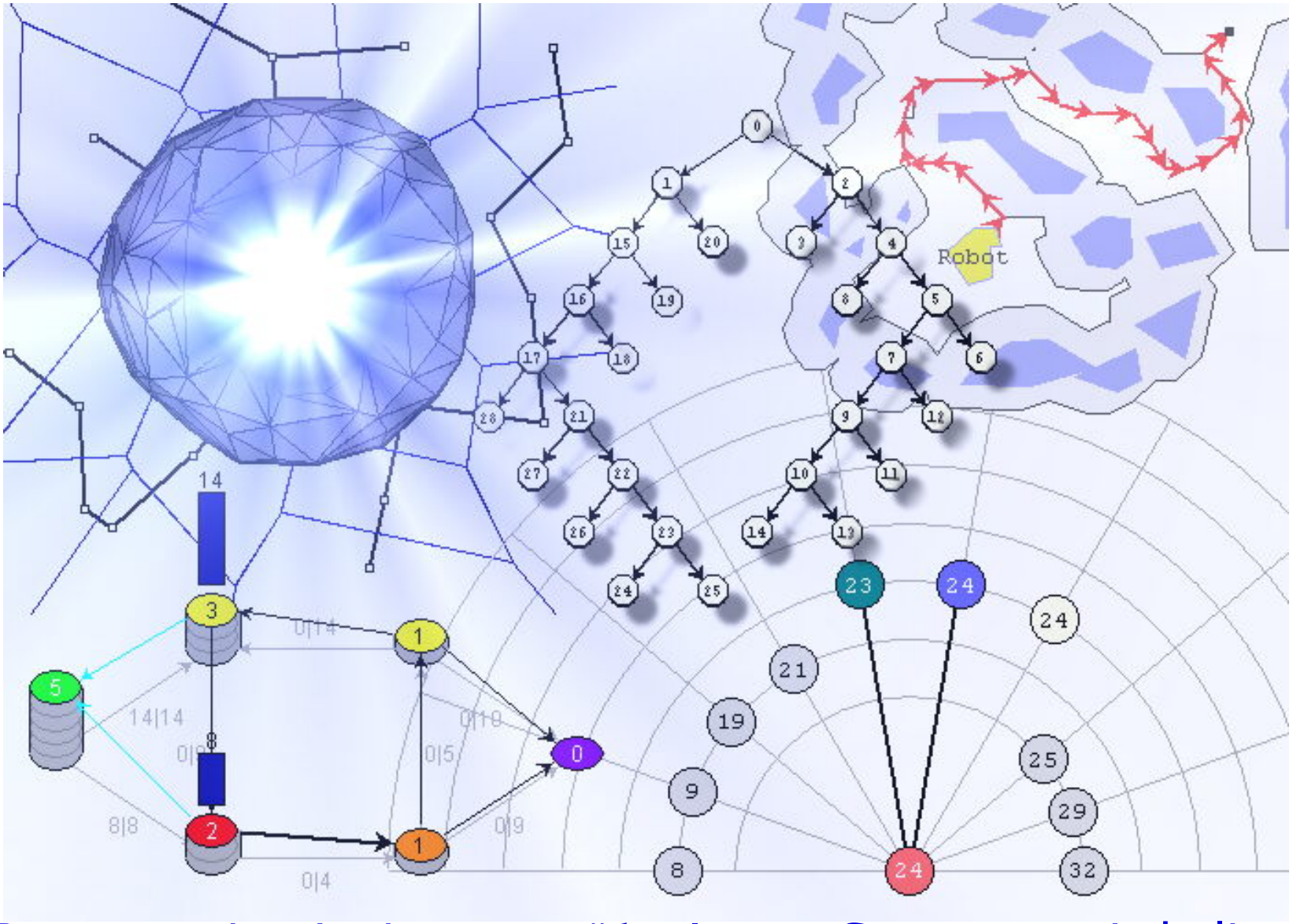
Data Structures and Algorithms



Programming Assignment #1 – Image Component Labeling

An application of DFS and BFS with Stacks and Queues

Due Date: Sunday, February 24, 2019

Image Component Labeling

- A digitized image is an m x m matrix of pixels.
  - *"pixel" is a word invented from "picture element"*
- In a binary image, each pixel is either 0 or 1.
  - A 0 pixel represents image background, while a 1 represents a pixel on an image component.
  - We will refer to pixels whose value is 1 as *component pixels*.
- Two pixels are adjacent if one is to the left, above, right, or below the other. Two component pixels that are adjacent are pixels of the same image component.
- The objective of component labeling is to label the component pixels so that

  two pixels get the same label

  if and only if

  they are pixels of the same image component.

