PA4.2 Graphs

Comprehensive Guide to Implementing Graph Applications

Lead TAs: Aamil and Faaiz

November 22, 2024

Contents

1 TASK 4 PART 1

In this part you'll be using two classes the Human and the Social Network class to create

1.1 Task 4 Part 1: Human Class

The `Human` class represents individuals in the social network. Each `Human` has attributes like name, age, and a unique identifier, along with a list of friends represented using a graph structure where the centre node is the human itself. You can use getPointer() to get the pointer for the human inside the class. The following functions are implemented for the `Human` class:

1.1.1 Function 4.1.1: *Human() Constructor*

- **Description**: Default constructor for the `Human` class. Initializes the `name`, `age`, and the graph to store friends.
- **Inputs**: None.
- **Outputs**: No return value.

1.1.2 Function 4.1.2: *Human(string name, int age) Constructor*

- **Description**: Parameterized constructor for the `Human` class. Sets the `name`, `age`, and initializes the graph to store friends. Assigns a unique ID to the human.
- **Inputs**:
  - *name* (`string`): The name of the human.
  - *age* (`int`): The age of the human.
- **Outputs**: No return value.

1.1.3 Function 4.1.3: *getUniqueID()*

- **Description**: Returns the unique ID of the human.
- **Inputs**: None.
- **Outputs**: An integer representing the unique ID of the human.

1.1.4 Function 4.1.4: *getName()*

- **Description**: Returns the name of the human.
- **Inputs**: None.
- **Outputs**: A string representing the name of the human.

1.1.5 Function 4.1.5: *getAge()*

- **Description**: Returns the age of the human.
- **Inputs**: None.
- **Outputs**: An integer representing the age of the human.

1.1.6 Function 4.1.6: *addFriend(*`shared_ptr<Human>` *friend)*

- **Description**: Adds a friend to the human's list of connections.
- **Inputs**:
  - *friend* (`shared_ptr<Human>`): A shared pointer to the friend being added.
- **Outputs**: No return value.
- **Edge Cases**:
  - If the `friend` pointer is `nullptr`, the function does nothing.
  - If the friend already exists in the list, the function does not duplicate the entry.

1.1.7 Function 4.1.7: *removeFriend(*`shared_ptr<Human>` *friend)*

- **Description**: Removes a friend from the human's list of connections.
- **Inputs**:
  - *friend* (`shared_ptr<Human>`): A shared pointer to the friend being removed.
- **Outputs**: No return value.
- **Edge Cases**:
  - If the `friend` pointer is `nullptr`, the function does nothing.
  - If the friend does not exist in the list, the function does nothing.

1.1.8 Function 4.1.8: *getFriends()*

- **Description**: Retrieves the list of friends connected to the human.
- **Inputs**: None.
- **Outputs**: A vector of shared pointers to the human's friends.
- **Edge Cases**:
  - If the human has no friends, the function returns an empty vector.

1.2 Task 4 Part 1: Social Network Class

In this task, you are required to implement functionality to manage a social network using the `SocialNetwork` and `Human` classes. These functions are designed to simulate a social network where each `Human` has a

set of friends, represented as vertices in a graph, so you can combine the graphs of those humans to create a social network graph.

1.2.1 Function 4.2.1: *addHuman(shared_ptr<Human> human)*

- **Description**: Adds a new human to the social network and initializes their connections.
- **Inputs**:
  - *human* (`shared_ptr<Human>`): A shared pointer to the human being added to the network.
- **Outputs**: No return value.
- **Edge Cases**:
  - If the provided `human` pointer is `nullptr`, the function does nothing.
  - If the human already exists in the social network, the function does not duplicate the entry.

1.2.2 Function 4.2.2: *removeHuman(shared_ptr<Human> human)*

- **Description**: Removes a human from the social network, ensuring that all their friendships are removed without affecting other connections in the network.
- **Inputs**:
  - *human* (`shared_ptr<Human>`): A shared pointer to the human being removed.
- **Outputs**: No return value.
- **Edge Cases**:
  - If the provided `human` pointer is `nullptr`, the function does nothing.
  - If the human does not exist in the social network, the function does nothing.

