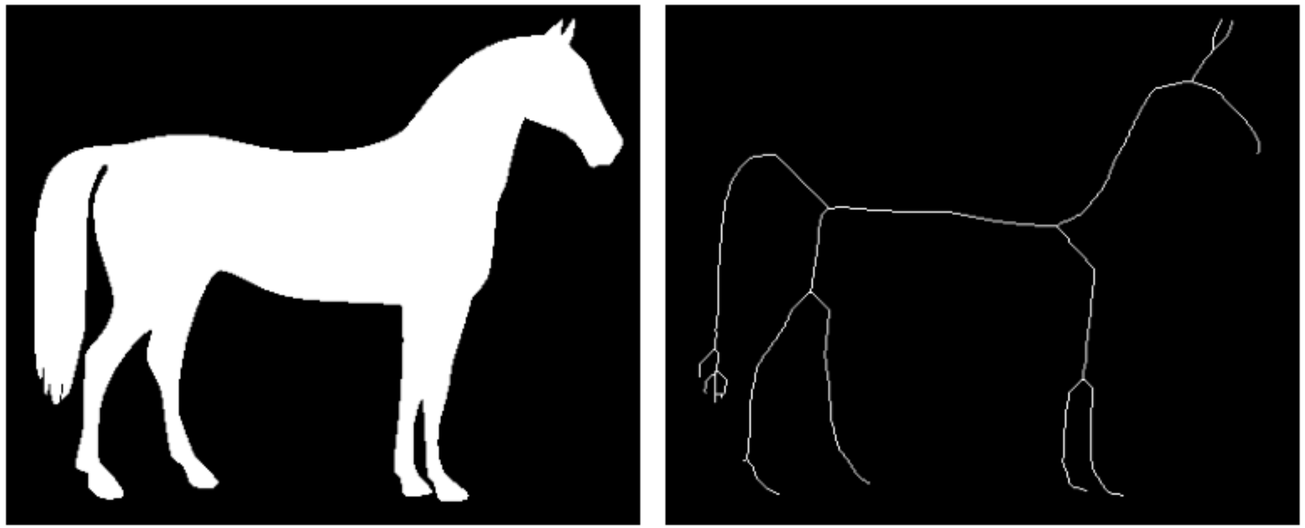
Topological Skeletons

Brooke Dai, CPSC 490 2019W2



What are topological skeletons?

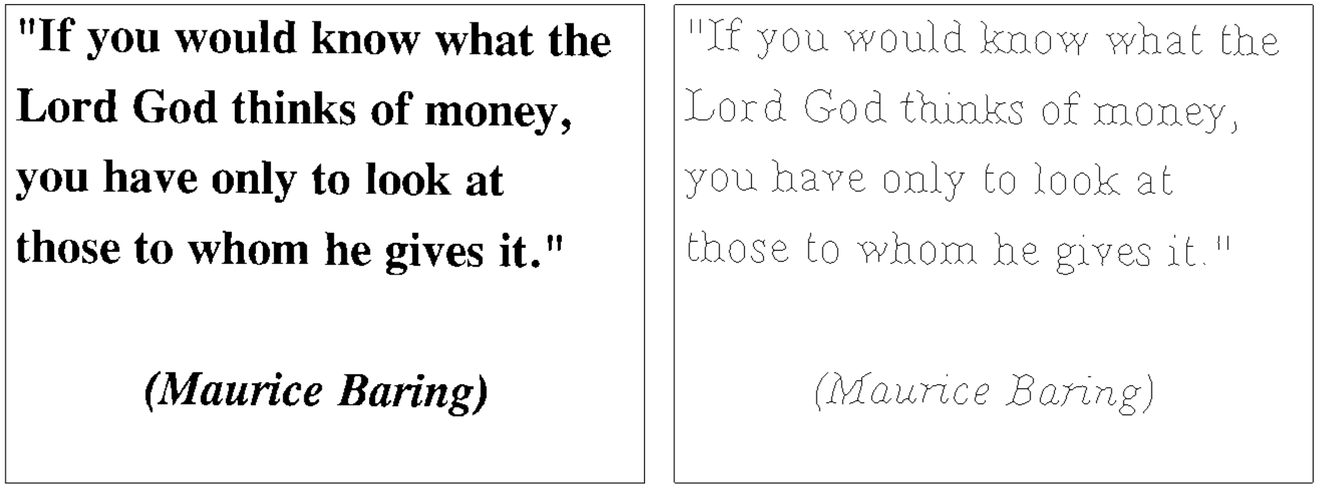
A “skeleton” of a shape which serves as a compact description of the shape.

Ideally, it has the following properties:

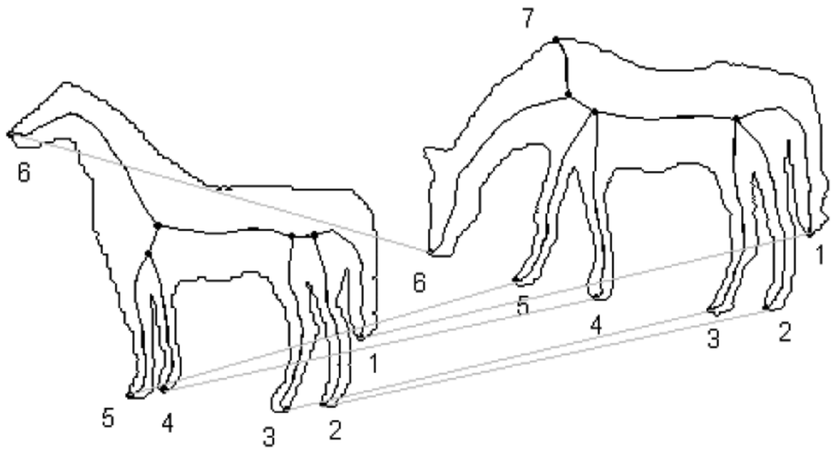
- It preserves the **topology** of the shape
- It is **thin**
- It is **centred** within the shape
- It preserves the **geometric features** of the shape
- It is **reversible**

What are they used for?

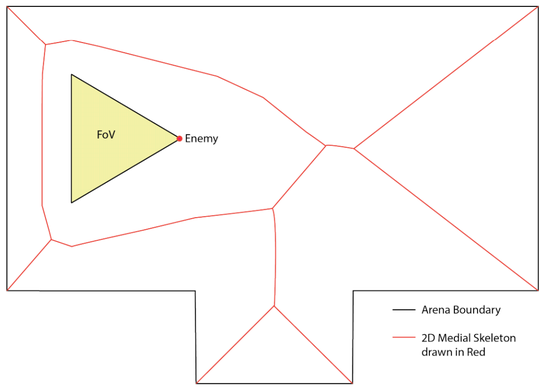
Image compression



Shape matching



Path finding (eg. in video games)



Where did they originate?

They were first introduced by Harry Blum in 1967 as a new way of describing biological shapes.

Previously, shape analysis methods were largely physics-based and could not capture some of the essential properties of the shape.

Blum came up with the concept of the **medial axis**, a compact way of describing a shape.

Where did they originate?

Grassfire propagation

Where did they originate?

Grassfire propagation

Imagine that the points inside the shape are dried grass.

Where did they originate?

