

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**

**KHOA CÔNG NGHỆ THÔNG TIN**

----o0o----



**BÁO CÁO BÀI TẬP LỚN**

**CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**

**Đề tài: Quản lý sinh viên.**

**Tìm hiểu và cài đặt về cây tiền tố (TRIE)*.***

**Giảng viên hướng dẫn:**

**TS. Hoàng Văn Thông**

**Sinh viên thực hiện:**

**Khuất Đăng Khoa - 231230812**

**Lớp:**

**Công Nghệ Thông Tin 1 – K64**

**MỤC LỤC**

[GIỚI THIỆU 5](#_Toc182135511)

[Định nghĩa bài toán 5](#_Toc182135512)

[Các nhiệm vụ và mức độ hoàn thành 6](#_Toc182135513)

[Phần A: 6](#_Toc182135514)

[I. Đề bài 6](#_Toc182135515)

[II. Phân tích bài toán 7](#_Toc182135516)

[1. Phân tích và thiết kế lớp đối tượng 7](#_Toc182135517)

[Lớp SinhVien: 7](#_Toc182135518)

[Lớp DanhSachSinhVien: 8](#_Toc182135519)

[Lớp APP: 8](#_Toc182135520)

[2. Các bước thực hiện 9](#_Toc182135521)

[III. Cài đặt các lớp và hàm main bằng C++ 9](#_Toc182135522)

[Lớp SinhVien 9](#_Toc182135523)

[Lớp DanhSachSinhVien 12](#_Toc182135524)

[Lớp App 18](#_Toc182135525)

[IV. Phân tích thời gian chạy thuật toán 23](#_Toc182135526)

[1. Lớp SinhVien 23](#_Toc182135527)

[2. Lớp DanhSachSinhVien 23](#_Toc182135528)

[3. Lớp APP 25](#_Toc182135529)

[V. Tài liệu tham khảo 25](#_Toc182135530)

[Phần B: 26](#_Toc182135531)

[I. Đề bài 26](#_Toc182135532)

[II. Phân tích bài toán 26](#_Toc182135533)

[1. Phân tích và thiết kế lớp đối tượng 26](#_Toc182135534)

[ Lớp TrieNode: 27](#_Toc182135535)

[ Lớp Trie: 28](#_Toc182135536)

[ Hàm main: 29](#_Toc182135537)

[2. Các bước thực hiện 29](#_Toc182135538)

[III. Cài đặt các lớp và hàm main bằng C++ 30](#_Toc182135539)

[Lớp TrieNode 30](#_Toc182135540)

[Lớp Trie 31](#_Toc182135541)

[Hàm main 38](#_Toc182135542)

[IV. Phân tích thời gian chạy thuật toán 41](#_Toc182135543)

[LớpTrieNode: 41](#_Toc182135544)

[Lớp Trie: 41](#_Toc182135545)

[V. Tài liệu tham khảo 42](#_Toc182135546)

**DANH SÁCH HÌNH ẢNH**

[Hình 2 Minh họa cây tiền tố 26](#_Toc182136360)

# GIỚI THIỆU

# Định nghĩa bài toán

*Bài toán 1: Bài toán quản lý – Quản lý Sinh Viên*

Input: Các thao tác

* Nhập: nhập danh sách học sinh từ bàn phím hoặc lấy danh sách học sinh từ file có sẵn
* Tìm kiếm: tìm kiếm sinh viên bằng tên và mã sinh viên hoặc tìm sinh viên có điểm cao, thấp nhất
* Thêm: thêm một sinh viên hoặc nhiều sinh viên
* Sửa: sửa sinh viên bằng mã sinh viên
* Xóa: xóa sinh viên bằng mã sinh viên
* Sắp xếp: sắp xếp hồ sơ học sinh theo thứ tự tăng điểm sinh viên

Ouput:

* Save: lưu danh sách học sinh vào file
* Xuất : in ra màn hình danh sách học sinh gồm tên và xếp loại,in thông tin đầy đủ của học sinh
* Với mỗi thao tác xuất ra thông tin yêu cầu

*Bài toán 2: (Bài số 14 trong danh sách BTL)*

Input: Các thao tác:

* Thêm: thêm xâu hoặc số điện thoại vào cây tiền tố
* Tìm kiếm: tìm kiếm xem có xâu nào là tiền tố của xâu nào không hoặc có số nào là tiền tố của số còn lại không.
* Tự động hoàn thành từ: nhập xâu và tìm kiếm xem có xâu nào đã tồn tại là tiền tố của xâu đó không

Output:

* Với mỗi thao xuất ra màn hình thông tin yêu cầu

# Các nhiệm vụ và mức độ hoàn thành

|  |  |
| --- | --- |
| Nhiệm vụ | Mức độ hoàn thành |
| * Tìm hiểu lớp vector * Lên ý tưởng bài toán 1 * Triển khai và cài đặt lớp vector có các thuật toán * Áp dụng vào bài | Đã hoàn thành |
| * Tìm hiểu cây tiền tố (TRIE) * Lên ý tưởng bài toán 2 * Triển khai và cài đặt cấu trúc cây tiền tố * Áp dụng vào bài | Đã hoàn thành |
| * Làm báo cáo * Lưu trữ code bằng Github | Đã hoàn thành |

