KECE471 Computer Vision

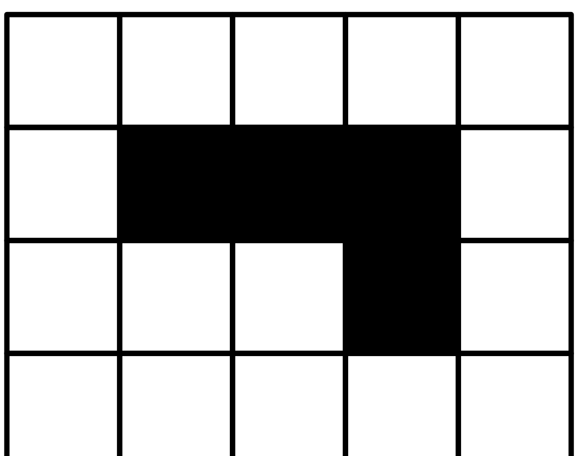
Binary Image Analysis

Chang-Su Kim

Chapter 3, Computer Vision by Shapiro and Stockman
Chapter 3, Computer Vision by Szeliski

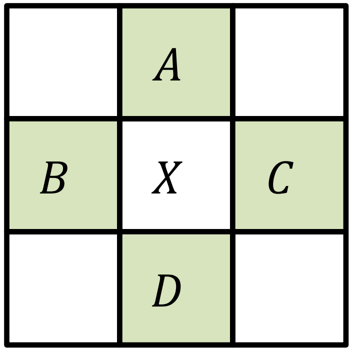
Binary Images

- Binary image **B**
- **B**[r, c]: binary value of the pixel at row r and column c
  - **B**[r, c] = 1 : [r, c] is a foreground (or black) pixel
  - **B**[r, c] = 0 : [r, c] is a background (or white) pixel

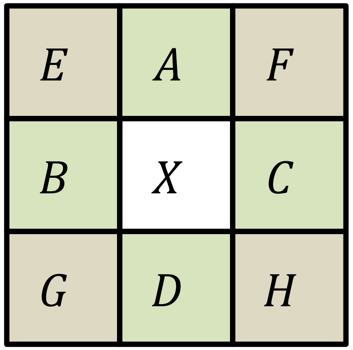


Neighborhoods

- 4-neighborhood N_4
  - $\{A, B, C, D\}$ is the 4-neighborhood of X
  - A neighbors X in the context of 4-neighborhood



- 8-neighborhood N_8
  - $\{A, B, C, D, E, F, G, H\}$ is the 8-neighborhood of X
  - C or F neighbors X in the context of 8-neighborhood



Applying Masks to Images

- It is like convolution

- For each pixel in the input image
  - Place the mask on top of the image with its origin lying on the pixel
  - Multiply the value of each input image pixel under the mask by the weight of the corresponding mask pixel, and then add those products together
  - Put the sum into the output image at the location of the input pixel being processed

Applying Masks to Images

Ex)

40	40	80	80	80
40	40	80	80	80
40	40	80	80	80
40	40	80	80	80
40	40	80	80	80

(*a*) Original gray-tone image

1	2	1
2	4	2
1	2	1

(*b*) 3 × 3 mask

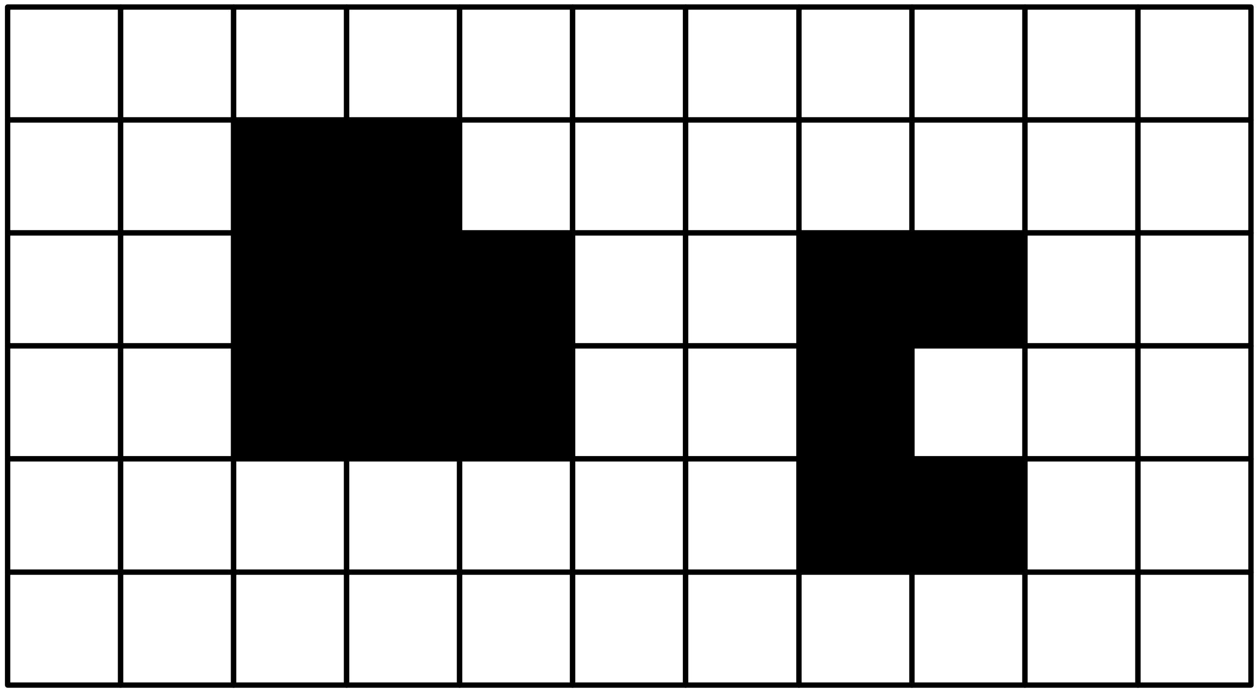
640	800	1120	1280	1280
640	800	1120	1280	1280
640	800	1120	1280	1280
640	800	1120	1280	1280
640	800	1120	1280	1280

(*c*) Result of applying the mask to the image

40	50	70	80	80
40	50	70	80	80
40	50	70	80	80
40	50	70	80	80
40	50	70	80	80

(*d*) Normalized result after division by the sum of the weights in the mask (16)

Counting Objects in an Image

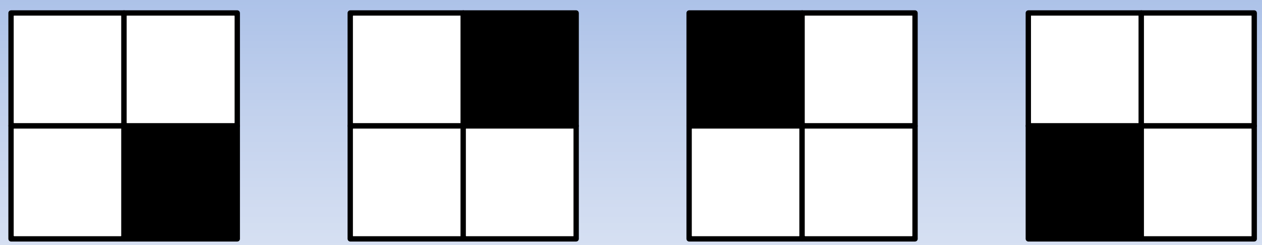


How many objects are there?

