Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Кафедра систем штучного інтелекту

Звіт

про виконання лабораторних та практичних робіт блоку № 6

На тему: «Програмування: алгоритм, програма, код. Системи числення. Двійкова система числення. Розробка та середовище розробки програми.»

з дисципліни: «Основи програмування»

до:

Практичних Робіт до блоку № 6

Виконав:
Студент групи ШІ-11
Вербицький Юрій Віталійович

Львів 2024

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

Мета: Засвоїти основи роботи з динамічними структурами даних, такими як черга, стек, списки та дерева. Ознайомитися з алгоритмами їх обробки для розв'язання різноманітних задач.

Теоретичні відомості та джерела:

Основи Динамічних Структур Даних: https://studfile.net/preview/7013685/page:10/
Стек: https://acode.com.ua/urok-111-stek-i-kupa/
Зв'язні Списки: https://uk.myservername.com/linked-list-data-structure-c-with-illustration#Linked_List_In_C
Дерева: https://foxminded.ua/binarne-derevo/

Виконання роботи:

VNS Lab 10v1:

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.

17.Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим номером. Додати К елементів у початок списку.

Algotester Lab 5v2:

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.

Algotester Lab 7-8 v2:

Ваше 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 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 і-го елементу
Ідентифікатор - get
Ви отримуєте ціле число - index, індекс елемента.
Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора []
Модифікація значення і-го елементу
Ідентифікатор - set
Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перегрузку оператора []
Вивід динамічного масиву на екран
Ідентифікатор - print
Ви не отримуєте аргументів.
Ви виводите усі елементи динамічного масиву через пробіл.
Реалізувати використовуючи перегрузку оператора <<

Class Practice Task:

Ваше завдання - власноруч реалізувати структуру даних "Двозв’язний список".
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 рядку написане число N - розмір списку, який треба вставити.
У третьому рядку N цілих чисел - список, який треба вставити на позицію index.
Видалення:
Ідентифікатор - erase

Ви отримуєте 2 цілих числа - index, індекс елемента, з якого почати видалення та nn - кількість елементів, яку треба видалити.

Визначення розміру:

Ідентифікатор - size

Ви не отримуєте аргументів.

Ви виводите кількість елементів у списку.

Отримання значення і-го елементу

Ідентифікатор - get

Ви отримуєте ціле число - index, індекс елемента.

Ви виводите значення елемента за індексом.

Модифікація значення іі-го елементу

Ідентифікатор - set

Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення.

Вивід списку на екран

Ідентифікатор - print

Ви не отримуєте аргументів.

Ви виводите усі елементи списку через пробіл.

Реалізувати використовуючи перегрузку оператора <<

Class 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

Self Practice

Task Algotester Lab 5v2:

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.
Клітинкі 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.

Код програми з посиланням на зовнішні ресурси

VNS Lab 10v1:

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

using namespace std;

// двозв'язний список
struct Node {
    string data;
    Node* next;
    Node* prev;
};

// створення списку
Node* createList() {
    return nullptr;
}

// друк списку
void printList(Node* head) {
    if (!head) {
        cout << "List is empty!" << endl;
        return;
    }
    Node* current = head;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

// додавання елементів на початок списку
void addToBeginning(Node*& head, const string& value) {
    Node* newNode = new Node{value, head, nullptr};
    if (head) head->prev = newNode;
    head = newNode;
}

```

```cpp
// видалення елемента зі списку за номером
void deleteByIndex(Node*& head, int index) {
    if (!head) {
        cout << "No elements for deleting." << endl;
        return;
    }
    Node* current = head;
    int currentIndex = 0;

    while (current && currentIndex < index) {
        current = current->next;
        currentIndex++;
    }

    if (!current) {
        cout << "Element with index " << index << " was not found." << endl;
        return;
    }

    if (current->prev) current->prev->next = current->next;
    if (current->next) current->next->prev = current->prev;
    if (current == head) head = current->next;

    delete current;
    cout << "Element with index " << index << " deleted." << endl;
}

