Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

A blue and white logo

Description automatically generated

**Звіт**

**Звіт**

**про виконання лабораторних та практичних робіт блоку № 6**

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

***з дисципліни:*** «Мови та парадигми програмування»

до:

ВНС Лабораторної Роботи № 10

Алготестер Лабораторної Роботи № 5

Алготестер Лабораторної Роботи № 7-8

Практичних Робіт № 6

**Виконала:**

Студентка групи ШІ-11

Труш Соломія Володимирівна

# **Тема роботи:**

Динамічні структури. Алгоритми обробки динамічних структур.

# **Мета роботи:**

Ознайомитись з динамічними структурами і алгоритмами їх обробки.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: **Динамічні структури**
* Тема №2: **Алгоритми обробки динамічних структур**

1. Індивідуальний план опрацювання теорії:

* Тема №1: **Динамічні структури**
  + Джерела Інформації
    - Відео 1. <https://www.youtube.com/watch?v=HKfj0l7ndbc&ab_channel=CodeBeauty>
    - Відео 2. <https://youtu.be/GBST5uQ_yos?si=GiUbOIUkMGCjKdhV>
    - Відео 3. <https://youtu.be/jaK4pn1jXTo?si=m18BkV7zbO4WPAjV>
    - Відео 4. <https://youtu.be/UHxtjVsOTHc?si=kR6xy7AVoWV4OEtW>
    - Стаття. <https://www.educative.io/blog/cpp-data-structures-interview-prep#linkedlists>
  + Що опрацьовано:
    - Ознайомилась з тим, що таке динамічні структури і для чого їх використовують
  + Статус: Ознайомлений
  + Початок опрацювання теми: 12.12
  + Звершення опрацювання теми: 15.12
* Тема №2: **Алгоритми обробки динамічних структур**
  + Джерела Інформації:
    - Відео 1. <https://www.youtube.com/watch?v=HKfj0l7ndbc&ab_channel=CodeBeauty>
    - Відео 2. <https://www.youtube.com/watch?v=6ysjqCUv3K4&ab_channel=CSDojo>
  + Що опрацьовано:
    - Коментар 1
    - Коментар 2
  + Статус: Ознайомлений
  + Початок опрацювання теми: 12.12
  + Звершення опрацювання теми: 15.12

# **Виконання роботи:**

## **1. Опрацювання завдання та вимог до програм та середовища:**

Завдання **VNS Lab 10 - Task 1**

* Варіант 11
* Деталі завдання

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.

11.Записи в лінійному списку містять ключове поле типу \*char(рядок

символів). Сформувати двонаправлений список. Знищити з нього елемент із

заданим ключем, додати елемент із зазначеним номером.

Завдання **Algotester Lab 5**

* Варіант 3
* Деталі завдання

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.

Клітинки 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.

* Важливі деталі для врахування в імплементації програми

1≤N,M≤10^3

1≤x≤N

1≤y≤M

Завдання **Algotester Lab 7-8**

* Варіант 1
* Деталі завдання

Ваше завдання - власноруч реалізувати структуру даних "Двозв’язний список".

Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи. Вам будуть поступати запити такого типу:

Вставка: 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 <<

* Важливі деталі для врахування в імплементації програми

Ціле число Q - кількість запитів.

У наступних рядках Q запитів у зазначеному в умові форматі.

0≤Q≤10^3

0≤li≤10^3

∥l∥≤10^3

Завдання **Class Practice Work**

* Деталі завдання

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 ⟹ 9→7→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 №54 - 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

Завдання **Self Practice Work**

* Деталі завдання

До Зеника і Марічки на Хелловін завітав їхній старий друг Андрій. Офіційна причина його візиту — дізнатися про справи друзів та просто побалакати з ними. Але Зенику і Марічці добре відомі справжні наміри Андрія — він просто хоче дістати від друзів цукерок. У Зеника і Марічки є по одному мішку цукерок. Усього в мішку Зеника є n цукерок, причому i-та із них коштує ai гривень. Аналогічно, у мішку Марічки є m цукерок, і ціна i-ї рівна bi гривень. Вони хочуть дати Андрію дві цукерки — одну з мішка Зеника, а іншу — з мішка Марічки. Будучи доволі жадібними, воно готові віддати Андрію лише найдешевші цукерки. Тобто, як і з першого мішка, так і з другого, вони виберуть по найдешевшій цукерці. Якою буде вартість отриманого Андрієм подарунка? Іншими словами, знайдіть сумарну вартість двох цукерок, які отримає Андрій. У єдиному рядку виведіть одне ціле число — вартість подарунку, який отримає Андрій.

