

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=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:**

```

1  #include <iostream>
2  #include <fstream>
3  #include <string>
4
5  using namespace std;
6
7  class Node {
8  public:
9      string data;
10     Node* next;
11     Node* prev;
12     Node(const string& data) : data(data), next(nullptr), prev(nullptr) {}
13 };
14
15 class DoublyLinkedList {
16 public:
17     Node* head;
18     Node* tail;
19     DoublyLinkedList() : head(nullptr), tail(nullptr) {}
20
21     void add_begin(const string& data) {
22         Node* new_node = new Node(data);
23         if (!head) {
24             head = tail = new_node;
25         }
26         else {
27             new_node->next = head;
28             head->prev = new_node;
29             head = new_node;
30         }
31     }
32     void add_end(const string& data) {
33         Node* new_node = new Node(data);
34         if (!tail) {
35             head = tail = new_node;
36         }
37         else {
38             tail->next = new_node;
39             new_node->prev = tail;
40             tail = new_node;
41         }
42     }
43     void delete_value(const string& value) {
44         Node* current = head;
45         while (current) {
46             if (current->data == value) {
47                 if (current->prev) current->prev->next = current->next;
48                 if (current->next) current->next->prev = current->prev;
49                 if (current == head) head = current->next;
50                 if (current == tail) tail = current->prev;
51
52                 delete current;
53                 cout << "Вузод зі значенням \" << value << "\"\" << "Видалений." << endl;
54                 return;
55             }
56             current = current->next;
57         }
58     }
59     void delete_id(int id) {
60         if (!head) {
61             cout << "Список порожній." << endl;
62             return;
63         }
64         Node* current = head;
65         int current_id = 0;
66
67         while (current && current_id < id) {
68             current = current->next;
69             current_id++;
70         }
71         if (current->prev) current->prev->next = current->next;
72         if (current->next) current->next->prev = current->prev;
73         if (current == head) head = current->next;
74         if (current == tail) tail = current->prev;
75
76         delete current;
77         cout << "Елемент з індексом " << id << " Видалений." << endl;
78     }
79     void print() {
80         if (!head) {

```

```

81         cout << "Список порожній." << endl;
82         return;
83     }
84     Node* current = head;
85     while (current) {
86         cout << current->data << " ";
87         current = current->next;
88     }
89     cout << endl;
90 }
91 void save(const string& filename) {
92     ofstream file(filename);
93     if (!file.is_open()) {
94         return;
95     }
96     Node* current = head;
97     while (current) {
98         file << current->data << endl;
99         current = current->next;
100     }
101     file.close();
102     cout << "Список збережено. \'" << filename << "\'" << endl;
103 }
104 void restore(const string& filename) {
105     ifstream file(filename);
106     if (!file.is_open()) {
107         return;
108     }
109     clear();
110     string data;
111     while (getline(file, data)) {
112         add_end(data);
113     }
114     file.close();
115     cout << "Список видалено. \'" << filename << "\'" << endl;
116 }
117 void clear() {
118     while (head) {
119         Node* temp = head;
120         head = head->next;
121         delete temp;
122     }
123     tail = nullptr;
124     cout << "Список очищений." << endl;
125 }
126 ~DoublyLinkedList() {
127     clear();
128 }
129 };
130
131 int main() {
132     DoublyLinkedList list;
133     string filename = "vns_lab_10_markiiian lytvyn.txt";
134
135     cout << "Створення списку." << endl;
136     list.add_begin("1");
137     list.add_begin("2");
138     list.add_begin("3");
139     list.add_begin("4");
140     list.add_begin("5");
141     list.add_begin("6");
142     list.add_begin("7");
143     list.add_begin("8");
144     list.add_begin("9");
145     list.add_begin("10");
146
147     cout << "Список: ";
148     list.print();
149
150     cout << "Додавання елементів на початок списку." << endl;
151     list.add_begin("11");
152     list.add_begin("12");
153     cout << "Список: ";
154     list.print();
155
156     int K;
157     cout << "Введіть індекс K:";
158     cin >> K;
159     cout << "Видалення елементів з індексом K." << endl;
160     list.delete_id(K);
161     cout << "Список: ";

