Harry Potter and Enchanted Even Paths

Problem Description

The Gryffindor common room clock chimes past lights out, but Harry isn't in bed. He's on a mission. Fred and George Weasley's greatest prank yet —a dazzling display of Weasleys' Wildfire Whiz-bangs—depends on him retrieving their last batch of fireworks before Filch finds them. To do this successfully, the Weasley twins have given the Marauder's Map to Harry.

The Marauder's Map reveals Hogwarts as a simple, **undirected graph**: rooms as *nodes*, secret passages as *edges*, where *each passage has a travel time associated with it*. Harry wants to know the shortest path to the fireworks, but that's not all. Ever since his last encounter with Voldemort, Harry has a rather odd fear of paths with an odd number of passages**.** Therefore, he must use the **shortest path to the fireworks while ensuring it contains an even number of passages.**

To navigate efficiently, Harry recalls Hermione's lessons on data structures and algorithms. Since Hermione taught him about **minimum priority queue**, he is comfortable only with those. However, STL priority queue was implemented by Professor Snape, and he always has information about which student uses his spells (and implementations). And Harry definitely doesn't want the professor to learn about his mischief. Therefore, he needs to implement a **generic minimum priority queue** first, to find the shortest path.

Requirements

You are supposed to implement a generic min-priority queue in C++ using templates.

Following this, you need to create a class **`Node`** to represent the nodes, which stores the room name and other details of every room in the graph. The design decisions for **`Node`** are left up to you. However, design decisions will be evaluated based on good programming practices (private attributes, getters and setters etc.)

You must use your implemented generic priority queue to instantiate a **priority queue of `Node` `(or Node*)` objects**, which will be used to implement the Dijkstra algorithm. For further information on implementation of priority queue, please refer to the **Notes** section.

Input and Output Format

Input Format:

1. The first line of each test case contains **n** and **m**, which represent the number of rooms and number of bidirectional passages respectively.
2. Following **n** lines contain **n** unique room IDs. Each room ID is an alphanumeric string and corresponds to a unique room.
3. Following **m** lines contain one edge each in the form $\mathbf{a_i, b_i, w_i,}$ where $\mathbf{a_i}$ and $\mathbf{b_i}$ represent the room IDs and $\mathbf{w_i}$ represents the time it takes to traverse this edge.
4. The last line contains two room IDs: room ID of the source and room ID of the destination.

Output Format:

Print a single integer, representing the minimum time required to traverse an even-length path from the source to the sink.

If such a path does not exist, print **-1**.

Notes

You are expected to implement priority queues which follow O(log n) insertions and deletions. Inefficient implementations will fetch you partial marks. Your priority queue interface must mimic the STL implementation.
Please implement the following functions. You can use any additional functions as per your requirements.

1. `const size_t size() const`**:** Returns the number of elements currently stored in the priority queue.

2. `void push(const T&)`**:** Inserts a new element into the priority queue while preserving its heap property in O(log n) time.

3. `void pop()`**:** Removes the top (minimum) element from the priority queue in O(log n) time.

4. `const T& top() const`**:** Retrieves the top (minimum) element from the priority queue without removing it.

5. `const bool empty() const`: Checks whether the priority queue is empty, returning true if it contains no elements.

Please note that since you will be using this implementation of the min-priority queue to store `Node/Node*` objects, you are expected to design your class accordingly and don't forget to overload the comparison operator within your class.