Karol Cwalina
Treść zadania, Opracowanie

Piotr Sankowski
Program

Wschód-Zachód

Umowy międzynarodowe podpisane przez Wielce Rozległe Państwo nakładają na nie obowiązek tranzytowy — musi ono umożliwić przewiezienie koleją odpadów nuklearnych z elektrowni u wschodniego sąsiada do punktów utylizacji za zachodnią granicą. Względy ekologiczne nakazują taką organizację ruchu, by pociągi z odpadami jak najszybciej opuściły terytorium kraju.

Sieć kolejowa w tym państwie ma bardzo szczególną strukturę. Składa się ona z n miast-węzłów kolejowych oraz $n-1$ dwukierunkowych odcinków torów, łączących miasta-węzły. Między każdymi dwoma miastami istnieje możliwość przejazdu. Ponadto istnieje taki odcinek sieci, którego końce nie są miastami granicznymi, oraz każdy przejazd z miasta na granicy wschodniej do miasta na granicy zachodniej prowadzi przez ten odcinek.

Wszystkie pociągi z odpadami przyjeżdżają na granicę wschodnią tego samego dnia przed świtem, przy czym każdy z nich do innego miasta. Ze względu na niebezpieczeństwo wypadku pociągi jeżdżą tylko w dzień i po żadnym odcinku nie może jednocześnie jechać więcej niż jeden pociąg z odpadami, natomiast dowolnie wiele takich pociągów może oczekiwać w mieście-węźle. Dodatkowo przejechanie jednego odcinka zajmuje pociągowi jeden dzień. Ruch musi być tak zorganizowany, by każdy pociąg z odpadami dojechał do innego przejścia na granicy zachodniej.

Ile co najmniej dni pociągi z odpadami muszą spędzić na terytorium Wielce Rozległego Państwa?

Zadanie

Zadanie polega na napisaniu programu, który:

- *wczyta ze standardowego wejścia opis sieci kolejowej i przejść granicznych, do których przyjechały pociągi z odpadami;*
- *wyznaczy minimalną liczbę dni, jakie musi trwać tranzyt;*
- *wypisze znalezioną liczbę na standardowe wyjście.*

Wejście

Pierwsza linia wejścia zawiera trzy pooddzielane pojedynczymi odstępami liczby naturalne $1 \leqslant n,w,z \leqslant 10^6$, $n \geqslant w+z+2$. Liczba n oznacza liczbę miast-węzłów (są one ponumerowane od 1 do n), zaś w i z oznaczają liczby przejść granicznych odpowiednio na granicy wschodniej i zachodniej. Przejścia na granicy wschodniej oznaczone są liczbami $1,\ldots,w$, zaś na zachodniej liczbami $n-z+1,\ldots,n$.

W kolejnych $n-1$ liniach znajduje się opis odcinków sieci kolejowej. Każda linia opisu zawiera dwie różne liczby naturalne oddzielone pojedynczym odstępem, $1 \leqslant a,b \leqslant n$. Są to numery miast-węzłów połączonych odcinkiem torów.

W $n+1$-ej linii znajduje się jedna liczba naturalna p, $1 \leqslant p \leqslant w$, $1 \leqslant p \leqslant z$, oznaczająca liczbę pociągów z odpadami. W następnej (i ostatniej) linii wejścia znajduje się p, pooddzielanych pojedynczymi odstępami, różnych liczb naturalnych, z których każda jest nie większa niż

w. Są to numery przejść granicznych na granicy wschodniej, do których przyjechały pociągi z odpadami.

Wyjście

Pierwsza i jedyna linia wyjścia powinna zawierać dokładnie jedną liczbę całkowitą, równą minimalnej liczbie dni, jakie odpady muszą spędzić na terytorium państwa.

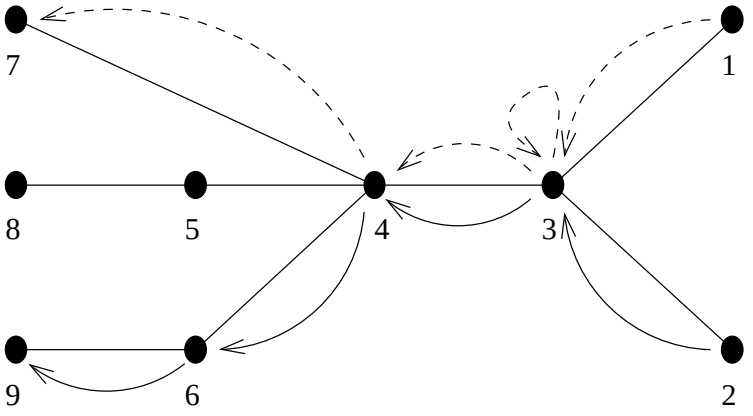
Przykład

Dla danych wejściowych:

```
9 2 3
1 3
2 3
4 3
4 5
4 6
7 4
5 8
9 6
2
1 2
```

poprawnym wynikiem jest:

```
4
```



Strzał ki przedstawiają ruch pociągów w kolejnych dniach dla jednej z optymalnych organizacji ruchu kolejowego —pętla oznacza, ˙ze danego dnia pociąg stał w mieście-wę´zle.