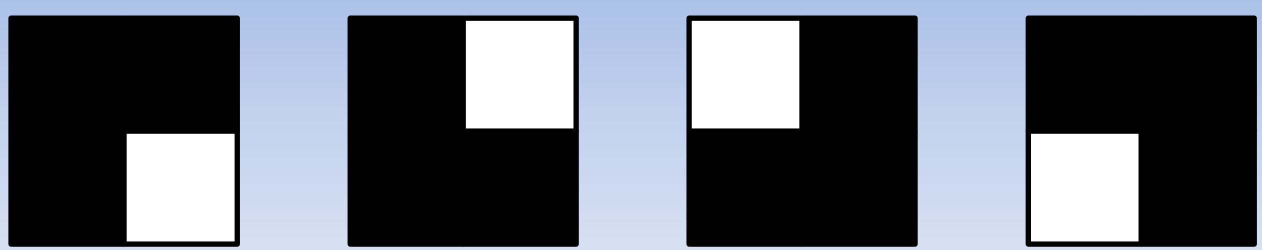
Counting Objects in an Image

- How can a computer count them?
- One approach is using the corner patterns

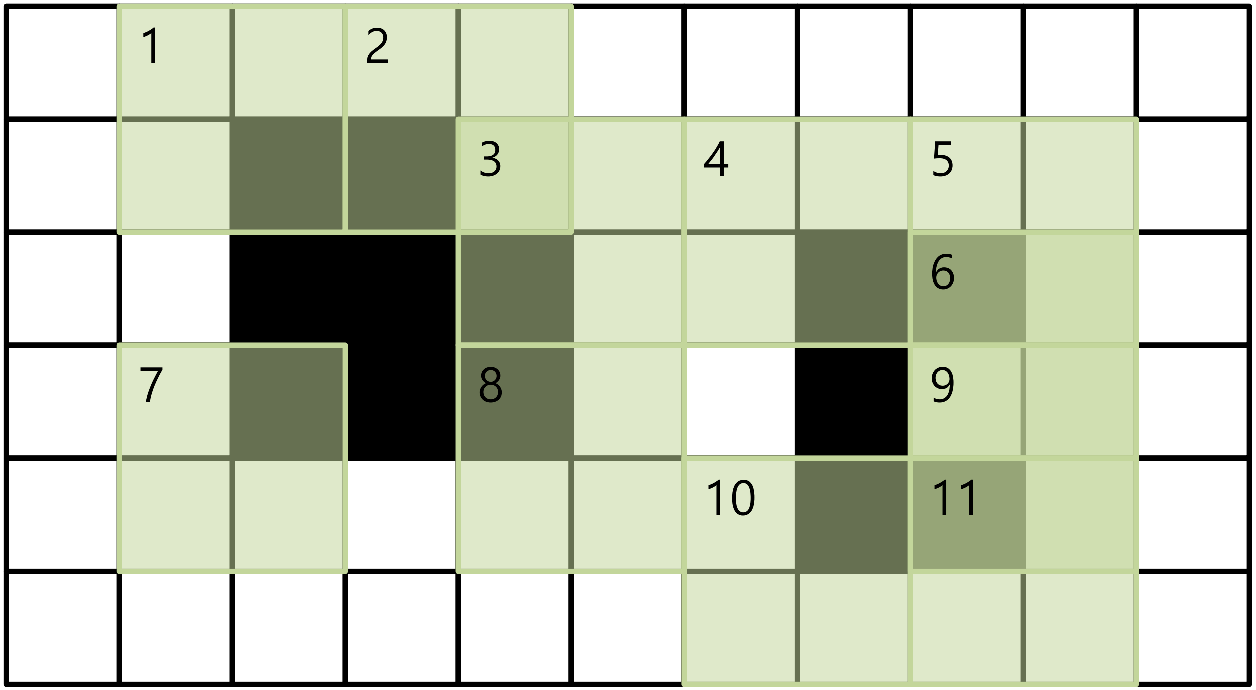
External Corners



Internal Corners

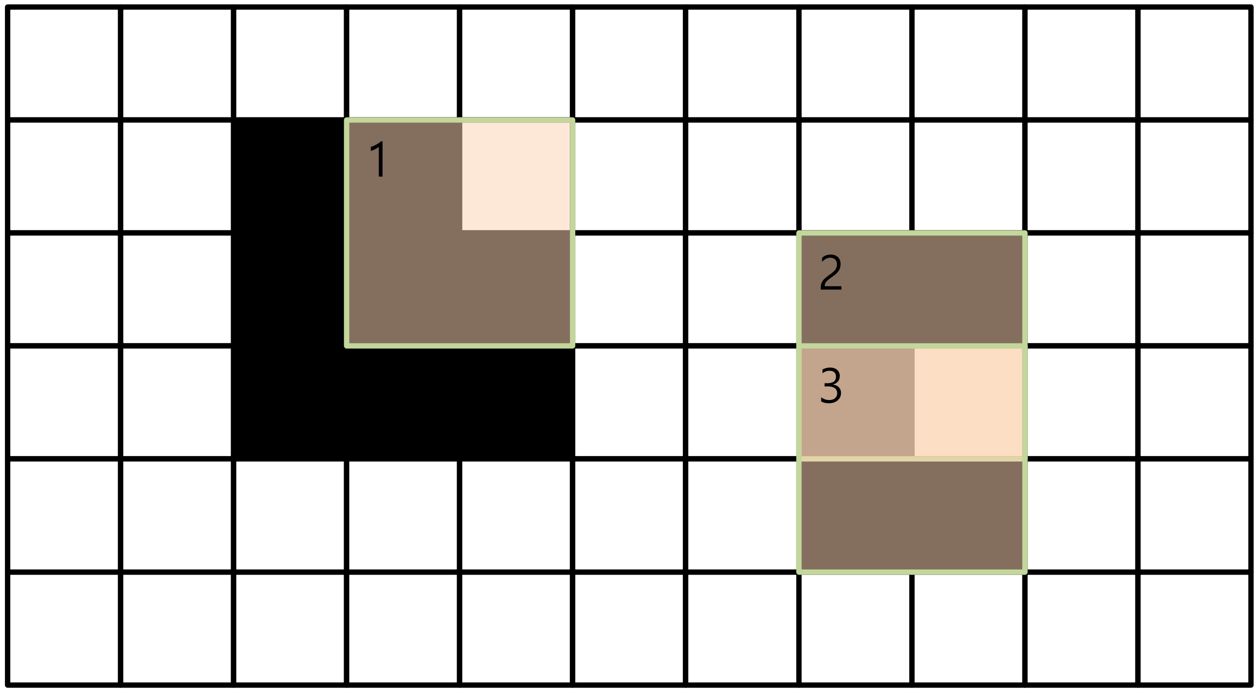


Counting Objects in an Image



There are 11 external corners (E = 11)

Counting Objects in an Image

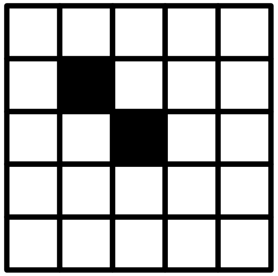


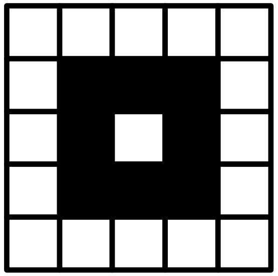
There are three internal corners (I = 3)

Counting Objects in an Image

of objects = (E – I)/4

- In an object, E – I = 4
  - This is obvious for a rectangle (E = 4, I = 0)
  - When you remove or paste a black pixel, it does not change the difference
- The formula does not hold if
  - different objects share a vertex or
  - objects contains holes

