Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

A blue and white logo

Description automatically generated

**Звіт**

**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

**Виконав:**

Студент групи ШІ-12

Михальчук Антон Євгенійович

# **Тема роботи:**

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

# **Мета роботи:**

# Мета роботи полягає в опануванні основних понять і принципів роботи з динамічними структурами даних, розумінні їхнього застосування та особливостей реалізації. Динамічні структури, такі як черга, стек, списки та дерева, є важливими інструментами для ефективного зберігання, обробки та управління даними, особливо коли їх кількість чи розмір змінюються під час виконання програми.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №\*.1: С++ Data structures

1. Індивідуальний план опрацювання теорії:

* Тема №\*.1: С++ Basics
  + Джерела Інформації
    - Відео. <https://www.youtube.com/watch?v=2UDMGCcRCjo>
    - Стаття. <https://www.w3schools.com/cpp/>
  + Що опрацьовано:
    - Вивчив базовий синтаксис та семантику мови C++.
    - Особливу увагу приділяв таким темам, як динамічні структури.
  + Статус: Ознайомлений
  + Початок опрацювання теми: 15.09.2024
  + Звершення опрацювання теми: 14.11.2024

# **Виконання роботи:**

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

Завдання №1 VNS Lab 10 Варіант: 9

* Деталі завдання:

Записи в лінійному списку містять ключове поле типу int. Сформувати

двонаправлений список. Знищити з нього К елементів перед елементом із

заданим номером, додати К елементів у кінець списку.

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.

Порядок виконання роботи

1. Написати функцію для створення списку. Функція може створювати

порожній список, а потім додавати в нього елементи.

2. Написати функцію для друку списку. Функція повинна передбачати вивід

повідомлення, якщо список порожній.

3. Написати функції для знищення й додавання елементів списку у

відповідності зі своїм варіантом.

4. Виконати зміни в списку й друк списку після кожної зміни.

5. Написати функцію для запису списку у файл.

6. Написати функцію для знищення списку.

7. Записати список у файл, знищити його й виконати друк (при друці повинне

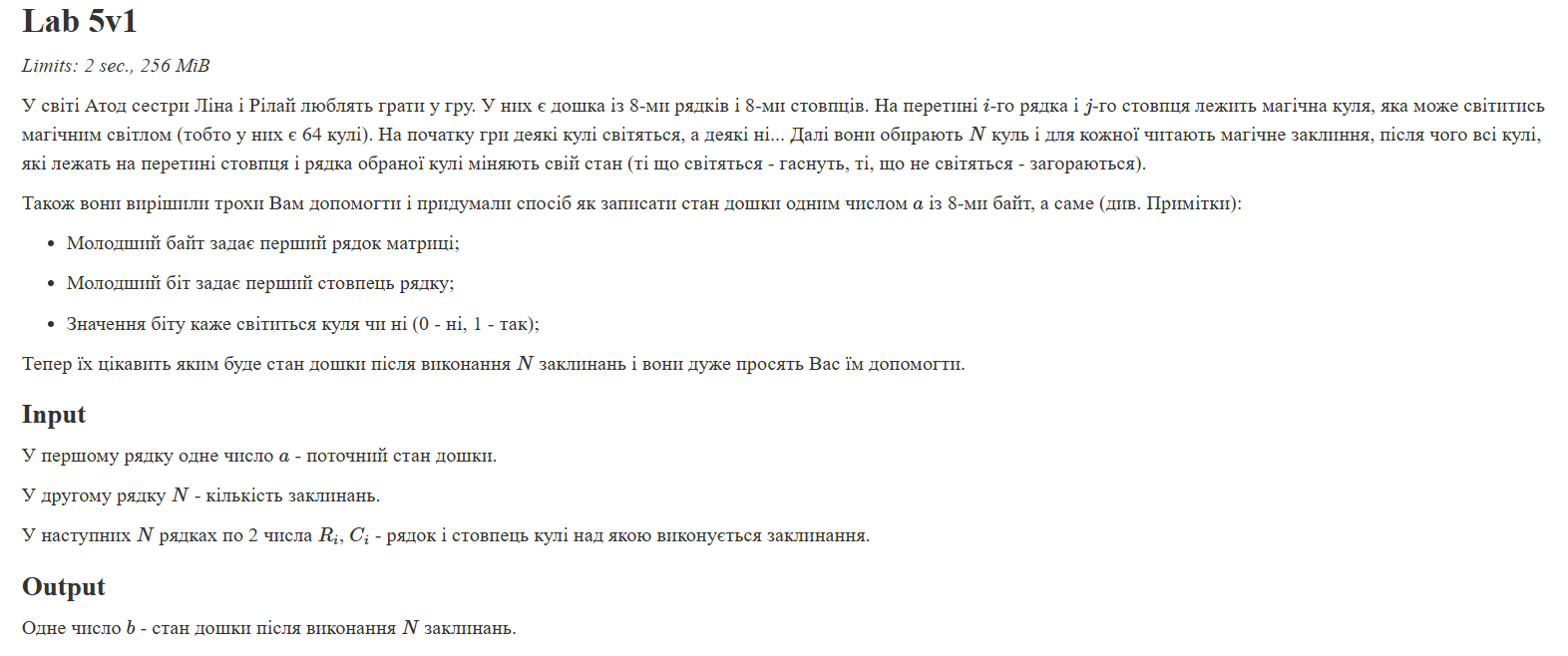
бути видане повідомлення "Список порожній").

8. Написати функцію для відновлення списку з файлу.

9. Відновити список і роздрукувати його.

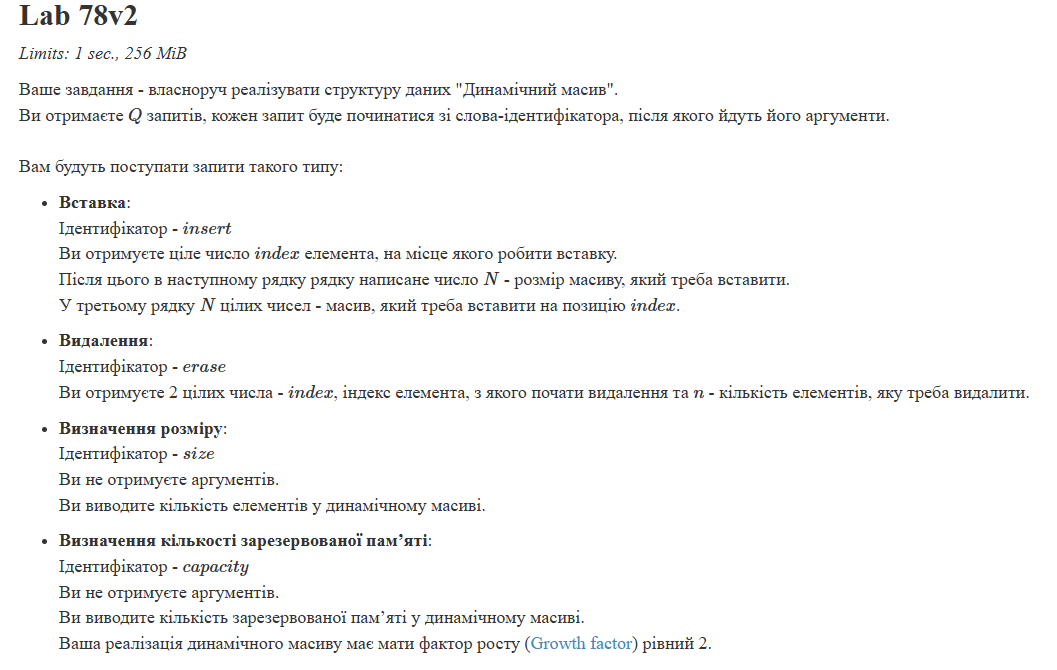
10.Знищити список.

