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
Гаврих Юрій Дмитрович

Львів 2024

Тема роботи:

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

Мета роботи:

Ознайомитись з основними струкрурами данних , навчитись ітеруватись по ним , ознайомитись з алгоритмами їх обробки.

Теоретичні відомості:

Linked List in C++

Binary Tree in C++

Виконання роботи

Task 3 - Lab# programming: VNS Lab 10

Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити K елементів з кінця списку. Додати елемент після елемента із заданим ключем.

***Час виконання* ~ 1,5 год.**

Розв'язок:

```cpp
#include <iostream>

using namespace std;

class Node {
public:
    char data;
    Node *next;
    Node *prev;

    Node(char value) {
        data = value;
        next = nullptr;
        prev = nullptr;
    }
};

class LinkedList {

public:
    LinkedList() {
        size = 0;
        head = nullptr;
        tail = nullptr;
    }

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

    void init(int listSize, char values[]);

    void erase_end(int count);

    void insert(int key, char element);

    void print();

private:
    Node *head;
    Node *tail;
    int size;
};

void LinkedList::init(int listSize, char values[]) {

    Node *current = new Node(values[0]);
    head = current;
    for (int j = 1; j < listSize; j++) {
        Node *newNode = new Node(values[j]);
        newNode->prev = current;
        current->next = newNode;
        current = newNode;
        size++;
    }
    tail = current;
}

void LinkedList::insert(int key, char element) {
    Node *current = head;
    Node *prev = nullptr;
```

```
    while ((current) && (current->data != key)) {
        prev = current;
        current = current->next;
    }
    prev = current;
    current = current->next;
    if (prev) {
        Node *newNode = new Node(element);
        newNode->next = current;
        newNode->prev = prev;

        prev->next = newNode;

        if(prev==tail)
            tail=newNode;
        else current->prev = newNode;
    } else cout << "Key element not found";

}

void LinkedList::erase_end(int count) {

    Node *current = tail;

    for (int j = 0; j < count; j++) {
        Node *nextNode = current->prev;
        delete current;
        current = nextNode;
        size--;
    }
    tail=current;
    tail->next= nullptr;
}

void LinkedList::print() {
    printf("Head: %c \nTail: %c\n",head->data, tail->data);
    Node *current = head;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

int main() {

    LinkedList list;
    cout << "n:";
    int n, count;
    cin >> n;
    char elements[n], key, newElement;
    cout << "Start elements:";
    for (int i = 0; i < n; i++) {
        cin >> elements[i];
    }
    list.init(n, elements);

    list.print();

    cout << "Count elements to erase:";
    cin >> count;
    list.erase_end(count);
    list.print();

    cout << "Key and new element:";
    cin >> key >> newElement;
    list.insert(key, newElement);
    list.print();

    return 0;
}
```

Результат виконання:

```
n:5
Start elements:1 2 3 4 5
Head: 1
Tail: 5
1 2 3 4 5
Count elements to erase:2
Head: 1
Tail: 3
1 2 3
Key and new element:2 7
Head: 1
Tail: 3
1 2 7 3
```

