4A – Areny

autorzy zadania: Mateusz Radecki i Marek Sokołowski

Treść

Dany jest graf skierowany o n wierzchołkach, w którym z każdego wierzchołka wychodzi przynajmniej jedna krawędź. Musimy rozwiązać następujące zadanie dla każdej liczby k z przedziału $[1, n]$:

Załóżmy, że każdy wierzchołek jest biały lub czarny. W jednym ruchu możemy kliknąć dowolny czarny wierzchołek o indeksie nieprzekraczającym k, a losowy wierzchołek, do którego prowadzi krawędź z klikniętego wierzchołka, staje się czarny (niezależnie od tego jakiego koloru był). Mówimy, że para wierzchołków (v, u) (gdzie $v \neq u$) jest dobra, jeśli zaczynając ze stanem, w którym wszystkie wierzchołki poza v są białe, a wierzchołek v jest czarny, jesteśmy w stanie wymusić, by w skończonej liczbie kroków kliknąć wierzchołek u. Zadaniem jest policzyć dobre pary wierzchołków.

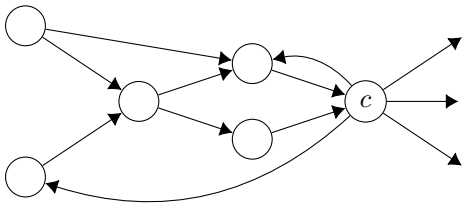
Rozwiązanie

Po pierwsze spróbujmy lepiej zrozumieć losowy proces opisany w treści zadania. Skoro musimy wymusić, by coś stało się w skończonym czasie, to możemy założyć, że wszystko co w opisanym procesie jest losowe, jak tak naprawdę dla nas złośliwe i dzieje się tak, jakbyśmy tego nie chcieli. Możemy więc o opisanym procesie myśleć tak, jakby istniał drugi gracz, który (dla ustalonej pary wierzchołków (v, u)) działa tak, byśmy na pewno nie byli w stanie dojść do wierzchołka u (o ile to dla niego możliwe).

Na początek spróbujmy obliczyć wynik dla $k = n$, czyli załóżmy, że możemy klikać we wszystkie wierzchołki. Spróbujmy policzyć, dla ustalonego wierzchołka u, ile istnieje wierzchołków v, że para (v, u) jest dobra. Jeśli z każdego wierzchołka wychodzi przynajmniej jedna krawędź, która nie prowadzi do u, to drugi gracz może przy każdym kliknięciu jakiegokolwiek wierzchołka zamalowywać na czarno wierzchołek różny od u. Oznaczałoby to, że nie istnieją takie dobre pary aren. Jeśli zaś istnieje wierzchołek (nazwijmy go w), z którego wszystkie krawędzie prowadzą do u, to jeśli on byłby czarny, to klikając w w, na pewno zamalujemy na czarno wierzchołek u, w który następnie będziemy w stanie kliknąć.

Jeśli znajdziemy taki wierzchołek w, to zmienia się jeszcze jedna rzecz: teraz, gdy istnieje wierzchołek z, z którego wszystkie krawędzie prowadzą do u lub do w, to jeśli jest on czarny, to również na pewno będziemy w stanie kliknąć w u. Gdyby drugi gracz zamalował na czarno wierzchołek u, to oczywiście możemy go kliknąć. Jeśli zaś zamaluje wierzchołek w, to wiemy, że jesteśmy w stanie wymusić kliknięcie u. Sprawia to, że para (z, u) również jest dobra. Proces ten możemy kontynuować. Powiedzmy, że wierzchołek x jest dobry, gdy $x = u$ lub para (x, u) jest dobra. Zawsze, gdy istnieje wierzchołek y, z którego krawędzie prowadzą do dobrych wierzchołków, to oznacza to, że on również jest dobry. Jeśli zaś nie istnieje taki wierzchołek, to należy zakończyć proces.

