ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA KHOA HỌC MÁY TÍNH

BÁO CÁO ĐỒ ÁN
NGHIÊN CỨU CẤU TRÚC DỮ LIỆU TRIE – PREFIX TREE

Sinh viên thực hiện: Cáp Kim Hải Anh

MSSV: 23520036

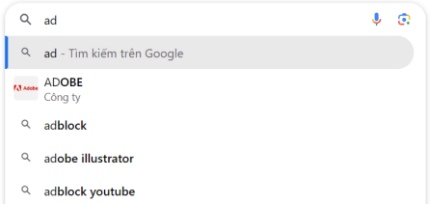
Lớp: IT003.O21.CTTN

Thành phố Hồ Chí Minh, tháng 5 năm 2024

MỤC LỤC

1. GIỚI THIỆU ĐỒ ÁN:

Đã bao giờ ta tự hỏi làm thế nào các công cụ tìm kiếm như Google có thể nhanh chóng tự động điền các đề xuất bắt đầu bằng bất kỳ nội dung nào bạn đã nhập không? Hãy xét ví dụ:



Cách tiếp cận này có thể là gì? Cách đơn giản nhất có thể là lấy văn bản mà người dùng đã nhập lần lượt các ký tự và kiểm tra xem có từ nào trong dữ liệu bắt đầu bằng chuỗi con đó không bằng cách tìm kiếm tuyến tính. Điều đó khó có thể hiệu quả trong việc xử lý nhiều truy vấn trong một cơ sở dữ liệu lớn.

Loại tương tác giống như tự động hoàn thành chuỗi ký tự này dựa vào đâu và hoạt động như thế nào? Đằng sau công cụ này, một cấu trúc dữ liệu được sử dụng có khả năng dự đoán và tự hoàn thành chuỗi là đặc điểm nổi bật của Trie (cây tiền tố).

Với mong muốn hiểu sâu về công cụ này. Đồ án được thực hiện nhằm tìm hiểu, nghiên cứu về cách tổ chức lữu trữ, hoạt động và xây dựng cấu trúc dữ liệu trie và các ứng dụng liên quan. Do hiểu biết còn hạn hẹp và chưa có kinh nghiệm trình bày nên báo cáo không tránh khỏi sai sót, rất mong nhận được sự nhận xét, đánh giá của thầy giáo để báo cáo được hoàn chỉnh hơn.

2. QUÁ TRÌNH THỰC HIỆN:

I. Tuần 1:

- Tìm hiểu, lựa chọn các CTDL/thuật toán có trong danh mục đăng ký: tìm hiểu tổng quát về các CTDL/thuật toán, có định hướng về cách trình bày, khả năng thực hiện và lựa chọn CTDL/thuật toán phù hợp.

- Hoàn thành phiếu đăng ký đồ án trên trang môn học: đặt ra những yêu cầu, nhiệm vụ, nội dung cần thực hiện (chi tiết có ở phiếu đăng ký nộp trên trang môn học).

- Xác định rõ yêu cầu về kết quả đạt được cùng như ý tưởng thực hiện, xây dựng kế hoạch thực hiện đồ án, phân chia thời gian thực hiện hợp lý.

- Nghiên cứu, tham khảo tài liệu, thực hiện các phần nội dung:

+ Đặt vấn đề – Giới thiệu về đồ án.

+ Các khái niệm cơ bản, đặc điểm, thuộc tính của CTDL; cách thức tổ chức, lưu trữ, và phương pháp hoạt động.

+ Xây dựng CTDL (định nghĩa nút, tạo nút mới, nút gốc, …): cài đặt, viết docstring.

+ Xây dựng các hàm thao tác trên CTDL (chèn, tìm kiếm, xóa, liệt kê, …), viết docstring; nghiên cứu, phân tích độ phức tạp, hiệu quả của các thao tác.

+ Tìm hiểu các kiểu dữ liệu, thư viện C++ hỗ trợ.

+ Thực hiện tạo các bộ test case cho chương trình tương ứng.

II. Tuần 2:

- Triển khai chương trình hoàn thiện: hoàn thành các hàm, docstring, đưa lên github.

- Thực hiện cài đặt chương trình bằng cách khác (cài đặt bằng mảng).

- Nghiên cứu, tham khảo tài liệu, thực hiện các phần nội dung:

+ Ứng dụng CTDL trong giải thuật và thực tế; Tìm hiểu các cấu trúc dữ liệu mở rộng, các biến thể (binary trie, radix tree), thực hiện cài đặt các hàm, chương trình, viết docstring, test case.

+ So sánh CTDL với các CTDL khác (bảng băm, cây tìm kiếm nhị phân, …): Ưu, nhược điểm, hiệu quả sử dụng.

+ Kết luận, tổng kết đề tài.

- Tự đánh giá về kết quả thực hiện đồ án: mục tiêu chung, mục tiêu cụ thể về các nhiệm vụ, nội dung đã đạt được hay không?... Từ đó, bổ sung, điều chỉnh những phần còn thiếu, sai sót: chỉnh sửa hoàn thiện theo yêu cầu mẫu báo cáo từ thầy giáo, rút gọn một số phần không cần thiết, hoàn thiện code, docstring trên github, … Hoàn thành mục Tài liệu tham khảo, Phụ lục.

