Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

**про виконання лабораторних та практичних робіт блоку № 6**

На тему:  «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.»

***з дисципліни:*** «Мови та парадигми програмування»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт № 6

**Виконав:**

студент групи ШІ-14

Нападайло Микита Олександрович

# **Тема роботи:**

Робота з динамічними структурами даних(чергами, списками, деревами), їхня обробка(сортування, пошук, додавання елементів, видалення елементів і т.д.)

# **Мета роботи:**

# Познайомитися з динамічними структурами: однозв’язними та двозв’язними списками, бінарними деревами, зокрема бінарними деревами пошуку, чергами, векторами; навчитися створювати і застосовувати їх, розібратися з перевагами та недоліками цих структур.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Черга.
* Тема №2: Однозв’язний список.
* Тема №3: Двозв’язний список.
* Тема №4: Бінарне дерево.
* Тема №5: Бінарне дерево пошуку.

1. Індивідуальний план опрацювання теорії:

* Тема №1: Черга.
  + Джерела Інформації
    - Стаття <https://www.geeksforgeeks.org/introduction-to-queue-data-structure-and-algorithm-tutorials/> .
  + Що опрацьовано:
    - Основний принцип роботи черги, способи її імплементації, функціонал.
  + Статус: Ознайомлений
  + Початок опрацювання теми: 01.12.2023
  + Завершення опрацювання теми: 03.12.2023
* Тема №2: Однозв’язний список.
  + - Стаття <https://www.scaler.com/topics/linked-list-in-cpp/>
    - Відео <https://youtu.be/m7rrk65GiXY>
  + Що опрацьовано:
    - Як влаштовані однозв’язні списки, як їх створювати та обробляти: функції для роботи з такими списками
  + Статус: Ознайомлений
  + Початок опрацювання теми: 09.12.2023
  + Завершення опрацювання теми: 10.12.2023
* Тема №3: Двозв’язний список.
  + Джерела Інформації
    - Стаття <https://www.scaler.com/topics/doubly-linked-list-cpp/>
  + Що опрацьовано:
    - Структура елементів двозв’язного списку, відмінність одно- та двозв’язного списку
  + Статус: Ознайомлений
  + Початок опрацювання теми: 09.12.2023
  + Завершення опрацювання теми: 09.12.2023
* Тема №4 Бінарне дерево.
  + - Відео <https://www.youtube.com/watch?v=_IhTp8q0Mm0> .
    - Стаття <https://www.geeksforgeeks.org/introduction-to-tree-data-structure-and-algorithm-tutorials/?ref=shm>
  + Що опрацьовано:
    - Структура бінарного дерева, імлементація основних функцій для роботи з бінарним деревом
  + Статус: Ознайомлений
  + Початок опрацювання теми: 11.12.2023
  + Завершення опрацювання теми: 12.12.2023
* Тема №5: Бінарне дерево пошуку.
  + Джерела Інформації
    - Стаття <https://www.geeksforgeeks.org/introduction-to-binary-search-tree-data-structure-and-algorithm-tutorials/>
  + Що опрацьовано:
    - Використання бінарних дерев пошуку, їхня відмінність від звичайних бінарних дерев
  + Статус: Ознайомлений
  + Початок опрацювання теми: 11.12.2023
  + Завершення опрацювання теми: 12.12.2023

# **Виконання роботи:**

## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання №1 **VNS Lab 10 - Task 1-N**

* Варіант завдання – 18
* Деталі завдання: Написати програму, у якій створюються динамічні структури й виконати їхню обробку у відповідності зі своїм варіантом. Для кожного варіанту розробити такі функції:

1. Створення списку.

2. Додавання елемента в список (у відповідності зі своїм варіантом).

3. Знищення елемента зі списку (у відповідності зі своїм варіантом).

4. Друк списку.

5. Запис списку у файл.

6. Знищення списку.

7. Відновлення списку з файлу.

Записи в лінійному списку містять ключове поле типу \*char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим ключем. Додати К елементів у початок списку.

Завдання №2 **Algotester Lab 5**

* Варіант завдання - 2
* Деталі завдання: В пустелі існує незвичайна печера, яка є двовимірною. Її висота це N, ширина - M. Всередині печери є пустота, пісок та каміння. Пустота позначається буквою О , пісок S і каміння X; Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння. Ваше завдання сказати як буде виглядати печера після землетрусу.

Завдання №3 **Algotester Lab 7-8 (struct)**

* Варіант завдання - 3
* Деталі завдання: Власноруч реалізувати структуру даних "Двійкове дерево пошуку".Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його параметри. Вам будуть поступати запити такого типу:

**Вставка**:  
Ідентифікатор – insert: Ви отримуєте ціле число value - число, яке треба вставити в дерево.

**Пошук**:  
Ідентифікатор – contains: Ви отримуєте ціле число value - число, наявність якого у дереві необхідно перевірити. Якщо value наявне в дереві - ви виводите Yes, у іншому випадку No.

