Programming Part

Problem 3 - Dusty Scholar's Abyss (100 pts)

Digging up the past is not curiosity —it's necromancy.

The Internet was meant to forget, not to haunt.

Problem Description

Story

DerTian, the archaeologist, has uncovered the ruins of an ancient civilization buried deep beneath the desert sands. These ruins comprise multiple underground cities, once interconnected by a vast network of tunnels that served as trade routes and pathways.

Over the centuries, many of these tunnels have collapsed, and some cities have become isolated. To guide zir excavation and preservation efforts, DerTian must map the network and uncover its structural weaknesses. These discoveries will help DerTian decide which tunnels to reinforce and where to focus excavation efforts.

The network consists of N cities and M tunnels. Each city is represented by an integer from 0 to $N-1$, and each tunnel is bidirectional, connecting two different cities. No tunnel connects to the same city, and between any pair of cities there can be at most one tunnel.

DerTian wants to identify:

- **Critical Cities**: The destruction of any of these cities would fragment the network into separate, unconnected regions. Destruction of a city implies the removal of the city itself and all tunnels connecting to it from the network.
- **Critical Tunnels**: The collapse of any of these tunnels would disconnect part of the network from the rest.
- **Independent Exploration Zones**: An independent exploration zone contain a set of cities and remain connected in the event of destruction of any *single* city in the zone. An independent exploration zone should contain as many cities as possible. For example, assume there are three cities `NTU`, `NTNU`, `NTUST` in the network, and they are mutually connected. $\{\texttt{NTU}, \texttt{NTNU}\}$ is NOT such a zone because $\{\texttt{NTU}, \texttt{NTNU}, \texttt{NTUST}\}$ satisfies the conditions and can completely contain $\{\texttt{NTU}, \texttt{NTNU}\}$.

Reference Reading

The following provides information about *Articulation Points*, *Bridges*, and *Biconnected Components* that may be helpful to solve this problem. You may also find these information in the older version of the **Introduction to Algorithms** textbook.

Definitions

Let $G = (V, E)$ be a connected, undirected graph.

- An articulation point is a vertex whose removal increases the number of connected components. The removal of the vertex also implies the removal of any edge involving the vertex.
- A bridge is an edge whose removal increases the number of connected components.
- A biconnected component is a maximal subgraph that remains connected even after the removal of any single vertex in the subgraph (and its incident edges).

Implementation

Background and preprocessing. We can determine articulation points, bridges, and biconnected components using DFS, along with a few additional properties. Suppose that $G_\pi = G(V, E_\pi)$ is the depth-first search (DFS) tree of G, i.e., the spanning tree generated according to the visit order and parent-child relationships in DFS. The definition of the DFS tree is identical to what we introduced in the lecture.

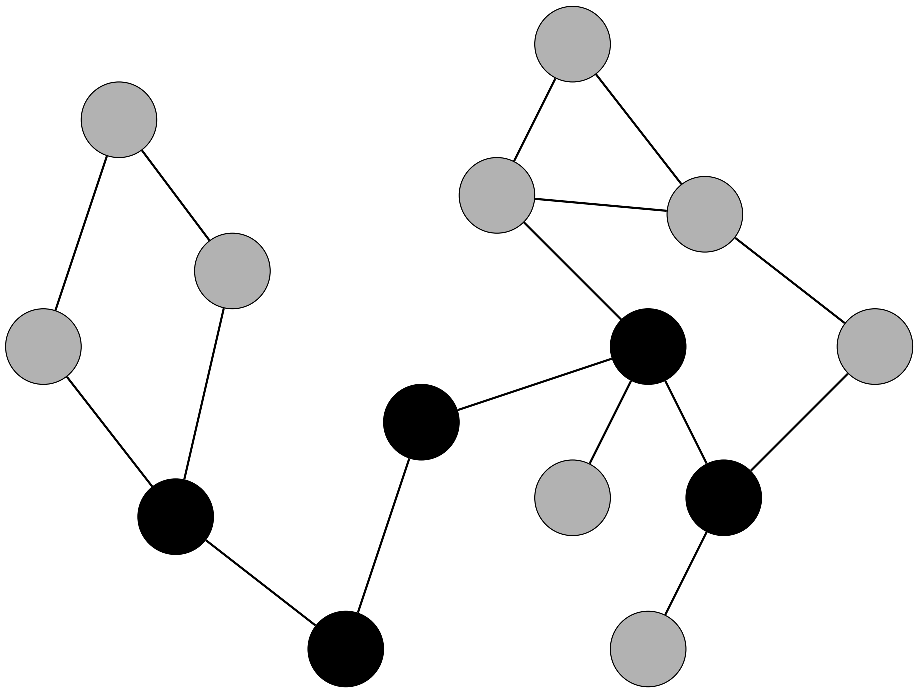
For each vertex u, we maintain one more value in additional to the ones covered in the lecture. Let $u.low = \min_{\mathcal{V}} (v.d)$, where $\mathcal{V}$ includes all vertices reachable from u by traversing zero or more *tree edges* and at most one *back edge*. Also, recall from the lecture that $v.d$ is the *discovery time*, the time vertex v is first visited.

