Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра систем штучного інтелекту

Звіт

про виконання лабораторних та практичних робіт блоку № 6

На тему: «Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.»

з дисципліни: «Основи програмування»

до:

ВНС Лабораторної Роботи № 10
Алготестер Лабораторної Роботи № 5
Алготестер Лабораторної Роботи № 7-8
Практичних Робіт до блоку № 6

Виконав:
Студент групи ШІ-11
Фарина Арсеній Петрович

Львів 2024

Тема роботи:

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

Мета роботи:

Застосувати на практиці вивчений матеріал, реалізувати Linked List (Однозв'язний список), бінарне дерево.

Теоретичні відомості:

- Тема №1: Основи Динамічних Структур Даних.
- Тема №2: Стек.
- Тема №3: Черга.
- Тема №4: Зв'язні списки.
- Тема №5: Дерева.
- Тема №6: Алгоритми Обробки Динамічних Структур.

1) Індивідуальний план опрацювання теорії:

- Тема №1: Основи Динамічних Структур Даних:
  - Джерела інформації:
    - Статті.

    https://www.youtube.com/watch?v=NyOjKd5Qruk&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=58
  - Що опрацьовано:
    - Вступ до динамічних структур.
    - Виділення пам'яті для структур даних (stack і heap)
    - Приклади простих динамічних структур

    Запланований час на вивчення 40 хвилин

    Витрачений час 40 хвилин.
- Тема №2: Стек:
  - Джерела інформації:
    - Статті.

    https://acode.com.ua/urok-111-stek-i-kupa/

    https://www.youtube.com/watch?v=ZYvYISxaNL0&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=141
  - Що опрацьовано:
    - Визначення та властивості стеку
    - Операції push, pop, top: реалізація та використання
    - Переповнення стеку

    Запланований час на вивчення 2 години.

    Витрачений час 2 години.

- Тема №3: Черга:
  - Джерела інформації:
    - Статті.

    https://www.youtube.com/watch?v=Yhw8NbjrSFA&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=142
  - Що опрацьовано
    - Визначення та властивості черги
    - Операціїї dequeue, front: реалізація та застосування
    - Приклади використання черги: обробка подій, алгоритми планування
    - Розширення функціоналу черги: пріоритети черги

Запланований час на вивчення 2 години.
Витрачений час 2 години.

- Тема №4: Зв’язні списки:
  - Джерела інформації:
    - Статті.

https://www.youtube.com/watch?v=-25REjF_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=139
https://www.youtube.com/watch?v=QLzu2-_QFoE&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=140

- Що опрацьовано
  - Визначення однозв’язного та двозв’язного списку
  - Принципи створення нових вузлів, втсавка між існуючими, видалення, створення кільця (circular linked list)
  - Основні операції: обхід списку, пошук, доступ до елементів та об’єднання списків
  - Приклади використання списків: управління пам’яттю, FIFO та LIFO структури.

Запланований час на вивчення 2 години.
Витрачений час 2 години.

- Тема № 5: Дерева:
  - Джерела інформації:
    - Статті.

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

- Що опрацьовано
  - Вступ до структури даних “дерево”: визначення, типи
  - 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

Запланований час на вивчення 2 години.
Витрачений час 2 години.

- Тема №6: Алгоритми Обробки Динамічних Структур:
  - Джерела інформації:
    - Статті.

https://www.youtube.com/watch?v=mnwDpO4zqLA&t=433s

- Що опрацьовано
  - Основи алгоритмічних патернів: ітеративні, рекурсивні
  - Алгоритми пошуку, сортування даних, додавання та видалення елементів

Запланований час на вивчення 2 години.
Витрачений час 2 години.

Також користувався Chat GPT який давав відповіді на конкретні питання по коду та теорії.

Виконання роботи:

1. Опрацювання завдання до програм.

Завдання №1

VNS LAB 10 – TASK 1 (VARIANT 2)

Написати програму, у якій створюються динамічні структури й виконати їхню обробку у відповідності зі своїм варіантом.

Для кожного варіанту розробити такі функції:
1. Створення списку.
2. Додавання елемента в список (у відповідності зі своїм варіантом).
3. Знищення елемента зі списку (у відповідності зі своїм варіантом).
4. Друк списку.
5. Запис списку у файл.
6. Знищення списку.
7. Відновлення списку з файлу.
Записи в лінійному списку містять ключове поле типу int. Сформувати однонаправлений список. Знищити з нього елемент із заданим ключем, додати елемент перед елементом із заданим ключем.

Завдання №2

ALGOTESTER LAB 5 (VARIANT 3)

У вас є карта гори розміром N × 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.
Вхідні дані
У першому рядку 2 числа N та M - розміри карти
у другому рядку 2 числа x та y - координати піку гори
Вихідні дані
N рядків по M елементів в рядку через пробіл - висоти карти.

Завдання №3

ALGOTESTER LAB 7-8 (VARIANT 1)

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 і-го елементу
Ідентифікатор - get
Ви отримуєте ціле число - index, індекс елемента.
Ви виводите значення елемента за індексом.

• Модифікація значення і-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 <<

Вхідні дані
Ціле число Q - кількість запитів.
У наступних рядках Q запитів у зазначеному в умові форматі.

Вихідні дані
Відповіді на запити у зазначеному в умові форматі.

Примітки
Гарантується, що усі дані коректні. Виходу за межі списку або розмір, більший ніж розмір
списку недопустимі.
Індекси починаються з нуля.

Завдання №4

Задача №1 - Реверс списку (Reverse list)

Реалізувати метод реверсу списку: Node* reverse(Node *head);
Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати метод реверсу;

- реалізувати допоміжний метод виведення вхідного і обернутого списків;

Мета задачі

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

Задача №2 - Порівняння списків

bool compare(Node *h1, Node *h2);

Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати функцію, яка ітеративно проходиться по обох списках і порівнює дані в кожному вузлі;
- якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.

Мета задачі

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

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.

Мета задачі

Розуміння операцій зі структурами даних: це завдання унаочнює практичне використання списка для обчислювальних потреб. Арифметичні операції з великими числами це окремий клас задач, для якого використання списків допомагає обійти обмеження у представленні цілого числа сучасними комп'ютерами.

