Міністерство освіти і науки України

Національний університет «Львівська політехніка»

Кафедра систем штучного інтелекту



**Звіт**

про виконання

**Лабораторних та практичних робіт № 6(Algolab 5. Algolab 78.VNS Lab 10. Class Practice Work. Self Practice Work)**

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

***з розділу***: «Епік 5: « Динамічні структури. Бінарне дерево. Динамічний масив. Лінкед ліст»

***Виконав:***

***Бачик Анатолій Андрійович*** групи ШІ-13

### Тема роботи:

Динамічні структури. Бінарне дерево. Динамічний масив. Лінкед ліст

### Мета роботи:

Ознайомлення з динамічними структурами,бінарним деревом, динамічним масивов, лінкед лістом

### Теоретичні відомості:

1. Теоретичні відомості з переліком важливих тем:

* Тема №1: Динамічні структури
* Тема №2: Бінарне дерево
* Тема №3 Динамічний масив
* Тема №4 Лінкед і дабл лінкед ліст

1. Індивідуальний план опрацювання теорії:

**Тема №1 Динамічні структури**

* + Джерела Інформації

https://purecodecpp.com/uk/archives/1538

* + Що опрацьовано:

Матеріал з джерел інформації

* + Статус: Ознайомлений
  + Початок опрацювання теми: 5.12.2023
  + Звершення опрацювання теми: 5.12.2023

**Тема №2 Бінарне дерево**

* + Джерела Інформації:

https://www.bestprog.net/ru/2023/08/01/c-binary-search-tree-binarytree-class-integers-ru/#google\_vignette

* + Що опрацьовано:

Матеріал з джерел інформації

* + Статус: Ознайомлений
  + Початок опрацювання теми: 5.12.2023
  + Звершення опрацювання теми: 5.12.2023

**Тема №3 Динамічний масив**

* + Джерела Інформації

https://acode.com.ua/urok-90-dynamichni-masyvy/

Що опрацьовано:

Матеріал з джерел інформації

* + Статус: Ознайомлений
  + Початок опрацювання теми: 5.12.2023
  + Звершення опрацювання теми: 5.12.2023

**Тема №4 Лінкед і дабл лінкед ліст**

* + Джерела Інформації

https://www.bestprog.net/uk/2022/02/13/c-an-example-of-the-implementation-of-a-linear-singly-linked-list-ua/

Що опрацьовано:

Матеріал з джерел інформації

* + Статус: Ознайомлений
  + Початок опрацювання теми: 5.12.2023
  + Звершення опрацювання теми: 5.12.2023

