

题型概述

在算法竞赛中，有多种多样的问题类型。



传统题

传统题 是目前算法竞赛中较为常见的题型。

选手需要提交源代码，评测系统会使用事先准备好一些输入数据和相应的输出数据作为测试点¹，将选手提交的源代码编译后²，让选手程序读入输入数据，通过将选手输出与事先准备好的输出比较，来判断选手程序是否正确。这种评测方式被称之为 **黑盒评测**³。

对于一个测试点，往往还会设置时间限制和空间限制。

时间限制，指的是程序运行时间的限制⁴。选手程序在一个测试点上的运行时间不能超过给定的时间限制。

空间限制，指的是程序使用的内存量的限制。选手程序在运行时占用的最大空间不能超过给定的空间限制。

在程序正常运行结束后，选手的输出会和测试点输出进行比对。这种比对一般采用过滤文末换行和行末空格之后，进行全文比对的方式。对于某些特殊的题目，会使用 **Special Judge** 来进行比对。

这一过程结束后，评测系统会根据程序的运行状态，给出不同的 **评测结果**⁵：

- Accepted (AC)：选手程序被接受。
- Compile Error (CE)：选手程序无法正常编译。
- Wrong Answer (WA)：选手程序正常结束，但是选手程序的输出与测试点输出不符。
- Presentation Error (PE)：选手程序正常结束，但是格式不符合要求⁶。
- Runtime Error (RE)：选手程序非正常结束（选手程序结束时的返回值不为零）。
- Time Limit Exceeded (TLE)：选手程序运行的时间超过了给定的时间限制。
- Memory Limit Exceeded (MLE)：选手程序占用的最大空间超过了给定的空间限制。
- Output Limit Exceeded (OLE)：选手程序输出的内容的量超过了最大限制。

在 ICPC 赛事中，你的程序需要在一道题目的所有测试点上都取得 AC 状态，才能视为通过相应的题目。在 OI 赛事中，在一个测试点中取得 AC 状态，即可拿到该测试点的分数⁷。

提交答案题

提交答案题 是直接提交答案的题目。该种题目一般会给出输入文件，要求提交包含有 `XXX1.out`、`XXX2.out`、`XXX3.out` ... `XXXn.out` 的压缩包、文件夹或纯文件。

提交答案后，评测系统会比较答案文件与标准答案，根据选手答案的优劣情况和任务完成度，给予一定的分数。

因为提交答案题不需要运行源程序，故提交答案题不存在时间和空间限制。

做这种题目一般有两种方法：

- 手玩。这种方法简单粗暴，但是遇到较大的数据就没辙了。
- 编写一个程序来获得答案文件。

交互题

交互题 是需要选手程序与测评程序交互来完成任务的题目。一类常见的情形是，选手程序向测评程序发出询问，并得到其反馈。测评程序可能对选手的询问作出限制，或调整应答策略来尽可能增加询问次数，这也给题目带来了更多变化。

更详细的交互题讲解可以看 [交互题](#)。

交互方式主要有如下两种。虽然技术上有不小的差异，但在考察算法的本质它们并没有实际区别。

STDIO 交互

STDIO 交互（标准 I/O 交互）是 Codeforces、AtCoder 等在线平台的交互手段，也是 ICPC 系列赛事中的标准。Codeforces 提供了一个更加简要的 [说明（英文）](#)。

ZQC 的迷宫



LOJ #559. 「LibreOJ Round #9」ZQC 的迷宫

请注意最下方添加内容。

本题是一道交互题。

位于 $n \times m$ 个方格组成的黑暗迷宫的你，需要走到这个迷宫的终点，以完成迷宫挑战。

最开始，你位于迷宫的起点即 $(1, 1)$ 处，且面向右侧，终点位于 (n, m) 处。迷宫中任意两个方格之间均连通，且仅有唯一的一条路径，两个相邻（即上、下、左、右四连通）方格间长度为一个单位长度。两个相邻方格之间可能会有墙壁，墙壁厚度相对于方格而言非常小，粗略不计。迷宫的边界均有墙壁，且每一堵墙壁均与边界连通。迷宫是完全黑暗的，这意味着，你无法得到除 (n, m) 以外的任何信息。

为了在黑暗条件下尽量不迷路，每次前进时你只能从当前格子出发，沿着左侧或右侧墙壁，左手或右手扶着墙壁前进，并且使扶着墙壁的手移动距离恰好为一个单位长度。需要注意的是，若左侧或右侧墙壁不存在，则沿该侧方向无法前进。

在黑暗中过久的你会感到恐惧，因此你需要在你尽早走出迷宫。如果你没有在限定步数内走出迷宫，挑战将会失败。

对于这类题目，选手只需像往常一样将询问写到标准输出，**刷新输出缓冲** 后从标准输入读取结果。选手程序刷新输出缓冲后，通过管道连接它的测评程序（称为交互器）才能立刻接收到这些数据。在 C/C++ 中，`fflush(stdout)` 和 `std::cout << std::flush` 可以实现这个操作（使用 `std::cout << std::endl` 换行时也会自动刷新缓冲区，但是 `std::cout << '\n'` 不会）；Pascal 则是 `flush(output)`。

