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
Бісюк Роман

Львів 2024

Тема лабораторної роботи:
Основи динамічних структур даних: стек, черга, зв'язний список, дерево.

Мета лабораторної роботи:
Метою цієї лабораторної роботи є освоєння основних принципів роботи з динамічними структурами даних, а також отримання навичок реалізації та використання таких структур, як стек, черга, зв'язний список і дерево. Я прагну зрозуміти, як динамічні структури дозволяють ефективніше використовувати пам'ять шляхом виділення ресурсів у динамічній області пам'яті (heap), а також які основні операції можна виконувати з кожною структурою. Я маю на меті навчитися обирати відповідну структуру даних для конкретної задачі, враховуючи її особливості та поведінку в пам'яті, а також зрозуміти алгоритми роботи з динамічними структурами, такі як додавання, видалення елементів і пошук. Це допоможе мені закласти основу для подальшого вивчення складніших структур даних та алгоритмів їх обробки.

Джерела:

-CS50 lectures and tasks about data structures and algorithms

-University lectures

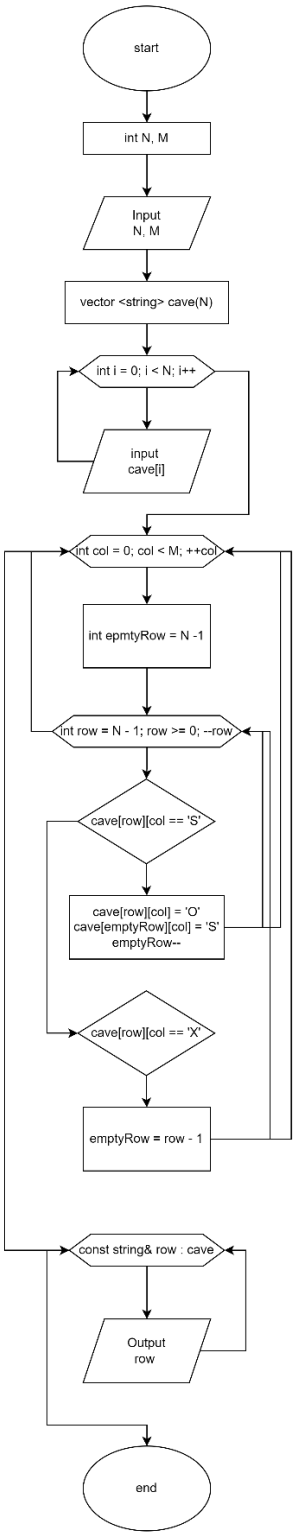
-aCode – data structures

-Google + ChatGPT for learning about different types of trees, stacks and queues with their implementations.

Виконання

Task 2 - Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7)

Time expected – 30m
Time spent – 30m



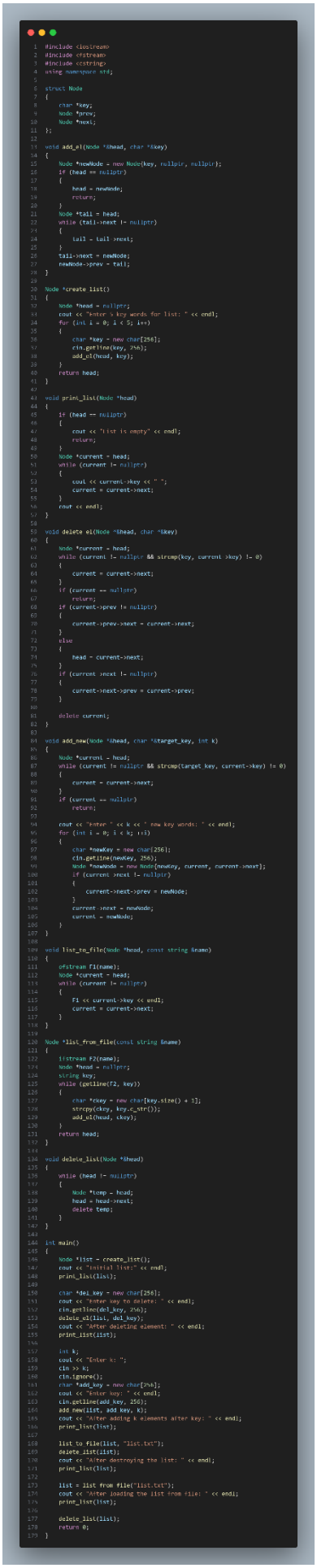
Task 3 - Lab# programming: VNS Lab 10

Time expected – 2hours
Time spent – 4 hours

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.