**Визначення розміру**:  
Ідентифікатор – size: Ви не отримуєте аргументів. Ви виводите кількість елементів у дереві.

**Вивід дерева на екран**  
Ідентифікатор – print: Ви не отримуєте аргументів. Ви виводите усі елементи дерева через пробіл. Реалізувати використовуючи перегрузку оператора <<

Завдання №4 **Algotester Lab 7-8 (class template)**

* Варіант завдання – 3
* Деталі завдання: Власноруч реалізувати структуру даних "Двійкове дерево пошуку".Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його параметри. Вам будуть поступати запити такого типу:

**Вставка**:  
Ідентифікатор – insert: Ви отримуєте ціле число value - число, яке треба вставити в дерево.

**Пошук**:  
Ідентифікатор – contains: Ви отримуєте ціле число value - число, наявність якого у дереві необхідно перевірити. Якщо value наявне в дереві - ви виводите Yes, у іншому випадку No.

**Визначення розміру**:  
Ідентифікатор – size: Ви не отримуєте аргументів. Ви виводите кількість елементів у дереві.

**Вивід дерева на екран**  
Ідентифікатор – print: Ви не отримуєте аргументів. Ви виводите усі елементи дерева через пробіл. Реалізувати використовуючи перегрузку оператора <<

* Важливі деталі для врахування в імплементації програми: структура має бути написана як шаблон класу, у якості параметру використати int. Використовувати STL заборонено.

Завдання №5 **Class Practice Work**

* Деталі завдання:

1) Реалізувати метод реверсу списку

2) Реалізувати метод порівняння списків

3) Реалізувати метод додавання великих чисел за допомогою зв’язних списків

4) Реалізувати метод віддзеркалення бінарного дерева

5) Реалізувати метод підрахунку сум підвузлів бінарного дерева

Завдання №6 **Self-Practice Work**

* Варіант завдання: **Algotester Lab 5v3**
* Деталі завдання: У вас є карта гори розміром N×M .Також ви знаєте координати {x,y}, у яких знаходиться вершина гори. Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число. Клітинки які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

