数据结构上之并查集与字典树
Data Structure - Union Find & Trie

课程版本 v6.0　　讲师 令狐冲

扫描二维码关注微信小程序/公众号
获取第一手求职资料

版权声明

九章的所有课程均受法律保护，不允许录像与传播录像

一经发现，将被追究法律责任和赔偿经济损失

今日大纲

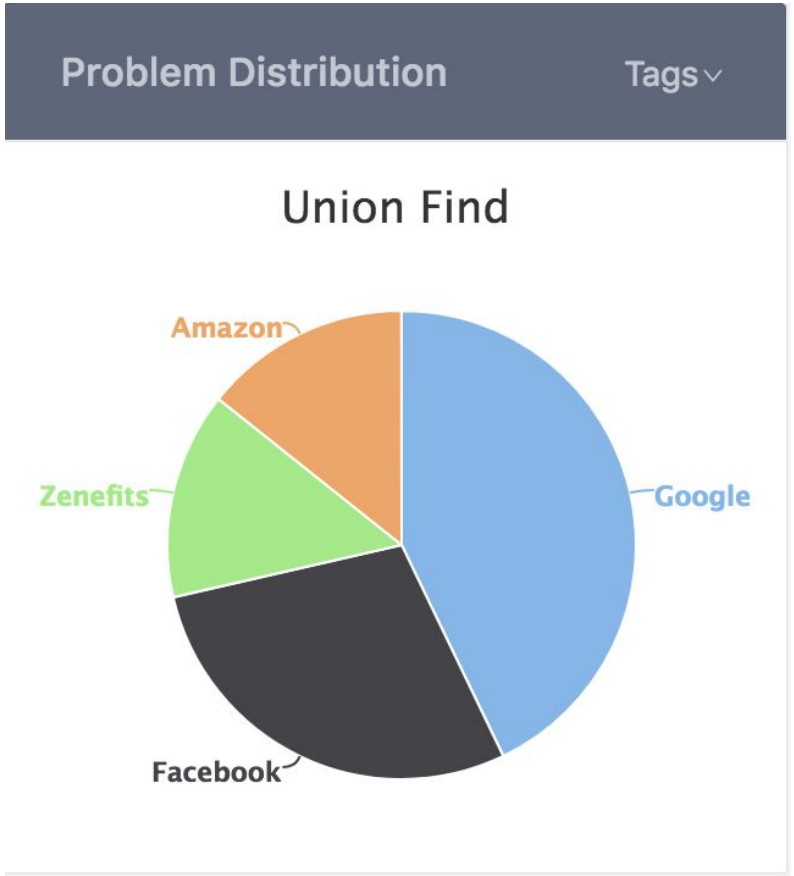
面试大厂的必备数据结构

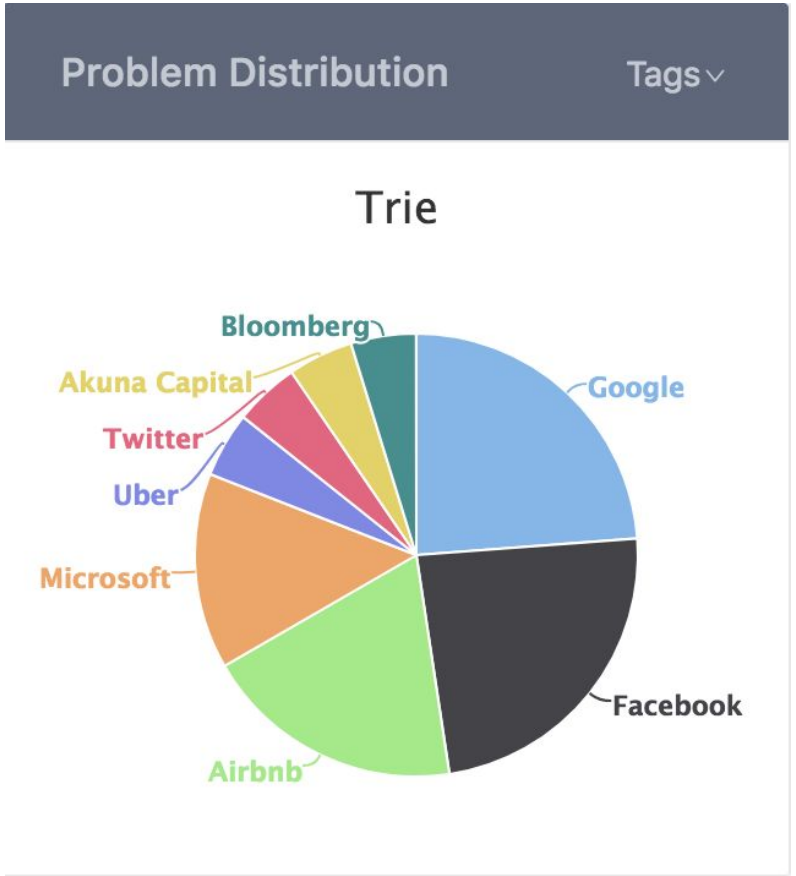
并查集 Union Find

- 可以解决什么问题
- 代码模板
- 例题

字典树 Trie

- 可以解决什么问题
- 代码模板
- 例题





并查集 Union Find

一种用于支持集合**快速合并和查找**操作的数据结构

O(1) 合并两个集合 - Union

O(1) 查询元素所属集合 - Find

并查集

公司并购 —— 合并两个集合

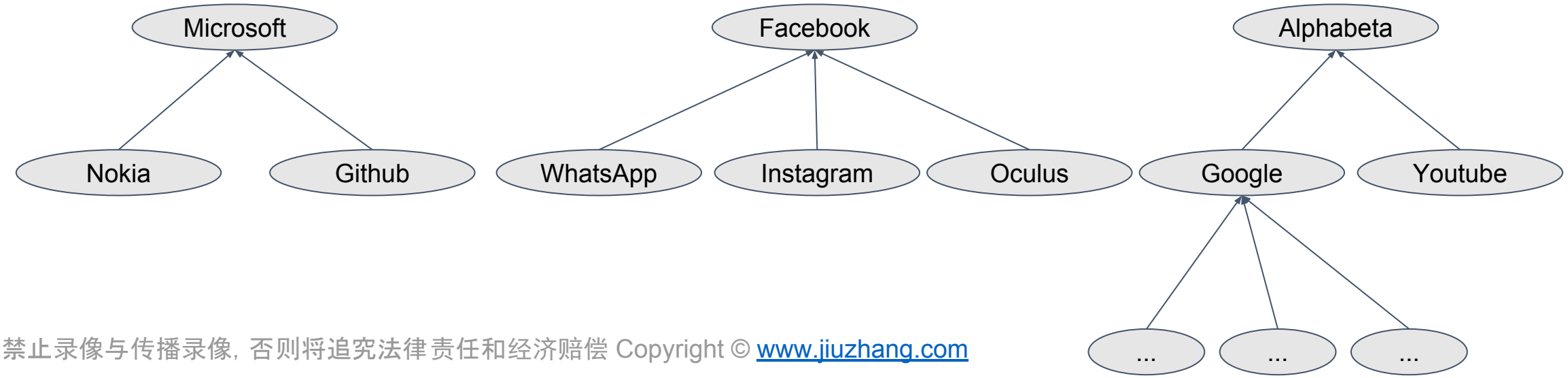
查询所在公司 —— 查询所在集合

判断两个员工是否在同一家公司

—— 判断两个员工所在集合是否相同

Union Find 是一棵多叉树

画图演示



底层数据结构

- 父亲表示法，用一个数组/哈希表记录每个节点的父亲是谁。
- father[“Nokia”] = “Microsoft”
- father[“Instagram”] = “Facebook”

查询所在集合

- 用所在集合最顶层的老大哥节点来代表这个集合

合并两个集合

- 找到两个集合中最顶层的两个老大哥节点 A 和 B
- father[A] = B // or father[B] = A 如果无所谓谁合并谁的话

代码模板 - 初始化

使用哈希表或者数组来存储每个节点的父亲节点

如果节点不是连续整数的话，就最好用哈希表来存储

最开始所有的父亲节点都指向自己

```
def __init__(self, n):
    self.father = {}
    for i in range(1, n + 1):
        self.father[i] = i
```

