Krzysztof Sikora
Treść zadania, Opracowanie

Tomasz Malesiński
Program

OI, Etap I

Samochodziki

Jasio jest trzylatkiem i bardzo lubi bawić się samochodzikami. Jasio ma n różnych samochodzików. Wszystkie samochodziki leżą na wysokiej półce, do której Jasio nie dosięga. W pokoju jest mało miejsca, więc na podłodze może znajdować się jednocześnie co najwyżej k samochodzików.

Jasio bawi się jednym z samochodzików leżących na podłodze. Oprócz Jasia w pokoju cały czas przebywa mama. Gdy Jasio chce się bawić jakimś innym samochodzikiem i jeśli ten samochodzik jest na podłodze, to sięga po niego. Jeśli jednak samochodzik znajduje się na półce, to musi mu go podać mama. Mama podając Jasiowi jeden samochodzik, może przy okazji zabrać z podłogi dowolnie wybrany przez siebie inny samochodzik i odłożyć go na półkę (tak, aby na podłodze nie brakło miejsca).

Mama bardzo dobrze zna swoje dziecko i wie dokładnie, którymi samochodzikami Jasio będzie chciał się bawić. Dysponując tą wiedzą mama chce zminimalizować liczbę przypadków, gdy musi podawać Jasiowi samochodzik z półki. W tym celu musi bardzo rozważnie odkładać samochodziki na półkę.

Zadanie

Napisz program który:

- *wczyta ze standardowego wejścia ciąg kolejnych samochodzików, którymi będzie chciał się bawić Jasio,*
- *obliczy minimalną liczbę przypadków zdejmowania przez mamę samochodzików z półki,*
- *wypisze wynik na standardowe wyjście.*

Wejście

W pierwszym wierszu standardowego wejścia znajdują się trzy liczby całkowite: n, k, p ($1 \leqslant k \leqslant n \leqslant 100\,000$, $1 \leqslant p \leqslant 500\,000$), pooddzielane pojedynczymi odstępami. Są to kolejno: łączna liczba samochodzików, liczba samochodzików mogących jednocześnie znajdować się na podłodze oraz długość ciągu samochodzików, którymi będzie chciał się bawić Jasio. W kolejnych p wierszach znajduje się po jednej liczbie całkowitej. Są to numery kolejnych samochodzików, którymi będzie chciał się bawić Jasio (samochodziki są ponumerowane od 1 do n).

Wyjście

W pierwszym i jedynym wierszu standardowego wyjścia należy zapisać jedną liczbę całkowitą — minimalną liczbę przypadków podania samochodzika z półki przez mamę.

Przykład

Dla danych wejściowych:

```
3 2 7
1
2
3
1
3
1
2
```

poprawnym wynikiem jest:

```
4
```

Rozwiązanie

Wprowadzenie

Problem przedstawiony w zadaniu jest związany z problemem realizacji pamięci wirtualnej oraz zagadnieniem stronicowania występującymi w systemach operacyjnych ([32], rozdział 9.3). Rozkazy wykonywane przez procesor oraz dane, na których on operuje, muszą znajdować się w pamięci operacyjnej. Jej wielkość jest jednak ograniczona i dlatego część danych jest przechowywana w wolniejszej pamięci zewnętrznej (na przykład na dysku). Dane w pamięci operacyjnej i na dysku podzielone są na porcje zwane stronami. Zawsze, gdy procesor potrzebuje strony, której nie ma aktualnie w pamięci operacyjnej, występuje *błąd braku strony*. System operacyjny musi wówczas ściągnąć potrzebną stronę z dysku do pamięci. Może jednak zdarzyć się, że w pamięci nie ma już wolnego miejsca na nową stronę. Wówczas najpierw trzeba przenieść inną, chwilowo niepotrzebną stronę na dysk. Ponieważ obsługa błędów braku strony jest kosztowna czasowo, zależy nam na zminimalizowaniu liczby ich wystąpień. Podstawową metodą pozwalającą osiągnąć ten cel jest odpowiednia strategia doboru stron, które przenosimy na dysk robiąc miejsce w pamięci operacyjnej.

W zadaniu, Jasio odgrywa rolę procesora, mama zaś spełnia zadania systemu operacyjnego. Odpowiednikiem ograniczonej pamięci operacyjnej jest podłoga, natomiast bardziej pojemną pamięć zewnętrzną reprezentuje półka. Wreszcie samochodziki, którymi bawi się chłopiec, to strony pamięci z potrzebnymi danymi. Podstawowa różnica pomiędzy historyjką przedstawioną w zadaniu, a rzeczywistą sytuacją polega na tym, że system operacyjny nie jest w stanie przewidzieć przyszłych potrzeb procesora tak dobrze, jak mama potrafi przewidzieć zachcianki Jasia. W praktycznych zastosowaniach algorytm *off–line*, który musi mieć na wejściu zadany z góry ciąg odwołań do stron, jest właściwie bezużyteczny. Jedynymi praktycznymi rozwiązaniami są algorytmy typu *on–line* wyznaczające strategię sprowadzania i usuwania stron z pamięci (stronicowania) na podstawie informacji o wykorzystaniu stron w przeszłości. Strategie *off–line* są stosowane tylko do sprawdzenia jakości innych algorytmów, gdyż jak się wkrótce przekonamy, istnieje *optymalny* algorytm stronicowania *off–line*, który powoduje minimalną liczbę błędów braku strony.

