

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 №5  
з навчальної дисципліни «Алгоритми та структури даних»

**Варіант 5**

Виконав студент:

Омельніцький Андрій Миколайович

Група: КН-10236

Перевірив:

Старший викладач

Бульба Сергій Сергійович

Харків-2024

# Зміст

<b>1</b>	<b>Мета роботи</b>	<b>2</b>
<b>2</b>	<b>Завдання</b>	<b>2</b>
<b>3</b>	<b>Хід виконання</b>	<b>2</b>
3.1	Підготовка до виконання . . . . .	3
3.2	Реалізація стандартної таблиці . . . . .	4
3.3	Реалізація економної таблиці . . . . .	5
3.4	Приклад роботи програми . . . . .	6
<b>4</b>	<b>Висновки</b>	<b>8</b>

# 1 Мета роботи

Набуття і закріплення навичок програмування розміщення в пам'яті специфічних масивів.

**Теми для попередньої роботи:**

- фізичне та логічне подання масивів;
- поняття про дискриптор;
- специфічні масиви: розріджені, асоціативні, симетричні, дерево відрізків.

# 2 Завдання

Розробити спосіб економного розміщення в пам'яті заданої розрідженої таблиці, де записані цілі числа. Розробити функції, що забезпечують доступ до елементів таблиці за номерами рядка і стовпця.

У програмі забезпечити запис і читання всіх елементів таблиці.

Визначити та порівняти час доступу до елементів таблиці при традиційному та економному поданні її в пам'яті. Зробити висновки.

Завдання обрати з табл. 5.1 згідно із своїм номером у журналі групи.

Номер з/п	Вміст розрідженої матриці
5	Усі елементи непарних рядків – нульові

Рис. 1. Завдання за варіантом (5)

# 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 Підготовка до виконання

Напишемо єдиний інтерфейс для обох таблиць TableMethods. А також напишемо макрос який буде реалізовувати Display trait для trait TableMethods.

Код програми:

```

1 pub trait TableMethods<T> {
2     fn get(&self, i: usize, j: usize) -> T;
3     fn set(&mut self, i: usize, j: usize, elem: T);
4     fn get_size(&self) -> (usize, usize);
5 }
6
7 #[macro_export]
8 macro_rules! impl_display_for_table {
9     ($table:ident) => {
10         impl<T: fmt::Display + Clone> fmt::Display for $table<T> {
11             fn fmt(&self, f: &mut fmt::Formatter<'_, >) -> fmt::Result {
12                 let (rows, columns) = self.get_size();
13                 for i in 0..rows {
14                     for j in 0..columns {
15                         write!(f, "{} ", self.get(i, j))?;
16                     }
17                     write!(f, "\n")?;
18                 }
19                 Ok(())
20             }
21         }
22     };
23 }
```

## 3.2 Реалізація стандартної таблиці

Для реалізації звичайної таблиці зробимо звичайний двовимірний масив який ми заповнюємо коли рядок парний.

Код програми:

```

1 use std::fmt;
2
3 use super::base::TableMethods;
4 use super::impl_display_for_table;
5
6 pub struct StandardTable<T> {
7     data: Vec<Vec<T>>,
8     default_value: T,
9     columns: usize,
10    rows: usize,
11 }
12
13 impl<T: Clone> TableMethods<T> for StandardTable<T> {
14     fn get(&self, i: usize, j: usize) -> T {
15         self.data[i][j].clone()
16     }
17
18     fn set(&mut self, i: usize, j: usize, elem: T) {
19         if i % 2 != 0 {
20             self.data[i][j] = self.default_value.clone();
21             return;
22         }
23         self.data[i][j] = elem;
24     }
25
26     fn get_size(&self) -> (usize, usize) {
27         (self.rows, self.columns)
28     }
29 }
30
31 impl<T: Clone> StandardTable<T> {
32     pub fn new(rows: usize, columns: usize, default: T) -> Self {
33         let mut data: Vec<Vec<T>> = Vec::new();
34         for _ in 0..rows {
35             data.push(vec![default.clone(); columns]);
36         }
37         StandardTable {
38             data,
39             default_value: default.clone(),
40             columns,
41             rows,
42         }
43     }
44 }
45
46 impl_display_for_table!(StandardTable);

```

### 3.3 Реалізація економної таблиці

Для реалізації економної таблиці зробимо одновимірний масив де ми будемо зберігати всі елементи. Для обрахунку індексу такого масиву використаємо функцію  $(i/2) * columns + j$

