**АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА**
**Fukushima**

Задачата комбинира две много стандартни и добре известни други задачи, затова е предвидена и като лесна в темата.

Ако просто ни беше даден началният граф и не ни беше разрешено да премахваме ребра, задачата щеше да е напълно еквивалентна на тази за намиране на лексикографски най-малкото топологично сортиране. В 50% от случаите е казано, че отговорът ще е -1, тоест ще сме точно в този случай. Както е намекнато и в самото условие, не може да имаме решение, ако в графа съществува цикъл. Следователно в тази част от тестовете даденият ни граф е DAG (Directed Acyclic Graph) и можем да изпечатаме -1, като след това приложим стандартния алгоритъм за да намерим лексикографски най-малкото топологично сортиране. Забележете, че насочен ацикличен граф винаги има топологично сортиране. Съществуват и по-бързи алгоритми, но малките ограничения за броя върхове (до 1000) ни разрешава да ползваме дори най-простия O(N^2) алгоритъм.
Първото нещо, което трябва да направим за него е да „обърнем" ребрата на входния ни граф – тоест ако ни е казано, че A зависи от B, ние трябва да добавим ребро от B към A. След това на всяка стъпка премахваме по един връх от графа, като за да изпълним условието за лексикографски най-малък отговор трябва винаги да премахваме най-малкия, който можем. Кога можем да премахнем връх? Когато в него не влизат ребра (тоест ако той не зависи от никой друг). За целта преди да започнем премахването на върхове, за всеки от тях трябва да преброим колко влизащи ребра има. Това става с прост O(M) цикъл по ребрата, като в deg() пазим броя на влизащите ребра. Очевидно, ако за някой връх този брой е 0, то този връх няма „бащи" и може да бъде взет на следващата стъпка.
След това правим два вложени цикъла, всеки от който от 1 до N. Тъй като трябва да изпечатаме N числа, външният цикъл реално показва до кое от тях сме стигнали. Във вътрешния трябва да намерим най-малкия връх от останалия граф, в който не влизат ребра - тоест търсим най-малкия индекс idx, за който taken(idx) == false и deg(idx) == 0. Такъв връх винаги ще има, освен ако всеки от останалите върхове не участва в поне един цикъл. След като го намерим го изпечатваме, премахваме го от графа, като с това премахваме и всички негови инцидентни ребра (те са само излизащи), като с това намаляме броя влизащи ребра на неговите съседи. Тоест foreach neighbour idxNeighbour of idx: deg(idxNeighbour)--;
С това и завършваме алгоритъма за намиране на топологично сортиране на граф, като неговата сложност (при тази имплементация) е O(M + N^2). Забележете, че броят ребра е винаги по-малък от N^2, следователно сложността се доминира от броя върхове, тоест е O(N^2).

Втората част от задачата е как да намерим най-голямото ниво на радиация, при което имаме отговор. Както казахме, за да има топологично сортиране, графът трябва да е DAG. Тоест трябва да намерим това число X, при което след премахване на всички ребра с тегло по-голямо от X, графът остава без цикли. Забележете, че колкото по-малко е X, толкова по-малко ребра остават в графа. При X == 0 графът остава без ребра, тоест тогава със сигурност имаме решение 1 2 ... N. Нещо повече, ако едно ребро не присъства в графа при някоя стойност на X, то същото ребро не присъства и за всяка стойност по-малка от X. Следователно функцията е монотонна и можем да приложим много известната техника двоично търсене. С нея можем да определим оптималното ниво на радиацията с най-много log(MAX_PRICE) питания. Тук е момента да споменем, че вместо да пишем отделна функция за намиране на цикли в графа, можем да модифицираме топологичното търсене да връща -1 (или някаква друга невалидна стойност) ако не може да завърши.
С добавяне на двоично търсене общата сложност на алгоритъма ни става O(log(MAX_PRICE) * N^2), което е напълно допустимо при дадените ограничения.

*Автор: Александър Георгиев*