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
Шийка Стефан

**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:

## Lab# programming: VNS Lab 10

## Time expected: 3h

## Time spent: 3h

**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.

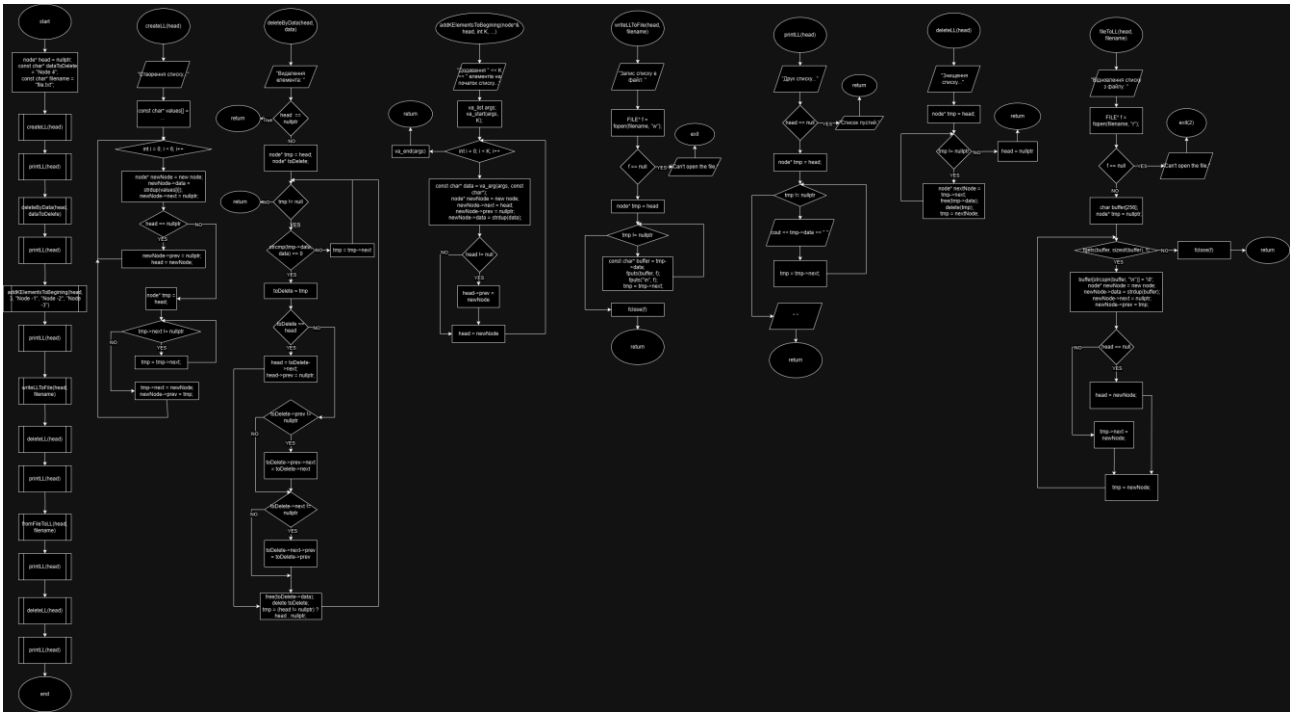
---

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.

18. Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити елемент із заданим ключем. Додати К елементів у початок списку.

```
Створення списку...
Друк списку...
Node 1 Node 2 Node 3 Node 4 Node 5 Node 6
Видалення елемента: Node 4 зі списку...
Друк списку...
Node 1 Node 2 Node 3 Node 5 Node 6
Додавання 3 елементів на початок списку...
Друк списку...
Node -3 Node -2 Node -1 Node 1 Node 2 Node 3 Node 5 Node 6
Запис списку в файл: file.txt...
Знищення списку...
Друк списку...
Список пустий.
Відновлення списку з файлу: file.txt...
Друк списку...
Node -3 Node -2 Node -1 Node 1 Node 2 Node 3 Node 5 Node 6
Знищення списку...
Друк списку...
Список пустий.
PS C:\Users\user\Desktop\c++\epic6>
```

```cpp
#include <iostream>
#include <cstring>
#include <cstdarg>

using namespace std;

struct node {
    char* data;
    node* next;
    node* prev;
};

void createLL(node*& head) {
    cout << "Створення списку..." << endl;
    const char* values[] = {"Node 1", "Node 2", "Node 3", "Node 4", "Node 5", "Node 6"};

    for(int i = 0; i < 6; i++){
        node* newNode = new node;
        newNode->data = strdup(values[i]);
        newNode->next = nullptr;

        if(head == nullptr){
            newNode->prev = nullptr;
            head = newNode;
        }else{
            node* tmp = head;
            while(tmp->next != nullptr){
                tmp = tmp->next;
            }

            tmp->next = newNode;
            newNode->prev = tmp;
        }
    }
}

void deleteByData(node*& head, const char* data) {
    cout << "Видалення елемента: " << data << " зі списку..." << endl;
    if(head == nullptr) return;
    node* tmp = head;
    node* toDelete;

    while (tmp != nullptr) {
        if (strcmp(tmp->data, data) == 0) {
            toDelete = tmp;

            if (toDelete == head) {
                head = toDelete->next;
                head->prev = nullptr;
            } else {
                if (toDelete->prev != nullptr) {
                    toDelete->prev->next = toDelete->next;
                }
                if (toDelete->next != nullptr) {
                    toDelete->next->prev = toDelete->prev;
                }
            }

            free(toDelete->data); //from the use of strdup()
            delete toDelete;
            tmp = (head != nullptr) ? head : nullptr;
        } else {
            tmp = tmp->next;
        }
    }
}

void printLL(node* head) {
    cout << "Друк списку..." << endl;

    if(head == nullptr){
        cout << "Список пустий." << endl;
        return;
    }

    node* tmp = head;
    while(tmp != nullptr){
        cout << tmp->data << " ";
        tmp = tmp->next;
    }
    cout << endl;
}

void addKElementsToBegining(node*& head, int K, ...) {
    cout << "Додавання " << K << " елементів на початок списку..." << endl;

    va_list args;
    va_start(args, K);

    for(int i = 0; i < K; i++){
        const char* data = va_arg(args, const char*);
        node* newNode = new node;
        newNode->next = head;
        newNode->prev = nullptr;
        newNode->data = strdup(data);

        if(head != nullptr){
            head->prev = newNode;
        }
        head = newNode;
    }
    va_end(args);
}

void writeLLToFile(node*& head, const char* filename) {
    cout << "Запис списку у файл: " << filename << "..." << endl;
    FILE* f = fopen(filename, "w");
    if(f == nullptr){
        cerr << "Can't open the file.";
        exit(1);
    }

    node* tmp = head;
    while(tmp != nullptr){
        const char* buffer = tmp->data;
        fputs(buffer, f);
        fputs("\n", f);
        tmp = tmp->next;
    }
    fclose(f);
}

void deleteLL(node*& head) {
    cout << "Знищення списку..." << endl;
    node* tmp = head;
    while(tmp != nullptr){
        node* nextNode = tmp->next;
        free(tmp->data);
        delete(tmp);
        tmp = nextNode;
    }
    head = nullptr;
}

void fromFileToLL(node*& head, const char* filename) {
    cout << "Відновлення списку з файлу: " << filename << "..." << endl;
    FILE* f = fopen(filename, "r");
    if(f == nullptr){
        cerr << "Can't open the file.";
        exit(2);
    }

    char buffer[256];
    node* tmp = nullptr;
    while(fgets(buffer, sizeof(buffer), f)){
        buffer[strcspn(buffer, "\n")] = '\0';

        node* newNode = new node;
        newNode->data = strdup(buffer);
        newNode->next = nullptr;
        newNode->prev = tmp;

        if(head == nullptr){
            head = newNode;
        }else{
            tmp->next = newNode;
        }
        tmp = newNode;
    }
    fclose(f);
}

int main() {
    node* head = nullptr;
    const char* dataToDelete = "Node 4";
    const char* filename = "file.txt";

    createLL(head);
    printLL(head);

    deleteByData(head, dataToDelete);
    printLL(head);

    addKElementsToBegining(head, 3, "Node -1", "Node -2", "Node -3");
    printLL(head);

    writeLLToFile(head, filename);
    deleteLL(head);
    printLL(head);

    fromFileToLL(head, filename);
    printLL(head);

    deleteLL(head);
    printLL(head);
}
```