Grassfire propagation

Set the boundary on fire.

Where did they originate?

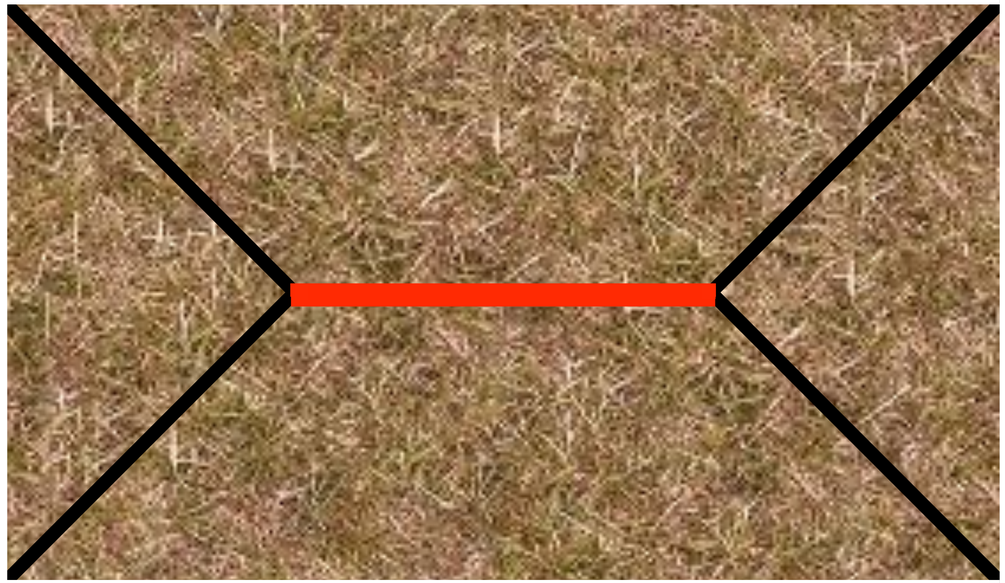
Grassfire propagation

The fire propagates inward, quenching where the fire fronts meet.

Where did they originate?

Grassfire propagation

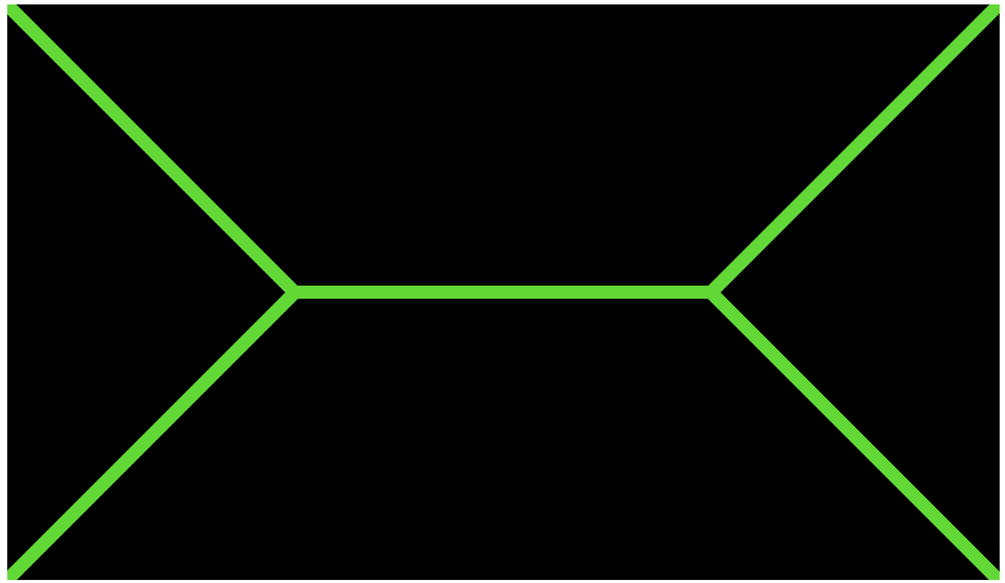
The fire propagates inward, quenching where the fire fronts meet.



Where did they originate?

Grassfire propagation

The medial axis will appear along the lines that the fire quenches itself.

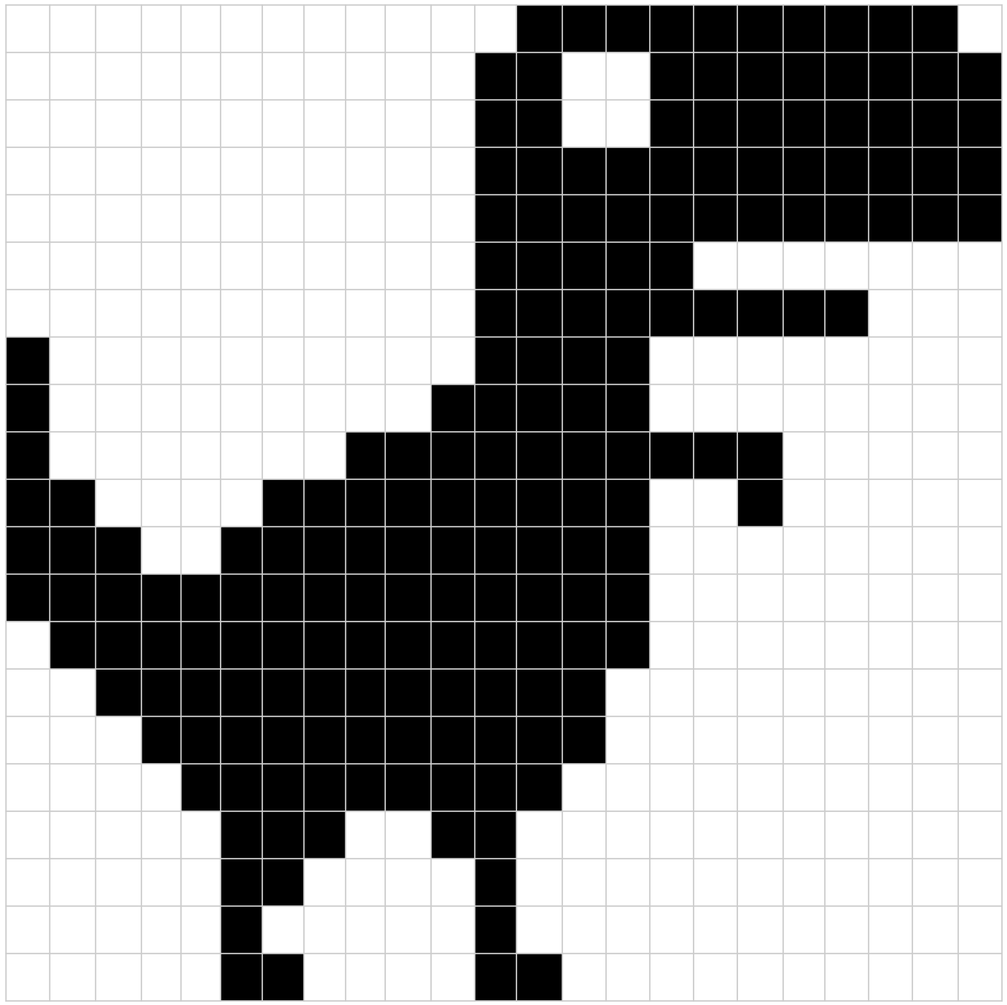


Skeletonization Algorithm

This algorithm is based off Chang's distance transform skeletonization algorithm (2007).

