Анализ на задача tree

Подзадача 1 и 2

Първите две подзадачи са предвидени за brute-force стил решения. При $n \leq 50$ можем за всеки елемент във всеки интервал $[l, r]$ да пуснем dfs. За $n \leq 300$ ще сме малко по-хитри и първо ще маркираме всеки връх, от който трябва да пуснем dfs, а в последствие ще пуснем един dfs, който ще маркира връх само ако е минал пред вече маркиран връх.

Подзадача 4

Условието може да бъде обобщено по следния начин "Намерете сумата на броя на маркираните върхове за всеки интервал". Така зададено то подвежда да извършим $O(n^2)$ процеса и да съберем някаква статистика за всеки от тях – прекалено много. Нека преформулираме условието така: "За всеки връх намерете при колко интервала той ще бъде маркиран – сумирайте тези резултати." Така вече търсим статистика върху всеки връх, а не върху всеки процес (тук приложихме така наречената sum-of-contributions техника). Да си представим как изглежда задачата от гледната точка на някакъв връх u – той може да бъде маркиран или от себе си, или от някой от родителите си в дървото. Нека тези върхове (или по-точно, съответните им числа) означим с $a_1, a_2 \cdots a_k$ – интересуваме се колко интервала $[l, r]$ има, такива че съществува $1 \leq i \leq k$, за което $l \leq a_i \leq r$. Тук по-лесно да търсим интервалите, които покриват нула от дадените числа и да ги извадим от общия брой валидни интервали. Тяхната бройка е равна на броя двойки (x, y) за $1 \leq x \leq y \leq (l-1)$ за $l = a_{i+1} - a_i - 1$ за всяко валидно i. Тук това може да се направи линейно по броя родители на всеки връх, защото идеалните двоичните дървета с n листа имат дълбочина $\leq \log_2 n$. Така сложността е $O(n \log_2 n)$. В общия случай такова решение ще е със сложност $O(nh)$.

Подзадача 5

Решението остава концептуално същото като на миналата подзадача – единствената съществена разлика е, че искаме да поддържаме множество от родителите, в което бързо да можем да добавяме родители, махаме родител (конкретно да rollback-ваме последното добавяне на родител) и да питаме колко интервала $1 \leq l \leq r \leq n$ покриват някой от родителите. Това са все лесни операции, ако ползваме *std::set*.

Когато добавяме родител (i), ние всъщност премахваме броя интервали, които са между най-големия, по-малък от него (l) и най-малкия, по-голям от него (r) (в езика на *std::set*, това са upper_bound и lower_bound съответно) и добавяме интервалите между $l+1$ и $i-1$ и интервалите между $i+1$ и $r-1$.

Rollback-овете са точно обратното на добавянията. Събиранията стават вадения, а добавянето на елемент в множество става изтриването му.

Анализ: Иван Лупов