Jan Kanty Milczek	Jan Kanty Milczek Bartosz Tarnawski	Bartosz Tarnawski
Treść zadania	Opracowanie	Program

Dostępna pamięć: 128 MB. *OI, etap III, dzień drugi, 14.03.2013*

Polaryzacja

Każdy wiedział, że to tylko kwestia czasu. Ale cóż z tego? Kiedy zagrożenie wisi nad ludźmi przez lata, staje się częścią codzienności. Traci swoją wartość...

Dzisiaj stała się jawna treść listu bitockiego chara Ziemobita do bajtockiego króla Bajtazara. Bitocja zażądała aneksji całej Bajtocji, grożąc Bitowym Magnesem Polaryzującym (BMP). W wyniku jego działania każda droga Bajtocji stałaby się jednokierunkowa. Wróg doskonale wie, że byłby to cios, którego sąsiad może nie wytrzymać, bo infrastruktura Bajtocji jest minimalna – między każdą parą miast da się przejechać na dokładnie jeden sposób.

Jak może wyglądać infrastruktura Bajtocji po działaniu BMP? Oblicz minimalną i maksymalną liczbę par miast, takich że z jednego będzie się dało przejechać do drugiego, nie naruszając skierowania dróg.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita n ($1 \leqslant n \leqslant 250\,000$), oznaczająca liczbę miast w Bajtocji. W kolejnych $n-1$ wierszach znajdują się opisy dróg. Każdy z nich składa się z dwóch liczb całkowitych u i v ($1 \leqslant u < v \leqslant n$), oznaczających, że istnieje (na razie jeszcze dwukierunkowa) droga łącząca miasta o numerach u i v.

W testach wartych 60% punktów zachodzi dodatkowy warunek $n \leqslant 10\,000$, a w części tych testów, wartych 30% punktów, zachodzi warunek $n \leqslant 100$.

Wyjście

W pierwszym wierszu standardowego wyjścia powinny znaleźć się dwie liczby całkowite oddzielone pojedynczym odstępem. Pierwsza z nich to minimalna, a druga – maksymalna liczba par różnych miast, między którymi będzie można przejechać po spolaryzowaniu dróg.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*

`4`
`1 2`
`1 3`
`1 4`
	`3 5`

a dla danych wejściowych:	*poprawnym wynikiem jest:*
8	7 28
1 2	
2 3	
3 4	
4 5	
5 6	
6 7	
7 8	

Testy „ocen":

`1ocen:` *$n = 10$, test typu „grzebień";*

`2ocen:` *$n = 15$, pełne drzewo binarne;*

`3ocen:` *$n = 100$, dwie „gwiazdy" połączone krawędzią;*

`4ocen:` *$n = 10\,000$, dwie „gwiazdy" połączone krawędzią;*

`5ocen:` *$n = 250\,000$, „gwiazda" o maksymalnym rozmiarze.*

Rozwiązanie

W zadaniu dane jest n-wierzchołkowe drzewo T. Niech s będzie funkcją przyporządkowującą każdej nieskierowanej krawędzi $\{x, y\}$ drzewa jedno z dwóch możliwych skierowań (x, y) lub (y, x). Funkcję tę nazwiemy *skierowaniem* drzewa, a przez T_s oznaczymy drzewo powstałe z T po skierowaniu jego krawędzi według s.

Powiemy, że para różnych wierzchołków (x, y) jest *dobra* w skierowaniu s, jeżeli w T_s istnieje ścieżka z x do y. Zadanie polega na znalezieniu minimalnej i maksymalnej liczby dobrych par w T, jakie możemy uzyskać przy różnych skierowaniach tego drzewa.

Zauważmy, że żadne skierowanie nie da nam wyniku mniejszego niż $n - 1$. Dzieje się tak dlatego, że każda krawędź, niezależnie od skierowania, tworzy co najmniej jedną dobrą parę. Łatwo zauważyć, że dokładnie taki wynik uzyskamy, jeśli skierujemy krawędzie tak, aby każdy wierzchołek był albo *źródłem* (wszystkie krawędzie z nim incydentne są skierowane od niego), albo *ujściem* (wszystkie krawędzie z nim incydentne są skierowane do niego). W tym celu możemy wykonać dwukolorowanie naszego drzewa, a następnie skierować wszystkie krawędzie tak, aby prowadziły z wierzchołka pierwszego koloru do wierzchołka koloru drugiego.

