**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: Bùi Minh Thương - Lớp CNTT 3 - K64

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

B. Xây dựng cây tìm kiếm nhị phân

Viết chương trình quản lý lịch công tác trong tháng

***Hà Nội, tháng 11 năm 2024***

Mục lục

[Phần A: Bài toán quản lý sinh viên 2](#_Toc182262210)

[1. Đề bài. 2](#_Toc182262211)

[**2. Phân tích bài toán.** 2](#_Toc182262212)

[2.1. Xác định yêu cầu bài toán, xác định các lớp, các thuộc tính, các phương thức của lớp. 2](#_Toc182262213)

[**3. Cài đặt các lớp, các cấu trúc và hàm main bằng C++** 7](#_Toc182262214)

[3.1. Class sinhvien. 7](#_Toc182262215)

[3.2. Class class\_. 10](#_Toc182262216)

[**4. Phân tích thời gian chạy của các lớp** 19](#_Toc182262217)

[4.1. Class sinhvien 19](#_Toc182262218)

[4.2. Class class\_ 20](#_Toc182262219)

[4.3. Class manage 22](#_Toc182262220)

[**Phần B.** 23](#_Toc182262221)

[**1. Đề bài** 23](#_Toc182262222)

[**Phần B sẽ chia làm 2 phần vì có 2 yêu cầu nhỏ của đề bài** 23](#_Toc182262223)

[**2. Cây tìm kiếm nhị phân (BST).** 23](#_Toc182262224)

[2..1. Xác định các yêu cầu của bài toán, xác định các lớp, các thuộc tính, các phương thức của lớp. 23](#_Toc182262225)

[**-** Yêu cầu bài toán: 24](#_Toc182262226)

[+ Xây dựng lớp tìm kiếm nhị phân. 24](#_Toc182262227)

[- Tạo lớp cây tìm kiếm với chức năng có thể tự sắp xếp dữ liệu. 24](#_Toc182262228)

[- Lớp sẽ có các phương thức như: Thêm một node mới, chèn, xóa, tìm kiếm, hiển thị, kiểm tra cây. 24](#_Toc182262229)

[2.2. Cài đặt các lớp và hàm main bằng C++. 25](#_Toc182262230)

[2.3. Phân tích thời gian chạy của lớp. 28](#_Toc182262231)

[**3. Quản lý lịch công tác trong tháng.** 29](#_Toc182262232)

[3.1. Phân tích bài toán 29](#_Toc182262233)

[**+** Chương trình quản lý công tác trong tháng. 30](#_Toc182262234)

[1. Xây dựng lớp cây tìm kiếm nhị phân (BST): 30](#_Toc182262235)

[ Trong mỗi nút NgàyLàmViệc, lưu trữ một danh sách liên kết (linked list) chứa các công việc cần làm trong ngày đó. 30](#_Toc182262236)

[2. Thông tin công việc: 30](#_Toc182262237)

[- Một công việc sẽ bao gồm: 30](#_Toc182262238)

[ Trạng thái công việc (trạngThái): đã hoàn tất hoặc chưa thực hiện 30](#_Toc182262239)

[4. Cấu trúc dữ liệu cần sử dụng: 30](#_Toc182262240)

[3.2. Cài đặt các lớp và hàm main bằng C++. 33](#_Toc182262241)

[Thuộc tính 47](#_Toc182262242)

[Phương thức riêng (Private Methods) 47](#_Toc182262243)

[Phương thức công khai (Public Methods) 47](#_Toc182262244)

[3.3. Phân tích thời gian chạy của lớp. 55](#_Toc182262245)

[**Tài liệu tham khảo.** 59](#_Toc182262246)

# Phần A: Bài toán quản lý sinh viên

## 1. Đề bài.

**Quản lý sinh viên bằng vector và list**

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

Viết chương trình cho phép thực hiện chức năng sau

* Nhập thêm sinh viên vào danh sách sinh viên với mỗi mã sinh viên chẵn thì đưa vào vector và sinh viên chẵn đưa vào list.
* Tìm sinh viên theo mã sinh viên.
* Sắp xếp sinh viên.
* Xóa một sinh viên.
* Hiển thị toàn bộ danh sách sinh viên.
* Tìm sinh viên nhiều điểm nhất.
* Thêm một sinh viên.
* Tìm kiếm một sinh viên.

## **2. Phân tích bài toán.**

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

**2.1.1. Xác định 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 trên danh sách sinh viên:

* Nhập danh sách sinh viên: Người dùng nhập thông tin của sinh viên vào hệ thống.
* Xuất danh sách sinh viên: Hệ thống sẽ xuất danh sách viên.
* Xóa sinh viên: Hệ thống sẽ xóa một sinh viên được khi người dùng nhập vào một thông tin mà hệ thông yêu cầu nhập vào.
* Sắp xếp sinh viên: Sắp xếp danh sách sinh viên bằng cách sử dụng những phương thức sort có sẵn của list và vector.
* Xóa đi một sinh viên: Xóa sinh viên khỏi danh sách.
* Hiển thị danh sách sinh viên: Hiển thị danh sách sinh viên sau khi nhập danh sách.
* Tìm sinh viên nhiều điểm nhất: Trả về sinh viên có điểm cao nhất.
* Thêm một sinh viên: Nhập dữ liệu của sinh viên cần thêm rồi đưa vào hệ thống.
* Tìm kiếm một sinh viên bất kì: Nhập thông tin sinh viên theo hệ thống yêu cầu rồi trả về sinh viên cần tìm.

**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.**

**A, Class sinhvien**

Lớp quản lý sinh viên dùng để quản lý thông tin của một sinh viên. Cấu trúc của lớp này gồm phương thức và thuộc tính sau.

|  |  |
| --- | --- |
| **Mô tả** | **Chức năng** |
| **Thuộc Tính** |  |
| string name | Lưu trữ tên của sinh viên |
| int stu\_num | Lưu trữ số sinh viên (mã sinh viên) |
| int age | Lưu trữ tuổi của sinh viên |
| float score | Lưu trữ điểm của sinh viên |
| **Phương thức** |  |
| friend istream& operator >> | Phương thức nhập thông tin của sinh viên từ người dùng |
| friend ostream& operator << | Phương thức xuất thông tin của sinh viên ra màn hình |
| student operator + | Phương thức cộng điểm của hai sinh viên |
| student() | Hàm tạo không đối |
| student(string name, int stu\_num, int age, float score) | Hàm tạo có đối, tạo đối tượng student với các thông tin cơ bản |
| friend bool operator > | So sánh hai sinh viên theo điểm để sắp xếp |
| int get\_stu\_num() | Trả về số sinh viên (mã sinh viên) |
| float get\_score() | Trả về điểm của sinh viên |

**B, Class lop**

Lớp lop dùng để quản lý sinh viên trong một lớp với các thuộc tính và phương thức sau đây.

|  |  |
| --- | --- |
| **Mô tả** | **Chức năng** |
| **Thuộc Tính** |  |
| vector<student> s\_odd | Lưu trữ danh sách sinh viên với số sinh viên lẻ |
| list<student> s\_even | Lưu trữ danh sách sinh viên với số sinh viên chẵn |
| **Phương thức** |  |
| friend istream& operator >> | Phương thức nhập danh sách sinh viên từ người dùng |
| friend ostream& operator << | Phương thức xuất danh sách sinh viên ra màn hình |
| void push\_(student x) | Thêm sinh viên vào s\_odd nếu số sinh viên lẻ, vào s\_even nếu số sinh viên chẵn |
| student max\_() | Tìm sinh viên có điểm cao nhất từ cả hai danh sách |
| student\* find(int x) | Tìm sinh viên theo số sinh viên, trả về con trỏ tới sinh viên nếu tìm thấy, ngược lại trả về nullptr |
| void delete\_(int x) | Xóa sinh viên theo số sinh viên khỏi cả hai danh sách |
| void sort\_byscore() | Sắp xếp danh sách sinh viên giảm dần theo điểm |
| void output\_odd\_stu() | Xuất thông tin các sinh viên trong s\_odd ra màn hình |
| void output\_even\_stu() | Xuất thông tin các sinh viên trong s\_even ra màn hình |

**C, Class manage**

Lớp manage đóng vai trò là giao diện và thực hiện các chức hệ thống có thể làm được và theo yêu cầu người dùng.

