VK Cup 2019 - Квалификация (Engine)

Statement is not available on English language

A. Скрытый друг

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Вам дан граф друзей VK. Недавно у пользователей появилась возможность скрывать друзей из социального графа. Для заданного графа друзей найдите скрытые дружеские связи, то есть такие ситуации, когда пользователь u находится в друзьях у пользователя v, но пользователь v не находится в друзьях у пользователя u.

Входные данные

В первой строке задано одно целое число n ($2 \le n \le 100$) — количество человек в графе друзей ВКонтакте. Пользователи пронумерованы целыми числами от 1 до n.

В следующих n строках дан граф друзей каждого из этих людей: в i-й из этих строк сначала дано количество друзей у i-го человека и список номеров его друзей, разделенные пробелами. Номера друзей в каждой из n этих строк не повторяются.

Выходные данные

В первой строке выведите одно число k — количество скрытых дружеских связей.

В следующих k строках выведите пары чисел u, v, означающие, что пользователь u скрыл пользователя v из друзей. Пары выводите в любом порядке.

Примеры

входные данные
5 3 2 3 4 4 1 3 4 5 0 2 1 2 3 4 3 1
выходные данные
6 3 5 4 5 5 2 3 1 1 5 3 2

входные данные
2 0 1 1
выходные данные
1 1 2

Statement is not available on English language

B. Code Review

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

У команды разработки движков ВКонтакте есть общий чат. После каждого сложного коммита в репозиторий, автор этого коммита присылает в чат сообщение с предложением провести ревью. Для одобрения или отклонения коммита достаточно, чтобы его проверил один разработчик, не принимавший участие в его написании.

Перед тем, как оставить новую заявку на ревью, каждый разработчик проводит ревью последней из оставленных заявок,

которые еще никем не проверены (если может — если он не является автором этой заявки) и только после этого оставляет новую заявку.

Вам дан лог заявок на ревью с момента появления чата в хронологическом порядке. Найдите коммиты, которые никто не проверил.

Входные данные
В первой строке записано одно целое число n ($1 \le n \le 50\,000$) — количество просьб о code review в чате.

В следующих n строках записаны внутренний целочисленный идентификатор разработчика i и хеш коммита h ($1 \le i \le 50\,000$; h состоит из строчных букв латинского алфавита от a до f и цифр). Все хеши коммитов уникальны и имеют длины от 1 до 20 символов, включительно.

Выходные данные
Выведите все хеши коммитов, которые не попали на ревью, в том же порядке, в котором они были даны во входных данных.

Пример

входные данные
7 1 0e813c50 1 00e9422b 1 520cb7b4 2 052dd9ad 3 9dd5f347 3 e35f067b 1 bb4d4a99
выходные данные
0e813c50 00e9422b 9dd5f347 bb4d4a99

Statement is not available on English language

C. #define Задача B ...

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Компания «ВКонтакте» активно использует языки C/C++ для разработки движков. Старший разработчик Вася — большой любитель языка C, ведь только в нем можно полностью раскрыть потенциал Define-Oriented Programming. В языке C директива `#define` позволяет сделать прямую подстановку куска кода в любое место программы. Например, при исполнении следующего кода значение переменной v будет равно 11 (в переменную v запишется результат выражения $3 + 4 \cdot 2$).

```
#define add(x) + x
#define mul(y) * y
int v = 3 add(4) mul(2);
```

Недавно Вася написал небольшую программу, которая заполняет массив большой длины. Программа выглядит следующим образом:

```
#define A0(x) x,
#define A1(x) A0(x) A0(x + 1) A0(x + 3) A0(x + 4)
#define A2(x) A1(x) A1(x + 1) A1(x + 3) A1(x + 4)
#define A3(x) A2(x) A2(x + 1) A2(x + 3) A2(x + 4)
#define A4(x) A3(x) A3(x + 1) A3(x + 3) A3(x + 4)
...
#define A24(x) A23(x) A23(x + 1) A23(x + 3) A23(x + 4)
#define A25(x) A24(x) A24(x + 1) A24(x + 3) A24(x + 4)

const long long values[1125899906842624] = { A25(0) };
```

К сожалению, его программа не компилируется в силу несовершенства его компьютера, но ему очень интересно знать, какие значения лежали бы в массиве $values$, если бы ему удалось скомпилировать и запустить программу. Помогите ему это узнать.

