lab13

HackerRank

Problem Description

In the **glorious age of the Mauryan Empire**, Emperor **Ashoka the Great** reigned over a vast and diverse kingdom. After his transformative victory in the **Kalinga campaign**, Ashoka shifted his focus toward maintaining peace, unity, and order across the realm. To safeguard every province and ensure uninterrupted communication between them, he strategically deployed **imperial sentinels** at key outposts.

These outposts are interconnected by roads that form a **tree-like structure**—a sprawling network that spans the length and breadth of the empire. Each sentinel stationed at an outpost is responsible for monitoring the roads connected to it. In his pursuit of efficiency, Ashoka now seeks to **activate the minimum number of sentinels** such that **every road in the kingdom is monitored** by at least one of its two endpoints.

To ensure discipline and authority in his ranks, Ashoka's sentinels belong to **three distinct military ranks**, each representing a unique level of command in the Mauryan hierarchy:

- **SENAPATI** – The supreme military commander, at the top of the hierarchy
- **DANDANAYAKA** – The general responsible for justice and discipline, ranking just below the Senapati
- **CHATURANGINI** – The guardian of the fourfold army divisions (infantry, cavalry, elephants, and chariots), representing the foundational command level

This rank hierarchy plays a crucial role in organizing and sorting the sentinels based on their authority and responsibility. When required, the system must respect this military order, where:

`SENAPATI` > `DANDANAYAKA` > `CHATURANGINI`

To aid his mission, Ashoka has summoned his royal tacticians—you—to design and implement a sophisticated system that models the deployment plan. Your task is to build the **Imperial Surveillance System**, a software solution that:

- **Calculates the minimum number of active sentinels** required to achieve complete surveillance of all roads in the empire.
- **Sorts and displays all sentinels** based on their rank using a custom comparator that respects the military hierarchy.
- **Count** the number of sentinels in the empire who hold a **higher rank than a given sentinel**.

By blending strategy with structure, and optimization with object-oriented design, you will bring order to Ashoka's defense network and demonstrate the power of intelligent military engineering.

Design Specifications

Build an object-oriented system that models the kingdom as a tree and implements DP to solve the Minimum Vertex Cover problem. Your solution must include the following OOP concepts:

Class Hierarchy

Implement an abstract base class `Sentinel`:

- Holds fields like `id`, `name`, and `rank`.
- Includes a pure virtual method `countHigherRanked()` to return the number of sentinels in the empire who hold a higher rank than the current sentinel. Only sentinels with a **strictly higher rank** are counted (ignore same-rank sentinels regardless of ID).
- `Senapati` will always return 0 from this method since no rank is higher than `Senapati`.

Implement derived classes:

- `Senapati`, `Dandanayaka`, and `Chaturangini`, each with their own implementation of `countHigherRanked()`.

Tree Structure

Implement a template class `Kingdom<NodeType>` to:

- Add roads (edges) between outposts (nodes).
- Track all sentinels and apply dynamic programming to compute minimum vertex cover.
- Each sentinel's ID is the same as the node assigned to them in the tree. Nodes are numbered from `0` to `n-1`.
- Sentinels are assigned to nodes in the same order as their details appear in the input.

Custom Comparator

Define a comparator functor to order sentinels based on rank hierarchy:
`Senapati > Dandanayaka > Chaturangini`
If two sentinels have the same rank, the one with the smaller ID is considered higher.

You can define additional classes and methods as needed. Use of STL is allowed.

Input Format

1. Each test case corresponds to one configuration of the empire.
2. The first line contains n, the number of outposts (nodes).
3. The next n-1 lines each contain two integers u v, representing a road between outposts.
4. The next n lines each contain details of a sentinel at node i, in the format:

`<Name> <Rank>` — Rank is one of `SENAPATI`, `DANDANAYAKA`, or `CHATURANGINI`.

5. The next line contains q, the number of queries. Each of the following q lines is one of the following:

   Query Types:

   - 1 → Output the minimum number of active sentinels required.
   - 2 → Output a sorted list of all sentinels based on rank.
   - 3 → Output the count of sentinels with a higher rank than the sentinel with the given ID.

Constraints

- `1 ≤ n ≤ 10^5`
- `1 ≤ q ≤ 10^3`

Rank will always be one of: `SENAPATI`, `DANDANAYAKA`, `CHATURANGINI`

Tree structure is guaranteed (connected and acyclic)
Output Format

- For query 1: Output one integer — the minimum number of active sentinels.
- For query 2: Output one line of sorted sentinel IDs, separated by spaces.
- For query 3: Output one integer — the count of sentinels with a higher rank than the given sentinel ID.

Use `\n` instead of `std::endl` to avoid TLE.

Sample Input 0

```
5
0 1
0 2
2 3
2 4
Karna SENAPATI
Vikram DANDANAYAKA
Arjun SENAPATI
Bhima CHATURANGINI
Abhimanyu SENAPATI
3
1
2
3 2
```

Sample Output 0

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

Explanation 0

The empire has 5 outposts connected in a tree. Using dynamic programming for vertex cover, nodes 0 and 2 are selected for activating sentinels (e.g., Karna and Arjun), which gives full coverage with just 2 active sentinels.

Another optimal solution is selecting nodes 1 and 2, these also give full coverage of the tree.

For query 2, sorting by rank (SENAPATI > DANDANAYAKA > CHATURANGINI), we get SENAPATI nodes (IDs 0, 2, 4) first, followed by DANDANAYAKA (ID 1), then CHATURANGINI (ID 3).

Query 3 asks how many sentinels are higher ranked than the sentinel at node 2 (Arjun, SENAPATI). Since no rank is higher than SENAPATI, the answer is 0.

Sample Input 1

```
4
0 1
1 2
1 3
Mitra DANDANAYAKA
Ajeet SENAPATI
Chitra CHATURANGINI
Lalit DANDANAYAKA
3
1
2
3 0
```

Sample Output 1

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

Explanation 1

Minimum vertex cover selects node 1, resulting in only 1 active sentinel.

Sorting by rank gives SENAPATI (1), then DANDANAYAKA (0 and 3), then CHATURANGINI (2).

Query 3 asks how many sentinels are higher ranked than ID 0 (Mitra, DANDANAYAKA). Only Ajeet (ID 1, SENAPATI) is higher ranked.

Sample Input 2

```
4
0 1
1 2
2 3
Raj SENAPATI
Ajay CHATURANGINI
Sanjay DANDANAYAKA
Vijay DANDANAYAKA
4
1
2
3 3
3 1
```

Sample Output 2

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

Explanation 2

Minimum vertex cover can select node 1 and 3, resulting in 2 active sentinel. (There can be multiple selections that give optimal vertex cover).

Sorting by rank gives SENAPATI (0), then DANDANAYAKA (2 and 3), then CHATURANGINI (1).

Query 3 asks how many sentinels are higher ranked than ID 3 (Vijay, DANDANAYAKA). Only Raj (ID 0, SENAPATI) is higher ranked.

Query 4 asks how many sentinels are higher ranked than ID 1 (Ajay, CHATURANGINI). There are 3 sentinels ranked higher: Raj (ID 0, SENAPATI), Sanjay (ID 2, DANDANAYAKA), and Vijay (ID 3, DANDANAYAKA).