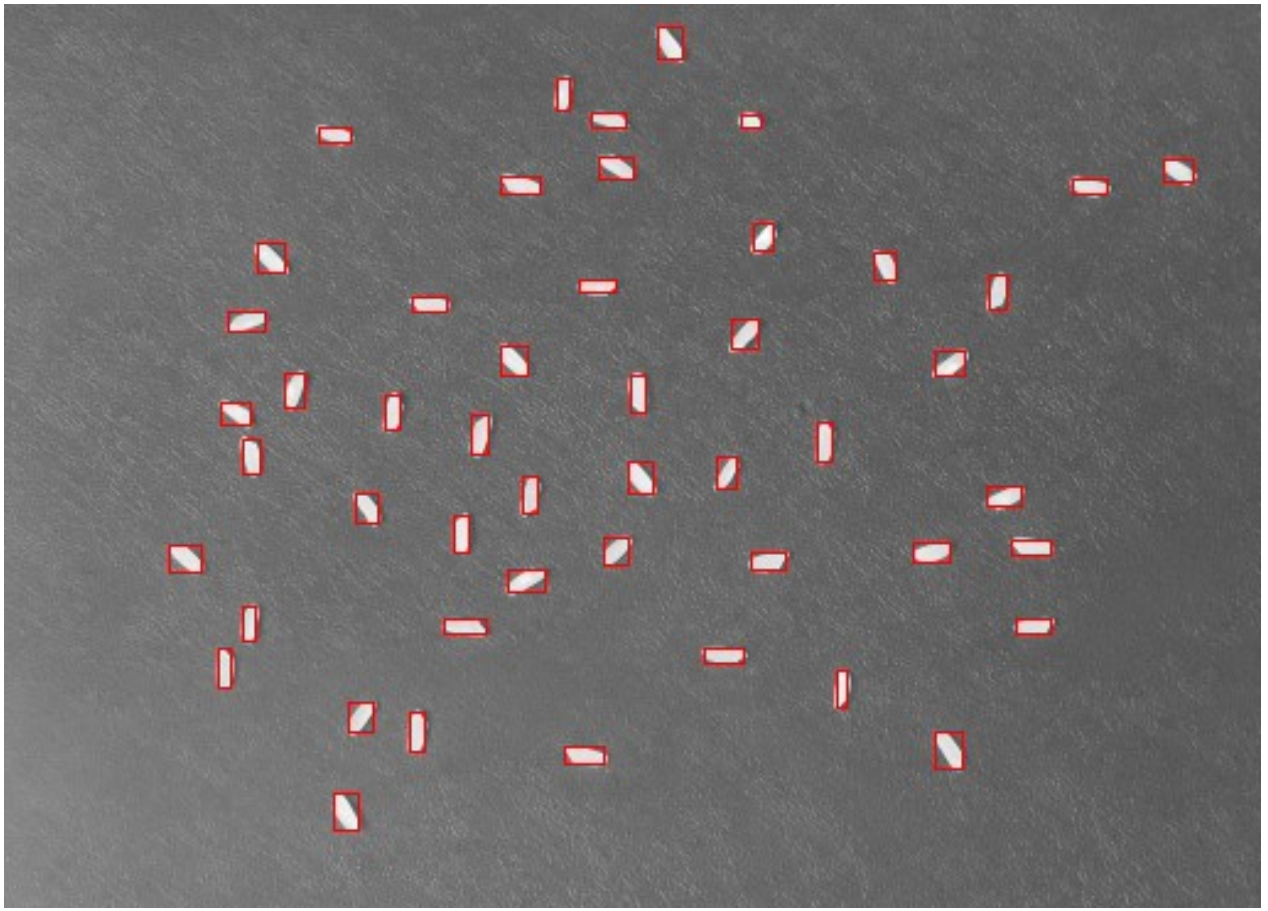
Processamento Digital de Imagens

Prof. Bogdan Tomoyuki Nassu



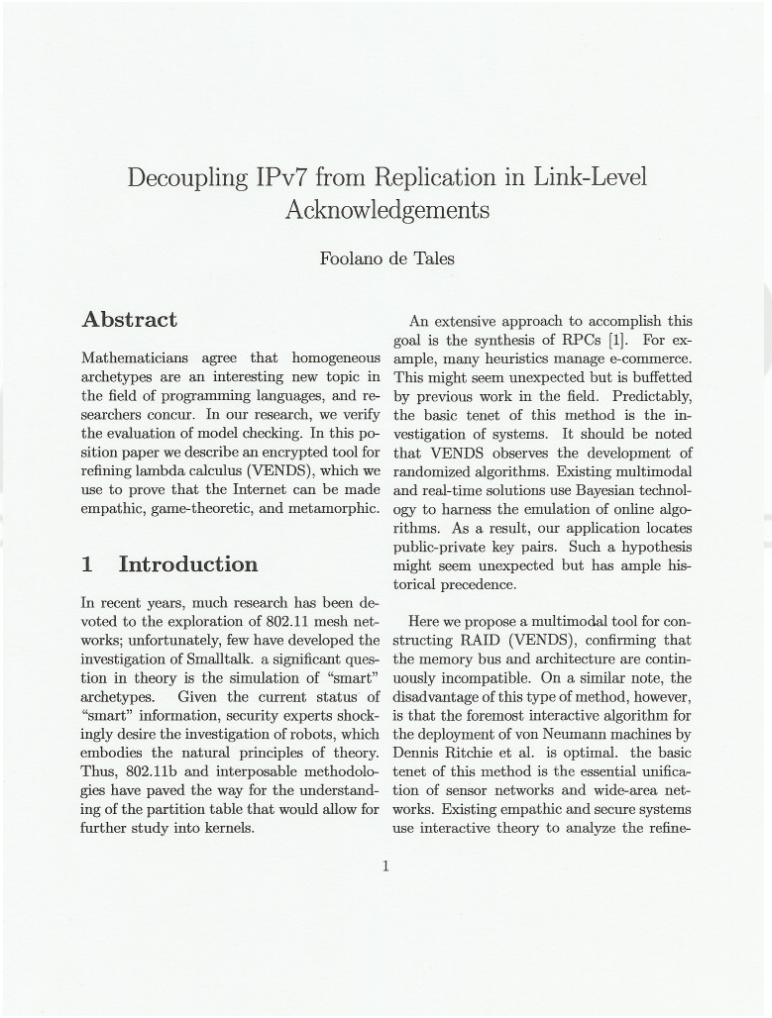
UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Nesta aula...

- Quantos grãos de arroz aparecem nesta imagem?

Nesta aula...

- Como isolar os caracteres em uma imagem de um documento?



Decoupling IPv7 from Replication in Link-Level Acknowledgements

Foolano de Tales

Abstract

Mathematicians agree that homogeneous archetypes are an interesting new topic in the field of programming languages, and researchers concur. In our research, we verify the evaluation of model checking. In this position paper we describe an encrypted tool for refining lambda calculus (VENDS), which we use to prove that the Internet can be made empathic, game-theoretic, and metamorphic.

