EEP702, SOFTWARE LABORATORY

Connected Component Labelling

Rishi Dua, 2010EE50557

April 16, 2014

1 PYTHON ASSIGNMENT

1.1 PROBLEM STATEMENT

1. There is a mass of land containing some finites number of water bodies. Let the piece of land be represented by a 2-D matrix, whose each cell represents a unit area. Value '0' of a cell denotes that it comes under land area and '1' denotes that it comes under water body. If the cells connected by a 4-point connectivity forms a single water body, find the total number of water bodies.

2. If the cells connected by a 8-point connectivity forms a single water body, find the total number of water bodies.

3. Label the water bodies so that if user specifies a cell (m,n), the program must tell whether it comes under a water body.

1.2 ABSTRACT

Connected-component labeling (alternatively connected-component analysis, blob extraction, region labeling, blob discovery, or region extraction) is an algorithmic application of graph theory, where subsets of connected components are uniquely labeled based on a given heuristic. Connected-component labeling is not to be confused with segmentation.

Connected-component labeling is used in computer vision to detect connected regions in binary digital images, although color images and data with higher dimensionality can also be processed. When integrated into an image recognition system or human-computer interaction interface, connected component labeling can operate on a variety of information. Blob

extraction is generally performed on the resulting binary image from a thresholding step. Blobs may be counted, filtered, and tracked.

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C. The language provides constructs intended to enable clear programs on both a small and large scale.

Python supports multiple programming paradigms, including object-oriented, imperative and functional programming or procedural styles. It features a dynamic type system and automatic memory management and has a large and comprehensive standard library.

1.3 Specification And Assumptions

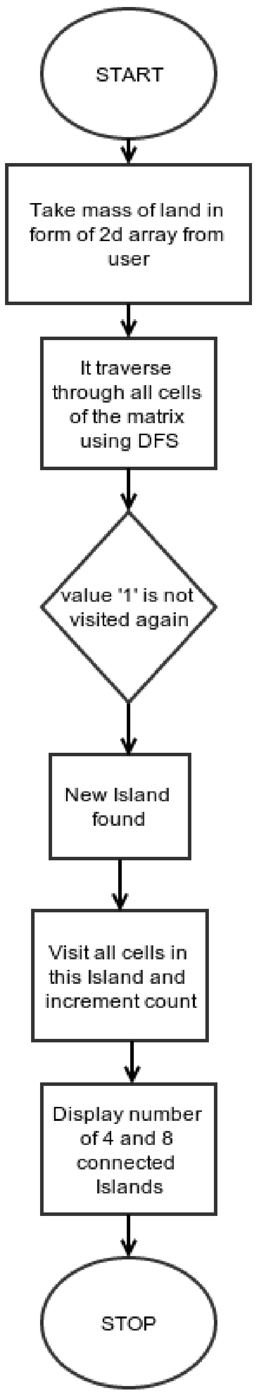
Tool Specifications:
Python 2.7
Platform: Ubuntu 12.04
Bash Version: GNU bash, version 4.2.25(1)-release (x86_64-pc-linux-gnu)

Problem specifications: Assume the following utilities are present on the system
Python
Also works if input is negative

Assumptions
The user gives inputs in the format specified

1.4 FLOW CHART

The flowcharts is as follows


START
Take mass of land in form of 2d array from user
It traverse through all cells of the matrix using DFS
value '1' is not visited again
New Island found
Visit all cells in this Island and increment count
Display number of 4 and 8 connected Islands
STOP

1.5 LOGIC IMPLEMENTATION

Two-pass Algorithm Relatively simple to implement and understand, the two-pass algorithm iterates through 2-dimensional, binary data. The algorithm makes two passes over the image: the first pass to assign temporary labels and record equivalences and the second pass to replace each temporary label by the smallest label of its equivalence class.

The input data can be modified in situ (which carries the risk of data corruption), or labeling information can be maintained in an additional data structure.

Connectivity checks are carried out by checking neighbor pixels' labels (neighbor elements whose labels are not assigned yet are ignored), or say, the North-East, the North, the North-West and the West of the current pixel (assuming 8-connectivity). 4-connectivity uses only North and West neighbors of the current pixel. The following conditions are checked to determine the value of the label to be assigned to the current pixel (4-connectivity is assumed)

Conditions to check:

