

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-12

Бісюк Роман

Львів 2024

## **Тема лабораторної роботи:**

Основи динамічних структур даних: стек, черга, зв'язний список, дерево.

## **Мета лабораторної роботи:**

Метою цієї лабораторної роботи є освоєння основних принципів роботи з динамічними структурами даних, а також отримання навичок реалізації та використання таких структур, як стек, черга, зв'язний список і дерево. Я прагну зрозуміти, як динамічні структури дозволяють ефективніше використовувати пам'ять шляхом виділення ресурсів у динамічній області пам'яті (heap), а також які основні операції можна виконувати з кожною структурою. Я маю на меті навчитися обирати відповідну структуру даних для конкретної задачі, враховуючи її особливості та поведінку в пам'яті, а також зрозуміти алгоритми роботи з динамічними структурами, такі як додавання, видалення елементів і пошук. Це допоможе мені закласти основу для подальшого вивчення складніших структур даних та алгоритмів їх обробки.

## **Джерела:**

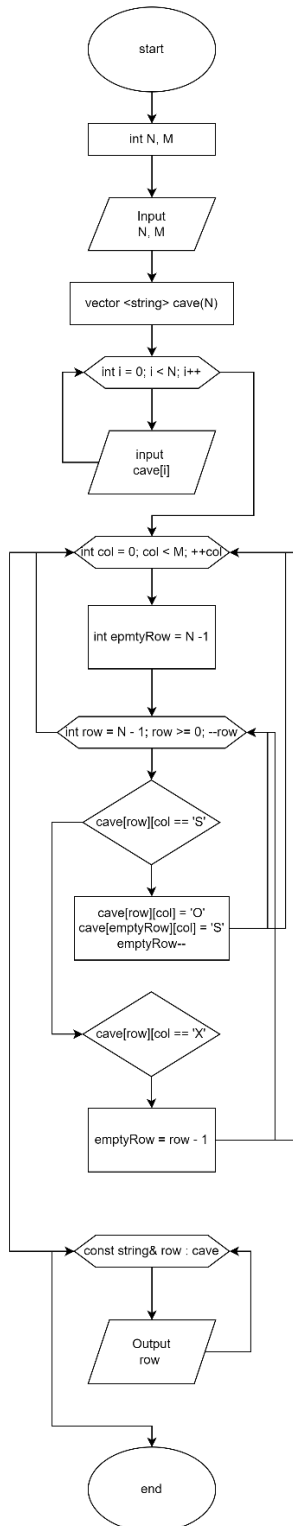
- CS50 lectures and tasks about data structures and algorithms
- University lectures
- aCode – data structures
- Google + ChatGPT for learning about different types of trees, stacks and queues with their implementations.