# Phần A:

## I. Đề bài

**Quản lí sinh viên bằng vector:**

* Ứng dụng lớp vector để quản lý các đối tượng là các sinh viên. Mỗi sinh viên cần quản lý các thông tin sau: mã sinh viên, tên , lớp, tuổi, điểm. Trong lớp sinh viên xây dựng toán tử so sánh , nhập xuất, …

Xây dựng lớp quản lí danh sách sinh viên có các chức năng:

* Thêm sinh viên
* Thêm danh sách sinh viên từ file
* Xuất danh sách sinh viên ra file
* Xuất danh sách
* Sắp xếp sinh viên theo điểm
* Tìm kiếm sinh viên theo mã, tên, điểm thấp nhất, cao nhất
* Thay dổi thông tin sinh viên

Xây dựng lớp APP để chạy chương trình

## II. Phân tích bài toán

### 1. Phân tích và thiết kế lớp đối tượng

Để thục hiện chức năng quản lí sinh viên, cần xây dựng 3 lớp: lớp SinhVien (chứa thông tin sinh viên), lớp DanhSachSinhVien (chứa danh sách sinh viên và các hàm chức năng), lớp App (chạy chương trình).

### Lớp SinhVien:

* + Thuộc tính:

|  |  |
| --- | --- |
| **Thuộc tính** | **Chức năng** |
| string msv  string ten  string lop  int tuoi  int diem | Mã sinh viên  Họ và tên của sinh viên  Lớp của sinh viên  Tuổi của sinh viên  Điểm của sinh viên |

* Phương thức:

|  |  |
| --- | --- |
| **Phương thức** | **Chức năng** |
| SinhVien() | Constructor (hàm khởi tạo) mặc định của lớp |
| string getMsv(),float getDiem(), ... | Hàm lấy giá trị thuộc tính |
| friend istream& operator>>() | Định nghĩa lại (nạp chồng) toán tử >> |
| friend ostream& operator<<() | Định nghĩa lại (nạp chồng) toán tử << |
| bool operator<() | Định nghĩa lại (nạp chồng) toán tử so sánh |

### Lớp DanhSachSinhVien:

* + Thành viên dữ liệu:

|  |  |
| --- | --- |
| **Thuộc tính** | **Chức năng** |
| vector<SinhVien> danhSach | Lưu trữ danh sách sinh viên |

* + Phương thức:

|  |  |
| --- | --- |
| **Phương thức** | **Chức năng** |
| void themSinhVien(const SinhVien& sv)  void themSinhVien() | Thêm sinh viên |
| void xuatDanhSach() | Xuất danh sách sinh viên |
| void xuatSinhVien(int x) | Xuất 1 sinh viên |
| void suaSinhVien(string masv) | Sửa sinh viên bằng msv |
| void xoaSinhVien(string masv) | Xóa sinh viên bằng msv |
| SinhVien maxDiem() | Tìm kiếm sinh viên có điểm cao nhất |
| SinhVien minDiem() | Tìm kiếm sinh viên có điểm thấp nhất |
| void docDanhSachTuFile(const string& tenFile) | Đọc danh sách sinh viên từ file |
| void xuatDanhSachRaFile(const string& tenFile) | Xuất danh sách sinh viên ra file |
| void sapXep() | Sắp xếp sinh viên theo điểm |
| SinhVien\* timKiemBangMsv(string masv) | Tìm kiếm sinh viên bằng msv |
| void timKiemBangTensv(string ten) | Tìm kiếm sinh viên bằng tên |

### Lớp APP:

* + Thuộc tính:
* DanhSachSinhVien dssv : Chứa danh sách sinh viên
  + Phương thức:

o Menu: Hiển thị menu chức năng để chọn chức năng cần chạy

### 2. Các bước thực hiện

* Xây dựng lớp SinhVien: Thiết kế và cài đặt các thuộc tính và phương thức cho đối tượng sinh viên.
* Xây dựng lớp DanhSachSinhVien: Xây dựng các phương thức quản lý danh sách sinh viên.
* Xây dựng lớp APP: Điều khiển luồng chương trình và xử lý menu cho người dùng.
* Kiểm thử: Chạy chương trình để kiểm tra các chức năng nhập, xuất, sắp xếp, tìm kiếm, và các thao tác khác.

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

### Lớp SinhVien

* Thuộc tính và hàm khởi tạo

class SinhVien{

private:

string msv;

string ten;

string lop;

int tuoi;

int diem;

public:

SinhVien(){

}

SinhVien(string ma,string tensv,string lopsv,int tuoisv,int diemsv){

msv=ma,ten=tensv,lop=lopsv,tuoi=tuoisv,diem=diemsv;

}

return ten;

}

* Hàm lấy thuộc tính

string getMsv(){

return msv;

}

string getTen(){

return ten;

}

float getDiem(){

return diem;

}

string getLop(){

return lop;

}

int getTuoi(){

return tuoi;

}

* Định nghĩa lại toán tử

friend istream& operator>>(istream& is,SinhVien& sv){

cout << "Nhap ma sinh vien: "; is>>sv.msv;

cout << "Nhap ten sinh vien: "; is.ignore();getline(is,sv.ten);

cout << "Nhap lop sinh vien: "; is>>sv.lop;

cout << "Nhap tuoi: ";is>>sv.tuoi;

cout << "Nhap diem: ";is>> sv.diem;

return is;

