Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту

A blue and white logo

Description automatically generated

**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

**Виконав:**

Студент групи ШІ-14

Іжелюк Назарій Миколайович

# **Тема роботи:**

Зв'язок динамічних структур даних (черга, стек, списки) з алгоритмами обробки дерев та їх оптимізація.

# **Мета роботи:**

Ознайомлення, вивчення та оптимізація роботи з динамічними структурами даних, такими як черга, стек, списки, а також алгоритмів обробки дерев, включаючи обхід дерев в ширину та в глибину, пошук, вставку та видалення елементів, з аналізом ефективності з точки зору часової складності та використання пам'яті, з метою оптимізації продуктивності обробки даних у деревах та їх практичним застосуванням у реалізації програм для вирішення конкретних завдань.

# **Теоретичні відомості:**

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Основні динамічні структури.
* Тема №2: Однозвязні та двозвязні списки.
* Тема №3: Дерева.
* Тема №4: Алгоритми обробки дерев.

1. Індивідуальний план опрацювання теорії:

* Тема №1: Основні динамічні структури.
  + Джерела Інформації
    - Стаття: <http://www.kievoit.ippo.kubg.edu.ua/kievoit/2016/65_C++/index.html>
  + Що опрацьовано:
    - Опрацьовано поняття динамічних структур.
    - Опрацьовано застосування динамічних структур.
  + Статус: **Ознайомлений**
  + Початок опрацювання теми: 20.11.2023
  + Звершення опрацювання теми: 24.11.2023
* Тема №2: Однозвязні та двозвязні списки.
  + Джерела Інформації:
    - Стаття:<https://erudyt.net/navchalni-predmety/informatika/prohramuvannya/linijnyj-odnozvyaznyj-spysok.html>
    - Стаття:https://www.bestprog.net/uk/2022/02/16/c-linear-doubly-linked-bidirectional-list-general-concepts-ua/
  + Що опрацьовано:
    - Опрацьовано алгоритми обробки однозвязного списку.
    - Опрацьовано алгоритми обробки двозвязного списку.
  + Статус: **Ознайомлений**
  + Початок опрацювання теми: 20.11.2023
  + Звершення опрацювання теми: 24.11.2023
* Тема №3: Дерева.
  + Джерела Інформації:
    - Відео: <http://surl.li/ohmoa>
  + Що опрацьовано:
    - Опрацьовано поняття і можливості застосування бінарного дерева.
    - Опрацьовано реалізацію бінарного дерева.
  + Статус: **Ознайомлений**
  + Початок опрацювання теми: 20.11.2023
  + Звершення опрацювання теми: 24.11.2023
* Тема №4 Алгоритми обробки дерев.
  + Джерела Інформації:
    - Стаття: <http://surl.li/ohmsh>
  + Що опрацьовано:
    - Опрацьовано алгоритми обходу дерева.
    - Пошук та вставлення елементів у дерево.
  + Статус: **Ознайомлений**
  + Початок опрацювання теми: 20.11.2023
  + Звершення опрацювання теми: 24.11.2023

# **Виконання роботи:**

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

Завдання №1 **VNS Lab 10**

* **Варіант завдання:** **23**
* **Деталі завдання:** 23.Запису в лінійному списку містять ключове поле типу \*char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим ключем.

