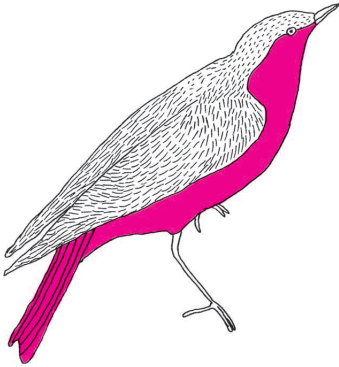


Złośliwy problem $(MAX, +)$ i kubełkowe struktury danych

Wojciech ŚMIETANKA*



Dany jest ciąg złożony z n nieujemnych liczb całkowitych $c = (c_1, c_2, \dots, c_n)$. Na ciągu c chcemy wykonywać operacje dwóch rodzajów:

- $\text{update}(i, j, w)$ – modyfikuje wartości wyrazów ciągu o indeksach z przedziału od i do j w sposób zależny od parametru w (dodatniej liczby całkowitej);
- $\text{query}(i, j)$ – podaje *zintegrowaną* wartość dla podciągu o indeksach od i do j .

W obydwu przypadkach zakładamy, że $1 \leq i \leq j \leq n$.

Będziemy rozważali dwie różne operacje update , które oznaczymy symbolicznie przez $+$ i MAX . W operacji $+$ należy do każdego wyrazu podciągu c_i, \dots, c_j dodać liczbę w . Z kolei operacja MAX polega na zmianie wartości wyrazu c_l ($i \leq l \leq j$) na $\max(c_l, w)$ – innymi słowy, jeśli wyraz miał wcześniej wartość nie mniejszą niż w , to jej nie zmienia, a jeśli jego wartość była mniejsza od w , to jego nową wartością jest w .

Podobnie definiujemy dwie operacje query : $+$ i MAX . Na zapytanie $+$ odpowiadamy, podając sumę wyrazów z podciągu c_i, \dots, c_j . Odpowiedzią na zapytanie typu MAX jest natomiast wartość największego wyrazu w rozważanym podciągu.

Biorąc pod uwagę wszystkie kombinacje poszczególnych operacji, otrzymujemy cztery różne warianty problemu. Każdy taki wariant oznaczamy za pomocą pary, której pierwszy wyraz opisuje rodzaj operacji modyfikacji, a drugi określa zapytanie. Mamy zatem następujące warianty: $(+, +)$, $(+, MAX)$, $(MAX, +)$ oraz (MAX, MAX) .

Okazuje się, że nasz problem ma kilka ciekawych zastosowań. Wariant $(+, MAX)$ można wykorzystać do implementacji systemu obsługi rezerwacji miejsc w pociągu na trasie łączącej $n + 1$ stacji. W pociągu znajduje się ustalona liczba miejsc siedzących. Każda rezerwacja dotyczy konkretnej liczby pasażerów i wskazuje numery dwóch stacji, na których pasażerowie zamierzają wsiąść i wysiąść. Naszym celem jest przyjmować kolejno wszystkie rezerwacje, które nie powodują przepełnienia pociągu.

Oznaczmy przez c_l aktualną liczbę pasażerów, którzy będą w pociągu na trasie pomiędzy stacjami l oraz $l + 1$. Rezerwację na podróż ze stacji i do stacji j dla w pasażerów możemy przyjąć, jeśli wartość $\text{query}(i, j - 1) + w$ jest nie większa niż liczba miejsc w pociągu. Po przyjęciu rezerwacji wykonujemy $\text{update}(i, j - 1, w)$.

Inaczej można na to spojrzeć jak na problem kontroli przesyłania danych między serwerami połączonymi jedną linią danych o określonej przepustowości albo problem obsługi pobierania danych z Internetu przez użytkowników sieci lokalnej w zadanych przedziałach czasowych (sieć ma ograniczony transfer).

Wariant (MAX, MAX) ma natomiast elegancką interpretację kombinatoryczną powiązaną z popularną grą Tetris. Odpowiada on mianowicie sytuacji, w której klocki, które opadają na planszę, mają regularną strukturę (np. mają kształt prostokątów), a naszym celem jest, dla każdego klocka, orzec, w którym miejscu się on zatrzyma, jeśli spuścimy go z zadanej pozycji początkowej (nie mamy możliwości wykonywania w locie obrotów ani przesunięć). Z kolei wariant $(+, +)$ reprezentuje dynamiczny problem obliczania sum częściowych ciągu.