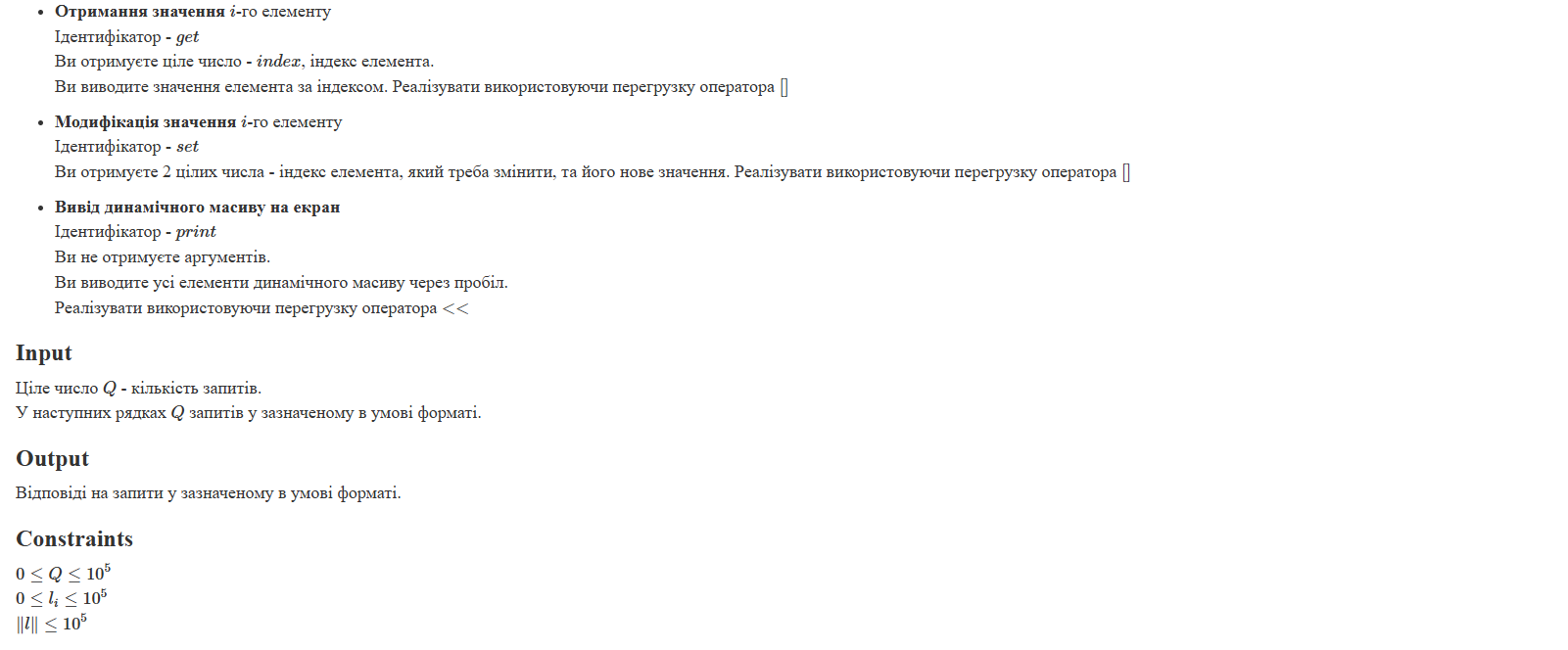
Завдання №2 Algotester Lab 5 Варіант: 1



Завдання №3 Algotester Lab 7-8 Варіант: 2

* Деталі завдання:





Гарантується, що усі дані коректні. Виходу за межі масиву або розмір, більший ніж розмір масиву недопустимі.  
Індекси починаються з нуля.  
**Для того щоб отримати**50%50%**балів за лабораторну достатньо написати свою структуру.  
Для отримання**100%100%**балів ця структура має бути написана як**[**шаблон класу**](https://cplusplus.com/doc/oldtutorial/templates/)**, у якості параметру використати**intint**.  
Використовувати STL заборонено.**

Завдання №4 Class practice Task

* Деталі завдання:

## Задача №1 - Реверс списку (Reverse list)

Реалізувати метод реверсу списку: Node\* reverse(Node \*head);

Умови задачі:

-       використовувати цілочисельні значення в списку;

-       реалізувати метод реверсу;

## -       реалізувати допоміжний метод виведення вхідного і обернутого списків;

Задача №2 - Порівняння списків

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

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.

Задача №3 – Додавання великих чисел

Node\* add(Node \*n1, Node \*n2);

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.

Задача №4 - Віддзеркалення дерева

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

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

Задача №5 - Записати кожному батьківському вузлу суму підвузлів

void tree\_sum(TreeNode \*root);

Умови задачі:

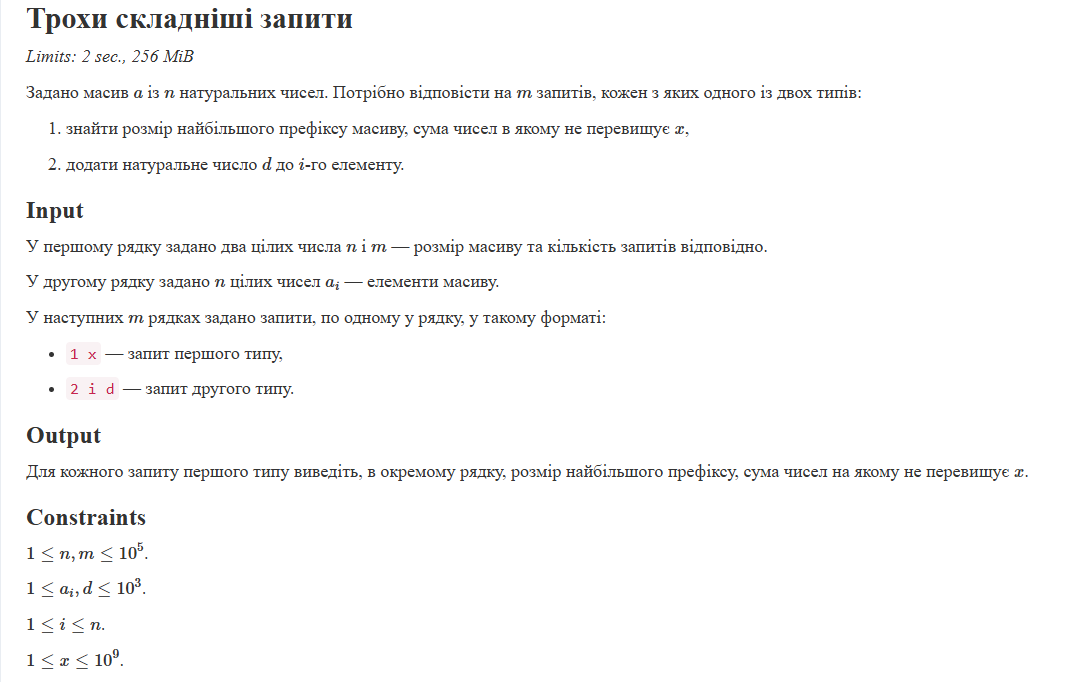
- використовувати цілочисельні значення у вузлах дерева;

- 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 Task Трохи складніші запити



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

Програма №1 VNS Lab 10 Варіант: 9

* Планований час на реалізацію: 1 год.

