数据结构与算法

面试篇

1. 树和二叉树

2. 图

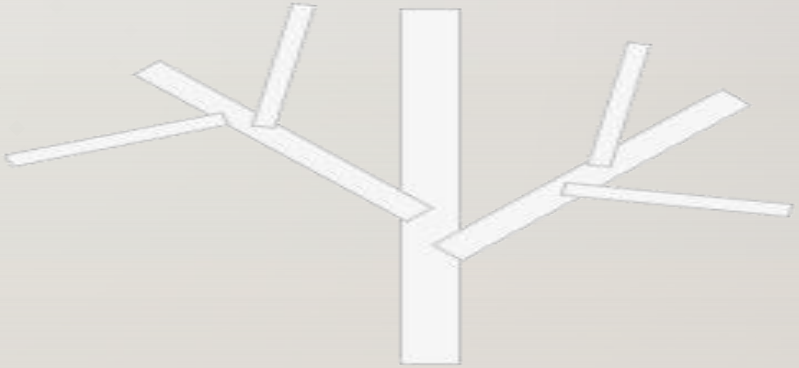
3. 基础算法

4. 答题技巧

5. leetCode分类及刷题技巧

树

- 用来模拟具有树状结构性质的数据集合。
- 树里的每一个节点有一个值和一个包含所有子节点的列表。从图的观点来看，树也可视为一个拥有N 个节点和N-I 条边的一个有向无环图。

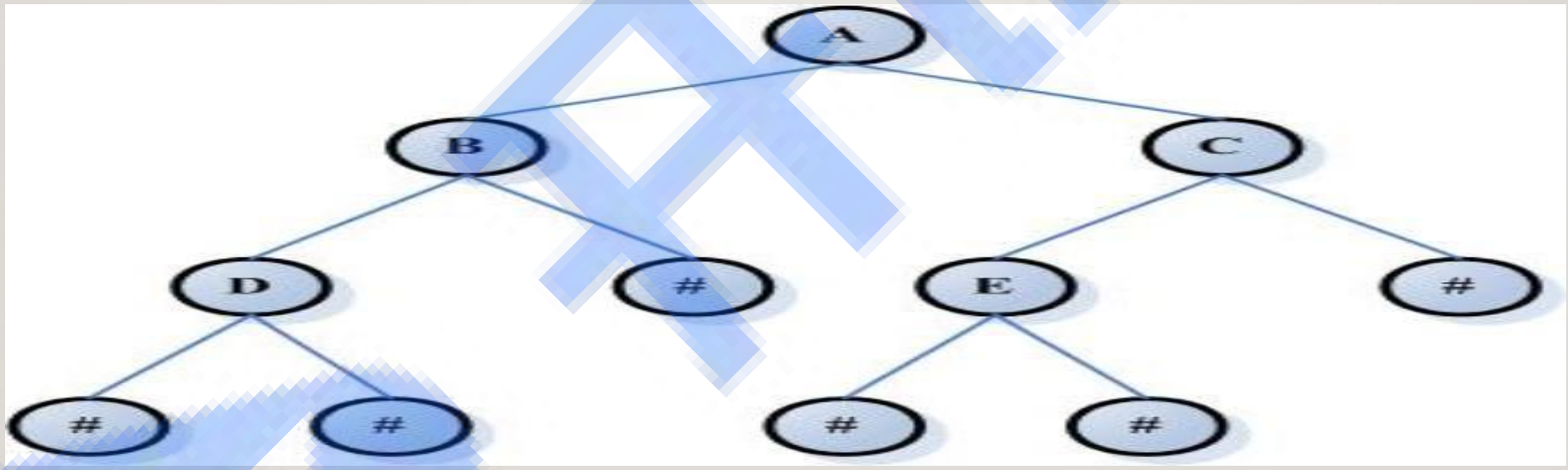


树

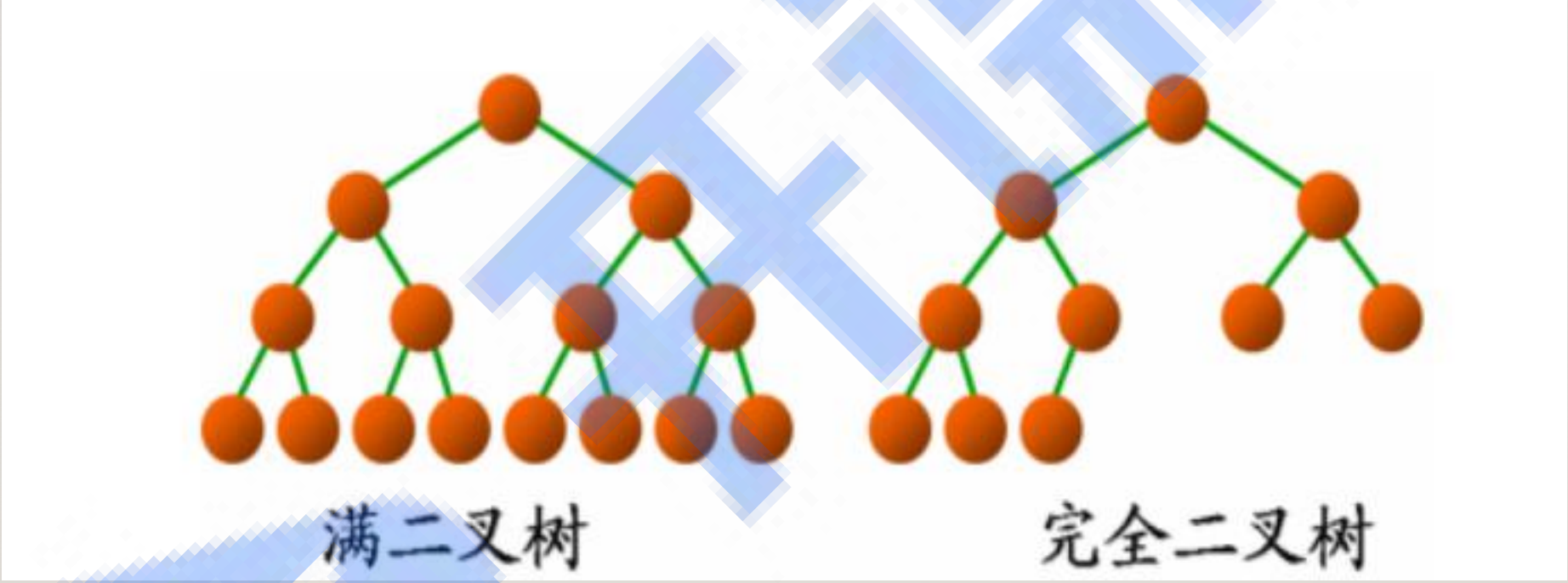
- 节点：树中的每个元素称为节点
  父子关系：相邻两节点的连线，称为父子关系
  根节点：没有父节点的节点
  叶子节点：没有子节点的节点
  父节点：指向子节点的节点
  子节点：被父节点指向的节点
  兄弟节点：具有相同父节点的多个节点称为兄弟节点关系
  节点的高度：节点到叶子节点的最长路径所包含的边数
  节点的深度：根节点到节点的路径所包含的边数
  节点的层数：节点的深度+1（根节点的层数是1）
  树的高度：等于根节点的高度