### Виконання роботи:

**Завдання №1**

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

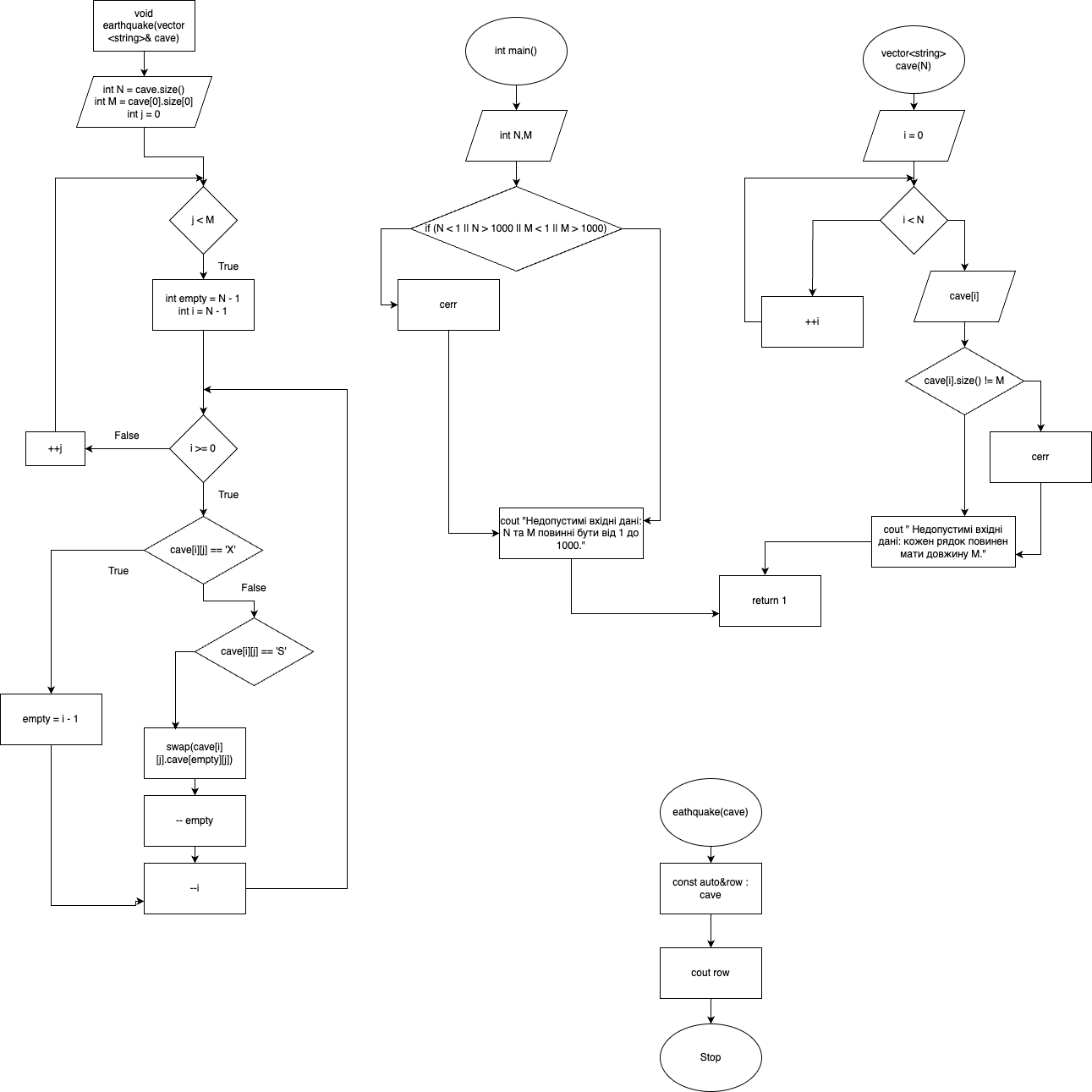
**Завдання №1**

Тут вказуєте

-Варіант 2,Algolab 5

У даній задачі, потрібно симулювати землетрус у двовимірній печері. В печері є пустота (означена символом "O"), пісок ("S") та каміння ("X"). Землетрус призводить до того, що весь пісок падає вниз до найнижчої пустої клітинки, але не може пройти через каміння. Потрібно розмістити пісок під камінням, якщо під камінням є пустота. Тобто, якщо нижня клітина каміння є пустою, пісок опиниться на цій клітинці. Процес продовжується до тих пір, поки пісок не опиниться на нижніх клітинах каміння або досягне дна печери.

1. *Дизайн та планована оцінка часу виконання завдань:*

**

1. *Код програми*

#include <vector>

#include <iostream>

using namespace std;

void earthquake(vector<string>& cave) {

int N = cave.size();

int M = cave[0].size();

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

int empty = N - 1;

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

if (cave[i][j] == 'X') {

empty = i - 1;

} else if (cave[i][j] == 'S') {

swap(cave[i][j], cave[empty][j]);

--empty;

}

}

}

}

int main() {

int N, M;

cin >> N >> M;

// Перевірка обмежень

if (N < 1 || N > 1000 || M < 1 || M > 1000) {

cerr << "Недопустимі вхідні дані: N та M повинні бути від 1 до 1000." << endl;

return 1;

}

vector<string> cave(N);

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

cin >> cave[i];

if (cave[i].size() != M) {

cerr << "Недопустимі вхідні дані: кожен рядок повинен мати довжину M." << endl;

return 1;

}

}

earthquake(cave);

for (const auto& row : cave) {

cout << row << endl;

}

return 0;

}

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

****

Потрачено часу: 1 день

**Завдання №2**

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

-Варіант 3,Algolab 78

- Для реалізації цієї задачі, нам потрібно створити структуру даних "Двійкове дерево пошуку" та реалізувати методи для вставки, пошуку, підрахунку розміру та виведення елементів дерева. Також потрібно використовувати перегрузку оператора << для виведення елементів дерева на екран.

Основні методи, які ми повинні реалізувати:

insert(value) - вставляє елемент у дерево.

contains(value) - перевіряє, чи міститься певне значення у дереві.

size() - повертає кількість елементів у дереві.

print() - виводить елементи дерева.

