Bartłomiej Dudek
Treść zadania, Opracowanie

Michał Adamczyk
Program

Dostępna pamięć: 128 MB.

OI, etap II, dzień drugi, 14.02.2013

Konewka

Zostałeś ogrodnikiem u królowej Bajtoliny. Wspaniale, prawda? Skoro tak uważasz, to chyba jeszcze nie wiesz wszystkiego o tej pracy. Obok zamku królowej znajduje się wielki ogród z n drzewami ustawionymi po kolei jedno za drugim. To jeszcze nic strasznego, ale czy potrafisz o każdej porze dnia i nocy odpowiedzieć swojej władczyni, które z jej drzewek są teraz dojrzałe? Zakładamy, że drzewo jest dojrzałe, gdy ma przynajmniej k bajtymetrów wysokości.

Czasem królowa prosi Cię, abyś niektóre z jej drzewek podlał za pomocą magicznej konewki. Każda taka operacja powoduje, że wszystkie podlane drzewa rosną o dokładnie jeden bajtymetr.

Udowodnij, że nadajesz się do tej pracy i szybko odpowiedz na wszystkie pytania królowej!

Komunikacja

Napisz bibliotekę komunikującą się z programem oceniającym. Powinna ona zawierać przynajmniej następujące trzy funkcje, wywoływane przez program oceniający:

- `inicjuj(n, k, D)` – *ta funkcja zostanie wywołana dokładnie raz, na początku sprawdzania. Możesz ją wykorzystać do inicjalizacji swoich struktur danych. Przyjmuje jako parametry liczbę drzew n, dolne ograniczenie wysokości dojrzałego drzewa k oraz tablicę D o długości n zawierającą początkowe wysokości wszystkich drzew. Drzewa są ponumerowane kolejnymi liczbami całkowitymi od 0 do $n-1$.*
  - **C/C++:** `void inicjuj(int n, int k, int *D);`
  - **Pascal:** `procedure inicjuj(n, k : LongInt; var D : array of LongInt);`
- `podlej(a, b)` – *oznacza, że królowa poprosiła Cię o podlanie wszystkich drzew od a-tego do b-tego włącznie ($0 \leqslant a \leqslant b \leqslant n-1$). Wywołanie tej funkcji oznacza, że każde z tych drzew rośnie o 1 bajtymetr.*
  - **C/C++:** `void podlej(int a, int b);`
  - **Pascal:** `procedure podlej(a, b : LongInt);`
- `dojrzale(a, b)` – *królowa pyta Cię, ile spośród drzew o numerach od a do b włącznie ($0 \leqslant a \leqslant b \leqslant n-1$) jest już dojrzałych.*
  - **C/C++:** `int dojrzale(int a, int b);`
  - **Pascal:** `function dojrzale(a, b : LongInt) : LongInt;`

Twoja biblioteka **nie może** *czytać żadnych danych (ani ze standardowego wejścia, ani z plików).* **Nie może** *również nic wypisywać do plików ani na standardowe wyjście. Może pisać na standardowe wyjście diagnostyczne (*`stderr`*) – pamiętaj jednak, że zużywa to cenny czas.*

Jeśli piszesz w C/C++, Twoja biblioteka **nie może** *zawierać funkcji* `main`*. Jeśli piszesz w Pascalu, powinieneś dostarczyć moduł (patrz przykładowe programy na Twoim dysku).*

Twoje rozwiązanie uzyska punkty za dany test tylko wtedy, gdy spełni określone powyżej wymogi techniczne oraz odpowie poprawnie na wszystkie pytania królowej.

Ograniczenia

We wszystkich testach zachodzą następujące warunki:

- $1 \leqslant n \leqslant 300\,000$,
- $1 \leqslant k \leqslant 10^9$,
- *łączna liczba wywołań funkcji* `podlej` *i funkcji* `dojrzale` *nie przekracza 300 000*,
- *początkowe wysokości wszystkich drzew to dodatnie liczby całkowite nieprzekraczające* 10^9.

W testach wartych łącznie 20% punktów zachodzą dodatkowe warunki:

- $n \leqslant 2000$,
- *łączna liczba wywołań funkcji* `podlej` *i funkcji* `dojrzale` *nie przekracza 10 000*.

W testach wartych łącznie 50% punktów wszystkie wywołania funkcji `podlej` *następują przed wszystkimi wywołaniami funkcji* `dojrzale`.