Solution Strategy

- The components are determined by scanning the pixels by rows, from top to bottom, and within each row by columns, from left to right. Accordingly, the scanning order is similar to reading a page of English text.
- When an unlabeled component pixel is encountered, it is given a component identifier/label (new color). Labels are assigned starting with 2, because 1 denotes foreground and 0 denotes background.
- This pixel forms the seed of a new component.
  - We determine the remaining pixels in the component by identifying and labeling all component pixels that are adjacent to the seed.
  - Labeling adjacent component pixels allows to discover more adjacent component pixels (neighbor of neighbor is a neighbor).
  - This exploration can be directed in "Depth First Search" using a stack as in the Rat-in-the-Maze problem or in "Breadth First Search" using a queue as in Lee's Wire Routing problem.
  - This process continues until no new unlabeled adjacent component pixels are found.

Image Component Labeling Example

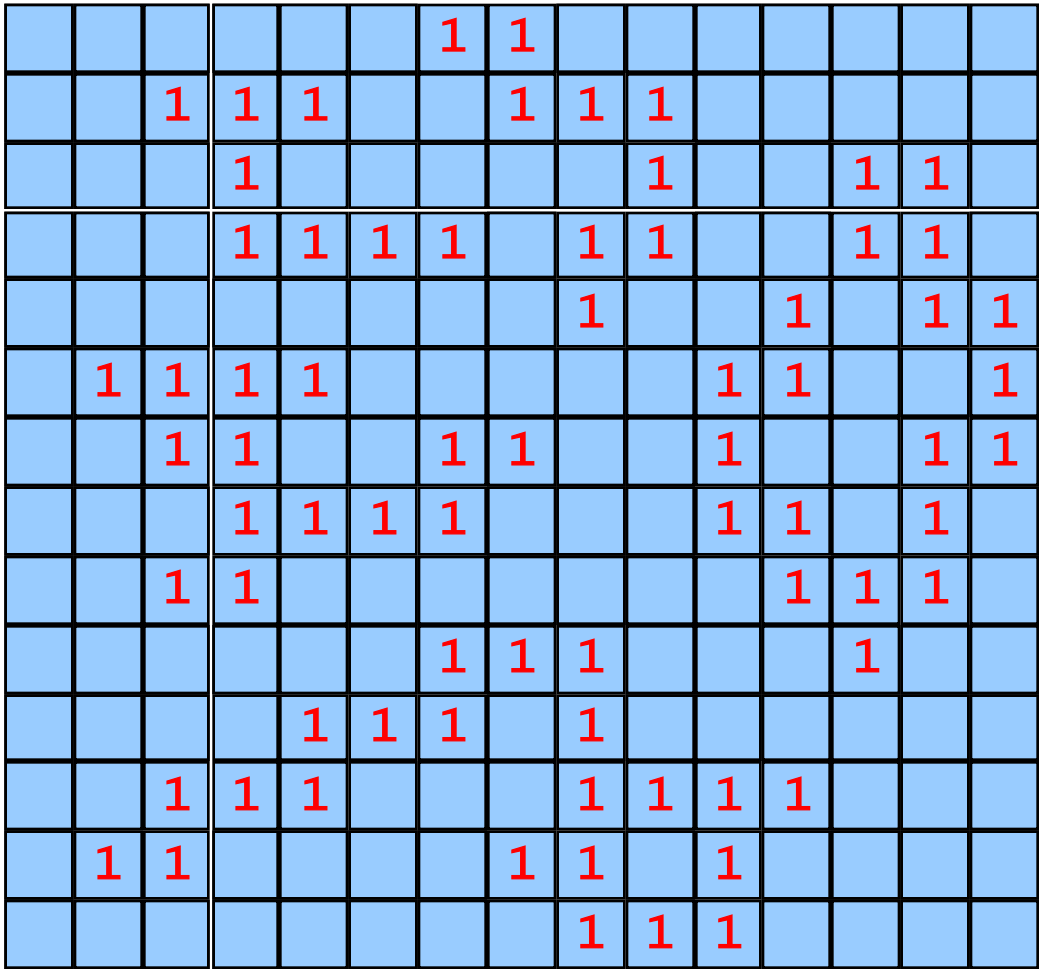
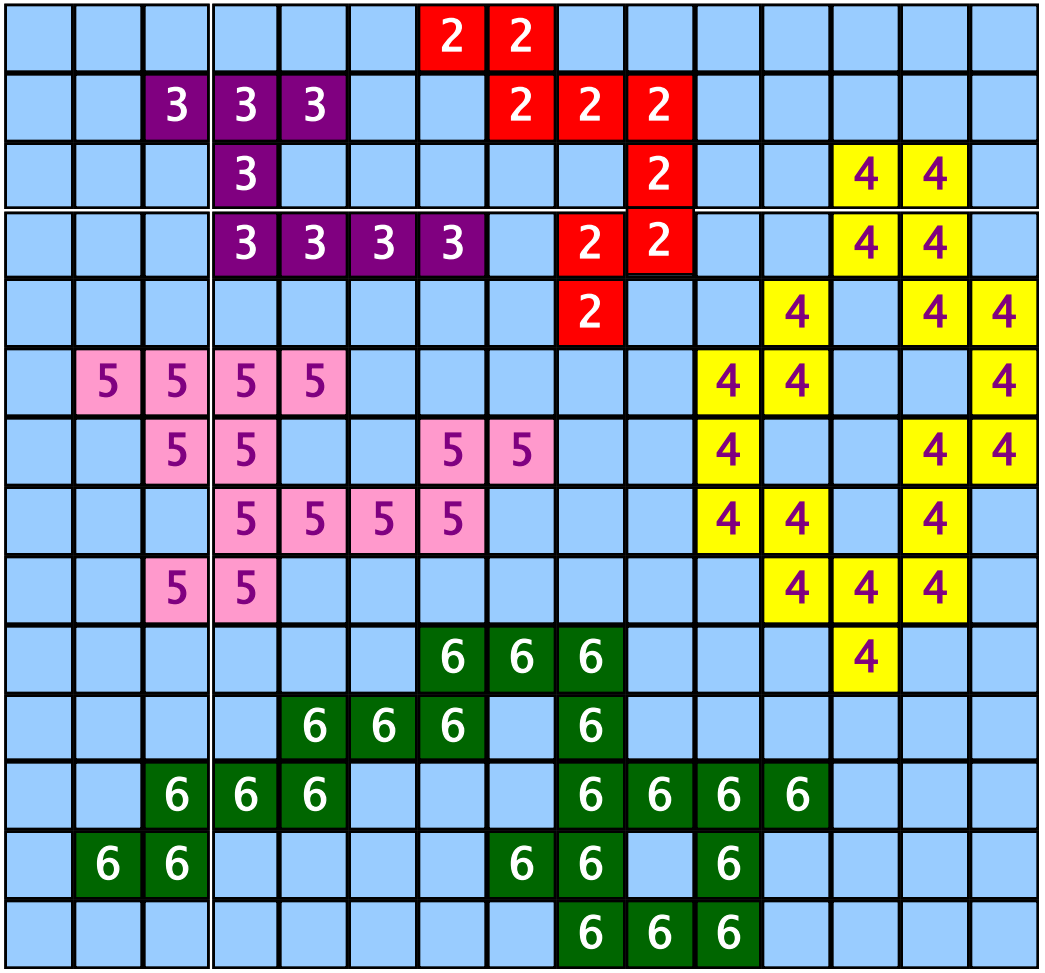


