

堆序集合 双向链表 非通用 DFS ② 拔木
锯木
伐木

Stack : ① match 标号 . true .

用 index 获得左右括号的下标，一致则匹配

fast 变量列表 → 推导式

② 替换进制 → 余数压栈并反相输出

任意进制 → ① 栈的特性
弹栈下标获取字符串 对应字符
② 一个字符串表示：1-16位

③ 前中后序表达式转换：

① 操作数 → 末尾 左括号 → 压栈

右括号 → 出栈至左括号

运算符 → 压栈前取

优先级更高的操作符出栈

② 中序表达式计算：操作数 → 压栈

操作符 → 计算并压栈

to do : 复习从数据结构单论 >>

网课

gitlab

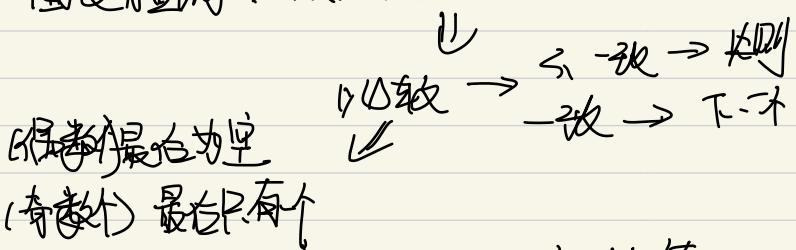
leetcode

NLP 未完

队列：queue：先进先出

插入：循环 m 次，最后一个加入队首，队尾删除
直至只剩一个

双端队列：回文检测：从队首、队尾拿出元素



无序链表：节点 `node` 类 {
 `data` \Rightarrow 插入初始值。
 `next` 下一个元素
} 定义头结点。

UnorderedList 无序链表：

`self.head` 定义头结点。
判断为空 \Rightarrow 如果 `head` 为 `None`,
 \Rightarrow 如果 `head` 不为 `None`,

添加：
① 新建一个 `node` 对象。

② 新节点的 `next` 为当前列表最后一个节点。

③ 将头结点为新头结点。 \Downarrow `self.head`。
当前列表 \Rightarrow 头结点。
第十元素。

遍历：`self.head` 开始 \rightarrow 直到当前元素是 `None`

remove：记住前一个。 前一个的下一个赋给当前的下一个

\Downarrow
`prev = current`

`current = current.next`

`prev.setNext(current.next)`

exception: 删除是第一个 \Rightarrow `previous = None`.

