

Assignment 9

C313e - Elements of Software Design

Graph Flood Fill

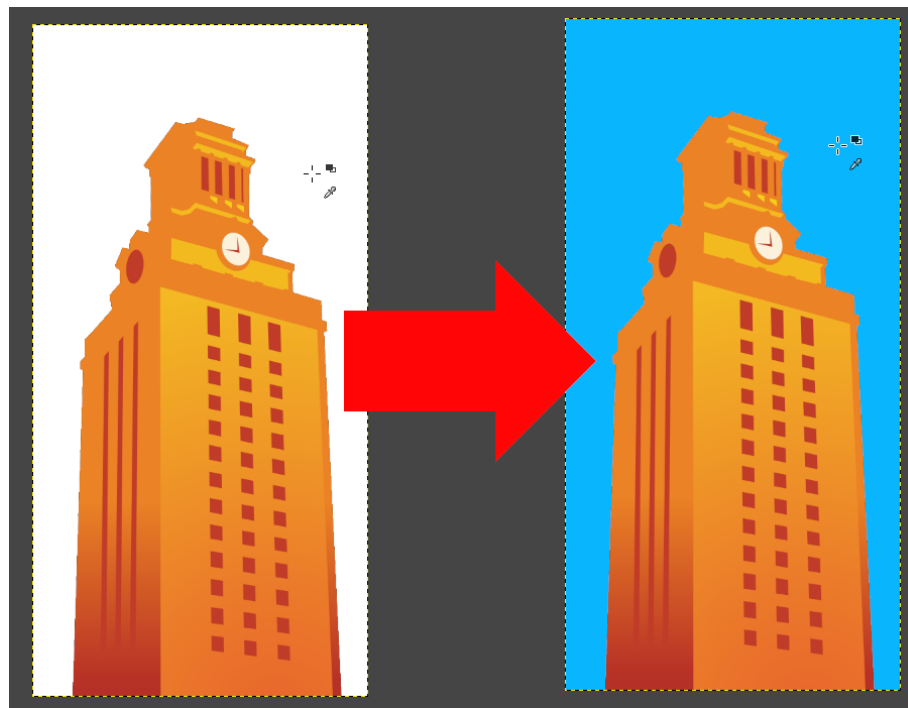
(100 points)

Due Date on Canvas and Gradescope

1 Description

In this assignment you will print the adjacency matrix for a graph and implement Breadth-First Search and Depth-First Search to flood fill pixels in images.

You may also know this feature as "bucket fill" from graphics applications. It allows you to select a pixel in an image and it will fill the selected pixel and all connected pixels of the same color with a new color, thereby allowing you to change the color of a large area of an image.



You will implement this feature using the Breadth-First Search and Depth-First Search algorithms. We treat each pixel of an input image as a node in a graph. We start at a given node/pixel and explore the graph from there, changing the color of each pixel as we visit it.

The template reads in a file containing nodes, consisting of an x-&y-coordinate and a color, and edges between the nodes to build a graph. The skeleton code also provides the function `ImageGraph.print_image()` to display the graph of `ColorNodes` as an image in the console.

In this assignment your task is to complete a python program with the name **graph_fill.py**.

2 Tasks

You are given the following tasks:

1. Read the code and make sure you understand how the graph is created (don't worry about the functions at the top of the file; these are helper functions to print out the images).
2. Print the adjacency matrix of the graph.
3. Complete the Breadth-First Search function of the Graph class.
4. Complete the Depth-First Search function of the Graph class.

Rules for the search algorithms:

- **Only visit nodes that have the same color as the starting node.**
- Color visited nodes in the new color.
- Call the `ImageGraph.print_image()` function after coloring a pixel.

Apart from the `ImageGraph` and `ColorNode` classes, you will also find a `Stack` and a `Queue` class. You might find these helpful in your search algorithm implementations.

Input:

You will read your input data from the provided *.in files. The file **small.in** contains the smallest amount of nodes and might be a good starting point to test your implementation. The format of the files will be as follows:

```
1 5
2 5
3 1,1,red
4 2,1,red
5 3,1,red
6 2,2,red
7 3,2,red
8 5
9 0,1
10 1,2
11 1,3
12 2,4
13 3,4
14 2,green
```

The first line will be the dimension of the image (width and height). This number is used to initialize the ImageGraph. The second line is the number of lines following afterwards describing the nodes. Each node description has the format

`x,y,color`.

