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

Виконала:
Студентка групи ШІ-13
Щербан Ярина Олегівна

Львів - 2024

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

Мета : навчитись працювати з динамічними структурами, реалізувати зв’язний список та навчитись працювати з бінарними деревами.

Теоретичні відомості:

1. Основи Динамічних Структур Даних:
    - Вступ до динамічних структур даних: визначення та важливість
    - Виділення пам'яті для структур даних (stack і heap)
2. Стек:
    - Визначення та властивості стеку
    - Операції push, pop, top: реалізація та використання
    - Приклади використання стеку: обернений польський запис, перевірка балансу дужок
    - Переповнення стеку
3. Черга:
    - Визначення та властивості черги
    - Операції enqueue, dequeue, front: реалізація та застосування
4. Зв'язні Списки:
    - Визначення однозв'язного та двозв'язного списку
    - Принципи створення нових вузлів, вставка між існуючими, видалення, створення кільця(circular linked list)
    - Основні операції: обхід списку, пошук, доступ до елементів, об'єднання списків
    - Приклади використання списків: управління пам'яттю, FIFO та LIFO структури
5. Дерева:
    - Бінарні дерева: вставка, пошук, видалення
    - Обхід дерева: в глибину (preorder, inorder, postorder), в ширину
    - Застосування дерев: дерева рішень, хеш-таблиці
    - Складніші приклади дерев: AVL, Червоно-чорне дерево
6. Алгоритми Обробки Динамічних Структур:
    - Основи алгоритмічних патернів: ітеративні, рекурсивні
    - Алгоритми пошуку, сортування даних, додавання та видалення елементів

Опрацювання теоретичного матеріалу :

1. Вивчення мови C++ за допомогою сайтів : https://www.w3schools.com/, https://acode.com.ua/
2. Робота з блок-схемами та Draw io https://www.programiz.com/article/flowchart-programming
3. Опрацювала відео про бінарні дерева : https://youtu.be/zuuAPYiMYDA?feature=shared

Виконання роботи:

1) Опрацювання завдання та вимог до програми та середовища

Завдання №1 Епік 6 : Практичне завдання :

Опис : 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 $\implies$ 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 №2 VNS Lab 10 - Task 1 – Variant 21 :

Опис :

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.

Умова : 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 №3 Algotester Lab 5 - Variant 1 :

Опис : У світі Атод сестри Ліна і Рілай люблять грати у гру. У них є дошка із 8-ми рядків і 8-ми стовпців. На перетині ii-го рядка і j-го стовпця лежить магічна куля, яка може світитись магічним світлом (тобто у них є 64 кулі). На початку гри деякі кулі світяться, а деякі ні... Далі вони обирають N куль і для кожної читають магічне заклинання, після чого всі кулі, які лежать на перетині стовпця і рядка обраної кулі міняють свій стан (ті що світяться - гаснуть, ті, що не світяться - загораються).
Також вони вирішили трохи Вам допомогти і придумали спосіб як записати стан дошки одним числом a із 8-ми байт, а саме (див. Примітки):

- Молодший байт задає перший рядок матриці;
- Молодший біт задає перший стовпець рядку;

- Значення біту каже світиться куля чи ні (0 - ні, 1 - так);

Тепер їх цікавить яким буде стан дошки після виконання N заклинань і вони дуже просять Вас їм допомогти.

Завдання №4 Algotester Lab 7-8 - Variant 2 :

Опис : Ваше завдання - власноруч реалізувати структуру даних "Динамічний масив". Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.

Умова **:** Для того щоб отримати **50%** балів за лабораторну достатньо написати свою структуру.
Для отримання **100%** балів ця структура має бути написана як **шаблон класу**, у якості параметру використати **intint**.
Використовувати STL заборонено.

Завдання №5 Algotester Lab 5 – Variant 2 :

Опис : В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це N, ширина - M. Всередині печери є пустота, пісок та каміння. Пустота позначається буквою O , пісок SS 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.

2) Дизайн та планована оцінка часу виконання завдань

Завдання №1 Епік 6 : Практичне завдання :

Запланований час виконання - 2 год

Завдання №2 VNS Lab 10 - Task 1 – Variant 21 :

Запланований час виконання - 2 год

Завдання №3 Algotester Lab 5 - Variant 1 :