* Важливі деталі для врахування в імплементації програми

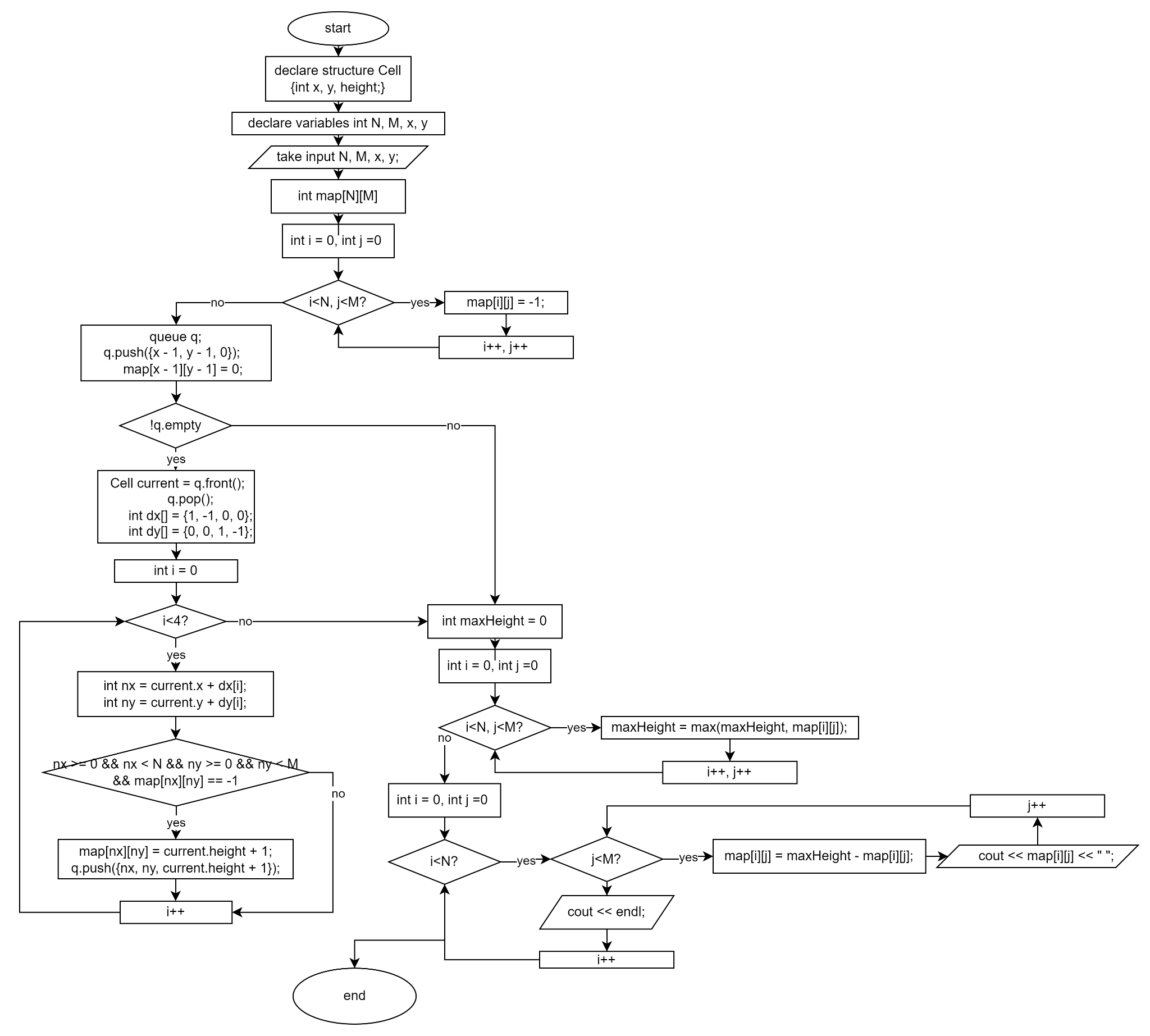
1≤n,m≤100

1≤ai,bi≤1000

## **2. Дизайн та планована оцінка часу виконання завдань:**

**Algotester Lab 5v3**

* Блок-схема



*Figure 1. Flowchart Algotester 5v3*

## **3. Конфігурація середовища до виконання завдань:**

Середовище сконфігуровано під час виконання Epic 1.

## **4. Код програм з посиланням на зовнішні ресурси:**

Завдання **VNS Lab 10 - Task 1**

Посилання на файл програми у пул-запиті GitHub

<https://github.com/artificial-intelligence-department/ai_programming_playground/blob/epic_6_practice_and_labs_solomiia_trush/ai_11/solomiia_trush/Epic%206/vns_lab_10_task_1_variant_11_solomiia_trush.cpp>

#include <iostream>

#include <fstream>

using namespace std;

struct Element\_of\_linked\_list {

char data;

Element\_of\_linked\_list\* previous;

Element\_of\_linked\_list\* next;

};

Element\_of\_linked\_list\* head;

Element\_of\_linked\_list\* tail;

void print (Element\_of\_linked\_list\* head) {

Element\_of\_linked\_list\* element = head;

if (element == NULL) {

cout << "The list is empty" << endl;

} else {

while (element != NULL) {

cout << element->data << " ";

element = element->next;

}

}

cout << endl;

}

Element\_of\_linked\_list\* create (int n) {

Element\_of\_linked\_list\* head = NULL;

for (int i=0; i<n; i++) {

Element\_of\_linked\_list\* new\_element = new Element\_of\_linked\_list();

if (i==0 && head!=NULL) {

new\_element->next = head;

head->previous = new\_element;

head = new\_element;

} else if (i==0 && head==NULL) {

head = new\_element;

} else {

Element\_of\_linked\_list\* previous\_element = head;

for (int j=0; j<i-1; j++) {

previous\_element = previous\_element->next;

}

new\_element->previous = previous\_element;

