算法基础

Foundation of Algorithms

余启帆

中国科学技术大学

2022 年 1 月 6 日

目录

第 0 章　绪论

第一部分

基础知识

第 1 章　算法概念

1.1　算法的作用

1.1.1　一些概念

1 算法

(1) 输入

(2) 输出

† Announced above the program!

2 问题

3 问题实例

例题 1.1 (排序)

(1) 稳定排序

(2) 不稳定排序

† 就地排序

计数排序 (Counting Sort)

4 算法的正确性

† 正确性

† 有限性

1.1.2　算法与问题求解

应用问题 $\Longrightarrow$ 计算问题

1 问题求解的过程

(1) 将一个应用问题, 加以分析, 给出相关的 (数学、计算) 模型;

(2) 将应用问题及其模型 $\Longrightarrow$ 具体的计算问题;

(3) 选择和设计有效求解这些计算问题的算法, 进行性能分析;

(4) 选择合适的数据结构, 将算法加以实现, 设计出求解应用问题的系统或程序.

1.2　作为一种技术的算法

第 2 章　算法基础

2.1　何为算法?

2.1.1　插入排序

2.1.2　循环不变式

† 抽取什么作为 “循环不变式”?

1 初始化

2 保持

3 终止

2.1.3　用循环表达式证明插入排序算法的正确性

第 j 次循环时, $A[1\cdots j-1]$ 已经有序, 插入 $A[j]$ 后, $A[1\cdots j]$ 仍然有序

1 初始化　$A[1\cdots j-1]=A[1]$ 有序

2 保持　一次循环后, $A[1\cdots j]$ 仍然有序

3 终止　每次迭代 `j += 1`, 因此必有 `j = n + 1` 时, 循环终止.

2.1.4　算法的正确性

† 功能正确: 输入输出行为正确

† 正确性的分类

(1) 部分正确性: 要求算法若返回结果, 返回结果时, 这一结果是正确的;

(2) 完全正确性: 要求算法必须能够结束.

2.2　分析算法

† 通常用

▷ 运行时间

▷ 内存空间

来评判算法需要的资源

† 客观标准　评价体系　计算模型

2.2.1　计算模型

1 Random-access Machine(RAM 模型)

(1) 单位时间运算
(2) 单位时间内存访问
(3) 不区分数据类型

2.2.2 影响运行时间的主要因素

1 输入规模 number of **bits**

† 明确 “输入规模” 和 “运行时间” 的概念

2 输入数据的分布

3 将算法的运行时间描述成输入规模的函数

4 * 算法实现所用的底层数据结构

2.2.3 基本概念

1 输入规模

(1) 项数
(2) 最大数的位数

2 运行时间 基本操作数与机器无关, 运行每行需要常量时间

3 最好、最坏、平均运行时间

† 通常用**最坏**运行时间来衡量

抽象常数 C, 考虑运行时间的主要贡献 (首项)

2.3 设计算法

2.3.1 分治法

分解 - 解决 - 合并

† 子问题规模不一定相同

1 Merge Sort 对 n 个元素排序 $\Longrightarrow$ 对 $\frac{n}{2}$ 个元素排序 $\Longrightarrow \cdots \Longrightarrow$ 合并

2 监视哨 (哨兵) 在数组中增加一个不出现的特殊数字/字符, 从而减少变量的比较次数.

3 使用哨兵的归并排序 通过需要复制的项数控制循环的终止, 在两组数据末端均添上 ∞, 从而一组数据复制完后, 被复制的总是另一组数据.

4 不使用哨兵的归并排序 当一个数组被复制完后, 直接将另一数组的剩余部分复制.

2.3.2 分析分治算法

$$T(n) = \begin{cases} O(1), & n \leqslant c, \\ aT\left(\frac{n}{b}\right) + D(n) + C(n), & \text{else} \end{cases} \tag{2.1}$$

† $D(n)$: 分解所需的时间

† $C(n)$: 合并所需的时间

1 归并排序的时间复杂度

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n), & n > 1 \end{cases} \implies T(n) = \Theta(n \log n) \tag{2.2}$$

第 3 章　函数的增长

3.1　渐近记号

3.1.1　Θ-notation

$$f(n) = \Theta(g(n)) \iff C_1 g(n) \leqslant f(n) \leqslant C_2 g(n) \tag{3.1}$$

3.1.2　O-notation

$$f(n) = O(g(n)) \iff 0 \leqslant f(n) \leqslant Cg(n) \tag{3.2}$$

† $f(n)$ 无穷大的阶不大于 $g(n)$ 的阶
† 若 $f(n) = \Theta(g(n)) \implies f(n) = O(g(n))$

3.1.3　Ω-notation

$$f(n) = \Omega(g(n)) \iff 0 \leqslant Cg(n) \leqslant f(n) \tag{3.3}$$

† $O(n)$ 和 $\Omega(n)$ 分别是 $\Theta(n)$ 的右半和左半

第 4 章　分治策略

4.0.4　求解递归式的方法

1 代入法

2 递归树法

3 主方法

4.1　代入法 (数学归纳法)

$$T(n) = 2T\left(\left[\frac{n}{2}\right] + 17\right) + n \tag{4.1}$$

$$g(m) = T(m + 17) \tag{4.2}$$

$$\Longrightarrow\ S(n) = 2g\left(\frac{n}{2}\right) + n \tag{4.3}$$

4.2　递归树 (用于估算紧致解)

先分解, 后合并

4.3　主方法 (主定理)

定理 4.1 (主定理)　递归式

$$T(n) = aT\left(\frac{n}{b}\right) + f(n), \quad a \geqslant 1,\ b > 1, \tag{4.4}$$

(1) 若 $\exists \varepsilon > 0$, $f(n) = O(n^{\log_b a - \varepsilon})$, 则 $T(n) = \Theta(n^{\log_b a})$;

† 主要贡献来自 $aT\left(\frac{n}{b}\right)$

(2) 若 $f(n) = \Theta(n^{\log_b a} \lg^k n)$ $(k \geqslant 0)$, 则 $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$;

† 特别地, $k = 0$ 时, 若 $f(n) = \Theta(n^{\log_b a})$, 则 $T(n) = \Theta(n^{\log_b a} \lg n)$;

(3) 若 $\exists \varepsilon > 0$, $f(n) = \Omega(n^{\log_b a + \varepsilon})$, 且 $\exists c < 1$, $\forall n > N$, 有 $af\left(\frac{n}{b}\right) \leqslant cf(n)$, 则 $T(n) = \Theta(f(n))$.