- Rút ra những kiến thức từ kết quả đề tài, những kinh nghiệm sau khi thực hiện đồ án.

- Rà soát, hoàn thiện toàn bộ nội dung. Nộp đồ án trên trang môn học.

3. KẾT QUẢ ĐẠT ĐƯỢC:

I. Tổng quan về cấu trúc dữ liệu:

- Cây trie là cây có khả năng lưu trữ dữ liệu một cách trật tự và có hiệu quả. Nó còn là một loại cây tìm kiếm cụ thể, trong các nút thường được khóa bằng chuỗi.

- Trie được sử dụng nổi bật trong việc thực hiện duyệt theo thứ tự hoặc tìm kiếm hiệu quả các khóa bắt đầu bằng một tiền tố cụ thể. Đơn giản chẳng hạn như ví dụ kể trên.

Trong bài nghiên cứu này, chúng ta sẽ tìm hiểu về cấu trúc dữ liệu trie – cây tiền tố.

II. Khái niệm – Đặc điểm – Thuộc tính:

1. Khái niệm:

Cấu trúc dữ liệu **Trie** - còn được gọi là **cây tiền tố** được định nghĩa là cấu trúc dữ liệu nâng cao có dạng cây được sử dụng để lưu trữ một số tập hợp chuỗi và thực hiện các hoạt động tìm kiếm, truy xuất các cặp khóa – giá trị hiệu quả trên chúng.

2. Đặc điểm:

2.1. Đặc điểm cơ bản:

- Không giống như cây BST, không có nút nào trong cây lưu trữ khóa liên kết với nút đó; thay vào đó, vị trí của nó xác định khóa mà nó được liên kết. Tất cả các nút con của một nút đều có tiền tố chung của chuỗi được liên kết với nút đó và nút gốc được liên kết với chuỗi rỗng.

- Trie có ứng dụng rộng rãi trong đời sống hằng ngày. *(có đề cập ở phần sau)*

2.2. Đặc điểm nổi bật:

- Cấu **trúc dữ liệu Trie** được sử dụng để **lưu trữ** và **truy xuất** dữ liệu và các hoạt động tương tự có thể được thực hiện bằng cách sử dụng cấu trúc dữ liệu khác là **bảng băm,** nhưng cấu trúc dữ liệu Trie có thể thực hiện các hoạt động này hiệu quả hơn. *(có đề cập ở phần sau)*

- Tối ưu hóa độ phức tạp của tìm kiếm. Nếu tìm kiếm từ/chuỗi trong cây tìm kiếm nhị phân (BST) thì độ phức tạp về thời gian có thể lên tới O(m*logn) với ***m*** là độ dài của từ được chèn và ***n*** là tổng số từ được chèn. Tuy nhiên, khi sử dụng Trie, có thể thực hiện tìm kiếm trên một tập hợp chuỗi lớn với độ phức tạp O(n), với ***n*** là số từ trong chuỗi cần truy vấn. Thời gian tìm kiếm này có thể nhỏ hơn nếu chuỗi truy vấn không tồn tại trong trie.

3. Thuộc tính:

- Mỗi Trie có một nút gốc trống (NULL), có liên kết đến các nút khác.

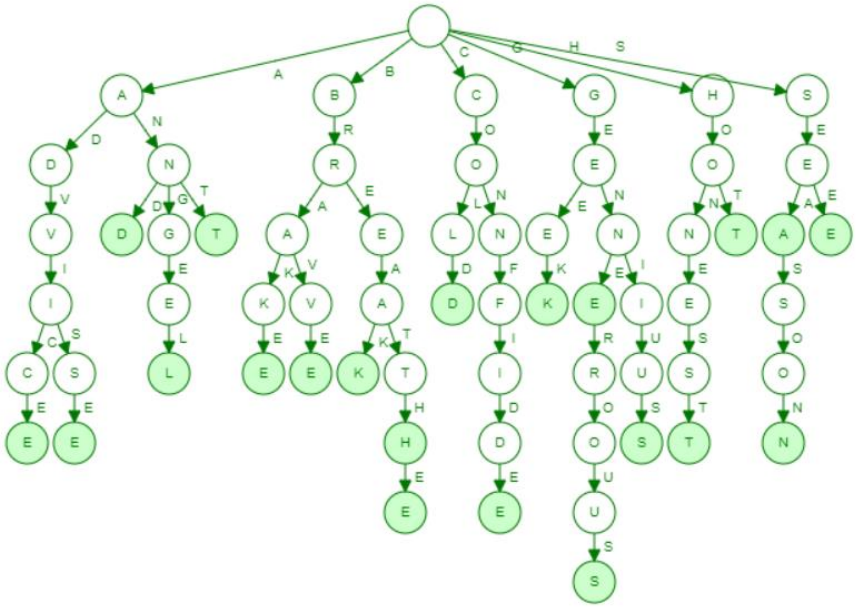
- Mỗi nút chứa một ký tự đơn và bao gồm bản đồ băm hoặc một mảng con trỏ, với mỗi chỉ mục biểu thị một ký tự và một cờ boolean để cho biết sự kết thúc của một chuỗi ở nút đó.

