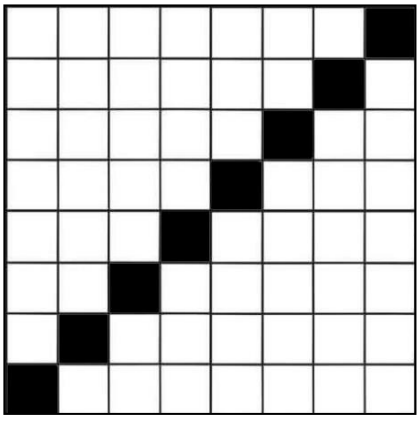
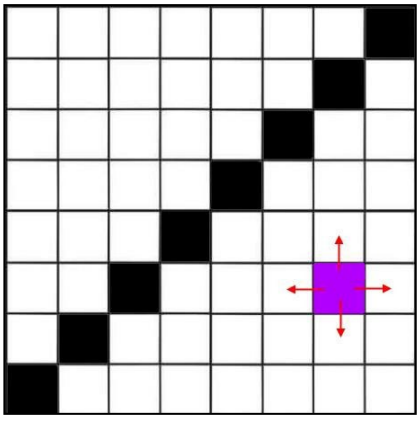
Flood Fill (ou preenchimento de inundação) é um algoritmo que tem como objetivo alterar a informação de nós (nodes) que estão conectados. Esse algoritmo é utilizado na ferramenta "balde" de softwares de desenho (como paint) e pode ser utilizado também em jogos como Go e Campo Minado para determinar quais posições serão limpas.
A proposta desta atividade é a implementação do algoritmo Flood Fill de forma simplificada, utilizando Pilha e Fila para armazenamento dos pixels. Normalmente este algoritmo é implementado utilizando recursividade, mas se preferir pode utilizar um loop de repetição no lugar. Segue abaixo um exemplo do funcionamento do algoritmo:

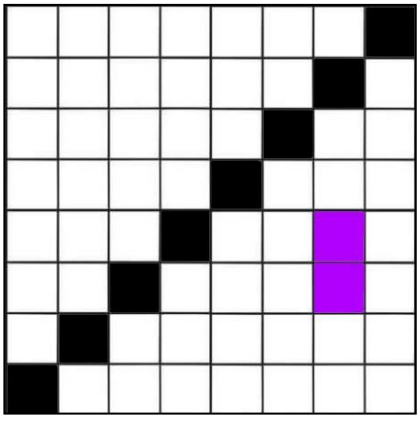
Imagine que cada quadrado na imagem é um pixel de uma imagem qualquer. No exemplo a imagem possui um fundo inteiro branco, e uma linha preta na diagonal. Essa imagem pode ser representada por uma matriz de pixels no nosso código.



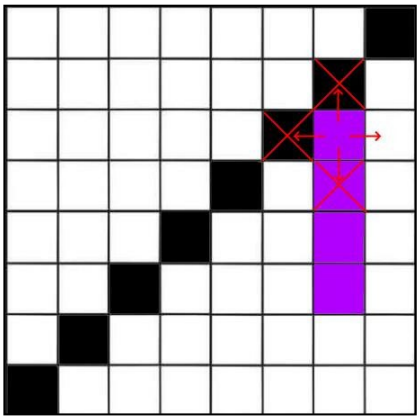
Informamos ao programa uma coordenada inicial para obter um ponto de partida e guardamos a cor de fundo em uma variável. Esse ponto é armazenado numa Pilha/Fila e então começa o nosso loop de repetição. O ponto inicial é desempilhado/desenfileirado e preenchido com a nova cor.
Em seguida, empilhamos/enfileiramos os 4 vizinhos laterais deste ponto.



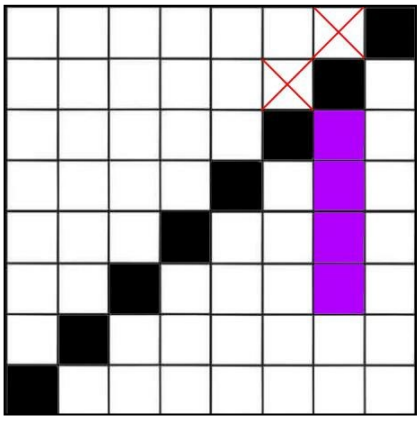
Começamos o loop novamente, e então pintamos o pixel que foi desempilhado/desenfileirado. É importante que sempre ocorra uma verificação para checar se o pixel não está ultrapassando o limite da matriz (*Index Out Of Bounds*) e se a cor é igual a cor de fundo que foi armazenada no início do programa em uma variável (no caso desse exemplo, branco). Só iremos pintar o pixel caso essa condição seja atendida. Esse processo continua se repetindo enquanto a condição for atendida.



