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
Силіч Богдан

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:

1. Основи Динамічних Структур Даних:
    - Вступ до динамічних структур даних: визначення та важливість
    - Виділення пам'яті для структур даних (stack і heap)
    - Приклади простих динамічних структур: динамічний масив
2. Стек:
    - Визначення та властивості стеку
    - Операції push, pop, top: реалізація та використання
    - Приклади використання стеку: обернений польський запис, перевірка балансу дужок
    - Переповнення стеку
3. Черга:
    - Визначення та властивості черги
    - Операції enqueue, dequeue, front: реалізація та застосування
    - Приклади використання черги: обробка подій, алгоритми планування
    - Розширення функціоналу черги: пріоритетні черги
4. Зв'язні Списки:
    - Визначення однозв'язного та двозв'язного списку
    - Принципи створення нових вузлів, вставка між існуючими, видалення, створення кільця(circular linked list)
    - Основні операції: обхід списку, пошук, доступ до елементів, об'єднання списків
    - Приклади використання списків: управління пам'яттю, FIFO та LIFO структури
5. Дерева:
    - Вступ до структури даних "дерево": визначення, типи

- Бінарні дерева: вставка, пошук, видалення
- Обхід дерева: в глибину (preorder, inorder, postorder), в ширину
- Застосування дерев: дерева рішень, хеш-таблиці
- Складніші приклади дерев: AVL, Червоно-чорне дерево

6. Алгоритми Обробки Динамічних Структур:
   - Основи алгоритмічних патернів: ітеративні, рекурсивні
   - Алгоритми пошуку, сортування даних, додавання та видалення елементів

Індивідуальний план опрацювання теорії:

Основи Динамічних Структур Даних
Стек
Черга
Зв'язні Списки
Дерева
Алгоритми Обробки Динамічних Структур

Джерела:

- Chat gpt
- Список відтворення на YouTube (https://youtube.com/playlist?list=PLiPRE8VmJzOpn6PzYf0higmCEyGzo2A5g&si=sXvmPdnGkwvJLXUi)
- Лекції та практичні

Виконання роботи:

VNS Lab 10:

Порядок виконання роботи

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.

6. Записи в лінійному списку містять ключове поле типу int. Сформувати двонаправлений список. Знищити з нього елемент із заданим номером, додати елемент у початок списку.

Algotester Lab 5v3:

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.

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.

Вхідні дані

У першому рядку 2 числа N та M - розміри карти

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

Вихідні дані

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

Обмеження

$1 \leq N, M \leq 10^3$

$1 \leq x \leq N$

$1 \leq y \leq M$

Algotester Lab 7-8 v3:

Ваше завдання - власноруч реалізувати структуру даних "Двійкове дерево пошуку".
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його параметри.

Вам будуть поступати запити такого типу:

- **Вставка**:
  Ідентифікатор - $insert$
  Ви отримуєте ціле число $value$ - число, яке треба вставити в дерево.
- **Пошук**:
  Ідентифікатор - $contains$
  Ви отримуєте ціле число $value$ - число, наявність якого у дереві необхідно перевірити.
  Якщо $value$ наявне в дереві - ви виводите Yes, у іншому випадку No.
- **Визначення розміру**:
  Ідентифікатор - $size$
  Ви не отримуєте аргументів.
  Ви виводите кількість елементів у дереві.
- **Вивід дерева на екран**
  Ідентифікатор - $print$
  Ви не отримуєте аргументів.
  Ви виводите усі елементи дерева через пробіл.
  Реалізувати використовуючи перегрузку оператора <<

Вхідні дані

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

Вихідні дані

Відповіді на запити у зазначеному в умові форматі.

Обмеження

$0 \leq Q \leq 10^3$
$0 \leq t_i \leq 10^3$

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. $379 \implies 9 \rightarrow 7 \rightarrow 3$);
- 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

Self Practice Task (5 v1):

Обмеження: 2 сек., 256 МБ

У світі Атод сестри Ліна і Рілай люблять грати у гру. У них є дошка із 8-ми рядків і 8-ми стовпців. На перетині i-го рядка і j-го стовпця лежить магічна куля, яка може світитись магічним світлом (тобто у них є 64 кулі). На початку гри деякі кулі світяться, а деякі ні... Далі вони обирають N куль і для кожної читають магічне заклинання, після чого всі кулі, які лежать на перетині стовпця і рядка обраної кулі міняють свій стан (ті що світяться - гаснуть, ті, що не світяться - загораються).

