Анализ на задача `saddle`

Тагове: интерактивна задача, ориентиран ацикличен граф, двоично търсене

Понятието в условието *локална седлова точка* всъщност не е коректно, защото по принцип *седлова точка* в таблица е клетка, която е например максималната в реда и минималната в колоната. Правилното понятие е *локална минимална точка*, но беше малко късно, за да се преименува задачата и всички файлове от `saddle` на `locmin`, а и се получи уникално име на задача 😄. За по-кратко от сега нататък ще казваме само *седлова точка*.

Начални разсъждения - 10 точки

За 100 точки, трябва с много малко въпроси (под 2024 спрямо общо 250 000 клетки) да открием *седлова точка*, което означава, че трябва да има хубав начин да разбираме, че някъде трябва да има клетка с това свойство. Освен това, всъщност, ограничението, че е гарантирано да има поне една *седлова точка* е ненужно и може да се докаже, че то следва от определението и това, че няма съседни клетки в таблицата с равни стойности. Интуитивно това означава, че ако разгледаме дадена клетка и тя не е *седлова точка*, то трябва да има съседна клетка с по-малка стойност и можем да направим същото разсъждение за тази съседна клетка и така малко, по-малко да стигнем наистина до *седлова точка*. Формално, ако представим клетките на таблицата, като върхове в граф и имаме ребра между съседни клетки от клетката с по-голяма стойност към клетката с по-малка стойност, то този граф ще е *DAG* (ориентиран ацикличен граф) и като такъв със сигурност ще има върхове, от които не излизат ребра и те именно са *седловите точки*. Това ни дава и някакво нетривиално решение на задачата - избираме произволна клетка и следваме съседните клетки с по-малка стойност, докато намерим *седлова точка*. За съжаление, такова решение в най-лошия случай не е по-добро от тривиалното намиране на всички клетки на таблицата и затова получава само гарантирания минимум от точки за вярно решение - 10 точки. По-нататък ще покажем една по-добра версия на това решение.

Един имплементационен детайл, който се използва във всички решения, е свързан с това да не повтаряме въпроси. За тази цел най-удобно е да направим наша функция, която се обръща към `value`, но преди това проверява дали не знаем стойността на съответната клетка, като си пазим една таблица с вече познатите стойности.

Постигнат брой заявки (в най-лошия случай): NM

Решение на втора подзадача - 21 точки

Използваме горните разсъждания, както и това, че реално таблицата е просто масив. Стандартно в интерактивни задачи, често се прилага двоично търсене и това не е изключение. Особено тази подзадача направо "крещи" да се използва такъв подход - имаме непознат масив и можем да питаме въпрос каква е стойността на някоя клетка. Нека първо питаме въпроси за клетката в средата и двата ѝ съседа, за да разберем дали е *седлова точка* или не. Ако е *седлова точка* сме готови. Ако не е *седлова точка*, то значи имаме съсед с по-малка стойност. От разсъжденията преди малко, това означава, че в тази половинка на масива трябва да имаме *седлова точка*. Така ще можем да разглеждаме само лявата, или само дясната част на масива и да приложим същото разсъждение. Понеже всеки път намаляваме големината на масива два пъти, то след грубо $\log_2 M$ въпроса ще стигнем до 1 клетка, която трябва да е *седлова точка*.

Постигнат брой заявки (в най-лошия случай): $3.\lceil \log_2 M \rceil$

Рандомизирано решение на трета подзадача - около 20 точки

Можем да подобрим началния подход, при който почваме от една клетка на произволен принцип, като се пробваме да започнем от няколко клетки с цел някоя от тях да е доста близо до *седлова точка*. Това е добре като идея, но как да знаем коя клетка е наистина близо до *седлова точка*? Можем да използваме стойността като ориентир, т.е. в началото гледаме много произволни клетки на таблицата и след това тръгваме от тази с минимална стойност да търсим *седлова точка*. Очевидно такава трябва да има, защото все ще стигнем до връх без изходящи ребра (в графа на таблицата). Тази идея беше дадена и реализирана от *Борис Михов*.

Оказва се, че е доста трудно да се направят тестове срещу такова решение, защото то е гарантирано, че ще уцели някоя клетка, която е близо до истинската *седлова точка* (в повечето тестове има само една *седлова точка*, за да са по-силни). Може математически да се сметне, че например ако в началото питаме за произволни 1000 клетки, то шансът да не сме в радиус 15 от единствената *седлова точка* е по-малък от 3%, т.е. силно вероятно е, ако пробваме всички клетки в радиус 15 от тази с минимална стойност, да намерим *седлова точка*. Затова грейдърът на задачата (програмата на журито) е адаптивен и се стреми накрая да направи *седлова точка* максимално далече от всички въпроси. Този грейдър стана много добър и дори спомогна за откриване на дребна грешка в авторовото решение.

