极客大学算法训练营
期末复习

覃超

过遍数 —> 脑图记忆

- 书先是越读越厚，然后越读越薄
- 看山是山，看山不是山，看山還是山

数据结构

- 一维：
  - 基础：数组 array (string), 链表 linked list
  - 高级：栈 stack, 队列 queue, 双端队列 deque, 集合 set, 映射 map (hash or map), etc. TreeMap, HashMap
- 二维：
  - 基础：树 tree, 图 graph
  - 高级：二叉搜索树 binary search tree (red-black tree, AVL), 堆 heap, 并查集 disjoint set, 字典树 Trie, etc
- 特殊：
  - 位运算 Bitwise, 布隆过滤器 BloomFilter
  - LRU Cache

时间复杂度

Data Structure	Time Complexity								Space Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

- https://www.bigocheatsheet.com/

算法

- If-else, switch —> branch
- for, while loop —> Iteration
- 递归 Recursion (Divide & Conquer, Backtrace)
- 搜索 Search: 深度优先搜索 Depth first search, 广度优先搜索 Breadth first search, A*, etc
- 动态规划 Dynamic Programming
- 二分查找 Binary Search
- 贪心 Greedy
- 数学 Math , 几何 Geometry

注意：在头脑中回忆上面每种算法的思想和代码模板

算法模板 + 脑图总结

- 模板：https://shimo.im/folder/WjP9V3HWpJYhCTKr
- 优秀学员的脑图示例
- 建议放在手机、电脑、云上，经常翻看、反复记忆。

化繁为简的思想

1. 人肉递归低效、很累 —> 画出递归的状态树
   https://leetcode-cn.com/problems/climbing-stairs/solution/pa-lou-ti-by-leetcode/
   https://leetcode-cn.com/problems/coin-change/solution/322-ling-qian-dui-huan-by-leetcode-solution/
   https://leetcode-cn.com/problems/permutations/solution/

2. 找到最近最简方法，将其拆解成可重复解决的问题

3. 数学归纳法思维

本质：寻找重复性 —> 计算机指令集

学习要点

- 基本功是区别业余和职业选手的根本。深厚功底来自于 — 过遍数
- **最大的误区：只做一遍**
- 五毒神掌
- 刻意练习 - 练习缺陷弱点地方、不舒服、枯燥
- 反馈 - 看题解、看国际版的高票回答

