

313E Programming Assignment 12

Image Bucket Fill

Setup

This assignment may be completed individually or with a pair programming partner. Be sure to include your and your partner's name and EID in the Python file header. If you are working alone, you may delete one of them.

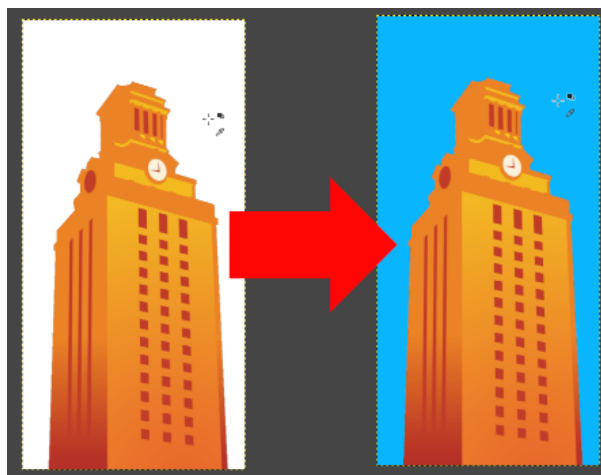
Complete these steps to prepare for the Assignment.

Check	Description
<input type="checkbox"/>	Download graph.py, small.in, chess.in, f1.in, flags.in, heart.in, horns.in, smile.in, and spiral.in. You will be working on the graph.py file. You may not change the file names. Otherwise, the grading script will not work.
<input type="checkbox"/>	Place all files in the same folder/directory.

If you would like to turn on autosave in VSCode, click on **File -> Autosave**.

Problem Description

For this assignment, you will be performing a “bucket fill” operation on an image. For example, consider the image below. If the user chose “bucket fill” and the color blue, then clicked on a pixel in the white background, all the adjacent white pixels would turn blue. (Note that the white in the clock does not turn blue, because these pixels are not adjacent to the selected pixel.)



Requirements

We have defined the **Stack**, **Queue**, and **ColoredVertex** classes for you. Based on these classes, you will be working on the **ImageGraph** class and some functions. You may add as many helper functions as needed. There are a few restrictions you must follow:

1. Your program uses the provided text (.in) files to create your final output. You will be using input redirection to read from these files.
2. **You may not change the names or parameters of the functions listed. They must have the functionality as given in the specifications. You can always add more functions than those listed.**
3. **You may not import any additional external libraries in your solution besides sys, os, and deque from collections.**

Using **small.in** as an example, you will modify the following methods. Further details breaking down the input files are listed in the **Input** section.

```
5
5
1, 1, red
2, 1, red
3, 1, red
2, 2, red
3, 2, red
5
0, 1
1, 2
1, 3
2, 4
3, 4
2, green
```

def dfs (self, start_index, color)

Do a depth first search in a graph. start_index is the index of the currently visited vertex, and color is the color you will be using to fill. You will not be returning anything in this method. For your own debugging purposes you can print the state of the graph after visiting a vertex by calling print_image().

In the example above, the start_index is **2** and the color is **green**. Below, the vertex at index **2** is filled in first. We then continue the depth first search and color vertex at index **1**, then the vertex at index **0**, and so on. The complete example for this method is in the Input section.

Starting DFS; initial state:



Visited node 2



Visited node 1



def bfs (self, start_index, color)

Do a breadth first search in a graph. Start_index is the index of the currently visited vertex, and color is the color you will be using to fill. You will not be returning anything in this method. For your own debugging purposes you can print the state of the graph after each vertex by calling print_image().

In the example above, the start_index is 2 and the color is green. Below, the vertex at index 2 is filled in first. We then continue the depth first search and color vertex at index 1, then the vertex at index 5, and so on. The complete example for this method is in the Input section.

Starting BFS; initial state:



Visited node 2



Visited node 1



def print_adjacency_matrix (self)

Print the adjacency matrix. The adjacency for the example above is as follows:

```
01000
10110
01001
01001
00110
```

def create_graph (data)

Using the input data, create the graph. You will need the size of the image and the number of vertices to create your graph and vertices. Each vertex will be constructed with (x, y, color). You will need to create edges between the vertices, with format “fromIndex, toIndex”. Make sure that you create the edges both ways, for instance, connecting vertex 1 to vertex 2, and vertex 2 to vertex 1. Lastly, you will return the ImageGraph object, along with the starting position and color as a tuple in that order.

