Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра ІПІ

Звіт

з лабораторної роботи №6 з дисципліни
«Алгоритми та структури даних 2. Структури даних»

„ **Піраміди**”

Виконав ІП-45 Янов Богдан Євгенійович

Перевірив Соколовський Владислав Володимирович

Київ 2025

Лабораторна робота №6

Піраміди

Мета: вивчити основні імплементації алгоритмів побудови та обробки пірамід (heap) та їх застосування для обчислення медіан у динамічному потоці даних.

Піраміди (heap): це спеціальні структури даних у вигляді бінарного дерева, які задовольняють властивість купи:

Max-heap: кожен батьківський вузол ≥ дочірніх.

Min-heap: кожен батьківський вузол ≤ дочірніх.

Вони зберігаються як масив, де для елемента з індексом i:

лівий нащадок: 2i + 1, правий: 2i + 2, батько: (i - 1) / 2.

Псевдокод алгоритмів

```
struct Heap {
	data
	is_max
}

peek(heap) {
	heap.data[0]
}

push(heap, val) {
	heap.data.push(val)
	idx = heap.data.len() - 1
	while idx > 0 {
		parent = (idx - 1) / 2
		should_swap = if heap.is_max {
			heap.data[parent] < self.data[idx]
		} else {
			heap.data[parent] > self.data[idx]
		}
		if should_swap {
			heap.data.swap(parent, idx)
			idx = parent
		} else {
			break
		}
	}
}

pop(heap) {
	if heap.data.is_empty() return
	last = heap.data.pop()
	if self.data.empty() return last

	ret = heap.data[0]
```

```
	heap.data[0] = last
	idx = 0

	loop {
		left, right = 2 * idx + 1, 2 * idx + 2
		best = idx

		for child in left, right {
			if child >= heap.data.len() continue

			let should_choose = if heap.is_max {
			heap.data[child] > heap.data[best]
			} else {
			heap.data[child] < heap.data[best]
			}
			if should_choose {
			best = child
			}
		}
		if best != idx {
			heap.data.swap(idx, best)
			idx = best
		} else {
			break
		}
	}
	ret
}
```

Вихідний код

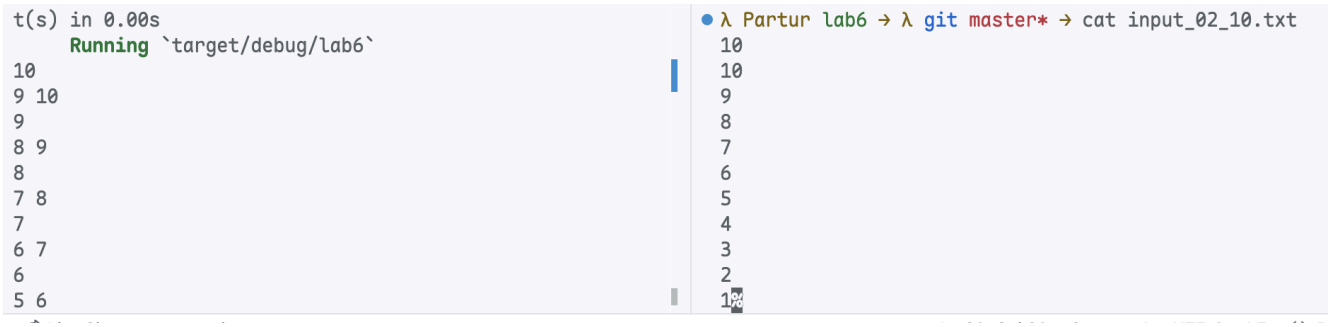
```
use std::{
    cmp::Ordering,
    fs::File,
    io::{self, BufRead, BufReader},
};

struct Heap {
    data: Vec<i64>,
    is_max: bool,
}

impl Heap {
    fn new(is_max: bool) -> Self {
        Heap {
            data: Vec::new(),
```

```
            is_max,
        }
    }

    fn len(&self) -> usize {
        self.data.len()
    }
    fn peek(&self) -> Option<&i64> {
        self.data.get(0)
    }

    fn push(&mut self, val: i64) {
        self.data.push(val);
        let mut idx = self.data.len() - 1;

        while idx > 0 {
            let parent = (idx - 1) / 2;

            let should_swap = if self.is_max {
                self.data[parent] < self.data[idx]
            } else {
                self.data[parent] > self.data[idx]
            };

            if should_swap {
                self.data.swap(parent, idx);
                idx = parent;
            } else {
                break;
            }
        }
    }

    fn pop(&mut self) -> Option<i64> {
        if self.data.is_empty() {
            return None;
```

```
        }

        let last = self.data.pop()?;
        if self.data.is_empty() {
            return Some(last);
        }

        let ret = self.data[0];
        self.data[0] = last;
        let mut idx = 0;

        loop {
            let (left, right) = (2 * idx + 1, 2 * idx + 2);
            let mut best = idx;

            for &child in [left, right].iter().filter(|&&c| c <
self.data.len()) {
                let should_choose = if self.is_max {
                    self.data[child] > self.data[best]
                } else {
                    self.data[child] < self.data[best]
                };

                if should_choose {
                    best = child;
                }
            }

            if best != idx {
                self.data.swap(idx, best);
                idx = best;
            } else {
                break;
            }
        }
```

```
            Some(ret)
        }
    }

fn main() -> io::Result<()> {
    let mut lines =
BufReader::new(File::open("input_02_10.txt")?).lines();
    let n: usize = lines.next().unwrap()?.trim().parse().unwrap();

    let (mut low, mut high) = (Heap::new(true), Heap::new(false));

    for _ in 0..n {
        let x: i64 = lines.next().unwrap()?.trim().parse().unwrap();

        if low.len() == 0 || x <= *low.peek().unwrap() {
            low.push(x);
        } else {
            high.push(x);
        }

        if low.len() > high.len() + 1 {
            high.push(low.pop().unwrap());
        } else if high.len() > low.len() + 1 {
            low.push(high.pop().unwrap());
        }

        match low.len().cmp(&high.len()) {
            Ordering::Equal => println!("{} {}", low.peek().unwrap(),
high.peek().unwrap()),
            Ordering::Greater => println!("{}", low.peek().unwrap()),
            Ordering::Less => println!("{}", high.peek().unwrap()),
        }
    }

    Ok(())
}
```

Приклад роботи



Висновок

Програма успішно реалізує алгоритм обчислення медіани у потоці даних із логарифмічною складністю. Піраміди були реалізовані вручну без використання вбудованих структур, що дозволяє повністю контролювати логіку та баланс пірамід.