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: «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
Єдинець Євген Русланович

Львів - 2024

Тема роботи:

Вивчення динамічних структур даних у C++: черга, стек, списки та дерева. Алгоритми обробки динамічних структур для ефективного зберігання та маніпулювання даними.

Мета роботи:

Ознайомитися з динамічними структурами даних у C++, зокрема з чергою, стеком, списками та деревами. Набути навичок створення та використання цих структур для зберігання та обробки даних. Вивчити алгоритми обробки динамічних структур, що дозволяють ефективно працювати з великою кількістю даних та забезпечувати оптимальну швидкодію програм.

Теоретичні відомості:

1. **Стек**
   https://www.youtube.com/watch?v=ZYvYISxaNL0&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=141

2. **Черга**

   https://www.youtube.com/watch?v=Yhw8NbjrSFA&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=142

3. **Бінарне дерево**

   https://www.youtube.com/watch?v=qBFzNW0ALxQ&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=144

4. **Однозв’язний список**

   https://www.youtube.com/watch?v=-25REjF_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=139

5. **Двозв’язний список**

   https://www.youtube.com/watch?v=QLzu2-_QFoE&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=140

Виконання роботи:

1) Опрацювання завдання та вимог до програм та середовища:

Завдання 1

VNS Lab 10

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.

Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити елементи перед і після елемента із заданим ключем. Додати по К елементів у початок й у кінець списку.

Завдання 2

Algotester Lab 5

https://algotester.com/uk/ContestProblem/DisplayWithEditor/135601

Завдання 3

Algotester Lab 78

https://algotester.com/uk/ContestProblem/DisplayWithEditor/135608

Завдання 4

Class Practice Work

Зв’язаний список

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;

Мета задачі

Розуміння структур даних: Реалізація методу реверсу для зв’язаних списків є чудовим способом для поглиблення розуміння зв’язаних списків як фундаментальної структури даних. Він заохочує практичний підхід до вивчення того, як структуруються пов’язані списки та як ними маніпулювати.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Перевертання пов’язаного списку вимагає логічного підходу до маніпулювання покажчиками, що є ключовим навиком у інформатиці.

Засвоїти механізми маніпуляції з покажчиками: пов’язані списки значною мірою залежать від покажчиків. Це завдання покращить навички маніпулювання вказівниками, що є ключовим аспектом у таких мовах, як C++.

Розвинути навички розв’язувати задачі: перевернути пов’язаний список непросто й вимагає творчого й логічного мислення, таким чином покращуючи свої навички розв’язування поставлених задач.

Пояснення прикладу

Спочатку ми визначаємо просту структуру ***Node*** для нашого пов’язаного списку.
Потім функція ***reverse*** ітеративно змінює список, маніпулюючи наступними покажчиками кожного вузла.
printList — допоміжна функція для відображення списку.
Основна функція створює зразок списку, демонструє реверсування та друкує вихідний і обернений списки.

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***.

Мета задачі

Розуміння рівності в структурах даних: це завдання допомагає зрозуміти, як визначається рівність у складних структурах даних, таких як зв'язані списки. На відміну від примітивних типів даних, рівність пов'язаного списку передбачає порівняння кожного елемента та їх порядку.

Поглиблення розуміння зв'язаних списків: Порівнюючи зв'язані списки, дозволяють покращити своє розуміння обходу, фундаментальної операції в обробці зв'язаних списків.

Розуміння ефективність алгоритму: це завдання також вводить поняття ефективності алгоритму. Студенти вчаться ефективно порівнювати елементи, що є навичкою, важливою для оптимізації та зменшення складності обчислень.

Розвинути базові навики роботи з реальними програми: функції порівняння мають вирішальне значення в багатьох реальних програмах, таких як виявлення змін у даних, синхронізація структур даних або навіть у таких алгоритмах, як сортування та пошук.

Розвинути навик вирішення проблем і увага до деталей: це завдання заохочує скрупульозний підхід до програмування, оскільки навіть найменша неуважність може призвести до неправильних результатів порівняння. Це покращує навички вирішення проблем і увагу до деталей.

Пояснення прикладу

- Для пов'язаного списку визначено структуру ***Node***.
- Функція ***compare*** ітеративно проходить обидва списки одночасно, порівнюючи дані в кожному вузлі.
- Якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.
- Основна функція ***main*** створює два списки та демонструє порівняння.