new\_element->next = previous\_element->next;

if (previous\_element->next != NULL) {

previous\_element->next->previous = new\_element;

}

previous\_element->next = new\_element;

}

}

return head;

}

void insert () {

cout << "Enter the position to add elements : ";

int index;

cin >> index;

cout << "Enter how many elements you want to add : ";

int n;

cin >> n;

cout << "Enter the elements : " ;

for (int i=0; i<n; i++) {

Element\_of\_linked\_list\* new\_element = new Element\_of\_linked\_list();

cin >> new\_element->data;

if (index==0 && head!=NULL) {

new\_element->next = head;

head->previous = new\_element;

head = new\_element;

}

if (index==0 && head==NULL) {

head = new\_element;

tail = new\_element;

}

if (index!=0) {

Element\_of\_linked\_list\* previous\_element = head;

for (int j=0; j<index-1; j++) {

previous\_element = previous\_element->next;

}

new\_element->previous = previous\_element;

new\_element->next = previous\_element->next;

if (previous\_element->next != NULL) {

previous\_element->next->previous = new\_element;

}

previous\_element->next = new\_element;

}

index++;

}

Element\_of\_linked\_list\* element = head;

while (element->next!=NULL) {

element = element->next;

}

tail = element;

}

void delete\_element\_with\_set\_value () {

cout << "Enter the element to delete : ";

char input;

cin >> input;

Element\_of\_linked\_list\* needed\_element = head;

while (needed\_element != NULL && input != needed\_element->data) {

needed\_element = needed\_element->next;

}

if (needed\_element != NULL) {

Element\_of\_linked\_list\* previous = needed\_element->previous;

if (needed\_element->previous != NULL) {

needed\_element->previous->next = needed\_element->next;

if (needed\_element->next != NULL) {

needed\_element->next->previous = needed\_element->previous;

