**АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА**

**ИЗКАЧВАНЕ**

Тази задача е относително сложна, но в същото време донякъде стандартна, съответно очаквам около 2-4 човека да я решат за пълен брой точки. Допълнително, има и различни стратегии за "чийтване" на задачата, което ще увеличи броя хора с резултат 40-50, че и нагоре точки, дори с относително прости решения. Някои от чийтовете, за които се сетих, се опитах да убия с тестовете, но все пак е сложно да се направят тестове срещу всички… И в момента тестгенераторът е почти 600 реда, не ми се мисли ако се опитвам а се справя с всичко ☺

Задачата се свежда до намиране на най-дълъг път в DAG (Directed Acyclic Graph). Тази задача стандартно се решава с динамично оптимиране, което трябваше да направим и тук.

Графът има до 250,000 върха и потенциално огромен брой ребра (най-големият брой ребра е от порядъка на O(N2 \* M2)). Специфичното задаване на графа, обаче, позволява да оптимизираме намирането на най-доброто продължение от дадена клетка за O(log(N) \* log(M)), вместо наивното O(N \* M). Това става с техника "оптимизиране на вътрешния цикъл", която обикновено изисква някакво умно наблюдение, структура данни, или намиране на изпъкнала обвивка. В случая, тук ни трябваше структура данни.

Първото нещо, което трябва да направим, е да сортираме клетките по тяхната височина и да ги обработваме в обратен ред (попълвайки динамичната таблица). Наистина, от клетката (клетките) с най-голямото число няма да можем да отидем никъде, съответно отговорът за тях е 1. От тези с малко по-малки числа, от които можем да стигнем единствено до тези с най-голямото отговорът е 2. Продължавайки така, можем да обходим всички клетки в намаляващ ред и да намерим дължината на най-дългия път, започвайки от всяка от тях. Ще запазим тези числа в таблица с размер [N][M] (което ще е таблицата на динамичното ни, което този път правим итеративно).

**Наивно динамично**

Ако просто имплементираме попълването на таблицата, като за всяка клетка разглеждаме всички съседи на разстояние до D ще получим сложност от рода на O(N \* M \* D2), тъй като имаме O(N \* M) клетки, и за всяка от тях имаме О(D \* D) възможни продължения. Тази сложност можем да считаме и като O(N2 \* M2), тъй като реално D няма смисъл да е повече от N + M. Макар и твърде бавно за дадените ограничения, с такова решение бихме хванали около 20 от 50 теста. Тъй като (за да убия някои от чийтовете) искам тестовете да са групирани по петици, реално такова решение би хванало около 20 точки.

**Чийтове**

Повечето "чийтърски" решения, които ми хрумват, се базират на това да разглеждаме само част от съседните O(D \* D) клетки. Това можем да правим по няколко начина. Един от тях е да разглеждаме до 300-500 от предходно-разгледаните клетки. Има логика това да са клетките с възможно най-ниски стойности (тоест тъкмо разгледаните), тъй като от тях се очаква отговорът да е възможно най-голям. За тези от тях, които са на разстояние D или по-малко, ъпдейтваме отговора за текущата клетка като отговора за тях + 1. Избираме броя разгледани продължения да е около 300-500, тъй като имаме да разгледаме O(N \* M) клетки, за всяка от които трябва да разгледаме толкова продължения. При по-голям брой разгледани продължения рискуваме да имаме Time Limit. Такова решение хваща около 37 от 50 теста и би хванало (след групирането по петици) 40 точки.

Една идея по-умно е да разглеждаме най-добрите 300-500 от предходно-разгледаните клетки, вместо най-близките по височина. Тях можем да пазим в сортиран масив (ръчно намирайки позицията на всяка нова клетка), като така постигаме сложност отново O(N \* M \* 500) операции. С това решение бихме хванали 45 от 50 теста, което, след групирането, би донесло 50 точки.

**Реална идея**

Ако вместо Манхатъново разстояние беше казано, че момичетата могат да се движат до D наляво, надясно, нагоре или надолу, то около текущата клетка щеше да се образува квадрат (или правоъгълник, ако клетката е близо до някой от ръбовете), за който можеше да ползваме двумерно [RMQ](http://www.informatika.bg/lectures/range-minimum-query) (потенциално имплементирано чрез сегментни дървета, тъй като стойностите не са статични). Така щяхме да можем да намерим най-доброто продължение за O(log(N) \* log(M)) за всяка клетка, постигайки сложност за цялото решение O(N \* log(N) \* M \* log(M)).

Манхатъновото разстояние, обаче, ни дава ромб, вместо квадрат. Ромбът реално е квадрат, ротиран на 90 градуса. Защо, тогава, да не ротираме цялата дъска на 90 градуса, като така ромбът би станал квадрат?

За целта ще ни трябват двойно повече клетки както по хоризонтала, така и по вертикала. Клетка с координати (row, col) получава нови координати (М - 1 - col + row, row + col). Отново трябва да сортираме клетките по височина в намаляващ ред и да ги обходим така. Вече можем да ползваме сегментното дърво за да забързаме намирането на най-добрия съсед.

Трябва да имплементираме update() на единична клетка в двумерно сегментно дърво и query() за интервал (тоест, правоъгълник, тъй като работим в 2D). Това не е много трудно и можете да видите как да направите в тази тема: <http://www.informatika.bg/lectures/segment-trees>

Ако в началото инициализираме дървото с нули, и вкарваме клетките в намаляващ ред то по всяко време всички вече вкарани клетки ще са по-високи, тоест ще можем да ходим до тях (стига, разбира е, да са достатъчно близо). Преди да сложим всяка от клетките в сегментното дърво, взимаме максимум от досега сложените в интервала [(Ri - D, Ci - D), (Ri + D, Ci + D)] плюс 1 (за текущата клетка). Ъпдейтваме отговора с полученото число и го добавяме в сегментното дърво на позицията на клетката (Ri, Ci).

*Автор: Александър Георгиев*