Michał Włodarczyk
Treść zadania, Opracowanie

Bartosz Łukasiewicz
Program

Dostępna pamięć: 256 MB. *OI, etap II, dzień próbny, 10.02.2015*

Logistyka

Bajtazar ma firmę logistyczną. Klienci firmy często zlecają przewiezienie dużych ilości towarów, które nie mieszczą się w pojedynczej ciężarówce. Wtedy Bajtazar wysyła konwój. Czasami do konwoju jest przypisanych więcej kierowców niż ciężarówek. Zapasowi kierowcy jadą wtedy jako pasażerowie. Przyjmujemy, że każda ciężarówka może zabrać dowolnie wielu pasażerów. W każdej chwili kierowcy mogą zdecydować się na postój. Wtedy cały konwój zatrzymuje się, a przed wznowieniem jazdy kierowcy mogą wsiąść do dowolnych ciężarówek i zamieniać się za kierownicą. Nie ma żadnych dolnych ani górnych ograniczeń na liczbę postojów na trasie.

Aby zwiększyć bezpieczeństwo na drogach, bajtockie ministerstwo transportu wprowadziło ograniczenia czasu pracy kierowców ciężarówek. Każdy z kierowców, po przejściu okresowych testów psychofizycznych, dostaje wpis do prawa jazdy, ile kilometrów może spędzić za kierownicą pojazdu w czasie jednej podróży.

Bajtazar poprosił Cię o napisanie programu, który pomoże mu zarządzać jego zespołem n kierowców. Program musi obsługiwać dwa typy zdarzeń:

- *Uaktualnienie wpisu w prawie jazdy i-tego kierowcy. Zakładamy, że na początku żaden kierowca nie ma wpisu w prawie jazdy. Dopóki go nie otrzyma, nie może prowadzić ciężarówki.*
- *Zapytanie, czy jest możliwe wysłanie konwoju złożonego z c ciężarówek na trasę o długości s kilometrów. Podczas trasy kierowcy mogą jeździć jako pasażerowie i przesiadać się pomiędzy ciężarówkami. Zlecenia są obsługiwane pojedynczo, tzn. kolejny konwój rusza w trasę dopiero po powrocie poprzedniego.*

Wejście

Pierwszy wiersz standardowego wejścia zawiera dwie liczby całkowite n i m ($1 \leqslant n, m \leqslant 1\,000\,000$) oddzielone pojedynczym odstępem, oznaczające liczbę kierowców i liczbę zdarzeń. W kolejnych m wierszach znajdują się opisy kolejnych zdarzeń.

Jeśli jest to zdarzenie uaktualnienia wpisu, to wiersz składa się z litery `U` *oraz dwóch liczb całkowitych k i a ($1 \leqslant k \leqslant n$, $0 \leqslant a \leqslant 1\,000\,000\,000$) oznaczających, że k-ty kierowca może od tej pory przejechać za kierownicą a kilometrów podczas jednej podróży. Jeśli jest to zapytanie, to wiersz składa się z litery* `Z` *oraz dwóch liczb całkowitych c i s ($1 \leqslant c \leqslant n$, $1 \leqslant s \leqslant 1\,000\,000\,000$) oznaczających pytanie, czy jest możliwe przejechanie c ciężarówkami na trasie o długości s kilometrów.*

W testach wartych 33% punktów zachodzi dodatkowy warunek $n, m \leqslant 1000$. W testach wartych 66% punktów zachodzi dodatkowy warunek $a, s \leqslant 1\,000\,000$.

