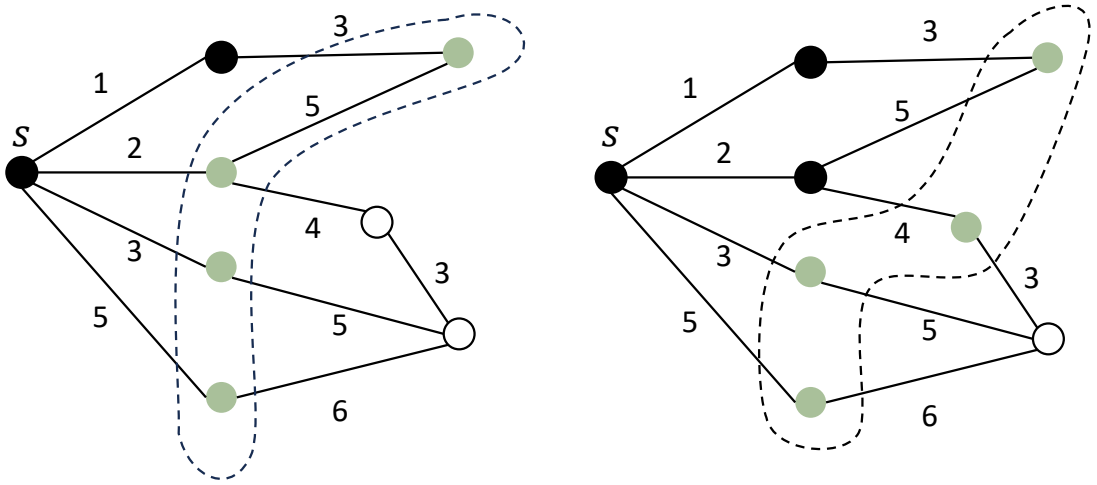
Exploration Boundary

Input file:	`standard input`
Output file:	`standard output`
Time limit:	4 seconds
Memory limit:	1024 megabytes

Recently, a paper "Universal Optimality of Dijkstra via Beyond-Worst-Case Heaps" on FOCS raised Fuyu's attention, which talks about the good old shortest-path algorithm achieving optimality on every type of graph by the heap with the working set property. To describe the optimality, there is a concept called *exploration boundary*, that is: pause the Dijkstra's algorithm at some time, the exploration boundary is the frontier of exploration on the graph, or, namely, the vertices that have been updated but with distances not yet determined.



An example of exploration boundaries, where the right boundary is based on the left boundary after determining the vertex with distance 2 from s and updating its neighbors.

To define it more formally, consider the following description of Dijkstra's algorithm. A non-empty set of vertices is called an *exploration boundary* if it can be obtained by extracting the vertices in Q when the algorithm is running to line 6.

