Състезание Mycamp`4
Задача 2. Грозни числа - Анализ

Красимир Георгиев

Още от пръв поглед за ограниченията на задачата ($N \leq 10^{18}$), става ясно, че прост брутфорс няма да работи. Необходим ни е някакъв алгоритъм с поне логаритмична сложност. Начинът, по който е зададена задачата, не ни дава много информация за точния вид на числата, които търсим. Затова си заслужава да отделим време за по-дълбок анализ на задачата.

Нека имаме едно число $A = \overline{a_n a_{n-1} a_{n-2} \dots a_0}$, записано в двоична бройна система:

$$A = a_n 2^n + a_{n-1} 2^{n-1} + \cdots + a_0 2^0.$$

Тогава сумата на цифрите му е:

$$S(A) = a_n + a_{n-1} + a_{n-2} + \cdots + a_0$$

Така, всяко число, събрано със сумата на цифрите си може да се представи във следния много по-удобен за изследване вид:

$$A + S(A) = a_n(1 + 2^n) + a_{n-1}(1 + 2^{n-1}) + \cdots + a_0(1 + 2^0). \quad (1)$$

Така оригиналната задача придобива следния вид:

Да се намери броят на числата, по-малки или равни на N, непредставими във вида:

$$a_n(1 + 2^n) + a_{n-1}(1 + 2^{n-1}) + \cdots + a_0(1 + 2^0),$$

където $a_0, a_1, \dots, a_n$ са нули или единици.

Как ни помага това?

Първо, като използваме новото представяне, можем бързо да проверим дали произволно число е грозно. Нека $G(n,k)$ е броят на представянията (1) на числото n, в които $a_k = 1, a_i = 0, i > k$, а $S(n,k)$ е броят на представянията (1) на числото, в които $a_i = 0, i > k$. Например, понеже:

$$5 = 1.5 + 0.3 + 0.2 = 0.5 + 1.3 + 1.2,$$

то:

$$S(0,5) = 0, S(1,5) = 1, S(2,5) = 2.$$

Нека $d_n = 2^n + 1, s_n = d_0 + d_1 + \cdots + d_n = 2^{n+1} + n.$

Тогава очевидна е рекурсивната зависимост:

$$G(n,k) = S(n - d_k, k-1)$$

$$G(d_k, k) = 1$$

$$G(n,k) = 0, \text{за } a < d_k \text{ или } a > s_k$$

$$S(n,k) = G(n,k) + G(n,k-1) + \cdots + G(n,0).$$

Така, следният код намира ефективно броят на представянията (1) на произволно число:

```
long long d(int k){
	return 1+(1ll<<k);
}

long long s(int k){
	return k+(1ll<<(k+1));
}

long long S(long long n, int k);
long long G(long long n, int k);

long long S(long long n, int k){
	long long res = 0;
	for(int i = 0; i <= k; i++) res += G(n, i);
	return res;
}

long long G(long long n, int k){
	if(n == d(k)) return 1;
	if(n < d(k) || n > s(k)) return 0;
	return S(n - d(k), k-1);
}

// Finds the number of representations of n
const int max_k = 63;
inline long long Sr(long long n){
	return S(n, max_k);
}
```

Тук максималната стойност на k се определя от ограниченията.

Следващата стъпка е да забележим, че ако едно число A е грозно в неговото представяне (1), $a_n = 1, a_i = 0, i > n$, то и числото $A + d_{n+1}$ почти винаги е грозно. Наистина, ако разгледаме представянията (1) на не-грозните числа, можем да получим всички не-грозни числа, като добавим d_{n+1} към някое вече съществуващо (добавяме единица отпред в представянето (1) на произволно число). Например:

```
1 = 1.d_0 = '1
2 = 1.d_1 + 0.d_0 = '10
3 = 1.d_2 + 0.d_1 + 0.d_0 = '100
5 = '1000 = '110
5 + 1 = 6 = '1000 + '1 = '1001
5 + 2 = 7 = '1000 + '10 = '1010
5 + 3 = 8 = '1000 + '100 = '1100
```

Това ни показва, че сумата на две не-грозни числа е също не-грозно число.

Сега лесно се съобразява, че ако едно число е грозно, то може да се разшири до друго грозно, освен във случая, когато числото лежи в интервала $[d_k; s_{k-1}]$. Какво да правим в този случай? Ключовото наблюдение е, че този интервал е много малък $s_{k-1} - d_k = k = O(\log N)$, което ни позволява в този интервал да търсим грозните числа по наивен начин.

Така за броят на грозните числа, по-малки или равни на N,$\#N$, получаваме следното:

$$\#N = \begin{cases} \#[1, N] \text{ за малко } N \\ \#d_k + \#[d_k + 1, N], N \in [d_k; s_{k-1}] \\ \#s_{k-1} + \#(N - d_k) - \#(s_{k-1} - d_k) \end{cases}$$

Стойностите $\#d_k, \#s_{k-1}, \#(s_{k-1} - d_k)$ могат да бъдат преизчислени. Но за изискванията на задачата дори само преизчислението на $\#d_k$ е достатъчно. Сложността на този подход е близка до логаритмична. Наистина, за да намерим $\#N$, трябва да намерим $\#(N - d_k)$, но $N - d_k \leq \frac{N}{2}$.

```
// This finds the number of ugly numbers
// in the interval [start; end], using
// bruteforce
long long E(long long start, long long end){
	long long i, res = 0;
	for(i = start; i <= end; i++)
		if(Sr(i) == 0) res++;
	return res;
}

// Using map for memoization
#include <map>
map<long long, long long> N_map;

// This finds the number of ugly numbers
// smaller or equal to n
long long N(long long n){
	long long dk, skm1, t;

	// Do bruteforce for little values of n
	if(n < 20) return E(0, n);

	// Find if already computed
	t = N_map[n];
	if(t != 0) return t;

	// Find the largest d_k <= n
	int k = 0;
	while(d(k) <= n) k++;
	k--;
	dk = d(k);
	skm1 = s(k-1);

	// 1. If n is of the type d_k
	if(n == dk){
#ifdef PRECOMPUTE
```

```
	// Precompute code for finding #d_k
		t = N(n-1);
		if(Sr(n) == 0) t++;
		N_map[n] = t;
		return t;
#else
	// Now the precomuted values of #d_k are in the array Ndk
	return Ndk[k];
#endif
	}

	// 2. If n is in the interval [d_k, s_(k-1)]
	if(n < skm1)
		t = N(dk) + E(dk+1, n);
	// 3. Otherwise
	else
		t = N(dk) + E(dk+1, skm1) + N(n - dk) - N(skm1 - dk);

	N_map[n] = t;
	return t;
}
```