Тагове	На пълното решение	На подзадачите
	„Малки и големи заявки“ Разширение на граф	Обхождания на граф

Анализ

В условието е дефиниран граф, където върховете са потребителите, а ребрата между тях са приятелствата. В задачата има двутипни заявки, съответно за добавяне на ребро и за въпрос дали два върха имат общ съсед. Във всяко решение на подзадачите, графът се представя със списък на съседите, като по този начин се обхождат съседите на върховете.

Подзадача №1

В тази подзадача е тестовият пример. Тя е за обратна връзка от системата.

Подзадача №2

Втора подзадача е предвидена за всякакви overkill-ове, които са бавни.

Очаквана сложност: $O(Q \times (N + M))$.

Имплементация: Не успях да напиша нещо, което има логика и хваща втора подзадача, което да не хваща трета ☺. Все пак вярвам, че на състезанието някой ще успее да го направи.

Подзадача №3

Какъв е най-лесният начин за проверка дали два върха имат общ съсед съсед? Обхождат се съседите на единия връх от заявката и в булев масив се отбелязват с `true`. Обхождат се и съседите на другия връх и се проверява дали някой от тях е отбелязан с `true`. След проверката се занулява булевият масив.

Постигната сложност: $O(Q \times N)$.

Имплементация: `Second_15p.cpp`

Подзадача №4

В условието от подзадачата е казано, че в графа няма цикли, следователно графът е гора. В гора задачата с добавяне на ребра е много по-забавна, но на състезанието не предоставих възможност да се реши. Коренуваме всяко дърво в гората в някой от върховете му. Два върха в графа имат общ съсед, само когато се намират в едно и също дърво и едно от двете условия е спазено:

1) Имат общ родител
2) Родителят на единия връх е дете на другия.

Тези проверки могат да се постигнат с предварително обхождане на графа. За всеки връх се намира родителя му, дълбочината му и in и out времената му в дървото. С in и out времената и дълбочината се следи второто условие, като се проверява дали единият връх е в поддървото на другия и разликата в дълбочините им е точно 2.

Постигната сложност: $O(N + M + Q)$.

Имплементация: `Third_18p.cpp`

Подзадача №5

За да се реши подзадачата, е нужно да се навлезе в същината на решението на задачата. Често, когато има заявки в граф, и той не е дърво, те се отговарят с похвата, описан в следващите изречения. Върховете се разделят на два вида, съответно малки и големи. Малките върхове са тези, които са със степен $\leq \sqrt{M}$. Останалите върхове, а именно тези със степен $> \sqrt{M}$, са големите. Забележете, има най-много $\sqrt{M} - 1$ големи върха. Тогава заявките могат да се разделят на три вида, съответно:

- Между два малки върха
- Между малък и голям връх
- Между два големи върха

Сега въпросът е как да се отговаря всяка заявка от трите вида. За отговор на всяка заявка ще се стремим за максимум $\mathrm{O}(\sqrt{M})$ сложност.

Заявка между два малки върха

Прилага се подходът от трета подзадача.

Сложност за заявка: $O(\sqrt{M})$.

Заявка между малък и голям връх

За всеки голям връх се пази булев масив с N елемента `areConnected`$[\sqrt{M}][N]$ ($\sqrt{M}$ заради броя на големите върхове). Съседите на големия връх се отбелязват с `true`. Обхождат се съседите на малкия връх и в случай, че някой от тях е отбелязан с `true` в булевия масив на големия връх, те имат общ съсед.

Сложност за заявка: $O(\sqrt{M})$.

Заявка между два големи върха

Най-лесният начин да се отговарят заявките е да се преизчисли отговора за всяка една възможна от тях. В имплементацията отговорът се пази в масива `acquaintances`$[\sqrt{M}][\sqrt{M}]$.

Сложност за заявка: $O(1)$.

Преизчисление

За да се изчислят `areConnected` и `acquaintances`, е нужно да се направят няколко линейни обхождания. За всеки голям връх е нужно да се знае кои върхове са му съседи (могат да се достигнат с 1 ребро) и с кои върхове има общ съсед (могат да се достигнат с 2 ребра). За целта може да се ползва DFS или BFS обхождане на графа, като се разшири с $[2]$ измерение. Всъщност и отговорите на заявките между малък и голям връх могат да се преизчислят и да се отговарят за $O(1)$ време, но отговорите им няма да може да се поддържат при добавяне на ребра.

Описаният похват може да го срещнете като „малки и големи заявки“, заради това съм го отбелязал така в таговете в анализа. Една примерна задача с него е hideandseek от InfO(1)Cup 2022 година. Предвиденото решение за 100 точки на тази задача не е с този похват, но с малко чийтване успях да изкарам 100 точки на състезанието.

Постигната сложност: $O((N + M + Q) \times \sqrt{M})$.

Имплементация: `Forth_51p.cpp`

Подзадача №6

Заявките могат да се отговорят по начина, описан в горната подзадача, само с една лека промяна. Тъй като в графа вече може да има до $M + Q$ ребра, условието дали един връх е малък или голям, ще бъде промено. Вече, един връх би бил малък, ако има степен $\leq \sqrt{M + Q}$, и голям, при степен $> \sqrt{M + Q}$. С добавяне на ребра вече може един малък връх да стане голям. Когато се случи това, се пуска линейното обхождане, описано в горната подзадача. Също, стойностите на `areConnected` и `acquaintances` може да се променят. `areConnected` може много лесно да се следи как се променя, въпросът е как се променя `acquaintances`.

Поддържане на `acquaintances`

За да се поддържа `acquaintances`, за всеки връх се пазят съседите му, които са големи. Когато се добави ребро между два върха x и y, като x е голям, се обхожда всеки голям съсед на y. Аналогично е обхождането, когато y е голям. Забележете, добавянето на ребро в графа променя само стойности в `acquaintances`, които са `false`.

Постигната сложност: $O((N + M + Q) \times \sqrt{M + Q})$.

Имплементация: `social_100p.cpp`

П.П: Задачата беше псевдо-интерактивна, за да бъде наложено онлайн отговаряне на заявките. Очаквам, че има чийтове, които биха искали да прочетат първо самите заявки и после да ги отговорят.

Автор: Борис Михов