// запис списка у файл
void saveToFile(Node* head, const string& filename) {
    ofstream file(filename);
    if (!file.is_open()) {
        cerr << "Error open file for writing." << endl;
        return;
    }
    Node* current = head;
    while (current) {
        file << current->data << endl;
        current = current->next;
    }
```

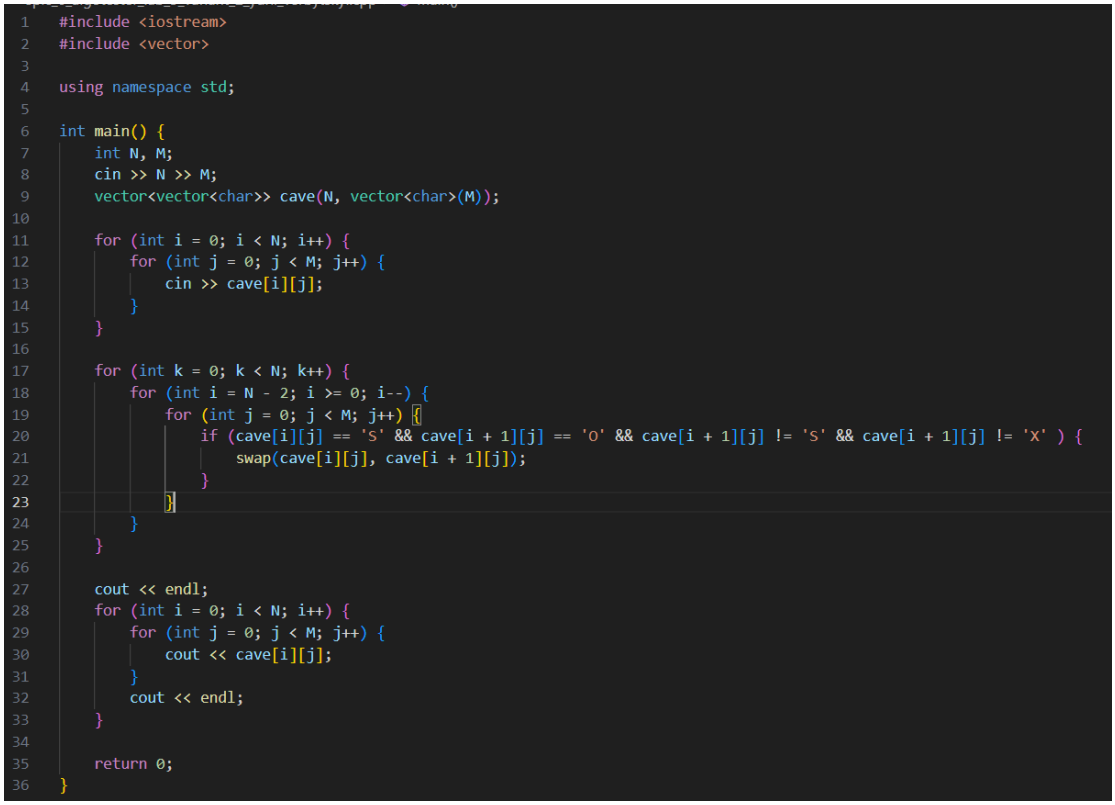
```
    file.close();
    cout << "Список записано у файл " << filename << endl;
}

// знищення списку
void destroyList(Node*& head) {
    while (head) {
        Node* temp = head;
        head = head->next;
        delete temp;
    }
    cout << "List deleted." << endl;
}

// відновлення списку з файлу
void restoreFromFile(Node*& head, const string& filename) {
    ifstream file(filename);
    if (!file.is_open()) {
        cerr << "Error open file for reading." << endl;
        return;
    }

    destroyList(head);

    string line;
    while (getline(file, line)) {
        addToBeginning(head, line);
    }
    file.close();
    cout << "List restored from file " << filename << endl;
}

// Головна функція
int main() {
    Node* list = createList();
    string filename = "list_data.txt";

    // 1. Створення списку
```

```
    // 1. Створення списку
    cout << "Create list." << endl;
    addToBeginning(list, "el3");
    addToBeginning(list, "el2");
    addToBeginning(list, "el1");

    // 2. Друк списку
    cout << "Current list: ";
    printList(list);

    // 3. Додавання кількох елементів у початок
    cout << "Add elements to beginning." << endl;
    addToBeginning(list, "new1");
    addToBeginning(list, "new2");
    cout << "List after adding: ";
    printList(list);

    // 4. Видалення елемента за номером
    cout << "Delete element with index 2." << endl;
    deleteByIndex(list, 2);
    cout << "List after deleting the element: ";
    printList(list);

    // 5. Запис списку у файл
    saveToFile(list, filename);

    // 6. Знищення списку
    destroyList(list);
    cout << "List after deleting: ";
    printList(list);

    // 7. Відновлення списку з файлу
    restoreFromFile(list, filename);
    cout << "List after recovery: ";
    printList(list);

