Adam Karczmarz — Treść zadania, Opracowanie
Mateusz Baranowski — Program

Dostępna pamięć: 128 MB. *OI, etap III, dzień pierwszy, 28.03.2012*

Pensje

Bajtocka Fabryka Oprogramowania (BFO) zatrudnia n pracowników. W hierarchii pracowników każdy ma swojego bezpośredniego przełożonego, z wyjątkiem dyrektora BFO, któremu (pośrednio lub bezpośrednio) podlegają wszyscy pracownicy BFO. Pracownicy mają ustalone miesięczne pensje, przy czym każdy pracownik zarabia inną kwotę, od 1 do n bajtalarów. Każdy przełożony zarabia więcej niż każdy z jego podwładnych.

Zgodnie z bajtockim prawem, pensje pracowników na pewnych stanowiskach mogą być publicznie znane. Ponadto, jeśli pensja pewnego pracownika jest publicznie znana, to pensja jego przełożonego także jest znana.

Bajtocki Urząd Podatkowo–Antykorupcyjny (BUPA) postanowił prześwietlić BFO. Zanim BUPA wkroczy z kontrolą do BFO, chce najpierw poznać wysokość pensji wszystkich pracowników BFO, których pensje nie są publicznie znane, ale których wysokość wynika jednoznacznie z publicznie znanych wysokości pensji innych pracowników BFO.

Wejście

W pierwszym wierszu standardowego wejścia podana jest jedna liczba całkowita n ($1 \leqslant n \leqslant 1\,000\,000$) oznaczająca liczbę pracowników BFO. Pracownicy są ponumerowani od 1 do n.

Kolejne n wierszy zawiera informacje o pracownikach. Wiersz o numerze $i+1$ opisuje pracownika numer i za pomocą dwóch liczb całkowitych p_i i z_i ($1 \leqslant p_i \leqslant n$, $0 \leqslant z_i \leqslant n$) oddzielonych pojedynczym odstępem. Liczba p_i to numer bezpośredniego przełożonego pracownika numer i. Jeżeli $p_i = i$, to i jest numerem dyrektora BFO. Jeśli $z_i > 0$, to jest to wysokość pensji pracownika numer i. Jeśli zaś $z_i = 0$, to wysokość pensji pracownika numer i nie jest publicznie znana. Dodatnie liczby z_i są parami różne.

Dane wejściowe będą tak skonstruowane, że będzie istniał co najmniej jeden sposób przypisania płac pracownikom zgodny z ich hierarchią.

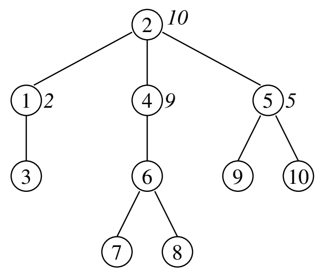
W testach wartych łącznie 54% punktów zachodzi dodatkowy warunek $n \leqslant 10\,000$.

Wyjście

Twój program powinien wypisać na standardowe wyjście n wierszy, z których każdy powinien zawierać jedną liczbę całkowitą. Jeżeli pensja pracownika numer i jest jawna lub może zostać wywnioskowana na podstawie znanych publicznie pensji innych pracowników, to i-ty wiersz wyjścia powinien zawierać pensję pracownika numer i. W przeciwnym przypadku i-ty wiersz wyjścia powinien zawierać 0.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`10`	`2`
`2 2`	`10`
`2 10`	`1`
`1 0`	`9`
`2 9`	`5`
`2 5`	`8`
`4 0`	`0`
`6 0`	`0`
`6 0`	`0`
`5 0`	`0`
`5 0`	



Wyjaśnienie do przykładu: *Na rysunku liczby w kółkach to numery pracowników, a liczby zapisane kursywą to publicznie znane pensje pracowników. Pracownik numer 3 musi zarabiać 1 bajtalar, a pracownik numer 6 musi zarabiać 8 bajtalarów. Zarobki pracowników numer 7, 8, 9 i 10 nie wynikają jednoznacznie z publicznie znanych zarobków innych pracowników.*

