Big-O Blue

LECTURE 14

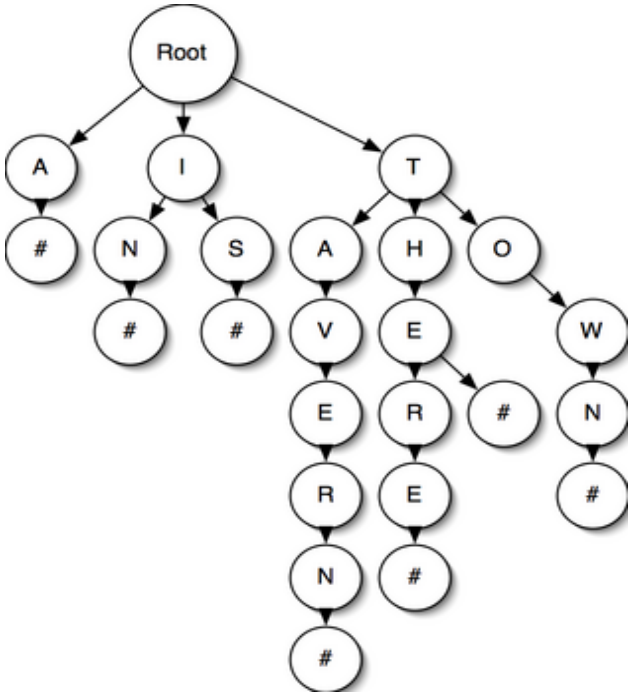
TRIE

Phạm Nguyễn Sơn Tùng

Email: sontungtn@gmail.com

Cây Trie

Trie (Cây tiền tố) là một cấu trúc dữ liệu dạng cây dùng để lưu danh sách các từ 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 từ theo dạng cấu trúc cây tiền tố sẽ giúp cho việc **thêm** (**add/insert**), **xóa** (**delete/remove**) và **tìm kiếm** (**search/find**) trở nên hiệu quả hơn.



Cấu trúc cây Trie

Một cây Trie gồm nhiều node, mỗi node gồm **2 thành phần**:

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


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


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

Cấu trúc cây Trie

```
1.  class Node {
2.      static final int MAX = 26;
3.      public Node[] child;
4.      public int countWord;
5.      public Node() {
6.          countWord = 0;
7.          child = new Node[MAX];
8.      }
9.  }
```


```
10. class Trie {
11.     public static final int MAX = 26;
12.     private Node root;
13.     public Trie() {
14.         root = new Node();
15.     }
16. }
```

Thao tác trên cây Trie

1. **Thêm** một từ vào cây.
2. **Tìm kiếm** một từ trong cây.
3. **Xóa** một từ khỏi cây.

Độ phức tạp: **O(string_length)**

Lưu ý:

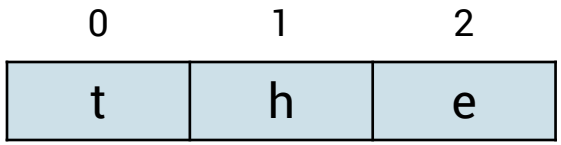
1. Độ phức tạp có thể tăng lên nếu dung cấu trúc dữ liệu đơn giản.

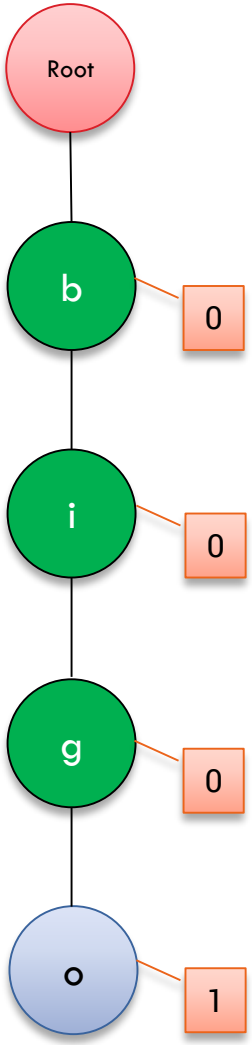
*2. Khóa trong **Trie** không nhất thiết phải là chuỗi mà có thể là một danh sách có thứ tự của bất kì đối tượng nào, chẳng hạn như chữ số, hình dạng hoặc một dạng nào đó khác.*

1. Thêm từ vào cây

Trường hợp 1: Từ cần thêm lần đầu xuất hiện trong cây Trie.

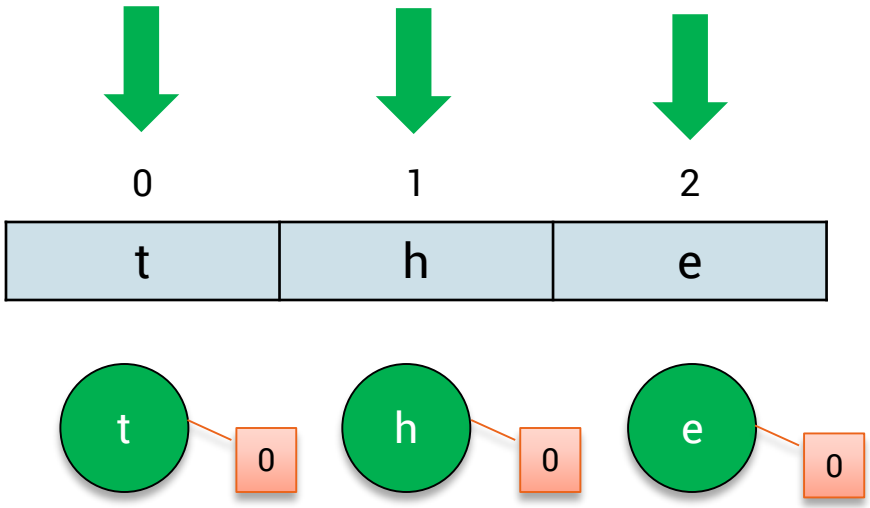
Từ cần thêm vào là “the”:

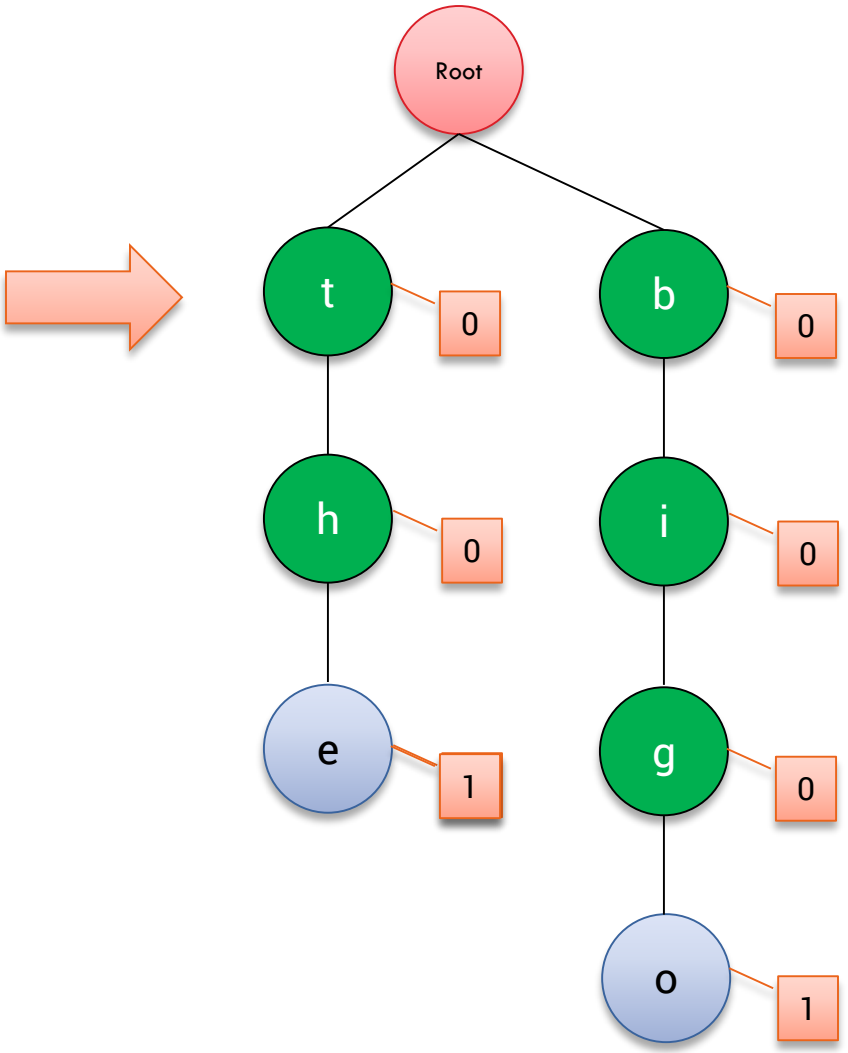




1. Thêm từ vào cây

Trường hợp 1: Từ cần thêm lần đầu xuất hiện trong cây Trie.