- Mỗi đường dẫn từ gốc đến bất kỳ nút nào đại diện cho một từ hoặc một chuỗi

III. Phương pháp, cách thức hoạt động của cấu trúc dữ liệu:

1. Ví dụ đơn giản về cấu trúc dữ liệu Trie:

Giả sử ta đang xây dựng một cơ sở dữ liệu gồm các từ ***and, angel, break, breath, gene, generous,*** .... Để tìm kiếm không mất thời gian, thay vì lưu trữ nguyên các từ này, điều sẽ làm là tạo một cây Trie. Dưới đây là một cây tiền tố trie tương ứng: **[2]**



- Với nút gốc là rỗng, mỗi nút đại diện cho một ký tự trong chuỗi. Chuỗi có thể là từ hoàn chỉnh ***(advice, see)*** hoặc là tiền tố dẫn đến một từ ***(br-*** trong ***break*** và ***breath).*** Mỗi nhánh ra khỏi một nút để tạo nút mới chứa ký tự mới biểu thị cho việc thêm một ký tự vào cuối chuỗi.

- Trong đó, tất cả các nút lá đều là nút kết thúc chuỗi, nhưng đối với những nút không phải là nút lá, cần đặt cờ boolean để thể hiện điều đó (ví dụ **"gene"** và **"generous"** cần đặt cờ boolean ở nút **"e"** để biểu thị đó là nút kết thúc chuỗi **"gene",** tương tự với các chuỗi khác)

Mẫu phân nhánh này cho phép giảm không gian tìm kiếm hiệu quả. Nếu người dùng đã nhập chuỗi con ***bra-*** thì ta sẽ không bao giờ xem xét các đường dẫn phân nhánh ***bre-*** chứ chưa nói đến tất cả các đường dẫn phân nhánh bắt đầu bằng ***b-*** trong một cơ sở dữ liệu lớn hơn.

2. Cách hoạt động của cấu trúc dữ liệu:

- Cây trie có gốc được tạo thành từ nhiều nút. Mỗi nút chứa không hoặc nhiều liên kết tương ứng với một ký tự duy nhất.

- Các lần thử được tạo thành từ nhiều nút. Các từ được lưu trữ trong trie được tìm thấy bằng cách di chuyển từ nút gốc đến bất kỳ nút nào có đánh dấu cho biết từ kết thúc tại nút đó.

- Truy tìm đường dẫn từ gốc của trie tới một nút cụ thể sẽ tạo ra tiền tố cho một từ mà chúng ta biết. Ta gọi chuỗi sau một loạt kết quả khớp gọi là kết quả khớp tiền tố.

- Có thể tìm thấy tất cả các từ được lưu trữ cho một tiền tố cụ thể một cách hiệu quả. Trên hình minh họa, khi được cho một tiền tố là **"an",** trie có thể nhanh chóng tìm và trả về tất các các từ bắt đầu bằng **"an"**, đó là **"and", "ant", "angel".** Điều này rất hữu ích cho các công cụ tìm kiếm

IV. Xây dựng cấu trúc dữ liệu:

1. Cấu trúc tổng quát của một cây trie:

- Có một nút gốc đại diện cho phần đầu trie, là nút rỗng.

- Các nút con của nút gốc này đại diện cho các chữ cái bắt đầu duy nhất của các từ/ chuỗi.

- Tiếp tục như vậy, các chữ cái thứ hai là con của các nút này, …

- Ký tự cuối cùng của mỗi từ được đánh dấu bằng cờ boolean trong nút chứa ký tự đó.

2. Hoạt động – Thao tác cơ bản:

- Chèn một chuỗi mới vào trie: **insert_word**

- Tìm kiếm một chuỗi, tiền tố có trong trie: **search_word**

- Kiểm tra tiền tố: **isPrefixExist**

- Xóa một chuỗi trong trie: **delete_word**

- Liệt kê tất cả các từ trong trie bằng một tiền tố cho trước: **list_word_byPrefix**

3. Thực hiện:

Đoạn code (có docstring mô tả phương pháp) được lưu trên github:

3.1. Triển khai nút của Trie (Trie Node): *tại đây*

3.2. Tạo một nút Trie: ***tại đây***

3.3. Giải phóng con trỏ: ***tại đây***

3.4. Tạo nút gốc của Trie (Trie Root Node): ***tại đây***

V. Các thao tác trên cấu trúc dữ liệu:

1. Chèn:

1.1. Cách hoạt động:

- Giả sử muốn lưu trữ từ **"cat"** trong trie. Bắt đầu xây dựng cây trie.

