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
Цяпа Остап Андрійович

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 Microsoft Copilot 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 9)

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 int. Сформувати двонаправлений список. Знищити з нього K елементів перед елементом із заданим номером, додати K елементів у кінець списку.

Завдання №2

ALGOTESTER LAB 5 (VARIANT 2)

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 - i-й рядок матриці, яка відображає стан печери до землетрусу.

Вихідні дані

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

Завдання №3

ALGOTESTER 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 Y es, 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 №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

Зеник та Марічка люблять шукати халяву. Тож ось вона.

У Зеника є n синіх кульок, на i-ій кульці записане число a_i. Зенику цікаво, скількома способами він може пофарбувати деякі кульки в жовтий колір так, щоб усі числа, записані на жовтих кульках, були строго меншими за числа на синіх кульках.

Зауважте, що Зеник повинен пофарбувати хоча б одну кульку в жовтий колір, також він може пофарбувати всі кульки.

Вхідні дані

У першому рядку задано одне ціле число n — кількість кульок.
У другому рядку задано n цілих чисел a_i — числа, записані на кульках.

Вихідні дані

У єдиному рядку виведіть одне ціле число — скількома способами він може пофарбувати деякі кульки в жовтий колір.

Завдання №6
SELF PRACTICE WORK ALGOTESTER

Зеник та Марічка грають у поле чудес. Спочатку Зеник пише на дошці загадане слово й закриває всі його букви. За один хід Марічка називає букву, а Зеник відкриває всі такі букви у слові.

Вам необхідно визначити, за яку мінімальну кількість ходів Марічка зможе відкрити всі букви у слові.

Наприклад, якщо Зеник загадав слово `MAMA`, то Марічка зможе його відкрити за два ходи, назвавши букви `M` та `A`.

Вхідні дані

Вхідні дані містять рядок s — загадане Зеником слово.

Вихідні дані

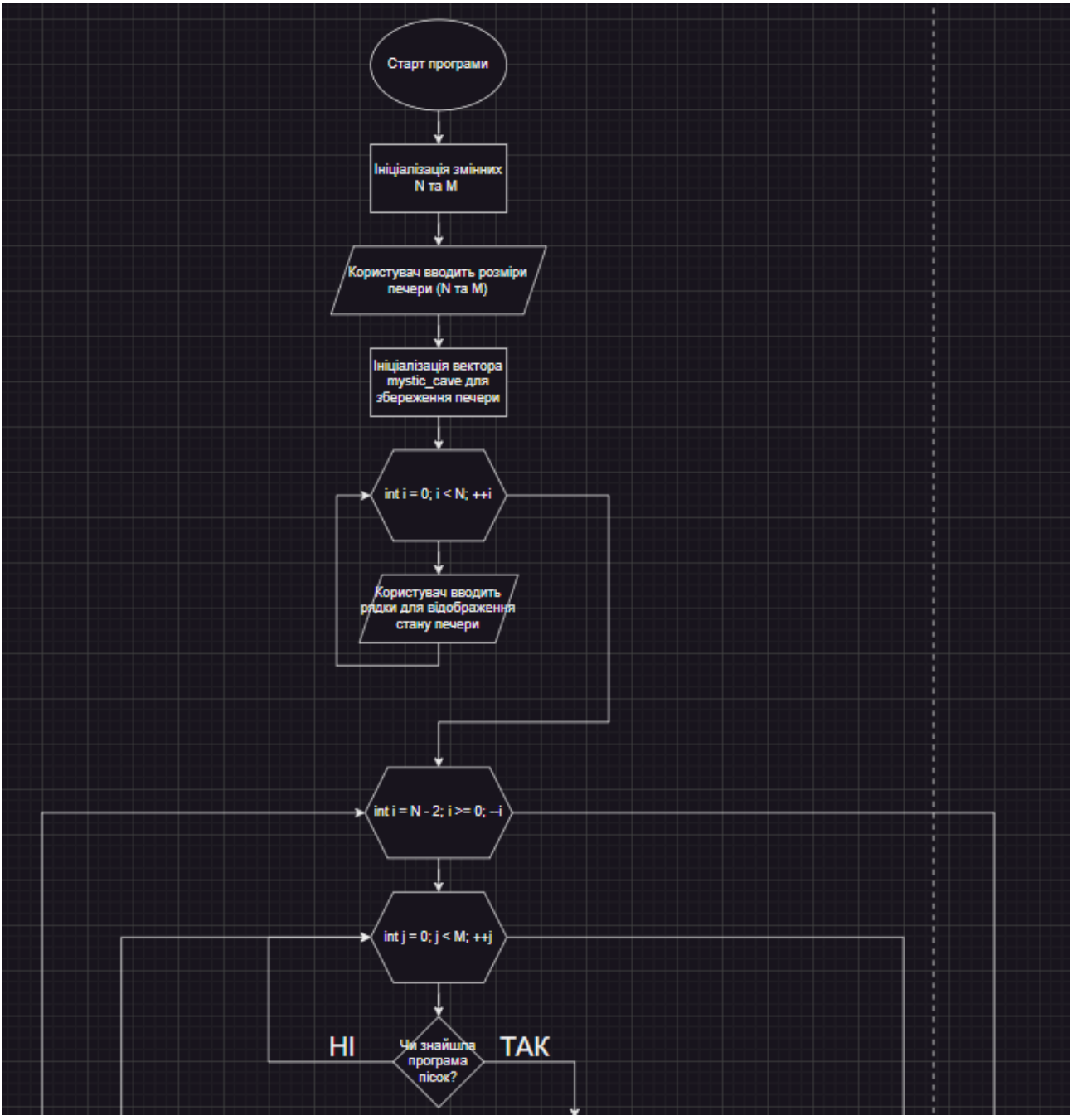
В одному рядку виведіть ціле число — мінімальну кількість ходів.

