A. Заполни форму (5 баллов)

1 секунда, 512 мегабайт

Это необычная задача. Её цель — проинформировать вас о необходимости заполнить форму.

Прежде чем вы приступите к решению задач, мы хотим вам напомнить о необходимости заполнить форму по ссылке https://forms.office.com/e/nDwCDTBa69. Форма закроется **5 февраля в 23:59 (московское время)**. Пожалуйста, не забудьте заполнить эту форму.

Если вы уверены, что справитесь с заполнением формы в срок, то у вас есть возможность получить дополнительные 5 баллов! Не нужно форму заполнять прямо сейчас во время Контеста (она довольно длинная). Просто сделайте это после окончания Контеста до дедлайна.

В качестве решения этой задачи отошлите программу, которая выводит в стандартный вывод (консоль) фразу «`I am sure that I will fill out the form by 23:59 on February 5, 2023.`».

Входные данные

В этой задаче нет входных данных. Ваша программа не должна что-либо считывать.

Выходные данные

Выведите фразу «`I am sure that I will fill out the form by 23:59 on February 5, 2023.`» (без кавычек).

входные данные

выходные данные
`I am sure that I will fill out the form by 23:59 on February 5, 2023.`

B. Морской бой (10 баллов)

2 секунды, 512 мегабайт

Это наиболее простая задача Контеста. Надеемся, что вы познакомились с тестирующей системой Codeforces, приняв участие в Песочнице. В любом случае напоминаем вам несколько моментов.

Правильные решения задач должны проходить все заранее заготовленные тесты жюри и укладываться в ограничения по времени/памяти на каждом тесте. В некоторых задачах допустимы частичные решения (это написано в условии).

Ниже перечислены технические требования к решениям:

- решение располагается в одном файле исходного кода;
- решение читает входные данные со стандартного ввода (экрана);
- решение пишет выходные данные на стандартный вывод (экран);
- решение не взаимодействует как-либо с другими ресурсами компьютера (сеть, жесткий диск, процессы и прочее);
- решение использует только стандартную библиотеку языка;
- решение располагается в пакете по-умолчанию (или его аналоге для вашего языка), имеют стандартную точку входа для консольных программ;
- гарантируется, что во всех тестах выполняются все ограничения, что содержатся в условии задачи — как-либо проверять входные данные на корректность не надо, все тесты строго соответствуют описанному в задаче формату;
- выводи ответ в точности в том формате, как написано в условии задачи (не надо выводить «поясняющих» комментариев типа *введите число* или *ответ равен*);
- решения можно отправлять сколько угодно раз (пожалуйста, только без абьюза системы).

Для вашего удобства большинство тестов, на которых будут тестироваться ваши решения, являются открытыми. В каждой задаче можно скачать архив тестов (смотрите сайдбар справа, раздел «Материалы соревнования»).

Обратите внимание на SQL-задачу. Не забудьте попробовать свои силы в её решении.

Перейдём к задаче.

Вы участвуете в разработке подсистемы проверки поля для игры «Морской бой». Вам требуется написать проверку корректности количества кораблей на поле, учитывая их размеры. Напомним, что на поле должны быть:

- четыре однопалубных корабля,
- три двухпалубных корабля,
- два трёхпалубных корабля,
- один четырёхпалубный корабль.

Вам заданы 10 целых чисел от 1 до 4. Проверьте, что заданные размеры соответствуют требованиям выше.

Входные данные

В первой строке записано целое число t ($1 \le t \le 1000$) — количество наборов входных данных в тесте.

Наборы входных данных в тесте независимы. Друг на друга они никак не влияют.

Каждый набор входных данных состоит из одной строки, которая содержит 10 целых чисел $a_1, a_2, \ldots, a_{10}$ ($1 \le a_i \le 4$) — размеры кораблей на поле в произвольном порядке.

Обратите внимание, что уже гарантируется, что кораблей на поле ровно 10 и их размеры от 1 до 4, включительно. Вам необходимо проверить, что количество кораблей каждого типа соответствует правилам игры.

Выходные данные

Для каждого набора входных данных в отдельной строке выведите:

- `YES`, если заданные размеры кораблей на поле соответствуют правилам игры;
- `NO` в противном случае.

Вы можете выводить `YES` и `NO` в любом регистре (например, строки `yEs`, `yes`, `Yes` и `YES` будут распознаны как положительный ответ).

входные данные

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

выходные данные

`YES`
`NO`
`YES`
`YES`
`NO`

C. Автомобильные номера (15 баллов)

2 секунды, 512 мегабайт

В Берляндии автомобильные номера состоят из цифр и прописных букв латинского алфавита. Они бывают двух видов:

- либо автомобильный номер имеет вид `буква-цифра-цифра-буква-буква` (примеры корректных номеров первого вида: `R48FA`, `O000O`, `A99OK`);
- либо автомобильный номер имеет вид `буква-цифра-буква-буква` (примеры корректных номеров второго вида: `T7RR`, `A9PQ`, `O0OO`).

Таким образом, каждый автомобильный номер является строкой либо первого, либо второго вида.

Вам задана строка из цифр и прописных букв латинского алфавита. Можно ли разделить её пробелами на последовательность корректных автомобильных номеров? Иными словами, проверьте, что заданная строка может быть образована как последовательность корректных автомобильных номеров, которые записаны подряд без пробелов. В случае положительного ответа выведите любое такое разбиение.

Входные данные

В первой строке записано целое число t ($1 \le t \le 1000$) — количество наборов входных данных в тесте.

Наборы входных данных в тесте независимы. Друг на друга они никак не влияют.

Каждый набор входных данных — непустая строка s, которая состоит из цифр и прописных букв латинского алфавита. Длина строки — от 1 до 50 символов.

Выходные данные

Выведите n строк: очередная строка должна содержать ответ для соответствующего набора входных данных.

Если ответ отрицательный — то есть заданную строку s невозможно представить как последовательность номеров автомобилей — строка в выводе должна содержать единственный символ '–' (минус, ASCII-код 45).

В случае положительного ответа выведите любое разбиение заданной строки s на последовательность корректных номеров. Каждый номер должен соответствовать одному из двух видов (см. условие). Номера разделяйте пробелами. Вы можете выводить произвольное количество пробелов и даже лишние пробелы после последнего номера.

входные данные

```
6
R48FAO00000000A99OKA99OK
R48FAO00000000A99OKA99O
A9PQ
A9PQA
A99AAA99AAA99AAA99AA
AP9QA
```

выходные данные

```
R48FA O000O O0OO A99OK A99OK
-
A9PQ
-
A99AA A99AA A99AA A99AA
-
```

D. Результаты соревнования (20 баллов)

3 секунды, 512 мегабайт

В соревновании по бегу приняли участие n спортсменов: i-й из них пробежал дистанцию за t_i секунд. Жюри хочет назначить места участникам по следующим правилам:

- места пронумерованы от 1 и далее (лучшее место — первое);
- если у двух спортсменов результаты одинаковые или отличаются на одну секунду, то они делят место (в этом случае считаем, что они делят лучшее из поделенных мест);
- участники делят место только в результате применения предыдущего правила (возможно, несколько раз);
- если k участников делят место p, то места следующих за ними участников нумеруются начиная с $k + p$.