Rozwiązanie

Wprowadzenie

Aby uprościć nasze rozważania, sformułujmy problem z zadania w języku matematyki. Struktura Bajtockiej Fabryki Oprogramowania tworzy drzewo ukorzenione, którego wierzchołki odpowiadają pracownikom BFO. Dokładniej, korzeń drzewa odpowiada dyrektorowi BFO, a ojcowie pozostałych wierzchołków odpowiadają ich bezpośrednim przełożonym. Przypisanie pensji pracownikom to nic innego jak wpisanie w każdy z wierzchołków innego klucza z zakresu od 1 do n tak, aby spełniony był *warunek kopca*. Warunek kopca jest spełniony wtedy i tylko wtedy, gdy dla każdego wierzchołka, jego klucz jest mniejszy niż klucz jego ojca.

Wyobraźmy sobie, że po przypisaniu kluczy wszystkim wierzchołkom wymazaliśmy klucze w niektórych poddrzewach. Naszym zadaniem jest wskazanie wierzchołków, dla których jesteśmy w stanie stwierdzić z całą pewnością, jaki klucz był im pierwotnie przypisany.

Analiza

Dla danego drzewa T, oznaczmy liczbę jego wierzchołków przez $|T|$. Poddrzewo o korzeniu v zawierające wszystkich potomków v oznaczać będziemy przez $T(v)$.

Na wstępie zauważmy, że dokładna struktura „górnej” części drzewa, z której nie zostały wymazane klucze, nie ma większego znaczenia dla wyniku, który chcemy uzyskać. Mianowicie, na to, jak duże mogą być klucze w wierzchołkach danego poddrzewa, ma wpływ jedynie klucz ojca tego poddrzewa. To spostrzeżenie pozwala nam przeformułować zadanie w następujący sposób. Mamy dane t drzew $T_1, T_2, \ldots, T_t$, w których klucze wierzchołków muszą być mniejsze niż, odpowiednio, $a_1, a_2, \ldots, a_t$, a także zbiór K kluczy, którymi chcemy wypełnić te drzewa. Chcemy wskazać wierzchołki, które otrzymają ten sam klucz w każdym przyporządkowaniu kluczy zachowującym warunek kopca i podane ograniczenia.

Zbadajmy najpierw prosty przypadek, w którym mamy do wypełnienia tylko jedno drzewo, a więc gdy $t = 1$. Liczba elementów zbioru $K = \{k_1 < k_2 < \ldots < k_m\}$ musi być wtedy równa wielkości drzewa, a każdy z elementów tego zbioru musi być mniejszy niż a_1. Jasne jest, że w korzeniu drzewa należy umieścić największą liczbę z K. Jeśli korzeń drzewa ma tylko jednego syna, to synowi należy nadać klucz k_{m-1}. Jeżeli syn korzenia ma także tylko jednego syna, z całą pewnością otrzyma on klucz k_{m-2} itd. Stąd wnioskujemy, że jeśli drzewo jest ścieżką, jest tylko jedna możliwość przypisania kluczy wierzchołkom, a więc wszystkie klucze są wyznaczone jednoznacznie.

Zastanówmy się, jak to wygląda w sytuacji, gdy drzewo zawiera rozgałęzienie. Niech v będzie najbliższym korzeniowi wierzchołkiem drzewa (być może samym korzeniem), który ma co najmniej dwóch synów. Niech w będzie dowolnym potomkiem v, a s — synem v, do którego poddrzewa należy w. Załóżmy ponadto, że w ma jednoznacznie przyporządkowany klucz k. Zbiór kluczy $T(s)$ może być dowolnym z $|T(s)|$-elementowych podzbiorów zbioru $K' = \{k_1, k_2, \ldots, k_{|T(v)|-1}\}$. Ale ponieważ v ma więcej niż jednego syna, więc $|T(s)| < |T(v)| - 1$ i istnieje $|T(s)|$-elementowy podzbiór zbioru K' niezawierający k. To oznacza, że w pewnym przyporządkowaniu kluczy, k nie występuje wśród kluczy $T(s)$, więc tym bardziej nie może być on przypisany do wierzchołka w. Doszliśmy do sprzeczności z założeniem o jednoznacznym przyporządkowaniu wierzchołkowi w klucza k.

