Polygon

Filled Area Primitives

By: Ms. Suhani Chauhan

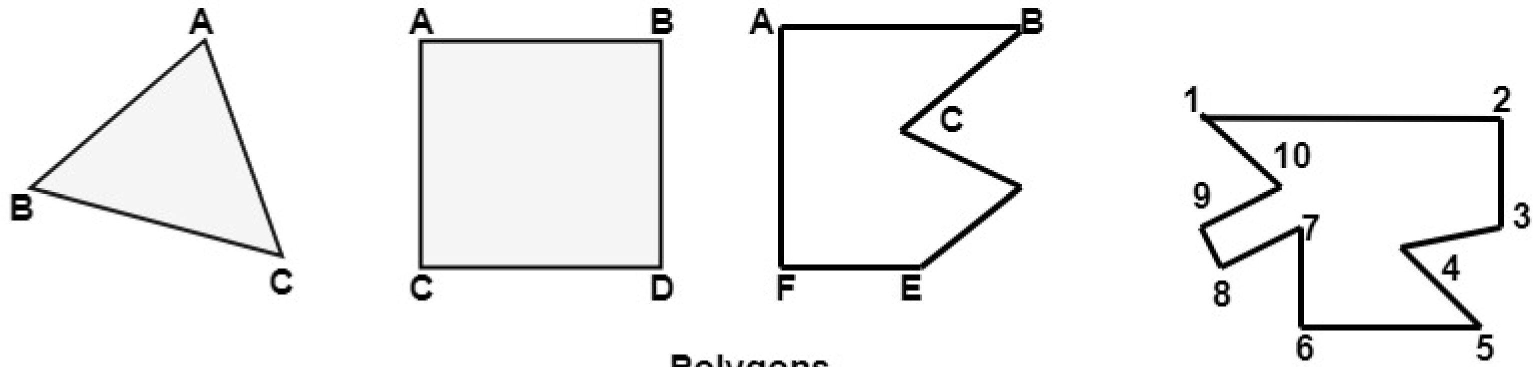
Polygon:

Polygon is a representation of the surface. It is primitive which is closed in nature. It is formed using a collection of lines. It is also called as many-sided figure. The lines combined to form polygon are called sides or edges. The lines are obtained by combining two vertices.

Example of Polygon:

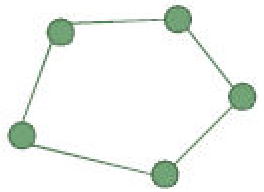
1. Triangle
2. Rectangle
3. Hexagon
4. Pentagon

Following figures shows some polygons.



Polygons

Polygon:



Polygon is a word derived from two Greek words poly and Gon. Poly means multi and Gon means angle. Thus, the meaning of polygon is multi angled figure.

Polygon is a closed figure with many vertices and edges and at each vertex exactly two edges meet and no edge crosses other.

The line segments which make up a polygon boundary are called edges. The end points of the edges are called vertex of polygon.

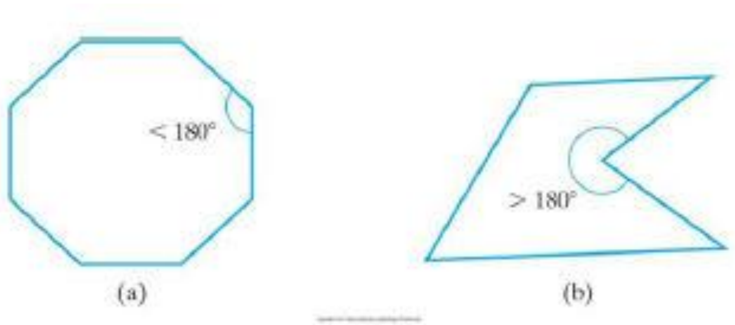
The simple form of possible polygon is triangle having 3 edges and 3 vertex points.

A plane figure specified by a set of three or more vertices, that are connected in sequence by straight-line segments (edges). Here refer only to those without crossing edges: simple (standard) polygon

Polygon Classifications

Convex Polygon (凸) : All interior angles are less than or equal to 180o degrees. This polygon is a polygon in which if you take any two points of polygon then all the points on the line segment joining these two points fall within the polygon itself.

Concave Polygon (凹): Greater than 180o degrees. This polygon is a polygon in which if the line joining any two points of a polygon does not fall entirely within the polygon.



A convex polygon (a) and a concave polygon (b)

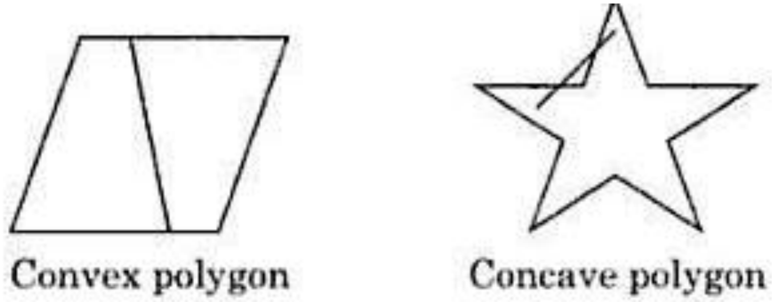


FIGURE 4.3 Classes of polygons.

Polygon Representation

The polygon can be represented by listing its n vertices in an ordered list.

P = {(x1, y1), (x2, y2), ……., (xn, yn)}.

The polygon can be displayed by drawing a line between (x1, y1), and (x2, y2), then a line between (x2, y2), and (x3, y3), and so on until the end vertex. In order to close up the polygon, a line between (xn, yn), and (x1, y1) must be drawn.

For objects described by many polygons with many vertices, this can be a time consuming process.

One method for reducing the computational time is to represent the polygon by the (absolute) location of its first vertex, and represent subsequent vertices as relative positions from the previous vertex. This enables us to translate the polygon simply by changing the coordinates of the first vertex.

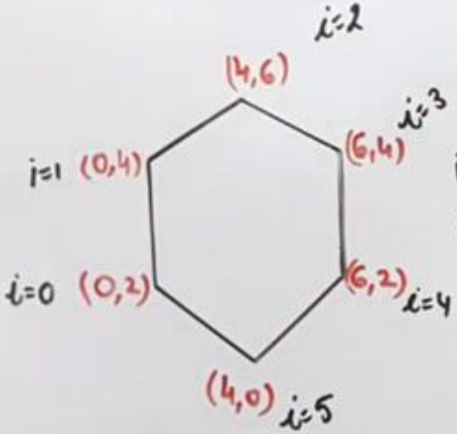
Polygon Representation

There are three approaches to represent polygons in graphics system:

1. Polygon Drawing Primitive Approach: They can directly draw the polygon shapes. They are regular polygon with default size and angle.
2. Trapezoid Primitive Approach: In such devices trapezoids are formed from two scan lines and two line segments.
3. Line and Point Approach: In this display, file cannot be stored only with series of line commands because they do not specify how many of the lines are part of polygon.

   New command is used in that display file. The opcode for new command itself specify the no. of line segment in the polygon.

POLYGON AREA FILLING :- LINE & POINT APPROACH — ENTERING POLYGON



	DF_OP	DF_x	DF_y
i=0	6	0	2
i=1	2	0	4
	2	4	6
	2	6	4
	2	6	2
	2	4	0
	2	0	2

ALGORITHM:

```
1) READ 'Ax' & 'AY' OF LENGTH 'N':
   // Ax & AY ARE ARRAY CONTAINING VERTICES &
   N IS NO. OF SIDES OF POLYGON.
                                        N=6

2) i=0 // INITIALIZE COUNTER TO COUNT NO. OF SIDES
   DF_OP[i] ← N
   DF_x[i] ← Ax[i]
   DF_y[i] ← AY[i]
   i ← i+1      [i=1]

3) do {
      DF_OP[i] ← 2
      DF_x[i] ← Ax[i]
      DF_Y[i] ← AY[i]
      i ← i+1
   } WHILE (i<N) [ENTER LINE
                  COMMAND]

4) DF_OP[i] ← 2
   DF_x[i] ← Ax[0]
   DF_Y[i] ← AY[0]
   // ENTER LAST LINE COMMAND

5) STOP
```

Inside-Outside Tests

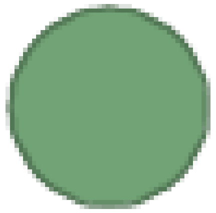
Inside-Outside test

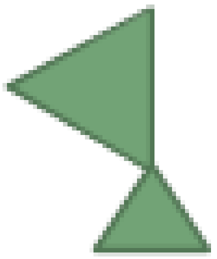
Area-filling algorithms and other graphics processes often need to identify interior regions of objects. For simple object, it is a straightforward process.

For complex objects, graphics packages normally use either:

1. Odd-Even rule (Odd-Parity rule or Even-Odd rule)

2. winding number method





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

1. Draw a line: (the line path doesn’t intersect any line-segment endpoints) From any position P to a distant point outside the coordinate extents of the polygon.

2. Counting the number of edge crossing along the line.

3. If the number of polygon edges crossed by this line is odd then P is an interior point.

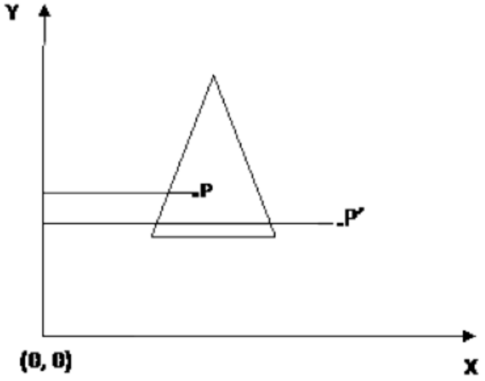
Else P is an exterior point

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

Let P be a point to be tested. From P draw a line parallel to the X axes as shown in the above figure. Count the no. of intersection points with edges of polygon.

If the no. of intersection points is odd then the point is inside the polygon. For P there is one intersection point. Therefore P is inside.

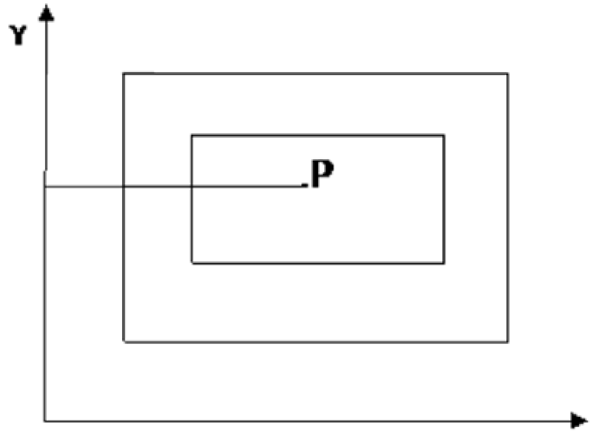
If the no. of intersection points is even then the point is outside the polygon. For P' there is two intersection points. Therefore P' is outside.



Circumstances where Even-Odd method fails

Overlapping Polygon:

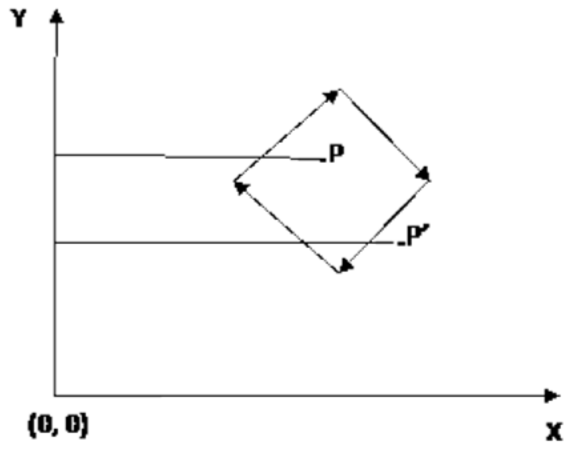
As shown in the figure, for point P numbers of intersection are even. Thus according to Even – Odd method the point P must be outside the polygon. But this point is inside the polygon so in such a condition where a polygon is complex or overlapped then the Even – Odd method will not work. For that the winding no. method is used.



Winding Number method:

In this method, give direction to each and every edge as shown in following figure. Check the direction of edges and number of edges intersected by the straight line parallel to X axes. If the direction is upward, count –1 and if the direction is downwards then count is +1.

No. of winding is not equal to zero than point is inside or if it is equals to 0 then the point is outside the polygon.



Winding Number method:

For the figure shown below for point p summation of winding no. is,

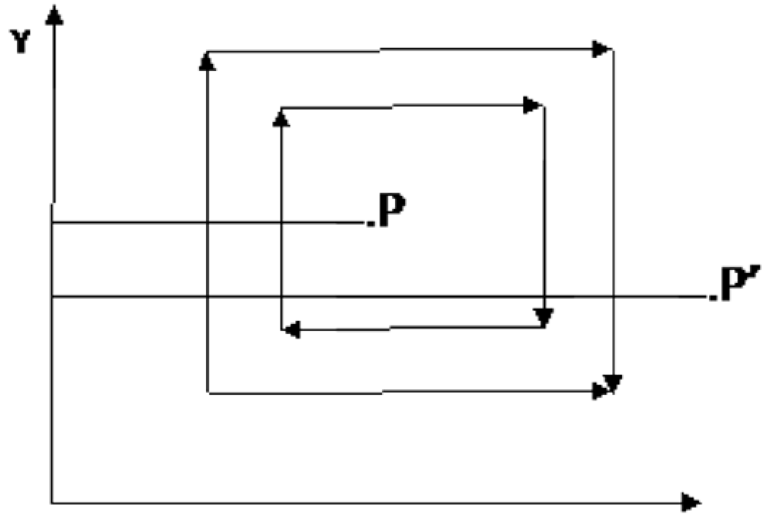
For Point P,

Σ winding no. = -1 -1 = -2 , that is not equal to zero so point is inside the polygon.

For Point P’,

Σ winding no. = -1 -1 +1 +1 = 0 ,

that is equal to zero so point is outside the polygon



Polygon Area Filling

To make the polygon figure to look like a solid object, it is required to be fill in with some color or pattern polygon area filling means changing the color of pixel belonging to the polygon. This can be done by different methods which are used popularly for polygon filling.

There are Two different methods

1. Seed Fill Method

    Boundary fill method

    Flood fill method

2. Scan Line Fill Method

    Edge list method

  Edge fill method

  Fence fill method

Boundary Fill Algorithm

This approach to area filling is to 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.

A boundary-fill procedure accepts as input the coordinate of the interior point (x, y), a fill color, and a boundary color.

The following steps illustrate the idea of the recursive boundary-fill algorithm:

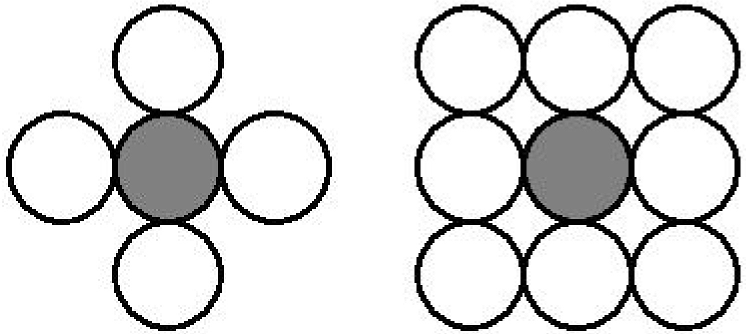
1. Start from an interior point.

2. 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.

3. Select the next pixel from the stack, and continue with step 2.

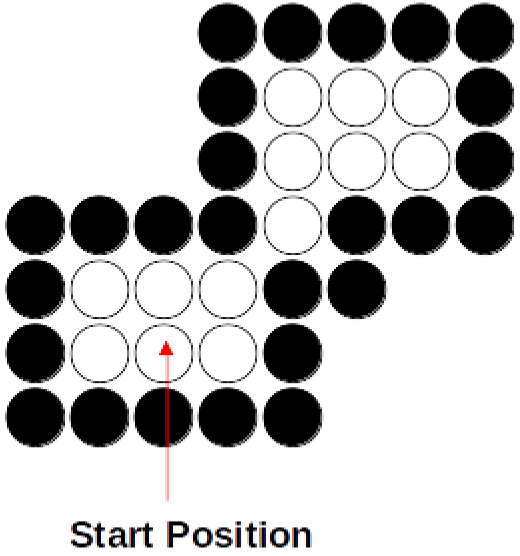
Boundary Fill Algorithm

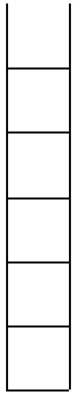
The order of pixels that should be added to stack using 4-connected is above, below, left, and right. For 8-connected is above, below, left, right, above-left, above-right, below-left, and below-right.