|  |  |
| --- | --- |
| **Mô tả** | **Chức năng** |
| **Phương thức** |  |
| void quanlisinhvien() | Hàm quản lý các chức năng về sinh viên, bao gồm: |
|  | 1. Nhập sinh viên từ file sv.txt. |
|  | 2. Xuất danh sách sinh viên. |
|  | 3. Sắp xếp sinh viên theo điểm, xuất danh sách sinh viên lẻ và chẵn sau khi sắp xếp. |
|  | 4. Tìm và xuất sinh viên có điểm cao nhất. |
|  | 5. Tìm sinh viên theo mã sinh viên do người dùng nhập vào và xuất thông tin. |
|  | 6. Xóa sinh viên theo mã sinh viên do người dùng nhập vào. |
|  | 7. Thêm sinh viên mới bằng cách nhập thông tin từ người dùng và thêm vào danh sách. |
|  | 8. Thoát chương trình. |

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

### 3.1. Class sinhvien.

Class có các thuộc tính cũng như các phương thức để thao tác với kiểu dữ liệu sinh viên.

|  |
| --- |
| class student{  private:  string name;  int stu\_num;  int age;  float score;  public:  friend istream& operator >> (istream& in, student& s);  friend ostream& operator << (ostream& out, student& s);    student operator + (student s);  student(){  name = "";  stu\_num = 0;  age = 0;  score = 0;  }    student(string name, int stu\_num, int age, float score) {  this->name = name;  this->stu\_num = stu\_num;  this->age = age;  this->score = score;  }    friend bool operator > (student s1, student s2){  return s1.score > s2.score;  }    int get\_stu\_num(){  return stu\_num;  }    float get\_score(){  return score;  }    };  istream& operator>>(istream& in, student& s) {  getline(in, s.name);  in >> s.stu\_num >> s.age >> s.score;  in.ignore(1000, '\n');  return in;  }  ostream& operator << (ostream& out, student& s){  out << s.name << ' ' << s.stu\_num << ' ' << s.age << ' ' << s.score << endl;  return out;  }  student student::operator + (student s){  student temp = \*this;  temp.score = this->score + s.score;  return temp;  } |

Trong lớp gồm có phương thức và chức năng sau:

* Phương thức istream &operator>>(istream &in, student &s)  
  Nhận dữ liệu đầu vào từ người dùng cho một đối tượng student, bao gồm các thông tin như tên, số sinh viên, tuổi, và điểm. Các dữ liệu này sẽ được lưu vào các thuộc tính tương ứng của đối tượng.
* Phương thức ostream &operator<<(ostream &out, student &s)  
  Xuất thông tin của một đối tượng student, bao gồm tên, số sinh viên, tuổi, và điểm ra màn hình.
* Phương thức student operator+(student s)  
  Tính tổng điểm của hai đối tượng student. Phương thức này trả về một đối tượng mới với điểm là tổng điểm của đối tượng hiện tại và đối tượng s.
* Phương thức bool operator>(student s1, student s2)  
  So sánh điểm của hai đối tượng student. Trả về true nếu điểm của đối tượng s1 lớn hơn điểm của đối tượng s2, dùng trong việc sắp xếp danh sách sinh viên theo điểm.
* Phương thức int get\_stu\_num()  
  Trả về số sinh viên của đối tượng student.
* Phương thức float get\_score()  
  Trả về điểm của đối tượng student.

### 3.2. Class class\_.

Class\_ là lớp có các thuộc tính và phương thức nhằm hướng tới quản lý danh sách sinh viên lớp đó

|  |
| --- |
| class class\_{  private:  vector <student> s\_odd;  list <student> s\_even;  public:  friend istream& operator >> (istream& in, class\_& c);  friend ostream& operator << (ostream& out, class\_& c);  void push\_(student x){  if (x.get\_stu\_num() % 2 == 0){  s\_odd.push\_back(x);  }  else s\_even.push\_back(x);  }  student max\_() {  if (s\_odd.empty() && s\_even.empty()) {  return student();  }  student tmp1, tmp2;  if (!s\_odd.empty()) {  tmp1 = s\_odd[0];  for (auto x : s\_odd) {  if (x.get\_score() > tmp1.get\_score()) tmp1 = x;  }  }  if (!s\_even.empty()) {  tmp2 = s\_even.front();  for (auto x : s\_even) {  if (x.get\_score() > tmp2.get\_score()) tmp2 = x;  }  }  return (tmp1.get\_score() > tmp2.get\_score()) ? tmp1 : tmp2;  }  student \*find (int x){  for (auto& a : s\_odd){  if (a.get\_stu\_num() == x) return &a;  }  for (auto& a : s\_even){  if (a.get\_stu\_num() == x) return &a;  }  return nullptr;  }  void delete\_(int x){  if (s\_odd.empty() && s\_even.empty()) return;  s\_odd.erase(remove\_if(s\_odd.begin(), s\_odd.end(), [x](student& s){  return s.get\_stu\_num() == x;  }), s\_odd.end());  s\_even.remove\_if([x](student& s){  return s.get\_stu\_num() == x;  });  }  void sort\_byscore(){  //sx giam dan theo diem  //sx s\_odd  sort(s\_odd.begin(), s\_odd.end(), [] (student &s1, student &s2){  return s1 > s2;  });  //sx s\_even  s\_even.sort([](student &s1, student &s2){  return s1 > s2;  });  }  void output\_odd\_stu(){  for (auto s : s\_odd){  cout << s << endl;  }  }  void output\_even\_stu(){  for (auto s : s\_even){  cout << s << endl;  }  }  };  istream& operator >> (istream& in, class\_& c){  int n;  in >> n;  in.ignore(1000, '\n');  for (int i = 0 ; i < n ; i++){  student s;  in >> s;  if (s.get\_stu\_num() % 2 != 0) c.s\_odd.push\_back(s);  else c.s\_even.push\_back(s);  }  return in;  }  ostream& operator << (ostream& out, class\_& c){  cout << endl;  for (auto s : c.s\_odd){  out << s << endl;  }  for (auto s : c.s\_even){  out << s << endl;  }  return out;  } |

Trong lớp gồm phương thức và chức năng sau.

* **Phương thức** istream &operator>>(istream &in, class\_ &c)  
  Nhận dữ liệu đầu vào từ người dùng cho đối tượng class\_, bao gồm thông tin về số lượng sinh viên và các sinh viên có trong lớp. Các sinh viên sẽ được phân loại và lưu vào danh sách s\_odd hoặc s\_even tùy thuộc vào tính chẵn/lẻ của số sinh viên.
* **Phương thức** ostream &operator<<(ostream &out, class\_ &c)  
  Xuất thông tin của tất cả sinh viên trong đối tượng class\_, bao gồm cả các sinh viên có số lẻ trong s\_odd và số chẵn trong s\_even.
* **Phương thức** void push\_(student x)  
  Thêm một đối tượng student vào s\_odd nếu số sinh viên là số chẵn và vào s\_even nếu số sinh viên là số lẻ.
* **Phương thức** student max\_()  
  Tìm sinh viên có điểm cao nhất từ cả hai danh sách s\_odd và s\_even. Nếu cả hai danh sách trống, phương thức sẽ trả về một đối tượng student mặc định.
* **Phương thức** student\* find(int x)  
  Tìm kiếm sinh viên có số sinh viên là x trong cả s\_odd và s\_even. Phương thức trả về con trỏ tới sinh viên tìm được, hoặc nullptr nếu không tìm thấy.
* **Phương thức** void delete\_(int x)  
  Xóa sinh viên có số sinh viên là x khỏi cả s\_odd và s\_even nếu tìm thấy.
* **Phương thức** void sort\_byscore()  
  Sắp xếp s\_odd và s\_even theo điểm từ cao xuống thấp.
* **Phương thức** void output\_odd\_stu()  
  Xuất thông tin của các sinh viên trong s\_odd ra màn hình.
* **Phương thức** void output\_even\_stu()  
  Xuất thông tin của các sinh viên trong s\_even ra màn hình.