† 主要贡献来自 $f(n)$.

注意　$\varepsilon > 0$ 的存在性问题!

第二部分

排序与选择

第 5 章　基于比较的排序算法

5.1　排序基本概念

5.1.1　稳定性

相同数值顺序在排序前后不变

5.1.2　内排序与外排序

排序过程是否全部在内存中进行

5.1.3　前述排序算法

1 插入排序

2 归并排序

5.2　简单排序 & Shell 排序

5.2.1　简单选择排序

每次未排序子数组的最小值调整至最前面

† 不稳定, $\Theta(n^2)$

5.2.2　冒泡排序

每次比较相邻两元素

5.2.3　Shell 排序

对于同一组数据, 调用多次插入排序反而比只用一次插入排序要快: $n^2 \geqslant 3 \cdot \left(\frac{n}{3}\right)^2$

† 2-有序或 3-有序序列至多 n 次比较变为 1- 有序序列 (间隔为 2 的序列分别有序)

† 4-有序或 6-有序至多 n 次比较变为 3- 有序序列.

5.3　堆排序

5.3.1　二叉堆

完全二叉树

本书中:

(1) 完全二叉树: 指 “满二叉树”
(2) 满二叉树: 指 “严格二叉树”

(1) 完全二叉树
(2) 满二叉树
(3) 严格二叉树: 每个非叶结点有且只有两个子节点

5.3.2 大根堆、小根堆

堆排序中使用大根堆
堆用数组存储: 第 i 个结点的左右叶子结点分别是 $2i, 2i+1$
高度: 边的高度 $= [\log n]$ (数据结构中定义为结点的高度)

5.3.3 维护堆

大根堆调整: 递归

5.3.4 建堆

自底向上, 从最后一个结点的父节点开始做大根堆调整
时间复杂度: $O(n)$
† 为何不是 $O(n \log n)$?

5.3.5 堆排序

将堆顶元素和堆尾元素交换, 然后剥离最后一个元素, 调整新的堆为大根堆

5.4 快速排序

† 分治思想: 先划分, 后合并
† 归并排序: 先各自排序, 后合并 (合并时工作较多)

第 6 章　线性时间排序算法

6.1　排序算法的下界

任何基于比较的排序算法最坏情况时间复杂度必须为 $\Omega(n \log n)$

1 决策树模型　至少有 $n!$ 个叶结点, 至少有 $2n! + 1$, 因此高度至少为

$$h \geqslant \lfloor \log_2(2n! + 1) \rfloor \geqslant \lg(n!) = \Omega(n \lg n) \tag{6.1}$$

6.2　计数排序

† 稳定排序

6.3　基数排序

6.3.1　思想

从最低位开始, 依次对每一位进行稳定排序 (例如, 选取计数排序)

6.3.2　性能分析

For n b-bit binary numbers, view r-bit as a digit, then number of digit $d = \left\lceil \frac{b}{r} \right\rceil$,

$$T(n) = \Theta\left(\frac{b}{r} \cdot (n + 2^r)\right) \tag{6.2}$$

Optimal value r:

$$r = \begin{cases} b, & b < \lfloor \log n \rfloor, \\ \lfloor \log n \rfloor, & b \geqslant \lfloor \log n \rfloor \end{cases} \tag{6.3}$$

6.4　桶排序

第 7 章　选择问题

† 中值: $\left\lfloor \frac{n+1}{2} \right\rfloor$

7.1　选择问题

第 i 小的元素: 有且只有 $i-1$ 个元素比它小

7.2　最大值和最小值

1 找最大值/最小值

2 同时找最大值 & 最小值　先找最大值, 再找最小值, 这样至少需要 $2n-3$ 次比较.

同时找最大值 & 最小值问题需要元素间比较次数至少为 $\left\lceil \frac{3n}{2} \right\rceil - 2$

证明　所有元素都可以划分为以下 4 种类型, 记:

N: 从未参与比较;

S: 参与过比较, 总是小;

L: 参与过比较, 总是大;

M: 参与过比较, 时大时小.

每次比较:

$N:N$: 可得到两个信息 (元素改变一次身份记为一条信息)

$N:S, N:L$: 最坏情况只能获得一个信息

$S:L, M:L, M:S, M:M$: 最坏情况只能获得 0 个信息

初始: n 个 N

结束: 1 个 S, 1 个 L, $n-2$ 个 M

需要 $1+1+2\times(n-2)=2n-2$ 次信息转换.

希望比较次数尽可能少, 因此每次比较获取信息尽可能多, 因此优先安排 $N:N$ 型比较, 最多 $\left\lfloor \frac{n}{2} \right\rfloor$ 次, 获得 $\left\lfloor \frac{n}{2} \right\rfloor \times 2$ 条信息, 还需 $2n-2-\left\lfloor \frac{n}{2} \right\rfloor \times 2 = 2\times\left\lceil \frac{n}{2} \right\rceil - 2$ 次比较, 因此总共至少需要 $\left\lfloor \frac{n}{2} \right\rfloor + 2\times\left\lceil \frac{n}{2} \right\rceil - 2 = \left\lceil \frac{3n}{2} \right\rceil - 2$　□

7.3　平均性能为 $\Theta(n)$ 的选择算法

† 沿用快速排序中 Partition 的递归算法

第三部分

算法设计基本策略

第 15 章　动态规划法

15.1　动态规划原理

动态规划适用于子问题重叠的情况, 每个子问题只求解一次, 然后存储求解结果, 而分治法可能对于同一个子问题反复求解.

15.1.1　最优化问题

包含一组约束条件 + 一个优化函数

† 可行解

† 最优解

15.1.2　最优子结构 (最优性原理)

1 最优子结构　“乘法原理”: 已经进行一部分决策, 剩下的决策要对于已经达成的状态进行最优的决策

如果一个问题的最优解包含其子问题的最优解, 则称此问题具有 “最优子结构”

† 全局最优一定局部最优

▷ 剪切法

用动态规划方法求解问题, 用子问题的最优解来构造原问题的最优解.