# Lab# programming: Algotester Lab 5

# Time expected: 15 min

# Time spent: 1h+

### 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 <queue>

using namespace std;

//BFS algorithm
int main(){
    int N, M, x, y;
    cin >> N >> M;
    cin >> x >> y;
    x--;
    y--;
    int map[N][M];
    int dx[] = {-1, 1, 0, 0};
    int dy[] = {0, 0, -1, 1};
    for(int i = 0; i < N; i++){
        for(int j = 0; j < M; j++){
            map[i][j] = -1;
        }
    }
    queue<pair<int, int>> q;
    map[x][y] = 0;
    q.push({x,y});

    //the order of numbers will be reversed at first
    while(!q.empty()){
        int curr_x = q.front().first;
        int curr_y = q.front().second;
        int curr_height = map[curr_x][curr_y];
        q.pop();

        for (int i = 0; i < 4; i++){
            int new_x = curr_x + dx[i];
            int new_y = curr_y + dy[i];

            //check if the coordinates are not out of bounds and are not altered yet
            if(new_x >= 0 && new_x < N && new_y >= 0 && new_y < M && map[new_x][new_y] == -1){
                map[new_x][new_y] = curr_height + 1;
                q.push({new_x, new_y});
            }
        }
    }

    //find the max value of the map
    int max = map[0][0];
    for(int i = 0; i < N; i++){
        for(int j = 0; j < M; j++){
            if(map[i][j] > max) max = map[i][j];
        }
    }

    //reverse the heights
    for(int i = 0; i < N; i++){
        for(int j = 0; j < M; j++){
            map[i][j] = max - map[i][j];
            cout << map[i][j] << " ";
        }
        cout << endl;
    }


}
```

```
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
PS C:\Users\user\D
```

| | | | | | |
|---|---|---|---|---|---|
| 40 minutes ago | Lab 5v3 - Lab 5v3 | C++ 23 | Accepted | 0.119 | 6.848 | 1859439 |

# Lab# programming: Algotester Lab 7-8

# Time expected: 2.5h

# Time spent: 4.5h

# Var – 2

## 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 <<

| | | | | | | |
|---|---|---|---|---|---|---|
| 7 minutes ago | Lab 78v2 - Lab 78v2 | C++ 23 | Accepted | 0.006 | 1.176 | 1859647 |
| 9 minutes ago | Lab 78v2 - Lab 78v2 | C++ 23 | Run Time Error 4 | 0.004 | 0.941 | 1859645 |
| 9 minutes ago | Lab 78v2 - Lab 78v2 | C++ 23 | Run Time Error 4 | 0.004 | 1.031 | 1859643 |

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

using namespace std;

enum Operation {
    INSERT,
    ERASE,
    SIZE,
    CAPACITY,
    GET,
    SET,
    PRINT,
    UNKNOWN
};

Operation getOperation(const string& command) {
    if (command == "insert") return INSERT;
    if (command == "erase") return ERASE;
    if (command == "size") return SIZE;
    if (command == "capacity") return CAPACITY;
    if (command == "get") return GET;
    if (command == "set") return SET;
    if (command == "print") return PRINT;
    return UNKNOWN;
}

template<typename T = int> //int by default
class DynamicArray{
private:
    T* data;
    int size_;
    int capacity_;

    void resize(int newCapacity){
        T* newData = new T[newCapacity];
        //copy
        for(int i = 0; i < size_; i++){
            newData[i] = data[i];
        }
        delete[] data;
        data = newData;
        capacity_ = newCapacity;
    }

public:
    DynamicArray() : size_(0), capacity_(1){
        data = new T[capacity_];
    }

    ~DynamicArray(){
        delete[] data;
    }

    void insert(int index, int N, T* arr){
    if (size_ + N > capacity_) {
        while (capacity_ <= size_ + N) {
            capacity_ *= 2;
        }
        resize(capacity_);
    }
        //move a part of the array to the right to insert by index
        for(int i = size_ - 1; i >= index; i--){
            data[i + N] = data[i];
        }
        for(int i = 0; i < N; i++){
            data[index + i] = arr[i];
        }
        size_ += N;
        if(size_ == capacity_){
            capacity_ *= 2;
            resize(capacity_);
        }
    }

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

    int size(){
        return size_;
    }

    int capacity(){
        return capacity_;
    }

    //eccess elements by index
    T& operator[](int index) { return data[index]; }

    //Functionality to be able to cout by using <<
    friend ostream& operator<<(ostream& os, const DynamicArray& arr){
        for (int i = 0; i < arr.size_; i++) {
            os << arr.data[i] << ' ';
        }
        return os;
    }

};

int main(){
    int Q;
    cin >> Q;
    DynamicArray<int> dynArr;

    for(int i = 0; i < Q; i++){
        string option;
        cin >> option;
        Operation operation = getOperation(option);

        switch (operation) {
            case INSERT: {
                int index, N;
                cin >> index >> N;
                int arr[N];
                for(int i = 0; i < N; i++){
                    cin >> arr[i];
                }
                dynArr.insert(index, N, arr);
                break;
            }
            case ERASE: {
                int index, n;
                cin >> index >> n;
                dynArr.erase(index, n);
                break;
            }
            case SIZE: {
                cout << dynArr.size() << endl;
                break;
            }
            case CAPACITY: {
                cout << dynArr.capacity() << endl;
                break;
            }
            case GET: {
                int index;
                cin >> index;
                cout << dynArr[index] << endl;
                break;
            }
            case SET: {
                int index, value;
                cin >> index >> value;
                dynArr[index] = value;
                break;
            }
            case PRINT: {
                cout << dynArr << endl;
                break;
            }
            default:
                break;
        }
    }
}
```

