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: TS. Hoàng Văn Thông

Sinh viên thực hiện: Phạm Duy Phương

Lớp: CNTT3

# Hà Nội tháng 11 năm 2023

BÀI 1 (BÀI SỐ 4 TRONG DANH SÁCH BÀI TẬP)

**1.1.** Đề bài

1. ﻿Xây dựng lớp danh sách liên kết kép mẫu.
2. Ứng dụng lớp danh sách liên kết kép xây dựng lớp để quản lý các đối tượng là các sinh viên có các chức năng:

﻿a. Đọc danh sách sinh viên từ file vào danh sách

b. Ghi danh sách sinh viên vào file

c. Nhập thêm một sinh viên vào cuối danh sách

g. Xóa đi một sinh viên theo mã sinh viên

d. Hiển thị toàn bộ danh sách sinh viên hiện có trong danh sách

e. Cập nhật thông tin của một sinh viên bất kỳ trong danh sách khi biết mã sinh viên và các thông tin cần cập nhật.

f. Tìm kiếm sinh viên theo họ và tên bằng phương pháp tìm kiếm tuần tự

*Mỗi sinh viên cần quản lý các thông tin sau: Mã SV, Họ và tên, Ngày sinh, giới tính, lớp.*

1. Viết hàm main thực hiện các chức năng trên.

**1.2.** Phân tích bài toán

* Yêu cầu bài toán:

Bài toán yêu cầu xây dựng một lớp danh sách liên kết kép để quản lý các đối tượng là sinh viên. Lớp này cần cung cấp các chức năng sau:

a. Đọc danh sách sinh viên từ file vào danh sách

b. Ghi danh sách sinh viên vào file

c. Nhập thêm một sinh viên vào cuối danh sách

g. Xóa đi một sinh viên theo mã sinh viên

d. Hiển thị toàn bộ danh sách sinh viên hiện có trong danh sách

e. Cập nhật thông tin của một sinh viên bất kỳ trong danh sách khi biết mã sinh viên và các thông tin cần cập nhật.

f. Tìm kiếm sinh viên theo họ và tên bằng phương pháp tìm kiếm tuần tự

*Mỗi sinh viên cần quản lý các thông tin sau: Mã SV, Họ và tên, Ngày sinh, giới tính, lớp.*

* Dựa trên yêu cầu bài toán, ta có thể xây dựng các lớp sau:
  + Lớp SinhVien: Đại diện cho một sinh viên và chứa các thông tin của sinh viên như Mã SV, Họ và tên, Ngày sinh, Giới tính, Lớp.
* Lớp SinhVien bao gồm các thuộc tính: MaSv, HoTen, Lop, NgaySinh.
* Lớp Sinhvien bao gồm các phương thức: Khởi tạo, set/get.
  + Lớp Node: Đại diện cho một node trong danh sách liên kết kép, chứa một đối tượng SinhVien và các con trỏ liên kết.
* Lớp Node bao gồm các thuộc tính: data, prev, next
* Lớp Node bao gồm các phương thức: Khởi tạo, set/get.
  + Lớp DanhSachSinhVien: Đại diện cho danh sách liên kết kép, chứa các phương thức để thực hiện các chức năng yêu cầu.
* Lớp DanhSachSinhVien bao gồm các thuộc tính sau: head, tail.
* Lớp DanhSachSinhVien bao gồm các phương thức sau:

1. DanhSachSinhVien(): Constructor của lớp DanhSachSinhVien, khởi tạo con trỏ head và tail thành nullptr.
2. ~DanhSachSinhVien(): Destructor của lớp DanhSachSinhVien, giải phóng bộ nhớ bằng cách xóa từng nút trong danh sách.
3. void docDanhSachSinhVienTuFile(const string& tenFile): Phương thức này đọc danh sách sinh viên từ một file được chỉ định. Nó mở file và đọc dữ liệu từ file, sau đó tạo các đối tượng SinhVien từ dữ liệu và thêm chúng vào danh sách bằng cách gọi phương thức themSinhVien(). Nếu không thể mở file, phương thức sẽ in ra thông báo "Khong mo duoc file!".
4. void ghiDanhSachSinhVienVaoFile(string tenFile): Phương thức này ghi danh sách sinh viên vào một file được chỉ định. Nó mở file và ghi thông tin về mỗi sinh viên trong danh sách vào file. Nếu không thể mở file, phương thức sẽ in ra thông báo "Khong mo duoc file!".
5. void themSinhVien(SinhVien sv): Phương thức này thêm một sinh viên mới vào danh sách. Nó tạo một nút mới với đối tượng SinhVien được truyền vào, sau đó thêm nút đó vào cuối danh sách.
6. void xoaSinhVien(string maSV): Phương thức này xóa một sinh viên khỏi danh sách dựa trên mã sinh viên (maSV). Nó tìm nút chứa sinh viên có mã sinh viên tương ứng, sau đó xóa nút đó khỏi danh sách.
7. void hienThiDanhSachSinhVien(): Phương thức này hiển thị thông tin về tất cả sinh viên trong danh sách. Nó duyệt qua từng nút trong danh sách và in ra thông tin của từng sinh viên.
8. void capNhatSinhVien(string maSV, SinhVien sv): Phương thức này cập nhật thông tin của một sinh viên trong danh sách dựa trên mã sinh viên (maSV). Nó tìm nút chứa sinh viên có mã sinh viên tương ứng, sau đó cập nhật thông tin của sinh viên đó bằng đối tượng SinhVien được truyền vào.
9. void timKiemSinhVien(string hoTen): Phương thức này tìm kiếm sinh viên trong danh sách dựa trên họ tên (hoTen). Nó duyệt qua từng nút trong danh sách và in ra thông tin của sinh viên nếu họ tên trùng khớp. Nếu không tìm thấy sinh viên nào, phương thức sẽ in ra thông báo "Khong tim thay sinh vien!".

﻿**1.3.** Cài đặt các lớp và hàm main bằng C++

* **Cài đặt lớp SinhVien:**

#ifndef **SOURCEOCODE\_SINHVIEN\_H**#define **SOURCEOCODE\_SINHVIEN\_H**#include <iostream>  
#include <fstream>  
#include <string>  
  
using namespace std;  
  
class SinhVien {  
private:  
 string maSV;  
 string hoTen;  
 string ngaySinh;  
 string gioiTinh;  
 string lop;  
public:  
 SinhVien(string maSV, string hoTen, string ngaySinh, string gioiTinh, string lop) {  
 this->maSV = maSV;  
 this->hoTen = hoTen;  
 this->ngaySinh = ngaySinh;  
 this->gioiTinh = gioiTinh;  
 this->lop = lop;  
 }  
 SinhVien() {}  
 string getMaSV() {  
 return maSV;  
 }  
 string getHoTen() {  
 return hoTen;  
 }  
 string getNgaySinh() {  
 return ngaySinh;  
 }  
 string getGioiTinh() {  
 return gioiTinh;  
 }  
 string getLop() {  
 return lop;  
 }  
 void setMaSV(string maSV) {  
 this->maSV = maSV;  
 }  
 void setHoTen(string hoTen) {  
 this->hoTen = hoTen;  
 }  
 void setNgaySinh(string ngaySinh) {  
 this->ngaySinh = ngaySinh;  
 }  
 void setGioiTinh(string gioiTinh) {  
 this->gioiTinh = gioiTinh;  
 }  
 void setLop(string lop) {  
 this->lop = lop;  
 }  
};  
#endif *//SOURCEOCODE\_SINHVIEN\_H*

* **Cài đặt lớp Node:**

#ifndef **SOURCEOCODE\_NODE\_H**#define **SOURCEOCODE\_NODE\_H**#include <iostream>  
#include <fstream>  
#include <string>  
#include "SinhVien.h"  
  
using namespace std;  
  
class Node {  
private:  
 SinhVien data;  
 Node \*prev;  
 Node \*next;  
public:  
 Node(SinhVien data) {  
 this->data = data;  
 prev = nullptr;  
 next = nullptr;  
 }  
 SinhVien getData() {  
 return data;  
 }  
 Node \*getPrev() {  
 return prev;  
 }  
 Node \*getNext() {  
 return next;  
 }  
 void setData(SinhVien data) {  
 this->data = data;  
 }  
 void setPrev(Node \*prev) {  
 this->prev = prev;  
 }  
 void setNext(Node \*next) {  
 this->next = next;  
 }  
};  
#endif *//SOURCEOCODE\_NODE\_H*