Завдання №2 **Algotester Lab 5**

* **Варіант завдання: 2**
* **Деталі завдання:**

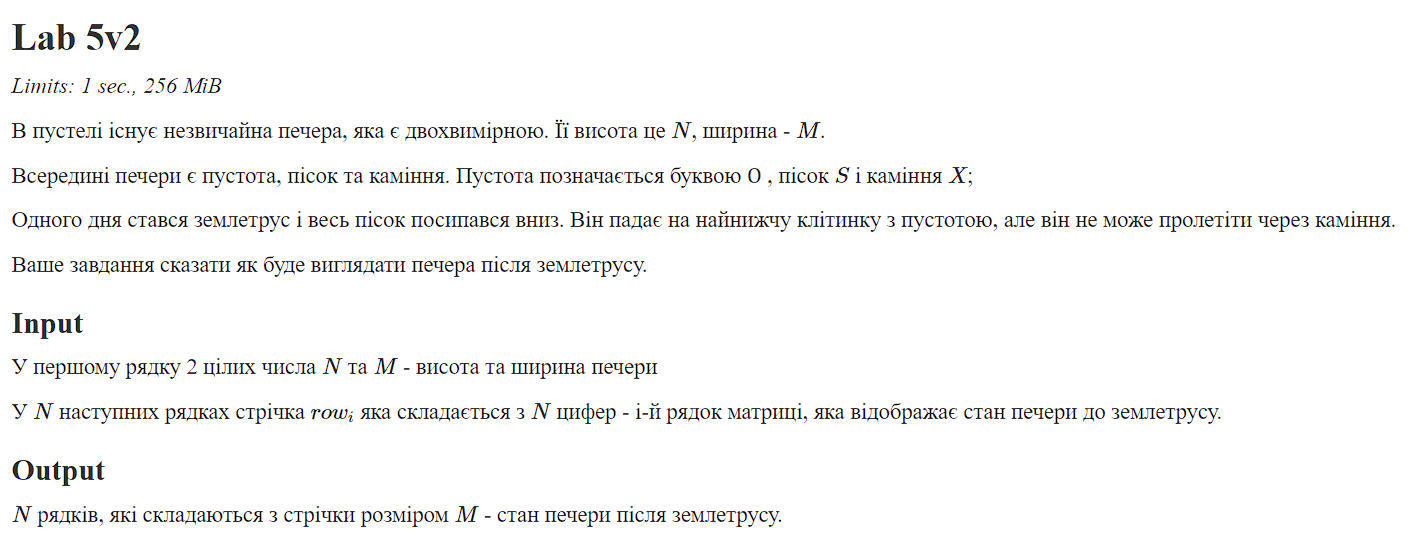
****

Рис. Завдання Algotester Lab 5

* Важливі деталі для врахування в імплементації програми: Програма повинна пройти перевірку на платформі algotester.

Завдання №3 **Algotester Lab 78**

* **Варіант завдання: 1**
* **Деталі завдання:**

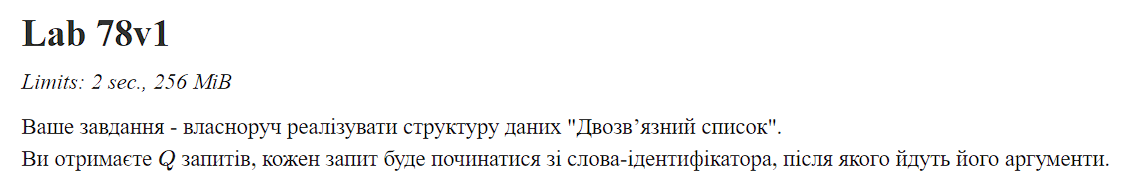
****

Рис. Завдання Algotester Lab 78

* Важливі деталі для врахування в імплементації програми: Програма повинна пройти перевірку на платформі algotester.