\Downarrow 头 = 当前的下一个。

有向链接

通过有序 \Rightarrow 不用遍历所有元素

当前元素不等于目标元素且大于目标元

素 \Rightarrow 停止查找

add：找到比需要插入大的元素 \Rightarrow 停止

判断此元素是否是头结点 \Rightarrow 第一个元素就以此为头
 $\Downarrow \text{previous} = \text{None}$

① 是头结点 \rightarrow 目标元素

下一个元素为当前头结点

② 目标元素为头结点

prev \leftarrow 上一个元素的下一个元素为当前元素

... 当前元素下一个节点 \rightarrow 当前遍历节点

③ 基本情况

递归：
① 基本情况
② 向基本情况靠近
③ 递归地调用自己

进阶转换：前 10 个数查表

若 $n < \text{base}$ ：直接查表

$n \% \text{base}$

else：function ($n // \text{base}$, base) + 查表结果

汉诺塔，背包问题，斐波那契数列，找零问题，编辑距离

递归 \rightarrow 大问题 \rightarrow 调用自身 \rightarrow 小问题 \rightarrow 基本问题

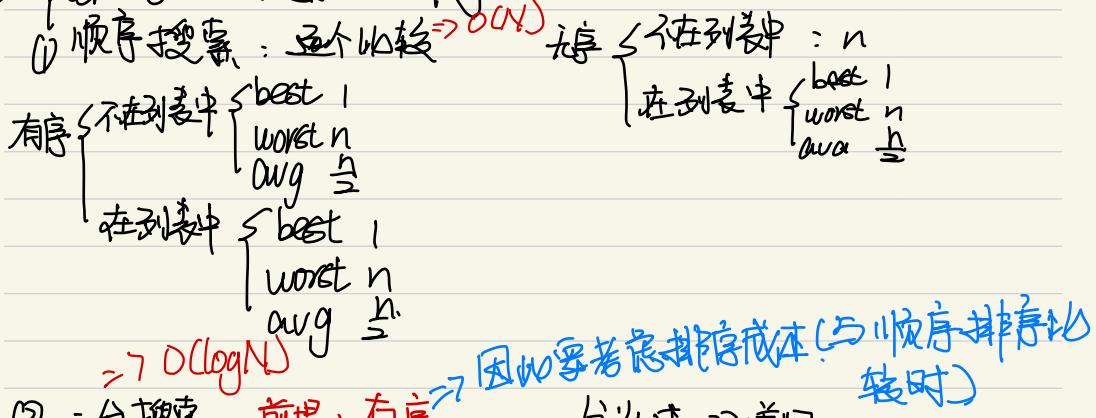
动态规划 \rightarrow 小问题 \rightarrow 叠加成大问题 \Rightarrow 遍历前序所有情况

\Downarrow 增加回溯表

基于上一步的最少步

递归不一定比动态规划高效

Chapter 5 搜索与排序



$$(n/2)/2/2$$

$$\frac{n}{2} \times \frac{1}{2} \times \frac{1}{2} \times \frac{1}{2} = \frac{n}{2^4} = 1 \text{ 最后仅有 一个元素}$$

$$2^i = n \quad i = \log_2 n \quad O(\log n)$$

(3) Hash 散列 $\Rightarrow O(1)$

1) 取余函数 槽 index = 元素值 % 槽的长度
占用率: 负荷因子 $\lambda = \frac{\text{元素个数}}{\text{散列槽数}}$

完美散列函数

冲突数少, 计算方便, 元素均匀分布

散列函数 ① 扩容法

散列表中

所有元素等长相加 $\Rightarrow \%$ 散列长度.

② 平方取中法

$(\text{元素})^2 \Rightarrow$ 中间几位 % 散列长度.

解决冲突: 1) 扩容进散表. 从头找第一个空槽 \rightarrow 链地址法

2) 用 hash

3) 平权探测

4) 冲突处用链表连接

取头入 => 握握程度

开放定址法： $\frac{1}{2}(1 + \frac{1}{r_x})$ 成功 AVG
 $\frac{1}{2}(1 + (\frac{1}{r_x})^2)$ 失败

链接法： $1 + \frac{\lambda}{2}$ 成功 失败入

排序：①冒泡排序 n个元素 第一轮 $n-i$ 次

排 $n-1$ 轮，每一轮排序比上一轮 = $n-2$

少一个，每一次只把最大的数调到最后 = $n-3$

只比以下一个大，如果以下一个大，则交换顺序 ...

$\frac{n-1}{2} n-n+1 = 1$
轮.

$$n-1 + n-2 + n-3 + \dots + n-n+2 + n-n+1$$

$$1+2+3+\dots+n-3+n-2+n-1$$

$\underbrace{\quad\quad\quad}_{n-1 \text{ 个数}}$

$$\frac{(1+n-1)(n-1)}{2} = \frac{n^2-n}{2} = \frac{n^2}{2} - \frac{n}{2}$$

∴ $\mathcal{O}(N^2)$

② 选择排序 改进冒泡，每一轮只交换最大的
选择每一轮最大的元素。 $n-1$ 轮 最后一个元素不用排

只在内层循环结束的时候做一次交换，而不是每一步都

进行一次交换 $\mathcal{O}(N^2)$

比较次数相同，交换次数减少

③ 插入排序 $\mathcal{O}(N^2)$

假设第 0 个元素已在一个有序列表中。

从第一个元素起，前边所有元素已经有序

因此比较从前一个元素，有序列表的最后一个元素开始。

确定是插入步子当前位置或做读向左（此时向左）

元素占据当前位置.

④ 希尔排序

通过步长，切分 为 子列表 \Rightarrow 应用 插入排序.

