ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

[NHÓM K]

Nguyễn Đình Văn – 20127662

Phạm Trần Minh Ngọc – 20127403

Trần Hoàng Minh Quang – 20127299

Lưu Minh Phát – 20127061

BÁO CÁO

[GIÁO VIÊN HƯỚNG DẪN]

TS. Nguyễn Thanh Phương

TS. Nguyễn Ngọc Thảo

ThS. Bùi Huy Thông

DATA STRUCTURE & ALGORITHMS

Thành phố Hồ Chí Minh – 2021

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

NHÓM K

BÁO CÁO

[Topic 16]

ADVANCED TREE 1 – TRIE

DATA STRUCTURE & ALGORITHMS

Thành phố Hồ Chí Minh – 2021

LỜI CẢM ƠN

Chúng em xin gửi lời cảm ơn chân thành đến quý thầy, cô giáo và trường Đại Học Khoa Học Tự Nhiên đã luôn tạo điều kiện tốt nhất để chúng em có thể hoàn thành đồ án một cách tốt đẹp. Mặc dù có nhiều khó khăn do đại dịch Covid-19 nhưng thầy cô vẫn luôn quan tâm, hướng dẫn và tạo thuận lợi cho chúng em bằng nhiều cách. Qua đồ án lần này, chúng em đã có thể nắm vững được cách vận hành, độ phức tạp và ứng dụng thực tiễn của cấu trúc dữ liệu Trie. Hơn nữa, còn rèn luyện cho bản thân kĩ năng làm việc nhóm, kĩ năng phân chia công việc và quản lí thời gian tốt hơn.

Bài báo cáo đồ án thực hiện trong khoảng thời gian gần 4 tuần. Trong suốt quá trình thực hiện đồ án, do vẫn còn nhiều hạn chế về kinh nghiệm nên chúng em không thể nào tránh khỏi những sai sót. Chúng em rất mong nhận được những góp ý, chỉ bảo để có thêm kinh nghiệm và cũng mong thầy cô bỏ qua.

Chúng em xin chân thành cảm ơn!

MỤC LỤC

DANH MỤC HÌNH ẢNH

DANH MỤC BẢNG

GIỚI THIỆU

Giáo viên hướng dẫn môn Data structure & Algorithms

Giáo viên lý thuyết: TS. Nguyễn Thanh Phương

Giáo viên thực hành:

- TS. Nguyễn Ngọc Thảo
- ThS. Bùi Huy Thông

Group ID: K

Thành viên:

- Nhóm trưởng: Nguyễn Đình Văn – 20127662
- Phạm Trần Minh Ngọc – 20127403
- Trần Hoàng Minh Quang – 20127299
- Lưu Minh Phát – 20127061

Phân chia công việc:

- Phần Coding: Nguyễn Đình Văn, Phạm Trần Minh Ngọc.
- Phần Report: Lưu Minh Phát, Trần Hoàng Minh Quang.

YÊU CẦU ĐỀ TÀI NGHIÊN CỨU

Về phần Mã nguồn, thuật toán (Source code):

Đảm bảm các yêu cầu cơ bản của một cấu trúc cây (Tree data structure)

- Kiểm tra xem một Trie có rỗng
- Lấy các đối tượng trong một Trie
- Tìm kiếm đối tượng trong một Trie
- Thêm/chèn một đối tượng vào trong Trie
- Xóa một đối tượng khỏi Trie
- Xây dựng một Trie từ các đối tượng được cho
- Xóa hết tất cả các đối tượng của một Trie

Xem xét từ điển nhó cái từ tiếng Anh từ tập tin theo định dạng sau

- Dòng đầu: một só dương N, thể hiện số từ của từ điển
- N dòng tiếp theo: mỗi dòng là một từ tiếng Anh

Xây dựng một Trie dựa vào từ điển trên

Không có giới hạn trong việc tổ chức thuật toán. Trong hàm chính (main), thể hiện code để xây dựng Trie với từ điển được cho và một vài ví dụ trong việc tra cứu các từ trong Trie

Về phần Báo cáo (Report)

Xét một Trie rỗng của các chuỗi. Hãy thể hiện từng bước, cách mà Trie thay đổi trong khi trải qua việc chèn và xóa đối tượng.

Báo cáo độ phức tạp thời gian với mỗi hoạt động trên

Nhận biết được ứng dụng của Trie trog thực tế

NỘI DUNG ĐỀ TÀI NGHIÊN CỨU – TRIE

PHẦN I. GIỚI THIỆU ADVANCED TREE 1 – TRIE

Khi học môn Cấu trúc dữ liệu và Giải thuật, chúng ta đã biết về cấu trúc cây nhị phân (BST – Binary Search Tree) và một số cấu trúc cây khác. Nhưng tất cả các cây tìm kiếm được sử dụng để tập hợp các giá trị số nhưng chúng không phù hợp để lưu trữ tập hợp các từ hoặc chuỗi. Ở đề tài nghiên cứu này sẽ được đề cập đến Trie (Cây tiền tố) (một cấu trúc cây nâng cao (advanced tree)).

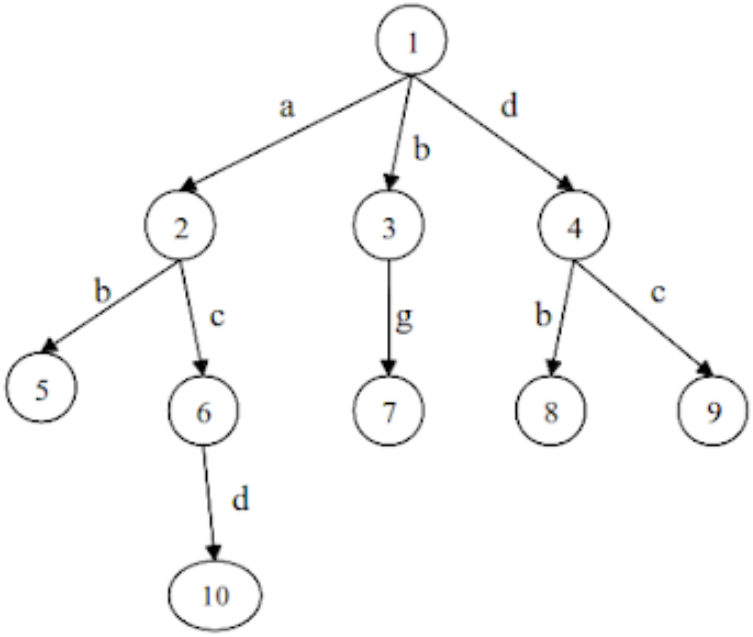
Trong khoa học máy tính, Trie hay cây tiền tố (digital tree or prefix tree), là một cấu dữ liệu cây có thứ tự, được sử dụng để xác định vị trí khóa (key) từ trong một tập hợp. Các khóa (key) này thường là các chuỗi ký tự, mà liên kết giữa các nút (nodes) không được xác định bởi toàn bộ khóa mà bởi các ký tự riêng biệt. Để truy cập vào một khóa (key) (khôi phục giá trị, thay đổi hoặc xóa nó), Trie được duyệt theo độ sâu trước, theo liên kết giữa các nút (nodes), đại diện cho từng ký tự trong khóa (key).

Không giống với một cây nhị phân tìm kiếm (BST), các nút (nodes) trong Trie không lưu trữ khóa liên quan của chúng. Thay vào đó, vị trí của một nút (node) xác định khóa mà nó được liên kết. Việc này phân phối giá trị của mỗi khóa (key) trên toàn bộ cấu trúc dữ liệu, và có nghĩa là không phải nút (node) nào cũng cần thiết giá trị liên kết.

Tất cả các nút con (children node, leaf node) của một nút (node) đều có tiền tố chung của chuỗi được liên kết với nút cha (parent node) đó và gốc (root) được liên kết với chuỗi trống. Nhiệm vụ lưu trữ dữ liệu có thể truy cập được bằng tiền tố của nó này có thể được thực hiện theo cách tối ưu hóa bộ nhớ bằng cách sử dụng Trie (cây tiền tố).

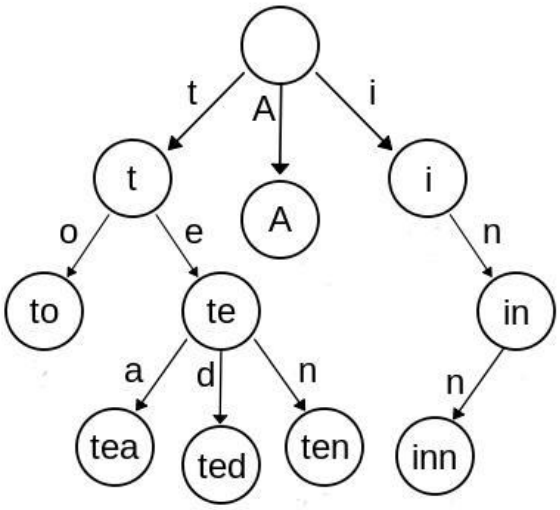
Mặc dù các lần thử có thể được khóa bằng các chuỗi ký tự, nhưng chúng không nhất thiết phải như vậy. Các thuật toán tương tự có thể được điều chỉnh cho các danh sách có thứ tự thuộc bất kỳ loại cơ bản nào, ví dụ: hoán vị của các chữ số hoặc hình dạng. Đặc biệt, một trie bitwise được khóa trên các bit riêng lẻ tạo nên một phần dữ liệu nhị phân có độ dài cố định, chẳng hạn như số nguyên hoặc địa chỉ bộ nhớ.

Khá là khó hiểu nhưng sau đây là một vài hình ảnh biểu diễn của Trie từ nhiều nguồn khác nhau:



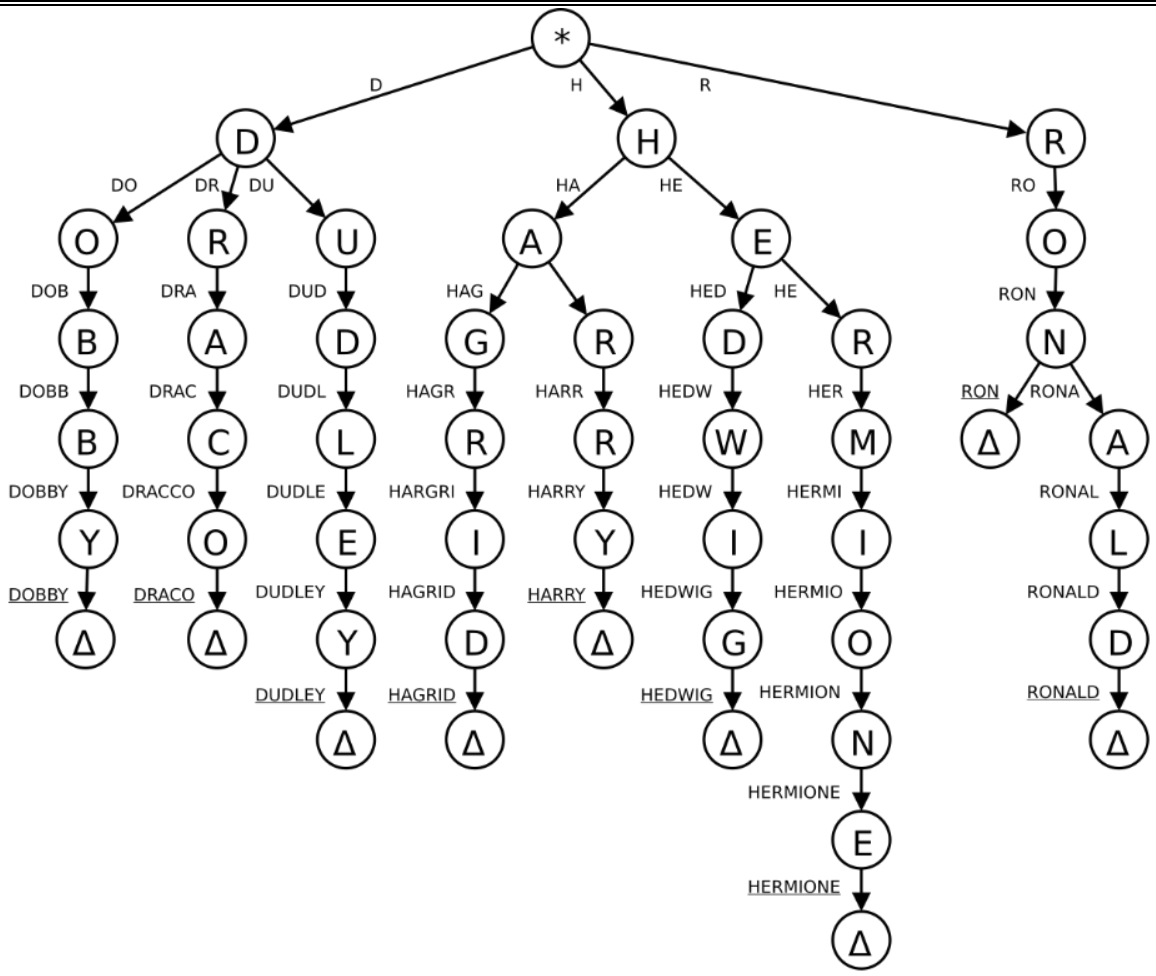
I. 1: Hình ảnh ví dụ về một Trie (nguồn: thuytrangcoding.wordpress.com)

Ví dụ hình trên từ nút 1 đến nút 8 đại diện cho xâu tiền tố "db", từ nút 1 đến nút 10 đại diện cho xâu "acd".