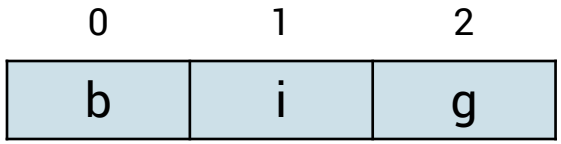


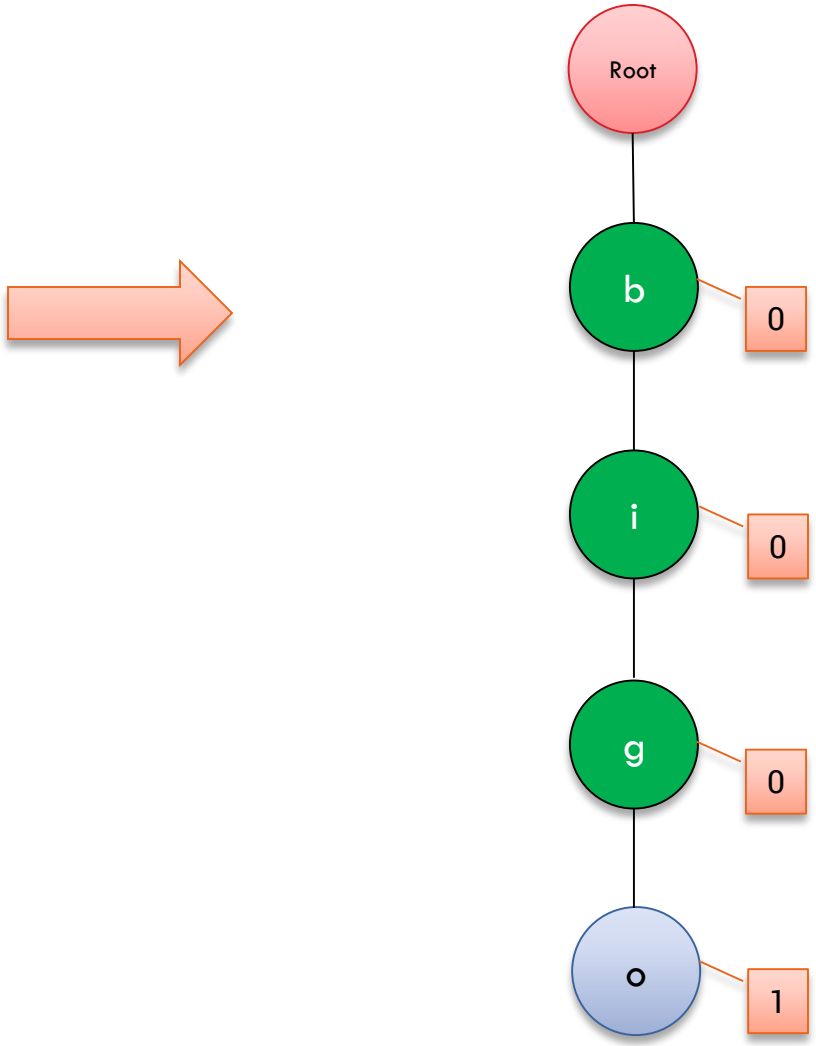


1. Thêm từ vào cây

Trường hợp 2: Từ cần thêm là tiền tố của chuỗi khác trong cây Trie.

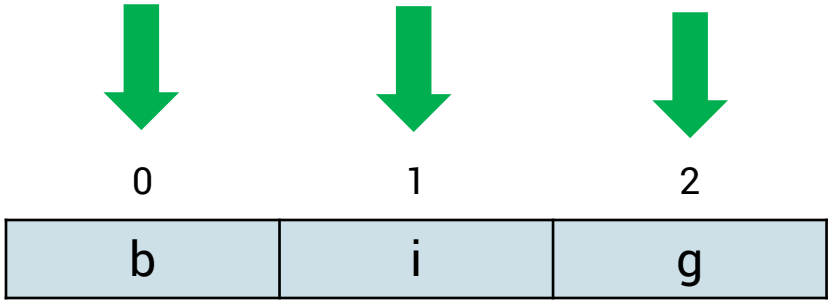
Từ cần thêm vào là “big”:

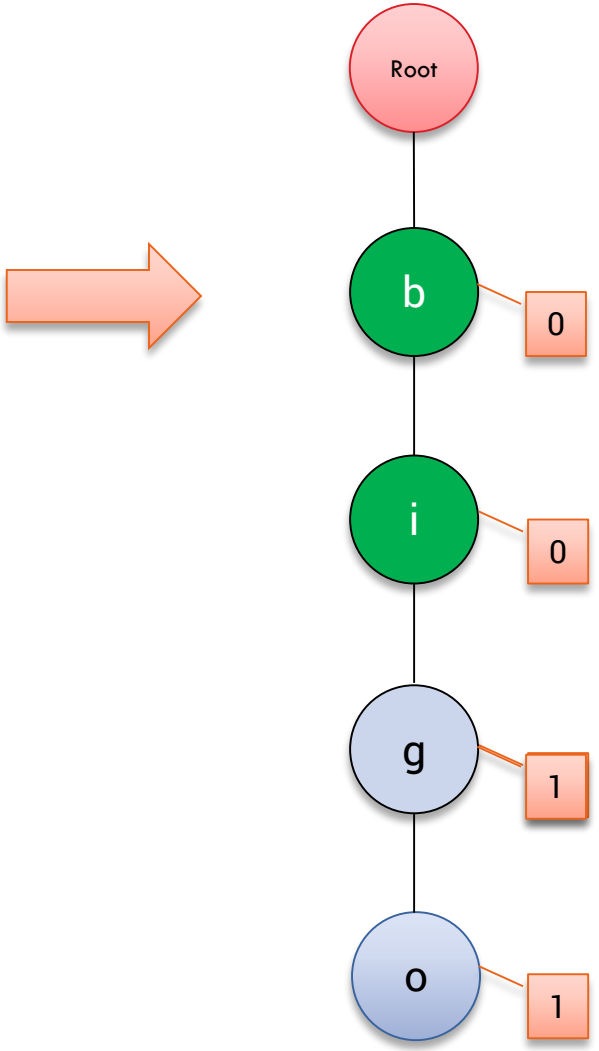




1. Thêm từ vào cây

Trường hợp 2: Từ cần thêm là tiền tố của chuỗi khác trong cây Trie.



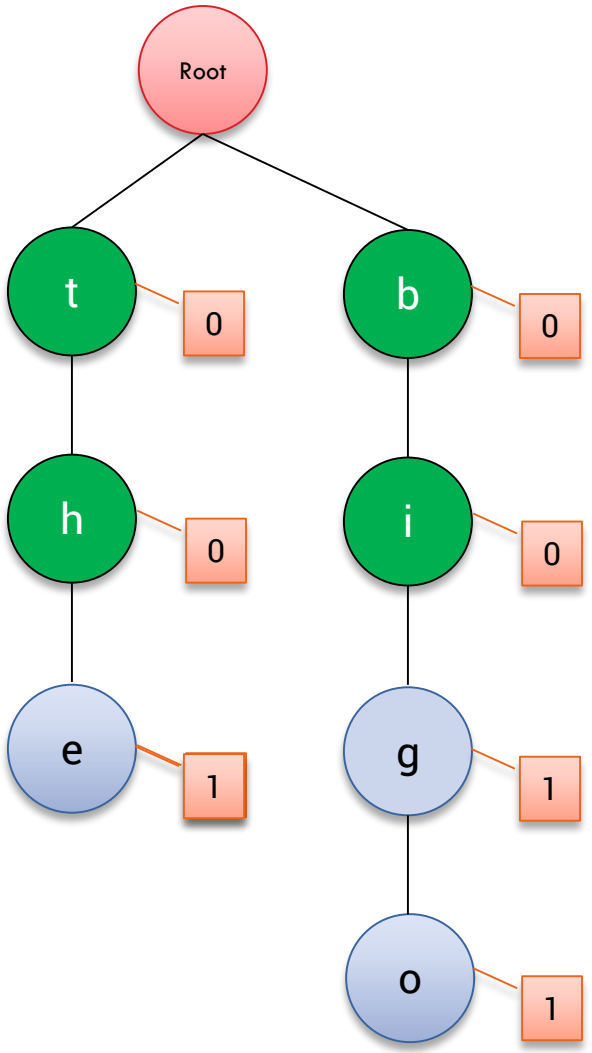


1. Thêm từ vào cây

Trường hợp 3: Từ cần thêm đã có sẵn tiền tố của chuỗi khác trong cây Trie.

