| | | | | | | | | |

-----1-------2--------3--------4--------5-------6--------7-------8-------9----- = -----------

| 13 | 11 | 14 | 10 | 13 | 13 | 13 | 15 | 6 | 112

CV Spring 2021 Exam I (Part 1 - Essay questions) Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**\*\*\* Reminding you that this exam is a closed book exam. Rules are given in the online exam instruction posted in google classroom and if you cheat part of part of this exam, -999 will be the score of this exam.**

1. What is the run-length encoding for? (1)

Image Compression

1. What is a run in the run-length encoding? (1)

Run happens when same pixelValue appears consecutively. For instance: first row of the image might have all zero. That means, there is a single run of zero.

1. How do we represent a run in the run-length encoding? (1)

We represent a run using 4 integers. Row #, Col #, pixelValue, runLength

1. Among the four run-length encoding methods, which method has the best compression rate? And which method has the worst compression rate? (2)

Best= Wrap Around No Zero

Worst= No Wrap Include Zero

1. Among the four run-length encoding methods, which method is used in fax machine? (1)

No Wrap Include Zero

1. What kind of image would yield the worst compression rate to use run-length encoding no matter which method to use? Illustrate an example. (2)

Image where every consecutive pixelvalue is different and therefore results in unit run length for each of those pixelValue. Instead of original image where every pixel is stored as a single integer, we now introduce four integers to represent that single pixel as a run of unit length.

1. Write the results of the run-length encoding using: (A) Include zero's and wrapped around and (B) no zero's and wrapped around. Don't forget to write the image header! (5)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (A) | | |  |  |  |  | (B) | | |  |
| 7 | 10 | 0 | 9 |  |  |  | 7 | 10 | 0 | 9 |
| 0 | 0 | 0 | 6 |  |  |  | 0 | 6 | 3 | 16 |
| 0 | 6 | 3 | 16 |  |  |  | 2 | 2 | 4 | 8 |
| 2 | 2 | 4 | 8 |  |  |  | 0 | 0 | 7 | 6 |
| 0 | 0 | 7 | 6 |  |  |  | 3 | 6 | 5 | 7 |
| 3 | 6 | 5 | 7 |  |  |  | 4 | 3 | 1 | 4 |
| 4 | 3 | 1 | 4 |  |  |  | 5 | 0 | 6 | 8 |
| 4 | 7 | 0 | 3 |  |  |  |  |  |  |  |
| 5 | 0 | 6 | 8 |  |  |  |  |  |  |  |
| 5 | 8 | 0 | 12 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |

7 10 0 9 🡨image header

0 1 2 3 4 5 6 7 8 9

0 0 0 0 0 0 0 3 3 3 3

1 3 3 3 3 3 3 3 3 3 3

2 3 3 4 4 4 4 4 4 4 4

3 7 7 7 7 7 7 5 5 5 5

4 5 5 5 1 1 1 1 0 0 0

5 6 6 6 6 6 6 6 6 0 0

6 0 0 0 0 0 0 0 0 0 0

1. Given the image on the left, apply the **1st pass** of the **8-connected** component algorithm, modify the original image on the right to show the result. You must initialize the EqTable and update it during the process, and as will as update newLabel. (6)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 2 | 0 | 0 | 3 | 0 | 0 |  |  |  |  |  |  | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 2 | 0 | 3 | 0 | 0 |  |  |  |  |  |  | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 2 | 0 | 0 | 3 | 0 |  |  |  |  |  |  | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 4 | 1 | 0 | 1 | 1 | 0 | 0 | 3 | 0 |  |  |  |  |  |  | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 3 | 0 | 0 |  |  |  |  |  |  | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 5 | 0 | 6 | 0 | 0 | 3 | 0 |  |  |  |  |  |  | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 5 | 0 | 0 | 6 | 0 | 3 | 0 |  |  |  |  |  |  | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 7 | 5 | 0 | 0 | 0 | 6 | 3 | 0 | 0 |  |  |  |  |  |  | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **NewLabel:** | | | 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  | **EqTable** | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | . . |  |  |  |  |  |  |  |
|  |  |  |  |  | 0 | 1 | 2  1 | 3 | 4  1 | 5 | 6  3 | 7  5 |  |  |  |  |  | . . |  |  |  |  |  |  |  |