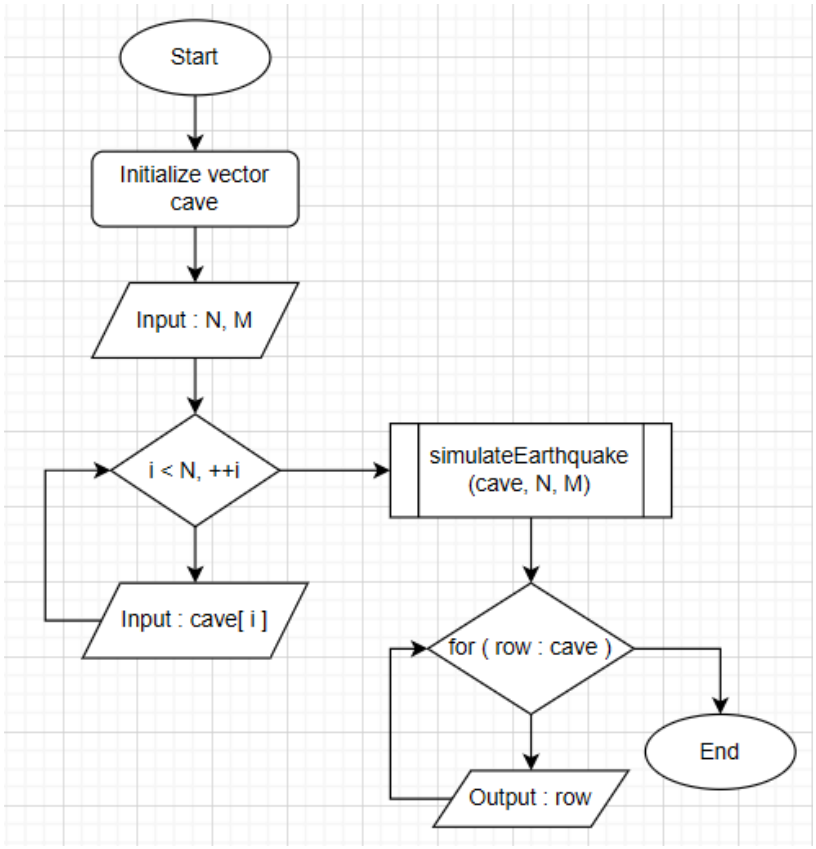
Запланований час виконання - 40 хв

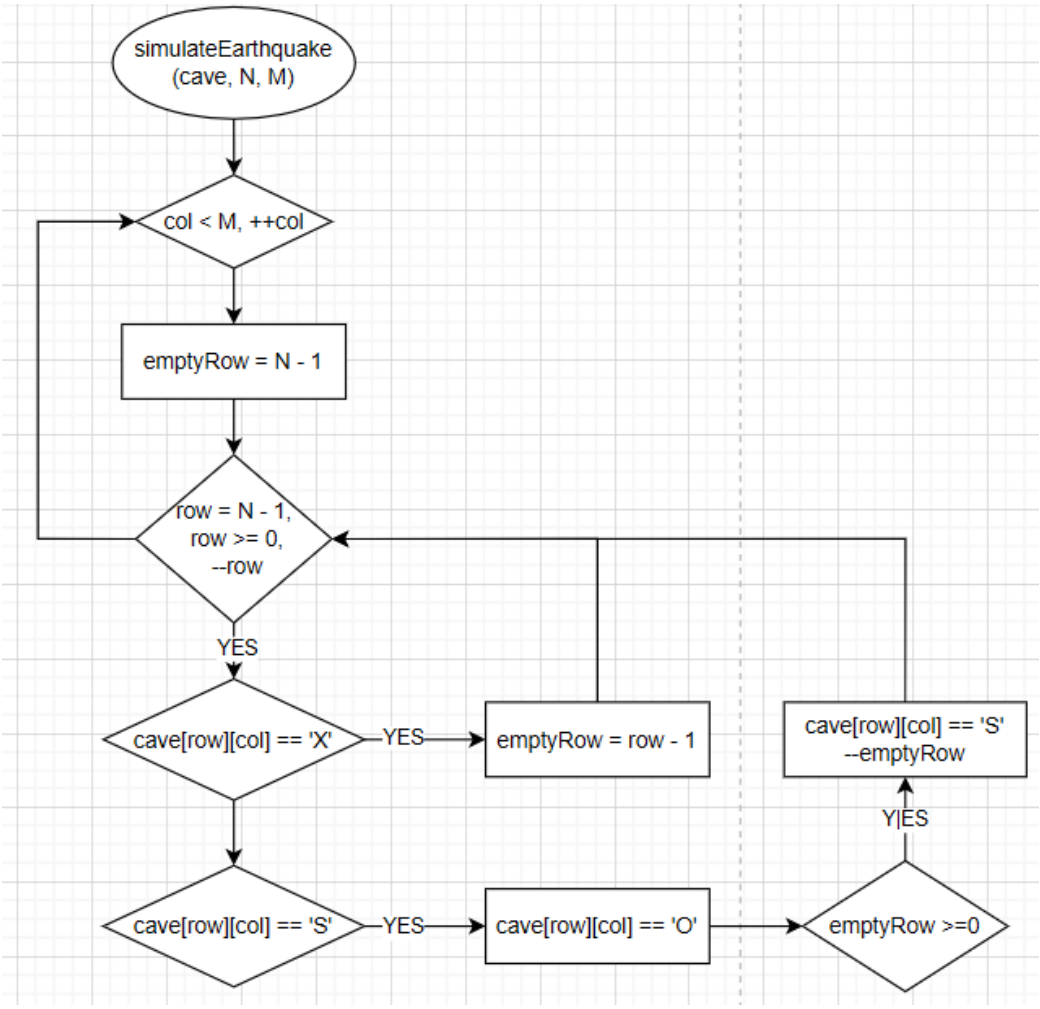
Завдання №4 Algotester Lab 7-8 - Variant 2 :

