TRIE

1. Mở đầu

Trong lập trình, việc tìm kiếm sự xuất hiện của một dãy ký tự (xâu ký tự) là một trong những bài toán thường gặp. Ví dụ, tìm kiếm nghĩa của một từ tiếng Anh (chẳng hạn "hello") trong một từ điển; tìm kiếm các trang web xuất hiện một dãy ký tự nào đó giống như google thực hiện....

Cách tiếp cận cổ điển là lần lượt so sánh các xâu ký tự trong từ điển với xâu ký tự mẫu. Nếu phát hiện ra xâu trong từ điển xuất hiện trong xâu mẫu thì đưa ra một kết quả trả lời. Dễ thấy cách tiếp cận như trên không hiệu quả về thời gian thực hiện chương trình đặc biệt khi số lượng các xâu trong từ điển lớn (thường khoảng 10^6 xâu ký tự khác nhau như trong từ điển tiếng Anh chẳng hạn)

Trong khoa học máy tính, có một cấu trúc dữ liệu đặc biệt hiệu quả để tìm kiếm sự xuất hiện của các xâu ký tự trong một từ điển. Đó là cấu trúc **trie** hay còn gọi là cấu trúc cây tiền tố (preffix tree).

Ngoài việc ứng dụng cho việc tìm kiếm xâu trong từ điển, trie còn có rất nhiều ứng dụng khác trong các bài toán số học khi ta coi một số nguyên như là một dãy ký tự nhị phân. Việc tìm kiếm một số trong trường hợp này đưa về việc tìm kiếm dãy nhị phân biểu diễn số đó.

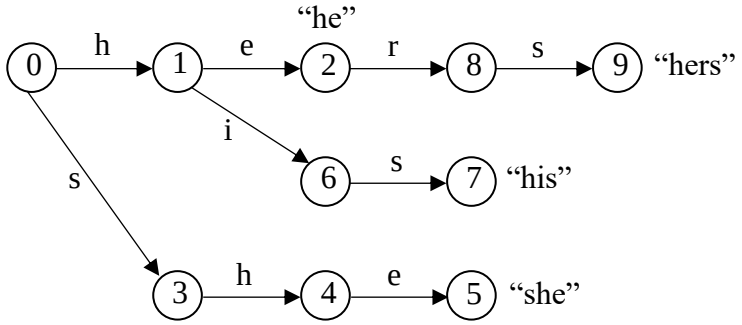
Sáng kiến kinh nghiệm này không có ý định trình bày chi tiết về cấu trúc trie cũng như các ứng dụng của nó vì đây là kiến thức chuyên sâu của bộ môn khoa học máy tính trong các trường đại học. Mục tiêu chính của sáng kiến này là tìm cách trình bày đơn giản nhất có thể về trie với thể hiện bằng ngôn ngữ C++ cũng như minh họa các ứng dụng trie qua một số bài tập tin học thường gặp.

Mặc dù vậy, có thể thấy đối tượng phục vụ chủ yếu của sáng kiến kinh nghiệm này là học sinh các trường THPT Chuyên trong cả nước. Đối với các thầy cô giảng dạy tin học ở bậc phổ thông, tôi cũng hy vọng nó sẽ là một tài liệu tham khảo bổ ích.

2. Trie và những mô tả cơ bản

Để mô tả về trie, cách đơn giản nhất là minh họa qua ví dụ.

Ví dụ trie với các xâu mẫu là “he”, “she”, “his”, “hers” như sau:



Như vậy trie có thể hình dung như là một cây định hướng từ một nút gốc. Mỗi cạnh của cây là một chữ cái và tập hợp các chữ cái trên một đường đi từ gốc đến một nút nào đó tạo thành tiền tố (preffix) của một từ trong từ điển. Việc tìm kiếm một từ tương đương với việc xây dựng một đường đi từ nút gốc đến một nút nào đó.

