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(ла):
Студент групи ШІ-11
Зубрицький Арсеній Юрійович

Львів 2024

Тема роботи:

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

Мета роботи:

Реалізувати різні динамічні структури і функції для роботи з ними.

Теоретичні відомості:

Теоретичні відомості з переліком важливих тем:

1. Основи Динамічних Структур Даних:
    - Вступ до динамічних структур даних: визначення та важливість
    - Виділення пам'яті для структур даних (stack і heap)
    - Приклади простих динамічних структур: динамічний масив
2. Стек:
    - Визначення та властивості стеку
    - Операції push, pop, top: реалізація та використання
    - Приклади використання стеку: обернений польський запис, перевірка балансу дужок
    - Переповнення стеку
3. Черга:
    - Визначення та властивості черги
    - Операції enqueue, dequeue, front: реалізація та застосування
    - Приклади використання черги: обробка подій, алгоритми планування
    - Розширення функціоналу черги: пріоритетні черги
4. Зв'язні Списки:
    - Визначення однозв'язного та двозв'язного списку
    - Принципи створення нових вузлів, вставка між існуючими, видалення, створення кільця(circular linked list)
    - Основні операції: обхід списку, пошук, доступ до елементів, об'єднання списків
    - Приклади використання списків: управління пам'яттю, FIFO та LIFO структури
5. Дерева:
    - 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
6. Алгоритми Обробки Динамічних Структур:

    - Основи алгоритмічних патернів: ітеративні, рекурсивні
    - Алгоритми пошуку, сортування даних, додавання та видалення елементів
- Індивідуальний план опрацювання теорії:
    - C++ • Теорія • Урок 139 • ADT • Однозв'язний список
    - C++ • Теорія • Урок 140 • ADT • Двозв'язний список
    -

Виконання роботи:

1. Опрацювання завдання та вимог до програм та середовища:

Завдання №1 **Vns Lab_5_variant_2**

Lab 5v2

Обмеження: 1 сек., 256 МіБ

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це N, ширина - M.

Всередині печери є пустота, пісок та каміння. Пустота позначається буквою O , пісок S і каміння X;

Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння.

Ваше завдання сказати як буде виглядати печера після землетрусу.

Вхідні дані

У першому рядку 2 цілих числа N та M - висота та ширина печери

У N наступних рядках стрічка row_i яка складається з N цифер - і-й рядок матриці, яка відображає стан печери до землетрусу.

Вихідні дані

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

Завдання №2 **VNS Lab_7_8_variant_3**

Ваше завдання - власноруч реалізувати структуру даних "Двійкове дерево пошуку".
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його параметри.

Вам будуть поступати запити такого типу:

- **Вставка**:
  Ідентифікатор - $insert$
  Ви отримуєте ціле число $value$ - число, яке треба вставити в дерево.
- **Пошук**:
  Ідентифікатор - $contains$
  Ви отримуєте ціле число $value$ - число, наявність якого у дереві необхідно перевірити.
  Якщо $value$ наявне в дереві - ви виводите Yes, у іншому випадку No.
- **Визначення розміру**:
  Ідентифікатор - $size$
  Ви не отримуєте аргументів.
  Ви виводите кількість елементів у дереві.
- **Вивід дерева на екран**
  Ідентифікатор - $print$
  Ви не отримуєте аргументів.
  Ви виводите усі елементи дерева 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.

Завдання №3 **VNS Lab_10_task_1_4**

2. Постановка завдання

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.

Порядок виконання роботи

1. Написати функцію для створення списку. Функція може створювати порожній список, а потім додавати в нього елементи.
2. Написати функцію для друку списку. Функція повинна передбачати вивід повідомлення, якщо список порожній.
3. Написати функції для знищення й додавання елементів списку у відповідності зі своїм варіантом.

---

4. Виконати зміни в списку й друк списку після кожної зміни.
5. Написати функцію для запису списку у файл.
6. Написати функцію для знищення списку.
7. Записати список у файл, знищити його й виконати друк (при друці повинне бути видане повідомлення "Список порожній").
8. Написати функцію для відновлення списку з файлу.
9. Відновити список і роздрукувати його.
10. Знищити список.