Також вони вирішили трохи Вам допомогти і придумали спосіб як записати стан дошки одним числом a із 8-ми байт, а саме (див. Примітки):

- Молодший байт задає перший рядок матриці;
- Молодший біт задає перший стовпець рядку;
- Значення біту каже світиться куля чи ні (0 - ні, 1 - так);

Тепер їх цікавить яким буде стан дошки після виконання N заклинань і вони дуже просять Вас їм допомогти.

Вхідні дані

У першому рядку одне число a - поточний стан дошки.

У другому рядку N - кількість заклинань.

У наступних N рядках по 2 числа R_i, C_i - рядок і стовпець кулі над якою виконується заклинання.

Вихідні дані

Одне число b - стан дошки після виконання N заклинань.

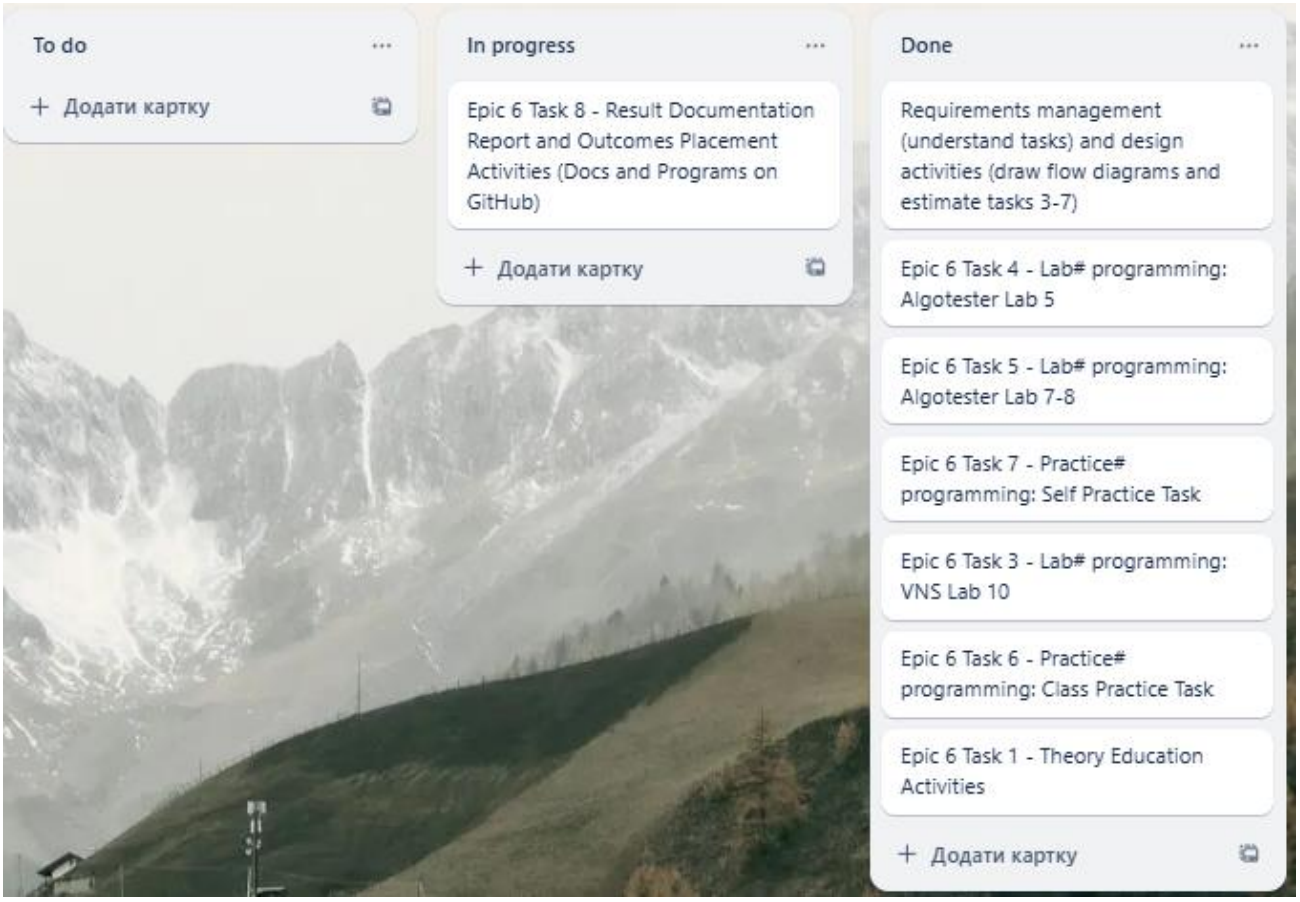
Обмеження

$0 \le N \le 10^3$

$1 \le R_i, C_i \le 8$

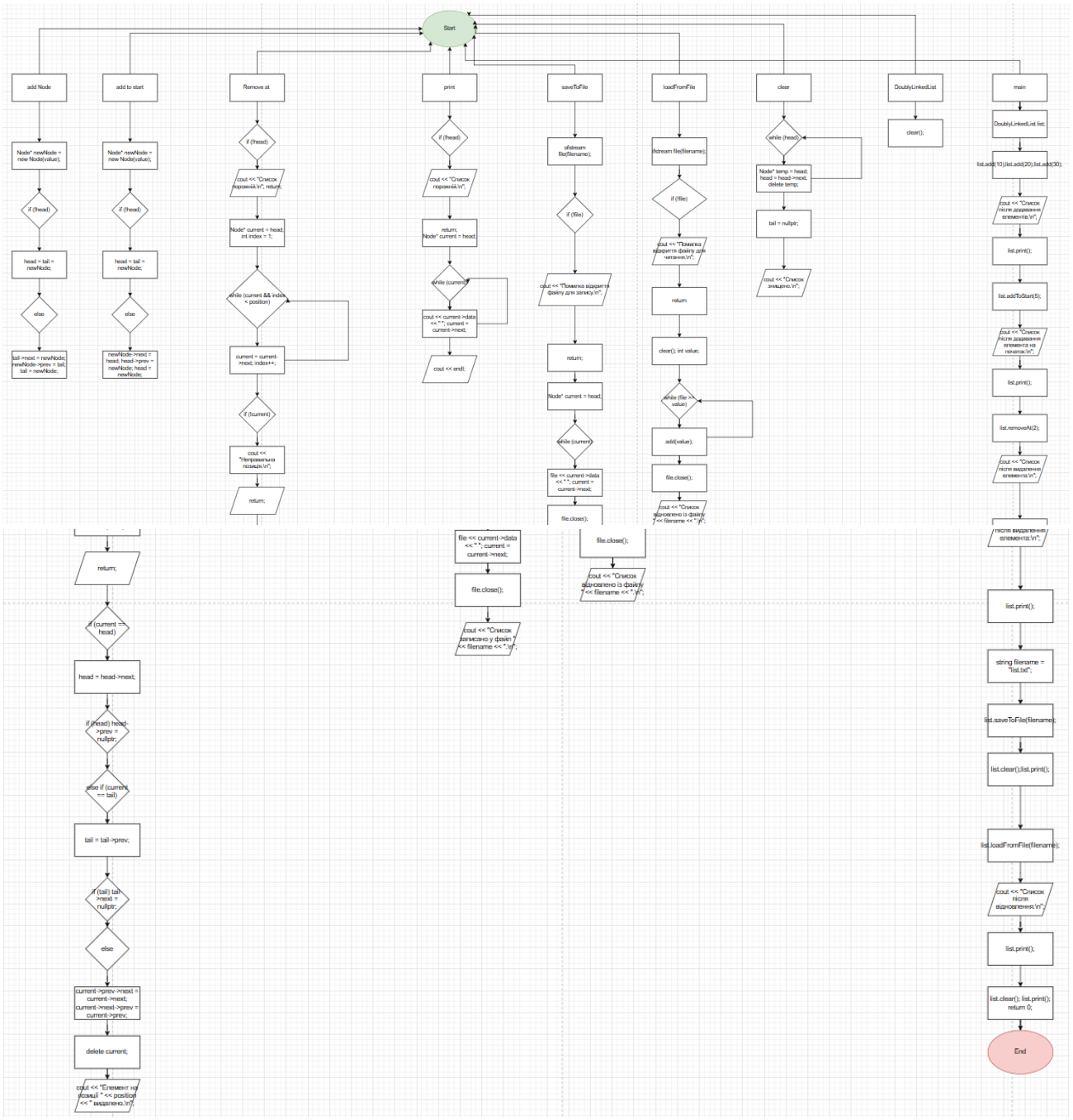
$0 \le a, b < 2^{64}$

Team Trello dashboard



Team meeting

VNS Lab 10 затратність ~3год


Start
add Node
Node* newNode = new Node(value);
if (!head)
head = tail = newNode;
else
tail->next = newNode; newNode->prev = tail; tail = newNode;
add to start
Node* newNode = new Node(value);
if (!head)
head = tail = newNode;
else
newNode->next = head; head->prev = newNode; head = newNode;
Remove at
if (!head)
cout << "Список порожній.\n"; return;
Node* current = head; int index = 1;
while (current && index < position)
current = current->next; index++;
if (!current)
cout << "Неправильна позиція.\n";
return;
return;
current->prev->next = current->next; current->next->prev = current->prev;
delete current;
cout << "Елемент на позиції " << position << " видалено.\n";
print
if (!head)
cout << "Список порожній.\n";
return;
Node* current = head;
while (current)
cout << current->data << " "; current = current->next;
cout << endl;
saveToFile
ofstream file(filename);
if (!file)
