Jakub Wojtaszczyk
Treść zadania, Opracowanie

Bartosz Górski
Program

Dostępna pamięć: 128 MB.

OI, Etap I, 18.10–15.11.2010

Konspiracja

Wroga Bitocja napadła zdradziecko na Bajtocję i okupuje znaczną jej część. Król Bajtocji, Bajtazar, chce zorganizować na okupowanych terenach ruch oporu. Bajtazar rozpoczął od wytypowania osób, które utworzą strukturę organizacyjną ruchu oporu. Należy je podzielić na dwie grupy: **konspiratorów**, *którzy będą prowadzić bezpośrednią działalność na terenie okupowanym, oraz* **grupę wsparcia**, *która będzie działać z terytorium wolnej Bajtocji.*

Pojawił się jednak pewien problem — podział taki musi spełniać następujące warunki:

- *Każde dwie osoby z grupy wsparcia powinny się znać — zagwarantuje to, że będą stanowiły zwartą i zgraną grupę.*
- *Konspiratorzy nie powinni się znać nawzajem.*
- *Żadna z grup nie może być pusta, tzn. musi być przynajmniej jeden konspirator i jedna osoba w grupie wsparcia.*

Bajtazar zastanawia się, ile jest różnych sposobów podziału wytypowanych osób na dwie grupy zgodnie z powyższymi warunkami, a przede wszystkim, czy taki podział jest w ogóle możliwy. Jako że sam nie całkiem umie sobie z tym poradzić, poprosił Cię o pomoc.

Wejście

W pierwszym wierszu standardowego wejścia znajduje się jedna liczba całkowita n ($2 \leqslant n \leqslant 5\,000$), oznaczająca liczbę osób zaangażowanych w organizację ruchu oporu. Osoby te są ponumerowane od 1 do n. W kolejnych n wierszach opisane jest, które osoby się znają. W i-tym z tych wierszy znajduje się opis znajomych osoby nr i, złożony z liczb całkowitych pooddzielanych pojedynczymi odstępami. Pierwsza z tych liczb, k_i ($0 \leqslant k_i \leqslant n-1$), oznacza liczbę znajomych osoby nr i. Dalej w wierszu znajduje się k_i liczb całkowitych $a_{i,1}, a_{i,2}, \ldots, a_{i,k_i}$. Liczby $a_{i,j}$ spełniają $1 \leqslant a_{i,j} \leqslant n$, $a_{i,j} \neq i$ oraz są podane w kolejności rosnącej. Możesz założyć, że jeśli w ciągu liczb a_i występuje liczba x, to także w ciągu liczb a_x występuje liczba i.

Wyjście

W pierwszym i jedynym wierszu standardowego wyjścia Twój program powinien wypisać jedną liczbę całkowitą: liczbę sposobów, na które osoby mające utworzyć ruch oporu mogą zostać podzielone na grupę wsparcia i grupę działającą na terenach okupowanych. Jeżeli nie istnieje żaden podział wytypowanych osób na dwie grupy spełniający podane warunki, wówczas poprawnym wynikiem jest 0.

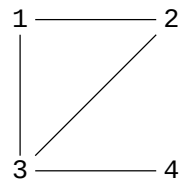
Przykład

Dla danych wejściowych:

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

poprawnym wynikiem jest:

```
3
```



Wyjaśnienie do przykładu: *Dla tej grupy spiskowców są możliwe trzy podziały na grupy. Grupę konspiratorów mogą stanowić uczestnicy o numerach 1 i 4, o numerach 2 i 4 lub sam uczestnik o numerze 4.*

Rozwiązanie

Wprowadzenie

W zadaniu mamy dany graf nieskierowany o n wierzchołkach i m krawędziach. Każdy wierzchołek odpowiada jednemu z uczestników ruchu oporu. Pomiędzy dwoma wierzchołkami jest krawędź, jeśli ci uczestnicy znają się wzajemnie. Naszym celem jest obliczenie, na ile sposobów można podzielić zbiór wierzchołków na dwa zbiory tak, aby spełnione były następujące warunki:

- każde dwa wierzchołki pierwszego zbioru są połączone krawędzią (taki zbiór wierzchołków nazywa się *kliką*)
- pomiędzy żadnymi dwoma wierzchołkami drugiego zbioru nie ma krawędzi (taki zbiór wierzchołków nazywa się *zbiorem niezależnym*)
- żaden z dwóch zbiorów nie jest pusty.

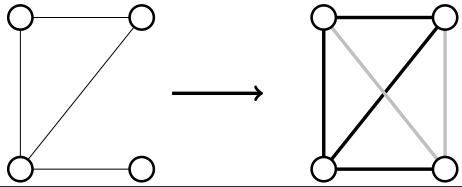
Czyli chcemy obliczyć liczbę sposobów podziału naszego grafu na dwa niepuste podgrafy, tak aby jednym z nich była klika, a drugim zbiór niezależny.

Łatwo spostrzec, że w tym zadaniu występuje symetria pomiędzy klikami a zbiorami niezależnymi. Ze względu na tę symetrię sprowadzimy nasz problem szukania podziałów do problemu szukania kolorowań wierzchołków. To uprości język opisu rozwiązania.

Na początku dopełnimy zatem nasz graf do grafu pełnego i pokolorujemy każdą z krawędzi na czarno lub szaro. Krawędzie, które znajdowały się w grafie przed dopełnieniem, pokolorujemy na czarno, a pozostałe na szaro, patrz rys. 1.