二叉树

- 二叉树是一种更为典型的树状结构。如它名字所描述的那样，二叉树是每个节点最多有两个子树的树结构，通常子树被称作“左子树”和“右子树”。

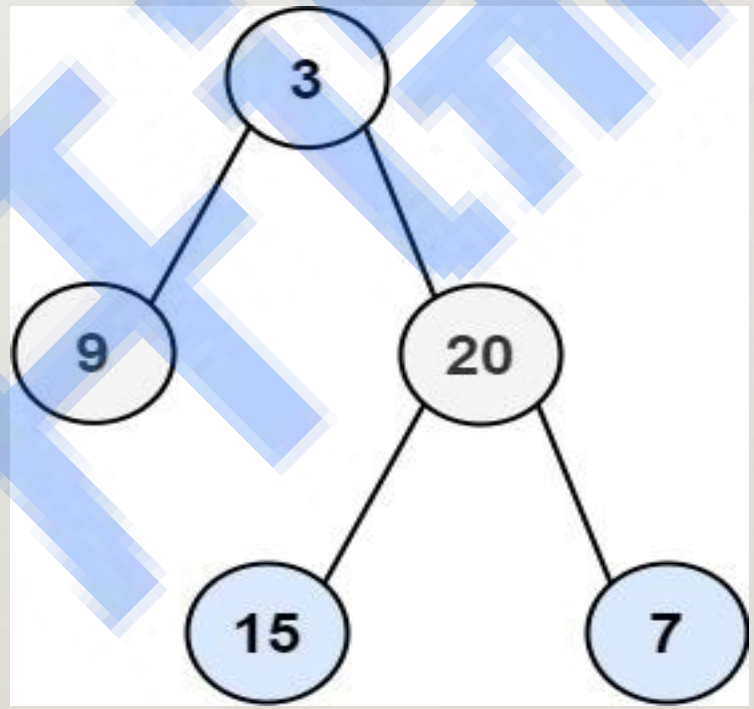


二叉树



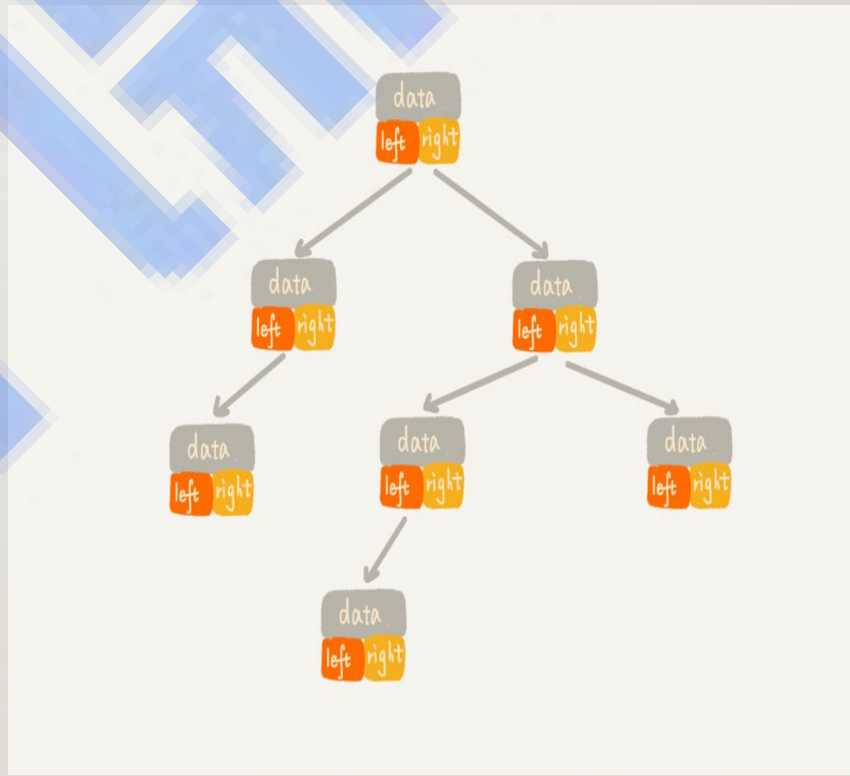
二叉树

[3,9,20,null,null,15,7]



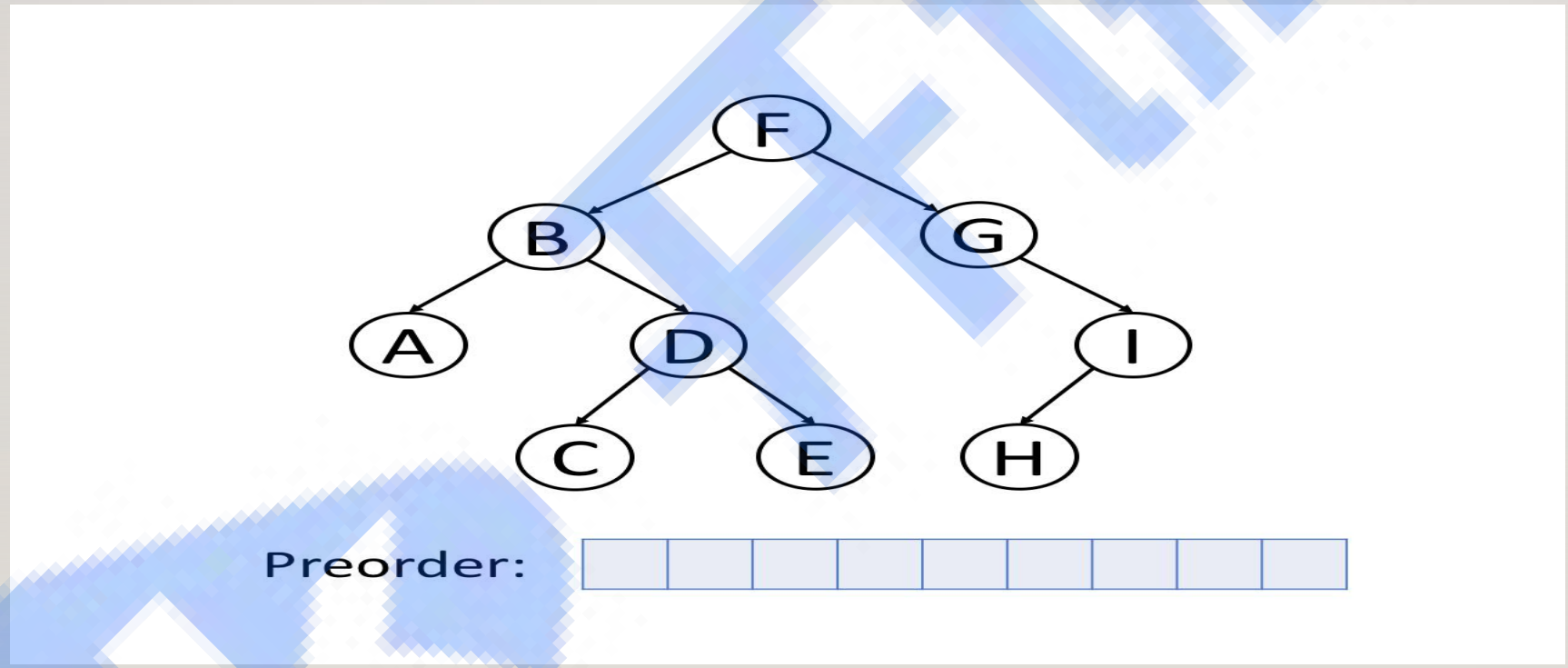
二叉树

```
public class TreeNode {
      int val;
     TreeNode left;
      TreeNode right;
      TreeNode() {}
       TreeNode(int val) { this.val = val; }
       TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
       this.right = right;
  }
}
```



二叉树

- **前序遍历:**前序遍历首先访问根节点，然后遍历左子树，最后遍历右子树。



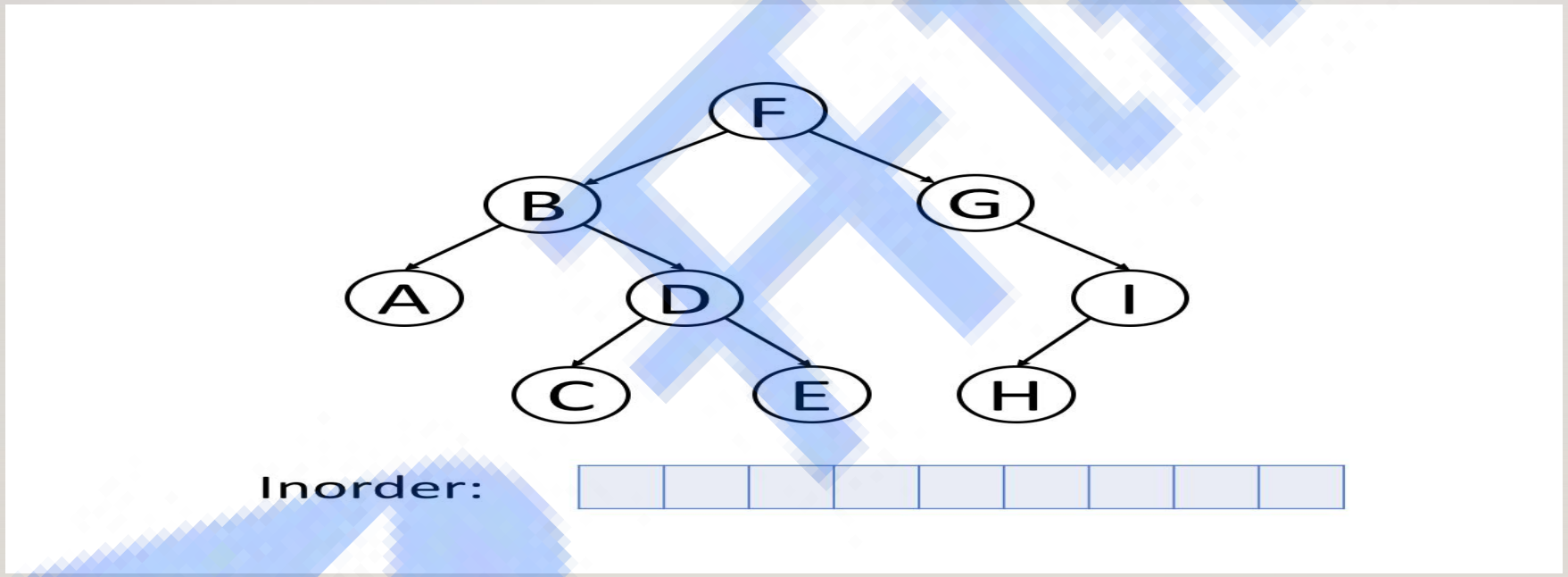
二叉树

- **前序遍历:**前序遍历首先访问根节点，然后遍历左子树，最后遍历右子树。
- 前序遍历的递推公式：preOrder(r) = print r->preOrder(r->left)->preOrder(r->right)

```
void preOrder(Node* root)
 {
        if (root == null)
            return;
        print root // 此处为伪代码，表示打印root节点
        preOrder(root->left);
        preOrder(root->right);
}
```

