Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

A blue and white logo

Description automatically generated

**Звіт**

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

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

***з дисципліни:*** «Основи програмування»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт до блоку № 6

**Виконав:**

Студент групи ШІ-11

Левченко Денис

**Тема:**

Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.

**Мета:**

Ознайомитися з динамічними структурами даних, такими як черга, стек, списки та дерево, зрозуміти їхні особливості та області застосування. Також необхідно навчитися основним алгоритмам обробки цих структур, зокрема додавання, видалення та пошуку елементів, з метою ефективного управління даними в програмах.

## **Теоретичні відомості:**

1. Основи Динамічних Структур Даних
2. Стек
3. Черга
4. Зв'язні Списки
5. Дерева
6. Алгоритми Обробки Динамічних Структур

## **Індивідуальний план опрацювання теорії:**

1. <https://www.youtube.com/watch?v=eSxLVD5vfqM>
2. <https://www.youtube.com/watch?v=jUJngLO_c_0>
3. <https://www.youtube.com/watch?v=Yhw8NbjrSFA>
4. <https://www.youtube.com/watch?v=-25REjF_atI>
5. <https://m.youtube.com/watch?v=qBFzNW0ALxQ>
6. <https://www.youtube.com/watch?v=999IE-6b7_s>

## **Виконання роботи:**

## **Завдання 1:** VNS Lab 10 - Task 1. Варіант - 16Написати 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. 16.Записи в лінійному списку містять ключове поле типу \*char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим ключем. Додати К елементів у кінець списку.

## **Завдання 2:** Algotester lab 5 Варіант 1

**Завдання 3:** Algotester lab 7 8 Варіант 3  
Ваше 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:** Class Practice Work *Умови задачі:*

- використовувати цілочисельні значення в списку;

- реалізувати метод реверсу;

- реалізувати допоміжний метод виведення вхідного і обернутого списків;

*Умови задачі:*

- використовувати цілочисельні значення в списку;

- реалізувати функцію, яка ітеративно проходиться по обох списках і порівнює дані в кожному вузлі;

- якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.

*Умови задачі:*

- використовувати цифри від 0 до 9 для значень у списку;

- реалізувати функцію, яка обчислює суму двох чисел, які збережено в списку; молодший розряд числа записано в голові списка (напр. 379 ⟹ 9→7→3);

- функція повертає новий список, передані в функцію списки не модифікуються.

*Умови задачі:*

- використовувати цілі числа для значень у вузлах дерева

- реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева

- функція повертає нове дерево, передане в функцію дерево не модифікується

*Умови задачі:*

- використовувати цілочисельні значення у вузлах дерева;

- реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів

- вузол-листок не змінює значення

- значення змінюються від листків до кореня дерева

**Завдання 5:** Self Practice Work  
Ваше завдання - власноруч реалізувати структуру даних "Динамічний масив".  
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.  
  
Вам будуть поступати запити такого типу:

* **Вставка**:  
  Ідентифікатор - insert  
  Ви отримуєте ціле число index елемента, на місце якого робити вставку.  
  Після цього в наступному рядку рядку написане число N - розмір масиву, який треба вставити.  
  У третьому рядку N цілих чисел - масив, який треба вставити на позицію index.
* **Видалення**:  
  Ідентифікатор - erase  
  Ви отримуєте 2 цілих числа - index, індекс елемента, з якого почати видалення та n - кількість елементів, яку треба видалити.
* **Визначення розміру**:  
  Ідентифікатор - size  
  Ви не отримуєте аргументів.  
  Ви виводите кількість елементів у динамічному масиві.
* **Визначення кількості зарезервованої пам’яті**:  
  Ідентифікатор - capacity  
  Ви не отримуєте аргументів.  
  Ви виводите кількість зарезервованої пам’яті у динамічному масиві.  
  Ваша реалізація динамічного масиву має мати фактор росту ([Growth factor](https://en.wikipedia.org/wiki/Dynamic_array" \l "Growth_factor)) рівний 2.
* **Отримання значення** i-го елементу  
  Ідентифікатор - get  
  Ви отримуєте ціле число - index, індекс елемента.  
  Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора []
* **Модифікація значення** i-го елементу  
  Ідентифікатор - set  
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перегрузку оператора []
* **Вивід динамічного масиву на екран**  
  Ідентифікатор - print  
  Ви не отримуєте аргументів.  
  Ви виводите усі елементи динамічного масиву через пробіл.  
  Реалізувати використовуючи перегрузку оператора <<

