**Анализ**

Първата подзадача е само за 5 точки. Тя е дадена за хората, които реализират директно описаното в условието без никакви оптимизации. Проверяваме за всяка позиция в писмения източник дали се среща двумерната дума. Най-простата проверка дали се среща на дадена позиция думата е с два вложени цикъла да се прегледа за съвпадение на съответните букви. Общо трябват четири вложени цикъла или сложност .

Втората подзадача е за 15 точки. Идеята е да забързаме проверката дали думата се среща на дадено място. Като цяло се прилага сравнително стандартна техника при този тип задачи – *rolling hash*. Първо намираме хеша за всеки ред от думата, а след това с *rolling hash* намираме хеша на всяка ***C***-торка от символи в писмения източник. Така при проверката за съвпадение на думата е достатъчно да обходим само редовете и да видим дали съответния хеш в източника е същият като хеша на съответния ред на думата. Аналогично можеше да се гледат хешовете по колони, но няма значение при големи тестове, които са с квадратни матрици. Тук сложността е или .

Третата подзадача е за 30 точки. Допълнително забързваме проверката за срещане до константа. За да направим това трябва да ползваме двумерен хеш за таблица, което е неприятно и освен това малката памет, която имаме, няма да ни позволи. Всъщност тя е толкова малка, че дори нямаме достатъчно памет да си запазим целия вход 😊. Отново ще хешираме редовете на думата. Сега обаче ще продължим с хеширането, така че да получим едно число. Приемаме хешовете на редовете като редица от числа и използваме същия начин за хеширане. Очевидно този начин можем да приложим и при хеширането на писмения източник, като направим и по двете измерения *rolling hash*. Съответно минималната памет, която можем да използваме сега ще е – след като минем ***R*** реда от входа, изваждаме от текущите хешове по колони (които съответстват на матрици ***R***x***C***) хешовете на съответните ***C***-торки на ***R*** реда преди сегашния и той вече няма да ни трябва. Използвайки оптимизацията на паметта и този двоен *rolling hash*, проверката е константна за срещане на думата – сравняването на две числа и сложността по време е .

Последната подзадача е за 50 точки. Това, което пречи на предното решение да е пълно е отново паметта. Интересен факт е, че в последните тестове на подзадачата има контра тест на това решение с хеширане при конкретния модул на автора. Разбира се невъзможно е да се направи за всеки модул, макар че при използване на единичен хеш при това хеширане на цяла матрица би трябвало да е сравнително лесно да има колизия, а при използване на двоен или троен хеш времето на решението ще нарасне 2-3 пъти заради големите константи. Пълното решение не използва такава несигурна техника като хеширане и е чисто линейно по писмения източник. Стандартно за едномерната задача се използва *KMP* като алтернативен подход. Очевидно само това не върши работа тук, защото най-много да решим до втората подзадача, ако сравняваме за всеки ред/колона. Все пак подходът за сравняване по редове може да подскаже какво ще е същинското решение. Ако успеем за всяка ***С***-торка от последователни символи да кажем на кой ред от думата съответства или, че не съответства на нито един ред, последвалата проверка ще е лесна – ще правим *KMP* по колоните. Един детайл е какво правим ако имаме редове, които са едни и същи в двумерната дума. Този проблем можем да разрешим като преномерираме редовете, така че различни като стойност редове да имат различен индекс, а равните – еднакъв индекс. Сега остана да уточним най-важния детайл, а именно как ще преценяваме с подходяща сложност за всяка ***С***-торка от последователни символи на кой индекс на ред от думата съответстват. Но тази задача е еквивалентна на следната – имаме ***R*** думи и трябва за въведен текст (ред от писмения източник) да кажем къде и кои думи се срещат. Така стигаме до идеята да използваме алгоритъма на Ахо-Корасик за ***R***-те реда на двумерната дума. В тази задача можем да реализираме малко по-кратко алгоритъма – не ни трябват връзки при успех, защото думите са с една и съща дължина ***С***, т.е. на една позиция точно една от уникалните думи може да се среща. Освен това при изграждането на *trie* можем да преномерираме редовете на двумерната дума спрямо уникалността им. Понеже ще искаме алгоритъма да върви линейно по писмения източник, тук ще трябва да използвам масив (а не например *map*) за пазенето на дървото. Така паметта малко се увеличава и става от порядъка на в най-лошия случай, което все пак е по-добре от паметта при предното решение. Първият етап, както казахме е да построим *trie* и връзките при провал, като отново ще се възползваме от малката азбука и построения масив – ако от даден връх няма излизаща буква, то вместо невалиден индекс ще сложим индекса на съответна буква, гледайки върха на връзката при провал на съответния връх. Това ще ни позволи буквално линейно намиране в съответен ред от писмения източник къде и кои редове от двумерната дума се срещат. Вторият етап на алгоритъма ще е следният. По масива с уникалните индекси на редовете на двумерната дума ще направим *fail* масива. За да имаме отново чисто линейно търсене при *KMP* ще се възползваме от малкия брой колони и освен *fail* масива ще направим двумерен масив *next*, който ще указва по дадена позиция на ред и следващ индекс къде трябва да продължим сред редовете на двумерната дума. Строенето му е по аналогия с подобната част в алгоритъма на Ахо-Корасик. След тази подготовка (тук сме използвали само въведената двумерна дума) е третият етап – при въвеждане на писмения източник, като заради малката памет ще пазим само текущ ред. Него ще пускаме по *trie* и ще пазим къде кои редове от думата сме намерили. Паралелно ще правим *KMP* по всички колони – за всяка колона ще пазим до позицията на кой ред в думата сме стигнали (т.е. са съвпадали редовете), като новия индекс при текущата ***С***-торка за колона ще смятаме, използвайки едно преглеждане на съдържанието на *next* масива. В крайна сметка паметта се доминира от тази за дървото, т.е е , а времевата сложност се доминира от частта с пускане на редовете на писмения източник по дървото и тази при паралелните *KMP*, т.е. е

Задачата на пръв поглед е стандартно разширение на стандартната задача да търсене на модел в текст, но по-лесното решение, което е разширение на едномерното хеширане се спира от ограничението по памет (а май и от константите?). Разширението на едномерното *KMP* не е чак толкова директно и става логично с алгоритъма на Aхо-Корасик, като ако паметта беше абсолютно централна можеше да е още по-малко – със слагане на *log* фактор или амортизирана константа при използване на *map* или *unordered\_map* съответно щеше да стане само по източника. Но така времето щеше да се покачи много при големите ограничения за писмения източник. Интересното, което се оказва е, че за това решение нужната памет е само за дървото, за разлика от *rolling hash,* където ни трябва допълнително заради преизчисляването на хешовете.

*Автор: Илиян Йорданов*