插入排序相当于 增量为 1 的 希尔排序 \Rightarrow 最后一步是 插入排序

$$O(N) < \text{shellsort} < O(N^2)$$

⑤ 归并排序： 分治法

递归 - 一个列表 不断 = 分， 直至每个列表只有一个元素

合并： 从左右两个列表中 每次挑 较小的那个

如果有一方有剩余，则全部 添加至末尾.

分割： 二分法 $\log n$

合并： 每个元素 需要 处理 n

$$n \cdot \log n.$$

⑥ 快速排序： 分治法

选取基准值.

$$n \log n.$$

选取基准值： 三数取中法： 第一个， 中间， 尾元素

三者找中间值.

查找 $\rightarrow O(N)$
 $\rightarrow O(\log N)$
 $\rightarrow O(1)$

排序： 冒泡 $O(N^2)$
选择排序 $O(N^2)$
插入排序 $O(N^2)$
shell. $O(N) \sim O(N^2)$

额外空间 \Rightarrow 归并排序 $\rightarrow O(n \log n)$ $\{ O(n \log n) \}$
原有左右 \Rightarrow 快速排序 $\rightarrow O(n \log n)$
分割占进中部 \Rightarrow 分割占进中部 $\rightarrow O(n \log n)$

树

① 层次性，越靠叶子节点越细

② X的子结点与另一结点子结点无关

③ 叶子结点无子

① 有一个根节点
② 除根节点外，都只有唯一父节点
③ 从根节点到任何结点，有且只有一条途径
④ 每个结点，只有两个结点 \rightarrow 二叉树

一棵树要么为空，要么由一个根节点和0或多棵子树构成，子树本身是一棵树。
每棵子树的根结点是一条边的父结点

列表之列表，嵌套列表表示子树

0 枚： 1 左子树， 2 右子树

建立树时注意递归，每一次插入，在左子树变成插入子树的右子树都是插入一个子树而不仅是-一个结点。
 \hookrightarrow 子树的 len > 1 (有左、右子树)
若 len ≤ 1 ，则为叶子节点
(根，左子树，右子树)

解析树：遇到左括号，添加左子结点，下沉。

数字 \rightarrow 当前结点值为数字，返回父节点

1. 如何识别父节点？操作符 \rightarrow 设为操作符 \rightarrow 右子树，剥去子树
 \Downarrow
操作符
右括号，回到父节点

2. 如何移动 \Rightarrow 利用当前节点变量的

left, right 方法。

计算解析树，有左右孩子 \rightarrow 递归得到结果
 \nearrow 说明是操作符

没有 \rightarrow 当前是叶子结点 \rightarrow 只能是数字，返回对应的值
 \Rightarrow 递归：如果为空，叶子结点的下一结点
当前空不为空

