LECTURE 17

TRIE

Phạm Nguyễn Sơn Tùng

Email: sontungtn@gmail.com

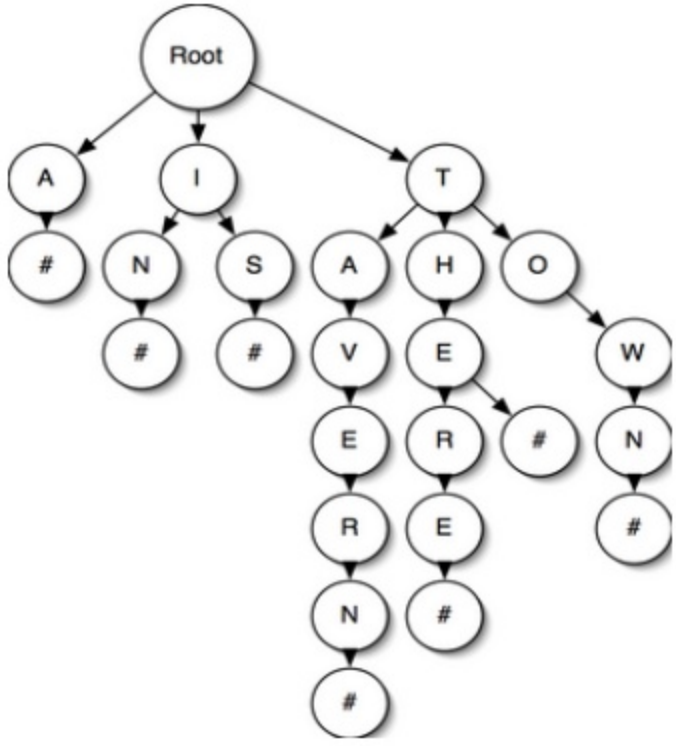
Định nghĩa Trie

Trie là một cấu trúc dữ liệu dùng để lưu danh sách các chuỗi trên cây. Trie là một cây nhưng không phải là cây nhị phân, nó là cấu trúc cây bình thường nhưng với việc lưu các chuỗi theo dạng cấu trúc cây sẽ giúp cho việc "thêm" (add/insert) "xóa" (delete/remove) và "tìm kiếm" (search/find) trở nên rất thuận tiện.

Cấu trúc của cây Trie

Một cây Trie gồm **2 thành phần**:

- Một node chứa ký tự. Node này có thể kết nối với một node khác.
- Một biến "cờ" là số nguyên. Nếu số nguyên **khác 0** nghĩa là node đó là kết thúc của một từ.



Source Code minh họa


```cpp
#include <string>
#include <iostream>
using namespace std;
#define MAX 26
struct node
{
    struct node *child[MAX];
    int countLeaf;
};
```

```python
class Node:
    countLeaf = 0
    child = dict()
    def __init__(self):
        self.countLeaf = 0
        self.child = dict()
```

Source Code minh họa - Java

```java
class Node {
    static final int MAX = 26;
    public Node[] child;
    public int countLeaf;
    public Node() {
        countLeaf = 0;
        child = new Node[MAX];
    }
}
```


```java
class Trie {
    public static final int MAX = 26;
    private Node root;
    public Trie() {
        root = new Node();
    }
```

Bài toán Trie giải quyết

- **Thêm** một chuỗi vào cây.
- **Tìm kiếm** một chuỗi có tồn tại trong cây hay không.
- **Xóa** một chuỗi khỏi cây.

Tuy nhiên khóa trong **Trie** không nhất thiết phải là xâu kí tự mà có thể là một danh sách có thứ tự của bất kì đối tượng nào, chẳng hạn như chữ số, hay hình dạng.

Thêm chuỗi vào cây

1. Lần lượt thêm từng ký tự vào cây, khi ký tự được thêm vào, đại diện ký tự là một con số (a → 0, b → 1, c → 2 …)
2. Mỗi ký tự của 1 từ lần lượt sẽ nằm ở 1 tầng khác nhau của cây.
3. Kết thúc ký tự ta sẽ tăng giá trị nút lá lên 1.

