2019 CCF 大学生计算机系统与程序设计竞赛

时间：2019 年 10 月 16 日 09:00 ~ 21:00

目 录

A. 摘水果 (fruit)

【题目描述】

在我的后园，可以看见墙外有两株树 [1]，一株是苹果树，还有一株是梨树。收获的季节到了，两棵树上分别结了 n 个果子；苹果树的果子用 1 到 n 的整数编号，而梨树的果子用 $n+1$ 到 $2n$ 的整数编号。我们为每个果子评出了美味度，编号为 i 的果子美味度为 a_i。

对于编号为 i 的果子，我们给出 b_i，表示仅当 i 号果子被摘下后，b_i 号果子才能被摘下（如果 $b_i = 0$，表示这个果子不产生限制）。如果某个果子被多条限制所影响，那么必须等到所有的条件都满足时才能被摘下；而没受到限制的果子总能被摘下。另外，我们对所有 i 保证 $b_i < i$，并且要么 b_i 果子与 i 号果子在同一棵树上，要么 $b_i = 0$。这样，我们总有办法按照某种顺序摘下两棵树上的所有果子，不会发生由于限制而无法摘下某些果子的情况。

现在我们要将这些果子包装并出售：每次，我们都从苹果树与梨树上分别摘下一个果子，然后将其作为一包出售，其价值是两个果子的美味度的按位异或的结果。这个过程会重复 n 次，直到所有的果子都被摘完。每次，我们会按照以下标准选择摘取的果子：

- 需要优先最大化两个果子作为一包时的价值；
- 如果有多种方式使得价值最大，则要在此基础上最大化苹果的美味度；
- 如果仍有平局，则要在此基础上最大化苹果的编号；
- 如果仍有平局，则要在此基础上最大化梨的编号（此时不可能再出现平局了）。

你需要依次给出每一包果子的价值。

[1]: 鲁迅．野草［M］

【输入格式】

从标准输入读入数据。

第一行输入一个正整数 n。

第二行依次输入 $2n$ 个非负整数 $a_1, a_2, \ldots, a_{2n}$。

第三行依次输入 $2n$ 个非负整数 $b_1, b_2, \ldots, b_{2n}$。

同一行相邻的两个数之间用一个空格隔开。

【输出格式】

输出到标准输出。

输出一行，包含 n 个数，依次表示每一包果子的价值。

同一行相邻的两个数之间用一个空格隔开。

【样例输入】

```
3
3 9 4 1 3 4
0 0 2 0 4 4
```

【样例输出】

```
7 13 2
```

【样例解释】

仅有 3 号果子被摘下后，才能摘下 2 号果子；仅有 5 号果子与 6 号果子均被摘下后，才能摘下 4 号果子。

第 1 次，摘下了编号为 3、美味度为 4 的苹果与编号为 5、美味度为 3 的梨，得到了价值为 7（3 与 4 异或）的一包果子。注意，假设选取编号为 1 的苹果与编号为 6 的梨，虽然价值同样为 7，但由于苹果的美味度不如前述方案，所以不这样选择。

第 2 次，摘下了编号为 2、美味度为 9 的苹果与编号为 6、美味度为 4 的梨，得到了价值为 13（9 与 4 异或）的一包果子。

第 3 次，摘下了编号为 1、美味度为 3 的苹果与编号为 4、美味度为 1 的梨，得到了价值为 2（3 与 1 异或）的一包果子。

【子任务】

所有数据保证 $n \le 100$，$0 \le a_1, a_2, \ldots, a_{2n} \le 100$。

- （31 分）对 $i \ne n+1$ 的 i 满足 $b_i = i - 1$，而 $b_{n+1} = 0$，这意味着任意时刻任何一棵树只有一个可以摘的果子；
- （47 分）对任意的 i 满足 $b_i = 0$，这意味着所有果子都没有受到任何的限制；
- （22 分）没有特殊的限制。

B. 纸牌计数 (card)

我们有一副纸牌，它由 n 张牌组成，其中每张牌上标有一个数字（0 到 9）和一个大写字母（A 到 Z），例如 2C、1F。

我们如下定义一个字符串与一张牌之间的匹配关系：

- 字符串 ?? 与任何一张牌都匹配。
- 第一位为 ? 而第二位为字母的字符串，与任何一张标有该字母的牌匹配。例如，字符串 ?C 与任何标有 C 的牌匹配。
- 第一位为数字而第二位为字母的字符串，仅与内容完全一致的牌匹配。例如，字符串 0C 与内容为 0C 的牌匹配。
- 不会出现第一位为数字而第二位为 ? 的字符串。

我们称 m 个字符串 $S_1,\ldots,S_m$ 与 m 张牌 $C_1,\ldots,C_m$ 是匹配的，当且仅当：存在集合 $\{1,\ldots,m\}$ 上的一一映射 σ，使得对于任意 $1 \le i \le m$，S_i 与 $C_{\sigma(i)}$ 匹配。

例如，对于 ?? 和 ?C 这两个字符串，可以匹配内容为 1C 和 2C 的牌，因为字符串 ?? 与内容为 2C 的牌匹配且字符串 ?C 与内容为 1C 的牌匹配，而 ?? 和 ?C 这两个字符串不能匹配内容为 1S 和 1P 的牌。

现在，给定 m 个字符串，你需要求解：如果在 n 张牌中（无放回地）随机选取 m 张，有多大概率与这些字符串匹配？

【输入格式】

从标准输入读入数据。

每个输入文件包含多个输入数据。

第一行输入该文件的数据个数 T。

接下来依次输入每个数据。每个数据包含三行，其中：

- 第一行输入两个正整数 n 和 m；
- 第二行输入以空格分隔的 n 个字符串，表示每张纸牌的内容（每个字符串第一位为数字，第二位为大写字母）；
- 第三行输入以空格分隔的 m 个字符串，表示每个需要匹配的字符串（每个字符串第一位为数字，第二位为大写字母；或第一位为 ?，第二位为大写字母；或为 ??）。