1. *Код програми*

#include <iostream>

#include <string>

using namespace std;

class TreeNode {

int data;

TreeNode\* left;

TreeNode\* right;

public:

TreeNode(int data) : data(data), left(nullptr), right(nullptr) {}

friend class Tree;

};

class Tree {

TreeNode\* root;

TreeNode\* addNode(TreeNode\* node, int data) {

if (node == nullptr) {

return new TreeNode(data);

}

if (data < node->data) {

node->left = addNode(node->left, data);

} else if (data > node->data) {

node->right = addNode(node->right, data);

}

return node;

}

bool findNode(TreeNode\* node, int data) const {

if (node == nullptr) {

return false;

}

if (data == node->data) {

return true;

} else if (data < node->data) {

return findNode(node->left, data);

} else {

return findNode(node->right, data);

}

}

int countNodes(TreeNode\* node) const {

if (node == nullptr) {

return 0;

}

return 1 + countNodes(node->left) + countNodes(node->right);

}

void displayTree(TreeNode\* node, ostream& os) const {

if (node != nullptr) {

displayTree(node->left, os);

os << node->data << " ";

displayTree(node->right, os);

}

}

public:

Tree() : root(nullptr) {}

void addNode(int data) {

root = addNode(root, data);

}

bool findNode(int data) const {

return findNode(root, data);

}

int countNodes() const {

return countNodes(root);

}

friend ostream& operator<<(ostream& os, const Tree& tree) {

tree.displayTree(tree.root, os);

return os;

}

};

int main() {

int numQueries;

cin >> numQueries;

Tree tree;

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

string queryType;

cin >> queryType;

if (queryType == "insert") {

int data;

cin >> data;

tree.addNode(data);

} else if (queryType == "contains") {

int data;

cin >> data;

cout << (tree.findNode(data) ? "Yes" : "No") << endl;

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

cout << tree.countNodes() << endl;

} else if (queryType == "print") {

cout << tree << endl;

