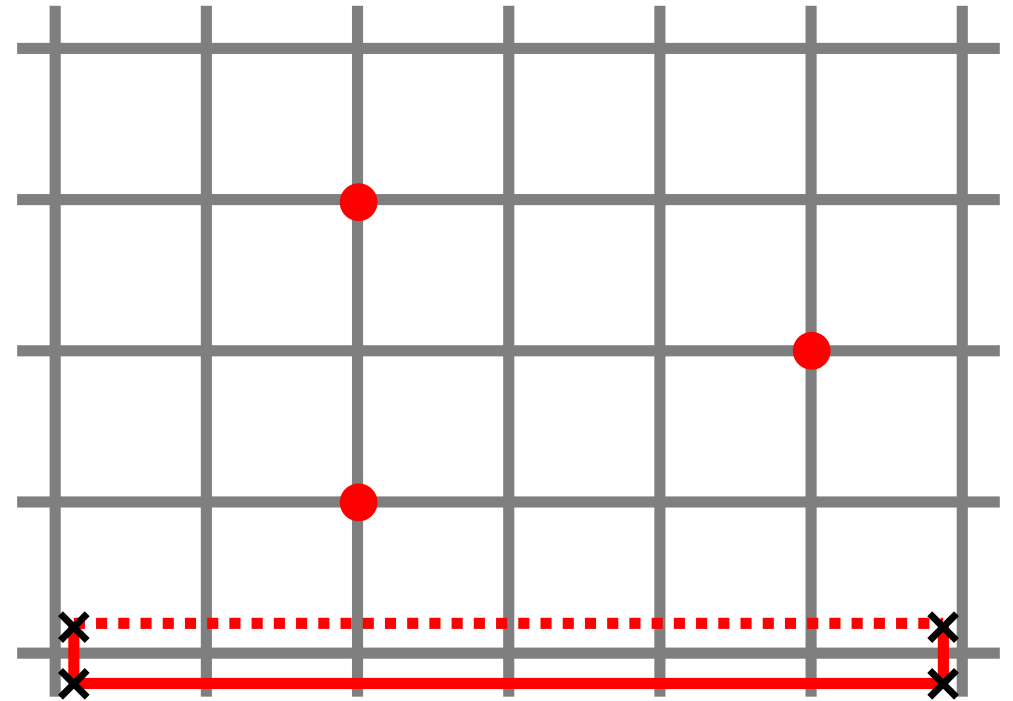
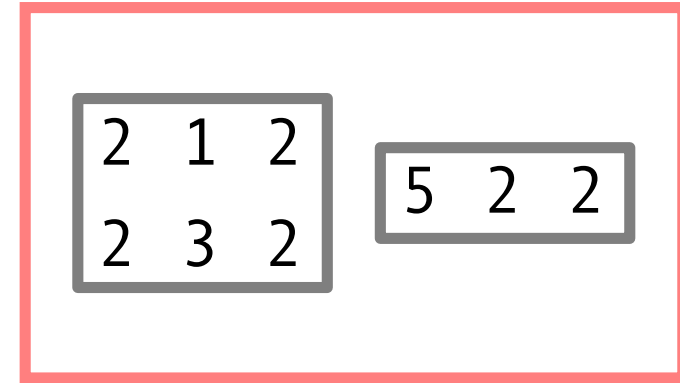
Implementation

- Group the villages by colour:
 $O(n)$
- Sort them by their x-coordinate:
 $O(n \log n)$
- Sort villages sharing colour and column by their y-coordinate:
 $O(n \log n)$ (in total)