23. Запису в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим ключем. Додати К елементів після елемента із заданим ключем.



```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

struct Node
{
    char *key;
    Node *prev;
    Node *next;
};

void add_el(Node *&head, char *&key)
{
    Node *newNode = new Node(key, nullptr, nullptr);
    if (head == nullptr)
    {
        head = newNode;
        return;
    }
    Node *tail = head;
    while (tail->next != nullptr)
    {
        tail = tail->next;
    }
    tail->next = newNode;
    newNode->prev = tail;
}

Node *create_list()
{
    Node *head = nullptr;
    cout << "Enter 5 key words for list: " << endl;
    for (int i = 0; i < 5; i++)
    {
        char *key = new char[256];
        cin.getline(key, 256);
        add_el(head, key);
    }
    return head;
}

void print_list(Node *head)
{
    if (head == nullptr)
    {
        cout << "list is empty" << endl;
        return;
    }
    Node *current = head;
    while (current != nullptr)
    {
        cout << current->key << " ";
        current = current->next;
    }
    cout << endl;
}

void delete_el(Node *&head, char *&key)
{
    Node *current = head;
    while (current != nullptr && strcmp(key, current->key) != 0)
    {
        current = current->next;
    }
    if (current == nullptr)
        return;
    if (current->prev != nullptr)
    {
        current->prev->next = current->next;
    }
    else
    {
        head = current->next;
    }
    if (current->next != nullptr)
    {
        current->next->prev = current->prev;
    }

    delete current;
}

void add_new(Node *&head, char *&target_key, int k)
{
    Node *current = head;
    while (current != nullptr && strcmp(target_key, current->key) != 0)
    {
        current = current->next;
    }
    if (current == nullptr)
        return;

    cout << "Enter " << k << " new key words: " << endl;
    for (int i = 0; i < k; ++i)
    {
        char *newKey = new char[256];
        cin.getline(newKey, 256);
        Node *newNode = new Node{newKey, current, current->next};
        if (current->next != nullptr)
        {
            current->next->prev = newNode;
        }
        current->next = newNode;
        current = newNode;
    }
}

void list_to_file(Node *head, const string &name)
{
    ofstream f1(name);
    Node *current = head;
    while (current != nullptr)
    {
        f1 << current->key << endl;
        current = current->next;
    }
}

Node *list_from_file(const string &name)
{
    ifstream f2(name);
    Node *head = nullptr;
    string key;
    while (getline(f2, key))
    {
        char *ckey = new char[key.size() + 1];
        strcpy(ckey, key.c_str());
        add_el(head, ckey);
    }
    return head;
}

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

int main()
{
    Node *list = create_list();
    cout << "Initial list:" << endl;
    print_list(list);

    char *del_key = new char[256];
    cout << "Enter key to delete: " << endl;
    cin.getline(del_key, 256);
    delete_el(list, del_key);
    cout << "After deleting element: " << endl;
    print_list(list);

    int k;
    cout << "Enter k: ";
    cin >> k;
    cin.ignore();
    char *add_key = new char[256];
    cout << "Enter key: " << endl;
    cin.getline(add_key, 256);
    add_new(list, add_key, k);
    cout << "After adding k elements after key: " << endl;
    print_list(list);

    list_to_file(list, "list.txt");
    delete_list(list);
    cout << "After destroying the list: " << endl;
    print_list(list);

    list = list_from_file("list.txt");
    cout << "After loading the list from file: " << endl;
    print_list(list);

    delete_list(list);
    return 0;
}
```

```
Enter 5 key words for list:
one
two
three
four
five
Initial list:
one two three four five
Enter key to delete:
two
After deleting element:
one three four five
Enter k: 3
Enter key:
three
Enter 3 new key words:
NewWorld1
NewWorld2
NewWorld3
After adding k elements after key:
one three NewWorld1 NewWorld2 NewWorld3 four five
After destroying the list:
List is empty
After loading the list from file:
one three NewWorld1 NewWorld2 NewWorld3 four five
```

Task 4 - Lab# programming: Algotester Lab 5

Time expected - 30 minutes
time spent - 45 minutes

Lab 5v2

Limits: 1 sec., 256 MiB

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.

