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-13
Литвин Маркіян Назарович

Львів 2024

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

Мета роботи: Навчитися працювати з різними видами динамічних структур, створювати чіткі і структуровані програми. Ознайомитись з алгоритмами їх обробки.

Теоретичні відомості:

- Стек
- Дерево
- Списки
- Черга

Джерела:

- https://www.youtube.com/watch?v=ZYvYISxaNL0
- https://www.youtube.com/watch?v=qBFzNW0ALxQ
- https://www.youtube.com/watch?v=25REjF_atI&list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&index=139
- https://www.youtube.com/watch?v=Yhw8NbjrSFA

Виконання роботи

Завдання 1: VNS Lab 10 - Task 1-17

Умова:

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.

Розв’язок:

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

using namespace std;

class Node {
public:
    string data;
    Node* next;
    Node* prev;
    Node(const string& data) : data(data), next(nullptr), prev(nullptr) {}
};

class DoublyLinkedList {
public:
    Node* head;
    Node* tail;
    DoublyLinkedList() : head(nullptr), tail(nullptr) {}

    void add_begin(const string& data) {
        Node* new_node = new Node(data);
        if (!head) {
            head = tail = new_node;
        }
        else {
            new_node->next = head;
            head->prev = new_node;
            head = new_node;
        }
    }
    void add_end(const string& data) {
        Node* new_node = new Node(data);
        if (!tail) {
            head = tail = new_node;
        }
        else {
            tail->next = new_node;
            new_node->prev = tail;
            tail = new_node;
        }
    }
    void delete_value(const string& value) {
        Node* current = head;
        while (current) {
            if (current->data == value) {
                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;

                delete current;
                cout << "Вузол зі значенням \"" << value << "\" видалений." << endl;
                return;
            }
            current = current->next;
        }
    }
    void delete_id(int id) {
        if (!head) {
            cout << "Список порожній." << endl;
            return;
        }
        Node* current = head;
        int current_id = 0;

        while (current && current_id < id) {
            current = current->next;
            current_id++;
        }
        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;

        delete current;
        cout << "Елемент з індексом " << id << " видалений." << endl;
    }
    void print() {
        if (!head) {
```

```
            cout << "Список порожній." << endl;
            return;
        }
        Node* current = head;
        while (current) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }
    void save(const string& filename) {
        ofstream file(filename);
        if (!file.is_open()) {
            return;
        }
        Node* current = head;
        while (current) {
            file << current->data << endl;
            current = current->next;
        }
        file.close();
        cout << "Список збережено. \"" << filename << "\"" << endl;
    }
    void restore(const string& filename) {
        ifstream file(filename);
        if (!file.is_open()) {
            return;
        }
        clear();
        string data;
        while (getline(file, data)) {
            add_end(data);
        }
        file.close();
        cout << "Список видалено. \"" << filename << "\"" << endl;
    }
    void clear() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        cout << "Список очищений." << endl;
    }
    ~DoublyLinkedList() {
        clear();
    }
};

int main() {
    DoublyLinkedList list;
    string filename = "vns_lab_10_markiian_lytvyn.txt";

    cout << "Створення списку." << endl;
    list.add_begin("1");
    list.add_begin("2");
    list.add_begin("3");
    list.add_begin("4");
    list.add_begin("5");
    list.add_begin("6");
    list.add_begin("7");
    list.add_begin("8");
    list.add_begin("9");
    list.add_begin("10");

    cout << "Список: ";
    list.print();

    cout << "Додавання елементів на початок списку." << endl;
    list.add_begin("11");
    list.add_begin("12");
    cout << "Список: ";
    list.print();


    int K;
    cout << "Введіть індекс K:";
    cin >> K;
    cout << "Видалення елементів з індексом K." << endl;
    list.delete_id(K);
    cout << "Список: ";
```

```
    cout << "Список: ";
    list.print();


    cout << "Видалення вузла зі значенням \"4\"." << endl;
    list.delete_value("4");
    cout << "Список: ";
    list.print();

    list.save(filename);
    list.clear();
    cout << "Список після очищення: ";
    list.print();

    list.restore(filename);
    cout << "Список після відновлення: ";
    list.print();

    list.clear();
    cout << "Список після остаточного очищення: ";
    list.print();

    return 0;
}

```

Результат:

```
Створення списку.
Список: 10 9 8 7 6 5 4 3 2 1
Додавання елементів на початок списку.
Список: 12 11 10 9 8 7 6 5 4 3 2 1
Введіть індекс K:4
Видалення елементів з індексом K.
Елемент з індексом 4 видалений.
Список: 12 11 10 9 7 6 5 4 3 2 1
Видалення вузла зі значенням "4".
Вузол зі значенням "4" видалений.
Список: 12 11 10 9 7 6 5 3 2 1
Список збережено. "vns_lab_10_markiian_lytvyn.txt"
Список очищений.
Список після очищення: Список порожній.
Список очищений.
Список видалено. "vns_lab_10_markiian_lytvyn.txt"
Список після відновлення: 12 11 10 9 7 6 5 3 2 1
Список очищений.
Список після остаточного очищення: Список порожній.
Список очищений.
PS D:\Epics>
```

Час виконання ~ 1.5 год

Завдання 2: Algotester Lab 5v2

Умова:

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це NN, ширина - MM.
Всередині печери є пустота, пісок та каміння. Пустота позначається буквою OO , пісок SS і каміння XX;
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 NN та MM - висота та ширина печери
У NN наступних рядках стрічка rowirowi яка складається з NN цифер - i-й рядок матриці, яка відображає стан печери до землетрусу.

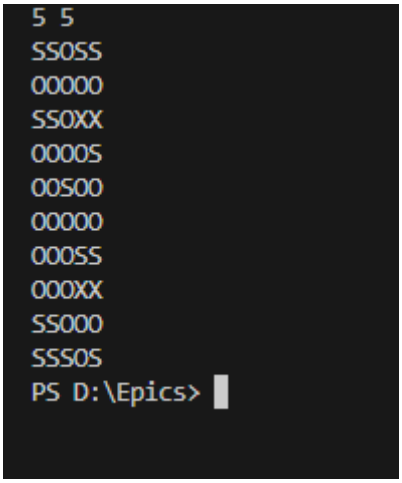
Output

NN рядків, які складаються з стрічки розміром MM - стан печери після землетрусу.

Розв’язок:

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

using namespace std;

int main() {
    int N, M;
    cin >> N >> M;
    vector<vector<char>> arr(N, vector<char>(M));

    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            cin >> arr[i][j];
        }
    }
    for (int j = 0; j < M; ++j) {
        int row = N - 1;
        for (int i = N - 1; i >= 0; --i) {
            if (arr[i][j] == 'X') {
                row = i - 1;
            } else if (arr[i][j] == 'S') {
                arr[i][j] = '0';
                arr[row][j] = 'S';
                row--;
            }
        }
    }

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

```

Результат:



Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a few seconds ago	C++ 23	Accepted	0.071	2.250	View

Час виконання ~ 30 хв

Завдання 3: Algotester Lab 78v1

Умова:

Ваше завдання - власноруч реалізувати структуру даних "Двозв’язний список".
Ви отримаєте QQ запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.

Вам будуть поступати запити такого типу:

- **Вставка**:
  Ідентифікатор - insertinsert
  Ви отримуєте ціле число indexindex елемента, на місце якого робити вставку.
  Після цього в наступному рядку рядку написане число NN - розмір списку, який треба вставити.
  У третьому рядку NN цілих чисел - список, який треба вставити на позицію indexindex.
- **Видалення**:
  Ідентифікатор - eraseerase
  Ви отримуєте 2 цілих числа - indexindex, індекс елемента, з якого почати видалення та nn - кількість елементів, яку треба видалити.
- **Визначення розміру**:
  Ідентифікатор - sizesize
  Ви не отримуєте аргументів.
  Ви виводите кількість елементів у списку.
- **Отримання значення** ii-го елементу
  Ідентифікатор - getget
  Ви отримуєте ціле число - indexindex, індекс елемента.
  Ви виводите значення елемента за індексом.

- **Модифікація значення** ii-го елементу
  Ідентифікатор - setset
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення.
- **Вивід списку на екран**
  Ідентифікатор - printprint
  Ви не отримуєте аргументів.
  Ви виводите усі елементи списку через пробіл.
  Реалізувати використовуючи перегрузку оператора <<<<

Input

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

Output

Відповіді на запити у зазначеному в умові форматі.