}

friend ostream& operator<<(ostream& os,const SinhVien& sv){

os << "Ma SV: " << sv.msv << "\t Ten:" << sv.ten << "\t Lop:" << sv.lop

<< "\tTuoi:" << sv.tuoi << "\tDiem:" << sv.diem<<endl;

return os;

}

bool operator<(const SinhVien& sosanh){

return diem<sosanh.diem;

}

bool operator>(const SinhVien& sosanh){

return diem>sosanh.diem;

}

};

### Lớp DanhSachSinhVien

* Thuộc tính và hàm thêm, xuất sinh viên

class DanhSachSinhVien{

private:

vector<SinhVien> danhSach;

public:

void themSinhVien(const SinhVien& sv){

danhSach.push\_back(sv);

}

void themSinhVien(){

SinhVien x;

cin>> x;

danhSach.push\_back(x);

}

void xuatDanhSach(){

cout << "Danh Sach cac sinh vien:\n";

for(const auto& sv: danhSach){

cout << sv;

}

}

void xuatSinhVien(int x){

cout << danhSach[x];

}

* Hàm sửa, xóa sinh viên

void suaSinhVien(string masv){

bool found=false;

SinhVien\* sv=timKiemBangMsv(masv);

SinhVien x;

cin >> x;

\*sv = x;

}

void xoaSinhVien(string masv){

bool found=false;

for(int i=0;i<danhSach.size();i++){

if(danhSach[i].getMsv()==masv){

found=true;

danhSach.erase(danhSach.begin()+i);

cout << "Da xoa sinh vien co msv " <<masv<<endl;

}

}

if(!found){

cout << "Khong tim thay sinh vien co msv "<<masv <<endl;

}

}

* Hàm tìm kiếm, sắp xếp sinh viên

SinhVien maxDiem(){

int vt=0;

float max=danhSach[0].getDiem();

for(int i=1;i<danhSach.size();i++){

if(max<danhSach[i].getDiem()){

max=danhSach[i].getDiem();

vt=i;

}

}

return danhSach[vt];

}

SinhVien minDiem(){

int vt=0;

float min=danhSach[0].getDiem();

for(int i=1;i<danhSach.size();i++){

if(min>danhSach[i].getDiem()){

min=danhSach[i].getDiem();

vt=i;

}

}

return danhSach[vt];

}

SinhVien\* timKiemBangMsv(string masv) {

for(auto& sv:danhSach){

if(sv.getMsv()==masv){

return &sv;

}

}

return nullptr;

}

void timKiemBangTensv(string ten){

int count=0;

cout << "Sinh vien co ten "<<ten<< " la :\n";

for(int i=0;i<danhSach.size();i++){

if(danhSach[i].getTen()==ten){

cout << danhSach[i];

count++;

}

}

if(count==0){

cout << "Khong co sinh vien nao ten "<< ten<<endl;

}

}

void sapXep(){

sort(danhSach.begin(),danhSach.end());

}

* Hàm nhập, xuất sinh viên

void docDanhSachTuFile(const string& tenFile){

ifstream file(tenFile);

if(!file.is\_open()){

cout << "Khong mo duoc file " << tenFile << endl;

return;

}

danhSach.clear();

string msv,ten,lop;

int tuoi;

float diem;

while(getline(file,msv)&&getline(file,ten)&& file >> lop >> tuoi >> diem){

file.ignore();

SinhVien sv(msv,ten,lop,tuoi,diem);

danhSach.push\_back(sv);

}

file.close();

cout << "Da doc danh sach sinh vien tu file " << tenFile <<endl;

}

void xuatDanhSachRaFile(const string& tenFile){

ofstream file(tenFile);

if (!file.is\_open()) {

cout << "Khong mo duoc file " << tenFile << endl;

return;

}

for(auto& sv: danhSach){

file << sv.getMsv() <<endl << sv.getTen() << endl << sv.getLop() << " " << sv.getTuoi() << " " << sv.getDiem() <<endl;

}

file.close();

cout << "Da xuat danh sach sinh vien ra file " << tenFile << endl;

}

### Lớp App

class App{

private:

DanhSachSinhVien dssv;

public:

void menu(){

int choice;

do{

cout << "\n--- MENU ---\n";

cout << "1. Them 1 sinh vien\n";

cout << "2. Them nhieu sinh vien\n";

cout << "3. Xuat danh sach\n";

cout << "4. Sua sinh vien\n";

cout << "5. Tim sinh vien diem cao nhat\n";

cout << "6. Tim sinh vien diem thap nhat\n";

cout << "7. Tim kiem sinh vien theo ma\n";

cout << "8. Tim kiem sinh vien theo ten\n";

cout << "9. Xoa sinh vien theo ma\n";

cout << "10. Sap xep theo diem\n";

cout << "11.Them danh sach sinh vien tu file\n";

cout << "12.Xuat danh sach sinh vien ra file\n";

cout << "0. Thoat\n";

cout << "Lua chon cua ban: ";

cin >> choice;

switch(choice){

case 1:{

dssv.themSinhVien();

break;

