Engineering the Most Cost-Effective Routes

Your task is to design and implement the **most cost-effective network of pathways** connecting key locations across **IIT Madras**. The objective is to **minimize construction costs** while accounting for **real-world constraints** such as:

- **Green zones** (higher costs due to environmental impact)
- **High-traffic areas** (increased costs for durable roadways)
- **Department zones** (moderate costs due to infrastructure considerations)

Problem Statement

You are given a set of **nodes (locations)** and **edges (potential pathways)**, each associated with a **base cost**. However, the final construction cost may vary based on environmental and infrastructural factors:

- If a path passes through a **green zone**, the cost **doubles** (Base cost × 2).
- If a path is in a **high-traffic area**, the cost increases **fivefold** (Base cost × 5).
- If a path intersects a **department zone**, the cost is **tripled** (Base cost × 3).

Construct a **road network** that **connects all key locations** while **minimizing the total adjusted construction cost**, ensuring an efficient and practical pathway design for students, faculty, and visitors.

Input

- The **first line** contains two integers: **V** (number of vertices) and **E** (number of edges).
- The next **E lines** each contain: `u v wt type`

  where:

  - **u** = Source node
  - **v** = Destination node

- **wt** = Base weight of the edge
- **type** = Type of terrain affecting cost (e.g., green zone, traffic area, department zone)

Output

- Print the **minimum construction cost**, **modulo (1e9 + 7)**.

Algorithms to Use

Shortest Path Algorithms

- **Parallel Dijkstra's Algorithm**
- **Parallel Bellman-Ford Algorithm**
- **Parallel Johnson's Algorithm**

Minimum Spanning Tree (MST) Algorithms

- **Parallel Prim's Algorithm**
- **Parallel Borůvka's Algorithm**
- **Parallel Kruskal's Algorithm**

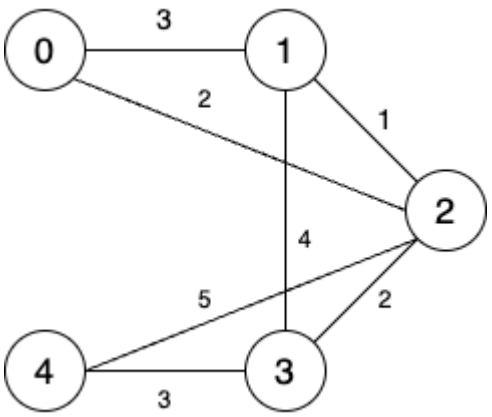
Note: You may not need to use all the algorithms to implement and solve the assignment. Above is the list of algorithms you are allowed to use. Make sure that your code is as parallel as possible.

Example Input

```
5 7
0 1 3 green
0 2 2 traffic
1 2 1 normal
1 3 4 dept
2 3 2 green
2 4 5 traffic
3 4 3 normal
```

Explanation

- **5** nodes, **7** edges. The graph will look like below,



Given Input Graph

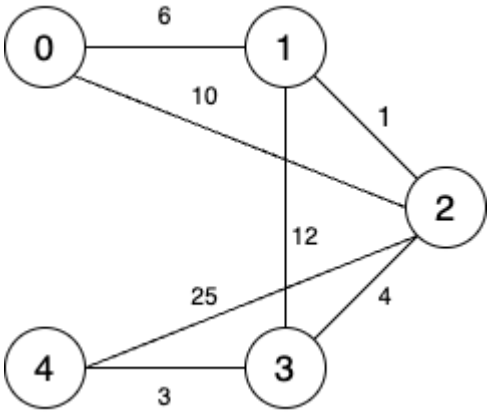
- Edge weights are adjusted based on the type:
  - **Green zone** → Cost ×2
  - **Traffic area** → Cost ×5
  - **Department zone** → Cost ×3

After normalizing the costs, the input graph will appear as shown below,

Modified Weights After Adjustments

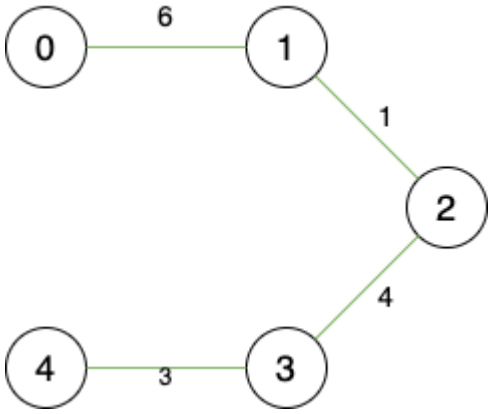
Edge	Base Weight	Type	Adjusted Weight
(1 - 2)	3	Green	**6** (3×2)
(1 - 3)	2	Traffic	**10** (2×5)
(2 - 3)	1	Normal	**1**
(2 - 4)	4	Dept	**12** (4×3)
(3 - 4)	2	Green	**4** (2×2)
(3 - 5)	5	Traffic	**25** (5×5)
(4 - 5)	3	Normal	**3**

Cost Table of Input Graph



Input Graph (Normalized Cost)

After selecting the minimum edges to connect all key locations, it will appear as shown below



Output Graph

Now, **Total cost** = 6+1+4+3 =14

Output = 14 mod (1e9+7) = 14

Example Output

```
14
```

Constraints

To be decided.

Note

- Submit a zip file with roll number (e.g., cs23m006.zip) which contains a single .cu file named with your roll number (e.g., cs23m006.cu).
- Ensure the output matches the expected format to avoid penalties.
- Execution time will be evaluated for this assignment.
- Submitting sequential code will be harshly penalized.
- You should call all the kernels between the timings shared in your `.cu` file.
- `int main()` can only be used for taking input, output and copying data.

- For any doubts please contact **Abhirup (cs23m006)** or **Tanmay (cs23m069)**.