**3.3. Class manage và hàm main.**

Class này để tương tác với người dùng, thực hiện theo yêu cầu của người dùng.

|  |
| --- |
| #include <bits/stdc++.h>  #include "lop.cpp"  using namespace std;  using ll = long long;  class manage{  public:  void quanlisinhvien(){  class\_ c;  fstream fin;  do{  cout << "\n1. Nhap sinh vien\n";  cout << "2. Xuat sinh vien\n";  cout << "3. Sap xep sinh vien theo diem\n";  cout << "4. Sinh vien diem cao nhat\n";  cout << "5. Tim sinh vien\n";  cout << "6. Xoa sinh vien\n";  cout << "7. Them sinh vien\n";  cout << "8. Thoat";  cout << "\nnhap lua chon cua ban\n";  int x;  cin >> x;  system("cls");  if (x == 1) {  fin.open("sv.txt", ios::in);  fin >> c;  fin.close();    }  if (x == 2){  cout << "\ndanh sach sinh vien\n";  cout << c;  }  if (x == 3){  c.sort\_byscore();  cout << "\nsap xep sinh vien theo so le la\n";  c.output\_odd\_stu();  cout << "\nsap xep sinh vien theo so chan la\n";  c.output\_even\_stu();  }  if (x == 4){  cout << "sinh vien diem cao nhat la: ";  student a = c.max\_();  cout << a << endl;  }  if (x == 5){  cout << "\nnhap ma sinh vien can tim: ";  int a;  cin >> a;  cout << \*c.find(a) << endl;  }  if (x == 6){  cout << "\nnhap ma sinh vien can xoa";  int a;  cin >> a;  c.delete\_(a);  }  if (x == 7){  cout << "\nnhap thong tin sinh vien can them: \n";  cin.ignore();  string name;  getline(cin, name);  int stu\_num;  int age;  float score;  cin >> stu\_num >> age >> score;  student s(name, stu\_num, age, score);  c.push\_(s);  cout << s;  // student s;  // cin >> s;  // c.push\_(s);  }  if (x == 8) return ;  }  while (1);  }  };  main(){  manage m;  m.quanlisinhvien();  } |

Lớp và hàm gồm phương thức và chức năng sau.

* **Phương thức** void quanlisinhvien()  
  Quản lý thông tin sinh viên qua các lựa chọn tương tác trong một vòng lặp, bao gồm các tính năng: nhập, xuất, sắp xếp, tìm kiếm, xóa, và thêm sinh viên mới.

Các lựa chọn của người dùng được hiển thị dưới dạng menu:

* + **Nhập sinh viên**: Mở tệp sv.txt và đọc dữ liệu vào đối tượng class\_ c thông qua toán tử >> đã được định nghĩa trong lớp class\_.
  + **Xuất sinh viên**: In danh sách sinh viên từ đối tượng c ra màn hình bằng toán tử <<.
  + **Sắp xếp sinh viên theo điểm**: Gọi phương thức sort\_byscore() để sắp xếp s\_odd và s\_even trong c theo điểm từ cao xuống thấp. Sau đó, in ra danh sách sinh viên lẻ và chẵn.
  + **Sinh viên điểm cao nhất**: Gọi phương thức max\_() để tìm sinh viên có điểm cao nhất trong c và in ra màn hình.
  + **Tìm sinh viên**: Nhập mã sinh viên từ người dùng và tìm sinh viên có mã tương ứng trong c bằng phương thức find(). Nếu tìm thấy, in thông tin sinh viên đó.
  + **Xóa sinh viên**: Nhập mã sinh viên cần xóa và gọi phương thức delete\_() để xóa sinh viên khỏi c.
  + **Thêm sinh viên**: Nhập thông tin sinh viên mới, khởi tạo đối tượng student với thông tin đó và thêm vào c bằng phương thức push\_().
  + **Thoát**: Thoát khỏi vòng lặp và kết thúc chương trình.
* **Phương thức** main()  
  Khởi tạo đối tượng manage m và gọi phương thức quanlisinhvien() để bắt đầu quản lý sinh viên.

## **4. Phân tích thời gian chạy của các lớp**

### 4.1. Class sinhvien

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Thời gian** | **Phân tích** |
| student() | O(1) | Phương thức khởi tạo mặc định, gán giá trị mặc định cho các thuộc tính. Thời gian thực hiện không phụ thuộc vào số lượng dữ liệu. |
| student(string, int, int, float) | O(1) | Khởi tạo và gán giá trị cho các thuộc tính từ các tham số đầu vào. Mỗi lệnh gán đều có thời gian thực hiện O(1). |
| operator>>(istream& in, student& s) | O(1) cho từng sinh viên | Đọc dữ liệu đầu vào cho từng thuộc tính của student. Mỗi thao tác getline, >>, và ignore có thời gian O(1). Không phụ thuộc vào kích thước của dữ liệu khác. |
| operator<<(ostream& out, student& s) | O(1) cho từng sinh viên | Xuất thông tin của sinh viên ra màn hình. Các thao tác xuất << là O(1) cho từng thuộc tính, tổng cộng có bốn thao tác. |
| operator+(student s) | O(1) | Tính tổng điểm của hai sinh viên, tạo bản sao student và gán score là tổng điểm. Tất cả các thao tác đều là O(1). |
| operator>(student s1, student s2) | O(1) | So sánh điểm số của hai sinh viên để xác định sinh viên nào có điểm cao hơn. Đây là một phép so sánh trực tiếp với thời gian thực hiện O(1). |
| get\_stu\_num() | O(1) | Trả về giá trị của stu\_num, thời gian không phụ thuộc vào dữ liệu khác. |
| get\_score() | O(1) | Trả về giá trị của score, không có phép tính nào khác và không phụ thuộc vào dữ liệu. |

Các phương thức đều có độ phức tạp O(1) do chúng chỉ thực hiện các thao tác gán, lấy giá trị hoặc so sánh đơn giản. Các thao tác này không phụ thuộc vào kích thước của bất kỳ cấu trúc dữ liệu phức tạp nào khác.

### 4.2. Class class\_

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Thời gian** | **Phân tích** |
| istream& operator>>(istream& in, class\_& c) | O(n) | Đọc n đối tượng student từ đầu vào và phân loại chúng vào s\_odd hoặc s\_even dựa trên số sinh viên (stu\_num). Mỗi phép đọc và phân loại có độ phức tạp O(1). |
| ostream& operator<<(ostream& out, class\_& c) | O(m + k) | Xuất thông tin của tất cả sinh viên trong s\_odd (m phần tử) và s\_even (k phần tử). Mỗi phép xuất có độ phức tạp O(1). |
| void push\_(student x) | O(1) | Thêm student vào s\_odd hoặc s\_even tùy theo tính chẵn/lẻ của số sinh viên. Việc kiểm tra và thêm phần tử vào vector hoặc list có thời gian O(1). |
| student max\_() | O(m + k) | Tìm sinh viên có điểm cao nhất từ s\_odd và s\_even. Quét qua s\_odd (m phần tử) và s\_even (k phần tử) để so sánh điểm, tổng độ phức tạp là O(m + k). |
| student\* find(int x) | O(m + k) | Tìm sinh viên có số sinh viên (stu\_num) bằng x trong cả s\_odd và s\_even. Phải duyệt qua từng phần tử trong mỗi danh sách, nên độ phức tạp là O(m + k). |
| void delete\_(int x) | O(m + k) | Xóa sinh viên có số sinh viên bằng x trong cả s\_odd và s\_even. Hàm remove\_if trong vector và list đều có độ phức tạp O(m + k) trong trường hợp tệ nhất. |
| void sort\_byscore() | O(m log m + k log k) | Sắp xếp s\_odd và s\_even theo điểm số giảm dần. vector::sort có độ phức tạp O(m log m), và list::sort có độ phức tạp O(k log k). |
| void output\_odd\_stu() | O(m) | Xuất thông tin của tất cả sinh viên trong s\_odd. Mỗi phép xuất là O(1), nên tổng thời gian là O(m). |
| void output\_even\_stu() | O(k) | Xuất thông tin của tất cả sinh viên trong s\_even. Mỗi phép xuất là O(1), nên tổng thời gian là O(k). |

Trong bảng, m là số phần tử trong s\_odd, và k là số phần tử trong s\_even.