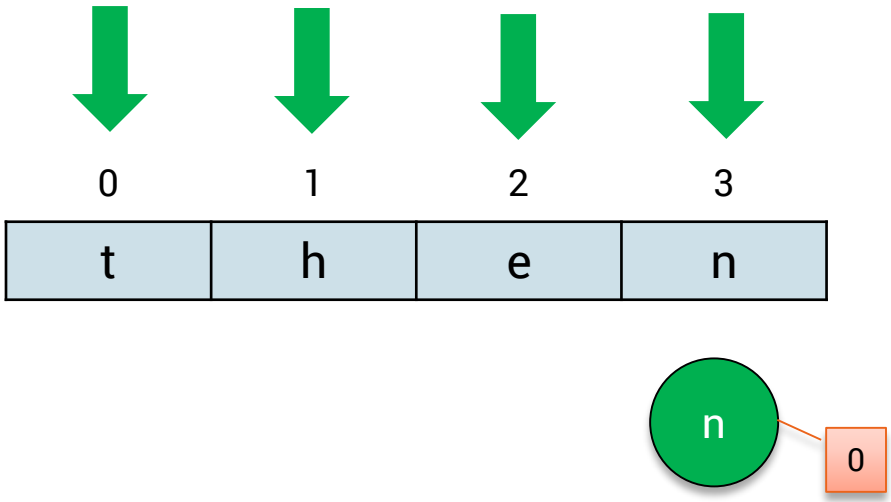
Từ cần thêm vào là “then”:

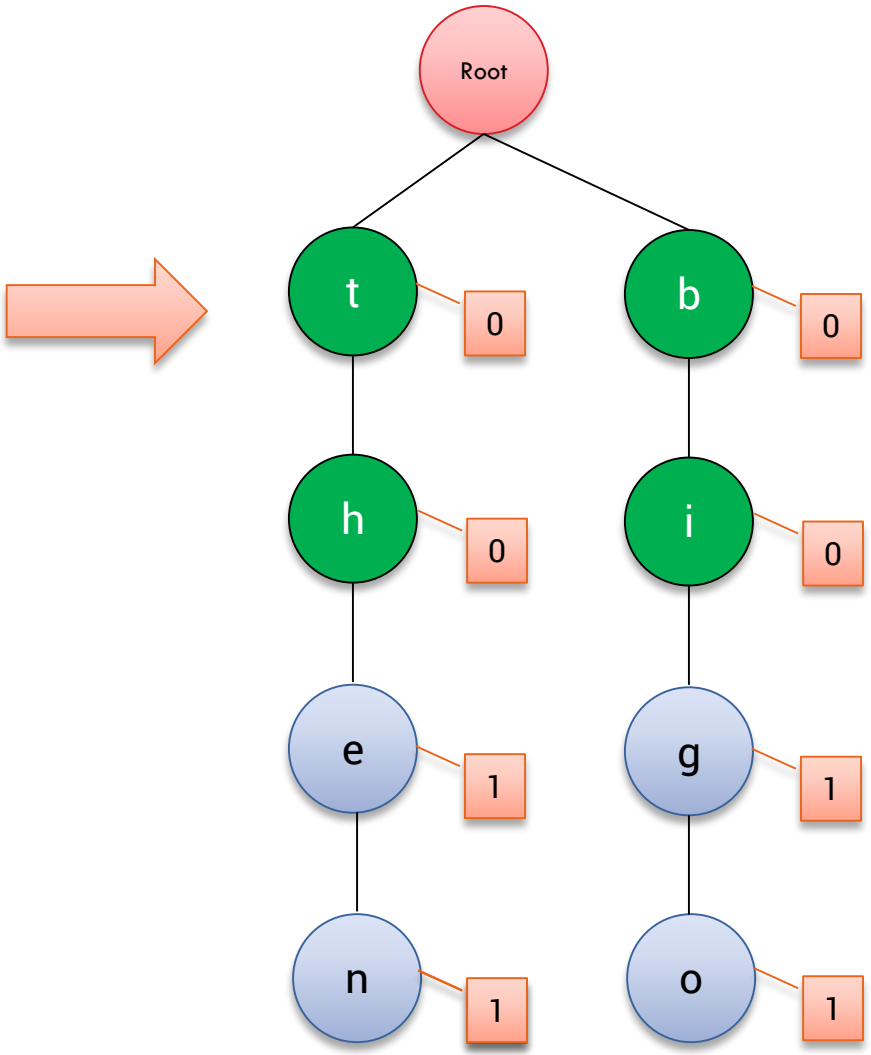
0	1	2	3
t	h	e	n



1. Thêm từ vào cây

Trường hợp 3: Từ cần thêm đã có sẵn tiền tố của chuỗi khác trong cây Trie.





Source Code minh họa

```
1.  struct Node *newNode()
2.  {
3.      struct Node *node = new Node;
4.      node->countWord = 0;
5.      for (int i = 0; i < MAX; i++)
6.      {
7.          node->child[i] = NULL;
8.      }
9.      return node;
10. }
```

Source Code minh họa

```
11. void addWord(struct Node *root, string s)
12. {
13.     int ch;
14.     struct Node *temp = root;
15.     for (int i = 0; i < s.size(); i++)
16.     {
17.         ch = s[i] - 'a';
18.         if (temp->child[ch] == NULL)
19.         {
20.             temp->child[ch] = newNode();
21.         }
22.         temp = temp->child[ch];
23.     }
24.     temp->countWord++;
25. }
```

Source Code minh họa

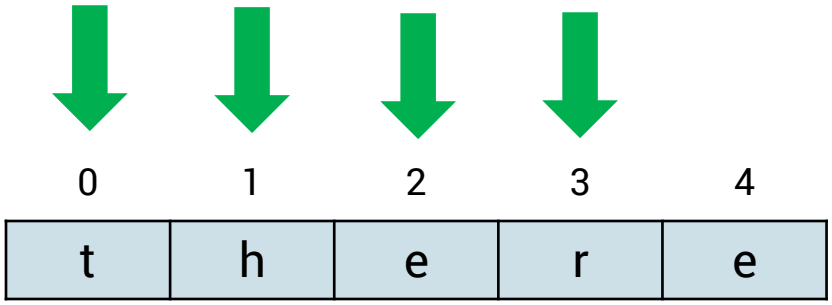
```
1. def addWord(root, s):
2.     temp = root
3.     for ch in s:
4.         if ch not in temp.child:
5.             temp.child[ch] = Node()
6.         temp = temp.child[ch]
7.     temp.countWord += 1
```

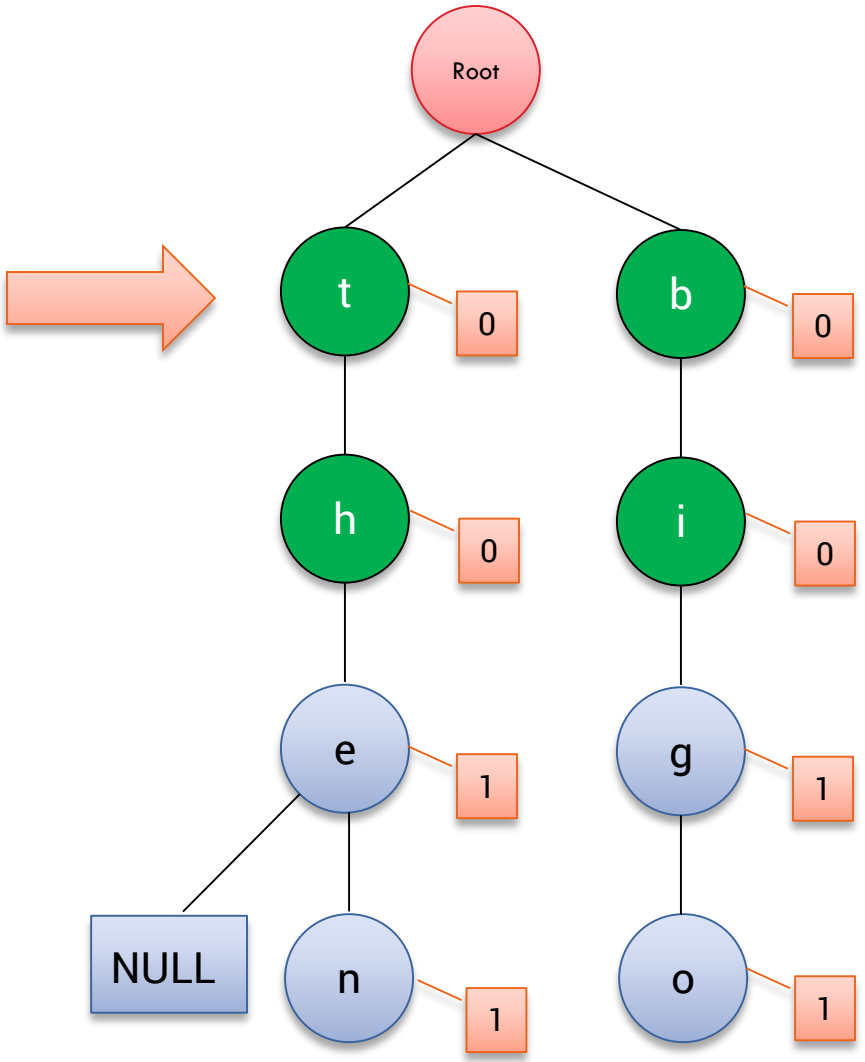
Source Code minh họa

```
1.  public void addWord(String s) {
2.      int ch;
3.      Node temp = root;
4.      for (int i = 0; i < s.length(); i++) {
5.          ch = s.charAt(i) - 'a';
6.          if (temp.child[ch] == null) {
7.              Node x = new Node();
8.              temp.child[ch] = x;
9.          }
10.         temp = temp.child[ch];
11.     }
12.     temp.countWord++;
13. }
```

