Содержание

Задача A. Max [max, 3 секунды, 256 мегабайт]

Есть массив целых неотрицательных чисел.

Найдите значение самого большого элемента.

Формат входных данных

Заголовочный файл называется «max.h»; смотрите примеры входных файлов. В ней определены два метода:

- `GetN()`: Не принимает аргументов, возвращает 64-битное целое число — количество блинов в стопке.
- `GetElement(i)`: Принимает 64-битное число i в пределах $0 \leqslant i <$ GetStackSize(). Возвращает 64-битное число: значение i-го элемента в тесте.

$1 \leqslant$ `GetN()` $\leqslant 10^9$, $0 \leqslant$ `GetElement(i)` $\leqslant 10^9$.

Формат выходных данных

Выведите единственное целое число — значение максимума.

Примеры

стандартный ввод	стандартный вывод
4 0 2 3 1	3
6 1 1 1 1 1 1	1
1 227	227

Задача B. Блины [pancakes, 2 секунды, 256 мегабайт]

В доме Бесконечных Блинчиков D голодных посетителей сидят за круглым столом. Они пронумерованы с 0, начиная с определенной точки стола, и до $D-1$ по часовой стрелке. Каждый посетитель любит определенный вид блинов, имеющих тип с таким же номером, как и тот посетитель.

Вы — официант со стопкой блинов. Вы сейчас стоите рядом с посетителем 0 и собираетесь ходить вокруг стола по часовой стрелки, возможно, несколько раз. Каждый раз, когда вы проходите мимо посетителя, и блин наверху стопки совпадает с предпочтениями этого посетителя, вы отдадите этот блин соответствующему едоку. Вы продолжаете так делать до тех пор, пока блин на вершине стопки не будет совпадать с текущим пользователем; в этом случае вы переходите к следующему едоку по часовой стрелке.

Каждый раз, когда вы проходите полный круг (переходите от посетителя $D-1$ к 0), вы проверяете, опустела ли ваша стопка блинов. Если это так, то вы заканчиваете свою работу. Иначе вы продолжаете обходить стол по часовой стрелке, раздавая блины. Заметьте, что могут существовать посетители, которым не достанется ни одного блинчика.

По заданной стопке блинов определите, сколько полных кругов вокруг стола вам придется сделать.

Формат входных данных

Библиотека называется «pancakes»; смотрите примеры входных файлов. В ней определены три метода:

- `GetStackSize()`: Не принимает аргументов, возвращает 64-битное целое число — количество блинов в стопке.
- `GetNumDiners()`: Не принимает аргументов, возвращает 64-битное целое число — количество посетителей за столом.
- `GetStackItem(i)`: Принимает 64-битное число i в пределах $0 \leqslant i <$ GetStackSize(). Возвращает 64-битное число: тип i-го сверху блина в стопке.

$1 \leqslant$ `GetStackSize()` $\leqslant 10^8$, $1 \leqslant$ `GetNumDiners()` $\leqslant 10^9$.

Формат выходных данных

Выведите единственное целое число — количество полных кругов вокруг стола.

Примеры

стандартный ввод	стандартный вывод
4 4 3 1 2 0	3
6 4 0 0 0 2 2 3	1
7 6 0 1 3 2 1 3 0	4

Задача C. Ящики [crates, ? секунд, 256 мегабайт]

Вы — менеджер склада на самом большом причале в округе. Ящики в этом складе организованы в один ряд, состоящий из столбцов ящиков. К сожалению, разгрузка кораблей — суетливое занятие, поэтому количество ящиков в столбцах может сильно различаться.

К вам скоро нагрянет ревизия, и вы хотите оставить хорошее впечатление. Вы решили использовать старый кран, чтобы переорганизовать ящики. Кран может выдержать только один ящик за раз, и может только переместить один ящик с одного столбца на какой-то из соседних. Кран начинает у самого левого столбца, в котором нет ящиков. Он может делать только последовательность шагов, которое мы назовем действием:

- Переместить кран к любому столбцу с хотя бы одним ящиком.
- Взять верхний ящик с этой стопки.
- Положить этот ящик в соседний столбец.

Вы должны использовать минимальное количество действий, чтобы все столбцы стали как можно ближе, а оставшиеся ящики были распределены среди левых столбцов. Если у вас N столбцов и C ящиков, вы хотите $C \bmod N$ левых столбцов из $\lceil \frac{C}{N} \rceil$ ящиков каждый, а все оставшиеся столбцы — из $\lfloor \frac{C}{N} \rfloor$ ящиков. Например, если $N = 3$ и $C = 8$, вы хотели бы, чтобы столбцы ящиков выглядели, как $3, 3, 2$ слева направо. Поскольку количество действий может быть большим, выведите его по модулю $10^9 + 7$.

Формат входных данных

Библиотека называется «crates»; смотрите примеры входных файлов. В ней определены три метода:

- `GetNumStacks()`: Не принимает аргументов, возвращает 64-битное целое число — количество столбцов ящиков на складе.
- `GetStackHeight(i)`: Принимает 64-битное число i в пределах $1 \leqslant i \leqslant$ GetStackSize(). Возвращает 64-битное число: начальное количество ящиков в i-м столбце.