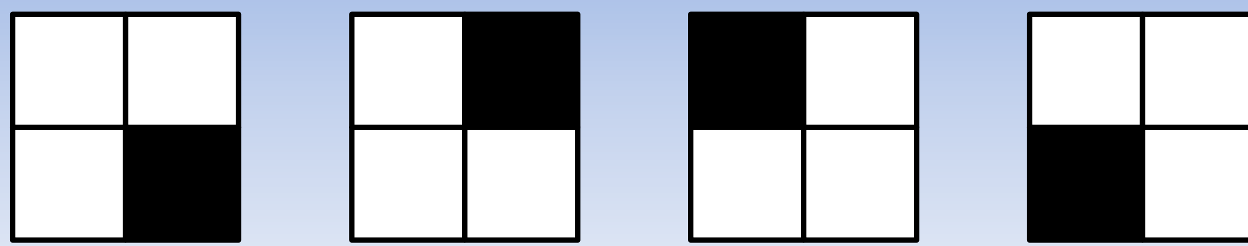




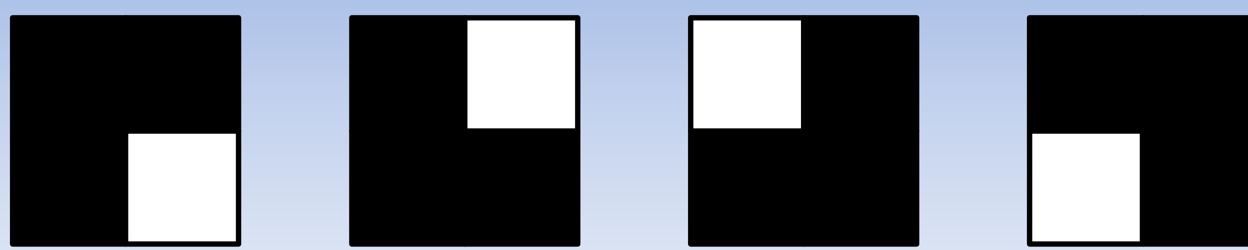
Sketch of Proof

# of objects = (E – I)/4

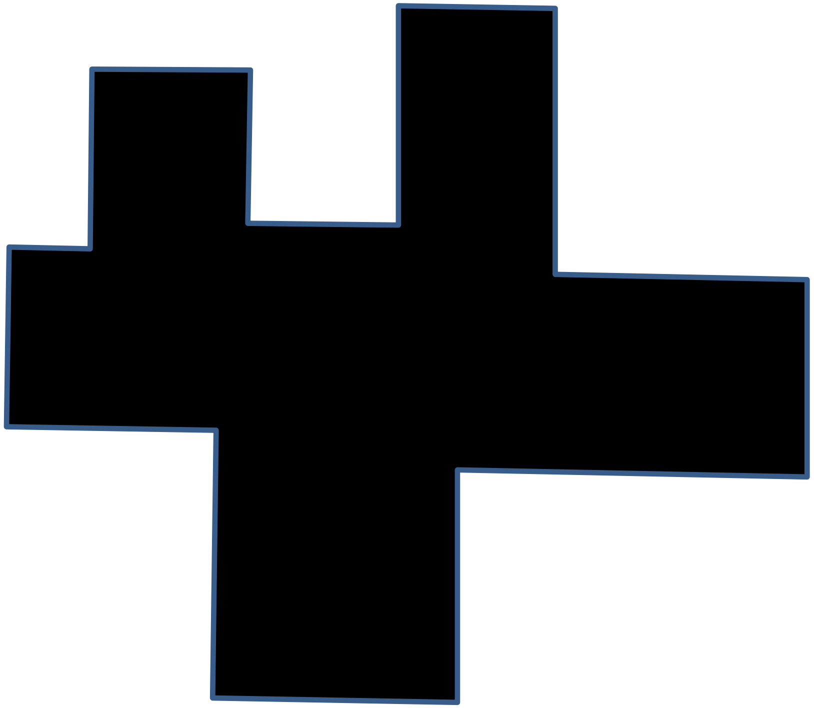
External Corners



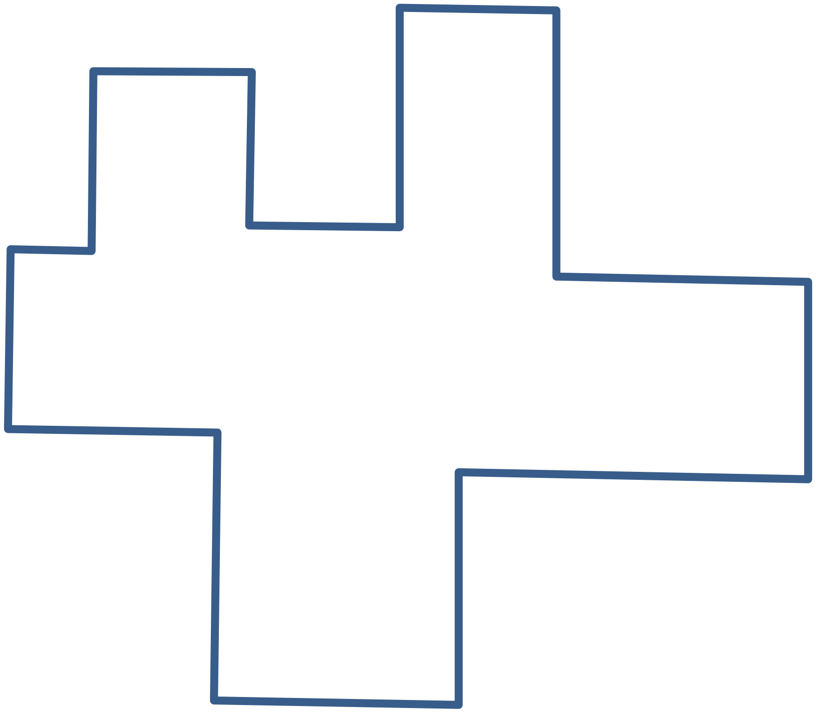
Internal Corners



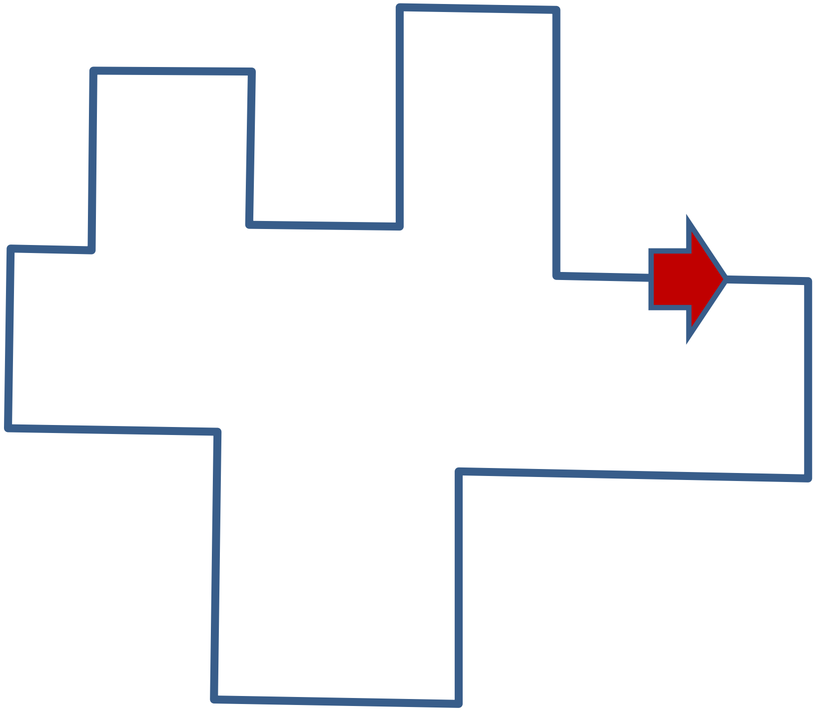
Counting Objects in an Image



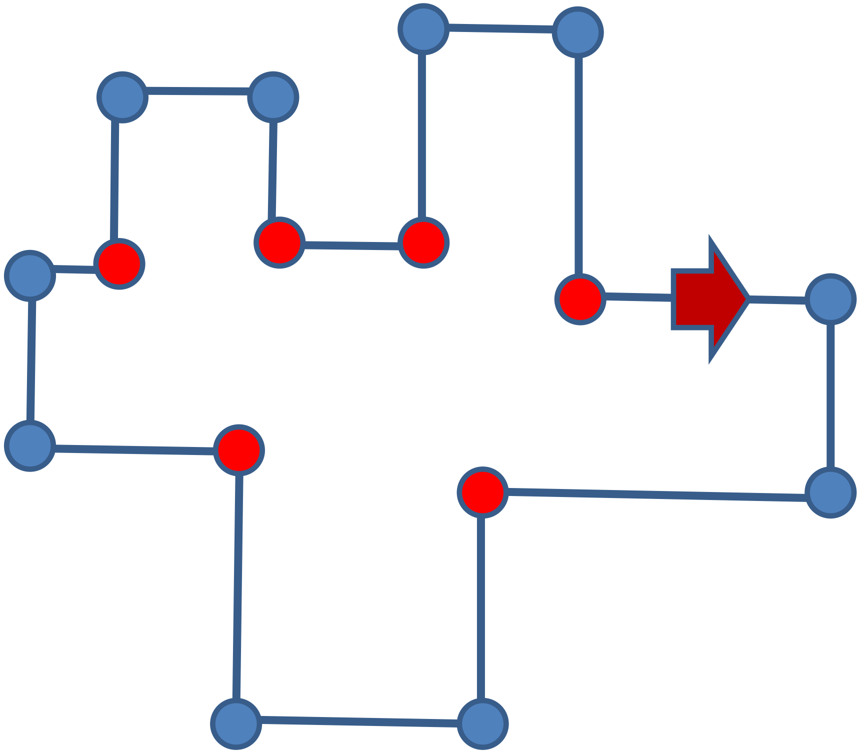
Counting Objects in an Image



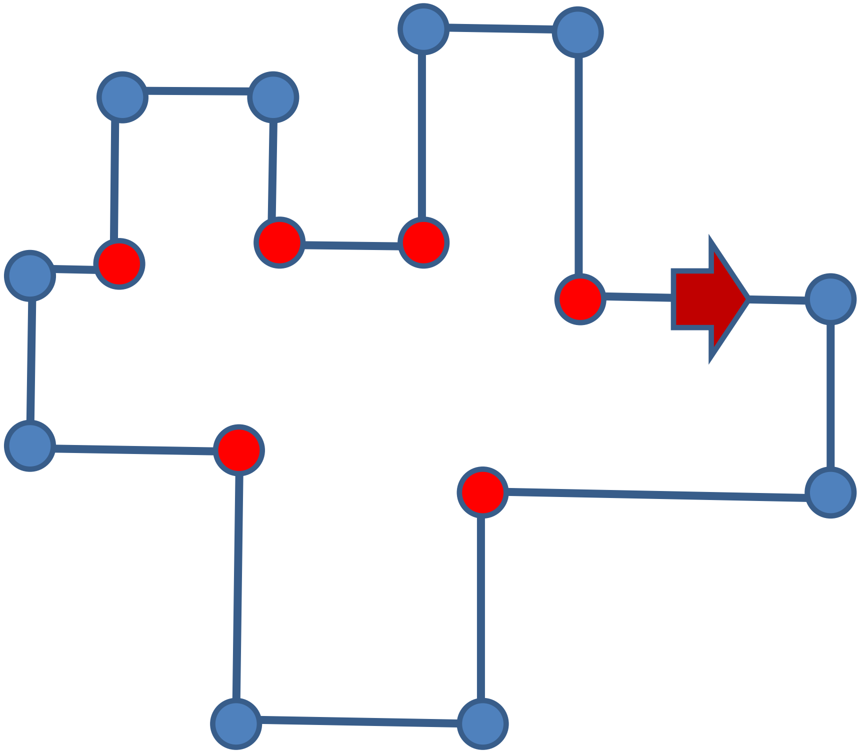
Counting Objects in an Image



Counting Objects in an Image



Counting Objects in an Image



You need four more right turns than left turns to make a round trip

Spectacle

Creators vs Spectators

Connectedness

- A pixel $[r, c]$ is connected to another pixel $[r', c']$ with respect to value v
  - if there is a sequence of pixels

$$[r, c] = [r_0, c_0], [r_1, c_1], \ldots , [r_n, c_n] = [r', c'] \quad (1)$$

  such that

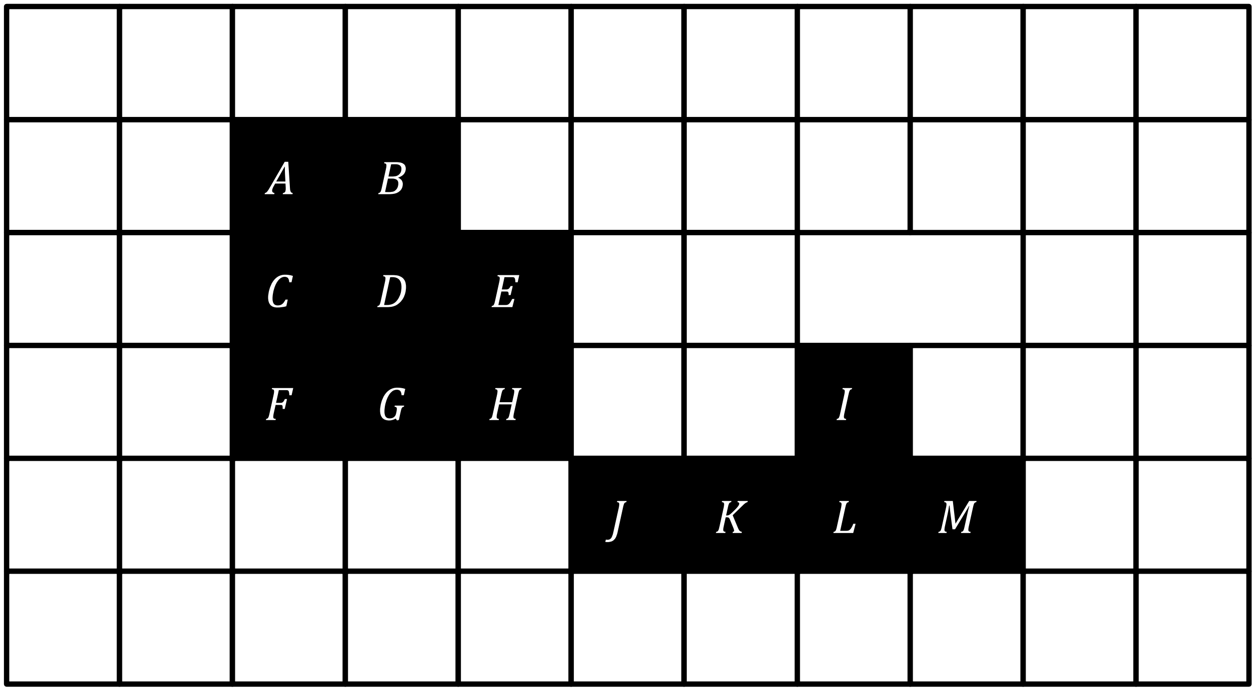
$$\boldsymbol{B}[r_i, c_i] = v \text{ for all } 0 \leq i \leq n \text{ and}$$

$$[r_i, c_i] \text{ neighbors } [r_{i-1}, c_{i-1}] \text{ for all } 1 \leq i \leq n$$

- The sequence in (1) is called a path from $[r, c]$ to $[r', c']$
- A connected component is a maximum set of pixels, such that every pair of pixels in the set are connected.

  Note: all definitions can be made in terms of the 4-neighborhood or 8-neighborhood.

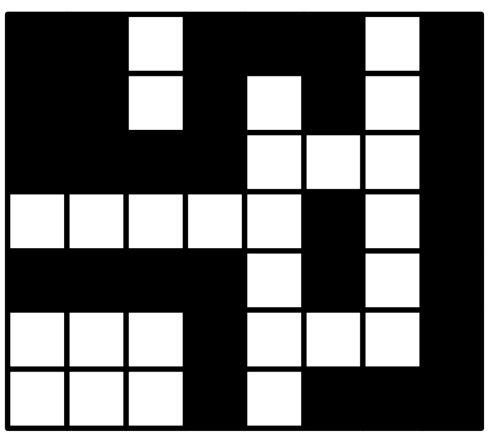
Connectedness



- 4-neighborhood
  - A and H are connected
  - A and K are not connected
  - (A, D, H) is a path from A to H
  - $\{A, B, C, D, E, F, G, H\}$ is a connected component
- 8-neighborhood
  - A and H are connected
  - A and K are connected
  - (A, D, H, J, K) is a path from A to K
  - $\{A, B, C, D, E, F, G, H\}$ is not a connected component

Connected Components Labeling

- A connected components labeling of a binary image $\boldsymbol{B}$ is a labeled image $\boldsymbol{L}$ in which the value of each foreground pixel is the label of its connected component
  - background pixels are assigned 0



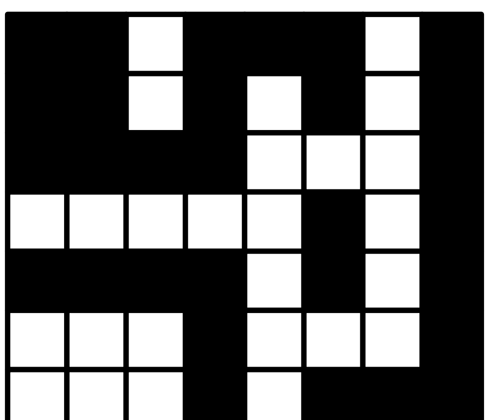
B

		0				0	
		0		0		0	
				0	0	0	
0	0	0	0	0		0	
				0		0	
0	0	0		0	0	0	
0	0	0		0			

L

Connected Components Labeling

- A connected components labeling of a binary image **B** is a labeled image **L** in which the value of each foreground pixel is the label of its connected component
  - background pixels are assigned 0



B

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

L

Connected Components Labeling

- Two algorithms
  - Recursive labeling
    - Random access to the whole image is possible
  - Row-by-row labeling
    - Image is big and processed in row-by-row manner
    - Only two rows are processed at a time
    - Self-study

Recursive Labeling

```
void recurisve_labeling(B, L)
{
    L = negate(B); // 1 -> -1
    label = 0;
    find_components(L, label);
    print(L);
}
```

		1					
	1	1	1			1	
		1	1		1	1	
		1			1		

```
void find_components(L, label)
{
    for(r=0 to MaxR) for(c=0 to MaxC)
    if(L(r, c) == -1){
            label++;
            search(L, label, r, c)
    }
}
```

		-1					
	-1	-1	-1			-1	
		-1	-1		-1	-1	
		-1			-1		

Recursive Labeling

```
void search(L, label, r, c)
{
    L[r, c] = label;
    Nset = neighbors(r, c);  // Nset becomes the 4-neighborhood of [r, c]
    for each [r', c'] in Nset {
        if(L[r', c'] == -1)
            search(L, label, r', c');   // recursion
    }
}
```