Zacznijmy od następującego spostrzeżenia.

Obserwacja 1. Jeśli jakieś dwa wierzchołki są w oryginalnym grafie połączone krawędzią, to przynajmniej jeden z nich należy do kliki. Jeśli nie są połączone krawędzią, to przynajmniej jeden z nich należy do zbioru niezależnego.

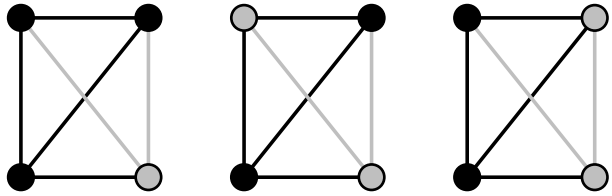


Rys. 1: Sposób pokolorowania krawędzi dla przykładowego grafu z treści zadania.

Ta obserwacja orzeka, że w zmodyfikowanym grafie chcemy znaleźć liczbę wszystkich kolorowań wierzchołków na czarno lub szaro, które spełniają poniższe warunki:

1. istnieje para wierzchołków o różnych kolorach
2. każda czarna krawędź ma co najmniej jeden z końców pokolorowany na czarno
3. każda szara krawędź ma co najmniej jeden z końców pokolorowany na szaro.

Czyli tak naprawdę chcemy, aby czarne wierzchołki tworzyły czarną klikę (żeby każda krawędź między dwoma czarnymi wierzchołkami była czarna), a szare szarą klikę, patrz też rys. 2.



Rys. 2: Poprawne kolorowania grafu z rys. 1.

W dalszych rozważaniach łatwiej będzie zapomnieć o pierwszym z powyższych warunków. Zauważmy, że wystarczy umieć zliczyć wszystkie kolorowania spełniające dwa ostatnie warunki. Zaiste — jeśli umiemy to zrobić, to możemy następnie sprawdzić, czy któreś z kolorowań „wszystko na czarno" i „wszystko na szaro" spełnia warunki 2 i 3 (nie mogą obydwa jednocześnie spełniać, jako że $n \geqslant 2$, a zatem w pokolorowanym grafie znajduje się przynajmniej jedna krawędź i musi istnieć przynajmniej jeden wierzchołek tego samego koloru co ta krawędź), a jeśli tak, to odpowiednio zmniejszyć wynik uzyskany z pominięciem pierwszego warunku. Sprawdzenie tych dwóch specjalnych kolorowań możemy wykonać w czasie $O(n^2)$, a zatem wystarczająco szybko. W dalszym ciągu będziemy zatem poszukiwać kolorowań spełniających warunki 2 i 3. Takie kolorowania nazwiemy *dobrymi*.

Rozwiązanie wzorcowe – analiza stopni

Zaczniemy od zaprezentowania efektownego rozwiązania tego zadania, które wymaga wyłącznie znajomości stopni wierzchołków w grafie, bez znajomości konkretnych kra-

wędzi. Aby na końcu uzyskać bardzo prosty algorytm, zbadamy dokładnie strukturę dobrych kolorowań.

Obserwacja 2. Załóżmy, że mamy pewne dobre kolorowanie, w którym wierzchołek v jest czarny, a wierzchołek w — szary. Wtedy liczba czarnych krawędzi wychodzących z v jest nie mniejsza niż liczba czarnych krawędzi wychodzących z w.

Dowód: Załóżmy, że w rozwiązaniu jest k czarnych wierzchołków. Wtedy z v wychodzi przynajmniej $k-1$ czarnych krawędzi (do każdego z pozostałych czarnych wierzchołków), zaś z w — co najwyżej k czarnych krawędzi, jako że szare wierzchołki muszą być połączone z w szarą krawędzią. Zatem aby z w wychodziło więcej czarnych krawędzi niż z v, musi być tak, że z w wychodzi dokładnie k czarnych krawędzi, zaś z v — dokładnie $k-1$. Ale skoro z v wychodzi $k-1$ czarnych krawędzi, to krawędź vw jest szara, a zatem z w nie może wychodzić k czarnych krawędzi, sprzeczność. ■

W dalszym ciągu liczbę czarnych krawędzi wychodzących z danego wierzchołka nazwiemy *czarnym stopniem* i oznaczać będziemy przez $cs(v)$.

Obserwacja 3. Niech dane będzie dowolne kolorowanie wierzchołków grafu na czarno i szaro, w którym jest k czarnych wierzchołków. To kolorowanie jest dobre wtedy i tylko wtedy, gdy

$$\sum_{v \text{ czarny}} cs(v) \;=\; k(k-1) + \sum_{v \text{ szary}} cs(v).$$

Dowód: Wpierw załóżmy, że mamy kolorowanie spełniające podaną wyżej tożsamość. Niech CC oznacza liczbę czarnych krawędzi łączących dwa czarne wierzchołki, SS liczbę czarnych krawędzi łączących dwa szare wierzchołki, zaś CS — liczbę czarnych krawędzi łączących dwa wierzchołki różnych kolorów. Wtedy lewa strona tożsamości jest równa $2CC + CS$, zaś suma znajdująca się po prawej stronie tożsamości to $2SS + CS$. Wstawiając to do tożsamości, otrzymujemy $2CC = k(k-1) + 2SS$.

