CodeZeroOne
(решение)

За решението на задачата състезателите трябваше да проявят известна съобразителност, тъй като за да направят задачата за 100 точки, беше нужно да се направят две важни наблюдения.

Първото от тях беше нужно за да могат бързо да намират дали цифрата на позиция *i* в редицата на Ели е нула или единица.

Един начин беше просто състезателите да генерират редицата.

Wait, what? Нали в най-лошия случай от нея ни трябват първите 1,000,000,007 цифри!? Е, да, ама за модерен компютър това не е ЧАК толкова много, особено ако ползваме побитови операции за да забързаме сметките около 30 ПЪТИ. Реално можем да пазим 1,000,000,007 / 32 = 31,250,001unsigned числа, което е едва около 120 мегабайта. Всеки бит от тези числа ще задава една цифра от редицата.

Самото генериране започва с това да генерираме първите 32 цифри (бита) и да ги сложим в един unsigned int. Нека кръстим масива, в който ще държим редицата unsigned bits(31250001). Така първите 32 ще бъдат в bits(0).

За да намерим следващите 32 бита можем да направим bits(1) = ~bits(0). Реално операцията ~ (побитов not) прави точно това, което иска дефиницията на Ели – заменя всяка нула с единица, и обратно – всяка единица с нула. След това bits(2) = ~bits(0), bits(3) = ~bits(1). Аналогично можем да запълним и останалите около 32 милиона unsigned-а, за едва 32 милиона операции!

За да вземем цифрата на позиция idx (индексирано от нула) ни интересува idx % 32-рия бит на idx / 32-рия елемент на bits. Ползвайки следния код можем да видим дали е нула или едно:

```
!!(bits[idx / 32] & (1u << (idx % 32)))
```

което също можем да запишем и като

```
!!(bits[idx >> 5] & (1u << (idx & 31)))
```

за да избегнем деление и модул (макар че компилаторът би трябвало да е достатъчно умен да се сети да го оптимизира). Яко, нали?

По-наблюдателните (а може би и по-опитните от вас*) биха могли да направят алтернативно, но по-сложно за измисляне наблюдение – цифрата на позиция idx е равна на popcount(idx) % 2 (където функцията popcount(int num) връща броя сетнати битове в числото num). Даже състезателите можеха да ползват доста бързата вградена имплементация __builtin_popcount(). Казах, че по-опитните може да са виждали тази редица и преди, тъй като тя е свързана с генериране на гадни тестове срещу хеширане с овърфлоу. В случай, че нямате идея за какво говоря, можете да погледнете следната статия: http://codeforces.com/blog/entry/4898

Независимо кой от двата начина изберат състезателите, те ще могат да намират *i*-тата цифра на редицата за O(1).

Само с това наблюдение можем да направим решение, което просто обхожда всички позиции във всички интервали и сумира битовете. Това е бавно, но ще хване около 30 точки.

За да хванем повече, трябва да измислим някакъв ефективен начин да намираме сумата на всички битове в интервал. Един вариант е да ползваме префиксни суми (още един масив с около 32 милиона елемента), но няма да го разглеждаме, тъй като другият е значително по-поучителен и елегантен.

Вместо да ползваме префиксен масив, ще ползваме само една идея от него: ако ни питат за сума в интервал (L, R) и можем лесно да намерим сумата в (0, X), то лесно можем да намерим и тази в (L, R): това е просто (0, R) – (0, L – 1).

Остава да решим малко по-лесната задача – как ефективно да намираме сумата в (0, X)?

За това ще ни трябва второто наблюдение, което, за наше щастие, е значително по-лесно: ако разделим редицата на двойки цифри (0-лева и 1-ва, 2-ра и 3-та, 4-та и 5-та…), то във всяка двойка има точно една нула и точно една единица. Това следва директно от начина, по който генерираме редицата.

Как ни помага това? Ами – броят единици в (0, X) е много близо до X / 2. Реално ако X е нечетно (тоест имаме четен брой индекси в (0, X), то бройката е точно X / 2. Ако е четно, обаче, не сме сигурни дали последната позиция може да е както 0, така и 1. За да намерим това ползваме метода, който сме си избрали от първото наблюдение.

И сме готови! Всяко питане е константно, тоест O(1), тъй като:

- Смятането на (0, R) е константно (максимум едно O(1) ползване на първото наблюдение и едно деление).
- Смятането на (0, L – 1) е константно (по същите съображения)
- Смятането на разликата, разбира се, също е константна операция.

Решението на цялата задача е O(N) и ако се имплементира добре става на едва 40-тина кратки реда.

Автор: Александър Георгиев