4. Записи в лінійному списку містять ключове поле типу int. Сформувати однонаправлений список. Знищити з нього елемент із заданим номером, додати К елементів, починаючи із заданого номера;

Завдання №4 **Робота з різними типами структурами**

Зв’язаний список

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.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Перевертання пов’язаного списку вимагає логічного підходу до маніпулювання покажчиками, що є ключовим навиком у інформатиці.

Засвоїти механізми маніпуляції з покажчиками: пов’язані списки значною мірою залежать від покажчиків. Це завдання покращить навички маніпулювання вказівниками, що є ключовим аспектом у таких мовах, як C++.

Розвинути навички розв’язувати задачі: перевернути пов’язаний список непросто й вимагає творчого й логічного мислення, таким чином покращуючи свої навички розв’язування поставлених задач.

Пояснення прикладу

Спочатку ми визначаємо просту структуру ***Node*** для нашого пов’язаного списку.
Потім функція ***reverse*** ітеративно змінює список, маніпулюючи наступними покажчиками кожного вузла.
printList — допоміжна функція для відображення списку.
Основна функція створює зразок списку, демонструє реверсування та друкує вихідний і обернений списки.

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.

Поглиблення розуміння зв'язаних списків: Порівнюючи зв'язані списки, дозволяють покращити своє розуміння обходу, фундаментальної операції в обробці зв'язаних списків.

Розуміння ефективність алгоритму: це завдання також вводить поняття ефективності алгоритму. Студенти вчаться ефективно порівнювати елементи, що є навичкою, важливою для оптимізації та зменшення складності обчислень.

Розвинути базові навики роботи з реальними програми: функції порівняння мають вирішальне значення в багатьох реальних програмах, таких як виявлення змін у даних, синхронізація структур даних або навіть у таких алгоритмах, як сортування та пошук.

Розвинути навик вирішення проблем і увага до деталей: це завдання заохочує скрупульозний підхід до програмування, оскільки навіть найменша неуважність може призвести до неправильних результатів порівняння. Це покращує навички вирішення проблем і увагу до деталей.

Пояснення прикладу

- Для пов'язаного списку визначено структуру ***Node***.
- Функція ***compare*** ітеративно проходить обидва списки одночасно, порівнюючи дані в кожному вузлі.
- Якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.
- Основна функція ***main*** створює два списки та демонструє порівняння.

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 $\Longrightarrow$ $9 \rightarrow 7 \rightarrow 3$);
- 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.

Поглиблення розуміння зв'язаних списків: Застосовування зв'язаних списків для арифметичних операції з великими числами дозволяє покращити розуміння операцій з обробки зв'язаних списків.

Розуміння ефективність алгоритму: це завдання дозволяє порівняти швидкість алгоритму додавання з використанням списків зі швидкістю вбудованих арифметичних операцій. Студенти вчаться розрізняти позитивні та негативні ефекти при виборі структур даних для реалізації практичних програм.

Розвинути базові навики роботи з реальними програми: арифметичні операції з великими числами використовуються у криптографії, теорії чисел, астрономії, та ін.

Розвинути навик вирішення проблем і увага до деталей: завдання покращує розуміння обмежень у представленні цілого числа сучасними комп'ютерами та пропонує спосіб його вирішення.

Бінарні дерева

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.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева продемонструє розгортання рекурсивного виклику.

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.

Розвиток алгоритмічне мислення: Це завдання розвиває алгоритмічне мислення. Прохід всіх вузлів дерева демонструє розгортання рекурсивного виклику.

2. Код програм

Завдання №1

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

using namespace std;

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

    // Створення матриці печери
    vector<vector<char>> cave(N, vector<char>(M));

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

    // Обробка кожного стовпця окремо
    for (int j = 0; j < M; ++j) {
        int sandPos = N - 1; // Початкова позиція для піску, найнижча клітинка
        // Проходимо знизу вгору
        for (int i = N - 1; i >= 0; --i) {
            if (cave[i][j] == 'S') {  // Знайшли пісок
                // Переміщаємо пісок вниз до першої вільної клітинки
                cave[i][j] = 'O';
                cave[sandPos][j] = 'S';
                --sandPos;  // Знижуємо позицію для наступного піску
            } else if (cave[i][j] == 'X') {
                // Якщо зустрічаємо камінь, зупиняємо падіння піску
                sandPos = i - 1;
            }
        }
    }

```

