Paweł Parys
Treść zadania, Opracowanie

Karol Pokorski
Program

Dostępna pamięć: 64 MB.

OI, etap II, dzień pierwszy, 12.02.2014

Przestępcy

Bajtogród jest pięknym miastem położonym nad rzeką. Znajduje się w nim n domów, ponumerowanych kolejnymi liczbami naturalnymi od 1 do n, i w tej kolejności są ułożone wzdłuż brzegu rzeki.

Bajtogród był bardzo spokojnym miastem, w którym wszystkim dobrze się powodziło. Niestety, od niedawna grasuje tam gang dwóch groźnych przestępców – Bitka i Bajtka. Dokonali już wielu napadów, przez co mieszkańcy boją się wychodzić z domów.

W trakcie rabunku, Bitek i Bajtek wychodzą ze swoich domów i idą naprzeciw siebie – żaden z nich się nie cofa. Bitek idzie w kierunku rosnących numerów, a Bajtek w kierunku malejących numerów domów. Po drodze (zanim się spotkają) każdy z nich wybiera kilka domów, do których się włamuje i kradnie cenne przedmioty (i ważne dane). Po dokonaniu napadów spotykają się w pewnym domu i dzielą się łupami. Mieszkańcy Bajtogrodu mają tego dość – wszyscy chcieliby, aby w mieście ponownie zapanował spokój. Poprosili o pomoc detektywa Bajtoniego.

Detektyw ustalił, że bandyci mieszkają w domach tego samego koloru, niestety nie wiadomo dokładnie w których, ani jaki to kolor. Dodatkowo, anonimowy informator powiedział detektywowi, że bandyci są właśnie w trakcie kolejnego skoku. Z obawy o swoje bezpieczeństwo, informator nie podał dokładnie, w których domach ma nastąpić włamanie. Podał jednak kolory kolejno rabowanych domów. Okazało się, że bandyci są dość przesądni – każdy bandyta obrabuje dom każdego koloru co najwyżej raz.

Bajtoni nie może już dłużej czekać. Chce urządzić zasadzkę w miejscu spotkania przestępców. Pomóż Bajtoniemu, pisząc program, który znajdzie wszystkie możliwe miejsca spotkania przestępców.

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite n oraz k ($3 \leqslant n \leqslant 1\,000\,000$, $1 \leqslant k \leqslant 1\,000\,000$, $k \leqslant n$) oddzielone pojedynczym odstępem, oznaczające liczbę domów oraz liczbę kolorów domów w Bajtogrodzie. Kolory są ponumerowane kolejnymi liczbami naturalnymi od 1 do k. W drugim wierszu wejścia znajduje się ciąg n liczb całkowitych $c_1, c_2, \ldots, c_n$ ($1 \leqslant c_i \leqslant k$), pooddzielanych pojedynczymi odstępami. Są to kolory kolejnych domów w Bajtogrodzie.

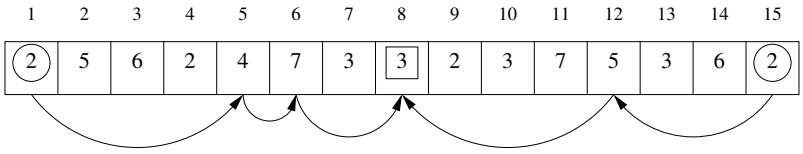
W trzecim wierszu wejścia znajdują się dwie liczby całkowite m oraz l ($1 \leqslant m,l \leqslant n$, $m + l \leqslant n - 1$) oddzielone pojedynczym odstępem i określające liczbę domów odwiedzonych kolejno przez Bitka i Bajtka. W czwartym wierszu wejścia znajduje się m parami różnych liczb całkowitych $x_1, x_2, \ldots, x_m$ ($1 \leqslant x_i \leqslant k$), pooddzielanych pojedynczymi odstępami. Są to kolory domów odwiedzonych przez Bitka podane w kolejności odwiedzania (wyłączając dom Bitka). W piątym i ostatnim wierszu wejścia znajduje się l parami różnych liczb całkowitych $y_1, y_2, \ldots, y_l$ ($1 \leqslant y_i \leqslant k$), pooddzielanych pojedynczymi odstępami. Są to kolory domów odwiedzonych przez Bajtka podane w kolejności odwiedzania (wyłączając dom Bajtka). Zachodzi warunek $x_m = y_l$, jest to kolor domu, w którym bandyci będą dzielić łupy.

