CS 307 – Introduction to Computer Graphics
Semester II, 2019/20

Topic 4: Filling

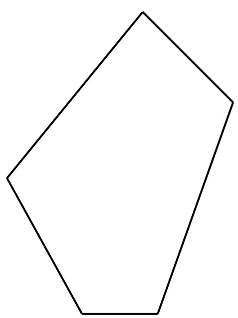
Filling Polygons

✓ how to draw lines and circles

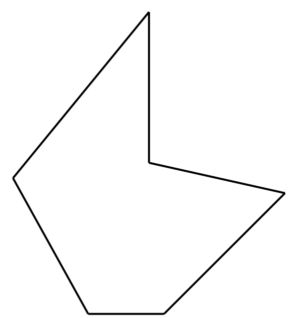
- How to draw polygons?
  - use an incremental algorithm
  - known as the scan-line algorithm

Filling Polygons

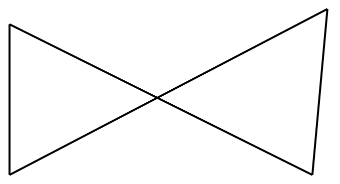
- Three types of polygons



Simple convex



simple concave

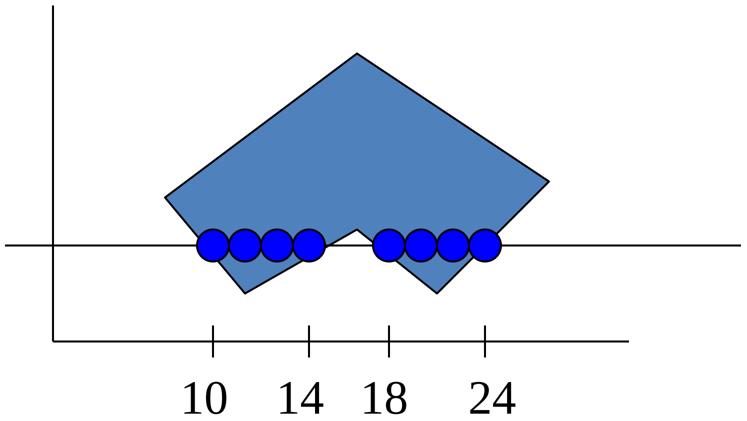


non-simple
(self-intersection)

Area filling on raster systems

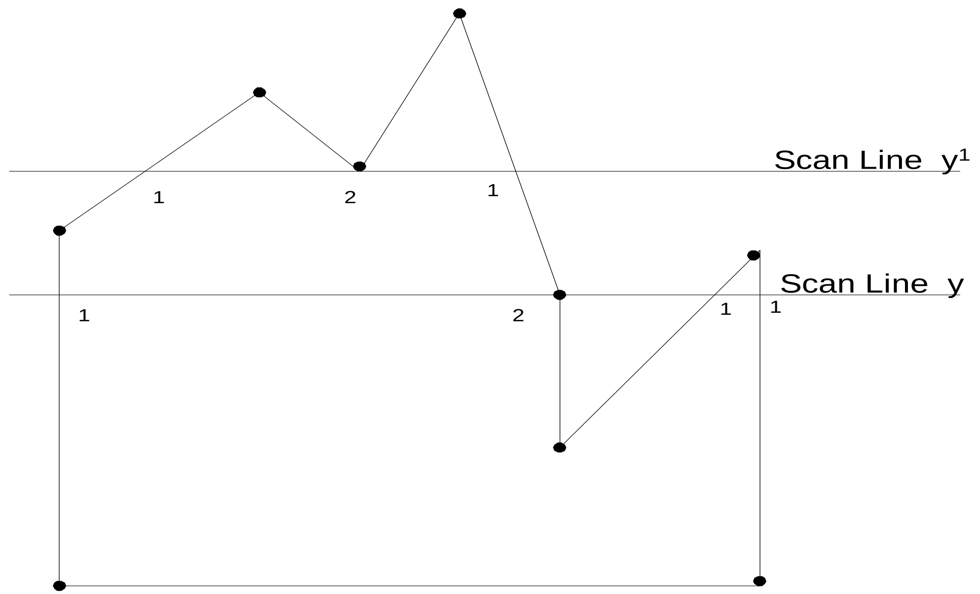
- Two basic approaches
  - Determine the overlap intervals for scan lines that cross the area
    - Used in general graphical packages to fill polygons, circles, ellipses etc
  - Start from a given interior position and paint outward until a specified boundary is reached
    - Used for more complex boundaries and interactive paining

Scan-line polygon filling



- For each scan line crossing the polygon
    - Locate the intersection positions of the scan line with the polygon edges
    - These intersection points are then sorted from left to right , and the corresponding frame buffer positions between each intersection pair are set to specified color

Scan-line polygon filling cont



- Scan line y - two intersecting edges sharing a vertex are on opposite sides of the scan line
- Scan line y’ - two intersecting edges are both above the scan line

Scan-line polygon filling cont

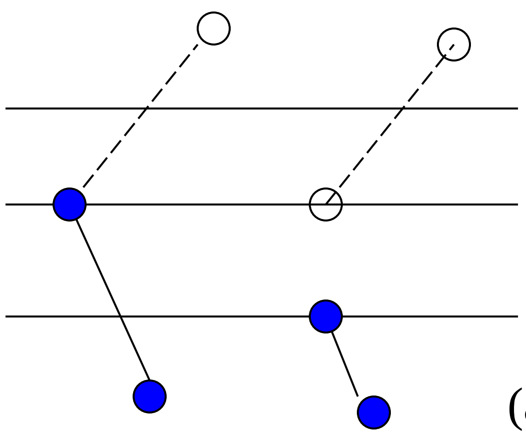
- Vertices that have connecting edges on opposite sides of scan line require additional processing
  - trace around the polygon boundary either in clock-wise or anti-clockwise order
  - observe the relative changes in vertex y coordinates as we move from one edge to the next
  - If the endpoint y values of two consecutive edges monotonically increase or decrease count the middle vertex as a single intersection point for any scan line passing through that vertex

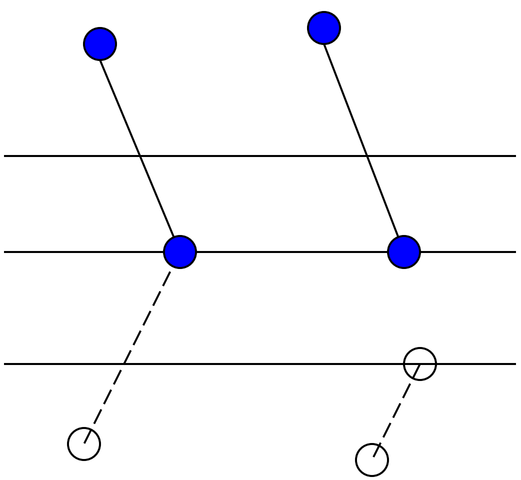
Scan-line polygon filling cont

- If not the shared vertex represents a local extremum on the polygon boundary
  - the two edge intersections with the scan line passing through that vertex can be added to the intersection list
- Counting an intersection as one vertex or two
  - One way to resolve this is also to shorten some polygon edges
    - split those vertices that should be counted as one intersection

Scan-line polygon filling cont

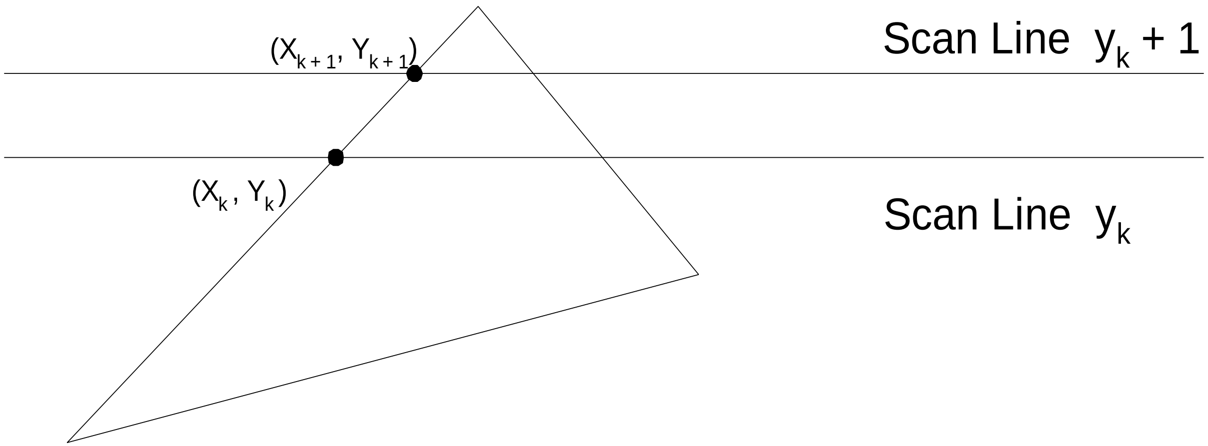
- End point y coordinates of the two edges are increasing
  - y value of the upper endpoint for the current edge is decreased by 1
- Endpoint y values are monotonically decreasing
  - y coordinate of the upper endpoint of the edge following the current edge is decreased


(a)


(b)

Scan-line polygon filling cont

- Relationships between parts of the scene (coherence properties) can be used to reduce processing



- Slope of the edge is constant from one scan line to the other
  - Can set up incremental calculations along any edge

Scan-line polygon filling cont

- Slope of the polygon boundary in terms of scan-line intersection coordinates:

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

- Change of y coordinates between two scan lines:

$$y_{k+1} - y_k = 1$$

- x- intersection value x_{k+1} can be determined by x- intersection value x_k

$$x_{k+1} = x_k + \frac{1}{m}$$

  - Each successive x intercept can be calculated by adding intercept and rounding to the nearest integer

Efficient polygon filling

- Store the polygon boundary in a *sorted edge table*
  - Proceed around the edges in clockwise/ anti-clockwise order
  - Only non horizontal edges are entered in the table
  - Use a bucket /bin sort to store edges, sorted on the smallest y value of each edge
  - Each entry in the table for a particular scan line contains:
    - Maximum y value for that edge
    - X-intercept value (at lower vertex) for the edge
    - Inverse slope of the edge

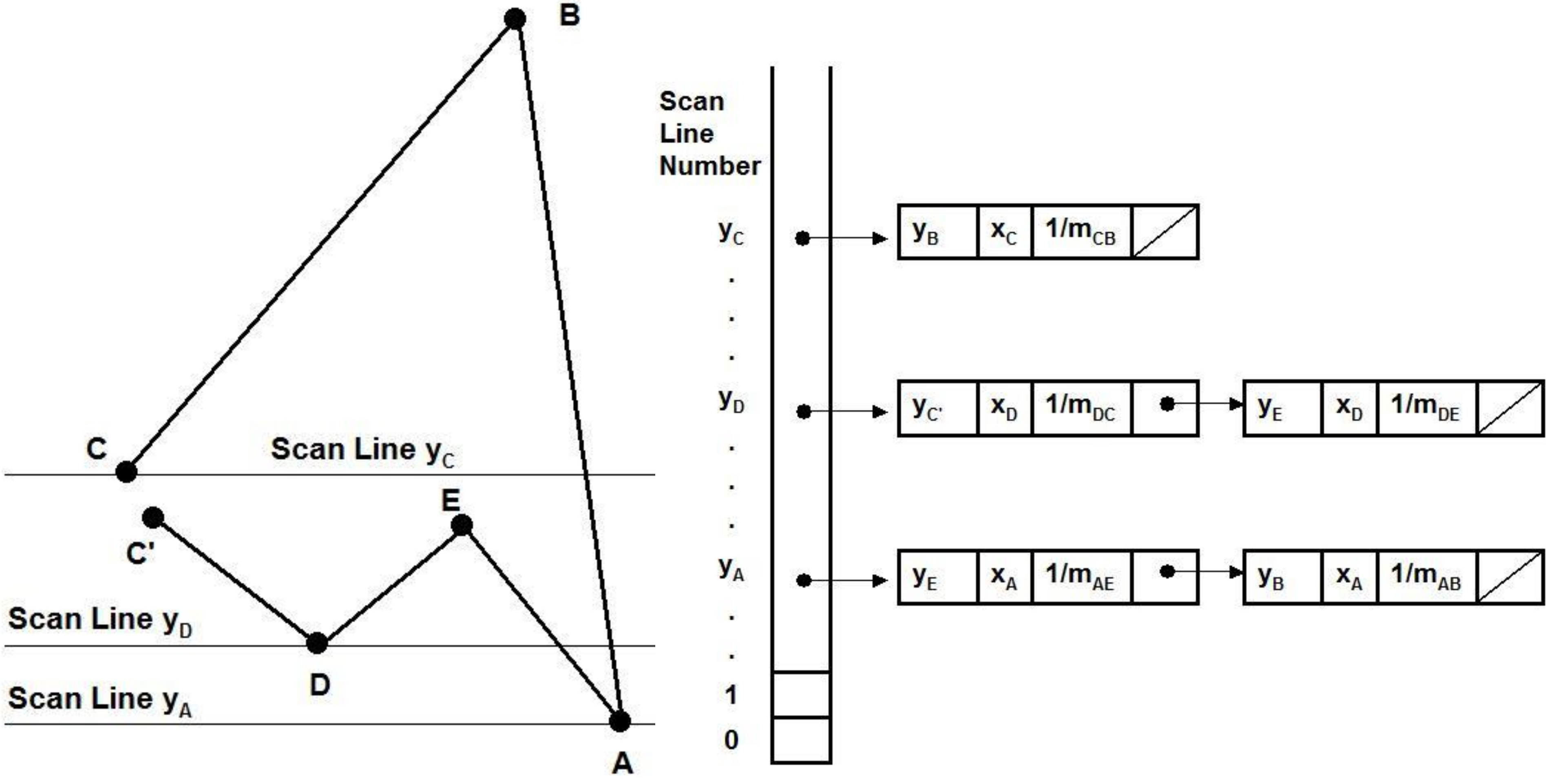
Efficient polygon filling cont.

