Анализ на задача `plans`

Тагове: силно-свързани компоненти, DAG, топологично сортиране, динамично програмиране, включване-изключване, FFT, NTT

Условието задава, че искаме да преброим ориентираните прости графи (без мултиребра и примки) с номерирани (различими) върхове, които имат K силно-свързани компоненти. Наистина свойството, което е описано в условието – избиране на най-много върхове, които могат да се подредят, така че да отговарят на пътищата в графа, ни позволява да вземем най-много 1 връх от всяка силно-свързана компонента. От друга страна кондензираният граф на силно-свързаните компоненти (всяка силно-свързана компонента е превърната във връх и ребрата вече са между силно-свързаните компоненти) е *DAG* и има топологична наредба, така че като вземем по един връх от силно-свързана компонента можем да използваме топологичната наредба на съответните силно-свързани компоненти, за да получим правилна наредба на върховете.

За краткост като говорим за графи, ще разбираме ориентирани прости графи, а като броим графите винаги ще считаме, че техните върхове са номерирани.

Решение на първа подзадача – 9 точки

Предвидено е да направим пълно изчерпване на всички възможни графи. Възможните ребра са $N(N-1)$, така че броят графи е $2^{N(N-1)}$. За всеки граф намираме броя силно-свързани компоненти и проверяваме дали е равен на K. Може да се използва и по-бавен алгоритъм за това.

Сложност: $O(2^{N(N-1)}N^3)$.

Решение на втора подзадача – 12 точки

Понеже $K = N$, то търсим броя *DAG*-ове. Ограничението на подзадачата за N ни насочва, че вече не можем да изчерпваме възможните ребра, но можем да правим някакво изчерпване за върховете. За да е ацикличен графа, трябва да има топологично сортиране. Полезно е да си мислим как се намира то с итеративния алгоритъм. Започваме със *source* върховете (тези без влизащи ребра), след което ги премахваме и получаваме нови *source* върхове спрямо оставащия граф и т.н. докато се изчерпят всички върхове. Така можем да разглеждаме върховете разделени на нива спрямо топологичния им ред в графа – в нулевото ниво са началните *source* върхове, в първото ниво са следващите получени и т.н. Всички ребра са от върхове от едно ниво към върхове от следващите нива.

Ако пробваме по този начин да преброим графите, ще установим, че някои ще бъдат броени по няколко пъти. Например нищо не задължава върхове от първо ниво да не са част от нулевото ниво – ако няма ребро, което да влиза в тях, то те със сигурност ще са от нулево ниво. За да оправим този проблем е достатъчно за всеки връх от ниво $l > 0$ да сложим влизащи ребра от върховете от ниво $l - 1$. Така тези ребра задължават върховете да се разделят топологично по точно този начин на нива.

Ясно е, че точните номера на върховете във всяко ниво не променят броя ребра, затова можем да направим пълно изчерпване за броя върхове във всяко ниво. Това е еквивалентно числото N да го разделим на няколко положителни събираеми, като редът има значение. Ако имаме разделянето c_0, c_1, c_2 ($c_0 + c_1 + c_2 = N$), то възможните ребра са:

$$(2^{c_0} - 1)^{c_1}(2^{c_1} - 1)^{c_2}(2^{c_0})^{c_2}$$

Така осигуряваме за всеки връх от ниво 1 поне едно влизащо ребро от ниво 0, за всеки връх от ниво 2 поне едно влизащо ребро от ниво 1, както и възможни ребра между върховете от ниво 0 и ниво 2. Това изчерпва всички възможни ребра за това разделяне, като допълнително трябва

всеки път да смятаме и броя начини за това кои върхове ще са във всяко ниво чрез комбинации. Трябва да направим преизчисляване на всички комбинации, всички нужни степени на двойката, както и на $(2^x-1)^y$ (за ребрата между съседни нива), за да смятаме директно броя графи, които се получават при фиксирано разделяне на нива.