Задача №4 - Віддзеркалення дерева

TreeNode *create_mirror_flip(TreeNode *root);

Умови задачі:

- використовувати цілі числа для значень у вузлах дерева
- реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева
- функція повертає нове дерево, передане в функцію дерево не модифікується

Мета задачі

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

Задача №5 - Записати кожному батьківському вузлу суму підвузлів

void tree_sum(TreeNode *root);
Умови задачі:

- використовувати цілочисельні значення у вузлах дерева;
- реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів
- вузол-листок не змінює значення
- значення змінюються від листків до кореня дерева

Мета задачі

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

Завдання №5
SELF PRACTICE WORK ALGOTESTER

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.

Input

У першому рядку 2 цілих числа N та M - висота та ширина печери
У N наступних рядках стрічка row_i яка складається з N цифер - i-й рядок матриці, яка відображає стан печери до землетрусу.

Output

N рядків, які складаються з стрічки розміром M - стан печери після землетрусу.

2. Вимоги та планувальна оцінка часу виконання завдань:

Програма №1

- Важливі деталі для реалізації програми.
- Усі операції з однозв'язним списком передбачають використання динамічної пам’яті (new і delete), тому важливо уникати витоків пам’яті, видаляючи вузли після видалення або очищення списку. Використовувати біблотеку fstream для запису даних у файл.
- Плановий час на реалізацію 2.5 години.

Програма №2

- Блок – схема

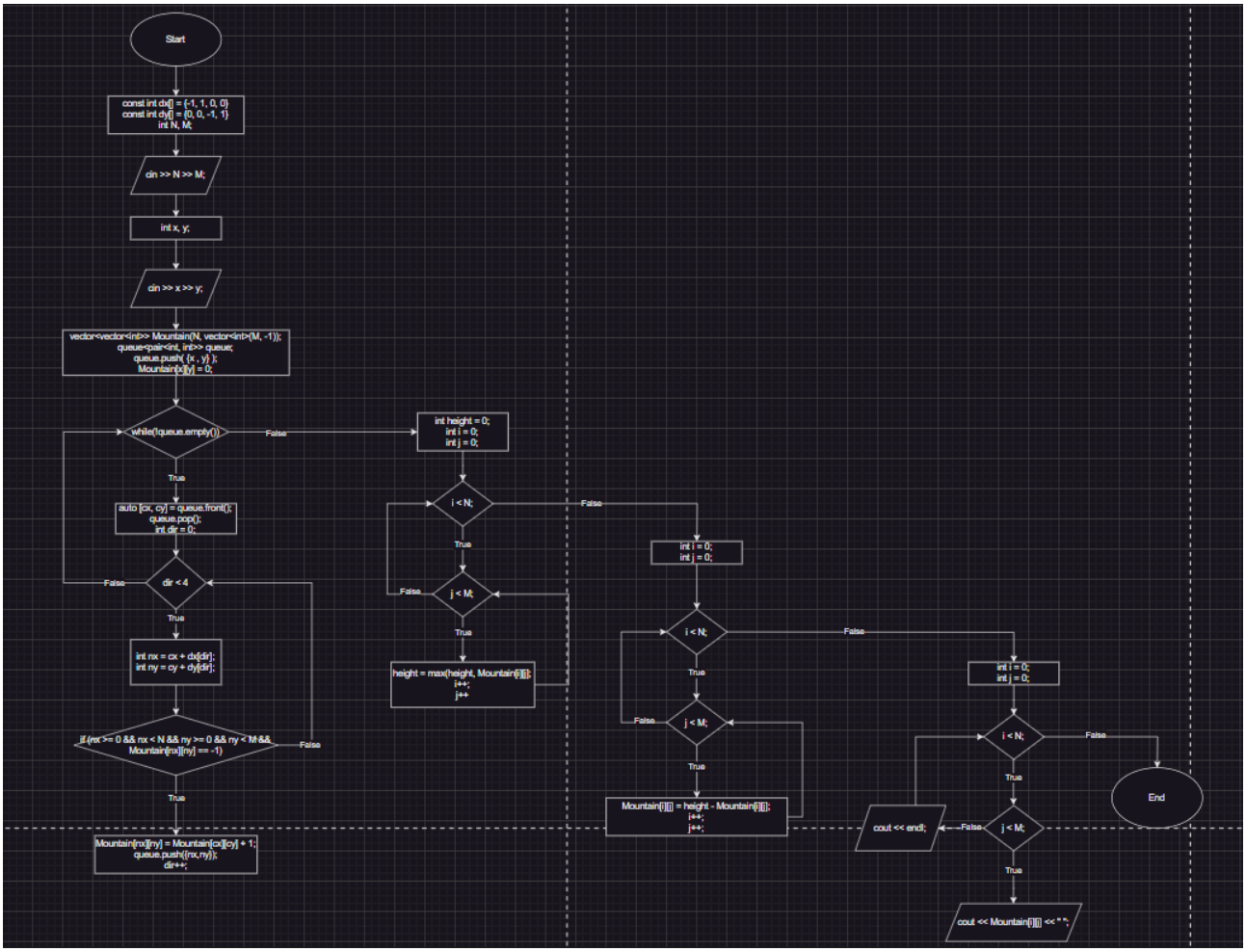


Рисунок 2.1. Блок – схема до програми 1

- Важливі деталі для реалізації програми.
- Використовувати BFS алгоритм пошуку вузлів для того щоб поширювати значення рівномірно від початкової точки в усі напрямки (як хвиля), а також для того щоб знайти найкоротшу відстань або кількість кроків до кожної точки з початкової точки.
- Плановий час на реалізацію 2 години.

Програма №3

- Важливі деталі для реалізації програми.
- Зрозуміти, що таке Linked List (Однозв’язний список) і навчитися його реалізовувати за допомогою вказівників та структури і перетворити це все у шаблон класу.
- Плановий час на реалізацію 5 годин.