Ale jeśli czarnych wierzchołków jest k, to krawędzi między nimi jest dokładnie $k(k-1)/2$, a zatem $CC \leqslant k(k-1)/2$. Mamy więc

$$k(k-1) \;\geqslant\; 2CC \;=\; k(k-1) + 2SS,$$

skąd $SS = 0$, zaś $CC = k(k-1)/2$. Zatem faktycznie wszystkie krawędzie łączące czarne wierzchołki są czarne, zaś wszystkie krawędzie łączące szare wierzchołki są szare.

Teraz rozważmy dowolne dobre kolorowanie. Dla takiego kolorowania mamy $CC = k(k-1)/2$ oraz $SS = 0$. To kolorowanie spełnia zatem tożsamość z obserwacji. ■

Teraz jesteśmy gotowi do sformułowania rozwiązania. Zaczynamy od wyznaczenia dla każdego wierzchołka liczby $cs(v)$ — te liczby to po prostu liczby k_i podane na wejściu. W tym rozwiązaniu nie będziemy musieli w ogóle zapamiętywać liczb a_{ij} (choć oczywiście chcemy je wczytać, by dojść do kolejnej liczby k_i). Porządkujemy liczby $cs(v)$ w kolejności nierosnącej. Następnie dla kolejnych $k = 1, 2, \ldots, n$ obliczamy sumę pierwszych k spośród liczb $cs(v)$ i sprawdzamy, czy jest ona o dokładnie

$k(k-1)$ większa od sumy pozostałych liczb cs. Tu uwaga implementacyjna — jako że $n \leqslant 5\,000$, to nasze rozwiązanie będzie wystarczająco szybkie, nawet jeśli będziemy dla każdej liczby k obliczali te sumy od zera, ale oczywiście bardziej eleganckie jest obliczanie sum dla k poprzez dodanie do pierwszej sumy wartości $cs(v_k)$ i odjęcie tej wartości od drugiej sumy.

Jeżeli okaże się, że zachodzi tożsamość z Obserwacji 3, to znaczy, że znaleźliśmy rozwiązanie. Jeśli istnieje jakieś rozwiązanie, to na mocy Obserwacji 2 znajdziemy je — wystarczy przeglądać wierzchołki w kolejności nierosnących stopni cs. Trzeba jeszcze zwrócić uwagę na to, że może zdarzyć się sytuacja, w której więcej niż jedna liczba ma tę samą wartość cs — wówczas może być więcej rozwiązań. Jako że na mocy Obserwacji 3 same liczby cs mówią nam, czy jakiś zbiór wierzchołków jest rozwiązaniem, czy nie, to jeżeli znajdziemy rozwiązanie, w którym spośród q liczb o tej samej wartości cs wzięliśmy do rozwiązania l, to wybór dowolnych innych l też da nam rozwiązanie — zatem za jednym razem znaleźliśmy $\binom{q}{l}$ rozwiązań.

Ostatecznie, gdy w wyniku dodawania kolejnych liczb w którymś momencie będzie zachodzić tożsamość z Obserwacji 3, to sprawdzamy, jak wiele wierzchołków ma tę samą wartość cs co właśnie dodany wierzchołek (oznaczamy tę liczbę przez q) oraz jak wiele z nich jest w rozwiązaniu (tę liczbę oznaczamy przez l), i do wyniku dodajemy $\binom{q}{l}$.

To daje nam poprawne rozwiązanie bez jakiejkolwiek dalszej analizy. Wnikliwy Czytelnik dostrzeże (i samodzielnie udowodni albo wywnioskuje to z rozważań przedstawionych przy omówieniu rozwiązań alternatywnych), że l może być równe tylko 0, 1, $q-1$ lub q, a zatem otrzymane wyniki nie będą zbyt duże, dzięki czemu nie musimy wcześniej obliczać liczb $\binom{q}{l}$. Dowód tego faktu pozostawiamy Czytelnikowi.

Rozwiązanie to zostało zaimplementowane w pliku `kon11.cpp`. Działa w złożoności czasowej i pamięciowej $O(n+m)$, jako że czarne stopnie wierzchołków możemy uporządkować nierosnąco za pomocą sortowania kubełkowego.

Rozwiązanie alternatywne – trójkąty

Opis tego rozwiązania zacznijmy od dwóch prostych obserwacji.

Obserwacja 4. Niech A, B i C będą trzema wierzchołkami i niech krawędzie AB i BC będą czarne, a krawędź AC szara. Wtedy w każdym dobrym kolorowaniu wierzchołek B jest czarny.

Dowód: Jeśli wierzchołek B byłby szary, to A i C musiałyby być czarne (ponieważ każda czarna krawędź musi mieć co najmniej jeden koniec czarny). Wtedy szara krawędź AC miałaby oba końce czarne, co nie jest możliwe. ■

Zauważmy zatem, że w każdym trójkącie, którego krawędzie *nie są* jednego koloru, możemy jednoznacznie wyznaczyć kolor jednego z wierzchołków. Taki trójkąt ma zawsze dwie krawędzie tego samego koloru — wspólny wierzchołek tych dwóch krawędzi ma jednoznacznie wyznaczony kolor.

Obserwacja 5. Jeśli mamy n niepokolorowanych wierzchołków, które tworzą jednokolorową klikę (każda krawędź między tymi wierzchołkami jest tego samego koloru), to możemy je dobrze pokolorować na $n+1$ sposobów.

Dowód: Załóżmy, że wierzchołki tworzą czarną klikę (dowód w przypadku szarej jest analogiczny). Możemy wtedy:

- pokolorować wszystkie wierzchołki na czarno lub
- wybrać jeden wierzchołek, pokolorować go na szaro, zaś resztę pokolorować na czarno.

