В първите две подзадачи големия брой заявки не е проблем. Можем да минем през всички полета участващи в тях и да отбележим че са оцветени.

Подзадача 1. Решение O(N^2.M^2) би взело десет точки. Най-тривиалното може би е да разгледаме всяко поле (i, j) за горен-ляв ъгъл на търсен квадрат. За всеки горен-ляв ъгъл започваме да увеличаваме страната на квадрата на 1, 2, 3, ... При всяко увеличаване минаваме през всички новодобавени полета и проверяваме дали са свободни. В момента в който срещнем заето поле спираме и преминаваме към следващия горен-ляв ъгъл.

Подзадача 2. Тази задача позволява различни подходи. Едно стандартно решение би било с динамично програмиране. В dp[i][j] се пази страната на най-големия квадрат който има долен десен ъгъл в полето (i, j). Ако полето (i, j) е оцветено в черно, то очевидно dp[i][j]=0. В противен случай е в сила следната зависимост, която лесно може да проверите dp[i][j]=1+min(dp[i][j-1], dp[i-1][j], dp[i-1][j-1]).

Друго лесно решение би било ако се използват двумерни префиксни суми. Така за константно време може да проверим дали даден квадрат е изцяло бял(има сума нула). Сега може за всяка клетка (i, j) да намираме най-големия квадрат за който тя е горен-ляв ъгъл. Един вариант е с двоично по страната на квадрата за тази клетка. Всъщност има още по-добра идея без двоично като използваме максимално намерената страна до момента и за всяка нова клетка започваме да промеряваме от тукщия отговор нагоре.

Подзадача 3. Големия брой заявки става проблемен заради големия брой колони. Няма как да обходим полетата от всички заявки. За да може да разберем дали едно поле е свободно или не трябва да използваме нещо различно. Може да си запазим заявките за отделните редове и колони. Да разгледаме всички заявки за ред i. Имаме няколко интервала и трябва да разберем кои полета участват в някои интервал и кои не. Един лесен начин е на началото на всеки интервал да запишем +1, а на първото поле след края -1. Сега с едно обхождане отляво надясно пазим текущата сума и като стъпим на някое поле добавяме числото от това поле. Ако текущата сума е положителна има поне един интервал, който включва това поле. Ако е нула значи полето е бяло. Можем по този начин да пресметнем за всички полета дали са оцветени или не, след което отново използваме решението с динамично.

Подзадача 4. Тук за да проверим една клетка дали е оцветена няма да стане лесно като в предния случай. Няма да може да ползваме големи масиви и ще трябва в map да пазим +1/-1. Относно динамичното основната разлика е, че не ни трябва да пазим цялата таблица. Достатъчно е да имаме последните две колони.

Подзадача 5. Това е първата стъпка към сериозните ограничения. Първото нещо е да свържем квадрата който търсим с дадените точки. За целта ще търсим най-големия квадрат, който се допира в някоя от дадените точки, като квадата е над точката. Ако най-големия квардат не се допира в някоя точка, то може да го мести надолу докато не допре някоя точка или не покрие последния ред. Следващия проблем е, че не знаем колко голям квадрат да търсим. Тук вече е необходими предварително да знаем каква страна на квадрата търсим. За целта използваме двоично търсене по страната на квадрата. Така имаме фиксирана страната на квадрата и знаем, че той ще се намира над някоя от дадените точки или ще е най-долу в таблицата. Да видим как ще намерим дали над някоя точка (r, c), може да намерим квадрат с фиксирана страна S. От това, че страната е фиксирана означава, че нас ни интересува какви точки имаме само S реда над разглежданата. Т.е. точките с ред в интервала [r-S, r-1]. Сега ще ни интересуват само оцветените в черно точки в този интервал от редове. Ако една точка (x, y) е в него то няма как квадрата със страна S да съдържа колоната y, понеже точката (x, y) ще бъде в него, а тя е оцветен. За тази точка реда x всъщност няма значение, важно е само че е в интервала, който разглеждаме - [r-S, r-1]. Така за всички точки в този интервал ни интересуват само колоните. Ако в една колона има поне една точка, то тази колона не може да е част от квадрата. Сега ще обхождаме точките по редове и във всеки момент ще пазим в диапазона [r-S, r-1] кои колони са оцветени в черно. За всяка точка трябва да видим от запазените колони най-близката колона преди и след дадената точка. За да го правим бързо може да запазваме точките в set/multiset/map.

Подзадача 6. Тази подзадача е продължение на предходната. Сега обаче проблеми идва, че не може да използваме set понеже имаме отсечки и не може да дабяваме всяка точка поотделно. За целта сменяме структурата със сегментно дърво, в което ще търсим най-голяма “дупка”(последователни неоцветени квадратчета) в даден интервал. Така за да сметнем отговора за родителя на две деца се налага да пазим допълнителна информация за най-голямата “дупка” в левия и десния край на интервала. Заради хоризонталните отсечки ще трябва да използваме lazy propagation. За вертикалните отсечки е достатъчно да ги запошим в първия момент в който ги видим и да ги махаме като стигнем последния им ред.

Подзадача 7. Тук е момента да забележим, че двоичното търсене не е необходио. Може да приложим 2-pointers подход по редовете които разглеждаме, като сравняваме ограничения квадрат спряма броя редове и най-голямата “дупка”. Достатъчно е да пазим първия firstRow и последния ред lastRow, които разглеждаме. Ако най-голямата “дупка” в този интервал е по-малка от lastRow-firstRow, то квадрата е ограничен от дупката и реално трябва да намалим редовете за да се увеличи дупката. За това увеличаваме firstRow. В противен случай вдигаме lastRow.

*Автор: Петър Петров*