**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

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



**Bài tập lớn môn học**

**Cấu trúc dữ liệu và giải thuật**

Giảng viên hướng dẫn: Ths. Phạm Xuân Tích

Sinh viên thực hiện: Hoàng Đức Trọng – Lớp CNTT 3 – K64

Đề tài: A: Xây dựng lớp quản lý sinh viên

B: Xây dựng cấu trúc hàng đợi

Xây dựng đồ thị vô hướng

***Hà Nội, tháng 11 năm 2024***

**Mục lục**

[Phần A: Bài toán quản lý sinh viên 3](#_Toc182185495)

[I. Đề bài. 3](#_Toc182185496)

[II. Phân tích bài toán. 3](#_Toc182185497)

[2.1. Xác định các yêu cầu của bài toán, xác định các lớp, các thuộc tính, các phương thức của lớp. 3](#_Toc182185498)

[2.1.1. Yêu cầu bài toán. 3](#_Toc182185499)

[2.1.2. Xác định và mô tả chức năng của các lớp, các cấu trúc, các thuộc tính, các phương thức của các lớp và các cấu trúc. 4](#_Toc182185500)

[2.1.2.1. Class SinhVien 4](#_Toc182185501)

[2.1.2.2. Class QuanLySinhVien 5](#_Toc182185502)

[2.1.2.3. Class App 6](#_Toc182185503)

[III. Cài đặt các lớp, các cấu trúc và hàm main bằng C++ 7](#_Toc182185504)

[3.1. Class SinhVien 7](#_Toc182185505)

[3.2. Class QuanLySinhVien, App và hàm main 9](#_Toc182185506)

[IV. Phân tích thời gian chạy của từng phương thức có trong các lớp 15](#_Toc182185507)

[4.1. Class SinhVien 15](#_Toc182185508)

[4.2. Class QuanLySinhVien 16](#_Toc182185509)

[4.3. Class App 17](#_Toc182185510)

[Phần B 18](#_Toc182185511)

[I. Đề tài 18](#_Toc182185512)

[II. Phân tích bài toán 18](#_Toc182185513)

[2.1. Xác định các yêu cầu của bài toán, xác định các lớp, các thuộc tính, các phương thức của lớp. 18](#_Toc182185514)

[2.1.1. Yêu cầu của bài toán 18](#_Toc182185515)

[2.1.2. Xác định và mô tả chức năng của các lớp, các cấu trúc, các thuộc tính, các phương thức của các lớp và các cấu trúc. 19](#_Toc182185516)

[2.1.2.1. Xây dựng cấu trúc dữ liệu queue 19](#_Toc182185517)

[2.1.2.2. Xây dựng lớp biểu diễn đồ thị vô hướng có trọng số bằng ma trận kề 21](#_Toc182185518)

[III. Cài đặt các lớp và hàm main bằng C++ 22](#_Toc182185519)

[3.1. Cài đặt queue 22](#_Toc182185520)

[3.2. Xây dựng lớp biểu diễn đồ thị vô hướng có trọng số bằng ma trận kề để thực hiện các phương thức của đề 23](#_Toc182185521)

[3.2.1. Xây dựng cấu trúc vector 23](#_Toc182185522)

[3.2.2. Xây dựng lớp biểu diễn đồ thị vô hướng có trọng số bằng ma trận kề để thực hiện các phương thức của đề và hàm main 26](#_Toc182185523)

[IV. Thời gian chạy của từng phương thức có trong các lớp 28](#_Toc182185524)

[4.1. Class Queue 28](#_Toc182185525)

[4.2. Class DoThi 30](#_Toc182185526)

[V. Tài liệu tham khảo 32](#_Toc182185527)

# Phần A: Bài toán quản lý sinh viên

# I. Đề bài.

**Quản lý sinh viên bằng vector**

Ứng dụng vector để quản lý các đối tượng là các sinh viên. Mỗi sinh viên cần quản lý các thông tin sau: Mã sinh viên, họ tên, ngày sinh, giới tính, lớp. Trong lớp sinh viên xây dựng toán tử nhập, xuất, so sánh theo họ tên.

Viết chương trình cho phép thực hiện chức năng sau:

- Nhập thêm các danh sách sinh viên, sau khi hoàn thành nhập thông tin của một sinh viên, chương trình đưa ra câu hỏi có nhập nữa không (c/k)? Nếu người dùng nhập: c thì tiếp tục nhập, nhập k thì kết thúc.