}

} else {

head = needed\_element->next;

if (needed\_element->next != NULL) {

needed\_element->next->previous = NULL;

}

}

delete needed\_element;

}

}

void write\_in\_file (Element\_of\_linked\_list\* head) {

Element\_of\_linked\_list\* element = head;

ofstream file\_with\_list("doubly\_linked\_list\_file.txt", ios::out | ios::trunc);

if (file\_with\_list.is\_open()) {

while (element != NULL) {

file\_with\_list << element->data;

element = element->next;

}

file\_with\_list.close();

}

}

void delete\_list(Element\_of\_linked\_list\*& head) {

while (head != nullptr) {

Element\_of\_linked\_list\* temp = head;

head = head->next;

delete temp;

}

}

void restore(Element\_of\_linked\_list\*& head) {

ifstream file\_with\_list("doubly\_linked\_list\_file.txt");

if (file\_with\_list.is\_open()) {

string line;

getline(file\_with\_list, line);

for (char c : line) {

Element\_of\_linked\_list\* new\_element = new Element\_of\_linked\_list();

new\_element->data = c;

if (head == nullptr) {

head = tail = new\_element;

} else {

tail->next = new\_element;

new\_element->previous = tail;

tail = new\_element;

}

}

file\_with\_list.close();

}

}

int main () {

int n = 10;

head = create(n);

cout << "The list have been created" << endl;

print(head);

insert();

print(head);

delete\_element\_with\_set\_value();

print(head);

write\_in\_file(head);

cout << "The list have been written in file" << endl;

delete\_list(head);

cout << "The list have been destroyed" << endl;

print(head);

restore(head);

cout << "The list have been restored" << endl;

print(head);

delete\_list(head);

}

Завдання **Algotester Lab 5**

Посилання на файл програми у пул-запиті GitHub

<https://github.com/artificial-intelligence-department/ai_programming_playground/blob/epic_6_practice_and_labs_solomiia_trush/ai_11/solomiia_trush/Epic%206/algotester_lab_5_variant_3_solomiia_trush.cpp>

#include <iostream>

#include <queue>

using namespace std;

struct Cell {

int x, y, height;

};

int main() {

int N, M, x, y;

cin >> N >> M >> x >> y;

int map[N][M];

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

map[i][j] = -1; // -1 means unvisited

}

}

queue<Cell> q;

q.push({x - 1, y - 1, 0});

map[x - 1][y - 1] = 0;

while (!q.empty()) {

Cell current = q.front();

q.pop();

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

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

for (int i = 0; i < 4; i++) {

int nx = current.x + dx[i];

int ny = current.y + dy[i];

if (nx >= 0 && nx < N && ny >= 0 && ny < M && map[nx][ny] == -1) {

map[nx][ny] = current.height + 1;

q.push({nx, ny, current.height + 1});

}

}

}

int maxHeight = 0;

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

maxHeight = max(maxHeight, map[i][j]);

}

}

for (int i = 0; i < N; i++) {

for (int j = 0; j < M; j++) {

map[i][j] = maxHeight - map[i][j];

cout << map[i][j] << " ";

}

cout << endl;

}

return 0;

}

Завдання **Algotester Lab 7-8**

Посилання на файл програми у пул-запиті GitHub

<https://github.com/artificial-intelligence-department/ai_programming_playground/blob/epic_6_practice_and_labs_solomiia_trush/ai_11/solomiia_trush/Epic%206/algotester_lab_7_8_variant_1_solomiia_trush.cpp>

#include <iostream>

using namespace std;

class Doubly\_Linked\_List {

private:

struct Element\_of\_linked\_list {

int data;

Element\_of\_linked\_list\* previous;

Element\_of\_linked\_list\* next;

};

Element\_of\_linked\_list\* head;

Element\_of\_linked\_list\* tail;

public:

Doubly\_Linked\_List() {

head = nullptr;

tail = nullptr;

}

void print() const {

Element\_of\_linked\_list\* current = head;

while (current != nullptr) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

void insert() {

int index, n;

cin >> index >> n;