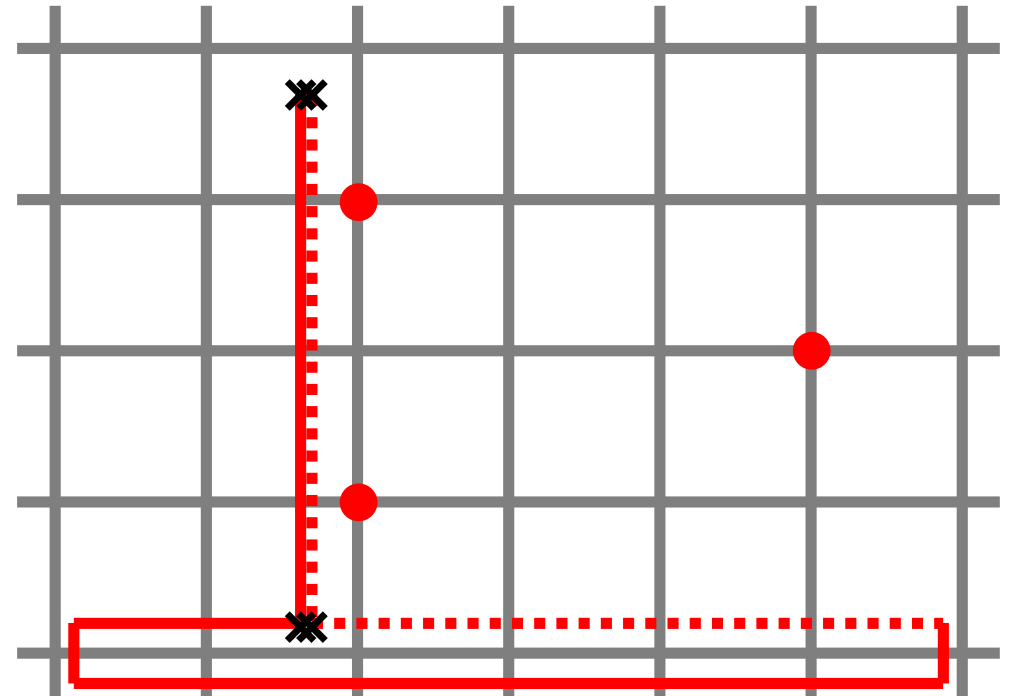
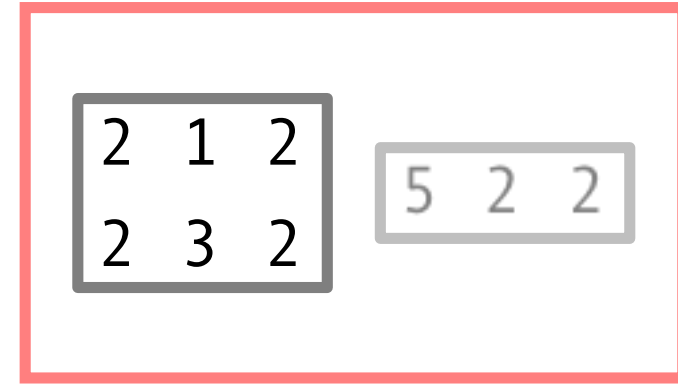
Implementation

