АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА ОБЪРНАТИ ЧИСЛА

Задачата има тривиално решение – за всяка заявка (A_m, B_m) и за всяко число k в интервала [A_m, B_m] да се пресметне $\text{rev}_S(k)$ и да се сумират получените числа по модул 1000000001. Пресмятането на $\text{rev}_S(k)$ става със сложност $O(S)$ и сложността на това решение в най-лошия случай е $O(M.S.2^S)$. Всъщност, $M.2^S$ може да се замени във всеки конкретен случай със сумата $W = \Sigma(B_m - A_m)$ на разликите $B_m - A_m$, $m = 1, 2,\ldots, M$. Затова, тривиалното решение може да приключи работа в определеното време само за тестовете, при които тази сума е ограничена (в случая до 20 000 000).

Несложен алгоритъм, който би могъл да реши задачата в някои случаи, при които сумата W надхвърля ограничението от 20 000 000, се получава с техниката динамично програмиране. За целта създаваме едномерна таблица T с $1 + \max B_m$ елемента, където $T_k = 0$, а за всяко $k = 1, 2, ..., \max B_m$, $T_k = \Sigma_{j=1,2,\ldots,k} \text{rev}_S(j) \pmod{1000000001} = T_{k-1} + \text{rev}_S(k) \pmod{1000000001}$. Попълването на таблицата ще стане за време $O(S.\max B_m)$. Отговорът на всяка заявка (A,B) в този случай е $T_B - T_{A-1}$, пресмята се за константно време и затова сложността на алгоритъма в най-лошия случай е $O(M + S.2^S)$. Ясно е, че този алгоритъм няма да може да завърши работа за нормално време при големи стойности на $\max B_m$. При големи стойности на $\max B_m$ ще се прояви и насъщният дефект на схемата Динамично програмиране – няма да има достатъчно място за разполагане на таблицата. Затова, този алгоритъм ще бъде успешен само при ограничени стойности на $\max B_m$.

С предварителен анализ на входните данни и избор на по-подходящия от двата изложени алгоритъма, може да се получи нов алгоритъм, който се справя и с двата вида тестове.

Ефективното решение на задачата, обаче, се състои в намирането на бърз алгоритъм за пресмятане на сумите $\Sigma_{j=1,2,\ldots,k} \text{rev}_S(j) \pmod{1000000001}$, които по-горе означихме с T_k и намирането на резултата за заявката (A, B) като $T_B - T_{A-1}$. За целта да разгледаме двоични представяния на числата от 0 до k при зададеното S (за илюстрация на разсъжденията, вижте в таблицата представянията на числата от 0 до 20 с 5 цифри). Не е трудно да се докаже, че при подреждане на двоичните представяния с S цифри на числата от 0 до k едно над друго, в колоната (0) – тази на най-младшата цифра, се редуват една след друга 0 и 1. В колоната (1) имаме редуване на две нули с две единици, и т.н., в колоната (i) се редуват 2^i нули с 2^i единици. Да означим с $E_i(k)$, $i = 0, 1, \ldots, S - 1$, броят на единиците в i-тата колона на таблицата на двоичните представяния на числата от 0 до k. В тази колона имаме точно $\lfloor (k + 1)/2^{i+1} \rfloor$ цели блока от 2^i нули и 2^i единици (с $\lfloor x \rfloor$ означаваме най-голямото цяло, по-малко или равно на x) и един непълен блок с $R = (k + 1)\%2^{i+1}$ цифри. Ако $R > 2^i$, тогава в непълния блок има $R - 2^i$ единици, а в противен случай няма единици. Значи

$E_i(k) = \lfloor (k + 1)/2^{i+1} \rfloor.2^i$, ако $R \le 2^i$,

а

$E_i(k) = \lfloor (k + 1)/2^{i+1} \rfloor.2^i + (k + 1)\%2^{i+1} - 2^i$, ако $R > 2^i$.

(4)	(3)	(2)	(1)	(0)
0	0	0	0	0
0	0	0	0	1
0	0	0	1	0
0	0	0	1	1
0	0	1	0	0
0	0	1	0	1
0	0	1	1	0
0	0	1	1	1
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	1	1	0	0
0	1	1	0	1
0	1	1	1	0
0	1	1	1	1
1	0	0	0	0
1	0	0	0	1
1	0	0	1	0
1	0	0	1	1
1	0	1	0	0

Остана да съобразим, че единиците от колоната (i), след огледалното обръщане на двоичните представяния, ще отидат в колоната ($S - i - 1$) и затова всяка такава единица ще добавя в търсената сума T_k по 2^{S-i-1}, т.е. $T_k = \sum_{i=0,1,\ldots,S-1} E_i(k).2^{S-i-1} \pmod{1000000001}$. Така за всяка заявка ($A$, B) ще ни трябват пресмятане на T_{A-1} и T_B, всяко от които със сложност $O(S)$. Получаваме алгоритъм със сложност $O(M.S)$. Алгоритъмът се реализира от следната програма (забележете, че цикълът във функцията `partsum` трябва да бъде прекратен, когато броят на единиците пресмятан на поредната стъпка в променливата `E` се окаже 0 – в останалите колони няма единици):

```
#include <stdio.h>
#define MM 1000000001
unsigned S,M;
unsigned partsum(unsigned k)
{   long long E; unsigned Sum=0,i,e,f,g;
    if(k== -1 || k==0) return 0;
    f=2;g=1;
    for(i=1;i<=S;i++)
    {  E=((k+1)>>i)<<(i-1);
       e=(k+1)%f;
       if(e>g) E=E+e-g;
       if(E>0) {E=(E<<(S-i))%MM; Sum=(Sum+E)%MM;}
       else break;
       f=f<<1;g=g<<1;
    }
    return Sum;
}
int main()
{   unsigned k,l,a,b;
    scanf("%d %d",&S,&M);
    for(k=1;k<=M;k++)
    {  scanf("%d %d",&a,&b);
       a=partsum(a-1);b=partsum(b);
       if(b<a) b+=MM;
       printf("%d\n",b-a);
    }
    return 0;
}
```

Забележка. Входните данни и изходните данни в тази задача са толкова обемни, че всеки опит да се чете входа от потока `cin` и да се записва изхода в потока `cout` ще доведе до плачевен резултат.

Автор:Красимир Манев