

绪论

下界：代数判定树

要是你达到一个界线，你不能越过它，那你就会倒霉；
但是你越过了它，也许你会更倒霉。

就好比假设有四样东西，我们正在寻找其中的一个。不管它在哪里，如果我们一开始就知道这样东西在哪里的话，那么剩下的就不是什么问题了。又或者我们可以首先找到其余的三样东西，那么剩下的显然就是第四个了。

邓俊辉

deng@tsinghua.edu.cn

难度与下界

- ❖ 由前述实例可见，同一问题的不同算法，复杂度可能相差悬殊
- ❖ 在可解的前提下，可否谈论问题的**难度**？如何比较不同问题的难度？
- ❖ 问题P若存在算法，则所有算法中**最低**的复杂度称为P的难度
- ❖ 为什么要确定问题的难度？给定问题P，如何确定其难度？
- ❖ 两个方面着手：设计复杂度更低的算法 + 证明更高的问题难度下界
- ❖ 一旦算法的复杂度达到难度下界，则说明就大O记号的意义而言，算法已经最优
- ❖ 例如，排序问题下界为 $\Omega(n \log n)$ ，而且是**紧的**...

时空性能 + 稳定性

❖ 多种角度估算的时间、空间复杂度

- 最好 / best-case
- 最坏 / worst-case
- 平均 / average-case
- 分摊 / amortized

❖ 其中，对最坏情况的估计最保守、最稳妥

因此，首先应考虑**最坏情况最优**的算法

//worst-case optimal

❖ 排序所需的时间，主要取决于

- 关键码比较次数 / # {key comparison}
- 元素交换次数 / # {data swap}

❖ 就地 (in-place) :

除输入数据本身外，只需 $O(1)$ 附加空间

❖ 稳定 (stability) :

关键码雷同的元素，在排序后相对位置保持

最坏情况最优 + 基于比较

❖ 排序算法，最快能够有多快？

- 语境1：就最坏情况最优而言
- 语境2：就**某一大类**主流算法而言...

❖ **基于比较**的算法 (comparison-based algorithm)

