Marcin Kubica
Treść zadania, Opracowanie

Anna Niewiarowska
Program

OI, Etap III, dzień próbny, 27.03.2007

Koleje

Bajtockie Koleje Państwowe (BKP) stanęły przed koniecznością restrukturyzacji i redukcji sieci linii kolejowych. Po jej dokładnym przeanalizowaniu zdecydowano, które stacje kolejowe mają być zlikwidowane, a które mają pozostać. Zdecydowano się także zredukować sieć linii kolejowych. Trzeba jeszcze wybrać, które linie kolejowe mają być zlikwidowane, a które mają pozostać.

Sieć linii kolejowych składa się z odcinków torów łączących stacje kolejowe. Wiadomo, że z każdej stacji kolejowej da się dojechać do każdej innej (potencjalnie odwiedzając stacje pośrednie). Odcinki torów są dwukierunkowe. Między każdą parą stacji może być co najwyżej jeden odcinek torów. Każdy taki odcinek torów charakteryzuje się **kosztem utrzymania** *— dodatnią liczbą całkowitą. Odcinki torów, które mają pozostać, muszą być tak wybrane, aby:*

- *dało się przejechać między każdymi dwiema stacjami, które mają pozostać,*
- *ich sumaryczny koszt utrzymania był mały (może być co najwyżej dwukrotnie większy od najmniejszego kosztu, jaki da się uzyskać, zachowując poprzedni warunek).*

Wszystkie pozostałe odcinki torów zostaną zlikwidowane. Linie kolejowe, które mają pozostać, mogą przebiegać przez stacje, które zostaną zlikwidowane.

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia opis sieci linii kolejowych oraz stacje, które mają pozostać,*
- *wyznaczy, które odcinki torów mają pozostać, a które mają być zlikwidowane,*
- *wypisze na standardowe wyjście, które odcinki torów mają pozostać oraz poda ich sumaryczny koszt utrzymania.*

Wejście

W pierwszym wierszu standardowego wejścia zapisane są dwie dodatnie liczby całkowite n i m, $2 \leqslant n \leqslant 5\ 000$, $1 \leqslant m \leqslant 500\ 000$ ($m \leqslant \frac{n(n-1)}{2}$), oddzielone pojedynczym odstępem. Liczba n to liczba stacji kolejowych, a m — liczba odcinków torów. Stacje kolejowe są ponumerowane od 1 do n. W kolejnych m wierszach są opisane odcinki torów kolejowych, po jednym w wierszu. W każdym z tych wierszy są zapisane trzy dodatnie liczby całkowite a, b i u, $1 \leqslant a, b \leqslant n$, $a \neq b$, $1 \leqslant u \leqslant 100\ 000$. Liczby a i b to numery stacji, które łączy dany odcinek torów, a u to jego koszt utrzymania. W $m+2$ wierszu zapisany jest ciąg liczb całkowitych pooddzielanych pojedynczymi odstępami. Pierwsza z nich to p — liczba stacji, które mają pozostać ($1 \leqslant p \leqslant n$, $p \cdot m \leqslant 15\,000\,000$). Dalej w tym wierszu wymienione są numery tych stacji w kolejności rosnącej.

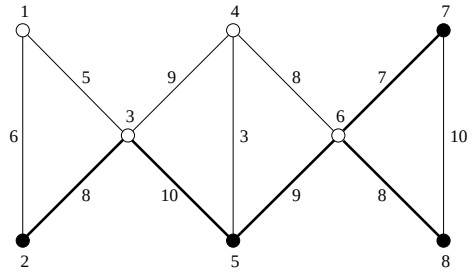
Wyjście

W pierwszym wierszu standardowego wyjścia program powinien wypisać dwie liczby całkowite c i k oddzielone pojedynczym odstępem, gdzie c jest sumarycznym kosztem utrzymania pozostawionych odcinków, a k — liczbą tych odcinków. W każdym z kolejnych k wierszy powinny znaleźć się dwie liczby a_i oraz b_i oddzielone pojedynczym odstępem — numery stacji połączonych przez pozostawione odcinki torów. Sumaryczny koszt utrzymania odcinków torów może być co najwyżej **dwukrotnie większy** *od najmniejszego kosztu, możliwego do uzyskania.*

