Jakub Radoszewski, Wojciech Rytter — Treść zadania, Opracowanie
Michał Pilipczuk — Program

OI, Etap II, dzień pierwszy, 7.02.2008

BBB

Bajtazar ma konto w Bajtockim Banku Bitowym (w skrócie BBB). Na początku na koncie było p, a na końcu q bajtalarów. Wszystkie transakcje polegały na wpłacie bądź wypłacie jednego bajtalara. Bilans konta nigdy nie zszedł poniżej zera. Kasjer przygotował wyciąg z konta: pasek papieru z ciągiem plusów i minusów (plus oznacza wpłatę, a minus wypłatę jednego bajtalara). Okazało się, że operacje wykonywane na koncie nie zawsze były poprawnie księgowane. Kasjer nie może wydrukować nowego wyciągu, lecz musi poprawić ten już wydrukowany. Wyciąg nie musi być zgodny z rzeczywistością; wystarczy, że ciąg operacji na wyciągu będzie spełniał następujące dwa warunki:

- *saldo końcowe będzie się zgadzać z saldem początkowym i ciągiem operacji na wyciągu,*
- *zgodnie z ciągiem operacji na wyciągu, saldo konta nigdy nie spadło poniżej zera.*

Twoim zadaniem jest policzenie minimalnego czasu, jaki kasjer potrzebuje na korektę wyciągu. Kasjer może:

- *w ciągu x sekund zmienić wybraną operację na wyciągu na przeciwną, lub*
- *w ciągu y sekund przenieść ostatnią operację na początek wyciągu.*

Jeśli $p = 2$, $q = 3$, to na przykład $--++-+-++-+-+$ jest poprawnym wyciągiem. Natomiast wyciąg $---++++++$ nie jest poprawny, gdyż po trzeciej operacji saldo konta spadłoby poniżej zera, a ponadto saldo końcowe powinno wynosić 3, a nie 5. Możemy go jednak poprawić, zmieniając przedostatni symbol na przeciwny i następnie przenosząc ostatnią operację na początek wyciągu.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia aktualny wygląd wyciągu z konta Bajtazara (ciąg plusów i minusów) oraz liczby p, q, x i y,*
- *wypisze na standardowe wyjście minimalną ilość czasu, potrzebną na taką korektę wyciągu, żeby początkowy i końcowy bilans zgadzały się oraz żeby saldo konta w żadnym momencie nie spadło poniżej zera.*

Wejście

Pierwszy wiersz zawiera 5 liczb całkowitych n, p, q, x oraz y ($1 \leqslant n \leqslant 1\,000\,000$, $0 \leqslant p,q \leqslant 1\,000\,000$, $1 \leqslant x,y \leqslant 1000$), pooddzielanych pojedynczymi odstępami i oznaczających odpowiednio: liczbę transakcji wykonanych przez Bajtazara, początkowe i końcowe saldo konta oraz liczby sekund potrzebne na wykonanie pojedynczej zamiany i przesunięcia operacji na wyciągu. Drugi wiersz zawiera ciąg n plusów i/lub minusów, niezawierający odstępów.

Wyjście

Pierwszy i jedyny wiersz wyjścia powinien zawierać jedną liczbę całkowitą, równą minimalnemu czasowi potrzebnemu na poprawienie wyciągu. Jeśli wyciąg nie wymaga poprawek, tą liczbą powinno być zero. Możesz założyć, że odpowiedni ciąg modyfikacji wyciągu będzie istniał dla każdych danych testowych.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`9 2 3 2 1` `---++++++`	`3`

Rozwiązanie

O co chodzi w zadaniu?

Przedstawmy zadanie BBB w nieco krótszy i przystępniejszy sposób, co ułatwi nam poszukiwanie jego rozwiązania. Mamy dany ciąg a złożony z plusów i minusów (oznaczających odpowiednio plus jedynki i minus jedynki). Taki ciąg wygodnie jest przedstawić jako wykres stanu konta w zależności od czasu, na którym + jest zilustrowany jako odcinek jednostkowy skierowany w górę pod kątem 45°, natomiast - jako podobny odcinek, tyle że skierowany w dół (przykłady takich wykresów znajdują się na rysunkach w dalszej części rozwiązania). Mamy do dyspozycji operacje zamiany znaku na przeciwny (koszt x) oraz przesunięcia cyklicznego ciągu o 1, czyli przesunięcia ostatniego znaku na początek (koszt y). Chcielibyśmy doprowadzić zadany wykres do postaci, w której zaczyna się na wysokości p, kończy na wysokości q oraz nie schodzi nigdy poniżej zera. Dodatkowo, mamy w zadaniu zagwarantowane to, że poszukiwane przekształcenie ciągu jest możliwe.

