Jakub Wojtaszczyk
Treść zadania, Opracowanie

Łukasz Bieniasz-Krzywiec
Program

Dostępna pamięć: 64 MB.

OI, Etap II, dzień drugi, 11.02.2010

Teleporty

Król Bajtazar włada całym układem słonecznym, w którym znajduje się n planet. Planety te są ponumerowane od 1 do n. Jego pałac znajduje się na planecie nr 1, zaś jego baza wojskowa na planecie nr 2. Dawno temu Bajtazar wybudował teleport, który w ciągu dwustu pięćdziesięciu minut (czyli nieco ponad czterech godzin) jest w stanie przenieść go z planety nr 1 na planetę nr 2 lub z powrotem.

Obecnie dostępna jest bardziej zaawansowana technologia, która umożliwia tworzenie teleportów zapewniających transport między dwiema planetami w czasie jednej godziny (teleporty takie są z natury dwukierunkowe). Niektóre planety układu są już połączone takimi teleportami. Teleporty te umożliwiają przebycie trasy pomiędzy planetami 1 i 2, na szczęście nie szybciej niż prywatny teleport Bajtazara (co oczywiście zagrażałoby jego bezpieczeństwu).

Obecnie każda para planet, która nie jest jeszcze bezpośrednio połączona teleportem nowej generacji, złożyła wniosek o utworzenie takiego teleportu łączącego te planety. Bajtazar chce zgodzić się na jak największą liczbę takich wniosków, ale tak, aby ciągle nie dało się pokonać trasy z planety nr 1 do planety nr 2 szybciej niż przy pomocy jego prywatnego teleportu. Pomóż mu określić, na ile nowych teleportów może się zgodzić.

Wejście

W pierwszym wierszu standardowego wejścia znajdują się dwie liczby całkowite n oraz m ($2 \leqslant n \leqslant 40\,000$, $0 \leqslant m \leqslant 1\,000\,000$), oddzielone pojedynczym odstępem, oznaczające odpowiednio liczbę planet w królestwie Bajtazara oraz liczbę istniejących teleportów nowej generacji. W kolejnych m wierszach opisane są już istniejące teleporty. W każdym z tych wierszy znajdują się po dwie liczby całkowite a_i i b_i ($1 \leqslant a_i < b_i \leqslant n$) oddzielone pojedynczym odstępem, oznaczające, że istnieje teleport łączący planety a_i i b_i. Każda para liczb występuje na wejściu co najwyżej raz. Możesz założyć, że istniejące teleporty umożliwiają przedostanie się z planety nr 1 na planetę nr 2, ale nie w czasie krótszym niż 250 minut.

Wyjście

Twój program powinien wypisać w pierwszym wierszu standardowego wyjścia jedną liczbę całkowitą, oznaczającą maksymalną liczbę nowych teleportów, na utworzenie których może zgodzić się Bajtazar.

Przykład

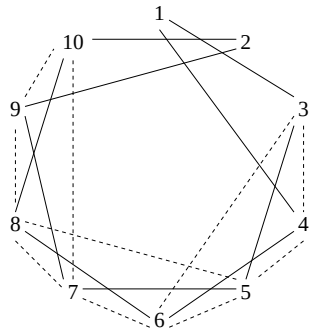
Dla danych wejściowych:

```
10 10
1 3
3 5
5 7
```

```
7 9
2 9
1 4
4 6
6 8
8 10
2 10
```

poprawnym wynikiem jest:

```
10
```



Linią ciągłą na rysunku zaznaczono istniejące teleporty, a przerywaną te, na budowę których Bajtazar może pozwolić.