I. 2: Hình ảnh ví dụ về một Trie (nguồn lqdoj.edu.vn)

Hình trên trừ đỉnh gốc, các đỉnh còn lại khác rỗng tượng trưng cho tiền tố nhận được bằng cách ghép lần lượt tất cả các chữ cái trên các cạnh thuộc đường đi từ đỉnh gốc tới đỉnh đó.



I. 3: Hình ảnh ví dụ khác về Trie (nguồn: datastructures.maximal.io)

Hình ảnh ví dụ này thì có thể thấy rõ hơn các xâu ký tự được ghép.

Bản chất của cấu trúc Trie là một cây có gốc (root). Bản thân gốc (root) không chứa thông tin (NULL), nhưng mỗi cạnh nối 2 nút trên cây tương ứng với 1 ký tự; do đó đường đi từ nút gốc tới một nút bất kỳ trên cây này cho biết tập đang xét có chứa xâu tiền tố biểu diễn bởi tập các cạnh thể hiện đường đi từ nút gốc tới nút đang đề cập đến.

Một số lợi thế chính của Trie (cây tiền tố) so với BST (cây nhị phân tìm kiếm):

- Thời gian tìm kiếm ít hơn. Thao tác tìm kiếm 1 một khóa có độ dài m cần O(m) phép so sánh ký tự. Một cây nhị phân tìm kiếm sử dụng O($\log n$) phép so sánh xâu (n là số lượng khóa). Trong trường hợp xấu nhất, cây nhị phân tìm kiếm cần dùng O($m \log n$) phép so sánh ký tự.
- Trie sử dụng ít bộ nhớ hơn bởi các tiền tố chung chỉ cần được lưu trữ một lần.
- Trie cho phép tìm kiếm tiền tố trùng hợp dài nhất.
- Số lượng nút từ gốc (root) tới lá (leaf) đúng bằng độ dài khóa.

Một số lợi thế của Trie so với Hash Table (bảng băm)

- Trie cho phép liệt kê các khóa theo thứ tự từ điển
- Trie cho phép tìm kiếm tiền tố trùng hợp dài nhất
- Do không phải tính hàm băm nên trie thường nhanh hơn bảng băm trong trường hợp khóa bé chẳng hạn như số nguyên hay con trỏ.

Một số bất lợi của Trie:

- Trong một vài trường hợp, Trie có thể chậm hơn Hashtable (bảng băm), đặc biệt nếu dữ liệu được truy cập trực tiếp trên ổ đĩa cứng hoặc một số thiết bị lưu trữ phụ khác, nơi mà có thời gian truy cập ngẫu nhiên cao hơn so với bộ nhớ chính.
- Vài Tries có thể chiếm nhiều không gian bộ nhớ hơn Hashtable (bảng băm) vì bộ nhớ có thể được cấpcho mỗi ký tự trong chuỗi tìm kiếm, thay vì một đoạn nhớ duy nhất của toàn bộ mục nhập, như trong hầu hết Hashtable (bảng băm).

Như các cây nhị phân khác (BST), Trie cho phép:

- Thêm một xâu vào Trie
- Xóa một xâu khỏi Trie
- Kiểm tra xâu có tồn tại hay không

PHẦN II. STEP BY STEP

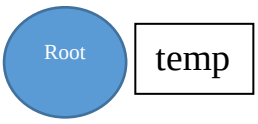
❖ Chèn một xâu vào trong một Trie (cây tiền tố) rỗng (empty)

Giả sử cho một Trie rỗng và chèn 2 từ là “cat”, “cause”.

Để chèn đối tượng (xâu ký tự) thực hiện các bước sau:

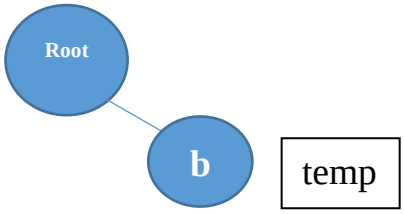
- *Bắt đầu với từ “cat”*

+ Bước 1: Đặt một biến temp = Root

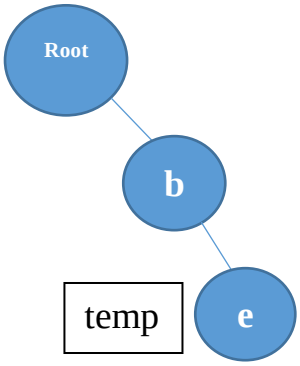


+ Bước 2: Xét từng ký tự của từ “cat”

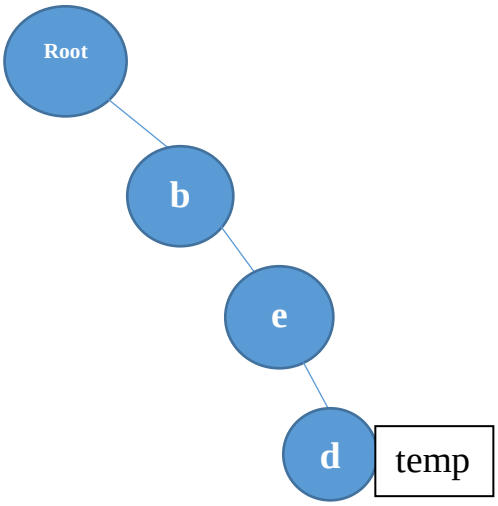
- Xét ký tự ‘b’ đầu tiên. Ta thấy temp chưa có liên kết nào với node tương ứng với ký tự ‘b’ → tạo Node ‘b’ và cho temp liên kết với Node ‘b’. Sau đó temp = Node ‘b’.



- Xét tiếp ký tự ‘e’. Ta thấy temp không có liên kết nào tướng ứng với ký tự ‘e’ → tạo Node ‘e’ và cho temp liên kết với Node ‘e’. Sau đó temp = Node ‘e’



- Xét tiếp kí tự cuối 'd'. Biến temp không có liên kết nào với Node tương ứng với kí tự 'd' → tạo Node ‘d’ và cho temp liên kết vs Node ‘d’. Sau đó ta cho temp = Node ‘d’.

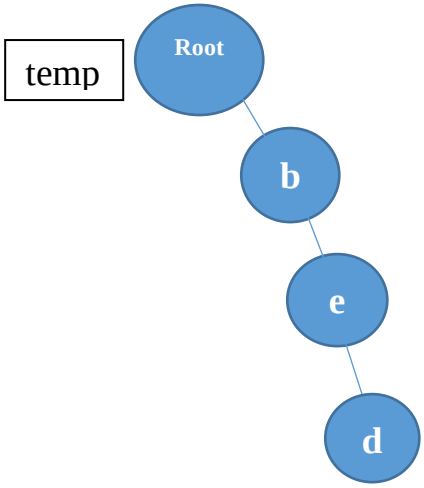


➔ Ta đã xét hết từ “bed”.

+ Bước 3: Đánh dấu Node ‘d’ là node của ký tự cuỗi cùng của từ (kết thúc từ).

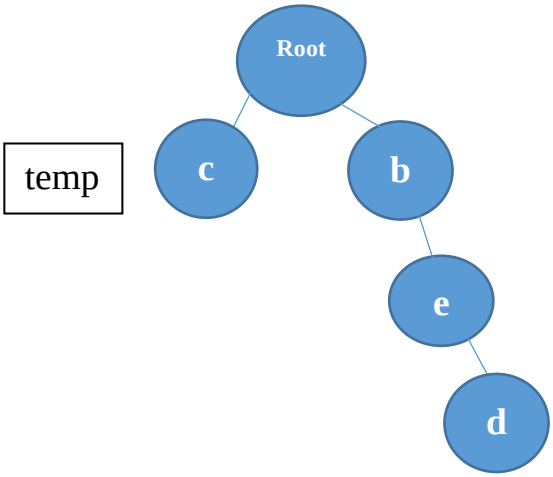
- *Tiếp theo từ thứ 2 là "catch"*

+ Bước 1: Đặt một biến temp = Root

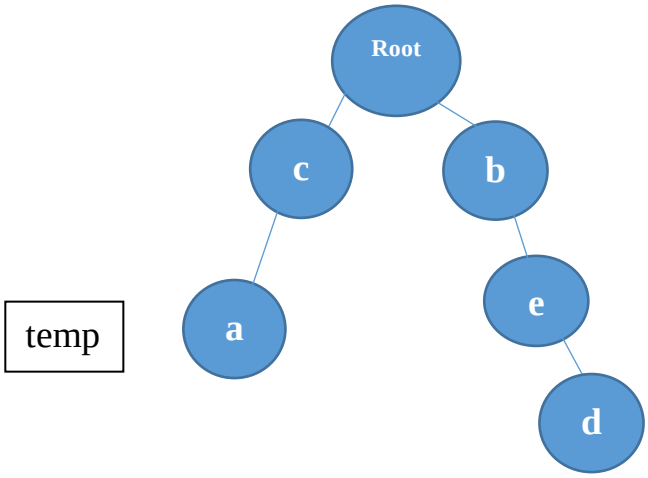


+ Bước 2: Xét từng ký tự của từ "catch":

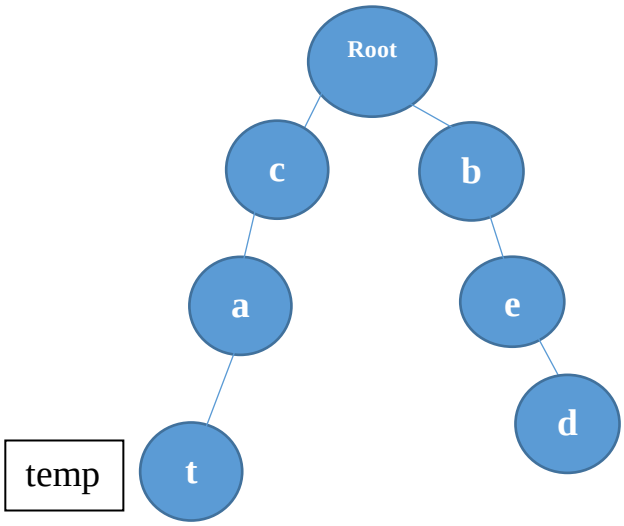
- Xét tiếp kí tự tiếp theo 'c'. Ta thấy temp không có liên kết nào với Node tương ứng với kí tự 'c' → ta tạo Node 'c' và cho temp liên kết vs Node 'c'. Sau đó ta cho temp = Node 'c'.