Widzimy zatem, że pierwszą z liczb, o jakich wypisanie jesteśmy proszeni na wyjściu, jest zawsze $n - 1$. Możemy podejrzewać, że najtrudniejsza część zadania jest dopiero przed nami...

W celu wyznaczenia maksymalnej liczby dobrych par pomocne nam będą następujące obserwacje.

Obserwacja 1. Jeśli odwrócimy skierowanie w całym drzewie, to liczba dobrych par nie zmieni się.

Jest to oczywiste, gdyż każdej dobrej parze (x, y) w starym skierowaniu będzie odpowiadać dobra para (y, x) w nowym skierowaniu.

Obserwacja 2. W optymalnym rozwiązaniu tylko liście mogą być źródłami i ujściami.

Intuicyjnym jest, że jeśli chcąc zminimalizować liczbę dobrych par, maksymalizowaliśmy liczbę wierzchołków będących źródłami i ujściami, to chcąc zmaksymalizować wynik, powinniśmy minimalizować liczbę źródeł i ujść. Jeśli jakikolwiek wierzchołek wewnętrzny (nie liść) byłby źródłem lub ujściem, to moglibyśmy wybrać jego dowolne poddrzewo i odwrócić skierowanie wszystkich jego krawędzi. Zgodnie z obserwacją 1 liczba dobrych par w tym poddrzewie nie zmieniłaby się, za to w całym drzewie doszłaby co najmniej jedna nowa dobra para. Od teraz we wszystkich rozważaniach zakładamy, że żaden wierzchołek wewnętrzny nie jest źródłem ani ujściem.

Jeżeli istnieje wierzchołek, który pojawia się na ścieżce z x do y dla każdej dobrej pary *liści* (x, y), nazwiemy go *superwierzchołkiem*. Skierowania zawierające superwierzchołek mają bardzo regularną strukturę – ukorzeniwszy drzewo w superwierzchołku, zobaczymy, że każde jego poddrzewo jest w całości skierowane albo do niego, albo od niego. Wykażemy teraz, że prawdziwa jest

Obserwacja 3. Optymalne rozwiązanie zawsze zawiera superwierzchołek.

Jeśli w skierowaniu nie istnieje superwierzchołek, to istnieje para wierzchołków (x, y) taka, że:

- istnieje ścieżka z x do y;
- każdy wierzchołek pomiędzy x a y (może ich być zero, jeśli istnieje krawędź $\{x, y\}$) ma stopień 2;
- od x wychodzi co najmniej jedna krawędź nieprowadząca do y;
- do y wchodzi co najmniej jedna krawędź nieprowadząca od x.

Udowodnienie tego faktu pozostawiamy jako ćwiczenie dla Czytelnika (dowód jest intuicyjny i prosty, ale mozolny). Oznaczmy teraz przez m_x liczbę wierzchołków, z których istnieje ścieżka do x, a przez m_y liczbę wierzchołków, do których istnieje ścieżka z y. Jeśli $m_x \leqslant m_y$, to odwracając skierowanie poddrzew wychodzących z wierzchołka x (oprócz tego zawierającego y), poprawimy wynik o co najmniej jeden. Jeśli zaś $m_y \leqslant m_x$, to odwracając skierowanie poddrzew wchodzących do y (oprócz tego zawierającego x), również poprawimy wynik o co najmniej jeden. Wykonajmy jedno z tych odwróceń i znowu poszukajmy superwierzchołka. Jeśli znów takiego nie będzie, znajdźmy nową parę (x, y) i powtórzmy rozumowanie. Jako że wynik nie może się zwiększać w nieskończoność, w końcu dotrzemy do sytuacji z superwierzchołkiem.

Rozwiązanie $O(n^2)$

Skonstruujemy teraz rozwiązanie bazujące na powyższych obserwacjach. Dla każdego wierzchołka drzewa chcemy wiedzieć, jaki byłby wynik, gdyby to on był superwierzchołkiem.

Ustalmy zatem wierzchołek drzewa v. Policzenie liczby dobrych par znajdujących się wewnątrz poddrzew wierzchołka v możemy wykonać np. algorytmem DFS w czasie $O(n)$. Liczba ta nie zależy od skierowania poddrzew (obserwacja 1). Do wyniku dochodzi jeszcze iloczyn liczby wierzchołków w poddrzewach skierowanych do v i liczby wierzchołków w poddrzewach skierowanych od v. Musimy zatem dla każdego poddrzewa tak wybrać skierowanie, aby ten iloczyn był jak największy. Iloczyn postaci $a(n-1-a)$ jest największy, gdy a jest możliwie najbliższe $\frac{n-1}{2}$. Używając algorytmu pakowania plecaka (wydawania reszty), ustalamy, jak bardzo możemy się zbliżyć do tej wartości. Złożoność czasowa tego algorytmu to $O(n \cdot \deg(v))$, gdzie $\deg(v)$ to stopień wierzchołka v.

Ostatecznie, łączna złożoność czasowa dla całego drzewa będzie więc wynosiła $O\big(\sum_v (n + n \cdot \deg(v))\big) = O(n^2)$. Rozwiązanie to zostało zaimplementowane w plikach `pols5.cpp` oraz `pols6.pas`, dostaje ok. 50% punktów.

Rozwiązanie $O(n^2)$, wersja 2

Zauważmy, że jeśli uruchomimy rozwiązanie problemu plecakowego dla wierzchołka v, w którym jedno z poddrzew (oznaczmy, że jest ukorzenione w wierzchołku u) ma rozmiar większy niż $\frac{n}{2}$, to wszystkie inne poddrzewa skierujemy w przeciwną stronę. W takim razie, całe poddrzewo ukorzenione w v zostanie skierowane w stronę u. Zatem równie dobre rozwiązanie moglibyśmy uzyskać, rozpatrując problem plecakowy skonstruowany w wierzchołku u. Kontynuując to rozumowanie, dotrzemy w końcu do wierzchołka, którego wszystkie poddrzewa mają rozmiar nie większy niż $\frac{n}{2}$. Taki wierzchołek nazywa się *centroidem* drzewa. Każde drzewo ma co najwyżej dwa centroidy, można zatem wykorzystać spostrzeżenie, że superwierzchołkiem musi być centroid, i tylko dla nich rozwiązywać problem plecakowy. Niestety, jeśli zrobimy to w czasie $O(n^2)$, to nie uzyskamy lepszej złożoności niż w poprzednim rozwiązaniu. Jednak w przypadku poprzedniego rozwiązania czas wykonania zdominowany jest przez n-krotne przeszukiwanie drzewa. Jeżeli wyznaczymy najpierw centroid, to większość czasu zajmą operacje na tablicy i rozwiązanie będzie działać istotnie szybciej na wszystkich wygenerowanych testach. Takie rozwiązanie uzyskiwało ok. 60% punktów. Zostało zaimplementowane w plikach `pols3.cpp` oraz `pols4.pas`.

Rozwiązanie wzorcowe $O(n\sqrt{n})$

Rozwiązanie można przyspieszyć, sprytniej rozwiązując problem plecakowy. Zauważmy, że występuje w nim co najwyżej n liczb o łącznej sumie $n-1$. Takie programowanie dynamiczne można zaimplementować w czasie $O(n\sqrt{n})$. Korzystamy ze spostrzeżenia, że przedmiotów większych niż $\sqrt{n}$ jest co najwyżej $\sqrt{n}$, natomiast przedmioty mniejsze niż $\sqrt{n}$ rozpatrujemy w pakietach złożonych z jednakowych przedmiotów. Rozwiązanie to zostało opisane szczegółowo w opracowaniu zadania *Banknoty* z XII OI [12]. Jeden ze sposobów na przyspieszenie rozwiązania został pokrótce opisany poniżej.

Załóżmy, że mamy pewną grupę przedmiotów tej samej wielkości s. Rozpatrując $(j+1)$-szy przedmiot z tej grupy, sprawdzamy jedynie, czy osiągalne są sumy postaci $x + s$, gdzie x jest sumą osiągalną dopiero po dodaniu j-tego przedmiotu wielkości s.

Wtedy łączny czas przetwarzania przedmiotów, które nie są rozpatrywane jako pierwsze w swoim rozmiarze, szacuje się przez sumaryczną liczbę wartości, które w pewnym momencie stały się dostępne. Tych zaś jest najwyżej n. Takie rozwiązanie zostało zaimplementowane w plikach `pol.cpp` i `pol2.pas`. Rozwiązanie autorskie, które jest podobne i trochę prostsze, zostało zaimplementowane w pliku `pol1.cpp`.

Rozwiązanie alternatywne $O(n\sqrt{n})$

Każda liczba ze zbioru $\{0, 1, \ldots, m\}$ jest sumą elementów pewnego podzbioru $\{2^0, 2^1, 2^2, \ldots, 2^{t-1}, m - (2^t - 1)\}$ dla $t = \lfloor \log_2 m \rfloor$ i odwrotnie, suma elementów każdego podzbioru tego (multi)zbioru należy do $\{0, 1, \ldots, m\}$.

Jeżeli m przedmiotów ma taką samą wielkość s, to zamiast m-krotnie uaktualniać tablicę dostępnych wartości, możemy „posklejać" przedmioty z tej grupy w bloki wielkości $s, 2s, \ldots, 2^{t-1}s, (m - (2^t - 1))s$.

Złożoność czasowa takiego rozwiązania łatwo szacuje się przez $O(n\sqrt{n}\log n)$. Tak naprawdę złożoność to $O(n\sqrt{n})$, ale dowód tego nie jest istotą tego opracowania. Takie rozwiązanie zostało zaimplementowane w plikach `pol5.cpp` oraz `pol6.pas`.

Rozwiązanie $O(n^2)$, wersja 3

Nie korzystając z obserwacji o dystrybucji wielkości przedmiotów, można przyspieszyć rozwiązanie problemu plecakowego, korzystając z masek bitowych. Zamiast trzymać tablicę zmiennych logicznych informującą, czy dana wartość jest osiągalna, reprezentujemy ją jako kolejne bity słów 32-bitowych. Dzięki temu zmniejsza się stała przy aktualizacji dostępnych stanów po dodaniu nowego elementu. Rozwiązanie działające w czasie $O(n^2)$ z tą optymalizacją zostało zaimplementowane w plikach `pols1.cpp` oraz `pols2.pas`. Dostaje ok. 70–80% punktów.

Rozwiązanie alternatywne $O(n\sqrt{n})$, wersja 2

Tak samo korzystając z masek bitowych, można zmniejszyć stałą w rozwiązaniu $O(n\sqrt{n})$ (pliki `pol3.cpp` oraz `pol4.pas`). Rozwiązanie to działa szybciej od wzorcowego, przy czym różnica w czasie wykonania staje się znacząca dopiero dla danych wejściowych przekraczających limity z treści zadania.

Rozwiązania wolne

Istnieje rozwiązanie wielomianowe nie korzystające z powyższych obserwacji, które także opiera się na programowaniu dynamicznym. W tym rozwiązaniu wypełniamy tablicę d, w której $d[v, x, y]$ to maksymalny wynik częściowy dla poddrzewa ukorzenionego w wierzchołku v przy założeniu, że

(a) istnieje x wierzchołków w poddrzewie v, z których prowadzi ścieżka do v,

(b) istnieje y wierzchołków w poddrzewie v, do których prowadzi ścieżka z v.

Wartości tablicy można obliczyć podczas przeszukiwania drzewa, a stan można wyznaczyć w czasie $O(n)$, jeżeli pamięta się dodatkowo najlepsze wyniki przy założeniu samego warunku (a) lub (b). Sumaryczny czas działania tego algorytmu to $O(n^4)$,

został on zaimplementowany w plikach `pols7.cpp` oraz `pols8.pas`. Dostaje ok. 30% punktów.

Najprostszym, choć niestety bardzo wolnym, podejściem było generowanie wszystkich (lub wielokrotne losowanie) skierowań krawędzi, a następnie liczenie liczby dobrych par algorytmem DFS (wywoływanym raz lub z każdego wierzchołka). Przykładowe rozwiązanie wykładnicze, które sprawdza wszystkie możliwe skierowania krawędzi w czasie $O(2^n \cdot n)$, zostało zaimplementowane w plikach `pols9.cpp` i `pols10.pas`. Takie rozwiązanie dostawało 10% punktów.

Rozwiązania niepoprawne

Ponieważ maksymalnym wynikiem jest $\binom{n}{2}$, zadanie wymaga użycia arytmetyki 64-bitowej. Rozwiązanie o złożoności $O(n\sqrt{n})$, ale używające do reprezentacji wyniku typu 32-bitowego, otrzymuje 80% punktów (`polb7.cpp`).

Zaimplementowane rozwiązania, które dostają 0-10% punktów, to:

1. Wielokrotne generowanie losowego skierowania krawędzi (`polb1.cpp`).
2. Szukanie centrum (środka średnicy drzewa) zamiast centroidu (`polb2.cpp`).
3. Szukanie wierzchołka o największym stopniu zamiast centroidu (`polb3.cpp`).
4. Ustawianie synów centroidu w ciąg według rosnących wielkości poddrzew, szukanie prefiksu tego ciągu o sumie wielkości poddrzew możliwie zbliżonej do $\frac{n-1}{2}$ (`polb4.cpp`).
5. Heurystyka polegająca na tym, że po znalezieniu centroidu nie rozwiązuje się problemu plecakowego, ale uznaje się, że suma $\frac{n-1}{2}$ jest osiągalna (`polb5.cpp`).

Heurystyka zaimplementowana w pliku `polb6.cpp` zdobywa maksymalnie 30% punktów. Po znalezieniu centroidu wielokrotnie losowo ustawia jego synów w ciąg, a następnie szuka prefiksu o sumie poddrzew najbardziej zbliżonej do $\frac{n-1}{2}$.

Testy

Funkcje generujące testy do zadania dzielą się na dwie grupy: ogólne (generujące drzewa według pewnych parametrów, ale bez wyróżnionych wierzchołków) oraz z ustalonym centroidem (wybierające jeden wierzchołek jako centroid, a następnie generujące poddrzewa jego synów o ustalonych własnościach).

1. Testy ogólne:
   (a) gwiazda,
   (b) ścieżka,
   (c) ścieżka, do której podczepione są mniejsze drzewa,
   (d) losowe drzewo,
   (e) drzewo binarne (zrównoważone i niezrównoważone),
   (f) połączenie ścieżki z jednym z drzew wymienionych powyżej.

2. Testy z ustalonym centroidem:
   (a) Synowie centroidu mają poddrzewa o losowych wielkościach; ustalone są następujące parametry: liczba synów centroidu, maksymalna i minimalna wielkość ich poddrzew.
   (b) Jak wyżej, ale synowie centroidu podzieleni są na dwie grupy, dla każdej osobno ustalamy parametry.
   (c) Wielkości poddrzew rzadko się powtarzają.
   (d) Wielkości poddrzew często się powtarzają.
   (e) Wielkości wszystkich poddrzew są podzielne przez ustaloną liczbę.
   (f) a synów centroidu ma poddrzewa wielkości b, b synów centroidu ma poddrzewa wielkości a oraz $\mathrm{NWD}(a, b) = 1$. Ten test skutecznie wykrywa heurystykę `polb6.cpp`.

Testy są grupowane po 6, jest 10 grup. W każdej grupie znajdują się 2–3 testy ogólne, wśród nich zawsze pojawia się gwiazda i ścieżka. W każdej grupie pojawia się też test typu *2e*.

XXV Międzynarodowa Olimpiada Informatyczna,

Brisbane, Australia 2013