Image Component Labeling Example



Implementation

- The program to label component pixels uses much of the development used for the "Rat in the Maze" and "Lee's Wire-Routing" problems.
  - To move around the image with ease, we surround the image with a wall of blank (here 0 pixels), *unlike Rat in the Maze and Lee's Wire (Why?).*
  - We use the "offset" array to determine the pixels adjacent to a given pixel.
- *See starter code on Assignment 1 Canvas webpage...*
- Your task is to complete the labelComponents( ) method and adapt the DFS and BFS algorithms from "Rat in Maze" and "Lee's Wire Router" in order to implement Depth First Search and Breadth First Search.

Important Design Requirement

- Your design must be based on Modularity and "Separation of Concerns".
- The Stack and Queue Data Structure implementations must be based on "Information Hiding" and "Encapsulation".
- The Application Code (e.g. Rat In Maze, Wire Router, Image Component Labeling, ...) know about the Data Structures only through their Interfaces (APIs).
- Remember that interfaces represent behavior, while classes represent implementation.

General Strategy

1) Prompt the user for two values:
   - The DIMENSION of the image in pixels (square grid of pixels)
     Must be an integer between 5 and 20 inclusive (default is 15)
   - The DENSITY of foreground pixels
     Must be a floating number between 0.0 and 1.0 (default is 0.33)
     For example, DENSITY = 0.25, means that 25% of the pixels should be foreground and the remaining 75% should be background.

2) Your program maintains a 2-D grid (array) of objects, where each object keeps track of two values, the image component label (to be assigned by DFS or BFS), and the order in which the pixel was discovered, which depends on the search strategy (DFS or BFS). Accordingly, each pixel is represented by an object that encapsulates a label (component label: 2, 3, 4, ...) and (order of discovery: 1, 2, 3, ...).

