# 2025 华为软件精英挑战赛

# 决赛任务书

| | |
|---|---|
| 文档版本 | v1 |
| 发布日期 | 2025-04-14 |

华为技术有限公司

# 目 录

# 1 更新记录

表1-1 更新记录

| 版本 | 修改说明 | 发布时间 |
| --- | --- | --- |
| v1 | 正式发布 | 2025-04-14 |

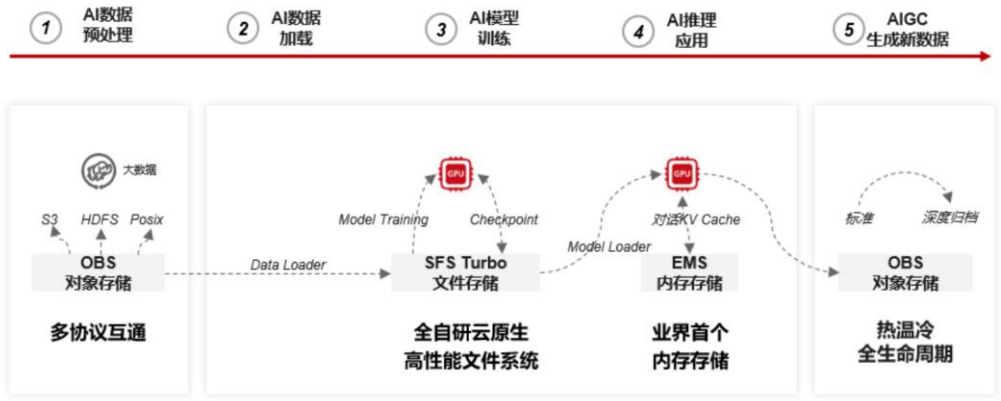
# 2 赛题背景

* 术语表（下述背景中如 ***对象（斜体、下划线、加粗）*** 的字样为相关术语，可在此查阅）

| | | |
|---|---|---|
| **背景术语** | ***对象存储服务*** | 【链接】一种基于 ***对象*** 的存储服务，为客户提供海量、安全、高可靠、低成本的数据存储能力，使用时无需考虑容量限制。 |
| | ***OBS*** | 【链接】华为云提供的 ***对象存储服务*** ，提供安全稳定、性能领先、无限弹性扩展的存储能力，降低使用成本，驱动千行百业数据价值变现。 |
| | ***SFS Turbo*** | 【链接】华为云提供的高性能弹性文件服务，提供按需扩展的高性能文件存储，用于 AI 训练、AIGC、自动驾驶、渲染、EDA 仿真、企业 NAS 应用等高性能场景，为其提供共享存储访问能力。 |
| **通用术语** | ***机械硬盘*** | 【链接】传统普通硬盘，有盘片，磁头，盘片转轴等组成。***随机读写*** 时需要频繁寻道，也就需要磁头和探针频繁的转动，性能远差于 ***顺序读写*** 。 |
| | ***磁头*** | 【链接】硬盘存储的结构中的一部分，负责读取与写入数据至硬盘。从硬盘中读取数据时，它会在硬盘上方浮动。 |
| | ***随机读写*** | 【链接】在存储设备上的不同位置进行非连续的读取或写入操作。每次操作的目标位置是随机的。对于机械硬盘而言，磁头需要频繁移动以定位数据。 |
| | ***顺序读写*** | 【链接】在存储设备上从某个位置开始按顺序连续读取或写入。对于机械硬盘友好，性能为 ***随机读写*** 场景的数倍。 |
| **对象存储相关术语** | ***对象*** | 【链接】***对象存储服务*** 提供的数据存储基本单位，一个对象是一个文件的数据与其相关属性信息（元数据）的集合体。用户上传至对象存储服务的数据都以对象的形式保存。 |
| | ***对象标签*** | 【链接】用户可以对上传的 ***对象*** 进行标签标记，通常来说，同一标签的对象具有潜在的不固定相似性（例如文件格式、文件大小、***生命周期*** 等）。 |
| | ***生命周期*** | 【链接】对象的生命周期指的是对象从创建开始，经过一系列的状态变化，直到最终被删除或归档的整个过程。 |
| | ***碎片化*** | 【链接】指存储空间使用效率低下，结果导致功能、运行效率变低等现象。 |
| | ***副本*** | 将对象复制多份分别存储到多块硬盘上，每一份为该对象的一个副本。多副本存储可避免单次故障导致的数据丢失，提升存储系统的可靠性与可用性。 |
| **赛题规则** | ***时间片*** | 真实业务场景中存储系统与外界的交互过程过于复杂，本题将一段时间内交互抽象为在一个时间片内的交互。时间片为本题的“单位时间”。 |
| | ***令牌*** | 本题将 ***磁头*** 进行一系列动作需要的时间开销抽象为令牌这一概念。磁头每一个动作均需要消耗令牌，且在单位时间内只能消耗有限数量的令牌。 |
| | ***对象块*** | 指对象在存储系统中可进一步细分的更小单元，在本题中为划分对象的“最小单元”。 |
| | ***对象大小*** | 每个对象拆分成 ***对象块*** 的数量。 |
| | ***控制模块*** | 分布式存储系统中用于全局管理和调度的逻辑组件，在本题中控制系统的写入、读取和删除对象操作。 |
| | ***存储单元*** | 硬盘上存储数据的单位。在本题中，一个存储单元可以恰好存放一个对象块。 |

