O dwóch równoważnych problemach

Jakub RADOSZEWSKI *

Autor podaje przykład (a w zasadzie dwa przykłady) *redukcji* jednego problemu do drugiego. Redukcja jest techniką pozwalającą porównywać złożoność obliczeniową różnych zadań i dowodzić dolnego ograniczenia na tę złożoność. Klasycznym przykładem jest redukcja problemu sortowania do problemu znajdowania otoczki wypukłej na płaszczyźnie, która pokazuje, że nie można znaleźć otoczki n punktów szybciej niż w czasie $O(n \log n)$. O klasę złożoności wyżej plasują się wielomianowe redukcje dowodzące NP-zupełności trudnych problemów.

RMQ i LCA. RMQ i LCA to dwa klasyczne zagadnienia algorytmiczne, które nierzadko okazują się podproblemami w praktycznych zastosowaniach algorytmiki. Z pozoru nie mają one ze sobą nic wspólnego, jednakże pokażemy, że są w istocie równoważne, czyli że można efektywnie przekształcić rozwiązanie dowolnego z nich, tak by rozwiązywało ono drugie z tych zagadnień.
W zagadnieniu **RMQ** (ang. range minimum query) mamy dany n-elementowy ciąg liczbowy $a_1, \ldots, a_n$ i musimy umieć szybko odpowiadać na zapytania $RMQ(i, j)$ o minimalny element fragmentu tego ciągu $a_i, a_{i+1}, \ldots, a_{j-1}, a_j$, wyznaczonego przez i-ty i j-ty jego wyraz. Wyjaśnijmy to na przykładzie; dla ciągu $7, 3, 4, 1, 6, 8, 2, 5$ możliwe są między innymi następujące zapytania:

- $RMQ(6, 8)$ – odpowiedzią jest siódmy element ciągu równy 2,
- $RMQ(1, 8)$ – jest to równoważne z szukaniem minimum całego ciągu, czyli elementu 1.

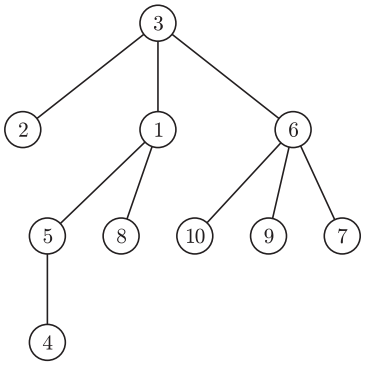
Problem **LCA** (ang. lowest common ancestor) dotyczy z kolei drzew ukorzenionych, których wierzchołki numerujemy liczbami od 1 do n.

Więcej o drzewach można przeczytać w książce Wilsona „Wprowadzenie do teorii grafów".

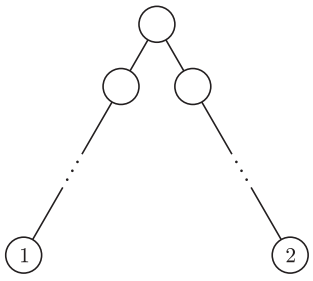
Dla danego drzewa chcemy odpowiadać na zapytania $LCA(i, j)$ o najniższego wspólnego przodka (czyli przodka najbardziej oddalonego od korzenia drzewa) danych wierzchołków i oraz j. Nasze drzewo możemy reprezentować, pamiętając w każdym węźle ojca tego węzła, a także listę wszystkich jego dzieci. Dla przykładowego drzewa (rys. 1) mamy następujące wyniki poniższych zapytań:

- $LCA(4, 8)$ to wierzchołek o numerze 1,
- $LCA(10, 2)$ odpowiada korzeniowi drzewa (numer 3),
- $LCA(6, 7) = 6$ – poszukiwanym przodkiem może być jeden z wierzchołków z zapytania.