General Strategy – continue

- For the purpose of illustration here, assume only for now, that the user chooses a DIMENSION of 15x15 pixels.
- Create a 2-dimensional array of size 17 by 17 (15 + 2 for the surrounding walls).
- Generate a grid of 15x15 cells, in the inner part of the array (i.e. excluding the surrounding walls), where each cell is either 0 (zero) for background, or 1 (one) for foreground. *See slide 10 below.*
- Important Observation: Notice that for the "Image component Labeling", we ignore the background 0's and trace through the foreground 1's, which is the opposite of "Rat in Maze" and "Lee's Wire Router", where we avoid the 1's which are walls or transistors, and trace through the 0's, which are open paths.

General Strategy – continue

3) Create imageDFS by randomly generating the pixel values.

The generateImage operation will populate the pixel [ ] [ ] square array that represents the image with 1's and 0's for foreground and background, respectively. Don't forget the *artificial wall* around the image. Initially, the 0/1 randomly generated number may be stored in each pixel[row][col].label

(i.e. the label field of each pixel[row][col] object)

Generating a Random Images

- Let R be the DENSITY, where $0 \leq R \lneq 1$
- The following pseudocode is used to populate the pixel [ ] [ ] square array (image):

```
for (int row = 1; row <= DIMENSION; row++)
   for (int col = 1; col <= DIMENSION; col++)
   {
      // generate a Random number R between 0 and 1;
      if (R < DENSITY)
            pixel[row][col].label = 1;    // foreground
      else
            pixel[row][col].label = 0;    // background
   }
```

  - If for example, DENSITY = R = 0.25, then the image will be 75% background and 25% foreground.

General Strategy – continue

4) Create imageBFS as an identical copy of imageDFS.
   - ➙ After populating the grid with pixel values and surrounding walls, clone the grid so that the first copy is used for DFS and the identical second copy for BFS.

5) Apply the Depth First Search algorithm to imageDFS

   Apply the Breadth First Search Algorithm to imageBFS

   Proceed as in the adapted "Rate in Maze" for DFS and then as in the adapted "Lee's Wire Router" for BFS.

   The image components are labeled in the order discovered by the respective search algorithm.

   All pixels of the same component will have the same label value and an order number which starts at 1 and keeps counting per image component pixel.

6) Printout the two resulting images (see slides 13 and 14 below).

Example – Initial (After random pixel generation)

- Suppose the user entered:
  - DIMENSION = 7
  - DENSITY = 0.33 (default)
- Then your program may generate the following image with 16 foreground pixels out of the 49 image pixels. (This doesn't account for the surrounding wall.)

  PS: The Red/Black colors are just for ease of reading here.

  The x values are unset ORDER ivars, to be set by the DFS or BFS algorithm.

0,x	0,x	0,x	0,x	0,x	0,x	0,x
0,x	1,x	1,x	1,x	0,x	0,x	0,x
0,x	1,x	1,x	0,x	1,x	1,x	1,x
0,x	1,x	1,x	1,x	0,x	1,x	1,x
0,x	0,x	0,x	0,x	0,x	1,x	1,x
0,x	0,x	0,x	0,x	0,x	1,x	0,x
0,x	0,x	0,x	0,x	0,x	0,x	0,x

- Submit source code and output corresponding to DFS and BFS traversals (see next 2 slides).

Example – Corresponding DFS

- The Depth First Search algorithm will printout the following corresponding grid of pixels:

0,x	0,x	0,x	0,x	0,x	0,x	0,x
0,x	1,1	1,2	1,3	0,x	0,x	0,x
0,x	1,8	1,4	0,x	2,1	2,2	2,3
0,x	1,7	1,5	1,6	0,x	2,8	2,4
0,x	0,x	0,x	0,x	0,x	2,6	2,5
0,x	0,x	0,x	0,x	0,x	2,7	0,x
0,x	0,x	0,x	0,x	0,x	0,x	0,x

Example – Corresponding BFS

- And the Breadth First Search algorithm will printout the following corresponding grid of pixels:

0,x	0,x	0,x	0,x	0,x	0,x	0,x
0,x	1,1	1,2	1,4	0,x	0,x	0,x
0,x	1,3	1,5	0,x	2,1	2,2	2,3
0,x	1,6	1,7	1,8	0,x	2,4	2,5
0,x	0,x	0,x	0,x	0,x	2,6	2,7
0,x	0,x	0,x	0,x	0,x	2,8	0,x
0,x	0,x	0,x	0,x	0,x	0,x	0,x

Have Fun!