- Produce the *active edge list*
  - Process the scan lines from the bottom of the polygon to the top
  - Produce the list for each scan line crossing the polygon boundaries
    - Contains all edges crossed the scan line
    - Edge intersections obtained by iterative coherence calculations

Efficient polygon filling cont.

- Each entry in the table for a particular scan line contains
  - the maximum y value for that edge,
  - the x-intercept value (at the lower vertex) for the edge, and
  - the inverse slope of the edge.

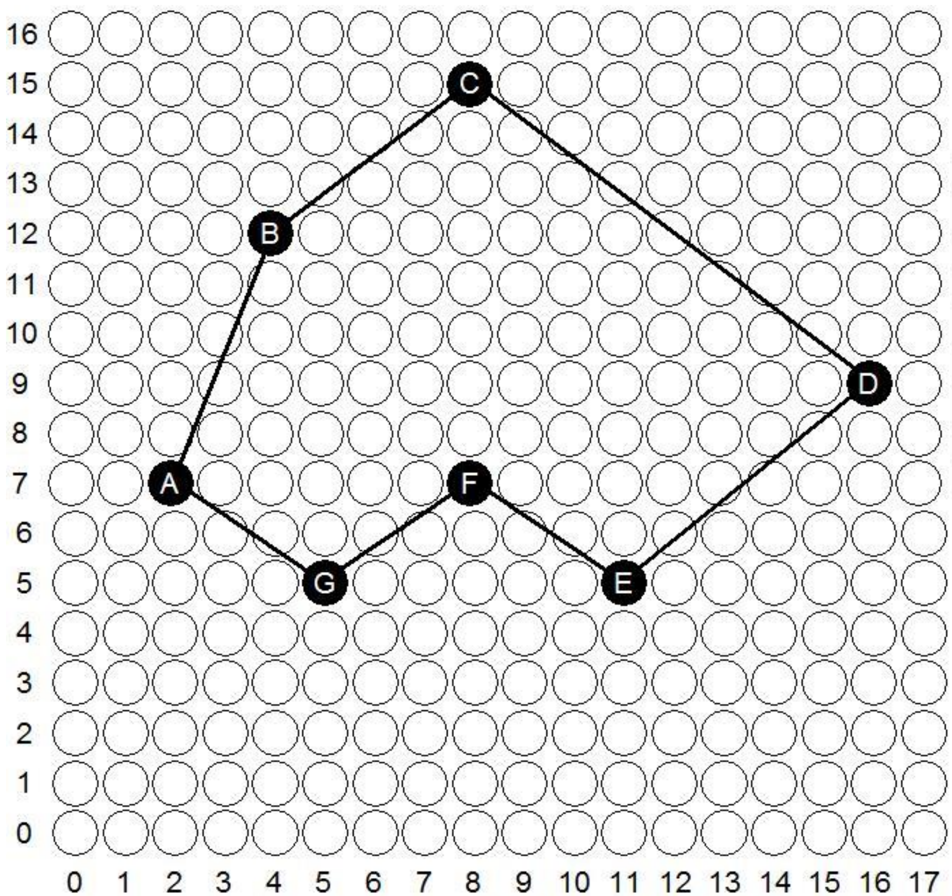
The Scan-Line Polygon Fill Algorithm



The Scan-Line Polygon Fill Algorithm

Polygon = {A, B, C, D, E, F, G}

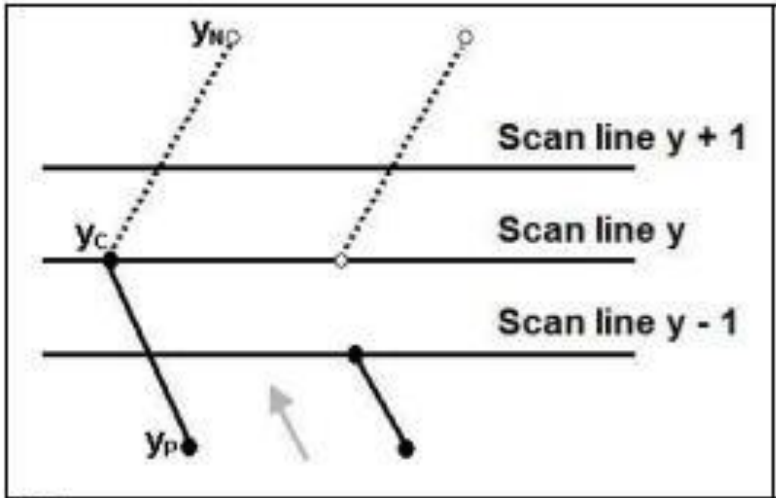
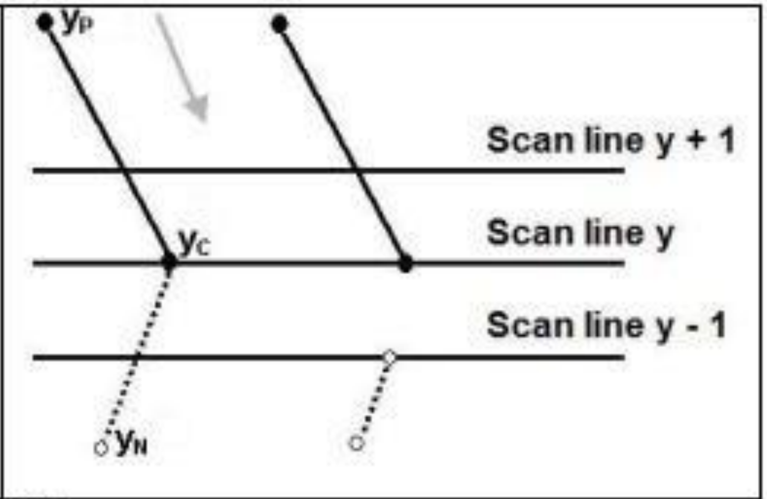
Polygon = {(2, 7), (4, 12), (8,15), (16, 9), (11, 5), (8, 7), (5, 5)}



Edge Table						
#	Edge		1/m	y_{min}	x	y_{max}
0	A (2, 7)	B (4, 12)	2/5	7	2	12
1	B (4, 12)	C (8,15)	4/3	12	4	15
2	C (8,15)	D (16, 9)	– 8/6	9	16	15
3	D (16, 9)	E (11, 5)	5/4	5	11	9
4	E (11, 5)	F (8, 7)	– 3/2	5	11	7
5	F (8, 7)	G (5, 5)	3/2	5	5	7
6	G (5, 5)	A (2, 7)	– 3/2	5	5	7

Scan-Line Polygon Fill Algorithm

Example

	
If $y_P < y_C < y_N$ **Then** y_C is decreased by one. The new edges become $(x_P, y_P) \to (x'_C, y_C - 1)$ and $(x_C, y_C) \to (x_N, y_N)$	**If** $y_P > y_C > y_N$ **Then** y_C is decreased by one. The new edges become $(x_P, y_P) \to (x_C, y_C)$ and $(x'_C, y_C - 1) \to (x_N, y_N)$
$m = (y_P - y_C) / (x_P - x_C)$ $x'_C = x_P + (1/m)(y_C - 1 - y_P)$	$m = (y_N - y_C) / (x_N - x_C)$ $x'_C = x_N + (1/m)(y_C - 1 - y_N)$

Scan-Line Polygon Fill Algorithm

Example

Previous Vertex	Current Vertex	Next Vertex	y_P ? y_C ? y_N	Current Vertex Type	Action
G (5, **5**)	**A** (2, **7**)	B (4, **12**)	$y_P < y_C < y_N$	Not local extremum	Split **A**
A (2, **7**)	**B** (4, **12**)	C (8, **15**)	$y_P < y_C < y_N$	Not local extremum	Split **B**
B (4, **12**)	**C** (8, **15**)	D (16, **9**)	$y_P < y_C > y_N$	Local Maximum	None
C (8, **15**)	**D** (16, **9**)	E (11, **5**)	$y_P > y_C > y_N$	Not local extremum	Split **D**
D (16, **9**)	**E** (11, **5**)	F (8, **7**)	$y_P > y_C < y_N$	Local Minimum	None
E (11, **5**)	**F** (8, **7**)	G (5, **5**)	$y_P < y_C > y_N$	Local Maximum	None
F (8, **7**)	**G** (5, **5**)	A (2, **7**)	$y_P > y_C < y_N$	Local Minimum	None

- **Vertex A** should be split into two vertices **A'** ($x_{A'}$, **6**) and **A**(**2, 7**)

$$m = (5 - 7)/(5 - 2) = -2/3$$

$$x'_A = 5 + (-3/2)(7 - 1 - 5) = 7/2 = 3.5 \cong 4$$

The vertex **A** is split to **A'** (**4**, **6**) and **A**(**2, 7**)

Scan-Line Polygon Fill Algorithm

Example

Previous Vertex	Current Vertex	Next Vertex	y_P ? y_C ? y_N	Current Vertex Type	Action
G (5, **5**)	**A** (2, **7**)	B (4, **12**)	$y_P < y_C < y_N$	Not local extremum	Split **A**
A (2, **7**)	**B** (4, **12**)	C (8, **15**)	$y_P < y_C < y_N$	Not local extremum	Split **B**
B (4, **12**)	**C** (8, **15**)	D (16, **9**)	$y_P < y_C > y_N$	Local Maximum	None
C (8, **15**)	**D** (16, **9**)	E (11, **5**)	$y_P > y_C > y_N$	Not local extremum	Split **D**
D (16, **9**)	**E** (11, **5**)	F (8, **7**)	$y_P > y_C < y_N$	Local Minimum	None
E (11, **5**)	**F** (8, **7**)	G (5, **5**)	$y_P < y_C > y_N$	Local Maximum	None
F (8, **7**)	**G** (5, **5**)	A (2, **7**)	$y_P > y_C < y_N$	Local Minimum	None

- **Vertex B** should be split into two vertices **B'** ($x_{B'}$, **11**) and **B**(**4, 12**)

$$m = (7 - 12)/(2 - 4) = 5/2$$

$$x'_A = 2 + (2/5)(12 - 1 - 7) = 18/5 = 3.6 \cong 4$$

The vertex **B** is split to **B'** (**4**, **11**) and **B**(**4, 12**)

Scan-Line Polygon Fill Algorithm

Example

Previous Vertex	Current Vertex	Next Vertex	y_P ? y_C ? y_N	Current Vertex Type	Action
G (5, **5**)	**A** (2, **7**)	B (4, **12**)	$y_P < y_C < y_N$	Not local extremum	Split **A**
A (2, **7**)	**B** (4, **12**)	C (8,**15**)	$y_P < y_C < y_N$	Not local extremum	Split **B**
B (4, **12**)	**C** (8,**15**)	D (16, **9**)	$y_P < y_C > y_N$	Local Maximum	None
C (8,**15**)	**D** (16, **9**)	E (11, **5**)	$y_P > y_C > y_N$	Not local extremum	Split **D**
D (16, **9**)	**E** (11, **5**)	F (8, **7**)	$y_P > y_C < y_N$	Local Minimum	None
E (11, **5**)	**F** (8, **7**)	G (5, **5**)	$y_P < y_C > y_N$	Local Maximum	None
F (8, **7**)	**G** (5, **5**)	A (2, **7**)	$y_P > y_C < y_N$	Local Minimum	None

- **Vertex D** should be split into two vertices **D**(**16, 9**) and **D'** ($x_{D'}$, **8**)

$$m = (5 - 9)/(11 - 16) = 4/5$$

$$x'_D = 11 + (5/4)(9 - 1 - 5) = 59/4 = 14.75 \cong 15$$

The vertex **D** is split to **D**(**16, 9**) and **D'** (**15**, **8**)

Scan-Line Polygon Fill Algorithm

Example

Modified Edge Table							
#	Edge		1/m	y_{min}	x	y_{max}
0	A (2, 7)	B' (4, 11)	2/5	7	2	**11**
1	B (4, 12)	C (8,15)	4/3	12	4	15
2	C (8,15)	D (16, 9)	– 8/6	9	16	15
3	D' (15, 8)	E (11, 5)	5/4	5	11	**8**
4	E (11, 5)	F (8, 7)	– 3/2	5	11	7
5	F (8, 7)	G (5, 5)	3/2	5	5	7
6	G (5, 5)	A' (4, 6)	– 3/2	5	5	**6**

Activation Table											
y	5	6	7	8	9	10	11	12	13	14	15
Activated Edge #s	3, 4, 5, 6		0		2			1			

Scan-Line Polygon Fill Algorithm

Example

Edge number 0

#	Edge		1/m	y_{min}	x	y_{max}
0	A (2, 7)	B' (4, 11)	2/5 = 0.4	7	2	11

Scan line	x-intersection
y = 7	2
y = 8	2 + 0.4 = 2.4 ~ 2
y = 9	2.4 + 0.4 = 2.8 ~ 3
y = 10	2.8 + 0.4 = 3.2 ~ 3
y = 11	4