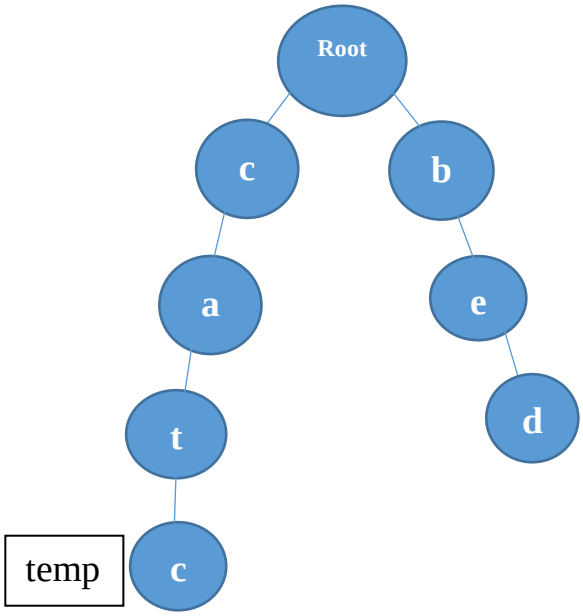
- Xét tiếp kí tự tiếp theo 'a'. Ta thấy temp không có liên kết nào với Node tương ứng với kí tự 'a' → ta tạo Node ‘a’ và cho temp liên kết vs Node ‘a’. Sau đó ta cho temp = Node ‘a’.



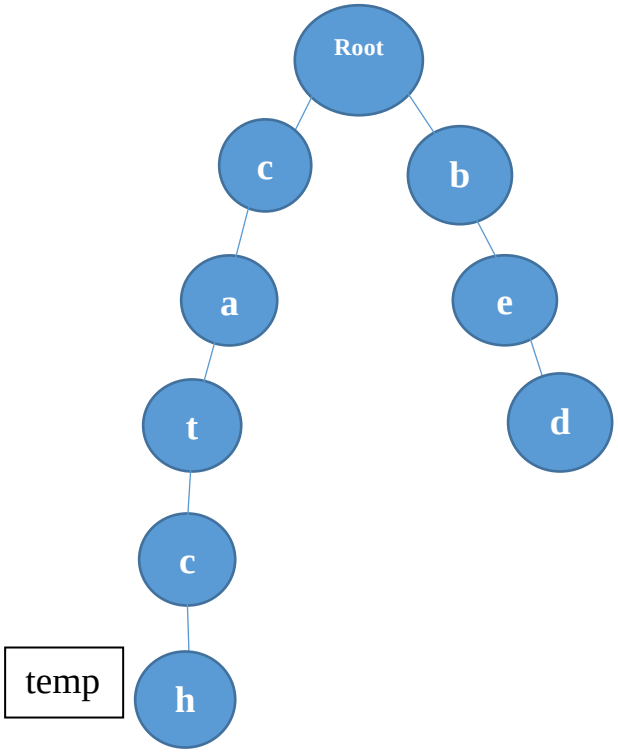
- Xét tiếp kí tự tiếp theo 't'. Ta thấy temp không có liên kết nào với Node tương ứng với kí tự 't' → ta tạo Node ‘t’ và cho temp liên kết vs Node ‘t’. Sau đó ta cho temp = Node ‘t’.



- Xét tiếp kí tự tiếp theo 'c'. Ta thấy temp không có liên kết nào với Node tương ứng với kí tự 'c' → ta tạo Node ‘c’ và cho temp liên kết vs Node ‘c’. Sau đó ta cho temp = Node ‘c’.



- Xét tiếp kí tự tiếp theo 'h'. Ta thấy temp không có liên kết nào với Node tương ứng với kí tự 'h' → ta tạo Node ‘h’ và cho temp liên kết vs Node ‘h’. Sau đó ta cho temp = Node ‘h’.

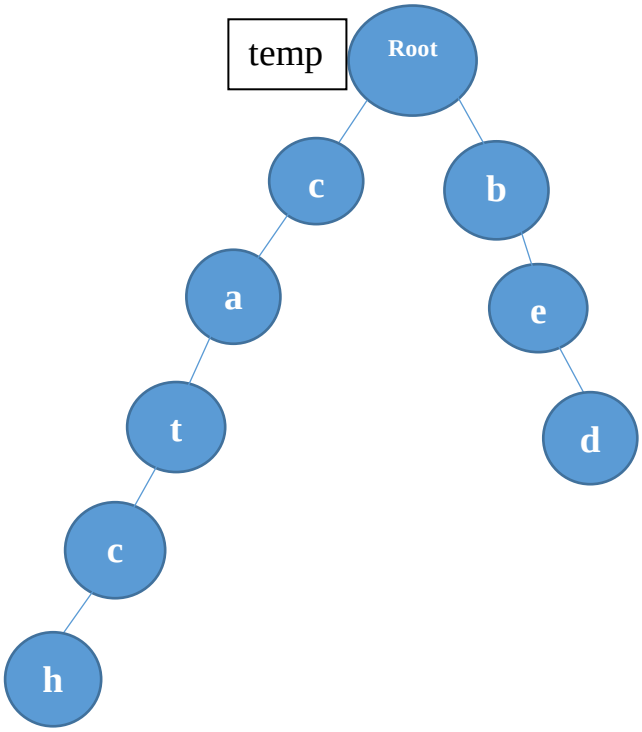


➔ Ta xét hết từ “catch”

+ Bước 3: Đánh dấu Node ‘h’ là node của ký tự cuối cùng của từ (kết thúc từ).

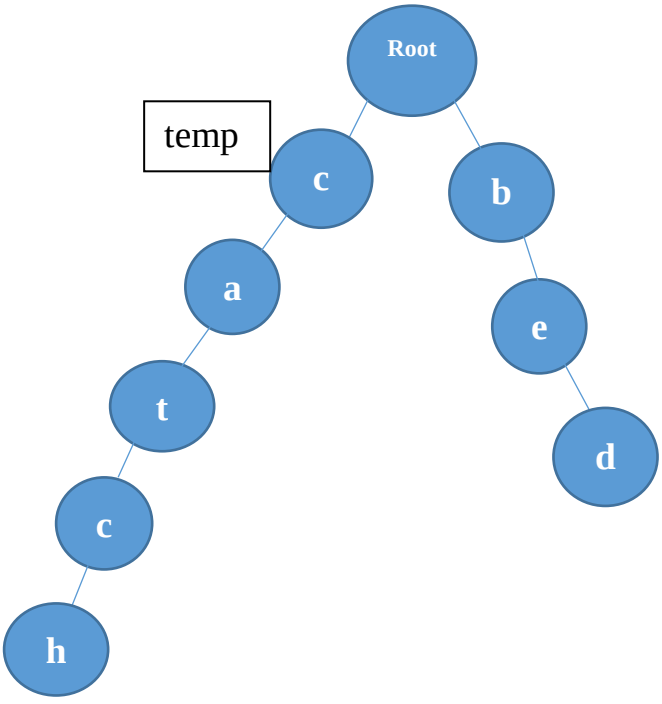
- *Tiếp theo từ thứ 3 là “cat”*

- + Bước 1: Đặt một biến temp = Root

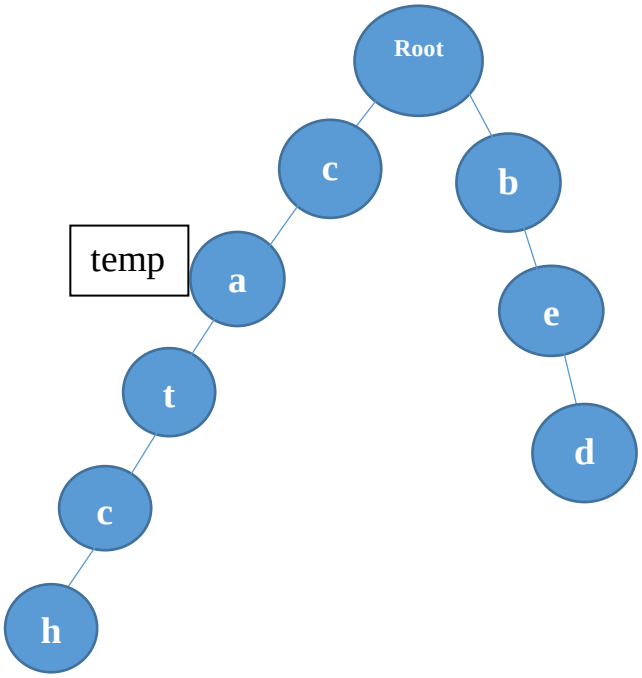


+ Bước 2: Xét từng ký tự của từ “cat”:

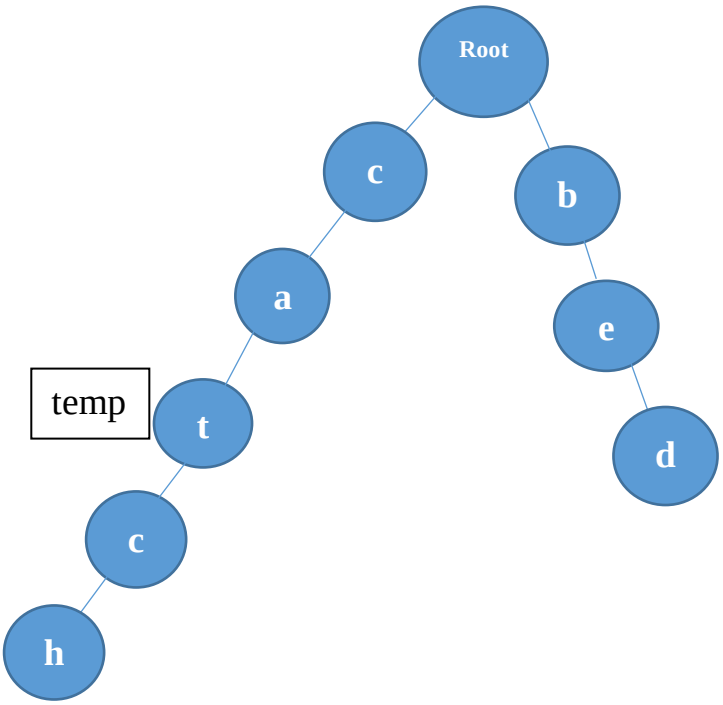
- Xét kí tự đầu tiên 'c'. Ta thấy temp đã có liên kết nào với Node tương ứng của kí tự 'c' → ta cho temp = Node ‘c’.



- Xét kí tự đầu tiên 'a'. Ta thấy temp đã có liên kết nào với Node tương ứng của kí tự 'a' → ta cho temp = Node ‘a’.



- Xét kí tự đầu tiên 't'. Ta thấy temp đã có liên kết nào với Node tương ứng của kí tự 't' → ta cho temp = Node ‘t’.

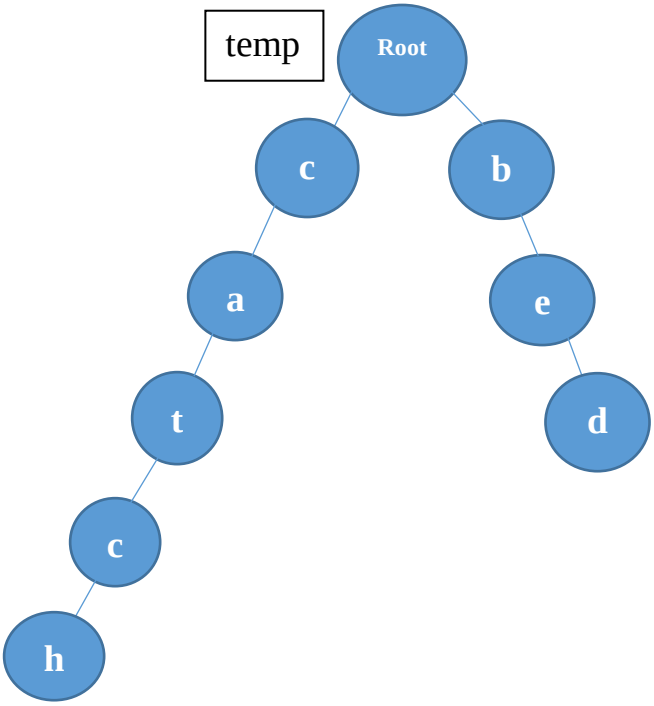


➔ Ta xét hết từ “cat”.

+ Bước 3: Đánh dấu Node ‘t’ là node của ký tự cuối cùng của từ (kết thúc từ).

