Cycle Finding

You are given a directed graph,
and your task is to find out if it contains a negative cycle, and also give an example of such a cycle

Input

The first input line has two integers n and m: the number of nodes and edges.
The nodes are numbered 1,2,...,n.

After this, the input has m lines describing the edges.
Each line has three integers a, b, and c: there is an edge from node a to node b whose length is c.

Output

If the graph contains a negative cycle, print first "YES", and then the nodes in the cycle in their correct order.
If there are several negative cycles, you can print any of them.
If there are no negative cycles, print "NO".

Constraints

1 <= n <= 2500
1 <= m <= 5000
1 <= a,b <= n
10^9 <= c <= 10^9

Example

Input

```
4 4
1 2 1
2 4 1
3 1 1
4 1 -3
```

Output

```
YES
1 2 4 1
```

Problem Breakdown

Lets understand the problem First

We are given a directed weighted graph with n nodes and m edges, and we're asked to:

Detect if there exists a negative weight cycle.

If yes, print the actual cycle.

Why Bellman-Ford Works Here

Bellman-Ford is a powerful algorithm that:

Can handle negative weights.

Can detect negative weight cycles.

Works in O(n * m) time.

💡 The key observation:

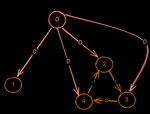
If after n iterations, we can still relax an edge, that means there's a negative cycle reachable from the source.

But... we don't have a source. Solution?

here we start from 1 as visual then we can never be able to detect the cycle.

We will use a Dummy Node

Let's see



We will add these new edges to our graph

Now from this 0th(dummy node) we will apply Bellman Ford and get our answers and also detect the cycle

So what we are using a dummy node we will initialize the distance array as below
0, 0, 1, 2, 3, 4, ∞

Now let's see without using Dummy Node

Bellman Ford algorithm will detect a negative cycle even if the graph is disconnected

Ah, negative cycle detection doesn't depend on reachability

We initialize all distances as 0 (instead of LLONG_MAX), we essentially treat every node as a potential starting point.

This works because:

If there's a negative cycle, some distances will decrease below 0 during the iterations.

The algorithm will detect this when it checks for relaxations in the n-th iteration.

1: 0, 2: 0, 3: 0, 4: 0

Cycle Detection

Even after detection of a cycle we are still stuck like the vertex which will be relaxed at nth iteration will be

Type:- 1

Its part of the negative cycle
like here, 2,3,4

Type:- 2

Its not part of the negative cycle
like here 1 and 5

We can easily handle Type 1 but Type 2 is tricky we are doing this in order to print the cycle so if anyhow get the vertex which is part of the cycle we just print it by tracing back.

So in Type 2 we will first want to reach the vertex which is part of negative cycle so for that we will maintain a relaxant array which will tell who relaxed whom necessary for us to trace back.

How to reach the cycle, in worst case we have to trace back n times like in this case

If this is detected

So just do
x = relaxant[x];

And for relaxing

inside the Bellman Ford condition

```
path[v] = path[u] + wt;
relaxant[v] = u;
x = v;
```

Algorithm

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;
typedef tuple<int, int, int> Edge;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n, m;
    cin >> n >> m;

    vector<Edge> edges;

    for (int i = 0; i < m; ++i) {
        int u, v, wt;
        cin >> u >> v >> wt;
        edges.push_back({u, v, wt});
    }

    vector<ll> dist(n + 1, 0);
    vector<int> relaxant(n + 1, -1);

    int x = -1;
    for (int i = 0; i < n; ++i) {
        x = -1;
        for (auto [u, v, w] : edges) {
            if (dist[u] + w < dist[v]) {
                dist[v] = dist[u] + w;
                relaxant[v] = u;
                x = v;
            }
        }
    }

    if (x == -1) {
        cout << "NO\n";
        return 0;
    }

    // Move x back n times to ensure we are in the cycle
    for (int i = 0; i < n; ++i) {
        x = relaxant[x];
    }

    vector<int> cycle;
    for (int curr = x;; curr = relaxant[curr]) {
        cycle.push_back(curr);
        if (curr == x && cycle.size() > 1) {
            break;
        }
    }
    reverse(cycle.begin(), cycle.end());

    cout << "YES\n";
    for (int node : cycle) {
        cout << node << " ";
    }
    cout << "\n";

    return 0;
}
```

Reading Edges as tuples

Initializing distance array with 0 as all the nodes perform as source

Relaxant array is initialized by -1

Bellman Ford to detect negative cycle

Tracing back to go to cycle if in Type 2 situation

Printing cycle

Time Complexity

O(n * m)

Space Complexity

O(n + m)

Common Questions

1. Why do we need the dummy node 0?
2. What happens if we don't reverse the cycle?
3. How does relaxant[] help us trace the cycle?
4. Can we use DFS/BFS for this? Why not?
5. What if the graph had no cycles — how do we know?