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
Стик Назарій Олегович

Львів 2024

Тема роботи:

Динамічні структури (Черга, Стек, Списки, Дерево). Алгоритми обробки динамічних структур.

Мета роботи:

Ознайомлення з основами роботи з динамічними структурами даних, такими як черга, стек, списки та дерево. Вивчення їхньої внутрішньої організації, ключових властивостей і методів доступу. Розгляд алгоритмів для роботи з динамічними структурами, зокрема для додавання, видалення, пошуку та сортування елементів. Вивчення ефективності алгоритмів обробки динамічних структур та їх застосування в програмуванні. Особлива увага приділяється практичному використанню стандартних бібліотек для реалізації цих структур і створенню власних оптимізованих реалізацій.

Теоретичні відомості:

У даній роботі розглядаються основи роботи з динамічними структурами даних, такими як черга, стек, списки та дерево. Черга є структурою даних типу FIFO (перший увійшов — перший вийшов), яка використовується для організації послідовності задач або процесів. Стек реалізує принцип LIFO (останній увійшов — перший вийшов) і широко застосовується для управління викликами функцій та обробки рекурсій. Списки забезпечують зручну роботу з динамічною кількістю елементів завдяки зв'язкам між вузлами, а дерева представляють ієрархічні структури даних, які ефективно застосовуються для пошуку, сортування та організації даних. Розглядаються алгоритми роботи з динамічними структурами, включаючи додавання, видалення, пошук та оновлення елементів. Вивчаються різні способи реалізації динамічних структур за допомогою стандартних бібліотек мов програмування, а також створення власних реалізацій для забезпечення високої ефективності та адаптивності коду. Окрему увагу приділено оцінці часової та просторової складності алгоритмів для визначення їхньої оптимальності у різних сценаріях використання.

Джерела:

- Декілька відео на YouTube:
  - Динамічні структури - https://www.youtube.com/watch?v=A3ZUpyrnCbM&t=1s
  - Основи динамічного розподілу пам'яті - https://www.youtube.com/watch?v=udfbq4M2Kfc
  - Стек - https://www.youtube.com/watch?v=GBST5uQ_yos

- Певну інформацію брав на сайтах:
  - Динамічні структури - https://studfile.net/preview/7013685/page:10/

- Основи динамічного розподілу пам'яті - https://studfile.net/preview/9189217/page:15/
- Стек - https://itproger.com/ua/spravka/cpp/stack

- Також вивчив багато інформації за допомогою ChatGPT.

Виконання роботи:

- **Завдання №1:** Theory Education Activities

Очікувано часу: **4 дні.**
Витрачено часу: **3 дні.**

- **Завдання №2:** Requirements management (understand tasks) and design activities (draw flow diagrams and estimate tasks 3-7)

Очікувано часу: **1 година.**
Витрачено часу: **1 година.**

- **Завдання №3:** Lab# programming: VNS Lab 10 (Variant 19)
  Умова:

  19. Записи в лінійному списку містять ключове поле типу *char (рядок символів). Сформувати двонаправлений список. Знищити K елементів із заданими номерами. Додати K елементів у початок списку.

Код:

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

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

Node* CreateNewNode(const char* str)
{
    // 1. Виділяємо пам'ять
    Node* newNode = new Node;

    // 2. Копіюємо рядок у вузол
    newNode->data = new char[strlen(str) + 1];
    strcpy(newNode->data, str);

    // 3. Ініціалізуємо вказівники
    newNode->next = nullptr;
    newNode->prev = nullptr;

    // 4. Повертаємо адресу нового вузла
    return newNode;
}

void AddNodeToFront(Node*& head, const char* str)
{
    // 1. Створюємо новий вузол
    Node* newNode = CreateNewNode(str);

    // 2. Якщо список не порожній — оновлюємо вказівники
    if(head != nullptr)
    {
        head->prev = newNode;
        newNode->next = head;
    }

    // Змінюємо голову списку
    head = newNode;
}

void DeleteNode(Node*& head, int index)
{
    if (head == nullptr)
    {
        return;
    }

    Node* current = head;
    int currentIndex = 0;

    while (current != nullptr && index > 0)
    {
        current = current->next;
        currentIndex++;
    }
```

```cpp
    if (current == nullptr)
    {
        return;
    }

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

    delete[] current->data;
    delete current;