算法执行的进程，取决于一系列的数值（这里即关键码）比对结果

- 比如，`max()` 和 `bubbleSort()`

❖ 任何CBA在最坏情况下，都需 $\Omega(n \log n)$ 时间才能完成排序

判定树

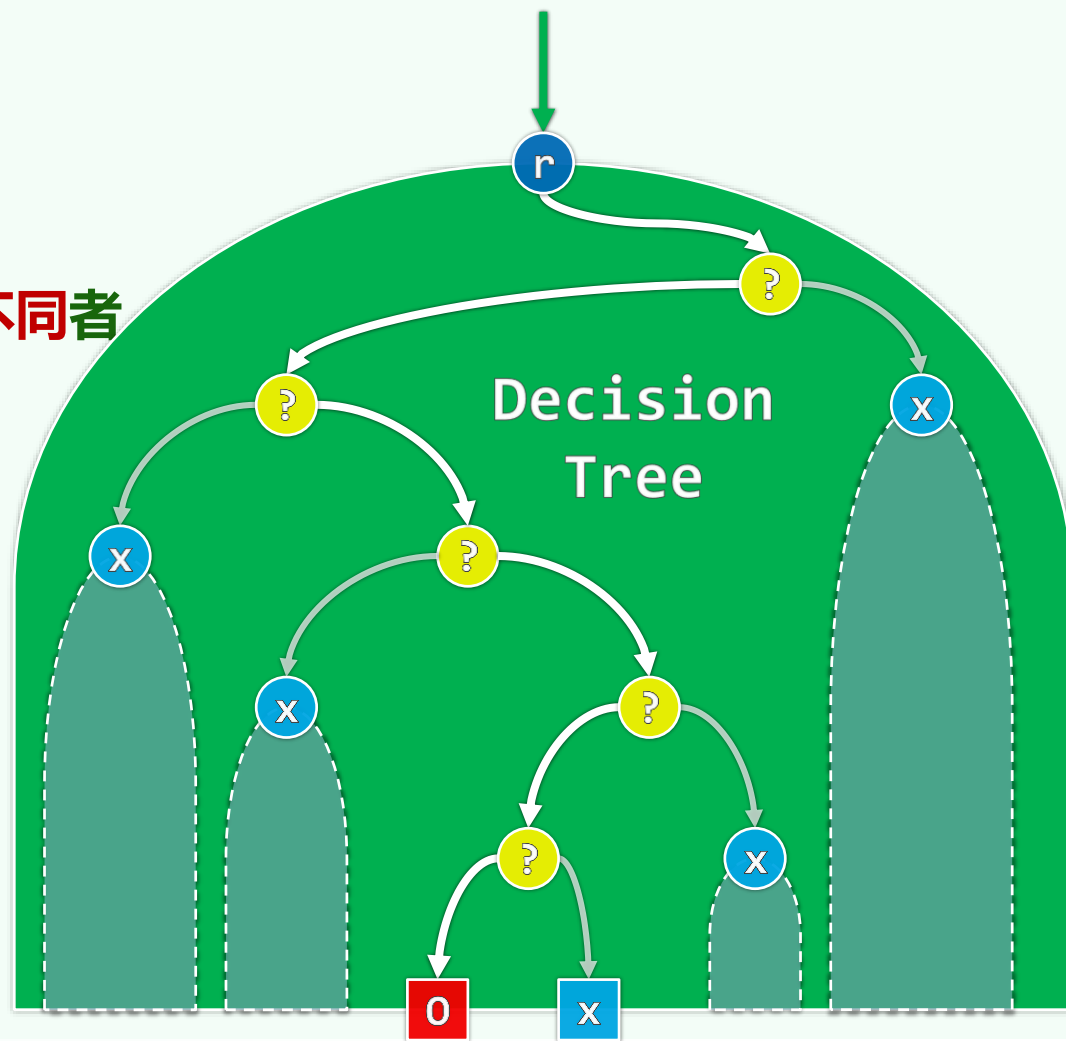
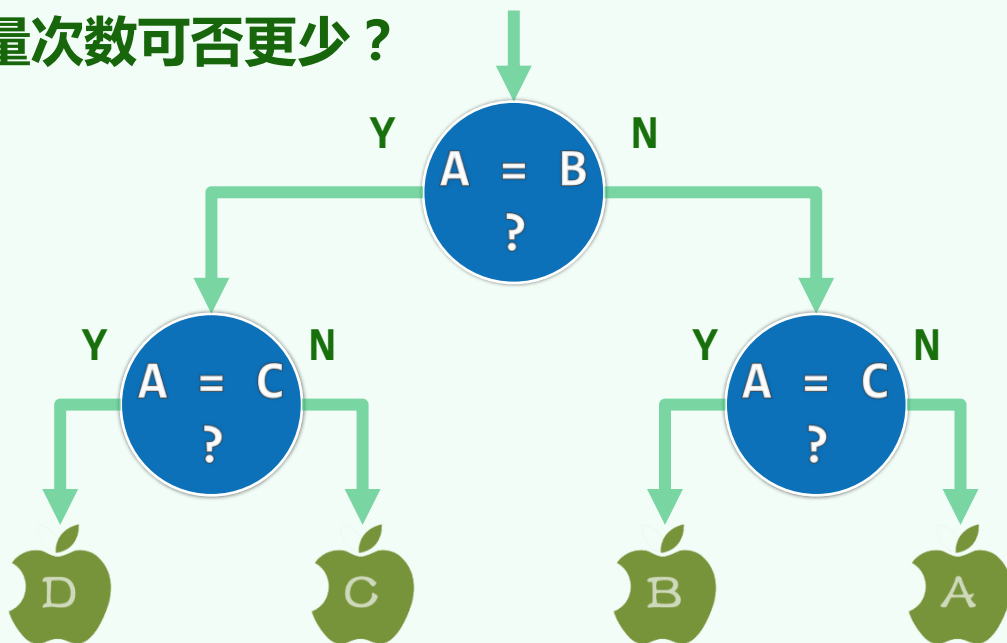
❖ 与CBA算法对应：