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 $\Rightarrow$ $9 \rightarrow 7 \rightarrow 3$);
- функція повертає новий список, передані в функцію списки не модифікуються.

Мета задачі

Розуміння операцій зі структурами даних: це завдання унаочнює практичне використання списка для обчислювальних потреб. Арифметичні операції з великими

числами це окремий клас задач, для якого використання списків допомагає обійти обмеження у представленні цілого числа сучасними комп'ютерами.

Поглиблення розуміння зв'язаних списків: Застосовування зв'язаних списків для арифметичних операції з великими числами дозволяє покращити розуміння операцій з обробки зв'язаних списків.

Розуміння ефективність алгоритму: це завдання дозволяє порівняти швидкість алгоритму додавання з використанням списків зі швидкістю вбудованих арифметичних операцій. Студенти вчаться розрізняти позитивні та негативні ефекти при виборі структур даних для реалізації практичних програм.

Розвинути базові навики роботи з реальними програми: арифметичні операції з великими числами використовуються у криптографії, теорії чисел, астрономії, та ін.

Розвинути навик вирішення проблем і увага до деталей: завдання покращує розуміння обмежень у представленні цілого числа сучасними комп'ютерами та пропонує спосіб його вирішення.

Бінарні дерева

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

Мета задачі

Розуміння структур даних: Реалізація методу віддзеркалення бінарного дерева покращує розуміння структури бінарного дерева, виділення пам'яті для вузлів та зв'язування їх у єдине ціле. Це один з багатьох методів роботи з бінарними деревами.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева продемонструє розгортання рекурсивного виклику.

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

Мета задачі

Розуміння структур даних: Реалізація методу підрахунку сум підвузлів бінарного дерева покращує розуміння структури бінарного дерева. Це один з багатьох методів роботи з бінарними деревами.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева демонструє розгортання рекурсивного виклику.

