|  |
| --- |
| **TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI**  **KHOA ĐÀO TẠO QUỐC TẾ**    **BÀI TẬP LỚN MÔN HỌC**  **CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**  **Giảng viên hướng dẫn: TS.Hoàng Văn Thông**  **Sinh viên: Nguyễn Minh Quân**  **Lớp: CNTT VA 1 – Khóa 63**  **Hà Nội, tháng 12/2022** |

# Lời cảm ơn

Để hoàn thành đề tài này chúng em chân thành cảm ơn hãng Microsoft đã xây dựng phần mềm soạn thảo MS Word cho chúng em có môi trường soạn thảo văn bản. Phần mềm này được ứng dụng rộng rãi trên nhiều quốc gia, nâng cao kỹ thuật trình bày văn bản, dễ dàng biểu diễn, hình ảnh, sơ đồ.

Qua đây chúng em cũng chân thành cảm ơn tới Khoa CNTT – ĐHGTVT và đặc biệt là thầy Phạm Xuân Tích đã tận tình giúp đỡ chúng em thực hiện và hoàn thành đề tài nghiên cứu này.

Mục lục

[Lời cảm ơn 2](#_Toc149007100)

**Danh mục hình ảnh**

[Hình 1. Tổng quan về ngành kỹ thuật dữ liệu (Data Engineering) 5](#_Toc122438451)

[Hình 2. Sự liên kết giữa kỹ sư dữ liệu – nhà khoa học dữ liệu - kỹ sư phần mềm 7](#_Toc122438452)

[Hình 3. Các ngôn ngữ lập trình và công nghệ cần cho một kỹ sư dữ liệu 8](#_Toc122438453)

[Hình 4. Quá trình trích xuất, biến đổi và tải dữ liệu của ETL 9](#_Toc122438454)

[Hình 5. Một số công nghê cơ sở dữ liệu (Database) 9](#_Toc122438455)

Bài 1(Bài số 11 trong danh sách bài tập)

Đề bài: Để kiểm tra lỗi chính tả trong các văn bản, người ta thực hiện kiểm tra từng từ của văn bản có trong từ điển của ngôn ngữ đó hay không? Việc kiểm tra lỗi chính tả được sử dụng trong nhiều ứng dụng như các phần mềm soạn thảo văn bản.

Hãy viết một chương trình kiểm tra lỗi chính tả văn bản tiếng anh theo các yêu cầu sau:

- Nạp một từ điển ngôn ngữ tiếng Anh vào một cấu trúc thích hợp từ file

- Viết chương trình nhập vào một đoạn văn, thực hiện kiểm tra xem trong đoạn văn đó có từ

nào sai lỗi chính tả hay không? Liệt kê các từ sai lỗi chính tả.

- Ghi từ điển vào file.

1. **Các lớp, các thuộc tính, các phương thức của lớp được sử dụng trong bài toán:**
2. **Lớp dictionary:**
3. Thuộc tính:

* set<string> s: Một tập hợp lưu trữ các từ trong từ điển.

1. Phương thức:

* void readFile(const string& filename): Phương thức này đọc từ điển từ một tệp văn bản, chuyển các từ thành chữ thường sau khi loại bỏ ký tự dấu câu, và sau đó chèn các từ vào tập hợp s. Nó kiểm tra xem việc mở tệp có thành công hay không.
* void exportFile(const string filename): Phương thức này xuất từ điển ra một tệp văn bản. Nó mở tệp và ghi từng từ trong từ điển vào tệp.
* void check(): Phương thức này cho phép người dùng nhập một đoạn văn và kiểm tra xem các từ trong đoạn văn có trong từ điển hay không. Nó chuyển các từ thành chữ thường sau khi loại bỏ ký tự dấu câu, sau đó so sánh với các từ trong tập hợp s. Nếu một từ không tồn tại trong từ điển, nó thông báo rằng từ đó sai chính tả.

1. **Hàm transform:**
2. string transform(const string &word): Hàm này nhận một chuỗi word và thực hiện các bước sau:
   * Loại bỏ ký tự dấu câu ở cuối chuỗi.
   * Chuyển tất cả các ký tự trong chuỗi thành chữ thường.
   * Hàm trả về chuỗi kết quả đã được chuẩn hóa.
3. **Hàm main:**
4. int main(): Hàm main là hàm chính của chương trình.
   * Tạo một đối tượng dictionary (được đặt tên là dic).
   * Gọi phương thức readFile để đọc từ điển từ tệp "words.txt".
   * Gọi phương thức check để kiểm tra lỗi chính tả trong đoạn văn nhập từ bàn phím.
5. **Tính toán độ phức tạp của thuật toán theo big O:**
6. **Phương thức transform**:
   * Phân tích: Phương thức này duyệt qua từng ký tự trong chuỗi **word** và thực hiện một số thao tác cơ bản trên mỗi ký tự. Thời gian chạy phụ thuộc vào độ dài của chuỗi **word**.
   * Thời gian chạy: O(n), trong đó n là độ dài của chuỗi **word**.
7. **Phương thức readFile**:
   * Phân tích: Phương thức này đọc từng dòng trong tệp từ điển, thực hiện việc chuyển đổi và chèn từng từ vào tập hợp **s**. Thời gian chạy phụ thuộc vào số lượng ký tự trong tệp từ điển.
   * Thời gian chạy: O(m), trong đó m là tổng số ký tự trong tệp từ điển.
8. **Phương thức exportFile**:
   * Phân tích: Phương thức này duyệt qua tất cả các từ trong tập hợp **s** và ghi chúng vào tệp. Thời gian chạy phụ thuộc vào số từ trong tập hợp **s**.
   * Thời gian chạy: O(n), trong đó n là số từ trong từ điển.
9. **Phương thức check**:
   * Phân tích: Phương thức này nhận một đoạn văn đầu vào, chia thành từng từ, và kiểm tra xem từng từ có trong từ điển hay không. Thời gian chạy phụ thuộc vào số từ trong đoạn văn và số từ trong từ điển.
   * Thời gian chạy: O(p \* m), trong đó p là số từ trong đoạn văn đầu vào và m là số từ trong từ điển.

