VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Cấu Trúc Dữ Liệu và Giải Thuật (DSA)

DSA1 - HK242

QUẢN LÝ KHO HÀNG SỬ DỤNG DANH SÁCH

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 1/2025

Mục lục

1 Kiến thức cơ bản về Doubly Linked List

Lớp `DLinkedList` đại diện cho một danh sách liên kết đôi, nơi mỗi nút có hai con trỏ: một trỏ đến nút kế tiếp và một trỏ đến nút trước đó. Lớp này cung cấp các phương thức để thao tác với danh sách liên kết đôi, bao gồm thêm, xóa và truy cập các phần tử.

Các Thành Phần Chính

- **head:** Con trỏ đến nút đầu tiên trong danh sách. Nút này không chứa dữ liệu người dùng mà chỉ là điểm bắt đầu của danh sách.
- **tail:** Con trỏ đến nút cuối cùng trong danh sách. Nút này không chứa dữ liệu người dùng mà chỉ là điểm kết thúc của danh sách.
- **count:** Biến lưu trữ số lượng phần tử hiện tại trong danh sách.

Các Phương Thức Cơ Bản

- **Constructor:** Khởi tạo danh sách liên kết đôi với các con trỏ `head` và `tail` được đặt thành `nullptr`, và `count` thành 0.
- **Destructor:** Giải phóng bộ nhớ của các nút trong danh sách để tránh rò rỉ bộ nhớ.
- **Copy Constructor và Assignment Operator:** Sao chép một đối tượng `DLinkedList` sang một đối tượng khác.
- **Thêm và Xóa Phần Tử:** Thêm hoặc xóa các phần tử khỏi danh sách, điều chỉnh `count` và các con trỏ `head` và `tail` nếu cần.
- **Truy Xuất Phần Tử:** Truy cập phần tử tại một vị trí cụ thể trong danh sách.

Ví Dụ Minh Họa

```cpp
template <typename T>
class DLinkedList {
public:
    class Node {
    public:
        T data;
        Node* next;
        Node* prev;
    };
protected:
    Node<T>* head;  // Con trỏ đến nút đầu tiên
    Node<T>* tail;  // Con trỏ đến nút cuối cùng
    int count;      // Số lượng phần tử hiện tại

public:
    // Constructor
    DLinkedList() : head(nullptr), tail(nullptr), count(0) {}

    // Destructor
    ~DLinkedList() {
        Node<T>* current = head;
        while (current != nullptr) {
            Node<T>* next = current->next;
            delete current;
            current = next;
```

```
        }
    }

    // Copy Constructor
    DLinkedList(const DLinkedList& other) : head(nullptr), tail(nullptr), count(0) {
        Node<T>* current = other.head;
        while (current != nullptr) {
            add(current->data);  // Giả sử có phương thức add()
            current = current->next;
        }
    }

    // Assignment Operator
    DLinkedList& operator=(const DLinkedList& other) {
        if (this != &other) {
            // Xóa tài nguyên cũ
            while (head != nullptr) {
                Node<T>* temp = head;
                head = head->next;
                delete temp;
            }
            count = 0;

            // Sao chép tài nguyên mới
            Node<T>* current = other.head;
            while (current != nullptr) {
                add(current->data);  // Giả sử có phương thức add()
                current = current->next;
            }
        }
        return *this;
    }

    // Thêm phần tử vào danh sách
    void add(const T& item) {
        Node<T>* newNode = new Node<T>{item, nullptr, tail};
        if (tail != nullptr) {
            tail->next = newNode;
        }
        tail = newNode;
        if (head == nullptr) {
            head = newNode;
        }
        ++count;
    }

    // Xóa phần tử tại vị trí cụ thể
    void remove(int index) {
        if (index < 0 || index >= count) {
            throw std::out_of_range("Index out of bounds");
        }

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