Запланований час виконання - 2 год

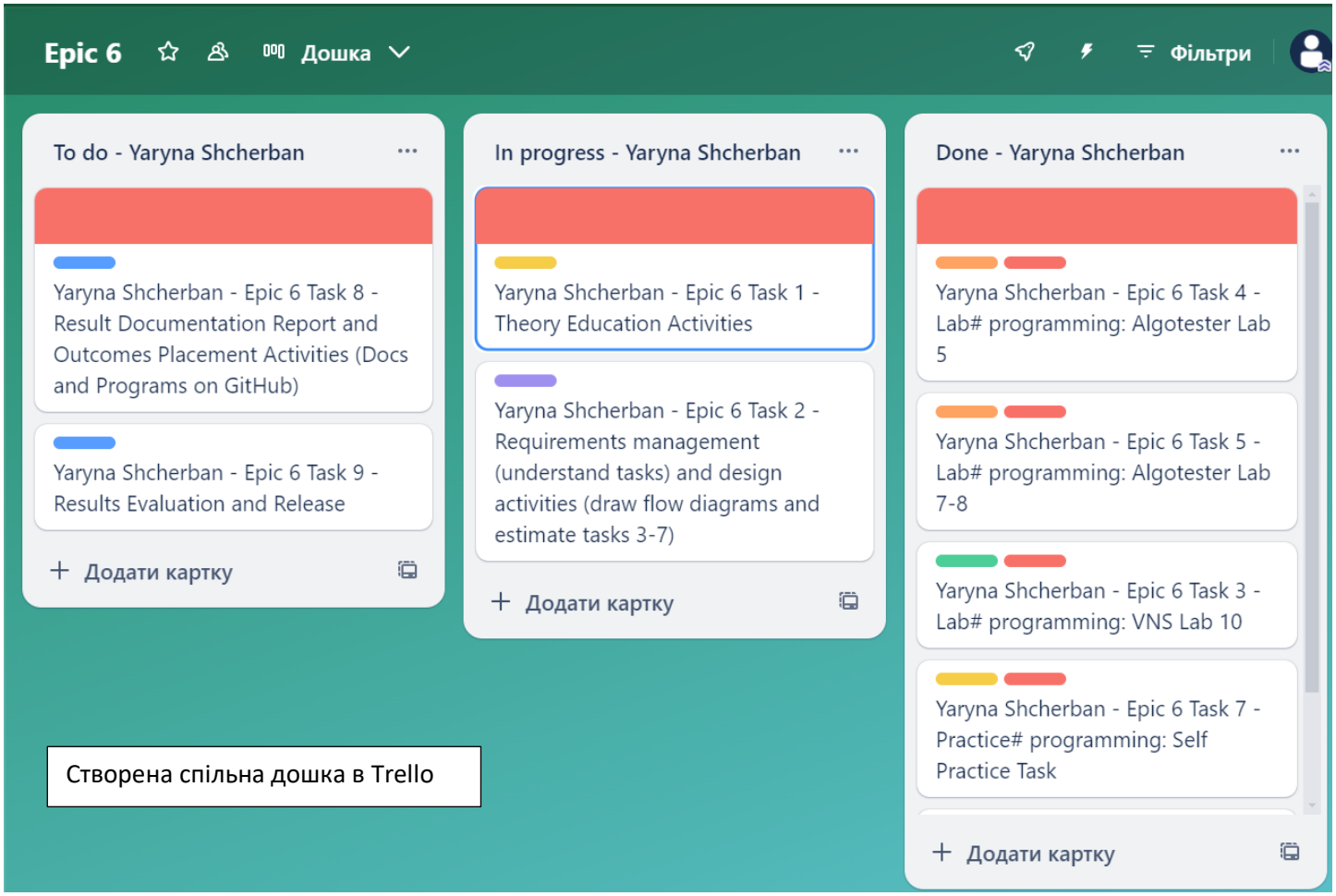
Завдання №5 Algotester Lab 5 – Variant 2 :

