



UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

Relatório **Trabalho prático GenericStuff**

Processamento de Imagens - Turma 01
Prof. Dr. Beatriz Trinchão Andrade

Gabriel de Oliveira Santos
g4briel720@academico.ufs.br
gabriel.santos1@dcomp.ufs.br



São Cristóvão – Sergipe

2023

1

Primeira Parte

1.1 Com base no que foi solicitado, o que foi implementado pelo grupo?

A solução em si, em sua essência, consiste na aplicação de um algoritmo de busca em profundidade, também conhecido como *Depth-First Search*, adaptado para o contexto do problema.

Essa adaptação foi feita para que a DFS (depth-first search) siga de acordo com a definição de vizinhança de um pixel pertencente a um objeto e também com a definição de vizinhança de um pixel pertencente a um furo. Sendo assim, a DFS que busca por objetos na imagem irá expandir a busca dentro da imagem levando em consideração os pixels pretos ($imagem(x,y) = I$) em uma vizinhança 8, ou seja, aplicando os seguintes movimentos: $[(1,0), (-1,0), (0,1), (0,-1), (-1,1), (1,1), (1,-1), (-1,-1)]$, e, durante a sua expansão, além de identificar os objetos da imagem, também os armazena em uma estrutura de dados.

Já a DFS que leva em consideração os pixels brancos (furos e fundo da imagem), usada no algoritmo de flood fill, percorre a imagem de acordo com a vizinhança de um pixel pertencente a um furo que, neste caso, se trata de uma vizinhança 4. Sendo assim, aplica os seguintes movimentos: $[(1,0), (-1,0), (0,1), (0,-1)]$.

As vizinhanças dos pixels podem ser vistas de maneira visual, desta forma:

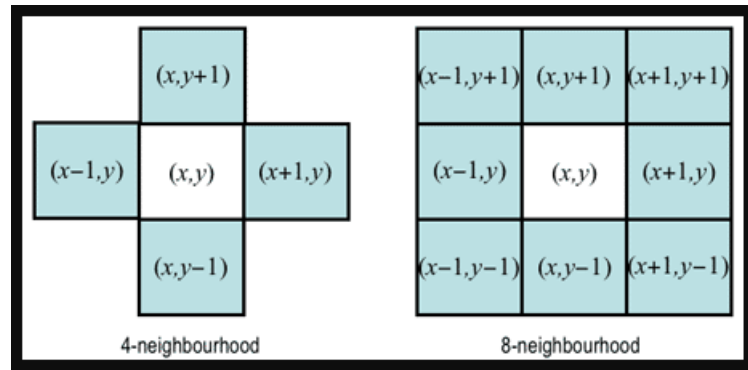


Figura 1 – À esquerda, uma vizinhança 4 e à direita, uma vizinhança 8.

Tendo em mente como funciona os dois principais algoritmos da solução, a mesma pode ser descrita, através de um pseudo-código simples, da seguinte maneira:

Algorithm 1 GenericStuffSolution

```

1: function GENERICSTUFFSOLUTION(imagem)
2:   AddBorderImg(imagem)
3:   CountObjectsInImg( imagem, true, validObjectCoords )
4:   imageSegments ← GetImageSegments(img, validObjectCoords)
5:   segmentsWithHoles ← GetSegmentsWithHoles(img, validObjectCoords)
6:   Print(imageSegments.Length)
7:   Print(segmentsWithHoles.Length)
8: end function

```

Em relação a cada etapa do algoritmo, têm-se:

- AddBorderImg(*imagem*): adiciona bordas à imagem com pixels brancos. Consequentemente, os segmentos obtidos nas etapas posteriores também terão essa borda e isso facilitará a aplicação do algoritmo de Flood Fill baseado em uma DFS;
- CountObjectsInImg(*imagem*, *true*, *validObjectCoords*): conta os objetos pertencentes à imagem e os armazena na estrutura **validObjectCoords**. O parâmetro **true** sinaliza para a DFS que a vizinhança adotada será a vizinhança 8;
- *imageSegments* = GetImageSegments(*img*, *validObjectCoords*): obtém os segmentos da imagem através dos objetos armazenados na estrutura citada na etapa anterior. Cada objeto dará origem a um segmento da imagem que será armazenado na estrutura **imageSegments**;
- *segmentsWithHoles* = GetSegmentsWithHoles(*imageSegments*): nessa etapa, para cada segmento obtido na etapa anterior, será aplicado um Flood Fill e um negativo. Desta forma, caso o segmento possua um furo, ele será destacado após apenas o fundo ser preenchido com a cor preta e, por conta da inversão, toda a região preenchida se tornará branca e o furo, consequentemente, preto. Caso o segmento não possua um furo, após a aplicação

dos métodos citados, a imagem resultante é totalmente vazia (branca) e não será retornada à estrutura de dados resultante;

- `Print(imageSegments.Length)`: printa no console a quantidade de segmentos (ou seja, objetos) identificados na imagem;
- `Print(segmentsWithHoles.Length)` printa no console a quantidade de segmentos que possuem um ou mais furos (ou seja, objetos furados) identificados na imagem.

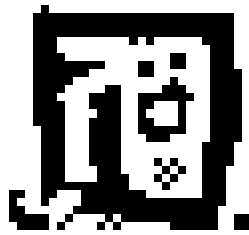


Figura 2 – Imagem .PBM de exemplo

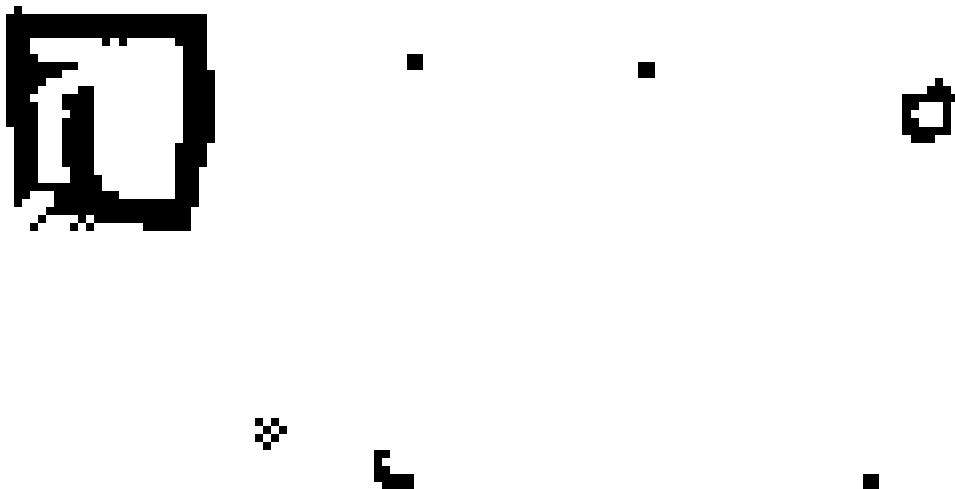


Figura 3 – Objetos/segmentos armazenados na estrutura **imageSegments**

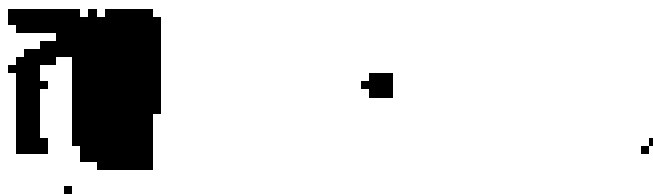


Figura 4 – Segmentos com os furos de cada objeto furado, armazenados na estrutura **segmentsWithHoles**

1.2 Quais as técnicas aprendidas na disciplina foram aplicadas e que parâmetros foram usados?

Utilizou-se a técnica de leitura de imagens do tipo PGM ASCII vista no exercício prático proposto no início da disciplina.

