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-12

Бобровицький Олександр Сергійович

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

Мета: ознайомитись з темами лабораторної, опрацювати їх теоретично та навчитися використовувати отриманні знання для вирішення практичних задач

Теоретичні відомості:

Тема №1 : Основи Динамічних Структур Даних:

- Джерела:
  - https://acode.com.ua/urok-90-dynamichni-masyvy/
  - https://acode.com.ua/urok-89-dynamichne-vydilennya-pam-yati/
  - https://acode.com.ua/urok-111-stek-i-kupa/
- Що опрацьовано:
  - https://acode.com.ua/urok-90-dynamichni-masyvy/
  - https://acode.com.ua/urok-89-dynamichne-vydilennya-pam-yati/
  - https://acode.com.ua/urok-111-stek-i-kupa/
- Статус: Ознайомлений
- Початок опрацювання теми: 15.10
- Звершення опрацювання теми: 15.11

Тема №2 : Зв'язні Списки:

- Джерела:
  - https://www.geeksforgeeks.org/linked-list-data-structure/
  - https://www.geeksforgeeks.org/cpp-linked-list/
  - https://www.programiz.com/dsa/linked-list
- Що опрацьовано:
  - https://www.geeksforgeeks.org/linked-list-data-structure/
  - https://www.geeksforgeeks.org/cpp-linked-list/
  - https://www.programiz.com/dsa/linked-list
- Статус: Ознайомлений
- Початок опрацювання теми: 10.11
- Звершення опрацювання теми: 24.11

Тема №3 : Дерева:

- Джерела:
  - https://www.geeksforgeeks.org/introduction-to-binary-tree/
  - https://www.geeksforgeeks.org/tree-c-cpp-programs/
- Що опрацьовано:
  - https://www.geeksforgeeks.org/file-handling-c-classes/
  - https://www.w3schools.com/cpp/cpp_files.asp
- Статус: Ознайомлений
- Початок опрацювання теми: 10.11

– Звершення опрацювання теми: 25.11

Виконання роботи:

1. Опрацювання завдання та вимог до програм та середовища:

Завдання №1 Algotester Lab 5

- 2 варіант
- Деталі завдання:
  - 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 S і каміння X ;
  - Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння. Ваше завдання сказати як буде виглядати печера після землетрусу.

Завдання №2 VNS Lab 10

- 11 варіант
- Деталі завдання:
  - Записи в лінійному списку містять ключове поле типу *char(рядок символів). Сформувати двонаправлений список. Знищити з нього елемент із заданим ключем, додати елемент із зазначеним номером.

Завдання №3 Algotester Lab 7-8 v1

- 1 варіант
- Деталі завдання:
  - Ваше завдання - власноруч реалізувати структуру даних "Двозв’язний список".
    Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.
  - Вам будуть поступати запити такого типу: Вставка, Видалення, Визначення розміру, Отримання значення і-го елементу, Модифікація значення і-го елементу, Вивід списку на екран.
  - **Для того щоб отримати 50% балів за лабораторну достатньо написати свою структуру.**

Завдання №4 Algotester Lab 7-8 v2

- 1 варіант
- Деталі завдання:
  - Ваше завдання - власноруч реалізувати структуру даних "Двозв’язний список".
    Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.

- Вам будуть поступати запити такого типу: Вставка, Видалення, Визначення розміру, Отримання значення і-го елементу, Модифікація значення і-го елементу, Вивід списку на екран.
- **Для отримання 100% балів ця структура має бути написана як шаблон класу, у якості параметру використати int. Використовувати STL заборонено.**

Завдання №5 Class practice work

- 1 варіант
- Деталі завдання:
  - Реалізувати метод реверсу списку.
  - реалізувати функцію, яка ітеративно проходиться по обох списках і порівнює дані в кожному вузлі.
  - реалізувати функцію, яка обчислює суму двох чисел, які збережено в списку молодший розряд числа записано в голові списка.
  - реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева
  - реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів

Завдання №6 Self practice work

- 1172 варіант
- Деталі завдання:
  - Зеник і Марічка мають масив з n цілих чисел ai. Вони хочуть перевпорядкувати елементи масиву так, щоб сума будь-яких двох послідовних елементів була непарною. Будь ласка, допоможіть їм зробити це або скажіть, що це неможливо.

2. Дизайн та планована оцінка часу виконання завдань:

Програма №1 VNS Lab 10

Планований час на реалізацію – 90min

Програма №2 Algotester Lab 5

Планований час на реалізацію – 20min

Програма №3 Algotester Lab 7-8 v1

Планований час на реалізацію – 90min

Програма №4 Algotester Lab 7-8 v2

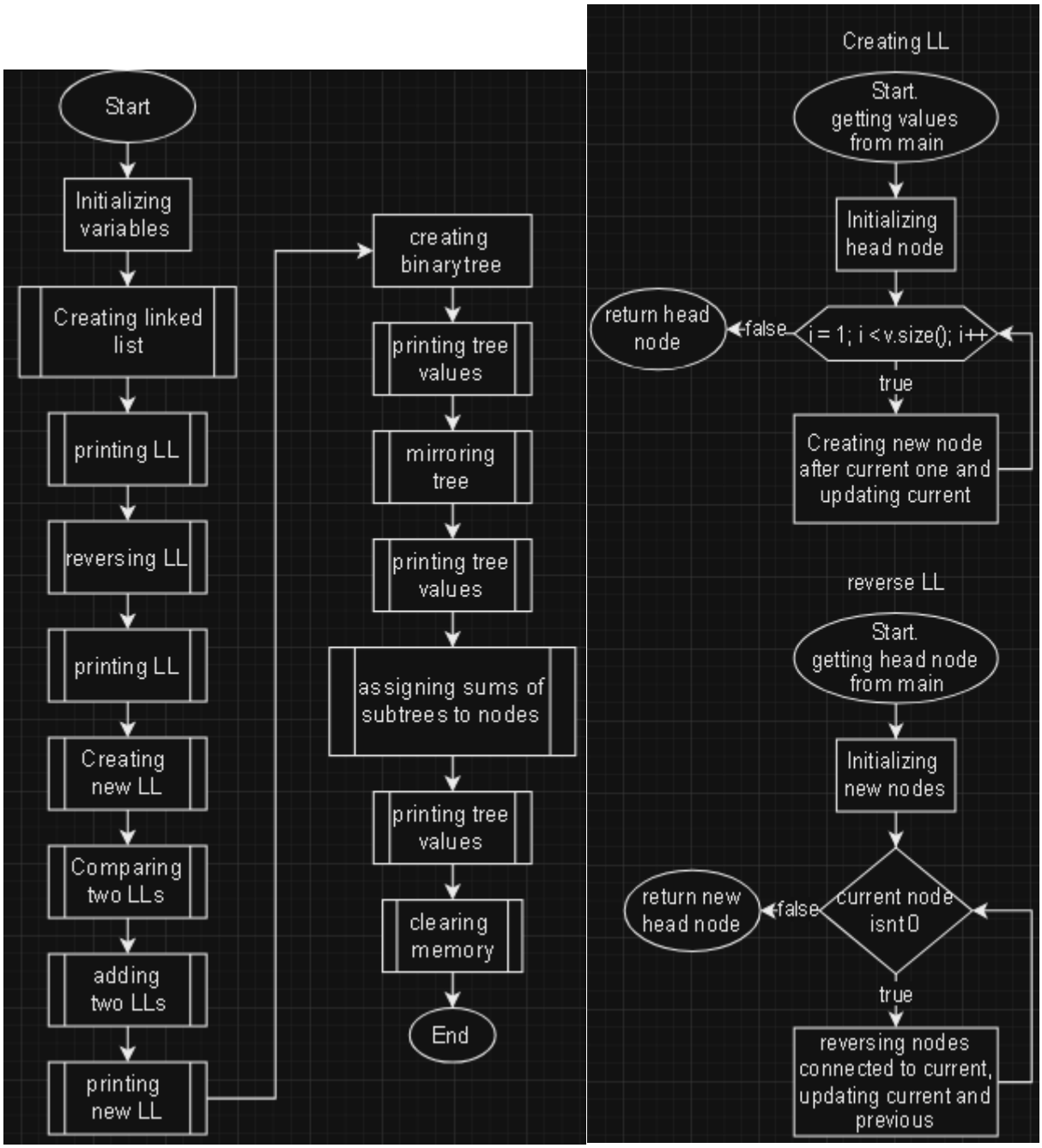
Планований час на реалізацію – 90min