【输出格式】

输出到标准输出。

对于每个输入数据，输出一行，表示所求的概率。概率需要以最简分数 u/v 的形式输出，其中 $0 \le u \le v$ 且 u,v 为互质的整数。

特殊地，对于 0 请输出 0/1，对于 1 请输出 1/1。

【样例】

见题目目录下的 ***1.in*** 与 ***1.ans***。

【子任务】

对于所有的数据，$1 \le m \le n \le 60$；$T \le 1000$。

- （11 分）$m = 1$；
- （23 分）$m = n$；
- （27 分）$n = 30$ 且所有的牌为 `0C 1C 2C 3C 4C 5C 6C 7C 8C 9C 0S 1S 2S 3S 4S 5S 6S 7S 8S 9S 0P 1P 2P 3P 4P 5P 6P 7P 8P 9P`；
- （22 分）$n = 40$ 且所有的牌为 `0C 1C 2C 3C 4C 5C 6C 7C 8C 9C 0C 1C 2C 3C 4C 5C 6C 7C 8C 9C 0S 1S 2S 3S 4S 5S 6S 7S 8S 9S 0P 1P 2P 3P 4P 5P 6P 7P 8P 9P`；
- （17 分）没有特殊的限制。

【提示】

对于分数 a/b，设 $g = \gcd(a, b)$，那么其最简分数形式为 $(a/g)/(b/g)$。

$\gcd(a, b)$ 可以这样计算：如果 $a = 0$ 则答案为 b，否则为 $\gcd(b \bmod a, a)$（需要递归计算）。

C. SQL 查询 (sql)

【题目描述】

顿顿想要建立一个简单的教务管理数据库，用于存储学生的考试成绩并支持一些基本的查询操作。

数据格式

该数据库包含以下五张数据表：

`Student(sid, dept, age)`

学生信息表：`sid` 为主键，表示学生 ID；`dept` 表示学生所在院系名称；`age` 表示学生的年龄。

`Course(cid, name)`

课程信息表：`cid` 为主键，表示课程 ID；`name` 表示课程名称。

`Teacher(tid, dept, age)`

教师信息表：`tid` 为主键，表示教师 ID；`dept` 表示教师所在院系名称；`age` 表示教师的年龄。

`Grade(sid, cid, score)`

成绩信息表：`sid` 和 `cid` 分别来自 `Student` 表和 `Course` 表的主键（即每条记录中的 `sid` 和 `cid` 一定存在于相应的表中，下同），二者一起作为该表的主键；`score` 表示该学生这门课程的成绩。

`Teach(cid, tid)`

授课信息表：`cid` 和 `tid` 分别来自 `Course` 表和 `Teacher` 表的主键，二者一起作为该表的主键；需要注意的是，一门课程可以有多个老师授课。

注：主键（**Primary Key**）可以唯一标识表中的每条记录，换言之在一张表中所有记录的主键互不相同。

在上述表中，`age` 和 `score` 是 $[0, 100]$ 的整数，其余都是长度小于等于 10 的非空字符串。其中三个主键 `sid`、`cid` 和 `tid` 由数字 `0...9` 组成，而 `name` 和 `dept` 则由大小写字母组成。

查询语句

该数据库支持使用一种简单的 SELECT 语句进行查询，文法定义如下所述。最终所有的查询语句都可以由 QUERY 符号推导而来。

在所有定义中，::= 表示其左侧的符号可以推导为右侧的符号串，其中带单引号的符号表示此符号为固定的字符串，不带引号的则表示符号还需要进行进一步推导。[] 表示文法中的可选内容。如果一个符号存在多种推导方式 A ::= B 和 A ::= C，则可以简写为 A ::= B | C。

```
QUERY ::= 'SELECT' '*' 'FROM' TABLES ['WHERE' EXPR]
QUERY ::= 'SELECT' COLUMNS 'FROM' TABLES ['WHERE' EXPR]
```

从指定的表（TABLES）中查询满足筛选条件（EXPR）的记录，并按照指定的列（COLUMNS）输出。如果没有给出筛选条件，则返回所有的记录。

```
TABLES ::= TABLE_NAME | TABLE_NAME ',' TABLE_NAME
TABLE_NAME ::= 'Student' | 'Grade' | 'Course' | 'Teach' | 'Teacher'
```

TABLES 可以包含多张表，此时新表中的列是这些表中列的笛卡尔积，一个例子如下所示。

```
+---------+ +---------+ +------------+ +------------+
| Table A | | Table B | | Table A, B | | Table B, A |
+---------+ +---------+ +------------+ +------------+
| a1      | | b1      | | a1, b1     | | b1, a1     |
| a2      | | b2      | | a1, b2     | | b1, a2     |
+---------+ | b3      | | a1, b3     | | b2, a1     |
            +---------+ | a2, b1     | | b2, a2     |
                        | a2, b2     | | b3, a1     |
                        | a2, b3     | | b3, a2     |
                        +------------+ +------------+
```

本题中约定 TABLES 仅会包含一张或两张不同的表。

```
COLUMNS ::= COLUMN | COLUMNS ',' COLUMN
COLUMN ::= [TABLE_NAME '.'] COLUMN_NAME
COLUMN_NAME ::= 'sid' | 'cid' | 'tid' | 'age' | 'score' | 'dept' | 'name'
```

COLUMNS 也是递归定义，即由逗号分隔的至少一个 COLUMN 组成。在无歧义时（即 TABLES 中只有唯一的表含有该列），COLUMN 中的 TABLE_NAME 是可选的。SELECT age

`FROM Student, Teacher` 是一种典型的歧义句式，因为无法确定其中想要选取的 `age` 是学生还是教师的年龄。

虽然在文法上一个 `COLUMN` 可以由任意 `TABLE_NAME` 和 `COLUMN_NAME` 组合而成，但在实际处理时，这两者需要符合上面定义的表结构。比如 `Course.sid` 就是一种错误的写法，因为 `Course` 表中并没有 `sid` 这一列。

```
EXPR ::= COND | EXPR 'AND' COND
COND ::= COLUMN CMP CONSTANT | COLUMN '=' COLUMN
CMP ::= '>' | '=' | '<'
```