2. Tìm kiếm từ trong cây

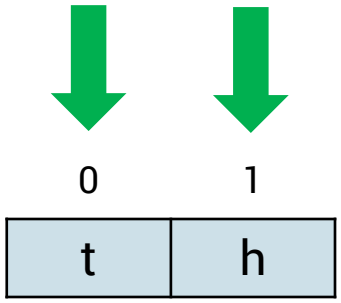
Trường hợp 1: Không tồn tại từ cần tìm trong cây.

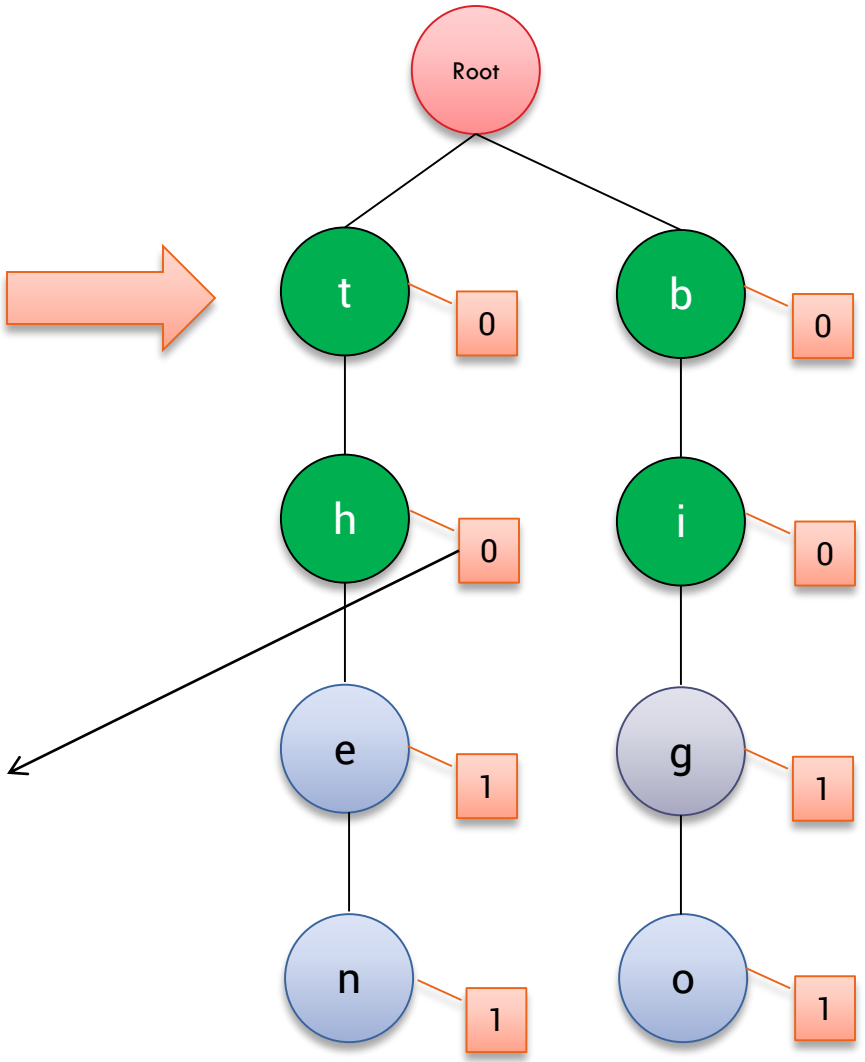




2. Tìm kiếm từ trong cây

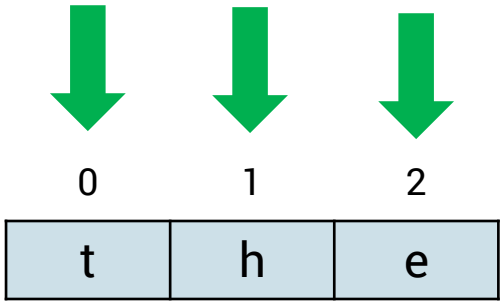
Trường hợp 2: Từ cần tìm tồn tại nhưng không phải là một từ trong cây (chỉ đóng vai trò là tiền tố của từ khác trong cây).

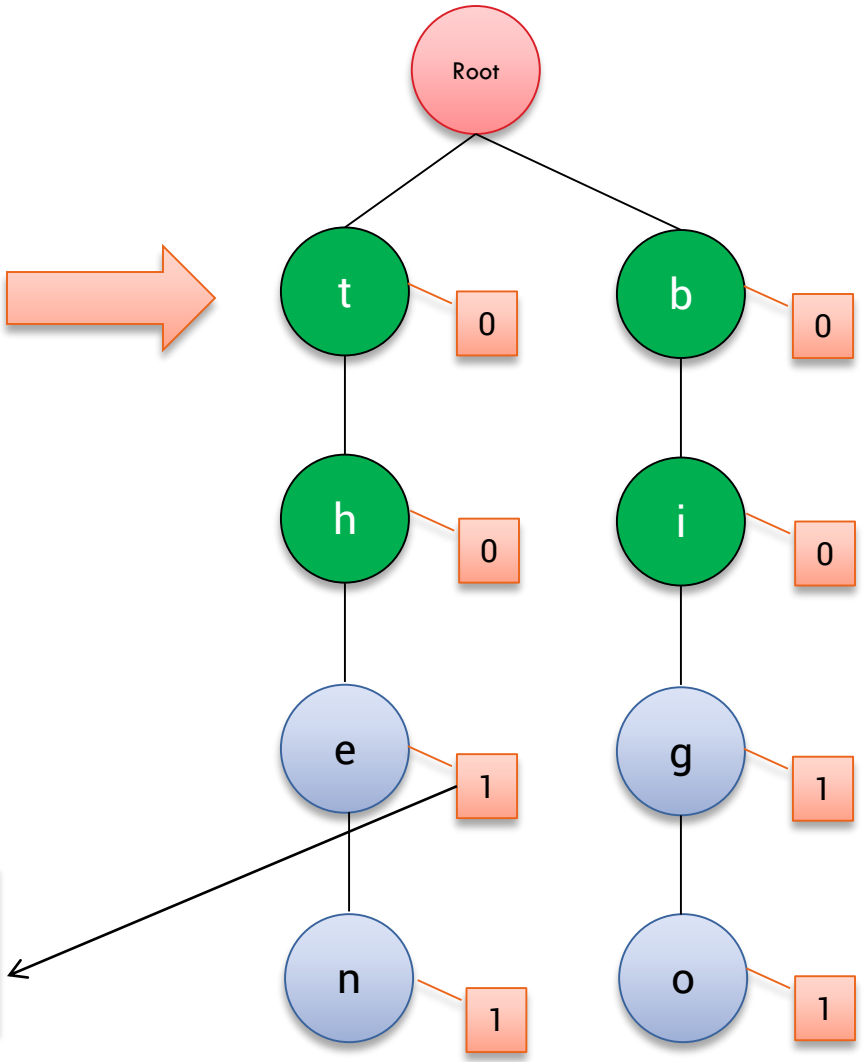




2. Tìm kiếm từ trong cây

Trường hợp 3: Tìm thấy từ cần tìm trong cây.





