CCF 大学生计算机系统与程序设计分赛区竞赛

CCSP 2019

时间：2019 年 5 月 18 日 09:30 ~ 15:30

题目名称	CCSP 子图计数	更强的 RAID5	简化流操作
题目类型	传统型	传统型	传统型
输入	标准输入	标准输入	标准输入
输出	标准输出	标准输出	标准输出
每个测试点时限	1.0 秒	1.0 秒	1.0 秒
内存限制	512 MB	64 MB	512 MB
子任务数目	8	10	0
测试点是否等分	否	是	是

CCSP 子图计数（p1）

【题目描述】

本题中，我们考虑的图都是无向图，且没有自环，没有重边，边上无权值，每个点上标有一个字符，为 C、S 或 P。

我们称一个图是“CCSP 图”，当且仅当该图满足以下所有条件：

- 有 4 个点、4 条边的连通图；
- 4 个点上的字符（不考虑顺序）为两个 C、一个 S、一个 P；
- 图中存在一个长度为 3 的环。

现在，给你一个图，你需要统计其中“CCSP 子图”的个数。即你需要在图中选出 4 个点与 4 条连接这些点的边，统计多少种选法会使得选出的图是“CCSP 图”。注意：如果两种选法选取的点相同而边不同，也认为是不同的选法。

【输入格式】

从标准输入读入数据。

第一行包含两个整数 n, m，分别表示图的点数和边数。

第二行包含一个长度为 n 的仅包含 C、S、P 的字符串，其中第 i 个字符表示 i 号点上的字符。

接下来 m 行，每行包含两个整数。设其中第 j 行的整数分别为 u_j, v_j，则表示存在一条连接 u_j 与 v_j 的边。

图上的点以从 1 到 n 的不同正整数编号；保证输入的图没有自环与重边；每行相邻的两个整数之间用一个空格隔开。

【输出格式】

输出到标准输出。

仅输出一行，包含一个整数，表示图中“CCSP 子图”的个数。

【样例 1 输入】

```
4 5
SCPC
1 2
2 3
3 4
4 1
1 3
```

【样例 1 输出】

```
4
```

【样例 2 输入】

```
7 10
CCCSSPP
1 2
2 3
3 1
3 4
4 5
5 6
6 7
7 4
2 4
3 7
```

【样例 2 输出】

```
5
```

【子任务】

对于所有的数据，保证 $0 \leq n, m \leq 10^6$，每条边满足 $1 \leq u_j, v_j \leq n$。

本题以子任务的方式评测。每个子任务包含若干测试点，你需要通过某个子任务的所有测试点才能得到该子任务的分数。

- （14 分）$n, m \leq 3$。
- （34 分）$n, m \leq 4$。
- （21 分）$n, m \leq 10$。
- （13 分）$n, m \leq 100$。
- （8 分）$n, m \leq 1000$。
- （5 分）$n, m \leq 10^4$。
- （3 分）$n, m \leq 10^5$。
- （2 分）$n, m \leq 10^6$。

除样例外的所有测试数据均为随机生成的，具体生成方式如下：

- 人工指定 n, m；
- 独立生成每个点的字符，其中 `C` 的概率为 $1/2$，`S` 与 `P` 的概率各为 $1/4$；
- 依次生成每一条边：每个点被连接的概率正比于它的度数 $+1/\sqrt{n}$，但是如果生成出了自环或重边，则丢弃之并重新生成。

【提示】

本题最后的一些部分分难度较大而分数较少，性价比很低，请合理规划比赛时间。

更强的 RAID5（raid5）

【题目背景】

独立硬盘冗余阵列（RAID, Redundant Array of Independent Disks）是一种现代常用的存储技术。它以一定形式，将数据分散、冗余地存储在多个磁盘上，从而当部分磁盘不可用时，仍然能保证数据的完整性。RAID 分为多种级别，提供了丰富的冗余性和性能的搭配方案选择。本题中，我们主要研究一种十分常见的 RAID 级别——RAID5。

