Анализ на задача `statement`

Тагове: булеви изрази, двойкосъчетание, алгоритъм на Кун, small-to-large, неявен граф

Задачата е пример за улеснение на известната `NP-пълна` задача `SAT`. В случая нашата задача е решима полиномиално и дори оптимално за линейно време. Ако ползваме терминологията от условието, то в общия случай всеки критерий може да се среща неограничен брой пъти в списъците от изисквания. Може би това е накарало някои състезатели да си мислят, че тази задача е трудна.

Допълнително ще отбележим, че понеже всеки критерий се среща най-много общо два пъти, то общо в списъците имат най-много $2N$ изисквания, а техният брой M е от порядъка на N, затова ще оценяваме сложността само с N.

Решение на първа подзадача – 7 точки

Първо трябва да разкодираме с какво ни помага ограничението в подзадачата. При тези условия всеки списък или директно се удовлетворява като изпълним единственото му изискване, или има изискване, което може да изпълним без противоречия с други списъци. Второто е вярно, защото ограничението ни задава, че за списък с няколко изисквания в единия случай има поне едно изискване, което се среща и в друг списък (и понеже всеки критерий се среща най-много два пъти, то няма други срещания) и удовлетворяването му се пренася и за другия списък. Другият случай според ограничението е да има изискване с критерий, който всъщност не се среща в други списъци, така че директно удовлетворяваме това изискване.

Така за решаването на подзадачата е достатъчно да открием критериите, които ще наричаме *свободни*, като те ще са важни и за част от по-нататъшните решения. Това са такива критерии, чието изпълняване можем директно да фиксираме, защото или се срещат само веднъж в изискване, или се срещат с едно и също желание за изпълняване в две изисквания (които трябва да са едни и същи). Когато ги открием, всички списъци с повече от едно изискване се удовлетворяват, като изпълним изискване със *свободен* критерий (а такова има винаги заради ограничението). След това директно удовлетворяваме списъците с по едно изискване. Единствено трябва да следим за случая с два списъка с по едно изискване, което е различно желание за изпълняване на един и същ критерий. Тогава имаме директно противоречие и нямаме решение.

Сложност: $O(N)$.

Решение на втора подзадача – 7 точки

Това е по-странична подзадачи от другите и няма особено връзка с решенията. Понеже всеки критерий се среща най-много общо два пъти, то ограничението задава сравнително проста структура на списъците. Всеки по-голям списък включва критериите на нула, един или повече по-малки списъци, като те не могат да включват критериите от други списъци и помежду си нямат общи критерии. Това означава, че за да получим решение е достатъчно първо да удовлетворим по-малките списъци и след това по-големите. Наистина, ако стигнем до даден списък и всички критерии в него вече са с фиксирано изпълняване, то означава, че всички негови критерии са се съдържали в по-малки списъци и то са били единствените в тях (иначе щяхме да имаме поне един нефиксиран критерий от по-малките списъци). Може да се види, че ако имаме два списъка с еднаква дължина и с едни и същи критерий, то тази обосновка остава вярна.

Така за решаването на подзадачата сортираме списъците по дължина в нарастващ ред и всеки път удовлетворяваме списък, като търсим вече удовлетворено изискване (с фиксиран критерий от по-рано) или изискване с все още нефиксиран критерий. Ако не може да намерим такова изискване, то няма решение. Може да забележим, че този подход решава и предната подзадача.

Сложност: $O(N)$ с *count sort* или $O(N \log N)$.

Решение на трета подзадача – 18 точки

Ограничението в тази подзадача превръща задачата в частен случай на `2-SAT` задачата. Съществува линейно решение на общия случай на тази задача, може да прочетете повече за него тук. В зависимост от имплементацията може да е леко трудно минаването на най-големите тестове, но една лесна оптимизация е предварително да заделим по 2 клетки за векторите за списъците на съседите (все пак всеки критерий се среща най-много общо два пъти).

Заради простатата структура съществува директно много по-лесно решение. Нека разгледаме произволен списък и удовлетворим първото изискване. Критерият в него може да се среща в друг списък с противоположно желание за изпълняване. Но това означава, че за да удовлетворим този списък трябва да има поне още едно изискване. Понеже всеки списък има най-много две, то може или да няма такова, или да има точно едно, което задължително трябва да удовлетворим. Така може да продължим по същия начин, докато или достигнем удовлетворен списък, или удовлетворим изискване, чийто критерий не се среща с противоположно желание в друг списък, или не успеем да удовлетворим списък. В първия и втория случай успешно сме удовлетворили част от списъците, като няма противоречия с останалите списъци. В третия случай намираме противоречие и това означава, че началният избор е грешен и не трябва да удовлетворяваме началното изискване, а трябва да удовлетворим другото изискване (ако съществува).

