一　算法效率分析

1. 基本操作：算法最内层循环中最费时的操作

2. 渐近符号

(1) 例：$f(n) \in O(g(n))$

(2) O：$f(n) \le cg(n)$　　$\leqslant$

(3) Ω：$f(n) \ge cg(n)$　　$\geqslant$

(4) Θ：$c_1 g(n) \le f(n) \le c_2 g(n)$　　$=$

(5) o：$f(n) < cg(n)$　　$<$

3. 用极限证明增长次数（洛必达）

(1) $$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & \mathrm{O} \\ c > 0 \Rightarrow & \Theta \\ \infty & \Omega \end{cases}$$

(2) 斯特林公式：$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

4. 渐近效率类型名称

(1) 常量、对数、线性、线性对数、平方、立方、指数、阶乘

5. 非递归算法复杂度分析：解求和表达式

6. 递归算法复杂度分析：解递推式

(1) 反向替换法：减小规模后反复代入。P57 例

(2) 平滑规则定理：先仅在 $n=2^k$ 的情况下对递推式求解，再用对数转换回原来变量 n 的函数。P59 例

(3) 对数答案精确解：向下取整

(4) $a^{\log_b c} = c^{\log_b a}$

7. 通用分治递推式复杂度分析：主定理

(1) $T(n) = aT(\frac{n}{b}) + f(n), \quad f(n) \in \Theta(n^d)$

(2) $$T(n) \in \begin{cases} \Theta(f(n)) & , a < b^d \\ \Theta(f(n) \cdot \log n), & a = b^d \\ \Theta(n^{\log_b a}) & , a > b^d \end{cases}$$

(3) 更换渐近符号也成立

二　蛮力法

1. 选择排序

   (1) 每趟排序选择出未排序区中最小的，放到已排序区末尾

   (2) n^2

2. 冒泡排序

   (1) 每趟排序将最大的元素通过连续 swap 沉底

   (2) n^2

3. 顺序查找

4. 字符串匹配（蛮力法）

   (1) 当字符失配，就将模式串整体右移一格，重新开始匹配

   (2) 最坏 nm，随机数据下平均 n

5. 最近点对问题（蛮力法）

6. 凸包问题（蛮力法）

   (1) 对于集合中的两个点，当且仅当集合中所有其他点都位于该两点连线的同一侧时，该线段是凸包的一部分。

   (2) 对 n^2 个点对进行检验，每次 n。共 n^3 。

7. 旅行商问题（蛮力法）

   (1) 穷举查找所有可能的排列

   (2) n!

8. 背包问题（蛮力法）

   (1) 穷举查找所有可能的组合

   (2) 2^n

9. 分配问题（蛮力法）

(1) 穷举查找所有可能的解（人员分配的排列）

(2) n!

(3) 匈牙利算法求解二分图最大匹配，$ve = n^3$

10. 深度优先查找

(1) DFS 森林：树向边-实线、回边-虚线

(2) 邻接矩阵表示图：v^2，邻接链表表示：v+e

11. 广度优先查找

(1) BFS 森林：树向边-实线，交叉边-虚线（同层按字母顺序访问）

(2) 邻接矩阵表示图：v^2，邻接链表表示：v+e

三　减治法

1. 分类

   (1) 减去常量（减一法）

   (2) 减去常量因子（减半法）

   (3) 减去的规模可变

2. 插入排序

   (1) 每趟排序将无序部分中的第一个元素插入有序部分中

   (2) 最好已有序 n，最坏倒序 n^2，平均 n^2。

3. 拓扑排序

   (1) DFS 解法：执行一次 DFS 遍历，并记住顶点变成死端（退出遍历栈）的顺序。将该次序反过来就是拓扑排序的一个解。遍历时遇到回边则说明无解。

   (2) 源删除算法（基于 BFS）

4. 生成排列

   (1) 减一法自底向上生成排列：将 n 插入 n-1 个元素的所有排列的所有位置中。先从右到左，每次变换方向，可以满足最小变化要求。

   (2) Johnson-Trotter 算法：初始化箭头为全部朝左。移动元素定义为箭头指向相邻较小元素的元素。每次选取最大的移动元素 k，移动一格，调转所有大于 k 的元素的箭头方向。

