**[A] Pattern pre-processing algorithms**

**(1) Smoothing and cleaning algorithms**

Smoothing is used to straighten the edges and remove noises from an image. Smoothing and noise removal can be done by a filtering mask, where using an algorithm, the output image is determined by matching values of the pixels in the neighborhood of the corresponding input pixel with the selected filter mask. Single pixel noises and one pixel wide noises can be removed using smoothing.

**Smoothing using 3x3 linear filter mask**

**Algorithm:**

**Input**:

* + A 2 dimensional array of integer, **‘Binary Matrix’** with binary values (0, 1) from **Binary Image**
  + A 2 dimensional array of integers, **‘Filter Mask’** with 4 filter values

[ = , = , = ] [ x , = , = ] [ x , x , x ] [ = , = , x ]

[ = , **T** , = ] [ x , **T** , = ] [ = , **T** , = ] [ = , **T** , x ]

[ x , x , x ] [ x , = , = ] [ = , = , = ] [ = , = , x ]

**Output**: A 2 dimensional array of integer, **‘Result Matrix’** with pixels matching the smoothing condition of 3x3 filter mask with filter value 4.

**pixelChange** <- **false**

**for** each pixel from lower right corner of **Binary** **Matrix**

**target** <- pixel in the center of **Binary** **Matrix**

**if** neighbors of target pixel are aligned with **‘=’** in the **Filter Mask**

**set target<-** match the pixel value denoted by **‘=’**

**pixelChange<- true**

**end if**

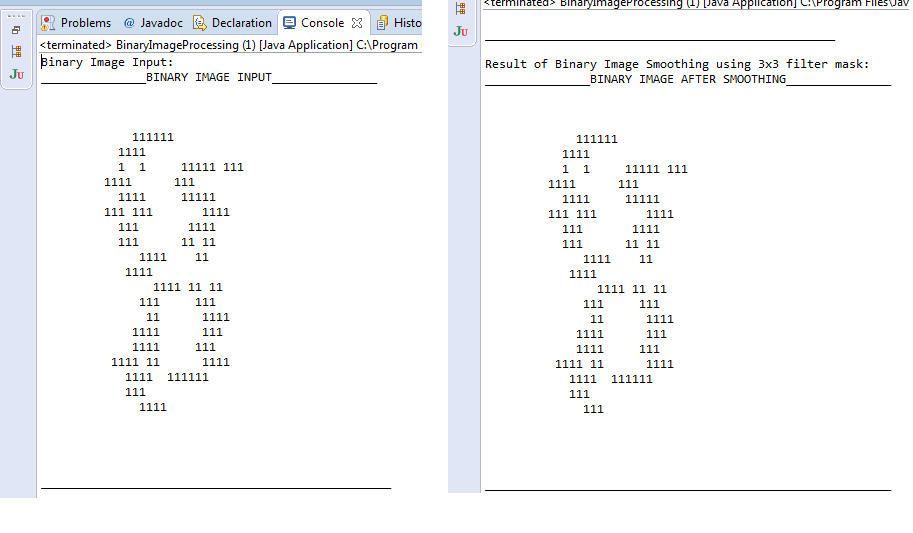
**end for**

**return Result Matrix**

**Results:**

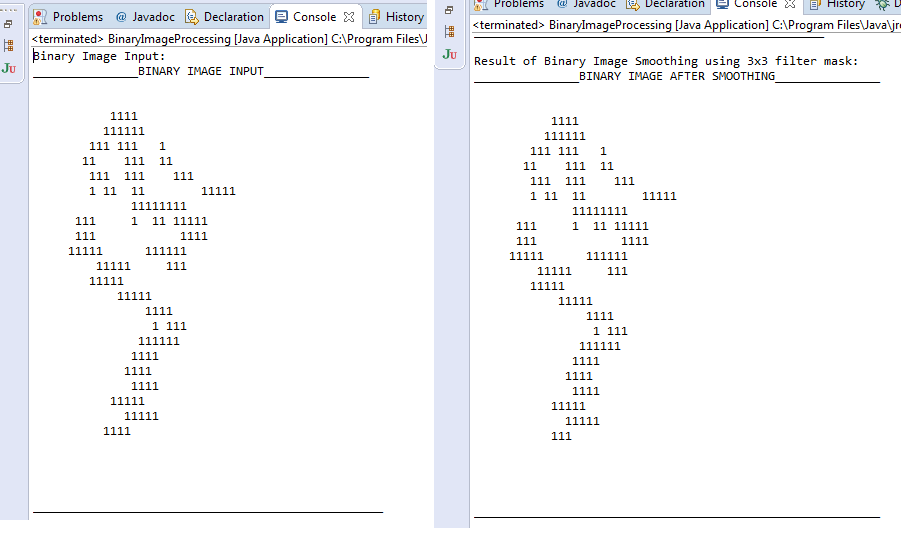
**INPUT: Image 8 OUTPUT: Image 8 after smoothing using 3x3**