Edge number 1

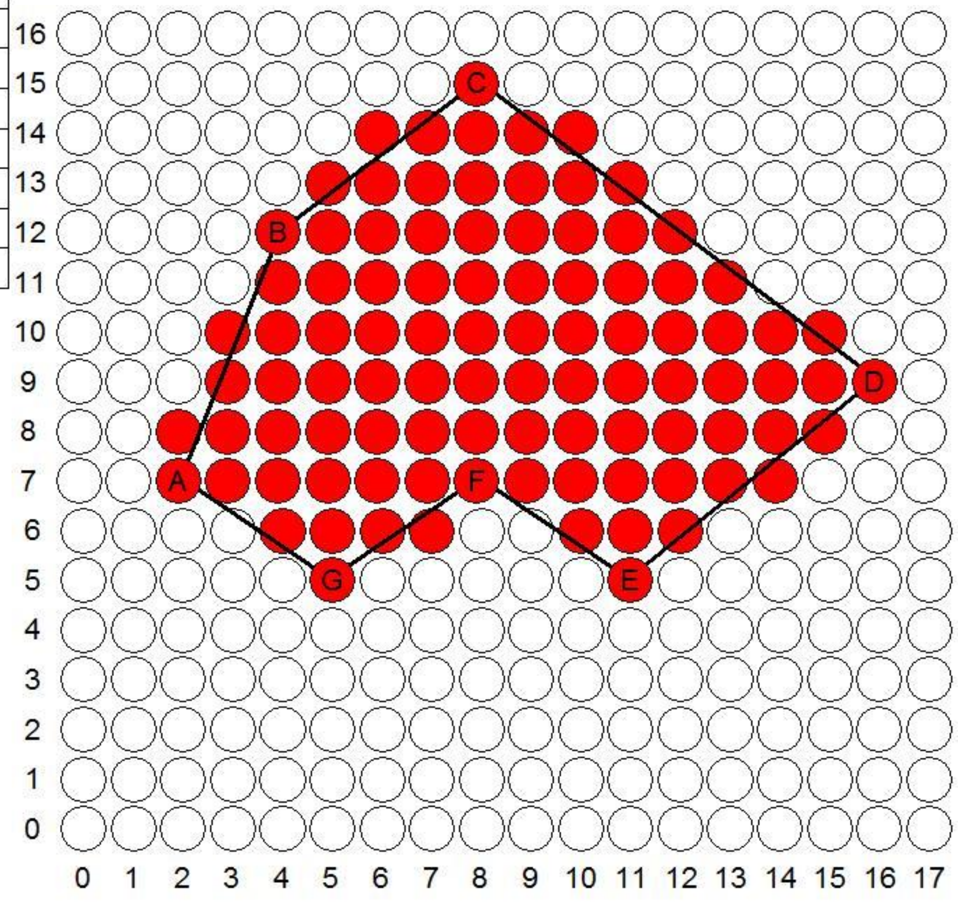
#	Edge		1/m	y_{min}	x	y_{max}
1	B (4, 12)	C (8,15)	4/3 = 1.3	12	4	15

Scan line	x-intersection
y = 12	4
y = 13	4 + 1.3 = 4.3 ~ 4
y = 14	4.3 + 1.3 = 5.6 ~ 6
y = 15	8

Scan-Line Polygon Fill Algorithm

Example

Scan line	x-intersections Edge#							x-intersections pair Ascending order
	0	1	2	3	4	5	6	
5				11	11	5	5	(5, 5), (11, 11)
6				12	10	7	4	(4, 7), (10, 12)
7	2			14	8	8		(2, 8), (8,14)
8	2			15				(2,15)
9	3		16					(3, 16)
10	3		15					(3, 15)
11	4		13					(4, 13)
12		4	12					(4,12)
13		4	11					(4, 11)
14		6	10					(6, 10)
15		8	8					(8, 8)



Boundary Fill Algorithm

- Another approach to area filling
  - start at a point inside a region and paint the interior outward toward the boundary.
  - If the boundary is specified in a single color, the fill algorithm processed outward pixel by pixel until the boundary color is encountered.
- Inputs
  - the coordinate of the interior **point (x, y)**, a **fill color**, and a **boundary color**.

Boundary Fill Algorithm

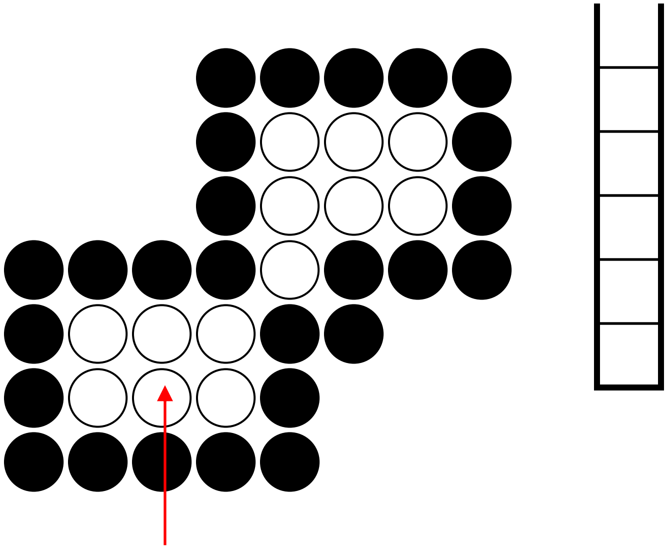
- **recursive** boundary-fill algorithm:
  – Start from an interior point.
  – If the current pixel is **not already** filled and if it is not an edge point, then set the pixel with the fill color, and store its neighboring pixels (**4** or **8-connected**) in the stack for processing.
  – Store only neighboring pixel that is **not already** filled and is not an edge point.
  – Select the next pixel from the stack, and continue with step **2**.

Boundary Fill Algorithm

- The order of pixels that should be added to stack
- **4-connected** - above, below, left, and right.
- **8-connected -** above, below, left, right, above-left, above-right, below-left, and below-right.

Boundary Fill Algorithm

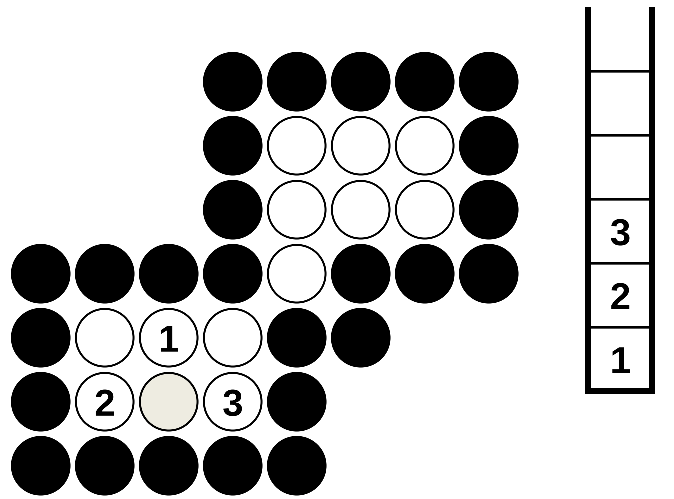
4-connected (Example)



Start Position

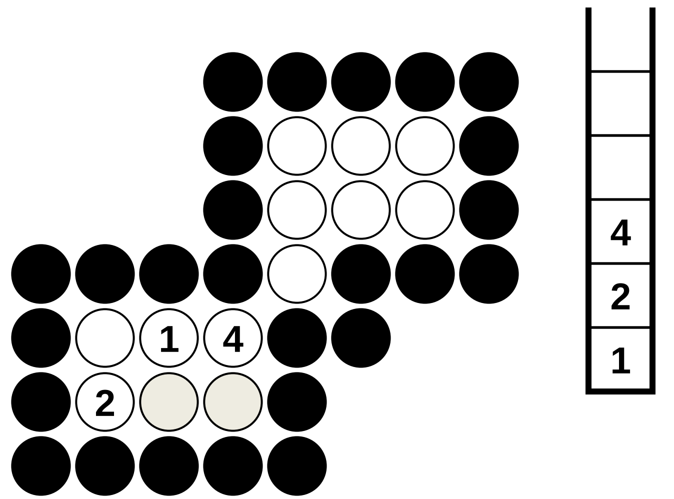
Boundary Fill Algorithm

4-connected (Example)



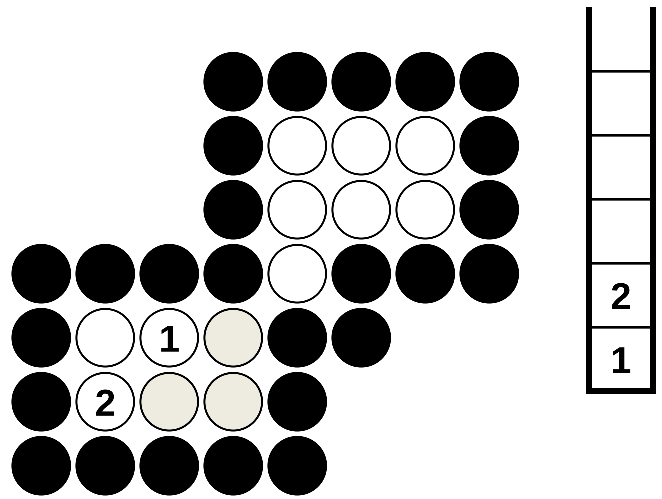
Boundary Fill Algorithm

4-connected (Example)



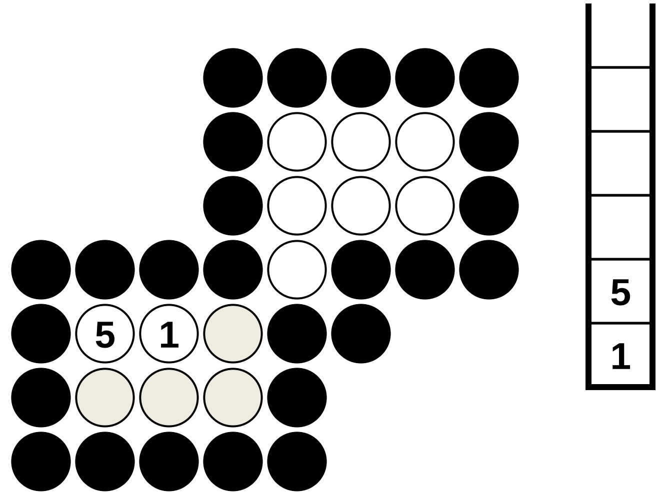
Boundary Fill Algorithm

4-connected (Example)



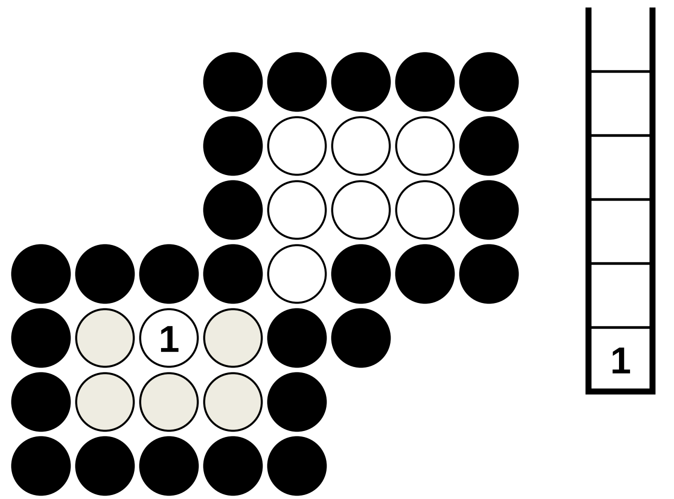
Boundary Fill Algorithm

4-connected (Example)



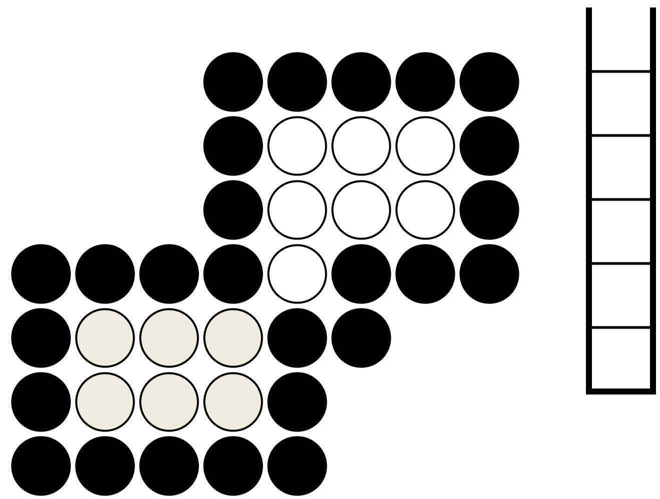
Boundary Fill Algorithm

4-connected (Example)



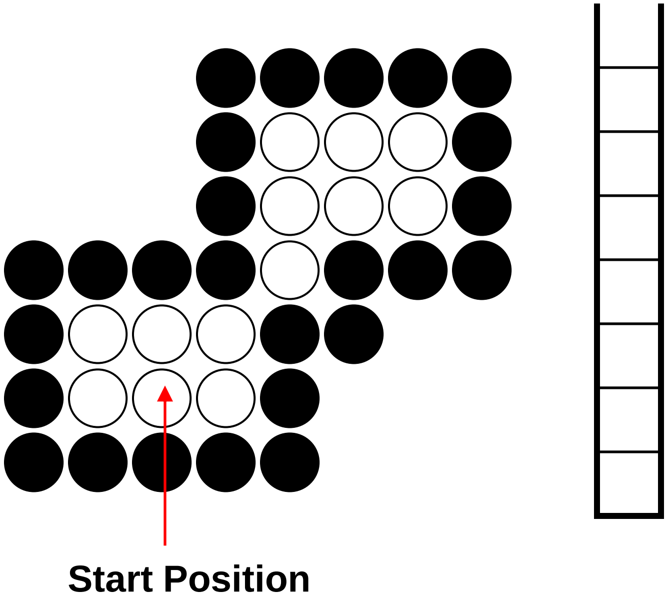
Boundary Fill Algorithm

4-connected (Example)



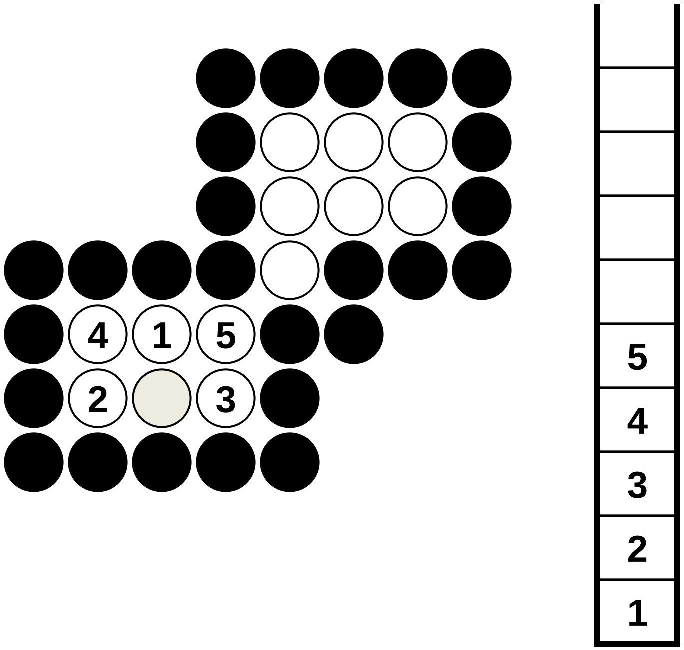
Boundary Fill Algorithm

8-connected (Example)