Рассмотрите следующие примеры, чтобы понять принцип назначения мест:

- допустим, $n = 4$ и $t = [20, 10, 20, 30]$, тогда места имеют вид $[2, 1, 2, 4]$ (второй спортсмен прибежал первым — у него первое место, первый и третий поделили второе место, четвёртый занял последнее четвёртое место);
- допустим, $n = 3$ и $t = [5, 7, 6]$, тогда места имеют вид $[1, 1, 1]$ (так как $t_1 = 5$ и $t_3 = 6$ отличаются на 1, то первый и третий спортсмены должны занять одинаковое место, аналогично со вторым и третьим спортсменами, следовательно, все трое делят первое место);
- допустим, $n = 5$ и $t = [6, 3, 4, 3, 1]$, тогда места имеют вид $[5, 2, 2, 2, 1]$;
- допустим, $n = 5$ и $t = [200, 10, 100, 11, 200]$, тогда места имеют вид $[4, 1, 3, 1, 4]$.

По заданным значениям n и $t_1, t_2, \ldots, t_n$ выведите последовательность мест, занятых спортсменами.

Неполные решения этой задачи (например, недостаточно эффективные) могут быть оценены частичным баллом.

Входные данные

В первой строке записано целое число t ($1 \le t \le 1000$) — количество наборов входных данных в тесте.

Наборы входных данных в тесте независимы. Друг на друга они никак не влияют.

Первая строка каждого набора входных данных содержит целое число n ($1 \le n \le 2 \cdot 10^5$) — количество спортсменов.

Вторая строка набора содержит последовательность целых чисел $t_1, t_2, \ldots, t_n$ ($1 \le t_i \le 10^9$), где t_i — время в секундах, за которое i-й спортсмен пробежал дистанцию.

Сумма значений n по всем наборам входных данных теста не превосходит $2 \cdot 10^5$.

Выходные данные

Для каждого набора входных данных выведите n положительных чисел $r_1, r_2, \ldots, r_n$, где r_i — место i-го спортсмена.

входные данные

```
6
4
20 10 20 30
3
5 7 6
5
6 3 4 3 1
5
200 10 100 11 200
1
1000000000
11
13 8 12 1 7 10 1 8 10 2 17
```

выходные данные

```
2 1 2 4
1 1 1
5 2 2 2 1
4 1 3 1 4
1
9 4 9 1 4 7 1 4 7 1 11
```

E. Товарные знаки (20 баллов)

5 секунд, 512 мегабайт

В Торгово-промышленную палату (ТПП) Берляндии были поданы n заявок на регистрацию товарных знаков. Каждая заявка — это непустая строка из букв латинского алфавита.

ТПП отклоняет очередную заявку, если ранее был зарегистрирован *похожий* товарный знак. Если такого не было, то товарный знак регистрируется палатой.

Два товарных знака *похожи*, если их можно сделать равными с помощью нуля или более таких операций:

- возьмём любой из двух знаков,
- найдём в знаке две или более одинаковые идущие подряд буквы,
- добавим в этот блок одинаковых букв ещё одну такую же букву.

Например:

- товарные знаки `Booble` и `Boooble` похожи — берём в первом товарном знаке две подряд идущие буквы `o` и добавляем к ним ещё одну букву `o`, так получим второй товарный знак `Boooble`;
- товарные знаки `yyyess` и `yyessss` похожи — сначала изменим второй товарный знак: `yyessss` $\to$ `yyyessss`, потом два раза изменим первый товарный знак: `yyyess` $\to$ `yyyesss` $\to$ `yyyessss`. Так с помощью последовательности операций получилось сделать оба знака равными `yyyessss`;
- товарные знаки `oooops` и `oooops` похожи — операций производить не надо, знаки уже равны;
- товарные знаки `ooooooopppss` и `oooopppssss` похожи — например, сначала изменим второй товарный знак `oooopppssss` $\to$ `ooooopppssss` $\to$ `oooooopppssss` $\to$ `ooooooopppssss`, затем изменим первый: `ooooooopppss` $\to$ `oooooopppsss` $\to$ `ooooooopppssss`.

Обратите внимание, что добавлять букву можно только в блок идущих подряд двух или более одинаковых букв. Строчные и прописные буквы следует различать.

Примеры пар непохожих товарных знаков: `a` и `aa`, `MMM` и `mmm`, `yess` и `yyes`.

Считая, что до обработки n заявок в ТПП не было зарегистрировано других товарных знаков, выведите количество знаков, которые будут в итоге зарегистрированы.

Неполные решения этой задачи (например, недостаточно эффективные) могут быть оценены частичным баллом.

Входные данные

В первой строке входных данных записано целое число t ($1 \le t \le 1000$) — количество наборов входных данных.

Наборы входных данных в тесте независимы. Друг на друга они никак не влияют.

В первой строке каждого набора записано целое число n ($1 \le n \le 10^5$) — количество заявок. Далее следуют сами заявки, по одной заявке в строке. Каждая заявка является непустой строкой из букв латинского алфавита.

Гарантируется, что сумма длин заявок по всем наборам входных данных теста не превосходит 10^6.

Выходные данные

Выведите t чисел — для каждого набора входных данных выведите суммарное количество товарных знаков, которые будут зарегистрированы палатой. Обратите внимание, что наборы входных данных следует обрабатывать независимо (они не влияют друг на друга).

входные данные

```
3
8
Booble
yyyess
oooops
ooooooopppss
Boooble
yyessss
oooops
oooopppssss
6
a
aa
MMM
mmm
yess
yyes
5
rrrrrrrr
rrrrrr
rrr
rr
r
```

выходные данные

```
4
6
2
```

F. Печать документа (20 баллов)

3 секунды, 512 мегабайт

Необходимо напечатать документ из k страниц. Его страницы пронумерованы от 1 до k.

Некоторые его страницы были уже предварительно напечатаны. Известно, что на принтер было отослано одно задание, которое содержало список страниц для печати.

Этот список содержит хотя бы одну страницу, и хотя бы одна страница от 1 до k не попадает в этот список. Список страниц состоит из перечисленных через единичную запятую элементов списка, где каждый элемент — это:

- либо конкретный номер одной страницы (целое число от 1 до k),
- либо диапазон страниц, записанный в формате «l-r», где l — начало диапазона печати, а r — конец диапазона печати (и l и r целые числа, удовлетворяющие неравенству $1 \le l \le r \le k$).

Страница может многократно присутствовать в списке на печать, но будет напечатана лишь единожды.

Иными словами, список страниц имеет формат, аналогичный тому, что используется в «Microsoft Word» или других подобных программах.

Например, если $k = 8$, то допустимые списки страниц:

- `7` (была напечатана лишь страница 7),
- `1,7,1` (были напечатаны страницы 1 и 7),
- `1-5,1,7-7` (были напечатаны страницы $1, 2, 3, 4, 5, 7$).

Примеры неправильных диапазонов для $k = 8$ (такие входные данные недопустимы):

- `1-8` (хотя бы одна страница документа должна быть еще не напечатана),
- `1,,3` (две запятые не могут идти подряд),
- `7-9` (девятую страницу нельзя послать на печать),
- `1-5,` (каждая запятая должна разделять два элемента, заканчивать список она не может),
- `,1,2,3-5` (каждая запятая должна разделять два элемента, начинать список она не может),
- `3-4-7` (нарушен формат элемента, так нельзя).

Выведите кратчайший список страниц в аналогичном формате, который надо дополнительно послать на печать, чтобы в итоге напечатать все страницы от 1 до k, не напечатанные ранее.