### 4.3. Class manage

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Thời gian** | **Phân tích** |
| void quanlisinhvien() | O(n \* m log m + n \* k log k) | Vòng lặp do-while chứa các thao tác quản lý sinh viên như nhập, xuất, tìm kiếm, và xóa sinh viên, tất cả dựa trên lựa chọn của người dùng. Thời gian phụ thuộc vào số lượng phần tử n, m, k trong danh sách và các phương thức của lớp class\_. |
| **Lựa chọn 1: Nhập sinh viên** | O(n) | Đọc dữ liệu sinh viên từ tệp sv.txt và thêm vào danh sách sinh viên. Thời gian đọc n sinh viên từ tệp có độ phức tạp O(n). |
| **Lựa chọn 2: Xuất sinh viên** | O(m + k) | Xuất danh sách sinh viên từ s\_odd và s\_even. Với m phần tử trong s\_odd và k phần tử trong s\_even, độ phức tạp là O(m + k). |
| **Lựa chọn 3: Sắp xếp sinh viên theo điểm** | O(m log m + k log k) | Gọi hàm sort\_byscore() để sắp xếp s\_odd và s\_even theo điểm số giảm dần. sort có độ phức tạp là O(m log m) và list::sort có độ phức tạp O(k log k). |
| **Lựa chọn 4: Sinh viên điểm cao nhất** | O(m + k) | Gọi hàm max\_() để tìm sinh viên có điểm cao nhất từ hai danh sách s\_odd và s\_even, với độ phức tạp là O(m + k) do cần duyệt qua toàn bộ phần tử. |
| **Lựa chọn 5: Tìm sinh viên** | O(m + k) | Gọi hàm find(int x) để tìm sinh viên với stu\_num bằng x trong s\_odd và s\_even. Duyệt qua mỗi danh sách với độ phức tạp là O(m + k). |
| **Lựa chọn 6: Xóa sinh viên** | O(m + k) | Gọi hàm delete\_(int x) để xóa sinh viên với stu\_num bằng x trong s\_odd và s\_even. Hàm remove\_if có độ phức tạp O(m + k). |
| **Lựa chọn 7: Thêm sinh viên** | O(1) | Thêm một sinh viên vào s\_odd hoặc s\_even tùy vào số sinh viên là chẵn hay lẻ. Việc thêm vào vector hoặc list có độ phức tạp O(1). |

Trong bảng, n là số sinh viên, m là số sinh viên lẻ, và k là số sinh viên chẵn.

# **Phần B.**

## **1. Đề bài**

- Xây dựng lớp cây tìm kiếm nhị phân.

- Viết chương trình quản lý lịch công tác trong tháng đơn giản: cho phép nhập vào nội dung công việc cần làm theo ngày, theo giờ. Trong một ngày có thể có nhiều công việc, mỗi công việc có giờ bắt đầu, tên công việc, nội dung công việc, tính chất công việc {r*ất quan trọng, quan trọng, bình thường, không cần thiết*}, …

Chương trình có các chức năng chính như sau:

* Nhập nội dung công việc cần làm theo ngày, theo giờ.
* Xem lịch công tác theo ngày yêu cầu.
* Xem các công việc theo tính chất: rất quan trọng, quan trọng, …
* Xem các công việc đã hoàn tất.
* Xem các công việc chưa thực hiện.
* Xem các công việc từ ngày a đến ngày b.
* Xóa hay điều chỉnh lịch công tác. Nếu sau khi điều chỉnh, ngày nào không còn việc phải làm sẽ xóa khỏi lịch công tác.

Yêu cầu

* Cấu trúc dữ liệu được sử dụng là cây nhị phân tìm kiếm (BST), trong đó:
* Mỗi nút trên cây BST là một ngày của lịch công tác.
* Trong mỗi nút ngày trên cây lại chứa một danh sách liên kết lưu thông tin các công việc.
* Khi thêm một công việc vào một ngày đã tồn tại trên cây thì công việc này sẽ được đưa vào danh sách liên kết chứa các công việc theo thứ tự tăng dần của giờ bắt đầu.

## **Phần B sẽ chia làm 2 phần vì có 2 yêu cầu nhỏ của đề bài**

## **2****. Cây tìm kiếm nhị phân (BST).**

**2.. 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.**

**-** Yêu cầu bài toán:

+ Xây dựng lớp tìm kiếm nhị phân.

* Tạo lớp cây tìm kiếm với chức năng có thể tự sắp xếp dữ liệu.
* Lớp sẽ có các phương thức như: Thêm một node mới, chèn, xóa, tìm kiếm, hiển thị, kiểm tra cây.

**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 cây tìm kiếm nhị phân (BST).**

|  |  |
| --- | --- |
| **Mô tả** | **Chức năng** |
| **Thuộc Tính** |  |
| T data | Lưu trữ dữ liệu của nút |
| node \*left | Con trỏ trỏ đến nút con bên trái |
| node \*right | Con trỏ trỏ đến nút con bên phải |
| **Phương thức** |  |
| node \*make\_node(T x) | Tạo một nút mới có dữ liệu là `x`, với con trỏ `left` và `right` trỏ đến `NULL` |
| node \*insert(node \*root, T x) | Chèn một giá trị `x` vào cây nhị phân tìm kiếm tại vị trí phù hợp |
| void display(node \*root) | Duyệt cây theo thứ tự giữa (in-order traversal) và in ra các giá trị từ nhỏ đến lớn |
| node \*min\_node(node \*root) | Tìm và trả về con trỏ đến nút có giá trị nhỏ nhất trong cây |
| node \*delete\_node(node \*root, T x) | Xóa một nút có giá trị `x` khỏi cây với ba trường hợp: không có con, có một con, và có hai con |
| node\* find(node \*root, T x) | Tìm và trả về con trỏ đến nút có giá trị `x` hoặc `nullptr` nếu không tìm thấy |
| bool emty(node \*root) | Kiểm tra xem cây có rỗng hay không, trả về `true` nếu cây rỗng, `false` nếu không |
| T& get\_data() | Trả về tham chiếu đến dữ liệu của nút hiện tại |

Vì cây nhị phân tìm kiếm (BST) có thời gian thực thi phụ thuộc vào chiều cao của cây, nên trong trường hợp lý tưởng khi cây cân bằng, các phương thức insert, find, và delete\_node có thể hoạt động trong thời gian O(log n). Tuy nhiên, nếu cây không cân bằng và trở thành danh sách liên kết đơn, thời gian thực thi có thể đạt O(n) trong trường hợp xấu nhất.

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

**2.2.1. Cài đặt lớp cây tìm kiếm nhị phân**

Lớp có các thuộc tính và phương thức cơ bản cho cây tìm kiếm nhị phân.

|  |
| --- |
|  |

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  using ll = long long;  template <class T>  class node{  private:  T data;  node \*left;  node \*right;  public:  node() : left(NULL), right(NULL) {}  node \*make\_node (T x){  node \*n = new node();  n->data = x;  n->left = NULL;  n->right = NULL;  return n;  }  node \*insert (node \*root, T x){  if ( root == NULL ){  return make\_node(x);  }  if (x < root->data){  root->left = insert(root->left, x);  }  else if (x > root->data){  root->right = insert(root->right, x);  }  return root;  }  void display(node \*root){  if (root != NULL){  display(root->left);  cout << root->data << ' ';  display(root->right);  }  }  node \*min\_node (node \*root){  node \*temp = root;  while (temp != NULL && temp->left != NULL){  temp = temp->left;  }  return temp;  }  node \*delete\_node (node \*root, T x){  if (root == NULL) return;  if (x < root->data){  root->left = delete\_node(root->left, x);  }  else if (x > root->data){  root->right = delete\_node(root->right, x);  }  else {  if (root->left == NULL){  node \*temp = root->right;  delete root;  return temp;  }  if (root->right == NULL){  node \*temp = root->left;  delete root;  return temp;  }  else {  node \*temp = min\_node (root->right);  root->data = temp->data;  root->right = delete\_node(root, temp->data);  }  }  return root;  }  node\* find (node \*root, T x){  if (root == NULL) return;  node \*tmp = root;  while (tmp != NULL){  if (tmp->data == x) return tmp;  if (tmp->data > x){  tmp = tmp->left;  }  if (tmp->data < x){  tmp = tmp->right;  }  }  return nullptr;  }  bool emty(node \*root){  return root == NULL;  }  T& get\_data(){  return data;  }  }; |

Trong lớp gồm phương thức và chức năng sau:

