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
Іваник Тарас

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

Мета лабораторної роботи:
Ознайомитися з динамічними структурами (Черга, Стек, Списки, Дерево). Навчитися застосовувати алгоритми обробки динамічних структур, навчитись працювати і розрізняти Виділення пам'яті для структур даних (stack і heap). Працювати з динамічними масивами. Навчитись реалізовувати операції push, pop, top, розуміти наввіщо їх використовувати, де вони потрібні. Вміти використовувати черги, застосовути операції enqueue, dequeue, front. Навчитись реалізовувати однозв'язні і двозв'язні списки, бінарні дерева, також реалізовувати основні операції: обхід списку, пошук, доступ до елементів, об'єднання списків. Вміти використовувати списки: управління пам'яттю, FIFO та LIFO структури

- **Джерела:**
- -Acode lessons
- -Blogan, BroCode (Youtube)
- -CS50 course
- -GeeksForGeek
- -University lectures

Lab# programming: VNS Lab 10 (200 хв)

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.

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.

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

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

struct Node{
    int key;
    Node* next;
    Node* prev;
    Node(int k) : key(k), next(nullptr), prev(nullptr) {};
};

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

public:
    DoubleLinkedList() : head(nullptr), tail(nullptr) {};

    void CreateEmptyList(){
        head = tail = nullptr;
        cout << "Empty list was createn! " << endl;
    }
    void addToEnd(int key){
        Node* newNode = new Node(key);
        if(!head){
            head = tail = newNode;
        } else{
            tail->next = newNode;
            newNode->prev = tail;
            tail = newNode;
        }
    }
    void printList() {
        if (!head) {
            cout << "Список порожній." << endl;
            return;
        }
        Node* temp = head;
        while (temp) {
            cout << temp->key << " ";
            temp = temp->next;
        }
        cout << endl;
    }
    void deleteAfter(int position) {
        if (!head) {
            cout << "Список порожній. Видалення неможливе." << endl;
            return;
        }
        Node* temp = head;
        int index = 0;
        while (temp && index < position) {
            temp = temp->next;
        }

        if (temp && temp->next) {
            Node* toDelete = temp->next;
            temp->next = toDelete->next;
            if (toDelete->next) {
                toDelete->next->prev = temp;
            } else {
                tail = temp;
            }
            delete toDelete;
            cout << "Елемент після позиції " << position << " видалено." << endl;
        } else {
            cout << "Видалення неможливе. Немає елемента після заданої позиції." << endl;
        }
    }
    void addToStart(int K, int startKey) {
        for (int i = 0; i < K; ++i) {
            Node* newNode = new Node(startKey + i);
            if (!head) {
                head = tail = newNode;
            } else {
                newNode->next = head;
                head->prev = newNode;
                head = newNode;
            }
        }
        cout << K << " елементів додано на початок списку." << endl;
    }
    void saveToFile(const string& filename) {
        ofstream file(filename);
        if (!file) {
            cerr << "Помилка відкриття файлу для запису!" << endl;
            return;
        }
        Node* temp = head;
        while (temp) {
            file << temp->key << " ";
            temp = temp->next;
        }
        file.close();
        cout << "Список збережено у файл '" << filename << "'." << endl;
    }
    void destroyList() {
        while (head) {
            Node* temp = head;
            head = head->next;
            delete temp;
        }
        tail = nullptr;
        cout << "Список знищено." << endl;
    }
    void loadFromFile(const string& filename) {
```