Wyjście

Jeśli na wejściu znajduje się q zapytań, to na standardowe wyjście należy wypisać q wierszy: w i-tym z nich powinno znajdować się słowo `TAK` *lub* `NIE` *oznaczające odpowiedź na i-te zapytanie z wejścia.*

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`3 8`	`NIE`
`U 1 5`	`TAK`
`U 2 7`	`NIE`
`Z 2 6`	`TAK`
`U 3 1`	
`Z 2 6`	
`U 2 2`	
`Z 2 6`	
`Z 2 1`	

Testy „ocen”:

`1ocen`: *jeden kierowca, kilka wpisów i zapytań z odpowiedziami pozytywnymi i negatywnymi;*

`2ocen`: *1000 kierowców, każdy może przejechać po 1000 km; chcemy wysłać 1000 ciężarówek w trasę 1 km;*

`3ocen`: *500 000 kierowców mogących przejechać po 1 km; wysyłamy jedną ciężarówkę w trasę 500 000 km.*

Rozwiązanie

Jesteśmy proszeni o wielokrotne modyfikowanie bazy uprawnień kierowców i odpowiadanie na zapytania o zdolność wykonania zleceń firmy Bajtazara. W zadaniach tego typu zazwyczaj opłaca się poświęcić trochę czasu na opracowanie struktury danych, która nie będzie wymagała obliczania wszystkiego od nowa po każdej drobnej zmianie (doświadczeni Czytelnicy już w tym momencie pomyślą o strukturach drzewiastych). Zanim jednak zaczniemy projektowanie struktury danych, sprowadźmy zadawane nam pytanie do możliwie prostego warunku.

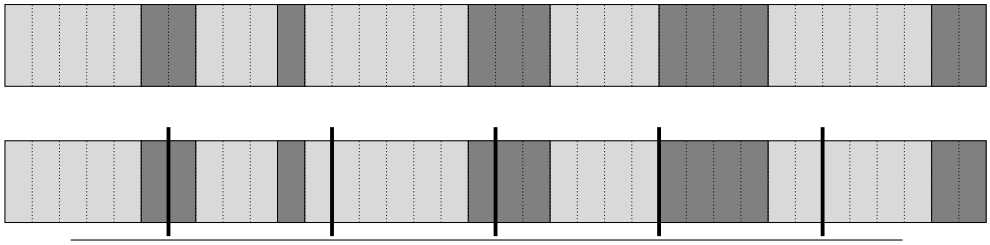
Warunek dla pojedynczego zlecenia

Oznaczmy przez $(a_i)_{i=1}^n$ ciąg ograniczeń na długość przejechanej trasy kierowców. Dopóki i-ty kierowca nie posiada żadnego wpisu w swoim prawie jazdy, przyjmujemy $a_i = 0$. Musimy odpowiedzieć na pytanie, czy taki zespół jest w stanie przejechać s kilometrów konwojem złożonym z c ciężarówek.

Lemat 1. Zespół kierowców opisany ciągiem $(a_i)_{i=1}^n$ może wykonać zlecenie na s kilometrów i c ciężarówek wtedy i tylko wtedy, gdy

$$\sum_{i=1}^{n} \min(a_i, s) \geqslant sc. \tag{1}$$

Dowód: Łatwo udowodnić, że powyższy warunek jest konieczny do powodzenia podróży. Sumaryczna liczba kilometrów do przejechania wynosi sc, zaś i-ty kierowca może przejechać w trasie maksymalnie $\min(a_i, s)$ kilometrów.



Rys. 1: Ilustracja do dowodu lematu 1. W przykładzie $s = 6$, $c = 6$, zaś elementy ciągu a to $(5, 2, 3, 1, 10, 3, 4, 4, 20, 5)$.

Wykorzystamy geometryczną interpretację wzoru (1), aby pokazać, że przestawiony warunek jest także wystarczający. Naszkicujmy pasek (patrz rysunek 1) o długości sc jednostek i przypiszmy początkowe $\min(a_1, s)$ jednostek z lewej strony pierwszemu kierowcy. Kolejne $\min(a_2, s)$ jednostek przyporządkujmy drugiemu kierowcy, po czym kontynuujmy ten proces, aż cały pasek będzie zajęty – nierówność (1) gwarantuje nam, że uda się go zapełnić do końca.