每一可能的输出，都对应于至少一个叶节点

每一次运行过程，都对应于起始于根的某条路径

❖ 经2/4次称量，必可从4/16只苹果中找出唯一的重量不同者

称量次数可否更少？



代数判定树

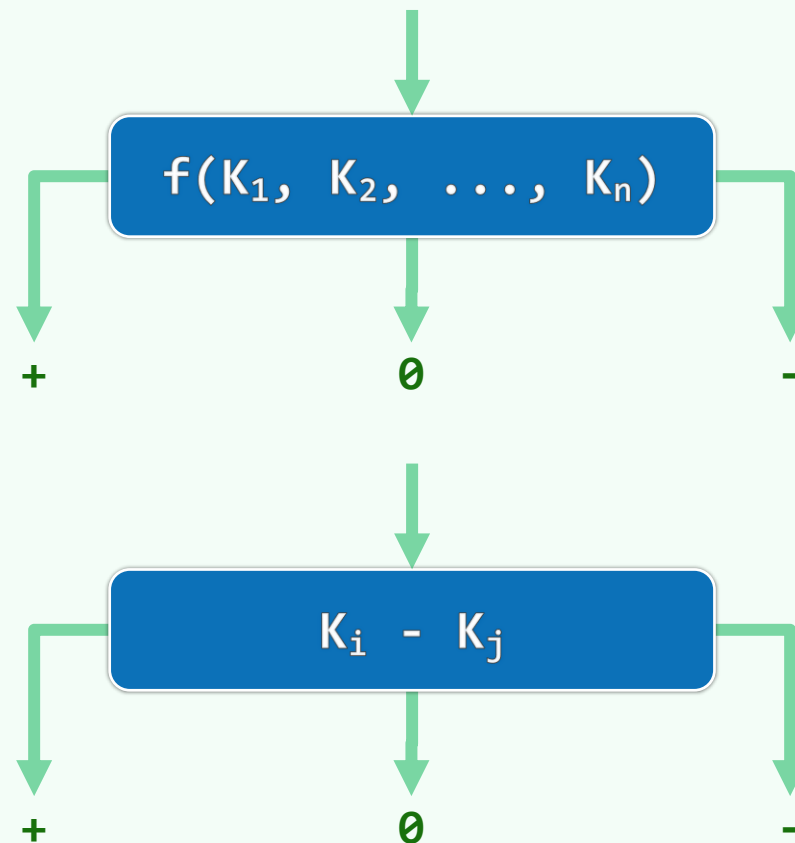
❖ Algebraic Decision Tree

- 针对“比较-判定”式算法的计算模型
- 给定输入的规模，将所有可能的输入所对应的一系列判断表示出来

❖ 代数判定：

- 使用某一常次数代数多项式
将任意一组关键码做为变量，对多项式求值
- 根据结果的符号，确定算法推进方向

❖ Comparison Tree：最简单的ADT，二元一次多项式，形如： $K_i - K_j$



下界： $\Omega(n \log n)$

❖ 比较树是三叉树 (ternary tree)

内部节点至多三个分支 (+| \emptyset |-)

❖ 每一叶节点，各对应于

- 起自根节点的一条通路
- 某一可能的运行过程
- 运行所得的输出

❖ 叶节点深度 ~ 比较次数 ~ 计算成本

❖ 树高 ~ 最坏情况时的计算成本

树高的下界 ~ 所有CBA的时间复杂度下界

❖ 对于排序算法所对应的ADT，必有 $N \geq n!$

- ADT的每一输出（叶子），对应于某一置换
依此置换，可将输入序列转换为有序序列
- 算法的输出，须覆盖所有可能的输入

❖ 包含 N 个叶节点的排序算法ADT，高度不低于

$$\begin{aligned}\log_3 N &\geq \log_3 n! \\ &= \log_3 e \cdot [n \ln n - n + \mathcal{O}(\ln n)] = \Omega(n \cdot \log n)\end{aligned}$$

❖ 这一结论，还可进一步推广到

理想平均情况、**随机**情况（概率 $\geq 25\%$ ）...