**Анализ**

*Тагове: дървета, структури от данни, линеаризация на дърво, Ойлерово дърво, merge-sort дърво, sqrt декомпозиция, амортизиран анализ, двумерно сегментно дърво*

Условието описва кореново дърво (работниците са върховете) с битове във върховете. Заявките, с които трябва да се справим, са обновявания или въпроси за поддърво на някакъв връх *x*, ограничено до дълбочина *k*.

Първата подзадача е за 11 точки. Правим точно това, което условието иска от нас. Първо, построяваме дървото с битове в началото. Когато имаме обновяваща заявка (тип 1) *x* *k*, слагаме единици на всички върхове на *k*-подчинените на *x* с прост *DFS*. Когато имаме заявка за въпрос (тип 2) *x* *k*, броим единиците във върховете на *k*-подчинените на *x* с друг *DFS*. Очевидно сложността е .

Втората подзадача е за 15 точки. Тук заявките са по-прости, защото са за цели поддървета на върховете. Един начин да направим оптимално решение със сложност е да модифицираме предишното решение. Главните наблюдения са, че един връх става с бит 1 от 0 само веднъж и когато имаме заявка за обновяване за поддърво, всички последвали заявки за въпрос в него са с ясен отговор (просто броят на върховете в поддърво) и всички други заявки за обновяване в него са безсмислени. Ние нямаме нужда от толкова бързо решение и ще опишем в детайли едно друго решение, което е много по-значимо за следващите решения. Когато имаме заявки за поддърво, много полезна техника е да го линеаризираме, като попълним масив *T* с някоя специална поредица на върховете. Можем да го направим по следния начин. Нека да направим *DFS* на дървото, в който поставяме върховете в *T* на in-момента. По този начин, всяко поддърво ще бъде последователен сегмент в този масив и всеки връх ще участва точно веднъж в него (така че размера на *T* е ***N***). Това означава, че заявката за промяна е да направим всички елементи единици за някакъв сегмент на *T* и заявката за въпрос е да преброим единиците в някакъв сегмент на *T*. Нека направим сегментно дърво от *T* и в него лесно ще можем да обработваме всяка от тези заявки за . Сложността тук е .

Третата подзадача е за 17 точки. Тук имаме само заявки от тип 2 (заявки за въпрос). Отново ще линеаризираме дървото в масив *T*. Но този път ще построим така нареченото *merge-sort дърво* от дълбочините на върховете в *T*. Разбира се, ще включим само върховете с бит 1 (защото трябва да броим единици). По този начин, възел на *merge-sort tree*, който съответства на някакъв сегмент в *T*, ще има сортиран списък с дълбочините на върховете с 1 в . Нека мислим, че имаме някаква заявка за въпрос *x* *k*. Сегментът в *T* е с върховете от поддървото на *x*. Ще обработим възлите, които съответстват на тази заявка, в *merge-sort дървото*. За такъв възел можем да направим двоично търсене в списъка му за дълбочина ≤ *k* и това ще ни даде бройката на исканите върхове. Когато сумираме тези бройки, ще изчислим отговорът за заявката. Сложността тук може да бъде или , ако сме построили *merge-sort дървото* наивно.

Четвъртата подзадача е за 26 точки. Можем да направим предишното решение да обработва също обновяващи заявки, като имаме *treap-ове* във възлите на *merge-sort дървото*. Този път ще включваме в *merge-sort дървото* всички върхове (не само тези с единица). Очевидно, *x-ключът* за възлите в *treap-овете* ще бъде дълбочината на върховете. Ще ни трябва още някаква информация, но нека да помислим как можем да правим обновяваща заявка на първо време. Един голям проблем, е че когато обновяваме някакъв възел *nd* в *merge-sort дървото*, ще трябва да обновим информацията за възлите над него. Но обновяването може да е със сравнително голяма информация. Един, по-бърз начин да се справим с обновяващите заявки е за всяка такава заявка да премахнем върховете с нули в съответните *treap-ове* (т.е. ние запазваме дълбочините само на върховете с нули в *treap-овете*). Информацията за обновяването, която трябва да предадем нагоре може да е голяма, но понеже ние махаме определена 0 само веднъж, то сумарната работа за това, няма как да надвиши . Един друг подход, който няма да обсъждаме в детайли е да направим *sqrt* декомпозиция на заявки. Очакваните сложности за тази подзадача са или с големи константи.

Последната подзадача е за 31 точки. Тук сложността няма да е по-добра от тази в предната подзадача, но решението ще е доста по-бързо. Вместо да правим сегментно дърво върху линеаризацията на дървото и да построяваме някакви структури данни във възлите на дървото, ще правим обратното. Ще построим сегментно дърво върху дълбочините на върховете и за всеки възел в това сегментно дърво, ще запазим върховете със съответните дълбочини и ще направим сегментно дърво за in-времената на върховете. Тези сегментни дървета ще бъдат за сума на върховете с единица, които имат in-времена в някакъв интервал. Ако ги направим по-умно, те няма да заемат много памет или време за построяване – листата във всяко сегментно дърво ще съответстват на различните върхове във възела, за който се строи сегментното дърво. Така че, аргументът, използван за сравнително малкото време и памет в *merge-sort дървото* също важи тук! Понеже нямаме *treap-ове*, заявките за въпрос ще бъдат много по-бързи, въпреки че сложността остава същата. За заявката за промяна, ще използваме друга идея, възползвайки се от това, че всяка единица става нула само веднъж. Ще използваме сегментните дървета, за да пазим информация за всеки от техните върхове дали имат връх със останала единица или не. Когато имаме заявка за промяна, ще търсим за всеки връх с 1 в заявката и ще променяме тези върхове и съответни възли на сегментните дървета. Ако отново сумираме работата, тя няма да надвиши . Описаното решение съответства на предоставеното пълно решение, но то може да се направи и за сегментното дърво, което се построява върху линеаризацията на дървото, както в предната подзадача. Крайната сложност е също .

Задачата беше предвидена да е по-трудна, но в крайна сметка, решихме да я направим по-лесна, за да е по-подходяща за състезателния ден. Може би някой ден, оригиналната версия ще види свят.

*Автор: Илиян Йорданов  
Решения: Илиян Йорданов, Енчо Мишинев*