Element\_of\_linked\_list\* current = head;

for (int i = 0; i < index; i++) {

if (current == nullptr) {

return;

}

current = current->next;

}

for (int i = 0; i < n; i++) {

Element\_of\_linked\_list\* new\_element = new Element\_of\_linked\_list();

cin >> new\_element->data;

if (current == nullptr) {

new\_element->previous = tail;

if (tail != nullptr) {

tail->next = new\_element;

}

tail = new\_element;

if (head == nullptr) {

head = new\_element;

}

} else {

new\_element->next = current;

new\_element->previous = current->previous;

if (current->previous != nullptr) {

current->previous->next = new\_element;

} else {

head = new\_element;

}

current->previous = new\_element;

}

}

}

void size() const {

Element\_of\_linked\_list\* current = head;

int sizeee = 0;

while (current != nullptr) {

current = current->next;

sizeee++;

}

cout << sizeee << endl;

}

void get() const {

int index;

cin >> index;

Element\_of\_linked\_list\* current = head;

for (int i = 0; i < index; i++) {

if (current == nullptr) {

return;

}

current = current->next;

}

if (current != nullptr) {

cout << current->data << endl;

}

}

void set() {

int index, new\_value;

cin >> index >> new\_value;

Element\_of\_linked\_list\* current = head;

for (int i = 0; i < index; i++) {

if (current == nullptr) {

return;

}

current = current->next;

}

if (current != nullptr) {

current->data = new\_value;

}

}

void erase() {

int index, quantity;

cin >> index >> quantity;

Element\_of\_linked\_list\* current = head;

for (int i = 0; i < index; i++) {

if (current == nullptr) {

return;

}

current = current->next;

}

for (int i = 0; i < quantity && current != nullptr; i++) {

Element\_of\_linked\_list\* next\_element = current->next;

if (current->previous != nullptr) {

current->previous->next = next\_element;

} else {

head = next\_element;

if (head != nullptr) {

head->previous = nullptr;

}

}

if (next\_element != nullptr) {

next\_element->previous = current->previous;

}

delete current;

current = next\_element;

}

if (current == nullptr) {

tail = head;

while (tail != nullptr && tail->next != nullptr) {

tail = tail->next;

}

}

}

void request(int n) {

for (int i = 0; i < n; i++) {

string request;

cin >> request;

if (request == "print") {

print();

} else if (request == "insert") {

insert();

} else if (request == "size") {

size();

} else if (request == "get") {

get();

} else if (request == "set") {

set();

} else if (request == "erase") {

erase();

}

}

}

};

int main() {

Doubly\_Linked\_List List;

int Q;

cin >> Q;

List.request(Q);

return 0;

}

Завдання **Class Practice Work**

Посилання на файл програми у пул-запиті GitHub

1-3

<https://github.com/artificial-intelligence-department/ai_programming_playground/blob/epic_6_practice_and_labs_solomiia_trush/ai_11/solomiia_trush/Epic%206/practice_work_task_1_2_3_solomiia_trush.cpp>

#include <iostream>

#include <ctime>

using namespace std;

struct Element\_of\_linked\_list {

int data;

Element\_of\_linked\_list\* previous;

Element\_of\_linked\_list\* next;

};

Element\_of\_linked\_list\* create (int n) {

srand(time(NULL));

Element\_of\_linked\_list\* head = NULL;

int index = 0;

for (int i=0; i<n; i++) {

Element\_of\_linked\_list\* new\_element = new Element\_of\_linked\_list();

int number = rand()%21;

new\_element->data = number;

if (index==0 && head!=NULL) {

new\_element->next = head;

head->previous = new\_element;

head = new\_element;

} else if (index==0 && head==NULL) {

head = new\_element;

} else {

Element\_of\_linked\_list\* previous\_element = head;

for (int j=0; j<index-1; j++) {

previous\_element = previous\_element->next;

}

new\_element->previous = previous\_element;

new\_element->next = previous\_element->next;

if (previous\_element->next != NULL) {

previous\_element->next->previous = new\_element;

