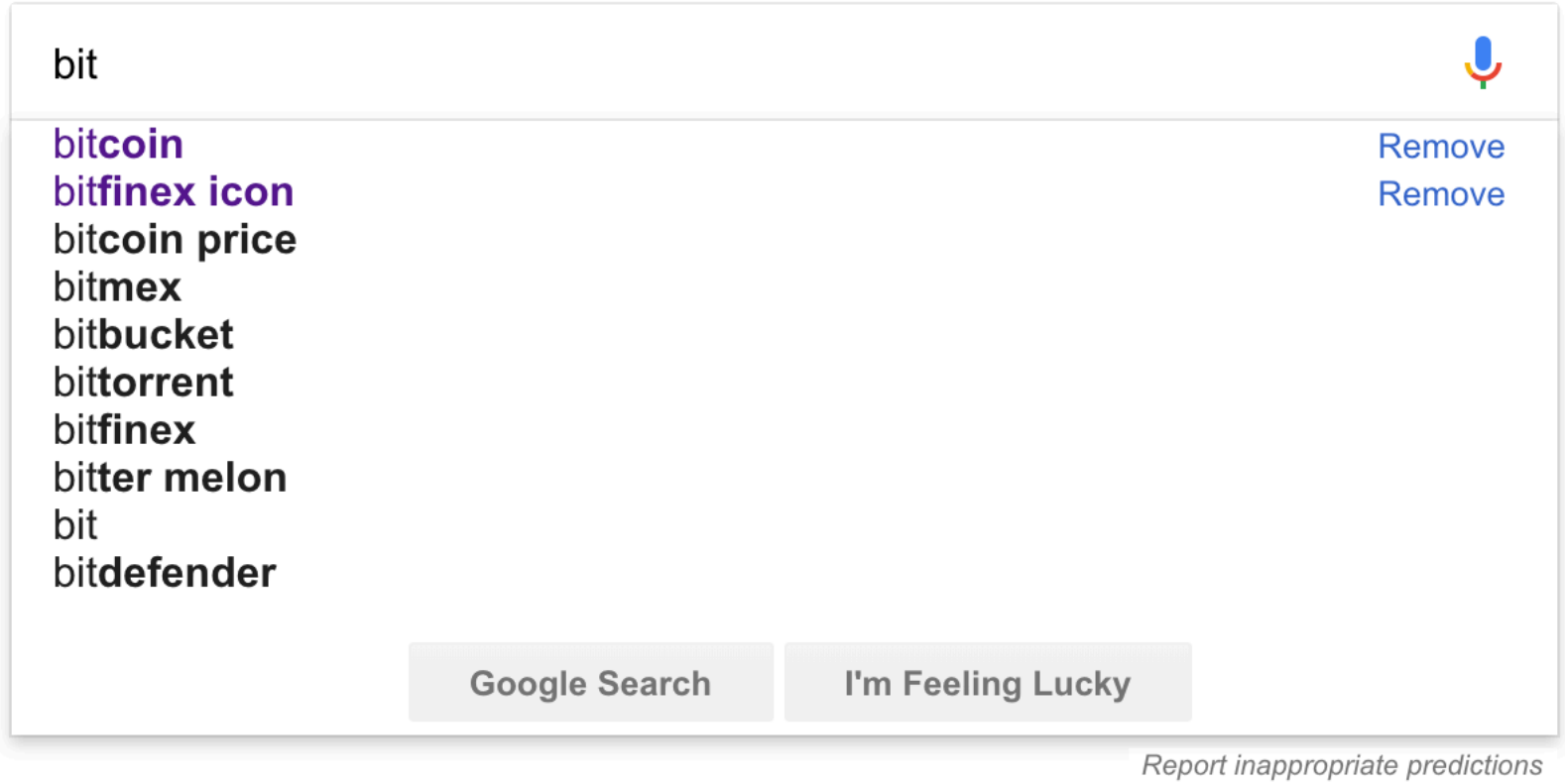
字典树（Trie)

