Анализ на задача sequential

Анализът ще премине върху идеите на решението, а не подзадачите.

Решение чрез алгоритъма на Мо

Първо да обосновем избора ни за алгоритъм – с изключение на някои доста езотерични задачи, задача със заявки често ще се решава чрез сегментно дърво / коренова декомпозиция / алгоритъм на Мо. Така че е добра стратегия участник да отдели малко време на всяка от идеите.

В конкретния случай офлайн заявките и липсата на ъпдейти са добри индикатори, че алгоритъма на Мо, може да доведе някакво добро решение. Ако се опитаме да набутаме някакво сегментно дърво / коренова декомпозиция доста ще ни затрудни писането на *merge* функция – всеки връх отговаря за твърде много информация, която няма как леко да се мести.

Сега задачата ни се свежда до нуждата от структура от данни, която да обработна следните заявки:

- включване на число;
- изключване на число;
- намиране на дължината на най-дългия интернал $[l, r]$, за който всички числа са включени в структурата.

Сегментното дърво сега е перфектно за нуждите ни – имаме онлайн заявки и ъпдейти, а за всеки връх е достатъчно да държи информацията за най-дългия си "включен" префикс и суфикс. *merge*-функцията също не е сложна.

Тук обаче решението ни ще стане $O(n\sqrt{n}\log n)$, което обикновено е сигнал, че не сме си дорешили задачата. Добре написано решението дотук ще изкара 49 точки.

Разкарване на сегментното дърво

Във финалното решение въобще няма да ползваме сегментно дърво. Обикновено, силата му е в бързи заявки и ъпдейти, но тук се оказва, че нямаме полза от това. При алгоритъма на Мо ще ползваме $O(q\sqrt{n})$ пъти операциите "включване на число" и "изключване на число", но само $O(q)$ пъти операцията "намиране на дължината на най-дългия интернал $[l, r]$, за който всички числа са включени в структурата". Видимо ще бъдем облагодетелствани от структура от данни, която по-бързо обработва ъпдейти, от колкото заявки.

Такава не мога да измисля. 😊

Изначална промяна на алгоритъма на Мо

Тук е добре участник да е запознат с алтернативно изпълнение на алгоритъма на Мо – за всяка заявка $[l, r]$ ще намерим бъкета, в който се намира лявата граница. Ще групираме всички заявки така по бъкети. Сортираме ги по нарастващо r и *разделяме всяка заявка на две части* – $[l, B]$ и $[B + 1, r]$. Вторите части на всички заявки в група ще са лесни за обработване, защото те ще споделят една и съща лява граница $B + 1$, а десните им граници са нарастващи. Първите части пък са сравнително малки – дължината им е по-малка от $\sqrt{n}$. Можем едновременно да обработваме първите и вторите части на заявките, но все още нямаме структура от данни, която ефективно да обработва горните три заявки.

Можем да ползваме `dsu c rollback` (наистина включването на дадено число x може да се моделира като добавяне на ребра към $x - 1$ и $x + 1$, стига те да съществуват и да са предварително включени. Така заявката ни става "намери най-голямата компонента в графа") – връщането на някакви добавени ребра ще се случва когато сменяме бъкета, в който работим, или когато обработваме първата част на някоя заявка. И двете са ясно ограничени от $O(q\sqrt{n})$ връщания. Така финалната сложност възлиза на $O(q\sqrt{n}\alpha(n))$.

Може би е полезно да отбележим, че първоначалния алгоритъм на Мо нямаше да работи дори с `dsu c rollback`, понеже чрез него можем да премахваме само последно добавеното ребро (като в стек), а при нормално Мо трябва да премахваме неща от началото и края на интервала (като в deque).

Допълнителни материали

- Наблюдението да опростим ъпдейтите за сметка на заявките се споменава тук.
- Тренировъчни задачи след тази са Div.2 Round 419 problem E и Educational Round 46 problem F.

Автор: Виктор Кожухаров
Анализ: Иван Лупов