Ezoteryczne Kartki
Drzewo falkowe

Jakub Bachurski

wersja 1.0.3

1 Na wstępie

Będziemy rozważać strukturę danych działającą na ciągu $s = (s_1, s_2, ..., s_n)$. Elementy ciągu należą do pewnego alfabetu wielkości σ (stąd będziemy wygodnie nazywać je znakami), a znaki w tym alfabecie mają zdefiniowany pewien porządek.

Za pomocą tej struktury będziemy odpowiadać na takie zapytania na spójnych podciągach $z = (s_l, s_{l+1}, ..., s_{r-1}, s_r)$:

- Zliczanie znaków, dla których $z_i \geqslant c$, dla wybranego c.
- *Quantile* (kwantyl) o parametrze k – szukanie k-tego najmniejszego znaku w z. Dokładniej, szukamy z'_k, gdzie z' to posortowany ciąg z. Na przykład, gdy podciąg $z = (5, 4, 2, 4, 3, 2)$ i $k = 4$ (indeksując od 0), poszukiwany kwantyl wynosi 4, bo $z' = (2, 2, 3, 4, \mathbf{4}, 5)$.

Możemy także wykonywać proste modyfikacje ciągu: `push_back`, `pop_back`, zamiana sąsiednich elementów. Wszystkie te operacje są obsługiwane w $O(\log \sigma)$.

Wzbogacając strukturę i zgadzając się na zwiększoną złożoność $O(\log \sigma \log n)$ (także w zapytaniach) można wykonywać dodatkowe modyfikacje: `push_front`, `pop_front`, wyłączanie i włączanie (`toggle`) pewnego indeksu (gdy indeks jest wyłączony, jest ignorowany podczas odpowiadania na zapytania).

2 Materiały

Bardzo przydatna jest praca z konferencji IOIa:

`https://users.dcc.uchile.cl/~jperez/papers/ioiconf16.pdf`

Na innych pracach z konferencji można znaleźć też trochę innych ciekawych rzeczy.

Natomiast w tym blogu są ładne ilustracje, niestety mało jest o zastosowaniach: `https://alexbowe.com/wavelet-trees/`.

3 Idea

Nazwa sugeruje, że będziemy chcieli zbudować drzewo. Zwykle, rozkładaliśmy ciągi na kawałki względem indeksów – np. w drzewie przedziałowym. Tym razem, rozłożymy drzewo po alfabecie. Od tego zabiegu pochodzi nazwa struktury – drzewo falkowe[1]

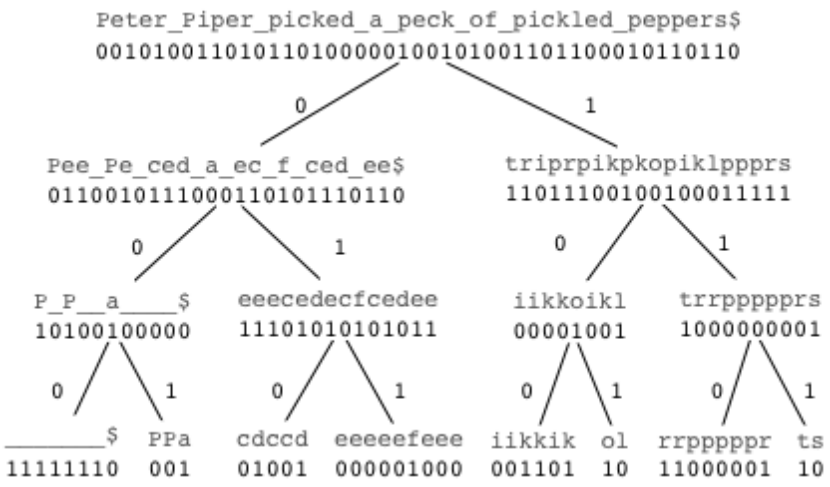
Weźmy nasz ciąg s. Znaki, które się w nim znajdują, należą do przedziału $[a, b)$ (na razie załóżmy, że alfabet jest duży). Wybierzmy teraz pewną wartość podziału c i skonstruujmy bitmapę B w następujący sposób[2]:

$$B_i = [s_i \geqslant c]$$

Możemy tak zrobić, bo wcześniej założyliśmy, że na znakach zdefiniowaliśmy porządek.

Co nam to daje? Możemy teraz trywialnie odpowiadać na opisane wcześniej zapytania, ale jedynie rozróżniając znaki $< c$ oraz $\geqslant c$. A co z pozostałymi przypadkami? Ponieważ podzieliśmy alfabet na fragmenty $[a, c)$ oraz $[c, b)$, po prostu zbudujmy dwa poddrzewa rekurencyjnie na znakach które należą do odpowiednich fragmentów alfabetu. Proces ten możemy powtarzać, dopóki alfabet rozpatrywany przez poddrzewo ma więcej niż dwa znaki.

Za pomocą tej konstrukcji otrzymamy drzewo[3]:



Ponieważ wysokość drzewa wynika z wybranego punktu podziału, a tworzymy dwa fragmenty, najlepiej dzielić alfabet na połowę – $c = \lfloor \frac{a+b}{2} \rfloor$. W ten sposób wysokość jest $O(\log \sigma)$.

[1] Falka (ang. *wavelet*) na Wikipedii: `https://pl.wikipedia.org/wiki/Falki`. Nazwa drzewa pochodzi od analogii do podobnego procesu wykorzystywanego w rozkładzie sygnału radiowego przy jego przetwarzaniu.

[2] Notacja nawiasów Iversona jest zdefiniowana jako $[p] = 1$ jeżeli p jest prawdziwe, w przeciwnym przypadku $[p] = 0$.

[3] Ilustracja pochodzi z `https://alexbowe.com/wavelet-trees/`

4 Drzewo

Podsumowując, w drzewie chcemy utrzymywać:

- Rozpatrywany zakres alfabetu – a, b
- Punkt podziału zakresu – c
- Bitmapę podziału – B
- Wskaźniki na poddrzewa (chyba, że mamy do czynienia z liściem) – L, R

5 Zapytania

5.1 Kwantyl

Rozpatrujemy pewne drzewo T i chcemy rozwiązać kwantyl:

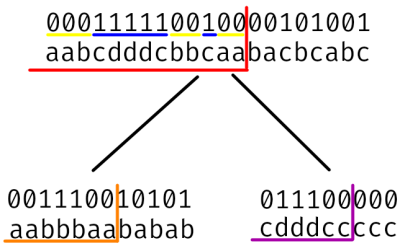
$$\texttt{quantile}(T, l, r, k)$$

Parametr k rozważamy w indeksowaniu od 0. Dla takiego zapytania chcemy podać k-ty co do wielkości znak na przedziale $[l, r)$.

Rozpatrzmy przypadki dla małych alfabetów. Jeżeli $b - a = 1$ (tylko jeden znak), to rozwiązaniem jest po prostu a. Jeżeli $b - a = 2$, to musimy zliczyć, czy na tym przedziale w B jest co najmniej k zer – jeżeli tak, to wynikiem jest a, w przeciwnym wypadku b. Przypadki dla większych alfabetów rozwiążemy analogicznie, przy czym będziemy musieli, zamiast od razu zwrócić wynik, odwołać się do lewego lub prawego poddrzewa (zamiast zwrócić, odpowiednio, zero i jedynkę).

Pojawiła się potrzeba zliczania na przedziale zer i jedynek. Możemy to osiągnąć zwykłymi sumami prefiksowymi po bitmapie (aczkolwiek są inne sposoby, dzięki którym możemy znacznie zaoszczędzić na zużyciu pamięci – lecz nie są one nam tak potrzebne). Operację zliczenia wśród pierwszych p wartości bitmapy B zer oznaczamy jako $\texttt{rank}_0(p)$, analogicznie piszemy $\texttt{rank}_1(p)$. Oczywiście, zachodzi równość $\texttt{rank}_1(p) = p - \texttt{rank}_0(p)$.

Zauważmy jeszcze, że schodząc do niższego poddrzewa, nasz przedział może się przesunąć. Schodząc do lewego poddrzewa, znikają znaki będące dotychczasowymi jedynkami. Taka operacja przemapowania indeksu sprowadza się to do zamiany danego indeksu i na $\texttt{rank}_0(i)$ (lub $\texttt{rank}_1(i)$). Łatwiej jest to zrozumieć, wpatrując się w obrazek:



Biorąc to wszystko pod uwagę, umiemy zatem opisać jako funkcję rekurencyjną (najpierw oznaczając $\texttt{rank}_0(r) - \texttt{rank}_0(l)$ jako t):

$$\texttt{quantile}(T, l, r, k) = \begin{cases} a, & b - a = 1 \\ \texttt{quantile}(L, \texttt{rank}_0(l), \texttt{rank}_0(r), k), & t > k \\ \texttt{quantile}(R, \texttt{rank}_1(l), \texttt{rank}_1(r), k - t), & t \leqslant k \end{cases}$$

W ten sposób po $O(\log \sigma)$ iteracjach sprowadzimy zapytanie do trywialnego przypadku z jednoznakowym alfabetem.

5.2 Zliczanie znaków większych

Warto zauważyć, że zliczanie na przedziale znaków $\geqslant d$ jest równoważne ze zliczaniem znaków $> d, \leqslant d, < d$ czy, najogólniej, $\in [e, f)$. Dodatkowo, zliczanie na przedziale jest równoważne ze zliczaniem na pewnym prefiksie. Opiszmy zatem $\texttt{count}_{\geqslant}(T, p, d)$ jako liczbę wartości $\geqslant d$ na prefiksie długości p słowa opisanego przez drzewo falkowe T.

Po chwili zastanowienia nad postacią rekurencyjną otrzymamy:

$$\texttt{count}_{\geqslant}(T, p, d) = \begin{cases} 0, & b \leqslant d \\ p, & a \geqslant d \\ \texttt{count}_{\geqslant}(L, \texttt{rank}_0(p), d) + \texttt{count}_{\geqslant}(R, \texttt{rank}_1(p), d), & a < d < b \end{cases}$$

Aby poznać złożoność, trzeba przyjrzeć się, do jakich przedziałów trafia d. Jeżeli d jest gdzieś w środku przedziału alfabetu rozważanego przez T, to wywołanie zejdzie do obu poddrzew. Jednakże, tylko w jednym z tych poddrzew d będzie należało do przedziału alfabetu – $[a, c)$ lub $[c, b)$ – w drugim z nich natychmiast zwrócimy wynik. Zatem wykonamy $O(\log \sigma)$ wywołań.

6 Modyfikacje

Rozważmy jeszcze przez chwilę modyfikacje. Głównym problemem dodawania w dowolnym miejscu ciągu czy usuwania z niego jest to, że wtedy problematyczna jest struktura obsługująca dodawanie w dowolnym miejscu *oraz* odpowiednią modyfikację sum prefiksowych. Zatem ograniczamy się do charakterystycznych miejsc ciągu – początku oraz końca.

Dodawanie na koniec możemy łatwo obsłużyć dodając odpowiedni bit na końcu kazdej z bitmap, do której trafiłby znak, który dodajemy. Sumy prefiksowe także łatwo poprawić. Wykonamy to w $O(\log \sigma)$. Ważna jest tutaj obserwacja, że konkretny indeks występuje w $O(\log \sigma)$ poddrzewach.

Tworząc bardziej skomplikowane modyfikacje będziemy musieli umieć szybko poprawiać sumy prefiksowe. Możemy do tego wykorzystać inną strukturę, w której będziemy trzymać B – drzewo Fenwicka.

Inne modyfikacje wymienione są we wstępie oraz w pracy z konferencji IOI.

7 Implementacja

7.1 Partycja

Podczas konstrukcji, do lewego oraz prawego poddrzewa chcemy przekazać tylko elementy $\leqslant c$ oraz $> c$. Aby tego dokonać, możemy wykorzystać funkcję standardową `std::stable_partition` (`stable` jest potrzebne, aby zachować kolejność elementów taką samą, jak w oryginalnym ciągu).

7.2 Wielkość alfabetu

Jeżeli $\sigma > n$, to z pewnością w ciągu nie występują niektóre znaki z alfabetu. Możemy wtedy je odpowiednio przeskalować, zachowując porządek między nimi. W ten sposób sprowadzamy złożoność do $O(\log n) < O(\log \sigma)$.

8 *Zadanka*

Większość zadań na wavelet tree (ahem, drzewo falkowe) polega na zaimplementowaniu pewnych jego operacji, lecz czasami możemy je przypadkiem spotkać w dziczy. Ponieważ jest ono mało popularne, może stanowić bardziej oczywiste, niekoniecznie przewidziane przez autorów rozwiązanie.

Kluczem do wykorzystania tej struktury jest spostrzeżenie jednego z obsługiwanych zapytań, np: „zlicz na przedziale znaki, które są większe bądź równe x", bądź „znajdź na przedziale medianę".

Te trzy zadania zostały stworzone przez autorów pracy z konferencji IOI jako eksperyment społeczny, mający zbadać, jak użytkownicy SPOJ-a rozwiążą zadania, które najłatwiej jest rozwiązać wavelet tree. Żaden z submitujących nie napisał drzewa falkowego, a AC otrzymali głównie najlepsi programiści. Zwykle jako alternatywę wykorzystywało się *mergesort tree*, pierwiastki, czy trwałe drzewa przedziałowe.

- [SPOJ] I Love Kd-Trees – Na prefiksie długości i znajdź k-ty co do wielkości element x, a następnie znajdź jego l-te wystąpienie w całym ciągu.
- [SPOJ] I Love Kd-Trees II – Na przedziale podaj ile jest wartości k, przy czym dodatkowo występują operacje `toggle` na indeksie (wyłącz/włącz).
- [SPOJ] I Love Kd-Trees III – Zapytania jak w pierwszym, ale dodatkowo należy umieć zamieniać elementy na sąsiednich pozycjach.

A to zadanie ma inne wzorcowe rozwiązanie, ale można je przekształcić na zapytania na drzewie falkowym:

- [ILOCAMP] (Treść) Ranking

Inne ciekawe zadanie:

- [Kattis] Palindromes