АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА
LightsOut

Играта LightsOut (http://en.wikipedia.org/wiki/Lights_Out_%28game%29) всъщност наистина съществува от 1995-та година като електронна игра. В тази задача се искаше състезателите да напишат решение (не задължително оптимално като брой натискания) за дъски с определена големина. А решения съществуват много, както ще се уверите след малко!

Начални наблюдения

Първо ще споменем няколко наблюдения, които до една или друга степен могат да ни помогнат да решим задачата. Двете най-лесни са споменати в самото условие:

- Всяка клетка трябва или да бъде натисната веднъж, или нито веднъж.
- Редът на натискане на клетките няма значение.

Допълнително, хубаво беше състезателите да се сетят:

- Ограниченията, зададени като общ брой клетки, се опитват да замаскират факта, че *по-малката* страна на дъската е относително малка. Така състезателите можеше да "завъртят" дъската така, че броят колони да е по-малък или равен на броя редове. Ползвайки това можем да видим, че дори в най-големите тестове (с до 1000 клетки) броят колони би бил най-много едва sqrt(1000) = 31. До края на обяснението ще считаме, че броят колони е по-малък или равен на броя редове (и е максимално sqrt(NUMBER_OF_CELLS)).
- Тъй като редът на натискане няма значение, много би ни помогнало да въведем ред, който ни е удобен. Един такъв ред на натискане е почвайки от най-горния ред и слизайки надолу.

Наивно решение (~15 точки)

Наивното решение беше да се пробват всички възможни натискания (с до 25 клетки имаме 2^25 варианта). Реално решение се намира бързо, тъй като първите три теста са малко по-малки от 25 клетки. Би трябвало състезателите в тази група да нямат никакъв проблем с това решение.

Малко по-добро бавно решение (~35 точки)

За да вдигнем точките си до около 35 (а даже малко повече) трябва да направим още едно наблюдение:

- Кликанията по първия ред еднозначно определят кликанията по всички останали редове.

Това е така, тъй като (след като сме избрали кои клетки да натиснем на първия ред), ако има останали включени клетки, единственият ни шанс да ги изключим е като цъкаме под тях на втория ред. Аналогично, ако дадена клетка е изключена на първия ред, то не можем да цъкнем под нея на втория, тъй като ще я включим и няма да има как да я изключим по-късно. Наблюдението също така важи и за 2-рия и 3-тия ред; 3-тия и 4-тия; и т.н., до края на дъската.

Така при ограничение до 250 клетки, броят колони ще е най-много sqrt(250) = 15. Можем да направим решение със сложност O(T * 2^M * N * M), което прави около 80 милиона операции – съвсем окей за това подмножество от тестове.

Оптимизирано бавно решение (~50 точки)

Възползвайки се от сравнително малкия брой колони (не случайно е най-много 31!) можем да ползваме друга оптимизация: да представяме всеки ред като битова маска (все пак всяка клетка е или включена, или изключена, което значително ни напомня на бит в двоично число). Така с побитови операции можем да имплементираме цъканията върху произволен брой клетки на даден ред за O(1), което забързва решението ни O(M) пъти, правейки го O(T * 2^M * N). Така можем да решим всички тестове до N * M <= 500, а също така и някои от другите.

По-умно решение (~75 точки)
До сега не ползвахме едно важно нещо от входа на задачата – макар и да има много (T на брой) подтестове, те не са *напълно* независими – те са върху дъска с един и същ размер. Защо, аджеба, задачата е зададена така – не можеше ли за всяка дъска да е даден нейния размер? Това нямаше да направи входа особено по-сложен или по-голям, следователно има друга причина.

По-опитните състезатели би трябвало да са стигнали до извода, че може да се направи прекалкулация, която да работи за *всички* подтестове. Как, обаче, би работела тя, при положение, че входните дъски са различни?

Тук състезателите трябваше да се сетят, че тъй като цъканията върху една и съща клетка се неутрализират, ако имаме две различни входни дъски и съответните им решения, ако вземем XOR-а на входните дъски бихме могли да го решим с XOR-а на решенията за всяка от тях.