Task 4 - Lab# programming: Algotester Lab 5

Час виконання **~ 1,5 год.**

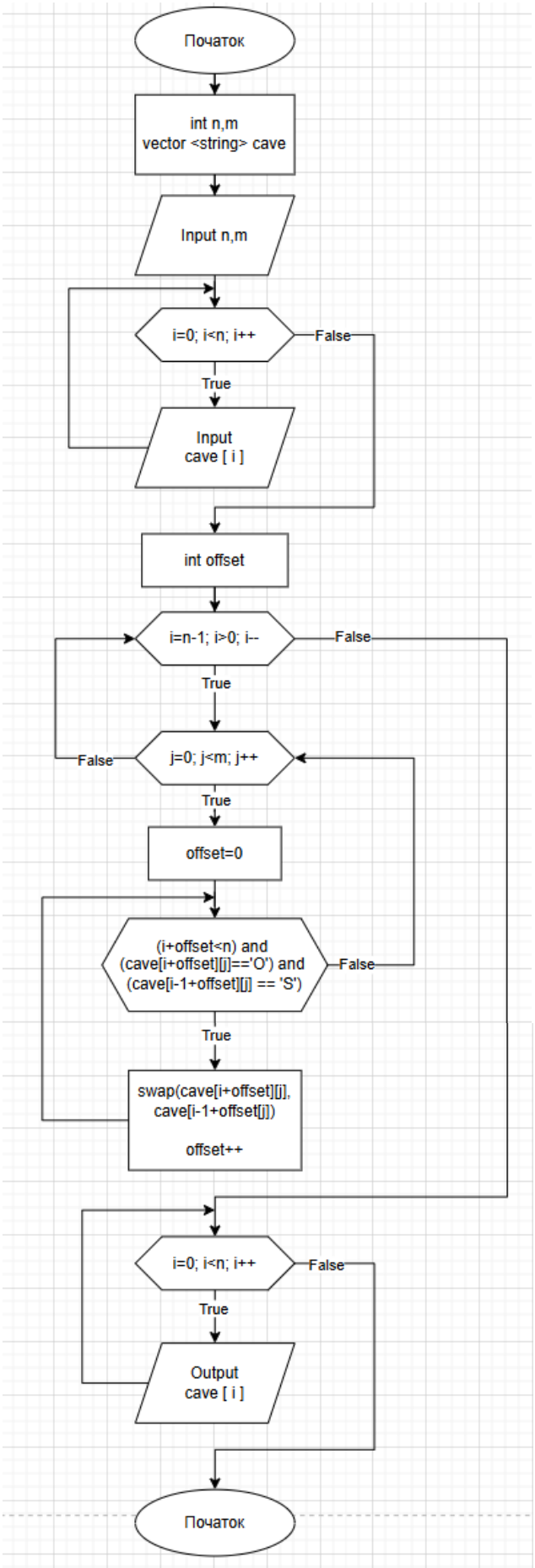
Розв'язок:

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

using namespace std;

int main() {
    int n, m;
    cin >> n >> m;
    vector <string> cave(n);
    for (int i = 0; i < n; i++) {
        cin>>cave[i];
    }
    int offset;
        for (int i = n - 1; i > 0; i--) {
            for (int j = 0; j < m; j++) {
                offset=0;
                while ( (i+offset<n) && (cave[i+offset][j] == 'O') && (cave[i - 1 + offset ][j] == 'S')) {
                    swap(cave[i+offset][j],cave[i-1+offset][j]);
                    offset++;
                }
            }
        }
    for (int i = 0; i < n; i++) {
        cout<<cave[i]<<endl;
    }
    return 0;
}
```

Результат виконання:

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

Блок-схема:


Початок
int n,m
vector <string> cave
Input n,m
i=0; i<n; i++
False
True
Input
cave [i]
int offset
i=n-1; i>0; i--
False
True
False
j=0; j<m; j++
True
offset=0
(i+offset<n) and
(cave[i+offset][j]=='O') and
(cave[i-1+offset][j] == 'S')
False
True
swap(cave[i+offset][j],
cave[i-1+offset[j])
offset++
i=0; i<n; i++
False
True
Output
cave [i]
Початок

Task 5 - Lab# programming: Algotester Lab 7-8

***Час виконання* ~ 4 год.**

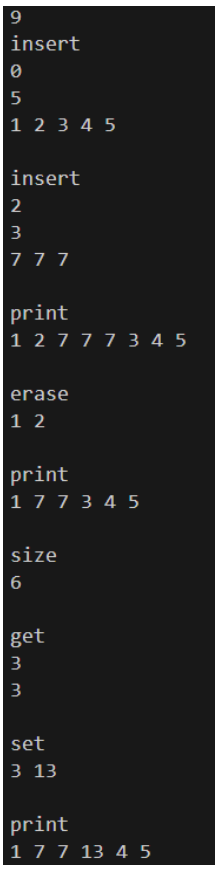
Розв'язок:

```cpp
#include <iostream>

using namespace std;

template<typename T>
class Node {
public:
    T data;
    Node<T> *next;
    Node<T> *prev;

    Node(T value) {
        data = value;
        next = nullptr;
        prev = nullptr;
    }
};

template<typename T>
class LinkedList {

public:
    LinkedList() {
        size = 0;
        head = nullptr;
        tail = nullptr;
    }
    ~LinkedList() {
        Node<T> *current = head;
        while (current) {
            Node<T> *nextNode = current->next;
            delete current;
            current = nextNode;
        }
    }

    void insert(int index, int listSize, T values[]);

    void erase(int index, int count);

    int get(int index);

    void set(int index, T value);

    int getSize();

    void print();

private:
    Node<T> *head;
    Node<T> *tail;
    int size;
};

template<class T>
void LinkedList<T>::insert(int index, int listSize, T values[]) {
    if (index < 0 || index > size || listSize <= 0) return;

    Node<T> *current = head;
    Node<T> *prev = nullptr;

    for (int i = 0; i < index; i++) {
        prev = current;
        current = current->next;
    }
```

```
    for (int j = listSize - 1; j >= 0; j--) {
        Node<T> *newNode = new Node(values[j]);
        newNode->next = current;
        if (current) {
            current->prev = newNode;
        } else {
            tail = newNode;
        }
        current = newNode;
        size++;
    }

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


template<class T>
void LinkedList<T>::erase(int index, int count) {
    if (index < 0 || index >= size || count <= 0) return;

    Node<T> *current = head;
    Node<T> *prev = nullptr;

    for (int i = 0; i < index; i++) {
        prev = current;
        current = current->next;
    }

    for (int j = 0; j < count && current; j++) {
        Node<T> *nextNode = current->next;
        if (nextNode) {
            nextNode->prev = current->prev;
        } else {
            tail = prev;
        }
        delete current;
        current = nextNode;
        size--;
    }

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



template<class T>
int LinkedList<T>::get(int index) {
    if (index < 0 || index >= size) {
        return 0;
    }

    Node<T> *current = head;

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

    return current->data;
}
```

```cpp
template<class T>
void LinkedList<T>::set(int index, T value) {
    if (index < 0 || index >= size) {
        return;
    }

    Node<T> *current = head;

    for (int i = 0; i < index; i++) {
        current = current->next;
    }
    current->data = value;
}

template<class T>
int LinkedList<T>::getSize() {
    return size;
}

template<class T>
void LinkedList<T>::print() {
    Node<T> *current = head;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

template<typename T>
void Choice(LinkedList<T> &list, const string &choice) {
    if (choice == "insert") {
        int index, n;
        cin >> index >> n;
        T elements[n];
        for (int i = 0; i < n; i++) {
            cin >> elements[i];
        }
        list.insert(index, n, elements);
    } else if (choice == "erase") {
        int index, count;
        cin >> index >> count;
        list.erase(index, count);
    } else if (choice == "get") {
        int index;
        cin >> index;
        cout << list.get(index) << endl;
    } else if (choice == "set") {
        int index;
        T value;
        cin >> index >> value;
        list.set(index, value);
    } else if (choice == "size") {
        cout << list.getSize() << endl;
    } else {
        list.print();
    }
}

int main() {

    LinkedList<int> list;

    int q;
    cin >> q;
    string choice;
    for (int i = 0; i < q; i++) {
        cin >> choice;
        Choice(list, choice);
    }