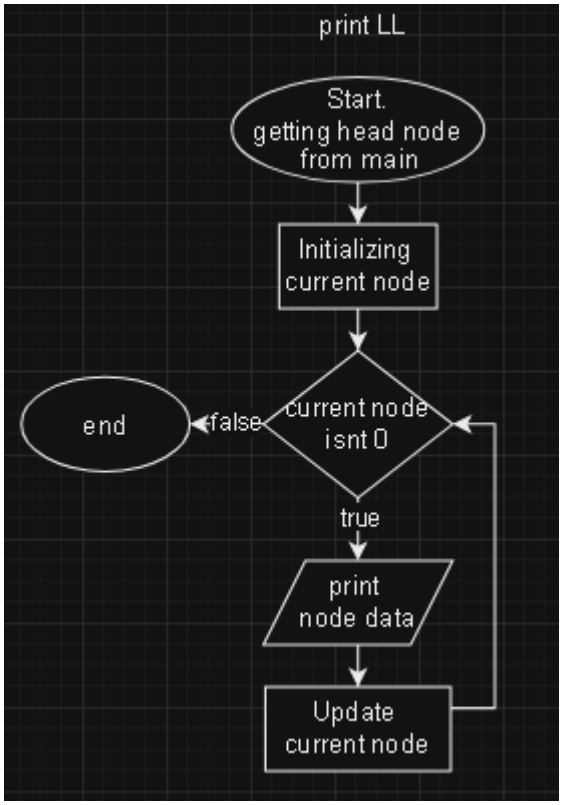
Програма №5 Class practice work

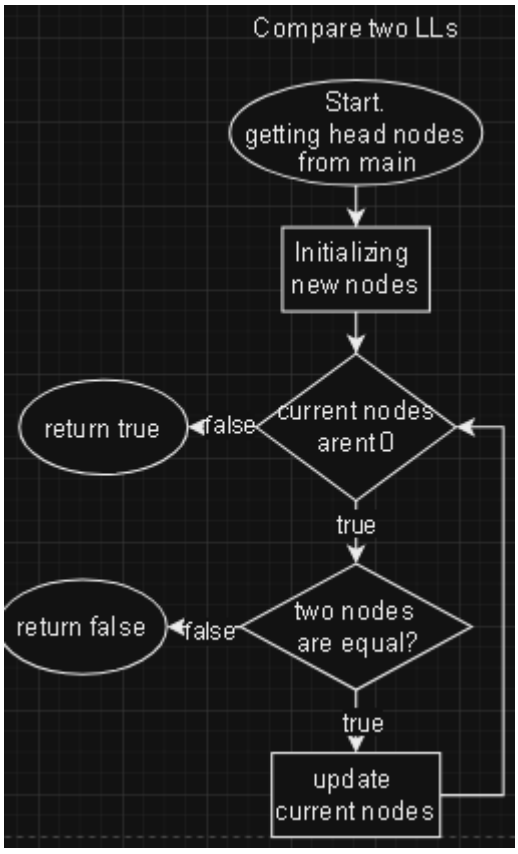
Планований час на реалізацію – 90min

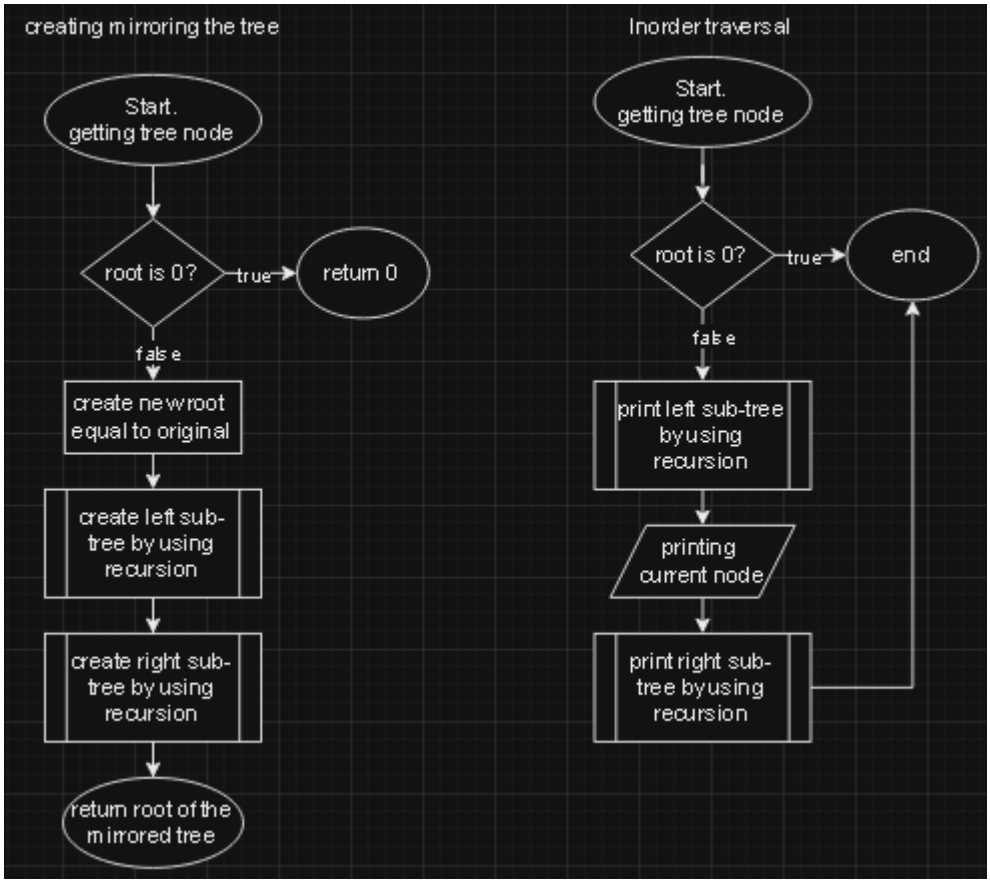
Програма №6 Self -practice work

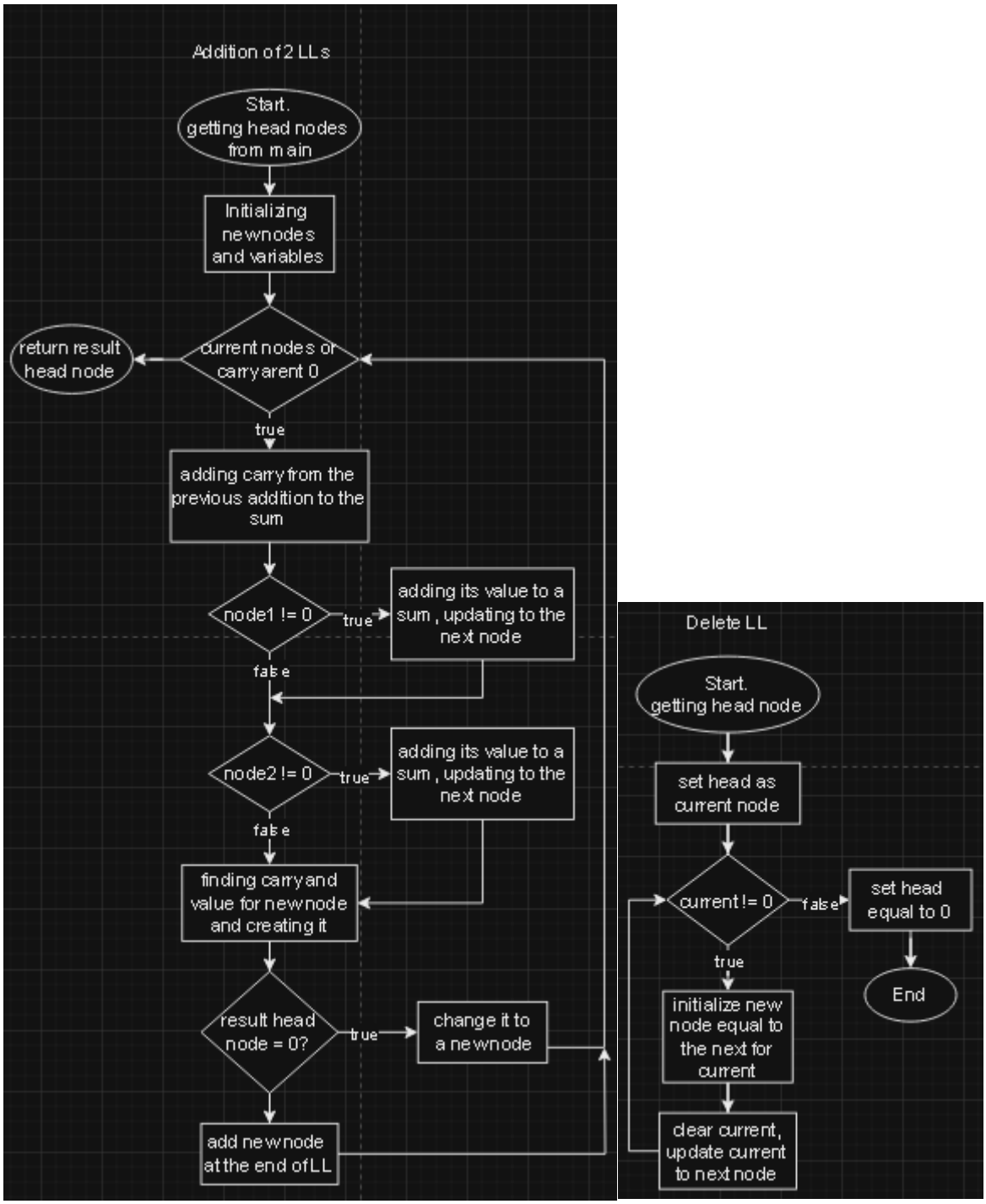
Блок-схема:

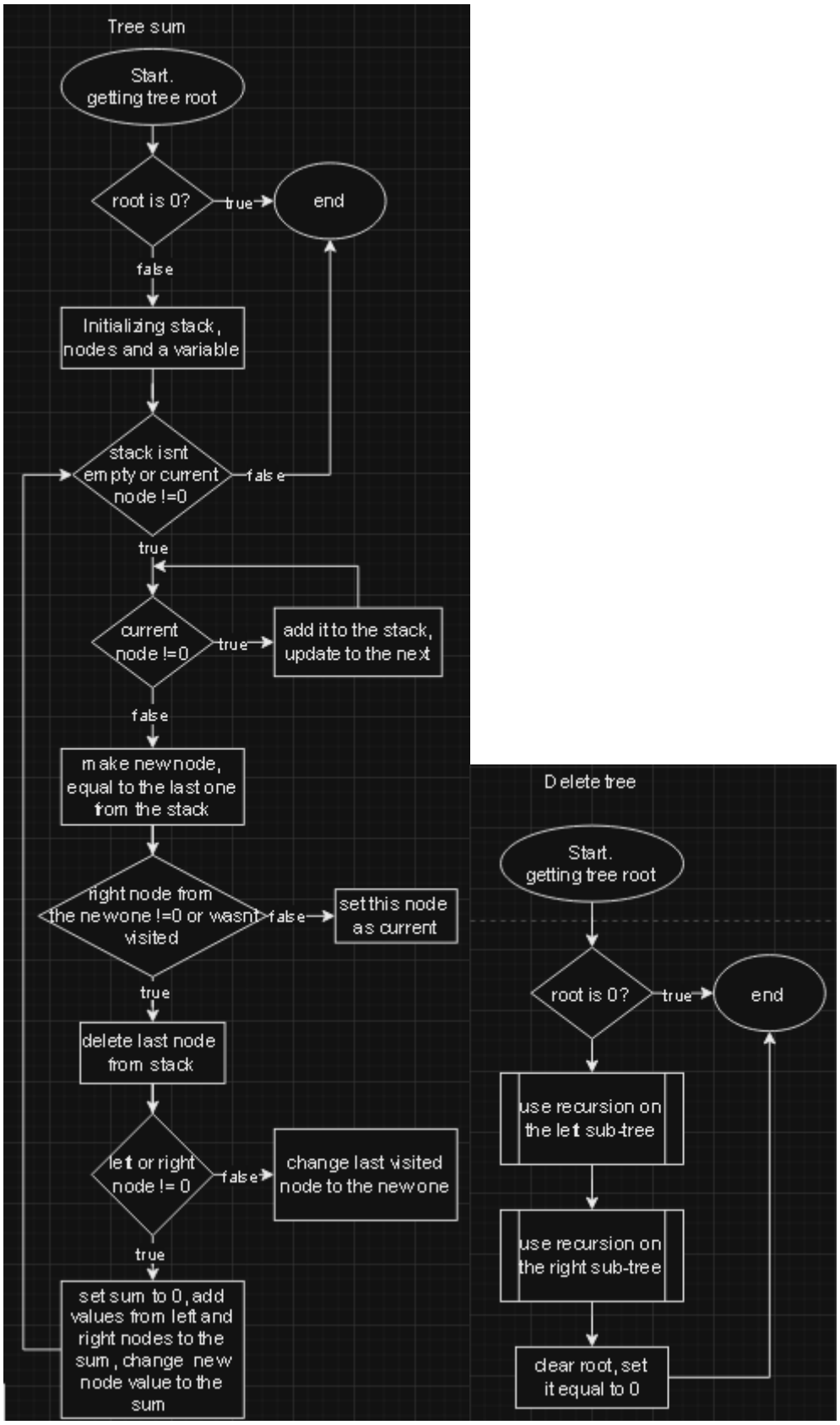



print LL
Start.
getting head node
from main
Initializing
current node
current node
isnt 0
false
end
true
print
node data
Update
current node


Compare two LLs
Start.
getting head nodes
from main
Initializing
new nodes
current nodes
arent 0
false
return true
true
two nodes
are equal?
false
return false
true
update
current nodes


creating mirroring the tree
Start.
getting tree node
root is 0?
true
return 0
false
create new root
equal to original
create left sub-
tree by using
recursion
create right sub-
tree by using
recursion
return root of the
mirrored tree
Inorder traversal
Start.
getting tree node
root is 0?
true
end
false
print left sub-tree
by using
recursion
printing
current node
print right sub-
tree by using
recursion


Addition of 2 LLs
Start.
getting head nodes
from main
Initializing
newnodes
and variables
current nodes or
carryarent 0
return result
head node
true
adding carry from the
previous addition to the
sum
node1 != 0
true
adding its value to a
sum , updating to the
next node
false
node2 != 0
true
adding its value to a
sum , updating to the
next node
false
finding carryand
value for newnode
and creating it
result head
node = 0?
true
change it to
a newnode
add newnode
at the end ofLL
Delete LL
Start.
getting head node
set head as
current node
current != 0
false
set head
equal to 0
End
true
initialize new
node equal to
the next for
current
clear current,
update current
to next node



Планований час на реалізацію – 25min

3. Код програм з посиланням на зовнішні ресурси:

Завдання №1

Посилання на файл програми у пул-запиті GitHub https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/491/commits/d2422006e96ab147ac47ccee460642e5859e96c1#diff-4e457b528fa7afa68927a632df1f4e012e0b2d1641d8175ddfc8e7b5658057fd

```cpp
#include <bits/stdc++.h>

using namespace std;

struct Node
{
    string data;
    Node *next;
    Node *prev;

    Node(string d) : data(d), next(nullptr), prev(nullptr) {}
};

Node *createList()
{
    return nullptr;
}

void addElement(Node *&head, string data, int position)
{
    Node *NewNode = new Node(data);
    if (head == nullptr)
    {
        head = NewNode;
        return;
    }

    if (position == 0)
    {
        NewNode->next = head;
        head->prev = NewNode;
        head = NewNode;
        return;
    }

    Node *temp = head;
    int index = 0;

    while (temp->next && index < position - 1)
    {
        temp = temp->next;
        index++;
    }

    NewNode->next = temp->next;
    NewNode->prev = temp;
    if (temp->next)
    {
        temp->next->prev = NewNode;
    }
    temp->next = NewNode;
}

void printList(Node *head)
{
    if (head == nullptr)
    {
        cout << "list is empty" << endl;
        return;
    }

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

void deleteElement(Node *&head, string key)
{
    if (head == nullptr)
    {
        cout << "list is empty" << endl;
        return;
    }

    Node *temp = head;

    while (temp->data != key)
    {
        temp = temp->next;
    }

    if (temp == nullptr)
    {
        cout << "No elment is found for key: " << key << endl;
        return;
    }

    if (temp->prev != nullptr)
        temp->prev->next = temp->next;
    if (temp->next != nullptr)
        temp->next->prev = temp->prev;
    if (temp == head)
        head = temp->next;

    delete temp;

}
```

```cpp
void writeToFile(Node *head, string filename)
{
    ofstream file;
    file.open(filename),

    if (!file.is_open())
    {
        cout << "Error: unable to open the file" << endl;
        return;
    }

    Node *temp = head;
    while (temp != nullptr)
    {
        file << temp->data << endl;
        temp = temp->next;
    }

    file.close();
}

void deleteList(Node *&head)
{
    if (head == nullptr)
    {
        cout << "list is empty" << endl;
        return;
    }

    Node *temp = head;
    while (temp != nullptr)
    {
        Node *NextNode = temp->next;
        delete temp;
        temp = NextNode;
    }
    cout << "list is deleted" << endl;
    head = nullptr;
}

Node *restoreFromFile(string Filename)
{
    string key;

    ifstream file;
    file.open(Filename);

    if (!file.is_open())
    {
        cout << "Error: unable to open the file" << endl;
        return nullptr;
    }

    Node *head = nullptr;
    Node *current = nullptr;

    while(getline(file, key))
    {
        Node* newNode = new Node(key);
        if(head == nullptr)
        {
            head = current = newNode;
        }
        else
        {
            current->next = newNode;
            newNode->prev = current;
            current = newNode;
        }
    }

    cout << "list is restored " << endl;

    file.close();

    return head;
}

int main()
{
    Node *head = createList();
    string key, filename;
    int pos;

    filename = "C:/Users/sabob/projects/ai_programming_playground_2024/ai_12/oleksandr_bobrovytskyi/epic_6/list.txt";

    addElement(head, "a", 0);
    addElement(head, "b", 1);
    addElement(head, "c", 2);
    addElement(head, "d", 3);
    cout << "Original list: " << endl;
    printList(head);

    cout << "Enter a key to delete elements containing it: ";
    cin >> key;

    deleteElement(head, key);

    cout << "List after deletion of elments containing key: " << endl;
    printList(head);

    cout << "Enter the position for inserting new element: ";
    cin >> pos;

    addElement(head, "e", pos);

    cout << "List after adding new element: " << endl;
    printList(head);

    writeToFile(head, filename);

    deleteList(head);
    printList(head);

    head = restoreFromFile(filename);
    printList(head);

    deleteList(head);

    return 0;
}
```