- Xóa đi một sinh viên

- Sửa đổi thông tin của một sinh viên bất kỳ trong danh sách

- Sắp xếp danh sách sinh viên theo họ tên

- Hiển thị toàn bộ danh sách sinh viên hiện có trong vector

- Tìm kiếm sinh viên theo họ tên

# II. Phân tích bài toán.

## 2.1. Xác định các yêu cầu của bài toán, xác định các lớp, các thuộc tính, các phương thức của lớp.

### 2.1.1. Yêu cầu bài toán.

Bài toán yêu cầu xây dựng một chương trình quản lý sinh viên. Chương trình có khả năng thực hiện các thao tác sau trên danh sách sinh viên:

* Nhập danh sách sinh viên: Người dùng có thể nhập thông tin của nhiều sinh viên vào hệ thống
* Thêm một sinh viên mới: Hệ thống cho phép thêm thông tin của một sinh viên mới vào danh sách hiện có.
* Xóa một sinh viên theo mã: Xóa một sinh viên khỏi danh sách dựa trên mã sinh viên.
* Sửa thông tin của một sinh viên: Cập nhật thông tin của một sinh viên cụ thể thông qua mã sinh viên.
* Sắp xếp danh sách sinh viên theo tên: Sắp xếp danh sách sinh viên theo tên để tiện theo dõi và quản lý.
* Hiển thị danh sách sinh viên: Hiển thị toàn bộ danh sách sinh viên hiện có trong hệ thống.
* Tìm kiếm sinh viên theo tên: Tìm kiếm một sinh viên trong danh sách theo tên hoặc một phần tên để hiển thị thông tin của sinh viên đó.

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

#### 2.1.2.1. Class SinhVien

Lớp SinhVien trong chương trình dùng để quản lý thông tin của một sinh viên. Cấu trúc của lớp này gồm các thuộc tính và phương thức sau:

|  |  |  |
| --- | --- | --- |
|  | Mô tả | Chức năng |
| Thuộc Tính | string ma | Lưu trữ mã sinh viên |
| string ten | Lưu trữ tên của sinh viên |
| string ns | Lưu trữ ngày sinh sinh viên |
| string gt | Lưu trữ giới tính sinh viên |
| string lop | lưu trữ lớp sinh viên |
| Phương thức | friend istream &operator >> (istream &is, SinhVien&) | Phương thức nhập thông tin của sinh viên từ người dùng. |
| friend ostream &operator << (ostream &os, SinhVien) | Phương thức xuất thông tin của sinh viên từ người dùng. |
| friend bool operator < (SinhVien,SinhVien) | So sánh hai sinh viên theo tên để sắp xếp. |
| string getMa() | Trả về mã sinh viên |
| string getTen() | Trả về tên sinh viên. |
| SinhVien() | Hàm tạo không đối |
| SinhVien(string ma, string ten, string ns, string gt, string lop) | Hàm tạo có đối, tạo đối tượng SinhVien với các thông tin cơ bản. |

#### 2.1.2.2. Class QuanLySinhVien

Lớp QuanLySinhVien quản lý danh sách sinh viên. Nó cung cấp các phương thức để thực hiện các thao tác trên danh sách sinh viên.

|  |  |  |
| --- | --- | --- |
|  | Mô tả | Chức năng |
| Thuộc tính | vector<SinhVien> v | Lưu trữ danh sách các sinh viên. Dùng cấu trúc vector để lưu trữ sinh viên. |
| Phương thức | istream &operator >> (istream &is, QuanLySinhVien &qlsv) | Cho phép nhập nhiều sinh viên từ người dùng. |
|  | ostream &operator<<(ostream &os, QuanLySinhVien qlsv) | Hiển thị toàn bộ danh sách sinh viên. |
| void themsinhvien() | Thêm một sinh viên mới vào danh sách sinh viên. |
| void xoasinhvien() | Xóa sinh viên khỏi danh sách dựa trên mã sinh viên. |
| SinhVien\*suathongtinsinhvien() | Sửa thông tin của sinh viên dựa trên mã sinh viên. |
| void sapxepsinhvien() | Sắp xếp danh sách sinh viên theo tên. |
| SinhVien\*timkiemsinhvientheoten() | Tìm và hiển thị thông tin sinh viên dựa trên tên sinh viên. |

#### 2.1.2.3. Class App

Lớp App đóng vai trò là giao diện người dùng, quản lý các thao tác với lớp QuanLySinhVien.

|  |  |  |
| --- | --- | --- |
|  | Mô tả | Chức năng |
| Thuộc tính | QuanLySinhVien qlsv | Đối tượng dùng để thao tác với danh sách sinh viên. |
| Phương thức | void luachon() | Cung cấp các tùy chọn để thực hiện các thao tác trên danh sách sinh viên. |

# III. Cài đặt các lớp, các cấu trúc và hàm main bằng C++

## 3.1. Class SinhVien

Class này đại diện cho một sinh viên, với các thuộc tính và phương thức để thao tác với dữ liệu của sinh viên.

