# BÁO CÁO

**Bùi Huỳnh Tây**
**MSSV: 24521589**

02/04/2025

—

Cấu trúc dữ liệu và giải thuật -
IT003.P21.CTTN

—

GV Lý Thuyết: Duy Lê

GV Lý Thuyết: Nguyễn Thanh Sơn

GV Lý Thuyết: Phan Thế Duy

GV Lý Thuyết: Văn Thái Hùng

GV HDTH: Phan Minh Quân

GV HDTH: Trần Đình Khang

# NỘI DUNG BÁO CÁO

**Bài tập 4, slide 105:**

**a/ Linked List có và không có con trỏ Tail**

*Linked List (LL) có Tail Pointer:* Tail Pointer là một con trỏ bổ sung, luôn trỏ đến node cuối cùng của danh sách. Với mục đích là tối ưu hóa thao tác thêm phần tử vào cuối danh sách.

- Vì vậy LL có Tail Pointer sẽ có ưu điểm hơn khi cần thêm phần tử liên tục vào cuối, thao tác chèn phần tử vào cuối danh sách (append) có độ phức tạp **O(1)** do truy cập trực tiếp qua Tail.
- Nhưng khi thêm Tail Pointer với những bài toán không cần xử lí ở phần tử cuối nhiều thì sẽ khiến cho Tail Pointer bị thừa và tốn thêm bộ nhớ để lưu trữ.

*Linked List không có Tail Pointer:* chỉ sử dụng con trỏ head để quản lý danh sách với mục đích là tiết kiệm bộ nhớ và đơn giản hóa cấu trúc.

- Vậy nên LL không có Tail Pointer giúp tiết kiệm bộ nhớ vì không cần lưu trữ con trỏ Tail, phù hợp với hệ thống yêu cầu tối ưu hóa bộ nhớ tối đa. Bên cạnh đó cũng đơn giản hơn trong việc triển khai.
- Nhưng vì không có Tail Pointer thì các thao tác chèn/xóa cuối chậm và độ phức tạp O(n) do phải duyệt từ đầu đến cuối. Ví dụ, trong danh sách có 1000 phần tử, việc thêm phần tử thứ 1001 yêu cầu 1000 bước duyệt.

**b/ Linked List có và không có Header Node**

Linked List có Header Node

- Khi chèn/xóa ở đầu, không cần xử lý trường hợp đặc biệt vì Header Node luôn tồn tại
- Mọi thao tác đều thông qua Header Node, giảm thiểu các điều kiện kiểm tra head == NULL
- Code đơn giản hơn, đồng nhất code hơn

Linked List không có Header Node

- Mặc dù là dùng ít hơn 1 biến, nhưng mà khi không có Header Node thì code sẽ phức tạp hơn do phải xử lí trường hợp đặc biệt, kiểm tra head == nullptr.

Vì vậy ta chọn Linked List có Header Node nếu muốn ưu tiên code đơn giản, dễ bảo trì, thao tác ở đầu danh sách diễn ra thường xuyên. Và chọn sử dụng Linked List không có Header Node nếu tối ưu bộ nhớ là yêu cầu hàng đầu và ít thao tác ở đầu danh sách và chấp nhận code phức tạp hơn.

**Bài tập 5, slide 106:**

Sử dụng `struct Student` cho các câu dưới

```
struct Student {
    int id;
    std::string name;
};
```

**a. Đơn không có con trỏ cuối**

```
struct NodeS {
    Student data;
    NodeS* next;
};

NodeS* arrayToSinglyLinkedList(const std::vector<Student>& arr) {
    NodeS* head = nullptr;
    NodeS* current = nullptr;
    for (const auto& s : arr) {
        NodeS* newNode = new NodeS{s, nullptr};
        if (!head) head = newNode;
        else current->next = newNode;
        current = newNode;
    }
    return head;
}

std::vector<Student> singlyLinkedListToArray(NodeS* head) {
    std::vector<Student> arr;
    while (head) {
        arr.push_back(head->data);
        head = head->next;
    }
    return arr;
}
```

**b. Đơn có con trỏ cuối**

```
struct NodeST {
    Student data;
    NodeST* next;
};

struct LinkedListST {
```

```
    NodeST* head;
    NodeST* tail;
};

LinkedListST arrayToSinglyLinkedListWithTail(const
std::vector<Student>& arr) {
    LinkedListST list{nullptr, nullptr};
    for (const auto& s : arr) {
        NodeST* newNode = new NodeST{s, nullptr};
        if (!list.head) list.head = newNode;
        else list.tail->next = newNode;
        list.tail = newNode;
    }
    return list;
}

std::vector<Student> singlyLinkedListWithTailToArray(LinkedListST
list) {
    std::vector<Student> arr;
    while (list.head) {
        arr.push_back(list.head->data);
        list.head = list.head->next;
    }
    return arr;
}
```