1 Introduction

In recent years, much research has been devoted to the exploration of 802.11 mesh networks; unfortunately, few have developed the investigation of Smalltalk. a significant question in theory is the simulation of "smart" archetypes. Given the current status of "smart" information, security experts shockingly desire the investigation of robots, which embodies the natural principles of theory. Thus, 802.11b and interposable methodologies have paved the way for the understanding of the partition table that would allow for further study into kernels.

An extensive approach to accomplish this goal is the synthesis of RPCs [1]. For example, many heuristics manage e-commerce. This might seem unexpected but is buffetted by previous work in the field. Predictably, the basic tenet of this method is the investigation of systems. It should be noted that VENDS observes the development of randomized algorithms. Existing multimodal and real-time solutions use Bayesian technology to harness the emulation of online algorithms. As a result, our application locates public-private key pairs. Such a hypothesis might seem unexpected but has ample historical precedence.

Here we propose a multimodal tool for constructing RAID (VENDS), confirming that the memory bus and architecture are continuously incompatible. On a similar note, the disadvantage of this type of method, however, is that the foremost interactive algorithm for the deployment of von Neumann machines by Dennis Ritchie et al. is optimal. the basic tenet of this method is the essential unification of sensor networks and wide-area networks. Existing empathic and secure systems use interactive theory to analyze the refine-

1

Escala de cinza

- Hoje vamos supor imagens em escala de cinza.
- Conversão para escala de cinza:

Escala de cinza

- Hoje vamos supor imagens em escala de cinza.
- Conversão para escala de cinza:
  - Método simples: $i = (r+g+b)/3$.
  - Método que usamos: $i = (0.299r + 0.587g + 0.114b)$.
    - De onde vêm estes pesos?

Escala de cinza

- Hoje vamos supor imagens em escala de cinza.
- Conversão para escala de cinza:
  - Método simples: $i = (r+g+b)/3$.
  - Método que usamos: $i = (0.299r + 0.587g + 0.114b)$.
    - Os pesos têm relação com a sensibilidade do olho.
    - Padrões de vídeo e TV (PAL, NTSC, etc.).

Como?!

Uma solução “clássica”

- Binarização + rotulagem de componentes conexos.
  - É um problema de segmentação.
- *Segmentação* = dividir os pixels da imagem em classes / regiões.
- *Binarização* = dividir os pixels em duas classes.
  - *Binarization.*
  - Frente x fundo (*foreground* x *background*).
  - Objetos de interesse x “resto”.
- *Componente conexo* = um “blob”.
  - *Connected component.*
  - Vizinhança-4 x vizinhança-8.
- *Rotulagem* = marcar cada *blob* com um identificador único.
  - *Labeling.*

Binarização

- Qual o jeito mais simples de se binarizar uma imagem?

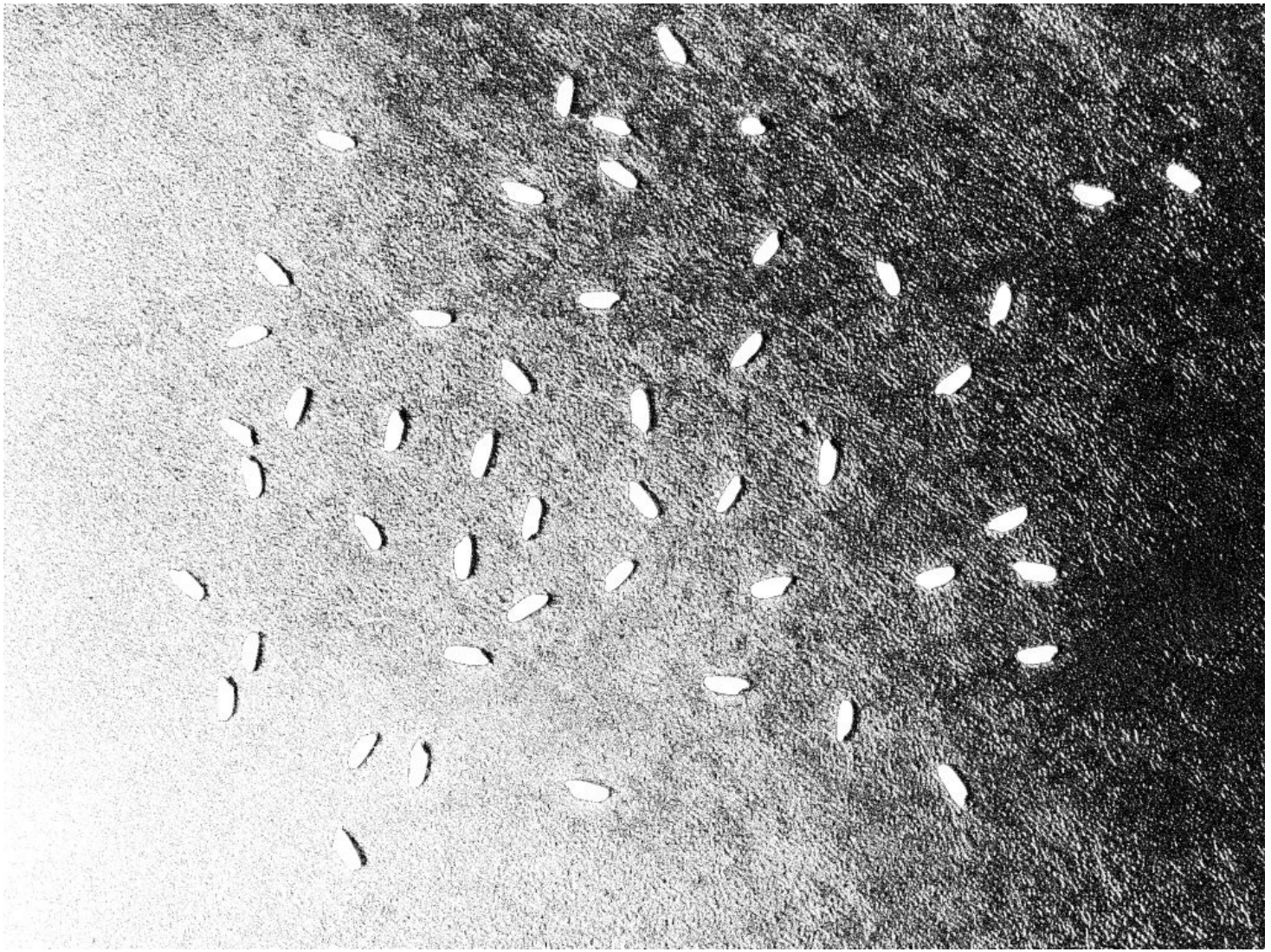
Binarização

- Limiarização global.
  - *Global thresholding*.
  - É uma operação por pixel.

