Week 1：导引

- 这里是三五堂提高组课程的第一节课。

  由易到难，思维体系训练
  实战结合，创新协作培养
  兴趣导向，未来职业引领

  https://www.35tang.com　　https://www.三五堂.com

  扫码关注公众号　　添加辅导老师

- 算法竞赛概述

  几个近义词：算法竞赛、信息学竞赛（OI）、程序设计竞赛、竞技编程（CP）

  - 比赛流程
    - USACO
      - 前一年12月~当年2月：
        - 三场线上比赛，分为铜、银、金、铂金组。
      - 当年4月：
        - US Open（线上），分为铜、银、金、铂金组。
      - 当年5月：
        - Training Camp（25人）
      - 当年6月：
        - IOI
    - 国内

        - （最近两年内发生了很大变化，不排除会继续变化）
        - （难度逐级递增，跨度很大）
        - 9月~12月：
            - CSP-J/S初赛
            - CSP-J/S复赛
            - NOIP
        - 12月~次年2月：
            - NOIWC
            - PKUWC/THUWC
        - 次年2月~次年8月：
            - 省队选拔
            - NOI
        - 次年11月~再次年6月：
            - 北大集训 等一系列国家队选拔赛
            - IOI
    - 其他
        - 大学生竞赛：ICPC/CCPC
        - 线上非官方比赛：CodeForces/TopCoder/IPSC/...
- 常见赛制
    - COI赛制：（CSP，NOIP，NOI等）
        - 3~5h，3~4题，有部分分，无实时反馈，不计用时，个人
    - IOI赛制：（IOI，USACO，北大集训等）
        - 3~5h，3题，有部分分，有实时反馈，不计用时，个人
    - ICPC赛制：（ICPC，CCPC等）

        - 5h，10+题，无部分分，有实时反馈，记录用时，组队或个人
    - 部分赛制支持多种语言，但最适合算法竞赛的语言只有C++。
- 常用资源
    - CodeForces：codeforces.com
    - AtCoder：atcoder.jp
    - LibreOJ：loj.ac
    - 洛谷：www.luogu.com.cn
    - 搜索引擎
    - 《挑战程序设计竞赛》秋叶拓哉，岩田阳一，北川宜稔
    - 《算法竞赛入门经典》刘汝佳（“紫书”）
- 知识结构
    - 数学竞赛的知识结构（简洁）：
        - 四个互相独立的板块：ACGN（代数、组合、几何、数论）
    - 算法竞赛的知识结构（复杂）：
        - 四个大板块：图论、数据结构（DS）、动态规划（DP）、组合数学
        - 许多个小板块：字符串、数论、贪心、线性代数、计算几何、构造、提答……
        - 各个板块之间并不独立，互相之间往往有很大的重叠。
    - 大体上可分为算法和数学两条主线。
        - 刚入门时几乎不会用到任何数学；越往高处走，数学占比越大。
- 与计算机科学的关系
    - 计算机科学（CS）可分为理论、系统、应用三个大方向。

            - 理论：什么问题在理论上能够通过计算来解决？（图灵机）
            - 系统：在实践上如何建立复杂的硬件和软件系统？（超算、网络、数据库）
            - 应用：如何用算法解决工程和科研中的实际问题？（人工智能、交叉学科）
        - 算法竞赛研究的是理论这一大方向中的一个小方向，即算法与分析。
            - 算法与分析虽然只是计算机科学的一小部分，但却是整个学科的基石。
        - 因此，算法竞赛中处理的问题往往是抽象而脱离实际的。但是，为解决这些问题所提出的方法和思想，在现实问题中有着重要作用。
- 课程概述
    - 关于本课程
        - 三五堂计划推出一系列提高组课程，包括算法和数学两条主线。我负责设计和讲授。
        - 这一系列课程计划覆盖全部的提高组内容，和少量的更高级内容。本期课是该系列的第一期。
        - 每个板块会被按难度顺序分为若干个单元（例如“初步”→“进阶”→“高级”）。课程会将不同的板块穿插起来，以单元为单位进行组织，保证从易到难的学习顺序。
    - 如何学习本课程
        - 一名选手的核心竞争力：
            - 1. 经验：<u>熟练使用常见技巧</u>的能力
            - 2. 思维能力：<u>想出巧妙新颖的解法</u>的能力
            - 3. 代码能力：<u>多快好省地写出复杂代码</u>的能力
            - 这三项能力的提高都是无止境的！

