



Nawigacja

 3 sek.  512 MB

Dany jest **spójny nieskierowany prosty graf postaci kaktusa**¹, który ma $N \leq 1000$ wierzchołków i M krawędzi. Wierzchołki mają przypisane kolory (oznaczone przez nieujemne liczby całkowite od 0 do 1499). Początkowo wszystkie wierzchołki mają kolor 0. **Deterministyczny robot bez pamięci**² chodzi po grafie przemieszczając się pomiędzy wierzchołkami. Jego zadaniem jest odwiedzić wszystkie wierzchołki co najmniej raz i następnie zakończyć swoje działanie.

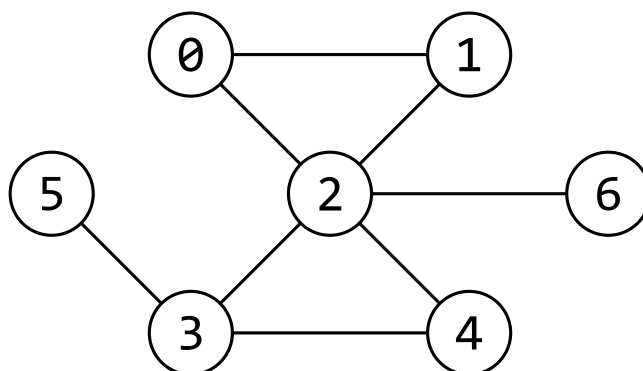
Robot zaczyna w pewnym wierzchołku, który może być dowolnym wierzchołkiem grafu. W każdym kroku, widzi kolor wierzchołka w którym się znajduje i kolor wszystkich sąsiadów tego wierzchołka; **kolejność sąsiadów danego wierzchołka jest niezmienna** (ponowne odwiedzenie wierzchołka nie zmieni kolejności podania kolorów wierzchołków sąsiadujących z nim, nawet jeśli ich kolory się zmieniają). Robot wykonuje jedną z podanych akcji:

1. Kończy swoje działanie.
2. Wybiera nowy (może się nie zmienić) kolor dla obecnego wierzchołka i do którego z sąsiadujących wierzchołków się ruszy. Sąsiadujące wierzchołki są ponumerowane liczbami od 0 do $D - 1$, gdzie D to liczba sąsiadów.

W przypadku drugiej opcji, obecnemu wierzchołkowi zostaje przypisany nowy kolor (może zostać ten sam), a robot przemieszcza się do wybranego sąsiedniego wierzchołka. Jest to powtarzane aż do zakończenia działania robota lub wyczerpania limitu kroków. Robot odnosi sukces jeśli odwiedzi wszystkie wierzchołki i zakończy swoje działania, nie przekraczając liczby kroków $L = 3000$ (w przeciwnym razie robot przegrywa).

Twoim zadaniem jest zaprojektowanie strategii robota, tak żeby odniósł sukces dla każdego grafu o postaci kaktusa. Dodatkowo, powinieneś zminimalizować liczbę różnych kolorów użytych przez robota. Kolor 0, jest zawsze liczony jako użyty.

¹Spójny nieskierowany prosty graf postaci kaktusa, jest spójnym nieskierowanym prostym grafem (każdy wierzchołek jest osiągalny z każdego innego; krawędzie są nieskierowane; nie zawiera pęteli i multikrawędzi), w którym każda krawędź należy do co najwyżej jednego cyklu prostego (cykl prosty, jest cyklem który przechodzi przez każdy wierzchołek co najwyżej raz). Poniższy obrazek jest przykładem takiego grafu.



²Robot jest deterministyczny i bez pamięci, jeśli każdy jego krok zależy tylko od jego obecnych danych (nie przechowuje żadnych danych z poprzednich kroków), i zawsze wybiera tę samą akcję dla tych samych danych.



Szczegóły implementacji

Strategia robota powinna być zaimplementowana w następującej funkcji:

```
std::pair<int, int> navigate(int currColor, std::vector<int> adjColors)
```

Funkcja ta dostaje jako parametry, kolor obecnego wierzchołka, i kolory jego sąsiadów. Powinna zwrócić parę, której pierwszy element to nowy kolor obecnego wierzchołka, a drugi to indeks sąsiada z podanego wektora kolorów, do którego powinien przejść robot. Jeśli robot powinien zakończyć swoje działania, funkcja powinna zwrócić parę $(-1, -1)$.

Ta funkcja będzie wywoływana wiele razy, dla określenia akcji robota. Ponieważ powinna być deterministyczna, jeśli `navigate` była wywołana raz, nie będzie nigdy więcej wywołana z tymi samymi parametrami; zostanie użyta poprzednio zwrócona przez nią wartość. Dodatkowo każdy test, zawiera $T \leq 5$ podtestów (różnych grafów i/lub różnych pozycji), które mogą być sprawdzane równocześnie (twoja funkcja może być wywoływana na zmianę dla różnych podtestów). Dodatkowo, wywołania `navigate` mogą nastąpić **w różnych uruchomieniach** twojego programu (ale mogą nastąpić również w tym samym uruchomieniu). Sumaryczna liczba uruchomień twojego programu nie przekroczy $P = 100$. Z tych powodów twój program nie powinien sobie przekazywać informacji pomiędzy wywołaniami.