```
    }
    void loadFromFile(const string& filename) {
        ifstream file(filename);
        if (!file) {
            cerr << "Помилка відкриття файлу для читання!" << endl;
            return;
        }
        destroyList();
        int key;
        while (file >> key) {
            addToEnd(key);
        }
        file.close();
        cout << "Список відновлено з файлу '" << filename << "'." << endl;
    }
    ~DoubleLinkedList() {
        destroyList();
    }
};

int main() {
    DoubleLinkedList list;

    list.CreateEmptyList();
    list.addToEnd(10);
    list.addToEnd(20);
    list.addToEnd(30);
    cout << "Список після створення:" << endl;
    list.printList();

    list.addToStart(2, 50);
    cout << "Список після додавання елементів у початок:" << endl;
    list.printList();

    list.deleteAfter(1);
    cout << "Список після видалення елемента після позиції 1:" << endl;
    list.printList();

    list.saveToFile("list.txt"); // 4 (запис)

    list.destroyList();
    cout << "Список після знищення:" << endl;
    list.printList();

    list.loadFromFile("list.txt"); // відновлення списку
    cout << "Список після відновлення з файлу:" << endl;
    list.printList();

    list.destroyList();
    cout << "Список після остаточного знищення:" << endl;
    list.printList();

    return 0;
}
```

```
Empty list was createn!
Список після створення:
10 20 30
2 елементів додано на початок списку.
Список після додавання елементів у початок:
51 50 10 20 30
Елемент після позиції 1 видалено.
Список після видалення елемента після позиції 1:
51 50 20 30
Список збережено у файл 'list.txt'.
Список знищено.
Список після знищення:
Список порожній.
Список знищено.
Список відновлено з файлу 'list.txt'.
Список після відновлення з файлу:
51 50 20 30
Список знищено.
Список після остаточного знищення:
Список порожній.
Список знищено.
```

Lab# programming: Algotester Lab 5 (140 хв)

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.

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 N та M - розміри карти

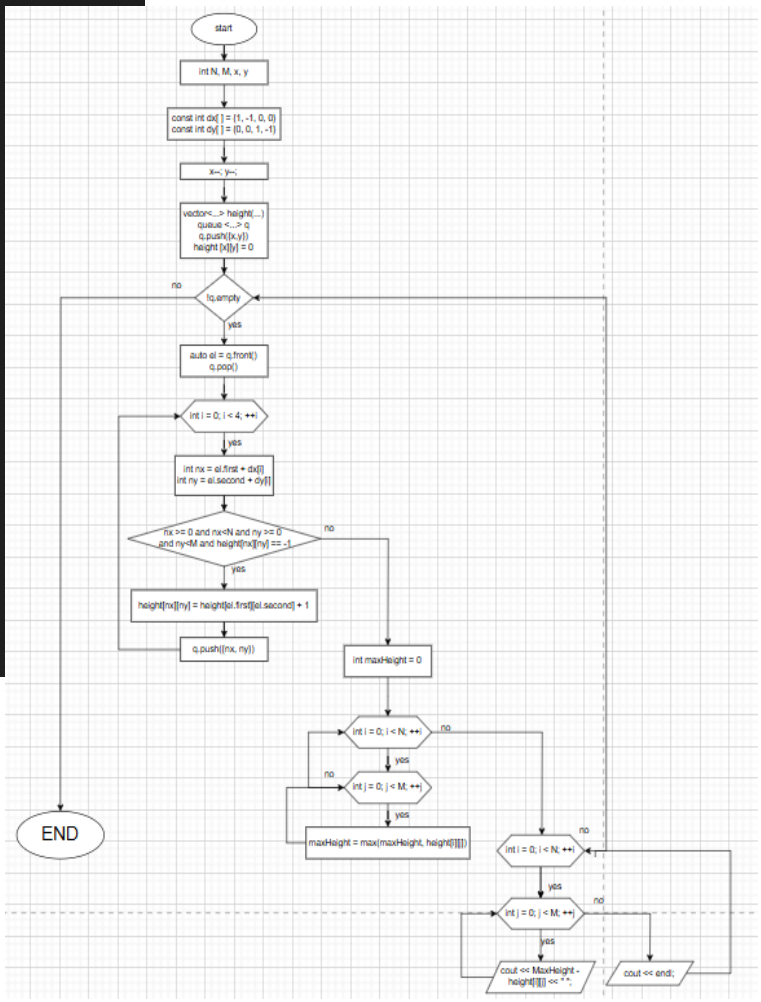
у другому рядку 2 числа x та y - координати піку гори

Output

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

