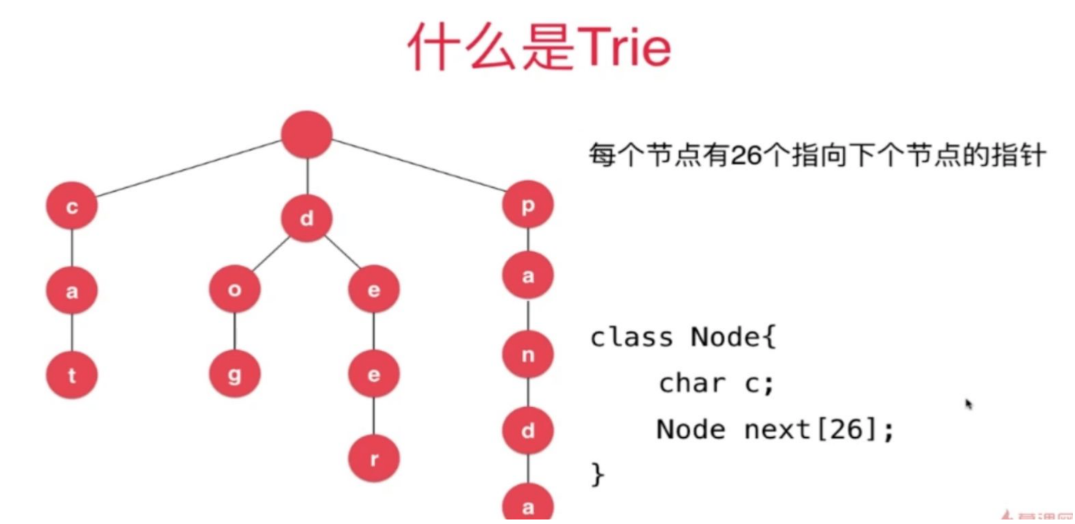
数据结构与算法 11- 字典树Trie

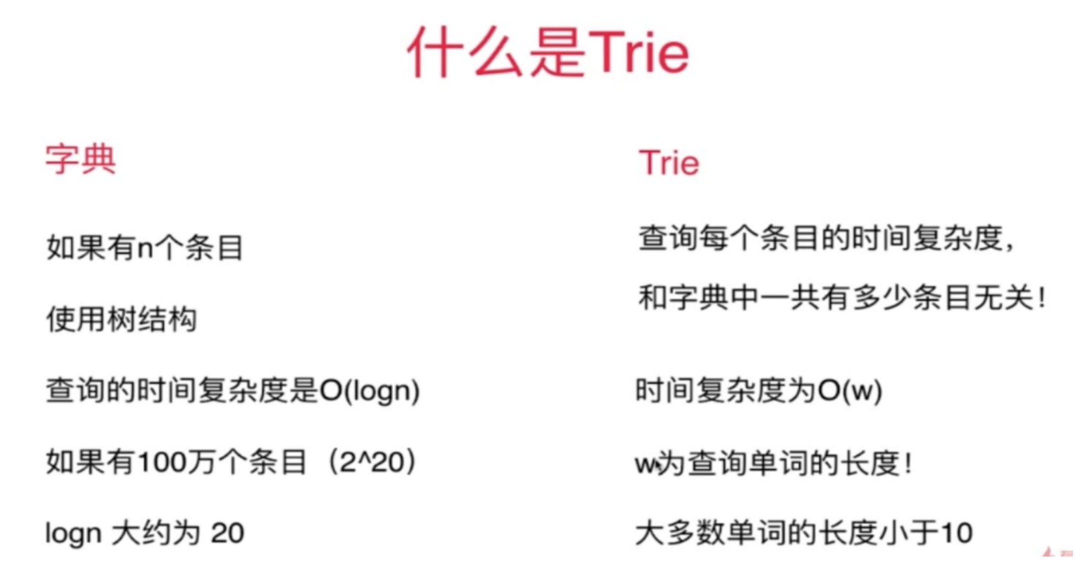
笔记本:	我的笔记		
创建时间:	2020/10/9 22:51	**更新时间:**	2020/10/9 23:15
作者:	liuhouer		
标签:	算法		



什么是Trie

每个节点有26个指向下个节点的指针

```
class Node{
    char c;
    Node next[26];
}
```



什么是Trie

字典

如果有n个条目

使用树结构

查询的时间复杂度是O(logn)

如果有100万个条目（2^20）

logn 大约为 20

Trie

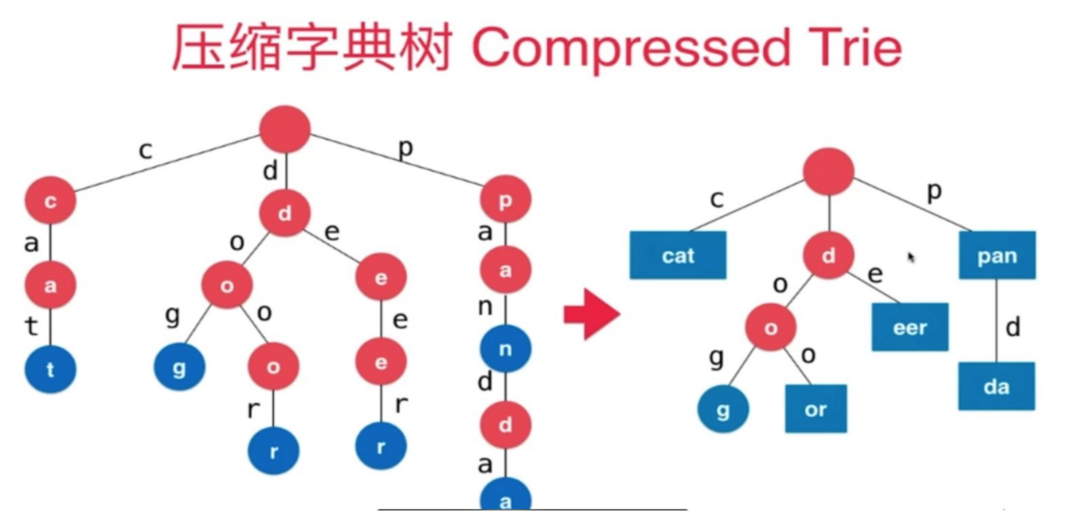
查询每个条目的时间复杂度，

和字典中一共有多少条目无关！

时间复杂度为O(w)