五毒神掌

第一遍：不要死磕，要看代码学习（一定要看国际版的高票回答）

第二遍：自己写

第三遍：24小时后

第四遍：一周后（时间紧的话，就只需要看脑图）

第五遍：面试前

面试技巧 - 周四模拟面试

1. Clarification：明确题目意思、边界、数据规模

2. Possible solutions：穷尽所有可能的解法
 - compare time/space
 - optimal solution

3. Coding：代码简洁、高性能、美感
 https://shimo.im/docs/rHTyt8hcpT6D9Tj8

4. Test cases
normal case / corner case / failure case / stress case

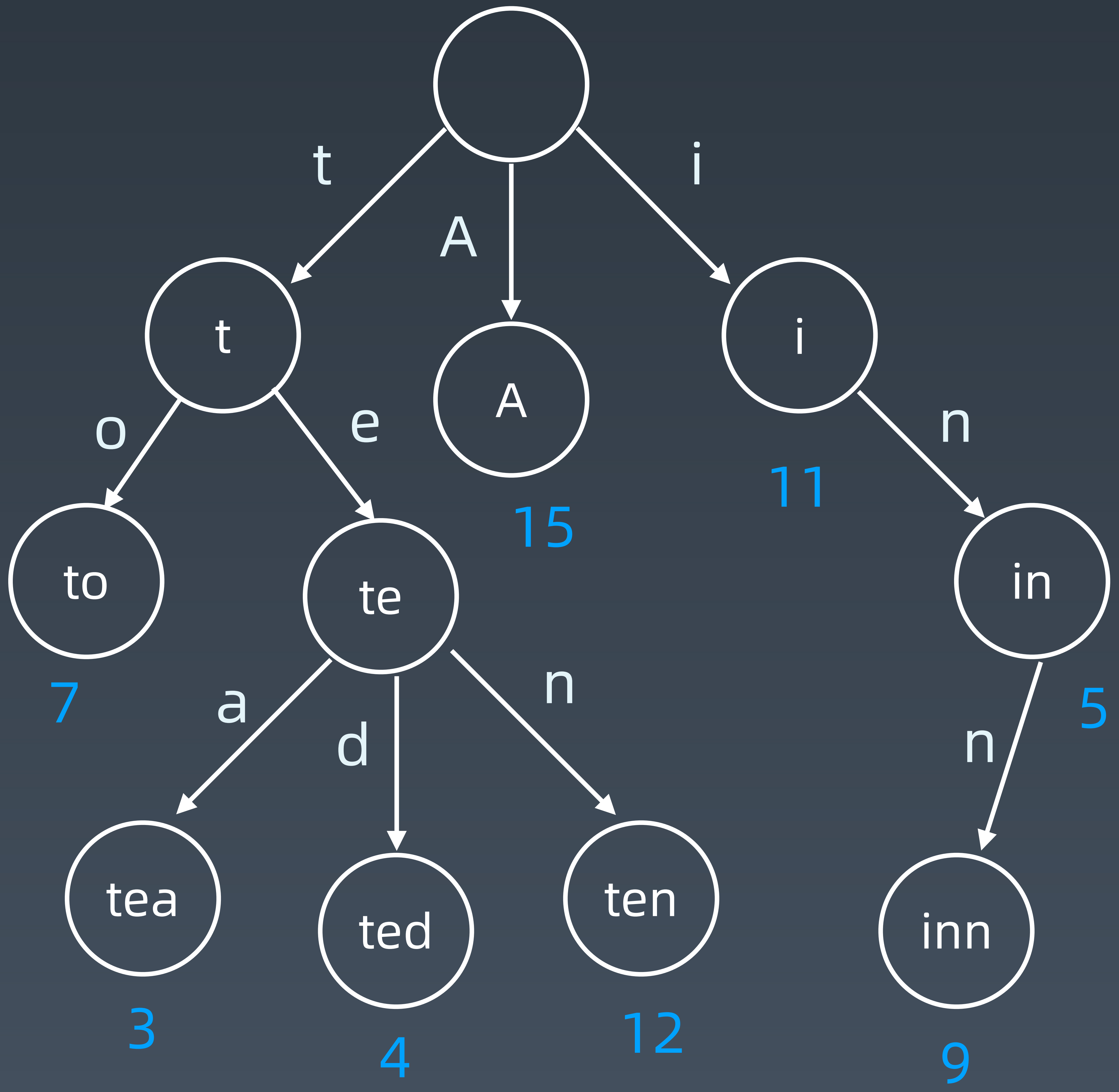
调整对于面试的观念

- 作为和未来同事的一次合作；
- 并肩作战、解决问题 –> 所以沟通、配合、互相肯定很重要；

  （多想想王者荣耀、LOL里的五人局）
- 减少压力。

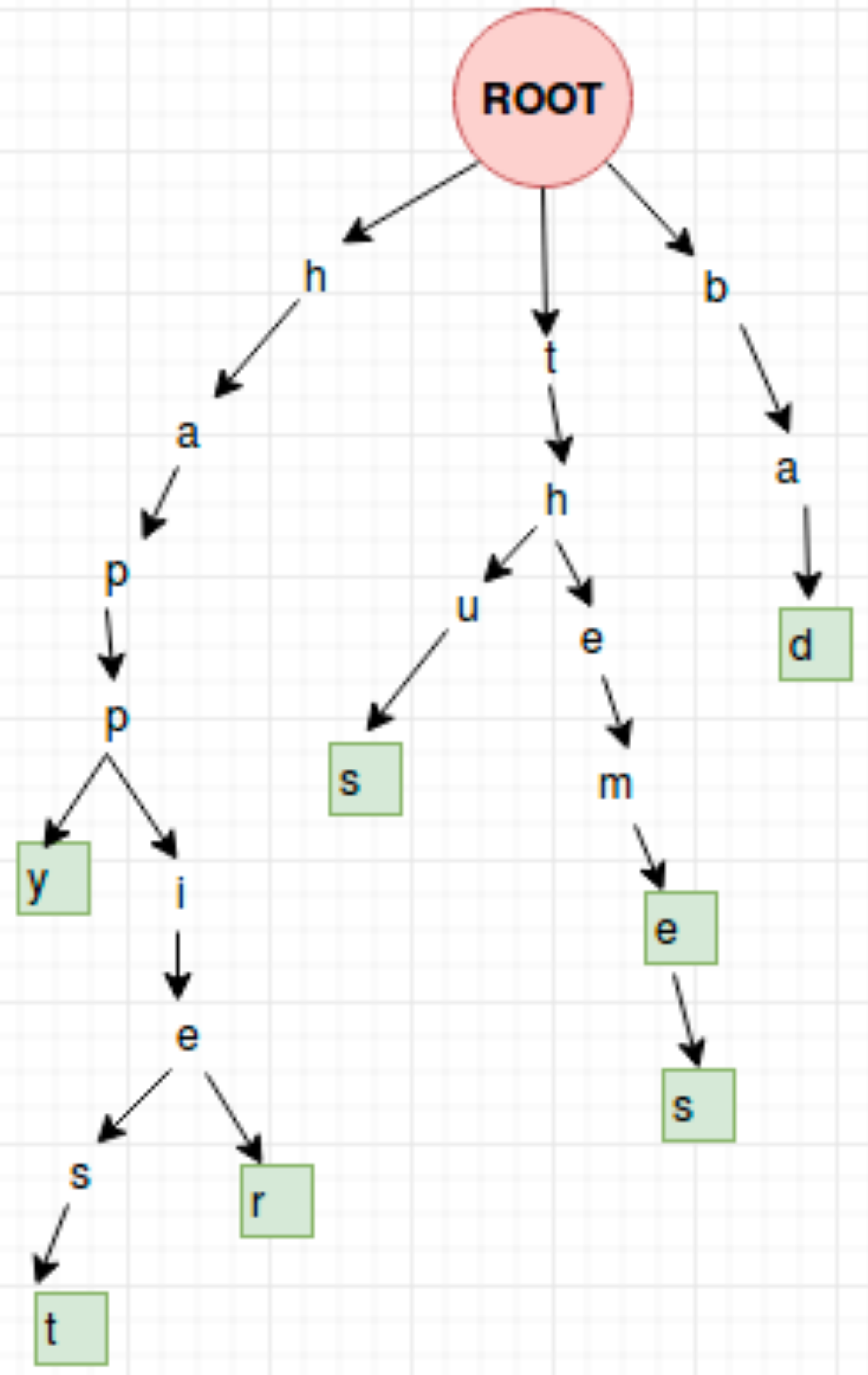
高级数据结构

Trie - 中文



Keywords

happy
happier
happiest
theme
themes
thus
bad



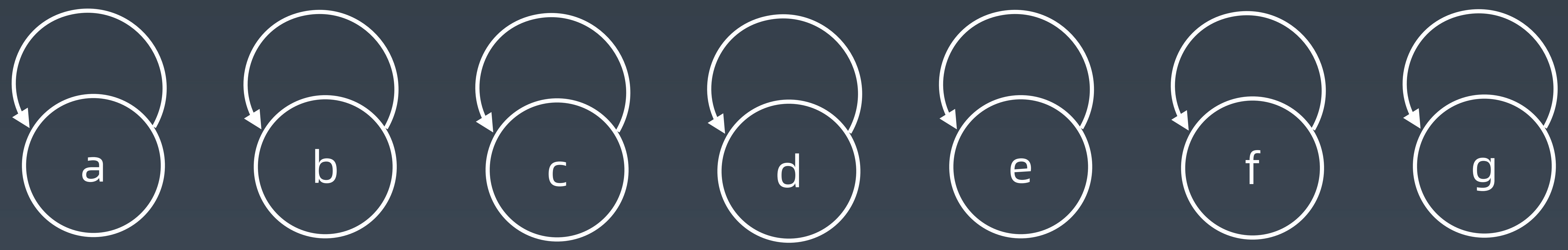
```python
class TrieNode:
    def __init__(self):
        self.children = defaultdict(TrieNode)
        self.is_word = False

class Trie(object):

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

    def insert(self, word):
        cur = self.root
        for letter  in word:
            cur = cur.children[letter]
        cur.is_word = True

    def search(self, word):
        cur = self.root
        for letter in word:
            cur = cur.children.get(letter)
            if cur is  None:
                return False
        return cur.is_word
```

```java
class Trie {
    private Node root;

    private class Node {
        private TreeMap<Character, Node> next;
        private boolean isWord;

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

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

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

    public void insert(String word) {