W obu problemach poszukujemy struktury danych, którą będziemy w stanie efektywnie skonstruować i dzięki której odpowiadanie na żądane zapytania będzie mogło odbywać się (znacznie) szybciej niż w najbardziej siłowy sposób. W przypadku RMQ najprostszy algorytm polegałby na każdorazowym przeglądaniu zadanego fragmentu ciągu, a w LCA – na przechodzeniu z obydwu węzłów do korzenia drzewa (obsługa zapytań $RMQ(1, n)$ w pierwszym przypadku i $LCA(1, 2)$ dla drzewa z rysunku 2 wymagałyby wówczas wykonania po $O(n)$ operacji).



Rys. 1



Rys. 2

Od RMQ do LCA. Dla danego ciągu $a_1, \ldots, a_n$ pokażemy algorytm, za pomocą którego przekształcimy ten ciąg do drzewa takiego, że odpowiedzi na zapytania postaci $RMQ(i, j)$ dla ciągu będziemy mogli uzyskać za pomocą

Rozwiązanie zadania F 700.
Ponieważ Ziemia się obraca, więc na równiku występuje przyspieszenie odśrodkowe równe $\frac{v^2}{r}$, gdzie v to prędkość punktu równika (czyli 40 000 km na dobę), a r to promień Ziemi (czyli 6 356,8 km) – zmniejsza ono przyciąganie ziemskie. Po przerachowaniu okazuje się, że niedobór wagi wyniesie około 5 gramów. Zatem nawet złoto kupowane na równiku, a sprzedawane na biegunie, nie przyniosłoby zysku pokrywającego koszty transportu. Sytuację trochę poprawia uwzględnienie faktu, że promień równikowy jest większy od biegunowego – przy optymistycznym rachunku różnica może wynieść kolejne 5 gramów.

*student, Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski

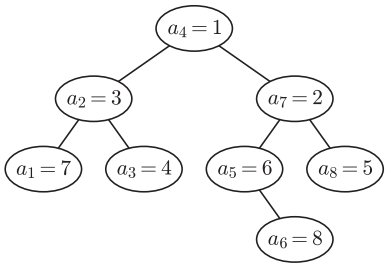
odpowiednich zapytań $LCA(i', j')$ dla i', j' będących numerami pewnych wierzchołków drzewa. Poszukiwaną strukturą jest tak zwane *drzewo kartezjańskie*, będące drzewem binarnym (każdy węzeł ma co najwyżej dwoje dzieci) o następujących właściwościach:

- węzły są ponumerowane liczbami od 1 do n, z czego i-ty węzeł drzewa odpowiada wyrazowi a_i ciągu,
- wyrazy ciągu odpowiadające potomkom węzła i są dla każdego i nie mniejsze od a_i, $(*)$
- numery potomków lewego dziecka węzła i są mniejsze od i, a numery potomków prawego dziecka i – większe od i (do potomków węzła wliczamy też sam węzeł). $(**)$

Dla zaznajomionych z klasycznymi strukturami danych oznacza to tyle, że drzewo kartezjańskie jest kopcem binarnym ze względu na wartości węzłów (wartości odpowiadających im elementów ciągu), a drzewem poszukiwań binarnych (BST) ze względu na ich numery. Drzewo kartezjańskie dla ciągu

$$7, 3, 4, 1, 6, 8, 2, 5$$

jest przedstawione na rysunku 3.



Rys. 3

Czemu to działa? Twierdzimy, że po skonstruowaniu drzewa kartezjańskiego dla ciągu $a_1, \ldots, a_n$ zapytanie $RMQ(i, j)$ można zasymulować przez $LCA(i, j)$ (zakładamy odtąd, że $i \leqslant j$). Sprawdźmy najpierw na przykładzie, że to rzeczywiście działa, przypominając sobie przykładowe zapytania RMQ dla naszego ciągu: rzeczywiście