Програма №4

- Важливі деталі для реалізації програми.
- Зрозуміти, що таке Бінарне дерево пошуку та навчитися його реалзовувати.
- Плановий час на реалізацію 5 годин.

Програма №5

- Важливі деталі для реалізації програми.
- Використовувати для реалізації матрицю чарів(вектор у векторі) а також цикли, для того щоб проходитися по стовпцях та рядках матриці.
- Плановий час на реалізацію 2 години.

3. Код програм з посиланням на зовнішні ресурси та фактично затрачений час:

Завдання №1

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

using namespace std;

struct Node{
    int key;
    Node* next;
};

// 1. Create list
Node* createList() {
    Node* head = new Node{10, NULL};

    Node* second = new Node{20, NULL};
    head->next = second;

    Node* third = new Node{30, NULL};
    second->next = third;

    return head;
}

Node* list = createList();


// 2. Add element before element with key
void addBeforeTheKey(Node*& head, int targetKey, int newKey){
    Node* newNode = new Node {newKey, NULL};

    if (!head){
        cout << "List is empty. Adding is impossible." << endl;
        delete newNode;
        return;
    }

    //search for a node before a node with a given key
    Node* current = head;
    while (current->next && current->next->key != targetKey){
        current = current->next;
    }

    if (current->next){
        newNode->next = current->next;
        current->next = newNode;
    }
    else {
        cout << "Element with key " << targetKey << " isn't found." << endl;
        delete newNode;
    }
}

// 3. Killing element with specified key
void killElementWithKey(Node*& head, int keyForDelete){

    if (!head){
        cout << "List is empty. Adding is impossible." << endl;
        return;
    }

    if (head->key == keyForDelete){
        Node* temp = head;
        head = head->next;
        delete temp;
        return;
    }

    //search for a node before a node with a given key
    Node* current = head;
    while (current->next && current->next->key != keyForDelete){
        current = current->next;
    }

    if (current->next){
        Node* temp = current->next;
        current->next = current->next->next;
        delete temp;
    }
    else{
        cout << "Element with key " << keyForDelete << " isn't found." << endl;
    }
}
```

```cpp

// 4. Print list
void printList(Node*& head){

    if (!head){
        cout << "List is empty. Printing is impossible. " << endl;
        return;
    }

    Node* current = head;
    while (current) {
        cout << current->key << " -> ";
        current = current->next;
    }
    cout << "Null" << endl;

}

// 5. Writing to file
void writingToFile(Node*& head){

    ofstream outFile("file.txt");

    if(!outFile){
        cerr << "Error while opening file for writing." << endl;
        return;
    }

    Node* current = head;
    while (current){
        outFile << current->key << " ";
        current = current->next;
    }

    outFile.close();
    cout << "List is writing to file.txt successfully" << endl;

}

// 6. Restore list from file
void restoreListFromFile(Node*& head){
    ifstream inFile("file.txt");

    if (!inFile){
        cerr << "Error while opening file for reading." << endl;
        return;
    }

    int key;
    while (inFile >> key){
        Node* newNode = new Node {key, NULL};
        if (!head){
            head = newNode;
        }
        else {
            Node* current = head;
            while (current->next){
                current = current->next;
            }
            current->next = newNode;
        }
    }
    inFile.close();
    cout << "List is restored from file.txt " << endl;

}

// 7. Killing list
void killList(Node*& head){
    while (head){
        Node* temp = head;
        head = head->next;
        delete temp;
    }
    cout << "List is deleted." << endl;
}

```

```
160
161 int main (){
162     Node* list = createList();
163     int choice, key, targetKey;
164
165     do{
166         cout << "1. Add element before element with key. " << endl;
167         cout << "2. Killing element with specified key. " << endl;
168         cout << "3. Print list. " << endl;
169         cout << "4. Writing to file. " << endl;
170         cout << "5. Restore list from file" << endl;
171         cout << "6. Killing list" << endl;
172         cout << "7. Exit" << endl;
173         cout << "Please, choose an option. " << endl;
174         cin >> choice;
175
176         switch(choice){
177             case 1:
178                 cout << "Enter key for search: ";
179                 cin >> targetKey;
180                 cout << "Enter new key: ";
181                 cin >> key;
182                 addBeforeTheKey(list, targetKey, key);
183                 break;
184
185             case 2:
186                 cout << "Enter key for deleting: ";
187                 cin >> key;
188                 killElementWithKey(list, key);
189                 break;
190
191             case 3:
192                 printList(list);
193                 break;
194
195             case 4:
196                 writingToFile(list);
197                 break;
198
199             case 5:
200                 killList(list);
201                 restoreListFromFile(list);
202                 break;
203
204             case 6:
205                 killList(list);
206                 break;
207
208             case 7:
209                 cout << "Exit. Have a nice day:)";
210                 break;
211
212             default:
213                 cout << "You've entered wrong number. Please try again." << endl;
214         }
215
216     } while (choice != 7);
217
218
219     return 0;
220 }
```

Рисунок 3.1. Код до програми № 1

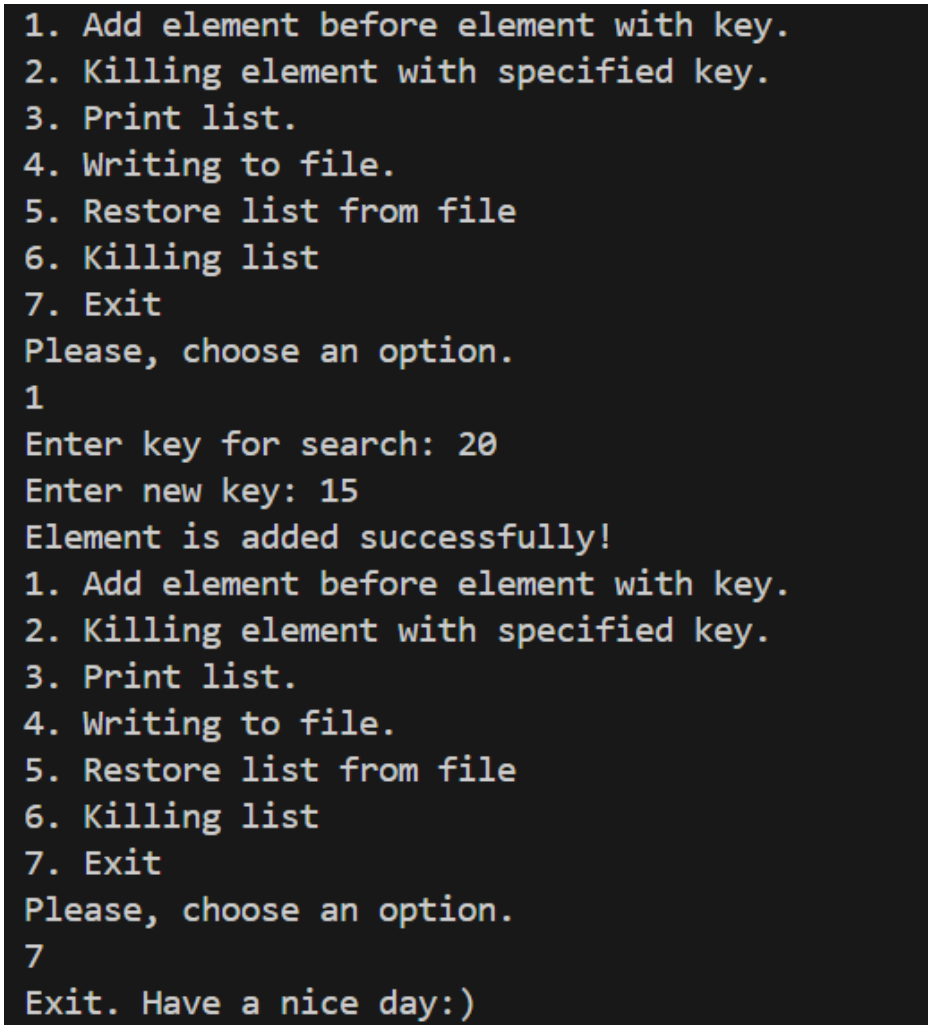


Рисунок 3.2. Приклад виконання програми № 1

Фактично затрачений час 3 години.

Посилання на файл у пулл реквесті

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/468

Завдання №2