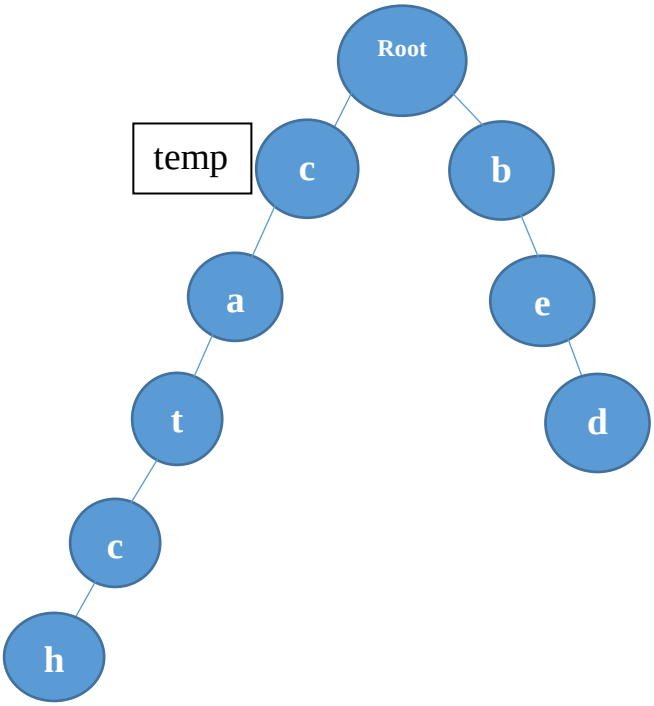
- *Tiếp theo từ thứ 4 là "car"*

+ Bước 1: Đặt một biến temp = Root

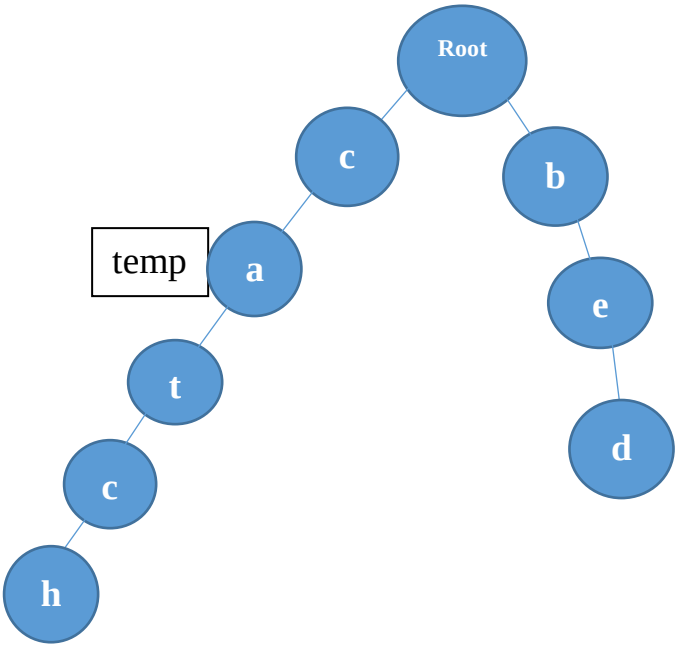


+ Bước 2: Xét từng ký tự của từ "car":

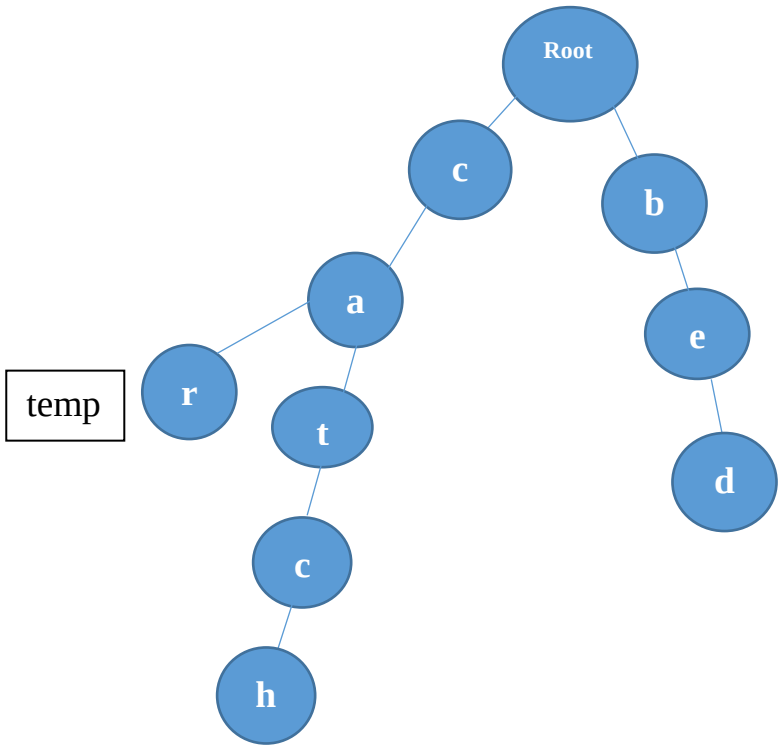
- Xét kí tự đầu tiên 'c'. Ta thấy temp đã có liên kết nào với Node tương ứng của kí tự 'c' → ta cho temp = Node 'c'.



- Xét kí tự đầu tiên 'a'. Ta thấy temp đã có liên kết nào với Node tương ứng của kí tự 'a' → ta cho temp = Node ‘a’.



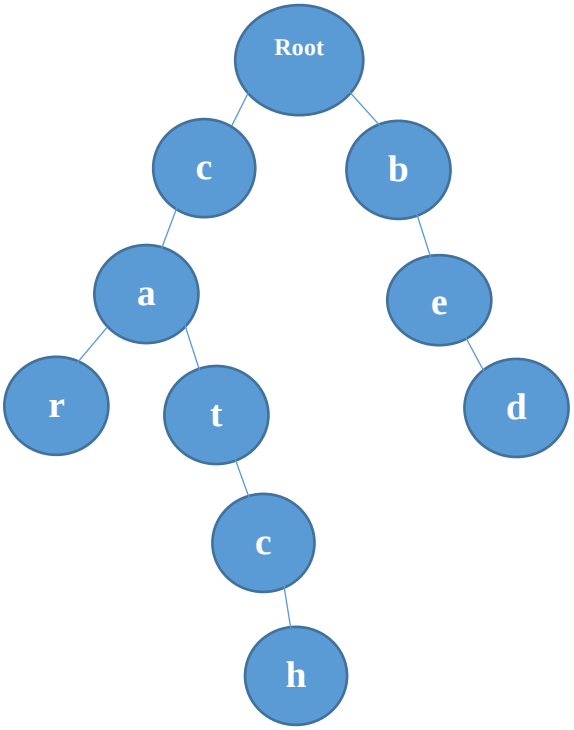
- Xét tiếp kí tự tiếp theo 'r'. Ta thấy temp không có liên kết nào với Node tương ứng với kí tự 'r' → ta tạo Node ‘r’ và cho temp liên kết vs Node ‘r’. Sau đó ta cho temp = Node ‘r’.



➔ Ta xét hết từ “car”.

+ Bước 3: Đánh dấu Node ‘r’ là node của ký tự cuối cùng của từ (kết thúc từ).

❖ Xóa một xâu khỏi Trie (cây tiền tố)

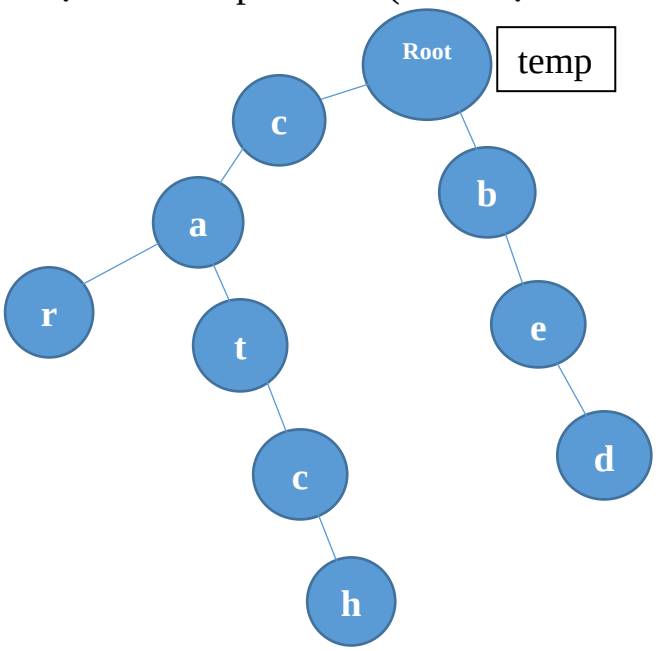


Trong Trie trên bao gồm các từ "cat", "catch", "car", bed".

Trường hợp 1: Chuỗi bắt đầu từ Root đến hết không nhánh.

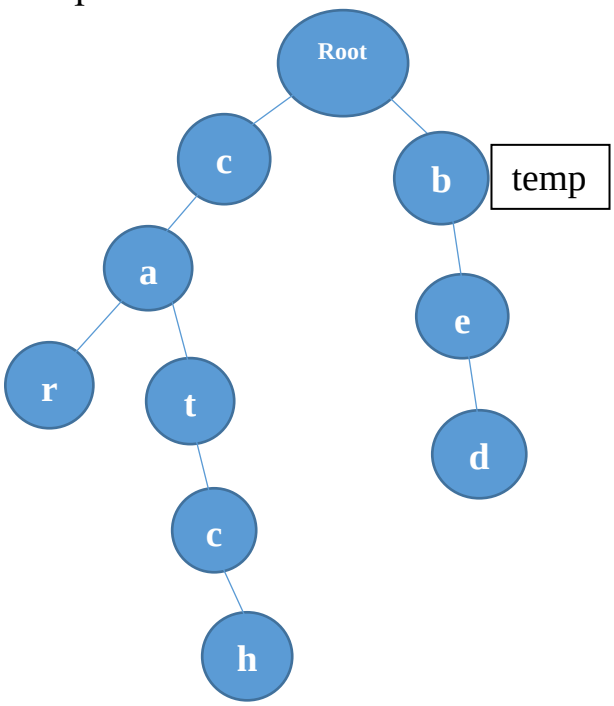
- *Xóa từ "bed" khỏi Trie (cây tiền tố)*

+ Bước 1: Đặt một biến temp = Root (sẽ trở lại nếu Root không có liên kết nào).

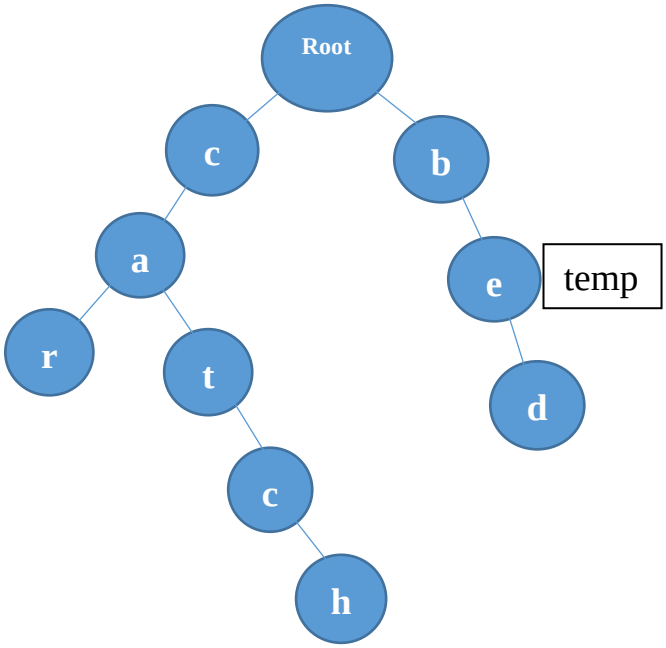


+ Bước 2: Tra từ trong Trie (kiểm tra so sánh từng ký tự)

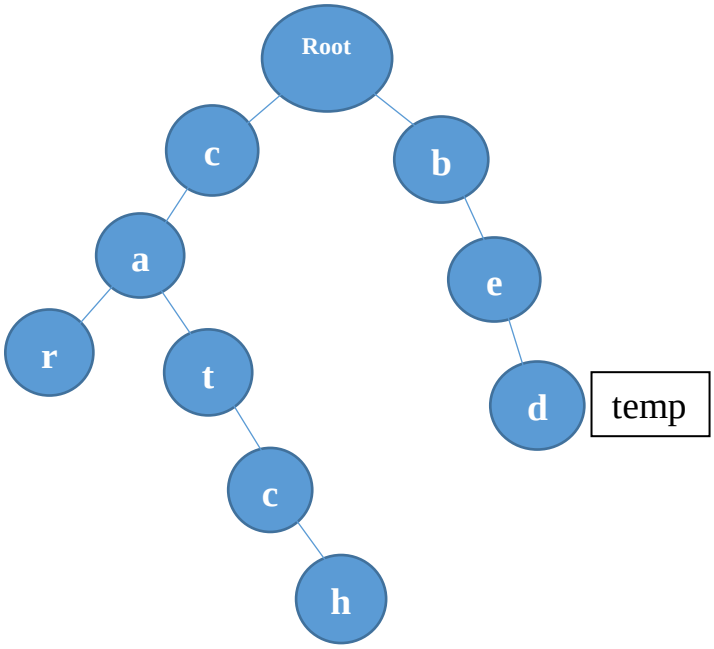
- Đệ qui lần 1 (Temp = địa chỉ của Root): Xét ký tự ‘b’ đầu tiên, kiểm tra xem temp có liên kết với node tương ứng với ký tự ‘b’ hay không. Kiểm tra thấy có Node ‘b’ tương ứng và chưa xét hết từ, Node ‘b’ không có đánh dấu kết thúc từ ta cho temp = Node ‘b’.