## **2. Дизайн та планована оцінка часу виконання завдань:**

Програма №1 **VNS Lab 10 - Task 1-N**

* Планований час на реалізацію – 2 години

Програма №2 **Algotester Lab 5**

* Планований час на реалізацію – 1 година

Програма №3 **Algotester Lab 7-8 (struct)**

* Планований час на реалізацію – 2 години

Програма №4 **Algotester Lab 7-8 (class template)**

* Планований час на реалізацію – 2 години

Програма №5 **Class Practice Work**

* Планований час на реалізацію – 5 годин

Програма №6 **Self-Practice Work**

* Блок-схема

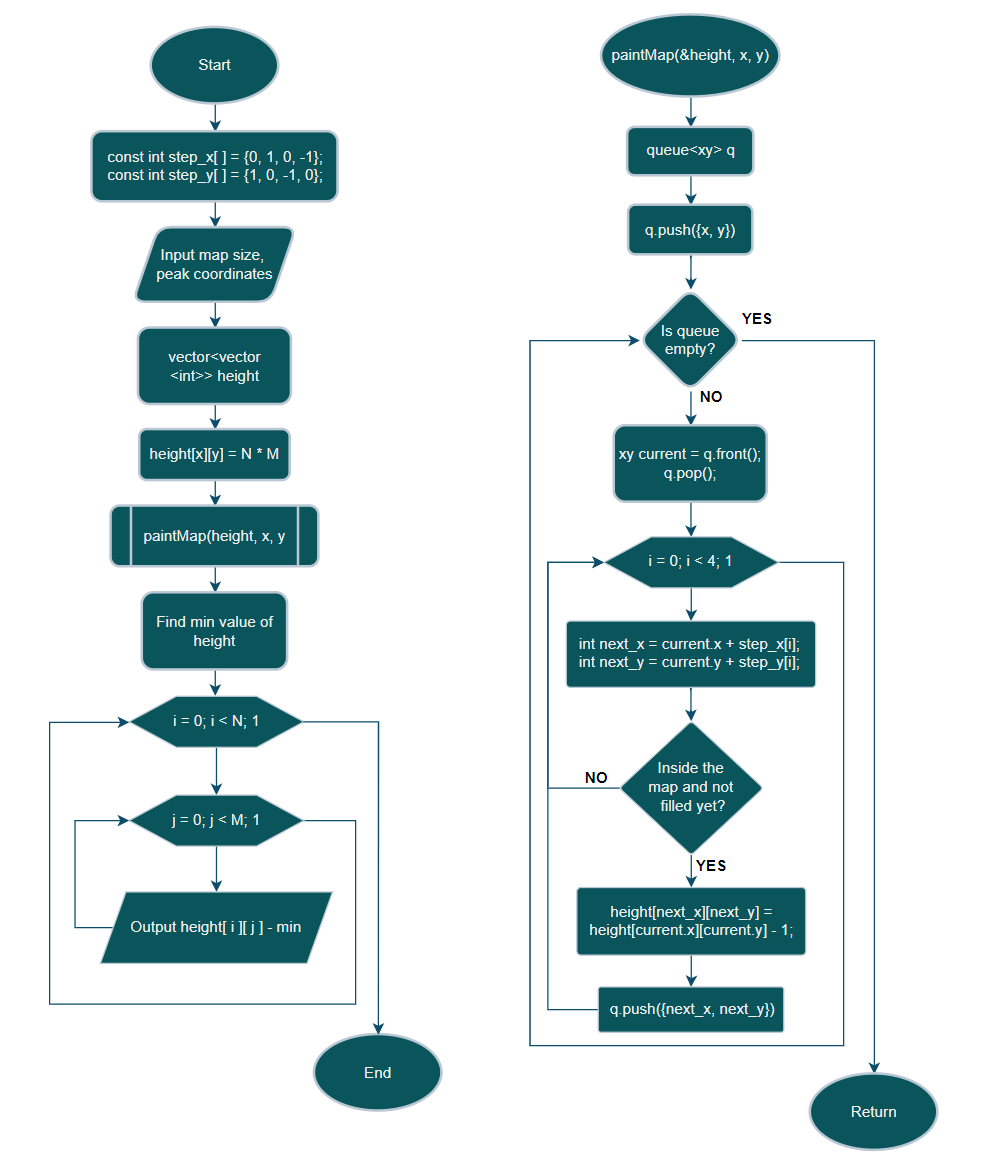


Рисунок 1: блок-схема до програми 6

* Планований час на реалізацію – 2 години

## **3. Конфігурація середовища до виконання завдань:**

Використав конфігурацію з Епіку 1.

## **4. Код програм з посиланням на зовнішні ресурси:**

Завдання №1 **VNS Lab 10 - Task 1-N**

#include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

struct Node

{

    char\* key;

    Node\* prev;

    Node\* next;

};

struct DLL

{

    Node\* head;

    Node\* tail;

};

DLL\* create\_List();

void AddNode(DLL\* list, const char\* data);

void DeleteNode(DLL\* list, const char\* target);

void PrintList(DLL\* list);

void Write2File(DLL\* list, const char\* name);

void DeleteList(DLL\* list);

DLL\* RestoreFromFile(const char\* filename);

int main()

{

    DLL\* list\_ptr = create\_List();

    AddNode(list\_ptr, "programming");

    AddNode(list\_ptr, "like");

    AddNode(list\_ptr, "not");

    AddNode(list\_ptr, "do");

    AddNode(list\_ptr, "I");

    DeleteNode(list\_ptr, "not");

    Write2File(list\_ptr, "list\_backup.txt");

    DeleteList(list\_ptr);

    DLL\* list = RestoreFromFile("list\_backup.txt");

    DeleteList(list);

}

DLL\* create\_List()

{

    DLL\* list = new DLL;

    list -> head = list -> tail = nullptr;

    return list;

}

void AddNode(DLL\* list, const char\* data)

{

    Node\* new\_Node = new Node;

    new\_Node -> key = new char[strlen(data) + 1];

    strcpy(new\_Node -> key, data);

    new\_Node -> prev = nullptr;

    new\_Node -> next = list -> head;

    if (list -> head != nullptr) {

    list -> head -> prev = new\_Node;

    }

    list -> head = new\_Node;

    if (list -> tail == nullptr)

    {

        list -> tail = new\_Node;

    }

    PrintList(list);

}

void DeleteNode(DLL\* list, const char\* target)

{

    Node\* current = list -> head;

    while (current != nullptr)

    {

        if (strcmp(current -> key, target) == 0)

        {

            if (current == list -> head)

            {

                list -> head = current -> next;

            }

            else

            {

                current -> prev -> next = current -> next;

                if (current -> next != nullptr)

                {

                    current -> next -> prev = current -> prev;

                }

            }

            delete[] current -> key;

            delete current;

            return;

        }

        current = current -> next;

    }

    PrintList(list);

}

void PrintList(DLL\* list)

{

    Node\* current = list -> head;

    if (current == nullptr)

    {

        cout << "Ooops, this list is empty...\n";

        return;

    }

    while (current != nullptr)

    {

        cout << current -> key << ' ';

        current = current -> next;

    }

    cout << endl;

}

void Write2File(DLL\* list, const char\* name)

{

    ofstream fout(name);

    if (!fout)

    {

        cerr << "Failed to write a list into the file\n";

        return;

    }

    Node\* current = list -> head;

    while (current != nullptr)

    {

        fout << current -> key << endl;

        current = current -> next;

    }

    fout.close();

}

void DeleteList(DLL\* list)

{

    Node\* current = list -> head;

    while (current != nullptr)

    {

        Node\* tmp = current;

        current = current -> next;

        delete tmp;

    }

    list -> head = list -> tail = nullptr;

    PrintList(list);

}

DLL\* RestoreFromFile(const char\* filename)

{

    ifstream fin (filename);

    if (!fin)

    {

        cerr << "Failed to open the file\n";

        return nullptr;

    }

    DLL\* list = create\_List();

    char line[256];

    while (fin.getline(line, 256))

    {

        AddNode(list, line);

    }

    fin.close();

    Node\* current = list -> head;

    Node\* tmp = nullptr;

    while (current != nullptr)

    {

        tmp = current -> prev;

        current -> prev = current -> next;

        current -> next = tmp;

        current = current -> prev;

    }

    if (tmp != nullptr)

    {

        list -> head = tmp -> prev;

    }

    PrintList(list);

    return list;

}

Завдання №2 **Algotester Lab 5**

#include <iostream>

using namespace std;

void swap(char &a, char &b)

{

    char tmp = a;

    a = b;

    b = tmp;

}

int main()

{

    int N, M;

    cin >> N >> M;

    char cave[N][M];

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < M; j++)