Програма №2 Algotester Lab 5 Варіант: 1

* Планований час на реалізацію: 1 год.

Програма №3 VNS Algotester Lab 7-8 Варіант: 1

* Планований час на реалізацію: 2 год.

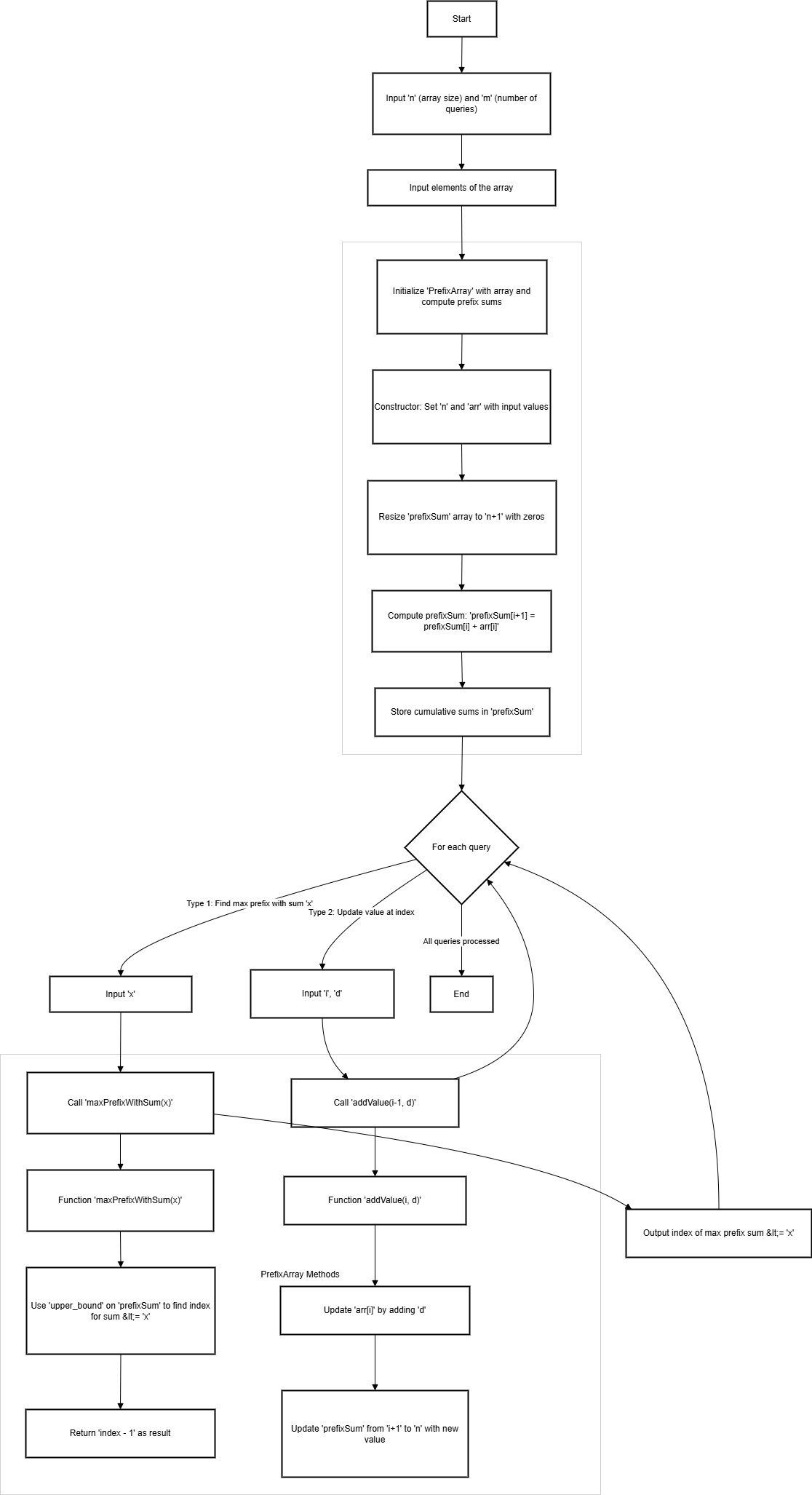
Програма №3.2 VNS Algotester Lab 7-8 Варіант: 1

* Планований час на реалізацію: 10 хв.

Програма №4 Class Practice Task

* Планований час на реалізацію: 5 год.

Програма №5 Self Practice Task Щасливий результат

* Планований час на реалізацію: 10 хв
* Блок схема:
* 

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

Завдання №1

<https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/303/files>

#include <iostream>

#include <fstream>

#include <cstdlib>

#include <ctime>

#include <vector>

using namespace std;

struct Node

{

    int data;

    Node \*next;

    Node \*prev;

    Node(int value)

    {

        data = value;

        next = nullptr;

        prev = nullptr;

    }

};

struct DoublyLinkedList

{

private:

    Node \*head;

    Node \*tail;

    int size;

public:

    DoublyLinkedList()

    {

        head = nullptr;

        tail = nullptr;

        size = 0;

    }

    ~DoublyLinkedList()

    {

        while (head)

        {

            Node \*temp = head;

            head = head->next;

            delete temp;

        }

    }

    bool isEmpty()

    {

        return head == nullptr;

    }

    void print()

    {

        if (!tail)

            cout << "List is empty" << endl;

        else

        {

            Node \*current = head;

            while (current)

            {

                cout << current->data << " ";

                current = current->next;

            }

            cout << endl;

        }

    }

    void insertToEnd(vector<int> values)

    {

        for (int i = 0; i < values.size(); i++)

        {

            Node \*newNode = new Node(values[i]);

            if (tail == nullptr)

            {

                head = newNode;

                tail = newNode;

            }

            else

            {

                tail->next = newNode;

                newNode->prev = tail;

                tail = newNode;

            }

            size++;

        }

    }

    void eraseBefore(int index, int k)

    {

        if (index - k <= 0)

            return;

        Node \*left = head;

        for (int i = 0; i < index - k - 1; i++)

        {

            left = left->next;

        }

        Node \*toDelete = left->next;

        Node \*newLeft = left;

        for (int i = 0; i < k; i++)

        {

            Node \*tempNode = toDelete;

            toDelete = toDelete->next;

            delete tempNode;

        }

        newLeft->next = toDelete;

        toDelete->prev = newLeft;

    }

    void writeInFile()

    {

        ofstream file("lab10.txt");

        if (file.is\_open())

        {

            Node \*current = head;

            while (current)

            {

                file << current->data << " ";

                current = current->next;

            }

            file << endl;

            file.close();

            cout << "List saved to file 'lab10.txt'." << endl;

        }

        else

        {

            cout << "Failed to open file for writing." << endl;

        }

    }

    void deleteList()

    {

        while (head)

        {

            Node \*temp = head;

            head = head->next;

            delete temp;

        }

        tail = nullptr;

        cout << "List deleted." << endl;

    }

    void updateFromFile()

    {

        vector<int> fromFile;

        ifstream file("lab10.txt");

        if (file.is\_open())

        {

            int number;

            while (file >> number)

            {

                fromFile.push\_back(number);

            }

            cout << "List updated from file 'lab10.txt'." << endl;

        }

        else

        {

            cout << "Failed to open file for reading." << endl;

        }

        file.close();

        insertToEnd(fromFile);

    }

};

int main()

{

    DoublyLinkedList list;

    int k, indel;

    vector<int> initVec, addVec;

    cout << "Initialized list: " << endl;

    list.print();

    srand(time(0));

    for (int i = 0; i < 10; ++i)