|  |
| --- |
| class SinhVien{  private:  string ma, ten, ns, gt, lop;  public:  SinhVien(){}  SinhVien(string ma, string ten, string ns, string gt, string lop){  this->ma = ma;  this->ten = ten;  this->ns = ns;  this->gt = gt;  this->lop = lop;  }  friend istream &operator>>(istream &is, SinhVien&);  friend ostream &operator<<(ostream &os, SinhVien);  friend bool operator<(SinhVien,SinhVien);  string getMa(){  return ma;  }  string getTen(){  return ten;  }  };  istream &operator>>(istream &is, SinhVien &s){  cout << "Nhap ma sinh vien (chan): ";  is >> s.ma;  is.ignore();  cout << "Nhap ho ten sinh vien: ";  getline(is,s.ten);  cout << "Nhap ngay sinh sinh vien (dd/mm/yyyy): ";  is >> s.ns;  cout << "Nhap gioi tinh (Nam/Nu): ";  is >> s.gt;  cout << "Nhap lop: ";  is >> s.lop;  return is;  }  ostream &operator<<(ostream &os, SinhVien s){  os << setw(30) << s.ma << " | " << setw(15) << s.ten <<" | " << setw(5) << s.ns << " | " << setw(3) << s.gt << " | " << setw(3) << s.lop;  return os;  }  bool operator<(SinhVien a, SinhVien b){  return a.ten < b.ten;  } |

- Phương thức istream &operator>>(istream &is, SinhVien &s)

Nhận dữ liệu đầu vào từ người dùng cho một đối tượng SinhVien, bao gồm các thông tin như mã sinh viên, tên, ngày sinh, giới tính, và lớp học. Các dữ liệu này sẽ được lưu vào các thuộc tính tương ứng của đối tượng.

- Phương thức ostream &operator<<(ostream &os, SinhVien s)

Xuất thông tin của một đối tượng SinhVien, bao gồm mã sinh viên, tên, ngày sinh, giới tính, và lớp học ra màn hình.

- Phương thức bool operator<(SinhVien a, SinhVien b)

So sánh tên của hai đối tượng SinhVien. Trả về true nếu tên của đối tượng a nhỏ hơn tên của đối tượng b, dùng trong việc sắp xếp danh sách sinh viên theo tên.

- Phương thức string getMa()

Trả về mã sinh viên của đối tượng SinhVien.

- Phương thức string getTen()

Trả về tên sinh viên của đối tượng SinhVien.

## 3.2. Class QuanLySinhVien, App và hàm main

|  |
| --- |
| #include <bits/stdc++.h>  #include <vector>  #include <algorithm>  #include "sinhvien.cpp"  using namespace std;  #define ll long long  #define fastIO \  ios\_base::sync\_with\_stdio(0); \  cin.tie(0); \  cout.tie(0);  class QuanLySinhVien {  private:  vector<SinhVien> v;  public:  friend istream &operator>>(istream &is, QuanLySinhVien&);  friend ostream &operator<<(ostream &os, QuanLySinhVien);  void themsinhvien();  void xoasinhvien();  SinhVien \*suathongtinsinhvien();  void sapxepsinhvien();  SinhVien \*timkiemsinhvientheoten();  };  istream &operator>>(istream &is, QuanLySinhVien &qlsv) {  int n;  cout << "Nhap so luong sinh vien: ";  is >> n;  for(int i = 0; i < n; i++) {  is.ignore(1);  SinhVien sv;  is >> sv;  qlsv.v.push\_back(sv);  }  return is;  }  ostream &operator<<(ostream &os, QuanLySinhVien qlsv) {  for(auto s : qlsv.v) {  os << s << "\n";  }  return os;  }  void QuanLySinhVien::themsinhvien() {  char c;  do {  SinhVien sv;  cin >> sv;  v.push\_back(sv);  cout << "\nBan co muon nhap nua khong? (c/k): ";  cin >> c;  } while(c == 'c');  }  void QuanLySinhVien::xoasinhvien() {  string masv;  cout << "\nNhap ma sinh vien can xoa: ";  cin >> masv;  for (auto it = v.begin(); it != v.end(); ++it) {  if (it->getMa() == masv) {  v.erase(it);  cout << "Da xoa sinh vien co ma " << masv << endl;  return;  }  }  cout << "Khong tim thay sinh vien co ma " << masv << endl;  }  SinhVien \*QuanLySinhVien::suathongtinsinhvien() {  string masv;  cout << "\nNhap ma sinh vien can sua: ";  cin >> masv;  for (auto &sv : v) {  if (sv.getMa() == masv) {  cout << "Nhap thong tin moi cho sinh vien: \n";  cin >> sv;  return &sv;  }  }  cout << "Khong tim thay sinh vien voi ma: " << masv << "\n";  return nullptr;  }  void QuanLySinhVien::sapxepsinhvien() {  sort(v.begin(), v.end());  }  SinhVien \*QuanLySinhVien::timkiemsinhvientheoten() {  string ten;  cout << "\nNhap ten sinh vien can tim: ";  cin.ignore();  getline(cin,ten);  for (auto &sv : v) {  if (sv.getTen() == ten || sv.getTen().find(ten) != string::npos) {  cout << sv << "\n";  return &sv;  }  }  cout << "Khong tim thay sinh vien voi ten: " << ten << "\n";  return nullptr;  }  int menu() {  int select;  cout << "\n--------------------------------------------------";  cout << "\n-----------------------MENU-----------------------";  cout << "\n1. Nhap danh sach sinh vien";  cout << "\n2. Them mot sinh vien";  cout << "\n3. Xoa mot sinh vien";  cout << "\n4. Sua doi thong tin mot sinh vien";  cout << "\n5. Sap xep sinh vien theo ho ten";  cout << "\n6. Hien thi danh sach sinh vien";  cout << "\n7. Tim kiem thong tin theo ho ten";  cout << "\n8. Thoat";  cout << "\nMoi ban chon: ";  cin >> select;  if(1 <= select && select <= 8) return select;  return menu();  }  class App {  private:  QuanLySinhVien qlsv;  public:  App (QuanLySinhVien qlsv) {this->qlsv = qlsv;}  void luachon() {  do {  int mn = menu();  // system("cls");  switch(mn) {  case 1:  cin >> qlsv;  break;  case 2:  qlsv.themsinhvien();  break;  case 3:  qlsv.xoasinhvien();  break;  case 4:  qlsv.suathongtinsinhvien();  break;  case 5:  qlsv.sapxepsinhvien();  break;  case 6:  cout << qlsv;  break;  case 7: {  qlsv.timkiemsinhvientheoten();  break;  }  case 8:  return;  }  system("pause");  } while(1);  }  };  int main() {  QuanLySinhVien qlsv;  App app(qlsv);  app.luachon();  return 0;  } |

