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
Федоришин Микола Володимирович

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 14)

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 *char (рядок символів).
Сформувати двонаправлений список. Знищити з нього К елементів із зазначеними номерами. Додати К елементів із зазначеними номерами.

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

Коли Коля та Вася прийшли робити ремонт на «Екстралогіку» — першим, що вони побачили в офісі, був стіл для настільного тенісу. Поки всі інші працювали, Коля та Вася вирішили пограти. Через декілька годин прийшов директор і накричав на заробітчан через те, що вони нічим не займаються. Тож Вася і Коля мусили йти працювати.

По дорозі вони сперечалися, хто ж виграв і з яким рахунком. Оскільки вони записували результати кожної подачі, то це можна порахувати. Але оскільки гра тривала дуже довго — порахувати це вручну дуже тяжко.

Всього відбулося nn подач. Про кожну з них ми знаємо, хто переміг. За виграну подачу гравець отримує одне очко. Партія вважається виграною, коли один з гравців набере не менше одинадцяти очок з перевагою щонайменше у два очки. Наприклад, за рахунків 11:9, 4:11, 15:13 партія закінчується, а за рахунків 11:10 та 99:98 — ні. Як тільки Коля і Вася закінчили одну партію — вони починають іншу.
Знаючи, хто переміг кожної подачі — виведіть загальний рахунок по партіях в грі Коля-Вася. А якщо вони не дограли останню партію, то і її рахунок теж.

Вхідні дані

У першому рядку задано ціле число n — загальна кількість подач.
У другому рядку задано n символів ci. ci=K, якщо ii-ту подачу виграв Коля, та ci=V, якщо i-ту подачу виграв Вася.

Вихідні дані

У першому рядку виведіть загальний рахунок гри по партіях у форматі k:v, де k — кількість партій, у яких переміг Коля, а v — кількість партій, у яких переміг Вася.
Якщо вони не дограли останню партію, то в другому рядку в такому ж форматі виведіть рахунок останньої партії.

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 бібліотеку fstream для запису даних у файл.
- Плановий час на реалізацію 2.5 години.