Иными словами, найдите такую наиболее короткую строку, которая является корректным списком страниц и содержит те и только те страницы, которые еще не были напечатаны.

Если ответов несколько, то выведите любой из них.

Входные данные

В первой строке входных данных записано целое число t ($1 \le t \le 100$) — количество наборов входных данных.

Наборы входных данных в тесте независимы. Друг на друга они никак не влияют.

В первой строке каждого набора записано целое число k ($2 \le k \le 100$) — количество страниц в документе.

Вторая строка каждого набора содержит список страниц, которые уже были посланы на печать. Этот список корректен и отформатирован строго по правилам, написанным выше. Он содержит только страницы от 1 до k, содержит хотя бы одну страницу, и хотя бы одна страница от 1 до k в него не входит. Строка не содержит пробелы или какие-либо другие дополнительные символы. Длина этой строки — от 1 до 400, включительно.

Выходные данные

Выведите t строк — для каждого набора входных данных выведите кратчайшую строку, содержащую корректный список страниц, которые надо допечатать и только их. Если оптимальных ответов несколько, то выведите любой из них.

входные данные

```
7
8
7
8
1,7,1
8
1-5,1,7-7
10
1-5
10
1,2,3,4,5,6,8,9,10
3
1-2
100
1-2,3-7,10-20,100
```

выходные данные

```
8,1-6
2-6,8
6,8
6-10
7
3
9,21-99,8
```

G. Записи ко врачу (25 баллов)

3 секунды, 512 мегабайт

Врач за день может принять до n пациентов. Окна времени для приёма пронумерованы от 1 до n в хронологическом порядке. Ассистент записал m человек, i-го человека на окно w_i ($1 \le w_i \le n$). Выяснилось, что некоторые пациенты могут быть записаны на одно окно. Может ли ассистент попросить подвинуться некоторых пациентов на одно окно вперёд или назад так, чтобы у врача была возможность принять всех?

Обратите внимание, что для каждого пациента возможны три сценария:

- оставить его запись как есть,
- изменить его запись на одно окно назад (то есть заменить значение w_i на $w_i - 1$),
- изменить его запись на одно окно вперёд (то есть заменить значение w_i на $w_i + 1$).

Конечно, в двух последних случаях запись должна остаться в окне от 1 до n.

Определите, можно ли исправить записи требуемым образом так, чтобы врач смог принять всех пациентов. В случае положительного ответа выведите любой из способов это сделать.

Обратите внимание, что вам не требуется минимизировать количество перенесённых записей. Достаточно найти любой способ перенести произвольное количество записей требуемым образом, чтобы врач смог принять всех пациентов.

Неполные решения этой задачи (например, недостаточно эффективные) могут быть оценены частичным баллом.

Входные данные

В первой строке входных данных записано целое число t ($1 \le t \le 1000$) — количество наборов входных данных.

Наборы входных данных в тесте независимы. Друг на друга они никак не влияют.

В первой строке каждого набора записаны два целых числа n и m ($1 \le m \le n \le 3 \cdot 10^5$), где n — количество окон приёма у врача, а m — количество записанных пациентов.

Во второй строке записано m целых чисел $w_1, w_2, \ldots, w_m$ ($1 \le w_i \le n$), где w_i — номер окна, на которое записан i-й пациент.

Гарантируется, что сумма значений n по всем наборам входных данных теста не превосходит $3 \cdot 10^5$.

Выходные данные

Выведите m строк, j-я строка должна содержать ответ на j-й набор входных данных.

Если перенести записи требуемым образом невозможно, выведите `x` (строчную латинскую букву «икс»).

Иначе выведите строку без пробелов из m символов, где каждый символ — это:

- `0` (ноль), если запись соответствующего пациента менять не надо,
- `-` (минус), если запись соответствующего пациента надо изменить на одно окно назад,
- `+` (плюс), если запись соответствующего пациента надо изменить на одно окно вперёд.

Если существует несколько способов изменить записи требуемым образом, то выведите любой из них. Количество изменений записей минимизировать не требуется.

входные данные

```
4
5 5
2 1 3 5 2
5 4
1 1 1 1
200000 4
100000 100001 100001 100000
2 2
2 1
```

выходные данные

```
00+0+
x
-0+0
00
```

H. Block Jigsaw Puzzle (30 баллов)

10 секунд, 512 мегабайт

Слышали о Block Jigsaw Puzzle? Сейчас расскажем!

Игра происходит на поле 8×8. Каждая клетка поля либо свободна, либо занята блоком. Например, ниже на картинке изображено одно из возможных текущих состояний поля.

Вам поступают фигуры, которые вы должны укладывать на поле. Вращать фигуры вы не можете. Когда вы кладёте фигуру на поле, все клетки, соответствующие этой фигуре, должны быть свободны. Вы можете выбрать любой из корректных способов положить фигуру.

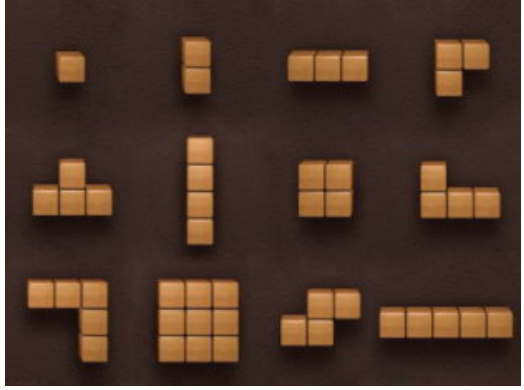
Например, допустим, вы хотите положить следующую фигуру на поле:

Есть всего один корректный способ положить её:

Как только фигура оказалась на поле, все полностью заполненные блоками строки и столбцы поля очищаются. Это действие происходит мгновенно, оставшиеся заполненные блоки остаются неподвижными.

В рассматриваемом примере очистятся одна строка и один столбец. После этого поле примет следующий вид:

Возможные фигуры для укладывания могут выглядеть следующим образом:



Могут быть даны любые повороты и зеркальные отражения фигур с картинки выше. В процессе укладывания фигуры вращать её нельзя.

Вам задано текущее состояние поля и набор фигур для укладывания. Ваша цель — уложить фигуры так, чтобы минимизировать количество блоков на поле в конце игры. Фигуры можно укладывать в любом порядке. Обратите внимание, что каждый раз после размещения очередной фигуры на поле происходит очищение полностью заполненных блоками строк и столбцов.

Требуется последовательно уложить все фигуры и вывести минимальное количество блоков, которое останется на поле после укладывания всех фигур. Выведите `-1`, если уложить все фигуры невозможно.

Неполные решения этой задачи (например, недостаточно эффективные) могут быть оценены частичным баллом. Для набора 15 баллов достаточно решить частный случай задачи с $k = 1$.

Входные данные

Первые 8 строк содержат по 8 символов каждая — текущее состояние поля. Символ `'.'` (точка) соответствует пустой клетке, а символ `'*'` (звёздочка) — клетке с блоком. Гарантируется, что текущее состояние поля таково, что в нём нет сплошных строк и/или столбцов из блоков.

Следующая строка содержит целое число k ($1 \le k \le 3$) — количество заданных фигур. Обратите внимание, что первые тесты содержат только $k = 1$, и для набора 15 баллов достаточно решить частный случай задачи с $k = 1$.

