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

Виконала:

Студентка групи ШІ-13

Ходацька Аліна Віталіївна

Львів 2024

Тема роботи:

Динамічні структури. Алгоритми обробки динамічних структур.

Мета роботи:

Навчитись працювати з динамічними структурами даних, такими як списки, черги, стеки та дерева, а також удосконалити навички застосування алгоритмів їх ефективної обробки для розв’язання практичних задач.

Теоретичні відомості:

- динамічні структури
- стек
- черга
- зв’язні списки
- дерева
- алгоритми обробки динамічних структур

Використані джерела:

- https://acode.com.ua/urok-90-dynamichni-masyvy/
- https://acode.com.ua/urok-111-stek-i-kupa/
- https://www.youtube.com/watch?v=0BE0yxidMdE

Виконання роботи

Завдання №1 VNS Lab 10 Task 1 Variant 12

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.

Завдання

Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити з нього елементи, з однаковими ключовими полями. Додати елемент після елемента із заданим ключовим полем.

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

struct Node { // Структура для представлення елемента списку
    char* data;
    Node* next;
    Node* prev;

    // Конструктор
    Node(const char* str) {
        data = new char[strlen(str) + 1];
        strcpy_s(data, strlen(str) + 1, str);
        next = prev = nullptr;
    }

    // Деструктор
    ~Node() {
        delete[] data;
    }
};

class DoublyLinkedList { // Клас для представлення двоспрямованого списку
private:
    Node* head;
    Node* tail;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    ~DoublyLinkedList() {
        clear();
    }

    void add(const char* data) { // Додавання елемента у кінець списку
        Node* newNode = new Node(data);
        if (!head) {
            head = tail = newNode;
        }
        else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void remove(const char* key) { // Видалення елемента зі списку
        Node* current = head;
        while (current) {
            if (strcmp(current->data, key) == 0) {
                if (current->prev)
                    current->prev->next = current->next;
                if (current->next)
                    current->next->prev = current->prev;
                if (current == head)
                    head = current->next;
                if (current == tail)
                    tail = current->prev;
                Node* toDelete = current;
                current = current->next;
                delete toDelete;
            }
            else {
                current = current->next;
            }
        }
    }

    void addAfter(const char* key, const char* data) { // Додавання елемента після заданого ключа
        Node* current = head;
        while (current) {
            if (strcmp(current->data, key) == 0) {
                Node* newNode = new Node(data);
                newNode->next = current->next;
                newNode->prev = current;
                if (current->next)
                    current->next->prev = newNode;
                current->next = newNode;
                if (current == tail)
                    tail = newNode;
                break;
            }
            current = current->next;
        }
    }

    void print() const { // Друк списку
        if (!head) {
            std::cout << "List is empty\n";
            return;
        }
        Node* current = head;
        while (current) {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << "\n";
    }

    void saveToFile(const char* filename) const { // Запис списку у файл
        std::ofstream outFile(filename);
        Node* current = head;
        while (current) {
            outFile << current->data << "\n";
            current = current->next;
        }
        outFile.close();
    }

    void loadFromFile(const char* filename) { // Відновлення списку з файлу
        clear();
        std::ifstream inFile(filename);
        char buffer[256];
        while (inFile.getline(buffer, 256)) {
            add(buffer);
        }
        inFile.close();
    }

    void clear() { // Знищення списку
        Node* current = head;
        while (current) {
            Node* toDelete = current;
            current = current->next;
            delete toDelete;
        }
        head = tail = nullptr;
    }
};
```

```cpp
int main() {
    DoublyLinkedList list;

    // Додавання елементів у список
    list.add("apple");
    list.add("banana");
    list.add("cherry");

    // Друк списку
    std::cout << "List after adding items: ";
    list.print();

    // Додавання елемента після заданого ключа
    list.addAfter("banana", "blueberry");

    std::cout << "List after adding 'blueberry' after 'banana': ";
    list.print();

    // Видалення елемента зі списку
    list.remove("banana");

    std::cout << "List after 'banana' removal: ";
    list.print();

    // Запис списку у файл
    list.saveToFile("list.txt");

    // Знищення списку
    list.clear();
    std::cout << "List after destruction: ";
    list.print();