- Phương thức istream &operator>>(istream &is, QuanLySinhVien &qlsv)

+ Nhận số lượng sinh viên và thông tin của từng sinh viên từ người dùng. Mỗi sinh viên sẽ được nhập và lưu vào vector v của QuanLySinhVien.

- Phương thức ostream &operator<<(ostream &os, QuanLySinhVien qlsv)

+ Xuất danh sách sinh viên từ vector v ra màn hình, hiển thị thông tin của từng sinh viên.

- Phương thức void themsinhvien()

+ Nhập thông tin cho một sinh viên mới và thêm vào vector v. Sau khi thêm, hỏi người dùng có muốn thêm sinh viên khác không.

- Phương thức void xoasinhvien()

+ Nhập mã sinh viên từ người dùng và tìm kiếm trong vector v. Nếu có sinh viên với mã nhập vào, sinh viên đó sẽ bị xóa khỏi danh sách.

- Phương thức SinhVien \*suathongtinsinhvien()

+ Nhập mã sinh viên từ người dùng để tìm kiếm sinh viên cần sửa. Nếu tìm thấy sinh viên có mã đó, thông tin của sinh viên sẽ được nhập lại và cập nhật. Nếu không tìm thấy, thông báo rằng không có sinh viên với mã đã nhập.

- Phương thức void sapxepsinhvien()

+ Sắp xếp danh sách sinh viên trong vector v theo tên sử dụng hàm sort và toán tử < của SinhVien.

- Phương thức SinhVien \*timkiemsinhvientheoten()

+ Nhập tên sinh viên từ người dùng để tìm kiếm. Duyệt qua vector v, nếu tìm thấy sinh viên có tên trùng hoặc chứa tên đã nhập, sẽ hiển thị thông tin sinh viên đó và trả về con trỏ tới đối tượng sinh viên.

- Phương thức void luachon()

+ Hiển thị menu chức năng và nhận lựa chọn của người dùng. Tùy theo lựa chọn, thực hiện các thao tác với danh sách sinh viên (thêm, xóa, sửa, sắp xếp, hiển thị, hoặc tìm kiếm) bằng cách gọi các phương thức tương ứng của QuanLySinhVien.

# IV. Phân tích thời gian chạy của từng phương thức có trong các lớp

## 4.1. Class SinhVien

|  |  |  |
| --- | --- | --- |
| Phương thức | Thời gian | Phân tích |
| istream &operator >> (istream &is, SinhVien &s) | O(1) | Việc nhập dữ liệu cho một đối tượng SinhVien bao gồm một số thao tác nhập các chuỗi và số. Do chỉ nhập các thuộc tính của một đối tượng, thời gian chạy là hằng số. |
| ostream &operator << (ostream &os, SinhVien s) | O(1) | Việc xuất dữ liệu của một đối tượng SinhVien chỉ bao gồm thao tác in ra một số chuỗi và số cố định, nên thời gian chạy là hằng số. |
| bool operator < (SinhVien a, SinhVien b) | O(n) | So sánh tên của hai đối tượng SinhVien có thể cần duyệt qua toàn bộ tên (chuỗi ký tự). Tuy nhiên, do độ dài của tên là không đáng kể so với số lượng sinh viên, nên thời gian này thường được xem là O(1) trong thực tế |
| string getMa() | O(1) | Các phương thức này chỉ truy xuất giá trị của thuộc tính, nên thời gian chạy là hằng số. |
| string getTen() | O(1) | Các phương thức này chỉ truy xuất giá trị của thuộc tính, nên thời gian chạy là hằng số. |