}

case 2: {

int n;

cout << "Nhap so luong sinh vien muon them: ";

cin >> n;

for(int i=0;i<n;i++){

cout << "Nhap sinh vien thu "<<i+1 << " :\n";

dssv.themSinhVien();

}

break;

}

case 3:{

dssv.xuatDanhSach();

break;

}

case 4:{

cout << "Nhap ma sinh vien muon sua: ";

string x;

cin >> x;

dssv.suaSinhVien(x);

break;

}

case 5:{

cout << "Sinh vien co diem cao nhat la: \n";

SinhVien a = dssv.maxDiem();

cout << a <<endl;

break;

}

case 6:{

cout << "Sinh Vien co diem thap nhat la: \n";

SinhVien b= dssv.minDiem();

cout << b <<endl;

break;

}

case 7:{

cout << "Nhap msv can tim kiem: ";

string x;

cin >> x;

SinhVien \*a = dssv.timKiemBangMsv(x);

if(a==nullptr){

cout << "Khong co sinh vien nao co msv "<< x;

}

else cout << \*a;

break;

}

case 8:{

cout << "Nhap ten sinh vien can tim kiem: ";

string x;

cin.ignore();

getline(cin,x);

dssv.timKiemBangTensv(x);

break;

}

case 9:{

string x;

cout << "Nhap msv can xoa: ";

cin >> x;

dssv.xoaSinhVien(x);

break;

}

case 10:{

dssv.sapXep();

break;

}

case 11:{

cout << "Nhap ten file nhap: ";

string x;cin >> x;

dssv.docDanhSachTuFile(x);

break;

}

case 12: {

cout << "Nhap ten file xuat: ";

string x;cin >> x;

dssv.xuatDanhSachRaFile(x);

break;

}

case 0:{

cout << "---Cam on ban da su dung chuong trinh---";

break;

}

default:

cout << "Lua chon khong hop le.\n";

}

}while(choice!=0);

}

};

* Hàm main

int main() {

App app;

app.menu();

return 0;

}

## IV. Phân tích thời gian chạy thuật toán

### 1. Lớp SinhVien

* **Constructor mặc định** và **constructor tham số**: Các constructor này chỉ thực hiện phép gán giá trị cho các thuộc tính như msv, ten, lop, v.v., nên thời gian chạy của mỗi constructor là **O(1)** vì không có vòng lặp hay thao tác phức tạp.
* **Toán tử nhập >>**: Toán tử này sử dụng getline() để nhập các chuỗi như msv, ten, lop, và thao tác cin cho các số nguyên tuoi, diem. Thời gian chạy phụ thuộc vào số lượng ký tự nhập vào trong các chuỗi, do đó trung bình có độ phức tạp là **O(n)** (với n là độ dài chuỗi nhập vào).
* **Toán tử xuất <<**: Toán tử này chỉ in các thuộc tính của đối tượng ra màn hình. Vì mỗi thao tác cout có thời gian **O(1)**, nên toàn bộ toán tử này cũng có thời gian chạy **O(1)**.
* **Toán tử so sánh >, <,** : Các toán tử này chỉ so sánh thuộc tính diem của đối tượng SinhVien, do đó mỗi toán tử có thời gian chạy **O(1)**.
* **Các phương thức truy xuất** (getMsv(), getTen(), getLop(), getDiem(), getTuoi()): Mỗi phương thức chỉ trả về một giá trị thuộc tính đơn giản, vì vậy thời gian chạy của chúng là **O(1)**.

**Tóm tắt độ phức tạp thời gian:**

* O(1): Constructor mặc định, constructor tham số, toán tử so sánh >, <, toán tử xuất <<, các phương thức truy xuất (getMsv(), getTenr(), getLop(), getTuoi(), getDiem()),
* O(n): Toán tử nhập >> (với n là độ dài chuỗi nhập vào).