树的遍历：记录根节点的顺序 前序 \rightarrow 根左右

\Rightarrow 递归：如果为空，叶子结点的下一结点

中序 \rightarrow 左根右

1. 打印根结点

后序 \rightarrow 左右根

变成 binary tree 的方法后，要分别检查左、右子节点，并调用左右子节点的方法（因为无法用递归检查）

二叉堆 \Rightarrow 独占队列

入队/出队 $O(\log n)$

最小堆 优先级小元素

最大堆 优先级大元素

结构属性：完全二叉树，除了叶子结点全都是满的

根 p ，左节点 $2p$ ，右节点 $2p+1$

有序性：根元素小于左右子节点元素

添加元素：队尾添加元素，如果以 父节点小，交换其与父节点，循环

删除元素：将最后一个元素赋值给第一个元素，删除根节点不断与自己最小的子节点比较（因为父节点必须比两个儿子都小），直到不能再交换。

列表构造堆 $O(n)$ 从中间开始，和子节点比较

超过中点的都是叶子节点，全长为 x ，半点 $\frac{x}{2}$ ，中点半量

起始端

$\frac{x}{2}$, $\frac{x}{2}+1$

二叉搜索树 先左后右 二叉搜索性：左节点 小于父节点

parent, key, value, left, right 利用兄弟数 右节点大于父节点

binary search tree 和 treeNode 类

put：如果 没有根节点 \rightarrow key 放到根节点

如果有，分别和其左右节点比较，当不再有任何节点可以比较

插入： \Rightarrow 新建 TreeNode 对象 不要忘记增加偏移量

如果比 X 小，只在 左子树 中寻找 \rightarrow 当此左子树叶子节点时，

插入处一定是叶子结点 且没有任何子节点 \rightarrow 插入，父节点

指向当前叶子节点

get：如果 无root, None

如果是 root, kv 当前小 \rightarrow 左子树遍历

否则 \rightarrow 右子树遍历

当前节点小 \rightarrow 返回当前

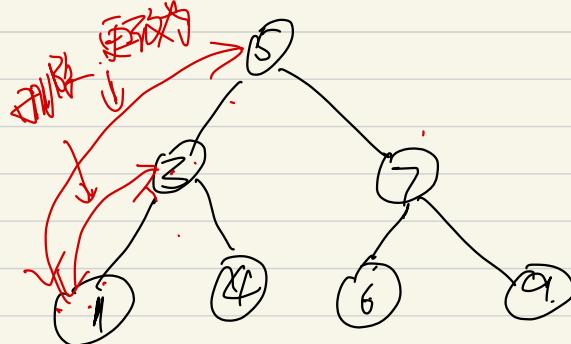
剩余叶子节点也找到 \rightarrow 返回 None

contains

delete.. { 只有一个 Node \rightarrow root
 如果不止一个 node \rightarrow - get 选旧找到要删除的 node
 不等于要删除的 key \Rightarrow 返回 None
 $size - 1$.

三种情况：① 待删除结点是叶子结点 \rightarrow 将其 Parent Node 的 child 设为 None ④ 定有一个父结点

② 待删除结点有一个子节点，且从左到右，连接左子节点和右子节点
 若为右子节点，连接父结点和右子节点
 若当前结点无父节点 \Rightarrow 根节点 \rightarrow replace



③ 待删除结点有两个子节点
找树体结点 \rightarrow 中序遍历顺序 将当前节点设为右孩子
中序后一个 点

有右子数，右子树最左边的结点

没有右子节点，且父结点的左子节点，后继节点为父结点
 没有右子节点，且父结点的右子树，父结点的另一
 后继节点

删除该节点：如果是叶子节点 \Rightarrow 把父节点对应子节点设为 None

否则：如果该节点有任何子节点
 将父节点和子节点连结

否则：将父节点、对应子节点连结

- remove 方法总结：
- ① 叶子节点：父节点设为 None
 - ② 两个节点（左，右）
→ 1) 找到右孩子的
2) 将右孩子的子节点与其次子
连接
3) 替换当前节点
 - ③ 只有一个节点：将对立节点为其父节点连接
如果只有父节点 → 根节点
⇒ 替换为对应子节点

中序遍历二叉搜索树 \Rightarrow yield 生成器 中序遍历 for 循环

完全平衡二叉树： $2^{h+1} - 1$ h：树的高度
总节点数

Put: $O(\log n)$ O(h)

平衡二叉搜索树：不平衡时 O(n)
AVL

叶子节点：平衡因子为 0
其他任何平衡因子
都在根节点看

平衡因子：左右子树的高度之差。
-1, 0, 1 称为平衡。
高度为 h 时，节点数 $N_h = 1 + N_{h-1} + N_{h-2}$

更新 balance factor。如果 $balance\ factor > 1$ 或 $< -1 \Rightarrow$ 平衡
如果是左节点，+1；右节点，-1。如果 parent balance
 $= 0$ ，继续向上 update \Rightarrow 直到 root。

子树 balance factor = 0，祖先不再有变化
新插入节点对父节点平衡因子有影响

return 空字符串

函数

↑ 旧根节点右孩子

左旋：新根结点抬升，旧根节点之新根结点之左子树，此时旧根结点无右孩子（原右子节点已成为新根节点），将新根节点原有的左子节点作为新左子节点（旧根的右孩子，新根左子结点的父节点，为旧根左）

