TRƯỜNG ĐẠI HỌC GIAO THÔNG VÂN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

---------------o0o---------------



**BÀI TẬP LỚN MÔN HỌC**

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

Giảng viên hướng dẫn: Hoàng Văn Thông

Sinh viên thực hiện: Lê Xuân Quỳnh - Lớp CNTT 5-K61

**Hà Nội, tháng 12 năm 2021**

Mục lục

[I. Đề bài 3](#_Toc89186318)

[II. Phân tích bài toán 3](#_Toc89186319)

[1. Xác định các lớp, các thuộc tính, các phương thức của lớp và mô tả chức năng của từng lớp, từng phương thức. 3](#_Toc89186320)

[a. Lớp Node (Doubly Linked Node) 3](#_Toc89186321)

[b. Lớp dlist\_iterator (Doubly Linked List Iterator) 4](#_Toc89186322)

[c. Lớp List (Doubly Linked List) 5](#_Toc89186323)

[d. Lớp KhachHang 6](#_Toc89186324)

[2. Vẽ sơ đồ khối của những phương thức có chứa các cấu trúc if, cấu trúc lặp 7](#_Toc89186325)

[a. Hàm tạo có đối List(\_nSize, value) 7](#_Toc89186326)

[b. push\_front(element) 8](#_Toc89186327)

[c. push\_back(element) 9](#_Toc89186328)

[d. pop\_front() 10](#_Toc89186329)

[e. pop\_back() 11](#_Toc89186330)

[f. erase(it) 12](#_Toc89186331)

[3. Đánh giá thời gian chạy của phương thức. 16](#_Toc89186332)

[a. Hàm tạo có đối List(\_nSize, value) 16](#_Toc89186333)

[b. push\_front(element) 17](#_Toc89186334)

[c. push\_back(element) 18](#_Toc89186335)

[d. pop\_front() 19](#_Toc89186336)

[e. pop\_back() 20](#_Toc89186337)

[f. erase(it) 21](#_Toc89186338)

[III. Cài đặt các lớp và hàm main bằng C++ 25](#_Toc89186339)

[IV. Danh sách tài liệu tham khảo 25](#_Toc89186340)

# Đề bài

Ứng dụng lớp List để quản lý các đối tượng là các Khách hàng. Mỗi Khách hàng cần quản lý các thông tin sau: Mã KH là 1 số nguyên, Họ tên, SĐT.

1. Bổ sung vào lớp List phương thức sắp xếp theo thuật toán QuickSort, để sắp xếp các phần tử theo thứ tự một hàm so sánh được vào cho hàm Quicksort

2. Viết chương trình cho phép thực hiện các chức năng sau:

* Nhập thêm các khách hàng, sau khi hoàn thành nhập thông tin của một khách hàng, chương trình đưa ra câu hỏi có nhập nữa không (y/n)? Nếu người dùng nhập: Y thì tiếp tục nhập, nhập N thì kết thúc.
* Xóa đi một khách hàng
* Sửa đổi thông tin họ tên, SĐT của một khách hàng bất kỳ trong danh sách
* Sắp xếp danh sách khách hàng theo Mã KH
* Sắp xếp danh sách khách hàng theo họ tên.
* Hiển thị toàn bộ danh sách khách hàng hiện có.

# Phân tích bài toán

## Xác định các lớp, các thuộc tính, các phương thức của lớp và mô tả chức năng của từng lớp, từng phương thức.

### Lớp Node (Doubly Linked Node)

Doubly Linked Node là một Node trong danh sách các Node liên kết với nhau tạo thành Doubly Linked List.