Na skróty

Dla niecierpliwych Czytelników już na samym początku umieszczamy krótki opis algorytmu rozwiązującego niniejsze zadanie. Jeśli to Wam wystarczy, to gratulujemy. W przeciwnym razie zapraszamy do lektury dalszej części opracowania, gdzie znajdziecie dokładne omówienie tego rozwiązania.

Przypomnijmy, że dla ciągu $a_1, a_2, \ldots, a_n$ ciąg sum częściowych $b_1, b_2, \ldots, b_n$ jest zdefiniowany (dla każdego i) następująco:

$$b_i = a_1 + a_2 + \ldots + a_i.$$

Algorytm wzorcowy

1. Wyznaczamy tablicę `MinPrefSuma`$[0..n-1]$. `MinPrefSuma`$[i]$ to minimalna spośród sum częściowych ciągu $a_{n-i+1}, \ldots, a_n, a_1, \ldots, a_{n-i}$.

2. Na podstawie powyższej tablicy obliczamy tablicę `ZamianyBezObrotów`$[0..n-1]$. `ZamianyBezObrotów`$[i]$ to minimalna liczba zamian znaków potrzebna, aby ciąg $a_{n-i+1},\ldots,a_n,a_1,\ldots,a_{n-i}$ przekształcić w poprawny wyciąg z konta.

3. Wyznaczamy wynik równy:

$$\min\{x\cdot \texttt{ZamianyBezObrotów}[i]+y\cdot i:\ 0\leqslant i<n\}.$$

Sposób realizacji punktu pierwszego w złożoności czasowej $O(n)$ został opisany w rozdziale *Ostatni problem*. Związek między pierwszym i drugim punktem jest wyjaśniony w rozdziale *Wersja ogólna*. Sposób wyznaczenia wartości `ZamianyBezObrotów`$[i]$ na podstawie `MinPrefSuma`$[i]$ w czasie stałym (czyli realizacja punktu drugiego) znajduje się w rozdziale *Łatwiejszy problem*.

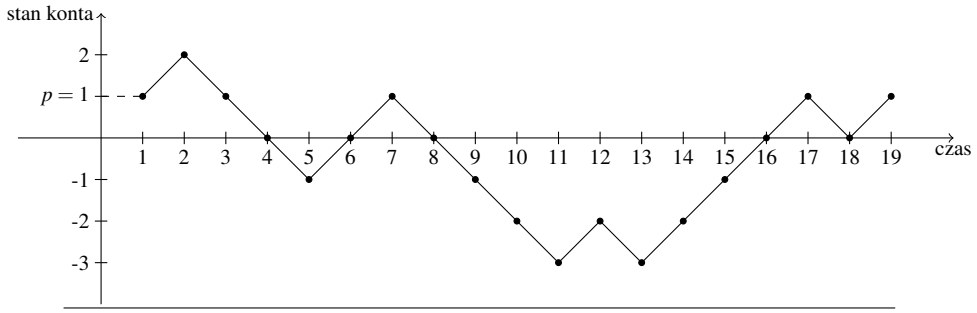
Łatwiejszy problem

Zastanawiając się nad rozwiązaniem niniejszego zadania, warto rozpocząć od uproszczenia problemu. Spróbujmy więc najpierw ograniczyć zakres dopuszczalnych operacji na ciągu i skupmy się jedynie na takich przekształceniach, które polegają tylko na zamianach znaków. W związku z tym możemy na jakiś czas zapomnieć o kosztach operacji x, y i dążyć jedynie do minimalizacji liczby wykonanych zamian. Niech

$$W = \texttt{ZamianyBezObrotów}[0].$$

Pokażemy, że wartość W zależy tylko od p i q oraz dwóch dodatkowych parametrów wyjściowego ciągu:

- różnicy salda końcowego i początkowego, czyli sumy wszystkich plus jedynek i minus jedynek z ciągu (s);
- minimalnej wartości salda w ciągu, czyli najniższego punktu na wykresie stanu konta (m [1]).