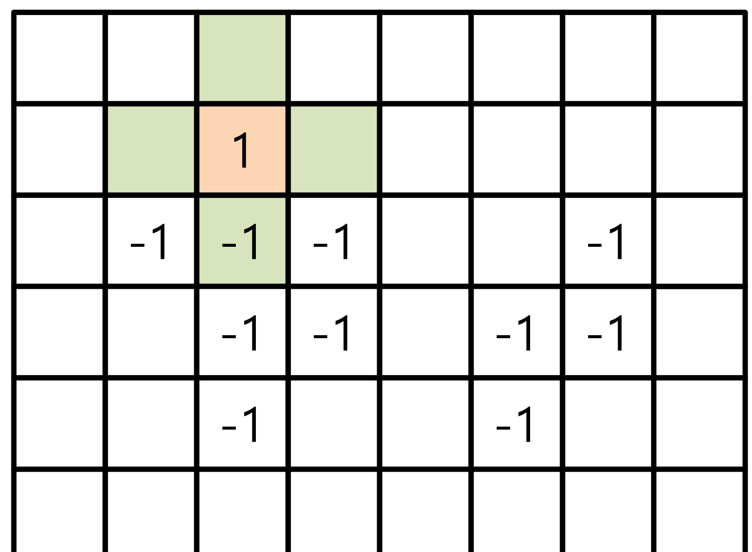
r=1, c=2, label =1

		1					
	-1	-1	-1			-1	
		-1	-1		-1	-1	
		-1			-1		

Recursive Labeling

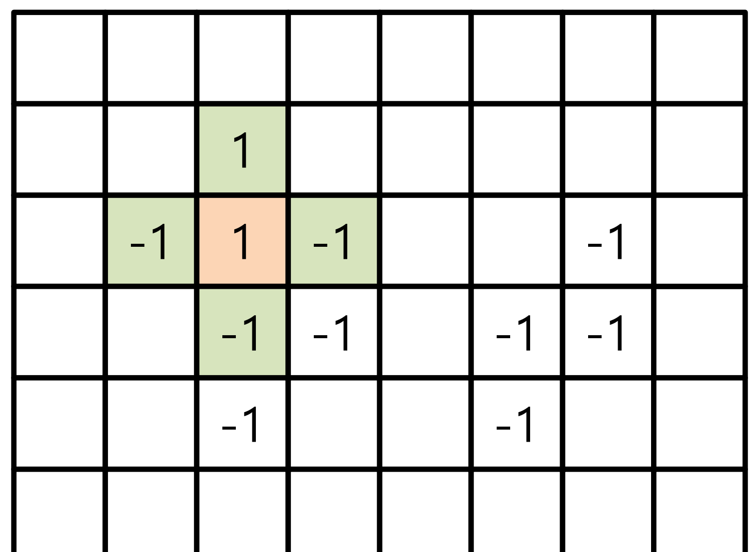
```
void search(L, label, r, c)
{
    L[r, c] = label;
    Nset = neighbors(r, c);  // Nset becomes the 4-neighborhood of [r, c]
    for each [r', c'] in Nset {
        if(L[r', c'] == -1)
            search(L, label, r', c');   // recursion
    }
}
```

Nset contains
north, west, east, south pixels
in that order



Recursive Labeling

```
void search(L, label, r, c)
{
    L[r, c] = label;
    Nset = neighbors(r, c);  // Nset becomes the 4-neighborhood of [r, c]
    for each [r', c'] in Nset {
        if(L[r', c'] == -1)
            search(L, label, r', c');   // recursion
    }
}
```



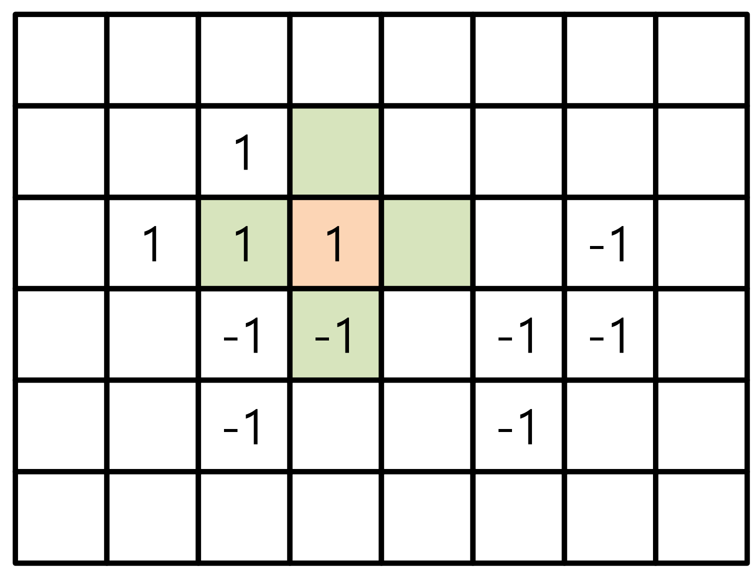
Recursive Labeling

```
void search(L, label, r, c)
{
    L[r, c] = label;
    Nset = neighbors(r, c);  // Nset becomes the 4-neighborhood of [r, c]
    for each [r', c'] in Nset {
        if(L[r', c'] == -1)
            search(L, label, r', c');   // recursion
    }
}
```

		1					
	1	1	-1			-1	
		-1	-1		-1	-1	
		-1			-1		

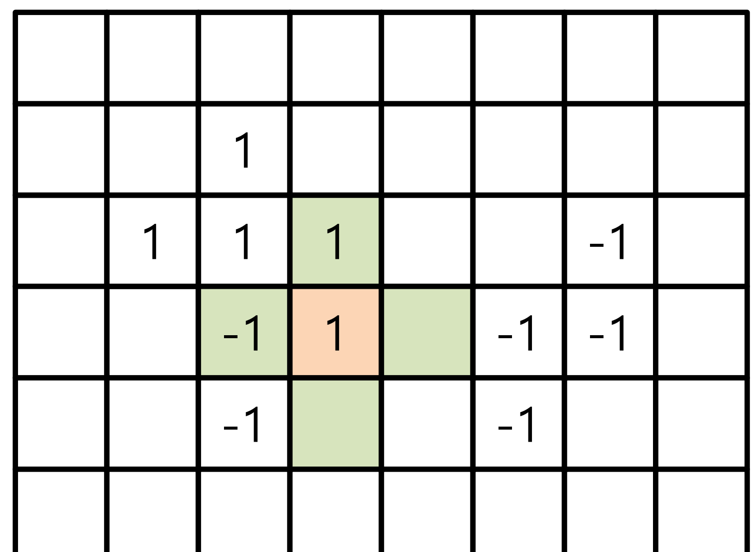
Recursive Labeling

```
void search(L, label, r, c)
{
    L[r, c] = label;
    Nset = neighbors(r, c);  // Nset becomes the 4-neighborhood of [r, c]
    for each [r', c'] in Nset {
        if(L[r', c'] == -1)
            search(L, label, r', c');   // recursion
    }
}
```



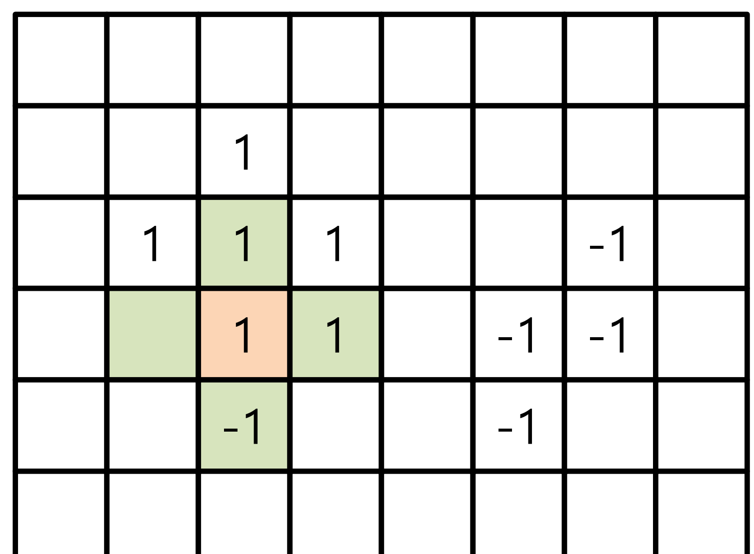
Recursive Labeling

```
void search(L, label, r, c)
{
    L[r, c] = label;
    Nset = neighbors(r, c);  // Nset becomes the 4-neighborhood of [r, c]
    for each [r', c'] in Nset {
        if(L[r', c'] == -1)
            search(L, label, r', c');   // recursion
    }
}
```



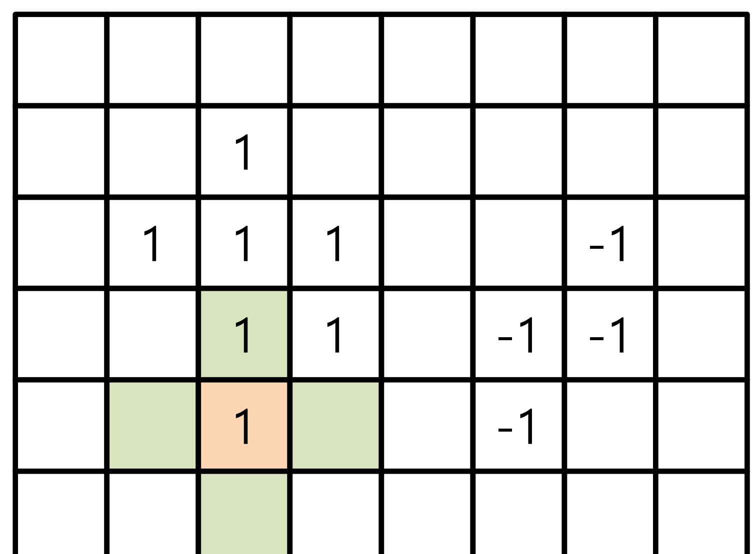
Recursive Labeling

```
void search(L, label, r, c)
{
    L[r, c] = label;
    Nset = neighbors(r, c);  // Nset becomes the 4-neighborhood of [r, c]
    for each [r', c'] in Nset {
        if(L[r', c'] == -1)
            search(L, label, r', c');   // recursion
    }
}
```



Recursive Labeling

```
void search(L, label, r, c)
{
    L[r, c] = label;
    Nset = neighbors(r, c);  // Nset becomes the 4-neighborhood of [r, c]
    for each [r', c'] in Nset {
        if(L[r', c'] == -1)
            search(L, label, r', c');   // recursion
    }
}
```



Recursive Labeling

```
void search(L, label, r, c)
{
    L[r, c] = label;
    Nset = neighbors(r, c);  // Nset becomes the 4-neighborhood of [r, c]
    for each [r’, c’] in Nset {
        if(L[r’, c’] == -1)
            search(L, label, r’, c’);   // recursion
    }
}
```

		1					
	1	1	1			2	
		1	1		-1	-1	
		1			-1		

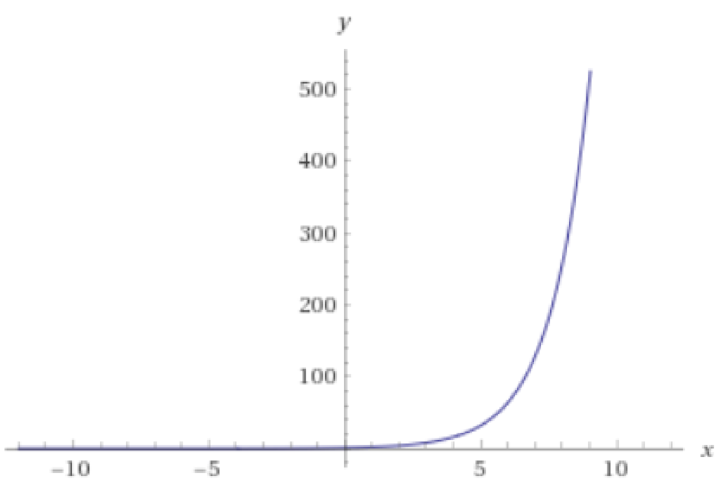
Review of Recursion

```
int FiboR (int n)
{
  if(n<=1) return 1;
  else
     return FiboR(n-1)+FiboR(n-2);
}
```