本节内容

1. Trie 树的数据结构
2. Trie 树的核心思想
3. Trie 树的基本性质

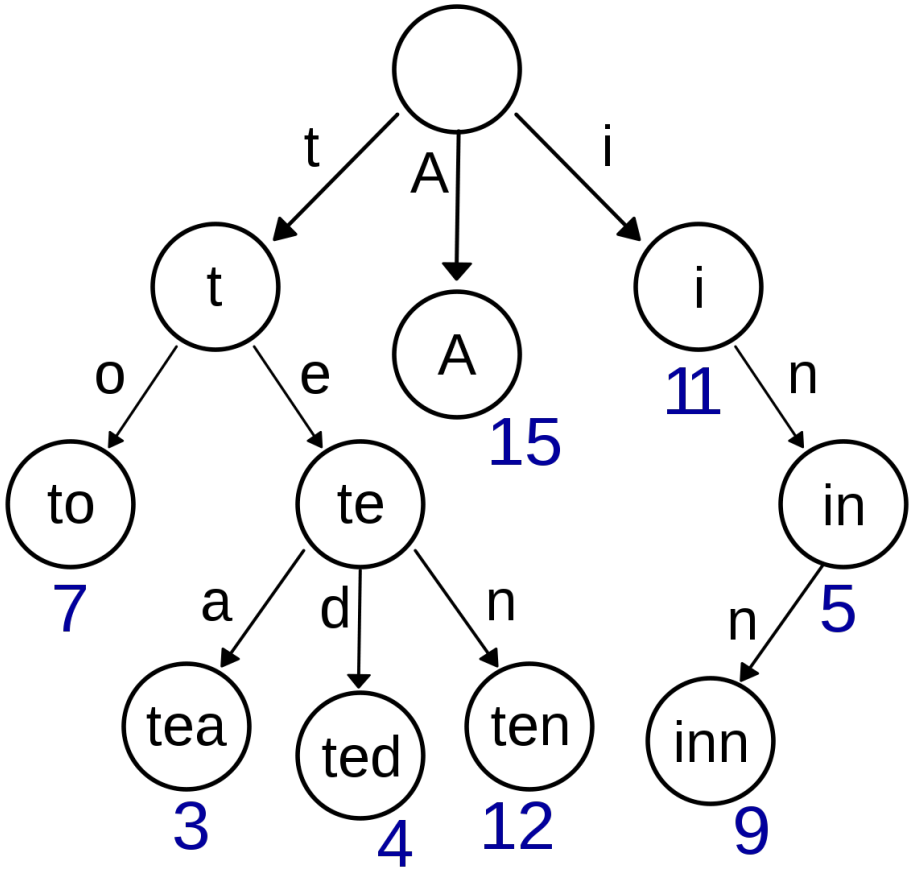
实际要解决的问题


bit
bitcoin
Remove
bitfinex icon
Remove
bitcoin price
bitmex
bitbucket
bittorrent
bitfinex
bitter melon
bit
bitdefender
Google Search
I'm Feeling Lucky
Report inappropriate predictions

基本结构

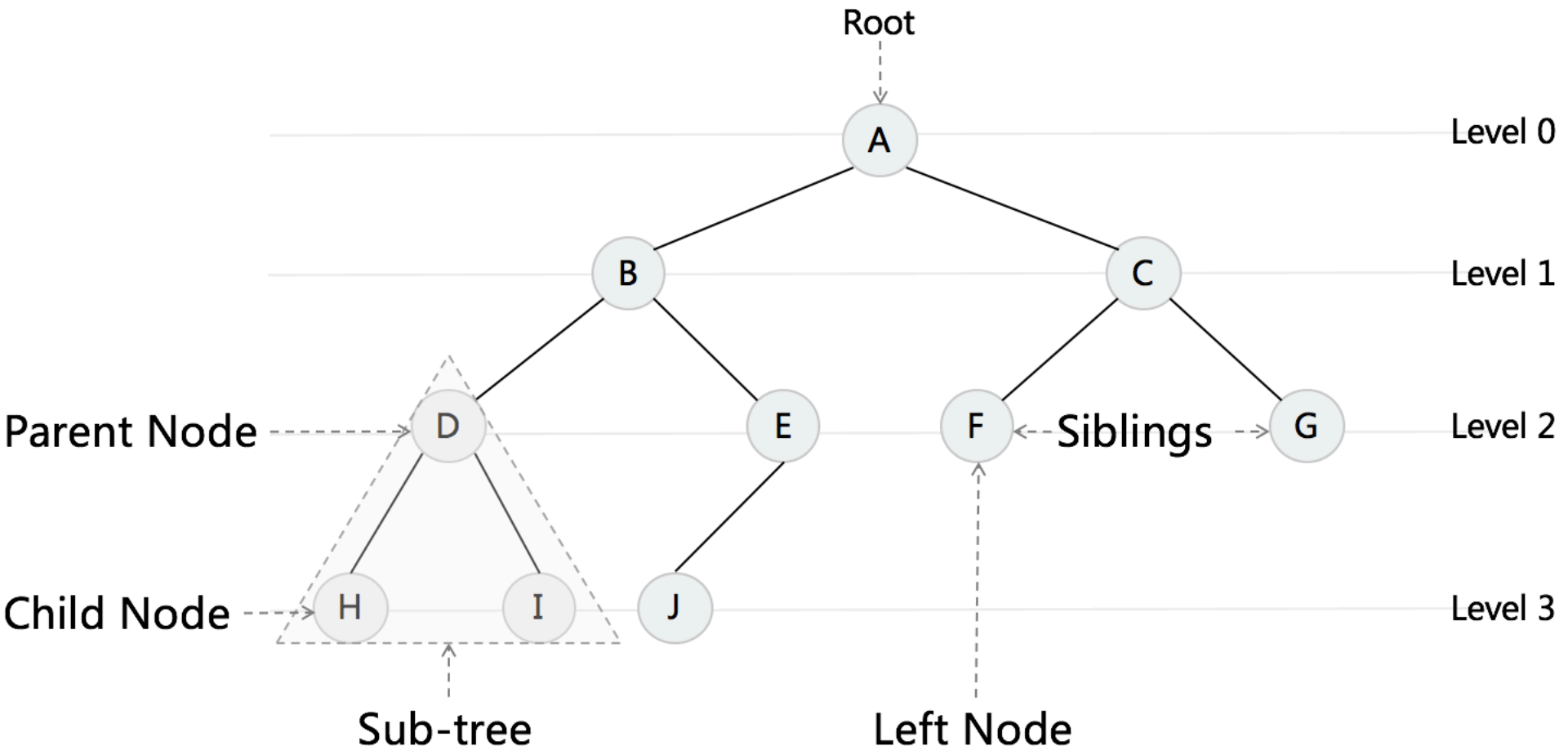
Trie树，即字典树，又称单词查找树或键树，是一种树形结构，是一种哈希树的变种。典型应用是用于统计和排序大量的字符串（但不仅限于字符串），所以经常被搜索引擎系统用于文本词频统计。