其中：

- `EXPR` 是一个合取布尔表达式，由若干个**比较条件**（`COND`）用 `AND` 连接而成，表示**筛选条件**（即满足该条件的**记录**才会被输出）。
- `CONSTANT` 没有文法定义，它表示一个**常量**，且其类型和数据范围与相比较的 `COLUMN` 一致；如果是字符串类型，还需用双引号括起（例如 `"Sam"`）。

这里我们额外规定：

- **列**与**常量**进行比较时（`COLUMN CMP CONSTANT`），整数类型（`age`、`score`）可以进行大于、小于和等于三种判断，字符串类型只能做等于判断；
- 两**列**进行比较时（`COLUMN = COLUMN`），它们需来自不同的**表**且二者的**列名**（`COLUMN_NAME`）必须相同；
- 在一个 `EXPR` 中，`COLUMN CMP CONSTANT` 类型的 `COND` 个数不限，但 `COLUMN = COLUMN` 最多只能出现一次。

查询语句大小写敏感，全部的**保留字**包括：

```
SELECT, *, FROM, WHERE, AND, Student, Course, Teacher, Grade, Teach
sid, cid, tid, dept, name, age, score
```

全部的**分隔符**包括：`,`, `.`, `>`, `=`, `<`, （空格）。

双引号（`"`）应视作字符串**常量**的一部分而非**保留字**或**分隔符**。

每个**保留字**和**常量**（`CONSTANT`）可视作查询语句中的一个**基本单位**，在格式上我们约定：

- 每条查询语句占一行；
- **基本单位**不可分割；
- 为避免歧义，两个**基本单位**之间应至少有一个**分隔符**，并且允许存在任意多的空格；
- 一行总长度不超过 100 个字符。

结果输出

对每条查询输出一张表，列由 COLUMNS 指定，每行为一条满足筛选条件（EXPR）的记录（不同的列间用一个空格分隔）。这里只需要输出所有满足条件的记录，不需要打印表头；查询结果为空则不输出；字符串类型无需输出双引号。若想输出所有的列，可以用 * 代替 COLUMNS；若 TABLES 中只有一张表，此时应输出该表的全部列；如果有第二张表，应输出第一张表的全部列与二张表的全部列的笛卡尔积；表内部的列按照上文定义时给出的顺序排序。

如果 TABLES 仅含一张表，则按该表的顺序依次输出符合条件的记录。若 TABLES 包含两张表（形如 A,B），则先考虑 A 的顺序，再考虑 B 的顺序（参考前文中两张表做笛卡尔积的例子）。

【输入格式】

从标准输入读入数据。

首先依次输入五张表 Student、Course、Teacher、Grade 和 Teach 的数据。

对于第 i 张表，第一行输入一个正整数 N_i，表示该表中记录的个数（亦即行数）。接下来 N_i 行，每行输入一条记录，其中不同列之间用一个空格分隔且字符串字段不会用双引号括起。

然后输入一个正整数 M，表示需要处理的查询语句个数。最后 M 行每行输入一条查询语句，保证每条查询语句均符合上述所有要求。

【输出格式】

输出到标准输出。

按要求输出每条查询语句的结果。

【样例 1 输入】

```
3
2019001 law 19
2019002 info 20
2019003 info 19
2
20190001 math
20190002 English
2
2019101 info 49
2019102 info 38
```

```
2
2019002 20190001 100
2019003 20190002 0
1
20190001 2019102
4
SELECT * FROM Student
SELECT Student . dept FROM Student
SELECT Student.sid,Grade.sid,dept,score FROM Student,Grade
SELECT * FROM Student, Grade WHERE Grade .sid=Student. sid
```

【样例 1 输出】

```
2019001 law 19
2019002 info 20
2019003 info 19
law
info
info
2019001 2019002 law 100
2019001 2019003 law 0
2019002 2019002 info 100
2019002 2019003 info 0
2019003 2019002 info 100
2019003 2019003 info 0
2019002 info 20 2019002 20190001 100
2019003 info 19 2019003 20190002 0
```

【样例 1 解释】

第 1 – 3 行、4 – 6 行、7 – 12 行和 13 – 14 行依次对应四条查询的结果。

【样例 2】

见题目目录下的 ***2.in*** 与 ***2.ans***。

【样例 2 解释】

第 1 – 2 行、3 – 4 行、5 – 6 行、7 – 14 行、15 – 16 行和 17 – 18 行依次对应前六条查询的结果，最后一条查询结果为空因为不存在 ID 为 `2019` 的课程。

【子任务】

本题共有六个子任务，每个子任务有一个或多个测试点。若你的程序对一个子任务的**全部**测试点，都能给出正确的输出，则得该子任务的满分，否则该子任务得 0 分。本题满分共计 100 分。

- 子任务一（14 分）：所有测试点保证每条查询仅涉及**一张表**，且 $N_i \leq 100$、$M \leq 1$，查询语句形如 `SELECT * FROM TABLE_NAME`；
- 子任务二（16 分）：所有测试点保证每条查询仅涉及**一张表**，且 $N_i \leq 100$、$M \leq 1$，查询语句形如 `SELECT * FROM TABLE_NAME WHERE EXPR`；
- 子任务三（19 分）：所有测试点保证每条查询仅涉及**一张表**，且 $N_i \leq 100$、$M \leq 1$；
- 子任务四（23 分）：所有测试点保证每条查询最多涉及**两张表**，每个 expr 中**均不存在** `COLUMN = COLUMN` 类型的 `COND`，且 $N_i \leq 10^4$、$M \leq 10$；
- 子任务五（15 分）：所有测试点保证每条查询最多涉及**两张表**，每个 expr 中**最多一个** `COLUMN = COLUMN` 类型的 `COND`，且 $N_i \leq 100$、$M \leq 10$；
- 子任务六（13 分）：所有测试点保证每条查询最多涉及**两张表**，每个 expr 中**最多一个** `COLUMN = COLUMN` 类型的 `COND`，且 $N_i \leq 10^4$、$M \leq 10$。

