Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ КОМП'ЮТЕРНИХ НАУК ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Катедра «Комп'ютерна інженерія та програмування»

ЗВІТ

про виконання лабораторної роботи №13

з навчальної дисципліни «Алгоритми та структури даних»

Варіант 5

Виконав студент:

Омельніцький Андрій Миколайович
Група: КН-1023б

Перевірив:

Старший викладач

Бульба Сергій Сергійович

Харків - 2024

Зміст

1 Мета роботи

Набути досвіду практичної роботи з бінарними деревами.
Теми для попередньої роботи:

- нелінійні списки;
- графи;
- дерева;
- операції на деревах.

2 Завдання

Розробити програму, що дозволяє створити бінарне дерево та вирішити індивідуальне завдання. У завданнях 7 – 16 включно (де не вказані обходи) реалізувати два алгоритми обходу, які обрати самостійно.

Розробити програму турнірного сортування.

3 Хід виконання

Для виконання завдання було обрано мову Rust. Увесь код також додатково був розміщений в GitHub репозитарії: https://github.com/blackgolyb/algos-labs.

3.1 Турнірне сортування

```rust
use std::fmt::Debug;

use super::super::sort_preamble::*;

#[derive(Debug)]
struct Node<T> {
    value: Option<*const T>,
    index: Option<usize>,
}

impl<T> Copy for Node<T> {}
impl<T> Clone for Node<T> {
    fn clone(&self) -> Self {
        Node::new(self.value, self.index)
    }
}

impl<T> Node<T> {
    fn new(value: Option<*const T>, index: Option<usize>) -> Self {
        Self { value, index }
    }

    fn clear(&mut self) {
        self.index = None;
        self.value = None;
    }
}

impl<T> PartialEq for Node<T> {
    fn eq(&self, other: &Self) -> bool {
        self.index == other.index
    }
}
impl<T> Eq for Node<T> {}

impl<T> Node<T>
where
    T: Ord,
{
    fn winner(left: &Node<T>, right: &Node<T>) -> Node<T> {
        match (left.value, right.value) {
            (Some(a), Some(b)) => {
                let is_le = unsafe {
                    let a = &*a as &T;
                    let b = &*b as &T;
                    a.le(b)
                };
                if is_le {
                    left.clone()
                } else {
                    right.clone()
                }
            }
            (Some(_), None) => left.clone(),
            (None, Some(_)) => right.clone(),
            (None, None) => Node::new(None, None),
        }
    }
}

struct TournamentTree<'a, T> {
    nodes: Vec<Node<T>>,
    stack: Vec<usize>,
    logger: &'a mut Logger,
}

impl<'a, T> TournamentTree<'a, T>
where
    T: Ord,
{
    pub fn new(logger: &'a mut Logger) -> Self {
        Self {
```

```
            nodes: Vec::new(),
            stack: Vec::new(),
            logger,
        }
    }

    pub fn fill(&mut self, values: &mut Vec<T>) {
        let n = values.len();
        let tree_len = 2 * n - 1;
        let tree_height = (tree_len as f64).log2().ceil() as usize;

        self.nodes.reserve(tree_len);
        self.nodes.clear();
        self.stack.reserve(tree_height);
        self.stack.clear();

        unsafe {
            self.nodes.set_len(tree_len);
        }

        // Заповнюємо листя дерева
        for i in 0..n {
            self.nodes[n - 1 + i] = Node::new(Some(&values[i]), Some(i));
            self.logger.log_swap();
        }

        self.build_tree(n);
    }

    fn build_tree(&mut self, n: usize) {
        for i in (0..n - 1).rev() {
            self.nodes[i] = Node::winner(&self.nodes[2 * i + 1], &self.nodes[2 * i + 2]);
            self.logger.log_swap();
        }
    }

    pub fn winner(&self) -> Option<usize> {
        self.nodes[0].index
    }

    pub fn next_winner(&mut self) {
        self.stack.clear();
        let mut i = 0;
        let n = self.nodes.len();
        let root = self.nodes[0];

        // Видаляємо померднього переможця
        loop {
            self.stack.push(i);
            self.nodes[i].clear();

            let l = i * 2 + 1;
            let r = i * 2 + 2;

            self.logger.log_compare();
            if l < n && root == self.nodes[l] {
                i = l;
            } else if r < n && root == self.nodes[r] {
                i = r;
                self.logger.log_compare();
            } else {
                break;
            }
        }

        // Знахдимо нового переможця
        self.stack.pop();
        for a in (0..self.stack.len()).rev() {
            let i = self.stack[a];
            self.nodes[i] = Node::winner(&self.nodes[2 * i + 1], &self.nodes[2 * i + 2]);
            self.logger.log_swap();
        }
    }
}

sort! {
```

```
    TournamentSort + Debug | args:  SortArgs<T> | {
        let  arr  = args.0;
        let  sort  = args.1;

        let  n = arr.len();
        if  n <= 1 {
            return;
        }

        //  Створюємо дерево для турнірного сортування
        let  mut tree  = TournamentTree::new(sort.logger());
        tree.fill(arr);

        let  mut sorted  = Vec::with_capacity(n);

        //  Виконуємо сортування
        while let  Some(winner) = tree.winner()  {
            //  Переносимо значення з масиву у відсортований масив
            sorted.push(std::mem::replace(&mut  arr[winner],  unsafe { std::mem::zeroed()  }));
            tree.next_winner();
        }

        //  Переміщуємо відсортовані елементи назад у початковий масив
        for (i,  item) in sorted.into_iter().enumerate()  {
            arr[i]  = item;
            sort.log_swap();
            sort.log_swap();
        }
    }
}
```

