자동완성

문제 설명

자동완성

포털 다음에서 검색어 자동완성 기능을 넣고 싶은 라이언은 한 번 입력된 문자열을 학습해서 다음 입력 때 활용하고 싶어 졌다. 예를 들어, `go` 가 한 번 입력되었다면, 다음 사용자는 `g` 만 입력해도 `go` 를 추천해주므로 `o` 를 입력할 필요가 없어진다! 단, 학습에 사용된 단어들 중 앞부분이 같은 경우에는 어쩔 수 없이 다른 문자가 나올 때까지 입력을 해야 한다.
효과가 얼마나 좋을지 알고 싶은 라이언은 학습된 단어들을 찾을 때 몇 글자를 입력해야 하는지 궁금해졌다.

예를 들어, 학습된 단어들이 아래와 같을 때

```
go
gone
guild
```

- `go` 를 찾을 때 `go` 를 모두 입력해야 한다.
- `gone` 을 찾을 때 `gon` 까지 입력해야 한다. (`gon` 이 입력되기 전까지는 `go` 인지 `gone` 인지 확신할 수 없다.)
- `guild` 를 찾을 때는 `gu` 까지만 입력하면 `guild` 가 완성된다.

이 경우 총 입력해야 할 문자의 수는 `7` 이다.

라이언을 도와 위와 같이 문자열이 입력으로 주어지면 학습을 시킨 후, 학습된 단어들을 순서대로 찾을 때 몇 개의 문자를 입력하면 되는지 계산하는 프로그램을 만들어보자.

입력 형식

학습과 검색에 사용될 중복 없는 단어 `N` 개가 주어진다.
모든 단어는 알파벳 소문자로 구성되며 단어의 수 `N` 과 단어들의 길이의 총합 `L` 의 범위는 다음과 같다.

- 2 <= `N` <= 100,000
- 2 <= `L` <= 1,000,000

출력 형식

단어를 찾을 때 입력해야 할 총 문자수를 리턴한다.

Trie Tree

사전식으로 단어를 찾아야 할 때 효율적으로 해답을 구할 수 있는 자료구조

=> (전화번호부, 사전찾기)

일반적인 구조는

`Map<Character, Node>` 로 각각을 구분하고,

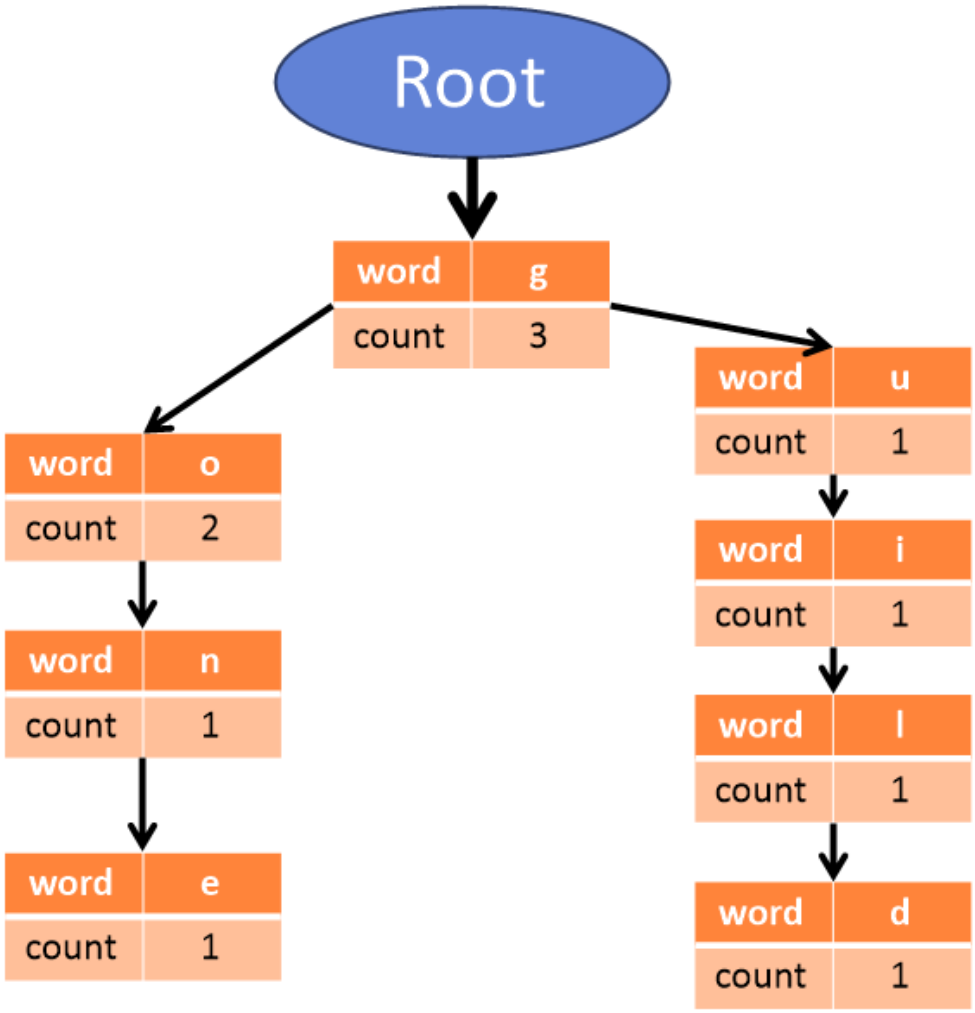
현재 탐색한 값이 유일한 형태인지를 판단하는 `boolean`값의 구조를 갖는다.