此外，所有测试数据保证每条查询的结果均不超过 10^4 行。

【提示】

在下发的文件中，我们还为选手提供了一些与真实测试数据生成方式相同的的输入文件，其中涵盖了实际测试中用到的**所有**查询语句。

D. 调度器 (scheduler)

【题目背景】

调度器是操作系统的一部分，它决定计算机何时运行什么任务。通常，调度器能够暂停一个运行中的任务，将它放回到等待队列当中，并运行一个新任务，这一机制称为抢占（preemption）。抢占的实现往往需要通过硬件时钟（timer）定时发起中断（interrupt）信号，告知调度器一定时间周期已经过去，并由调度器决定下一个运行的任务。

调度器可能会针对不同的目标设计，例如：吞吐率最大化、响应时间最小化、延迟最小化或公平性最大化。在实践中，这些目标通常存在冲突；因此，调度器会实现一个权衡利弊的折中方案，根据用户的需求和目的，侧重以上一个或多个方面。

在实时（realtime）环境，例如工业上用于自动控制（如机器人）的嵌入式系统，调度器必须保证进程的调度不能超过最后期限——这是保持系统稳定运行的关键因素。

【题目描述】

要求

本题要求设计一个面向单核的调度策略，并实现调度器对应接口。本题要求调度策略尽可能达到实时系统的要求（即所谓“准实时”调度），并根据接口实现的正确性与任务及时完成率进行评分。本题根据选手全场最优成绩评分，选手可通过实时提交评估调度策略性能。

任务假设

- 任务包含完成截止时间（deadline），调度器应尽可能在截止时间前完成任务。
- 任务分为高优先级与低优先级，调度器应倾向优先完成高优先级任务。
- 每个任务包括一段以上 CPU 计算与零或多段 IO 操作，分别使用计算机的 CPU 资源和 IO 资源。调度器可以执行任务 T_{cpu} 的 CPU 计算时，并行执行任务 T_{io} 的 IO 操作。
- 在一个任务使用 IO 资源时，不允许其同时使用 CPU 资源。
- 每个任务都以 CPU 计算开始与结束。

调度规则

使用系统资源：任何时刻，最多只有一个任务 T_{cpu} 使用系统 CPU 资源进行计算，最多只有一个任务 T_{io} 使用系统 IO 资源进行操作。系统 CPU 资源可以在任意时刻被

调度器切换并执行新任务的 CPU 计算；系统 IO 资源必须完成当前 IO 操作后，才能执行新任务的 IO 操作。

调度新任务：调度器在新任务到达、任务结束、任务请求 IO 操作、任务结束 IO 操作、时钟中断到来时被唤醒并收到通知，并调用选手实现的策略接口决定接下来被调度的任务。策略将输出任务 T'_{cpu} 以抢占当前 CPU 资源。T'_{cpu} 可以等于 T_{cpu}，即继续将 CPU 资源分配给旧任务；T'_{cpu} 可以为空，即将 CPU 资源空置。当不存在能够进行 CPU 计算的任务时，空置 CPU 资源是合理的。当 IO 资源空闲时，策略可输出任务 T'_{io} 以使 IO 资源服务新的任务。注意，因为 IO 资源无法实时切换，当旧任务 T_{io} 未完成时，不能开始新的 IO 操作。

【子任务】

本题共 16 个测试点：

1. 测试点 1、2、3：任务特性较为相似，截止时间较为宽裕，优先级随机分布，任务随机出现；
2. 测试点 4、5、6：在第 1 条基础上，任务特征差异较大；
3. 测试点 7、8：在第 2 条基础上，任务特征分布随时间变化；
4. 测试点 9、10：在第 2 条基础上，截止时间较为紧张；
5. 测试点 11、12：在第 2 条基础上，具有相对紧张截止时间的任务倾向于拥有高优先级；
6. 测试点 13、14：在第 2 条基础上，任务在特定时刻会更加频繁出现；
7. 测试点 15、16：在第 2 条基础上，任务特征分布随时间变化，截止时间较为紧张，具有相对紧张截止时间的任务倾向于拥有高优先级，任务在特定时刻会更加频繁出现。

【答题接口】

本题要求选手实现调度策略 `policy` 接口，此函数将在上述描述的事件发生时被调用，此函数的输出将决定调度器接下来的操作。此处将描述该函数输入、输出参数语义，编程语言相关的细节将在后文具体描述。

`policy` 函数接收三个参数：

第一个参数为**事件列表**，包含此时刻同时发生的所有事件信息，绝大部分时候此列表长度为 1。事件包括下列类型：

- 时钟中断到来（`Timer`）；
- 新任务到达（`TaskArrival`），表示一个用户发起了新任务；
- 任务请求 IO 操作（`IoRequest`），表示一个任务**需要**进行 IO 操作；
- 任务结束 IO 操作（`IoEnd`），表示一个任务完成了一次 IO 操作，并**需要**使用 CPU 资源；

- 任务完成（TaskFinish），表示一个任务完成了它所有的 CPU 和 IO 操作。

除事件类型、时间外，与任务相关的事件信息还有相应的任务信息，包括任务 ID、到达时间、截止时间与优先级。

第二、三个参数为此时刻 CPU 与 IO 分别服务的任务的任务 ID。注意，当调度策略输出非法操作指令以及任务当前 CPU、IO 操作完成时，这两个参数可能会与上次 policy 指定的任务 ID 不同。

policy 函数输出调度策略的操作指令，即 CPU、IO 资源接下来分别服务任务的任务 ID。

根据调度规则，在 IO 资源未空闲时指定其他任务 ID 是非法的；在一个任务占用 IO 资源时，试图让其占用 CPU 资源也是非法的。进行任何非法操作会导致你在该测试点获得 0 分。

任务 ID 值的说明

所有任务的任务 ID 大于 0。policy 函数输入参数中为 0 的任务 ID 代表对应资源空闲；输出操作指令中为 0 的 CPU 资源任务 ID 代表不使用 CPU 资源（即到下一个事件来临之前，CPU 将处于空闲状态），为 0 的 IO 资源任务 ID 代表不进行 IO 资源调度（即到下一个事件来临之前，IO 将保持当前状态）。