3.2 Приклад роботи програми

Код програми для перевірки:

```
use crate::libs::sort::{testing::test_perf,              variants::TournamentSort};

pub fn main() {
    let test_cases:    Vec<usize> = vec![20,   1000,  5000,  10000,  50000];

    let mut sort  = TournamentSort::new();
    let file_name  = "turnament.csv".to_string();
    test_perf(&mut sort,    file_name,   test_cases.clone());
}
```

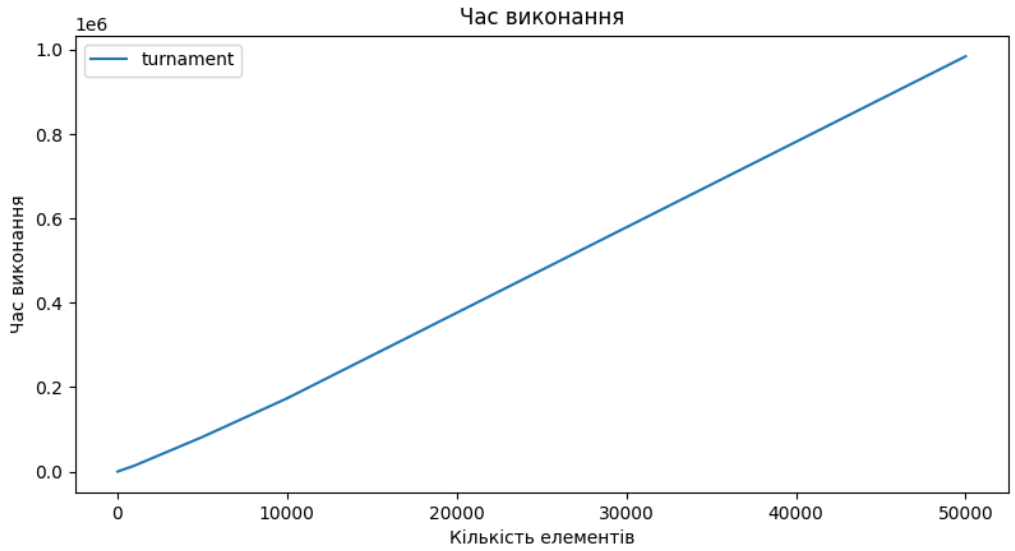