Затем заданы фигуры. Описание каждой фигуры начинается со строки, которая содержит целое число l_i ($1 \le l_i \le 5$) — количество строк в описании фигуры. Затем заданы l_i строк с описанием фигуры в виде прямоугольника минимального размера, который её содержит. Формат задания аналогичен тому, как задано поле. Каждая фигура является одной из 12 допустимых (перечислены на картинке выше), возможно, повёрнутая и/или отражённая.

Выходные данные

Выведите одно целое число — минимальное количество блоков на поле после укладывания k заданных фигур в произвольном порядке оптимальным способом.

Выведите `-1`, если не существует способа уложить все фигуры на поле.

Заметим, что последний пример иллюстрирует случай $k > 1$. Этот пример не обязательно должен проходить, чтобы вы набрали 15 баллов.

входные данные

```
*****.**
***.*.**
**..****
**..*...
........
****.***
...*.***
******..
1
3
***
..*
..*
```

выходные данные

```
29
```

входные данные

```
.*******
******.*
*****.**
*.....**
*****.**
*.***.**
**.*****
*******.
1
3
.*
**
.*
```

выходные данные

```
33
```

входные данные

```
*.******
..******
*..**.**
*.*.****
*.***.**
*.**..**
*.*.**.*
*.*****.
1
2
**.
.**
```

выходные данные

```
28
```

входные данные

```
***.****
*.******
.***....
*.*....*
*.***..*
*.....*.
*...*.**
.******.
3
2
**
**
5
*
*
*
*
*
2
.**
**.
```

выходные данные

```
18
```

I. Отображение страниц (40 баллов)

10 секунд, 512 мегабайт

В этой задаче вам предстоит написать отображение в текстовом виде простейших HTML-таблиц.

В задаче используется очень ограниченное подмножество HTML-разметки. Внимательно прочтите условие задачи.

В этой задаче все таблицы имеют строго прямоугольный вид, то есть содержат не менее одной строки, каждая строка содержит не менее одной ячейки, и во всех строках каждой отдельной таблицы одинаковое количество ячеек.

Таблица описывается следующей грамматикой:

- ⟨TABLE⟩ := <table>⟨ROWS⟩</table>
- ⟨ROWS⟩ := ⟨ROW⟩|⟨ROW⟩⟨ROWS⟩
- ⟨ROW⟩ := <tr>⟨CELLS⟩</tr>
- ⟨CELLS⟩ := ⟨CELL⟩|⟨CELL⟩⟨CELLS⟩
- ⟨CELL⟩ := <td></td>|<td>⟨TABLE⟩</td>

Таким образом, каждая ячейка таблицы либо пуста, либо содержит ровно одну другую таблицу.

Вот пример минимальной корректной таблицы: <table><tr><td></td></tr></table>.

Вот пример минимальной корректной таблицы размера 2×3: <table><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>.

Вот пример таблицы 2×1, в нижнюю клетку которой вложена минимальная таблица: <table><tr><td></td></tr><tr><td><table><tr><td></td></tr></table></td></tr></table>.

Ваша задача — отобразить заданную таблицу в текстовом виде. Каждая из таблиц (как внешняя, так и любая внутренняя) должна занимать минимальную площадь. Содержимое любой ячейки прижимается к её левому верхнему углу.

Изучите примеры и точно следуйте всем особенностям форматирования, которые продемонстрированы в них. В этой задаче вам нужно вывести в точности ожидаемое представление таблицы — не выводите лишние незначащие пробелы или какие-либо другие символы.

Входные данные

В первой строке теста записано целое число t ($1 \le t \le 100$) — количество наборов входных данных в тесте.

Далее следуют описания наборов. Наборы входных данных в тесте независимы. Друг на друга они никак не влияют.

Каждый набор начинается строкой, которая содержит целое число k ($1 \le k \le 1000$) — количество строк, в которых записан очередной набор.

Далее следуют k строк, в которых записано описание ровно одной таблицы. Описание произвольным образом разбито на k строк и отформатировано пробелами, которые следует полностью игнорировать. Пробелы могут даже разрывать ключевые слова в HTML-разметке.

Каждая таблица (и внешняя, и все вложенные) соответствует грамматике выше и имеет строго прямоугольную форму: количества ячеек в каждой строке равны в рамках каждой отдельной таблицы.

Размер файла каждого теста не превосходит 1 мегабайта. Глубина вложенности таблиц не превосходит 10. Гарантируется, что размер вывода для любого теста не превосходит 1 мегабайта.

Выходные данные

Выведите t заданных таблиц, следуя примерам. Каждая таблица должна занимать минимальную возможную площадь. Содержимое любой ячейки прижимается к её левому верхнему углу.

Точно следуйте примерам, они демонстрируют все особенности и требования форматирования вывода.

Гарантируется, что размер вывода для любого теста не превосходит 1 мегабайта.

входные данные

```
4
5
<table>
    <tr>
        <td></td>
    </tr>
</table>
4
<table>
    <tr><td></td><td></td><td></td></tr>
    <tr><td></td><td></td><td></td></tr>
</table>
5
<t                      a
ble><tr><td></td></tr><tr
><td><table><tr><td></td>
</tr></table></td></tr></
table                   >
35
<table>
    <tr>
        <td>
        </td>
        <td>
            <table>
                <tr><td>
                    <table><tr><td></td></tr></table>
                </td></tr>
            </table>
        </td>
        <td>
        </td>
    </tr>
    <tr>
        <td>
            <table><tr>
                <td></td>
                <td></td>
                <td></td>
                <td></td>
            </tr></table>
        </td>
        <td>
            <table><tr><td></td></tr></table>
        </td>
        <td>
            <table>
                <tr><td></td></tr>
                <tr><td></td></tr>
                <tr><td></td></tr>
            </table>
        </td>
    </tr>
</table>
```

выходные данные

```
+-+
|.|
+-+
+-+-+-+
|.|.|.|
+-+-+-+
|.|.|.|
+-+-+-+
+---+
|...|
+---+
|+-+|
||.||
|+-+|
+---+
+---------+-----+---+
|.........|+---+|...|
|.........||+-+||...|
|.........|||.|||...|
|.........||+-+||...|
|.........|+---+|...|
+---------+-----+---+
|+-+-+-+-+|+-+..|+-+|
||.|.|.|.|||.|..||.||
|+-+-+-+-+|+-+..|+-+|
|.........|.....||.||
|.........|.....|+-+|
|.........|.....||.||
|.........|.....|+-+|
+---------+-----+---+
```

S1. Контест: простые задачи (SQL, 15 баллов)

15 секунд, 1024 мегабайта

Это необычная задача — вам надо написать SQL-запрос. В качестве решения вы должны отослать один запрос к базе данных, который возвращает требуемые данные. Запрос может содержать произвольное количество подзапросов, других конструкций, быть сколь угодно навороченным, но это должен быть один запрос (в нём не должна встречаться точка с запятой для разделения разных запросов).

При проверке вашего решения используется PostgreSQL 15.1. В качестве входных данных вам предоставляется дамп состояния базы данных. Обратите внимание, что время работы вашего решения на тесте включает восстановление состояния базы данных из дампа, но это время значительно меньше ограничения по времени. Вы можете использовать сервис http://sqlfiddle.com/ как инструмент для запуска запросов.

В этой задаче вам предстоит написать запрос к базе данных простейшей системы проведения соревнований по программированию. Вы прямо сейчас участвуете в подобном соревновании. Время почувствовать себя в роли разработчика системы для проведения таких соревнований!