After the nodes, there's another line with a single number telling you how many edge descriptions follow. Each edge description has the format

`from_node_index,to_node_index`.

The code template connects `from_node` to `to_node` and `to_node` to `from_node`.

Finally, there's a line telling you which node index to start the BFS and DFS algorithms on and which color to use for the flood fill.

Output:

There's three parts to the output:

1. The adjacency matrix
2. The intermediate images produced by the BFS algorithm
3. The intermediate images produced by the DFS algorithm


The adjacency matrix has a one at position x, y if there is an edge connecting the node at position x in the ImageGraph.nodes list and the node at position y in the list; otherwise it is zero in this position. There are no spaces or other delimiters between the numbers.

Make sure to call ImageGraph.print_image() after each pixel that you color and don't remove the empty `print ()` statements that add new lines from the function/add more new lines to the output.

Here's what the output for **small.in** could look like:

```
Adjacency matrix:
01000
10110
01001
01001
00110

Starting BFS; initial state:


Visited node 2



Visited node 1


Visited node 4


Visited node 0


Visited node 3


Starting DFS; initial state:


Visited node 2


Visited node 1


Visited node 0


Visited node 3


Visited node 4

```

You can additionally find desired outputs for the example ***.in** files we provided in the **out_files** directory. Use the **print_out.py** script and pass in one of the ***.out** files to print the desired outputs in the terminal. Note that there are multiple ways to traverse the graphs. The grader will accept all correct bfs/dfs implementations. The example output files show one possible solution that might help you understand what the expected output should look like.

The file that you will be turning in will be called GraphFill.py. You will follow the standard coding conventions in Python.

You may not change the names of the functions listed. They must have the functionality as given in the specifications. You can always add more functions than those listed.

For this assignment you may work with a partner. Both of you must read the paper on Pair Programming¹ and abide by the ground rules as stated in that paper. If you are working with a partner then only one of you will be submitting the code. But make sure that your partner's name and UT EID is in the header. If you are working alone then remove the partner's name and eid from the header.

2.1 Turnin

Turn in your assignment on time on Gradescope system on Canvas. For the due date of the assignments, please see the Gradescope and Canvas systems.

2.2 Academic Misconduct Regarding Programming

In a programming class like our class, there is sometimes a very fine line between "cheating" and acceptable and beneficial interaction between students (In different assignment groups). Thus, it is very important that you fully understand what is and what is not allowed in terms of collaboration with your classmates. We want to be 100% precise, so that there can be no confusion.

The rule on collaboration and communication with your classmates is very simple: you cannot transmit or receive code from or to anyone in the class in any way – visually (by showing someone your code), electronically (by emailing, posting, or otherwise sending someone your code), verbally (by reading code to someone) or in any other way we have not yet imagined. Any other collaboration is acceptable.

The rule on collaboration and communication with people who are not your classmates (or your TAs or instructor) is also very simple: it is not allowed in any way, period. This disallows (for example) posting any questions of any nature to programming forums such as **StackOverflow**. As far as going to the web and using Google, we will apply the "**two line rule**". Go to any web page you like and do any search that you like. But you cannot take more than two lines of code from an external resource and actually include it in your assignment in any form. Note that changing variable names or otherwise transforming or obfuscating code you found on the web does not render the "two line rule" inapplicable. It is still a violation to obtain more than two lines of code from an external resource and turn it in, whatever you do to those two lines after you first obtain them.

Furthermore, you should cite your sources. Add a comment to your code that includes the URL(s) that you consulted when constructing your solution. This turns out to be very helpful when you're looking at something you wrote a while ago and you need to remind yourself what you were thinking.

We will use the following Code plagiarism Detection Software to automatically detect plagiarism.

- **Staford MOSS**

<https://theory.stanford.edu/~aiken/moss/>

- **Jplag - Detecting Software Plagiarism**

<https://github.com/jplag/jplag> and <https://jplag.ipd.kit.edu/>

¹Read this paper about Pair Programming <https://collaboration.csc.ncsu.edu/laurie/Papers/Kindergarten.PDF>