Програма №2

- Блок – схема

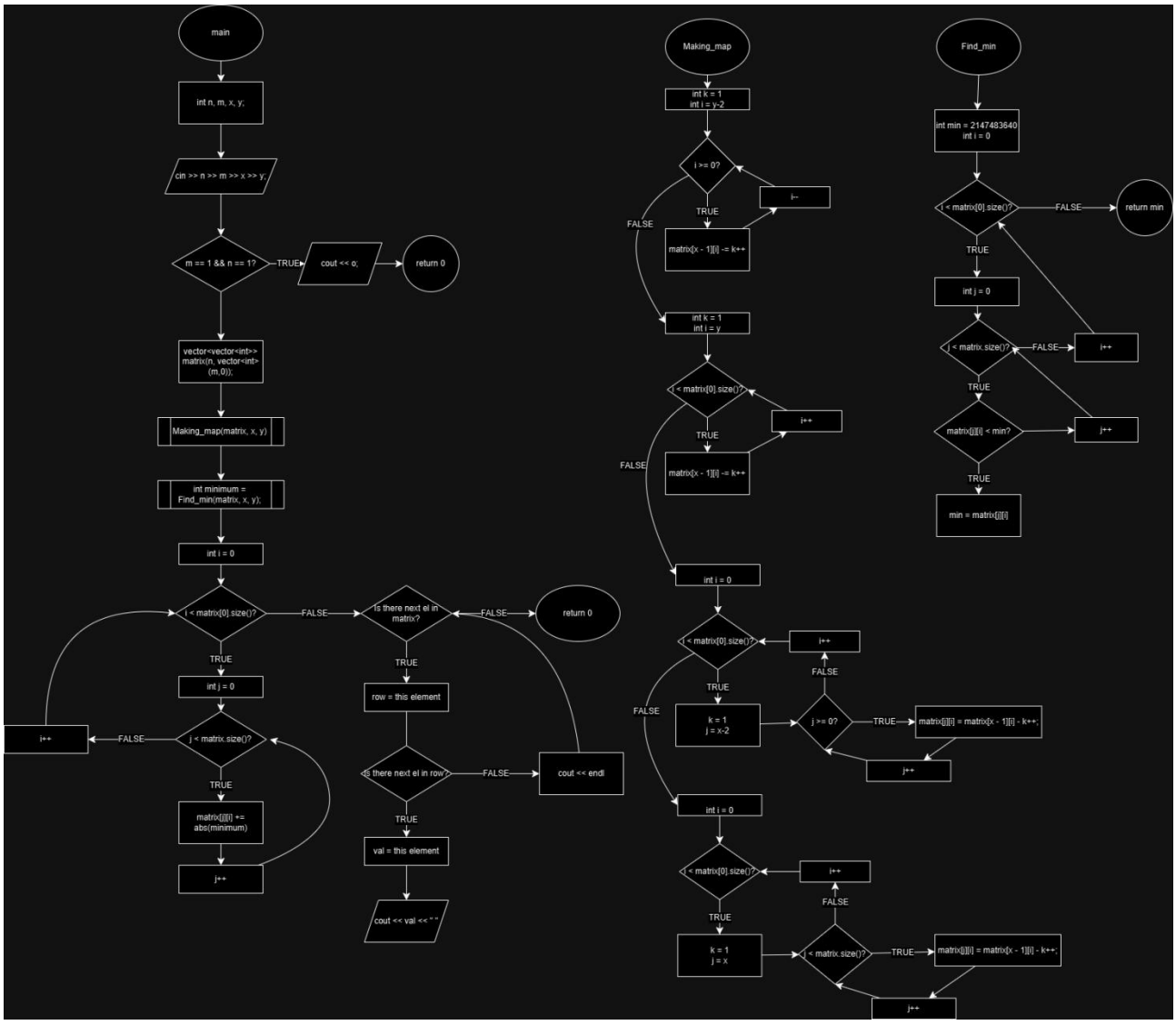

Рисунок 2.1. Блок – схема до програми 1

- Плановий час на реалізацію 2 години.

Програма №3

- Важливі деталі для реалізації програми.
- Зрозуміти, що таке List 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

- Плановий час на реалізацію 1 година.

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

Завдання №1

vns_lab_10_task_1_variant_14_mykola_fedoryshyn.cpp > main()

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

// Структура для вузла двонаправленого списку
struct Node {
    std::string data;
    Node* prev;
    Node* next;

    Node(const std::string& value) : data(value), prev(nullptr), next(nullptr) {}
};

// Клас для роботи з двонаправленим списком
class List {
private:
    Node* head;
    Node* tail;

public:
    List() : head(nullptr), tail(nullptr) {}

    // Деструктор для знищення списку
    ~List() {
        clear();
    }
    // Допоміжні функції
    int Size(){
        int i = 0;
        Node* temp = head;
        while(temp != nullptr){
            i++;
            temp = temp->next;
        }
        return i;
    }
    Node* getNode(int index){
        Node* current = head;
        for (int i = 0; i < index; ++i) {
            current = current->next;
        }
        return current;
    }
    //////////////////////


    void createList() {
        head = nullptr;
        tail = nullptr;
    }

    void addElement(const std::string& value) {
        Node* newNode = new Node(value);
        if (!head) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void insertEl(const std::string& value, int& index){//////////////////////////
        if (index < 0 || index > Size()) return;

        Node* newNode = new Node(value);
        if(index==0){
            newNode->next = head;
            if (head) head->prev = newNode;
            head = newNode;
            if (!tail) tail = newNode;
```

```cpp
    void insertEl(const std::string& value, int& index){//////////////
        if (index < 0 || index > Size()) return;

        Node* newNode = new Node(value);
        if(index==0){
            newNode->next = head;
            if (head) head->prev = newNode;
            head = newNode;
            if (!tail) tail = newNode;
        }else if(index == Size()){
            newNode->prev = tail;
            if (tail) tail->next = newNode;
            tail = newNode;
        }else{
            Node* current = getNode(index);
            newNode->next = current;  // 1 2 3 4 5 | 1 2 7->3 4 5 (7 
            newNode->prev = current->prev; // 1 2 3 4 5 | 1 2<-7->3 4
            current->prev->next = newNode;
            current->prev = newNode;
            }
            ++index;
    }

    // 3. Видалення елемента за номером (індексом)
    void deleteElement(int position) {
        if (!head || position < 0) return;

        Node* temp = getNode(position);

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

            if (temp == head) head = temp->next;
            if (temp == tail) tail = temp->prev;

            delete temp;
        }
    }

    // 4. Друк списку
    void printList() const {
        Node* temp = head;
        if(!temp){
            std::cout << "Список порожній\n";//////////////////////
        }
        while (temp) {
            std::cout << temp->data << " ";
            temp = temp->next;
        }
        std::cout << std::endl;
    }