```cpp
#include <iostream>
#include <vector>
#include <queue> // for queue
#include <algorithm> // for max

using namespace std;

// left, right, up, down
const int dx[] = {-1, 1, 0, 0};
const int dy[] = {0, 0, -1, 1};

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

    x--;
    y--;

    // create map
    vector<vector<int>> Mountain(N, vector<int>(M, -1));

    // BFS
    queue<pair<int, int>> queue;
    queue.push({x, y});
    Mountain[x][y] = 0;

    while (!queue.empty()) {
        auto [cx, cy] = queue.front();
        queue.pop();

        for (int dir = 0; dir < 4; dir++) {
            int nx = cx + dx[dir];
            int ny = cy + dy[dir];

            if (nx >= 0 && nx < N && ny >= 0 && ny < M && Mountain[nx][ny] == -1) {
                Mountain[nx][ny] = Mountain[cx][cy] + 1;
                queue.push({nx, ny});
            }
        }
    }

    int height = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            height = max(height, Mountain[i][j]);
        }
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            Mountain[i][j] = height - Mountain[i][j];
        }
    }

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cout << Mountain[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Рисунок 3.3. Код до програми № 2

```
3 4
2 2
1 2 1 0
2 3 2 1
1 2 1 0
```

Рисунок 3.4. Приклад виконання програми № 2

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a minute ago	C++ 23	Accepted	0.117	6.879	View

Рисунок 3.5. Статус задачі на алготестері

Фактично затрачений час 3 години.

Посилання на файл у пулл реквесті

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/468

Завдання №3

```cpp
#include <iostream>

using namespace std;

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

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

template< typename T >
class LinkedList {

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

    void insert(int index, int listSize, T values[]);
    void erase(int index, int count);
    T get(int index);
    void set(int index, T value);
    int getSize();
    void print();

    friend std::ostream& operator<<(std::ostream& os, const LinkedList& list) {
        Node<T>* current = list.head;

        while(current) {
            os << current->data << " ";
            current = current->next;
        }
        os << std::endl;
        return os;
    }

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

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

    // Якщо вставка на початок списку
    if (index == 0) {
        for (int j = listSize - 1; j >= 0; j--) {  // Вставляємо елементи у зворотньому порядку
            Node<T>* newNode = new Node<T>(values[j]);
            newNode->next = head;
            head = newNode;
            size++;
        }
    }
    else {
        Node<T>* current = head;

        // Знайдемо позицію перед вставкою
        for (int i = 0; i < index - 1; i++) {
            current = current->next;
        }

        // Тепер вставляємо нові елементи
        for (int j = 0; j < listSize; j++) {
            Node<T>* newNode = new Node<T>(values[j]);
            newNode->next = current->next;
            current->next = newNode;
            current = newNode;
            size++;
        }
    }
}
```

```cpp

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

    // Якщо видаляємо елементи з початку
    if (index == 0) {
        Node<T>* current = head;
        for (int i = 0; i < count; i++) {
            if (current == nullptr) break;
            Node<T>* nextNode = current->next;
            delete current;
            current = nextNode;
            size--;
        }
        head = current; // Оновлюємо head на новий початок
    }
    else {
        Node<T>* current = head;

        // Знайдемо позицію перед елементом, з якого потрібно почати видалення
        for (int i = 0; i < index - 1; i++) {
            current = current->next;
        }

        // Видалимо елементи
        for (int j = 0; j < count; j++) {
            if (current->next == nullptr) break; // Перевіряємо, чи є наступний елемент
            Node<T>* nodeToDelete = current->next;
            current->next = current->next->next; // Зв'язуємо поточний вузол з наступним після видаленого
            delete nodeToDelete;
            size--;
        }
    }
}

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

    Node<T>* current = head;

    // find node at index
    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;

    // find node at index
    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) {
        std::cout << current->data << " ";
        current = current->next;
    }
    std::cout << std::endl;
}