Rozwiązanie

Wprowadzenie

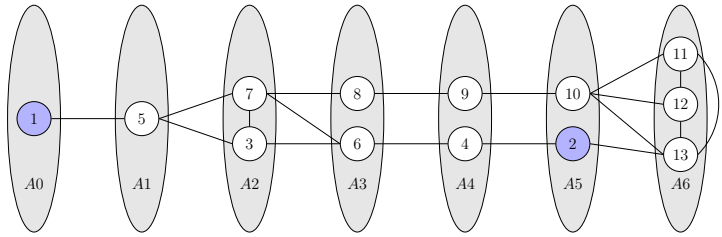
Postawiony przed nami problem można łatwo wyrazić w terminologii grafowej. Mamy dany graf nieskierowany $G = (V, E)$, którego wierzchołki odpowiadają planetom w układzie słonecznym. Dwa wierzchołki są połączone krawędzią, gdy między odpowiadającymi im planetami istnieje teleport nowej generacji. Nasz graf, zgodnie z treścią zadania, ma n wierzchołków ($n \geqslant 2$) i m krawędzi. Graf ten będziemy nazywać grafem *wejściowym*. Wiadomo, że wierzchołki o numerach 1 i 2 są połączone ścieżką, bo istniejące teleporty pozwalają na przedostanie się z planety nr 1 na planetę nr 2. Co więcej, odległość wierzchołków 1 i 2 w naszym grafie (czyli długość najkrótszej łączącej je ścieżki) jest równa co najmniej 5.

Powiemy, że krawędź $e \in E$ jest *niebezpieczna*, jeśli przebiega przez nią jakakolwiek ścieżka długości 4 lub mniejszej łącząca wierzchołki 1 i 2 (takich krawędzi mieć nie chcemy). Graf H nazwiemy *nadgrafem* grafu G, jeśli ma ten sam zbiór wierzchołków oraz każda krawędź grafu G jest też krawędzią grafu H. Poszukujemy zatem nadgrafu H wejściowego grafu G, który nie zawiera niebezpiecznych krawędzi i ma największą możliwą liczbę krawędzi. Dowolny nadgraf H grafu wejściowego, który nie zawiera żadnej niebezpiecznej krawędzi, będziemy nazywać grafem *wynikowym*.

Szybko można spostrzec, że z uwagi na możliwy duży rozmiar danych wejściowych sprawdzenie wszystkich opcji (czyli wszystkich możliwych grafów wynikowych) nie wchodzi w grę. Ponadto, nietrudno dojść do wniosku, że kluczowe jest zrozumienie wpierw, które krawędzie można dodawać, a następnie — jeśli mamy wybór — jak wybierać, by otrzymać największą możliwą liczbę krawędzi w grafie wynikowym. Poniżej sformułujemy obserwacje, które odpowiadają na te pytania. Warto podkreślić, że jakkolwiek wszystkie rozwiązania koniec końców musiały wyglądać dość podobnie, to drogi rozumowania do nich prowadzące mogły wyglądać różnie. Podana niżej droga nie powinna być zatem traktowana jako jedyna możliwa.

Podział na warstwy i analiza zadania

Spójrzmy na dowolny graf wynikowy H. Podzielimy jego wierzchołki na *warstwy* wedle odległości od wierzchołka 1. Przez A_k oznaczmy zbiór tych wierzchołków v, dla których najkrótsza ścieżka z 1 do v ma długość dokładnie k. Oczywiście 2 leży w warstwie A_k dla pewnego skończonego $k \geqslant 5$ i wszystkie warstwy od A_0 do A_k są niepuste (zawierają co najmniej po jednym wierzchołku z najkrótszej ścieżki z 1 do 2). Zauważmy, że jeśli w grafie H istnieje wierzchołek v, który nie jest połączony ścieżką z wierzchołkiem 1, to nie jest on też połączony z wierzchołkiem 2 (bo 1 i 2 łączy ścieżka). Możemy zatem dodać krawędź $1v$ i będzie ona bezpieczna (kwestią optymalności tego wyboru zajmiemy się dalej). Będziemy więc rozważać tylko grafy wynikowe H, w których wszystkie wierzchołki są połączone ścieżką z 1, czyli leżą w jakiejś warstwie.



Rys. 1: Przykładowy graf z podziałem na warstwy.

Zaczniemy od prostej obserwacji, która uzasadnia sens podziału na warstwy.

Obserwacja 1. Rozważmy dwa wierzchołki u i v, które leżą w tej samej warstwie lub w dwóch sąsiednich warstwach, lecz nie są połączone krawędzią. Jeżeli dodamy tę krawędź, to podział grafu H na warstwy nie zmieni się.