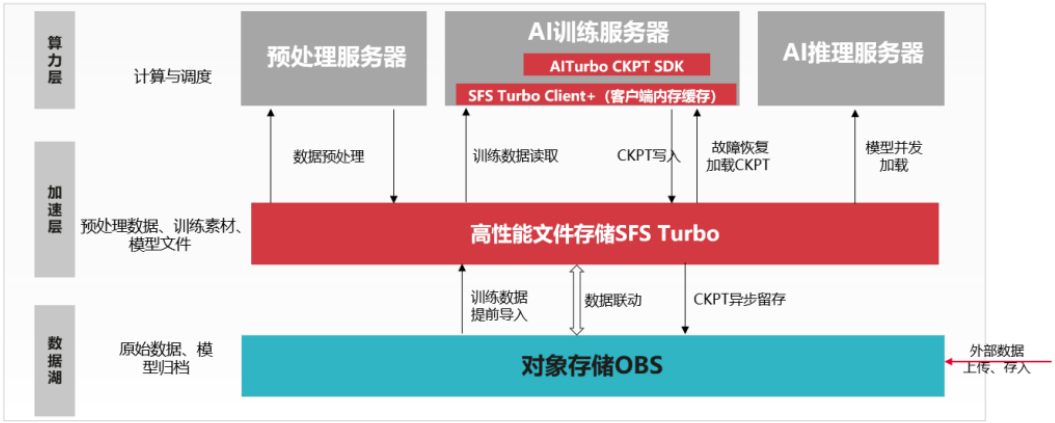
大模型的生成式 AI 技术的突破性发展推动了人工智能应用范围从传统的分类任务向广泛的生成任务扩展，引发了 AI 应用的爆发式增长，并引领 IT 产业进入全新的“AI 时代”。在此背景下，AI 产业的快速发展也促使云计算基础设施从以通用算力为核心向以智能算力为核心转变。在这一新型云计算基础设施中，数据的“算力”与“存力”相辅相成。尽管云数据中心在智能算力方面取得了显著进展，但存力方面的不足已成为制约效率的关键瓶颈。具体而言，存力问题主要体现在以下三个方面：海量数据存储成本高昂、大模型训练过程中的存储性能不足，以及推理过程中显存面临的“内存墙”问题。为有效解决这些存力挑战，华为云针对 AI 场景推出了 AI-Native 智算存储解决方案，通过提供高效且具有成本优势的先进存力，助力客户降低训练和推理成本，提升整体效率。

**图 1-1** AI-Native 智算存储加速 AI 全生命周期业务




*Source：《AI-Native 智算存储技术白皮书》*

针对 AI 训练环节中对海量小文件高性能、低时延处理以及大模型训练 CheckPoint 快速存储与恢复的特殊需求，在数据湖 OBS 之上，华为云引入了高性能存储加速缓存层 ***SFS Turbo*** 作为补充。SFS Turbo 能够为 AI 训练集群提供高性能数据读写能力，满足训练数据加载和故障恢复过程中 CheckPoint 机制的性能要求。然而，由于成本优化的需求，SFS Turbo 中的冷数据会被分级存储到 OBS 中。如何实现 OBS 中数据向 SFS Turbo 的快速加载，是云存储面临的重要挑战，同时也对加速 AI 训练效率具有关键意义。

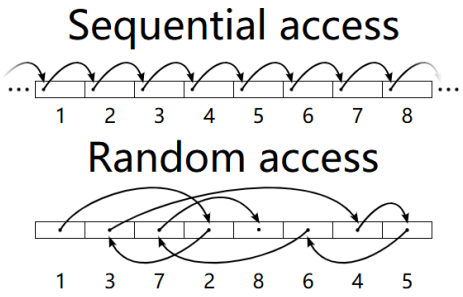



*基于 OBS+SFS Turbo 的华为云 AI 云存储解决方案*

# 3 赛题描述

## 3.1 赛题概述

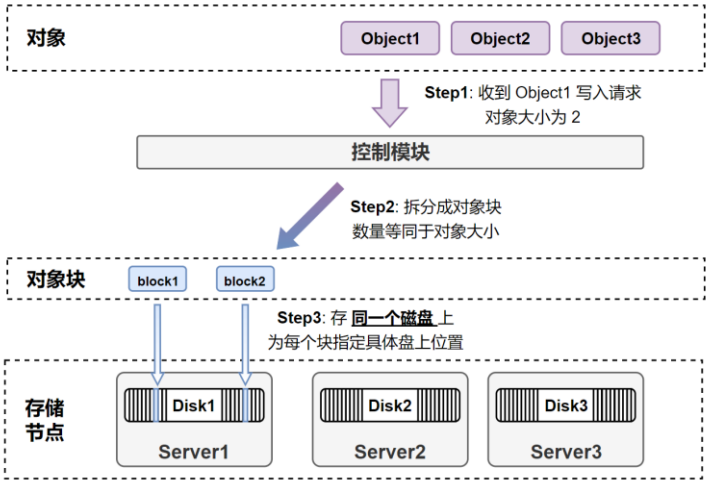
在数据存储领域，尤其是在 ***对象存储服务*** 等产品中，硬盘的读写性能是衡量产品竞争力的关键指标之一。在相同的 ***机械硬盘*** 上，***顺序读写*** 性能显著优于 ***随机读写*** 性能。在本题中，选手需要设计一个分布式对象存储系统，接受外界的写入、读取和删除对象请求。选手需要根据题目提供的 ***对象标签*** 、***对象大小*** 等信息，将具有相似特征的对象尽可能聚合写入，降低硬盘上数据的 ***碎片化*** 程度。在读取时，选手需要合理规划 ***磁头*** 的动作，提高系统读取对象的效率。




*Source：wikipedia*

## 3.2 分布式对象存储系统介绍

一个分布式存储系统由多个包含复数块硬盘的节点组成，并以基于对象的接口对外提供服务。外部服务可向该系统发起写入、读取和删除对象的请求。这些请求将被转发至系统中的 ***控制模块*** 并由控制模块执行。在本题中，你需要为该系统实现负责处理外部请求的控制模块。为简化问题，**本题中每个节点仅管理一块硬盘**。下图展示了这个系统的基本组成与一个 ***对象*** 的写入流程案例。