Rozwiązanie

Chwila namysłu pozwala wyrobić w sobie przekonanie, że stawiane przed nami zadanie jest w istocie problemem grafowym. I tak faktycznie jest: miasta-węzły są wierzchołkami grafu, zaś połączenia kolejowe jego krawędziami. To może w pewien sposób ukierunkować nasze

poszukiwania rozwiązania, bo dla grafów opracowano już bardzo wiele efektywnych algorytmów, np. przechodzenia wszerz lub w głąb. Niemniej jednak, problem wydaje się opierać nachalnym próbom zastosowania jednego z nich — sprawia wrażenie, jakby wymagał metody znacznie rozwijającej algorytmy poszukiwania najkrótszych ścieżek.

Na szczęście okazuje się że to nieprawda, a to czego nam brakuje, to zrozumienia szczególnej struktury tego grafu. W treści zadania możemy znaleźć dwie ważne informacje:

1. z każdego miasta-węzła można dojechać do każdego innego, czyli innymi słowy graf jest **spójny**;
2. w n-węzłowej sieci kolejowej jest dokładnie $n-1$ dwukierunkowych odcinków torów.

Te dwa fakty wskazują, że graf ma strukturę acyklicznego grafu spójnego, czyli drzewa. To jest intuicyjnie oczywiste, gdy wyobrazimy sobie, że graf nie jest cały dany w jednej chwili, tylko stopniowo się rozbudowuje. Najpierw mamy n wierzchołków, a potem w $n-1$ krokach dokładamy po jednej krawędzi. Zatem, startujemy w sytuacji, gdy graf ma n spójnych składowych, a kończymy tylko z jedną składową (spójna składowa to dowolny maksymalny zbiór wierzchołków takich, że pomiędzy każdymi dwoma istnieje ścieżka). Ponieważ dołożenie krawędzi może zmniejszyć liczbę spójnych składowych grafu co najwyżej o 1, to aby dołożenie $n-1$ krawędzi uczyniło graf spójnym, każda kolejna krawędź musi łączyć dwie różne składowe, co wyklucza powstanie cyklu.

To „odkrycie" jest milowym krokiem na naszej drodze do rozwiązania zadania, bo oznacza, że każdy pociąg na jeden tylko sposób może przejechać pomiędzy każdymi dwoma miastami (oczywiście z dokładnością do wyboru miejsc postojów). W efekcie każdy odcinek toru ma jednoznacznie określone końce: wschodni i zachodni, a każda trasa pociągu przebiega odcinki od końca wschodniego w kierunku końca zachodniego (nie rozważamy cykli, gdyż można je zastąpić wielodniowym pobytem w jednym mieście-węźle).

Dla wygody nazwijmy wyróżniony odcinek sieci jej **gardłem**, a jego końce niech to będą **węzeł wschodni** i **węzeł zachodni**. Wszystkie miasta leżące na wschód od węzła zachodniego nazwijmy **miastami wschodnimi**, zaś leżące na zachód od węzła wschodniego — **miastami zachodnimi**.

Symulacja

Ponieważ nic już więcej nie wyczytamy z treści zadania, nadszedł czas na poszukiwanie rozwiązania. Zastanówmy się najpierw, jak mogłaby wyglądać organizacja ruchu kolejowego w takiej sieci, po czym spróbujemy zasymulować ten proces w naszym grafie.

Trasa każdego pociągu dzieli się niewątpliwie na dwie fazy — w pierwszej pociąg dojeżdża do węzła zachodniego (i w tej fazie może być zmuszony do postojów we wschodnich miastach-węzłach), w drugiej zaś już bez postojów może jechać do wybranego miasta na granicy zachodniej. Postawiony wymóg, by pociągi dojechały do różnych miast na granicy zachodniej, oznacza, że trzeba jeszcze określić, do którego miasta powinien udać się każdy pociąg wyjeżdżający z węzła zachodniego. To jest jednak oczywiste: ostatni pociąg powinien pojechać do najbliższego, przedostatni do drugiego najbliższego, a pierwszy do p-tego najbliższego takiego miasta.