* *Các thuộc tính:*

+, element: Giá trị kiểu mẫu, dùng để lưu trữ dữ liệu của Node

+, pNext: Con trỏ kiểu Node\* , dùng để trỏ đến Node đằng sau nó trong danh sách

+, pPre: Con trỏ kiểu Node\* , dùng để trỏ đến Node phía trước nó trong danh sách

* *Các phương thức:*

+, Hàm tạo không đối Node(): Dùng để khởi tạo Node mới không trỏ đến Node nào cả, có pNext = pPre = nullptr (con trỏ null)

+, getElement(): Dùng để lấy giá trị của Node hiện tại

+, getNext(): Dùng để lấy địa chỉ mà con trỏ pNext đang trỏ tới

+, getPre(): Dùng để lấy địa chỉ mà con trỏ pPre đang trỏ tới

+, setElement(\_element): Dùng để gán giá trị cho Node bằng cách gán giá trị thuộc tính element = \_element

+, setNext(\_pNext): Dùng để gán giá trị cho con trỏ pNext bằng cách gán giá trị thuộc tính pNext = \_pNext

+, setPre(\_pPre): Dùng để gán giá trị cho con trỏ pPre bằng cách gán giá trị thuộc tính pPre = \_pPre

### Lớp dlist\_iterator (Doubly Linked List Iterator)

Doubly Linked List Iterator đóng vai trò là bộ lặp có tác dụng lặp qua từng phần tử trong List

* *Các thuộc tính:*

+, pCurrent: Con trỏ kiểu Node\* , dùng để lưu trữ địa chỉ của Node đang được iterator trỏ tới

* *Các phương thức:*

+, Hàm tạo không đối dlist\_iterator(): Dùng để khởi tạo iterator mới có pCurrent trỏ đến nullptr

+, Hàm tạo có đối dlist\_iterator(\_pCurrent): Dùng để khởi tạo iterator mới có pCurrent trỏ đến một giá trị địa chỉ là \_pCurrent được truyền vào

+, getCurrent(): Dùng để lấy giá trị địa chỉ của Node đang được iterator trỏ tới (trả về pCurrent)

+, operator=(it): Nạp chồng toán tử gán, dùng để gán giá trị của iterator này cho iterator khác

+, operator==(it): Nạp chồng toán tử so sánh ==, kiểm tra xem hai iterator có bằng nhau hay không

+, operator!=(it): Nạp chồng toán tử so sánh !=, kiểm tra xem hai iterator có khác nhau hay không

+, operator++(): Nạp chồng toán tử ++ (dạng tiền tố), dùng để trỏ iterator đến Node phía sau Node hiện tại

+, operator++(int): Nạp chồng toán tử ++ (dạng hậu tố), dùng để trỏ iterator đến Node phía sau Node hiện tại

+, operator--(): Nạp chồng toán tử -- (dạng tiền tố), dùng để trỏ iterator đến Node phía trước Node hiện tại

+, operator--(int): Nạp chồng toán tử -- (dạng hậu tố), dùng để trỏ iterator đến Node phía trước Node hiện tại

+, operator\*(): Nạp chồng toán tử một ngôi \*, dùng để lấy giá trị của Node đang được iterator trỏ tới

### Lớp List (Doubly Linked List)

Doubly Linked List là một cấu trúc dữ liệu tuyến tính bao gồm một nhóm các Node tạo thành một danh sách. Dùng để lưu trữ danh sách các đối tượng (Ví dụ: Danh sách khách hàng, Danh sách sinh viên,… )

* *Các thuộc tính:*

+, pHead: Con trỏ kiểu Node\* , dùng để trỏ đến Node đầu tiên của danh sách

+, pTrail: Con trỏ kiểu Node\* , dùng để trỏ đến Node cuối cùng của danh sách

+, nSize: giá trị kiểu int, dùng để lưu trữ số lượng phần tử có trong List

* *Các phương thức:*

+, Hàm tạo không đối List(): Dùng để khởi tạo List rỗng có pHead = pTrail = nullptr (con trỏ null) và nSize = 0.