There are three main steps:

1. Apply the distance transform
2. Identify a set of ridge point candidates
3. Build the skeleton from the ridge point candidates



Skeletonization Algorithm

1. Apply the distance transform

Start with the binary image of the shape.

0 - background

1 - shape

The distance transform algorithm uses Manhattan distance and takes two passes.

The first pass processes each pixel from top->bottom and left->right.

The second pass processes each pixel from bottom->top and right->left.

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

Skeletonization Algorithm

1. Apply the distance transform

First pass

```
let D(x, y) be the distance value of (x, y)
for each row y in image from top to bottom:
  for each column x in image from left to right:
    if (x, y) is in shape:
      set D(x, y) to min(D(x, y - 1) + 1,
                         D(x - 1, y) + 1)
    else:
      set D(x, y) to 0
```

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	1	2	2	2	2	2	2	1
0	0	0	0	0	0	0	0	0	0	0	1	2	0	0	1	2	3	3	3	3	3	2
0	0	0	0	0	0	0	0	0	0	0	1	2	1	1	2	3	4	4	4	4	4	3
0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	3	4	5	5	5	5	5	4
0	0	0	0	0	0	0	0	0	0	0	1	2	3	3	4	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	1	1	1	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	2	3	4	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	2	3	4	5	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1	2	3	4	5	6	1	1	1	0	0	0	0	0
1	1	0	0	0	0	1	1	2	2	3	4	5	6	7	0	0	1	0	0	0	0	0
1	2	1	0	0	1	2	2	3	3	4	5	6	7	8	0	0	0	0	0	0	0	0
1	2	2	1	1	2	3	3	4	4	5	6	7	8	9	0	0	0	0	0	0	0	0
0	1	2	2	2	3	4	4	5	5	6	7	8	9	10	0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	5	6	6	7	8	9	10	0	0	0	0	0	0	0	0	0
0	0	0	1	2	3	4	5	6	7	8	9	10	11	0	0	0	0	0	0	0	0	0
0	0	0	0	1	2	3	4	5	6	7	8	9	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	2	3	0	0	1	2	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	2	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

Skeletonization Algorithm

1. Apply the distance transform

Second pass

```
let D(x, y) be the distance value of (x, y)
for each row y in image from bottom to top:
  for each column x in image from right to left:
    if (x, y) is in shape:
      set D(x, y) to min(D(x, y),
                         D(x, y - 1) + 1,
                         D(x - 1, y) + 1)
    else:
      set D(x, y) to 0
```

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	2	2	2	2	2	2	1
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	2	3	3	3	3	2	1
0	0	0	0	0	0	0	0	0	0	0	1	2	1	1	2	2	2	2	2	2	2	1
0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	2	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	1	2	3	2	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	2	3	2	1	1	1	1	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	2	2	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	2	3	2	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1	2	3	4	3	2	1	1	1	0	0	0	0	0
1	1	0	0	0	0	1	1	2	2	3	4	3	2	1	0	0	1	0	0	0	0	0
1	2	1	0	0	1	2	2	3	3	4	4	3	2	1	0	0	0	0	0	0	0	0
1	2	2	1	1	2	3	3	4	4	5	4	3	2	1	0	0	0	0	0	0	0	0
0	1	2	2	2	3	4	4	4	4	5	4	3	2	1	0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	4	3	3	4	3	2	1	0	0	0	0	0	0	0	0	0
0	0	0	1	2	3	4	3	2	2	3	3	2	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	2	3	2	1	1	2	2	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	2	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

Skeletonization Algorithm

2. Identify a set of ridge point candidates

Finding exact ridge points may be impossible.

Instead, we first find a set of *ridge point candidates,* then choose from that set.

First, we find the distance difference between a point and its neighbour using scan lines.

Second, we label each point based on how strong of a ridge indicator the point is.

0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	2	2	2	2	2	2	1
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	2	3	3	3	3	2	1
0	0	0	0	0	0	0	0	0	0	0	1	2	1	1	2	2	2	2	2	2	2	1
0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	2	1	1	1	1	1	1	1
0	0	0	0	0	0	0	0	0	0	0	1	2	3	2	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	1	2	3	2	1	1	1	1	1	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	2	2	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	1	2	3	2	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	1	2	3	4	3	2	1	1	1	0	0	0	0	0
1	1	0	0	0	0	1	1	2	2	3	4	3	2	1	0	0	1	0	0	0	0	0
1	2	1	0	0	1	2	2	3	3	4	4	3	2	1	0	0	0	0	0	0	0	0
1	2	2	1	1	2	3	3	4	4	5	4	3	2	1	0	0	0	0	0	0	0	0
0	1	2	2	2	3	4	4	4	4	5	4	3	2	1	0	0	0	0	0	0	0	0
0	0	1	2	3	4	5	4	3	3	4	3	2	1	0	0	0	0	0	0	0	0	0
0	0	0	1	2	3	4	3	2	2	3	3	2	1	0	0	0	0	0	0	0	0	0
0	0	0	0	1	2	3	2	1	1	2	2	1	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	2	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0

Skeletonization Algorithm

2. Identify a set of ridge point candidates

Scan map (x direction)

For each row of points, we scan from left to right and fill in our scan map in the x direction.

```
for each row y in image from top to bottom:
  for each column x in image from left to right:
    set scanX(x, y) to D(x + 1, y) - D(x, y)
    set scanY(x, y) to D(x, y + 1) - D(x, y)
```

0	0	0	0	0	0	0	0	0	0	0	0	0	+	0	0	0	0	0	0	0	0	0	-	0
0	0	0	0	0	0	0	0	0	0	0	0	+	0	-	0	+	+	0	0	0	0	0	-	-
0	0	0	0	0	0	0	0	0	0	0	0	+	0	-	0	+	+	+	0	0	0	-	-	-
0	0	0	0	0	0	0	0	0	0	0	0	+	+	-	0	+	0	0	0	0	0	0	-	-
0	0	0	0	0	0	0	0	0	0	0	0	+	+	0	0	0	-	0	0	0	0	0	0	-
0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	-	-	-	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	-	-	0	0	0	0	-	0	0	0
0	+	-	0	0	0	0	0	0	0	0	0	+	+	0	-	-	0	0	0	0	0	0	0	0
0	+	-	0	0	0	0	0	0	0	+	+	+	-	-	-	0	0	0	0	0	0	0	0	0
0	+	-	0	0	0	0	0	0	+	0	+	+	+	-	-	-	0	0	-	0	0	0	0	0
0	+	0	-	0	0	0	+	0	+	0	+	+	-	-	-	-	0	+	-	0	0	0	0	0
0	+	+	-	-	0	+	+	0	+	0	+	0	-	-	-	-	0	0	0	0	0	0	0	0
0	+	+	0	-	0	+	+	0	+	0	+	-	-	-	-	-	0	0	0	0	0	0	0	0
0	0	+	+	0	0	+	+	0	0	0	+	-	-	-	-	-	0	0	0	0	0	0	0	0
0	0	0	+	+	+	+	+	-	-	0	+	-	-	-	-	0	0	0	0	0	0	0	0	0
0	0	0	0	+	+	+	+	-	-	0	+	0	-	-	-	0	0	0	0	0	0	0	0	0
0	0	0	0	0	+	+	+	-	-	0	+	0	-	-	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	+	+	-	-	0	+	0	-	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	+	0	-	0	0	0	+	-	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	+	-	0	0	0	0	+	-	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	+	0	-	0	0	0	+	0	-	0	0	0	0	0	0	0	0	0	0