접근 방식

각 단어를 조사 할 때, 루트부터 내 자식 노드에 있는지 판별하고,

이미 들어있다면, `Node` 의 `count` 값을 증가 시키는 것으로 중복되는 횟수를 저장한다.

ex) `"go"` , `"gone"` , `"guild"` 의 예시의 경우

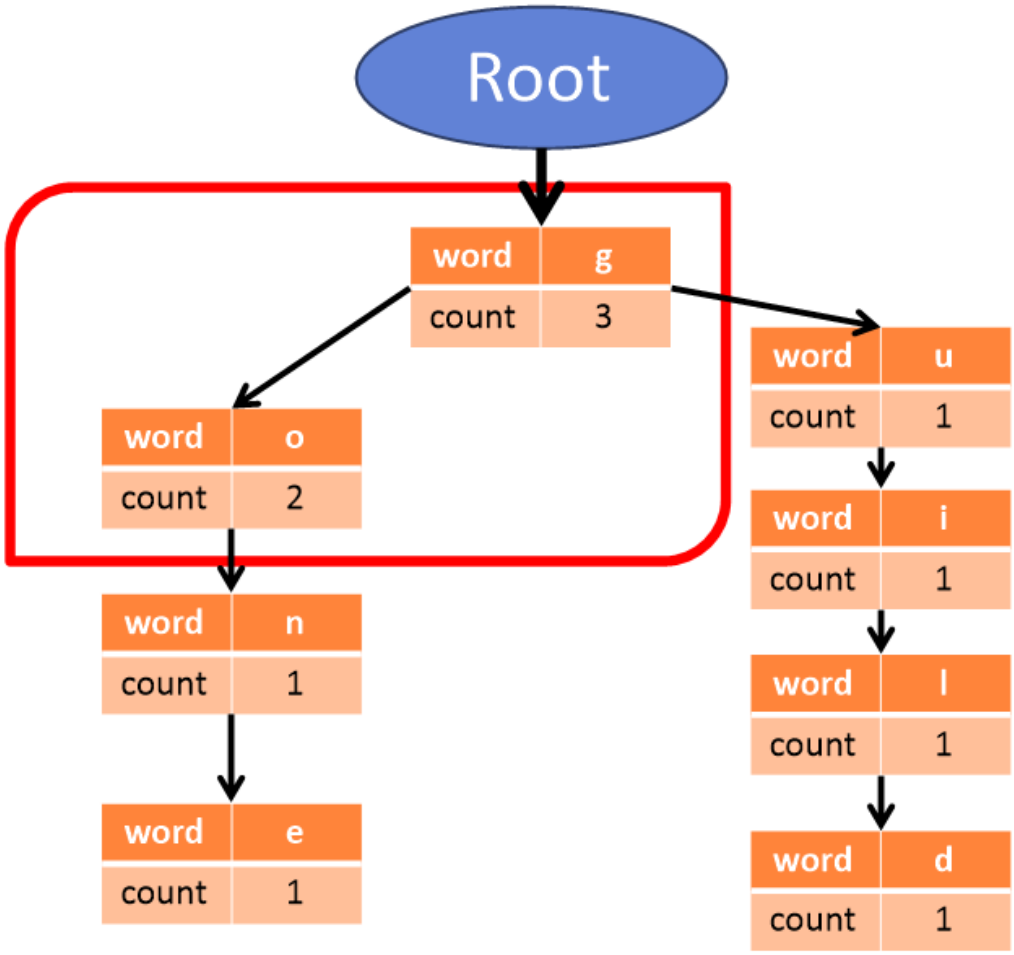


"go" 를 찾을 경우

word 의 길이만큼 탐색을 했는데 count가 1인 노드를 발견하지 못하였다.