Dowód: Istotnie — ani odległość od 1 do u, ani do v nie zmieni się (bo różnica tych odległości już była nie większa niż 1, więc nie zyskujemy na dodaniu tej krawędzi). Odległość od 1 do żadnego innego wierzchołka również się nie zmieni, bo początkowy fragment dowolnej ścieżki zawierającej krawędź uv możemy podmienić na nie dłuższą ścieżkę do v (ew. do u, jeśli szliśmy tą krawędzią w drugą stronę) niezawierający uv. ■

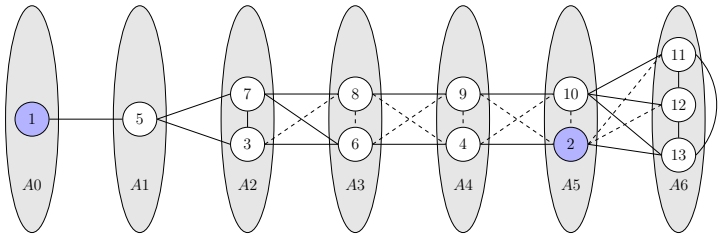
W szczególności, krawędź opisana w Obserwacji 1 jest bezpieczna (bo 2 wciąż leży w warstwie o numerze nie mniejszym niż 5).

Możemy zatem założyć, że w grafie wynikowym każde dwa wierzchołki leżące w tej samej warstwie lub w dwóch sąsiednich warstwach są połączone krawędzią. Z drugiej strony, dwa wierzchołki w warstwach odległych o przynajmniej 2 nie mogą być połączone krawędzią (bo dawałoby to krótszą ścieżkę do wierzchołka z dalszej warstwy). Stąd, znając podział grafu wynikowego na warstwy, możemy obliczyć liczbę

jego krawędzi — będzie to[1]:

$$\frac{1}{2}\sum_{k=0}^{\infty}\big(|A_k|\cdot(|A_{k-1}|+|A_k|+|A_{k+1}|-1)\big). \tag{1}$$

Czynnik 1/2 jest w powyższym wyrażeniu skutkiem tego, że każdą krawędź liczymy dwukrotnie — na jej początku i na jej końcu, zaś odjęcie jedynki w nawiasie odpowiada temu, że wierzchołek nie jest połączony sam ze sobą. Wprowadziliśmy też dla wygody zapisu sztuczne oznaczenie A_{-1}, ten zbiór jest oczywiście pusty. Będziemy teraz szukać takiego nadgrafu grafu wejściowego H, dla którego wartość (1) jest jak największa. Graf, w którym dodaliśmy wszystkie krawędzie, których dodanie umożliwia Obserwacja 1, będziemy nazywać *pełnym grafem wynikowym*.



Rys. 2: Przykładowy, pełny graf wynikowy dla grafu wejściowego z poprzedniego rysunku.

Rozważmy dowolny wierzchołek $v \in A_k$, $k > 1$, w pewnym pełnym grafie wynikowym H. Powiemy, że można go *przesunąć* do warstwy A_{k-1}, jeśli żadna z krawędzi łączących v z wierzchołkami warstwy A_{k+1} nie występowała w grafie wejściowym G. Przesunięcie polega wtedy na usunięciu wszystkich krawędzi łączących v z A_{k+1}, przesunięciu v do A_{k-1} oraz dodaniu krawędzi do każdego wierzchołka z A_{k-2}. Analogicznie definiujemy przesunięcie z A_k do A_{k+1} dla $k > 0$. Przykładowo, na rysunku 2 można przesunąć dowolny z wierzchołków 11, 12, 13 z warstwy A_6 do warstwy A_5 (w sumie to mało ciekawy przykład, ale lepszy rydz niż nic).

Otrzymany w wyniku przesunięcia pojedynczego wierzchołka graf będzie poprawnym grafem wynikowym, o ile nie przesunęliśmy wierzchołka 2 z A_5 do A_4. Faktycznie, utrzymujemy niezmiennik, który mówi, że najkrótsza ścieżka z wierzchołka 1 do dowolnego wierzchołka w warstwie A_k ma długość dokładnie k, a podczas przesuwania nie usuwamy krawędzi pochodzących z grafu wejściowego G.

Rozważmy teraz graf H, który jest jednym z grafów wynikowych o największej możliwej liczbie krawędzi, oraz jego podział na warstwy. Spójrzmy, w jakich warstwach mogą leżeć jego wierzchołki.