## 4.2. Class QuanLySinhVien

|  |  |  |
| --- | --- | --- |
| Phương thức | Thời gian | Phân tích |
| istream &operator >> (istream &is, QuanLySinhVien &qlsv) | O(n) | Phương thức này nhập thông tin cho n sinh viên. Mỗi lần nhập có độ phức tạp O(1), nên tổng thời gian chạy là O(n) |
| ostream &operator << (ostream &os, QuanLySinhVien qlsv) | O(n) | Phương thức này duyệt qua vector v để xuất thông tin của từng sinh viên. Thời gian chạy là O(n) |
| void themsinhvien() | O(1) | Phương thức này thêm một sinh viên vào cuối vector v, nên thời gian chạy trung bình là O(1) nhờ cơ chế phân bổ động của vector trong C++. |
| void xoasinhvien() | O(n) | Phương thức này tìm kiếm sinh viên trong vector v theo mã. Việc tìm kiếm mất O(n) thời gian và xóa phần tử cũng tốn O(n) do phải di chuyển các phần tử còn lại. |
| SinhVien \*suathongtinsinhvien() | O(n) | Phương thức này tìm kiếm sinh viên theo mã để sửa thông tin. Thao tác tìm kiếm mất O(n) thời gian, còn việc sửa thông tin có độ phức tạp O(1). |
| void sapxepsinhvien() | O(nlogn) | Phương thức này sử dụng sort để sắp xếp vector v dựa trên tên sinh viên. Sắp xếp có độ phức tạp O(nlogn) trong trường hợp trung bình và trường hợp tốt nhất. |
| SinhVien \*timkiemsinhvientheoten() | O(n\*m) với m là độ dài của chuỗi tên | Phương thức này tìm kiếm sinh viên trong vector v theo tên, có thể so khớp một phần tên. Vì vậy, trong trường hợp xấu nhất, nó cần so sánh toàn bộ tên của nnn sinh viên, mỗi lần so sánh mất O(m). |

## 4.3. Class App

|  |  |  |
| --- | --- | --- |
| Phương thức | Thời gian | Phân tích |
| void luachon() | O(1) | Phương thức này chỉ hiển thị menu và gọi phương thức tương ứng dựa trên lựa chọn của người dùng. Do đó, thời gian chạy của nó phụ thuộc vào phương thức của QuanLySinhVien được gọi và không có tác động lớn từ luachon riêng biệt. |

# Phần B

# I. Đề tài

- Xây dựng cấu trúc dữ liệu hàng đợi

- Xây dựng lớp biểu diễn đồ thị vô hướng có trọng số bằng ma trận kề có các phương thức:

+ Nhập đồ thị từ file

+ Ghi đồ thị ra file

+ Duyệt đồ thị theo chiều rộng (BFS) để tìm đường đi ngắn nhất giữa 2 đỉnh bất kỳ

Viết hàm main thực hiện các công việc trên

# II. Phân tích bài toán

## 2.1. Xác định các yêu cầu của bài toán, xác định các lớp, các thuộc tính, các phương thức của lớp.

### 2.1.1. Yêu cầu của bài toán

- Xây dựng cấu trúc dữ liệu hàng đợi (Queue):

+ Tạo lớp biểu diễn hàng đợi để có thể lưu trữ và truy xuất các phần tử theo quy tắc "vào trước, ra trước" (FIFO).

+ Lớp này sẽ hỗ trợ các thao tác như:

* Thêm phần tử vào hàng đợi.
* Lấy phần tử ra khỏi hàng đợi.
* Kiểm tra xem hàng đợi có rỗng không.

- Xây dựng lớp biểu diễn đồ thị vô hướng có trọng số bằng ma trận kề (Adjacency Matrix):

* Tạo lớp biểu diễn đồ thị vô hướng có trọng số sử dụng ma trận kề. Các yêu cầu đối với lớp đồ thị bao gồm:
* Nhập đồ thị từ file:
* Viết phương thức để đọc dữ liệu từ file và xây dựng ma trận kề cho đồ thị.
* File sẽ chứa thông tin về các đỉnh và trọng số của các cạnh.
* Ghi đồ thị ra file:
  + Viết phương thức để ghi ma trận kề của đồ thị vào file, lưu lại cấu trúc của đồ thị.
* Duyệt đồ thị theo chiều rộng (Breadth-First Search - BFS):
  + Viết phương thức duyệt đồ thị theo chiều rộng (BFS) để tìm đường đi ngắn nhất giữa hai đỉnh bất kỳ.
  + Phương thức này sẽ trả về đường đi ngắn nhất giữa hai đỉnh hoặc thông báo nếu không có đường đi.