Nie możemy pokolorować dwóch lub więcej wierzchołków na szaro, bo między każdą parą wierzchołków jest czarną krawędź, a ona nie może mieć dwóch końców szarych. Mamy zatem dokładnie $n + 1$ rozwiązań. ■

Rozwiązanie powolne – $O(n^4)$

Jeśli znajdziemy jakiś różnokolorowy trójkąt, to możemy odpowiedni jego wierzchołek pokolorować na odpowiedni kolor (dokładniej, możemy wyznaczyć jeden z wierzchołków i kolor, który ten wierzchołek będzie miał w *każdym* dobrym kolorowaniu). Następnie możemy sprawdzić, czy pokolorowanie tego wierzchołka nie wymusza pokolorowania jego sąsiadów. Jeśli jakiś wierzchołek pokolorujemy na czarno (odpowiednio na szaro), możemy wszystkich jego sąsiadów, z którymi jest on połączony szarą (odpowiednio czarną) krawędzią, pokolorować na szaro (odpowiednio czarno). Dla każdego pokolorowanego sąsiada możemy też sprawdzić rekurencyjnie, czy jego pokolorowanie wymusza kolejne pokolorowania. Wszystkie pokolorowania, które wykonamy w ten sposób, są zgodne z dowolnym dobrym kolorowaniem grafu. W szczególności, gdybyśmy w wyniku takiego pokolorowania otrzymali dwa wierzchołki tego samego koloru połączone krawędzią przeciwnego koloru, to wiemy, że nie istnieje żadne dobre kolorowanie tego grafu. Po sprawdzeniu wszystkich wymuszeń ograniczamy nasz graf G do podgrafu G' zawierającego tylko niepokolorowane wierzchołki i rozwiązujemy rekurencyjnie nasz problem z mniejszym n. Zauważmy, że zachodzi następujący fakt.

Obserwacja 6. Każde dobre kolorowanie grafu G' (skonstruowanego w opisany powyżej sposób) tworzy wraz z wyznaczonym wcześniej pokolorowaniem pozostałych wierzchołków dobre kolorowanie grafu G.

Dowód: Załóżmy przeciwnie. To znaczy, że istnieje pewna krawędź uv w G, dla której u i v obydwa mają inny kolor niż krawędź uv. Nie może być tak, że u i v oba są w G', bo G' jest z założenia dobrze pokolorowane. Nie może być też tak, że u i v oba są poza G', bo już na etapie wymuszeń stwierdzilibyśmy, że nie istnieją dobre kolorowania. Załóżmy zatem, że $u \in G'$, $v \notin G'$. Ale wtedy krawędź łącząca u z v jest innego koloru niż v — zatem kolor u zostałby ustalony na etapie rekurencyjnych sprawdzeń. ■

Załóżmy zatem, że po przejrzeniu wszystkich różnokolorowych trójkątów i wykonaniu wszystkich wymuszeń rekurencyjnych pozostało nam jeszcze k niepokolorowanych wierzchołków. Skoro nie ma żadnego różnokolorowego trójkąta o wszystkich wierzchołkach wśród tych k, to tworzą one jednokolorową klikę (czarną lub szarą). Taką klikę możemy pokolorować na $k + 1$ sposobów.

Na początek możemy zatem rozważyć następujące naiwne rozwiązanie:

```
1:  function koloruj(int A, char K)
2:  begin
3:    kolor[A] := K;
4:    for B := 1 to n do
5:      if kolor(A, B) ≠ kolor[A] then begin
6:        if kolor[B] = '?' then return false;
7:        if kolor[B] = '?' then
8:          if not koloruj(B, kolor(A, B)) then return false;
9:      end
10:   return true;
11: end
12:
13: program Konspiracja
14: begin
15:   for A := 1 to n do kolor[A] := '?';
16:   zmieniany := true;
17:   while zmieniany do begin
18:     zmieniany := false;
19:     foreach A, B, C ∈ [1, n] do
20:       if kolor[A] = kolor[B] = kolor[C] = '?' then
21:         if (kolor(A, B) ≠ kolor(A, C)) and
22:               (kolor(A, B) ≠ kolor(B, C)) then begin
23:           zmieniany := true;
24:           if not koloruj(C, kolor(A, C)) then return 0;
25:         end
26:   end
27:   wynik := 1;
28:   for A := 1 to n do
29:     if kolor[A] = '?' then wynik := wynik + 1;
30:   return wynik;
31: end
```

Za każdym obrotem pętli **while** liczba niepokolorowanych wierzchołków (tych o kolorze '?') maleje przynajmniej o 1, zatem tych obrotów jest $O(n)$. Wyszukiwanie trójkątów zajmuje czas $O(n^3)$. Zanalizujmy jeszcze rekurencyjne kolorowanie (czyli funkcję *koloruj*). Otóż funkcja *koloruj* jest wywoływana tylko dla wierzchołka bez koloru i go koloruje — zatem w trakcie całego programu jest co najwyżej n wywołań tej funkcji. Pojedyncze wywołanie tej funkcji, nie licząc zejścia rekurencyjnego, jest realizowane w czasie $O(n)$ (przejrzenie wszystkich sąsiadów), a zatem łączny czas potrzebny na wszystkie wywołania funkcji *koloruj* w całym algorytmie to $O(n^2)$. Ten fakt warto zapamiętać, z funkcji *koloruj* będziemy jeszcze korzystać w następnych rozwiązaniach, a teraz zobaczyliśmy, że same wywołania tej funkcji zajmują wystarczająco mało czasu, byśmy mogli je wykonać w rozwiązaniu wzorcowym.