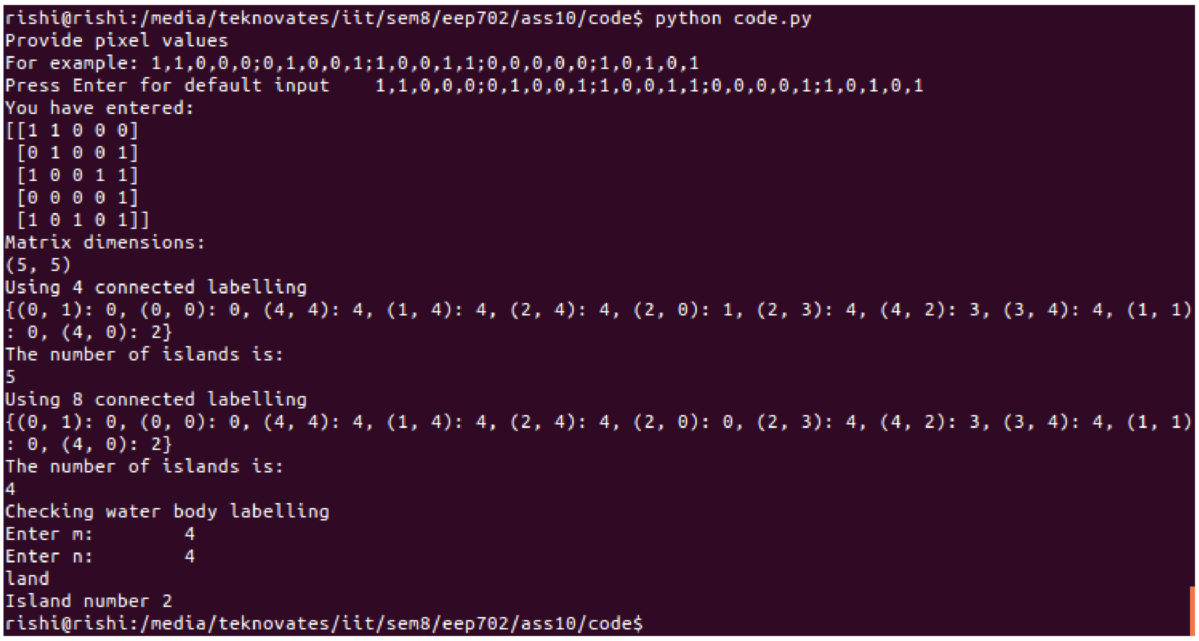
- Does the pixel to the left (West) have the same value as the current pixel?
  Yes: We are in the same region. Assign the same label to the current pixel
  No: Check next condition
- Do both pixels to the North and West of the current pixel have the same value as the current pixel but not the same label?
  Yes : We know that the North and West pixels belong to the same region and must be merged. Assign the current pixel the minimum of the North and West labels, and record their equivalence relationship
  No : Check next condition
- Does the pixel to the left (West) have a different value and the one to the North the same value as the current pixel?
  Yes : Assign the label of the North pixel to the current pixel
  No : Check next condition
- Do the pixel's North and West neighbors have different pixel values than current pixel?
  Yes : Create a new label id and assign it to the current pixel

The algorithm continues this way, and creates new region labels whenever necessary. The key to a fast algorithm, however, is how this merging is done. This algorithm uses the union-find data structure which provides excellent performance for keeping track of equivalence relationships. Union-find essentially stores labels which correspond to the same blob in a disjoint-set data structure, making it easy to remember the equivalence of two labels by the use of an interface method E.g.: findSet(l). findSet(l) returns

1.6 Execution Directive

pthon code.py

1.7 Output Of The Program



```
rishi@rishi:/media/teknovates/iit/sem8/eep702/ass10/code$ python code.py
Provide pixel values
For example: 1,1,0,0,0;0,1,0,0,1;1,0,0,1,1;0,0,0,0,0;1,0,1,0,1
Press Enter for default input    1,1,0,0,0;0,1,0,0,1;1,0,0,1,1;0,0,0,0,1;1,0,1,0,1
You have entered:
[[1 1 0 0 0]
 [0 1 0 0 1]
 [1 0 0 1 1]
 [0 0 0 0 1]
 [1 0 1 0 1]]
Matrix dimensions:
(5, 5)
Using 4 connected labelling
{(0, 1): 0, (0, 0): 0, (4, 4): 4, (1, 4): 4, (2, 4): 4, (2, 0): 1, (2, 3): 4, (4, 2): 3, (3, 4): 4, (1, 1)
: 0, (4, 0): 2}
The number of islands is:
5
Using 8 connected labelling
{(0, 1): 0, (0, 0): 0, (4, 4): 4, (1, 4): 4, (2, 4): 4, (2, 0): 0, (2, 3): 4, (4, 2): 3, (3, 4): 4, (1, 1)
: 0, (4, 0): 2}
The number of islands is:
4
Checking water body labelling
Enter m:        4
Enter n:        4
land
Island number 2
rishi@rishi:/media/teknovates/iit/sem8/eep702/ass10/code$
```

1.8 Result

Successfully finds 4 connected and 8 connected islands and shows labels for a given block
Problems encountered:

1. Execute rights error

Successfully computes the total number of water bodies where the cells form 4-point connectivity.

Also, computes the total number of water bodies where the cells form 8-point connectivity and gives label for a given input point.

1.9 Conclusion

Learnt techniques like using recursive functions to make the job simpler to implement DFS algorithm, using function parameter calling, using multi-dimensional arrays; using exception handling; using looping and conditional statements to name a few.