* **Cài đặt lớp DanhSachSinhVien:**

#ifndef **SOURCEOCODE\_DANHSACHSINHVIEN\_H**#define **SOURCEOCODE\_DANHSACHSINHVIEN\_H**#include "Node.h"  
#include "SinhVien.h"  
#include <fstream>  
  
class DanhSachSinhVien {  
private:  
 Node\* head;  
 Node\* tail;  
public:  
 DanhSachSinhVien() {  
 head = nullptr;  
 tail = nullptr;  
 }  
 ~DanhSachSinhVien() {  
 Node\* current = head;  
 while (current != nullptr) {  
 Node\* next = current->getNext();  
 delete current;  
 current = next;  
 }  
 }  
 void docDanhSachSinhVienTuFile(const string& tenFile) {  
 ifstream input(tenFile);  
 if (input.is\_open()) {  
 string maSV, hoTen, ngaySinh, gioiTinh, lop;  
 while (!input.eof()) {  
 getline(input, maSV, ',');  
 if(maSV.empty()) break;  
 getline(input, hoTen, ',');  
 getline(input, ngaySinh, ',');  
 getline(input, gioiTinh, ',');  
 getline(input, lop);  
 SinhVien sv(maSV, hoTen, ngaySinh, gioiTinh, lop);  
 themSinhVien(sv);  
 }  
 input.close();  
 }  
 else {  
 cout << "Khong mo duoc file!" << endl;  
 }  
 cout << "Doc file thanh cong!" << endl;  
 }  
 void ghiDanhSachSinhVienVaoFile(string tenFile) {  
 ofstream file(tenFile);  
 if (file.is\_open()) {  
 Node\* current = head;  
 while (current != nullptr) {  
 SinhVien sv = current->getData();  
 file << sv.getMaSV() << ",";  
 file << sv.getHoTen() << ",";  
 file << sv.getNgaySinh() << ",";  
 file << sv.getGioiTinh() << ",";  
 file << sv.getLop() << endl;  
 current = current->getNext();  
 }  
 file.close();  
 }  
 else {  
 cout << "Khong mo duoc file!" << endl;  
 }  
 cout << "Ghi file thanh cong!" << endl;  
 }  
 void themSinhVien(SinhVien sv) {  
 Node\* newNode = new Node(sv);  
 if (head == nullptr) {  
 head = newNode;  
 tail = newNode;  
 }  
 else {  
 newNode->setPrev(tail);  
 tail->setNext(newNode);  
 tail = newNode;  
 }  
 }  
 void xoaSinhVien(string maSV) {  
 Node\* current = head;  
 while (current != nullptr) {  
 if (current->getData().getMaSV() == maSV) {  
 Node\* prevNode = current->getPrev();  
 Node\* nextNode = current->getNext();  
 if (prevNode != nullptr) {  
 prevNode->setNext(nextNode);  
 }  
 else {  
 head = nextNode;  
 }  
 if (nextNode != nullptr) {  
 nextNode->setPrev(prevNode);  
 }  
 else {  
 tail = prevNode;  
 }  
 delete current;  
 break;  
 }  
 current = current->getNext();  
 }  
 }  
 void hienThiDanhSachSinhVien() {  
 Node\* current = head;  
 while (current != nullptr) {  
 SinhVien sv = current->getData();  
 cout << "Ma SV: " << sv.getMaSV() << endl;  
 cout << "Ho ten: " << sv.getHoTen() << endl;  
 cout << "Ngay sinh: " << sv.getNgaySinh() << endl;  
 cout << "Gioi tinh: " << sv.getGioiTinh() << endl;  
 cout << "Lop: " << sv.getLop() << endl;  
 current = current->getNext();  
 }  
 cout << endl;  
 }  
 void capNhatSinhVien(string maSV, SinhVien sv) {  
 Node\* current = head;  
 while (current != nullptr) {  
 if (current->getData().getMaSV() == maSV) {  
 current->setData(sv);  
 break;  
 }  
 current = current->getNext();  
 }  
 }  
 void timKiemSinhVien(string hoTen) {  
 bool check = false;  
 Node\* current = head;  
 while (current != nullptr) {  
 if (current->getData().getHoTen() == hoTen) {  
 check = true;  
 SinhVien sv = current->getData();  
 cout << "Ma SV: " << sv.getMaSV() << endl;  
 cout << "Ho ten: " << sv.getHoTen() << endl;  
 cout << "Ngay sinh: " << sv.getNgaySinh() << endl;  
 cout << "Gioi tinh: " << sv.getGioiTinh() << endl;  
 cout << "Lop: " << sv.getLop() << endl;  
 cout << endl;  
 }  
 current = current->getNext();  
 }  
 if (!check) {  
 cout << "Khong tim thay sinh vien!" << endl;  
 }  
 }  
};  
#endif *//SOURCEOCODE\_DANHSACHSINHVIEN\_H*