Umiemy zatem w czasie wielomianowym znaleźć wszystkie dobre pary wierzchołków. Musimy jednak zatroszczyć się, by cały proces przebiegał szybko. Wprowadźmy pojęcie „grupy" wierzchołków oraz „czoła" grupy. Niech grupą wierzchołków będzie podzbiór wierzchołków wraz ze wszystkimi krawędziami, które z nich wychodzą. Każda grupa powinna posiadać jeden wierzchołek, który nazywamy czołem. Wszystkie krawędzie z wierzchołków różnych od czoła muszą prowadzić do wierzchołków, które należą do grupy. Krawędzie z czoła mogą prowadzić zarówno poza grupę, jak i do niej. Dodatkowo, nie może istnieć cykl składający się jedynie z wierzchołków należących do grupy i niebędących jej czołem. Przykładowa grupa, z czołem zaznaczonym literą c, może wyglądać następująco:



Ustalmy więc, na początku działania algorytmu, że każdy wierzchołek jest niezależną grupą. Naszym zadaniem będzie wykrywać sytuację, gdy wszystkie krawędzie z pewnej grupy x (a dokładniej z jej czoła) trafiają do tej samej grupy y (oraz $x \neq y$).

W tym celu, możemy dla każdego wierzchołka v, wylosować pewną wartość $rand[v]$. Dla każdego wierzchołka v, będziemy również pamiętać wartość $hash[v]$, będącą sumą po wartościach $rand[head[u]]$ po wszystkich krawędziach $v \to u$. Wartość $head[v]$ oznacza czoło grupy, do której aktualnie należy v. Znając dowolną krawędź wychodzącą z v łatwo jest sprawdzić, czy grupę, w której czołem jest v, należy podłączyć do innej grupy. Jeśli ta krawędź wychodzi do wierzchołka u, to łączyć grupy musimy gdy $hash[v] = deg(v) \cdot rand[head[u]]$ oraz $head[u] \neq v$, gdzie $deg(v)$ jest liczbą krawędzi, które wychodzą z v.

Pytanie, które musimy sobie zadać, to jak sprytnie łączyć grupy, aby skończyć z dobrą złożonością czasową. Odpowiedź jest dość prosta: możemy przepisywać mniejszą grupę do większej! Za każdym razem, gdy musimy połączyć dwie grupy, iterujemy się po wszystkich wierzchołkach, należących do mniejszej z nich oraz po wszystkich krawędziach, które do nich wchodzą. Dzięki temu możemy łatwo aktualizować wszystkie wspomniane wartości i wykrywać kiedy należy połączyć jakieś grupy. Złożoność czasowa będzie taka, jak zawsze gdy korzystamy z przepisywania mniejszej grupy do większej – $\mathcal{O}(m \cdot \log(n))$ (gdzie m jest liczbą krawędzi w grafie). Podczas implementowania, warto rozważyć niereprezentowania swojej grupy przez jej głowę, co może ułatwić przepisywanie. Koniecznie jednak musimy pamiętać, który wierzchołek jest czołem grupy.

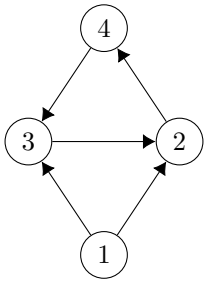
W momencie, w którym nie musimy już łączyć żadnych grup, warto zastanowić się, jak jakaś grupa może wyglądać. Nie może być dobrych par wierzchołków, w których oba wierzchołki należą do różnych grup, więc możemy traktować je kompletnie niezależnie. Ustalmy zatem jedną, konkretną grupę, i nie przejmujmy się innymi.