$$RMQ(6, 8) = LCA(6, 8) = a_7 = 2 \text{ (rys. 4)},$$

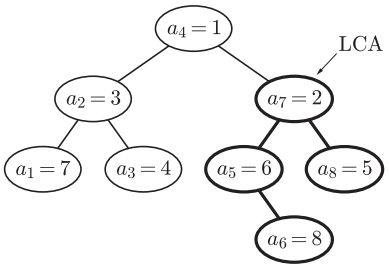
a także

$$RMQ(1, 8) = LCA(1, 8) = a_4 = 1 \text{ (rys. 5)}.$$

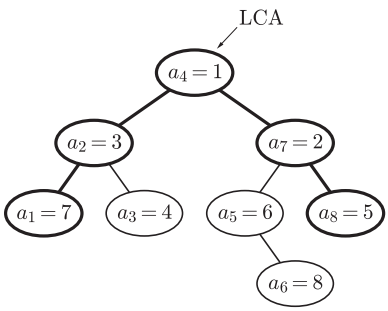
Czytelnik na pewno sam poradzi sobie z dowodem, że $LCA(i, j)$ jest elementem żądanego podciągu $a_i, \ldots, a_j$. Pozostaje więc pokazać, że węzły odpowiadające wszystkim elementom podciągu $a_i, \ldots, a_j$ są potomkami $LCA(i, j)$ – to wobec własności $(*)$ uzasadni, że $LCA(i, j)$ jest minimalną wartością w tym podciągu. Zauważmy więc, że wszystkie pozostałe wierzchołki drzewa albo leżą na ścieżce od $LCA(i, j)$ do korzenia drzewa, albo są potomkami tego typu wierzchołków, ale z „przeciwnej gałęzi" niż ta prowadząca z $LCA(i, j)$ (rys. 6). Jeżeli ścieżka prowadząca od $LCA(i, j)$ do korzenia wchodzi do danego węzła od strony lewego syna, to widzimy, że numery tego węzła i wszystkich potomków jego prawego syna są większe od j, czyli nie mieszczą się w zadanym fragmencie ciągu. W przypadku prawostronnym analogicznie wnioskujemy, że numery rozważanych wierzchołków są mniejsze niż i, co kończy dowód.

Jak szybko to działa? Trzeba jeszcze pokazać, jak szybko można skonstruować drzewo kartezjańskie dla danego ciągu. Budowanie drzewa będzie się odbywało metodą przyrostową, dokładniej, będziemy dokładać węzły odpowiadające kolejnym elementom ciągu. Nietrudno zauważyć, że w każdym kroku dokładany węzeł a_i będzie się znajdował na najbardziej prawostronnej ścieżce drzewa. Dlatego też odpowiednie miejsce dla a_i znajdziemy, poruszając się od najniższego węzła prawostronnej ścieżki drzewa aż do napotkania węzła z wartością mniejszą od a_i albo aż do wyczerpania węzłów na ścieżce. W pierwszym z tych przypadków prawym synem znalezionego węzła uczynimy węzeł zawierający a_i, a całą pozostałą część ścieżki podepniemy do a_i od lewej strony, w drugim zaś a_i uczynimy korzeniem drzewa, a całe dotychczasowe drzewo podepniemy do niego z lewej strony (przykład działania algorytmu na rys. 7).

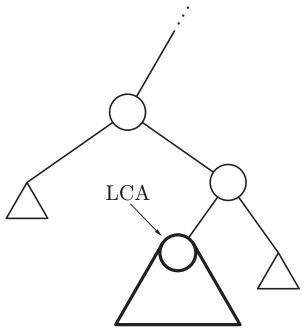
Na pierwszy rzut oka widzimy jedynie, że w każdym kroku algorytmu wykonywanych jest co najwyżej n kroków w górę wzdłuż prawostronnej ścieżki, a zatem jego złożoność czasową możemy oszacować przez $O(n^2)$. Jest to jednak bardzo zgrubne oszacowanie. Jeżeli w każdym kroku będziemy pamiętali najdalszy od korzenia węzeł ścieżki prawostronnej drzewa, to w takim przypadku łączna liczba kroków algorytmu może zostać