- Đệ qui lần 2 (Temp = địa chỉ của Node ‘b’): Xét tiếp ký tự ‘e’, kiểm tra temp có liên kết với node tương ứng với ký tự ‘e’ không. Kiểm tra thấy có Node ‘e’ tương ứng và chưa xét hết từ, Node ‘e’ không có đánh dấu kết thúc từ ta cho temp = Node ‘e’.

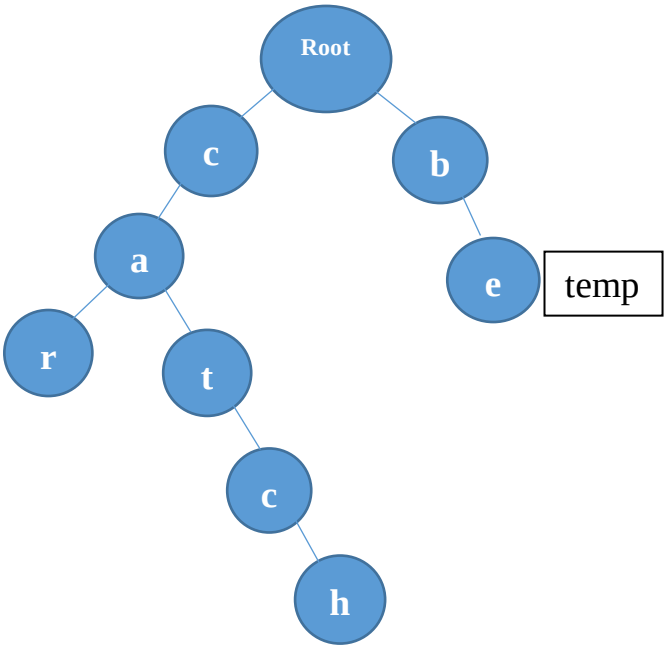


- Đệ qui lần 3 (Temp = địa chỉ của Node ‘e’): Xét tiếp ký tự ‘d’, kiểm tra temp có liên kết với node tương ứng với ký tự ‘d’ không. Kiểm tra thấy có Node ‘d’ tương ứng và đã xét hết ký tự của từ, Node ‘d’ có đánh dấu kết thúc từ, ta cho temp = Node ‘d’.

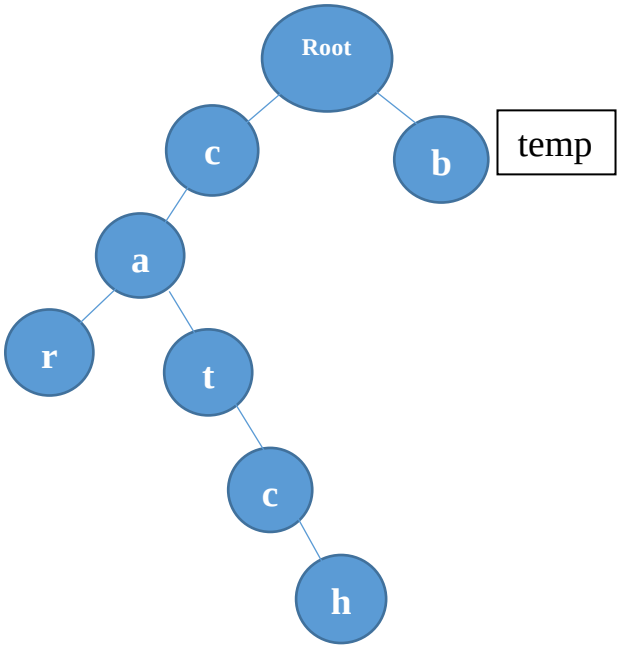


+ Bước 3: Xóa các node

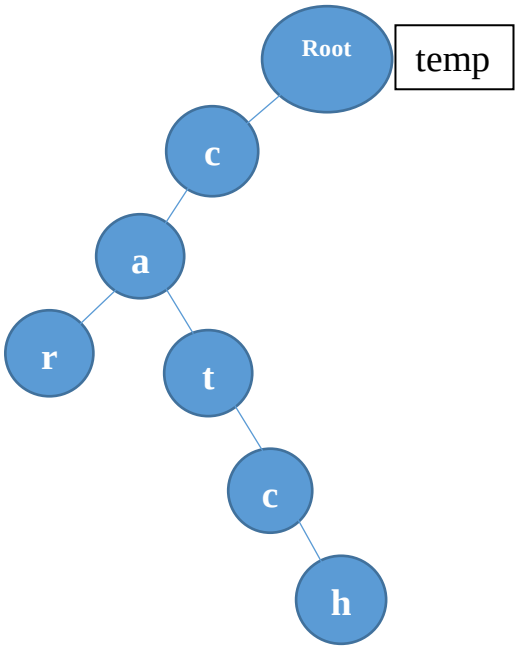
- Đệ qui lần 4 (temp = địa chỉ của Node ‘d’): Ta thấy temp đang ở địa chỉ của Node cuối cùng, xóa đánh dấu ký tự kết thúc của Node ‘d’ (False) xóa Node ‘d’. Lúc này kết thúc lượt đệ qui cuối cùng, tiến hành thực hiện các lệnh sau đệ qui và thoát đệ qui.



- Thoát đệ qui lần 4 (Hàm đệ qui đã trả về Trie không có Node ‘d’), biến temp đang ở Node ‘e’, kiểm tra không có đánh dấu kết thúc từ và không có liên kết với node khác, xóa Node ‘e’.



- Thoát đệ qui lần 3 (Hàm đệ qui đã trả về Trie không có Node ‘e’), biến temp đang ở Node ‘b’, kiểm tra không có đánh dấu kết thúc từ và không có liên kết với node khác, xóa Node ‘b’.



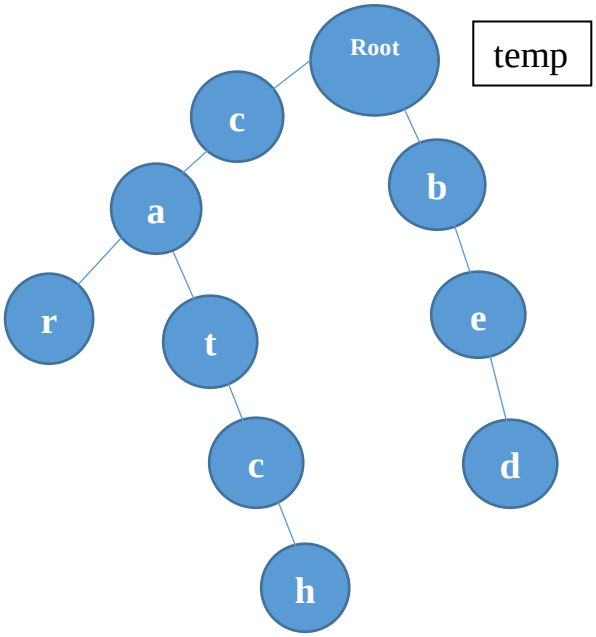
- Thoát đệ qui lần 2 (Hàm đệ qui đã trả về Trie không có Node ‘b’), biến temp đang ở root, kiểm tra không có đánh dấu kết thúc từ nhưng có liên kết với node khác nên không thực hiện xóa. Thoát đệ qui lần 1 và Kết Thúc hàm.

➔ Đã xóa xong từ “bed” khỏi Trie.

Trường hợp 2: Chuỗi muốn xóa có liên kết với chuỗi khác.

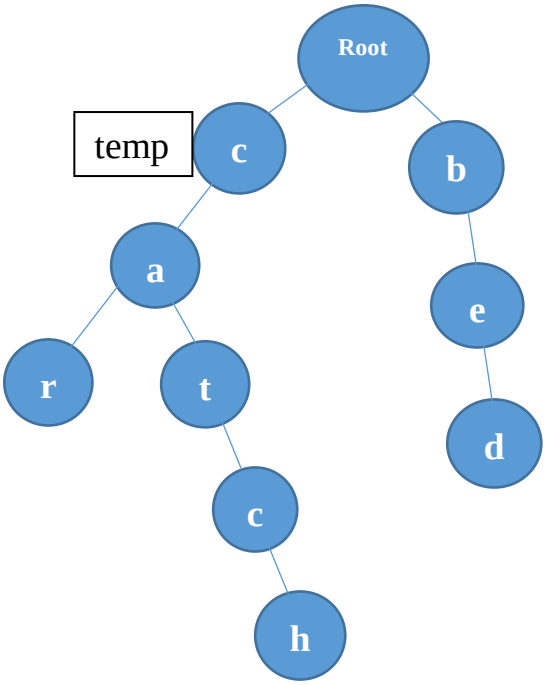
- *Xóa từ "car" khỏi Trie (cây tiền tố)*

+ Bước 1: Đặt một biến temp = Root (sẽ trở lại nếu Root không có liên kết nào).

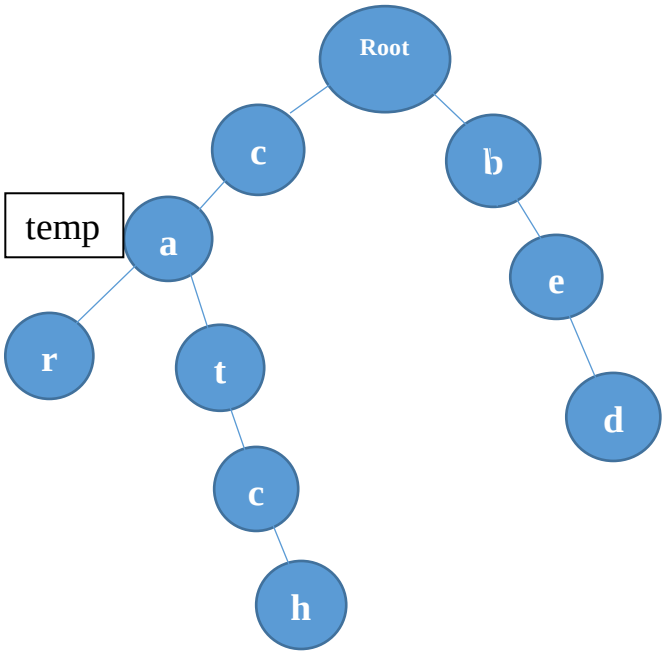


+ Bước 2: Tra từ trong Trie (kiểm tra so sánh từng ký tự)

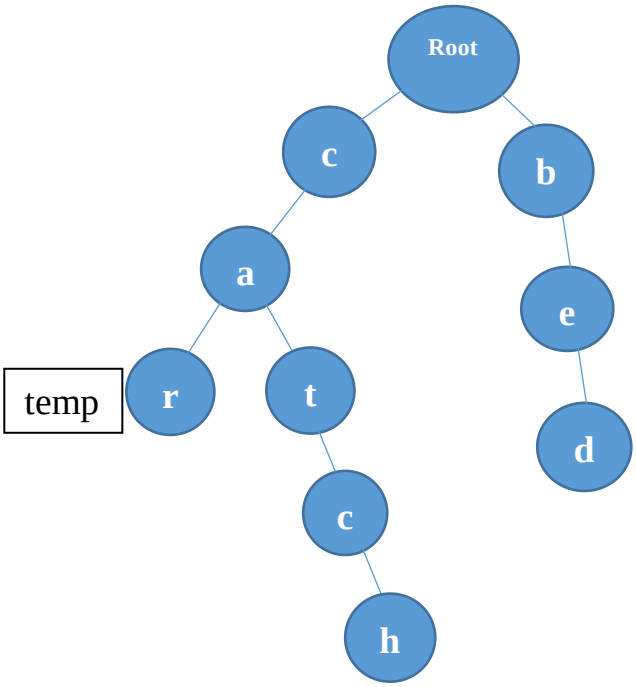
- Đệ qui lần 1 (Temp = địa chỉ của Root): Xét ký tự 'c' đầu tiên, kiểm tra xem temp có liên kết với node tương ứng với ký tự 'c' hay không. Kiểm tra thấy có Node 'c' tương ứng và chưa xét hết từ, Node 'c' không có đánh dấu kết thúc từ ta cho temp = Node 'c'.