Zatem łączny koszt czasowy tego algorytmu to $O(n^4)$, a pamięciowy — $O(n^2)$. Nieco gorszą jego implementację można znaleźć w plikach `kons1.cpp` i `kons2.pas`. Na zawodach za takie rozwiązanie można było uzyskać do 30 punktów.

Rozwiązanie powolne – $O(n^3)$

Można zauważyć, że w poprzednim rozwiązaniu często zdarza się, że wielokrotnie przeglądamy jeden trójkąt. A zamiast tego możemy przeglądać wszystkie trójkąty po kolei, nie wracając już do tych raz przejrzanych. Wszystkich trójkątów jest $O(n^3)$, a wymuszanie amortyzuje się do czasu $O(n^2)$.

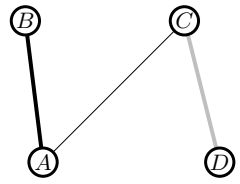
Rozwiązanie to zostało zaimplementowane w plikach `kons3.cpp` i `kons4.pas`. Działa w złożoności czasowej $O(n^3)$ i pamięciowej $O(n^2)$. Uzyskuje 50 punktów.

Rozwiązanie efektywne – $O(n^2)$

Postaramy się poprawić poprzednie rozwiązanie. Chcemy szybciej wyszukiwać różnokolorowe trójkąty, które wymuszą nam kolorowanie jeszcze nie pokolorowanych wierzchołków.

Obserwacja 7. Jeśli mamy dwie różnokolorowe krawędzie AB i CD, to z nich możemy od razu otrzymać różnokolorowy trójkąt.

Dowód: Jeśli te krawędzie mają wspólny wierzchołek, to już mamy ten trójkąt (bo krawędź między dwoma różnymi wierzchołkami będzie miała inny kolor niż krawędź AB lub CD).



Rys. 3: Ilustracja drugiej części dowodu Obserwacji 7.

Teraz załóżmy, że te krawędzie nie mają wspólnego wierzchołka. Prowadzimy krawędź AC. Jeśli AC ma taki sam kolor jak AB, to krawędzie AC i CD tworzą różnokolorowy trójkąt. Jeśli AC ma taki sam kolor jak CD, to krawędzie AC i AB tworzą różnokolorowy trójkąt. ■

Nasz ulepszony algorytm działa następująco. Tworzymy dwie listy krawędzi; na jednej liście będziemy trzymać czarne krawędzie, a na drugiej szare. Początkowo każdą krawędź wkładamy na odpowiednią listę. Krawędź będziemy zdejmować z listy dopiero wtedy, kiedy przynajmniej jeden z jej końców będzie już pokolorowany.

Załóżmy, że w pewnym momencie wykonywania algorytmu obie listy są niepuste. Weźmy dwie różnokolorowe krawędzie, będące pierwszymi elementami list. Wówczas, korzystając z Obserwacji 7, możemy w czasie stałym wygenerować różnokolorowy trójkąt. Każdy wierzchołek tego trójkąta jest końcem jednej z naszych krawędzi. Na mocy Obserwacji 4 możemy teraz dla jednego z wierzchołków trójkąta wywołać funkcję *koloruj*.

Za każdym razem, kiedy wywołujemy funkcję *koloruj* i przeglądamy sąsiadów wierzchołka, próbujemy usunąć z odpowiedniej listy krawędzie łączące ten wierzchołek

z jego sąsiadami. (Uwaga implementacyjna: łatwiej jest, zamiast usuwać krawędź z listy w momencie kolorowania jednego z jej końców, sprawdzać dopiero przy pobieraniu z listy, czy dana krawędź nie powinna była zostać usunięta). Jeśli w trakcie kolorowania otrzymamy sprzeczność, to wypisujemy zero i kończymy działanie programu.

Powiedzmy zatem, że któraś z list, po pewnej liczbie kroków, stanie się pusta (kroków jest co najwyżej n, bo co najwyżej tyle razy możemy wywołać funkcję *koloruj*). To oznacza, że niepokolorowane wierzchołki tworzą jednokolorową klikę, a zatem, na mocy Obserwacji 5, mamy $k+1$ rozwiązań, przy czym k to liczba niepokolorowanych wierzchołków.

Tworzenie i przeglądanie list zajmuje czas $O(n^2)$ (tyle jest krawędzi), wszystkie wywołania funkcji *koloruj* zajmują łącznie również czas $O(n^2)$.

Rozwiązanie to zostało zaimplementowane w `kon.cpp`, `kon1.pas` i `kon2.cpp`. Działa w złożoności czasowej $O(n^2)$ i pamięciowej $O(n^2)$. Uzyskuje 100 punktów.

Rozwiązanie alternatywne – 2-SAT z poprawianiem

W tym rozwiązaniu zauważamy, że jeśli mamy dobre kolorowanie, to inne dobre kolorowania nie mogą się od niego za bardzo różnić.