Входные данные
В первой строке дано одно целое число n ($1 \le n \le 1\,000$) — количество элементов массива, значения которых интересуют Васю.

Следующие n строк содержат n целых чисел pos_i ($0 \le pos_i \le 1\,125\,899\,906\,842\,623$) — позиции, для которых нужно узнать

значение в массиве.

Выходные данные
Выведите n строк. В i-й строке должно содержаться значение элемента массива на позиции pos_i.

Пример

входные данные
5 0 1 3 5 8
выходные данные
0 1 4 2 3

Примечание
Начало этого массива выглядит следующим образом:

```
const long long values[1125899906842624] = { 0, 1, 3, 4, 1, 2, 4, ... };
```

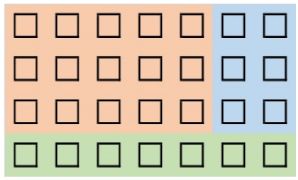
Statement is not available on English language

D. Storage2

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Для надежного, но компактного хранения картинок во ВКонтакте используются избыточные коды. Все картинки разбиваются на группы по 15, и для каждой группы картинок вычисляются 13 избыточных кодов. Избыточные коды необходимы, чтобы суметь восстановить некоторые картинки, если они будут утеряны вследствие поломки жесткого диска.

Перед вычислением избыточных кодов 15 картинок из одной группы записываются в матрицу из 3 строк и 5 столбцов. Далее к получившейся матрице справа добавляются два столбца с 6 избыточными кодами (каждая пара кодов вычисляется на основе картинок в той же строке), а затем снизу добавляется еще одна строка из 7 избыточных кодов (каждый из них считается на основе картинок или избыточных кодов в том же столбце).



Будем называть картинки и избыточные коды блоками. Таким образом, из 15 картинок мы получили 28 блоков, расставленных в виде матрицы 4 на 7. В данной задаче вам не будут даны формулы вычисления избыточных кодов, но даны их свойства: а именно по 3 любым блокам из одного столбца можно однозначно восстановить утерянный четвертый блок в том же столбце, а также по 5 любым блокам из одной строки можно однозначно восстановить оставшиеся два из той же строки, если они были утеряны. Естественно, операцию восстановления можно применять несколько раз к разным строкам и столбцам в любом порядке, а также в следующих операциях использовать блоки, восстановленные ранее.

Пусть у вас есть 28 блоков (15 картинок и 13 избыточных кодов), и вы потеряли случайные k из них (равновероятно среди всех вариантов). Ваша задача посчитать вероятность, что хотя бы одну из 15 картинок будет невозможно восстановить, а также матожидание количества потерянных картинок.

Входные данные
В единственной строке входных данных записано целое число k ($0 \le k \le 28$) — количество потерянных блоков.

Выходные данные
Выведите два числа — вероятность, что хотя бы одну картинку невозможно восстановить из оставшихся блоков, и матожидание количества картинок, которые невозможно восстановить из оставшихся блоков.

Ответы будут считаться корректными, если их абсолютная или относительная погрешность не превосходит 10^{-9}.