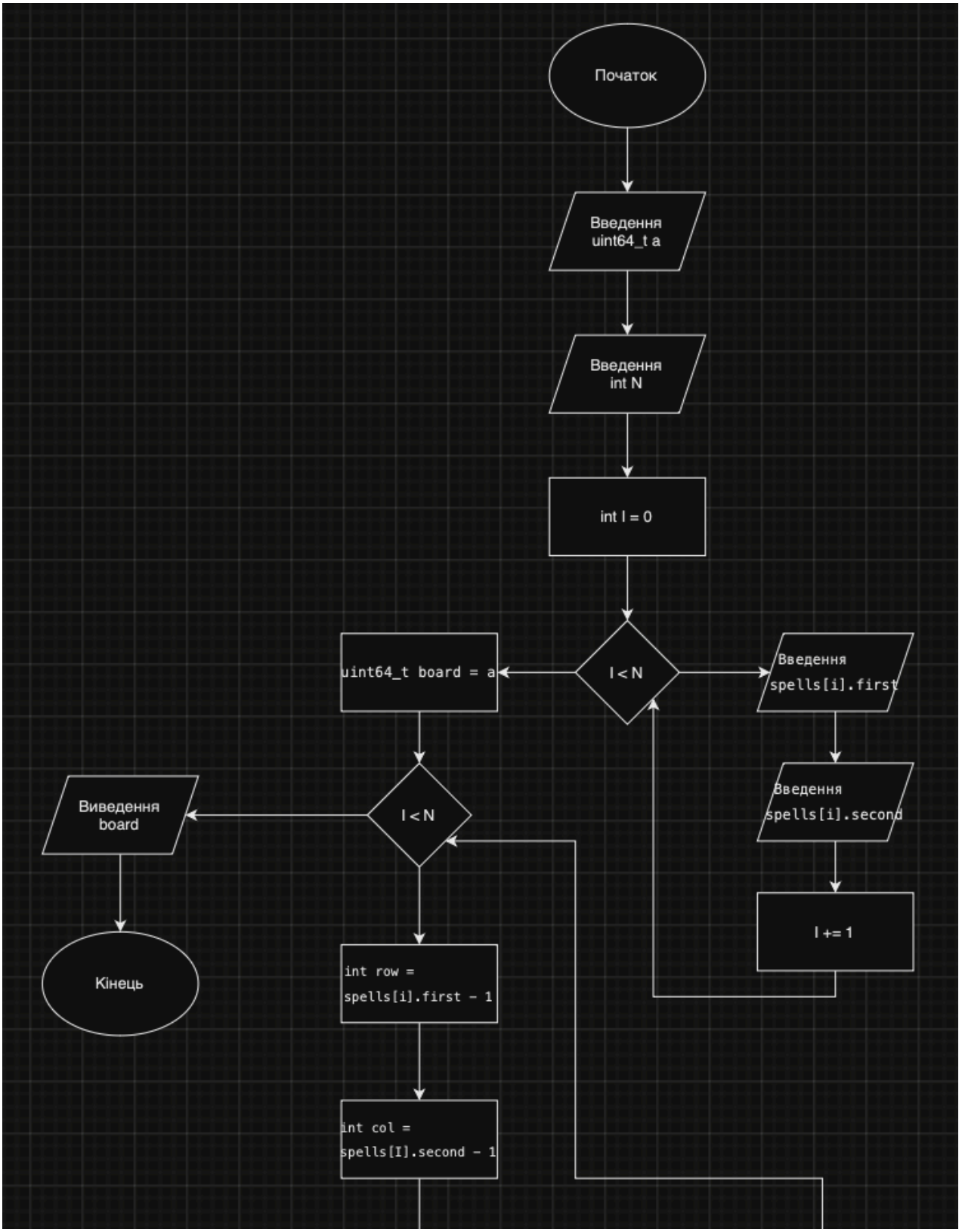
Завдання 5

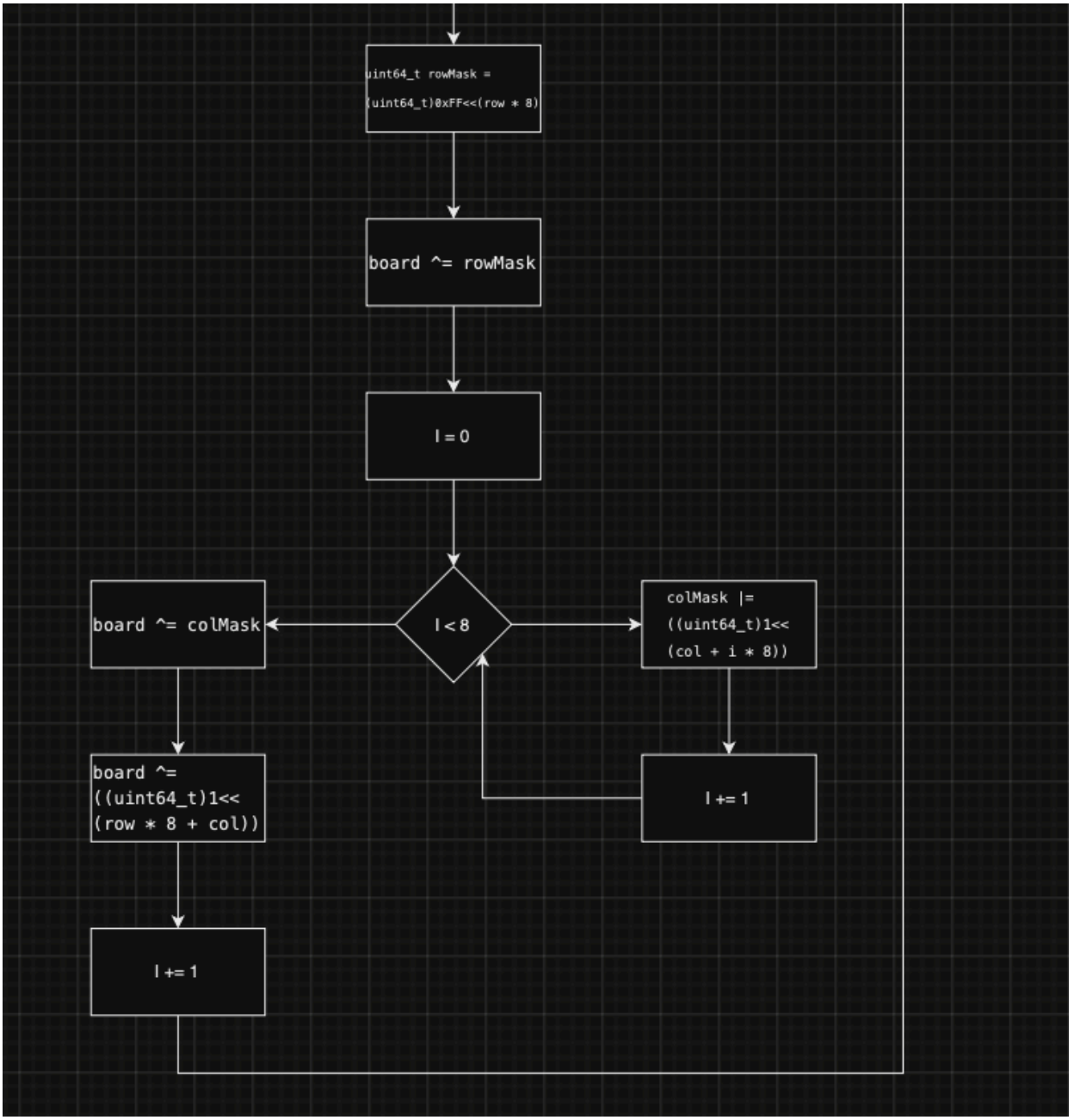
Self Practice Work

https://algotester.com/uk/ContestProblem/DisplayWithEditor/135609

2) Дизайн та планована оцінка часу виконання завдань:

Завдання 2


Початок
Введення uint64_t a
Введення int N
int I = 0
I < N
uint64_t board = a
Введення spells[i].first
Введення spells[i].second
I += 1
I < N
Виведення board
Кінець
int row = spells[i].first - 1
int col = spells[I].second - 1



Планований час виконання: 1 год. - 1 год. 20 хв.

3) **Код програм:**

Завдання 1:

```cpp
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

struct Node {
    string data;
    Node* next;
    Node* prev;
    Node(const string& value) : data(value), next(nullptr), prev(nullptr) {}
};

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    ~DoublyLinkedList() {
        clear();
    }

    void add_to_start(const string& value) {
        Node* new_node = new Node(value);
        if (!head) head = tail = new_node;
        else {
            new_node->next = head;
            head->prev = new_node;
            head = new_node;
        }
    }

    void add_to_end(const string& value) {
        Node* new_node = new Node(value);
        if (!tail) head = tail = new_node;
        else {
            new_node->prev = tail;
            tail->next = new_node;
            tail = new_node;
        }
    }

    void remove_before_and_after(const string& key) {
        Node* current = head;
        while (current && current->data != key) current = current->next;
        if (!current) return;
        if (current->prev) {
            Node* to_delete = current->prev;
            if (to_delete->prev) to_delete->prev->next = current;
            else head = current;
            current->prev = to_delete->prev;
            delete to_delete;
        }
        if (current->next) {
            Node* to_delete = current->next;
            if (to_delete->next) to_delete->next->prev = current;
