**АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА**

**СНЕГОВАЛЕЖ**

Задачата Snowfall се очакваше да бъде една от по-лесните в темата, като изискваше единствено относително лесни и стандартни алгоритми.

За да бъде решена, състезателите трябваше да направят едно много лесно, но важно наблюдение:

*Докато графът е свързан, с падането на още сняг броят на опасните градове може единствено да нарасне.*

Твърдението се доказва тривиално. Повече сняг означава повече блокирани улици, тоест по-малко ребра в графа. "Опасните" градове в граф с M ребра са също опасни и ако премахнем едно или повече от ребрата. Премахването на ребро обаче може да доведе до нови опасни градове, така че техният брой или остава същия или нараства.

Наблюдението трябваше да е доста интуитивно: например, в пълен граф (с N > 2 върха) никой град не е опасен, докато в дърво с N върха всички без листата са опасни.

Има три основни подхода, които видях за тази задача:

1. Пълно изчерпване по всичко: За всяко възможно количество сняг проверяваме всеки град дали е "опасен".
2. Двоично търсене и пълно изчерпване: Правим двоично търсене по количеството сняг и за всяко избрано количество проверяваме всеки град дали е "опасен".
3. Двоично търсене и Артикулационни точки: Правим двоично търсене по количеството сняг и за всяко избрано количество ползваме умен алгоритъм да намерим "опасните" градове.

Нека разгледаме в повече детайли какво прави всеки един от тези подходи.

**Пълно изчерпване по всичко**

Първият възможен подход беше да пробваме всеки възможен отговор (количество сняг) – 1, 2, 3, … сантиметра сняг, докато стигнем до K или повече опасни градове (или графът стане несвързан).

Намирането на опасните градове също може да стане с брутфорс: за всеки град, пробваме да го блокираме и проверяваме дали графът остава свързан. Това изисква N проверки, всяка от които потенциално обхожда целия граф (със сложност O(N + M)). Тъй като знаем, че графът е свързан, знаем, че M ≥ N-1, което ни позволява да сведем сложността на обхождането на графа до O(M). Така цялата проверка кои от N-те града са опасни е O(N \* M).

Тъй като количеството сняг, което трябва да падне за да се блокира всеки от пътищата е доста голямо (до 1,000,000,000), това решение е сериозно ограничено. Сложността на тази идея е O(max(Si) \* N \* M). В тестове за около 25 точки числата бяха достатъчно малки това да върви, като награда за хората, които все пак са го имплементирали.

**Двоично търсене + пълно изчерпване**

Както споменахме при предходното решение, бавната част беше да намерим колко сняг трябва да падне за да се стигне до отговора (или да установим, че никое количество сняг не ни върши работа).

Също така по-рано споменахме, че получаваме повече опасни градове само с повече сняг. Това трябва да наведе състезателите на една много стандартна и известна техника: двоично търсене. Така можем драстично да подобрим бързодействието на решението, като намалим броя пъти, в които търсим "опасни" градове от 1,000,000,000 на около 30 (тъй като сложността на двоичното търсене е логаритмична, а двоичен логаритъм на 1,000,000,000 = 30).

В това решение отново ще ползваме същия начин да намерим опасните градове – пробвайки всеки град по отделно за O(N \* M). Така общата сложност на тази идея е O(log(max(Si)) \* N \* M), което би донесло на състезателите около 50 точки.

**Двоично търсене + Артикулационни точки**

След като заменихме пълното изчерпване от наивното решение с по-умен алгоритъм, сега е време да оптимизираме и намирането на "опасните" градове. Както се оказва, описаните "опасни" градове – върхове, които, когато бъдат премахнати, разделят графа – са добре познато и известно нещо, наречено "артикулационни точки". Съществува и (донякъде известен) линеен алгоритъм за намирането им.

Идеята е да се ползва обхождане в дълбочина на графа, като на всеки посетен връх се дава число, отговарящо на "времето" (стъпката), в което е бил посетен за пръв път. След това артикулационните точки могат да бъдат намерени ако отговарят на някое от две условия.

Нека изберем някой от върховете на графа като "корен" на рекурсията (без ограничение можем да изберем произволен връх, например връх 1). DFS-дървото от обхождането на графа реално го прави насочен, което ще ползваме малко по-късно. По всяко време, когато не сме в корена, върховете по пътя нагоре до корена ще наричаме "предшественици", а децата (и техните деца и т.н.) – "наследници".

Ако пуснем рекурсията в едно от децата на корена и поне едно друго негово дете остане непосетено, това значи, че коренът е артикулационна точка (ако го премахнем, посетеното и непосетеното дете ще бъдат разделени).

Може би забелязахте, че това е точно което правихме на всяка стъпка от O(N \* M) алгоритъма, който ползвахме до сега. Ще видим, обаче, че можем да ползваме допълнителна информация, която събираме по време на рекурсията, за да намерим с едно обхождане \*всички\* артикулационни точки.

Второто условие някой връх да е артикулационна точка е ако премахването му води до разделяне на някои от предшествениците и някои от наследниците му. За целта ще ползваме ID-тата, които дадохме на всеки от върховете (стъпката, в която сме ги посетили). Нека рекурсията в момента е в даден връх X (който не е корен на дървото). Ако под-дървото на някое от децата на X не съдържа ребро към някой от предшествениците на X, то, премахвайки X, ние ще ги разделим, съответно X също е артикулационна точка.

Можете да прочетете кратка и добре-написана статия за артикулационните точки (с графика и малко по-подробно обяснение) ето тук:

<http://www.geeksforgeeks.org/articulation-points-or-cut-vertices-in-a-graph/>

Имплементацията на алгоритъма за намиране на артикулационни точки е доста прост (~20-25 реда код), и в същото време много бърз. Можем да имплементираме всички проверки за константно (O(1)) време ако държим за всеки връх минималния връх, до който имаме ребро от неговото под-дърво (жертвайки O(N) допълнително памет). Тъй като обхождаме графа само веднъж за всяка стъпка на двоичното търсене, като вътрешните операции в рекурсията са O(1), сложността за намиране на артикулационните точки е O(N + M). Както отбелязахме по-горе, в тази задача можем да твърдим, че е и O(M).

Вземайки предвид и двоичното търсене, сложността на цялата идея става O(log(max(Si)) \* M), което е достатъчно бързо да се хванат 100 точки.

*Автор: Александър Георгиев*