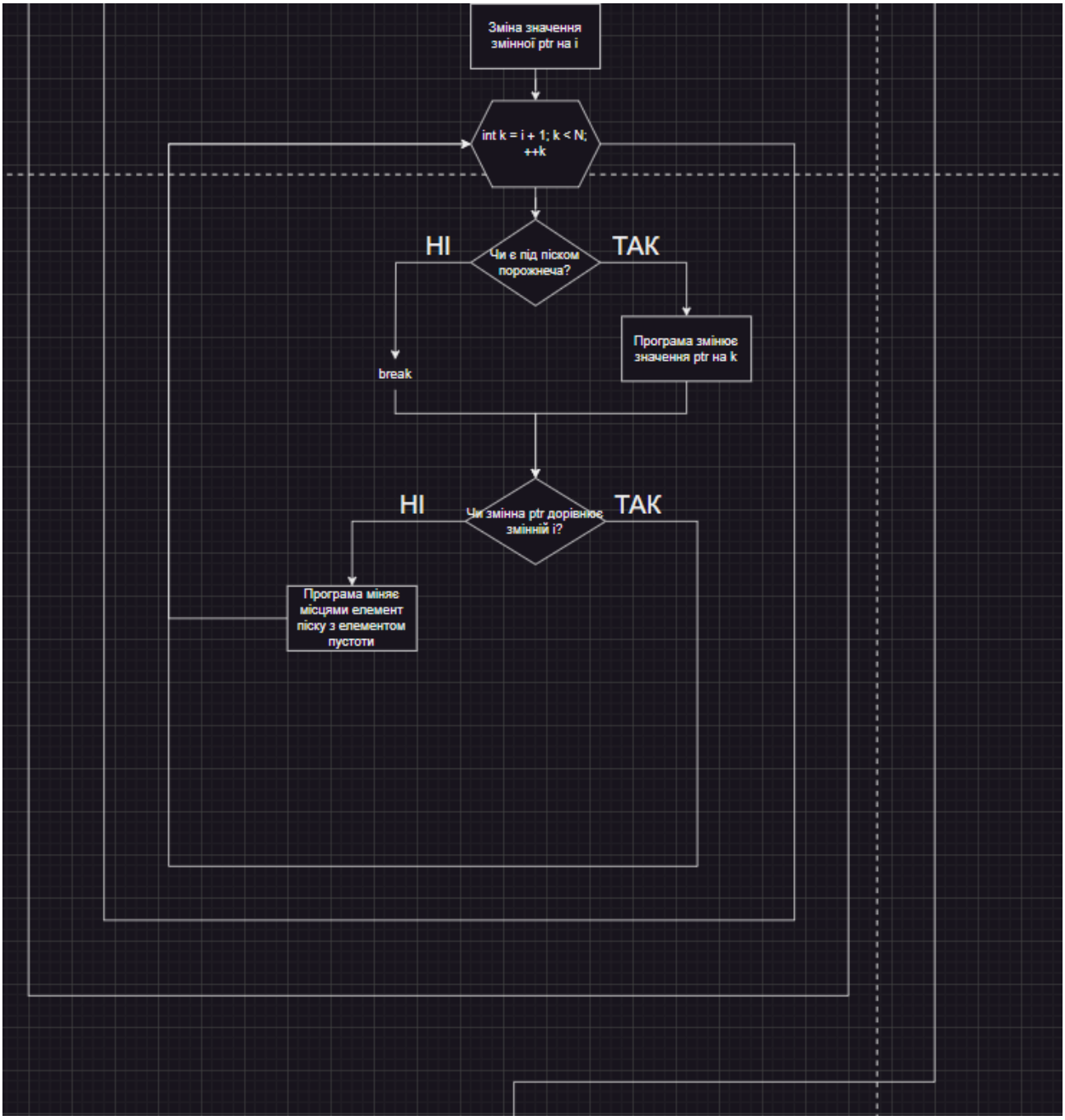
2. Вимоги та планувальна оцінка часу виконання завдань:
Програма №1