Além disso, utilizou-se os conceitos de segmentação/visão dos planos da imagem que foram vistos em alguns exercícios de fixação. Além disso, utilizou-se a operação pontual **negativa**.

A técnica de Flood Fill não chegou a ser abordada oficialmente em aula, porém, ao consultar os materiais didáticos (livro texto e internet), achou-se interessante aplicar a técnica no projeto.

1.3 Se foram necessárias asserções sobre o tamanho ou forma dos objetos. Caso sim, explicitar quais foram.?

Não foi necessário.

1.4 Instruções sobre como compilar e executar o projeto e possíveis dependências no Linux

Para compilar e executar o projeto será necessário ter em mãos a pasta do projeto e instalar o SDK do DotNet, pois o mesmo foi feito em C#.

Obs1.: se atentar à versão do SDK a ser instalada, é imprescindível que seja instalada a versão 6.0, pois o projeto foi feito em cima dela.

Obs2.: para instalar em outras versões do Ubuntu, basta substituir o primeiro comando por algum destes:

- 22.04 (LTS): o repositório da MS já consta no sistema, basta executar o último comando;
- 20.04 (LTS): `wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb`
- 18.04 (LTS): `wget https://packages.microsoft.com/config/ubuntu/18.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb`
- 16.04 (LTS): `wget https://packages.microsoft.com/config/ubuntu/16.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb`

Para instalar o SDK do DotNet, basta executar os seguintes comandos no terminal:

- `wget https://packages.microsoft.com/config/ubuntu/20.04/packages-microsoft-prod.deb -O packages-microsoft-prod.deb`
- `sudo dpkg -i packages-microsoft-prod.deb`
- `rm packages-microsoft-prod.deb`
- `sudo apt-get update && sudo apt-get install -y dotnet-sdk-6.0`

Depois de instalar o SDK, ainda no cmd, navegue até o diretório raiz do projeto e siga as seguintes etapas, executando os comandos listados:

- compile o projeto: `dotnet build`
- executando o projeto (modo interativo): `dotnet run`