2 判断是否具有 “ 最优子结构”

(1) 无权最短路径: 具有最优子结构

(2) 无权最长简单路径: 不具有最优子结构, 子问题的解可以使原问题的解出现环路 (非简单路径)

15.1.3　重叠子问题

(1) 反复求解重叠子问题 $\Longrightarrow$ 动态规划法 (备忘)

(2) 每次生成不同子问题 $\Longrightarrow$ 分治法

† 通过是否带备忘功能来区分

15.1.4　动态规划法求解步骤

1　刻画一个最优解的结构特征: 最优子结构, 全局最优一定局部最优

2　递归地定义最优解的值: 列出递推式, 并化简 (可能可以通过修改递推式减少需要求解的子问题数量, 如钢条切割问题)

3 计算最优解的值, 通常采用自底向上或带记忆功能的自顶向下方法 (从子问题的解构造原问题的解), 同时记录用于重构最优解的辅助信息

4 利用计算出的信息构造一个最优解: 递归输出

† 区分最优解的**值**和最优解, 分别对应最优结果和方案. 例如, 在导航问题中,

▷ 值: 时间

▷ 最优解: 路线

15.2 钢条切割问题

15.2.1 递推式

$$r_n = \max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \cdots, r_{n-1} + r_1\} \tag{15.1}$$

求解每个子问题:

$$r_1 + r_{n-1} = p_1 + r_{n-1} \tag{15.2}$$

$$r_2 + r_{n-2} = \max\{\underline{p_1 + p_1 + r_{n-2}}, p_2 + r_{n-2}\} \tag{15.3}$$

† 划线问题已经在前一个子问题中被解决, 不用再考虑

▷ 每次只用考虑 1 个新的子问题

$$r_n = \max_{1 \leqslant i \leqslant n}\{p_i + r_{n-i}\} \tag{15.4}$$

讨论 (1) 当 $P[1 \cdots n]$ 满足什么条件时, 不需要切分? (不需切分的充分条件)

讨论 (2) 是否存在必要条件? (不需切分的必要条件)

15.2.2 计算

1 自底向上方法 先求解规模较小的子问题

2 自顶向下方法 — 带记忆功能的递归 始终维护同一个数组

† 避免重复求解子问题是动态规划法的核心!

15.2.3 重构解

维护数组 $s[j]$ 表示长为 j 的钢条从 $(s[j], s[j]+1)$ 中间切分.

15.2.4 时间复杂度

$\Theta(n)$ 个子问题 $\implies T(n) = O(n^2)$

15.3 矩阵链乘问题

n 个矩阵链乘的排列数: 卡特兰数 $\dfrac{1}{n+1}\binom{2n}{n}$

考虑计算 $A_i, \cdots, A_j$ 这 $j-i+1$ 个矩阵所需的最小乘法次数.

只需知道 $p=[p_0, p_1, \cdots, p_n]$ 这 $n+1$ 个数, 从而 $A_1: p_0 \times p_1, A_2: p_1 \times p_2, \cdots, A_i: p_{i-1} \times p_i, \cdots$

假设从 A_k 处划分最优, 则 $A_i \cdots A_k$, $A_{k+1} \cdots A_j$ 的方案也应当是最优的 (全局最优一定局部最优 $\implies$ 满足最优性原理 $\implies$ 可以用动态规划法求解)

记 $m[i,j]$ 是计算 $A_i, \cdots, A_j$ 所需的最小乘法次数, 则

$$m[i,j] = \min_{i \leqslant k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} \tag{15.5}$$

然后用 Bottom-Up 方法求解 $m[i,j]$ 对 $i,j = \{1,2,\cdots,n-1\} \times \{i+1, \cdots, n\}$ 循环 $\iff$ $l(=j-i), i = \{1,2,\cdots,n-1\} \times \{1,2,\cdots,n-l\}$ 循环, 类似于求和的交换顺序:

$$\sum_i \sum_j \cdot = \sum_{j-i} \sum_i \cdot \tag{15.6}$$

先对每一个固定的 $j-i$ 去求和

15.3.1 时间复杂度

$\Theta(n^2)$ 个子问题 $\implies T(n) = O(n^3)$

15.4 最长公共子序列

† 注意区分子序列和子串!

15.4.1 问题

问题 15.1 (最长公共子序列, Longest-common-subsequence, LCS) 两个序列 $X = \{x_1, x_2, \cdots, x_m\}$, $Y = \{y_1, y_2, \cdots, y_n\}$, 求 X, Y 长度最长的公共子序列.

按照动态规划法的 4 个步骤求解.

15.4.2 刻画最长公共子序列的特征

定理 15.1 (LCS 的最优子结构) 设 $Z = \{z_1, z_2, \cdots, z_k\}$ 是 X, Y 的任意 LCS. 考虑 X, Y, Z 的最后一个元素.

(1) 若 $x_m = y_n$, 则 $z_k = x_m = y_n$ 且 Z_{k-1} 是 X_{m-1}, Y_{n-1} 的一个 LCS;

(2) 若 $x_m \neq y_n$,

(a) 则 $z_k \neq x_m$ 表示 Z 是 X_{m-1}, Y 的一个 LCS;

(b) 则 $z_k \neq y_m$ 表示 Z 是 X, Y_{m-1} 的一个 LCS.

15.4.3 构造递归解

定义 X_i, Y_j 的 LCS 长度

$$c[i,j] = \begin{cases} 0, & i=0 \text{ or } j=0 \\ c[i-1,j-1]+1, & i,j>0,\ x_i = y_j \\ \max(c[i,j-1], c[i-1,j]), & i,j>0,\ x_i \neq y_j \end{cases} \tag{15.7}$$

15.4.4 设计 Bottom-up 方法求解

```
c[0, 1..n] = c[1..n, 0] = 0
for i = 1 to n:
    for j = 1 to n:
        # Calculate c[i, j]
```

15.4.5 Further: 输出所有 LCS

记录所有可能的路径 $b[i,j]$, 注意到不同的 LCS $\iff$ 斜边不同, 因此遍历输出时, 考虑两层递归:

(1) 主递归: 对每条不同斜边进行 DF 遍历, 对走过斜边后的下一个点 (斜边左上角的点) 进入辅助递归

(2) 辅助递归: 返回从每个点出发可达的第一条斜边

† 大大降低时间复杂度!