+, Hàm tạo có đối List(\_nSize, value): Dùng để khởi tạo List có \_nSize phần tử, mỗi phần tử trong List đều có cùng giá trị value

+, size(): Trả về kích thước hiện tại của List

+, empty(): Kiểm tra xem List có rỗng hay không

+, front(): Trả về giá trị của Node đầu tiên

+, back(): Trả về giá trị của Node cuối cùng

+, push\_front(element): Thêm một Node mới có giá trị element vào đầu danh sách

+, push\_back(element): Thêm một Node mới có giá trị element vào cuối danh sách

+, pop\_front(): Xoá bỏ Node đầu tiên trong danh sách

+, pop\_back(): Xoá bỏ Node cuối cùng trong danh sách

+, begin(): Trả về iterator trỏ tới Node đầu tiên trong danh sách

+, end(): Trả về iterator trỏ tới pNext của Node cuối cùng trong danh sách (trỏ tới nullptr)

+, erase(it): xoá bỏ một nút trong danh sách đang được iterator it trỏ tới

**+, Thuật toán Quick Sort:**

* isEqual(a, b, cmp): Sử dụng để biến đổi toán tử so sánh a và b (a > b, a < b, ...) trong hàm cmp thành toán tử so sánh a == b
* isDifferent(a, b, cmp): Sử dụng để biến đổi toán tử so sánh a và b (a > b, a < b, ...) trong hàm cmp thành toán tử so sánh a != b
* partition(i, j, cmp): Hàm phân hoạch List thành 3 dãy con S1, S2, S3. Trong đó: S2 chỉ có một phần tử và S2 = k là vị trí phân chia sao cho S2 nằm giữa S1 và S3 (S1 và S3 có thể rỗng). Sau khi phân hoạch xong ta được tất cả các phần tử của dãy S1 S2 < S3 (dãy tăng) hoặc S1 S2 > S3 (dãy giảm), điều kiện để dãy tăng hay dãy giảm phụ thuộc vào hàm so sánh cmp ta truyền vào.
* \_\_quick\_\_sort\_\_(i, j, cmp): Dùng để gọi đệ quy phân hoạch tiếp dãy S1 thành 3 dãy con và S2 thành 3 dãy con tương tự như dãy ban đầu cho đến khi dãy chỉ còn 1 phần tử thì dừng đệ quy. Sau khi quá trình đệ quy kết thúc ta được dãy ban đầu đã được sắp xếp tăng hoặc giảm (phụ thuộc vào hàm so sánh cmp được truyền vào)
* quick\_sort(cmp): Dùng để đơn giản hoá cách gọi hàm sắp xếp bằng thuật toán Quick Sort, người sử dụng chỉ cần truyền vào hàm so sánh cmp để sắp xếp dãy mà không cần quan tâm đến điểm đầu và điểm cuối của dãy cần sắp.

### Lớp KhachHang

Dùng để lưu trữ thông tin của đối tượng là Khách Hàng

* *Các thuộc tính:*

+, maKH: Dùng để lưu mã khách hàng của một khách hàng, giá trị là một số nguyên (int)

+, ho\_ten: Dùng để lưu trữ họ tên của một khách hàng, giá trị là một chuỗi kí tự (string)

+, sdt: Dùng để lưu trữ số điện thoại của một khách hàng, giá trị là một số nguyên (int)