Rozwiązanie wzorcowe

Wprowadźmy następujące oznaczenia:

- n — liczba wszystkich samochodzików,
- k — liczba samochodzików mogących znajdować się jednocześnie na podłodze,
- p — długość ciągu samochodzików, którymi będzie chciał się bawić Jasio,
- $S(t)$ — samochodzik, którym Jasio chce się bawić w chwili t ($t = 1, \ldots, p$);
- $P(t)$ — zbiór samochodzików przygotowanych dla Jasia do zabawy w chwili t (w tym celu mama być może musi odłożyć na półkę jakiś samochodzik i zdjąć z półki inny — wykonuje to wszystko pomiędzy chwilą $t-1$ a chwilą t, w dalszej części będziemy pisać, że mama wykonuje te czynności *tuż przed chwilą* t).

Rodzina zbiorów $P(t)$ dla $t = 0, \ldots, n$ dokładnie opisuje czynności wykonywane przez mamę w trakcie zabawy Jasia (ze zbioru $P(t) \setminus P(t-1)$ można wywnioskować, który samochodzik został zabrany, a który podany przez mamę tuż przed chwilą t).

Definicja 1 Rodzinę zbiorów P nazwiemy *strategią*, a mówiąc algorytm P, będziemy mieć na myśli algorytm, który usuwa samochodziki z podłogi zgodnie ze strategią P.

Strategia P jest *poprawna*, jeśli spełnia następujące warunki:

1. $P(0) = \emptyset$,
2. $S(t) \in P(t)$ dla $t = 1, \ldots, p$,
3. $|P(t)| \leqslant k$ dla $t = 1, \ldots, p$.

Strategia P jest *optymalna*, jeśli jest poprawna i nie istnieje inna strategia, w której mama wykonuje mniej operacji zdjęcia samochodzika z półki.

Fakt 1 *Jeśli w pewnej strategii optymalnej P mama zabiera z podłogi na półkę samochodzik s tuż przed chwilą t oraz $|P(t)| < k$, to strategia P', różniąca się tylko tym, że mama nie zabiera tuż przed chwilą t z podłogi samochodzika s, jest nadal optymalna.*

Zatem możemy poszukiwać strategii optymalnej tylko pośród tych, w których mama nie odkłada na półkę samochodzika, dopóki na podłodze jest miejsce na kolejny. Pozostaje ustalić, który samochodzik mama ma zabrać z podłogi, gdy wystąpi taka konieczność. Okazuje się, że wystarczy, jeśli będzie to samochodzik, którym chłopiec najdłużej nie będzie chciał się bawić, spośród tych, które obecnie znajdują się na podłodze.

Twierdzenie 2 *Strategia OPT, która polega na odkładaniu na półkę samochodzika, o który Jasio poprosi ponownie najpóźniej, jest optymalna.*

Dowód Pokażemy, że mając daną dowolną strategię optymalną P, możemy skonstruować strategię OPT powodującą tyle samo błędów „braku samochodzika", co P, w której zawsze usuwany jest z podłogi samochodzik, o który Jaś poprosi ponownie najpóźniej.

Przypuśćmy, że w strategii P powyższa zasada nie jest przestrzegana i t_1 jest pierwszą chwilą, w której na półkę jest odkładany samochodzik s inny niż ten, którym Jasio najdłużej nie będzie się bawił (oznaczmy go s'). Rozważmy strategię P' równą P z dokładnością do s i s', to znaczy:

1. $P'(t) = P(t)$ dla $t < t_1$;
2. tuż przed chwilą t_1 w strategii P' odkładamy na półkę samochodzik s', a pozostawiamy s (w strategii P odkładamy s i zatrzymujemy s');
3. przez wszystkie kolejne chwile t, dopóki chłopiec nie zechce bawić się s, a mama w strategii P nie decyduje się odstawić na półkę samochodzika s', obie strategie mogą działać identycznie dla samochodzików innych niż s i s', tzn. $P'(t) = P(t) \setminus \{s'\} \cup \{s\}$.

W strategii P przychodzi w końcu chwila (oznaczmy ją t_2), że chłopiec chce się bawić samochodzikiem s. Może także zdarzyć się, że tuż przed pewną chwilą (oznaczmy ją t_3) mama decyduje się odstawić s' na półkę.

Przypadek 1: Przypuśćmy, że $t_3 < t_2$. Wówczas definiujemy strategię P' tak, by tuż przed chwilą t_3 mama odstawiała na półkę samochodzik s. W dalszej części obie strategie są identyczne.

