# GCD - Анализ

**Наивни идеи**

Първо ще разгледаме няколко наивни идеи, които могат да хванат само първия тест. Една такава идея е просто да итерираме през всички възможности от $0$ до $MAX\_X$, което ще означим с $M$. За дадена стойност $Y$ правим заявката $(M-Y, Y)$, тук отговорът ще е $M$, тогава и само тогава когато $X = Y$. Това решение прави средно $(M+1)/2$ заявки и получава под 2 точки. Можем да пробваме да открием делителите на $X$ като питаме за всяко просто число $P$: $\left(0, P^k\right)$, където $k$ е най-голямото цяло число, такова че $P^k < 2M$. След като направим това за всички прости до $M$, знаем че $X$ е равно на произведението на отговорите на заявките. Това решение прави около $M/\ln M$ заявки. Всъщност можем да спираме и по-рано, ако текущото открито $X$ (т.е. произведението на отговорите до сега) умножено по текущото просто число би надвишило $M$. Решението хваща първия тест за 10 точки, но не хваща втория.

**Цифри в двоична**

Първата по-смислена идея е да открием числото в двоична бройна система, цифра по цифра. Първо ще открием най-десния бит със заявката $(0,2)$. Ако отговорът е $2$, то числото е четно, т.е. най-десният му бит е $0$, а иначе той е $1$. Ще следим $A'$ и $B'$, които в началото ще са $0$ и $1$. На дадена стъпка знаем, че $X = A' \pmod{B'}$. Тогава питаме заявка $(2B' - A', 2B')$. Нека дефинираме следното: $R = \gcd(X + 2B' - A', 2B')\ /B'$ (отговорът винаги ще се дели на $B'$). $R$ винаги е или $1$, или $2$. Ако $R = 2$, то $X + 2B' - A' = 0 \pmod{2B'}$; следва, че $X = \mathrm{A}' \pmod{2B'}$; с други думи, текущият бит е $0$. Следващата стъпка е $\mathrm{A}'_{new} = A'$ и $B'_{new} = 2B'$. В другия случай, $R = 1$, тогава остава $X + 2B' - A' = B' \pmod{2B'}$; следва, че $X = B' + A' \pmod{2B'}$; с други думи, текущият бит е $1$. Следващата стъпка е $\mathrm{A}'_{new} = B' + A'$ и $B'_{new} = 2B'$ Спираме, когато $B' \geq M$, тогава $X = A'$. Това решение прави $\log_2 M$ заявки и изкарва около 26 точки.

Забележете, че не се възползваме изцяло от заявките си. Даваме $B = 2B'$, което може да е доста по-малко от $2M$. Вместо това нека $k$ е максималното цяло число, такова че $2^k < 2M$. Винаги ще питаме заявки с $B = 2^k$. Нека отново започнем с $\mathrm{A}' = 0$ и $B' = 1$. Питаме подобни заявки: $(B - A', B)$ и дефинираме $R$ по подобен начин: $R = \gcd(X + B - A', B)\ /B'$. Тук възможните стойности на $R$ са повече, може да е всякаква (под $M/B'$) степен на двойката. Това ни дава някаква бройка нули в двоичната репрезентация на $X$. Получаваме, че $X = \mathrm{A}' \pmod{RB'}$, но това не е всичко. Това са само нулите, но фактът, че сме получили $R$, а не $2R$, значи, че следващата цифра е в двоичния запис на $X$ е $1$. Това всъщност не важи, ако $RB' = B$, но в стандартния случай следва, че $X = RB' + A' \pmod{2RB'}$, т.е. следващата стъпка е: $\mathrm{A}'_{new} = PB' + A'$ и $B'_{new} = 2RB'$. Отново спираме, когато $B' \geq M$, т.е. когато $B' = B$. Забележете, че това решение всъщност прави по една заявка за всяка единица в записа на числото, плюс още една заявка, ако последната цифра е нула. За произволно число, в средния случай половината му цифри са единици и последната цифра е нула в половината случаи, т.е. решението прави $(\log_2 M + 1)/2$ заявки и изкарва около 41 точки.

**Китайска теорема за остатъците**

Преди да разгледаме решения за повече точки, нека видим едно алтернативно за подобни точки, което също ще ни послужи за пълното решение. Ако знаем че $X = A'_i \pmod{B'_i}$ за няколко различни $i$, можем да открием $X = A' \pmod{B'}$, където $B' = \prod_i B'_i$, а $A'$ смятаме по стандартен начин: Нека $H_i = B'/B'_i$ и нека $H_i^{-1} H_i = 1 \pmod{B'_i}$, тогава $A' = \sum_i A'_i H_i H_i^{-1} \pmod{B'}$. Знаейки това, можем да направим решение, което открива $X$ по модул степени на първите няколко прости числа, например: $16, 9, 25, 7, 11, 13, 17, 19$ (защото произведението им е над $10^9$). Сега въпросът е как да открием $A'_i$-тата. Най-простият начин е с наивно итериране за всяко $i$ по отделно, за дадено $i$, това отнема по средно $(B'_i + 1)/2$ заявки. Това решение получава около 18 точки.