**Filter**

****

**INPUT: Image 9 OUTPUT: Image 9 after smoothing using 3x3**

**Filter**

****

**(2) Filling: 4 Neighbors filling**

**4 Neighbors Filling** algorithm is used to fill the pixels which are blank in the image with the 4 immediate neighbors (East, West, North, and South) of Target Pixel. 4 Neighbors filling uses its immediate neighbors for the filling of pixels.ie, if Target pixels immediate 3 neighbors (East or West or North or South) are filled, then the Target pixel is filled else the Target pixel is not changed.

|  |  |  |
| --- | --- | --- |
|  | **N** |  |
| **W** | **T** | **E** |
|  | **S** |  |

**Algorithm:**

**Input:** A 2 dimensional array of integer, **‘Binary Matrix’** with binary values (0, 1) from **smoothing process**

**Output**: A 2 dimensional array of integer, **‘Result Matrix’** with pixels matching **4 neighbors filling** conditions

**for** each pixel from upper left corner of **Binary** **Matrix**

**Target Pixel** <- pixel in the center of **Binary** **Matrix**

**Left Pixel <-** left pixel adjacent to the **Target Pixel**

**Right Pixel <-** right pixel adjacent to the **Target Pixel**

**Upper Pixel <-** Upper pixel adjacent to the **Target Pixel**

**Lower Pixel <-** lower pixel adjacent to the **Target Pixel**

**Target Pixel <- Target Pixel || ( Upper Pixel & Lower Pixel & ( Left Pixel |**

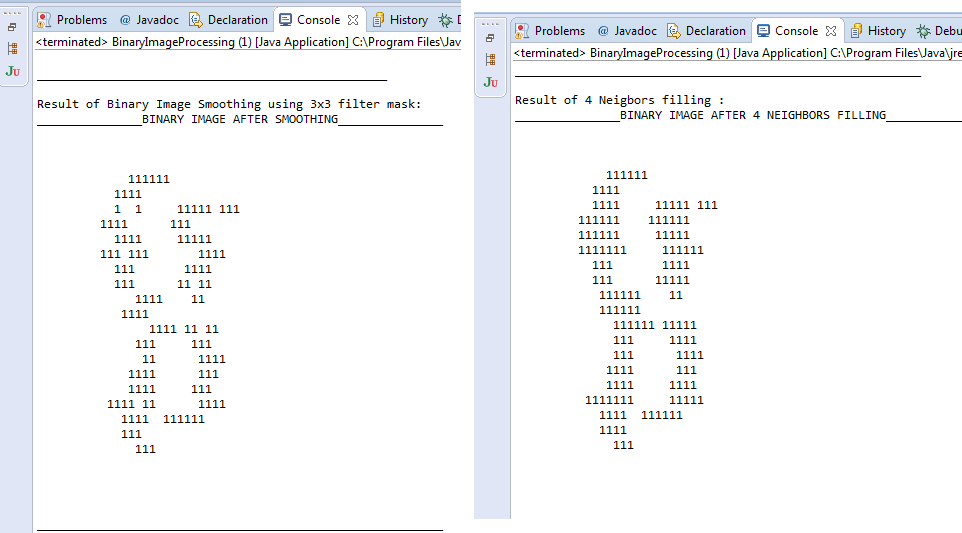
**Right Pixel ) ) || ( Left Pixel & Right Pixel & ( Upper Pixel | Lower Pixel ) ) ;**