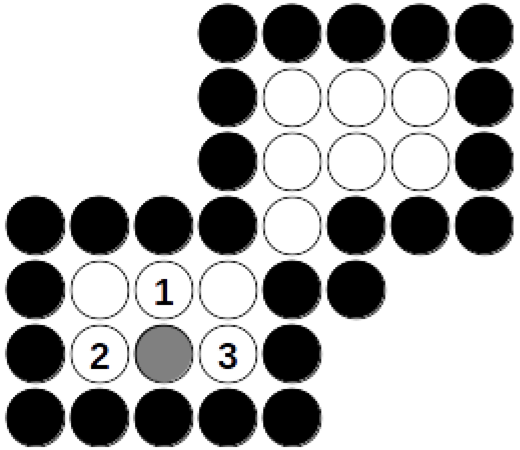
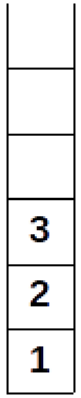


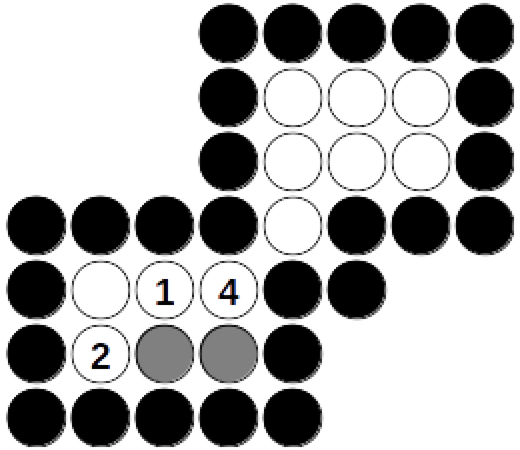
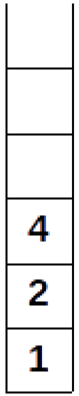
```
Void fill4 (int x, int y, int fill, int boundary)
{
 int current;
current= getpixel(x, y);
if (( current != boundary) && (current!= fill)
{setcolor (fill);
setpixel(x,y);
fill4 (x+1, y, fill, boundary);
fill4 (x-1, y, fill, boundary);
fill4 (x, y+1, fill, boundary);
fill4 (x, y-1, fill, boundary);
}}
```

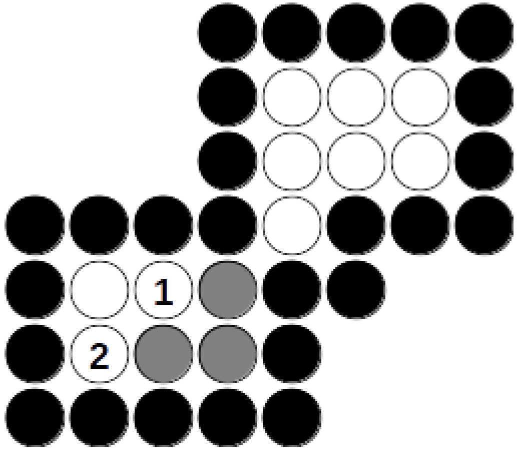
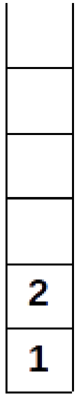
Boundary Fill Algorithm: 4-connected (Example)

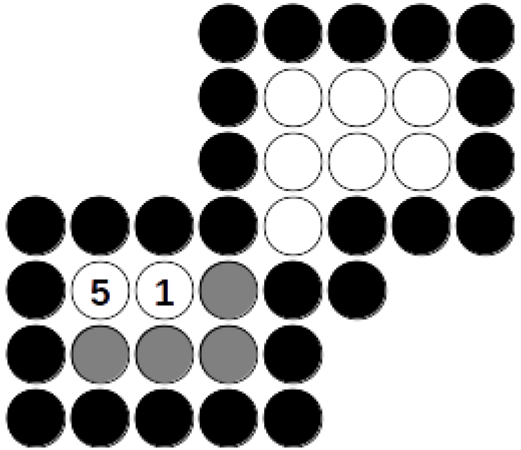
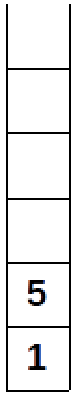


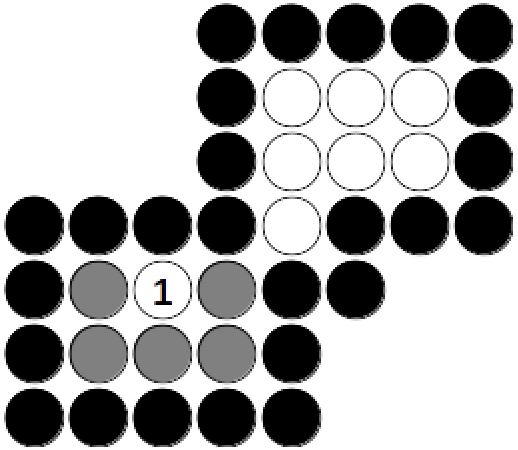


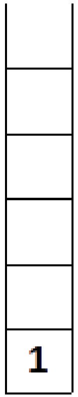

1
2
3

3
2
1

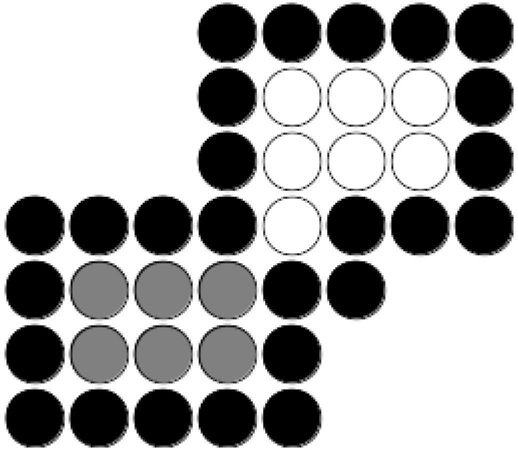
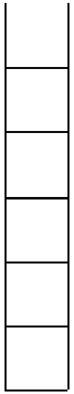

1
4
2

4
2
1


1
2

2
1

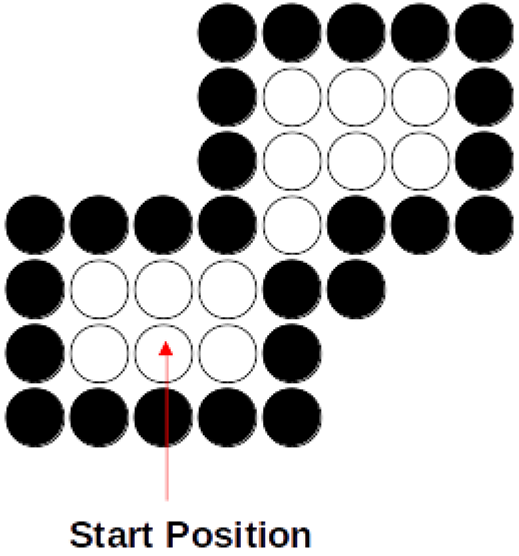

5
1

5
1

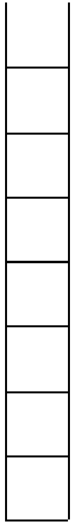

1

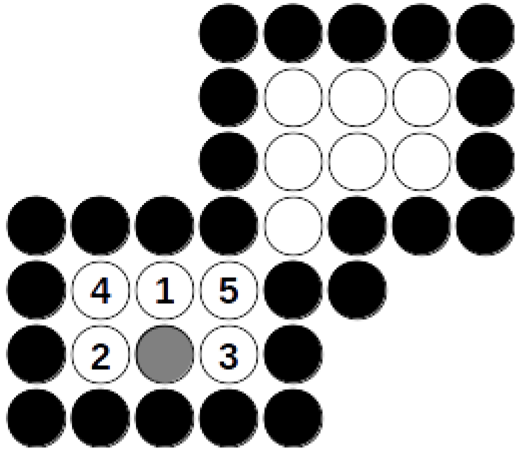
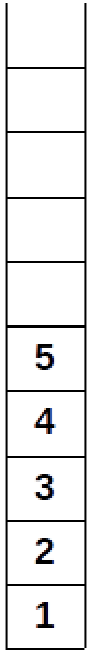

1

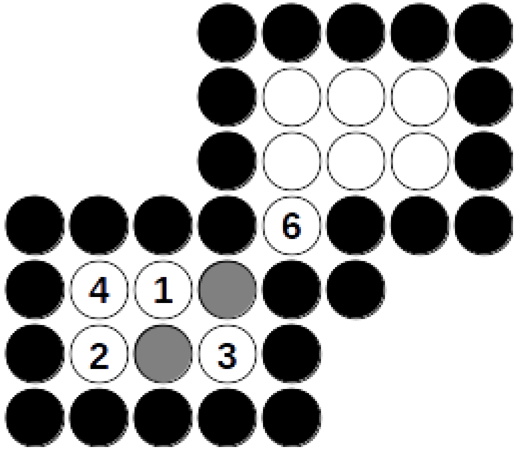
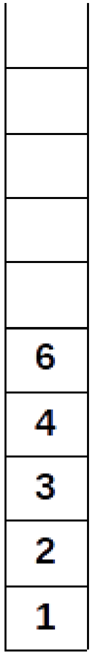



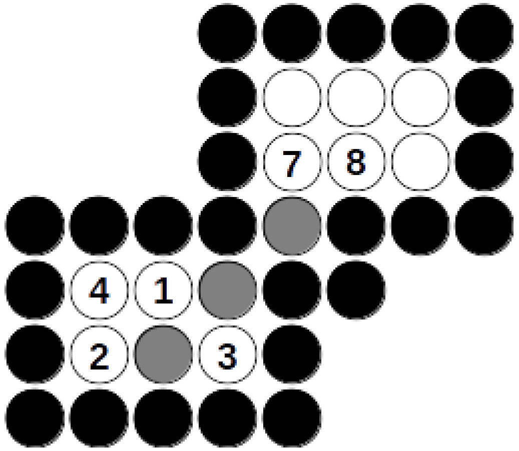
Boundary Fill Algorithm: 8-connected (Example)

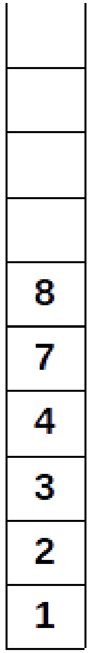


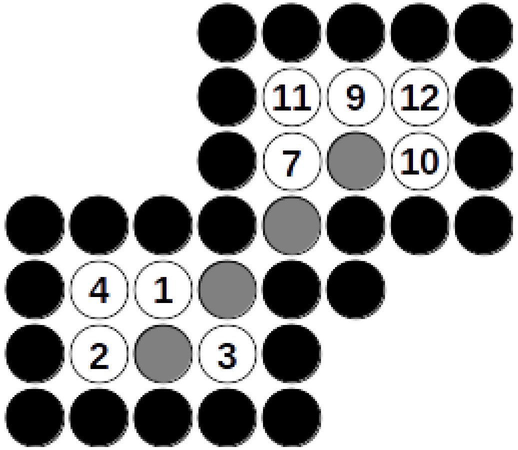


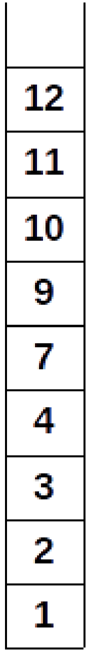

4
1
5
2
3

5
4
3
2
1

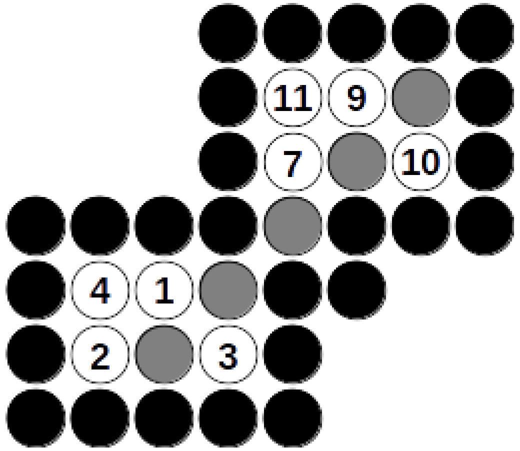

6
4
1
2
3

6
4
3
2
1

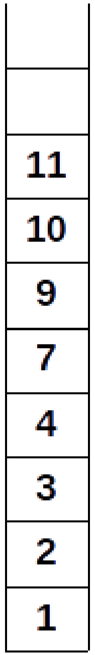

7
8
4
1
2
3

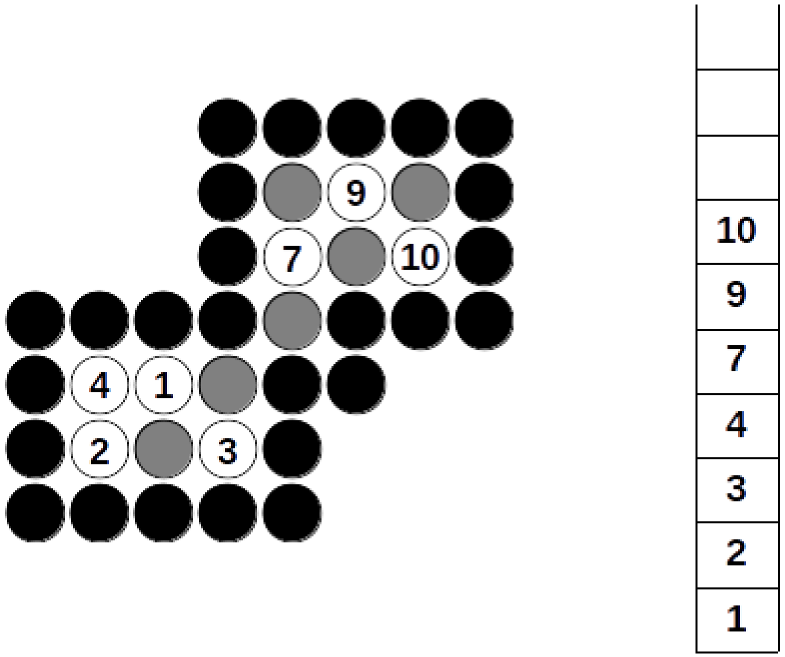

8
7
4
3
2
1

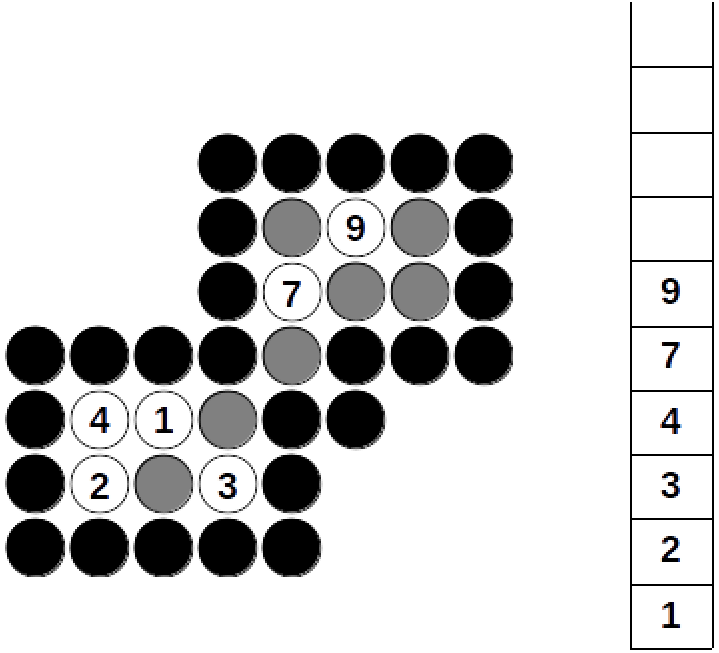

11
9
12
7
10
4
1
2
3

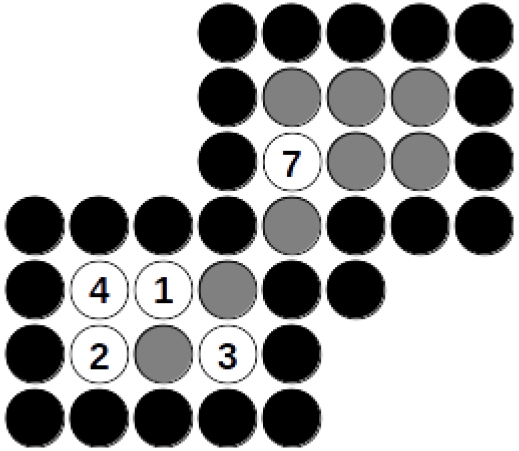

12
11
10
9
7
4
3
2
1

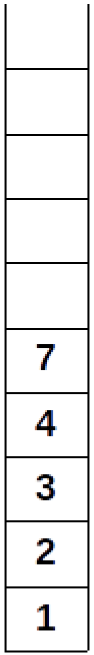

11
9
7
10
4
1
2
3

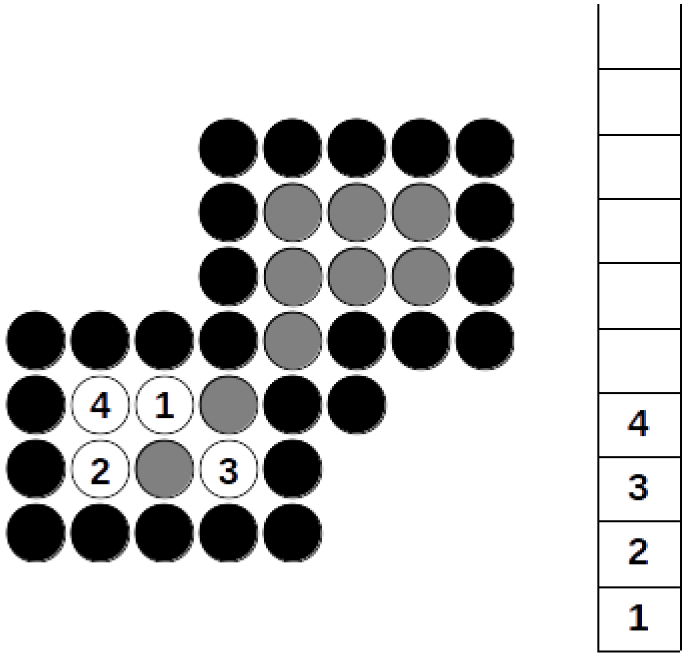

11
10
9
7
4
3
2
1

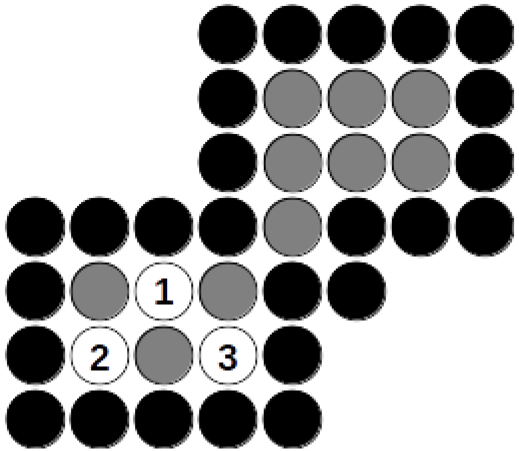
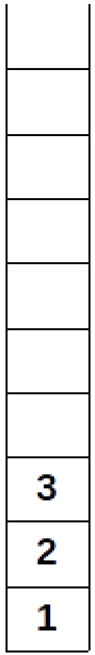