```

```

162     cout << "Список: ";
163     list.print();
164
165
166     cout << "Видалення вузла зі значенням \"4\"." << endl;
167     list.delete_value("4");
168     cout << "Список: ";
169     list.print();
170
171     list.save(filename);
172     list.clear();
173     cout << "Список після очищення: ";
174     list.print();
175
176     list.restore(filename);
177     cout << "Список після відновлення: ";
178     list.print();
179
180     list.clear();
181     cout << "Список після остаточного очищення: ";
182     list.print();
183
184     return 0;
185
186

```

**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_markian_lytvyn.txt"
Список очищений.
Список після очищення: Список порожній.
Список очищений.
Список видалено. "vns_lab_10_markian_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:

```
1  √ #include <iostream>
2    #include <vector>
3    #include <string>
4
5    using namespace std;
6
7  √ int main() {
8      int N, M;
9      cin >> N >> M;
10     vector<vector<char>> arr(N, vector<char>(M));
11
12     for (int i = 0; i < N; ++i) {
13         for (int j = 0; j < M; ++j) {
14             cin >> arr[i][j];
15         }
16     }
17     for (int j = 0; j < M; ++j) {
18         int row = N - 1;
19         for (int i = N - 1; i >= 0; --i) {
20             if (arr[i][j] == 'X') {
21                 row = i - 1;
22             } else if (arr[i][j] == 'S') {
23                 arr[i][j] = '0';
24                 arr[row][j] = 'S';
25                 row--;
26             }
27         }
28     }
29
30     for (int i = 0; i < N; ++i) {
31         for (int j = 0; j < M; ++j) {
32             cout << arr[i][j];
33         }
34         cout << endl;
35     }
36     return 0;
37 }
38
```

### Результат:

```
5 5
SSOSS
00000
SSOXX
00005
00500
00000
00055
000XX
SS000
SSS05
PS D:\Epics> |
```

Created	Compiler	Result	Time (sec.)	Memory (MiB)	Actions
a few seconds ago	C++ 23	Accepted	0.071	2.250	<a href="#">View</a>

Час виконання ~ 30 хв

### Завдання 3: Algotester Lab 78v1

#### Умова:

Ваше завдання - власноруч реалізувати структуру даних "Двоzv'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 елемента**  
Ідентифікатор - getget  
Ви отримуєте ціле число - indexindex, індекс елемента.  
Ви виводите значення елемента за індексом.

- **Модифікація значення іі-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:

```

1 #include <iostream>
2 #include <vector>
3 #include <string>
4 using namespace std;
5
6 template <typename T>
7 class Node {
8 public:
9     T data;
10    Node* prev;
11    Node* next;
12
13    Node(T value) : data(value), prev(nullptr), next(nullptr) {}
14 };
15
16 template <typename T>
17 class DoubleLinkedList {
18 private:
19     Node<T>* head;
20     Node<T>* tail;
21     size_t list_size;
22
23 public:
24     DoubleLinkedList() : head(nullptr), tail(nullptr), list_size(0) {}
25
26     ~DoubleLinkedList() {
27         while (head) {
28             Node<T>* temp = head;
29             head = head->next;
30             delete temp;
31         }
32     }
33
34     void insert(int index, const vector<T>& values) {
35         Node<T>* current = (index == list_size) ? nullptr : getNodeAt(index);
36
37         for (const T& value : values) {
38             Node<T>* new_node = new Node<T>(value);
39
40             if (!current) {
41                 if (!tail) {
42                     head = tail = new_node;
43                 } else {
44                     tail->next = new_node;
45                     new_node->prev = tail;
46                     tail = new_node;
47                 }
48             } else if (current->prev) {
49                 new_node->prev = current->prev;
50                 new_node->next = current;
51                 current->prev = new_node;
52             }
53         }
54         list_size += values.size();
55     }
56
57     Node<T>* getNodeAt(int index) const {
58         if (index < 0 || index > list_size) return nullptr;
59         if (index < list_size / 2) {
60             Node<T>* current = head;
61             for (int i = 0; i < index; i++) current = current->next;
62         } else {
63             Node<T>* current = tail;
64             for (int i = list_size - 1 - index; i > 0; i--) current = current->prev;
65         }
66         return current;
67     }
68
69     int size() const { return list_size; }
70 };
71
72 int main() {
73     int qq;
74     cin >> qq;
75
76     DoubleLinkedList<int> list;
77
78     while (qq--) {
79         string op;
80         cin >> op;
81
82         if (op == "insert") {
83             int index;
84             vector<int> values;
85             int value;
86             while (value != -1) {
87                 cin >> value;
88                 values.push_back(value);
89             }
90             list.insert(index, values);
91         } else if (op == "print") {
92             list.print();
93         }
94     }
95
96     return 0;
97 }