**end for**

**return Result Matrix**

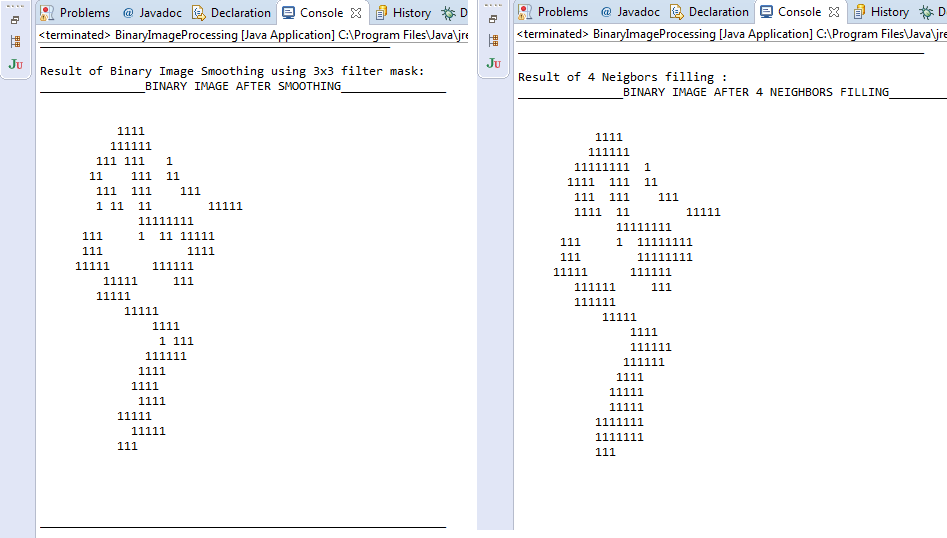
**Results:**

**INPUT: Image 8 after smoothing OUTPUT: Image 8 after 4 neighbors filling**

****

**Results:**

**INPUT: Image 9 after smoothing OUTPUT: Image 9 after 4 neighbors filling**

****

**(2) Filling: 8 neighbors filling**

**8 Neighbors Filling** algorithm is used to fill the pixels which are blank in the image with the 4 immediate neighbors (East, West, North, and South) and 4 diagonal neighbors (Northeast, Northwest, Southeast and Southwest) of Target Pixel. 8 Neighbors filling uses its diagonal neighbors for the filling of pixels.ie, if Target pixels diagonal pixels are filled then the Target pixel is filled else the Target pixel is not changed.

|  |  |  |
| --- | --- | --- |
| **NW** | **N** | **NE** |
| **W** | **T** | **E** |
| **SW** | **S** | **SE** |

**Algorithm:**

**Input:** A 2 dimensional array of integer, **‘Binary Matrix’** with binary values (0, 1) from **4 neighbors filling** process

**Output**: A 2 dimensional array of integer, **‘Result Matrix’** with pixels matching **8 neighbors filling** conditions

