1 Szukanie najniższego wspólnego przodka i minimum przedziału

1.1 Definicje problemów

W tym wykładzie zajmiemy się dwoma, na pozór nie mającymi ze sobą nic wspólnego, problemami:

Problem (Najniższy wspólny przodek)**.** *W (statycznym) problemie najniższego wspólnego przodka (ang. LCA, czyli Lowest Common Ancestor) chcemy dla danego ukorzenionego drzewa T zbudować strukturę danych odpowiadającą na zapytania o najniższego wspólnego przodka dowolnych dwóch wierzchołków T.*

Problem (Minimum przedziału)**.** *W (statycznym) problemie minimum przedziału (ang. RMQ, czyli Range Minimum Query) chcemy dla danej tablicy liczb A zbudować strukturę danych odpowiadającą na zapytania o pozycję na której występuje minimum w przedziale $A[i \dots j]$ dla dowolnych i, j.*

W obu przypadkach interesuje nas optymalizacja jednocześnie:

- rozmiaru struktury danych s,
- złożoności budowania struktury danych t,
- i czasu odpowiedzi na zapytanie q.

1.2 Zaskakujące fakty

Ćwiczenie 1. *Udowodnij, że problem RMQ sprowadza się do problemu LCA, tzn. jeśli LCA ma rozwiązanie o parametrach s, t, q, to RMQ też.*

Rozwiązanie. Dla danej tablicy A rozpatrzmy następujące binarne drzewo $T(A)$ (tzw. drzewo kartezjańskie). Korzeniem $T(A)$ jest dowolny najmniejszy element A, niech będzie nim $A[i]$. Lewe poddrzewo $T(A)$ jest drzewem kartezjańskim dla $A[1 \dots i-1]$, a prawe poddrzewo jest drzewem kartezjańskim dla $A[i+1 \dots n]$. Niech $A[i]$, $A[j]$ będą dowolnymi elementami A, a v_i, v_j odpowiadającymi im wierzchołkami drzewa $T(A)$. Łatwo zauważyć, że wierzchołek $\mathrm{LCA}(T(A), v_i, v_j)$ odpowiada elementowi $\mathrm{RMQ}(A, i, j)$.

Pozostaje pytanie, czy można zbudować drzewo $T(A)$ nie zwiększając złożoności t budowy struktury danych dla problemu LCA. Okazuje się, że tak. Można to bowiem zrobić w czasie liniowym przy pomocy algorytmu idącego od lewej do prawej. Tę konstrukcję widzieliśmy już przy okazji liniowego algorytmu sortowania liczb całkowitych, przy czym wtedy konstruowaliśmy nie drzewo kartezjańskie, a skompresowane drzewo trie. Sprawdź to!

Ćwiczenie 2. *Udowodnij, że problem LCA sprowadza się do problemu RMQ.*

Rozwiązanie. Rozważmy dowolny cykl Eulera C w T i wypiszmy wierzchołki w kolejności w jakiej się na nim pojawiają (z powtórzeniami) zaczynając i kończąc na korzeniu: $v_1, \ldots, v_{2n+1}$ (dlaczego wierzchołków jest $2n + 1$?). Niech $d(v)$ będzie głębokością wierzchołka v i niech A będzie tablicą taką, że $A[i] = d(v_i)$. Łatwo sprawdzić, że jeśli v i u są dowolnymi dwoma wierzchołkami, a v_i i v_j pewnymi ich wystąpieniami na cyklu Eulera, to RMQ(A, i, j) odpowiada ich LCA.

1.3 Proste rozwiązania

Tablicując wszystkie odpowiedzi można uzyskać $q = O(1)$ przy $s = O(n^2)$ i $t = O(n^2)$.

Ćwiczenie 3. *Pokaż, że można ten wynik poprawić do* $s = O(n \log n)$ *i* $t = O(n \log n)$.

Wskazówka. W problemie RMQ, dla dowolnego indeksu i, pamiętaj nie wszystkie RMQ(A, i, j), a tylko te postaci RMQ$(A, i, i + 2^k)$. Pokaż, że korzystając z takiej struktury danych można w czasie stałym odpowiadać na dowolne zapytania RMQ(A, i, j) oraz że można tę strukturę danych zainicjalizować w czasie $O(n \log n)$.

1.4 Rozwiązanie w pamięci $O(n)$

Rozwiązanie opisane powyżej jest często satysfakcjonujące, ale dla bardzo (, bardzo, bardzo) dużych danych chcielibyśmy mieć $s = O(n)$ i $t = O(n)$, i oczywiście $q = O(1)$. Okazuje się, że złożoność tę można osiągnąć.