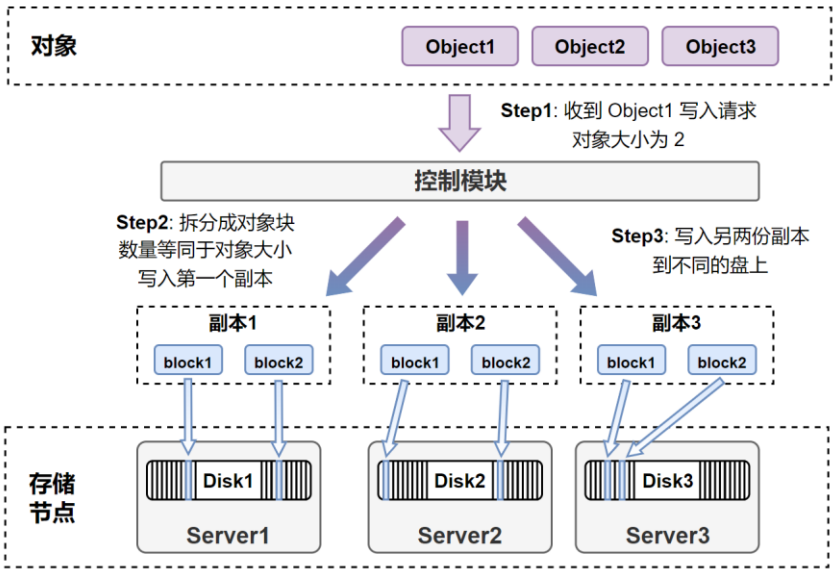

*图例 1：对象经过控制模块拆分成对象块，以对象块的粒度存储到具体的存储节点的硬盘*

- **Step1**：系统收到了 Object1 对象的写入请求，该对象的大小是 2。
- **Step2**：控制模块需要先将其拆分成 2 个 ***对象块***，然后指定一个盘，将 2 个对象块写入**同一个盘**中。
- **Step3**：控制模块需要为每个对象块指定盘上的具体写入位置。

## 3.3 对象冗余机制介绍

对于分布式对象存储系统来说，数据可靠性是重中之重。在本题中，数据可靠性由多副本存储机制保证，每个对象需要保证恰好有三个 ***副本*** （包括本体，总共三份），且你需要**保证同一个副本的对象块写在同一块盘上，不同副本的对象块写在不同的盘上**，引入冗余机制后，Object1 的写入流程案例如下：

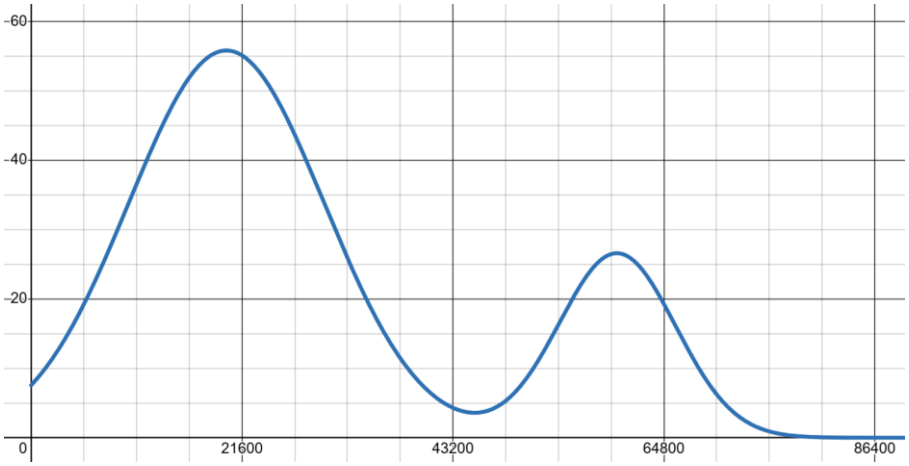
- **Step1**：系统收到了 Object1 对象的写入请求，该对象的大小是 2。
- **Step2**：控制模块需先将其拆分成 2 个对象块，然后指定一个盘，和每个对象块的具体写入位置，将 2 个对象块写入，完成第一个副本的写入。
- **Step3**：（重复两次）再次**指定一个不同的盘**，将 2 个对象块再次写入，指定每个对象块的具体写入位置，完成第二/三个副本的写入。




*图例 2：每个对象需要写入三个副本，三个副本写入不同的盘上，每个副本内的对象块各自写入指定的盘上位置*

## 3.4 对象标签介绍

***对象标签*** 是对象的一种属性。通常，对象标签由用户指定，同一标签的对象在某些特征上具有相似性（例如文件格式、文件大小、***生命周期*** 等）。在本题中，对象标签提炼自真实业务场景。如果两个对象具有相同的标签，**他们将会有更大概率在同一时间或者较近时间被读取**。




*图例 3：一种标签读取请求数量随时间变化示例图*

对于每一种对象标签，请求在时间轴上的分布具有规律性。**选手可以通过赛题组提供的公开数据数据观察和运用该规律，利用标签信息降低硬盘上数据的碎片化程度，最终使存储系统性能得到提升。**例如上图描述了一个标签在时间轴上读请求数量大致分布。关于下发的数据，请参照附件《题目与判题器补充说明》。

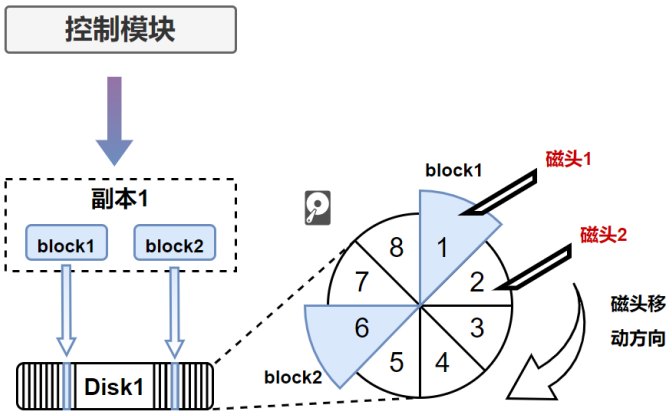
在实际业务中，由用户指定的对象标签在多数场景下是不明确的。换句话说，在对象写入时无法提前获取对象的标签信息，需要在后续读写过程中，根据用户行为总结出标签信息。

在决赛中，选手需要设计合理的算法，根据用户行为反向推理出标签信息和聚合信息，同时运用垃圾回收机制，将拥有相似特征的对象尽可能聚合写在相邻位置。为了体现选手对象标签预测的准确性，决赛将会有两轮完整的交互。

