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 Лабораторних Робіт № 5, 7-8
Практичних Робіт до блоку № 6

Виконав:
Студент групи ШІ-12
Горішний Микола

Львів 2024

Тема роботи

Динамічні структури, види динамічних структур, їх використання, алгоритми їх обробки.

Мета роботи

1. Навчитись створювати та використовувати динамічні структури, такі як: Списки, Дерево.
2. Навчитись виконувати алгоритми обробки динамічний структур.

Джерела:

-- https://www.youtube.com/watch?v=CCBCUQY8HNo;
-- https://studfile.net/preview/7013685/page:10/;
-- https://acode.com.ua/urok-89-dynamichne-vydilennya-pam-yati/;
-- https://www.youtube.com/watch?v=qBFzNW0ALxQ

Виконання роботи

Завдання №1 -VNS lab 10 Task 1 (3)

2. Постановка завдання

Написати програму, у якій створюються динамічні структури й виконати їхню обробку у відповідності зі своїм варіантом.

Для кожного варіанту розробити такі функції:

1. Створення списку.
2. Додавання елемента в список (у відповідності зі своїм варіантом).
3. Знищення елемента зі списку (у відповідності зі своїм варіантом).
4. Друк списку.
5. Запис списку у файл.
6. Знищення списку.
7. Відновлення списку з файлу.

3. Написати функції для знищення й додавання елементів списку у відповідності зі своїм варіантом.

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

using namespace std;

struct Node
{
    int data;
    unique_ptr<Node> next;

    Node(int val) : data(val), next(nullptr)
    {
    }
};
class LinkedList
{
private:
    unique_ptr<Node> head;

public:
    LinkedList() : head(nullptr)
    {
    }

    void addElement(int value)
    {
        unique_ptr<Node> newNode = make_unique<Node>(value);
        if (!head || value < head->data)
        {
            newNode->next = move(head);
            head = move(newNode);
            return;
        }

        Node* current = head.get();
        while (current->next && current->next->data < value)
        {
            current = current->next.get();
        }
```

```
        }
        newNode->next = move(current->next);
        current->next = move(newNode);
    }

    void deleteElementsLessThan(int threshold)
    {
        while (head && head->data < threshold)
        {
            head = move(head->next);
        }

        Node* current = head.get();
        while (current && current->next)
        {
            if (current->next->data < threshold)
            {
                current->next = move(current->next->next);
            }
            else {
                current = current->next.get();
            }
        }
    }

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

int main()
{
```

```
int main()
{
    LinkedList list;

    list.addElement(100);
    list.addElement(50);
    list.addElement(20);

    cout << "List after adding elements: ";
    list.printList();

    list.deleteElementsLessThan(10);
    cout << "List after deleting elements < 10: ";
    list.printList();