Напишите запрос к базе данных, который возвращает все задачи, которые были решены не менее чем двумя пользователями. Найденные задачи следует отсортировать по `id`.

Внимательно ознакомьтесь с примерами вывода. Ваш запрос должен иметь **в точности** такой же вывод на примерах.

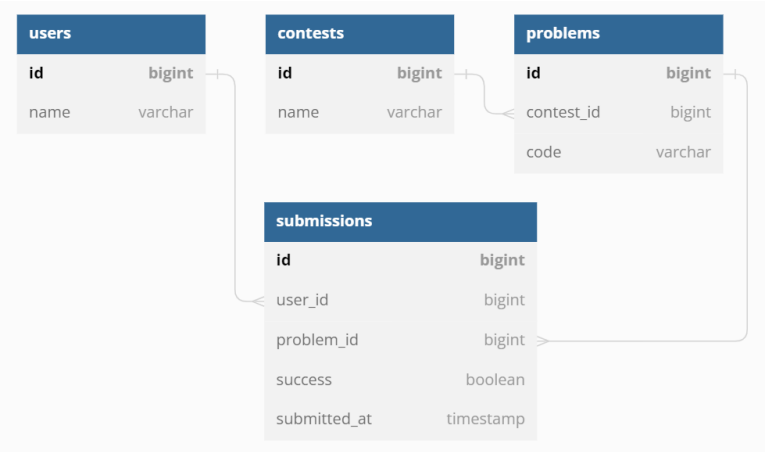
Схема базы данных содержит четыре таблицы:

- `users` — пользователи системы (описываются двумя полями: `id` и `name`),
- `contests` — контесты в системе (описываются двумя полями: `id` и `name`),
- `problems` — задачи в системе, каждая задача принадлежит одному контесту (описываются тремя полями: `id`, `contest_id` и `code`, где `code` — это кодовое короткое название задачи),

- `submissions` — отосланные попытки решения задач, каждая попытка принадлежит одной задаче и одному пользователю (описываются 5 полями: `id`, `user_id`, `problem_id`, `success` и `submitted_at`, где `success` — это булевское значение была ли попытка успешной и `submitted_at` — дата-время, когда попытка была совершена).

Таким образом, `contests` и `problems` находятся в отношении «один ко многим», `submissions` и `users` находятся в отношении «многие к одному», `submissions` и `problems` находятся в отношении «многие к одному».

Изучите входные данные примера, чтобы подробно ознакомиться со схемой базы данных. Диаграмма ниже иллюстрирует схему базы данных.



Входные данные

Входными данными в этой задаче является дамп базы данных. Вам он может быть полезен для ознакомления с состоянием базы данных для конкретного теста. В качестве решения вы должны отправить один SQL-запрос.

Выходные данные

Ваш SQL-запрос должен вывести все подходящие задачи в порядке возрастания их `id`.

Внимательно ознакомьтесь с примерами вывода. Ваш запрос должен иметь **в точности** такой же вывод на примерах.

входные данные

```
create table users (
  id bigint primary key,
  name varchar not null
);

create table contests (
  id bigint primary key,
  name varchar not null
);

create table problems (
  id bigint primary key,
  contest_id bigint,
  code varchar not null,
  constraint fk_problems_contest_id foreign key (contest_id)
references contests (id)
);

create unique index on problems (contest_id, code);

create table submissions (
  id bigint primary key,
  user_id bigint,
  problem_id bigint,
  success boolean not null,
  submitted_at timestamp not null,
  constraint fk_submissions_user_id foreign key (user_id)
references users (id),
  constraint fk_submissions_problem_id foreign key (problem_id)
references problems (id)
);

insert into users
values (1, 'Marie Curie'),
       (2, 'Stephen Hawking'),
       (3, 'Ada Lovelace'),
       (4, 'Albert Einstein'),
       (5, 'Archimedes');

insert into contests
values (1, 'Sandbox-Juniors'),
       (2, 'Sandbox-Seniors'),
       (3, 'Contest-Juniors'),
       (4, 'Contest-Seniors');

insert into problems
values (1, 1, 'A'),
       (2, 2, 'A'),
       (3, 3, 'A'),
       (4, 3, 'B'),
       (5, 4, 'A'),
       (6, 4, 'B');

insert into submissions
values (1, 2, 2, false, '2023-02-05 11:01:00'),
       (2, 2, 2, true, '2023-02-05 11:02:00'),
       (3, 2, 6, true, '2023-02-05 11:03:01'),
       (4, 2, 1, true, '2023-02-05 11:04:00'),
       (5, 2, 1, true, '2023-02-05 11:05:00'),
       (6, 3, 6, true, '2023-02-05 11:06:00'),
       (17, 1, 6, true, '2023-02-05 11:03:00'),
       (8, 1, 2, true, '2023-02-05 11:08:00'),
       (9, 1, 1, false, '2023-02-05 11:09:00'),
       (10, 3, 1, false, '2023-02-05 11:10:00'),
       (11, 5, 5, false, '2023-02-05 11:11:00'),
       (13, 2, 6, true, '2023-02-05 11:03:00'),
       (14, 3, 6, false, '2023-02-05 11:05:59'),
       (15, 1, 6, true, '2023-02-05 11:04:00');
```

выходные данные

```
 id | contest_id | code
----+------------+------
  2 |          2 | A
  6 |          4 | B
(2 rows)
```

входные данные

```
create table users (
  id bigint primary key,
  name varchar not null
);

create table contests (
  id bigint primary key,
  name varchar not null
);

create table problems (
  id bigint primary key,
  contest_id bigint,
  code varchar not null,
  constraint fk_problems_contest_id foreign key (contest_id)
references contests (id)
);

create unique index on problems (contest_id, code);

create table submissions (
  id bigint primary key,
  user_id bigint,
  problem_id bigint,
  success boolean not null,
  submitted_at timestamp not null,
  constraint fk_submissions_user_id foreign key (user_id)
references users (id),
  constraint fk_submissions_problem_id foreign key (problem_id)
references problems (id)
);

insert into users
values (1, 'Olivia'),
       (2, 'Henry'),
       (3, 'Lucas'),
       (4, 'John'),
       (5, 'Charlotte'),
       (6, 'Henry');

insert into contests
values (3, 'Main'),
       (1, 'Practice');

insert into problems
values (1, 3, 'A'),
       (2, 3, 'B'),
       (3, 1, 'A');

insert into submissions
values (10, 3, 2, false, '2023-02-05 11:05:12'),
       (20, 3, 2, true, '2023-02-05 11:07:49'),
       (30, 3, 2, true, '2023-02-05 11:07:49'),
       (40, 3, 1, false, '2023-02-05 11:01:32'),
       (50, 3, 1, false, '2023-02-05 11:11:46'),
       (60, 3, 1, false, '2023-02-05 11:27:05'),
       (70, 6, 2, false, '2023-02-05 11:04:00'),
       (80, 6, 2, true, '2023-02-05 11:05:00'),
       (90, 6, 2, false, '2023-02-05 11:06:00'),
       (100, 6, 2, true, '2023-02-05 11:07:00'),
       (110, 6, 1, false, '2023-02-05 11:08:00'),
       (120, 6, 1, true, '2023-02-05 11:09:00'),
       (130, 2, 2, false, '2023-02-05 11:00:01'),
       (150, 5, 1, false, '2023-02-05 11:07:48'),
       (160, 5, 1, true, '2023-02-05 11:07:49'),
       (170, 5, 1, true, '2023-02-05 11:07:50'),
       (180, 1, 3, false, '2023-02-04 15:00:01'),
       (190, 1, 3, true, '2023-02-04 15:00:01'),
       (200, 5, 3, true, '2023-02-04 15:00:01'),
       (210, 5, 3, false, '2023-02-04 15:00:01'),
       (220, 2, 3, false, '2023-02-04 15:00:01'),
       (230, 6, 3, false, '2023-02-04 15:00:01'),
       (240, 6, 3, false, '2023-02-04 15:00:01');
```