    // Відновлення списку з файлу
    list.loadFromFile("list.txt");

    std::cout << "List after recovery from file: ";
    list.print();

    // Знищення списку
    list.clear();
    std::cout << "List after destruction: ";
    list.print();

    return 0;
}
```

Microsoft Visual Studio Debu

```
List after adding items: apple banana cherry
List after adding 'blueberry' after 'banana': apple banana blueberry cherry
List after 'banana' removal: apple blueberry cherry
List after destruction: List is empty
List after recovery from file: apple blueberry cherry
List after destruction: List is empty
```

Завдання №2 Algotester Lab 5 Variant 3

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.

Клітинці 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.

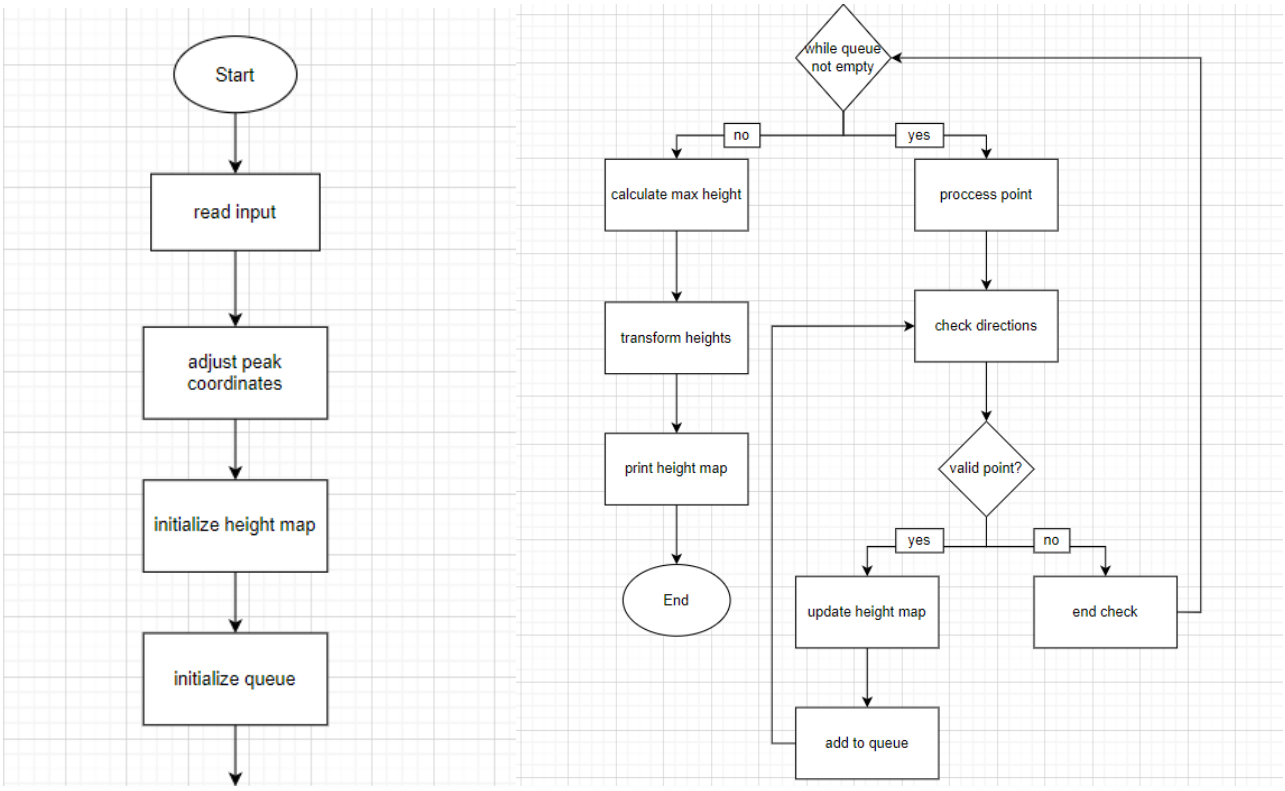
Input

У першому рядку 2 числа N та M - розміри карти

у другому рядку 2 числа x та y - координати піку гори

Output

N рядків по M елементів в рядку через пробіл - висоти карти.