$O(2^n)$

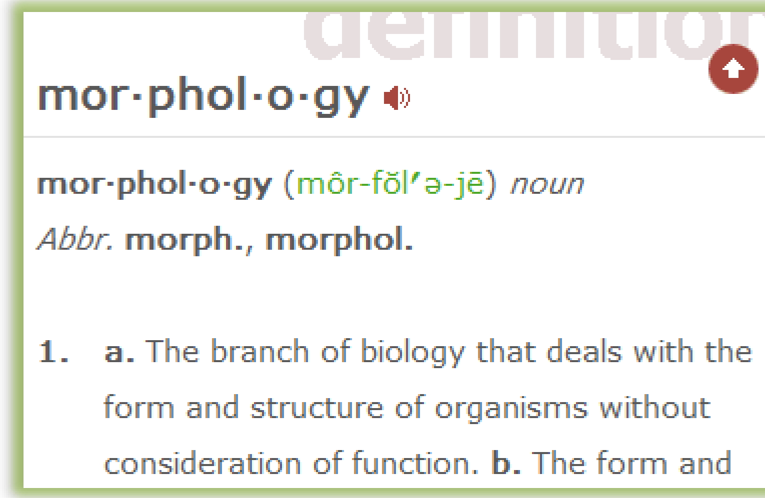
```
int FiboD (int n)
{
  if(n<=1) return 1;
  else{
    int *temp = new int[n+1];
    temp[0] = temp[1] = 1;
    for(int i=2; i<=n; i++)
        temp[i] = temp[i-1]+temp[i-2];

    int result = temp[n];
    delete temp;
    return result;
  }
}
```

$O(n)$


y
500
400
300
200
100
x
−10
−5
5
10

Binary Image Morphology

- Structuring elements
  - One pixel is denoted as its origin
- Basic operations
  - Translation
  - Dilation
  - Erosion
  - Closing
  - Opening

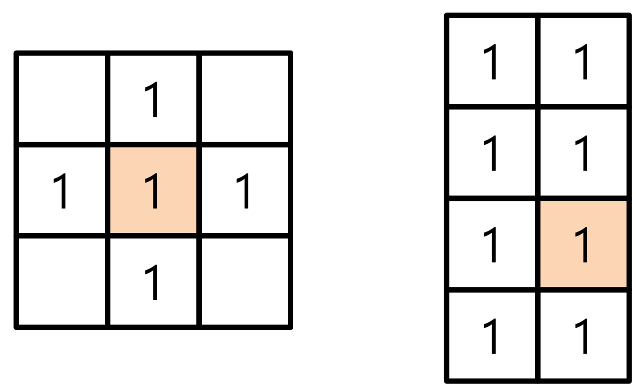

mor·phol·o·gy

mor·phol·o·gy (môr-fŏl′ə-jē) *noun*

Abbr. **morph., morphol.**

1. **a.** The branch of biology that deals with the form and structure of organisms without consideration of function. **b.** The form and

Ex) Structuring elements with their origins

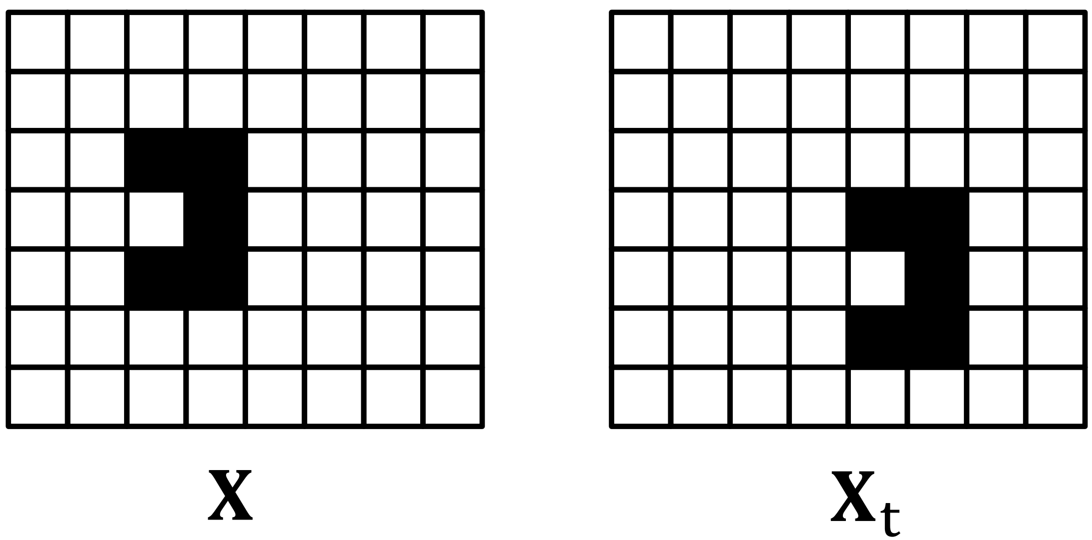


Translation

- The translation $\mathbf{X}_t$ of a set of pixels $\mathbf{X}$ by a position vector t

$$\mathbf{X}_t = \{x + t | x \in \mathbf{X}\}$$

  - In this and following definitions, sets contain the coordinates of 1 (black) pixels



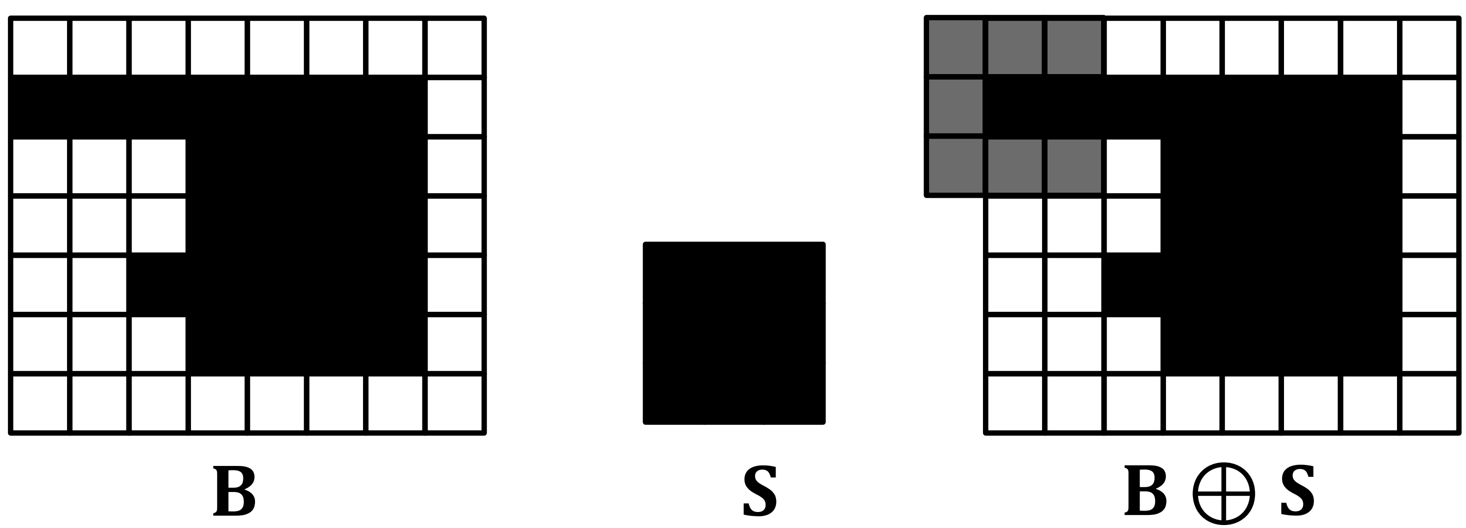
What is t?

Dilation

- The dilation of a binary image $\mathbf{B}$ by a structuring element $\boldsymbol{S}$

$$\mathbf{B} \oplus \boldsymbol{S} = \bigcup_{\mathrm{b} \in \mathbf{B}} \boldsymbol{S}_{\mathrm{b}}$$

  - The structuring element is put over each black pixel in $\mathbf{B}$
  - All the black pixels compose the dilation result.

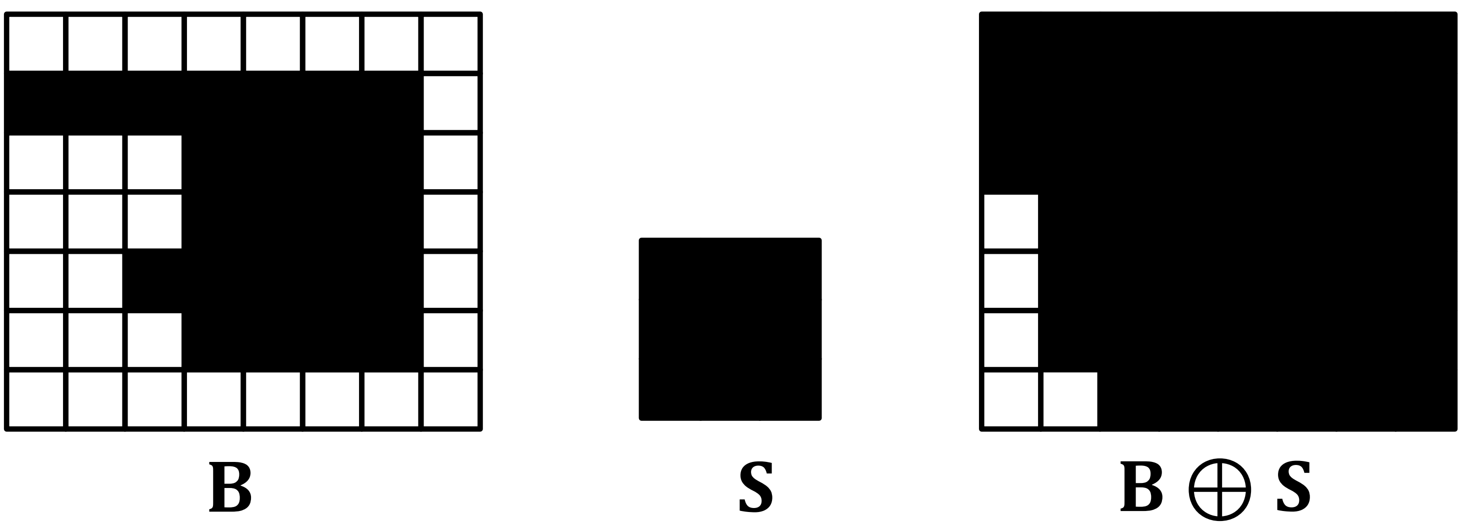


Dilation

- The dilation of a binary image $\mathbf{B}$ by a structuring element $\boldsymbol{S}$

$$\mathbf{B} \oplus \boldsymbol{S} = \bigcup_{\mathrm{b} \in \mathbf{B}} \boldsymbol{S}_{\mathrm{b}}$$

  – The structuring element is put over each black pixel in $\mathbf{B}$
  – All the black pixels compose the dilation result.