```



```

50         head->prev = new_node;
51         head = new_node;
52     } else {
53         Node<T>* prev_node = current->prev;
54         prev_node->next = new_node;
55         new_node->prev = prev_node;
56         new_node->next = current;
57         current->prev = new_node;
58     }
59     ++list_size;
60 }
61 }
62
63 void erase(int index, int amount) {
64     Node<T>* current = getNodeAt(index);
65
66     for (int i = 0; i < amount; ++i) {
67         Node<T>* del = current;
68         current = current->next;
69
70         if (del->prev) {
71             del->prev->next = del->next;
72         } else {
73             head = del->next;
74         }
75
76         if (del->next) {
77             del->next->prev = del->prev;
78         } else {
79             tail = del->prev;
80         }
81
82         delete del;
83         --list_size;
84     }
85 }
86
87 size_t size() const {
88     return list_size;
89 }
90
91 T get(int index) const {
92     return getNodeAt(index)->data;
93 }
94
95 void set(int index, const T& value) {
96     getNodeAt(index)->data = value;
97 }
98
99 void print() const {
100     Node<T>* current = head;
101     while (current) {
102         cout << current->data << " ";
103         current = current->next;
104     }
105     cout << endl;
106 }
107
108 private:
109     Node<T>* getNodeAt(int index) const {
110         Node<T>* current = (index < list_size / 2) ? head : tail;
111         if (index < list_size / 2) {
112             for (int i = 0; i < index; ++i) {
113                 current = current->next;
114             }
115         } else {
116             for (int i = list_size - 1; i > index; --i) {
117                 current = current->prev;
118             }
119         }
120         return current;
121     }
122 };
123
124 int main() {
125     DoubleLinkedList<int> list;
126     int Q;
127     cin >> Q;
128
129     while (Q-- > 0) {
130         string answer;
131         cin >> answer;
132
133         if (answer == "insert") {
134             int index, N;
135             cin >> index >> N;
136             insert(index, value/N);

```

```

136     vector<int> values(N);
137     for (int i = 0; i < N; ++i) {
138         cin >> values[i];
139     }
140     list.insert(index, values);
141 }
142 else if (answer == "erase") {
143     int index, count;
144     cin >> index >> count;
145     list.erase(index, count);
146 }
147 else if (answer == "size") {
148     cout << list.size() << endl;
149 }
150 else if (answer == "get") {
151     int index;
152     cin >> index;
153     cout << list.get(index) << endl;
154 }
155 else if (answer == "set") {
156     int index, value;
157     cin >> index >> value;
158     list.set(index, value);
159 }
160 else if (answer == "print") {
161     list.print();
162 }
163 }
164
165 return 0;
166 }

```

**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	<a href="#">View</a>

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 списку (напр. 379  $\Rightarrow$  9 $\rightarrow$ 7 $\rightarrow$ 3);
- функція повертає новий список, передані в функцію списки не модифікуються.