        Node cur = root;
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            cur.next.putIfAbsent(c,new Node());
            cur = cur.next.get(c);
        }
        if (!cur.isWord) {
            cur.isWord = true;
        }
    }

    public boolean search(String word) {
        Node cur = root;
        for (int i = 0; i < word.length(); i++) {
            char c = word.charAt(i);
            if (!cur.next.containsKey(c)) {
                return false;
            } else {
                cur = cur.next.get(c);
            }
        }
        return cur.isWord;
    }
```

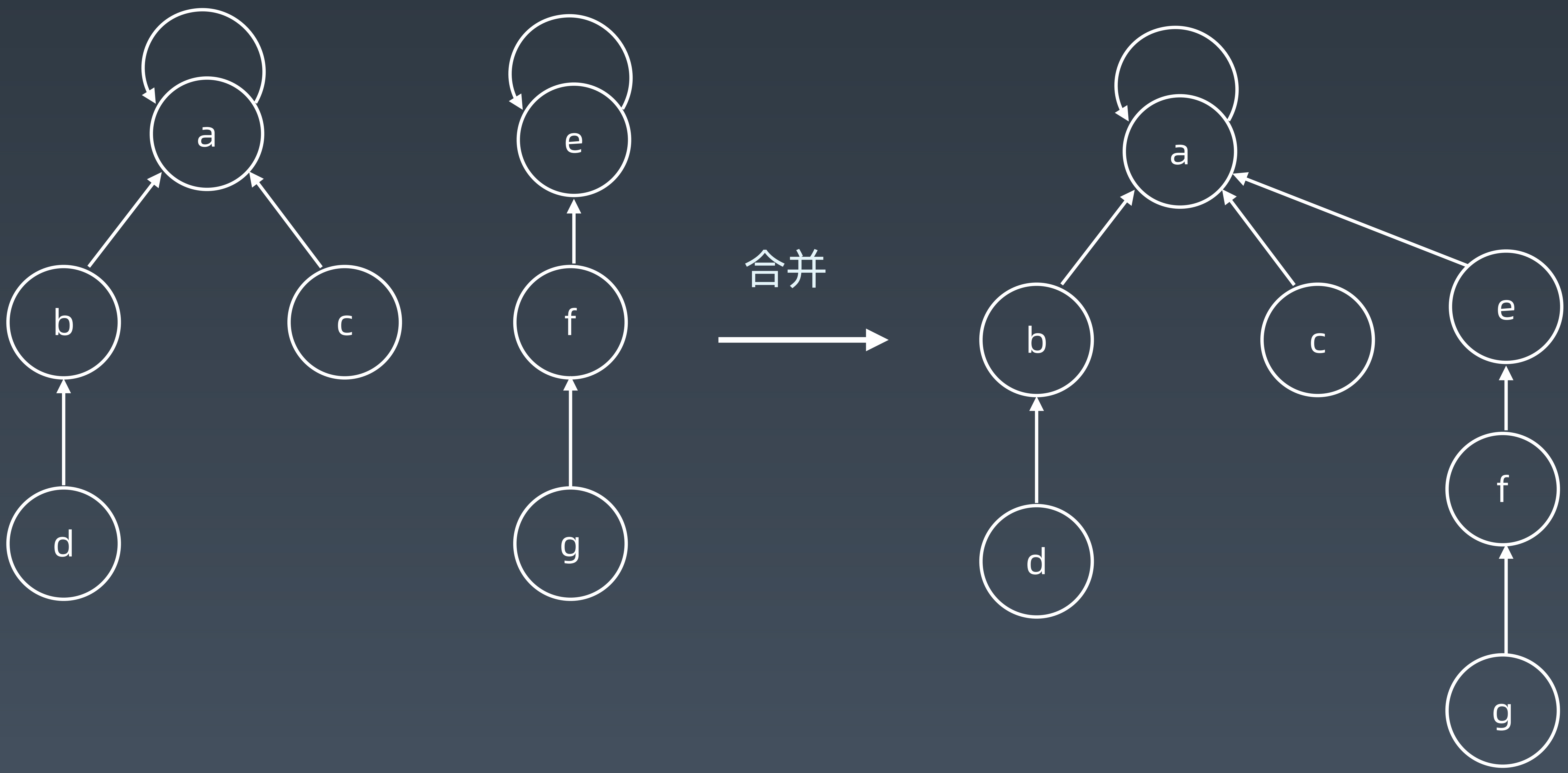
```java
class Trie {

    class TireNode {
        private boolean isEnd;
        TireNode[] next;

        public TireNode() {
            isEnd = false;
            next = new TireNode[26];
        }
    }

    private TireNode root;

    public Trie() {
        root = new TireNode();
    }

    public void insert(String word) {
        TireNode node = root;
        for (char c : word.toCharArray()) {
            if (node.next[c - 'a'] == null) {
                node.next[c - 'a'] = new TireNode();
            }
            node = node.next[c - 'a'];
        }
        node.isEnd = true;
    }

    public boolean search(String word) {
        TireNode node = root;
        for (char c : word.toCharArray()) {
            node = node.next[c - 'a'];
            if (node == null) {
                return false;
            }
        }
        return node.isEnd;
    }
```

