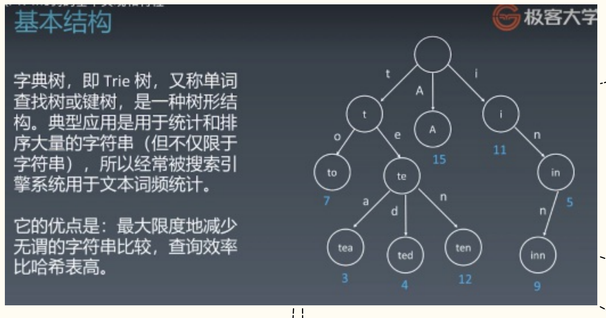
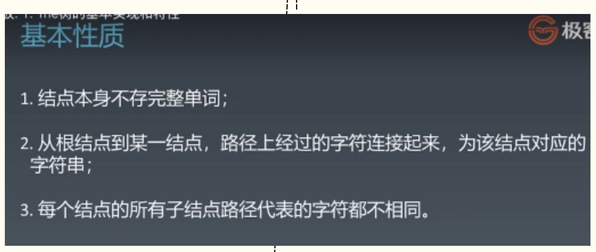
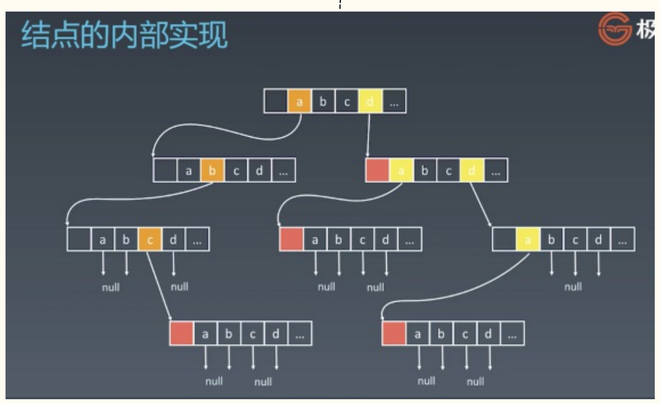






字典树

- 词频感应(google 搜索)
  输入的字母进入多叉树进行分流
- 数据结构 — 多叉树
- 核心思想 — 树的核心思想是 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
- 基本特质

蓝色的字代表当前子串的搜索频次，基于它，可以推荐给用户

26个字母的话 就会创建 26个分叉，树的每层都会分叉26个，所以空间需求很大。
但 Trie 树在存储多个具有相同前缀的键时可以使用较少的空间。

还有其他的数据结构，如平衡树和哈希表，使我们能够在字符串数据集中搜索单词。为什么我们还需要 Trie 树呢？尽管哈希表可以在 O(1)O(1) 时间内寻找键值，却无法高效的完成以下操作：
- 找到具有同一前缀的全部键值。
- 按词典序枚举字符串的数据集。

```java
class TrieNode {
    // R links to node children
    private TrieNode[] links;

    private final int R = 26;

    private boolean isEnd;

    public TrieNode() {
        links = new TrieNode[R];
    }

    public boolean containsKey(char ch) {
        return links[ch -'a'] != null;
    }
    public TrieNode get(char ch) {
        return links[ch -'a'];
    }
    public void put(char ch, TrieNode node) {
        links[ch -'a'] = node;
    }
    public void setEnd() {
        isEnd = true;
    }
    public boolean isEnd() {
        return isEnd;
    }
}
```

```java
class Trie {
    private TrieNode root;

    /** Initialize your data structure here. */
    public Trie() {
        root = new TrieNode();
    }

    /** Inserts a word into the trie. */
    public void insert(String word) {
        TrieNode node = root;
        for (int i = 0; i < word.length(); ++i) {
            char currChar = word.charAt(i);
            if (!node.containsKey(currChar))
                node.put(currChar, new TrieNode());
            node = node.get(currChar);
        }
        node.setEnd();
    }

    /** Returns if the word is in the trie. */
    public boolean search(String word) {
        TrieNode node = searchPrefix(word);
        return node != null && node.isEnd();
    }

    /** Returns if there is any word in the trie that starts with the given prefix. */
    public TrieNode searchPrefix(String prefix) {
        TrieNode node = root;
        for (int i = 0; i < prefix.length(); ++i) {
            char currChar = prefix.charAt(i);
            if (node.containsKey(currChar))
                node = node.get(currChar);
            else
                return null;
        }
        return node;
    }

    public boolean startsWith(String prefix) {
        return searchPrefix(prefix) != null;
    }
}
```

并查集 disjoint set

这个数据结构用于解决固定领域问题
背熟悉代码模版，套用即可

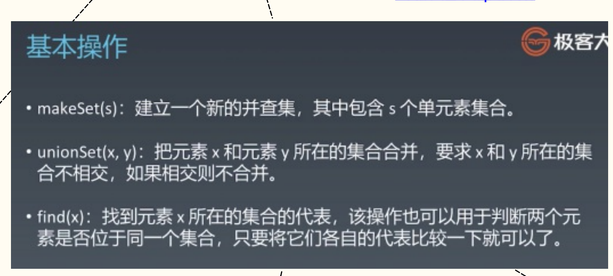
使用场景：
- 组团、配对
- Group or not?

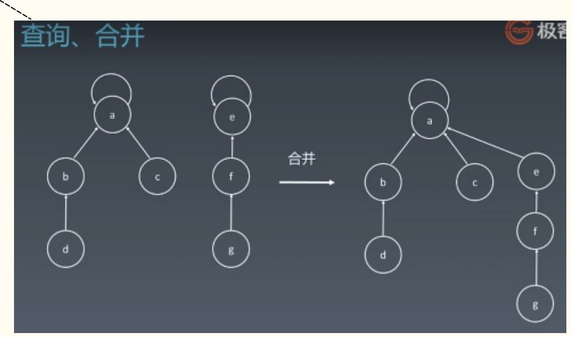
需要很快的判断两个个体是不是在一个集合之中

UnionFind
https://www.youtube.com/watch?v=0jNmHPfA_yE

UnionFInd Path Compress
https://www.youtube.com/watch?v=VHRhJWacxis

UnionFind Code
https://www.youtube.com/watch?v=KbFIZYCpONw






```java
class UnionFind {
    private int count = 0;
    private int[] parent;
    public UnionFind(int n) {
        count = n;
        parent = new int[n];
        for (int i = 0; i < n; i++) {
            parent[i] = i;
        }
    }
    public int find(int p) {
        while (p != parent[p]) {
            parent[p] = parent[parent[p]];
            p = parent[p];
        }
        return p;
    }
    public void union(int p, int q) {
        int rootP = find(p);
        int rootQ = find(q);
        if (rootP == rootQ) return;
        parent[rootP] = rootQ;
        count--;
    }
}
```