Zauważmy, że sprowadzając problem LCA do problemu RMQ dostajemy bardzo specyficzne egzemplarze tego problemu. Takie mianowicie, w których kolejne liczby różnią się ± 1. Nazwijmy problem RMQ dla takich danych problemem RMQ $\pm$ 1. Jeśli uda nam się rozwiązać ten problem w pamięci $O(n)$ to nasze sprowadzenia dadzą nam analogiczne rozwiązania dla problemu LCA i ogólnego problemu RMQ. Robimy tu coś bardzo zabawnego: aby rozwiązać RMQ, sprowadzamy go do LCA, a potem z powrotem do szczególnego przypadku RMQ !!

Jak rozwiązać RMQ $\pm$ 1 w pamięci $O(n)$. Idea jest prosta. Dzielimy tablicę A na $O(n/\log n)$ kawałków rozmiaru $\frac{1}{2} \log n$. Aby odpowiedzieć na pytanie RMQ(A, i, j), znajdujemy:

- minimum pomiędzy $A[i]$, a końcem jego kawałka P_i,
- minimum pomiędzy $A[j]$, a początkiem jego kawałka P_j,
- minimum w kawałkach pomiędzy kawałkami P_i i P_j.

(osobno musimy rozpatrzyć przypadek, w którym i i j są w tym samym kawałku).

Minimum z kawałków pomiędzy P_i i P_j łatwo znaleźć. Wystarczy znaleźć minima w każdym z kawałków tablicy A, stworzyć nową tablicę M zawierającą wszystkie minima i zbudować dla niej strukturę danych o złożoności pamięciowej $O(n \log n)$.

Ćwiczenie 4. *Pokaż, że można zbudować w czasie $O(n)$ strukturę o złożoności pamięciowej $O(n)$ która odpowiada w czasie $O(1)$ na pytania o $RMQ(A, i, j)$ dla i, j leżących w jednym kawałku.*

Rozwiązanie. Jeśli w ciągu A każdy kolejny element różni się od poprzedniego o ± 1, to na przedziale długości $\frac{1}{2}\log n$ ciąg ten może się zachowywać na $\leq \sqrt{n}$ różnych sposobów – wartość pierwszego elementu nie ma wpływu na pozycję minimum, ważne jest tylko to, na których pozycjach ciąg rośnie, a na których maleje. Możemy stablicować odpowiedzi dla wszystkich możliwych rodzajów ciągów, oraz wszystkich możliwych pozycji początkowych i końcowych, razem $\sqrt{n}\log^2 n = o(n)$ możliwości.

1.5 Zastosowania

Na zakończenie zajmiemy się zastosowaniami omówionych w tym wykładzie wyników.

Rozważmy następujący problem:

Problem. *W problemie Document Retrieval (DR) mamy dany zbiór tekstów $T_1, \ldots, T_k$ i chcemy odpowiadać na pytania postaci: w których tekstach można znaleźć wzorzec P.*

Wydaje się, że problem jest prosty, wystarczy zbudować skompresowane drzewo sufiksowe dla wszystkich tekstów (drzewa sufiksowe zostały omówione w wykładzie Analiza Algorytmów). Pojawia się tu jednak pewien problem. Algorytm odpowiadający na zapytania ma złożoność $|P|$+(łączna liczba wystąpień P w tekstach), podczas gdy chcielibyśmy uzyskać złożoność postaci $|P|$+(liczba tekstów w których P występuje). Różnica między tymi złożonościami może być ogromna.

Okazuje się, że można osiągnąć pożądaną złożoność sprowadzając problem DR do problemu RMQ. Niech L będzie tablicą liści drzewa sufiksowego w kolejności leksykograficznej (tzn. od lewej do prawej w drzewie). Dla danego liścia l, niech $t(l)$ będzie numerem tekstu, któremu odpowiada l. Niech A będzie zdefiniowana następująco: jeśli $L[i]$ jest pierwszym liściem odpowiadającym tekstowi $t(L[i])$, to $A[i] = -1$, wpp $A[i]$ jest równy pozycji na której w tablicy L znajduje się poprzedni liść odpowiadający tekstowi $t(L[i])$.

Ćwiczenie 5. *Sprowadź problem DR dla tekstów $T_1, \ldots, T_k$ do problemu RMQ dla tablicy A opisanej powyżej.*

Wskazówka. Niech $L[i \ldots j]$ będzie przedziałem odpowiadającym P. Przedział ten może być duży, a nas interesuje znalezienie po jednym liściu dla każdego tekstu, w którym pojawia się P. Niech to będą pierwsze liście z lewej. Dla takich liści w tablicy A mamy -1, jeśli jest to pierwszy liść dla danego tekstu, wpp liczbę $< i$. Wystarczy więc znaleźć w tablicy $A[i \ldots j]$ wszystkie liczby $< i$.

Jak to zrobić?