Запланований час виконання - 30 хв

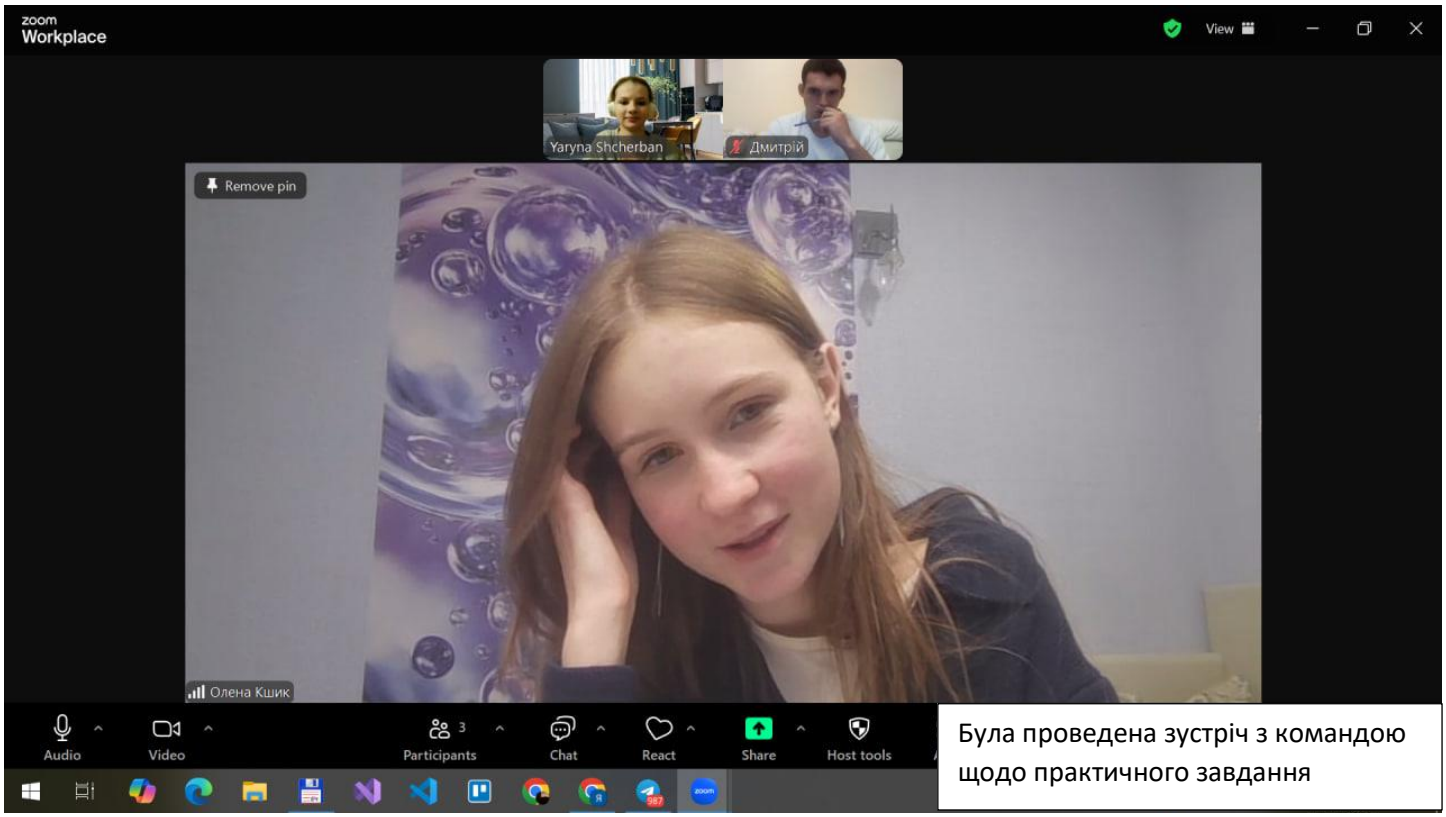




3) Конфігурація середовища до виконання завдань:



Створена спільна дошка в Trello



Була проведена зустріч з командою щодо практичного завдання

4) *Код програми з посиланням на зовнішні ресурси*

Завдання №1 Епік 6 : Практичне завдання :

practice_work_team_task_1_yaryna_shcherban.cpp

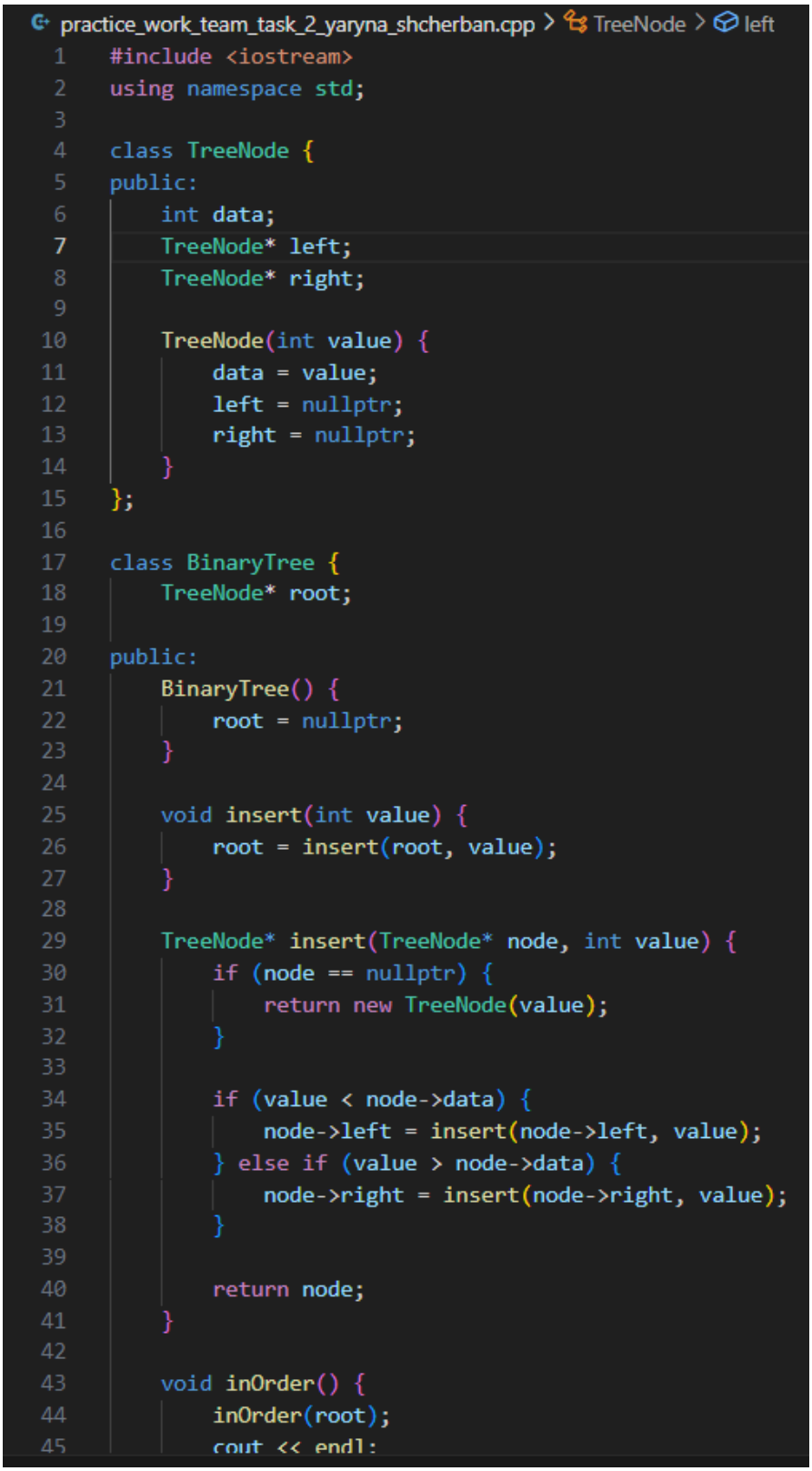
practice_work_team_task_1_yaryna_shcherban.cpp > compare(Node *, No

```cpp
#include <iostream>
#include <vector>
#include <string>
using namespace std;

class Node {
public:
    int data;
    Node* next;

    Node(int value) {
        data = value;
        next = nullptr;
    }
};

class LinkedList {
    Node* head;
public:
    LinkedList() {
        head = nullptr;
    }

    void append(vector<int>& arr) {
        for (int i = arr.size() - 1; i >= 0; i--) {
            Node* newNode = new Node(arr[i]);
            newNode->next = head;
            head = newNode;
        }
    }

    void reverse() {
        Node* prev = nullptr;
        Node* current = head;
        Node* next = nullptr;

        while (current != nullptr) {
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }
        head = prev;
    }
```