### 2. Lớp DanhSachSinhVien

* **Hàm themSinhVien(const SinhVien& sv)**: Phương thức này thêm một đối tượng SinhVien vào cuối vector danhSach bằng cách sử dụng push\_back(). Thao tác thêm một đối tượng vào cuối vector có độ phức tạp trung bình là O(1). Vì vậy, thời gian chạy của phương thức này là O(1).
* **Hàm themSinhVien()**: Phương thức này bắt đầu bằng việc nhập thông tin cho một đối tượng SinhVien mới từ người dùng. Sau khi nhập xong, đối tượng SinhVien được thêm vào danhSach bằng push\_back(), thao tác có độ phức tạp trung bình là O(1). Giả sử việc nhập thông tin cho một SinhVien mất O(m), với mmm là độ dài của các chuỗi nhập vào, nên tổng thời gian chạy của phương thức là O(m).
* **Hàm xuatDanhSach()**: Phương thức này duyệt qua danhSach và in thông tin của từng sinh viên. Với nnn phần tử trong danh sách, và mỗi lần in một sinh viên mất O(1), nên tổng thời gian chạy của phương thức là O(n).
* **Hàm xuatSinhVien(int x)**: Phương thức này truy cập sinh viên ở vị trí x trong danhSach và in thông tin của sinh viên đó. Truy cập một phần tử trong vector theo chỉ số có độ phức tạp O(1). Do đó, tổng thời gian chạy của phương thức này là O(1).
* **Hàm suaSinhVien(string masv)**: Phương thức này tìm sinh viên theo masv bằng cách sử dụng hàm timKiemBangMsv. Với nnn là số lượng sinh viên, việc tìm kiếm có độ phức tạp O(n). Sau khi tìm thấy sinh viên, phương thức tiến hành sửa đổi, mất O(1). Tổng thời gian chạy của phương thức này là O(n).
* **Hàm docDanhSachTuFile(const string& tenFile)**: Phương thức này đọc từng dòng trong file và tạo ra đối tượng SinhVien, sau đó thêm vào danhSach. Với mmm là số dòng trong file và mỗi lần thêm một đối tượng SinhVien vào danhSach mất O(1), tổng thời gian chạy của phương thức là O(m).
* **Hàm xuatDanhSachRaFile(const string& tenFile)**: Phương thức này duyệt qua danhSach và ghi thông tin của từng sinh viên vào file. Với nnn phần tử trong danh sách, tổng thời gian chạy của phương thức là O(n).
* **Hàm sapXep()**: Phương thức này sử dụng hàm sort() trong STL để sắp xếp các phần tử SinhVien trong danhSach. Với nnn phần tử, sort() có độ phức tạp O(nlogn). Do đó, tổng thời gian chạy của phương thức là O(nlogn).
* **Hàm maxDiem() và minDiem()**: Hai phương thức này duyệt qua toàn bộ danh sách để tìm sinh viên có điểm cao nhất/thấp nhất. Với nnn phần tử, tổng thời gian chạy của mỗi phương thức là O(n).
* **Hàm timKiemBangMsv(string masv)**: Phương thức này duyệt qua từng sinh viên trong danhSach để tìm sinh viên có masv phù hợp. Với nnn phần tử, tổng thời gian chạy của phương thức là O(n).
* **Hàm timKiemBangTensv(string ten)**: Phương thức này duyệt qua toàn bộ danh sách để tìm sinh viên có ten khớp. Với nnn phần tử trong danh sách, tổng thời gian chạy của phương thức là O(n).
* **Hàm xoaSinhVien(string masv)**: Phương thức này tìm sinh viên theo masv và xóa sinh viên đó khỏi danhSach. Với nnn là số lượng sinh viên, việc tìm kiếm và xóa có độ phức tạp O(n).

Tóm tắt độ phức tạp thời gian của các phương thức trong lớp DanhSachSinhVien:

* **O(1)**: themSinhVien(const SinhVien& sv), xuatSinhVien(int x).
* **O(n)**: themSinhVien(), xuatDanhSach(), suaSinhVien(string masv), docDanhSachTuFile(const string& tenFile), xuatDanhSachRaFile(const string& tenFile), maxDiem(), minDiem(), timKiemBangMsv(string masv), timKiemBangTensv(string ten), xoaSinhVien(string masv).
* **O(nlogn)**: sapXep().

### 3. Lớp APP

* **menu()**: Phương thức này hiển thị các lựa chọn cho chương trình quản lý sinh viên và chạy vòng lặp để người dùng tương tác. Vì chỉ có các thao tác in chuỗi tĩnh bằng cout, thời gian chạy cho hiển thị menu là O(1).
* **Tổng thời gian chạy của menu()**: Thời gian chạy của menu() phụ thuộc vào các lựa chọn người dùng nhập vào trong vòng lặp. Mỗi lựa chọn có thời gian chạy tối đa là O(nlogn), do vậy thời gian chạy tối đa của mỗi vòng lặp menu() là O(nlogn).

## V. Tài liệu tham khảo

1. **GitHub - Student Management System**: Hệ thống quản lý sinh viên đơn giản bằng C++, với chức năng thêm, tìm kiếm và xóa sinh viên. <https://github.com/DHRUVxMISHRA/Student-Management-System>
2. **Student Management System - CodeProject**: Hướng dẫn xây ý sinh viên với các thao tác CRUD. <https://github.com/kasrababazadeh/student-management-system>.
3. **DHRUVxM​ ment-System**: Mã nguồn mở cho hệ thống quản lý sinh viên, sử dụng file để lưu trữ dữ liệu. <https://gist.github.com/susanta96/e142c377e7b5f19a221964136331ee8a>.

# Phần B:

## I. Đề bài

**Bài 14**

1. Tìm hiểu về cây tiền tố (TRIE)

2. Áp dụng cài đặt cho các bài toán sau:

a. Cho n xâu kí tự hỏi có tồn tại xâu nào là tiền tố của xâu nào không?

b. Cho n số điện thoại gồm các chữ số từ 0 đến 9 hỏi có số nào là tiền tố của các số còn lại không