Завдання №4 **Class Practice**

* **Варіант завдання:**
* **Деталі завдання:**

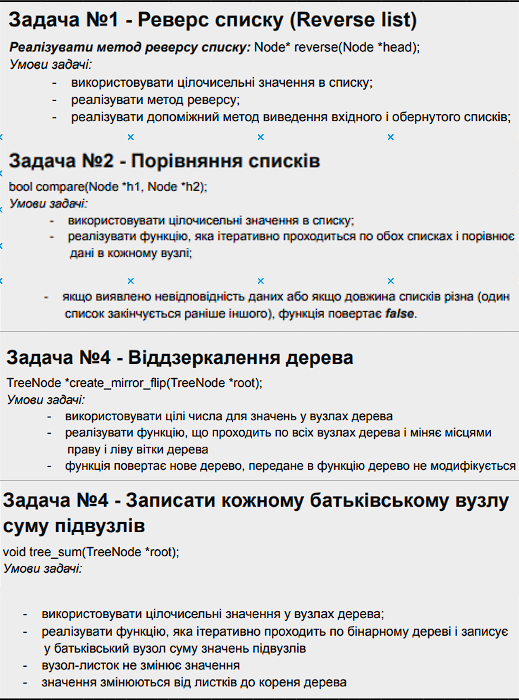
****

Рис. Завдання Class Practice

* Важливі деталі для врахування в імплементації програми: Потрібно дотримуватись вимог завдання.

Завдання №5 **Self Practice**

* **Варіант завдання:**
* **Деталі завдання:**
* Важливі деталі для врахування в імплементації програми

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

Програма №2 **Algotester Lab 5**

* Блок-схема:

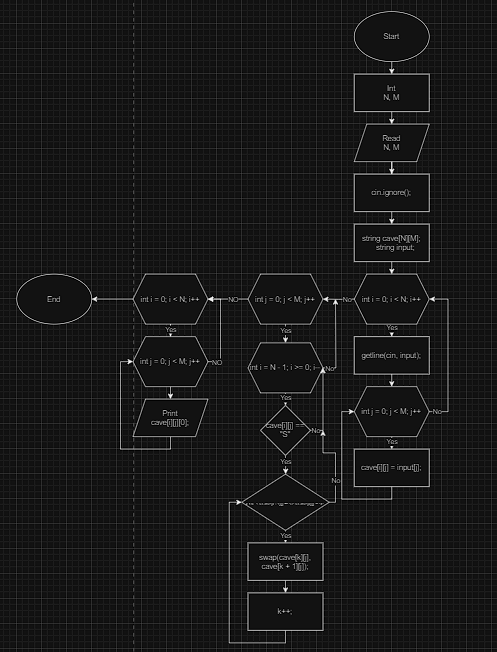


Рис. Блок схема Algotester Lab 5

* Планований час на реалізацію: 40 хв.

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

Конфігурацію середовища не змінював.

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

Завдання №1 VNS Lab 10

#include<iostream>

#include<string>

using namespace std;

struct Node {

int data;

Node\* previous;

Node\* next;

};

class Double\_list {

public:

Node\* head;

Node\* tail;

int size;

Double\_list() {

head = nullptr;

tail = nullptr;

size = 0;

}

void addElements(int N) {

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

int value;

cin >> value;

Node\* newNode = new Node();

newNode->data = value;

newNode->next = nullptr;

if (head == nullptr) {

newNode->previous = nullptr;

head = newNode;

tail = newNode;

} else {

newNode->previous = tail;

tail->next = newNode;

tail = newNode;

}

size++;

}

}

void print() {

Node\* current = head;

cout << "Double list: " ;

while (current != nullptr) {

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

current = current->next;

}

cout << endl;

}

void deleteK(int K) {

Node\* current = head;

for (int i = 1; i < K; ++i) {

current = current->next;

}

current->previous->next = current->next;

current->next->previous = current->previous;

delete current;

size--;

}

};

int main() {

int N, K;

cout << "Enter the number of elements: ";

cin >> N;

Double\_list list1;

cout << "Enter elements: ";

list1.addElements(N);

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

cin >> K;

list1.deleteK(K);

list1.print();

return 0;