            else tail = current;
            current->next = to_delete->next;
            delete to_delete;
        }
    }
```

```
    void print() const {
        if (!head) {
            cout << "List is empty.\n";
            return;
        }
        for (Node* current = head; current; current = current->next)
            cout << current->data << " ";
        cout << "\n";
    }

    void write_to_file(const string& filename) const {
        ofstream file(filename);
        for (Node* current = head; current; current = current->next)
            file << current->data << "\n";
    }

    void clear() {
        while (head) {
            Node* next_node = head->next;
            delete head;
            head = next_node;
        }
        tail = nullptr;
    }

    void restore_from_file(const string& filename) {
        clear();
        ifstream file(filename);
        string line;
        while (getline(file, line)) add_to_end(line);
    }
};

int main() {
    DoublyLinkedList list;
    list.add_to_start("Element1");
    list.add_to_end("Element2");
    list.add_to_end("Element3");
    list.add_to_end("KeyElement");
    list.add_to_end("Element4");
    list.add_to_end("Element5");

    cout << "Initial list: ";
    list.print();

    list.remove_before_and_after("KeyElement");
    cout << "After removing elements before and after 'KeyElement': ";
    list.print();

    list.add_to_start("StartElement1");
    list.add_to_start("StartElement2");
    list.add_to_end("EndElement1");
    list.add_to_end("EndElement2");
    cout << "After adding K elements: ";
    list.print();

    list.write_to_file("list.txt");

    list.clear();
    cout << "After clearing the list: ";
    list.print();

    list.restore_from_file("list.txt");
    cout << "After restoring from file: ";
    list.print();

    list.clear();
    cout << "After final clearing: ";
    list.print();

    return 0;
}
```