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

    Node* getHead() {
        return head;
    }
};

void add(Node* n1, Node* n2) {
    vector<int> tmp;
    int carry = 0;

    while (n1 != nullptr || n2 != nullptr || carry > 0) {
        int sum = carry;

        if (n1 != nullptr) {
            sum += n1->data;
            n1 = n1->next;
        }

        if (n2 != nullptr) {
            sum += n2->data;
            n2 = n2->next;
        }

        carry = sum / 10;               // Знаходимо перенiс
        tmp.insert(tmp.begin(), sum % 10); // Зберiгаємо залишок вiд дiлення
    }

    LinkedList list;
    list.append(tmp);

    // Формуємо суму як рядок для вiдображення
    string forsum = "";
    for (int digit : tmp) {
        forsum += to_string(digit);
    }
```

```
    cout << "Number of sum is: " << forsum << endl;
    list.print();
}

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;
}

int main() {
    LinkedList list1;
    LinkedList list2;

    vector<int> arr1 = {6, 4, 1, 2, 3, 11};
    vector<int> arr2 = {9, 0, 7, 8, 4, 1};

    list1.append(arr1);
    list2.append(arr2);

    cout << "Original lists: " << endl;
    list1.print();
    list2.print();

    // Task 1: Reverse lists
    list1.reverse();
    list2.reverse();
    cout << "Reversed lists: " << endl;
    list1.print();
    list2.print();

    // Task 2: Compare lists
    cout << "Comparing lists: ";
    if (compare(list1.getHead(), list2.getHead())) {
        cout << "Lists are equal." << endl;
    } else {
        cout << "Lists are different." << endl;
    }

    // Task 3: Add lists
    cout << "Adding lists: " << endl;
    add(list1.getHead(), list2.getHead());

    return 0;
}
```

practice_work_team_task_2_yaryna_shcherban.cpp




```cpp
#include <iostream>
using namespace std;

class TreeNode {
public:
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int value) {
        data = value;
        left = nullptr;
        right = nullptr;
    }
};

class BinaryTree {
    TreeNode* root;

public:
    BinaryTree() {
        root = nullptr;
    }

    void insert(int value) {
        root = insert(root, value);
    }

    TreeNode* insert(TreeNode* node, int value) {
        if (node == nullptr) {
            return new TreeNode(value);
        }

        if (value < node->data) {
            node->left = insert(node->left, value);
        } else if (value > node->data) {
            node->right = insert(node->right, value);
        }

        return node;
    }

    void inOrder() {
        inOrder(root);
        cout << endl;
```

```
48      void inOrder(TreeNode* node) {
49          if (node != nullptr) {
50              inOrder(node->left);
51              cout << node->data << " ";
52              inOrder(node->right);
53          }
54      }
55
56      TreeNode* mirrorFlip(TreeNode* node) {
57          if (node == nullptr) {
58              return nullptr;
59          }
60
61          TreeNode* flipped = new TreeNode(node->data);
62          flipped->left = mirrorFlip(node->right);
63          flipped->right = mirrorFlip(node->left);
64
65          return flipped;
66      }
67
68      void sumSubtrees(TreeNode* node) {
69          if (node == nullptr) return;
70
71          sumSubtrees(node->left);
72          sumSubtrees(node->right);
73
74          if (node->left != nullptr) {
75              node->data += node->left->data;
76          }
77          if (node->right != nullptr) {
78              node->data += node->right->data;
79          }
80      }
81
82      void sumSubtrees() {
83          sumSubtrees(root);
84      }
85
86      BinaryTree mirror() {
87          BinaryTree mirroredTree;
88          mirroredTree.root = mirrorFlip(root);
89          return mirroredTree;
90      }
```

