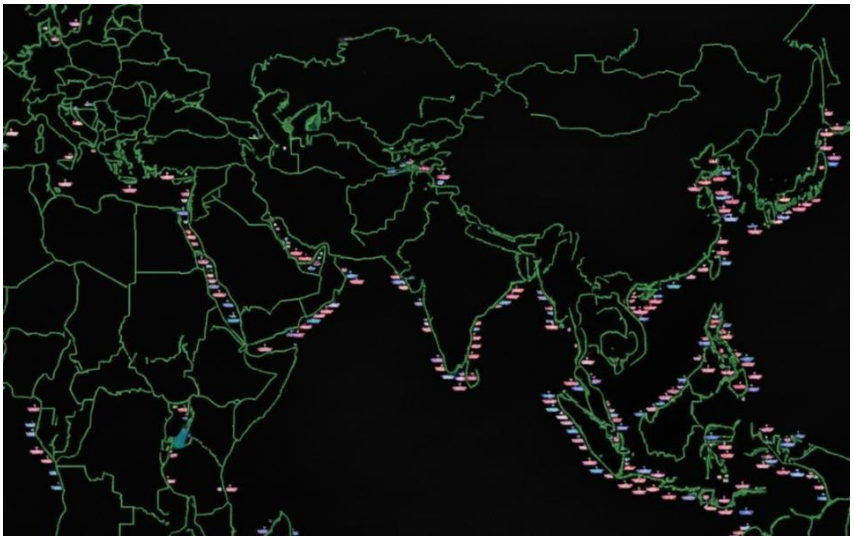
OceanRoute Nav – Maritime Navigation Optimizer

General Instructions

- **Group Setup:**
  - Work in groups of 2 (individual submissions allowed in rare cases).
- **Technologies:**
  - **Graphics:** Use SFML or GLUT for graphical implementation.
  - **Data Structures:**
    - Do not use built-in data structures or the STL library (no std::vector, std::map, etc.).
    - Implement your own **linked lists**, **queues**, **stacks**, **trees**, and **graphs** to handle port data, ship routes, cargo transfers, and user preferences.
    - You may use any number of data structures necessary for different modules.
- **Marking Criteria:**
  - **Understanding:** Clear demonstration of concepts and algorithms used.
  - **Implementation:** Correct and modular implementation of data structures, pathfinding algorithms, and graphics visualization.
  - **Innovation:** Inclusion of creative or extra features that enhance user experience or realism.
- **Deadline:** Submit by **7th December 2025, 11:59 pm (no extensions).**
- **Additional Guidelines:**
  - Use provided data files: Routes.txt and PortCharges.txt.
  - Submit well-documented code and a PDF of GPT prompts used during development (bonus marks for effective prompting).
  - Maintain modular code design — separate route finding, cargo scheduling, and visualization logic clearly.

Project Description

OceanRoute Nav is an intelligent **maritime navigation and logistics visualization system** that maps and optimizes cargo ship routes between international ports.
It processes complex oceanic route data — including port origins and destinations, dates, voyage durations, shipping company details, and docking delays.
The system helps logistics managers or captains select optimal routes by balancing **travel time**, **fuel cost**, and **weather impact**, while also managing port stays and layover times efficiently.



Core Features and Their Data Structures & Graphical Implementations

1. Route Data Representation

a) **Feature Description:**
Data from Routes.txt is parsed into a **graph** where vertices represent **ports** and edges represent **sea routes** between them.
Each edge stores voyage details such as **shipping company**, **fuel cost**, **voyage duration**, and **departure date**.

b) **Example:**

A user searches for routes from **Karachi** to **Singapore** on a given date.
The system displays both **direct** and **connecting** routes, showing time, cost, and ship details.

c) **Graphics Implementation:**

Ports are nodes on the map; sea routes are edges with width/color representing fuel cost or duration. Hovering over a route displays voyage details.

2. Ship Route Booking

a) **Feature Description:**
   Users can "book" cargo routes by selecting their **origin**, **destination**, and **date**.
   If no direct route exists, connecting ports are shown, ensuring docking and cargo transfer times are feasible.
b) **Example:**
   If a route from **Gwadar** to **Rotterdam** requires connections through **Dubai** and **Athens**, the system ensures docking and cargo handling times are compatible.
c) **Graphics Implementation:**
   All possible direct and connecting sea routes from the selected port are highlighted. Connecting paths appear in sequential animation to simulate movement.

3. Shortest or Cheapest Route Finder

a) **Feature Description:**
   Implements **Dijkstra's** and **A*** algorithms to compute **minimum cost** or **shortest time** paths.
   Uses a **priority queue** (heap) to enable efficient node processing, reducing computation time for optimal route discovery.
b) **Example:**
   A query from **Mumbai** to **New York** finds the least expensive path, perhaps via **Cape Town** and **Lisbon**, optimizing total cost and avoiding storm zones.
c) **Graphics Implementation:**
   The computed optimal path is highlighted in real-time.
   Ports light up as the algorithm explores them; the final route glows brightly once found.