```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 }; // зміщення клітинок для знаходження сусідів

    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]; // нові координати сусідів
            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 9
1 2
8 9 8 7 6 5 4 3 2
7 8 7 6 5 4 3 2 1
6 7 6 5 4 3 2 1 0
```



Lab# programming: Algotester Lab 78v1 (6 год)

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

Input

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

Output

Відповіді на запити у зазначеному в умові форматі.

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

template <typename T>
class DoublyLinkedList {
private:
    struct Node {
        T value;
        Node* next;
        Node* prev;
        Node(T val) : value(val), next(nullptr), prev(nullptr) {}
    };

    Node* head;
    Node* tail;
    int size;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}

    ~DoublyLinkedList() { clear(); }

    void insert(int index, int n, T* values) {
        if (index < 0 || index > size) return;

        Node* current = head;
        Node* prevNode = nullptr;

        for (int i = 0; i < index; ++i) {
            prevNode = current;
            current = current->next;
        }

        for (int i = 0; i < n; ++i) {
            Node* newNode = new Node(values[i]);

            if (!head) {
                head = tail = newNode;
            } else if (!prevNode) {
                newNode->next = head;
                head->prev = newNode;
                head = newNode;
            } else {
                newNode->next = current;
                newNode->prev = prevNode;
                if (prevNode) prevNode->next = newNode;
                if (current) current->prev = newNode;
            }

            prevNode = newNode;
            if (!current) tail = newNode;
        }

        size += n;
    }

    void erase(int index, int n) {
        if (index < 0 || index >= size || n <= 0) return;

        Node* current = head;

        for (int i = 0; i < index; ++i) {
            current = current->next;
        }

        for (int i = 0; i < n && current; ++i) {
            Node* toDelete = current;
            if (toDelete->prev) toDelete->prev->next = toDelete->next;
            if (toDelete->next) toDelete->next->prev = toDelete->prev;

            if (toDelete == head) head = toDelete->next;
            if (toDelete == tail) tail = toDelete->prev;

            current = current->next;
            delete toDelete;
            size--;
        }
    }

    int getSize() const {
        return size;
    }

    T get(int index) const {
        if (index < 0 || index >= size) throw out_of_range("Index out of range");

        Node* current = head;
        for (int i = 0; i < index; ++i) {
            current = current->next;
        }

        return current->value;
    }

    void set(int index, T value) {
        if (index < 0 || index >= size) throw out_of_range("Index out of range");

        Node* current = head;
        for (int i = 0; i < index; ++i) {
            current = current->next;
        }

        current->value = value;
    }

    void print() const {
        Node* current = head;
        while (current) {
```

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

    void clear() {
        Node* current = head;
        while (current) {
            Node* toDelete = current;
            current = current->next;
            delete toDelete;
        }

        head = tail = nullptr;
        size = 0;
    }

    friend ostream& operator<<(ostream& os, const DoublyLinkedList<T>& list) {
        Node* current = list.head;
        while (current) {
            os << current->value << " ";
            current = current->next;
        }
        return os;
    }
};