# Виконання

## Task 2 - Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7)

Time expected – 30m

Time spent – 30m



### Task 3 - Lab# programming: VNS Lab 10

Time expected – 2hours

Time spent – 4 hours

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

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

23.Запису в лінійному списку містять ключове поле типу `*char` (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим ключем. Додати K елементів після елемента із заданим ключем.

```

1 #include <iostream>
2 #include <string>
3 #include <cstring>
4 using namespace std;
5
6 struct Node
7 {
8     char *key;
9     Node *prev;
10    Node *next;
11 };
12
13 void add_el(Node *head, char *key)
14 {
15     Node *newNode = new Node(key, nullptr, nullptr);
16     if (head == nullptr)
17     {
18         head = newNode;
19         return;
20     }
21     Node *tail = head;
22     while (tail->next != nullptr)
23     {
24         tail = tail->next;
25     }
26     tail->next = newNode;
27     newNode->prev = tail;
28 }
29
30 Node *create_list()
31 {
32     Node *head = nullptr;
33     cout << "Enter 5 key words for list: " << endl;
34     for (int i = 0; i < 5; i++)
35     {
36         char *key = new char[256];
37         cin.getline(key, 256);
38         add_el(head, key);
39     }
40     return head;
41 }
42
43 void print_list(Node *head)
44 {
45     if (head == nullptr)
46     {
47         cout << "list is empty" << endl;
48         return;
49     }
50     Node *current = head;
51     while (current != nullptr)
52     {
53         cout << current->key << " ";
54         current = current->next;
55     }
56     cout << endl;
57 }
58
59 void delete_el(Node *head, char *key)
60 {
61     Node *current = head;
62     while (current != nullptr && strcmp(key, current->key) != 0)
63     {
64         current = current->next;
65     }
66     if (current == nullptr)
67     {
68         return;
69     }
70     if (current->prev != nullptr)
71     {
72         current->prev->next = current->next;
73     }
74     else
75     {
76         head = current->next;
77     }
78     if (current->next != nullptr)
79     {
80         current->next->prev = current->prev;
81     }
82     delete current;
83 }
84
85 void add_new(Node *head, char *target_key, int k)
86 {
87     Node *current = head;
88     while (current != nullptr && strcmp(target_key, current->key) != 0)
89     {
90         current = current->next;
91     }
92     if (current == nullptr)
93     {
94         return;
95     }
96     cout << "Enter " << k << " new key words: " << endl;
97     for (int i = 0; i < k; i++)
98     {
99         char *newkey = new char[256];
100         cin.getline(newkey, 256);
101         Node *newNode = new Node(newkey, current, current->next);
102         if (current->next != nullptr)
103         {
104             current->next->prev = newNode;
105         }
106         current->next = newNode;
107         current = newNode;
108     }
109 }
110
111 void list_to_file(Node *head, const string &name)
112 {
113     ofstream f(name);
114     Node *current = head;
115     while (current != nullptr)
116     {
117         f << current->key << endl;
118         current = current->next;
119     }
120 }
121
122 Node *list_from_file(const string &name)
123 {
124     ifstream f(name);
125     Node *head = nullptr;
126     string key;
127     while (getline(f, key))
128     {
129         char *key_ = new char[key.size() + 1];
130         strcpy(key_, key.c_str());
131         add_el(head, key_);
132     }
133     return head;
134 }
135
136 void delete_list(Node *head)
137 {
138     while (head != nullptr)
139     {
140         Node *temp = head;
141         head = head->next;
142         delete temp;
143     }
144 }
145
146 int main()
147 {
148     Node *list = create_list();
149     cout << "Initial list: " << endl;
150     print_list(list);
151
152     char *del_key = new char[256];
153     cout << "Enter key to delete: " << endl;
154     cin.getline(del_key, 256);
155     delete_el(list, del_key);
156     cout << "After deleting element: " << endl;
157     print_list(list);
158
159     int k;
160     cout << "Enter k: ";
161     cin >> k;
162     cin.ignore();
163     char *add_key = new char[256];
164     cout << "Enter key: " << endl;
165     cin.getline(add_key, 256);
166     add_new(list, add_key, k);
167     cout << "After adding k elements after key: " << endl;
168     print_list(list);
169
170     list_to_file(list, "list.txt");
171     delete_list(list);
172     cout << "After destroying the list: " << endl;
173     print_list(list);
174
175     list = list_from_file("list.txt");
176     cout << "After loading the list from file: " << endl;
177     print_list(list);
178
179     delete_list(list);
180     return 0;
181 }

```

Enter 5 key words for list:

one

two

three

four

five

Initial list:

one two three four five

Enter key to delete:

two

After deleting element:

one three four five

Enter k: 3

Enter key:

three

Enter 3 new key words:

NewWorld1

NewWorld2

NewWorld3

After adding k elements after key:

one three NewWorld1 NewWorld2 NewWorld3 four five

After destroying the list:

List is empty

After loading the list from file:

one three NewWorld1 NewWorld2 NewWorld3 four five

## Task 4 - Lab# programming: Algotester Lab 5

Time expected - 30 minutes

time spent - 45 minutes

### Lab 5v2

Limits: 1 sec., 256 MiB

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.