注：

也有的方法中是将父亲节点指向空

虽然可行但是没有指向自己操作起来那么方便

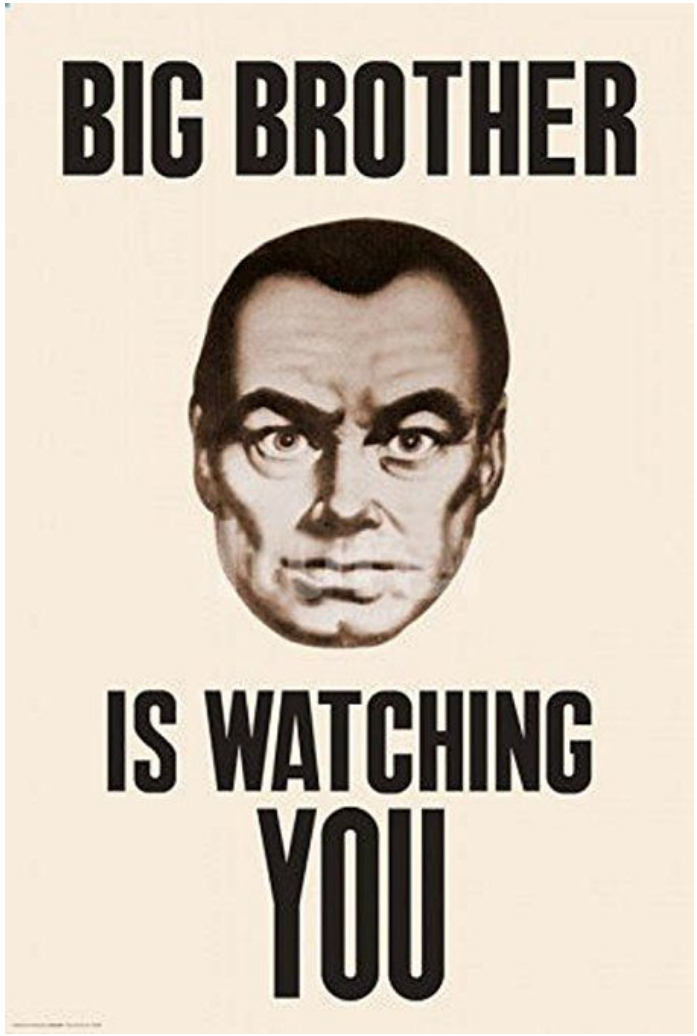
代码模板 - 查找老大哥

沿着父亲节点一路往上走就能找到老大哥

下面这份代码有什么问题？

```
def find(self, node):
    while self.father[node] != node:
        node = self.father[node]

    return node
```

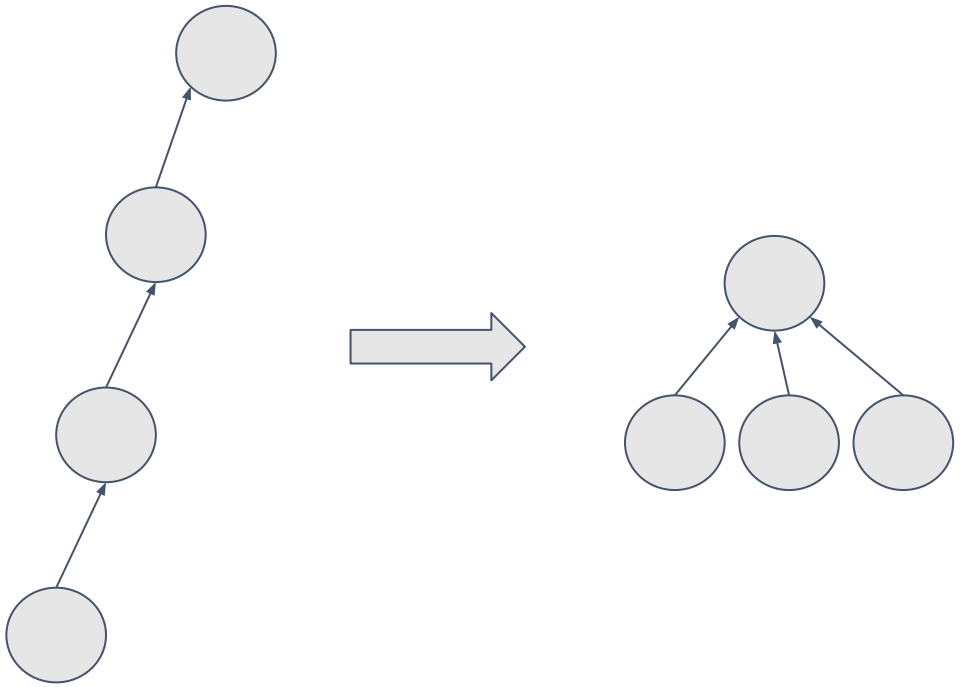


代码模板 - 路径压缩

路径压缩 —— 在找到老大哥以后，还需要把一路上经过的点都指向老大哥

分为两种实现方法：递归 vs 非递归

推荐非递归



```python
def find(self, node):
    if node == self.father[node]:
        return node

    self.father[node] = self.find(self.father[node])
    return self.father[node]
```

```python
def find(self, node):
    path = []
    while self.father[node] != node:
        path.append(node)
        node = self.father[node]

    for n in path:
        self.father[n] = node

    return node
```

