

Задача Навігація

 3 sec.  512 MB

Дано **зв'язний неорієнтований простий граф-кактус**¹ з $N \leq 1000$ вершинами та M ребрами. Його вершини мають кольори (позначені невід'ємними числами від 0 до 1499). Спочатку всі вершини мають колір 0. **Детерменістичний робот без пам'яті**² досліджує граф ходячи від вершини до вершини. Він має відвідати всі вершини хоча б один раз і потім зупинитися.

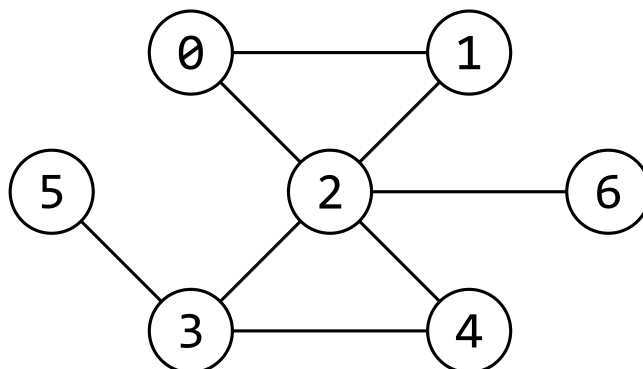
Робот починає в якійсь вершині, це може бути будь-яка з вершин графу. На кожному кроці він бачить колір вершини в якій знаходиться та кольори всіх сусідніх вершин **в порядку, який для теперешньої вершини є фіксованим** (тобто при повторному відвідуванні цієї вершини робот отримає таку саму послідовність сусідніх вершин, навіть якщо їх кольори змінилися). Робот робить одну з наступних дій:

1. Вирішує зупинитися.
2. Вибирає колір для поточної вершини(можливо такий самий як зараз) і в яку сусідню вершину перейти. Та сусідня вершина ідентифікується індексом від 0 до $D - 1$, де D це кількість сусідніх вершин.

В другому випадку поточна вершина перефарбовується (або можливо не перефарбовується) і робот переходить в обрану сусідню вершину. Це повторюється до того часу, поки робот не вирішить зупинитися або не привищить ліміт кроків. Робот виграє якщо він відвідує всі вершини а потім зупиняється в межах ліміту $L = 3000$ кроків (інакше він програє).

Ви повинні розробити стратегію для робота, яка дозволяє вирішити цю задачу на будь-якому кактусі, який задовольняє обмеженням. Також, ви повинні спробувати мінімізувати кількість різних кольорів, які використовуються в вашому рішенні. Колір 0 завжди враховується як використаний.

¹Зв'язний неорієнтований простий граф-кактус це зв'язний неорієнтований простий (Кожна вершина досяжна; ребра двонаправленні; в графі відсутні петлі та мульти-ребра) в якому кожне ребро належить не більш ніж одному простому циклу. (Простий цикл – цикл в якому кожна вершина зустрічається не більше одного разу). Знизу є приклад.



²Робот називається детерміністичним та без пам'яті, якщо його дії залежать тільки від його вхідних даних (тобто він не зберігає дані між кроками) і він завжди виконує одну і ту ж саму дію, коли отримає однакові вхідні дані.



Деталі реалізації

Стратегія робота повинна бути реалізована як функція:

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

Вона отримує колір поточної вершини та кольори всіх її сусідів (в порядку) як аргументи. Вона має повертати пару, перший елемент якої це новий колір тепершньої вершини, а другий – індекс в списку сусідів вершини, в яку робот має перейти. Якщо робот має зупинитися, то ця функція повинна повернути $(-1, -1)$

Ця функція буде викликатися неодноразово, щоб вибрати дії робота. Оскільки вона детермінована, якщо `navigate` вже була викликана з деякими аргументами, вона ніколи більше не буде викликана з тими самими параметрами; натомість буде повторно використано її попередньо повернене значення. Крім того, кожен тест може містити $T \leq 5$ підтестів (окремих графів та/або початкових позицій), і вони можуть виконуватися одночасно (тобто ваша програма може отримувати чергуючі виклики з різних підтестів). Нарешті, виклики `navigate` можуть відбуватися в **окремих виконаннях** вашої програми (але іноді вони також можуть відбуватися в одному виконанні). Загальна кількість виконань вашої програми становить $P = 100$. Через усе це ваша програма не повинна пробувати передавати інформацію між різними викликами.