**for** each pixel from upper left corner of **Binary** **Matrix**

**Target Pixel** <- pixel in the center of **Binary** **Matrix**

**Left Pixel <-** left pixel adjacent to the **Target Pixel**

**Right Pixel <-** right pixel adjacent to the **Target Pixel**

**Upper Pixel <-** Upper pixel adjacent to the **Target Pixel**

**Lower Pixel <-** lower pixel adjacent to the **Target Pixel**

**Left Upper Pixel <-** left upper pixel adjacent to the **Target Pixel**

**Left Lower Pixel <-** left lower pixel adjacent to the **Target Pixel**

**Right Upper Pixel <-** right upper pixel adjacent to the **Target Pixel**

**Right Lower Pixel <-** right lower adjacent to the **Target Pixel**

**Target Pixel <- Target Pixel**

**|| Left Upper Pixel && Right Lower Pixel**

**|| Upper Pixel && Lower Pixel**

**|| Right Upper Pixel && Left Lower Pixel**

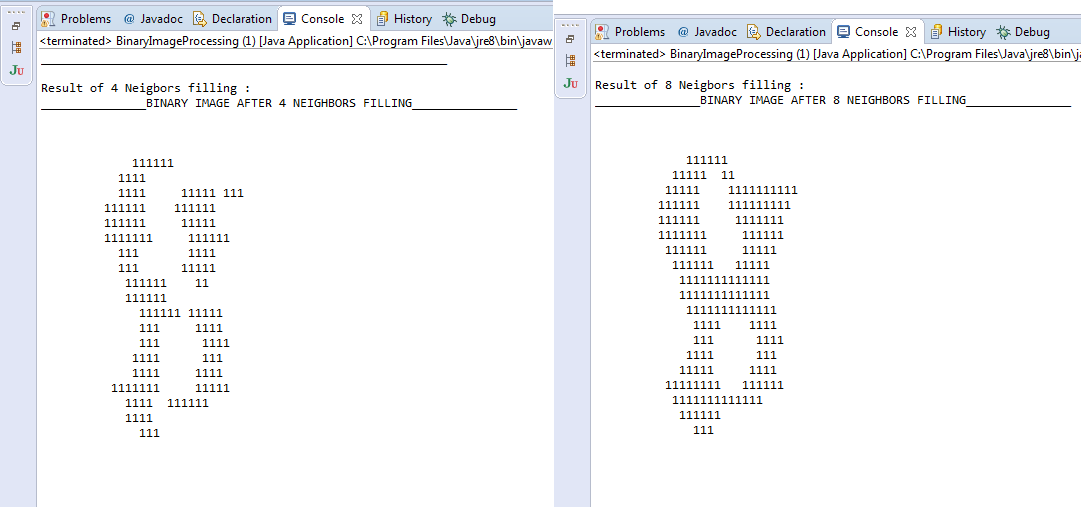
**|| Left Pixel && Right Pixel**

**end for**

**return Result Matrix**

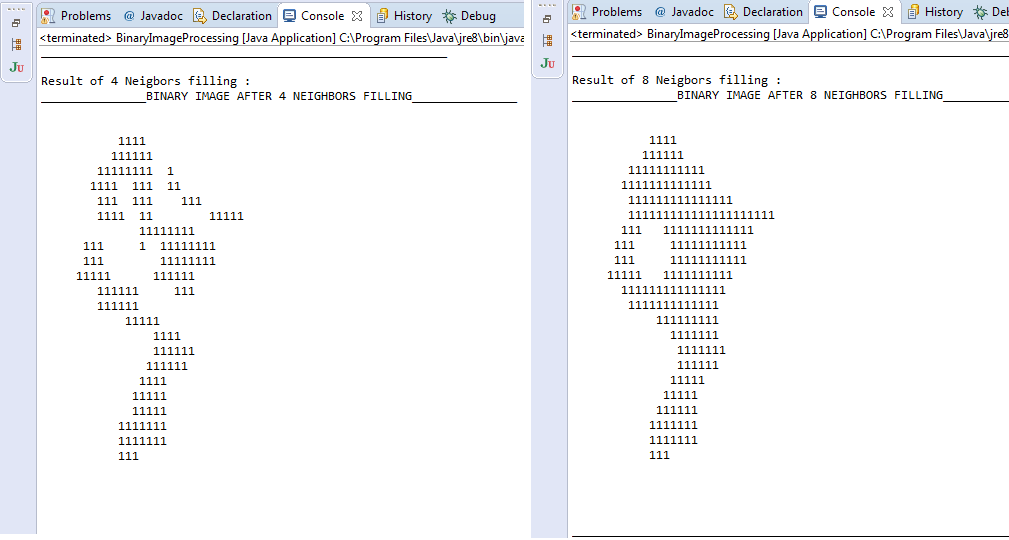
**Results:**

**INPUT: Image 8 after 4 neighbors filling OUTPUT: Image 8 after 8 neighbors filling**

****

**Results:**

**INPUT: Image 9 after 4 neighbors filling OUTPUT: Image 9 after 8 neighbors filling**

****

**[A] (ii) Normalization**

**Size Normalization:** The goal of the size normalization is to reduce the shape of the original image to a standard pre-defined size. Image length and width are reduced to a standard pane with a pre-defined length and width.

Using linear normalization method, width and height (W1 and H1) of original character is reduced normalized image size (W2 and H2).Using backward mapping process, From left top corner of Image each pixel is scanned and index of each pixel is multiplied with the normalization ratio(alpha – length, beta- Width) and the pixel index is set.