cout << "Помилка відкриття файлу для запису.\n";
return;
Node* current = head;
while (current)
file << current->data << " "; current = current->next;
file.close();
cout << "Список записано у файл " << filename << ".\n";
loadFromFile
ifstream file(filename);
if (!file)
cout << "Помилка відкриття файлу для читання.\n";
return
clear(); int value;
while (file >> value)
add(value);
file.close();
cout << "Список відновлено із файлу " << filename << ".\n";
clear
while (head)
Node* temp = head; head = head->next; delete temp;
tail = nullptr;
cout << "Список знищено.\n";
DoublyLinkedList
clear();
main
DoublyLinkedList list;
list.add(10);list.add(20);list.add(30);
cout << "Список після додавання елементів:\n";
list.print();
list.addToStart(5);
cout << "Список після додавання елемента на початок:\n";
list.print();
list.removeAt(2);
cout << "Список після видалення елемента:\n";
list.print();
string filename = "list.txt";
list.saveToFile(filename);
list.clear();list.print();
list.loadFromFile(filename);
cout << "Список після відновлення:\n";
list.print();
list.clear(); list.print(); return 0;
End

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

struct Node {
    int data;
    Node* next;
    Node* prev;

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

class DoublyLinkedList {
private:
    Node* head;
    Node* tail;

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

    void add(int value) {
        Node* newNode = new Node(value);
        if (!head) {
            head = tail = newNode;
        } else {
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }

    void addToStart(int value) {
        Node* newNode = new Node(value);
        if (!head) {
            head = tail = newNode;
        } else {
            newNode->next = head;
            head->prev = newNode;