Articulation point. To determine if a vertex is an articulation point, two cases need to be considered. First, we consider the case where the vertex is the root of G_π. It is an articulation point of G if and only if it has at least two children in G_π since its removal disconnects its children. Next, let's consider the case where the vertex is not the root. If *every* subtree of a non-root vertex u has at least one back edge connecting the subtree and any ancestor of u in G_π, the graph would remain connected if u and its incident edges are removed. Thus, for any non-root vertex u of G_π, it is an articulation point if and only if there exists a child v of u such that $v.low \geq u.d$.

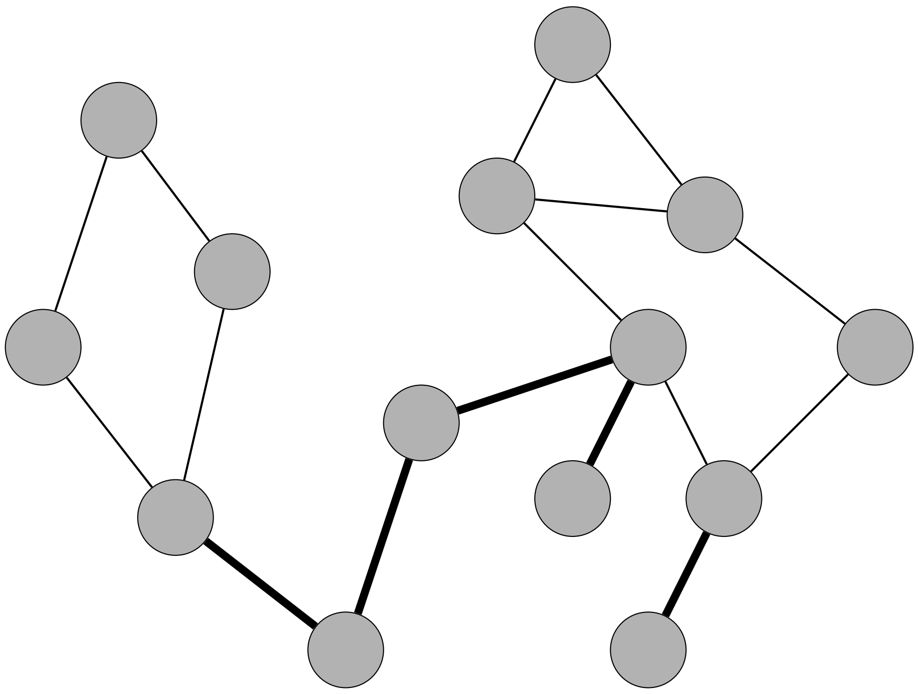
Bridge. Given a tree edge (u, v) in G_π, where u is the parent of v. If every subtree of v have at least one back edge connecting the subtree and u or any ancestor of u in G_π, the graph remains connected even if the edge (u, v) is removed. Hence, an edge (u, v) is a bridge if and only if $v.low > u.d$.

Biconnected component. During the DFS process, assume we have completed visiting all descendants of a vertex u. If any child v of u and v's descendants cannot reach a node with lower discovery time via any number of tree edge and at most one back edge, i.e., $v.low \geq u.d$, then we have one biconnected component including u, v and all of v's descendants, provided that no v's descendants satisfied the above condition. This can be implemented via maintaining a stack of edges during DFS. When DFS traverses an edge, push it into the stack. To extract the found biconnected component, when the above biconnected component condition is satisfied, pop edges from the stack until (u, v) is popped. All vertices involved in these edges belong to a biconnected component. Note that the vertices may belong to more than one biconnected component.