## Розв'язок:

```

1  #include <iostream>
2  using namespace std;
3
4  struct Node {
5      int data;
6      Node* next;
7
8      Node(int val) : data(val), next(NULL) {}
9  };
10
11
12  // Task 1
13  Node* reverse(Node* head) {
14      Node* prev = NULL;
15      Node* current = head;
16      Node* next = NULL;
17
18      while (current != NULL) {
19          next = current->next;
20          current->next = prev;
21          prev = current;
22          current = next;
23      }
24
25      return prev;
26  }
27
28  void printList(Node* head) {
29      while (head != NULL) {
30          cout << head->data << " -> ";
31          head = head->next;
32      }
33      cout << endl;
34  }
35
36  // Task 2
37  bool compare(Node* h1, Node* h2) {
38      while (h1 != NULL && h2 != NULL) {
39          if (h1->data != h2->data)
40              return false;
41          h1 = h1->next;
42          h2 = h2->next;
43      }
44      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);
    }

```

```

86
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:

```
1  #include <iostream>
2  #include <algorithm>
3  using namespace std;
4
5  template <typename T>
6  class BinaryTree {
7  private:
8      struct Node {
9          T data;
10         Node* left;
11         Node* right;
12
13         Node(T value) : data(value), left(nullptr), right(nullptr) {}
14     };
15     Node* root;
16     size_t size(const Node* node) const {
17         return node == nullptr ? 0 : 1 + size(node->left) + size(node->right);
18     }
19     void mirror_n(Node* node) {
20         if (node == nullptr)
21             return;
22
23         swap(node->left, node->right);
24         mirror_n(node->left);
25         mirror_n(node->right);
26     }
27     T sum(Node* node) const {
28         if (node == nullptr)
29             return 0;
30
31         T left_sum = sum(node->left);
32         T right_sum = sum(node->right);
33
34         return node->data + left_sum + right_sum;
35     }
36     void convert(Node* node) {
37         if (node == nullptr)
38             return;
39
40         T left_sum = node->left ? sum(node->left) : 0;
41         T right_sum = node->right ? sum(node->right) : 0;
42         node->data = left_sum + right_sum;
43
44         convert(node->left);
45         convert(node->right);
46     }
47     Node* insert_n(Node* node, T value) {
48         if (node == nullptr)
49             return new Node(value);
```

```

49         return new Node(value);
50
51         if (value < node->data)
52             node->left = insert_n(node->left, value);
53         else if (value > node->data)
54             node->right = insert_n(node->right, value);
55         return node;
56     }
57     bool find_n(const Node* node, const T& value) const {
58         if (node == nullptr)
59             return false;
60
61         if (node->data == value)
62             return true;
63         else if (value < node->data)
64             return find_n(node->left, value);
65         else
66             return find_n(node->right, value);
67     }
68     void print(const Node* node) const {
69         if (node == nullptr)
70             return;
71
72         print(node->left);
73         cout << node->data << " ";
74         print(node->right);
75     }
76
77 public:
78     BinaryTree() : root(nullptr) {}
79     ~BinaryTree() {
80         clear(root);
81     }
82     void clear(Node* node) {
83         if (node == nullptr)
84             return;
85
86         clear(node->left);
87         clear(node->right);
88         delete node;
89     }
90     size_t size() const {
91         return size(root);
92     }
93     void insert(T value) {
94         root = insert_n(root, value);
95     }
96     bool find(const T& value) const {
97         return find_n(root, value);
98     }
99     void mirror() {
100         mirror_n(root);
101     }
102     T sum_tree() const {
103         return sum(root);
104     }
105     void convert() {
106         convert(root);
107     }
108     void print() const {
109         print(root);
110         cout << endl;
111     }
112 };
113
114 int main() {
115     BinaryTree<int> tree;
116
117     int values[] = {4, 1, 2, 3, 5, 6, 0};
118     for (int value : values)
119         tree.insert(value);
120
121     cout << "Дерево: ";
122     tree.print();
123
124     cout << "Розмір дерева: " << tree.size() << endl;
125
126     cout << "Сума вузлів: " << tree.sum_tree() << endl;
127
128     tree.convert();
129     cout << "Дерево після перетворення: ";
130     tree.print();
131
132     tree.mirror();
133     cout << "Віддзеркалене дерево: ";
134     tree.print();
135
136     return 0;
137 }

```

## Результат:

```
Дерево: 0 1 2 3 4 5 6
Розмір дерева: 7
Сума вузлів: 21
Дерево після перетворення: 0 5 3 0 17 6 0
Віддзеркалене дерево: 0 6 17 0 3 5 0
PS D:\Epic>
```

Час виконання ~ 2.5 години

## Завдання 6: Self Practice Work

### Умова:

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

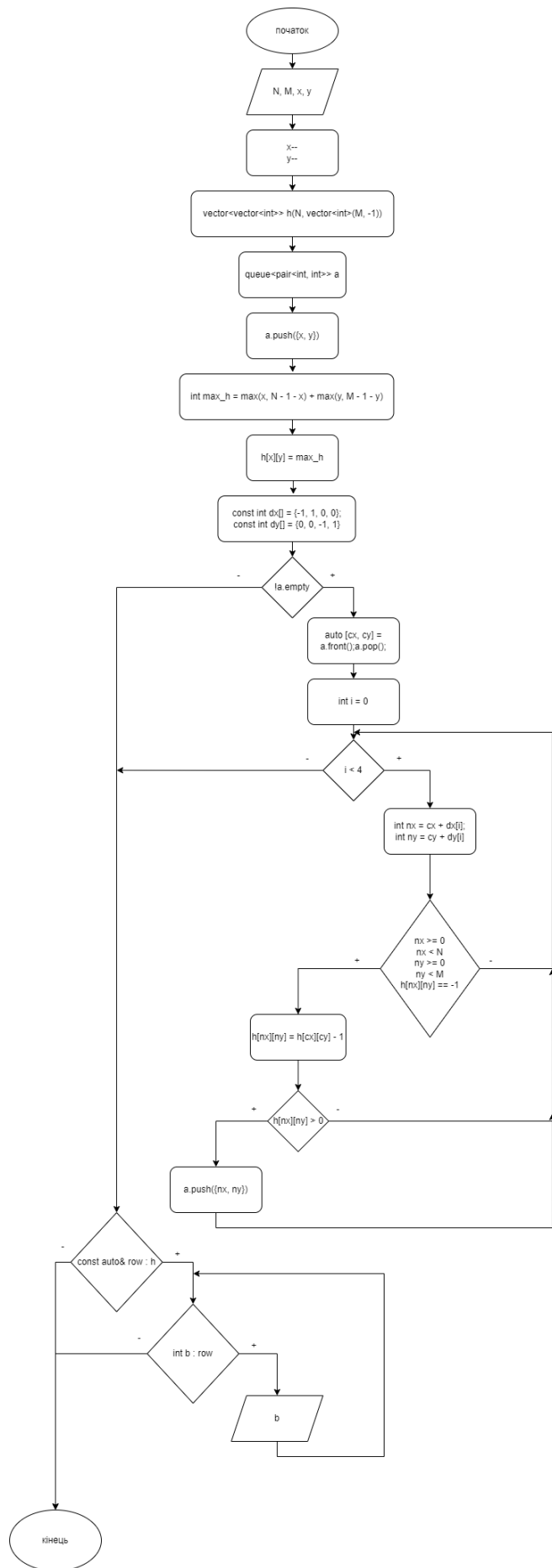
У першому рядку 2 числа  $NN$  та  $MM$  - розміри карти

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:





## Розв'язок:

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

## 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	<a href="#">View</a>

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.