

Task Navigation

 3 sec.  512 MB

Se dă un **graf cactus simplu neorientat și conex**¹ cu $N \leq 1000$ noduri și M muchii. Nodurile sale au culori (notate cu numere întregi ne-negative de la 0 la 1499). Inițial toate nodurile au culoarea 0. Un **robot determinist fără memorie**² explorează graful mișcându-se dintr-un nod în altul. Acesta trebuie să viziteze fiecare nod cel puțin o dată, apoi să termine explorarea.

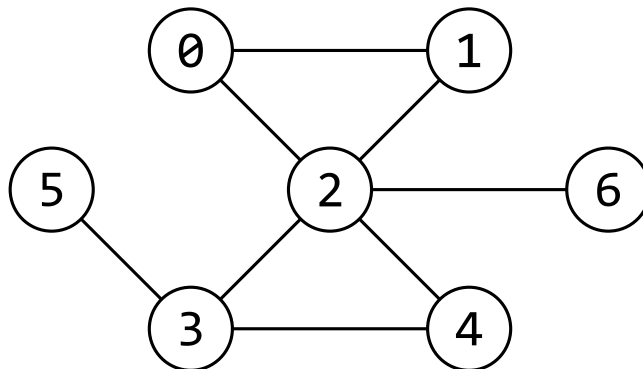
Robotul începe într-un nod, care ar putea fi oricare dintre nodurile din graf. La fiecare pas, el vede culoarea nodului curent și culorile tuturor nodurilor adiacente **într-o ordine fixată pentru nodul curent** (cu alte cuvinte, revizitarea nodului va oferi robotului aceeași secvență de noduri adiacente, chiar dacă culorile lor sunt diferite de cele dinainte). Robotul va face una dintre cele două acțiuni:

1. Decide să termine explorarea.
2. Alege o nouă (sau posibil aceeași) culoare pentru nodul curent și un nod adiacent în care se va muta. Nodul adiacent este identificat printr-un număr de la 0 la $D - 1$, unde D este numărul de noduri adiacente.

În al doilea caz, nodul curent este recolorat (sau posibil își păstrează culoarea) iar robotul se mută la nodul adiacent ales. Acest lucru se repetă până când robotul termină explorarea sau până când atinge limita de iterații. Robotul câștigă dacă vizitează toate nodurile și termină într-un număr de cel mult $L = 3000$ de iterații (altfel pierde).

Trebuie să proiectați o strategie pentru robot care rezolvă problema pentru orice graf cactus valid. În plus, trebuie să încercați să minimizați numărul de culori distincte pe care soluția le folosește. Considerăm culoarea 0 ca fiind mereu folosită.

¹Un *graf cactus simplu neorientat și conex* este un *graf simplu neorientat conex* (orice nod poate vizita orice alt nod; muchiile sunt bidirecționale; nu are muchii de la un nod la el însuși și nici muchii multiple) în care fiecare muchie aparține cel mult unui ciclu simplu (un *ciclu simplu* este un *ciclu* ce conține fiecare nod cel mult o dată). Un exemplu este imaginea de mai jos.



²Un robot este determinist și fără memorie, dacă acțiunile sale depind doar de intrările



curente (cu alte cuvinte, nu stochează nicio informație de la un pas la altul), și mereu face același lucru atunci când primește aceeași intrare.



Detalii de implementare

Strategia robotului trebuie implementată prin următoarea funcție:

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

Funcția primește ca parametri culoarea nodului curent și culorile tuturor nodurilor adiacente (în ordine). Trebuie să returneze o pereche al cărei prim element este culoarea nouă a nodului curent, iar al doilea element este indexul de adiacență al nodului la care robotul trebuie să se mute. Dacă în schimb robotul trebuie să termine, funcția va returna perechea $(-1, -1)$.

Această funcție va fi apelată în mod repetat pentru a alege acțiunile robotului. Deoarece este determinist, dacă `navigate` a fost deja apelată cu niște parametri, nu va mai fi apelată încă o dată cu aceeași parametri; în schimb valoarea returnată anterior va fi refolosită. În plus, fiecare test poate conține $T \leq 5$ subteste (grafuri distincte și/sau poziții de start distincte) iar acestea ar putea fi rulate concurent (cu alte cuvinte, programul vostru ar putea primi alternativ apeluri despre diferite subteste). În final, apelurile funcției `navigate` se pot întâmpla în **execuții separate** ale programului (dar ar putea să se întâmple și în cadrul aceleiași execuții). Numărul total de execuții ale programului vostru este $P = 100$. Astfel, programul vostru nu ar trebui să trimită informații între apeluri diferite.