Обмеження

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



Оцінювання

Частка балів S від максисальної кількості балів за підзадачу залежить від C - максисальної кількості різних кольорів які ваше рішення використовує (включаючи колір 0) серед тестів цієї підзадачі і всіх необхідних для неї підзадач:

- Якщо ваше рішення не працює на якомусь з тестів, то $S = 0$.
- Якщо $C \leq 4$, то $S = 1.0$.
- Якщо $4 < C \leq 8$, то $S = 1.0 - 0.6 \frac{C-4}{4}$.
- If $8 < C \leq 21$, то $S = 0.4 \frac{8}{C}$.
- If $C > 21$, то $S = 0.15$.



Підзадачі

Підзадача	Бали	Необхідні підзадачі	N	Додаткові обмеження
0	0	—	≤ 300	Приклад.
1	6	—	≤ 300	Граф - цикл. ¹
2	7	—	≤ 300	Граф - зірка. ²
3	9	—	≤ 300	Граф - шлях. ³
4	16	2 – 3	≤ 300	Граф - дерево. ⁴
5	27	—	≤ 300	Всі вершини мають не більше ніж 3 сусідні вершини і вершина, в якій робот починає шлях, має 1-го сусіда.
6	28	0 – 5	≤ 300	—
7	7	0 – 6	—	—

¹Цикл має ребра: $(i, (i + 1) \bmod N)$ для всіх $0 \leq i < N$.

²Зірка має ребра: $(0, i)$ для всіх $1 \leq i < N$.

³Шлях має ребра: $(i, i + 1)$ для всіх $0 \leq i < N - 1$.

⁴Дерево це граф без циклів.



Приклад

Розглянемо приклад графу з зображення в умові, який має $N = 7$, $M = 8$ і ребра $(0, 1)$, $(1, 2)$, $(2, 0)$, $(2, 3)$, $(3, 4)$, $(4, 2)$, $(3, 5)$ та $(2, 6)$. Оскільки порядок елементів в списку сусідів вершини важливий, ми вказали його в цій таблиці:

Вершина	Сусідні Вершини
0	2, 1
1	2, 0
2	0, 3, 4, 6, 1
3	4, 5, 2
4	2, 3
5	3
6	2

Припустимо що робот починає в вершині 5. Тоді могла б бути така (не успішна) послідовність кроків:

#	Колір	Вершина	Виклик navigate	Повернене значення
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>

Тут робот використав загалом 6 різних кольорів: 0, 1, 2, 3, 4 та 8 (зауважте, що 0 вважався б використаним, навіть якби робот ніколи не повертав колір 0, оскільки всі



вершини починаються з кольору 0). Робот виконав 9 кроків, перш ніж зупинитися. Однак він зазнав невдачі, оскільки зупинився, не відвідавши вершину 1.

¹Зверніть увагу, що виклик `navigate` на кроці 4 насправді не відбудеться. Це тому, що він еквівалентний виклику на кроці 1, тому градер просто повторно використовує повернене значення вашої функції з того виклику. Однак це все одно вважається як крок робота.



Приклад градера

Зразковий градер не робить кілька виконань вашої програми, тому всі виклики `navigate` будуть в одному виконанні вашої програми.

Формат вхідних даних наступний: Спочатку зчитується T (кількість підтестів). Потім для кожного підтесту:

- рядок 1: дві цілих числа – N та M ;
- рядок $2 + i$ (для $0 \leq i < M$): два числа – A_i та B_i , які є вершинами з'єднані ребром i ($0 \leq A_i, B_i < N$).

Потім зразковий градер виведе кількість різних кольорів, використаних у вашому розв'язку, та кількість кроків, яка була йому потрібна перед зупинкою. Або ж виведе повідомлення про помилку, якщо ваше розв'язання не вдалося.

За замовчуванням, зразковий градер друкує детальну інформацію про те, що бачить і робить робот на кожному кроці. Ви можете вимкнути це, змінивши значення `DEBUG` з `true` на `false`.