Примеры

входные данные
28

выходные данные
1.0000000000000000000 15.0000000000000000000

входные данные
6
выходные данные
0.00055741360089186175 0.00179168657429526999

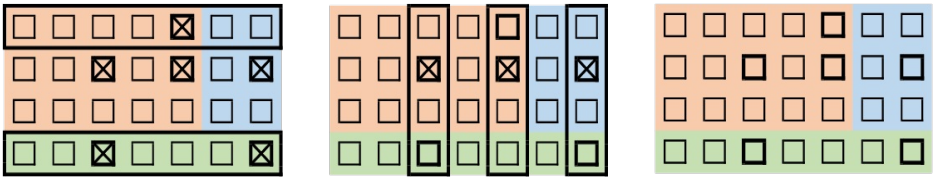
Примечание

В первом примере утеряны все 28 блоков, очевидно, с 100% вероятностью потеряны все 15 картинок.

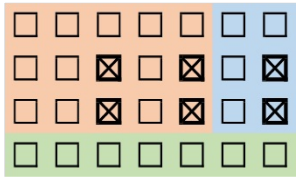
Во втором примере только в 210 случаев из 376740 будет невозможно восстановить хотя бы одну картинку, поэтому вероятность $210/376740$. В зависимости от расположения утерянных блоков будет невозможно восстановить от 1 до 6 картинок, матожидание равно $15/8372$.

Приведены два возможных способа потерять 6 блоков.

На первом случае утеряны 3 картинки и 3 избыточных кода (отмечены крестиками). Вначале мы можем восстановить один утерянный блок в первой строке (используя 5 других из этой же строки) и восстановить два утерянных блока в четвертой строке (используя 5 других из той же строки). Затем мы можем восстановить три оставшихся блока (используя 3 других блока из соответствующих столбцов). Заметим, что при восстановлении по столбцу мы использовали в том числе и восстановленные на первом шаге блоки. Таким образом, нам удалось восстановить все блоки и не было потеряно ни одной картинки.



На втором случае утеряны 4 картинки и 2 избыточных кода. Во второй и третьей строках по три утерянных блока, а значит мы не можем однозначно восстановить ни один из блоков в них. Аналогично, в третьем, пятом и седьмом столбцах по два утерянных блока, а значит мы не можем однозначно восстановить ни один из блоков в них. В остальных строках и столбцах все блоки остались целыми, восстанавливать нечего. Таким образом, нам не удастся восстановить ни один из шести утерянных блоков, но только из **четыре** из них были картинками.



Statement is not available on English language

E1. Контрольная сумма

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Данные пользователей ВКонтакте хранятся на десятках тысяч серверов. Для того, чтобы можно было определять ошибки при записи данных на диск, на диск регулярно записываются текущие контрольные суммы CRC32 (Wiki, IEEE 802-3). Благодаря этому, при чтении данных можно заново вычислить контрольную сумму и проверить, что данные и контрольная сумма записаны корректно.

Разумеется, проверки на совпадение контрольной суммы встроены в большинство сервисов ВКонтакте. Но как-то раз оказалось, что в одном сегменте данных нужно изменить значение четырех последовательных байт на новое — нужно заменить значения последовательности $a_i, a_{i+1}, a_{i+2}, a_{i+3}$ на x_0, x_1, x_2, x_3. При этом, нужно чтобы значение контрольной суммы CRC32 осталось прежним. Конечно, при изменении последовательности ее контрольная сумма изменится, поэтому кроме изменения этих четырех байт на новое значение, были выбраны четыре байта $a_j, a_{j+1}, a_{j+2}, a_{j+3}$, которым можно назначить любое значение. Ваша задача выбрать им новые значения так, чтобы CRC32 данной последовательности не изменился, или определить, что это невозможно. Поскольку изменение данных — операция серьезная, перед самим изменением нужно определить, как изменится последовательность для q независимых тестовых случаев.

Обратите внимание, что в этой версии задачи есть всего один тест с $n = 16, q = 8$, который вы можете скачать по этой ссылке. Ваше решение не обязано проходить тесты из условия, и не будет на них протестировано. Если вы хотите проверить ваше решение на тестах из условия, отправляйте его в задачу E3, где первые два теста соответствуют примерам из условия.