Restricții

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



Punctare

Ponderea S a punctelor pe care le primiți pentru un subtask depinde de C - numărul maxim de culori distincte pe care soluția voastră le folosește (inclusiv culoarea 0) pe fiecare test din acel subtask sau din orice alt subtask necesar:

- Dacă soluția voastră eșuează pe un subtest, atunci $S = 0$.
- Dacă $C \leq 4$, atunci $S = 1.0$.



- Dacă $4 < C \leq 8$, atunci $S = 1.0 - 0.6 \frac{C-4}{4}$.
- Dacă $8 < C \leq 21$, atunci $S = 0.4 \frac{8}{C}$.
- Dacă $C > 21$, atunci $S = 0.15$.

Subtask-uri

| Subtask | Puncte | Subtask-uri necesare | N | Restricții adiționale |
|---------|--------|----------------------|------------|--|
| 0 | 0 | — | ≤ 300 | Exemplu. |
| 1 | 6 | — | ≤ 300 | Graful este un ciclu. ¹ |
| 2 | 7 | — | ≤ 300 | Graful este o stea. ² |
| 3 | 9 | — | ≤ 300 | Graful este un lanț. ³ |
| 4 | 16 | 2 – 3 | ≤ 300 | Graful este un arbore. ⁴ |
| 5 | 27 | — | ≤ 300 | Toate nodurile au cel mult 3 noduri adiacente și nodul din care începe robotul are 1 nod adiacent. |
| 6 | 28 | 0 – 5 | ≤ 300 | — |
| 7 | 7 | 0 – 6 | — | — |

¹Un graf ciclu are muchiile: $(i, (i + 1) \bmod N)$ pentru $0 \leq i < N$.

²Un graf stea are muchiile: $(0, i)$ pentru $1 \leq i < N$.

³Un graf lanț are muchiile: $(i, i + 1)$ pentru $0 \leq i < N - 1$.

⁴Un arbore este un graf fără cicluri.

Exemplu

Considerăm exemplul de graf din imaginea din enunț, care are $N = 7$, $M = 8$ și muchiile $(0, 1)$, $(1, 2)$, $(2, 0)$, $(2, 3)$, $(3, 4)$, $(4, 2)$, $(3, 5)$ și $(2, 6)$. În plus, deoarece ordinele elementelor din listele de adiacență ale nodurilor sunt relevante, le dăm în acest tabel:



| Nod | Noduri adiacente |
|-----|------------------|
| 0 | 2, 1 |
| 1 | 2, 0 |
| 2 | 0, 3, 4, 6, 1 |
| 3 | 4, 5, 2 |
| 4 | 2, 3 |
| 5 | 3 |
| 6 | 2 |

Să presupunem că robotul începe din nodul 5. Atunci, următoarea secvență de operații (fără succes) ar fi posibilă:

| # | Culori | Noduri | Apelul navigate | Valoarea returnată |
|---|---------------------|--------|--|-----------------------|
| 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 | ¹ <code>navigate(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> |

Aici robotul a folosit 6 culori distincte: 0, 1, 2, 3, 4 și 8 (observați că 0 ar fi numărat ca utilizat chiar dacă robotul nu ar returna niciodată 0, deoarece toate nodurile au culoarea 0 la început). Robotul s-a mișcat timp de 9 iterații înainte de terminare. Cu toate acestea, a eșuat deoarece a terminat explorarea fără a vizita nodul 1.

¹Observați că apelul funcției `navigate` la iterația 4 nu se întâmplă de fapt. Acest lucru se întâmplă deoarece este echivalent cu apelul de la iterația 1, deci grader-ul doar va refolosi valoarea returnată de la acel apel. Totuși, aceasta se numără ca o iterație făcută de robot.



Exemplu de grader

Exemplul de grader nu conține multiple execuții ale programului vostru, deci toate apelurile funcției `navigate` vor fi în aceeași execuție a programului.

Formatul datelor de intrare este următorul: Prima dată T (numărul de subteste) este citit. Apoi, pentru fiecare subtest:

- linia 1: două numere întregi - N și M ;
- linia $2 + i$ (pentru $0 \leq i < M$): două numere întregi - A_i și B_i , care sunt cele două noduri pe care muchia i le conectează ($0 \leq A_i, B_i < N$).

Exemplul de grader va afișa numărul de culori distincte pe care soluția voastră l-a folosit și numărul de iterații de care a avut nevoie înainte de a se termina. Alternativ, va afișa un mesaj de eroare în caz că soluția voastră eșuează.

În mod implicit, exemplul de grader afișează informații detaliate legate de ceea ce vede și face robotul la fiecare iterație. Puteți dezactiva aceasta prin schimbarea valorii `DEBUG` din `true` în `false`.