- 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 біблотеку fstream для запису даних у файл.
- Плановий час на реалізацію 3 години.

Програма №2

- Важливі деталі для реалізації програми.
- Блок – схема


Старт програми
Ініціалізація змінних N та M
Користувач вводить розміри печери (N та M)
Ініціалізація вектора mystic_cave для збереження печери
int i = 0; i < N; ++i
Користувач вводить рядки для відображення стану печери
int i = N - 2; i >= 0; --i
int j = 0; j < M; ++j
НІ
Чи знайшла програма пісок?
ТАК


Зміна значення змінної ptr на i
int k = i + 1; k < N; ++k
НІ
Чи є під піском порожнеча?
ТАК
Програма змінює значення ptr на k
break
НІ
Чи змінна ptr дорівнює змінній i?
ТАК
Програма міняє місцями елемент піску з елементом пустоти

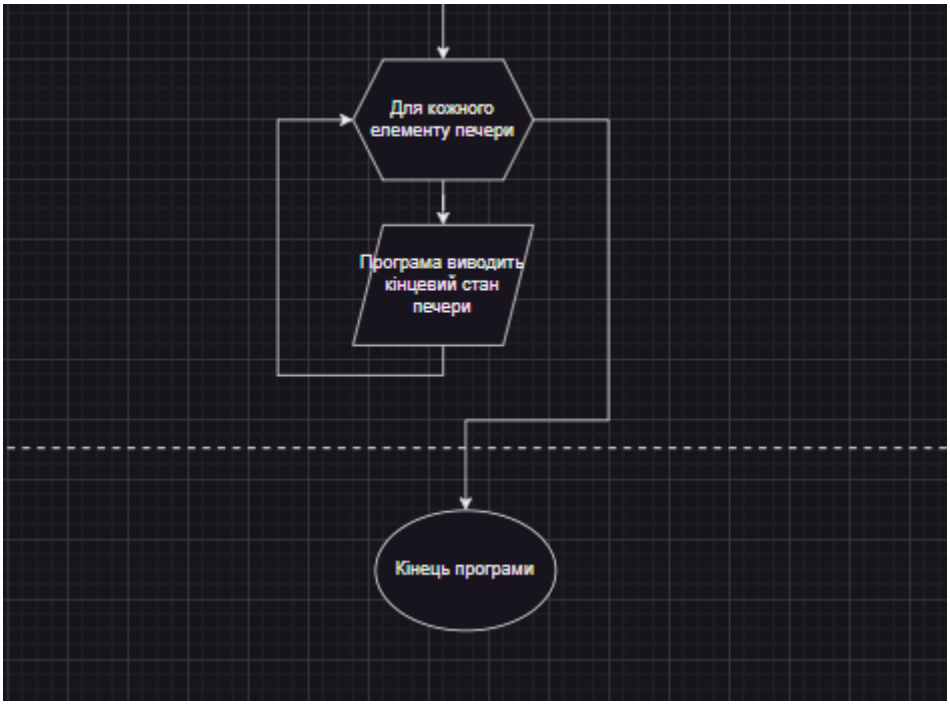


Рисунок 2.1. Блок – схема до програми 2

- Використовувати для реалізації матрицю чарів(вектор у векторі) а також цикли, для того щоб проходитися по стовпцях та рядках матриці.
- Плановий час на реалізацію 2 години.

Програма №3

- Важливі деталі для реалізації програми.
- Зрозуміти, що таке Linked List (Однозв’язний список) 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 реалізовувати.
- Плановий час на реалізацію 5 годин.

Програма №5

- Важливі деталі для реалізації програми.
- Використовувати для реалізації бібліотеку set для швидкого вирішення.
- Плановий час на реалізацію 2 години.

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

Завдання №1

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

// Структура для двонаправленого списку
struct Node {
    int data;          // Значення вузла
    Node* next;        // Вказівник на наступний вузол
    Node* prev;        // Вказівник на попередній вузол
    Node(int val) : data(val), next(nullptr), prev(nullptr) {} // Конструктор для ініціалізації вузла
};

class DoublyLinkedList {
private:
    Node* head;  // Вказівник на голову списку
    Node* tail;  // Вказівник на хвіст списку

public:
    // Конструктор для ініціалізації порожнього списку
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    // Деструктор для очищення списку при знищенні об'єкта
    ~DoublyLinkedList() {
        clear();
    }

    // 1. Створення порожнього списку
    void createList() {
        head = nullptr;
        tail = nullptr;
        cout << "Список створено" << endl;
    }

    // 2. Друк списку
    void printList() const {
        if (head == nullptr) {
            cout << "Список порожній" << endl;
            return;
        }
        Node* temp = head;
        while (temp != nullptr) {
            cout << temp->data << " ";
            temp = temp->next;
        }
        cout << endl;
    }

    // 3. Додавання елемента у список
    void addAtEnd(int data) {
        Node* newNode = new Node(data);  // Створення нового вузла
        if (tail == nullptr) {
            head = tail = newNode;  // Якщо список порожній, новий вузол стає головою і хвостом
        } else {
            tail->next = newNode;   // Додаємо новий вузол після хвоста
            newNode->prev = tail;   // Вказуємо попередній вузол для нового вузла
            tail = newNode;         // Новий вузол стає новим хвостом
        }
        cout << "Додано елемент " << data << " в кінець списку" << endl;
    }

    // 3. Знищення елемента зі списку
    void deleteAtPosition(int position) {
        if (head == nullptr) return; // Якщо список порожній, нічого не робимо
        Node* temp = head;
        for (int i = 0; temp != nullptr && i < position; i++) {
            temp = temp->next; // Переходимо до вузла на позиції
        }
```