```
37      // Виведення результату
38      for (int i = 0; i < N; ++i) {
39          for (int j = 0; j < M; ++j) {
40              cout << cave[i][j];
41          }
42          cout << endl;
43      }
44
45      return 0;
46  }
47
```

Посилання на pull-request:
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/856/files#diff-d646b2569ac0ec70707b5c30bfd9d0cc49ddea95b31b0798aa6f215458ed7251

Завдання №2

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

using namespace std;

// Шаблонний клас вузла дерева
template <typename T>
struct TreeNode {
    T value;
    TreeNode* left;
    TreeNode* right;

    TreeNode(T val) : value(val), left(nullptr), right(nullptr) {}
};

// Шаблонний клас двійкового дерева пошуку
template <typename T>
class BinarySearchTree {
private:
    TreeNode<T>* root;
    int treeSize;

    // Рекурсивна функція вставки
    TreeNode<T>* insert(TreeNode<T>* node, T value) {
        if (!node) {
            treeSize++;
            return new TreeNode<T>(value);
        }
        if (value < node->value) {
            node->left = insert(node->left, value);
        } else if (value > node->value) {
            node->right = insert(node->right, value);
        }
        return node;
    }

    // Рекурсивна функція пошуку
    bool contains(TreeNode<T>* node, T value) const {
        if (!node) return false;
        if (value == node->value) return true;
        if (value < node->value) return contains(node->left, value);
        return contains(node->right, value);
    }

    // Рекурсивна функція для інфіксного обходу
    void inorder(TreeNode<T>* node, ostream& os) const {
        if (!node) return;
        inorder(node->left, os);
        os << node->value << " ";
        inorder(node->right, os);
    }

```

```
    // Рекурсивне видалення дерева
    void clear(TreeNode<T>* node) {
        if (!node) return;
        clear(node->left);
        clear(node->right);
        delete node;
    }

public:
    // Конструктор
    BinarySearchTree() : root(nullptr), treeSize(0) {}

    // Деструктор
    ~BinarySearchTree() {
        clear(root);
    }

    // Функція вставки
    void insert(T value) {
        root = insert(root, value);
    }

    // Функція пошуку
    bool contains(T value) const {
        return contains(root, value);
    }

    // Функція визначення розміру
    int size() const {
        return treeSize;
    }

    // Перевантаження оператора виведення
    friend ostream& operator<<(ostream& os, const BinarySearchTree& tree) {
        tree.inorder(tree.root, os);
        return os;
    }
};
```

```cpp
int main() {
    int Q;
    cin >> Q;

    BinarySearchTree<int> bst;

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

        if (command == "insert") {
            int value;
            cin >> value;
            bst.insert(value);
        } else if (command == "contains") {
            int value;
            cin >> value;
            if (bst.contains(value)) {
                cout << "Yes" << endl;
            } else {
                cout << "No" << endl;
            }
        } else if (command == "size") {
            cout << bst.size() << endl;
        } else if (command == "print") {
            cout << bst << endl;
        }
    }

    return 0;
}
```

Посилання на pull-request:
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/856/files#diff-b27b1a7faac4b91b631b06b258e23844e88ca83553f75b4bda5bf60b6ddcb49a

Завдання №3

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

using namespace std;

// Структура елемента списку
struct Node {
    int key;        // Ключове поле
    Node* next;     // Вказівник на наступний елемент
};

// Функція для створення порожнього списку
Node* createList() {
    return nullptr;  // Порожній список
}

// Функція для друку списку
void printList(Node* head) {
    if (head == nullptr) {
        cout << "Список порожній." << endl;
        return;
    }

    Node* temp = head;
    while (temp != nullptr) {
        cout << temp->key << " ";
        temp = temp->next;
    }
    cout << endl;
}

// Функція для додавання елемента в список
void addNode(Node*& head, int value) {
    Node* newNode = new Node();
    newNode->key = value;
    newNode->next = head;
    head = newNode;
}
```

```cpp
// Функція для знищення елемента за номером
void deleteNode(Node*& head, int position) {
    if (head == nullptr) return;

    Node* temp = head;

    // Якщо видаляється перший елемент
    if (position == 1) {
        head = temp->next;
        delete temp;
        return;
    }