```cpp
#include <iostream>
#include <vector>
#include <deque> // Для реалізації черги
#include <algorithm>

using namespace std;

struct Point {
    int x, y;
    int height;
};

// Заповнення висотної карти
void fillHeightMap(vector<vector<int>>& heightMap, int N, int M, int peakX, int peakY) {
    deque<Point> dq; // Черга для обходу в ширину
    dq.push_back({ peakX, peakY, 0 }); // Початкова висота - 0 для спрощення
    heightMap[peakX][peakY] = 0;

    // Напрямки для переміщення (вправо, вліво, вниз, вгору)
    int directions[4][2] = { {0, 1}, {0, -1}, {1, 0}, {-1, 0} };

    while (!dq.empty()) {
        Point p = dq.front();
        dq.pop_front();

        for (auto& dir : directions) {
            int newX = p.x + dir[0];
            int newY = p.y + dir[1];

            if (newX >= 0 && newX < N && newY >= 0 && newY < M && heightMap[newX][newY] == -1) {
                heightMap[newX][newY] = p.height + 1;
                dq.push_back({ newX, newY, p.height + 1 });
            }
        }
    }

    // Максимальна висота - відстань від піку до найвіддаленішої точки
    int maxHeight = 0;
    for (const auto& row : heightMap) {
        maxHeight = max(maxHeight, *max_element(row.begin(), row.end()));
    }

    // Перетворення висот на правильні значення
    for (auto& row : heightMap) {
        for (auto& height : row) {
            height = maxHeight - height;
        }
    }
}

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

    // Коригування координат для 0-індексації
    peakX--;
    peakY--;

    vector<vector<int>> heightMap(N, vector<int>(M, -1));

    fillHeightMap(heightMap, N, M, peakX, peakY);

    for (const auto& row : heightMap) {
        for (const auto& height : row) {
            cout << height << " ";
        }
        cout << endl;
    }