- Độ phức tạp hàm thêm chuỗi là **O(string_length).**
- Bộ nhớ yêu cầu **O(Alphabet_size * string_length * N)**

  (Với N là số lượng chuỗi trong cây)

Source Code minh họa C++

```
struct node *newNode()
{
    struct node *Node = new struct node;
    Node->countLeaf = 0;
    for (int i = 0; i < MAX; i++)
    {
        Node->child[i] = NULL;
    }
    return Node;
}
```

Source Code minh họa C++

```
void addWord(struct node *root, string s)
{
    int ch;
    struct node *temp = root;
    for (int i = 0; i < s.size(); i++)
    {
        ch = s[i] - 'a';
        if (temp->child[ch] == NULL)
        {
            temp->child[ch] = newNode();
        }
        temp = temp->child[ch];
    }
    temp->countLeaf++;
}
```

Source Code minh họa python

```python
def addWord(root, s):
    tmp = root
    for ch in s:
        if ch in tmp.child:
            tmp = tmp.child[ch]
        else:
            tmp.child[ch] = Node()
            tmp = tmp.child[ch]
    tmp.countLeaf += 1
```

Source Code minh họa Java

```java
public void addWord(String s) {
    int ch;
    Node temp = root;
    for (int i = 0; i < s.length(); i++) {
        ch = s.charAt(i) - 'a';
        if (temp.child[ch] == null) {
            Node x = new Node();
            temp.child[ch] = x;
        }
        temp = temp.child[ch];
    }
    temp.countLeaf++;
}
```

Tìm kiếm chuỗi trong cây

1. Lần lượt chuyển từng ký tự thành số (chuyển giống hàm thêm chuỗi).
   - Nếu ký tự này tồn tại trong cây thì tìm kiếm xuống tầng tiếp theo.
   - Nếu không tìm thấy trả về false thoát khỏi hàm.
2. Sau khi tất cả ký tự duyệt qua, cần kiểm tra nút lá có khác 0 hay không? Nếu khác 0 nghĩa nghĩa là tìm được.

- Độ phức tạp hàm thêm chuỗi là **O(string_length).**
- Bộ nhớ yêu cầu **O(Alphabet_size * string_length * N)**

  (Với N là số lượng chuỗi trong cây)

Source Code minh họa C++

```
bool findWord(struct node *root, string s)
{
    int ch;
    struct node *temp = root;
    for (int i = 0; i < s.size(); i++)
    {
        ch = s[i] - 'a';
        if (temp->child[ch] == NULL)
        {
            return false;
        }
        temp = temp->child[ch];
    }
    return temp->countLeaf > 0;
}
```

Source Code minh họa python

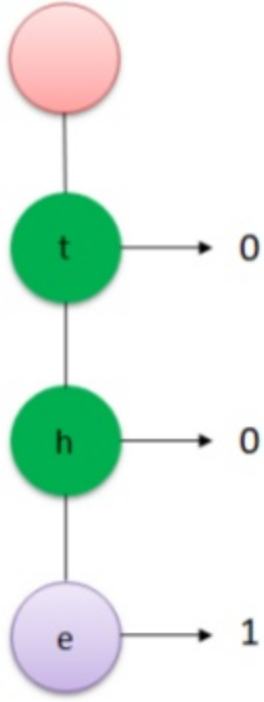
```python
def findWord(root, s):
    tmp = root
    for ch in s:
        if ch not in tmp.child:
            return False
        tmp = tmp.child[ch]
    return tmp.countLeaf > 0
```

Source Code minh họa Java

```java
public boolean findWord(String s) {
    int ch;
    Node temp = root;
    for (int i = 0; i < s.length(); i++)
    {
        ch = s.charAt(i) - 'a';
        if (temp.child[ch] == null)
        {
            return false;
        }
        temp = temp.child[ch];
    }
    return temp.countLeaf > 0;
}
```

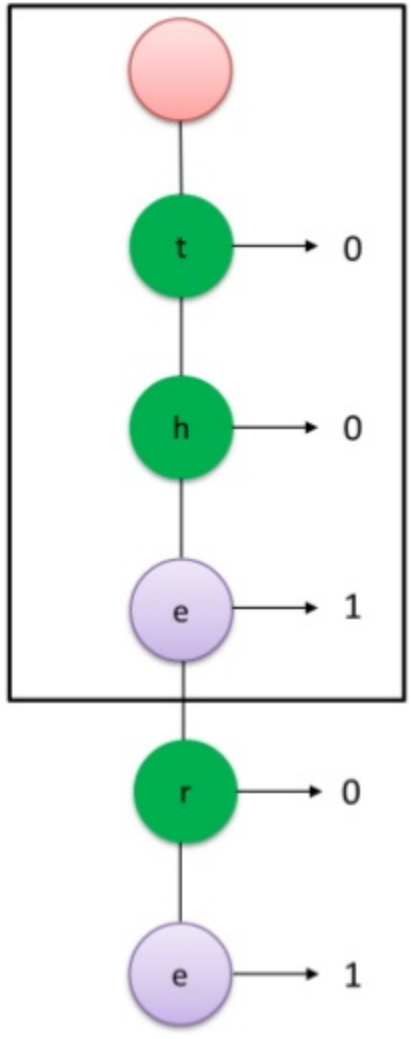
Xóa từ trong cây

Trường hợp 1: Xóa từ khi từ đó là từ độc lập, không ràng buộc bởi các từ khác.



