Informatyczny kącik olimpijski (51): Nurkowanie

Tym razem zajmiemy się zadaniem *Nurkowanie* z Obozu Naukowo-Treningowego im. Antoniego Kreczmara w 2007 roku.

Zadanie brzmi tak. Grupa n nurków musi przedostać się przez podwodną szczelinę. Niestety, mają oni przy sobie tylko jedną wodoodporną latarkę. Szczelina jest wąska, więc jednocześnie mogą nią płynąć co najwyżej dwie osoby. Nurkowie postanowili powtarzać następujący schemat, dopóki będzie to konieczne: dwóch nurków przepłynie przez szczelinę, a następnie jeden z nich wróci, aby kolejni nurkowie mogli skorzystać z latarki. Każdy z nurków ma swoją maksymalną prędkość – jeśli dwóch nurków płynie jednocześnie, płyną z prędkością wolniejszego z nich. Ile minimalnie czasu zajmie im przeprawa przez szczelinę, jeśli m par spośród nich nie lubi się wzajemnie i nie może płynąć razem?

Załóżmy dla uproszczenia, że nurkowie są ponumerowani od 1 do n tak, że czasy, jakie potrzebują oni na przepłynięcie szczeliny, spełniają warunek $t_1 \leqslant t_2 \leqslant \ldots \leqslant t_n$. Sytuację daną w zadaniu opiszemy jako graf G, którego wierzchołki $\{1, 2, \ldots, n\}$ odpowiadają nurkom. Nurkowie i i j są połączeni w G krawędzią wtedy i tylko wtedy, gdy się lubią.

Zastanówmy się najpierw, kiedy problem nie ma rozwiązania. Każdy kolejny krok schematu jest równoważny wybraniu dwóch wierzchołków połączonych krawędzią w G, a następnie usunięciu z G jednego z nich, odpowiadającego nurkowi, który został po drugiej stronie szczeliny i już nigdy stamtąd nie wróci. Jeśli więc G na początku nie jest spójny, to na pewno przeprawa jest niewykonalna: po wykonaniu kursu liczba składowych nie maleje, więc gdy już nie będzie można wykonać żadnego kursu, pozostanie grupa co najmniej dwóch nurków, z których każdych dwóch się nie lubi.

Załóżmy więc, że G jest spójny. Okazuje się, że wtedy rozwiązanie istnieje i każdy możliwy scenariusz przeprawy możemy skutecznie modelować za pomocą drzewa ukorzenionego T. Ustanowimy nurka r, który uczestniczy w ostatniej turze przeprawy, korzeniem naszego drzewa. Jeśli natomiast i i j płyną razem, po czym i wraca, to j zostanie synem i w T. Jest jasne, że wykonywanie scenariusza przeprawy odpowiada ucinaniu liści naszego drzewa. Podobnie, mając dowolne ukorzenione drzewo rozpinające G, jesteśmy w stanie na jego podstawie skonstruować scenariusz przeprawy – wystarczy w dowolnej kolejności ucinać liście. Przypiszmy krawędziom drzewa T koszty czasowe odpowiadające kolejnym krokom. Jeśli i jest ojcem j w T, to krawędź między nimi ma koszt $\max(t_i, t_j) + t_i$, który, niestety, zależy nie tylko od wyboru nurków, ale także od skierowania krawędzi między nimi. Kosztem T niech będzie suma kosztów jego krawędzi pomniejszona o czas t_r potrzebny ostatniemu nurkowi na przeprawę (r nie musi wracać podczas ostatniego kroku).

Okazuje się, że przy tak ustalonych wagach da się uprościć problem, aby móc zapomnieć o skierowaniu krawędzi i wyróżnionym korzeniu. Obliczmy koszt drzewa T ((i, j) oznacza krawędź skierowaną od ojca i do syna j):

$$\begin{aligned}
&\sum_{(i,j)\in T} (\max(t_i, t_j) + t_i) - t_r = \\
&\quad = \sum_{(i,j)\in T} (\max(t_i, t_j) + t_i + t_j) - \sum_{(i,j)\in T} t_j - t_r = \\
&\quad = \sum_{(i,j)\in T} (\max(t_i, t_j) + t_i + t_j) - \sum_{i=1}^{n} t_i.
\end{aligned}$$

Druga suma jest stała, więc koszt zależy tylko od pierwszego członu wyrażenia! Wystarczy więc znaleźć minimalne drzewo rozpinające grafu G, w którym krawędź (i, j) ma przypisany koszt $\max(t_i, t_j) + t_i + t_j$.

Ponieważ na wejściu mamy dane *explicite* dopełnienie grafu G, możemy rozwiązać zadanie w czasie $O(n^2)$, uruchamiając np. *algorytm Prima*. Aby rozwiązać zadanie szybciej, zastosujemy podejście podobne jak w *algorytmie Borůvki*. Przypomnijmy, że w algorytmie tym utrzymujemy las rozpinający G i wykonujemy $O(\log n)$ faz; w każdej fazie działania dodajemy do konstruowanego rozwiązania najtańszą krawędź wychodzącą z każdej dotychczasowej składowej lasu (rozstrzygając remisy w dowolny uporządkowany sposób).

Pokażemy, jak zrealizować jedną fazę algorytmu Borůvki. Oznaczmy przez Z_i zbiór nurków, których nie lubi nurek i. Niech $A = \{a_1, a_2, \ldots, a_k\}$ będzie zbiorem nurków dowolnej składowej dotychczasowego rozwiązania (na początku każdy nurek stanowi osobną składową). Niech (c, d) będzie szukaną najtańszą krawędzią wychodzącą z A ($c \in A$, $d \notin A$). Jeśli $|A \cup Z_{a_1}| < n$, to niech b będzie nurkiem o najmniejszym numerze nienależącym do $A \cup Z_{a_1}$; krawędź (a_1, b) jest wtedy najtańszą krawędzią wychodzącą od nurka a_1 i kandydatem na najtańszą krawędź z A. Wartość b możemy znaleźć w czasie $O(|A| + |Z_{a_1}|)$ przy założeniu, że zbiory reprezentujemy jako uporządkowane listy.

Pokażemy, jak to jeszcze usprawnić. Niech

$$Y_i = \Big(\bigcap_{j=1}^{i} Z_{a_j}\Big) \setminus A.$$

Jeśli $i > 1$ i krawędź (a_i, b) jest najtańszą krawędzią wychodzącą z A, to $b \in Y_{i-1} \setminus Z_{a_i}$, w przeciwnym razie $b \notin Z_{a_j}$ dla pewnego $j < i$, a krawędź (a_j, b) byłaby tańsza. Wobec tego zarówno sprawdzenie każdego sensownego kandydata b dla nurka a_i, jak i obliczenie zbioru Y_i wykonujemy w czasie $O(|Y_{i-1}| + |Z_{a_i}|) = O(|Z_{a_{i-1}}| + |Z_{a_i}|)$. Sumując koszty dla każdego wierzchołka składowej i dla każdej składowej, otrzymujemy sumaryczny koszt jednej fazy algorytmu:

$$\sum_{A} O\Big(|A| + \sum_{i=1}^{k(A)} |Z_{a_i}|\Big) = O(n + m).$$

Złożoność czasowa całego rozwiązania wynosi zatem $O((n + m) \log n)$.

Adam KARCZMARZ
student, Wydział Matematyki, Informatyki i Mechaniki
Uniwersytetu Warszawskiego