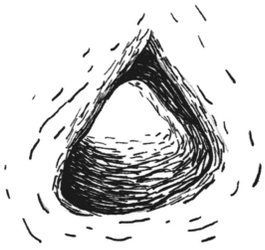
Informatyczny kącik olimpijski (50): Jaskinia



W tym kąciku omówimy zadanie *Jaskinia*, które pojawiło się w zeszłym roku na Akademickich Mistrzostwach Polski w Programowaniu Zespołowym. Tytułowa jaskinia składa się z n komnat połączonych $n-1$ korytarzami, które tworzą drzewo. Grotołazi chcą podzielić się na kilka grup, a następnie chcą przydzielić każdej grupie zbiór komnat, tak by każda komnata została przydzielona dokładnie jednej ekipie grotołazów oraz by każda grupa dostała tyle samo komnat do zbadania i była w stanie poruszać się pomiędzy przydzielonymi komnatami bez przechodzenia przez komnaty innych ekip. Na ile grup mogą podzielić się grotołazi?

Poniższa obserwacja daje warunek konieczny i wystarczający na istnienie podziału:

Drzewo o n wierzchołkach można podzielić na spójne kawałki rozmiaru k wtedy i tylko wtedy, gdy w tym drzewie znajduje się dokładnie $\frac{n}{k}-1$ krawędzi, które łączą poddrzewa o rozmiarach będących wielokrotnościami k.

Dowód jest prosty: jeśli podzieliliśmy drzewo na kawałki rozmiaru k zgodnie z warunkami zadania, to tych kawałków jest $\frac{n}{k}$. Krawędzi, które łączą wierzchołki należące do różnych kawałków, jest dokładnie $\frac{n}{k}-1$, a ponieważ poddrzewa połączone takimi krawędziami składają się z całych kawałków, wobec tego rozmiar każdego z tych poddrzew jest podzielny przez k. Krawędzie wewnątrz kawałków nie mają tej własności.

W drugą stronę: usuwamy kolejno wszystkie $\frac{n}{k}-1$ „dobrych” krawędzi. Po każdym usunięciu dostajemy zbiór niepustych kawałków o rozmiarach podzielnych przez k. Na końcu zostanie $\frac{n}{k}$ kawałków, czyli wszystkie muszą mieć rozmiar k.

Powyższa obserwacja pozwala nam na stworzenie następującego rozwiązania: ukorzeniamy drzewo w dowolnym wierzchołku i dla każdego $k \mid n$ przechodzimy drzewo od liści do korzenia, zliczając „dobre” krawędzie, tzn. takie, które łączą poddrzewa o rozmiarach będących wielokrotnościami k (rozmiary poddrzew obliczamy na bieżąco prostym programowaniem dynamicznym).

Złożoność czasowa tego rozwiązania to $O(n\sigma_0(n))$, gdzie $\sigma_0(n)$ oznacza liczbę dzielników n. Wśród liczb $n \leqslant 10^7$ możemy znaleźć takie, które mają dość dużo dzielników (rekord to $\sigma_0(8648640) = 448$), zatem powyższe rozwiązanie jest niezbyt szybkie.

Można to zrobić sprytniej, wykonując tylko jedno przeszukiwanie drzewa. Rozważmy bowiem krawędź, która łączy dwa poddrzewa o rozmiarach i oraz $n-i$, i zastanówmy się, dla jakich k będzie ona „dobrą” krawędzią. Ano, dla takich k, które dzielą zarówno i, jak i $n-i$, zatem dla wszystkich dzielników liczby $\mathrm{nwd}(i, n-i)$. Możemy zatem postąpić następująco. W tablicy t będziemy zliczać „dobre” krawędzie. W pierwszym kroku przechodzimy drzewo od liści do korzenia i dla każdej krawędzi zwiększamy licznik $t[\mathrm{nwd}(i, n-i)]$. Przejście drzewa wykonujemy w czasie $O(n \log n)$, gdyż dla każdej krawędzi obliczenie największego wspólnego dzielnika możemy wykonać w czasie $O(\log n)$, korzystając z algorytmu Euklidesa.

Następnie przeglądamy dzielniki n w kolejności rosnącej i dla każdego takiego dzielnika k zwiększamy o wartość $t[k]$ liczniki $t[j]$ dla $j \mid k$. Jeśli dla każdego k w poszukiwaniu jego dzielników przejrzymy wszystkie liczby mniejsze od k, to ten krok wykonamy w czasie $\sum_{k|n} O(k)$, czyli czasie proporcjonalnym do sumy dzielników n, czyli $\sigma_1(n) = O(n \log\log n)$.

Odpowiedzią są wszystkie liczby $\frac{n}{k}$, które spełniają $t[k] = \frac{n}{k} - 1$. Złożoność czasowa tego rozwiązania to $O(n \log n)$.

Przejście drzewa moglibyśmy zrealizować w czasie liniowym, gdybyśmy znali dla każdego i wartość $\mathrm{nwd}(i, n-i) = \mathrm{nwd}(i, n)$. Można te wartości obliczyć następującym algorytmem, który przypomina metodę sita Eratostenesa:

```
for i := 1 to n do
  if n mod i = 0 then
    j := i;
    while j ⩽ n do
      d[j] := i;
      j := j + i;
```

Po wykonaniu powyższego kodu będziemy mieli $d[j] = \mathrm{nwd}(j, n)$ dla każdego $j = 1, \ldots, n$. Dla każdego $i \mid n$ wewnętrzna pętla wykona się $\frac{n}{i}$ razy, zatem w całym algorytmie wykona się $\sigma_1(n)$ razy. Ostatecznie daje nam to algorytm o złożoności czasowej $O(n \log\log n)$.

Tomasz IDZIASZEK

Która liczba jest większa?

$$P = \binom{1000}{500} \quad \text{czy} \quad R = 2^{1000}$$