Erosion

- The erosion of a binary image $\mathbf{B}$ by a structuring element $\boldsymbol{S}$

$$\mathbf{B} \ominus \boldsymbol{S} = \{\mathrm{t} | \boldsymbol{S}_{\mathrm{t}} \subset \mathbf{B}\}$$

  – If the translated $\boldsymbol{S}_{\mathrm{t}}$ is wholly contained in $\mathbf{B}$, t is set black in the erosion result

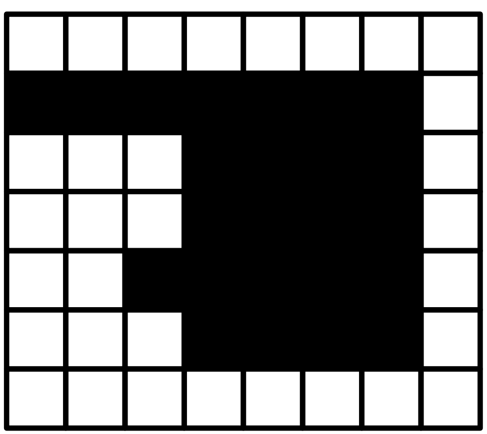
$\mathbf{B}$ $\boldsymbol{S}$ $\mathbf{B} \ominus \boldsymbol{S}$

Erosion

- The erosion of a binary image $\mathbf{B}$ by a structuring element $\boldsymbol{S}$

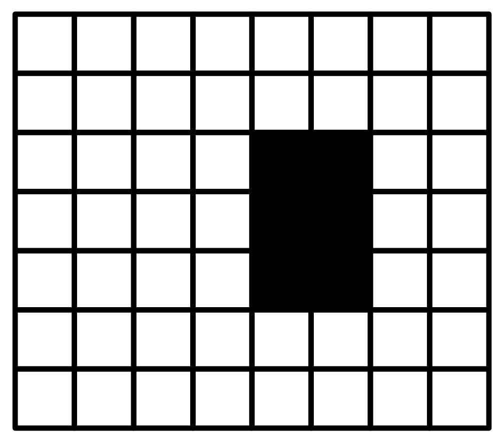
$$\mathbf{B} \ominus \boldsymbol{S} = \{\mathrm{t} | \boldsymbol{S}_{\mathrm{t}} \subset \mathbf{B}\}$$

  - If the translated $\boldsymbol{S}_{\mathrm{t}}$ is wholly contained in $\mathbf{B}$, t is set black in the erosion result



$\mathbf{B}$

$\boldsymbol{S}$

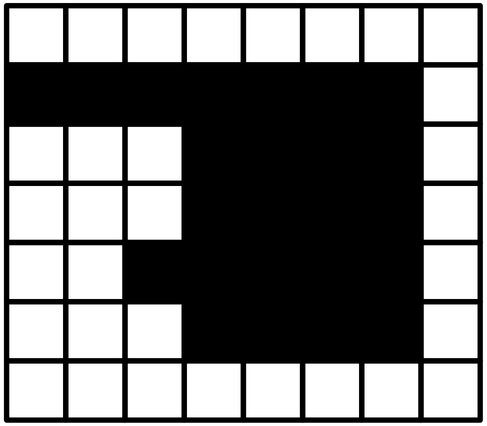


$\mathbf{B} \ominus \boldsymbol{S}$

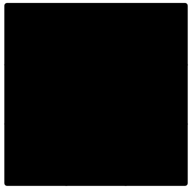
Closing

- The closing of a binary image **B** by a structuring element ***S***

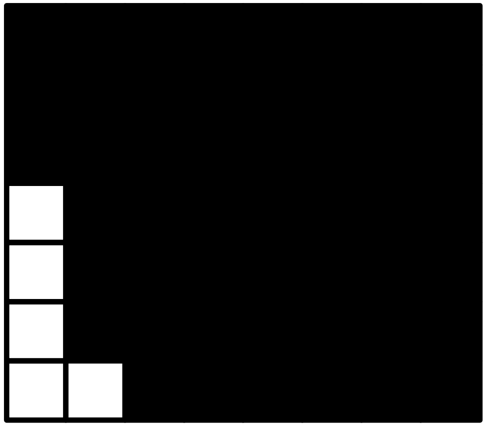
$$\mathbf{B} \bullet \boldsymbol{S} = (\mathbf{B} \oplus \boldsymbol{S}) \ominus \boldsymbol{S}$$



$\mathbf{B}$



$\boldsymbol{S}$

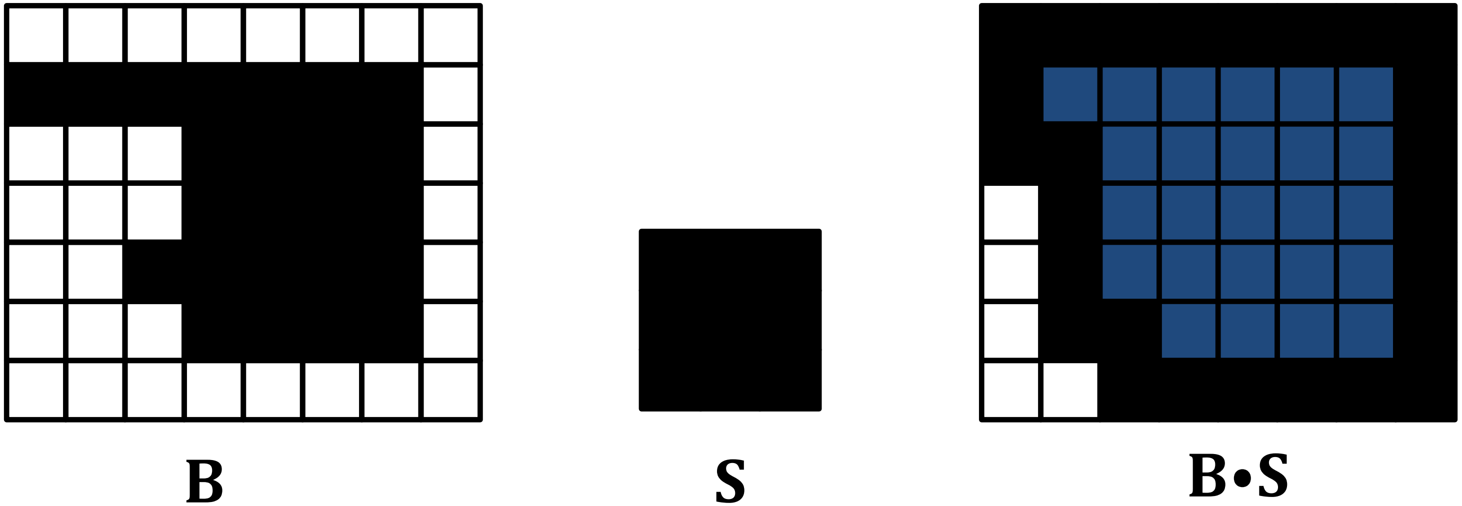


$\mathbf{B} \oplus \boldsymbol{S}$

Closing

- The closing of a binary image **B** by a structuring element ***S***

$$\mathbf{B} \bullet \boldsymbol{S} = (\mathbf{B} \oplus \boldsymbol{S}) \ominus \boldsymbol{S}$$

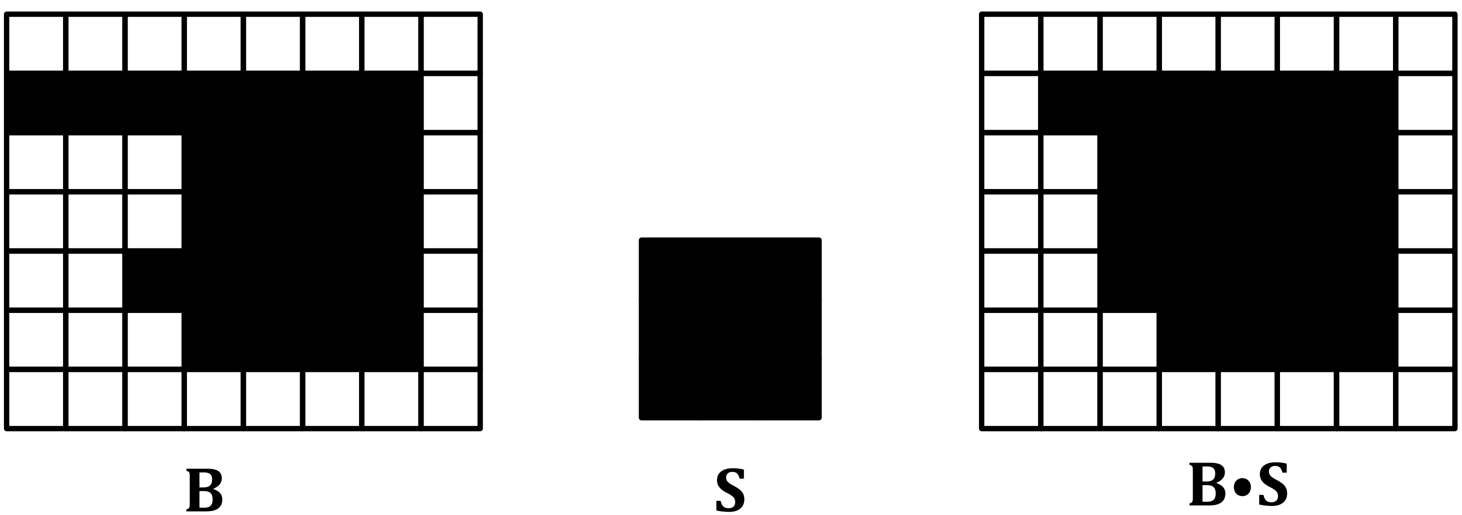


Closing

- The closing of a binary image **B** by a structuring element ***S***

$$\mathbf{B} \bullet \boldsymbol{S} = (\mathbf{B} \oplus \boldsymbol{S}) \ominus \boldsymbol{S}$$

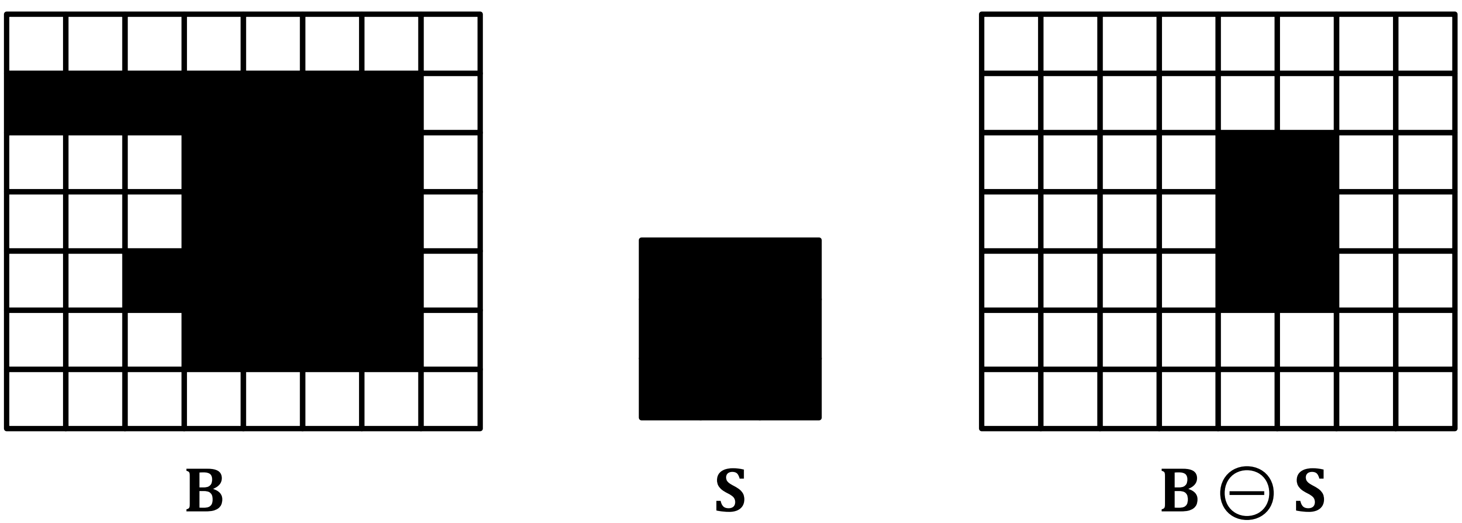
  - Ignoring boundary effects, the closing makes the input bigger
  - The closing fills tiny gaps in the input image