RAID5 基本算法

RAID5 可以提供一块硬盘的冗余度，即阵列中至多允许有一块硬盘故障而不丢失数据。RAID5 的基本原理是异或运算（$\oplus$）。考虑数

$$a_1, a_2, \ldots, a_n$$

今令

$$p = a_1 \oplus a_2 \oplus \cdots \oplus a_n = \bigoplus_{i=1}^{n} a_i$$

易知

$$a_k = p \oplus \bigoplus_{i=1\ldots n, I \neq k} a_i$$

上式意味着，在 p 与 $a_1 \ldots a_n$ 这 $(n+1)$ 个数中，由任意 n 个可以推知其余的一个，这便是 RAID5 的基本原理。

由此，一种朴素的 $(n+1)$ 块盘的存储方案是：将数据分块存放在前 n 块盘中，然后在第 $(n+1)$ 块盘中存储前 n 块盘上相应位置处数据的异或结果。这种方案的确可以实现 1 块硬盘的冗余度，但是很显然，如果所有的硬盘都没有发生故障，当读取数据时，最后一块盘完全不会被利用起来，在性能上较为浪费。因此现行 RAID5 的存储方式采取了条带（stripe）化的方式，将被存储的数据均匀分布在所有磁盘上，从而提高了读写的性能。

硬盘及其组织

现代的硬盘可以被看作是一种能按块随机读写的持久化存储装置。大多数硬盘每块的大小是 512 字节或 4096 字节，硬盘上所有的块，从 0 开始连续编号。每次对硬盘的读写，都应该以块为单位，在读写时，传入被读写的块的编号，一次读写一整个块。

RAID 装置，可以将多个硬盘组织成一定的存储结构。在上层软件看起来，被组织后的存储结构好像一整块大硬盘。因此上层软件向 RAID 发送的读写指令里面的块的编号，应当被 RAID 装置进行适当地转换，反映到对具体某个硬盘或某几个硬盘的操作上。