- Đệ qui lần 2 (Temp = địa chỉ của Node ‘c’): Xét tiếp ký tự ‘a’, kiểm tra temp có liên kết với node tương ứng với ký tự ‘a’ không. Kiểm tra thấy có Node ‘a’ tương ứng và chưa xét hết từ, Node ‘a’ không có đánh dấu kết thúc từ ta cho temp = Node ‘a’.

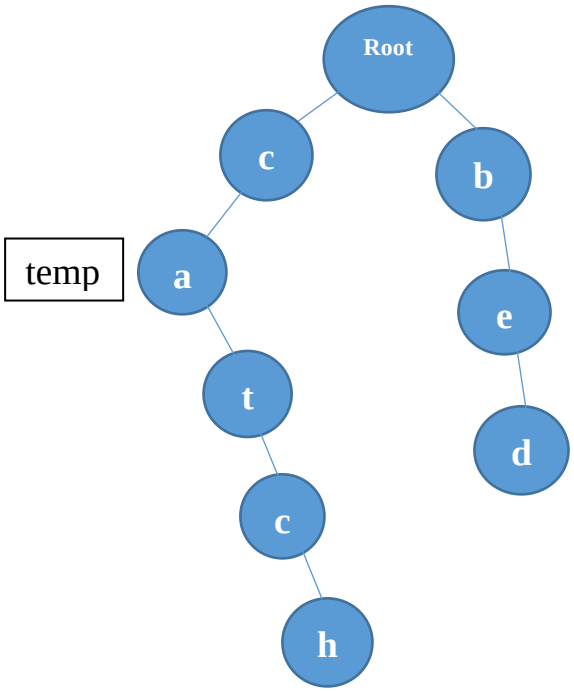


- Đệ qui lần 3 (Temp = địa chỉ của Node ‘a’): Xét tiếp ký tự ‘r’, kiểm tra temp có liên kết với node tương ứng với ký tự ‘r’ không. Kiểm tra thấy có Node ‘r’ tương ứng và đã xét hết ký tự của từ và node có đánh dấu kết thúc, ta cho temp = Node ‘r’.

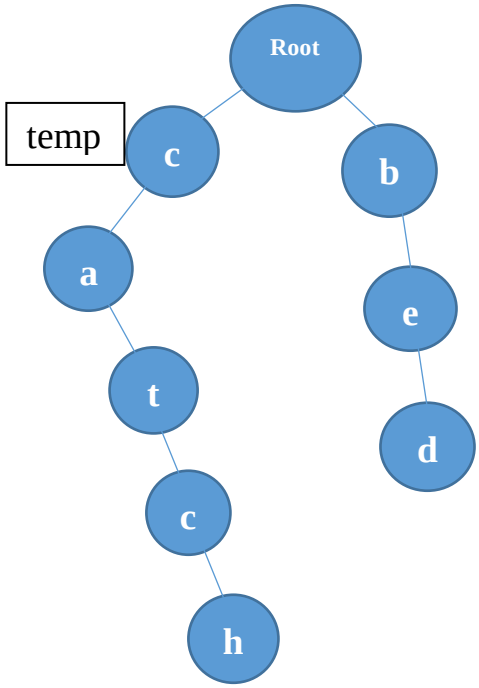


+ Bước 3: Xóa các node

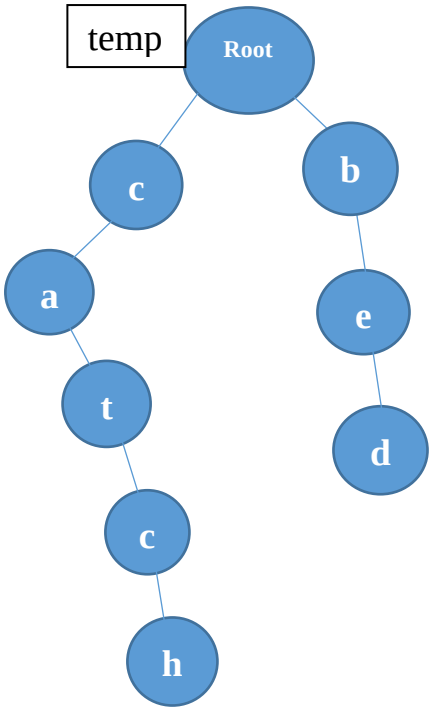
- Đệ qui lần 4 (Temp = địa chỉ của Node ‘r’): Ta thấy temp đang ở địa chỉ của Node cuối cùng, xóa đánh dấu ký tự kết thúc của Node ‘r’ (False) xóa Node ‘r’. Lúc này kết thúc lượt đệ qui cuối cùng, tiến hành thực hiện các lệnh sau đệ qui và thoát đệ qui.



- Thoát đệ qui lần 4 (Hàm đệ qui đã trả về Trie không có Node ‘r’), biến temp đang ở Node ‘a’, kiểm tra không có đánh dấu kết thúc từ nhưng có liên kết với node khác nên không thực hiện xóa.



- Thoát đệ qui lần 3 (Hàm đệ qui đã trả về Trie không có Node ‘r’), biến temp đang ở Node ‘c’, kiểm tra không có đánh dấu kết thúc từ nhưng có liên kết với node khác nên không thực hiện xóa.



- Thoát đệ qui lần 2 (Hàm đệ qui đã trả về Trie không có Node ‘r’), biến temp đang ở root, kiểm tra không có đánh dấu kết thúc từ nhưng có liên kết với node khác nên không thực hiện xóa.
- Thoát đệ qui lần 1 và Kết Thúc hàm.

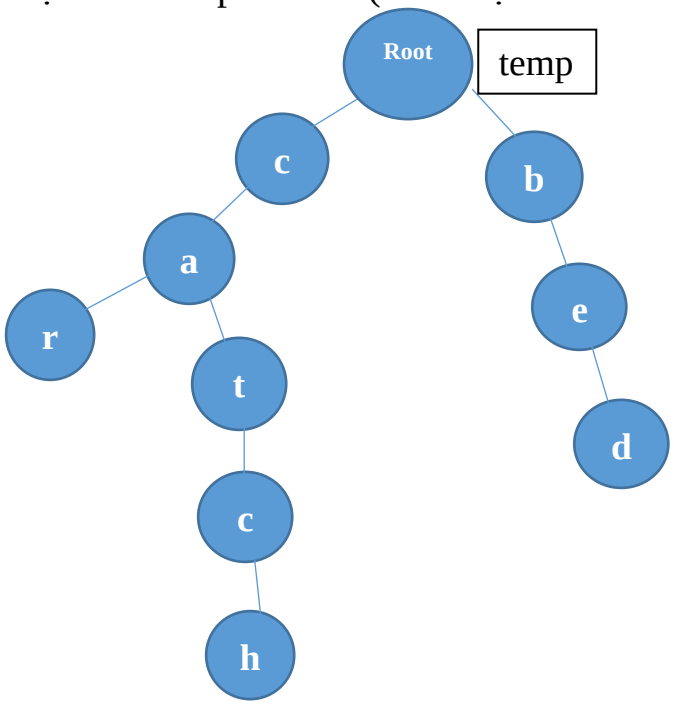
➔ Đã xóa xong từ “car” khỏi Trie

- Bổ sung: Có thể xóa từ “catch” khỏi Trie theo các bước trên tương tự như từ “car”

Trường hợp 3: Chuỗi muốn xóa nằm trong chuỗi khác.

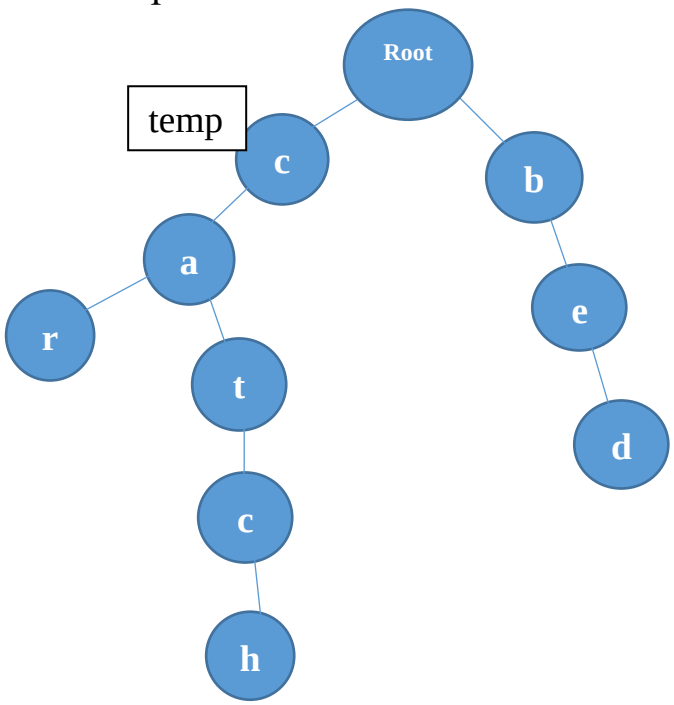
Xóa từ "cat" khỏi Trie (cây tiền tố)

+ Bước 1: Đặt một biến temp = Root (sẽ trở lại nếu Root không có liên kết nào).

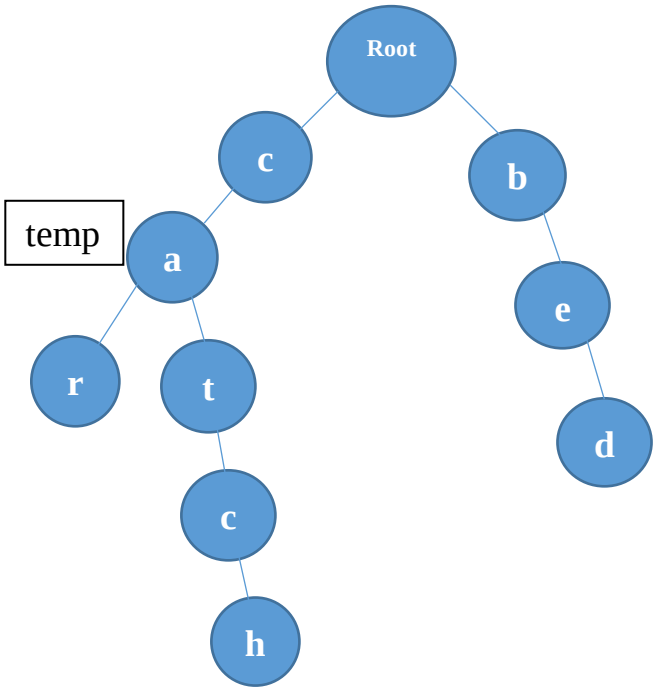


+ Bước 2: Tra từ trong Trie (kiểm tra so sánh từng ký tự)

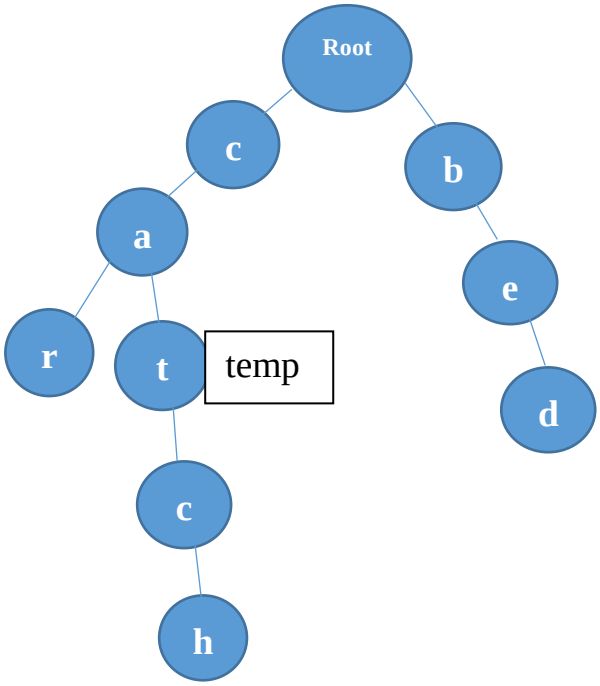
- Đệ qui lần 1 (Temp = địa chỉ của Root): Xét ký tự 'c' đầu tiên, kiểm tra xem temp có liên kết với node tương ứng với ký tự 'c' hay không. Kiểm tra thấy có Node 'c' tương ứng và chưa xét hết từ, Node 'c' không có đánh dấu kết thúc từ ta cho temp = Node 'c'.