**Algorithm:**

**Input:** A 2 dimensional array of integer, **‘Binary Matrix’** with binary values (0, 1) from **8 neighbors filling** process

**Output**: A 2 dimensional array of integer, **‘Normalized Result Matrix’** with **Normalized row size** and **Normalized Column size,** after the Size Normalization

**Set:**

**Alpha <-** Normalization ratio for length of the Binary Image

**Beta <-** Normalization ratio for breadth of the Binary Image

**for** each **X** value from upper corner of **Binary** **Matrix**

**for** each **Y** value from upper corner of **Binary** **Matrix**

**X’ Index <- X Index / Alpha**

**Y’ Index <- Y Index / Beta**

**Result Matrix( X’ Index, Y’ Index) <- Binary Matrix( X Index, Y Index)**

**end for**

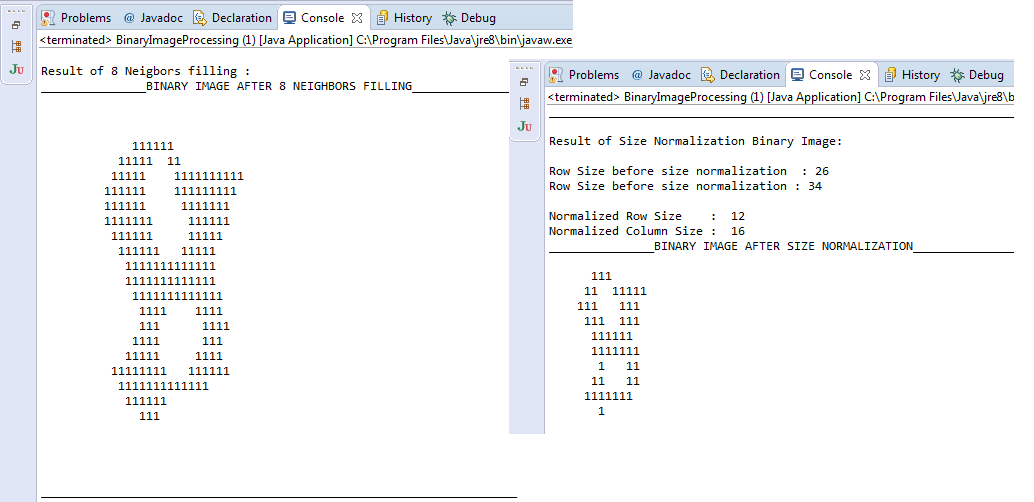
**end for**

**return Result Matrix**

**Results:**

**Size reduced to 50%**

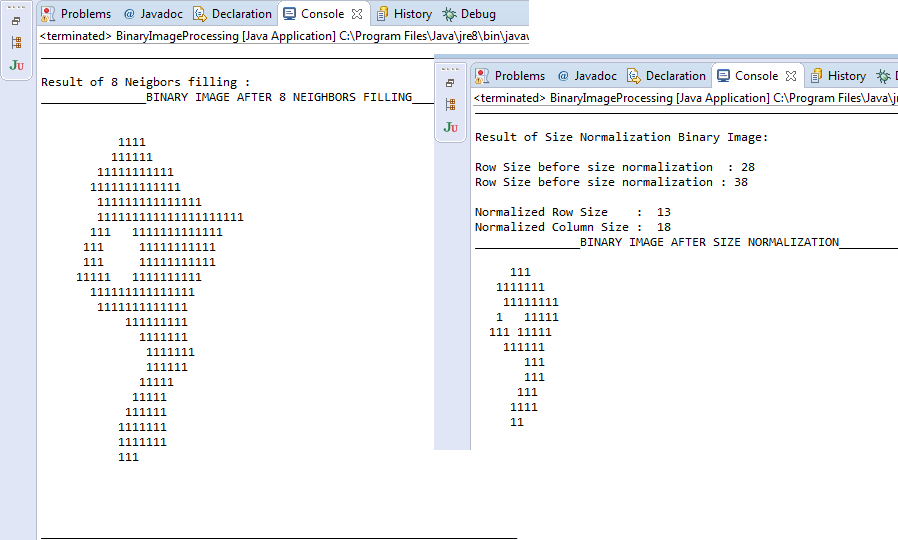
**INPUT: Image 8 after 8 neighbor filling OUTPUT: Image 8 after Size Normalization**