- Ban đầu trie có một nút gốc rỗng. Trước tiên ta thấy không có chuỗi nào có chung tiền tố với từ **"cat"**. Vì vậy, ta tiếp tục chèn theo từng ký tự và di chuyển xuống các nút từ ký tự đầu tiên **"c"** *(hình 1.1)* cho cuối cùng **"t"** *(hình 1.2)* rồi đánh dấu nó. *(hình 1)*

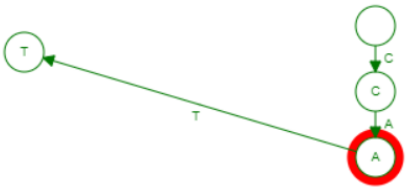
- Tiếp theo thêm từ **"do"** vào trie. Giống như từ trước, ta thực hiện chèn. *(hình 2)*

- Tiếp theo thêm từ **"card"** vào trie. Đối với từ này, các nút tương ứng với tiền tố **"ca"** đã tồn tại. Vì vậy ta bỏ qua chúng và chỉ thêm liên kết từ ký từ còn thiếu **"r"** *(hình 3.1)* đến ký tự cuối cùng **"d"** và đánh dấu ký tự cuối cùng. *(hình 3)*

Từ cách biểu diễn trên, có thể thấy rằng từ **"cat"** và từ **"card"** có một số nút chung (tiền tố **"ca"**), điều này từ thuộc tính của cấu trúc dữ liệu Trie là nếu hai chuỗi có một tiền tố chung thì chúng sẽ có cùng một tổ tiên trong trie.

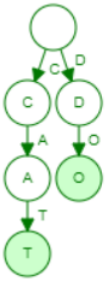
- Bây giờ thêm từ **"car"** vào trie. Đối với từ này, ta di chuyển qua các nút **c->a->r** và vì nút cho ký tự cuối cùng **"r"** *(hình 4.1)* chưa được đánh dấu nên ta đánh dấu nó. *(hình 4)*

Hình 1.1

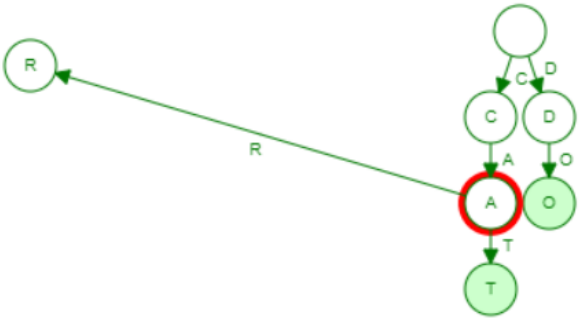


Hình 1.2

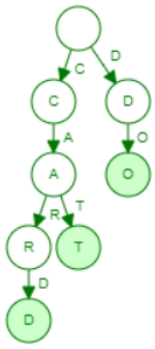
Hình 1



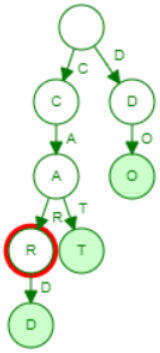
Hình 2



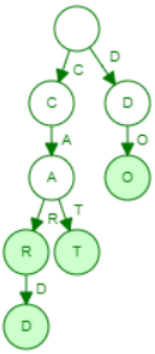
Hình 3.1



Hình 3



Hình 4.1



Hình 4

1.2. Phương pháp và triển khai chèn:

Đoạn code (có docstring mô tả phương pháp) được lưu trên github: tại đây

1.3. Độ phức tạp của thao tác chèn:

- Độ phức tạp về thời gian của thao tác chèn là O(n) với ***n*** là độ dài của từ cần chèn.

- Không gian phụ trợ: O(n*m) với ***n*** là độ dài của từ cần chèn, ***m*** là số lượng chuỗi được lưu trữ trong Trie.

2. Tìm kiếm:

Có hai cách tiếp cận tìm kiếm là tìm kiếm tiền tố và tìm kiếm từ.

2.1. Cách hoạt động: Giả sử đang có một cây trie như *hình 4.*

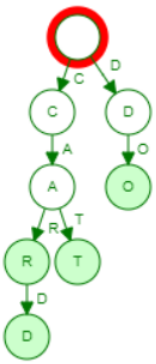
2.1.a. Tìm kiếm tiền tố:

- Tìm kiếm tiền tố **"ca"**: Bắt đầu từ nút gốc *(hình 5-6),* duyệt qua tất cả các liên kết **c->a**, đến ký tự cuối cùng trong tiền tố cần tìm kiếm vẫn tồn tại trong trie *(hình 5),* ta kết luận tìm thấy!

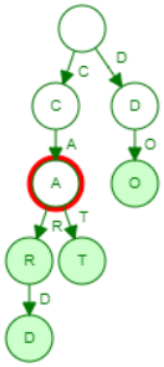
2.1.b. Tìm kiếm từ:

- Tìm kiếm từ **"card"**: Từ nút gốc *(hình 5-6)*, duyệt qua các nút **c->a->r->d**. Nút chứa ký tự cuối cùng **"d"** được đánh dấu là kết thúc từ. *(hình 6.1)* Vì vậy, ta kết luận tìm thấy từ!

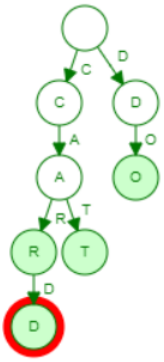
- Tìm kiếm từ **"ca"**: Từ nút gốc *(hình 5-6)*, duyệt qua các nút **c->a**. Nút chứa ký tự cuối cùng **"a"** không được đánh dấu là kết thúc từ. *(hình 6.2)* Vì vậy, ta kết luận không tìm thấy từ!



