

## Data Structure

### ① Array / String

Array: 优点: 构建简单  
[ 在  $O(1)$  的时间根据 index 查询某个元素 ]  
缺点: [ 构建时必须分配一段连续的空间  
    查询某个元素需要遍历整个数组,  $O(n)$   
    添加及删除某个元素  $\rightarrow O(n)$  ]

### 242. 字母单位词



### ② Linked list 链表

单链表  $\boxed{0} \rightarrow \boxed{1} \rightarrow \boxed{2} \rightarrow \text{null}$

双链表  $\text{null} \leftarrow \boxed{0} \leftarrow \boxed{1} \leftarrow \boxed{2} \rightarrow \text{null}$

优点: 灵活分配内存空间  
能在  $O(1)$  时间内删除 or 添加元素 (已知前 or 后)

缺点: 查询元素需要  $O(n)$  时间

解题技巧: ①利用快慢指针 (有时候需要用到3个指针)  
②构建一个虚假的链表头  $\rightarrow$  不然每次需判断链表头是否为空

例: 两个排序 linked list, 进行整合排序

将 linked list 的奇偶数按原定顺序分离, 生成前半部分为奇数  
后半部分为偶数的链表.

## 25. K个一组翻转链表

① 链表长度大于 K

Example: k=3

Null → [1] → [2] → [3] → [4] → [5] → Null

Null ← [1] ← [2] ← [3]    [4] → [5] → Null

## ③ Stack 栈

特点：后进先出 (LIFO)

可以用单链表来实现

处理完上一次操作后，能在 O(1) 时间内找到更前一次的操作

26. 有效的括号

239. 每日温度

## ④ Queue 队列

特点：先进先出 (FIFO)

常用场景：广度优先搜索

## ⑤ Deque 双端队列

239. 滑动窗口最大值

## ⑥ Tree 树 (recursion)

普通二叉树

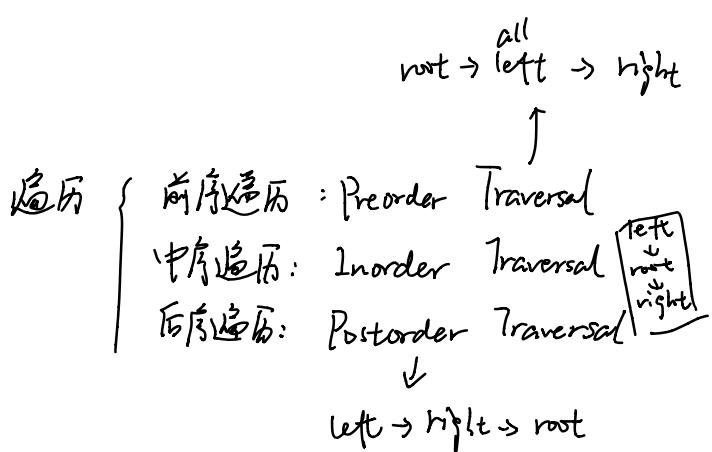
平衡二叉树

二叉搜索树

完全二叉树

四叉树

多叉树



230. 二叉搜索中第 k 小的元素

## Data Structure 2.0

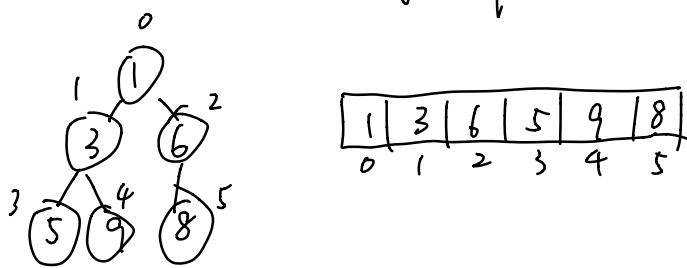
Priority Queue 优先队列  
 Graph 图  
 Trie 前缀树  
 Segment Tree 线段树  
 Fenwick Tree / Binary Indexed Tree 权状数组

### ① Priority Queue 优先队列

区别：保证每次取出的元素是队列中优先级最高的，  
优先级如何定义。

常用场景：从杂乱无章的数据中按一定的顺序（或优先级）筛选数据  
例：找出前 k 个大的数

本质：二叉堆 (Binary heap)



array[0]：最高的优先级

let i be given :