# Var – 3

## Lab 78v3

*Limits: 1 sec., 256 MiB*

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 $<<$

### Input

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

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 minutes ago | Lab 78v3 - Lab 78v3 | C++ 23 | Accepted | 0.009 | 1.176 | 1860171 |

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

using namespace std;

enum Operation {
    INSERT,
    SIZE,
    PRINT,
    CONTAINS,
    UNKNOWN
};

Operation getOperation(const string& command) {
    if (command == "insert") return INSERT;
    if (command == "size") return SIZE;
    if (command == "print") return PRINT;
    if (command == "contains") return CONTAINS;
    return UNKNOWN;
}

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

    Node* root;
    int treeSize;

    void destroyTree(Node* root) {
        if (root != nullptr) {
            destroyTree(root->left);
            destroyTree(root->right);
            delete root;
        }
    }

    void insertRecursively(Node* node, T value) {
        if (value < node->value) {
            if (node->left == nullptr) {
                node->left = new Node(value);
                treeSize++;
            } else {
                insertRecursively(node->left, value);
            }
        } else if (value > node->value) {
            if (node->right == nullptr) {
                node->right = new Node(value);
                treeSize++;
            } else {
                insertRecursively(node->right, value);
            }
        }
    }

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

    void printTree(Node* node, ostream& os) const {
        if (node != nullptr) {
            printTree(node->left, os);
            os << node->value << " ";
            printTree(node->right, os);
        }
    }

public:
    binaryTree() : root(nullptr), treeSize(0) {}

    ~binaryTree() {
        destroyTree(root);
    }

    void insert(T value) {
        if (root == nullptr) {
            root = new Node(value);
            treeSize++;
        } else {
            insertRecursively(root, value);
        }
    }

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

    int getSize() const {
        return treeSize;
    }

    // Overload << operator to print the tree in-order
    friend ostream& operator<<(ostream& os, const binaryTree& tree) {
        tree.printTree(tree.root, os);
        return os;
    }
};

int main() {
    int Q;
    cin >> Q;
    binaryTree<int> tree;

    for (int i = 0; i < Q; i++) {
        string option;
        cin >> option;
        Operation operation = getOperation(option);