выходные данные

```
 id | contest_id | code
----+------------+------
  1 |          3 | A
  2 |          3 | B
  3 |          1 | A
(3 rows)
```

S2. Контест: статистика пользователей (SQL, 15 баллов)

15 секунд, 1024 мегабайта

Это необычная задача — вам надо написать SQL-запрос. В качестве решения вы должны отослать один запрос к базе данных, который возвращает требуемые данные. Запрос может содержать произвольное количество подзапросов, других конструкций, быть сколь угодно навороченным, но это должен быть один запрос (в нём не должна встречаться точка с запятой для разделения разных запросов).

При проверке вашего решения используется PostgreSQL 15.1. В качестве входных данных вам предоставляется дамп состояния базы данных. Обратите внимание, что время работы вашего решения на тесте включает восстановление состояния базы данных из дампа, но это время значительно меньше ограничения по времени. Вы можете использовать сервис http://sqlfiddle.com/ как инструмент для запуска запросов.

В этой задаче вам предстоит написать запрос к базе данных простейшей системы проведения соревнований по программированию. Вы прямо сейчас участвуете в подобном соревновании. Время почувствовать себя в роли разработчика системы для проведения таких соревнований!

Напишите запрос к базе данных, который возвращает всех пользователей вместе с некоторой дополнительной информацией:

- количеством соревнований, в которых он принял участие и решил там хотя бы одну задачу, соответствующую колонку в выводе следует назвать `solved_at_least_one_contest_count`,
- количеством соревнований, в которых он принял участие (то есть сделал хотя бы одну попытку решения), соответствующую колонку в выводе следует назвать `take_part_contest_count`.

Строки в выводе сортируйте в первую очередь по невозрастанию `solved_at_least_one_contest_count`, затем по невозрастанию `take_part_contest_count`, затем по возрастанию `id`.

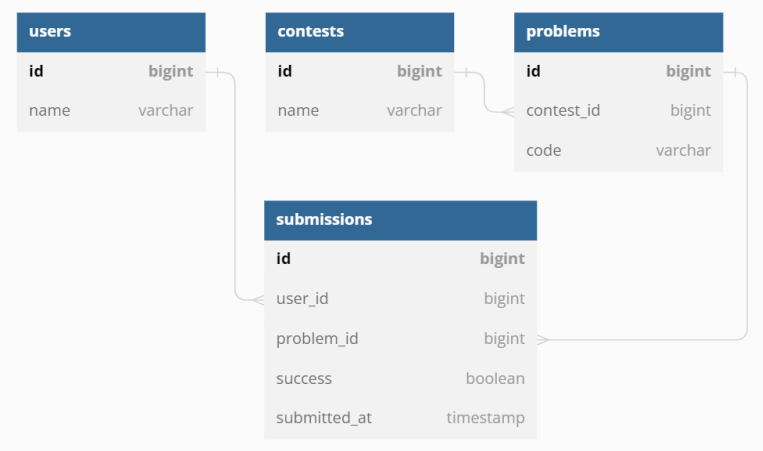
Внимательно ознакомьтесь с примерами вывода. Ваш запрос должен иметь **в точности** такой же вывод на примерах.

Схема базы данных содержит четыре таблицы:

- `users` — пользователи системы (описываются двумя полями: `id` и `name`),
- `contests` — контесты в системе (описываются двумя полями: `id` и `name`),
- `problems` — задачи в системе, каждая задача принадлежит одному контесту (описываются тремя полями: `id`, `contest_id` и `code`, где `code` — это кодовое короткое название задачи),
- `submissions` — отосланные попытки решения задач, каждая попытка принадлежит одной задаче и одному пользователю (описываются 5 полями: `id`, `user_id`, `problem_id`, `success` и `submitted_at`, где `success` — это булевское значение была ли попытка успешной и `submitted_at` — дата-время, когда попытка была совершена).

Таким образом, `contests` и `problems` находятся в отношении «один ко многим», `submissions` и `users` находятся в отношении «многие к одному», `submissions` и `problems` находятся в отношении «многие к одному».

Изучите входные данные примера, чтобы подробно ознакомиться со схемой базы данных. Диаграмма ниже иллюстрирует схему базы данных.



Входные данные

Входными данными в этой задаче является дамп базы данных. Вам он может быть полезен для ознакомления с состоянием базы данных для конкретного теста. В качестве решения вы должны отправить один SQL-запрос.

Выходные данные

Ваш SQL-запрос должен вывести всех пользователей вместе с дополнительными колонками, как написано выше. В точности следуйте требованиям к именованию дополнительных колонок и порядку сортировки.

Внимательно ознакомьтесь с примерами вывода. Ваш запрос должен иметь **в точности** такой же вывод на примерах.

входные данные

```
create table users (
  id bigint primary key,
  name varchar not null
);

create table contests (
  id bigint primary key,
  name varchar not null
);

create table problems (
  id bigint primary key,
  contest_id bigint,
  code varchar not null,
  constraint fk_problems_contest_id foreign key (contest_id)
references contests (id)
);

create unique index on problems (contest_id, code);

create table submissions (
  id bigint primary key,
  user_id bigint,
  problem_id bigint,
  success boolean not null,
  submitted_at timestamp not null,
  constraint fk_submissions_user_id foreign key (user_id)
references users (id),
  constraint fk_submissions_problem_id foreign key (problem_id)
references problems (id)
);

insert into users
values (1, 'Marie Curie'),
       (2, 'Stephen Hawking'),
       (3, 'Ada Lovelace'),
       (4, 'Albert Einstein'),
       (5, 'Archimedes');

insert into contests
values (1, 'Sandbox-Juniors'),
       (2, 'Sandbox-Seniors'),
       (3, 'Contest-Juniors'),
       (4, 'Contest-Seniors');

insert into problems
values (1, 1, 'A'),
       (2, 2, 'A'),
       (3, 3, 'A'),
       (4, 3, 'B'),
       (5, 4, 'A'),
       (6, 4, 'B');

insert into submissions
values (1, 2, 2, false, '2023-02-05 11:01:00'),
       (2, 2, 2, true, '2023-02-05 11:02:00'),
       (3, 2, 6, true, '2023-02-05 11:03:01'),
       (4, 2, 1, true, '2023-02-05 11:04:00'),
       (5, 2, 1, true, '2023-02-05 11:05:00'),
       (6, 3, 6, true, '2023-02-05 11:06:00'),
       (17, 1, 6, true, '2023-02-05 11:03:00'),
       (8, 1, 2, true, '2023-02-05 11:08:00'),
       (9, 1, 1, false, '2023-02-05 11:09:00'),
       (10, 3, 1, false, '2023-02-05 11:10:00'),
       (11, 5, 5, false, '2023-02-05 11:11:00'),
       (13, 2, 6, true, '2023-02-05 11:03:00'),
       (14, 3, 6, false, '2023-02-05 11:05:59'),
       (15, 1, 6, true, '2023-02-05 11:04:00');
```