Pozostaje jeszcze pytanie, jaką strategię należy przyjąć w pierwszej fazie. Odpowiedź brzmi banalnie i można ją ująć w trzech słowach: byle do przodu. W ewidentny sposób

opłaca się by pociąg jechał, jeśli tylko potrzebny mu tor jest wolny. Gdy wiele pociągów potrzebuje pewnego toru, to oczywiście może przejechać tylko jeden, a pozostałe muszą czekać do następnego dnia. Ponieważ nie jest powiedziane, który pociąg ma dojechać do którego miasta na granicy zachodniej, możemy przepuścić dowolny z oczekujących pociągów.

To już pozwala nam naszkicować pierwszą wersję rozwiązania:

```
1:  znajdź węzeł zachodni oraz dla każdego miasta wschodniego v znajdź
    następujące po nim na trasie do węzła zachodniego miasto nast[v]
2:  dla każdego miasta v policz odległości dist[v] od węzła zachodniego
3:  uporządkuj miasta wschodnie w kolejności rosnących odległości
    od węzła zachodniego
4:
5:  ile_jeszcze_jedzie := p;
6:  dzien := 0;
7:  odp := 0;
8:  while ile_jeszcze_jedzie > 0 do begin
9:    Inc(dzien);
10:   for v ∈ miasta wschodnie do
11:     if ile_w[v] > 0 then begin
12:       Dec(ile_w[v]); Inc(ile_w[nast[v]];
13:     end;
14:   if ile_w[węzeł zachodni] = 1 then begin
15:     odp := MAX(odp, dzien+dist[miasto docelowe tego pociągu]);
16:     ile_w[węzeł zachodni] := 0;
17:     Dec(ile_jeszcze_jedzie);
18:   end;
19: end;
```

Choć szkic ten jest daleko niepełny, bo w szczególności nie potrafimy jeszcze znaleźć węzła zachodniego, to pozwala nam ocenić tę próbę. Gołym okiem widać, że ten algorytm działa w czasie $\Omega(n^2)$, czyli jest zdecydowanie zbyt wolny.

Niewątpliwie wewnętrzna pętla **for** zamiast przeglądać wszystkie miasta wschodnie, mogłaby ograniczyć się tylko do tych, z których choć jeden pociąg chce wyruszyć na trasę. To jednak nie poprawia złożoności.

Co więc jest słabym punktem wybranego przez nas podejścia? O tym w następnej części, zatytułowanej „Rozwiązanie wzorcowe”.

Rozwiązanie wzorcowe

Wystarczy chwila zastanowienia, by na naszego wroga numer jeden wytypować symulację. Choć wiernie oddaje to, jak odbywa się ruch kolejowy, jest zbyt szczegółowa — nam wystarczy wiedzieć kiedy kolejne pociągi dojeżdżają do węzła zachodniego.

Spróbujmy teraz trochę „pogdybać”. Być może rozważenie pewnych przypadków szczególnych pozwoli nam znaleźć drogę do rozwiązania.

Gdyby... tylko jeden pociąg przewoził odpady, dzień wjazdu do węzła zachodniego byłby równy odległości (mierzonej liczbą odcinków torów) miasta na granicy wschodniej, z którego wyjechał pociąg od węzła zachodniego.

Gdyby... były dwa pociągi, moglibyśmy napotkać na pewne trudności w przypadku, gdy odległości od węzła zachodniego obu miast, z których wyjeżdżają pociągi, były takie same. Wtedy w jednym czasie próbowałyby przejechać przez wspólny odcinek toru — co najmniej jeden taki istnieje, np. gardło sieci — więc jeden z pociągów zmuszony byłby czekać. Ale potem oba mogłyby bez przeszkód dojechać do węzła zachodniego. Tym samym, jeden z pociągów dojechałby po czasie odpowiadającym odległości jaką przebył, a drugi następnego dnia. Zauważmy, że możemy na tę sytuację spojrzeć jeszcze w ten sposób, że jeden z pociągów spędza pierwszy dzień w swoim mieście na granicy wschodniej, co uniemożliwia spotkanie z drugim pociągiem, więc eliminuje też konieczność oczekiwania na zwolnienie toru.