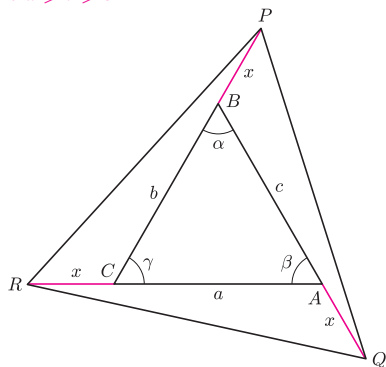
Trzy z podanych wariantów wyjściowego problemu mają zatem jasno określoną interpretację. Co więcej, znane są ich efektywne rozwiązania, w których koszt obsługi ciągu m zapytań to $O(n + m \log n)$ (szukaną strukturą danych są tzw. drzewa przedziałowe, o których więcej można przeczytać lub posłuchać na stronie <http://was.zaa.mimuw.edu.pl>).

*pracownik firmy Google



Rozwiązanie zadania M 1381.

Przyjmijmy oznaczenia boków i kątów trójkąta ABC jak na rysunku i załóżmy, że $a \geq b \geq c$.



Wówczas $\alpha \geq \beta \geq \gamma$, a stąd $\cos \alpha \leq \cos \beta \leq \cos \gamma$, ponieważ funkcja cos jest malejąca na przedziale $(0, \pi)$. Przyjmijmy, że $AP = BQ = CR = x$. Wówczas z twierdzenia cosinusów otrzymujemy

$$QR^2 = x^2 + (a+x)^2 + 2x(a+x)\cos\beta,$$

$$PQ^2 = x^2 + (c+x)^2 + 2x(c+x)\cos\alpha.$$

Gdyby było $a > c$, to ponieważ $\cos \beta \geq \cos \alpha$, mielibyśmy $QR > PQ$, co przeczyłoby założeniu, że trójkąt PQR jest równoboczny. W takim razie $a = b = c$.

Za to czwarty wariant, $(MAX, +)$, jest inny. Jego interpretacja kombinatoryczna nie jest wcale aż tak jasna, a do tego nie znamy sposobu rozwiązania go w czasie $O(n + m \log n)$ (w tym przypadku klasyczne drzewa przedziałowe nie sprawdzają się). Okazuje się jednak, że można go w miarę prosto rozwiązać, korzystając z pewnej *kubelkowej* struktury danych. Nasze rozwiązanie będzie działało w czasie $O(n + m\sqrt{n} \log n)$.

Zacznijmy od podziału wyrazów ciągu pomiędzy kubelki, umieszczając w każdym kubelku, z wyjątkiem ostatniego, $k = \lceil \sqrt{n} \rceil$ wyrazów. Jeśli ostatni kubek jest mniejszy, możemy uzupełnić go sztucznymi wyrazami do rozmiaru k . Oprócz kolejnych wyrazów ciągu kubek będzie przechowywał pewne dane pomocnicze. Zanim opiszemy strukturę kubka, zaproponujmy trochę inny sposób myślenia o operacji $update(i, j, w)$. Zamiast mówić, że dla każdego c_l , takiego że $i \leq l \leq j$, wykonujemy $c_l := \max(c_l, w)$, można wyobrazić sobie, że dodajemy nowe ograniczenie dolne na liczby na pozycjach od i do j . Wtedy wartość liczby w kubku będzie równa maksimum z jej początkowej wartości oraz wszystkich ograniczeń dolnych jej dotyczących. Takie ograniczenia dolne będą trzymane osobno dla każdego kubka.

Struktura kubka będzie następująca:

- $t[1..k]$ – tablica kolejnych wyrazów ciągu, które znajdują się w kubku;
- $sorted_t[1..k]$ – posortowana tablica liczb t ;
- $sum_sorted_t[0..k]$ – sumy prefiksowe tablicy $sorted_t[1..k]$, czyli $sum_sorted_t[p] = sorted_t[1] + sorted_t[2] + \dots + sorted_t[p]$;
- $minimum$ – dolne ograniczenie na wszystkie wyrazy ciągu przechowywane w kubku, czyli maksimum z wartości w z operacji $update(i, j, w)$ dotyczących całego przedziału kubka; aktualna wartość l -tej liczby z kubka to zawsze $\max(t[l], minimum)$.

Każdy kubek umożliwia wykonywanie następujących operacji pomocniczych (szczegółowy opis ich implementacji znajduje się w dalszej części artykułu):

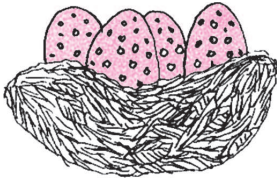
1. $sum_all()$ – obliczenie sumy aktualnych wartości wszystkich wyrazów ciągu zawartych w kubku (działa w czasie $O(\log k)$);
2. $sum(i, j)$ – obliczenie sumy wyrazów o indeksach od i do j , dla $1 \leq i \leq j \leq k$ (działa w czasie $O(k)$);
3. $update_all(new_minimum)$ – aktualizacja dolnego ograniczenia na całym przedziale do wartości co najmniej $new_minimum$ (działa w czasie $O(1)$);
4. $update(i, j, new_minimum)$ – zwiększenie wyrazów o indeksach od i do j , dla $1 \leq i \leq j \leq k$, do wartości co najmniej $new_minimum$ (działa w czasie $O(k \log k)$).