    return 0;
}
```

```
Microsoft Visual Studio Debu
3 4
2 2
1 2 1 0
2 3 2 1
1 2 1 0
```

Завдання №3 Algotester Lab 7-8 Variant 2

Завдання - власноруч реалізувати структуру даних "Динамічний масив".
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 написане число N - розмір масиву, який треба вставити.
  У третьому рядку N цілих чисел - масив, 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 динамічному масиві.
- Визначення кількості зарезервованої пам’яті:
  Ідентифікатор - capacity
  Ви не отримуєте аргументів.
  Ви виводите кількість зарезервованої пам’яті у динамічному масиві.
  Ваша реалізація динамічного масиву має мати фактор росту (Growth factor) рівний 2.
- Отримання значення i-го елементу
  Ідентифікатор - get
  Ви отримуєте ціле число - index, індекс елемента.
  Ви виводите значення елемента за індексом. Реалізувати використовуючи перевантаження оператора []
- Модифікація значення i-го елементу
  Ідентифікатор - set
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перевантаження оператора []

- Вивід динамічного масиву на екран
  Ідентифікатор - print
  Ви не отримуєте аргументів.
  Ви виводите усі елементи динамічного масиву через пробіл.
  Реалізувати використовуючи перевантаження оператора <<

Input

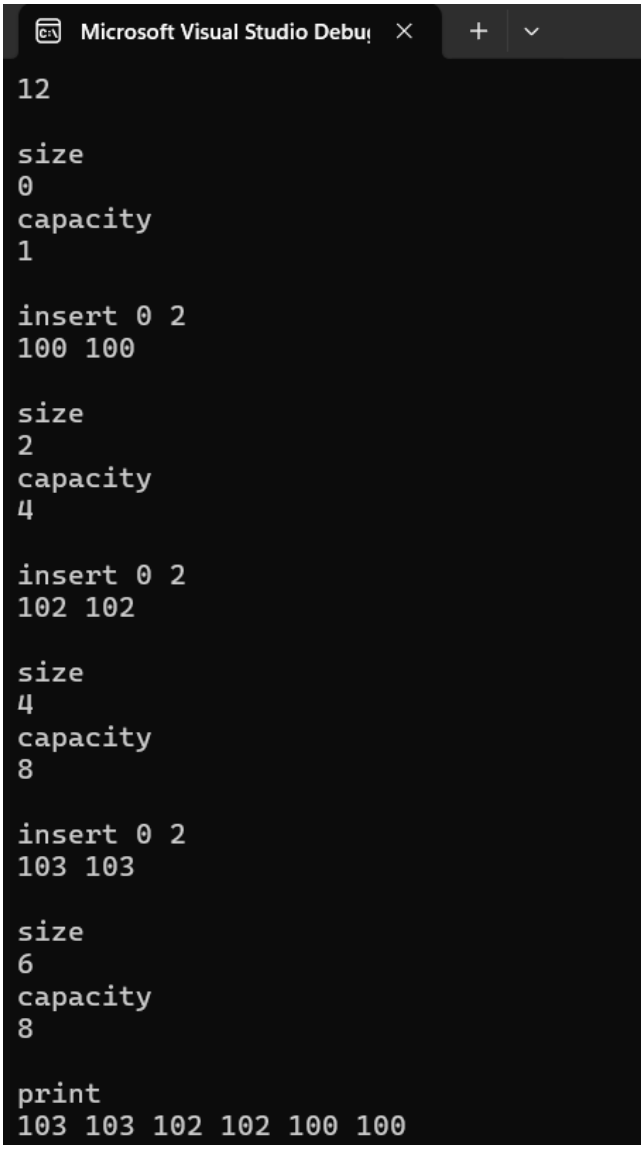
Ціле число Q - кількість запитів.
У наступних рядках Q запитів у зазначеному в умові форматі.

Output

Відповіді на запити у зазначеному в умові форматі.

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

template <typename T>
class DynamicArray
{
private:
    T* array;

public:
    int capacity;  // ємність
    int size;  // розмір
    DynamicArray(); // конструктор
    void AddElements(int index, const T* values, int num); // додавання елементів
    void erase(int index, int amount); // видалення елементів
    T get(int index); // отримання елементу
    void set(int index, const T& value); // встановлення елементу
    void print(); // виведення
};

template <typename T>
DynamicArray<T>::DynamicArray() // конструктор
{
    this->size = 0;
    this->capacity = 1;
    this->array = new T[1];
}

template <typename T>
void DynamicArray<T>::AddElements(int index, const T* values, int num)  // функція додавання
{елементів
    while (size + num >= capacity)
    {
        capacity *= 2;
    }
    T* arrTemp = new T[capacity];
    for (int i = 0; i < index; i++)
    {
        arrTemp[i] = array[i];
    }
    for (int i = 0; i < num; i++)
    {
        arrTemp[i + index] = values[i];
    }
    for (int i = index; i < size; i++)
    {
        arrTemp[i + num] = array[i];
    }
    this->size += num;
    delete[] array;
    array = arrTemp;
}

template <typename T>
void DynamicArray<T>::erase(int index, int amount)  // функція видалення елементів
{
    for (int i = index; i < size - amount; i++)
    {
        array[i] = array[i + amount];
    }

    size -= amount;
}

template <typename T>
T DynamicArray<T>::get(int index) // функція отримання елементу
{
    return array[index];
}

template <typename T>
void DynamicArray<T>::set(int index, const T& value) // функція встановлення елементу
{
    array[index] = value;
}

template <typename T>
void DynamicArray<T>::print()  // функція виведення
{
    for (int i = 0; i < size; i++)
    {
        cout << array[i] << " ";
    }
    cout << endl;
}

int main()
{
    DynamicArray<int> arr;  // створення об'єкту
    int Q; // кількість запитів
    string ch; // команда
    cin >> Q; // введення кількості запитів
    for (int i = 0; i < Q; i++)  // цикл для кожного запиту
    {
        cin >> ch;
        if (ch == "insert")
        {
            int index, N;
            cin >> index >> N;
            int* array1 = new int[N];
            for (int i = 0; i < N; i++)
            {
                cin >> array1[i];
            }
            arr.AddElements(index, array1, N);
            delete[] array1;
        }
        else if (ch == "erase")
        {
            int index, N;
            cin >> index >> N;
            arr.erase(index, N);
        }
        else if (ch == "size")
        {
            cout << arr.size << endl;
        }
        else if (ch == "capacity")
        {
            cout << arr.capacity << endl;
        }
        else if (ch == "get")
        {
            int index;
            cin >> index;
            cout << arr.get(index) << endl;
        }
        else if (ch == "set")
        {
            int index, num;
            cin >> index >> num;
            arr.set(index, num);
        }
        else if (ch == "print")
        {
            arr.print();
        }
    }
}
```

