ĐẠI HỌC QUỐC GIA TP. HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KHOA HỌC MÁY TÍNH

BÁO CÁO ĐỒ ÁN
MÔN : CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT
ĐỀ TÀI
TRIE - PREFIX TREE

HỌC KÌ : II **NĂM HỌC : 2023 - 2024**

Giảng viên hướng dẫn : Nguyễn Thanh Sơn
Sinh viên thực hiện: Nguyễn Thiện Nhân MSSV:23521083
Lớp : IT003.O21.CTTN

TP. Hồ Chí Minh, tháng 5 năm 2024

MỤC LỤC

CHƯƠNG 1 : GIỚI THIỆU

1.1 Giới thiệu chung

Tiền tố là một chuỗi con bắt đầu của một xâu. Khi hai xâu giống nhau tất cả các tiền tố đều sẽ giống nhau. Nếu hai xâu khác nhau thì ta so sánh bằng cách tìm tiền tố chung dài nhất giống mà và rồi so sánh kí tự ngay sau chúng. Và Trie (hay cây tiền tố) là một cấu trúc dữ liệu giúp chúng ta dạng cây giúp chúng ta quản lý các tiền tố một cách dễ hiểu và dễ cài đặt.

1.2 Quá trình thực hiện

1.2.1 Tuần 1

- Tìm hiểu lên ý tưởng
- Đọc các nguồn tại liệu tham khảo
- Hoàn thiện sườn của đồ án

1.2.2 Tuần 2

- Tạo các đoạn code, hình ảnh để thêm vào đồ án
- Chỉnh sửa hoàn thiện đồ án

1.3 Đôi nét về lịch sử

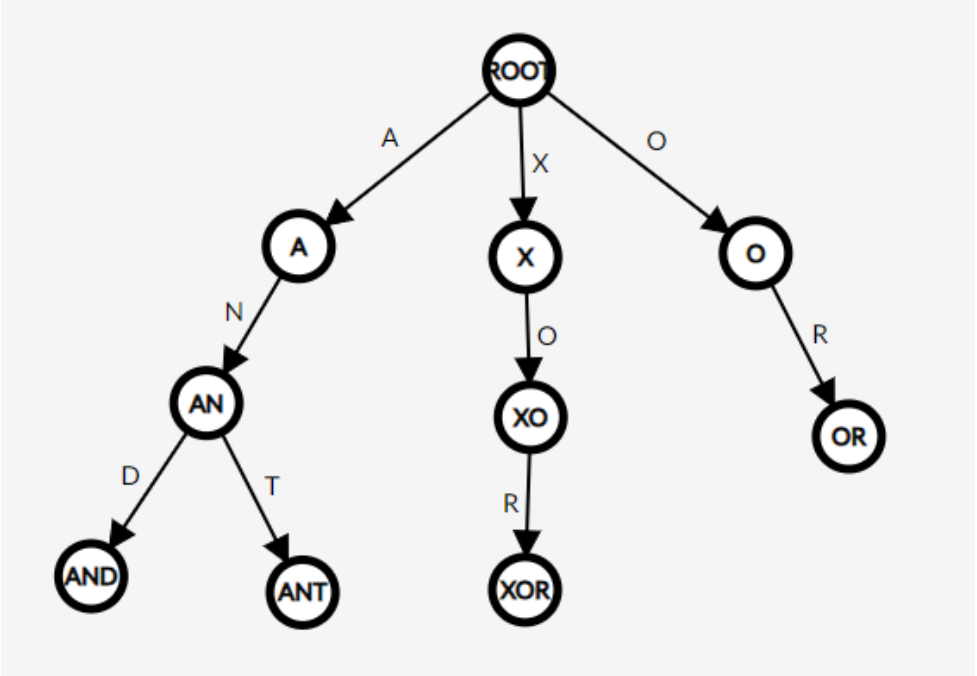
Theo Wikipedia thì ý tưởng về tập hợp các xâu được mô tả một cách trừu tượng lần đầu tiên vào năm 1912 bởi Axel Thue. Được thử nghiệm trên máy tính vào 1959 do René de la Briandais thực hiện. Và mô tả độc lập vào năm 1960 bởi tác giả Edward Fredkin, ông cũng là người đặt cho thuật toán này cái tên trie