}

void DeleteKNodes(Node*& head, int K, const int* indices)
{
    for (int i = 0; i < K; i++)
    {
        DeleteNode(head, indices[i]-1);
    }
}

void AddKNodes(Node*& head, int K, const char** data)
{
    for (int i = 0; i < K; i++)
    {
        AddNodeToFront(head, data[i]);
    }
}

int main()
{
    int K; // Кількість елементів щоб видалити та додати
    int* indices; // Масив індексів елементів щоб видалити
    Node* head = nullptr;

    AddNodeToFront(head, "Hello");
    AddNodeToFront(head, "World");
    AddNodeToFront(head, "How are you?");
    AddNodeToFront(head, "I am fine");
    AddNodeToFront(head, "Thank you");

    cout << "First list: ";
    Node* current = head;
    while (current != nullptr)
    {
        cout << current->data << " ";
        current = current->next;
    }
```

```
    }
    cout << endl;

    cout << "Enter the number of elements to delete and add: ";
    cin >> K;
    indices = new int[K];
    cout << "Enter the indices of elements to delete: ";
    for (int i = 0; i < K; i++)
    {
        cin >> indices[i];
    }

    DeleteKNodes(head, K, indices);

    const char* newData[] = {"Good", "Morning", "My", "Friend"};
    AddKNodes(head, K, newData);

    cout << "Second list: ";
    current = head;
    while (current != nullptr)
    {
        cout << current->data << " ";
        current = current->next;
    }
    cout << endl;

    delete[] indices;
    while (head != nullptr)
    {
        Node* temp = head;
        head = head->next;
        delete[] temp->data;
        delete temp;
    }

    return 0;
}
```

Результат:

```
First list: Thank you I am fine How are you? World Hello
Enter the number of elements to delete and add: 2
Enter the indices of elements to delete: 2 3
Second list: Morning Good Thank you I am fine How are you? World Hello
PS D:\VS Code\Projects>
```

Очікувано часу: **1 година.**
Витрачено часу: **2 години.**

- **Завдання №4:** Lab# programming: Algotester Lab 5 (Variant 2)

Умова:

Lab 5v2

Обмеження: 1 сек., 256 MiB

В пустелі існує незвичайна печера, яка є двохвимірною. Її висота це N, ширина - M.
Всередині печери є пустота, пісок та каміння. Пустота позначається буквою , пісок S і каміння X;
Одного дня стався землетрус і весь пісок посипався вниз. Він падає на найнижчу клітинку з пустотою, але він не може пролетіти через каміння.
Ваше завдання сказати як буде виглядати печера після землетрусу.

Вхідні дані

У першому рядку 2 цілих числа N та M - висота та ширина печери
У N наступних рядках стрічка row_i яка складається з N цифер - і-й рядок матриці, яка відображає стан печери до землетрусу.

Вихідні дані

N рядків, які складаються з стрічки розміром M - стан печери після землетрусу.

Обмеження

$1 \leq N, M \leq 1000$
$|row_i| = M$
$row_i \in \{X, S, O\}$

Приклади

Вхідні дані (*stdin*)	Вихідні дані (*stdout*)
5 5 SSOSS OOOOO SOOXX OOOOS OOSOO	OOOOO OOOSS OOOXX SOOOO SSSOS

Посилання:

https://algotester.com/uk/ContestProblem/DisplayWithEditor/135602

Код:

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main()
{
    int M; // висота
    int N; // ширина
    do
    {
    cin >> M >> N;;
    }while(!(N >= 1 && N <= 1000 && M >= 1 && M <= 1000));

    vector<vector<char>> cave(M, vector<char>(N));

    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            char material;
            cin >> material;
            if (material == 'S' || material == 'O' || material == 'X')
            {
                cave[i][j] = material;
            }
        }
    }

    for (int i = M - 2; i >= 0; i--) // M-2, бо останній рядок не обробляється
    {
        for (int j = 0; j < N; j++)
        {
            if (cave[i][j] == 'S')
            {
                int k = i;
                while (k + 1 < M && cave[k + 1][j] == 'O')
                {
                    cave[k + 1][j] = 'S';
                    cave[k][j] = 'O';
                    k++;
                }
            }
        }
    }
    cout << endl;

    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            cout << cave[i][j];
        }
        cout << endl;
    }
}
```

Результат:

```
3 3
SSS
OXO
OOX

OSO
OXS
SOX
PS D:\VS Code\Projects>
```

Очікувано часу: **1 година.**
Витрачено часу: **1 година.**

- **Завдання №5:** Lab# programming: Algotester Lab 7-8 (Variant 1,3)

Варіант – 1

Lab 78v1

Обмеження: 1 сек., 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 <<

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.