Сложност: $O(2^N N)$ (всички разделяния са 2^{N-1}).

Решение на трета подзадача – 30 точки (=0+12+18)

Ако използваме предната идея е сравнително лесно да оптимизираме смятането като забележим, че за всяко ниво е достатъчно единствено да знаем колко върхове е имало в предишното ниво (за ребрата между съседните нива) и общо колко върха е имало в останалите нива (за ребрата от върхове през поне едно ниво). Така можем да смятаме $dp[left][prev]$, което брои възможните разделяния и ребра за останалите $left$ на брой върхове, ако в предното ниво е имало $prev$ на брой върхове. Като фиксираме броя върхове в текущото ниво $curr$, то получаваме следната рекурентна зависимост:

$$dp[left][prev] = \sum_{curr=1}^{left} \binom{left}{curr} (2^{prv}-1)^{curr} (2^{n-left-prv})^{curr} dp[left-curr][curr]$$

Сега ще покажем по-добър подход за решение, който е полезен за крайното решение. Нека дефинираме подобно динамично, но малко по-общо и по-естествено: $dp[N][S]$ – брой ациклични графи с N върха и имащи S на брой *source* върха, а $dp[N] = \sum_{S=1}^{N} dp[N][S]$. Ще разгледаме всички възможни ребра от *source* върховете към останалите. Отново имаме подобен проблем, че част от останалите върхове е възможно и те да се превърнат в *source* върхове за началния граф, ако в тях не влиза ребро. Но сега ще оправим проблема като извадим графите, които стават с повече от S *source* върха. Ако разгледаме един такъв граф и той има S' *source* върха, то ние знаем точно колко пъти ще сме го броили – $\binom{S'}{S}$, всеки възможен начин в началото да сме почнали с S от неговите *source* върхове. Така имаме следната рекурентна зависимост:

$$dp[N][S] = \binom{N}{S} (2^S)^{N-S} dp[N-S] - \sum_{S'=S+1}^{N} \binom{S'}{S} dp[N][S']$$

Интересно е, че можем да премахнем второто измерение, като е по-лесно да го направим комбинаторно с директно смятане на $dp[N]$ – броя ациклични графи с N върха. Нека преброим графите, в които даден връх е *source*. Този брой е $2^{N-1}dp[N-1]$, а сумарно за всички възможни върхове е $N \times 2^{N-1}dp[N-1]$. Сумата обаче не е равна на $dp[N]$, защото ще броим по няколко пъти графите с повече от един *source* връх. Подобно можем да сметнем броя графи, в които дадени два върха са *source*: $2^{N-2}dp[N-2]$. Тях сме ги броили два пъти в началната сметка, така че сега трябва да ги извадим, за да стане точно: $N \times 2^{N-1}dp[N-1] - \binom{N}{2} \times 2^{N-2}dp[N-2]$. Новата сметка брои правилно графите с 1 и 2 *source* върха, но тези с 3 не участват (те са броени $\binom{3}{1}$ пъти при тези с 1 и $\binom{3}{2}$ пъти при тези с 2 или общо $\binom{3}{1} - \binom{3}{2} = 0$). Това всъщност е точно включване-изключване на A_i – множеството от графите, в които връх i е *source* връх. Така ако използваме формулата за включване-изключване получаваме:

$$dp[N] = \left| \bigcup_{i=1}^{N} A_i \right| = \sum_{s=1}^{N} (-1)^{s+1} \binom{N}{s} (2^s)^{N-s} dp[N-s]$$

Тази сметка всъщност използва точно първите събираеми от формулата за $dp[N][S]$. Алгебрично може да се получи същото, ако се разпише $\sum_{S=1}^{N} (-1)^{S+1} dp[N][S]$, откъдето да се

изведе формулата за $\sum_{S=1}^{N} dp[N][S] = dp[N]$, но това не представлява интерес и го оставяме за упражнение.