            head = newNode;
        }
    }

    void removeAt(int position) {
        if (!head) {
            cout << "Список порожній.\n";
            return;
        }
        Node* current = head;
        int index = 1;

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

        if (!current) {
            cout << "Неправильна позиція.\n";
            return;
        }
```

```
    if (current == head) {
        head = head->next;
        if (head) head->prev = nullptr;
    } else if (current == tail) {
        tail = tail->prev;
        if (tail) tail->next = nullptr;
    } else {
        current->prev->next = current->next;
        current->next->prev = current->prev;
    }
    delete current;
    cout << "Елемент на позиції " << position << " видалено.\n";
}

void print() {
    if (!head) {
        cout << "Список порожній.\n";
        return;
    }
    Node* current = head;
    while (current) {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;
}

void saveToFile(const string& filename) {
    ofstream file(filename);
    if (!file) {
        cout << "Помилка відкриття файлу для запису.\n";
        return;
    }
    Node* current = head;
    while (current) {
        file << current->data << " ";
        current = current->next;
    }
    file.close();
    cout << "Список записано у файл " << filename << ".\n";
}

void loadFromFile(const string& filename) {
    ifstream file(filename);
    if (!file) {
        cout << "Помилка відкриття файлу для читання.\n";
        return;
    }
    clear();
    int value;
    while (file >> value) {
        add(value);
    }
    file.close();
    cout << "Список відновлено із файлу " << filename << ".\n";
}
```

```
    void clear() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        cout << "Список знищено.\n";
    }

