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: | Керівник: |
| студент групи КМ-01 | *Доцент кафедри ПМА*  *Ліскін В. О.* |
| *Романецький М.С.* |  |

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:

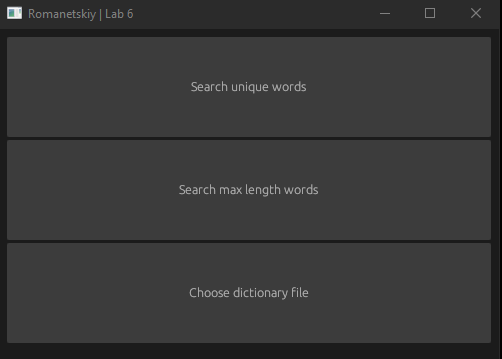
Словник — "cat" , "dog" , "guitar" , "happy"

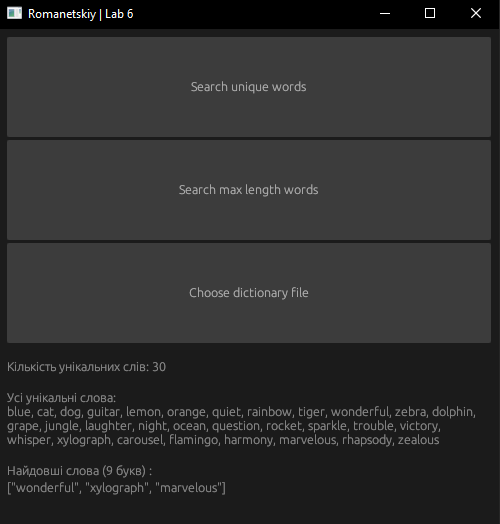
унікальні слова "cat" , "dog" , "guitar"

Програма завантажується і має такий інтерфейс.

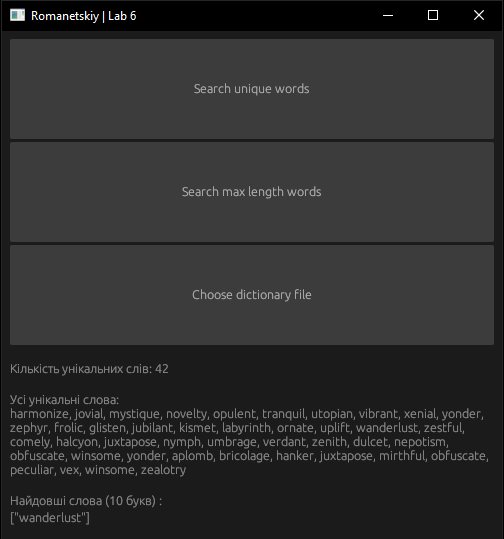
Інтерфейс має 3 кнопки:

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>,

}

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);

}