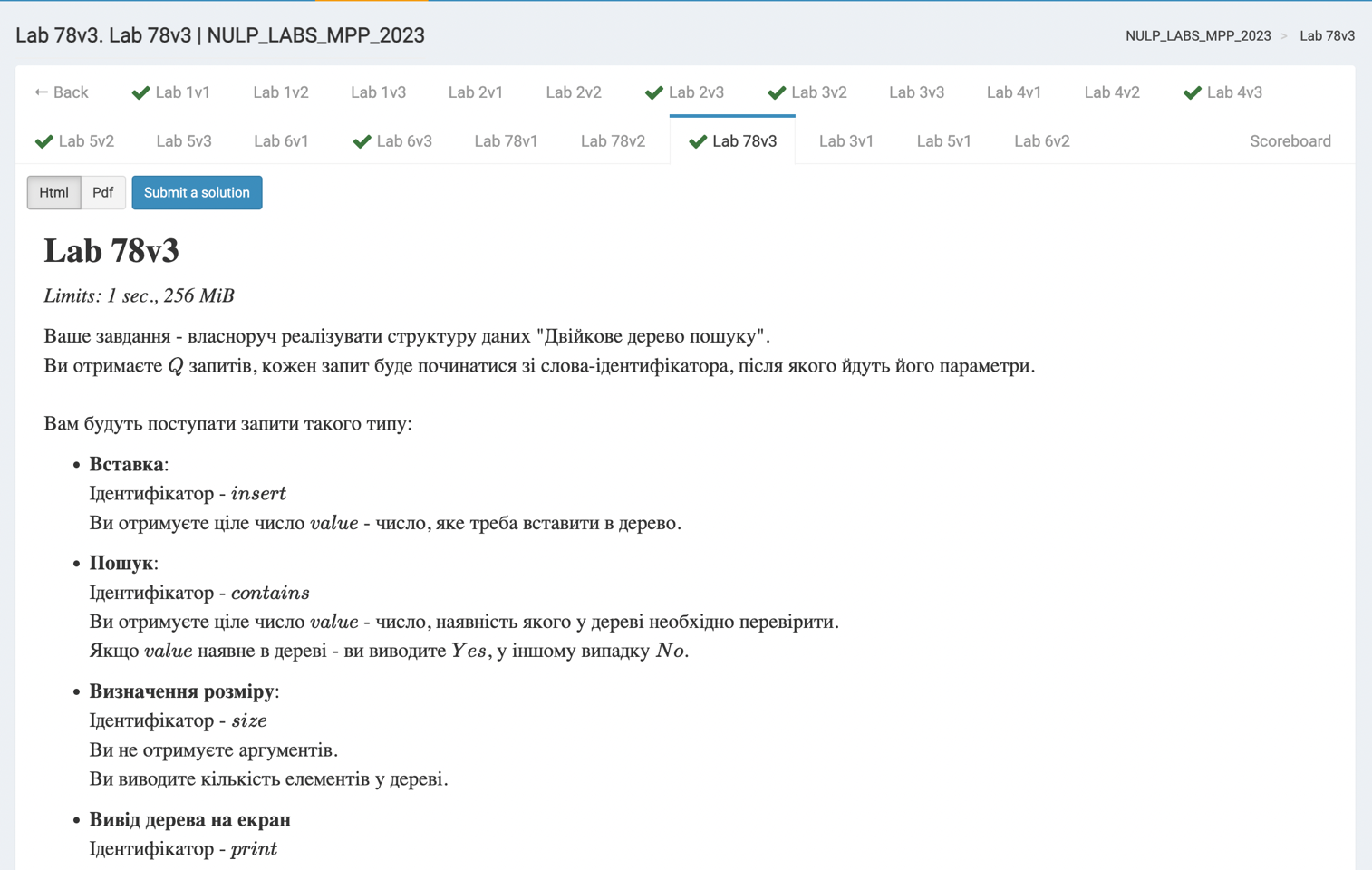
}

}

return 0;

}

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



Потрачено часу: 1 день

**Завдання №3**

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

-Варіант 4,VNS Lab 10

Описана задача полягає у розробці програми, яка працює з лінійним списком і виконує різні операції з ним:

1. Створення списку: Написання функції для створення порожнього списку, яка може додавати елементи в цей список.
2. Друк списку: Функція, яка друкує елементи списку, а також виводить повідомлення, якщо список порожній.
3. Додавання елементів у список: Функція для додавання та видалення елементів зі списку відповідно до варіанта.
4. Виконання змін в списку та друк: Після кожної зміни в списку потрібно виводити оновлений список.
5. Запис списку у файл: Функція для записування списку у файл.
6. Знищення списку: Функція, яка повністю знищує список.
7. Запис та знищення списку з друкуванням: Запис списку у файл, потім його знищення і виведення повідомлення, якщо список порожній.
8. Відновлення списку з файлу: Функція для відновлення списку з файлу.
9. Роздруковування відновленого списку: Відновлення та роздруковування списку з файлу.
10. Знищення списку, додавання і видалення елементів: Створення однонаправленого списку, знищення елементу за заданим номером, а потім додавання К елементів, починаючи з заданого номера.
11. *Код програми*

#include <iostream>

#include <fstream>

using namespace std;

struct Node {

int data;

Node\* next;

};

// Функція для створення порожнього списку

Node\* createEmptyList() {

return nullptr;

}

// Функція для додавання елементів в кінець списку

void addEl(Node\* &head, int value) {

Node\* newNode = new Node;

newNode->data = value;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr) {

temp = temp->next;

}

temp->next = newNode;

}

}

// Функція для друку списку

void printList(Node\* head) {

if (head == nullptr) {

cout << "Список порожній" << endl;

return;

}

Node\* temp = head;

while (temp != nullptr) {

cout << temp->data << " ";

temp = temp->next;

}

cout << endl;

}

// Функція для знищення списку

void destroyList(Node\* &head) {

while (head != nullptr) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

// Функція для запису списку у файл

void writeListToFile(Node\* head, const string& filename) {

ofstream file(filename);

Node\* temp = head;

while (temp != nullptr) {

file << temp->data << " ";

temp = temp->next;

}

file.close();

}

// Функція для відновлення списку з файлу

Node\* restoreListFromFile(const string& filename) {

ifstream file(filename);

Node\* head = nullptr;

int value;

while (file >> value) {

addEl(head, value);

}

file.close();

return head;

}

int main() {

Node\* List = createEmptyList();

addEl(List, 10);

addEl(List, 20);

addEl(List, 30);

cout << "Початковий список: ";

printList(List);

writeListToFile(List, "List.txt");

destroyList(List);

cout << "Список після знищення: ";

printList(List);

List = restoreListFromFile("List.txt");

cout << "Відновлений список: ";

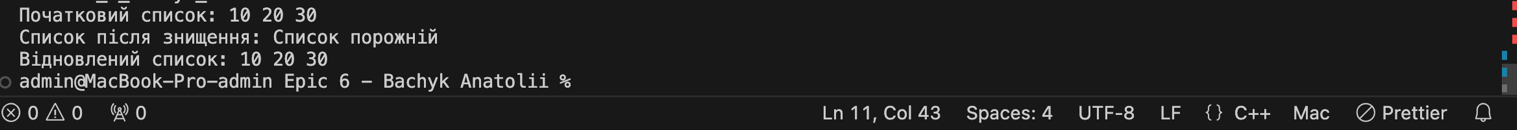
printList(List);

destroyList(List);

return 0;