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a few seconds ago	C++ 23	Accepted	0.025	1.855	View

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

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

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

    return 0;
}
```

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

Time expected – 2 hours

time spent – 4 hours

Lab 78v1

Limits: 2 sec., 256 MiB

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**:
  Ідентифікатор - $insert$
  Ви отримуєте ціле число $index$ елемента, на місце якого робити вставку.
  Після цього в наступному рядку рядку написане число N - розмір списку, який треба вставити.
  У третьому рядку N цілих чисел - список, який треба вставити на позицію $index$.
- **Видалення**:
  Ідентифікатор - $erase$
  Ви отримуєте 2 цілих числа - $index$, індекс елемента, з якого почати видалення та n - кількість елементів, яку треба видалити.
- **Визначення розміру**:
  Ідентифікатор - $size$
  Ви не отримуєте аргументів.
  Ви виводите кількість елементів у списку.
- **Отримання значення** i-го елементу
  Ідентифікатор - get
  Ви отримуєте ціле число - $index$, індекс елемента.
  Ви виводите значення елемента за індексом.
- **Модифікація значення** i-го елементу
  Ідентифікатор - set
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення.
- **Вивід списку на екран**
  Ідентифікатор - $print$
  Ви не отримуєте аргументів.
  Ви виводите усі елементи списку через пробіл.
  Реалізувати використовуючи перегрузку оператора $<<$

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
2 hours ago	C++ 23	Accepted	0.008	1.348	View
2 hours ago	C++ 23	Accepted	0.008	1.414	View

```cpp
#include <iostream>

using namespace std;

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

    Node(int 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 insert(int index, int listSize, int values[]);

    void erase(int index, int count);

    int get(int index);

    void set(int index, int value);

    int getSize();

    void print();

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

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

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

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

    for (int j = listSize - 1; j >= 0; j--) {
        Node *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;
    }
}

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

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

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

    for (int j = 0; j < count && current; j++) {
        Node *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;
    }
}

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

    Node *current = head;

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

    return current->data;
}

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

    Node *current = head;

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

    current->data = value;
}

int LinkedList::getSize() {
    return size;
}

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

void Choice(LinkedList &list, const string &choice) {
    if (choice == "insert") {
        int index, n;
        cin >> index >> n;
        int 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;
        int value;
        cin >> index >> value;
        list.set(index, value);
    } else if (choice == "size") {
        cout << list.getSize() << endl;
    } else {
        list.print();
    }
}

int main() {

    LinkedList list;

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

    return 0;
}
```

```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;
}

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;
}
```

Task 6 - Practice# programming: Class Practice Task

Time expected – 3 hours

Time spent – 5 hours

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 $\Longrightarrow$ $9 \rightarrow 7 \rightarrow 3$);
- 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

```
Linked List with no changes: 7 9 11 12
Reversed Linked List: 12 11 9 7
Are lists equal? No
Are lists equal after changes? No
Sum: 6 8 8
Original Tree: 2 3 5 1 4
Mirrored Tree: 4 1 5 3 2
Tree before summing subtrees: 4 2 5 1 3
Tree after: 4 9 5 5 3
```

```cpp
#include <iostream>

using namespace std;

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

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

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

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 dummy(0);
    Node *current = &dummy;
    int carry = 0;

    while (n1 != nullptr || n2 != nullptr || carry)
    {
        int sum = carry;
        if (n1 != nullptr)
        {
            sum += n1->data;
            n1 = n1->next;
        }
        if (n2 != nullptr)
        {
            sum += n2->data;
            n2 = n2->next;
        }
        carry = sum / 10;
        current->next = new Node(sum % 10);
        current = current->next;
    }
    return dummy.next;
}

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;

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

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

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

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

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

int main()
{
    Node *head = new Node(7);
    head->next = new Node(5);
    head->next->next = new Node(11);
    head->next->next->next = new Node(12);

    cout << "Linked List with no changes: ";
    printlist(head);

    head = reverse(head);
    cout << "Reversed Linked List: ";
    printlist(head);

    Node* list1 = new Node(2);
    list1->next = new Node(6);
    list1->next->next = new Node(7);

    Node* list2 = new Node(3);
    list2->next = new Node(5);
    list2->next->next = new Node(8);

    cout << "Are lists equal? " << (compare(list1, list2) ? "Yes" : "No") << endl;

    list2->next->next->data = 4;
    cout << "Are lists equal after changes? " << (compare(list1, list2) ? "Yes" : "No") << endl;

    Node* num1 = new Node(5);
    num1->next = new Node(6);
    num1->next->next = new Node(7);

    Node* num2 = new Node(4);
    num2->next = new Node(9);
    num2->next->next = new Node(1);

    Node* sum = add(num1, num2);
    cout << "Sum: ";
    printlist(sum);

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

    cout << "Original tree: ";
    printTree(root);
    cout << endl;