=> 다른 변수가 있어 모든 값을 입력하는 단어였다는 뜻

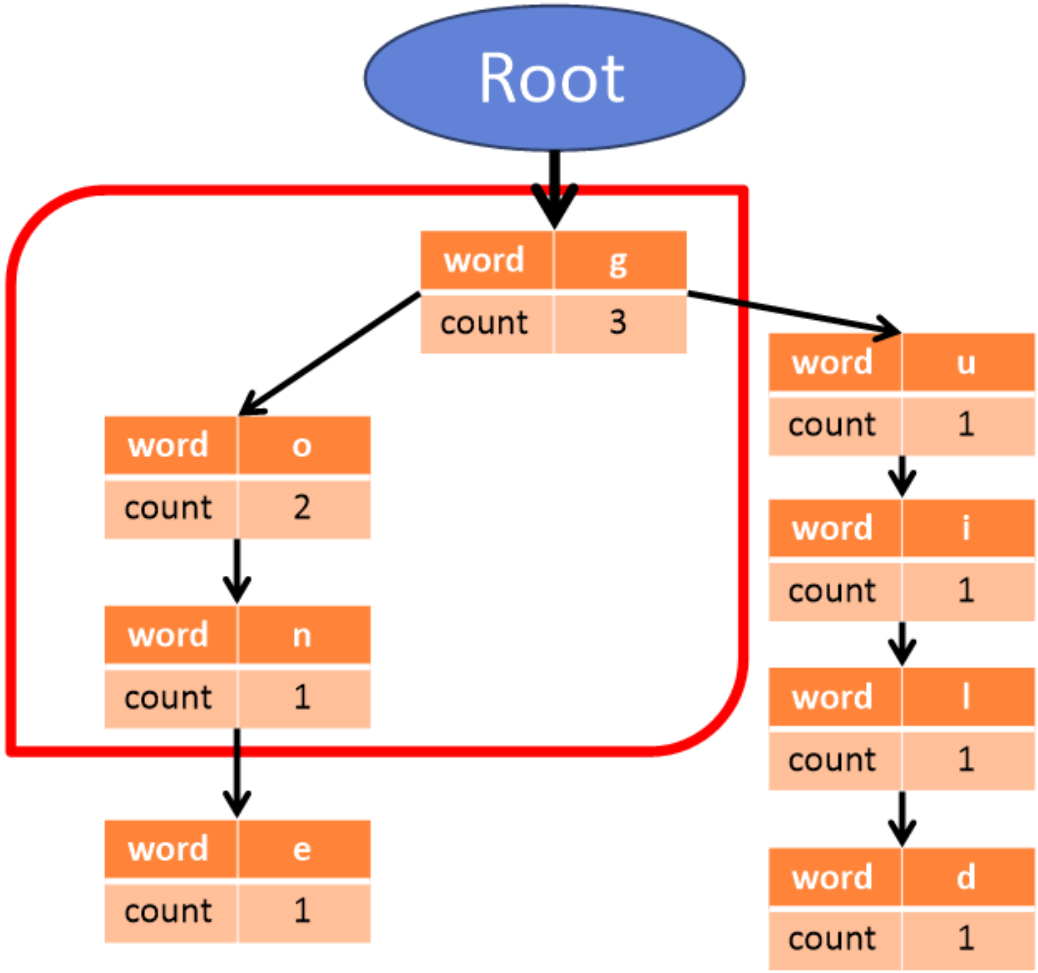
answer에 word의 길이만큼 추가 된다.