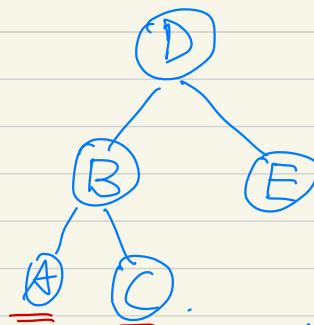
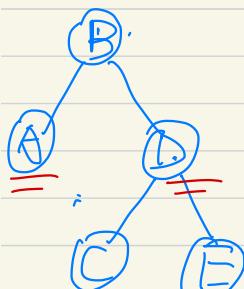
：新根节点父节点 → 旧根节点

旧根如果是整棵树根结点 → 更改根结点为新根节点
如果不是，将对应分支指向新根结点

新根结点 → 左子节点 → 旧根

旧根 → 右子节点 → 新根

更新 balance factor



$$\text{old}(B) = h_A - h_D$$

$$\begin{aligned} \textcircled{1} - \textcircled{2} &= h(A) - h(D) - (h(A) - h(C)) \\ &= -h(D) + h(C) \quad \textcircled{4} \end{aligned}$$

$$\textcircled{3} h(D) = 1 + \max(h_C, h_E)$$

∴ 把 $\textcircled{3}$ 代入 $\textcircled{4}$ 得

$$-h(C) + (1 + \max(h_C, h_E))$$

$$= 1 + \max(h_C, h_E) - h(C) \quad \because \max(a, b) - c = \max(a - c, b - c)$$

$$\therefore = 1 + \max(h_C - h_C, h_E - h_C) = \max(h_C, h_E)$$

$$= 1 + \max(0, \underline{h(E) - h(C)}) = -h(D). \text{old}$$

$$\text{new}(B) = \text{old}(B) + 1 + \max(0, -h(D)).$$

$$= \text{old}(B) + 1 - \min(D) \sim h(D)$$

new node

$$\text{oldroot}, BF^{\bar{D} \text{ root}} = \text{oldroot}, BF + 1 - \min_{(\text{new node}, BF)}$$

$$\text{old}(D) = h(C) - h(E)$$

$$\text{new}(D) = h(B) - h(E)$$

$$\therefore \text{new}(D) - \text{old}(D) = h(B) - h(E) - h(C) + h(E)$$

$$= h(B) - h(C)$$

$$\therefore \text{new}(D) = \text{old}(D) + h(B) - h(C)$$

$$= \text{old}(D) + 1 + \max(h(A), h(C)) - h(C)$$

$$= \text{old}(D) + 1 + \max(h(A) - h(C), 0)$$

$$= \text{old}(D) + 1 + \max(h(B), 0)$$

左旋，检查右子树。右子树左倾 \rightarrow 右子树右旋
 \rightarrow 左旋。

右旋 杠直左子树；左子树右倾 \rightarrow 左子树左倾
 \rightarrow 右倾；else，直接左倾

映射	列表	散列	二叉搜索 AVL
put.	$O(n)$	$O(1)$	$O(n)$ $O(\log n)$
get	$O(\log n)$	$O(1)$	$O(n)$ $O(\log n)$
in	$O(\log n)$	$O(1)$	$O(n)$ $O(\log n)$
del	$O(n)$	$O(1)$	$O(n)$ $O(\log n)$

第七章 图：

顶点(端)：有效载荷
边 有向/双向
权重

$$G = (V, E)$$

$e \in E$ $v, e \in V$
子图

路径 $(v_i, v_{i+1}) \in E$
无权：边
有权：权重之和

环：起/终点相同

direct acyclic graph DAG

Graph()

addVertex / addEdge.
get.

邻接矩阵：行列数相等，单元格表示行列的权重

优点：简单，直观显示哪些相连

缺点：稀疏，低效

邻接表

每个顶点，记录与其相连的顶点



dict {↓; 权重}

vertex：一个端点与权重之值，和对应直接 dict

Graph：用字典储存对应关系

广度优先搜索

词梯问题：任意两个只差一个字母的单词，都可以连接起来
优化

用 bucket，把符合 bucket 条件的单词放一个桶中，同一桶内的单词相连

词梯实现 $\text{Pattern} = [\text{word}_1, \text{word}_2]$

$\text{Pattern} = i \text{ for } i \text{ in } \text{len}(\text{word})$
 $\text{word}[1:i] + \dots + \text{word}[i+1:]$