   (3) 以字典序生成排列：找最长递减后缀，把后缀前一个元素和后缀中大于它的最小元素交换，将新后缀颠倒。

5. 生成子集

(1) 自底向上生成子集：将新元素添加到上一次的所有子集中

(2) 二进制位串和子集的对应关系

6. 折半查找

(1) R=m-1 或 L=m+1

(2) 最坏和平均都是 logn

7. 假币问题

(1) 分成两堆，改进：分成三堆

8. 俄式乘法

(1) n 不断÷2（向下取整），m 不断×2，当 n 为奇数时，把 m 往右写，最后把右边的数加起来就是答案

9. 约瑟夫斯问题

(1) J(2k)=2J(k)-1, J(2k+1)=2J(k)+1

(2) J(n)=n 向左循环移位

10. 选择问题

(1) 寻找第 k 个最小元素

(2) 快速选择算法，递归左右，平均 n，最坏 n^2

11. 插值查找

(1) 假设数组线性递增

(2) 通过直线方程计算期望位置

(3) 平均 loglogn+1 次，最坏 n

12. 二叉查找树查找

(1) 所有左子树的元素都小于根节点，所有右子树的元素都大于根节点

(2) 最坏 n，平均 logn

四　分治法

1. 合并排序

   (1) 递归左右，线性合并

   (2) 最坏 nlogn-n+1，稳定

2. 快速排序

   (1) 划分，递归排序左右，不稳定

   (2) Hoare 划分：i, j 从两端向中间扫描，遇到不合法的对则 swap，继续扫描直到相遇，然后 j 与枢轴交换。

   (3) 最好 nlogn，最坏 n^2（当数组本身有序时），平均 nlogn

   (4) 改进：随机选择枢轴、三平均划分法、当子数组足够小时改用插入排序、三路划分

3. 二叉树遍历

   (1) 求高度。边的数量（课本定义）：空树 return -1:。点的数量：空树 return 0。

4. 大整数乘法

   (1) 朴素：n^2 次位乘

   $$c = a \times b = c_2 10^n + c_1 10^{\frac{n}{2}} + c_0$$

   (2) $c_2 = a_1 \times b_1$，$c_0 = a_0 \times b_0$

   $$c_1 = (a_1 + a_0) \times (b_1 + b_0) - (c_2 + c_0)$$

   (3) n 位数乘法需要对 n/2 位数乘法做 3 次乘法运算，$\Theta(n^{\log_2 3})$

5. Strassen 矩阵乘法

6. 最近点对问题（分治法）P151

7. 凸包问题（分治法）

五　变治法

1. 分类

   (1) 实例化简：变为更简单的实例

   (2) 改变表现：变为同样实例的不同表现

   (3) 问题化简：变为另一个问题的实例

2. 检验数组元素唯一性：朴素 n^2，预排序 nlogn

3. 模式（众数）计算：朴素 n^2，预排序 nlogn

4. 高斯消去法

5. AVL 树

   (1) 最小失衡子树是左右子树高度差为 2 的子树，先找最小失衡子树的根结点，确定新插入节点与根节点的相对位置

   (2) LL 型：右单旋转

   (3) RR 型：左单旋转

   (4) LR 型：先左旋再右旋

   (5) RL 型：先右旋再左旋

6. 2-3 树

   (1) 插入时正常插入到叶子对应位置

   (2) 若插入后该节点满 3 个键，则分裂（向上合并或产生新根）

7. 自底向上堆构造

   (1) 特点：完全二叉树、父母优势

   (2) 数组存储：左右孩子：2i 和 2i+1，父节点：floor(i/2)

   (3) 对已有的数组形成一颗完全二叉树

(4) 从最后一个非叶节点开始向上逐个检查，检查其是否具有父母优势

(5) 若不满足优势，与其孩子的较大者对换，继续**向下检查**

(6) 最多需要 $2(n-\log_2(n+1)) \in O(n)$次比较

8. 自顶向下堆构造

(1) 从零开始构造新堆

(2) 将元素插入到最后位置，并**向上调整**

(3) 插入操作复杂度为堆的高度 logn，构造总复杂度为 nlogn

9. 堆删除

(1) 将要删除的元素与最后一个叶子交换

(2) 删除最后一个叶子

(3) 对根沿着树进行**向下检查（堆化）**

(4) logn

10. 堆排序

(1) 先用原数组自底向上构造出堆

(2) 逐个删除最大键

(3) 复杂度 n+nlogn=nlogn

11. 霍纳法则

(1) 两行，第一行写系数

(2) 第二行写前一格结果乘 x 加当前格系数

(3) res=P[n],

(4) for i=n-1 downto 0 :　　res=x*res + P[i]