Załóżmy, że w pewnym dobrym kolorowaniu K wierzchołki u i v są czarne. Ale wtedy, ponieważ K jest dobre, mamy, że krawędź uv jest czarna — a zatem w dowolnym dobrym kolorowaniu co najwyżej jeden z wierzchołków u, v jest szary. Stąd, spośród wierzchołków, które są pokolorowane na czarno w K, w dowolnym innym dobrym pokolorowaniu co najwyżej jeden jest szary (i analogicznie spośród szarych w K co najwyżej jeden jest czarny). To oznacza, że dowolne inne dobre kolorowanie mogę dostać z K przez jedną z trzech następujących akcji:

- przemalowanie jednego czarnego wierzchołka na szaro
- przemalowanie jednego szarego wierzchołka na czarno
- zamiana jednego czarnego wierzchołka z jednym szarym wierzchołkiem (czyli przemalowanie czarnego wierzchołka na szaro, a szarego na czarno).

W tym rozwiązaniu znajdziemy jakieś dobre kolorowanie i obliczymy, ile innych pokolorowań da się uzyskać z tego kolorowania za pomocą wyżej wymienionych akcji.

Problem 2-SAT

Chcemy zacząć od znalezienia jakiegokolwiek rozwiązania. W tym celu sprowadzimy nasz problem do znanego problemu 2-SAT. W problemie 2-SAT mamy dane N zmiennych logicznych (przyjmujących wartości **true** i **false**). Mamy też dany ciąg M *klauzul*, czyli formuł logicznych postaci $a \vee b$, przy czym każda z liter a, b oznacza albo pojedynczą zmienną, albo jej negację. W problemie 2-SAT pytamy, czy istnieje takie przypisanie zmiennym wartości, że każda z M klauzul jest spełniona (tzn. prawdziwa), a jeśli tak, to prosimy o podanie takiego wartościowania. Problem 2-SAT można rozwiązać w czasie $O(N + M)$, taki algorytm można znaleźć np. w opisie rozwiązania

zadania *Śluzy* z Bałtyckiej Olimpiady Informatycznej 2008 [1], zapisany w niejawny sposób w opracowaniu zadania *Spokojna komisja* z VIII Olimpiady Informatycznej [8] albo na stronie `http://was.zaa.mimuw.edu.pl/?q=node/39`.

Jak ten problem ma się do naszego zadania? Otóż stowarzyszymy z każdym wierzchołkiem v zmienną x_v, która przyjmuje wartość **true**, jeśli x_v jest pokolorowany na czarno, a **false**, jeśli na szaro. Z każdą krawędzią czarną uv kojarzymy klauzulę $x_u \vee x_v$ (bo jeden z jej końców musi być czarny), a z każdą szarą krawędzią kojarzymy $\neg x_u \vee \neg x_v$ (bo któryś koniec musi być szary). I teraz, faktycznie, rozwiązanie problemu 2-SAT dla tego zestawu zmiennych i klauzul jest równoważne dobremu kolorowaniu.

Algorytm 2-SAT będzie tu działał w czasie $O(n^2)$, gdyż w naszym przypadku $N = n$, $M = n(n-1)/2$.

Rozwiązanie powolne – $O(n^4)$

Przy pomocy algorytmu 2-SAT generujemy jakiekolwiek rozwiązanie. Następnie wykonujemy każdą z $O(n^2)$ możliwych akcji i sprawdzamy, czy uzyskane rozwiązanie jest poprawne. Mamy $O(n)$ możliwych akcji przemalowania jednego wierzchołka na inny kolor i $O(n^2)$ par wierzchołków do zamiany kolorów. Zliczamy otrzymane rozwiązania (łącznie z pierwotnym, wygenerowanym przez algorytm 2-SAT) i wypisujemy odpowiedź.

Rozwiązanie to zostało zaimplementowane w plikach `kons5.cpp` i `kons6.pas`. Działa w złożoności czasowej $O(n^4)$ i pamięciowej $O(n^2)$. Na zawodach uzyskiwało 30 punktów.

Rozwiązanie powolne – $O(n^3)$

Widać, że najbardziej czasochłonne jest sprawdzanie akcji typu „zamiana wierzchołków". Zauważmy, że wierzchołek czarny można zamienić z szarym tylko wtedy, gdy jest to jego jedyny szary sąsiad, z którym jest połączony czarną krawędzią, ewentualnie gdy w ogóle nie ma żadnych takich szarych sąsiadów połączonych czarną krawędzią. Faktycznie, jeśli byłoby więcej takich sąsiadów, to po zamianie tej pary dwa szare wierzchołki byłyby połączone czarną krawędzią, co dałoby sprzeczność. Podobnie, jeśli mamy dokładnie jednego takiego sąsiada i próbowalibyśmy dokonać zamiany z jakimś innym szarym wierzchołkiem.

Dzięki temu spostrzeżeniu liczba par do sprawdzenia w porównaniu do poprzedniego rozwiązania zmalała nam z $O(n^2)$ do $O(n)$. Mamy dalej $O(n)$ możliwych akcji przemalowania jednego wierzchołka na inny kolor i tylko $O(n)$ par wierzchołków do zamiany kolorów. Poprawność rozwiązania weryfikujemy w czasie $O(n^2)$.

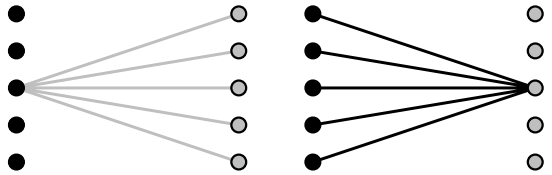
Rozwiązanie to zostało zaimplementowane w plikach `kons7.cpp` i `kons8.pas`. Działa w złożoności czasowej $O(n^3)$ i pamięciowej $O(n^2)$. Uzyskuje 50 punktów.