Така цялата идея е да обхождаме последователно списъците и за все още неудовлетворените да изпробваме всички варианти. За даден вариант откриваме всички критерии, които трябва да се фиксират заради този избор, и ако не получим противоречие, продължаваме с обхождането на списъците. Ако получим противоречие за всички варианти, то нямаме решение. Този алгоритъм е линеен, защото ще посетим всеки списък най-много 3 пъти.

Сложност: $O(N)$.

Решение на четвърта подзадача – 5 точки

Подзадачата е за класическо решение с пълно изчерпване. Генерираме всички варианти за изпълняването на критериите и търсим вариант, който удовлетворява всички списъци.

Сложност: $O(2^N N)$.

Решение на пета подзадача – 47 (=7+7+0+5+28) или 65 точки (=7+7+18+5+28)

Задачата е много подходяща за решение с *matching*. Нека първо се оправим със *свободните* критерии. Тях може да ги фиксираме директно, което ще удовлетвори част от списъците. Така за всеки от останалите критерии знаем, че се срещат в две противоположни изисквания. Това директно ни определя двуделен граф, като в левия дял ще сложим останалите критерии, а в десния дял – списъците. Ще свържем с две ребра всеки критерий със съответните два списъка, в които се среща. Понеже се среща в противоположни изисквания, то можем да удовлетворим точно един от списъците и съответно едното ребро е фиксиране на това критерият да не бъде изпълнен, а другото – да бъде изпълнен. Това е точно задачата за намиране на *maximum matching* – искаме по възможност за всеки връх от десния дял да изберем по едно ребро, свързващо го с връх от левия дял (така ще бъде удовлетворен съответния списък и ще се фиксира изпълняването на някой от критериите).

Ако използваме алгоритъма на Кун за намиране на *maximum matching* (повече информация тук), то може да хванем всички досегашни подзадачи и дори тестове с $N, M \leq 10^5$. Това е заради специфичната структура на нашия двуделен граф. Алгоритъмът на Диниц макар да има по-добра сложност в най-лошия случай, се държи значително по-бавно.

Сложност: $O(N^2)$.

Пълни решения – 100 точки

Решение с резолюция

Има решение с метод от математическата логика, наречен ”резолюция”, който се използва при автоматизирани доказателства на теореми. Идеята е, че когато имаме два списъка за удовлетворяване с противоположни изисквания, ако съставим списък, в който сложим всички изисквания без тези двете, то той също трябва да е удовлетворен. Това действие се извършва, докато се получават нови списъци. Ако се получи празен списък, то това означава, че не всички списъци е можело да бъдат удовлетворени, защото няма как да удовлетворим празния списък.

В нашата задача улеснение за този метод имаме от това, че всеки критерий се среща най-много общо два пъти. Това означава, че за всеки критерий може да прилагаме най-много веднъж резолюцията. Затова решение е да минем през всички критерии и за даден критерий да проверим дали има два различни списъка, в които той се среща в противоположни изисквания, и ако е така, да направим резолюция. По принцип при резолюция трябва да премахнем изискванията за критерия, спрямо който сме я извършили, да премахнем дублирания на изисквания и да проверим дали не получаваме списък с противоположни изисквания, който вече е удовлетворен.

За да е бърз този подход, ще добавяме по-малкия списък към по-големия на принципа *small-to-large*, като за бързодействие може да не трием по-малкия, а само да отбелязваме кои списъци са все още активни. Когато получим обединения списък, няма да премахваме изискванията за критерия, спрямо който сме направили резолюцията, както и да премахваме дублиращи се изисквания. Единствено ще следим, ако се получи списък с противоречиви изисквания, защото този списък вече е удовлетворен и няма нужда да участва в последващи резолюции. След този процес е достатъчно да минем през активните списъци и да проверим дали някой от тях не е станал празен като не гледаме критериите от резолюциите. Ако това е така, то няма решение, иначе използваме изискване с такъв критерий, за да удовлетворим съответния списък.

За да са удовлетворени всички начални списъци трябва да се върнем назад. Знаем, че всеки обединен списък вече е удовлетворен, но при обединяването е имало два списъка, за които трябва да осигурим, че са удовлетворени. Най-лесният начин е да запишем всички обединявания в един стек и после да ги обходим. Съответно при всяко обединение обхождаме по-късия списък и намираме дали е удовлетворен спрямо фиксираните досега променливи. Фиксираме изпълняването на критерия, спрямо който е направена резолюцията, така че да удовлетворим този списък, ако не е удовлетворен, а ако е, то така че другият списък да е удовлетворен.

