

Algorytmy strumieniowe

W dzisiejszym świecie cyfrowym mamy do czynienia z olbrzymią ilością danych, wielu Czytelników słyszało zapewne modne ostatnio hasło „Big Data”. I trzeba sobie z tym radzić, a problemy mogą pojawiać się w nieoczekiwanych miejscach. Przyzwyczajeni jesteśmy do myślenia, że programy mają pewne dane na wejściu i te dane są tam na stałe, program może je w dowolnym momencie przeczytać. Czasami jednak nie do końca przystaje to do rzeczywistości. Wyobraźmy sobie, że analizujemy kamerę online, która transmituje obraz 24 godziny na dobę i chcemy po miesiącu wydobyć pewne statystyki z tego, co nagrała. Możemy, oczywiście, zapisać całość na dysk i potem obliczyć to, co trzeba. Łatwo sobie jednak wyobrazić, że potrzeba na to wiele pamięci, którą niekoniecznie zamierzamy właśnie na to przeznaczać. Dobrze byłoby na bieżąco obliczać to, co chcemy, i nie zapisywać całego filmu – pamiętać tylko statystyki z przeszłości i uaktualniać je w czasie rzeczywistym. Podobne problemy spotykamy w wielu miejscach, gdy rozważamy duży strumień danych, przykładowo film lub dźwięk, ale również inne, jak ruch pakietów TCP/IP w sieci. Taka jest geneza powstania dziedziny algorytmów strumieniowych, czyli działających na strumieniu danych i obliczających na bieżąco pewne jego własności. Jej matematyczne podstawy są, moim zdaniem, wyjątkowo piękne.

Założymy, że nasz strumień danych to ciąg a_1, \dots, a_m dla pewnego dużego $m \in \mathbb{N}$, oraz że $a_i \in \{1, \dots, n\}$ dla każdego i . Niech m_i będzie liczbą wystąpień liczby i w całym strumieniu. Naszym celem będzie obliczenie pewnych własności strumienia w pamięci istotnie mniejszej niż $\mathcal{O}(m)$.

Na początek przyjrzyjmy się następującej zagadce. Rozważmy strumień, w którym pewna wartość występuje więcej niż na połowie miejsc. Czyli istnieje i takie, że $m_i > m/2$. Jak wtedy znaleźć i ? Nie jest jasne, jak to zrobić bez trzymania aktualnych wartości m_i dla wszystkich i podczas przeglądania strumienia. Okazuje się, że działa algorytm naprawdę banalny. W każdej chwili pamiętamy parę liczb. Pierwsza liczba to kandydat na odpowiedź, a druga liczba to coś jakby jego przewaga nad resztą. Inicjalizujemy parę jako $(0, 0)$. Powiedzmy, że po przeczytaniu a_1, \dots, a_j pamiętamy parę (i, c) . Są dwie możliwości. Jeśli $c = 0$, to po przeczytaniu a_{j+1} ustawiamy parę na $(a_{j+1}, 1)$. Jeśli natomiast $c > 0$, to rozważamy dwa przypadki. Jeżeli $a_{j+1} = i$, to zwiększamy przewagę, czyli ustawiamy $(i, c + 1)$. Jeśli natomiast $a_{j+1} \neq i$, to ustawiamy $(i, c - 1)$. Na końcu zwracamy pierwszy element z pary. Zauważmy jednak, że algorytm wcale nie trzyma prawdziwego dotychczasowego rekordzisty wraz z jego przewagą. Po przeczytaniu pierwszych sześciu elementów ciągu 1, 1, 1, 2, 2, 3 będzie trzymał parę $(2, 0)$, a więc po przeczytaniu ostatniego wyrazu parę $(3, 1)$. Dlaczego więc działa poprawnie? Przypomnijmy, że pewien element i występuje na więcej niż połowie miejsc. Niech wartość pary (j, c) będzie równa c , o ile $i = j$ oraz $-c$, o ile $i \neq j$. Zauważmy, że napotkanie elementu i zawsze zwiększa o jeden wartość pary, a innego niż i – zmniejsza ją lub zwiększa o jeden. Na początku wartość pary to 0. Zatem po przeczytaniu całego ciągu wartość pary będzie dodatnia, czyli pierwszy jej element to faktycznie i .