Ograniczenia

- $3 \leq N \leq 1000$
- $0 \leq \text{Kolor} < 1500$
- $L = 3000$
- $T \leq 5$
- $P = 100$



Ocenianie

Ułamek S punktów, które otrzymasz za podzadanie zależy od C - maksymalnej liczby kolorów, które twój program użyje (wliczając kolor 0) ze wszystkich testów tego podzadania lub dowolnego innego podzadania wymaganego dla tego podzadania:

- Jeśli twoje rozwiązania będzie błędne dla dowolnego z podtestów, wtedy $S = 0$.
- Jeśli $C \leq 4$, wtedy $S = 1.0$.
- Jeśli $4 < C \leq 8$, wtedy $S = 1.0 - 0.6 \frac{C-4}{4}$.
- Jeśli $8 < C \leq 21$, wtedy $S = 0.4 \frac{8}{C}$.
- Jeśli $C > 21$, wtedy $S = 0.15$.



Podzadania

Podzadanie	Punkty	Wymagane podzadania	N	Dodatkowe ograniczenia
0	0	—	≤ 300	Test przykładowy.
1	6	—	≤ 300	Graf jest cyklem. ¹
2	7	—	≤ 300	Graf jest gwiazdą. ²
3	9	—	≤ 300	Graf jest ścieżką. ³
4	16	2 – 3	≤ 300	Graf jest drzewem. ⁴
5	27	—	≤ 300	Wszystkie wierzchołki mają co najwyżej 3 sąsiadów, a wierzchołek z którego startuje robot ma co najwyżej 1 sąsiada.
6	28	0 – 5	≤ 300	—
7	7	0 – 6	—	—

¹Cykl ma krawędzie: $(i, (i + 1) \bmod N)$ dla $0 \leq i < N$.

²Gwiazda ma krawędzie: $(0, i)$ dla $1 \leq i < N$.

³Ścieżka ma krawędzie: $(i, i + 1)$ dla $0 \leq i < N - 1$.

⁴Drzewo to graf bez cykli.



Przykład

Rozważmy graf z testu przykładowego, który jest na poniższym rysunku, gdzie $N = 7$, $M = 8$, a krawędzie to: $(0, 1)$, $(1, 2)$, $(2, 0)$, $(2, 3)$, $(3, 4)$, $(4, 2)$, $(3, 5)$ i $(2, 6)$. Dodatkowo, ponieważ kolejności wierzchołków na listach sąsiedztwa są istotne, podane są w poniższej tabelce:

Wierzchołek	Sąsiedzi
0	2, 1
1	2, 0
2	0, 3, 4, 6, 1
3	4, 5, 2
4	2, 3
5	3
6	2

Przyjmijmy, że robot startuje z wierzchołka 5. Wtedy to jest możliwa (nieudana) sekwencja interakcji:

#	Kolory	Wierzchołek	Wywołanie <code>navigate</code>	Zwrócona wartość
1	0, 0, 0, 0, 0, 0, 0	5	<code>navigate(0, {0})</code>	<code>{1, 0}</code>
2	0, 0, 0, 0, 0, 1, 0	3	<code>navigate(0, {0, 1, 0})</code>	<code>{4, 2}</code>
3	0, 0, 0, 4, 0, 1, 0	2	<code>navigate(0, {0, 4, 0, 0, 0})</code>	<code>{0, 3}</code>
4	0, 0, 0, 4, 0, 1, 0	6	<code>¹navigate(0, {0})</code>	<code>{1, 0}</code>
5	0, 0, 0, 4, 0, 1, 1	2	<code>navigate(0, {0, 4, 0, 1, 0})</code>	<code>{8, 0}</code>
6	0, 0, 8, 4, 0, 1, 1	0	<code>navigate(0, {8, 0})</code>	<code>{3, 0}</code>
7	3, 0, 8, 4, 0, 1, 1	2	<code>navigate(8, {3, 4, 0, 1, 0})</code>	<code>{2, 2}</code>
8	3, 0, 2, 4, 0, 1, 1	4	<code>navigate(0, {2, 4})</code>	<code>{1, 1}</code>
9	3, 0, 2, 4, 1, 1, 1	3	<code>navigate(4, {1, 1, 2})</code>	<code>{-1, -1}</code>

Robot użył 6 różnych kolorów: 0, 1, 2, 3, 4 i 8 (0 liczy się jako użyty kolor, nawet jeśli robot nigdy nie zwróciłby koloru 0, bo wszystkie wierzchołki zaczynają z kolorem 0). Robot wykonał 9 kroków przed zakończeniem działania. Mimo to robot przegrał, bo nie odwiedził wierzchołka 1.

¹Zwróć uwagę, że wywołanie `navigate` w kroku 4 nie wydarzyłoby się, ponieważ jest identyczne do wywołania w kroku 1, więc funkcja sprawdzająca użyłaby wartości zwróconej przez twoją funkcję z wcześniejszego wywołania. Ten krok nadal byłby jednak liczony do sumarycznej liczby kroków robota.



Przykładowa biblioteczka

Przykładowa biblioteczka nie uruchamia twojego programu wiele razy, więc wszystkie wywołania `navigate` nastąpią w tym samym uruchomieniu twojego programu.

Format wejścia jest następujący: najpierw jest wczytane T (liczba podtestów). Później dla każdego podtestu wczytane są:

- linia 1: dwie liczby całkowite – N oraz M ;
- linia $2 + i$ (dla każdego $0 \leq i < M$): dwie liczby całkowite – A_i oraz B_i , które są wierzchołkami, które łączy i -ta krawędź ($0 \leq A_i, B_i < N$).



Następnie przykładowa biblioteczka wypisze ile kolorów zużył robot przed zakończeniem działania lub wypisze komunikat błędu jeśli twój program będzie błędny.

Domyślnie, przykładowa biblioteczka wypisuje dokładne informacje o tym co robot widzi i robi, w każdej iteracji. Możesz to wyłączyć, zmieniając wartość `DEBUG` z `true` na `false`.