Przykład

Dla danych wejściowych:

```
8 11
1 2 6
3 1 5
2 3 8
3 4 9
3 5 10
5 4 3
5 6 9
6 4 8
6 8 8
6 7 7
8 7 10
4 2 5 7 8
```



poprawnym wynikiem jest:

```
42 5
2 3
3 5
5 6
6 7
6 8
```

Rozwiązanie

Analiza problemu

Problem przedstawiony w treści zadania jest znany jako grafowe sformułowanie[1] problemu *drzew Steinera*. Dany jest graf nieskierowany $G = (V, E)$, w którym każdej krawędzi $e \in E$ przypisana jest dodatnia waga w_e, oraz zbiór wierzchołków $V' \subseteq V$. Oznaczmy przez $c(X)$ (dla $X \subseteq E$) sumę wag krawędzi należących do zbioru X. Szukamy takiego drzewa złożonego z krawędzi $T \subseteq E$, że:

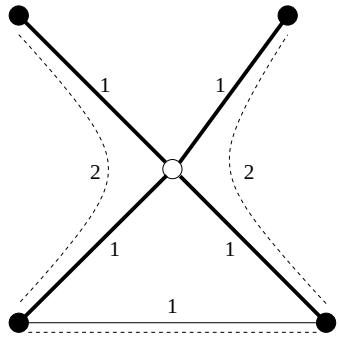
[1] W innym sformułowaniu problemu drzew Steinera mamy dany zbiór punktów na płaszczyźnie i szukamy najkrótszej sieci dróg łączących te punkty. Taką wersję problemu także można przedstawić w postaci grafu, choć może to wymagać wprowadzenia nieskończonego zbioru wierzchołków.

- każde dwa wierzchołki ze zbioru V' są połączone pewną ścieżką złożoną z krawędzi należących do zbioru T,
- $c(T)$ jest minimalne.

Niestety, problem ten jest NP-zupełny[2], co oznacza, że nie jest znany algorytm znajdujący takie drzewo w czasie wielomianowym. Na szczęście, w niniejszym zadaniu nie wymagamy, aby znalezione drzewo było optymalne (nawet nie wymagamy, aby to było drzewo)! Dopuszczamy, aby suma wag krawędzi była większa od najmniejszej możliwej — ale nie więcej niż dwukrotnie. Rozwiązanie zadania polega więc na napisaniu algorytmu *aproksymacyjnego*, czyli obliczającego rozwiązanie bliskie optymalnemu. W problemach aproksymacyjnych za miarę przybliżenia przyjmuje się często stosunek wielkości znalezionego rozwiązania do wielkości rozwiązania optymalnego. W tym przypadku stosunek ten nie może przekraczać 2 — takie przybliżenie jest nazywane 2-aproksymacją.

Problem drzew Steinera przypomina problem znajdowania minimalnego drzewa rozpinającego. Jednak ten ostatni nie jest NP-zupełny i potrafimy go rozwiązywać w czasie wielomianowym. Służą do tego na przykład algorytmy zachłanne Kruskala i Prima ([19], p. 24).

Spróbujemy użyć algorytmu znajdującego minimalne drzewo rozpinające do przybliżenia optymalnego drzewa Steinera. Najpierw jednak musimy zmienić graf, w którym będziemy szukać minimalnego drzewa rozpinającego. Będzie to graf pełny $G' = (V', E')$, gdzie dla każdej pary wierzchołków $u, v \in V'$ wagą krawędzi (u, v) jest długość najkrótszej (tj. najlżejszej) ścieżki w grafie G. W grafie G' znajdujemy minimalne drzewo rozpinające T' — jego krawędzie to ścieżki w oryginalnym grafie G. Niech T będzie zbiorem tych krawędzi z E, które należą do którejkolwiek krawędzi-ścieżki w drzewie T' (w razie potrzeby usuwamy powtórzenia krawędzi T występujących w kilku krawędziach-ścieżkach T'). Graf T łączy wszystkie wierzchołki z V', natomiast nie musi być drzewem — może okazać się, że łącząc ścieżki z T', uzyskamy cykle — jest to jednak dopuszczalne! Zauważmy, że $c(T) \leqslant c(T')$, gdyż w drzewie T' wagi krawędzi powtarzających się na wielu ścieżkach są liczone wielokrotnie, a w drzewie T są liczone tylko raz.