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a few seconds ago	C++ 23	Accepted	0.025	1.855	<a href="#">View</a>

```
1  #include <iostream>
2  #include <vector>
3
4  using namespace std;
5
6  int main(){
7      int N, M;
8      cin >> N >> M;
9      vector<string> cave(N);
10     for(int i = 0; i < N; i++){
11         cin >> cave[i];
12     }
13     for (int col = 0; col < M; ++col) {
14         int emptyRow = N - 1;
15         for (int row = N - 1; row >= 0; --row) {
16             if (cave[row][col] == 'S') {
17                 cave[row][col] = 'O';
18                 cave[emptyRow][col] = 'S';
19                 emptyRow--;
20             } else if (cave[row][col] == 'X') {
21                 emptyRow = row - 1;
22             }
23         }
24     }
25
26     for(const string& row : cave){
27         cout << row << endl;
28     }
29
30     return 0;
31 }
```

## Task 5 - Lab# programming: Algotester Lab 7-8

Time expected – 2 hours

time spent – 4 hours

### Lab 78v1

Limits: 2 sec., 256 MiB

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  $n$  - кількість елементів, яку треба видалити.

- **Визначення розміру:**

Ідентифікатор - *size*

Ви не отримуєте аргументів.

Ви виводите кількість елементів у списку.

- **Отримання значення  $i$ -го елемента**

Ідентифікатор - *get*

Ви отримуєте ціле число - *index*, індекс елемента.

Ви виводите значення елемента за індексом.

- **Модифікація значення  $i$ -го елемента**

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 `<<`

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
2 hours ago	C++ 23	Accepted	0.008	1.348	<a href="#">View</a>
2 hours ago	C++ 23	Accepted	0.008	1.414	<a href="#">View</a>