    // Запис списку у файл
    void saveToFile(const std::string& filename) const {
        std::ofstream file(filename);
        if (!file.is_open()) {
            std::cerr << "Не вдалося відкрити файл для запису.\n";
            return;
        }

        Node* temp = head;
        while (temp) {
            file << temp->data << "\n";
            temp = temp->next;
```

```
123
124         Node* temp = head;
125         while (temp) {
126             file << temp->data << "\n";
127             temp = temp->next;
128         }
129         file.close();
130     }
131
132     void clear() {
133         Node* temp = head;
134         while (temp) {
135             Node* nextNode = temp->next;
136             delete temp;
137             temp = nextNode;
138         }
139         head = tail = nullptr;
140     }
141
142     // Відновлення списку з файлу
143     void loadFromFile(const std::string& filename) {
144         clear();
145         std::ifstream file(filename);
146         if (!file.is_open()) {
147             std::cerr << "Не вдалося відкрити файл для читання.\n";
148             return;
149         }
150
151         std::string line;
152         while (std::getline(file, line)) {
153             addElement(line);
154         }
155         file.close();
156     }
157
158 };
159
160 int main() {
161     int k; //кількість ел
162     List list;
163     int app_index[k], del_index, current_app_index;
164
165     list.createList();
166     list.addElement("First");
167     list.addElement("Second");
168     list.addElement("Third");
169     std::cout << "Список з початку:\n";
170     list.printList();
171
172
173     std::cout << "Введіть кількість елементів які потрібно додати і їх індекси[0 - " << list.Size() <<"] ";
174     std::cin >> k;
175     for(int i = 0; i<k; i++){
176         std::cin >> app_index[i];
177     }
178     std::cin.ignore();
179
180     std::string add_el;
181     for(int i = 0; i<k; i++){
182         current_app_index = app_index[i]+i;
183         std::cout << "Введіть значення(рядок) елементу за індексом " << current_app_index << " : ";
184         std::getline(std::cin, add_el);
185         list.insertEl(add_el, current_app_index);
186     }
187
```

```
    std::string add_el;
    for(int i = 0; i<k; i++){
        current_app_index = app_index[i]+i;
        std::cout << "Введіть значення(рядок) елементу за індексом " << current_app_index << " : ";
        std::getline(std::cin, add_el);
        list.insertEl(add_el, current_app_index);
    }



    std::cout << "Список після додавання елементів:\n";
    list.printList();

    std::cout << "Введіть кількість елементів які потрібно видалити(макс. " << list.Size() << " ), а потім їх
    std::cin >> k;

    for(int i = 0; i<k; i++){
        std::cin >> del_index;
        list.deleteElement(del_index-i);
    }

    std::cout << "Список після видалення елементів:\n";
    list.printList();

    std::cout << "\nСписок завантажуємо у файл\n\n";
    list.saveToFile("list.txt");
    list.clear();

    std::cout << "Список після очищення:\n";
    list.printList();

    list.loadFromFile("list.txt");
    std::cout << "Список після завантаження з файлу:\n";
    list.printList();

    return 0;
}
```

Рисунок 3.1. Код до програми № 1

```
Список з початку:
First Second Third
Введіть кількість елементів які потрібно додати і їх індекси[0 - 3] 2 0 3
Введіть значення(рядок) елементу за індексом 0 : null
Введіть значення(рядок) елементу за індексом 4 : four
Список після додавання елементів:
null First Second Third four
Введіть кількість елементів які потрібно видалити(макс. 5 ), а потім їх індекс: 2
0 1
Список після видалення елементів:
Second Third four