c. Một ứng dụng khác của cây tiền tố do các bạn sinh viên tự đề xuất

## II. Phân tích bài toán

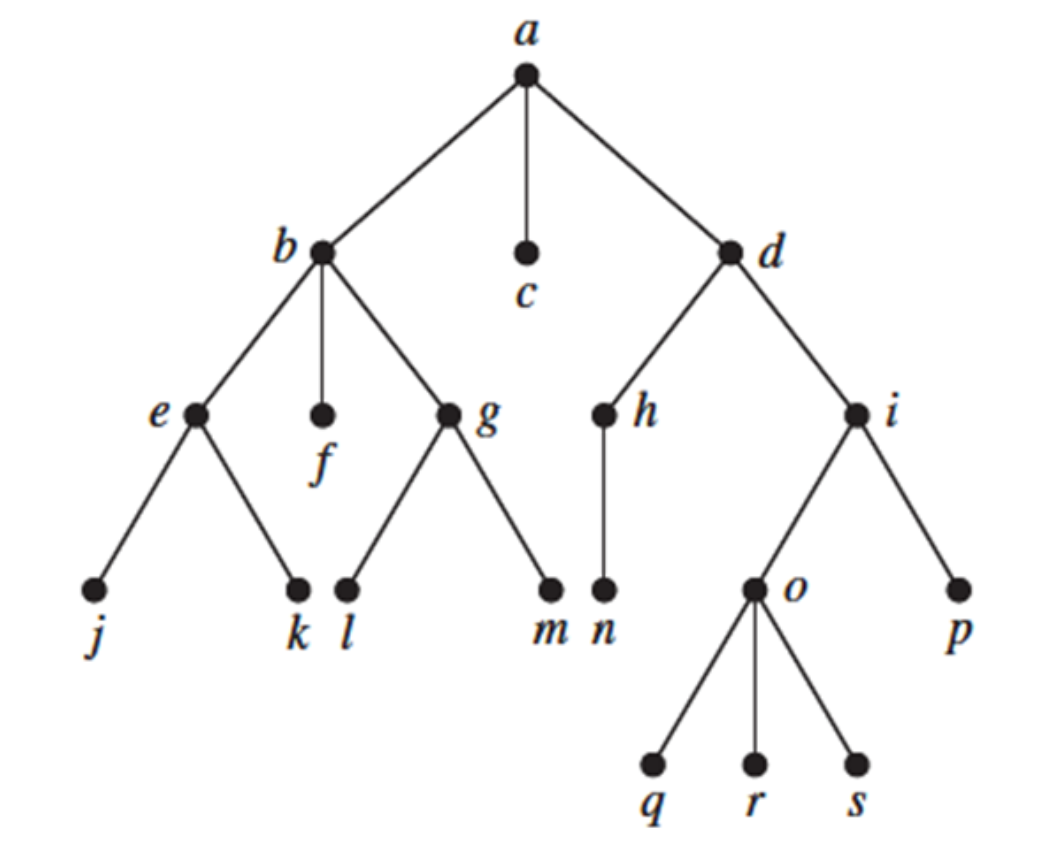
### Phân tích và thiết kế lớp đối tượng

* **CÂY TIỀN TỐ (TRIE)**

Trie, hay một số tài liệu gọi là cây tiền tố, là một cấu trúc dữ liệu dạng cây hữu dụng được dùng để quản lý một tập hợp các xâu. Mặc dù dễ hiểu và dễ cài đặt, trie lại có rất nhiều ứng dụng. Do vậy, trie thường xuyên xuất hiện trong các cuộc thi lập trình ở Việt Nam nói riêng và quốc tế nói chung.

Một trie cơ bản có thể thực hiện ba thao tác sau với độ phức tạp thời gian tuyến tính:

* Thêm một xâu vào tập hợp.
* Xóa một xâu khỏi tập hợp.
* Kiểm tra một xâu có nằm trong tập hợp đó hay không.



Hình 2 Minh họa cây tiền tố

Trong một trie, mỗi cạnh được biểu diễn bằng một ký tự, mỗi đỉnh và đường đi từ gốc đến đỉnh đó biểu diễn một xâu gồm các ký tự thuộc các cạnh trên đường đi đó.

-Để thực hiện chức năng quản lý số điện thoại và tiền tố của các số điện thoại, cần xây dựng ba lớp: lớp TrieNode, lớp Trie, và lớp App.

### Lớp TrieNode:

• **Thuộc tính:**

| **Thuộc tính** | **Chức năng** |
| --- | --- |
| TrieNode\* children[26] | Mảng con trỏ chứa các con của nút (mỗi chỉ số tương ứng với một ký tự trong bảng chữ cái) |
| bool isEndOfWord | Biến đánh dấu xem nút này có phải là kết thúc của một từ không |

• **Phương thức:**

| **Phương thức** | **Chức năng** |
| --- | --- |
| TrieNode() | Constructor, khởi tạo mảng con trỏ và đánh dấu không phải kết thúc từ |
| hasChildren() | Kiểm tra xem có bất kỳ con nào của nút hiện tại hay không |

### Lớp Trie:

• **Thuộc tính:**

| **Thuộc tính** | **Chức năng** |
| --- | --- |
| TrieNode\* root | Gốc của cây Trie, chứa các nút con của cây Trie |