Входные данные

В первой строке дано два целых числа n и q — количество байт в файле и количество запросов, для которых нужно решить задачу ($8 \le n \le 2 \cdot 10^5$; $1 \le q \le 10^5$).

Во второй строке дано n чисел $a_0, a_1, \ldots, a_{n-1}$ — содержимое файла в байтах ($0 \le a_i \le 255$).

В следующих q строках дано по шесть чисел i, j, x_0, x_1, x_2, x_3 — позиция i, начиная с которой нужно заменить четыре байта на x_0, x_1, x_2, x_3, и позиция j, начиная с которой можно менять четыре байта как угодно ($0 \le i, j \le n - 4$; $0 \le x_0, x_1, x_2, x_3 \le 255$). Гарантируется, что отрезки $[i; i + 3]$ и $[j; j + 3]$ не пересекаются.

Выходные данные
Для каждого запроса выведите четыре целых числа z_0, z_1, z_2, z_3, на которые нужно заменить четыре байта с номерами $j, j + 1, j + 2, j + 3$, чтобы crc32 не изменился. Обратите внимание, что все запросы независимы, и на самом деле последовательность не изменяется.

Если существует несколько решений, выведите любое, а если для данного запроса валидного решения нет, выведите `No solution`.

Примеры

входные данные
8 1 1 2 3 4 5 6 7 8 0 4 0 0 0 0
выходные данные
212 34 127 159

входные данные
16 3 4 5 6 7 0 0 0 0 0 0 0 0 85 200 47 47 0 11 0 0 0 0 0 3 12 7 0 0 0 0 11 0 0 0 0
выходные данные
0 0 0 0 200 47 47 0 0 0 0 0

Примечание
CRC32 байтовой последовательности из первого примера (`1 2 3 4 5 6 7 8`) равен `3fca88c5`, CRC32 измененной последовательности (`0 0 0 0 212 34 127 159`) так же равен `3fca88c5`. Стандартная утилита crc32 из большинства дистрибутивов линукса должна посчитать от них данную контрольную сумму.

CRC32 последовательности из второго примера равен `ecbb4b55`.

Statement is not available on English language

E2. Контрольная сумма

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Данные пользователей ВКонтакте хранятся на десятках тысяч серверов. Для того, чтобы можно было определять ошибки при записи данных на диск, на диск регулярно записываются текущие контрольные суммы CRC32 (Wiki, IEEE 802-3). Благодаря этому, при чтении данных можно заново вычислить контрольную сумму и проверить, что данные и контрольная сумма записаны корректно.

Разумеется, проверки на совпадение контрольной суммы встроены в большинство сервисов ВКонтакте. Но как-то раз оказалось, что в одном сегменте данных нужно изменить значение четырех последовательных байт на новое — нужно заменить значения последовательности $a_i, a_{i+1}, a_{i+2}, a_{i+3}$ на x_0, x_1, x_2, x_3. При этом, нужно чтобы значение контрольной суммы CRC32 осталось прежним. Конечно, при изменении последовательности ее контрольная сумма изменится, поэтому кроме изменения этих четырех байт на новое значение, были выбраны четыре байта $a_j, a_{j+1}, a_{j+2}, a_{j+3}$, которым можно назначить любое значение. Ваша задача выбрать им новые значения так, чтобы CRC32 данной последовательности не изменился, или определить, что это невозможно. Поскольку изменение данных — операция серьезная, перед самим изменением нужно определить, как изменится последовательность для q независимых тестовых случаев.

Обратите внимание, что в этой версии задачи есть всего один тест с $n = 50\,000, q = 8$, который вы можете скачать по этой ссылке. Ваше решение не обязано проходить тесты из условия, и не будет на них протестировано. Если вы хотите проверить ваше решение на тестах из условия, отправляйте его в задачу E3, где первые два теста соответствуют примерам из условия.