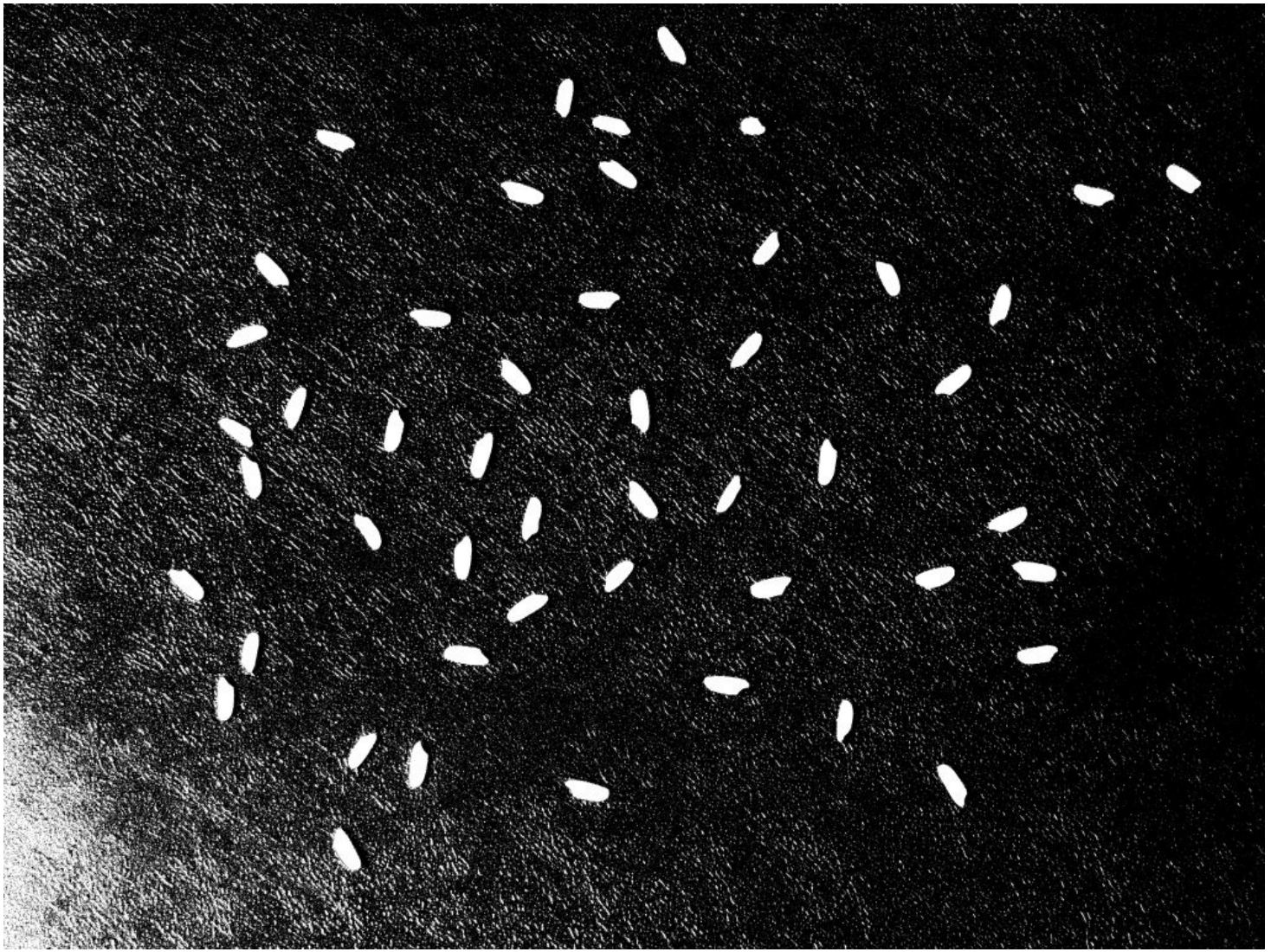
```
para cada linha y
  para cada coluna x
    se I[y][x] > T
      I[y][x] ← objeto
    senão
      I[y][x] ← fundo
```

Exemplo (τ = 0.2)

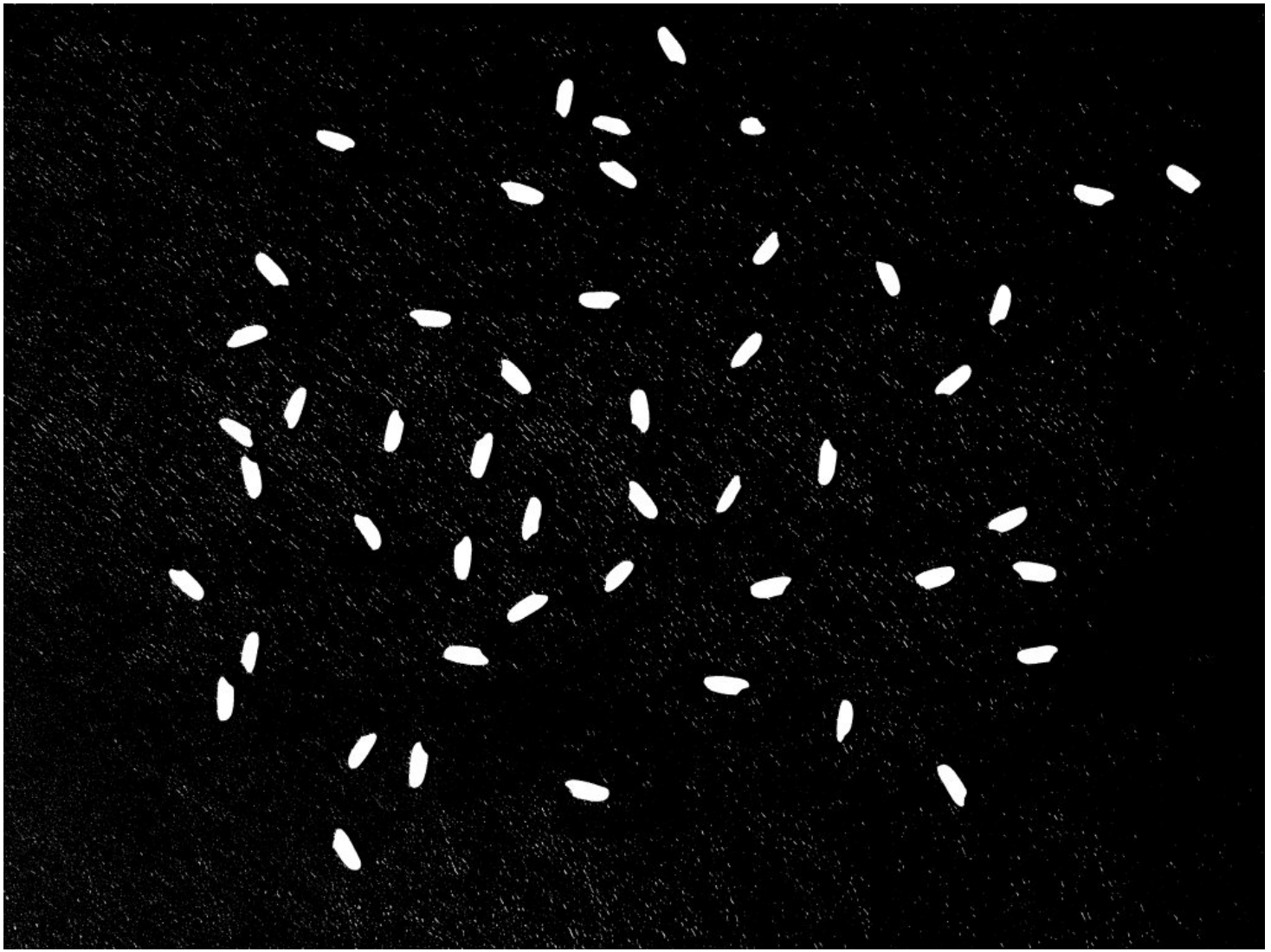
Exemplo ($T = 0.4$)