• **Phương thức:**

| **Phương thức** | **Chức năng** |
| --- | --- |
| insert(const string& word) | Chèn một từ vào Trie, tạo các nút con nếu chưa có |
| insertNum(const string& phoneNumber) | Chèn một số điện thoại vào Trie, xử lý các ký tự là số từ 0-9 |
| startsWith(const string& prefix) | Kiểm tra xem có bất kỳ từ nào trong Trie bắt đầu với tiền tố cho trước |
| checkPrefixExists(const string& word) | Kiểm tra xem tiền tố có tồn tại trong Trie và có phải là tiền tố của một số khác không |
| isPrefixOfAll(const string& prefix, int totalNumbers) | Kiểm tra xem tiền tố có phải là tiền tố của tất cả các số điện thoại trong Trie hay không |
| countAllNumbers(TrieNode\* node) | Đếm số lượng số điện thoại trong cây con bắt đầu từ một nút cho trước |
| findPrefixOfAll(const vector<string>& phoneNumbers) | Kiểm tra xem có số điện thoại nào là tiền tố của tất cả các số còn lại trong danh sách không |
| searchPrefix(const string& prefix) | Tìm kiếm một tiền tố trong Trie và trả về nút cuối cùng của tiền tố |
| collectWords(TrieNode\* node, string prefix, vector<string>& results) | Thu thập tất cả các từ có tiền tố từ một nút con nhất định |
| autocomplete(const string& prefix) | Tìm và trả về tất cả các từ có tiền tố cho trước, dùng để tự động hoàn thiện từ |

### Hàm main:

* Hiển thị các phương thức cho người dùng chọn và thực hiện các hàm

### 2. Các bước thực hiện

* **Xây dựng lớp** TrieNode**:** Thiết kế và cài đặt các thuộc tính như mảng con trỏ children[26] và isEndOfWord.
* **Xây dựng lớp** Trie**:** Cài đặt các phương thức để thêm, tìm kiếm tiền tố và số điện thoại, kiểm tra các tiền tố của số điện thoại.
* **Kiểm thử:** Kiểm tra các chức năng nhập, xuất, tìm kiếm và xử lý tiền tố trong danh sách số điện thoại.

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

### Lớp TrieNode

class TrieNode {

public:

TrieNode\* children[26];

bool isEndOfWord;

TrieNode() {

isEndOfWord = false;

for (int i = 0; i < 26; i++) {

children[i] = nullptr;

}

}

bool hasChildren() {

for (int i = 0; i < 26; i++) {

if (children[i] != nullptr) {

return true;

}

}

return false;

}

};

### Lớp Trie

Thuộc tính và hàm khởi tạo

class Trie {

private:

TrieNode\* root;

public:

Trie() {

root = new TrieNode();

}

Hàm insert xâu

void insert(const string& word) {

TrieNode\* current = root;

for (char c : word) {

int index = c - 'a';

if (!current->children[index]) {

current->children[index] = new TrieNode();

}

current = current->children[index];

}

current->isEndOfWord = true;

}

Hàm insert số điện thoại

void insertNum(const string& phoneNumber) {

TrieNode\* current = root;

for (char c : phoneNumber) {

int index = c - '0';

if (!current->children[index]) {

current->children[index] = new TrieNode();

}

current = current->children[index];

}

current->isEndOfWord = true;

}

Hàm check xem có là tiền tố

bool startsWith(const string& prefix) {

TrieNode\* current = root;

for (char c : prefix) {

int index = c - 'a';

if (!current->children[index]) {

return false;

}

current = current->children[index];

}

return true;

}

Phần a, hàm check xem có xâu nào là tiền tố xâu nào không

int checkPrefixExists(const string& word) {

int check=1;

TrieNode\* current = root;

for (char c : word) {

int index = c - 'a';

if (current->children[index]==nullptr) {

check=0;

break;

}

current = current->children[index];

}

if(check==1&&!current->hasChildren()){ //check xem co trung khong

check=0;

}

return check;

}

Phần b, các hàm check xem số này có phải là tiền tố của tất cả số khác hay không

bool isPrefixOfAll(const string& prefix, int totalNumbers) {

TrieNode\* node = root;

for (char c : prefix) {

int index = c - '0';

if (node->children[index] == nullptr) {

return false;

}

node = node->children[index];

}

int count = countAllNumbers(node);

return (count == totalNumbers);

}

//tinh sdt nay la tien to cua bao nhieu sdt khac

int countAllNumbers(TrieNode\* node) {

int count;

if(node->isEndOfWord){

count=1;

}

else count=0;

for (int i = 0; i < 10; ++i) {

if (node->children[i] != nullptr) {

count += countAllNumbers(node->children[i]);

}

}

return count;

}

//ham check xem co so nao la tien to cac so con lai khong

void findPrefixOfAll(const vector<string>& phoneNumbers) {

int check=1;

int n = phoneNumbers.size();

for (const string& number : phoneNumbers) {

if (isPrefixOfAll(number, n)) {

cout << "Co !!! . Ket qua: \n" << number << endl;

check=0;

break;

}

}

if(check){

cout << "KHONG CO!!!\n";

}

}

Phần c, Ứng dụng của cây Trie do sinh viên đề xuất

-Tự động hoàn thành từ

//tim kiem prefix va tra ve node

TrieNode\* searchPrefix(const string& prefix) {

TrieNode\* node = root;

for (char c : prefix) {

int index = c - 'a';

if (node->children[index] == nullptr) {

return nullptr;

}

node = node->children[index];

}

return node;

}

// ham kiem cac ki tu va ghep vao tu

void collectWords(TrieNode\* node, string prefix, vector<string>& results) {

if (node->isEndOfWord) {

results.push\_back(prefix);

}

for (int i = 0; i < 26; ++i) {

if (node->children[i] != nullptr) {

char nextChar = 'a' + i;

collectWords(node->children[i], prefix + nextChar, results);