Kompilacja

Twoja biblioteka – `kon.c`, `kon.cpp` *lub* `kon.pas` *– zostanie skompilowana z programem oceniającym przy użyciu następujących poleceń:*

- **C:** `gcc -O2 -static -lm kon.c kongrader.c -o kon`
- **C++:** `g++ -O2 -static -lm kon.cpp kongrader.cpp -o kon`
- **Pascal:**
```
ppc386 -O2 -XS -Xt kon.pas
ppc386 -O2 -XS -Xt kongrader.pas
mv kongrader kon
```

Przykładowe wykonanie

Poniższa tabela zawiera przykładowy ciąg wywołań funkcji oraz poprawne wyniki funkcji `dojrzale`.

Wywołanie	Wynik	Wyjaśnienie
`inicjuj(4, 6, D)` (`D[0]`=5, `D[1]`=4, `D[2]`=3, `D[3]`=7)		*Mamy $n = 4$ drzew o wysokościach 5, 4, 3 i 7, a $k = 6$.*
`dojrzale(2, 3)`	*1*	*Ile dojrzałych drzew jest na przedziale $[2, 3]$?*
`podlej(0, 2)`		*Podlej drzewa 0, 1 i 2.*
`dojrzale(1, 2)`	*0*	*Ile dojrzałych drzew jest na przedziale $[1, 2]$?*
`podlej(2, 3)`		*Podlej drzewa 2 i 3.*
`podlej(0, 1)`		*Podlej drzewa 0 i 1.*
`dojrzale(0, 3)`	*3*	*Ile dojrzałych drzew jest na przedziale $[0, 3]$?*

Testowanie

W katalogu `/home/zawodnik/kon/` *na dysku Twojego komputera możesz znaleźć przykładowy program oceniający (*`kongrader.c`*,* `kongrader.cpp` *i* `kongrader.pas`*). Żeby uruchomić program oceniający z Twoją biblioteką, powinieneś umieścić ją w pliku* `kon.c`*,* `kon.cpp` *lub* `kon.pas` *w odpowiednim katalogu (*`c`*,* `cpp` *lub* `pas`*). Na początku zawodów znajdziesz tam przykładowe, błędne rozwiązania tego zadania. Program oceniający wraz z Twoją biblioteką możesz skompilować za pomocą polecenia:*

```
make kon
```

które działa dokładnie tak, jak opisano w sekcji „Kompilacja". Kompilacja rozwiązania w C/C++ wymaga pliku `koninc.h`*, który również znajduje się w odpowiednich katalogach.*

Tak skompilowany program oceniający wczytuje ze standardowego wejścia specjalnie przygotowany opis testu, wywołuje odpowiednie funkcje Twojej biblioteki, a wyniki wypisuje na standardowe wyjście.

Opis testu powinien być w następującym formacie. Pierwszy wiersz testu składa się z dwóch liczb całkowitych n i k. Drugi wiersz zawiera n liczb oznaczających wysokości początkowe kolejnych drzew. Trzeci wiersz zawiera liczbę q. Dalej następuje q wierszy, z których każdy zawiera pojedynczą literę `p` *lub* `d` *oraz dwie nieujemne liczby całkowite. Litera określa, która funkcja powinna być wywołana:* `p` *dla* `podlej` *i* `d` *dla* `dojrzale`*, zaś liczby – z jakimi argumentami należy tę funkcję wywołać. Funkcja* `inicjuj` *zostanie wywołana od razu po wczytaniu pierwszych dwóch wierszy. Pamiętaj jednak, że przykładowy program oceniający nie sprawdza, czy dane są sformatowane poprawnie ani czy spełnione są ograniczenia podane w sekcji „Ograniczenia".*

W katalogu `/home/zawodnik/kon/` *znajdziesz plik* `kon0.in`*, który odpowiada przykładowemu wykonaniu programu opisanemu powyżej. Aby uruchomić program oceniający na podanym przykładzie, użyj następującego polecenia:*

```
./kon < kon0.in
```

Wyniki wywołań funkcji `dojrzale` *zostaną wypisane na standardowe wyjście. Poprawny wynik dla powyższego przykładowego wykonania znajdziesz na dysku w pliku* `kon0.out`*.*

Aby sprawdzić, czy wynik wypisany przez Twoje rozwiązanie jest prawidłowy, możesz również wysłać rozwiązanie (Twoją bibliotekę) do SIO.

