Graph-Based Adversarial Thinking Assignment

## Title: Analyzing Fastest Routes and Simulating Adversarial Attacks in El Paso

### Introduction:

In this assignment, you will delve into the fundamental concepts of graph theory to explore real-world transportation networks. El Paso's locations, represented as nodes, will connect through possible routes constructed as edges. In graph theory, nodes symbolize entities while edges show relationships or connections between these entities. With a focus on understanding travel routes, you'll assess potential adversarial risks displayed as alterations in node connections or weights, simulating disruptions. The task touches on Bloom's Taxonomy "remembering" level, necessitating comprehension of basic graph constructions to make informed decisions. Localized knowledge brings a crucial understanding of how disruptions might influence familiar region pathways, aiding in developing strategies to maintain route integrity.

### Programming Task:

Implement a Python script using `networkx` to construct a graph representing specific El Paso locations. Your goal is to compute and display the shortest path, representing the fastest route based on hypothetical weights. Simulate an adversarial attack that alters the route network and rectify the altered paths.

### Code Implementation Steps:

1. **Setup Python Environment**:

- Ensure you have `networkx` and `matplotlib` installed. Use the command: `pip install networkx matplotlib`.

2. **Define Locations and Distances**:

- Represent each location as a node.

- For this task, manually construct a graph with estimated distances (weights) between key locations.

3. **Construct the Graph**:

- Use `networkx` to create a graph with nodes representing each location in the provided list.

- Add weighted edges to simulate approximate driving distances.

4. **Find the Fastest Route**:

- Use Dijkstra's shortest path algorithm from `networkx` to determine and print the fastest route between two selected nodes (e.g., from "Walmart Supercenter" to "Lost Kingdom Water Park").

5. **Simulate Adversarial Attack**:

- Introduce a disturbance by increasing the weight of a crucial edge or removing a node.

- Recalculate the shortest path and observe the effect on the route.

6. **Visualize the Graph**:

- Employ `matplotlib` to visualize the original and modified graphs, marking out the shortest path clearly.

### Expected Output:

- Submit a Python script that prints the shortest path details (before and after attack).

- Provide graph visualizations representing the two scenarios (pre and post-attack).

### Adversarial Angle:

Simulated adversarial interference in the network signifies potential real-world threats that can disrupt transportation systems. The goal is to understand the impact of these disruptions on infrastructure security and evaluate how networks can adapt to ensure uninterrupted services.

### Sense of Belonging:

Nodes depicted as familiar El Paso locations give localized insights into route intricacies. Recognition of real-world destinations underlines the significance of graph theory in comprehending regional transportation dynamics and demonstrates how disruptions could impact daily commutes.

```python
import networkx as nx

import matplotlib.pyplot as plt

# Sample data with nodes and manually assigned weights

locations = {

'Walmart Supercenter': (31.7770748, -106.3846817),

'Abercrombie Kids': (31.7753257, -106.3791262),

'Lost Kingdom Water Park': (31.7906268, -106.4149103),

'First American Bank': (31.7609075, -106.3575582),

'WestStar': (31.7784415, -106.4214496),
'Cielo Vista': (31.7750435, -106.37921),

'Rack Room Shoes': (31.7831908, -106.4130581),

'Harbor Freight': (31.7904524, -106.333426),

'Eastwood High School': (31.7722351, -106.3555024)

}

# Constructing graph

G = nx.Graph()

# Adding nodes

for place, coords in locations.items():

G.add_node(place, pos=coords)

# Adding edges with hypothetical weights

edges = [

('Walmart Supercenter', 'Abercrombie Kids', 3),

('Abercrombie Kids', 'Lost Kingdom Water Park', 5),

('Walmart Supercenter', 'Rack Room Shoes', 7),

('Rack Room Shoes', 'Lost Kingdom Water Park', 2),

('First American Bank', 'Eastwood High School', 4),

('WestStar', 'Lost Kingdom Water Park', 6)

]

G.add_weighted_edges_from(edges)

# Display shortest path before attack

source, target = 'Walmart Supercenter', 'Lost Kingdom Water Park'

shortest_path = nx.dijkstra_path(G, source, target)

print("Shortest path before adversarial attack:", shortest_path)

# Visualize original graph and path

pos = nx.get_node_attributes(G, 'pos')

nx.draw(G, pos, with_labels=True, node_size=2000, node_color='lightblue')
nx.draw_networkx_edges(G, pos, edgelist=edges, edge_color='gray')

nx.draw_networkx_edges(G, pos, edgelist=nx.utils.pairwise(shortest_path), edge_color='red', width=2)

plt.title("Original Fastest Route")

plt.show()

# Simulate adversarial attack (increased weight)

G.edges['Walmart Supercenter', 'Rack Room Shoes']['weight'] *= 3

# Recalculate shortest path after attack

altered_path = nx.dijkstra_path(G, source, target)

print("Shortest path after adversarial attack:", altered_path)

# Visualize modified graph and path

nx.draw(G, pos, with_labels=True, node_size=2000, node_color='lightblue')

nx.draw_networkx_edges(G, pos, edgelist=edges, edge_color='gray')

nx.draw_networkx_edges(G, pos, edgelist=nx.utils.pairwise(altered_path), edge_color='red', width=2)

plt.title("Altered Fastest Route")

plt.show()
```

This code establishes the basis for understanding network disruptions in transportation routes, empowering students to analyze how such disruptions manifest and can be prevented.