}

}

}

Hàm main

Trie trie;

vector<string> words;

int choice;

do{

cout << "\n---MENU---";

cout << "1.Them n xau" <<endl;

cout << "2.Them n so dien thoai" <<endl;

cout << "3.Co ton tai xau nao la tien to cua xau nao khong?" <<endl;

cout << "4.Co so nao la tien to cua cac so con lai khong?" <<endl;

cout << "5.Tu dong hoan thanh tu"<<endl;

cout << "0.Thoat"<<endl;

cout << "Lua chon cua ban: ";

cin >> choice;

// tim kiem cac tu tra ve vector string ketqua

vector<string> autocomplete(const string& prefix) {

vector<string> results;

TrieNode\* node = searchPrefix(prefix);

if (node != nullptr) {

collectWords(node, prefix, results);

}

return results;

}

};

Cin >> choice;

switch(choice){

case 1: {

cout << "Nhap so luong xau muon them: ";

int n;

cin >> n;

for (int i = 0; i < n; i++) {

string x;

cin >> x;

words.push\_back(x);

trie.insert(x);

}

break;

}

case 2:{

cout << "nhap so luong sdt can them: " << endl;

int n;cin >> n;

for (int i = 0; i < n; i++) {

string x;

cin >> x;

words.push\_back(x);

trie.insertNum(x);

}

break;

}

case 4:{

trie.findPrefixOfAll(words);

break;

}

case 5:{

string prefix;

cout << "Nhap tien to can goi y: ";

cin >> prefix;

vector<string> suggestions = trie.autocomplete(prefix);

cout << "cac goi y cho tien to la: \"" << prefix << "\":\n";

if (suggestions.empty()) {

cout << "Khong co goi y nao.\n";

} else {

for (const string& word : suggestions) {

cout << word << endl;

}

}

break;

}

default:{

cout << "Lua chon khong hop le.\n";

break;

}

}

}while(choice!=0);

}

## IV. Phân tích thời gian chạy thuật toán

### **Lớp** TrieNode:

* + **Constructor**: Mỗi lần khởi tạo một TrieNode, ta chỉ thực hiện việc gán giá trị mặc định cho các thuộc tính và khởi tạo các con trỏ. Thời gian chạy là **O(1)** vì không có vòng lặp hay phép toán phức tạp.

### **Lớp** Trie:

* + **insert(const string& word)**:
    - Hàm này duyệt qua từng ký tự của từ cần thêm và kiểm tra, chèn vào cây trie. Với độ dài từ L, thời gian chạy của insert là **O(L)**, vì mỗi ký tự được xử lý một lần.
  + **insertNum(const string& phoneNumber)**:
    - Tương tự như insert(), nhưng xử lý với chuỗi số điện thoại (các ký tự từ '0' đến '9'). Thời gian chạy vẫn là **O(L)**, trong đó L là độ dài của chuỗi số điện thoại.
  + **startsWith(const string& prefix)**:
    - Hàm này kiểm tra xem một tiền tố có tồn tại trong trie hay không. Thời gian chạy là **O(L)**, vì nó chỉ duyệt qua từng ký tự của tiền tố.
  + **checkPrefixExists(const string& word)**:
    - Tương tự như startsWith(), nhưng kiểm tra thêm việc tiền tố có phải là tiền tố của các từ khác không. Thời gian chạy của hàm này là **O(L)**.
  + **isPrefixOfAll(const string& prefix, int totalNumbers)**:
    - Hàm này kiểm tra xem tiền tố có phải là tiền tố của tất cả các số điện thoại trong trie không. Thời gian chạy bao gồm hai phần: duyệt qua tiền tố (O(L)) và đếm số điện thoại bắt đầu với tiền tố đó (O(N), với N là số lượng số điện thoại trong trie). Tổng thời gian là **O(L + N)**.
  + **countAllNumbers(TrieNode\* node)**:
    - Hàm này đếm số lượng số điện thoại bắt đầu với một tiền tố. Thời gian chạy của hàm này là **O(N)**, với N là số lượng nút con trong trie cần duyệt qua.
  + **autocomplete(const string& prefix)**:
    - Hàm này tìm tất cả các từ bắt đầu với một tiền tố. Thời gian chạy của hàm này là **O(L + W)**, trong đó L là độ dài tiền tố và W là số lượng từ tìm được trong trie.

Tóm tắt độ phức tạp thời gian:

* **O(1)**: Các thao tác đơn giản như khởi tạo đối tượng TrieNode, truy cập các thuộc tính của TrieNode và các phương thức như insertNum (với độ dài ngắn).
* **O(L)**: Các thao tác như insert, insertNum, startsWith, checkPrefixExists, và searchPrefix có độ phức tạp là **O(L)**, với L là độ dài của chuỗi (từ hoặc số điện thoại).
* **O(L + N)**: Các phương thức như isPrefixOfAll và countAllNumbers có độ phức tạp là **O(L + N)**, với L là độ dài tiền tố và N là số lượng số điện thoại trong trie.
* **O(L + W)**: Hàm autocomplete có độ phức tạp **O(L + W)**, với L là độ dài tiền tố và W là số lượng từ tìm được trong trie.

## V. Tài liệu tham khảo

Vnoi.info:: <https://vnoi.info/wiki/algo/data-structures/trie>