Jakub Pachocki, Wojciech Rytter — Treść zadania, Opracowanie
Igor Adamski — Program

Dostępna pamięć: 64 MB. *OI, etap I, 15.10–12.11.2012*

Usuwanka

Mała Bajtosia dostała w prezencie grę o nazwie **Usuwanka**. *W tej grze dany jest ciąg n przylegających do siebie klocków, ponumerowanych kolejno od 1 do n. Każdy klocek jest albo biały, albo czarny, przy czym białych klocków jest k razy więcej niż czarnych. Gra jest jednoosobowa. Celem gry jest usunięcie, za pomocą określonych ruchów, wszystkich klocków z ciągu.*

Pojedynczy ruch polega na usunięciu dokładnie k białych i jednego czarnego klocka bez zmiany pozycji pozostałych klocków. Ruch jest dozwolony, jeśli między żadnymi dwoma usuwanymi w tym ruchu klockami nie ma „dziury", czyli pustego miejsca po uprzednio usuniętym klocku.

Pomóż Bajtosi, ona tak bardzo Cię prosi... Znajdź dowolny ciąg dozwolonych ruchów prowadzący do usunięcia wszystkich klocków.

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite n oraz k ($2 \leqslant n \leqslant 1\,000\,000$, $1 \leqslant k \leqslant n-1$), oddzielone pojedynczym odstępem, oznaczające liczbę klocków w grze oraz liczbę białych klocków, które należy usunąć w każdym ruchu. We wszystkich testach zachodzi warunek $k+1 \mid n$.

W drugim wierszu znajduje się napis złożony z n liter `b` *oraz* `c`. *Kolejne litery napisu określają kolor kolejnych klocków:* `b` *– biały,* `c` *– czarny. Możesz założyć, że dla danych testowych istnieje szukany ciąg ruchów prowadzący do usunięcia wszystkich klocków.*

W testach wartych łącznie 40% punktów zachodzi dodatkowy warunek $n \leqslant 10\,000$.

Wyjście

Twój program powinien wypisać na standardowe wyjście $\frac{n}{k+1}$ wierszy. Kolejne wiersze powinny opisywać kolejne ruchy. Każdy z tych wierszy powinien zawierać $k+1$ liczb, podanych w kolejności rosnącej oraz rozdzielonych pojedynczymi odstępami, oznaczających numery klocków, które należy usunąć w danym ruchu.

Przykład

Dla danych wejściowych:

```
12 2
ccbcbbbbbbcb
```

poprawnym wynikiem jest:

```
1 8 12
2 6 7
3 4 5
9 10 11
```

Wyjaśnienie do przykładu: *Niech* ␣ *oznacza puste miejsce po usuniętym klocku. Wykonując podane powyżej ruchy, uzyskujemy kolejno następujące układy klocków:*

1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
c	c	b	c	b	b	b	b	b	b	c	b
␣	c	b	c	b	b	b	␣	b	b	c	␣
␣	␣	b	c	b	␣	␣	␣	b	b	c	␣
␣	␣	␣	␣	␣	␣	␣	␣	b	b	c	␣
␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣	␣

Rozwiązanie

Zastąpmy w naszym ciągu początkowym klocki białe przez wartości $+1$, a klocki czarne przez wartości $-k$. Otrzymamy w ten sposób ciąg liczbowy o sumie 0. Tego typu ciągi (o sumie zero, składające się z wartości $+1$ i $-k$) będziemy w tym zadaniu nazywać ciągami *usuwankowymi*.

Z treści zadania wiemy, że dla początkowego ciągu istnieje sekwencja dozwolonych ruchów usuwająca wszystkie klocki; powiemy, że ciąg ten jest *rozwiązywalny*. Ostatni ruch musi usuwać *spójny* ciąg $k+1$ klocków, tzn. ciąg klocków parami sąsiednich. Wynika z tego, że w naszym ciągu istnieje spójny usuwankowy podciąg o długości $k+1$. Okazuje się, że tę własność spełnia *dowolny* ciąg usuwankowy:

Lemat 1. W dowolnym ciągu usuwankowym istnieje spójny usuwankowy podciąg o długości $k+1$.

Dowód: Niech n oznacza długość ciągu. Podzielmy ciąg na $\frac{n}{k+1}$ bloków o długości $k+1$. Wyrazów równych $-k$ w ciągu również jest $\frac{n}{k+1}$, więc w każdym bloku jest dokładnie jedno $-k$ albo istnieje blok, w którym nie ma żadnego $-k$. W pierwszym przypadku każdy z bloków jest szukanym podciągiem. W drugim przypadku wybieramy ten blok, który zawiera same $+1$, i przesuwamy go w prawo bądź lewo do najbliższego $-k$, uzyskując w ten sposób podciąg o sumie 0. ■

Lemat 2. Każdy ciąg usuwankowy jest rozwiązywalny.