Входные данные
В первой строке дано два целых числа n и q — количество байт в файле и количество запросов, для которых нужно решить

задачу ($8 \le n \le 2 \cdot 10^5$; $1 \le q \le 10^5$).

Во второй строке дано n чисел $a_0, a_1, \ldots, a_{n-1}$ — содержимое файла в байтах ($0 \le a_i \le 255$).

В следующих q строках дано по шесть чисел i, j, x_0, x_1, x_2, x_3 — позиция i, начиная с которой нужно заменить четыре байта на x_0, x_1, x_2, x_3, и позиция j, начиная с которой можно менять четыре байта как угодно ($0 \le i, j \le n - 4$; $0 \le x_0, x_1, x_2, x_3 \le 255$). Гарантируется, что отрезки $[i; i + 3]$ и $[j; j + 3]$ не пересекаются.

Выходные данные

Для каждого запроса выведите четыре целых числа z_0, z_1, z_2, z_3, на которые нужно заменить четыре байта с номерами $j, j + 1, j + 2, j + 3$, чтобы crc32 не изменился. Обратите внимание, что все запросы независимы, и на самом деле последовательность не изменяется.

Если существует несколько решений, выведите любое, а если для данного запроса валидного решения нет, выведите `No solution`.

Примеры

входные данные
8 1 1 2 3 4 5 6 7 8 0 4 0 0 0 0
выходные данные
212 34 127 159

входные данные
16 3 4 5 6 7 0 0 0 0 0 0 0 0 85 200 47 47 0 11 0 0 0 0 0 3 12 7 0 0 0 0 11 0 0 0 0
выходные данные
0 0 0 0 200 47 47 0 0 0 0 0

Примечание

CRC32 байтовой последовательности из первого примера (`1 2 3 4 5 6 7 8`) равен `3fca88c5`, CRC32 измененной последовательности (`0 0 0 0 212 34 127 159`) так же равен `3fca88c5`. Стандартная утилита crc32 из большинства дистрибутивов линукса должна посчитать от них данную контрольную сумму.

CRC32 последовательности из второго примера равен `ecbb4b55`.

Statement is not available on English language

E3. Контрольная сумма

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Данные пользователей ВКонтакте хранятся на десятках тысяч серверов. Для того, чтобы можно было определять ошибки при записи данных на диск, на диск регулярно записываются текущие контрольные суммы CRC32 (Wiki, IEEE 802-3). Благодаря этому, при чтении данных можно заново вычислить контрольную сумму и проверить, что данные и контрольная сумма записаны корректно.

Разумеется, проверки на совпадение контрольной суммы встроены в большинство сервисов ВКонтакте. Но как-то раз оказалось, что в одном сегменте данных нужно изменить значение четырех последовательных байт на новое — нужно заменить значения последовательности $a_i, a_{i+1}, a_{i+2}, a_{i+3}$ на x_0, x_1, x_2, x_3. При этом, нужно чтобы значение контрольной суммы CRC32 осталось прежним. Конечно, при изменении последовательности ее контрольная сумма изменится, поэтому кроме изменения этих четырех байт на новое значение, были выбраны четыре байта $a_j, a_{j+1}, a_{j+2}, a_{j+3}$, которым можно назначить любое значение. Ваша задача выбрать им новые значения так, чтобы CRC32 данной последовательности не изменился, или определить, что это невозможно. Поскольку изменение данных — операция серьезная, перед самим изменением нужно определить, как изменится последовательность для q независимых тестовых случаев.

Входные данные

В первой строке дано два целых числа n и q — количество байт в файле и количество запросов, для которых нужно решить задачу ($8 \le n \le 2 \cdot 10^5$; $1 \le q \le 10^5$).

Во второй строке дано n чисел $a_0, a_1, \ldots, a_{n-1}$ — содержимое файла в байтах ($0 \le a_i \le 255$).