- 课前：
    - 一般没有要求，但有时可能会布置预习任务，需阅读指定资料。
- 课上：
    - 善用暂停键。在遇到疑问或**给出例题**时，停下来思考。
    - 是否记笔记随你便。
- 课后：
    - 尽可能完成作业（不强求），若有多余时间则自行练习。
    - 独立思考，独立编程。必要时阅读题解。
    - 完成一道题目后进行**回顾**：
        - 阅读高手的优秀代码，学习如何写出更好的代码。（“为什么”）
        - 结合以前做过的题目，总结提炼本题中用到的思维方法和技巧。
            - 没有任何一套资料/课程能够教给你全部的技巧；必须要有自行总结提炼的能力。

- 数学基础：数学语言
    - 下面介绍的概念和记号无需死记硬背，我们以后会在使用中逐渐熟悉它们。
    - 集合
        - $\{\circ, \triangle, \square\}$
        - $\{1, -10.5, \sqrt{2}, \pi\}$
        - $\{x | x^2 = 1\} = \{1, -1\}$
            - 也记作 $\{x : x^2 = 1\}$
        - “属于”：$2 \in \{2,3\}, 3 \in \{2,3\}, 4 \notin \{2,3\}$

    - 子集（“包含于”）：$\{2,3\} \subseteq \{2,3,4\}, \{2,3\} \subseteq \{2,3\}$
    - 真子集（“真包含于”）：$\{2,3\} \subsetneq \{2,3,4\}$
    - 交集：$\{2,3\} \cap \{3,4\} = \{3\}$
    - 并集：$\{2,3\} \cup \{3,4\} = \{2,3,4\}$
    - 补集：$\{2,3,4\} \setminus \{2,3\} = \{4\}$
- 元组
    - $(x,y), (a,b,c), (\text{年龄}, \text{性别})$
    - 笛卡尔积：$\{1,2\} \times \{3,4\} = \{(1,3),(1,4),(2,3),(2,4)\}$
- 数集
    - $\mathbb{N}, \mathbb{Z}, \mathbb{Q}, \mathbb{R}$
    - $\mathbb{R}_{\geq 0}$
    - $\mathbb{R} \times \mathbb{R} = \mathbb{R}^2 = \{(x,y) | x \in \mathbb{R}, y \in \mathbb{R}\}$
- “对于所有”和“存在”
    - $\forall$ 读作“对于所有”；形如反着的字母A（All）。
    - $\exists$ 读作“存在”；形如反着的字母E（Exists）。
    - $\forall a \in \mathbb{Z}, a^2 \geq 0$（对于所有整数 a，都有 $a^2 \geq 0$ 成立）
    - 为避免歧义，可采用另一种语序（英语语序）：$a^2 > 0, \forall a \in \mathbb{Z}$
    - $\exists x \in \mathbb{R} : x^2 = 2$（存在实数 x 满足 $x^2 = 2$）
    - $\forall s \in \mathbb{R}_{\geq 0}, \exists x \in \mathbb{R} : x^2 = s$（对于所有非负实数 s，都存在实数 x 使得 $x^2 = s$）
- 序列和下标
    - 下标常用来表示标号：$a_1, a_2, a_3, \cdots, a_n$
        - 标号不一定得是数字：$h_{\text{小明}}, h_{\text{小红}}, h_{\text{小刚}}$ 分别表示三人的身高。