    {

        int randomNumber = rand() % 100 + 1;

        initVec.push\_back(randomNumber);

    }

    list.insertToEnd(initVec);

    cout << "Filled the list with random numbers: " << endl;

    list.print();

    cout << "Enter k: ";

    cin >> k;

    cout << "Enter index of element before which you want to delete elements: ";

    cin >> indel;

    list.eraseBefore(indel, k);

    cout << "List after deletion " << k << " elements before " << indel << "th. element:" << endl;

    list.print();

    for (int i = 0; i < k; ++i)

    {

        int randomNumber = rand() % 100 + 1;

        addVec.push\_back(randomNumber);

    }

    list.insertToEnd(addVec);

    cout << "List after appending " << k << " elements to the end:" << endl;

    list.print();

    list.writeInFile();

    list.deleteList();

    cout << "Deletion of list: ";

    list.print();

    list.updateFromFile();

    list.print();

    list.deleteList();

    return 0;

}

Завдання №2

<https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/303/files>

#include <iostream>

#include <vector>

using namespace std;

void toggleRow(uint64\_t &board, int row) {

    for (int col = 0; col < 8; col++) {

        board ^= (1ULL << (row \* 8 + col));

    }

}

void toggleColumn(uint64\_t &board, int col) {

    for (int row = 0; row < 8; row++) {

        board ^= (1ULL << (row \* 8 + col));

    }

}

int main() {

    uint64\_t board;

    int n;

    cin >> board >> n;

    for (int i = 0; i < n; i++) {

        int r, c;

        cin >> r >> c;

        r--;

        c--;

        toggleRow(board, r);

        toggleColumn(board, c);

        board ^= (1ULL << (r \* 8 + c));

    }

    cout << board << endl;

    return 0;

}

Завдання №3.1

<https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/303/files>

#include <iostream>

#include <string>

using namespace std;

struct DynamicArray{

private:

    int\* data;

    int size;

    int capacity;

    void resize(int newCapacity){

        int\* newData = new int[newCapacity];

        for(int i = 0; i < size; i++){

            newData[i] = data[i];

        }

        delete[] data;

        data = newData;

        capacity = newCapacity;

    }

public:

    DynamicArray() : size(0), capacity(1){

        data = new int[capacity];

    }

    ~DynamicArray(){

        delete[] data;

    }

    void insert(int index, int N, int\* arr){

    if (size + N > capacity) {

        while (capacity <= size + N) {

            capacity \*= 2;

        }

        resize(capacity);

    }

        for(int i = size - 1; i >= index; i--){

            data[i + N] = data[i];

        }

        for(int i = 0; i < N; i++){

            data[index + i] = arr[i];

        }

        size += N;

        if(size == capacity){

            capacity \*= 2;

            resize(capacity);

        }

    }

    void erase(int index, int n){

        for(int i = index; i < size - n + n; i++){

            data[i] = data[i+n];

        }

        size -= n;

    }

    int getSize(){

        return size;

    }

    int getCapacity(){

        return capacity;

    }

    int& operator[](int index) { return data[index]; }

    friend ostream& operator<<(ostream& os, const DynamicArray& arr){

        for (int i = 0; i < arr.size; i++) {

            os << arr.data[i] << ' ';

        }

        return os;

    }

};

int main(){

DynamicArray arr;

    int Q;

    cin >> Q;

    for (int i = 0; i < Q; i++) {

        string command;

        cin >> command;

        if (command == "size") {

            cout << arr.getSize() << endl;

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

            cout << arr.getCapacity() << endl;

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

            int index, N;

            cin >> index >> N;

            int\* elements = new int[N];

            for (int j = 0; j < N; j++) {

                cin >> elements[j];

            }

            arr.insert(index, N, elements);

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

            int index, n;

            cin >> index >> n;

            arr.erase(index, n);

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

            int index;

            cin >> index;

            cout << arr[index] << endl;

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

            int index;

            int value;

            cin >> index >> value;

            arr[index] = value;

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

            cout << arr << endl;

        }

    }

    return 0;

}

Завдання №3.2

<https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/303/files>

#include <iostream>

#include <string>

using namespace std;

template<typename T = int>

class DynamicArray{

private:

    T\* data;

    int size;

    int capacity;

    void resize(int newCapacity){

        T\* newData = new T[newCapacity];

        for(int i = 0; i < size; i++){

            newData[i] = data[i];

        }

        delete[] data;

        data = newData;

        capacity = newCapacity;

    }

public:

    DynamicArray() : size(0), capacity(1){

        data = new T[capacity];

    }

    ~DynamicArray(){

        delete[] data;

    }

    void insert(int index, int N, T\* arr){

    if (size + N > capacity) {

        while (capacity <= size + N) {

            capacity \*= 2;

        }

        resize(capacity);

    }

        for(int i = size - 1; i >= index; i--){

            data[i + N] = data[i];

        }

        for(int i = 0; i < N; i++){

            data[index + i] = arr[i];

        }

        size += N;

        if(size == capacity){

            capacity \*= 2;

            resize(capacity);

        }

    }

    void erase(int index, int n){

        for(int i = index; i < size - n + n; i++){

            data[i] = data[i+n];

        }

        size -= n;

    }

    int getSize(){

        return size;

    }

    int getCapacity(){

        return capacity;

    }

    T& operator[](int index) { return data[index]; }

    friend ostream& operator<<(ostream& os, const DynamicArray& arr){

        for (int i = 0; i < arr.size; i++) {

            os << arr.data[i] << ' ';

        }

        return os;

    }

};

int main(){

DynamicArray arr;

    int Q;

    cin >> Q;

    for (int i = 0; i < Q; i++) {

        string command;

        cin >> command;

        if (command == "size") {

            cout << arr.getSize() << endl;

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

            cout << arr.getCapacity() << endl;

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

            int index, N;

            cin >> index >> N;

            int\* elements = new int[N];

            for (int j = 0; j < N; j++) {

                cin >> elements[j];

            }

            arr.insert(index, N, elements);

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

            int index, n;

            cin >> index >> n;

            arr.erase(index, n);

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

            int index;

            cin >> index;

            cout << arr[index] << endl;

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

            int index;

            int value;

            cin >> index >> value;

            arr[index] = value;

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

            cout << arr << endl;

        }

    }

    return 0;

}

Завдання №4

<https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/303/files>

#include <iostream>

#include <string>

#include <vector>

#include <ctime>

using namespace std;

struct Node {

    int data;

    Node\* next;

    Node(int value) {

        data = value;

        next = nullptr;

    }

};

void printLL(Node\* head) {

    Node\* current = head;

    while (current) {

        cout << current->data << " ";

        current = current->next;

    }

    cout << endl;

}

Node\* insert(Node\* head, const vector<int>& values) {

    Node\* newNode = new Node(values[0]);

    head = newNode;

    Node\* current = head;

    for (int i = 1; i < values.size(); i++) {

        Node\* newNode = new Node(values[i]);

        current->next = newNode;

        current = newNode;

    }

    return head;

}

Node\* reverse(Node\* head) {

    Node\* prev = nullptr;

    Node\* current = head;

    Node\* next = nullptr;

    while (current != nullptr) {

        next = current->next;

        current->next = prev;

        prev = current;

        current = next;

    }

    return prev;

}

bool compare(Node\* head1, Node\* head2) {

    while (head1 && head2) {

        if (head1->data != head2->data) return false;

        head1 = head1->next;

        head2 = head2->next;

    }

    return head1 == nullptr && head2 == nullptr;

}

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

    Node\* result = new Node(0);

    Node\* current1 = head1;

    Node\* current2 = head2;