(5) 表格第二列可以作为 p(x)除以 $x-x_0$ 的结果（幂次-1）

12. 从左至右二进制幂

```
product ← a
for i ← I − 1 downto 0 do
    product ← product * product
    if b_i = 1 product ← product * a
return product
```

例 2　用从左至右二进制幂算法计算 a^{13}，这里 $n = 13 = 1101_2$。因此，我们有

n 的二进制位	1	1	0	1
累乘器	a	$a^2 \times a = a^3$	$(a^3)^2 = a^6$	$(a^6)^2 \times a = a^{13}$

13. 从右至左二进制幂

```
term ← term * term
if b_i = 1 product ← product * term
return product
```

例 3　用从右至左二进制幂算法计算 a^{13}，这里 $n = 13 = 1101_2$。因此，我们有下面这个从右到左填写的表格。

1	1	0	1	n 的二进制位
a^8	a^4	a^2	a	项 a^{2^i}
$a^5 \times a^8 = a^{13}$	$a \times a^4 = a^5$	a	1	累乘器

14. 计算图中路径数量

(1) 邻接 01 矩阵的 k 次方就是距离为 k 的路径数量

六　动态规划

1. 币值最大化问题

   (1) 选择的硬币互不相邻

   (2) 状态定义：F[n]为只能选前 n 枚硬币时的最大答案

   (3) F[n] = max (c[n]+F[n-2], F[n-1]) , n > 1

   (4) F[0] = 0 ,　F[1] = c[1]

2. 找零问题

   (1) 状态定义：F[n]为凑齐金额 n 所需最少钞票张数，有 m 种面值

   (2) F[n] = min { F[n - d[i]] } +1 , n > 0　（min 中条件为 n>=d[i]）

   (3) F[0] = 0

   (4) DP 复杂度 O(nm)，贪心复杂度 O(n+m)

3. 硬币收集问题

   (1) 状态定义：F[i][j]为截止到第 i 行第 j 格能收集的最大硬币数

   (2) F[i][j] = max(F[i-1][j], F[i][j-1]) + c[i][j],

   (3) F[0][j] = F[i][0] = 0

4. 背包问题（动态规划法）

   (1) 状态定义：F[i][j]为背包承重量为 j、只考虑前 i 个物品时，所能获取的最大价值

   (2) $F[i][j]=\begin{cases}\max(\mathrm{F[i-1][j]},\ \mathrm{F[i-1][j-w[i]]+v[i]}), & j-w[i]>=0\\ \mathrm{F[i-1][j]}, & j-w[i]<0\end{cases}$

5. 记忆化搜索

(1) 该格有记忆则直接返回记忆，无记忆则递归调用并写入记忆

6. 最优二叉查找树

7. Warshall 算法求传递闭包

(1) 框，先列再行，与原有结果比对更新

(2) 伪代码 for kij: r[k][i][j] = r[k-1][i][j] | r[k-1][i][k] & r[k-1][k][j]

(3) n^3，改进：从每个顶点开始用 DFS 遍历更新可达顶点，n^2

8. Floyd 算法计算完全最短路径

(1) 框，列分别与行加，和与列所在和同行对比

(2) 伪代码：for kij: D[i][j] = min(D[i][j], D[i][k] + D[k][j])

七　贪心算法

1. Prim 算法求最小生成树

   (1) 每次选树外点中，距离树中顶点最近的点进行扩展，并更新其邻接点离树的最近距离

   (2) 图由邻接矩阵存储，优先队列由无序数组实现，则 v^2

   (3) 图由邻接链表存储，优先队列由最小堆实现，则 elogv

2. Kruskal 算法求最小生成树

   (1) 按照权重对边进行排序，尝试加入生成树，若产生回路则跳过

   (2) 每次加入最小边，避圈

   (3) 若使用并查算法和高效排序，则 eloge

3. Dijkstra 算法求单起点最短路径问题

   (1) 每次从余下顶点中选择离起点最近的加入闭集，并松弛

   (2) 图由邻接矩阵存储，优先队列由无序数组实现，则 v^2

   (3) 图由邻接链表存储，优先队列由最小堆实现，则 elogv

4. 哈夫曼树和编码 P262

   (1) 变长编码，将较短的代码字分配给更常用的字符，将较长的代码字分配给不常用的字符

   (2) 哈夫曼树的点权为字符出现概率（和）

   (3) 左 0 右 1，小的在左、大的在右

   (4) $$\text{压缩率}=\frac{\text{定长编码位长}-\text{变长编码平均位长}}{\text{定长位长}}\times 100\%$$