Fakultät für Informatik
Institut für optische Systeme
Prof. Dr. G. Umlauf

Hochschule Konstanz
Technik, Wirtschaft und Gestaltung

Konstanz, 05.11.2022

Übungsblatt 7 für WIN und BIT

„Algorithmen und Datenstrukturen WS 2022/23“

Abgabe: bis 24.01.2023

Aufgabe 11 — 6+4+5 Punkte

Vorbereitung

Kopieren Sie die Datei **`FloodFill.zip`** in dieses neue Verzeichnis von Moodle, entpacken Sie diese Datei und starten das Projekt in Ihrer IDE.

Programmierung

In dieser Aufgabe sollen Sie ein Programm schreiben, das ein Bild (**`.bmp`**) einliest, die Hintergrundfarbe mit dem flood-fill-Algorithmus ändert und das geänderte Bild wieder abspeichert (**`.bmp`**).

Zum Lesen/Schreiben eines Bildes im **`bmp`**-Format sowie zum Ändern der Farbe eines einzelnen Pixels des Bildes steht Ihnen eine sehr einfache Bitmap-Klasse (**`BitmapImage.java`**), eine Farb-Klasse (**`Color.java`**) und eine Pixel-Klasse (**`Pixel.java`**) zur Verfügung. Hier brauchen Sie nichts zu ändern. Nähere Informationen finden Sie in den entsprechenden **`java`**-Dateien.

Zum Ändern der Hintergrundfarbe (**`fromColor`**) auf eine neue Farbe (**`toColor`**) muss zuerst ein Pixel des Hintergrundes als sogenannter Seed (Samen) ausgewählt werden. Ausgehend von diesem Seed-Pixel werden die acht Nachbarpixel getestet, ob sie auch zum Hintergrund gehören, d.h. auch die Hintergrundfarbe haben. Falls nein, muss mit ihnen nichts gemacht werden; falls ja, müssen auch sie umgefärbt werden. Dazu werden sie auf einen Stack gelegt, damit sie zu einem späteren Zeitpunkt umgefärbt werden können und damit auch ihre jeweiligen Nachbarn getestet werden können, ob diese auch zum Hintergrund gehören. Solange der Stack also Pixel enthält, sind nicht alle Hintergrund-Pixel umgefärbt. Der flood-fill-Algorithmus läuft im Wesentlichen so ab, wie in dem Pseudo-Code-Fragment **Algorithmus 1** dargestellt.

```
...
Push seed-pixel to stack;

while(stack not empty) {
   pop top pixel from stack;
   change color of pixel to toColor;
   for ( each neighbor )
   {
      if( color of neighbor is fromColor and
          neighbor is/was not on stack) push neighbor to stack;
   }
}
...
```

Algorithmus 1: Pseudo-Code des flood-fill-Algorithmus.

Dabei kann es dazu kommen, dass ein Pixel mehrfach auf den Stack gelegt werden soll, siehe **Abbildung 1**. Um dies zu verhindern, wird in einer zweidimensionalen Matrix ein flag für jedes Pixel gespeichert, das **`true`** ist, wenn das jeweilige Pixel bereits im Stack ist/war, und **`false`** anderenfalls.

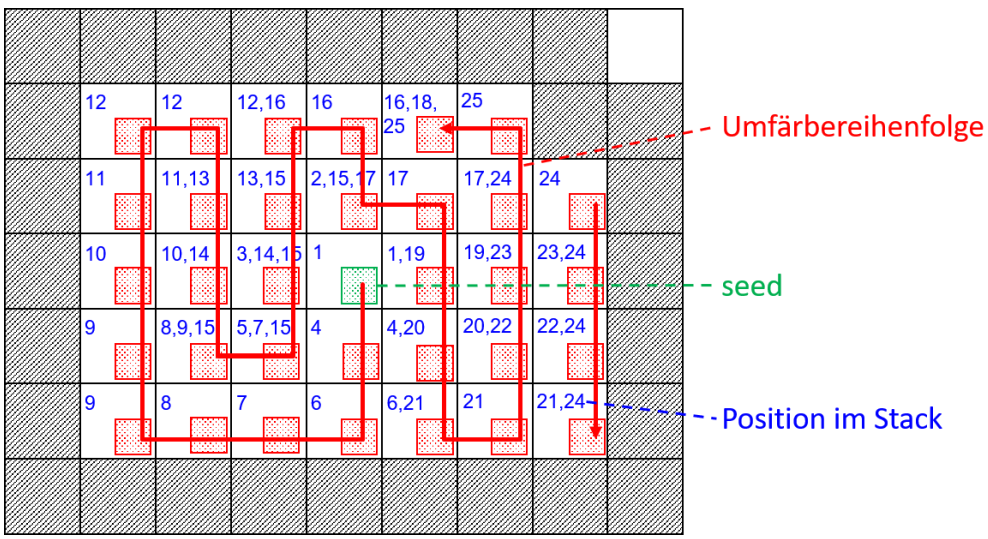

Abbildung 1: Schematische Darstellung des flood-fill-Algorithmus.

Aufgabe

1. **Generische Stack-Klasse:** Implementieren Sie eine generische Klasse für Stacks. Implementieren Sie dazu alle notwendigen Methoden auf Basis eines dynamischen Feldes, das wachsen und schrumpfen kann und soll.

2. **Generische Matrix-Klasse:** Implementieren Sie eine generische Klasse für Matrizen basierend auf einem zweidimensionalen dynamischen Feld.

3. **Hauptprogramm:** Implementieren Sie den flood-fill-Algorithmus wie oben beschrieben mit Hilfe eines Stacks von Pixeln und einer Flag-Matrix von `bool`-Werten.