    // 8. Знищення списку
    destroyList(list);
```

```
    cout << "List after final deleting: ";
    printList(list);

    return 0;
}
```

Витрачено ~3 години

Algotester Lab 5v2:



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

using namespace std;

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

    for (int k = 0; k < N; k++) {
        for (int i = N - 2; i >= 0; i--) {
            for (int j = 0; j < M; j++) {
                if (cave[i][j] == 'S' && cave[i + 1][j] == 'O' && cave[i + 1][j] != 'S' && cave[i + 1][j] != 'X' ) {
                    swap(cave[i][j], cave[i + 1][j]);
                }
            }
        }
    }

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

    return 0;
}
```

Витрачено ~1 годину

Algotester Lab 7-8 v2:

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

template <class T>
class dynamicArray {
private:
    T *data;

public:
    int size;
    int capacity;

    dynamicArray() {
        this->size = 0;
        this->capacity = 1;
        this->data = new T[1];
    }

    void insert(int index, int quantity, T *elements) {
        while (size + quantity >= capacity)
            capacity *= 2;
        T *temp = new T[capacity];

        for (int i = 0; i < index; i++)
            temp[i] = data[i];

        for (int i = 0; i < quantity; i++)
            temp[i + index] = elements[i];

        for (int i = index; i < size; i++)
            temp[i + quantity] = data[i];

        this->size += quantity;
        delete[] data;
        data = temp;
    }

    void erase(int index, int quantity) {
        T *temp = new T[capacity];
        int newSize = 0;
        for (int i = 0; i < this->size; i++)
        {
            if (i < index || i >= index + quantity)
            {
                temp[newSize] = data[i];
                newSize++;
            }
        }
        this->size -= quantity;
        delete[] data;
        data = temp;
    }

    T get(int index) {
        return this->data[index];
    }

    void set(int index, T value) {
        this->data[index] = value;
    }

    void print(const string &space) {
        for (int i = 0; i < this->size; i++) {
            cout << data[i];
            if (i < size - 1) cout << space;
        }
        cout << endl;
    }
};

int main() {
    dynamicArray<int> array;

    int Q;
    cin >> Q;
```

```
76
77      while (Q--) {
78
79
80          string command;
81          cin >> command;
82          if (command == "insert") {
83              int index, quantity;
84              cin >> index >> quantity;
85              int *elements = new int[quantity];
86
87              for (int i = 0; i < quantity; i++) {
88                  cin >> elements[i];
89              }
90
91              array.insert(index, quantity, elements);
92              delete[] elements;
93          } else if (command == "erase") {
94              int index, quantity;
95              cin >> index >> quantity;
96              array.erase(index, quantity);
97          } else if (command == "size") {
98              cout << array.size << endl;
99          } else if (command == "capacity") {
100             cout << array.capacity << endl;
101         } else if (command == "get") {
102             int index;
103             cin >> index;
104             cout << array.get(index) << endl;
105         } else if (command == "set") {
106             int index, quantity;
107             cin >> index >> quantity;
108             array.set(index, quantity);
109         } else if (command == "print") {
110             array.print(" ");
111         }
112     }
113     return 0;
```

Витрачено ~4 години

Class Practice Task 1:

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

struct Node {
    int data;
    Node* next;

    Node(int value) : data(value), next(nullptr) {} // Конструктор
};

// Функція для перевертання списку
Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* current = head;

    while (current != nullptr) {
        Node* nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }
    return prev; // новий head списку
}

// Функція для порівняння двох 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;
    }

    // Якщо один з списків закінчився раніше
    return h1 == nullptr && h2 == nullptr;
}

// Функція для додавання двох чисел(у вигляді списків)
Node* add(Node* n1, Node* n2) {
    Node* result = 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;
        int digit = sum % 10;

        Node* newNode = new Node(digit);
        if (result == nullptr) {
            result = newNode;
            tail = result;
        } else {
            tail->next = newNode;
            tail = tail->next;
        }
    }

    return result;
}

// Функція для додавання елемента в кінець списку
Node* addInEndNode(Node* head, int value) {
    Node* newNode = new Node(value);
    if (head == nullptr) {
        return newNode;
```

```
        return newNode;
    }

    Node* temp = head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;

    return head;
}

// Функція для створення списку з масиву
Node* createList(const int arr[], int size) {
    Node* head = nullptr;
    for (int i = 0; i < size; ++i) {
        head = addInEndNode(head, arr[i]);
    }
    return head;
}

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

int main() {
    int arr1[] = {2, 7, 3, 4, 5};
    int arr2[] = {2, 7, 3, 4, 5};
    int arr3[] = {1, 5, 9};
    int arr4[] = {4, 5, 1};

    int size1 = sizeof(arr1) / sizeof(arr1[0]);
    int size2 = sizeof(arr2) / sizeof(arr2[0]);
    int size3 = sizeof(arr3) / sizeof(arr3[0]);
    int size4 = sizeof(arr4) / sizeof(arr4[0]);

    Node* head1 = createList(arr1, size1);
    Node* head2 = createList(arr2, size2);
    Node* head3 = createList(arr3, size3);
    Node* head4 = createList(arr4, size4);

    cout << "Source list 1: ";
    printList(head1);
    cout << "Source list 2: ";
    printList(head2);

    // Порівнюємо список 1 і 2
    if (compare(head1, head2)) {
        cout << "Lists are the same!" << endl;
    } else {
        cout << "Lists are different!" << endl;
    }

    // Перевертаємо список 1
    head1 = reverse(head1);
    cout << "Reversed list 1: ";
    printList(head1);

    // Додаємо список 3 і 4
    Node* sumHead = add(head3, head4);
    cout << "Adding list 3 and list 4: ";
    printList(sumHead);

    return 0;
}
```