1.2.3 Function 4.2.3: *addFriendship(shared_ptr<Human> human1, shared_ptr<Human> human2)*

- **Description**: Establishes a friendship (connection) between two humans within the social network.
- **Inputs**:
  - *human1* (`shared_ptr<Human>`): A shared pointer to the first human.
  - *human2* (`shared_ptr<Human>`): A shared pointer to the second human.
- **Outputs**: No return value.
- **Edge Cases**:
  - If either of the human pointers is `nullptr`, the function does nothing.
  - If the humans are already friends, the function does not duplicate the connection.

1.2.4 Function 4.2.4: *removeFriendship(shared_ptr<Human> human1, shared_ptr<Human> human2)*

- **Description**: Removes the friendship (connection) between two humans within the social network.
- **Inputs**:
  - *human1* (`shared_ptr<Human>`): A shared pointer to the first human.
  - *human2* (`shared_ptr<Human>`): A shared pointer to the second human.
- **Outputs**: No return value.
- **Edge Cases**:
  - If either of the human pointers is `nullptr`, the function does nothing.
  - If the humans are not friends, the function does nothing.

1.2.5 Function 4.2.5: *getMutualFriends(shared_ptr<Human> human1, shared_ptr<Human> human2)*

- **Description**: Identifies and returns the mutual friends shared between two humans in the social network.
- **Inputs**:
  - *human1* (`shared_ptr<Human>`): A shared pointer to the first human.
  - *human2* (`shared_ptr<Human>`): A shared pointer to the second human.
- **Outputs**: A vector of shared pointers to the mutual friends.
- **Edge Cases**:
  - If either of the human pointers is `nullptr`, the function returns an empty vector.
  - If the humans have no mutual friends, the function returns an empty vector.

1.2.6 Function 4.2.6: *getGroups()*

- **Description**: Returns a vector of groups, where each group consists of humans who are all connected to each other (directly or indirectly). Each group is represented as a vector of shared pointers to `Human` objects.
- **Inputs**: None.
- **Outputs**: A vector of groups, where each group is a vector of shared pointers to humans.
- **Edge Cases**:
  - If the social network is empty, the function returns an empty vector.
  - If each human has no friends, the function returns a vector where each group contains a single human.

1.2.7 Function 4.2.7: *canBeConnected(shared_ptr<Human> human1, shared_ptr<Human> human2)*

- **Description**: Checks if two humans in the social network can be connected either directly or through a chain of mutual connections.
- **Inputs**:
  - *human1* (`shared_ptr<Human>`): A shared pointer to the first human.
  - *human2* (`shared_ptr<Human>`): A shared pointer to the second human.
- **Outputs**: Returns `true` if the two humans can be connected, otherwise returns `false`.
- **Edge Cases**:
  - If either of the `human` pointers is `nullptr`, the function returns `false`.
  - If `human1` and `human2` are the same, the function returns `true`.
  - If the two humans are in different disconnected groups, the function returns `false`.

1.2.8 Function 4.2.8: *connectionOrder(shared_ptr<Human> human1, shared_ptr<Human> human2)*

- **Description**: Finds and returns the connection chain between two humans if it exists. The chain represents the shortest path of mutual friends connecting `human1` to `human2`.
- **Inputs**:
  - *human1* (`shared_ptr<Human>`): A shared pointer to the first human.
  - *human2* (`shared_ptr<Human>`): A shared pointer to the second human.
- **Outputs**: A vector of shared pointers representing the chain of connections, including `human1` and `human2`. If no connection exists, the function returns an empty vector.
- **Edge Cases**:
  - If either of the `human` pointers is `nullptr`, the function returns an empty vector.
  - If `human1` and `human2` are the same, the function returns a vector containing only `human1`.
  - If no connection exists between `human1` and `human2`, the function returns an empty vector.

2 Task 4: Part 2 - Implementation of Flight Network Classes

This section details the implementation of the `Flight`, `Airport`, and `FlightNetwork` classes, which collectively define the behavior and structure of the flight network. Each class and its associated functions are explained below.

2.1 Class: `Flight`

2.1.1 Function 1.1: *Flight()*

- **Description**: Default constructor for the `Flight` class. Initializes all attributes to default values.
- **Inputs**: None.
- **Outputs**: None.