Завдання №2
Посилання на файл програми у пул-запиті GitHub https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/491/commits/d2422006e96ab147ac47ccee460642e5859e96c1#diff-3b57c2bb5741061a6519e22ac5e1379e7fafe1281ec6c3d8e0c0bfa42c17edfd

```cpp
#include<bits/stdc++.h>

using namespace std;

int main()
{
    int N, M, emptyRow;

    cin >> N >> M;

    if(N < 1 || N > 1000 || M < 1 || M > 1000)
    {
        return 1;
    }

    vector<string> cave(N);

    for(int i = 0; i < N; i++)
    {
        cin >> cave[i];
        if(cave[i].size() != M)
        {
            return 1;
        }
    }

    for(int j = 0; j < M; j++)
    {
        emptyRow = N - 1;
        for(int i = N - 1; i >= 0; i--)
        {
            if(cave[i][j] == 'X')
            {
                emptyRow = i - 1;
            }
            else if(cave[i][j] == 'S')
            {
                cave[i][j] = 'O';
                cave[emptyRow][j] = 'S';
                emptyRow--;
            }
        }
    }

    for(int i = 0; i < N; i++)
    {
       cout << cave[i] << endl;
    }

    return 0;
}
```

Завдання №3

Посилання на файл програми у пул-запиті GitHub https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/491/commits/d2422006e96ab147ac47ccee460642e5859e96c1#diff-27b62119ca1dc77ffd766b2e655acee3b78b24ebf41bfd297c04871cea4ee172

```cpp
#include <bits/stdc++.h>

using namespace std;

struct Node
{
    T data;
    Node *next;
    Node *prev;

    Node(T d) : data(d), next(nullptr), prev(nullptr) {}
};

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

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}

    void insert(vector<int> &data, int position)
    {
        Node *current = head;
        Node *previous = nullptr;
        int index = 0;

        for (int i = 0; i < position && current; i++)
        {
            previous = current;
            current = current->next;
        }

        for (int i = 0; i < data.size(); i++)
        {
            Node *NewNode = new Node(data[i]);

            if (!previous)
            {
                NewNode->next = head;
                if (head)
                {
                    head->prev = NewNode;
                }
                head = NewNode;
                previous = head;
            }
            else
            {
                NewNode->next = current;
                NewNode->prev = previous;
                previous->next = NewNode;
                if (current != nullptr)
                {
                    current->prev = NewNode;
                }

                previous = NewNode;
            }
            size++;
        }

        if (current == nullptr)
        {
            tail = previous;
        }
    }

    void erase(int index, int num)
    {
        Node *current = head;
        for (int i = 0; i < index && current != nullptr; i++)
        {
            current = current->next;
        }

        for (int i = 0; i < num && current != nullptr; i++)
        {
            Node *Next = current->next;

            if (current->prev != nullptr)
            {
                current->prev->next = Next;
            }
            else
            {
                head = Next;
            }

            if (Next != nullptr)
            {
                Next->prev = current->prev;
            }
            else
            {
                tail = current->prev;
            }

            delete current;
            current = Next;
            size--;
        }
    }
```

```cpp
int getSize()
    {
        return size;
    }

    int get(int index)
    {
        Node *current = head;
        for (int i = 0; i < index && current != nullptr; i++)
        {
            current = current->next;
        }
        return current->data;
    }

    void set(int index, int NewValue)
    {
        Node *current = head;
        for (int i = 0; i < index && current != nullptr; i++)
        {
            current = current->next;
        }

        current->data = NewValue;
    }

    friend ostream &operator<<(ostream &os, const DoublyLinkedList &list)
    {
        Node *current = list.head;
        while (current != nullptr)
        {
            os << current->data << " ";
            current = current->next;
        }
        return os;
    }

    ~DoublyLinkedList()
    {
        Node *current = head;
        while (current)
        {
            Node *nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

int main()
{
    DoublyLinkedList list;
    int num, index, value;
    int numOfActions;
    string command;

    cin >> numOfActions;

    for (int i = 0; i < numOfActions; i++)
    {
        cin >> command;

        if (command == "insert")
        {
            cin >> index;
            cin >> num;
            vector<int> values(num);
            for (int i = 0; i < num; i++)
            {
                cin >> values[i];
            }
            list.insert(values, index);
        }
        else if (command == "erase")
        {
            cin >> index >> num;
            list.erase(index, num);
        }
        else if (command == "size")
        {
            cout << list.getSize() << endl;
        }
        else if (command == "get")
        {
            cin >> index;
            cout << list.get(index) << endl;
        }
        else if (command == "set")
        {
            cin >> index >> value;
            list.set(index, value);
        }
        else if (command == "print")
        {
            cout << list;
        }
    }

    return 0;
}
```