khác 0 (ký tự kết thúc của ít nhất một từ trong cây)

Source Code minh họa

```cpp
bool findWord(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->countWord > 0;
}
```

Source Code minh họa python

```
1. def findWord(root, s):
2.     temp = root
3.     for ch in s:
4.         if ch not in temp.child:
5.             return False
6.         temp = temp.child[ch]
7.     return temp.countWord > 0
```

Source Code minh họa Java

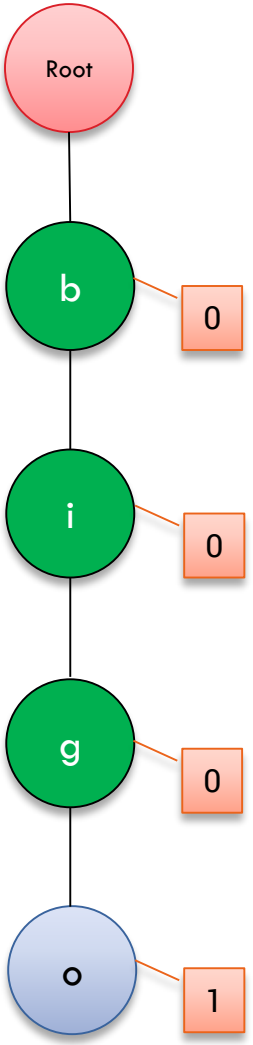
```java
public boolean findWord(String s) {
    int ch;
    Node temp = root;
    for (int i = 0; i < s.length(); i++) {
        ch = s.charAt(i) - 'a';
        if (temp.child[ch] == null) {
            return false;
        }
        temp = temp.child[ch];
    }
    return temp.countWord > 0;
}
```

3. Xóa từ trong cây

Trường hợp 1: Từ cần xóa là từ độc lập, khi xóa không làm ảnh hưởng đến các từ khác trong cây.

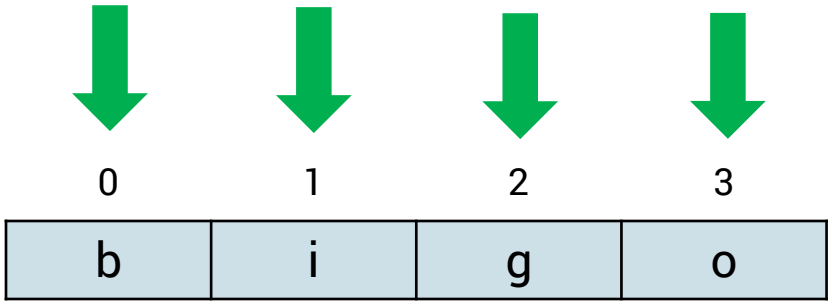
Từ cần xóa:

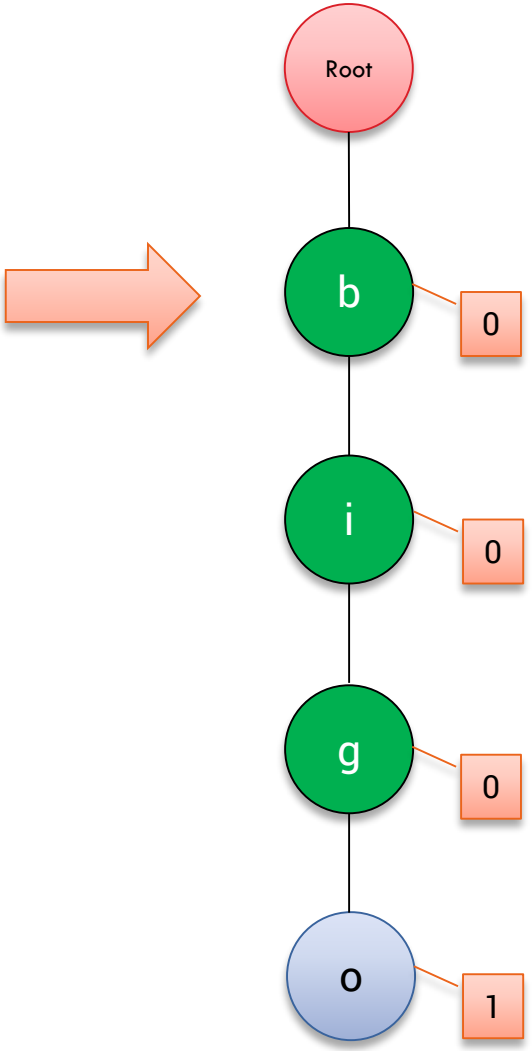
0	1	2	3
b	i	g	o



3. Xóa từ trong cây

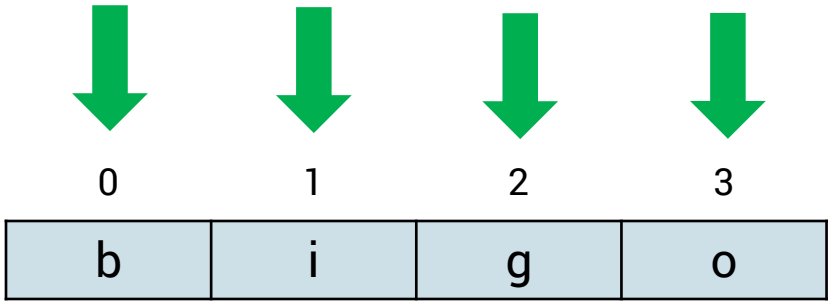
Bước 1: Duyệt từ đầu đến cuối từ cần xóa để xác định từ có tồn tại trong cây hay không.



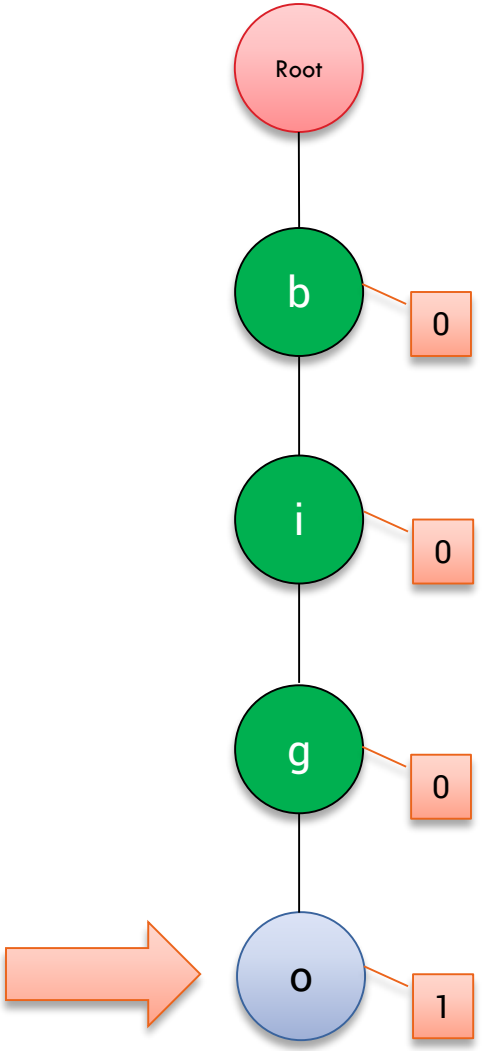


3. Xóa từ trong cây

Bước 2: Xét từng node một xem có thỏa điều kiện để xóa hay không.



- Node đó có phải kết thúc của một từ nào đó hay không?
- Node đó có là tiền tố của node nào khác hay không?