****

**Results:**

**Size reduced to 50%**

**INPUT: Image 8 after 8 neighbor filling OUTPUT: Image 8 after Size Normalization**

****

**[A] (III) Center of Gravity**

**Center of Gravity** is the point around which the black pixels (1’s in binary image) is equally distributed.

Center of Gravity can be calculated by sum of the pixels in the region each multiplied by its value, divided by the sum of these values.

|  |  |  |
| --- | --- | --- |
| **0** | **1** | **0** |
| **1** | **COG** | **1** |
| **0** | **1** | **0** |

**Algorithm:**

**Input:** A 2 dimensional array of integer, **‘Binary Matrix’** with binary values (0, 1)

**Output**: A **Pixel(X Index, Y Index)** which denotes the center of gravity for the Binary Image

Set: **X Index Sum** <- 0

**Y Index Sum** <- 0

**Pixel Count** <- 0

**for** each **X** value from upper corner of **Binary** **Matrix**

**for** each **Y** value from upper corner of **Binary** **Matrix**

**if Binary Matrix(X, Y)** is a **true**

set: **X Index Sum = X Index Sum + X**

**Y Index Sum = Y Index Sum + Y**

Increment **Pixel Count**

**end if**

**end for**

**end for**

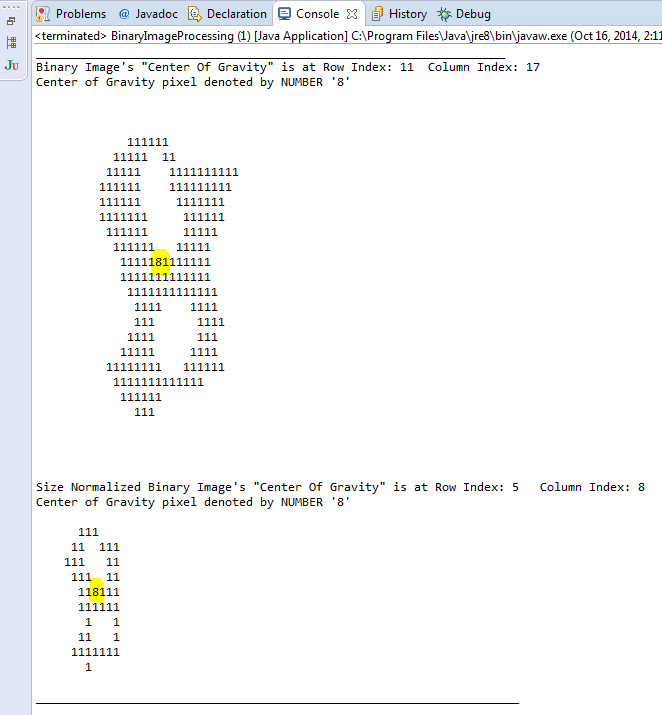
**COG X Index = X Index Sum / Pixel Count**