父节点： $\lceil \frac{i-1}{2} \rceil$

左叶子节点： $2 \times i + 1$

右叶子节点： $2 \times i + 2$

每个元素的优先级都高于它两侧子节点

其基本操作为以下两个

向上筛选 (sift up / bubble up) : 加新数据 → 放在底部 → 与父节点一一比较

向下筛选 (sift down / bubble down)

另一个重要的时间复杂度：优先队列的初始化

$$\begin{aligned}
 T(n) &= \sum_{h=0}^{\log(n)} \left\lceil \frac{n}{2^{h+1}} \right\rceil \times O(h) \\
 &= O(n \times \sum_{h=0}^{\log(n)} \frac{h}{2^h}) \\
 &= O(n \times \frac{\infty}{2^{\log(n)}}) = O(n)
 \end{aligned}$$

347. 前 k 个高频元素

哈希表  $\rightarrow$  优先队列  $\rightarrow$  前 k 个

## ② Graph 图

阶、度

树、森林、林

有向图，无向图，完全有向图，完全无向图

连通图，连通分量

图的存储和表达方式：邻接矩阵、邻接链表

算法：图的遍历：深度优先，广度优先

环的检测：有向图，无向图

拓扑排序

最短路径算法：Dijkstra, Bellman-Ford, Floyd Warshall

连通性相关算法：Kosaraju, Tarjan, 求解孤岛的数量，判断是否有树图的着色，旅行商问题等

必须掌握：图的存储和表达方式：邻接矩阵、邻接链表

图的遍历：BFS, DFS

二部图的检测 (Bipartite)、树的检测、环的检测：有向图，无向图

拓扑排序

联合-查找算法 (Union-Find)

最短路径：Dijkstra, Bellman-Ford

## 785. 判断二分图

BFS, DFS

### ③ Trie 前缀树

也称字典树：这种数据结构被广泛地运用在字典查找中

Example：给定一系列构成字典的字符串，要求在字典中找出所有以 'ABC' 开头的字符串

一、暴力： $O(M \times N)$

二、前缀树： $O(M)$

经典应用：搜索框输入搜索文字，会罗列以搜索词开头的相关搜索输入法。

Property:

- 每个节点至少包含两个重要属性

- children：数组或者集合，罗列每个分支当中包含的所有字符
- isEnd：布尔值，表示该节点是否为某字符串的结尾

1° 根节点是空的。

2° 除了根节点，其他所有节点可能都是单词的结尾，叶子节点一定都是单词的结尾。

基本操作：创建

1° 遍历一遍输入的字符串，对每个字符串的字符进行遍历

2° 从前缀树的根节点开始，将每个字符加入到节点的 children 字符集部

3° 如果字符串已经包含了这个字符，跳过

4° 如果当前字符是字符串的最后一个，把当前节点的 isEnd 标记为真

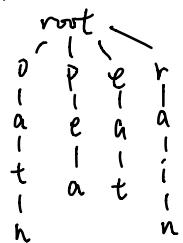
搜索

1° 从 Trie 的 root 出发，逐个匹配输入的前缀字符

2° 如果遇到了，继续往下一层搜索

3° 如果没遇到，立即返回

212. 单词搜索 I DFS



O	a	a	n
e	t	a	e
i	h	k	r
l	f	l	v

## ④ Segment Tree 线段树

从一个例题出发：

假设我们有一个数组  $\text{array}[0, \dots, n-1]$ ，里面有  $n$  个元素，现在我们要经常对这个数组做两件事：

1. 更新数组元素的数值

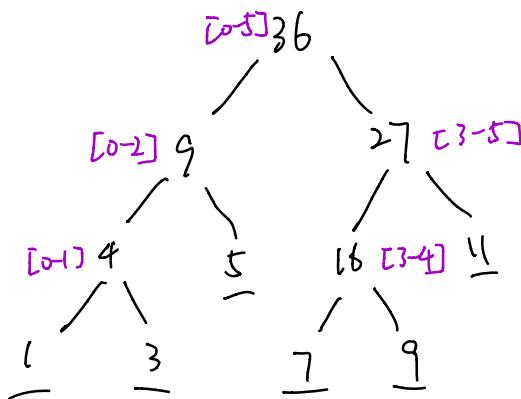
2. 求数组任意一段区间里元素的总和（或者平均值）

Solution 1: 遍历一遍数组 :  $O(n)$  (不适用于多 & 频繁更新的数组)