Список завантажуємо у файл

Список після очищення:
Список порожній

Список після завантаження з файлу:
Second Third four
PS C:\GitHub\ai_programming_playground_2024\ai_11\mykola_fedoryshyn\epic_6>
```

Рисунок 3.2. Приклад виконання програми № 1

Фактично затрачений час 3 години.

Посилання на файл у пулл реквесті

Завдання №2

algotester_lab_5_variant_1_mykola_fedoryshyn.cpp > Making_map(vector<vector<i

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

void Making_map(vector<vector<int>>& matrix, int x, int y);
int Find_min(vector<vector<int>>& matrix, int x, int y);

int main() {
    int n, m, x, y;
    cin >> n >> m >> x >> y;

    if (m == 1 && n == 1) {
        cout << 0;
        return 0;
    }

    vector<vector<int>> matrix(n, vector<int>(m,0));

    Making_map(matrix, x, y);
    int minimum = Find_min(matrix, x, y);


    for (int i = 0; i < matrix[0].size(); i++){
        for (int j = 0; j < matrix.size(); j++){
            matrix[j][i] += abs(minimum);
        }
    }
    for (const auto& row : matrix) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
    }
    return 0;
}
```

```cpp

void Making_map(vector<vector<int>>& matrix, int x, int y) {
    int k = 1;
    for (int i = y-2; i >= 0; i--) {
        matrix[x - 1][i] -= k++;
    }
    k = 1;
    for (int i = y; i < matrix[0].size(); i++) {
        matrix[x - 1][i] -= k++;
    }

    for (int i = 0; i < matrix[0].size(); i++) {
        k = 1;
        for (int j = x-2; j >= 0; j--){
            matrix[j][i] = matrix[x - 1][i] - k++;
        }
    }
    for (int i = 0; i < matrix[0].size(); i++) {
        k = 1;
        for (int j = x; j < matrix.size(); j++){
            matrix[j][i] = matrix[x - 1][i] - k++;
        }
    }
}

int Find_min(vector<vector<int>>& matrix, int x, int y){
    int min = 2147483640;
    for (int i = 0; i < matrix[0].size(); i++){
        for (int j = 0; j < matrix.size(); j++){
            if (matrix[j][i] < min) min = matrix[j][i];
        }
    }
    return min;
}

```

Рисунок 3.3. Код до програми № 2

```
3 4
2 2
1 2 1 0
2 3 2 1
1 2 1 0
PS C:\GitHub\ai
```

Рисунок 3.4. Приклад виконання програми № 2

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
4 дні тому	C++ 20	Зараховано	0.089	7.746	Перегляд

Рисунок 3.5. Статус задачі на алготестері

Фактично затрачений час 2 години.

Посилання на файл у пулл реквесті

Завдання №3

algotester_lab_7_8_variant_1_mykola_fedoryshyn.cpp > DoublyLinkedList<T> > insert(int, int, T *)

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


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

    Node(const T& value) : data(value), prev(nullptr), next(nullptr) {}
};

// Двозв'язний список
template<typename T>
class DoublyLinkedList {
private:
    Node<T>* head;
    Node<T>* tail;
    int size_;

    // Отримання вузла за індексом
    Node<T>* getNode(int index){
        Node<T>* current = head;
        for (int i = 0; i < index; ++i) {
            current = current->next;
        }
        return current;
    }

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

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


    void insert(int index, int N, T* elements) {
        if (index < 0 || index > size_) return;

        for (int i = 0; i < N; ++i) {
            Node<T>* newNode = new Node<T>(elements[i]);

            if (index == 0) {
                newNode->next = head;
                if (head) head->prev = newNode;
                head = newNode;
                if (!tail) tail = newNode;
```

```
53                  if (!tail) tail = newNode;
54
55              } else if (index == size_) {
56                  newNode->prev = tail;
57                  if (tail) tail->next = newNode;
58                  tail = newNode;
59
60              } else {
61                  Node<T>* current = getNode(index);
62                  newNode->next = current;  // 1 2 3 4 5 | 1 2 7->3 4 5 (7 connec
63                  newNode->prev = current->prev; // 1 2 3 4 5 | 1 2<-7->3 4 5
64                  current->prev->next = newNode;
65                  current->prev = newNode;
66              }
67              ++index;
68              ++size_;
69          }
70      }
71
72      // Видалення n елементів з позиції index
73      void erase(int index, int n) {
74          if (index < 0 || index >= size_ || n <= 0) return;
75
76          for (int i = 0; i < n && index < size_; ++i) {
77              Node<T>* toDelete = getNode(index);
78
79              if (toDelete->prev) toDelete->prev->next = toDelete->next; // 1->2-
80              if (toDelete->next) toDelete->next->prev = toDelete->prev; // 1<-2<
81              if (toDelete == head) head = toDelete->next;
82              if (toDelete == tail) tail = toDelete->prev;
83
84              delete toDelete;
85              --size_;
86          }
87      }
88
89      int size(){
90          return size_;
91      }
92
93      T get(int index){
94          return getNode(index)->data;
95      }
96
97      void set(int index, const T& value) {
98          getNode(index)->data = value;
99      }
100
101     friend ostream& operator<<(ostream& os, const DoublyLinkedList<T>& list) {
102         Node<T>* current = list.head;
```

Рисунок

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

    int main() {
        int Q;
        cin >> Q;
        DoublyLinkedList<int> dll;

        while (Q--) {
            string command;
            cin >> command;

            if (command == "insert") {
                int index, N;
                cin >> index >> N;
                int* elements = new int[N];
                for (int i = 0; i < N; ++i) {
                    cin >> elements[i];
                }
                dll.insert(index, N, elements);
                delete[] elements;

            } else if (command == "erase") {
                int index, n;
                cin >> index >> n;
                dll.erase(index, n);

            } else if (command == "size") {
                cout << dll.size() << endl;

            } else if (command == "get") {
                int index;
                cin >> index;
                cout << dll.get(index) << endl;

            } else if (command == "set") {
                int index, value;
                cin >> index >> value;
                dll.set(index, value);

            } else if (command == "print") {
                cout << dll << endl;
            }
        }

        return 0;
    }
```