        switch (operation) {
            case INSERT: {
                int value;
                cin >> value;
                tree.insert(value);
                break;
            }
            case SIZE: {
                cout << tree.getSize() << endl;
                break;
            }
            case CONTAINS: {
                int value;
                cin >> value;
                cout << (tree.contains(value) ? "Yes" : "No") << endl;
                break;
            }
            case PRINT: {
                cout << tree << endl;
                break;
            }
            default:
                break;
        }
    }
}

```

# Practice# programming: Class Practice Task

# Time expected: 2 h

# Time spent: 3.5h

## 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 <cstring>
#include <vector>

using namespace std;

struct node { // for LL
    int data;
    node* next;
};

struct Node { // for tree
    int data;
    Node* left;
    Node* right;

    Node(int value) {
        data = value;
        left = nullptr;
        right = nullptr;
    }
};

void createLL(node*& head, vector<int>& vec) {
    cout << "Створення списку..." << endl;
    node* tail = nullptr;  // pointer to the last node

    for (int i = 0; i < vec.size(); i++) {
        node* newNode = new node;
        newNode->data = vec[i];
        newNode->next = nullptr;

        if (head == nullptr) {
            head = newNode;
            tail = newNode;
        } else {
            tail->next = newNode;
            tail = newNode;
        }
    }
}

void reverseLL(node*& head){
    node* current = head;
    node* next = nullptr;
    node* prev = nullptr;

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

void printLL(node* head) {
    cout << "Друк списку..." << endl;

    if(head == nullptr){
        cout << "Список пустий." << endl;
        return;
    }

    node* tmp = head;
    while(tmp != nullptr){
        cout << tmp->data << " ";
        tmp = tmp->next;
    }
    cout << endl;
}

bool compare(node* h1, node* h2){
    node* tmp1 = h1;
    node* tmp2 = h2;

    while(tmp1 != nullptr && tmp2 != nullptr){
        if(tmp1->data != tmp2->data) return false;
        tmp1 = tmp1->next;
        tmp2 = tmp2->next;
    }
    if(tmp1 != tmp2) return false;  // one ll is shorter than the other
    return true;
}

node* add(node* h1, node* h2){
    int a = 0;
    int sum, x, y;
    node* h = nullptr;
    node* current = nullptr;

    while(h1 != nullptr || h2 != nullptr){
        if(h1 == nullptr){
            x = 0;
        }else{
            x = h1->data;
        }
        if(h2 == nullptr){
            y = 0;
        }else{
            y = h2->data;
        }

        sum = x + y + a;
        a = sum / 10;
        node* tmp = new node;
        tmp->data = sum % 10;
        tmp->next = nullptr;

        if(h == nullptr){
            h = tmp;
            current = tmp;
        }else{
            current->next = tmp;
            current = tmp;
        }

        if(h1 != nullptr) h1 = h1->next;
        if(h2 != nullptr) h2 = h2->next;
    }

    if (a > 0) {
        node* tmp = new node;
        tmp->data = a;
        tmp->next = nullptr;
        current->next = tmp;
    }

    return h;
}

void freeLL(node*& head){
    while(head != nullptr){
        node* tmp = head;
        head = head->next;
        delete tmp;
    }
}
```

```cpp
void insertIntoTree(Node*& root, int value){
    if(root == nullptr) {
        root = new Node(value);
        return;
    }

    if(value < root->data){
        insertIntoTree(root->left, value);
    }else{
        insertIntoTree(root->right, value);
    }
}

Node* mirror(Node* root){
    if(root == nullptr){
        return nullptr;
    }

    Node* mirroredNode = new Node(root->data);

    mirroredNode->left = mirror(root->right);
    mirroredNode->right = mirror(root->left);

    return mirroredNode;
}

void sumTree(Node* root){
    if(root == nullptr) return;

    sumTree(root->left);
    sumTree(root->right);

    if(root->left != nullptr || root->right != nullptr){
        int leftValue = (root->left != nullptr) ? root->left->data : 0;
        int rightValue = (root->right != nullptr) ? root->right->data : 0;
        root->data = leftValue + rightValue;
    }
}

void printTree(Node* root){
    if(root == nullptr) return;
    cout << root->data << " ";
    printTree(root->left);
    printTree(root->right);
}

void freeTree(Node*& root){
    if(root == nullptr) return;

    freeTree(root->left);
    freeTree(root->right);
    delete root;
}

int main() {
    vector<int> vec1 = {3, 7, 9};
    vector<int> vec2 = {2, 4, 8};

    node* head1 = nullptr;
    node* head2 = nullptr;

    cout << "----------------------------------------" << endl;
    cout << "               LINKED LISTS" << endl;

    createLL(head1, vec1);
    cout << "Перший список: ";
    printLL(head1);

    createLL(head2, vec2);
    cout << "Другий список: ";
    printLL(head2);

    node* result = add(head1, head2);
    cout << "Результат додавання: ";
    printLL(result);

    reverseLL(head1);
    cout << "Перший список після реверсу: ";
    printLL(head1);

    reverseLL(head2);
    cout << "Другий список після реверсу: ";
    printLL(head2);

    bool isEqual = compare(head1, head2);
    cout << "Чи рівні списки: " << (isEqual ? "Так" : "Ні") << endl;
    freeLL(head1);
    freeLL(head2);

    cout << "----------------------------------------" << endl;
    cout << "                  TREES" << endl;

    Node* root = nullptr;
    vector<int> treeValues = {4, 2, 6, 1, 3, 5, 7};
    int len = treeValues.size();
    for(int i = 0; i < len; i++){
        insertIntoTree(root, treeValues[i]);
    }
    cout << "Оригінальне дерево (виведене по рівнях): ";
    printTree(root);
    cout << endl;

    Node* mirroredRoot = mirror(root);
    cout << "Дерево після дзеркального відображення: ";
    printTree(mirroredRoot);
    cout << endl;

    sumTree(root);
    cout << "Дерево після обчислення сум підвузлів: ";
    printTree(root);
    cout << endl;

    freeTree(root);
    freeTree(mirroredRoot);
}
```