并查集
Disjoint Set

初始化



查询、合并



Java 实现

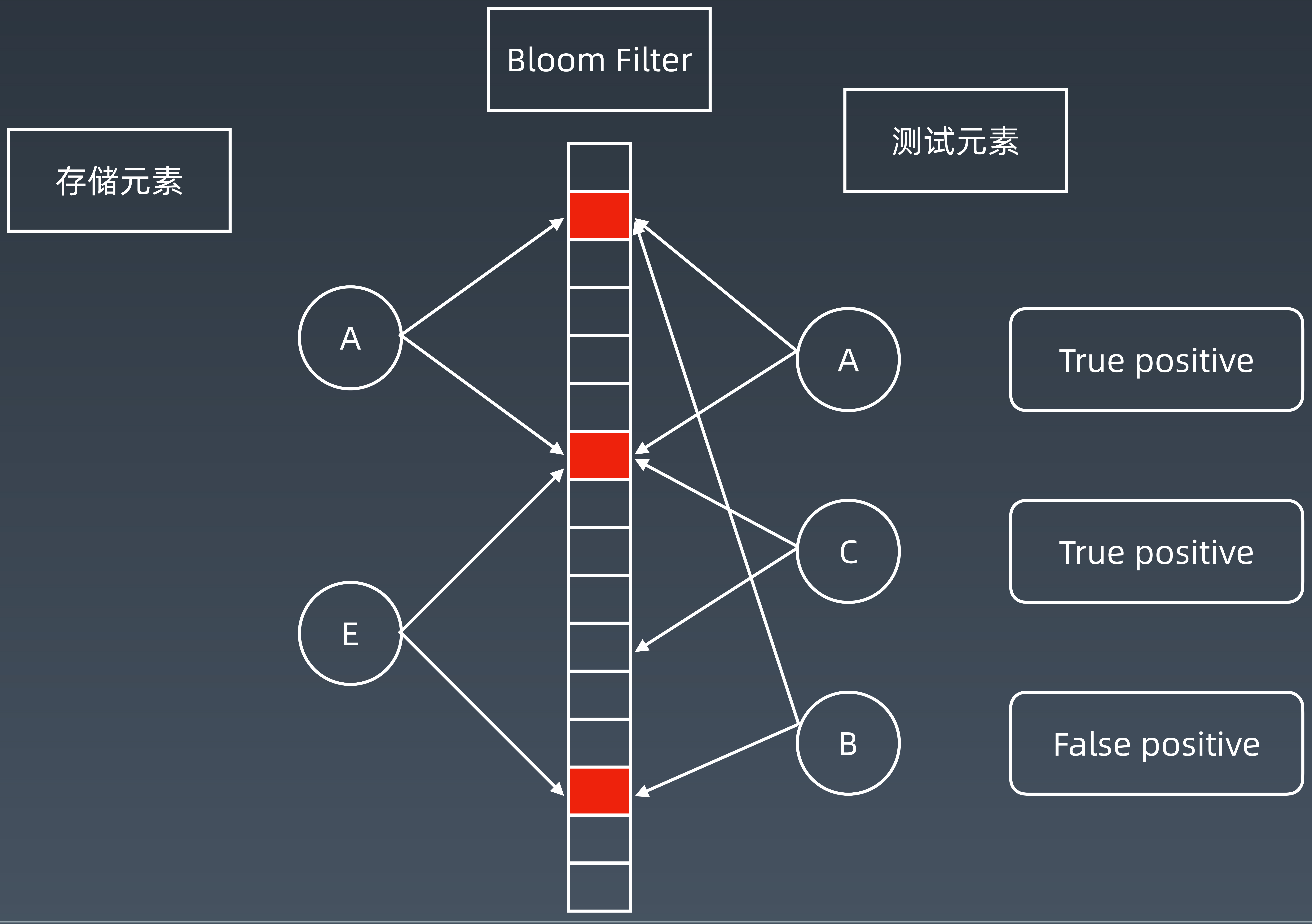
```java
class UnionFind {
  private int count = 0;
  private int[] parent;
  public UnionFind(int n) {
    count = n;
    parent = new int[n];
    for (int i = 0; i < n; i++) {
      parent[i] = i;
    }
  }
  public int find(int p) {
    while (p != parent[p]) {
      parent[p] = parent[parent[p]];
      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--;
  }
}
```

Python实现

```python
def init(p):
  # for i = 0 .. n: p[i] = i;
  p = [i for i in range(n)]

def union(self, p, i, j):
  p1 = self.parent(p, i)
  p2 = self.parent(p, j)
  p[p1] = p2

def parent(self, p, i):
  root = i
  while p[root] != root:
    root = p[root]
  while p[i] != i:
    x = i; i = p[i]; p[x] = root
  return root
```