        {

            cin >> cave[i][j];

        }

    }

    for (int i = N - 1; i >= 0; i--)

    {

        for (int j = 0; j < M; j++)

        {

            if (cave[i][j] == 'S')

            {

                int t = i;

                while (t + 1 < N && cave[t + 1][j] == 'O')

                {

                    swap(cave[t][j], cave[t + 1][j]);

                    t++;

                }

            }

        }

    }

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < M; j++)

        {

            cout << cave[i][j];

        }

        cout << endl;

    }

}

Завдання №3 **Algotester Lab 7-8 (struct)**

#include <iostream>

using namespace std;

struct Node

{

    int key;

    Node \*left, \*right;

};

Node\* newTree(int value)

{

    Node \*root = new Node;

    root -> key = value;

    root -> left = root -> right = nullptr;

    return root;

}

Node\* insert(Node \*parent, int value)

{

    if (parent == nullptr)

    {

        parent = newTree(value);

        return parent;

    }

    if (value == parent -> key) return parent;

    if (value < parent -> key) parent -> left = insert(parent -> left, value);

    else parent -> right = insert(parent -> right, value);

    return parent;

}

bool contains(Node \*parent, int value, bool present = false)

{

    if (parent == nullptr) return false;

    if (value == parent -> key || present)

    {

        present = true;

        return true;

    }

    if (value < parent -> key) return contains(parent -> left, value, present);

    else return contains(parent -> right, value, present);

}

void print(Node \*parent)

{

    if (parent != nullptr)

    {

        print(parent -> left);

        cout << parent -> key << ' ';

        print(parent -> right);

    }

}

int size(Node \*parent)

{

    if (parent == nullptr) return 0;

    else return 1 + size(parent -> left) + size(parent -> right);

}

int main()

{

    int Q;

    cin >> Q;

    Node \*r = nullptr;

    while (Q--)

    {

        string request;

        cin >> request;

        switch (request[0])

        {

            case 'i':

            {

                int value;

                cin >> value;

                r = insert(r, value);

                break;

            }

            case 'c':

            {

                int value;

                cin >> value;

                cout << (contains(r, value) ? "Yes\n" : "No\n");

                break;

            }

            case 's':

            {

                cout << size(r) << endl;

                break;

            }

            default:

            {

                print(r);

            }

        }

    }

}

Завдання №4 **Algotester Lab 7-8 (class template)**

#include <iostream>

using namespace std;

template <class T> struct Node

{

    T key;

    Node<T> \*left, \*right;

};

template <class T> class BST

{

    private:

        Node<T> \*root;

        Node<T>\* newTree(T value)

        {

            Node<T> \*root = new Node<T>;

            root -> key = value;

            root -> left = root -> right = nullptr;

            return root;

        }

        Node<T>\* insertPrivate(Node<T> \*parent, T value)

        {

            if (parent == nullptr)

            {

                parent = newTree(value);

                return parent;

            }

            if (value == parent -> key) return parent;

            if (value < parent -> key) parent -> left = insertPrivate(parent -> left, value);

            else parent -> right = insertPrivate(parent -> right, value);

            return parent;

        }

        bool containsPrivate(Node<T> \*parent, T value, bool present)

        {

            if (parent == nullptr) return false;

            if (value == parent -> key || present)

            {

                present = true;

                return true;

            }

            if (value < parent -> key) return containsPrivate(parent -> left, value, present);

            else return containsPrivate(parent -> right, value, present);

        }

        void printPrivate(Node<T> \*parent)

        {

            if (parent != nullptr)

            {

                printPrivate(parent -> left);

                cout << parent -> key << ' ';

                printPrivate(parent -> right);

            }

        }

        int sizePrivate(Node<T> \*parent)

        {

            if (parent == nullptr) return 0;

            else return 1 + sizePrivate(parent -> left) + sizePrivate(parent -> right);

        }

    public:

        BST() : root(nullptr) {}

        void insert(int value)

        {

            root = insertPrivate(root, value);

        }

        bool contains(int value)

        {

            return containsPrivate(root, value, false);

        }

        void print()

        {

            printPrivate(root);

        }

        int size()

        {

            return sizePrivate(root);

        }

};

int main()

{

    int Q;

    cin >> Q;

    BST<int> tree;

    while (Q--)

    {

        string request;

        cin >> request;

        switch (request[0])

        {

            case 'i':

            {

                int value;

                cin >> value;

                tree.insert(value);

                break;

            }

            case 'c':

            {

                int value;

                cin >> value;

                cout << (tree.contains(value) ? "Yes\n" : "No\n");

                break;

            }

            case 's':

            {

                cout << tree.size() << endl;

                break;

            }

            default:

            {

                tree.print();

            }

        }

    }

}

Завдання №5 **Class Practice Work**

#include <iostream>

using namespace std;

struct Node

{

    int key;

    Node\* next;

};

struct TreeNode {

    int key;

    TreeNode \*left;

    TreeNode \*right;

    TreeNode(int x) : key(x), left(nullptr), right(nullptr) {}

};

Node\* createList(int value)

{

    Node\* ptr = new Node;

    ptr -> key = value;

    ptr -> next = nullptr;

    return ptr;

}

void addNode(Node\* list, int value)

{

    Node\* tmp = list;

    while (tmp -> next != nullptr)