    return 0;
}
```

Результат виконання:



Task 6 - Practice# programming: Class Practice Task

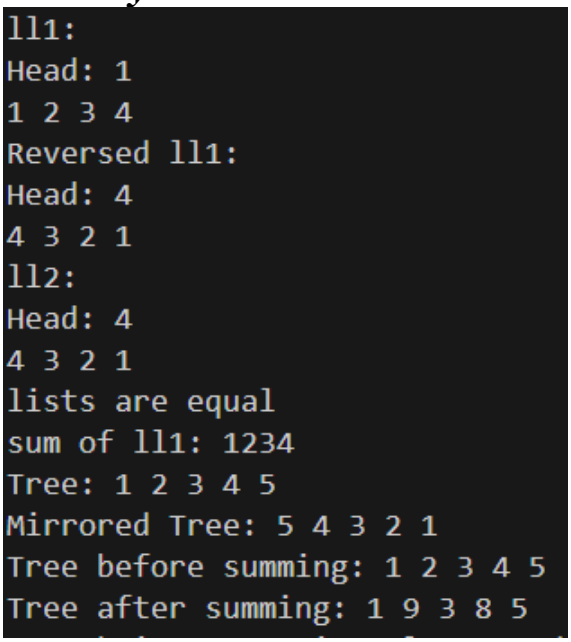
***Час виконання* ~ 2,5 год.**

```cpp
#include <iostream>
#include<math.h>

using namespace std;

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

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

class LinkedList {

public:
    Node *head;

    LinkedList() {
        size = 0;
        head = nullptr;
    }

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

    void init(int listSize, int values[]);

    void reverse();

    int add();

    void print();

private:

    int size;
};

void LinkedList::init(int listSize, int values[]) {

    Node *current = new Node(values[0]);
    head = current;
    for (int j = 1; j < listSize; j++) {
        Node *newNode = new Node(values[j]);
        current->next = newNode;
        current = newNode;
        size++;
    }
}

void LinkedList::reverse() {
    Node *prev = head;
    Node *current = prev->next;
    prev->next = nullptr;
    Node *nextNode = current->next;
    while (nextNode) {
        current->next = prev;
        prev = current;
        current = nextNode;
        nextNode = current->next;
    }
    current->next = prev;
    head = current;
}

int LinkedList::add() {
    int rez = 0;
    int digit = 0;
    Node *current = head;
    while (current) {
        rez += current->data * pow(10, digit);
        current = current->next;
        digit++;
    }
    return rez;
}
```

```
void LinkedList::print() {
    printf("Head: %d\n", head->data);
    Node *current = head;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

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

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

    TreeNode(int val) : value(val), left(nullptr), right(nullptr) {}
};
TreeNode* create_mirror_flip(TreeNode* root) {
    if (root == nullptr) return nullptr;

    TreeNode* new_root = new TreeNode(root->value);
    new_root->left = create_mirror_flip(root->right);
    new_root->right = create_mirror_flip(root->left);

    return new_root;
}

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

    int left_sum = 0;
    int right_sum = 0;

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

    if (root->left != nullptr) {
        left_sum += root->left->value;
    }
    if (root->right != nullptr) {
        right_sum += root->right->value;
    }


    if(!root->left && !root->right) return;
    root->value = left_sum + right_sum;
}

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

```cpp
int main() {
    //Linked List
    LinkedList list_1;
    int size_1 = 4;
    int elements[] = {1, 2, 3, 4};
    list_1.init(size_1, elements);
    cout << "ll1:" << endl;
    list_1.print();
    list_1.reverse();
    cout << "Reversed ll1:" << endl;
    list_1.print();

    LinkedList list_2;
    int size_2 = 4;
    int elements2[] = {4, 3, 2, 1};
    list_2.init(size_2, elements2);
    cout << "ll2:" << endl;
    list_2.print();

    bool is_equal = compare(list_1.head, list_2.head);
    if (is_equal)
        cout << "lists are equal" << endl;
    else
        cout << "Lists aren't equal" << endl;
    cout << "sum of ll1: " << list_1.add() << endl;

    //TREE

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

    cout << "Tree: ";
    printTree(root);
    cout << endl;

    TreeNode* mirroredRoot = create_mirror_flip(root);
    cout << "Mirrored Tree: ";
    printTree(mirroredRoot);
    cout << endl;

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

    cout << "Tree before summing: ";
    printTree(sumTreeRoot);
    cout << endl;

    tree_sum(sumTreeRoot);
    cout << "Tree after summing: ";
    printTree(sumTreeRoot);
    cout << endl;

    return 0;
}
```

Результат виконання:



Task 9 - Practice# programming: Self Practice Task

Час виконання 1 год

Algotester 2231 [All-Star]

В цій задачі потрібно зробити дерево зіркоподібної форми за мінімальну кількість операцій.