布隆过滤器

Bloom Filter


Bloom Filter
存储元素
测试元素
A
E
A
C
B
True positive
True positive
False positive

LRU Cache

LRU Cache

- Hash Table + Double LinkedList
- O(1) 查询
  O(1) 修改、更新
- LRU (least recently used)
  FIFO/FILO/LRU/LFU/LFRU/ARC

  https://leetcode-cn.com/problems/lru-cache-lcci/

位运算

指定位置的位运算

1. 将 x 最右边的 n 位清零：x & (~0 << n)
2. 获取 x 的第 n 位值（0 或者 1）： (x >> n) & 1
3. 获取 x 的第 n 位的幂值：x & (1 << n)
4. 仅将第 n 位置为 1：x | (1 << n)
5. 仅将第 n 位置为 0：x & (~ (1 << n))

实战位运算要点

- 判断奇偶：
  x % 2 == 1 —> (x & 1) == 1
  x % 2 == 0 —> (x & 1) == 0

- x >> 1 —> x / 2.
  即： x = x / 2; —> x = x >> 1;
  mid = (left + right) / 2; —> mid = (left + right) >> 1;

- X = X & (X-1) 清零最低位的 1: x = 01101000, x-1=

- X & -X => 得到最低位的 1

- X & ~X => 0

N皇后问题

Java

```
class Solution {
  private int size;
  private int count;

  private void solve(int row, int ld, int rd) {
    if (row == size) {
      count++;
      return;
    }
    int pos = size & (~(row | ld | rd));
    while (pos != 0) {
      int p = pos & (-pos);
      pos -= p; // pos = pos & (pos - 1);    去掉最低位的1
      solve(row | p, (ld | p) << 1, (rd | p) >> 1);
    }
  }

  public int totalNQueens(int n) {
    count = 0;
    size = (1 << n) - 1;
    solve(0, 0, 0);
    return count;
  }
}
```

高级二叉搜索树

AVL，2-3树，红黑树，B树，B+树

1. https://juejin.im/post/6844903859974848520

2. https://zhuanlan.zhihu.com/p/27700617

递归、分治、回溯、动态规划复习

递归 - 函数自己调用自己

```
public void recur(int level, int param) {

    // terminator
    if (level > MAX_LEVEL) {
      // process result
      return;
    }

    // process current logic
    process(level, param);

    // drill down
    recur( level: level + 1, newParam);

    // restore current status

}
```

分而治之

Divide & Conquer

分治代码模板

```
def divide_conquer(problem, param1, param2, ...):
  # recursion terminator
  if problem is None:
    print_result
    return

  # prepare data
  data = prepare_data(problem)
  subproblems = split_problem(problem, data)

  # conquer subproblems
  subresult1 = self.divide_conquer(subproblems[0], p1, ...)
  subresult2 = self.divide_conquer(subproblems[1], p1, ...)
  subresult3 = self.divide_conquer(subproblems[2], p1, ...)
  …

  # process and generate the final result
  result = process_result(subresult1, subresult2, subresult3, …)

  # revert the current level states
```

感触

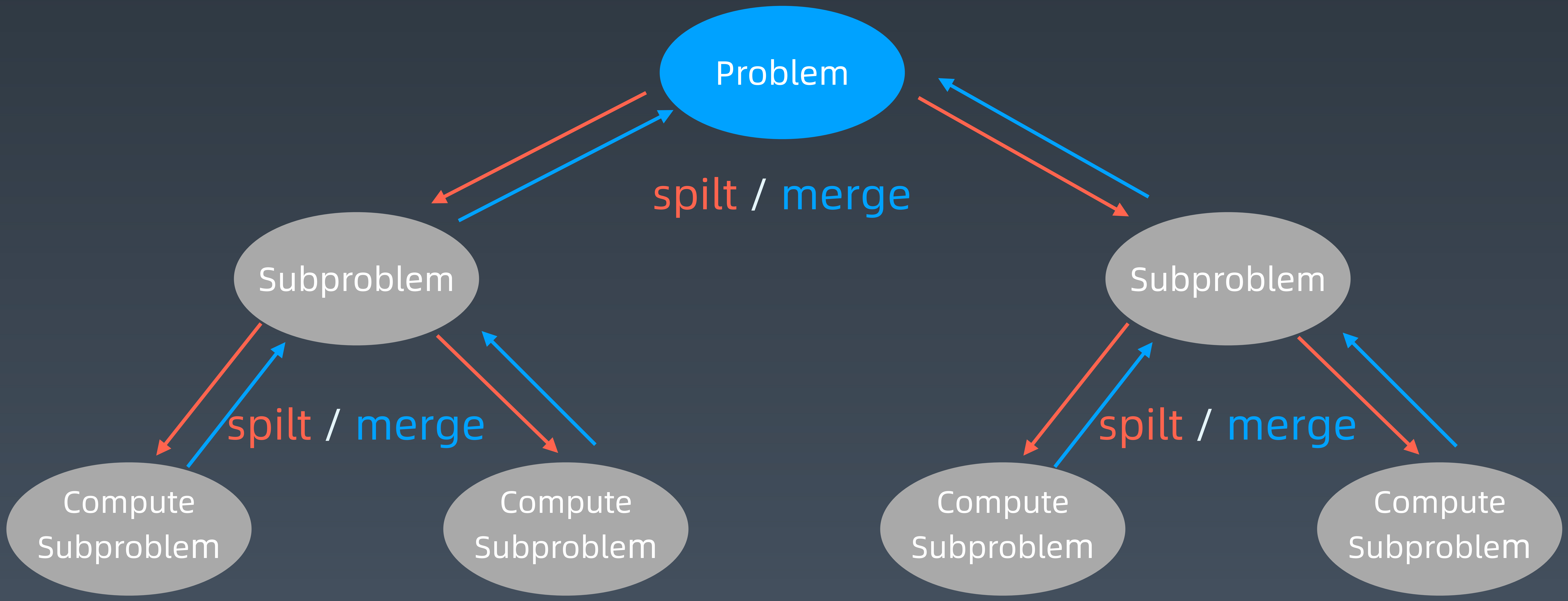
1. 当发现问题复杂时 —> “人肉递归低效”