## 3.5 存储介质（硬盘）介绍

在决赛中，存储介质为经高度抽象的机械硬盘，每块硬盘由数个存储单元和**两个**磁头组成：

- **存储单元**：每块硬盘由$V$个 ***存储单元*** 组成，存储单元编号为1到$V$。每块存储单元的大小是相同的，可以恰好存放一个对象块。存储单元是环形排列的，第$V$个存储单元的下一个存储单元编号为1。
- **磁头**：每块硬盘有**两个**可以单向移动的 ***磁头*** 。磁头可以用于读取数据。

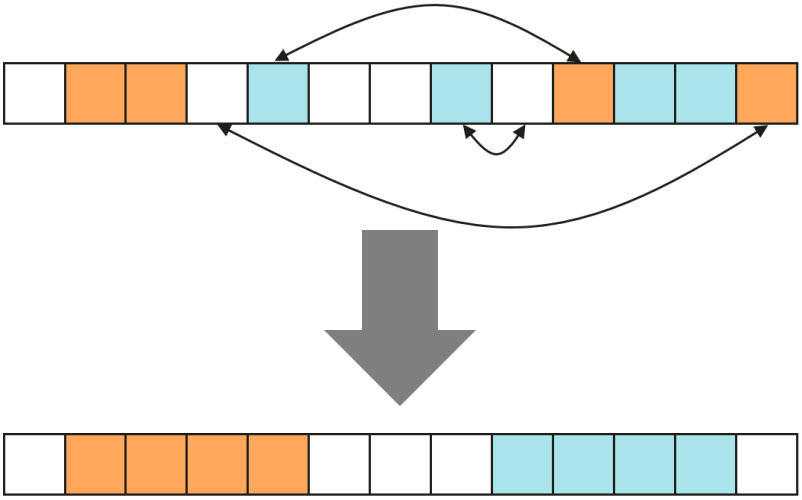



*图例 4：存储单元在磁盘中排成环形结构。磁头需要挪到放置了对象块的存储单元才能读取该对象块*

## 3.6 存储系统垃圾回收机制介绍

分布式存储系统中的垃圾回收（Garbage Collection，GC）机制是指系统自动回收不再使用的存储资源的过程。垃圾回收还具有整理数据，减少数据碎片化的功能。

垃圾回收的底层原理为存储系统自发进行的数据搬迁，即将一些存储单元中的数据迁移到另外的存储单元的行为。垃圾回收一般是定时触发的，为了减少对前台业务的影响，系统会限制单次搬迁数据的总量。



*复赛新增图例：进行垃圾回收后，具有相同对象标签的数据具有更好的连续性*

在本题中，规定每逢时间片编号为1800的倍数时，可以对每个硬盘进行最多$K$次交换存储单元的操作，每次操作为同一个盘内两个不同的存储单元交换其数据。为了简化垃圾回收系统，与写入、删除类似，垃圾回收系统无需转动磁头，选手的每次操作将瞬间成功。

## 3.7 存储系统过载保护机制介绍

分布式对象存储系统也需要有过载保护功能。过载保护是指在系统负载超过其设计标准时，采取一系列措施来防止系统性能急剧下降或崩溃的过程。这些措施通常包括限制新请求的进入、调整资源分配、或者释放部分负载的行为。

过载保护机制中的快速失败策略旨在在系统即将达到其处理极限时，迅速识别并拒绝或延迟处理部分请求，以保护系统的核心功能。这种策略有助于确保系统在高负载下仍然能够提供基本的服务质量，并避免因过载而导致的更严重的后果。

在本题中，控制系统需要对读取操作进行过载保护，当存储系统发现某个读取操作无法在规定时间内完成时，需要快速对请求返回失败。每一个读取请求都必须在 105 个时间分片内被响应，若无法完成该请求，存储系统可以上报“请求繁忙”并拒绝该请求。拒绝请求会导致扣分，请求拒绝地越早，扣分越少。

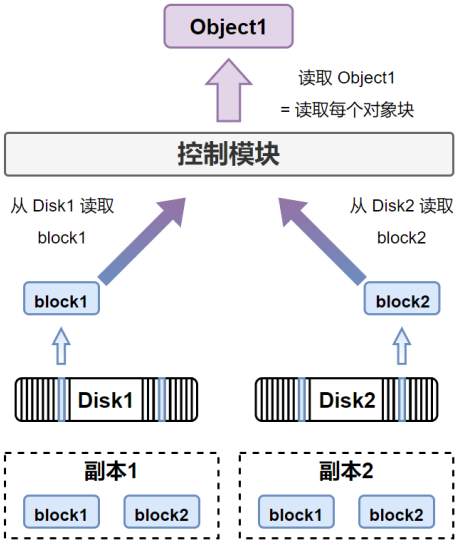
# 4 判题过程

在决赛中，整个判题过程包含两轮，每一轮包含若干次时间片交互，每次交互包含数个事件。判题第二轮和第一轮的事件信息均一致（包括删除对象编号，写入对象编号、大小、标签，读取对象请求编号和对象编号），为了减少输入输出的总量，判题器不再给出删除，写入，读取，垃圾回收事件的输入，仅给出时间片对齐事件的信息。第一轮给出事件类型如下：