    {

        tmp = tmp -> next;

    }

    Node\* new\_Node = new Node;

    new\_Node -> key = value;

    new\_Node -> next = nullptr;

    tmp -> next = new\_Node;

}

void printList(Node\* head, string whtspc)

{

    Node\* current = head;

    while (current != nullptr)

    {

        cout << current -> key << whtspc;

        current = current -> next;

    }

    cout << endl;

}

Node\* reverse(Node\* list1)

{

    Node \*current = list1;

    Node \*next = nullptr;

    Node \*prev = nullptr;

    while (current != nullptr)

    {

        next = current -> next;

        current -> next = prev;

        prev = current;

        current = next;

    }

    return prev;

}

bool compare(Node \*list1, Node \*list2)

{

    Node \*cur1 = list1, \*cur2 = list2;

    while (cur1 != nullptr && cur2 != nullptr)

    {

        if (cur1 -> key != cur2 -> key) return false;

        cur1 = cur1 -> next;

        cur2 = cur2 -> next;

    }

    return (cur1 == nullptr && cur2 == nullptr);

}

void deleteList(Node \*head)

{

    while (head != nullptr)

    {

        Node \*tmp = head;

        head = head -> next;

        delete tmp;

    }

}

Node \*add(Node \*list1, Node \*list2)

{

    Node \*h1 = list1, \*h2 = list2;

    Node \*SUM = nullptr;

    Node \*tail = nullptr;

    int carry = 0;

    while (h1 != nullptr || h2 != nullptr || carry)

    {

        int sum = carry;

        if (h1 != nullptr)

        {

            sum += h1->key;

            h1 = h1->next;

        }

        if (h2 != nullptr)

        {

            sum += h2->key;

            h2 = h2->next;

        }

        carry = sum / 10;

        sum = sum % 10;

        Node \*newNode = new Node;

        newNode->key = sum;

        newNode->next = nullptr;

        if (SUM == nullptr)

        {

            SUM = tail = newNode;

        } else

        {

            tail->next = newNode;

            tail = newNode;

        }

    }

    return SUM;

}

TreeNode\* create\_mirror\_flip(TreeNode \*root)

{

    if (root == nullptr)

    {

        return nullptr;

    }

    TreeNode \*newNode = new TreeNode(root -> key);

    newNode -> left = create\_mirror\_flip(root -> right);

    newNode -> right = create\_mirror\_flip(root -> left);

    return newNode;

}

void inorderTraversal(TreeNode \*root)

{

    if (root == nullptr)

    {

        return;

    }

    inorderTraversal(root -> left);

    cout << root -> key << " ";

    inorderTraversal(root->right);

}

void treeSum(TreeNode \*root)

{

    if (root == nullptr || root -> left == nullptr && root -> right == nullptr) return;

    treeSum(root -> left);

    treeSum(root -> right);

    int value\_left = (root -> left != nullptr) ? root -> left -> key : 0;

    int value\_right = (root -> right != nullptr) ? root -> right -> key : 0;

    root -> key = value\_left + value\_right;

}

int main()

{

    // Task 1

    cout << "\t TASK 1\n";

    Node\* head = createList(1);

    for (int i = 2; i <= 8; i++)

    {

        addNode(head, i);

    }

    printList(head, " ");

    head = reverse(head);

    printList(head, " ");

    deleteList(head);

    cout << "=========================\n";

    // Task 2

    cout << "\t TASK 2\n";

    Node \*head1 = createList(1);

    Node \*head2 = createList(1);

    for (int i = 2; i <= 8; i++)

    {

        addNode(head1, i);

        addNode(head2, i);

    }

    addNode(head2, 9);

    printList(head1, " ");

    printList(head2, " ");

    if (compare(head1, head2))

    {

        cout << "These lists are equal\n";

    }

    else cout << "These lists are not equal\n";

    deleteList(head1);

    deleteList(head2);

    cout << "=========================\n";

    // Task 3

    cout << "\t TASK 3\n";

    Node \*num1 = createList(3);

    addNode(num1, 7);

    addNode(num1, 9);

    cout << "The first number: ";

    printList(num1, "");

    Node \*num2 = createList(9);

    addNode(num2, 7);

    addNode(num2, 3);

    cout << "The second number: ";

    printList(num2, "");

    Node \*sum = reverse(add(num1, num2));

    cout << "The sum of two numbers: ";

    printList(sum, "");

    deleteList(num1);

    deleteList(num2);

    deleteList(sum);

    cout << "=========================\n";

    // Task 4

    cout << "\t TASK 4\n";

    TreeNode \*root = new TreeNode(1);

    root -> left = new TreeNode(2);

    root -> right = new TreeNode(3);

    root -> left -> left = new TreeNode(4);

    root -> left -> right = new TreeNode(5);

    root -> right -> left = new TreeNode(6);

    root -> right -> right = new TreeNode(7);

    cout << "Original tree: ";

    inorderTraversal(root);

    cout << endl;

    TreeNode \*mirrorFlipped = create\_mirror\_flip(root);

    cout << "Mirror-flipped tree: ";

    inorderTraversal(mirrorFlipped);

    cout << endl;

    cout << "=========================\n";

