[Đồ án tin học]



**Nguyễn Thanh Hoài**

**DH52200700**

**[DH52200700@student.stu.edu.vn](mailto:DH52200700@student.stu.edu.vn)**

**D22\_TH04**

# Đề 9:

Việc cài đặt một chương trình kiểm tra lỗi chính tả cơ bản thì chỉ dựa thuần tuý trên việc kiểm tra trong một từ điển được xây dựng sẵn trên một cấu trúc dữ liệu nào đó (Tries). Tuy nhiên có những chương trình còn thực hiện một thao tác là nếu như từ đưa vào không đúng chính tả thì sẽ đưa ra một số các từ đúng chính tả, gợi ý để người dùng có thể nhanh chóng sửa lại lỗi của mình. Viết chương trình cho phép cho phép kiểm tra lỗi chính tả và gợi ý từ đúng cho người dùng.

Yêu cầu: - Ngôn ngữ: C++ hoặc C# hoặc tuỳ ý - SV phải tự xây dựng cấu trúc dữ liệu để lưu trữ dữ liệu (Nếu sử dụng ngôn ngữ C# hoặc các ngôn ngữ hướng đối tượng thì SV không được dùng cấu trúc có sẵn của ngôn ngữ)

* + Nội dung công việc TUẦN 1:
    - SV đăng ký và nhận bài tập.
    - Mỗi SV làm 1 bài.
    - SV tìm lời giải bài toán.
    - Tìm hiểu về cấu trúc dữ liệu mà SV chọn để hiện thực bài toán.
    - Sử dụng mã giả, flow chart để mô tả giải thuật.

Documentation:

* <https://www.interviewcake.com/concept/java/trie>
* [https://www.javatpoint.com/trie-data-structure-in-cpp](https://www.interviewcake.com/concept/java/trie)
* [https://the-algorithms.com/algorithm/radix-tree](https://www.interviewcake.com/concept/java/trie)
* [https://iq.opengenus.org/tries/](https://www.interviewcake.com/concept/java/trie)

# [Giới thiệu](https://www.interviewcake.com/concept/java/trie)

[Xét bài toán người dụng nhập văn bản](https://www.interviewcake.com/concept/java/trie) **[X](https://www.interviewcake.com/concept/java/trie)** [nhưng họ lại nhập không đúng chính tả. Và điều đố làm văn bản trở nên khó hiểu và gây hoang mang đối với người đọc . Câu "what are you doinh?" Hoang mang nhỉ!.](https://www.interviewcake.com/concept/java/trie)

[Giả sử chúng ta biết các từ đúng chính tả được lấy từ các từ điển. Vậy khi người dùng nhập một từ mới thì chúng ta có biết được người dùng nhập đúng hay sai? và người dùng định nhập từ gì ? Nếu có thì hàm kiểm tra và dự đoán có dạng như thế nào? cấu trúc dữ liệu nào phù hợp với việc giải quyết bài toán trên?](https://www.interviewcake.com/concept/java/trie)

# Problem data structure.

Xem xét bài toán ta nhìn thấy được rằng đầu vào của chúng ta là một chuỗi **X** và đầu ra là một chưởi tương ứng **Y** với đầu vào.

Yêu cầu của bài toán:

* Kiểm tra lỗi chính tả dựa trên một tập từ điển có sẵn.
* Đề xuất từ đúng chính tả nếu như từ đó có lỗi.