    return 0;
}
```

```
Your first list : 5 10 20 70
Last list with numbers < 10 : 10 20 70
```

Завдання №2 **A**lgotester lab 5 (1)

Lab 5v1

Обмеження: 2 сек., 256 МіБ

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

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

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

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

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

void makeBoard(bitset<64> binBoard, int board[8][8])
{
    int count = 0;
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            board[i][j] = binBoard[count];
            count++;
        }
    }
}

int main()
{
    uint64_t a;
    cin >> a;
    bitset<64> binBoard(a);
    int board[8][8] = {};
    makeBoard(binBoard, board);
    int questions;
    cin >> questions;
    int coord[questions][2] = {};
    for (int i = 0; i < questions; i++)
    {
        cin >> coord[i][0] >> coord[i][1];
        coord[i][0]--;
        coord[i][1]--;
    }
    int row, column;
    for (int i = 0; i < questions; i++)
    {
        row = coord[i][0];
```

```
        column = coord[i][1];
        board[row][column] = (board[row][column] == 1) ? 0 : 1;
        for (int j = 0; j < 8; j++)
        {
            board[row][j] = (board[row][j] == 1) ? 0 : 1;
            board[j][column] = (board[j][column] == 1) ? 0 : 1;
        }
    };

    uint64_t result = 0;

    int count = 0;
    for (int i = 0; i < 8; i++)
    {
        for (int j = 0; j < 8; j++)
        {
            if (board[i][j] == 1)
            {
                uint64_t c = pow(2, count);
                result += c;
            }
            count++;
        }
    }
    cout << result;

    return 0;
}
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
декілька секунд тому	C++ 23	Зараховано	0.003	1.367	Перегляд

Завдання №3 Algotester lab 7-8 /3.1 / 3.2 (2)

Lab 78v1

Обмеження: 2 сек., 256 МіБ

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 $index$.

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

template <typename T>
struct Node
{
    T value;
    Node* next;
    Node* prev;

    Node(T val) : value(val), next(nullptr), prev(nullptr)
    {
    }
};

template <typename T>
class DoublyLinkedList
{
private:
    Node<T>* head;
    Node<T>* tail;
    int count;

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

    ~DoublyLinkedList()
    {
        Node<T>* current_node = head;
        while (current_node) {
            Node<T>* next_node = current_node->next;
            delete current_node;
            current_node = next_node;
        }
    }

    void insert(int index, const T arr[], int N)
    {
```

```cpp
void insert(int index, const T arr[], int N)
{
    Node<T>* new_nodes_start = nullptr;
    Node<T>* new_nodes_end = nullptr;

    for (int i = 0; i < N; ++i) {
        Node<T>* new_node = new Node<T>(arr[i]);
        if (!new_nodes_start) new_nodes_start = new_node;
        else {
            new_nodes_end->next = new_node;
            new_node->prev = new_nodes_end;
        }
        new_nodes_end = new_node;
    }

    if (index == 0)
    {
        new_nodes_end->next = head;
        if (head) head->prev = new_nodes_end;
        head = new_nodes_start;
        if (!tail) tail = new_nodes_end;
    }
    else {
        Node<T>* current_node = head;
        int curr_index = 0;

        while (current_node && curr_index < index) {
            current_node = current_node->next;
            curr_index++;
        }

        if (current_node) {
            new_nodes_end->next = current_node;
            if (current_node->prev) current_node->prev->next = new_nodes_start;
            new_nodes_start->prev = current_node->prev;
            current_node->prev = new_nodes_end;
        }
    }
    count += N;
}
```

```
        count += N;
    }

    void erase(int index, int n)
    {
        Node<T>* current_node = head;
        int curr_index = 0;

        while (current_node && curr_index < index) {
            current_node = current_node->next;
            curr_index++;
        }

        if (!current_node) return;

        for (int i = 0; i < n && current_node; ++i)
        {
            Node<T>* next_node = current_node->next;

            if (current_node == head) head = next_node;
            if (current_node == tail) tail = current_node->prev;

            if (next_node) next_node->prev = current_node->prev;
            if (current_node->prev) current_node->prev->next = next_node;

            delete current_node;
            current_node = next_node;
            count--;
        }
    }

    int size() const { return count; }

    void get(int index) const {
        Node<T>* current_node = head;
        int curr_index = 0;

        while (current_node && curr_index < index) {
            current_node = current_node->next;
```

```
            curr_index++;
        }

        if (current_node) cout << current_node->value << endl;
        else cout << "Invalid index" << endl;
    }

    void set(int index, T value)
    {
        Node<T>* current_node = head;
        int curr_index = 0;

        while (current_node && curr_index < index) {
            current_node = current_node->next;
            curr_index++;
        }

        if (current_node) current_node->value = value;
    }

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

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

    DoublyLinkedList<int> list;

    for (int i = 0; i < Q; ++i) {
```

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

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

            int arr[N];
            for (int j = 0; j < N; ++j) {
                cin >> arr[j];
            }
            list.insert(index, arr, N);
        }
        else if (command == "erase") {
            int index, n;
            cin >> index >> n;
            list.erase(index, n);
        }
        else if (command == "size") {
            cout << list.size() << endl;
        }
        else if (command == "get") {
            int index;
            cin >> index;
            list.get(index);
        }
        else if (command == "set") {
            int index, value;
            cin >> index >> value;
            list.set(index, value);
        }
        else if (command == "print") {
            list.print();
        }
    }
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
годину тому	C++ 23	Зараховано	0.008	1.449	Перегляд

Lab 78v2

Обмеження: 1 сек., 256 МіБ

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 $index$.