    // Task 5

    cout << "\t TASK 5\n";

    cout << "Original tree: ";

    inorderTraversal(root);

    cout << endl;

    treeSum(root);

    cout << "A tree of sums: ";

    inorderTraversal(root);

}

Завдання №6 **Self-Practice Work**

#include <iostream>

#include <vector>

#include <queue>

using namespace std;

struct xy

{

    int x;

    int y;

};

const int step\_x[] = {0, 1, 0, -1};

const int step\_y[] = {1, 0, -1, 0};

void paintMap(vector<vector<int>> &height, int x, int y)

{

    int N = height.size();

    int M = height[0].size();

    queue<xy> q;

    q.push({x, y});

    while (!q.empty())

    {

        xy current = q.front();

        q.pop();

        for (int i = 0; i < 4; i++)

        {

            int next\_x = current.x + step\_x[i];

            int next\_y = current.y + step\_y[i];

            if (next\_x >= 0 && next\_x < N && next\_y >= 0 && next\_y < M && height[next\_x][next\_y] == 0)

            {

                height[next\_x][next\_y] = height[current.x][current.y] - 1;

                q.push({next\_x, next\_y});

            }

        }

    }

}

int main()

{

    int N, M, x, y;

    cin >> N >> M >> x >> y;

    x--, y--;

    vector<vector<int>> height(N, vector<int>(M, 0));

    height[x][y] = N \* M;

    paintMap(height, x, y);

    int min = height[0][0];

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < M; j++)

        {

            if (height[i][j] < min) min = height[i][j];

        }

    }

    for (int i = 0; i < N; i++)

    {

        for (int j = 0; j < M; j++)

        {

            cout << height[i][j] - min << " ";

        }

        cout << endl;