C++ 接口

C++ 接口定义以下结构、函数。

```
struct Event {
  enum class Type {
      kTimer, kTaskArrival, kTaskFinish, kIoBegin, kIoEnd
  };
  struct Task {
    enum class Priority { kHigh, kLow };

    int arrivalTime;
    int deadline;
    Priority priority;
    int taskId;
  };

  Type type;
  int time;
```

```
  Task task;
};

struct Action {
  int cpuTask, ioTask;
};

Action policy(const std::vector<Event>& events, int currentCpuTask,
              int currentIoTask);
```

请在相应文件中实现 policy 函数。其他辅助性结构、函数可定义、实现在同一文件中。本文件将使用 C++ 17 语言标准进行编译。

Java 接口

Java 接口定义以下结构、函数。

```
class Event {
    Task task;
    int time;
    EventType type;
}

class Task {
    int arrivalTime;
    int deadline;
    PriorityType priority;
    int taskId;
}

enum EventType {
    Timer,
    TaskArrival,
    TaskFinish,
    IoRequest,
    IoEnd;
}
```

```
enum PriorityType {
    High,
    Low;
}

class Action {
    int cpuTask;
    int ioTask;
}

Action policy(List<Event> events,
              int currentCpuTask, int currentIoTask);
```

请在相应文件中实现 `policy` 函数。其他辅助性类、函数可定义、实现在同一文件中。本文件将使用 Java 11 语言标准进行编译、运行。

Python 接口

Python 接口定义以下函数。

```
def policy(events, currentCpuTask, currentIoTask)
```

其中 `events` 是一组字典的列表，每个字典对象对应一项事件，示例如下：

```
{
    "task":
    {
        "arrivalTime": 1304,
        "deadline": 1856,
        "priority": "High",
        "taskId": 37
    },
    "time": 1679,
    "type":"TaskFinish"
}
```

1. `type` 字段表示事件的类型，其有 5 种枚举值 `"Timer"`，`"TaskArrival"`，`"TaskFinish"`，`"IoRequest"`，`"IoEnd"`；
2. `time` 字段表示事件触发的时间；

3. `task` 字段对应一项任务：
    1. `arrivalTime` 字段，表示此任务的开始时间；
    2. `deadline` 字段，表示此任务的最后期限；
    3. `priority` 字段，表示任务的优先级，2 种枚举值 `"High"`，`"Low"`；
    4. `taskId` 字段，表示任务的全局识别符。

你需要返回一个字典作为策略的结果，定义如下：

```
{
    "cpuTask": 0,
    "ioTask": 0
}
```

请在相应文件中实现 `policy` 函数。其他辅助性函数可定义、实现在同一文件中。本文件将使用 Python 3.6.8 运行。

【评分方式】

正确性

考虑以下 FIFO 调度策略：维护任务队列 Q，当任务到达时将任务加入 Q，当且仅当上一任务完成时推出队首任务并开始执行。令 FIFO 调度下所有任务完成所需时间为 t_{FIFO}，当选手实现策略完成所有任务用时 t 小于等于 t_{FIFO} 时，认为选手策略实现正确。

性能指标：综合完成率

综合完成率 r 为高、低优先级任务在截止时间内及时完成率的加权平均。假设高优先级任务完成率为 r_{hi}，低优先级任务完成率为 r_{lo}，则综合完成率 r 满足：

$$r = 70\% \times r_{hi} + 30\% \times r_{lo}$$

总分

总分 100 分，其中 16 分为正确性得分：总共 16 输入文件，每个文件为 1 分。84 分为性能评比得分：每个测试点满分为 5.25 分，只有当选手得到正确性得分后才能得到性能分。设 $r_{\max}$ 为所有选手本测试点的 r 的最大值，该测试点的性能得分为：

$$5.25 \times \frac{r}{r_{\max}}$$

【提示】

本题通过模拟方式评估选手调度策略性能，请选手不要以现实世界的时间单位来认知事件的时间（`time`）属性。

模拟过程中的时钟中断间隔长度固定，且大于 1。

E. 评测鱼 (risc-v)

【题目背景】

评测鸭是为程序设计竞赛和训练打造的稳定精确评测系统，它的小伙伴有评测鸽、评测猫，还有评测鱼。

目前通行的处理器架构设计模式共有两种，分别为 RISC（Reduced Instruction Set Computer）与 CISC（Complex Instruction Set Computer）。相比于 CISC，RISC 对指令数目和寻址方式都做了精简，使其实现更容易，并具有更好的指令并行执行程度，相应编译器的效率也较高。RISC-V 是由加州大学伯克利分校设计的一套 RISC 的开源指令集架构。

现在我们有一台基于 RISC-V 架构的计算机，希望你能够在这台计算机上编写特定算法的代码，并取得最高的执行效率。

【题目描述】

处理器架构

本题的目标平台为一种 `RV32IM` 处理器，这意味着其指令长度和字长均为 32 位，并且实现了 `RV32I` 基本整数指令集和 `RV32M` 标准整数乘除法扩展指令集。关于这一指令集的更详细描述，包括指令格式和详细定义，可以参见下发的 `riscv-spec.pdf`。

在本题中的处理器具体架构为 **5 级流水、顺序单发射、无分支预测**，说明如下：

- 5 级流水：处理器上的每条指令执行都需要经历 5 个阶段：取指 (IF)、译码 (ID)、执行 (EX)、访存 (MEM)、写回 (WB)，指令流经每个阶段均**固定花费 1 个时钟周期**。
- 顺序单发射：处理器的取指译码部件只有一套，即每个时钟周期**有且仅有 1 条指令**（包括插入的气泡）进入流水线开始执行。并且指令的执行顺序和发射顺序严格按照其在存储器中的顺序。
- 无分支预测：对于条件指令，处理器不进行分支预测。一旦控制流发生转移，则流水线中尚未执行的指令会被刷新。

流水线中的指令执行过程

本题使用的 `RV32IM` 指令集中的指令可分为以下四类：

- 整数运算指令：在寄存器之间，或者寄存器和立即数之间进行算术或逻辑运算。包括算术、逻辑、比较、移位指令。
- 加载存储指令：在内存和寄存器间传输数据的访存指令。包括取数和存数指令。
- 控制转移指令：无条件跳转指令和条件跳转指令。