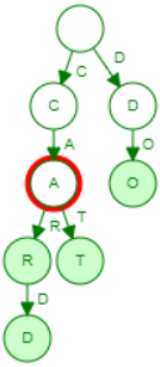
Hình 5-6



Hình 5



Hình 6.1



Hình 6.2

2.2. Phương pháp và triển khai tìm kiếm:

Đoạn code (có docstring mô tả phương pháp) được lưu trên github:

2.2.a. Tìm kiếm từ: ***tại đây***

2.2.b. Tìm kiếm tiền tố: ***tại đây***

2.3. Độ phức tạp của thao tác tìm kiếm:

- Độ phức tạp về thời gian của thao tác tìm kiếm là O(n) với ***n*** là độ dài của từ tìm kiếm.

- Không gian phụ trợ: O(1)

***Nhận xét:** Trong cả hai thao tác chèn và tìm kiếm, chúng ta không bao giờ phải duyệt cây, chẳng hạn như sử dụng tìm kiếm theo chiều sâu hoặc theo chiều rộng. Việc duyệt cây chưa bao giờ cần thiết vì đường dẫn mà ta phải đi theo đã được cung cấp trong chính đầu vào.

3. Xóa:

Có ba trường hợp khi xóa một từ khỏi Trie:

(1) Từ bị xóa là tiền tố của các từ khác trong Trie.

(2) Từ bị xóa có chung tiền tố với các từ khác trong Trie.

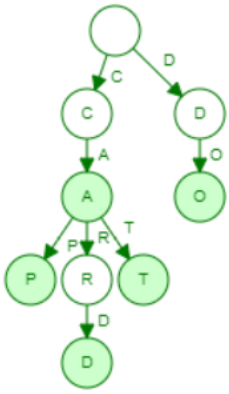
(3) Từ bị xóa không có chung tiền tố nào với các từ khác trong Trie.

3.1. Cách hoạt động: Giả sử đang có một cây trie như *hình 7.*

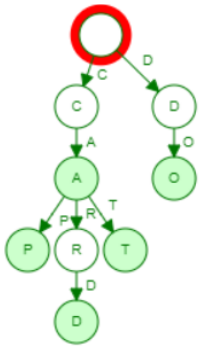
3.1.a. Từ bị xóa là tiền tố của các từ khác trong Trie:

- Giả sử từ cần xóa **"ca"** có chung một tiền tố hoàn toàn với các từ khác **"cat", "car", …**

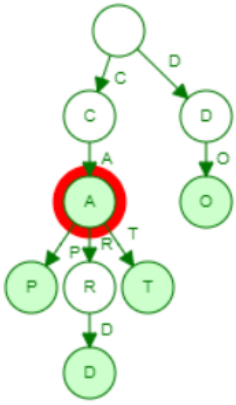
- Từ nút gốc *(hình 8-9-10)*, duyệt qua các liên kết **c->a**, đến ký tự cuối cùng **"a"** có tồn tại trong trie *(hình 8.1)*, xóa đánh dấu **is_leaf** cho biết nút đó không là kết thúc của từ **"ca"** nữa. *(hình 8)*



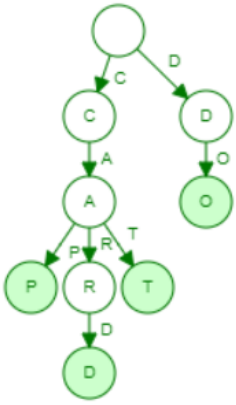
Hình 7



Hình 8-9-10



Hình 8.1



Hình 8

3.1.b. Từ bị xóa có chung tiền tố với các từ khác trong Trie.

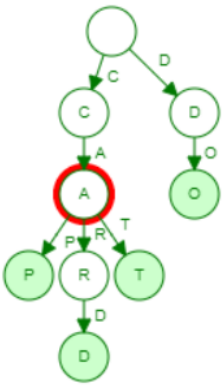
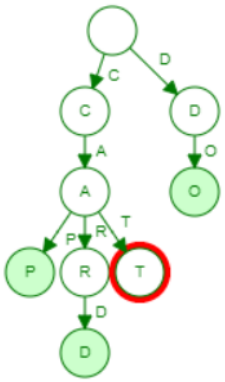
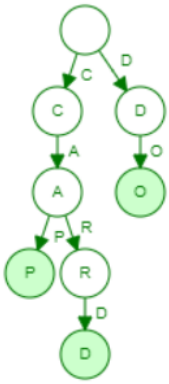
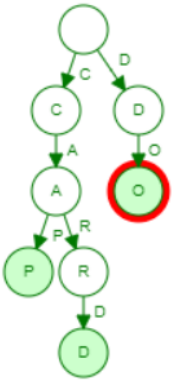
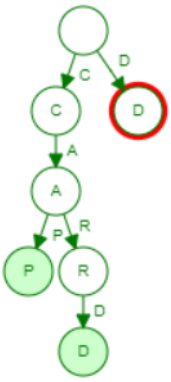
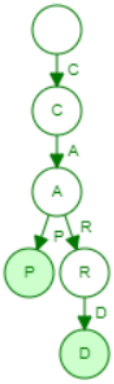
- Giả sử tiếp tục xóa từ **"cat"**, từ này có chung tiền tố **"ca"** với các từ khác **"cap"**, **"card"**.