* **Cài đặt hàm main:**

#include <iostream>  
#include "SinhVien.h"  
#include "DanhSachSinhVien.h"  
  
using namespace std;  
  
int main() {  
 DanhSachSinhVien danhSach;  
 danhSach.docDanhSachSinhVienTuFile("/Users/Documents/outsource/ctdl/sourceocode/bai4/DanhSachSinhVien.txt");  
  
 cout << "Danh sach sinh vien sau khi doc file: " << endl;  
 danhSach.hienThiDanhSachSinhVien();  
  
 string maSV, hoTen, ngaySinh, gioiTinh, lop;  
 cout << "\nThem sinh vien:" << endl;  
 cout << "Nhap thong tin sinh vien: " << endl;  
 cout << "Ma SV: ";  
 getline(cin, maSV);  
 cout << "Ho ten: ";  
 getline(cin, hoTen);  
 cout << "Ngay sinh: ";  
 getline(cin, ngaySinh);  
 cout << "Gioi tinh: ";  
 getline(cin, gioiTinh);  
 cout << "Lop: ";  
 getline(cin, lop);  
  
 SinhVien sv(maSV, hoTen, ngaySinh, gioiTinh, lop);  
 danhSach.themSinhVien(sv);  
  
 cout << "\nDanh sach sinh vien sau khi them sinh vien: " << endl;  
 danhSach.hienThiDanhSachSinhVien();  
  
 string maSVXoa;  
 cout << "Xoa sinh vien:" << endl;  
 cout << "Nhap ma SV can xoa: ";  
 getline(cin, maSVXoa);  
 danhSach.xoaSinhVien(maSVXoa);  
  
 cout << "Danh sach sinh vien sau khi xoa sinh vien: " << endl;  
 danhSach.hienThiDanhSachSinhVien();  
  
 cout << "Cap nhat thong tin sinh vien: " << endl;  
 string maSVCapNhat;  
 cout << "Nhap ma SV can cap nhat: ";  
 getline(cin, maSVCapNhat);  
 cout << "Nhap thong tin sinh vien moi: " << endl;  
 cout << "Ho ten: ";  
 getline(cin, hoTen);  
 cout << "Ngay sinh: ";  
 getline(cin, ngaySinh);  
 cout << "Gioi tinh: ";  
 getline(cin, gioiTinh);  
 cout << "Lop: ";  
 getline(cin, lop);  
  
 SinhVien svCapNhat(maSVCapNhat, hoTen, ngaySinh, gioiTinh, lop);  
 danhSach.capNhatSinhVien(maSVCapNhat, svCapNhat);  
  
 cout << "Danh sach sinh vien sau khi cap nhat sinh vien: " << endl;  
 danhSach.hienThiDanhSachSinhVien();  
  
 string hoTenTimKiem;  
 cout << "Nhap ho ten sinh vien can tim: ";  
 getline(cin, hoTenTimKiem);  
 danhSach.timKiemSinhVien(hoTenTimKiem);  
  
danhSach.ghiDanhSachSinhVienVaoFile("/Users/Documents/outsource/ctdl/sourceocode/bai4/DanhSachSinhVien.txt");  
  
 return 0;  
}

**1.4.** Phân tích thời gian chạy của từng phương thức có trong các lớp

Các phương thức khởi tạo, set/get trên đều có thời gian chạy là O(1).