Адаптивният грейдър направи, така че това рандомизирано решение дори с произволен сийд и с допълнителни трикове (например при няколко по-малки съседа, на произволен принцип да се отиде в някой от тях) да не изкарва над 20 точки.

Частично решение на трета подзадача - около 56 точки

Можем сравнително лесно да адаптираме идеята на втора подзадача от едномерния случай към двумерния случай в общата задача. Нека се опитаме да разберем дали в горната част на таблицата има *седлова точка*, или има *седлова точка* в долната част. Фиксираме средния ред и намираме всички клетки в него. Да допуснем, че никоя от тях не е *седлова точка*. Искаме да сме сигурни по някакъв начин, че този ред ще служи като граница и гарантирано ще трябва да имаме *седлова точка* отгоре или отдолу. Но кога може да се наложи да прекосим този ред при търсене на *седлова точка*? Единствено ако в него има клетка с по-малка стойност от някоя друга. Така ако разгледаме клетка с минимална стойност в реда (която знаем, че не е *седлова точка*), то отгоре или отдолу трябва да има клетка с по-малка стойност и имаме важното свойство, че ако се спуснем към нея да търсим *седловата точка*, то няма как да прекосим отново този фиксиран ред по-късно, защото всички клетки в него ще имат по-големи стойности. Така в зависимост от това къде имаме клетка с по-малка стойност от минимума на реда, ще ни остане само горната или само долната половина на таблицата, в която гарантирано ще имаме *седлова точка*. Рекурсивно правим същото и така отново получаваме решение, което е подобно на двоично търсене само че по интервала от редове. Важно е също като стигнем един ред да не се пробваме да прилагаме подхода от втора подзадача, защото при него имахме само съседство отляво и отдясно, а тук може някоя клетка да няма по-малка вляво или вдясно, но да не е *седлова точка*, защото има по-малка клетка отгоре или отдолу.

Ако направим директно тази идея броят на заявките ще е $3N.\lceil \log_2 M \rceil$, което са твърде много заявки. Можем да забележим, че не е нужно да проверяваме за всички клетки от средния ред дали са *седлова точка* - единственото важно е да направим тази проверка за клетката с минимална стойност в реда. Дори и някоя друга от клетките да се окаже, че е била *седлова точка*, то ние със сигурност ще намерим *седлова точка*. Иначе бихме изхабили твърде много заявки да гледаме какви са стойностите на горния и долния ред спрямо средния, за да разберем дали нямаме случайно *седлова точка* в него (а в тестовете с една *седлова точка* точка това очевидно ще даде само допълнителни заявки).

Постигнат брой заявки (в най-лошия случай): $N.\lceil \log_2 M \rceil$

Решение на трета подзадача - 100 точки

Остана само да подобрим още търсенето - малко по-добре да адаптираме едномерната идея към двумерния случай. Нещо важно, което видяхме в предното решение е, че ако разделим таблицата на някакви райони, след което намерим минималната клетка по границата на районите, то тя ще ни укаже точно в кой район трябва да има *седлова точка*. Преди правихме разделението просто по средния ред. Нека сега направим разделение на четири района - правим кръст, който се състои от средния ред и средната колона (за да може района, в който имаме *седлова точка* да намалява освен като брой редове, и като брой колони). Оказва се, че това е най-оптималното разделение (подобно на обосновката защо е най-оптимално да правим двоично търсене). Всеки път намираме минималната клетка в кръста и, ако тя не е *седлова точка*, то ще знаем в кой от четирите квадранта да ходим да търсим *седлова точка* (няма как минималната клетка да е в центъра на кръста, защото е с минимална стойност и не е *седлова точка*). Така в началото ще питаме стойността в $N+M-1$ (броят клетки в кръста)$+2$ (останалите два съседа на минималната клетка) или общо в $N+M+1$ клетки. На следващата стъпка ще питаме стойността в $N/2+M/2+1$ клетки и т.н. общо въпросите в най-лошия случай (или по-скоро ако не сме големи късметлии) ще са грубо: $(N+N/2+N/4+...+1)+(M+M/2+M/4+...+1)+min(\log_2 N, \log_2 M) \approx 2N+2M+min(\log_2 N, \log_2 M)$ (добре известно е, че $N+N/2+N/4+...+1 = N.(1+1/2+1/4+...) \approx N.2$).

Постигнат брой заявки (в най-лошия случай): грубо $2N+2M+min(\log_2 N, \log_2 M)$ или на практика на тестовете 1990.

Идея: Румен Михов
Реализация: Илиян Йорданов