}

previous\_element->next = new\_element;

}

index++;

}

return head;

}

void print (Element\_of\_linked\_list\* head) {

Element\_of\_linked\_list\* element = head;

if (element == NULL) {

cout << "The list is empty" << endl;

} else {

while (element != NULL) {

cout << element->data << " ";

element = element->next;

}

}

cout << endl;

}

void reverse\_list (Element\_of\_linked\_list\*& head) {

Element\_of\_linked\_list\* temp = NULL;

Element\_of\_linked\_list\* current = head;

while (current != NULL) {

temp = current->previous;

current->previous = current->next;

current->next = temp;

current = current->previous;

}

if(temp != NULL ) {

head = temp->previous;

}

}

bool compare (Element\_of\_linked\_list\* head1, Element\_of\_linked\_list\* head2) {

Element\_of\_linked\_list\* element1 = head1;

Element\_of\_linked\_list\* element2 = head2;

while (element1 != nullptr && element2 != nullptr) {

if (element1->data != element2->data) {

cout << "Lists are different." << endl;

return false;

}

element1 = element1->next;

element2 = element2->next;

}

cout << "Lists are the same." << endl;

return element1 == nullptr && element2 == nullptr;

}

void delete\_list(Element\_of\_linked\_list\*& head) {

while (head != nullptr) {

Element\_of\_linked\_list\* temp = head;

head = head->next;

delete temp;

}

}

Element\_of\_linked\_list\* add(Element\_of\_linked\_list\* number1, Element\_of\_linked\_list\* number2) {

Element\_of\_linked\_list\* result = nullptr;

Element\_of\_linked\_list\* current = nullptr;

int carry = 0;

while (number1 || number2 || carry) {

int data1 = (number1 ? number1->data : 0);

int data2 = (number2 ? number2->data : 0);

int sum = data1 + data2 + carry;

carry = sum / 10;

Element\_of\_linked\_list\* new\_element = new Element\_of\_linked\_list();

new\_element->data = sum % 10;

if (!result) {

result = current = new\_element;

current->previous = nullptr; // Set previous to nullptr for the first element

} else {

current->next = new\_element;

new\_element->previous = current;

current = new\_element;

}

if (number1) number1 = (number1->next ? number1->next : nullptr);

if (number2) number2 = (number2->next ? number2->next : nullptr);

}

return result;

}

int main () {

int n = 10;

Element\_of\_linked\_list\* head1 = create(n);

print(head1);

Element\_of\_linked\_list\* head2 = create(n);

print(head2);

compare(head1, head2);

reverse\_list(head2);

print(head2);

compare(head1, head2);

delete\_list(head1);

delete\_list(head2);

int n1, n2;

cout << "Enter numbers to add : ";

cin >> n1 >> n2;

Element\_of\_linked\_list\* number1 = new Element\_of\_linked\_list;

number1->data = n1;

number1->next = nullptr;

number1->previous = nullptr;

Element\_of\_linked\_list\* number2 = new Element\_of\_linked\_list;

number2->data = n2;

number2->next = nullptr;

number2->previous = nullptr;

Element\_of\_linked\_list\* result = add(number1, number2);

reverse\_list(result);

print(result);

return 0;

}

4)

<https://github.com/artificial-intelligence-department/ai_programming_playground/blob/epic_6_practice_and_labs_solomiia_trush/ai_11/solomiia_trush/Epic%206/practice_work_task_4_solomiia_trush.cpp>

#include <iostream>

using namespace std;

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

};

TreeNode\* mirror\_tree(TreeNode\* root) {

if (root == nullptr) {

return nullptr;

} else {

TreeNode\* mirror = new TreeNode;

mirror->data = root->data;

mirror->left = mirror\_tree(root->right);

mirror->right = mirror\_tree(root->left);

return mirror;

}

}

void printTree(TreeNode\* root) {

if (root == nullptr) {

return;

} else {

printTree(root->left);

printTree(root->right);

cout << root->data << " ";

}

}

void deleteTree(TreeNode\* root) {

if (root == nullptr) {

return;

