**Khans**

(Анализ)

Задачата Khans беше предвидена като една от сложните в това състезание. Тя приличаше на модификация на задачата за търговския пътник, с няколко улеснения:

Върховете може да бъдат повтаряни (след определено време)

Графът е много рядък (с най-много четири ребра от всеки връх)

Много малко възможни стойности на "печалба" от всеки връх

Все пак, решението изглежда експоненциално на повечето състезатели, и всъщност е такова. Въпросът е как да го направим по-малко експоненциално ☺

Сещам се за три основни посоки, в които състезателите могат да тръгнат (подредени по бързодействие):

* Пълно изчерпване
* Динамично оптимиране
* Търсене на цикли

Нека разгледаме всяко едно от тях.

**Пълно Изчерпване**

Най-очевидният подход беше просто да се пробват всички възможни пътища и да се избере оптималният от тях. Този подход беше обречен на неуспех, тъй като е твърде бавен. Все пак, с няколко оптимизации и подобрения можеше да се хванат поне някакви точки.

Чистото изчерпване (с обновяване на цялата дъска на всяка стъпа) беше най-простият за написване, но за съжаление и най-бавният. Той би донесъл на състезателите около 15 точки.

Ако се добави бързо обновяване на дъската (тъй като реално има нужда да обновим не повече от 8 клетки, вместо 100 при N = 10, M = 10, както ще видим след малко), то става значително по-бързо и носи около 25 точки.

Ако се добави и умно рязане на търсенето (ползвайки някакви евристики), можеше да бъдат постигнати дори още повече точки. Не съм експериментирал достатъчно да видя колко точно точки могат да се хванат така, тъй като има \*много\* възможни евристики, които могат да бъдат приложени.

**Динамично Оптимиране**

Очакваното решение на задачата беше базирано на динамично оптимиране, като и тук има няколко възможни подхода. Преди да мислят за който и да е от тях, състезателите трябваше да забележат следните неща:

* Размерите на матрицата N и M бяха много малки
* Стойностите на матрицата Aij бяха много малки

Всъщност, фактът, че стойностите на Aij бяха толкова малки, както и това, че храната се удвоява всяка година, можеше да доведе до следното наблюдение:

*Всеки регион се възстановява до максимума си след не повече от 8 години (тоест, със сигурност ще можем да го посетим в началото на 9-тата година, след като сме излезли от него).*

Примерът в условието подсказваше това – дори ако максималната храна в региона беше в горния край 100, то десетте години след като хановете са напуснали региона биха имали следните стойности: 0, 1, 2, 4, 8, 16, 32, 64, 100, 100.

Как можем да използваме това? Ако има начин, по който да пазим кои са последните 8 посетени региона, бихме имали всичко, което ни трябва да зачетем правилото за повторно посещение на регион. И тъй като движението се случва само в съседни по страна региони, можем да пазим:

* Текущия регион
* Последните седем посоки, които хановете са ползвали

Тъй като всяка посока е една от четири, бихме се нуждаели от [10][10] за текущия регион и още [47] = [16384] за посоките.

Това, само по себе си, не е твърде много, но комбинирано с текущата година (която също трябва да включим в стейта), таблицата [K][N][M][47] става твърде голяма: нямаме достатъчно памет да пазим толкова много клетки.

Фактът, че годините текат последователно, може да ни помогне: можем да направим динамичното итеративно, като компресираме размерността [K] в [2] (всеки път алтернирайки между 0 и 1 в зависимост от четността на годината). Така таблицата [2][10][10][16384] се събира лесно в дадената ни памет. За съжаление, това решение е твърде бавно.

Интересно наблюдение, което можем да направим, е че някои (много) от клетките на динамичната таблица ще останат неизползвани. Това обикновено означава, че стейтът ни не е оптимален, и като страничен ефект води до това рекурсивната реализация на динамичното да е забележимо по-бърза от итеративната (тъй като посещава единствено стейтовете, които се ползват). Така имаме следните два избора:

1. Да ползваме рекурсивно динамично, като ползваме вектори за да направим динамичната таблица \*точно\* толкова голяма, колкото ни трябва, както и да ползваме short-ове вместо int-ове (тъй като отговорът е най-много K \* max(Aij) = 100 \* 100 = 10000), като така ползваме точно толкова памет, колкото ни трябва. Това решение е доста бързо, но за сметка на това при големи стойности на N, M, и K заема твърде много памет. Така хващаме около 45 точки (и получаваме ML на останалите тестове).
2. Да ползваме итеративно динамично, като компресираме едната от размерностите на таблицата. Така нямаме проблеми с паметта, но за сметка на това не знаем кои клетки от таблицата ще са ни нужни, затова се налага да изчислим всички. Така също хващаме около 45-50 точи (и получаваме TL на останалите тестове

Аааа! Единственият начин да продължим напред е да видим \*защо\* стейтът ни не е оптимален и да се опитаме да го подобрим.