二叉树

- **中序遍历:**中序遍历是先遍历左子树，然后访问根节点，然后遍历右子树。



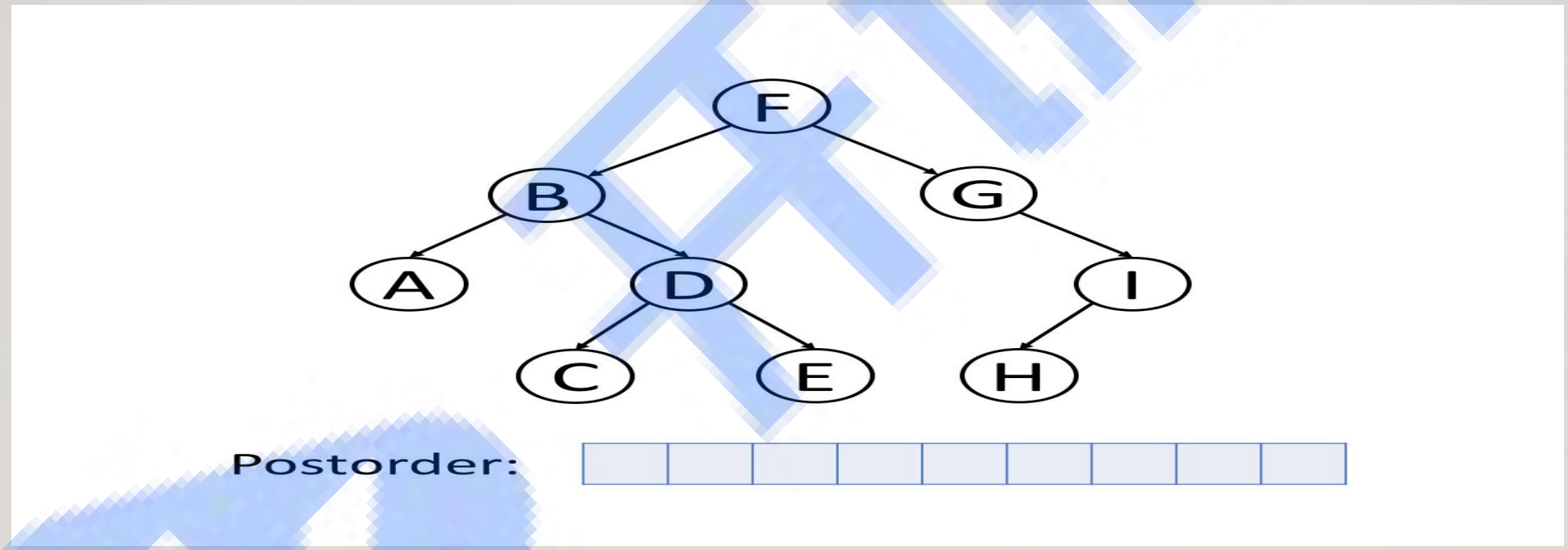
二叉树

- **中序遍历:**中序遍历是先遍历左子树，然后访问根节点，然后遍历右子树。
- 中序遍历的递推公式：inOrder(r) = inOrder(r->left)->print r->inOrder(r->right)

```
void inOrder(Node* root)
{
        if (root == null)
                return;
        inOrder(root->left);
        print root // 此处为伪代码，表示打印root节点
        inOrder(root->right);
}
```

二叉树

- **后序遍历:**后序遍历是先遍历左子树，然后遍历右子树，最后访问树的根节点。



二叉树

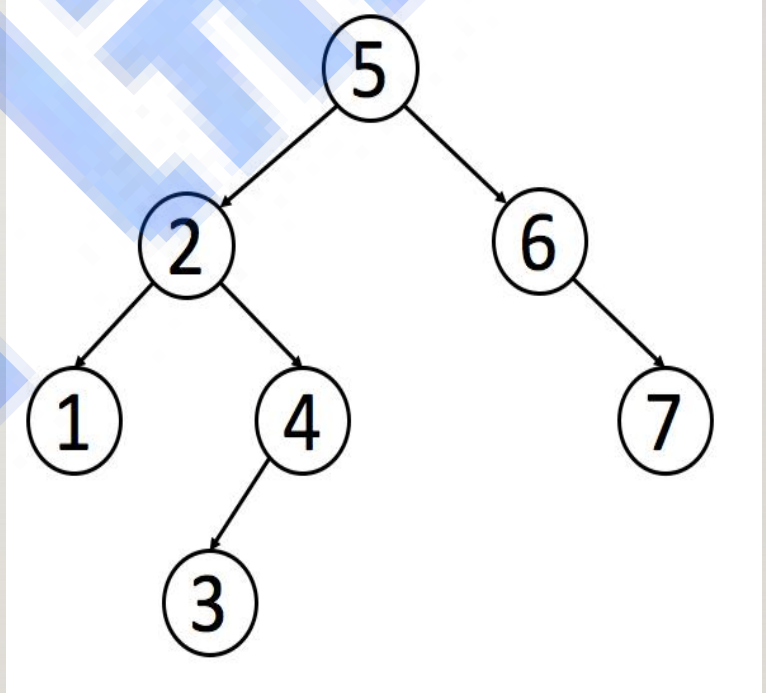
- **后序遍历:**后序遍历是先遍历左子树，然后遍历右子树，最后访问树的根节点。
- 后序遍历的递推公式：postOrder(r) = postOrder(r->left)->postOrder(r->right)->print r

```
void postOrder(Node* root) {
        if (root == null)
            return;
        postOrder(root->left);
        postOrder(root->right);
        print root // 此处为伪代码，表示打印root节点
}
```

二叉搜索树

二叉搜索树（BST）是二叉树的一种特殊表示形式，它满足如下特性：

- 每个节点中的值必须**大于**存储在其左侧子树中的任何值。
- 每个节点中的值必须**小于**存储在其右子树中的任何值。



二叉搜索树

- 给你一个二叉树的根节点 root， 判断其是否是一个有效的二叉搜索树。

示例 1:

- 输入: root = [2,1,3]
- 输出: true
- 示例 2:
- 输入: root = [5,1,4,null,null,3,6]
- 输出: false
- 解释: 根节点的值是 5， 但是右子节点的值是 4 。