    Node\* currentr = result;

    int sum, carry = 0;

    while (current1 != nullptr || current2 != nullptr || carry != 0) {

        sum = carry;

        if (current1 != nullptr) {

            sum += current1->data;

            current1 = current1->next;

        }

        if (current2 != nullptr) {

            sum += current2->data;

            current2 = current2->next;

        }

        carry = sum / 10;

        currentr->next = new Node(sum % 10);

        currentr = currentr->next;

    }

    return result->next;

}

struct TreeNode {

    int data;

    TreeNode\* left;

    TreeNode\* right;

    TreeNode(int value) {

        data = value;

        left = nullptr;

        right = nullptr;

    }

};

TreeNode\* createMirrorFlip(TreeNode\* root) {

    if (root == nullptr) {

        return nullptr;

    }

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

    newNode->left = createMirrorFlip(root->right);

    newNode->right = createMirrorFlip(root->left);

    return newNode;

}

void printTree(TreeNode\* root) {

    if (root == nullptr) {

        return;

    }

    printTree(root->left);

    cout << root->data << " ";

    printTree(root->right);

}

int treeSum(TreeNode\* root) {

    if (root == nullptr) {

        return 0;

    }

    if (root->left == nullptr && root->right == nullptr) {

        return root->data;

    }

    int leftSum = treeSum(root->left);

    int rightSum = treeSum(root->right);

    root->data = leftSum + rightSum;

    return root->data;

}

int main() {

    Node\* list = nullptr;

    vector<int> initVec;

    srand(time(0));

    for (int i = 0; i < 10; ++i) {

        int randomNumber = rand() % 100 + 1;

        initVec.push\_back(randomNumber);

    }

// FIRST TASK

    cout << "First Task" << endl;

    list = insert(list, initVec);

    cout << "Original List: ";

    printLL(list);

    Node\* originalList = insert(nullptr, initVec);

    Node\* reversedList = reverse(list);

    cout << "Reversed List: ";

    printLL(reversedList);

// SECOND TASK

    cout << endl << "Second Task" << endl;

    if (compare(originalList, originalList)) {

        printLL(originalList);

        printLL(originalList);

        cout << "Lists are identical";

    }

    else {

        printLL(originalList);

        printLL(originalList);

        cout << "Lists are different";

    }

    cout << endl << endl;

    if (compare(originalList, reversedList)) {

        printLL(originalList);

        printLL(reversedList);

        cout << "Lists are identical";

    }

    else {

        printLL(originalList);

        printLL(reversedList);

        cout << "Lists are different";

    }

// THIRD TASK

    cout << endl << endl << "Third Task" << endl;

    vector<int> num1 = {6, 7, 5, 1, 2};

    vector<int> num2 = {5, 6, 2};

    Node\* list1 = insert(list1, num1);

    Node\* list2 = insert(list2, num2);

    cout << "Number 1: ";

    printLL(list1);

    cout << "Number 2: ";

    printLL(list2);

    Node\* result = add(list1, list2);

    cout << "Sum: ";

    printLL(result);

// FORTH TASK

    cout << endl << "Forth Task" << endl;

    TreeNode\* root = new TreeNode(10);

    root->left = new TreeNode(5);

    root->right = new TreeNode(15);

    root->left->left = new TreeNode(3);

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

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

    root->right->right = new TreeNode(2);

    TreeNode\* mirrorRoot = createMirrorFlip(root);

    cout << "Original tree (in order): ";

    printTree(root);

    cout << endl;

    cout << "Mirror tree (in order): ";

    printTree(mirrorRoot);

    cout << endl;

// FIFTH TASK

    cout << endl << "Fifth Task" << endl;

    cout << "Original tree: ";

    printTree(root);

    cout << endl;

    treeSum(root);

    cout << "Tree after updating node values with subtree sums: ";

    printTree(root);

    cout << endl;

    return 0;

}

Завдання №5

<https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/303/files>

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct PrefixArray {

    vector<int> arr;

    vector<int> prefixSum;

    int n;

    PrefixArray(int size, const vector<int>& elements) : n(size), arr(elements) {

        prefixSum.resize(n + 1, 0);

        for (int i = 0; i < n; ++i) {

            prefixSum[i + 1] = prefixSum[i] + arr[i];

        }

    }

    int maxPrefixWithSum(int x) {

        int idx = upper\_bound(prefixSum.begin(), prefixSum.end(), x) - prefixSum.begin() - 1;

        return idx;

    }

    void addValue(int index, int value) {

        arr[index] += value;

        for (int j = index + 1; j <= n; ++j) {

            prefixSum[j] += value;

        }

    }

};

int main() {

    int n, m;

    cin >> n >> m;

    vector<int> elements(n);

    for (int i = 0; i < n; ++i) {

        cin >> elements[i];

    }

    PrefixArray prefixArray(n, elements);

    for (int q = 0; q < m; ++q) {

        int type;

        cin >> type;

        if (type == 1) {

            int x;

            cin >> x;

            cout << prefixArray.maxPrefixWithSum(x) << endl;

        } else if (type == 2) {

            int i, d;

            cin >> i >> d;

            prefixArray.addValue(i - 1, d);

        }

    }

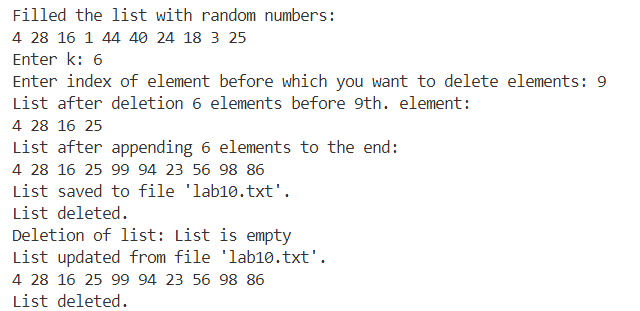
    return 0;

}

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

Завдання №1

Створив дві структури: Node та DoublyLinkedList. Для структури двонаправленого списку створив конструктор і деструктор. Також методи print(), insertToEnd(), eraseBefore(), writeInFile(), deleteList(), updateFromFile()



Час затрачений на виконання завдання: 5 год.

Завдання №2

Число board є не більшим за 264 , тому потрібно використовувати unsigned long long.

Найважливішим моментом розв’язку є цей рядок у функція перемикання рядків і стовпців: board ^= (1ULL << (row \* 8 + col));

Побітовий зсув створює число заповнене нулями, окрім біта відповідному до координат закляття. Далі операція суворої диз’юнкції перемикає потрібний біт за тією логікою, що якщо він дорівнював 1, то 1 ^ 1 = 0, а якщо 0, то 0 ^ 1 = 1.



Час затрачений на виконання завдання: 1 год.

Завдання №3 1 і 2

Особливої різниці між тим, що динамічний масив буде класом, чи структурою при написанні коду не має.