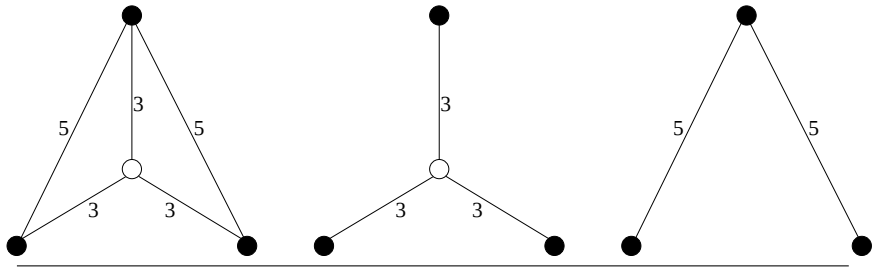


Rys. 1: Drzewo Steinera w grafie G i minimalne drzewo rozpinające w grafie G'.

Minimalne drzewo rozpinające T' w grafie G' nie musi odpowiadać drzewu Steinera w grafie G. Wynika to stąd, że w poszukiwaniu tych drzew kierujemy się nieco innymi

[2]Problemy NP-zupełne są przedstawione np. w [19, 21].

kryteriami optymalności — koszt drzewa Steinera to zwykła suma wag krawędzi, a koszt drzewa T' to suma długości ścieżek, wśród których mogą występować powtórzenia krawędzi. Ilustruje to następujący przykład. Łączna waga krawędzi w drzewie Steinera wynosi 9,



Rys. 2: Dany graf G, drzewo Steinera i minimalne drzewo rozpinające w grafie G'.

natomiast minimalne drzewo rozpinające w grafie G' ma wagę 10. Pokażemy, że choć waga minimalnego drzewa rozpinającego w grafie G' może być większa niż waga drzewa Steinera w grafie G, to jednak nie więcej niż dwukrotnie. Oznacza to, że T' będzie wystarczającym dla naszych celów przybliżeniem rozwiązania optymalnego.

Lemat 1 (na podstawie [36]) *Niech $G = (V,E)$ będzie danym grafem, w którym każdej krawędzi $e \in E$ przypisana jest dodatnia waga w_e, oraz niech $V' \subseteq V$ będzie danym zbiorem wierzchołków. Niech także G' będzie grafem skonstruowanym jak powyżej. Oznaczmy przez T drzewo Steinera rozpinające w G wierzchołki ze zbioru V', a przez T' minimalne drzewo rozpinające w G'. Wówczas zachodzi nierówność $c(T') \leqslant 2 \cdot c(T)$.*

Dowód Rozważmy eulerowskie obejście drzewa T, tzn. taki cykl, który prowadzi wyłącznie przez krawędzie z T, odwiedza każdy wierzchołek drzewa T i przez każdą krawędź z T przechodzi dokładnie dwukrotnie. Dla każdego wierzchołka $v \in V'$ wybierzmy jedno jego wystąpienie w takim cyklu. Następnie podzielmy ten cykl na ścieżki, przecinając go w wybranych wystąpieniach wierzchołków z V'. W ten sposób otrzymujemy ścieżki $\sigma_1, \sigma_2, \ldots, \sigma_p$. Zauważmy, że:

$$c(\sigma_1) + c(\sigma_2) + \cdots + c(\sigma_p) = 2 \cdot c(T)$$

Zauważmy też, że ścieżki $\sigma_1, \sigma_2, \ldots, \sigma_p$ odpowiadają krawędziom w grafie G' (dokładniej, są nie lżejsze od odpowiadających im krawędzi G'), tworzącym w nim cykl przechodzący przez każdy wierzchołek dokładnie raz (tzw. cykl Hamiltona). Czyli $\sigma_1, \sigma_2, \ldots, \sigma_{p-1}$ odpowiadają ścieżce przechodzącej przez każdy wierzchołek G' dokładnie raz (tzw. ścieżce Hamiltona). Ścieżka to jednak szczególny przypadek drzewa. Tak więc ścieżki $\sigma_1, \sigma_2, \ldots, \sigma_{p-1}$ mają wagę nie mniejszą od wagi pewnego drzewa rozpinającego w G', które z kolei nie może mieć wagi mniejszej niż T' — minimalne drzewo rozpinające dla G'.