* **Phương thức** node \*make\_node(T x)**:** Tạo một nút mới với dữ liệu x. Phương thức này sẽ trả về con trỏ đến nút mới được tạo, trong đó dữ liệu là x, và các con trỏ left và right đều trỏ đến NULL.
* **Phương thức node** \*insert(node \*root, T x): Chèn một giá trị x vào cây nhị phân tìm kiếm tại vị trí phù hợp. Nếu giá trị x nhỏ hơn dữ liệu của root, nó sẽ được chèn vào cây con bên trái; ngược lại, nếu lớn hơn, nó sẽ được chèn vào cây con bên phải.
* **Phương thức** void display(node \*root): Duyệt cây theo thứ tự giữa (in-order traversal) và in ra các giá trị của các nút từ nhỏ đến lớn, nhờ đó hiển thị cây dưới dạng danh sách sắp xếp tăng dần.
* **Phương thức node** \*min\_node(node \*root): Tìm và trả về con trỏ đến nút có giá trị nhỏ nhất trong cây. Thường là nút nằm ở phía trái nhất của cây.
* **Phương thức node** \*delete\_node(node \*root, T x): Xóa một nút có giá trị x khỏi cây. Phương thức này xử lý ba trường hợp: (1) nút không có con, (2) nút có một con, và (3) nút có hai con. Nếu nút bị xóa có hai con, phương thức sẽ tìm nút nhỏ nhất trong cây con bên phải để thay thế.
* **Phương thức** node\* find(node \*root, T x): Tìm và trả về con trỏ đến nút có giá trị x. Nếu không tìm thấy, phương thức sẽ trả về nullptr.
* **Phương thức** bool emty(node \*root): Kiểm tra xem cây có rỗng hay không bằng cách kiểm tra xem root có trỏ đến NULL hay không. Trả về true nếu cây rỗng và false nếu ngược lại.
* **Phương thức** T& get\_data(): Trả về tham chiếu đến dữ liệu của nút hiện tại, giúp truy cập hoặc sửa đổi dữ liệu của nút đó.

### 2.3. Phân tích thời gian chạy của lớp.

2.3.1. Lớp cây tìm kiếm nhị phân.

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Thời gian** | **Phân tích** |
| make\_node(T x) | O(1) | Tạo một node mới với giá trị x. Thời gian là O(1) vì chỉ cần tạo một node và gán giá trị. |
| insert(node \*root, T x) | O(h) | Thêm một node mới vào cây. Thời gian phụ thuộc vào chiều cao của cây h, vì có thể cần duyệt qua tất cả các mức của cây trong trường hợp cây không cân bằng. |
| display(node \*root) | O(n) | Duyệt cây theo thứ tự inorder và in ra giá trị của mỗi node. Thời gian O(n) vì cần duyệt qua tất cả n phần tử trong cây. |
| min\_node(node \*root) | O(h) | Tìm node có giá trị nhỏ nhất (nằm ở xa bên trái nhất của cây). Thời gian phụ thuộc vào chiều cao h vì có thể cần duyệt qua các mức bên trái của cây. |
| delete\_node(node \*root, T x) | O(h) | Xóa một node khỏi cây. Thời gian là O(h) vì có thể cần duyệt đến node chứa giá trị x theo chiều cao của cây. Nếu cần thay thế, tìm node có giá trị nhỏ nhất ở cây con bên phải. |
| find(node \*root, T x) | O(h) | Tìm node có giá trị x trong cây. Thời gian là O(h) vì duyệt cây từ gốc đến khi tìm thấy giá trị x hoặc đạt đến cuối cây theo chiều cao của nó. |
| emty(node \*root) | O(1) | Kiểm tra xem cây có rỗng không. Thời gian là O(1) vì chỉ cần kiểm tra con trỏ root. |
| get\_data() | O(1) | Lấy dữ liệu từ node. Thời gian là O(1) vì chỉ cần truy cập vào biến dữ liệu của node. |

Vì cây nhị phân tìm kiếm (BST) có thời gian thực thi phụ thuộc vào chiều cao của cây, nên trong trường hợp lý tưởng khi cây cân bằng, các phương thức insert, find, và delete\_node có thể hoạt động trong thời gian O(log n). Tuy nhiên, nếu cây không cân bằng và trở thành danh sách liên kết đơn, thời gian thực thi có thể đạt O(n) trong trường hợp xấu nhất.

## **3. Quản lý lịch công tác trong tháng.**

### 3.1. Phân tích bài toán

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

**3.1.1.1. Yêu cầu của bài toán.**

**+** Chương trình quản lý công tác trong tháng.

* + - 1. Xây dựng lớp cây tìm kiếm nhị phân (BST):
* Mỗi nút của cây là một đối tượng NgàyLàmViệc, đại diện cho một ngày cụ thể trong lịch công tác.
* Trong mỗi nút NgàyLàmViệc, lưu trữ một danh sách liên kết (linked list) chứa các công việc cần làm trong ngày đó.

2. Thông tin công việc:

* Một công việc sẽ bao gồm:
* Giờ bắt đầu (giờBắtĐầu)
* Tên công việc (tênCôngViệc)
* Nội dung công việc (nộiDungCôngViệc)
* Tính chất công việc (tínhChất) với các mức: rất quan trọng, quan trọng, bình thường, không cần thiết
* Trạng thái công việc (trạngThái): đã hoàn tất hoặc chưa thực hiện

3. Các chức năng chính:

* Nhập nội dung công việc: Cho phép người dùng thêm công việc theo ngày và giờ. Nếu ngày đã tồn tại trên cây, thêm công việc vào danh sách của ngày đó theo thứ tự tăng dần của giờ bắt đầu.
* Xem lịch công tác theo ngày: Cho phép người dùng xem công việc của một ngày cụ thể.
* Xem công việc theo tính chất: Hiển thị các công việc dựa trên tính chất (rất quan trọng, quan trọng...).
* Xem công việc đã hoàn tất/chưa thực hiện: Hiển thị công việc đã hoàn tất hoặc chưa thực hiện.
* Xem công việc từ ngày A đến ngày B: Liệt kê công việc trong khoảng từ ngày A đến ngày B.
* Xóa hoặc điều chỉnh lịch công tác: Cho phép người dùng xóa hoặc điều chỉnh công việc theo ngày. Nếu sau khi điều chỉnh, một ngày không còn công việc, ngày đó sẽ bị xóa khỏi cây BST.

### 4. Cấu trúc dữ liệu cần sử dụng:

* **BST (Binary Search Tree)**:
  + Cấu trúc chính để lưu các ngày, với mỗi ngày là một nút trên cây.
  + Mỗi nút sẽ chứa một đối tượng NgàyLàmViệc với thông tin ngày tháng.
* **Danh sách liên kết**:
  + Để lưu các công việc trong mỗi ngày. Các công việc trong danh sách được sắp xếp theo giờ bắt đầu.

**3.1.1.2. 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.**

**3.1.1.2.1. Xây dựng lớp** work.

|  |  |
| --- | --- |
| **Mô tả** | **Chức năng** |
| **Thuộc tính** |  |
| int time | Thời gian của công việc. |
| string name | Tên của công việc. |
| int priority | Mức độ ưu tiên của công việc. |
| bool done | Trạng thái hoàn thành của công việc. |
| work \*next | Con trỏ đến công việc tiếp theo trong danh sách liên kết. |
| **Phương thức** |  |
| work(string n, int t, int p, bool d) | Hàm khởi tạo, khởi tạo các thuộc tính của công việc. |

**3.1.1.2.2. Xây dựng lớp** day\_work.

|  |  |
| --- | --- |
| **Mô tả** | **Chức năng** |
| **Thuộc tính** |  |
| string day | Ngày làm việc. |
| work \*w | Con trỏ đến danh sách các công việc trong ngày. |
| day\_work \*left | Con trỏ đến ngày làm việc bên trái trong cây nhị phân. |
| day\_work \*right | Con trỏ đến ngày làm việc bên phải trong cây nhị phân. |
| **Phương thức** |  |
| day\_work(string d) | Hàm khởi tạo, khởi tạo các thuộc tính của ngày làm việc. |