  $1 \leqslant$ `GetNumStacks(), GetStackHeight(i)` $\leqslant 10^9$.

Формат выходных данных

Выведите единственное целое число — минимальное количество действий по модулю $10^9 + 7$.

3 2 2 4	3
4 1 2 5 1	5
3 2 2 2	0

Задача D. Камень-Ножницы-Бумага [rps, ? секунд, 256 мегабайт]

Вы проводите очередной турнир по игре в Камень-Ножницы-Бумагу. Это будет турнир на выбывание из N раундов. Участвуют 2^N игроков с уникальными номерами от 0 до $2^N - 1$ включительно.

Изначально, игроки расположены слева направо по возрастанию номеров. В каждом раунде, первый и второй игроки (начиная слева) играют между собой, затем третий и четвертый играют между собой и так далее (все матчи проходят одновременно). Победители этих матчей остаются в том же самом порядке, проигравшие уходят домой. Затем начинается турнир по тем же правилам. Так происходит, пока не останется один человек, этот игрок будет объявлен победителем.

Каждый матч играется по стандартным правилам игры Камень-Ножницы-Бумага. Вы устали от ничьих, поэтому, если игроки показывают одинаковый предмет, победителем матча считается игрок слева. Вы знаете, что игроки не очень сильны в стратегии, и у каждого есть предпочтительный предмет, который он будет всегда показывать. Если вы в курсе любимых предметов всех игроков, игрок с каким номером выиграет турнир?

Формат входных данных

Библиотека называется «rps»; смотрите примеры входных файлов. В ней определены три метода:

- `GetN()`: Не принимает аргументов, возвращает 64-битное целое число, в турнире участвует $2^{\text{GetN()}}$ — количество игроков в турнире.
- `GetFavoriteMove(i)`: Принимает 64-битное число i в пределах $0 \leqslant i < 2^{\text{GetN()}}$. Возвращает один из трех символов `R`, `P` или `S`: любимый предмет игрока с номером i;

$1 \leqslant$ `GetN()` $\leqslant 28$.

Формат выходных данных

Выведите единственное целое число — номер игрока, победившего в турнире.

Примеры

3 RPSRPSRP	5
2 RRRR	0
2 SRPP	2

Задача E. Запрос смерти [query_of_death, ? секунд, 256 мегабайт]

Мы запланировали простую задачку для вас: найти сумму нескольких чисел. Вы можете вызвать функцию `GetLength()`, чтобы узнать количество чисел, и функцию `GetValue(i)`, чтобы узнать i-е число. Для простоты, каждое из чисел равно 0 или 1. К сожалению, у нас случилась техническая проблема, но уже слишком поздно ее исправить.

Проблема в том, что существует такое i_{qod}, что если вызвать `GetValue(`i_{qod}`)` на каком-то узле, функция вернет верное значение, но любые следующие запросы `GetValue(`i`)` в том же узле вернут случайные значения, вне зависимости от i. Другие узлы не зависят от этого, но тоже, в свою очередь, могут сломаться от такого же вызова.

Значение i_{qod}, которое вызывает поломку, одинаково для всех узлов внутри одного теста (но может быть различным для разных тестов). Если узел ломается в одном тесте, то он будет снова работать корректно перед началом следующего теста.

Например, предположим, что мы имеем два корректных узла A и B, и два значенич i_{ok} и i_{qod}. Следующая последовательность вызовов приведет к следующим результатам:

- `GetValue(`i_{ok}`)` на узле A: возвращает корректное значение.
- `GetValue(`i_{qod}`)` на узле A: возвращает корректное значение, но узел A ломается.
- `GetValue(`i_{ok}`)` на узле B: возвращает корректное значение.
- `GetValue(`i_{ok}`)` на узле A: возвращает случайное значение.
- `GetValue(`i_{qod}`)` на узле A: возвращает случайное значение.
- `GetValue(`i_{qod}`)` на узле B: возвращает корректное значение, но узел B ломается.
- `GetValue(`i_{qod}`)` на узле B: возвращает случайное значение.
- `GetValue(`i_{ok}`)` на узле B: возвращает случайное значение.
- `GetValue(`i_{qod}`)` на узле A: возвращает случайное значение.
- `GetValue(`i_{ok}`)` на узле A: возвращает случайное значение.

Извиняемся за неудобство, но вы сможете найти сумму чисел?

Формат входных данных

Библиотека называется «query_of_death»; смотрите примеры входных файлов. В ней определены три метода:

- `GetLength()`: Не принимает аргументов, возвращает 64-битное целое число — количество чисел.
- `GetValue(i)`: Принимает 64-битное число i в пределах $0 \leqslant i <$ GetLength(). Возвращает 32-битное число (всегда либо 0, либо 1): значение i-го числа, если узел не сломан, и случайное значение из $\{0, 1\}$ иначе.

$1 \leqslant$ `GetLength()` $\leqslant 10^8$

Формат выходных данных

Выведите единственное целое число — сумму чисел. Смотрите тестовые примеры, имитирующие процесс поломки узлов. Реальные примеры имеют похожее, но не обязательно совпадающее поведение.