Wyjście

Twój program powinien wypisać na standardowe wyjście dokładnie dwa wiersze. W pierwszym wierszu powinna znaleźć się liczba domów, w których zgodnie z warunkami zadania mogą spotkać się bandyci. W drugim wierszu powinien znaleźć się rosnący ciąg numerów tych domów, pooddzielanych pojedynczymi odstępami. Jeśli bandyci nie mogą się spotkać w żadnym miejscu, to pierwszy wiersz ma zawierać liczbę 0, a drugi wiersz ma być pusty.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`15 7`	`3`
`2 5 6 2 4 7 3 3 2 3 7 5 3 6 2`	`7 8 10`
`3 2`	
`4 7 3`	
`5 3`	



Wyjaśnienie do przykładu: *W powyższym przykładzie bandyci mogą mieszkać w domach o kolorach 2 (Bitek mieszkałby wtedy w domu o numerze 1 lub 4, a Bajtek w domu numer 15) lub 6 (Bitek mieszkałby w domu numer 3, natomiast Bajtek w domu numer 14). Przykładowy scenariusz drogi Bitka (z domu numer 1) jest następujący: odwiedza domy numer: 5 (koloru 4), 6 (koloru 7), a potem 7, 8 lub 10 (koloru 3). Bajtek może wtedy rabować dom numer 12 (koloru 5), a następnie spotkać się z Bitkiem w domu 7, 8 lub 10 (koloru 3). Powyższy rysunek przedstawia sytuację, w której bandyci spotykają się w domu numer 8.*

Testy „ocen":

`1ocen`: $n = 7$, $k = 3$, $m = 2$, $l = 3$, *bandyci mogą dzielić łupy tylko w jednym domu;*

`2ocen`: $n = 10$, $k = 3$, $m = 3$, $l = 2$, *bandyci nie mogą się spotkać w żadnym miejscu;*

`3ocen`: $n = 1000$, $k = 1$, $m = 1$, $l = 1$, *wszystkie domy są tego samego koloru;*

`4ocen`: $n = 1\,000\,000$, $k = 1000$, $m = l = 100$, *ciąg domów składa się z 1000 identycznych fragmentów po 1000 domów, pokolorowanych kolejnymi kolorami od 1 do 1000; ciągi domów odwiedzanych przez obu bandytów są postaci* $1, 2, 3, \ldots, 100$.

Rozwiązanie

Rozwiązanie wzorcowe

Siłowym sposobem obliczenia wyniku byłoby sprawdzanie wszystkich możliwych par domów, w których mogą mieszkać Bitek i Bajtek, a dla każdej z nich wszystkich możliwych ciągów kolejno napadanych domów, spełniających warunki zadania. Jednak program działający w ten sposób byłby oczywiście bardzo wolny, dlatego powinniśmy najpierw zastanowić się, jak zmniejszyć liczbę możliwości, które musimy przeanalizować. Jak można się domyślać z rozmiaru danych na wejściu, rozwiązanie powinno działać w czasie liniowym względem n.

Po pierwsze zauważmy, że jeśli bandyci mieszkają w domach o jakimś kolorze c, to możemy założyć, że Bitek mieszka w pierwszym od lewej domu tego koloru, a Bajtek w ostatnim z nich (stosujemy tutaj konwencję, że dom o numerze 1 znajduje się po lewej stronie, a dom o numerze n po prawej). Istotnie, jeśli Bitek może napaść na pewien ciąg domów, zaczynając z domu o numerze i, to może napaść na ten sam ciąg domów, także zaczynając z domu o mniejszym numerze; analogicznie, lecz odwrotnie, jest z Bajtkiem.