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

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

        if (command == "insert") {
            int index, n;
            cin >> index >> n;

            int* values = new int[n];
            for (int i = 0; i < n; ++i) {
                cin >> values[i];
            }

            list.insert(index, n, values);
            delete[] values;
        } else if (command == "erase") {
            int index, n;
            cin >> index >> n;
            list.erase(index, n);
        } else if (command == "size") {
            cout << list.getSize() << endl;
        } else if (command == "get") {
            int index;
            cin >> index;
            try {
                cout << list.get(index) << endl;
            } catch (const out_of_range& e) {
                cout << "Error: " << e.what() << endl;
            }
        } else if (command == "set") {
            int index, value;
            cin >> index >> value;
            try {
                list.set(index, value);
            } catch (const out_of_range& e) {
                cout << "Error: " << e.what() << endl;
            }
        } else if (command == "print") {
            cout << list << endl;
        }
    }

    return 0;
}
```

```
9
insert
0
5
1 2 3 4 5

insert
2
3
7 7 7

print
1 2 7 7 7 3 4 5

erase
1 2

print
1 7 7 3 4 5

size
6

get
3
3

set
3 13

print
1 7 7 13 4 5
```

Lab# programming: Algotester Lab 78v2 (5 год)

Lab 78v2

Limits: 1 sec., 256 MiB

Ваше завдання - власноруч реалізувати структуру даних "Динамічний масив".
Ви отримаєте Q запитів, кожен запит буде починатися зі слова-ідентифікатора, після якого йдуть його аргументи.

Вам будуть поступати запити такого типу:

- **Вставка**:
  Ідентифікатор - ***insert***
  Ви отримуєте ціле число ***index*** елемента, на місце якого робити вставку.
  Після цього в наступному рядку рядку написане число N - розмір масиву, який треба вставити.
  У третьому рядку N цілих чисел - масив, який треба вставити на позицію ***index***.
- **Видалення**:
  Ідентифікатор - ***erase***
  Ви отримуєте 2 цілих числа - ***index***, індекс елемента, з якого почати видалення та n - кількість елементів, яку треба видалити.
- **Визначення розміру**:
  Ідентифікатор - ***size***
  Ви не отримуєте аргументів.
  Ви виводите кількість елементів у динамічному масиві.
- **Визначення кількості зарезервованої пам'яті**:
  Ідентифікатор - ***capacity***
  Ви не отримуєте аргументів.
  Ви виводите кількість зарезервованої пам'яті у динамічному масиві.
  Ваша реалізація динамічного масиву має мати фактор росту (Growth factor) рівний 2.
- **Отримання значення** i-го елементу
  Ідентифікатор - ***get***
  Ви отримуєте ціле число - ***index***, індекс елемента.
  Ви виводите значення елемента за індексом. Реалізувати використовуючи перегрузку оператора []
- **Модифікація значення** i-го елементу
  Ідентифікатор - ***set***
  Ви отримуєте 2 цілих числа - індекс елемента, який треба змінити, та його нове значення. Реалізувати використовуючи перегрузку оператора []
- **Вивід динамічного масиву на екран**
  Ідентифікатор - ***print***
  Ви не отримуєте аргументів.
  Ви виводите усі елементи динамічного масиву через пробіл.
  Реалізувати використовуючи перегрузку оператора <<

Input

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

Output

Відповіді на запити у зазначеному в умові форматі.

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

template<typename T>
class DynamicArray{
    private:
    T* data;
    int size;
    int capacity;

    void r_size(int new_capacity){
        T* new_data = new T[new_capacity];
        for (int i = 0; i < size; i++)
        {
            new_data[i] = data[i];
        }
        delete[] data;
        data = new_data;
        capacity = new_capacity;
    }
    public:
    DynamicArray() : size(0), capacity(1){
        data = new T[capacity];
    }

    ~DynamicArray()  {
        delete[] data;
    }

    void insert(int index, int n, T* values){
        if (size + n > capacity)
        {
            int new_capacity = capacity;
            while (size + n >= new_capacity)
            {
                new_capacity *= 2;
            }
            r_size(new_capacity);
        }
        for (int i = size - 1; i >= index; i--)
        {
            data[i + n] = data[i];
        }
        for (int i = 0; i < n; i++)
        {
            data[index + i] = values[i];
        }
        size += n;
        if (size == capacity)
        {
            capacity *= 2;
            r_size(capacity);
        }

    }
    void erase(int index, int n){
        for (int i = index + n; i < size; i++)
        {
            data[i - n] = data[i];
        }
        size -= n;

    }
    int get_size() const{
        return size;
    }
    int get_capacity() const{
        return capacity;
    }
    T& operator[](int index){
        if (index < 0 || index >= size)
        {
            throw out_of_range("ERROR");
        }
        return data[index];
    }
    friend ostream& operator<<(ostream& os, const DynamicArray& arr){
        for (int i = 0; i < arr.size; i++)
        {
            os << arr.data[i] << (i < arr.size - 1 ? " " : "");
        }
        return os;
    }
};
int main(){
    DynamicArray<int> arr;
    int Q;
    cin >> Q;

    for (int i = 0; i < Q; i++)
    {
        string c;
        cin >> c;

        if (c == "insert"){
            int index, n;
            cin >> index >> n;
            int* values = new int[n];
            for (int j = 0; j < n; j++){
                cin >> values[j];
            }
            arr.insert(index, n, values);
            delete[] values;
        }else if (c == "erase"){
            int index, n;
            cin >> index >> n;
            arr.erase(index, n);

        }else if (c == "size"){ cout << arr.get_size() << endl;}

        else if (c == "capacity") { cout << arr.get_capacity() << endl;}

        else if (c == "get"){
            int index;
            cin >> index;
            cout << arr[index] << endl;

        }else if (c == "set"){
            int index, value;
            cin >> index >> value;
            arr[index] = value;

        }else if (c == "print") { cout << arr << endl;}
    }
    return 0;
}
```

```
12

size
0
capacity
1

insert 0 2
100 100

size
2
capacity
4

insert 0 2
102 102

size
4
capacity
8

insert 0 2
103 103

size
6
capacity
8

print
103 103 102 102 100 100
```