Рис. 1. Залежність часу пошуку від кількості елементів

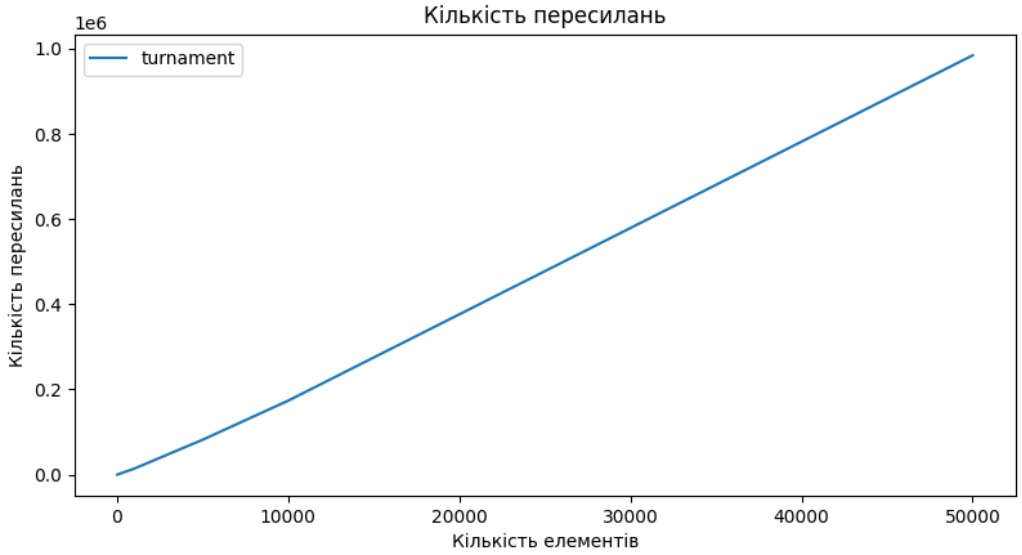

Рис. 2. Залежність кількості пересилань від кількості елементів

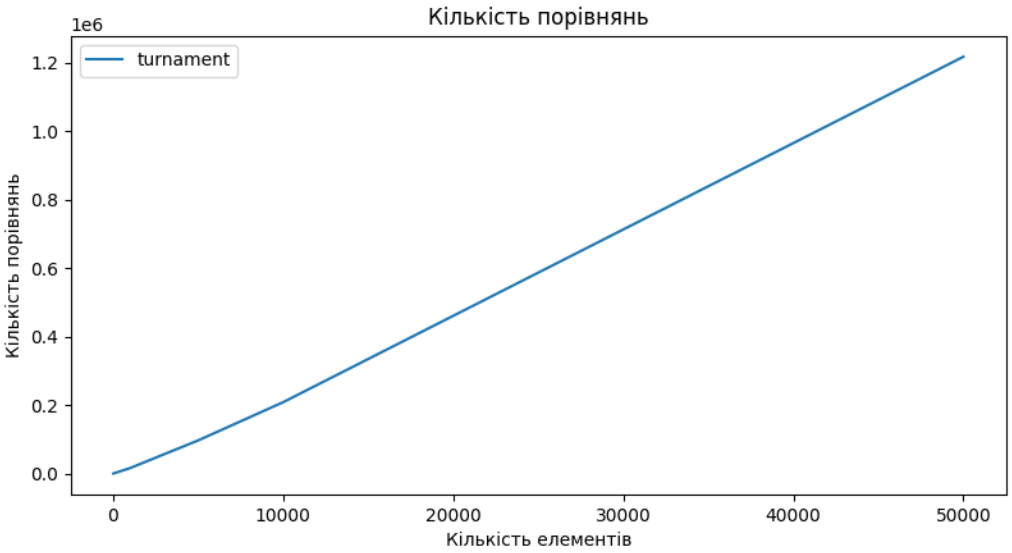

Рис. 3. Залежність кількості порівнянь від кількості елементів

4 Висновки

В ході виконання лабораторної робити було створено турнірне сортуванн. Також його було протестовано на різних обємах даних та побудовано графіки для наглядної демонстрації характеристик характеристик.