Mamy więc ostatecznie:

$$c(T') \leqslant c(\sigma_1) + c(\sigma_2) + \cdots + c(\sigma_{p-1}) \leqslant c(\sigma_1) + c(\sigma_2) + \cdots + c(\sigma_p) = 2 \cdot c(T).$$

■

Rozwiązanie oparte o konstrukcję minimalnego drzewa rozpinającego

Algorytm opisany w poprzednim punkcie jest dość prosty koncepcyjnie, ale jego efektywna implementacja może przysporzyć kłopotów. Najpierw musimy skonstruować graf G', czyli wyznaczyć najkrótsze ścieżki między wszystkimi parami wierzchołków ze zbioru V'. Prostym rozwiązaniem jest zastosowanie algorytmu Floyda-Warshalla dla całego grafu G, a następnie zawężenie wyników do wierzchołków z G'. Jednak za tę prostotę płacimy złożonością czasową rzędu $O(n^3)$. Możemy też p-krotnie zastosować algorytm Dijkstry, wyznaczając odległości od kolejnych wierzchołków z V' do wszystkich pozostałych wierzchołków. Oczywiście, oprócz odległości musimy też być w stanie szybko zrekonstruować najkrótsze ścieżki realizujące te odległości. Zakładając, że użyjemy kolejki priorytetowej o logarytmicznym czasie wykonywania operacji, sumaryczna złożoność czasowa takiego rozwiązania to $O(p \cdot (n+m) \cdot \log n)$. Zważywszy, że $p \cdot m \leqslant 15\,000\,000$, jest to istotnie lepsze rozwiązanie. Jednak w obu przypadkach trzeba zapamiętać długości krawędzi grafu G'. Prosty rachunek wykazuje, że zmieścimy się w zadanym limicie pamięci tylko dla p maksymalnie rzędu 2000.

Oba algorytmy wyznaczania minimalnego drzewa rozpinającego: Kruskala i Prima (zakładając, że w algorytmie Prima także użyjemy kolejki priorytetowej o logarytmicznym czasie wykonywania operacji) działają w czasie $O(p^2 \log p)$. Nie jest to więc dominująca faza obliczeń. Pozostało jeszcze przejrzenie zbioru krawędzi tworzących ścieżki oraz usunięcie powtórzeń. Ponieważ mamy $p-1$ ścieżek, każda nie dłuższa niż $n-1$ krawędzi, to wymaga to czasu rzędu $O(p \cdot n + m)$. W rezultacie otrzymujemy algorytm działający w czasie $O(p \cdot (n+m) \cdot \log n)$, jednak o zbyt dużej złożoności pamięciowej.

Okazuje się, że możemy nie konstruować grafu G' *a priori*. W algorytmie Prima budujemy drzewo rozpinające w sposób zachłanny, zaczynając od pojedynczego wierzchołka i dodając do niego kolejne krawędzie, za każdym razem wybierając najlżejszą krawędź, która nie domyka żadnego cyklu. Zamiast najpierw konstruować graf G', a potem budować rozpinające go drzewo, można „w locie" wyznaczać długości interesujących nas krawędzi-ścieżek w grafie G'. Oznaczmy przez D zbiór wierzchołków grafu G', które (w danym momencie) należą do budowanego drzewa. Konstrukcję drzewa zaczynamy od pojedynczego wierzchołka $v \in V'$, $D = \{v\}$. Następnie zachłannie rozszerzamy budowane drzewo. Szukamy najkrótszej krawędzi w grafie G' łączącej którykolwiek wierzchołek $v \in D$ z dowolnym wierzchołkiem $w \in V' \setminus D$. Ponieważ nie wyznaczyliśmy wcześniej grafu G', musimy w tym momencie znaleźć najkrótszą ścieżkę w grafie G łączącą którykolwiek wierzchołek $v \in D$ z dowolnym wierzchołkiem $w \in V' \setminus D$. Możemy tutaj zastosować algorytm Dijkstry, rozpoczynając przeszukiwanie grafu G równocześnie ze wszystkich wierzchołków ze zbioru D i kończąc je w momencie osiągnięcia pierwszego wierzchołka należącego do $V' \setminus D$.

Stosując w rozwiązaniu kolejkę priorytetową o logarytmicznym koszcie operacji, otrzymujemy taką samą złożoność czasową jak poprzednio, czyli $O(p \cdot (n+m) \cdot \log n)$, lecz złożoność pamięciowa jest dużo mniejsza! Musimy oczywiście pamiętać dany graf G, jednak wszystkie dodatkowe struktury danych, w tym kolejka priorytetowa wierzchołków, mają rozmiar $O(n)$. Tak więc łączna złożoność pamięciowa jest rzędu $O(n+m)$ i pozwala zmieścić się w zadanym limicie pamięciowym.