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

template <typename T>
class DynamicArray
{
private:
    T* array;

public:
    int capacity;
    int size;
    DynamicArray();
    void AddElements(int index, const T* values, int num);
    void erase(int index, int amount);
    T get(int index);
    void set(int index, const T& value);
    void print();
};

template <typename T>
DynamicArray<T>::DynamicArray() : size(0), capacity(1)  // конструктор
{
    this->array = new T[1];
}

template <typename T>
void DynamicArray<T>::AddElements(int index, const T* values, int num) // додавання
{елементів
    while (size + num >= capacity)
    {
        capacity *= 2;
    }
    T* arrTemp = new T[capacity];
    for (int i = 0; i < index; i++)
    {
        arrTemp[i] = array[i];
    }
    for (int i = 0; i < num; i++)
    {
        arrTemp[i + index] = values[i];
    }
    for (int i = index; i < size; i++)
    {
        arrTemp[i + num] = array[i];
    }
    this->size += num;
    delete[] array;
    array = arrTemp;
}

template <typename T>
void DynamicArray<T>::erase(int index, int amount) // видалення елементів
{
    for (int i = index; i < size - amount; i++)
    {
        array[i] = array[i + amount];
    }

    size -= amount;
}

template <typename T>
T DynamicArray<T>::get(int index) // отримання елемента
{
    return array[index];
}

template <typename T>
void DynamicArray<T>::set(int index, const T& value) // зміна елемента
{
    array[index] = value;
}

template <typename T>
void DynamicArray<T>::print() // виведення масиву
{
    for (int i = 0; i < size; i++)
    {
        cout << array[i] << " ";
    }
    cout << endl;
}