Podobnie rzecz ma się z kolejnymi domami odwiedzanymi przez bandytów: jeśli Bitek jest w domu o numerze i i ma napaść na dom o ustalonym kolorze c, to może wybrać pierwszy dom po i mający kolor c; analogicznie dla Bajtka. Istotnie, możliwości kolejnych rabunków z tego domu mogą być tylko większe niż z dalszych domów o tym samym kolorze. Nie dotyczy to oczywiście ostatniego napadanego domu (czyli miejsca spotkania), gdyż w jego przypadku interesuje nas wyznaczenie wszystkich możliwości.

Widzimy więc, że chcemy rozważać ciągi $s_0, s_1, \ldots, s_{m-1}$ numerów domów odwiedzanych przez Bitka (przy czym s_0 to miejsce jego zamieszkania, natomiast miejsce spotkania jest pominięte) spełniające następujące warunki:

- $1 \leqslant s_0 < s_1 < \ldots < s_{m-1} \leqslant n$, czyli Bitek idzie tylko w prawo;
- $c_{s_i} = x_i$ dla każdego $i \in \{1, \ldots, m-1\}$, czyli napadane domy mają kolory jak podano na wejściu;
- $c_j \neq c_{s_0}$ dla każdego $j \in \{1, \ldots, s_0 - 1\}$, czyli Bitek mieszka w pierwszym domu pewnego koloru;
- $c_j \neq x_i$ dla wszystkich $i \in \{1, \ldots, m-1\}$ oraz $j \in \{s_{i-1} + 1, \ldots, s_i - 1\}$, czyli napadany jest zawsze pierwszy możliwy dom danego koloru (nie dotyczy to ostatniego napadu – miejsca spotkania).

Ciąg taki nazwiemy *dobrym ciągiem Bitka*. Podobnie, *dobrym ciągiem Bajtka* nazwiemy ciąg $r_0, r_1, \ldots, r_{l-1}$ taki, że:

- $1 \leqslant r_{l-1} < r_{l-2} < \ldots < r_0 \leqslant n$;
- $c_{r_i} = y_i$ dla każdego $i \in \{1, \ldots, l-1\}$;
- $c_j \neq c_{r_0}$ dla każdego $j \in \{r_0 + 1, \ldots, n\}$;

- $c_j \neq y_i$ dla wszystkich $i \in \{1, \ldots, l-1\}$ oraz $j \in \{r_i + 1, \ldots, r_{i-1} - 1\}$.

Bandyci mogą spotkać się w domu numer a, jeśli istnieje dobry ciąg Bitka $s_0, s_1, \ldots, s_{m-1}$ oraz dobry ciąg Bajtka $r_0, r_1, \ldots, r_{l-1}$ takie, że

- $c_{s_0} = c_{r_0}$, czyli bandyci mieszkają w domu o tym samym kolorze;
- $c_a = x_m$, czyli miejsce spotkania ma podany kolor;
- $s_{m-1} < a < r_{l-1}$, czyli Bitek przychodzi na miejsce spotkania z lewej, a Bajtek z prawej.

Wprost z powyższej definicji wynika, że dla ustalonego koloru c istnieje co najwyżej jeden dobry ciąg Bitka zaczynający się od domu o kolorze c (tzn. taki, że $c_{s_0} = c$). Łączna długość wszystkich takich ciągów, dla wszystkich c, jest rzędu $O(k \cdot m)$, więc wyznaczenie ich wszystkich w całości byłoby zbyt wolne. Widzimy jednak, że cały ciąg nie jest nam potrzebny, wystarczy znać ostatni jego element, tzn. s_{m-1}; oznaczmy go $d[c]$. Podobnie, przez $e[c]$ oznaczmy ostatni element dobrego ciągu Bajtka $r_0, r_1, \ldots, r_{l-1}$ takiego, że $c_{r_0} = c$. (Dla niektórych kolorów c wartość $d[c]$ lub $e[c]$ może być niezdefiniowana, jeśli żaden dobry ciąg nie istnieje). Daje to nam podział zadania na dwa kroki:

1. Dla każdego c oblicz $d[c]$ oraz $e[c]$.
2. Znajdź wszystkie a takie, że $c_a = x_m$ oraz $d[c] < a < e[c]$ dla pewnego c.

