



Task Navigation

⌚ 3 sec. 💾 512 MB

There is a **connected undirected simple cactus graph**¹ with $N \leq 1000$ nodes and M edges. Its nodes have colors (denoted with non-negative integers from 0 to 1499). Initially all nodes have color 0. A **deterministic memoryless robot**² explores the graph by moving from node to node. It must visit all nodes at least once and then terminate.

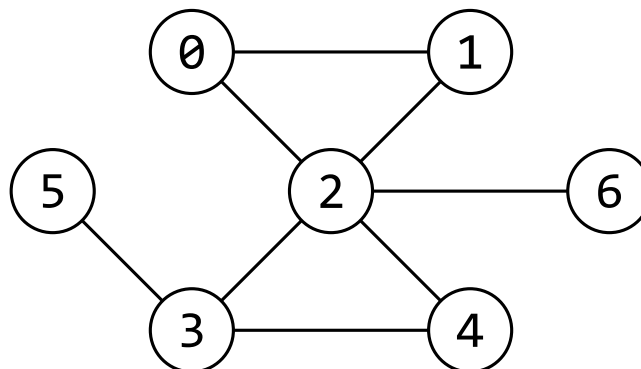
The robot starts at some node, which could be any of the nodes in the graph. At each step, it sees the color of its current node and the colors of all adjacent nodes **in some order fixed for the current node** (i.e. revisiting the node will give the robot the same sequence of adjacent nodes, even if their colors are different than before). The robot does one of the following two actions:

1. Decides to terminate.
2. Chooses a new (or possibly the same) color for the current node and which adjacent node to move to. The adjacent node is identified by an index from 0 to $D - 1$, where D is the number of adjacent nodes.

In the second case, the current node is recolored (or possibly stays the same color) and the robot moves to the chosen adjacent node. This repeats until the robot terminates or until it reaches the iteration limit. The robot wins if it visits all nodes and then terminates within the iteration limit of $L = 3000$ steps (otherwise it loses).

You should design a strategy for the robot that can solve the problem on any such cactus graph. Additionally, you should try to minimize the number of distinct colors your solution uses. Here, color 0 always counts as used.

¹A *connected undirected simple cactus graph* is a connected undirected simple graph (every node is reachable from every other node; edges are bi-directional; has no self-loops or multi-edges) in which every edge belongs to at most one simple cycle (a simple cycle is a cycle that contains each node at most once). The below image is an example.



²A robot is *deterministic and memoryless*, if its action depends only on its current inputs (i.e. it stores no data from step to step), and it always chooses the same action when given the same inputs.



Implementation details

The robot's strategy should be implemented as the following function:

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

It receives as parameters the color of the current node and the colors of all adjacent nodes (in order). It must return a pair whose first element is the new color for the current node and whose second element is the adjacency index of the node the robot should move to. If instead the robot should terminate, the function should return the pair $(-1, -1)$.

This function will be called repeatedly in order to choose the actions of the robot. Since it is deterministic, if `navigate` was already called with some parameters, it will never be called with those same parameters again; instead its previous return value will be reused. Additionally, each test may contain $T \leq 5$ subtests (distinct graphs and/or starting positions) and they may be run concurrently (i.e. your program may get alternating calls about different subtests). Finally, the calls to `navigate` may happen in **separate executions** of your program (but they may also sometimes happen in the same execution). The total number of executions of your program is $P = 100$. Due to all of this, your program should not try to pass information between different calls.



Constraints

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



Scoring

The fraction S of the points for a subtask that you get depends on C - the maximum number of distinct colors your solution uses (including color 0) on any test in that subtask or any other required subtask:

- If your solution fails on any subtest, then $S = 0$.
- If $C \leq 4$, then $S = 1.0$.
- If $4 < C \leq 8$, then $S = 1.0 - 0.6 \frac{C-4}{4}$.
- If $8 < C \leq 21$, then $S = 0.4 \frac{8}{C}$.
- If $C > 21$, then $S = 0.15$.



Subtasks

Subtask	Points	Required subtasks	N	Additional constraints
0	0	—	≤ 300	The example.
1	6	—	≤ 300	The graph is a cycle. ¹
2	7	—	≤ 300	The graph is a star. ²
3	9	—	≤ 300	The graph is a path. ³
4	16	2 – 3	≤ 300	The graph is a tree. ⁴
5	27	—	≤ 300	All nodes have at most 3 adjacent nodes and the node the robot starts at has 1 adjacent node.
6	28	0 – 5	≤ 300	—
7	7	0 – 6	—	—

¹A cycle graph has edges: $(i, (i + 1) \bmod N)$ for $0 \leq i < N$.

²A star graph has edges: $(0, i)$ for $1 \leq i < N$.

³A path graph has edges: $(i, i + 1)$ for $0 \leq i < N - 1$.

⁴A tree is a graph with no cycles.



Example

Consider the sample graph from the image in the statement, which has $N = 7$, $M = 8$ and edges $(0, 1)$, $(1, 2)$, $(2, 0)$, $(2, 3)$, $(3, 4)$, $(4, 2)$, $(3, 5)$ and $(2, 6)$. Additionally, since the orders of the elements in the nodes' adjacency lists are relevant, we give them in this table:

Node	Adjacent nodes
0	2, 1
1	2, 0
2	0, 3, 4, 6, 1
3	4, 5, 2
4	2, 3
5	3
6	2

Suppose the robot starts at node 5. Then the following is one possible (unsuccessful) sequence of interactions:



#	Colors	Node	Call to navigate	Return value
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>

Here the robot used a total of 6 distinct colors: 0, 1, 2, 3, 4 and 8 (note that 0 would have counted as used even if the robot never returned color 0, since all nodes start in color 0). The robot ran for 9 iterations before terminating. However, it failed since it terminated without having visited node 1.

¹Note the call to `navigate` at iteration 4 would not actually happen. This is because it is equivalent to the call at iteration 1, so the grader would simply reuse the return value of your function from that call. However, this still counts as an iteration of the robot.



Sample grader

The sample grader does not run multiple executions of your program, so all calls to `navigate` will be in the same execution of your program.

The input format is the following: First T (the number of subtests) is read. Then for each subtest:

- line 1: three integers - N , M and S (the starting node of the robot);
- line $2 + i$ (for $0 \leq i < M$): two integers - A_i and B_i , which are the two nodes that edge i connects ($0 \leq A_i, B_i < N$).

The sample grader will then print out the number of distinct colors your solution used and the number of iterations it needed before it terminated. Alternatively, it will print out an error message, if your solution failed.

By default, the sample grader prints detailed information on what the robot sees and does at each iteration. You can disable this, by changing the value of `DEBUG` from `true` to `false`.