1. Apply the **2st pass** of the **8-connected** component algorithm on the left (the result of the 1st pass 8-CC), and modify the original given on the right to show the result. Update EqTable during the process as needed. (5)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 3 | 0 |  |  |  |  |  |  | 0 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 2 | 0 |
| 0 | 4 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 0 |  |  |  |  |  |  | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 2 | 0 | 0 |
| 0 | 0 | 4 | 1 | 0 | 1 | 0 | 0 | 2 | 0 |  |  |  |  |  |  | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 2 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 | 0 | 0 |  |  |  |  |  |  | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |  |  |  |  |  |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 |
| 0 | 5 | 5 | 0 | 0 | 6 | 0 | 0 | 2 | 0 |  |  |  |  |  |  | 0 | 5 | 5 | 0 | 0 | 2 | 0 | 0 | 2 | 0 |
| 0 | 0 | 0 | 5 | 0 | 0 | 6 | 0 | 2 | 0 |  |  |  |  |  |  | 0 | 0 | 0 | 5 | 0 | 0 | 2 | 0 | 2 | 0 |
| 0 | 7 | 0 | 0 | 0 | 0 | 6 | 2 | 0 | 0 |  |  |  |  |  |  | 0 | 7 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |  |  |  |  |  |  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **NewLabel :** | | | 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  | **EqTable** | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | . . |  |  |  |  |  |  |
|  |  |  |  |  |  | 0 | 1 | 2 | 2 | 1 | 2 | 2 | 7 | 8 | 9 | 10 | 11 | 12 | . . |  |  |  |  |  |  |

|  |  |  |
| --- | --- | --- |
| a | b | c |
| d | x | e |
| f | g | h |

1. Given the 3x3 diagram below, describe in cases, how to assign pixel x's label, during the1st pass and the 2nd pass of the **4-**connected component algorithm.

a) 1st pass of **4-**connected component: (3)

For the 1st pass in 4 connectedness, we look at a,b,c,d

1st case: b=d=0

X=newLabel++

2nd case: Some/all of the labels are same

X=same label

3rd case: Some/all labels are different

X=min(b,d)

b) 2nd pass of **4-**connected component: (3)

For 2nd pass , we look at x,e,g

1st case: e=g=0

Keep the X

2nd case: e=g=x

Keep the X

3rd case: Some/all of e, g, x are different

X=min(e,g,x)

1. What is morphology? (1)

It is the study of shapes.

1. Given a 1-D image and the structuring element (with origin **1** ) below, apply 1-D morphological “dilation”; write the result after arrow 🡪. (2)

0 0 1 1 0 1 0 1 1 0 1 0 0 1 0 1 0 0 1 **1** 1

Dilation result 🡪 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0

1. Given the 1-D image and the structuring element (with origin **1** ) below, apply 1-D morphological “erosion” and write the result after arrow 🡪. (2)

0 0 1 1 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 **1** 1

Erosion result 🡪 0 0 0 1 1 0 0 0 1 1 0 0 0 0 0 0 0 0

1. Given a 2-D image on the left and a structuring element (with origin **1** ) on the top, apply 2-D ‘dilation’ operation using the structuring element; modify the image given on the right to show the result of dilation. (3)

**1** 1

1 1

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

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

0 0 0 0 1 0 0 0 1 0 dilation 0 0 0 0 1 1 0 0 1 1

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

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

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

1. Given a 2-D image on the left and a structuring element (with origin **1** ) on the top, apply 2-D ‘erosion’ operation using the structuring element; modify the image given on the right to show the result of erosion. (3)

**1** 1

1 1

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

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

0 0 1 1 0 1 1 0 0 0 erosion 0 0 1 0 0 1 0 0 0 0

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

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

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

1. In general, when do we use the Opening operation? Explain. (1)

Opening is used when we want to extract object of certain characteristic as given by structuring element.

1. In general, when do we use the Closing operation? Explain. (1)

Closing is used when we want to fill in holes in an object.