**Дизайн та планувальна оцінка часу виконання завдань:**

## **Завдання 1:** Запланований час виконання 1-1.5 години.

## **Завдання 2:** Запланований час виконання 1 година.

## **Завдання 3:** Запланований час виконання 1-2 години.

## **Завдання 4:** Запланований час виконання 2 години.

## **Завдання 5:**

## Запланований час виконання 2 години.

**Код програм з посиланням на зовнішні ресурси:  
Завдання 1:**

#include <iostream>

#include <fstream>

#include <cstring>

using namespace std;

struct *Node* {

    char key[100];

*Node*\* next;

*Node*\* prev;

};

struct *DoublyLinkedList* {

*Node*\* head;

*Node*\* tail;

    DoublyLinkedList() {

        head = nullptr;

        tail = nullptr;

    }

    void createEmptyList() {

        head = tail = nullptr;

    }

    void addElement(const char\* *key*) {

*Node*\* newNode = **new** *Node*;

        strcpy(newNode->key, *key*);

        newNode->next = nullptr;

        newNode->prev = tail;

        if (tail != nullptr) {

            tail->next = newNode;

        }

        tail = newNode;

        if (head == nullptr) {

            head = newNode;

        }

    }

    void deleteElement(const char\* *key*) {

*Node*\* temp = head;

        while (temp != nullptr) {

            if (strcmp(temp->key, *key*) == 0) {

                if (temp->prev != nullptr) {

                    temp->prev->next = temp->next;

                } else {

                    head = temp->next;

                }

                if (temp->next != nullptr) {

                    temp->next->prev = temp->prev;

                } else {

                    tail = temp->prev;

                }

**delete** temp;

                cout << "Елемент з ключем '" << *key* << "' видалено.\n";

                return;

            }

            temp = temp->next;

        }

        cout << "Елемент з ключем '" << *key* << "' не знайдений.\n";

    }

    void printList() {

        if (head == nullptr) {

            cout << "Список порожній.\n";

            return;

        }

*Node*\* temp = head;

        while (temp != nullptr) {

            cout << temp->key << " ";

            temp = temp->next;

        }

        cout << endl;

    }

    void saveToFile(const char\* *filename*) {

*ofstream* file(*filename*);

        if (!file) {

            cout << "Не вдалося відкрити файл для запису.\n";

            return;

        }

*Node*\* temp = head;

        while (temp != nullptr) {

            file << temp->key << endl;

            temp = temp->next;

        }

        file.close();

    }

    void loadFromFile(const char\* *filename*) {

*ifstream* file(*filename*);

        if (!file) {

            cout << "Не вдалося відкрити файл для читання.\n";

            return;

        }

        char key[100];

        while (file.getline(key, 100)) {

            addElement(key);

        }

        file.close();

    }

    void destroyList() {

*Node*\* temp = head;

        while (temp != nullptr) {

*Node*\* nextNode = temp->next;

**delete** temp;

            temp = nextNode;

        }

        head = tail = nullptr;

        cout << "Список знищено.\n";

    }

};

int main() {

*DoublyLinkedList* list;

    list.createEmptyList();

    list.addElement("element1");

    list.addElement("element2");

    list.addElement("element3");

    cout << "Список після додавання елементів:\n";

    list.printList();

    list.deleteElement("element2");

    cout << "Список після видалення елемента:\n";

    list.printList();

    list.addElement("element4");

    list.addElement("element5");

    cout << "Список після додавання нових елементів:\n";

    list.printList();

    list.saveToFile("list.txt");

    list.destroyList();

    list.loadFromFile("list.txt");

    cout << "Список після відновлення з файлу:\n";

    list.printList();

    list.destroyList();

    return 0;

}

## **Завдання 2:**

#include <iostream>

#include <vector>

#include <cstdint>

using namespace std;

void toggle(*uint64\_t* &*board*, int *row*, int *col*) {

    int position = *row* \* 8 + *col*;

*board* ^= (1ULL << position);

}

int main() {

*uint64\_t* board;

    int N;

    cin >> board;

    cin >> N;

    vector<pair<int, int>> spells(N);

    for (int i = 0; i < N; ++i) {

        cin >> spells[i].first >> spells[i].second;

        spells[i].first--;

        spells[i].second--;