9
7
10
4
1
2
3
10
9
7
4
3
2
1

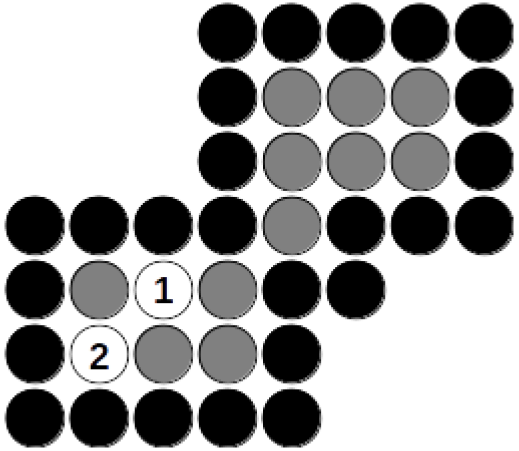

9
7
4
1
2
3
9
7
4
3
2
1

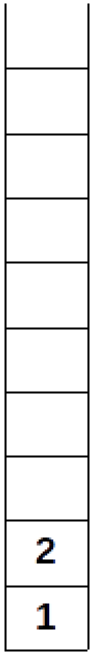

4
1
2
3
7

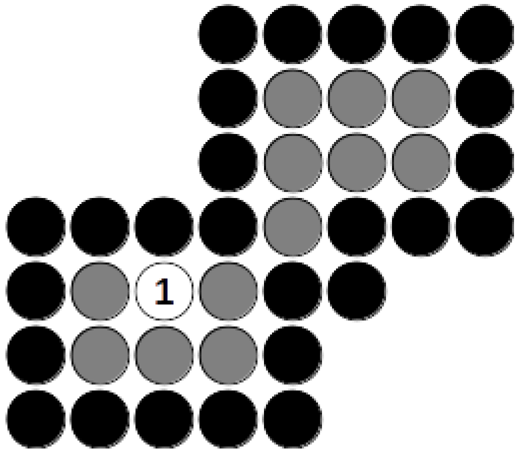
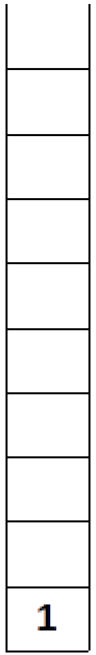

7
4
3
2
1

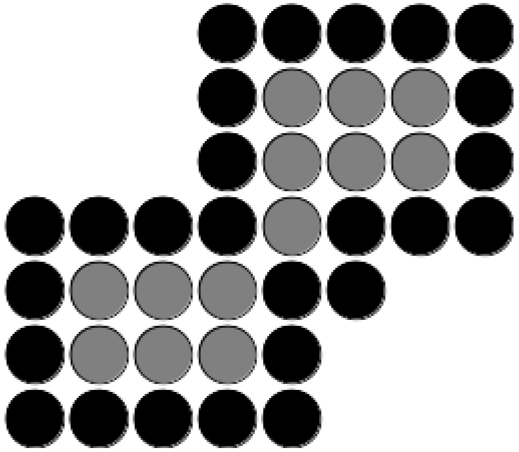
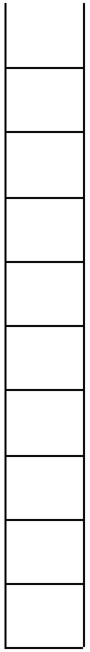

4
1
2
3
4
3
2
1


1
2
3

3
2
1


1
2


2
1


1

1

Boundary Fill Algorithm

Since the previous procedure requires considerable stacking of neighboring pixels, more efficient methods are generally employed.

These methods (Span Flood-Fill) fill horizontal pixel spans across scan lines, instead of proceeding to 4-connected or 8-connected neighboring pixels.

Then we 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

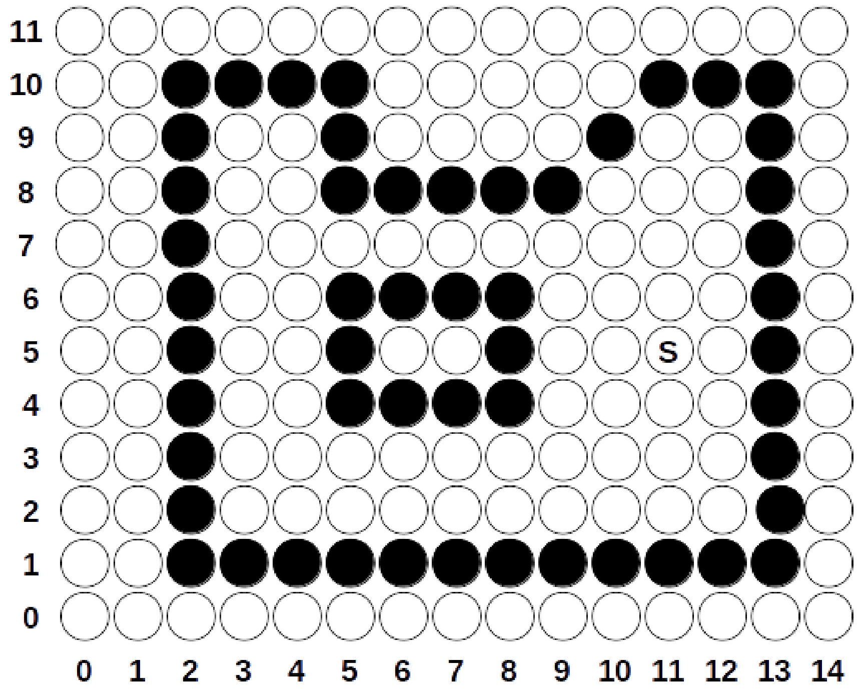
The algorithm is summarized as follows:

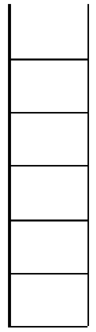
Starting from the initial interior pixel, then fill in the contiguous span of pixels on this starting scan line.

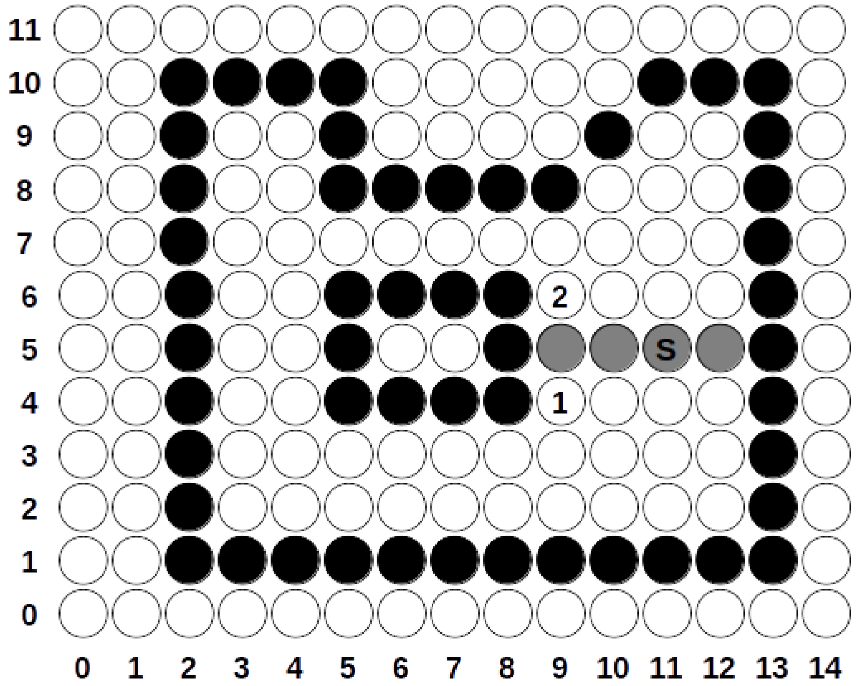
Then locate and stack starting positions for spans on the adjacent scan lines, where spans are defined as the contiguous horizontal string of positions bounded by pixels displayed in the area border color.

At each subsequent step, unstack the next start position and repeat the process.

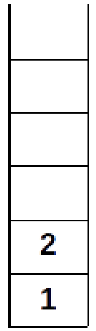
Span Flood-Fill Algorithm (example)



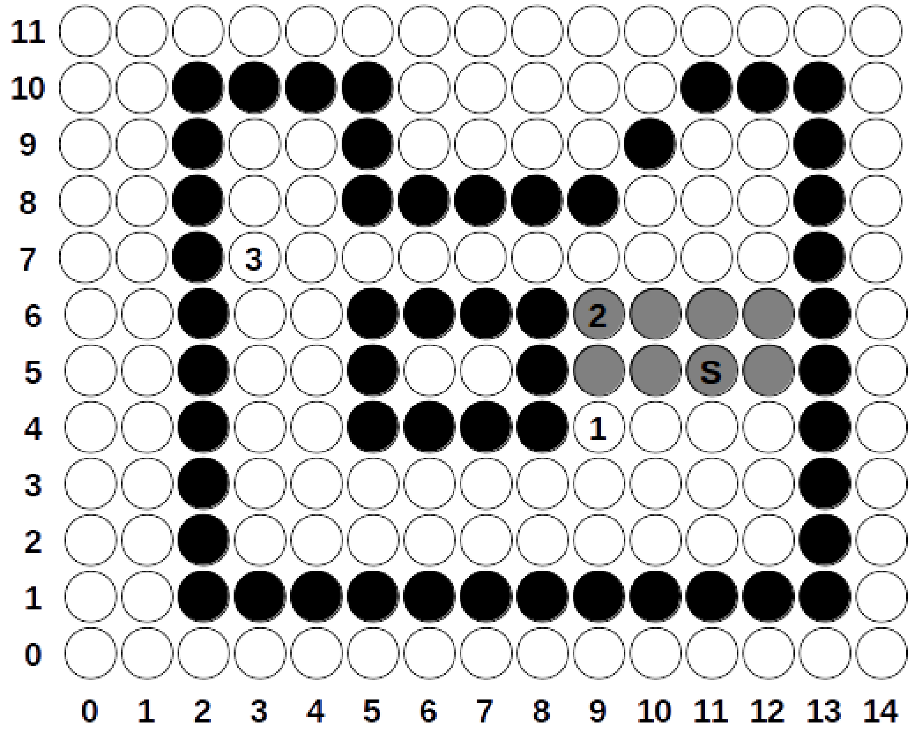




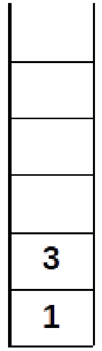
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
2
S
1