Zauważmy teraz, że dowolny zakres indeksów od i do j ($1 \leq i \leq j \leq n$) można rozbić na pewną liczbę pełnych kubków (oczywiście, nie więcej niż \sqrt{n}) oraz na co najwyżej dwa niepełne kubki (te skrajne).

Dzięki temu zapytanie o sumę liczb na przedziale od i do j można podzielić na nie więcej niż \sqrt{n} zapytań $sum_all()$ oraz co najwyżej dwa zapytania $sum(i', j')$. Stąd każde takie zapytanie obsługujemy w czasie $O(\sqrt{n} \log n)$. Operację modyfikacji ciągu na indeksach od i do j można także wykonać w czasie $O(\sqrt{n} \log n)$, podobnie rozbijając cały przedział na nie więcej niż \sqrt{n} kubków, na których wykonujemy operację $update_all(new_minimum)$, i co najwyżej dwa brzegowe kubki z wykonywaną operacją $update(i', j', new_minimum)$.

Przykład. W tabeli na następnej stronie przedstawiono, jak zmienia się przykładowa kubelkowa struktura danych ($n = 9$) w wyniku pojedynczej modyfikacji. Warto zwrócić uwagę, że w środkowym kubku zmienia się tylko dolne ograniczenie, tj. $minimum$.





	Kubek 1			Kubek 2			Kubek 3					
	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9			
t	4	2	1	7	8	1	1	5	2			
$sorted_t$	1	2	4	1	7	8	1	2	5			
sum_sorted_t	0	1	3	7	0	1	8	16	0	1	3	8
$minimum$	0			0			0					
update(3, 8, 4)												
t	4	2	4	7	8	1	4	5	2			
$sorted_t$	2	4	4	1	7	8	2	4	5			
sum_sorted_t	0	2	6	10	0	1	8	16	0	2	6	11
$minimum$	0			4				0				

Pozostaje opisać, jak korzystając z przechowywanych danych, efektywnie wykonać cztery operacje pomocnicze oferowane przez kubek.

```

function sum_all()
    pos := ostatnia pozycja taka, że sorted_t[pos] < minimum
    return minimum · pos + sum_sorted_t[k] – sum_sorted_t[pos]

```

Ostatnią pozycję pos znajdujemy, wyszukując binarnie. Zauważmy, że w tablicy t jest dokładnie pos liczb mniejszych od $minimum$ oraz dokładnie $k - pos$ liczb nie mniejszych niż $minimum$. Skoro aktualna wartość każdej liczby to $\max(minimum, t[i])$, więc suma aktualnych wartości wynosi $minimum \cdot pos$ (sumujemy pos najmniejszych liczb) plus suma $k - pos$ największych liczb, czyli $sum_sorted_t[k] - sum_sorted_t[pos]$.

```

function sum(i, j)
    result := 0
    for p := i to j do
        result := result + max(minimum, t[p])
    return result

```

W powyższej funkcji sumujemy aktualne wartości wyrazów z kubelka z określonych pozycji.

Aktualizacja dolnego ograniczenia na cały kubek jest bardzo łatwa:

```

function update_all(new_minimum)
    minimum := max(new_minimum, minimum)

```

Jeśli pojawia się nowe dolne ograniczenie, które nie dotyczy całego zakresu obejmowanego przez kubek, lecz jedynie jego części, należy zaktualizować liczby tylko z tego zakresu. Niestety, po takiej operacji tablice $sorted_t$ oraz sum_sorted_t stają się nieaktualne, więc obliczamy je ponownie. Oto zapis stosownego algorytmu:

```

function update(i, j, new_minimum)
    for p := i to j do
        t[p] := max(t[p], new_minimum)
    sorted_t := sort(t[1..k])
    sum_sorted_t[0] := 0
    for p := 1 to k do
        sum_sorted_t[p] := sum_sorted_t[p - 1] + sorted_t[p]

```

Zaprezentowane rozwiązanie jest dosyć szybkie, jednak nie widać powodu, dla którego problemu ($MAX, +$) nie można by rozwiązać jeszcze szybciej. Być może któremuś z Czytelników uda się skonstruować takie rozwiązanie?

Czytelnik, który do końca lipca 2013 roku przyśle do redakcji rozwiązanie opisanego problemu o najlepszej złożoności, nie gorszej jednak niż $O(n + m\sqrt{n})$, zostanie nagrodzony kilogramem szwajcarskiej czekolady w przesyłce prosto z Zurychu. W przypadku wielu rozwiązań o tej samej złożoności nagrodzimy subiektywnie najładniejsze, w przypadku tego samego najładniejszego rozwiązania – pierwsze spośród otrzymanych.