Grader 交互

Grader 交互方式常见于 IOI、APIO 等国际 OI 赛事（特别是 CMS 平台的竞赛）。

Gap



UOJ #206. 【APIO2016】Gap

有 N 个严格递增的非负整数 a_1, a_2, \dots, a_N ($0 \leq a_1 < a_2 < \dots < a_N \leq 10^{18}$)。你需要找出 $a_{i+1} - a_i$ ($0 \leq i \leq N - 1$) 里的最大的值。

你的程序不能直接读入这个整数序列，但是你可以通过给定的函数来查询该序列的信息。关于查询函数的细节，请根据你所使用的语言，参考下面的实现细节部分。

你需要实现一个函数，该函数返回 $a_{i+1} - a_i$ ($0 \leq i \leq N - 1$) 中的最大值。

对于这类题目，选手只需编写一个特定的函数完成某项任务，它通过调用给定的若干辅助函数来进行交互。为了便于选手在本地测试，题目会下发一个头文件与一个参考测评程序 `grader.cpp`（对于 Pascal 语言是一个库 `graderlib`），选手将自己的程序与 `grader.cpp` 一同编译方可得到可执行文件。

```
1  g++ grader.cpp my_solution.cpp -o my_solution -Wall -O2
2  ./my_solution      # 执行程序
```

编译得到的程序表现与传统题程序类似。它会打开固定的文件，以固定的格式读取数据，调用选手编写的函数，并将结果和若干信息（例如询问的次数、答案正确性）显示在标准输出上。

实际测评时，选手的程序会与一个不同的 `grader.cpp` 编译。这个 `grader.cpp` 将以类似的方式调用选手编写的函数，并记录其得分。一般来说，这个版本的 `grader.cpp` 所有全局符号都会设为 `static`，也即不能通过冲突命名的方式破解它，但是任何尝试突破 `grader` 限制的行为都会被判失格 (disqualification)。

差别

STDIO 交互的一个明显优势在于它可以支持任何编程语言，但是输入输出的耗时容易成为问题设计的瓶颈，导致有时无法区分程序的时间效率差别；Grader 交互则恰好相反，由于函数调用的开销不大，常常可以允许 10^6 数量级的询问次数，但是语言的限制是其短板。

如果自己设计题目或举办比赛，需要对二者认真权衡和比较。

通信题

通信题 是需要两个选手程序进行通信，合作完成某项任务的题目。第一个程序接收问题的输入，并产生某些输出；第二个程序的输入会与第一个的输出相关（有时是原封不动地作为一个参数，有时会由评测端处理得到），它需要产生问题的解。

通信题的例子有：[UOJ #178. 新年的贺电](#)，[#454. 【UER #8】打雪仗](#) 等。

本地测试的方法由于题目设定的不同而多种多样，常用的形式如：

- 手工输入
- 编写一个辅助程序，转换第一个程序的输出到第二个程序的输入
- 用双向管道将两个程序的标准输入/输出连接起来

由于评测平台对于通信题的支持有限，因而目前为止，通信题只常见于 IOI 系列赛和 UOJ 等少数在线平台举办的比赛。它仍是一个有待探索的领域。

函数补全题

函数补全题 是需要选手补全程序的题目。可以理解为在一道交互题中，题目给定了选手代码，要求编写辅助函数。

通常有以下几种形式：

- 给定一个程序，并告知要求补全的代码块将被嵌入在哪里。
- 不给出程序，而将输入信息作为待提交函数的参数。

这种题在 [LeetCode](#) 和 [PTA - 拼题 A](#) 上比较多见。

其他类型

Quine

Quine

写一个程序，使其能输出自己的源代码。

代码中必须至少包含十个可见字符。

题目很经典，但是在绝大多数 OJ 上都很难实现。

参考代码

注意：源代码不包含下方第一行（即 `// clang-format off`）。

```
1 // clang-format off
2 #include<stdio>
3
4 char *s={"#include<stdio>%cchar *s={%c%s%c};%cint main()
5 {printf(s,10,34,s,34,10);return 0;}"};
6
7 int main(){printf(s,10,34,s,34,10);return 0;}
```

参考资料与注释

1. 因为技术上和资源上的限制，一道题目的测试点大多数情况下不能覆盖满足数据范围的全部数据。

←


2. 对于 Python 这样的解释性语言则直接由解释器解释运行程序。 ←

3. 事实上评测系统的实现远比这个复杂，这里只是大概介绍了评测系统的评测过程。 ←

4. 准确来说，一般是程序的用户态时间。 [←](#)
5. 这里的评测结果大多也适用于其他类型题目。 [←](#)
6. 大多数评测系统会将 PE 状态归到 WA 状态当中。 [←](#)
7. 一些测试点可能会有部分分，选手在完成一个测试点的部分任务，或者选手的输出正确但不够优的情况下，可以获得一定比例的分。 [←](#)

 本页面最近更新：2025/5/3 19:43:25，[更新历史](#)

 发现错误？想一起完善？ [在 GitHub 上编辑此页！](#)

 本页面贡献者：[StudyingFather](#), [NachtgeistW](#), [c-forrest](#), [CBW2007](#), [Chrogeek](#), [countercurrent-time](#), [Enter-tainer](#), [glerium](#), [H-J-Granger](#), [hsfzLZH1](#), [lr1d](#), [kawa-yoiko](#), [Konano](#), [megakite](#), [nanmenyangde](#), [shuzhouliu](#), [sshwy](#), [Suyun514](#), [Xeonacid](#)

© 本页面的全部内容 [在 CC BY-SA 4.0 和 SATA 协议之条款下](#) 提供，附加条款亦可能应用