Завдання №4
Посилання на файл програми у пул-запиті GitHub https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/491/commits/d2422006e96ab147ac47ccee460642e5859e96c1#diff-a925b2c8df7ea7e0a61e03a7a39027716f48b8c4485693df19fe6843d6f50ea8

```cpp
#include <bits/stdc++.h>

using namespace std;

template<typename T>
struct Node
{
    T data;
    Node *next;
    Node *prev;

    Node(T d) : data(d), next(nullptr), prev(nullptr) {}
};

template<typename T>
class DoublyLinkedList
{
private:
    Node<T> *head;
    Node<T> *tail;
    int size;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}

    void insert(vector<int> &data, int position)
    {
        Node<T> *current = head;
        Node<T> *previous = nullptr;
        int index = 0;

        for (int i = 0; i < position && current; i++)
        {
            previous = current;
            current = current->next;
        }

        for (int i = 0; i < data.size(); i++)
        {
            Node<T> *NewNode = new Node(data[i]);

            if (!previous)
            {
                NewNode->next = head;
                if (head)
                {
                    head->prev = NewNode;
                }
                head = NewNode;
                previous = head;
            }
            else
            {
                NewNode->next = current;
                NewNode->prev = previous;
                previous->next = NewNode;
                if (current != nullptr)
                {
                    current->prev = NewNode;
                }

                previous = NewNode;
            }
            size++;
        }

        if (current == nullptr)
        {
            tail = previous;
        }
    }

    void erase(int index, int num)
    {
        Node<T> *current = head;
        for (int i = 0; i < index && current != nullptr; i++)
        {
            current = current->next;
        }

        for (int i = 0; i < num && current != nullptr; i++)
        {
            Node<T> *Next = current->next;

            if (current->prev != nullptr)
            {
                current->prev->next = Next;
            }
            else
            {
                head = Next;
            }

            if (Next != nullptr)
            {
                Next->prev = current->prev;
            }
            else
            {
                tail = current->prev;
            }

            delete current;
            current = Next;
            size--;
        }
    }

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

```cpp

    int get(int index)
    {
        Node<T> *current = head;
        for (int i = 0; i < index && current != nullptr; i++)
        {
            current = current->next;
        }
        return current->data;
    }

    void set(int index, int NewValue)
    {
        Node<T> *current = head;
        for (int i = 0; i < index && current != nullptr; i++)
        {
            current = current->next;
        }

        current->data = NewValue;
    }

    friend ostream &operator<<(ostream &os, const DoublyLinkedList &list)
    {
        Node<T> *current = list.head;
        while (current != nullptr)
        {
            os << current->data << " ";
            current = current->next;
        }
        return os;
    }

    ~DoublyLinkedList()
    {
        Node<T> *current = head;
        while (current)
        {
            Node<T> *nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }
};

int main()
{
    DoublyLinkedList<int> list;
    int num, index, value;
    int numOfActions;
    string command;

    cin >> numOfActions;

    for (int i = 0; i < numOfActions; i++)
    {
        cin >> command;

        if (command == "insert")
        {
            cin >> index;
            cin >> num;
            vector<int> values(num);
            for (int i = 0; i < num; i++)
            {
                cin >> values[i];
            }
            list.insert(values, index);
        }
        else if (command == "erase")
        {
            cin >> index >> num;
            list.erase(index, num);
        }
        else if (command == "size")
        {
            cout << list.getSize() << endl;
        }
        else if (command == "get")
        {
            cin >> index;
            cout << list.get(index) << endl;
        }
        else if (command == "set")
        {
            cin >> index >> value;
            list.set(index, value);
        }
        else if (command == "print")
        {
            cout << list;
        }
    }

    return 0;
}
```

Завдання №5
Посилання на файл програми у пул-запиті GitHub https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/491/commits/d2422006e96ab147ac47ccee460642e5859e96c1#diff-62d98e65c7e565c169990cd295e5d45e87d8bd0902d2eaf7352dc44e9ee49613

```cpp
#include <bits/stdc++.h>

using namespace std;

struct Node
{
    int data;
    Node *next;

    Node(int value) : data(value), next(nullptr) {}
};

struct TreeNode
{
    int data;
    TreeNode *left;
    TreeNode *right;

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

void deleteList(Node *&head)
{
    Node *current = head;
    while (current != nullptr)
    {
        Node *next = current->next;
        delete current;
        current = next;
    }
    head = nullptr;
}

void deleteTree(TreeNode *&root)
{
    if (root == nullptr)
    {
        return;
    }

    deleteTree(root->left);
    deleteTree(root->right);

    delete root;
    root = nullptr;
}

Node *createList(vector<int> v)
{
    Node *head = new Node(v[0]);
    Node *current = head;

    for (int i = 1; i < v.size(); i++)
    {
        current->next = new Node(v[i]);
        current = current->next;
    }

    return head;
}

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

    while (current != nullptr)
    {
        next = current->next;
        current->next = previous;
        previous = current;
        current = next;
    }

    return previous;
}

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

bool compare(Node *h1, Node *h2)
{
    Node *temp1 = h1;
    Node *temp2 = h2;
    while (temp1 != nullptr && temp2 != nullptr)
    {
        if ((temp1->data != temp2->data) || (temp1 == nullptr && temp2 != nullptr) || (temp2 == nullptr && temp1 != nullptr))
        {
            return false;
        }

        temp1 = temp1->next;
        temp2 = temp2->next;
    }

    return true;
}

Node *add(Node *n1, Node *n2)
{
    Node *resultHead = nullptr;
    Node *resultTail = nullptr;
    Node *temp1 = n1;
    Node *temp2 = n2;
    int num = 0, carry = 0, sum = 0;

    while (temp1 != nullptr || temp2 != nullptr || carry != 0)
    {
        sum = carry;
        if (temp1 != nullptr)
        {
            sum += temp1->data;
            temp1 = temp1->next;
        }

        if (temp2 != nullptr)
        {
            sum += temp2->data;
            temp2 = temp2->next;
        }

        carry = sum / 10;

        num = sum % 10;

        Node *newNode = new Node(num);

        if (resultHead == nullptr)
        {
            resultHead = newNode;
        }
        else
        {
            resultTail->next = newNode;
        }
        resultTail = newNode;
    }

    return resultHead;
}
```

```
TreeNode *create_mirror_flip(TreeNode *root)
{
    if (root == nullptr)
    {
        return nullptr;
    }