Tổng cộng, thời gian chạy của chương trình sẽ phụ thuộc vào phương thức có thời gian chạy lâu nhất, trong trường hợp này là phương thức **check**. Thời gian chạy của chương trình sẽ là O(p \* m), trong đó p là số từ trong đoạn văn đầu vào và m là số từ trong từ điển.

Bài 2 (Bài số 31 trong danh sách bài tập)

Đề bài:

1. Xây dựng lớp cây nhị phân.

2. Ứng dụng cây nhị phân giải các bài toán sau:

a. Cho một cây nhị phân. Hãy liệt kê tất cả các đường đi từ gốc đến lá.

b. Cho một cây nhị phân chỉ gồm các số từ 0 đến 9, mỗi đường đi từ gốc đến lá có thể biểu diễn một

số. Ví dụ: đường đi từ gốc đến lá là 1->2->3 biểu diễn số 123. Tìm tổng cộng tất cả các số từ gốc đến

lá.

Ví dụ:

Đầu vào: [1,2,3]

1

/ \

2 3

Đầu ra: 12 + 13 = 25

c. Cho một cây nhị phân và một số a, xác định xem có đường đi từ gốc đến lá nào mà tổng giá trị của

nó bằng a.

Đầu vào: Với cây nhị phân bên dưới và a = 22

5

/ \

4 8

/ / \

11 13 4

/ \ \

2 7 1

Đầu ra: True, tồn tại đường đi 5->4->11->2 mà tổng là 22

Giải thích các phương thức:

1. **Lớp Node:**

* **Thuộc tính:**
  + **int value**: Lưu trữ giá trị của nút.
  + **Node\* left**: Con trỏ tới nút con bên trái.
  + **Node\* right**: Con trỏ tới nút con bên phải.
  + **Node\* parent**: Con trỏ tới nút cha.
* **Phương thức:**
  + **Node()**: Hàm khởi tạo mặc định, tạo một nút không có giá trị và không có con.
  + **Node(int value)**: Hàm khởi tạo với giá trị ban đầu, tạo một nút có giá trị và không có con.
  + **int getValue()**: Lấy giá trị của nút.
  + **Node\* getParent()**: Lấy con trỏ tới nút cha.
  + **Node\* getLeft()**: Lấy con trỏ tới nút con bên trái.
  + **Node\* getRight()**: Lấy con trỏ tới nút con bên phải.
  + **void setValue(int value)**: Đặt giá trị của nút.
  + **void setParent(Node\* newNode)**: Đặt nút cha của nút hiện tại.
  + **void setLeft(Node\* newNode)**: Đặt nút con bên trái của nút hiện tại.
  + **void setRight(Node\* newNode)**: Đặt nút con bên phải của nút hiện tại.
  + **bool hasLeft(Node\* node)**: Kiểm tra xem nút có nút con bên trái không.
  + **bool hasRight(Node\* node)**: Kiểm tra xem nút có nút con bên phải không.

1. **Lớp BinaryTree:**

* **Thuộc tính:**
  + **Node\* root**: Con trỏ tới nút gốc của cây.
* **Phương thức:**
  + **BinaryTree()**: Hàm khởi tạo mặc định, tạo một cây rỗng.
  + **Node\* getRoot()**: Lấy con trỏ tới nút gốc của cây.
  + **Node\* insert(Node\* parent, int value)**: Chèn một nút mới có giá trị vào cây với nút cha là **parent**.
  + **void preOrder(Node\* node)**: Duyệt cây theo thứ tự trước (pre-order) và in giá trị của các nút.
  + **void rootPath(Node\* node, vector<int>& path)**: Lập danh sách các đường đi từ gốc đến lá và in chúng.
  + **void sumPath(Node\* node, vector<int>& path, int& sumRoads)**: Tính tổng của tất cả các đường đi từ gốc đến lá và in kết quả.
  + **void findPath(Node\* node, vector<int>& path, int& findRoads, int a)**: Tìm các đường đi có tổng giá trị bằng **a** và in chúng.
  + **void remove(Node\* v)**: Xóa một nút và tất cả các con của nó.

Lớp **Node** đại diện cho các nút trong cây, trong khi lớp **BinaryTree** quản lý cây nhi phân và cung cấp các phương thức để thực hiện các thao tác trên cây.