выходные данные

```
 id |       name       | solved_at_least_one_contest_count |
take_part_contest_count
----+------------------+-----------------------------------+------
-------------------
  2 | Stephen Hawking |                                 3 |
3
  1 | Marie Curie     |                                 2 |
3
  3 | Ada Lovelace    |                                 1 |
2
  5 | Archimedes      |                                 0 |
1
  4 | Albert Einstein |                                 0 |
0
(5 rows)
```

входные данные

```
create table users (
  id bigint primary key,
  name varchar not null
);

create table contests (
  id bigint primary key,
  name varchar not null
);

create table problems (
  id bigint primary key,
  contest_id bigint,
  code varchar not null,
  constraint fk_problems_contest_id foreign key (contest_id)
references contests (id)
);

create unique index on problems (contest_id, code);

create table submissions (
  id bigint primary key,
  user_id bigint,
  problem_id bigint,
  success boolean not null,
  submitted_at timestamp not null,
  constraint fk_submissions_user_id foreign key (user_id)
references users (id),
  constraint fk_submissions_problem_id foreign key (problem_id)
references problems (id)
);

insert into users
values (1, 'Olivia'),
       (2, 'Henry'),
       (3, 'Lucas'),
       (4, 'John'),
       (5, 'Charlotte'),
       (6, 'Henry');

insert into contests
values (3, 'Main'),
       (1, 'Practice');

insert into problems
values (1, 3, 'A'),
       (2, 3, 'B'),
       (3, 1, 'A');

insert into submissions
values (10, 3, 2, false, '2023-02-05 11:05:12'),
       (20, 3, 2, true, '2023-02-05 11:07:49'),
       (30, 3, 2, true, '2023-02-05 11:07:49'),
       (40, 3, 1, false, '2023-02-05 11:01:32'),
       (50, 3, 1, false, '2023-02-05 11:11:46'),
       (60, 3, 1, false, '2023-02-05 11:27:05'),
       (70, 6, 2, false, '2023-02-05 11:04:00'),
       (80, 6, 2, true, '2023-02-05 11:05:00'),
       (90, 6, 2, false, '2023-02-05 11:06:00'),
       (100, 6, 2, true, '2023-02-05 11:07:00'),
       (110, 6, 1, false, '2023-02-05 11:08:00'),
       (120, 6, 1, true, '2023-02-05 11:09:00'),
       (130, 2, 2, false, '2023-02-05 11:00:01'),
       (150, 5, 1, false, '2023-02-05 11:07:48'),
       (160, 5, 1, true, '2023-02-05 11:07:49'),
       (170, 5, 1, true, '2023-02-05 11:07:50'),
       (180, 1, 3, false, '2023-02-04 15:00:01'),
       (190, 1, 3, true, '2023-02-04 15:00:01'),
       (200, 5, 3, true, '2023-02-04 15:00:01'),
       (210, 5, 3, false, '2023-02-04 15:00:01'),
       (220, 2, 3, false, '2023-02-04 15:00:01'),
       (230, 6, 3, false, '2023-02-04 15:00:01'),
       (240, 6, 3, false, '2023-02-04 15:00:01');
```

выходные данные

```
 id |    name     | solved_at_least_one_contest_count |
take_part_contest_count
----+-----------+-----------------------------------+------------
-------------
  5 | Charlotte |                                 2 |
2
  6 | Henry     |                                 1 |
2
  1 | Olivia    |                                 1 |
1
  3 | Lucas     |                                 1 |
1
  2 | Henry     |                                 0 |
2
  4 | John      |                                 0 |
0
(6 rows)
```

S3. Контест: таблица результатов (SQL, 30 баллов)

15 секунд, 1024 мегабайта

Это необычная задача — вам надо написать SQL-запрос. В качестве решения вы должны отослать один запрос к базе данных, который возвращает требуемые данные. Запрос может содержать произвольное количество подзапросов, других конструкций, быть сколь угодно навороченным, но это должен быть один запрос (в нём не должна встречаться точка с запятой для разделения разных запросов).

При проверке вашего решения используется PostgreSQL 15.1. В качестве входных данных вам предоставляется дамп состояния базы данных. Обратите внимание, что время работы вашего решения на тесте включает восстановление состояния базы данных из дампа, но это время значительно меньше ограничения по времени. Вы можете использовать сервис http://sqlfiddle.com/ как инструмент для запуска запросов.

В этой задаче вам предстоит написать запрос к базе данных простейшей системы проведения соревнований по программированию. Вы прямо сейчас участвуете в подобном соревновании. Время почувствовать себя в роли разработчика системы для проведения таких соревнований!

Напишите запрос к базе данных, который построит таблицу результатов для соревнования с максимальным `id`.

Вывод должен включать всех пользователей, кто сделал хотя бы одну попытку в этом соревновании. Вывод должен включать 5 колонок:

- `rank` — место пользователя в контесте (пользователи с одинаковыми результатами делят место);
- `user_id` — `id` пользователя;
- `user_name` — `name` пользователя;
- `problem_count` — количество решённых в контесте задач (если одна задача решена многократно, то всё-равно учитывается как одна задача);
- `latest_successful_submitted_at` — время, когда была решена последняя из решённых задач у этого пользователя (если одна задача решена многократно, то задача считается решённой в момент первого решения), иными словами, последний момент времени, когда у пользователя увеличилось количество решённых задач.

Строки следует сортировать по невозрастанию `problem_count`, затем по неубыванию `latest_successful_submitted_at`, затем по возрастанию `user_id`.

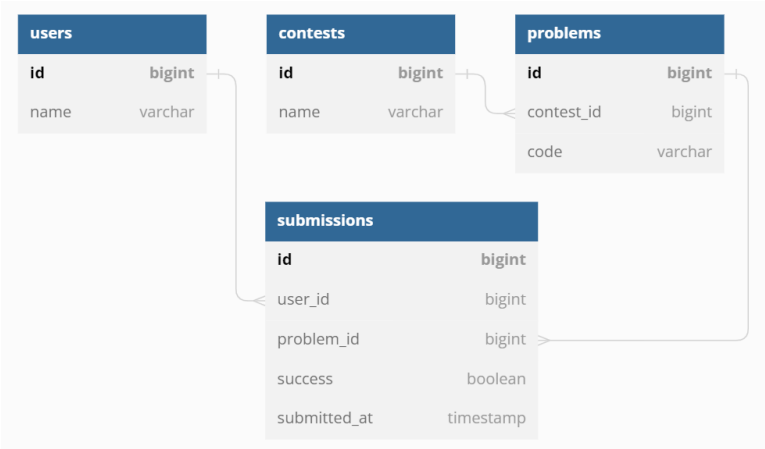
Пользователи, которые решили одинаковое количество задач (имеют равные `problem_count`) и имеют равные значения `latest_successful_submitted_at`, должны поделить одно место. Обратите внимание, что если несколько пользователей делят места, то в нумерации мест образуется разрыв. Например, если первое место делят два пользователя, то следующий пользователь должен получить место 3 (то есть последовательность мест имеет вид: $1, 1, 3$).

Внимательно ознакомьтесь с примерами вывода. Ваш запрос должен иметь **в точности** такой же вывод на примерах.