* *Các phương thức:*

+, getMaKH(): Dùng để lấy mã khách hàng của một khách hàng

+, getHoTen(): Dùng để lấy họ tên của một khách hàng

+, nhap(): Dùng để nhập thông tin của một khách hàng từ bàn phím

+, xuat(): Dùng để xuất thông tin của một khách hàng ra màn hình

## Vẽ sơ đồ khối của những phương thức có chứa các cấu trúc if, cấu trúc lặp

* **Lớp List:**

### Hàm tạo có đối List(\_nSize, value)

Diagram

Description automatically generated

### push\_front(element)

Diagram

Description automatically generated

### push\_back(element)

Diagram

Description automatically generated

### pop\_front()

Diagram

Description automatically generated

### pop\_back()

Diagram

Description automatically generated

### erase(it)

Diagram

Description automatically generated

1. **Thuật toán Quick Sort:**

* **isEqual(a, b, cmp)**

Diagram

Description automatically generated

* **isDifferent(a, b, cmp)**

Diagram

Description automatically generated

* **\_\_quick\_\_sort\_\_(i, j, cmp)**

Diagram

Description automatically generated

* **partition(i, j, cmp)**

**Chart, funnel chart

Description automatically generated**

* **quick\_sort(cmp)**

Diagram

Description automatically generated

## Đánh giá thời gian chạy của phương thức.

* **Lớp List:**

### Hàm tạo có đối List(\_nSize, value)

**Số phép toán cơ bản**

List(int \_nSize, const T &value = 0)

{

pHead = pTrail = nullptr; 2

nSize = 0; 1

while (\_nSize--) 3n

{

push\_back(value); n

}

}

* **Số phép toán cơ bản nhiều nhất là:**

T(n) = 2 + 1 + 3n + n = 4n + 3

* **Độ phức tạp tiệm cận của thuật toán là:** O(n)

### b. push\_front(element)

**Số phép toán cơ bản**

void push\_front(const T &element)

{

Node<T> \*new\_node = new Node<T>; 2

new\_node->setElement(element); 1

if (empty()) 2

{

pHead = pTrail = new\_node; 0

}

else

{

new\_node->setNext(pHead); 1

pHead->setPre(new\_node); 1

pHead = new\_node; 1

}

nSize++; 2

}

* **Số phép toán cơ bản nhiều nhất là:**

T(n) = 2 + 1 + 2 + 0 + 1 + 1 + 1 + 2 = 10

* **Độ phức tạp tiệm cận của thuật toán là:** O(1)

### c. push\_back(element)

**Số phép toán cơ bản**

void push\_back(const T &element)

{

Node<T> \*new\_node = new Node<T>; 2

new\_node->setElement(element); 1

if (empty()) 2

{

pHead = pTrail = new\_node; 0

}

else

{

pTrail->setNext(new\_node); 1

new\_node->setPre(pTrail); 1

pTrail = new\_node; 1

}

nSize++; 2

}

* **Số phép toán cơ bản nhiều nhất là:**

T(n) = 2 + 1 + 2 + 0 + 1 + 1 + 1 + 2 = 10

* **Độ phức tạp tiệm cận của thuật toán là:** O(1)

### d. pop\_front()

**Số phép toán cơ bản**

void pop\_front()

{

Node<T> \*p = pHead; 1

if (empty()) 2

{

return; 0

}

else if (nSize == 1) 1

{

pHead = pTrail = nullptr; 0

}

else

{

pHead = pHead->getNext(); 2

pHead->setPre(nullptr); 1

}

delete p; 1

nSize--; 2

}

* **Số phép toán cơ bản nhiều nhất là:**

T(n) = 1 + 2 + 0 + 1 + 0 + 2 + 1 + 1 + 2 = 10

* **Độ phức tạp tiệm cận của thuật toán là:** O(1)

### pop\_back()

**Số phép toán cơ bản**

void pop\_back()

{

Node<T> \*p = pTrail; 1

if (empty()) 2

{

return; 0

}

else if (nSize == 1) 1

{

pHead = pTrail = nullptr; 0

}

else

{

pTrail = pTrail->getPre(); 2

pTrail->setNext(nullptr); 1

}

delete p; 1

nSize--; 2

}

* **Số phép toán cơ bản nhiều nhất là:**

T(n) = 1 + 2 + 0 + 1 + 0 + 2 + 1 + 1 + 2 = 10

* **Độ phức tạp tiệm cận của thuật toán là:** O(1)

### erase(it)

**Số phép toán cơ bản**

void erase(iterator &it)