    // Знайдемо попередній елемент
    for (int i = 1; temp != nullptr && i < position - 1; i++) {
        temp = temp->next;
    }

    // Якщо позиція неправильна
    if (temp == nullptr || temp->next == nullptr) return;

    Node* next = temp->next->next;
    delete temp->next;
    temp->next = next;
}

// Функція для запису списку в файл
void saveListToFile(Node* head, const string& filename) {
    ofstream file(filename);

    if (file.is_open()) {
        Node* temp = head;
        while (temp != nullptr) {
            file << temp->key << " ";
            temp = temp->next;
        }
        file.close();
    }
    else {
        cout << "Не вдалося відкрити файл для запису." << endl;
    }
}

// Функція для знищення списку
void deleteList(Node*& head) {
    Node* current = head;
    Node* nextNode;

    while (current != nullptr) {
        nextNode = current->next;
        delete current;
        current = nextNode;
    }

    head = nullptr;
}
```

```cpp
// Функція для відновлення списку з файлу
void restoreListFromFile(Node*& head, const string& filename) {
    ifstream file(filename);

    if (file.is_open()) {
        int value;
        while (file >> value) {
            addNode(head, value);
        }
        file.close();
    } else {
        cout << "Не вдалося відкрити файл для зчитування." << endl;
    }
}

int main() {
    Node* list = createList();

    // Додавання елементів
    addNode(list, 10);
    addNode(list, 20);
    addNode(list, 30);

    cout << "Список після додавання елементів: ";
    printList(list);

    // Видалення елемента з позиції 2
    deleteNode(list, 2);
    cout << "Список після видалення елемента на позиції 2: ";
    printList(list);

    // Запис списку в файл
    saveListToFile(list, "list.txt");

    // Знищення списку
    deleteList(list);
    cout << "Після знищення списку: ";
    printList(list);

    // Відновлення списку з файлу
    restoreListFromFile(list, "list.txt");
    cout << "Список після відновлення з файлу: ";
    printList(list);

    // Знищення списку
    deleteList(list);
    return 0;
}

```

Посилання на pull-request:
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/856/files#diff-5d94f28e7eac5e9a642af83e4efc23e98b1736bc84c4ebfe6e2755af4671dab9

Завдання №4_1

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

// Структура вузла
struct Node {
    int data;
    Node* next;
};

// Функція для створення нового вузла
Node* createNode(int data) {
    return new Node{data, nullptr};
}

// Функція для створення списку з масиву
Node* createList(const int arr[], int size) {
    if (size == 0) return nullptr;
    Node* head = createNode(arr[0]);
    Node* current = head;
    for (int i = 1; i < size; ++i) {
        current->next = createNode(arr[i]);
        current = current->next;
    }
    return head;
}

// Функція для друку списку
void printList(Node* head, bool reverse = false) {
    if (reverse) {
        if (!head) return;
        printList(head->next, true);
        cout << head->data;
    } else {
        Node* current = head;
        while (current) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
}

// Реверс списку
Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* current = head;
    while (current) {
        Node* next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}
```

```cpp
// Порівняння двох списків
bool compare(Node* h1, Node* h2) {
    while (h1 && h2) {
        if (h1->data != h2->data) return false;
        h1 = h1->next;
        h2 = h2->next;
    }
    return !h1 && !h2;
}

// Додавання великих чисел
Node* add(Node* n1, Node* n2) {
    Node* result = nullptr;
    Node* tail = nullptr;
    int carry = 0;

    while (n1 || n2 || carry) {
        int sum = carry + (n1 ? n1->data : 0) + (n2 ? n2->data : 0);
        carry = sum / 10;
        sum %= 10;

        Node* newNode = createNode(sum);
        if (!result) {
            result = tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }

        if (n1) n1 = n1->next;
        if (n2) n2 = n2->next;
    }

    return result;
}
```

```cpp
int main() {
    // Створення списку з масиву
    int arr1[] = {1, 2, 3, 4, 5};
    Node* list1 = createList(arr1, 5);

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

    // Реверс списку
    Node* reversedList = reverse(list1);
    cout << "Reversed list: ";
    printList(reversedList);

    // Порівняння списків
    cout << "Lists are " << (compare(list1, reversedList) ? "equal." : "not equal.") << endl;

    // Додавання великих чисел
    int num1[] = {9, 9, 9};
    int num2[] = {1};
    Node* number1 = createList(num1, 3);
    Node* number2 = createList(num2, 1);

    Node* sum = add(number1, number2);
    cout << "Sum of numbers: ";
    printList(sum, true);  // Виводимо в прямому порядку
    cout << endl;

    return 0;
}
```