- Đệ qui lần 2 (Temp = địa chỉ của Node ‘c’): Xét tiếp ký tự ‘a’, kiểm tra temp có liên kết với node tương ứng với ký tự ‘a’ không. Kiểm tra thấy có Node ‘a’ tương ứng và chưa xét hết từ, Node ‘a’ không có đánh dấu kết thúc từ ta cho temp = Node ‘a’.

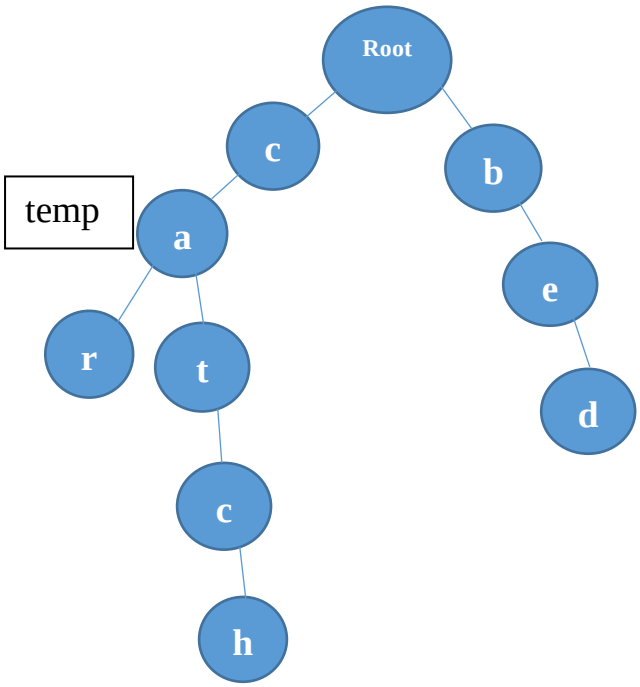


- Đệ qui lần 3 (Temp = địa chỉ của Node ‘a’): Xét tiếp ký tự ‘t’, kiểm tra temp có liên kết với node tương ứng với ký tự ‘t’ không. Kiểm tra thấy có Node ‘t’ tương ứng và đã xét hết ký tự của từ và Node ‘t’ có đánh dấu kết thúc, ta cho temp = Node ‘t’.

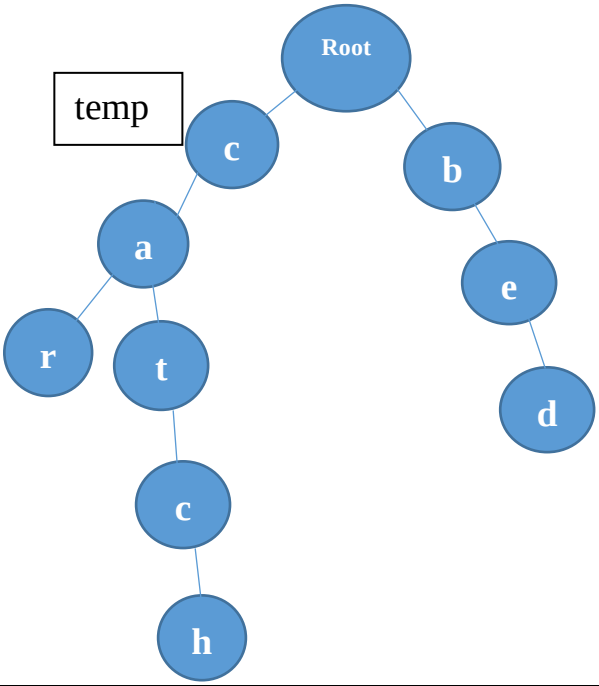


+ Bước 3: Xóa các node

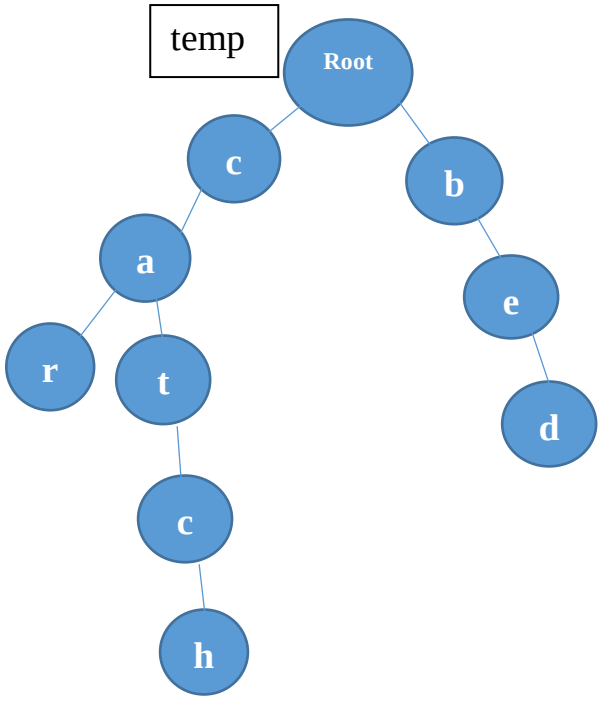
- Đệ qui lần 4 (Temp = địa chỉ của Node ‘t’): Ta thấy temp đang ở địa chỉ của Node ‘t’ được đánh dấu cuối cùng của từ đang liên kết với Node khác, nên ta đánh dấu lại Node ‘t’ này không phải node cuối cùng. Vì Node ‘t’ còn liên kết với Node khác nên không thực hiện xóa Node ‘t’. Lúc này kết thúc lượt đệ qui cuối cùng, tiến hành thực hiện các lệnh sau đệ qui và thoát đệ qui.



- Thoát đệ qui lần 4 (Hàm đệ qui đã trả về cây Trie với các Node như cũ), biến Temp đang ở Node ‘a’, kiểm tra không có đánh dấu kết thúc từ nhưng có liên kết với node khác nên không thực hiện xóa.



- Thoát đệ qui lần 3 (Hàm đệ qui đã trả về Trie với các Node như cũ), biến Temp đang ở Node 'c', kiểm tra không có đánh dấu kết thúc từ nhưng có liên kết với node khác nên không thực hiện xóa.



- Thoát đệ qui lần 2 (Hàm đệ qui đã trả về Trie với các Node như cũ), biến Temp đang ở root, kiểm tra không có đánh dấu kết thúc từ nhưng có liên kết với node khác nên không thực hiện xóa.
- Thoát đệ qui lần 1 và Kết Thúc hàm.

PHẦN III. ĐỘ PHỨC TẠP

Dựa trên ước lượng lý thuyết, nhóm đưa ra được giá trị Big-O cho các chức năng theo yêu cầu của bài nghiên cứu như sau:

1. Test whether a Trie is empty:

Với n là tổng số key trong Trie cùng với vòng lặp for nên dựa trên lý thuyết ta có thể ước lượng độ phức tạp thời gian được tính trong trường hợp xấu nhất là O (n) và trường hợp tốt nhất tại O (0).

2. Get the number of items in a Trie:

Trong trường hợp này, độ phức tạp về thời gian phụ thuộc vào n là tổng các key trong Trie và m là độ dài của key. Từ đó, dựa trên các ước lượng lý thuyết ta rút ra được đồ phức tạp trong trường hợp xấu nhất là O(mn).

3. Find the item in a Trie:

Cũng với vòng lặp tương tự, việc chèn trong giới hạn thời gian chạy với n là độ dài của key, thì sử dụng độ phức tạp thời gian trong trường hợp xấu nhất để tìm kiếm bản ghi được liên kết với key này cũng là O (n).

4. Insert a new item into the Trie:

Việc loại bỏ một chuỗi không phải là một tính năng điển hình của cấu trúc dữ liệu Trie. Xóa một nút cũng là O (n), với n là độ dài của key cần xóa.

5. Remove an item from the Trie

Việc loại bỏ một chuỗi không phải là một tính năng điển hình của cấu trúc dữ liệu Trie. Xóa một nút cũng là O (n), với n là độ dài của key cần xóa

6. Build a Trie from given items:

Theo các cấu trúc trên, khi xem xét độ phức tạp theo thời gian Big O của cấu trúc dữ liệu Trie. Khoảng thời gian cần thiết để tạo một Trie được liên kết trực tiếp với số lượng từ / key mà Trie đó chứa và thời lượng các key đó có thể có. Thời gian chạy trong trường hợp xấu nhất để tạo một Trie là kết hợp của m (độ dài của key dài nhất trong Trie) có độ phức tạp thời gian là O(m) và n (tổng số key trong Trie) có độ phức tạp thời gian là O(n). Do đó, thời gian chạy trường hợp xấu nhất để tạo một Trie là O (mn).

7. Remove all elements from the Trie:

Tương tự như việc tìm số lượng các phần tử trong Trie, ta cũng rút ra được độ phức tạp thời gian trong trường hợp xấu nhất là O(mn) với n là tổng các key trong Trie và m là độ dài của key

➔ Kết luận

- Độ phức tạp về thời gian của việc tìm kiếm, chèn và xóa khỏi một Trie phụ thuộc vào độ dài của từ a đang được tìm kiếm, chèn hoặc xóa và tổng số từ, n, làm cho thời gian chạy của các thao tác này là O (an) . Tất nhiên, đối với từ dài nhất trong bộ ba, việc chèn, tìm kiếm và xóa sẽ tốn nhiều thời gian và bộ nhớ hơn so với từ ngắn nhất trong bộ ba
- Thời gian truy xuất / chèn của Trie trong trường hợp xấu nhất tốt hơn hashTable và cây tìm kiếm nhị phân, cả hai đều lấy thời gian trong trường hợp xấu nhất là O (n) để truy xuất / chèn

PHẦN IV. ỨNG DỤNG THỰC TẾ

Trong cấu trúc Trie bạn sẽ không bị giới hạn bởi các từ nối tiền tố và Trie có thể lưu trữ địa chỉ IP, số điện thoại, các thuộc tính trên đối tượng mà bạn có thể tìm kiếm, và còn hơn thế nữa …Trie đóng một vai trò vô cùng lớn trong việc biến mọi thứ trở nên hiệu quả và tiện lợi hơn. Ở đây tôi sẽ nêu ra 4 lợi ích mà Trie đem lại cho sự phát triển công nghệ, mà đặc biệt là trong lĩnh vực tìm kiếm thông tin:

❖ **Autocomplete(tự động hoàn thành):** có thể coi đây là lợi ích tuyệt vời nhất khi mà ứng dụng của nó là vô cùng lớn và xu hướng sử dụng rộng rãi được tính theo cấp số nhân trong thời đại công nghệ hiện nay. Trie cho phép phần mềm dự đoán phần còn lại của từ mà người dùng đang nhập. Trong phần giao diện, người dùng có thể gõ một trong các tác vụ kèm theo như phím tab, dấu mũi tên,…

Tính năng này đẩy mạnh tốc độ tìm kiếm giữa người dùng và thiết bị. Điều này góp phần nâng cao trải nghiệm và đặc biệt là Trie có thể dự đoán một cách chính xác từ mà người dùng muốn nhập chỉ sau một vài lần gõ. Tính năng này sẽ càng chính xác hơn khi được giới hạn tại phạm vi các từ có khả năng hay các từ có tính thông dụng hơn.