15.5 最优二叉检索树

结点 k_i, 伪结点 $d_i : (k_i, k_{i+1})$, 平均检索深度定义为平均访问的结点数量

$$E[depth] = \sum_{i=1}^{n}(depth(k_i)+1)\cdot p_i + \sum_{i=0}^{n}(depth(d_i)+1)\cdot q_i \tag{15.8}$$

† 也有称为 “比较数” 的, 但这种称呼并不确切, 要注意检索到 k_i 与 d_{i-1}, d_i 的比较次数相同

† 若概率相同 $\implies$ 折半查找树

最优二叉检索树: 期望检索代价最小的二叉检索树

结点 $k_1, k_2, \cdots, k_n$, 伪结点 $d_0, d_1, \cdots, d_n$

包含 $k_i, \cdots, k_j$ 的子树概率之和

$$w[i,j] = \sum_{k=i}^{j} p_k + \sum_{k=i-1}^{j} q_k \tag{15.9}$$

最小代价递归公式

$$e[i,j] = \begin{cases} q_{i-1}, & j = i-1 \\ \min\limits_{i\leqslant r\leqslant j}\{e[i,r-1]+e[r+1,j]+w[i,j]\}, & i \leqslant j \end{cases} \tag{15.10}$$

辅助数组 $r[i,j]$ 记录包含 $k_i, \cdots, k_j$ 的子树的根节点 k_r.

15.5.1 改进: 时间复杂度为 $O(n^2)$ 的算法

仍是 3 层循环, 减小循环范围:

$K_{i\cdots j}$ 比 $K_{i\cdots j-1}$ 多 2 个结点: K_j, d_j

设 $K_{i\cdots j-1}$ 的解为 K_{r_1}, 直接将 2 个结点加到右子树, 右边子树变 “ 重”, 因此后者最优解的根节点 $r_0 \geqslant r_1$

设 $K_{i+1\cdots j}$ 的解为 K_{r_2}, 同理有 $r_0 \leqslant r_2$

$\Longrightarrow\ r_1 \leqslant r_0 \leqslant r_2$

即, 循环范围改为

```
for r = r[i, j − 1] to r[i + 1, j]
```

对于同一个 l, 不同的 i, r 的循环范围不重叠[1] $\Longrightarrow$ 对同一个 l, 时间复杂度为 $\Theta(n)$ $\Longrightarrow$ 总的时间复杂度 $\Theta(n^2)$.

[1]这是因为 $r[i+1,(j+1)-1] = r[i+1,j]$, 即下一次 i 循环的 r 下界 $=$ 上一次循环的上界.

第 16 章　贪心算法

† 逐步构造局部最优解

† 替代动态规划法, 求解最优化问题

16.1　贪心算法原理

16.1.1　贪心算法

1 最优子结构　全局最优一定局部最优

2 贪心选择性质　局部最优一定全局最优

† 可用贪心算法: 其中一个最优选择为贪心选择

通过对输入进行预处理或使用合适的数据结构 (通常是优先队列), 可以减少贪心选择时要考虑的选项数, 使算法更高效.

† NP 完全问题

† TSP 问题: 贪心策略求解平均获得最优解 85 % 的问题

16.1.2　是否能用贪心算法求解?

1 0-1 背包问题　金块不可拆分

2 分数背包问题　金块可拆分

都具有最优子结构, 但只有后者具有贪心选择性质, 能够用贪心算法求解.

16.2　活动选择问题

† 对同一资源进行安排

▷ 排课表

问题 16.1 (活动选择问题)　求解最大相容的**活动个数**及相应集合

† 只要求活动个数最大, 并不要求资源利用的总时长最长.

16.2.1　最优子结构与动态规划法

定义

(1) 集合 $S_{ij} = \{a_k | f_i \leqslant s_k < f_k \leqslant s_j\}$ 为在 a_i 结束之后开始且在 a_j 开始之前结束的活动集合,

(2) A_{ij} 为 S_{ij} 中最大相容子集,

(3) $c[i,j]=|A_{ij}|$

动态规划法

$$c[i,j]=\begin{cases}0, & S_{ij}=\varnothing,\\ \max\limits_{a_k\in S_{ij}}\{c[i,k]+c[k,j]+1\}, & S_{ij}\neq\varnothing.\end{cases}\tag{16.1}$$

† 每个问题生成两个子问题 $\Longrightarrow$ 指数时间 $\Longrightarrow$ 如何让其中一个子问题退化?

16.2.2 贪心算法

选择 a_m 为 S_{ij} 中最早结束的活动

$$f_m=\min\{f_k|a_k\in S_{ij}\}\tag{16.2}$$

† 就是 S_{ij} 中的第一个活动

定理 16.1 a_m 在 $S_k=\{a_i|s_i\geqslant f_k\}$ 的某个最大相容活动子集中.

提示 将某个最大相容活动子集中最早结束的活动用 a_m 替代.

算法 16.1 (活动选择问题, 贪心算法) 反复选择最早结束的活动, 保留与此活动兼容的活动, 重复此过程.

† 贪心算法通常都是自顶向下设计: 做出一个选择, 然后求解剩下的子问题.

16.2.3 递归贪心算法

```python
def recursive_activity_selector(s: list, f: list, i, j):
    # f has been sorted by increasing order.
    m = i + 1
    while m <= j and s[m] < f[i]:
        m += 1
    if m <= j:
        return [m] + recursive_activity_selector(s, f, m, j)
    else:
        return []


# Initial call
recursive_activity_selector(s, f, 0, n)
```

16.2.4 迭代贪心算法

```python
def iterative_activity_selector(s, f):
    n = len(s)
    A = [1]      # earliest finished activity
    k = 1
    for m in range(2, n + 1):
        if s[m] >= f[k]:     # k is the last activity in A.
```

```
            A += [m]
            k = m
    return A
```

时间复杂度: $O(n)$

16.3 Huffman 编码

16.3.1 代价

$$B(T) = \sum_{c \in C} c.freq \cdot d_T(c) = \sum_{e \in \text{Internal nodes}} e.freq \tag{16.3}$$

加权外部路径长度最小的二叉树

$$E = \sum_{x \in Leaves} depth(x) \times w(x) \tag{16.4}$$

† 不唯一

16.3.2 Further: k 进制 Huffman 编码

如果是 k 进制编码, 是否可以直接构造一个 k 叉树?

† 以三进制为例, 可能无法构成一个完整的三叉树

▷ 不是最优的! 因为结点会优先成为深度更大的叶子.