1. Given a binary image, where objects (1's) in the image are spaghetti, meat balls, small pieces of meat. The task is to extract only meat balls, no spaghetti, and no small pieces of meat. Design a sequence of morphological operations with the shape, the size, and the origin of the structuring element for the task. You may draw an object-process diagram or write in English showing the steps by steps of the sequence of morphological operations and the expected result after each opertion. (5)

Basically, we need to perform opening to extract only the meatballs. However, the detail algorithm for doing that is as given below:

Step 1: Apply erosion with structural element S1 whose shape is circular, size is smaller than smallest meatballs but is bigger than small pieces of meat or spaghetti. The origin for the structural element will be at the center.

Step 2: Use the same structural element S1 to apply dilation operation .

1. What is the main purpose of framing the input image prior to image processing operations? (2)

Framing the input image is done because we might need neighborhood pixel values for all the pixels in the image. Edge pixels clearly do not have all the neighborhood pixels, therefore, we introduce those neighbors by framing. We can do either mirror framing or zero framing depending on what kind of neighborhood operation we want to perform.

1. Given a 3x3 neighbors of pixel, p (i, j) below, write the computation formula for the three image noise filters you were taught in class: (4)

|  |  |  |
| --- | --- | --- |
| a | b | c |
| d | (i,j) | e |
| f | g | h |
|  |  |  |
|  | Mask | |
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |

(a) 3x3 averaging: p’(i,j) = (a+b+c+d+P(i,j)+e+f+g+h)/9

(b) 3x3 median-filter: p’(i,j) = 5th item(Sort(a,b,c, d, P(i,j),e,f,g,h))

(c) Gaussian filter (with a given 3x3 Mask):

p’(i,j) =(a+2b+c+2d+4P(i,j)+2e+f+2g+h)/16

1. Why the 3x3 median-filter and the 3x3 averaging-filter may wipe out corners of objects in the image? Explain your answer for credits. (2)

Median and average filters may wipe out the corners of the objects because majority of the neighboring pixels in the corner of an object are background pixels. Hence our mean and median will give us value closer to background pixel rather than corner’s original value, thereby wiping out the corners.

1. Write the algorithm steps for the corner-preserve-averaging filter. (5)

Regular averaging filters do not preserve corners and edges at all. Therefore, we need to modify the averaging algorithm to be able to preserve the corners. We call this modified algorithm corner preserving averaging algorithm. We achieve it by following the algorithm given below:

1. Mirror frame the original image.
2. Scan the image from left to right, top to bottom.
3. For each pixel value, we have a neighborhood of size N\*N (say). We divide the the neighborhood into 8 groups.

8 groups will of of different shapes as each of them will preserve corners and edges in different sides.

1. We compute the average for each of the groups. We store them in a1,…,a8.
2. We compute the differences between the currentPixel and all 8 averages. We store it in d1, d2,…d8.
3. We find the minimum from d1, d2,…d8 and store the corresponding value of ai.
4. We update our average for the current Pixel to be the average (ai) for which the difference was the minimum.
5. Repeat all steps from 3-7 for all pixels in the image.
6. Why do we need the image header in every image? (2)

We need image header in order to be able to process the image. One example of why image header might be necessary would when we want to allocate an array to store all image’s pixelValue. Without image header, we would have to count the number of rows and columns first and then allocate an array before we can finally put the image’s pixelValue into the memory for further processing.

1. What is the histogram of an image? (1)

Histogram of an image is the counting of different pixel values in an image. If an image has 100 pixelValue of 1, hist[1] would be 100.

1. Can we reconstruct the image from its histogram? Explain your answer. (1)

No, we cannot reconstruct the image from its histogram because all histograms gives us is counter of how many times a certain pixelValue has appeared in the image which is clearly not enough to form the image. We need to know where are each of the pixel’s located to form the original image.

1. Can we guess the content of an image from its histogram? Explain your answer. (1)

No we cannot guess the content of an image but we might have general idea of what colors comprises the image and therefore, we might even be able to guess how many kinds of objects are present in the image and maybe even size of background.

1. Will the histogram of a grey-scale image changed drastically after a noise cleaning operation from before the operation? Explain your answer for credits. (2)

No, it wont change drastically but the curve of the histogram will be more uniform.

