Кратко въведение към LCA

Предварителни изчисления

Бързото намиране на LCA е възможно благодарение на dp таблица, която предварително създаваме (техниката се нарича Binary Jumping/Binary Lifting). Тази таблица в клетка dp[i][j] пази 2^j-тия предшественик на връх i (dp[i][0] = родителят на i). Генерирането на таблицата става чрез проста рекурентна зависимост – dp[i][j] = dp[dp[i][j-1]][j-1] (тъй като скокът е 2^j, можем да го разбием на два скока 2^{j-1} – стойности, които вече сме изчислили).

Защо това работи? Нека си представим, че искаме да направим скок с размер k. Тъй като всяко естествено число има единствено представяне в двоична бройна система, то винаги ще има възможност да направим скок k чрез последователност от подскоци с размер 2^j. Например при $k=5_{(10)}=101_{(2)}$, можем да направим един скок с размер 2^0 и един с размер 2^2 (и наистина, 5 = 1+4). Така никога няма да ни се наложи да направим повече от $\log_2(k)+1$ подскока.

Генерирането на тази таблица е необходимо в първите два метода за намиране на LCA, описани по-надолу.

Три бързи начина за намиране на LCA

1. Изравняване на дълбочините

За всеки връх в графа пазим неговата височина в dep[i]. Когато имаме заявка за LCA(A,B), фиксираме A да бъде върхът с по-голяма дълбочина (dep[A]>dep[B]). Нека разликата в дълбочините бъде diff=dep[A]-dep[B]. Тогава можем да представим … в двоична бройна система (както е описано по-горе) и да симулираме подскоците, повдигайки A. Така гарантираме, че A и B са вече на еднаква височина. Остава само да ги придвижим нагоре, за да достигнем техния LCA. За целта ще постъпим така: за всяка степен j от максималната възможна към най-малката възможна, проверяваме дали dp[A][j]!=dp[B][j] и изпълняваме подскока

само ако това условие е изпълнено. Необходимо е методът да се извърши по този начин, защото ако dp[A][j]==dp[B][j], няма как да сме сигурни дали не сме прескочили LCA(A,B).

Когато сме готови с описания алгоритъм, извеждаме dp[A][0] – в крайна сметка заради начинът на подскачане, двата върха A и B остават различни до последно, но се намират точно под техния общ предшественик. Затова извеждаме техния родител като отговор.

2. Таймер и проверка за предшественик

Нека обхождаме даденото дърво в дълбочина. За всеки връх ще пазим t_in[i] – моментът в който за пръв път влизаме при даден връх, и t_out[i] – моментът в който сме обходили цялото поддърво на този връх. Изчисляването на тези масиви може да стане чрез глобална променлива, която се увеличава на всяка стъпка.

Когато искаме да намерим LCA(A,B), ще използваме следната идея:

Нека имаме функция isAncestor(A,B), която връща true ако A е предшественик на B. Проверката е лесна – ако t_in[A] <=t_in[B] && t_out[B]<=t_out[A], връщаме true, а иначе – false. Сега, пробвайки за всяка степен j от най-голямата към най-малката, ще повдигаме A само ако isAncestor(dp[A][j],B)==false. Накрая, както в предишния алгоритъм, ще завършим точно под търсения предшественик. Затова накрая извеждаме dp[A][0] като отговор.

3. Ойлеров тур и RMQ (чрез Sparse Table)

Правим Ойлеров тур на дървото и в масив pos[i] пазим на коя позиция се намира връх i в реда от тура. Освен това правим втори масив dep[i], който пази дълбочините на върховете. Построяваме sparse таблица tab[][] по този dep[] масив. Сега намирането на LCA(A,B) се свежда до това да се намери минималната стойност в tab[][] в интервала между pos[A] и pos[B], и да се изведе номерът на върха на тази позиция с минимална стойност (все пак се интересуваме от индекса на LCA(A,B), а не само от неговата дълбочина).

Задачи за упражнение

https://cses.fi/problemset/task/1750

https://codeforces.com/contest/587/problem/C

https://codeforces.com/contest/609/problem/E

https://codeforces.com/contest/519/problem/E

https://arena.olimpiici.com/#/catalog/533/problem/101326

https://codeforces.com/contest/1142/problem/B

https://usaco.org/index.php?page=viewproblem2&cpid=817

https://codeforces.com/contest/1843/problem/F2

https://usaco.org/index.php?page=viewproblem2&cpid=648

*в някои от задачите е необходим само Binary Lifting;

*задачите са подредени по относителна трудност – от лесни към трудни.