Opening

- The opening of a binary image **B** by a structuring element ***S***

$$\mathbf{B} \circ \boldsymbol{S} = (\mathbf{B} \ominus \boldsymbol{S}) \oplus \boldsymbol{S}$$



Opening

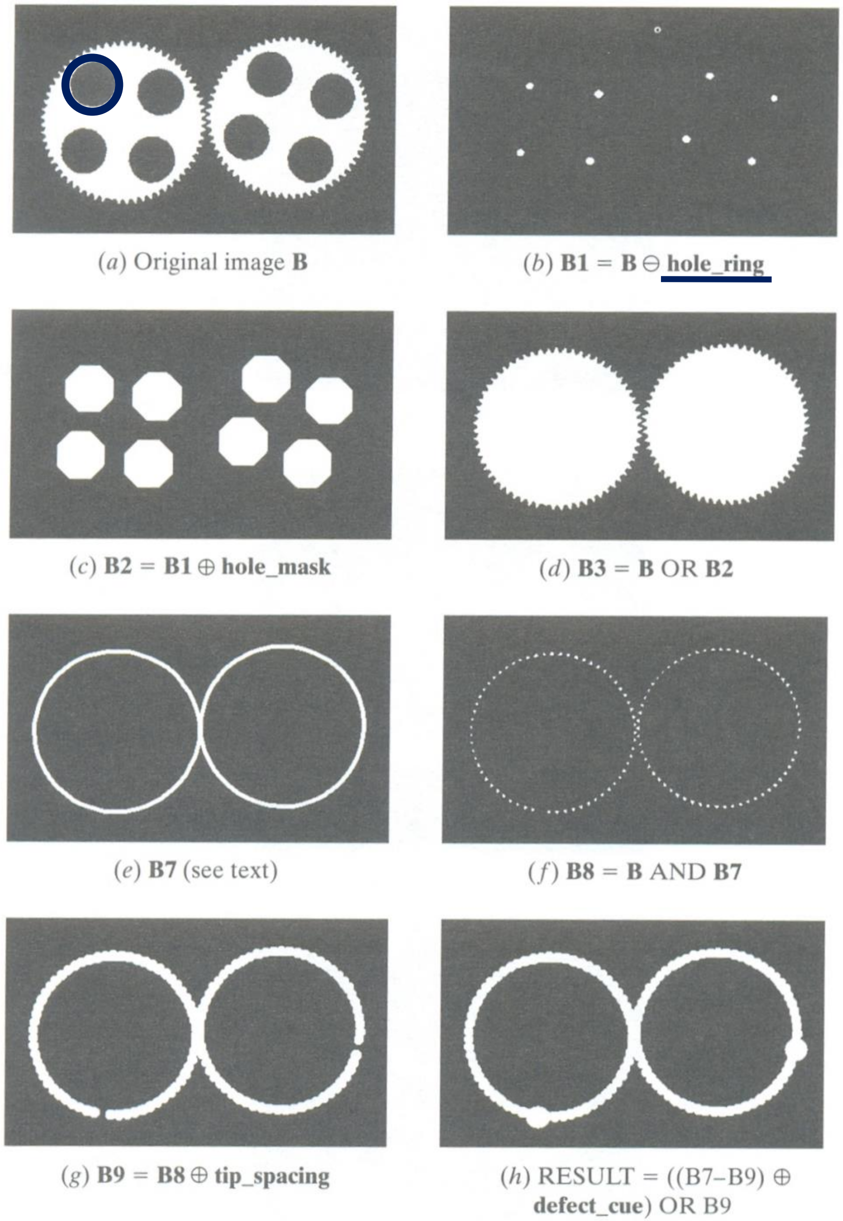
- The opening of a binary image $\mathbf{B}$ by a structuring element $\boldsymbol{S}$

$$\mathbf{B} \circ \boldsymbol{S} = (\mathbf{B} \ominus \boldsymbol{S}) \oplus \boldsymbol{S}$$

  - The opening makes the input smaller
  - The opening erases tiny components or thin extrusions

$\mathbf{B}$ $\boldsymbol{S}$ $\mathbf{B} \circ \boldsymbol{S}$

Application: Gear-Tooth Inspection



- B7
  - Open B3 to remove the teeth (B4)
  - Dilate B4 to make it larger (B5)
  - Dilate B5 to make it even larger (B6)
  - B7 = B6 – B5

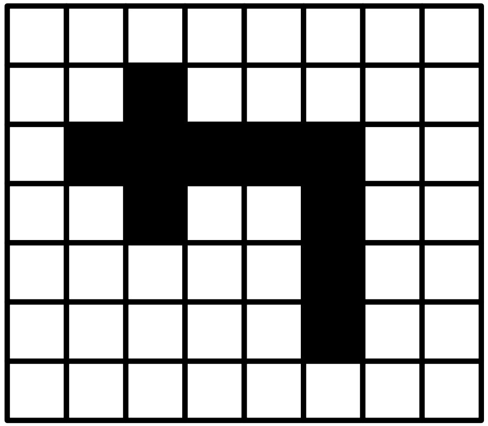
Region Properties

- Let *R* denote a region or the set of its pixel coordinates

- Area $$A = \sum_{(r,c)\in R} 1$$

- Centroid $(\bar{r}, \bar{c})$

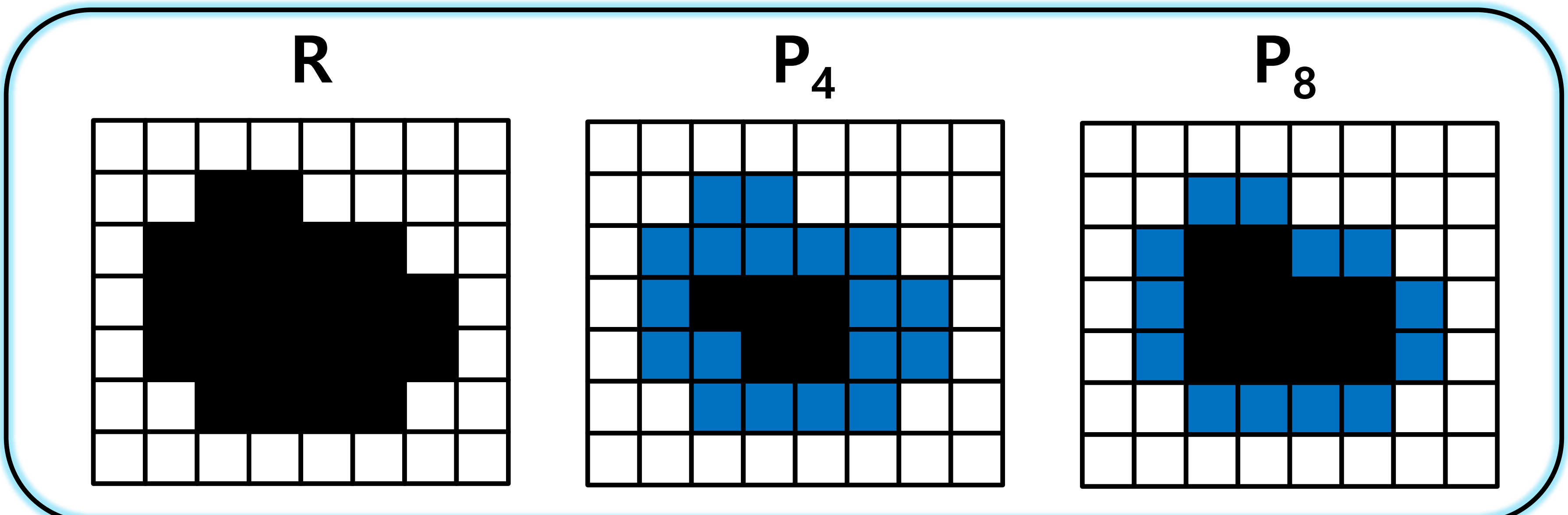
$$\bar{r} = \frac{1}{A}\sum_{(r,c)\in R} r \quad \text{and} \quad \bar{c} = \frac{1}{A}\sum_{(r,c)\in R} c$$



Area = 10
Centroid = (2.6, 3.4)

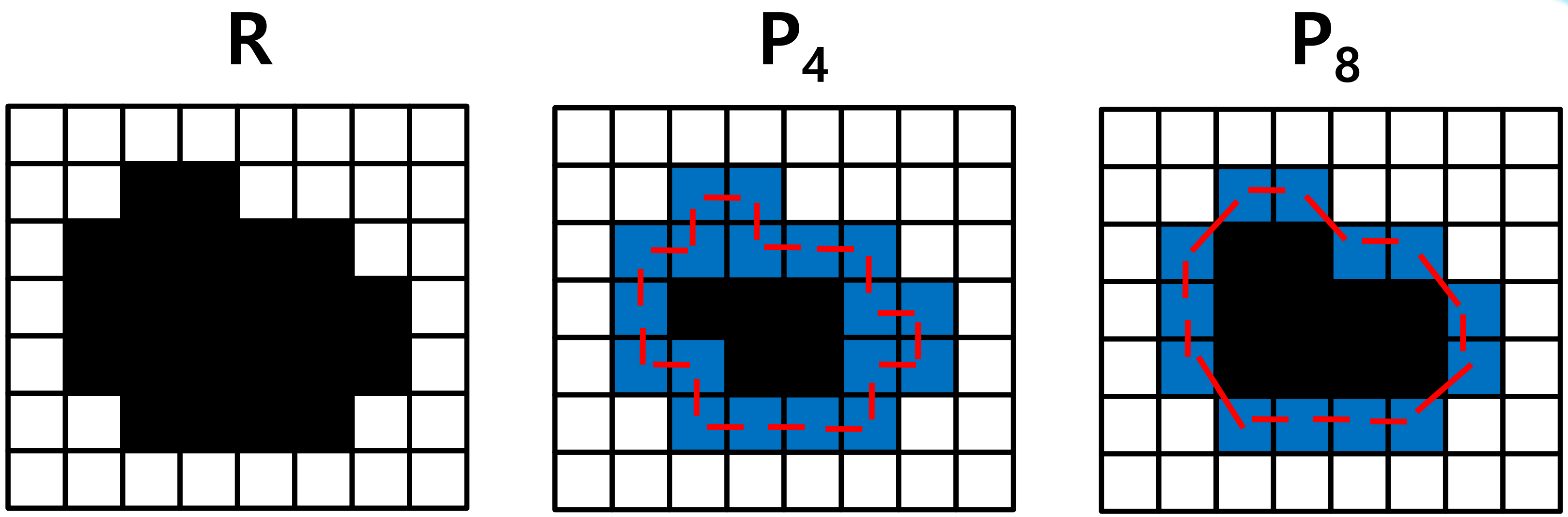
Region Properties

- Perimeter
  - 4-connected perimeter
    $$P_4 = \{(r,c) \in R \mid N_8(r,c) - R \neq \phi\}$$
  - 8-connected perimeter
    $$P_8 = \{(r,c) \in R \mid N_4(r,c) - R \neq \phi\}$$