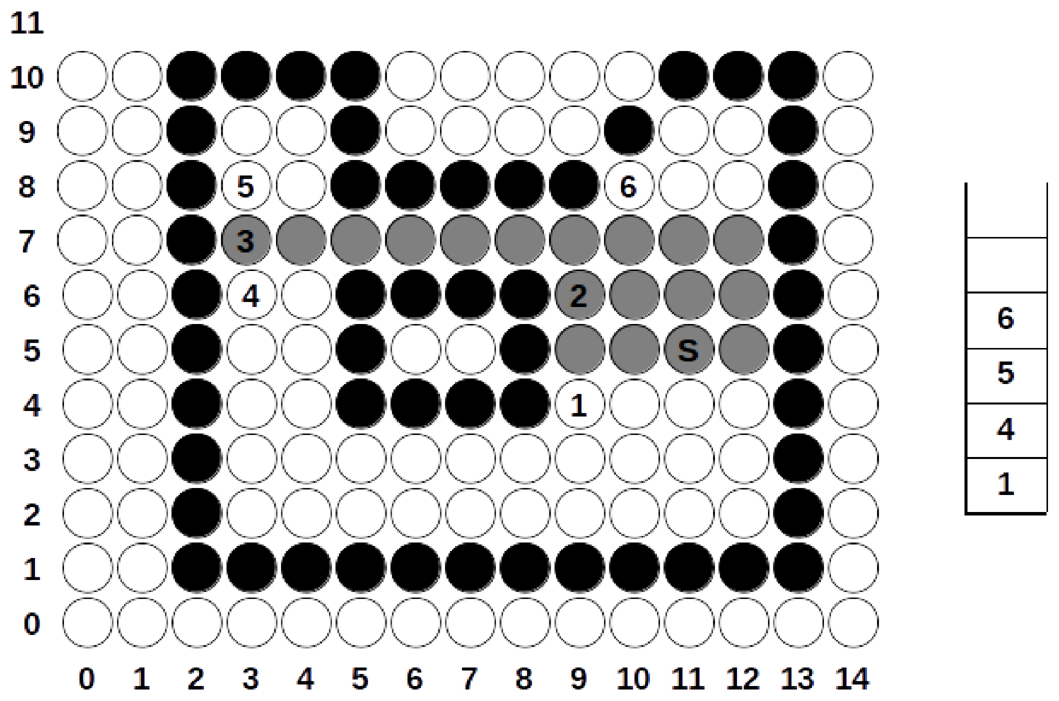
2
1

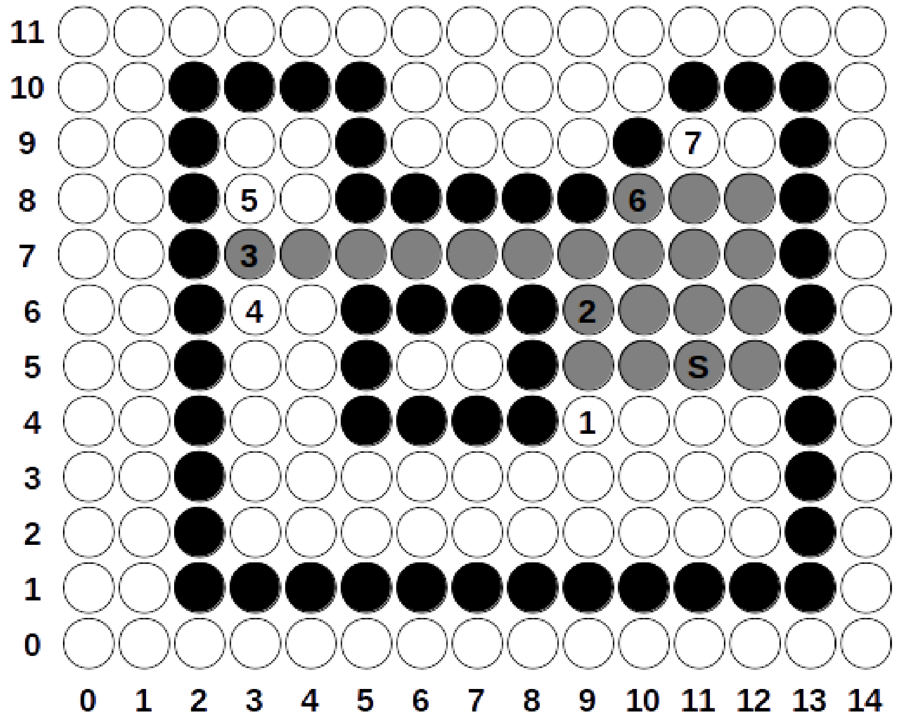


11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
3
2
S
1

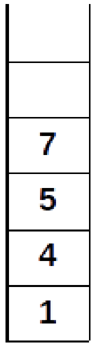


3
1

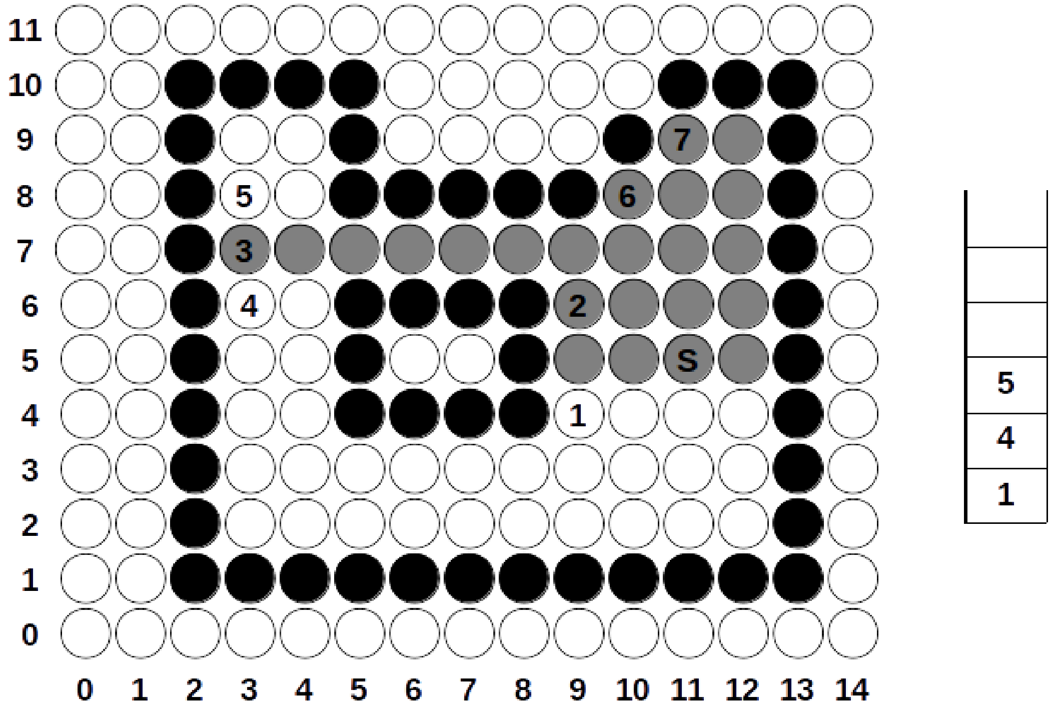

11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
5
6
3
4
2
S
1
6
5
4
1

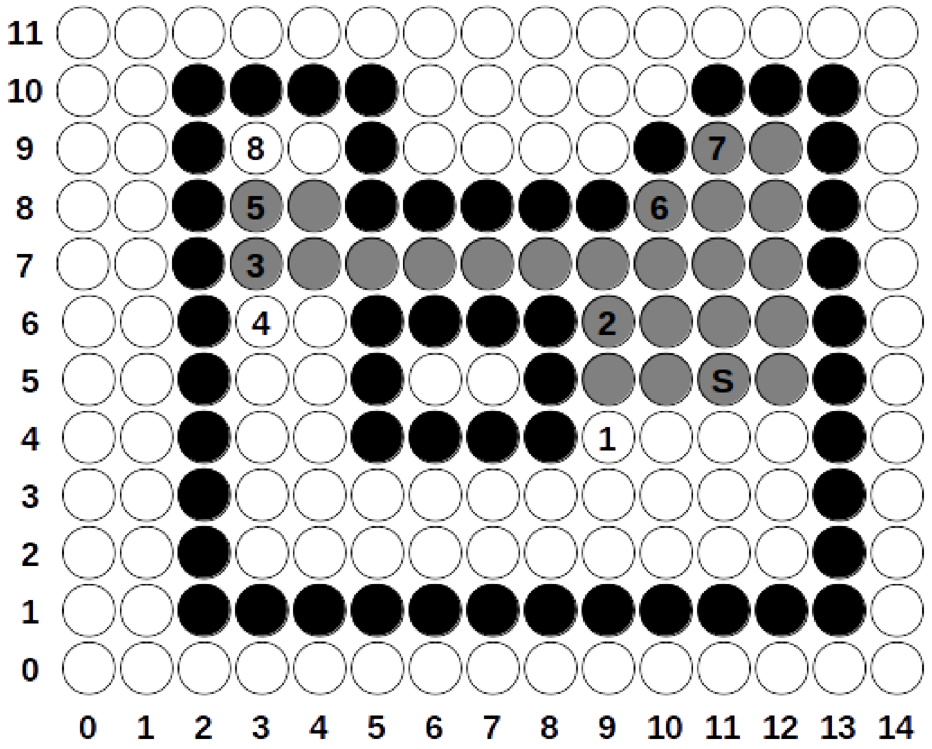


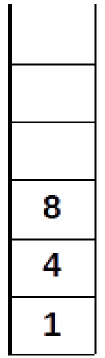
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
S
1
2
3
4
5
6
7

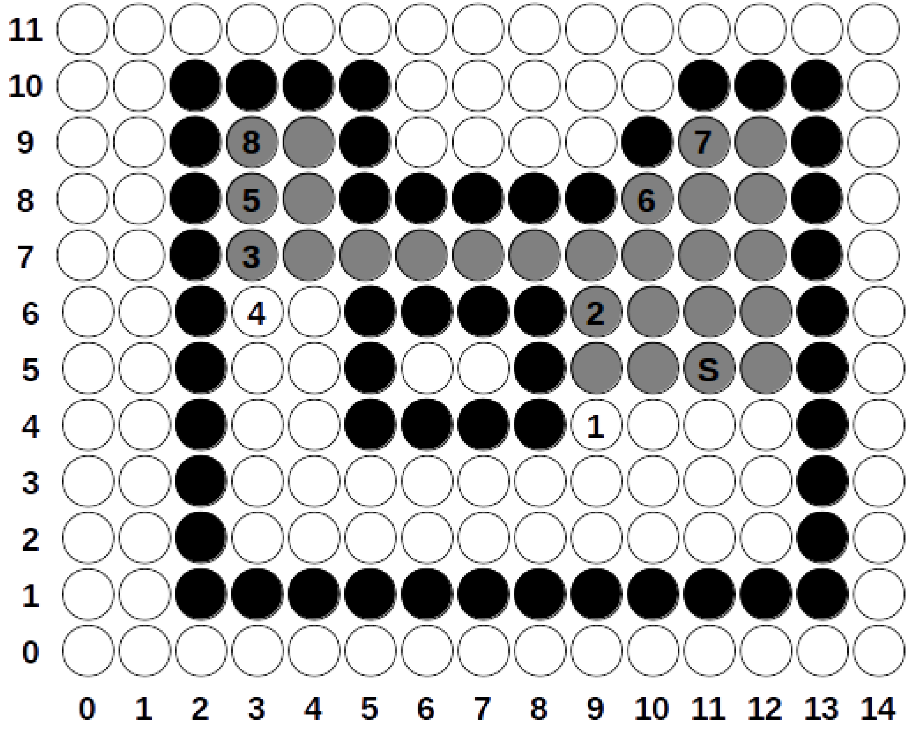


7
5
4
1

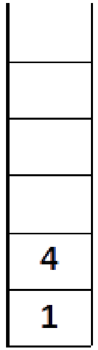

S
1
2
3
4
5
6
7
5
4
1


11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
8
7
5
6
3
4
2
S
1

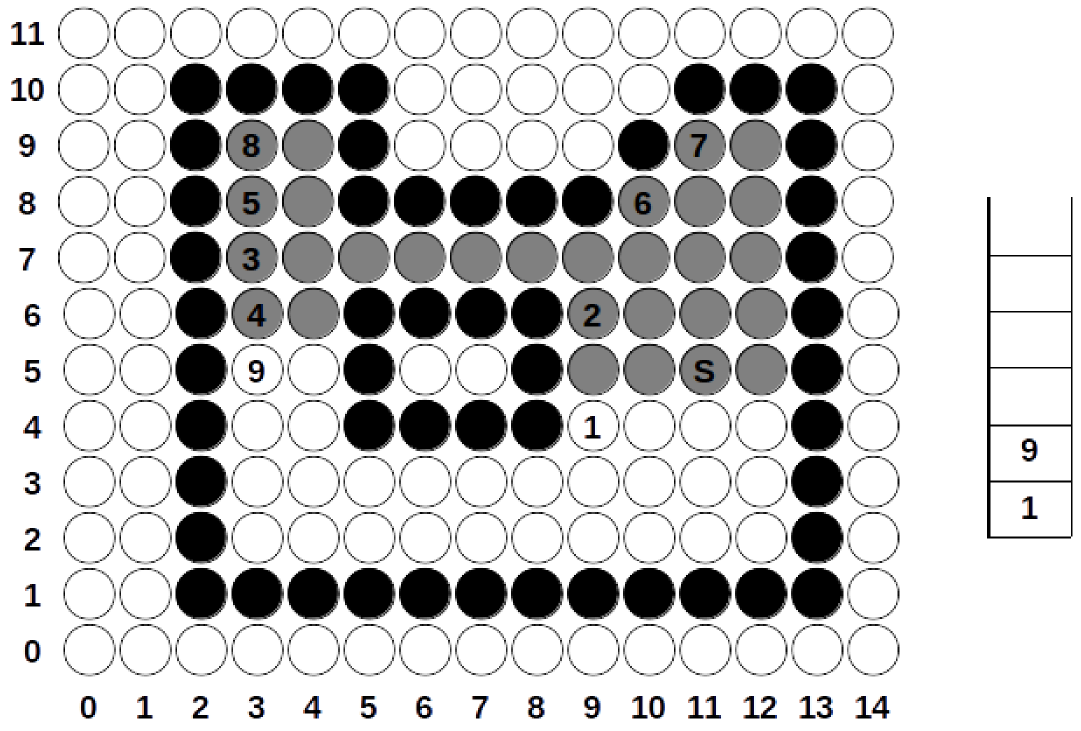

8
4
1



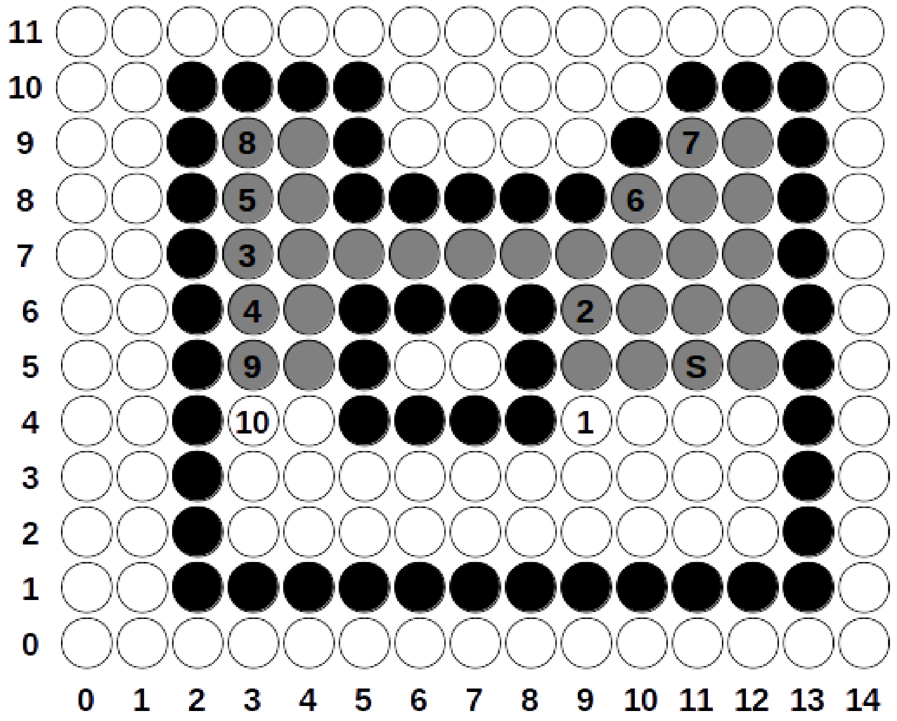
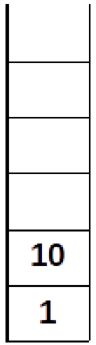
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
8
7
5
6
3
4
2
S
1

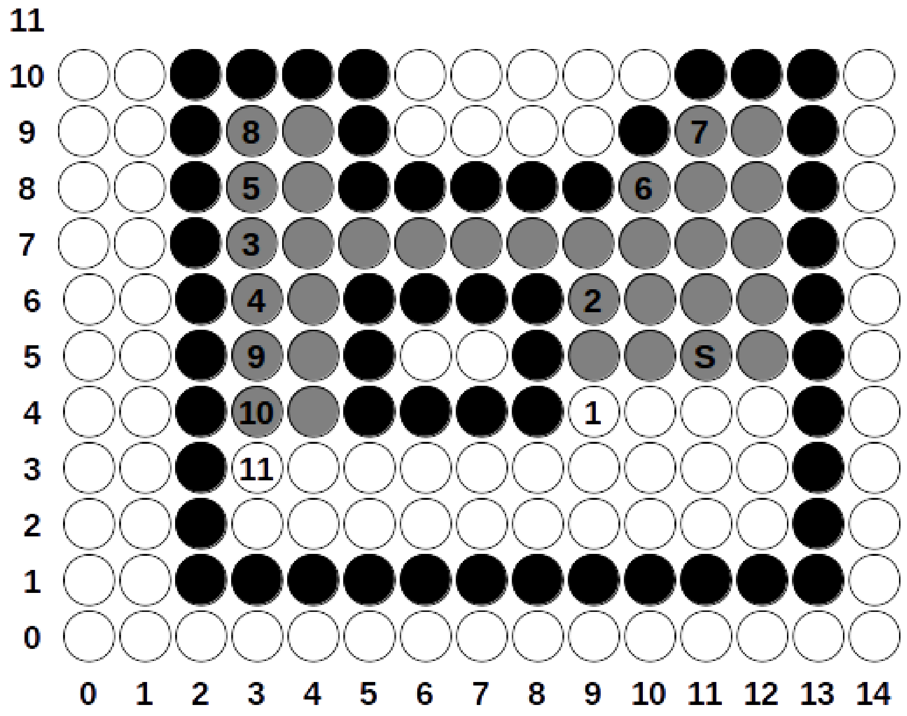


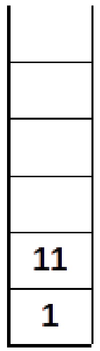
4
1

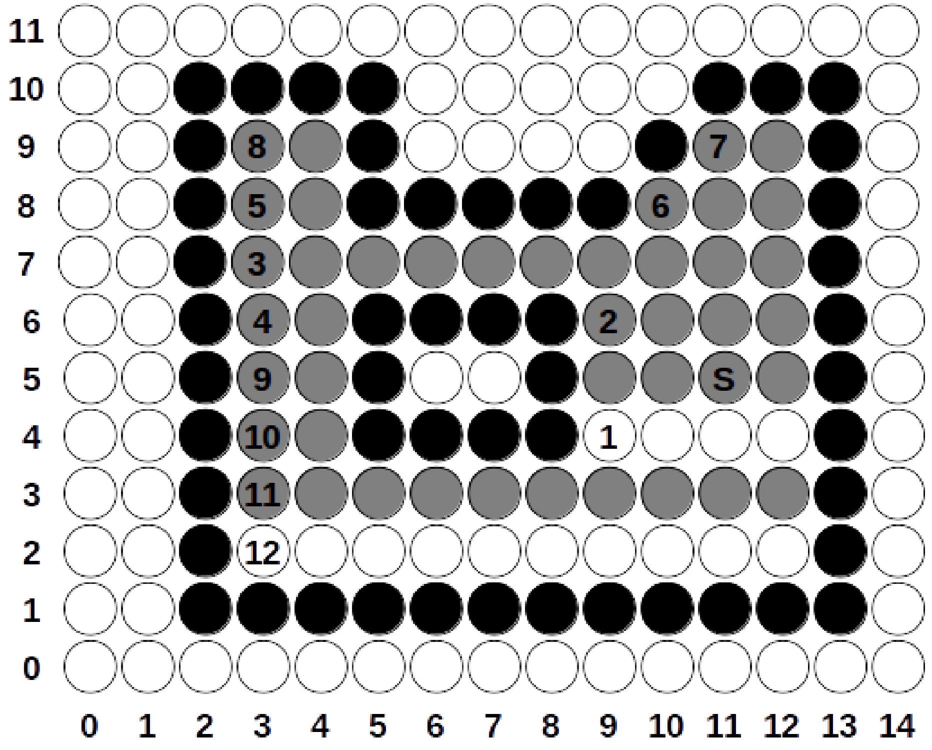


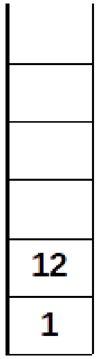
11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
8
7
5
6
3
4
2
9
S
1
9
1