Practice# programming: Class Practice Task (5 год)

Задача №1 - Реверс списку (Reverse list)

Реалізувати метод реверсу списку: Node* reverse(Node *head);
Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати метод реверсу;
- реалізувати допоміжний метод виведення вхідного і обернутого списків;

Задача №2 - Порівняння списків

bool compare(Node *h1, Node *h2);
Умови задачі:

- використовувати цілочисельні значення в списку;
- реалізувати функцію, яка ітеративно проходиться по обох списках і порівнює дані в кожному вузлі;

- якщо виявлено невідповідність даних або якщо довжина списків різна (один список закінчується раніше іншого), функція повертає ***false***.

Задача №3 – Додавання великих чисел

Node* add(Node *n1, Node *n2);
Умови задачі:

- використовувати цифри від 0 до 9 для значень у списку;
- реалізувати функцію, яка обчислює суму двох чисел, які збережено в списку; молодший розряд числа записано в голові списка (напр. 379 $\Longrightarrow$ 9→7→3);
- функція повертає новий список, передані в функцію списки не модифікуються.

Задача №4 - Віддзеркалення дерева

TreeNode *create_mirror_flip(TreeNode *root);
Умови задачі:

- використовувати цілі числа для значень у вузлах дерева
- реалізувати функцію, що проходить по всіх вузлах дерева і міняє місцями праву і ліву вітки дерева
- функція повертає нове дерево, передане в функцію дерево не модифікується

Задача №5 - Записати кожному батьківському вузлу суму підвузлів

void tree_sum(TreeNode *root);
Умови задачі:

- використовувати цілочисельні значення у вузлах дерева;
- реалізувати функцію, яка ітеративно проходить по бінарному дереві і записує у батьківський вузол суму значень підвузлів
- вузол-листок не змінює значення
- значення змінюються від листків до кореня дерева

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

struct Node {
    int data;
    Node* next;

