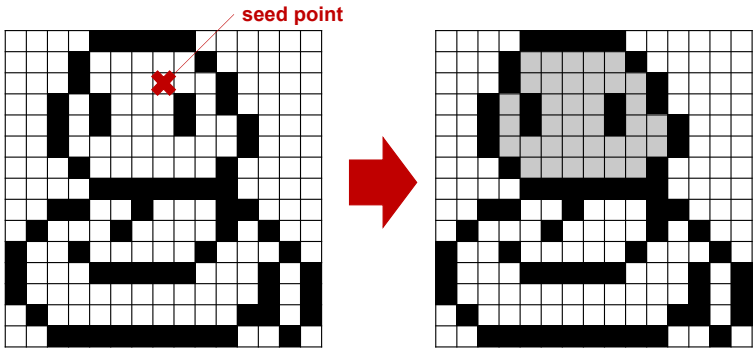
# Coursework 1: Bucket Fill

COMP70053 Python Programming (Autumn 2021)

**Deadline: Friday 15th Oct 2021 (7pm BST) on CATe**




## 1 Overview

In this coursework, you will design and implement an algorithm to 'fill' a given pixel-based line drawing given a seed point.

The objectives of this coursework are:

- to enable you to apply what you have learnt so far to design an algorithm for a practical programming problem;
- to allow you to practise implementing the algorithm you designed in Python; and
- to give you hands-on experience in testing your implementation.

This coursework covers topics in Lessons 1 to 6 of the course materials. You may also consider using recursion as discussed in Lesson 7. This coursework can be solved iteratively (via loops) or recursively. If using recursion, please ensure that your implementation works for input image sizes up to $25 \times 25$ or you may lose marks.

You are limited to the **Python Standard Library** for this coursework. Do **NOT** use any external libraries such as `NumPy`.

## 2 Tasks

You are supplied with some skeleton code (see section 3).

### 2.1 Task 1: Design and implement your algorithm

Your task is the **complete the `fill()` function** in `bucket_fill.py`. The function takes two input arguments:

1. `image` is a two-dimensional nested list representation of an image (see Figure 1 below);
2. `seed_point` is a tuple `(row, col)` of the seed point to start filling.

You may assume that elements in the input `image` consist only of the integers `0` or `1`, where

- `0` corresponds to an unfilled pixel
- `1` corresponds to a boundary pixel (a drawn 'line')

The function should fill the given `image` with the integer `2`, starting from the seed point and extending to its enclosing boundary (i.e. until it encounters `1`). The function should return a copy of the filled `image`.

For example, the image in Figure 1a is represented by the following `list`:

```
original_image = [[0, 0, 0, 0, 0, 0],
                  [1, 0, 1, 1, 1, 0],
                  [0, 1, 0, 0, 0, 1],
                  [0, 0, 1, 1, 1, 0],
                  [0, 0, 0, 0, 0, 0]]
```

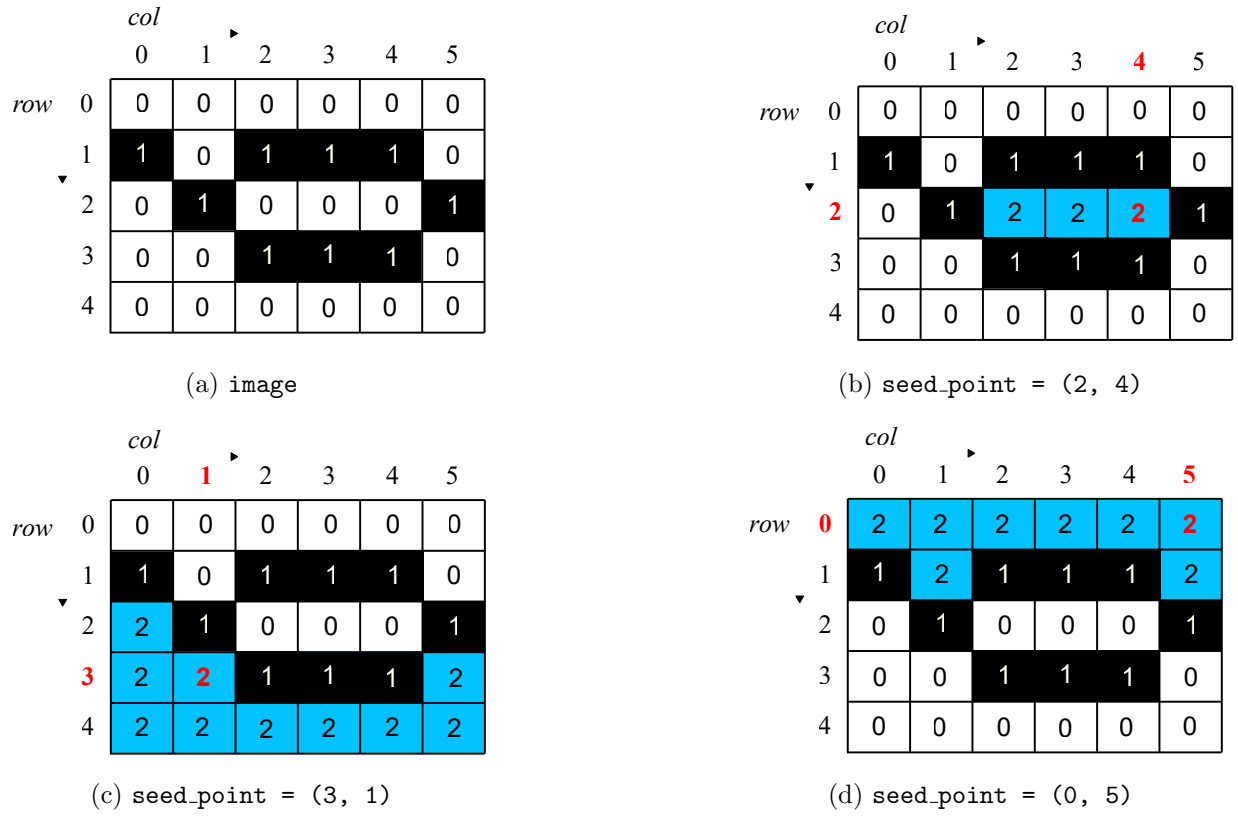
Applying `fill()` to this image with a seed point `(2, 4)` should return the following `list` (see Figure 1b):

```
filled_image = [[0, 0, 0, 0, 0, 0],
                [1, 0, 1, 1, 1, 0],
                [0, 1, 2, 2, 2, 1],
                [0, 0, 1, 1, 1, 0],
                [0, 0, 0, 0, 0, 0]]
```