† 当 $k-1|n-1$ 时, 仍然可以用原来的算法

▷ 总共 n 棵树 $\Longrightarrow$ 1 棵树, 减少 $n-1$ 棵; 但 $k \Longrightarrow$ 1 每次减少 $k-1$ 棵树

▷ 不满足上述条件时, 可以通过补全一个或多个权重为 0 的叶子结点来实现

L24 + Huffman 编码

第 17 章　平摊分析

17.1　聚合分析

17.1.1　栈操作

n 个 `Pop()`, `Push()`, `MultiPop()` 序列构成的操作总代价为 $O(n)$, 因为对于初始为空栈的情况, 最多 `Pop()` n 次 $\Longrightarrow$ 平摊代价 $O(1)$.

17.1.2　二进制计数器递增

遇到 1 改成 0, 直到遇到 0 改成 1.

第 0 位, 每次都改变;

第 1 位, 每 2^1 次变一次;

第 k 位, 每 2^k 次变一次

n 次操作总代价

$$n+\left\lfloor\frac{n}{2}\right\rfloor+\left\lfloor\frac{n}{2^2}\right\rfloor+\cdots+\left\lfloor\frac{n}{2^k}\right\rfloor+\cdots\leqslant\sum_{i=0}^{\lfloor\log n\rfloor}\left\lfloor\frac{n}{2^i}\right\rfloor\leqslant\sum_{i=0}^{\lfloor\log n\rfloor}\frac{n}{2^i}\leqslant 2n \tag{17.1}$$

17.2　核算法 (会计法)

† 提前支付

17.2.1　栈操作

将所有代价都赋予 `Push()` 操作, 每次代价为 2, 一次用于自身 `Push()` 操作, 另一次留给 `Pop()` 操作.

17.2.2　二进制计数器

每次给 2 块钱, 一次操作消耗 1 块钱, 各位中 1 的个数表示剩余钱数, 因此 $O(2n)$ 代价足够.

17.3　势能法

平摊代价

$$\hat{c}_i=c_i+\Phi(D_i)-\Phi(D_{i-1}) \tag{17.2}$$

总平摊代价 (总代价)

$$\sum_{i=1}^{n}\hat{c}_i=\sum_{i=1}^{n}c_i+\Phi(D_n)-\Phi(D_0) \tag{17.3}$$

若存在某种势函数的定义, 使得 $\Phi(D_n)\geqslant\Phi(D_0)$, 则

$$\sum_{i=1}^{n}\hat{c}_i\geqslant\sum_{i=1}^{n}c_i, \tag{17.4}$$

即 $\sum_{i=1}^{n}\hat{c}_i$ 给出了总代价 $\sum_{i=1}^{n}c_i$ 的一个上界, 用于平摊分析估计代价.

17.3.1 栈操作

定义 $\Phi(D_i)$ 为栈 D_i 中元素个数, 则各操作的势函数变化、平摊代价如下:

1 `Push()`

$$\Phi(D_i)=\Phi(D_{i-1})+1 \tag{17.5}$$

$$\hat{c}_i=c_i+\Phi(D_i)-\Phi(D_{i-1})=1+1=2 \tag{17.6}$$

2 `Pop()`

$$\Phi(D_i)=\Phi(D_{i-1})-1 \tag{17.7}$$

$$\hat{c}_i=c_i+\Phi(D_i)-\Phi(D_{i-1})=1-1=0 \tag{17.8}$$

3 `MultiPop()`

$$\Phi(D_i)=\Phi(D_{i-1})-\min(k,s) \tag{17.9}$$

$$\hat{c}_i=c_i+\Phi(D_i)-\Phi(D_{i-1})=\min(k,s)-\min(k,s)=0 \tag{17.10}$$

† $\min(k,s)=\Phi(D_{i-1})$

† 只有 `Push()` 操作有平摊代价.

思考 对于栈操作, 势函数是否可以有别的定义方法?

17.3.2 二进制计数器

定义 $\Phi(D_i)$ 为计数器 D_i 中 1 的个数.

设第 i 次操作之前, 计数器中有 b 个 1; 第 i 次操作时, 将 t 个 1 改为 0, 则

$$c_i\leqslant t+1 \tag{17.11}$$

$$\Phi(D_i)-\Phi(D_{i-1})\leqslant(b-t+1)-b=1-t \tag{17.12}$$

$$\Longrightarrow\ \hat{c}_i=c_i+\Phi(D_i)-\Phi(D_{i-1})\leqslant 2 \tag{17.13}$$

17.4 动态表

每次扩张翻倍

装载因子 $\alpha(T)$.

满时扩张, $\alpha(T) < \frac{1}{4}$ 时收缩.

† $\alpha(T)$ 收缩的阈值不能设置为 $\frac{1}{2}$, 否则在扩张边缘反复横跳

下面对此问题用前述 3 种方法分别讨论.

17.4.1 聚合法

n 次 `Insert()` 操作, 最多 $\log n$ 次扩张,

$$c_i = \begin{cases} i, & i-1 = 2^k, \\ 1, & \text{else} \end{cases} \tag{17.14}$$

$$\Longrightarrow \sum_{i=1}^{n} c_i \leqslant n + \sum_{j=0}^{\lfloor \log n \rfloor} 2^j < n + 2n = 3n \tag{17.15}$$

17.4.2 会计法

`Insert()` 操作的平摊代价为 3:

(1) 自身插入操作

(2) 移动自身

(3) 造成其它数据第二次移动

长度为 m 时存储可用代价为 0, 当长度为 $2m$ 时, 新插入的 m 项携带了自身移动所需的代价, 同时存储了造成前 m 个数据移动所需的代价.

17.4.3 势能法

$$\Phi(T) = 2 \times T.num - T.size \geqslant 0 \tag{17.16}$$

† 表满时为 $T.size$

† 扩张后为 0

每次积累 2, 另外还有一次用于自身操作, 平摊代价仍为 3.

按照第 i 次操作是否触发扩张来计算 $\hat{c}_i$.