Така идеята тук е да се направи прекалкулация на празна дъска – ако цъкнем на еди кои си клетки на първия ред и следваме алгоритъма с изключването на останалите редове, какво остава накрая на последния ред. След малко ще покажем и как точно можем да ползваме това.

За всяка от входните конфигурации ще искаме да видим какво става ако не цъкнем нито една клетка на първия ред и после се опитаме да изгасим остатъка от дъската по гореописания алгоритъм (цъкайки само под включени лампи от горния ред). Така накрая цялата дъска ще е изключена, потенциално с изключение на някои клетки на последния ред.

Тук идва на помощ резултата от прекалкулацията върху празна дъска с всички възможни цъкания на първия ред. Ако на празна дъска сме цъкнали някакво множество от клетки F на първия ред, и в резултат на това накрая на последния ред са останали включени множество от клетки L, и в същото време на дадена входна дъска без да натискаме нито една клетка на първия ред, на последния остават включени същото множеството L, то, цъкайки F на първия ред на входната конфигурация накрая бихме получили празна дъска! Найс, а?

Правейки тази прекалкулация вече можем да решим всеки подтест за O(N*M). Самата прекалкулация обаче е по-бавна - тя е O(2^M * N) (ползвайки всички други досегашни оптимизации). В крайна сметка сложността на това решение е O(2^M * N + N * M) – което е точно T пъти по-бързо от предходното.

Решение, базирано на Meet-in-the-Middle (100 точки)
В предишното решение казахме, че прекалкулацията е бавното нещо – само ако можеше да я оптимизираме! Оказва се, че можем – отново ползвайки "наслагването" на решения, можем да ползваме подхода "meet-in-the-middle" за да изчислим частично кои клетки на последния

ред биха се образували ако цъкнем определено множество от клетки на първия върху празна дъска.

Вместо да пробваме всички 2^M подмножества, ще разделим колоните на две (що-годе) равни части, които ще наричаме "лява" и "дясна" и ще включват, съответно, клетки само от първите M/2 и само от последните M/2 клетки на първия ред. Така за всяка от тях ще можем за O(2^(M/2) * N) да намерим кои клетки на последния ред включва. Интересно е, че можем да комбинираме две такива цъкания (едно от "лявата" и едно от "дясната" част) за да получим някакво множество на последния ред. Нещо повече, по дадени клетки на последния ред и фиксирано множество от, примерно, лявата част, можем с двоично търсене да намерим в дясната дали съществува подмножество, което да изключва тези клетки от последния ред.

Така прекалкулацията става със сложност O(2^(M/2) * N + 2^(M/2) * log(2^(M/2)) = O(2^(M/2) * N), а всеки подтест - с O(log(2^(M/2) + N * M) = O(N * M). Реално това решение е адски бързо за дадените ограничения – то прави едва около милион операции. (То би работило дори ако броят клетки беше до 1600.) Все пак реших да дам задачата с ограничения до 1000, тъй като от една страна така може да се ползва оптимизацията с битовите маски с променливи от тип int, а от друга – за да заблудя състезатели, които по ограниченията стигат до извода какво е решението (да, Енчо, теб).

Частни случаи
Състезателите трябваше да внимават за тестове, в които има един единствен ред (тоест "първият" и "последният" съвпадат. Частен случай на частния случай е когато цялата дъска се състои от една единствена клетка.

Други решения
Оказва се, че съществува и друго решение на задачата, базирано на Гаусова елиминация в полето Z_2. С него могат да се решат дъски много по-големи дъски (с N, M <= 500 - Гаус е O(N^3)). Но тези знания изобщо не влизат в материала на учениците и затова реших, че това не е подходящо за тях.

Подобни задачи
Докато разглеждах за решения, подобни на Гаусовата елиминация, намерих доста подобна задачка в TopCoder: http://community.topcoder.com/tc?module=ProblemDetail&rd=14423&pm=8744. Тя, обаче, може да бъде решена единствено с Гаус.

Оптимално решение
Интересна е и задачката за "оптимално" решение в смисъл на решение, изискващо минимален брой цъкания по дъската. За домашно помислете кои от гореописаните решения могат да бъдат модифицирани за да намират оптимално решение? Как?

Автор: Александър Георгиев