НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
Факультет прикладної математики
Кафедра прикладної математики

Звіт
із лабораторної роботи №6
із дисципліни «Розподілені і хмарні обчислення»

Виконав:
студент групи КМ-03
Шаповалов Г. Г.

Керівник:
Ліскін В. О.

Київ — 2023

Мета роботи

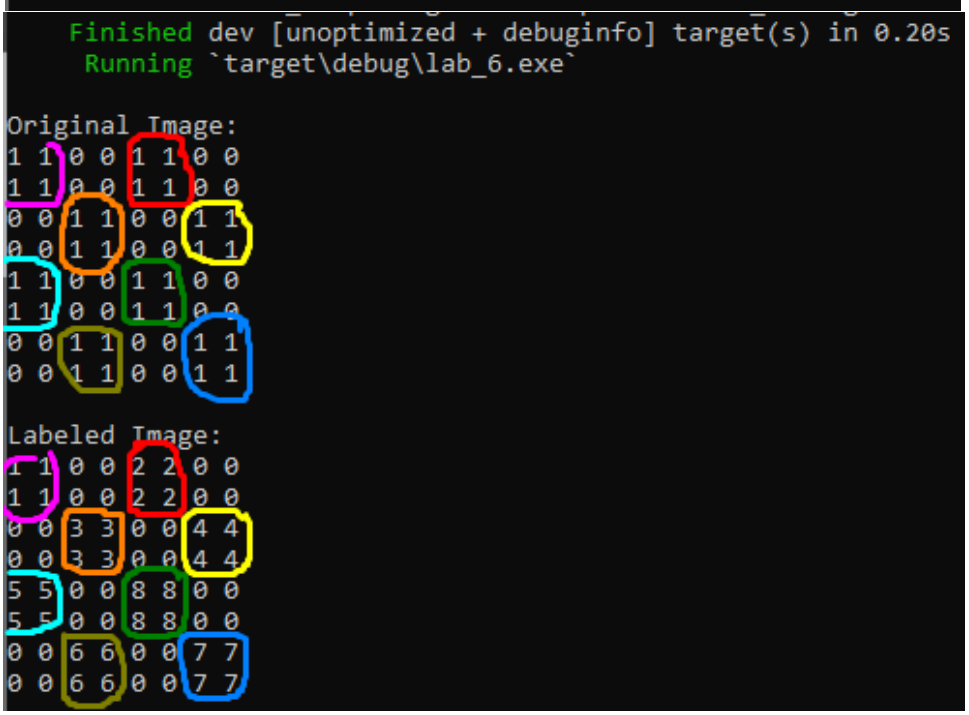
Завдання виділення областей зображення. Уявити зображення матрицею чисел, що визначають колір пікселя. Визначити кількість областей, складових зображення, використовуючи алгоритм пульсації для міжпроцесорної взаємодії. Два пікселя належать одній області, якщо вони є сусідами по горизонталі або вертикалі. Для вирішення задачі можна використовувати матрицю міток областей, призначаючи мітці максимальне значення серед сусідів.

Опис програми

```
    Finished dev [unoptimized + debuginfo] target(s) in 0.20s
     Running `target\debug\lab_6.exe`

Original Image:
1 1 0 0 1 1 0 0
1 1 0 0 1 1 0 0
0 0 1 1 0 0 1 1
0 0 1 1 0 0 1 1
1 1 0 0 1 1 0 0
1 1 0 0 1 1 0 0
0 0 1 1 0 0 1 1
0 0 1 1 0 0 1 1

Labeled Image:
1 1 0 0 2 2 0 0
1 1 0 0 2 2 0 0
0 0 3 3 0 0 4 4
0 0 3 3 0 0 4 4
5 5 0 0 8 8 0 0
5 5 0 0 8 8 0 0
0 0 6 6 0 0 7 7
0 0 6 6 0 0 7 7
```



Висновки

Програма впоралась із задачею і виділила області, які позначно ‘кольоровими’ (1) в початковій матриці. Решту лишила 0-ми, що каже, що це не замальована область. В якості сусідів брались лише значення по вертикалі та горизонталі. Діагональний елемент програмою не сприймається як ‘сусід’.

Лістинг програми

```
extern crate nalgebra as na;
extern crate rayon;

use std::sync::{Mutex};

use na::DMatrix;
use rayon::prelude::*;

fn get_neighbors(image: &DMatrix<u32>) -> DMatrix<u32> {
    let (rows, cols) = image.shape();
    let mut labels = DMatrix::from_element(rows, cols, 0u32);
    let _labels = Mutex::new(&mut labels);
    let next_label = Mutex::new(1u32);

    (0..rows).into_par_iter().for_each(|i| {
        (0..cols).into_par_iter().for_each(|j| {
            if image[(i, j)] != 0 {
                let mut neighbor_labels = Vec::new();

                let mut _labels = _labels.lock().unwrap();

                if i > 0 && _labels[(i - 1, j)] != 0 && image[(i - 1, j)] ==
image[(i, j)] {
                    neighbor_labels.push(_labels[(i - 1, j)]);
                }
                if i < rows - 1 && _labels[(i + 1, j)] != 0 && image[(i + 1, j)]
== image[(i, j)] {
                    neighbor_labels.push(_labels[(i + 1, j)]);
                }
                if j > 0 && _labels[(i, j - 1)] != 0 && image[(i, j - 1)] ==
image[(i, j)] {
                    neighbor_labels.push(_labels[(i, j - 1)]);
                }
                if j < cols - 1 && _labels[(i, j + 1)] != 0 && image[(i, j + 1)]
== image[(i, j)] {
                    neighbor_labels.push(_labels[(i, j + 1)]);
                }

                match neighbor_labels.iter().cloned().min() {
```

```
                    Some(min_label) => {
                        _labels[(i, j)] = min_label;
                        for &label in &neighbor_labels {
                            if label != min_label {
                                relabel_regions(&mut _labels, label, min_label);
                            }
                        }
                    }
                    None => {
                        let mut _next_label = next_label.lock().unwrap();
                        _labels[(i, j)] = *_next_label;
                        *_next_label += 1;
                    }
                }
            }
        });
    });

    labels
}

fn relabel_regions(labels: &mut DMatrix<u32>, old_label: u32, new_label: u32) {
    labels
        .iter_mut()
        .for_each(|label| {
            if old_label == *label {
                *label = new_label
            };
        });
}

fn print_matrix(matrix: &DMatrix<u32>) {
    for i in 0..matrix.nrows() {
        for j in 0..matrix.ncols() {
            print!("{} ", matrix[(i, j)]);
        }
        println!();
    }
}

fn main() {
    let image = vec![
        vec![1, 1, 0, 0, 1, 1, 0, 0],
        vec![1, 1, 0, 0, 1, 1, 0, 0],
        vec![0, 0, 1, 1, 0, 0, 1, 1],
        vec![0, 0, 1, 1, 0, 0, 1, 1],
        vec![1, 1, 0, 0, 1, 1, 0, 0],
        vec![1, 1, 0, 0, 1, 1, 0, 0],
        vec![0, 0, 1, 1, 0, 0, 1, 1],
        vec![0, 0, 1, 1, 0, 0, 1, 1],
    ];
```

```
    let data = DMatrix::from_vec(image.len(), image[0].len(),
image.into_iter().flatten().collect());

    let labeled_image = get_neighbors(&data);

    println!("\nOriginal Image:");
    print_matrix(&data);
    println!("\nLabeled Image:");
    print_matrix(&labeled_image);
}
```