    }

    for (const auto &spell : spells) {

        int row = spell.first;

        int col = spell.second;

        for (int c = 0; c < 8; ++c) {

            toggle(board, row, c);

        }

        for (int r = 0; r < 8; ++r) {

            if (r != row) {

                toggle(board, r, col);

            }

        }

    }

    cout << board << endl;

    return 0;

}

## **Завдання 3:**

#include <iostream>

using namespace std;

template <typename *T*>

struct TreeNode {

*T* value;

*TreeNode*\* left;

*TreeNode*\* right;

    TreeNode(*T* *val*) : value(*val*), left(nullptr), right(nullptr) {}

};

template <typename *T*>

class BinarySearchTree {

private:

    TreeNode<*T*>\* root;

    int tree\_size;

    void insert(TreeNode<*T*>\*& *node*, *T* *value*) {

        if (*node* == nullptr) {

*node* = **new** TreeNode<*T*>(*value*);

            ++tree\_size;

        } else if (*value* < *node*->value) {

            insert(*node*->left, *value*);

        } else if (*value* > *node*->value) {

            insert(*node*->right, *value*);

        }

    }

    bool contains(TreeNode<*T*>\* *node*, *T* *value*) const {

        if (*node* == nullptr) return false;

        if (*value* == *node*->value) return true;

        if (*value* < *node*->value) return contains(*node*->left, *value*);

        return contains(*node*->right, *value*);

    }

    void print(TreeNode<*T*>\* *node*) const {

        if (*node* == nullptr) return;

        print(*node*->left);

        cout << *node*->value << " ";

        print(*node*->right);

    }

    void clear(TreeNode<*T*>\* *node*) {

        if (*node* == nullptr) return;

        clear(*node*->left);

        clear(*node*->right);

        delete *node*;

    }

public:

    BinarySearchTree() : root(nullptr), tree\_size(0) {}

    ~BinarySearchTree() { clear(root); }

    void insert(*T* *value*) { insert(root, *value*); }

    bool contains(*T* *value*) const { return contains(root, *value*); }

    int size() const { return tree\_size; }

    void print() const { print(root); cout << endl; }

};

int main() {

    BinarySearchTree<int> bst;

    int Q;

    cin >> Q;

    for (int i = 0; i < Q; ++i) {

*string* command;

        cin >> command;

        if (command == "insert") {

            int value;

            cin >> value;

            bst.insert(value);

        } else if (command == "contains") {

            int value;

            cin >> value;

            cout << (bst.contains(value) ? "Yes" : "No") << endl;

        } else if (command == "size") {

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

        } else if (command == "print") {

            bst.print();

        }

    }

    return 0;

}

**Завдання 4:**

#include <iostream>

using namespace std;

struct *Node*{

    int data;

*Node*\* next;

    Node(int *data*, *Node*\* *next*) : data(*data*), next(*next*) { }

    Node() : next(nullptr) {}

};

struct *TreeNode* {

    int data;

*TreeNode*\* left;

*TreeNode*\* right;

    TreeNode(int *data*, *TreeNode*\* *left* = nullptr, *TreeNode*\* *right* = nullptr)

        : data(*data*), left(*left*), right(*right*) {}

};

*Node*\* reverse(*Node* \**head*)

{

*Node*\* tmp = *head*;

*Node*\* tmpCopy = nullptr;

*Node*\* prev = nullptr;

    while(tmp != nullptr)

    {

        tmpCopy = tmp->next; // head = 1 : tmpCopy = 4

        tmp->next = prev; // 1 -> nullptr

        prev = tmp;

        tmp = tmpCopy;

    }

    return prev;

}

void printList(*Node*\* *head*) {

    while (*head* != nullptr) {

        cout << *head*->data << " ";

*head* = *head*->next;

    }

    cout << "\n";

}

bool compare(*Node* \**h1*, *Node* \**h2*)

{

    unsigned int size1 = 0, size2= 0;

*Node*\* h1Copy = *h1*, \*h2Copy = *h2*;

    while(h1Copy)

    {

        size1++;

        h1Copy = h1Copy->next;

    }

    while(h2Copy)

    {

        size2++;

        h2Copy = h2Copy->next;

    }

    if(size1 != size2) { return false; }

    else

    {

        while(*h1* && *h2*)

        {

            if(*h1*->data != *h2*->data) return false;

*h1* = *h1*->next;

*h2* = *h2*->next;

        }

    }

    return true;