```

```cpp
int main() {

    LinkedList<int> list;
    int Q;
    cin >> Q;

    for (int i = 0; i < Q; i++) {
        string choice;
        cin >> choice;

        if (choice == "insert") {
            int index, N;
            cin >> index >> N;

            int* values = new int[N];
            for (int k = 0; k < N; k++) {
                cin >> values[k];
            }
            list.insert(index, N, values);
            delete[] values;
        }
        else if (choice == "erase") {
            int index, n;
            cin >> index >> n;
            list.erase(index, n);
        }
        else if (choice == "size") {
            cout << list.getSize() << endl;
        }
        else if (choice == "get") {
            int index;
            cin >> index;
            cout << list.get(index) << endl;
        }
        else if (choice == "set") {
            int index, value;
            cin >> index >> value;
            list.set(index, value);
        }
        else if (choice == "print") {
            list.print();
        }
    }

    return 0;
}
```

Рисунок 3.6. Код до програми № 3

```
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
```

Рисунок 3.7. Приклад виконання програми №3

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a few seconds ago	C++ 23	Accepted	0.008	1.297	View

Рисунок 3.8. Статус задачі на Algotester

Фактично затрачений час 6 годин.

Посилання на файл у пулл реквесті

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/468

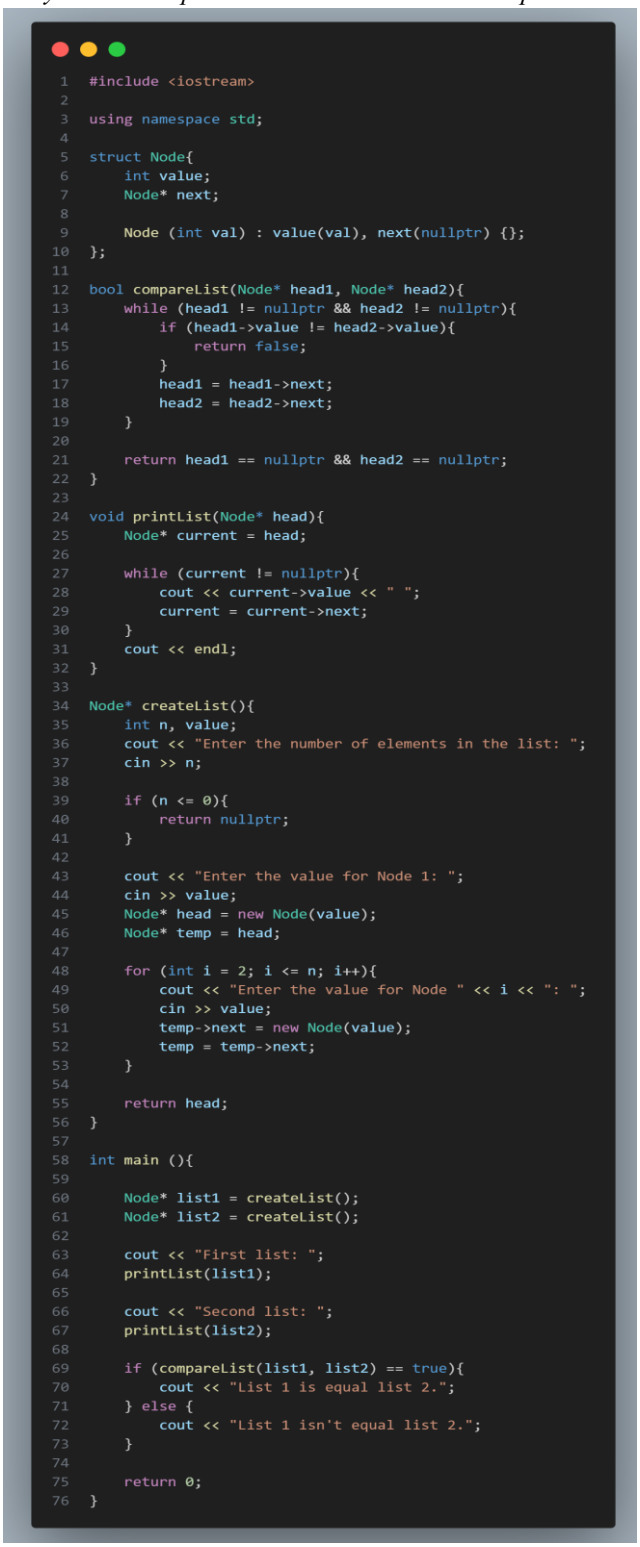
Завдання №4

```cpp
#include <iostream>

using namespace std;

struct Node{
    int value;
    Node* next;

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

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

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

// for printing list
void printList(Node* head){
    Node* current = head;

    while (current != nullptr) {
        cout << current->value << " ";
        current = current->next;
    }
    cout << endl;
}

// for creating list to allow user write elements to revers them
Node* createList(){
    int n, value;
    cout << "Enter the number of elements in the list to make reverse: ";
    cin >> n;

    if (n <= 0){
        return nullptr;
    }

    cout << "Enter the value for Node 1: ";
    cin >> value;
    Node* head = new Node(value);
    Node* temp = head;

    for (int i = 2; i <= n; i++){
        cout << "Enter the value for Node " << i << ": ";
        cin >> value;
        temp->next = new Node(value);
        temp = temp->next;
    }

    return head;
}

int main (){

    Node* head = createList();

    cout << "User's list: ";
    printList(head);

    head = reverse(head);

    cout << "Reversed list is: ";
    printList(head);

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

    return 0;
}
```

Рисунок 3.9. Код до задачі номер 1

```
Enter the number of elements in the list to make reverse: 5
Enter the value for Node 1: 2
Enter the value for Node 2: 5
Enter the value for Node 3: 4
Enter the value for Node 4: 7
Enter the value for Node 5: 9
User's list: 2 5 4 7 9
Reversed list is: 9 7 4 5 2
```