Завдання 2:

```cpp
#include <iostream>
#include <vector>
#include <cstdint>

using namespace std;

int main() {
    uint64_t a;
    int N;

    cin >> a;
    cin >> N;

    vector<pair<int, int>> spells(N);

    for (int i = 0; i < N; ++i) {
        cin >> spells[i].first >> spells[i].second;
    }

    uint64_t board = a;

    for (const auto& spell : spells) {
        int row = spell.first - 1;
        int col = spell.second - 1;

        uint64_t rowMask = (uint64_t)0xFF << (row * 8);

        board ^= rowMask;

        uint64_t colMask = 0;
        for (int i = 0; i < 8; ++i) {
            colMask |= ((uint64_t)1 << (col + i * 8));
        }

        board ^= colMask;

        board ^= ((uint64_t)1 << (row * 8 + col));
    }

    cout << board << endl;

    return 0;
}
```

Завдання 3:

```cpp
#include <iostream>
#include <vector>
#include <cstdint>

using namespace std;

int main() {
    uint64_t a;
    int N;

    cin >> a;
    cin >> N;

    vector<pair<int, int>> spells(N);

    for (int i = 0; i < N; ++i) {
        cin >> spells[i].first >> spells[i].second;
    }

    uint64_t board = a;

    for (const auto& spell : spells) {
        int row = spell.first - 1;
        int col = spell.second - 1;

        uint64_t rowMask = (uint64_t)0xFF << (row * 8);

        board ^= rowMask;

        uint64_t colMask = 0;
        for (int i = 0; i < 8; ++i) {
            colMask |= ((uint64_t)1 << (col + i * 8));
        }

        board ^= colMask;

        board ^= ((uint64_t)1 << (row * 8 + col));
    }

    cout << board << endl;

    return 0;
}
```

```cpp
#include <iostream>

using namespace std;

template <class T>
class DynamicArray {
private:
    T *elements;

public:
    int currentSize;
    int currentCapacity;

    DynamicArray() {
        this->currentSize = 0;
        this->currentCapacity = 1;
        this->elements = new T[1];
    }

    void insertAt(int pos, int count, T *values) {
        while (currentSize + count >= currentCapacity)
            currentCapacity *= 2;
        T *tempArray = new T[currentCapacity];

        for (int i = 0; i < pos; i++)
            tempArray[i] = elements[i];

        for (int i = 0; i < count; i++)
            tempArray[i + pos] = values[i];

        for (int i = pos; i < currentSize; i++)
            tempArray[i + count] = elements[i];

        this->currentSize += count;
        delete[] elements;
        elements = tempArray;
    }

    void removeAt(int pos, int count) {
        T *tempArray = new T[currentCapacity];
        int newSize = 0;
        for (int i = 0; i < this->currentSize; i++)
        {
            if (i < pos || i >= pos + count)
            {
                tempArray[newSize] = elements[i];
                newSize++;
            }
        }
        this->currentSize -= count;
        delete[] elements;
        elements = tempArray;
    }

    T getElement(int pos) {
        return this->elements[pos];
    }

    void setElement(int pos, T value) {
        this->elements[pos] = value;
    }

    void display(const string &separator) {
        for (int i = 0; i < this->currentSize; i++) {
            cout << elements[i];
            if (i < currentSize - 1) cout << separator;
        }
        cout << endl;
    }
};