Код програми:

```

1 use std::fmt;
2
3 use super::base::TableMethods;
4 use super::impl_display_for_table;
5
6 pub struct CompactTable<T> {
7     data: Vec<T>,
8     default_value: T,
9     columns: usize,
10    rows: usize,
11 }
12
13 impl<T: Clone> TableMethods<T> for CompactTable<T> {
14     fn get(&self, i: usize, j: usize) -> T {
15         if i % 2 != 0 {
16             return self.default_value.clone();
17         }
18         self.data[(i / 2) * self.columns + j].clone()
19     }
20
21     fn set(&mut self, i: usize, j: usize, elem: T) {
22         if i % 2 != 0 {
23             return;
24         }
25         self.data[(i / 2) * self.columns + j] = elem;
26     }
27
28     fn get_size(&self) -> (usize, usize) {
29         (self.rows, self.columns)
30     }
31 }
32
33 impl<T: Clone> CompactTable<T> {
34     pub fn new(rows: usize, columns: usize, default: T) -> Self {
35         CompactTable {
36             data: vec![default.clone(); columns * rows.div_ceil(2)],
37             default_value: default.clone(),
38             columns,
39             rows,
40         }
41     }
42 }
43
44 impl_display_for_table!(CompactTable);

```

## 3.4 Приклад роботи програми

Код програми для перевірки обох таблиць:

```

1 use std::fmt::Display;
2 use std::time::Instant;
3
4 use super::variants::base::TableMethods;
5 use super::variants::compact::CompactTable;
6 use super::variants::standard::StandardTable;
7
8 fn table_test<T: TableMethods<i32> + Display>(title: &str, mut table: T) {
9     println!("{:=^60}", title);
10    let cyles = 1000;
11    let (rows, columns) = table.get_size();
12
13    for i in 0..rows {
14        for j in 0..columns {
15            table.set(i, j, (i + j) as i32);
16        }
17    }
18    println!("{}", table);
19
20    // odd
21    let mut read: u128 = 0;
22    let mut write: u128 = 0;
23    for _ in 0..cyles {
24        for i in 0..rows {
25            if i % 2 == 0 {
26                continue;
27            }
28            for j in 0..columns {
29                let now = Instant::now();
30                table.set(i, j, (i + j) as i32);
31                write += now.elapsed().as_nanos();
32            }
33        }
34    }
35    for _ in 0..cyles {
36        for i in 0..rows {
37            if i % 2 == 0 {
38                continue;
39            }
40            for j in 0..columns {
41                let now = Instant::now();
42                table.get(i, j);
43                read += now.elapsed().as_nanos();
44            }
45        }
46    }
47    println!("~Odd_rows_Times_(Read/Write): {}ns/{}ns", read, write);
48
49    // even
50    let mut read: u128 = 0;
51    let mut write: u128 = 0;
52    for _ in 0..cyles {
53        for i in 0..rows {
54            if i % 2 != 0 {
55                continue;
56            }
57            for j in 0..columns {
58                let now = Instant::now();
59                table.set(i, j, (i + j) as i32);
60                write += now.elapsed().as_nanos();
61            }
62        }
63    }
64    for _ in 0..cyles {
65        for i in 0..rows {
66            if i % 2 != 0 {
67                continue;
68            }
69            for j in 0..columns {
70                let now = Instant::now();

```

```

71         table.get(i, j);
72         read += now.elapsed().as_nanos();
73     }
74 }
75 }
76 println!("Even rows Times (Read/Write): {} ns / {} ns", read, write);
77 println!("{}", "");
78 }
79
80 pub fn main() {
81     let rows = 6;
82     let columns = 10;
83
84     table_test("Compact Table", CompactTable::<i32>::new(rows, columns, 0));
85     table_test(
86         "Standard Table",
87         StandardTable::<i32>::new(rows, columns, 0),
88     );
89 }

```

```

=====Standard Table=====
0 1 2 3 4 5 6 7 8 9
0 0 0 0 0 0 0 0 0 0
2 3 4 5 6 7 8 9 10 11
0 0 0 0 0 0 0 0 0 0
4 5 6 7 8 9 10 11 12 13
0 0 0 0 0 0 0 0 0 0

Odd rows Times (Read/Write): 2039047 ns / 1945139 ns
Even rows Times (Read/Write): 1872317 ns / 1887925 ns
=====

```

Рис. 2. Приклад роботи для стандартної таблиці

```

=====Compact Table=====
0 1 2 3 4 5 6 7 8 9
0 0 0 0 0 0 0 0 0 0
2 3 4 5 6 7 8 9 10 11
0 0 0 0 0 0 0 0 0 0
4 5 6 7 8 9 10 11 12 13
0 0 0 0 0 0 0 0 0 0

Odd rows Times (Read/Write): 901848 ns / 1534855 ns
Even rows Times (Read/Write): 1344671 ns / 1600224 ns
=====

```

Рис. 3. Приклад роботи для економної таблиці



## 4 Висновки

В ході виконання лабораторної роботи було створено стандартну та економну версію таблиць. Після порівняння швидкодії обох таблиць, компактна таблиця працює швидше, а особливо швидкість на читання для непарних строк, але й швидкість доступу до парних строк виросла, бо в нас тільки один рівень вкладеності в масиві.