Następnie podzielmy pasek na c równych części, odpowiadających poszczególnym ciężarówkom. Zauważmy, że czytając i-tą część paska od lewej strony do prawej, dostajemy pewien plan prowadzenia i-tej ciężarówki, czyli informację, który kierowca będzie ją prowadzić na poszczególnych kilometrach trasy. Musimy jedynie się upewnić, że jeden kierowca nie będzie przypisany w tym samym momencie do dwóch różnych ciężarówek. Na szczęście taka sytuacja jest niemożliwa, ponieważ gdyby odcinek zawierał segmenty paska odpowiadające temu samemu kilometrowi trasy dla różnych ciężarówek, to musiałby mieć długość co najmniej $s + 1$ jednostek, podczas gdy każdemu kierowcy przyporządkowaliśmy spójny odcinek o długości nie większej niż s jednostek. ■

Bezpośrednie sprawdzanie warunku (1) dla każdego zlecenia prowadzi do algorytmu o złożoności czasowej $O(nm)$, który działa wystarczająco szybko na testach spełniających warunek $n, m \leqslant 1000$, wartych w sumie 33 punkty. Implementacja takiego rozwiązania znajduje się w pliku `logs0.cpp`.

Rozwiązanie wzorcowe

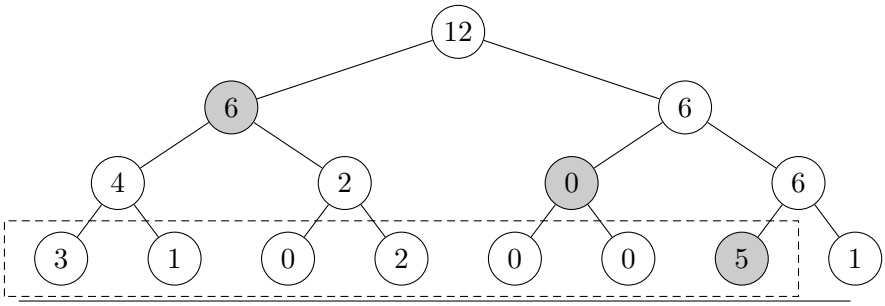
Oznaczmy przez w_i liczbę kierowców, którzy mogą przejechać za kierownicą dokładnie i kilometrów, oraz niech $x_i = iw_i$. Przekształćmy sumę z warunku (1) do nieco wygodniejszej postaci:

$$\sum_{i=1}^{n} \min(a_i, s) = \sum_{a_i \leqslant s} a_i + \sum_{a_i > s} s = \sum_{i \leqslant s} x_i + s \sum_{i > s} w_i. \tag{2}$$

Kiedy dowiadujemy się o zmianie wpisu w uprawnieniach kierowcy z a na b kilometrów, wystarczy wprowadzić następujące modyfikacje:

1. $w_a := w_a - 1$,
2. $x_a := x_a - a$,
3. $w_b := w_b + 1$,
4. $x_b := x_b + b$.

Musimy teraz tylko umieć dynamicznie wyznaczać sumy częściowe ciągów w_i oraz x_i. Standardową strukturą danych pomocną w takiej sytuacji jest *drzewo przedziałowe*[1]. Aby obliczyć sumę dla dowolnego przedziału, rozbijamy go na $O(\log n)$ przedziałów bazowych (patrz rysunek 2), dla których będziemy pamiętać obliczone sumy. Podobnie zmiana pojedynczej wartości ciągu będzie wymagać $O(\log n)$ operacji.



Rys. 2: Przykład działania drzewa przedziałowego dla ciągu długości 8. Na szaro zaznaczono rozbicie przedziału $[1 : 7]$ na przedziały bazowe $[1 : 4]$, $[5 : 6]$, $[7]$.

Według treści zadania, w testach wartych 66% punktów zachodzi dodatkowy warunek $a, s \leqslant 10^6$. Oczywiście w_i, x_i będą równe zero dla i powyżej maksymalnej wartości a, zatem przy takim ograniczeniu moglibyśmy przechowywać oba ciągi dla $i = 1, \ldots, 10^6$. Przykład takiego rozwiązania znajdziemy w pliku `logb0.cpp`. Jeśli jednak chcemy zdobyć pełne 100 punktów za zadanie, musi nam wystarczyć słabsze założenie $a, s \leqslant 10^9$.

