АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА СРИВ

Задачата може да се разглежда като вариация на известната задача за намиране на минимален срез (Minimum Cut), като в текущия случай е приложена върху доста специален тип граф – дърво.

За улеснение ще наричаме стаите – върхове, а коридорите – ребра; стаята, в който се случва аварията – върха-цел, а потенциално-засегнатите – специалните върхове.

Анализът ще върви по различните подзадачи по нарастваща сложност:

1. Най-тривиалното решението е да се разгледат всяко подмножеството на ребрата и да се определи дали съществува път между върхът-цел и специалните върхове. Такова решение може да се реализира със сложност от $O(2^{N-1} * N)$ до $O(2^{N-1} * N^3)$ за всяка заявка в зависимост от използвания алгоритъм за определяне на свързаност в граф. Подобни решения успешно се справят с първата подзадача.
2. Всяка заявка може да бъде решена с $O(N)$ като се използва Depth-First-Search (обхождане в дълбочина) от върха цел. Такова решение се справя успешно с първите две подзадачи.
3. За да се справим с останалите 3 подзадачи е необходимо да търсим решение, което е по-бързо от $O(N)$ в случаите, в които множеството $\boldsymbol{H}$ съдържа по-малко от N елемента. Това се постига като се разглеждат само специални върхове и малка част от останалите върхове, които представляват техни най-ниски предшественици (Lowest-Common-Ancestor). Също така е нужно бързо да се намира минималното тегло на ребро между два от върховете на дървото. Тази информация (намиране на LCA за всяка двойка върхове и минималното тегло на ребро между два върха) може да бъде пресметната със сложност $O(N^2)$ за всяка двойка върхове в дървото и да се отговаря на всяка нейна единична инстанция със сложност $O(1)$. От друга страна, броя на върховете, които са интересни за всяка заявка е $O(K)$. Отговор на заявка със сложност $O(K^2)$ или $O(K^2*\log N)$ успешно се справя с третата подзадача.
4. За да се справим с четвъртата подзадача (а по този начин с първите четири като цяло) е необходимо да оптимизираме решението за отделна заявка до $O(K*\log N)$, което се постига след като се забележи, че специалните върхове всъщност са $O(K)$ и е възможно те да бъдат намерени за $O(K*\log N)$ време, а намирането на LCA между два върха в дървото и намирането на реброто с минимално тегло между два върха в дървото може да се направи с $O(\log N)$.
5. Решението на петата подзадача е идентично с това на четвъртата и разчита на едно наблюдение, което представлява само няколко реда код към решението на четвъртата подзадача.

Решения със сложности от $O(2^{N-1} * N)$ до $O(2^{N-1} * N^3)$ за заявка

Броя на ребрата в дървото е точно N–1. Достатъчно е да разгледаме всяко подмножество от тези ребра (има точно 2^{N-1} такива подмножества) и да изберем онова с най-малка сума от ребрата, в което върхът-цел е в различна компонента от

специалните върхове. Проверката за свързаност може да се реализира с различни алгоритми: Floyd–Warshall - $O(N^3)$, DFS или BFS – $O(V + E)$, което е $O(V)$, понеже броя на ребрата е O(V); използването на Disjoint-Sets структура също води до $O(V + E)$ за проверката за свързаност и $O(2^{N-1} * N^3)$.

Решение със сложност O(N) за заявка

Нека кореноваме дървото с корен – върха-цел. Трябва да намерим множеството от ребра с минимална сума, което отделя върха цел от специалните върхове. Нека опитаме да решим задачата с едно обхождане в дълбочина. За всеки връх ще намираме следната информация:

каква е минималната сума на ребра в поддървото на върха V, което отделя V от специалните върхове в поддървото на V

Важно наблюдение е, че ако имаме отговорите на синовете за даден връх V, лесно може да решим задачата за върха V.

Отговорът на всяка заявка всъщност е отговорът на горната задача за корена на дървото – R.

Решение със сложност $O(K^2)$ или $O(K^2*logN)$ за заявка

Основната идея, за да открием алгоритъм, който не обхожда всички N върха и N-1 ребра, е да забежим, че голяма част от върховете, които ще обхожда DFS-то ще имат само един наследник (стига големината на множеството H да е доста по-малка спрямо N), който има отговор различен от нула. Реално върховете, които имат отговор различен от нула, задължително съдържат поне един потенциално-засегнат връх в своето поддърво.

Нека наричаме върховете, които имат поне два наследник с отговор различен от нула, интересни. Това твърдение е еквивалентно на следното: най-нисък общ предшественик на два потенциално-засегнати върхове е интересен.

Един начин, за да открием всички интересни върхове, е да разгледаме най-ниските общи предшественици на подмножество от специалните върхове. Те обаче са до 2^K.

Нека разглеждаме специалните върхове като интересни.

1) Вземаме най-ниския необходен интересен връх (в началото това задължително е някой от специалните върхове).
2) Маркираме като обходен.
3) Намираме LCA на него и всеки от текущото множество от интересни върхове, и ако има такива, които не са в множеството, ги добавяме.

Тази процедура има сложност $O(K^2)$ и намира всички интересни върхове. Техният брой е O(K), което ще бъде доказано по-късно.