Skeletonization Algorithm

2. Identify a set of ridge point candidates

Scan map (y direction)

For each row of points, we scan from top to bottom and fill in our scan map in the y direction.

```
for each row y in image from top to bottom:
  for each column x in image from left to right:
    set scanX(x, y) to D(x + 1, y) - D(x, y)
    set scanY(x, y) to D(x, y + 1) - D(x, y)
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	+	+	+	+	+	+	0
0	0	0	0	0	0	0	0	0	0	0	+	0	-	-	0	+	+	+	+	+	+	+
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	0	0
0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	0	-	-	-	-	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	+	+	0	-	-	-	-	-	-	0
0	0	0	0	0	0	0	0	0	0	0	0	0	+	0	-	-	-	-	-	-	-	-
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	0	0	0
+	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	0	0	0
0	0	0	0	0	0	0	0	0	0	+	+	+	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	+	+	+	+	+	+	+	+	+	1	0	0	0	0	0
0	+	0	0	0	0	+	+	+	+	+	+	-	-	-	-	-	1	0	0	0	0	0
0	+	+	0	0	+	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0
0	0	+	+	+	+	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0
-	-	0	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	-	-	0	+	+	+	0	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0
0	0	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0
0	0	0	0	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	-	-	0	0	-	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	+	0	0	0	0	0	+	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	-	-	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0

Skeletonization Algorithm

2. Identify a set of ridge point candidates

Label possible ridge points

We look in both x and y directions for *prominent sign barriers*, patterns of transitions that suggest a ridge's existence, and label accordingly.

Again, x direction is scanned left to right and y direction is scanned top to bottom.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	+	+	+	+	+	+	0
0	0	0	0	0	0	0	0	0	0	0	+	0	-	-	0	+	+	+	+	+	+	+
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	0	0
0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	0	-	-	-	-	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	+	+	0	-	-	-	-	-	-	0
0	0	0	0	0	0	0	0	0	0	0	0	0	+	0	-	-	-	-	-	-	-	-
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	0	0	0
+	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	0	0	0
0	0	0	0	0	0	0	0	0	0	+	+	+	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	+	+	+	+	+	+	+	+	+	1	0	0	0	0	0
0	+	0	0	0	0	+	+	+	+	+	+	-	-	-	-	-	1	0	0	0	0	0
0	+	+	0	0	+	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0