- Viết hàm main:

* Viết hàm main để thực hiện các chức năng trên:
  + Nhập dữ liệu của đồ thị từ file.
  + Ghi ma trận kề của đồ thị ra file.
  + Duyệt đồ thị theo chiều rộng (BFS) để tìm đường đi ngắn nhất giữa hai đỉnh.
* main sẽ giúp kiểm tra và chạy từng chức năng của chương trình theo đúng yêu cầu.

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

#### 2.1.2.1. Xây dựng cấu trúc dữ liệu queue

|  |  |  |
| --- | --- | --- |
|  | Mô tả | Chức Năng |
| Thuộc tính | int n | Lưu trữ số lượng phần tử hiện có trong hàng đợi (size). |
| int cap | Dung lượng tối đa của mảng lưu trữ hàng đợi tại thời điểm hiện tại. Nếu số phần tử vượt quá dung lượng này, mảng sẽ được mở rộng (double capacity). |
| int F | Chỉ số của phần tử đầu tiên (front) trong hàng đợi. |
| int L | Chỉ số tiếp theo của phần tử cuối cùng (back) trong hàng đợi. Để biết phần tử cuối, ta truy cập buf[(L ? L : cap) - 1]. |
| T \*buf | Con trỏ đến mảng động lưu trữ các phần tử của hàng đợi. |
| Phương thức | Queue() | Khởi tạo hàng đợi rỗng với n = cap = F = L = 0 và buf = NULL. |
| ~Queue() | Giải phóng bộ nhớ đã cấp phát cho mảng buf khi hàng đợi bị hủy. |
| bool empty() | Kiểm tra xem hàng đợi có rỗng không. |
| int size() | Trả về số lượng phần tử hiện có trong hàng đợi (giá trị của n). |
| T &front() | Trả về tham chiếu đến phần tử đầu tiên trong hàng đợi, phần tử tại buf[F] |
| T &back() | Trả về tham chiếu đến phần tử cuối cùng trong hàng đợi. |
| void pop() | Xóa phần tử đầu tiên (front) khỏi hàng đợi. |
| void push(T x) | Thêm một phần tử mới x vào cuối hàng đợi. |

#### 2.1.2.2. Xây dựng lớp biểu diễn đồ thị vô hướng có trọng số bằng ma trận kề

|  |  |  |
| --- | --- | --- |
|  | Mô tả | Chức năng |
| Thuộc tính | int V | Lưu trữ số lượng đỉnh của đồ thị. Được sử dụng để xác định kích thước của ma trận kề. |
| VECTOR<VECTOR<int>> matran | Ma trận kề lưu trữ đồ thị vô hướng có trọng số. Nếu có cạnh giữa hai đỉnh i và j với trọng số w, thì matran[i][j] = w và matran[j][i] = w. Nếu không có cạnh, giá trị mặc định là 0. |
| Phương thức | void NhapFile() | Đọc đồ thị từ file "input.txt". |
| void XuatFile() | Ghi đồ thị ra file "output.txt". |
| VECTOR<int> BFS (int batdau, int ketthuc) | Tìm đường đi ngắn nhất giữa hai đỉnh batdau và ketthuc bằng thuật toán duyệt theo chiều rộng (BFS). |

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

## 3.1. Cài đặt queue

|  |
| --- |
| #ifndef \_\_queue\_cpp\_\_  #define \_\_queue\_cpp\_\_  template<class T>  struct Queue{  int n,cap,F,L; // cap - capaccity, n-size, F-First, L-1 - Last  T \*buf;  Queue(){  n=cap=F=L=0; buf = NULL;  }  ~Queue(){  if(buf) delete []buf;  }  bool empty(){  return n==0;  }  int size(){  return n;  }  T &front(){  return buf[F];  }  T &back(){  return buf[(L?L:cap)-1];  }  void pop(){  F=(F+1)%cap;  n--;  }  void push(T x){  if(n==cap){ //mo rong mang  cap=cap?cap\*2:1;  T \*tem = buf;  buf = new T[cap];  for (int i=F,j=0; i<F+n;i++){  buf[j++] = tem[i%n];  }  if(tem){  delete []tem;  }    F=0;  L=n;  }  buf[L] = x;  L=(L+1)%cap;  n++;  }  };  #endif |