Посилання:
https://algotester.com/uk/ContestProblem/DisplayWithEditor/135607

Код:

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

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

    // Конструктор вузла
    Node(int val)
    {
        data = val;
        prev = nullptr;
        next = nullptr;
    }
};

struct DoublyLinkedList
{
    Node* head;
    Node* tail;
    int size;
    DoublyLinkedList()
    {
        head = nullptr;
        tail = nullptr;
        size = 0;
    }


    void insert(int index, const std::vector<int>& values)
    {
        if (index < 0 || index > size)
        {
            return;
        }

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

        for (int value : values)
        {
            Node* new_node = new Node(value);

            if (!head)
            {
                head = tail = new_node;
            }
            else if (current == head)
            { // початок
                new_node->next = head;
                head->prev = new_node;
                head = new_node;
            }
            else if (!current)
            { // кінець
                new_node->prev = tail;
                tail->next = new_node;
                tail = new_node;
            }
            else
            { // середина
                new_node->prev = current->prev;
                new_node->next = current;
                current->prev->next = new_node;
                current->prev = new_node;
            }

            size++;
        }
    }


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

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

```
        }

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

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

            delete to_delete;
            size--;
        }
    }

    int get(int index)
    {
        if (index < 0 || index >= size) return -1;

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

    // Змінити значення за індексом
    void set(int index, int value)
    {
        if (index < 0 || index >= size) return;

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

    int getSize()
    {
        return size;
    }

    void print()
    {
        Node* current = head;
        while (current)
        {
            std::cout << current->data << " ";
            current = current->next;
        }
        std::cout << std::endl;
    }
};

int main()
{
    DoublyLinkedList list;
    int Q;
    std::cin >> Q;

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

        if (command == "insert")
        {
            int index, n;
            std::cin >> index >> n;
            std::vector<int> values(n);
            for (int i = 0; i < n; ++i)
            {
                std::cin >> values[i];
            }
            list.insert(index, values);
        }
        else if (command == "erase")
        {
            int index, n;
            std::cin >> index >> n;
            list.erase(index, n);
        }
        else if (command == "size")
        {
            std::cout << list.getSize() << std::endl;
        }
        else if (command == "get")
        {
            int index;
            std::cin >> index;
            std::cout << list.get(index) << std::endl;
        }
        else if (command == "set")
        {
            int index, value;
            std::cin >> index >> value;
            list.set(index, value);
        }
        else if (command == "print")
        {
            list.print();
        }
    }

    return 0;
}
```

Результат:

```
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
PS D:\VS Code\Projects>
```

Очікувано часу: **1 година.**
Витрачено часу: **2 години.**

Варіант – 2

Lab 78v3

Обмеження: 1 сек., 256 МіБ

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.

Посилання:
https://algotester.com/uk/ContestProblem/DisplayWithEditor/135609

Код:

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

struct Node
{
    int data;
    Node* left;
    Node* right;

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


struct BinarySearchTree
{
    Node* root;
    int size;

    BinarySearchTree()
    {
        root = nullptr;
        size = 0;
    }

    void insert(int value)
    {
        if (!contains(value))
        {
            root = insertRec(root, value);
            size++;
        }
    }

    Node* insertRec(Node* node, int value)
    {
        if (node == nullptr)
        {
            return new Node(value);
        }

        if (value < node->data)
        {
            node->left = insertRec(node->left, value);
        }
        else
        {
            node->right = insertRec(node->right, value);
        }

        return node;
    }

    bool contains(int value)
    {
        return containsRec(root, value);
    }

    bool containsRec(Node* node, int value)
    {
        if (node == nullptr)
        {
            return false;
        }
        if (node->data == value)
        {
            return true;
        }
        if (value < node->data)
        {
            return containsRec(node->left, value);
        }
        else
        {
            return containsRec(node->right, value);
        }
    }

```

```
    int getSize()
    {
        return size;
    }

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

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

int main() {
    BinarySearchTree tree;
    int Q;
    cin >> Q;