```
92      void displayTree(TreeNode* node) {
93          if (node == nullptr) return;
94
95          cout << node->data << " ";
96          displayTree(node->left);
97          displayTree(node->right);
98      }
99
100     void displayTree() {
101         displayTree(root);
102         cout << endl;
103     }
104 };
105
106 int main() {
107     BinaryTree tree;
108     tree.insert(10);
109     tree.insert(5);
110     tree.insert(15);
111     tree.insert(3);
112     tree.insert(7);
113
114     cout << "Original tree (In-order traversal): ";
115     tree.inOrder();
116
117     cout << "\nMirrored tree: ";
118     BinaryTree mirroredTree = tree.mirror();
119     mirroredTree.displayTree();
120
121     cout << "\nTree after adding subtree sums: ";
122     tree.sumSubtrees();
123     tree.displayTree();
124
125     return 0;
126 }
```

Завдання №2 VNS Lab 10 - Task 1 – Variant 21 :

vns_lab_10_task_1_variant_21_yaryna_shcherban.cpp

vns_lab_10_task_1_variant_21_yaryna_shcherban.cpp > deleteAroundKey(D

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

using namespace std;

// Структура-вузол двонаправленого списку
struct Node {
    char* pc;
    Node* next;
    Node* prev;
};

// Структура для зберігання двонаправленого списку
struct DblLinkedList {
    size_t size;
    Node* head;
    Node* tail;
};

// Створення порожнього двонаправленого списку
DblLinkedList* createDblLinkedList() {
    DblLinkedList* list = new DblLinkedList;
    list->size = 0;
    list->head = list->tail = nullptr;
    return list;
}

// Видалення списку та всіх його елементів
void deleteDblLinkedList(DblLinkedList* list) {
    Node* current = list->head;
    Node* next = nullptr;

    while (current != nullptr) {
        next = current->next;
        delete[] current->pc;
        delete current;
        current = next;
    }

    delete list;
}
```

```
47  void pushBack(DblLinkedList* list, const char* value) {
48      Node* newNode = new Node;
49      newNode->pc = new char[strlen(value) + 1];
50      strcpy(newNode->pc, value);
51      newNode->next = nullptr;
52      newNode->prev = list->tail;
53
54      if (list->tail != nullptr) {
55          list->tail->next = newNode;
56      }
57      list->tail = newNode;
58      if (list->head == nullptr) {
59          list->head = newNode;
60      }
61      list->size++;
62  }
63
64  // Додавання елемента на початок списку
65  void pushFront(DblLinkedList* list, const char* value) {
66      Node* newNode = new Node;
67      newNode->pc = new char[strlen(value) + 1];
68      strcpy(newNode->pc, value);
69      newNode->prev = nullptr;
70      newNode->next = list->head;
71
72      if (list->head != nullptr) {
73          list->head->prev = newNode;
74      }
75      list->head = newNode;
76      if (list->tail == nullptr) {
77          list->tail = newNode;
78      }
79      list->size++;
80  }
81
82  // Видалення елемента з початку списку
83  const char* popFront(DblLinkedList* list) {
84      if (list->head == nullptr) {
85          throw runtime_error("List is empty!");
86      }
87
88      Node* temp = list->head;
89      const char* data = temp->pc;
90      list->head = list->head->next;
```

```
92      if (list->head != nullptr) {
94      } else {
95          list->tail = nullptr;
96      }
97
98      delete temp;
99      list->size--;
100     return data;
101 }
102
103 // Видалення елемента з кінця списку
104 const char* popBack(DblLinkedList* list) {
105     if (list->tail == nullptr) {
106         throw runtime_error("List is empty!");
107     }
108
109     Node* temp = list->tail;
110     const char* data = temp->pc;
111     list->tail = list->tail->prev;
112
113     if (list->tail != nullptr) {
114         list->tail->next = nullptr;
115     } else {
116         list->head = nullptr;
117     }
118
119     delete temp;
120     list->size--;
121     return data;
122 }
123
124 // Видалення елементів перед і після елементу із заданим ключем
125 void deleteAroundKey(DblLinkedList* list, const char* key) {
126     Node* current = list->head;
127     // Знайти вузол з ключем
128     while (current != nullptr) {
129         if (strcmp(current->pc, key) == 0) {
130             // Видалити попередній елемент
131             if (current->prev != nullptr) {
132                 Node* toDelete = current->prev;
133                 current->prev = toDelete->prev;
134                 if (toDelete->prev != nullptr) {
135                     toDelete->prev->next = current;
```

```
136                 } else {
139                     delete[] toDelete->pc;
140                     delete toDelete;
141                     list->size--;
142                 }
143                 // Видалити наступний елемент
144                 if (current->next != nullptr) {
145                     Node* toDelete = current->next;
146                     current->next = toDelete->next;
147                     if (toDelete->next != nullptr) {
148                         toDelete->next->prev = current;
149                     } else {
150                         list->tail = current; // Якщо видаляємо останній елемент
151                     }
152                     delete[] toDelete->pc;
153                     delete toDelete;
154                     list->size--;
155                 }
156                 return; // Виходимо після видалення
157             }
158             current = current->next;
159         }
160         cout << "Key not found!" << endl;
161     }
162
163     // Виведення списку на екран
164     void listPrint(DblLinkedList* list) {
165         if (list->head == nullptr) {
166             cout << "List is empty!" << endl;
167             return;
168         }
169
170         Node* current = list->head;
171         while (current != nullptr) {
172             cout << current->pc << " ";
173             current = current->next;
174         }
175         cout << endl;
176     }
```

```
179  void writeFile(DblLinkedList* list, const char* filename) {
180      ofstream file(filename);
181      if (!file.is_open()) {
182          throw runtime_error("Error opening file!");
183      }
184
185      Node* current = list->head;
186      while (current != nullptr) {
187          file << current->pc << endl;
188          current = current->next;
189      }
190  }
191
192  // Відновлення списку з файлу
193  void readFile(DblLinkedList* list, const char* filename) {
194      ifstream file(filename);
195      if (!file.is_open()) {
196          throw runtime_error("Error opening file!");
197      }
198
199      string line;
200      while (getline(file, line)) {
201          pushBack(list, line.c_str());
202      }
203  }
204
205  int main() {
206      DblLinkedList* list = createDblLinkedList();
207
208      // Створення списку з файлу
209      try {
210          readFile(list, "test.txt");
211          cout << "List after reading from file:" << endl;
212          listPrint(list);
213      } catch (const exception& e) {
214          cout << e.what() << endl;
215      }
216
217      // Додавання елементів на початок і в кінець
218      pushFront(list, "Hello");
219      pushBack(list, "World");
220      pushBack(list, "Additional");
221      pushBack(list, "Data");
222      pushFront(list, "NewStart");
223      cout << "List after pushing front and back:" << endl;
```

```
224     listPrint(list);
225
226     // Видалення елементів
227     try {
228         cout << "Pop front: " << popFront(list) << endl;
229         cout << "Pop back: " << popBack(list) << endl;
230         cout << "List after pop operations:" << endl;
231         listPrint(list);
232     } catch (const exception& e) {
233         cout << e.what() << endl;
234     }
235
236     // Видалення елементів перед і після заданого ключа
237     const char* key = "Hello"; // Задання ключа для видалення
238     deleteAroundKey(list, key);
239     cout << "List after deleting around key '" << key << "':" << endl;
240     listPrint(list);
241
242     // Запис списку в файл
243     try {
244         writeFile(list, "output.txt");
245         cout << "List written to output.txt" << endl;
246     } catch (const exception& e) {
247         cout << e.what() << endl;
248     }
249
250     // Видалення списку
251     deleteDblLinkedList(list);
252     return 0;
253 }
```