```
82
83          if (current->prev != nullptr) {
84              current->prev->next = current->next;
85          }
86          if (current->next != nullptr) {
87              current->next->prev = current->prev;
88          }
89          if (current == head) {
90              head = current->next;
91          }
92          if (current == tail) {
93              tail = current->prev;
94          }
95
96          delete current;
97          --count;
98      }
99
100     // Truy xuất phần tử tại vị trí cụ thể
101     T get(int index) const {
102         if (index < 0 || index >= count) {
103             throw std::out_of_range("Index out of bounds");
104         }
105         Node<T>* current = head;
106         for (int i = 0; i < index; ++i) {
107             current = current->next;
108         }
109         return current->data;
110     }
111
112     // Số lượng phần tử hiện tại
113     int size() const {
114         return count;
115     }
116 };
```

Giải Thích

- **add:** Thêm phần tử vào cuối danh sách và cập nhật con trỏ `tail`.
- **remove:** Xóa phần tử tại một vị trí cụ thể và cập nhật các con trỏ liên kết.
- **Copy Constructor và Assignment Operator:** Đảm bảo sao chép đúng cách danh sách liên kết đôi, bao gồm sao chép các nút và điều chỉnh liên kết.
- **Exception Handling:** Xử lý lỗi như chỉ số ngoài phạm vi để đảm bảo tính ổn định của chương trình.

2 Thiết kế một danh sách liên kết đôi (Doubly Linked List) với hai node giả (dummy nodes)

Node giả (hay còn gọi là **sentinel nodes**) là các node đặc biệt không chứa dữ liệu thật của danh sách, mà chỉ đóng vai trò làm **điểm đánh dấu** ở đầu (head) và cuối (tail) của danh sách. Trong thiết kế này:

- Có **hai node giả**: Một node giả ở đầu (gọi là "head dummy") và một node giả ở cuối (gọi là "tail dummy").
- Chúng luôn tồn tại, ngay cả khi danh sách rỗng.

Vai trò của node giả:

- **Đơn giản hóa thao tác**: Loại bỏ các trường hợp đặc biệt khi danh sách rỗng hoặc khi thêm/xóa ở đầu/cuối.
- **Đánh dấu biên**: Giúp xác định rõ ranh giới của danh sách mà không cần kiểm tra xem danh sách có phần tử nào không.

2.1 Cấu trúc danh sách với hai node giả

Hãy tưởng tượng danh sách như một chuỗi các node được nối với nhau. Với hai node giả, cấu trúc ban đầu (khi danh sách rỗng) trông như sau:

- [**Head Dummy**] <-> [**Tail Dummy**]
  - Head Dummy có con trỏ next trỏ tới Tail Dummy.
  - Tail Dummy có con trỏ prev trỏ tới Head Dummy.
  - Không có dữ liệu thật nào giữa chúng.

Khi thêm phần tử, các node thật sẽ được chèn **giữa** Head Dummy và Tail Dummy. Ví dụ, nếu thêm 3 giá trị (1, 2, 3), danh sách sẽ thành:

- [**Head Dummy**] <-> [**1**] <-> [**2**] <-> [**3**] <-> [**Tail Dummy**]

```
DoublyLinkedList() {
    head = new Node();  // Tạo node giả đầu
    tail = new Node();  // Tạo node giả cuối
    head->next = tail;  // Liên kết head với tail
    tail->prev = head;  // Liên kết tail với head
}

~DoublyLinkedList() {
    // Xóa tất cả các node thật
    Node* current = head->next;
    while (current != tail) {
        Node* temp = current;
        current = current->next;
        delete temp;
    }
    // Xóa node giả
    delete head;
    delete tail;
}
```

Đặc điểm:

- Head Dummy luôn là node đầu tiên, Tail Dummy luôn là node cuối cùng.

- Các node thật (chứa dữ liệu) nằm giữa hai node giả.
- Con trỏ next của Head Dummy trỏ tới node thật đầu tiên (nếu có), hoặc Tail Dummy (nếu danh sách rỗng).
- Con trỏ prev của Tail Dummy trỏ tới node thật cuối cùng (nếu có), hoặc Head Dummy (nếu danh sách rỗng).

2.2 Các thao tác cơ bản với node giả

Tôi sẽ giải thích cách các thao tác hoạt động mà không cần xử lý trường hợp đặc biệt nhờ node giả.

a. Thêm phần tử vào cuối (Append)

- Để thêm một phần tử mới (giả sử giá trị là X) vào cuối danh sách:
  1. Tạo một node mới chứa X.
  2. Chèn node này **trước Tail Dummy**.
  3. Điều chỉnh con trỏ:
     - Gắn prev của node mới vào node thật cuối cùng (hoặc Head Dummy nếu danh sách rỗng).
     - Gắn next của node mới vào Tail Dummy.
     - Cập nhật con trỏ của các node lân cận để liên kết với node mới.

Ví dụ:

- Ban đầu: [**Head Dummy**] <-> [**Tail Dummy**]
- Thêm 1: [**Head Dummy**] <-> [**1**] <-> [**Tail Dummy**]
- Thêm 2: [**Head Dummy**] <-> [**1**] <-> [**2**] <-> [**Tail Dummy**]

b. Thêm phần tử vào đầu (Prepend)

- Để thêm một phần tử mới (giả sử giá trị là X) vào đầu danh sách:
  1. Tạo một node mới chứa X.
  2. Chèn node này **sau Head Dummy**.
  3. Điều chỉnh con trỏ:
     - Gắn next của node mới vào node thật đầu tiên (hoặc Tail Dummy nếu danh sách rỗng).
     - Gắn prev của node mới vào Head Dummy.
     - Cập nhật con trỏ của các node lân cận.

Ví dụ:

- Ban đầu: [**Head Dummy**] <-> [**Tail Dummy**]
- Thêm 0: [**Head Dummy**] <-> [**0**] <-> [**Tail Dummy**]

c. Xóa phần tử

- Để xóa một node cụ thể (giả sử là node N):
  1. Kết nối node trước N (N->prev) với node sau N (N->next).
  2. Cụ thể:
     - Gắn next của N->prev vào N->next.
     - Gắn prev của N->next vào N->prev.
  3. Giải phóng node N.

Ví dụ:

- Ban đầu: [**Head Dummy**] <-> [**1**] <-> [**2**] <-> [**Tail Dummy**]
- Xóa 1: [**Head Dummy**] <-> [**2**] <-> [**Tail Dummy**]
- Node giả đảm bảo không cần kiểm tra xem N có phải là đầu/cuối không.

d. Duyệt danh sách

Để hiển thị hoặc duyệt qua danh sách:

- Bắt đầu từ Head Dummy->next (node thật đầu tiên).
- Tiếp tục đi theo con trỏ next cho đến khi gặp Tail Dummy thì dừng.