Сложност: $O(N \log N)$.

Най-добро решение

Загатнахме за това решение още в трета подзадача, но сега ще го формализираме. Нека приемем, че нямаме *свободните* критерии. Ще направим следния неориентиран граф, който ще ни помогне да знаем какво е последствието от фиксирането на даден критерий. Върховете на графа ще са списъците, а критериите ще са ребрата. По-точно, ще добавим по едно ребро за всеки критерии между двата списъка, в които се среща с противоположно изискване. Това ни позволява да извършваме удовлетворяването на списъците като *DFS*. За да удовлетворим даден връх (списък), фиксираме някой негов критерий и тогава съседният му връх относно реброто, съответстващо на критерия, трябва да използва някой друг критерий за удовлетворяването си. От друга страна съседните върхове относно останалите ребра могат да използват критериите от тези ребра, за да бъдат удовлетворени, съответни техните съседни върхове – критериите от ребрата, които ги свързват с предишните съседни върхове и т.н. Това означава, че единствено е неясно дали можем да удовлетворим останалите върховете, които срещаме като тръгнем от реброто на фиксирания в началото критерий.

Можем да забележим, че има само един случай, когато имаме проблем и това е ако имаме компонента дърво. Както и да фиксираме критериите, то винаги ще стигнем до някое листо, за което няма да можем да ползваме критерия, свързващ го с баща му, защото той вече ще е използван за удовлетворяване на бащата. Ако имаме цикъл, винаги ще имаме един допълнителен критерий. Най-просто за фиксирането е да започнем от връх, участващ в цикъл. Ако направим покриващо дърво от него, то началният връх можем да удовлетворим с критерий, който е за ребро извън покриващото дърво. Това ще означава, че за всички други върхове можем да използваме за удовлетворяване критериите, свързвайки ги с бащите. Можем дори да избегнем започването от специален връх, както е направено в авторовата реализация.

Остана да изясним какво става ако имаме *свободни* критерии. Тогава, ако в дадена свързана компонента има *свободен* критерий, тя винаги ще има решение, защото съответният връх ще бъде удовлетворен чрез този допълнителен критерий (спрямо компонентата). Отново това може лесно да бъде обобщено и реализирано с модификация на едно *DFS* обхождане.

Сложност: $O(N)$.

Алтернативно решение

Това решение в началото беше опит за *cheat*. Идеята е всеки път да удовлетворяваме списъци, за които това е еднозначно. Ще следим за всички досега неудовлетворени списъци колко от критериите им не са фиксирани. Докато има списъци с точно един нефиксиран критерий, удовлетворяваме тези списъци, като фиксираме изпълняването на критерия спрямо съответното изискване. За тези списъци вече нямаме друга възможност. По някое време ще останем с необработени списъци, всички от които имат 0, 2 или повече нефиксирани критерии. Интересуваме се от тези, които са с 2 или повече. Оказва се, че ако има решение, няма значение кой от тези списъци ще изберем и ще удовлетворим чрез някой от останалите нефиксирани критерии. Това най-лесно се вижда ако си представим графа от предното решение, защото когато сме в тази ситуация, то при "махането" на ребро при удовлетворяването или няма да се получи компонента дърво без свободен критерий, или ако се получи такава, то ще сме "махнали" реброто, свързващо двете ѝ листа и един от върховете вече ще е удовлетворен.

Ако накрая има списък, за който всички критерии са фиксирани, но той не е удовлетворен, то нямаме решение. Имплементирането става с два вектора – един с директните за удовлетворяване списъци (с останали ≤ 1 нефиксирани критерии) и друг с останалите необработени списъци (с останали 0 или ≥ 2 нефиксирани критерии). Главното ни итериране е по първия списък, а когато той е празен – по вторият, докато или първият вече не е празен, или приключим с всички списъци. За повече детайли може да разгледате реализацията в `statement_100p_alt.cpp`.

Сложност: $O(N)$.

Оказва се, че тази задача е от видовете задачи, в които е достатъчно да се тръгне в някакво направление за откриване на пълно решение и обикновено то ще води до решение. Това може да се види и при алтернативното решение. Съществуват и други идеи за пълно решение, например една вдъхновена от намиране на Ойлерови цикли.

Остава отворен въпросът каква е сложността в най-лошия случай на алгоритъма на Кун в специалния вид двуделни графи. На някои тестове константата е по-голяма, между $O(\log N)$ и $O(\sqrt{N})$, но не бяха конструирани тестове, така че да се достига $O(N)$.

Идея: Атанас Димитров
Реализация: Илиян Йорданов