}

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

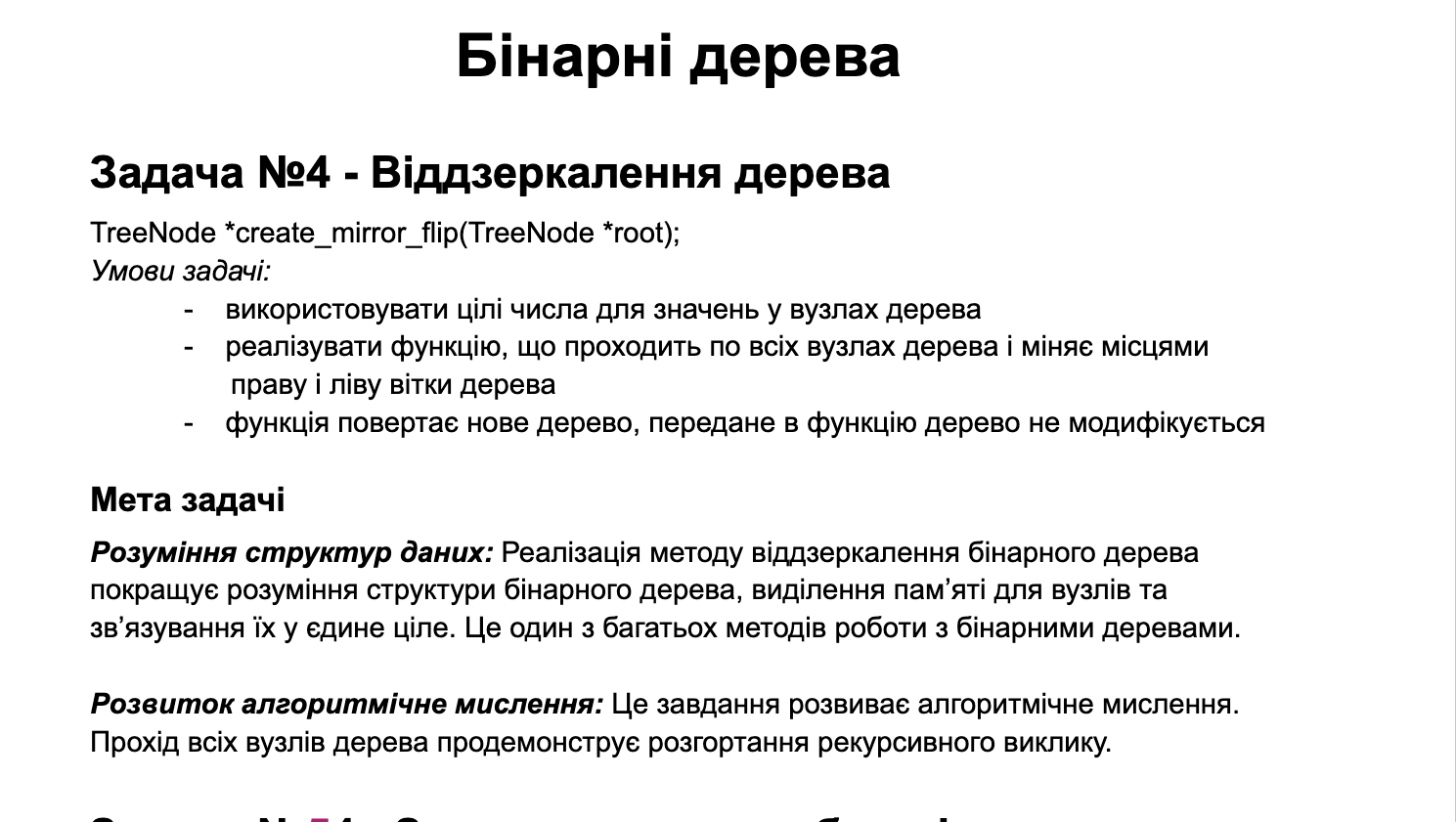


Потрачено часу: 1 день

**Завдання №4**

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

- Class Practice Work



1. *Код програми*

#include <iostream>

using namespace std;

// Створення структури для вузла бінарного дерева

struct TreeNode {

int val;

TreeNode \*left;