Các phương thức khác:

1. DanhSachSinhVien(): Phương thức này chỉ thực hiện việc khởi tạo con trỏ head và tail, do đó có độ phức tạp O(1).
2. ~DanhSachSinhVien(): Phương thức này giải phóng bộ nhớ bằng cách xóa từng nút trong danh sách. Với n sinh viên trong danh sách, việc xóa từng nút có độ phức tạp O(n).
3. void docDanhSachSinhVienTuFile(const string& tenFile): Phương thức này đọc danh sách sinh viên từ file và thêm chúng vào danh sách. Với n sinh viên trong danh sách, việc đọc và thêm từng sinh viên vào danh sách có độ phức tạp O(n).
4. void ghiDanhSachSinhVienVaoFile(string tenFile): Phương thức này ghi danh sách sinh viên vào file. Với n sinh viên trong danh sách, việc ghi thông tin sinh viên vào file có độ phức tạp O(n).
5. void themSinhVien(SinhVien sv): Phương thức này thêm một sinh viên mới vào danh sách bằng cách tạo một nút mới và thêm nút đó vào cuối danh sách. Do công việc chỉ liên quan đến việc thêm một nút mới vào cuối danh sách, nên độ phức tạp của phương thức này là O(1).
6. void xoaSinhVien(string maSV): Phương thức này xóa một sinh viên khỏi danh sách dựa trên mã sinh viên (maSV). Việc tìm nút chứa sinh viên có mã sinh viên tương ứng có thể mất đến O(n) trong trường hợp tồn tại ở cuối danh sách. Sau đó, việc xóa nút đó khỏi danh sách có độ phức tạp O(1). Vì vậy, độ phức tạp tổng thể của phương thức này là O(n).
7. void hienThiDanhSachSinhVien(): Phương thức này duyệt qua từng nút trong danh sách và in ra thông tin của từng sinh viên. Với n sinh viên trong danh sách, việc duyệt qua từng nút và in ra thông tin có độ phức tạp O(n).
8. void capNhatSinhVien(string maSV, SinhVien sv): Phương thức này cập nhật thông tin của một sinh viên trong danh sách dựa trên mã sinh viên (maSV). Việc tìm nút chứa sinh viên có mã sinh viên tương ứng có thể mất đến O(n). Sau đó, việc cập nhật thông tin sinh viên có độ phức tạp O(1). Vì vậy, độ phức tạp tổng thể của phương thức này là O(n).
9. void timKiemSinhVien(string hoTen): Phương thức này tìm kiếm sinh viên trong danh sách dựa trên họ tên (hoTen). Việc duyệt qua từng nút trong danh sách và kiểm tra tên sinh viên có độ phức tạp O(n). Vì vậy, độ phức tạp tổng thể của phương thức này là O(n).

BÀI 2 (BÀI SỐ 30 TRONG DANH SÁCH BÀI TẬP)

**2.1.** Đề bài

﻿Tìm hiểu về cấu trúc đống nhị phân.

1. Ứng dụng cấu trúc đống nhị phân để cài đặt cấu trúc hàng đợi ưu tiên có các chức năng:
   1. Thêm một phần tử vào hàng đợi.
   2. Lấy giá trị của một phần tử.
   3. Xóa một phần tử khỏi hàng đợi.
   4. Lấy số phần tử trong hàng đợi.
2. Ứng dụng cấu trúc hàng đợi ưu tiên ở trên để cài đặt thuật toán tìm cây khung nhỏ nhất Prim.

**2.2.** Phân tích bài toán

* **Yêu cầu bài toán:**

Tìm hiểu về cấu trúc đống nhị phân và ứng dụng nó để cài đặt cấu trúc hàng đợi ưu tiên. Cấu trúc hàng đợi ưu tiên này có các chức năng sau:

1. Thêm một phần tử vào hàng đợi.
2. Lấy giá trị của một phần tử.
3. Xóa một phần tử khỏi hàng đợi.
4. Lấy số phần tử trong hàng đợi.