Dowód: Dowód przeprowadzimy przez indukcję ze względu na liczbę klocków. Dla $n = k+1$ teza jest oczywista. Jeśli $n > k+1$, to na mocy lematu 1 istnieje spójny podciąg długości $k+1$, który sumuje się do 0. Możemy go usunąć i scalić resztę (jeśli usunięcie rozspójniło ciąg). Na mocy indukcji wiemy, że istnieje sekwencja ruchów usuwająca nowo powstały ciąg, ponieważ ciąg ten jest krótszy (a nadal jest usuwankowy). Wstawiając z powrotem usunięty wcześniej podciąg, uzyskujemy poprawną sekwencję ruchów, która usuwa wszystko prócz tego fragmentu. Na końcu wystarczy zatem usunąć ten fragment. ■

Rozwiązanie wzorcowe

Rozwiązanie wzorcowe to efektywna implementacja dowodu lematu 2. W rozwiązaniu wykorzystamy stos S z następującymi operacjami:

- *push*(i, S) – połóż element i na szczyt stosu S,
- *pop*(S) – usuń element leżący na szczycie stosu S i zwróć jego wartość.

Dodatkowo będziemy chcieli mieć możliwość odpytywania o sumę elementów ciągu wejściowego na pozycjach odpowiadających „górnym" $k+1$ elementom stosu, do czego wykorzystamy operację *topSum*(S). Jeśli rozmiar stosu jest mniejszy niż $k+1$, to operacja ta będzie zwracać -1.

```
 1: function Usuwanka()
 2: begin
 3:   S := S' := pusty stos;
 4:   for i := 1 to n do begin
 5:     push(i, S);
 6:     if topSum(S) = 0 then
 7:       { przełóż k+1 górnych elementów z S do S' }
 8:       for j := 1 to k+1 do
 9:         push(pop(S), S');
10:   end
11:   for i := 1 to n do begin
12:     write(pop(S'));
13:     if i | k+1 then writeln();
14:   end
15: end
```

Łatwo zrealizować operację *topSum*(S) w czasie $O(k)$, co da nam algorytm działający w czasie $O(nk)$. Szybszą implementację uzyskamy, trzymając tablicę *sum*$[0..n]$, w której będziemy pamiętać sumy prefiksowe elementów ciągu wejściowego odpowiadające kolejnym elementom stosu S, tzn. *sum*$[i]$ jest sumą tych elementów ciągu wejściowego, które odpowiadają i „dolnym" elementom stosu S.

Podczas operacji *push*(i, S) musimy uaktualnić tablicę *sum*. Jeśli m jest wielkością stosu S po wykonaniu tej operacji, a a_i jest i-tym elementem ciągu wejściowego, to dajemy $sum[m] := sum[m-1] + a_i$. Operacja *topSum*(S) zwraca po prostu $sum[m] - sum[m-k-1]$ lub -1 jeśli $m < k+1$.

Tak zapisane rozwiązanie wzorcowe działa w czasie $O(n)$. Można je znaleźć w plikach `usu.cpp` i `usu1.pas`. Alternatywne rozwiązanie liniowe zapisano w plikach `usu2.cpp` i `usu3.pas`, natomiast w pliku `usu4.cpp` zaimplementowano rozwiązanie o złożoności $O(n \log k)$, które również otrzymywało na zawodach maksymalną liczbę punktów.

Testy

Testy *1-4* są w większości małymi testami losowymi. Testy *5a-8a* sprawiają kłopoty rozwiązaniom o złożoności $O(\frac{n^2}{k})$, a testy *5b-8b* mają tę samą strukturę, co odpowiadający test *a*, ale z dużym k. Testy z grupy *9* są testami maksymalnymi. Przy każdym teście w opisie podano, jak został on wygenerowany, przy czym *losowy*(b, c) oznacza losowy ciąg, który zawiera b białych klocków i c czarnych.

Nazwa	n	k	Opis
usu1a.in	2	1	bc – test minimalny
usu1b.in	15	2	$b^5c^5b^5$
usu1c.in	45	2	$(bcb)^{15}$
usu1d.in	9	2	*cbcbbbbbc*
usu2a.in	110	10	$b^{100}c^{10}$
usu2b.in	319	10	test losowy
usu3a.in	1200	5	test losowy
usu3b.in	1633	70	test losowy
usu4a.in	7994	3996	$(b^{\frac{2k}{3}}cb^{\frac{k}{3}})^2$
usu4b.in	9898	100	test losowy
usu5a.in	120 000	5	$c^{10\,000}b^{100\,000}c^{10\,000}$
usu5b.in	100 002	50 000	$b^{50\,000}ccb^{50\,000}$
usu6a.in	220 088	21	$cc(b^{19}cc)^{5000}b^{115\,084}cc$
usu6b.in	271 612	12 345	$cc(b^{12\,343}cc)^{10}b^{148\,160}$
usu7a.in	505 000	100	$b^{247\,500}losowy(5000, 5000)b^{247\,500}$
usu7b.in	300 010	30 000	$b^{150\,000}losowy(10, 10)b^{149\,990}$
usu8a.in	728 000	90	$losowy(300, 10)^{400}b^{480\,000}losowy(300, 10)^{400}$
usu8b.in	800 008	100 000	$losowy(150\,000, 2)^2b^{200\,000}losowy(150\,000, 2)^2$
usu9a.in	1 000 000	1	$b^{500\,000}c^{500\,000}$ – test maksymalny
usu9b.in	1 000 000	999 999	$cb^{999\,999}$ – test maksymalny
usu9c.in	1 000 000	9 999	$b^{499\,950}c^{100}b^{499\,950}$ – test maksymalny
usu9d.in	1 000 000	99	$(ccb^{396}cb^{250\,000}losowy(244\,604, 4996)c)^2$ – maksymalny test losowy

Zawody II stopnia

opracowania zadań