- Từ nút gốc *(hình 8-9-10)*, duyệt qua các liên kết **c->a**, đến ký tự **"a"** có liên kết đến 3 ký tự khác **(p, r, t)** *(hình 9.1)*. Nút này là nút tiền tố chung cuối cùng giữa từ cần xóa và các từ được lưu trữ trong trie. Ta xóa từ này bằng cách cho con trỏ trỏ đến **NULL** *(hình 9.2)*. Từ đã được xóa nhưng vẫn giữ nguyên tiền tố chung của các từ khác trong trie. *(hình 9)*

3.1.c. Từ bị xóa không có chung tiền tố nào với các từ khác trong Trie.

- Giả sử xóa từ **"do"**, từ này không có tiền tố chung nào với bất kỳ từ nào khác.

- Từ nút gốc *(hình 8-9-10).* Ta tìm thấy nút tiền tố chung có giá trị **NULL**, ta chỉ thực hiện xóa tất cả các nút. *(hình 10.1 – hình 10.2 – hình 10)*

Hình 9.1 *Hình 9.2* *Hình 9* *Hình 10.1* *Hình 10.2* *Hình 10*

3.2. Phương pháp và triển khai xóa:

Đoạn code (có docstring mô tả phương pháp) được lưu trên github: tại đây

3.3. Độ phức tạp của thao tác xóa:

- Độ phức tạp về thời gian của thao tác xóa là O(n) với ***n*** là độ dài của từ cần xóa.

- Không gian phụ trợ: O(1)

4. Liệt kê tất cả các từ bắt đầu bằng một tiền tố cho trước:

Đoạn code (có dostring mô tả phương pháp) được lưu trên github: tại đây

5. In cây trie:

Đoạn code (có dostring mô tả phương pháp) được lưu trên github: tại đây

VI. Kiểu dữ liệu được dùng để cài đặt, một số thư viện C++ hỗ trợ:

- Các kiểu dữ liệu được sử dụng: kiểu kí tự **(char)**, kiểu số nguyên **(int)**, logic **(bool), …**

- Các cấu trúc dữ liệu được sử dụng: mảng con trỏ **(pointer array)**, kiểu cấu trúc **(struct)**

- Thư viện C++ hỗ trợ: **<iostream>, <fstream>, <string>, <vector>, ….**

VII. Các bài toán được áp dụng: [4], [5]

- Sắp xếp một danh sách các xâu.

- Xử lí truy vấn tiền tố chung dài nhất của hai xâu.

- Xử lí truy vấn tìm xâu có thứ tự từ điển thứ k.

- Xử lí truy vấn tìm XOR lớn nhất với giá trị được cho.

VIII. Ứng dụng trong thực tế:

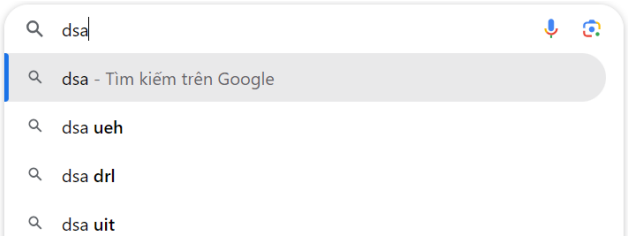
1. Tính năng tự động hoàn thành:

- Tự động hoàn thành cung cấp các đề xuất dựa trên nội dung nhập vào hộp tìm kiếm. Trie cung cấp một cách đơn giản để tìm một từ thay thế để hoàn thành từ đó vì:

+ Trie cung cấp một bộ lọc các mục theo thứ tự bảng chữ cái theo khóa của nút.

+ Chỉ theo dõi các con trỏ để lấy nút đại diện cho chuỗi được người dùng nhập vào.

+ Ngay khi bắt đầu nhập, trie sẽ cố gắng hoàn thành việc nhập liệu.



Các công cụ tìm kiếm như Google, Bing, … đều sử dụng trie nhưng với mức độ phức tạp hơn

- Một cách gần gũi hơn, khi tìm kiếm trong danh bạ, chỉ cần gõ vài ký tự đầu tiên, tất cả các số có thể sẽ được truy xuất nhờ sử dụng cấu trúc dữ liệu Trie.

2. Chức năng tra cứu:

- Được sử dụng trong các công cụ tìm kiếm để đề xuất các cụm từ bắt đầu bằng mẫu cụ thể.

- Nhanh chóng tìm kiếm từ với thời gian tỷ lệ thuận với kích thước của từ cần tìm kiếm.

- Lưu trữ từ điển dữ liệu và giúp xây dựng thuật toán tìm kiếm từ trong từ điển dễ dàng hơn, cung cấp dánh sách các từ hợp lệ cho gợi ý.

3. Kiểm tra chính tả:

- Nếu từ đã nhập không xuất hiện trong từ điển thì nó sẽ hiển thị các gợi ý dựa trên nội dung bạn đã nhập. Quá trình này gồm 3 bước: Kiểm tra từ trong từ điển dữ liệu > Tạo các từ đề xuất > Sắp xếp các đề xuất có mức độ ưu tiên cao hơn ở trên cùng.

- Hữu ích cho việc tìm kiếm tiền tố và đề xuất sửa lỗi (ví dụ: sửa **“trer”** thành **“tree”**). Đối với bảng băm là thực hiện lặp qua tất cả các từ trong O(n), rất mất thời gian so với sử dụng trie.