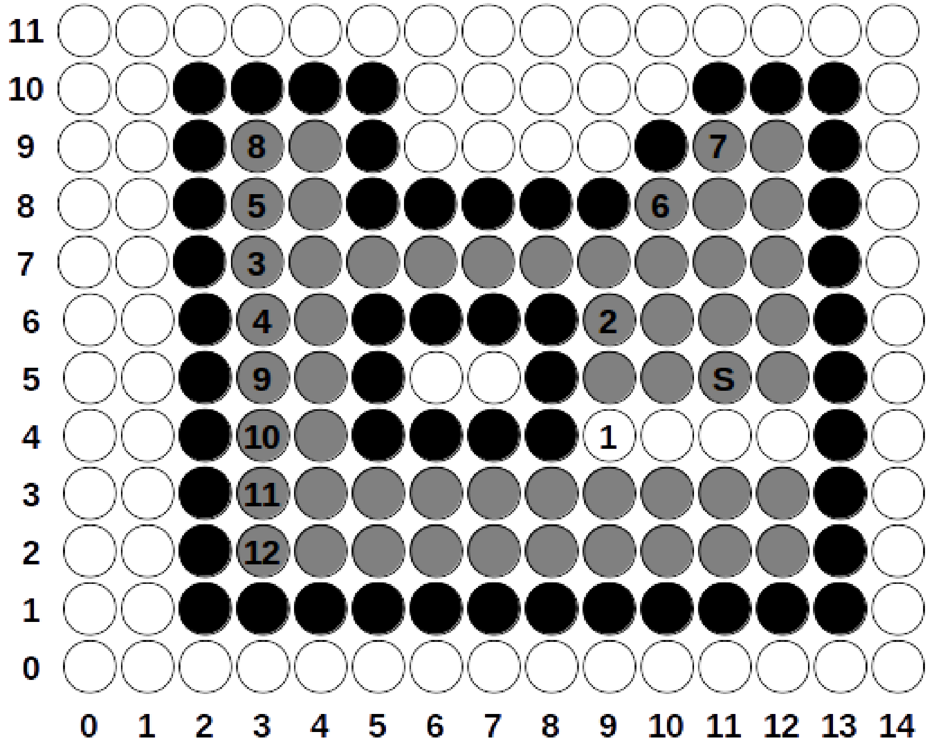

11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
8
7
5
6
3
4
2
9
S
10
1

10
1


11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
8
7
5
6
3
4
2
9
S
10
1
11

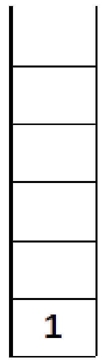

11
1

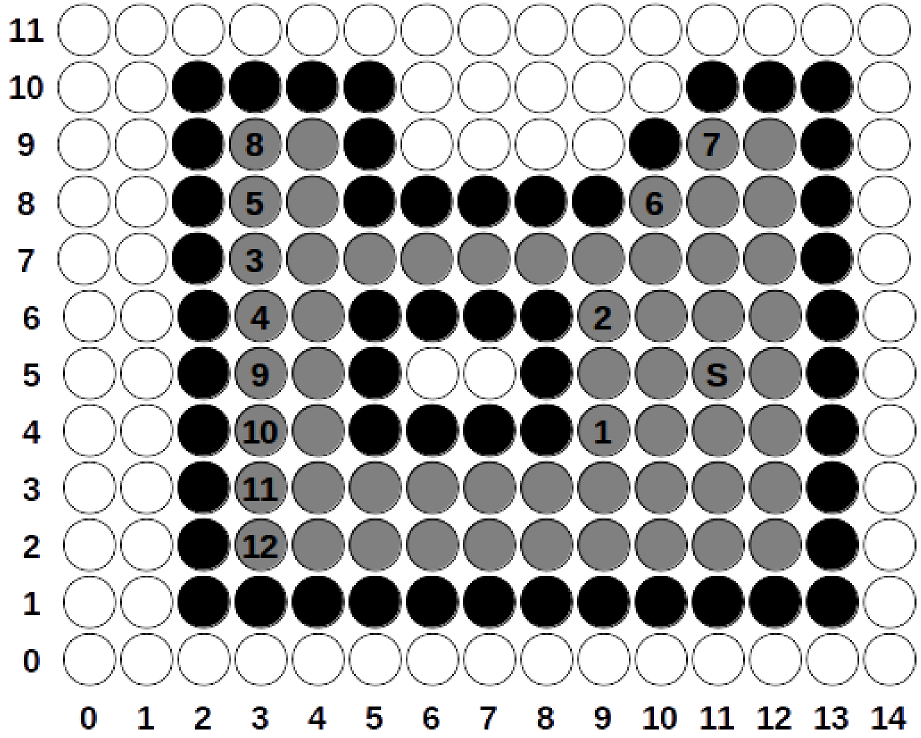

11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
8
7
5
6
3
4
2
9
S
10
1
11
12

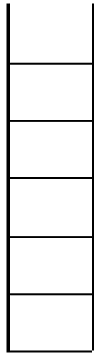

12
1



S
1
2
3
4
5
6
7
8
9
10
11
12


1


11
10
9
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
8
7
5
6
3
4
2
9
S
10
1
11
12

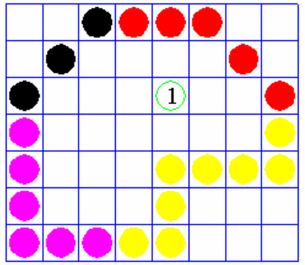


Flood Fill Method (seed fill method):

Sometimes we want to fill in (recolor) an area that is not defined within a single color boundary. We paint such areas by replacing a specified interior color instead of searching for a boundary color value. This approach is called a flood-fill algorithm.

It is also known as seed fill method. In this method, a point inside a polygon is needed. This point is called as seed point and the process of filling the polygon area is started from this points.

From the seed point, a point is taken and interior color is replaced by the required color. This process is done till a pixel of boundary color is reached. This process is performed simultaneously in all directions.



This can be done in two ways:

Four Connected Point Neighborhood Method

Eight Connected Point Neighborhood Method

In first method, four surrounded pixels are taken simultaneously and replaced with the required color. In second method, eight surrounded pixels are taken simultaneously and replaced with the required color. Obviously, recursive function is the best suited concept for the flood fill algorithm.

```
Void fill4 (int x, int y, int fill, int oldcolor) {

If( getpixel (x, y) == oldcolor)

{

Putpixel (x, y, fill);

fill4 (x+1, y, fill, oldcolor);

fill4 (x-1, y, fill, oldcolor);

fill4 (x, y+1, fill, oldcolor);

fill4 (x, y-1, fill, oldcolor);

}
```

If we do not have a seed point to fill the polygon, we can turn on all the pixels which are inside the polygon but the problem is that, these needs inside test to be perform with each and every pixel on the screen which is very time consuming.

Difference between flood fill and boundary fill algorithm

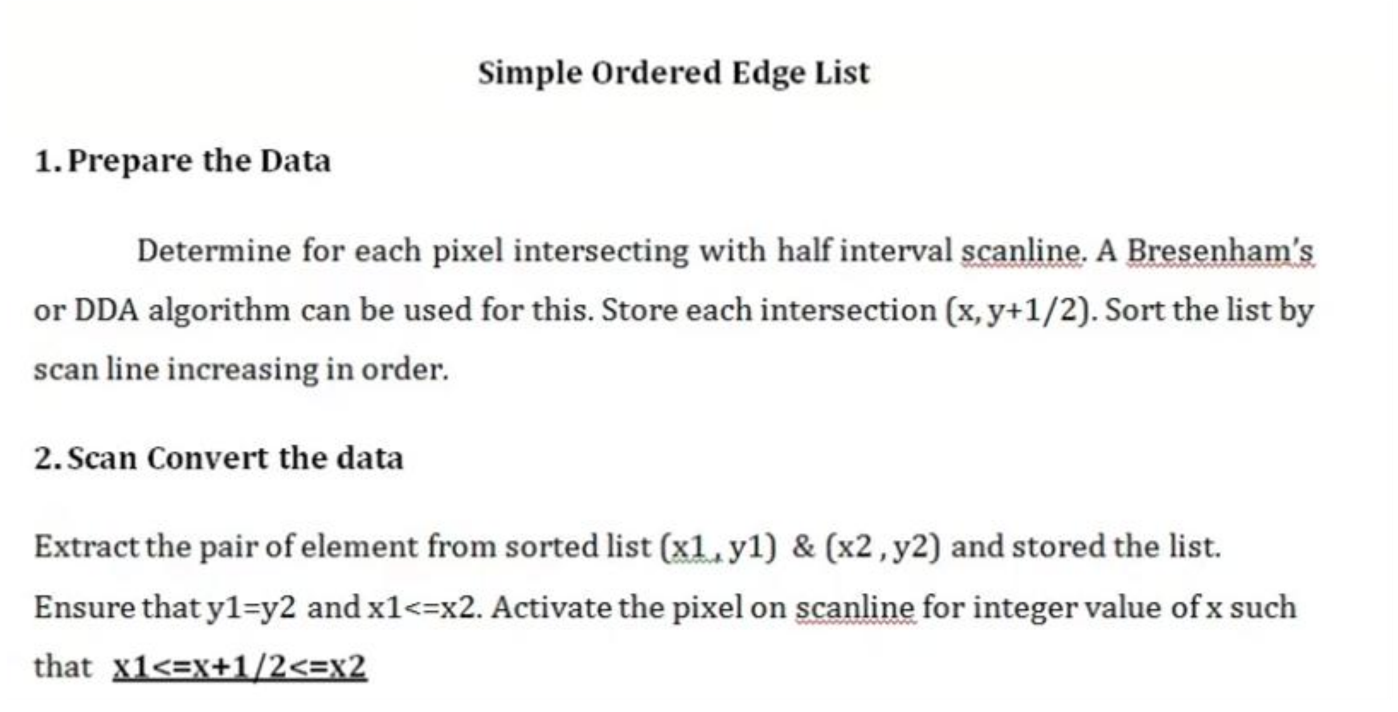
Boundary Fill Algorithm:

The boundary fill algorithm works as its name. This algorithm picks a point inside an figure and starts to fill until it reaches the boundary of the figure. The color of the boundary and the color that we fill should be different for this algorithm to work. In this algorithm, we assume that color of the boundary is same for the entire figure. The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

Flood Fill Algorithm:

Sometimes we come across a figure where we want to fill the area and its boundary of the figure with different colors. We can paint such figure with a specific color instead of searching for particular boundary color as in boundary filling algorithm. Instead of relying on the boundary of the figure, it relies on the fill color. In other words, it replaces the interior color of the figure with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.

Edge List Algorithm



Simple Ordered Edge List

1. Prepare the Data

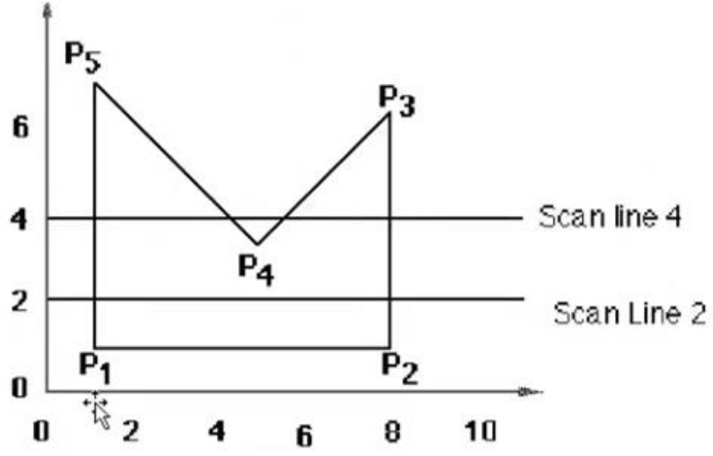
Determine for each pixel intersecting with half interval scanline. A Bresenham's or DDA algorithm can be used for this. Store each intersection (x, y+1/2). Sort the list by scan line increasing in order.

2. Scan Convert the data

Extract the pair of element from sorted list (x1 , y1) & (x2 , y2) and stored the list. Ensure that y1=y2 and x1<=x2. Activate the pixel on scanline for integer value of x such that **x1<=x+1/2<=x2**

Edge List Algorithm

Consider the polygon shown in Figure.

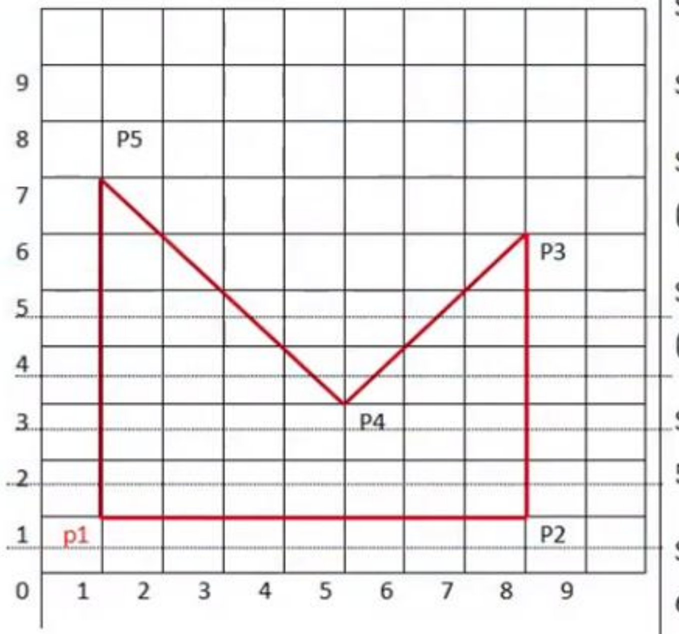


The polygon vertices are

$P_1(1, 1)$, $P_2(8, 1)$, $P_3(8, 6)$, $P_4(5, 3)$ and $P_5(1, 7)$

Edge List Algorithm

Intersections with the half interval scan lines are



Scan line 1.5 : (8, 1.5), (1, 1.5)

Scan line 2.5 : (8, 2.5), (1, 2.5)

Scan line 3.5 : (8, 3.5), (5.5, 3.5), (4.5, 3.5), (1, 3.5)

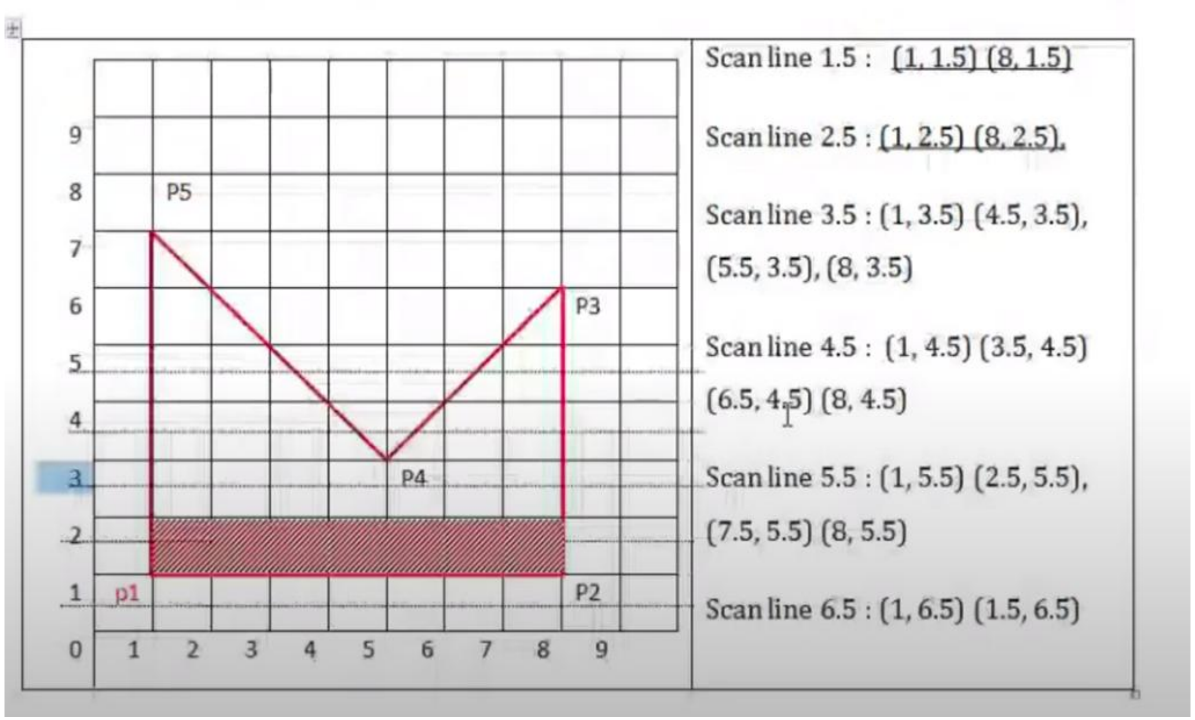
Scan line 4.5 : (8, 4.5), (6.5, 4.5), (3.5, 4.5), (1, 4.5)

Scan line 5.5 : (8, 5.5), (7.5, 5.5), (2.5, 5.5), (1, 5.5)

Scan line 6.5 : (1.5, 6.5), (1, 6.5)

Edge List Algorithm

Sort the list



Scan line 1.5 : (1, 1.5) (8, 1.5)

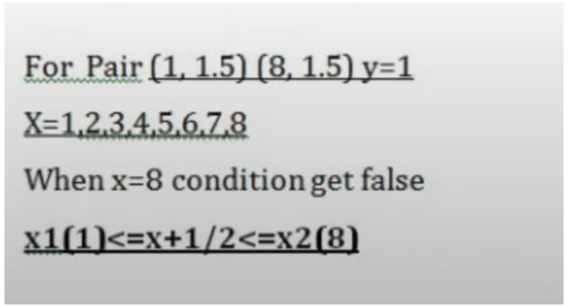
Scan line 2.5 : (1, 2.5) (8, 2.5).