Rys. 1: Przykładowy wykres stanu konta w zależności od czasu ($n=18$, $p=1$) dla wyciągu `+--++----+-++++-+`. Dla tego wykresu mamy $s=0$, $m=-3$. Wyciąg z wykresu jest błędny, gdyż występują w nim ujemne salda.

[1] Warto zauważyć, że $m = \texttt{MinPrefSuma}[0]+p$. Pozwala to śledzić związek aktualnych rozważań z przedstawionym na początku algorytmem.

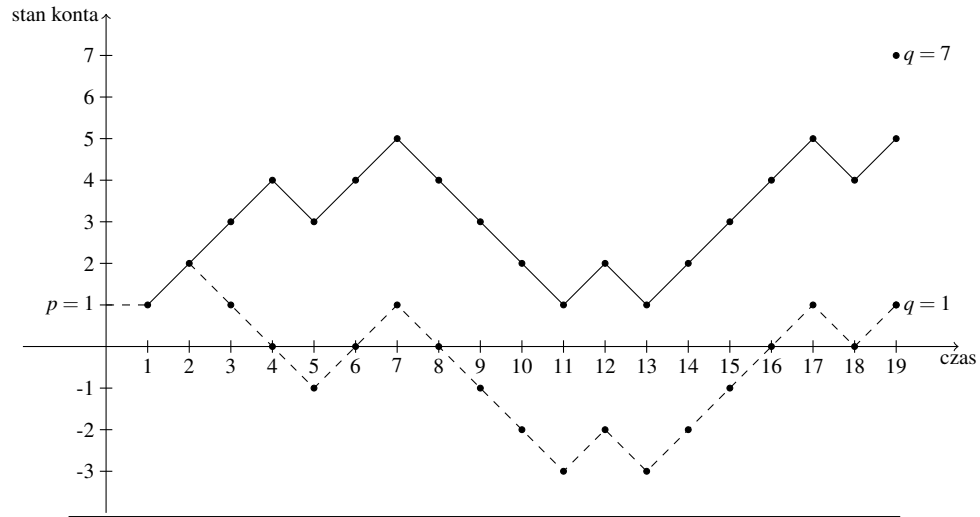
Aby skorygować ciąg operacji, musimy zapewnić właściwe saldo na końcu i nieujemny stan konta w całym okresie, którego dotyczy wyciąg. Policzymy najpierw wartość w — minimalną liczbę zmian znaków w ciągu konieczną do tego, aby stan konta był zawsze nieujemny. Jeżeli $m \geqslant 0$, to oczywiście żadne zamiany nie są potrzebne, czyli $w = 0$. W przeciwnym przypadku wystarczy zamienić $w = \lceil \frac{-m}{2} \rceil$ początkowych minusów na plusy[2]. Każda taka zamiana podnosi o 2 jednostki dalszą część wykresu stanu konta, więc po wykonaniu w takich zamian:

- początkowy odcinek ciągu, w którym dokonujemy korekty, składa się z samych plusów, więc wykres nie może na tym odcinku spaść poniżej zera;
- cały wykres po ostatniej operacji zamiany podniesie się o

$$2 \cdot w = 2 \cdot \left\lceil \frac{-m}{2} \right\rceil \geqslant -m,$$

  a zatem na tym odcinku wykres także nie spada poniżej zera.

Warto także zauważyć, że mniejsza liczba operacji zamiany nie wystarcza do skorygowania wyciągu.



Rys. 2: Wyciąg z rys. 1 poprawiony tak, by stan konta nie spadał poniżej zera. Dwa pierwsze minusy zostały zamienione na plusy, więc reszta wykresu podniosła się o 4 jednostki. W tym wyciągu saldo końcowe nie musi się jednak zgadzać z wymaganym (może być zbyt duże, np. dla $q = 1$, lub zbyt małe, np. dla $q = 7$).

Teraz pozostaje nam zmienić jak najmniej znaków, aby doprowadzić saldo końcowe wyciągu do wymaganej wartości. Musimy przy tym uważać, żeby nie popsuć już osiągniętej właściwości nieujemności stanu konta.