Розв’язок:

```
                head->prev = new_node;
                head = new_node;
            } else {
                Node<T>* prev_node = current->prev;
                prev_node->next = new_node;
                new_node->prev = prev_node;
                new_node->next = current;
                current->prev = new_node;
            }
            ++list_size;
        }
    }

    void erase(int index, int amount) {
        Node<T>* current = getNodeAt(index);

        for (int i = 0; i < amount; ++i) {
            Node<T>* del = current;
            current = current->next;

            if (del->prev) {
                del->prev->next = del->next;
            } else {
                head = del->next;
            }

            if (del->next) {
                del->next->prev = del->prev;
            } else {
                tail = del->prev;
            }

            delete del;
            --list_size;
        }
    }

    size_t size() const {
        return list_size;
    }

    T get(int index) const {
        return getNodeAt(index)->data;
    }

    void set(int index, const T& value) {
        getNodeAt(index)->data = value;
    }

    void print() const {
        Node<T>* current = head;
        while (current) {
            cout << current->data << " ";
            current = current->next;
        }
        cout << endl;
    }

private:
    Node<T>* getNodeAt(int index) const {
        Node<T>* current = (index < list_size / 2) ? head : tail;
        if (index < list_size / 2) {
            for (int i = 0; i < index; ++i) {
                current = current->next;
            }
        } else {
            for (int i = list_size - 1; i > index; --i) {
                current = current->prev;
            }
        }
        return current;
    }
};

int main() {
    DoubleLinkedList<int> list;
    int Q;
    cin >> Q;

    while (Q--) {
        string answer;
        cin >> answer;

        if (answer == "insert") {
            int index, N;
            cin >> index >> N;
            vector<int> values(N);
```

```cpp
            vector<int> values(N);
            for (int i = 0; i < N; ++i) {
                cin >> values[i];
            }
            list.insert(index, values);
        }
        else if (answer == "erase") {
            int index, count;
            cin >> index >> count;
            list.erase(index, count);
        }
        else if (answer == "size") {
            cout << list.size() << endl;
        }
        else if (answer == "get") {
            int index;
            cin >> index;
            cout << list.get(index) << endl;
        }
        else if (answer == "set") {
            int index, value;
            cin >> index >> value;
            list.set(index, value);
        }
        else if (answer == "print") {
            list.print();
        }
    }

    return 0;
}
```

Результат:

```
5
insert
0 3
1 2 3
erase
0 2
set
insert
0 3
1 2 3
erase
0 2
set
0 10
size
0 10
size
1
print
10
print
10
10
PS D:\Epics>
```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a few seconds ago	C++ 23	Accepted	0.008	1.266	View

Час виконання ~ 2.5 год

Завдання 4: Class Practice Work

Умова:

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 $\implies$ 9→7→3);
- функція повертає новий список, передані в функцію списки не модифікуються.

Розв’язок:

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

struct Node {
    int data;
    Node* next;

    Node(int val) : data(val), next(NULL) {}
};


// Task 1
Node* reverse(Node* head) {
    Node* prev = NULL;
    Node* current = head;
    Node* next = NULL;

    while (current != NULL) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    return prev;
}

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

// Task 2
bool compare(Node* h1, Node* h2) {
    while (h1 != NULL && h2 != NULL) {
        if (h1->data != h2->data)
            return false;
        h1 = h1->next;
        h2 = h2->next;
    }
    return h1 == NULL && h2 == NULL;
```

```
    return h1 == NULL && h2 == NULL;
}

// Task 3
Node* add(Node* n1, Node* n2) {
    Node* result = NULL;
    Node* tail = NULL;
    int carry = 0;

    while (n1 != NULL || n2 != NULL || carry != 0) {
        int sum = carry;
        if (n1 != NULL) {
            sum += n1->data;
            n1 = n1->next;
        }
        if (n2 != NULL) {
            sum += n2->data;
            n2 = n2->next;
        }

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

        if (result == NULL) {
            result = newNode;
            tail = result;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }

    return result;
}



int main() {
    // Task 1
    Node* list1 = new Node(1);
    list1->next = new Node(2);
    list1->next->next = new Node(3);
```