Zacznijmy od prostszej części, czyli od rozwiązania kroku 2. Mamy tutaj dane pewne przedziały: dla każdego c przedział od $d[c]$ do $e[c]$. Chcemy znaleźć wszystkie a należące któregokolwiek z tych przedziałów (spośród nich odfiltrujemy w trywialny sposób tylko takie, że $c_a = x_m$). Na początek odrzućmy takie c, że $d[c]$ lub $e[c]$ jest niezdefiniowane lub $d[c]$ nie jest mniejsze niż $e[c]$. Następnie posortujmy wszystkie początki oraz końce przedziałów (czyli liczby $d[c]$ oraz $e[c]$). Aby uzyskać złożoność liniową można użyć sortowania kubełkowego, aczkolwiek w praktyce spokojnie wystarczy QuickSort z biblioteki standardowej. Teraz będziemy iść od lewej do prawej po kolejnych numerach domów, trzymając licznik ilości otwartych przedziałów. Gdy zaczyna się nowy przedział, licznik zwiększamy; gdy jakiś przedział się kończy – zmniejszamy; jeśli licznik ma wartość dodatnią, to numer analizowanego domu możemy wziąć jako a: znajduje się on wewnątrz pewnego przedziału. Nie musimy się przy tym zastanawiać nad faktem czy przedziały są otwarte czy domknięte, gdyż końce przedziałów mają kolor x_{m-1} lub y_{l-1}, który z założenia jest inny niż x_m.

Wróćmy teraz do kroku 1. Tutaj potrzeba minimalnej ilości sprytu, gdyż liczenie dobrych ciągów dla każdego koloru z osobna byłoby za wolne. Skoncentrujmy się na dobrych ciągach Bitka, czyli na liczbach $d[c]$; liczby $e[c]$ oblicza się analogicznie. Będziemy wyznaczać dobre ciągi Bitka po kolei dla wszystkich możliwych s_0, idąc od lewej do prawej. Dla $s_0 = 1$ robimy to zupełnie wprost: znajdujemy najmniejszy numer domu $s_1 > s_0$ mającego kolor x_1, następnie najmniejszy numer domu $s_2 > s_1$ mającego kolor x_2, itd. W ten sposób obliczyliśmy $d[c_1]$. Następnie (w pętli) przesuwamy s_0 na kolejny dom, nie zapominając ani nie zmieniając liczb $s_1, \ldots, s_{m-1}$. Interesują nas przy tym tylko takie s_0, że żaden dom wcześniej nie ma takiego samego koloru jak

dom numer s_0. Oczywiście po takim zwiększeniu liczby s_0 nasz ciąg być może nie jest już dobrym ciągiem Bitka; trzeba go wówczas poprawić. Robimy to tak: dla kolejnych i (zaczynając od 1, aż do $m-1$) sprawdzamy, czy $s_{i-1} < s_i$. Jeśli dla pewnego i jest to prawdą, to przerywamy poprawianie: dostaliśmy dobry ciąg Bitka, czyli jako $d[c_{s_0}]$ powinniśmy wziąć s_{m-1}. A jeśli nie, to przesuwamy s_i na kolejny dom tego samego koloru, aż do momentu, gdy liczba s_i stanie się większa od s_{i-1}. Po takim przesuwaniu musimy sprawdzić kolejne i, gdyż teraz być może przestało być prawdą, że $s_i < s_{i+1}$.