    TreeNode *mirrored = new TreeNode(root->data);

    mirrored->left = create_mirror_flip(root->right);
    mirrored->right = create_mirror_flip(root->left);

    return mirrored;
}

void inorderTraversal(TreeNode *root)
{
    if (root == nullptr)
        return;

    inorderTraversal(root->left);
    cout << root->data << " ";
    inorderTraversal(root->right);
}

void tree_sum(TreeNode *root)
{
    if (root == nullptr)
        return;

    stack<TreeNode *> NodeStack;
    TreeNode *lastVisited = nullptr;
    TreeNode *current = root;
    int sum;

    while (!NodeStack.empty() || current != nullptr)
    {
        while (current != nullptr)
        {
            NodeStack.push(current);
            current = current->left;
        }

        TreeNode *topNode = NodeStack.top();

        if (topNode->right == nullptr || lastVisited == topNode->right)
        {
            NodeStack.pop();

            if (topNode->left != nullptr || topNode->right != nullptr)
            {
                sum = 0;
                sum += (topNode->left != nullptr) ? topNode->left->data : 0;
                sum += (topNode->right != nullptr) ? topNode->right->data : 0;
                topNode->data = sum;
            }

            lastVisited = topNode;
        }
        else
        {
            current = topNode->right;
        }
    }
}

int main()
{
    int n = 6;
    vector<int> values1 = {4, 6, 9};
    vector<int> values2 = {3, 2, 5};

    Node *head = createList(values1);

    cout << "Original list: ";
    printList(head);

    head = reverse(head);

    cout << "\nReversed list: ";
    printList(head);

    Node *head2 = createList(values2);

    cout << "\nResult of the comparison: " << compare(head, head2) << endl;

    Node *additionHead = add(head, head2);

    printList(additionHead);

    TreeNode *root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);

    cout << "\nInorder traversal of the original tree: " << endl;
    inorderTraversal(root);

    TreeNode *mirrored = create_mirror_flip(root);

    cout << "\nInorder traversal after mirroring: " << endl;
    inorderTraversal(mirrored);

    tree_sum(root);

    cout << "\nInorder traversal after finding sum of nodes: " << endl;
    inorderTraversal(root);

    deleteList(head);
    deleteList(head2);
    deleteList(additionHead);
    deleteTree(mirrored);
    deleteTree(root);

    return 0;
}
```

Завдання №6
Посилання на файл програми у пул-запиті GitHub https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/491/commits/d2422006e96ab147ac47ccee460642e5859e96c1#diff-6a55165f4a113b293060c3a9ec97f256da870412844800744ba7440d920acafb

```cpp
#include <bits/stdc++.h>
using namespace std;

int main()
{
    int n;
    cin >> n;

    vector<int> a(n), evens, odds;
    for (int i = 0; i < n; i++)
    {
        cin >> a[i];
        if (a[i] % 2 == 0)
        {
            evens.push_back(a[i]);
        }
        else
        {
            odds.push_back(a[i]);
        }
    }

    if (abs((int)evens.size() - (int)odds.size()) > 1)
    {
        cout << -1 << endl;
        return 0;
    }

    vector<int> result;
    bool evenTurn = evens.size() >= odds.size();
    while (!evens.empty() || !odds.empty())
    {
        if (evenTurn && !evens.empty())
        {
            result.push_back(evens.back());
            evens.pop_back();
        }
        else if (!evenTurn && !odds.empty())
        {
            result.push_back(odds.back());
            odds.pop_back();
        }
        evenTurn = !evenTurn;
    }

    for(int i = 0; i < result.size(); i++)
    {
        cout << result[i] << " ";
    }

    return 0;
}
```

4. Результати виконання завдань, тестування та фактично затрачений час:

Завдання №1 Деталі по виконанню і тестуванню програми

```
Original list:
a b c d
Enter a key to delete elements containing it: a
List after deletion of elments containing key:
b c d
Enter the position for inserting new element: 1
List after adding new element:
b e c d
List is deleted
list is empty
list is restored
b e c d
List is deleted
```

Блок №1 Результат виконання завдання
Час затрачений на виконання завдання – 90min

Завдання №2 Деталі по виконанню і тестуванню програми

```
5 5
SSOSS
OOOOO
SOOXX
OOOOS
OOSOO
OOOOO
OOOSS
OOOXX
SOOOO
SSSOS
```

Блок №1 Результат виконання завдання
Час затрачений на виконання завдання – 25min

Завдання №3 Деталі по виконанню і тестуванню програми

```
9
insert
0
5
1 2 3 4 5

insert
2
3
7 7 7

print
1 2 7 7 7 3 4 5
```

```
erase
1 2

print
1 7 7 3 4 5
size
6

get
3
3

set
3 13

print
1 7 7 13 4 5
```

Блок №1 Результат виконання завдання
Час затрачений на виконання завдання – 90min

Завдання №4 Деталі по виконанню і тестуванню програми

```
9
insert
0
5
1 2 3 4 5

insert
2
3
7 7 7

print
1 2 7 7 7 3 4 5
```

```
erase
1 2

print
1 7 7 3 4 5
size
6

get
3
3

set
3 13

print
1 7 7 13 4 5
```

Блок №1 Результат виконання завдання
Час затрачений на виконання завдання – 5min

Завдання №5 Деталі по виконанню і тестуванню програми

```
Original list: 4 6 9
Reversed list: 9 6 4
Result of the comparison: 0
2 9 9
Inorder traversal of the original tree:
4 2 5 1 6 3 7
Inorder traversal after mirroring:
7 3 6 1 5 2 4
Inorder traversal after finding sum of nodes:
4 9 5 22 6 13 7
```

Блок №1 Результат виконання завдання
Час затрачений на виконання завдання – 90min

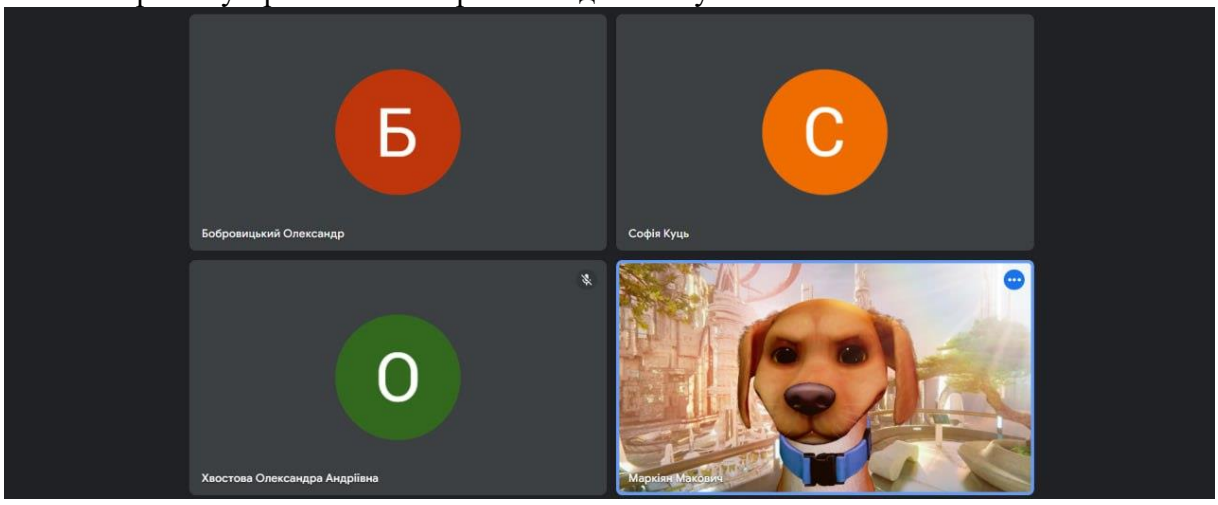
Завдання №6 Деталі по виконанню і тестуванню програми