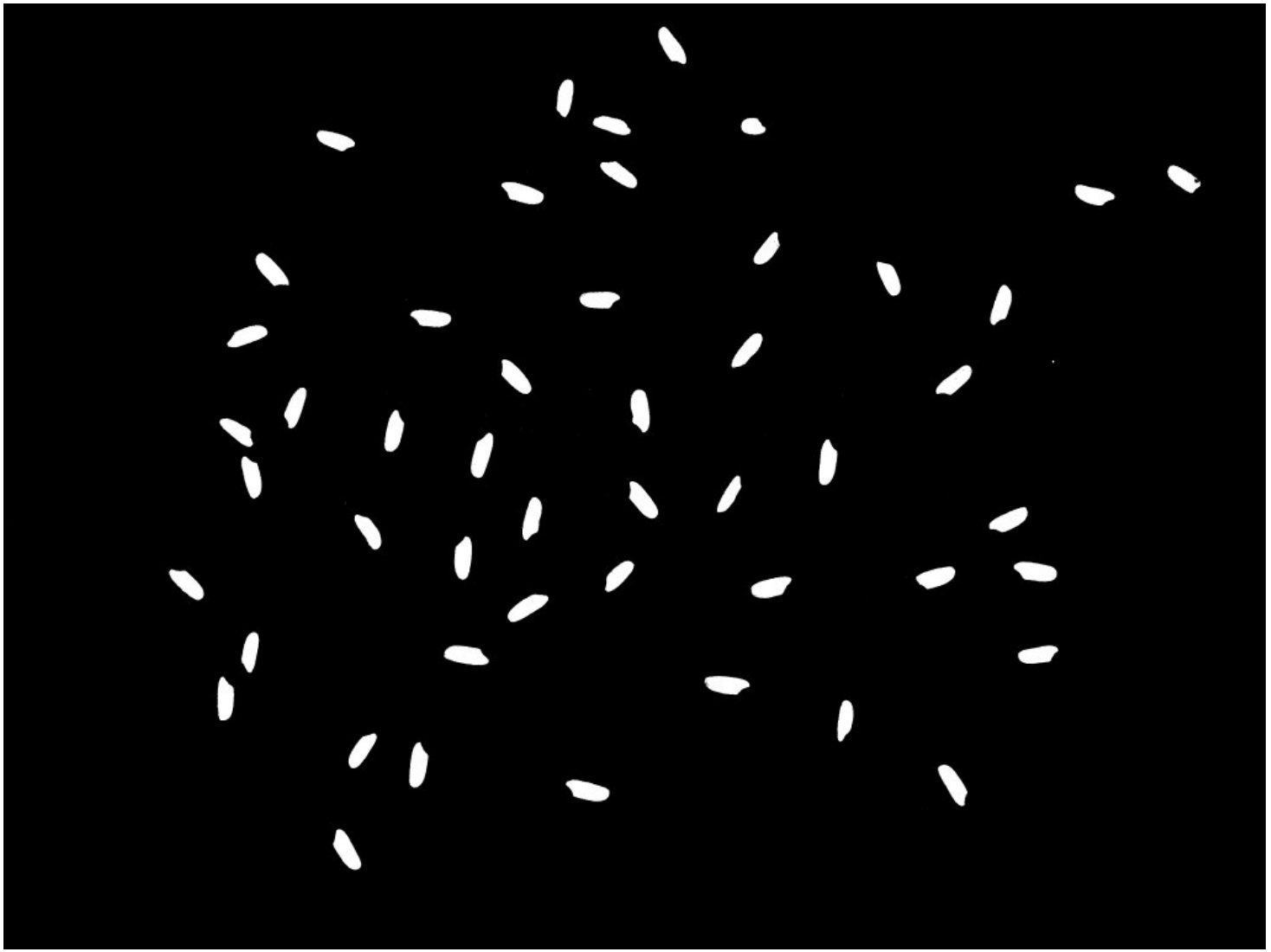
Exemplo (T = 0.5)