    ~DoublyLinkedList() {
        clear();
    }
};

int main() {
    DoublyLinkedList list;

    list.add(10);
    list.add(20);
    list.add(30);
    cout << "Список після додавання елементів:\n";
    list.print();

    list.addToStart(5);
    cout << "Список після додавання елемента на початок:\n";
    list.print();

    list.removeAt(2);
    cout << "Список після видалення елемента:\n";
    list.print();

    string filename = "list.txt";
    list.saveToFile(filename);

    list.clear();
    list.print();

    list.loadFromFile(filename);
    cout << "Список після відновлення:\n";
    list.print();

    list.clear();
    list.print();

    return 0;
}
```

```
Список після додавання елементів:
10 20 30
Список після додавання елемента на початок:
5 10 20 30
Елемент на позиції 2 видалено.
Список після видалення елемента:
5 20 30
Список записано у файл list.txt.
Список знищено.
Список порожній.
Список знищено.
Список відновлено із файлу list.txt.
Список після відновлення:
5 20 30
Список знищено.
Список порожній.
Список знищено.
```

Algotester Lab 5v3 затратність ~1год

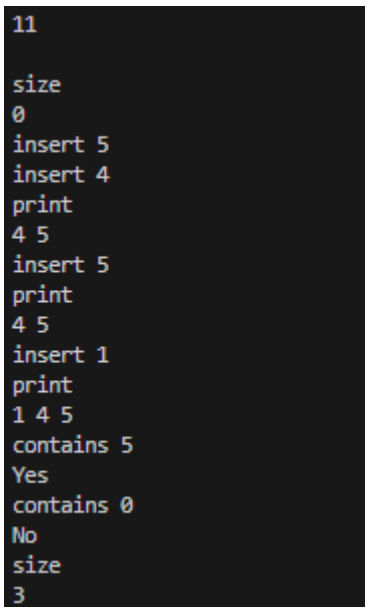
```cpp
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

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

    const int dx[] = { 1, -1, 0, 0 };
    const int dy[] = { 0, 0, 1, -1 };

    x--; y--;

    vector<vector<int>> height(N, vector<int>(M, -1));
    queue<pair<int, int>> q;
    q.push({x, y});
    height[x][y] = 0;

    while (!q.empty()) {
        auto el = q.front();
        q.pop();

        for (int i = 0; i < 4; ++i) {
            int nx = el.first + dx[i];
            int ny = el.second + dy[i];

            if (nx >= 0 && nx < N && ny >= 0 && ny < M && height[nx][ny] == -1) {
                height[nx][ny] = height[el.first][el.second] + 1;
                q.push({ nx, ny });
            }
        }
    }
    int maxHeight = 0;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            maxHeight = max(maxHeight, height[i][j]);
        }
    }
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < M; ++j) {
            cout << maxHeight - height[i][j] << " ";
        }
        cout << endl;
    }
    return 0;
}
```

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

Algotester Lab 7-8 v3 затратність ~2год

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

template <typename T>
class BinarySearchTree {
private:
    struct Node {
        T value;
        Node* left;
        Node* right;
        Node(T val) : value(val), left(nullptr), right(nullptr) {}
    };

    Node* root;

    Node* insert(Node* node, T value) {
        if (!node) return new Node(value);
        if (value < node->value)
            node->left = insert(node->left, value);
        else if (value > node->value)
            node->right = insert(node->right, value);
        return node;
    }

    bool contains(Node* node, T value) {
        if (!node) return false;
        if (value == node->value) return true;
        if (value < node->value)
            return contains(node->left, value);
        return contains(node->right, value);
    }

    void print(Node* node) {
        if (!node) return;
        print(node->left);
        cout << node->value << " ";
        print(node->right);
    }

    void clear(Node* node) {
        if (!node) return;
        clear(node->left);
        clear(node->right);
        delete node;
    }

    int size(Node* node) {
        if (!node) return 0;
        return 1 + size(node->left) + size(node->right);
    }

public:
    BinarySearchTree() : root(nullptr) {}

    ~BinarySearchTree() {
        clear(root);
    }
```

```
    ~BinarySearchTree() {
        clear(root);
    }

    void insert(T value) {
        root = insert(root, value);
    }

    bool contains(T value) {
        return contains(root, value);
    }

    void print() {
        print(root);
        cout << endl;
    }

    int size() {
        return size(root);
    }

    friend ostream& operator<<(ostream& os, BinarySearchTree& tree) {
        tree.print();
        return os;
    }
};

int main() {
    BinarySearchTree<int> bst;
    int Q;
    cin >> Q;

    while (Q--) {
        string command;
        cin >> command;
        if (command == "insert") {
            int value;
            cin >> value;
            bst.insert(value);
        } else if (command == "contains") {
            int value;
            cin >> value;
            cout << (bst.contains(value) ? "Yes" : "No") << endl;
        } else if (command == "size") {
            cout << bst.size() << endl;
        } else if (command == "print") {
            bst.print();
        }
    }

    return 0;
}
```