Rozwiązanie wzorcowe

Rozwiązanie wzorcowe jest podobne do opisanego powyżej zastosowania algorytmu Prima, z wyszukiwaniem „w locie" (za pomocą algorytmu Dijkstry) najkrótszych ścieżek w grafie G. Różnica między tym rozwiązaniem a opisanym poprzednio, polega na uproszczeniu procedury rozbudowy drzewa D o kolejne krawędzie-ścieżki. Zamiast dodawać do drzewa w każdym kroku najkrótszą ścieżkę łączącą pewien wierzchołek z $V' \cap D$ (należący już do budowanego drzewa) z pewnym wierzchołkiem z $V' \setminus D$ (nienależącym jeszcze do budowanego drzewa), dodajemy najkrótszą ścieżkę łączącą pewien wierzchołek z $V \cap D$ (*wierzchołek ten nie musi należeć do* V') z pewnym wierzchołkiem z $V' \setminus D$. Algorytm ten możemy wyrazić w postaci następującego pseudokodu:

```
1:  Wybieramy dowolny v ∈ V';
2:  D := {v};
3:  R := ∅;
4:  while V' ⊈ D do begin
5:    σ := najkrótsza ścieżka w G łącząca dowolny v ∈ D z w ∈ V' \ D;
6:    D := D ∪ wierzchołki(σ);
7:    R := R ∪ krawędzie(σ);
8:  end
9:  return R;
```

Zbiór D zawiera wierzchołki, a zbiór R — krawędzie budowanego drzewa. Na pierwszy rzut oka algorytm ten powinien dawać przynajmniej tak dobrą aproksymację, jak algorytm opisany w poprzednim punkcie. Należy to jednak pokazać formalnie.

Lemat 2 *Niech (D,R) będzie drzewem utworzonym przez algorytm wzorcowy, rozpinającym w danym grafie G wierzchołki z V'. Niech $c_{\min}$ będzie sumą wag krawędzi w minimalnym drzewie rozpinającym w grafie G'. Wówczas $c(R) \leqslant c_{\min}$.*

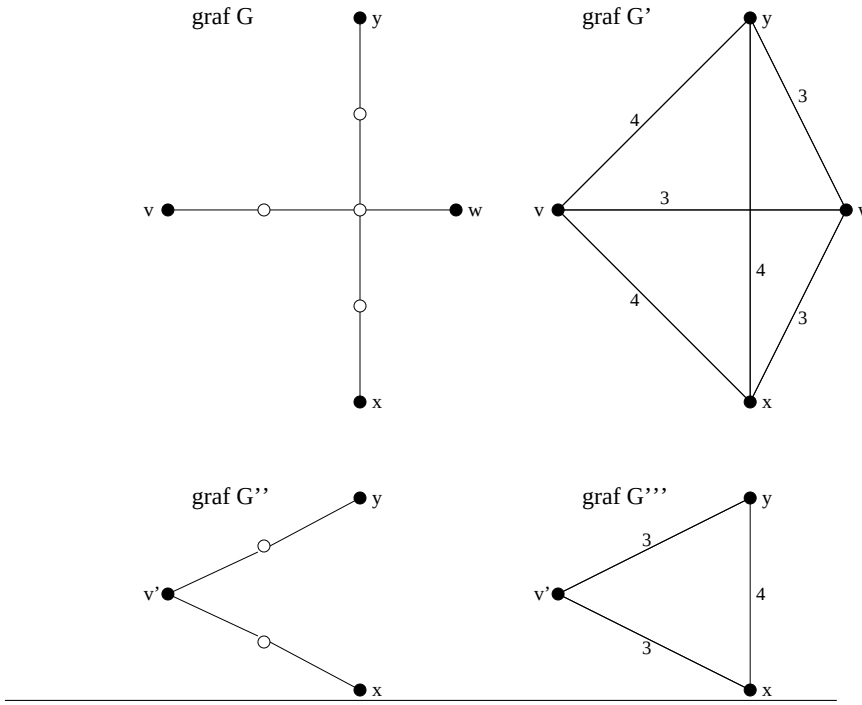
Dowód Dowód przebiega przez indukcję po liczbie wybranych wierzchołków $p = |V'|$.