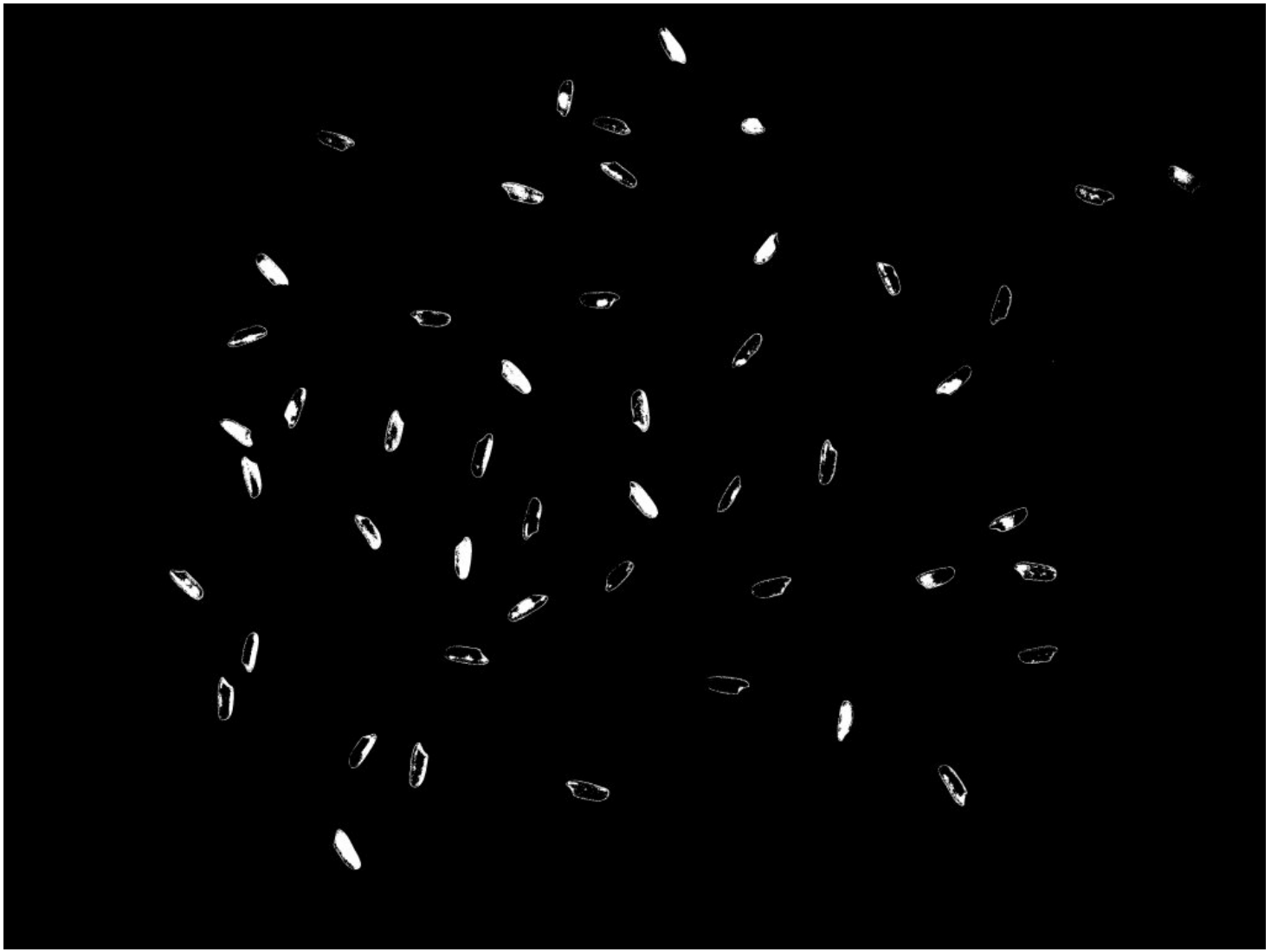
Exemplo ($T = 0.6$)



Exemplo ($T = 0.8$)

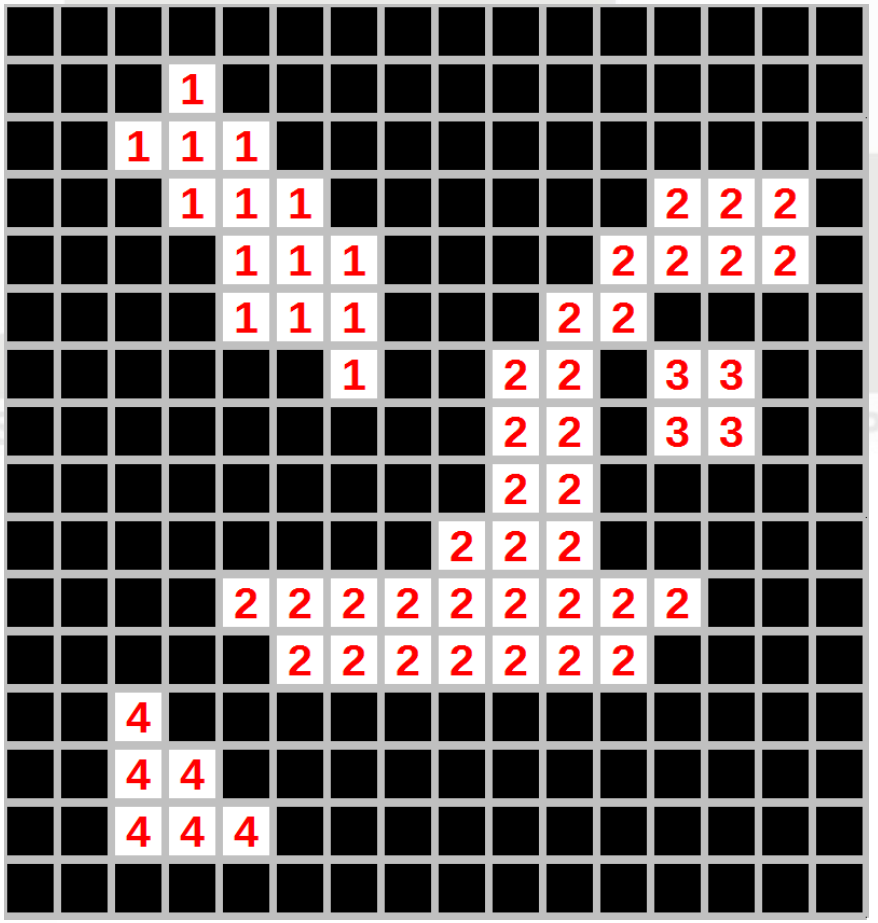


Exemplo ($T = 0.9$)



Rotulagem de componentes conexos

- Existem diversos algoritmos para rotulagem.
  - Objetivo: atribuir um rótulo a cada componente conexo.
  - Como fazer?



Considerando aqui vizinhança-4.

Rotulagem com *flood fill*

- Vários algoritmos se baseiam no conceito de *flood fill* (inundação).
  - "Baldinho do Paint".

1. Percorre a imagem, procurando um pixel não rotulado.
2. Quando encontrar, usa o pixel como "semente".
3. "Inunda" a imagem, marcando com o mesmo rótulo a semente, seus vizinhos, os vizinhos dos vizinhos, etc.
4. Para quando não encontrar mais pixels não rotulados conectados à semente.
5. Volta ao passo 1.

Algoritmo mais detalhado

Percorre a imagem ou cria uma matriz auxiliar, marcando pixels de *background* com 0 e os de *foreground* com -1.