**COG Y Index = Y Index Sum / Pixel Count**

return **Pixel( COG X Index, COG Y Index)**

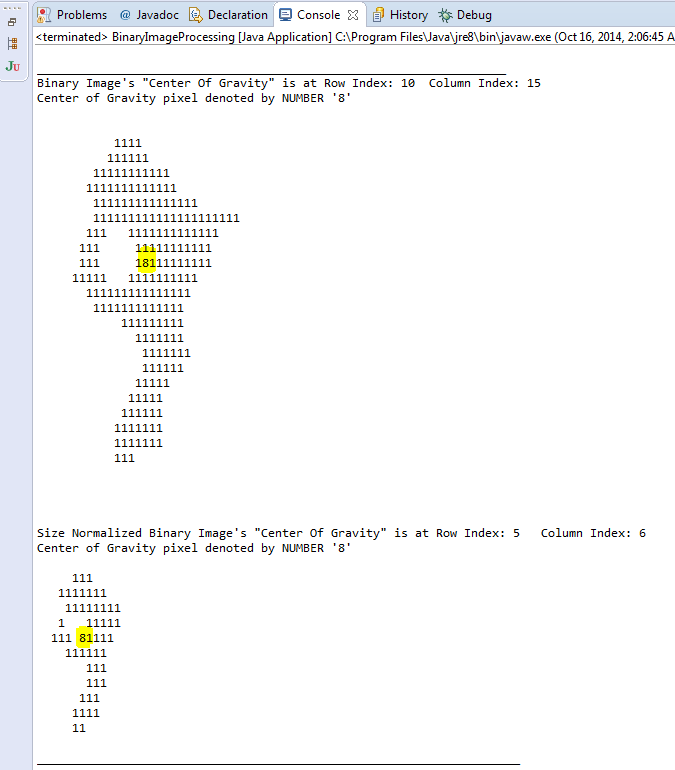
**Results:**

**OUTPUT: Image 8 & Normalized Image 8 with Center of Gravity**

****

**Results:**

**OUTPUT: Image 9 & Normalized Image 9 with Center of Gravity**

****

**(B) Skeleton extraction**

**Skeletonizing** is a process for reducing foreground regions (number of 1's) in a binary image to a skeletal remnant that largely preserves the extent and connectivity of the original region (binary image of size 1) while removing most of the original foreground pixels.

**Zhang-Suen Skeletonizing**

This skeletonizing algorithm repeatedly removes foreground pixels from a region in a binary image until skeletal remnant is present. **Zhang-Suen Skeletonizing** uses two iterations algorithm to remove the foreground pixels. The 1st pass checks for the 8-neighbors of target pixel, which uses southeast boundary and northwest corner point and checks if the pixel can be removed without disturbing the ideal skeleton. The 2nd pass uses the same procedure from northwest boundary points and southeast corner pixel. Iterations are done until there are no changes of pixels are detected in the two passes in the binary image.

**Algorithm:   
Input:** A 2 dimensional array of integer, **‘Binary Matrix’** with binary values (0, 1) from Size Normalization