Scan line 3.5 : (1, 3.5) (4.5, 3.5), (5.5, 3.5), (8, 3.5)

Scan line 4.5 : (1, 4.5) (3.5, 4.5) (6.5, 4.5) (8, 4.5)

Scan line 5.5 : (1, 5.5) (2.5, 5.5), (7.5, 5.5) (8, 5.5)

Scan line 6.5 : (1, 6.5) (1.5, 6.5)



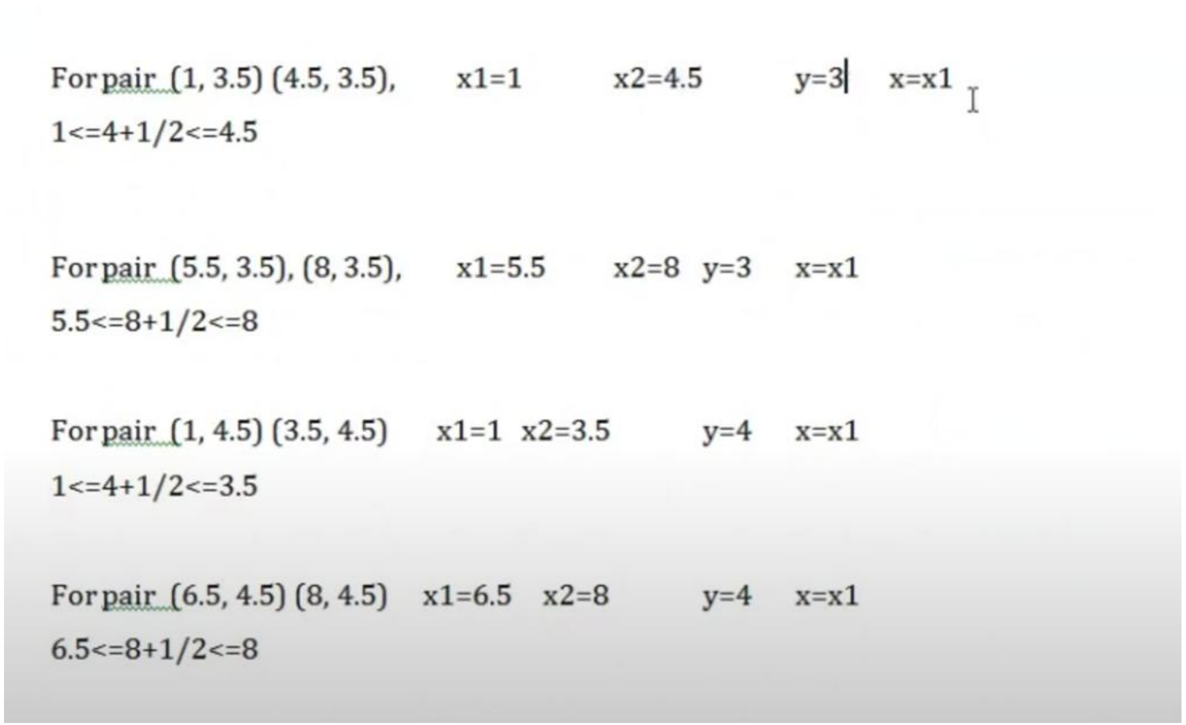
For Pair (1, 1.5) (8, 1.5) y=1

X=1,2,3,4,5,6,7,8

When x=8 condition get false

x1(1)<=x+1/2<=x2(8)

Edge List Algorithm



For pair (1, 3.5) (4.5, 3.5), x1=1 x2=4.5 y=3 x=x1

1<=4+1/2<=4.5

For pair (5.5, 3.5), (8, 3.5), x1=5.5 x2=8 y=3 x=x1

5.5<=8+1/2<=8

For pair (1, 4.5) (3.5, 4.5) x1=1 x2=3.5 y=4 x=x1

1<=4+1/2<=3.5

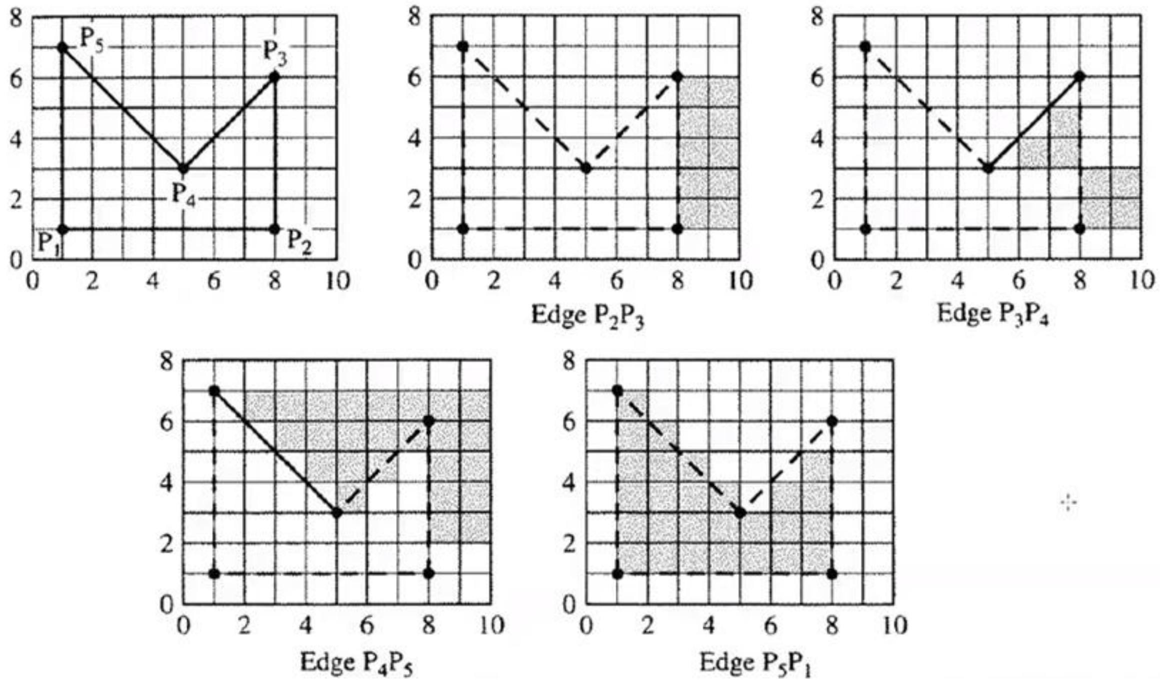
For pair (6.5, 4.5) (8, 4.5) x1=6.5 x2=8 y=4 x=x1

6.5<=8+1/2<=8

Edge Fill Algorithm

Edge fill:

For each scan line intersecting a polygon edge at (x1 y1) complement all pixels whose midpoint lie to the right of (x1 y1) i.e. for (x1 y1) such that x+½>x1

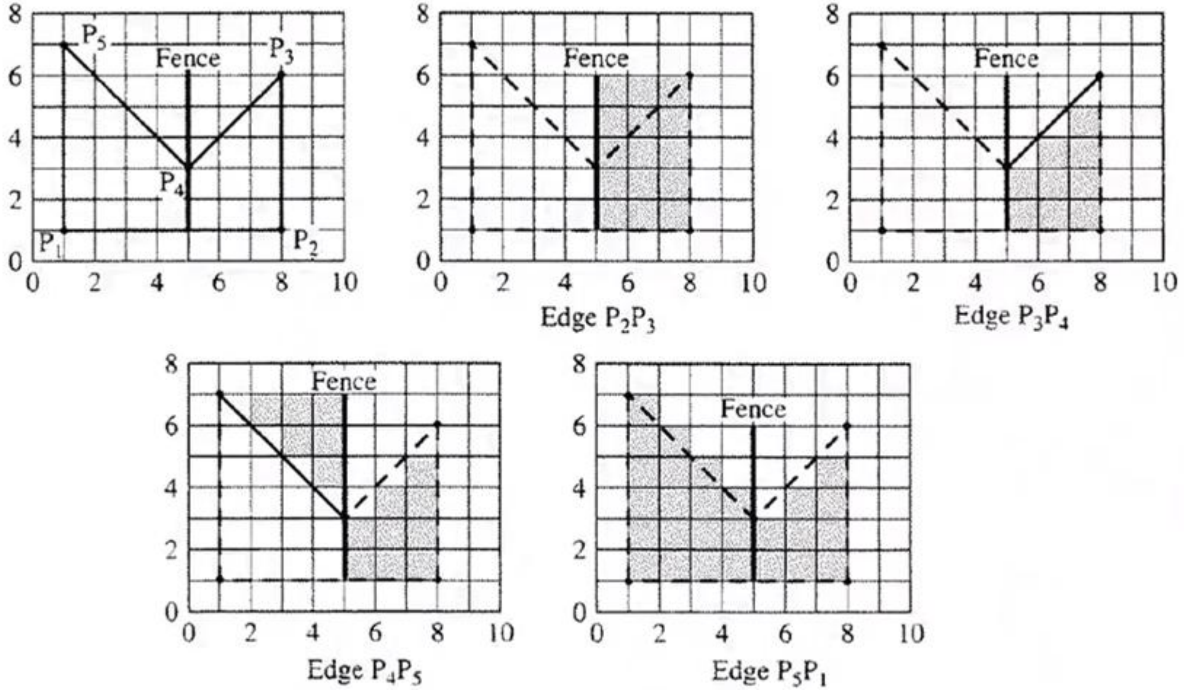


Fence fill Algorithm

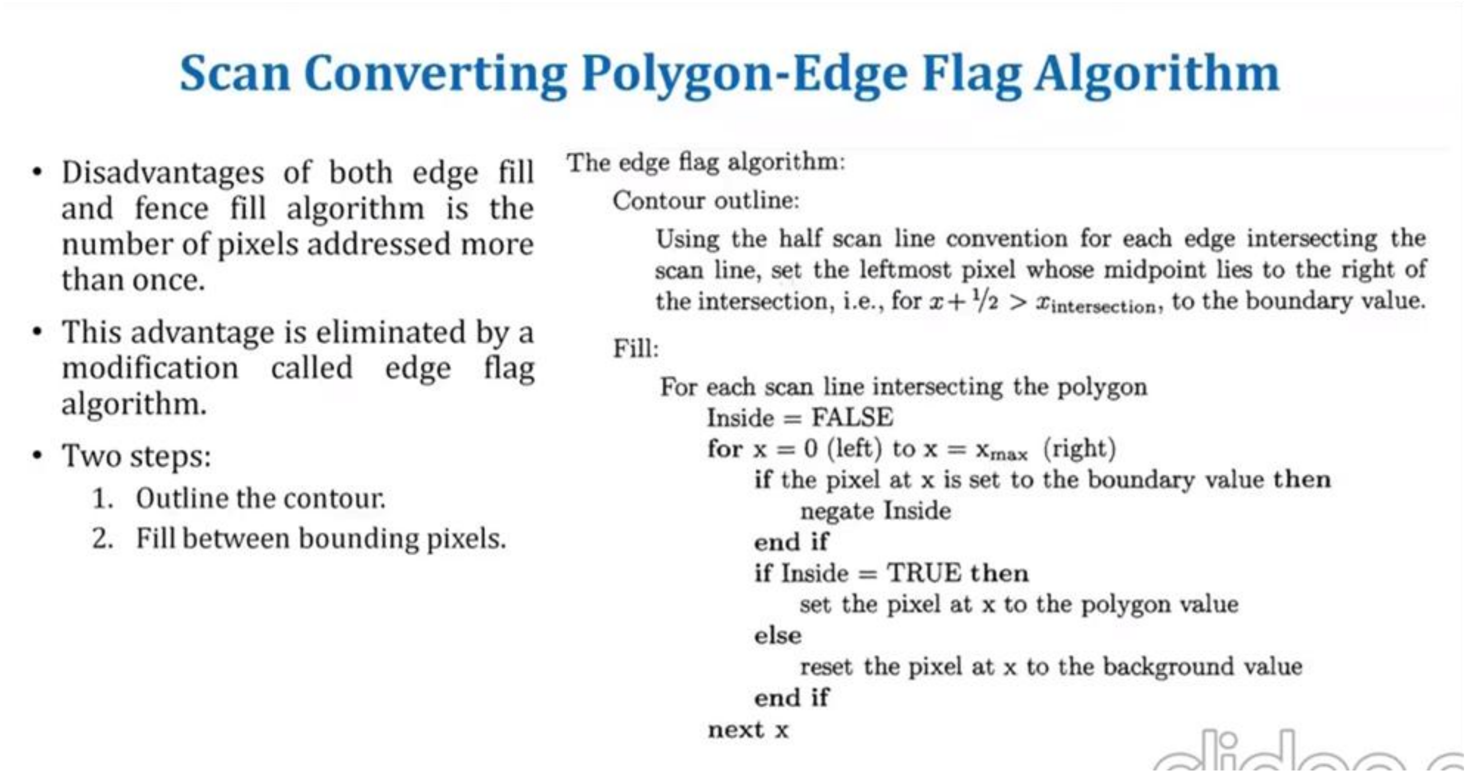
For each scan line intersecting a polygon edge :

If intersection is to left of fence, complement all pixels to the right of the intersection and to left of fence.

Otherwise, complement all pixels to the left of or on the intersection and to the right of fence.



Edge Flag Algorithm



Scan Converting Polygon-Edge Flag Algorithm

- Disadvantages of both edge fill and fence fill algorithm is the number of pixels addressed more than once.
- This advantage is eliminated by a modification called edge flag algorithm.
- Two steps:
  1. Outline the contour.
  2. Fill between bounding pixels.

The edge flag algorithm:

Contour outline:

Using the half scan line convention for each edge intersecting the scan line, set the leftmost pixel whose midpoint lies to the right of the intersection, i.e., for $x + {}^{1}/_{2} > x_{\text{intersection}}$, to the boundary value.

Fill:

```
For each scan line intersecting the polygon
    Inside = FALSE
    for x = 0 (left) to x = $x_{max}$ (right)
        if the pixel at x is set to the boundary value then
            negate Inside
        end if
        if Inside = TRUE then
            set the pixel at x to the polygon value
        else
            reset the pixel at x to the background value
        end if
    next x
```

Scan Converting Polygon-Edge Flag Algorithm

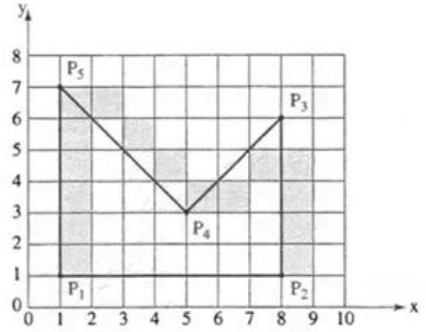
Consider the application of the edge flag algorithm to the example polygon of Fig. 2–34. First the contour is outlined. The result is shown in Fig. 2–42a. Pixels at (1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (1, 6), (2, 6), (3, 5), (4, 4), (5, 3), (6, 3), (7, 4), (8, 4), (8, 3), (8, 2), (8, 1) are activated.

The polygon is then filled. To illustrate this the scan line at 3 is extracted and shown in Fig. 2–47b. Pixels at $x = 1$, 5, 6, and 8 on this scan line are activated to outline the contour. Applying the fill algorithm yields

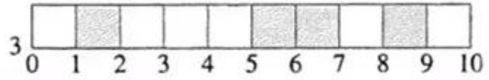
Initially

Inside = FALSE

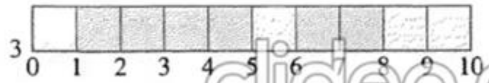
For $x = 0$ The pixel is not set to the boundary value and Inside = FALSE. Therefore, no action is taken.