找到两个元素所在集合的两个老大哥 A 和 B

将其中一个老大哥的父指针指向另外一个老大哥

```
def union(self, a, b):
    self.father[self.find(a)] = self.find(b)
```

完整代码


```python
class UnionFind:

    def __init__(self, n):
        self.father = {}
        for i in range(1, n + 1):
            self.father[i] = i

    def union(self, a, b):
        self.father[self.find(a)] = self.find(b)

    def find(self, node):
        path = []
        while node != self.father[node]:
            path.append(node)
            node = self.father[node]

        for n in path:
            self.father[n] = node

        return node
```

x	$\lg^* x$
$(-\infty, 1]$	0
$(1, 2]$	1
$(2, 4]$	2
$(4, 16]$	3
$(16, 65536]$	4
$(65536, 2^{65536}]$	5

时间复杂度

都是 O(log* n)，约等于 O(1)

https://en.wikipedia.org/wiki/Proof_of_O(log*n)_time_complexity_of_union%E2%80%93find

https://en.wikipedia.org/wiki/Iterated_logarithm

Connecting Graph

http://www.lintcode.com/problem/connecting-graph/

https://www.jiuzhang.com/solution/connecting-graph/

判断两个点是否在一个集合

Connecting Graph II

http://www.lintcode.com/problem/connecting-graph-ii/

https://www.jiuzhang.com/solution/connecting-graph-ii/

获得某个集合的元素个数

Connecting Graph III

http://www.lintcode.com/problem/connecting-graph-iii/

https://www.jiuzhang.com/solution/connecting-graph-iii/

获得集合总数

并查集可以做的事情总结

1. 合并两个集合
2. 查询某个元素所在集合
3. 判断两个元素是否在同一个集合
4. 获得某个集合的元素个数
5. 统计当前集合个数

Number of Islands II

https://www.lintcode.com/problem/number-of-islands-ii/

https://www.jiuzhang.com/solution/number-of-islands-ii/

离线算法(BFS) vs 在线算法(UnionFind)

Graph Valid Tree

https://www.lintcode.com/problem/graph-valid-tree/

https://www.jiuzhang.com/solution/graph-valid-tree/

跟连通性有关的问题

都可以使用 BFS 和 Union Find

什么时候无法使用 Union Find?

跟连通性有关的问题

都可以使用 BFS 和 Union Find

什么时候无法使用 Union Find?