```
87      cout << "Вхідний список: ";
88      printList(list1);
89      list1 = reverse(list1);
90      cout << "Обернений список: ";
91      printList(list1);
92
93      // Task 2
94      Node* list2 = new Node(1);
95      list2->next = new Node(2);
96      list2->next->next = new Node(3);
97
98      // Task 3
99      Node* num1 = new Node(9);
100     num1->next = new Node(9);
101     Node* num2 = new Node(1);
102     Node* sum = add(num1, num2);
103     cout << "Сума чисел: ";
104     printList(sum);
105     return 0;
106 }
107
```

Результат:

```
Вхідний список: 1 -> 2 -> 3 ->
Вхідний список: 1 -> 2 -> 3 ->
Обернений список: 3 -> 2 -> 1 ->
Сума чисел: 0 -> 0 -> 1 ->
PS D:\Epics>
```

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

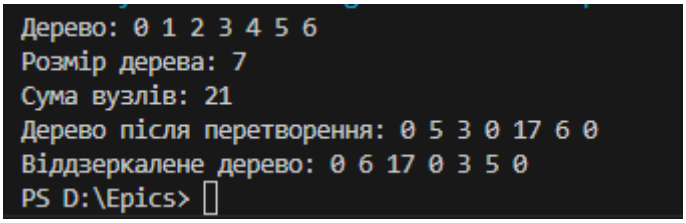
Розв’язок:

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

template <typename T>
class BinaryTree {
private:
    struct Node {
        T data;
        Node* left;
        Node* right;

        Node(T value) : data(value), left(nullptr), right(nullptr) {}
    };
    Node* root;
    size_t size(const Node* node) const {
        return node == nullptr ? 0 : 1 + size(node->left) + size(node->right);
    }
    void mirror_n(Node* node) {
        if (node == nullptr)
            return;

        swap(node->left, node->right);
        mirror_n(node->left);
        mirror_n(node->right);
    }
    T sum(Node* node) const {
        if (node == nullptr)
            return 0;

        T left_sum = sum(node->left);
        T right_sum = sum(node->right);

        return node->data + left_sum + right_sum;
    }
    void convert(Node* node) {
        if (node == nullptr)
            return;

        T left_sum = node->left ? sum(node->left) : 0;
        T right_sum = node->right ? sum(node->right) : 0;
        node->data = left_sum + right_sum;

        convert(node->left);
        convert(node->right);
    }
    Node* insert_n(Node* node, T value) {
        if (node == nullptr)
            return new Node(value);
```

```
            return new Node(value);

        if (value < node->data)
            node->left = insert_n(node->left, value);
        else if (value > node->data)
            node->right = insert_n(node->right, value);
        return node;
    }
    bool find_n(const Node* node, const T& value) const {
        if (node == nullptr)
            return false;

        if (node->data == value)
            return true;
        else if (value < node->data)
            return find_n(node->left, value);
        else
            return find_n(node->right, value);
    }
    void print(const Node* node) const {
        if (node == nullptr)
            return;

        print(node->left);
        cout << node->data << " ";
        print(node->right);
    }

public:
    BinaryTree() : root(nullptr) {}
    ~BinaryTree() {
        clear(root);
    }
    void clear(Node* node) {
        if (node == nullptr)
            return;

        clear(node->left);
        clear(node->right);
        delete node;
    }
    size_t size() const {
        return size(root);
    }
    void insert(T value) {
        root = insert_n(root, value);
    }
```

```
    }
    bool find(const T& value) const {
        return find_n(root, value);
    }
    void mirror() {
        mirror_n(root);
    }
    T sum_tree() const {
        return sum(root);
    }
    void convert() {
        convert(root);
    }
    void print() const {
        print(root);
        cout << endl;
    }
};

int main() {
    BinaryTree<int> tree;

    int values[] = {4, 1, 2, 3, 5, 6, 0};
    for (int value : values)
        tree.insert(value);

    cout << "Дерево: ";
    tree.print();

    cout << "Розмір дерева: " << tree.size() << endl;

    cout << "Сума вузлів: " << tree.sum_tree() << endl;

    tree.convert();
    cout << "Дерево після перетворення: ";
    tree.print();

    tree.mirror();
    cout << "Віддзеркалене дерево: ";
    tree.print();

    return 0;
}
```

Результат:



Час виконання ~ 2.5 години

Завдання 6: Self Practice Work

Умова:

У вас є карта гори розміром N×MN×M.
Також ви знаєте координати {x,y}{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.

