**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 до , което ще означим с . За дадена стойност правим заявката , тук отговорът ще е , тогава и само тогава когато . Това решение прави средно заявки и получава под 2 точки. Можем да пробваме да открием делителите на като питаме за всяко просто число : , където е най-голямото цяло число, такова че . След като направим това за всички прости до , знаем че е равно на произведението на отговорите на заявките. Това решение прави около заявки. Всъщност можем да спираме и по-рано, ако текущото открито (т.е. произведението на отговорите до сега) умножено по текущото просто число би надвишило . 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 . Ако отговорът е , то числото е четно, т.е. най-десният му бит е , а иначе той е . Ще следим и , които в началото ще са и . На дадена стъпка знаем, че . Тогава питаме заявка . Нека дефинираме следното: (отговорът винаги ще се дели на ). винаги е или , или . Ако , то ; следва, че ; с други думи, текущият бит е . Следващата стъпка е и . В другия случай, , тогава остава ; следва, че ; с други думи, текущият бит е . Следващата стъпка е и Спираме, когато , тогава . Това решение прави заявки и изкарва около 26 точки.

Забележете, че не се възползваме изцяло от заявките си. Даваме , което може да е доста по-малко от . Вместо това нека е максималното цяло число, такова че . Винаги ще питаме заявки с . Нека отново започнем с и . Питаме подобни заявки: и дефинираме по подобен начин: . Тук възможните стойности на са повече, може да е всякаква (под ) степен на двойката. Това ни дава някаква бройка нули в двоичната репрезентация на . Получаваме, че , но това не е всичко. Това са само нулите, но фактът, че сме получили , а не , значи, че следващата цифра е в двоичния запис на е . Това всъщност не важи, ако , но в стандартния случай следва, че , т.е. следващата стъпка е: и . Отново спираме, когато , т.е. когато . Забележете, че това решение всъщност прави по една заявка за всяка единица в записа на числото, плюс още една заяква, 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 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 за няколко различни , можем да открием , където , а смятаме по стандартен начин: Нека и нека , тогава . Знаейки това, можем да направим решение, което открива по модул степени на първите няколко прости числа, например: (защото произведението им е над ). Сега въпросът е как да открием -тата. Най-простият начин е с наивно итериране за всяко по отделно, за дадено , това отнема по средно заявки. Това решение получава около 18 точки.

Можем да го подобрим като обаче зададем и итерираме от до , т.е. до . Сега на всяка от тези стъпки имаме . За всяко , проверяваме дали дели . Ако да, то . В най-лошия случай това решение използва , но често е по-малко. За , средната бройка заявки е . 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 , можем с до две заявки да откриваме текущата цифра. Нека . Ако , то цифрата е ; ако , то цифрата е ; ако е четно, то цифрата е четна и с една заявка откриваме дали е или ; случаят с нечетно е аналогичен. Забележете, че накрая може да е нужно да направим една или две заявки в двоична, защото може да е повече от . В този вид това решение прави приблизително заявки и изкарва около 33 точки, но можем да го разширим със същия вид оптимизация като преди, като сложим 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 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 . Затова ще фиксираме, например, . Самото решение е идентично, с изключението, че когато „изчерпаме“ всички степени на двойката в , т.е. когато дели , няма как да си гарантираме, че ще правим прогрес, та ще сменим . Всъщност е възможно да оптимизираме това като динамично сменяме -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 ), също понякога може да открие над една двоична цифра, но понякога не, ако например текущо дели . Такива трудности ще стават все по-чести за по сложните решения, та затова от сега нататък ще казваме средния брой заявки за като конкретно число. Тук те са и решението получава приблизително 66 точки. Подобно решение с динамично избиране на -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 по модул , но да направим заявка с сравнимо с по модул , т.е. да не извличаме никаква информация по модул . Вместо това можем изцяло независимо да правим решението в двоична и в троична, като следим , , и . Решението в двоична ще процедира както вече описано. В троична, идеята е подобна, но трябва да следим и , което е какви стойности сме пробвали за текущата цифра. Т.е. първо ще пробваме , после , а ако и двете не станат следва, че цифрата е . Вместо да разглеждаме заявки от вида , за по-удобно от сега нататък, ще използвам . Заявките в двоична са, както и досега, с и , а заявките в троична са с и (забележете, че можем да въведем и , но то винаги е равно на ). Сега за да получим цялостна заявка: , а намираме с Китайска теорема за остатъците, такова че и . След като питаме заявката, дефинираме , където . Накрая, лесно можем да намерим степен на двойката и степен на тройката , такива че и да направим нужните ъпдейти за двойката и за тройката поотделно. Остава да изберем и ; това са „колко още степени на и да сложим в , над вече откритите“. Когато работим само с и като основи, не е много важно как правим това, затова просто ще вземем (или ако е нужно ), така че . Решението прави средно заявки и изкарва около 70 точки.

**Много прости основи едновременно**

Всъщност нито една част от теорията горе не беше специфична за точно и като основи. Лесно можем да разширим всички части за произволна бройка прости числа като основи, например първите осем. Само става по-труден изборът на -та. За сега ще зададем или 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 заявки и изкарва около 79 точки. Сега нека да разгледаме по-пълния случай, в който избираме всякакви -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 -ична е бита, а бройка заявки нужни до откриване на следващата цифра в -ична е най-много , та ще зададем евристика . За по-горните степени, печелим информация само, когато познаем текущата цифра (вероятността за което е ) и когато междинните цифри са всички нули. Т.е. Вероятността да получим информация от степен (където ) е . А колко информация ще бъде това? Ами това е еквивалентно да направим нормална заявка за на първа степен, след като сме открили междинните нули и имаме , т.е. следва, че . Сега просто ще сметнем всички тези -та и ще ги сортираме в намаляващ ред. Ще итерираме през тях и всеки път ще включваме текущата степен на просто в общото ни , ако то ще остане под , а иначе ще я прескачаме. Това решение прави средно 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 , т.е. на текущата стъпка никога не откриваме никаква информация в основа . Въпреки това, обаче имаме , което ненужно хаби част от капацитетът на заявката. Вместо това можем да задаваме , ако , и отново , ако . 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 заявки и получава около 97 точки.

**Раница**

Последната стъпка, само за истински отдадените състезатели, е да избираме -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 и , където и , 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 като допълнителен делител, например). Решението ни с тази оптимизация средно прави заявки и получава пълните 100 точки.

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