2. 找到最近最简方法，将其拆解成可重复解决的问题

3. 数学归纳法思维

本质：寻找重复性 —> 计算机指令集

递归状态树



Problem
spilt / merge
Subproblem
Subproblem
spilt / merge
spilt / merge
Compute Subproblem
Compute Subproblem
Compute Subproblem
Compute Subproblem

Fib(6) 状态树、重复子状态

动态规划 Dynamic Programming

1. “Simplifying a complicated problem by breaking it down into simpler sub-problems”
   (in a recursive manner)

2. Divide & Conquer + Optimal substructure
   分治 + 最优子结构

3. 顺推形式： 动态递推

DP模板

```
function DP():

  Dp = [][] # 三维情况

  For I = 0 .. M {
    For j = 0 .. N {
       Dp[i][j] = _Function(Dp[i’][j’]…)
    }
  }

  Return Dp[M][N]
```

关键点

动态规划 和 递归或者分治 没有根本上的区别（关键看有无最优的子结构）

拥有共性：找到重复子问题

差异性：最优子结构、中途可以淘汰次优解

中场休息

常见的DP题目和状态方程

LC 70. Climbing Stairs

1D, Counting

Recursion formula:

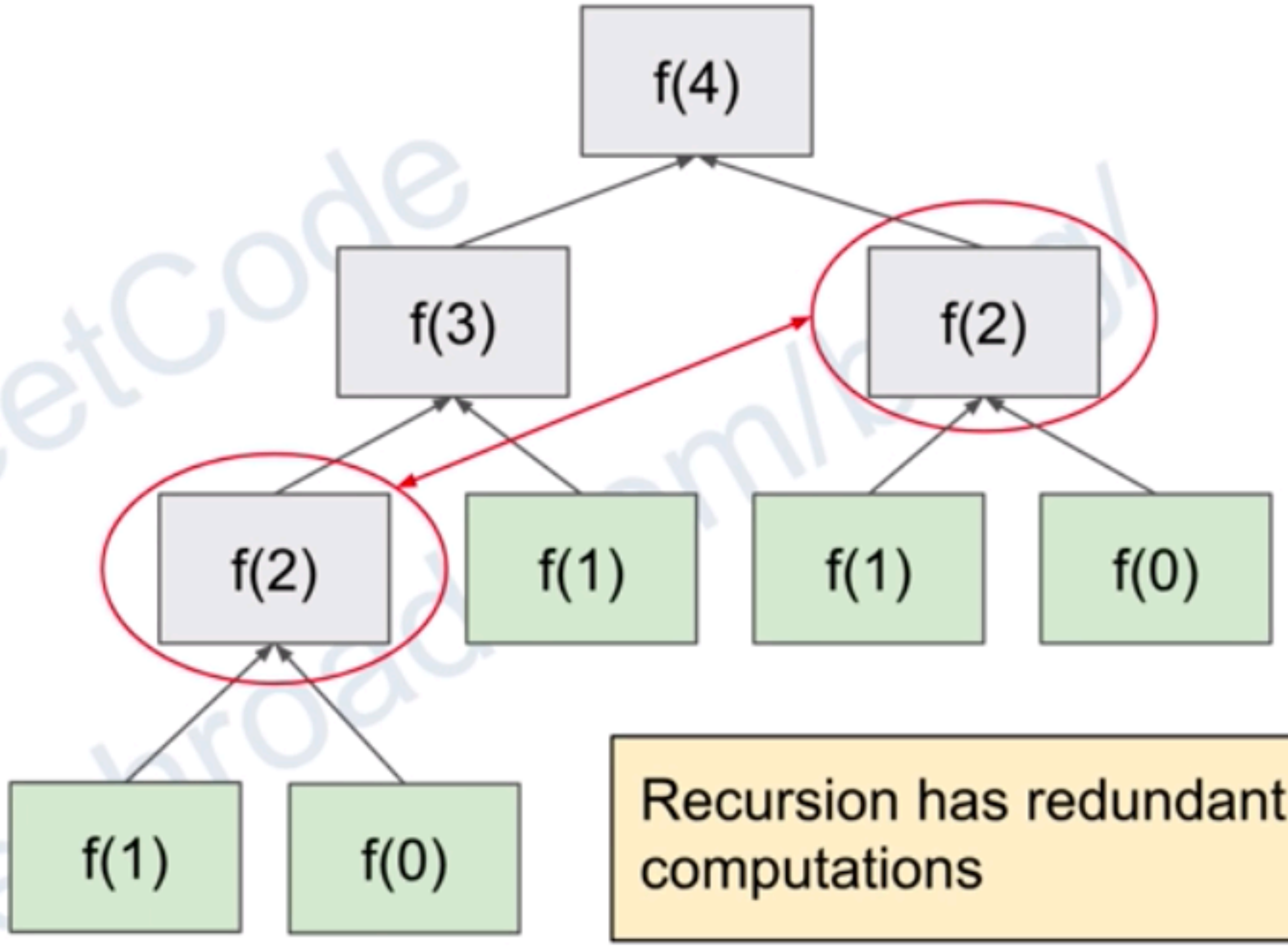
```
f(n) = f(n - 1) + f(n - 2)
f(1) = 1, f(0) = 1
```

```
def f(n):
  if n <= 1: return 1
  return f(n - 1) + f(n - 2)
```

$O(2^n)$, $O(n)$

```
def f(n):
  if n <= 1: return 1
  if n not in mem:
    mem[n] = f(n - 1) + f(n - 2)
  return mem[n]
```

$O(n)$, $O(n)$



Recursion has redundant computations

```
def f(n):
  dp = [1] * (n + 1)
  for i in range(2, n + 1):
    dp[i] = dp[i - 1] + dp[i - 2]
  return dp[n]
```