CHƯƠNG 2: KẾT QUẢ ĐẠT ĐƯỢC

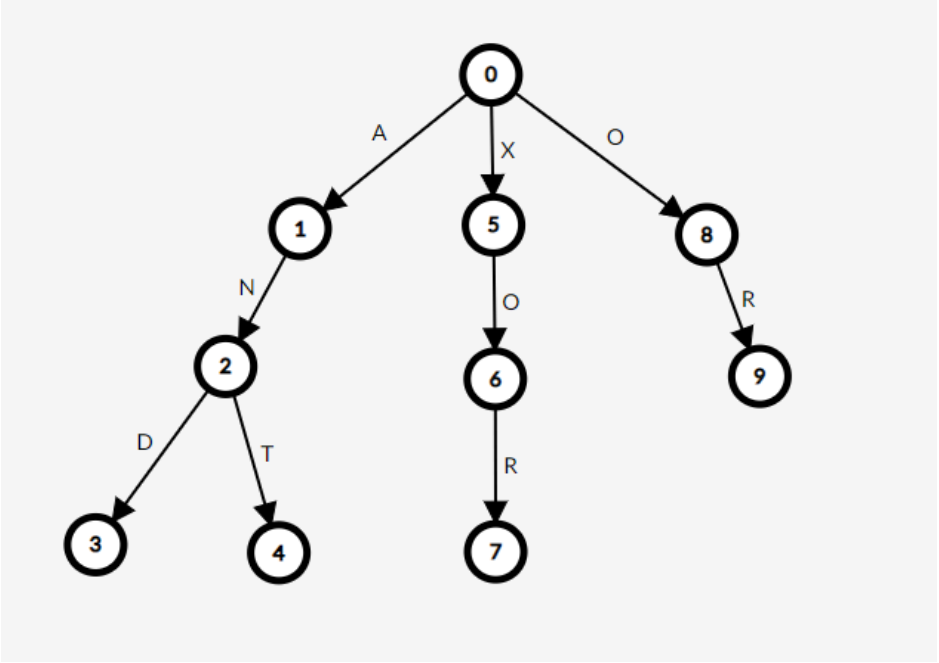
2.1 Ý tưởng

Ban đầu cây chỉ gồm một gốc. Khi ta thêm các xâu vào cây, từ mỗi kí tự của xâu ta sẽ thêm vào một cạnh đại diện cho nó và cũng thêm vào một đỉnh trên cây. Một xâu sẽ được biểu diễn bằng một đỉnh và đường đi từ gốc đến đỉnh đó.

Ví dụ :



Trie lưu 4 xâu: AND, ANT, XOR, OR



Trie biểu diễn 4 xâu : AND, ANT, XOR, OR (nhưng các nút được đánh số theo thứ tự dfs)

Trie xử dụng một cấu trúc chính là Child(u, c) = v. Đại diện cho hai đỉnh u và v được nối với nhau bởi một cạch có giá trị là c, hoặc v = -1 khi không tồn tại cạnh u, v. Hình trên thì child(2, D) = 3, child(4, A) = -1;

2.2 Các đặc trưng nổi bật

- Độ phức tạp tuyến tính giúp các thao tác tìm kiếm, xóa, thêm diễn ra hiệu quả
- Dễ dàng hiểu và cài đặt
- Không gian lưu trữ không quá lớn

2.3 Các thao tác cơ sở (sử dụng mảng)

2.3.1 Tổ chức dữ liệu

Do cấu trúc child(u, c) = v nên dễ thấy với mỗi nút u ta sẽ sử dụng một mảng child[26] để lưu các đường đi. Bên cạnh đó ta còn phải kiểm soát thêm 2 việc là có bao nhiêu xâu tiền tố đi đến nút u và có bao nhiêu xâu kết thúc ở nút u.

```
const int NUMBEROFNODES = ...; // Số lượng các nút thường sẽ ít hơn tổng các kí tự của các xâu
struct Trie{
    struct Node{
        int child[26]; // Lưu các cạnh từ nút đang xét
        int exist, cnt; // Lưu số lượng xâu kết thúc và lượng tiền tố ở nút đang xét
    } nodes[NUMBEROFNODES];

    int cur; // Hiện trong trie đang có bao nhiêu đỉnh
    Trie() : cur(0) { // Tạo nút gốc cho Trie là đỉnh 0 khi khởi tạo Trie
        memset(nodes[0].child, -1, sizeof(nodes[cur].child));
        nodes[0].exist = nodes[0].cnt = 0;
    };
    ...
}
```