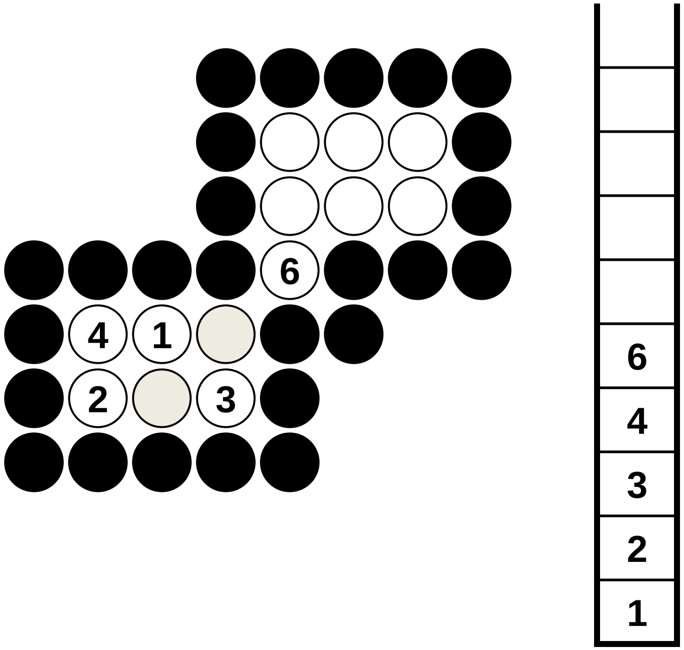
Boundary Fill Algorithm

8-connected (Example)



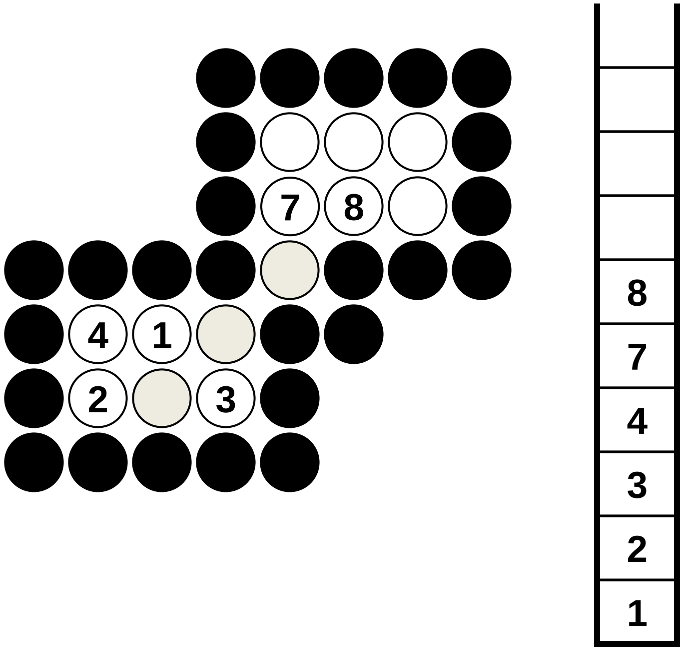
Boundary Fill Algorithm

8-connected (Example)



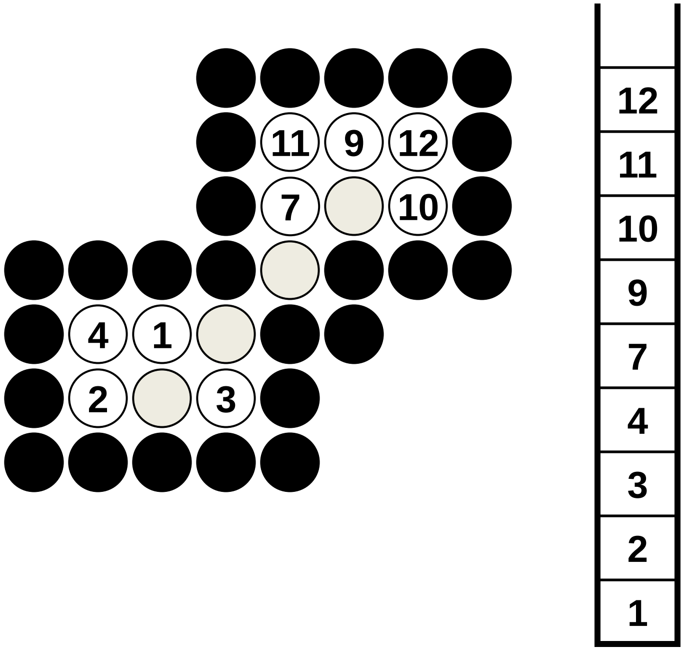
Boundary Fill Algorithm

8-connected (Example)



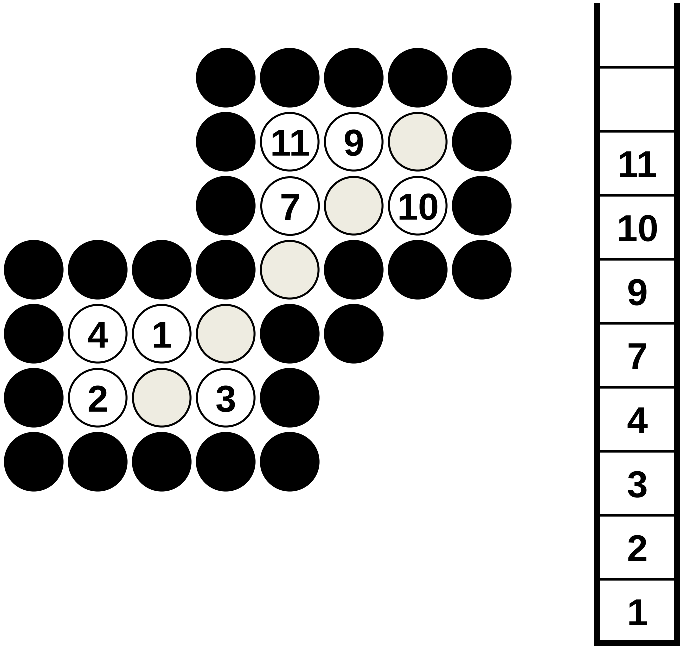
Boundary Fill Algorithm

8-connected (Example)



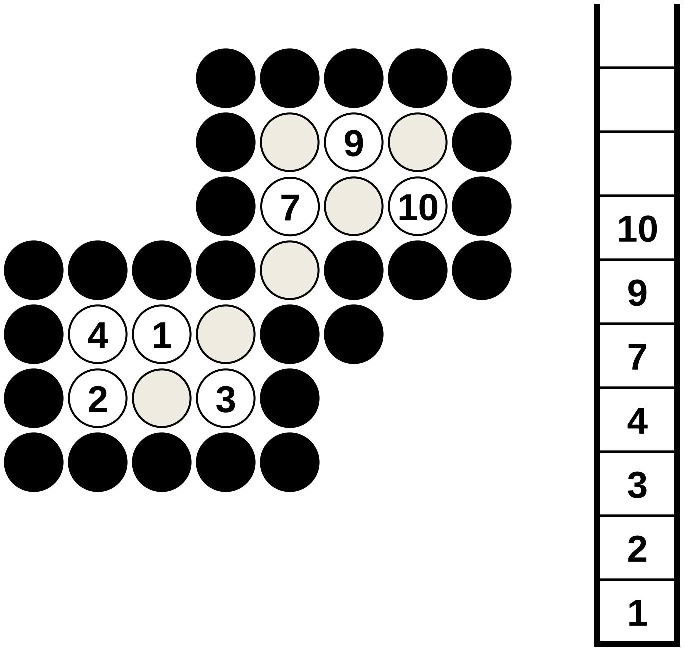
Boundary Fill Algorithm

8-connected (Example)



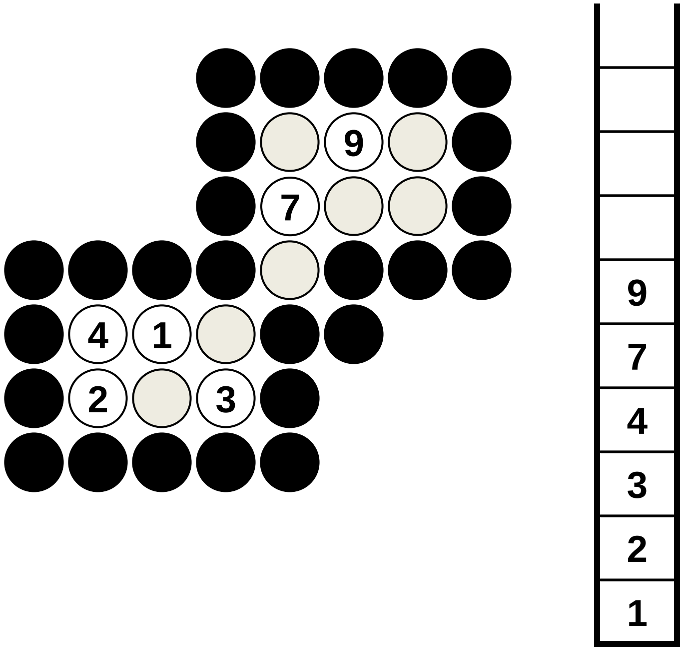
Boundary Fill Algorithm

8-connected (Example)



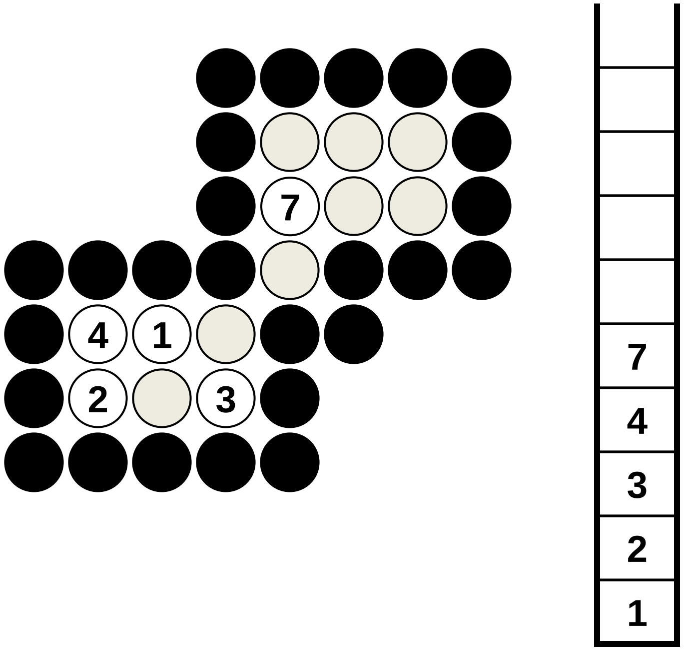
Boundary Fill Algorithm

8-connected (Example)



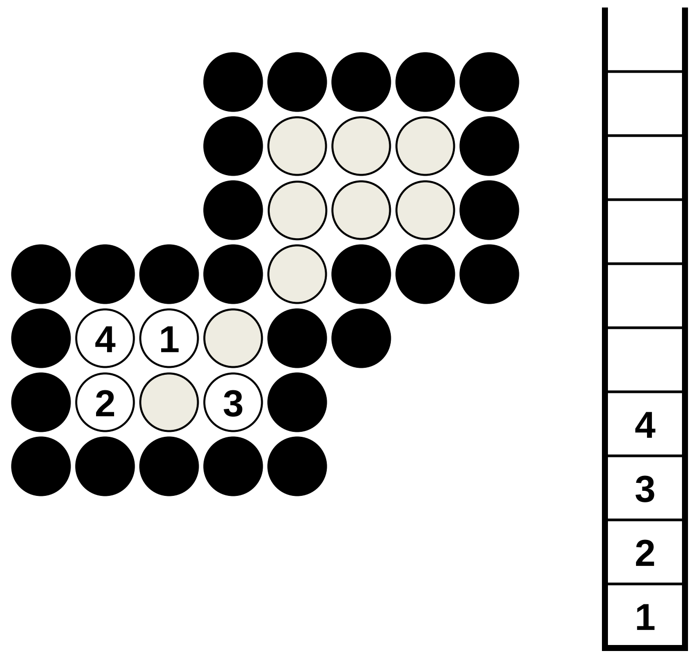
Boundary Fill Algorithm

8-connected (Example)



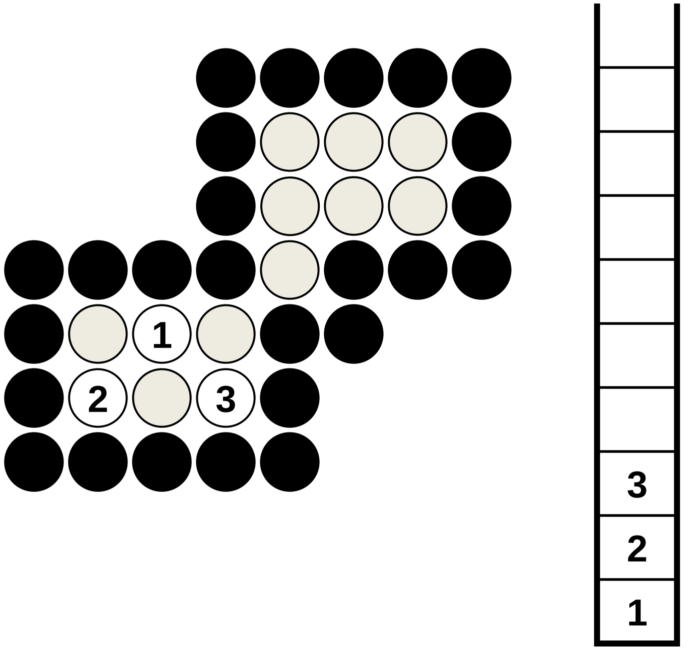
Boundary Fill Algorithm

8-connected (Example)