} else {

deleteTree(root->left);

deleteTree(root->right);

delete root;

}

}

int main() {

TreeNode\* root = new TreeNode();

root->data = 1;

root->left = new TreeNode();

root->left->data = 2;

root->right = new TreeNode();

root->right->data = 3;

root->left->left = new TreeNode();

root->left->left->data = 4;

root->left->right = new TreeNode();

root->left->right->data = 5;

cout << "Original Tree : ";

printTree(root);

cout << endl;

TreeNode\* mirrorRoot = mirror\_tree(root);

cout << "Mirror Tree : ";

printTree(mirrorRoot);

cout << endl;

deleteTree(root);

deleteTree(mirrorRoot);

return 0;

}

5)

<https://github.com/artificial-intelligence-department/ai_programming_playground/blob/epic_6_practice_and_labs_solomiia_trush/ai_11/solomiia_trush/Epic%206/practice_work_task_5_solomiia_trush.cpp>

#include <iostream>

using namespace std;

struct TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

};

int tree\_sum(TreeNode\* root) {

if (root == nullptr) {

return 0;

}

int leftSum = tree\_sum(root->left);

int rightSum = tree\_sum(root->right);

int nodeSum = root->data + leftSum + rightSum;

if (root->left || root->right) {

root->data = leftSum + rightSum;

}

return nodeSum;

}

void printTree(TreeNode\* root) {

if (root == nullptr) {

return;

} else {

printTree(root->left);

cout << root->data << " ";

printTree(root->right);

}

}

void deleteTree(TreeNode\* root) {

if (root == nullptr) {

return;

} else {

deleteTree(root->left);

deleteTree(root->right);

delete root;

}

}

int main() {

TreeNode\* root = new TreeNode();

root->data = 1;

root->left = new TreeNode();

root->left->data = 2;

root->right = new TreeNode();

root->right->data = 3;

root->left->left = new TreeNode();

root->left->left->data = 4;

root->left->right = new TreeNode();

root->left->right->data = 5;

cout << "Original Tree Structure : ";

printTree(root);

cout << endl;

tree\_sum(root);

cout << "Tree Structure after tree\_sum : ";

printTree(root);

cout << endl;

delete(root);

return 0;

}

Завдання **Self Practice Work**

Посилання на файл програми у пул-запиті GitHub

<https://github.com/artificial-intelligence-department/ai_programming_playground/blob/epic_6_practice_and_labs_solomiia_trush/ai_11/solomiia_trush/Epic%206/self_practice_work_algotester_task_1_solomiia_trush.cpp>

#include <iostream>

using namespace std;

int main() {

int an[100];

int bm[100];

int n, m, amin, bmin, sum;

cin >> n >> m;

for (int i = 0; i<n; i++) {

cin >> an[i];

}

for (int j = 0; j<m; j++) {

cin >> bm[j];

}

amin = an[0];

for (int y = 0; y<n; y++) {

if (amin>an[y]) {

amin = an[y];

}

}

bmin = bm[0];

for (int u = 0; u<m; u++) {

if (bmin > bm[u]) {

bmin = bm[u];

}

}

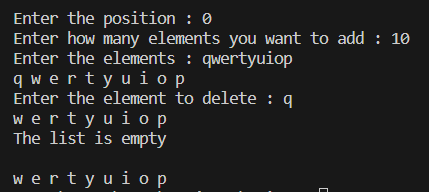
sum = amin + bmin;

cout << sum;

}

## **5. Результати виконання завдань, тестування та фактично затрачений час:**

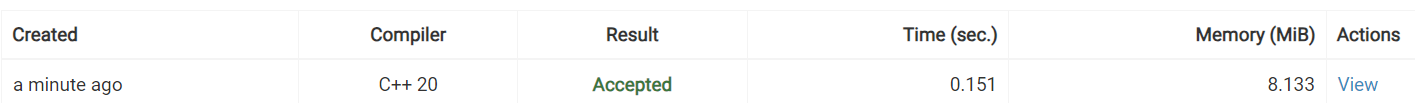
Завдання **VNS Lab 10 - Task 1** варіант 11



