Informatyczny kącik olimpijski (81): Kuglarz

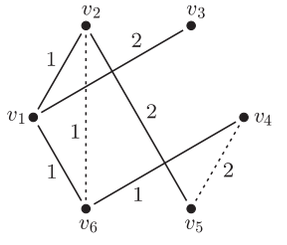
W tym miesiącu omówimy zadanie *Kuglarz* z pierwszej rundy Potyczek Algorytmicznych 2014. Tytułowy kuglarz zaprasza przechodniów do następującej gry. Na stoliku w rzędzie ustawił n kubków z numerami $1, 2, \ldots, n$, a zawczasu pod niektórymi schował kauczukowe kulki. Jeśli grający dokładnie odgadnie, które to kubki, to dostaje nagrodę. Kuglarz odpłatnie udziela grającemu podpowiedzi. Za c_{ij} bajtogroszy (dla $1 \leqslant i < j \leqslant n+1$) gotów jest zdradzić, jaka jest parzystość liczby kulek schowanych pod kubkami o kolejnych numerach $i, i+1, \ldots, j-1$. Znając ceny wszystkich możliwych podpowiedzi, należy wyznaczyć koszt zebrania informacji, które pozwolą określić z całą pewnością, pod którymi kubkami znajdują się kulki. Ściślej rzecz biorąc, należy znaleźć najmniejszą taką liczbę k, że istnieje strategia zadawania pytań, która niezależnie od odpowiedzi kuglarza pozwala na zlokalizowanie kulek za co najwyżej k bajtogroszy.

$i \setminus j$	2	3	4	5	6
1	1	2	3	4	1
2		4	3	2	1
3			3	4	5
4				2	1
5					5

Rys. 1. Przykładowa tabela kosztów dla $n = 5$ kubków. Koszt c_{ij} pytania $[i, j]$ znajduje się na przecięciu wiersza i z kolumną j.

Spróbujmy wyznaczyć rozwiązanie dla tabelki kosztów z rysunku 1. Naiwne zadanie pytań o jednokubkowe przedziały kosztowałoby $1 + 4 + 3 + 2 + 5 = 15$ bajtogroszy. Czy da się lepiej? Oznaczmy przez $[i, j]$ podpowiedź na temat parzystości liczby kulek pod kubkami o numerach $i, i+1, \ldots, j-1$. Zacznijmy od pytań o małych kosztach: pytanie $[1, 2]$ daje nam informację, czy pod pierwszym kubkiem znajduje się kulka, a pytanie $[1, 6]$ o parzystość liczby wszystkich kulek. Zadanie ich kosztuje $c_{12} + c_{16} = 2$ bajtogrosze. Następne pytanie o małym koszcie to $[2, 6]$, ale czy rzeczywiście opłaca się je zadawać? Nie, gdyż na podstawie poprzednich podpowiedzi, znamy parzystość kulek na pozycjach od 2 do 5.

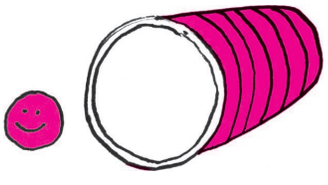
Tę obserwację można nieco uogólnić: na podstawie odpowiedzi na dwa dowolne pytania z trójki $[i, j]$, $[j, k]$ oraz $[i, k]$ jesteśmy w stanie wyznaczyć odpowiedź na trzecie z tych pytań. Tutaj następuje kluczowy pomysł: zbudujmy graf pusty G o $n+1$ wierzchołkach $v_1, v_2, \ldots, v_{n+1}$. Dla każdego zadanego przez nas pytania $[i, j]$, będziemy w tym grafie łączyć krawędzią wierzchołki v_i oraz v_j. Zachodzi teraz następujący fakt: jeśli wierzchołki v_i oraz v_j należą do tej samej spójnej składowej grafu, to albo zadaliśmy już kiedyś pytanie $[i, j]$, albo jesteśmy w stanie wyznaczyć na nie odpowiedź na podstawie dotychczasowych podpowiedzi kuglarza. Istotnie: jeśli wierzchołki te łączy ścieżka $v_i = v_{p_0}, v_{p_1}, \ldots, v_{p_\ell} = v_j$, to przez indukcję można pokazać, że możemy wyznaczyć odpowiedzi na kolejne pytania $[p_0, p_1], [p_0, p_2], \ldots, [p_0, p_\ell]$.



Rys. 2. Graf G po rozważeniu podpowiedzi o kosztach 1 i 2. Zadajemy pytania odpowiadające pogrubionym krawędziom.

Z powyższych rozważań wynikają następujące wnioski. Nie opłaca się prosić o podpowiedzi, które spowodują powstanie cyklu w grafie, zatem po zadaniu dokładnie n pytań graf G, który nam powstanie, będzie drzewem, a zdobyte informacje umożliwią zlokalizowanie wszystkich kulek (obecność kulki pod i-tym kubkiem to odpowiedź na pytanie $[i, i+1]$). Zauważmy, że musimy zadać n pytań, bo każda podpowiedź kuglarza daje nam co najwyżej 1 bit informacji, a musimy zgromadzić n bitów, żeby wyznaczyć lokalizację kulek. Ponadto, kolejność zadawania pytań nie ma znaczenia. Pozostaje kwestia, jak wybierać pytania. Rozważając je w kolejności od najmniejszych kosztów i pomijając pytania, na które znamy odpowiedź, możemy znaleźć strategię dla przykładu o koszcie 7 bajtogroszy (rys. 2).

Zauważmy, że postępując w taki sposób, obliczyliśmy nic innego, jak drzewo rozpinające o minimalnym koszcie dla nieskierowanego grafu pełnego, w którym każda para wierzchołków v_i i v_j jest połączona krawędzią o wadze c_{ij}. Nasza strategia zachłanna działa dokładnie tak jak algorytm Kruskala, służący do wyznaczania tego drzewa. Ponieważ, jak powiedzieliśmy wyżej, każda strategia zadawania pytań wyznacza nam drzewo rozpinające, zatem, aby znaleźć optymalną strategię, należy znaleźć drzewo rozpinające o minimalnej wadze.



Standardowo robi się to, korzystając właśnie z algorytmu Kruskala lub algorytmu Prima z kolejką priorytetową zrealizowaną za pomocą kopca binarnego. Dla grafu o n wierzchołkach i m krawędziach oba algorytmy działają w czasie $O(m \log n)$. W naszym przypadku mamy jednak do czynienia z grafem pełnym, w którym $m = \Omega(n^2)$, co daje czas $O(n^2 \log n)$. Można ten czas poprawić, zastępując kolejkę priorytetową w algorytmie Prima zwykłą tablicą, w której czas wyszukiwania będzie $O(n)$, ale czas aktualizacji krawędzi będzie stały. Dzięki temu złożoność rozwiązania zmniejszy się do $O(n^2)$.

Tomasz IDZIASZEK