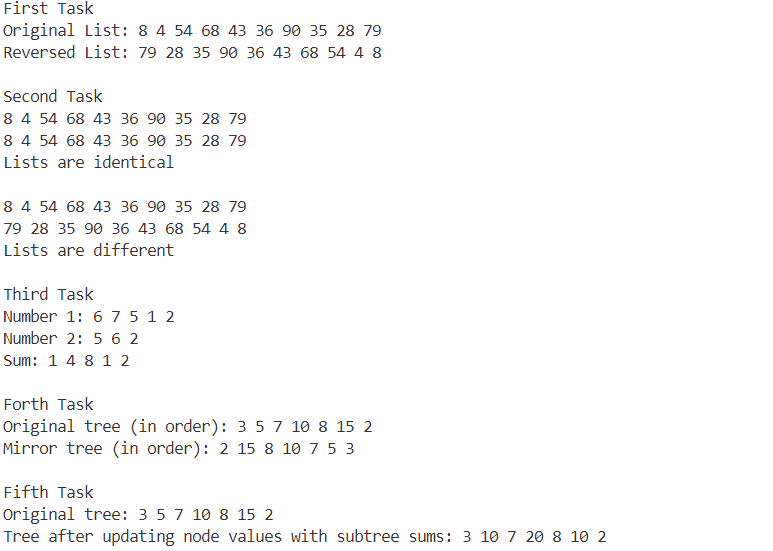
Найважчим завданням було створення зміни розміру масива, а разом з цим і зміна ємності згідно Grow Force рівному 2.

Також вважливою частиною завдання було перевантаження операторів, щоб кастомізовано використовувати їх до нашого динамічного масиву. 

Завдання №4

Цікавою імплементацією цих завдань було те що для створення однозв’язного списку та бінарного дерева, не обов’язково було використовувати батьківські структури, а всі методи діяли напряму до вузлів.

У бінарному дереві всі методи повинні використовувати рекурсію.



Завдання №5

Ініціалізація: обчислює префіксні суми для заданого масиву arr.

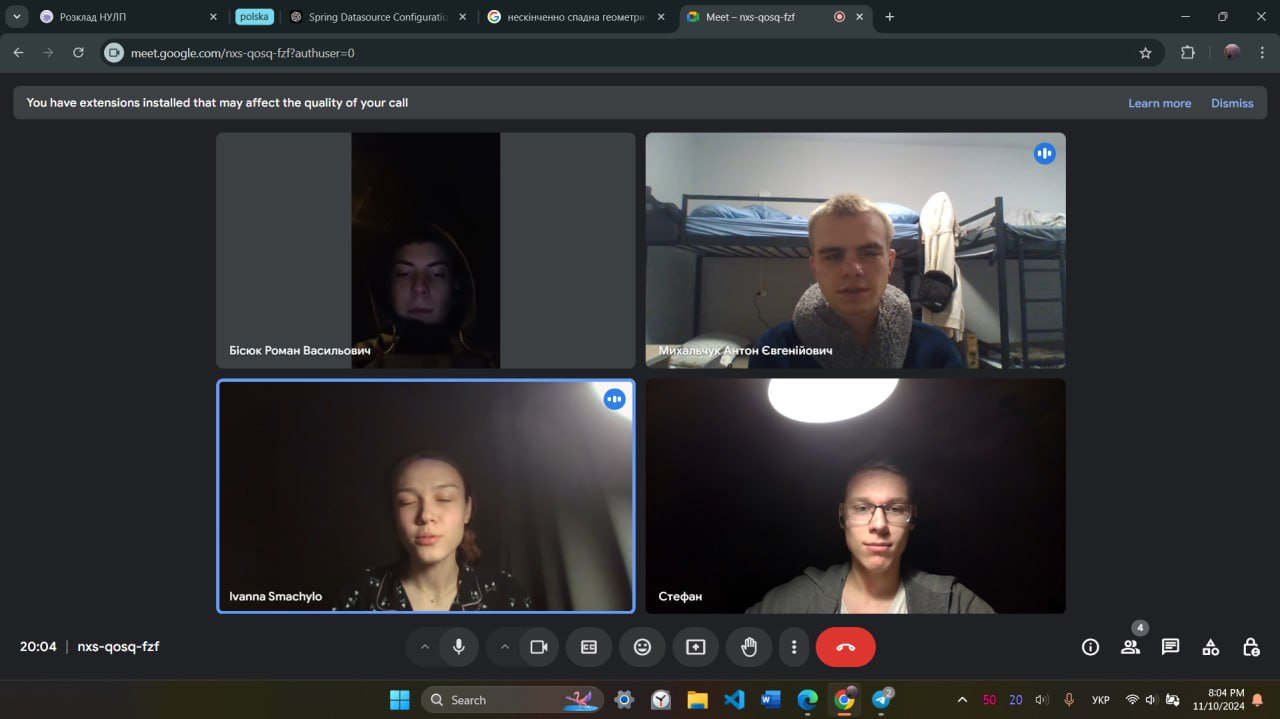
Метод maxPrefixWithSum: знаходить максимальну кількість елементів з префіксною сумою, що не перевищує x.

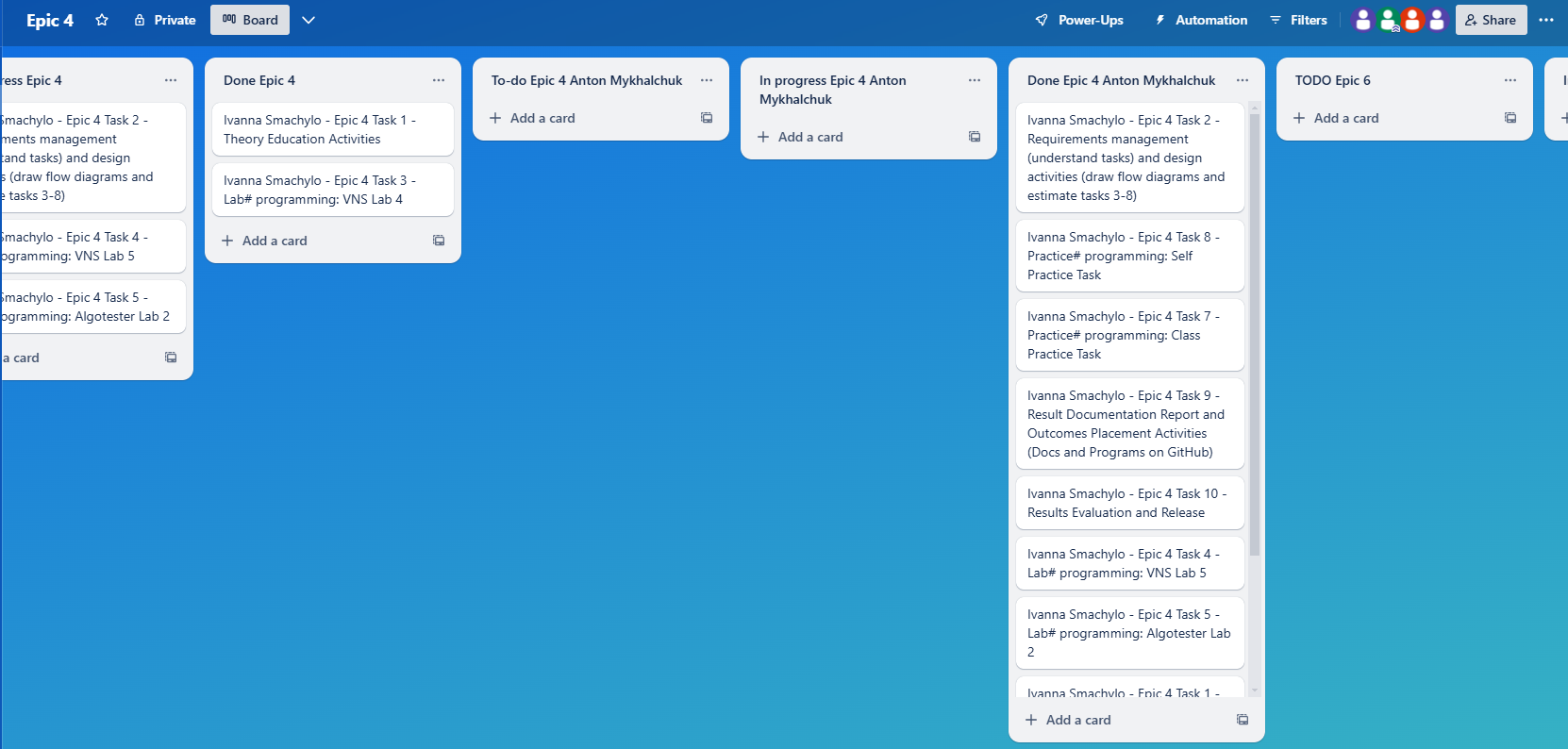
Метод addValue: додає значення до елемента масиву й оновлює відповідні префіксні суми.