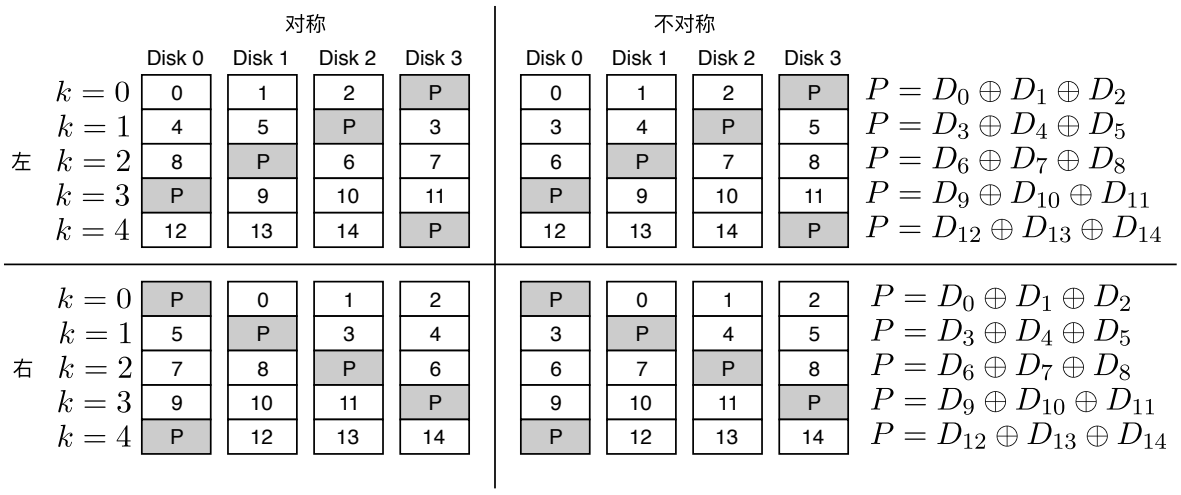
条带化与数据布局

在 RAID5 中，一个重要的参数是条带大小。**条带**是指连续、等大的数据块，是 RAID5 进行数据布局的基本单元。例如，假设某硬盘有 1024 个块，条带大小是 8 个块，那么这个硬盘一共被分为 128 个条带，编号在 $[0,8)$ 的块在第一个条带上，编号 $[8,16)$ 的块在第二个条带上，依次类推。一般地，如果将条带也从 0 开始编号，且条带大小为 s 个块，那么编号为 b 的块所在的条带编号是 $\lfloor \frac{b}{s} \rfloor$。

对于有 $(n+1)$ 块硬盘的 RAID5 存储，我们利用每块硬盘上编号为 k 的条带，存储编号为 $[kn,(k+1)n)$ 的条带（共 n 个）。这 n 个条带具体的存储方法是：先按一定规则，选中某个硬盘作为校验盘，存储这 n 个条带的异或和，然后再按一定顺序，将这 n 个条带存入其余的硬盘中。选中校验盘的规则有两种：左和右；存入数据条带的顺序也有两种：对称和不对称。这样，条带的布局算法共有四种：左对称（Left-symmetric）、左不对称（Left-asymmetric）、右对称（Right-symmetric）、右不对称（Right-asymmetric）。

左是指随着 k 递增，从最后一块盘开始，依次递减地选取校验盘，直到第一块盘为止，然后重新从最后一块盘开始；**右**是指随着 k 递增，从第一块盘开始，依次递增地选取校验盘，直到最后一块盘为止，然后重新从第一块盘开始。**不对称**是指，除去被选中的校验盘，从第一块盘开始，依次存入 n 个数据条带，直到最后一块盘为止；**对称**是指，从被选中的校验盘的下一块盘开始，依次存入数据条带，直到最后一块盘为止，然后从第一块盘开始，依次存入其余的数据条带，直到被选中的校验盘的上一块盘为止。

我们以 $n=3$ 为例，示意性地给出左对称、左不对称、右对称、右不对称这四种布局算法下，条带的布局情况。



【题目描述】

传说中，每个成功的运维背后，都坏着一打 RAID 阵列。有一天，汉东省政法大学丁香学生公寓楼下的外卖又被偷了！正在调取监控录像的关键时刻，汉东省政法大学监控中心的 RAID5 阵列发生了故障！老师拿来了一摞同一个型号的“圆邪”牌硬盘，找到了你。作为成功的运维，你能不能帮忙恢复里面的数据呢？

该型号硬盘的块大小是 8 字节，每个硬盘的最后一个块中存储有描述这个硬盘所在的 RAID5 阵列的信息。这个块的前三个字节依次为 `0x00 0xBE 0xEF`，接下来是一个 8 位无符号整数，表示该硬盘在阵列中的顺序号，接下来是一个 32 位大端序存储的无符号整数，表示该硬盘所属阵列的唯一标识符。这个整数的高 8 位表示这个阵列的条带大小（单位：块），接下来的 8 位表示这个阵列的成员硬盘数目，接下来的一位表示布局算法是左（`0`）还是右（`1`），接下来的一位表示布局算法是对称（`0`）还是不对称（`1`），最后的 14 位是用于区别不同阵列的标识数据：同一个阵列中的所有硬盘，这 32 位的唯一标识符相同。上述数据结构如下所示：

```
|     Byte 0    |     Byte 1    |     Byte 2    |     Byte 3    |
|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|7 6 5 4 3 2 1 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      0x00     |      0xBE     |      0xEF     |    sequence   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     strip     |     num of    |L|S|             id              | \
|     size      |     members   |R|A|                             |
   <- UUID
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ /
```

当你正打算撸起袖子加油码代码的时候，电话响了，老师焦急的声音从听筒里传来：“哎呀呀，我好像混进去了几块我从邮北人上下载电影用的硬盘了……”