Class Practice Task затратність ~2.5год

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

struct Node {
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

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

int main() {
    Node* h1 = new Node(1);
    h1->next = new Node(2);
    h1->next->next = new Node(3);

    Node* h2 = new Node(1);
    h2->next = new Node(2);
    h2->next->next = new Node(3);

    cout << (compare(h1, h2) ? "Lists are equal" : "Lists are not equal") << endl;

    h2->next->next->data = 4;
    cout << (compare(h1, h2) ? "Lists are equal" : "Lists are not equal") << endl;

    return 0;
}
```

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

struct Node {
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

Node* add(Node* n1, Node* n2) {
    Node* dummy = new Node(0);
    Node* current = dummy;
    int carry = 0;

    while (n1 || n2 || carry) {
        int sum = carry;
        if (n1) {
            sum += n1->data;
            n1 = n1->next;
        }
        if (n2) {
            sum += n2->data;
            n2 = n2->next;
        }
        carry = sum / 10;
        current->next = new Node(sum % 10);
        current = current->next;
    }

    return dummy->next;
}

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

int main() {
    Node* n1 = new Node(9);
    n1->next = new Node(7);
    n1->next->next = new Node(3);

    Node* n2 = new Node(6);
    n2->next = new Node(8);

    Node* result = add(n1, n2);

    printList(result);

    return 0;
}
```

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

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int value) : val(value), left(nullptr), right(nullptr) {}
};

TreeNode* create_mirror_flip(TreeNode* root) {
    if (!root) return nullptr;
    TreeNode* mirrored = new TreeNode(root->val);
    mirrored->left = create_mirror_flip(root->right);
    mirrored->right = create_mirror_flip(root->left);
    return mirrored;
}

void printTree(TreeNode* root) {
    if (!root) return;
    printTree(root->left);
    cout << root->val << " ";
    printTree(root->right);
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);

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

    TreeNode* mirrored = create_mirror_flip(root);

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

    return 0;
}
```

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

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode(int value) : val(value), left(nullptr), right(nullptr) {}
};

int tree_sum(TreeNode* root) {
    if (!root) return 0;
    if (!root->left && !root->right) return root->val;

    int leftSum = tree_sum(root->left);
    int rightSum = tree_sum(root->right);
    root->val = leftSum + rightSum;
    return root->val + leftSum + rightSum;
}

void printTree(TreeNode* root) {
    if (!root) return;
    printTree(root->left);
    cout << root->val << " ";
    printTree(root->right);
}

int main() {
    TreeNode* root = new TreeNode(1);
    root->left = new TreeNode(2);
    root->right = new TreeNode(3);
    root->left->left = new TreeNode(4);
    root->left->right = new TreeNode(5);

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

    tree_sum(root);

    cout << "Tree with sums: ";
    printTree(root);
    cout << endl;

    return 0;
}
```

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

struct Node {
    int data;
    Node* next;
    Node(int value) : data(value), next(nullptr) {}
};

Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* current = head;
    while (current) {
        Node* next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    return prev;
}

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

int main() {
    Node* head = new Node(1);
    head->next = new Node(2);
    head->next->next = new Node(3);
    head->next->next->next = new Node(4);

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

    head = reverse(head);

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

    return 0;
}
```

Self Practice Task затратність ~1год

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

int main() {
    uint64_t a;
    int N;
    cin >> a >> N;

    vector<pair<int, int>> spells(N);
    for (int i = 0; i < N; ++i) {
        int Ri, Ci;
        cin >> Ri >> Ci;
        spells[i] = {Ri - 1, Ci - 1};
    }

    for (const auto& spell : spells) {
        int row = spell.first;
        int col = spell.second;


        for (int j = 0; j < 8; ++j) {
            a ^= (1ULL << (row * 8 + j));
        }


        for (int i = 0; i < 8; ++i) {
            if (i != row) {
                a ^= (1ULL << (i * 8 + col));
            }
        }
    }

    cout << a << endl;

    return 0;
}
```

Висновки:

Я навчився застосовувати динамічні структури для ефективного зберігання та обробки даних в програмах. Також отримав розуміння алгоритмів для роботи з чергою, стеком, списками та деревами, що дозволяє вирішувати складні обчислювальні задачі.