Solution 2: Segment Tree:  $O(\lg n)$  更新 & 求和

Example:  $[1, 3, 5, 7, 9, 11]$

⇒ Segment tree



315. 计算右侧小于当前元素的个数

## ⑤ Fenwick Tree / Binary Indexed Tree 树状数组

从一个例题出发：

假设我们有一个数组  $\text{array}[0, \dots, n-1]$ ，里面有 n 个元素，现在我们要经常对这个数组做两件事：

1. 更新数组元素的数值

2. 求数组前 k 个元素的总和（或者平均值）

方法一：segment tree  $O(\log n)$

方法二：Binary indexed tree

重要基本特征： 1° 利用数组来表示多叉树的结构，和优先队列有些类似

2° 优先队列是用数组来表示完全二叉树，而树状数组是多叉树

3° 树状数组的第一个元素是空节点

4° 如果节点  $\text{tree}[y]$  是  $\text{tree}[x]$  的父节点，那么需要满足  $y = x - (x \& 1 - x)$

Exercise: 308

## Sorting

基本排序算法: Bubble Sort 冒泡排序  
Insertion Sort 插入排序

常用的排序算法: Merge sort 归并排序  
Quick sort 快速排序  
Topological Sort 拓扑排序

其它排序算法: Heap Sort 堆排序  
Bucket Sort 桶排序

### Bubble Sort [2, 1, 7, 9, 5, 8]

1 2 7 9 5 8	extra space $O(1)$
1 2 7 9 5 8	running time $O(n^2)$
1 2 7 9 5 8	
1 2 7 9 5 8	
1 2 7 5 9 8	
1 2 7 5 9 8	
1 2 5 7 9 8	
1 2 5 7 8 9	

### Insertion Sort

2 1 7 5 9 8 extra space  $O(1)$   
 $1 \leftrightarrow 2$  7 9 5 8 running time  $O(n^2)$

1 2 7 9  $\longleftrightarrow$  5 8  
1 2 7 5 9 8  
1 2 5 7 9 8  
1 2 5 7 8 9

(141)

## Merge Sort

divide  $\rightarrow$  conquer

分到只剩 1 个

$$T(n) = 2 T(\frac{n}{2}) + O(n)$$

对于规模为  $n$  的问题，一共要进行  $\log(n)$  层的大小切分；  
每一层的合并复杂度都是  $O(n)$

running time:  $O(n \log n)$

extra space:  $O(n)$

## Quick Sort

原数组  $\rightarrow$  较大及较小两个子数组  $\rightarrow$  遍历排序两个子数组

[2 1 7 9 5 8]

7: 基准值

[2 1 5] [7 9 8]

2: 基准值 8: 基准值

[1] [2 5] [7] [9 8]

[2] [5] [8] [9]

$$T(n) = 2 T(\frac{n}{2}) + O(n)$$

running time  
最坏  $O(n^2)$

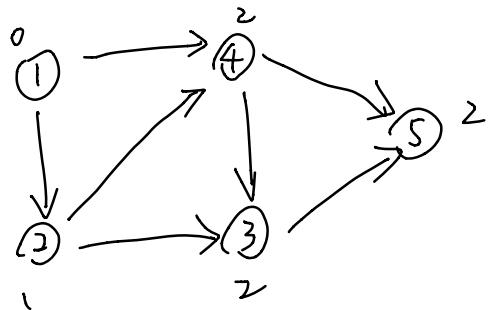
extra space  $O(\log n)$

215 leetcode

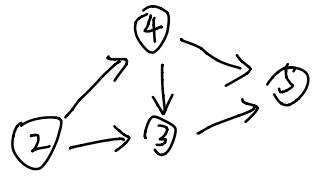
## Topological Sort 拓扑排序

应用场合：将图论里面的顶点按照相连的性质排序

前提：必须是有向图  
图里没有环



输出结果：①



running time:  $O(n)$

① ② ④ ③ ⑤

## Recursion & Backtracking

递归的基本性质：函数调用本身

- 把大规模的问题不断地变小，再进行推导的过程

回溯：利用递归的性质

- 从问题的起始点出发，不断尝试

- 返回一步甚至多步再做选择，直至抵达终点的过程

Recursion : Top- Down

Exercise: 汉诺塔 (hanoi)



