Lab 7: Recursion

SUNY Korea - Francois Rameau

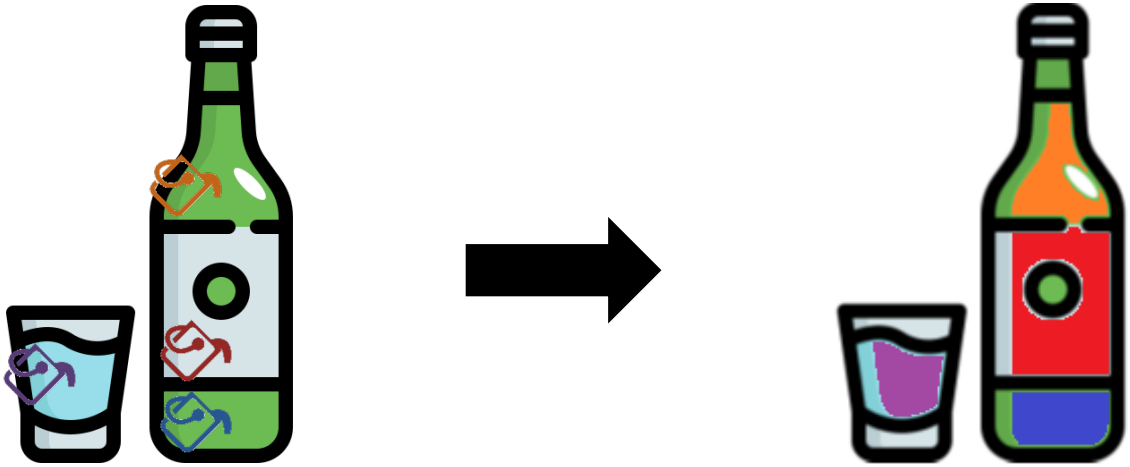
GitHub Classroom

Lab 7 – Recursion

MS-Paint

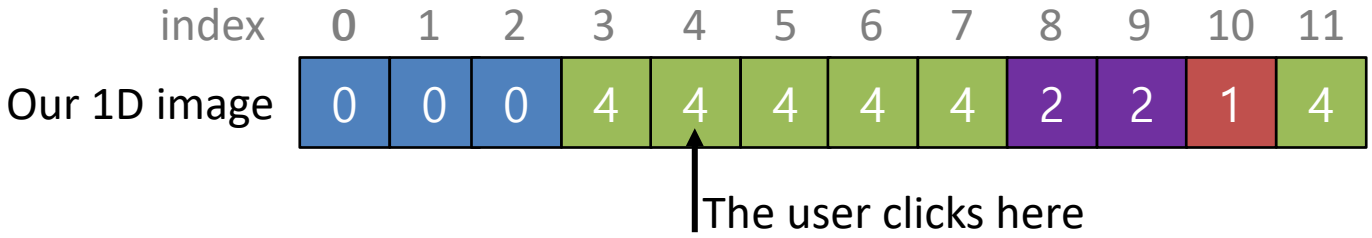
Do you know this symbol in paint/photoshop?

When we bring the bucket to a pixel and click, the color of the region of that pixel is replaced with a new selected color.



Task1: implement 1D bucket tool

Instead of a 2D image, we will start in 1D

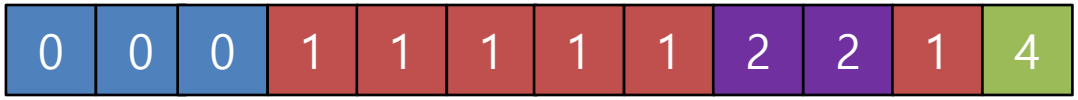


1 User new color!

Goal

1. The user select a “new color” (red)
2. The user clicks on one pixel → pixel index x=4
3. We look at the value of the selected pixel → old color 4
4. All the adjacent green pixels have to be replace with the new color

After our filling!



Task1: implement 1D bucket tool

You will have to implement this strategy in a recursive manner in a single function:

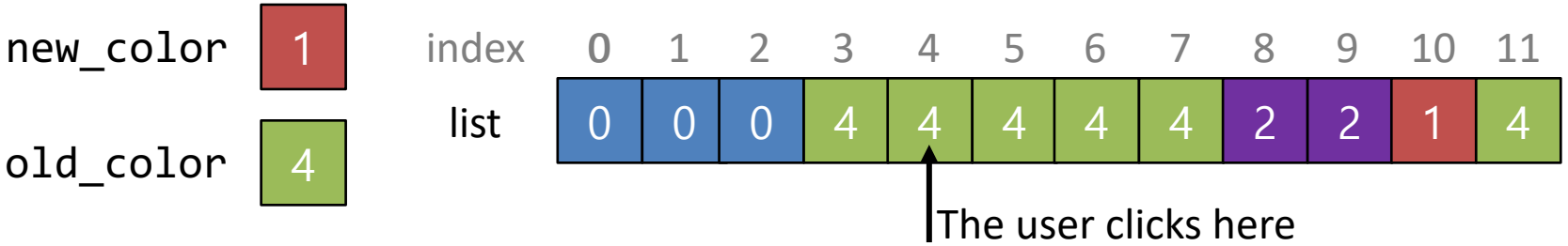
```
colorFill1D(lst, x, new_color, old_color)
```