二叉搜索树

二叉搜索树主要支持三个操作：搜索、插入和删除。

搜索

根据BST的特性，对于每个节点：

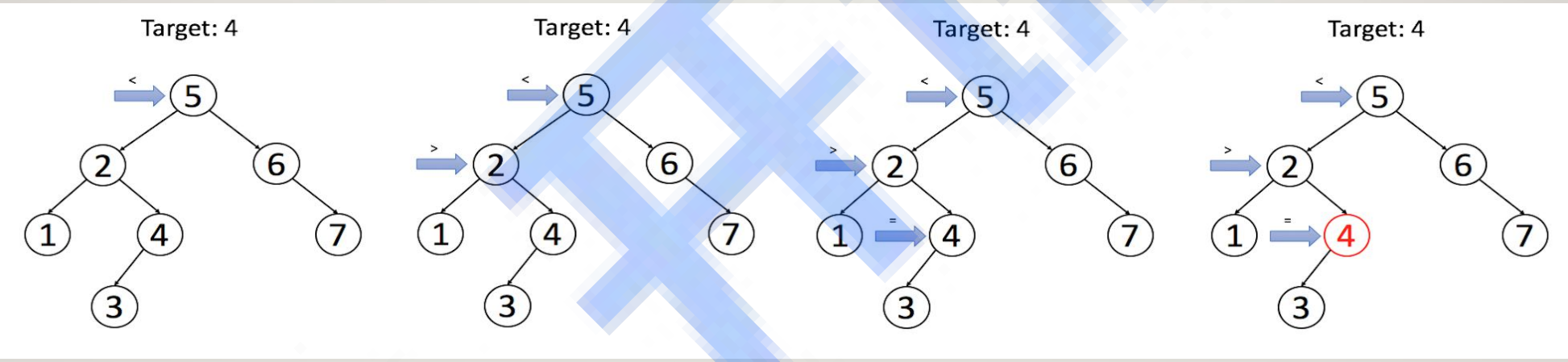
如果目标值等于节点的值，则返回节点；

如果目标值小于节点的值，则继续在左子树中搜索；

如果目标值大于节点的值，则继续在右子树中搜索。

二叉搜索树

搜索



二叉搜索树

插入

与搜索操作类似，对于每个节点，我们将：

1 根据节点值与目标节点值的关系，搜索左子树或右子树；

2 重复步骤 1 直到到达外部节点；

3 根据节点的值与目标节点的值的关系，
将新节点添加为其左侧或右侧的子节点。

Input Array: [5, 2, 6, 1, 7, 4, 3]

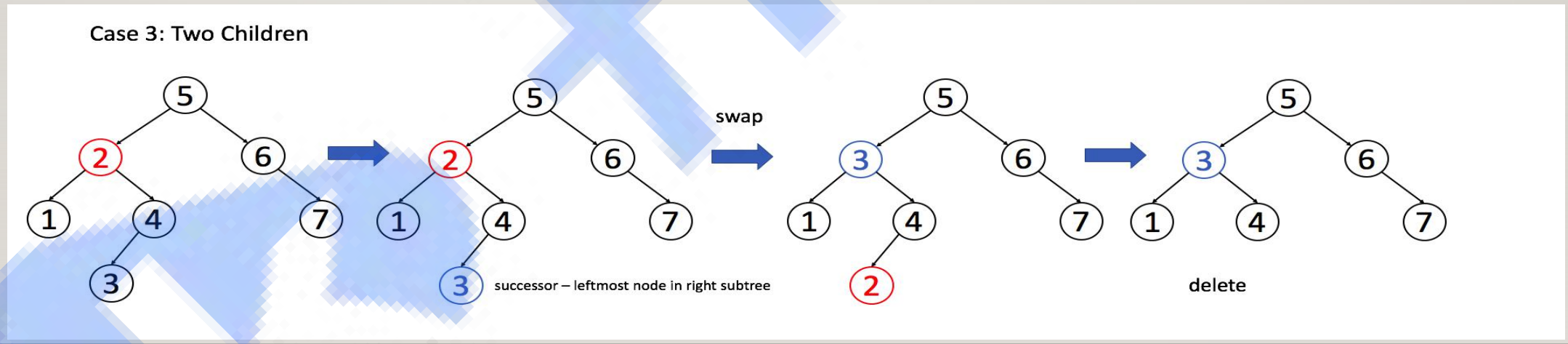
二叉搜索树

删除

1. 如果目标节点没有子节点，我们可以直接移除该目标节点。

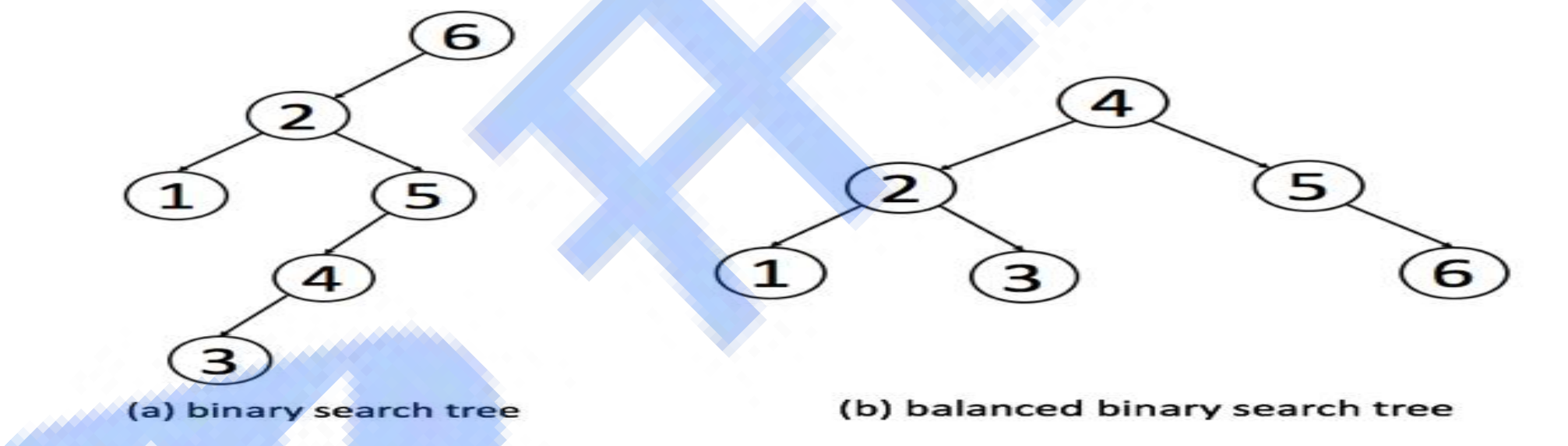
2. 如果目标节只有一个子节点，我们可以用其子节点作为替换。

3. 如果目标节点有两个子节点，我们需要用其中序后继节点或者前驱节点来替换，再删除该目标节点。



平衡二叉树

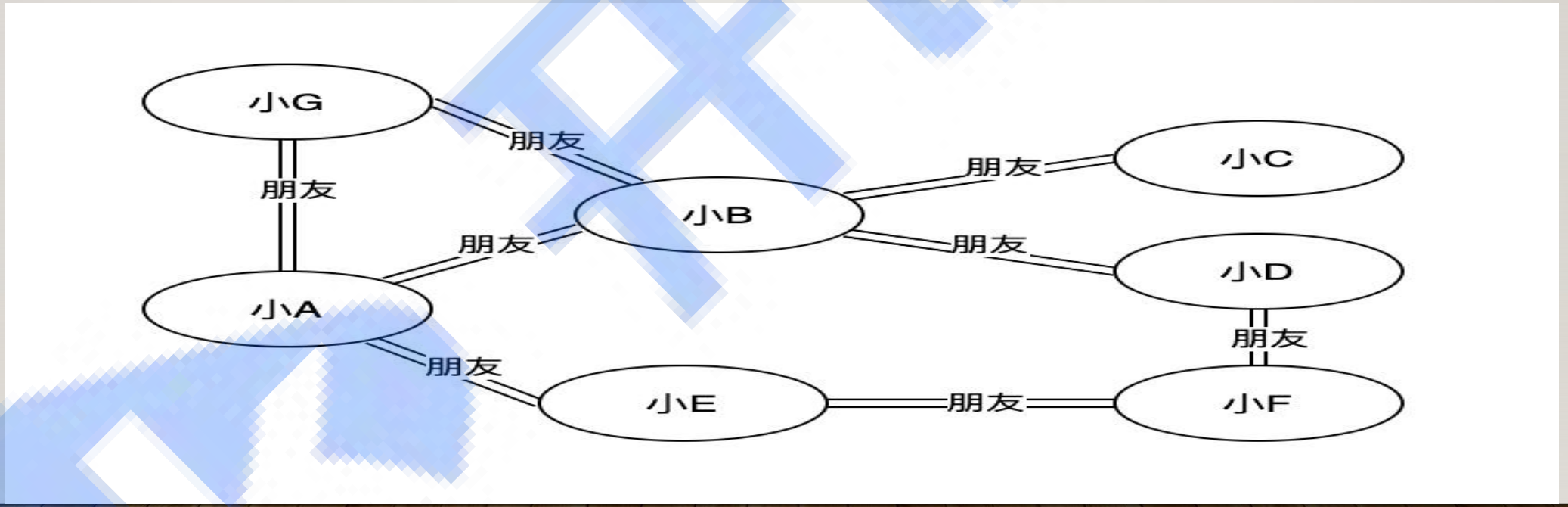
一个高度平衡的二叉搜索树（平衡二叉搜索树）是在插入和删除任何节点之后，可以自动保持其高度最小。。