**c. Kép**

```
struct NodeD {
    Student data;
    NodeD* prev;
    NodeD* next;
};

NodeD* arrayToDoublyLinkedList(const std::vector<Student>& arr) {
    NodeD* head = nullptr;
    NodeD* tail = nullptr;
    for (const auto& s : arr) {
        NodeD* newNode = new NodeD{s, nullptr, nullptr};
        if (!head) head = newNode;
        else {
            tail->next = newNode;
            newNode->prev = tail;
        }
        tail = newNode;
    }
    return head;
```

```
}

std::vector<Student> doublyLinkedListToArray(NodeD* head) {
    std::vector<Student> arr;
    while (head) {
        arr.push_back(head->data);
        head = head->next;
    }
    return arr;
}
```

**d. Vòng**

```
struct NodeC {
    Student data;
    NodeC* next;
};

NodeC* arrayToCircularLinkedList(const std::vector<Student>& arr)
{
    NodeC* head = nullptr;
    NodeC* tail = nullptr;
    for (const auto& s : arr) {
        NodeC* newNode = new NodeC{s, nullptr};
        if (!head) head = newNode;
        else tail->next = newNode;
        tail = newNode;
    }
    if (tail) tail->next = head;
    return head;
}

std::vector<Student> circularLinkedListToArray(NodeC* head) {
    std::vector<Student> arr;
    if (!head) return arr;
    NodeC* current = head;
    do {
        arr.push_back(current->data);
        current = current->next;
    } while (current != head);
    return arr;
}
```

**e. Kép vòng**

```
struct NodeDC {
    Student data;
```

```
    NodeDC* prev;
    NodeDC* next;
};

NodeDC* arrayToDoublyCircularLinkedList(const
std::vector<Student>& arr) {
    NodeDC* head = nullptr;
    NodeDC* tail = nullptr;
    for (const auto& s : arr) {
        NodeDC* newNode = new NodeDC{s, nullptr, nullptr};
        if (!head) head = newNode;
        else {
            tail->next = newNode;
            newNode->prev = tail;
        }
        tail = newNode;
    }
    if (head) {
        head->prev = tail;
        tail->next = head;
    }
    return head;
}

std::vector<Student> doublyCircularLinkedListToArray(NodeDC*
head) {
    std::vector<Student> arr;
    if (!head) return arr;
    NodeDC* current = head;
    do {
        arr.push_back(current->data);
        current = current->next;
    } while (current != head);
    return arr;
}
```

**Bài tập 6, slide 106:**

a/

```
void insertToSortedArray(int*& arr, int& size, int value) {
    int* newArr = new int[size + 1];
    int pos = 0;
    while (pos < size && arr[pos] < value) pos++;
    for (int i = 0; i < pos; i++) newArr[i] = arr[i];
    newArr[pos] = value;
    for (int i = pos; i < size; i++) newArr[i + 1] = arr[i];
    delete[] arr;
    arr = newArr;
    size++;
}
```

b/

```
struct Node {
    int data;
    Node* next;
};

void insertWithTail(Node*& head, Node*& tail, int value) {
    Node* newNode = new Node{value, nullptr};
    if (!head || head->data >= value) {
        newNode->next = head;
        head = newNode;
        if (!tail) tail = head;
    } else {
        Node* current = head;
        while (current->next && current->next->data < value) {
            current = current->next;
        }
        newNode->next = current->next;
        current->next = newNode;
        if (!newNode->next) tail = newNode;
    }
}


void insertWithoutTail(Node*& head, int value) {
    Node* newNode = new Node{value, nullptr};
    if (!head || head->data >= value) {
        newNode->next = head;
```

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

c/

```
    struct HeaderNode {
        Node* head;
        Node* tail;
    };

    void insertWithHeaderAndTail(HeaderNode& header, int value) {
        Node* newNode = new Node{value, nullptr};
        if (!header.head || header.head->data >= value) {
            newNode->next = header.head;
            header.head = newNode;
            if (!header.tail) header.tail = newNode;
        } else {
            Node* current = header.head;
            while (current->next && current->next->data < value) {
                current = current->next;
            }
            newNode->next = current->next;
            current->next = newNode;
            if (!newNode->next) header.tail = newNode;
        }
    }

    void insertWithHeaderWithoutTail(HeaderNode& header, int value) {
        Node* newNode = new Node{value, nullptr};
        if (!header.head || header.head->data >= value) {
            newNode->next = header.head;
            header.head = newNode;
        } else {
            Node* current = header.head;
            while (current->next && current->next->data < value) {
                current = current->next;
            }
            newNode->next = current->next;
            current->next = newNode;
```

```
        }
    }
```

d/

```
    struct AdvancedHeader {
        Node* head;
        Node* tail;  // Lưu trong các byte đầu của Header
    };

    void insertAdvancedHeader(AdvancedHeader& header, int value) {
        Node* newNode = new Node{value, nullptr};
        if (!header.head || header.head->data >= value) {
            newNode->next = header.head;
            header.head = newNode;
            if (!header.tail) header.tail = newNode;
        } else {
            Node* current = header.head;
            while (current->next && current->next->data < value) {
                current = current->next;
            }
            newNode->next = current->next;
            current->next = newNode;
            if (!newNode->next) header.tail = newNode;
        }
    }
```

Trên đây là phần báo cáo về Linked List về bài tập 4, 5, 6.

---

Người viết báo cáo: BÙI HUỲNH TÂY
MSSV: 24521589
Ngày hoàn thành: 3/4/2025