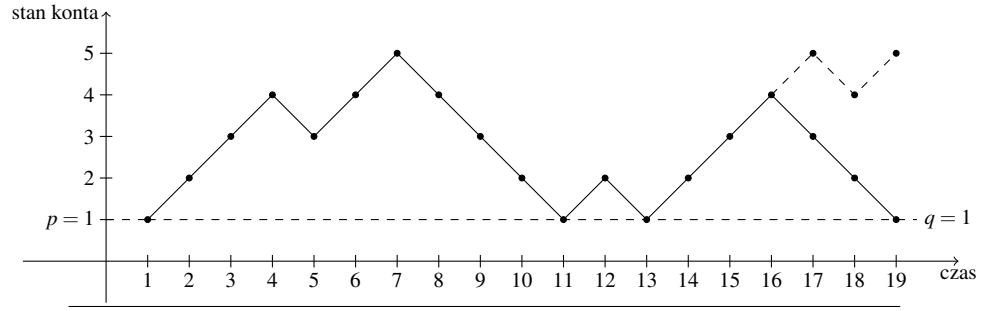
[2] Dla formalności dodajmy, że w ciągu znajduje się co najmniej $-m$ minusów, gdyż startujemy od nieujemnego stanu konta $p \geqslant 0$.

Po wykonaniu zamian z poprzedniej fazy różnica między saldem końcowym a początkowym wynosi $s+2w$, a powinna wynosić $q-p$. Z porównania tych liczb wychodzą dwa proste przypadki do rozważenia.

Jeżeli $q-p<s+2w$, to musimy obniżyć aktualne saldo końcowe, czyli zamienić

$$\frac{(s+2w)-(q-p)}{2} \tag{1}$$

plusów na minusy. Aby nie doprowadzić przy tym do ujemnego stanu konta, do zamiany wybierzemy ostatnie tyle plusów z ciągu. W ten sposób oczywiście osiągniemy pożądane saldo. Dodatkowo, ponieważ końcowy fragment wyciągu będzie składał się z samych minusów, więc na pewno wprowadzone zmiany nie spowodują zejścia wykresu poniżej zera.

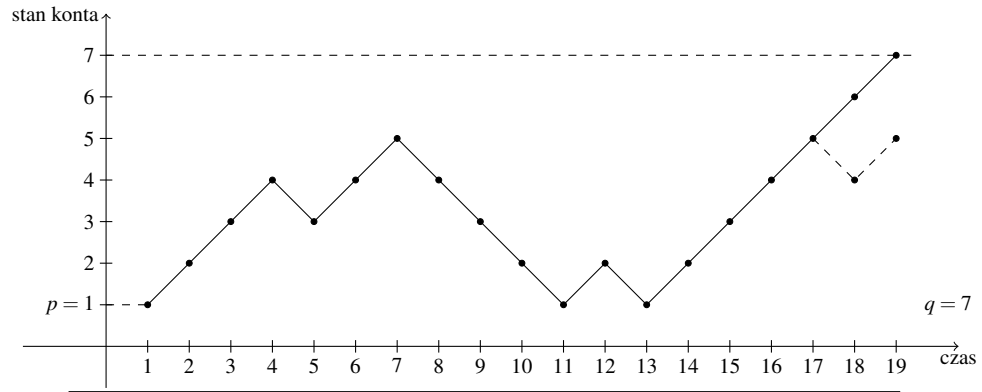


Rys. 3: Jeżeli $q=1$, to zachodzi pierwszy przypadek: $q-p=0<4=s+2w$. Aby naprawić sytuację, zmieniamy $4/2=2$ ostatnie plusy na minusy.

Jeżeli zaś $q-p\geqslant s+2w$, to musimy podwyższyć aktualne saldo końcowe, czyli zamienić

$$\frac{(q-p)-(s+2w)}{2} \tag{2}$$

minusów na plusy. Ponieważ żadna taka zamiana nie może spowodować obniżenia wykresu, w szczególności jego zejścia poniżej zera, to możemy do tego celu wybrać dowolne minusy z ciągu operacji, na przykład te końcowe.



Rys. 4: Jeżeli $q=7$, to zachodzi drugi przypadek: $q-p=6>4=s+2w$. Aby naprawić sytuację, zmieniamy $2/2=1$ ostatni minus na plus.