Przypadek 2a: Niech $t_2 < t_3$ i w strategii P mama usuwa z podłogi tuż przed chwilą t_2 samochodzik $s'' \neq s'$, by zrobić miejsce dla s. Wówczas w strategii P' tuż przed chwilą t_2 usuwamy z podłogi s'' i zestawiamy z półki s'. Dalej obie strategie przebiegają identycznie.

Przypadek 2b: Niech $t_2 < t_3$ i w strategii P mama usuwa z podłogi tuż przed chwilą t_2 samochodzik s', by zrobić miejsce dla s. Wówczas w strategii P' nie musimy dokonywać żadnej zamiany tuż przed chwilą t_2 i począwszy od tej chwili mamy sytuację identyczną jak w P. Osiągamy ją jednak wykonując jedną zamianę mniej, co stoi w sprzeczności z założeniem, że P była optymalna. Opisany przypadek nie może więc wystąpić.

Zastanówmy się, co zyskaliśmy konstruując strategię P'. Otrzymaliśmy strategię nadal optymalną, bo wymagającą takiej samej liczby zamian co P, ale zachowującą dłużej zasadę odstawiania na półkę samochodzika, którym chłopiec będzie chciał się bawić najpóźniej. Jeżeli w P' zasada ta jest nadal łamana (w jakimś późniejszym momencie niż w P), to możemy przekształcić tym razem P' w kolejną optymalną strategię odsuwając jeszcze dalej w czasie złamanie zasady. W ten sposób w skończonej liczbie kroków otrzymujemy strategię *OPT* (optymalną, bo wymagającą tyle samo zamian, co wyjściowa strategia P), w której w każdej chwili jest przestrzegana omawiana zasada. To kończy dowód twierdzenia.

■

Implementacja

Kluczowym elementem implementacji jest wybór struktury danych umożliwiającej wydajne sprawdzenie, który samochodzik należy usunąć z podłogi. W rozwiązaniu wzorcowym użyto kopca binarnego. Znajdują się w nim numery samochodzików będących aktualnie na podłodze, uporządkowane malejąco według czasów następnego żądania Jasia (na szczycie znajduje się auto, którym chłopiec najdłużej nie będzie się bawił). Dzięki takiemu wyborowi struktury danych, operacje wyznaczenia samochodzika do odłożenia na półkę oraz dołączenia nowego — podanego Jasiowi — do struktury można wykonać w czasie $O(\log k)$.

Dodatkowo dla każdego samochodzika utrzymywany jest stos zawierający czasy jego żądań uporządkowanych rosnąco (na szczycie stosu znajduje się czas następnego żądania). Pozwala to w czasie stałym wyznaczyć czas kolejnego żądania wykorzystywany podczas operacji na kopcu. Poniższy program przedstawia pseudokod algorytmu wzorcowego:

```
1:  { zainicjowanie zmiennej zliczającej }
2:  { liczbę operacji zdjęcia samochodzika z półki }
3:  wynik:=0;
4:  { zainicjowanie pustego kopca }
5:  HeapInit(h);
6:  { zainicjowanie stosów dla każdego samochodzika }
7:  for i:=1 to n do
8:    begin
9:      { pusty stos }
10:     StackInit(Pos[i]);
11:     { element symbolizujący nieskończoność }
12:     Push(Pos[i], p+1);
13:   end
14: for t:=p downto 1 do
15:   begin
16:     { budowa stosów czasów żądań samochodzików }
17:     Push(Pos[S(t)], t);
18:   end
19: for t:=1 to p do
20:   begin
21:     Pop(Pos[S(t)]);
22:     if S(t) nie należy do kopca h then
23:       begin
24:         { nie ma miejsca na podłodze }
25:         if Size(h) = k then
26:           begin
27:           { usuń numer samochodu, którego Jaś zażąda ponownie najpóźniej }
28:               HeapDeleteMax(h);
29:           end
30:         { dodaj do kopca element S(t) z czasem Top(Pos[S(t)]) }
31:         HeapInsert(h,S(t));
32:         wynik:=wynik+1;
33:       end
34:     else
35:       begin
36:         { zwiększ wartość elementu S(t) na czas Top(Pos[S(t)]) }
37:         HeapIncreaseKey(h,S(t));
38:       end
39:   end
```

Łatwo, można zauważyć, że algorytm działa w czasie $O(n + p\log k)$. Złożoność pamięciowa wynosi $O(p+n+k)$.

Testy

Zadanie testowane było na zestawie 10 danych testowych.

Nazwa	n	k	p	Opis
sam1.in	10	3	20	prosty test poprawnościowy
sam2.in	100	30	500	test losowy
sam3.in	500	200	2000	test losowy
sam4.in	1000	200	20000	test z wolno zmieniającym się zbiorem
sam5.in	5000	2000	40000	test losowy
sam6.in	10000	3000	80000	test losowy
sam7.in	20000	1000	100000	test z wolno zmieniającym się zbiorem
sam8.in	50000	20000	300000	test losowy
sam9.in	100000	50000	500000	test losowy
sam10.in	100000	80000	500000	test losowy