Можем да го подобрим като обаче зададем $B = B'$ и итерираме $A$ от 0 до $\max B_i - 1$, т.е. до 24. Сега на всяка от тези стъпки имаме $Q = \gcd(X + B - A, B)$. За всяко $i$, проверяваме дали $B'_i$ дели $Q$. Ако да, то $A'_i = Q \pmod{B'_i}$. В най-лошия случай това решение използва $\max B_i - 1$, но често е по-малко. За $M = 10^9$, средната бройка заявки е $15.8$. Решението получава около 40 точки.

**Цифри в шестична**

Сега ще разгледаме как да разширим решението от двоична в повече или по интересни основи. Една идея е да откриваме числото в шестична. Тъй като $6 = 2 \times 3$, можем с до две заявки да откриваме текущата цифра. Нека $R = \gcd(X + 6B' - A', 6B')\ /B'$. Ако $R = 6$, то цифрата е 0; ако $R = 3$, то цифрата е 3; ако $R$ е четно, то цифрата е четна и с една заявка откриваме дали е 2 или 4; случаят с нечетно $R$ е аналогичен. Забележете, че накрая може да е нужно да направим една или две заявки в двоична, защото $6B'$ може да е повече от $2M$. В този вид това решение прави приблизително $5/3 \times \log_6 M \approx 0.64 \times \log_2 M$ заявки и изкарва около 33 точки, но можем да го разширим със същия вид оптимизация като преди, като сложим $B = 6^k$ с малко следената на какъв остатък остава за следващата стъпка. Тогава правим по една заявка за всяка единица, двойка и тройка и по две заявки за четворки и петици, плюс още една заявка ако последната цифра е нула. Общо решенето прави около $7/6 \times \log_6 M \approx 0.45 \times \log_2 M$ и изкарва към 49 точки.

**Цифри в двоична с екстри**

Друга простичка идея е да запазим същото решение както „оптимизираното“ за цифри в двоична и да видим, че всъщност всички уравнения и заключения работят независимо на какво е равно $B$. Затова ще фиксираме, например, $B = 27 \times 25 \times 7 \times 11 \times 13 \times 2^k$. Самото решение е идентично, с изключението, че когато „изчерпаме“ всички степени на двойката в $B$, т.е. когато $2^k$ дели $B'$, няма как да си гарантираме, че ще правим прогрес, та ще сменим $B = B' \times 2^t \geq M$. Всъщност е възможно да оптимизираме това като динамично сменяме $B$-то, а не само веднъж, но това ще разгледаме в контекста на пълното решение. Трудно е да се сметне броя заявки нужни на това решение, защото понякога има шанс да открие седмична цифра, например, но по-късно не (след като вече знае числото по модул 7), също понякога може да открие над една двоична цифра, но понякога не, ако например текущо $2^{k-1}$ дели $B'$. Такива трудности ще стават все по-чести за по сложните решения, та затова от сега нататък ще казваме средния брой заявки за $M = 10^9$ като конкретно число. Тук те са $10.11$ и решението получава приблизително 66 точки. Подобно решение с динамично избиране на $B$-то изкарва около 74 точки.

**Цифри в двоична и троична едновременно**

Можем да видим, че при нито едно от горните две не максимизираме информацията, която извличаме на заявка. Например, във второто, може вече да знаем, че числото не е сравнимо с $1$ по модул $3$, но да направим заявка с $A'$ сравнимо с $1$ по модул $3$, т.е. да не извличаме никаква информация по модул $3$. Вместо това можем изцяло независимо да правим решението в двоична и в троична, като следим $A'_2$, $B'_2$, $A'_3$ и $B'_3$. Решението в двоична ще процедира както вече описано. В троична, идеята е подобна, но трябва да следим и $C'$, което е какви стойности сме пробвали за текущата цифра. Т.е. първо ще пробваме $C' = 0$, после $C' = 1$, а ако и двете не станат следва, че цифрата е $2$. Вместо да разглеждаме заявки от вида $(A, B)$, за по-удобно от сега нататък, ще използвам $(B - A'', B)$. Заявките в двоична са, както и досега, с $B_2 = B'_2 \times 2^{t_2}$ и $A''_2 = A'_2$, а заявките в троична са с $B_3 = B'_3 \times 3^{t_3}$ и $A''_3 = A'_3 + C_3 B'_3$ (забележете, че можем да въведем и $C_2$, но то винаги е равно на $0$). Сега за да получим цялостна заявка: $B = B_2 B_3$, а $A''$ намираме с Китайска теорема за остатъците, такова че $A'' = A_2 \pmod{B_2}$ и $A'' = A_3 \pmod{B_3}$. След като питаме заявката, дефинираме $R = \gcd(X + B - A'', B)/B'$, където $B' = B'_2 B'_3$. Накрая, лесно можем да намерим степен на двойката $R_2$ и степен на тройката $R_3$, такива че $R = R_2 R_3$ и да направим нужните ъпдейти за двойката и за тройката поотделно. Остава да изберем $t_2$ и $t_3$; това са „колко още степени на $2$ и $3$ да сложим в $B$, над вече откритите“. Когато работим само с $2$ и $3$ като основи, не е много важно как правим това, затова просто ще вземем $k_2 = k_3$ (или ако е нужно $t_2 = t_3 + 1$), така че $B \geq M$. Решението прави средно $9.53$ заявки и изкарва около 70 точки.