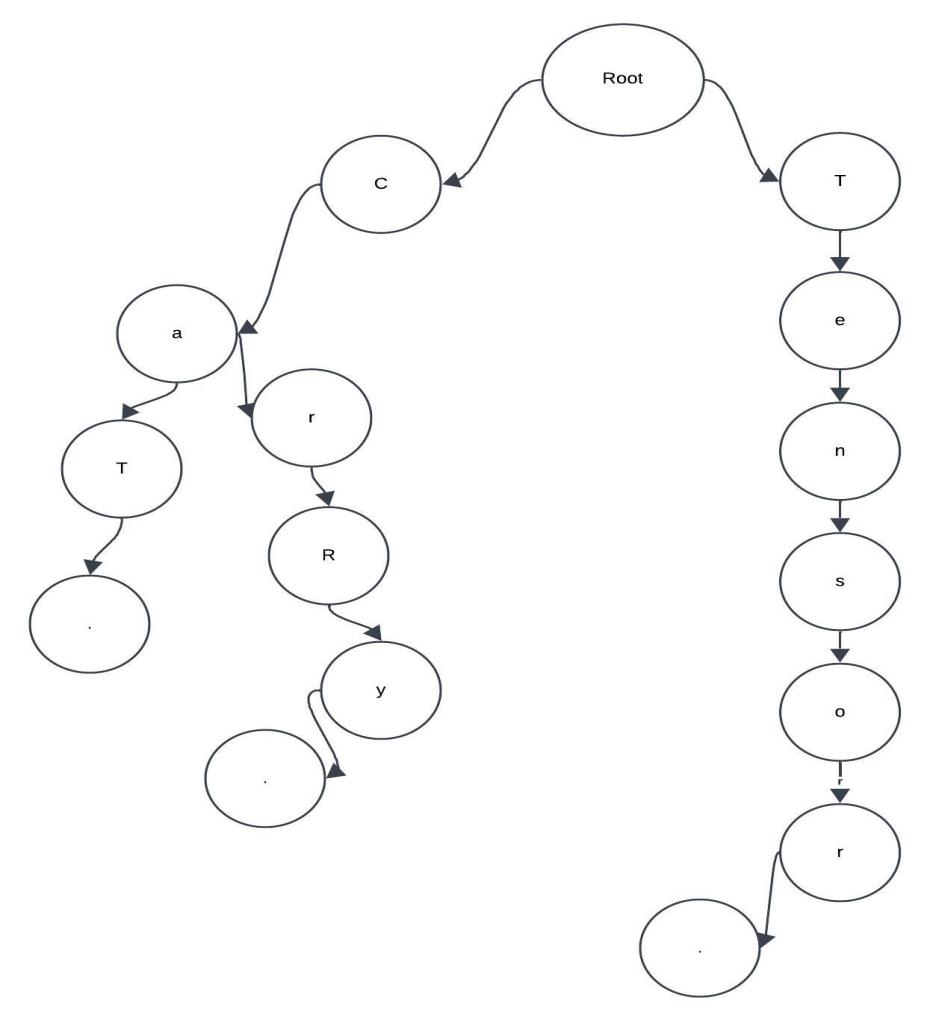
Nhận thấy ta có thể lấy mấu tử các từ điển sau đó đem các từ muốn kiểm tra vào và so sanh với các từ trong từ điển nhưng nếu một từ điển có N từ thì chung ta có độ phức tạp BigO là O(N). thời gian thực thi rất chậm. Thay vào đó chúng ta sẽ tách các từ ra thành các đơn vị nhỏ hơn và so sánh chúng từ đố tốc độ lookup, insert, delete,.. tăng mạnh.

Bằng ý tưởng trên chung ta 3 cấu trúc giải quyết bài toán:

* Sử dụng tries (cấp ký tự).
* Sử dụng sets.
* Radix trees (1 biến thể của tries) nhưng tốt hơn.

# 2.1 tries

Tries (prefix tree) : cây tiền tố là 1 kiểu của cây tìm kiếm. Dùng để lưu trữ 1 tập các chuỗi động.

**VD:** chúng ta có các từ như sau cần lưu trữ: Cat, Carry, Tensorflow.

Tries:



Biểu đồ biểu diễn tries

k là số lượng tự.

N là kích thước của mỗi node.

Xem xét cấu trúc dữ liệu tries chung ta có thể thấy rằng:

+ tries thực chất là một cây.

+ Mỗi node trên đây lưu trữ một ký tự.

+ Thuật toán để tìm kiếm một từ trên cây có độ phức tạp là O(m) với m là số lượng ký tự của từ. Tốc độ cực nhanh cho việc tiềm kiếm hoặc đề xuất.

Nhìn chung tries rất phù hợp để chọn diễn tả cho bài toán nhưng nó cũng có một số nhược điểm:

+ Các ký tự được đặt ở những địa chỉ rời rạc nhau làm cho bộ nhớ không thân thiện với bộ đêm (cache).

# 2.2 Sets

# {“cat” , “tensorflow” , “carry”}

# Ở ngôn ngữ C++:

|  |  |
| --- | --- |
| insert | O(log(n)) |
| Delete | O(log(n)) |
| Find | O(log(n)) |

Ở ngôn ngữ Python: vì được tổ chúc bằng hash table nên::

|  |  |
| --- | --- |
| insert | O(1) |
| Delete | O(1) |
| Find | O(1) |

Nhìn có vẽ rất tốt nhưng về mặt tìm kiếm nhưng vì Cấu trúc của set (xét trong c++) được lưu trong cây. Mỗi từ khác nhau được lưu một node khác nhau. Một cuốn từ điển khoảng 10 000 từ ([\*](https://www.fahasa.com/15-000-tu-tieng-anh-phai-biet.html?utm_source=google&utm_medium=cpc_pmax&utm_campaign=pmax_allproduct&utm_content=21026749153&utm_id=419704&gad_source=1&gclid=Cj0KCQjwveK4BhD4ARIsAKy6pMKfDBWRZaTM9gk22xP8t-wQY5xWg-ZlOKqxXyS2raY1X0Fj-C)) được đưa vào cây. Có thể gây tràn bộ nhớ (out of memory).