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

template <typename T>
struct Node
{
    T value;
    Node* next;
    Node* prev;

    Node(T val) : value(val), next(nullptr), prev(nullptr)
    {
    }
};

template <typename T>
class DoublyLinkedList
{
private:
    Node<T>* head;
    Node<T>* tail;
    int count;

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

    ~DoublyLinkedList()
    {
        Node<T>* current_node = head;
        while (current_node) {
            Node<T>* next_node = current_node->next;
            delete current_node;
            current_node = next_node;
        }
    }

    void insert(int index, const T arr[], int N)
    {
```

```cpp
void insert(int index, const T arr[], int N)
{
    Node<T>* new_nodes_start = nullptr;
    Node<T>* new_nodes_end = nullptr;

    for (int i = 0; i < N; ++i)
    {
        Node<T>* new_node = new Node<T>(arr[i]);
        if (!new_nodes_start) new_nodes_start = new_node;
        else {
            new_nodes_end->next = new_node;
            new_node->prev = new_nodes_end;
        }
        new_nodes_end = new_node;
    }

    if (index == 0) {
        new_nodes_end->next = head;
        if (head) head->prev = new_nodes_end;
        head = new_nodes_start;
        if (!tail) tail = new_nodes_end;
    }
    else {
        Node<T>* current_node = head;
        int curr_index = 0;

        while (current_node && curr_index < index) {
            current_node = current_node->next;
            curr_index++;
        }

        if (current_node) {
            new_nodes_end->next = current_node;
            if (current_node->prev) current_node->prev->next = new_nodes_start;
            new_nodes_start->prev = current_node->prev;
            current_node->prev = new_nodes_end;
```

```
            current_node->prev = new_nodes_end;
        }
    }
    count += N;
}

void erase(int index, int n) {
    Node<T>* current_node = head;
    int curr_index = 0;

    while (current_node && curr_index < index) {
        current_node = current_node->next;
        curr_index++;
    }

    if (!current_node) return;

    for (int i = 0; i < n && current_node; ++i) {
        Node<T>* next_node = current_node->next;

        if (current_node == head) head = next_node;
        if (current_node == tail) tail = current_node->prev;

        if (next_node) next_node->prev = current_node->prev;
        if (current_node->prev) current_node->prev->next = next_node;

        delete current_node;
        current_node = next_node;
        count--;
    }
}

int size() const { return count; }

void get(int index) const {
    Node<T>* current_node = head;
    int curr_index = 0;

    while (current_node && curr_index < index)
    {
```

```
            curr_index++;
        }

        if (current_node) cout << current_node->value << endl;
        else cout << "Invalid index" << endl;
    }

    void set(int index, T value)
    {
        Node<T>* current_node = head;
        int curr_index = 0;

        while (current_node && curr_index < index) {
            current_node = current_node->next;
            curr_index++;
        }

        if (current_node) current_node->value = value;
    }

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

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

    DoublyLinkedList<int> list;

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

```
            int N;
            cin >> N;

            int arr[N];
            for (int j = 0; j < N; ++j) {
                cin >> arr[j];
            }

            list.insert(index, arr, N);
        }
        else if (command == "erase") {
            int index, n;
            cin >> index >> n;
            list.erase(index, n);
        }
        else if (command == "size") {
            cout << list.size() << endl;
        }
        else if (command == "get") {
            int index;
            cin >> index;
            list.get(index);
        }
        else if (command == "set") {
            int index, value;
            cin >> index >> value;
            list.set(index, value);
        }
        else if (command == "print")
        {
            list.print();
        }
    }

    return 0;
}
```

Створено	Компілятор	Результат	Час (сек.)	Пам'ять (МіБ)	Дії
5 годин тому	C++ 23	Зараховано	0.006	1.391	Перегляд

Завдання №4 Class Practice Work

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;