1. Write in English or in algorithm steps for the general idea of the **deepest-concavity** automatic threshold selection method. (4)
2. Create a histogram from an image. It needs to be bimodal for deepest concavity to work.
3. Histogram must be made uniform by using median filter. We can use mirror framing to calculate median for the edge pixelValue in histogram.
4. Then we can proceed to find two peaks say x1 and x2. We form an equation of line X1X2.
5. We move x (currentX) from x1 to x2. We record the x value(threshold) where the difference between hist[X] and y value given by line X1X2 is maximum.
6. We return the threshold value.
7. Write in English or in algorithm steps for the general idea of the **bi-means** automatic threshold value selection. (4)
8. Given an image, produce a histogram.
9. Smooth the histogram using Median Filter.
10. Suppose the histogram extends from a to b. Then we add small constant which is about 10% of the total (say c) to the minimum to get the div.
11. We now can fit a gaussian curve from min to div and another gaussian from div to maximum.
12. We now find the difference in area between both the gaussian curves and the histogram and store it in a variable say Ai.
13. Now we add 1 to the div.
14. We repeat step 4 to 6 until div is greater than maximum-c.
15. Our threshold value is now given by min(A1, A2,…)

CV Spring 2021 Exam I (Part 2 - coding) Name:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\* **When ask to write a method, you MUST write the header/prototype of the method, NOT just method body.**

1. (In C++) Implement the following steps (write in the blank space after steps):

Step 0: Write C++ library you include in your C++ projects. (2)

#include <iostream>

#include <fstream>

#include <string>

#include <cstdarg>

#include <cmath>

Step 1: inFile, outFile 🡨 Open the input and output files from argv (1st and 2nd), (2)

thrVal 🡨 get from 3rd argv. (1)

string inputName=argv[1];

string outputName=argv[2];

int thrVal=argv[3];

Step 2: data 🡨 inFile

If data < thrVal

outFile 🡨 0

else

outFile 🡨 data

Step 3: repeat step 2 until at the end of inFile. (4)

For(int i=0;i<numRows;i++){

For(int j=0;j<numCols;j++){

inFile>>data;

if(data<thrVal){

outFile<<0;

}

Else{

outFile<<data;

}

}

}

1. **(In C++)** a) Dynamically allocates the 2D mirrowFramedAry (use MFAry for short) size of numRows + frameSize by numCols + frameSize. (b) Reads and loads data from inFile into inside frame of MFAry. (5)

MFAry=new int \*[numRows+frameSize];

For(int i=0;i<numRows+frameSize;i++){

MFAry=new int[numCols+frameSize];

}

For(int i=frameSize;i<numRows+frameSize;i++){  
for(int j=frameSize;j<numCols+frameSize;j++){

inFile>>MFAry[i][j];

}

}

1. **(in C++)** Write the method, mirrorFraming (MFAry, frameSize) for 3x3 median filter where frameSize is 3. (5)

Void mirrorFraming(int \*\*mfary, int frameSize){  
int totalRows=numRows+2\*frameSize;

int totalCols=numCols+2\*frameSize;

for(int i=frameSize-1;i>=0;i++){

for(int j=frameSize-1){  
mfary[][]

}

}

}

1. **(in C++)** Write the method, convolution (MFAry, i, j, mask), the method computes the convolution using a given 5x5 mask onto the pixel MFAry[i, j]'s 5x5 neighborhood and returns the convolution result. (6)
2. **(In Java)** Implement the following steps. (4)

step 1: i 🡨 rowFrameSize

step 2: j 🡨 colFrameSize

step 3: if inAry [i,j] > 0

dilation (i, j, inAry, outAry) // call dilation method without define it. You will write the dilation method next.

step 4: j++

step 5: repeat step 3 to step 4 while j < (numImgCols + colFrameSize)

step 6: i++

step 7: repeat step 2 to step 6 while i < (numImgRows + rowFrameSize)

1. **(In Java)** Write the method, dilation (i, j, inAry, outAry, structAry) that performs dilation on inAry [i, j] with structAry and outputs to outAry[i, j]. You may use rowOrigin, colOrigin, numStructRows, and numStructCols without define them. (6)

1. **This question is for graduatae students who registered in cs 780 . Undergrade will get extra points if answered.**

**(In Java)** Write the method, erosin (i, j, inAry, outAry, structAry) that performs eroson on inAry [i, j] with structAry and outputs to outAry[i, j]. You may use rowOrigin, colOrigin, numStructRows, and numStructCols without define them. (-4 to 0). If you get it wrong, -4; if correct, 0, partial correct from -1 to -3, depends on the correctness.