【输入格式】

从标准输入读入数据。

输入的第一行包含一个正整数 n，表示得到的硬盘的数目。

接下来的一行包含一个非负整数 m，表示读取操作的数目。

接下来的 m 行，每行表示一个读取操作，包括空格分隔的一个字符串和一个非负整数。每行的开头是一个长度为 8 的字符串，该字符串仅包含数字 `0`～`9` 和大写字母 `A`～`F`，表示一个按大端序显示的 32 位无符号整数，指示要进行操作的 RAID5 阵列的唯一标识符。接下来是一个非负十进制整数，表示要读取的块的编号。

另外在程序运行的工作目录下有 n 个大小相同的只读的文件，保存了诸硬盘的原始二进制内容，分别命名为 `0.dsk`、`1.dsk`、…、$(n-1)$`.dsk`。各文件的大小（以字节为单位）是 8 的倍数。输入硬盘集合保证，除去混入的非阵列成员硬盘以外，向该硬盘集合补充若干适当内容的硬盘，即可使它们恰好组成若干合法的、数据没有差错的 RAID5 阵列。混入的非阵列成员硬盘，其最后一个块的前三个字节不符合上述数据结构。输入数据保证：若阵列的条带大小（以块为单位）大于 1，则它除每块硬盘的大小（以块为单位）必余 1。

【输出格式】

输出到标准输出。

输出的第一行包括空格分隔的两个非负整数，分别是总共找到的 RAID5 阵列的数目（包括可能损坏的），以及并非 RAID5 阵列的成员硬盘的数目。

接下来的输出有 m 行。

对于每一个读操作，产生一行输出。如果该读操作能进行，或能由现存的硬盘数据推算出来，则输出长度为 16 的字符串，该字符串仅包含数字 `0`～`9` 和大写字母 `A`～`F`，每两个字符构成一个 16 进制数字，表示读取到的数据块的内容。如果该读操作由于下列情形之一无法进行，则输出一个减号（`-`）：

- 阵列不完整，且被读取的块所在的硬盘缺失，且该数据无法由现存的硬盘数据推算出来；
- 指定的块超出阵列总长度；
- 不存在指定阵列。

【样例 1 输入】

```
3
4
01020001 0
01020001 1
01020001 2
01020003 2
End of Standard Input

hexdump of 0.dsk
00000000: 0001 0203 0405 0607 1011 1213 1415 1617
00000010: 00be ef00 0102 0001

hexdump of 1.dsk
00000000: 0001 0203 0405 0607 1011 1213 1415 1617
00000010: 00be ef01 0102 0001

hexdump of 2.dsk
00000000: a738 c8f2 f428 f3ce 1e19 33f3 87d0 89a1
00000010: 76c4 46e1 ca1b 5c55
```

【样例 1 输出】

```
1 1
0001020304050607
1011121314151617
-
-
```

【样例 1 解释】

由题意，给出的三块硬盘中，2.dsk 不是硬盘阵列的成员。其余两块盘组成了一个 RAID5 阵列，条带大小是 1 块（8 字节）长。注意到当 RAID5 的阵列中只有两块盘时，有 $p = a_1$，因此两块盘中数据是相同的，都是 RAID 阵列中的内容，因此可以任取一块盘进行读取操作。每个硬盘的最后一个块保存有阵列的信息，该信息并非 RAID 阵列可以读取的数据，因此这个阵列的有效长度为 2 块，所以第三个读取操作不能进行。第四个读取操作不能进行，是因为不存在指定的阵列。