Using the example above, the size of the graph created would be of size **5** and the color of the vertices at creation is red.





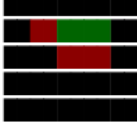





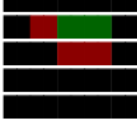
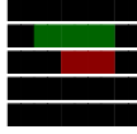


Input

Several input files (*.in) are provided. Each file includes a text representation of a graphic image, followed by the pixel and color for the bucket fill request. This is the **small.in** data, with descriptions of the file format:

Info	Input File Content	Description
Image size	5	The size of the square image (height and width).
Vertices/pixels	5 1, 1, red 2, 1, red 3, 1, red 2, 2, red 3, 2, red	The first line is the number of lines in the file representing one vertex/pixel. The following lines are the x value, y value and color for one vertex/pixel. Vertices are 0-indexed.
Edges	5 0, 1 1, 2 1, 3 2, 4 3, 4	The first line is the number of lines in the file representing one edge. The following lines are the two vertices that are adjacent and share the same color. Note that each edge is listed only one time, with the smaller number first.
Type of bucket fill	2, green	The index of the vertex/pixel to begin the bucket fill with and the color to change the vertices /pixels to.

Output

Your program will print to standard output. If you have chosen to print the state of your graph after each vertex, these will be printed as part of the output. Otherwise, only the adjacency matrix will be printed. These will display one after the other on the console. Note that there are multiple possible solutions for the BFS and DFS Bucket Fills.

Adjacency Matrix	BFS Bucket Fill	DFS Bucket Fill
01000 10110 01001 01001 00110	<p>Starting BFS; initial state:</p>  <p>Visited node 2</p>  <p>Visited node 1</p>  <p>Visited node 4</p>  <p>Visited node 0</p>  <p>Visited node 3</p> 	<p>Starting DFS; initial state:</p>  <p>Visited node 2</p>  <p>Visited node 1</p>  <p>Visited node 0</p>  <p>Visited node 3</p>  <p>Visited node 4</p> 

Running Your Program

You can test your program using the **.in** files we have provided for you. You can run the program by typing the following command into your terminal, replacing the **.in** file with the desired input file:

```
python3 graph.py < small.in
```

Alternatively, if you are using a Windows computer and are having trouble with input redirection **<**, you can run:

```
cat small.in | python3 graph.py
```

or depending on what python version you installed,

```
cat small.in | python graph.py
```

If you are having a hard time viewing your output (because there are so many vertices to color, and you are printing the image after every vertex), then you can try **output redirection**. For Mac, the command should be: `python3 graph.py < small.in > small.ansi`

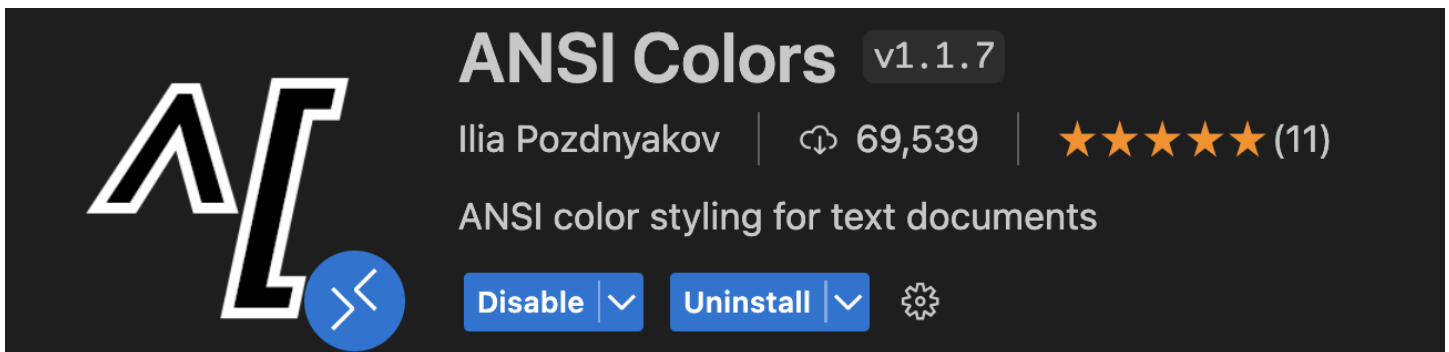
For Windows, the command should be:

```
cat small.in | python3 graph.py > small.ansi
```

or

```
cat small.in | python graph.py > small.ansi
```

The `> small.ansi` will now create a file called `small.ansi`. It will contain all of your program's output. You can then install the ANSI Colors VSCode extension:



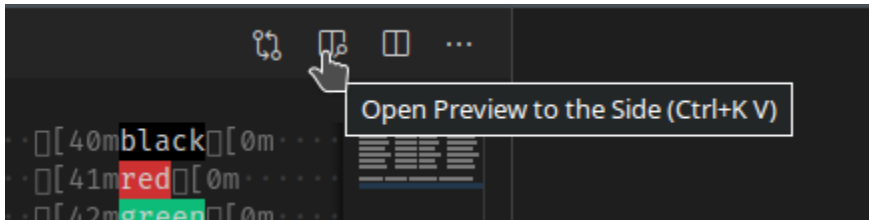
Select the ANSI Text language mode to highlight text marked up with ANSI escapes. Files with the `.ans` and `.ansi` extensions will be highlighted by default.



Run ANSI Text: Open Preview in the command palette (`cmd+shift+P` for Mac or `ctrl+shift+p` for Windows) for the prettified read-only preview.

```
Preview: demo.ans X
1  black·····bright black·····black·····bright black
2  red·····bright red·····red·····bright red
3  green·····bright green·····green·····bright green
4  yellow·····bright yellow·····yellow·····bright yellow
5  blue·····bright blue·····blue·····bright blue
6  magenta·····bright magenta·····magenta·····bright magenta
7  cyan·····bright cyan·····cyan·····bright cyan
8  white·····bright white·····white·····bright white
9
10 bold dim italic underline
11 |
```

Or try clicking the preview icon in the editor title to open the preview in a new tab. Alt-click to open in the current tab.



Grading

Visible Test Cases - 75/100 points

The program output exactly matches the expected output for each function. To receive full credit, the program must produce the correct output based on all requirements in this document and pass all the test cases.

For the visible, test cases - if your code works as intended and the test cases pass - the expected output on the Gradescope will actually be nothing as shown below.

0) Test create_graph chess.in (2.5/2.5)

Expected Output:

Your Output

If there is an error in your code, some information will be provided in your output for what has failed.

Hidden Test Cases - 15/100 points

The program output exactly matches the expected output for each function. To receive full credit, the program must produce the correct output based on all requirements in this document and pass all the test cases.

Style Grading - 10/100 points

The graph.py file must comply with the [PEP 8 Style Guide](#).

If you are confused about how to comply with the style guide, paste the error or the error ID (e.g. C0116 for missing function or method docstring) in the search bar here in the [Pylint Documentation](#), and you should find an example for how to fix your problem.

The TAs will complete a manual code review for each assignment to confirm that you have followed the requirements and directions on this document. Deductions will occur on each test case that fails to follow requirements.

Submission

Follow these steps for submission.

Check	Description
<input type="checkbox"/>	Verify that you have no debugging statements left in your code.
<input type="checkbox"/>	Submit only graph.py (the starter code file with your updates) to the given assignment in Gradescope. This will cause the grading scripts to run.
<input type="checkbox"/>	If you have a partner, make sure you are submitting it for you and your partner on Gradescope. See this Gradescope documentation for more details.

When the grading scripts are complete, check the results. If there are errors, evaluate the script feedback, work on the fix, test the fix, and then re-submit the file to Gradescope. You can submit as many times as necessary before the due date.

NOTE: By default, only the most recent submission will be considered for grading. If you want to use a previous submission for your final grade, you must activate it from your submission history before the due date.

Academic Integrity

Please review the Academic Integrity section of the syllabus. We will be using plagiarism checkers, and we will cross-reference your solution with AI-generated solutions to check for similarities. Remember the goal in this class is not to write the perfect solution; we already have many of those! The goal is to learn how to problem solve, so:

- Don't hesitate to ask for guidance from the instructional staff.
- Discuss with peers about **only** bugs, their potential fixes, **high-level** design decisions, and project clarifying questions. Do not look at, verbalize, or copy another student's code when doing so.

Attribution

Thanks to Dr. Carol Ramsey and Dr. Kia Teymourian for the instructions template and this assignment.