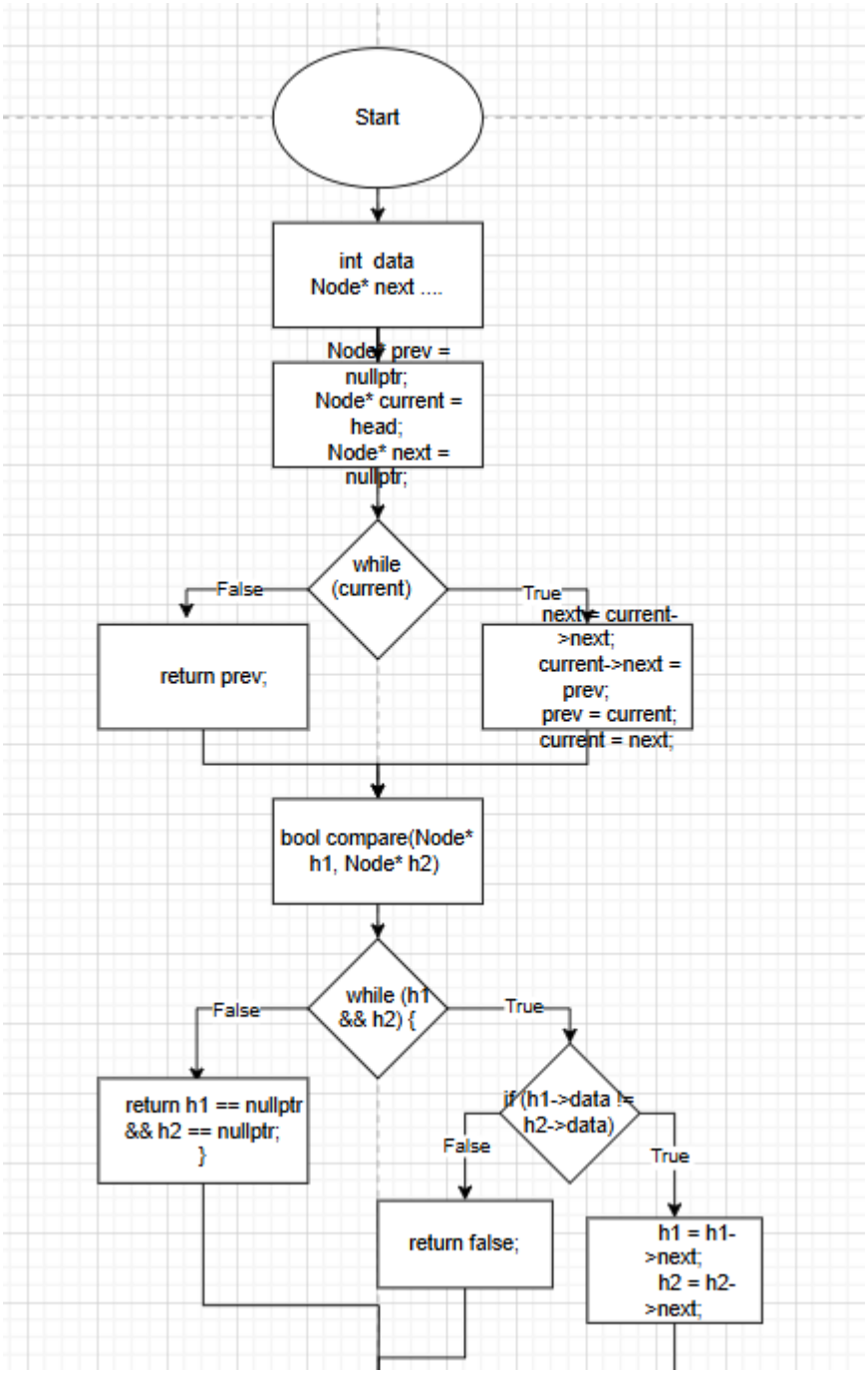
Мета задачі

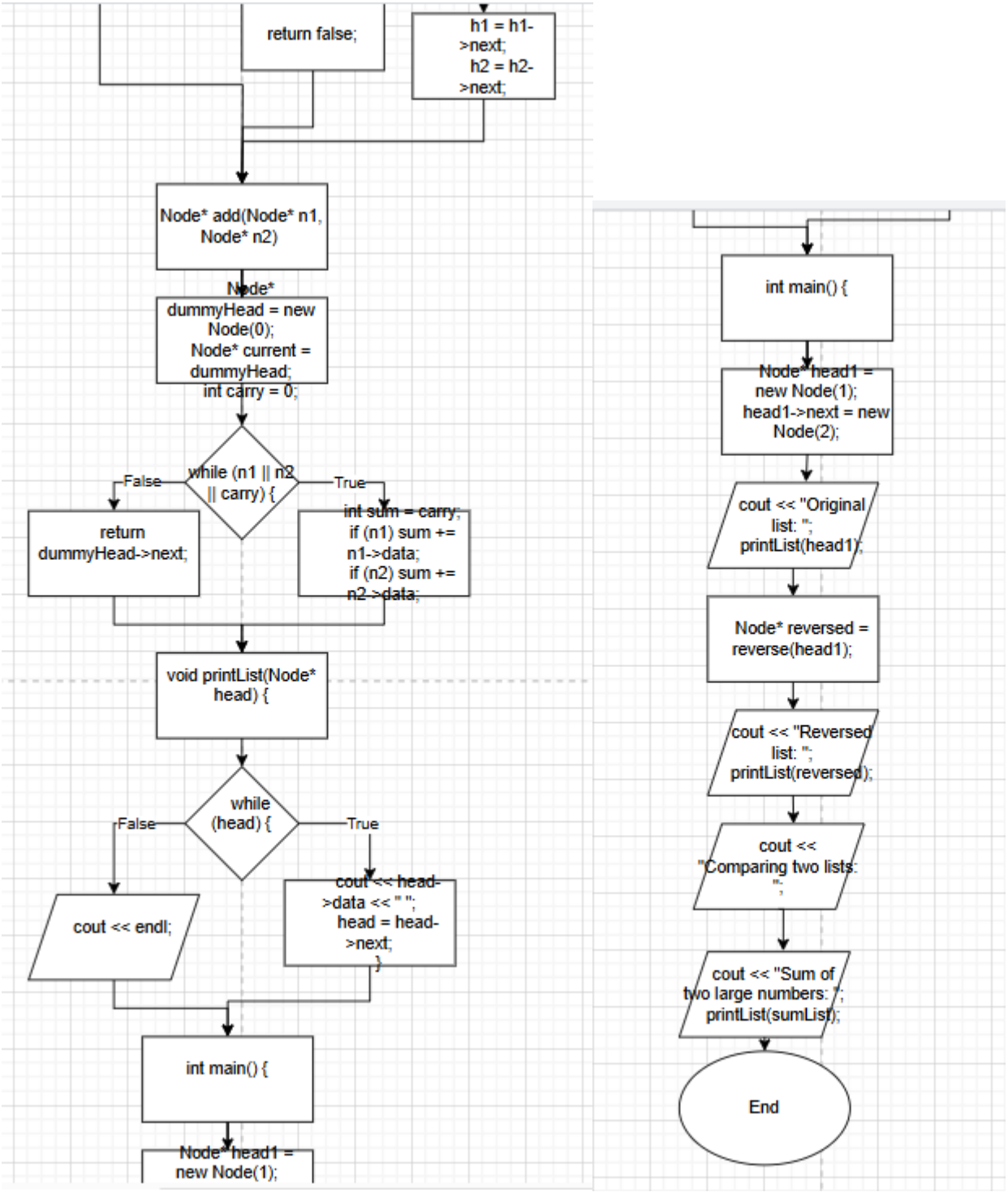
Розуміння структур даних: Реалізація методу реверсу для зв'язаних списків є чудовим способом для поглиблення розуміння зв'язаних списків як фундаментальної структури даних. Він заохочує практичний підхід до вивчення того, як структуруються пов'язані списки та як ними маніпулювати.