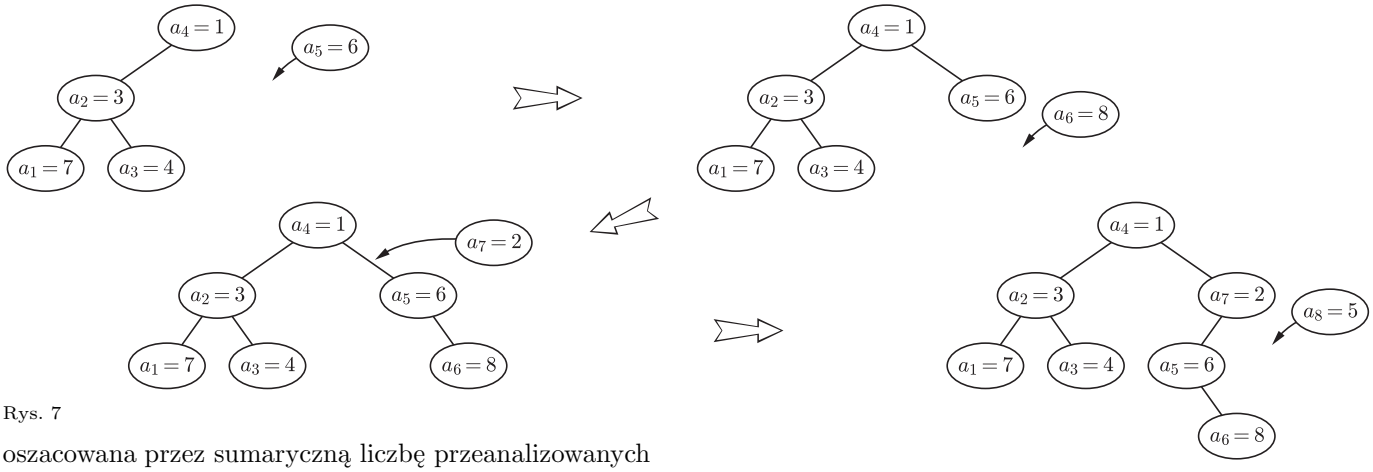
Rys. 4



Rys. 5



Rys. 6



Rys. 7

oszacowana przez sumaryczną liczbę przeanalizowanych wierzchołków ścieżek prawostronnych. Skoro jednak raz przejrzany wierzchołek na zawsze przestaje się już znajdować na ścieżce prawostronnej, a także każdy wierzchołek zostaje do niej dodany dokładnie raz, to *łączna* liczba kroków algorytmu może zostać oszacowana przez sumaryczną liczbę wierzchołków drzewa, czyli $O(n)$.

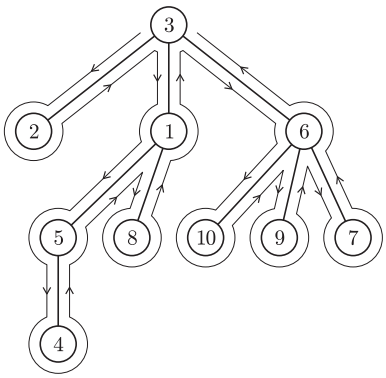
Ostatecznie udało nam się sprowadzić rozwiązywanie RMQ do zagadnienia LCA, tak że złożoność czasowa przejścia jest liniowa względem rozmiaru problemu (n), czyli praktycznie najszybciej, jak się nam mogło udać.