    Node(int val) : data(val), next(nullptr) {}
};
Node* reverse(Node* head) {
    Node* prev = nullptr;
    Node* current = head;

    while (current) {
        Node* nextNode = current->next;
        current->next = prev;
        prev = current;
        current = nextNode;
    }

    return prev;
}
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;
}

Node* add(Node* n1, Node* n2) {
    Node* result = nullptr;
    Node* tail = nullptr;
    int carry = 0;

    while (n1 || n2 || carry) {
        int sum = (n1 ? n1->data : 0) + (n2 ? n2->data : 0) + carry;
        carry = sum / 10;
        Node* newNode = new Node(sum % 10);

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

        if (n1) n1 = n1->next;
        if (n2) n2 = n2->next;
    }

    return result;
}
```

```cpp
void printList(Node* head) {
    while (head) {
        cout << head->data << " ";
        head = head->next;
    }
    cout << endl;
}
struct TreeNode {   //  вузол дерева
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int val) : data(val), left(nullptr), right(nullptr) {}
};
TreeNode* create_mirror_flip(TreeNode* root) {
    if (!root) return nullptr;

    TreeNode* newRoot = new TreeNode(root->data);
    newRoot->left = create_mirror_flip(root->right);
    newRoot->right = create_mirror_flip(root->left);

    return newRoot;
}
int tree_sum(TreeNode* root) {
    if (!root) return 0;

    if (!root->left && !root->right) return root->data;

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

    root->data += leftSum + rightSum;
    return root->data;
}
void printTree(TreeNode* root) {
    if (!root) return;
    cout << root->data << " ";
    printTree(root->left);
    printTree(root->right);
}

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

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

    Node* reversedHead = reverse(head);
    cout << "Reversed list: ";
    printList(reversedHead);
```

```
Original list: 1 2 3
Reversed list: 3 2 1
Are lists equal? Yes
Sum of numbers: 5 2 9
Original tree (pre-order): 1 2 4 5 3
Mirrored tree (pre-order): 1 3 2 5 4
Tree with subtree sums (pre-order): 15 11 4 5 3
```

```cpp
    Node* list1 = new Node(1);
    list1->next = new Node(2);
    list1->next->next = new Node(3);

    Node* list2 = new Node(1);
    list2->next = new Node(2);
    list2->next->next = new Node(3);

    cout << "Are lists equal? " << (compare(list1, list2) ? "Yes" : "No") << endl;

    Node* num1 = new Node(9);     // Додавання великих чисел
    num1->next = new Node(7);
    num1->next->next = new Node(3);

    Node* num2 = new Node(6);
    num2->next = new Node(4);
    num2->next->next = new Node(5);

    Node* sumList = add(num1, num2);
    cout << "Sum of numbers: ";
    printList(sumList);

    TreeNode* tree = new TreeNode(1);   //  Віддзеркалення дерева
    tree->left = new TreeNode(2);
    tree->right = new TreeNode(3);
    tree->left->left = new TreeNode(4);
    tree->left->right = new TreeNode(5);

    cout << "Original tree (pre-order): ";
    printTree(tree);
    cout << endl;

    TreeNode* mirroredTree = create_mirror_flip(tree);
    cout << "Mirrored tree (pre-order): ";
    printTree(mirroredTree);
    cout << endl;

    tree_sum(tree);                                    // Додавання сум підвузлів
    cout << "Tree with subtree sums (pre-order): ";
    printTree(tree);
    cout << endl;