## Task 6 - Practice# programming: Class Practice Task

Time expected – 3 hours

Time spent – 5 hours

### 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  $\Rightarrow$  9  $\rightarrow$  7  $\rightarrow$  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

```
Linked List with no changes: 7 9 11 12
Reversed Linked List: 12 11 9 7
Are lists equal? No
Are lists equal after changes? No
Sum: 6 8 8
Original Tree: 2 3 5 1 4
Mirrored Tree: 4 1 5 3 2
Tree before summing subtrees: 4 2 5 1 3
Tree after: 4 9 5 5 3
```

```

1 #include <iostream>
2
3 using namespace std;
4
5 struct Node
6 {
7     int data;
8     Node *next;
9     Node(int value) : data(value), next(nullptr) {}
10 }
11
12 Node *reverse(Node *head)
13 {
14     Node *previous = nullptr;
15     Node *current = head;
16     Node *next = nullptr;
17
18     while (current != nullptr)
19     {
20         next = current->next;
21         current->next = previous;
22         previous = current;
23         current = next;
24     }
25     return previous;
26 }
27
28 bool compare(Node *N1, Node *N2)
29 {
30     while (N1 != nullptr && N2 != nullptr)
31     {
32         if (N1->data != N2->data)
33         {
34             return false;
35         }
36         N1 = N1->next;
37         N2 = N2->next;
38     }
39     return N1 == nullptr && N2 == nullptr;
40 }
41
42 Node *add(Node *N1, Node *N2)
43 {
44     Node dummy(0);
45     Node *current = &dummy;
46     int carry = 0;
47
48     while (N1 != nullptr || N2 != nullptr || carry)
49     {
50         int sum = carry;
51         if (N1 != nullptr)
52         {
53             sum += N1->data;
54             N1 = N1->next;
55         }
56         if (N2 != nullptr)
57         {
58             sum += N2->data;
59             N2 = N2->next;
60         }
61         carry = sum / 10;
62         current->next = new Node(sum % 10);
63         current = current->next;
64     }
65     return dummy->next;
66 }
67
68 struct TreeNode
69 {
70     int value;
71     TreeNode *left;
72     TreeNode *right;
73 };
74
75 TreeNode *insert(Node *root, int val)
76 {
77     if (root == nullptr)
78     {
79         return new TreeNode(val);
80     }
81     if (val < root->value)
82     {
83         root->left = insert(root->left, val);
84     }
85     if (val > root->value)
86     {
87         root->right = insert(root->right, val);
88     }
89     return root;
90 }
91
92 void treeSum(TreeNode *root)
93 {
94     if (root == nullptr)
95     {
96         return;
97     }
98     int leftSum = 0;
99     int rightSum = 0;
100     if (root->left != nullptr)
101     {
102         leftSum = treeSum(root->left);
103     }
104     if (root->right != nullptr)
105     {
106         rightSum = treeSum(root->right);
107     }
108     root->value = leftSum + rightSum + root->value;
109 }
110
111 void printList(Node *head)
112 {
113     while (head != nullptr)
114     {
115         cout << head->data << " ";
116         head = head->next;
117     }
118     cout << endl;
119 }
120
121 void printTree(TreeNode *root)
122 {
123     if (root == nullptr)
124     {
125         return;
126     }
127     printTree(root->left);
128     cout << root->value << " ";
129     printTree(root->right);
130 }
131
132 int main()
133 {
134     Node *head = new Node(7);
135     head->next = new Node(3);
136     head->next->next = new Node(11);
137     head->next->next->next = new Node(13);
138
139     cout << "Linked list after no changes: ";
140     printList(head);
141
142     head = reverse(head);
143     cout << "Reversed linked list: ";
144     printList(head);
145
146     Node *list1 = new Node(1);
147     list1->next = new Node(4);
148     list1->next->next = new Node(7);
149
150     Node *list2 = new Node(1);
151     list2->next = new Node(2);
152     list2->next->next = new Node(8);
153
154     cout << "Are lists equal: " << (compare(list1, list2) ? "yes" : "no") << endl;
155
156     list1->next->next->data = 4;
157     cout << "Are lists equal after changes: " << (compare(list1, list2) ? "yes" : "no") << endl;
158
159     Node *root = new Node(5);
160     root->next = new Node(6);
161     root->next->next = new Node(7);
162
163     Node *root2 = new Node(1);
164     root2->next = new Node(3);
165     root2->next->next = new Node(11);
166
167     Node *sum = add(root, root2);
168     cout << "sum: ";
169     printList(sum);
170
171     TreeNode *root3 = new TreeNode(1);
172     root3->left = new TreeNode(2);
173     root3->right = new TreeNode(4);
174     root3->left->left = new TreeNode(3);
175     root3->left->right = new TreeNode(5);
176
177     cout << "Original tree: ";
178     printTree(root3);
179     cout << endl;
180
181     TreeNode *root4 = create_mirror_tree(root3);
182     cout << "Mirrored tree: ";
183     printTree(root4);
184     cout << endl;
185
186     TreeNode *root5 = new TreeNode(1);
187     root5->next->left = new TreeNode(2);
188     root5->next->right = new TreeNode(3);
189     root5->next->left->left = new TreeNode(4);
190     root5->next->left->right = new TreeNode(5);
191
192     cout << "Tree before summing subtrees: ";
193     printTree(root5);
194     cout << endl;
195
196     treeSum(root5);
197     cout << "Tree after summing subtrees: ";
198     printTree(root5);
199     cout << endl;
200
201     return 0;
202 }

```

# Task 7 - Practice# programming: Self Practice Task

time expected – 1 Hour

time spent – 50 minutes

## Lab 5v3

Limits: 1 sec., 256 MiB

У вас є карта гори розміром  $N \times M$ .

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.