}

*Node*\* add(*Node* \**head1*, *Node* \**head2*)

{

*Node*\* reversehead1 = reverse(*head1*);

*Node*\* reversehead2 = reverse(*head2*);

    int numb1 = 0 , numb2 = 0;

*Node*\* reversehead1Copy = reversehead1;

*Node*\* reversehead2Copy = reversehead2;

    while(reversehead1Copy)

    {

        numb1 = numb1 \* 10 + reversehead1Copy->data;

        reversehead1Copy = reversehead1Copy->next;

    }

    while(reversehead2Copy)

    {

        numb2 = numb2 \* 10 + reversehead2Copy->data;

        reversehead2Copy = reversehead2Copy->next;

    }

    int sum = numb1 + numb2;

*Node*\* newhead = nullptr;

*Node*\* current = nullptr;

    do {

        int digit = sum % 10;

*Node*\* newNode = **new** *Node*(digit, nullptr);

        if (newhead == nullptr) {

            newhead = newNode;

            current = newhead;

        } else {

            current->next = newNode;

            current = current->next;

        }

        sum /= 10;

    } while (sum > 0);

    return newhead;

}

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

    if (*root* == nullptr) {

        return nullptr;

    }

*TreeNode*\* newValue = **new** *TreeNode*(*root*->data);

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

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

    return newValue;

}

void printTree(*TreeNode*\* *root*, int *level* = 0) {

    if (*root* == nullptr) return;

    printTree(*root*->right, *level* + 1);

    cout << *string*(*level* \* 4, ' ') << *root*->data << endl;

    printTree(*root*->left, *level* + 1);

}

*TreeNode*\* tree\_sum(*TreeNode*\* *root*) {

    if (*root* == nullptr) {

        return nullptr;

    }

*TreeNode*\* leftSumTree = tree\_sum(*root*->left);

*TreeNode*\* rightSumTree = tree\_sum(*root*->right);

    int leftSum = (leftSumTree != nullptr) ? leftSumTree->data : 0;

    int rightSum = (rightSumTree != nullptr) ? rightSumTree->data : 0;

*TreeNode*\* newNode = **new** *TreeNode*(*root*->data + leftSum + rightSum, leftSumTree, rightSumTree);

    return newNode;

}

void deleteList(*Node*\* *head*) {

    while (*head* != nullptr) {

*Node*\* temp = *head*;

*head* = *head*->next;

**delete** temp;

    }

}

void deleteTree(*TreeNode*\* *root*) {

    if (*root* == nullptr) return;

    deleteTree(*root*->left);

    deleteTree(*root*->right);

**delete** *root*;

}

int main(){

    // Node\* head1 =  new Node(9,new Node(9,new Node(9,nullptr)));

    // Node \*copy1 = head1;

    // Node\* head2 =  new Node(1, nullptr);

    // cout << "Listт 1: ";

    // printList(head1);

    // cout << endl;

    // cout << "Listт 2: ";

    // printList(head2);

    // cout << endl;

    //  boolalpha(cout);

    // cout << "Is equal: " << compare(head1,head2);

    // cout << endl;

    // cout << "Summ of 2 listd equal: " << endl;

    // printList(add(head1,head2));

    //  cout << endl;

    // Node\* reversedList = reverse(copy1);

    // cout << "Reversed List: ";

    // printList(reverse(copy1));

    // cout << endl;

*Node*\* head = **new** *Node*(1,nullptr);

    head->next = **new** *Node*(2,nullptr);

    head->next->next = **new** *Node*(3,nullptr);

    head->next->next->next = **new** *Node*(4,nullptr);

    head->next->next->next->next = **new** *Node*(5,nullptr);

    cout << "List:" << endl;

    printList(head);

    head = reverse(head);

    cout << "Reversed list:" << endl;

    printList(head);

*Node*\* list1 = **new** *Node*(1,nullptr);

    list1->next = **new** *Node*(2,nullptr);

    list1->next->next = **new** *Node*(3,nullptr);

*Node*\* list2 = **new** *Node*(1,nullptr);

    list2->next = **new** *Node*(2,nullptr);

    list2->next->next = **new** *Node*(3,nullptr);

    cout << (compare(list1, list2) ? "Lists equal" : "Lists not equal") << endl;

*Node*\* num1 = **new** *Node*(9,nullptr);

    num1->next = **new** *Node*(9,nullptr);

    num1->next->next = **new** *Node*(9,nullptr);