- 序列 $a_1, a_2, \cdots, a_n$ 简记作 $\{a_n\}$ 或 $a_{1\cdots n}$

- 求和、求积、求最值
  - $a_1 + a_2 + \cdots + a_n$ 简记作 $\sum_{i=1}^n a_i$
    - “i 跑遍从 1 到 n 的整数，计算所有 a_i 的和”
    - 相当于 `for(int i=1;i<=n;++i) sum+=a[i]`
  - $a_2 + a_4 + a_6 + \cdots$ 简记作 $\sum\limits_{i\geq 2, i \bmod 2 = 0} a_i$
    - “i 跑遍所有满足 $i \geq 2, i \bmod 2 = 0$ 的取值，计算所有 a_i 的和”
  - $a_1 \times a_2 \times a_3 \times \cdots \times a_n$ 简记作 $\prod_{i=1}^n a_i$
  - $a_2 \times a_4 \times a_6 \times \cdots$ 简记作 $\prod\limits_{i\geq 2, i \bmod 2 = 0} a_i$
  - $a_{1\cdots n}$ 中的最小值记作 $\min_{i=1}^n a_i$，最大值记作 $\max_{i=1}^n a_i$
  - $a_2, a_4, a_6, \cdots$ 中的最小值记作 $\min_{i\geq 2, i \bmod 2 = 0} a_i$，最大值记作 $\max_{i\geq 2, i \bmod 2 = 0} a_i$

- 命题
  - 一个**数学命题**是一个要么为真要么为假的数学陈述。例如：
    - “非负实数必有实平方根。”（真命题）
    - “$x^2 > 0, \forall x \in \mathbb{R}$”（假命题）
  - 数学命题也可能依赖于在外部定义的量。例如：
    - “$x > 0$” 在大部分情况下不是数学命题，因为 x 未定义，自然无从谈论这句话的真假。
    - 但如果我们预先规定了 $x = -5$，那么此时 “$x > 0$” 就是一个数学命题（而且是假命题）。
  - 数学命题之间存在关系。例如：
    - 假定 x 为实数，则不论 x 取何值，如果命题 “$x \neq 0$” 为真，则命题 “$x^2 > 0$” 一定也为真。

            - 这种关系称为**推出**，记作 $(x \neq 0) \Rightarrow (x^2 > 0)$ ，读作“$x > 0$推出$x^2 > 0$”。
        - 另一方面，假定 x 为实数，则显然也有 $(x^2 > 0) \Rightarrow (x \neq 0)$ 。所以这两个命题同真同假。
            - 这种关系称为**等价**，记作 $(x \neq 0) \Leftrightarrow (x^2 > 0)$ ，读作“$x > 0$等价于$x^2 > 0$”。
            - 如果命题 P 与 Q 满足 $P \Leftrightarrow Q$，则一定有 $P \Rightarrow Q, Q \Rightarrow P$。
            - 如果命题 P 与 Q 满足 $P \Rightarrow Q, Q \Rightarrow P$，则一定有 $P \Leftrightarrow Q$。
            - 有时我们会说“ P 成立当且仅当 Q”，这等于是在说 $P \Leftrightarrow Q$ 。
- 数学基础：时间复杂度
    - 大体思想
        - 一个程序在一组数据上的运行时间，由数据中的问题规模决定，例如序列长度、图的结点数。
        - 假定CPU执行每种基本操作需要相同的时间，则运行时间正比于程序执行的基本运算次数。
            - 什么是“基本操作”？一种常见的定义是：
                - 加减乘除、位运算、逻辑运算
                - 基本变量的定义、赋值和查值
        - 记 $f(n)$ 表示问题规模为 n 的时候，程序进行的基本操作次数；则 $f(n)$ 可用于衡量程序的效率。
        - 最后注意一点：一般来说，我们默认考虑的是**最坏情况**，即 $f(n)$ 表示所有规模为 n 的问题实例中，能够让程序执行的最大的基本操作数。（有时也会考虑平均情况，但那样的话会特别说明。）
    - 形式的简化