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

        if (command == "insert")
        {
            int value;
            cin >> value;
            tree.insert(value);
        }
        else if (command == "contains")
        {
            int value;
            cin >> value;
            if (tree.contains(value))
            {
                cout << "Yes" << endl;
            }
            else
            {
                cout << "No" << endl;
            }
        }
        else if (command == "size")
        {
            cout << tree.getSize() << endl;
        }
        else if (command == "print")
        {
            tree.print();
        }
    }

    return 0;
}
```

Результат:

```
11

size
0
insert 5
insert 4
print
4 5
insert 5
print
4 5
insert 1
print
1 4 5
contains 5
Yes
contains 0
No
size
3
PS D:\VS Code\Projects>
```

Очікувано часу: **1 година.**
Витрачено часу: **2 години.**

- **Завдання №6:** Practice# programming: Class Practice Task

Задача №1:

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;

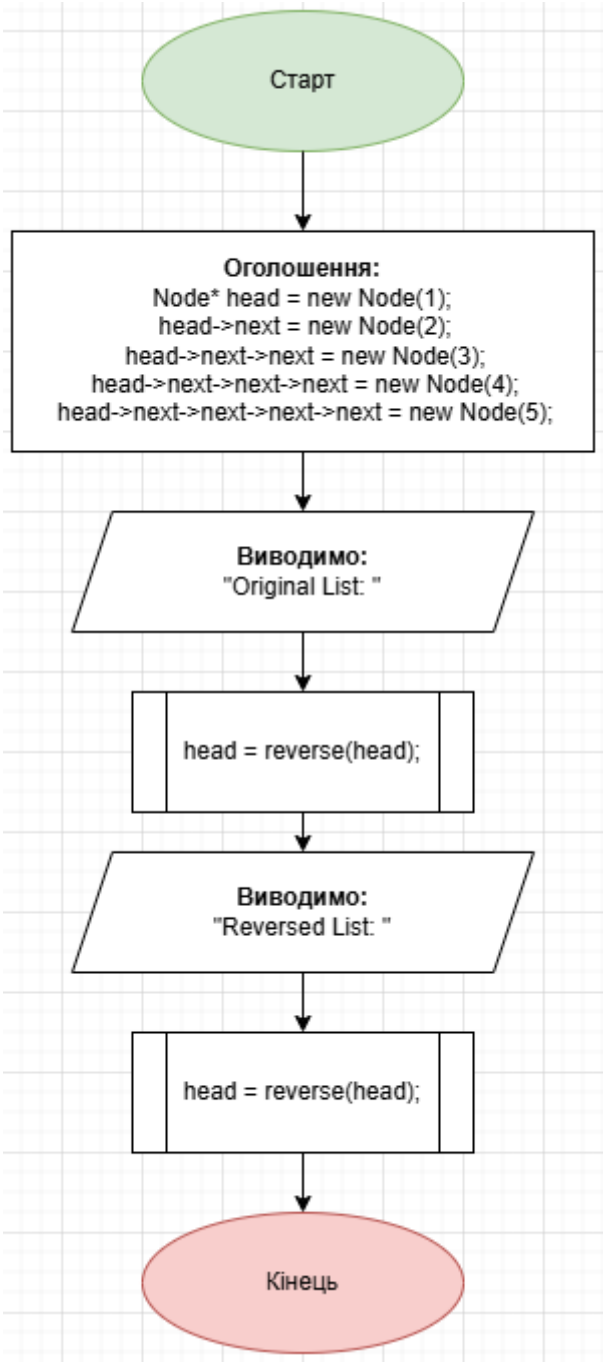
Код:

```cpp
#include <iostream>
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;
    Node* next = nullptr;

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

    return prev;
}

void printList(Node* head)
{
    Node* temp = head;
    while (temp != nullptr)
    {
        cout << temp->data << " ";
        temp = temp->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);
    head->next->next->next->next = new Node(5);

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

    head = reverse(head);
    cout << "Reversed List: ";
    printList(head);

    return 0;
}
```

Результат:

```
Original List: 1 2 3 4 5
Reversed List: 5 4 3 2 1
PS D:\VS Code\Projects>
```

Блок-схема:



Задача №2:

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***.

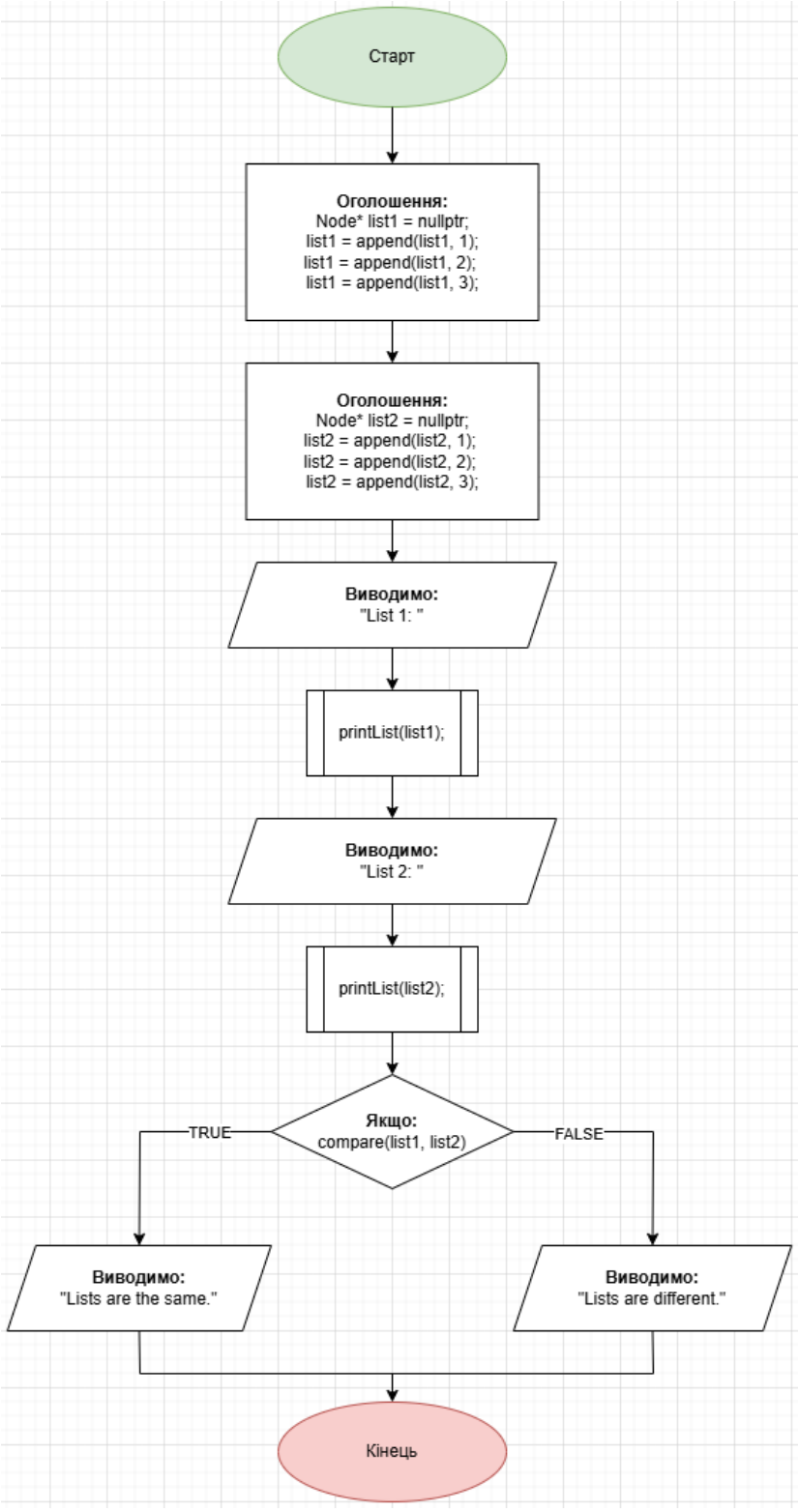
Код:

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

struct Node
{
    int data;
    Node* next;

    Node(int val)
    {
        data = val;
        next = nullptr;
    }
};

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

    return h1 == nullptr && h2 == nullptr;
}

Node* append(Node* head, int value)
{
    if (head == nullptr)
    {
        return new Node(value);
    }
    Node* temp = head;
    while (temp->next != nullptr)
    {
        temp = temp->next;
    }
    temp->next = new Node(value);
    return head;
}

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

int main()
{
    Node* list1 = nullptr;
    list1 = append(list1, 1);
    list1 = append(list1, 2);
    list1 = append(list1, 3);

    Node* list2 = nullptr;
    list2 = append(list2, 1);
    list2 = append(list2, 2);
    list2 = append(list2, 3);

    cout << "List 1: ";
    printList(list1);
    cout << "List 2: ";
    printList(list2);

    if (compare(list1, list2))
    {
        cout << "Lists are the same." << endl;
    }
    else
    {
        cout << "Lists are different." << endl;
    }

    return 0;
}
```

Результат:

```
List 1: 1 2 3
List 2: 1 2 3
Lists are the same.
PS D:\VS Code\Projects>
```

Блок-схема:



Задача №3:

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.