高度平衡的二叉搜索树对提高性能起着重要作用

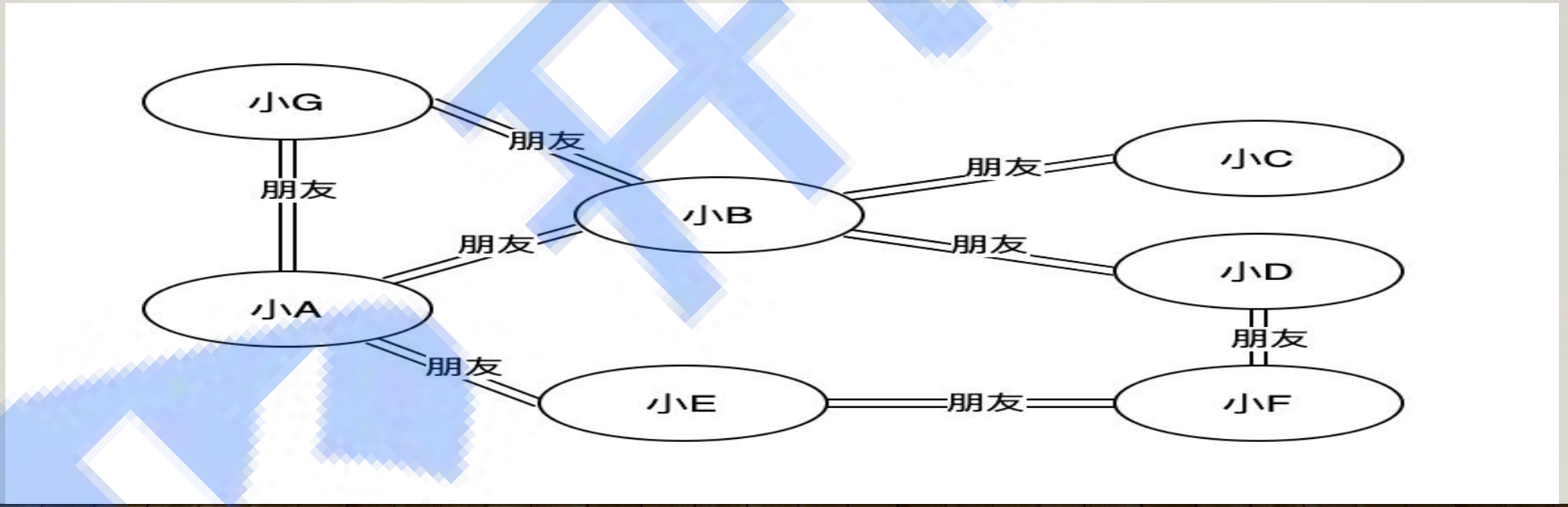
图

「图」大概是最接近生活的一种数据结构了，生活中各种关系图便是「图」最真实的写照。比如，我们每个人的朋友交际圈就是一个巨大的「图」。



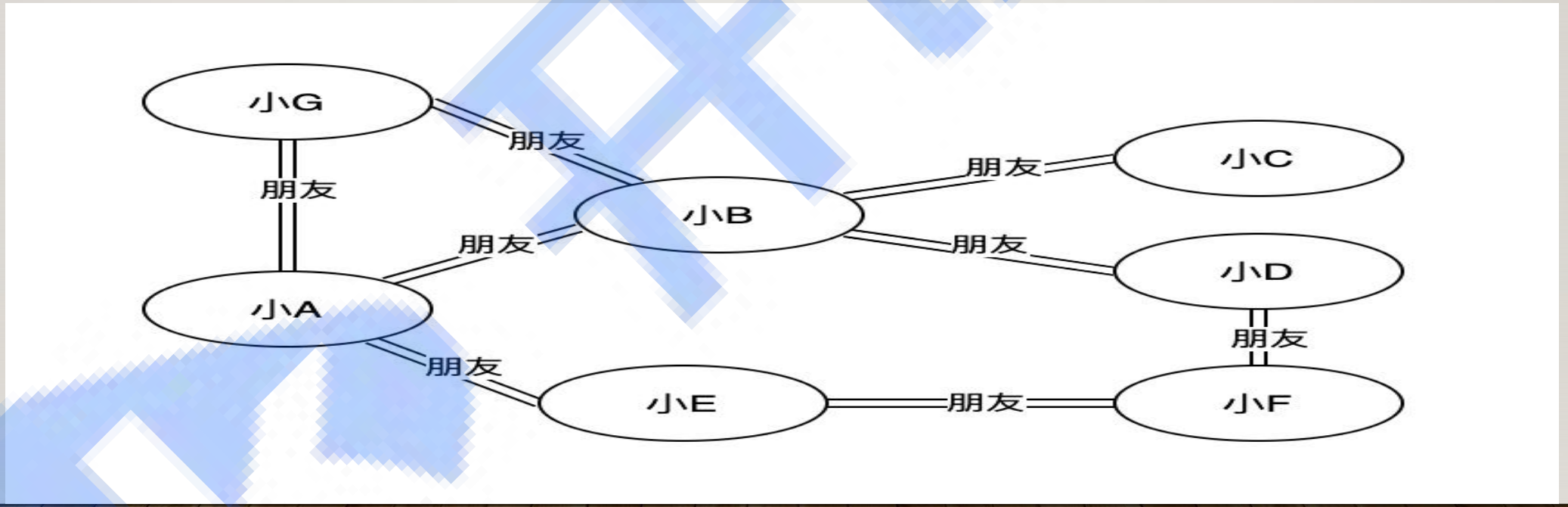
图

「图」大概是最接近生活的一种数据结构了，生活中各种关系图便是「图」最真实的写照。比如，我们每个人的朋友交际圈就是一个巨大的「图」。



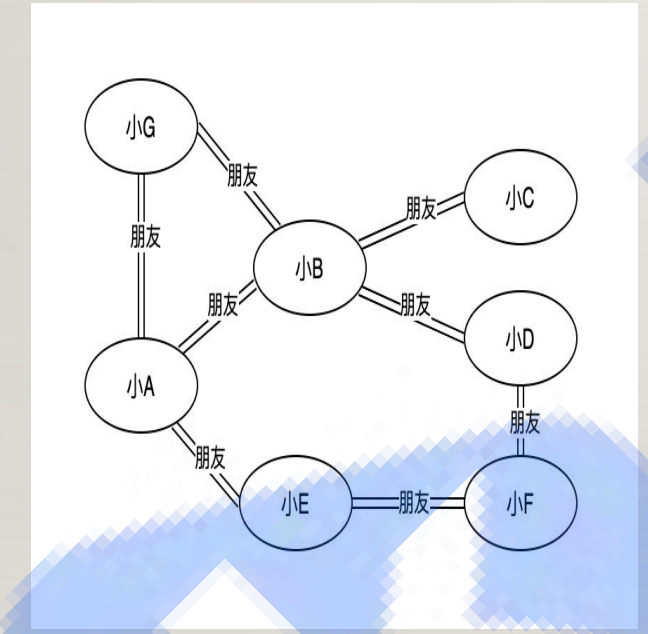
图

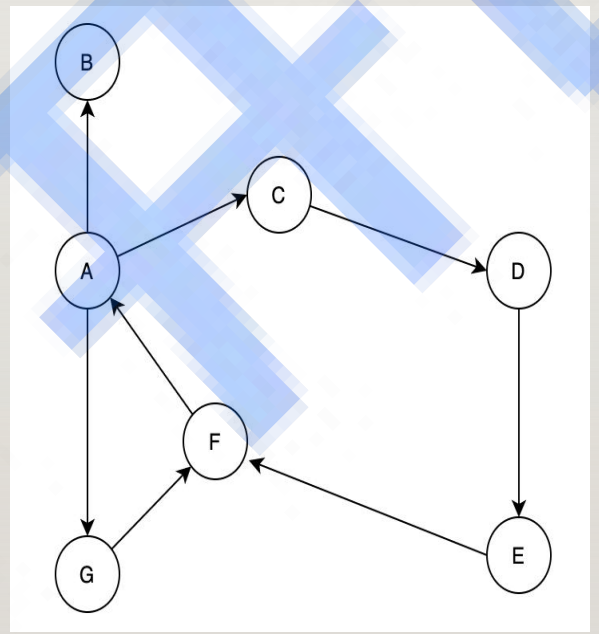
「图」大概是最接近生活的一种数据结构了，生活中各种关系图便是「图」最真实的写照。比如，我们每个人的朋友交际圈就是一个巨大的「图」。