Какво все още не сме използвали? Може би най-забележимото нещо от ограниченията са странните граници на Aij (10 ≤ Aij ≤ 100). Защо 10, а не 1? Ами, едно от нещата, до които това води, е че никога няма хановете да се върнат в клетката, от която са дошли (тя ще има храна 1, което със сигурност е по-малко от максимума (10 или повече)). Така, същинският брой посоки, в които хановете могат да се придвижат, е 3, а не 4 както твърдяхме по-горе. Това намаля последната размерност на таблицата от 47 = 16384 до 37 = 2187. Всъщност, все пак ни трябва една пълна посока, а всички останали да са определени спрямо предходните, тоест ни трябват 4 \* 36 = 2916 възможности, но това е достатъчно близо.

Така, крайният стейт е следния:

* [K] = [100] или [2] (за рекурсивно / итеративно динамично): текущата година
* [N][M] = [10][10]: Текущият регион
* [4]: Посоката, с която сме дошли в текущия регион
* [3][3][3][3][3][3]: Посоката (изключвайки предходната посока) за достигане до 6-те региона преди това

Този стейт вече е достатъчно малък да държим цялата таблица (включително пълната година): [100][10][10][4][36] = 29,160,000 клетки. Тъй като, както отбелязахме по-горе, можем да ползваме short вместо int за стойностите, тази таблица изисква малко под 60 мегабайта памет.

Понеже стейтът единствено гарантира, че не сме се върнали в клетката, от която сме дошли, но изобщо не гарантира, че не напускаме дъската или че не сме се върнали в не напълно възстановен регион по-рано, бихме очаквали, че същинският брой посетени клетки все пак ще е по-малък от 29-те милиона, които пазим. Ще се възползваме от бързината на рекурсивното динамично (базирано на мемоизация) в тези случаи за да обходим само тези, които наистина ни трябват. (Най-добрия тест, който успях да направя, посещава малко по-малко от 5 милиона уникални стейта, тоест е около 6 пъти по-бърз от итеративното динамично.)

Ако се имплементира добре, този подход ще изисква константен брой операции вътре в рекурсията (просто да провери 4-те възможни посоки, продължавайки рекурсивно в най-много 3 от тях). Откъм сложност, това е O(N \* M \* K \* 4 \* 36) – просто броя клетки на динамичната таблица. На теория трябва да игнорираме 4 \* 36, но нека не задълбаваме там… Това беше и решението, предвидено за 100 точки.

**Търсене на Цикли**

Имаше и тотално различна насока, в която състезателите можеха да тръгнат: да предположат, че голяма част от пътя е "цикъл" – последователност от региони, която се повтаря отново и отново. Така един оптимален път би изглеждал по следния начин:

1. Започвайки от началната клетка, да стигнем до цикъла
2. Да се въртим там известен брой пъти
3. Потенциално накрая да посетим още няколко клетки извън цикъла

Макар и най-вероятно възможно, това решение има \*много\* частни случаи, с които състезателите трябва да се справят, допълнително към това да намерят хубав цикъл, както и да имплементират стигането до него и обхождането на финалните клетки. Предимство тук е, че решението би било полиномиално, тоест много по-бързо от авторовото (като няма да има проблеми нито с TL нито с ML).

Доста вероятно е, обаче, да получава WA. При положение, че задачата е дадена с пълен фийдбек може би би могло да се "чийтне" по този начин, но дори тогава най-вероятно решението няма да е напълно вярно – има голям шанс да съществуват случаи, за които авторът не се е сетил и решението му не решава правилно. Има безброй начина, по които може да се направи такова решение, затова не си играх да пиша решения от този тип – само добавих няколко ръчно-направени теста срещу няколко от идеите, които имах.

*Автор: Александър Георгиев*