{

if (it == begin()) 2

{

return pop\_front(); 0

}

if (it == end()) 2

{

return pop\_back(); 0

}

it.getCurrent()->getPre()->setNext(it.getCurrent()->getNext()); 5

it.getCurrent()->getNext()->setPre(it.getCurrent()->getPre()); 5

delete it.getCurrent(); 2

nSize--; 2

}

* **Số phép toán cơ bản nhiều nhất là:**

T(n) = 2 + 0 + 2 + 0 + 5 + 5 + 2 + 2 = 18

* **Độ phức tạp tiệm cận của thuật toán là:** O(1)

1. **Thuật toán Quick Sort:**

* **partition(i, j, cmp)**

**Số phép toán cơ bản**

|  |  |
| --- | --- |
| Node<T> \*partition(Node<T> \*i, Node<T> \*j, bool (\*cmp)(T &, T &))  {  Node<T> \*left = i;  Node<T> \*right = j;  T pivot = i->getElement();  while (left != right && left != right->getNext())  {  while ((cmp(left->getElement(), pivot) || isEqual(left->getElement(), pivot, cmp)) && left != right && left != right->getNext())  {  left = left->getNext();  }  while (cmp(pivot, right->getElement()) && isDifferent(pivot, right->getElement(), cmp))  {  right = right->getPre();  }  if (left != right && left != right->getNext())  {  std::swap(left->getElement(), right->getElement());  }  }  if (i != right)  {  std::swap(i->getElement(), right->getElement());  }  return right;  } | 1  1  2  4  10  2n /2 = n  5  2n /2 = n  4  3  1  3  1 |
|  |  |

* **Số phép toán cơ bản nhiều nhất là:**

T(n) = 4 + 4 (10n + 5n + 7) + 5 = 60n + 37

* **Độ phức tạp tiệm cận của thuật toán là:** O(n)
* **\_\_quick\_\_sort\_\_(i, j, cmp)**

**Số phép toán cơ bản**

|  |  |
| --- | --- |
| void \_\_quick\_\_sort\_\_(Node<T> \*i, Node<T> \*j, bool (\*cmp)(T &, T &))  {  if (i != nullptr && j != nullptr)  {  if (i != j && i != j->getNext())  {  Node<T> \*k = partition(i, j, cmp);  \_\_quick\_\_sort\_\_(i, k->getPre(), cmp);  \_\_quick\_\_sort\_\_(k->getNext(), j, cmp);  }  }  } | 3  4  2 (60n+37)  ( |

* **Số phép toán cơ bản nhiều nhất là:**
* Mỗi lần đệ quy sẽ gọi hàm partition một lần, mà có lần gọi đệ quy. Vậy tổng số phép tính là:

T(n) = [7 + 2 (60n + 37)] \* = (120n + 81) \*

* **Độ phức tạp tiệm cận của thuật toán là:** O()
* **Trường hợp xấu nhất:**

Trường hợp dãy đã được sắp xếp dẫn tới tập con bên trái hoặc bên phải luôn luôn rỗng nên số lần gọi đệ quy là n lần

-> Tổng số phép tính là: [7 + 2 (60n + 37)] \* n = 120n2 + 81n

-> Độ phức tạp: O(n2)

# Cài đặt các lớp và hàm main bằng C++

* Các lớp và hàm main được cài đặt bằng C++ được lưu trữ trong thư mục source\_code. Trong đó:

+, Lớp List, Node và dlist\_iterator được lưu trữ trong cùng một thư mục tên là lib

+, Hàm main và lớp KhachHang được cài đặt trong file qly\_khach\_hang.cpp

# Danh sách tài liệu tham khảo

1. Slide bài giảng môn học Cấu trúc dữ liệu và Giải thuật, *Lecture 7 Linked List.ppt*, Trường Đại Học Giao Thông Vận Tải
2. Slide bài giảng môn học Cấu trúc dữ liệu và Giải thuật, *Lectures 12 Sorting - nlogn.ppt*, Trường Đại Học Giao Thông Vận Tải