(a)



(b)

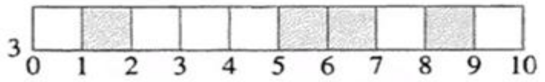


(c)

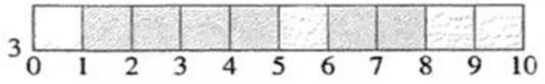
Scan Converting Polygon-Edge Flag Algorithm

For $x = 1$	The pixel is set to the boundary value, Inside is negated to TRUE. Inside = TRUE, so the pixel is set to the polygon value.
For $x = 2, 3, 4$	The pixel is not set to the boundary value. Inside = TRUE, so the pixel is set to the polygon value.
For $x = 5$	The pixel is set to the boundary value, Inside is negated to FALSE. Inside = FALSE, so the pixel is set to the background value.
For $x = 6$	The pixel is set to the boundary value, Inside is negated to TRUE. Inside = TRUE, so the pixel is set to the polygon value.
For $x = 7$	The pixel is not set to the boundary value. Inside = TRUE, so the pixel is set to the polygon value.
For $x = 8$	The pixel is set to the boundary value, Inside is negated to FALSE. Inside = FALSE, so the pixel is set to the background.

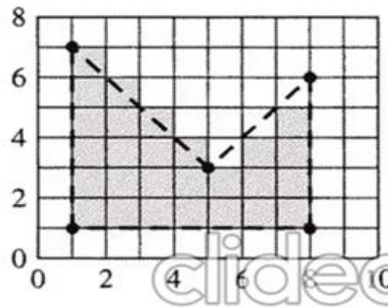
The result is shown in Fig. 2–47c. The final result for the complete polygon is the same as for the edge fill algorithm as it is shown in Fig. 2–45.



(b)



(c)



Scan Line Fill Method:

Figures on a computer screen can be drawn using polygons. To fill those figures with color, we need to develop some algorithm.

There are two famous algorithms for this purpose: Boundary fill and Scanline fill algorithms. This method is also called “alternative fill method”.

The scan line fill method is a very efficient & cheaper alternative. This is the method in which scan line are used for the filling process. Scan line fill method works very efficiently with self-intersecting polygons as well.

This algorithm works by intersecting scan line with polygon edges and fills the polygon between pairs of intersections. The following steps depict how this algorithm works.

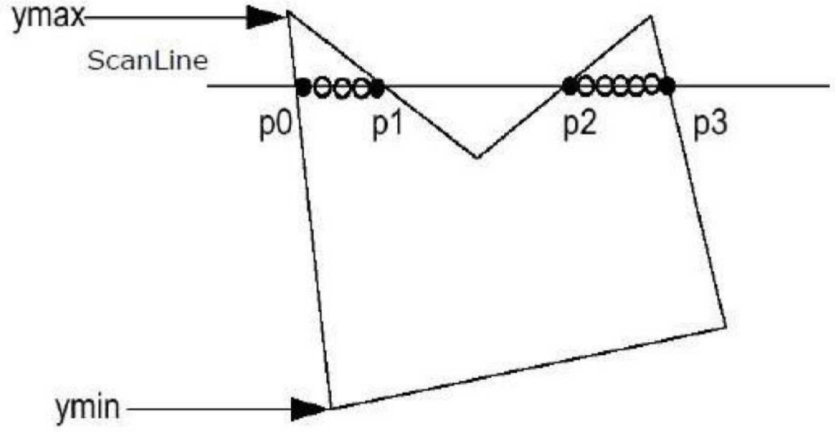
Scan Line Fill Method:

Step 1 − Find out the Ymin and Ymax from the given polygon.

Step 2 − Scan Line intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

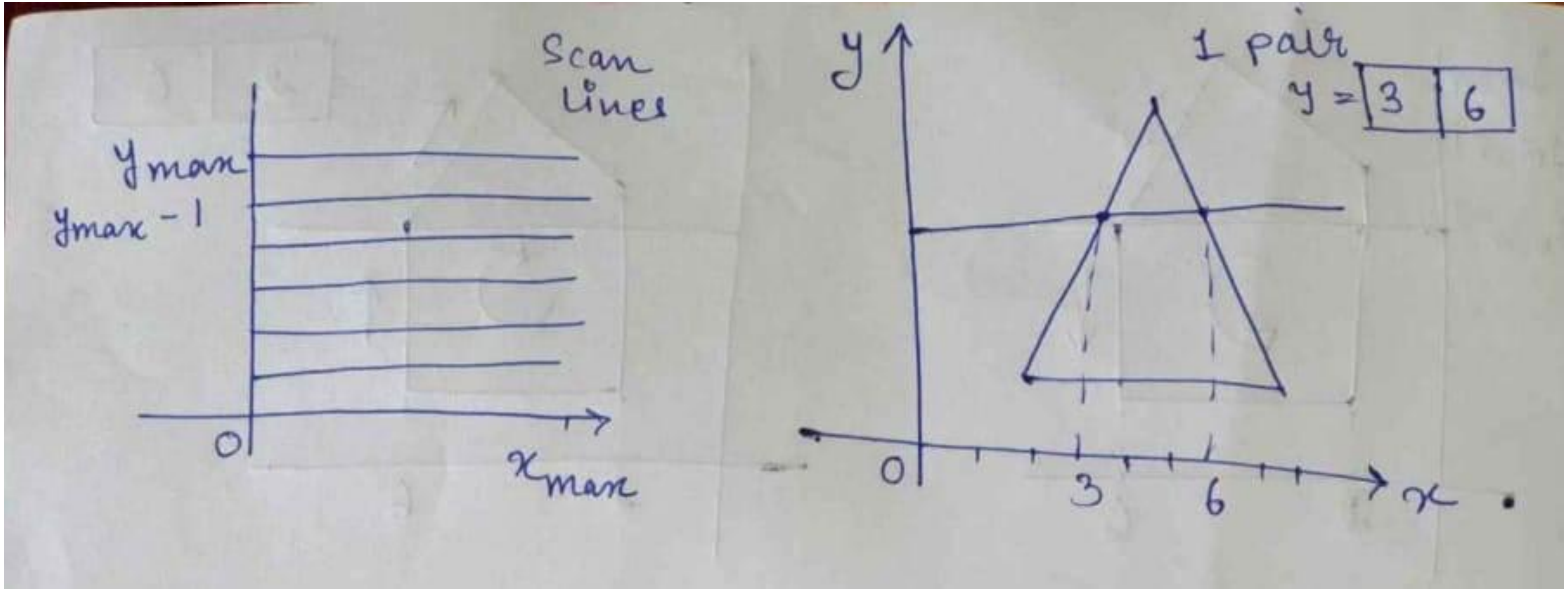
Step 3 − Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

Step 4 − Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

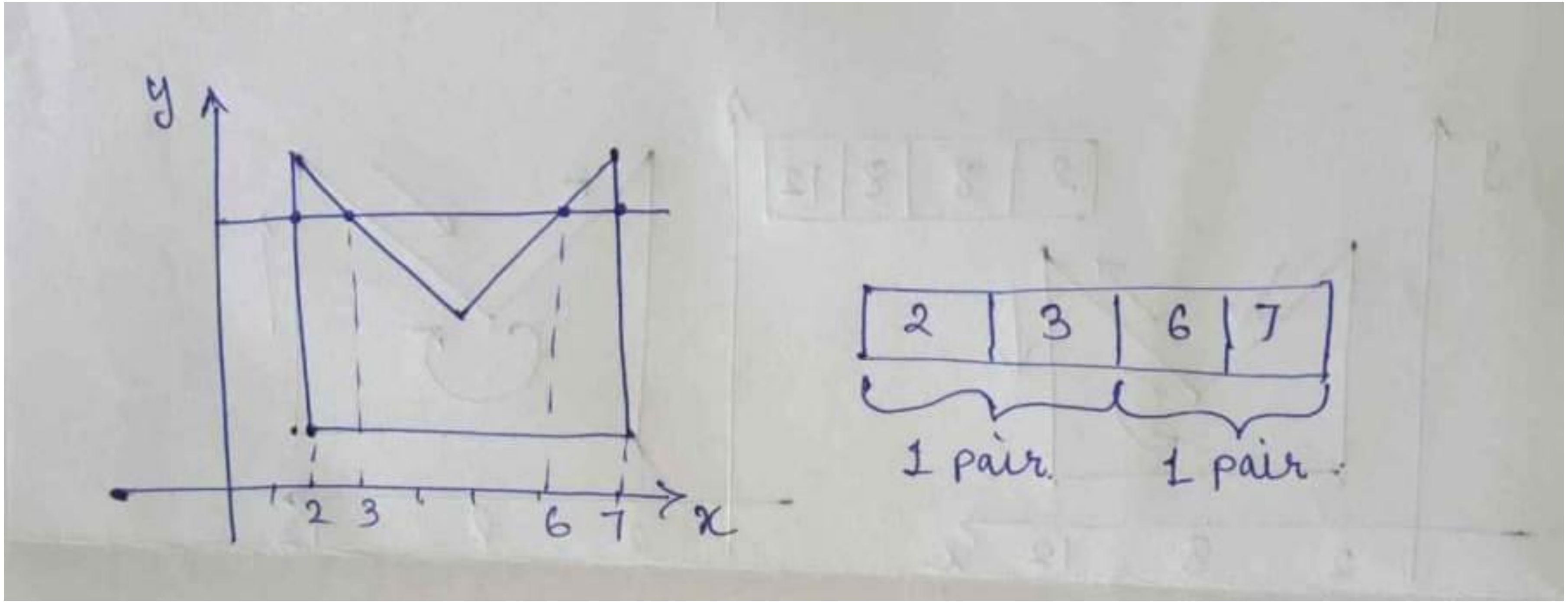


Here the color is filled with scan lines.

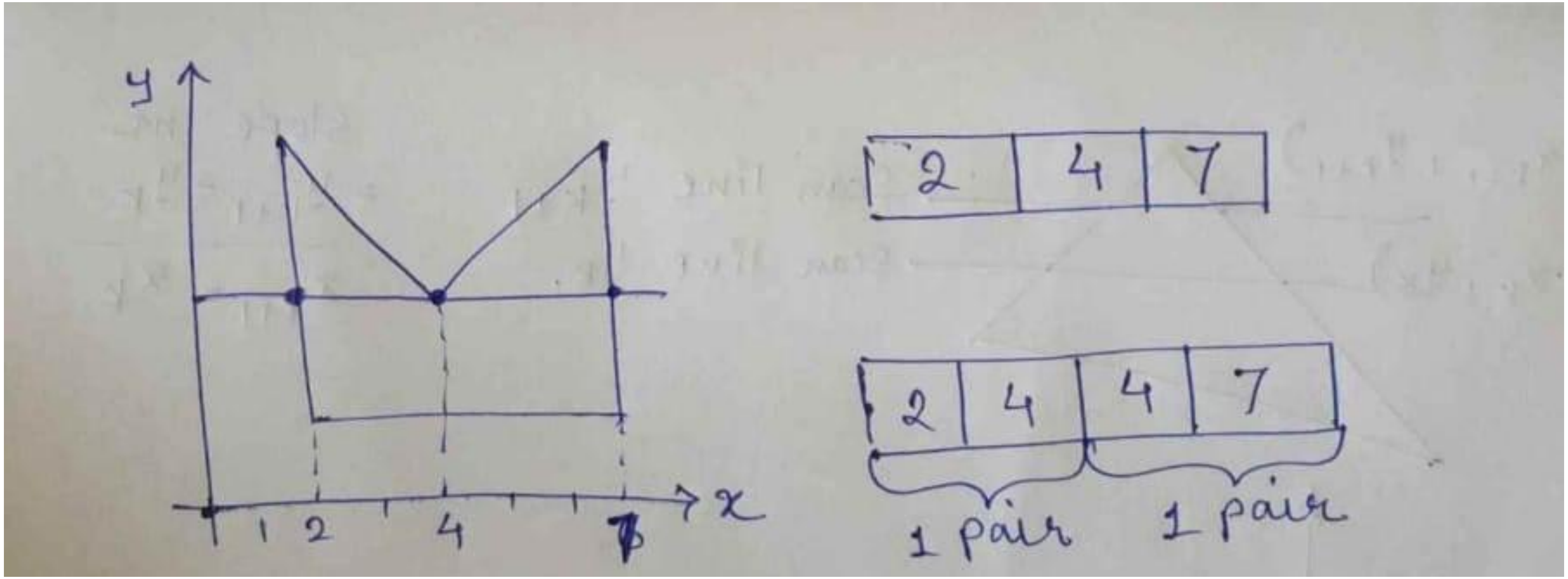
Find the intersection between the scanlines & the edges. The 2 points (3,6) is stored in the frame buffer in the sorted order & pixel are colored from (3,6).



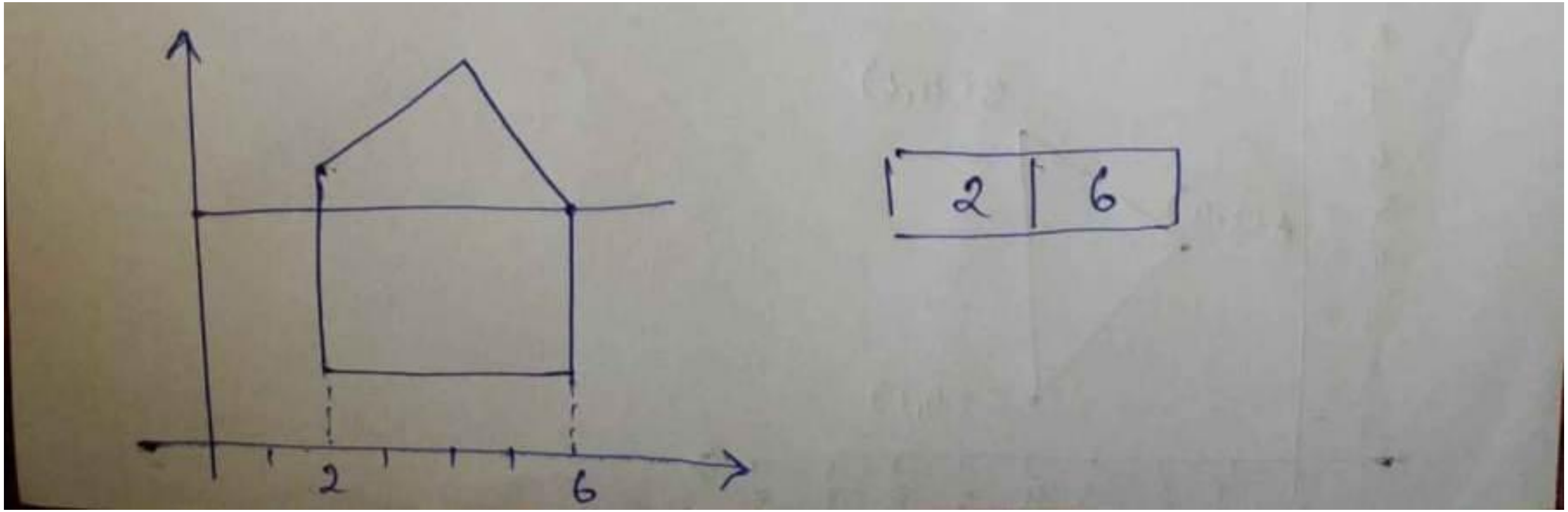
Scan line intersection pair



If scan line is passing through a vertex and the no. of intersection point is odd, then the vertex should be considered as a 2 intersection points.



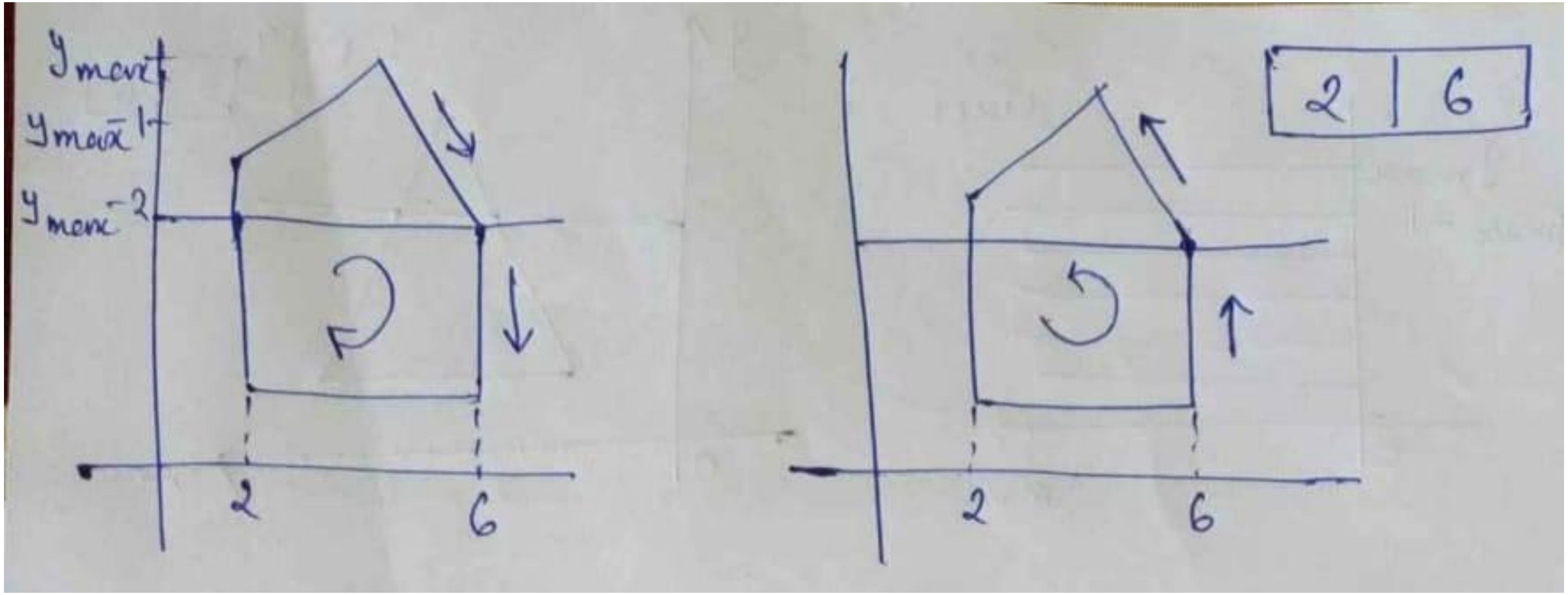
If scan line is passing through a vertex and the no. of intersection point is even, then the vertex should be considered as a 1 intersection points.