- 环境调用指令：系统环境调用指令 `ECALL`，本题进行输入、输出、打印等操作时需要使用。

四类指令在流水线各阶段的执行内容如下表所示：

	取指 (IF)	译码 (ID)	执行 (EX)	访存 (MEM)	写回 (WB)
整数运算指令	取指令	指令译码，并访问所需寄存器取出操作数	对操作数进行运算	无操作	将计算结果写回寄存器
加载存储指令	取指令	指令译码，并访问所需寄存器取出操作数	计算访存地址	访问内存进行加载或存储数据操作	对于加载指令，将计算结果写回寄存器
控制转移指令	取指令	指令译码，访问所需寄存器取出操作数，计算转移地址和判断转移条件	无操作	无操作	无操作
环境调用指令	取指令	指令译码，并访问所需寄存器取出操作数	进行环境调用	无操作	无操作

从流水线的执行过程可以发现，本处理器执行指令过程中会遇到**数据冲突**和**条件冲突**：

- **数据冲突**：邻近的指令对同一个寄存器进行操作，之后的指令要用到之前指令的结果。
- **控制冲突**：控制流转移成功时，流水线会排空。

当邻近的指令为整数运算指令时，本处理器使用了数据前递技术 (也叫旁路、定向) 以避免这些指令之间发生数据冲突。但遇到其他数据冲突和控制冲突时，处理器会向流水线中插入气泡（即空指令）。所以除了算法本身的效率外，你需要尽可能减少冲突，以取得最高的执行效率。

CPU 模拟器

为了让你更直观地了解给定的架构，也为了方便你进行针对性的设计和优化，我们提供了模拟器 `Ripes`，分为图形界面和命令行两个执行方式：

- 图形界面（`RipesGUI`）可以直接加载文本或者二进制格式的汇编指令运行，并以直观的方式追踪流水线的运行状态。请注意，它**无法加载** ELF 文件，也**无法**针对特定的题目进行评测或调试，只用于方便选手熟悉处理器的工作原理。
- 命令行（`RipesRunner`）可以加载符合特定标准的 ELF 文件执行，并统计精确的执行周期数。这也是评测时**唯一参考**的性能指标。

Ripes 的图形界面中包括以下选项卡：

- Editor 用于编辑源代码和查看二进制代码。首先通过 ***File->Load Example->assembly*** 选择一个示例。在 **Editor** 选项卡中，您现在应该会在左侧文本框中看到一些 RISC-V 代码，右侧文本框可以选择显示左侧文本框汇编后的二进制代码 (***Binary***) 或二进制代码的反汇编版本 (***Disassembled***)，在两个文本框间可以点击代码行设置断点。

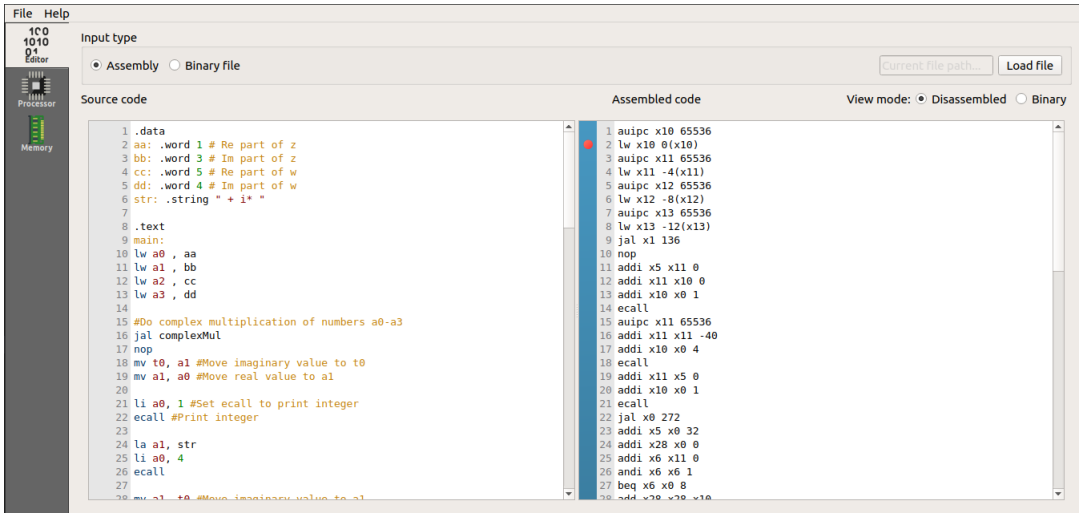


图 1: Editor

- Processor 中您将看到典型的 5 级 RISC-V 流水线体系结构图，为减少混乱，控制器和控制信号没有在视图中显示。如果您的汇编代码中没有语法错误，它将自动被汇编并加载到模拟器中。除体系结构图外，该选项卡包含以下元素：

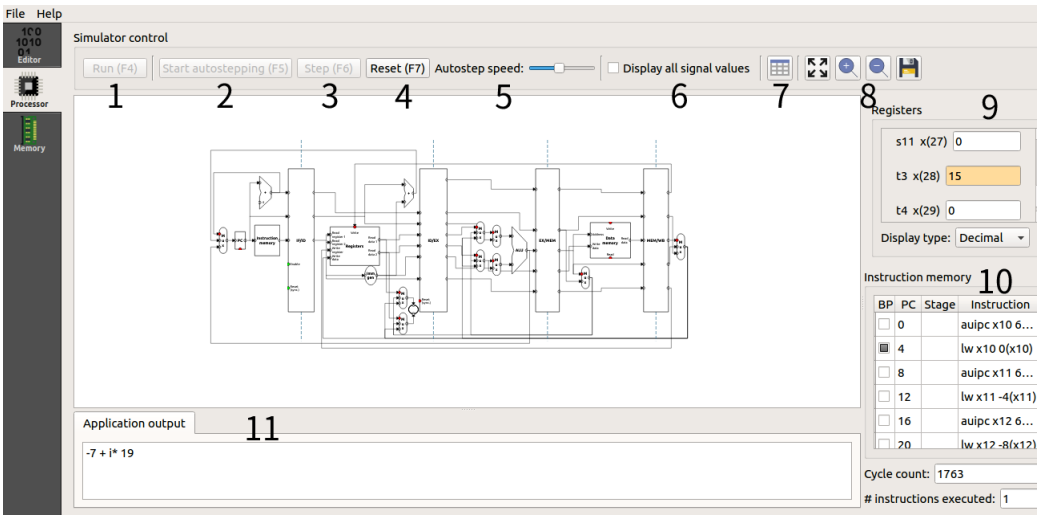
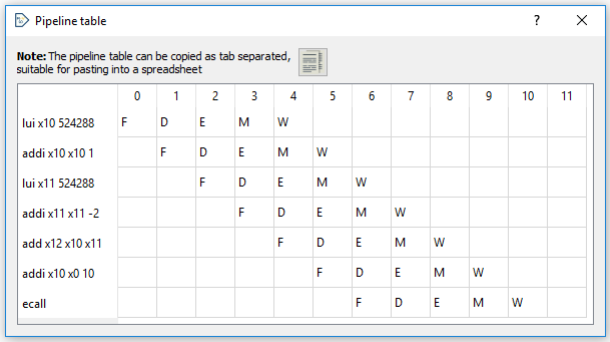