След като сме намерили интересни върхове – тези, които имат поне двама преки наследника, чиито отговори не са нула, очевидно останалите върхове или имат

отговор нула, или имат точно един наследник с отговор различен от нула. Върховете с отговор нула може да се игнорират, защото не променят нищо (добавянето на нула към сума не оказва влияние). Нека разгледаме върховете, които имат точно един наследник, с ненулев отговор.

Важно наблюдение е, че тези върхове (с един ненулев наследник) образуват пътища, които имат интересни наследници единствено в краищата си. За да сметнем отговора на дадена заявка, не може просто да игнорираме ребрата между тези върхове. Но, може да ги разглеждаме като групи, тъй като това, което намира DFS-то между два интересни върха реално предствлява минималното тегло на ребро между тях.

Лесно може да пресметнем за $O(N^2)$ минималното тегло на ребро между всяка двойка върхове като използваме прекъмпютинг. Аналогично може да намерим със същата сложност и LCA на всеки два върха. Достъчно е да ги запазим в двумерни масиси, за да може да отговоряме с константна сложност за всяка двойка върхове.

Такова решение успешно се справя с първите три подзадачи.

*Решение със сложност O(K*logN) за заявка*

Нека R е корен на дървото.

От решението на горната подзадача се вижда, че е доста важно бързо да намираме най-ниския общ предшественик на два върха в дървото и реброто с най-малко тегло между два върха. Докато в горната задача беше възможно да прекъмпютнем тази информация със сложност $O(N^2)$, то при 250'000 върхове трябва да използваме друг подход. Интересно е, че може да съчетаем тези подзадачи по такъв начин, че да намираме отговора на единична тяхна заявка за $O(\log N)$ време.

Нека дефинираме ancestor[node][2^k] като 2^k-тия предшественик на върха node. Тази информация може да бъде прекъмпютната за $O(N*\log N)$ като използваме следната рекурентна зависимост:

ancestor[node][0] = parent[node];
ancestor[node][2^k] = ancestor[ancestor[node][2^{k-1}]][2^{k-1}]

parent[node] е предшественика на върха node (или т.н. родителя на node в дървото). Използвайки тази информация за $O(\log N)$ може да се намери LCA-то на два върха.

Основната идея на горния прекъмпютинг е да намираме информация през степени на двойката. Тази информация е достатъчна, за да се намери най-ниския общ предшественик на двойка върхове в дървото за логаритмично време като използваме модифицирано двоично търсене.

Тази идея може да бъде приложена успешно за намирането на реброто с най-малко тегло между два върха от дървото (като единия задължително е предшественик на другия) благодарение на следанта зависимост:

min_cost_edge[node][0] = cost[node, parent[node]]

$$min\_cost\_edge[node][2^k] = min\_cost\_edge[min\_cost\_edge[node][2^{k-1}]][2^{k-1}]$$

Използвайки ancestor[][] и mon_cost_edge[][] информацията можем, от една страна, да намираме най-ниския общ предшественик на двойка върхове в дървото, а от друга – да намираме реброто с най-малко тегло между двойка върхове (като и двете може да се реализират за логаритмично време на заявка и O(N log N) предварителни изчисления (прекъмпютинг).

Алгоритъмът е следния:

1) Правим preorder traversal на дървото точно един път и пресмятаме prenum на всеки връх.
2) Сортираме върховете от множеството ***H*** относно техния prenum в нарастващ ред.
3) Използваме стек, в който добавяме върховете в реда от 2) и в зависимост относителната височина на LCA на последните два върха в стека и на последния и следващия връх от ***H*** определяме дали добавяме в стека следващия връх от 2) или намираме LCA на предпоследните два, премахваме ги и добавяме това LCA.

Основната идея на този стек е, че всеки два съседни върха от него ще имат LCA, които ще образуват път в дървото, като всеки следващ LCA ще е (не задължително пряк) наследник на всеки предишен.

Нека разгледаме три последн A, B, C, които са в правилен ред относно prenum обхождане (споменато горе). Има два случая:

depth[LCA(A, B)] > depth[LCA(B, C)]
В този случай LCA(B, C) е предшественик на LCA(A, B) и съответно, за да ги обходим върховете в правилния ред е необходимо да направим нужните изчисления в поддървото на LCA(A, B) първо, да премахнем A и B, и добавим в стека LCA(A, B).

depth[LCA(A, B)] < depth[LCA(B, C)]
В този случай LCA(A, B) е предшественик на LCA(B, C) и не противоречи на правилото за върховете в стека и затова го добавяме.

Решение за 100 точки

Решението за 100 точки е почти идентично с това, което решава четвъртата подзадача. Проблемът е, че може върхът-цел да е различен.

Един начин да се справим с този проблем е да забележим, че може да "компресираме" дървото за всяка заявка да съдържа единствено интересните върхове и върхът-цел. Реално създаваме ново дърво, което ще съдържа споменатите върхове, а ребро между тях ще съществува единствено, когато съответните върхове от началното дърво са свързани с път, който не съдържа нито един специален връх. Теглата на ребрата в компресираното ребро ще бъдат минималните по тегла ребра по пътя между съответните върхове в началното дърво.

Това свеждане е вярно и получаваме дърво с O(K) върха за всяка заявка. Достатъчно е просто да приложим решението на втората подзадача (обхождане в дълбочина) на компресираното дърво, за да намерим отговора на дадена заявка. Трябва да се отбележи, че такова решение се справя със задачата, дори когато има различни върхове, в които се случват авариите.

Автор: Антон Анастасов