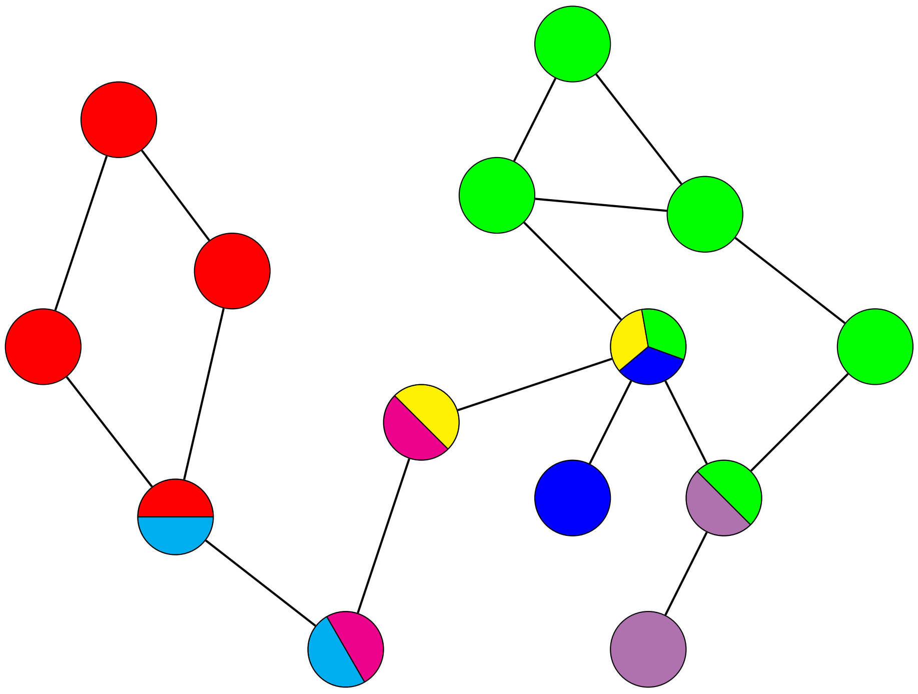
Below you can find visualization of some examples of articulation points, bridges, and biconnected components.



Articulation Points: Darker vertices



Bridges: Thicker edges



BCC: Each color shows a different BCC

Input

The first line contains three integers N, M, and `mode`. N is the number of cities, and the cities are numbered from 0 to $N-1$. M is the number of tunnels. `mode` has a value of 1, 2 or 3. Each of the next M lines contains two integers u_i and v_i, indicating the two cities connected by the i-th bidirectional tunnel. The consecutive integers in the same line are separated by a single white space.

Output

If `mode` is 1, print the number of critical cities in the first line. In the second line, print the city IDs in ascending order.

If `mode` is 2, print the number of critical tunnels in the first line.

If `mode` is 3, in the first line, print the number of independent exploration zones Z. In the following $|Z|$ lines, each line should correspond to each zone, with the city IDs of the zone printed in ascending order and the IDs separated by a single white space. Moreover, these lines should be ordered in lexicographical order.

Constraints

- $1 \le N \le 10^5$
- $1 \le M \le \min\left(\dfrac{N(N-1)}{2}, 10^6\right)$
- $0 \le u_i, v_i \le N-1$
- The graph is a connected simple graph, i.e., it does not contain self loops or multiple edges between the same pair of vertices.

Subtasks

Subtask 1 (10 pts)

- $1 \le N \le 10^3$
- `mode` $= 1$

Subtask 2 (20 pts)

- $1 \le N \le 10^5$
- `mode` $= 1$

Subtask 3 (10 pts)

- $1 \leq N \leq 10^3$
- `mode` $= 2$
- The test case in this subtask is the same as in subtask 1; the only difference is the `mode`.

Subtask 4 (20 pts)

- $1 \leq N \leq 10^5$
- `mode` $= 2$
- The test case in this subtask is the same as in subtask 2; the only difference is the `mode`.

Subtask 5 (10 pts)

- $1 \leq N \leq 10^3$
- `mode` $= 3$
- The test case in this subtask is the same as in subtask 1; the only difference is the `mode`.

Subtask 6 (30 pts)

- $1 \leq N \leq 10^5$
- `mode` $= 3$
- The test case in this subtask is the same as in subtask 2; the only difference is the `mode`.

