Even
(Анализ)

Задачата има няколко възможни подхода, които биха дали различен брой точки на състезателите.

Тривиален
Просто обхождаме всички числа в дадения интервал и за всяко от тях проверяваме дали има по четен брой цифри. Лесно решение, но за съжаление много бавно, тъй като има линейна сложност по големината на интервала. Това решение би дало на състезателите около 15 точки.

Рекурсивен с прунинг
Можем малко да оптимизираме горното, като напишем рекурсия, която да търси числата. Плюс тук е, че можем да спрем рекурсията рано, ако видим, че няма как резултатът да е с четен брой от всяка от цифрите. Например, нека по време на рекурсията сме генерирали число, което има 4 цифри, които се срещат нечетен брой пъти, но само три оставащи позиции, преди да стане по-голямо от **R**. Очевидно няма как да го довършим по валиден начин, съответно си спестяваме рекурсията в тези близо 1000 клона на рекурсията.

Бидейки малко по-бързо, това решение хваща около 25 точки (но все още е бавно, тъй като се нуждае от поне толкова операции, колкото е отговорът за съответния тест – а реално много повече).

Генериране
Развивайки идеята от рекурсивното генериране, можем да видим, че немалко време отива в "задънени" улици – дори да прекъсваме рекурсията рано, което ползвахме като оптимизация в предния вариант, все пак това отнема време (и е огромна част от числата, които генерираме). Можем ли да ги генерираме така, че да си гарантираме, че те няма да са невалидни? Ами, всъщност – можем!

Първо трябва да съобразим, че за да може едно число да съдържа четен брой от всяка от цифрите си, то самото число трябва да е с четен брой цифри. Така можем да генерираме само числа с по 2, 4, 6, … 18 цифри.

Нека сме фиксирали броя на цифрите, които текущото генерирано число ще има. Нека намерим първата позиция, на която още не сме сложили цифра. Нека пробваме да сложим там всяка една цифра, и след това намерим другата позиция, с която тя ще се "чифтоса". Трябва да внимаваме да не генерираме повтарящи се числа (тъй като, например, ако сложим цифра 4 на първа и пета позиция, после на трета и десета, то е същото, като да сложим цифра 4 на първа и трета и после на пета и десета).

Така генерираме само красиви числа, но трябва за всяко от тях да проверим дали е в интервала [**L**, **R**]. Тъй като това отново е подход с директно броене, той зависи от големината на отговора. Тъй като самият отговор може да е много голям

(примерно, за максималния интервал от 1 до 10^{18} е 2,105,532,412,794,693) то това решение е гарантирано бавно за големи интервали. За него са предвидени около 30 точки.

Прекалкулация
Локално можем да прекалкулираме колко хубави числа има (да кажем, ограничени до 1,000,000,000) и да запишем в масив в кода ни, за някакви интервали (примерно, за групи от по 100,000 числа: [0, 99999], [100000, 199999], … [999900000, 999999999]). За да обработим даден въпрос, неговите L и R трябва да са по-малки от 1,000,000,000. Изчисляваме по глупавия начин всички числа от непълен интервал (в началото и края на [L, R]), а по средата взимаме директно сметнатите бройки, което е в най-лошия случай по около 300,000 операции на въпрос (често много по-малко). Това решение също ще хване около 30 точки.

Динамично оптимиране
Реалното решение, което се очакваше от състезателите, е базирано на техниката динамично оптимиране.

Често в задачи, в които трябва да сметнем нещо за даден интервал [**L**, **R**], е по-лесно вместо това да го сметнем за интервала [1, **R**] и след това да извадим бройката за [1, **L** – 1]. Тук това също е така, затова ще разгледаме как да решим задачата за интервал [1, X].

Ще генерираме числото от най-старшите към най-младшите цифри, за да можем да следим дали генерираното число е по-малко или равно на X. Ако лимитът ни X е, да кажем, 133742, и до сега сме генерирали първите две цифри да са 13????, то знаем, че на трета позиция можем да ползваме само 0, 1, 2 или 3 – в противен случай, генерираното число би станало по-голямо от X. Ако пък сме генерирали 12????, то на трета позиция можем да имаме всяка една цифра – включително по-големи от 3, тъй като числото вече е гарантирано по-малко от 133742.

До тук стейтът на динамичното трябва да пази до коя позиция на генерираното число сме стигнали, и дали то вече е строго по-малко от X, което е [18][2] ([index][smaller]) като размерност.

Макар и водещи нули да не са разрешени, ние искаме да позволим "празни" позиции (за да генерираме числа с различен брой цифри). Можем да симулираме това като разрешим водещи нули, но не ги считаме към бройката за четност. Така стейтът ни би станал [18][2][2] ([index][smaller][started]).

Финално, трябва и все пак да пазим за всяка цифра дали се среща четен или нечетен брой пъти до момента. Тъй като имаме 10 възможни цифри, това можем лесно да направим в битова маска с 10 бита, като ако даден бит е 0, то имаме четен брой срещания. Ако имаме 1, то дадената цифра се среща нечетен брой пъти. Така финалният стейт става [18][2][2][1024] ([index][smaller][started][mask]).

В случая, за всеки индекс трябва да пробваме всяка възможна цифра, като така сложността ни става 10 * 18 * 2 * 2 * 1024 = 737,280, което е много малко операции за модерен компютър – решението работи буквално за милисекунди на въпрос.

Автор: Александър Георгиев