3.6. Код до програми № 3

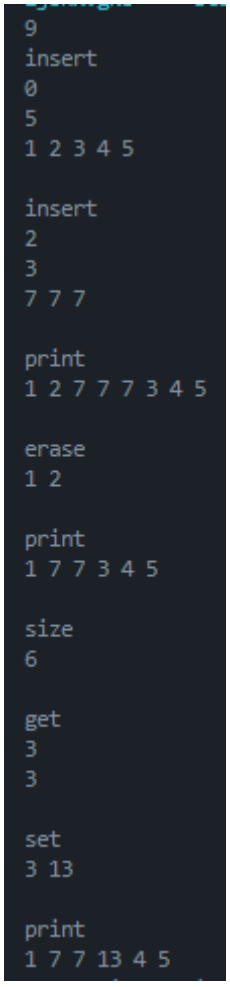

Рисунок 3.7. Приклад виконання програми №3

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
16 годин тому	C++ 20	Зараховано	0.008	1.387	Перегляд

Рисунок 3.8. Статус задачі на Algotester

Фактично затрачений час 6 годин.

Посилання на файл у пулл реквесті

Завдання №4

practice_work_task_1_mykola_fedoryshyn.cpp > reverse(Node *)

```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;
```

```
46
47      cout << "Enter the value for Node 1: ";
48      cin >> value;
49      Node* head = new Node(value);
50      Node* temp = head;
51
52      for (int i = 2; i <= n; i++){
53          cout << "Enter the value for Node " << i << ": ";
54          cin >> value;
55          temp->next = new Node(value);
56          temp = temp->next;
57      }
58
59      return head;
60  }
61
62  int main (){
63
64      Node* head = createList();
65
66      cout << "User's list: ";
67      printList(head);
68
69      head = reverse(head);
70
71      cout << "Reversed list is: ";
72      printList(head);
73
74      while (head != nullptr){
75          Node* temp = head;
76          head = head->next;
77          delete temp;
78      }
79
80      return 0;
81  }
```

Рисунок 3.9. Код до задачі номер 1

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

Рисунок 3.10. Приклад виконання задачі номер 1

practice_work_task_2_mykola_fedoryshyn.cpp > ...

```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;
}
```

```
    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

```
Enter the number of elements in the list: 3
Enter the value for Node 1: 1
Enter the value for Node 2: 2
Enter the value for Node 3: 3
Enter the number of elements in the list: 3
Enter the value for Node 1: 1
Enter the value for Node 2: 2
Enter the value for Node 3: 3
First list: 1 2 3
Second list: 1 2 3
List 1 is equal list 2.
PS C:\GitHub\ai_programming_playground_2024\ai_
```

Рисунок 3.12. Приклад виконання задаіч номер 2

practice_work_task_3_mykola_fedoryshyn.cpp > ...

```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;
```

```
55          prev = current;
56          current = next;
57      }
58      return prev;
59  }
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
```

```
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

```
Enter the number of elements in the list: 4
Enter the value for Node 1: 5
Enter the value for Node 2: 4
Enter the value for Node 3: 6
Enter the value for Node 4: 2
5 4 6 2
Enter the number of elements in the list: 4
Enter the value for Node 1: 2
Enter the value for Node 2: 4
Enter the value for Node 3: 1
Enter the value for Node 4: 6
2 4 1 6
Sum: 7 8 7 8
```

Рисунок 3.14. Приклад виконання програми номер 3

practice_work_task_4_mykola_fedoryshyn.cpp > ...

```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
```

```
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
PS C:\GitHub\ai_programming_pla
```