Розв'язок:

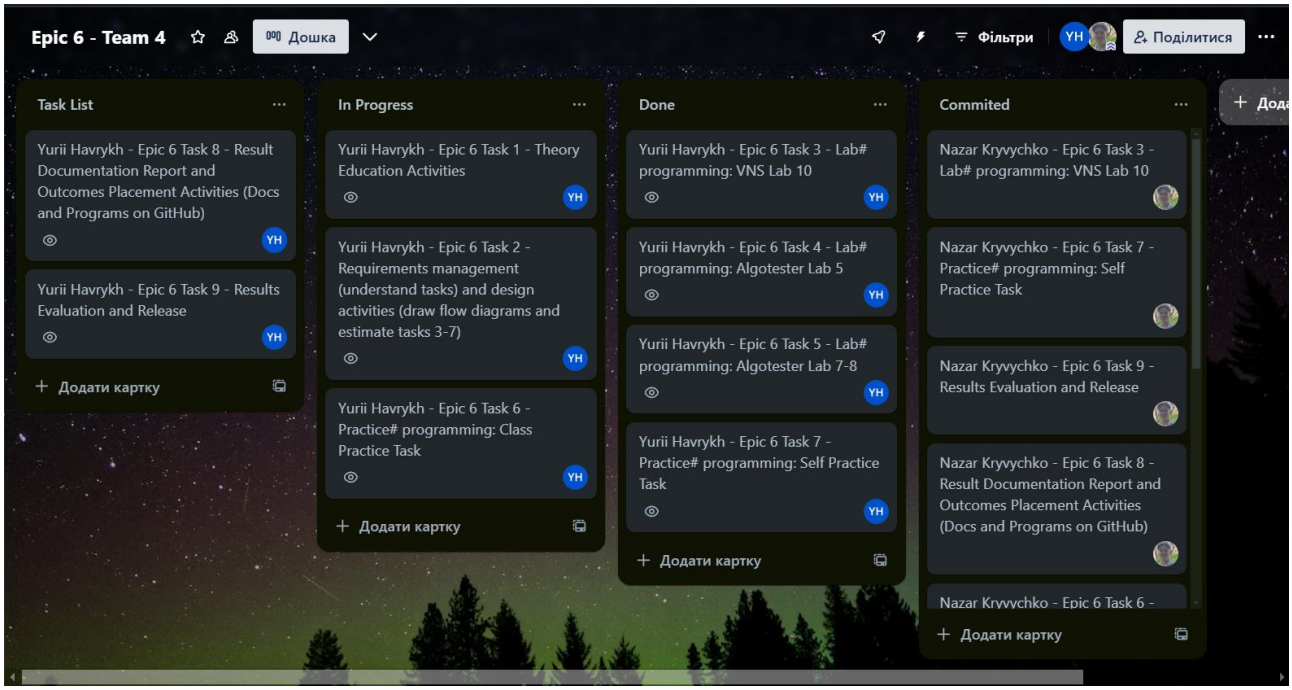
```cpp
#include<bits/stdc++.h>

using namespace std;

int find_vertex(int n, int vertex, const vector<vector<int>> &graph) {
    queue<int> q;
    vector<int> dist(n + 1, 1000);
    dist[vertex] = 0;
    for (auto i: graph[vertex]) {
        q.push(i);
        dist[i] = 1;
    }
    int result = 0;
    while (!q.empty()) {
        int cur = q.front();
        q.pop();
        for (auto i: graph[cur]) {
            if (dist[i] == 1000) {
                q.push(i);
                dist[i] = dist[cur] + 1;
                result++;
            }
        }
    }
    return (result);
}

void find_vertexes(int my_vertex, int n, int vertex, const vector<vector<int>> &graph) {
    queue<int> q;
    vector<int> dist(n + 1, 1000);
    dist[vertex] = 0;
    for (auto i: graph[vertex]) {
        q.push(i);
        dist[i] = 1;
    }
    int result = 0;
    while (!q.empty()) {
        int cur = q.front();
        q.pop();
        for (auto i: graph[cur]) {
            if (dist[i] == 1000) {
                q.push(i);
                dist[i] = dist[cur] + 1;
                cout << my_vertex << " " << cur << " " << i << endl;
            }
        }
    }
}

int main() {
    int n, x, y;
    cin >> n;
    vector<vector<int>> graph(n + 1);
    for (int i = 0; i < n - 1; i++) {
        cin >> x >> y;
        graph[x].push_back(y);
        graph[y].push_back(x);
    }
    int result = 1e7, vertex;
    for (int i = 1; i <= n; i++) {
        int rez = find_vertex(n, i, graph);
        if (result > rez) {
            result = rez;
            vertex = i;
        }
    }
    cout << result << endl;
    find_vertexes(vertex, n, vertex, graph);
}
```

