Wojciech Nadara — Treść zadania, Opracowanie
Marek Sokołowski — Program

Dostępna pamięć: 256 MB. *OI, etap III, dzień pierwszy, 16.04.2015*

Tablice kierunkowe

Po całym roku wytężonego programowania Bajtazar postanowił udać się na wakacje. Gdy jechał swoim samochodem na wyczekiwane wczasy, mijał wiele **tablic kierunkowych**, *na których były napisane aktualne odległości (w kilometrach) do różnych miast w Bajtlandii. O ile dokładny dystans pomiędzy tablicą a miastem nie musi wyrażać się całkowitą liczbą kilometrów, o tyle na tablicy mogą się znaleźć tylko liczby całkowite. Dlatego podane na tablicach odległości są* **zaokrągleniami** *prawdziwych odległości* **w dół**.

Po podróży Bajtazar stwierdził, że informacje umieszczone na mijanych tablicach wyglądały podejrzanie. Uważa, że nie wszystkie tablice zostały rozmieszczone przez ludzi kompetentnych i informacje na niektórych z nich były sprzeczne. Bohater chciałby się dowiedzieć, jak bardzo dane na nich umieszczone były nieprawdziwe. W tym celu postanowił znaleźć największy zbiór tablic, na których informacje nie są wzajemnie sprzeczne. Niestety jest to dla niego zbyt trudne zadanie i poprosił Ciebie o pomoc. Na szczęście Bajtazar ma bardzo dobrą pamięć, zatem pamięta wszystkie tablice, które minął. Nie patrzył on jednak na licznik kilometrów w samochodzie, zatem nie wie, kiedy je zobaczył. Nawet kolejność ich napotykania nie jest dla niego jasna.

Zakładamy, że Bajtlandia jest prostą, a miasta są na tyle małe, że możemy je utożsamić z punktami na tej prostej. Przyjmujemy także, że w trakcie swojej podróży Bajtazar **nie minął żadnego z miast**. *Zbiór tablic jest niesprzeczny, jeśli da się dobrać współrzędne tablic i miast tak, aby zaokrąglone odległości na tablicach były zgodne z prawdą. Oczywiście ani miasta, ani tablice nie muszą znajdować się w punktach o współrzędnych całkowitych. Żadne dwa miasta ani żadne dwie tablice nie mogą znajdować się w tych samych punktach. Co ciekawe, Bajtazar wie, że bajtlandzcy drogowcy nie są całkowicie niekompetentni (sam kiedyś nadzorował budowę drogi, którą jechał). Jest pewien, że istnieje zbiór co najmniej 20% tablic, na których widnieją niesprzeczne informacje.*

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite n i m ($1 \leqslant n \leqslant 1000$, $1 \leqslant m \leqslant 200$) oddzielone pojedynczym odstępem, oznaczające odpowiednio liczbę tablic napotkanych przez Bajtazara oraz liczbę miast w Bajtlandii. Każdy z kolejnych n wierszy zawiera opis jednej tablicy; w i-tym z tych wierszy znajduje się ciąg m liczb całkowitych $d_{i,1}, d_{i,2}, \ldots, d_{i,m}$ ($1 \leqslant d_{i,j} \leqslant 10^6$) pooddzielanych pojedynczymi odstępami, gdzie $d_{i,j}$ oznacza odległość do miasta o numerze j (w kilometrach) na i-tej tablicy, zaokrągloną w dół do najbliższej liczby całkowitej.

W testach wartych 60% punktów zachodzą dodatkowe warunki $n \leqslant 500$, $m \leqslant 50$. W podzbiorze tych testów wartym 40% punktów zachodzi warunek $n \leqslant 100$, a w podzbiorze wartym 20% punktów zachodzi $n \leqslant 15$.

Wyjście

W pierwszym wierszu standardowego wyjścia powinna się znaleźć jedna liczba całkowita t oznaczająca największą możliwą liczbę tablic, które przedstawiały niesprzeczne informacje. W drugim wierszu powinno się znaleźć t liczb całkowitych oznaczających numery tych tablic. Powinny one zostać podane w kolejności, w której mógł je napotykać Bajtazar. Jeżeli jest wiele możliwych rozwiązań, Twój program powinien wypisać którekolwiek.

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*

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

Wyjaśnienie do przykładu: *Jeżeli druga tablica będzie stała w punkcie $x = 0$, a pierwsza w punkcie $x = \frac{1}{2}$, pierwsze miasto będzie się mieściło w punkcie $x = 2\frac{1}{2}$, a drugie w punkcie $x = 3$, to odległości podane na tablicach o numerach 1 i 2 będą zaokrągleniami w dół prawdziwych odległości miast od tablic. Dla tablic o numerach 1 i 3 również istnieje poprawne rozmieszczenie.*

Jest jasne, że druga i trzecia tablica są ze sobą sprzeczne, zatem nie istnieje rozmieszczenie tablic i miast, w którym wszystkie trzy tablice podawałyby prawidłowe informacje.

Testy „ocen":

`1ocen`: *$n = 5$, $m = 1$; tablice kierunkowe wskazują różne zaokrąglone odległości do jedynego miasta;*

`2ocen`: *$n = 5$, $m = 2$; każde dwie tablice kierunkowe są ze sobą sprzeczne; w odpowiedzi należy podać dowolną z nich;*

`3ocen`: *$n = 200$, $m = 199$; na wszystkich tablicach widnieją niesprzeczne informacje – przykładowo, można umieścić i-tą tablicę w punkcie $\frac{i}{n}$, a j-te miasto w punkcie $10^6 + \frac{j}{n}$.*

Rozwiązanie

Zależności pomiędzy tabliczkami

Na początek zastanówmy się, kiedy zbiór tabliczek jest niesprzeczny. Dwa zbiory tabliczek nazwijmy *równoważnymi*, jeżeli oba są sprzeczne albo oba są niesprzeczne. Dla ustalenia uwagi załóżmy, że Bajtazar jechał samochodem z lewej na prawo. Zauważmy, że jeżeli wszystkie odległości na pewnej tabliczce zmniejszymy (zwiększymy) o 1, to otrzymamy równoważny zbiór tabliczek, ponieważ potencjalne umiejscowienie tabliczek i miast w drugim z układów można utrzymać, przesuwając tę tabliczkę o 1 w prawo w pierwszym z układów (i na odwrót). Podobnie rzecz się ma, gdybyśmy

chcieli zmniejszyć (zwiększyć) odległości do konkretnego miasta o 1 na wszystkich tabliczkach. Poprawne rozmieszczenie w drugim z układów otrzymamy, przesuwając dane miasto o 1 w lewo (prawo) w pierwszym z układów. W ten sposób na niektórych tabliczkach mogą chwilowo pojawić się liczby ujemne – nie jest to jednak problemem, o ile zadbamy, by na końcu tego procesu wszystkie znów były nieujemne.

Odległość do miasta j na tabliczce i oznaczmy zgodnie z treścią zadania przez $d_{i,j}$. Następnie wykonajmy operację $d_{i,j} := d_{i,j} - d_{1,j}$, otrzymując układ równoważny (od odległości do każdego miasta odjęliśmy tyle samo). Dzięki temu pierwsza tabliczka składa się z samych zer. Następnie dla każdej tabliczki i odejmijmy od odległości na niej napisanych $\min_j\{d_{i,j}\}$, ponownie otrzymując układ tabliczek równoważnych. Wtedy na każdej tabliczce będą widniały nieujemne liczby całkowite i na każdej będzie co najmniej jedno 0, a ponadto na pierwszej tabliczce wciąż będą same zera. Otrzymany układ odległości na tabliczce o numerze i nazwijmy *znormalizowaną tabliczką o numerze i względem tabliczki* 1. Analogicznie można wprowadzić pojęcie normalizacji względem dowolnej innej tabliczki.

Bez straty ogólności możemy umieścić pierwszą tabliczkę w punkcie $x = 0$. Wtedy wszystkie miasta muszą leżeć na odcinku $[0, 1)$, gdyż ich odległości od tej tabliczki po zaokrągleniu w dół są równe 0. Co więcej, wszystkie tabliczki mogą znaleźć się na odcinku $(-1, 0]$, gdyż dla każdej z nich odległość od co najmniej jednego z miast zaokrąglona w dół jest równa 0. Jeżeli wobec tego na którejkolwiek z tabliczek widnieje liczba co najmniej 2, to układ tabliczek jest sprzeczny. Zatem jeśli układ jest niesprzeczny, na wszystkich tabliczkach muszą być umieszczone wyłącznie liczby 0 i 1.

Zauważmy teraz, że w żadnym niesprzecznym układzie tabliczek nie może zachodzić $d_{i_1,j_1} > d_{i_2,j_1}$ oraz $d_{i_1,j_2} < d_{i_2,j_2}$ (patrz przykład w treści zadania). Pozwala nam to wprowadzić liniowy porządek na tabliczkach, który będzie oznaczał porządek, w którym znormalizowane tabliczki będą umieszczone na odcinku $(-1, 0]$. Innymi słowy, dla dowolnych dwóch tabliczek o indeksach i_1 oraz i_2, dla każdego miasta j zachodzi albo $d_{i_1,j} \leqslant d_{i_2,j}$, albo $d_{i_2,j} \leqslant d_{i_1,j}$. W pierwszym z tych przypadków wiemy, że tabliczka o indeksie i_2 stoi wcześniej na osi liczbowej, a w drugim przypadku wcześniej stoi tabliczka i_1. Jeżeli tabliczki mają dokładnie takie same odległości, to możemy je umieścić w dowolnej kolejności. Okazuje się, że jeżeli w naszym przypadku, do którego sprowadziliśmy początkowe pytanie, ani nie zachodzi $d_{i_1,j_1} > d_{i_2,j_1}$ oraz jednocześnie $d_{i_1,j_2} < d_{i_2,j_2}$ dla odpowiednich indeksów, ani po znormalizowaniu na tabliczkach nie ma liczb większych od 1, to jest to już warunek wystarczający do tego, aby oryginalny zbiór tabliczek był niesprzeczny. Fakt, że na tabliczkach da się wprowadzić opisany porządek, oznacza tyle, że jeżeli posortujemy tabliczki niemalejąco względem sumy odległości na nich napisanych, to zbiór miast, do których odległość na pewnej tabliczce wynosi 1, zawiera się w analogicznym zbiorze miast dla następnej tabliczki (odpowiada to kolejności tabliczek od prawej do lewej na odcinku $(-1, 0]$). Jeżeli wszystkie te warunki są spełnione, to możemy w kolejności sumy liczb na tabliczkach stawiać je w odstępach $\frac{1}{n}$ na odcinku $(-1, 0]$. Łatwo zauważyć, że dla każdego miasta da się je postawić na dokładnie jednym odcinku długości $\frac{1}{n}$ wewnątrz odcinka $[0, 1)$, tak aby wszystkie odległości do tego miasta napisane na tabliczkach były prawdziwe.

Przykładowo, jeśli mamy $m = 4$ miasta i $n = 5$ pięć tabliczek, które po znormalizowaniu wyglądają następująco:

$$0\,0\,0\,0,\;\; 0\,1\,0\,0,\;\; 0\,1\,0\,0,\;\; 0\,1\,1\,0,\;\; 1\,1\,1\,0,$$

to tabliczki postawimy kolejno w punktach 0, $-\frac{1}{5}$, $-\frac{2}{5}$, $-\frac{3}{5}$ i $-\frac{4}{5}$. Miasto 2 możemy teraz umieścić gdziekolwiek na odcinku $(\frac{4}{5}, 1)$, miasto 3 na odcinku $(\frac{2}{5}, \frac{3}{5})$, miasto 1 na odcinku $(\frac{1}{5}, \frac{2}{5})$, a miasto 4 na odcinku $(0, \frac{1}{5})$.

W ten sposób możemy stworzyć rozwiązanie wykładnicze sprawdzające dla każdego podzbioru tabliczek, czy jest on niesprzeczny. Otrzymamy rozwiązanie działające w czasie $O(2^n n^2 m)$; przykładową implementację można znaleźć w pliku `tabs1.cpp`.

Rozwiązanie brutalne, aczkolwiek rokujące

Poczynione do tej pory obserwacje pozwalają nam już przymierzyć się do stworzenia algorytmu działającego w czasie wielomianowym. Ustalmy i-tą tabliczkę i dokonajmy normalizacji pozostałych względem niej. W takim zbiorze tabliczek będziemy szukać jak największego zbioru tabliczek spełniającego warunki opisane w poprzednim akapicie, tzn. takiego ciągu tabliczek, że liczby na nich napisane to wyłącznie 0 i 1 oraz dla każdych dwóch tabliczek zbiór pozycji, na których występują jedynki na wcześniejszej z nich, zawiera się w zbiorze pozycji, na których występują jedynki na późniejszej z nich. Tabliczki, na których występują liczby różne od 0 i 1, możemy całkowicie zignorować.

Stwórzmy zatem graf, w którym krawędź od tabliczki a do tabliczki b istnieje wtedy i tylko wtedy, gdy zbiór pozycji z odległością 1 na tabliczce a zawiera się w zbiorze pozycji z odległością 1 na tabliczce b – taką uporządkowaną parę tabliczek (a, b) nazwijmy dopuszczalną. Zauważmy, że postawiony w zadaniu problem sprowadza się do znalezienia najdłuższej ścieżki w DAG-u (czyli w acyklicznym grafie skierowanym), co jesteśmy w stanie zrobić w czasie $O(n^2)$ za pomocą programowania dynamicznego w kolejności wierzchołków zgodnej z porządkiem topologicznym. To daje nam pierwszy wielomianowy algorytm na rozwiązanie tego zadania (powtarzamy to dla normalizacji względem każdej możliwej tabliczki). Krawędzie w tym grafie wyznaczamy, sprawdzając dla każdej pary, czy jest dopuszczalna.

Porządek topologiczny, w którym powinniśmy przetwarzać wierzchołki, jest równy porządkowi wyznaczonemu przez sumy liczb na tabliczkach. Jeżeli dwie tabliczki mają identyczną sumę odległości, to w porządku kolejność między nimi możemy ustalić dowolnie.

W takim razie po wyznaczeniu rzeczonego grafu i znalezieniu w nim długości najdłuższej ścieżki (i powtórzeniu tego dla każdej tabliczki jako tabliczki, względem której normalizujemy) już wiemy, jak duży zbiór tabliczek stanowi odpowiedź. Z tego samego programowania dynamicznego jesteśmy także w stanie w łatwy sposób odzyskać samą najdłuższą ścieżkę, czyli najlepszy zbiór tabliczek. Pozostaje jeszcze problem, jak odtworzyć oryginalną kolejność, w której Bajtazar napotykał tabliczki. Jednak aby ją odzyskać, wystarczy jedynie posortować tabliczki nierosnąco względem sumy odległości (oryginalnych!) na nich napisanych.

Nasuwa się jeszcze techniczny problem: jak szybko sprawdzić, czy dana para tabliczek (a, b) jest dopuszczalna. Można jawnie przeiterować się po wszystkich pozycjach i sprawdzić, czy nie istnieje taka pozycja, na której na tabliczce a widnieje 1, a na tabliczce b widnieje 0, i wtedy zajmie nam to $O(m)$ czasu dla ustalonej pary. Zauważmy, że informacja zawarta na jednej tabliczce przedstawia się jako $m \leqslant 200$ bitów, a tyle możemy zawrzeć w czterech zmiennych typu całkowitego 64-bitowego, np. `long long`

w C++. Po zamienieniu ciągów bitów na odpowiadające im liczby, sprawdzenie, czy jeden ciąg bitów zawiera się w drugim, to po prostu wykonanie pewnych operacji bitowych na liczbach je reprezentujących, co wykonuje się w czasie stałym. Zatem jeżeli zamieniliśmy odpowiednio wcześniej ciągi m bitów na odpowiadające im czwórki zmiennych typu `long long`, to sprawdzenie, czy dana para tabliczek jest dopuszczalna, odbywa się za pomocą 4 operacji bitowych. W ogólności jest to złożoność $O(\frac{m}{B})$, gdzie B jest długością słowa maszynowego w systemie, na którym pracujemy. W naszym przypadku mamy $B = 64$. Implementacja tej części w języku C++ znacznie się upraszcza, jeżeli zamiast trzymania ciągu bitów w kilku zmiennych typu `long long` użyjemy struktury `bitset`. Pozwala ona wykonywać w istotnie łatwiejszy sposób wiele operacji na ciągach bitów długości m w czasie $O(\frac{m}{B})$. Czytelników, którzy nie znają tej struktury, zdecydowanie zachęcamy do zapoznania się z nią oraz jej interfejsem.

Tak oto otrzymaliśmy algorytm rozwiązujący zadanie w czasie $O(n^2m+n^3+\frac{n^3m}{B})$. Istotnie, dla każdej z tabliczek, których jest n, najpierw musimy znormalizować wszystkie tabliczki względem niej, co zajmuje $O(nm)$ czasu. Potem stosujemy programowanie dynamiczne szukające najdłuższej ścieżki w DAG-u (w którym musimy sprawdzać istnienie potencjalnie n^2 krawędzi), co zajmuje $O(n^2+\frac{n^2m}{B})$ czasu. Stąd pojedyncza faza zajmuje $O(nm+n^2+\frac{n^2m}{B})$ czasu, czyli n faz zajmuje $O(n^2m+n^3+\frac{n^3m}{B})$ czasu. Złożoność pamięciowa wynosi $O(nm)$. Takie rozwiązanie w zupełności wystarcza do rozwiązania testów wartych 60% punktów (implementacje: `tabs4.cpp` – z typem `long long`, `tabs5.cpp` – z typem `bitset`). Przy nieuważnej implementacji bez masek bitowych rozwiązanie działa w czasie $O(n^3m)$ i zdobywało na zawodach ok. 40% punktów (implementacje: `tabs2.cpp`, `tabs3.cpp`).

Rozwiązanie wzorcowe

Zauważmy ciekawą własność przedstawionego rozwiązania. Mianowicie optymalną odpowiedź znajdziemy w każdej fazie, w której normalizujemy wszystkie tabliczki względem **dowolnej** tabliczki należącej do optymalnego rozwiązania (dlaczego?). To podsuwa myśl, że jeżeli rozwiązanie jest duże, to powinniśmy prędko je znaleźć, co pozwala zastosować algorytm randomizowany! Możemy mianowicie w każdym kroku normalizować względem losowo wybranej tabliczki. Jeżeli trafimy na tabliczkę należącą do rozwiązania, to normalizując względem niej, znajdziemy rozwiązanie. Jeżeli rozwiązanie będzie się składało z $\geqslant \frac{n}{5}$ tabliczek, to prawdopodobieństwo, że nie znajdziemy go w ciągu 50 faz, wynosi $(\frac{4}{5})^{50} \approx 1{,}5 \cdot 10^{-5}$, co jest w pełni satysfakcjonujące. Takie rozwiązanie będzie działało w czasie $O(f \cdot (nm + n^2 + \frac{n^2m}{B}))$, gdzie f jest liczbą faz, co dla $f = 50$, $n \leqslant 1000$, $m \leqslant 200$ powinno być wystarczająco szybkie i udzielać błędnej odpowiedzi w pomijalnie małej liczbie przypadków. Wystarcza ono do rozwiązania właściwej wersji zadania.

Implementacje takiego rozwiązania można znaleźć w plikach `tab.cpp`, `tab2.c` i `tab3.pas`.