BFS：从第一个开始，加入队列；
遍历 neighbor，依次将设置为加入 neighbor 的 neighbor
注意邻居可能是 neighbor，再遍历 neighbor，
遍历完所有 neighbor，再

回溯：打印当前 ID，一直有 pred，打印 pred ID,
 $O(V+E)$

DFS：建立图

建立骑士图：建立位置 \rightarrow 邻接
获取可能的下一位置
 \Downarrow $i \rightarrow i+1 \Rightarrow$ 加入图

- ① disable the point can be used
- ② unvisited

DFS \rightarrow recursion

深度优先搜索树 \rightarrow 多棵 \rightarrow 广度优先搜索树

通用 \rightarrow 设定所有子节点未访问 \rightarrow 沿向某个 (灰色)
 \rightarrow 病历白色白勺子节点 \rightarrow 遍历
病历完成所有子节点 (设为黑色)

任一节点的子节点, 比父节点, 发现晚, 结束早 (0.000)
叶子节点, 开始结束早 |

拓扑排序 次序 / 事件优先级 DAG \rightarrow 包含所有节点的线性序列
 \rightarrow 所有子节点, 病历完的时间

① DFS. 因存储 \rightarrow 结束时间高成:
最近结束的排第一个

强连通分量: 网站的聚集

聚集簇 \Rightarrow 强连通分量 C 为最大点子集 $C \subset V$
对每对顶点 $W, U \subset C$, 要有一条从 U 到 W 和
从 W 到 U 的路径

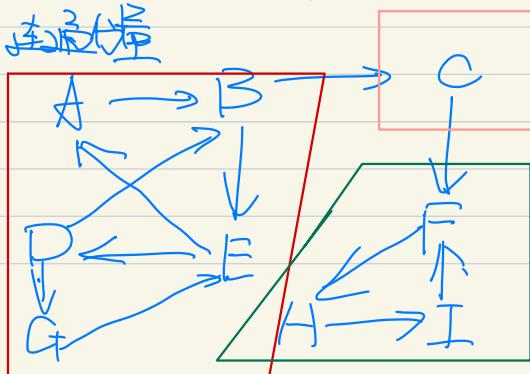
图的转置 $G \rightarrow G^T$. 所有边与 图的方向相反
两个图强连通分量不变

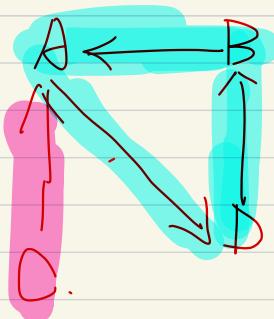
(1) DFS, 节点结束时间

G^T

(3) $G^T \rightarrow$ DFS, 结束时间逆成访问

强连通





$G \xrightarrow{G^T}$ 互通分量
子图

最短路径

权重总和 \Rightarrow 如果权重相等 \Rightarrow 路径长短

只适用于权重为正的情况，即尚完整的图

① Dijkstra 迭代：某个顶点到其他所有顶点的最短路
 $O((V+E)\log V)$

$dist$ 起点到当前顶点最小权重路径的总权重
将该顶点放入一个优先队列 \Rightarrow 引进 $(dist, v)$ v 的总权值最小
依次 \rightarrow 最小的 (不仅是二叉堆)

遍历其所有 neighbor. $dist = \text{目前节点 distance}$

- 一个点到其余各点。
+ 该节点到下一节点

(neighbor) 的 权重

进而求 $dist$ 下一节点的 distance:

$dist = \text{下一节点的 distance}$

下一点的前驱 \Rightarrow 当前结点

Priority queue. decrease key (next vertex, dist)
 \downarrow

移至头部

检查每个顶点，看是否由当前顶点到达其相邻顶点比
其相邻顶点的路径更近。

Greedy (每次选择权重最小边) \rightarrow kruskal \rightarrow 挂起 \rightarrow 合并 \rightarrow 并查集

② Prim 算法 最小生成树： 树 T 是 $G(V, E)$ 中 V 的子集
包含所有节点的总权值。 子集 并连接 VR 有向边

总权值是一条最小树连接所有边 (不是单点相连的路径)

