АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА

ОЦВЕТЯВАНЕ

Да формулираме задачата формално. Дадено е дърво с корен, чийто върхове имат номера от едно до *n*, като коренът има номер 1 и всеки родител има номер, който е по-малък от номерата на синовете му. Има няколко заявки от вида (*pi*, *ki*), където *pi* е номер на връх, а *ki* е цяло положително число. Трябва да се избере интервал [*L, R*] с минимална дължина, такъв че за всеки връх *pi* да съществува връх с номер в избрания интервал, за който *pi* е предшественик от ниво *ki*.

За всяка заявка (*pi*, *ki*) да означим с *Li* списъка с номерата на върховете, които се явяват наследници на *pi* от ниво *ki*. Тогава задачата се свежда до това да бъде намерен интервал [*L, R*] с минимална дължина (и разположен най-вляво, ако има повече от един подходящ интервал), такъв че всяко множество *Li* да има поне един общ елемент с него.

Най-общата идея за решаване на задачата е следната: нека по някакъв начин сме фиксирали левия край на интервала *L* (как ще правим това ще стане ясно по-долу). Тогава *L*трябва да притежава следното свойство: ако с *vi* означим максималния номер в списъка *Li*, а с *vmin* – минималната стойност сред всички *vi*, то *L ≤ vmin*. Доказателството е тривиално – ако *L>vmin*, то ще съществува поне едно множество измежду всички *Li*, което няма общи елементи с интервала [*L, R*].

 **При фиксиран ляв край *L*** на интервала, можем да намерим най-късия интервал, който има общи елементи с всяко множество *Li* по следния начин: нека *wi* е най-малкия номер в множеството *Li*, за който е изпълнено *wi* ≥ *L.* Тогава *R ≥ wi*; От тук следва, че R=max(*wi* | *wi* е най-малкия номер в множеството *Li*, за който е изпълнено *wi* ≥ *L*; *i* пробягва индексите на всички множества *Li*).

 Да не забравяме, че все още не сме определили как ще фиксираме левия край на интервала, но все пак знаем, че *L ≤ vmin* (вижте по-горе). Тогава (при липса на по-добра идея) можем да направим следното: фиксираме последователно левия край със стойности от 1 до *vmin* и за всяка такава стойност намираме десния край на най-късия интервал **при така фиксиран ляв край,** който има поне един общ елемент с всяко от множествата *Li*. От всички такива интервали избираме най-късия и най-ляв (ако има няколко най-къси с еднаква дължина) и това ще бъде решение на задачата.

 Да оценим каква е сложността на такова решение, при положение, че всичко вършим по „най-хамалския“ начин. Броят на всички списъци *Li* може да бъде от порядъка на *O(n).* Построяването на всeки от списъците *Li* е със сложност *O(n)*, така че построяването на всички списъци е със сложност *O(n2)*. След това, ако преглеждането на всеки списък се извършва по най-обикновен начин, то при фиксиран ляв край, определянето на десния край на съответния интервал ще бъде със сложност *O(n2).* Като сложим и това, че левият край може да бъде фиксиран по *O(n)* начина, то получаваме сложност на решението *O(n3).* Такова решение се справя с подзадача 1.

 Да си представим, че сме сортирали всички списъци *Li* по нарастващ ред на номерата в тях. Това действие ще бъде със сложност *O(n2log n)* (сортирането на всеки списък е със сложност *O(nlog n)* и броят на списъците е с порядък *O(n)*)*.* Тогава за всеки фиксиран ляв край на интервала, започвайки от 1 до *vmin*, търсенето на съответните *wi* в множествата  *Li* не трябва да започва всеки път от началото на списъка, а при увеличаване на левия край с 1 може да продължи след последния елемент на списъка, до който сме стигнали при предишната стойност на левия край. По този начин, общото количество операции при преминаване на стойностите от 1 до *vmin* в качеството им на ляв край на интервала ще бъде от порядъка на общия брой на елементите във всички списъци, който е *O(n2).* Така получаваме решение със сложност *O(n2log n)*, което решава подзадача 2.

 Подзадача 3 е лесна – при нея всяко множество *Li*  се състои от точно един елемент. Намирането на интервала [*L, R*] се извършва, като се намерят съответно минималният и максималният номер на заявен (а не на заявяващ) връх, присъстващи в подадените заявки.

 Остава да решим задачата в общия случай. Всъщност това, което измислихме със сортирането никак не е лошо. Бедата е, че броят на списъците *Li* може да бъде много голям и съответно общият брой на елементите в тях може да е *O(n2).* Както е казано в условието, някои заявки могат да се повтарят, но дори и това да не се случва, може да има много двойки списъци, за които няма смисъл да се разглеждат и двата списъка. Да разгледаме пример.

 Нека дървото е следното:

Нека са дадени 3 заявки (1,1), (3,1) и (1,2). Тогава множествата *Li* са следните (преди сортирането):

*L1* = {2, 3}; *L2* = {4, 5} и *L3* = {6, 7, 4, 5}

Забележете, че *L2* се съдържа в *L3*. Като се замислим, това се дължи на факта, че връх 3 е насленик на връх 1 и ниво 1 спрямо връх 3 съвпада с ниво 2 спрямо връх 1 в общото дърво. И така ще бъде винаги, когато имаме две заявки *(Ui, ki)* и *(Uj, kj)*, такива че *Ui* е предшественик на *Uj*, а *ki* и *kj* са такива, че върховете, които са на ниво *ki* спрямо *Ui* и върховете, които са на ниво *kj* спрямо *Uj* са на едно и също ниво в общото дърво.

 Но щом *L2* се съдържа в *L3*, то всеки интервал от индекси, който съдържа елемент от *L2*, ще съдържа и елемент от *L3*. Тогава се ражда следната идея. Разглеждайки всички множества *Li*, можем да изкажем следното твърдение: две множества *Li* и *Lj*, върховете от които се намират на едно и също ниво в общото дърво, или се съдържат едно в друго (ако *Ui* е предшественик на *Uj*), или въобще нямат общи елементи. Когато се съдържат едно в друго, то по-голямото (с не по-малко елементи) можем да го изключим от разглеждането в по-горе дадения алгоритъм и по-този начин ще останат само множества, които нямат общи елементи. Но общият брой на елементите в такава съвкупност от множества е *O(n)* и сортирането на останалите списъци (тези, които имат значение) ще бъде със сложност *O(n logn)*, а последващото обхождане за търсене на десния край на интервала – със сложност *O(n).* Да видим как да премахваме „излишните“ списъци.

 Правим обхождане в дълбочина на дървото (най-добре е да го предствим със списъци на съседство), като за всяко ниво в общото дърво правим списък с върховете на това ниво в реда, в който влизаме в тях. Забележете, че елементите от едно множество *Li* ще бъдат последователни в този списък, т.е. ще образуват подсписък от последователни елементи (ако списъкът реализирате в масив или, по-добре, във вектор, то ще образуват сегмент от масива/вектора). Както казахме по-горе, два такива сегмента (множества *Li*, които се намират на едно ниво в общото дърво) или се съдържат един в друг или нямат общи елементи. Тогава за всяко ниво в дървото, сортирайки тези сегменти, лесно можем да изчистим излишните, т.е. тези, които съдържат някой друг сегмент. Сложността на цялата тази операция е *O(n logn).*

 Така сложността на цялото решение става *O(n logn).*

*Автори: Павел Петров, Руско Шиков*