```
        if (temp == nullptr) return; // Якщо вузла на такій позиції не існує, нічого не робимо
        if (temp->prev != nullptr) temp->prev->next = temp->next; // Якщо це не голова списку, змінюємо вказівник попереднього вузла
        if (temp->next != nullptr) temp->next->prev = temp->prev; // Якщо це не хвіст списку, змінюємо вказівник наступного вузла
        if (temp == head) head = temp->next; // Якщо це голова списку, змінюємо голову
        if (temp == tail) tail = temp->prev; // Якщо це хвіст списку, змінюємо хвіст
        cout << "Видалено елемент на позиції " << position << endl;
        delete temp; // Видаляємо вузол
    }

    // 4. Здійснення змін у списку
    void modifyAndPrint(int deleteCount, int targetPosition, int addCount, int addValue) {
        for (int i = 0; i < deleteCount; ++i) {
            deleteAtPosition(targetPosition - deleteCount + i); // Знищуємо задану кількість елементів перед заданою позицією
        }
        printList(); // Друк списку після видалення
        for (int i = 0; i < addCount; ++i) {
            addAtEnd(addValue); // Додаємо задану кількість елементів у кінець списку
        }
        printList(); // Друк списку після додавання
    }

    // 5. Запис списку у файл
    void writeToFile(const string& filename) const {
        ofstream file(filename); // Відкриваємо файл для запису
        Node* temp = head;
        while (temp != nullptr) {
            file << temp->data << " "; // Записуємо дані у файл
            temp = temp->next;
        }
        file.close(); // Закриваємо файл
        cout << "Список записано у файл " << filename << endl;
    }

    // 6. Знищення списку
    void clear() {
        Node* current = head;
        while (current != nullptr) {
            Node* next = current->next;
            delete current; // Видаляємо поточний вузол
            current = next;
        }
        head = tail = nullptr; // Очищаємо вказівники на голову і хвіст
        cout << "Список очищено" << endl;
    }

    // 8. Відновлення списку з файлу
    void readFromFile(const string& filename) {
        clear(); // Очищаємо поточний список
        ifstream file(filename); // Відкриваємо файл для читання
        int data;
        while (file >> data) {
            addAtEnd(data); // Додаємо дані з файлу у кінець списку
        }
        file.close(); // Закриваємо файл
        cout << "Список відновлено з файлу " << filename << endl;
    }
};

int main() {
    DoublyLinkedList list;

    // 1. Створення списку
    list.createList();
    list.printList();
```

```
132
133     // Додавання елементів у список
134     list.addAtEnd(1);
135     list.addAtEnd(2);
136     list.addAtEnd(3);
137     list.addAtEnd(4);
138     list.printList();
139
140     // 4. Виконання змін у списку та друк після кожної зміни
141     list.modifyAndPrint(2, 2, 2, 5); // Знищити 2 елементи перед 2-м і додати 2 елементи зі значенням 5
142
143     // 5. Запис списку у файл
144     list.writeToFile("list.txt");
145
146     // 6. Знищення списку
147     list.clear();
148     list.printList(); // Має вивести "Список порожній"
149
150     // 8. Відновлення списку з файлу
151     list.readFromFile("list.txt");
152     list.printList();
153
154     // 10. Знищити список
155     list.clear();
156     list.printList(); // Має вивести "Список порожній"
157
158     return 0;
159 }
160
```

Рисунок 3.1. Код до програми № 1