**3.1.1.2.3. Xây dựng lớp** schedule.

|  |  |
| --- | --- |
| **Mô tả** | **Chức năng** |
| **Thuộc tính** |  |
| day\_work \*root | Con trỏ đến gốc của cây nhị phân lưu trữ các ngày làm việc. |
| **Phương thức** |  |
| find\_day(day\_work \*root, string &x) | Tìm kiếm và trả về con trỏ đến ngày làm việc có ngày là x. |
| addwork\_inday(day\_work\* day, string& name, int time, int priority, bool done) | Thêm công việc vào ngày làm việc day với các thông tin chi tiết. |
| viewWorkCompleted(day\_work\* node) | Hiển thị danh sách các công việc đã hoàn thành trong cây nhị phân. |
| unfinishedBusiness(day\_work\* node) | Hiển thị danh sách các công việc chưa hoàn thành trong cây nhị phân. |
| viewBetween(day\_work \*node, string& start, string& end) | Hiển thị lịch công tác trong khoảng thời gian từ start đến end. |
| minDay(day\_work\* node) | Tìm và trả về con trỏ đến ngày làm việc nhỏ nhất trong cây nhị phân. |
| deleteDay(day\_work\* node, string& date) | Xóa ngày làm việc có ngày là date khỏi cây nhị phân. |
| schedule() | Hàm khởi tạo, khởi tạo root là nullptr. |
| add\_day(string& date) | Thêm một ngày làm việc mới vào cây nhị phân. |
| addwork(string& date, int time, string name, int priority, bool done = false) | Thêm công việc vào ngày làm việc có ngày là date. |
| viewDaySchedule(const string& date) | Hiển thị lịch công tác của ngày làm việc có ngày là date. |
| viewWorkCompleted() | Hiển thị danh sách các công việc đã hoàn thành. |
| unfinishedBusiness() | Hiển thị danh sách các công việc chưa hoàn thành. |
| viewBetween(string& start, string& end) | Hiển thị danh sách công việc trong khoảng thời gian từ start đến end. |
| deleteDay(string& date) | Xóa ngày làm việc có ngày là date. |
| deleteWork(string& date, string& name) | Xóa công việc có tên là name trong ngày làm việc có ngày là date. |
| uppdateWork(string& date, string& name, int newTime = -1, int newPriority = -1, bool newdone = NULL) | Cập nhật thông tin công việc có tên là name trong ngày làm việc có ngày là |

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

#### **3.2.1. Cài đặt lớp** work**.**

Lớp gồm các thuộc tính của một công việc.

|  |
| --- |
| class work {  public:  int time;  string name;  int priority;  bool done;  work \*next;  work(string n, int t, int p, bool d): name(n), time(t), priority(p), done(d), next(nullptr) {}  }; |

Lớp gồm các thuộc tính và các phương thứ như sau:

* **Thuộc tính time** (int time): Thuộc tính này lưu trữ thời gian (có thể là thời điểm hoặc khoảng thời gian) cần thiết để thực hiện công việc. Bạn có thể quy ước đây là thời gian theo phút, giờ, hoặc một đơn vị thời gian cụ thể nào đó.
* **Thuộc tính name** (string name): Đây là chuỗi lưu tên của công việc, giúp phân biệt và mô tả từng nhiệm vụ riêng biệt.
* **Thuộc tính priority** (int priority): Đây là mức độ ưu tiên của công việc, có thể là số nguyên đại diện cho thứ tự ưu tiên (càng cao hoặc thấp tùy vào quy ước của bạn thì công việc càng quan trọng). Thuộc tính này hỗ trợ cho việc sắp xếp và ưu tiên thực hiện công việc.
* **Thuộc tính done** (bool done): Biến boolean này cho biết công việc đã hoàn thành hay chưa. Nếu done là true, công việc được xem là đã hoàn tất; ngược lại, nếu là false, công việc vẫn đang trong trạng thái cần thực hiện.
* **Thuộc tính next** (work \*next): Đây là con trỏ trỏ đến đối tượng work tiếp theo, cho phép các đối tượng work liên kết thành một danh sách liên kết. Việc sử dụng danh sách liên kết hỗ trợ tổ chức nhiều công việc vào cùng một ngày (được quản lý trong lớp day\_work).
* **Phương thức khởi tạo work(string n, int t, int p, bool d)**: Đây là hàm khởi tạo, cho phép tạo một đối tượng work mới với các giá trị cụ thể cho tên công việc (n), thời gian (t), mức độ ưu tiên (p), và trạng thái hoàn thành (d). Thuộc tính next ban đầu được khởi tạo là nullptr, vì lúc tạo mới, chưa có công việc nào đi sau công việc này.

#### **3.2.2. Cài đặt lớp** day\_work**.**

|  |
| --- |
| class day\_work {  public:  string day;  work \*w;  day\_work \*left;  day\_work \*right;  day\_work(string d): day(d), w(nullptr), left(nullptr), right(nullptr) {}  }; |

Lớp gồm các thuộc tính và phương thức như.

* **Thuộc tính day** (string day): Đây là chuỗi lưu trữ thông tin về ngày mà nhiệm vụ sẽ được thực hiện. Có thể là định dạng như "2023-11-11" hoặc bất kỳ chuỗi nào mà bạn quyết định.
* **Thuộc tính w** (work \*w): Con trỏ w trỏ đến đối tượng kiểu work, có thể được dùng để lưu trữ thông tin về công việc (chi tiết, mô tả, thời gian thực hiện, v.v.) liên quan đến ngày đó. Nếu nullptr, nghĩa là chưa có công việc nào được gán cho ngày này.
* **Thuộc tính left và right** (day\_work \*left, day\_work \*right): Đây là hai con trỏ kiểu day\_work, đại diện cho các nút con trái và phải trong cấu trúc cây tìm kiếm nhị phân. Khi cấu trúc được sắp xếp theo ngày, các thuộc tính này giúp tổ chức các ngày theo thứ tự, trong đó các nút nhỏ hơn nằm bên trái và nút lớn hơn nằm bên phải.
* **Phương thức khởi tạo** day\_work(string d): Đây là hàm khởi tạo của lớp, được sử dụng để khởi tạo đối tượng day\_work với giá trị day là chuỗi d. Ban đầu, w, left và right được đặt thành nullptr, vì chưa có công việc hay nút con nào được gán cho ngày này khi đối tượng mới được tạo.