В следующих q строках дано по шесть чисел i, j, x_0, x_1, x_2, x_3 — позиция i, начиная с которой нужно заменить четыре байта на x_0, x_1, x_2, x_3, и позиция j, начиная с которой можно менять четыре байта как угодно ($0 \le i, j \le n - 4$;

$0 \le x_0, x_1, x_2, x_3 \le 255$). Гарантируется, что отрезки $[i; i+3]$ и $[j; j+3]$ не пересекаются.

Выходные данные
Для каждого запроса выведите четыре целых числа z_0, z_1, z_2, z_3, на которые нужно заменить четыре байта с номерами $j, j+1, j+2, j+3$, чтобы crc32 не изменился. Обратите внимание, что все запросы независимы, и на самом деле последовательность не изменяется.

Если существует несколько решений, выведите любое, а если для данного запроса валидного решения нет, выведите `No solution`.

Примеры

входные данные
8 1 1 2 3 4 5 6 7 8 0 4 0 0 0 0
выходные данные
212 34 127 159

входные данные
16 3 4 5 6 7 0 0 0 0 0 0 0 0 85 200 47 47 0 11 0 0 0 0 0 3 12 7 0 0 0 0 11 0 0 0 0
выходные данные
0 0 0 0 200 47 47 0 0 0 0 0

Примечание
CRC32 байтовой последовательности из первого примера (1 2 3 4 5 6 7 8) равен 3fca88c5, CRC32 измененной последовательности (0 0 0 0 212 34 127 159) так же равен 3fca88c5. Стандартная утилита crc32 из большинства дистрибутивов линукса должна посчитать от них данную контрольную сумму.

CRC32 последовательности из второго примера равен ecbb4b55.

Statement is not available on English language

F. Шардирование постов

ограничение по времени на тест: 2 секунды
ограничение по памяти на тест: 512 мегабайт
ввод: стандартный ввод
вывод: стандартный вывод

Это интерактивная задача.

Когда данных становится слишком много и они не помещаются на один сервер, их приходится шардировать. Рассмотрим систему хранения постов пользователей, которая расположена на S серверах, нумеруемых с единицы. Каждый раз когда пользователь пишет пост, ему выдается уникальный идентификатор в пределах от 1 до 10^{18} и сохраняется на случайный сервер. Чем позже был создан пост, тем больше его id. Иногда посты удаляются, поэтому на серверах может лежать существенно разное количество постов.

Рассмотрим все неудаленные id постов пользователя и отсортируем их по возрастанию. Вам хочется узнать k-й из них. Для этого вы можете послать не более 100 дополнительных запросов. Каждый запрос имеет формат «? i j». В ответ вы получите идентификатор j-го по возрастанию поста пользователя среди хранящихся на i-м сервере. Когда вы считаете, что знаете k-й по возрастанию идентификатор, вместо запроса необходимо вывести ответ в формате «! id».

Протокол взаимодействия
В первой строке записано два числа n и S ($1 \le n \le 100, 1 \le S \le 5$) — количество пользователей для которых необходимо независимо решить задачу, а также количество серверов, на которых хранятся посты.

Далее необходимо n раз решить задачу. Вначале необходимо считать S чисел $a_1, a_2, \ldots a_S$ ($0 \le a_i; \sum a_i \le 10^5$) — количество постов пользователя на каждом сервере. В следующей строке будет задано число k ($1 \le k \le \sum a_i$) — идентификатор какого по возрастанию поста необходимо узнать.

Далее необходимо совершить не более 100 запросов в формате, который описан в условии.

Заметим, что ограничение на количество запросов действует на каждого пользователя отдельно, поэтому при переходе к следующему тесту счетчик вопросов сбрасывается.

Пример

входные данные

1 2 3 2 3 3 5 10
выходные данные
? 1 2 ? 2 1 ? 1 3 ! 5

Примечание

В примере на первом сервере хранятся посты с id 1, 3 и 10. А на втором 5 и 7. Необходимо найти третье по возрастанию число, это 5.