4. Lịch sử trình duyệt:

Được sử dụng để hoàn thành URL trong trình duyệt. Trình duyệt lưu giữ lịch sử URL của các trang web đã truy cập.

5. Ứng dụng của thuật toán khớp độ dài tiền tố dài nhất:

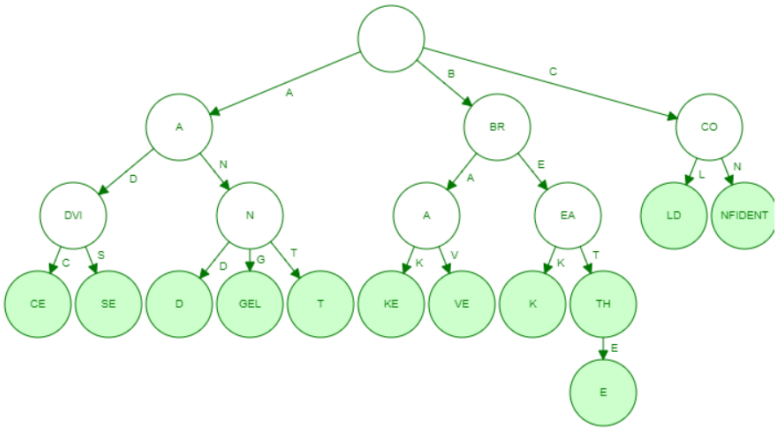
- Thuật toán này được sử dụng trong kết nối mạng bởi các thiết bị định tuyến trong mạng IP. Việc tối ưu hóa các tuyến mạng yêu cầu mặt nạ liền kề giới hạn độ phức tạp của việc tra cứu thời gian ở mức O(n), trong đó n là độ dài của địa chỉ URL tính bằng bit.

- Để tăng tốc quá trình tra cứu, các sơ đồ thử nhiều bit đã được phát triển để thực hiện tra cứu nhiều bit nhanh hơn.

IX. Cấu trúc dữ liệu mở rộng – các biến thể:

1. Cây trie cơ số (Radix Tree):

Cây **cơ số** giống như cây trie, nhưng nó **tiết kiệm không gian** bằng cách kết hợp các nút lại với nhau nếu chúng chỉ có một nút con. Dưới đây là hình minh họa cho trie cơ số **(radix tree),** cùng với bộ dữ liệu là những từ được minh họa bằng cây trie ở mục **III.1**



Nhận xét:

- Nó có ít nút và liên kết hơn phiên bản cây trie mà chúng ta đã tìm hiểu.

- Cây cơ số thân thiện với bộ đệm hơn trie vì các ký tự trong một nút thường được lưu trữ trong một mảng ký tự, liền kề nhau trong bộ nhớ. Hiệu quả hơn rất nhiều đối với các tập hợp nhỏ (đặc biệt các chuỗi dài) và các tập hợp các chuỗi có chung tiền tố dài.

Hàm thực hiện tìm kiếm từ trong cây cơ số (radix tree): **[3]**

2. Cây trie nhị phân (Binary Trie):

- CTDL có tên trie nhị phân được mở rộng từ trie được dùng để xử lí bài toán liên quan tới thao tác bit. Một trie nhị phân bao gồm các cạnh là bit 0/1 và các đỉnh là các số nguyên gồm các bit trên đường đi từ gốc đến nó.

- Các số được thêm vào sẽ được chuyển thành dạng nhị phân rồi thêm các bit 0 vào đầu sao cho độ dài các số nhị phân đều bằng nhau. Thông thường độ dài này sẽ là $\mathbf{log_2(max\ a_i)}$ với $\mathbf{a_i}$ là các số trong danh sách. Khi thêm vào trie, ta sẽ thêm các bit vào trie theo chiều từ trái sang phải.

Đoạn code về binary trie và các thao tác cơ bản trên nó được lưu trên github: ***tại đây***

X. So sánh với các cấu trúc dữ liệu khác:

1. Ưu điểm:

- Trie cho phép nhập, tìm và xóa các chuỗi trong thời gian O(n), trong đó ***n*** là độ dài của một từ. Nhanh hơn so với cả bảng băm và cây tìm kiếm nhị phân.

- Cung cấp khả năng lọc các mục theo thứ tự bảng chữ cái theo khóa của nút, do đó giúp in tất cả các từ theo thứ tự bảng chữ cái dễ dàng hơn, điều này khó để thực hiện bằng bảng băm.

- Trie chiếm ít không gian hơn khi so sánh với BST vì các khóa không được lưu rõ ràng mà thay vào đó, mỗi khóa chỉ yêu cầu một lượng không gian cố định được khấu hao để lưu trữ.

- Tìm kiếm tiền tố/so khớp tiền tố dài nhất có thể được thực hiện một cách hiệu quả. Các lượt thử có thể nhanh chóng trả lời các truy vấn về các từ có tiền tố chung, như: có bao nhiêu từ bắt đầu bằng **"choc",** chữ cái tiếp theo có khả năng xuất hiện trong một từ bắt đầu bằng **"strawb"**, …

- Triển khai nhanh hơn bảng băm đối với số nguyên và con trỏ vì Trie không cần bất kỳ hàm băm và chi phí nào để triển khai.