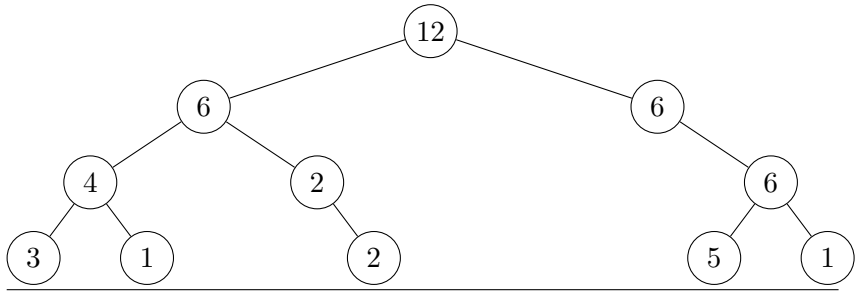
Dynamiczne drzewo przedziałowe

Limit pamięci uniemożliwia nam alokację nawet pojedynczej tablicy długości 10^9. Zauważmy jednak, że liczba wszystkich wpisów w dokumentach kierowców jest ograniczona przez m, która to wielkość jest nie większa niż 10^6. W takim razie liczba wszystkich i, dla których w_i, x_i będzie niezerowe, jest stosunkowo niewielka i możemy ograniczyć się do pamiętania tylko takich wyrazów ciągów.

W najprostszej implementacji drzewa przedziałowego, najpierw budujemy drzewo dla ciągu o ustalonej długości, a następnie modyfikujemy tylko wartości pamiętane w węzłach drzewa. Nazywamy to **statycznym** drzewem przedziałowym. Nieco większe możliwości daje **dynamiczna** wersja tej struktury danych. Jeśli zrezygnujemy

[1]Dokładny opis tej struktury pojawił się w opracowaniach zadań *Tetris 3D* z XIII Olimpiady Informatycznej [13] oraz *Koleje* z IX Olimpiady Informatycznej [9], a także w *Wykładach z Algorytmiki Stosowanej*, `http://was.zaa.mimuw.edu.pl`.

z wygodnej reprezentacji tablicowej drzewa, możemy dynamicznie dodawać węzły, dopiero kiedy będziemy ich potrzebować. Wysokość takiego drzewa nigdy nie przekroczy $\lceil \log_2(10^9) \rceil = 30$, zatem wszystkie operacje dalej będziemy wykonywać wystarczająco szybko.



Rys. 3: Przykład działania dynamicznego drzewa przedziałowego dla tych samych danych co na rysunku 2.

Pozostaje pytanie, czy zmieścimy się w dostępnym limicie pamięci. Ponieważ maksymalna liczba liści drzewa wynosi $m = 10^6$, a maksymalna wysokość drzewa wynosi 30, to zgrubne szacowanie liczby węzłów daje $30m$ (na każdym poziomie drzewa bierzemy maksymalną liczbę liści). Spróbujmy jednak znaleźć lepsze oszacowanie.