Рисунок 3.10. Приклад виконання задачі номер 1



```cpp
#include <iostream>

using namespace std;

struct Node{
    int value;
    Node* next;

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

bool compareList(Node* head1, Node* head2){
    while (head1 != nullptr && head2 != nullptr){
        if (head1->value != head2->value){
            return false;
        }
        head1 = head1->next;
        head2 = head2->next;
    }

    return head1 == nullptr && head2 == nullptr;
}

void printList(Node* head){
    Node* current = head;

    while (current != nullptr){
        cout << current->value << " ";
        current = current->next;
    }
    cout << endl;
}

Node* createList(){
    int n, value;
    cout << "Enter the number of elements in the list: ";
    cin >> n;

    if (n <= 0){
        return nullptr;
    }

    cout << "Enter the value for Node 1: ";
    cin >> value;
    Node* head = new Node(value);
    Node* temp = head;

    for (int i = 2; i <= n; i++){
        cout << "Enter the value for Node " << i << ": ";
        cin >> value;
        temp->next = new Node(value);
        temp = temp->next;
    }

    return head;
}

int main (){

    Node* list1 = createList();
    Node* list2 = createList();

    cout << "First list: ";
    printList(list1);

    cout << "Second list: ";
    printList(list2);

    if (compareList(list1, list2) == true){
        cout << "List 1 is equal list 2.";
    } else {
        cout << "List 1 isn't equal list 2.";
    }

    return 0;
}
```

Рисунок 3.11. Код до задачі номер 2

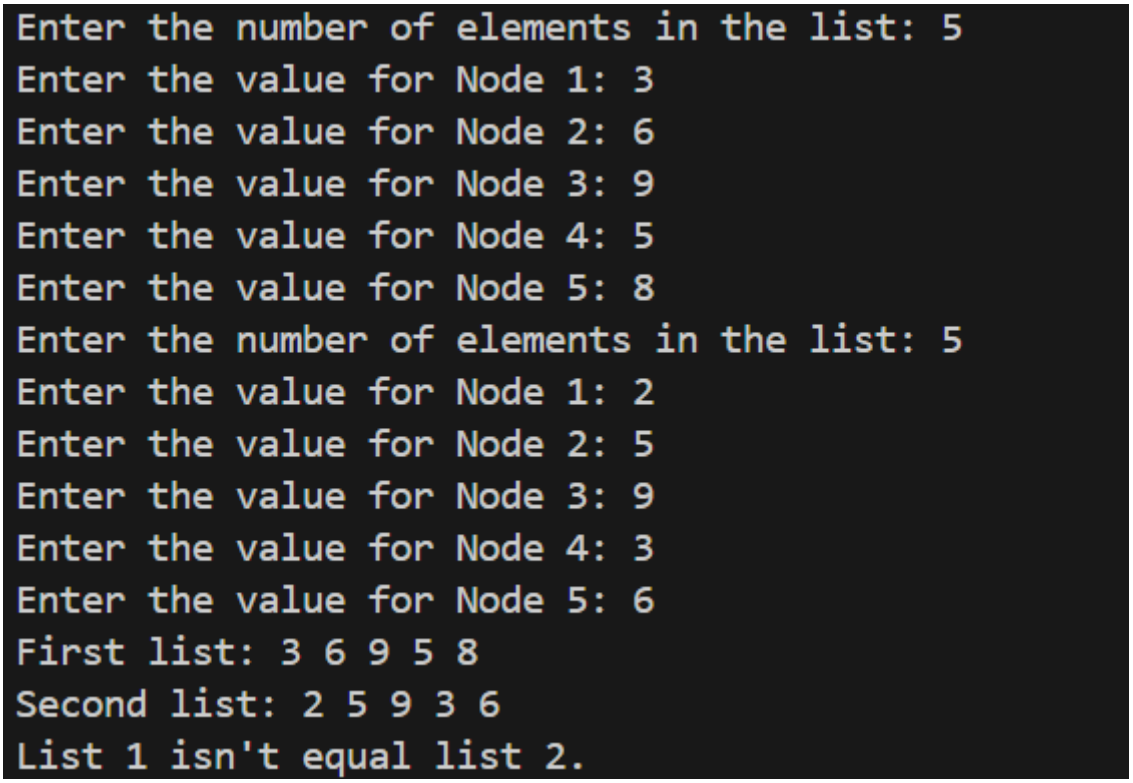


Рисунок 3.12. Приклад виконання задаіч номер 2

```cpp
#include <iostream>

using namespace std;

struct Node
{
    int value;
    Node* next;

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

void printList(Node* head){
    Node* current = head;

    while (current != nullptr){
        cout << current->value << " ";
        current = current->next;
    }
    cout << endl;
}

Node* createList(){
    int n, value;
    cout << "Enter the number of elements in the list: ";
    cin >> n;

    if (n <= 0){
        return nullptr;
    }

    cout << "Enter the value for Node 1: ";
    cin >> value;
    Node* head = new Node(value);
    Node* temp = head;

    for (int i = 2; i <= n; i++){
        cout << "Enter the value for Node " << i << ": ";
        cin >> value;
        temp->next = new Node(value);
        temp = temp->next;
    }

    return head;
}

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

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

```

```
60
61  Node* addAfterReverse(Node* n1, Node* n2) {
62      Node* result = nullptr;
63      Node* tail = nullptr;
64      int carry = 0;
65
66      while (n1 != nullptr || n2 != nullptr || carry != 0) {
67          int sum = carry;
68
69          if (n1 != nullptr) {
70              sum += n1->value;
71              n1 = n1->next;
72          }
73
74          if (n2 != nullptr) {
75              sum += n2->value;
76              n2 = n2->next;
77          }
78
79          carry = sum / 10;
80          int digit = sum % 10;
81
82          Node* newNode = new Node(digit);
83
84          if (result == nullptr) {
85              result = newNode;
86          } else {
87              tail->next = newNode;
88          }
89
90          tail = newNode;
91      }
92
93      return result;
94  }
95
96  Node* add(Node* n1, Node* n2){
97
98      n1 = reverse(n1);
99      n2 = reverse(n2);
100
101     Node* sum = addAfterReverse(n1, n2);
102
103     return reverse(sum);
104 }
105
106 int main (){
107
108     Node* list1 = createList();
109     printList(list1);
110
111     Node* list2 = createList();
112     printList(list2);
113
114     Node* result = add(list1, list2);
115
116     cout << "Sum: ";
117     printList(result);
118
119     return 0;
120
121 }
```