*Node*\* num2 = **new** *Node*(1,nullptr);

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

    cout << "Summ: " << endl;

    sum = reverse(sum);

    printList(sum);

*TreeNode*\* root = **new** *TreeNode*(1, nullptr, nullptr);

    root->left = **new** *TreeNode*(2, nullptr, nullptr);

    root->right = **new** *TreeNode*(3, nullptr, nullptr);

    root->left->left = **new** *TreeNode*(4, nullptr, nullptr);

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

    root->right->left = **new** *TreeNode*(8, nullptr, nullptr);

    root->right->right = **new** *TreeNode*(4, nullptr, nullptr);

    cout << "Tree in vertical format" << endl;

    printTree(root);

    cout << "Reversed tree in vertical format" << endl;

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

    printTree(mirroredRoot);

    cout << "Tree sum:" << endl;

*TreeNode*\* sumTree = tree\_sum(root);

    printTree(sumTree);

    return 0;

}

## **Завдання 5:**

#include <iostream>

using namespace std;

template <class *T*>

class SDA

{

private:

*T* \*arr;

public:

    int size;

    int capacity;

    SDA()

    {

*this*->size = 0;

*this*->capacity = 1;

*this*->arr = **new** *T*[1];

    }

    void insert(int *index*, int *amount*, *T* \**toInsert*)

    {

        while (size + *amount* >= capacity)

            capacity \*= 2;

*T* \*temp = **new** *T*[capacity];

        for (int i = 0; i < *index*; i++)

            temp[i] = arr[i];

        for (int i = 0; i < *amount*; i++)

            temp[*index* + i] = *toInsert*[i];

        for (int i = *index*; i < size; i++)

            temp[i + *amount*] = arr[i];

*this*->size += *amount*;

        delete[] arr;

        arr = temp;

    }

    void erase(int *index*, int *amount*)

    {

*T* \*temp = **new** *T*[capacity];

        int acc = 0;

        for (int i = 0; i < *this*->size; i++)

        {

            if (i < *index* || i >= *index* + *amount*)

            {

                temp[acc] = arr[i];

                acc++;

            }

        }

*this*->size -= *amount*;

            delete[] arr;

            arr = temp;

    }

*T* get(int *index*)

    {

        return *this*->arr[*index*];

    }

    void set(int *index*, *T* *value*)

    {

*this*->arr[*index*] = *value*;

    }

    void operator<<(*string* *separator*)

    {

        for (int i = 0; i < *this*->size; i++)

        {

            cout << arr[i] << *separator*;

        }

        cout << endl;

    }

};

int main()

{

    SDA<int> arr;

    int q;

    cin >> q;

    while (q--)

    {

*string* line;

        cin >> line;

        if (line == "insert")

        {

            int index, N;

            cin >> index >> N;

            cin.ignore();

            int \*temp = **new** int[N];

            for (int i = 0; i < N; i++)

            {

                int n;

                cin >> n;

                temp[i] = n;

            }

            arr.insert(index, N, temp);

**delete[]** temp;

        }

        else if (line == "erase")

        {

            int index, N;

            cin >> index >> N;

            arr.erase(index, N);

        }

        else if (line == "size")

        {

            cout << arr.size << endl;

        }

        else if (line == "capacity")

        {

            cout << arr.capacity << endl;

        }

        else if (line == "get")

        {

            int i;

            cin >> i;

            cout << arr.get(i) << endl;

        }

        else if (line == "set")

        {

            int i, value;

            cin >> i >> value;

            arr.set(i, value);

        }

        else if (line == "print")

        {

            arr << " ";

        }

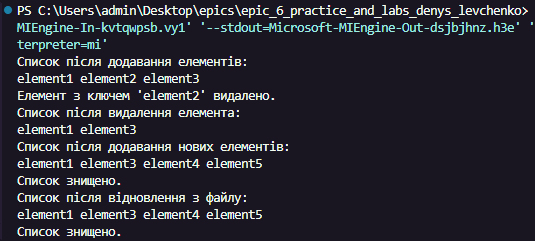
    }

    return 0;

}

## **Результат виконання завдань, тестування та фактично витрачений час:**

**Завдання 1:**

Фактично витрачений час: 2 години.