Сложност: $O(N^3)$ или $O(N^2)$.

Решение на четвърта подзадача – 59 точки (=9+12+18+20)

Общата задача може да бъде решена с подобен подход. Вместо *source* върхове е удачно да разглеждаме *source* компоненти – такива, в които не влизат ребра от върхове извън компонентата. Ако фиксираме една *source* силна-свързана компонента и пробваме да я навържем с останалите, то пак ще броим по няколко пъти графите с няколко *source* компоненти. Затова ще въведем $dp[N][K][S]$ – брой графи с N върха, K силно-свързани компоненти и S *source* компоненти. Също ще въведем $dp[N][K] = \sum_{S=1}^{K} dp[N][K][S]$. Имаме следната рекурентна зависимост за $dp[N][K][S]$, подобна на предходните:

$$dp[N][K][S] = \sum_{c=S}^{N-(K-S)} \binom{N}{c} dp[c][S][S] \times 2^{c(N-c)} \times dp[N-c][K-S] - \sum_{S'=S+1}^{K} \binom{S'}{S} dp[N][K][S']$$

Има две допълнително неща спрямо преди – трябва да разглеждаме колко върха ще заделяме за дадени компоненти и освен това по колко начина можем да свържем тези върхове в S силно-свързани компоненти, което е равно на броят графи, в които всяка силно-свързана компонента не е свързана с другите или $dp[c][S][S]$.

За жалост, тази рекурентна зависимост не може да се използва за смятане на стейтовете от вида $dp[N][S][S]$, защото се получава, че $dp[N][S][S] = dp[N][S][S]$. Затова ще въведем допълнително динамично само за този случай $dp'[N][S]$ – броят графи с N върха, състоящи се от S независими силно-свързани компоненти. Смятането ще стане като фиксираме върховете в една от силно-свързаните компоненти, разгледаме броя начини те да образуват силно-свързана компонента, която информация е в dp', и за останалите върхове използваме пак dp'. Така всяка възможност в $dp'[N][S]$ ще бъде броена S пъти, защото ще фиксираме всяка от компонентите за първа. Получаваме тази рекурентна зависимост:

$$dp'[N][S] = \frac{1}{S} \left(\sum_{t=1}^{N-(S-1)} \binom{N}{t} dp'[t][1] \times dp'[N-t][S-1] \right)$$

Отново има същият детайл, че не можем да смятаме $dp'[N][1]$, което представлява броя силно-свързани графи с N върха. Можем обаче да забележим, че когато смятаме $dp[N][K][S]$, ние нямаме нужда от $dp'[N][1]$ освен когато трябва да сметнем $dp[N][1][1]$. Но тогава е достатъчно тривиално от всички графи с N върха да извадим получените резултати за $dp[N][K]$ с $K \geq 2$. Това вече оправя всички проблеми и получаваме начин да смятаме цялостното динамично.

Сложност: $O(N^4)$.

Решение на пета подзадача – 73 точки (=9+12+18+20+14)

От сега нататък само трябва да оптимизираме смятането на динамичното. Смятането на dp' в момента става със сложност $O(N^3)$, така че ще се концентрираме върху главното динамично. За тази подзадача е предвидено да премахнем третото му измерение по подобие на динамичното за броя ациклични графи. Можем да смятаме $dp[N][K]$ чрез включване-изключване на A_V – множеството от графите, в които върховете от V са една *source* силно-свързана компонента. По

аналогия ще получим рекурентна зависимост за $dp[N][K]$:

$$dp[N][K] = \sum_{S=1}^{K}(-1)^{S+1} \sum_{c=S}^{N-(K-S)} \binom{N}{c} dp'[c][S] \times 2^{c(N-c)} \times dp[N-c][K-S]$$

Сложността за смятане ще е същата, но вече константата е по-малка и смятането работи по-бързо от преди.

Сложност: $O(N^4)$.