【样例 2 输入】

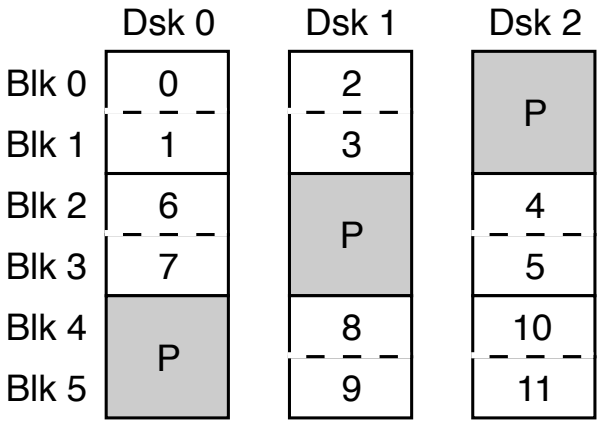
```
2
2
02030001 2
02030001 5
End of Standard Input

hexdump of 0.dsk
00000000: a0a1 a2a3 a4a5 a6a7 a8a9 aaab acad aeaf
00000010: b0b1 b2b3 b4b5 b6b7 b8b9 babb bcbd bebf
00000020: c0c1 c2c3 c4c5 c6c7 c8c9 cacb cccd cecf
00000030: 00be ef01 0203 0001

hexdump of 1.dsk
00000000: 0001 0203 0405 0607 0809 0a0b 0c0d 0e0f
00000010: 1011 1213 1415 1617 1819 1a1b 1c1d 1e1f
00000020: 2021 2223 2425 2627 2829 2a2b 2c2d 2e2f
00000030: 00be ef00 0203 0001
```

【样例 2 输出】

```
1 0
A0A1A2A3A4A5A6A7
A0A0A0A0A0A0A0A0
```

【样例 2 解释】

由题意，给出的两块硬盘中共含有一个 RAID5 阵列，该阵列由三块盘组成，条带大小是 2 块（16 字节）长，布局算法是左对称，并给出了 0 号、1 号盘的数据，缺失 2 号盘，因此整个 RAID5 阵列的布局情况如图所示。



图中，用虚线隔开的长方形表示一个块，连续的两个长方形组成的正方形表示一个条带。当读取编号为 2 的块时，该数据位于编号为 1 的盘的编号为 0 的块，因此结果是 `A0A1A2A3A4A5A6A7`；当读取编号为 5 的块时，该数据位于编号为 2 的盘的编号为 3 的块，该盘缺失，因此需要用其余两块盘相应位置处的数据进行异或运算得到 `18191A1B1C1D1E1F` $\oplus$ `B8B9BABBBCBDBEBF` $=$ `A0A0A0A0A0A0A0A0`。

【子任务】

测试点	n	m	是否存在缺失的硬盘	硬盘文件大小
1	$=10$	≤ 0	是	≤ 2 KB
2		$\leq 10^4$	否	$\leq 2,048$ KB
3,4,5,6	$=10^2$		是	≤ 8 KB
7,8,9,10				$\leq 8,192$ KB

【提示】

如何读取二进制文件

对于 C/C++ 语言，一种可行的方法是使用 `fopen`、`fread`、`ftell`、`fseek` 函数。你可以阅读这些函数的 man page。

对于 Java 语言，一种可行的方法是使用 `java.io.RandomAccessFile` 类，并调用其 `seek` 与 `read` 函数。

如何人工查看、修改二进制文件

请使用 `xxd` 命令查看二进制文件。上述命令的输出可以通过 `xxd -r` 命令转换回二进制文件。具体用法请参考该命令的 man page。

如何看到 man page

要查看 `foo` 的 man page，请执行命令 `man foo`。

特别提醒

请注意此题目的内存限制。

简化流操作（stream）

【题目背景】

MapReduce 最早是由 Google 公司研究提出的一种面向大规模数据处理的并行计算模型和方法，其核心操作是 `map()` 和 `reduce()`，概念借鉴于函数性编程语言。支持 MapReduce 的软件系统通常运行在由成百上千台服务器组成的分布式集群上，用于处理大规模数据的计算问题，例如 Google 搜索引擎中的排序、数据挖掘、机器学习等任务。