Độ phức tạo bộ nhớ sẽ nhỏ hơn số lượng kí tự * 26

2.3.2 Thêm một xâu vào tập hợp

Để thêm một xâu vào tập hợp ta sẽ :

- Xuất phát từ đỉnh gốc đi lần lượt theo các cạnh ứng với từng kí tự của xâu đồng thời tăng biến đếm số lượng cnt các tiền tố ở các nút đi qua

- Phát hiện một cạnh chưa tồn tại : thực hiện thêm nút, thêm cạnh nối với nút vừa được tạo.
- Sau khi kết thúc xâu tăng biến exist kiểm soát số lượng xâu ở nút cuối cùng trong quá trình thăm.

```
int new_node() { // Tạo và trả về giá trị của đỉnh mới được tạo ra
    cur++; // Tăng biến đếm
    memset(nodes[cur].child, -1, sizeof(nodes[cur].child)); // Khởi tạo mảng cạnh
    nodes[cur].exist = nodes[cur].cnt = 0; // Khởi tạo biến đếm
    return cur; // Trả về đỉnh mới được tạo
}

void add_string(string s) { // Đi lần lượt trên Trie và cập nhật
    int pos = 0;
    for (auto f : s) {
        int c = f - 'a';
        if (nodes[pos].child[c] == -1) { // Nếu cạnh tương ứng chữ cái c
                                         // chưa tồn tại thì ta tạo ra đỉnh mới
            nodes[pos].child[c] = new_node(); // Lưu nối cạnh vào đỉnh vừa mới được tạo
        }
        pos = nodes[pos].child[c];
        nodes[pos].cnt++; // Có thêm một xâu trong trie có tiền tố
                          // là xâu được thể hiện bằng đỉnh hiện tại
    }
    nodes[pos].exist++; // Đã tìm được đỉnh tương ứng với xâu s,
                        // ta tăng biến exist của đỉnh lên 1
}
```

Hàm new_node() có độ phức tạp khoảnh O(1)

Hàm add_string() có độ phức tạp là O(n) với n là độ dài của xâu chúng ta thêm vào

2.3.3 Kiểm tra một xâu có nằm trong tập hợp đó hay không

Để có thể kiểm tra xem xâu có tồn tại trong trie hay không ta sẽ đi theo từng kí tự của xâu nếu :

- Không tồn tại đường đi .Trả về False
- Kết thúc tại một đỉnh có biến exist kiểm tra kết thúc bằng 0 thì cũng không tồn tại xâu. Trả về False
- Đi hết được xâu và đỉnh cuối cùng có giá trị exist > 0. Suy ra xâu có tồn tại trong tập hợp trả về True

```
bool find_string(string s) {
    int pos = 0;
    for (auto f : s) {  // Duyệt qua các đường đi trên một nút
        int c = f - 'a';
        if (nodes[pos].child[c] == -1) return false; // Không tồn tại đường đi
        pos = nodes[pos].child[c];
    }
    return (nodes[pos].exist != 0); // Kiểm tra có xâu nào
                                    // kết thúc tại đỉnh này hay không
}
```

Hàm fing_string() có độ phức tạp là O(n) với n là độ dài xâu

2.3.4 Xóa một xâu khỏi tập hợp

Các bước để xóa một xâu khỏi tập hợp như sau :