Mamy więc pierwszy rezultat.

Fakt 1. *W przypadku jednego drzewa, jednoznacznie wyznaczone są tylko klucze w wierzchołkach na ścieżce od korzenia do pierwszego rozgałęzienia.*

Przypadek wielu drzew jest bardziej skomplikowany, ponieważ zbiór nieużytych kluczy K jest wspólny dla wszystkich drzew, a zbiory kluczy odpowiadające poszczególnym poddrzewom nie muszą być jednoznacznie wyznaczone. Pokażemy, jak zredukować problem wyznaczania jednoznacznie określonych kluczy dla $t > 1$ drzew do przypadku dla $t - 1$ drzew.

Załóżmy bez straty ogólności, że drzewa są tak uporządkowane, aby zachodziło $a_1 < a_2 < \ldots < a_t$. Jako zbiór kluczy dla drzewa T_1 musimy wybrać pewien $|T_1|$-elementowy podzbiór zbioru $K_{a_1} = \{k \in K : k < a_1\}$.

Jeśli $|K_{a_1}| = |T_1|$, mamy tylko jedną możliwość — do T_1 przypisujemy wszystkie klucze z K_{a_1}, a w celu wyznaczenia jednoznacznie określonych kluczy w drzewie T_1, stosujemy wcześniej wykazany fakt. Pozostało nam obliczyć wyniki dla drzew $T_2, \ldots, T_t$ i zbioru kluczy $K' = K \setminus K_{a_1}$, a następnie dodać do wyniku jednoznacznie określone klucze z T_1.

W przypadku, gdy $|K_{a_1}| > |T_1|$, jako zbiór kluczy drzewa T_1 możemy wybrać dowolny z $|T_1|$-elementowych podzbiorów zbioru K_{a_1}. Można to łatwo uzasadnić, jeśli zauważyć, że każdy z niewybranych do T_1 elementów K_{a_1} może znaleźć się w dowolnym z pozostałych drzew, co wynika z warunku $a_2, \ldots, a_t > a_1$. Żaden z kluczy nie jest więc jednoznacznie przypisany do T_1. Co więcej, żaden z kluczy z K_{a_1} nie może zostać jednoznacznie wyznaczony, niezależnie od tego, w którym drzewie się znajdzie. Stąd, dla ostatecznego wyniku nie ma znaczenia, które z kluczy K_{a_1} trafią do T_1, a które do pozostałych drzew.

Ta obserwacja pozwala nam przeprowadzić redukcję w następujący sposób: przypisujemy $|T_1|$ najmniejszych elementów zbioru K do pierwszego drzewa, a następnie usuwamy je z K, otrzymując K'. Rozwiązujemy problem dla pozostałych drzew i dostępnych kluczy K', otrzymując zbiór jednoznacznie przypisanych kluczy do wierzchołków drzew $T_2, \ldots, T_t$. Ostatecznie, zgodnie z obserwacją, musimy „poprawić" uzyskany wynik: ze zbioru jednoznacznie wyznaczonych par (wierzchołek, klucz) usuwamy pary z kluczem należącym do K_{a_1} (o ile takie istnieją).

Rozwiązanie wzorcowe

Z opisanej redukcji problemu do przypadku mniejszej liczby drzew wynika natychmiast algorytm rozwiązujący nasze zadanie. Niestety, bez dbałości o szczegóły techniczne i użycia odpowiednich struktur danych, może on działać w czasie kwadratowym. Aby uzyskać maksymalną liczbę punktów, potrzebujemy jego szybszej implementacji.

Algorytm będzie działał tak, że po każdym kroku będziemy mieli przeanalizowany pewien zbiór kluczy A. Dla każdego klucza ze zbioru A będziemy wiedzieli, czy jest on przypisany jednoznacznie do jakiegoś wierzchołka, czy też nie. Ponadto, pewna liczba najmniejszych kluczy ze zbioru A będzie już przyporządkowana do konkretnych drzew.