- Add the hub vertices: $O(1)$



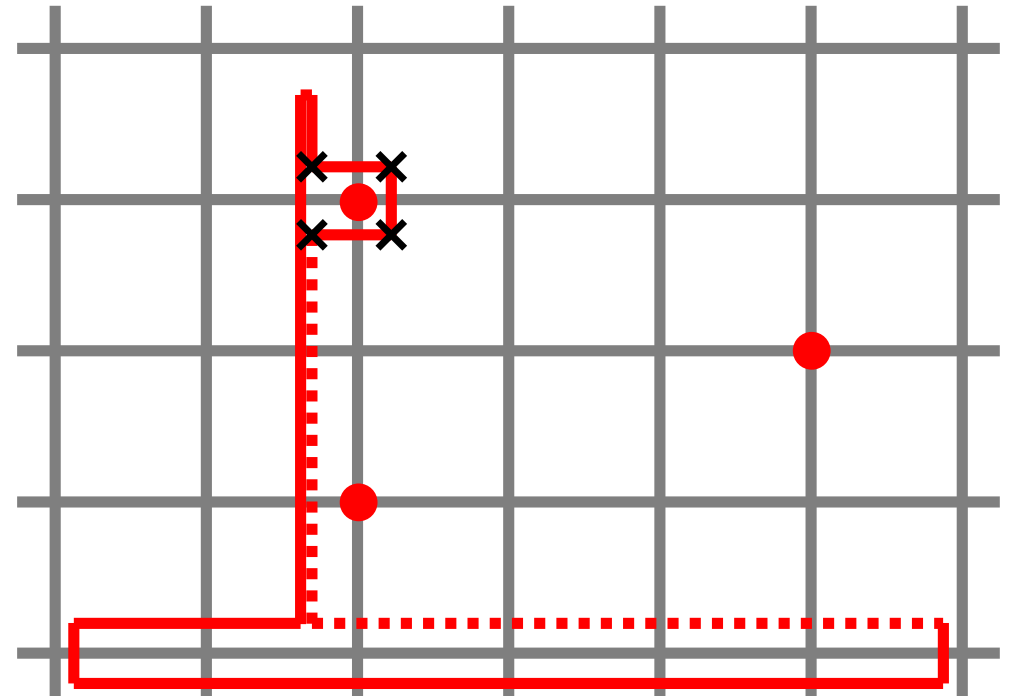
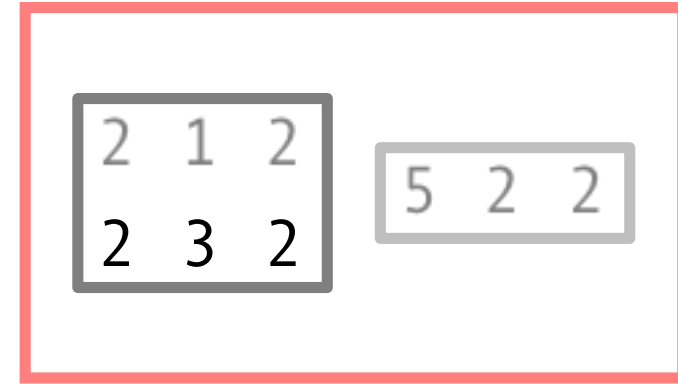
Implementation

- Add the hub vertices: $O(1)$
- Loop over the villages sorted by their x-coordinate: $O(n)$
 - Add the column's vertices: $O(1)$



