АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА ВЛАК

Задачата Train беше надграждане на много известна задача - тази за намиране на максимален брой непресичащи се интервали (мероприятия).

Всички или почти всички състезатели в А група трябва да знаят алчното решение на този проблем – интервалите се сортират по своя край и се взимат в този ред (като, разбира се, вземаме само тези интервали, които не се пресичат с никой от вече взетите). Това е една от известните задачи за greedy алгоритми (и всъщност 30 процента от тестовете бяха точно тази задача – тогава M = 1). Така всички би трябвало да хванат поне 30 точки.

Не много известен extension на тази задача е когато можем да имаме няколко покриващи се интервала, но не повече от M такива в която и да е точка. Интересно е, че грийдито продължава да работи – отново сортираме интервалите по техния край и взимаме всички интервали, които не биха покрили повече от M-1 от вече взетите в която и да е тяхна точка.

От тук нататък задачата е чисто техническа. Най-очевидното решение, в което просто сортираме интервалите и линейно проверяваме дали новият кандидат-интервал може да бъде взет (гледайки дали има точка, която вече да е покрита от M взети интервала) би хванало доста точки, но не е много очевидно как да се направи тази проверка.

Първият голям проблем е, че интервалите могат да бъдат потенциално много дълги. Това е сравнително стандартно и доста от състезателите трябва да са чували и да могат да ползват така наречената "компресия на координатите". Тъй като броят пътници, и, съответно, гари, които ни интересуват, са значително по-малко от общия брой гари (точки), то много от тях няма да допринасят с нищо към решението. Ако разгледаме два (пресичащи се) интервала, в началото единият от тях е сам, после вторият започва и известно време двата се пресичат, след което някой от тях свършва и другият продължава сам. Но нужно ли ни е да разглеждаме *всички* точки, в които интервалите се пресичат? Не – достатъчно е само точките, в които те почват да се пресичат и тези, в които спират да се пресичат. Но това са именно или началото или краят на някой от интервалите! Оказва се, че можем да разглеждаме единствено гарите (точките) в които интервал започва или свършва, и да игнорираме всички останали. Така L става от 1,000,000,000 на едва 2 * N = 200,000.

След като вече можем в масив да записваме колко пъти е "покрита" всяка точка от вече взети интервали, можем да направим сравнително просто O(N*N) решение. Сортираме интервалите по техния край и почваме да ги слагаме един по един, като пазим в масив коя гара колко пъти е вече покрита.

Това, обаче, също е бавно (макар и да хваща доста тестове). Всъщност част от тестовете са "гарантирани" да бъдат хванати с това решение – второто ограничение на задачата, че в други 30% от тестовете $E_i - B_i <= 100$ ни позволява точно това. Така решениет ни не е O(N*N) ами O(N*100).

Как можем да решим задачата за 100 точки? Трябва да измислим бърз начин, по който да проверяваме дали има точка в даден интервал, която да е покрита M пъти, а и също така да увеличим "покритостта" на даден интервал с 1. Лесно наблюдение е, че първото можем да решим като намерим максимално покритата точка в интервал (вместо проверка дали има точка, която е покрита M пъти). Наистина, ако максимумът в даден интервал е вече M, то ние няма да сложим този интервал и тя ще си остане M.

За целта ползваме структурата данни "интервални дървета" – в тях имаме query за интервал $[I_1, I_2]$, което ни връща максималната стойност в този интервал, а също така и update за интервал $[I_1, I_2]$ със стойност X, който добавя стойността X към всеки елемент в интервала. В нашия случай тази стойност е винаги 1, но това не ни помага особено.

Ако имплементираме горната структура данни добре, бихме могли да получим сложност O(logL) както за query-то, така и за update-а. С това решението ни става O(N * logN), тъй като имаме O(N*logN) за компресията на координатите, още O(N*logN) за сортирането им, и накрая O(logN) за N-те на брой куерита и ъпдейта (което дава общо отново O(N*logN)).

Автор: Александър Георгиев