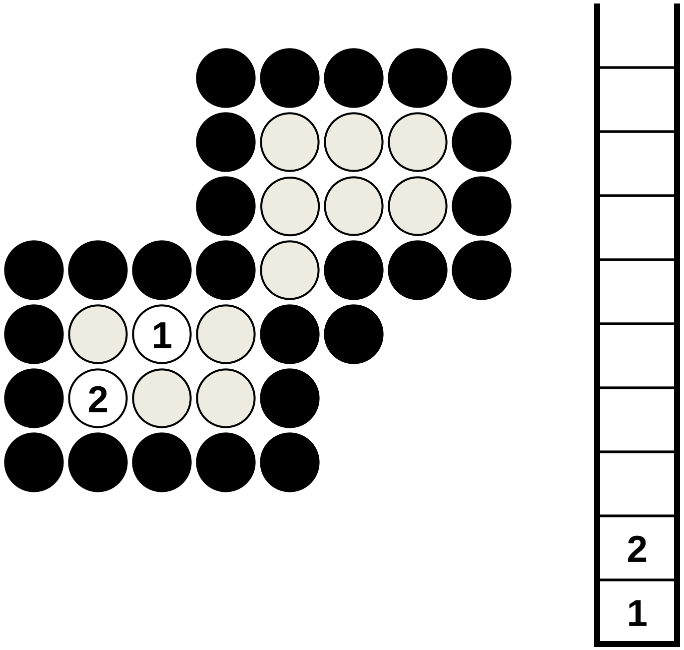
Boundary Fill Algorithm

8-connected (Example)



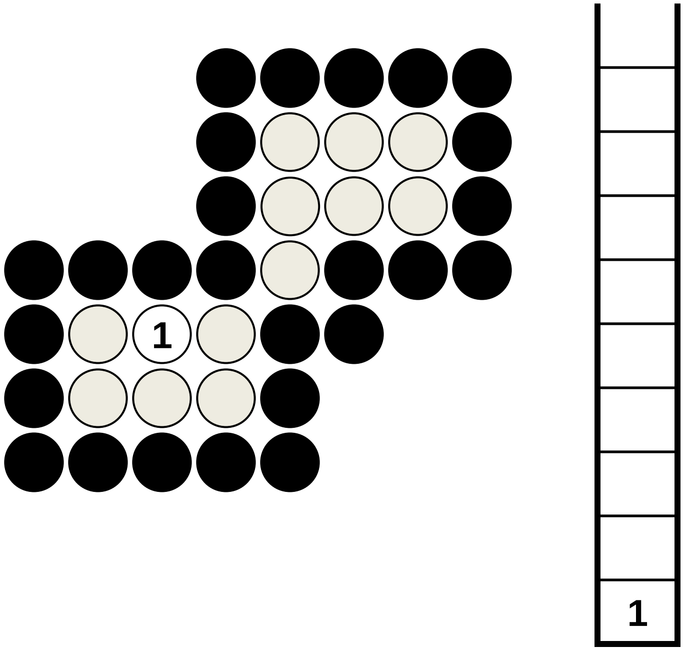
Boundary Fill Algorithm

8-connected (Example)



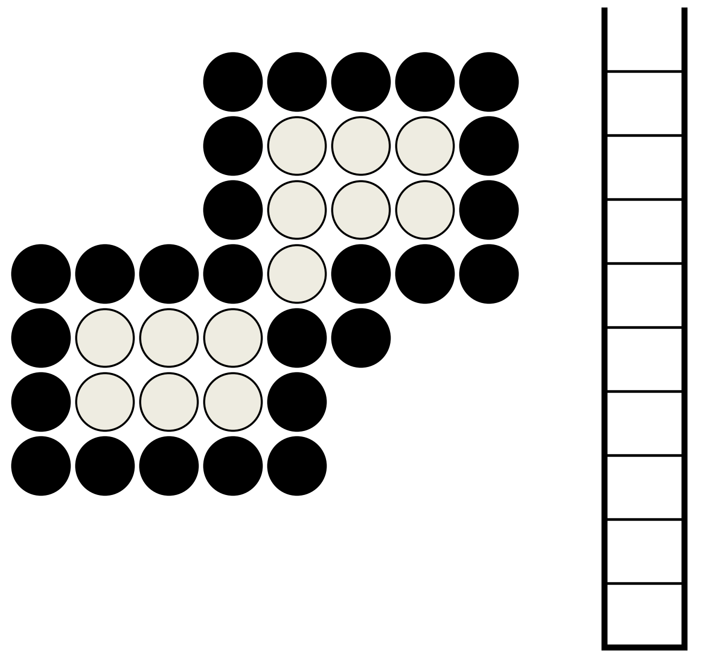
Boundary Fill Algorithm

8-connected (Example)



Boundary Fill Algorithm

8-connected (Example)



Boundary Fill Algorithm

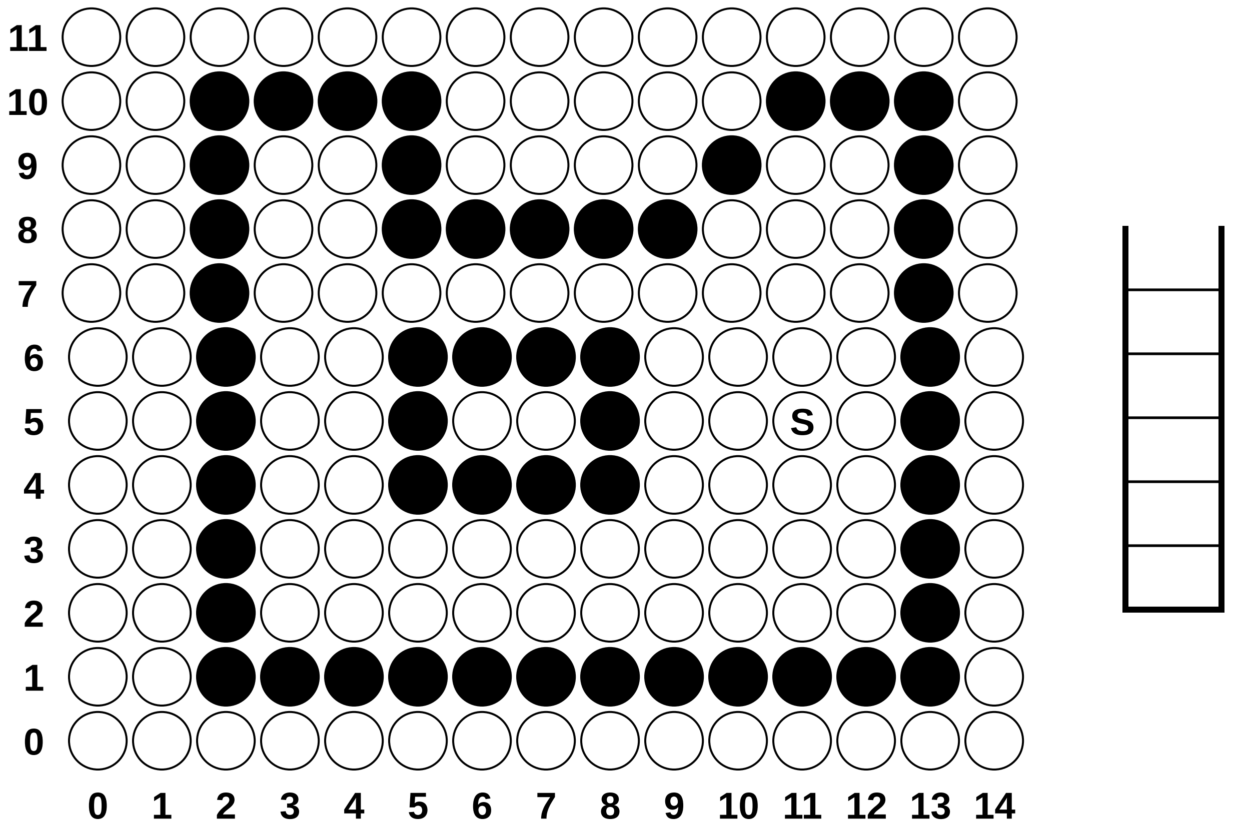
- requires considerable stacking of neighboring pixels
  - more efficient methods are generally used

- **Span Flood-Fill**
  - fill horizontal pixel spans across scan lines
    - need only stack a beginning position for each horizontal pixel spans, instead of stacking all unprocessed neighboring positions around the current position.

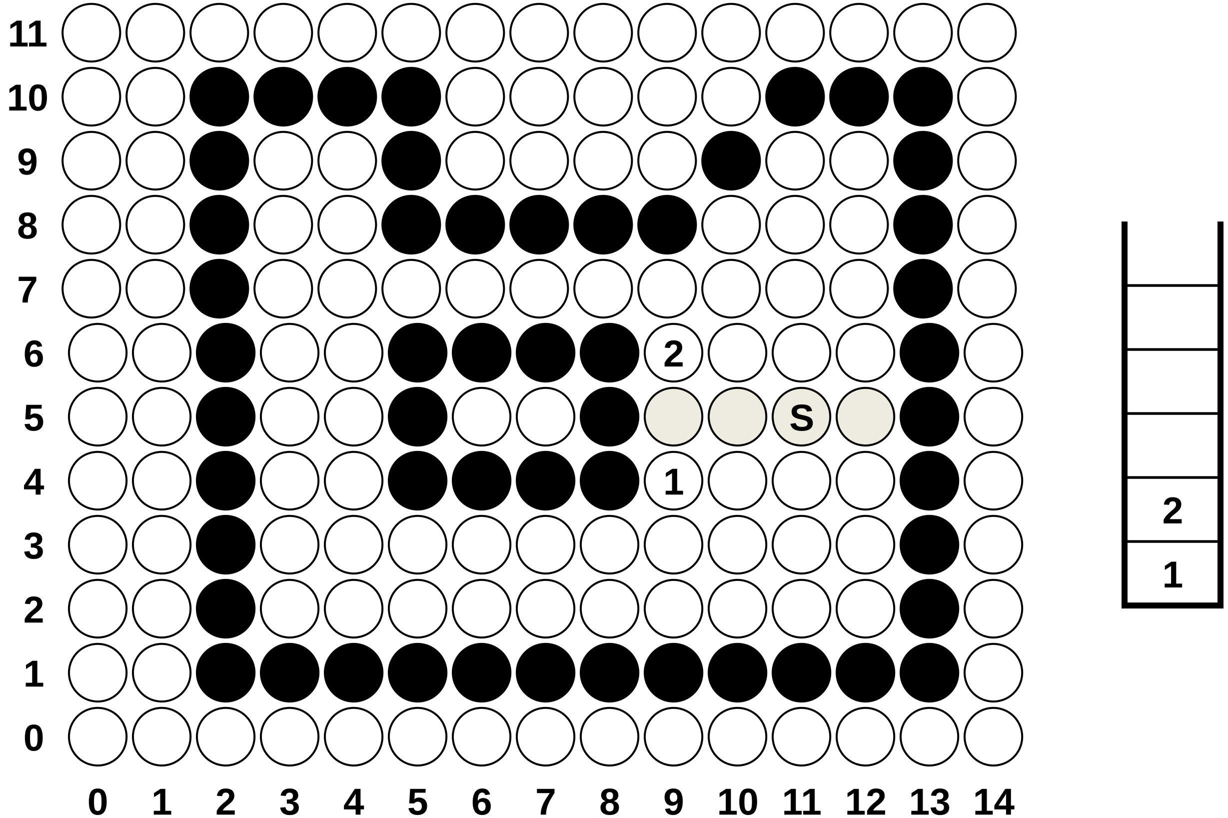
Span Flood-Fill Algorithm

- Algorithm
  - **Starting** from the initial interior pixel, fill in the contiguous span of pixels on this starting scan line.
  - **Then locate** and **stack** starting positions for spans on the adjacent scan lines,
    - where spans are defined as the contiguous horizontal string of positions bounded by pixels displayed in the area border color.
  - **At each subsequent step**, **unstack** the next start position and repeat the process.

