```h
#include <string>
#include "galaxy.h"
//  Class Planet is a node in the route graph.  It contains a sequence
//  of edges plus additional fields to allow implementation of
//  Dijkstra's shortest-path algorithm.
class Planet {
public:
    Planet(const std::string& name) : name(name) {}
    void add(Edge* e) { edges.push_back(e); }

    // reset() clears the fields set by Dijkstra's algorithm so the
    // algorithm may be re-run with a different origin planet.
    void reset() { predecessor = nullptr; best_leg = Leg(); }

    // search() computes the shortest path from the Planet to each of the
    // other planets and returns the furthest planet by travel time.
    Planet* search(PriorityQueue<Planet, int(*)(Planet*, Planet*)>& queue);

    // make_itinerary() builds the itinerary with the earliest arrival
    // time from this planet to the given destination planet.
    Itinerary* make_itinerary(Planet* destination);

    // arrival_time() is the time to arrive at this planet from the
    // origin planet that was used to compute the most recent search().
    Time arrival_time() const { return best_leg.arrival_time; }

    // Debug-friendly output.
    void dump(Galaxy* galaxy);

    // Functions for priority queue:
    int get_priority() { return priority; }
    void set_priority(int new_priority) { priority = new_priority; }
    static int compare(Planet* left, Planet* right) {
        return Leg::compare(left->best_leg, right->best_leg);
    }
    const std::string name;
private:
    // relax_neighbors(): for each neighboring planet of this planet,
    // determine if the route to the neighbor via this planet is faster
    // than the previously-recorded travel time to the neighbor.
    void relax_neighbors(PriorityQueue<Planet, int(*)(Planet*, Planet*)>& queue);

    // edges shows the connections between this planet and it's
    // neighbors.  See class Edge.
    std::vector<Edge*> edges;

    // For Dijkstra's algorithm:
    Planet* predecessor;
    Leg best_leg;
    int priority;
};
```