Ważnej informacji o własnościach strumienia dostarczają liczby F_k zdefiniowane jako $\sum_{i=1}^m m_i^k$. Dla $k = 0$ to liczba

różnych elementów, dla $k = 1$ to po prostu m , a dla $k = 2$ jest to ważna miara równomierności występowania wartości w ciągu. Bardzo ładny i elegancki jest przybliżony algorytm liczenia wartości F_2 w pamięci jedynie $\mathcal{O}(\log n + \log m)$. My najpierw pokażemy, jak to zrobić w pamięci $\mathcal{O}(n + \log m)$, co nie jest takie złe, bo to zwykle m jest olbrzymie, a n może być niewielkie. Algorytm używa losowości. Dla każdego $i \in \{1, \dots, n\}$ losujemy wartość $x_i \in \{-1, 1\}$ tak, że $\mathbb{P}(x_i = 1) = \mathbb{P}(x_i = -1) = 1/2$ oraz losowania są niezależne. Na początek ustalamy licznik c na 0 i zaczynamy przeglądać strumień. Gdy czytamy element i , to zwiększamy licznik o x_i . Na końcu zwracamy c^2 , twierząc, że jest to dobre przybliżenie F_2 . Algorytm prosty, ale dlaczego to w ogóle ma działać? Najpierw zauważmy, że każde i zwiększyło wartość c o x_i , czyli w sumie wszystkie i zwiększyły go o $x_i m_i$. A więc na końcu $c = \sum_{i=1}^n x_i m_i$. Obliczmy wartość oczekiwaną c^2 . Mamy $\mathbb{E}(c^2) = \mathbb{E}\left(\left(\sum_{i=1}^n x_i m_i\right)^2\right) = \mathbb{E}\left(\sum_{i,j=1}^n x_i x_j m_i m_j\right) = \sum_{i,j=1}^n m_i m_j \cdot \mathbb{E}(x_i x_j)$. Zauważmy, że $\mathbb{E}(x_i x_j) = 0$, o ile tylko $i \neq j$, bo x_i oraz x_j są niezależne. A zatem $\mathbb{E}(c^2) = \sum_{i=1}^n m_i^2 \cdot \mathbb{E}(x_i^2) = \sum_{i=1}^n m_i^2 \cdot 1 = F_2$. I ten fakt wydaje się najciekawszy. Wypadałoby jeszcze wykazać, że błąd tego obliczenia (czyli wariancja) jest nie za duży. My tutaj jedynie podamy na wiarę, że $\text{Var}(c^2) \leq F_2^2$. Ambitnych Czytelników zachęcamy do przerechowania. Błąd możemy łatwo zmniejszyć, uruchamiając 1000 takich samych algorytmów równocześnie, a na końcu uśredniając ich wyniki rezultat oznaczmy \bar{c}^2 . Wówczas wartość oczekiwana się nie zmieni, a wariancja zmaleje 1000 razy. Przypomnijmy nierówność Czebyszewa: $\mathbb{P}(|X - \mathbb{E}X| > t) \leq \text{Var}(X)/t^2$ dla dowolnego X . Dostajemy więc $\mathbb{P}(|\bar{c}^2 - F_2| > F_2/10) \leq 1/10$, czyli wynik wychodzi całkiem niezły.

Przypomnijmy, że nasz algorytm działa w pamięci $\mathcal{O}(n + \log m)$, a jednak n czasem może być duże. Liczba n bierze się stąd, że przez cały czas przeglądania strumienia pamiętamy n zmiennych x_i . Wydaje się to nieuniknione przy tej technice. Okazuje się jednak, że stosunkowo łatwo można zamienić n na $\log n$, używając bardzo pomysłowej metody. Przypomnijmy, jaka własność zmiennych x_i była nam potrzebna. Mianowicie chcieliśmy, by $\mathbb{E}(x_i x_j) = 0$ dla $i \neq j$, czyli by były one parami niezależne. Przy szacowaniu wariancji przydaje się ponadto niezależność czwórkami, dzięki której wiele ze składników postaci $\mathbb{E}(x_i x_j x_k x_\ell)$ znika. Można skonstruować obiekt wielkości $\mathcal{O}(\log n)$, z którego można wydobyć n zmiennych czwórkami niezależnych. My pokażemy jedynie, że da się tak zrobić dla zmiennych parami niezależnych. Zachęcamy Ambitnego Czytelnika do uogólnienia tego faktu. Dla uproszczenia załóżmy, że $n = 2^k - 1$, w tym przypadku mamy $k = \mathcal{O}(\log n)$. Wówczas pamiętamy k niezależnych zmiennych losowych y_1, \dots, y_k , takich, że $\mathbb{P}(y_i = 1) = \mathbb{P}(y_i = -1) = 1/2$. Dla dowolnego niepustego podzbioru $S \subseteq \{1, \dots, k\}$ definiujemy zmienną $x_S = \prod_{i \in S} y_i$. Łatwo sprawdzić, że x_S i x_T istotnie są niezależne dla dwóch różnych zbiorów S i T .

Algorytm obliczający przybliżoną wartość F_2 został opublikowany w 1999 roku przez Alona (patrz też str. 8), Matiasa i Szegedego jako chyba najbardziej zaskakujący wynik ich przełomowej pracy. Praca ta położyła fundamenty pod bardzo szybko rosnącą dziedzinę algorytmów strumieniowych, a autorzy otrzymali prestiżową nagrodę Gödla.

Wojciech CZERWIŃSKI