2.1.2 Function 1.2: *Flight(string flightNumber, shared$_p$tr < Airport > departureAirport, shared$_p$tr < Airport > destinationAirport, intdistance, intcost, FlightStatusstatus)*

- **Description**: Parameterized constructor for the `Flight` class. Initializes all attributes based on the provided arguments.
- **Inputs**:
  - *flightNumber* (`string`): Unique identifier for the flight.
  - *departureAirport* (`shared_ptr<Airport>`): Departure airport.
  - *destinationAirport* (`shared_ptr<Airport>`): Destination airport.
  - *distance* (`int`): Flight distance in kilometers.
  - *cost* (`int`): Flight cost in USD.
  - *status* (`FlightStatus`): Initial status of the flight, typically `SCHEDULED`.
- **Outputs**: None.

2.1.3 Function 1.3: *getDepartureAirport()*

- **Description**: Returns the departure airport for the flight.
- **Inputs**: None.
- **Outputs**: A shared pointer to the departure airport (`shared_ptr<Airport>`).

2.1.4 Function 1.4: *getDestinationAirport()*

- **Description**: Returns the destination airport for the flight.
- **Inputs**: None.
- **Outputs**: A shared pointer to the destination airport (`shared_ptr<Airport>`).

2.1.5 Function 1.5: *getFlightNumber()*

- **Description**: Returns the flight number.
- **Inputs**: None.
- **Outputs**: A string representing the flight number.

2.1.6 Function 1.6: *getCost()*

- **Description**: Returns the cost of the flight.
- **Inputs**: None.
- **Outputs**: An integer representing the cost of the flight in USD.

2.1.7 Function 1.7: *getDistance()*

- **Description**: Returns the distance of the flight.
- **Inputs**: None.
- **Outputs**: An integer representing the flight distance in kilometers.

2.1.8 Function 1.8: *getStatus()*

- **Description**: Returns the current status of the flight.
- **Inputs**: None.
- **Outputs**: The flight status (`FlightStatus`), which can be `SCHEDULED`, `TAKEN_OFF`, or `LANDED`.

2.1.9 Function 1.9: *setStatus(FlightStatus status)*

- **Description**: Updates the status of the flight.
- **Inputs**:
  - *status* (`FlightStatus`): The new status to set (`SCHEDULED`, `TAKEN_OFF`, or `LANDED`).
- **Outputs**: None.

2.2 Class: `Airport`

2.2.1 Function 2.1: *Airport()*

- **Description**: Default constructor for the `Airport` class. Initializes all attributes to default values.
- **Inputs**: None.
- **Outputs**: None.

2.2.2 Function 2.2: *Airport(string name, string city, string country)*

- **Description**: Parameterized constructor for the `Airport` class. Initializes all attributes based on the provided arguments.
- **Inputs**:
  - *name* (`string`): The name of the airport.
  - *city* (`string`): The city where the airport is located.
  - *country* (`string`): The country where the airport is located.
- **Outputs**: None.

2.2.3 Function 2.3: *getName()*

- **Description**: Returns the name of the airport.
- **Inputs**: None.
- **Outputs**: A string representing the name of the airport.

2.2.4 Function 2.4: *getCity()*

- **Description**: Returns the city where the airport is located.
- **Inputs**: None.
- **Outputs**: A string representing the city of the airport.

2.2.5 Function 2.5: *getCountry()*

- **Description**: Returns the country where the airport is located.
- **Inputs**: None.
- **Outputs**: A string representing the country of the airport.

2.2.6 Function 2.6: *addDepartureFlight(string flightNumber, shared*$_ptr < Airport > destination, intcost, intdistance, FlightStatusstatus)$

- **Description**: Adds a departure flight from this airport to the given destination airport. The flight is added to both the departure flight graphs based on distance and cost.
- **Inputs**:
  - *flightNumber* (`string`): Unique identifier for the flight.
  - *destination* (`shared_ptr<Airport>`): The destination airport.
  - *cost* (`int`): The cost of the flight in USD.
  - *distance* (`int`): The distance of the flight in kilometers.
  - *status* (`FlightStatus`): The status of the flight, typically `SCHEDULED`.
- **Outputs**: None.

2.2.7 Function 2.7: *addDepartureFlight(shared*$_ptr < Flight > flight)$

- **Description**: Adds a departure flight from this airport to the destination airport. The flight is added to the departure flight graphs based on distance and cost.
- **Inputs**:
  - *flight* (`shared_ptr<Flight>`): A shared pointer to the flight object.
- **Outputs**: None.