【题目描述】

在本题中，我们对 MapReduce 框架进行了相当的简化，只保留了几个简单的接口函数，且不涉及分布式存储和分布式计算的问题。你需要对接口函数进行具体实现。在评测时，会对这些接口函数进行调用，完成特定功能，进行正确性和性能测试，根据测试结果给分。

本题仅提供 C++/ Java 解题框架。

此外，为了让你更方便的理解相关知识与解决这一问题，我们还提供了一些相关的学习资料供你参考，其中既包括基本原理的阐述，也包括了一些相关的研究论文等等。我们并不保证这些资料对你的实现都有帮助。

任务目标

本次任务的目标为实现下列的接口，我们对于 C++/Java 分别给出实现说明。这些接口都是不可变（immutable）的，即所有返回 `Stream` 的接口都应该返回新的对象，而不应该改变被调用对象的状态。在 C++ 版本中，我们给函数添加了 `const` 限制符以保证你做到这一点；在 Java 版本中，请特别注意这一规定。我们将不变性的正确实现作为运行其他测试的前置条件；如果此特性实现不正确，则你无法获得任何分数。

C++ 接口说明 在 `Stream.hpp` 中，给出了 Stream 类的定义及成员函数，你需要在该文件中对成员函数进行实现，函数说明如下：

`Stream(const std::vector<T>& original_data)` `Stream` 类的构造函数。传入数据是装有类型 `T` 数据的 `std::vector`，生成一个基于类型 `T` 的 `Stream` 类对象。后续的操作即基于该 `vector` 中的数据进行。我们提供了一个该函数的简单实现。

注意：选手可根据需要对该函数的实现进行改动，也可在 `Stream` 类中增删成员变量和成员函数，只需保证已有的在 public 域中的函数可被调用且正确运行即可。

`Stream<M> map(std::function<M(const T&)> map_func) const;` `map` 函数的意义是将当前 `Stream` 对象基于的 `T` 数据依次变换为 M 类型的数据，并返回一个基

于 M 类型的数据的 Stream 对象。map 函数的参数也是一个函数，它的输入是一个 T 类型的数据，返回一个 M 类型的数据，它指明了变换操作的具体实现，是由调用接口的用户来编写的，换言之，你可以不用关心 map_func 的具体实现。

Stream<T> filter(std::function<bool(const T&)> filter_func) const; filter 函数的意义是对当前 Stream 对象基于的所有 T 类型的数据进行过滤，它的传入参数是函数 filter_func，返回一个基于所有通过过滤的 T 数据的 Stream 对象。filter_func 函数接受一个 T 类型的对象，并返回一个 bool 值，返回值为 false 的 T 数据将被从返回的 Stream 中移除。与 map_func 类似，filter_func 也是由调用接口的用户编写的。

std::vector<std::pair<K, Stream<V>>> group_by_key(std::function<K(const T&)> get_key, std::function<V(const T&)> get_value) const; group_by_key 函数的意义是将当前 Stream 对应的每个 T 数据根据不同的 key 进行分组。每个 T 对象对应的 key （类型为 K）和 value（类型为 V）可以通过用户传入的 get_key 和 get_value 函数获得。而后，你需要根据 key 对所有 (key, value) 二元组进行归类，将每个 key 对应的所有 value 对象转换为一个 Stream<V> 对象，并将所有这样的 std::pair<K, Stream<V>> 对象保存在一个 std::vector 中返回。

注意，我们保证传入的 K 类型重载了下列运算符：

- bool operator<(const K&, const K&)
- bool operator==(const K&, const K&)
- std::size_t std::hash<K>::operator()(const K&)

T reduce(T init, std::function<T(const T&, const T&)> combination_func) const; reduce 函数的意义是将当前 Stream 对象对应的 T 数据进行合并，最终得到一个 T 数据并返回。合并方式由传入函数 combination_func 决定，可以理解为这是一个二元运算，接受两个 T 类型的数据，返回一个。combination_func 是由调用接口的用户编写的。