1. Jeśli $p = 1$, to $V' = \{v\}$ i oba drzewa nie zawierają ani jednej krawędzi. Zatem $c(R) = c_{\min} = 0$.

2. Załóżmy, że $p > 1$. Niech v będzie wierzchołkiem wybranym z V' na początku algorytmu wzorcowego. Niech σ będzie ścieżką wybraną w pierwszym kroku pętli **while** w algorytmie wzorcowym. Łączy ona v z pewnym wierzchołkiem $w \in V' \setminus \{v\}$.

   Rozważmy graf $G'' = (V'', E'')$ powstały z G przez sklejenie wszystkich wierzchołków na ścieżce σ; wierzchołek powstały w wyniku sklejenia oznaczmy przez v'. Analogicznie, oznaczmy przez $G''' = (V''', E''')$ graf powstały ze sklejenia w grafie G' wierzchołków v i w w jeden wierzchołek v'. Intuicyjnie, para grafów (G'', G''') odpowiada parze grafów (G, G') po wykonaniu jednego kroku przez każdy z algorytmów. Przykład konstrukcji grafów G'' i G''' dla zadanych grafów G oraz G' jest zilustrowany na rys. 3.

   Niech (D', R') będzie drzewem wyznaczonym przez algorytm wzorcowy dla grafu G'' i wybranego na początku wierzchołka v'. Sklejenie wierzchołków leżących na ścieżce σ nie wpływa na dalsze działanie algorytmu wzorcowego, gdyż traktuje on



Rys. 3: Przykładowy wygląd grafów G i odpowiadającego mu grafu G' oraz postać grafów G'' i G'''. Dla uproszczenia zakładamy, że krawędzie grafów G oraz G'' mają wagi jednostkowe.

wierzchołki do tej pory skonstruowanego drzewa tak, jakby były sklejone. Stąd, $c(R) = c(\sigma) + c(R')$. Niech dalej $c'_{\min}$ będzie sumą wag krawędzi w minimalnym drzewie rozpinającym w grafie G'''. Z poprawności algorytmu Prima wiemy, że w grafie G' istnieje minimalne drzewo rozpinające, które zawiera krawędź odpowiadającą ścieżce σ. Stąd $c_{\min} = c(\sigma) + c'_{\min}$. Niech wreszcie $H = (G'')'$ będzie grafem skonstruowanym z G'' w ten sam sposób, w jaki z grafu G skonstruowaliśmy G', czyli grafem najkrótszych ścieżek między wierzchołkami wybranymi G''. Można zauważyć, że graf H musi wyglądać dokładnie tak samo jak G''', lecz potencjalnie może mieć krótsze niektóre krawędzie (2 zamiast 3 w przypadku dwóch krawędzi na rys. 3). Oznaczmy przez $c''_{\min}$ wagę minimalnego drzewa rozpinającego H. Wówczas $c''_{\min} \leqslant c'_{\min}$.

Ponieważ $|V'''| < |V'| = p$, to możemy skorzystać z założenia indukcyjnego dla grafów G'' oraz H, otrzymując $c(R') \leqslant c''_{\min}$. Nierówność ta implikuje, że $c(R') \leqslant c'_{\min}$. Stąd ostatecznie

$$c(R) = c(\sigma) + c(R') \leqslant c(\sigma) + c'_{\min} = c_{\min}.$$

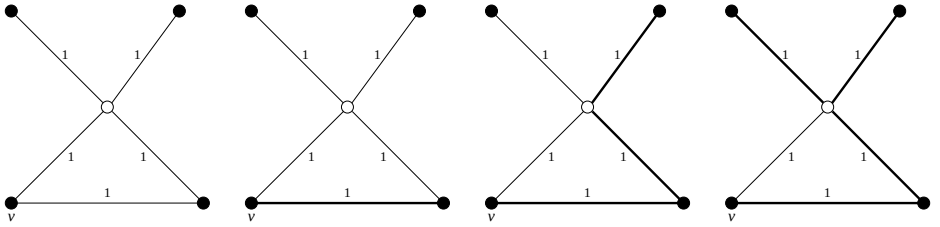
■

Z lematu tego wynika poprawność algorytmu wzorcowego.