w为查询单词的长度！

大多数单词的长度小于10



压缩字典树 Compressed Trie

1.实现字典树

```
import java.util.TreeMap;

public class Trie {

    private class Node{

        public boolean isWord;
        public TreeMap<Character, Node> next;

        public Node(boolean isWord){
            this.isWord = isWord;
            next = new TreeMap<>();
        }

        public Node(){
            this(false);
        }
    }

    private Node root;
    private int size;

    public Trie(){
        root = new Node();
        size = 0;
    }

    // 获得Trie中存储的单词数量
    public int getSize(){
        return size;
    }

    // 向Trie中添加一个新的单词word
    public void add(String word){

        Node cur = root;
        for(int i = 0 ; i < word.length() ; i ++){
            char c = word.charAt(i);
            if(cur.next.get(c) == null)
                cur.next.put(c, new Node());
            cur = cur.next.get(c);
        }
        if(!cur.isWord){
            cur.isWord = true;
            size ++;
        }
    }


    // 3-Trie字典树的查询 查询单词word是否在Trie中
    public boolean contains(String word){


        Node cur = root;
        for(int i = 0 ; i < word.length() ; i ++){
            char c = word.charAt(i);
            if(cur.next.get(c) == null)
                return false;
            cur = cur.next.get(c);
        }
        return cur.isWord;
    }


    //4-Trie字典树的前缀查询 查询是否在Trie中有单词以prefix为前缀
    public boolean isPrefix(String prefix){


        Node cur = root;
        for(int i = 0 ; i < prefix.length() ; i ++){
            char c = prefix.charAt(i);
            if(cur.next.get(c) == null)
                return false;
            cur = cur.next.get(c);
        }


        return true;
    }

    /** 5-Trie字典树和简单的模式匹配 Returns if the word is in the data
structure. A word could contain the dot character '.' to represent any one
letter. */
    public boolean search(String word) {
        return match(root, word, 0);
    }


    private boolean match(Node node, String word, int index){


        if(index == word.length())
            return node.isWord;


        char c = word.charAt(index);


        if(c != '.'){
            if(node.next.get(c) == null)
                return false;
            return match(node.next.get(c), word, index + 1);
        }
        else{
            for(char nextChar: node.next.keySet())
                if(match(node.next.get(nextChar), word, index + 1))
                    return true;
            return false;
        }
    }


}
```

2.使用HashMap的Trie

```
import java.util.HashMap;


// 使用HashMap的Trie
public class Trie2 {


    private class Node{


        public boolean isWord;
        public HashMap<Character, Node> next;


        public Node(boolean isWord){
            this.isWord = isWord;
            next = new HashMap<>();
        }


        public Node(){
            this(false);
        }
    }


    private Node root;
    private int size;


    public Trie2(){
        root = new Node();
        size = 0;
    }


    // 获得Trie中存储的单词数量
    public int getSize(){
        return size;
    }


    // 向Trie中添加一个新的单词word
    public void add(String word){


        Node cur = root;
        for(int i = 0 ; i < word.length() ; i ++){
            char c = word.charAt(i);
            if(cur.next.get(c) == null)
                cur.next.put(c, new Node());
            cur = cur.next.get(c);
        }


        if(!cur.isWord){
            cur.isWord = true;
            size ++;
        }
    }


    // 查询单词word是否在Trie中
    public boolean contains(String word){


        Node cur = root;
        for(int i = 0 ; i < word.length() ; i ++){
            char c = word.charAt(i);
            if(cur.next.get(c) == null)
                return false;
            cur = cur.next.get(c);
        }
        return cur.isWord;
    }
}
```

3.使用Array的Trie

```
public class Trie3 {


    private class Node{


        public boolean isWord;
        public Node[] next;


        public Node(boolean isWord){
            this.isWord = isWord;
            next = new Node[26];
        }


        public Node(){
            this(false);
        }
    }


    private Node root;
    private int size;


    public Trie3(){
        root = new Node();
        size = 0;
    }


    // 获得Trie中存储的单词数量
    public int getSize(){
        return size;
    }


    // 向Trie中添加一个新的单词word
    public void add(String word){


        Node cur = root;
        for(int i = 0 ; i < word.length() ; i ++){
            char c = word.charAt(i);
            if(cur.next[c-'a'] == null)
                cur.next[c-'a'] = new Node();
            cur = cur.next[c-'a'];
        }


        if(!cur.isWord){
            cur.isWord = true;
            size ++;
        }
    }


    // 查询单词word是否在Trie中
    public boolean contains(String word){


        Node cur = root;
        for(int i = 0 ; i < word.length() ; i ++){
            char c = word.charAt(i);
            if(cur.next[c-'a'] == null)
                return false;
            cur = cur.next[c-'a'];
        }
        return cur.isWord;
    }
}
```