}

Завдання №2 **Algotester Lab 5**

**#include <iostream>**

**#include <string>**

**using namespace std;**

**int main() {**

**int N, M;**

**cin >> N >> M;**

**cin.ignore();**

**string cave[N][M];**

**string input;**

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

**getline(cin, input);**

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

**cave[i][j] = input[j];**

**}**

**}**

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

**for (int i = N - 1; i >= 0; i--) {**

**if (cave[i][j] == "S") {**

**int k = i;**

**while (k < N - 1 && cave[k + 1][j] == "O" && cave[k][j] != "X") {**

**swap(cave[k][j], cave[k + 1][j]);**

**k++;**

**}**

**}**

**}**

**}**

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

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

**cout << cave[i][j][0];**

**}**

**cout << endl;**

**}**

**return 0;**

**}**

Завдання №3 **Algotester Lab 78**

**#include <iostream>**

**#include <vector>**

**using namespace std;**

**void insert(vector<int>& arr, int index, int N, vector<int>& elements) {**

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

**arr.insert(arr.begin() + index + i, elements[i]);**

**}**

**}**

**void erase(vector<int>& arr, int index, int n) {**

**arr.erase(arr.begin() + index, arr.begin() + index + n);**

**}**

**int getSize(const vector<int>& arr) {**

**return arr.size();**

**}**

**int get(const vector<int>& arr, int index) {**

**return arr[index];**

**}**

**void set(vector<int>& arr, int index, int newValue) {**

**arr[index] = newValue;**

**}**

**void print(const vector<int>& arr) {**

**for (int i = 0; i < arr.size(); ++i) {**

**cout << arr[i] << " ";**

**}**

**cout << endl;**

**}**

**int main() {**

**int Q;**

**cin >> Q;**

**vector<int> List;**

**for (int i = 0; i < Q; ++i) {**

**string input;**

**cin >> input;**

**if (input == "insert") {**

**int index, N;**

**cin >> index >> N;**

**vector<int> elements(N);**

**for (int j = 0; j < N; ++j) {**

**cin >> elements[j];**

**}**

**insert(List, index, N, elements);**

**} else if (input == "erase") {**

**int index, n;**

**cin >> index >> n;**

**erase(List, index, n);**

**} else if (input == "size") {**

**cout << getSize(List) << endl;**

**} else if (input == "get") {**

**int index;**

**cin >> index;**

**cout << get(List, index) << endl;**

**} else if (input == "set") {**

**int index, newValue;**

**cin >> index >> newValue;**

**set(List, index, newValue);**

**} else if (input == "print") {**

**print(List);**

**}**

**}**

**return 0;**

**}**

Завдання №4 **Class Practice**

**#include<iostream>**

**using namespace std;**

**struct Node**

**{**

**int data;**

**Node\* next;**

**};**

**void show(Node\* head){**

**Node\* curent = head;**

**while(curent != nullptr){**

**cout << curent->data << " ";**

**curent = curent->next;**

**}**

**}**

**Node\* reverse(Node\* head) {**

**Node \*prev = nullptr, \*current = head, \*nextNode = nullptr;**

**while (current != nullptr) {**

**nextNode = current->next;**

**current->next = prev;**

**prev = current;**

**current = nextNode;**

**}**

**return prev;**

**}**

**int main(){**

**Node\* head = nullptr;**

**head = new Node{3, nullptr};**

**head->next = new Node{5, nullptr};**

**head->next->next = new Node{45, nullptr};**

**head->next->next->next = new Node{11, nullptr};**

**show(head);**

**head = reverse(head);**

**cout << endl;**

**show(head);**