图 2: Processor

– 模拟器控制：

    - (1) **Run**：运行程序，直到遇到断点或执行完毕。
    - (2) **Autostepping**：以（5）中滑块的速度，按时钟周期执行程序。
    - (3) **Step**：激活电路中的时钟信号，执行一个时钟周期。
    - (4) **Reset**：重置模拟器，重置所有阶段并将程序计数器设置为指向第一条指令。
  - **(6) Display all signal values**：显示图中大多数部件的输出信号。
  - **(7) Pipeline table**：弹出窗口，显示当前程序运行的流水线状态。



	0	1	2	3	4	5	6	7	8	9	10	11
lui x10 524288	F	D	E	M	W							
addi x10 x10 1		F	D	E	M	W						
lui x11 524288			F	D	E	M	W					
addi x11 x11 -2				F	D	E	M	W				
add x12 x10 x11					F	D	E	M	W			
addi x10 x0 10						F	D	E	M	W		
ecall							F	D	E	M	W	

图 3: Pipeline table

  - **(8)**：各种缩放视图的按钮。
  - **(9) Registers**：显示处理器中所有寄存器的值，并高亮最近使用的寄存器。
  - **(10) Instruction memory**：显示处理器中当前加载的反汇编指令，并标记了流水线每个阶段当前的指令，在这也可以进行断点设置。
  - **(11) Aplication output**：显示程序的输出。
- `Memory` 显示内存的信息
  - **Registers**：显示类似于 `Processor` 选项卡中的寄存器状态。
  - **Memory**：显示程序的内存状态，可以进行滚动或通过 ***Go to*** 选择特定地址或预定义的地址标签。***Save address*** 能够添加地址标签以便之后查看。
  - **Memory accesses**：显示最近访问的内存地址。选择其中某地址并点击 ***Go to selected address***，能够设置选择的地址为 **Memory** 的中心。

【提交格式】

在本题中，你需要将每一个子任务使用 RISC-V 汇编实现为一个函数。具体地，对于每个子任务，在下发的相应文件夹内都有如下的文件：

- `main.c`：程序入口，你不应该更改这个文件，其中将使用支持库读取数据，调用你实现的函数，并向评测系统返回答案。

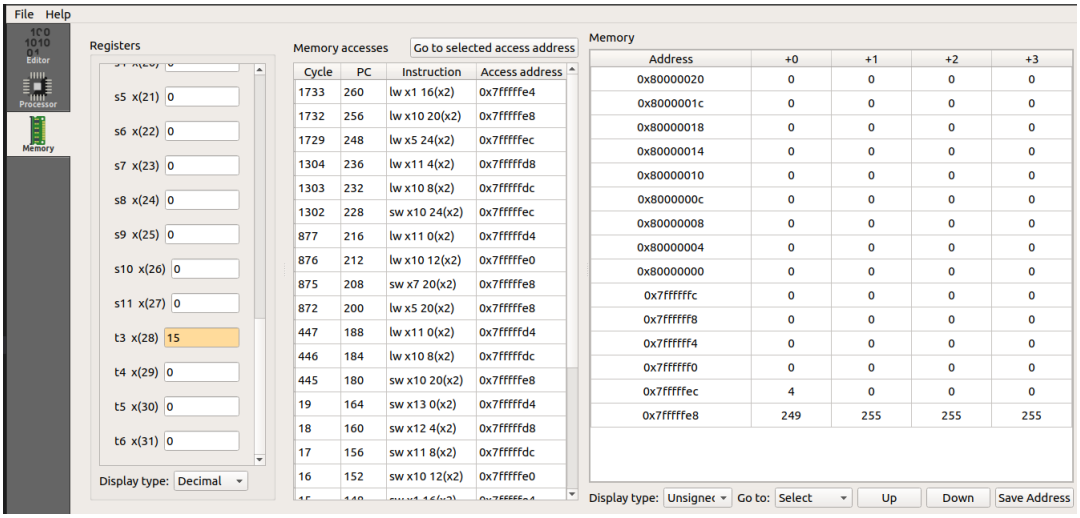


图 4: Memory

- `xxx.s`（其中 `xxx` 为题目名称）：你需要**填充内容和最终提交**的文件，在其中你应该实现具体的计算逻辑。参数的传入与返回约定需要严格遵照相应规范（可见提示部分），我们已经在框架中标记了必要的寄存器。
- `Makefile`：用于将主程序与你的代码分别编译和链接，并生成 ELF 格式可执行文件（注意这一文件仅供你自己测试用，**不要提交**）。
- `xxx.in/ans`：该题的一个样例输入与答案，前者将被 `Ripes` 读取，答案将与你产生的结果进行比较。

最终评测时，计算的周期数实际为你的函数使用的周期数减去空函数（即只包含一条 `ret` 指令）使用的周期数。

工具链与支持库

我们提供了如下工具可供使用：

- `riscv64-linux-gnu-gcc`
- `riscv64-linux-gnu-g++`
- `riscv64-linux-gnu-binutils`（套件中包括对应的 `objdump`, `readelf` 等程序）