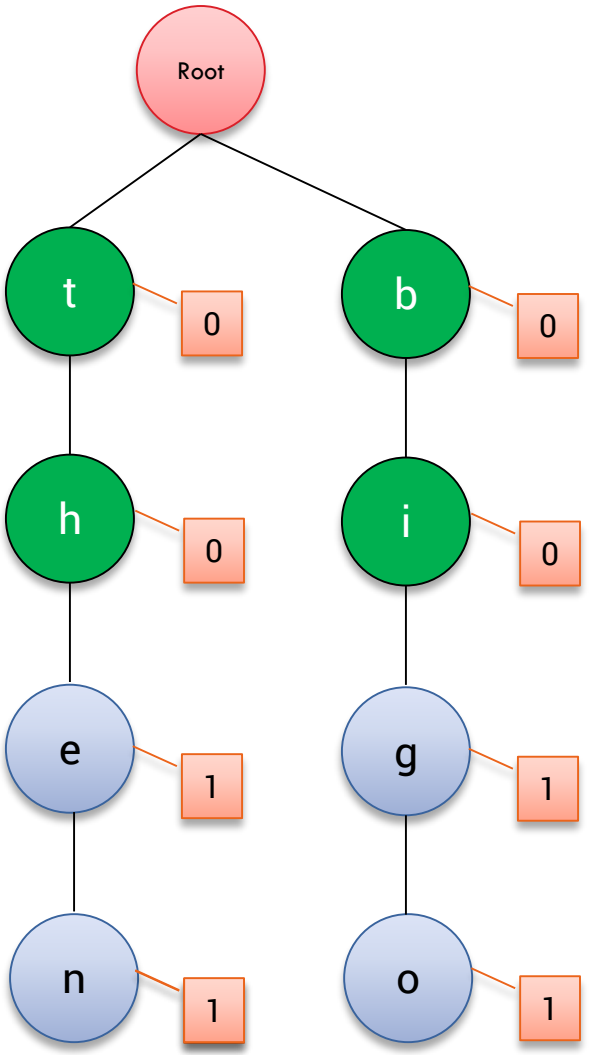


3. Xóa từ trong cây

Trường hợp 2: Từ cần xóa là tiền tố của từ khác.

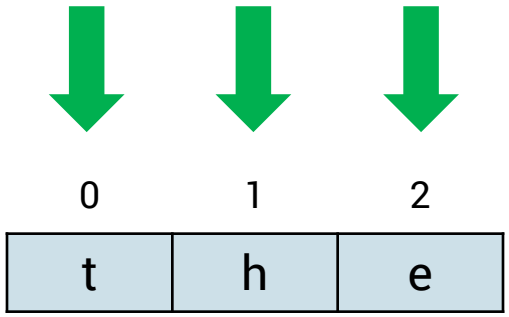
Từ cần xóa:

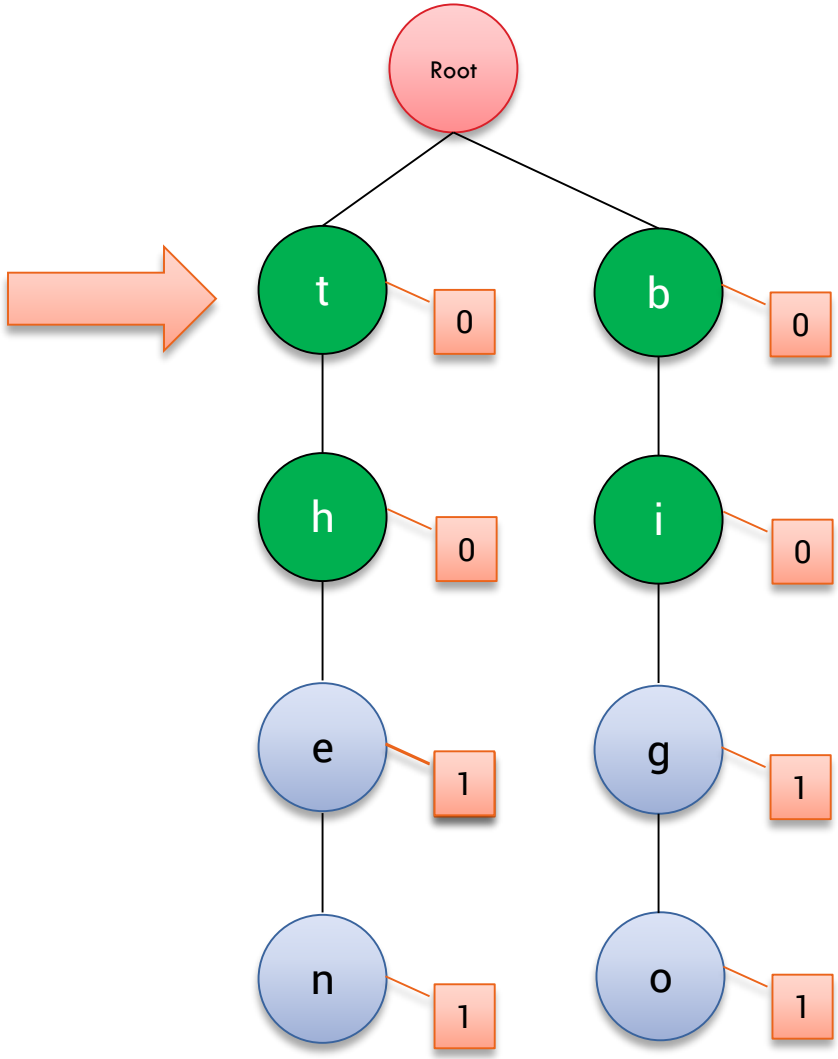
0	1	2
t	h	e



3. Xóa từ trong cây

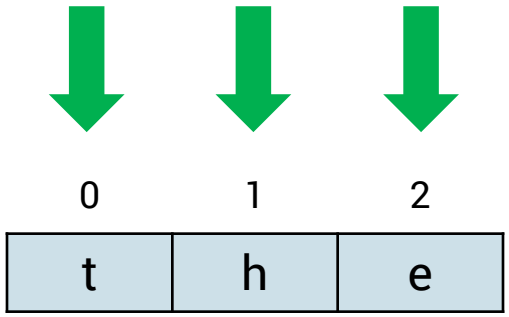
Bước 1: Duyệt từ đầu đến cuối từ cần xóa.



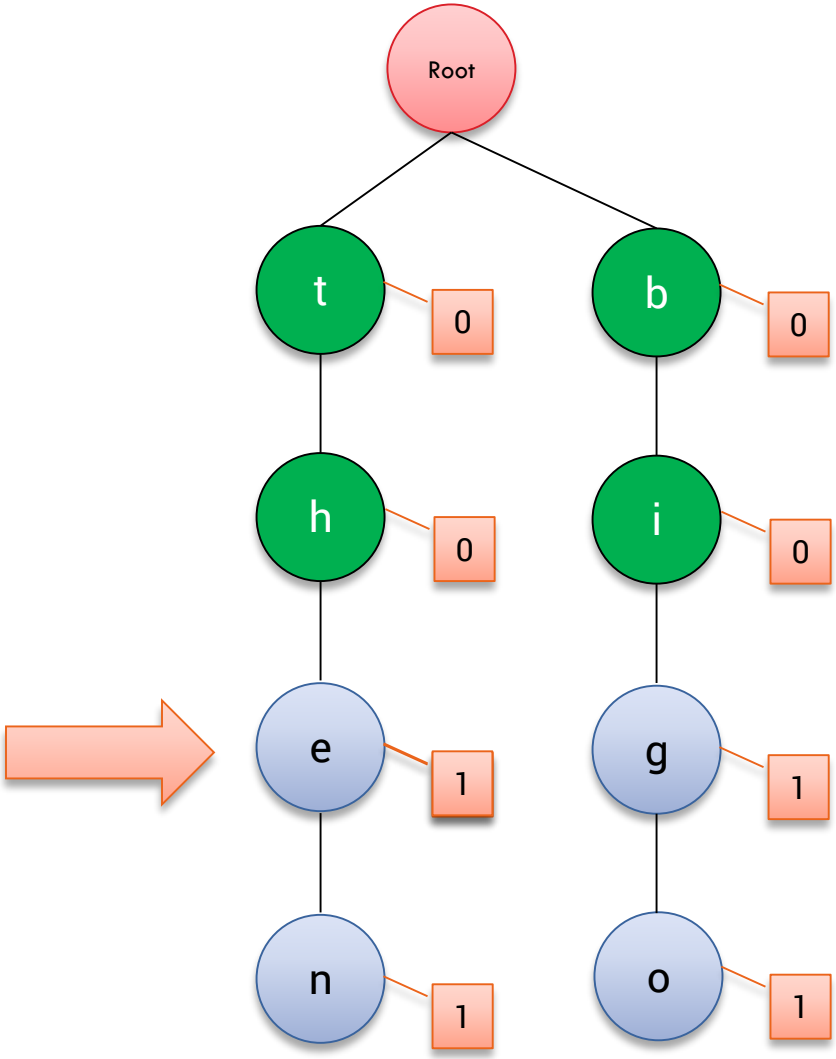


3. Xóa từ trong cây

Bước 2: Xét từng node một xem có thỏa điều kiện để xóa hay không.



- Node đó có phải kết thúc của một từ nào đó hay không?
- Node đó có là tiền tố của node nào khác hay không?