**3.2.3. Cài đặt lớp** schedule

|  |
| --- |
| class schedule {  private:  day\_work \*root;  day\_work \*find\_day(day\_work \*root, string &x) {  if (!root) return nullptr;  if (root->day == x) return root;  else if (root->day < x) return find\_day(root->right, x);  return find\_day(root->left, x);  }  void addwork\_inday(day\_work\* day, string& name, int time, int priority, bool done) {  work\* new\_work = new work(name, time, priority, done);  if (!day->w) {  day->w = new\_work;  } else {  work\* temp = day->w;  work\* prev = nullptr;  while (temp && temp->time < time) {  prev = temp;  temp = temp->next;  }  if (prev) {  prev->next = new\_work;  } else {  day->w = new\_work;  }  new\_work->next = temp;  }  }  void viewWorkCompleted(day\_work\* node) {  if (node == nullptr) return;    viewWorkCompleted(node->left);    work\* temp = node->w;  while (temp != nullptr) {  if (temp->done) {  cout << "- Cong viec: " << temp->name  << ", Gio: " << temp->time  << ", Tinh chat: " << temp->priority  << ", Trang thai: Da hoan thanh" << endl;  }  temp = temp->next;  }    viewWorkCompleted(node->right);  }  void unfinishedBusiness(day\_work\* node){  if (node == nullptr) return;    unfinishedBusiness(node->left);    work\* temp = node->w;  while (temp != nullptr) {  if (!temp->done) {  cout << "- Cong viec: " << temp->name  << ", Gio: " << temp->time  << ", Tinh chat: " << temp->priority  << ", Trang thai: Chua hoan thanh" << endl;  }  temp = temp->next;  }    unfinishedBusiness(node->right);  }  void viewBetween (day\_work \*node, string& start, string& end){  if (!node) return;  if (node->day < start) return viewBetween (node->left, start, end);  if (node->day >= start || node->day <= end) viewDaySchedule(node->day);  if (node->day > end) return viewBetween(node->right, start, end);  }  day\_work\* minDay(day\_work\* node){  day\_work\* temp = node;  while (temp != nullptr && temp->left != nullptr) {  temp = temp->left;  }  return temp;  }  day\_work\* deleteDay(day\_work\* node, string& date){  if (!node) return node;  if (node->day < date) return deleteDay(node->left, date);  else if(node->day > date) return deleteDay(node->right, date);  else {  if (node->left == nullptr){  day\_work \*temp = node->right;  delete node;  return temp;  }  else if (node->right == nullptr){  day\_work \*temp = node->left;  delete node;  return temp;  }  else {  day\_work \*temp = minDay(node);  node->day = temp->day;  temp->right = deleteDay(temp->right, temp->day);  }  }  return node;  }  public:  schedule(): root(nullptr) {}  void add\_day(string& date) {  day\_work\* newday = new day\_work(date);  if (!root) {  root = newday;  } else {  day\_work \*tmp = root;  while (true) {  if (date < tmp->day) {  if (tmp->left) {  tmp = tmp->left;  } else {  tmp->left = newday;  break;  }  } else if (date > tmp->day) {  if (tmp->right) {  tmp = tmp->right;  } else {  tmp->right = newday;  break;  }  } else {  break;  }  }  }  }  void addwork(string& date, int time, string name, int priority, bool done = false) {  day\_work\* day = find\_day(root, date);  if (!day) {  add\_day(date);  day = find\_day(root, date);  }  addwork\_inday(day, name, time, priority, done);  }  void viewDaySchedule(const string& date) {  day\_work\* dayNode = find\_day(root, const\_cast<string&>(date));  if (dayNode) {  work\* temp = dayNode->w;  cout << "Lich cong tac ngay " << date << ":" << endl;  while (temp) {  cout << "- Cong viec: " << temp->name  << ", Gio: " << temp->time  << ", Tinh chat: " << temp->priority  << ", Trang thai: " << (temp->done ? "Da hoan thanh" : "Chua hoan thanh")  << endl;  temp = temp->next;  }  } else {  cout << "Khong co cong viec nao trong ngay nay." << endl;  }  }  void viewWorkCompleted() {  cout << "Danh sach cong viec da hoan tat:" << endl;  viewWorkCompleted(root);  }  void unfinishedBusiness() {  cout << "Danh sach cong viec chua hoan tat:" << endl;  unfinishedBusiness(root);  }  void viewBetween(string& start, string& end){  cout << "Danh sach cong viec trong khoang thoi gian la " << endl;  viewBetween(root, start, end);  }  void deleteDay(string& date){  root = deleteDay(root, date);  }  void deleteWork(string& date, string& name){  day\_work\* dayNode = find\_day(root, date);  if (!dayNode) return;  cout << "Khong tim thay ngay " << date << " trong lich cong tac" << endl;  work\* current = dayNode->w;  work\* prev = nullptr;  while (current && current->name != name){  prev = current;  current = current->next;  }  if (!prev){  dayNode->w = current->next;  } else {  prev->next = current->next;  }  delete current;  cout << "Da xoa cong viec " << name << " trong ngay " << date << endl;  if (!dayNode->w){  deleteDay(date);  }  }  void uppdateWork(string& date, string& name, int newTime = -1, int newPriority = -1, bool newdone = NULL){  day\_work\* dayNode = find\_day(root, date);  if (!dayNode){  cout << "Khong tim thay ngay " << date << " trong lich cong tac" << endl;  return;  }  work \*current = dayNode->w;  while(current && current->name == name){  current = current->next;  }  if (!current){  cout << "Khong tim thay cong viec " << name << endl;  return;  }  if (newTime != -1) current->name = newTime;  if (newPriority != -1) current->priority = newPriority;  if (newdone == NULL) current->done = newdone;  cout << "Da cap nhat cong viec " << name << " trong ngay " << date << endl;  if (newTime != -1){  work\* temp = current->next;  current->next = nullptr;  addwork\_inday(root, name, current->time, current->priority, current->done);  work \*prev = dayNode->w;  while (prev && prev->next != current){  prev = prev->next;  }  if (prev) prev->next = temp;  }  }  }; |

Lớp gồm các thuộc tính và phương thức như.

### Thuộc tính

* **root** (day\_work \*root): Con trỏ này trỏ đến gốc của cây tìm kiếm nhị phân day\_work, là nơi tổ chức các ngày và công việc liên quan. Cây này giúp bạn sắp xếp các ngày một cách tuần tự, hỗ trợ việc tìm kiếm, thêm, và xóa các ngày.

### Phương thức riêng (Private Methods)

* **find\_day**: Tìm kiếm một ngày cụ thể trong cây day\_work dựa trên chuỗi x. Phương thức này duyệt cây nhị phân để tìm đối tượng day\_work có thuộc tính day trùng với x, hoặc trả về nullptr nếu không tìm thấy.
* **addwork\_inday**: Thêm một công việc mới vào danh sách công việc của một ngày cụ thể. Công việc được sắp xếp theo thứ tự thời gian để giúp danh sách công việc trong một ngày luôn được tổ chức hợp lý. Phương thức này cũng điều chỉnh các con trỏ để duy trì liên kết danh sách.
* **viewWorkCompleted**: Duyệt qua toàn bộ cây và in ra các công việc đã hoàn thành trong từng ngày. Phương thức này đệ quy qua cây và kiểm tra từng công việc có thuộc tính done là true.
* **unfinishedBusiness**: Tương tự như viewWorkCompleted, nhưng phương thức này chỉ in ra các công việc chưa hoàn thành (done == false).
* **viewBetween**: Hiển thị các công việc trong khoảng thời gian từ ngày start đến ngày end. Phương thức này đệ quy qua các ngày trong cây nhị phân và chỉ hiển thị những ngày nằm trong khoảng yêu cầu.
* **minDay**: Trả về ngày nhỏ nhất trong cây, được sử dụng trong việc xóa nút để duy trì cấu trúc cây nhị phân cân bằng.
* **deleteDay**: Xóa một ngày khỏi cây nhị phân. Nếu ngày có công việc, phương thức sẽ giải phóng bộ nhớ của đối tượng day\_work và điều chỉnh lại các nút con.

### Phương thức công khai (Public Methods)

* **schedule**: Hàm khởi tạo mặc định, khởi tạo root là nullptr để bắt đầu với cây rỗng.
* **add\_day**: Thêm một ngày mới vào cây day\_work. Phương thức này duyệt qua cây nhị phân và tìm vị trí thích hợp để chèn ngày mới sao cho cây duy trì được thứ tự.
* **addwork**: Thêm một công việc mới vào một ngày cụ thể. Nếu ngày không tồn tại, phương thức sẽ gọi add\_day để thêm ngày mới vào cây. Sau đó, nó gọi addwork\_inday để thêm công việc vào danh sách công việc trong ngày.
* **viewDaySchedule**: Hiển thị tất cả các công việc trong một ngày cụ thể. Phương thức này tìm ngày yêu cầu và in ra danh sách các công việc cùng với thông tin chi tiết.
* **viewWorkCompleted**: Hiển thị tất cả các công việc đã hoàn thành trong toàn bộ lịch trình.
* **unfinishedBusiness**: Hiển thị tất cả các công việc chưa hoàn thành trong lịch trình.
* **viewBetween**: Hiển thị công việc trong khoảng thời gian từ start đến end.
* **deleteDay**: Xóa một ngày cụ thể cùng các công việc liên quan trong ngày đó ra khỏi cây nhị phân.
* **deleteWork**: Xóa một công việc cụ thể trong ngày. Nếu danh sách công việc trong ngày trở nên trống, nó sẽ xóa luôn ngày đó.
* **uppdateWork**: Cập nhật thông tin của một công việc cụ thể (thời gian, mức độ ưu tiên, trạng thái hoàn thành). Nếu thời gian công việc thay đổi, nó sẽ tái cấu trúc danh sách công việc trong ngày để đảm bảo thứ tự thời gian.