Клітинки 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

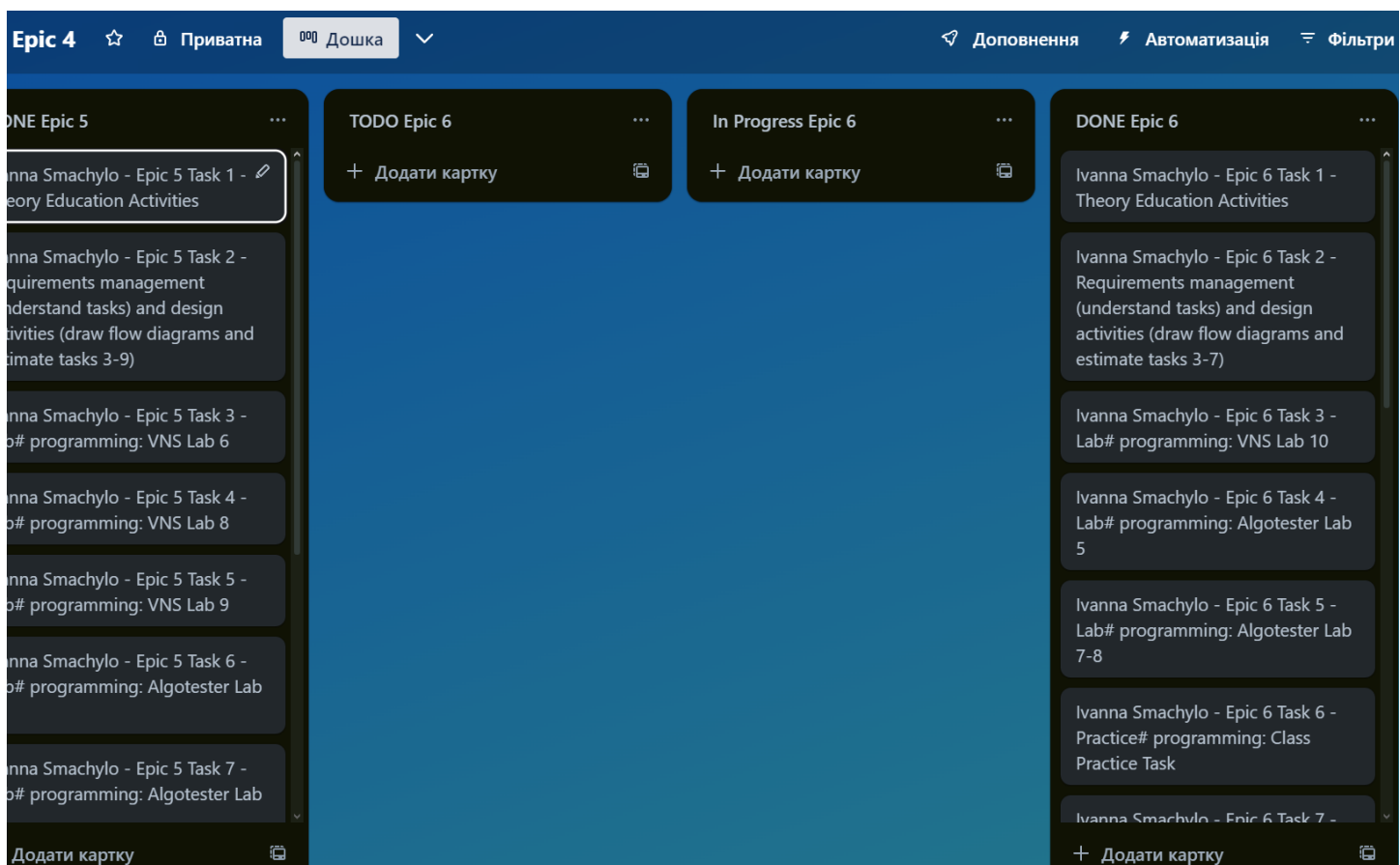
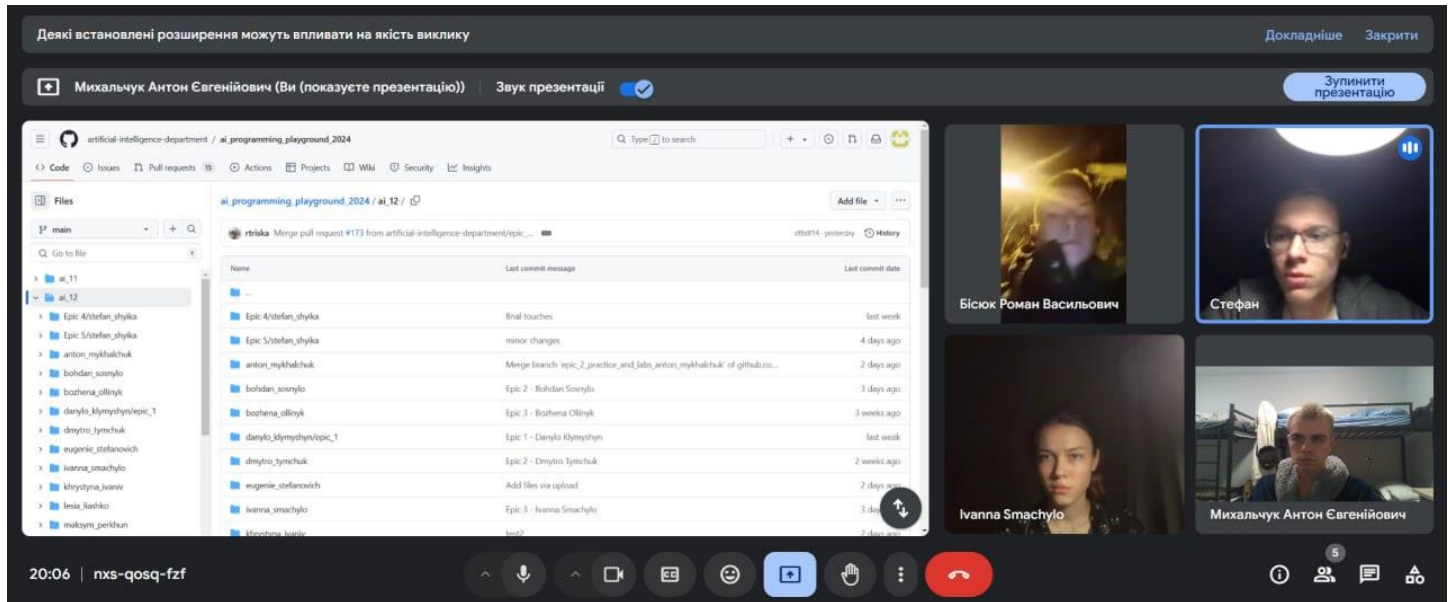
$N$  рядків по  $M$  елементів в рядку через пробіл - висоти карти.