Na razie zapomnijmy o krawędziach wychodzących z czoła. Wiemy, że każda ścieżka z każdego wierzchołka w grupie prowadzi w końcu do czoła. Dla każdego z nich ustalimy pierwszy taki wierzchołek, przez który na pewno prowadzi każda ścieżka wychodząca z niego (i nazwiemy go $next[v]$). Zauważmy, że jest to jeden konkretny wierzchołek, ponieważ jeśli startując z wierzchołka v na pewno przejdziemy przez wierzchołek u oraz na pewno przejdziemy przez wierzchołek w, to albo startując z u przejdziemy przez w, albo startując z w przejdziemy przez u, ale nie mogą zachodzić oba. Dzieje się tak, ponieważ grupa jest acykliczna. Z tego samego względu istnieje dla niej porządek topologiczny. Znajdźmy go zatem (lub zapamiętajmy go podczas łączenia grup). Rozważajmy wierzchołki w grupie (poza czołem) w kolejności narzuconej przez porządek topologiczny, zaczynając od tych najbliżej czoła. Powiedzmy, że rozważamy teraz wierzchołek v. Prosto przekonać się, że graf wyznaczany przez wartości $next[u]$ dla już rozważonych wierzchołków u, jest skierowanym drzewem ukorzenionym w czole grupy. Nieco trudniej przekonać się, że wartością $next[v]$ będzie najniższy wspólny przodek wszystkich wierzchołków u, dla których istnieją krawędzie $v \to u$, we wcześniej wspomnianym drzewie. Bardziej doświadczonych zawodników do uświadomienia sobie tego może przekonać fakt, że podzadanie, które właśnie rozwiązujemy, to szukanie dominatorów w grafie skierowanym. Powinniśmy zatem, w trakcie budowania drzewa, dla każdego dodawanego do niego wierzchołka, obliczać dla niego jump pointery, które pozwolą efektywnie znajdować najniższego wspólnego przodka.

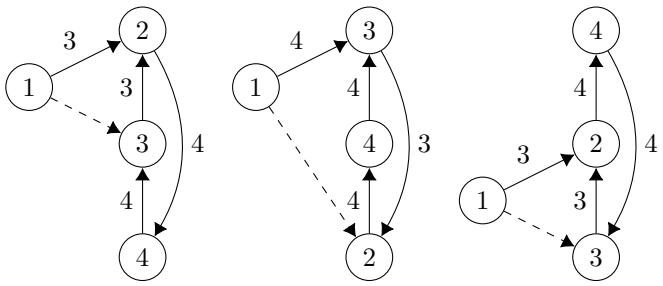
Musimy rozważyć teraz dwa przypadki: albo z czoła grupy prowadzi jakakolwiek krawędź poza nią, albo wszystkie krawędzie wychodzące z niego prowadzą z powrotem do niej. W tym pierwszym para wierzchołków (v, u) jest dobra wtedy i tylko wtedy, gdy u jest przodkiem v w drzewie wyznaczanym przez wartości $next$. W tym drugim wiemy, że po wyjściu z czoła na pewno wrócimy do tej samej grupy, więc być może istnieją też dobre pary wierzchołków, w których czoło jest pierwszym wierzchołkiem. Pierwszy wierzchołek, który na pewno napotkamy wychodząc z czoła, to znowu LCA wszystkich wierzchołków, do których prowadzą krawędzie wychodzące z czoła. Jeśli zatem ustawimy taką wartość $next[c]$, gdzie c jest czołem, to skierowane drzewo zamieni się w skierowaną meduzę. Dla niej również dość prosto jest obliczyć liczbę par wierzchołków takich, że z pierwszego istnieje ścieżka do drugiego.

Umiemy zatem w złożoności $\mathcal{O}(m \cdot \log(n))$ obliczyć wynik dla $k = n$, ale nie jest to jednak koniec zadania. Spójrzmy na jakiś wierzchołek v i na jego wartość $next[v]$. To, co chcielibyśmy zrobić, to znaleźć wierzchołek o największym indeksie, który leży na jakiejkolwiek ścieżce z v do $next[v]$ (wliczając w to v oraz $next[v]$). W tym celu możemy zmodyfikować działanie wcześniej wspomnianych jump pointerów. Skoro chcemy, by każda krawędź miała wagę, to owe jump pointery zamiast trzymać jedynie informację „w jakim wierzchołku wylądujemy skacząc o 2^p w górę", trzymać też informację „jakie jest maksimum z wag na krawędziach, które miniemy skacząc w ten sposób". W ten sposób w przypadku, gdy grupa jest drzewem, dla każdej dobrej pary aren (v, u), zacznie ona liczyć się do wyniku, gdy k będzie co najmniej takie, jak maksymalna wartość na ścieżce między nimi. Jest tak, ponieważ drugi gracz może wybrać owy wierzchołek o maksymalnym indeksie i prowadzić nas jedynie do niego, a jeśli k jest mniejsze niż indeks tego wierzchołka, to nie będziemy mogli w niego kliknąć.