0	0	+	+	+	+	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0
-	-	0	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	-	-	0	+	+	+	0	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0
0	0	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0
0	0	0	0	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	-	-	0	0	-	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	+	0	0	0	0	0	+	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	-	-	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0

Skeletonization Algorithm

2. Identify a set of ridge point candidates

Label possible ridge points

The prominent sign barriers are:

+ | -	(STRONG indication of ridge existence)
+ | **0** | -	(STRONG)
+ | 0	(WEAK)
0 | -	(WEAK)

WEAK can be promoted to GOOD when paired with WEAK from the other direction.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	+	+	+	+	+	+	0
0	0	0	0	0	0	0	0	0	0	0	+	0	-	-	0	+	+	+	+	+	+	+
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	0	0
0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	0	-	-	-	-	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	+	+	0	-	-	-	-	-	-	0
0	0	0	0	0	0	0	0	0	0	0	0	0	+	0	-	-	-	-	-	-	-	-
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	+	+	+	+	0	0	0
+	0	0	0	0	0	0	0	0	0	0	0	0	-	-	-	-	-	-	-	0	0	0
0	0	0	0	0	0	0	0	0	0	+	+	+	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	+	+	+	+	+	+	+	+	+	1	0	0	0	0	0
0	+	0	0	0	0	+	+	+	+	+	+	-	-	-	-	-	1	0	0	0	0	0
0	+	+	0	0	+	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0
0	0	+	+	+	+	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0
-	-	0	+	+	+	+	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	-	-	0	+	+	+	0	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0
0	0	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	-	-	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0
0	0	0	0	-	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	-	-	0	0	-	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	-	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	+	0	0	0	0	0	+	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	-	-	0	0	0	0	-	-	0	0	0	0	0	0	0	0	0	0

Skeletonization Algorithm

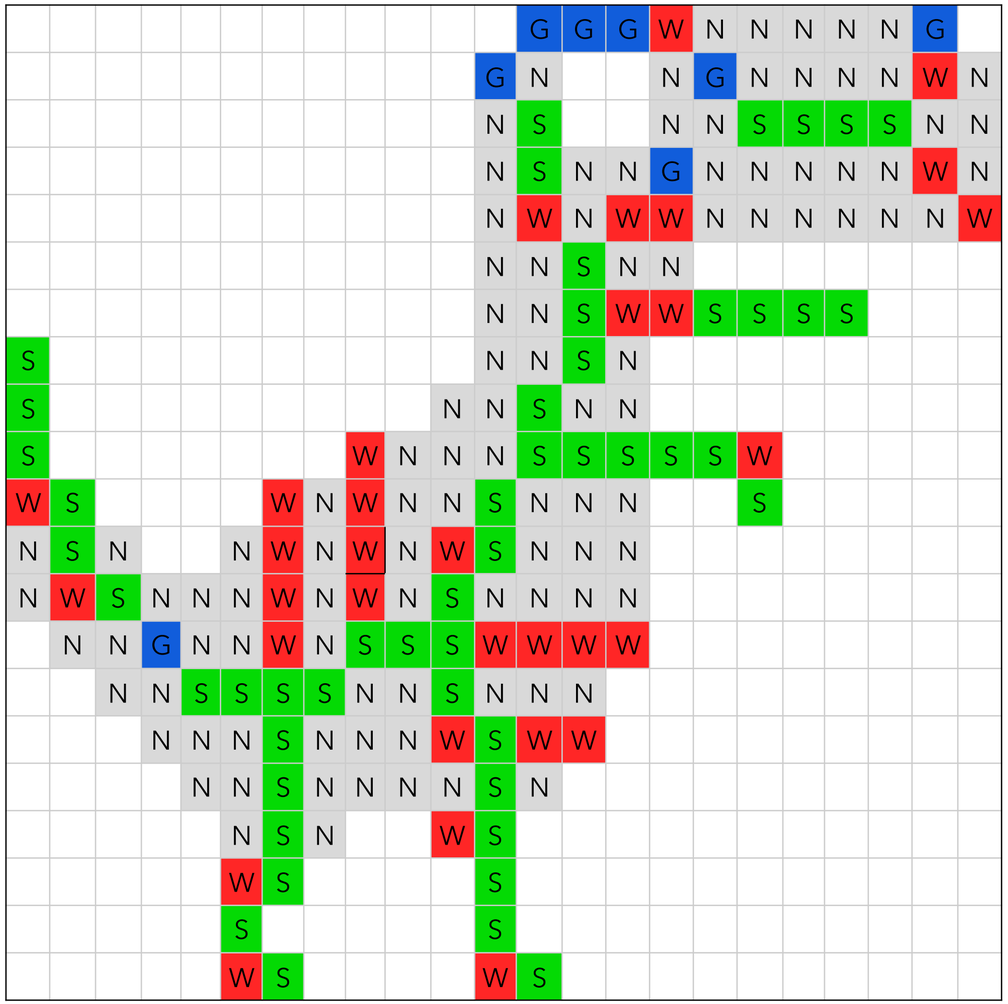
2. Identify a set of ridge point candidates

Label possible ridge points

The prominent sign barriers are:

+ | -	(STRONG indication of ridge existence)
+ | **0** | -	(STRONG)
+ | 0	(WEAK)
0 | -	(WEAK)

WEAK can be promoted to GOOD when paired with WEAK from the other direction.



Skeletonization Algorithm

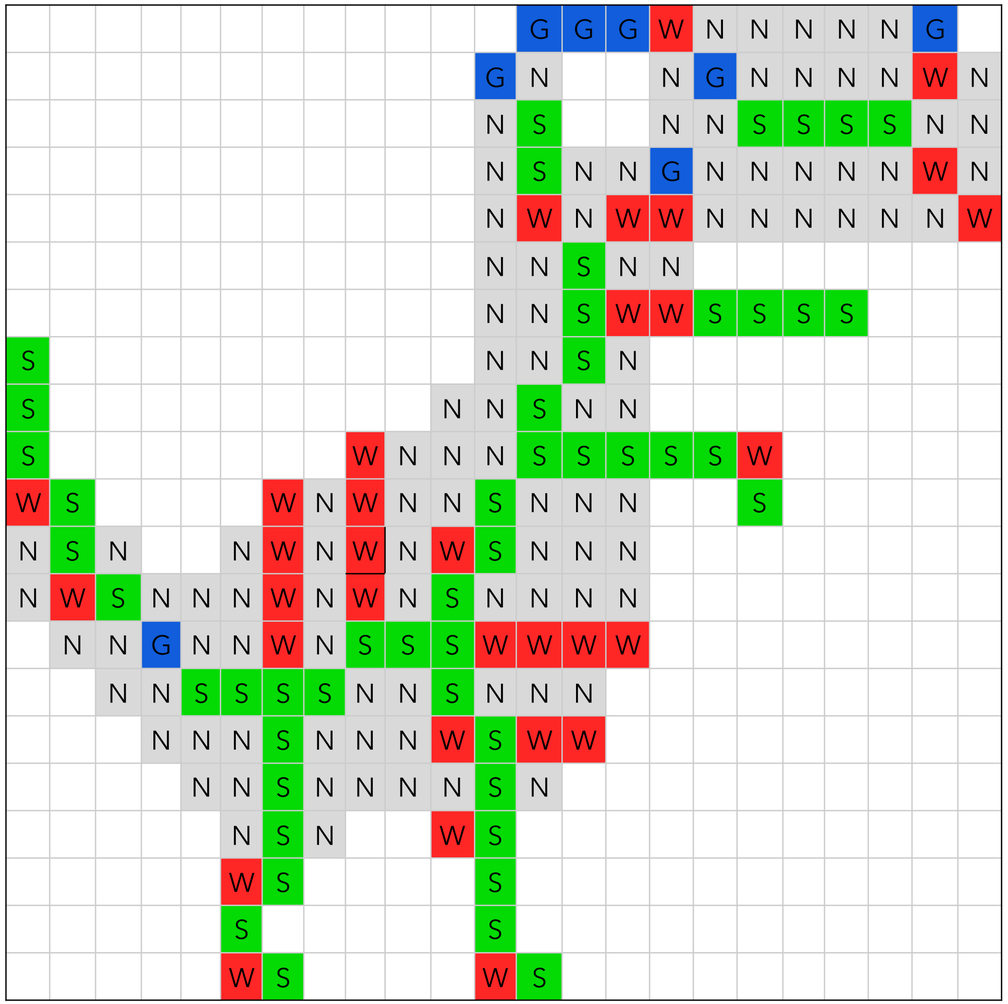
2. Identify a set of ridge point candidates

Label possible ridge points