需要拆开两个集合的时候无法使用Union Find

Accounts Merge

https://www.lintcode.com/problem/accounts-merge/

http://www.jiuzhang.com/solution/accounts-merge/

其他 Union Find 的练习题

https://www.lintcode.com/problem/set-union/

类似 Accounts Merge

https://www.lintcode.com/problem/surrounded-regions/

用 BFS 来做最简便，但是也可以作为 Union Find 的练习

https://www.lintcode.com/problem/maximum-association-set

找到最大的关联集合，在集合合并的时候打擂台即可

字典树 Trie

又名 Prefix Tree

来自单词 Retrieval，发音与 Tree 相同

Trie 的考点

实现一个 Trie

比较 Trie 和 Hash 的优劣

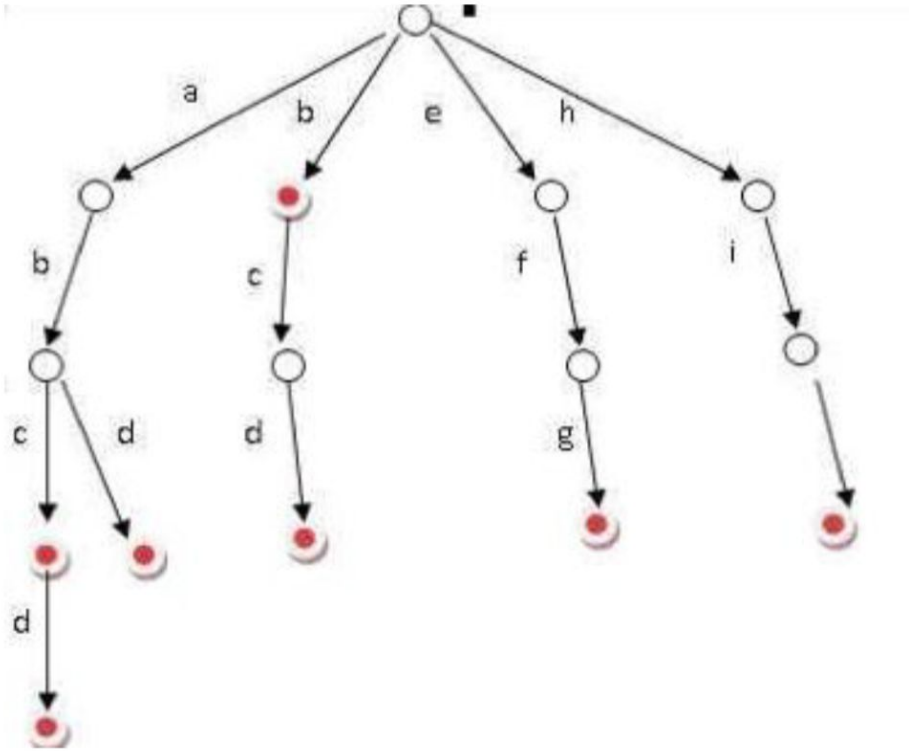
字符矩阵类问题使用 Trie 比 Hash 更高效

Implement Trie

https://www.lintcode.com/problem/implement-trie/

https://www.jiuzhang.com/solution/implement-trie/

假设有 [b, abc, abd, bcd, abcd, efg, hii] 这6个单词，查找abc 在不在字典里面



Implement Trie - 代码模板

需要一个新的类 TrieNode 代表 Trie 中的节点

```python
class TrieNode:

    def __init__(self):
        self.children = {}
        self.is_word = False
```

insert 插入一个单词

find 找到这个单词所在的 TrieNode

- 如果没有返回 None

```python
class Trie:

    def __init__(self):
        self.root = TrieNode()

    def insert(self, word):
        node = self.root
        for c in word:
            if c not in node.children:
                node.children[c] = TrieNode()
            node = node.children[c]

        node.is_word = True

    def find(self, word):
        node = self.root
        for c in word:
            node = node.children.get(c)
            if node is None:
                return None
        return node
```

Hash vs Trie

互相可替代

Trie 耗费更少的空间

单次查询 Trie 耗费更多的时间

（复杂度相同，Trie 系数大一些）

Add and Search Word

https://www.lintcode.com/problem/add-and-search-word/

https://www.jiuzhang.com/solution/add-and-search-word/

如果处理 “.” ？

Word Squares

https://www.lintcode.com/problem/word-squares/

https://www.jiuzhang.com/solution/word-squares/

如何剪枝？

Word Search II

https://www.lintcode.com/problem/word-search-ii/

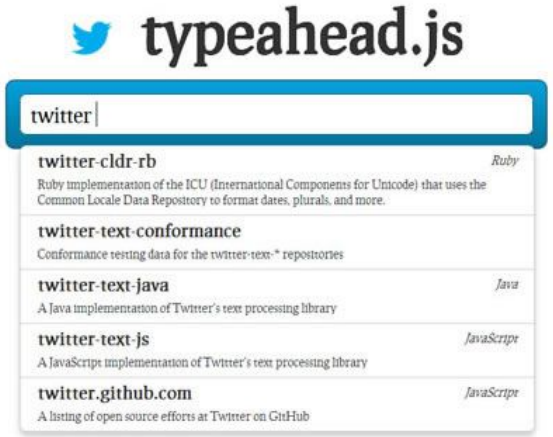
https://www.jiuzhang.com/solution/word-search-ii/

这个问题上使用 Hash 和 Trie 相比，时间复杂度是否相同？

Typeahead

Trie 在系统设计中的运用

请报名《系统设计班》学习



Q & A

常见问题 http://www.jiuzhang.com/qa/3/

扫描二维码关注微信小程序/公众号
获取第一手求职资料