- **时间片对齐事件**：给定当前时间片(TIMESTAMP)
  - **你的任务**：返回当前的时间片(TIMESTAMP)
  - **规则备注**：用于交互器和选手程序同步，方便选手调试使用。
- **对象删除事件**：给定 (id) 代表需要删除一个编号为 id 的对象。
  - **你的任务**：输出该对象当前所有还没完成的读取请求，这些请求将被直接取消（记为 0 分）。
  - **规则备注**：**删除与磁头的位置无关，且立即生效**，即盘上数据立即被删除。
- **对象写入事件**：给定 (id, size, tag) 代表需要写入一个编号为 id 、大小为 size、对象标签为 tag 的对象。在决赛中，对象标签可能是未知的，若 tag 为 0 代表当前该对象的对象标签需要选手在后续的行为中推断对象标签信息。
  - **你的任务**：针对这个对象的三个副本，给出每份副本各自写入目标的硬盘编号，以及每个副本中的每个对象块在该硬盘中的写入位置。写入的存储单元不能有其他对象的数据。
  - **规则备注**：
    - **写入与磁头的位置无关，且立即生效。**
    - **需要保证同一个副本的对象块写在同一块盘上，不同副本的对象块写在不同的盘上。**
    - 某个副本的对象块可以在硬盘上不连续存放。例如可以将对象某副本存放在存储单元 1,3 中。
    - 若选手无法选出三块有足够空间的硬盘存放该对象，则选手程序被判为 0 分。
- **对象读取事件**：给定 (req_id, id) 代表需要读取请求编号为 req_id，读取的是编号为 id 的对象。
  - **你的任务**：
    - 在接下来的时间片中（包含当前时间片），你需要给某些盘的磁头发送一系列的动作指令将该对象的所有对象块读出并组成完整的对象（见图例 5），或者发现读取请求无法在规定时间内读取完成，上报请求繁忙。
    - 每块硬盘有两个磁头。两个磁头初始位置均在存储单元1。磁头能向存储单元编号更大的一侧移动（特殊的，在第$V$个存储单元的磁头移动一次后到存储单元1）。
    - 磁头可以执行动作，每个动作均需要消耗 ***令牌*** 。每个磁头在每个时间片内最多消耗$G$个令牌，假设当前磁头在存储单元$x$，磁头可以执行下列动作：
      - **Jump**：跳跃到某个指定的存储单元$y$。磁头挪动至存储单元$y$。"Jump"动作固定消耗$G$个令牌。
        - 只能在每个时间片的开始执行"Jump"动作，且执行后该磁头不能再有其他动作。
      - **Pass**：忽略当前的存储单元$x$，磁头直接挪动至下一个存储单元$(x\%V)+1$。其中，%代表取模操作。"Pass"动作固定消耗1个令牌。
      - **Read**：读取当前的存储单元$x$，磁头挪动至下一个存储单元$(x\%V)+1$。读取消耗的令牌数规则如下：
        - 若该磁头上一次动作不是"Read"，消耗 64 个令牌。第一个时间片首次"Read"消耗 64 个令牌。
        - 否则，若磁头上一次动作消耗的令牌数为$pre_token$，本次动作消耗令牌数为$max(16, ceil(pre_token \times 0.8))$。其中，$ceil(x)$表示$x$向上取整。

★ 磁头在每个时间片的首次动作的上一个动作为**该磁头上一个时间片的最后一个动作**。

- 不同的磁头动作相互独立，同一硬盘上两个磁头可以指向同一存储单元。
- **对于每一个请求**，若其目标对象的每一个对象块均有至少一个副本**在选手程序收到请求后**被其所在硬盘读取过，则选手程序可上报该请求已完成。
  - 选手可以读取该对象的同一个副本，例如，如果你想完整的从这个盘上读取该对象的所有对象块（block1 和 block2），就需要分别挪动磁头到两个块位置进行读取，在读取途中也可以读取其他对象。
  - 选手也可以读取该对象不同副本的对象块组成完整的对象，从而实现更快的读取性能。例如：




*图例 5：从多个副本中分别读取同个对象的不同块*

- 每一个读取请求都必须在请求到达后的 105 个时间片内返回其中一种结果：请求的对象被删除（在对象删除事件中上报），请求读取成功，请求繁忙。
- 选手程序应在输出所有磁头动作后上报已完成的读取请求和繁忙的请求。

- **规则备注：**
  - 同一时间可能有多个读请求读同一个对象。
    - **在读取请求到达后**，若目标对象的每个对象块均被读取了至少一次，则选手可上报该请求读取成功。每个对象块的任意一个副本被读取可视为该对象块被读取。
      - ★ 例如，对于一个由2个对象块（block1，block2）组成的对象，在时间片1接收到读取请求1；在时间片2读取了 block1；在时间片3接收到读取请求2；在时间片4读取了 block2；那么在时间片4可以上报读取请求1成功。读取请求2在请求到来后（时间片3后）没有读取 block1，故不能上报读取成功。
  - 对象读取请求可跨越多个时间片，如在时间片$t$到达并在时间片$t+10$才完成。
  - 对于一个读取请求，在存储系统读取完毕后，可以选择不上报读取成功，且可以选择在之后某个时间片再上报读取成功，但是这样做不会让得分更高。
  - 磁头仅与读取操作相关，写入和删除操作无需移动磁头。

- 垃圾回收事件：在时间片编号为 1800 的倍数时触发，允许选手对每个硬盘进行一次垃圾回收。
  - 你的任务：
    - 对于每个硬盘，进行最多$K$次交换存储单元的操作，每次操作，需要选择存储单元 A 和存储单元 B，将原来存储单元 A 中数据搬迁至存储单元 B 中，并将原来存储单元 B 中的数据搬迁至存储单元 A 中。其中存储单元 A 和 B 中均可以没有数据。
    - **交换存储单元与磁头的位置无关，且立即生效。**
  - 规则备注：

- 选择的存储单元 A 和 B 必须位于同一块硬盘上，存储单元编号必须不一致。
- 搬迁动作是依据选手的输出顺序结算的，换言之，你可以先交换存储单元 A 和 B，再交换存储单元 B 和 C，原来在存储单元 A 中的数据会出现在 C 中。