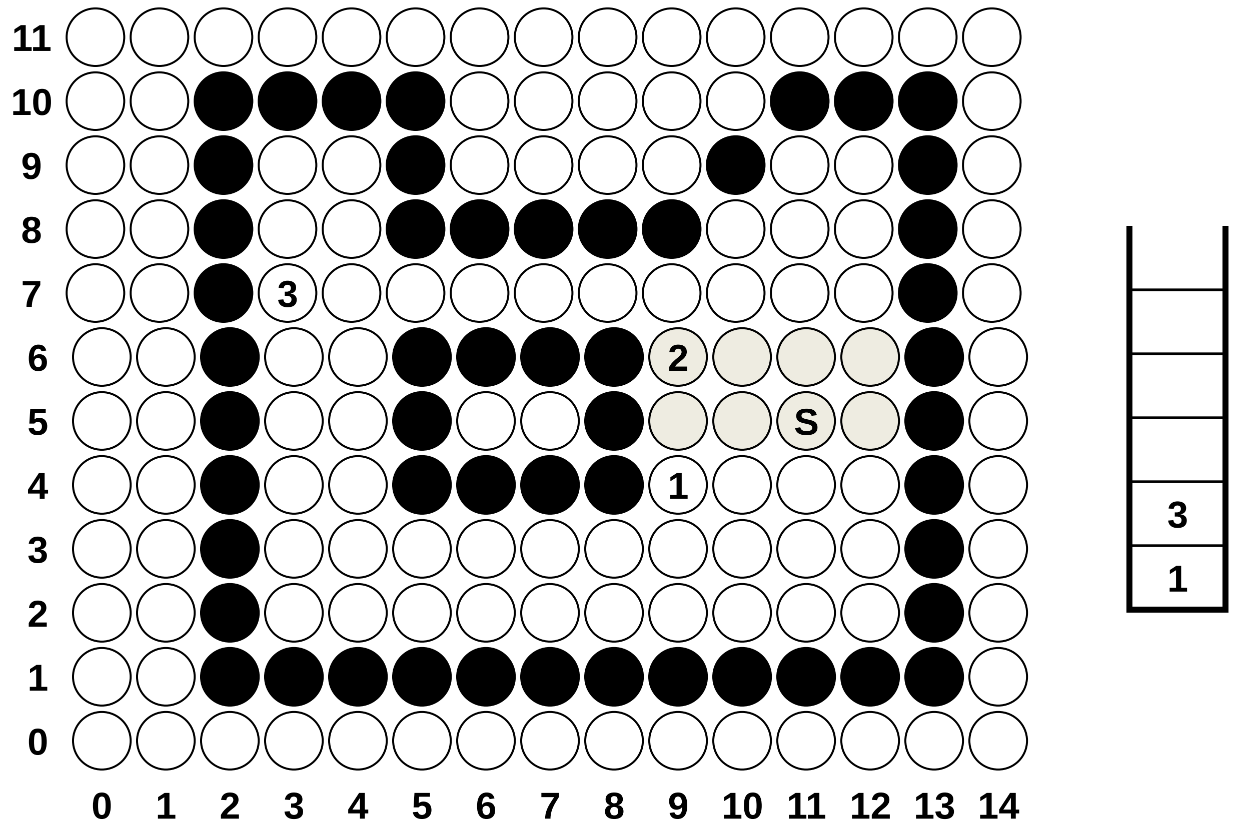
Span Flood-Fill Algorithm (example)



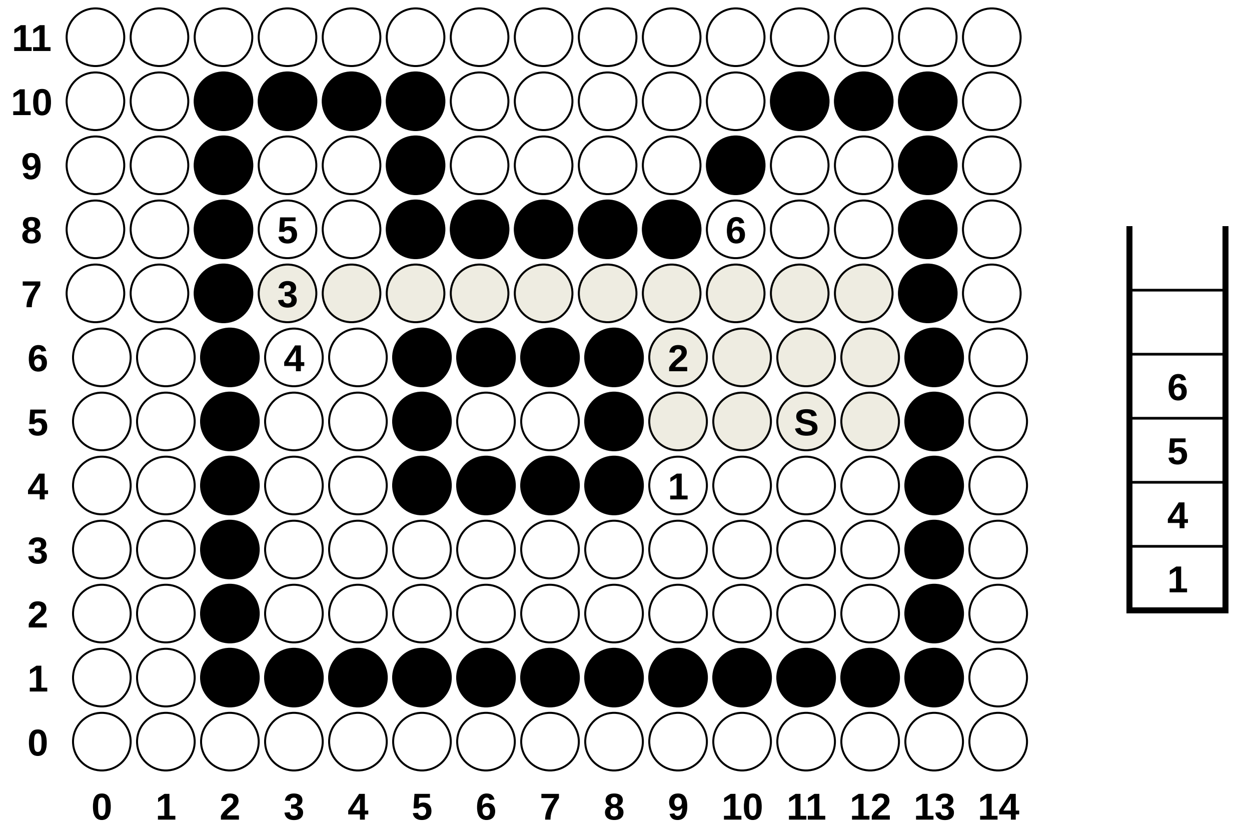
Span Flood-Fill Algorithm (example)



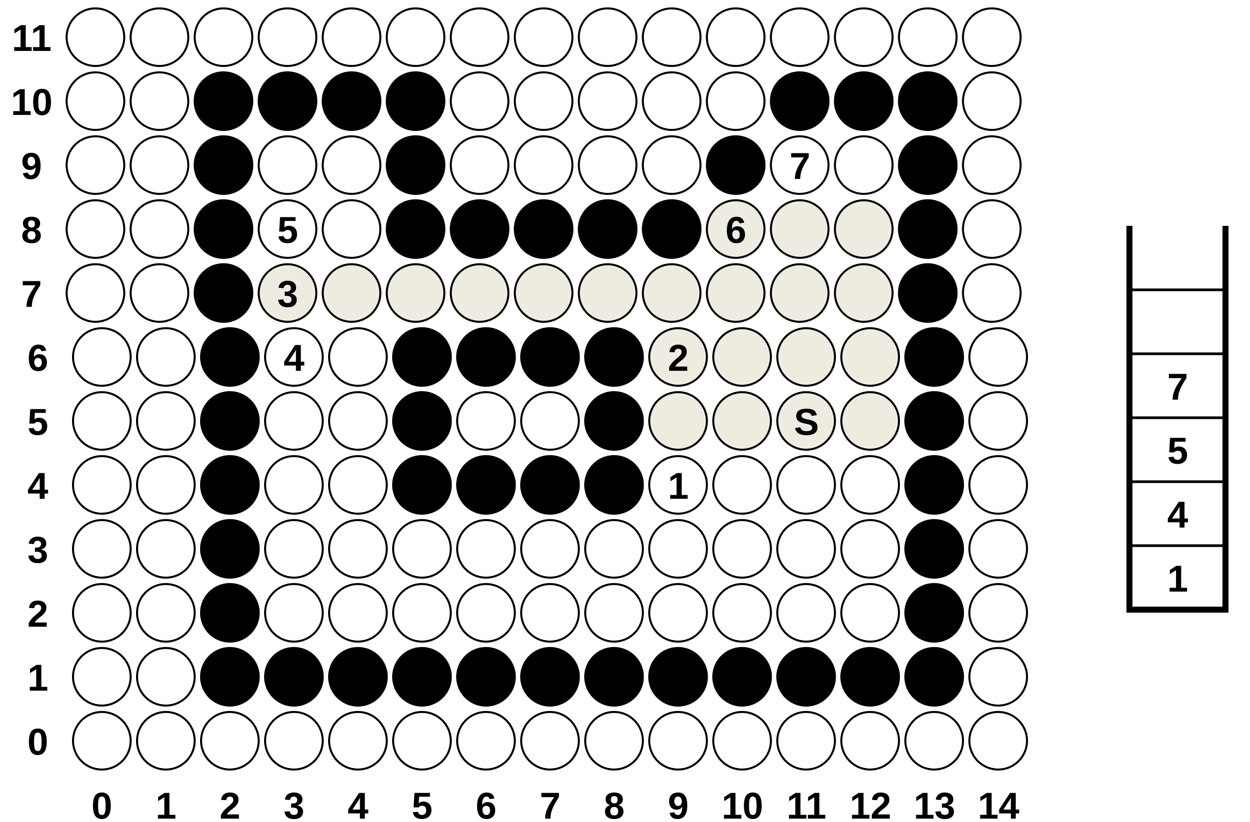
Span Flood-Fill Algorithm (example)



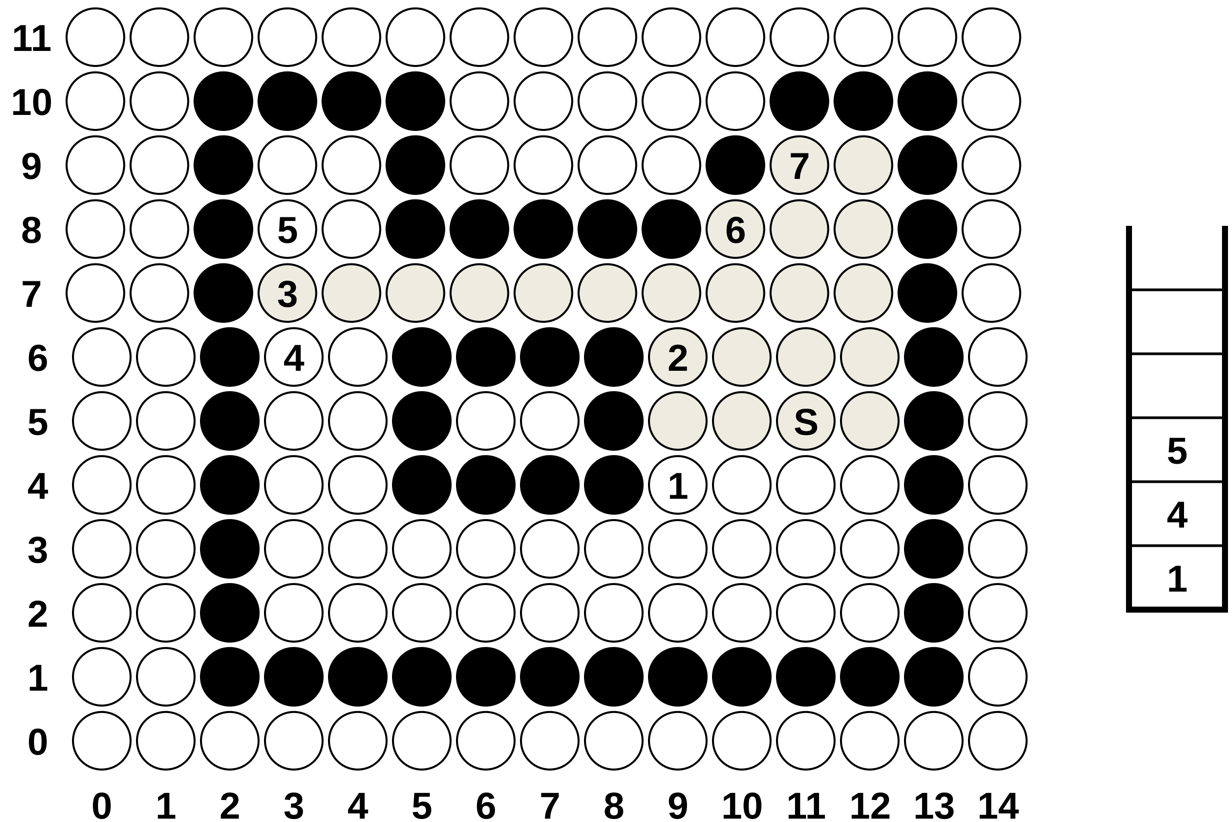
Span Flood-Fill Algorithm (example)



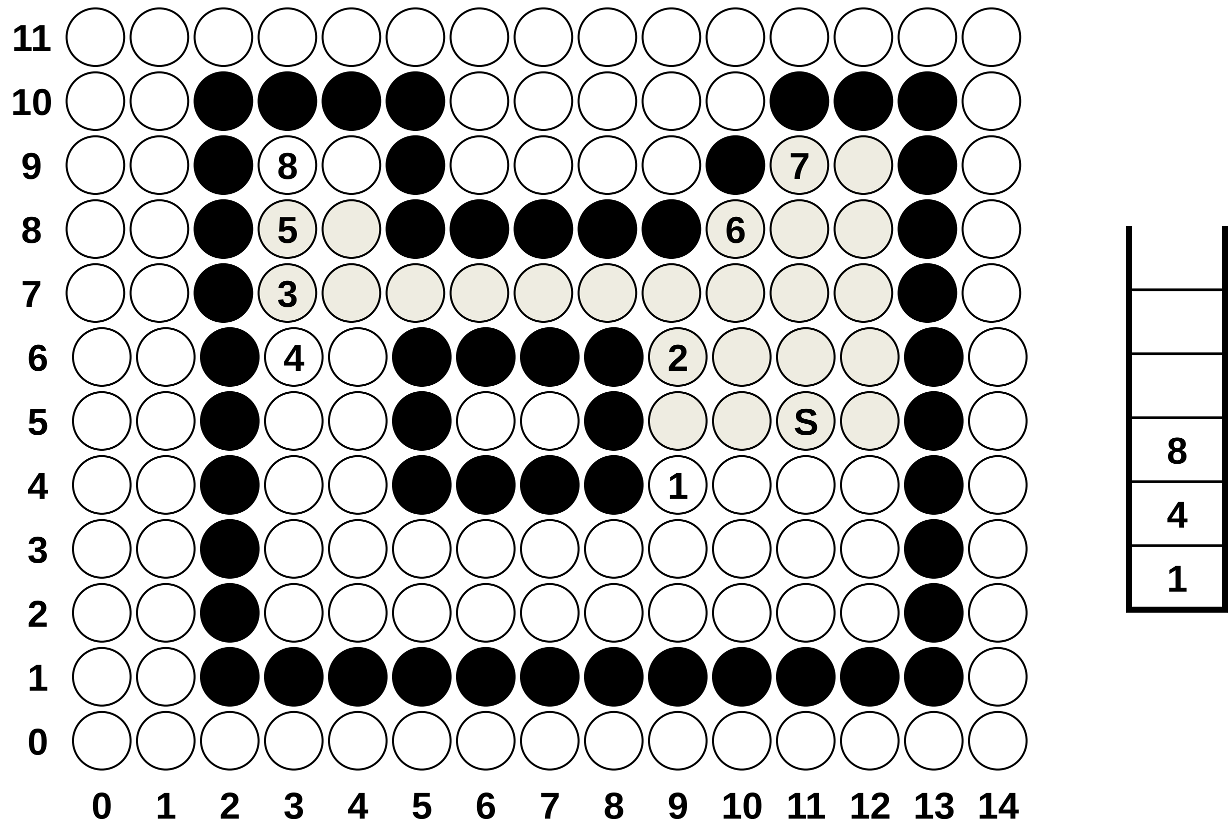
Span Flood-Fill Algorithm (example)