Витрачено ~4 години

Class Practice Task 2:

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

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

    TreeNode(int value) : data(value), left(nullptr), right(nullptr) {} // Конструктор
};

// Функція для створення віддзеркаленого дерева
TreeNode* create_mirror_flip(TreeNode* root) {
    if (root == nullptr) {
        return nullptr;
    }

    TreeNode* newNode = new TreeNode(root->data);

    // обмін
    newNode->left = create_mirror_flip(root->right);
    newNode->right = create_mirror_flip(root->left);

    return newNode;
}

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

// сума підвузлів
int tree_sum(TreeNode* root) {
    if (root == nullptr) {
        return 0;
    }

    if (root->left == nullptr && root->right == nullptr) { // для листка
        return root->data;
    }

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

    root->data = leftSum + rightSum;

    return root->data;
}

// заповнення дерева
TreeNode* insert(TreeNode* root, int value) {
    if (root == nullptr) {
        return new TreeNode(value);
    }

    if (value < root->data) {
        root->left = insert(root->left, value);
    } else {
        root->right = insert(root->right, value);
    }

    return root;
}

int main() {
    TreeNode* root = nullptr;

    root = insert(root, 10);
    insert(root, 5);
    insert(root, 15);
    insert(root, 3);
```

```
    insert(root, 7);
    insert(root, 12);
    insert(root, 18);

    cout << "Original tree (preorder): ";
    preorderPrint(root);
    cout << endl;

    TreeNode* mirroredTree = create_mirror_flip(root);

    cout << "Mirrored tree (preorder): ";
    preorderPrint(mirroredTree);
    cout << endl;

    tree_sum(root);
    cout << "Tree after sum (preorder): ";
    preorderPrint(root);