Результат виконання:

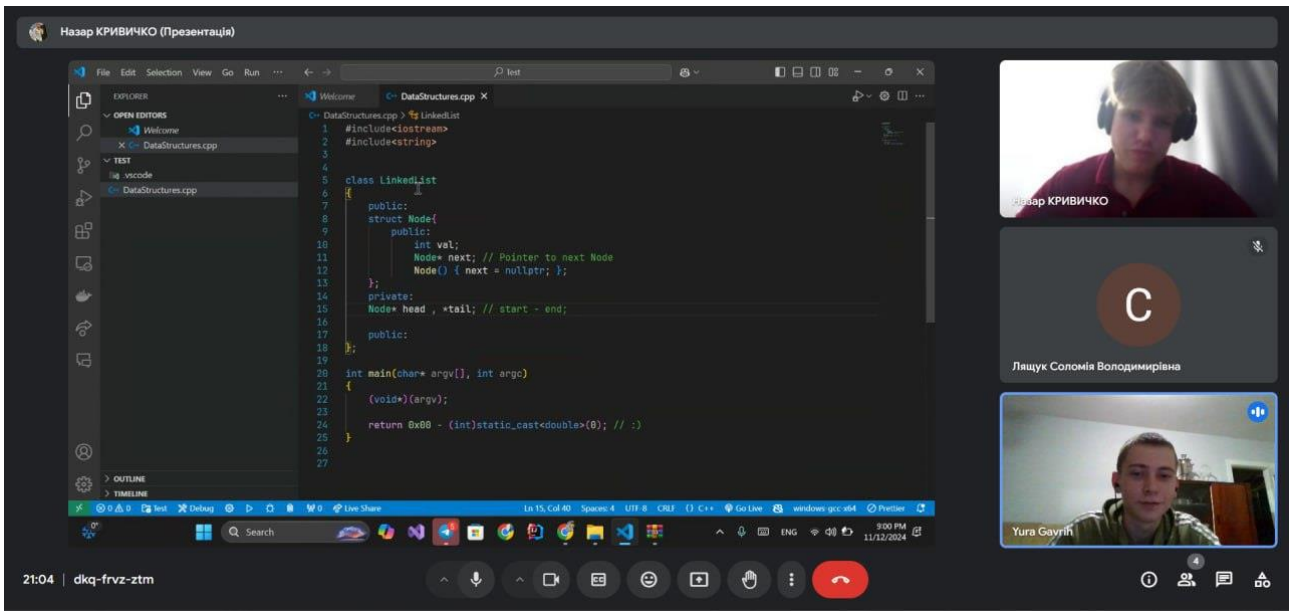
```
5
1 2
2 3
3 4
4 5
2
2 3 4
2 4 5
```

Task 10 - Result Documentation Report and Outcomes Placement Activities

Час виконання ~ 1,5 год.



Team meeting



Pull Request

Висновок:

В результаті виконання цієї роботи я закріпив знання стурктур данних та реалізував декілька варіантів ітерацій по них.