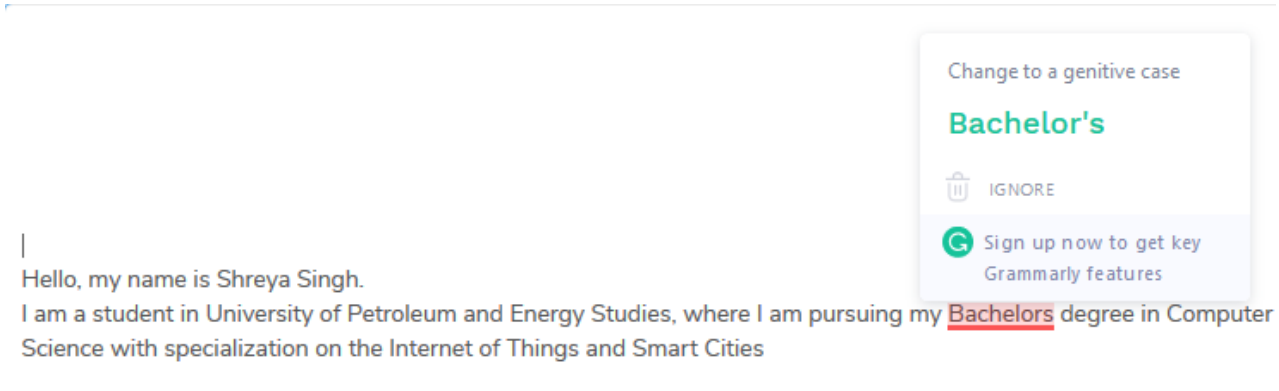
Một trong các lĩnh vực khoa học quan trọng nhất hiện nay là trí tuệ nhân tạo cũng cải tiến để giúp phần mềm dự đoán một cách hiệu quả hơn thông qua trải nghiệm và sử dụng của người dùng.

Một vài ứng dụng của autocomplete trong đời sống:

- **Trong từ điển**: giờ đây người dùng có thể dễ dàng tra cứu từ điển một cách nhanh chóng với chỉ vài lần gõ từ tiền tố.
- **Trong Email**: dựa trên lịch sử người dùng, Email dùng autocomplete để điền các địa chỉ email mà người dùng muốn tìm kiếm. Nhiều nhà cung cấp dịch vụ email như Gmail cũng cung cấp tính năng autocomplete trên các chuỗi nội dung có sử dụng trí thông minh nhân tạo.
- **Trong các công cụ tìm kiếm**: dựa trên những chi tiết hay lịch sử tìm kiếm thích hợp nhất để ứng dụng cho autocomplete. Ngoài ra còn dùng những thuật toán cải tiến hơn mà không phụ thuộc vào ngôn ngữ như Levenshtein Algorithm.
- **Trong công cụ truy vấn cơ sở dữ liệu**: tính năng autocomplete được sử dụng để hoàn thành việc truy vấn các tên bảng, cơ sở dữ liệu, thuộc tính,… Với tất cả những thứ được đề cập ở trên trong thư việ dữ liệu hay trong các định nghĩa siêu dữ liệu khác. Những yêu cầu bao gồm tất cả thông tin như bảng, giá trị, cơ sở dữ liệu that người dùng có thể truy cập. Ví dụ: *SQL Server Management Studio,…*

- **Trong bộ xử lý chữ**: nhằm cho việc giảm tải sự lặp lại của những từ hay những ý giống nhau dựa trên tài liệu gần dây và thiết lập dữ liệu được sử dụng cho cái thiện tính năng autocomplete.
- **Trong biên dịch Command Line**: bằng việc sử dụng danh sách các yêu cầu, người dùng có thể sử dụng tính năng tự động điền các yêu cầu này. Chỉ với một lần tab, tác vụ sẽ được hoàn thành và việc điền chữ gần như chuẩn xác. Các trình biên dịch Command Line sử dụng tính năng tự động điền rất phổ biến hiện nay bao gồm trình Bash của Linux hay trình PowerShell của Windows.

❖ **Spell Checkers/ Auto-correct (trình kiểm tra chính tả/ tự động sửa):**



IV. 1: Ứng dụng cho trinhf kiểm tra chính tả, tự động sửa

Dựa trên 3 bước sau đây, bạn sẽ hiểu được cách thức hoạt động của chúng:

Bước 1: Kiểm tra từ mà người dùng nhập trong thư viện dữ liệu.
Bước 2: Đưa ra những đề xuất hợp lý và thông dụng.
Bước 3: Sắp xếp các đề xuất đó theo mức độ ưu tiên từ cao tới thấp

Cấu trúc dữ liệu Trie còn được sử dụng để lưu trữ thư viện dữ liệu và thuật toán cho việc tìm kiếm những từ đó trong từ điển và cung cấp danh sách các từ hợp lý theo những đề xuất được đưa ra.

❖ **Longest Prefix Matching (đối sánh tiền tố dài nhất):** còn được gọi là đối sánh độ dài tiền tố lớn nhất, cái được sử dụng trong mạng bằng các thiết bị định tuyến trong IP. Nó được sử dụng để chọn bảng từ bảng định tuyến dựa trên lịch sử trước đó được chuyển đến mạng.

Việc tối ưu hóa, định tuyến mạng yêu cầu liên kết độ phức tạp của trường hợp xấu nhất cho thời gian tìm kiếm đến O(n), nơi n là độ dài của địa chỉ URL trong bit. Để nâng cao tốc độ xử lý tìm kiếm, Multiple Bit Trie Schemes được phát triển để thực hiện tra cứu đa bit nhanh hơn.

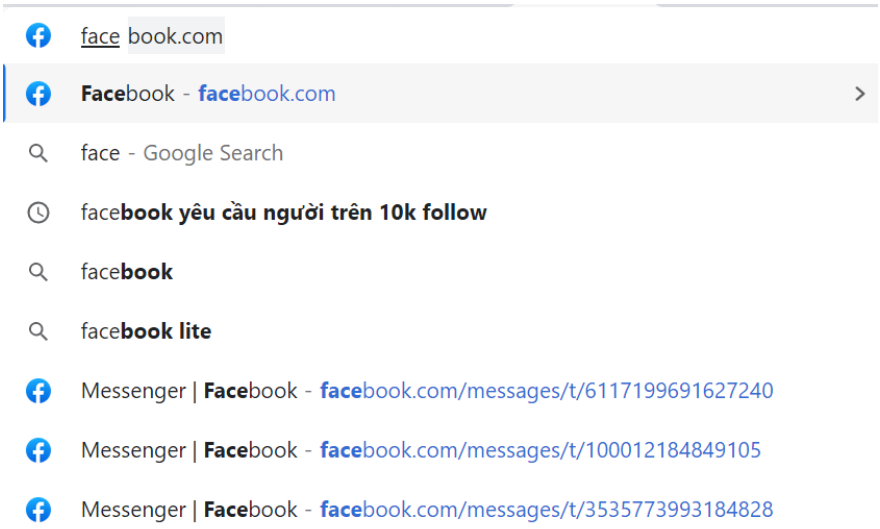
IP Prefix	Router
192.168.20.16/28	A
192.168.0.0/16	B

Hex Format	Binary Format
192.168.20.191	11000000.10101000.00010100.10111111
192.168.20.16/28	11000000.10101000.00010100.00010000
192.168.0.0/16	11000000.10101000.00000000.00000000

Bảng IV. 1: Ví dụ cho ứng dụng Longest Prefix Matching

❖ Browser History (lịch sử trình duyệt):

Các trang trình duyệt sẽ theo dõi lịch sử dùng web của người dùng. Bằng việc sắp xếp lịch sử, và với số lần truy cập đến những trang đặc biệt như là một từ khóa và tổ chức theo Trie. Theo cách này, người dùng được nhận những đề xuất của web khi lịch sử những tiền tố của lần truy cập trước được nhập trên thanh địa chỉ



IV. 2: Ứng dụng Trie trong lịch sử trình duyệt

TỔ CHỨC CODE

❖ Code được chia làm 3 File:

➢ Source.cpp: Chứa các hàm theo yêu cầu "Implement the tree data structure trie that includes the following bacsic operations".

- Kiểm tra xem một Trie có rỗng

```
bool isTrieEmpty(TrieNode *root)
```

- Đếm số đối tượng trong một Trie

```
unsigned int numberOfItem(TrieNode *root)
```

- Tìm kiếm đối tượng trong một Trie

```
bool find(TrieNode *root, string key)
```

- Thêm/chèn một đối tượng vào trong Trie

```
void insert(TrieNode *root, string key)
```

- Xóa một đối tượng khỏi Trie

```
TrieNode* remove(TrieNode* root, string key, int depth = 0)
```

- Xây dựng một Trie từ các đối tượng được cho

```
TrieNode *buildTrie(string key[], int n)
```

- Xóa hết tất cả các đối tượng của một Trie

```
TrieNode *removeTrie(TrieNode *root)
```

- Dictionary.cpp: Chương trình từ điển.
  - Tạo một Node mới

```
TrieNode *getNode()
```

  - Kiểm tra Trie có rỗng hay không

```
bool isTrieEmpty(TrieNode *root)
```

  - Tìm kiếm từ có tồn tại trong từ điển hay không

```
bool find(TrieNode *root, string key)
```

  - Chèn một từ vào từ điển

```
void insert(TrieNode *root, string key)
```

  - Tạo một Trie lưu trữ các từ của từ điển

```
TrieNode *buildTrie(string *key, int n)
```

  - Xóa hết từ điển

```
TrieNode *removeTrie(TrieNode *root)
```

  - Hiển trị Trie

```
void showTrie(TrieNode *root, int t)
```

  - Đọc từ tệp vào Trie

```
string * readInput(string filename, unsigned int &n)
```

  - Tạo menu

```
void menu(TrieNode *root)
```

  - Giao diện khi biên dịch file Dictionary.cpp:

```
*** Welcome to Dictionary Program ***

All funtion of my program are below:
1. Insert a word to Trie.
2. Looking up word.
3. Show the Trie Built.
4. Exit.
Your Selection:
```

Người dùng có thể chọn một trong bốn sự lựa chọn:

- Lựa chọn 1: Thêm một từ mới vào từ điển.
- Lựa chọn 2: Tìm kiếm từ.
- Lựa chọn 3: Hiển thị Trie của từ điển.
- Lựa chọn 4: Thoát chương trình.

- Input.txt: File chứa gồm 200 từ của từ điển.

TÀI LIỆU THAM KHẢO

- **Phần Coding**

https://www.geeksforgeeks.org/trie-insert-and-search/?fbclid=IwAR2P57utOGzWF7OcSf9N28WZRtMybajaRoy32DXbKiVNF8T51DH_yXzRpD8

https://www.geeksforgeeks.org/trie-delete/?fbclid=IwAR0exn0PyfatJbQ6gXQjrvqLM_mrRZgtSn_bIldThCtbcXwQgTEPIaZnFn0

- **Phần Report:**

Tìm hiểu về Trie nội dung, hình ảnh, ứng dụng:

https://en.wikipedia.org/wiki/Trie

https://vi.wikipedia.org/wiki/Trie

https://thuytrangcoding.wordpress.com/2018/02/11/string-trie/

https://medium.com/basecs/trying-to-understand-tries-3ec6bede0014?fbclid=IwAR0IOgmni4h7SlESq5Ofl3bmfRDDtbUczUD4vHU1KVUaI2egM0GIP1iwp9I

https://iq.opengenus.org/applications-of-trie/?fbclid=IwAR2IbdxqJWE3NzrhiIqL2GAUcaKg2P-5ZIcnQpwbItideZh_YjJPN1mOTMU

PHỤ LỤC