*Figure 2. VNS Lab 10 results*

Планований час виконання - 5 год, фактичний - 10 год

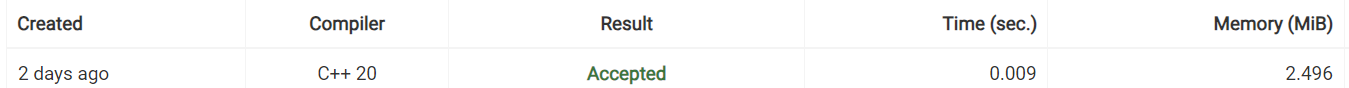
Завдання **Algotester Lab 5** варіант 3



*Figure 3. Algotester Lab 5v3 results*

Планований час виконання - 3 год , фактичний - 3 год

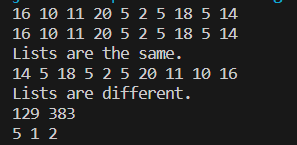
Завдання **Algotester Lab 7-8** варіант 1



*Figure 4. Algotester Lab 78v1 results*

Планований час виконання - 5 год , фактичний - 8 год

Завдання **Class Practice Work**

****

*Figure 5. Class Practise Task 1-3 results*

****

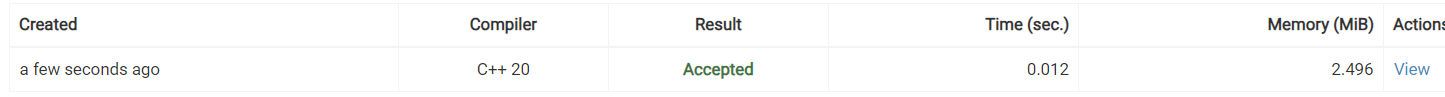
*Figure 6. Class Practice Task 4 results*

****

*Figure 7. Class Practice Task 5 results*

Планований час виконання - 4 год , фактичний - 7 год

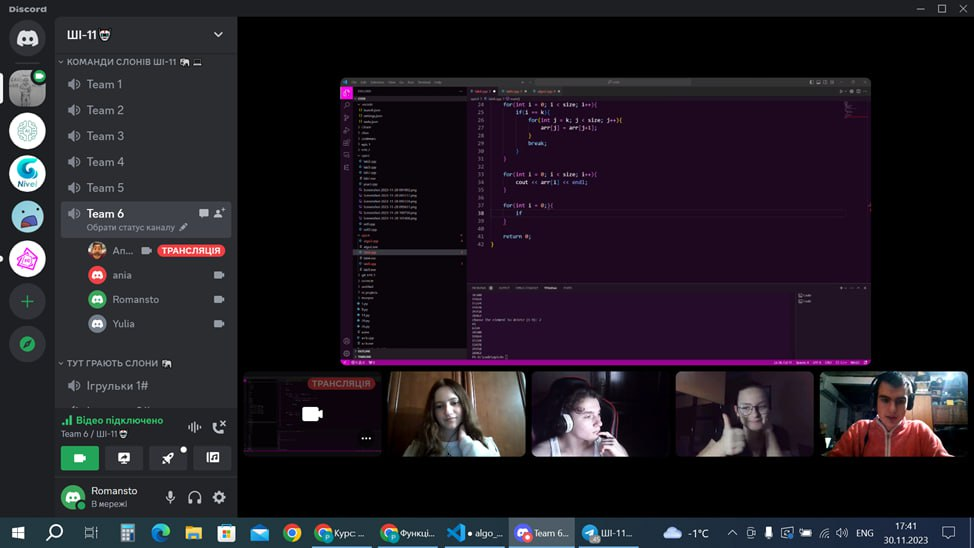
Завдання **Self Practice Work**

****

*Figure 8. Algotester Self Practice Task results*

Планований час виконання - 20 хв , фактичний - 15 хв

## **6. Кооперація з командою:**



*Figure 9. Team Cooperation*

# **Висновки:**

Ознайомилась з поняттям динамічних структур і алгоритмами їх обробки, використала засвоєні знання при виконанні практичних завдань.