**Много прости основи едновременно**

Всъщност нито една част от теорията горе не беше специфична за точно $2$ и $3$ като основи. Лесно можем да разширим всички части за произволна бройка прости числа като основи, например първите осем. Само става по-труден изборът на $t_p$-та. За сега ще зададем $t_p = 1$ или $t_p = 0$ за всяко просто, т.е. всяко просто число или е включено още веднъж, или не, като избираме кои да включим с някаква разумна евристика (ще опишем по-пълна такава след малко). Такова решение използва средно $8.24$ заявки и изкарва около 79 точки. Сега нека да разгледаме по-пълния случай, в който избираме всякакви $t_p$-та. Това ще направим като за всяка степен (до някакъв максимум) на всяко разгледано просто число сметнем приблизително колко „по-ценна“ е спрямо предишната степен. Първо да разгледаме първите степени. За дадено просто число, информацията на една цифра в $p$-ична е $\log_2 p$ бита, а бройка заявки нужни до откриване на следващата цифра в $p$-ична е най-много $p - C_p - 1$, та ще зададем евристика $E_p^1 = \log_2 p \,/\big(p - C_p - 1\big)$. За по-горните степени, печелим информация само, когато познаем текущата цифра (вероятността за което е $1/\big(p - C_p\big)$) и когато междинните цифри са всички нули. Т.е. Вероятността да получим информация от степен $i$ (където $i > 1$) е $1/\left(\big(p - C_p\big) \times \mathrm{p}^{i-2}\right)$. А колко информация ще бъде това? Ами това е еквивалентно да направим нормална заявка за $p$ на първа степен, след като сме открили междинните нули и имаме $C_p = 0$, т.е. следва, че $E_p^i = \log_2 p \,/\left(\big(p - C_p\big)(p-1)\mathrm{p}^{i-2}\right)$. Сега просто ще сметнем всички тези $E_p^i$-та и ще ги сортираме в намаляващ ред. Ще итерираме през тях и всеки път ще включваме текущата степен на просто в общото ни $B$, ако то ще остане под $2M$, а иначе ще я прескачаме. Това решение прави средно $7.01$ заявки и изкарва приблизително 91 точки.

**Изпускане на разни основи**

Последните 10тина точки в задачата изискват няколко трика и са предназначени само за хората, които са намерили за лесни стъпките до сега и са успели бързо да ги имплементират. Можем да видим, че в предното ни решение, понякога имаме $t_p = 0$, т.е. на текущата стъпка никога не откриваме никаква информация в основа $p$. Въпреки това, обаче имаме $B_p = B'_p$, което ненужно хаби част от капацитетът на заявката. Вместо това можем да задаваме $B_p = 1$, ако $t_p = 0$, и отново $B_p = B'_p \times p^{t_p}$, ако $t_p > 0$. Това значително подобрява ефикасността на заявките на решението, макар и по никакъв начин да не го „подтикваме“ към това да се възползва от това правило за капацитет (например може да е добра идея да пробваме да не включим нещо, което евристиката ни би искала да включи). Решението с тази оптимизация средно използва $6.51$ заявки и получава около 97 точки.

**Раница**

Последната стъпка, само за истински отдадените състезатели, е да избираме $t_p$-тата по по-умен начин. Ще решаваме задача подобна на стандартната 0-1 раница. Основните разлики са две. Има „редици“ от предмети, като от всяка „редица“ трябва да вземем префикс от предмети. Това не е трудна промяна, като просто използваме два вложени цикъла и не записваме директно върху ДП масива, а го ъпдейтваме само между редиците. Втората, по-съществена разлика, е това, че става дума за много големи капацитети и теглата/цените на предметите се умножават, а не събират. За щастие няма чак толкова много различни произведения (а и всяко може да се получи по точно един начин). Още по-важно е, да елиминираме опции стриктно по-лоши от съседите си, т.е. ако имаме опции $(B_1, E_1)$ и $(B_2, E_2)$, където $B_2 > B_1$ и $E_2 \leq E_1$, да премахнем втората. Така на всяка стъпка е нужно да държим не повече от стотина елемента в свързан списък, през който итерираме за всеки предмет (просто число на дадена степен) и който допълваме между всеки две „редици“ (т.е. когато сменяме от едно просто към друго). Това е достатъчно бързо за да мине комфортно в time limit-а, но ако по някаква причина не минава, можем да сложим по-ниски максимални степени за по-големите прости, без да смъкваме качеството на решението (доста интуитивно е, че никога няма да включим $19^4$ като допълнителен делител, например). Решението ни с тази оптимизация средно прави $6.30$ заявки и получава пълните 100 точки.

*Автори: Емил Инджев (решение) и Радослав Димитров (идея)*