```
3
4 7 47
47 4 7
```

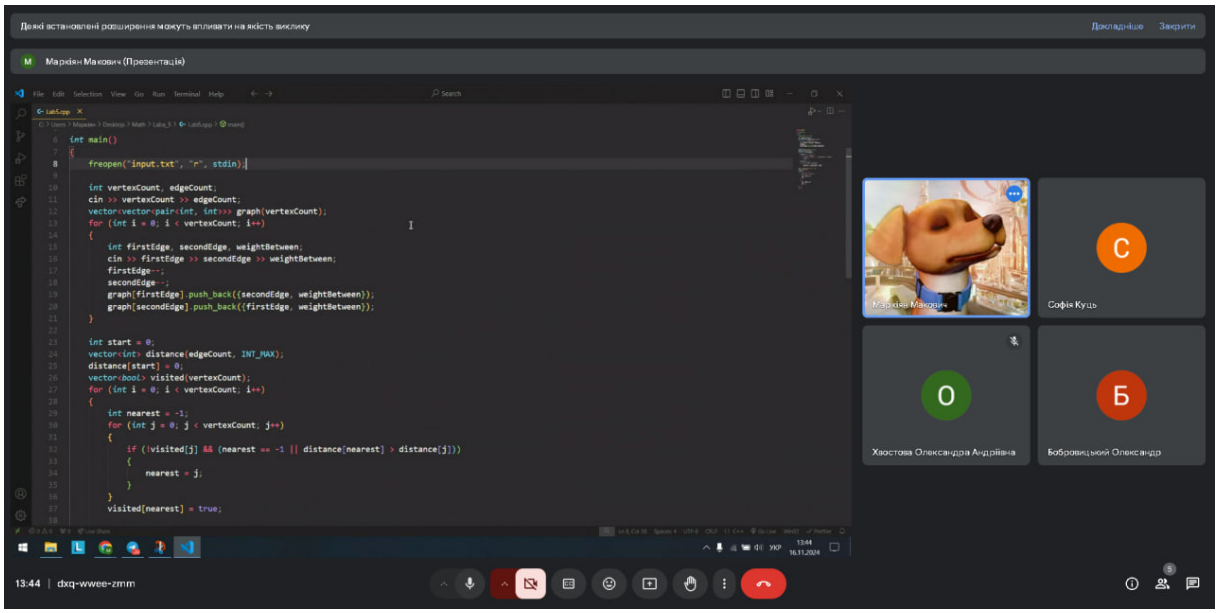
Блок №1 Результат виконання завдання
Час затрачений на виконання завдання – 20min

5. Кооперація з командою:

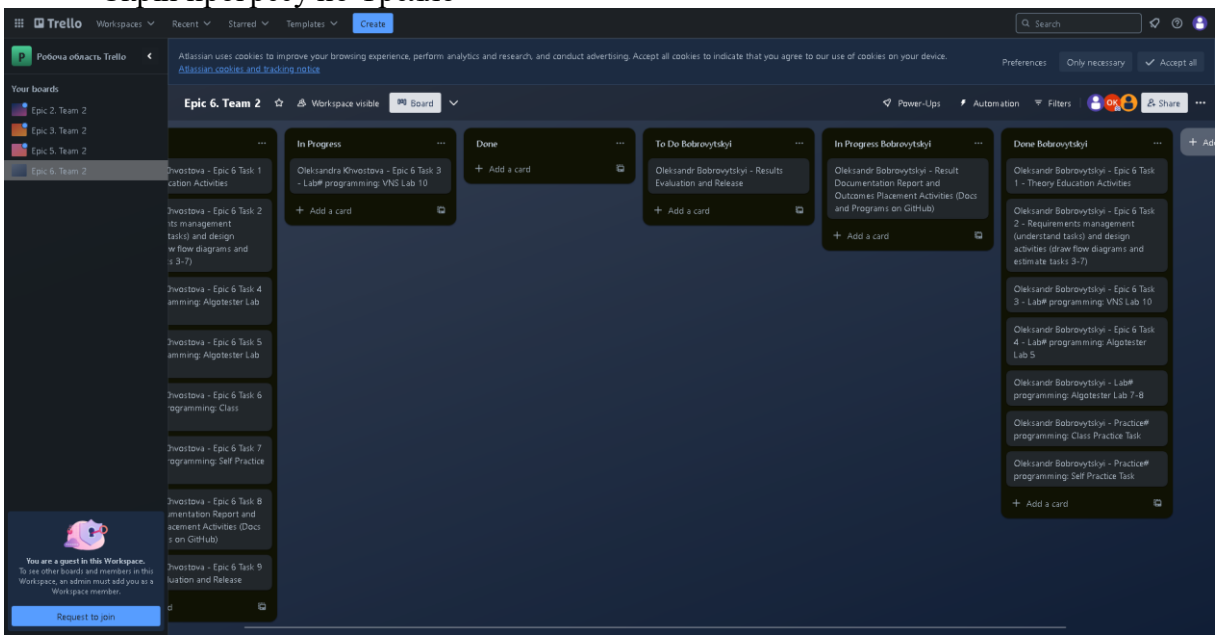
- Скрін з зустрічі по обговоренню задач Епіку 1



- Скрін з зустрічі по обговоренню задач Епіку 2



– Скрін прогресу по Трелло



Висновок: я ознайомився з темами лабораторної, опрацювати їх теоретично та навчитися використовувати отриманні знання для вирішення практичних задач.