Ezoteryczne Kartki
Tabela potęgowa

Jakub Bachurski

wersja 1.0.2.4

1 Wstęp

Tym razem będziemy rozważać strukturę najczęściej wykorzystywaną do rozwiązywania statycznej wersji problemu RMQ – Range Minimum Query (przedziałowe zapytania o minimum w pewnym ciągu). Łatwo jest to rozwiązać drzewem przedziałowym z konstrukcją w czasie $O(n)$ i zapytaniami $O(\log n)$. Jednakże, tabela potęgowa (*sparse table*, czasem spotyka się także nazwę słownik podsłów bazowych) rozwiąże ten problem z czasem konstrukcji $O(n \log n)$ i zapytaniami $O(1)$.

	Drzewo przedziałowe	Tabela potęgowa
Konstrukcja	$O(n)$	$O(n \log n)$
Pamięć	$O(n)$	$O(n \log n)$
Zapytania	$O(\log n)$	$O(1)$

2 Idea

Algorytmicy lubią potęgi dwójki. Skupimy się na dwóch własnościach tychże:

- Podwojenie potęgi dwójki daje kolejną potęgę dwójki.

$$2^k + 2^k = 2^{k+1}$$

- W ogólności, $2^{\lfloor \log_2 n \rfloor}$ jest największą potęgą dwójki nieprzekraczającą n. Podwojenie takich potęg dwójki przekracza n:

$$2^{\lfloor \log_2 n \rfloor} + 2^{\lfloor \log_2 n \rfloor} > n$$

Nie są to zbyt skomplikowane własności, pozostaje nam tylko znaleźć dla nich zastosowanie. Weźmy nasz ciąg $a = (a_1, a_2, ..., a_n)$. Powiedzmy, że chcemy preprocessować minima dla wszystkich przedziałów długości potęg dwójki. Własność pierwsza pozwoli nam to szybko zrobić, a za pomocą własności drugiej będziemy odpowiadać na zapytania.

3 Konstrukcja

Tabela potęgowa utrzymuje tablice $T_k(i)$, przechowujące minimum na przedziale $\left[i, i+2^k\right)$ dla każdego $0 \leqslant k \leqslant \log_2 n$ (większe k przechowywałyby przedziały dłuższe niż ciąg). Spamiętywane są wszystkie indeksy $0 \leqslant i \leqslant n - 2^k$ (inne ponownie wykraczałyby poza ciąg). Oczywistym jest, że $T_0(i) = a_i$. Natomiast z pomocą własności pierwszej z radością stwierdzamy, że:

$$\min(T_k(i), T_k(i+2^k)) = T_{k+1}(i)$$

A zatem, aby obliczyć wszystkie $T_k(i)$, wystarczy iterować się od najmniejszych potęg dwójki, i odwoływać się do wcześniej policzonej tablicy. W podobny sposób robimy to w drzewie przedziałowym, lecz tym razem robimy to dla wszystkich indeksów, a nie tylko dla ustalonych.

4 Zapytania

Otrzymujemy zapytanie o minimum na przedziale $[l, r)$ – musimy policzyć:

$$\min(a_l, a_{l+1}, ..., a_{r-2}, a_{r-1})$$

Oznaczmy jego długość jako $d = r - l$ i przyjrzyjmy się własności drugiej. Wynika z niej, że aby *pokryć* cały przedział $[l, r)$ wystarczą nam dwa przedziały wielkości $2^{\lfloor \log_2 d \rfloor}$. Skąd pokrywanie? Stąd, że możemy wykorzystać przydatną własność minimum – idempotentność:

$$\min(\min(a, b), \min(b, c)) = \min(a, b, c)$$

Zatem możemy wybrać pewne przedziały, na których znamy minimum, i z tych wartości wziąć minimum. W ten sposób otrzymamy minimum na zakresie będącym sumą wszystkich przedziałów, które właśnie wzięliśmy. W tym wypadku chcemy otrzymać minimum na $[l, r)$.