我们保证传入的 combination_func 满足以下性质：

- 交换律：交换两个参数，结果不变
- 结合律：多次函数调用可以任意结合

std::vector<T> collect(); collect 函数的功能是，取出当前 Stream 中所有数据的值，并存储在 vector 中返回。我们不对 vector 中元素的顺序进行要求，但是你需要保证 Stream 和 vector 中的元素能一一对应。

Java 接口说明 接口的语义与 C++ 语言的基本一致，我们只对语法上的不同进行说明。因此，你应该先仔细阅读 C++ 版本的接口说明，以理解各接口的语义。

需要特别提醒的是，我们使用的 `List<T>` 是一个接口，并不能直接作为容器使用，`ArrayList` 和 `Vector` 等容器都实现了这一接口。

`public MyStream(List<T> data)` 构造函数。

`public <U> MyStream<U> map(Function<T, U> transformer)` 对每个元素进行转换。对于一个类型为 `T` 的对象 `t`，可以调用 `transformer.apply(t)` 获得对应的类型为 `U` 的结果。

`public MyStream<T> filter(Predicate<T> condition)` 对元素进行过滤。对于一个类型为 `T` 的对象 `t`，可以调用 `condition.test(t)` 获得一个布尔值表明它是否应该保留。

`public <K extends Comparable<K>, V> List<Pair<K, MyStream<V>>> groupByKey(Function<T, K> keyGetter, Function<T, V> valueGetter)` 对元素进行按 Key 分组。`Function` 对象的使用与上面类似。请注意，此时的 `Key` 类型保证重载了以下的接口：

- `bool equals(Object o)`
- `int hashCode()`
- `int compareTo(Key k)`

`public T reduce(T initial, BinaryOperator<T> reducer)` 对数据进行合并，你可以使用 `reducer.apply(t1, t2)` 得到两个元素的运算结果。同样，我们也保证运算的交换性和结合性。

`public List<T> collect()` 将 `MyStream` 中的数据存储在 `List` 中返回，同样我们不对元素的顺序提出要求。

测试框架 在 `main.cpp` / `Main.java` 中，提供了一个用上述接口进行单词计数任务的样例，对文本文件中出现的以字母a开头的词进行计数，同时提供了 `test.txt` 供你测试使用。你可以通过这个例子来了解用户是怎么调用这些接口的，从而更好地进行实现。你也可以自己编写代码对接口进行测试。

提交说明 你只需要提交 `Stream.hpp` / `MyStream.java`，评测系统会对其进行编译并调用接口运行。原则上你需要保证文件名称、需调用的类名称、所有 `public` 的函数签名与所提供的文件保持一致。你可以任意更改这两个文件，只要能够通过 OJ 的编译即可。

【评分方式】

本题每一个测试点独立评分，共有 4 个测试点，分别占 10、20、30、40 分。

每个测试点的分数分为两部分：正确性分数和性能分数。

正确性分数占 30% ，评测系统会调用你实现的接口函数，若能给出正确的结果，则得到全部正确性分数，否则整个测试点得 0 分。

性能分数占 70% ，以程序运行时间为基准计算，规则如下：

1. 运行时间仅包含接口函数运行时间与一些必要的其他操作，不含输入输出等时间
2. 每个分赛区单独评分，赛区用时最短者得到全部性能分。假设赛区最短用时为 t_0 ，则用时为 t 的选手在功能正确的情况下，将得到总性能分数的 t_0/t。

【提示】

1. 在提交的代码中禁止进行任何 IO 操作，否则将被判为违规，本题成绩无效。
2. 你可以使用多线程等手段来提高你的程序性能，如 pthread 和 OpenMP 等。我们提供的评测机基于虚拟化技术构建，有 4 个逻辑处理器，你可以参考这一数据。
3. 禁止对提供的代码框架进行任何形式的逆向工程或攻击，否则将被判定为违规，并取消全部成绩。