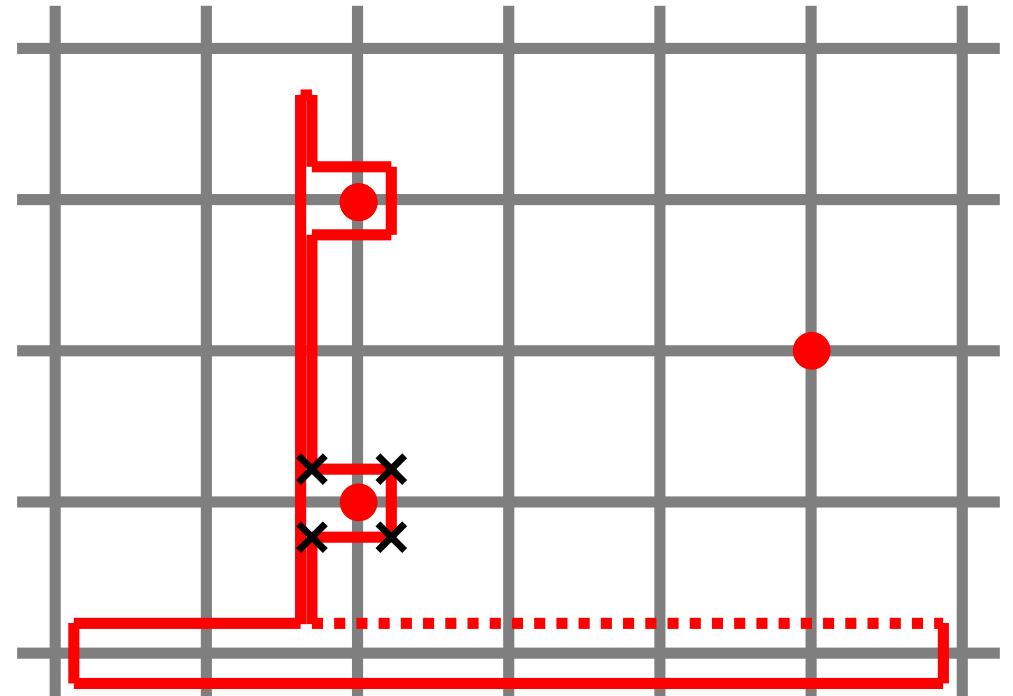
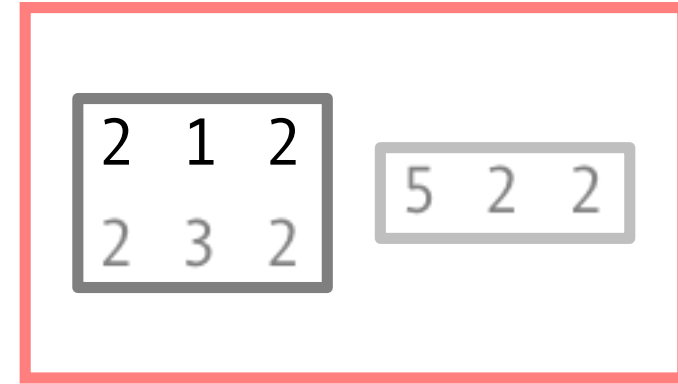
Implementation

- Add the hub vertices: $O(1)$
- Loop over the villages sorted by their x-coordinate: $O(n)$
 - Add the column's vertices: $O(1)$
 - Loop over the the villages sorted by y, to add the vertices around it: $O(n)$ (in total)



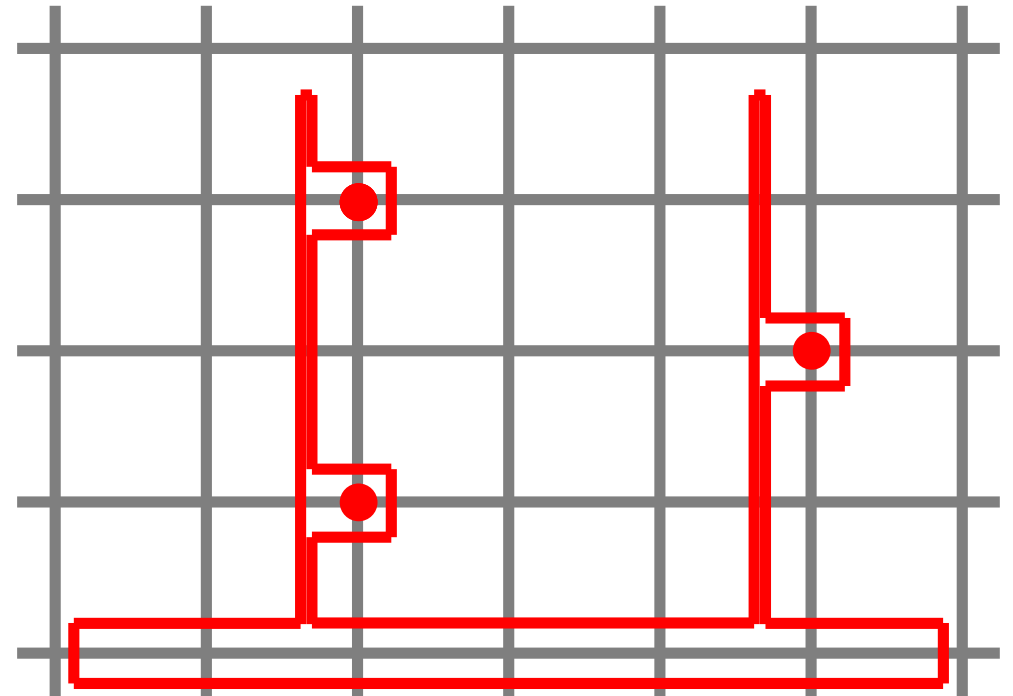
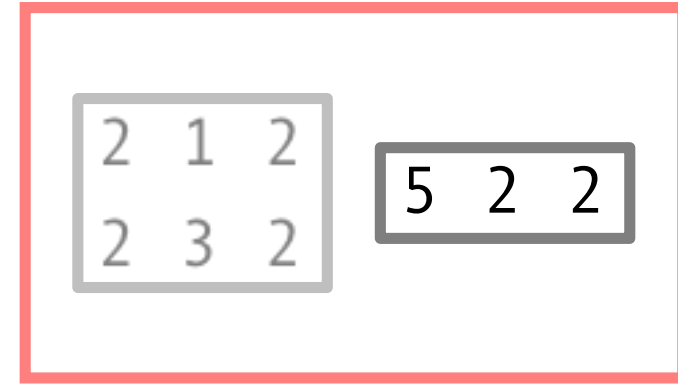
Implementation