Testy „ocen":

`1ocen`**:** *$n = 2000$, łączna liczba wywołań funkcji* `podlej` *i* `dojrzale` *wynosi 10 000, wszystkie te wywołania dotyczą pojedynczych drzew, początkowe wysokości drzew losowe z przedziału $[1, 100]$, $k = 100$;*

`2ocen`**:** *$n = 100\,000$, najpierw 100 000 wywołań funkcji* `podlej`*, dotyczące wszystkich drzew, potem 100 000 wywołań funkcji* `dojrzale` *o pojedyncze drzewa, początkowe wysokości drzew losowe z przedziału $[1, 1000]$, $k = 100\,500$.*

Rozwiązanie

Zanim przejdziemy do rozwiązania, opiszmy problem bardziej formalnym językiem. Dany jest ciąg liczb całkowitych długości n oraz q możliwych operacji, z których każda

jest typu `podlej` lub `dojrzale`. Dana jest także pewna stała całkowita k. Operacja `podlej`(a, b) powoduje zwiększenie o 1 wszystkich wyrazów ciągu na przedziale indeksów od a do b włącznie. Natomiast operacja `dojrzale`(a, b) jest pytaniem o liczbę wyrazów ciągu, o indeksach z przedziału od a do b włącznie, które są nie mniejsze niż k.

Zadaniem zawodnika jest przygotowanie biblioteki udostępniającej programowi oceniającemu trzy funkcje: `inicjuj`, która pozwala na przekazanie programowi zawodnika początkowych wartości, oraz `podlej` i `dojrzale` o działaniu opisanym powyżej. Zawodnik musi odpowiadać na zapytania `podlej` natychmiast, jeszcze przed poznaniem następnej operacji.

Rozwiązanie wolne

Od razu po zrozumieniu treści jesteśmy w stanie napisać rozwiązanie siłowe, które będzie wykonywało dokładnie to, co jest powiedziane w zadaniu. Dla każdego wywołania funkcji `podlej` rozwiązanie iteruje po wszystkich drzewach na przedziale $[a, b]$, zwiększając ich wysokość o jeden. Podobnie dla wywołań `dojrzale` iteruje po wszystkich drzewach na przedziale $[a, b]$, zliczając te o wysokości nie mniejszej niż k.

W pesymistycznym przypadku dla każdego z q zapytań nasze rozwiązanie będzie musiało przejrzeć wszystkie n drzew, zatem złożoność tego rozwiązania to $O(q \cdot n)$. Za takie rozwiązanie można było uzyskać 20 punktów. Implementacja tego rozwiązania znajduje się w plikach `kons1.cpp` i `kons2.pas`.

Rozwiązanie wzorcowe

Rozwiązanie wzorcowe opiera się na spostrzeżeniu, że każde drzewo tylko raz może stać się dojrzałe, a dojrzałe drzewo pozostanie dojrzałym na zawsze.

Rozważmy wszystkie drzewa, które są w danej chwili dojrzałe, i umieśćmy je w pewnej strukturze danych (nazwijmy ją A), udostępniającej dwie operacje:

- *WyznaczNaPrzedziale*(A, a, b) – wyznaczenie liczby drzew w strukturze znajdujących się na pozycjach z przedziału $[a, b]$; dzięki tej operacji wprost zaimplementujemy funkcję `dojrzale`.
- *UstawNaPozycji*(A, poz) – wstawienie jednego drzewa na pozycję poz.

W drugiej strukturze (nazwijmy ją B) przechowywać będziemy tylko drzewa niedojrzałe. Kiedy dzieje się cokolwiek z drzewami niedojrzałymi? Oczywiście tylko przy wywołaniu funkcji `podlej`, kiedy to wszystkie drzewa rosną o jeden bajtymetr i być może niektóre z nich stają się wtedy dojrzałe. Chcemy, aby po każdej takiej operacji nasze struktury były aktualne, tzn. musimy znaleźć w strukturze B wszystkie drzewa, które właśnie stały się dojrzałe, i przenieść je do struktury A.

Do efektywnego przechowywania i modyfikowania informacji o wysokościach niedojrzałych drzew będziemy potrzebowali struktury, która umożliwia szybkie wykonanie następujących operacji:

- *ZwiekszNaPrzedziale*(B, a, b) – zwiększenie o 1 wysokości wszystkich drzew na przedziale $[a, b]$;
- *PozycjaNajwyzszego*(B, a, b) oraz *WysokoscNaPozycji*(B, poz) – wyznaczenie pozycji najwyższego drzewa na przedziale $[a, b]$ i zwrócenie wysokości tego drzewa;
- *UstawNaPozycji*(B, poz, wys) – dowolna modyfikacja wysokości jednego drzewa na pozycji *poz*.

