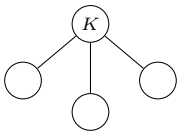
Informatyczny kącik olimpijski (122): *Zalesianie*

Tym razem omówimy zadanie *Zalesianie*, które pojawiło się na finale zawodów drużynowych *XII Olimpiady Informatycznej Gimnazjalistów*.

Zalesianie: *Danych jest n wierzchołków ponumerowanych od 1 do n. Ciąg $a = (a_1, a_2, \ldots, a_n)$ składa się z wartości przyporządkowanych kolejnym wierzchołkom (a_i to wartość przyporządkowana i-temu wierzchołkowi). Naszym zadaniem jest zbudować drzewo (spójny, acykliczny graf), przez ustalenie $n-1$ krawędzi między wierzchołkami oraz wybranie wierzchołka, który będzie korzeniem. Chcemy to zrobić w taki sposób, aby zminimalizować współczynnik niezadowolenia. Współczynnik niezadowolenia to maksymalna wartość wyrażenia $a_u \oplus a_v$, dla takich u i v, że u jest przodkiem v ($\oplus$ oznacza operację bitową xor).*

Zastanówmy się najpierw nad strukturą drzewa, które budujemy. Zauważmy, że korzeń jest przodkiem każdego innego wierzchołka w tym drzewie. Zatem, licząc współczynnik niezadowolenia, zawsze będziemy rozpatrywali każdą taką parę wierzchołków u i v, że u jest korzeniem drzewa. Okazuje się, że istnieje drzewo, w którym nie ma więcej par wierzchołków pozostających w relacji przodek-potomek. Wystarczy wybrać korzeń oraz połączyć go krawędziami ze wszystkimi pozostałymi wierzchołkami. Poniżej schemat takiego drzewa (K oznacza korzeń).



Ustaliliśmy już strukturę drzewa. Opiszemy teraz, w jaki sposób wybrać taki korzeń, który będzie minimalizował współczynnik niezadowolenia.

Rozwiązanie $O(n^2)$

Pierwszy, najbardziej intuicyjny pomysł, polega na naiwnym sprawdzeniu każdego z n sposobów ustalenia korzenia. Dla każdego wierzchołka $u \in \{1, 2, \ldots, n\}$ konstruujemy drzewo, którego korzeniem jest u oraz pozostałe wierzchołki są z nim połączone krawędzią. Następnie, dla każdego z tych drzew obliczamy (z definicji) współczynnik niezadowolenia. Spośród otrzymanych wyników wybieramy najmniejszy. Drzew mamy n (po jednym dla każdego kandydata na korzeń). Obliczenie współczynnika niezadowolenia dla jednego drzewa odbywa się w czasie $O(n)$. Zatem cały algorytm działa w czasie $O(n^2)$.

Rozwiązanie $O(n \cdot \log(\max_i a_i))$

Potraktujmy zapisy binarne elementów ciągu a jako słowa nad alfabetem $\{0, 1\}$. Niech $b' = (b'_1, b'_2, \ldots, b'_n)$ oznacza zapisy binarne kolejnych elementów ciągu a (b'_i oznacza zapis binarny a_i, który nie zawiera zer wiodących). Następnie „wyrównajmy" długości słów ciągu b'. Chcemy otrzymać słowa o takiej samej długości, równej długości najdłuższego słowa w ciągu b'. W tym celu wszystkie krótsze słowa należy poprzedzić odpowiednią liczbą zer wiodących. Kolejno otrzymane słowa nazwijmy $b = (b_1, b_2, \ldots, b_n)$.

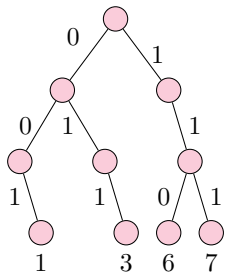
Podobnie jak w rozwiązaniu $O(n^2)$, dla każdego potencjalnego korzenia chcemy obliczyć współczynnik niezadowolenia. W istocie, dla każdego b_i chcemy znaleźć takie b_j, że $b_i \oplus b_j$ jest maksymalne. Na potrzeby tego artykułu zakładamy, że każdemu b_i jest przypisana wartość a_i oraz $b_i \oplus b_j$ ma wartość $a_i \oplus a_j$.

Weźmy $b_i = c_{m-1} \ldots c_1 c_0$, dla którego szukamy takiego $b_j = c'_{m-1} \ldots c'_1 c'_0$, że $b_i \oplus b_j$ jest maksymalne. Idealnie byłoby, gdyby istniało $b_j = \overline{c_{m-1}} \ldots \overline{c_1 c_0}$ ($\overline{c_i}$ oznacza negację c_i). Wtedy $b_i \oplus b_j = 2^m - 1$, czyli najlepszy możliwy wynik. Ta obserwacja daje nam pewną intuicję. Będziemy wyszukiwali b_j cyfra po cyfrze, w kolejności od najbardziej znaczących do najmniej znaczących. Załóżmy, że mamy już ustalony prefiks $c'_{m-1} c'_{m-2} \ldots c'_{k+1}$ słowa b_j. Zastanawiamy się teraz, jaką wartość może mieć c'_k. Oczywiście, gdyby $c'_k = \overline{c_k}$, wtedy wynik wzrósłby o 2^k. Zatem, jeśli $c'_{m-1} c'_{m-2} \ldots c'_{k+1} \overline{c_k}$ jest prefiksem jakiegoś słowa w ciągu b, to $c'_k = \overline{c_k}$, w przeciwnym przypadku $c'_k = c_k$. Prosty dowód pozostawiamy Czytelnikowi.

Pozostało nam jeszcze opisać, w jaki sposób dla danego słowa sprawdzać, czy jest ono prefiksem jakiegoś słowa w ciągu b. W tym celu zbudujemy *drzewo trie* nad słowami ciągu b. Wyszukiwanie b_j cyfra po cyfrze odpowiada wędrówce w drzewie trie od korzenia do liści. Załóżmy, że w danym kroku ustalamy wartość c'_k. Jeśli wierzchołek, w którym jesteśmy, ma dwóch synów, to poruszamy się krawędzią z etykietą $\overline{c_k}$. W przeciwnym przypadku nie mamy wyboru – idziemy krawędzią z etykietą c_k. Wędrówkę kończymy w liściu. Kolejno wybierane etykiety krawędzi tworzą szukane b_j.

Konstrukcja drzewa trie zajmuje czas $O(n \cdot \log(\max_i a_i))$. Wyznaczenie b_j dla danego b_i zajmuje czas $O(\log(\max_i a_i))$ (długość ścieżki od korzenia do liścia). Współczynnik niezadowolenia obliczamy dla każdego z n potencjalnych drzew, co w sumie daje $O(n \cdot \log(\max_i a_i))$ operacji. Zatem cały algorytm działa w czasie $O(n \cdot \log(\max_i a_i))$.

Rozważmy przykład. Niech $a = (1, 3, 6, 7)$. Wówczas $b' = (1, 11, 110, 111)$, $b = (001, 011, 110, 111)$, a drzewo trie wygląda następująco:



Bartosz ŁUKASIEWICZ