int main()
{
    DynamicArray<int> arr; // створення об'єкту
    int Q;
    string ch;
    cin >> Q;
    for (int i = 0; i < Q; i++) // цикл для виконання команд
    {
        cin >> ch;
        if (ch == "Insert")
        {
            int index, N;
            cin >> index >> N;
            int* array1 = new int[N];
            for (int i = 0; i < N; i++)
            {
                cin >> array1[i];
            }
            arr.AddElements(index, array1, N);
            delete[] array1;
        }
        else if (ch == "erase")
        {
            int index, N;
            cin >> index >> N;
            arr.erase(index, N);
        }
        else if (ch == "size")
        {
            cout << arr.size << endl;
        }
        else if (ch == "capacity")
        {
            cout << arr.capacity << endl;
        }
        else if (ch == "get")
        {
            int index;
            cin >> index;
            cout << arr.get(index) << endl;
        }
        else if (ch == "set")
        {
            int index, num;
            cin >> index >> num;
            arr.set(index, num);
        }
        else if (ch == "print")
        {
            arr.print();
        }
    }
}
```


Microsoft Visual Studio Debu
12
size
0
capacity
1
insert 0 2
100 100
size
2
capacity
4
insert 0 2
102 102
size
4
capacity
8
insert 0 2
103 103
size
6
capacity
8
print
103 103 102 102 100 100

Завдання №5 Practice Task

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 $\Longrightarrow$ 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

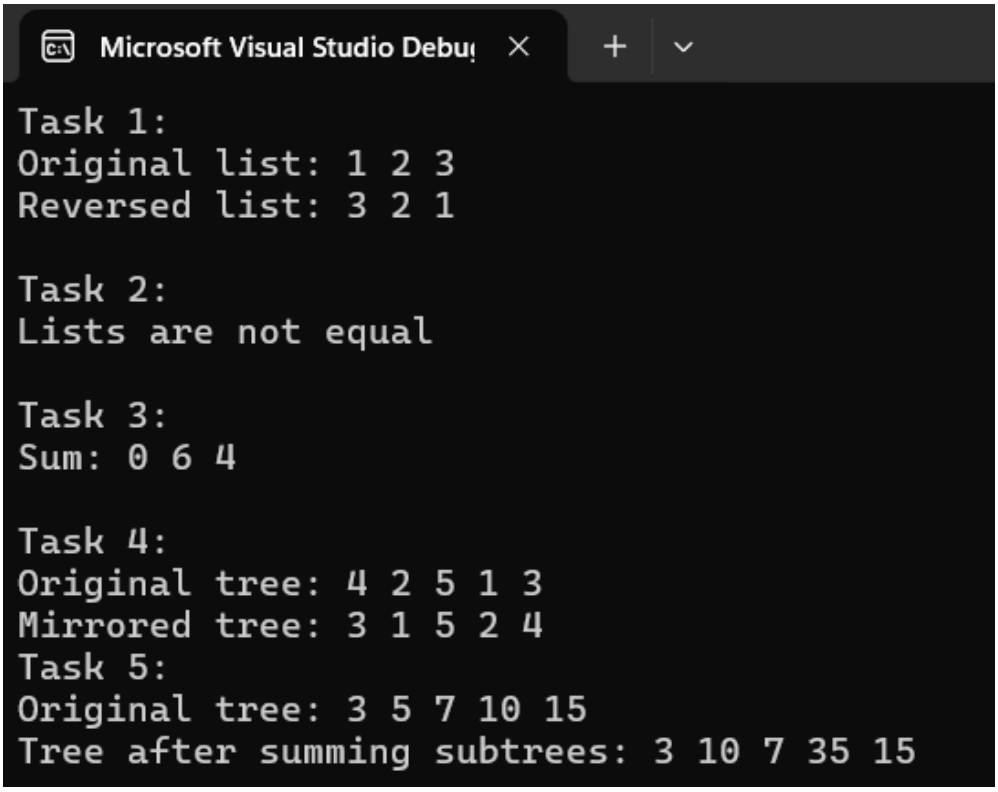
Задача №5 - Записати кожному батьківському вузлу суму підвузлів void tree_sum(TreeNode *root);

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

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

// Структура для списку
struct Node {
    int data;
    Node* next;
    Node(int val) : data(val), next(nullptr) {}
};

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

// Задача 1: Реверс списку
Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* curr = head;
    Node* next = nullptr;

    while (curr != nullptr) {
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    return prev;
}

// Допоміжна функція для виведення списку
void printList(Node* head) {
    while (head != nullptr) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}

// Задача 2: Порівняння списків
bool compare(Node* h1, Node* h2) {
    while (h1 != nullptr && h2 != nullptr) {
        if (h1->data != h2->data) {
            return false;
        }
        h1 = h1->next;
        h2 = h2->next;
    }
    return h1 == nullptr && h2 == nullptr;
}

// Задача 3: Додавання великих чисел
Node* add(Node* n1, Node* n2) {
    Node* head = nullptr;
    Node* tail = nullptr;
    int carry = 0;

    while (n1 != nullptr || n2 != nullptr || carry != 0) {
        int sum = carry;

        if (n1 != nullptr) {
            sum += n1->data;
            n1 = n1->next;
        }
        if (n2 != nullptr) {
            sum += n2->data;
            n2 = n2->next;
        }

        carry = sum / 10;
        Node* newNode = new Node(sum % 10);

        if (head == nullptr) {
            head = newNode;
            tail = head;
        }
        else {
            tail->next = newNode;
            tail = tail->next;
        }
    }
    return head;
}

// Задача 4: Віддзеркалення дерева
TreeNode* create_mirror_flip(TreeNode* root) {
    if (root == nullptr) {
        return nullptr;
    }

    TreeNode* left = create_mirror_flip(root->left);
    TreeNode* right = create_mirror_flip(root->right);

    root->left = right;
    root->right = left;

    return root;
}
```

```cpp
// Задача 5: Записати кожному батьківському вузлу суму підвузлів
int tree_sum(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }

    int leftSum = tree_sum(root->left);
    int rightSum = tree_sum(root->right);

    int totalSum = leftSum + rightSum;

    if (root->left != nullptr || root->right != nullptr) {
        root->data = totalSum;
    }

    return totalSum + root->data;
}

// Функція для виведення дерева
void printTree(TreeNode* root) {
    if (root != nullptr) {
        printTree(root->left);
        cout << root->data << " ";
        printTree(root->right);
    }
}

int main() {
    // Задача 1: Реверс списку
    cout << "Task 1: \n";
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);

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

    head = reverse(head);

    cout << "Reversed list: ";
    printList(head);
    cout << endl;

    // Задача 2: Порівняння списків
    cout << "Task 2: \n";
    Node* head2 = new Node(1);
    head2->next = new Node(2);
    head2->next->next = new Node(3);

    if (compare(head, head2)) {
        cout << "Lists are equal" << endl;
    }
    else {
        cout << "Lists are not equal" << endl;
    }
    cout << endl;

    // Задача 3: Додавання великих чисел
    cout << "Task 3: \n";
    Node* n1 = new Node(9);
    n1->next = new Node(7);
    n1->next->next = new Node(3); // 379

    Node* n2 = new Node(1);
    n2->next = new Node(8); // 81

    Node* result = add(n1, n2);

    cout << "Sum: ";
    printList(result);
    cout << endl;

    // Задача 4: Віддзеркалення дерева
    cout << "Task 4: \n";
    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 << "Original tree: ";
    printTree(root);
    cout << endl;

    root = create_mirror_flip(root);

    cout << "Mirrored tree: ";
    printTree(root);
    cout << endl;

    // Задача 5: Записати кожному батьківському вузлу суму підвузлів
    cout << "Task 5: \n";
    TreeNode* treeRoot = new TreeNode(10);
    treeRoot->left = new TreeNode(5);
    treeRoot->right = new TreeNode(15);
    treeRoot->left->left = new TreeNode(3);
    treeRoot->left->right = new TreeNode(7);

    cout << "Original tree: ";
    printTree(treeRoot);
    cout << endl;

    tree_sum(treeRoot);

    cout << "Tree after summing subtrees: ";
    printTree(treeRoot);
    cout << endl;

    return 0;
}
```


