Series-parallel graphs

Many problems are hard to solve for arbitrary undirected graphs, so it is necessary to design algorithms that will work correctly for special kinds of graphs. The simplest class of undirected graphs are undirected trees, and many problems that are difficult for general graphs are easy to solve for trees. A slightly bigger class of graphs is the series-parallel graphs.

An undirected tree is a connected undirected graph that can be reduced to a single node by repeatedly deleting a node of degree 1. Series-parallel graphs or more generally, multigraphs, are defined similarly with more general reduction operations.

A connected undirected multigraph is said to be series-parallel if it can be reduced to a single node using the following operations:

1. Delete a node of degree 1.
2. If there are two or more edges incident with the same pair of nodes, replace them by a single edge (parallel reduction).
3. If there is a node u of degree 2, adjacent to distinct nodes v, w, delete the node u and add an edge between v and w (series-reduction). There may already be an edge between v and w and this may give a multigraph.

In the first part of the problem, given an undirected multigraph, you have to determine whether it is a series-parallel multigraph or not.

A problem that is difficult to solve for general graphs is that of finding a maximum independent set. An independent set is a subset of nodes, such that no two nodes in the subset are adjacent. This problem can be solved efficiently for series-parallel graphs, and in the second part, you have to do this.

Algorithm: To decide whether a given graph is series-parallel or not, follow the reduction procedure as long as possible, till either every vertex has degree at least 3, or only a single node remains. To do this efficiently, maintain a queue of nodes of degree at most 2. This can be initialized when reading the graph. While the queue is not empty, and number of nodes is greater than one, pop a node from the queue and process it. To process a node of degree 1, simply delete it and the edge incident with it, and update the degree of its neighbour, queuing it if it becomes 2. If the node to be deleted has degree 2, again delete it and the two edges incident with it, and add the edge between its neighbours.

The main part is how to identify multiple edges and do parallel reduction. Instead of doing all possible parallel reductions, maintain a weaker condition: In the adjacency list of any node, if the list has at least 3 elements then the first 3 elements are distinct, hence the degree is at least 3, even after parallel reductions. We only need to know whether the degree becomes 2 after a node deletion and parallel reduction, in which case the node must be queued. A node deletion may cause several parallel reductions to happen at the neighbour, but the total number of reductions is bounded by the number of edges.

An implementation detail that is needed is that an edge vw occurs in the adjacency lists of v and w, and a pointer (or iterator) is needed from the occurrence of v in the list for w and vice-versa. This is because when v is deleted the edge vw must be removed from the list for w in $O(1)$ time. This deletion may violate the invariant for w, and may cause further parallel reductions at w, but the total time for this is $O(n+m)$.

For the maximum independent set part, for any reduced graph, we maintain 2 numbers for each node, and 4 numbers for each edge. A node v in the reduced graph could have had an arbitrary graph attached to it, which has been reduced to v. The two numbers indicate the maximum number of nodes in the reduced graph that can be added to an independent set if v is included and if not. Similarly, an edge in the reduced graph represents some subgraph of the original graph that has been reduced to the edge. For an edge uv, the 4 numbers are the maximum number of additional nodes in an independent set in the subgraph containing both u, v, only u, only v and neither. Initially all numbers are 0. When doing any reduction, the numbers can be updated by considering whether the deleted vertex is included in the maximum independent set or not. Finally, when a single vertex is left, the max of the two numbers gives the answer. This takes $O(1)$ time per reduction and hence $O(n+m)$ time in total. Note that if uv is an actual edge in the graph, there cannot be an independent set including both.

Input/Output: The first line will give the number of vertices n and the number of edges m, with $2 \leq n \leq 10^5$ and $1 \leq m \leq 10^6$. The next m lines will give the edges. There may be multiple edges in the input graph. The first line should output "yes" if the graph is series-parallel otherwise "no". The second line should give the maximum size of an independent set in the graph.

: Submission: Submit a single file named RollNo_12.cpp.