IIT Madras InstaInfluencer Fest '25

Hackerrank_link:

Background Story

It's March 2025, and after Shaastra, an **InstaInfluencer Fest** is happening at IIT Madras for the first time! This fest is a massive gathering of GenZ influencers, content creators, and fans. Events range from viral dance-offs, meme competitions, podcast sessions, to live-streaming challenges. Each event depends on certain other events being completed first (e.g., you can't have the "Final Dance-Off" without completing the preliminary rounds).

However, drama unfolds when some influencers demand that certain events happen before others, creating complex dependencies. If there is a directed edge between event $u \to v$, then event u must be completed before v. The organizers need your help to:

1. Identify if any drama-induced cyclic dependencies exist.
2. Find groups of tightly interdependent events (**Strongly Connected Components**) and the cardinality of the group with the maximal number of events.
3. Provide a valid order of events if possible.
4. Calculate the maximum "**hype score**" (don't worry it's described later) achievable from attending events in a valid path.

Problem Statement

You're given a **directed graph** representing events at InstaInfluencer Fest:

- **Nodes** represent events.
- **Directed edges** represent dependencies (event u must occur before event v).
  - Each event has an associated "**hype score**," indicating its popularity among IIT Madras students.

You must process queries of four types:

Query Type	Description
1	Check if the event schedule has any cyclic dependencies. Output "YES" if cycles exist; otherwise "NO".
2	Output the number of strongly connected components (**SCCs**) in the event dependency graph and the cardinality of the group with the maximal number of events.
3	Provide a valid topological order for the events if possible. Ensure that independent vertices are sorted **lexicographically** within their valid topological order. Incase of cycle output “NO”.
4	Compute the maximum total hype score achievable by attending events. Identify all SCCs in the graph. Condense each SCC in a single vertex, summing the hypescores of all vertices within the SCC representing the hypescore of this vertex. Now find the path with maximum total hypescore achievable in the condensed graph.

*A **Path** is a sequence of vertices connected by edges connecting two consecutive vertices in the sequence, and vertices are distinct(not repeated).

Input Format

1. The first line contains two integers N and M, representing the number of events and dependencies respectively.
2. The second line contains N integers $h_1, h_2, \ldots, h_N$, where h_i is the hype score of event i.
3. The next M lines each contain two integers u and v, indicating that event u must precede event v.
4. The next line contains an integer Q, representing the number of queries.
5. The next Q lines contain one integer per line indicating query type (1, 2, 3 or 4).

Output Format

For each query:

- For query type `1`: Output `"YES"` or `"NO"`.
- For query type `2`: Output two integers separated by space:
    - The number of SCCs in the graph.
    - The cardinality of the group with the maximal number of events.
- For query type `3`: Output a valid topological order separated by spaces; otherwise output `"NO"` if no valid order exists due to cycles.
- For query type `4`: Output an integer representing maximum total hype score achievable.

Constraints

$1 \leq N, M \leq 10^5$

$1 \leq Q \leq 10^5$

$1 \leq h_i \leq 10^4$

NOTE : The vertices are numbered from 1 to N, where N is the number of vertices.

Requirements

Design Requirements:

1. Create a class `GraphAlgorithm` which has a pure virtual function `void Query()` that is overloaded by all child classes:
    - `isCycle`: Detects cycles in the graph.
    - `indepComponent`: Computes SCCs and their cardinalities.
    - `validOrder`: Computes a valid topological order if possible.
    - `maxHype`: Computes maximum hype points on DAGs.
2. Implement a **Comparator Functor** .(Not mandatory)
3. Include explanatory comments or use self-explanatory variable and function names.

Sample Test Cases

Test Case 1:

Input:

4 4

10 20 30 40

1 2

2 3

3 1

3 4

4

1

2

3

4

Output:

YES

2 3

NO

100

Explanation:

- Query Type 1: A cycle exists (1 → 2 → 3 → 1), so output "YES".
- Query Type 2: Two SCCs are found: {1,2,3} with cardinality 3, and {4} with cardinality 1. Output is 2 3.
- Query Type 3: Topological sorting is impossible due to cycles, so output "NO".
- Query Type 4: Maximum hype score comes from attending {(1->2->3)->4}, which has a hype score of 60+40=100. (1->2->3) is a SCC.

Test Case 2:

Input:

8 8

10 20 10 10 30 40 10 20

1 2
1 5

2 3

3 4

5 6

5 7

7 3

7 8

4

4

3

1

2

Output:

80
1 2 5 6 7 3 4 8
NO
8 1

Explanation:

• Query Type 4: Maximum hype score is for the path {1-> 5-> 6} i.e. 80.

• Query Type 3: A valid topological order exists: 1-> 2-> 5-> 6-> 7-> 3-> 4-> 8. Since there is no cycle. Output is 1 2 5 6 7 3 4 8. After 1->2, there are two possible ways 3,5, but 3 cant be chosen as before 3,7 should be processed, thus will go with 5. similarly we'll process the graph.

• Query Type 1: No cycles exist; output "NO".

• Query Type 2: All the vertices are individual SCC this number of SCC is 8 and max cardinality is 1.