Jeśli zaś grupa jest meduzą, to chciałoby się powiedzieć coś bardzo podobnego. Dokładniej, że dla dobrej pary wierzchołków (v, u) zacznie się ona liczyć do wyniku od chwili równej maksymalnej wadze krawędzi na ścieżce od v do u w meduzie. Niestety, pojawia się pewien problem. Spójrzmy na poniższy przykład:



W zależności od tego, czy czołem zostanie wierzchołek 2, 3 czy 4, możemy w trakcie obliczania meduzy otrzymać różne drzewa (i później meduzy) wyznaczane przez wartości *next*:



Jak łatwo zauważyć, środkowa meduza jest niepoprawna i opisana wyżej metoda liczenia nie zwróci poprawnego wyniku. Dzieje się tak, ponieważ dla wierzchołka nieleżącego na cyklu nie musimy dokładnie wiedzieć, na który wierzchołek na cyklu musimy natrafić najpierw. Innymi słowy, musimy natrafić na każdy wierzchołek na cyklu, ale nie musimy mieć pewności co do tego, na który natrafimy jako pierwszy. Co jeśli natrafimy na złą meduzę i jak w ogóle to wykryć? Po pierwsze zauważmy, że sam cykl nigdy się nie zmieni – zawsze kolejność wierzchołków oraz wagi krawędzi na nim pozostaną takie same – dla każdego wierzchołka na cyklu wiemy, jaki inny wierzchołek z cyklu musimy napotkać jako pierwszy. Być może więc jesteśmy w stanie ustanowić inny wierzchołek leżący na cyklu czołem i przeliczyć całą meduzę od nowa? Okazuje się, że tak! Poprawnym czołem musi być wierzchołek leżący na cyklu, z którego wychodzi krawędź o maksymalnej wadze (spośród krawędzi leżących na cyklu w dowolnie policzonej meduzie).

Dlaczego tak jest? Wybierzmy jakikolwiek wierzchołek należący do meduzy, który nie leży na cyklu, ale krawędź bezpośrednio z niego wchodzi w wierzchołek leżący na cyklu. Nazwijmy go v. Nazwijmy też kolejne wierzchołki leżące na cyklu $c_1, c_2, \ldots, c_d$, gdzie c_d jest czołem. Oznaczmy też wagę krawędzi prowadzącej z c_i do c_{i+1} (lub z c_d do c_1) jako w_i. Powiedzmy, że znamy rosnący ciąg indeksów $j_1, j_2, \ldots, j_p$ taki, że wierzchołki $c_{j_1}, c_{j_2}, \ldots, c_{j_p}$ mogą być pierwszymi wierzchołkami leżącymi na cyklu, które napotkamy po wyjściu z v. Dodatkowo powiedzmy, że maksymalnym indeksem wierzchołka, który musimy minąć, aby dojść do cyklu (czyli zalicza się do tego również v oraz każdy z wierzchołków $c_{j_1}, c_{j_2}, \ldots, c_{j_p}$), jest q. Jeśli czołem ustanowiliśmy właśnie wierzchołek c_d, to wagą krawędzi, która w meduzie prowadzi z v do c_{j_p}, jest większa z dwóch wartości: q oraz maksimum z wag $w_{j_1}, w_{j_1+1}, \ldots, w_{j_p-2}, w_{j_p-1}$. Okazuje się, że to dobrze! Aby dojść do wierzchołka c_{j_p} możemy być zmuszeni przejść przez wierzchołek o właśnie takim indeksie, za to nie możemy być zmuszeni przejść przez nic innego. Aby dojść do c_{j_i} dla $i \neq p$, możemy po pierwsze być zmuszeni przejść przez wierzchołek q, ale również by przejść przez krawędź o wadze w_d (ponieważ możemy być zmuszeni wejść na cykl w wierzchołku c_{j_p}). Jako, że waga w_d jest największa ze wszystkich, to uwzględnione wartości są poprawne.