① 找到一条 safe edge ② 新边加入 T 直至 T 是一个生成树
一边在 T 中，一边不在 T 中
保证不出现环。

优先队列。
最小顶点。

最小顶点的 neighbor.

now cost = neighbour - 当前 edge + 与向边的
edge

第8章：附加： Python 列表 \Rightarrow C 的数组

读字节块

1. 能操作相同

类型
操作：索引/赋值

元素地址 = 起始地址 + 元素下标 \times 元素大小

① 值存指向其他句柄的引用 (指针)

② 过度分配 (比所需大的内存)

③ 填满后，分配更大数组。复制。

append：检查是否超过下标：超过 \Rightarrow 创建一个更大的数组
将旧的值 copy。
将新元素插入末尾。
 $i = 2^n$ lastIndex = 1
 $n + 1$

尾插扩容

$$KAYAK = n + \sum_{j=0}^{\lfloor \lg n \rfloor} 2^j \Rightarrow \begin{array}{l} \text{需要几次遍} \\ \text{扩容次数多少} \end{array}$$

$$= n + 2n = 3n \quad 3n/n = 3$$

insert: 从位开始复制: 最后一个先移下一位。
 n
 再将 $n-1$ 移到 n , 不会覆盖

质因数分解

mod % : 结果 0 - 9

恺撒密码 not B 移动 n 起始部分回补开头 (取模)

每个字母位置移动所对应的字母

对称性: 同一函数加密 / 解密

取模参数 \rightarrow 密钥

同余定理: a 和 b 对 n 取模有相同的同余 $a \equiv b \pmod{n}$

$a \equiv b \pmod{n} \therefore \forall c, a+c \equiv b+c \pmod{n}$

$\forall c, ac \equiv bc \pmod{n}$

$\forall p, p > 0, a^p \equiv b^p \pmod{n}$

$$x^n = \begin{cases} (x \cdot x)^{\frac{n}{2}} & n \text{ 为偶数} \\ (x \cdot x^{n-1}) = x(x \cdot x)^{\frac{n-1}{2}} & n \text{ 为奇数} \end{cases}$$

最大公因数与逆元

$ax \equiv 1 \pmod{m}$ $x > 0$ a 互质
 m 为 x 互质, x 关于 m 才有逆元
 \curvearrowleft 最大公因数为 1

RSA: 大素数 p, q $n = p \times q$ e/n 公钥
密钥 e 与 $(p-1)(q-1)$ 互素
 $\text{gcd}(e, (p-1)(q-1)) = 1$ 私钥

加密 $C = m^e \pmod{n}$
解密 $m = C^d \pmod{n}$

字符串 \rightarrow 十六进制 \rightarrow 十进制

跳转表 \rightarrow 二位键表

逐位，向下引用，向上引用

表头 向下引用 \Rightarrow 下层表头
向上引用 \Rightarrow 数据节点链接

数据节点，自底到顶 \rightarrow 塔

0 层：所有节点
每一层节点数与塔的高度有关

左边的大于右边的
左边的小于右边的

塔的层数 \rightarrow 抽屉原理，完全随机
循环插入

more info:
归并，Morita 算法，KMP 算法

量化图片 \Rightarrow 八叉树

统计每种颜色的数目，
从最少的开始 \Rightarrow 合并该叶子
节点，加入兄弟节点 \Rightarrow 直到小于
要求的节点数

通过原图片，像素点在八叉树中找到对应节点

更新图片

模式匹配. pattern = substring 子串

DNA: 腺嘌呤, 胸腺嘧啶, 鸟嘌呤, 胞嘧啶
(A) (T) (G) (C)

暴力法：每个字符都进行比较，如果匹配，就开始比较下一个，否则开头往左移

确定有限状态自动机：DFA $O(n)$

每匹配一个字符，进入下一个状态，如果不匹配。
状态数为 $O(n)$ 状态由下一个字符决定状态。

KMP算法 Knuth - Morris - Pratt

构建图，不匹配时提供‘滑动’的必要信息。

无论是成功都读入下一个字符

DFA易于使用，不易于构建； KMP既易于使用又易于构建