Podobnie jak poprzednio, łatwo sprawdzić, że wszystkie wykonane na tym etapie zamiany znaków są konieczne. Trzeba jeszcze chwilkę zastanowić się, czy są możliwe. Aby dało się je wykonać, wyrażenia (1) oraz (2) muszą mieć całkowite wartości i wartości te muszą być nie większe niż odpowiednio liczby plusów i minusów w wyjściowym ciągu. Okazuje się, że własności te wynikają stąd, że w zadaniu mamy zagwarantowane istnienie jakiegoś rozwiązania. Faktycznie:

- Żadna z operacji dopuszczonych w zadaniu nie zmienia parzystości sumy wszystkich plusów i minusów. Jeżeli więc jakieś rozwiązanie istnieje, to liczby s (wartość początkowa tej sumy) oraz $q-p$ są tej samej parzystości, a zatem wartości (1) i (2) są całkowite.
- Na mocy poczynionego założenia musi zachodzić $q-p \in [-n,n]$. Do tego samego przedziału musi oczywiście należeć także $s+2w$. Wystarczy teraz zauważyć, że z ciągu plus i minus jedynek o dowolnej sumie z przedziału $[-n,n]$ da się dojść do ciągu o dowolnej innej sumie z tego samego przedziału i o tej samej parzystości za pomocą zamian znaków.

Podsumowaniem powyższych rozważań jest następująca obserwacja:

Obserwacja 1. Przy oznaczeniu

$$w = \begin{cases} \lceil \frac{-m}{2} \rceil & \text{jeżeli } m < 0 \\ 0 & \text{w przeciwnym przypadku,} \end{cases} \tag{3}$$

minimalna liczba operacji korekty wyciągu, gdy nie dopuszczamy przesunięć cyklicznych, a zezwalamy jedynie na zamiany znaków, wynosi

$$W = \begin{cases} w + \frac{s+2w-q+p}{2} & \text{jeżeli } q-p < s+2w, \\ w + \frac{q-p-s-2w}{2} & \text{w przeciwnym przypadku.} \end{cases} \tag{4}$$

Zauważmy, że jeżeli znamy wartości s oraz m, to W możemy wyznaczyć w czasie stałym.

Wersja ogólna

Najwyższa pora powrócić do oryginalnej wersji zadania i przypomnieć sobie, że mamy do dyspozycji także cykliczne przesunięcia ciągu. Łatwo zauważyć, że zamiany i przesunięcia są całkowicie niezależne od siebie nawzajem — możemy założyć, że najpierw wykonywane są wszystkie przesunięcia cykliczne, a dopiero potem zamiany znaków. Możemy więc rozważyć wszystkich możliwe przesunięcia cykliczne zadanego ciągu (kolejno o 0 znaków, 1 znak, 2 znaki...), policzyć wartość W dla każdego z nich i wybrać optymalne rozwiązanie, uwzględniając koszty x, y operacji przesunięcia i zamiany. Odpowiada to obliczeniu zawartości tablicy `ZamianyBezObrotów`$[0..n-1]$ i wyznaczeniu końcowego minimum za pomocą wzoru z ostatniego punktu algorytmu wzorcowego.

Jak już zauważyliśmy, do efektywnego policzenia wartości W dla danego ciągu wystarczy znajomość wartości s oraz m dla tego ciągu. Oczywiście s nie zmienia się przy przesunięciu cyklicznym ciągu, więc możemy ją sobie policzyć raz na początku.

Większym problemem jest wyznaczenie wartości m, gdyż one mogą ulegać zmianie przy przesuwaniu cyklicznym ciągu. Musimy jednak jakoś sobie z nimi poradzić. Przede