```
for each row y in image from top to bottom:
    for each column x in image from left to right:
        let prevX be scanX(x - 1, y)
        let currX be scanX(x, y)
        let nextX be scanX(x + 1, y)
        let prevY be scanY(x, y - 1)
        let currY be scanY(x, y)
        let nextY be scanY(x, y + 1)

        // For the x direction, in the case of +-, +0, 0-,
        // prevX is the first sign and currX is the second.
        // in the case of +0-, prevX is the first sign,
       // currX is the second, and nextX is the third.
       // same applies to Y for its respective variables.
```

```
if +- or +0- in the x direction:
    set label of (x, y) to STRONG
if +- or +0- in the y direction:
    set label of (x, y) to STRONG
if current point is not labelled and
   +0 or 0- in the x direction and
   +0 or 0- in the y direction:
    set label of (x, y) to GOOD
if current point is not labelled and
   +0 or 0- in the x direction, or
   +0 or 0- in the y direction:
    set label of (x, y) to WEAK
if current point is not labelled:
    set label of (x, y) to NONE
```



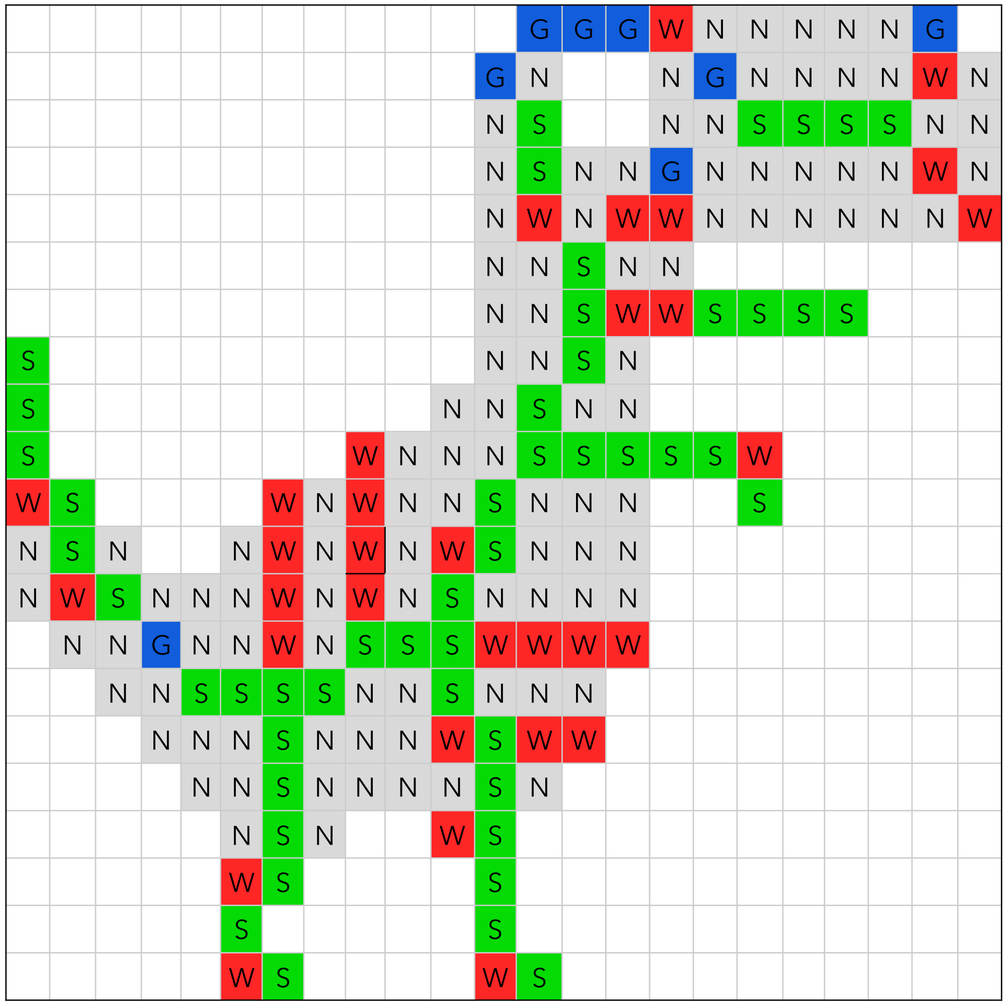
Skeletonization Algorithm

3. Build the skeleton

This step takes two passes.

The first pass adds all S and G points into the skeleton set.

The second pass joins the disconnected parts of the skeleton with W and N points if needed.



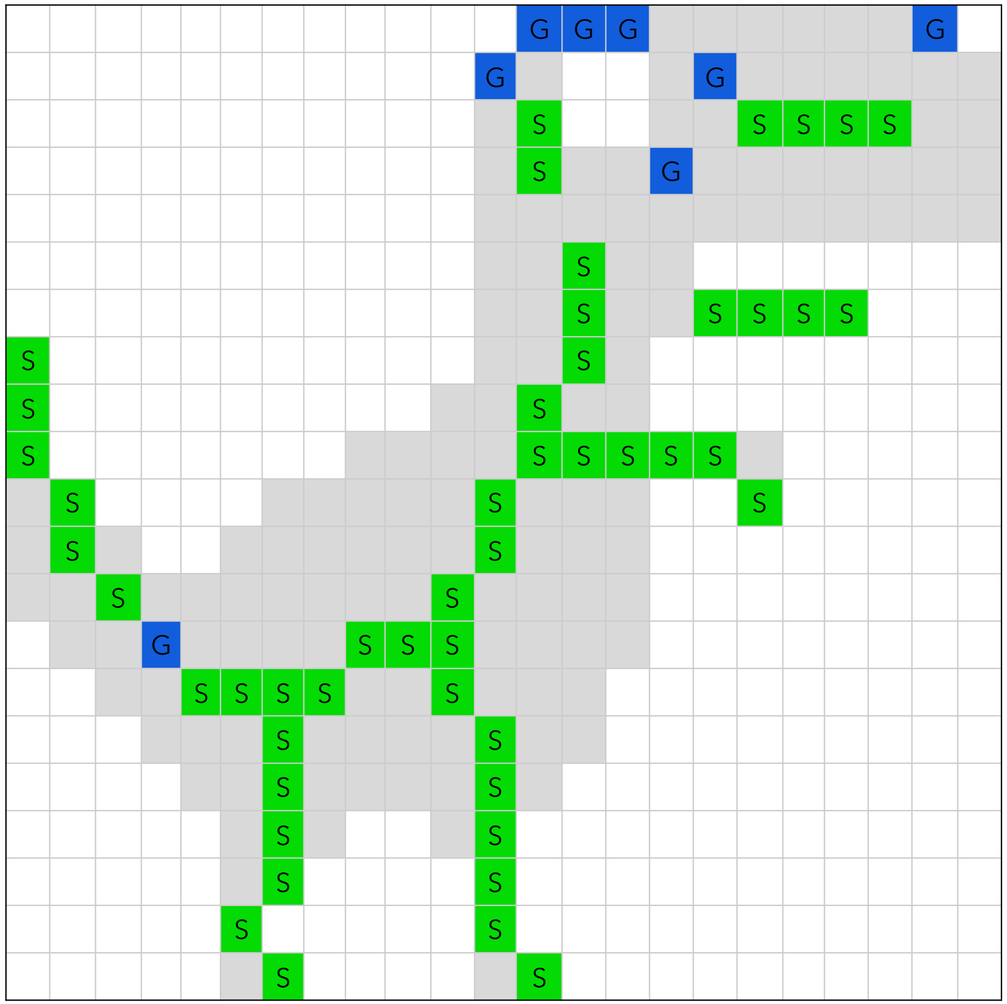
Skeletonization Algorithm

3. Build the skeleton

First pass

The first pass adds all S and G points into the skeleton set.