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

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

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

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

    return 0;
}
```

Task 7 - Practice# programming: Self Practice Task

time expected – 1 Hour

time spent – 50 minutes

Lab 5v3

Limits: 1 sec., 256 MiB

У вас є карта гори розміром $N \times M$.

Також ви знаєте координати $\{x, y\}$, у яких знаходиться вершина гори.

Ваше завдання - розмалювати карту таким чином, щоб найнижча точка мала число 0, а пік гори мав найбільше число.

Клітинкі які мають суміжну сторону з вершиною мають висоту на один меншу, суміжні з ними і не розфарбовані мають ще на 1 меншу висоту і так далі.

Input

У першому рядку 2 числа N та M - розміри карти

у другому рядку 2 числа x та y - координати піку гори

Output

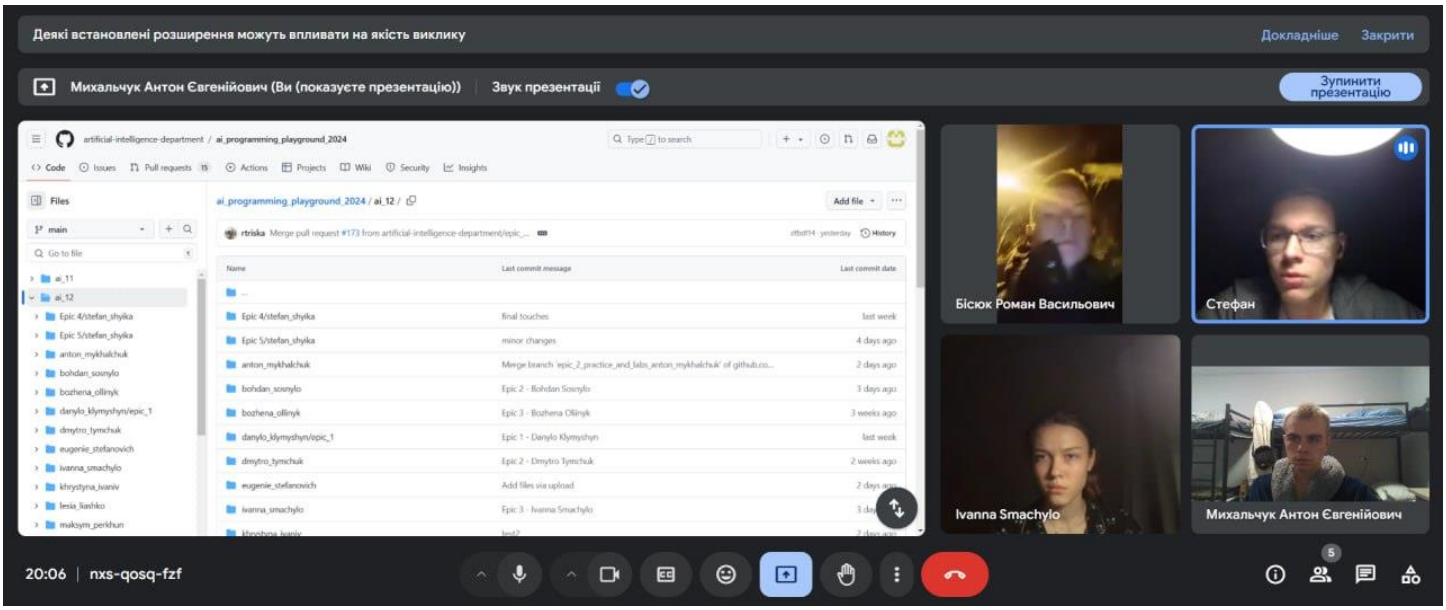
N рядків по M елементів в рядку через пробіл - висоти карти.

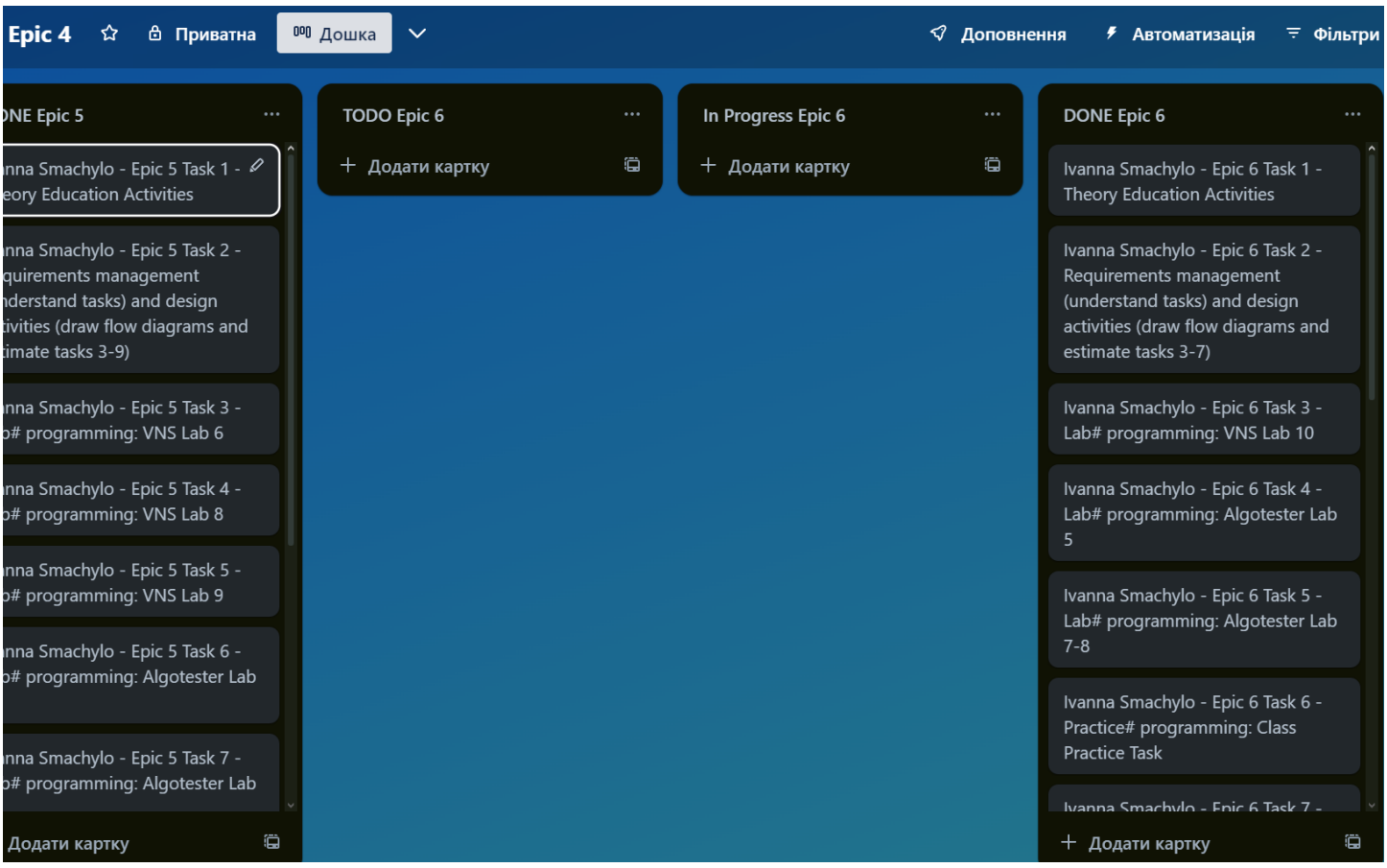
Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a minute ago	C++ 23	Accepted	0.117	6.695	View
a minute ago	C++ 23	Wrong Answer 1	0.002	0.930	View