wszystkim pozbądźmy się parametru p ze wzoru na m. Przy liczeniu m możemy po prostu założyć, że $p = 0$, wyznaczyć przy tym założeniu wartość m', a następnie przyjąć $m = m' + p$. Dla $p = 0$ wartość m możemy zdefiniować jako minimum w ciągu sum częściowych wyjściowego ciągu plus i minus jedynek. W ten sposób dochodzimy do problemu wyznaczenia wartości minimalnych ciągów sum częściowych dla wszystkich przesunięć cyklicznych wyjściowego ciągu, czyli tablicy `MinPrefSuma`$[0..n-1]$ z algorytmu wzorcowego. Można to oczywiście obliczyć bezpośrednio, lecz wtedy otrzymamy rozwiązanie o złożoności czasowej $O(n^2)$ (zaimplementowane w plikach `bbbs1.cpp` oraz `bbbs2.pas`). Rzut oka na ograniczenia z zadania pozwala stwierdzić, że taka metoda nie wystarcza (dodatkowo warto zauważyć, że od tego fragmentu zależy w tej chwili efektywność całego rozwiązania, gdyż wszystkie pozostałe obliczenia potrafimy już wykonać w czasie liniowym).

Ostatni problem

Sformułujmy problem, który pozostał nam do rozwiązania, abstrahując od plusów i minusów, sald bankowych oraz wszystkich parametrów z zadania. Mamy dany ciąg $a_1, \ldots, a_n$. Chcemy dla każdego przesunięcia cyklicznego tego ciągu o i znaleźć minimalną wartość `MinPrefSuma`$[i]$ występującą w jego ciągu sum częściowych.

Niech $b_1, \ldots, b_n$ będzie ciągiem sum częściowych wyjściowego ciągu $a_1, \ldots, a_n$. Jak wówczas wygląda ciąg sum częściowych $d_1^i, \ldots, d_n^i$ dla ciągu a, w którym i końcowych wyrazów przerzucimy na początek? Jest to, jak łatwo sprawdzić:

$$\begin{aligned}
d_1^i &= b_{n-i+1} - b_{n-i} &&= a_{n-i+1} \\
d_2^i &= b_{n-i+2} - b_{n-i} &&= a_{n-i+1} + a_{n-i+2} \\
&\vdots \\
d_i^i &= b_n - b_{n-i} &&= a_{n-i+1} + a_{n-i+2} + \ldots + a_n \\
d_{i+1}^i &= b_1 + (b_n - b_{n-i}) &&= a_{n-i+1} + a_{n-i+2} + \ldots + a_n + a_1 \\
d_{i+2}^i &= b_2 + (b_n - b_{n-i}) &&= a_{n-i+1} + a_{n-i+2} + \ldots + a_n + a_1 + a_2 \\
&\vdots \\
d_n^i &= b_{n-i} + (b_n - b_{n-i}) &&= a_{n-i+1} + a_{n-i+2} + \ldots + a_n + a_1 + a_2 + \ldots + a_{n-i}
\end{aligned} \tag{5}$$

Innymi słowy:

a) W przesunięciu cyklicznym i końcowych wyrazów ciągu a przerzucamy na początek. To powoduje, że i początkowych wyrazów ciągu sum częściowych jest obliczanych na podstawie tych wyrazów. Dodatkowo, tych i początkowych wyrazów różni się od i końcowych wyrazów ciągu sum częściowych oryginalnego ciągu a tylko o sumę

$$a_1 + a_2 + \ldots + a_{n-i} = b_{n-i}.$$

b) Kolejnych $n - i$ wyrazów ciągu sum częściowych różni się od oryginalnych, początkowych $n - i$ wyrazów ciągu sum częściowych dla ciągu a o wartość

$$a_{n-i+1} + a_{n-i+2} + \ldots + a_n = b_n - b_{n-i}.$$

Teraz przydadzą nam się dwa dodatkowe ciągi $pref$ oraz suf, zdefiniowane dla każdego j następująco:

$$\begin{aligned} pref_j &= \min\{b_1, b_2, \ldots, b_j\} \\ suf_j &= \min\{b_j, b_{j+1}, \ldots, b_n\}. \end{aligned}$$

Zauważmy, że minimum ciągu (5) możemy za ich pomocą wyrazić następująco:

$$\min(suf_{n-i+1} - b_{n-i},\ pref_{n-i} + (b_n - b_{n-i})). \tag{6}$$

Pierwszy człon w powyższym zapisie pochodzi z wyrazów przerzuconych w trakcie przesunięcia cyklicznego (opisanych w podpunkcie a)), natomiast drugi — z pozostałych (opisanych w podpunkcie b)).