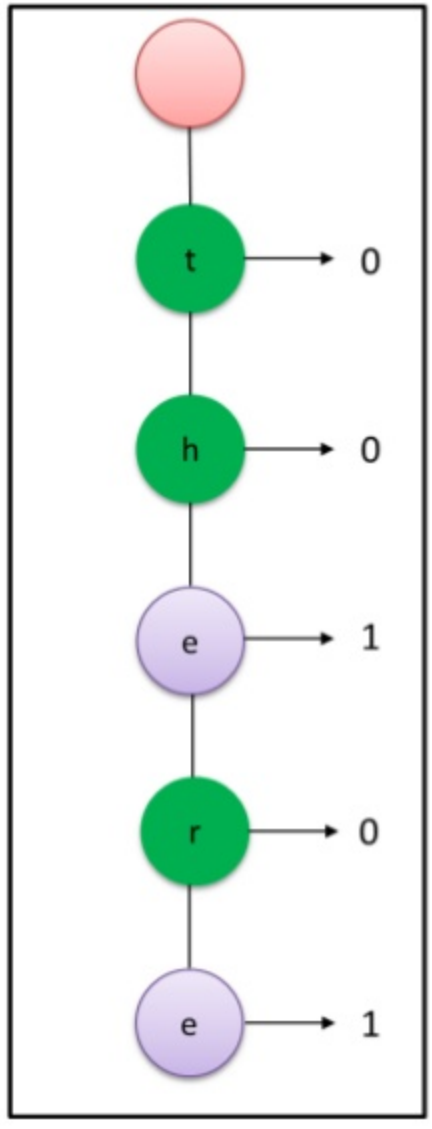
Xóa từ trong cây

Trường hợp 2: Xóa chuỗi, khi chuỗi là tiền tố của một chuỗi khác.



Xóa từ trong cây

Trường hợp 3: Xóa chuỗi, khi chuỗi chứa chuỗi khác.



Source Code minh họa C++

```
bool isWord(struct node *pNode)
{
    return (pNode->countLeaf != 0);
}
```


```
bool isEmpty(struct node *pNode)
{
    for (int i = 0; i < MAX; i++)
    {
        if (pNode->child[i] != NULL)
        {
            return false;
        }
    }
    return true;
}
```

Source Code minh họa C++

Xóa từ (part 1).

```
bool removeWord(struct node *root, string s, int level, int len)
{
    if (!root)
        return false;
    int ch = s[level] - 'a';
    if (level == len)
    {
        if (root->countLeaf > 0)
        {
            root->countLeaf--;
            return true;
        }
        return false;
    }
```

Source Code minh họa C++

Xóa từ (part 2).

```
    int flag = removeWord(root->child[ch], s, level + 1, len);
    if (flag && !isWord(root->child[ch]) && isEmpty(root->child[ch]))
    {
        delete root->child[ch];
        root->child[ch] = NULL;
    }
    return flag;
}
```

Source Code minh họa python

```python
def isWord(node):
    return node.countLeaf != 0
```


```python
def isEmpty(node):
    return len(node.child) == 0
```

Source Code minh họa python

```
def removeWord(root, s, level, len):
    if root == None:
        return False
    if level == len:
        if root.countLeaf > 0:
            root.countLeaf -= 1
            return True
        return False
    ch = s[level]
    flag = removeWord(root.child[ch], s, level + 1, len)
    if flag == True and isWord(root.child[ch]) == False and
isEmpty(root.child[ch]) == True:
        tmp = root.child[ch]
        del tmp
        del root.child[ch]
    return flag
```

Source Code minh họa Java

```java
private boolean isWord(Node pNode) {
    return (pNode.countLeaf != 0);
}
```


```java
private boolean isEmpty(Node pNode) {
    for (int i = 0; i < MAX; i++)
        if (pNode.child[i] != null)
            return false;
    return true;
}
```

Source Code minh họa Java

Part 1

```java
public boolean removeWord(String s) {
    return removeWord(root, s, 0, s.length());
}

private boolean removeWord(Node root, String s, int level, int len) {
    if (root == null)
        return false;
    if (level == len) {
        if (root.countLeaf > 0) {
            root.countLeaf--;
            return true;
        }
        return false;
    }
```

Source Code minh họa Java

Part 2

```
    int ch = s.charAt(level) - 'a';
    boolean flag = removeWord(root.child[ch], s, level + 1, len);
    if (flag && !isWord(root.child[ch]) && isEmpty(root.child[ch]))
    {
        root.child[ch] = null;
    }
    return flag;
}
```

Hỏi đáp