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-01

Романецький М.С.

Керівник:

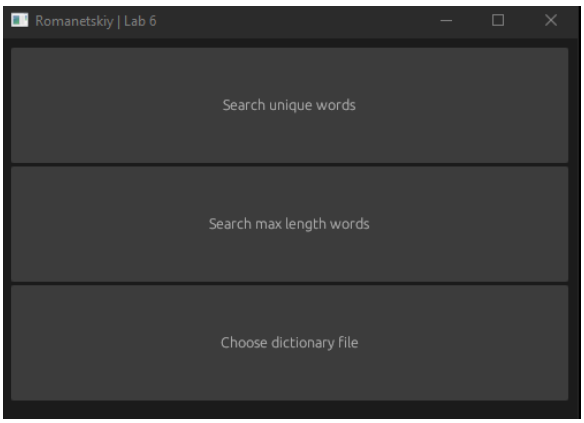
Доцент кафедри ПМА
Лiскін В. О.

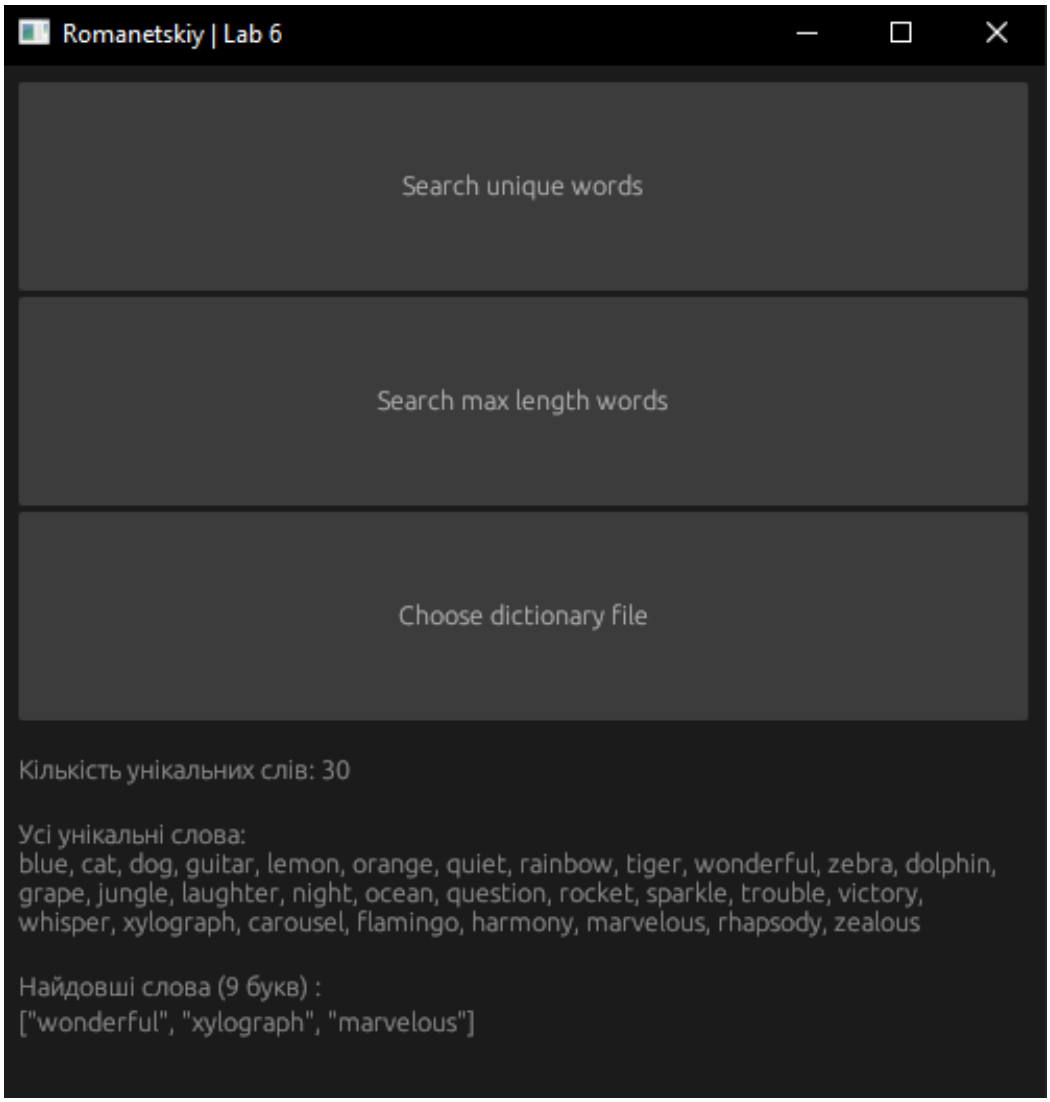
Київ — 2023

Мета роботи: Розпаралелити завдання підрахунку кількості унікальних слів в словнику (жодна буква не повторюється)