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a minute ago	C++ 23	Accepted	0.117	6.695	<a href="#">View</a>
a minute ago	C++ 23	Wrong Answer 1	0.002	0.930	<a href="#">View</a>

```
1  #include <iostream>
2  #include <vector>
3  #include <queue>
4  #include <algorithm>
5  using namespace std;
6  int main() {
7      int N, M, x, y;
8      cin >> N >> M;
9      cin >> x >> y;
10
11     const int dx[] = { 1, -1, 0, 0 };
12     const int dy[] = { 0, 0, 1, -1 };
13
14     x--; y--;
15
16     vector<vector<int>> height(N, vector<int>(M, -1));
17     queue<pair<int, int>> q;
18     q.push({x, y});
19     height[x][y] = 0;
20
21     while (!q.empty()) {
22         auto el = q.front();
23         q.pop();
24
25         for (int i = 0; i < 4; ++i) {
26             int nx = el.first + dx[i];
27             int ny = el.second + dy[i];
28
29             if (nx >= 0 && nx < N && ny >= 0 && ny < M && height[nx][ny] == -1) {
30                 height[nx][ny] = height[el.first][el.second] + 1;
31                 q.push({ nx, ny });
32             }
33         }
34     }
35     int maxHeight = 0;
36     for (int i = 0; i < N; ++i) {
37         for (int j = 0; j < M; ++j) {
38             maxHeight = max(maxHeight, height[i][j]);
39         }
40     }
41     for (int i = 0; i < N; ++i) {
42         for (int j = 0; j < M; ++j) {
43             cout << maxHeight - height[i][j] << " ";
44         }
45         cout << endl;
46     }
47     return 0;
48 }
```

# PULL

**MEETS:** Обговорили проблемні питання пофікисили проблему з додаванням комітів саги до епиків. Оновили трелло



**Висновок:** завдяки цій роботі я на практиці зрозумів, як працюють основні динамічні структури даних та їхні ключові операції, і тепер усвідомлюю, чому їх використовують у завданнях, що вимагають гнучкого управління пам'яттю.