Od LCA do RMQ. Sprowadzenie w tę stronę jest nieco łatwiejsze niż w przeciwną. Tym razem naszym zadaniem jest przekształcenie danego drzewa w pewien ciąg liczbowy tak, aby zapytania postaci $LCA(i, j)$ dały się łatwo zapisać w postaci zapytań $RMQ(i', j')$ dla skonstruowanego ciągu. W pierwszym kroku wykonamy przeszukiwanie drzewa w głąb, rozpoczynając od korzenia i za każdym razem, kiedy znajdziemy się w jakimś wierzchołku, dopiszemy go na koniec listy wynikowej wraz z jego głębokością (korzeń drzewa ma głębokość 0).

Więcej o przeszukiwaniu w głąb można przeczytać w książce Cormen, Leiserson, Rivest, Stein „Wprowadzenie do algorytmów".

Dla drzewa z rysunku 1 wynikiem tego przeszukiwania będzie lista: $(3,0)$, $(2,1)$, $(3,0)$, $(1,1)$, $(5,2)$, $(4,3)$, $(5,2)$, $(1,1)$, $(8,2)$, $(1,1)$, $(3,0)$, $(6,1)$, $(10,2)$, $(6,1)$, $(9,2)$, $(6,1)$, $(7,2)$, $(6,1)$, $(3,0)$ (patrz rys. 8).

Na powyższej liście liczba wystąpień każdego wierzchołka jest równa liczbie jego dzieci powiększonej o jeden. Łączny rozmiar tej listy jest więc równy liczbie krawędzi drzewa (łączna liczba wszystkich dzieci) powiększonej o liczbę jego wierzchołków (suma jedynek), czyli $2n - 1$; podobnie całe przeszukiwanie ma złożoność czasową liniową względem rozmiaru drzewa. Okazuje się, że otrzymana lista jest poszukiwanym przez nas ciągiem! Zapytanie $LCA(i, j)$ obsłużymy, identyfikując jakiekolwiek wystąpienia par (i, g_i) i (j, g_j) (g_x to głębokość węzła x) na liście, zapytując o RMQ na przedziale przez nie wyznaczonym (minimalizujemy drugi element pary, czyli głębokość węzła) i jako poszukiwanego najniższego wspólnego przodka przyjmując pierwszy element znalezionej pary. Czytelnik łatwo sprawdzi, że każde z wcześniej omówionych przykładowych zapytań LCA zostanie dla naszego drzewa prawidłowo obsłużone, a wynikami będą odpowiednio pary: $(1,1)$, $(3,0)$ i $(6,1)$.



Rys. 8

Jeżeli dla każdego węzła drzewa będziemy pamiętać położenie któregokolwiek jego wystąpienia na liście, to koszt czasowy zapytania LCA będzie dokładnie taki sam, jak koszt odpowiadającego mu RMQ. Pozostało więc odpowiedzieć na pytanie, czy nasze sprowadzenie jest rzeczywiście poprawne. W tym celu przede wszystkim zauważmy, że LCA rozważanych węzłów będzie zawsze występował w podciągu o końcach wyznaczonych przez (i, g_i) i (j, g_j). Faktycznie, jeżeli tym LCA jest któryś z węzłów i, j, to stwierdzenie to jest oczywiste, a w przeciwnym przypadku ścieżki prowadzące od i oraz j do ich LCA wchodzą przez różne jego dzieci (inaczej nie byłby to najniższy wspólny przodek), czyli między zagłębieniami rekurencyjnymi algorytmu przeszukiwania dochodzącymi do i oraz do j wierzchołek odpowiadający LCA zostanie wypisany. Podobnie można pokazać, że żaden wierzchołek o mniejszej głębokości nie może się w tym przedziale znaleźć, co ostatecznie dowodzi, że poszukiwane LCA będzie znalezionym RMQ.

Wnioski. Pokazaliśmy algorytmy sprowadzające RMQ do LCA i z powrotem w złożoności czasowej $O(n)$, co pozwala nam stwierdzić, iż te dwa problemy są rzeczywiście równoważne. Pozostało nam teraz znalezienie efektywnego rozwiązania któregokolwiek z tych zagadnień. Tym jednak zajmiemy się w listopadowym numerze *Delty*.