- Add the hub vertices: $O(1)$
- Loop over the villages sorted by their x-coordinate: $O(n)$
 - Add the column's vertices: $O(1)$
 - Loop over the the villages sorted by y, to add the vertices around it: $O(n)$ (in total)



Implementation

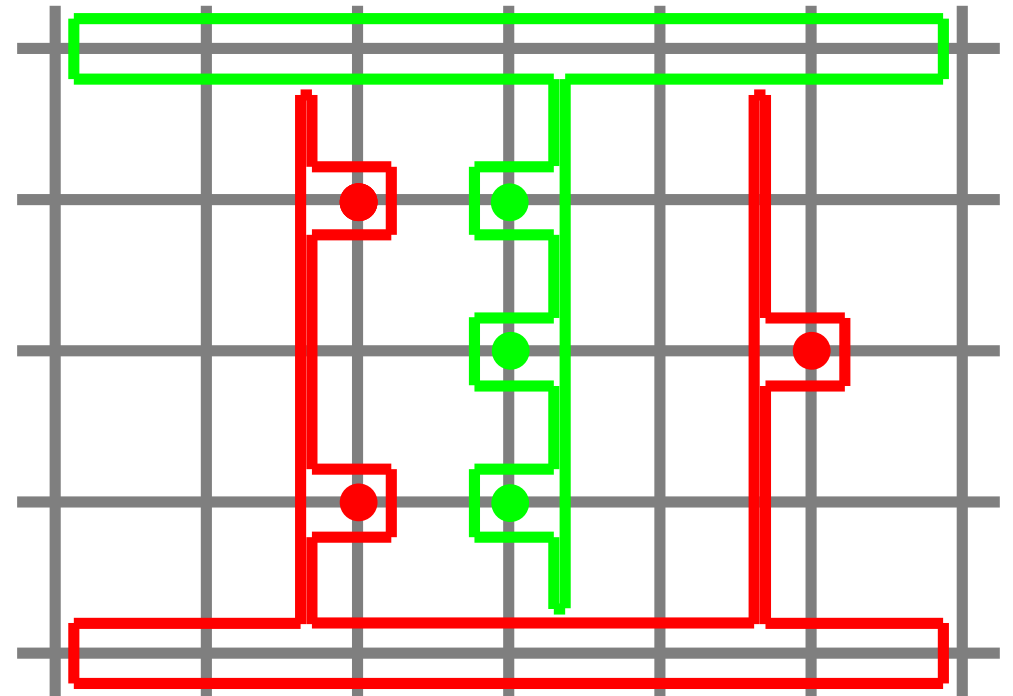
- Add the hub vertices: $O(1)$
- Loop over the villages sorted by their x-coordinate: $O(n)$
 - Add the column's vertices: $O(1)$
 - Loop over the the villages sorted by y, to add the vertices around it: $O(n)$ (in total)