在决赛中，第一轮交互结束后，判题器将额外给出部分对象的标签，详见章节“输入与输出”。

在决赛中，第二轮判题器只会给出时间片对齐事件的输入，但是选手仍然需要按顺序给出时间片对齐事件，对象删除事件，对象写入事件，对象读取事件，垃圾回收事件的输出。

# 5 得分规则

选手程序得分为每个读请求的得分之和，再减去因上报繁忙扣除的分数。对于每个读请求，其得分计算方式如下：

一个大小为$size$的对象，假设某个读取请求在第$i$个时间片发送给存储系统，在第$i+x$个时间片上报读取成功，则其得分$SCORES$为：

$$SCORES = f(x) * g(size)$$

其中：

$$f(x) = \begin{cases} -0.005x + 1, & 0 \le x \le 10 \\ -0.01x + 1.05, & 10 < x \le 105 \\ 0, & 105 < x \end{cases}$$

$$g(size) = (size + 1) * 0.5$$

一个大小为$size$的对象，假设某个读取请求在第$i$个时间片发送给存储系统，在第$i+x$个时间片上报请求繁忙，则其扣分$SCORES$为：

$$SCORES = h(x) * g(size)$$

其中：

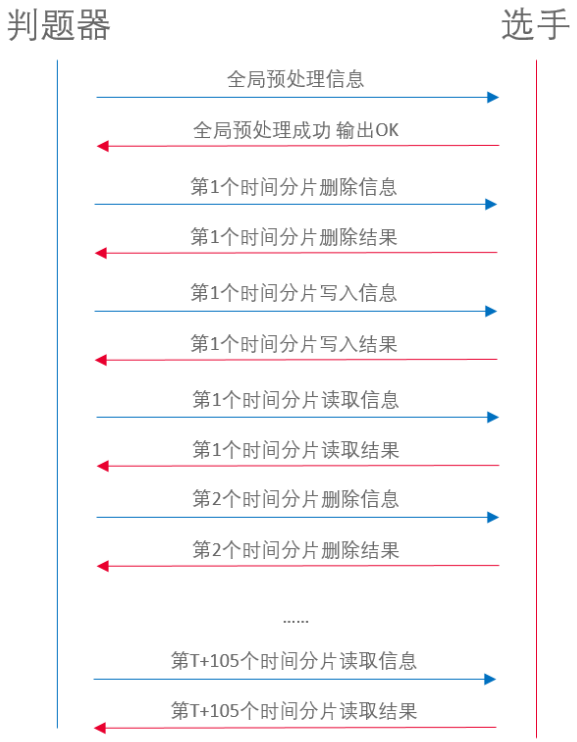
$$h(x) = x/105$$

若一个读请求在请求到达后 105 个时间片未上报结果，判题器会立即终止判题流程，选手成绩记 0 分。

更多与得分相关的问题，请参照附件《题目与判题器补充说明》。

# 6 输入与输出

选手程序与判题器的交互包含两个阶段：全局预处理阶段和每个时间片的交互。




## 6.1 全局预处理阶段

首先，判题器会给出以下输入：

```
T M N V G K
```

其中：

- T：代表本次数据有$T+105$个时间片，后续输入第二阶段（见章节 6.2）将循环交互$T+105$次。时间片编号为$1\sim T+105$。输入数据保证$1\le T\le 86400$。对于第$T+1\sim T+105$个时间分片，输入数据保证没有删除、写入和读取请求。
- M：代表对象标签数。对象标签编号为$1\sim M$。输入数据保证$1\le M\le 16$。
- N：代表存储系统中硬盘的个数，硬盘编号为$1\sim N$。输入数据保证$3\le N\le 10$。
- V：代表存储系统中每个硬盘的存储单元个数。存储单元编号为$1\sim V$。输入数据保证$1\le V\le 16384$，任何时间存储系统中空余的存储单元数占总存储单元数的至少10%。
- G：代表每个磁头每个时间片最多消耗的令牌数。输入数据保证$64\le G\le 500$。
- K：代表每次垃圾回收事件每个硬盘最多的交换存储单元的操作次数。输入数据保证$0\le K\le 100$。
- 决赛中，不会给出某个标签的对象在某个时间区间的操作总次数。

例如（更具体的样例解释，见附件《样例数据》）：

```
5 2 3 10 100 10
0
2
3
2
6
4
```

代表在本数据中，有$5+105$个时间片，2种对象标签，3个硬盘，每个硬盘有10个存储单元，每个磁头在每个时间片最多消耗100个令牌。在时间片1~110中，对象标签1,2分别总共删除了0,2个对象块，写入了3,2个对象块，读取了6,4个对象块。每次垃圾回收，每个盘允许最多 10 次交换操作。

对于全局预处理阶段，选手预处理完成后，需要输出：

```
OK
```

请注意：

- 选手输出完毕后，需要刷新输出缓冲区。具体操作和注意事项见章节 6.3。

# 6.2 每个时间片的交互

在第一轮中，每个时间片会按顺序进行时间片对齐事件，对象删除事件，对象写入事件，对象读取事件，以及在部分时间片进行垃圾回收事件。

在第二轮中，每个时间片只会给出时间片对齐事件的输入，其余输入不再给出。但是选手依然需要给出其他事件的输出。

交互将循环进行$T+105$次。

## 6.2.1 时间片对齐事件交互

判题器会给出以下输入：

```
TIMESTAMP current_timestamp
```

- 其中，大写的"TIMESTAMP"为固定的字符串，current_timestamp 为当前的时间片编号，范围为$1\sim T+105$且每次单调递增 1。

例如：

```
TIMESTAMP 1
```

对于时间片对齐事件，选手处理完成后，需要输出：

```
TIMESTAMP current_timestamp
```

- 其中，大写的"TIMESTAMP"为固定的字符串，current_timestamp 为当前的时间片编号。选手的输出和判题器的输入需要保持一致。

例如：

```
TIMESTAMP 1
```

请注意：

- 时间片对齐事件是方便选手调试代码的事件，其本身没有特殊含义。
- 判题器在两轮交互中均会给出时间片的输入。
- 选手输出完毕后，需要刷新输出缓冲区。具体操作和注意事项见章节 6.3。

## 6.2.2 对象删除事件交互

判题器会给出以下输入：

```
n_delete
```

```
obj_id[1]
obj_id[2]
...
obj_id[n_delete]
```

其中：

- n_delete：代表这一时间片被删除的对象个数。请注意，n_delete 可能为 0。输入数据保证总删除次数小于等于100000。
- 接下来 n_delete 行，每行一个数 obj_id[i]，代表删除对象的对象编号。输入数据保证删除的对象一定在存储系统中。

例如：

```
1
2
```

代表在本时间片中，共有1个对象被删除，其对象编号为2。

对于对象删除事件，选手处理完成后，需要输出：

```
n_abort
req_id[1]
req_id[2]
...
req_id[n_abort]
```

其中：

- n_abort：代表这一秒被取消的读取请求的数量。请注意，如果没有需要取消的请求，选手也需要输出一个"0"。
- 接下来 n_abort 行，每行一个数 req_id[i]，代表被取消的读取请求编号。这里编号可以按任意顺序输出。