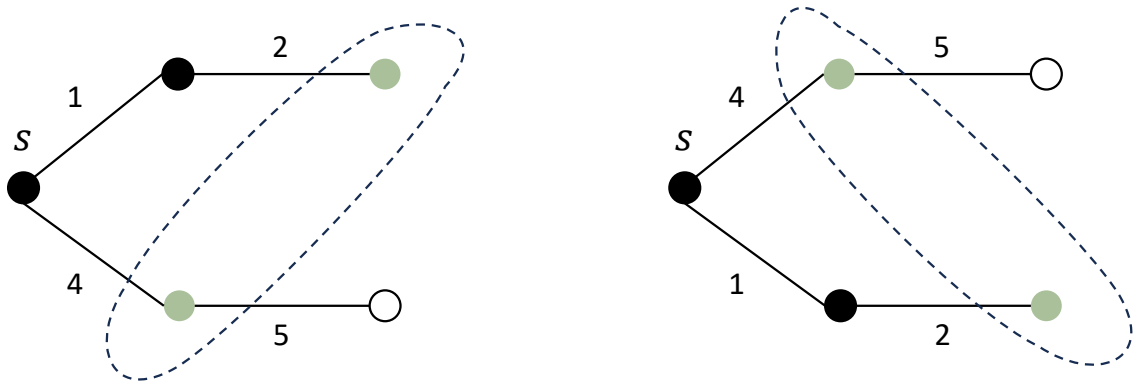
Algorithm 1: Dijkstra's Algorithm

Input: A graph $G = (V, E)$, weights w, a source vertex $s \in V$
Output: Distances D from s

```
1  S ← {} // A set of explored vertices, initially empty.
2  Q ← {} // Priority queue storing (distance, vertex) pairs, sorted by distance.  Initially empty.
3  P ← [∅, ..., ∅] // For each vertex v, a pointer into Q, or ∅, if v has not been inserted yet.
4  D ← [+∞, ..., +∞] // For each vertex, the current best distance from s.
5  P[s] ← Q.Insert((0, s)), D[s] ← 0
6  while |Q| ≠ ∅ do
7      (d_u, u) ← Q.DeleteMin()
8      S ← S ∪ {u}
9      forall (u, v) ∈ E where v ∉ S do
10         if P[v] = ∅ then
11             P[v] ← Q.Insert((+∞, v)) // Insert a dummy value first.
12         D[v] ← min(D[v], d_u + w(u, v)) // Use d_u to update the current best distance to v.
13         Q.DecreaseKey(P[v], (D[v], v)) // For simplicity, call even if D[v] did not change.
14 return D
```

For an undirected graph G, there are many different sets of exploration boundaries if we assign different positive weights w. The paper splits the graph into several boundaries to prove the best complexity of

the distance ordering problem on a certain fixed graph is the same as the time complexity of Dijkstra's algorithm using a working set heap. However, Fuyu noticed that some boundaries cannot be obtained at the same time with the same weight, even if they are disjoint. For example, the following two exploration boundaries are possible for different weights, but they cannot both appear in a single run of Dijkstra's algorithm.



An example of two exploration boundaries that can be obtained by different weights, but cannot both appear in a single run of Dijkstra's algorithm.

Given an undirected graph G and some desired exploration boundaries, let the source vertex $s = 1$, your task is to construct a weight for each edge if it's possible, such that:

- Each edge has a positive integer weight no greater than 10^9.
- No two vertices have the same distance from 1, so that the exploration boundaries can be uniquely determined during the execution of Dijkstra's algorithm.
- All exploration boundaries appeared during the run of Dijkstra's algorithm from 1.

Input

The input consists of multiple test cases. The first line contains an integer T ($1 \leq T \leq 10^5$), the number of test cases. For each test case:

- The first line contains two integers n and m ($2 \leq n \leq 2 \times 10^5, 1 \leq m \leq 2 \times 10^5$), the number of vertices and edges, respectively.
- The next m lines each contain two integers u and v ($1 \leq u, v \leq n$), describing an undirected edge between vertices u and v.
- The following line contains an integer k ($1 \leq k \leq 2 \times 10^5$), the number of desired exploration boundaries.
- Each of the next k lines describes a boundary: Each line starts with an integer b_i ($1 \leq b_i \leq n$), the number of vertices in the i-th boundary, followed by b_i integers denoting the vertices in the boundary. It is guaranteed that b_i vertices are pairwise distinct.

It's guaranteed that the input graph is connected with no self-loops and no duplicate edges, and two boundaries in the same test will not coincide. The sum of n, the sum of m, and the sum of b_i across all test cases do not exceed 10^6.

Output

For each test case:

- Output `Yes` in a single line if it is possible to construct such a weight assignment. In this case, on the next line, output m integers representing the weights for the edges in the order they were provided in the input. Each weight must be a positive integer within $[1, 10^9]$.
- Otherwise, output `No` in a single line.

Example

standard input	standard output
2 8 10 1 2 1 3 1 4 1 5 2 6 3 6 3 7 4 8 5 8 7 8 2 4 3 4 5 6 4 4 5 6 7 5 4 1 2 1 3 2 4 2 5 2 2 3 4 2 2 5	Yes 1 2 3 5 3 5 4 5 6 3 No