**return 0;**

**}**

**#include<iostream>**

**using namespace std;**

**struct Node**

**{**

**int data;**

**Node\* next;**

**};**

**void show(Node\* head){**

**Node\* curent = head;**

**while(curent != nullptr){**

**cout << curent->data << " ";**

**curent = curent->next;**

**}**

**}**

**bool compare(Node\* head, Node\* head2) {**

**while (head != nullptr && head2 != nullptr) {**

**if (head->data != head2->data) {**

**return false;**

**}**

**head = head->next;**

**head2 = head2->next;**

**}**

**if (head != nullptr || head2 != nullptr) {**

**return false;**

**}**

**return true;**

**}**

**int main(){**

**Node\* head = nullptr;**

**head = new Node{3, nullptr};**

**head->next = new Node{5, nullptr};**

**head->next->next = new Node{45, nullptr};**

**head->next->next->next = new Node{11, nullptr};**

**Node\* head2 = nullptr;**

**head2 = new Node{1, nullptr};**

**head2->next = new Node{33, nullptr};**

**head2->next->next = new Node{42, nullptr};**

**head2->next->next->next = new Node{5, nullptr};**

**cout << compare(head, head2);**

**return 0;**

**}**

**#include <iostream>**

**using namespace std;**

**struct Node {**

**int data;**

**Node\* left;**

**Node\* right;**

**Node(int value) {**

**data = value;**

**left = nullptr;**

**right = nullptr;**

**}**

**};**

**class BinaryTree {**

**private:**

**Node\* root;**

**public:**

**BinaryTree() {**

**root = nullptr;**

**}**

**void insert(int value) {**

**root = insertRecursive(root, value);**

**}**

**Node\* insertRecursive(Node\* current, int value) {**

**if (current == nullptr) {**

**return new Node(value);**

**}**

**if (value < current->data) {**

**current->left = insertRecursive(current->left, value);**

**} else if (value > current->data) {**

**current->right = insertRecursive(current->right, value);**

**}**

**return current;**

**}**

**void display() {**

**displayRecursive(root);**

**}**

**void displayRecursive(Node\* current) {**

**if (current != nullptr) {**

**displayRecursive(current->left);**

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

**displayRecursive(current->right);**

**}**

**}**

**Node\* create\_mirror\_flip(Node\* node) {**

**if (node == nullptr) {**

**return nullptr;**

**}**

**Node\* mirrored = new Node(node->data);**

**mirrored->left = create\_mirror\_flip(node->right);**

**mirrored->right = create\_mirror\_flip(node->left);**

**return mirrored;**

**}**

**Node\* getRoot() {**

**return root;**

**}**

**};**

**int main() {**

**BinaryTree tree;**

**tree.insert(8);**

**tree.insert(3);**

**tree.insert(10);**

**tree.insert(1);**

**tree.insert(6);**

**tree.insert(14);**

**tree.insert(4);**

**tree.insert(7);**

**tree.insert(13);**

**cout << "Дерево: ";**

**tree.display();**

**cout << endl;**

**Node\* mirroredRoot = tree.create\_mirror\_flip(tree.getRoot());**

**cout << "Відзеркалення дерева: ";**

**BinaryTree mirroredTree;**

**mirroredTree.displayRecursive(mirroredRoot);**

**cout << endl;**

**return 0;**

**}**

## **#include <iostream>**

## **using namespace std;**

## **struct Node {**

## **int data;**

## **Node\* left;**

## **Node\* right;**

## **Node(int value) {**

## **data = value;**

## **left = nullptr;**

## **right = nullptr;**

## **}**

## **};**

## **class BinaryTree {**

## **private:**

## **Node\* root;**

## **public:**

## **BinaryTree() {**

## **root = nullptr;**

## **}**

## **void insert(int