Project 8: Document text-line detection using Projection Profiles.

What you need to do for this project:

0) Given a binary document image.

1) Compute the bounding box that encloses all objects (1's) in the entire image; called it imgBox.

2) Dynamically allocate the Horizontal Projection Profile (HPP) and the Vertical Projection Profile (VPP) according to the height and the width of the imgBox, as height + 2 for HPP and width + 2 for VPP. Initialize both arrays to 0.

3) Compute the Horizontal Projection Profile (HPP) and the Vertical Projection Profile (VPP) within imgBox of the image, begins at 1, not 0.

4) Output the HPP and VPP to outFile with proper captions.

5) Threshold both HPP and VPP, using thrValue = 3, to obtain binary, called HPPBinary and VPPBinary.

6) Output the HPPBinary and VPPBinary to outFile with proper captions.

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

Call the results HPPMorph and VPPMorph

8) 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

9) Output the reading direction to outFile with proper caption.

10) Base on the reading direction, you are to compute the text-line bounding boxes using either HPPMorph (if reading is horizhontal) or VPPMorph (if reading is vertical).

11) 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, you only need use a queue to store the computed bounding boxes.

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

box format:

box type

minRow minCol maxRow maxCol

\*\* You will be given two data files, run your program on each file. print both result in your hard copies.

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

Language: **C++** (12 pts for the project.)

Due date (soft copy): 5/8/2019 Wednesday before Midnight

Early submission +6 deadline: 5/5/2019 Sunday before Midnight

Early submission +4 deadline: 5/6/2019 Monday before Midnight

Early submission +2 deadline: 5/7/2019 Tuesday before Midnight

On time submission +1 deadline: 5/8/2019 Wednesday before Midnight

After 5/10/2019, -5 pts for all students who did not submit soft copy

Due Date (hard copy): 5/9/2019 Thursday in class,

-1 pt for late hard copy submission after Thursday 5/9/2019

All projects without hard copy after 5/10/2019 will receive 0 pts even you have submitted soft copy on time and even if it works.

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

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

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

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

II. OutFile (argv[3]): Write all the outputs.

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

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\*)// initally point to a dummy node

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

//initally 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 by numCols.

* imgBox (box)

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

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

// needs to dynamically allocate at run time, size of ??

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

// needs to dynamically allocate at run time, size of ??

- 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:

- loadImage (imgAry)

- computeBBox (...) // computes the bounding box

- 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,j) < thrVal) binPP(i,j) set to 0, else set binPP(i,j) to 1

- printPP (PP) // print the given projection profile

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

// using 111 as the structing element on the given PPMorph

- 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.

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

- printBoxQueue (List) // Call

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

VI. main()

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

\*\*\* You may design your own algorithm based on the steps given in the above.