Region Properties

- Perimeter length



Length = 18

Length = $8+5\sqrt{2}=15.07$

Region Properties

- Haralick's circularity measure

$$C = \frac{\mu}{\sigma} = \frac{\frac{1}{K}\sum_{k=0}^{K-1}\left\|(r_k, c_k) - (\bar{r}, \bar{c})\right\|}{\left(\frac{1}{K}\sum_{k=0}^{K-1}\left[\left\|(r_k, c_k) - (\bar{r}, \bar{c})\right\| - \mu\right]^2\right)^{1/2}}$$

  - (r_k, c_k): border pixels on the perimeter
  - K: the number of border pixels
  - C is bigger as the region is more circular

Region Properties

- Spatial moments
  - Second-order row moment

$$\mu_{rr} = \frac{1}{A} \sum_{(r,c) \in R} (r - \bar{r})^2$$

  - Second-order column moment

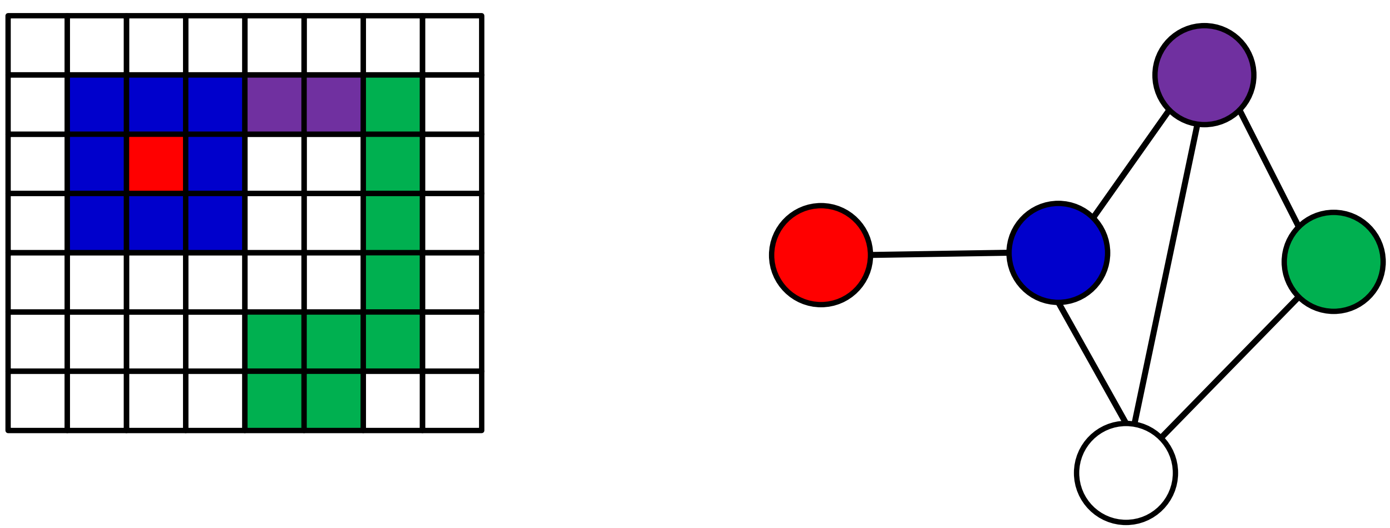
$$\mu_{cc} = \frac{1}{A} \sum_{(r,c) \in R} (c - \bar{c})^2$$

  - Second-order mixed moment

$$\mu_{rc} = \frac{1}{A} \sum_{(r,c) \in R} (r - \bar{r})(c - \bar{c})$$

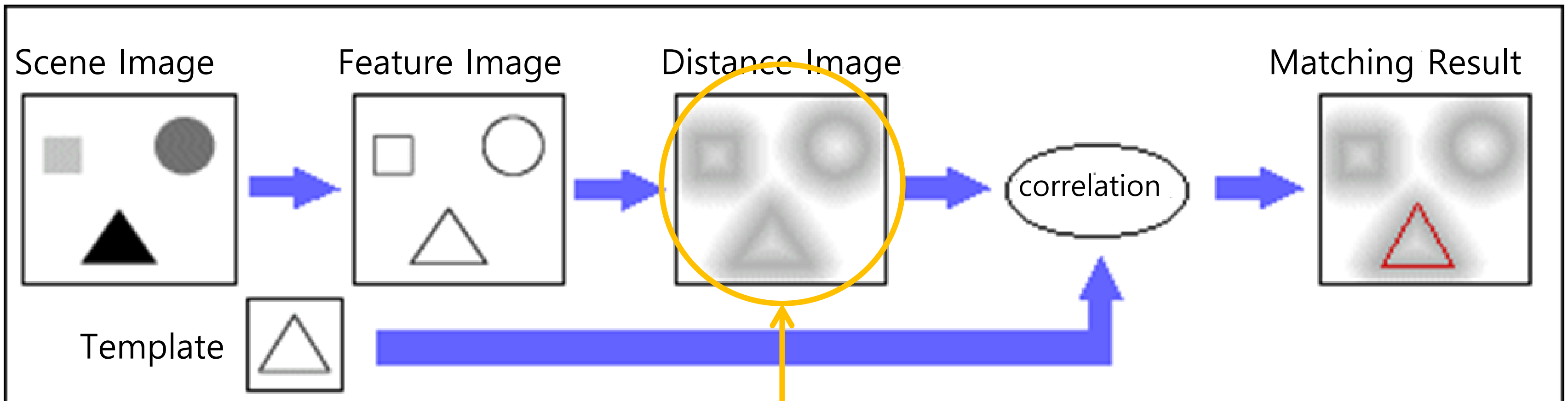
Region Adjacency Graph (RAG)

- Two regions **A** and **B** are adjacent if a pixel in **A** neighbors a pixel in **B**
- In RAG, each node represents a region of the image, and an edge connects two nodes if the corresponding regions are adjacent



Distance Transform

- Chamfer matching (binary shape matching)



Each pixel value denotes the distance to the nearest feature pixel

DT allows more variability between a template and an object of interest in the image because a distance image provides a smooth cost function.

D. M. Gavrila, "A Bayesian, exemplar-based approach to hierarchical shape matching," IEEE Trans. Pattern Anal. Mach. Intell., Vol.29, No.8, pp.1408-1421, 2007.

Distance Transform

- Distance between $p = (x_1, y_1)$ and $q = (x_2, y_2)$
  - Manhattan distance
$$d_1(p, q) = |x_1 - x_2| + |y_1 - y_2|$$
  - Euclidean distance
$$d_2(p, q) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

- We use d_1 in this application

- Distance transform
  - For p with $\mathbf{B}(p) = 1$
$$D(p) = \min_{\mathbf{B}(q)=0} d_1(q, p)$$
  - Compute the distance to the nearest background pixel

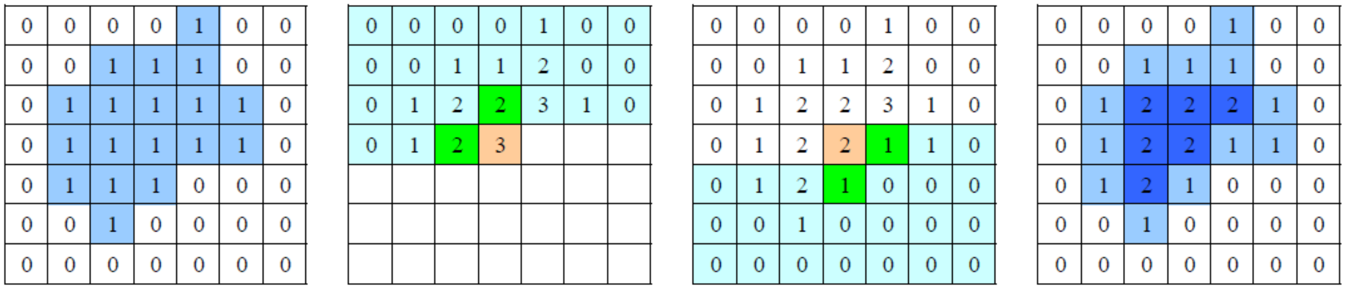
Distance Transform

- Example

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

0	0	0	0	1	0	0
0	0	1	1	1	0	0
0	1	2	2	2	1	0
0	1	2	2	1	1	0
0	1	2	1	0	0	0
0	0	1	0	0	0	0
0	0	0	0	0	0	0

Distance Transform



(a) (b) (c) (d)

- Procedure: Two sweeps for nonzero pixels only
  - (b) forward sweep

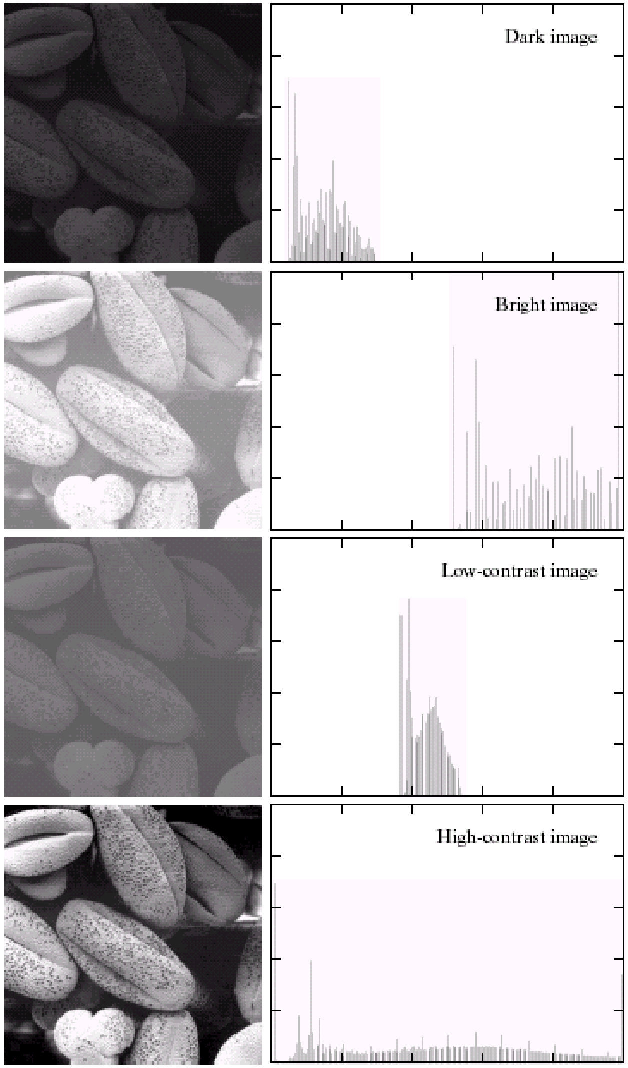
    $$D(r,c) = \min\{1 + D(r-1,c), 1 + D(r,c-1)\}$$

  - (c) backward sweep

    $$D(r,c) = \min\{D(r,c), 1 + D(r+1,c), 1 + D(r,c+1)\}$$

Thresholding Gray-Scale Images to Make Binary Images

- The histogram h of an image **I** is a function, given by
  - h(m) = the number of pixels in I which have value m



Thresholding Gray-Scale Images to Make Binary Images