Посилання на pull-request:
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/856/files#diff-ac5d1f8cf43bcf774b9be07707349d19a1746ebc64411040a81cb2c5d05e31db

Завдання №4_2

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

// Структура вузла дерева
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};

// Функція для створення нового вузла
TreeNode* createNode(int data) {
    return new TreeNode(data);
}

// Функція для друку дерева (обхід у порядку)
void printTree(TreeNode* root) {
    if (!root) return;
    printTree(root->left);
    cout << root->data << " ";
    printTree(root->right);
}

// Задача №4: Віддзеркалення дерева
TreeNode* create_mirror_flip(TreeNode* root) {
    if (!root) return nullptr;

    // Створюємо новий вузол з тим самим значенням
    TreeNode* newRoot = createNode(root->data);

    // Міняємо місцями ліву і праву вітки рекурсивно
    newRoot->left = create_mirror_flip(root->right);
    newRoot->right = create_mirror_flip(root->left);

    return newRoot;
}

// Задача №5: Записати кожному батьківському вузлу суму підвузлів
int tree_sum(TreeNode* root) {
    if (!root) return 0;

    // Якщо це листок, його значення не змінюється
    if (!root->left && !root->right) return root->data;

    // Рекурсивно обчислюємо суму підвузлів
    int leftSum = tree_sum(root->left);
    int rightSum = tree_sum(root->right);

    // Записуємо суму підвузлів у вузол
    root->data = leftSum + rightSum;

    return root->data;
}
```

```cpp
int main() {
    // Створення дерева
    TreeNode* root = createNode(1);
    root->left = createNode(2);
    root->right = createNode(3);
    root->left->left = createNode(4);
    root->left->right = createNode(5);
    root->right->left = createNode(6);
    root->right->right = createNode(7);

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

    // Задача №4: Віддзеркалення дерева
    TreeNode* mirroredTree = create_mirror_flip(root);
    cout << "Mirrored tree (in-order): ";
    printTree(mirroredTree);
    cout << endl;

    // Задача №5: Сума підвузлів
    tree_sum(root);
    cout << "Tree with summed nodes (in-order): ";
    printTree(root);
    cout << endl;

    return 0;
}

```

Посилання на pull-request:
https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/856/files#diff-a4a8f761d55cf3d8e22a9f905c9e52ac80e08022996dad92a83bdc084d2445c0

4. Результати виконання завдань, тестування та фактично затрачений час:

Завдання №1

```
5 5
SSOSS
OOOOO
SOOXX
OOOOS
OOSOO
OOOOO
OOOSS
OOOXX
SOOOO
SSSOS
```

Час затрачений на виконання завдання: 1 год

Завдання №2

```
11
size
0
insert5
insert 5
insert 4
print
4 5
insert 1
print
1 4 5
contains 5
Yes
contains 0
No
size
3
```

Час затрачений на виконання завдання: 4 год

Завдання №3

```
Список після додавання елементів: 30 20 10
Список після видалення елемента на позиції 2: 30 10
Після знищення списку: Список порожній.
Список після відновлення з файлу: 10 30
```

```
ai_11 > arsenii_zubrytskyi > epic_6 > list.txt
  1    30 10
```

Час затрачений на виконання завдання: 2 год

Завдання №4_1

```
Original list: 1 2 3 4 5
Reversed list: 5 4 3 2 1
Lists are not equal.
Sum of numbers: 1000
```

Час затрачений на виконання завдання: 3 год

Завдання №4_1

```
Original tree (in-order): 4 2 5 1 6 3 7
Mirrored tree (in-order): 7 3 6 1 5 2 4
Tree with summed nodes (in-order): 4 9 5 22 6 13 7
```

Час затрачений на виконання завдання: 3 год

Висновки: В ході виконання робіт з epic_6 я опрацював динамічні структури данних такі як черга, стек, зв’язні списки, дерева. На практиці реалізував основні принципи роботи з ними обхід, пошук додавання елементів та видалення.