Widzimy, że jeżeli znamy ciągi b, $pref$ oraz suf, to minimum dla ciągu przesuniętego cyklicznie o i liter możemy obliczyć za pomocą wzoru (6) w złożoności czasowej $O(1)$. Pozostaje zauważyć, że każdy z tych ciągów możemy wyznaczyć w złożoności $O(n)$ — najpierw liczymy b od lewej do prawej, a potem, na podstawie b, $pref$ również od lewej do prawej oraz suf od prawej do lewej.

W ten sposób uzyskujemy rozwiązanie wzorcowe, o złożoności czasowej i pamięciowej $O(n)$. Zostało ono zaimplementowane w plikach `bbb.cpp`, `bbb1.pas` oraz `bbb2.java`.

Inne rozwiązania

Zdecydowana większość alternatywnych rozwiązań tego zadania różni się od wzorcowego jedynie metodą podejścia do ostatniej fazy, czyli do wyznaczania minimów w ciągach sum częściowych dla przesunięć cyklicznych zadanego ciągu. Zauważmy, że wszystkie przesunięcia cykliczne ciągu a występują jako spójne n-elementowe fragmenty ciągu aa (czyli ciągu powstałego przez sklejenie dwóch kopii ciągu a). Niech $c_1, \ldots, c_{2n}$ oznacza ciąg sum częściowych ciągu aa. Wówczas łatwo zauważyć, że jeżeli od każdego wyrazu ciągu

$$c_i, c_{i+1}, \ldots, c_{i+n-1}$$

(gdzie $i \leqslant n$) odejmiemy sumę $a_1 + \ldots + a_{i-1} = c_{i-1}$, to otrzymamy ciąg sum częściowych dla przesunięcia cyklicznego ciągu a o $n-i$ wyrazów. To oznacza, że gdybyśmy potrafili dla każdego „okna" długości n w ciągu c wyznaczyć minimum, to na tej podstawie bylibyśmy w stanie łatwo obliczać szukane minima ciągów sum częściowych dla wszystkich przesunięć cyklicznych wyjściowego ciągu.

Jeszcze inny problem

Sprowadziliśmy zatem podproblem wyjściowego problemu do... jeszcze innego problemu. Tym razem mamy dany ciąg $c_1, \ldots, c_{2n}$ i chcielibyśmy poznać minimalną wartość w każdym oknie (tzn. spójnym fragmencie) długości n tego ciągu.

Ten klasyczny problem można rozwiązać na wiele sposobów. Prostą metodę (działającą także wówczas, gdy interesują nas okna różnych rozmiarów) o złożoności $O(n \log n)$ otrzymujemy przy użyciu statycznego drzewa przedziałowego czy też licznikowego,

w którego liściach zapisujemy elementy ciągu c, natomiast w węzłach wewnętrznych — minima z poddrzew im odpowiadających[3].

Bardziej bezpośrednie podejście do problemu polega na utrzymywaniu zbioru n kolejnych elementów ciągu c w strukturze danych, która pozwala na dodawanie i usuwanie elementów (co odpowiada przesunięciu okna o jedną pozycję w prawo) oraz na odpowiadanie na pytania o minimalną wartość zawartą w zbiorze. Taką strukturą danych, po dodaniu odpowiedniej tablicy, w której zapamiętamy bieżące pozycje elementów w strukturze, może być kopiec binarny (stóg)[4]. To rozwiązanie, o złożoności czasowej $O(n \log n)$, zostało zaimplementowane w plikach `bbb6.cpp` oraz `bbb7.pas` — uzyskiwało ono na zawodach maksymalną liczbę punktów. Zamiast kopca możemy zastosować także zrównoważone drzewo BST (otrzymamy wówczas dokładnie tę samą złożoność czasową). Programujący w C++ mogą wykorzystać gotową implementację, tj. strukturę `multiset` z biblioteki STL. Takie rozwiązanie znajduje się w pliku `bbbs7.cpp` — nie uzyskiwało ono na zawodach maksymalnej liczby punktów ze względu na nadmierne zapotrzebowanie na pamięć.