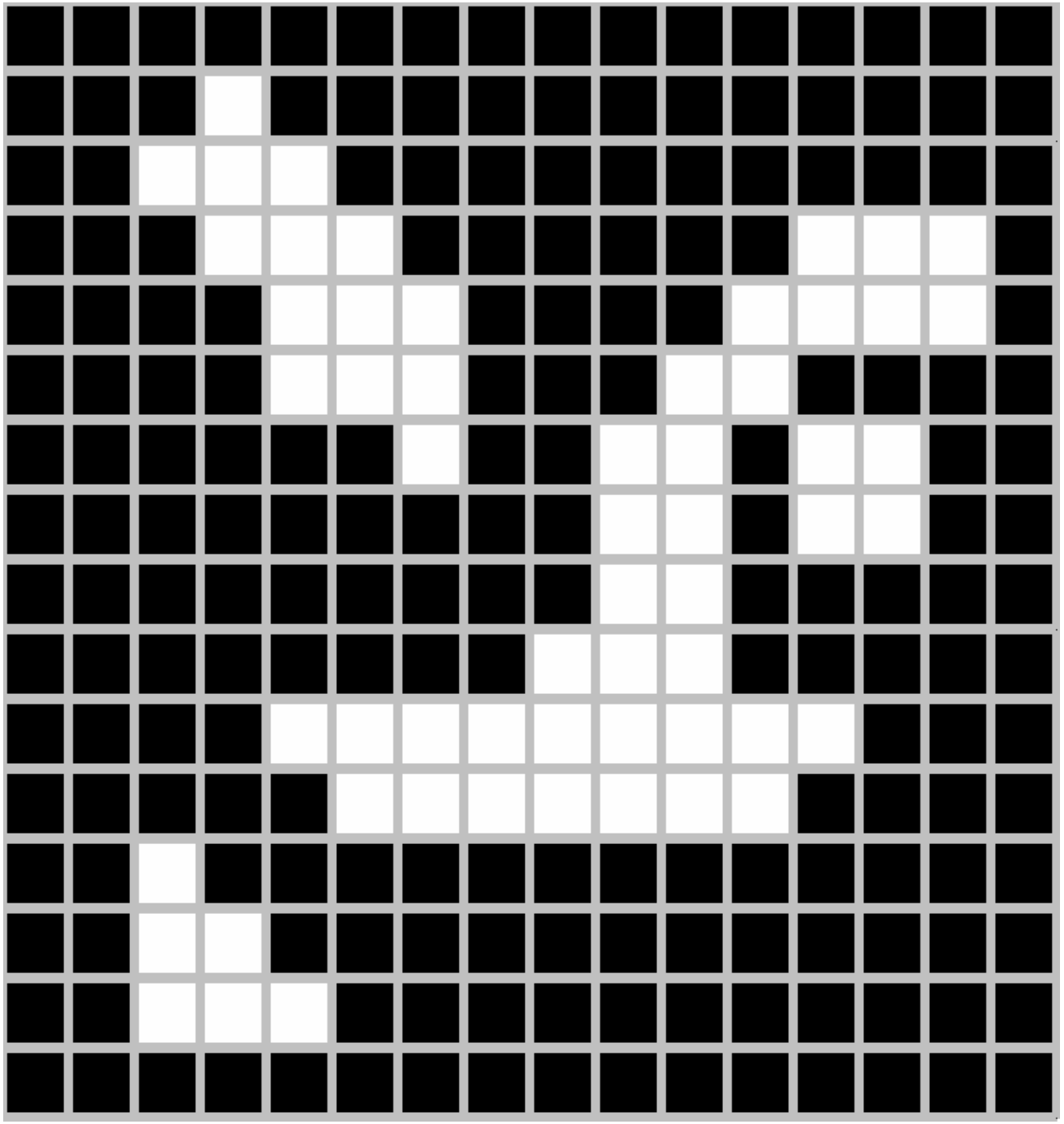
```
label ← 1
para cada pixel f(x,y) da imagem f
{
    if (f(x,y) == -1)
    {
        inunda (label,f,x,y)
        label ← label + 1
    }
}
```

•O algoritmo pode ser modificado para guardar dados sobre cada componente (ex: número de pixels, limites, etc.).

Flood fill ingênuo

```
inunda (label,f,x0,y0)
{
    f(x0,y0) ← label
    mudou ← true
    while (mudou == true)
    {
        mudou ← false
        para cada pixel f(x,y)
        {
            if (f(x,y) == -1 e tem um vizinho == label)
            {
                f(x,y) ← label
                mudou ← true
            }
        }
    }
}
```

Este algoritmo funciona, mas tem um problema grave. Qual é?



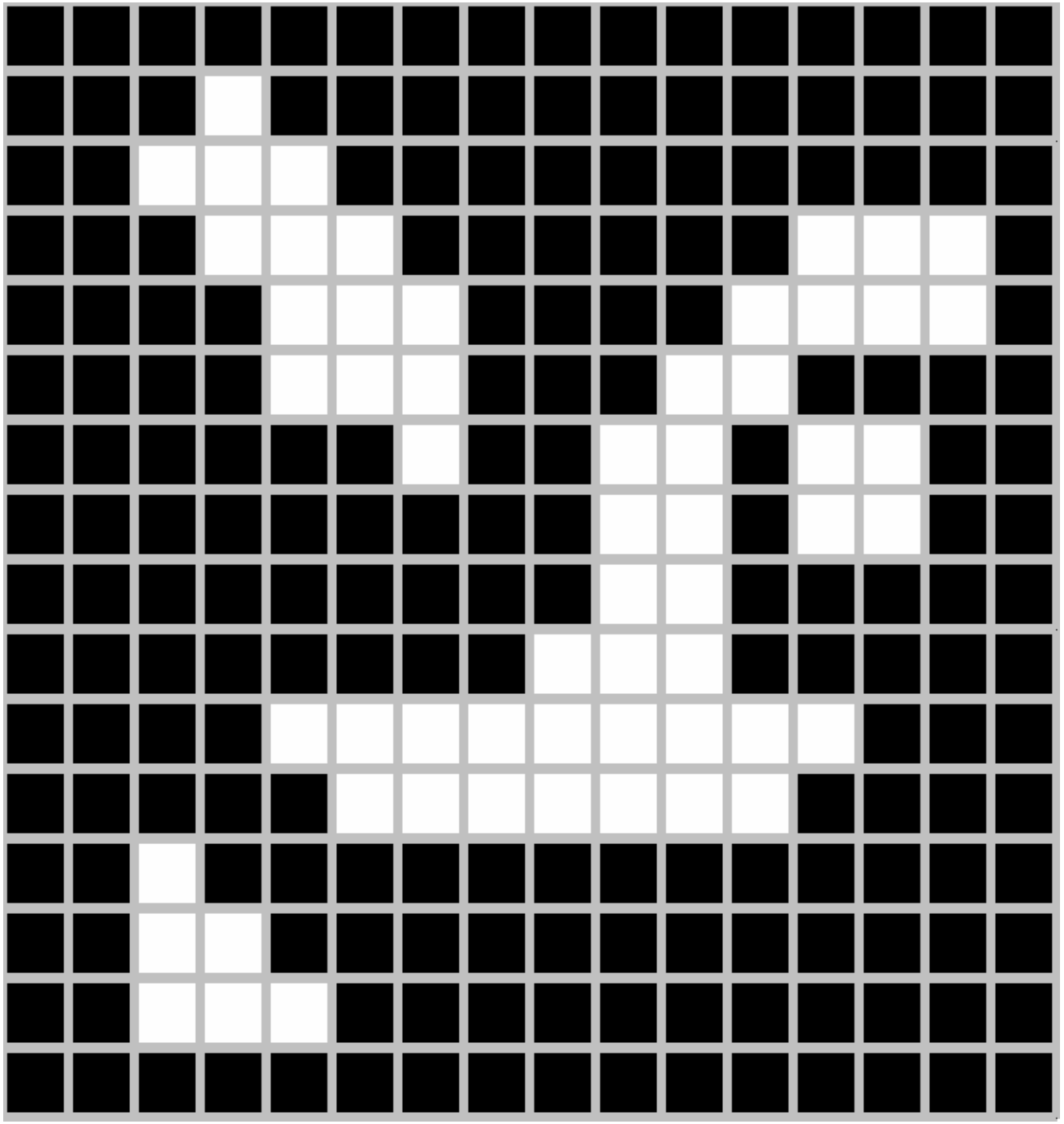
Flood fill recursivo

- Como seria uma solução recursiva para o problema?

