HiddenWords
(решение)

Задачата HiddenWords предостави относително лесна възможност състезателите да имат 100 точки – стига да се сетят как да представят графа. Самото решение не е нищо повече от DFS/BFS.

Състезателите в тази група може би все още не са виждали похвата "разширяване" на графа (в най-простата си форма), който е нужен за решаване на тази задача. Може би най-очевидното решение е, за всяка дума да търсим клетка от дъската, където тя може да започва, след което да правим рекурсия за да проверим дали тя наистина се среща, започвайки там. Това решение всъщност работи много добре за произволен вход (рандом букви по дъската), като би хванало около 50 точки (което беше казано и в условието). Все пак, защо не би хванало останалите тестове?
Time limit. Решението е базирано на пълно изчерпване, като макар и при рандом вход да работи страхотно, можем лесно да излъжем с малко по-хитър вход. Един прост пример е:

```
3 11
aaaaqweaaaa
aaaaasdaaaa
aaaazxcaaaa
1
aaaaaaaaaaqweaaaaaaaaaa
```

При този тест възникват няколко проблема с досегашното ни решение. Първо, не е много ясно от кое 'a' да тръгнем. В почти 50% от случаите бихме били обречени – ако тръгнем от някое от 'a'-тата от дясната страна няма да намерим думата (но нея всъщност я има, ако тръгнем от някои от 'a'-тата от лявата страна). По-големият проблем, обаче, е че на всяка стъпка имаме много избори накъде да продължим – често по 4 възможности. Това прави решението експоненциално дори само за една единствена дума (O(4^L), където L е дължината на думата). Наистина, можете да забележите, че макар и да има няколко решения (възможни разположения на думата в таблицата), те са сравнително малка част от общия брой пътища, които започват с част от думата. В тестовете са дадени дори по-брутални примери, чрез които биват залъгвани различни решения.

Как да процедираме тогава? Проблемът на досегашната ни идея е, че понякога имаме повече от един избор накъде да продължим. Например ако тръгнем от горната лява клетка (с координати (ред, колона) = (0, 0)), имаме избор надолу и надясно. Независимо коя от двете изберем, после можем да се върнем в клетката с координати (0, 0), а също така да отидем в (1, 1). Това, което трябва да забележим, е че *независимо* кой от двата пътя сме избрали, ако се озовем в, например, (1, 1) и сме покрили три букви (първите три 'a'-та от думата чрез или пътя (0, 0)->(0, 1)->(1, 1), или чрез (0, 0)->(1, 0)->(1, 1)), то няма значение как точно сме стигнали. Решението оттук нататък е независимо от досега направените избори (досегашния път).

Как можем да използваме това? До сега имаме нещо като граф (може да си представим всяка клетка от таблицата като връх в граф, имащ ребра само към съседните (до) 4 клетки. Това коя дума търсим в графа, както и докъде сме стигнали в

нея (колко букви вече сме "покрили"), налага известни допълнителни ограничения. Както казахме, ако знаем текущата клетка и докъде сме стигнали в думата, не ни интересува как сме стигнали до тук, нали? Тук идва въпросното "разширяване" на графа.
Всеки от досегашните върхове (разбирайте клетки) ще мултиплицираме 50 пъти (колкото е максималната дължина на думата), като ще отиваме в различно копие (клонинг) на клетката, в зависимост от това колко букви сме покрили от думата. Например, в случая с клетка (1, 1) ще отидем в третото копие, тъй като сме покрили 3 букви от думата. Така на всяка стъпка ако до сега сме били в C-тото копие на някой връх, на следващата стъпка ще сме в C+1-вото копие на някой от неговите съседни върхове. Забележете, че дори на следващата стъпка да се върнем обратно в предишната клетка, то реално ще сме в друг връх – C+2-рото копие на тази клетка.

Как ни помага това? Вече всеки връх еднозначно ни определя цялата информация, която ни интересува – в коя клетка на таблицата се намираме и докъде сме стигнали с покриването на думата. Можете ли да се сетите какво ще се опитваме да постигнем?

Ами, логично, да покрием цялата дума – тоест да се озовем L-тото копие на някой връх (без значение кой), където L е дължината на думата. Това, че сме там означава, че вече сме покрили L нейни символи – тоест всички такива – тоест сме намерили думата.

Колко е голям този граф? Казахме, че разклоняваме всяка клетка на таблицата 50 пъти (можем и по-малко за по-кратки думи). Броят на клетки в таблицата е N * M, при N, M <= 50. Значи имаме 50 * 50 * 50 = 125000 върха! Това не е никак малко, но графът е изключително рядък – всеки от този върхове е свързан с най-много 4 ребра. Както казахме, търсим дали има път от първото копие на някоя клетка, до L-тото такова. Как можем да намерим такъв път? Просто – някой от стандартните алгоритми за търсене на път в граф без цени по ребрата (търсене в дълбочина (DFS) или търсене в ширина (BFS)) би ни свършил чудесна работа!

Остава да пуснем по едно DFS или BFS за всяка дума, като максималният брой думи K е отново 50. Тъй като сложността на всяко от DFS/BFS-тата е O(V + E) (при V = 125000, E = 500000 в най-лошия случай), а броят думи K е 50 в най-лошия случай, то сложността на цялото решение става O(|Wi| * N * M * K). Това е около 25 милиона операции в най-най-лошия случай (взимайки предвид константата 4 за ребрата). Което на модерни процесори не е много и върви за доста под секунда.

Задачата е поучителна не заради основния алгоритъм, който прилагаме в нея (BFS или DFS), а заради похвата с разширяването на графа, който ще срещнете и в много други (включително значително по-сложни) задачи.

Автор: Александър Георгиев