Funkcja `podlej` może teraz wyglądać następująco:

```
1:  function podlej(a, b)
2:  begin
3:    ZwiekszNaPrzedziale(B, a, b);
4:    while true do begin
5:      poz := PozycjaNajwyzszego(B, a, b);
6:      wys := WysokoscNaPozycji(B, poz);
7:      if wys < k then break;
8:      { Będziemy przenosić drzewo z pozycji poz }
9:      UstawNaPozycji(B, poz, −∞);
10:     UstawNaPozycji(A, poz);
11:   end
12: end
```

Wybór struktur danych

Zauważmy, że **drzewo przedziałowe** (opisane dokładnie w rozwiązaniach zadań *Tetris 3D* z XIII OI [13] oraz *Koleje* z IX OI [9], a także na *Wykładach z Algorytmiki Stosowanej* `http://was.zaa.mimuw.edu.pl`) będzie doskonale spisywać się w przypadku zarówno struktury A, jak i struktury B. Wszystkie żądane operacje na odpowiednio zaimplementowanych drzewach przedziałowych będziemy mogli zrealizować w złożoności czasowej $O(\log n)$.

Uściślając: strukturę danych A można zaimplementować jako drzewo przedziałowe typu punkt–przedział (czyli wstawiamy informację w pewnym punkcie, pytamy o liczbę punktów na przedziale), a strukturę B jako drzewo przedziałowe typu przedział–przedział z wyszczególnioną parą operacji $(+, \max)$ (zwiększamy każdą liczbę na przedziale o jeden, pytamy o maksimum na przedziale) oraz dodatkową operacją umożliwiającą ustawienie dowolnej wartości dla pewnej konkretnej pozycji.

Jako strukturę danych A możemy również wykorzystać **drzewo potęgowe**, opisane w czasopiśmie *Delta* w numerze 10/2008. Ta struktura również umożliwi nam wykonanie wszystkich potrzebnych operacji w czasie $O(\log n)$.

Analiza złożoności rozwiązania wzorcowego

Dzięki temu, że odpowiedź na każde z zapytań `dojrzale` sprowadza się do wywołania jednej operacji na drzewie przedziałowym, takie zapytanie możemy przetworzyć w czasie $O(\log n)$.

Jeśli chodzi o zapytania `podlej`, to mogłoby się wydawać, że skoro dla każdego z nich możemy przejrzeć nawet n drzew, to złożoność całego rozwiązania będzie rzędu $O(n \cdot q \cdot \log n)$. Na szczęście możemy lepiej oszacować pracę związaną z tymi zapytaniami.

Skoro każde drzewo co najwyżej raz może zmienić swój stan z niedojrzałego na dojrzałe, to co najwyżej raz będziemy je usuwać ze struktury B i dodawać do struktury A. Z tego powodu, podczas działania programu wykonamy $O(n)$ operacji przeniesienia pojedynczego drzewa. Każdą z tych operacji wykonujemy w czasie $O(\log n)$, więc koszt poprawienia stanu obu struktur po wywołaniu `podlej` zamortyzuje się do $O(\log n)$, a poprawień takich możemy wykonać co najwyżej n. Ostatecznie, złożoność czasowa całego rozwiązania wyniesie $O((q+n)\log n)$. Z kolei złożoność pamięciowa wyniesie $O(n)$.

Uwagi dotyczące poszczególnych implementacji

Zauważmy, że jeśli w naszym algorytmie będziemy przenosić dojrzałe drzewa pomiędzy strukturami A i B nie po wykonaniu funkcji `podlej`, ale przed wyznaczeniem wyniku przy zapytaniu `dojrzale`, to cała wcześniejsza analiza będzie dalej poprawna. Rozwiązania wzorcowe różnią się między sobą implementacją pierwszej struktury danych oraz momentem i sposobem przenoszenia drzew:

- Rozwiązania `kon.c`, `kon1.cpp` i `kon2.pas` przenoszą wszystkie dojrzałe drzewa z przedziału $[a, b]$ na początku wykonania funkcji `dojrzale`. Rozwiązanie `kon3.c` przenosi zaś wszystkie dojrzałe drzewa z przedziału $[a, b]$ na końcu wykonania funkcji `podlej`.
- Rozwiązania `kon.c` oraz `kon1.cpp` różnią się między sobą sposobem przenoszenia drzew dojrzałych ze struktury B do struktury A. Rozwiązanie `kon.cpp` wywołuje funkcję *ZnajdzDojrzale*, która dla przedziału, w którym maksimum jest nie mniejsze niż k, dzieli przedział na dwie równe części i wywołuje się rekurencyjnie (chyba, że przedział jest wielkości 1, wtedy drzewo o pozycji w tym przedziale przenoszone jest do pierwszej struktury danych). Rozwiązanie `kon1.cpp` realizuje zaś *stricte* zaprezentowany wyżej schemat rozwiązania, tj. dopóki najwyższe drzewo na przedziale jest ma wysokość nie mniejszą niż k, przenosi to drzewo, po czym sprawdza wysokość najwyższego drzewa pozostałego po tym przeniesieniu.
- Rozwiązanie `kon2.pas` implementuje strukturę A za pomocą drzewa potęgowego.