```
Список створено
Список порожній
Додано елемент 1 у кінець списку
Додано елемент 2 у кінець списку
Додано елемент 3 у кінець списку
Додано елемент 4 у кінець списку
1 2 3 4
Видалено елемент на позиції 0
Видалено елемент на позиції 1
2 4
Додано елемент 5 у кінець списку
Додано елемент 5 у кінець списку
2 4 5 5
Список записано у файл list.txt
Список очищено
Список порожній
Список очищено
Додано елемент 2 у кінець списку
Додано елемент 4 у кінець списку
Додано елемент 5 у кінець списку
Додано елемент 5 у кінець списку
Список відновлено з файлу list.txt
2 4 5 5
Список очищено
Список порожній
Список очищено
```

Рисунок 3.2. Приклад виконання програми № 1

Фактично затрачений час 4 години.

Завдання №2

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

int main() {
    int N, M;
    cin >> N >> M;  // Зчитування розмірів печери
    vector<string> mystic_cave(N);  // Створення вектора рядків для зберігання печери

    for (int i = 0; i < N; ++i) {
        cin >> mystic_cave[i];  // Зчитування стану печери
    }

    // Проходження печери знизу вгору, крім останнього рядка
    for (int i = N - 2; i >= 0; --i) {
        for (int j = 0; j < M; ++j) {
            if (mystic_cave[i][j] == 'S') {  // Якщо знайшли пісок
                int ptr = i;
                // Знаходження найнижчої позиції, куди може впасти пісок
                for (int k = i + 1; k < N; ++k) {
                    if (mystic_cave[k][j] == '0') {
                        ptr = k;
                    } else {
                        break;
                    }
                }
                if (ptr != i) {
                    // Заміна поточного положення піску на пустоту та опускання піску вниз
                    swap(mystic_cave[i][j], mystic_cave[ptr][j]);
                }
            }
        }
    }

    cout << endl;
    // Виведення кінцевого стану печери
    for (const string& str : mystic_cave) {
        cout << str << endl;
    }

    return 0;
}
```

Рисунок 3.3. Код до програми № 2

```
5 5
0S50S
XX000
X000X
00000
XXX00

05000
XX00S
X000X
00S00
XXX00
```

Рисунок 3.4. Приклад виконання програми № 2

Рисунок 3.5. Статус задачі на алготестері

Фактично затрачений час 2 години.

Завдання №3

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

struct TreeNode {
    int value;
    TreeNode* left;
    TreeNode* right;

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

class BinaryTree {
private:
    TreeNode* root;
    size_t treeSize;

    TreeNode* insert(TreeNode* node, int value) {
        if (node == nullptr) {
            return new TreeNode(value);
        }
        if (value < node->value) {
            node->left = insert(node->left, value);
        } else if (value > node->value) {
            node->right = insert(node->right, value);
        }
        return node;
    }

    bool contains(TreeNode* node, int value) const {
        if (node == nullptr) {
            return false;
        }
        if (value == node->value) {
            return true;
        } else if (value < node->value) {
            return contains(node->left, value);
        } else {
            return contains(node->right, value);
        }
    }

    void inorder(TreeNode* node, ostream& os) const {
        if (node == nullptr) {
            return;
        }
        inorder(node->left, os);
        os << node->value << " ";
        inorder(node->right, os);
    }

public:
    BinaryTree() : root(nullptr), treeSize(0) {}

```

```
    void insert(int value) {
        if (!contains(root, value)) {
            root = insert(root, value);
            treeSize++;
        }
    }

    bool contains(int value) const {
        return contains(root, value);
    }

    size_t size() const {
        return treeSize;
    }

    friend ostream& operator<<(ostream& os, const BinaryTree& tree) {
        tree.inorder(tree.root, os);
        return os;
    }
};

int main() {
    BinaryTree tree;
    int Q;
    cin >> Q;

    vector<string> commands(Q);
    vector<int> values(Q, 0);

    for (int i = 0; i < Q; ++i) {
        cin >> commands[i];
        if (commands[i] == "insert" || commands[i] == "contains") {
            cin >> values[i];
        }
    }

    for (int i = 0; i < Q; ++i) {
        if (commands[i] == "insert") {
            tree.insert(values[i]);
        } else if (commands[i] == "contains") {
            cout << (tree.contains(values[i]) ? "Yes" : "No") << endl;
        } else if (commands[i] == "size") {
            cout << tree.size() << endl;
        } else if (commands[i] == "print") {
            cout << tree << endl;
        }
    }