Рисунок 3.16. Приклад виконання задачі номер 4

practice_work_task_5_mykola_fedoryshyn.cpp > main()

```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
```

```
    /*
    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

Завдання №5

self_practice_work_algotester_task_1_mykola_fedoryshyn.cpp.cpp > ...

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

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

    int kolya_wins = 0, vasya_wins = 0;
    int kolya_points = 0, vasya_points = 0;

    for (char c : results) {
        if (c == 'K') kolya_points++;
        else if (c == 'V') vasya_points++;

        // Перевірка, чи закінчилася поточна партія
        if ((kolya_points >= 11 || vasya_points >= 11) &&
            abs(kolya_points - vasya_points) >= 2) {
            if (kolya_points > vasya_points) kolya_wins++;
            else vasya_wins++;

            kolya_points = 0;
            vasya_points = 0;
        }
    }

    cout << kolya_wins << ":" << vasya_wins << endl;

    if (kolya_points > 0 || vasya_points > 0) {
        cout << kolya_points << ":" << vasya_points << endl;
    }

    return 0;
}
```

Рисунок 3.19. Код до програми №5

```
30
VVKVKKVVVVKVKKKKKVVKKVKKKKVVVV
1:0
2:4
PS C:\GitHub\ai_programming_playgro
```

Рисунок 3.20. Приклад виконання прграми номер 5

декілька секунд тому	C++ 20	Зараховано	0.005	1.113	Перегляд

Рисунок 3.21. Статус задачі на Algotester

Фактично затрачений час 2 години.

4. Робота з командою:

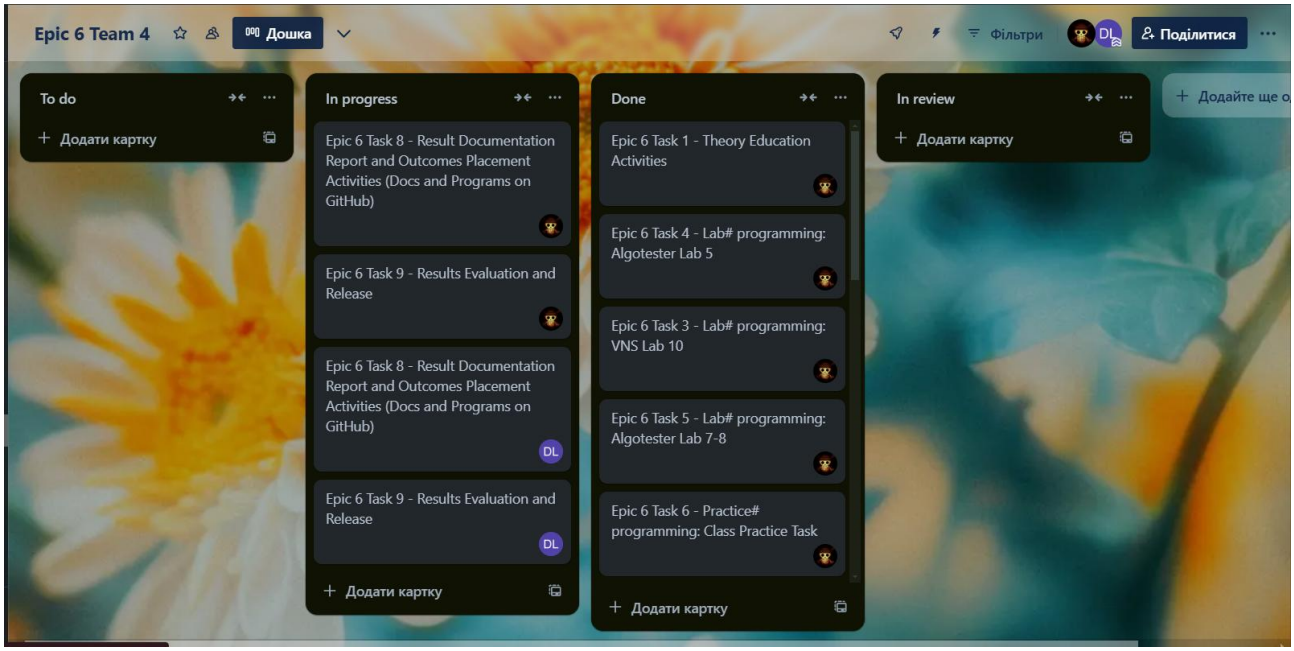

Рисунок 4.1. Командна дошка в Trello

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. Створив дошку в Trello для комфортної роботи з командою.