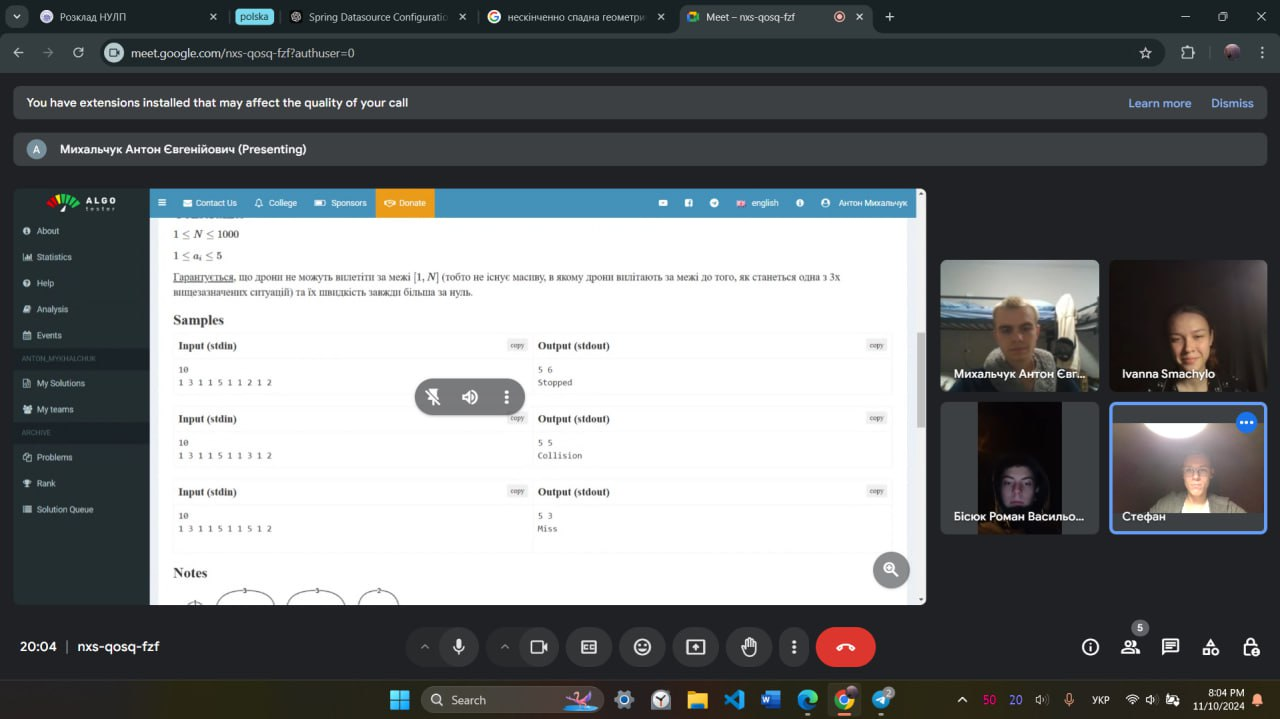
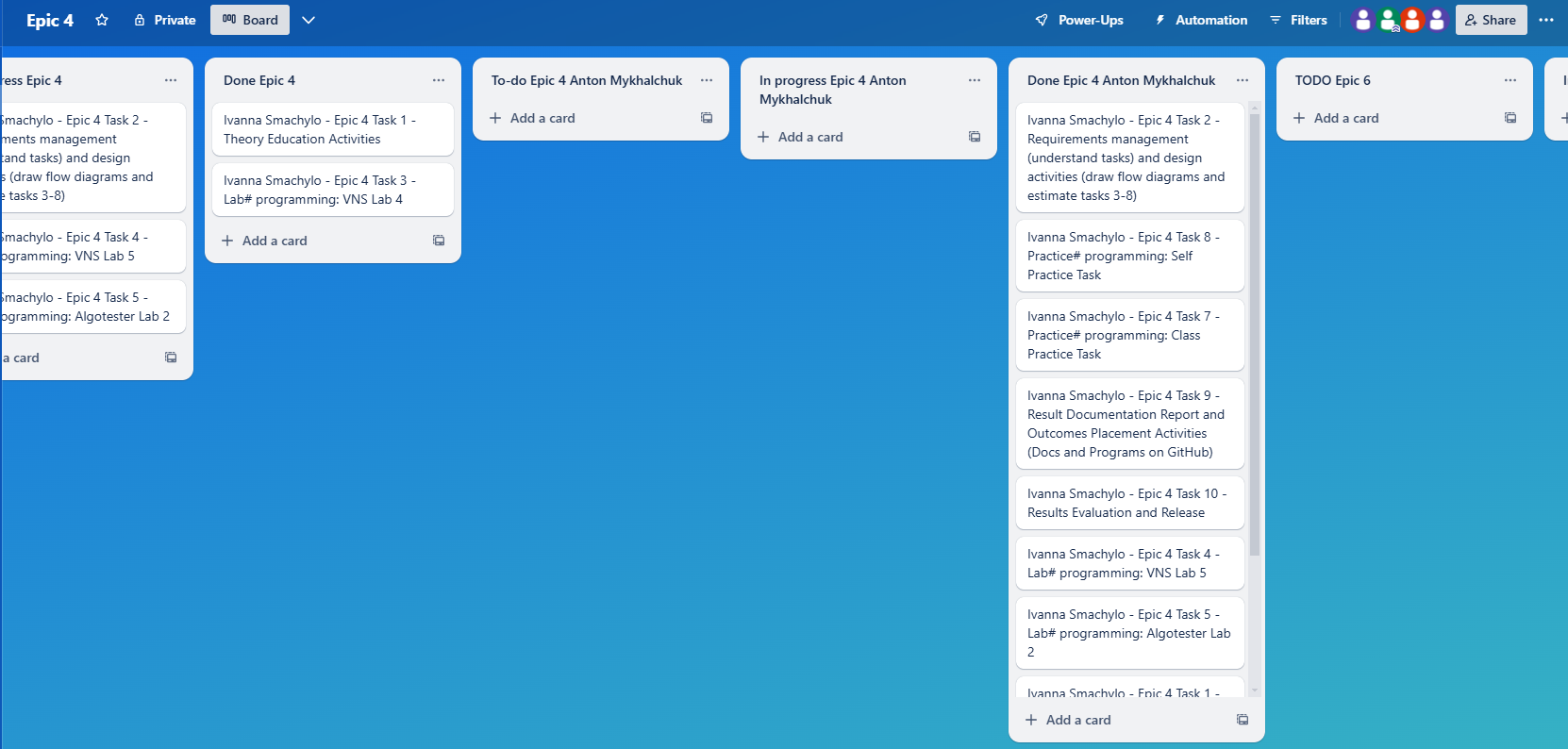
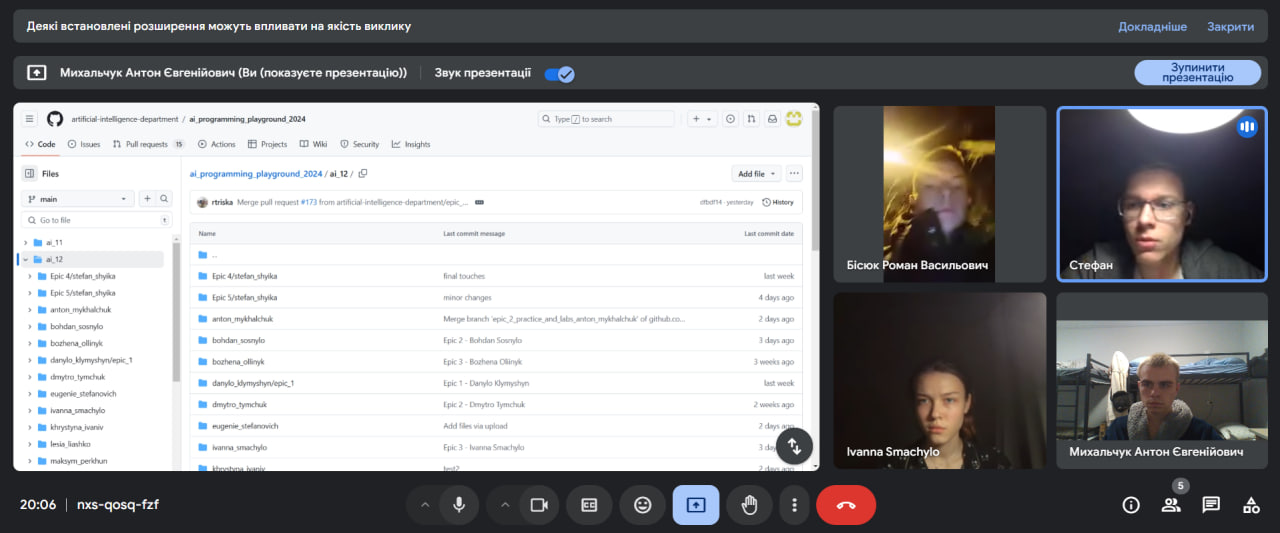
Основна програма: обробляє запити для знаходження префіксних сум (type 1) і оновлення елементів (type 2).