Wszystkie powyższe rozwiązania zdobywały na zawodach maksymalną punktację.

Rozwiązanie na połowę punktów

Rozwiązanie to zakłada, że wszystkie wywołania funkcji `podlej` nastąpią przed wywołaniami `dojrzale`. W takim wypadku można wczytać wszystkie przedziały, z jakimi wywoływana jest funkcja `podlej`, a przy pierwszym wywołaniu `dojrzale` obliczyć, dla każdego drzewa, o ile sumarycznie wcześniej urosło.

Aby to zrobić, przy pierwszym wywołaniu funkcji `dojrzale` przeglądamy wszystkie przedziały i w pomocniczej tablicy zaznaczamy, ile przedziałów w danym miejscu zaczyna się lub kończy. Teraz, przeglądając wszystkie drzewa po kolei od 0 do $n-1$, możemy jednocześnie łatwo wyznaczyć liczbę „otwartych" przedziałów, czyli takich, które obejmują swoim zakresem to drzewo. Znaleziona liczba będzie odpowiadała liczbie bajtymetrów, o jaką dane drzewo urosło, i na jej podstawie możemy powiedzieć, czy jest ono dojrzałe, czy nie.

Następnie w osobnej tablicy możemy zapamiętać dla każdego drzewa i, ile jest drzew dojrzałych spośród tych o numerach od 0 do i. Mając taką tablicę, można w czasie stałym odpowiadać na każde zapytanie o liczbę drzew dojrzałych na przedziale $[a, b]$.

Rozwiązanie to zostało zaimplementowane w pliku `konb1.cpp`. Zgodnie z treścią zadania można było za nie uzyskać 50 punktów.

Rozwiązania niepoprawne

W plikach `konb2.cpp`, `konb3.cpp`, `konb4.cpp`, `konb6.cpp` są przykłady rozwiązań, w których niepoprawnie zaimplementowano struktury danych przechowujące drzewa. Rozwiązania te dostawały 0 punktów.

Rozwiązanie `konb5.cpp` to rozwiązanie prawie wzorcowe – błąd polega na przyjęciu zbyt małej wartości nieskończoności, co dla niektórych testów powoduje wielokrotne zliczenie tego samego dojrzałego drzewa. Rozwiązanie to otrzymuje 60% maksymalnej punktacji. Należało pamiętać, że w trakcie całego programu wysokość drzewa może wzrosnąć o q, tak więc za $-\infty$ trzeba przyjąć liczbę mniejszą niż $k-q$.

Po co biblioteka?

Uważny Czytelnik na pewno zadaje sobie teraz pytanie, po co w tym zadaniu zostało wymuszone odpowiadanie „on-line" na zapytania, skoro nie widać łatwego rozwiązania wersji „off-line". Okazuje się, że „off-line" można rozwiązać nawet trudniejszy problem, w którym dodatkowo dla każdego drzewa jest określona wysokość h_i, od której dane drzewo jest uznawane za dojrzałe. Tę trudniejszą wersję da się rozwiązać w niewiele gorszej złożoności czasowej $O((q+n)\cdot \log n \cdot \log H)$, gdzie H to największa dopuszczalna wysokość drzew. Zachęcamy Czytelnika do rozwiązania tego problemu. Niestety, autorzy nie mogli dopuścić do wykorzystania takiego rozwiązania, gdyż z tą techniką uczestnicy Olimpiady zetknęli się już na finale XVIII OI w zadaniu pt. *Meteory* [18].

Testy

Każda grupa testów składa się z trzech testów, z czego pierwsze dwa są w całości losowe, a trzeci nie. W pierwszym z losowych testów w każdej grupie prawdopodobieństwo wystąpienia wywołania `podlej` było takie same, jak wywołania `dojrzale`. Testy w grupach o parzystych numerach zawierają wszystkie wywołania funkcji `podlej` przed pierwszym wywołaniem funkcji `dojrzale`.