第 18 章　分治法

18.1　最大子数组问题 (Max Subarray, MS)

18.1.1　分治法求解

$A[low..high] = A[low..mid] + A[mid..high]$ 的 MS 为 $A[i..j]$, 则必为 3 种情况之一:

(1) $[i, j] \subset [low, mid]$: 子问题

(2) $[i, j] \subset [mid, high]$: 子问题

(3) $i \leqslant mid \leqslant j$: 分别向左、向右扫描, 获得最大收益

18.1.2　线性扫描算法

从左往右扫描直至第一个 > 0 的开始计算 `sum`, 记录 `max` 值的位置及和, 直至 `sum < 0` 时开始重新记录扫描, 得到若干个局部最大值, 最后比较这些最大值即可.

```
def linear_scan_max_subarray(A: list):
  """Solve maximum subarray problem using linear scan

  Args:
    A (list): array

  Returns:
    m, low, high: max sum, lower bound, upper bound
  """
  m = -INFTY
  low, high = -1, -1
  # temp variables
  m_r = 0
  low_r = 0
  for i in range(0, len(A)):
    m_r += A[i]
    if m_r > m:   # find a new maximum subarray
      low, high = low_r, i
      m = m_r
    if m_r < 0:   # previous accumulation is useless, abandon and restart
      m_r = 0
```

```
22          low_r = i + 1
23      return m, low, high
```

18.2 矩阵乘法的 Strasssen 算法

第四部分

高级数据结构

第 12 章　二分检索树

12.1　什么是二叉检索树

12.1.1　中序遍历

得到有序序列

12.1.2　二分检索

(1) 折半查找

(2) Fibonacci 查找

(3) …

12.2　查询二叉检索树

12.2.1　指定关键字

12.2.2　最小关键字和最大关键字

12.2.3　前驱和后继

1 后继 (中序遍历)

(1) x 有右子树: 右子树的最小值为后继;

(2) x 没有右子树: 找最近的父节点, 使得 x 在 y 的左子树中

思考　先序遍历/后序遍历的前驱/后继如何找?

2 先序遍历寻找后继的顺序

(1) 左孩子

(2) 右孩子

(3) 最近的有右孩子的父亲的右孩子

12.3　插入和删除

12.3.1　* 删除

(1) 左子树为 NIL: 用右子树替换

(2) 右子树为 NIL:

(3) 左右子树均非空, 后继为右儿子: 用右儿子替换

(4) 找后继: $z \leftarrow y$, $y \leftarrow x$

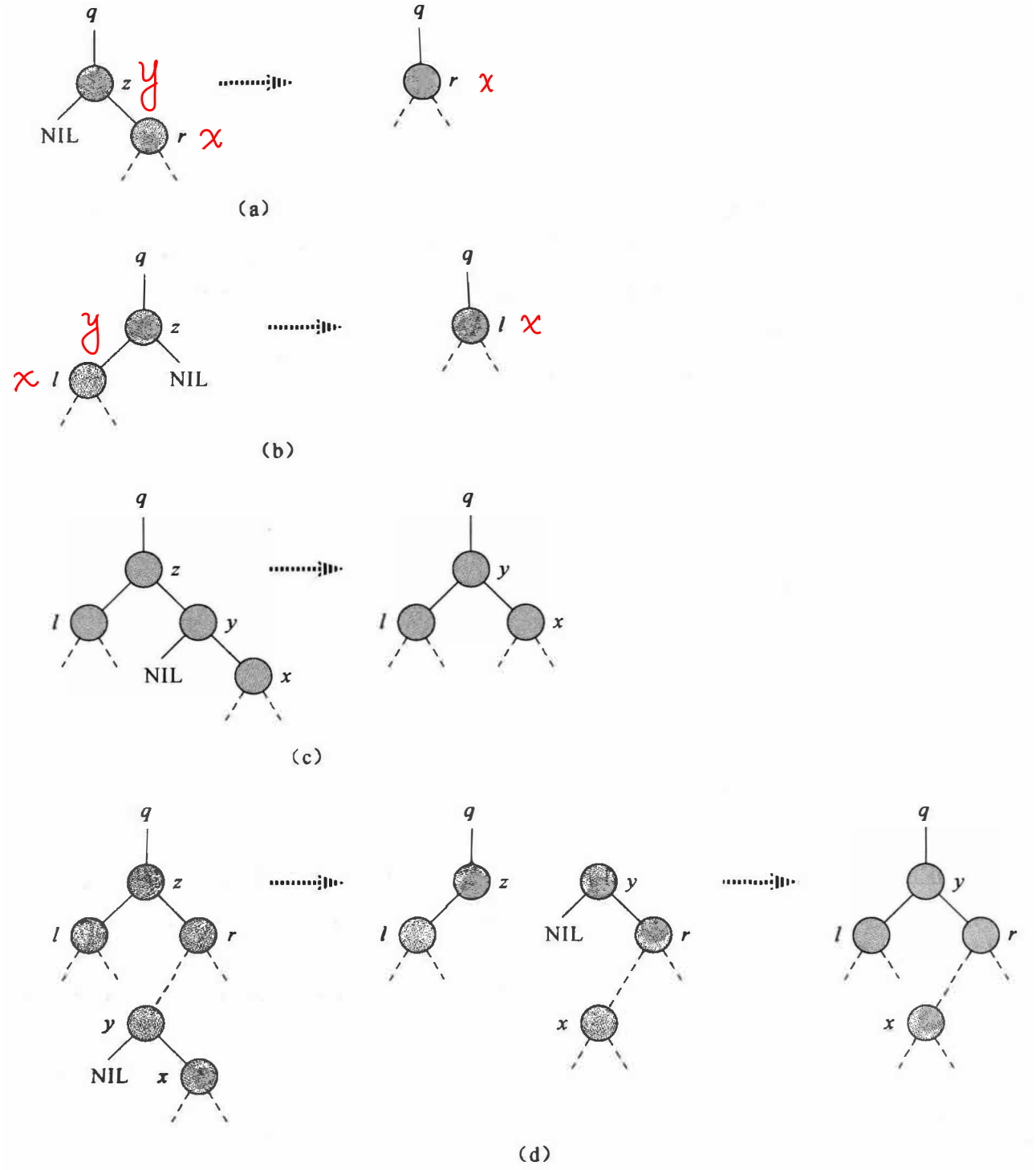

Figure 12.1 (图 12.4) 二叉检索树的删除

第 13 章　红黑树

定义 13.1 (红黑树)　平衡二叉检索树

13.1　红黑树的性质

满足下述红黑性质的二叉检索树:
(1) 每个结点为红/黑
(2) 根结点为黑
(3) 叶子结点 (NIL) 为黑
(4) 红结点的子结点为黑
† 不可能有两个红结点为父子关系
(5) 每个结点, 到任何一个叶子结点的路径上的黑结点的个数相同
† 黑高度相同

13.2　旋转

13.2.1　左旋 & 右旋

(1) LL
(2) RR
(3) LR
(4) RL

思考　空的平衡二叉树至少需要插入多少个结点, 才能使上述旋转操作至少各出现一次?