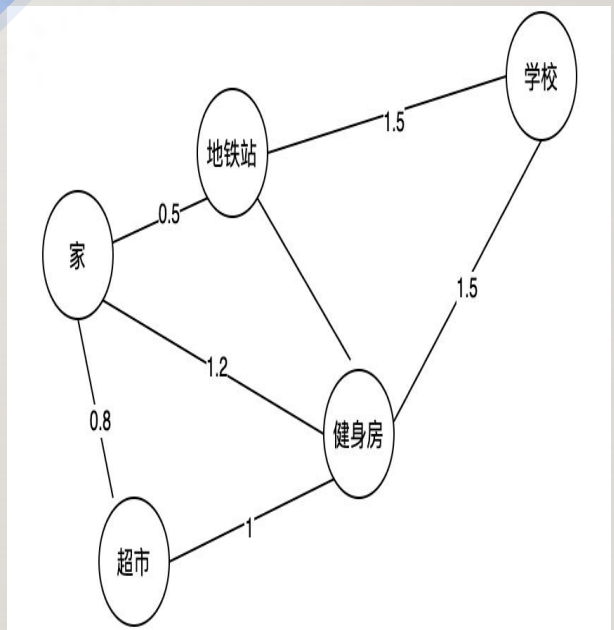


图

无向图，有向图，**加权图**

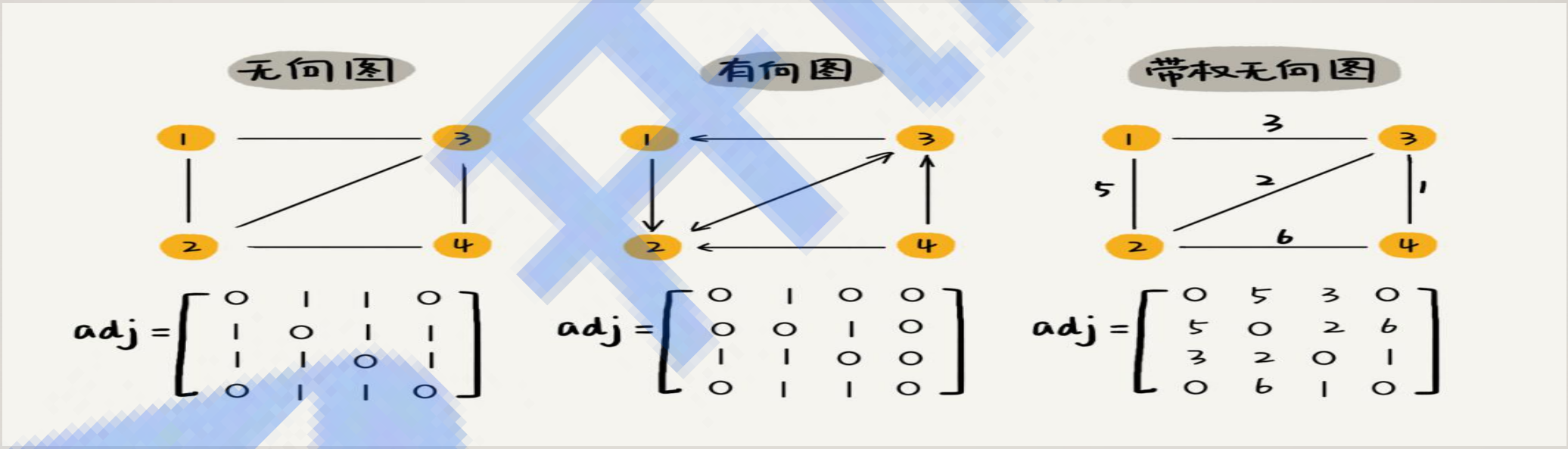

小G
朋友
朋友
小B
朋友
小C
朋友
朋友
小D
小A
朋友
朋友
小E
朋友
小F


B
C
A
D
F
E
G


学校
地铁站
1.5
0.5
家
1.5
1.2
0.8
健身房
超市
1

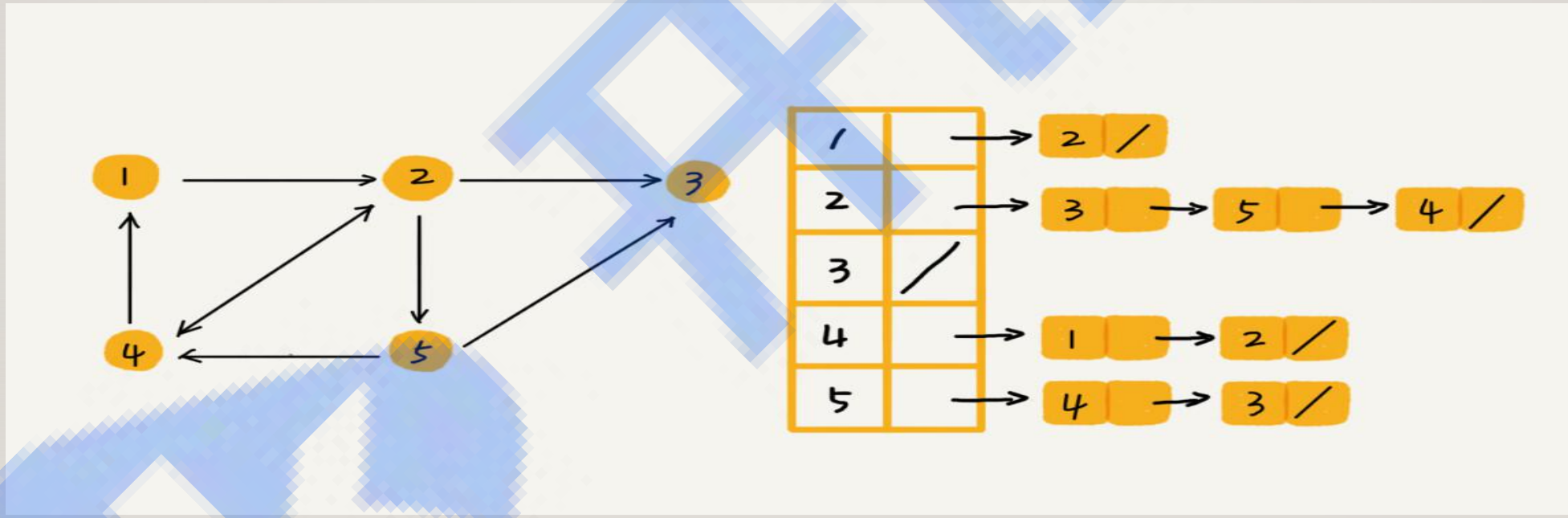
图

数组存储



图

邻接表



基础算法

- 枚举算法
- 递归算法
- 贪心算法
- 分治算法
- 动态规划
- 位运算

枚举算法

- 将问题的所有可能的答案一一列举，然后根据条件判断此答案是否合适，保留合适的，舍弃不合适的。

递归算法

- 递归第一个步骤：明确函数要做什么
- 对于递归，一个最重要的事情就是要明确这个函数的功能。这个函数要完成一样什么样的事情，是完全由程序员来定义的，当写一个递归函数的时候，先不要管函数里面的代码是什么，而要先明确这个函数是实现什么功能的。

递归算法

- 第一个步骤：明确函数要做什么
- 第二个步骤：明确递归的结束（退出递归）条件
- 第三个步骤：找到函数的等价关系式

```
// 计算n的阶乘（假设n不为0）
int f(int n) {
    if (n <= 2) {
        return n;
    }
    // 把n打出来看一下，你就能明白递归的原理了
    System.out.println(n);
    // 加入f(n)的等价操作逻辑
    return n * f(n - 1);
}
```

贪心算法

- 贪心算法（又称贪婪算法）是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的仅是在某种意义上的局部最优解。贪心算法不是对所有问题都能得到整体最优解，但对范围相当广泛的许多问题他能产生整体最优解或者是整体最优解的近似解。

- 第一步：明确到底什么是最优解？明确下来之后用小本本记下来！
- 第二步：明确什么是子问题的最优解？再用小本本记下来！
- 第三步：分别求出子问题的最优解再堆叠出全局最优解？这步不用记！

贪心算法

- **0-1背包问题**
- 有一个背包，最多能承载150斤的重量，现在有7个物品，重量分别为[35, 30, 60, 50, 40, 10, 25]，它们的价值分别为[10, 40, 30, 50, 35, 40, 30]，如果是你的话，应该如何选择才能使得我们的背包背走最多价值的物品？