2.2.8 Function 2.8: *addArrivalFlight(string flightNumber, shared*$_ptr < Airport > source, intcost, intdistance, FlightStatusstatus)$

- **Description**: Adds an arrival flight to this airport from the given source airport. The flight is added to both the arrival flight graphs based on distance and cost.
- **Inputs**:
  - *flightNumber* (`string`): Unique identifier for the flight.
  - *source* (`shared_ptr<Airport>`): The source airport.
  - *cost* (`int`): The cost of the flight in USD.
  - *distance* (`int`): The distance of the flight in kilometers.
  - *status* (`FlightStatus`): The status of the flight, typically `SCHEDULED`.
- **Outputs**: None.

2.2.9 Function 2.9: *addArrivalFlight(shared*$_ptr < Flight > flight)$

- **Description**: Adds an arrival flight to this airport from the source airport. The flight is added to the arrival flight graphs based on distance and cost.
- **Inputs**:
  - *flight* (`shared_ptr<Flight>`): A shared pointer to the flight object.
- **Outputs**: None.

2.2.10 Function 2.10: *addFlight(shared*$_ptr < Flight > flight)$

- **Description**: Adds a flight to the departure and arrival flight graphs (both based on distance and cost) using the departure and destination airports.
- **Inputs**:
  - *flight* (`shared_ptr<Flight>`): A shared pointer to the flight object.
- **Outputs**: None.

2.2.11 Function 2.11: *TakeOff(string flightNumber)*

- **Description**: Updates the status of the flight with the given flight number to `TAKEN_OFF`.
- **Inputs**:
  - *flightNumber* (`string`): The flight number to be updated.
- **Outputs**: None.

2.2.12 Function 2.12: *Land(string flightNumber)*

- **Description**: Updates the status of the flight with the given flight number to `LANDED`.
- **Inputs**:
  - *flightNumber* (`string`): The flight number to be updated.
- **Outputs**: None.

2.2.13 Function 2.13: *hasFlight(string flightNumber)*

- **Description**: Checks if a flight with the given flight number exists in the airport.
- **Inputs**:
  - *flightNumber* (`string`): The flight number to check for.
- **Outputs**:
  - `true` if the flight exists, `false` otherwise.

2.2.14 Function 2.14: *getFlight(string flightNumber)*

- **Description**: Retrieves the flight with the specified flight number.
- **Inputs**:
  - *flightNumber* (`string`): The flight number to retrieve.
- **Outputs**: A shared pointer to the flight object (`shared_ptr<Flight>`).

2.2.15 Function 2.15: *getAllFlights()*

- **Description**: Retrieves all flights in the airport.
- **Inputs**: None.
- **Outputs**: A vector of shared pointers to flight objects (`vector<shared_ptr<Flight>>`).

2.3 Class: `FlightNetwork`

2.3.1 Function 3.1: *FlightNetwork()*

- **Description**: Default constructor for the `FlightNetwork` class. Initializes all attributes to default values, including the lists for airports and flights.
- **Inputs**: None.
- **Outputs**: None.

2.3.2 Function 3.2: *addAirport(shared_ptr¡Airport¿ airport)*

- **Description**: Adds the given airport to the network if it doesn't already exist.
- **Inputs**:
  - *airport* (`shared_ptr<Airport>`): A shared pointer to an airport object.
- **Outputs**: None.

2.3.3 Function 3.3: *addFlight(shared_ptr¡Flight¿ flight)*

- **Description**: Adds the given flight to the network. It ensures that the source and destination airports of the flight exist in the network before adding the flight to both airports' flight lists.
- **Inputs**:
  - *flight* (`shared_ptr<Flight>`): A shared pointer to a flight object.
- **Outputs**: None.

2.3.4 Function 3.4: *hasAirport(string name)*

- **Description**: Checks if an airport with the given name exists in the network.
- **Inputs**:
  - *name* (`string`): The name of the airport to check for.
- **Outputs**:
  - `true` if the airport exists, `false` otherwise.

2.3.5 Function 3.5: *hasFlight(string flightNumber)*

- **Description**: Checks if a flight with the given flight number exists in the network.
- **Inputs**:
  - *flightNumber* (`string`): The flight number to check for.
- **Outputs**:
  - `true` if the flight exists, `false` otherwise.

