Project 11: Document text-line detection and reading direction using Projection Profiles.

Abstract: One major task in the document image analysis is to decompose a given document into a hierarchical tree structures where the root is the whole document (could be one page or multiple pages); the next level under a page are one or more column blocks, we called blocks as “zones” hereafter; each column zone could consists of text zones and other none-text zones (such as figures, graphic, table, math equations, ...); the next level under text-zones are paragraphs; below paragraphs are text-lines; below text-lines are text-words; below text-words are characters. An Optical Character Recognition (OCR) system begins it document recognition from bottom up of the document hierarchical tree; it first does character recognition, up to form words, up to form text-lines, and so for. A highly effective technique for document image decomposition is using the projection profiles of a given document to construct the document hierarchy top-down. The lecture note on this topic is posted at blackboard for your reference and study.

The Projection Profiles (also called a Signature Analysis of a Binary Image) is a Projection (summing up) object (none-zero) pixels within a given “text-zone” within the image, taking on a given direction i.e. vertical or horizontal, therefore are called Horizontal Projection Profile (HPP) and Vertical Projection Profile (VPP).

(Note: In your mock exam 1 and exam 2, the horizontal Projection Profile (HPP) is called sumRows and the Vertical Projection Profile (VPP) is call sumCols).

The HPP and VPP can also be used to determine the reading direction of a given document, by analyzing the patterns of HPP and VPP.

In this project, you are only to determine the bounding boxes of text-lines within the given text-zone and to determine the reading direction of the document.

What you need to do for this project:

0) Given a binary document image (with image header) consists only a text-zone.

1) Dynamically allocate a 2D imgAry for the input image size of numRows + 2 by numCols + 2. Zero-framed the imgAry.

2) Dynamically allocate a 1D array for the Horizontal Projection Profile (HPP) size of numRows +2, initialized to 0.

3) Dynamically allocate a 1D array for the Vertical Projection Profile (VPP) size of numCols +2, initialized to 0.

4) Compute the HPP and the VPP from the input image.

5) Output the HPP and the VPP to outFile2 with proper captions.

6) Threshold both HPP and VPP, try ThrValue >= 3, to obtain binary, called HPPBinary and VPPBinary.

7) Output the HPPBinary and VPPBinary to outFile2 with proper captions.

//You have already done the above in your mock exam 1 and exam 2.

8) Apply 1D morphological closing operation on both HPPBinary and VPPBinary, begin at 1, with 1D structuring element: 1**1**1 origin at the center.

Call the results HPPMorph and VPPMorph

9) Using HPPMorph and VPPMorph to determine the reading direction of the document image. There are a few methods to do so; a simple run-counting method is given below: (a run is a sequence of consecutive pixels with the same value.)

step 1: HPPruns 🡨 compute the number of runs in HPPMorph

VPPruns 🡨 compute the number of runs in VPPMorph

step 2: if HPPruns >= factor \* VPPruns // try factor == 3

return horizontal

else if VPPruns >= factor \* HPPruns

return vertical

else write an error message to the console:

“Can not determine the reading direction”

and exit the program

10) Output the reading direction to outFile2 with proper caption.

11) Base on the reading direction, you are to compute the text-line bounding boxes using either HPPMorph (if reading is horizontal) or VPPMorph (if reading is vertical). Recalled – A bounding box is represented by two points of four integers (minRow, minCol) and (maxRow, maxCol) as you have done in your previous projects.

12) Ideally, a document hierarchical linked list structure should be used to store the extracted bounding boxes in the matter of document decomposition hierarchy; however, to make the project easier, instead of constructing a tree hierarchy, you only need to use a queue to store the computed bounding boxes.

13) Draw the text-line bounding box onto the image array.

14) Pretty print the image array with bounding box overlaid to OutFile1.