- Trie hỗ trợ dễ dàng phép lặp theo thứ tự trong khi phép lặp trong bảng băm sẽ dẫn đến thứ thứ tự giả ngẫu nhiên do hàm băm đưa ra, thường cồng kềnh hơn.

2. Nhược điểm:

- Nhược điểm chính của Trie là cần nhiều bộ nhớ để lưu trữ tất cả các chuỗi. Đối với mỗi nút, có quá nhiều con trỏ nút bằng số lượng ký tự trong trường hợp xấu nhất. Việc tìm kiếm hiếm khi tiết kiệm dung lượng khi so sánh với việc lưu trữ chuỗi trong **set.**

- Nếu có một bảng băm được xây dựng hiệu quả (có nghĩa là hàm băm tốt và hệ số tải hợp lý) thì có thời gian tra cứu là O(1), nhanh hơn nhiều so với O(n) trong khi sử dụng trie.

- Hầu hết các ngôn ngữ không đi kèm với việc triển khai trie thích hợp.

XI. Kết luận – Tổng kết:

Trong bài nghiên cứu này, chúng ta đã hình thành tổng quát về cây trie và biết cách triển khai một cây tiền tố. Cụ thể là các thao tác chèn, tìm kiếm cơ bản cũng như chức năng tìm kiếm tiền tố. Ngoài ra, biết được nhiều ứng dụng của trie trong đời sống thực tế.

4. TÀI LIỆU THAM KHẢO:

[1] GeeksforGeeks. [Online] 4 17, 2024. [Cited: 4 27, 2024.] https://www.geeksforgeeks.org/introduction-to-trie-data-structure-and-algorithm-tutorials/.

[2] Trie Visualization. [Online] University of San Francisco. [Cited: 5 12, 2024.] https://www.cs.usfca.edu/~galles/visualization/Trie.html

[3] Wikipedia. [Online] 4 27, 2024. [Cited: 5 4, 2024.]. https://en.wikipedia.org/wiki/Radix_tree

[4] Quang, Ngô Nhất. VNOI Wiki. *Trie.* [Online] 2020. [Cited: 4 27, 2024.] https://vnoi.info/wiki/algo/data-structures/trie.md#%E1%BB%A9ng-d%E1%BB%A5ng

[5] Quang, Ngô Nhất. VNOI Wiki. *Trie.* [Online] 2020. [Cited: 4 27, 2024.] https://vnoi.info/wiki/algo/data-structures/trie.md#%C3%A1p-d%E1%BB%A5ng

[6] Baeldung. CS Tries (Prefix Trees). [Online] 2020. [Cited: 4 27, 2024.] https://www.baeldung.com/cs/tries-prefix-trees

5. PHỤ LỤC:

***Chương trình/Hàm (có docstring) được lưu trên github: tại đây.* Chi tiết:**

5.1. Xây dựng cấu trúc dữ liệu Trie:

5.1.a. Triển khai nút của Trie (Trie Node): *tại đây*

5.1.b. Tạo một nút Trie: ***tại đây***

5.1.c. Giải phóng con trỏ: ***tại đây***

5.1.d. Tạo nút gốc của Trie (Trie Root Node): ***tại đây***

5.2. Các thao tác cơ bản trên Trie: ***(có docstring mô tả phương pháp)***

5.2.a. Chèn: ***tại đây***

`5.2.b. Tìm kiếm từ: ***tại đây***

5.2.c. Tìm kiếm tiền tố: ***tại đây***

5.2.d. Xóa: ***tại đây***

5.2.e. Liệt kê: ***tại đây***

5.2.f. In cây trie: ***tại đây***

5.3. Chương trình hoàn thiện:

- Chương trình cài đặt (bằng con trỏ) chạy sẵn: ***tại đây***

- Chương trình cài đặt (bằng con trỏ) có test case: ***tại đây***

Dữ liệu vào: Đọc từ file **"input.txt"** có cấu trúc như sau:

+ Dòng đầu tiên chứa 4 số nguyên *len1, len2, len3, len4* lần lượt là số lượng từ cần chèn vào trie, số lượng từ truy vấn tìm kiếm, số lượng từ truy vấn xóa, số lượng tiền tố cho trước.

+ *len1* dòng tiếp theo chứa các từ cần chèn, *len2* dòng tiếp theo chứa các từ cần tìm kiếm, *len3* dòng tiếp theo chứa các từ cần xóa, *len4* dòng tiếp theo chứa các tiền tố cho trước

Dữ liệu ra: Ghi ra file **"output.txt"** có cấu trúc như sau:

+ Dòng đầu tiên in cây trie sau khi thực hiện thao tác chèn

+ *len2* dòng tiếp theo là kết quả tìm kiếm các từ, *len3* dòng tiếp theo là kết quả xóa các từ – in cây trie sau khi xóa, *len4* dòng tiếp theo là các từ được liệt kê bằng các tiền tố cho trước.

- Chương trình cài đặt (bằng mảng) (có mô tả input/ouput): ***tại đây***

5.4. Cấu trúc dữ liệu mở rộng – biến thể:

- Chương trình cài đặt cây trie nhị phân (có mô tả input/ouput): ***tại đây***

5.5. Chương trình giải một số bài toán được áp dụng bằng trie: ***tại đây***