# Radix_Tree2.3 Radix\_tree:

Biểu đổ biểu diễn radix tree.

Phân tích độ phức tạp thuật toán (big 0 notation) tương tự như tries nhưng thân thiện với bộ nhớ đệm. Trong trường hợp tồi tệ nhất thì cây radix tree sẽ trở thành cây tries.

Trong 3 cấu trúc dữ liệu trên thì cấu trúc radix tree là tốt nhất vì:

- Tiết kiệm không gian.

- Cấu trúc thân thiện hơn tries vì các ký tự được sếp kề nhau.

- Truy vấn tiền tố hiệu quả:

+Có bao nhiêu từ bắt đầu bằng “ ca ” ?

+Chữ cái nào có khả năng xuất hiện khi bắt đầu bằng “ ca ”…

# IMG_256Algorithms

root

radix tree

**Nhìn vào biểu đổ trên ta nhận định được rằng:**

+ 1 từ được cho là 1 từ đúng là 1 từ đả xuất hiện trong từ điển.

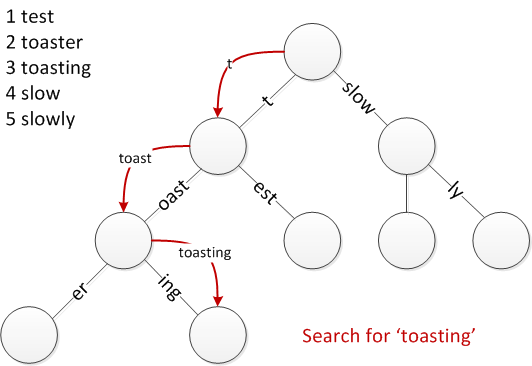
+ 1 từ là từ đúng khi có 1 đường thẳng nói từ điểm đó đi đến điểm có cờ ví dụ từ ( “ ***romane*** ”) có 1 đường nối từ ***r*** đến từ ***e* (\*)**. Vậy từ ***romane*** là 1 từ đúng chính tả.

+ Nếu 1 từ sai chính tả thì ta chỉ cần tìm 1 đường thẳng khác sao cho thoả điên khiên như **(\*).**

**+** Khi đề xuất các từ bắt đầu bằng tại node x ta chỉ cần tìm các các cây con của node x tạo thành 1 đường đi đến khi gặp các cờ tính hiệu. Tập hợp các cây con sẽ cho ra được danh sách các từ cần đề xuất.

***Các định nghĩa ban đầu:***

* Có 1 node ban đầu gọi là root. Và nó được gán là None.
* Mỗi node có cấu trúc là danh (sanh sách các địa chỉ node liền kề với nó, và boolean thể hiện đúng nếu như kết thúc từ. sai nếu như chưa kết thúc, giá trị của node hiện tại). (Có nhiều cách định nghĩa cấu trúc này.)



Giả sử chúng ta có một cấu trúc đả được xây dựng sẵn như trên: Việc tìm kiếm một từ “toasting” được thể hiện như hình phía trên:

* + Bắt đầu từ node góc có 2 nhánh là : **t** và **slow**. Trong từ được tìm kiếm có chữ **t** nên chúng ta sẽ tiếp tục tiềm kiếm bên trái của node gốc. Và cứ lần lược như vậy. Ta sẻ được một đường đi như trên.
  + Khi đề xuất bất đầu từ tiền tố đả biết vd: **toast.** Ta bắt đầu tim kiếm các node có giá trị trong có các giá trị tương ứng với câu. Nhìn trên hình có thể thấy được từ node gốc đi sang bên trái đến **t** rồi sang trái để đến **oast** bắt đầu từ node **oast** ta bắt đầu vét cạn tất cả các cây còn bắt đầu từ node đó. Nối các chuỗi theo chiều sâu của mỗi cây con ta được các từ đề xuất.

Vd: **toast** -) **toaster**, **toasting.**