    return 0;
}
```

Рисунок 3.6. Код до програми № 3

```
5
insert 5
insert 4
print
contains 5
size
4 5
Yes
2
```

Рисунок 3.7. Приклад виконання програми №3

Рисунок 3.8. Статус задачі на Algotester

Фактично затрачений час 4 години.

Завдання №4

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

// Визначення структури для зв'язаного списку
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

// Визначення структури для бінарного дерева
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int value) : val(value), left(nullptr), right(nullptr) {}
};

// Завдання №1 - Реверс списку
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;
}

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

// Завдання №2 - Порівняння списків
bool compare(Node* h1, Node* h2) {
    bool identical = true;
    while (h1 != nullptr && h2 != nullptr) {
        if (h1->data != h2->data) {
            cout << "Mismatch found: " << h1->data << " != " << h2->data << endl;
            identical = false;
        }
        h1 = h1->next;
        h2 = h2->next;
    }
    if (h1 != nullptr || h2 != nullptr) {
        cout << "Lists are of different lengths." << endl;
        identical = false;
    }
    return identical;
}

void printComparison(Node* h1, Node* h2) {
    printList(h1);
    printList(h2);
    if (compare(h1, h2)) {
        cout << "Lists are identical" << endl;
    } else {
        cout << "Lists are different" << endl;
    }
}

// Завдання №3 - Додавання великих чисел
Node* add(Node* n1, Node* n2) {
    Node* dummy = new Node(0);
    Node* p = dummy;
    int carry = 0;

    cout << "Adding numbers: " << endl;
    while (n1 != nullptr || n2 != nullptr || carry != 0) {
        int sum = carry;
        if (n1 != nullptr) {
            cout << n1->data << " ";
```

```
            sum += n1->data;
            n1 = n1->next;
        } else {
            cout << "0 ";
        }

        if (n2 != nullptr) {
            cout << "+ " << n2->data << " = ";
            sum += n2->data;
            n2 = n2->next;
        } else {
            cout << "+ 0 = ";
        }
        cout << sum % 10 << " (carry " << sum / 10 << ")" << endl;

        carry = sum / 10;
        p->next = new Node(sum % 10);
        p = p->next;
    }
    return dummy->next;
}

// Завдання №4 - Віддзеркалення дерева
TreeNode* create_mirror_flip(TreeNode* root) {
    if (root == nullptr) return nullptr;
    TreeNode* new_root = new TreeNode(root->val);
    new_root->left = create_mirror_flip(root->right);
    new_root->right = create_mirror_flip(root->left);
    return new_root;
}

// Завдання №5 - Записати кожному батьківському вузлу суму підвузлів
int tree_sum(TreeNode* root) {
    if (root == nullptr) return 0;
    if (root->left == nullptr && root->right == nullptr) return root->val;

    int left_sum = tree_sum(root->left);
    int right_sum = tree_sum(root->right);
    int original_val = root->val;
    root->val = left_sum + right_sum;

    return original_val + root->val;
}

void printTree(TreeNode* root, int level = 0) {
    if (root == nullptr) return;
    printTree(root->right, level + 1);
    for (int i = 0; i < level; ++i) cout << "    ";
    cout << root->val << endl;
    printTree(root->left, level + 1);
}

int main() {
    // Демонстрація реверсу списку
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);
    cout << "Original list: ";
    printList(head);
    head = reverse(head);
    cout << "Reversed list: ";
    printList(head);

    // Порівняння двох списків
    Node* list1 = new Node(1);
    list1->next = new Node(2);
    list1->next->next = new Node(3);
    Node* list2 = new Node(1);
    list2->next = new Node(2);
    list2->next->next = new Node(4);  // Місцева відмінність для демонстрації
    cout << "Comparison of lists:" << endl;
    printComparison(list1, list2);

    // Додавання великих чисел
    Node* num1 = new Node(2);
    num1->next = new Node(4);
```

```
159     num1->next->next = new Node(3);
160     Node* num2 = new Node(5);
161     num2->next = new Node(6);
162     num2->next->next = new Node(4);
163     cout << "Numbers being added: ";
164     printList(num1);
165     printList(num2);
166     Node* sum = add(num1, num2);
167     cout << "Sum: ";
168     printList(sum);
169
170     // Віддзеркалення бінарного дерева
171     TreeNode* root = new TreeNode(1);
172     root->left = new TreeNode(2);
173     root->right = new TreeNode(3);
174     root->left->left = new TreeNode(4);
175     root->left->right = new TreeNode(5);
176     root->right->left = new TreeNode(6);
177     root->right->right = new TreeNode(7);
178     cout << "Original tree:" << endl;
179     printTree(root);
180     TreeNode* mirrored_root = create_mirror_flip(root);
181     cout << "Mirrored tree:" << endl;
182     printTree(mirrored_root);
183
184     // Записати суму підвузлів
185     tree_sum(root);
186     cout << "Tree with sum of subnodes:" << endl;
187     printTree(root);
188
189     return 0;
190 }
191
```