Rozwiązanie efektywne – $O(n^2)$

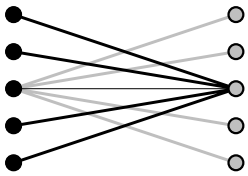
Wiemy, że może się zdarzyć, że jest $\Theta(n)$ rozwiązań (np. gdy graf wejściowy jest czarną kliką). Wobec tego będziemy musieli wykonać $\Omega(n)$ sprawdzeń poprawności

[1] Książeczka BOI 2008 jest dostępna pod adresem `http://b08.oi.edu.pl/downloads/booklet.pdf` (jedynie w wersji anglojęzycznej).

(czyli $\Omega(n)$ akcji). Zatem, by osiągnąć satysfakcjonującą nas efektywność algorytmu, musimy szybciej niż w czasie $O(n^2)$ sprawdzać poprawność rozwiązania. W tym celu zrezygnujemy z naiwnego sprawdzania poprawności.



Aby dało się przemalować wierzchołek czarny na szaro, musi on być połączony z każdym szarym wierzchołkiem szarą krawędzią. Jeśli byłby połączony z jakimś szarym wierzchołkiem czarną krawędzią, to po jego przemalowaniu ta krawędź prowadziłaby do sprzeczności. Analogiczne spostrzeżenie można poczynić przy przemalowywaniu szarego wierzchołka na czarno, patrz rysunek powyżej. W obydwu przypadkach otrzymany warunek na poprawność przemalowania jest zarazem konieczny i dostateczny.



Łącząc te dwa spostrzeżenia, można zauważyć, że czarny wierzchołek u możemy zamienić z szarym wierzchołkiem v wtedy i tylko wtedy, gdy u jest połączony z każdym szarym wierzchołkiem prócz v szarą krawędzią, a v jest połączony z każdym czarnym wierzchołkiem prócz u czarną krawędzią. Kolor krawędzi między zamienianymi wierzchołkami może być dowolny, patrz też rysunek powyżej.

Załóżmy, że nasze startowe rozwiązanie ma k czarnych wierzchołków i $n - k$ szarych. Wtedy, korzystając z powyższych spostrzeżeń, wnioskujemy, że:

- czarny wierzchołek u możemy przemalować na szaro, jeśli ma dokładnie $k - 1$ incydentnych czarnych krawędzi;
- analogicznie szary wierzchołek możemy przemalować na czarno, jeśli ma dokładnie k incydentnych czarnych krawędzi;
- wierzchołki u (czarny) i v (szary) możemy zamienić kolorami, jeśli uv jest czarna, zaś u i v mają po k czarnych sąsiadów, lub uv jest szara i u i v mają po $k - 1$ czarnych sąsiadów.

Jeśli na wstępie zliczymy szare krawędzie incydentne z każdym z wierzchołków (czas $O(n^2)$), to następnie poprawność każdej akcji możemy sprawdzić w czasie $O(1)$.

To rozwiązanie zostało zaimplementowane w plikach `kon5.cpp` i `kon6.pas`. Działa w złożoności czasowej $O(n^2)$ i pamięciowej $O(n^2)$. Uzyskuje 100 punktów.

Rozwiązanie alternatywne – konstrukcja rozwiązań

Tym razem podejdziemy do tego zadania jeszcze inaczej. Można, mianowicie, zauważyć, że wszystkich dobrych kolorowań będzie co najwyżej $n+1$ (wynika to z analizy dowolnego z poprzednich dwóch rozwiązań). Postaramy się skonstruować wszystkie rozwiązania od początku w sposób iteracyjny. Zaczniemy od $n = 1$ — wtedy są 2 rozwiązania. Teraz będziemy dodawać kolejne wierzchołki i poszerzać na bieżąco wszystkie rozwiązania.

Rozwiązanie powolne – $O(n^3)$

Dla każdego $k = 1, 2, \ldots, n$ utrzymujemy listę wszystkich poprawnych kolorowań dla pierwszych k wierzchołków. Tych kolorowań może być maksymalnie $k+1$. Teraz pokażemy, jak dodać jeden wierzchołek i uzyskać kolorowania dla $k+1$ wierzchołków.

Dodajemy nowy wierzchołek, nadając mu najpierw kolor czarny, a potem szary, i sprawdzamy, które z rozwiązań dla k są niesprzeczne po rozszerzeniu ich do $k+1$. Sprawdzenie wykonujemy w czasie $O(k)$, bo wystarczy sprawdzić, czy krawędzie wychodzące z nowo dodanego wierzchołka nie powodują sprzeczności.

Potencjalnie z każdego rozwiązania dla k wierzchołków można uzyskać dwa nowe rozwiązania, a zatem mamy $O(k)$ rozwiązań do sprawdzenia. W ten sposób wszystkie kolorowania dla $k+1$ wierzchołków konstruujemy w czasie $O(k^2)$. Zwróćmy uwagę, że w tym podejściu w istotny sposób korzystamy z tego, że *a priori* wiemy, iż rozważanych rozwiązań będzie w każdym kroku co najwyżej $O(k)$, co wcale z analizy tego algorytmu nie wynika.

Rozwiązanie to zostało zaimplementowane w plikach `kons9.cpp` i `kons10.pas`. Działa w złożoności czasowej $O(n^3)$ i pamięciowej $O(n^2)$. Na zawodach uzyskiwało 50 punktów.