Zastanówmy się, jaką wybrać strategię pokrycia przedziału. Zauważmy, że nie możemy wybrać przedziału niedotykającego jednego z końców (l i $r-1$), bo w ten sposób powstałe na końcach przedziały musiałyby zostać jeszcze raz zostać zapełnione. Rozważmy to na przykładzie podciągu $(a_3, a_4, ..., a_9)$:

$$a_3 \quad a_4 \quad a_5 \quad a_6 \quad a_7 \quad a_8 \quad a_9$$

Mamy $l = 3$, $r = 10$, $d = r - l = 7$, zatem $2^{\lfloor \log_2 d \rfloor} = 4$. Spróbujmy wykorzystać przedział $[4, 8)$.

$$a_3 \quad \boldsymbol{a_4} \quad \boldsymbol{a_5} \quad \boldsymbol{a_6} \quad \boldsymbol{a_7} \quad a_8 \quad a_9$$

Widać, że zgubiliśmy (a_3) oraz (a_8, a_9). Pozostaje jedyny słuszny wniosek: wykorzystać przedziały zaczepione na **początku** oraz na końcu.

$$\boldsymbol{a_3} \quad \boldsymbol{a_4} \quad \boldsymbol{a_5} \quad \underline{\boldsymbol{a_6}} \quad \underline{a_7} \quad \underline{a_8} \quad \underline{a_9}$$

A zatem pokazaliśmy, że wystarczające jest wykorzystać dwa przedziały. Oznaczmy $e = \lfloor \log_2 d \rfloor$. Wystarczy wykorzystać $T_e(l)$ oraz $T_e(r - 2^e)$.

5 Podsumowanie

5.1 Wybór operacji

Wykonywaną operacją nie musi być minimum. Możemy wykonywać dowolną operację $\circ$, o ile spełnia ona następujące własności:

- $a \circ a = a$ – po pierwsze, idempotentność. Wielokrotna aplikacja tego samego argumentu nie zmienia wyniku. Jest potrzebna, bo przedziały wykorzystywane przy odpowiadaniu na zapytanie będą się pokrywały.
- $a \circ b = b \circ a$ – przemienność.
- $(a \circ b) \circ c = a \circ (b \circ c)$ – łączność.

Przykłady takich operacji (idempotentnych):

- Minimum
- Maksimum
- Największy wspólny dzielnik (gcd), tudzież najmniejsza wspólna wielokrotność (lcm)
- `or` bitowy
- `and` bitowy

5.2 Dynamiczność

Niestety, ponieważ duża liczbie komórek tablic T_k zależy od danego indeksu, nie da się efektywnie modyfikować tabeli potęgowej.

5.3 Implementacja

- Aby liczyć podłogę logarytmu dwójkowego, możemy go preprocesować w oddzielnej tablicy, lecz możemy także wykorzystać funkcję `std::__lg`.
- Lepiej uważać na zużycie pamięci – sparse table zbudowane na ciągu 10^6 liczb 32-bitowych zużyje około 75MB.

5.4 Wielowymiarowa tabela potęgowa

Można napisać dwuwymiarową tabelę potęgową, działającą na prostokątnej tablicy $n \times m$, z konstrukcją w czasie $O(nm \log n \log m)$. W tym wypadku kluczem jest procesowanie wszystkich prostokątów wielkości $2^k \times 2^l$. Przy zapytaniach o prostokąt należy wykorzystywać prostokąty zaczepione w rogach prostokąta z zapytania.

6 *Zadanka*

Podobnie jak w przypadku innych struktur danych, kluczem do wykorzystania tabeli potęgowej jest sprowadzenie problemu do czegoś, co umie ona rozwiązać: zapytań o funkcję idempotentną na przedziałach statycznego ciągu. Warto zauważyć, że można z nią np. wyszukiwać binarnie.

Tabela potęgowa ma inną ciekawą cechę: jeżeli uda nam się w czas rozwiązania wepchnąć jej konstrukcję, to może nam znacząco uprościć implementację. Na przykład, gdy robimy algorytm z gąsienicą, zamiast utrzymywać dodatkowe zmienne utrzymujące wynik pewnej operacji na przedziale, możemy po prostu odwoływać się do struktury. Niestety, takie uproszczenie ma swoją cenę – narzut czasowy i zużycie pamięci.

- [SPOJ] Range Minimum Query – na przetestowanie struktury
- [XXVI OI] Klubowicze 2
- [Codeforces] CGCDSSQ

Spora lista zadań znajduje się tutaj: `https://cp-algorithms.com/data_structures/sparse-table.html#toc-tgt-5`.