Microsoft Visual Studio Debu
Task 1:
Original list: 1 2 3
Reversed list: 3 2 1
Task 2:
Lists are not equal
Task 3:
Sum: 0 6 4
Task 4:
Original tree: 4 2 5 1 3
Mirrored tree: 3 1 5 2 4
Task 5:
Original tree: 3 5 7 10 15
Tree after summing subtrees: 3 10 7 35 15

Завдання №6 Algotester Self Practice work Lab 5 Variant 2

В пустелі існує незвичайна печера, яка є двовимірною. Висота - 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.

Input

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

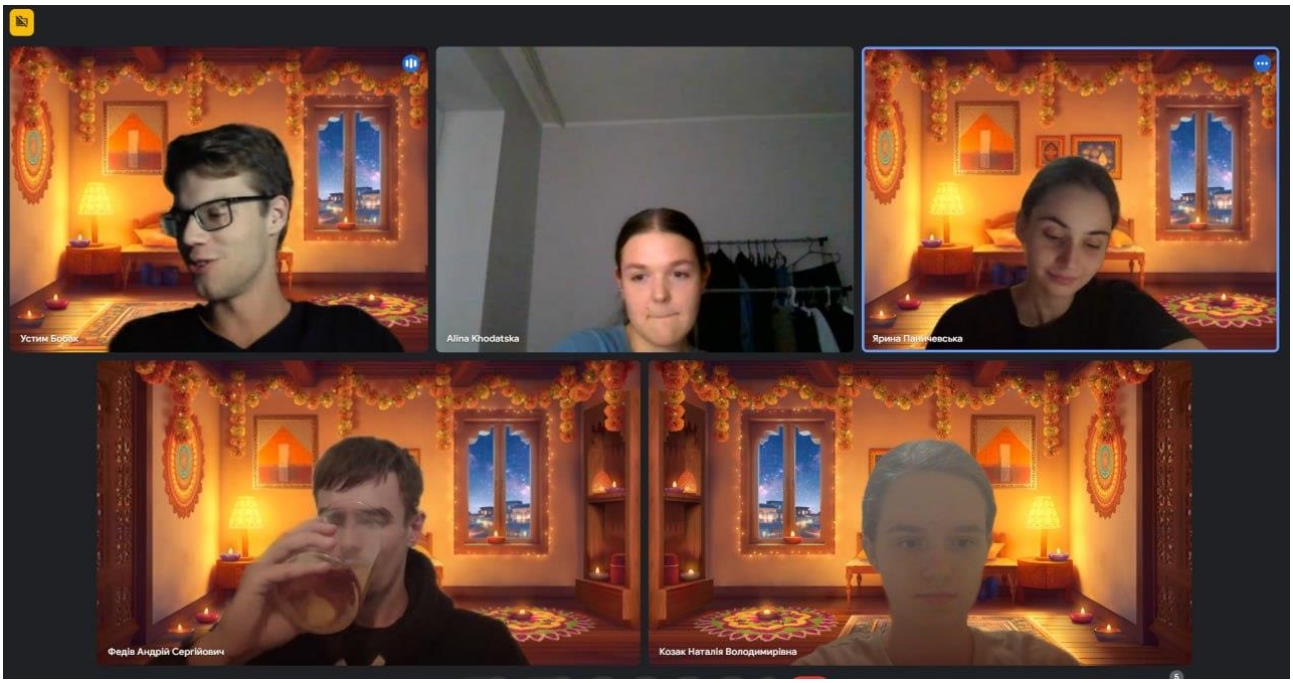
У N наступних рядках стрічка row_i яка складається з N цифр - і-й рядок матриці, яка відображає стан печери до землетрусу.