Gdyby... powtórzyć to rozumowanie dla większej liczby pociągów? Moglibyśmy dla każdego pociągu policzyć odległość jaką musi przebyć do węzła zachodniego i potraktować te odległości jako pierwsze przybliżenia czasów dojazdu do tego węzła. Oczywiście pozostaje wymóg tego, że pociągi muszą dojeżdżać tam w różnych dniach, czyli czasy musiałyby zostać „rozrzucone": dla każdego powtarzającego się czasu zastąpilibyśmy kolejne powtórzenia najmniejszymi „niezajętymi jeszcze" czasami.

Np. $\{2,2,2,4,7\}$ przeszłoby w $\{2,3$ `(powstało z 2)`$,4,5$ `(powstało z 2)`$,7\}$.

Nie ulega wątpliwości, że tak otrzymany plan jest optymalny, o ile tylko można go zrealizować. Na szczęście to nie przedstawia większych trudności, bo wystarczy, by każdy z pociągów odpowiednią liczbę dni (różnica pomiędzy przydzielonym mu momentem wjazdu, a odległością od węzła zachodniego) stał w mieście, z którego wyrusza. Dzięki temu żadne dwa pociągi nie spotkają się na trasie i nie będzie w związku z tym już żadnych opóźnień.

Możemy zatem zapisać szkic rozwiązania wzorcowego:

```
1:  znajdź węzeł zachodni
2:  dla każdego miasta granicznego policz odległości dist[v] od węzła zachodniego
3:
4:  ile_zachodnich := 0;
5:  for v ∈ miasta na granicy zachodniej do begin
6:    Inc(ile_zachodnich);
7:    na_zachodzie[ile_zachodnich] := dist[v];
8:  end;
9:  Sort(na_zachodzie);
10:
11: ile_wschodnich := 0;
12: nastepny_dzien := 0;
13: for v ∈ miasta, z których ruszają pociągi do begin
14:   Inc(ile_wschodnich);
15:   na_wschodzie[ile_wschodnich] := dist[v];
16: end;
17: Sort(na_wschodzie);
18: { a teraz będzie rozrzucanie }
19: for i := 1 to p do begin
20:   na_wschodzie[i] := MAX(na_wschodzie[i], nastepny_dzien);
```

```
21:     nastepny_dzien := na_wschodzie[i]+1;
22:   end;
23:
24:   odp := 0;
25:   for i := 1 to p do
26:     odp := MAX(odp, na_wschodzie[i]+na_zachodzie[p-i+1]);
```

To podejście jest już zdecydowanie lepsze od pierwszego. Wprawdzie nie powiedzieliśmy jeszcze jak znaleźć węzeł zachodni, ale reszta algorytmu ma złożoność taką jak sortowanie, czyli $O(n \log n)$, a nawet $O(n)$, jeśli zastosujemy sortowanie kubełkowe, gdyż znalezienie odległości od węzła zachodniego można wykonać dowolnym przeszukiwaniem w czasie $O(n)$.

Znajdowanie węzła zachodniego

Zajmiemy się teraz ostatnim, pomijanym do tej pory, aspektem naszego zadania, mianowicie problemem znajdowania węzła zachodniego. Na początek poszukajmy jakichś szczególnych własności tego węzła z punktu widzenia jadącego pociągu z odpadami.

Oto dość prosta i silna cecha: jest to taki węzeł, że przybywszy do niego z miasta na granicy wschodniej, można dojechać z niego do każdego miasta na granicy zachodniej, ale do żadnego na granicy wschodniej (nie rozważamy tras wielokrotnie odwiedzających pewne miasto). W przypadku gdyby istniało kilka takich wierzchołków, za węzeł zachodni może zostać uznany dowolny z nich.

Spostrzeżenie to można wykorzystać na kilka sposobów. Ciekawym pomysłem jest np. „zwijanie" drzewa sieci od strony wschodniej: odcinamy wszystkie miasta, z wyjątkiem granicznych zachodnich, o stopniu 1 oraz te, którym w tym postępowaniu stopień się zmniejszy do 1. Intuicyjnie wydaje się, że proces ten powinien doprowadzić do odcięcia wszystkich miast wschodnich i w efekcie do „wskazania" węzła zachodniego. W rzeczywistości poprawne zaimplementowanie tego pomysłu wymaga dużej dozy uwagi, choć prowadzi do krótkiego i efektywnego algorytmu. Takie rozwiązanie zostało przedstawione w pliku `wsc.cpp`.