Implementation

- Add the hub vertices: $O(1)$
- Loop over the villages sorted by their x-coordinate: $O(n)$
 - Add the column's vertices: $O(1)$
 - Loop over the the villages sorted by y, to add the vertices around it: $O(n)$ (in total)
- Repeat for the second polygon: $O(n)$

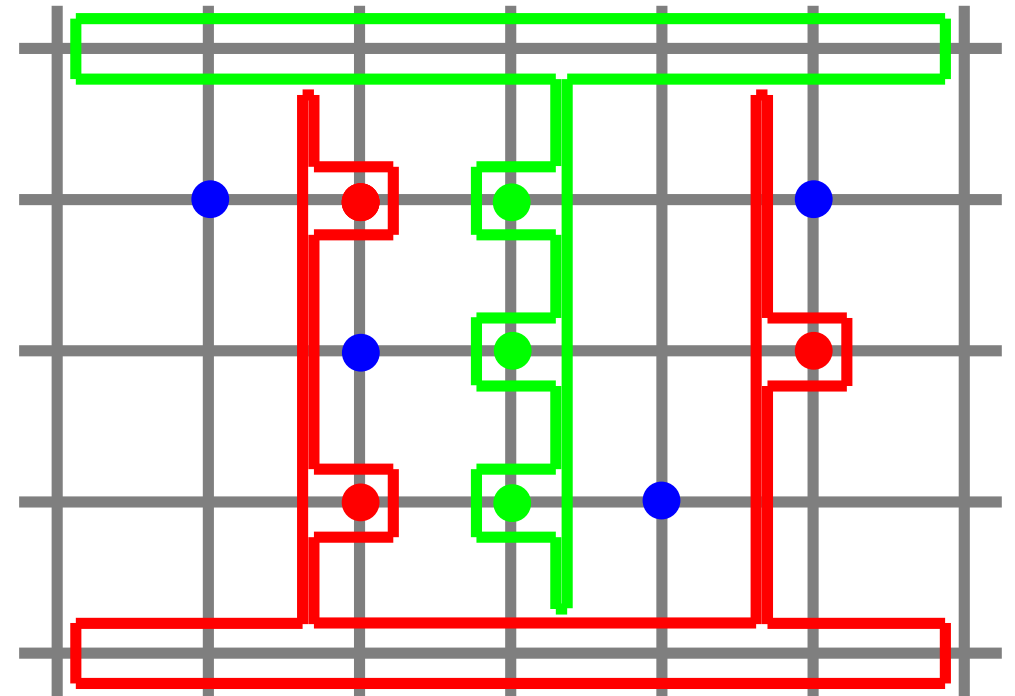
3	1	3
3	2	3
3	3	3



Implementation

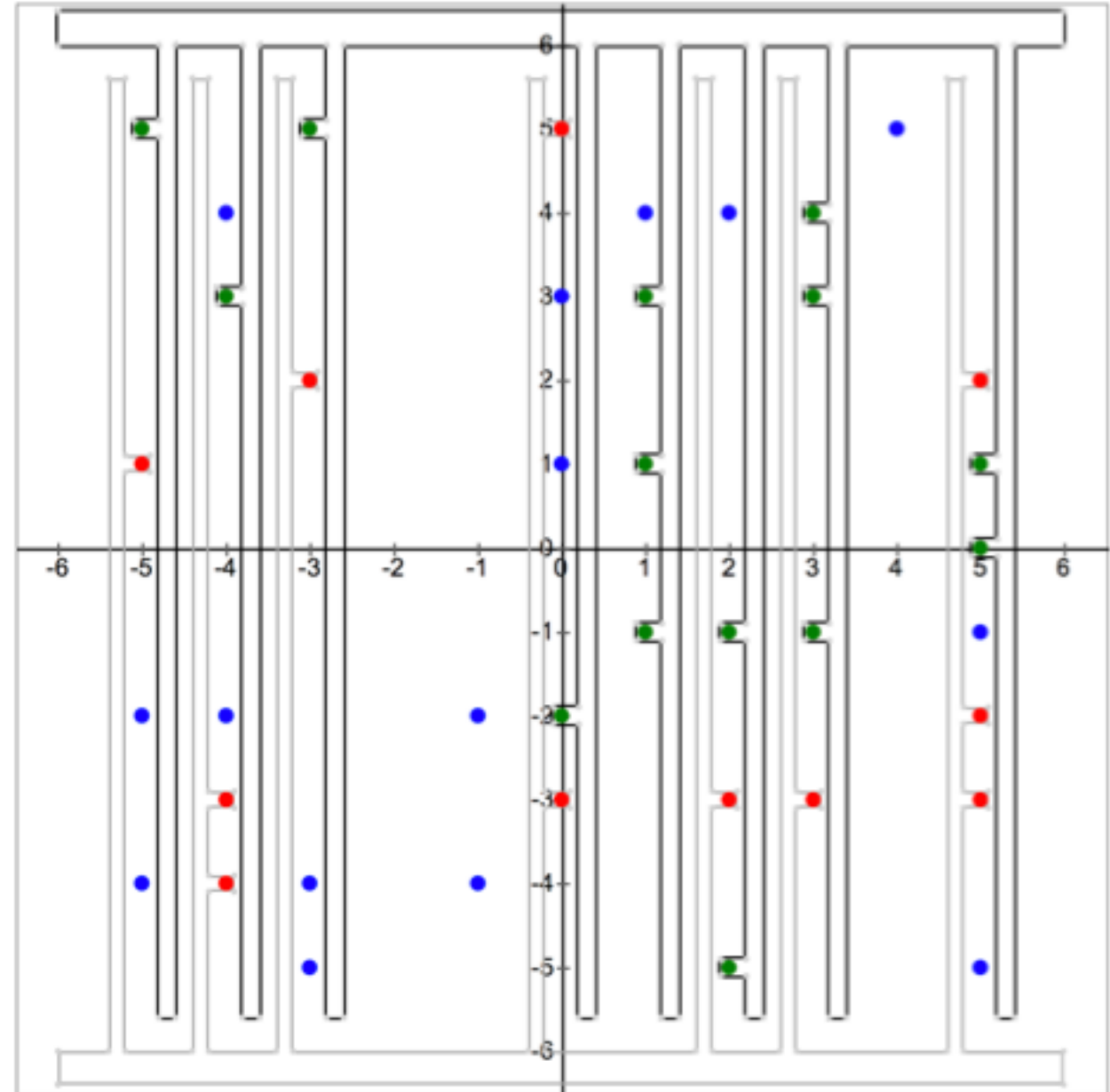
- Add the hub vertices: $O(1)$
- Loop over the villages sorted by their x-coordinate: $O(n)$
 - Add the column's vertices: $O(1)$
 - Loop over the the villages sorted by y, to add the vertices around it: $O(n)$ (in total)
- Repeat for the second polygon: $O(n)$
- Total: $O(n \log n)$ with $n \leq 49$

1	3	1
4	1	1
5	3	1
2	2	1



Visualizer

- Graphical representation of in- and outputs of the problem
- Enables the generation of new inputs for testing
- More applicable to smaller sized samples



Visualizer

- Doesn't check whether the output is within the constraints or correct
- Assumes correct format
- Allows the export of invalid inputs (e.g. too many, too few or out of bounds villages)