- `users` — пользователи системы (описываются двумя полями: `id` и `name`),
- `contests` — контесты в системе (описываются двумя полями: `id` и `name`),
- `problems` — задачи в системе, каждая задача принадлежит одному контесту (описываются тремя полями: `id`, `contest_id` и `code`, где `code` — это кодовое короткое название задачи),
- `submissions` — отосланные попытки решения задач, каждая попытка принадлежит одной задаче и одному пользователю (описываются 5 полями: `id`, `user_id`, `problem_id`, `success` и `submitted_at`, где `success` — это булевское значение была ли попытка успешной и `submitted_at` — дата-время, когда попытка была совершена).

Таким образом, `contests` и `problems` находятся в отношении «один ко многим», `submissions` и `users` находятся в отношении «многие к одному», `submissions` и `problems` находятся в отношении «многие к одному».

Изучите входные данные примера, чтобы подробно ознакомиться со схемой базы данных. Диаграмма ниже иллюстрирует схему базы данных.



Входные данные

Входными данными в этой задаче является дамп базы данных. Вам он может быть полезен для ознакомления с состоянием базы данных для конкретного теста. В качестве решения вы должны отправить один SQL-запрос.

Выходные данные

Ваш SQL-запрос должен вывести результаты соревнования с максимальным `id` в требуемом формате.

Внимательно ознакомьтесь с примерами вывода. Ваш запрос должен иметь **в точности** такой же вывод на примерах.

входные данные

```
create table users (
  id bigint primary key,
  name varchar not null
);

create table contests (
  id bigint primary key,
  name varchar not null
);

create table problems (
  id bigint primary key,
  contest_id bigint,
  code varchar not null,
  constraint fk_problems_contest_id foreign key (contest_id)
references contests (id)
);

create unique index on problems (contest_id, code);

create table submissions (
  id bigint primary key,
  user_id bigint,
  problem_id bigint,
  success boolean not null,
  submitted_at timestamp not null,
  constraint fk_submissions_user_id foreign key (user_id)
references users (id),
  constraint fk_submissions_problem_id foreign key (problem_id)
references problems (id)
);

insert into users
values (1, 'Marie Curie'),
       (2, 'Stephen Hawking'),
       (3, 'Ada Lovelace'),
       (4, 'Albert Einstein'),
       (5, 'Archimedes');

insert into contests
values (1, 'Sandbox-Juniors'),
       (2, 'Sandbox-Seniors'),
       (3, 'Contest-Juniors'),
       (4, 'Contest-Seniors');

insert into problems
values (1, 1, 'A'),
       (2, 2, 'A'),
       (3, 3, 'A'),
       (4, 3, 'B'),
       (5, 4, 'A'),
       (6, 4, 'B');

insert into submissions
values (1, 2, 2, false, '2023-02-05 11:01:00'),
       (2, 2, 2, true, '2023-02-05 11:02:00'),
       (3, 2, 6, true, '2023-02-05 11:03:01'),
       (4, 2, 1, true, '2023-02-05 11:04:00'),
       (5, 2, 1, true, '2023-02-05 11:05:00'),
       (6, 3, 6, true, '2023-02-05 11:06:00'),
       (17, 1, 6, true, '2023-02-05 11:03:00'),
       (8, 1, 2, true, '2023-02-05 11:08:00'),
       (9, 1, 1, false, '2023-02-05 11:09:00'),
       (10, 3, 1, false, '2023-02-05 11:10:00'),
       (11, 5, 5, false, '2023-02-05 11:11:00'),
       (13, 2, 6, true, '2023-02-05 11:03:00'),
       (14, 3, 6, false, '2023-02-05 11:05:59'),
       (15, 1, 6, true, '2023-02-05 11:04:00');
```

выходные данные

```
 rank | user_id |    user_name     | problem_count |
latest_successful_submitted_at
------+---------+------------------+---------------+--------------
-----------------
    1 |       1 | Marie Curie      |             1 | 2023-02-05
11:03:00
    1 |       2 | Stephen Hawking  |             1 | 2023-02-05
11:03:00
    3 |       3 | Ada Lovelace     |             1 | 2023-02-05
11:06:00
    4 |       5 | Archimedes       |             0 |
(4 rows)
```

входные данные

```
create table users (
  id bigint primary key,
  name varchar not null
);

create table contests (
  id bigint primary key,
  name varchar not null
);

create table problems (
  id bigint primary key,
  contest_id bigint,
  code varchar not null,
  constraint fk_problems_contest_id foreign key (contest_id)
references contests (id)
);

create unique index on problems (contest_id, code);

create table submissions (
  id bigint primary key,
  user_id bigint,
  problem_id bigint,
  success boolean not null,
  submitted_at timestamp not null,
  constraint fk_submissions_user_id foreign key (user_id)
references users (id),
  constraint fk_submissions_problem_id foreign key (problem_id)
references problems (id)
);

insert into users
values (1, 'Olivia'),
       (2, 'Henry'),
       (3, 'Lucas'),
       (4, 'John'),
       (5, 'Charlotte'),
       (6, 'Henry');

insert into contests
values (3, 'Main'),
       (1, 'Practice');

insert into problems
values (1, 3, 'A'),
       (2, 3, 'B'),
       (3, 1, 'A');

insert into submissions
values (10, 3, 2, false, '2023-02-05 11:05:12'),
       (20, 3, 2, true, '2023-02-05 11:07:49'),
       (30, 3, 2, true, '2023-02-05 11:07:49'),
       (40, 3, 1, false, '2023-02-05 11:01:32'),
       (50, 3, 1, false, '2023-02-05 11:11:46'),
       (60, 3, 1, false, '2023-02-05 11:27:05'),
       (70, 6, 2, false, '2023-02-05 11:04:00'),
       (80, 6, 2, true, '2023-02-05 11:05:00'),
       (90, 6, 2, false, '2023-02-05 11:06:00'),
       (100, 6, 2, true, '2023-02-05 11:07:00'),
       (110, 6, 1, false, '2023-02-05 11:08:00'),
       (120, 6, 1, true, '2023-02-05 11:09:00'),
       (130, 2, 2, false, '2023-02-05 11:00:01'),
       (150, 5, 1, false, '2023-02-05 11:07:48'),
       (160, 5, 1, true, '2023-02-05 11:07:49'),
       (170, 5, 1, true, '2023-02-05 11:07:50'),
       (180, 1, 3, false, '2023-02-04 15:00:01'),
       (190, 1, 3, true, '2023-02-04 15:00:01'),
       (200, 5, 3, true, '2023-02-04 15:00:01'),
       (210, 5, 3, false, '2023-02-04 15:00:01'),
       (220, 2, 3, false, '2023-02-04 15:00:01'),
       (230, 6, 3, false, '2023-02-04 15:00:01'),
       (240, 6, 3, false, '2023-02-04 15:00:01');
```

выходные данные

```
 rank | user_id | user_name | problem_count |
latest_successful_submitted_at
------+---------+-----------+---------------+--------------------
------------
    1 |       6 | Henry     |             2 | 2023-02-05 11:09:00
    2 |       3 | Lucas     |             1 | 2023-02-05 11:07:49
    2 |       5 | Charlotte |             1 | 2023-02-05 11:07:49
    4 |       2 | Henry     |             0 |
(4 rows)
```

Codeforces (c) Copyright 2010-2023 Михаил Мирзаянов
Соревнования по программированию 2.0