Завдання №3 Algotester Lab 5 - Variant 1 :

algotester_lab_5_task_variant_1_yaryna_shcherban.cpp

algotester_lab_5_task_variant_1_yaryna_shcherban.cpp > ...

```cpp
#include <iostream>
#include <cstdint> // Для uint64_t
using namespace std;

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--;
        for (int col = 0; col < 8; col++) {
            board ^= (1ULL << (r * 8 + col));
        }
        for (int row = 0; row < 8; row++) {
            board ^= (1ULL << (row * 8 + c));
        }
        board ^= (1ULL << (r * 8 + c));
    }
    cout << board << endl;
    return 0;
}
```

Завдання №4 Algotester Lab 7-8 - Variant 2 :

algotester_lab_7_8_task_1_variant_2_yaryna_shcherban.cpp

algotester_lab_7_8_task_1_variant_2_yaryna_shcherban.cpp > DynamicArray<T> > arr

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

using namespace std;

template <typename T>
struct DynamicArray {
    T *arr;
    int size;
    int capacity;
    DynamicArray() {
        size = 0;
        capacity = 1;
        arr = new T[capacity];
    }
    ~DynamicArray() {
        delete[] arr;
    }
};

template <typename T>
void insert(DynamicArray<T> &array, int index, int amount, T *toInsert) {
    while (array.size + amount >= array.capacity) {
        array.capacity *= 2;
    }
    T *temp = new T[array.capacity];
    for (int i = 0; i < index; i++) {
        temp[i] = array.arr[i];
    }
    for (int i = 0; i < amount; i++) {
        temp[index + i] = toInsert[i];
    }
    for (int i = index; i < array.size; i++) {
        temp[i + amount] = array.arr[i];
    }
    array.size += amount;
    delete[] array.arr;
    array.arr = temp;
}

template <typename T>
void erase(DynamicArray<T> &array, int index, int amount) {
    T *temp = new T[array.capacity];
    int acc = 0;
```

```
    for (int i = 0; i < array.size; i++) {
        if (i < index || i >= index + amount) {
            temp[acc] = array.arr[i];
            acc++;
        }
    }
    array.size -= amount;
    delete[] array.arr;
    array.arr = temp;
}

template <typename T>
T get(const DynamicArray<T> &array, int index) {
    return array.arr[index];
}

template <typename T>
void set(DynamicArray<T> &array, int index, T value) {
    array.arr[index] = value;
}

template <typename T>
void print(const DynamicArray<T> &array, const string &separator) {
    for (int i = 0; i < array.size; i++) {
        cout << array.arr[i] << separator;
    }
    cout << endl;
}

int main() {
    DynamicArray<int> arr;
    int q;
    cin >> q;
    while (q--) {
        string line;
        cin >> line;
        if (line == "insert") {
            int index, N;
            cin >> index >> N;
            int *temp = new int[N];
            for (int i = 0; i < N; i++) {
                cin >> temp[i];
            }
```

```
            insert(arr, index, N, temp);
            delete[] temp;
        } else if (line == "erase") {
            int index, N;
            cin >> index >> N;
            erase(arr, 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 << get(arr, i) << endl;
        } else if (line == "set") {
            int i, value;
            cin >> i >> value;
            set(arr, i, value);
        } else if (line == "print") {
            print(arr, " ");
        }
    }
    return 0;
}
```

algotester_lab_7_8_task_2_variant_2_yaryna_shcherban.cpp

algotester_lab_7_8_task_2_variant_2_yaryna_shcherban.cpp > main()

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

using namespace std;

template <typename T>
class DynamicArray {
private:
    T *arr;
    int size;
    int capacity;

public:
    // Конструктор
    DynamicArray() {
        size = 0;
        capacity = 1;
        arr = new T[capacity];
    }
    // Деструктор
    ~DynamicArray() {
        delete[] arr;
    }
    // Вставка елементів
    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];
        }
        size += amount;
        delete[] arr;
        arr = temp;
    }
    // Видалення елементів
    void erase(int index, int amount) {
        T *temp = new T[capacity];
```

```
46          int acc = 0;
47
48          for (int i = 0; i < size; i++) {
49              if (i < index || i >= index + amount) {
50                  temp[acc] = arr[i];
51                  acc++;
52              }
53          }
54
55          size -= amount;
56          delete[] arr;
57          arr = temp;
58      }
59      // Отримання елемента
60      T get(int index) const {
61          return arr[index];
62      }
63      // Встановлення значення елемента
64      void set(int index, T value) {
65          arr[index] = value;
66      }
67      // Отримання розміру масиву
68      int getSize() const {
69          return size;
70      }
71      // Отримання ємності масиву
72      int getCapacity() const {
73          return capacity;
74      }
75      // Виведення масиву
76      void print(const string &separator) const {
77          for (int i = 0; i < size; i++) {
78              cout << arr[i] << separator;
79          }
80          cout << endl;
81      }
82  };
83
84  int main() {
85      DynamicArray<int> arr;
86      int q;
87      cin >> q;
```

```
    while (q--) {
        string line;
        cin >> line;
        if (line == "insert") {
            int index, N;
            cin >> index >> N;
            int *temp = new int[N];
            for (int i = 0; i < N; i++) {
                cin >> temp[i];
            }
            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.getSize() << endl;
        } else if (line == "capacity") {
            cout << arr.getCapacity() << 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.print(" ");
        }
    }

    return 0;
}
```