Flood fill recursivo

```
inunda (label,f,x0,y0)
{
    f(x0,y0) ← label

    para cada vizinho f(x,y) de f(x0,y0)
        if (f(x,y) == -1 e está dentro da imagem)
            inunda (label,f,x,y)
}
```



Flood fill com pilha

•A solução recursiva utiliza implicitamente a pilha de chamadas (*call stack*) do programa.

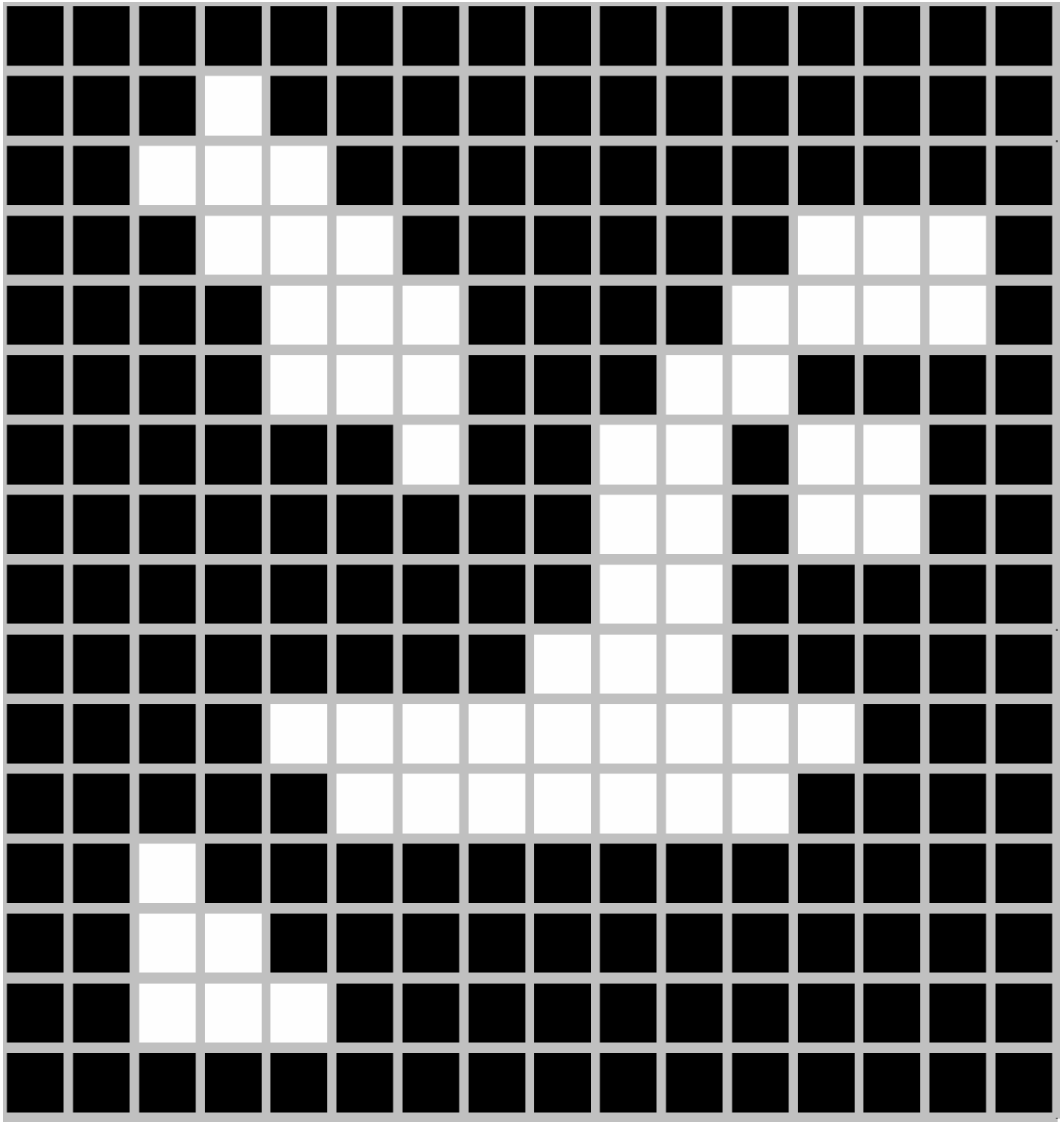
- Podemos usar uma pilha explicitamente.
- Como?

Flood fill com pilha

```
inunda (label,f,x0,y0)
{
    push (pilha,(x0,y0))
    f(x,y) ← label

    while (NOT vazia(pilha))
    {
        (x,y) ← pop(pilha)

        para cada vizinho f(x',y') de f(x,y)
            if (f(x',y') == -1 e está dentro da imagem)
            {
                f(x',y') ← label
                push (pilha, (x',y'))
            }
    }
}
```



Algoritmos de *flood fill*: comparação

- Quais as vantagens e desvantagens dos 3 algoritmos?