13.3　插入

13.3.1　看作二分检索树插入结点

需要更改结点颜色
总是插入红色结点, 然后自下而上调整颜色

13.4　删除

二叉树操作 + 颜色调整

第 14 章　数据结构的扩张

14.1　动态顺序统计

14.1.1　扩张

红黑树结点增加域 `size` 记录包含其本身的内部结点数, 则

$$x.size = x.left.size + x.right.size + 1 \tag{14.1}$$

14.1.2　基本操作

1 选择　选择以 x 为根的树中第 i 小的结点

```
OS-Select(x, i)
```

递归实现

2 排序　返回以 T 为根的树中 x 的排序

```
OS-Rank(T, x)
```

循环实现, 也可以有递归版本

14.1.3　维护

左右旋操作 `size` 域发生变化, 只与其左右结点的 `size` 值有关.

14.2　数据结构的扩张

14.2.1　步骤

(1) 选择基础数据结构

(2) 确定附加信息

(3) 检验原基本操作下附加信息的维护没有增加时间复杂度

† 例如, 二叉树下, 通常要求附加信息只与其左右结点的附加信息有关

(4) 定义新的基本操作

† 这是扩张数据结构的真正目的

14.3　区间树

以 `x.low` 左端点作为关键字, 增加域 `x.max` 为以 x 为根节点的子树的最大右端点

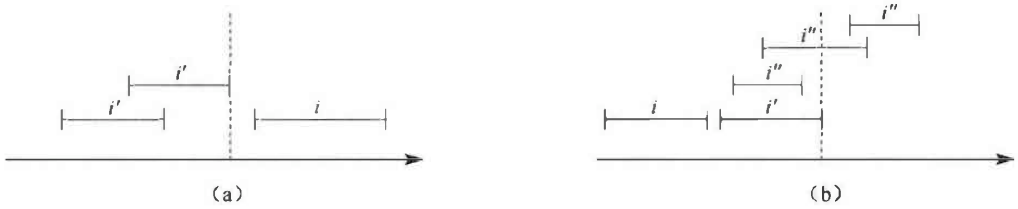

Figure 14.1 区间树 `Interval-Search()` 算法的正确性

14.3.1 查找

1 证明算法正确性

第 19 章　Fibonacci 堆

† 可合并堆
  ▷ 二叉堆不可合并 (合并代价大)

19.1　结构

19.2　基本操作

19.2.1　Make

19.2.2　Insert

19.2.3　Get-Min

19.2.4　Merge

19.2.5　Extract-Min

19.2.6　Decrease-Key

19.2.7　Delete

第 21 章　用于不相交集合的数据结构

21.1　不相交集合的操作

沿父节点路径找到根节点, 比较两个结点的根节点确定其是否处于同一个集合中.

21.2　不相交集合的链表表示

21.2.1　合并

将较短的表合并到较长的表中, 需要更新被合并表的每一个结点的 head 域.

21.3　不相交集合森林

21.3.1　静态链表

21.3.2　按秩合并

21.3.3　路径压缩

Listing 21.1　并查集

```
def find_set(x: Set):
    if x != x.p:
        x.p = find_set(x.p)
    return x.p
```

第五部分

图算法

第 22 章　基本的图算法

22.1　图的表示

22.1.1　数据结构

1 邻接表　存储的是结点的 index!

2 邻接矩阵

22.2　广度优先搜索, Breadth-first Search, BFS

注意　此处为搜索, 只能搜索到一个连通分量

1 搜索

```
1 Search(G, u)  # Search a connected component of u in G
```

2 遍历

```
1 Traverse(G)   # Visit every vertex of G
```

每个顶点有 3 种颜色:

(1) WHITE: 未访问

(2) GRAY: 已访问, 但邻接点未访问完

(3) BLACK: 已访问, 且邻接点已访问

维护一个队列, 按入队顺序进行访问

树的层次遍历也可以按此算法

22.3　深度优先搜索, Deep-first Search, DFS

注意　区分搜索和遍历! 此处为遍历

若干个连通分量 (?)

关键结点/关键边: 删去后图不连通

22.3.1　时间戳

发现时间 `u.d`, 完成时间 `u.f`

后续用于搜索关键结点/关键边时很有用!

思考　怎么利用? 怎么找?

22.3.2 4 种形式的边 (有向图)

1 树边 构成一颗深度优先树

2 回边 (后向边) 子孙 → 祖先, 是从祖先那来的, 这会构成一个环

3 前向边 祖先 → 子孙, 已经从子孙的子孙访问过了

4 交叉边 (横向边) 子孙 → 子孙, 子孙之间的边, 已经从兄弟的结点访问过了

有向图的最小生成树 (?) $\Longrightarrow$ 都不一定存在

22.3.3 无向图的边

1 树边

2 非树边

22.4 Topo 排序

(1) DFS

(2) 按完成时间逆向排序

(3) 构成 TOPO 排序链表

22.5 强连通分量

双向连通子图

(1) 深度优先遍历

(2) 转置图上按完成时间逆向遍历, 无法继续发现新结点时, 找到一个强连通分量

第 23 章　最小生成树

† 连通无向图

23.1　最小生成树的形成

1 轻边

23.2　贪心算法

23.2.1　Kruskal

将边排序后一一加入, 每次选择不构成环路的最小边

需要用到分离集合数据结构

适用于稀疏图

算法 23.1 (Kruskal)　使用分离集合数据结构

(1) 初始化: 每个结点为一个集合

(2) 生成: 每次选择横跨不同集合的轻边, 加入最小生成树, 同时合并结点集合

(3) 终止: 所有结点合并为一个集合

23.2.2　Prim

将结点一一并入, 每次选择与已并入结点集合相连最小的边

算法 23.2 (Prim)　维护一个优先队列, 将连接 v 与已在集合 S 中的结点的最小边作为 key.

(1) 初始化: 随机选择一个结点 v_0, 其余结点入队, 关键字为 $key = w(u, v_0)$;

(2) 生成: 提取最小值结点 v_i, 并入集合 S 中, 更新关键字: 可能为原来的值或 $w(u, v_i)$;

(3) 终止: 队列为空

23.2.3　Sollin

初始: 每个顶点视为一棵树

每次每棵树同时选择连向其它树的最小邻边, 合并树

† 适合并行

▷ 单处理器计算时间复杂度高

† 类似于 Kruskal 算法同时做多个选择

第 24 章　单源最短路径

24.1　Bellman-Ford 算法

24.1.1　松弛操作

距离矢量算法, 泛洪, 每个结点向其相邻的结点传递结点可达信息.