[1]Suma (1) formalnie jest nieskończona, jednak zawsze będzie zawierać jedynie skończenie wiele niezerowych składników.

Obserwacja 2. W grafie wynikowym o największej możliwej liczbie krawędzi warstwy A_k dla $k \geqslant 6$ są puste, zaś warstwy A_0 i A_5 składają się z dokładnie jednego wierzchołka (odpowiednio 1 i 2).

Dowód: Załóżmy, że A_k to ostatnia niepusta warstwa dla pewnego $k \geqslant 6$ i $v \in A_k$. Jako że w H istnieje ścieżka z wierzchołka 1 do dowolnego innego wierzchołka oraz $k \geqslant 6$, to warstwy A_{k-1} oraz A_{k-2} są niepuste. Z drugiej strony, warstwa A_{k+1} jest pusta z założenia, zatem możemy przesunąć v z A_k do A_{k-1}. Przy tym przesunięciu nie usuwamy żadnych krawędzi (bo A_{k+1} jest pusta), zaś dodamy przynajmniej jedną krawędź z v do A_{k-2}. Zatem otrzymany graf wynikowy ma więcej krawędzi niż H, co przeczy założeniu o H.

Oczywiście, w warstwie A_0 może być, z definicji, tylko wierzchołek 1. Jeśli w warstwie A_5 jest jakiś wierzchołek oprócz 2, to, podobnie jak powyżej, dowolny z nich możemy przesunąć do A_4. ■

Wykonanie tych przesunięć na grafie z rysunku 2 spowoduje przesunięcie wierzchołków 10, 11, 12 i 13 do warstwy A_4.

Obserwacja 3. Istnieje graf wynikowy o największej możliwej liczbie krawędzi, w którym w warstwie A_1 leżą tylko te wierzchołki, które sąsiadowały z wierzchołkiem 1 w grafie wejściowym G, zaś w A_4 tylko te, które sąsiadowały z 2 w G.

Dowód: Załóżmy, że mamy jakiś „dodatkowy" wierzchołek w warstwie A_1. Jeżeli przesuniemy go do A_2 (możemy, bo z założenia nie sąsiadował on z 1 w grafie G, a 1 to jedyny wierzchołek w warstwie A_0), to stracimy jedną krawędź do A_0, a zyskamy przynajmniej jedną krawędź do A_3 (warstwa A_3 jest niepusta, bo przechodzi przez nią ścieżka z wierzchołka 1 do wierzchołka 2). Zatem graf po takim przesunięciu będzie miał nie mniej krawędzi. Podobnie możemy przesunąć wszystkie „dodatkowe" wierzchołki z A_4 do A_3. ■

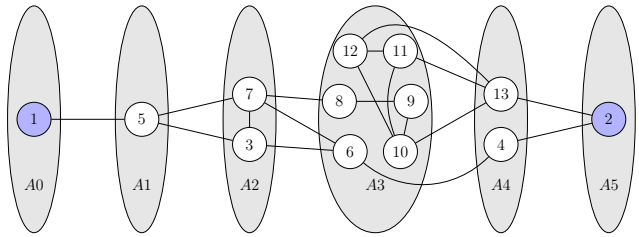
Zastosowanie tych przesunięć na naszym przykładowym grafie przepchnie wierzchołki 9, 10, 11 i 12 do warstwy A_3.

Znamy już więc dokładnie zawartość wszystkich warstw grafu H poza A_2 i A_3. Co więcej wierzchołki, które w G sąsiadowały z jakimkolwiek wierzchołkiem z warstwy A_1 (czyli — na przykładowym rysunku — wierzchołki 7 i 3), muszą leżeć w A_2, zaś wierzchołki, które sąsiadowały z jakimkolwiek wierzchołkiem, który umieścimy, na mocy Obserwacji 3, w warstwie A_4 (czyli, na przykładowym rysunku, wierzchołki 6, 10, 11 i 12, sąsiadujące z 4 lub 13), muszą leżeć w A_3. Niech U oznacza zbiór pozostałych wierzchołków (na przykładowym rysunku $U = \{8, 9\}$).

Obserwacja 4. Niech H będzie grafem z Obserwacji 3. Jeżeli $|A_1| \geqslant |A_4|$, to można założyć, że wierzchołki z U leżą w A_2. W przeciwnym razie można założyć, że $U \subset A_3$.