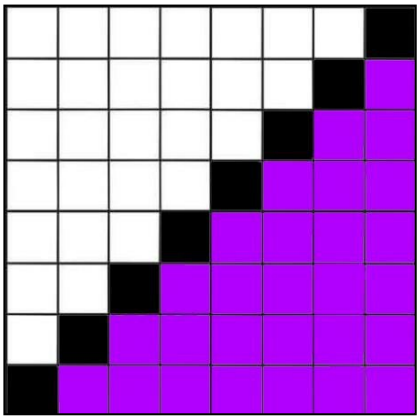
Pixels de cores diferentes do que a cor de fundo armazenada inicialmente são empilhados/enfileirados, mas nunca são pintados, pois não atendem as condições da checagem que é realizada dentro do loop (descrita anteriormente). O mesmo acontece com pixels inexistentes na imagem (fora da matriz).



Por não atenderem as condições de checagem, seus vizinhos também nunca são empilhados/enfileirados.



O algoritmo encerra quando todos os pixels tiverem sido preenchidos.



## 1. O Trabalho

As equipes deverão utilizar uma imagem qualquer como base e realizar a alteração das cores dos pixels. A imagem escolhida deve possuir cores sólidas ou apenas um fundo branco com divisões em preto. No caso se a imagem for uma foto não será obtida uma boa visualização do resultado.
Nesta versão você precisará utilizar as classes *File* e *BufferedImage* do Java. Pesquise na documentação oficial ou em fóruns e vídeos na internet como trabalhar com imagens em Java.
A implementação deve conter uma imagem como entrada e uma imagem como saída, ou seja, a imagem original que deve ser colorida e a imagem resultante.
A implementação que implementar uma animação da imagem sendo colorida, ou seja, armazenar uma nova imagem toda vez que um pixel for modificado (ou a cada X imagens caso a imagem seja muito grande), será adicionado 1,0 extra na nota do trabalho.

## 2. Especificações Gerais

As equipes devem ter no mínimo 2 e no máximo 3 integrantes. **Equipes com mais ou menos integrantes que não foram previamente aprovados pela professora terão seus trabalhos desconsiderados**.

O trabalho deve seguir os princípios da Programação Orientada a Objetos, ou seja, deve ser modulado em classes e não pode ser implementado todo na classe Main. Além disso, o algoritmo deve ter duas formas diferentes de armazenamento dos valores: um utilizando Pilha e outro utilizando Fila. As equipes devem utilizar suas próprias estruturas. **Trabalhos que utilizarem as estruturas prontas do Java serão zerados.**

Não é necessário utilizar recursividade na implementação, é possível utilizar um loop de repetição. Fica a critério de cada equipe decidir como prefere implementar.

Todas as equipes farão prova de autoria sobre a implementação realizada. Os integrantes que não realizarem as modificações solicitadas, **terão suas notas zeradas**.

**Códigos copiados ou parafraseados da internet serão considerados como plágio e terão suas notas zeradas. Assim como códigos iguais (ou extremamente semelhantes) entre as equipes, serão questionados em prova de autoria e, caso seja identificado plágio entre as equipes, ambos os trabalhos serão zerados**.

**Os estudantes que não comparecerem, sem justificativa formal, no dia estipulado para prova de autoria, terão suas notas zeradas**.

## 3. Composição da Nota

### 3.1. Requisitos Mínimos

1. O algoritmo apresenta uma solução utilizando Pilha (própria). (1,5)
2. O algoritmo apresenta uma solução utilizando Fila (própria). (1,5)
3. O algoritmo de Pilha e Fila foram implementados corretamente. (3,0)
4. O código segue os princípios e boas práticas da Programação Orientada a Objetos. (2,0)
5. O programa funciona corretamente do início ao fim (sem bugs) e foi implementado de acordo com asespecificações mencionadas nos itens 1 e 2 deste documento. (2,0)

### 3.2. Requisitos Extras

1. A execução do programa demonstra, com animação de imagens, o processo de alteração de cor de pixels. (1,0)