算法思想：将问题规模变小

再利用从小规模问题中得出的结果

结合当前的值或者情况，得出最终的结果

Leetcode 91. 解码方法

- 步骤：
- ① 判断当前情况是否非法
  - ② 判断是否满足条件结束递归
  - ③ 缩小问题规模
  - ④ 整合

Leetcode 247 中心对称 II

$$n=1 \Rightarrow 0, 1, 8$$

$$n=2 \Rightarrow 11, 69, 88, 96$$

$$n=3 \Rightarrow \begin{array}{c} 101, 808, 609, 906 \\ \quad \quad | \dots \\ \quad \quad 8 \dots \end{array}$$

$$018, 11, 88, 69, 96$$

$$n=4 \Rightarrow \begin{array}{c} 00, 11, 69, 88, 96 \\ \quad \quad \quad \quad , 11, 88, 69, 96 \end{array}$$

running time:

迭代法：(hanoi)  $T(n) = 2T(n-1) + O(1) = 2 \times 2^n - 1 \Rightarrow O(n) = 2^n$

公式法： $T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n) \quad O(n^{\log_b a})$

- 1°  $O(n^{\log_b^a}) > f(n) \Rightarrow O(n^{\log_b^a})$
- 2°  $O(n^{\log_b^a}) < f(n) \Rightarrow f(n)$
- 3°  $O(n^{\log_b^a}) = f(n) \Rightarrow O(n^{\log_b^a}) \log n$

## Backtracking

回溯算法是一种试探算法，与暴力搜索的最大区别在于：每一步试探都会评估是否继续。

- 精华：
- 出现非法情况时，可退到之前的情况，可返回一步或多步
  - 再去尝试别的路径和办法

- 步骤：
- 首先判断当前情况是否非法 → 非法就返回
  - 看当前情况是否已经满足条件
  - 在当前情况下，遍历所有可能出现的情况，并进行递归
  - 递归完毕后，立即回溯，回溯的方法就是取消前一步进行的尝试

Leetcode 39. 组合总和

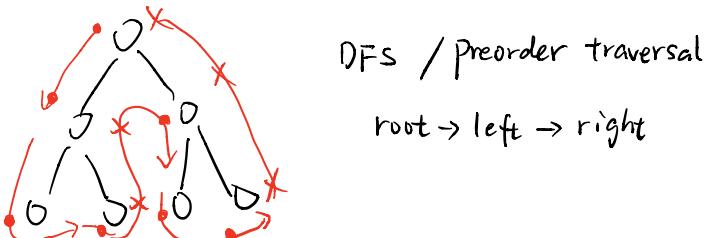
Leetcode 52. N皇后 II

$$T(n) = n T(n-1) + O(n^2)$$

$$T(n) = n \times (n-1) \times T(n-2) + (n-1)^2 + n^2$$

$$O(T(n)) = O(n!)$$

关于 Backtrack 的一些补充（来自 1point3acre）



Leetcode 46 permutations

Leetcode 78 Subsets

Leetcode 113 Path Sum 2

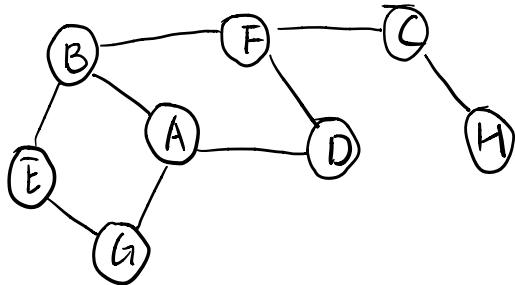
Leetcode 51 N-Queen

Video | YouTube Ravindrababu Ravula  
| bilibili Marty Stepp CS106B lec 9-11

**BFS & DFS** : 对图及树的遍历  
广度优先 深度优先

**DFS**: 解决连通性的问题，即给定一个起始点和一个终点，判断是否有一条路径能从起始点到终点  
(可能有很多条，且不在乎长短，只为了找出一条)

Example



如何进行DFS遍历？

1. 必须依赖 Stack (LIFO)

Stack | A B E G X G X E F C H X H X C D X D X F X B X A

结果: A B E G F C H D  
↓  
都被访问了就弹出

**BFS**: 一般用来解决最短路径问题

广度优先搜索是从起始点出发，一层一层地进行

每层当中的点距离起始点的步数都是相同的

双端BFS: 同时从起始点和终点开始进行的广度优先搜索称为双端BFS

提高搜索效率

例如：想判断社交应用程序中两个人之间需要经过多少朋友介绍才能相互认识

如何进行BFS? 用Queue (FIFO)