**3.3.4. Hàm main và các hàm tương tác người dùng**

|  |
| --- |
| **// Menu de lua chon thao tac**  **void menu() {**  **cout << "\n=== LICH CONG TAC ===" << endl;**  **cout << "1. Them cong viec" << endl;**  **cout << "2. Xem lich cong tac theo ngay" << endl;**  **cout << "3. Xem cong viec da hoan tat" << endl;**  **cout << "4. Xem cong viec chua hoan tat" << endl;**  **cout << "5. Xoa ngay" << endl;**  **cout << "6. Xoa cong viec" << endl;**  **cout << "7. Cap nhat cong viec" << endl;**  **cout << "8. Xem lich cong tac trong khoang thoi gian" << endl;**  **cout << "0. Thoat chuong trinh" << endl;**  **cout << "Nhap lua chon: ";**  **}**  **// Ham dieu khien chinh cho chuong trinh**  **void run() {**  **schedule schedule;**  **int choice;**  **do {**  **menu();**  **cin >> choice;**  **cin.ignore();**  **if (choice == 1) {**  **string date, task\_name;**  **int start\_hour, priority;**  **char progress;**  **cout << "Nhap ngay (dd-mm-yyyy): ";**  **getline(cin, date);**  **char continueAdding;**  **do {**  **cout << "Nhap ten cong viec: ";**  **getline(cin, task\_name);**  **cout << "Nhap gio bat dau: ";**  **cin >> start\_hour;**  **cout << "Nhap tinh chat cong viec: ";**  **cin >> priority;**  **cin.ignore();**  **cout << "Cong viec da hoan thanh? (y/n): ";**  **cin >> progress;**  **cin.ignore();**  **bool done = (progress == 'y' || progress == 'Y');**  **schedule.addwork(date, start\_hour, task\_name, priority, done);**  **cout << "Cong viec da duoc them vao lich cong tac." << endl;**  **cout << "Ban co muon them cong viec khac cho ngay nay khong? (y/n): ";**  **cin >> continueAdding;**  **cin.ignore();**  **} while (continueAdding == 'y' || continueAdding == 'Y');**  **} else if (choice == 2) {**  **string date;**  **cout << "Nhap ngay can xem (dd-mm-yyyy): ";**  **cin >> date;**  **schedule.viewDaySchedule(date);**  **} else if (choice == 3) {**  **schedule.viewWorkCompleted();**  **} else if (choice == 4) {**  **schedule.unfinishedBusiness();**  **} else if (choice == 5) {**  **string date;**  **cout << "Nhap ngay can xoa (dd-mm-yyyy): ";**  **cin >> date;**  **schedule.deleteDay(date);**  **cout << "Da xoa ngay " << date << " khoi lich cong tac." << endl;**  **} else if (choice == 6) {**  **string date, task\_name;**  **cout << "Nhap ngay cua cong viec can xoa (dd-mm-yyyy): ";**  **cin >> date;**  **cin.ignore();**  **cout << "Nhap ten cong viec can xoa: ";**  **getline(cin, task\_name);**  **schedule.deleteWork(date, task\_name);**  **} else if (choice == 7) {**  **string date, task\_name;**  **int new\_time = -1, new\_priority = -1;**  **char new\_progress;**  **cout << "Nhap ngay cua cong viec can cap nhat (dd-mm-yyyy): ";**  **cin >> date;**  **cin.ignore();**  **cout << "Nhap ten cong viec can cap nhat: ";**  **getline(cin, task\_name);**  **cout << "Nhap gio moi (hoac -1 neu khong doi): ";**  **cin >> new\_time;**  **cout << "Nhap tinh chat moi (hoac -1 neu khong doi): ";**  **cin >> new\_priority;**  **cin.ignore();**  **cout << "Cong viec da hoan thanh? (y/n/bo qua): ";**  **cin >> new\_progress;**  **bool new\_done = (new\_progress == 'y' || new\_progress == 'Y');**  **schedule.uppdateWork(date, task\_name, new\_time, new\_priority, new\_progress != 'b' ? new\_done : NULL);**  **} else if (choice == 8) {**  **string start, end;**  **cout << "Nhap ngay bat dau (dd-mm-yyyy): ";**  **cin >> start;**  **cout << "Nhap ngay ket thuc (dd-mm-yyyy): ";**  **cin >> end;**  **schedule.viewBetween(start, end);**  **} else if (choice == 0) {**  **cout << "Thoat chuong trinh." << endl;**  **} else {**  **cout << "Lua chon khong hop le, vui long thu lai." << endl;**  **}**  **} while (choice != 0);**  **}**  **int main() {**  **run();**  **return 0;** |

Hàm có chức năng sau.

* Hàm menu: Phụ trách hiển thị các thao tác để tương tác với người dùng.
* Hàm run: Đây là hàm điều khiển chính cho chương trình, quản lý các lựa chọn của người dùng từ menu.

### 3.3. Phân tích thời gian chạy của lớp.

#### 3.3.1. Lớp work.

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Thời gian** | **Phân tích** |
| work(string n, int t, int p, bool d) | O(1) | Hàm khởi tạo, khởi tạo các thuộc tính của đối tượng work. Thời gian thực hiện là hằng số vì chỉ có các phép gán đơn giản. |

#### 3.3.2. Lớp day\_work.

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Thời gian** | **Phân tích** |
| day\_work(string d) | O(1) | Hàm khởi tạo, khởi tạo các thuộc tính của đối tượng day\_work. Thời gian thực hiện là hằng số vì chỉ có các phép gán đơn giản. |

#### 3.3.3. Lớp schedule.

|  |  |  |
| --- | --- | --- |
| **Phương thức** | **Thời gian** | **Phân tích** |
| find\_day(day\_work \*root, string &x) | O(h) | Tìm kiếm ngày làm việc trong cây nhị phân. Thời gian thực hiện phụ thuộc vào chiều cao của cây (h). |
| addwork\_inday(day\_work\* day, string& name, int time, int priority, bool done) | O(n) | Thêm công việc vào danh sách liên kết trong ngày làm việc. Thời gian thực hiện phụ thuộc vào số lượng công việc trong ngày (n). |
| viewWorkCompleted(day\_work\* node) | O(n) | Hiển thị các công việc đã hoàn thành. Thời gian thực hiện phụ thuộc vào số lượng công việc trong cây (n). |
| unfinishedBusiness(day\_work\* node) | O(n) | Hiển thị các công việc chưa hoàn thành. Thời gian thực hiện phụ thuộc vào số lượng công việc trong cây (n). |
| viewBetween(day\_work \*node, string& start, string& end) | O(n) | Hiển thị lịch công tác trong khoảng thời gian. Thời gian thực hiện phụ thuộc vào số lượng ngày trong khoảng thời gian (n). |
| minDay(day\_work\* node) | O(h) | Tìm ngày làm việc nhỏ nhất trong cây nhị phân. Thời gian thực hiện phụ thuộc vào chiều cao của cây (h). |
| deleteDay(day\_work\* node, string& date) | O(h) | Xóa ngày làm việc khỏi cây nhị phân. Thời gian thực hiện phụ thuộc vào chiều cao của cây (h). |
| schedule() | O(1) | Hàm khởi tạo, khởi tạo root là nullptr. Thời gian thực hiện là hằng số. |
| add\_day(string& date) | O(h) | Thêm một ngày làm việc mới vào cây nhị phân. Thời gian thực hiện phụ thuộc vào chiều cao của cây (h). |
| addwork(string& date, int time, string name, int priority, bool done = false) | O(h + n) | Thêm công việc vào ngày làm việc. Thời gian thực hiện phụ thuộc vào chiều cao của cây (h) và số lượng công việc trong ngày (n). |
| viewDaySchedule(const string& date) | O(n) | Hiển thị lịch công tác của ngày làm việc. Thời gian thực hiện phụ thuộc vào số lượng công việc trong ngày (n). |
| viewWorkCompleted() | O(n) | Hiển thị danh sách các công việc đã hoàn thành. Thời gian thực hiện phụ thuộc vào số lượng công việc trong cây (n). |
| unfinishedBusiness() | O(n) | Hiển thị danh sách các công việc chưa hoàn thành. Thời gian thực hiện phụ thuộc vào số lượng công việc trong cây (n). |
| viewBetween(string& start, string& end) | O(n) | Hiển thị danh sách công việc trong khoảng thời gian. Thời gian thực hiện phụ thuộc vào số lượng ngày trong khoảng thời gian (n). |
| deleteDay(string& date) | O(h) | Xóa ngày làm việc khỏi cây nhị phân. Thời gian thực hiện phụ thuộc vào chiều cao của cây (h). |
| deleteWork(string& date, string& name) | O(h + n) | Xóa công việc khỏi ngày làm việc. Thời gian thực hiện phụ thuộc vào chiều cao của cây (h) và số lượng công việc trong ngày (n). |
| uppdateWork(string& date, string& name, int newTime = -1, int newPriority = -1, bool newdone = NULL) | O(h + n) | Cập nhật thông tin công việc. Thời gian thực hiện phụ thuộc vào chiều cao của cây (h) và số lượng công việc trong ngày (n). |

# **Tài liệu tham khảo.**

**https://viblo.asia/p/chuong-6-trees-7cay-tim-kiem-nhi-phanbinary-search-trees-bsts-W13VMgkQJY7**