```
let S be the set of skeleton points
let visited be array of points already
considered for S
for each row y from top to bottom:
  for each column x from left to right:
    if (x, y) is labelled STRONG or GOOD:
      add (x, y) to S
```



Skeletonization Algorithm

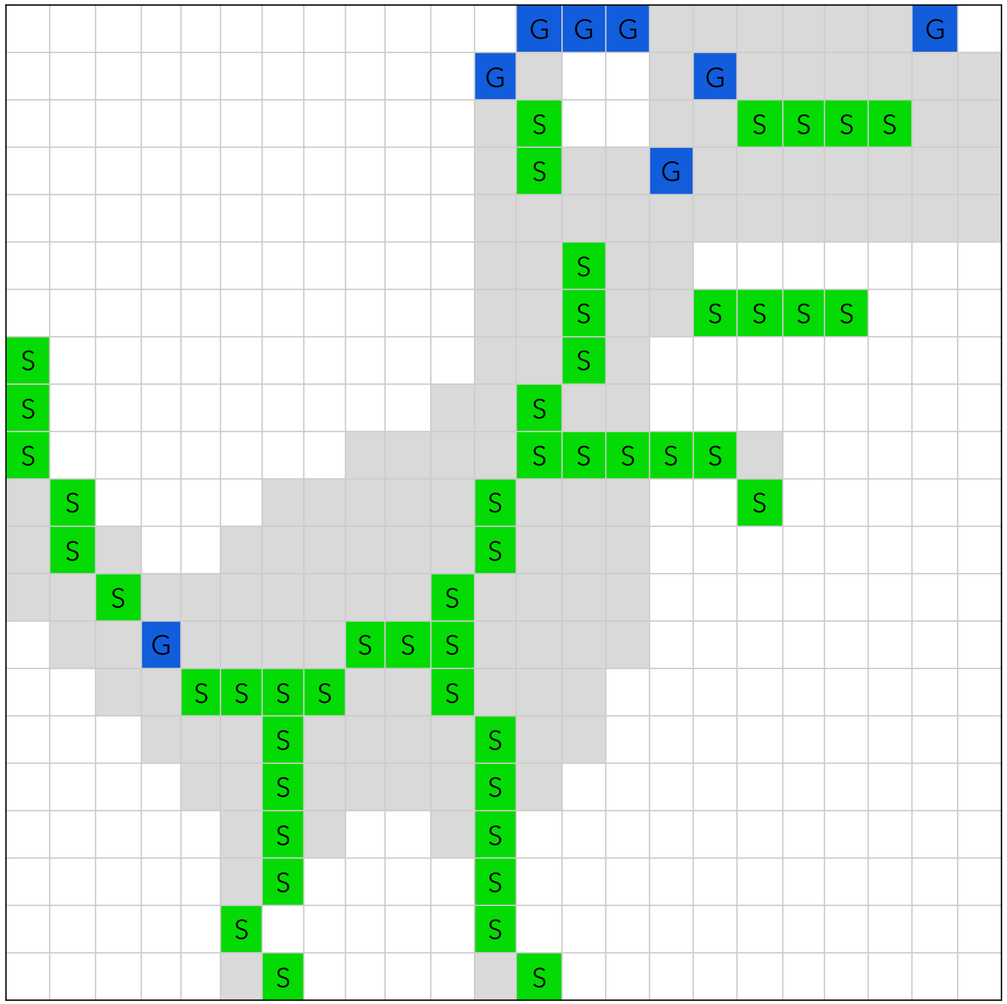
3. Build the skeleton

Second pass

The second pass joins the disconnected parts of the skeleton with W and N points if needed.

For each endpoint on the skeleton (< 2 neighbours), extend a tentative branch until a base case is reached:

- the branch collides with the shape boundary (discard)
- the tentative branch reaches a point in the skeleton set (keep)

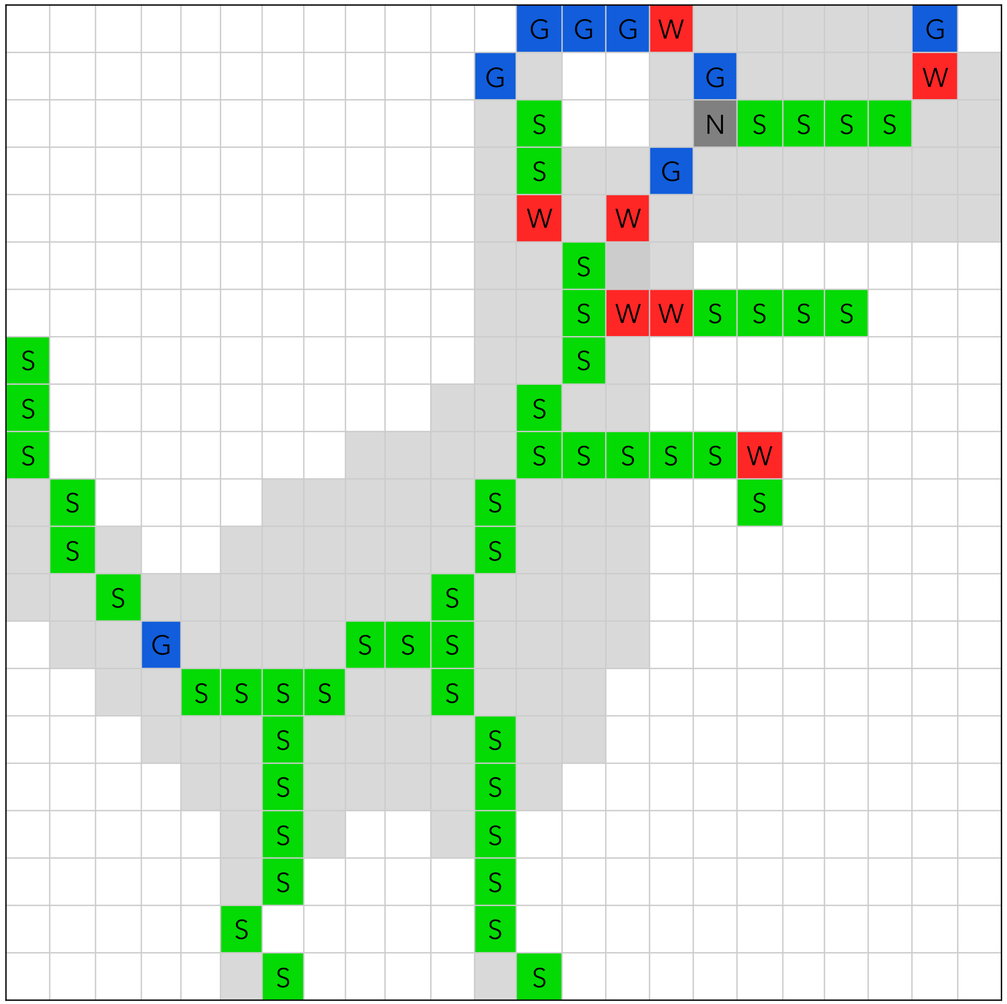


Skeletonization Algorithm

3. Build the skeleton

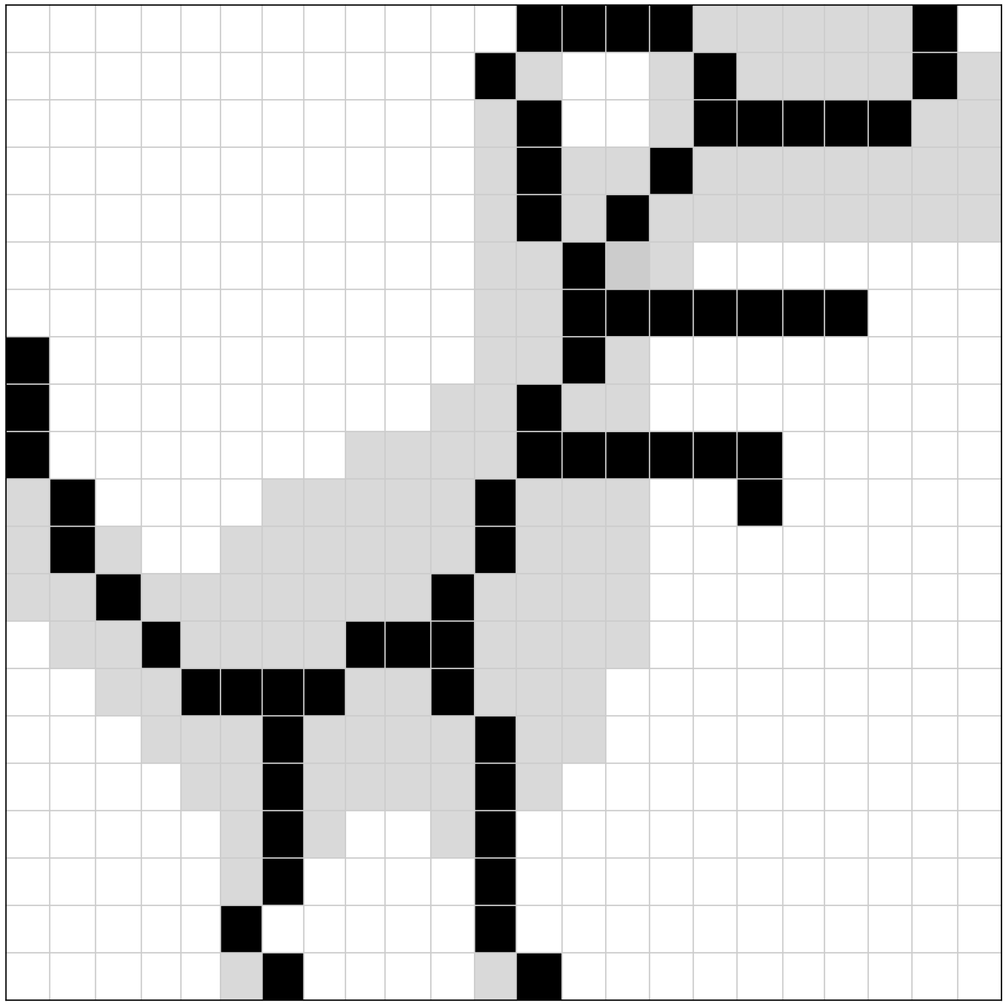
Second pass