Để thực hiện yêu cầu trên, chúng ta cần xác định các lớp và các thuộc tính, phương thức của chúng. Dưới đây là phân tích chi tiết:

Lớp "PriorityQueue":

* Thuộc tính: head, size, capacity.
* Phương thức:
  + Constructor: Khởi tạo hàng đợi ưu tiên rỗng.
  + isEmpty: Kiểm tra hàng đợi có rỗng hay không.
  + isFull: Kiểm tra hàng đợi có đầy hay không.
  + getSize: Trả về số lượng phần tử trong hàng đợi.
  + heapify: Sắp xếp lại độ ưu tiên cho các phần tử.
  + insert: Thêm một phần tử vào hàng đợi với độ ưu tiên tương ứng.
  + remove: Xóa phần tử có độ ưu tiên cao nhất khỏi hàng đợi.
  + getMinium: Trả về giá trị của phần tử có độ ưu tiên cao nhất trong hàng đợi.

Lớp "PrimAlgorithm":

* Phương thức:
  + primMST: Thực hiện thuật toán tìm cây khung nhỏ nhất Prim và trả về cây khung nhỏ nhất.

**2.3.** Cài đặt các lớp và hàm main bằng C++:

* **Cài đặt lớp PriorityQueue:**

#ifndef **PRIORITYQUEUE\_H**#define **PRIORITYQUEUE\_H**#include <iostream>  
#include <vector>  
  
using namespace std;  
  
class PriorityQueue {  
private:  
 vector<int> heap;  
 int size;  
 int capacity;  
public:  
 PriorityQueue(int capacity) {  
 this->heap.resize(capacity + 1);  
 this->size = 0;  
 this->capacity = capacity;  
 }  
 bool isEmpty() {  
 return size == 0;  
 }  
 bool isFull() {  
 return size == capacity;  
 }  
 void insert(int element) {  
 if (isFull()) {  
 cout << "Hang doi uu tien da day." << endl;  
 return;  
 }  
 size++;  
 int index = size;  
 heap[index] = element;  
 while (index > 1 && heap[index] < heap[index / 2]) {  
 swap(heap[index], heap[index / 2]);  
 index = index / 2;  
 }  
 }  
 int remove() {  
 if (isEmpty()) {  
 cout << "Hang doi uu tien rong." << endl;  
 return -1;  
 }  
 int minElement = heap[1];  
 heap[1] = heap[size];  
 size--;  
 heapify(1);  
 return minElement;  
 }  
 void heapify(int index) {  
 int smallest = index;  
 int left = 2 \* index;  
 int right = 2 \* index + 1;  
 if (left <= size && heap[left] < heap[smallest])  
 smallest = left;  
 if (right <= size && heap[right] < heap[smallest])  
 smallest = right;  
 if (smallest != index) {  
 swap(heap[index], heap[smallest]);  
 heapify(smallest);  
 }  
 }  
 int getMinimum() {  
 if (isEmpty()) {  
 cout << "Hang doi uu tien rong." << endl;  
 return -1;  
 }  
 return heap[1];  
 }  
 void deleteMinimum() {  
 remove();  
 }  
 int getSize() {  
 return size;  
 }  
};  
#endif *// PRIORITYQUEUE\_H*

* **Cài đặt lớp PriorityAlgorithm:**

#ifndef **BAI30\_PRIMALGORITHM\_H**#define **BAI30\_PRIMALGORITHM\_H**#include <vector>  
#include "PriorityQueue.h"  
  
using namespace std;  
  
struct Edge {  
 int dest;  
 int weight;  
  
 Edge(int d, int w) : dest(d), weight(w) {}  
};  
  
class PrimAlgorithm {  
public:  
 static long long PrimMST(vector<vector<Edge>>& graph, int V) {  
 PriorityQueue pq(V);  
 vector<bool> visited(V, false);  
 long long minCost = 0;  
  
 int startVertex = 0;  
  
 visited[startVertex] = true;  
 for (const auto& edge : graph[startVertex]) {  
 pq.insert(edge.weight);  
 }  
 while (!pq.isEmpty()) {  
 int currentWeight = pq.remove();  
 int currentVertex = -1;  
 for (int i = 0; i < V; ++i) {  
 if (visited[i]) {  
 for (const auto& edge : graph[i]) {  
 if (!visited[edge.dest] && edge.weight == currentWeight) {  
 currentVertex = edge.dest;  
 break;  
 }  
 }  
 }  
 if (currentVertex != -1)  
 break;  
 }  
 if (currentVertex != -1) {  
 minCost += currentWeight;  
 visited[currentVertex] = true;  
  