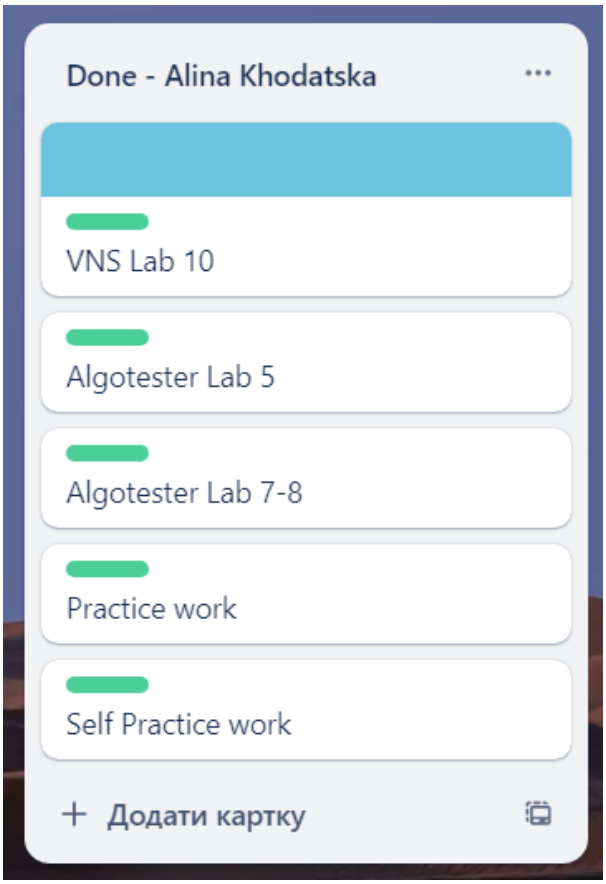
Output

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

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

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

    // Читання початкового стану печери
    vector<vector<char>> cave(N, vector<char>(M));
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cin >> cave[i][j];
        }
    }

    // Порожня лінія між вводу та виводу
    cout << endl;

    // Обробка кожного стовпця
    for (int j = 0; j < M; j++) {
        int emptyPos = N - 1;  // Позиція, куди будемо падати пісок

        // Проходимо стовпець знизу вверх
        for (int i = N - 1; i >= 0; i--) {
            if (cave[i][j] == 'S') {
                // Якщо це пісок, поміщаємо його в поточну вільну клітинку
                cave[i][j] = 'O';  // Очищаємо поточну клітинку
                cave[emptyPos][j] = 'S';  // Поміщаємо пісок в найнижче вільне місце
                emptyPos--;  // Зсуваємо позицію для наступного піску
            }
            // Якщо це камінь, просто пропускаємо
            else if (cave[i][j] == 'X') {
                emptyPos = i - 1;  // Камінь блокує падіння піску
            }
        }
    }

    // Виведення нового стану печери
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cout << cave[i][j];
        }
        cout << endl;
    }

    return 0;
}
```

Зустріч з командою та дошка в Trello





Pull request: https://github.com/artificial-intelligence-department/ai_programming_playground_2024/pull/662