Dowód jest identyczny jak poprzednio — przeniesienie wierzchołka z A_2 do A_3 usuwa $|A_1|$ krawędzi, zaś dodaje $|A_4|$.

Wiemy zatem, jak będzie wyglądał optymalny graf, i jesteśmy gotowi do implementacji rozwiązania.



Rys. 3: Optymalny podział grafu z rysunku 1 na warstwy.

Rozwiązanie wzorcowe

Zauważmy, że skoro graf G ma do 40 000 wierzchołków, to graf wynikowy H może mieć nawet 800 000 000 krawędzi, czyli więcej, aniżeli chcemy trzymać w pamięci i konstruować. Szczęśliwie, jeżeli wyznaczymy optymalny podział na warstwy, to używając wzoru (1), będziemy w stanie wyznaczyć liczbę krawędzi wynikowego grafu. Zatem zadanie sprowadza się do wyznaczenia optymalnego podziału wierzchołków grafu G na warstwy.

Wiemy, że warstwy powyżej A_5 będą puste, zaś warstwy A_0 i A_5 będą składać się z wierzchołków o numerach odpowiednio 1 i 2. Wierzchołki, które muszą znaleźć się w warstwach A_1, A_2, A_3 i A_4 przed zastosowaniem Obserwacji 4, można wyznaczyć, przeszukując graf wszerz[2]. Pozostaje nam pewna liczba „wolnych" wierzchołków, które możemy przydzielić do warstwy A_2 lub A_3 i które przydzielamy w zależności od liczności warstw A_1 i A_4, zgodnie z Obserwacją 4.

Przykładowy algorytm realizujący to zadanie wygląda następująco:

1: **Algorytm wzorcowy**
2: *Wejście:* Graf nieskierowany G w postaci list sąsiedztwa.
3: *Wyjście:* Rozmiar maksymalnego rozszerzenia grafu G.
4: $n := |V|;\ m := |E|;\ a := 0;\ b := 0;\ c := 0;\ d := 0;$
5: { $t_1[i]$ to odległość wierzchołka i-tego od wierzchołka numer 1 }
6: $t_1 := BFS(G, 1);$
7: { $t_2[i]$ to odległość wierzchołka i-tego od wierzchołka numer 2 }
8: $t_2 := BFS(G, 2);$
9: **for** $i := 1$ **to** n **do begin**
10: **if** $t_1[i] = 1$ **then** $a := a + 1$
11: **else if** $t_1[i] = 2$ **then** $b := b + 1;$
12: **if** $t_2[i] = 1$ **then** $c := c + 1$
13: **else if** $t_2[i] = 2$ **then** $d := d + 1;$
14: **end**
15: $e := n - 1 - a - b - 1 - c - d;$
16: { $a = |A_1|,\ b = |A_2|,\ c = |A_4|,\ d = |A_3|,\ e = |U|$ }

[2]Algorytm BFS, jego opis można znaleźć np. w książce [21].

17: **return** $n(n-1)/2-m-(n-1-a)-a(1+c+d)-b(1+c)-d-e(1+\min(a,c))$;

Złożoność czasowa i pamięciowa tego algorytmu wynosi $O(n+m)$.

Warto zauważyć, że w powyższym algorytmie można zastosować drobne usprawnienie. W procedurze BFS nie musimy przeszukiwać całego grafu. Wystarczy, że zatrzymamy się na wierzchołkach odległych o co najwyżej 2 od źródła.

Implementacje rozwiązania wzorcowego wraz z przedstawionym usprawnieniem znajdują się w plikach `tel.cpp` oraz `tel0.pas`.

Inne rozwiązania

Rozwiązania nieefektywne

Najprostszym siłowym rozwiązaniem tego zadania jest wyznaczenie wszystkich możliwych wyborów par planet, dla których zezwolimy na zbudowanie teleportu. Dla każdego takiego wyboru sprawdzamy, czy jest dopuszczalny (tj. czy nie istnieje zbyt krótka trasa z planety 1 na planetę 2). Jeśli tak, to porównujemy jego liczność z najlepszym dotychczas znalezionym i zapamiętujemy maksimum.

