Traveling Salesman

Given a list of cities, each have a latitude and a longitude value, write a function compute a path going through all the city exactly once, starting with the first city in the list, and return the total length & path, in the order the cities should be visited.

The distance between two cities will be Euclidean distance.

For example, the cities list is shown below:

```
[[Bac Giang, 21.2670, 106.2000],
[Bac Kan, 22.1333, 105.8333],
[Bac Lieu, 9.2804, 105.7200],
[Bac Ninh, 21.1861, 106.0763],
[Bien Hoa, 10.9700, 106.8301],
[Buon Me Thuot, 12.6670, 108.0500],
[Ca Mau, 9.1774, 105.1500]]
```

Vietnamese:
Cho một danh sách thành phố kèm kinh độ và vĩ độ, hãy viết một hàm tính toán một con đường đi qua tất cả thành phố, mỗi thành phố duy nhất một lần, bắt đầu từ thành phố đầu tiên trong danh sách. Kết quả trả về một tuple gồm đường đi (danh sách thành phố theo thứ tự sẽ đi) và tổng độ dài đường đi.

Khoảng cách giữa hai thành phố áp dụng khoảng cách Euclid.

Complete the `traveling_sale_man` function below. It MUST return a tuple contain a list of cities represent the path of each city should be visited and a number represent the total distance.

```
def traveling_sale_man(cities):
    """
    Return the list of cities in order they should be visited & total length.

    @param cities: a Python array representing a list of cities, each city is
    a list with the first value is the city name, second and third is
    latitude, longitude: Ex:

    [[Bac Giang, 21.2670, 106.2000],
    [Bac Kan, 22.1333, 105.8333],
    [Bac Lieu, 9.2804, 105.7200],
    [Bac Ninh, 21.1861, 106.0763],
    [Bien Hoa, 10.9700, 106.8301],
    [Buon Me Thuot, 12.6670, 108.0500],
    [Ca Mau, 9.1774, 105.1500]]

    @return: return a tuple contain a list of cities represent the path of
    each city should be visited and a number represent the total distance.
    """
    pass
```

For example:

```
>>> traveling_sale_man(
                        [[Bac Giang, 21.2670, 106.2000],
                        [Bac Kan, 22.1333, 105.8333],
                        [Bac Lieu, 9.2804, 105.7200],
                        [Bac Ninh, 21.1861, 106.0763],
                        [Bien Hoa, 10.9700, 106.8301],
                        [Buon Me Thuot, 12.6670, 108.0500],
                        [Ca Mau, 9.1774, 105.1500]]
                        )

(['Bac Giang', 'Bac Kan', 'Bac Ninh', 'Buon Me Thuot', 'Bien Hoa', 'Bac Lieu',
'Ca Mau'],15.354182520350108)
```

Non Overlapping Paths

Given an undirected and unweighted graph with n vertex, each vertex will be name from 1 to n (n <= 1000), each vertex will have a list of connected vertices that it have connection to. Write a function to find all non overlapping paths from a Start vertex to an End vertex within the graph (1 <= start,end <= n, start != end).

Two paths are Non Overlapping mean that they not contained any same vertex excerpt the Start and End vertices.

Only valid inputs will be provided to your function.

Vietnamese:
Cho một đồ thị vô hướng và không có trọng số gồm n (1 <= n <= 1000) đỉnh mỗi đỉnh được đánh số từ 1 đến 1000. Mỗi đỉnh có kèm theo danh sách các đỉnh có liên kết với nó. Hãy viết hàm tìm tất cả đường đi không trùng lắp từ một đỉnh bắt đầu đến một đỉnh kết thúc. (1 <= đỉnh bắt đầu,kết thúc <= 1000, đầu != kết thúc).

Đường đi không trùng lắp: là đường đi không có đỉnh chung nào khác ngoại trừ điểm đầu và điểm kết thúc.

Chỉ có dữ liệu đúng được truyền vào hàm.

Complete the `non_overlapping_paths` function below. It MUST return a list of paths, each path contain the list of vertices represent the order of each vertex should be visited to travel from Start vertex to End vertex (the Start and End are included).

```
def non_overlapping_paths(start,end,graph):
    """
    Find all non overlapping paths from start to end within the graph.

    @param start: start vertex, ex: 3
    @param end: end vertex, not the same as start, ex: 9
    @param graph: an undirected and unweighted graph represent as a list of
    vertex, each vertex is a tuple contain its number and a list of other
    vertices that it connected to. ex:
    [(1,[2]),
    (2,[1,3,7]),
    (3,[2,4]),
    (4,[3,5]),
    (5,[4,10]),
    (6,[7]),
    (7,[6,2,8]),
    (8,[7,9]),
    (9,[8,10]),
    (10,[9,5])]

    @return: return a list of paths, each path contain the list of vertices
    represent the order of each vertex should be visited to travel from Start
    vertex to End vertex (the Start and End are included).
    """
    pass
```

For example:

```
>>>
non_overlapping_paths(3,9,[(1,[2]),(2,[1,3,7]),(3,[2,4]),(4,[3,5]),(5,[4,10]),(
6,[7]),(7,[6,2,8]),(8,[7,9]),(9,[8,10]),(10,[9,5])])

[[3,2,7,8,9],[3,4,5,10,9]]
```