Zastanowienie się czemu uwzględnione maksimum będzie poprawne dla wierzchołków nienależących do $c_{j_1}, c_{j_2}, \ldots, c_{j_p}$ oraz dokończenie dowodu pozostawiamy jako ćwiczenie dla czytelnika.

Zostajemy zatem z następującym zadaniem: mając dane skierowane drzewo z wagami na krawędziach lub skierowaną meduzę z wagami na krawędziach, należy dla każdej liczby powiedzieć, ile jest par wierzchołków takich, że maksimum na ścieżce między tymi wierzchołkami jest równe właśnie tej liczbie. Sumy prefiksowe, użyte na wspomnianym ciągu, będą ostatecznym wynikiem w zadaniu. Możemy oczywiście łatwo sprowadzić przypadek drzewa do przypadku meduzy, poprzez dodanie cyklu długości 1 w korzeniu drzewa. To zadanie jest dość standardowym ćwiczeniem na pracę z drzewami oraz meduzami i da się je rozwiązać na wiele sposobów, jednak przedstawimy jedno z podejść przyjętych przez autorów zadania.

Meduzę możemy potraktować jako cykliczny ciąg ukorzenionych drzew, gdzie każdy korzeń ma wychodzącą skierowaną krawędź do korzenia następnego drzewa. Zacznijmy od policzenia wyniku dla par wierzchołków należących do tego samego drzewa. Narzucającą się techniką jest użycie dekompozycji centroidowej. Znajdźmy centroid drzewa i policzmy maksimum na wszystkich ścieżkach przechodzących przez niego. Oczywiście wszystkie ścieżki, które uwzględnimy, muszą prowadzić od jakiegoś wierzchołka do jego przodka – w końcu drzewo jest ukorzenione. Mając w ręku centroid, możemy policzyć maksima na ścieżkach od niego do każdego z jego potomków (będących w tym samym wywołaniu rekurencyjnym dekompozycji centroidowej) oraz maksima na ścieżkach od niego do każdego z jego przodków (również zawartych w tym samym wywołaniu rekurencyjnym). Mając dwie listy liczb, chcemy policzyć dla każdej wartości, dla ilu różnych par występuje ona jako max(liczba z pierwszej listy, liczba z drugiej listy). Łatwo to zrobić sortując obie listy oraz używając gąsieniczki.

Następnie, dla każdego drzewa, policzmy listę maksimów na ścieżkach od korzenia tego drzewa do każdego wierzchołka w nim. Możemy teraz użyć techniki dziel i zwyciężaj na ciągu drzew. Znając przedział drzew, chcemy podzielić go na pół i rekurencyjnie policzyć wyniki w obu z nich. Brakującym składnikiem jest to, by mając dane rozłączne przedziały $[a_1, b_1]$ oraz $[a_2, b_2]$, policzyć wyniki dla ścieżek zaczynających się w drzewach z przedziału $[a_1, b_1]$, a kończących w przedziale $[a_2, b_2]$, oraz policzyć wyniki dla ścieżek zaczynających się w drzewach z przedziału $[a_2, b_2]$, a kończących w przedziale $[a_1, b_1]$. Oba podproblemy również możemy łatwo rozwiązać używając odpowiednich gąsieniczek, jednak pozwolimy sobie pominąć dokładne szczegóły techniczne.

Opisanej procedury używamy oczywiście na każdej grupie w całym grafie. Używając technik opisanych wyżej, otrzymamy finalną złożoność $\mathcal{O}(m \cdot \log^2(n))$, która nie powinna mieć problemów z uzyskaniem kompletu punktów. Starannie unikając w niej sortowania, lub wybierając nieco inne sposoby radzenia sobie z opisanym podzadaniem na meduzie, możemy skończyć również z algorytmem działającym w złożoności $\mathcal{O}(m \cdot \log(n))$.