Код:

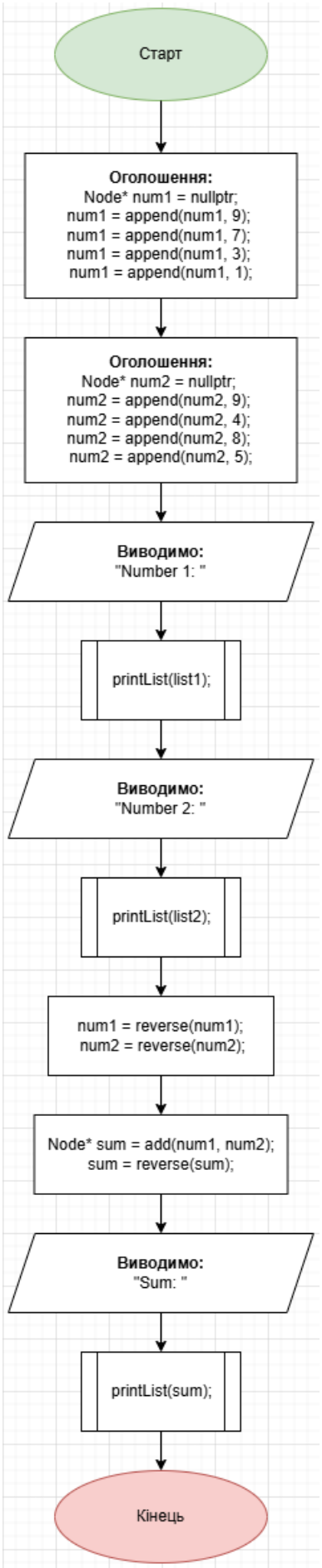
```cpp
#include <iostream>
using namespace std;

struct Node
{
    int data;
    Node* next;

    Node(int val)
    {
        data = val;
        next = nullptr;
    }
};

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

    while (n1 != nullptr || n2 != nullptr || carry > 0)
    {
        int sum = carry;

        if (n1 != nullptr)
        {
            sum += n1->data;
            n1 = n1->next;
        }

        if (n2 != nullptr)
        {
            sum += n2->data;
            n2 = n2->next;
        }

        carry = sum / 10;
        int digit = sum % 10;

        Node* newNode = new Node(digit);

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

    return result;
}

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

Node* append(Node* head, int value)
{
    if (head == nullptr)
    {
        return new Node(value);
    }
    Node* temp = head;
    while (temp->next != nullptr)
    {
        temp = temp->next;
    }
    temp->next = new Node(value);
    return head;
}
```

```
86
87      void printList(Node* head)
88      {
89          Node* temp = head;
90          while (temp != nullptr)
91          {
92              cout << temp->data;
93              temp = temp->next;
94          }
95          cout << endl;
96      }
97
98      int main()
99      {
100         Node* num1 = nullptr;
101         num1 = append(num1, 9);
102         num1 = append(num1, 7);
103         num1 = append(num1, 3);
104         num1 = append(num1, 1);
105
106         Node* num2 = nullptr;
107         num2 = append(num2, 9);
108         num2 = append(num2, 4);
109         num2 = append(num2, 8);
110         num2 = append(num2, 5);
111
112         cout << "Number 1: ";
113         printList(num1);
114         cout << "Number 2: ";
115         printList(num2);
116
117         num1 = reverse(num1);
118         num2 = reverse(num2);
119
120         Node* sum = add(num1, num2);
121
122         sum = reverse(sum);
123
124         cout << "Sum: ";
125         printList(sum);
126
127         return 0;
128     }
129
```

Результат:

```
Number 1: 9731
Number 2: 9485
Sum: 19216
PS D:\VS Code\Projects>
```

Блок-схема:


Старт
Оголошення:
Node* num1 = nullptr;
num1 = append(num1, 9);
num1 = append(num1, 7);
num1 = append(num1, 3);
num1 = append(num1, 1);
Оголошення:
Node* num2 = nullptr;
num2 = append(num2, 9);
num2 = append(num2, 4);
num2 = append(num2, 8);
num2 = append(num2, 5);
Виводимо:
"Number 1: "
printList(list1);
Виводимо:
"Number 2: "
printList(list2);
num1 = reverse(num1);
num2 = reverse(num2);
Node* sum = add(num1, num2);
sum = reverse(sum);
Виводимо:
"Sum: "
printList(sum);
Кінець

Задача №4:

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

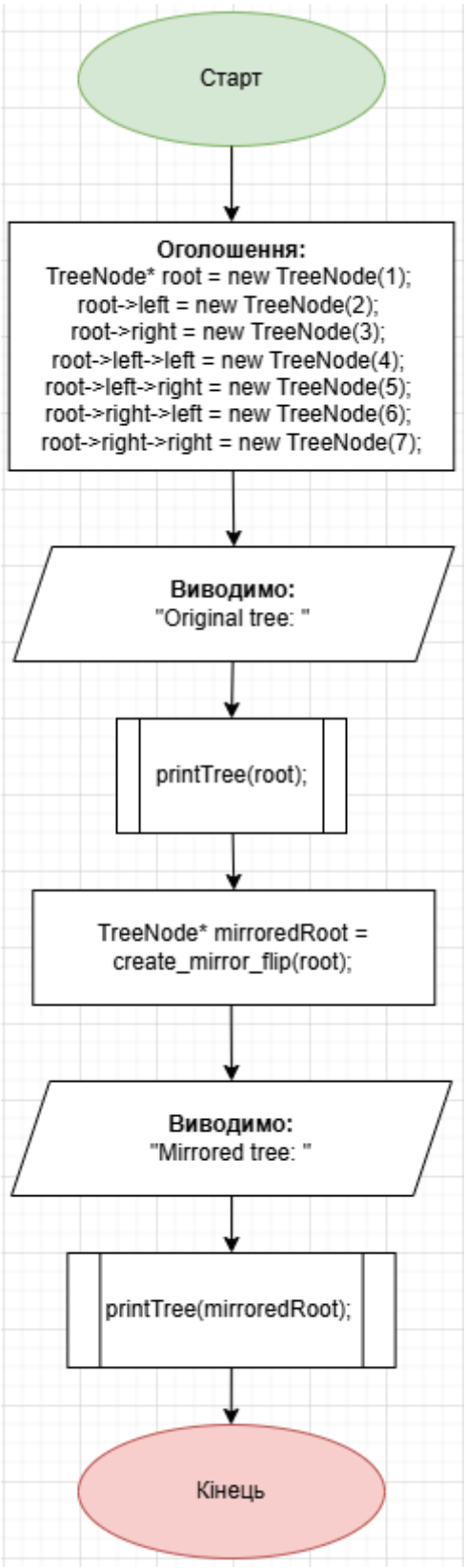
Код:

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

struct TreeNode
{
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int val)
    {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

TreeNode* create_mirror_flip(TreeNode* root)
{
    if (root == nullptr)
    {
        return nullptr;
    }

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

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

    return mirroredNode;
}

void printTree(TreeNode* root)
{
    if (root == nullptr)
    {
        return;
    }
    cout << root->data << " ";
    printTree(root->left);
    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);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);

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

    TreeNode* mirroredRoot = create_mirror_flip(root);

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

    return 0;
}
```

Результат:

```
Original tree: 1 2 4 5 3 6 7
Mirrored tree: 1 3 7 6 2 5 4
PS D:\VS Code\Projects>
```

Блок-схема:



Задача №5:

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

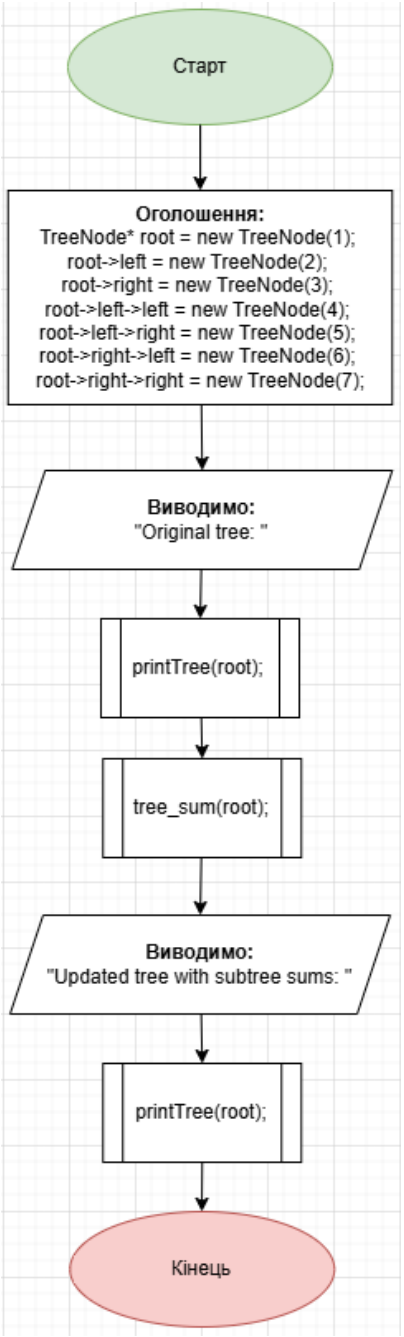
Код:

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

struct TreeNode
{
    int data;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int val)
    {
        data = val;
        left = nullptr;
        right = nullptr;
    }
};

int calculateSubtreeSum(TreeNode* root)
{
    if (root == nullptr)
    {
        return 0;
    }

    if (root->left == nullptr && root->right == nullptr)
    {
        return root->data;
    }

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

    root->data = leftSum + rightSum;

    return root->data;
}

void tree_sum(TreeNode* root) {
    calculateSubtreeSum(root);
}

void printTree(TreeNode* root)
{
    if (root == nullptr)
    {
        return;
    }
    cout << root->data << " ";
    printTree(root->left);
    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);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(7);

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

    tree_sum(root);

    cout << "Updated tree with subtree sums: ";
    printTree(root);
    cout << endl;

    return 0;
}
```

Результат:

```
Original tree: 1 2 4 5 3 6 7
Updated tree with subtree sums: 22 9 4 5 13 6 7
PS D:\VS Code\Projects>
```

Блок-схема:



Очікувано часу: **2 години.**
Витрачено часу: **4 години.**

- **Завдання №7:** Practice# programming: Self Practice Task

Цікава гра

Обмеження: 2 сек., 256 МіБ

Мале Бісеня та Дракон полюбляють проводити дозвілля разом. Сьогодні вони грають в одну дуже цікаву гру.

У них є дошка, що складається з n рядків та m стовпців, всі клітинки якої білі.

Гравці по черзі вибирають одну білу клітинку та зафарбовують її в чорний колір. Бісеня ходить першим. Гравець, який не може зробити хід, тобто на початку ходу якого вся дошка чорна, програє.

Погостривши зубки, Бісеня зрозуміло, що у Дракона велика перевага, адже він двоголовий, а, як то кажуть, «одна голова добре, а дві — краще». Тому воно просить вас допомогти. Вам потрібно сказати за заданими n та m, хто виграє у цій напруженій грі.

Вхідні дані

У єдиному рядку задані два цілих числа n та m — розміри дошки.

Вихідні дані

Єдине слово — `Imp`, якщо переможе Бісеня, та `Dragon`, якщо переможе Дракон.

Обмеження

$1 \le n, m \le 100$.

Приклади

Вхідні дані (*stdin*)	Вихідні дані (*stdout*)
7 4	Dragon

Посилання:https://algotester.com/uk/ArchiveProblem/DisplayWithEditor/20074

Код:

```cpp
#include <iostream>

int main()
{
    int m, n;
    std::cin >> m >> n;

    if((m * n) % 2 == 0)
    {
        std::cout << "Dragon" << std::endl;
    }
    else
    {
        std::cout << "Imp" << std::endl;
    }

    return 0;
}
```

Результат:

```
7 4
Dragon
PS D:\VS Code\Projects>
```

Очікувано часу: **30 хвилин.**
Витрачено часу: **15 хвилин.**

Pull-Request:

Висновок: У результаті виконання цієї роботи я ознайомився з основами роботи з динамічними структурами даних, такими як черга, стек, списки та дерево. Я вивчив їхню внутрішню організацію, ключові властивості і методи доступу, а також алгоритми для додавання, видалення, пошуку та сортування елементів. Особливу увагу я приділив практичному використанню стандартних бібліотек і створенню власних оптимізованих реалізацій. Отримані знання допоможуть мені ефективно використовувати динамічні структури даних у програмуванні.