24.1.2　计算最短路径

泛洪 $|V|-1$ 次, 每次对所有边进行松弛操作.

24.1.3　优化

可以通过设置 `flag`, 记录本轮泛洪过程中是否有最短路径长度发生改变, 若没有, 则循环终止.

24.2　有向无环图 (DAG) 图中的单源最短路径

按照 Topo 排序结果进行泛洪, 只需泛洪一次

24.3　Dijkstra 算法

要求所有边权重都为非负值!

† 最小生成树的 Prim 算法: 每次加入距离当前集合最近的结点;

† 单源最短路径的 Dijkstra 算法: 每次加入距离源点 s 最近的结点

▷ 每次对新加入的结点的出边进行松弛操作

第 25 章　所有结点对的最短路径问题

25.1　最短路径与矩阵乘法

若用 Bellman-Ford 算法, 利用矩阵表示优化时间复杂度

† 矩阵形式的思想!!!

25.2　Floyd-Warshall 算法

$d_{ij}^{(k)}$ 定义为 $i \to j$ 途径顶点序号最大为 k 的最短路径长度

$$d_{ij}^{(k)} = \begin{cases} w_{ij}, & k = 0, \\ \min\left\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right\} \overset{\dagger}{=} \min\left\{d_{ij}^{(k-1)}, d_{ik}^{(k)} + d_{kj}^{(k)}\right\}, & k \geqslant 1 \end{cases} \tag{25.1}$$

† 处说明, 可以在原地进行迭代, 只需要维护一个矩阵即可.

† 同时计算前驱矩阵

† $D^{(n)}$ 给出最终结果

25.3　Johnson 算法

如何改进 Dijstra 算法, 使得其适用于负权值?

重新赋予权重

定理 25.1　$\forall h : V \to \mathbb{R}$, 定义如下新的权重即可保证新定义的权重下的最短路径与先前一致

$$\hat{w}(u, v) = w(u, v) + h(u) - h(v) \tag{25.2}$$

算法 25.1 (Johnson)　调整边的权值, 使得 $\hat{w}(u, v) \geqslant 0$, 然后对每一个结点调用 Dijkstra 算法, 求各结点对的最短路径.

(1) 加入新的顶点 $G' = G \cup \{s\}$, 调用 Bellman-Ford 算法计算其到各个顶点的最短路径长度 $\delta(s, u) := h(u)$

(2) 定义新的权值 $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$, 在新的权值下, 对每个顶点调用 Dijkstra 算法计算最短路径长度 $\hat{d}(u, v)$

(3) 回溯: 原图的最短路径长度为 $d(u, v) = \hat{d}(u, v) + h(v) - h(u)$

第六部分

算法问题

第 30 章　分治法

30.1　大整数的乘法

$X = \overline{ab}, Y = \overline{cd}$, 相乘分 2 段进行
4 次乘法 $\implies$ 没有改进
减少乘法次数: $XY = ac \cdot ((a-b)(c-d) - ac - bd) \cdot bd$

30.2　矩阵乘法的 Strassen 算法

30.3　多项式与快速 Fourier 变换

多项式相乘: $\Theta(n^2)$
若用 FFT 实现, $\Theta(n \lg n)$ $\implies$ 代入 $x = 2, 10$, 实现大整数的乘法

30.3.1　多项式的表示

多项式的系数表示 $\iff$ n 个点的值
若能建立对应, 则多项式乘法可以将对应点的值相乘, 然后对应求出系数.
1 从系数到点值　　秦九韶算法, $\Theta(n)$

30.3.2　DFT & FFT

1 离散 Fourier 变换, DFT

$$A(x) = \sum_{j=0}^{n-1} a_j x^j \leftrightarrow \boldsymbol{a} = (a_0, a_1, \cdot, a_{n-1}) \tag{30.1}$$

的 DFT

$$y_k = A(\omega_n^k) = \sum_{j=0}^{n-1} a_j \omega_n^{kj} \leftrightarrow \boldsymbol{y} = (y_0, y_1, \cdots, y_{n-1}) \tag{30.2}$$

2 快速 Fourier 变换, FFT

30.3.3　高效 FFT 实现

1 位逆序变换与蝴蝶操作

† 例如, 长度为 8 的序列, $4 = 100$, 逆序 $\implies$ 001 $\implies$ 第 1 位.

第 32 章　串匹配算法

文本: $T[1\cdots n]$
模式: $P[1\cdots m]$, $m \leqslant n$ (通常 $n \gg m$)
串匹配原理: 滑动窗口模型

32.1　Brute-Force 算法 (朴素字符串匹配)

32.2　Rabin-Karp 算法

对长为 m 的串, 定义 hash 函数, 若函数值相同, 再进行窗口内部比较, 否则直接跳过.
对无向图深度优先遍历后, 如何在 $\Theta(1)$ 判断顶点 u, v 是否在同一个连通片上?
邻接表: 顶点数组增加域, 表示所在连通片 (第几轮搜索被搜索到)

32.3　FSM 字符串匹配

32.4　Knuth-Morris-Pratt 算法 (KMP)

考试范围

时间复杂度分析: 与底层数据结构、数据分布有关

渐近记号

排序: 只考书上有的 (增量排序不考)

矩阵乘法公式不用背 (?)

* 递归方程求解 3 种方法

排序: * 堆排序, 堆没有插入删除的基本操作, 二叉堆看作优先队列 (有插入删除操作), * 快速排序 (Partition), 线性时间排序 (计数、基数、桶排序)

选择问题: 同时找最大最小值 (最少比较次数的证明方法), 顺序统计

高级数据结构: 二分检索树 (遍历), 红黑树插入删除, 数据结构扩张, 分离集合数据结构 (路径压缩, 按秩合并)

动态规划, 贪心算法, 平摊分析: * 例子 (0-1 背包, 普通背包), 最优性原理, 平摊分析

FFT: 时间复杂度, 做 FFT 变换

图算法: BFS, DFS, 遍历 & 搜索, 最小生成树, Dijkstra 不适用权值 < 0 的边 $\Longrightarrow$ Johnson

串匹配: BM 算法, Quick-Search 算法不考, FSM 状态机的状态表, KMP 算法 Pi 函数, RK 算法

各种需要计算的表项 (动态规划法...)

TODO

32.4.1 红黑树的插入删除

32.4.2 主方法

32.4.3 课本 Bookmarks