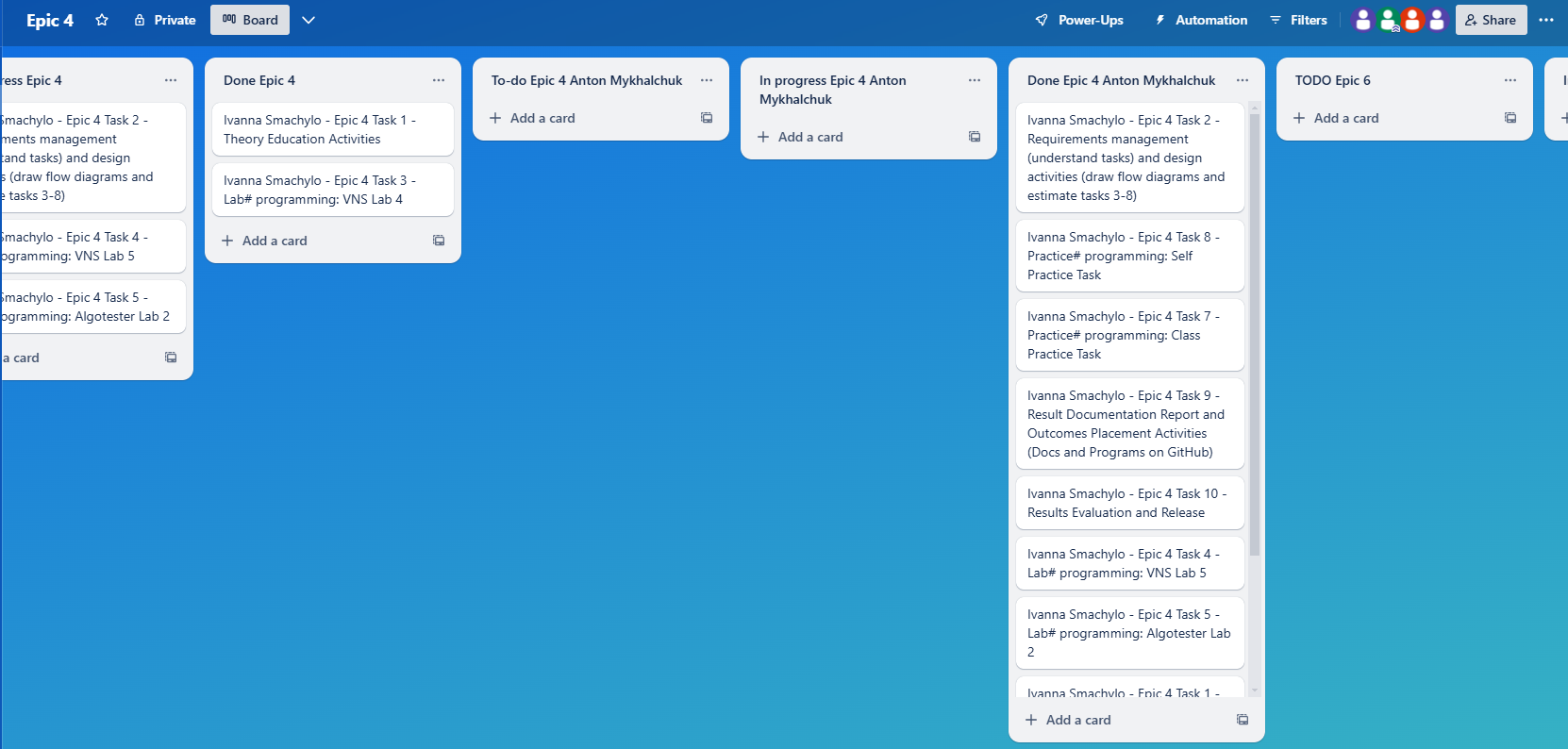
## **6. Кооперація з командою:**

* Скрін з 1-ї зустрічі по обговоренню задач Епіку та Скрін прогресу по Трелло





* Скрін з 2-ї зустрічі по обговоренню задач Епіку та Скрін прогресу по Трелло
* 
* 
* Скрін з 3-ї зустрічі по обговоренню задач Епіку та Скрін прогресу по Трелло (опційно)
* 



# **Висновки:**

Під час виконання роботи було опрацьовано основні динамічні структури даних: чергу, стек, списки та дерева. Кожна структура має свої особливості в реалізації та використанні для ефективного зберігання і обробки змінних даних, особливо у випадках, коли розмір або кількість елементів змінюється під час виконання програми. Завдяки практичним завданням було закріплено розуміння основних операцій над динамічними структурами: додавання, видалення, доступ до елементів, а також різноманітні специфічні операції, такі як реверсування списку, порівняння списків, сумування великих чисел у списках, віддзеркалення дерева, та підрахунок сум у піддеревах.

Досвід роботи з різними завданнями допоміг зрозуміти принципи побудови та обробки динамічних структур без використання стандартної бібліотеки шаблонів (STL). Це дало змогу розробити власні реалізації структур даних і алгоритмів, що дозволило глибше зрозуміти їх внутрішню логіку та особливості.

Важливим результатом є вивчення того, як динамічні структури можуть бути використані для оптимізації управління пам'яттю, а також для створення більш адаптивних програм, здатних ефективно працювати з великою кількістю даних. Такі знання є цінними для вирішення реальних практичних задач у програмуванні та подальшого розвитку в цій галузі.