- 一个问题是，$f(n)$ 的表达式可能很复杂，例如 $f(n)=3n^2+5.6n\sqrt{n}+\log_3 n+233$。
    - 对策：只保留最重要的一项。
    - 当 n 很大的时候，其他项相比 n^2 这一项都小得可以忽略不计，所以我们认为 $f(n)\approx 3n^2$。
- 另一个问题是，电脑的速度并不相同，同样的时间 在一台电脑上可以执行2个基本操作，在另一台电脑上可能只能执行1个基本操作。所以实际的运行时间 $t(n)$ 与 $f(n)$ 间的比值 $c=\frac{t(n)}{f(n)}$ 是未知的！这意味着，不论 $f(n)=2n^2$ 还是 $f(n)=3n^2$，n^2 前的系数无法给我们提供任何有用的信息，因为我们最终关心的是 $2cn^2, 3cn^2$，而不论是 $2c$ 还是 $3c$ 都是未知的量。
    - 对策：干脆放弃常系数，只保留 n 的次数。我们不再区分 $f(n)=2n^2$ 还是 $3n^2$，我们只知道 $f(n)\propto n^2$。
    - 放弃常系数之后，剩下的东西表达了什么？表达了 $f(n)$ 随 n 的**变化趋势**。
- 最终，对于 $f(n)=3n^2+5.6n\sqrt{n}+\log_3 n+233$ 这一函数，经过两重简化，我们保留下的只有"$f(n)$ 近似地正比于 n^2"这一信息，记作 $f(n)=O(n^2)$（大O记号）。这一信息表达了 $f(n)$随n 的变化趋势，称为程序的**时间复杂度**。

- 程序的例子
    - 例：八皇后，$O(1)$
    - 例：双层循环（`i=6 to n, j1=i-5 to i, j2=i to n`），$O(n^2)$
    - 例：双层循环，$O(n)$（指数不一定等于循环层数）
    - 例：二分，$O(\log n)$
    - 例：按位枚举，$O(2^n)$

    - 例：全排列，$O(n!)$
- 数学的例子
    - $\pi^{100} = O(1)$
    - $2n^3 + 3n = O(n^3)$
    - $2n^3 - 3n = O(n^3)$
    - $n^2m + nm^2 + n^2m^2 = O(n^2m^2)$（多元情况）
    - $5\log_2 n + 8\log_3 n = O(\log n)$
        - 换底公式：$\log_a n = \log_a b \cdot \log_b n$
        - 对数的底数只影响常系数，所以在大O记号中省略底数。
    - $2^n + 3^n + n^{100} = O(3^n)$
    - $n^{0.01} + (\log_2 n)^{100} = O(n^{0.01})$
    - $2^n + n! = O(n!)$
    - $n \cdot n! = O((n+1)!)$
- 复杂度分析的不足
    - 电脑执行各种基本操作真的用时相同吗？
        - 不。部分基本操作（如除法、特定情况下的变量查值）比加法要慢十倍左右。
    - 常系数真的可以忽略吗？
        - 不。即使采用了复杂度正确的解法，在题目中的得分也往往取决于时间开销上的常系数。
            - 相反，即使采用了复杂度不够好的解法，也可能因为常系数很小而拿到超出预期的分数。
        - 在这一系列课程中的晚些时候，会介绍一些降低常系数的技巧。
    - 最坏情况一定会出现在测试数据里吗？
        - 不一定。所以有时复杂度不够好的解法，如果很难构造出最坏情况的数据，也可能可以获得超出预期

的分数。

- 空间复杂度
  - 与时间开销类似，空间开销（内存使用量）也可表示为关于问题规模 n 的函数 $f(n)$ 。
  - 对时间开销，$f(n)$ 表示的是基本操作数；对空间开销，$f(n)$ 表示的是定义的所有变量的总字节数。
  - 因此，对空间开销也可使用大O记号。
  - 例：若定义 `int a[n]` ，则空间复杂度为 $O(n)$ 。
  - 例：若定义 `char a[10*n];int a[2*n][n+5]` ，则空间复杂度为 $O(n^2)$ 。
  - 例：若定义 `double a[1<<n][n]` ，则空间复杂度为 $O(2^n \cdot n)$ 。
- 实操解题
  - 题面阅读
  - 思考解法并分析时间开销
  - 实现代码、提交测评与调试
  - 对拍检验
  - 阅读题解和优秀代码
- 本次课无作业。谢谢聆听！