- Kiểm tra xem xâu có tồn tại trong trie hay không
- Đi trên trie lần lượt theo các kí tự của xâu, giảm biến đếm tiền tố cnt ở mỗi nút đi qua. Nếu một nút có biến cnt = 0 thì ta sẽ hiểu là nút đó không còn tồn tại và tiến hành xóa các cạnh đi đến nút đó.
- Ở đỉnh cuối cùng được thăm ta cũng giảm đi biến exist

```
bool delete_string_recursive(int pos, string& s, int i) { // Trả về liệu đỉnh pos
                                                          // có bị xóa đi hay không
    if (i != (int)s.size()) { // Nếu chưa đến đỉnh tương ứng với xâu s
                              // thì tiếp tục đệ quy xuống dưới
        int c = s[i] - 'a';
        bool isChildDeleted = delete_string_recursive(nodes[pos].child[c], s, i + 1);
        if (isChildDeleted) nodes[pos].child[c] = -1; // Nếu đỉnh con tương ứng bị xóa thì
                                                      // ta gán lại đỉnh tương ứng bằng -1
    }
    else nodes[pos].exist--; // Nếu đã đến đỉnh tương ứng với xâu s
                             // thì ta giảm biến exist của đỉnh đi 1

    if (pos != 0) { // Nếu đỉnh đang xét không phải gốc thì ta giảm biến cnt của đỉnh đi 1
                    // và kiểm tra đỉnh có bị xóa đi hay không
                    // Đỉnh bị xóa nếu không còn xâu nào đi qua nó, nói cách khác là
                    // không còn xâu nào có tiền tố là xâu được thể hiện bởi đỉnh pos
        nodes[pos].cnt--;
        if (nodes[pos].cnt == 0) return true;
    }
    return false;
}

void delete_string(string s) {
    if (find_string(s) == false) return; // Kiểm tra xâu s có trong
                                         // trie hay không
    delete_string_recursive(0, s, 0); // Gọi hàm đệ quy xóa xâu s khỏi trie
}
```

Hàm delete_string có độ phức tạo O(n)

Hàm delete_string_recursive có độ phức tạo là O(n) với n là độ dài xâu đang xét

2.3.4 Giới thiệu cài đặt bằng con trỏ

Ưu điểm :

- Linh động hơn trong việc khai báo bộ nhớ
- Như ta đã thấy ở cách cài đặt bằng mảng những đỉnh được xóa không thực sự bị xóa, mà chỉ xóa các cạnh đến đỉnh đó. Ở cách cài đặt bằng con trỏ ta có thể thực sự xóa những đỉnh này

Nhược điểm :

- Dễ gây nhầm lẫn nếu không nắm chắc kiến thức về con trỏ
- Tùy thuộc vào complier có thể gây chậm.

Xem code tại : Code tham khảo

2.4 Một số ứng dụng cơ bản

2.4.1 Xắp xếp các xâu

Nếu ta duyệt theo thứ tự DFS dễ thấy được ta sẽ thăm các xâu theo thứ tự tăng dần. Từ đó dễ dàng xắp xếp các xâu ở độ phức tạp tuyến tính.

2.4.2 Tìm tiền tố chung dài nhất của 2 xâu

Ta cần xác định 2 đỉnh cuối cùng của 2 xâu. Khi bài toán tìm tiền tố chung dài nhất sẽ trở thành bài toán tìm tổ tiên chung gần nhất (LCA).

2.4.3 Tìm xâu có thứ tự từ điển thứ k

Như đã nói ở phần 3.1.1 khi duyệt trên trie ta sẽ luôn duyệt các xâu theo thứ tự từ điển. Điều đó cho ta khả năng tìm kiếm xâu có thứ tự từ điển thứ K một cách nhanh chóng bằng cách liên tục duyệt và trừ đi K một lượng exist của nút khi K bằng 0 thì đường đi từ gốc đến nút ta đang đúng chính là xâu cần tìm.

2.5 Một số ứng dụng trong thực thế

2.5.1 Tự động hoàn thành và kiểm tra chính tả

Bằng cách lưu các từ trong từ điển vào trie. Khi người dùng nhập lần lượt các kí tự tạo ra tiền tố ta có thể đưa ra gợi ý cũng như kiểm tra chính tả cho người dùng. Được dùng tỏng các trình soạn thảo văn bản.

2.5.2 Sử dụng trong từ điển

Với trie thì việc tra cứu các từ bằng những chữ cái đầu sẽ trở nên nhanh chống và dễ dàng hơn

2.5.3 Sử dụng để đề xuất trong công cụ tìm kiếm

Trie có khả năng lưu được số lượng các tiền tố cũng như các xâu con, nên khi người dùng muốn tìm kiếm một thứ gì đó. Hệ thống có thể đề xuất được những phương án tìm kiếm phổ biến.

2.5.4 Nguyên cứu DNA và công nghệ sinh học

Trie được dùng để lưu trữ và phân tích DNA. Giúp các nhà khoa học có thể phân tích và xác định các mẫu của DNA một cách hiệu quả.

CHƯƠNG 3: PHỤ LỤC

3.1 Tài liệu tham khảo

Tham khảo tại các trang

- VNOI
- Wikipedia
- GeeksforGeeks

3.2 Một số đoạn code ví dụ

Xem tại : https://github.com/Ranisme/DoanDSA