$O(n)$, $O(n)$

One loop

```
def f(n):
  dp1, dp2 = 1, 1
  for i in range(2, n + 1):
    dp2, dp1 = dp1 + dp2, dp2
  return dp2
```

$O(n)$, $O(1)$

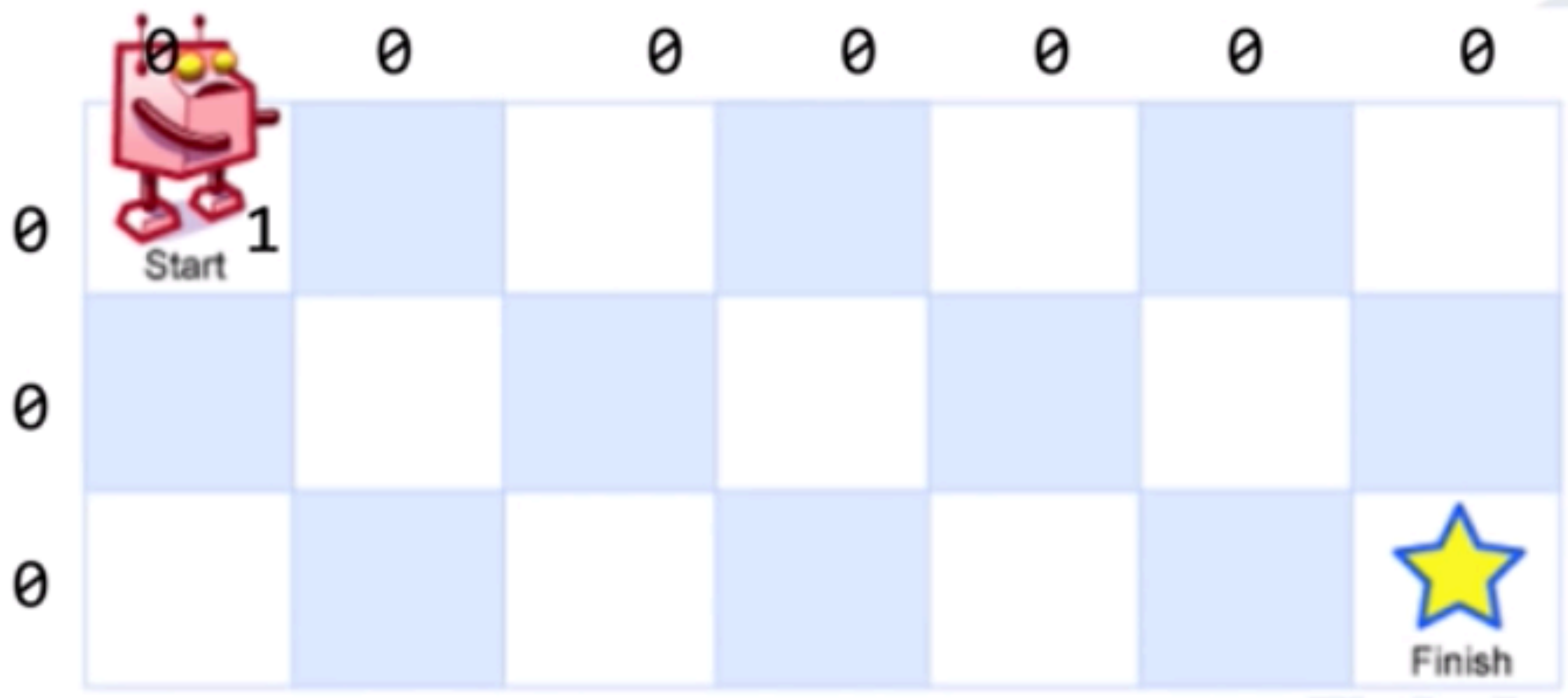
One param, no loop

LC 62. Unique Paths

2D, Counting

Recursion formula:

```
f(x, y) = f(x - 1, y) + f(x, y - 1)
f(0, 0) = 1, out of board = 0
```



DP: paddings required to handle out of board cases.
Actual indies start from 1 instead of 0

```
def f(x, y):
  if x <= 0 or y <= 0: return 0
  if x == 1 and y == 1: return 1
  return f(x-1, y) + f(x, y-1)
```

```
def f(x, y):
  if x <= 0 or y <= 0: return 0
  if x == 1 and y == 1: return 1
  if (x, y) not in mem:
    mem[(x, y)] = f(x-1, y) + f(x, y-1)
  return mem[(x, y)]
```

O(mn), O(mn)

2 params, No loops

Two loops

```
def f(x, y):
  dp = [[0] * (m+1) for _ in range(n+1)]
  dp[1][1] = 1
  for i in range(1, y+1):
    for j in range(1, x+1):
      dp[i][j] = dp[i-1][j] + dp[j][i-1]
  return dp[y][x]
```

O(mn), O(mn)

```
LC 64. Minimum Path Sum
dp[i][j] := minPath(A[0-i][0->j])
dp[i][j] = min(dp[i-1][j], dp[i][j-1]) + A[i][j]
```

```
LC 198 House Robber
dp[i] := max $ of robbing A[0->i]
dp[i]  = max(dp[i - 2] + A[i], dp[i - 1])
```