This function will take the following arguments

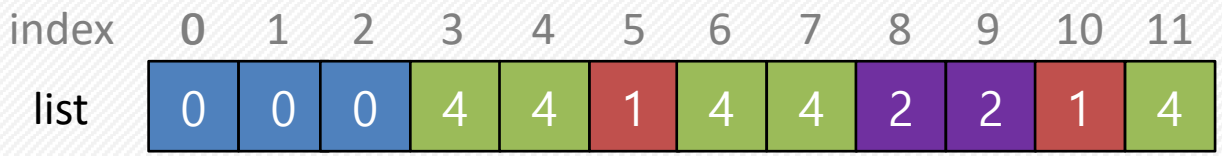
`lst`		The list of pixel you want to process
`x`		The position of the current pixel
`new_color`		The new color you will use to replace the old one
`old_color`		The old color you would like to replace

Task1: implement 1D bucket tool

How do we solve this problem recursively?

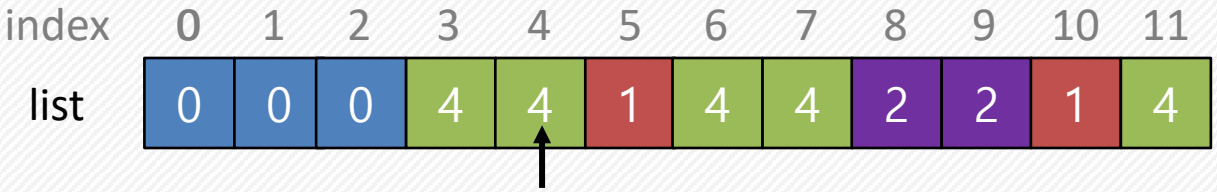


Step 1: `if the current pixel at index x is not "out of index" and if it has the old_color value replace it with the new_color`



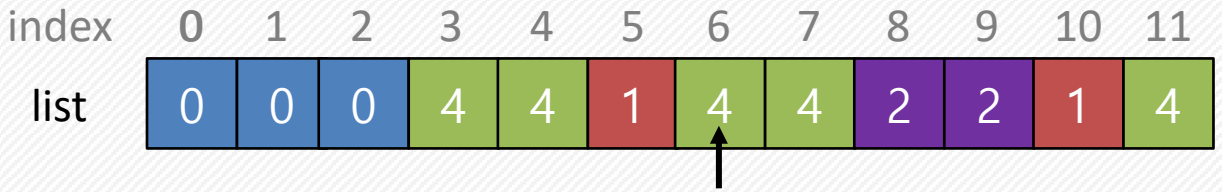
Task1: implement 1D bucket tool

Step 2:check the left neighbor



- If the left neighbor has the same value as the old color, recursively call the `colorFill1D` function with the left neighbor's index.

Step 3:check the right neighbor



- If the right neighbor has the same value as the old color, recursively call the `colorFill1D` function with the right neighbor's index.

Task1: implement 1D bucket tool

Tips

- While implementing the function, think about the base cases needed to stop the recursion! (when facing base case → `return`)

- Ensure you understand how the recursion works, especially when the function calls itself for neighboring elements.

- One possible base case might be when the index is outside of the list

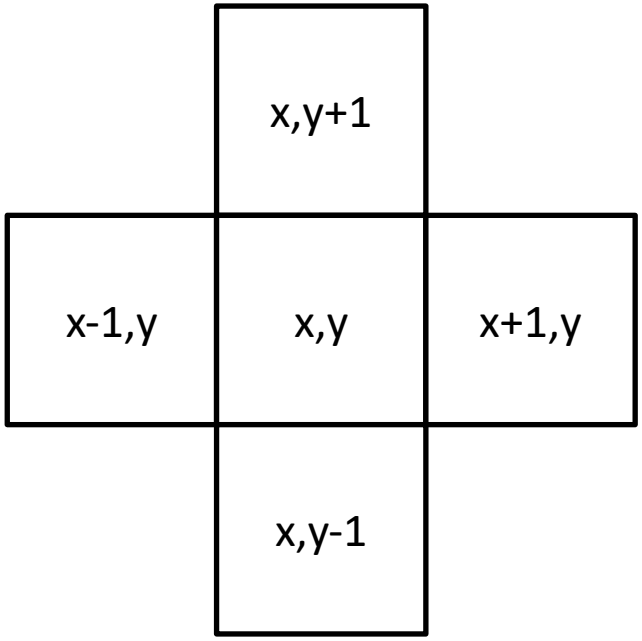
Task2: implement 2D bucket tool

Can we apply the same process in 2D?

0	0	0	4	4	1	4	4	2	3
1	1	0	4	4	1	4	4	3	3
0	0	0	4	1	1	4	4	3	3
0	0	0	4	1	1	1	1	3	2
0	2	0	4	4	1	4	4	2	2
2	2	2	4	1	1	4	4	2	2
0	0	2	4	4	1	4	4	2	2

Task2: implement 2D bucket tool

You will apply the very same strategy but considering 4 neighbors



Implement the 2D function in:

```
colorFill2D(matrix, x, y, new_color, old_color)
```