你可以使用 `gcc`/`g++` 将 C/C++ 语言编写的函数直接转换为汇编代码，使用以下命令即可（注意**不能缺少任何选项**，并且这样获得的汇编代码**无法**在 Ripes 的图形界面中运行）：

```
riscv64-linux-gnu-gcc -c xxx.c -o xxx.s -march=rv32im \
-mabi=ilp32 -ffreestanding -nostdlib
```

同时，我们还提供了一个支持库，位于 `lib/lib.h`，其中使用 C 语言包装了对多个 `ECALL` 的调用，支持向标准输出打印数字、字符串等，可供在 Ripes 中运行时调试

之用。请注意**不要在最终提交的代码中包含任何 ECALL 指令**，否则该提交将被视为非法。

为了方便你的理解和尝试，我们提供了一个例子，用于计算两个数的和，放置在 `aplusb` 文件夹中。在此文件夹中直接调用 `make` 即可生成 ELF 格式的可执行文件。然后，你可以使用如下的方式使用 `Ripes` 的命令行工具执行这一文件：

```
../RipesRunner aplusb 0 aplusb.in aplusb.out 50
```

其中程序的各个参数分别表示可执行文件路径、子任务编号（0 为样例，1-4 分别为四个正式任务）、输入文件路径、输入文件路径、允许程序运行的最大周期数。

`Ripes` 会向标准错误流中输出一些调试信息，并向标准输出流中输出两个整数，分别是程序运行状态和你的函数耗费的周期数。如果运行状态为 0（即成功退出），我们会将你的输出 `aplusb.out` 与标准答案 `aplusb.ans` 进行对比，如果一致，则认为运行结果正确。你可以通过向评测系统中提交 `aplusb.s` 来得到评测结果，此题无论提交与否都**不会**影响你的成绩。

【子任务】

我们将每一个子任务分离为单独的题目，请在正确的题目中提交相应的 `.s` 代码。

任务 1（整数乘法）

给定两个 32 位整数 A 和 B，请计算 $M = A \times B$ 的值。

你需要实现的函数原型为 `int amulb(int a, int b)`。

任务 2（斐波那契数列）

斐波那契数列的定义为 $F(1) = 1,\ F(2) = 1,\ F(n) = F(n-1) + F(n-2)(n \geq 3)$。

给定两个 32 位正整数 n 和 m，请计算非负整数 $p = F(n)$ 模 m 的**余数**。

你需要实现的函数原型为 `int fib(int n, int m)`。

任务 3（数组排序）

给定一个 32 位整数数组 A 及其长度 n，请将其中的元素排列为**非降序**。

你需要实现的函数原型为 `int *sort(int n, int *A)`。输入的元素存放在 `a[0]` 至 `a[n - 1]` 范围内，你返回的数组同样需要将数据存放在下标 `0` 至 `n - 1` 的范围内。

任务 4（摘水果）

给定一个非负整数 n 和两个长度为 $2n$ 的数组 $a_i, b_i (1 \le i \le 2n)$，你需要完成第一题“摘水果”描述的任务。

你需要实现的函数原型为 `int *fruit(int n, int *a, int *b)`。输入的元素存放在 **`a[1]` 至 `a[n * 2]`**、**`b[1]` 至 `b[n * 2]`** 范围内。你不应当进行超出范围的访问，例如访问 `a[0]` 等。你返回的数组同样需要将数据存放在下标 **`1` 至 `n`** 的范围内。

问题细节

子任务	总分	最大周期限制	测试点编号	测试点描述
1	10	50	1,2,3,4,5,6,7,8,9,10	$\|A\|, \|B\| < 10^4$
2	20	10^6	1	$n = 1, 10^8 < m < 10^9$
			2,3,4	$1 < n \le 30, 10^8 < m < 10^9$
			5,6,7	$1 < n \le 10^3, 10^8 < m < 10^9$
			8,9,10	$1 < n \le 10^5, 10^8 < m < 10^9$
3	30	5×10^7	1,2	$n \le 10^2$
			3,4	$n \le 10^3$
			5,6,7,8	$n \le 10^4$
			9,10	$n \le 10^5$
4	40		1,2,3,4	与第一题第一个子任务一致
			5,6,7,8	与第一题第二个子任务一致
			9,10	与第一题第三个子任务一致

【评分方式】

本题每个子任务有多个测试点。每个子任务的分数已经在上面标出，构成分为两部分：正确性分数（10%）和性能分数（90%）。若你的程序能够正确通过某一个子任务的**全部**测试点，则得该子任务正确性分数的满分，否则该子任务得 0 分。在得到正确性分数的前提下，按照下列规则计算性能分：

设 S_i 为第 i 个子任务的性能分数满分，并且该子任务共有 m 个测试点，t_{ij} 为选手程序在模拟器中运行第 j 个测试点耗费时钟周期数量，T_{ij} 为所有选手程序运行第 j 个测试点耗费时钟周期数量的最小值。那么选手在该子任务的性能得分 s_i 为：

$$s_i = \frac{S_i}{m} \sum_{j=1}^{m} \frac{T_{ij}}{t_{ij}}$$

本题中，你的程序运行的周期数将在“评测信息”中给出，而评测平台给出的“时间”和“空间”为模拟器运行消耗的资源，都没有实际的意义。我们为所有测试点设置了一个最大的运行周期数量限制（见问题规模描述），如果你的程序运行超过了这一周

期数，模拟器将会直接退出，并将这一个测试点标记为 `Time Limit Exceeded`。此外，你的代码生成的 ELF 文件尺寸不可超过 64MB（仅计算代码段与初始化后的数据段），否则模拟器将会将测试点标记为 `Memory Limit Exceeded`。

【提示】

你应当留意下发的 RISC-V 规范中的下列章节：

- 第 2、7 章中包含了各个指令的具体格式和语义；
- 第 25 章的 `RV32I` 和 `RV32M` 表中包含了所有可用的指令；
- 第 26 章包含了 RISC-V 的程序二进制接口（ABI），以及所有在汇编器中可用的伪指令。