    }

}

## **5. Результати виконання завдань, тестування та фактично затрачений час:**

Завдання №1 **VNS Lab 10 - Task 1-N**

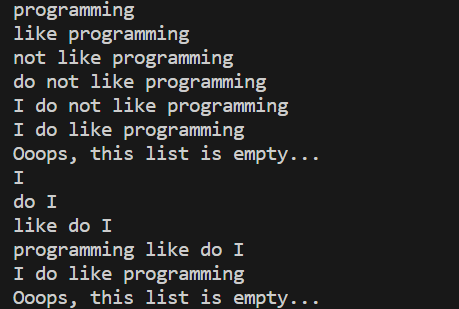


Рисунок 2: результат виконання програми 1

Час затрачений на виконання завдання: 3 години

Завдання №2 **Algotester Lab 5**

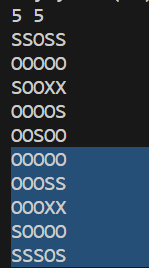


Рисунок 3: результат виконання програми 2

Час затрачений на виконання завдання: 50 хвилин

Завдання №3 **Algotester Lab 7-8 (struct)**

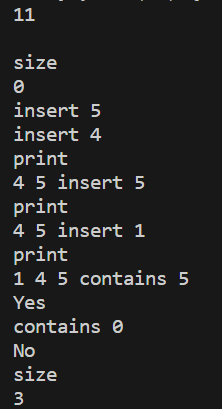


Рисунок 4: результат виконання програми 3

Час затрачений на виконання завдання: 2 години

Завдання №4 **Algotester Lab 7-8 (class template)**

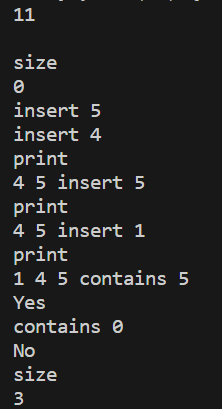


Рисунок 5: результат виконання програми 4

Час затрачений на виконання завдання: 1,5 години

Завдання №5 **Class Practice Work**

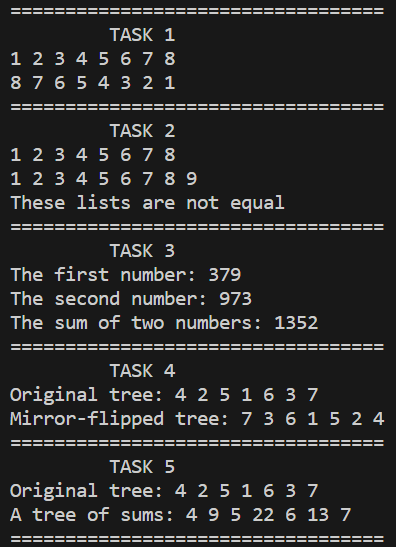


Рисунок 6: результат виконання програми 5