例如：

```
1
6
```

代表在本时间片中，共有 1 个读取请求被取消，其读取请求编号为 6。

请注意：

- 因为第二轮输入和第一轮保持一致，判题器在第二轮的交互中不再给出对象删除事件的输入。
- 选手输出的请求编号必须为删除对象的**当前未完成的读请求**，否则判题器会报错。
- 选手需要保证 n_abort 与接下来输出的 req_id 数量一致，否则判题器可能会报错或一直等待选手输入而卡住。
- 选手输出完毕后，需要刷新输出缓冲区。具体操作和注意事项见章节 6.3。

### 6.2.3 对象写入事件交互

判题器会给出以下输入：

```
n_write
obj_id[1] obj_size[1] obj_tag[1]
obj_id[2] obj_size[2] obj_tag[2]
...
obj_id[n_write] obj_size[n_write] obj_tag[n_write]
```

其中：

- n_write：代表这一时间片写入对象的个数。输入数据保证在一轮中总写入次数小于等于100000。
- 接下来 n_write 行，每行三个数 obj_id[i]、obj_size[i]、obj_tag[i]，代表当前时间片写入的对象编号，对象大小，对象标签编号。在决赛中，obj_tag[i]可能为 0，表示对象标签信息被隐藏。输入数据保证 obj_id 为1开始每次递增1的整数，且$1 \leq obj_size[i] \leq 5$，$0 \leq obj\_tag[i] \leq M$。

例如：

```
2
1 1 5
```

```
3. 2 2 6
```

代表在本时间片中，共有 2 个对象写入，1 号对象大小为 1，标签编号为 5。2 号对象大小为 2，标签编号为 6。

对于对象写入事件，选手处理完成后，需要输出：

```
1. obj_id[1]
2. rep[1] unit[1][1] unit[1][2] ... unit[1][obj_size[obj_id[1]]]
3. rep[2] unit[2][1] unit[2][2] ... unit[2][obj_size[obj_id[1]]]
4. rep[3] unit[3][1] unit[3][2] ... unit[3][obj_size[obj_id[1]]]
5. obj_id[2]
6. rep[1] unit[1][1] unit[1][2] ... unit[1][obj_size[obj_id[2]]]
7. rep[2] unit[2][1] unit[2][2] ... unit[2][obj_size[obj_id[2]]]
8. rep[3] unit[3][1] unit[3][2] ... unit[3][obj_size[obj_id[2]]]
9. ...
10. obj_id[n_write]
11. rep[1] unit[1][1] unit[1][2] ... unit[1][obj_size[obj_id[n_write]]]
12. rep[2] unit[2][1] unit[2][2] ... unit[2][obj_size[obj_id[n_write]]]
13. rep[3] unit[3][1] unit[3][2] ... unit[3][obj_size[obj_id[n_write]]]
```

其中：

- 输出包含$4 * n_write$行，每4行代表一个对象：
  - 第一行一个整数 obj_id[i]，表示该对象的对象编号。
  - 接下来一行，第一个整数 rep[1]表示该对象的第一个副本写入的硬盘编号，接下来对象大小(obj_size)个整数 unit[1][j]，代表第一个副本第$j$个对象块写入的存储单元编号。
  - 第三行，第四行格式与第二行相同，为写入第二，第三个副本的结果。

例如：

```
1. 2
2. 1 2 3
3. 2 4 5
4. 3 4 5
5. 1
6. 1 1
7. 2 3
8. 3 3
```

代表2号对象的第1个副本写入了1号盘的第2,3个存储单元中，2号副本写入了2号盘的第4,5个存储单元中，3号副本写入了3号盘的第4,5个存储单元中。1号对象的3个副本分别写入了1,2,3号盘的第1,3,3个存储单元。

请注意：

- 因为第二轮输入和第一轮保持一致，判题器在第二轮的交互中不再给出对象写入事件的输入。
- 当前时间片写入的对象数可能为0，此时选手无需有任何输出。
- 选手输出的写入对象顺序无需和输入保持一致，但是需要在该时间片处理所有的写入请求。
- 选手输出的每个副本的存储单元数需要与该对象大小一致，否则判题器可能会报错或一直等待选手的输入而卡住。
- 如果当前时间片有输入的情况下，选手输出完毕后，需要刷新输出缓冲区。具体操作和注意事项见章节 6.3。

## 6.2.4 对象读取事件交互

判题器会给出以下输入：

```
1. n_read
2. req_id[1] obj_id[1]
3. req_id[2] obj_id[2]
4. ...
5. req_id[n_read] obj_id[n_read]
```

其中：

- n_read：代表这一时间片读取对象的个数。输入数据保证在一轮中总读取次数小于等于30000000。
- 接下来 n_read 行，每行两个数 req_id[i]、obj_id[i]，代表当前时间片读取的请求编号和请求的对象编号。输入数据保证读请求编号为 1 开始每次递增 1 的整数，读取的对象在请求到来的时刻一定在存储系统中。

例如：

```
2
5 3
6 2
```

代表在本时间片中，共有2个对象读请求，5号请求为读取3号对象，6号请求为读取2号对象。

对于读取事件，选手处理完成后，需要输出：

```
action[1][1]
action[1][2]
action[2][1]
action[2][2]
...
action[N][1]
action[N][2]
n_rsp
req_id[1]
req_id[2]
...
req_id[n_rsp]
n_busy
req_id[1]
req_id[2]
...
req_id[n_busy]
```