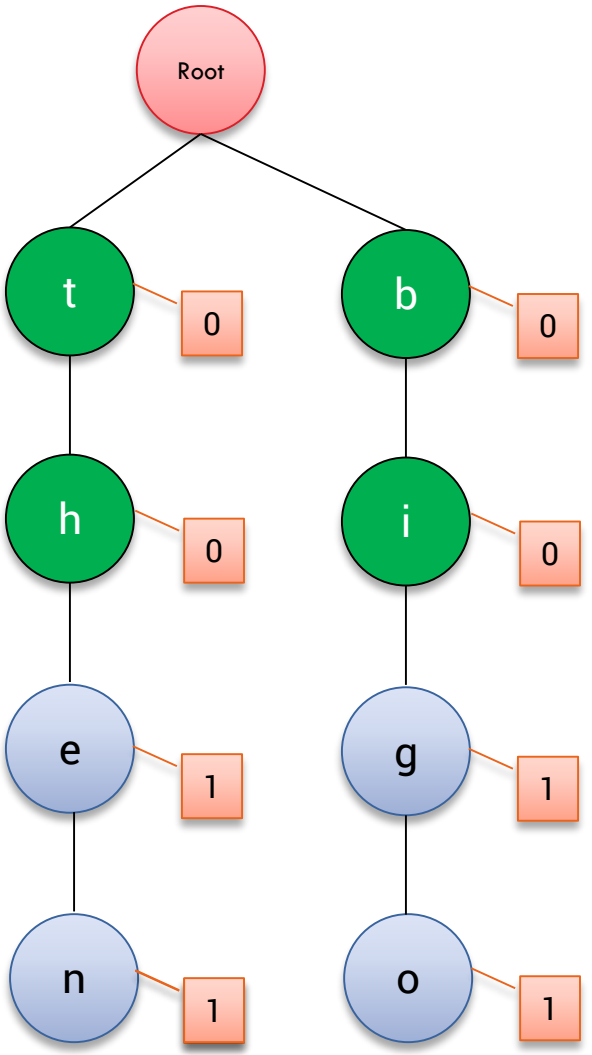


3. Xóa từ trong cây

Trường hợp 3: Từ cần xúa là từ đang chứa từ khác.

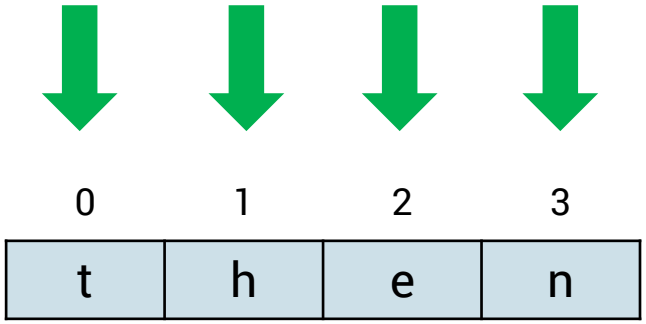
Từ cần xóa:

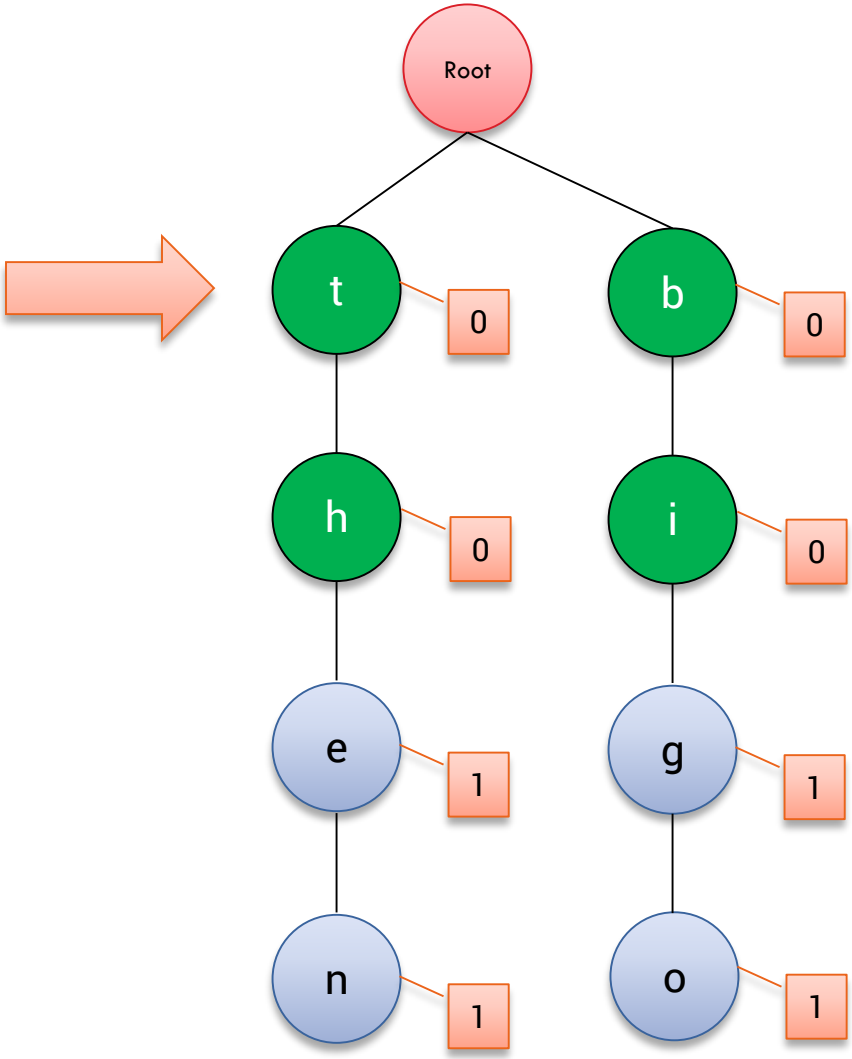
0	1	2	3
t	h	e	n



3. Xóa từ trong cây

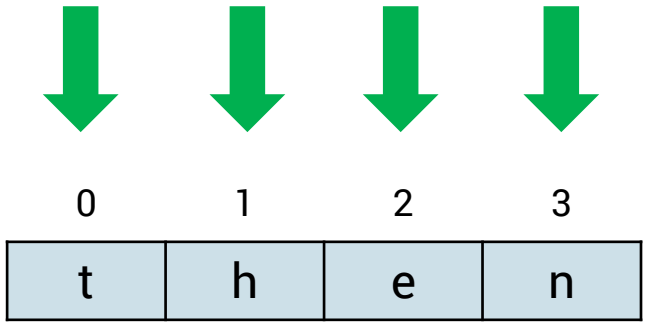
Bước 1: Duyệt từ đầu đến cuối từ cần xóa.



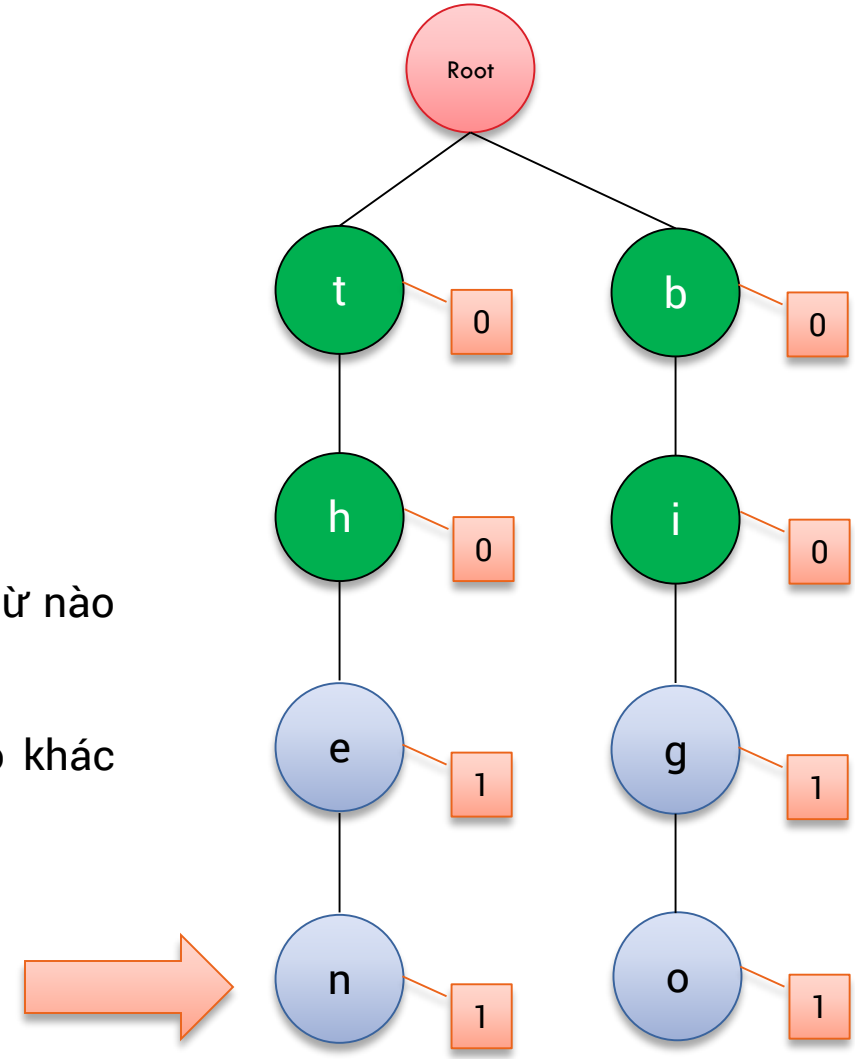


3. Xóa từ trong cây

Bước 2: Xét từng node một xem có thỏa điều kiện để xóa hay không.