Опис програми: Бібліотека ‘Rayon’ буде використовуватись для паралелізму.
Приклад розв’язку завдання:
Словник — "cat" , "dog" , "guitar" , "happy"
унікальні слова "cat" , "dog" , "guitar"

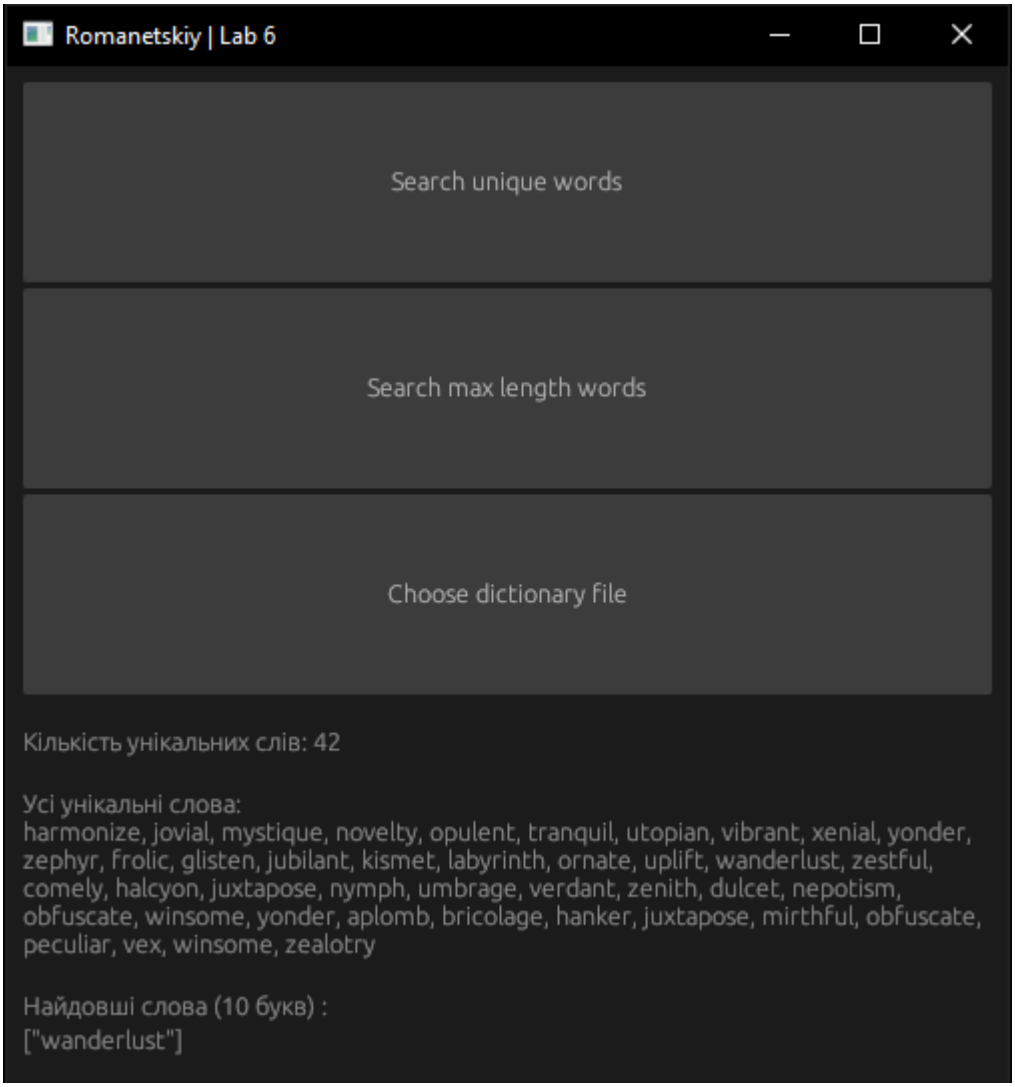
Програма завантажується і має такий інтерфейс.
Інтерфейс має 3 кнопки:

1. Пошук унікальних слів
2. Пошук довжини найдовших унікальних слів
3. Зміна словника





Коли словник базовий



Коли змінили на інший словник

Код програми:

```
// cd D:/KPI/Distributed_computing/Labs/lab_6


use rayon::prelude::*;
use std::collections::HashSet;
use std::fs::File;
use std::io::{self, BufRead};
use egui::vec2;
use eframe::{epi, egui::{self, CtxRef}};
use rfd::FileDialog;


struct MyApp {
    words: Vec<String>,
    unique_words: Vec<String>,
    longest_words: Vec<String>,
    dictionary_path: Option<String>,
}
```

```rust
impl Default for MyApp {
    fn default() -> Self {
        Self {
            words: Vec::new(),
            unique_words: Vec::new(),
            longest_words: Vec::new(),
            dictionary_path: Some("words.txt".to_string()), // Ініціалізація як
Option<String>
        }
    }
}


impl epi::App for MyApp {
    fn name(&self) -> &str {
        "Romanetskiy | Lab 6"
    }

    fn update(&mut self, ctx: &CtxRef, _frame: &mut epi::Frame) {
        egui::CentralPanel::default().show(ctx, |ui| {
            let button_size = vec2(ui.available_width(), 100.0);
            if ui.add_sized(button_size, egui::Button::new("Search unique
words")).clicked() {
                self.search_unique_words();
            }
            if ui.add_sized(button_size, egui::Button::new("Search max length
words")).clicked() {
                self.search_max_length_words();
            }
            if ui.add_sized(button_size, egui::Button::new("Choose dictionary
file")).clicked() {
                if let Some(path) = FileDialog::new().pick_file() {
                    self.dictionary_path =
Some(path.to_string_lossy().into_owned()); // Виправлене присвоєння
                    self.read_words_from_file().expect("Failed to read words from
file");
                }
            }
            if !self.unique_words.is_empty() {
                ui.label(format!("\nКількість унікальних слів: {}",
self.unique_words.len()));
                ui.label(format!("\nУсі унікальні слова: \n{}",
self.unique_words.join(", ")));
            }
            if !self.longest_words.is_empty() {
                ui.label(format!("\nНайдовші слова ({} букв) :",
self.longest_words[0].len()));
                ui.label(format!("{:?}", self.longest_words));
            }
```

```
        });
    }
}


impl MyApp {
    fn read_words_from_file(&mut self) -> io::Result<()> {
        if let Some(ref path) = self.dictionary_path {
            let file = File::open(path)?; // Використовуємо `path` як &str
            let reader = io::BufReader::new(file);

            self.words.clear(); // Очищення попередніх слів
            for line in reader.lines() {
                let word = line?;
                self.words.push(word);
            }
        }
        Ok(())
    }

    fn search_unique_words(&mut self) {
        self.unique_words = self.words.par_iter()
            .filter_map(|word| {
                let mut chars = HashSet::new();
                if word.chars().all(|c| chars.insert(c)) {
                    Some(word.clone())
                } else {
                    None
                }
            })
            .collect();
    }

    fn search_max_length_words(&mut self) {
        let max_length = self.unique_words.par_iter().map(|word|
word.len()).max().unwrap_or(0);
        self.longest_words = self.unique_words.par_iter()
            .filter(|word| word.len() == max_length)
            .cloned()
            .collect();
    }
}

fn main() {
    let mut app = MyApp::default();
    app.read_words_from_file().expect("Failed to read words from file");

    let mut native_options = eframe::NativeOptions::default();
    native_options.initial_window_size = Some(egui::vec2(500.0, 600.0));
    eframe::run_native(Box::new(app), native_options);
}
```