Powyższy algorytm działa w czasie $O(n^2 \cdot 2^{n^2/2})$ i zużywa $O(n^2)$ pamięci, a zatem jest bez szans na rozwiązanie zadania wobec ograniczeń podanych w treści. Można starać się go poprawiać różnymi sposobami stwierdzania, jakich krawędzi na pewno nie warto dodawać (przykładowo, sprawdzać poprawność po każdej dodanej krawędzi, a nie na końcu), natomiast należy oczekiwać, że takie rozwiązania wciąż będą zbyt wolne, a także zużyją zbyt dużo pamięci.

Implementacje tego rozwiązania znajdują się w plikach `tels0.cpp` i `tels1.pas`. W plikach `tels2.cpp` i `tels3.pas` znajdują się rozwiązania wzbogacone o heurystykę pozwalającą na odrzucanie części krawędzi. Rozwiązania te przechodziły 1 na 12 testów.

Ciekawe rozwiązanie bardzo niepoprawne

Na problem można też spojrzeć z nieco innej strony. Odwróćmy pytanie i zastanówmy się, ile najmniej krawędzi trzeba usunąć z grafu pełnego, aby wyeliminować wszystkie ścieżki między wierzchołkami 1 i 2 o długościach mniejszych niż 5.

Niech S będzie zbiorem wszystkich ścieżek prostych między wierzchołkami 1 i 2 o długościach mniejszych niż 5 w grafie pełnym $G_f = (V, E_f)$. Z każdą krawędzią $e \in E_f \setminus E$ wiążemy zbiór $S_e \subseteq S$ wszystkich ścieżek, które przez nią przechodzą. Podzbiory te oczywiście nie muszą być rozłączne. Rozwiązanie polega na znalezieniu najmniejszego (w sensie liczności) zbioru $F \subseteq E_f \setminus E$, takiego, że $S = \bigcup_{e \in F} S_e$. Wówczas maksymalne rozszerzenie grafu G to $E_f \setminus (E \cup F)$.

W ten sposób sprowadziliśmy nasze zadanie do problemu pokrycia zbioru (ang. *Set Cover*), który jest problemem NP-zupełnym. Pierwsze rozwiązanie niepoprawne wykorzystuje to podejście i implementuje następujący algorytm heurystyczny.

1: **Heurystyczne rozwiązanie problemu pokrycia zbioru**
2: *Wejście:* Graf nieskierowany G w postaci macierzy sąsiedztwa.
3: *Wyjście:* Moc maksymalnego rozszerzenia grafu G.

```
4:  Wyznacz zbiór S oraz zbiory S_e dla e ∈ E_f \ E.
5:  f := 0;
6:  while S ≠ ∅ do begin
7:    A := najliczniejszy ze zbiorów S_e;
8:    S := S \ A;
9:    foreach e ∈ E_f \ E do
10:     S_e := S_e \ A;
11:   f := f + 1;
12: end
13: return n(n − 1)/2 − m − f;
```

Można go zaimplementować, tak aby działał w pesymistycznym czasie $O(n^4)$ i zużywał $O(n^3)$ pamięci. Jest to algorytm $H_s = \sum_{k=1}^{s} \frac{1}{k} \leqslant \ln(s) + 1$ aproksymacyjny, gdzie s to moc najliczniejszego ze zbiorów S_e. To oznacza, że wybrane przez niego pokrycie jest co najwyżej $\ln(s) + 1$ razy zbyt liczne. Dowód tego faktu i bardziej szczegółowy opis tego algorytmu Czytelnik może znaleźć np. w książce [21].

Implementacja tego rozwiązania znajduje się w pliku `telb1.cpp`. Nie przechodziło ono żadnego z zestawów testów.

Ulepszone rozwiązanie niepoprawne

Poprzednie rozwiązanie można w łatwy sposób trochę ulepszyć. Zmiana polega na tym, że w pierwszym kroku wybieramy do pokrycia wszystkie zbiory odpowiadające krawędziom, których na pewno nie można dołączyć do grafu (nawet pojedynczo). To usprawnienie nie zmienia pesymistycznej złożoności algorytmu, ale powoduje, że rozwiązanie jest bardzo szybkie dla niektórych klas grafów. Znacznie poprawia również dokładność udzielanych przez algorytm odpowiedzi.