Рисунок 3.13. Код до задачі номер 3

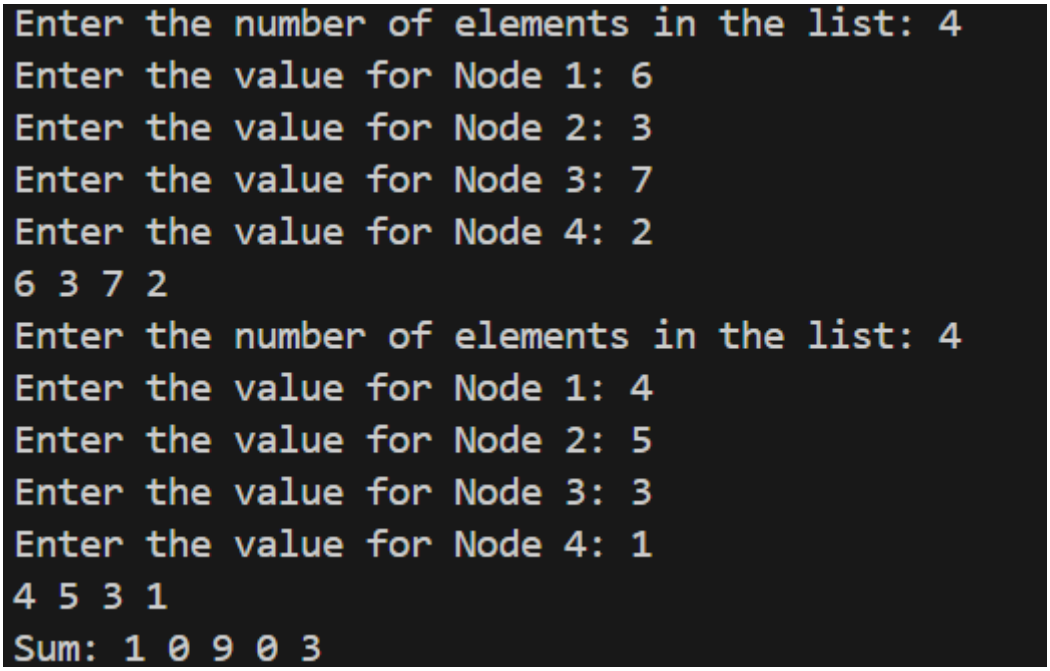


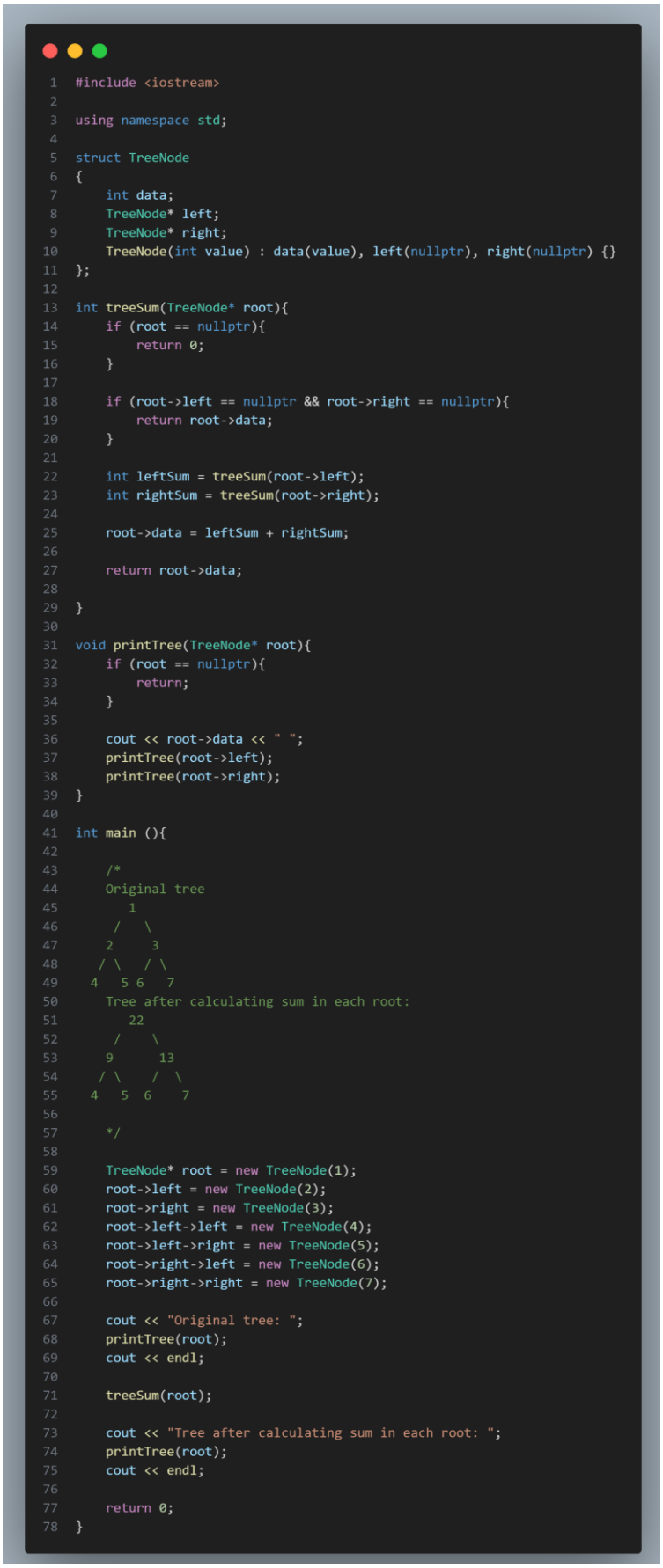
Рисунок 3.14. Приклад виконання програми номер 3

```cpp
#include <iostream>

using namespace std;

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

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

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

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

    return newRoot;
}

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

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

int main (){

    /* Original tree
        1
       /  \
      2    3
     / \  / \
     4 5  6  7
      */
    /*  Mirrored tree
        1
       /  \
      3    2
     / \  / \
    7  6  5  4

    */

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

    TreeNode* mirroredTree = create_mirror_flip(root);

    cout << "Mirrored tree: ";
    printTree(mirroredTree);

    return 0;
}
```