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

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

Node* reverse(Node* head)
{
    Node* prev = nullptr;
    Node* current = head;
    Node* next = nullptr;

    while (current)
    {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
    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* dummyHead = new Node(0);
    Node* current = dummyHead;
```

```
    Node* dummyHead = new Node(0);
    Node* current = dummyHead;
    int carry = 0;

    while (n1 || n2 || carry) {
        int sum = carry;
        if (n1) sum += n1->data;
        if (n2) sum += n2->data;

        carry = sum / 10;
        sum %= 10;

        current->next = new Node(sum);
        current = current->next;

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

    return dummyHead->next;
}

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

int main() {
    //Taks 1
    Node* head1 = new Node(1);
    head1->next = new Node(2);
    head1->next->next = new Node(3);
    head1->next->next->next = new Node(4);
    head1->next->next->next->next = new Node(5);

    cout << "Original list: ";
    printList(head1);
```

```
    cout << "Original list: ";
    printList(head1);

    Node* reversed = reverse(head1);
    cout << "Reversed list: ";
    printList(reversed);

    //Task 2
    Node* num1 = new Node(5);
    num1->next = new Node(6);
    num1->next->next = new Node(3);

    Node* num2 = new Node(5);
    num2->next = new Node(6);
    num2->next->next = new Node(3);

    cout << "Comparing two lists: ";
    cout << (compare(num1, num2) ? "True" : "False") << endl;

    // Task 3
    Node* bigNum1 = new Node(5);
    bigNum1->next = new Node(3);
    bigNum1->next->next = new Node(9);

    Node* bigNum2 = new Node(1);

    Node* sumList = add(bigNum1, bigNum2);

    cout << "Sum of two large numbers: ";
    printList(sumList);

    return 0;
}
```


Start
int data
Node* next ....
Node* prev = nullptr;
Node* current = head;
Node* next = nullptr;
while (current)
False
True
return prev;
next = current->next;
current->next = prev;
prev = current;
current = next;
bool compare(Node* h1, Node* h2)
while (h1 && h2) {
False
True
return h1 == nullptr && h2 == nullptr;
}
if (h1->data != h2->data)
False
True
return false;
h1 = h1->next;
h2 = h2->next;



Завдання №5 Self Practice Work

Гра в хованки

Обмеження: 2 сек., 256 МіБ

Одного разу, щоб пограти в хованки, зібралося n друзів. Марічка хоче розділити їх всіх на дві команди, враховуючи те, що є m пар таких друзів, що хочуть бути обов'язково в різних командах. Вам потрібно допомогти Марічці і визначити, як саме їй розділити друзів на дві команди.

Вхідні дані

У першому рядку задано два натуральних числа n та m — кількість друзів та кількість пар таких друзів, що хочуть бути обов'язково в різних командах, відповідно.

У наступних m рядках задано по два натуральних числа a_i та b_i. Друзі з номерами a_i та b_i повинні бути в різних командах.

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

const int MAXN = 100005;

vector<int> graph[MAXN];
int team[MAXN];

bool bfs(int n)
{
    for (int i = 1; i <= n; ++i)
    {
        if (team[i] == 0)
        {
            team[i] = 1;
            queue<int> q;
            q.push(i);

            while (!q.empty())
            {
                int curr = q.front();
                q.pop();

                for (int neighbor : graph[curr])
                {
                    if (team[neighbor] == 0) {
                        team[neighbor] = 3 - team[curr];
                        q.push(neighbor);
                    }
                    else if (team[neighbor] == team[curr])
                    {
                        return false;
                    }
                }
            }
        }
    }
    return true;
```

```
    }
    return true;
}

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

    for (int i = 0; i < m; ++i) {
        int a, b;
        cin >> a >> b;
        graph[a].push_back(b);
        graph[b].push_back(a);
    }

    if (!bfs(n)) {
        cout << "Impossible" << endl;
    }
    else {
        for (int i = 1; i <= n; ++i) {
            cout << team[i];
        }
        cout << endl;
    }

    return 0;
}
```

Діаграма:

Зустрічі з командою

Висновок:

Завдяки цій роботі я зрозумів принципи динамічних структур даних, їх ключові операції та відмінності між статичним і динамічним виділенням пам’яті. Практичні вправи зі зв’язним списком і деревом допомогли мені опанувати алгоритми обробки даних у пам'яті та ефективне управління ресурсами. Цей досвід дозволив мені створювати адаптивні рішення для роботи з великими та змінними обсягами даних, підвищуючи продуктивність програм.