Jest także kolejne rozwiązanie zadania o złożoności czasowej $O(n)$, do implementacji którego wystarczy nam jedna lista, którą oznaczymy przez L. W tym rozwiązaniu wykorzystujemy obserwację, że jeżeli w oknie znajdują się dwa elementy c_i oraz c_j, takie że $i < j$ oraz $c_i \geqslant c_j$, to c_i nie będzie miało wpływu na wynik ani dla bieżącej pozycji okna, ani też dla żadnej kolejnej (dlaczego tak jest?). Na liście L będziemy więc przechowywać tylko te elementy z aktualnego okna, które są mniejsze od wszystkich znajdujących się na prawo od nich w oknie. Łatwo widać, że L będzie zawsze posortowana rosnąco, więc jej pierwszy element będzie zarazem elementem najmniejszym dla rozważanego okna.

W jaki sposób należy zmodyfikować L przy przesunięciu okna o jedną pozycję w prawo? Jeżeli pierwszy (i zarazem najmniejszy) element L wypada przy przesunięciu poza okno, to należy go usunąć z L. Z kolei element, który dochodzi do okna z prawej strony, wymusi usunięcie z L wszystkich elementów większych od niego (czyli pewnej liczby elementów z końca listy), po czym sam zostanie dodany na końcu listy L.

Poza operacją usuwania elementów z końca listy L, wszystkie pozostałe ewidentnie sumują się do złożoności czasowej $O(n)$. Do przeanalizowania usuwania z listy wykorzystamy analizę kosztu zamortyzowanego, która w tym przypadku jest bardzo prosta: każdy element zostaje w algorytmie wstawiony na listę dokładnie raz, a łączna liczba usunięć z listy jest nie większa niż liczba wstawień na listę. To pokazuje, że również sumaryczny koszt usunięć można oszacować przez $O(n)$, co kończy analizę złożoności tego rozwiązania. Jego implementację można znaleźć w pliku `bbb3.cpp`.

Okazuje się, że jeżeli przypomnimy sobie o jednej szczególnej własności, którą posiada ciąg c z naszego zadania, to rozważany problem można rozwiązać jeszcze łatwiej, nadal w czasie liniowym. Zauważmy mianowicie, że c to ciąg sum częściowych ciągu złożonego tylko z wyrazów $+1$ i -1, czyli że każde dwa kolejne wyrazy ciągu c różnią się co najwyżej o 1. W takim przypadku zbiór liczb zawartych w oknie możemy zapamiętać w prostej tablicy zliczającej. Dopracowanie szczegółów tego pomysłu pozostawiamy jako ćwiczenie. W jego wykonaniu mogą być pomocne rozwiązania z plików `bbb4.cpp` oraz `bbb5.pas`.

Na koniec dodajmy, że dość dokładna analiza większości z opisanych tu rozwiązań, a także jeszcze jedno ciekawe rozwiązanie liniowe, znajdują się w opracowaniu zadania

[3] Więcej o statycznych drzewach przedziałowych można przeczytać np. w opisie rozwiązania zadania *Tetris 3D* z książeczki XIII Olimpiady Informatycznej [13].

[4] Więcej o kopcach można przeczytać np. w książce [20].

Sound z Bałtyckiej Olimpiady Informatycznej 2007. Materiały te można znaleźć (tylko w wersji anglojęzycznej) np. na stronie `http://www.boi2007.de/tasks/book.pdf`.

Testy

Rozwiązania zawodników były sprawdzane na 13 zestawach testów, których opisy są zawarte w poniższej tabeli.

Nazwa	n	q − p	Opis
bbb1a.in	22	2	mały test poprawnościowy
bbb1b.in	1	−1	przypadek brzegowy
bbb2a.in	61	1	mały test poprawnościowy
bbb2b.in	47	25	mały test poprawnościowy
bbb3a.in	288	12	mały test poprawnościowy
bbb3b.in	375	211	mały test poprawnościowy
bbb4a.in	2761	11	mały test poprawnościowy
bbb4b.in	2741	719	mały test poprawnościowy
bbb5a.in	8921	−41	mały test poprawnościowy
bbb5b.in	7654	5298	mały test poprawnościowy
bbb6.in	98867	19465	średni test
bbb7.in	201346	−50	średni test
bbb8.in	361555	201	średni test
bbb9a.in	499822	−10230	średni test
bbb9b.in	478963	108939	średni test
bbb10.in	808669	687	duży test
bbb11.in	912880	−2238	duży test
bbb12.in	970293	−853	duży test
bbb13a.in	999879	−4789	duży test
bbb13b.in	1000000	340354	duży test