    return 0;
}
```

Витрачено ~3,5 години

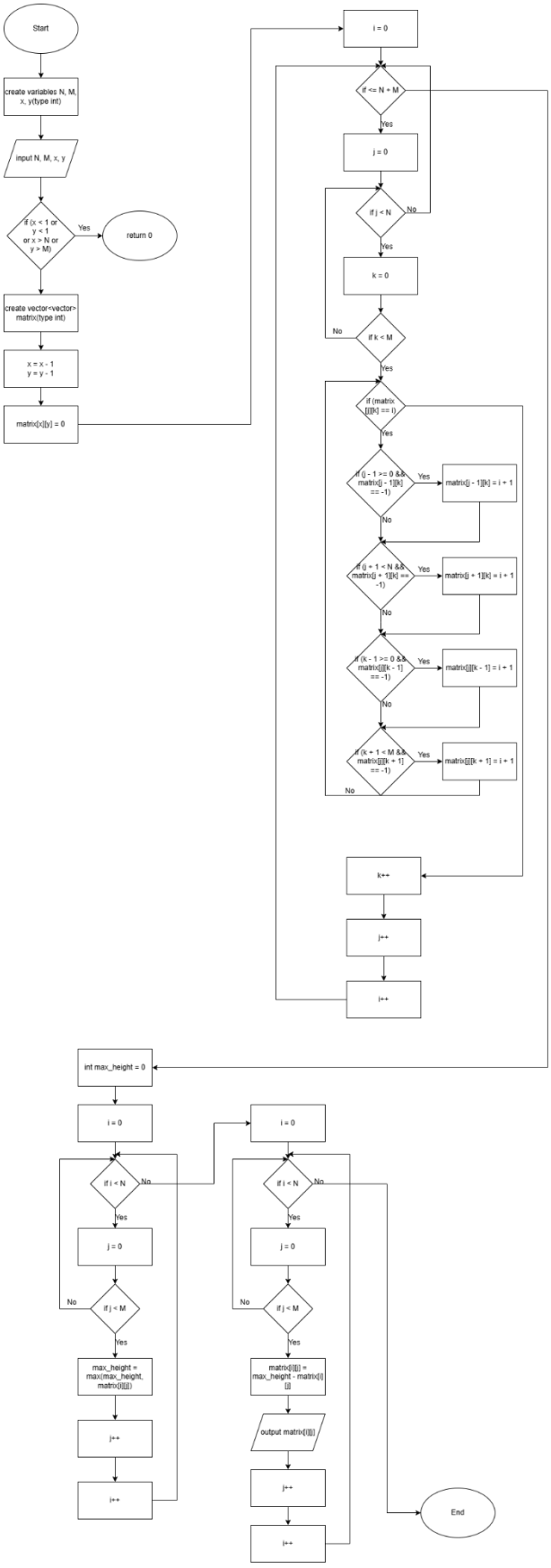
SelfPractice:

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

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

    int x, y;
    cin >> x >> y;

    if (x < 1 || y < 1 || x > N || y > M) return 0;

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

    x = x - 1, y = y - 1;
    matrix[x][y] = 0;

    // всі можливі висоти
    for (int i = 0; i <= N + M; i++) {
        for (int j = 0; j < N; j++) { // по вертикалі
            for (int k = 0; k < M; k++) { // по горизанталі
                if (matrix[j][k] == i) {
                    if (j - 1 >= 0 && matrix[j - 1][k] == -1) {
                        matrix[j - 1][k] = i + 1;
                    }
                    if (j + 1 < N && matrix[j + 1][k] == -1) {
                        matrix[j + 1][k] = i + 1;
                    }
                    if (k - 1 >= 0 && matrix[j][k - 1] == -1) {
                        matrix[j][k - 1] = i + 1;
                    }
                    if (k + 1 < M && matrix[j][k + 1] == -1) {
                        matrix[j][k + 1] = i + 1;
                    }
                }
            }
```

```
        }
    }

    int max_height = 0;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            max_height = max(max_height, matrix[i][j]);
        }
    }

    // перетворюємо висоти
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            matrix[i][j] = max_height - matrix[i][j];
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Витрачено ~3 години

Block-scheme for task Algotester Lab 5v3



Результати виконаних завдань, тестування та фактично затрачений час

VNS Lab 10v1: ~4 години

```
Create list.
Current list: el1 el2 el3
Add elements to beginning.
List after adding: new2 new1 el1 el2 el3
Delete element with index 2.
Element with index 2 deleted.
List after deleting the element: new2 new1 el2 el3
Список записано у файл list_data.txt
List deleted.
List after deleting: List is empty!
List deleted.
List restored from file list_data.txt
List after recovery: el3 el2 new1 new2
List deleted.
List after final deleting: List is empty!
```

Algotester Lab 5v2: ~1 годину

```
5 5
SSOSS
OOOOO
SOOXX
OOOOS
OOSOO

OOOOO
OOOSS
OOOXX
SOOOO
SSSOS
```

Algotester Lab 7-8 v2: ~5 години

```
12

size
0

insert 0 5

capacity
8
print
251 252 253 254 255

252
777

erase 1 3

get 1
255

size
2
print
251 255
```

✔ Lab 78v2

Class Practice Task 1: ~2 години

```
Source list 1: 2 7 3 4 5
Source list 2: 2 7 3 4 5
Lists are the same!
Reversed list 1: 5 4 3 7 2
Adding list 3 and list 4: 5 0 1 1
```

Class Practice Task 2: ~3 години

```
Original tree (preorder): 10 5 3 7 15 12 18
Mirrored tree (preorder): 10 15 18 12 5 7 3
Tree after sum (preorder): 40 10 3 7 30 12 18
```

SelfPractice: 50 хвилин

✔ Lab 5v3

```
3 4
2 2
1 2 1 0
2 3 2 1
1 2 1 0
```

Висновок:

Отже, в межах цього епіка я старався зрозуміти, що таке списки, дерева та як їх реалізовувати в коді. Практикувався з записом даних у файли, а також покращив роботу з масивами та алгоритмами.

Посилання на pull request: