# Problem Statement

## Objective

Your task is to create a zoo simulation program where different animals can exist in various enclosures, and a visitor can navigate through the zoo using a map represented by a graph.

## Class Descriptions

### Animal Base Class and Derived Classes

1. **Animal Class**

```
class Animal {
public:
    Animal(string name);
    virtual void makeSound() const = 0; // Pure virtual function
    virtual ~Animal(); // Virtual destructor

    // Copy constructor
    Animal(const Animal& other);

    // ... (if needed, other member functions and data members)

private:
    string name;
};
```

1. **Derived Classes**

Implement derived classes `Lion`, `Elephant`, and `Bird`. Each class will override the `makeSound()` method to produce a sound specific to the animal type (e.g., Lion roars, Elephant trumpets, Bird tweets).

```
class Lion : public Animal {
public:
    Lion(string name);
    void makeSound() const override; // Should print "Roar!"
    // ... (if needed, other member functions)
};

class Elephant : public Animal {
public:
    Elephant(string name);
    void makeSound() const override; // Should print "Trumpet!"
    // ... (if needed, other member functions)
};

class Bird : public Animal {
public:
    Bird(string name);
    void makeSound() const override; // Should print "Tweet!"
    // ... (if needed, other member functions)
```

```
};
```

## Enclosure Class

Implement an `Enclosure` class which will hold a collection of animals. It should have methods to add or remove animals and to list all animals in the enclosure.

```
class Enclosure {
public:
    Enclosure(string name);
    void addAnimal(Animal* animal);
    void removeAnimal(Animal* animal);
    void listAnimals() const;
    string getName() const;

    // ... (if needed, other member functions and operator overloads)
private:
    string name;
    vector<Animal*> animals;
};
```

## ZooGraph Class

```
class ZooGraph {
public:
    ZooGraph();
    void addEnclosure(Enclosure* enclosure);
    void addPath(string enclosureA, string enclosureB);
    vector<string> findShortestPath(string startEnclosure, string endEnclosure);

    // ... (if needed, other member functions and data members)
private:
    map<string, vector<string>> adjacencyList; // Data member to store adjacency
list representation of the graph
    map<string, Enclosure*> enclosures; // Data member to store enclosures by
their names
};
```

## Template Class

```
template<typename T>
class Info {
public:
    Info(T info);
    T getInfo() const;
    // ... (if needed, other member functions)
private:
    T info;
};
```

## Test Cases

Implement a `main` function to demonstrate the working of your classes and methods. Here are some test cases that you might use:

```
int main() {
    // Test Case 1: Animal Sounds
    Lion lion1("Lion King");
    lion1.makeSound(); // Output: Roar!

    Elephant elephant1("Dumbo");
    elephant1.makeSound(); // Output: Trumpet!

    Bird bird1("Tweety");
    bird1.makeSound(); // Output: Tweet!

    // Test Case 2: Enclosure Operations
    Enclosure enclosure1("Savannah");
    enclosure1.addAnimal(&lion1);
    enclosure1.addAnimal(&elephant1);
    enclosure1.listAnimals();
    // Output:
    // Animal: Lion King, Sound: Roar!
    // Animal: Dumbo, Sound: Trumpet!

    enclosure1.removeAnimal(&lion1);
    enclosure1.listAnimals();
    // Output: Animal: Dumbo, Sound: Trumpet!

    // Test Case 3: ZooGraph Operations
    ZooGraph zoo;

    Enclosure enclosure1("Savannah");
    enclosure1.addAnimal(new Lion("Lion King"));
    enclosure1.addAnimal(new Elephant("Dumbo"));

    Enclosure enclosure2("Bird House");
    enclosure2.addAnimal(new Bird("Tweety"));

    Enclosure enclosure3("Jungle");
    Enclosure enclosure4("Aquarium");
    Enclosure enclosure5("Desert");
    Enclosure enclosure6("Mountain");
    Enclosure enclosure7("Swamp");

    zoo.addEnclosure(&enclosure1);
    zoo.addEnclosure(&enclosure2);
    zoo.addEnclosure(&enclosure3);
    zoo.addEnclosure(&enclosure4);
    zoo.addEnclosure(&enclosure5);
    zoo.addEnclosure(&enclosure6);
    zoo.addEnclosure(&enclosure7);

    zoo.addPath("Savannah", "Jungle");
    zoo.addPath("Savannah", "Desert");
```

```
    zoo.addPath("Jungle", "Mountain");
    zoo.addPath("Mountain", "Swamp");
    zoo.addPath("Swamp", "Aquarium");
    zoo.addPath("Aquarium", "Bird House");
    zoo.addPath("Bird House", "Desert");
    zoo.addPath("Desert", "Savannah");

    vector<string> path1 = zoo.findShortestPath("Savannah", "Bird House");
    // Printing Path 1
    cout << "Path 1 (Savannah to Bird House): ";
    for(const auto& enclosure : path1) {
        cout << enclosure << " ";
    }
    cout << endl;
    // Path 1 (Savannah to Bird House): Savannah Jungle Mountain Swamp Aquarium
Bird House


    vector<string> path2 = zoo.findShortestPath("Jungle", "Desert");

    // Printing Path 2
    cout << "Path 2 (Jungle to Desert): ";
    for(const auto& enclosure : path2) {
        cout << enclosure << " ";
    }
    cout << endl;

    vector<string> path3 = zoo.findShortestPath("Mountain", "Bird House");
    // Path 2 (Jungle to Desert): Jungle Savannah Desert

    // Printing Path 3
    cout << "Path 3 (Mountain to Bird House): ";
    for(const auto& enclosure : path3) {
        cout << enclosure << " ";
    }
    cout << endl;
    // Path 3 (Mountain to Bird House): Mountain Swamp Aquarium Bird House

    // Test Case 4: Template Class Usage
    Info<string> info1("Some information");
    cout << info1.getInfo(); // Output: Some information

    // Test Case 5: Copy Constructor Usage
    {
        Lion originalLion("Simba", 4);
        Lion copiedLion = originalLion; // Using copy constructor

        cout << "Original Lion Name: " << originalLion.getName() << ", Age: " <<
originalLion.getAge() << endl;
        cout << "Copied Lion Name: " << copiedLion.getName() << ", Age: " <<
copiedLion.getAge() << endl;

        // Expected output:
        // Original Lion Name: Simba, Age: 4
        // Copied Lion Name: Simba, Age: 4
    }
```

```
    // Test Case for Destructor
    {
        Animal *dynamicAnimal = new Lion("Scar", 5);
        delete dynamicAnimal; // Here, destructor will be called, can have a
print statement inside destructor to check.
    }
    return 0;
}
```

## Note

- Ensure to release memory properly to prevent memory leaks.
- Use proper error handling wherever necessary.