Dość łatwo przekonać się o poprawności tej metody. W momencie, gdy sprawdzamy czy $s_{i-1} < s_i$, to dzięki wcześniejszemu poprawianiu wiemy, że $s_0 < s_1 < \ldots < s_{i-1}$, natomiast z poprzedniego kroku dużej pętli wiemy, że $s_i < \ldots < s_{m-1}$; jeśli dodatkowo okazuje się, że $s_{i-1} < s_i$, to kolejność wszystkich elementów ciągu staje się prawidłowa. W dodatku widzimy, że dzięki naszej „zachłanności", czyli wybieraniu zawsze pierwszej możliwości od lewej, pozostałe warunki dobrego ciągu będą spełnione.

Pozostaje drobny szczegół implementacyjny. Otóż, aby nasz algorytm działał szybko, powinniśmy umieć w czasie stałym przesunąć s_i na numer kolejnego domu tego samego koloru. Aby to było możliwe, musimy po prostu wyliczyć sobie na początku dla każdego domu wskaźnik na kolejny dom tego samego koloru. Łatwo to zrobić idąc od lewej do prawej i pamiętając dla każdego koloru ostatni dotychczas widziany dom tego koloru. Wtedy dla kolejno analizowanego domu mamy numer poprzedniego domu w takim samym kolorze; tworzymy między nimi wskaźnik. Przy okazji dla każdego domu możemy sobie zaznaczyć, czy istnieje wcześniejszy dom tego samego koloru – ta informacja jest przydatna podczas przesuwania s_0.

Przeanalizujmy teraz kwestię złożoności naszego algorytmu (dla kroku 1). Kluczowe jest tu założenie z treści zadania, że liczby $x_1, \ldots, x_m$ są parami różne. Zatem jako s_i próbujemy brać numery domu innego koloru niż jako s_j dla $j \neq i$ (gdzie $i, j \geqslant 1$). Innymi słowy, każdy numer domu pojawi się najwyżej raz jako s_0, oraz najwyżej raz jako pewien s_i dla $i \geqslant 1$. Ponadto dla każdej wartości s_0 być może dla wielu wartości i zwiększamy liczbę s_i, ale tylko dla jednego i wykonujemy porównanie s_i z s_{i-1} niekończące się przesunięciem s_i. Wynika z tego, że nasz algorytm działa w czasie liniowym, czyli tak jak chcieliśmy.

Rozwiązanie wzorcowe można znaleźć w plikach `prz.cpp` i `prz1.pas`. Wydaje się, że rozwiązania optymalne muszą działać z grubsza według powyższego opisu.

Rozwiązania nieoptymalne i błędne

Można próbować oczywiście naiwnie poprawiać dobre ciągi Bitka i Bajtka. Po pierwsze, bez użycia tablicy wskaźników do kolejnego domu tego samego koloru – idąc po prostu po kolejnych domach i sprawdzając, czy mają właściwy kolor. Wówczas każdym z m elementów ciągu odwiedzimy praktycznie wszystkie n domów. Po drugie, możemy nie przerywać poprawiania, gdy okaże się, że $s_{i-1} < s_i$ – czyli sprawdzać ten warunek zawsze dla wszystkich i. W tym przypadku też prawie dla każdego z n domów (jako elementu s_0) przejrzymy cały ciąg długości m. Oba te rozwiązania prowadzą do istotnie gorszej, kwadratowej złożoności obliczeniowej. Testy zostały tak dobrane, aby takie rozwiązania otrzymywały co najwyżej 40% maksymalnej punktacji. Rozwią-

zania pierwszego typu można znaleźć w plikach `przs1.cpp` i `przs2.pas`, a drugiego typu w plikach `przs3.cpp` i `przs4.pas`.

Podstawowym błędem może być pominięcie warunku o tym, że kolory domów bandytów muszą być jednakowe. Takie rozwiązania pomijają chyba najważniejszą trudność tego zadania, dlatego intencją Jury było, aby otrzymały 0 punktów. Przykładowa implementacja znajduje się w pliku `przb1.cpp`.