Час затрачений на виконання завдання: 5 годин

Завдання №6 **Self-Practice Work**

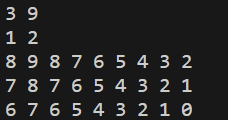


Рисунок 7: результат виконання програми 6

Час затрачений на виконання завдання: 12 годин

## **6. Кооперація з командою:**

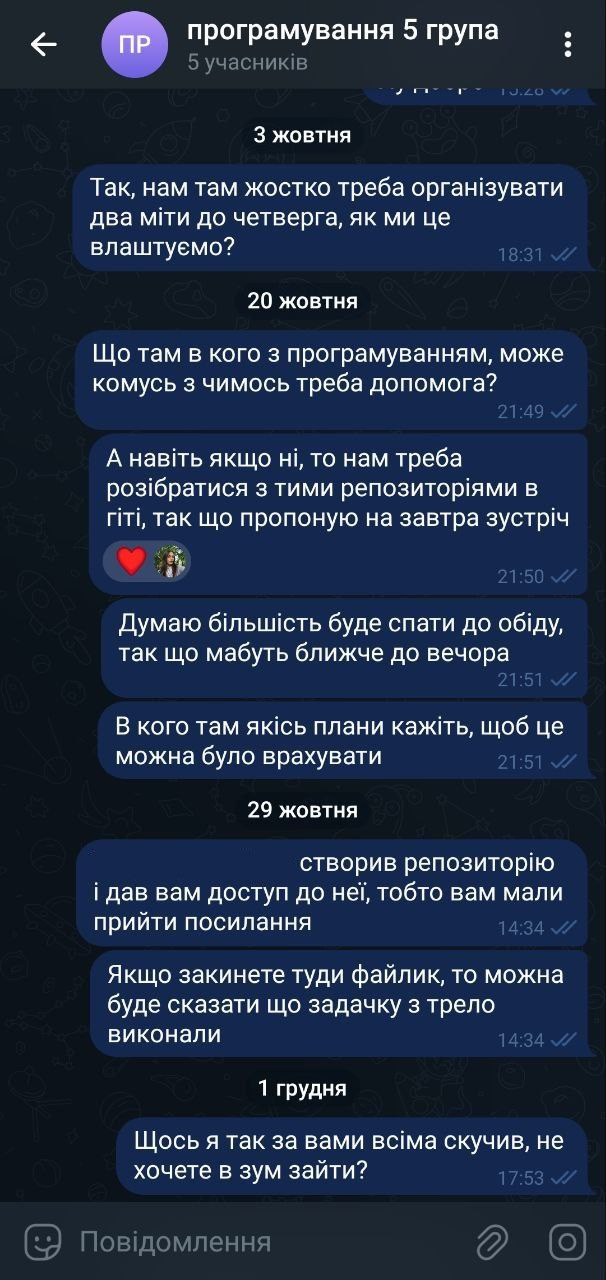


Рисунок 8: скрін спроб кооперації з командою

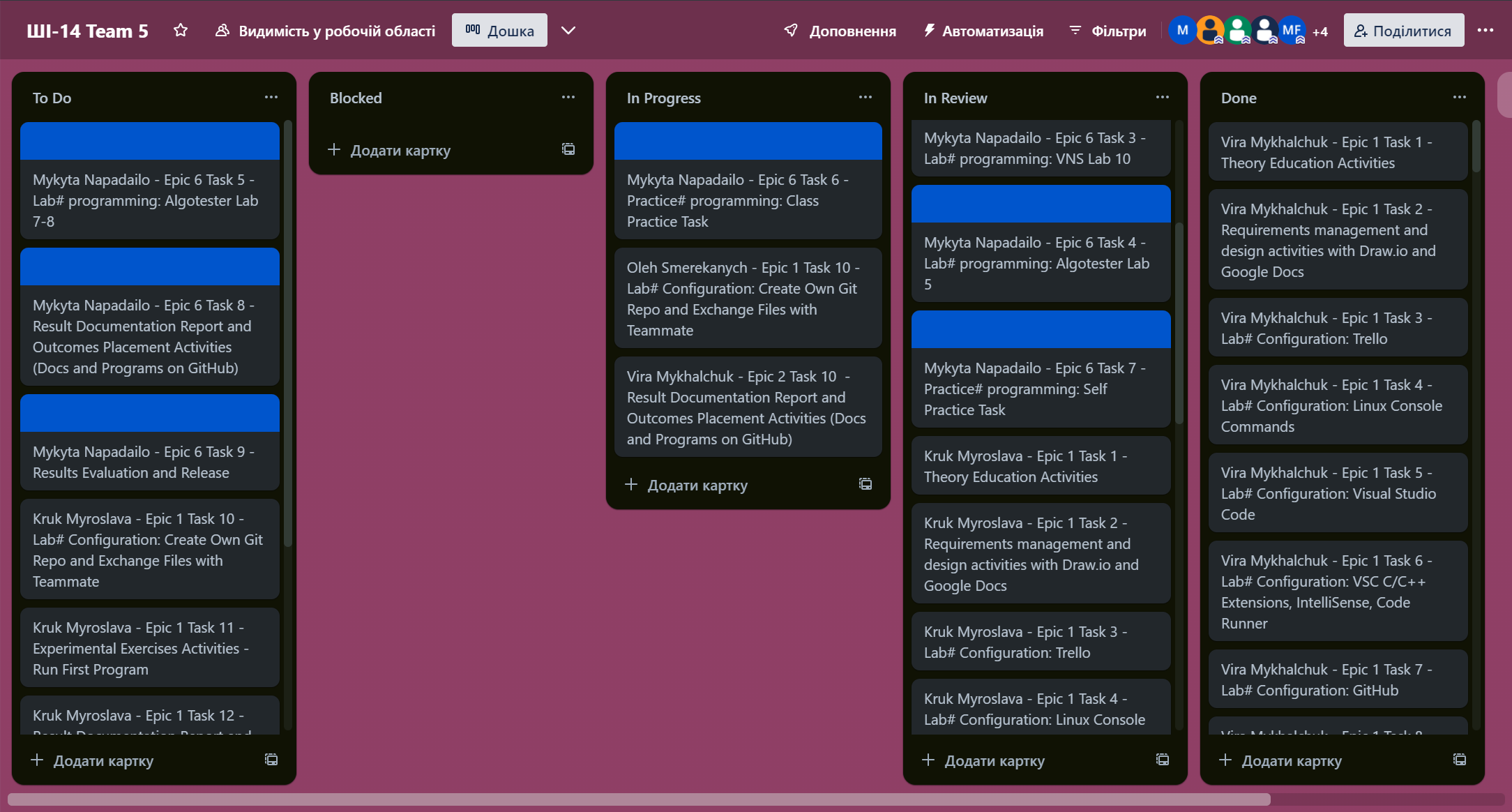
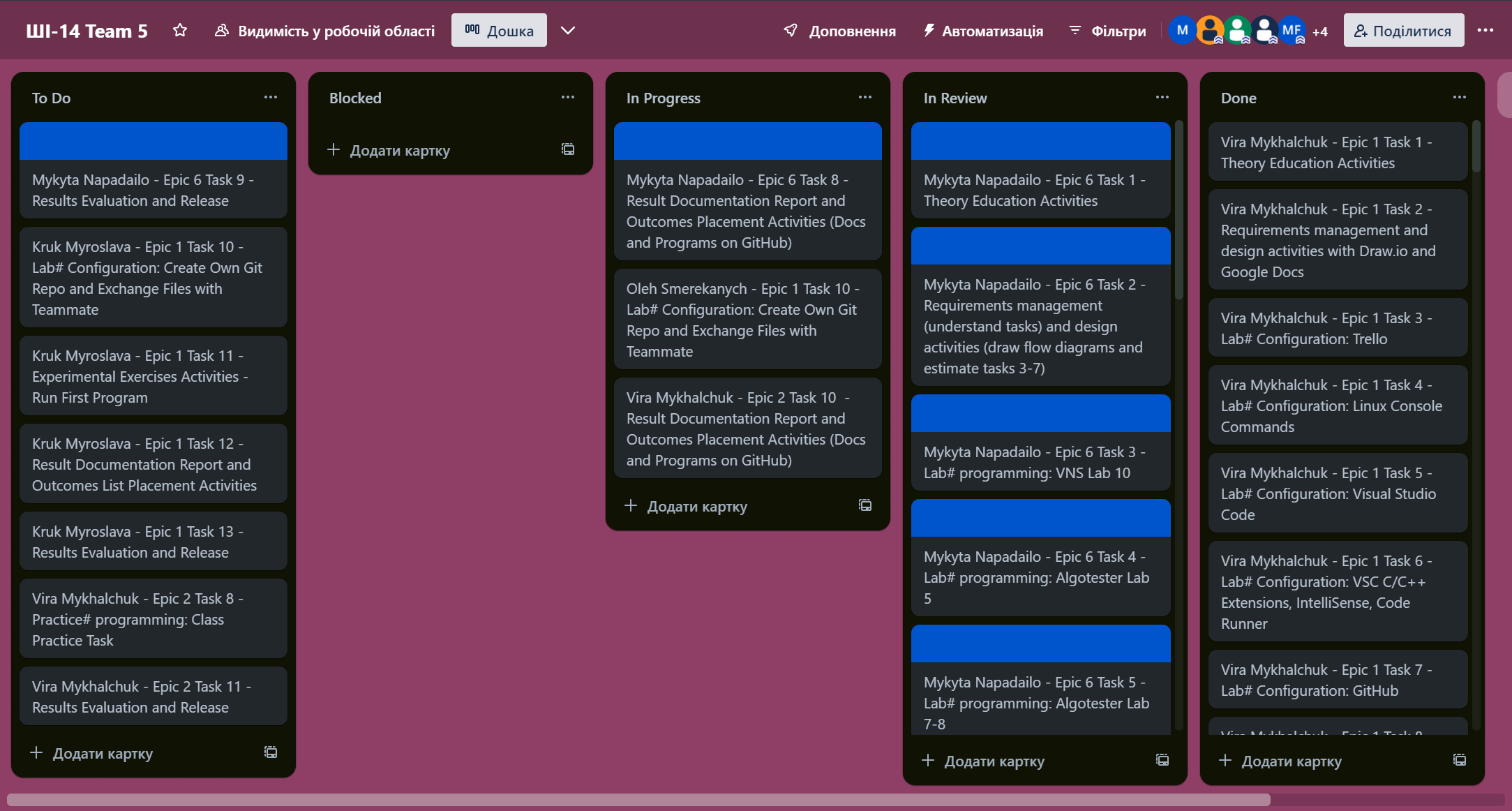


Рисунок 9: скрін прогресу Trello 11.12.23

Рисунок 10: скрін прогресу Trello 12.12.23

# **Висновки:**

В ході виконання лабораторних та практичних робіт в цьому епіку, я реалізував такі динамічні структури, як: одно- і двозв’язні списки, бінарне дерево та бінарне дерево пошуку – і навчився створювати і використовувати функції для їхньої обробки; також вперше використав чергу в своєму коді. Зробивши це, я ознайомився з недоліками та перевагами кожної з структур.