Mỗi một nút như vậy cần có các thông tin như sau:

- Địa chỉ của nút cha
- Địa chỉ của các nút con tương ứng với bảng chữ cái. Ví dụ, nếu các xâu ký tự chỉ bao gồm các chữ cái tiếng Anh in thường thì mỗi nút sẽ có 26 nút con (đánh số từ 0 đến 25 tương ứng với các chữ cái theo từ điển từ 'a' đến 'z'); nếu xâu ký tự chỉ gồm ký tự '0' và '1' thì mỗi nút sẽ có 2 nút con thứ tự 0 (ứng với ký tự '0')
- Các thông tin khác của nút (kiểm tra xem tại nút đó có từ không, giải nghĩa từ, số lượng từ đi qua nút....)

Việc mô tả trie trong C++ khá đơn giản bằng cách sử dụng STL vector:

2.1 Khai báo thông tin về một nút

```
const int K=26        // Kích thước bảng chữ cái

struct Tnode{
    int next[K];           // Liên kết đến nút con
    int p;                 // Liên kết đến cha
    int stop;              // Số lượng từ kết thúc tại nút
    int cnt;               // Số lượng từ đi qua nút

    // Hàm chạy khi một nút được cấp phát
    Tnode(int p=-1) : p(p) {
        fill(begin(next),end(next),-1);
    }
};
```

2.2 Khởi tạo cây

```
vector<Tnode> trie(1);
```

3. Các hàm thông dụng trên trie

3.1 Thêm một từ trong từ điển vào cây

Bắt đầu từ đỉnh gốc và đi theo nhánh con. Nếu nút tương ứng với nhánh con chưa có thì thêm nút này vào cây. Tiếp theo là đi dọc theo nhánh con này đến nút tiếp theo. Khi đến nút cuối cùng thì ghi nhận nút này có một từ kết thúc tại đó và tăng số lượn từ đi qua nút này lên 1:

```
// Thêm một từ vào cây
```

```
void AddWord(char* s) {
    int v=0;
    for(int i=0;s[i];++i) {
        int k=s[i]-'a';
        if (trie[v].next[k]==-1) {
            // Thêm một nút mới vào cây. Tạo liên kết với cha nó
            trie[v].next[k]=trie.size();
            trie.emplace_back(v);
        }
        v=trie[v].next[k];
    }
    trie[v].stop++;
    // Cập nhật các từ đi qua trên tất cả các nút tổ tiên
    while (v!=-1) ++trie[v].cnt, v=trie[v].p;
}
```

3.2 Đếm xem trong từ điển có bao nhiêu từ bằng s cho trước.

Kỹ thuật cũng tương tự như phần 3.1 tuy nhiên ở đây khi gặp một nhánh không đi được nữa thì dừng ngay (vì không có từ nào khớp):

```
int CountWord(char* s) {
    int v=0;
    for(int i=0; s[i]; ++i) {
        int k=s[i]-'a';
        if (trie[v].next[k]==-1) return 0;
        v=trie[v].next[k];
    }
    return trie[v].stop;
}
```

3.3 Đếm xem có bao nhiêu từ trong từ điển có tiền tố bằng s

Thuật toán được thực hiện hoàn toàn giống 3.2, tuy nhiên giá trị trả về là trường thông tin cnt (số lượng từ đi qua nút) thay vì trường stop (số lượng từ dừng tại nút)

```
int CountWord(char* s) {
    int v=0;
    for(int i=0; s[i]; ++i) {
        int k=s[i]-'a';
        if (trie[v].next[k]==-1) return 0;
        v=trie[v].next[k];
    }
    return trie[v].stop;
}
```

3.4 Đếm xem trong từ điển có bao nhiêu từ nhỏ hơn hoặc bằng s

Thuật toán cũng là đi dọc từ gốc theo các ký tự tương ứng. Chú ý rằng khi ta di chuyển theo nút k thì tất cả các từ đi qua các nút 0, 1, ..., $k-1$ đều nhỏ hơn từ đang xét.

```
int CountLE(char* s) {
    int v=0, res=0;
    for(int i=0; s[i]; ++i) {
        int k=s[i]-'a';
        for(int l=0;l<k;++l)
        res += trie[trie[v].next[l]].cnt;
        if (trie[v].next[k]==-1) return res;
        v=trie[v].next[k];
    }
    return res+trie[v].cnt;
}
```