To rozwiązanie przechodziło jeden zestaw testów (przykładowa implementacja w pliku `telb2.cpp`).

Jeszcze inne rozwiązanie niepoprawne

W tym rozwiązaniu do generowanego rozszerzenia nie dodajemy jedynie krawędzi, które (dodane pojedynczo) powodują zmniejszenie odległości między wierzchołkami 1 i 2 poniżej 5. Rozwiązanie to jest niepoprawne, gdyż łatwo może się zdarzyć, że dodanie dwóch krawędzi spowoduje powstanie ścieżki długości 2 pomiędzy wierzchołkami 1 i 2, pomimo iż każda z krawędzi z osobna nie skracała odległości z 1 do 2.

Implementacja tego rozwiązania znajduje się w pliku `telb3.cpp`. Ten program nie przechodził żadnego zestawu testów.

Testy

Programy zgłoszone przez uczestników były sprawdzane na dwunastu zestawach testowych. Każdy zestaw składał się z kilku testów. W poniższym zestawieniu przyjęliśmy oznaczenia zgodne z opisem rozwiązania, tzn.:

- n — liczba wierzchołków grafu,
- m — liczba krawędzi grafu,
- a — liczba wierzchołków w odległości 1 od wierzchołka 1,
- b — liczba wierzchołków w odległości 2 od wierzchołka 1,
- c — liczba wierzchołków w odległości 1 od wierzchołka 2,
- d — liczba wierzchołków w odległości 2 od wierzchołka 2.

Wszystkie testy poza `tel1c.in` i testami z odpowiedzią 0 były losowymi grafami o opisanych wyżej parametrach.

Zestawienie testów:

Nazwa	n	m	a	b	c	d	Opis
tel1a.in	6	5	1	1	1	1	poprawnościowy
tel1b.in	7	5	1	1	1	1	poprawnościowy
tel1c.in	7	6	1	1	1	1	poprawnościowy
tel1d.in	7	7	1	1	1	1	poprawnościowy
tel1e.in	8	9	2	1	2	1	poprawnościowy
tel2a.in	43	101	5	7	1	1	poprawnościowy
tel2b.in	54	300	10	17	3	3	poprawnościowy
tel2c.in	70	231	2	1	3	1	poprawnościowy
tel3a.in	80	79	1	1	1	1	poprawnościowy
tel3b.in	80	700	30	10	20	7	poprawnościowy
tel4a.in	102	3 120	25	25	25	25	wydajnościowy
tel4b.in	102	3 125	25	25	25	25	wydajnościowy; odpowiedź 0
tel4c.in	150	2 000	40	1	41	10	wydajnościowy
tel5a.in	202	12 538	51	49	48	52	wydajnościowy
tel5b.in	202	12 547	51	49	48	52	wydajnościowy; odpowiedź 0
tel5c.in	500	12 345	123	20	123	20	wydajnościowy
tel6a.in	1 000	100 000	1	300	23	77	wydajnościowy
tel6b.in	5 000	300 000	30	10	3 000	29	wydajnościowy
tel7a.in	10 000	200 000	10	3 000	2 300	770	wydajnościowy
tel7b.in	12 000	500 001	10 000	1	2	1	wydajnościowy
tel8a.in	15 000	300 000	3 001	3 003	3 002	2 999	wydajnościowy
tel8b.in	19 999	700 543	4	2	3	1	wydajnościowy
tel9a.in	25 000	600 000	5 007	4 988	1	10 000	wydajnościowy

Nazwa	n	m	a	b	c	d	Opis
tel9b.in	25 001	400 010	3	1	10 000	1	wydajnościowy
tel10a.in	30 000	700 000	5 007	4 988	1	10 000	wydajnościowy
tel10b.in	35 000	800 001	100	2	10 000	99	wydajnościowy
tel11a.in	40 000	1 000 000	7 690	8 030	8 008	7 777	wydajnościowy
tel11b.in	40 000	39 999	1	1	1	1	wydajnościowy
tel11c.in	39 999	999 999	20 000	100	101	99	wydajnościowy
tel12a.in	40 000	1 000 000	123	256	199	203	wydajnościowy
tel12b.in	40 000	1 000 000	5 000	4 000	9 000	303	wydajnościowy

Zawody III stopnia

opracowania zadań