    return 0;
}
```

Practice# programming: Self Practice Task (1 год) [Lab 5v1]

Lab 5v1

Limits: 2 sec., 256 MiB

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 заклиння, 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.

Input

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.

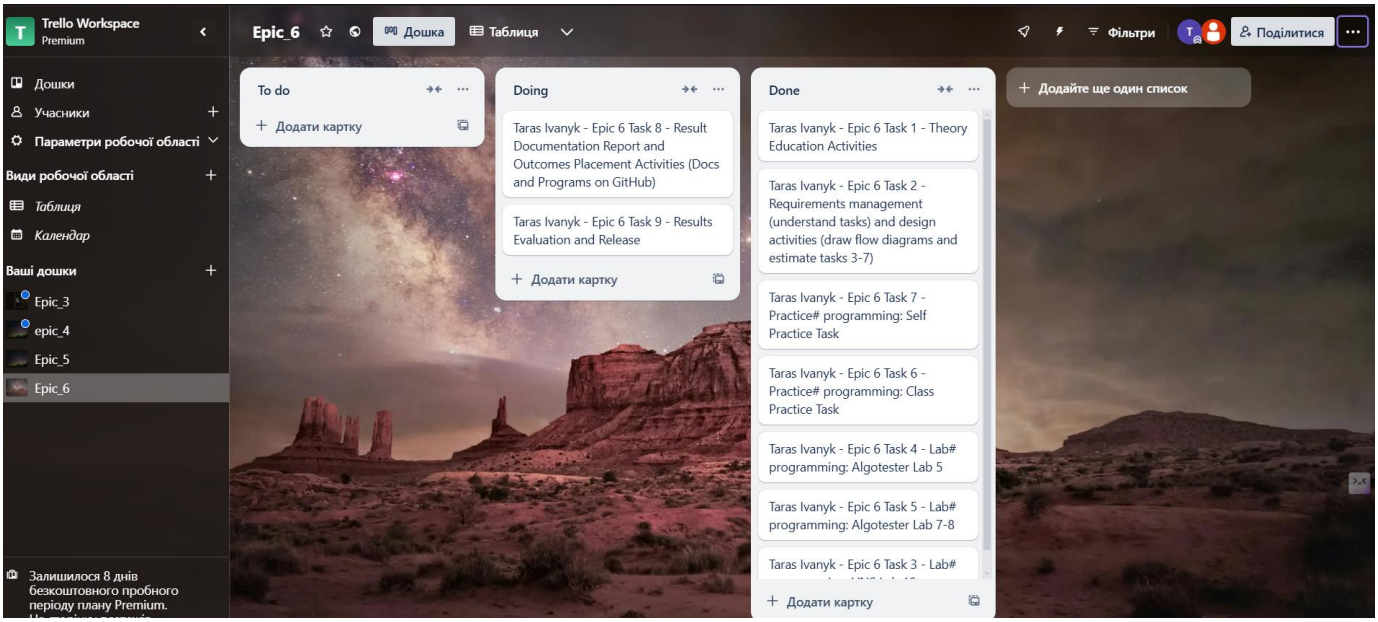
Output

Одне число b - стан дошки після виконання N заклинань.

```cpp
// algotester lab 5v1
#include <iostream>
#include <cstdint>
#include <vector>
using namespace std;

void BallCondition(uint64_t &table, int row, int col) {
    int position = row * 8 + col;
    table = table ^ (1ULL << position);  // зсув 1 біта на потрібну нам position
}
int main() {
    uint64_t table;
    int N;

    cin >> table;
    cin >> N;

    vector<pair<int, int>> mantras(N);
    for (int i = 0; i < N; ++i) {
        cin >> mantras[i].first >> mantras[i].second;
        mantras[i].first--;
        mantras[i].second--;
    }
    for (const auto &mantra : mantras) {
        int row = mantra.first;
        int col = mantra.second;

        for (int c = 0; c < 8; ++c) {
            BallCondition(table, row, c);
        }
        for (int r = 0; r < 8; ++r) {
            if (r != row) {
                BallCondition(table, r, col);
            }
        }
    }
    cout << table << endl;
    return 0;
}
```

```
0
4
1 1
8 8
1 8
8 1
9295429630892703873
```

TRELLO:



Offline meet:

PULL REQUEST

Висновок: під час виконання 6 епіку, я дізнався багато нового, зокрема, що таке однозв’язні, двозв’язні списки, черги, бінарні дерева та як з ними працювати.