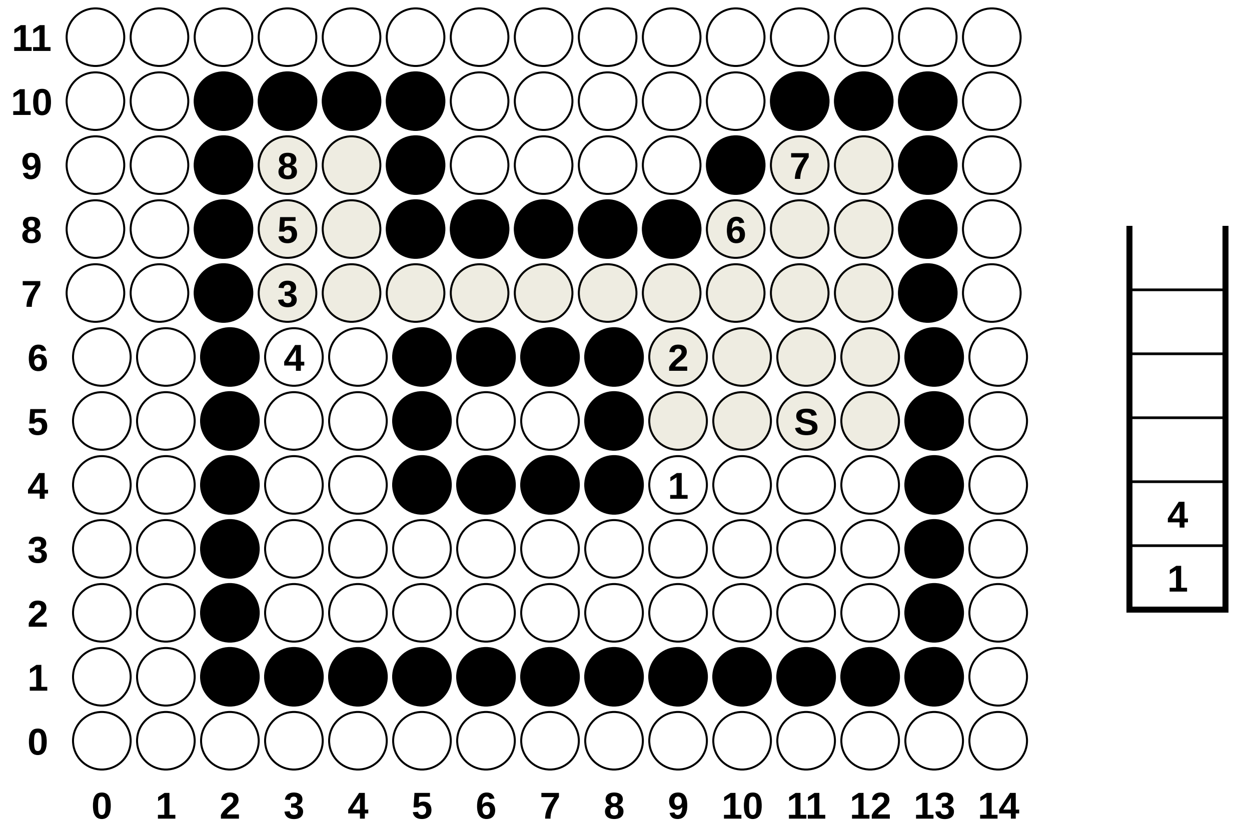
Span Flood-Fill Algorithm (example)



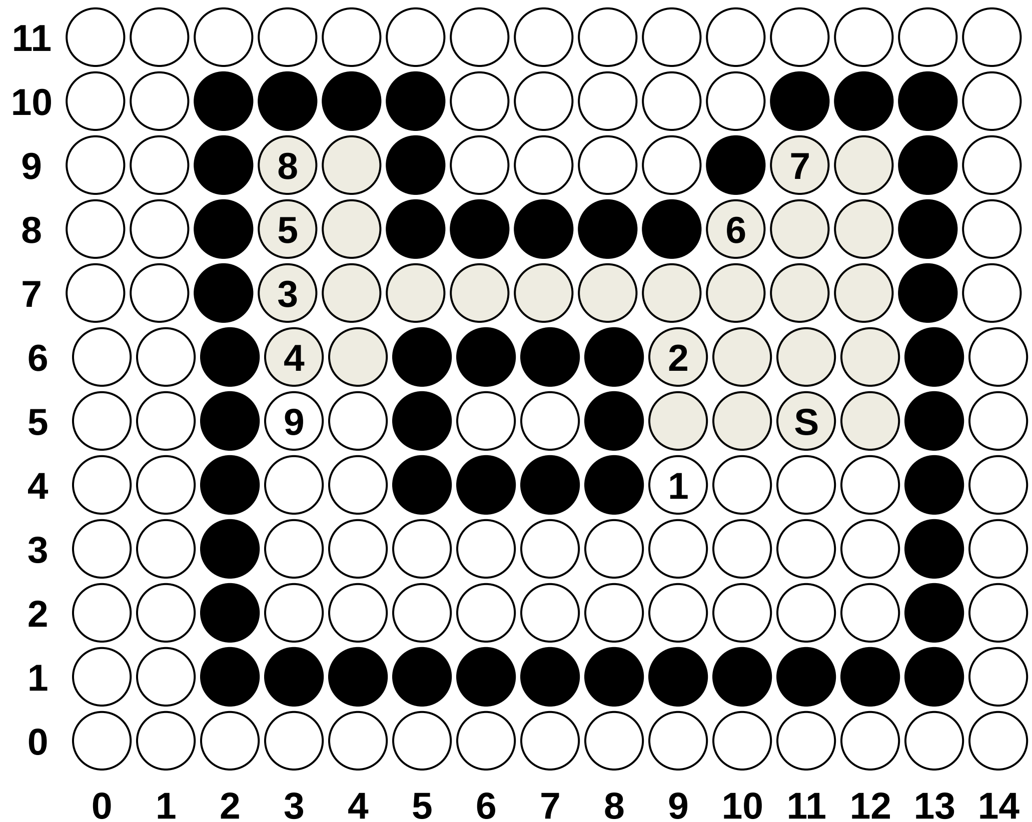
Span Flood-Fill Algorithm (example)

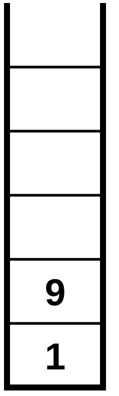


Span Flood-Fill Algorithm (example)

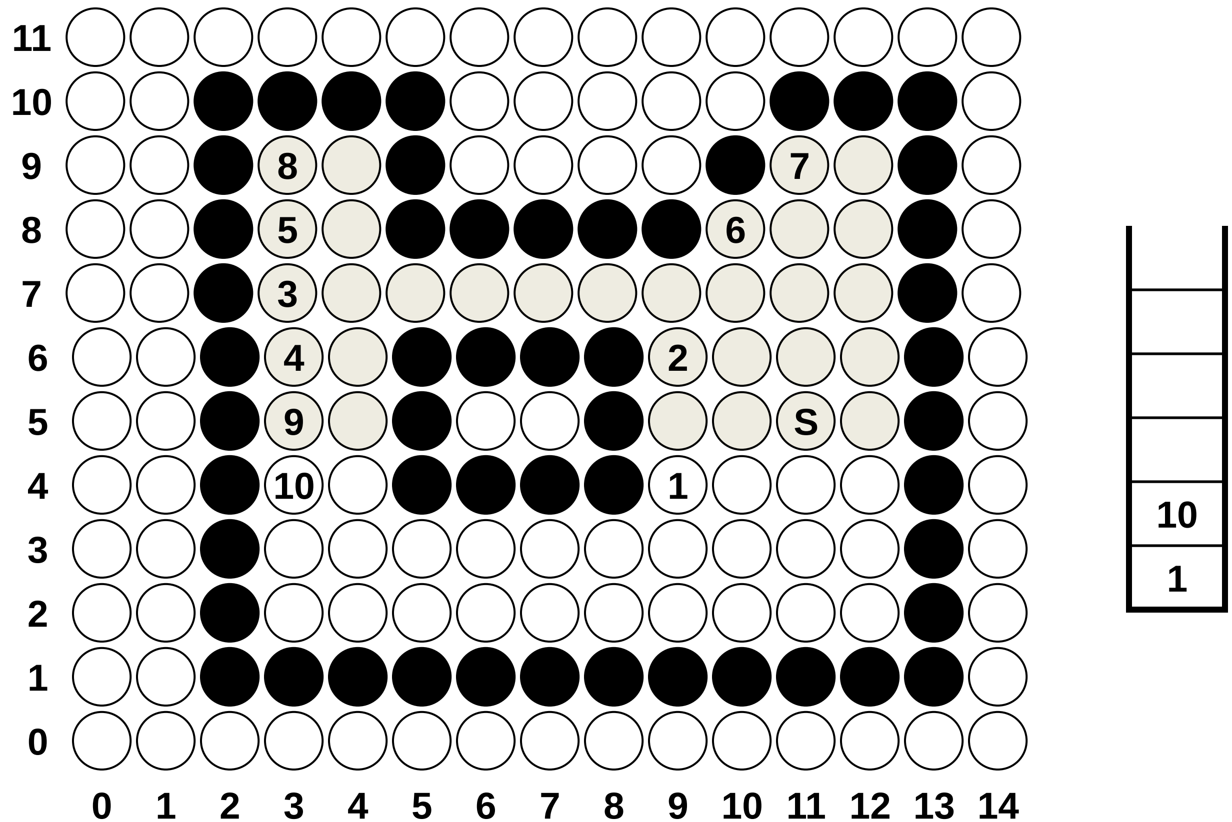


Span Flood-Fill Algorithm (example)