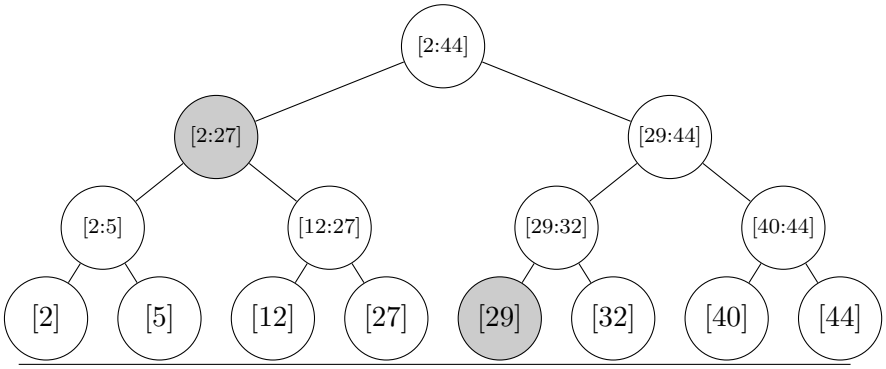
Maksymalna liczba węzłów znajdujących się w drzewie do głębokości 20 to $2^{21} - 1$. Na wszystkich niższych poziomach drzewa (których jest dokładnie 10) każdy liść daje maksymalnie 10 węzłów. Takie oszacowanie daje w sumie nie więcej niż $2^{21} + 10m \leqslant 1{,}21 \cdot 10^7$ węzłów. W pojedynczym węźle przechowujemy sumę podciągu (w_i) (zmienna typu `int`, czyli 4 bajty), sumę podciągu (x_i) (zmienna typu `long long`, czyli 8 bajtów) oraz wskaźniki do lewego i prawego syna (w sumie 8 bajtów). Cała struktura zajmie nie więcej niż $1{,}21 \cdot 10^7 \cdot 20B \leqslant 242MB$,[2] więc zostało nam jeszcze $14MB$ na pozostałe zmienne.

Powyższe rozwiązanie można znaleźć w pliku `log1.cpp`. Jego złożoność czasowa wynosi $O(n + m \log a_{max})$. Algorytm jest wystarczająco szybki, by przejść wszystkie przygotowane testy. Jednakże dynamiczne drzewo przedziałowe okazuje się nieprzyjemne w implementacji, zwłaszcza jeśli dysponujemy ograniczonym czasem na zawodach. Dlatego zanim zasiądziemy do klawiatury, warto pomyśleć, czy nie istnieje prostsze rozwiązanie.

Rozwiązanie offline

Okazuje się, że wystarczy nam statyczne drzewo przedziałowe, o ile dopuścimy się drobnego „oszustwa". Pisząc prawdziwe oprogramowanie, musielibyśmy być przygotowani na obsłużenie dowolnych danych, jednak w zadaniu możemy na początku podejrzeć przyszłość i sprawdzić, jakie wartości a_i pojawią się w całej historii zapytań. Następnie zbiór takich wartości możemy posortować i zbudować na nim od razu skompresowane drzewo przedziałowe (patrz rysunek 4).

[2] Lub $231MB$, jeśli zakładamy, że $1MB = 1024 \cdot 1024B$.



Rys. 4: W przykładzie pojawia się 8 wartości a_i. W każdym węźle zapisano kontrolowany przez niego przedział. Na szaro zaznaczono przedziały bazowe dla zapytania o przedział $[1 : 30]$.

Tym razem otrzymujemy złożoność czasową $O(n + m \log m)$ oraz pamięciową $O(n+m)$. Implementację takiego rozwiązania można znaleźć w plikach `log.cpp` oraz `log3.pas`.

Testy

Podstawowy rodzaj testów został zaprojektowany z myślą o następujących założeniach:

- podobny rozkład zapytań typu `U` i `Z`,
- podobny rozkład odpowiedzi `TAK` i `NIE`,
- wykrywanie rozwiązań, które w zapytaniach typu `Z` jedną z sum ze wzoru (2) obliczają siłowo,
- wielokrotne odwoływanie się do tego samego kierowcy, aby wymusić zmianę wpisu,
- wielokrotne przypisywanie tego samego wpisu różnym kierowcom, aby w ciągu (w_i) pojawiały się liczby większe od 1.

Pozostałe testy należały do jednej z poniższych grup:

1. test zupełnie losowy,
2. jeśli odpowiedź to `TAK`, to w każdej ciężarówce pojedzie dwóch kierowców,
3. najpierw dodawane są dodatnie wpisy, potem wpisy są znowu zerowane,
4. każdy kierowca może przejechać co najwyżej 1 kilometr.