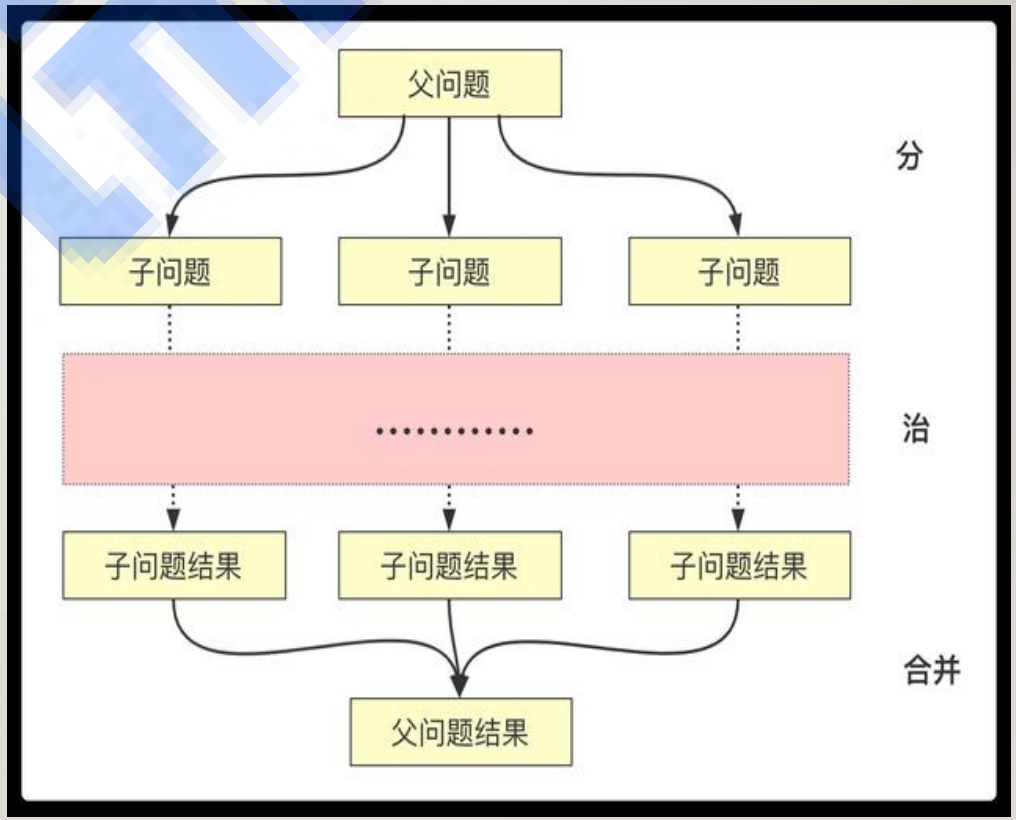
分治算法

- 分治算法，即分而治之，就是把原问题分解为几个类似原问题的子问题，解决完子问题，再把子问题的解合并在一起，就可以得到原问题的解。分治算法一般包括三个过程：

1. 分解：将原问题分解成若干个子问题。

2. 解决：递归求解各自子问题，如果子问题足够小，直接求解。

3.合并：合并这些子问题的解，即可得到原问题的结果。



位运算

- 461. Hamming Distance
  即求两个正整数的二进制对应位数不同的个数
- 输入A: 1010
- 输入B: 1100
- 异或运算结果: 0110
- 异或非(同或)运算结果: 1001

精通一个领域

- 切碎知识点
- 刻意练习
- 获得反馈

切碎知识点

数组 1075　字符串 537　动态规划 397　哈希表 395　数学 375　深度优先搜索 278　排序 238

广度优先搜索 222　树 222　贪心 207　二叉树 195　二分查找 176　双指针 171　数据库 168　矩阵 167

位运算 145　栈 137　设计 127　堆（优先队列） 111　回溯 103　链表 90　图 90　模拟 75　滑动窗口 72

前缀和 70　并查集 64　计数 61　递归 60　二叉搜索树 54　分治 49　字典树 49　单调栈 42　有序集合 37

队列 37　记忆化搜索 32　几何 31　状态压缩 29　线段树 24　拓扑排序 24　哈希函数 22　博弈 21　数据流 20

字符串匹配 18　交互 18　枚举 18　树状数组 17　滚动哈希 14　随机化 14　最短路 13　组合数学 13

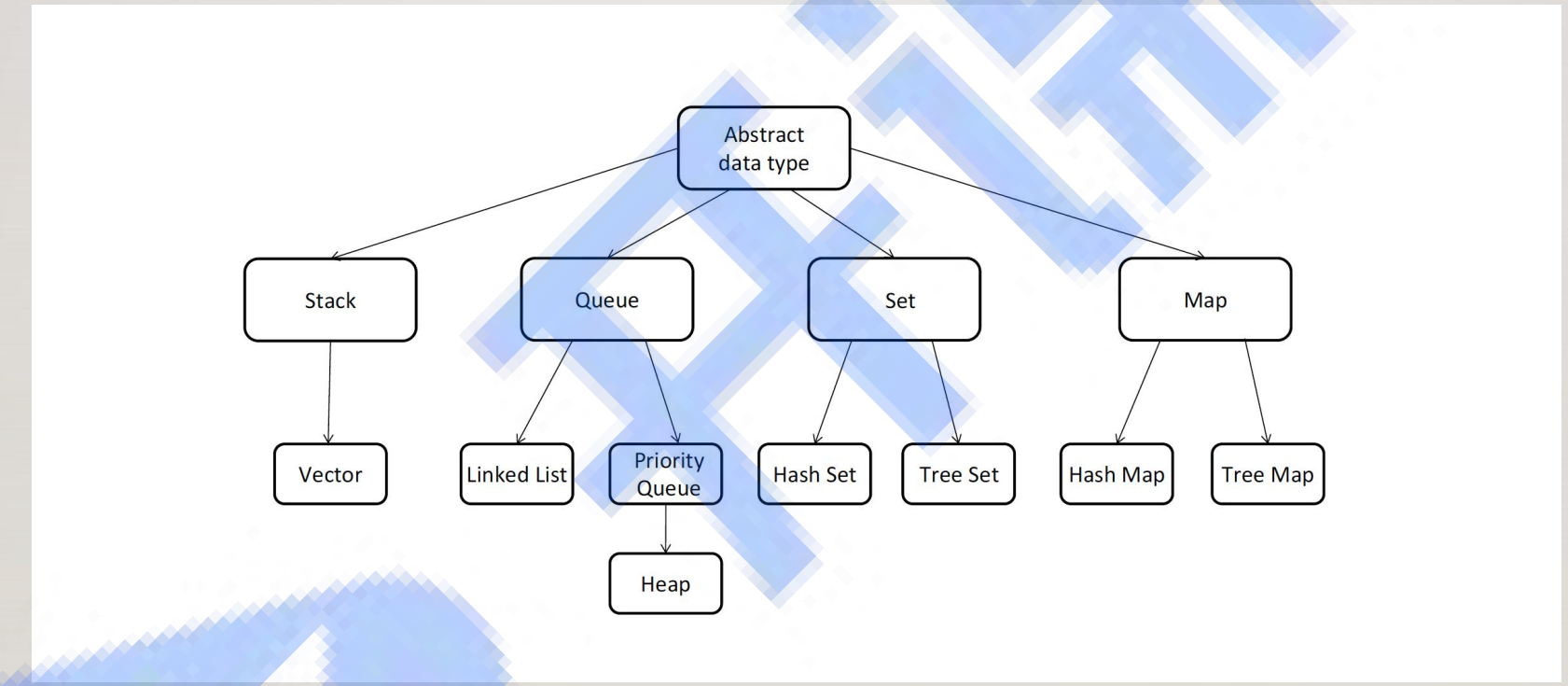
双向链表 11　归并排序 10　单调队列 10　迭代器 10　脑筋急转弯 10　概率与统计 9　快速选择 9　数论 9