"gone" 을 찾는 경우

세번 째 노드인 n에서 count가 1인 노드를 찾았으므로 자식 노드를 더 보지 않아도

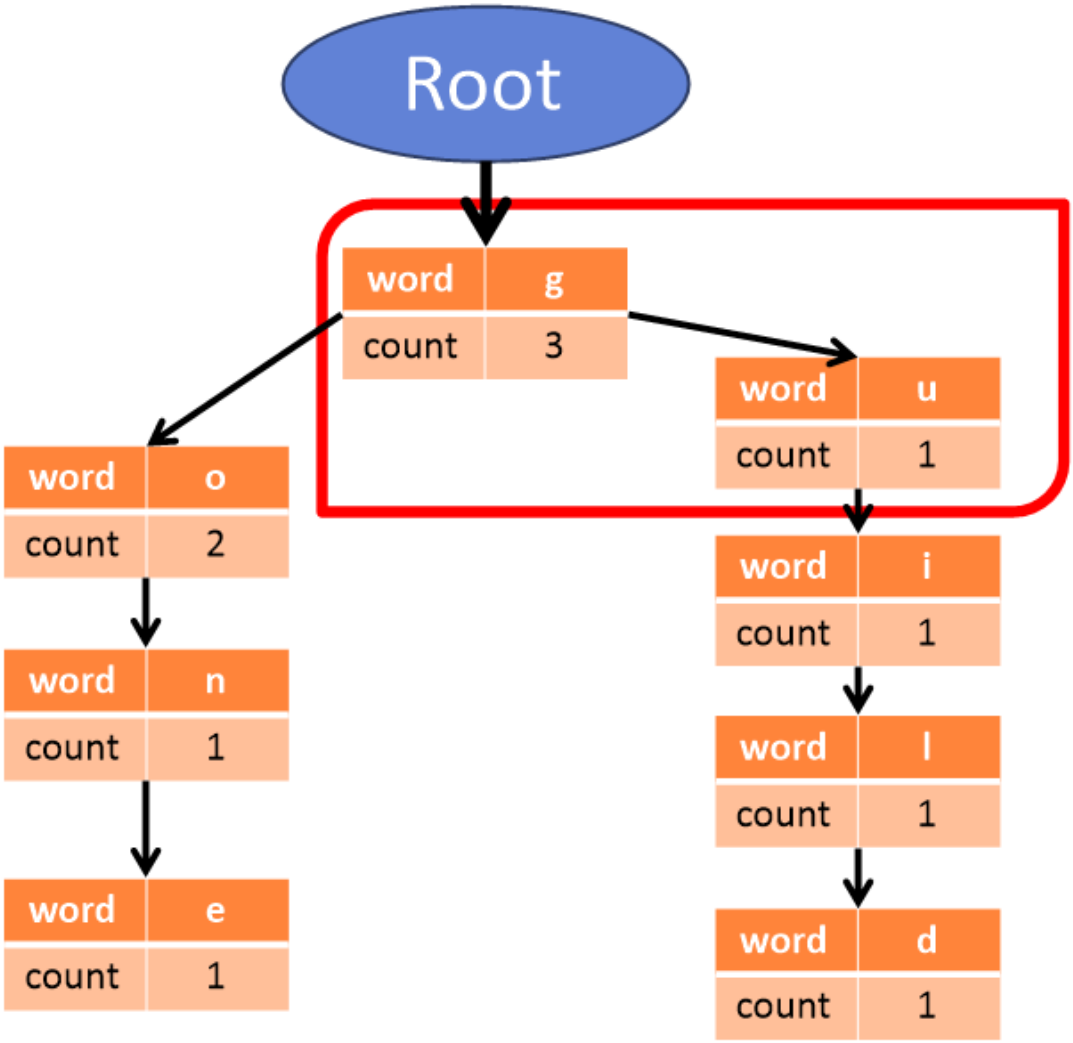
유일한 단어이므로 여태까지 내려온 단계의 수만큼만 answer에 더하고 종료



"guild" 를 찾는 경우

"gone" 과 마찬가지로 u 에서 count가 1인 노드를 찾았으므로

u 까지의 깊이만 계산하고 종료



코드

```java
package Algo_Study_Programmers;

import java.util.*;

class Node2 {
  int count;
  Map<Character, Node2> map;

  public Node2() {
    this.count = 1;
    map = new HashMap<Character, Node2>();
  }
```

```
}

public class Solution_자동완성2 {
  static Node2 root;
  static int answer;

  private static void insert(String word) {
    Node2 curHead = root;
    for(int i = 0; i < word.length(); i++) {
      if(curHead.map.containsKey(word.charAt(i))) {
        curHead.map.get(word.charAt(i)).count++;
        curHead = curHead.map.get(word.charAt(i));
      }
      else {
        Node2 tmp = new Node2();
        curHead.map.put(word.charAt(i), tmp);
        curHead = curHead.map.get(word.charAt(i));
      }
    }
  }
  private static void find(String word) {
    Node2 curHead = root;
    for(int i = 0; i < word.length(); i++) {
      if(curHead.map.containsKey(word.charAt(i))) {
        curHead = curHead.map.get(word.charAt(i));
        answer++;
      }

      if(curHead.count == 1) {
        return;
      }
    }
  }
    public static int solution(String[] words) {

      answer = 0;

      root = new Node2();

      for(int i = 0; i < words.length; i++) {
        insert(words[i]);
      }

      for(int i = 0; i < words.length; i++) {
        find(words[i]);
      }
      return answer;
    }
  public static void main(String[] args) {
    String[] tmp = { "go", "gone", "guild" };

    System.out.println(solution(tmp));
  }
}
```

어려웠던점 및 회고

Trie 구조에서 소문자 알파벳만 사용한다면, 각 서브 Node를

26칸의 배열 형태로 사용하여도 가능

특정 문제 풀이의 특화 된 자료구조 및 알고리즘에 대한 이해 및 활용도 연습이 더 필요하다고 느낌

Trie가 문자열을 다룰 때 가장 좋은 방법인가 ?

=> 시간 복잡도면에서는 충분한 이득을 볼 수 있겠지만,

시간 복잡도를 만족시키기 위해서는

O(포인터 크기 * 포인터 배열 개수 * 트라이에 존재하는 총 노드의 개수) 가 필요하기 때문에

공간 복잡도가 증가한다는 단점 또한 가지고 있다.