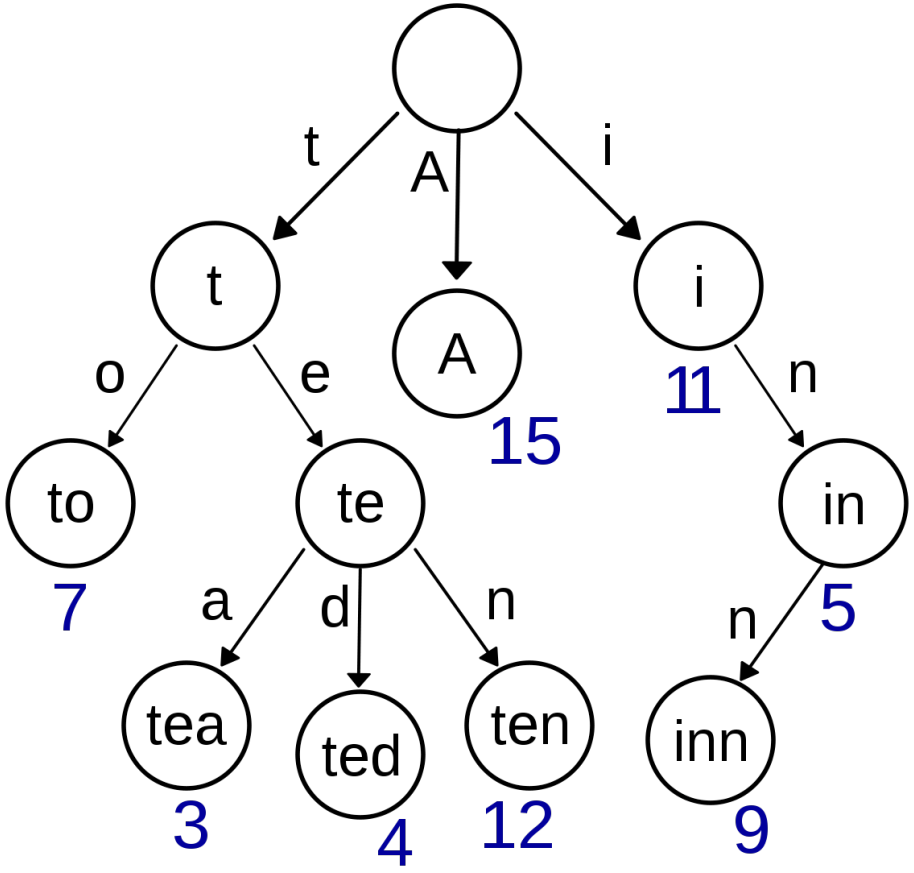
它的优点是：最大限度地减少无谓的字符串比较，查询效率比哈希表高。

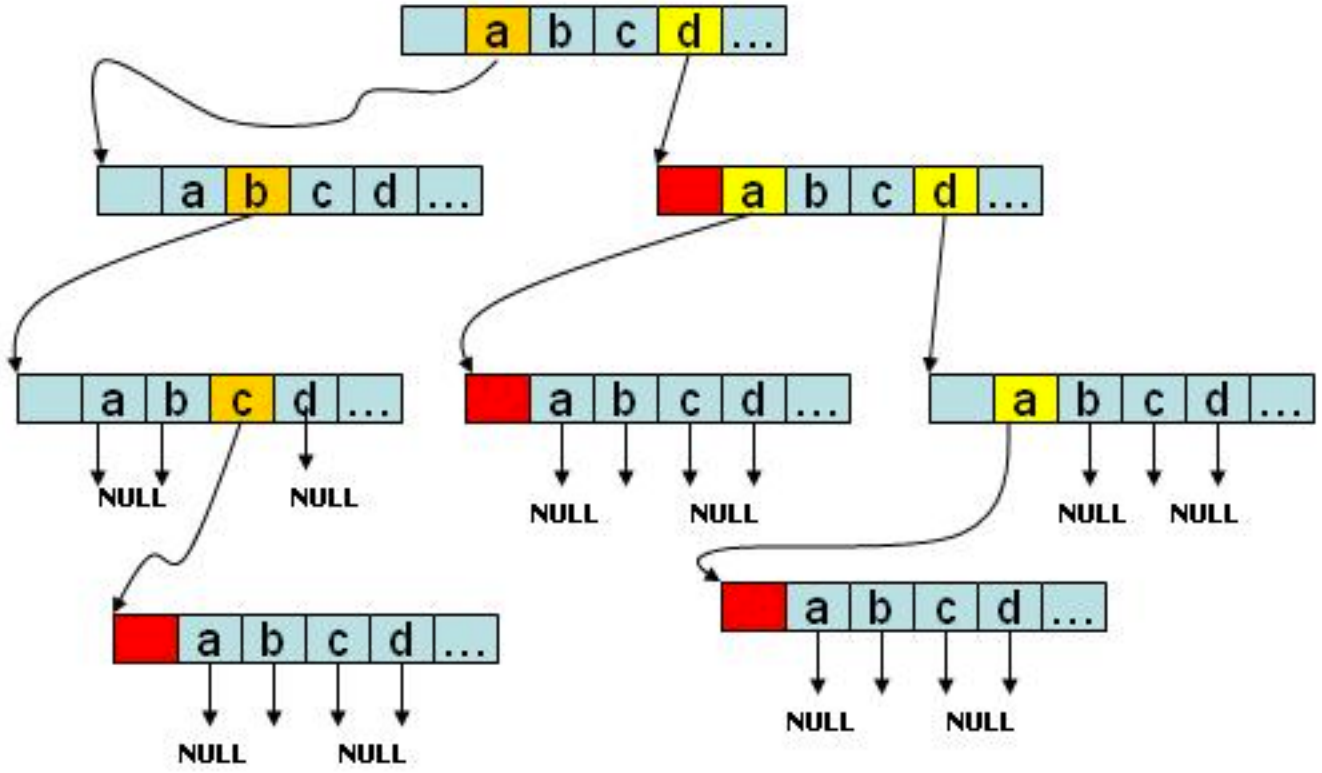


核心思想

Trie的核心思想是空间换时间。利用字符串的公共前缀来降低查询时间的开销以达到提高效率的目的。


Root
A
B
C
D
E
F
G
H
I
J
Level 0
Level 1
Level 2
Level 3
Parent Node
Siblings
Child Node
Sub-tree
Left Node




a b c d ...
a b c d ...
a b c d ...
a b c d ...
a b c d ...
a b c d ...
a b c d ...
a b c d ...
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL
NULL

```
static final int ALPHABET_SIZE = 256;

static class TrieNode {
    TrieNode[] children = new TrieNode[ALPHABET_SIZE];
    boolean isEndOfWord = false;
    TrieNode() {
        isEndOfWord = false;
        for (int i = 0; i < ALPHABET_SIZE; i++)
            children[i] = null;
    }
};
```

```
class TrieNode:

    # Trie node class
    def __init__(self):
        self.children = [None] * ALPHABET_SIZE

        # isEndOfWord is True if node represent
        # the end of the word
        self.isEndOfWord = False
```

基本性质

1. 根节点不包含字符，除根节点外每一个节点都只包含一个字符。
2. 从根节点到某一节点，路径上经过的字符连接起来，为该节点对应的字符串。
3. 每个节点的所有子节点包含的字符都不相同。

实战题目

1. https://leetcode.com/problems/implement-trie-prefix-tree/#/description

2. https://leetcode.com/problems/word-search-ii/

系统设计：search suggestion 搜索建议