Тут номерація рядочків має продовжуватися від верхнього скрина

```
----------------------------------------
               LINKED LISTS
Створення списку...
Перший список: Друк списку...
3 7 9
Створення списку...
Другий список: Друк списку...
2 4 8
Результат додавання: Друк списку...
5 1 8 1
Перший список після реверсу: Друк списку...
9 7 3
Другий список після реверсу: Друк списку...
8 4 2
Чи рівні списки: Ні
----------------------------------------
                  TREES
Оригінальне дерево (виведене по рівнях): 4 2 1 3 6 5 7
Дерево після дзеркального відображення: 4 6 7 5 2 3 1
Дерево після обчислення сум підвузлів: 16 4 1 3 12 5 7
PS C:\Users\user\Desktop\c++\epic6>
```

# Practice# programming:  Self Practice Task

# Time expected: 30min

# Time spent: 20 min

От і широковідомого у вузьких колах програміста Антона не минули стріли Амура — ось уже кілька тижнів йому з голови не йде одна прекрасна особа. Відверто кажучи, автор цього тексту не знає, як її звати, та це й не має ніякого значення. Коли наш закоханий іде вулицею і бачить вивіски, він машинально запам'ятовує їх, а потім думає, скільки разів можна скласти з усіх букв, які йому зустрілися, слово «весна». Або ім'я коханої. Або ще щось.

Допоможіть йому порахувати, скільки разів він зможе скласти задумане ним слово, якщо будь-яку букву можна використати не більше ніж один раз.

### Input

У першому рядку задано задумане Антоном слово.

У другому рядку заданоя натуральне число $n$ — кількість вивісок, які прочитав Антон.

У наступних $n$ рядках містяться вивіски, які прочитав Антон.

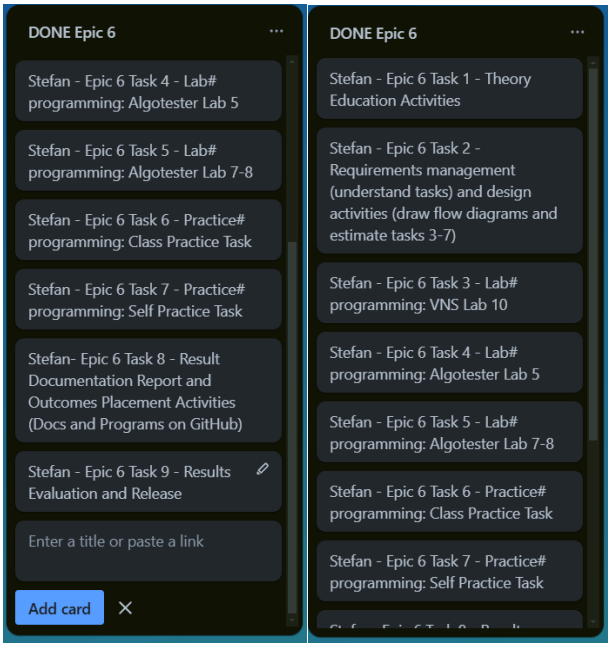
### Output

У єдиному рядку виведіть одне ціле число — скільки разів можна скласти задумане нашим героєм слово з букв, які зустрічаються в вивісках.