Twierdzenie 3 *Niech (D,R) będzie drzewem utworzonym przez algorytm wzorcowy, rozpinającym w danym grafie G wierzchołki ze zbioru V'. Niech c_S będzie sumą wag krawędzi w drzewie Steinera rozpinającym w grafie G wierzchołki ze zbioru V'. Wówczas $c(R) \leqslant 2 \cdot c_S$.*

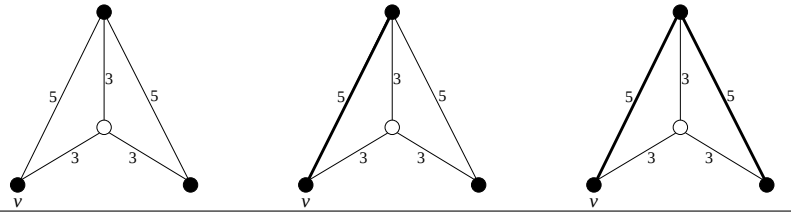
Dowód Z Lematu 2 wynika, że $c(R) \leqslant c_{\min}$, gdzie $c_{\min}$ jest sumą wag krawędzi w minimalnym drzewie rozpinającym w grafie G'. Z Lematu 1 wiemy, że $c_{\min} \leqslant 2c_S$. Stąd, $c(R) \leqslant 2 \cdot c_S$. ■

Prześledźmy działanie algorytmu wzorcowego na pokazanych wcześniej dwóch przykładowych grafach. Okazuje się, że algorytm wzorcowy może znaleźć drzewo Steinera, podczas gdy minimalne drzewo rozpinające w grafie G' ma większą wagę. Niestety nie zawsze się



Rys. 4: Konstrukcja drzewa Steinera o wadze 4 przez algorytm wzorcowy.

tak dzieje. Oto przykład grafu, dla którego algorytm wzorcowy znajduje dokładnie drzewo rozpinające w grafie G', a jego waga jest wyższa od wagi drzewa Steinera.



Rys. 5: Algorytm wzorcowy nie znajduje drzewa Steinera (waga 9), lecz drzewo o wadze 10.

W wierszu 5 algorytmu wzorcowego stosujemy algorytm Dijkstry do znalezienia w grafie G najkrótszej ścieżki łączącej dowolny $v \in D$ z $w \in V' \setminus D$. Możemy trochę poprawić efektywność tego algorytmu, jeżeli nie będziemy w każdym kroku pętli niezależnie wywoływać algorytmu Dijkstry, lecz wykorzystamy zawartość kolejki priorytetowej z poprzedniej iteracji do wykonania następnej iteracji. W kolejce priorytetowej trzymamy wierzchołki, które nie należą do drzewa, a ich priorytet odpowiada temu, co wiemy o ich odległości od konstruowanego drzewa. Początkowo konstruowane drzewo to pojedynczy wierzchołek $v \in V'$, $D = \{v\}$. Wierzchołki incydentne z nim mają priorytet równy długości łączących je krawędzi, a pozostałe wierzchołki mają priorytet równy ∞. Równocześnie, dla każdego wierzchołka o skończonym priorytecie pamiętamy ścieżkę, takiej długości jak jego priorytet, łączącą go z konstruowanym drzewem.

W kolejnych krokach algorytmu Dijkstry, za każdym razem wyciągamy z kolejki wierzchołek o najmniejszym priorytecie; oznaczmy go przez w. Oczywiście $w \notin D$.

Spośród wierzchołków znajdujących się w kolejce, jest to wierzchołek najbliżej położony konstruowanego drzewa. Możliwe są dwa przypadki:

- Jeżeli $w \notin V'$, to korygujemy priorytety wierzchołków incydentnych z w, uwzględniając ścieżki prowadzące przez niego.
- Jeżeli $w \in V' \setminus D$, to spośród wierzchołków z $V' \setminus D$ jest to wierzchołek położony najbliżej konstruowanego drzewa. Rozszerzamy więc konstruowane drzewo o najkrótszą ścieżkę łączącą je z w.

  Po rozszerzeniu drzewa nie musimy ponownie tworzyć kolejki priorytetowej z wierzchołkami spoza drzewa. Wystarczy skorygować istniejącą kolejkę — musimy poprawić priorytety wierzchołków incydentnych z dodawaną ścieżką, uwzględniając fakt, że stała się ona częścią konstruowanego drzewa.