- Node đó có phải kết thúc của một từ nào đó hay không?
- Node đó có là tiền tố của node nào khác hay không?



Source Code minh họa

```
1.  bool isWord(struct Node *node)
2.  {
3.      return (node->countWord != 0);
4.  }
```


```
5.  bool isEmpty(struct Node *node)
6.  {
7.      for (int i = 0; i < MAX; i++)
8.      {
9.          if (node->child[i] != NULL)
10.         {
11.             return false;
12.         }
13.     }
14.     return true;
15. }
```

Source Code minh họa

```
16. bool removeWord(struct node *root, string s, int level, int len)
17. {
18.     if (!root)
19.         return false;
20.     int ch = s[level] - 'a';
21.     if (level == len)
22.     {
23.         if (root->countWord > 0)
24.         {
25.             root->countWord--;
26.             return true;
27.         }
28.         return false;
29.     }
30.     int flag = removeWord(root->child[ch], s, level + 1, len);
31.     if (flag && !isWord(root->child[ch]) && isEmpty(root->child[ch]))
32.     {
33.         delete root->child[ch];
34.         root->child[ch] = NULL;
35.     }
36.     return flag;
37. }
```

Source Code minh họa python

```
1.  def isWord(node):
2.      return node.countWord != 0
```

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

Source Code minh họa python

```
5.  def removeWord(root, s, level, len):
6.      if root == None:
7.          return False
8.      if level == len:
9.          if root.countWord > 0:
10.             root.countWord -= 1
11.             return True
12.         return False
13.     ch = s[level]
14.     flag = removeWord(root.child[ch], s, level + 1, len)
15.     if flag == True and isWord(root.child[ch]) == False
                                    and isEmpty(root.child[ch]) == True:
16.         del root.child[ch]
17.         root.child[ch] = None
18.     return flag
```

Source Code minh họa Java

```
1. private boolean isWord(Node node) {
2.     return (node.countWord != 0);
3. }
```


```
4. private boolean isEmpty(Node node) {
5.     for (int i = 0; i < MAX; i++)
6.         if (node.child[i] != null)
7.             return false;
8.     return true;
9. }
```

Source Code minh họa Java

Part 1

```
10. public boolean removeWord(String s) {
11.     return removeWord(root, s, 0, s.length());
12. }

13. private boolean removeWord(Node root, String s, int level, int
    len) {
14.     if (root == null)
15.         return false;
16.     if (level == len) {
17.         if (root.countWord > 0) {
18.             root.countWord--;
19.             return true;
20.         }
21.         return false;
22.     }
// to be continued
```

Source Code minh họa Java

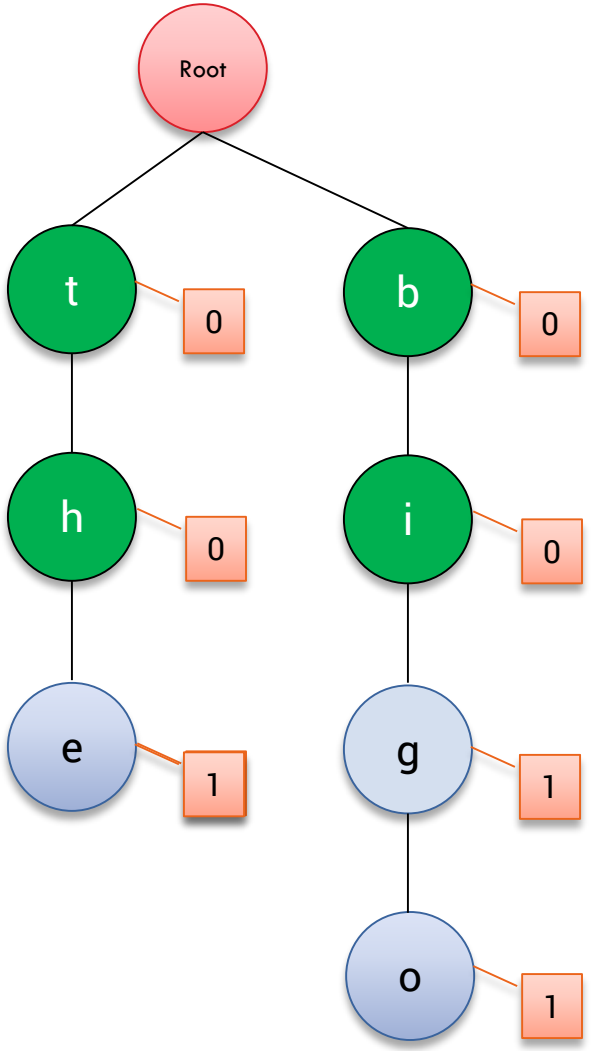
Part 2

```
23.      int ch = s.charAt(level) - 'a';
24.      boolean flag = removeWord(root.child[ch], s, level + 1, len);
25.      if (flag && !isWord(root.child[ch]) &&
                              isEmpty(root.child[ch])) {
26.          root.child[ch] = null;
27.      }
28.      return flag;
29.  }
```

In toàn bộ từ trong cây

Trong cây Trie hiện tại có 3 từ được duyệt và in lần lượt là:

- big
- bigo
- the



Source Code minh họa

```
1.  void printWord(struct Node* root, string s)
2.  {
3.      if (isWord(root))
4.      {
5.          cout << s << endl;
6.      }
7.      for (int i = 0; i < MAX; i++)
8.      {
9.          if (root->child[i])
10.         {
11.             printWord(root->child[i], s + (char)('a' + i));
12.         }
13.     }
14. }
```

Source Code minh họa

```
1.  def printWord(root, s):
2.      if isWord(root):
3.          print(s)
4.      for ch in root.child:
5.          printWord(root.child[ch], s + ch)
```

Source Code minh họa

```java
public void printWord() {
    printWord(root, "");
}
private void printWord(Node root, String s) {
    if (isWord(root)) {
        System.out.println(s);
    }
    for (int i = 0; i < MAX; i++) {
        if (root.child[i] != null) {
            printWord(root.child[i], s + (char)('a' + i));
        }
    }
}
```

HÀM MAIN GỌI THỰC HIỆN CHƯƠNG TRÌNH

Source Code minh họa

```cpp
int main()
{
    struct Node *root = newNode();
    addWord(root, "the");
    addWord(root, "then");
    addWord(root, "bigo");
    cout << findWord(root, "there") << endl;
    cout << findWord(root, "th") << endl;
    cout << findWord(root, "the") << endl;
    removeWord(root, "bigo", 0, 4);
    removeWord(root, "the", 0, 3);
    removeWord(root, "then", 0, 4);
    return 0;
}
```

Source Code minh họa

```
1.  if __name__ == '__main__':
2.      root = Node()
3.      addWord(root, "the")
4.      addWord(root, "then")
5.      addWord(root, "bigo")
6.      print(findWord(root, "there"))
7.      print(findWord(root, "th"))
8.      print(findWord(root, "the"))
9.      removeWord(root, "bigo", 0, 4)
10.     removeWord(root, "the", 0, 3)
11.     removeWord(root, "then", 0, 4)
```

Source Code minh họa

```java
public static void main(String[] args) {
    Trie trie = new Trie();
    trie.addWord("the");
    trie.addWord("then");
    trie.addWord("bigo");
    System.out.println(trie.findWord("there"));
    System.out.println(trie.findWord("th"));
    System.out.println(trie.findWord("the"));
    trie.removeWord("bigo");
    trie.removeWord("the");
    trie.removeWord("then");
}
```

Hỏi đáp