2.3.6 Function 3.6: *getAirport(string name)*

- **Description**: Returns the airport with the given name.
- **Inputs**:
  - *name* (`string`): The name of the airport to retrieve.
- **Outputs**:
  - A shared pointer to the airport with the specified name, or `nullptr` if no such airport exists.

2.3.7 Function 3.7: *getFlight(string flightNumber)*

- **Description**: Returns the flight with the given flight number.
- **Inputs**:
  - *flightNumber* (`string`): The flight number to retrieve.
- **Outputs**:
  - A shared pointer to the flight with the specified number, or `nullptr` if no such flight exists.

2.3.8 Function 3.8: *getShortestPath(shared_ptr¡Airport¿ source, shared_ptr¡Airport¿ destination)*

- **Description**: Retrieves the shortest path between the source and destination airports based on predefined criteria (e.g., shortest distance).
- **Inputs**:
  - *source* (`shared_ptr<Airport>`): The starting airport.
  - *destination* (`shared_ptr<Airport>`): The destination airport.
- **Outputs**: A vector of shared pointers to airports, representing the shortest path.

2.3.9 Function 3.9: *getCheapestPath(shared_ptr¡Airport¿ source, shared_ptr¡Airport¿ destination)*

- **Description**: Retrieves the cheapest path between the source and destination airports based on predefined criteria (e.g., cheapest cost).
- **Inputs**:
  - *source* (`shared_ptr<Airport>`): The starting airport.
  - *destination* (`shared_ptr<Airport>`): The destination airport.
- **Outputs**: A vector of shared pointers to airports, representing the cheapest path.

2.3.10 Function 3.10: *getFlightPlan(shared_ptr¡Airport¿ source, shared_ptr¡Airport¿ destination)*

- **Description**: Retrieves the flight plan for a route from the source to the destination, displaying the different available paths and their respective orders (e.g., shortest first, cheapest first).
- **Inputs**:
  - *source* (`shared_ptr<Airport>`): The starting airport.
  - *destination* (`shared_ptr<Airport>`): The destination airport.
- **Outputs**: A vector of shared pointers to airports representing the flight plan.

2.3.11 Function 3.11: *getAllFlights()*

- **Description**: Returns all the flights in the network, with no duplicates.
- **Inputs**: None.
- **Outputs**: A vector of shared pointers to all flights in the network.

2.3.12 Function 3.12: *getAllAirports()*

- **Description**: Returns all the airports in the network.
- **Inputs**: None.
- **Outputs**: A vector of shared pointers to all airports in the network.

2.3.13 Function 3.13: *getBusiestAirport()*

- **Description**: Returns the busiest airport in the network, based on traffic or other criteria.
- **Inputs**: None.
- **Outputs**: A shared pointer to the busiest airport in the network.

2.3.14 Function 3.14: *getLamestAirport()*

- **Description**: Returns the least busy airport in the network, based on traffic or other criteria.
- **Inputs**: None.
- **Outputs**: A shared pointer to the least busy (lamest) airport in the network.

2.3.15 Function 3.15: *OptimizedGraph(bool distance)*

- **Description**: Returns an optimized graph for the flight network, either based on distance or other criteria.
- **Inputs**:
  - *distance* (`bool`): A flag indicating whether to optimize by distance (`true`) or by cost (`false`).
- **Outputs**: A shared pointer to an optimized graph of airports.

2.3.16 Function 3.16: *alternateRouteForFlight(shared_ptr¡Flight¿ flight)*

- **Description**: Finds alternate routes for a flight in case the direct route is unavailable. Compares the new path's distance and cost to the original, selecting the least impacted route.
- **Inputs**:
  - *flight* (`shared_ptr<Flight>`): The flight for which alternate routes are being considered.
- **Outputs**: A vector of shared pointers to airports representing the alternate route with the least increase in distance and cost.

2.3.17 Function 3.17: *AirportsReachable(shared_ptr¡Airport¿ airport)*

- **Description**: Returns a list of airports reachable from a given airport, considering direct or indirect connections.
- **Inputs**:
  - *airport* (`shared_ptr<Airport>`): The starting airport from which the reachable airports are to be found.
- **Outputs**: A vector of vectors, where each inner vector represents airports reachable from the given airport.

3 TASK 4 : PART 3

Please read the comments in the H file