Assignment 2:

Step-by-Step Solution:

1. **Model the Problem as a Graph:**
    - Treat each city as a node.
    - Treat each ticket (connection between cities) as a directed edge between nodes.
2. **Input Representation:**
    - Store the tickets in a data structure where each ticket indicates a directed edge from one city (start) to another city (destination).
3. **Constructing the Graph:**
    - Use an adjacency list to represent the graph where each city points to a list of cities it directly connects to.
4. **Identify the Starting Point:**
    - Since it's given that your son started in Kiev, we'll begin our traversal from Kiev.
5. **Traversal Algorithm (DFS):**
    - Start from Kiev and perform a Depth-First Search to explore all possible paths.
    - Maintain a list to keep track of the visited cities to avoid cycles and revisitations.
6. **Route Reconstruction:**
    - As we traverse the graph starting from Kiev, construct the route by appending each city to a result list.
7. **Output:**
    - After traversal completes, the result list will contain the sequence of cities visited from Kiev based on the available tickets.

Python Implementation:

Here is a Python implementation using Depth-First Search (DFS):

```
python
Copy code
from collections import defaultdict

def find_route(start_city, tickets):
    # Build the graph using an adjacency list
    graph = defaultdict(list)
    for ticket in tickets:
        from_city, to_city = ticket.split('-')
        graph[from_city].append(to_city)

    # Initialize visited list and result list
    visited = []
    route = []

    # Define DFS function
    def dfs(city):
        nonlocal found
        if found:
            return
```

```
        visited.append(city)
        route.append(city)
        if len(route) == len(tickets) + 1:  # We found the full route
            found = True
            return
        for neighbor in graph[city]:
            if neighbor not in visited:
                dfs(neighbor)
                if found:
                    return
        visited.pop()
        route.pop()

    # Start DFS from start_city (Kiev)
    found = False
    dfs(start_city)

    return route if found else None

# Given tickets
tickets = [
    "Paris-Skopje", "Zurich-Amsterdam", "Prague-Zurich",
    "Barcelona-Berlin", "Kiev-Prague", "Skopje-Paris",
    "Amsterdam-Barcelona", "Berlin-Kiev", "Berlin-Amsterdam"
]

# Starting city (given in the problem)
start_city = "Kiev"

# Find the route
route = find_route(start_city, tickets)

# Output the route
if route:
    print("The route your son traveled is:", " -> ".join(route))
else:
    print("No valid route found.")
```

Explanation:

- **Graph Construction:** The `defaultdict(list)` helps in constructing an adjacency list where each key (city) maps to a list of neighboring cities.
- **DFS Traversal:** The `dfs` function recursively explores all possible paths starting from Kiev (`start_city`). It backtracks whenever a dead-end is reached or when all cities are visited.
- **Route Reconstruction:** The `route` list accumulates cities as they are visited, forming the sequence of cities traveled.
- **Output:** Finally, the route is printed in the format `Kiev -> ... -> Last City`.

This approach ensures that we find and print the exact sequence of cities visited by your son, starting from Kiev and using the provided train tickets.