More examples are shown in Figure 1c and Figure 1d.

The function should return the original `image` when an invalid `seed_point` is given. For example, `seed_point=(6, 2)` is an invalid input for the example above, as this `seed_point` is outside of the image.

You may add more functions as you see fit, as long as the `fill()` function is defined.




(a) `image`

(b) `seed_point = (2, 4)`

(c) `seed_point = (3, 1)`

(d) `seed_point = (0, 5)`

Figure 1: Bucket fill example

### 2.2 Task 2: Test your function

You should also **write some test cases** to ensure that your function is working correctly.

You may assume that the input `image` will always be valid (made up of only `0` and `1`'s), but can be of any size up to $25 \times 25$.

You may also assume that `seed_point` will always be a tuple, although the elements in the tuple may not always be valid. Hints for invalid cases are provided in the docstrings!

**Utility functions:** We have provided some utility functions in `bucket_fill.py` to aid your development and testing:

- `load_image(filename)` reads an image from a given text file, and returns a nested list that can be used directly as input to `fill()`.
- `show_image(image)` helps you visualise an image in the terminal.

### 2.3 Task 3: Describe your algorithm

Please prepare a PDF (**MAXIMUM ONE PAGE**) describing your algorithm. Name this file `readme.pdf`. The aim is to help the marker understand your algorithm easily. You can use plain English, pseudocodes, diagrams, hand-drawings, flowcharts, examples, etc. Even one short paragraph is acceptable. Keep this simple, to the point, and do not spend too much time on this - we will **NOT** be assessing the quality of your explanation or how beautiful your diagrams are. This is just to help us ensure that we do not misunderstand your submitted code.

## 3 Skeleton code

You are provided with a git repository containing the skeleton code. You will use `git` to push your code changes to the repository. To obtain the code, clone the repository using either SSH or HTTPS, replacing `<login>` with your College username.

- `git clone git@gitlab.doc.ic.ac.uk:lab2122_autumn/python_cw1_<login>.git`
- `git clone https://gitlab.doc.ic.ac.uk/lab2122_autumn/python_cw1_<login>.git`

Test your code with `LabTS` (`https://teaching.doc.ic.ac.uk/labts`) after pushing your changes to GitLab. Make sure your code runs successfully on LabTS without any syntax errors. You will lose marks otherwise. These automated tests are purely to ensure that your code runs correctly on the `LabTS` servers. **You are responsible for testing your own code more extensively**. The final assessment will be performed using a more extensive set of tests not made available to you.

## 4 Handing in your coursework

Go to `LabTS` (`https://teaching.doc.ic.ac.uk/labts`), and click on the "`Submit to CATe`" button next to the specific commit you wish to submit. This will automatically submit your commit SHA1 hash to CATe.

Additionally, please go to CATe and submit `readme.pdf` (see Section 2.3) separately.

Make sure that you submit **BOTH** items on time. CATe will consider your submission as late if either of these is late!

## 5 Grading scheme

| Criteria | Max | Details |
|---|---|---|
| Algorithm design & code correctness | 9 | Is your algorithm sensible, appropriate and logical? Does your code run without any syntax errors? Does your code give the expected output for valid standard test cases? |
| Code robustness | 4 | How well does your code pass our automated stress test with different kind of inputs, valid and invalid? Is it error free from both valid and invalid input? |
| Code readability | 4 | Did you use good coding style and naming conventions? Did you use meaningful names? Is your code highly abstracted, modular, and self-explanatory? Did you provide useful, meaningful comments where necessary? |
| Software testing | 3 | Did you write any test cases? How extensively did you test for valid/invalid inputs and images of different sizes? |
| **TOTAL** | **20** | Will be scaled to 4% of your final module grade. |

## Credits

Coursework designed and prepared by Luca Grillotti and Josiah Wang