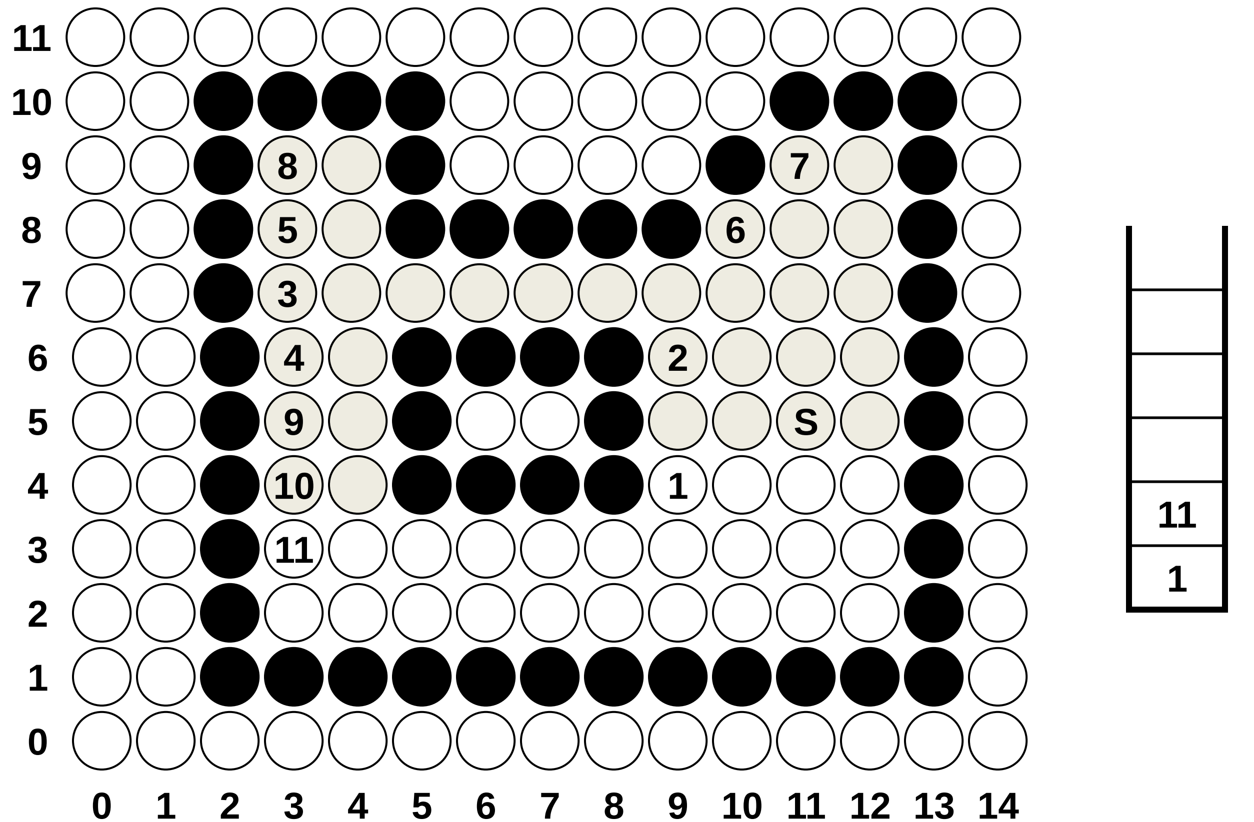




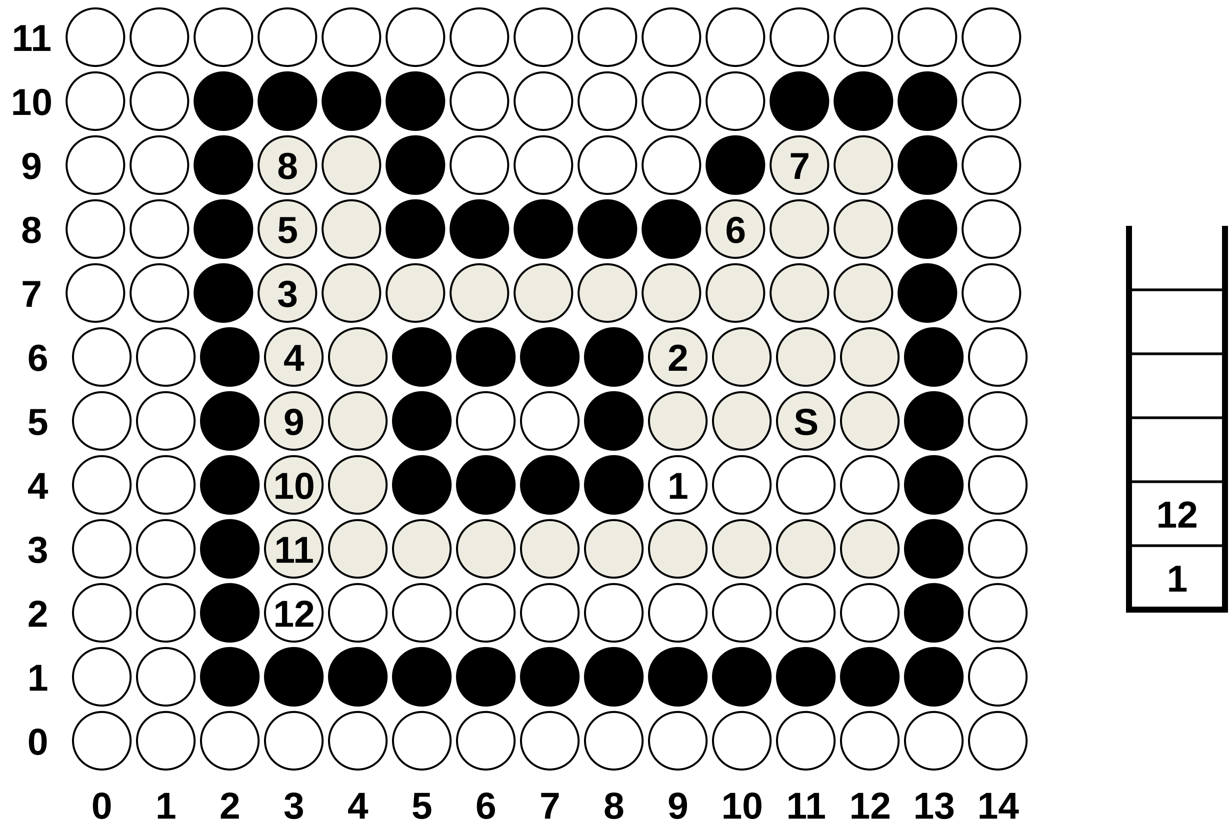
Span Flood-Fill Algorithm (example)



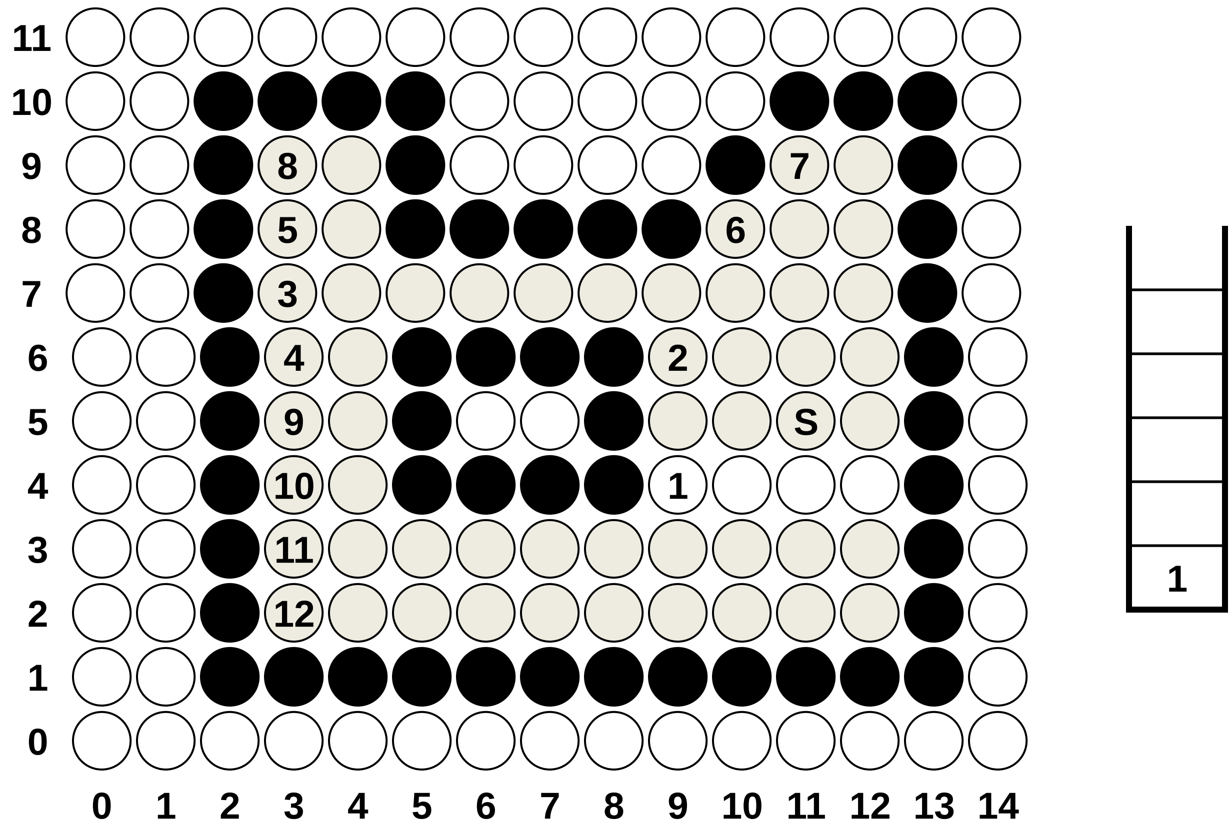
Span Flood-Fill Algorithm (example)



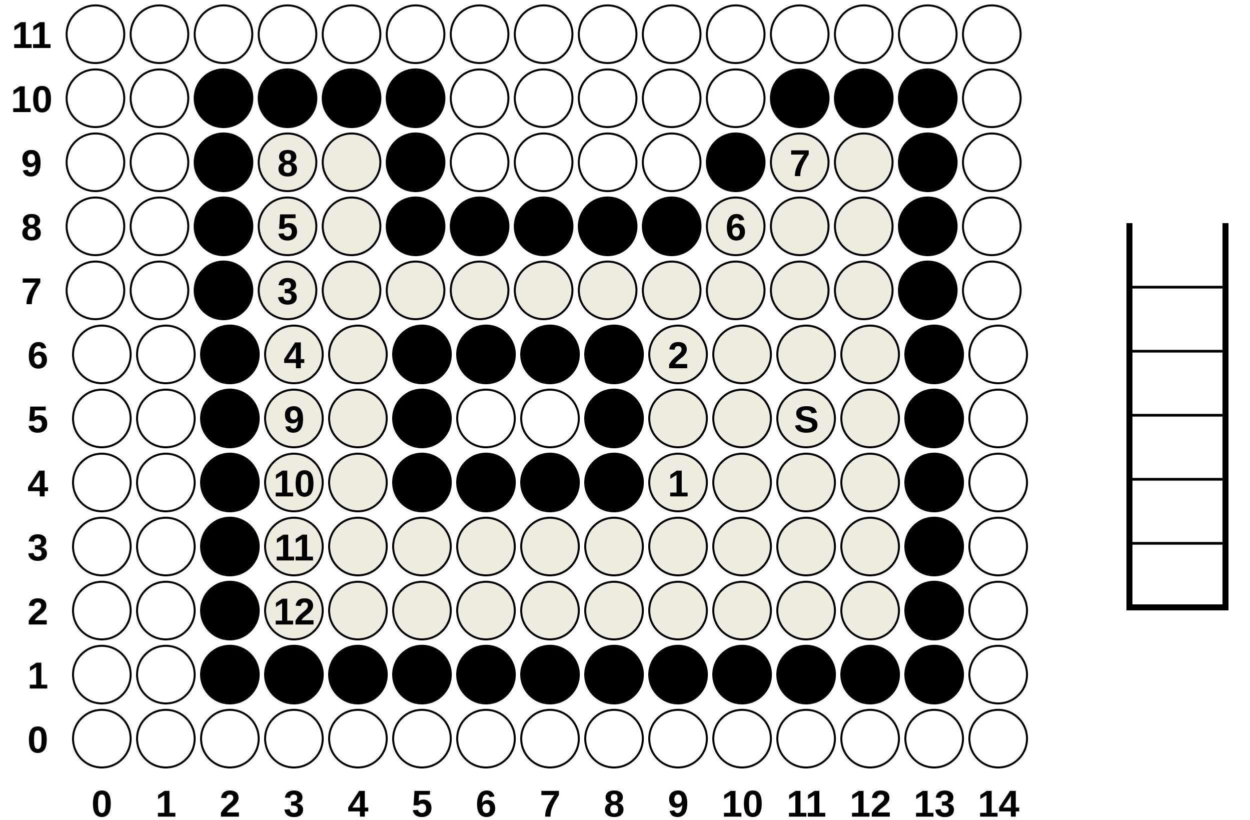
Span Flood-Fill Algorithm (example)



Span Flood-Fill Algorithm (example)



Span Flood-Fill Algorithm (example)



Inside – outside tests

- Area filling needs to identify interior regions of objects
- Odd-even rule/ odd parity rule
- Nonzero winding number rule

Inside-Outside Tests

Odd-Even rule (Odd Parity Rule, Even-Odd Rule):

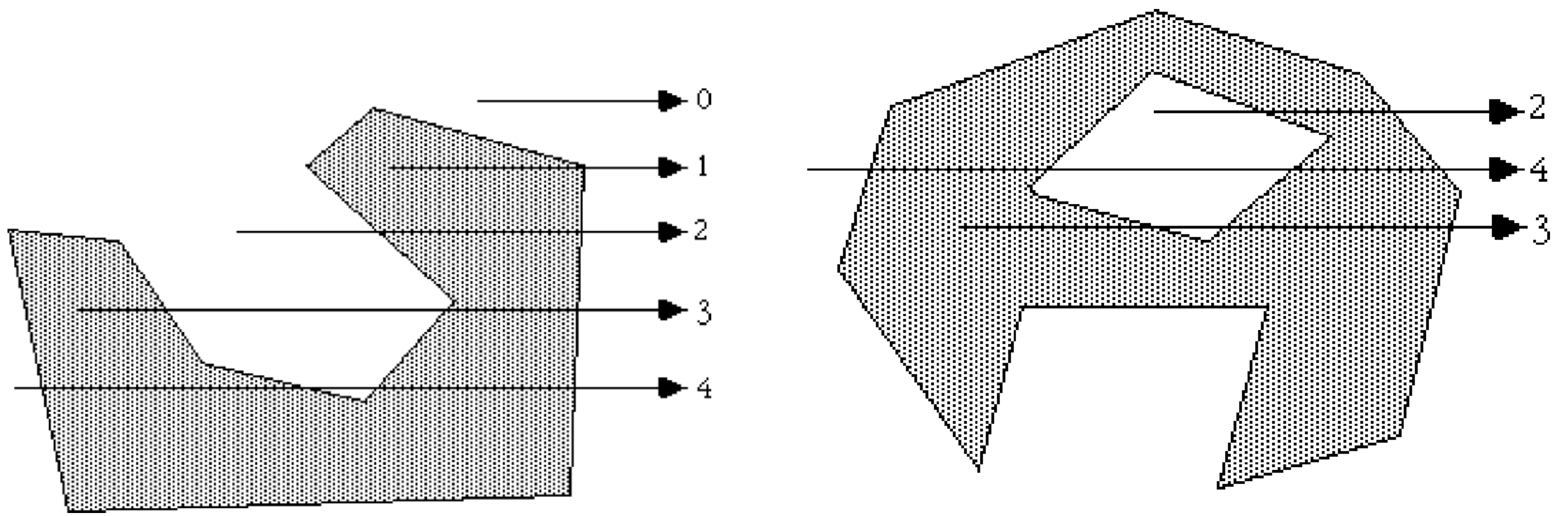
1. draw a line from any position P to a distant point outside the coordinate extents of the object and counting the number of edge crossings along the line.
2. If the number of polygon edges crossed by this line is **odd** then

   P is an **interior** point.

   Else

   P is an **exterior** point

Odd-Even rule



Inside-Outside Tests

Nonzero Winding Number Rule :

- Counts the number of times the polygon edges wind around a particular point in the counterclockwise direction.
- the interior points of a two-dimensional object are defined to be those that have a nonzero value for the winding number.

1. Initializing the winding number to 0.
2. Imagine a line drawn from any position P to a distant point beyond the coordinate extents of the object.

Inside-Outside Tests

Nonzero Winding Number Rule cont:

3. Count the number of edges that cross the line in each direction

   add 1 to the winding number every time you intersect a polygon edge that crosses the line from right to left, and subtract 1 every time you intersect an edge that crosses from left to right.

3. If the winding number is **nonzero**, then

   P is defined to be an **interior** point

   Else

   P is taken to be an **exterior** point**.**