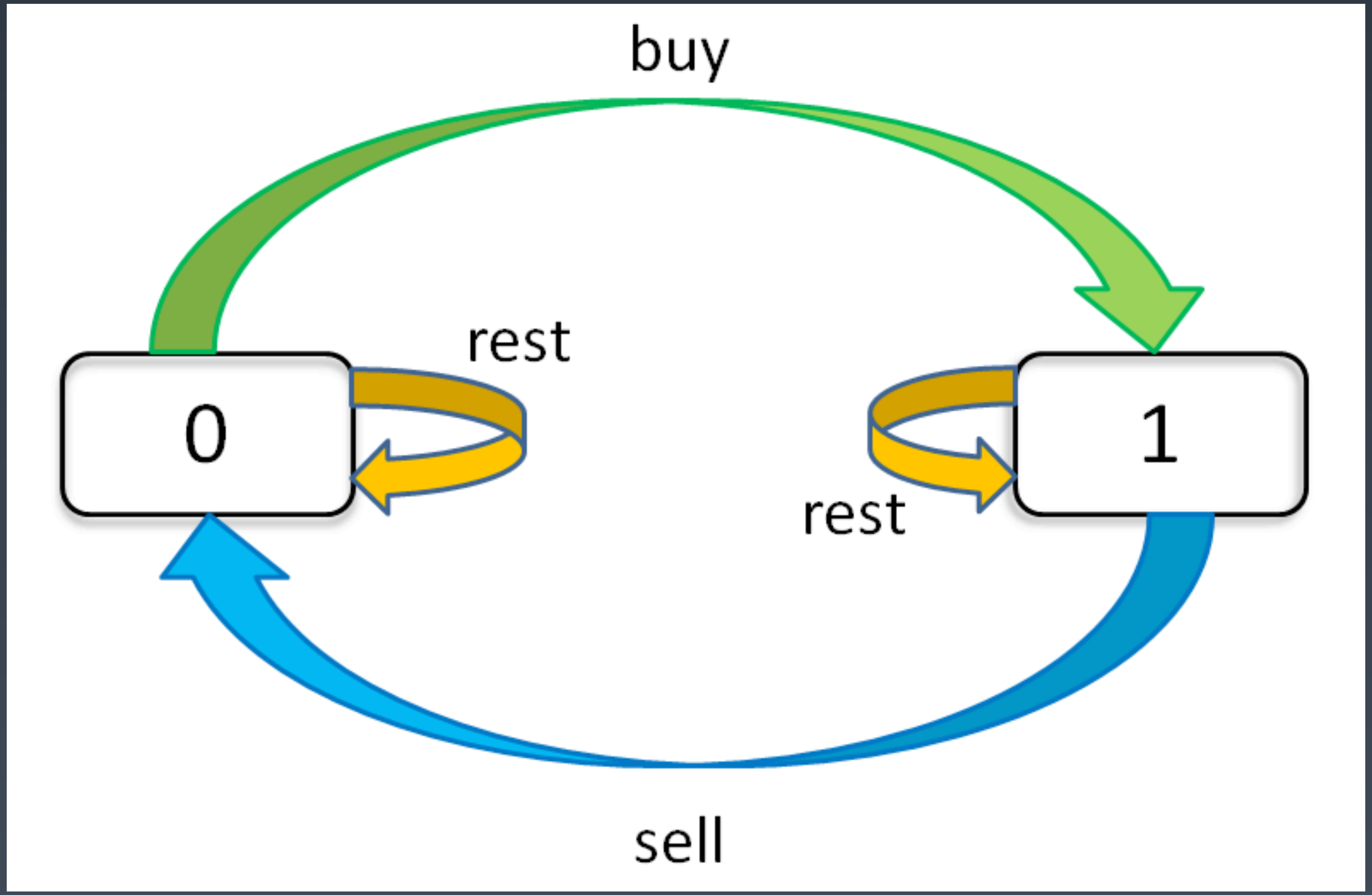
```
LC 198 House Robber
dp[i][0] := max $ of robbing A[0->i] w/o A[i]
dp[i][1] := max $ of robbing A[0->i] w/ A[i]
dp[i][0] = max(dp[i - 1][0], dp[i - 1][1]);
dp[i][1] = max(dp[i - 2][0], dp[i - 2][1])
           + nums[i - 1];
```

股票买卖

```
dp[i][k][0 or 1]
0 <= i <= n-1, 1 <= k <= K
n 为天数，大 K 为最多交易数
此问题共 n × K × 2 种状态，全部穷举就能搞定。

for 0 <= i < n:
    for 1 <= k <= K:
        for s in {0, 1}:
            dp[i][k][s] = max(buy, sell, rest)
```

https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock/solution/yi-ge-fang-fa-tuan-mie-6-dao-gu-piao-wen-ti-by-l-3/

股票买卖



https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock/solution/yi-ge-fang-fa-tuan-mie-6-dao-gu-piao-wen-ti-by-l-3/

股票买卖

```
dp[i][k][0] = max(dp[i-1][k][0], dp[i-1][k][1] + prices[i])
              max(   选择 rest  ,           选择 sell      )

解释：今天我没有持有股票，有两种可能：
要么是我昨天就没有持有，然后今天选择 rest，所以我今天还是没有持有；
要么是我昨天持有股票，但是今天我 sell 了，所以我今天没有持有股票了。

dp[i][k][1] = max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i])
              max(   选择 rest  ,           选择 buy         )

解释：今天我持有着股票，有两种可能：
要么我昨天就持有着股票，然后今天选择 rest，所以我今天还持有着股票；
要么我昨天本没有持有，但今天我选择 buy，所以今天我就持有股票了。
```

https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock/solution/yi-ge-fang-fa-tuan-mie-6-dao-gu-piao-wen-ti-by-l-3/

股票买卖

```
base case：
dp[-1][k][0] = dp[i][0][0] = 0
dp[-1][k][1] = dp[i][0][1] = -infinity

状态转移方程：
dp[i][k][0] = max(dp[i-1][k][0], dp[i-1][k][1] + prices[i])
dp[i][k][1] = max(dp[i-1][k][1], dp[i-1][k-1][0] - prices[i])
```

https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock/solution/yi-ge-fang-fa-tuan-mie-6-dao-gu-piao-wen-ti-by-l-3/

快排 、 归并、 堆排序

sort(...)