value) {**

## **root = insertRecursive(root, value);**

## **}**

## **Node\* insertRecursive(Node\* current, int value) {**

## **if (current == nullptr) {**

## **return new Node(value);**

## **}**

## **if (value < current->data) {**

## **current->left = insertRecursive(current->left, value);**

## **} else if (value > current->data) {**

## **current->right = insertRecursive(current->right, value);**

## **}**

## **return current;**

## **}**

## **void display() {**

## **displayRecursive(root);**

## **}**

## **void displayRecursive(Node\* current) {**

## **if (current != nullptr) {**

## **displayRecursive(current->left);**

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

## **displayRecursive(current->right);**

## **}**

## **}**

## **void updateParentNodeSums() {**

## **updateParentSumsRecursive(root);**

## **}**

## **int updateParentSumsRecursive(Node\* current) {**

## **if (current == nullptr) {**

## **return 0;**

## **}**

## **int leftSum = updateParentSumsRecursive(current->left);**

## **int rightSum = updateParentSumsRecursive(current->right);**

## **int sum = leftSum + rightSum + current->data;**

## **if (current->left != nullptr || current->right != nullptr) {**

## **current->data = leftSum + rightSum;**

## **}**

## **return sum;**

## **}**

## **};**

## **int main() {**

## **BinaryTree tree;**

## **tree.insert(8);**

## **tree.insert(3);**

## **tree.insert(10);**

## **tree.insert(1);**

## **tree.insert(6);**

## **tree.insert(14);**

## **tree.insert(4);**

## **tree.insert(7);**

## **tree.insert(13);**

## **cout << "Elements of the binary tree in order: ";**

## **tree.display();**

## **cout << endl;**

## **tree.updateParentNodeSums();**

## **cout << "Elements of the binary tree after updating parent nodes with sums: ";**

## **tree.display();**

## **cout << endl;**

## **return 0;**

## **}**

Завдання №4 **Self Practice**

**#include <iostream>**

**using namespace std;**

**int main() {**

**int n;**

**cin >> n;**

**string s;**

**cin >> s;**

**int signal = 15;**

**for(char c : s) {**

**if(c == 'k') {**

**signal = max(signal-1, 0);**

**}**

**else if(c == 'p') {**

**signal = (signal > 0) ? 15 : 0;**

**}**

**else {**

**signal = (signal > 0) ? 0 : 15;**

**}**

**}**

**cout << signal;**

**return 0;**

**}**

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

Завдання №1 VNS Lab 10

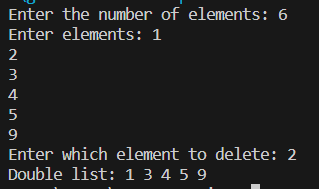


Рис. Результат тестування Vns Lab 10

Фактично затрачений час: 1.5 год

Завдання №2 Algotester Lab 5

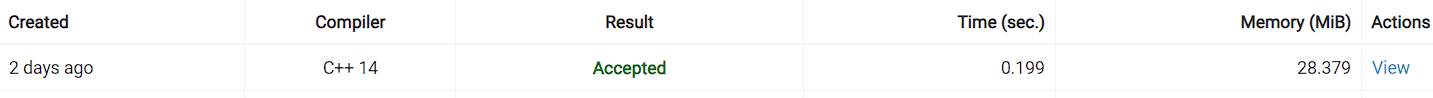


Рис. 6 Результат тестування Algotester Lab 5

Фактично затрачений час: 50 хв

Завдання №3 Algotester Lab 78



Рис. 7 Результат тестування Algotester Lab 78

Фактично затрачений час: 1.5 год

Завдання №4 Class Practice



Рис. 8 Результат тестування Class Practice(1)



Рис. 9 Результат тестування Class Practice(2)



Рис. 10 Результат тестування Class Practice(3)

Фактично затрачений час: 1.5 год

Завдання №5 Self Practice

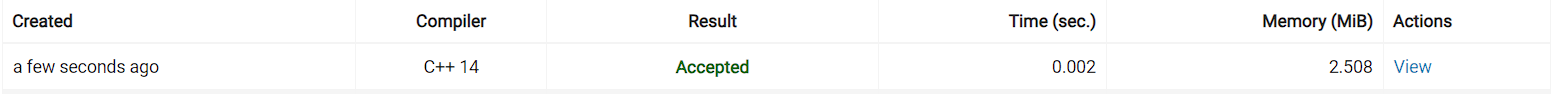


Рис. 11 Результат тестування Sеlf Practice

Фактично затрачений час: 30 хв.

# **Висновки:**

У ході виконання даної роботи було детально розглянуто зв'язок між динамічними структурами даних (черга, стек, списки) та алгоритмами обробки дерев. Основною метою було ознайомлення, вивчення та оптимізація роботи з цими структурами, а також алгоритмами, що стосуються обробки даних у деревах.