## 3.2. Xây dựng lớp biểu diễn đồ thị vô hướng có trọng số bằng ma trận kề để thực hiện các phương thức của đề

### 3.2.1. Xây dựng cấu trúc vector

|  |
| --- |
| template <class T>  struct vec\_ite //bo lap nguoc  {  T \*cur;  vec\_ite(T \*c=0) {cur=c;}  T &operator\*() {return \*cur;}  bool operator!=(vec\_ite<T> rit) {return cur!=rit.cur;}  vec\_ite<T> &operator++(){cur=cur-1;return cur;} //++it  vec\_ite<T> &operator++(int) //it++  {  vec\_ite<T> tem=cur;  cur=cur-1;  return tem;  }  };  template <class T>  struct VECTOR  {  int n,cap;  T \*buf;  VECTOR() {n=cap=0; buf=NULL;}  VECTOR(int \_n,T x=0)  {  n=cap=\_n;  buf=new T[cap];  for(int i=0;i<n;i++) buf[i]=x;  }  ~VECTOR() {if(buf) delete[]buf;}  int size() {return n;}  bool empty() {return n==0;}  T &front(){return buf[0];}  T &back() {return buf[n-1];}  void extra(int k) //mo rong bo nho  {  if(k>cap)  {  cap=k;  T\*tem=buf;  buf = new T[k];  for(int i=0;i<n;i++) buf[i]=tem[i];  if(tem) delete [] tem;  }  }  void pop\_back() {n--;}  void push\_back(T x)  {  if(n==cap) extra(cap?cap\*2:1);  buf[n++]=x;  }  void resize(int k,T x=0)  {  if(n<k)  {  extra(k);  for(int i=n;i<k;i++) buf[i]=x;  }  n=k;  }  int capacity() {return cap;}  typedef T\*iterator;  iterator begin(){return buf;}  iterator end(){return buf+n;}  T &operator[](int k) {return buf[k];}  void insert(iterator it,T x) //chen phan tu  {  if(n==cap) {int k=it-buf; extra(cap?cap\*2:1); it=buf+k;}  for(iterator it1=buf+n-1;it1>=it;it1--) \*(it1+1)=\*it1; //moi phan tu tien 1 buoc  \*it=x; //chen vao  n++;  }  void erase(iterator it)  {  for(iterator it1=it+1;it1<buf+n;it1++) \*(it1-1)=\*it1;  n--;  }  typedef vec\_ite<T> reverse\_iterator;  reverse\_iterator rbegin() {return buf+n-1;}  reverse\_iterator rend() {return buf-1;}  }; |

### 3.2.2. Xây dựng lớp biểu diễn đồ thị vô hướng có trọng số bằng ma trận kề để thực hiện các phương thức của đề và hàm main

|  |
| --- |
| class DoThi{  int V;  VECTOR<VECTOR<int>> matran;  public:  void NhapFile(){  fstream fin;  fin.open("input.txt", ios::in);  fin >> V;  matran.resize(V, VECTOR<int>(V,0)); // khoi tao ma tran  for (int i=0; i<V; i++){  for(int j=0; j<V; j++){  fin >> matran[i][j];  }  }  fin.close();  }  void XuatFile(){  fstream fout;  fout.open("output.txt", ios::out);  fout << V << "\n";  for(int i=0; i<V; i++){  for(int j=0; j<V; j++){  fout << matran[i][j] << " ";  }  }  fout.close();  }  VECTOR<int> BFS(int batdau, int ketthuc){  if(batdau == ketthuc) return {batdau};  VECTOR<int> khoangcach(V, INT\_MAX);  VECTOR<int> cha(V, -1);  khoangcach[batdau] = 0;  Queue<int> q;  q.push(batdau);  while (!q.empty()){  int u = q.front();  q.pop();  for (int v=0; v<V; v++){  if(matran[u][v] != 0 && khoangcach[v] == INT\_MAX){  khoangcach[v] = khoangcach[u] + 1;  cha[v] = u;  q.push(v);  if(v == ketthuc) break;  }  }  }  if(khoangcach[ketthuc] == INT\_MAX) return {};  VECTOR<int> duongdi;  for(int v=ketthuc; v!=-1; v=cha[v]){  duongdi.push\_back(v);  }  reverse(duongdi.begin(), duongdi.end());  return duongdi;  }  void in(){ // check  for(int i=0; i<V; i++){  for(int j=0; j<V; j++){  cout << matran[i][j] << " ";  }  cout << endl;  }  }  };  int main(){  DoThi dothi;  dothi.NhapFile();  dothi.XuatFile();  cout << "Do thi vua nhap tu file la: \n";  dothi.in();  int batdau, ketthuc;  cout << "Nhap dinh bat dau: ";  cin >> batdau;  cout << "Nhap dinh ket thuc: ";  cin >> ketthuc;  VECTOR<int> path = dothi.BFS(batdau, ketthuc);  if(!path.empty()){  cout << "Duong di ngan nhat giua " << batdau << " va " << ketthuc << " la: ";  for(int i=0; i<path.size(); i++){  cout << path[i];  if(i!=path.size()-1) cout << " -> ";  }  cout << endl;  }  else{  cout << "Khong ton tai duong di giua" << batdau << " va " << ketthuc << endl;  }  return 0;  } |

