	На пълното решение	На частичните решения
Тагове	Динамично оптимиране по цифри	Префиксни суми Пълни изчерпвания Математични наблюдения

Анализ

Към задачата може да се подходи по два начина. Ще разгледам възможните решения с двата подхода. Ще обознача броя начини за представяне на числото x в нашата бройна система с $f(x)$.

Първи подход

Първия подход не е за пълен брой точки (стига до 75) и е по скоро математичен. Ако искате, може директно да отскочите на втория подход.

Логично е да разгледаме възможностите за последна цифра на някое число x в нашата двоична бройна система. Нека разгледаме следните два случая:

- x е нечетно, следователно x завършва на 1 в побитовото си представяне. Следователно задължително последната му цифра в нашия двоичен запис ще трябва да бъде 1. Заради това ние може да си представим, че изтриваме тази цифра и преброяваме по колко начина може да се представи останалата част от x. Следователно броя начини да се представи x е равен на броя начини да се представи $\frac{(x-1)}{2}$.
- x е четно, следователно x завършва на 0 в побитовото си представяне. Следователно задължително последната му цифра в нашия двоичен запис ще трябва да бъде 0 или 2. Нека последната му цифра да е 0. Тогава може да си представим, че изтриваме последната цифра и преброяваме начините за $\frac{x}{2}$. Ако пък последната цифра е 2, то ще има пренос. Следователно може да си представим как изтриваме последата цифра и изваждаме 1 от останалото число. Тогава преброяваме начините за $\frac{x}{2} - 1$.

TLDR: $f(2k+1) = f(k); f(2k) = f(k) + f(k-1)$

Нека видим как може да използваме това.

Решение за $r_i \leq 10^6$

Може да изчислим $f(x)$ за всяко $1 \leq x \leq 10^6$ с горната рекурентна зависимост по лесен начин. Чрез префиксни суми може лесно да се поддържат сумите в интервал.

Постигната сложност: $O(\max r_i + Q)$

Имплементация: `binaryAlt_45p.cpp`

Решение за $r_i \leq 10^9$

Нека означим $g(x) = f(1) + f(2) + \cdots + f(x)$. Тогава $g(0) = 0$. За да изчислим сумата $f(l) + f(l+1) + \cdots + f(r)$, ние може първо да сметнем поотделно $g(l-1)$ и $g(r)$ и след това да извадим първото от второто. Нека видим как да сметнем $g(x)$ за някое $x \leq 10^9$. Всъщност нека разгледаме отново два случая:

- $x = 2k + 1$. Ако Бог е милостив, ние може да забележим, че равенството $g(2k+1) = 3 \times g(k) + 2 - f(k)$ важи. Равенството важи за $k = 0$, защото $g(1) = 3 * g(0) + 2 - f(0) = 0 + 2 - 1 = 1$.
- $x = 2k$. Тогава може да си улесним живота като $g(2k) = g(2k+1) - f(2k+1) = g(2k+1) - f(k) = 3 \times g(k) + 2 - 2 \times f(k)$.
- Ще оставя за упражнение да докажете по индукция останалото ☺. Само трябва да се поразпише малко.

Има различни подходи, които изкарват 75 точки. Този, които ми изглежда най-лесен беше да се оптимизира смятането на f и g с precalc на f до 4×10^7 и поддържане на пресметнатите стойности на f над 4×10^7 във unordered map. Решението не работи за 100 точки, защото тогава липсва плюсът, че може да precalc-нем голяма част от нужните f-ове.

Постигната сложност: $O(precalcSize + Q \max r_i / precalcSize)$

Имплементация: `binaryAlt2_75p.cpp`

Втори подход

Да се намери сбор на някакви неща под някакъв модул. Логично е да е някакъв вид динамично.

Решение за $r_i \leq 10^6$

Нека да намерим начин за пресмятане на $f(x)$. Нека да разгледаме възмостите за първата цифра на x в нашия двоичния запис. Ако първата цифра в побитовото представяне на x е 0, то първата цифра в двоичния запис задължително трябва да е също 0. Ако пък първата цифра в побитовото представяне е 1, то задължително първата цифра не може да е 2. Тогава първата цифра може хем да е 1, хем да е 0 – тогава ще искаме „пренос“ от следващите цифри. Така може да направим динамично със стейт `dp[digit][prenos]`, което изчислява $f(x)$ спрямо стойността на `digit`-тата му цифра и нужния пренос. Така пресмятаме $f(x)$ за всяко $1 \leq x \leq 10^6$ и поддържаме префиксни суми за да отговаряме на заявките.

Постигната сложност: $O(\max r_i \log_2 \max r_i + Q)$

Имплементация: `binary_45p.cpp`

Решение за $r_i \leq 10^{18}$

Логично е да мислим как да пресметнем $g(x) = f(1) + f(2) + \cdots + f(x)$. Ако може да го пресметнем бързо, то тогава отговора на заявка би бил $g(r) - g(l-1)$. Нека разгледаме горното динамично. Ние може леко да го променим, така че да намира $g(x)$ относително лесно. Как? Ами в момента то пресмята $f(x)$ спрямо цифрите на x в побитовото представяне. Ние може вместо да разглеждаме цифрите за някое x, то да разгледаме случая когато текущата цифра е 0 и когато текущата цифра е 1. Тоест, вместо както отгоре проверявахме дали най-горната цифра в побитовото представяне е 0 или 1, то да разгледаме случаите за всички числа, които имат първа цифра 0 и всички числа, които имат първа цифра 1. Кода на горното решение се променя много малко, като единствено към динамичното се добавя един параметър `[equal]`, който ни помага с определянето дали има число, което има текуща цифра 1 в побитовото си представяне, което е $\leq x$ (когато пресмятаме $g(x)$). За да стане максимално ясно, може да разгледате разликите в имплементациите на това и предишното решение.

Постигната сложност: $O(Q \log_2 \max r_i)$

Имплементация: `binary_100p.cpp`

Автор: Борис Михов