多线程 9　桶排序 8　计数排序 6　后缀数组 5　最小生成树 5　扫描线 4　Shell 4　水塘抽样 4　强连通分量 2

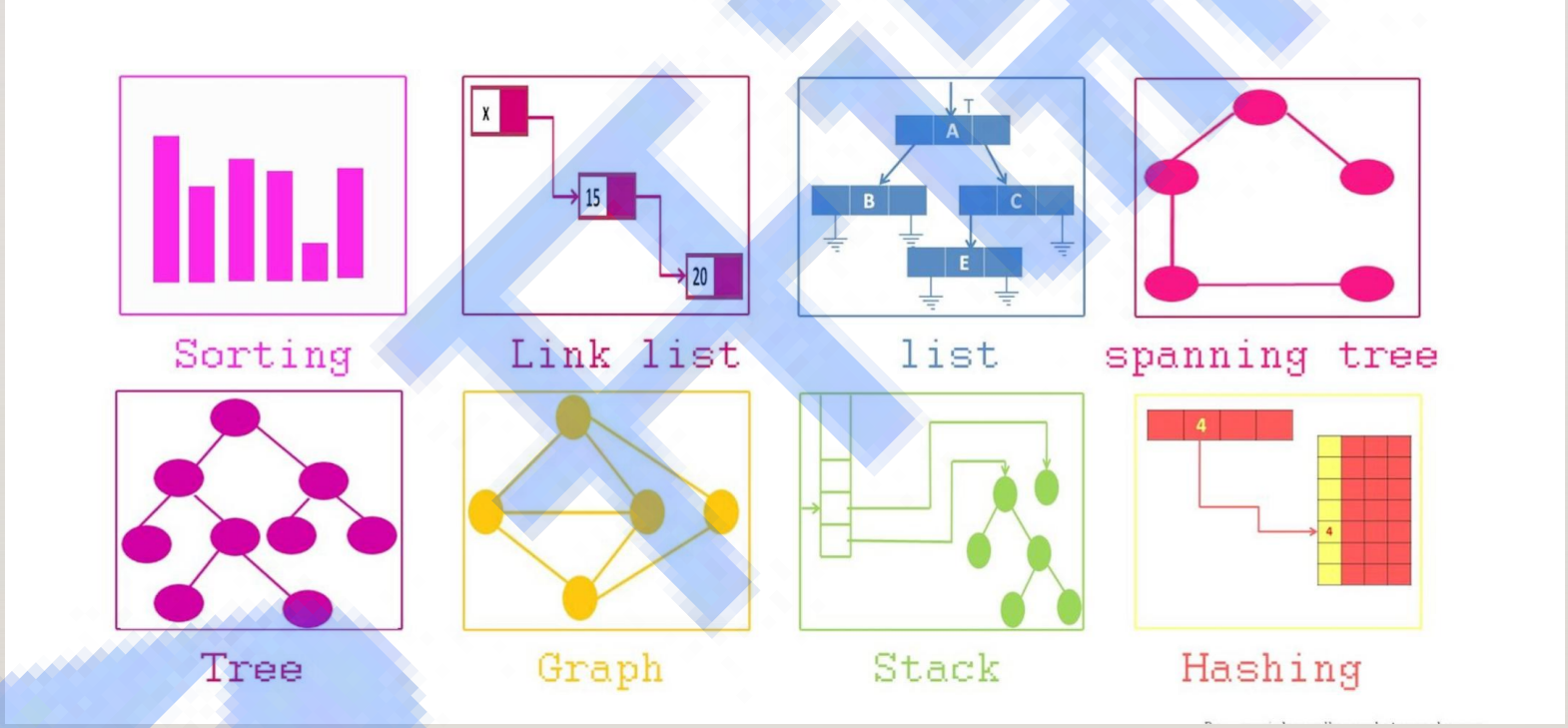
欧拉回路 2　拒绝采样 2　基数排序 2　双连通分量 1

收起 ^

切碎知识点


Abstract data type
Stack
Queue
Set
Map
Vector
Linked List
Priority Queue
Hash Set
Tree Set
Hash Map
Tree Map
Heap

切碎知识点


Sorting
X
15
20
Link list
T
A
B
C
E
list
spanning tree
Tree
Graph
Stack
4
4
Hashing

刻意练习

- 分类刷题
- 练习缺陷，不舒服，弱点地方
- 错题本

获得反馈

- 即时反馈
- 主动性反馈
  - 高手代码（Github, LeetCode），模仿去写
- 被动式反馈
  - Code review
  - 面试，获得面试官反馈

答题四件套

- 询问题目细节、边界条件、可能的极端错误情况
- 所有可能的解体都和面试官沟通一遍
  - 时间复杂度和空间复杂度
- 写代码
- 测试用例

数组

1. 实现一个支持动态扩容的数组

2. 实现一个大小固定的有序数组，支持动态增删改操作

3.实现两个有序数组合并为一个有序数组

Three Sum（求三数之和): https://leetcode-cn.com/problems/3sum/

Majority Element（求众数）: https://leetcode-cn.com/problems/majority-element/

Missing Positive（求缺失的第一个正数）: https://leetcode-cn.com/problems/first-missing-positive/

数组

LeetCode：

Three Sum（求三数之和)：https://leetcode-cn.com/problems/3sum/

Majority Element（求众数）：https://leetcode-cn.com/problems/majority-element/

Missing Positive（求缺失的第一个正数）：https://leetcode-cn.com/problems/first-missing-positive/

链表

1. 实现单链表、循环链表、双向链表，支持增删操作

2.实现单链表反转

3.实现两个有序的链表合并为一个有序链表

4.实现求链表的中间结点

链表

LeetCode:

Linked List Cycle I（环形链表）：：https://leetcode-cn.com/problems/linked-list-cycle/

Merge k Sorted Lists（合并 k 个排序链表）：https://leetcode-cn.com/problems/merge-k-sorted-lists/

栈

1.用数组实现一个顺序栈

2. 用链表实现一个链式栈

3. 编程模拟实现一个浏览器的前进、后退功能

栈

LeetCode:

Valid Parentheses（有效的括号）：https://leetcode-cn.com/problems/valid-parentheses/

Longest Valid Parentheses（最长有效的括号）：https://leetcode-cn.com/problems/longest-valid-parentheses/

Evaluate Reverse Polish Notatio（逆波兰表达式求值）：https://leetcode-cn.com/problems/evaluate-reverse-polish-notation/

队列

1. 用数组实现一个顺序队列

2. 用链表实现一个链式队列

3 .实现一个循环队列

队列

LeetCode

Design Circular Deque（设计一个双端队列）：https://leetcode-cn.com/problems/design-circular-deque/

Sliding Window Maximum（滑动窗口最大值）：https://leetcode-cn.com/problems/sliding-window-maximum/

递归

1. 编程实现斐波那契数列求值 f(n)=f(n-1)+f(n-2)
2. 编程实现求阶乘 n!
3. 编程实现一组数据集合的全排列

递归

LeetCode:

Climbing Stairs（爬楼梯）：https://leetcode-cn.com/problems/climbing-stairs/

排序和二分查找

1. 实现归并排序、快速排序、插入排序、冒泡排序、选择排序
2. 编程实现 O(n) 时间复杂度内找到一组数据的第 K 大元素
3. 实现一个有序数组的二分查找算法
4. 实现模糊二分查找算法（比如大于等于给定值的第一个元素）

排序和二分查找

LeetCode:

Sqrt(x) （x 的平方根）: https://leetcode-cn.com/problems/sqrtx/

散列表 HASH

1. 实现一个基于链表法解决冲突问题的散列表
2. 实现一个 LRU 缓存淘汰算法

字符串

LeetCode:

Reverse String （反转字符串）：https://leetcode-cn.com/problems/reverse-string/

Reverse Words in a String（翻转字符串里的单词）：https://leetcode-cn.com/problems/reverse-words-in-a-string/

String to Integer (atoi)（字符串转换整数 (atoi)）：https://leetcode-cn.com/problems/string-to-integer-atoi/

二叉树

1. 实现一个二叉查找树，并且支持插入、删除、查找操作
2. 实现查找二叉查找树中某个节点的后继、前驱节点
3. 实现二叉树前、中、后序以及按层遍历

二叉树

LeetCode:

Invert Binary Tree（翻转二叉树）：https://leetcode-cn.com/problems/invert-binary-tree/

Maximum Depth of Binary Tree（二叉树的最大深度）：https://leetcode-cn.com/problems/maximum-depth-of-binary-tree/

Validate Binary Search Tree（验证二叉查找树）：https://leetcode-cn.com/problems/validate-binary-search-tree/

图

1. 实现有向图、无向图、有权图、无权图的邻接矩阵和邻接表表示方法

2. 实现图的深度优先搜索、广度优先搜索

Number of Islands（岛屿的个数）：https://leetcode-cn.com/problems/number-of-islands/description/

Valid Sudoku（有效的数独）：https://leetcode-cn.com/problems/valid-sudoku/

算法

LeetCode:

Regular Expression Matching（正则表达式匹配）：https://leetcode-cn.com/problems/regular-expression-matching/

Minimum Path Sum（最小路径和）：https://leetcode-cn.com/problems/minimum-path-sum/

Coin Change （零钱兑换）：https://leetcode-cn.com/problems/coin-change/

Best Time to Buy and Sell Stock（买卖股票的最佳时机）：https://leetcode-cn.com/problems/best-time-to-buy-and-sell-stock/

Maximum Product Subarray（乘积最大子序列）：https://leetcode-cn.com/problems/maximum-product-subarray/

Triangle（三角形最小路径和）：https://leetcode-cn.com/problems/triangle/

多练，多想，多总结