Рисунок 3.9. Код до завдання номер 4

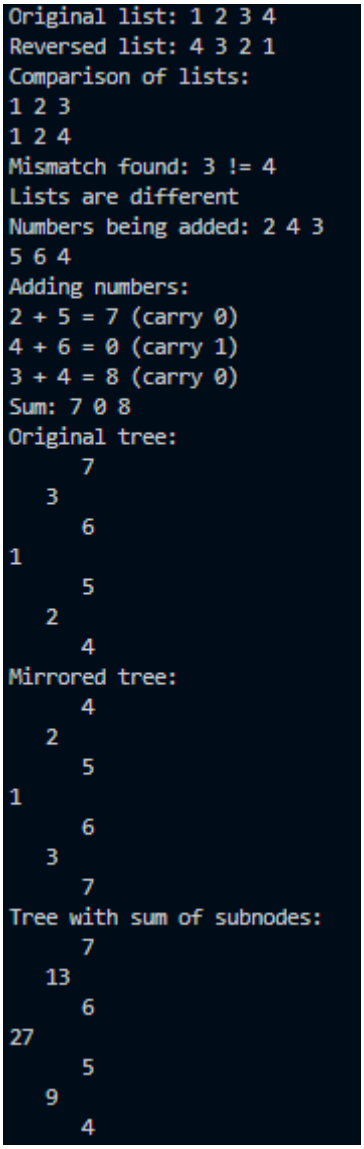


Рисунок 3.10. Приклад виконання завдання номер 4

Фактично затрачений час 5 годин.

Завдання №5

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

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

    vector<int> a(n);
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }

    set<int> uniqueNumbers(a.begin(), a.end());

    cout << uniqueNumbers.size() << endl;

    return 0;
}
```

Рисунок 3.19. Код до програми №5

```
5
2 4 4 2 2
2
```

Рисунок 3.20. Приклад виконання програми номер 5

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a few seconds ago	C++ 23	Accepted	0.068	2.328	View

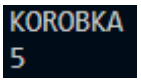
Рисунок 3.21. Статус задачі на Algotester

Фактично затрачений час 20 хвилин.

Завдання №6

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

int main() {
    string s;
    cin >> s;

    set<char> uniqueLetters(s.begin(), s.end());

    cout << uniqueLetters.size() << endl;

    return 0;
}
```

Рисунок 3.22. Код до програми №6



```
KOROBKA
5
```

Рисунок 3.23. Приклад виконання програми номер 5

декілька секунд тому	C++ 23	Зараховано	0.002	1.191	Перегляд

Рисунок 3.24. Статус задачі на Algotester

Фактично затрачений час 15 хвилин.

Посилання на пул реквест: [Epic 6 - Ostap Tsiapa by Ostap2007ter · Pull Request #669 · artificial-intelligence-department/ai_programming_playground_2024]

Висновок: У рамках практичних і лабораторних робіт блоку №6 я засвоїв безліч нового матеріалу, включаючи динамічні структури даних, такі як черги, стеки, списки та дерева, а також алгоритми для їх обробки. На практиці я працював із бінарними деревами, створював зв'язані списки та застосовував алгоритм BFS для розв'язання задач.