其中：

- 前$N*2$行是磁头的运动输出，其中 action[i][j]代表编号为$i$的硬盘的第$j$个磁头所对应的磁头的运动方式：
  - 1. 该磁头执行了"Jump"动作：这一行输出空格隔开的两个字符串，第一个字符串固定为"j"；第二个字符串为一个整数，表示跳跃到的存储单元编号。
  - 2. 该磁头没有执行"Jump"动作：这一行输出一个字符串，仅包含字符'p'、'r'、'#'，代表该磁头在当前时间片的所有动作，每一个字符代表一个动作。其中'p'字符代表"Pass"动作，'r'字符代表"Read"动作。运动结束用字符'#'表示。
- n_rsp：代表当前时间片上报读取完成的请求个数。
- 接下来 n_rsp 行，每行 1 个数 req_id[i]，代表本时间片上报读取完成的读取请求编号。
- n_busy：代表当前时间片的上报繁忙的请求个数。
- 接下来 n_busy 行，每行 1 个数 req_id[i]，代表本时间片上报繁忙的读取请求编号。

例如：

```
rr#
#
j 10
j 10
#
#
1
4
1
6
```

代表1号硬盘的 1 号磁头执行了2次"Read"动作，2 号磁头没有任何动作。2号硬盘的两个磁头均执行了"Jump"动作，跳跃到10号存储单元，3号硬盘两个磁头均没有进行任何动作。在本时间片中，有1个请求读取成功，为4号请求。有一个请求上报繁忙，为 6 号请求。

请注意：

- 因为第二轮输入和第一轮保持一致，判题器在第二轮的交互中不再给出对象读取事件的输入。
- n_rsp 和 n_busy 可以为 0，这意味着这一时间片内没有上报成功的读请求或者繁忙的请求。
- 请求必须在 105 个时间分片内上报读成功，繁忙或被取消，多次上报或不上报会被判题器报错。
- 在$T+105$个时间分片后，选手需要保证存储系统中所有读请求均被处理完毕。
- 请严格按照题目规定的格式输出，格式错误可能会被判题器判错。
- 选手输出完毕后，需要刷新输出缓冲区。具体操作和注意事项见章节 6.3。

### 6.2.5 垃圾回收事件交互

若当前时间片编号为 1800 的倍数，该时间片在对象读取事件后会进行一次垃圾回收事件。
判题器会给出以下输入：

```
GARBAGE COLLECTION
```

其中大写的"GARBAGE COLLECTION"为固定的字符串

对于垃圾回收事件，选手处理完成后，需要输出：

```
GARBAGE COLLECTION
n_gc[1]
s[1][1] t[1][1]
s[1][2] t[1][2]
...
s[1][n_gc[1]] t[1][n_gc[1]]
n_gc[2]
s[2][1] t[2][1]
s[2][2] t[2][2]
...
s[2][n_gc[2]] t[2][n_gc[2]]
...
n_gc[N]
s[N][1] t[N][1]
s[N][2] t[N][2]
...
s[N][n_gc[N]] t[N][n_gc[N]]
```

其中：

- 大写的"GARBAGE COLLECTION"为固定的字符串
- 接下来输出包含$N$块，每块代表一个盘的垃圾回收情况。
- 对于第$i$块，首先输出当前盘的交换操作次数 n_gc[i]。
- 接下来 n_gc[i]行，每行两个数 s[i][j], t[i][j]，代表本次操作为交换存储单元 s[i][j], t[i][j]中的数据。

例如：

```
GARBAGE COLLECTION
0
2
1 2
2 3
0
```

代表第1,3块盘没有需要交换操作，第2块盘交换了两次存储单元，第一次交换1,2存储单元中的数据，第二次交换2,3存储单元中的数据。

请注意：

- 因为第二轮输入和第一轮保持一致，判题器在第二轮的交互中不再给出垃圾回收事件的输入。
- 当前时间片某个硬盘可以没有数据需要搬迁。这种情况下，请输出 n_gc[i]为 0。
- 选手需要按硬盘编号顺序输出每一块盘的搬迁情况。
- 如果当前时间片编号大于 T，且为 1800 的倍数，仍然会有垃圾回收事件。
- 选手输出完毕后，需要刷新输出缓冲区。具体操作和注意事项见章节 6.3。

### 6.2.6 增量信息交互

第一轮结束后，判题器将会给出以下输入：

```
n_incre
obj_id[1] obj_tag[1]
obj_id[2] obj_tag[2]
...
obj_id[n_incre] obj_tag[n_incre]
```

其中：

- n_incre：第二轮相较于第一轮额外获得的对象标签数量。输入数据保证$0 \leq n_incre \leq 100000$。
- 接下来 n_incre 行，每行两个数 obj_id[i],obj_tag[i]，代表对象编号和其对应的对象标签编号。输入数据保证这些对象编号两两不重复，且均为第一轮隐藏了对象标签的对象。

例如：

```
2
1 3
2 4
```

代表对象 1 的标签为 3，对象 2 的标签为 4。

**对于增量信息交互，选手不需要输出任何内容。**

请注意：

- 增量信息交互后，存储系统中所有的状态均会清空，变为初始状态。
- 增量信息交互后，判题器会立即开启第二轮交互，第二轮交互仅给出所有的时间片对齐事件的输入，选手接受到输入后需要输出该时间片的所有的输出信息。

## 6.3 交互注意事项

选手在完成一个阶段交互后，需要进行刷新输出缓冲区操作，具体的：

- C 语言：fflush(stdout);
- C++语言：cout.flush();
- Java 语言：System.out.flush();
- Python 语言：stdout.flush();

特别提醒：

- 在任意阶段结束时，若选手有输出，都要以换行符为结束。例如，在全局预处理阶段，选手输出"OK"字符串后，需要输出一个换行符，再刷新输出缓冲区。
- 决赛特别提醒：linux fifo 管道最大只能容纳 64K 数据（windows 为 4K），第二轮数据请读入时间片后再输出结果，否则可能出现管道被数据塞满的情况。
- 选手不得输出多余的字符（包括空格和换行符），否则判题器可能会将选手程序判为 0 分。
- 判题器会在选手首次出错的位置报错，具体错误见附件《题目与判题器补充说明》。只有完成完整交互的程序才能获得得分。
- 选手可能会在同一时间片接受到大量请求。题目对单个时间片的运行时间不做限制，仅限制程序交互运行的总时间。
- 题目输入输出总量较大，请注意代码效率问题。
- 选手程序在判题平台运行时没有任何写权限，请不要将日志落盘，避免程序报错。
- 其他问题请参照附件《题目与判题器补充说明》。