**Output**: 1 pixel size width skeletonized 2 dimensional array of integer, **‘Result Matrix’**

|  |  |  |
| --- | --- | --- |
| **P8** | **P1** | **P2** |
| **P7** | **P** | **P3** |
| **P6** | **P5** | **P4** |

**A** <- Binary Image {0, 1}

**for** each **X** value from upper corner of **Binary** **Matrix A**

**for** each **Y** value from upper corner of **Binary** **Matrix A**

**P<-** pixel in the center of **Binary** **Matrix A**

**P1, P2, P3, P4, P5, P6, P7, P8** be the 8-neighbors of Pixel **P**

**B (P) <-** number of non-zero 8-neighbors pf **P**

**A (P) <-** number of zero-to-one transitions in the sequence of

P1->P2->P3->P4->P5->P6->P7->P8->P1

i**f**

**2<= B (P) <= 6 and A(P) == 1 and P1.P3.P5 == 0 and P3.P5.P7 == 0**

then remove Pixel **P**

**end if**

**else if**

**2<= B (P) <= 6 and A(P) == 1 and P1.P3.P7 == 0 and P1.P5.P7 == 0**

then remove Pixel **P**

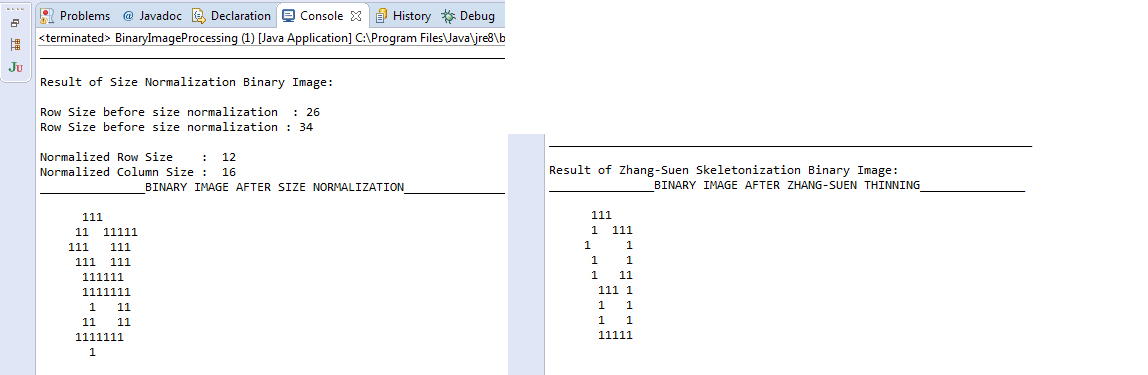
**end if**

**end for**

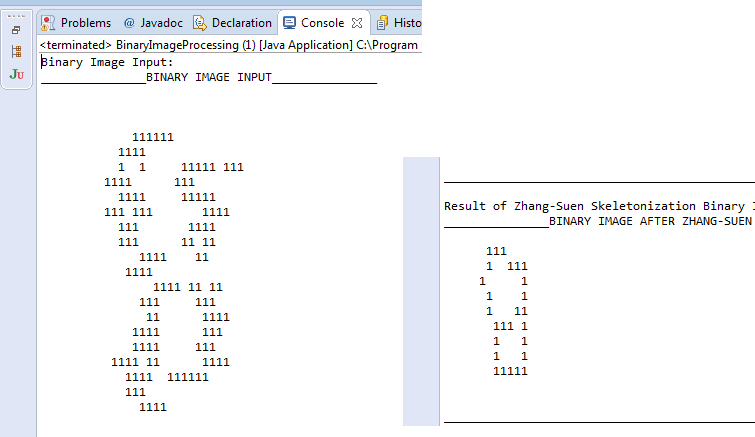
**end for**

**Results of Image 8**

**Output: Image 8 after Zhang-Suen Skeletonizing**

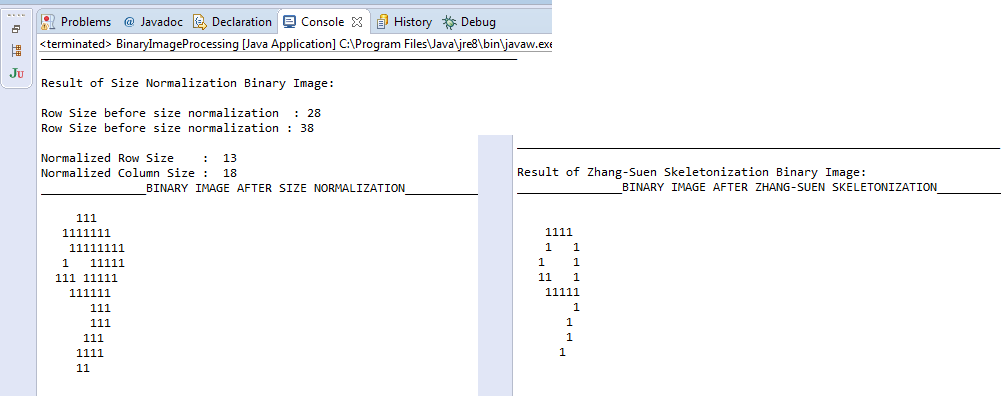
****

**Output: Input Image 8 and Image 8 after Zhang-Suen Skeletonizing**

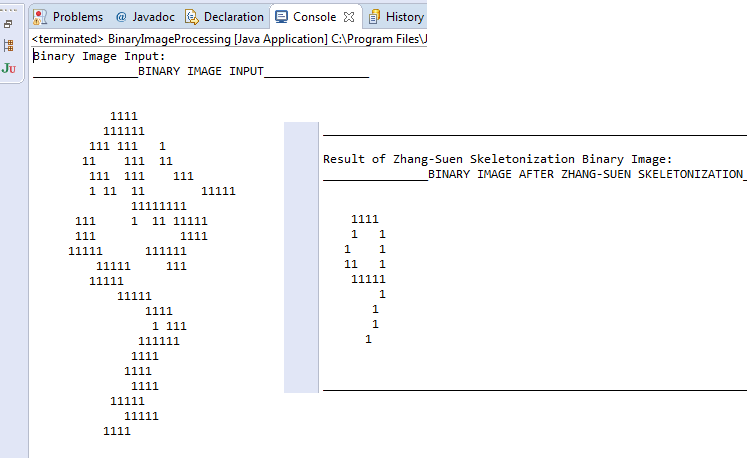
****

**Result of Image 9**

**Output: Image 9 after Zhang-Suen Skeletonizing**

****

**Output: Input Image 9 and Image 9 after Zhang-Suen Skeletonizing**

****