Inną metodą, prowadzącą do wykrycia węzła zachodniego, jest przejście drzewa w głąb z dowolnego miasta na granicy wschodniej i określanie dla każdego wierzchołka, czy wszystkie znajdujące się poniżej liście odpowiadają miastom z granicy zachodniej. Najpłycej znajdujący się w drzewie przeszukiwania wierzchołek, dla którego to zajdzie, będzie dobrym węzłem zachodnim.

Oba pomysły można zrealizować w liniowej złożoności — w ten sposób upadł ostatni bastion, z którym musieliśmy się zmierzyć w tym zadaniu.

Inne rozwiązania

Inne rozwiązania, to przede wszystkim nieoptymalne realizacje podejścia wzorcowego. Wśród nich są zarówno działające w czasie $O(n \log n)$ rozwiązania „prawie optymalne" (np. wykorzystujące algorytm Quicksort zamiast sortowania kubełkowego), jak i dalece nieoptymalne kwadratowe, odpowiadające naszym pierwszym próbom znalezienia rozwiązania.

Nawet te można jednak „poprawić”, stosując dość prostą, lecz często skuteczną optymalizację. Jeżeli ze stacji odjeżdżają pociągi w kolejnych dniach, to nie pamiętamy ich wszystkich, lecz tylko dzień odjazdu pierwszego i ich liczbę. Działa to szybko, gdy pociągów jest wiele w porównaniu do liczby torów, a wszystkie mają do przebycia podobne odległości. Choć algorytm nadal jest kwadratowy, może przejść wiele testów, zwłaszcza losowych — rozwiązanie to zostało zaimplementowane w pliku `wsc2.cpp`.

Oprócz rozwiązań poprawnych istnieje oczywiście całe mnóstwo niepoprawnych, z naszego punktu widzenia zupełnie bezwartościowych.

Testy

Zadanie testowane było na zestawie 13 danych testowych.

Testy zostały wygenerowane przy pomocy programu `wscingen.cpp`. Testy 1.-10. generowane są poprzez wygenerowanie dwóch drzew podobnych rozmiarów, a następnie połączenie ich korzeni przy pomocy krawędzi — gardła sieci. Rozważmy las drzew, na początku składający się z pewnej ilości drzew jednoelementowych. Drzewo możemy wygenerować łącząc pewną liczbę drzew ze zbioru w większe drzewo. Taką operację łączenia powtarzamy aż zostanie tylko jedno drzewo. Generowane testy dzielą się na dwie klasy, ze względu na sposób łączenia.

FIFO Do tworzonego węzła podpinane są kolejne w numeracji węzły, a nowy węzeł dostaje kolejny wolny numer — w ten sposób możemy otrzymać pełne drzewo.

Random Do tworzonego węzła podpinane są losowe węzły.

Testy 11.-13. zostały zaprojektowane specjalnie po to, by odrzucić „udoskonalone” algorytmy kwadratowe. Odległości od miast na granicy wschodniej do węzła zachodniego są w nich bardzo różne, w związku z czym pociągi rzadziej jeżdżą w kolejnych dniach po tym samym torze. W testach tych zachodni węzeł jest bezpośrednio połączony z wszystkimi miastami na zachodniej granicy.

W teście 11. miasta wschodnie tworzą jedną długą linię kolejową, na której co drugie miasto jest graniczne (czyli każdym odcinkiem pociągi jeżdżą co drugi dzień). W teście 12. znajduje się drzewo binarne, przy czym jeden syn jest podpięty bezpośrednio, a drugi za pośrednictwem pewnej liczby miast. W teście 13. mamy jedną długą linię, do której w losowych miejscach podpięte są miasta ze wschodniej granicy.

Nr	typ testu	n	p	wynik
0	Test przykładowy.	9	2	4
1	Mały test poprawnościowy – Random.	16	4	6
2	Pełne drzewa binarne – FIFO.	510	100	114
3	Średni test poprawnościowy – Random.	12235	5000	5003
4	Średni test poprawnościowy – FIFO.	33520	10000	10019
5	Test wydajnościowy, pełne drzewo binarne – FIFO.	131070	10000	10030
6	Test wydajnościowy – FIFO.	122261	10000	10012
7	Test wydajnościowy – Random.	199781	40000	40011
8	Test wydajnościowy – Random.	366762	100000	100004
9	Test wydajnościowy – FIFO.	453318	100000	100011
10	Test wydajnościowy – Random.	1000000	100000	100008
11	Test wydajnościowy o specyficznej strukturze.	1000000	300000	700000
12	Test wydajnościowy o specyficznej strukturze.	1000000	32769	508438
13	Test wydajnościowy o specyficznej strukturze.	1000000	100000	799998