 for (const auto& edge : graph[currentVertex]) {  
 if (!visited[edge.dest]) {  
 pq.insert(edge.weight);  
 }  
 }  
 }  
 }  
  
 return minCost;  
 }  
};  
#endif

* **Cài đặt hàm main:**

#include <iostream>  
#include <vector>  
#include "PrimAlgorithm.h"  
  
using namespace std;  
  
int main() {  
 int n;  
 int m;  
 cout << "Nhap so thanh pho: ";  
 cin >> n;  
 cout << "Nhap so tuyen duong: ";  
 cin >> m;  
 vector<vector<Edge>> graph(n);  
 for (int i = 0; i < m; ++i) {  
 int src, dest, weight;  
 cout << "Nhap tuyen duong thu " << i + 1 << ": ";  
 cin >> src >> dest >> weight;  
  
 graph[src - 1].push\_back(Edge(dest - 1, weight));  
 graph[dest - 1].push\_back(Edge(src - 1, weight));  
 }  
 long long minCost = PrimAlgorithm::PrimMST(graph, n);  
 cout << "Chi phi nho nhat: " << minCost << endl;  
 return 0;  
}

**2.4.** Phân tích thời gian chạy của từng phương thức có trong các lớp

**Các phương thức lớp PriorityQueue:**

1. Phương thức insert:
   * Thời gian chạy trong trường hợp tốt nhất: O(log n), trong đó n là số lượng phần tử hiện có trong hàng đợi ưu tiên.
   * Thời gian chạy trong trường hợp xấu nhất: O(log n).
   * Phương thức insert được sử dụng để chèn một phần tử vào hàng đợi ưu tiên. Trong quá trình chèn, phương thức sẽ thực hiện việc di chuyển phần tử vừa chèn lên trên trong heap để duy trì tính chất của heap.
2. Phương thức remove:
   * Thời gian chạy trong trường hợp tốt nhất: O(log n).
   * Thời gian chạy trong trường hợp xấu nhất: O(log n).
   * Phương thức remove được sử dụng để loại bỏ phần tử có giá trị nhỏ nhất trong hàng đợi ưu tiên và trả về giá trị đó. Sau khi loại bỏ, phương thức sẽ điều chỉnh lại heap để duy trì tính chất của heap.
3. Phương thức heapify:
   * Thời gian chạy trong trường hợp tốt nhất: O(log n).
   * Thời gian chạy trong trường hợp xấu nhất: O(log n).
   * Phương thức heapify được sử dụng để điều chỉnh heap sau khi loại bỏ phần tử. Phương thức này sẽ kiểm tra tính chất của heap từ một vị trí cụ thể và điều chỉnh các phần tử để đảm bảo tính chất đó.
4. Phương thức getMinimum:
   * Thời gian chạy: O(1).
   * Phương thức getMinimum trả về giá trị nhỏ nhất trong hàng đợi ưu tiên mà không loại bỏ phần tử này khỏi hàng đợi.
5. Phương thức deleteMinimum:
   * Thời gian chạy: O(log n).
   * Phương thức deleteMinimum loại bỏ phần tử nhỏ nhất khỏi hàng đợi ưu tiên.
6. Phương thức getSize:
   * Thời gian chạy: O(1).
   * Phương thức getSize trả về số lượng phần tử hiện có trong hàng đợi ưu tiên.

**Các phương thức lớp PrimAlgorithm:**

1. Phương thức PrimMST:
   * Thời gian chạy trong trường hợp tốt nhất và trường hợp xấu nhất: O(V^2 log V), trong đó V là số lượng đỉnh trong đồ thị.
   * Phương thức PrimMST sử dụng thuật toán Prim để tìm cây khung nhỏ nhất của đồ thị.
     + Trong quá trình này, phương thức sử dụng PriorityQueue để chọn cạnh có trọng số nhỏ nhất để mở rộng cây khung.
     + Để xác định cạnh có trọng số nhỏ nhất, phương thức duyệt qua tất cả các đỉnh đã được thăm và kiểm tra các cạnh kết nối với các đỉnh này.
     + Thời gian chạy của các phép toán trên PriorityQueue là O(log V) và phương thức PrimMST thực hiện V lần duyệt qua các đỉnh, do đó tổng thời gian chạy là O(V^2 log V).

============HẾT============