4. Custom Ship Preferences

a) **Feature Description:**
   Users can set preferences like **preferred shipping companies**, **avoid specific ports**, or **limit maximum voyage time**.
   The system filters routes accordingly and recalculates paths dynamically.
b) **Example:**
   A user who only wants to use **Maersk Line** ships or avoid **storm-affected routes** sees only filtered connections in the final visualization.
c) **Graphics Implementation:**
   Preferred ports are displayed with special icons, and filtered routes appear as highlighted paths with unique visual effects.

5. Docking & Layover Management (Queue Implementation)

a) **Feature Description:**
Each port maintains a **queue** representing ships waiting for docking or cargo handling.
The queue ensures first-come-first-serve processing while managing available docking slots and timing constraints.

b) **Example:**

A ship arriving at **Singapore Port** waits in queue for 8 hours before docking if two other ships are already being serviced.

c) **Graphics Implementation:**

Port icons display dynamic dock queues using dashed lines for layovers.
As ships are processed, animations show ships entering and leaving the port area dynamically.

6. Multi-leg Route Generation (Linked List Implementation)

a) **Feature Description:**
For multi-stop journeys, a **linked list** represents each segment of the voyage.
Each node corresponds to one leg (port-to-port segment) and dynamically updates if the user modifies the journey (allows for dynamic and flexible route generation).

b) **Example:**

A multi-stop route from **Karachi → Dubai → Athens → Hamburg → Rotterdam** is visualized as a linked sequence of nodes.
Users can add or remove ports mid-journey, and the structure updates accordingly.

c) **Graphics Implementation:**

Each leg is shown as a sequential arrow chain (arrows connecting each stop); users can click to modify a segment interactively.

Graphical Query and Subgraph Generation

a) **Feature Description:**

Generates subgraphs based on user queries — for example, routes of a specific company, or routes active under calm weather conditions.

Irrelevant ports are excluded to simplify visualization, optimizing the visualization and making it easier for users to focus on the most pertinent routes.

b) **Example:**
Filtering for **CMA CGM** shows only ports and routes served by that company.
Alternatively, a weather filter hides all routes passing through stormy regions.

c) **Graphics Implementation:**
The subgraph appears with **faded inactive ports** and **highlighted active routes**.
A small legend explains route color meanings (distance, cost, or risk).

Advanced Algorithm Adaptations and Visuals

- **Dijkstra's and A***: These algorithms are adapted with visual feedback, enabling users to watch the step-by-step decision process. Each node is evaluated dynamically, and the path is built in real-time.
- **Pathfinding Enhancements**: Bidirectional search techniques are applied to improve long-distance route efficiency, with visual feedback showing progressively highlighted paths.

Additional Instruction about Provided Files:

The **Routes.txt** file contains information about the **available sea routes between ports**.
In particular, each entry in the file contains the following information
(in the same order, separated by spaces):

- **Origin Port**
- **Destination Port**
- **Date of Voyage**
- **Departure Time**
- **Arrival Time**
- **Voyage Cost (in USD)**
- **Name of Shipping Company**

It is important to note that the **departure and arrival times** are included in the route data; however, users are **not allowed to manually select preferred times of departure**.
Rather, users can only select a **preferred date of voyage**, and the system automatically manages available departures and layovers based on that date.

The inclusion of voyage times is essential for the **correct identification of connecting sea routes** and **feasible cargo transfer durations** between ports.

For example, if a *MaerskLine* vessel travels from **Karachi** to **Dubai** from **08:00 AM** to **06:00 PM**, a connecting voyage from Dubai cannot depart **earlier than 06:00 PM**.
Similarly, if a ship's next voyage from Dubai is scheduled at **09:00 AM** the following day, it implies a **15-hour layover** at Dubai Port, during which docking and cargo handling operations are performed.

Moreover, the **PortCharges.txt** file contains information about the **daily docking or port usage charges** for vessels at different ports.
This represents the **cost incurred per day** when a ship remains docked for refueling, cargo transfer, maintenance, or extended layovers exceeding **12 hours**.

Graph Generation Instructions

A **main graph** should be generated by reading data from the provided **Routes.txt** and **PortCharges.txt** files.

- **Vertices:** represent **ports**.
- **Edges:** represent **available shipping routes** between ports.
- Each **edge** represents the availability of routes between the ports. Moreover, it should store detailed voyage information, including:
    - Shipping company name
    - Voyage cost
    - Date of voyage
    - Departure and arrival times

The **port charge information** can also be stored in the **vertices** of the respective ports, allowing the system to calculate total costs when a ship remains docked beyond its scheduled layover time.