## **Завдання 2:**

## Фактично витрачений час: 1.5 години.

## **Завдання 3:**

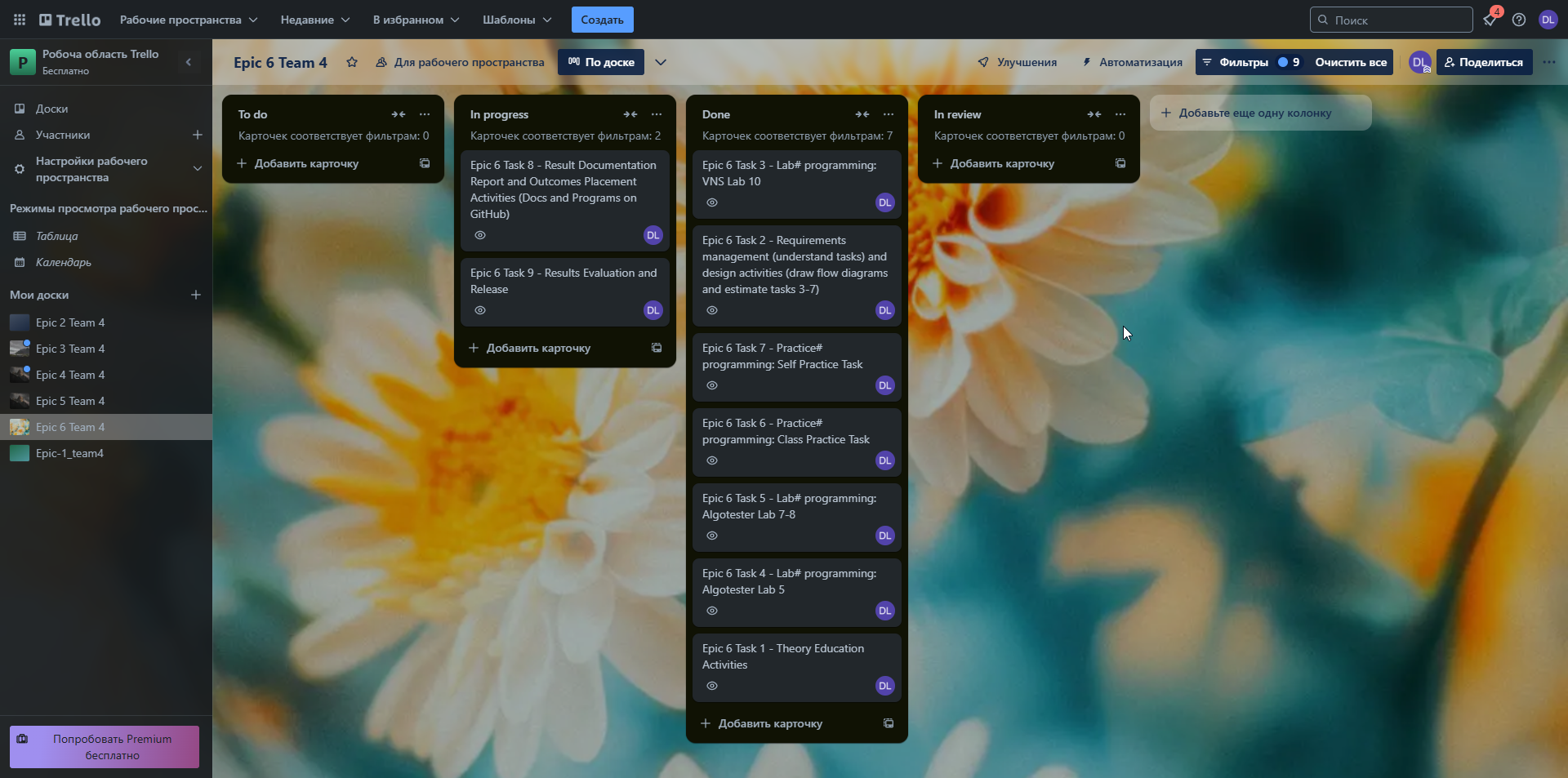
## Фактично витрачений час: 4 години.

## **Завдання 4:** Фактично витрачений час: 1.5 години.

## **Завдання 5:**

## Фактично витрачений час: 20 хвилин.

## **Зустріч з комадою та trello:**



**Висновок:** У цій лабораторній роботі я навчився працювати з динамічними структурами даних — чергою, стеком, списками та деревами, а також застосовувати алгоритми для їхньої обробки. Це дало мені розуміння ефективного управління даними в програмах.