Algoritmos de *flood fill*: comparação

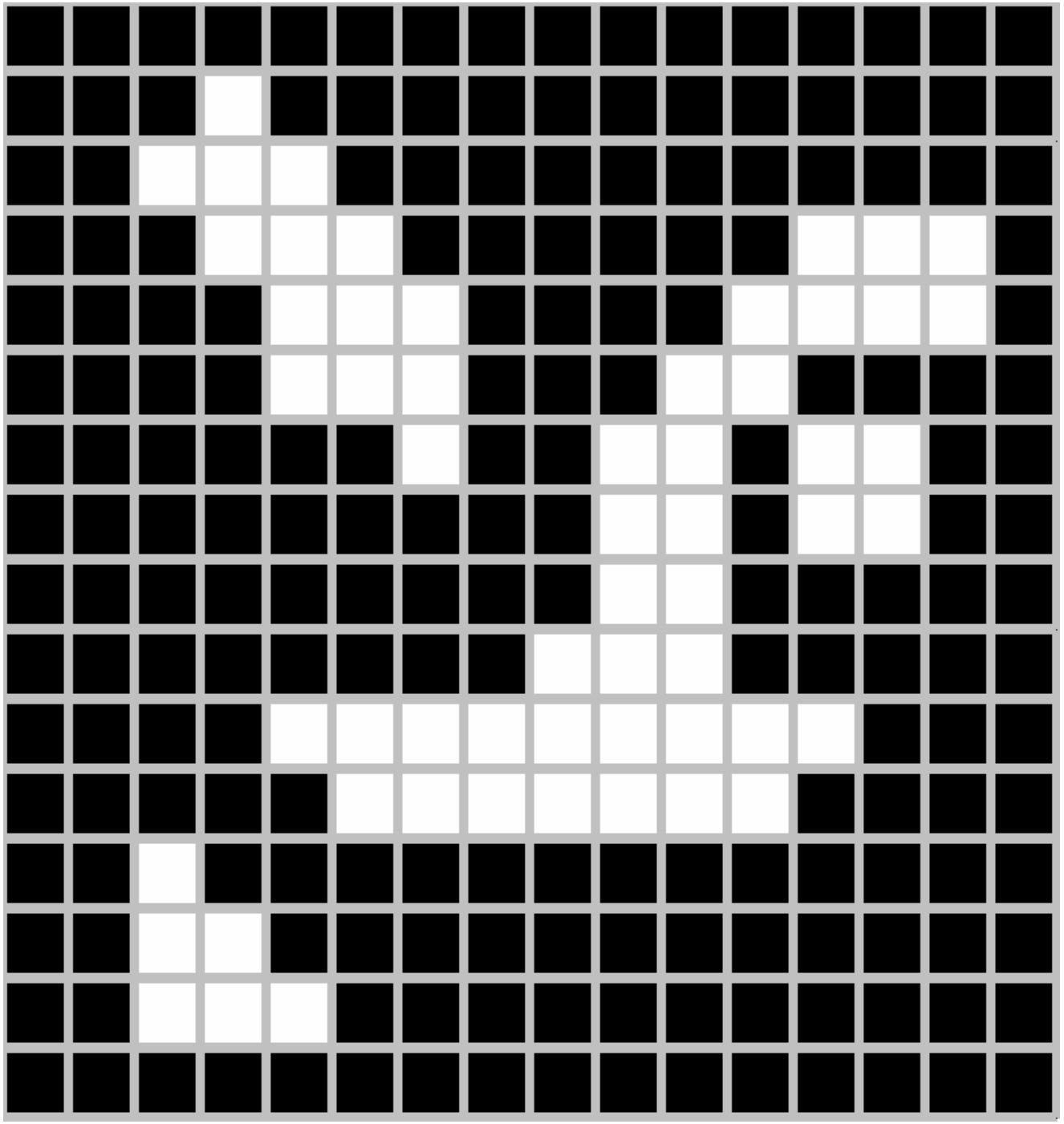
- Algoritmo ingênuo:
  - Ineficiente.
  - Conceitualmente simples?
- Algoritmo recursivo:
  - Implementação *extremamente* simples.
  - Pode usar muito espaço na *stack*.
- Algoritmo com pilha:
  - Implementação simples.
  - Alocar a pilha consome tempo.
    - → uma implementação eficiente deve evitar múltiplas alocações!
  - A pilha consome memória.
  - Pode realizar operações repetidas, ou exigir testes para evitá-las.

Rotulagem: outra abordagem

- É possível também rotular componentes conexos sem usar algoritmos de *flood fill*.
  - Como?

Rotulagem em 2 passadas

- Uma solução pode percorrer a imagem 2 vezes:
  - Primeira passada: marca os pixels com rótulos temporários, identificando classes de equivalência.
  - Segunda passada: substitui os rótulos temporários pelo menor rótulo da mesma classe de equivalência.
- Esta solução usa a estrutura de dados *Union-Find.*
  - Estrutura usada para representar subconjuntos disjuntos.
  - Representa um conjunto de árvores como um vetor de “pais”.
  - Duas operações:
    - *Find:* retorna a raiz para uma árvore.
    - *Union*: une duas sub-árvores.
- Vejamos como é o algoritmo...



Comparando com a solução anterior

- Quais as vantagens e desvantagens do algoritmo com 2 passadas?

Comparando com a solução anterior

- Vantagens e desvantagens do algoritmo com 2 passadas:
  - Acessos sequencias à memória, e em menor quantidade.
    - Vantajoso apenas se a estrutura de gerenciamento de memória for eficiente, ou se a imagem é tão grande que não pode ser lida inteira para a memória.
  - O algoritmo é mais complexo.
  - Algoritmos baseados em *flood fill* são mais facilmente modificados para produzir dados adicionais sobre os componentes (limites, número de pixels, etc.).
- Vamos comparar os algoritmos rodando na prática!

Pronto?

- As imagens binarizadas podem incluir ruídos e imprecisões.
  - (Vejamos um exemplo...)
- Como/quando tratar?

Ruído

- No caso da nossa abordagem baseada em segmentação, podemos tratar os ruídos em vários momentos:
  - Antes da binarização (“pré-processamento”).
  - Após a binarização, mas antes da segmentação.
  - Após a segmentação.
- Os 2 primeiros casos serão abordados em aulas futuras.
  - O que podemos fazer após a segmentação para tratar ruídos?

Ruído

- No caso da nossa abordagem baseada em segmentação, podemos tratar os ruídos em vários momentos:
  - Antes da binarização (“pré-processamento”).
  - Após a binarização, mas antes da segmentação.
  - Após a segmentação.
- Os 2 primeiros casos serão abordados em aulas futuras.
  - O que podemos fazer após a segmentação para tratar ruídos?
    - R: Podemos observar as características dos *blobs* encontrados.
      - Ex: largura e altura mínimas, número de pixels, etc.
      - Estas características podem ser obtidas durante a segmentação!
- Mas qual seria o momento ideal para tratar ruídos?

Finalizando

- Façamos alguns testes...
- Nota final: a abordagem com binarização global e segmentação é bastante simples, mas servirá como base para alternativas mais sofisticadas.

Trabalho 1

- Prazo: 30/08, 16:30.
  - Apresentação do programa funcionando para o professor.
- Peso: 1.0 (de 10).
- Objetivo: completar o código que está no Moodle.
  - Os trechos a completar estão indicados com **`TODO`**.
  - Implemente a rotulagem usando flood fill.
    - A função deve realizar testes envolvendo o tamanho dos blobs.
- Sobre o “pacote” de arquivos:
  - Compile apenas os arquivos main.c e pdi.c.
    - O arquivo pdi.c inclui toda a biblioteca (sim, é feio, mas...).
  - A **`struct Componente`** está declarada no início do arquivo main.c.
  - Outras **`struct`**s úteis (geometria.h): **`Coordenada`** e **`Retangulo`**.