```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;
int main() {
    int N, M, x, y;
    cin >> N >> M;
    cin >> x >> y;

    const int dx[] = { 1, -1, 0, 0 };
    const int dy[] = { 0, 0, 1, -1 };

    x--; y--;

    vector<vector<int>> height(N, vector<int>(M, -1));
    queue<pair<int, int>> q;
    q.push({x, y});
    height[x][y] = 0;

    while (!q.empty()) {
        auto el = q.front();
        q.pop();

        for (int i = 0; i < 4; ++i) {
            int nx = el.first + dx[i];
            int ny = el.second + dy[i];

            if (nx >= 0 && nx < N && ny >= 0 && ny < M && height[nx][ny] == -1) {
                height[nx][ny] = height[el.first][el.second] + 1;
                q.push({ nx, ny });
            }
        }
    }
    int maxHeight = 0;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            maxHeight = max(maxHeight, height[i][j]);
        }
    }
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            cout << maxHeight - height[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

PULL

MEETS: Обговорили проблемні питання пофікисили проблему з додаванням комітів саги до епіків. Оновили трелло





Висновок: завдяки цій роботі я на практиці зрозумів, як працюють основні динамічні структури даних та їхні ключові операції, і тепер усвідомлюю, чому їх використовують у завданнях, що вимагають гнучкого управління пам’яттю.