15) Print to OutFile2 the Queue from front to back using the format as follows.

box type // box type here is 3 for text-line

minRow minCol maxRow maxCol

box type // box type here is 3 for text-line

minRow minCol maxRow maxCol

box type // box type here is 3 for text-line

minRow minCol maxRow maxCol

:

:

16) You will be given two data files, run your program on each file. Print both results in your hard copies.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Language: C++

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Points: 12 pts

Due Date: Soft copy and pdf hard copies:

5/5/2020 Tuesday before midnight

Early submission: +1 4/30/2020 Thursday before midnight

1 day late: -1 pt 5/6/2020 Wednesday before midnight

2 days late: -2 pts 5/7/2020 Thursday before midnight

-12 pts: after 5/7/2020 Thursday \*after midnight

\*\*\* Name your pdf file using the same naming convention as given prior

\*\*\* All on-line submission MUST include Soft copy and pdf hard copy

\*\*\* in the same email with correct file names; otherwise, it would not count as submission.

Include in your hard copy:

- Cover page (include the algorithm steps you wrote in main())

- Source Code

- outFile1 from data1

- outFile2 from data1

- outFile1 from data2

- outFile2 from data2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

I. inFile1 (argv[1]): A binary image.

argv[2]: a threshold value, try 3 for this project.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

II. OutFile1 (argv[3]): The text-line bounding box overlaid on the input image

OutFile2 (argv[4]): All other outputs as given in the above.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

III. Data structure:

\*\* You may add/modify the following data structure as you see fit.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

- a imagePP class:

- numRows (int)

- numCols (int)

- minVal (int)

- maxVal (int)

- a box class

- minR (int)

- minC (int)

- maxR (int)

- maxC (int)

- a boxNode class

- boxType (int) // 1 -doc box; 2 -paragraph; 3 textLine; etc.

- BBox (box)

- next(boxNode\*)// points to boxNode in the same level.

* a boxQ class

- Qfront (boxNode\*)// initially point to a dummy node

- QBack (boxNode\*) // points to the last node in the list,

//initially points to dummy node

-insert (Q, newBoxNode) // insert the box at the end of Q

// QBack->next 🡨 newBoxNode.

// QBack 🡨 newBoxNode

- imgAry (int \*\*) a 2D array, need to dynamically allocate at run time

of size numRows +2 by numCols +2.

- thrVal (int) // the threshold value provided in argv[2]

- HPP(int\*) // a 1-D array to store the horizontal projection profile,

// dynamically allocate at run time, size of numRows + 2

- VPP(int\*) // a 1-D array to store the vertical projection profile,

// dynamically allocate at run time, size of numCols + 2

- HPPbin(int\*) // a 1-D array of the binarized HPP

// needs to dynamically allocate at run time, size as HPP

- VPPbin(int\*) // a 1-D array to store the binarized VPP,

// needs to dynamically allocate at run time, size as VPP

- HPPMorph (int\*) //

- VPPMorph (int\*) //

- HPPruns (int)

- VPPruns (int)

- methods:

- zeroFramed (imgAry)

- loadImage (imgAry)

- computeHPP (imgAry, imgBox, HPP) // compute the horizontal projection profile

// of imgAry within imgBox.

- computeVPP (imgAry, imgBox, VPP) // compute the vertical projection profile

// of imgAry within imgBox.

- threshold(PP, thrVal, binPP) // threshold on the given projection profile, PP // if PP(i) < thrVal) binPP(i) set to 0, else set binPP(i) to 1

- printPP (PP) // print the given projection profile can be either HPP or VPP

- morphClosing (PPMorph, structElem) // compute 1D morphological closing

// using 1**1**1 as the structuring element on the given PPMorph

// can be either HPP or VPP

- findLineBoxes(PP)

// given PP, find the bounding box of text-line, one-by-one in PP

// and insert each text-line box at the end of the Queue.

- printBoxInfo(Box) // print the bounding box info of given Box

- drawBoxes (...) // Draw the text-line bounding boxes onto

// in imgAry. Re-use code in your cc project.

- printBoxQueue (...)

- you may add more methods as needed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

VI. main()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\* Design your own algorithm steps here, based on the tasks given in the above, and re-use codes in your mock exam 1 and mock exam 2.