Zbiór P — nienapotkanych jeszcze kluczy, oraz zbiór K — kluczy nieprzypisanych do konkretnych drzew będziemy przechowywać w postaci uporządkowanych stosów, z których elementy będziemy wyjmować, począwszy od najmniejszego. Potrzebujemy także tablicy $\mathit{niejedn}[1..n]$, w której będziemy zaznaczać klucze, o których już stwierdziliśmy, że nie mogą być wyznaczone jednoznacznie. Początkowo zbiory P i K zawierają wszystkie klucze (poza wymienionymi na wejściu) i dla każdego klucza k zachodzi $\mathit{niejedn}[k] = \mathbf{false}$.

W i-tym kroku algorytmu wykonujemy, co następuje. Ze stosu K zdejmujemy $|T_i|$ (najmniejszych) kluczy — nazwijmy ten zbiór M_i. Jeśli element ze szczytu stosu K (o ile istnieje) jest większy niż a_i, to zgodnie z Faktem 1 wyznaczamy jednoznaczne klucze w drzewie T_i z dostępnym zbiorem kluczy M_i. Ten krok można łatwo wykonać w czasie liniowym, korzystając z naszego ulubionego algorytmu przeszukiwania drzewa. Ze stosu P zdejmujemy natomiast wszystkie klucze mniejsze niż a_i.

W przeciwnym wypadku, gdy element ze szczytu stosu K jest mniejszy niż a_i, zachodzi drugi przypadek redukcji. Wiemy, że w T_i żaden z wierzchołków nie ma jednoznacznie przypisanego klucza. Wiemy także, że wszystkie dostępne na tym etapie klucze mniejsze niż a_i nie mogą być jednoznacznie przypisane do żadnego wierzchołka. Ze stosu P zdejmujemy więc wszystkie klucze mniejsze niż a_i i dla każdego z nich, ustawiamy odpowiadającą mu wartość w tablicy *niejedn* na **true**. Zachodzi tutaj niezmiennik, że wszystkie klucze k ze zbioru K, które nie należą już do zbioru P, mają przypisane $\mathit{niejedn}[k] = \mathbf{true}$.

Po przetworzeniu wszystkich drzew, pewne wierzchołki mają przyporządkowane klucze. Jako jednoznaczne przyporządkowania wypisujemy jedynie te pary (v,k), dla których $niejedn[k] = \textbf{false}$.

Ponieważ każdy klucz zostaje zdjęty ze stosów K i P dokładnie raz, algorytm działa w czasie liniowym. Implementację opisanego rozwiązania można znaleźć w pliku `pen.cpp`.

Testy

Testy z grup *1* i *2* zostały ułożone ręcznie. Pozostałe grupy testów zostały wygenerowane w sposób losowy.

W poniższej tabeli n oznacza rozmiar drzewa, p oznacza liczbę publicznie znanych pensji, a w — łączną liczbę pensji, które da się wywnioskować z tych danych.

Nazwa	n	p	w
pen1a.in	1	0	1
pen1b.in	5	3	2
pen1c.in	4	2	0
pen1d.in	15	4	6
pen1e.in	7	4	3
pen2a.in	9	3	2
pen2b.in	10	4	3
pen2c.in	24	4	3
pen2d.in	14	10	1
pen2e.in	15	4	2
pen3a.in	49	8	15
pen3b.in	53	14	15
pen4a.in	49	10	11
pen4b.in	65	30	29

Nazwa	n	p	w
pen5a.in	96	10	26
pen5b.in	142	32	12
pen6.in	439	100	188
pen7.in	1 167	151	680
pen8.in	4 814	500	1 574
pen9.in	6 682	2 000	879
pen10.in	85 368	17 674	36 022
pen11.in	226 992	100 000	78 452
pen12.in	408 054	100 000	44 555
pen13.in	510 646	200 000	146 154
pen14.in	535 787	9 599	41 300
pen15.in	853 829	700 000	142 287
pen16.in	988 134	17 992	609 131