Решение на шеста подзадача – 83 точки (=9+12+18+20+14+10)

Можем още да подобрим константата, ако разменим реда на вложеното сумиране и преобразуваме формулата:

$$\begin{aligned} dp[N][K] &= \sum_{c=1}^{N}\sum_{S=1}^{c}(-1)^{S+1}\binom{N}{c} dp'[c][S]2^{c(N-c)} \times dp[N-c][K-S] = \\ &= \sum_{c=1}^{N}\binom{N}{c}2^{c(N-c)}\sum_{S=1}^{c}(-1)^{S+1} dp'[c][S] \times dp[N-c][K-S] \end{aligned}$$

Това помага най-вътрешната част да е само умножения на две числа и дори предвид малкия модул, можем просто да натрупваме сумата в променлива от тип `long long` и да смятаме по модул чак след като сметнем съответната вътрешна сума.

Сложност: $O(N^4)$.

Пълно решение – 100 точки

Последното преобразуване е важно и защото ни позволява вече да оптимизираме сложността. Може да се забележи как при най-вътрешната сума dp' и dp се използват с фиксирани редове, а номерата на колоните се сумират до K. Това всъщност може да се разглежда като намиране на определен коефициент при умножението на два полинома с коефициенти съответно $dp'[c][0], dp'[c][1], \ldots, dp'[c][N]$ и $dp[N-c][0], dp[N-c][1], \ldots, dp[N-c][N]$ (в ред от нулевата до N-тата степен на полинома). Затова последната оптимизация е преди да почнем да смятаме динамичните за дадено N, да намерим произведенията на полиноми, които ще ни трябват. Това ще са произведения между полиномите съответстващи на $dp'[1]$ и $dp[N-1]$, $dp'[2]$ и $dp[N-2]$, …, $dp'[N]$ и $dp[0]$.

Ефективно произведения на полиноми е възможно да се намират чрез бързото преобразуване на Фурие – *FFT* за време $O(N \log N)$. Разбира се, при малки N то не е толкова ефективно спрямо директното умножение на полиноми за $O(N^2)$. Затова е добре да се използва поне итеративния вариант, но ако се приложи директно не би трябвало да вземе повече от пета подзадача. Ако оптимизираме малко – например преизчислим предварително индексите, на които отиват коефициентите при смятане или пък комплексните числа, които се използват, то трябва да може да се изкара и шеста подзадача. За пълен брой точки трябва да се приложи още една оптимизация. Вместо всеки път да превръщаме полиномите в точковидна форма, можем да преизчисляваме точковидната им форма и директно да ги умножаваме, когато имаме нужда. Последното е малко по-досадно, защото при увеличаване на N започва да ни трябва точковидна форма за по-голяма степен на двойката, понеже се използват повече корени на единицата. Но не е бавно да ги преизчисляваме наново, когато започне да ни трябва по-голяма степен на двойката.

Допълнително модулът е специално подбран да е сравнително малък (за дробните сметки при *FFT*) и да е в удобен вид за *NTT* (Number theoretic transform), защото $12289 = 6.2^{11} + 1$,

а 11 е примитивен корен на 12289, откъдето 11^6 по модул 12289 е 2048-ми корен на единицата при този модул. Решение с *NTT*, в което са приложени горните оптимизации, е най-бързото и успява да се различи значително от оптимизираното $O(N^4)$ решение в предната подзадача.

Сложност: $O(N^3 \log N)$.

Поучителната част в задачата е, че макар някои подходи за броене да не изглеждат много надеждни, защото броят неща по много пъти нетривиално, пак е възможно да има удобен начин за оправяне. В случая смятането чрез фиксиране на една *source* силно-свързана компонента брои графите с няколко компоненти много пъти, но при фиксиране на броя на *source* силно-свързаните компоненти, сравнително лесно можем да се оправим да ги смятаме точно.

Идея: Радослав Димитров
Реализация: Илиян Йорданов