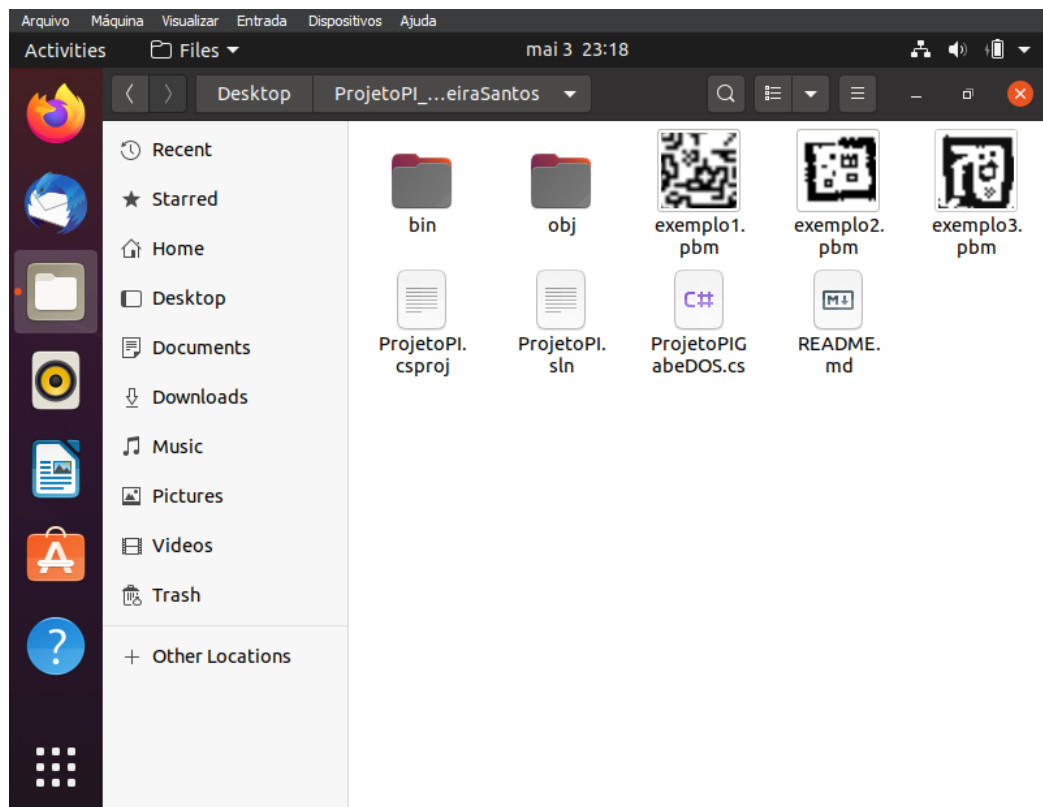
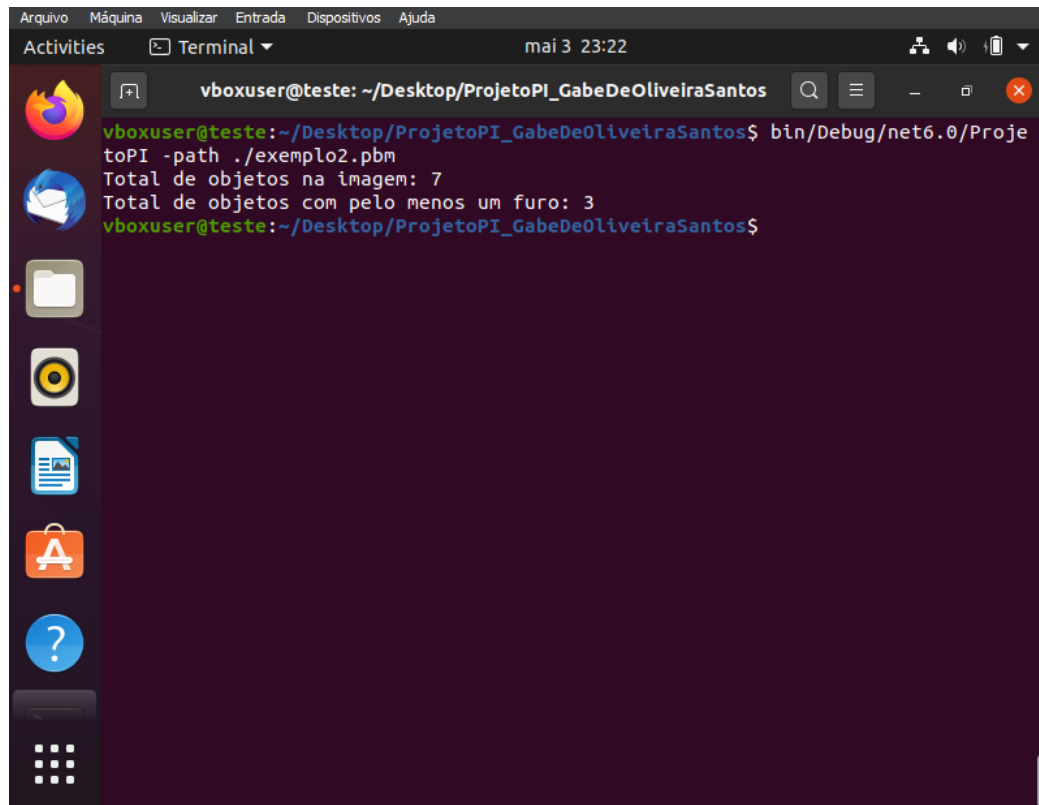


Figura 5 – Raiz do projeto

Também é possível executar o projeto via linha de comando e passar dois parâmetros de execução:

- para passar o diretório da imagem: `-path DIRETORIO_DA_IMAGEM`
- para ativar o modo debug: `-debugMode`
- exemplo de execução com os parâmetros (estando com o terminal aberto na raiz do projeto):
`bin/Debug/net6.0/ProjetoPI -debugMode -path ./exemplo1.pbm`



The image shows a terminal window on a Linux desktop. The window title is "vboxuser@teste: ~/Desktop/ProjetoPI_GabeDeOliveiraSantos". The terminal output shows the execution of a command to run a project in debug mode, passing a file path as an argument. The output displays the total number of objects in the image and the number of objects with at least one hole.

```
Arquivo  Máquina  Visualizar  Entrada  Dispositivos  Ajuda
Activities  Terminal  mai 3 23:22
vboxuser@teste: ~/Desktop/ProjetoPI_GabeDeOliveiraSantos
vboxuser@teste:~/Desktop/ProjetoPI_GabeDeOliveiraSantos$ bin/Debug/net6.0/ProjetoPI -path ./exemplo2.pbm
Total de objetos na imagem: 7
Total de objetos com pelo menos um furo: 3
vboxuser@teste:~/Desktop/ProjetoPI_GabeDeOliveiraSantos$
```

Figura 6 – Executando o projeto via linha de comando e passando a imagem exemplo2.pbm por parâmetro

Obs.: o modo debug além de printar o resultado de algumas operações no console, exporta esses resultados para o diretório de onde você executou o projeto. Sendo assim, caso o debugMode seja usado, é interessante acessar este diretório, conferir esses resultados e depois apagá-los para não provocar confusão caso o comando seja executado com outra imagem.

2

Segunda Parte

2.1 Que potencial de uso vocês imaginam para o algoritmo desenvolvido?

Ele tem potencial para ser usado como ferramenta de detecção de anomalias em organismos a partir de imagens digitais geradas por microscópios. Desta forma, seria possível não só detectar esses organismos como segmentar a imagem de acordo com os organismos encontrados e identificar anomalias (seriam representadas como furos) presentes nestes organismos.

Com isso, a depender da eficiência e precisão do algoritmo neste contexto, seria possível até embarcar esse algoritmo no software de microscópios mais modernos.

2.2 Que adaptações seriam necessárias para que sua ideia funcione?

Seria necessário aprimorar a leitura de imagens, tornando possível a leitura de imagens coloridas e, para que o algoritmo funcionasse, também seria necessário aplicar uma limiarização na imagem para que a mesma se tornasse binária.

Além disso, talvez haja a necessidade de realizar alguns ajustes/processamentos na imagem como ajustes no contraste e equalização. Outra adaptação possível, caso a metodologia atual não gerasse bons resultados, seria aplicar algum algoritmo de detecção de bordas e ir adaptando as demais etapas da solução.

2.3 Como sua ideia se encaixa no conceito de ética estudado pelo grupo?

Seguindo a ideia de transparência no uso dos algoritmos de processamento de imagem citada no trabalho anterior, a solução implementada neste trabalho apresenta a funcionalidade de debug (que pode e será aprimorada) que detalha e apresenta um log das etapas onde a imagem atual é processada e uma nova imagem é retornada afim de tornar o processo o mais transparente possível. Além disso, pensando na aplicação do algoritmo proposta no tópico 2.1, o algoritmo seria distribuído com um manual detalhando todo o processo referente ao processamento efetuado por ele.

Além do manual, seria distribuído também um termo afirmando o compromisso da empresa e da equipe responsável pelo desenvolvimento do algoritmo com a privacidade, transparência e direitos das imagens usadas no algoritmo e também um contrato que firmaria uma garantia de que o algoritmo seria usado apenas em soluções que respeitassem os valores da empresa e que fizessem um uso justo do mesmo.

3

Apêndice

3.1 Repositório do projeto

O código fonte do projeto pode ser acessado através deste link: [clique aqui!](#)