Queue | A \* B D G \* E F \* \* \* C \* H

result: A B D G E F C F

• 邻接表 (图里有V个顶点, E条边)  $O(V+E)$

• 邻接矩阵 (V个顶点, E条边) 每次要检查每个顶点与其他顶点是否有联系,  $O(V^2)$

## Dynamic Programming

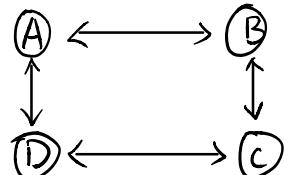
大问题  $\rightarrow$  若干子问题  $\rightarrow$  每个子问题最优解  $\rightarrow$  大问题的最优解

重要属性: ① 最优子结构 Optimal Substructure

- 状态转移方程  $f(n)$

② 重叠子问题 Overlapping Sub-problems

Example:



$A \rightarrow C$  最长 Path, 每个边一次

$A \rightarrow B \rightarrow C : A \rightarrow B \& B \rightarrow C$  (X DP)

$A \rightarrow D \rightarrow C$

### Leetcode 300 最长子序列的长度

给定一个无序的整数数组，找到其中最长子序列的长度 ( $O(n^2)$ )

{ 非空子数组:  $\frac{n(n+1)}{2}$  }

{ 非空子序列:  $2^n - 1$  }

[10, 9, 2, 5, 3, 7, 101, 4]



[10, 9, 2, 5]      [3, 7, 101, 4]

[2, 5]    [3, 7, 101]

$f(n)$  表示数组  $nums[0, 1, \dots, n-1]$  中最长的子序列

$f(1), f(2), \dots, f(n-1)$

解决动态规划问题最难的两个地方：

· 如何定义  $f(n)$ ,  $f(n)$  是以  $nums[n-1]$  结尾的最长上升子序列的长度

· 如何通过  $f(1), f(2), \dots, f(n-1)$  推导出  $f(n)$ , 即状态转移方程

$$f(n) = \max \{f(i)\} + 1, \text{ (其中 } 1 \leq i < n-1, \text{ and } nums[i-1] < nums[n-1]\)$$

计算时间复杂度：递归法，公式法

数据结构 + cache

## 线性规划 Linear Programming

① 各个子问题的规模以线性的方式分布

② 子问题的最佳状态或结果可以储存在一维线性的数据结构中，例如：一维数组，cache

③ Usually we  $dpi[i]$  denotes ith result or best result from 0 ~ i.

Leetcode 70 爬楼梯    Leetcode 198 打家劫舍    Leetcode 62 不同路径

## Interval Programming 区间规划

各个子问题的规模由不同区间来定

子问题的最佳状态或结果可以储存在二维数组中

时间复杂度通常为  $O(n^2)$

Leetcode 516 最长回文子序列

## Constraint Programming 约束规划

在普通的线性规划和区间规划里，一般题目有两种需求：统计和最优解

Example: 0-1 背包问题

给定 n 个物品  $Value_i$ ,  $weight_i$ , 在限定 Max W 内, 如何选择使总  $Value$  最大?

## 二分搜索算法 & 贪婪算法

Binary Search      Greedy

二分搜索算法：变形很多

贪婪算法：直观 & 难以证明正确性

Binary Search 也叫折半搜索

运用前提：数组必须是排好序的，输入不一定是数组，也可能是指定一个区间的起始位置

优点：running time  $O(\log n)$

缺点：要求待查找数组或区间是排好序的

若要对数组进行动态地删除和插入操作并完成查找  $\rightarrow O(n)$

采取自平衡的二叉查找树：

① 构建  $O(n \log n)$

② 搜索  $O(\log n)$

③ 删减和插入  $O(\log n)$

当 input 有序，不常变动，要求从中找出一个满足条件的元素  $\rightarrow$  Binary Search.

0 1 2 3 4 5 6 7 8 9 10

$\downarrow$   
 $=5?$     $>5?$     $<5?$   $\rightarrow$  确定哪个半边

模板 { 递归  
非递归 }

example：找确定的边界，模糊的边界，不定长的边界  $\rightarrow$  double to try the null  $\rightarrow$  遍历到第一个 null 知道总数

Leetcode 33. 旋转过的排序数组

Greedy：每步选择都是最优

背包问题：每次选取价值/重量最大的物品  
|  
| 价值最大  
| 重量最轻  
|  
|  
 $\Rightarrow$  都不一定对

适用场合：局部最优可以解决全局最优

Leetcode 253. 会议室 II

Solution：① 努力的做法

② Greedy

1° 会议都是按照它们的起始时间顺序进行的

2° 给 new meeting 找 room，先看有无 empty room

3° If not, open a new one