Завдання №5 Algotester Lab 5 – Variant 2 :

practice_work_self_algotester_tasks_yaryna_shcherban.cpp

practice_work_self_algotester_tasks_yaryna_shcherban.cpp > main()

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

using namespace std;

void simulateEarthquake(vector<string>& cave, int N, int M) {
    for (int col = 0; col < M; ++col) {
        int emptyRow = N - 1;
        for (int row = N - 1; row >= 0; --row) {
            if (cave[row][col] == 'X') {
                emptyRow = row - 1;
            } else if (cave[row][col] == 'S') {
                cave[row][col] = '0';
                if (emptyRow >= 0) {
                    cave[emptyRow][col] = 'S';
                    --emptyRow;
                }
            }
        }
    }
}

int main() {
    int N, M;
    cin >> N >> M;

    vector<string> cave(N);
    for (int i = 0; i < N; ++i) {
        cin >> cave[i];
    }

    simulateEarthquake(cave, N, M);

    for (const auto& row : cave) {
        cout << row << endl;
    }

    return 0;
}
```

5) *Результати виконаних завдань, тестування та фактично затрачений час*

Завдання №1 Епік 6 : Практичне завдання :

Фактично затрачений час – 2.5 год

```
Original lists:
6 -> 4 -> 1 -> 2 -> 3 -> 11 -> null
9 -> 0 -> 7 -> 8 -> 4 -> 1 -> null
Reversed lists:
11 -> 3 -> 2 -> 1 -> 4 -> 6 -> null
1 -> 4 -> 8 -> 7 -> 0 -> 9 -> null
Comparing lists: Lists are different.
Adding lists:
Number of sum is: 1549082
1 -> 5 -> 4 -> 9 -> 0 -> 8 -> 2 -> null
```

```
Original tree (In-order traversal): 3 5 7 10 15

Mirrored tree: 10 15 5 7 3

Tree after adding subtree sums: 40 15 3 7 15
```

Завдання №2 VNS Lab 10 - Task 1 – Variant 21 :

Фактично затрачений час – 2.5 год

```
Error opening file!
List after pushing front and back:
NewStart Hello World Additional Data
Pop front: NewStart
Pop back: Data
List after pop operations:
Hello World Additional
List after deleting around key 'Hello':
Hello Additional
List written to output.txt
```

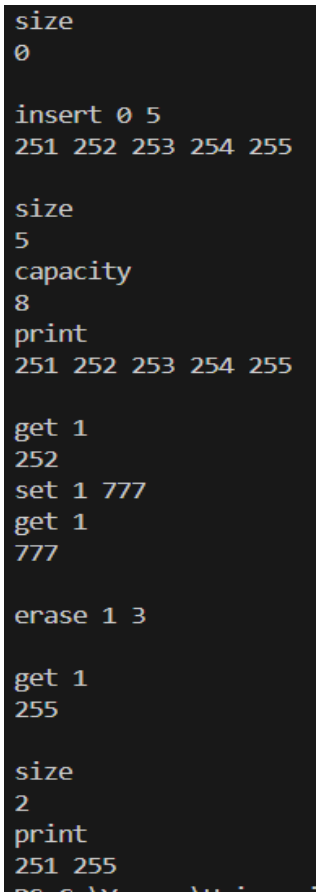
Завдання №3 Algotester Lab 5 - Variant 1 :

Фактично затрачений час – 40 хв

```
0
4
1 1
1 2
2 2
2 1
771
```

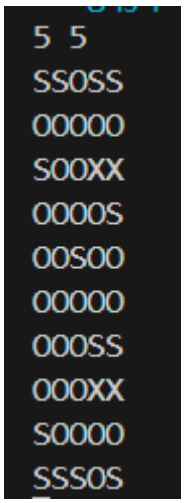
Завдання №4 Algotester Lab 7-8 - Variant 2 :

Фактично затрачений час – 3 год



Завдання №5 Algotester Lab 5 – Variant 2 :

Фактично затрачений час – 40 хв



Висновок : У ході виконання роботи було розглянуто основні динамічні структури даних: черги, стеки, зв’язні списки та бінарні дерева. Було реалізовано алгоритми для додавання, видалення та обробки елементів у зв’язному списку, а також операції з бінарним деревом, такі як вставка елементів, дзеркальне відображення та обчислення сум піддерев.

Посилання на Pull Request : https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/599