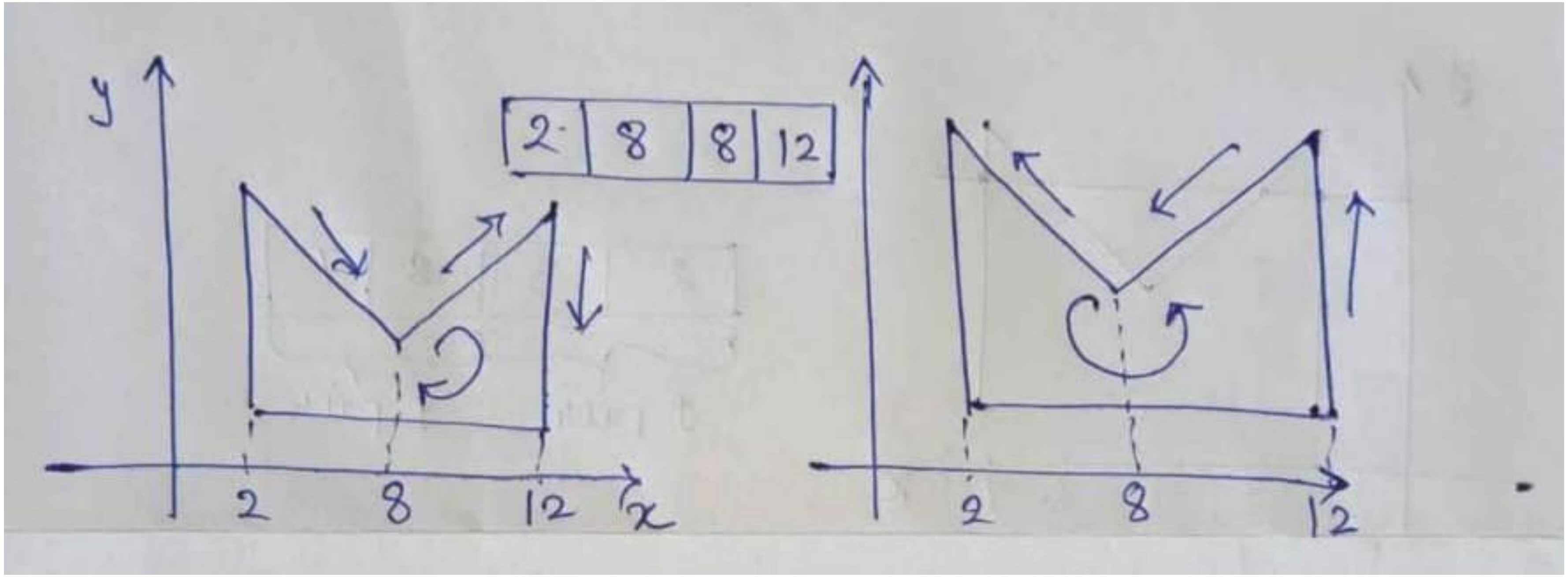
Test Method:

Traverse the edges in clockwise direction & observe the values of y. If y value is decreasing then consider it as one intersection point.

Traverse the edges in anticlockwise direction & observe the values of y. If y value is increasing then consider it as one intersection point.



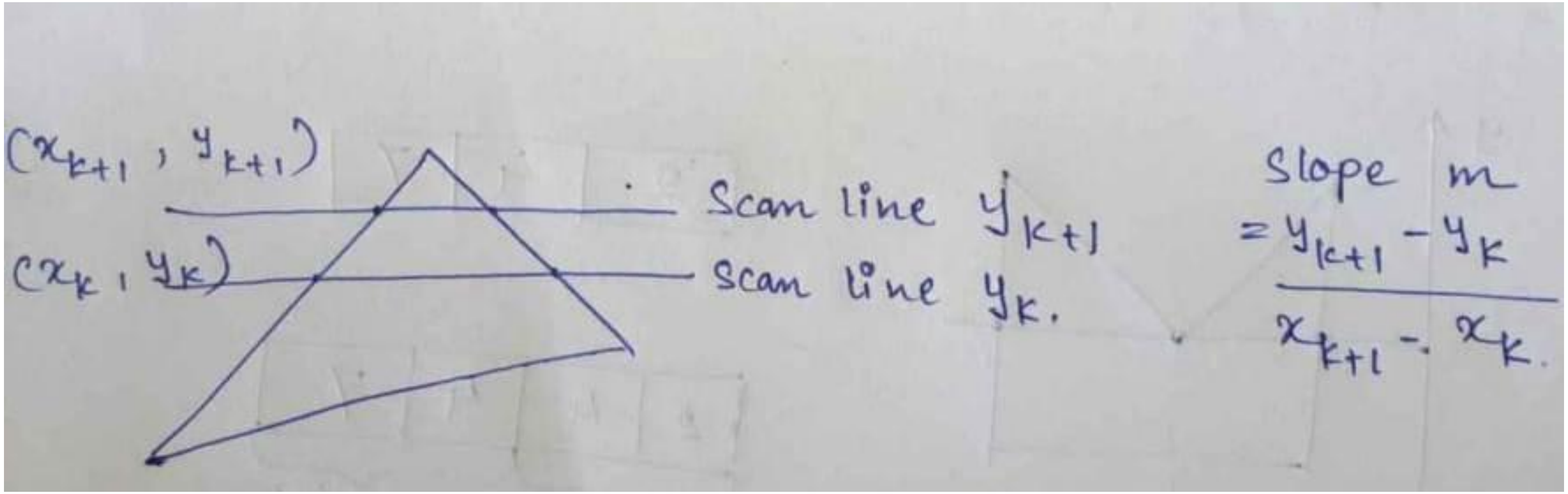
In that case at one edge the value of y is decreasing and other edge is increasing then consider the vertex as 2 intersection point.



Scan line algorithm

Next value of Y: change in y-coordinates between the scan line is,

$Y_{k+1} - Y_k = 1$ $Y_{k+1} = Y_k + 1$



Next value of x: successive x can be calculated by

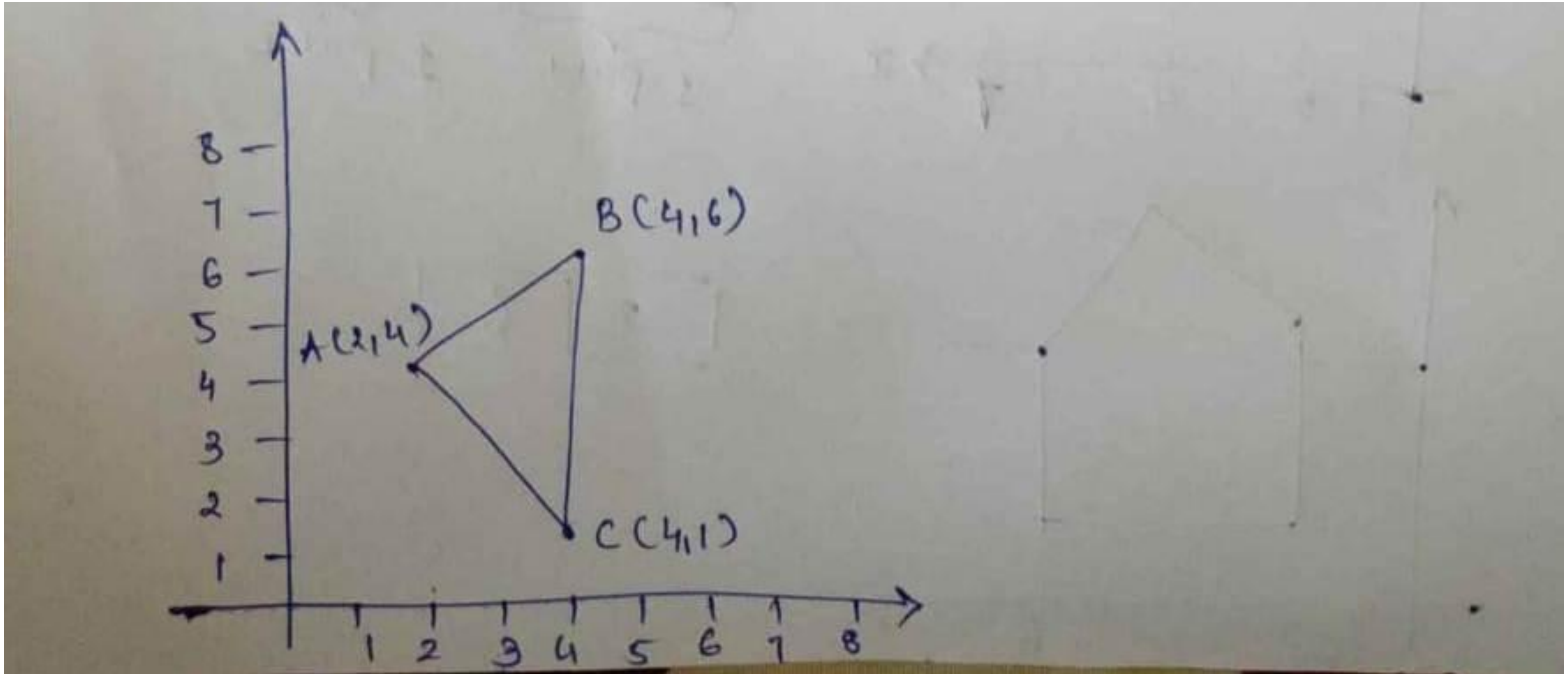
$X_{k+1} = X_k + 1/m$

$X_{k+1} - X_k = m$

$X_{k+1} = X_k + m$

Problem on scan line algorithm

Explain scan line polygon fill algorithm: Determine the content of the active edge table to fill the polygon with vertices A(2,4), B(4,6) and C(4,1) for y=1 to y=6.



Character generation

Letters, numbers, and other character are often displayed to label and annotate drawing and to give instructions and information to the user.

Most of the times characters are built into the graphics display devices, usually as hardware but sometimes through software.

There are basic three methods:

– Stroke method

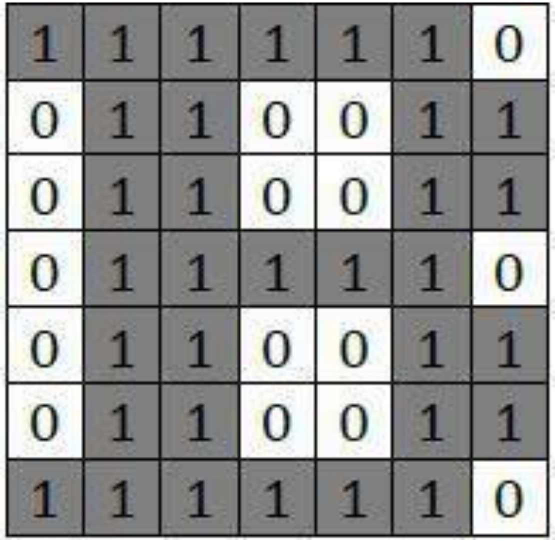
– Star bust method

– Bitmap method

Bitmap Font/ Bitmapped Font: Character Generation

A simple method for representing the character shapes in a particular typeface is to use rectangular grid patterns.

The figure below shows the pattern for the particular letter.



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

When the pattern in the figure copied to the area of the frame buffer, the 1 bits designate which pixel positions to displayed on the monitor.

Bitmap fonts the simplest to define and display as character grid only need to be mapped to a frame buffer position.

Bitmap fonts require more space because each variation (size and format) must stored in a font cache.

It possible to generate different size and other variation from one set but this usually does not produce the good result.

Stroke Method: Character Generation

This method uses small line segments to generate a character.

The small series of line segments are drawn like a strokes of a pen to form a character as shown in figure.

We can build our own stroke method.

By calling a line drawing algorithm.

Here it is necessary to decide which line segments are needed for each character and

Then drawing these segments using line drawing algo.

This method supports scaling of the character.

It does this by changing the length of the line segments used for character drawing.

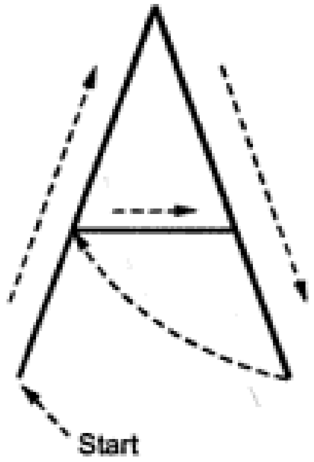


Fig. 2.31 Stroke method

Starburst Method

In this method a fix pattern of line segments are used to generate characters.

As shown in figure, there are 24 line segments.

Out of 24 line segments, segments required to display for particular character, are highlighted

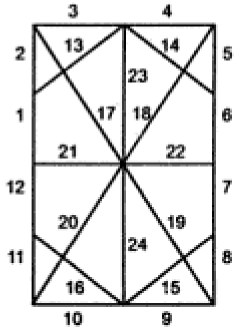
This method is called starbust method because of its characteristic appearance.

Fig shows the starburst patterns for character A and M.

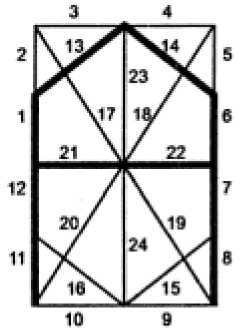
The patterns for particular characters are stored in the form of 24 bits code.

Each bit representing one line segment.

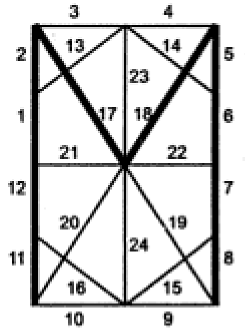
The bit is set to one to highlight the line segment otherwise it is set to zero.



a) Star bust pattern
of 24 line segments



b) Star bust pattern
for.character A



c) Star bust pattern
for character M

Character A : 0011 0000 0011 1100 1110 0001
Character M:0000 0011 0000 1100 1111 0011

This method of character generation is not used now a days because of following disadvantages:

The 24-bits are required to represent a character. Hence more memory is required.

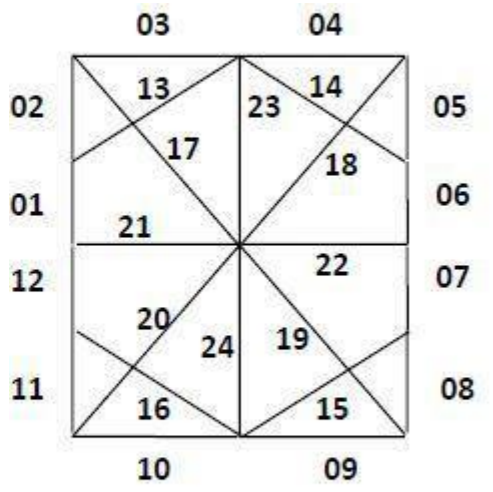
Requires code conversion software to display character from its 24 bits code.

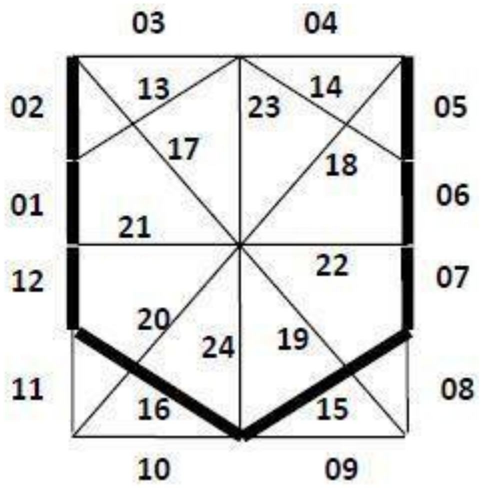
Character quality poor.

It doesn’t provide curve shapes, so worst for curve shaped character.

Code for letter V is

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


03
04
02
13
23
14
05
17
18
01
06
21
22
12
07
20
24
19
11
08
16
15
10
09


03
04
02
13
23
14
05
17
18
01
06
21
22
12
07
20
24
19
11
08
16
15
10
09