Rozwiązanie efektywne – $O(n^2)$

Aby powyższe rozwiązanie usprawnić, należy porządnie zanalizować, jaką postać mają wszystkie poprawne kolorowania. Przyglądając się np. pierwszym dwóm rozwiązaniom, widzimy, że istnieje zbiór wierzchołków, które mają ten sam kolor w każdym dobrym kolorowaniu, oraz jakaś reszta. Wszystkie krawędzie w obrębie reszty mają ten sam kolor, powiedzmy czarny. Jeśli wierzchołków w reszcie jest k, to mamy $k+1$ rozwiązań — jedno, które wszystkie k wierzchołków koloruje na czarno, i k, które dokładnie jeden z k wierzchołków kolorują na szaro.

Podobnie jak w poprzednim rozwiązaniu, będziemy konstruowali iteracyjnie kolorowania dla kolejnych k. Jednak tym razem nie będziemy trzymać wszystkich kolorowań, lecz będziemy pamiętać, które wierzchołki mają już wymuszone kolory i jakie, a które tworzą jednokolorową klikę.

Dodając nowy wierzchołek v, rozpoczynamy od sprawdzenia, czy krawędzie z v do już pokolorowanych wierzchołków nie wymuszają kolorowania v (czyli czy czarne krawędzie nie idą do szarych wierzchołków lub czy szare krawędzie nie idą do czarnych). Jeśli okaże się, że nowy wierzchołek ma wymuszony kolor, to sprawdzamy, czy nie wymusza on kolorów wierzchołków z kliki (jeśli wymusza, to kolorujemy je, usuwamy z kliki i przerzucamy je do zbioru wierzchołków pokolorowanych).

Załóżmy dla uproszczenia, że w klice mamy same czarne krawędzie. Wtedy jeżeli jakikolwiek wierzchołek kliki wymusimy na szaro, to wszystkie pozostałe wierzchołki kliki automatycznie wymuszają się na czarno, natomiast wymuszenie jakiegokolwiek wierzchołka kliki na czarno nie wymusza niczego na pozostałych wierzchołkach kliki. W szczególności, gdyby v miał wymusić dwa wierzchołki w klice na szaro, to możemy od razu zwrócić zero. Dzięki tym spostrzeżeniom, przy wymuszaniu nie musimy przeglądać krawędzi należących do kliki.

Jeżeli nowo dodany wierzchołek v miał wymuszony kolor, to po wymuszeniach w klice możemy przejść do dodawania następnego wierzchołka. Tu uwaga — może się zdarzyć, że nasza „klika", którą pamiętamy, stanie się w wyniku wymuszeń pusta albo jednoelementowa, kiedy to nie ma jednoznacznie określonego koloru. Dla algorytmu jest obojętne, jaki kolor w takiej sytuacji uznamy, że ma klika.

Jeśli pokolorowane wierzchołki nie wymuszają żadnego koloru nowemu wierzchołkowi, to teraz patrzymy na krawędzie z niego do kliki. Trzeba rozważyć kilka przypadków:

- jeśli wszystkie krawędzie do kliki są tego samego koloru co klika, to po prostu dodajemy ten wierzchołek do kliki;
- jeśli jest dokładnie jedna krawędź innego koloru, to wszystkie wierzchołki (poza tym, do którego idzie ta jedna krawędź) kolorujemy na kolor, jakiego była klika, a nową kliką staje się ta jedna krawędź innego koloru. Tu korzystamy ze spostrzeżenia o kolorowaniu trójkątów różnokolorowych;
- jeśli są co najmniej dwie krawędzie innego koloru, to dostajemy różnokolorowy trójkąt, który wymusza kolor nowo dodanego wierzchołka, i przechodzimy do przypadku, w którym dodawany wierzchołek ma wymuszony kolor. Tu, ponownie, korzystamy ze spostrzeżenia o trójkątach różnokolorowych.

Dodanie nowego wierzchołka zajmuje czas $O(k)$. Rozwiązanie to zostało zaimplementowane w plikach `kon7.cpp`, `kon8.cpp` i `kon9.pas`. Działa w złożoności czasowej i pamięciowej $O(n^2)$. Uzyskuje 100 punktów.

Testy

Zostało przygotowanych 10 zestawów testów po 3 testy w zestawie (w niektórych 5). Testy a to testy losowe, w których wynikiem zawsze jest 0. Testy b to testy losowe, w których wynik jest mały i rozwiązania mają prawie tyle samo czarnych i szarych wierzchołków. Testy c to testy losowe, w których wynik jest duży.

Nazwa	n
kon1a.in	10
kon1b.in	8
kon1c.in	2
kon1d.in	8

Nazwa	n
kon1e.in	2
kon2a.in	27
kon2b.in	29
kon2c.in	30

Nazwa	n
kon3a.in	90
kon3b.in	78
kon3c.in	84
kon4a.in	123

Nazwa	n
kon4b.in	196
kon4c.in	139
kon5a.in	195
kon5b.in	254
kon5c.in	187
kon6a.in	1 300
kon6b.in	1 534
kon6c.in	1 832

Nazwa	n
kon7a.in	1 500
kon7b.in	1 953
kon7c.in	2 343
kon8a.in	2 500
kon8b.in	2 693
kon8c.in	2 783
kon9a.in	3 900

Nazwa	n
kon9b.in	3 984
kon9c.in	4 032
kon10a.in	5 000
kon10b.in	4 963
kon10c.in	4 984
kon10d.in	5 000
kon10e.in	5 000