Рисунок 3.15. Код до задачі номер 4

```
Original tree: 1 2 4 5 3 6 7
Mirrored tree: 1 3 7 6 2 5 4
```

Рисунок 3.16. Приклад виконання задачі номер 4



```cpp
#include <iostream>

using namespace std;

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

int treeSum(TreeNode* root){
    if (root == nullptr){
        return 0;
    }

    if (root->left == nullptr && root->right == nullptr){
        return root->data;
    }

    int leftSum = treeSum(root->left);
    int rightSum = treeSum(root->right);

    root->data = leftSum + rightSum;

    return root->data;

}

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

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

int main (){

    /*
    Original tree
       1
     /   \
    2     3
   / \   / \
  4   5 6   7
    Tree after calculating sum in each root:
       22
     /    \
    9      13
   / \    /  \
  4   5  6    7

    */

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

    treeSum(root);

    cout << "Tree after calculating sum in each root: ";
    printTree(root);
    cout << endl;

    return 0;
}
```

Рисунок 3.17. Код до задачі номер 5

```
Original tree: 1 2 4 5 3 6 7
Tree after calculating sum in each root: 22 9 4 5 13 6 7
```

Рисунок 3.18. Приклад виконання задачі номер 5

Фактично затрачений час 5 годин.

Посилання на файл у пулл реквесті
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/468
Завдання №5

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

using namespace std;

int main(){

    int N, M;
    cin >> N >> M;

    vector<vector<char>> cave(N, vector<char>(M));

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

    for (int j = 0; j < M; j++){
        for (int i = N - 2; i >= 0; i--){
            if (cave[i][j] == 'S'){
                int count = i;
                while(count + 1 < N && cave[count + 1][j] == 'O'){
                    count++;
                }
                if (count != i && cave[count][j] != 'X'){
                    cave[count][j] = 'S';
                    cave[i][j] = 'O';
                }
            }
        }
    }

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

    return 0;
}
```

Рисунок 3.19. Код до програми №5

```
5 5
SSOSS
OOOOO
SOOXX
OOOOS
OOSOO
```

input

```
OOOOO
OOOSS
OOOXX
SOOOO
SSSOS
```

output

Рисунок 3.20. Приклад виконання прграми номер 5

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a few seconds ago	C++ 23	Accepted	0.068	2.328	View

Рисунок 3.21. Статус задачі на Algotester

Фактично затрачений час 2 години.

Посилання на файл у пулл реквесті

https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/468

- **4. Робота з командою:**

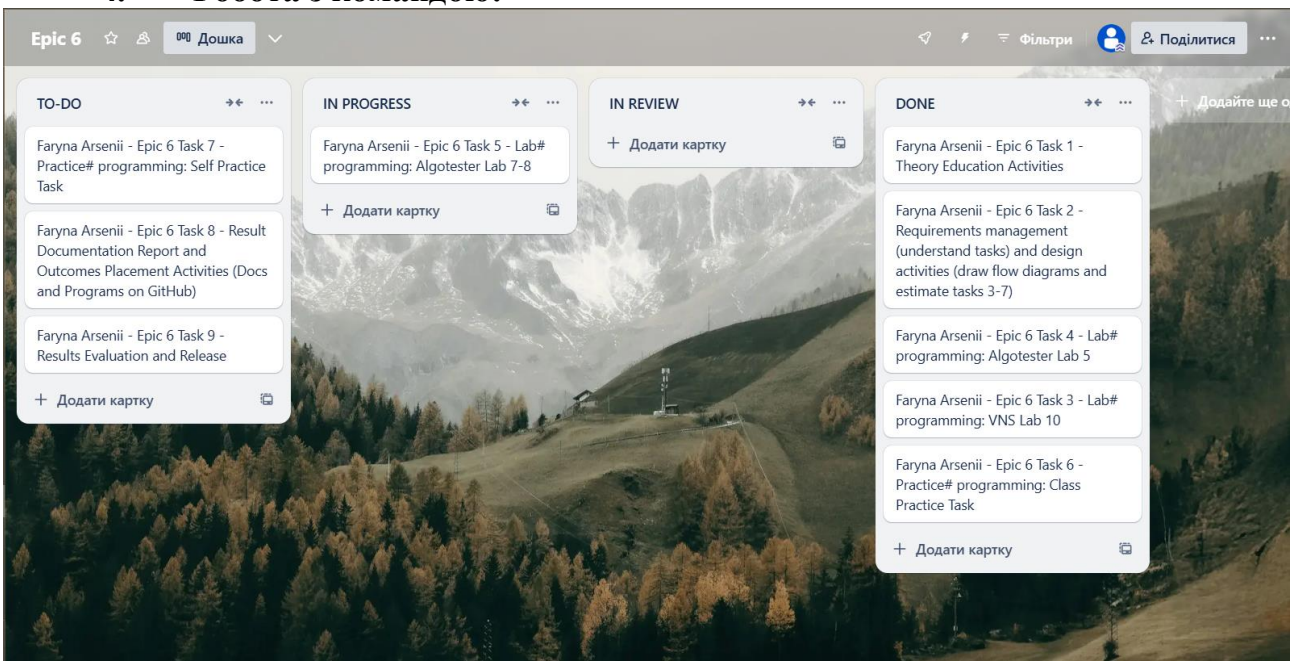


Рисунок 4.1. Командна дошка в Trello

Висновок: У межах практичних та лабораторних робіт блоку №6, я вивчив багато нового матеріалу, такого як: динамічні структури (черга, стек, списки, дерево), алгоритми обробки динамічних структур. На практиці попрацював з бінарними деревами, створив зв’язаний список, а також застосував BFS для розв’язку однієї задачі. Створив дошку в Trello для комфортної роботи з командою.