# IV. Thời gian chạy của từng phương thức có trong các lớp

## 4.1. Class Queue

|  |  |  |
| --- | --- | --- |
| Phương thức | Thời gian | Phân tích |
| Queue() | O(1) | Khởi tạo các biến với độ phức tạp thời gian O(1). |
| ~Queue() | O(1) | Kiểm tra buf và giải phóng bộ nhớ nếu buf không phải là NULL, có độ phức tạp thời gian O(1). |
| bool empty() | O(1) | So sánh biến n với 0, có độ phức tạp thời gian O(1). |
| int size() | O(1) | Trả về giá trị của n, có độ phức tạp thời gian O(1). |
| T &front() | O(1) | Truy cập phần tử tại chỉ số F trong mảng buf, có độ phức tạp thời gian O(1). |
| T &back() | O(1) | Tính chỉ số của phần tử cuối cùng bằng (L ? L : cap) - 1 và truy cập phần tử đó trong mảng buf, có độ phức tạp thời gian O(1). |
| void pop() | O(1) | Cập nhật F bằng (F + 1) % cap để chuyển con trỏ đầu hàng đợi sang phần tử tiếp theo.  Giảm giá trị của n xuống 1.  Cả hai phép toán này đều có độ phức tạp thời gian O(1)O(1)O(1). |
| void push(T x) | O(1) trung bình, O(n) khi mở rộng | - Trường hợp không cần mở rộng:  Nếu n < cap (tức là còn chỗ trong hàng đợi), thực hiện:  Thêm x vào vị trí L trong buf.  Cập nhật L = (L + 1) % cap.  Tăng n lên 1.  Các phép toán này có độ phức tạp thời gian O(1)  - Trường hợp cần mở rộng:  Nếu n == cap (hàng đợi đầy), cần mở rộng kích thước mảng:  Tăng cap gấp đôi: cap = cap ? cap \* 2 : 1.  Tạo một mảng mới buf với kích thước mới cap.  Sao chép các phần tử từ tem (mảng cũ) sang buf theo đúng thứ tự (phải xử lý vòng lặp của mảng cũ bằng cách sử dụng i % n). Việc sao chép này có độ phức tạp thời gian O(n).  Xóa mảng cũ tem.  Cập nhật lại F = 0 và L = n. |

## 4.2. Class DoThi

|  |  |  |
| --- | --- | --- |
| Phương thức | Thời gian | Phân tích |
| void NhapFile() | O(V^2) | Mở và đọc số đỉnh V, có độ phức tạp thời gian O(1).  Khởi tạo ma trận matran với kích thước V x V, trong đó mỗi phần tử là một VECTOR<int> với độ phức tạp O(V^2) do phải cấp phát bộ nhớ.  Đọc từng phần tử của ma trận từ file, với mỗi cặp đỉnh (i, j), chi phí là O(1).  Tổng thời gian đọc và lưu trữ ma trận là O(V^2). |
| void XuatFile() | O(V^2) | Ghi giá trị của V vào file có độ phức tạp O(1)O(1)O(1).  Duyệt qua từng phần tử của ma trận matran và ghi từng phần tử vào file, với mỗi phần tử có chi phí O(1).  Với V x V phần tử trong ma trận, tổng độ phức tạp thời gian là O(V2)O(V^2)O(V2). |
| VECTOR<int> BFS(int batdau, int ketthuc) | O(V^2) | Khởi tạo các vector khoangcach và cha với kích thước V, mỗi phép khởi tạo có độ phức tạp O(V).  Khởi tạo hàng đợi Queue<int> và thêm batdau vào hàng đợi, có độ phức tạp O(1).  Trong khi hàng đợi không rỗng, lặp qua từng đỉnh u và kiểm tra các cạnh từ u đến từng đỉnh v.  Với mỗi đỉnh u, duyệt qua tất cả các đỉnh v để kiểm tra cạnh (u, v), điều này có độ phức tạp O(V) cho mỗi đỉnh u.  Tổng cộng, vòng lặp sẽ duyệt tất cả các cạnh, với độ phức tạp thời gian O(V^2) (vì trong ma trận kề, số cạnh tối đa là V^2).  Sau khi tìm thấy đường đi, tạo VECTOR<int> duongdi và lần theo mảng cha để dựng lại đường đi từ ketthuc về batdau. Việc này có độ phức tạp O(V). |

# V. Tài liệu tham khảo