```
let S be the set of skeleton points
let visited be array of points already considered for S
for each row y from top to bottom:
      for each row x from left to right:
            if (x, y) is in S:
                  let tent be array of points in tentative branch
                  if (x, y) has less than 2 neighbours:
                        let currX be current x-coordinate of tent branch
                              let currY be current y-coordinate of tent branch
                              add (currX, currY) to tent
                              while neither of the two cases are satisfied:
                                    check the neighbours of (currX, currY) in
                                    the general direction away from existing
                                    neighbour or closest boundary point.
                                    if there exists a neighbouring point that is
                                    in the set of skeleton points:
                                      break
                                    if there are no valid neighbours:
                                      set currX and currY to an invalid
                                            point.
                                      break
                                    if there exists a WEAK point:
                                          update currX and currY to that point's
                                          coordinates
                                    else:
                                          Update currX and currY to the
                                          coordinates of the neighbour with the
                                          largest distance value
                                    if (currX, currY) is in visited:
                                          break
                              else:
                                          add (currX, currY) to visited
if (currX, currY) is invalid or not on the shape:
      continue
for each (arrX, arrY) in tent
      add (arrX, arrY) to S
```



Skeletonization Algorithm

Final skeleton



Sample problem

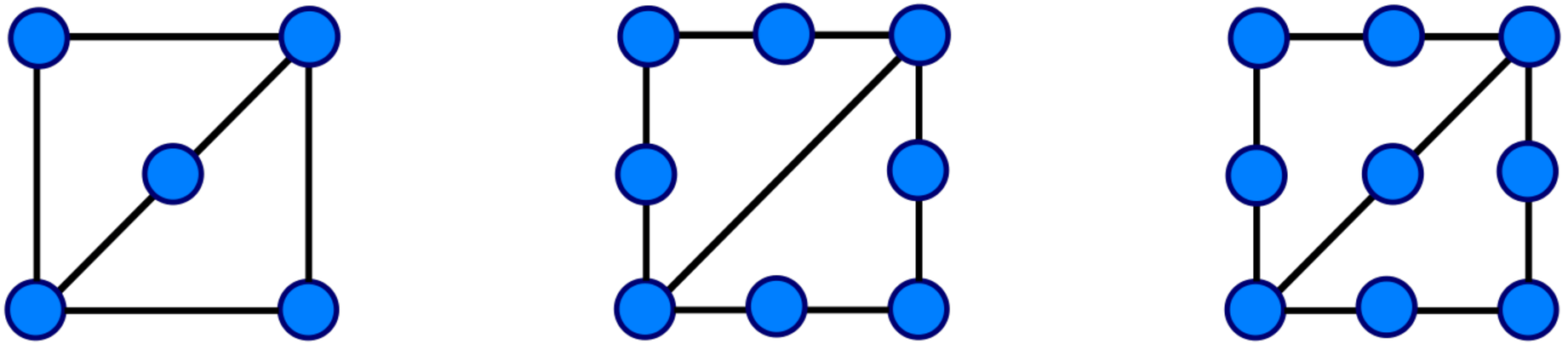
You are a research assistant helping a famous CS professor. Dr. Prof has made a groundbreaking discovery - that graphs can be made from shapes and used in shape matching. Dr. Prof has created a demo graph of a shape to present at a conference tomorrow.

The night before the conference, you accidentally delete the file containing Dr. Prof's graph. All you have left is the file containing the image of the original shape.

Can you recreate a homeomorphic graph of the shape before Dr. Prof fires you?

Sample problem

Example of homeomorphic graphs:



Two graphs that can be made equal through adding/removing vertices.

Solution to sample problem

We can create a skeleton of the shape and turn the points into nodes of the graph.

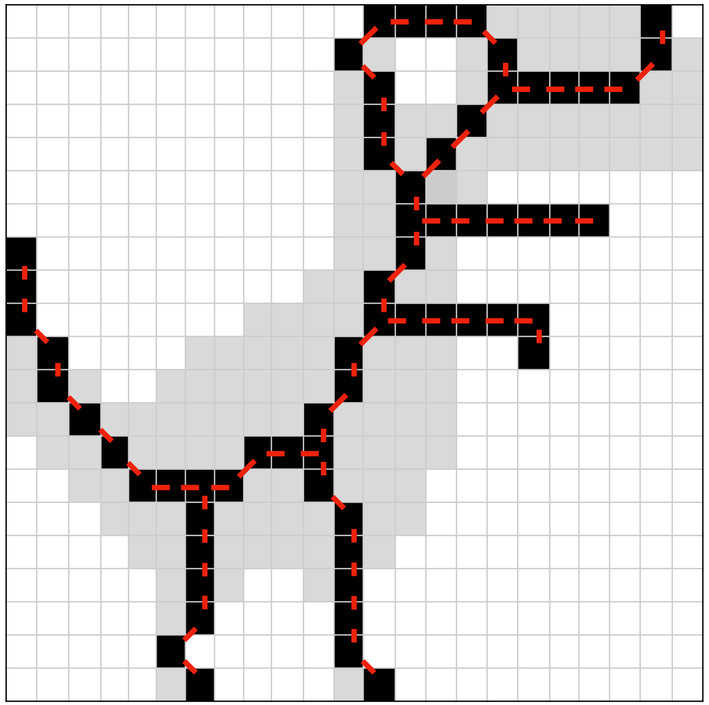


Image references

B skeleton: https://upload.wikimedia.org/wikipedia/commons/9/93/Skel.png

Horse: https://scikit-image.org/docs/dev/_images/sphx_glr_plot_skeleton_001.png

Pathfinding in game: http://gram.cs.mcgill.ca/theses/singh-15-using.pdf

Shape matching (horse): https://cis.temple.edu/~latecki/Papers/PAMIshape08.pdf

Image compression (text): http://www.inf.u-szeged.hu/~palagyi/skel/skel.html#Applications

Dry grass: https://i.pinimg.com/originals/72/da/81/72da81b9bea118513b17956f31b55fe6.jpg

Chrome dino: https://p7.hiclipart.com/preview/372/920/14/tyrannosaurus-dino-t-rex-t-rex-chrome-vr-jump-trex-runner-lava-jump-dinosaur.jpg

Homeomorphic graphs: https://en.wikipedia.org/wiki/Homeomorphism_(graph_theory)