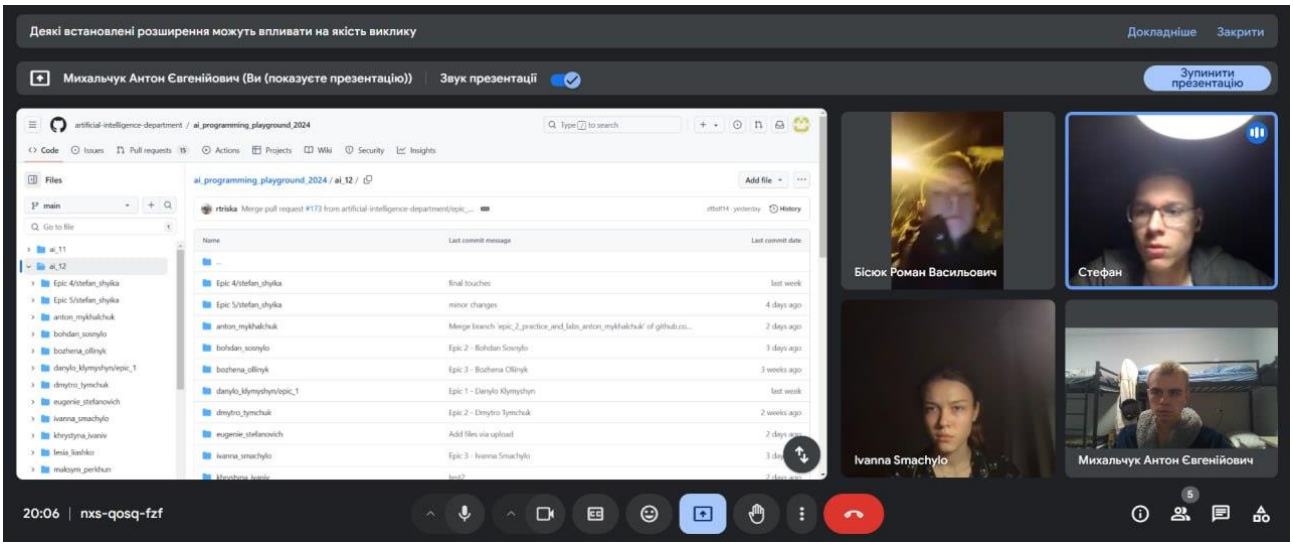
```cpp
#include <iostream>
#include <unordered_map>
#include <algorithm>
#include <string>
#include <climits>

using namespace std;

int main(){
    int n;
    string word;
    cin >> word;
    cin >> n;
    unordered_map<char, int> wordCount;

    for(char c : word){
        wordCount[c]++;
    }

    unordered_map<char, int> signCount;

    for (int i = 0; i < n; i++) {
        string sign;
        cin >> sign;
        for (char c : sign) {
            signCount[c]++;
        }
    }

    int max = INT_MAX;

    for(auto& pair : wordCount){
        char c = pair.first;
        int needed = pair.second;

        if(signCount[c] == 0){
            max = 0;
            break;
        }

        max = min(max, signCount[c] / needed);
    }

    cout << max;
}
```

| | | | | | | |
|---|---|---|---|---|---|---|
| a minute ago | 0404 - Весна | C++ 23 | Accepted | 0.003 | 1.051 | 1860318 |

# Trello:




# Meet:




# Pull

**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. Я також засвоїв відмінності між статичним та динамічним виділенням пам'яті, що дає мені краще уявлення про ефективне управління ресурсами. Практичні вправи з такими структурами, як стек, черга, зв'язний список і дерево, дозволили мені глибше зрозуміти принципи обробки даних у пам'яті, а також основи алгоритмів, що застосовуються для роботи з динамічними структурами.