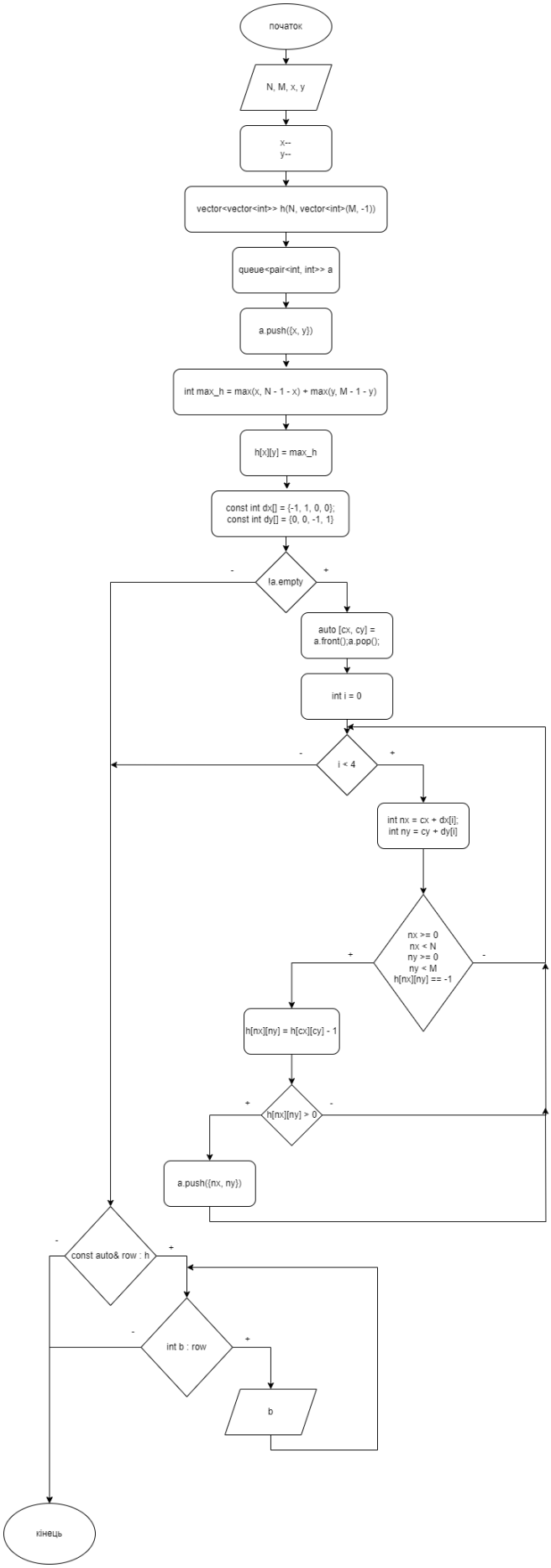
Input

У першому рядку 2 числа NN та MM - розміри карти
у другому рядку 2 числа xx та yy - координати піку гори

Output

NN рядків по MM елементів в рядку через пробіл - висоти карти.

Блок-схема:


початок
N, M, x, y
x--
y--
vector<vector<int>> h(N, vector<int>(M, -1))
queue<pair<int, int>> a
a.push({x, y})
int max_h = max(x, N - 1 - x) + max(y, M - 1 - y)
h[x][y] = max_h
const int dx[] = {-1, 1, 0, 0};
const int dy[] = {0, 0, -1, 1}
-
+
!a.empty
auto [cx, cy] = a.front();a.pop();
int i = 0
-
+
i < 4
int nx = cx + dx[i];
int ny = cy + dy[i]
nx >= 0
nx < N
ny >= 0
ny < M
h[nx][ny] == -1
+
-
h[nx][ny] = h[cx][cy] - 1
+
-
h[nx][ny] > 0
a.push({nx, ny})
-
+
const auto& row : h
-
+
int b : row
b
кінець

Розв’язок:

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

using namespace std;

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

    x--;
    y--;

    vector<vector<int>> h(N, vector<int>(M, -1));
    queue<pair<int, int>> a;

    a.push({x, y});
    int max_h = max(x, N - 1 - x) + max(y, M - 1 - y);
    h[x][y] = max_h;
    const int dx[] = {-1, 1, 0, 0};
    const int dy[] = {0, 0, -1, 1};
    while (!a.empty()) {
        auto [cx, cy] = a.front();
        a.pop();

        for (int i = 0; i < 4; i++) {
            int nx = cx + dx[i];
            int ny = cy + dy[i];
            if (nx >= 0 && nx < N && ny >= 0 && ny < M && h[nx][ny] == -1) {
                h[nx][ny] = h[cx][cy] - 1;
                if (h[nx][ny] > 0) {
                    a.push({nx, ny});
                }
            }
        }
    }
    for (const auto& row : h) {
        for (int b : row) {
            cout << b << " ";
        }
        cout << endl;
    }

    return 0;
}
```

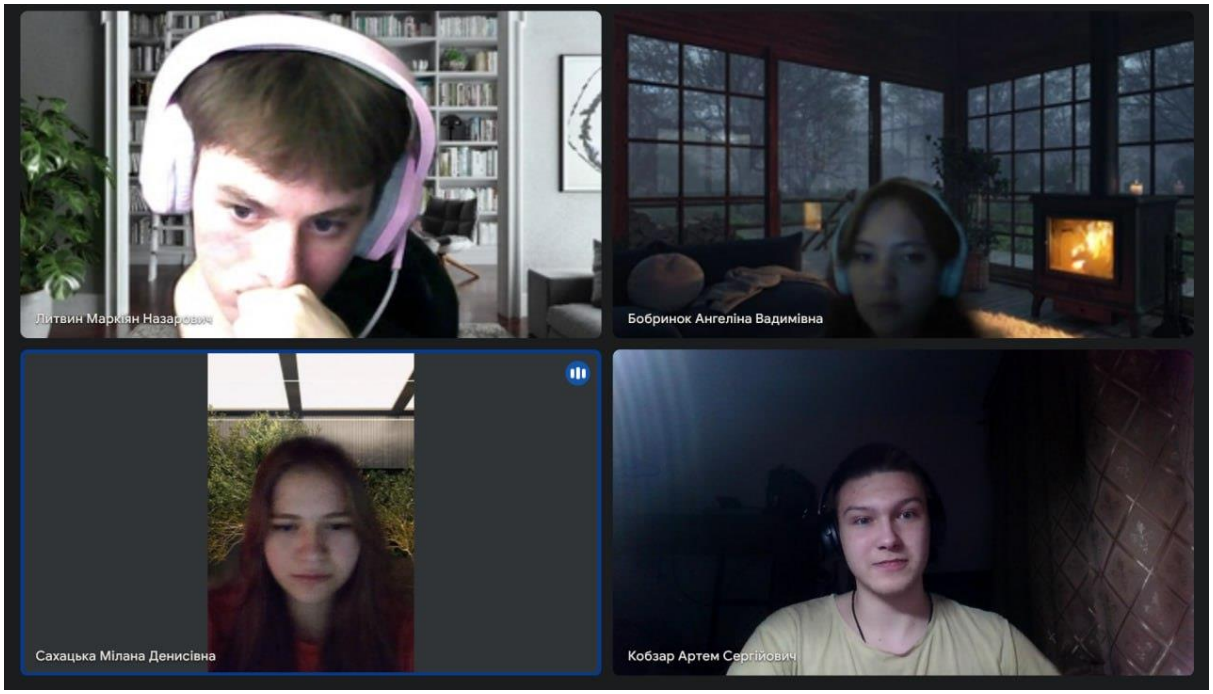
Результат:

```
3 9 1 2
8 9 8 7 6 5 4 3 2
7 8 7 6 5 4 3 2 1
```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a minute ago	C++ 23	Accepted	0.117	6.824	View

Час виконання ~ 45 хв

Зустрічі з командою:



Висновок: У цьому епіку я навчився працювати з різними видами динамічних структур, створювати чіткі і структуровані програми. Ознайомився з алгоритмами їх обробки.