  Dodatkowo, musimy zwrócić uwagę na wierzchołki $v \notin V'$, których priorytet zmienia się już po usunięciu z kolejki priorytetowej. Jeśli po rozszerzeniu drzewa priorytet v (czyli jego odległość od drzewa) spadł poniżej wartości z chwili, gdy był on wyjmowany z kolejki, to wierzchołek v należy ponownie wstawić do kolejki i rozważyć.

Wierzchołki ze zbioru V' są tylko raz wstawiane do kolejki priorytetowej i tylko raz są z niej wyjmowane. Wierzchołki spoza V' mogą być wielokrotnie wstawiane i usuwane z niej — jednak nie więcej niż p razy każdy. Jednak dla większości danych wejściowych, liczba wstawianych i usuwanych wierzchołków jest istotnie mniejsza. Przyjmując, że operacje na kolejce priorytetowej działają w czasie $O(\log n)$, uzyskujemy pesymistyczną złożoność czasową $O(p \cdot (n+m) \cdot \log n)$. Jest to taka sama złożoność jak w przypadku poprzednio opisanego rozwiązania, jednak dla większości danych algorytm wzorcowy działa istotnie szybciej. Algorytm ten został zaimplementowany w programach `kol.cpp` i `kol0.pas`.

Inne rozwiązania

Oprócz przedstawionych wcześniej rozwiązań wielomianowych, można się było spodziewać rozwiązań znajdujących drzewo Steinera, ale działających w czasie wykładniczym. Na przykład, algorytm taki mógłby rozważać wszystkie możliwe zbiory wierzchołków z $V \setminus V'$, które wchodzą w skład drzewa Steinera i dla tak ograniczonego grafu znajdować minimalne drzewo rozpinające (jeśli takowe istnieje). Rozwiązanie takie działa w czasie $O(2^{n-p} \cdot m \log m)$. Nie należy się więc spodziewać, że przejdzie ono jakiekolwiek testy poza małymi testami poprawnościowymi. Zostało ono zaimplementowane w programach `kols1.cpp` i `kols3.pas`.

Testy

Rozwiązania zawodników były sprawdzane na 10 testach. Zostały one podsumowane w poniższej tabelce.

Dużym problemem przy przygotowywaniu testów był fakt, że do właściwej oceny rozwiązań zawodników konieczna jest znajomość wielkości drzewa Steinera. W ogólności

jest to problem NP-zupełny. W przypadku testów nr 1, 2, 3 i 5 optymalny wynik można wyznaczyć w rozsądnym czasie za pomocą algorytmu wykładniczego. Dodatkowo, testy nr 3, 4, 6 i 9 mają na tyle prostą strukturę, że optymalny wynik można wyznaczyć ręcznie. W testach nr 7 i 10 mamy $p = 3$. Tak więc problem wyznaczenia drzewa Steinera sprowadza się do znalezienia takiego jego „środka", z którego rozchodzące się trzy najkrótsze ścieżki dają najlepszy wynik. Taki problem można już prosto rozwiązać, korzystając na przykład z algorytmu Floyda-Warshalla. Pozostał test nr 8. Składa się on z niedużego podgrafu losowego, dla którego można wyznaczyć optymalny wynik za pomocą algorytmu wykładniczego, oraz krawędzi i wierzchołków, co do których wiadomo, że nie są częścią drzewa Steinera, gdyż nie łączą one żadnych dwóch wierzchołków z V'.

Nazwa	n	m	koszt	Opis
kol1.in	25	161	32 143	prosty test poprawnościowy, losowy
kol2.in	41	443	220 078	prosty test poprawnościowy, losowy
kol3.in	100	540	774	test średniej wielkości, drzewo Steinera zawiera wszystkie wierzchołki
kol4.in	900	1 334	2 975	test średniej wielkości, drzewo Steinera zawiera jedynie wierzchołki z V'
kol5.in	999	14 800	5 068 199	test średniej wielkości, losowy
kol6.in	3 300	6 000	207 000	test wydajnościowy
kol7.in	3 000	405 181	2 670	test wydajnościowy
kol8.in	5 000	11 008	118 733	test wydajnościowy (maksymalna liczba wierzchołków)
kol9.in	4 841	9 504	50 600	test wydajnościowy
kol10.in	4 000	372 802	3 693	test wydajnościowy