TreeNode \*right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

// Функція для створення віддзеркаленого дерева

TreeNode\* create\_mirror\_flip(TreeNode \*root) {

if (root == nullptr) {

return nullptr;

}

TreeNode \*newNode = new TreeNode(root->val);

newNode->left = create\_mirror\_flip(root->right);

newNode->right = create\_mirror\_flip(root->left);

return newNode;

}

// Функція для виведення значень вузлів бінарного дерева

void print\_tree(TreeNode \*root) {

if (root == nullptr) {

return; // Якщо вузол порожній (null), завершуємо виконання функції

}

cout << root->val << " " ;

// Рекурсивний виклик для лівого та правого піддерев

print\_tree(root->left);

print\_tree(root->right);

}

int main() {

// створення бінарного дерева

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);

cout << "Оригінальне дерево: ";

print\_tree(root);

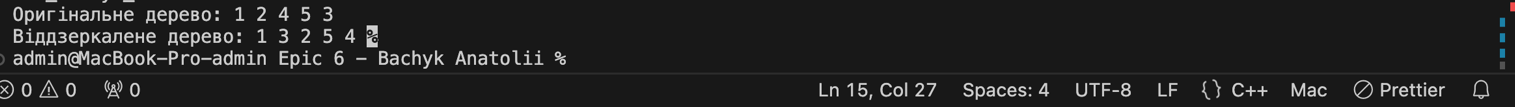
TreeNode \*mirrorTree = create\_mirror\_flip(root); // Створення віддзеркаленого дерева

cout << "\nВіддзеркалене дерево: ";

print\_tree(mirrorTree);

return 0;

}

**

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

Потрачено часу: 1 день

**Завдання №5**

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

-Self Practice Work

- Даний алгоритм має на меті визначити, які картини (індекси) не вистачають у колоді, враховуючи, що колода карт легша, ніж зазвичай.

Вхідні дані:

Перший рядок містить вагу карт в руках Тойлет-мена.

Другий рядок містить кількість карт у колоді.

Наступні рядки містять вагу кожної карти у колоді.

Вихідні дані:

Якщо неможливо визначити, які картини не вистачають, програма поверне єдине число 0.

У випадку, якщо є декілька можливих розв'язків, виведеся -1.

В іншому випадку виведуться індекси карт, які відсутні в колоді, в порядку зростання.

Ця задача вимагає аналізу ваги карт в руках гравця та у колоді, з метою визначення, які картини відсутні в колоді, припускаючи, що колода стала легшою.

1. *Код програми*

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Card {

int weight;

int index;

};

// Функція порівняння для сортування карт за їх вагою

bool compareByWeight(const Card &a, const Card &b) {

return a.weight < b.weight;

}

int main() {

int weightInHand;

cin >> weightInHand;

int n;

cin >> n;

vector<Card> cards(n);

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

cin >> cards[i].weight;

cards[i].index = i + 1;

}

sort(cards.begin(), cards.end(), compareByWeight);

int totalWeight = 0;

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

totalWeight += cards[i].weight;

}

vector<int> missingCards;

// Знаходимо різницю між вагою колоди та вагою карт у руках

int diff = weightInHand - totalWeight;

if (diff < 0) {

cout << 0 << endl; // Неможливо отримати таку вагу колоди

} else if (diff > 0) {

// Шукаємо карти, яких не вистачає в колоді

int i = 0;

while (diff > 0 && i < n) {

diff -= cards[i].weight;

missingCards.push\_back(cards[i].index);

++i;

}

if (diff != 0) {

cout << "Неможливо знайти відповідь" << endl;

} else {

// Виводимо індекси карт, яких не вистачає в колоді

for (int idx : missingCards) {

cout << idx << " " << endl;

}

}

} else {

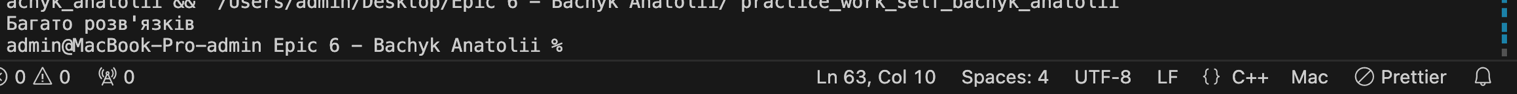
cout << "Багато розв'язків" << endl;

}

return 0;

}

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



Потрачено часу: *2 години*

### Висновки:

Ознайомлений з динамічними структурами,бінарним деревом, динамічним масивов, лінкед лістом.