```

Завдання 4:

```cpp
#include <iostream>

using namespace std;

struct Node {
    int data;
    Node* next;
    Node(int x) : data(x), next(nullptr) {}
};

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* h1, Node* h2) {
    while (h1 != nullptr && h2 != nullptr) {
        if (h1->data != h2->data) {
            return false;
        }
        h1 = h1->next;
        h2 = h2->next;
    }
    return h1 == nullptr && h2 == nullptr;
}

Node* add(Node* n1, Node* n2) {
    Node* result = nullptr;
    Node* prev = nullptr;
    Node* temp = nullptr;
    int carry = 0, sum;
    while (n1 != nullptr || n2 != nullptr || carry) {
        sum = carry;
        if (n1 != nullptr) {
            sum += n1->data;
            n1 = n1->next;
        }
        if (n2 != nullptr) {
            sum += n2->data;
            n2 = n2->next;
        }
        carry = sum / 10;
        sum %= 10;
        temp = new Node(sum);
        if (result == nullptr) {
            result = temp;
        } else {
            prev->next = temp;
        }
        prev = temp;
    }
    return result;
}

void printList(Node* head) {
    while (head != nullptr) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

int main() {
    cout << "=== Робота з пов'язаними списками ===\n";

    Node* list1 = new Node(1);
    list1->next = new Node(2);
    list1->next->next = new Node(3);
    cout << "Вхідний список: ";
    printList(list1);
    list1 = reverse(list1);
    cout << "Обернений список: ";
    printList(list1);

    Node* list2 = new Node(1);
    list2->next = new Node(2);
    list2->next->next = new Node(3);
    cout << "Списки рівні: " << (compare(list1, list2) ? "Так" : "Ні") << endl;

    Node* num1 = new Node(9);
    num1->next = new Node(9);
    Node* num2 = new Node(1);
    Node* sum = add(num1, num2);
    cout << "Сума чисел: ";
    printList(sum);

    return 0;
}
```

```cpp
#include <iostream>

using namespace std;

struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int x) : data(x), left(nullptr), right(nullptr) {}
};

TreeNode* create_mirror_flip(TreeNode* root) {
    if (root == nullptr) return nullptr;
    TreeNode* left = create_mirror_flip(root->left);
    TreeNode* right = create_mirror_flip(root->right);
    root->left = right;
    root->right = left;
    return root;
}

int tree_sum(TreeNode* root) {
    if (root == nullptr) return 0;
    int left_sum = tree_sum(root->left);
    int right_sum = tree_sum(root->right);
    if (root->left != nullptr || root->right != nullptr) {
        root->data = left_sum + right_sum;
    }
    return root->data + left_sum + right_sum;
}

void printTree(TreeNode* root) {
    if (root == nullptr) return;
    cout << root->data << " ";
    printTree(root->left);
    printTree(root->right);
}

int main() {
    cout << "=== Робота з бінарними деревами ===\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);
    cout << "Дерево до дзеркала: ";
    printTree(root);
    cout << endl;
    create_mirror_flip(root);
    cout << "Дерево після дзеркала: ";
    printTree(root);
    cout << endl;

    tree_sum(root);
    cout << "Дерево після підрахунку сум: ";
    printTree(root);
    cout << endl;

    return 0;
}
```

Завдання 5

```cpp
#include <iostream>

using namespace std;

template<typename T>
class Node {
public:
    T value;
    Node* left;
    Node* right;

    Node(T val) : value(val), left(nullptr), right(nullptr) {}
};

template<typename T>
class BinarySearchTree {
private:
    Node<T>* root;

    void insert(Node<T>*& node, T value) {
        if (node == nullptr) {
            node = new Node<T>(value);
        } else if (value < node->value) {
            insert(node->left, value);
        } else if (value > node->value) {
            insert(node->right, value);
        }
    }

    bool contains(Node<T>* node, T value) {
        if (node == nullptr) {
            return false;
        }
        if (value == node->value) {
            return true;
        } else if (value < node->value) {
            return contains(node->left, value);
        } else {
            return contains(node->right, value);
        }
    }

    void print(Node<T>* node) {
        if (node != nullptr) {
            print(node->left);
            cout << node->value << " ";
            print(node->right);
        }
    }

    int size(Node<T>* node) {
        if (node == nullptr) {
            return 0;
        }
        return 1 + size(node->left) + size(node->right);
    }

public:
    BinarySearchTree() : root(nullptr) {}

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

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

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

    int size() {
        return size(root);
    }
};
```

```cpp
int main() {
    BinarySearchTree<int> bst;
    int Q;
    cin >> Q;

    while (Q--) {
        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 == "print") {
            bst.print();
        } else if (command == "size") {
            cout << bst.size() << endl;
        }
    }

    return 0;
}
```

4) Результат виконання завдань та тестування:

Завдання 1:

```
Initial list: Element1 Element2 Element3 KeyElement Element4 Element5
After removing elements before and after 'KeyElement': Element1 Element2 KeyElement Element5
After adding K elements: StartElement2 StartElement1 Element1 Element2 KeyElement Element5 EndElement1 EndElement2
After clearing the list: List is empty.
After restoring from file: StartElement2 StartElement1 Element1 Element2 KeyElement Element5 EndElement1 EndElement2
After final clearing: List is empty.
```

Завдання 2:

C++ 23	Зараховано	0.003	1.223

https://algotester.com/uk/ContestProblem/DisplayWithEditor/135601

Завдання 3:

C++ 23	Зараховано	0.006	1.328

https://algotester.com/uk/ContestProblem/DisplayWithEditor/135608

Завдання 4:

```
=== Робота з пов'язаними списками ===
Вхідний список: 1 2 3
Обернений список: 3 2 1
Списки рівні: Ні
Сума чисел: 0 0 1
```

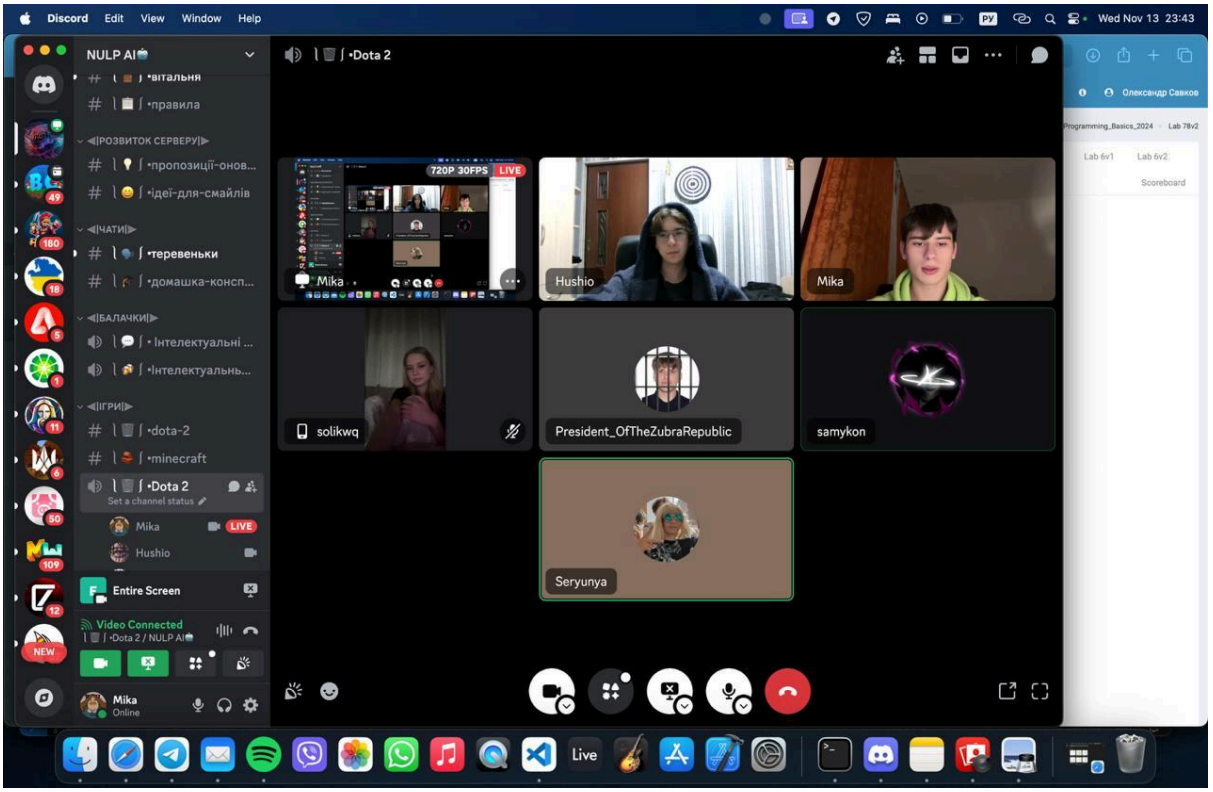
```
=== Робота з бінарними деревами ===
Дерево до дзеркала: 1 2 4 5 3
Дерево після дзеркала: 1 3 2 5 4
Дерево після підрахунку сум: 21 3 9 5 4
```

Завдання 5:

C++ 23	Зараховано	0.008	1.422

https://algotester.com/uk/ContestProblem/DisplayWithEditor/135609

Робота з командою:



Висновок:

Під час виконання цього епіку я вивчив динамічні структури даних у C++, зокрема чергу, стек, списки та дерева. Я засвоїв принципи їх створення та використання для зберігання і обробки даних. Окрім цього, я ознайомився з алгоритмами обробки динамічних структур, що дозволили мені ефективно працювати з великими обсягами даних та покращити

продуктивність програм. Ці навички стали основою для написання більш складних та оптимізованих програм, що використовують динамічні структури.