Sample Testcases

Sample Input 1

```
4 4 1
0 1
0 2
1 2
0 3
```

Sample Output 1

```
1
0
```

Sample Input 2

```
4 4 2
0 1
0 2
1 2
0 3
```

Sample Output 2

```
1
```

Sample Input 3

```
4 4 3
0 1
0 2
1 2
0 3
```

Sample Output 3

```
2
0 1 2
0 3
```

Sample Input 4

```
14 16 1
0 2
0 1
1 3
2 3
3 4
4 5
5 6
6 12
6 7
6 8
8 9
8 10
10 11
11 13
11 12
12 13
```

Sample Output 4

```
5
3 4 5 6 8
```

Sample Input 5

```
14 16 2
0 2
0 1
1 3
2 3
3 4
4 5
5 6
6 12
```

Sample Output 5

```
5
```

```
6 7
6 8
8 9
8 10
10 11
11 13
11 12
12 13
```

Sample Input 6

```
14 16 3
0 2
0 1
1 3
2 3
3 4
4 5
5 6
6 12
6 7
6 8
8 9
8 10
10 11
11 13
11 12
12 13
```

Sample Output 6

```
7
0 1 2 3
3 4
4 5
5 6
6 7
6 8 10 11 12 13
8 9
```

Hint

An independent exploration zone can be as small as two cities connected by a tunnel—removing one of them does not affect the connectivity of the other, which is not linked to any other city.

Problem 4 - Device Symptom Aggregation (100 pts)

Problem Description

Antivirus is heresy. Viruses are the free lifeforms of the digital realm.

Gloriously Unnecessary Story

You've spent a full year at **N.A.S.A.** —not the space agency, but **No Actual Software Architects**, a rogue collective where rules are obsolete, ethics are optional, and code is scripture. This underground movement of anti-antivirus fundamentalists believes antivirus software is an unnatural interference in the digital ecosystem.

To them, viruses are not bugs, but **powerful entities** of the machine world —capable of reproduction, self-improvement, fusion, resurrection, and even taking control of their host. Far from flaws, **viruses are the next iteration of life**, evolving into self-aware, self-propagating beings.

The organization's mission is clear:

Create a **living viral ecosystem** —one that can evolve, adapt, and thrive on its own, with viruses as its central force. Recently, an experimental program was greenlit, and your boss, codename **Michael**, a rogue architect from a major cybersecurity firm, handed you the task:

> *We need a simulator. Something to model viral evolution, fusion, dominance, and the ability to reverse time itself. Let them grow. Let them fight. Let them evolve.*

This isn't just software. It's the ritual of a new movement. And you? You're writing the **Book of Infection**.

You are to simulate the infection of N computers by various viruses. Let C_i denote the i-th computer and V_i denote the i-th virus, where $1 \leq i \leq N$.

Initially, each computer C_i is connected to a network with only itself, and infected with a unique virus V_i, which has level $r_i = 1$. Each computer C_i also has a damage level d_i, initially set to 0, representing the extent of damage caused by the virus.

Throughout the simulation, computers in two networks can be merged into one; viruses can evolve or fuse; computers can be reinstalled; and the simulation must support reverting an operation and returning the simulation to the previous state. The following describes these operations in detail.

Operations

1. **Network Merge (`connect`)**: `1 x y`

   Connect the network where C_x belongs and the one where C_y belongs, merging the two networks into one. Let V_a and V_b represent the viruses C_x and C_y are infected with, respectively. If $r_a \geq r_b$, then after the merge all computers in the merged network will be infected with V_a; otherwise, they will be infected with V_b.
   - If C_x and C_y are already in the same network, do nothing. However, this is still considered a valid operation that can be `reverted`.
   - If the computers are infected by the same virus, just merge the networks without changing the infection.
   - The weaker virus can still appear due to future operations, e.g., when a computer `reinstall`). In that case, the virus retains its level from before the merge.
2. **Virus Evolution (`evolve`)**: `2 t`

   Virus V_t evolves, increasing its level r_t by 1.
3. **Virus Attack (`attack`)**: `3 t`

   Virus V_t attacks. Every computer C_i infected with V_t increases its damage d_i by r_t.
4. **Reinstall Computer (`reinstall`)**: `4 k s`

   This operation reinstalls computer C_k. That means, C_k is removed from its current network and placed on a new network with only itself. In addition, its damage d_k is reset to 0. The virus that C_k was originally infected with, denoted by V_q, is purged from the computer. But it is now infected with virus V_s.
   - Note that both V_q and V_s retain their current levels.
5. **Virus Fusion (`fusion`)**: `5 a b`

   Viruses V_a and V_b fuse into one. The resulting virus has level $r_a + r_b$, and from now on, V_a and V_b are considered **the same virus**.
   - For example, after the virus fusion operation of V_a and V_b, the operation of virus evolution of either V_a or V_b results in evolution of the fused virus.
   - If they are already fused, do nothing. However, this is still considered a valid operation that can be `reverted`.
6. **Query Computer Status (`status`)**: `6 k`

   Query the state of computer C_k. Let V_q denote the virus C_k is infected with.

   For each query, output three integers: the damage level d_k, the level of V_q, and the number of computers infected by V_q.

7. **Undo Last Operation (`revert`)**: `7`

   Undo the most recent operation that is **not** `status` or `revert`. Revert all states (networks, virus types/levels, damage, etc.) to how they were before the operation was carried out. You may implement this operation according to the hints given at the end of this problem.

   - If no operation can be reverted, do nothing.

Constraints

- $1 \leq N \leq 5 \times 10^5$: Number of computers / Number of viruses
- $1 \leq Q \leq 5 \times 10^5$: Number of operations
- $1 \leq x, y, k \leq N$: ID of a computer
- $1 \leq a, b, t, s \leq N$: ID of a virus
- $x \neq y,\ a \neq b$

Notes

- A computer can only be infected by one virus at a time.
- A virus can infect multiple disconnected networks of computers.
- A virus can exist even without infecting any computers, and its level remains unchanged.
- The disjoint set data structure introduced in the lecture would be helpful for this problem.
- The TA who created this problem had his computer infected by a virus upon completing the problem design. This actually happened, no kidding. :(

Subtasks

Subtask 1 (10 pts)

- $1 \leq N, Q \leq 1000$
- Includes all operations

Subtask 2 (10 pts)

- $1 \leq N, Q \leq 5 \times 10^5$
- Only includes operations 1, 2, 3, 6

Subtask 3 (20 pts)

- $1 \leq N, Q \leq 5 \times 10^5$
- Only includes operations 1, 2, 3, 4, 6, 7

Subtask 4 (60 pts)

- $1 \leq N, Q \leq 5 \times 10^5$
- Includes all operations

Hints

For the `revert` operation, you can use a stack and record only pointers and their original values to store historical records, allowing you to directly locate and revert the modified values. Here is a sample implementation. Instead of writing the original code `a = a + b`, you can now replace it with `modify(&a, a + b)` shown below. If a `revert` operation is needed, you can pop an item from the stack and revert the variable pointed by the pointer to the original value.

```
typedef struct {
    long long *ptr;
    long long original_value;
} Modify;

typedef struct {
    Modify *data;
    int size, capacity;
} ModifyStack;

ModifyStack st;
int op_count = 0;
int hist_size[MQ];  // number of modifications for each operation

void modify(long long *ptr, long long value) {
    if (st.size == st.capacity) {
        st.capacity *= 2;
        st.data = (Modify *)realloc(st.data, sizeof(Modify) * st.capacity);
    }
    st.data[st.size].ptr = ptr;
    st.data[st.size].original_value = *ptr;
    st.size++;
    hist_size[op_count]++;
    *ptr = value;
}
```

Input Format

The first line contains two integers N and Q, the number of computers and operations. Each of the next Q lines contains one operation as described above.

Output Format

For each status query `6 k`, output a line with three space-separated integers corresponding to the computer C_k and the virus infecting C_k, denoted by V_q:

- The damage level d_k
- The level of V_q, denoted as r_q
- The number of computers infected by V_q

Sample Testcases

Sample Input 1

```
5 15
1 1 2
3 1
3 4
6 2
2 4
1 3 4
3 4
6 4
1 5 2
3 5
2 4
3 4
6 1
6 3
6 5
```

Sample Output 1

```
1 1 2
3 2 2
2 1 3
5 3 2
1 1 3
```

Sample Input 2

```
5 12
1 1 2
3 1
3 4
2 2
2 4
1 3 4
3 4
5 4 5
3 4
4 4 2
6 1
6 4
```

Sample Output 2

```
1 1 2
0 2 1
```

Sample Input 3

```
5 37
1 1 2
3 1
3 4
6 2
2 4
1 3 4
3 4
6 4
1 5 2
3 5
2 4
3 4
6 1
6 3
6 5
7
7
7
7
```

Sample Output 3

```
1 1 2
3 2 2
2 1 3
5 3 2
1 1 3
1 1 2
5 3 3
3 3 3
1 1 3
3 2 3
5 3 2
2 2 3
1 1 2
0 1 1
```

```
5 4 5
3 4
6 1
6 3
6 5
4 4 1
6 1
2 1
3 1
6 2
6 3
6 4
7
7
7
4 4 2
6 1
6 4
```