Assignment 2: My Spatial Databases

1820212030 Eng Jie

Introduction

This report provides an overview of the development and execution of a Python-based spatial querying system designed for identifying specific Points of Interest (POIs) such as ATMs and restaurants within a specified vicinity. The system has been developed using Python libraries such as Pandas, GeoPandas, Shapely, Rtree, and Haversine to manage and query spatial data effectively.

Objectives

The main objectives of the developed system include:

1. Building an in-memory spatial database for POIs to support efficient spatial range queries and nearest neighbor queries.
2. Demonstrating the efficiency of spatial indexing in querying processes.
3. Utilizing the spatial database to identify the nearest ATM and count the number of restaurants within a specified distance from predefined points.

Methodology

The system implementation involved the following steps:

1. Data Preparation: Loading POI data from a CSV file into a GeoDataFrame and constructing a spatial index using the R-tree algorithm.
2. Spatial Indexing: Implementing an index-building function to facilitate efficient spatial queries.
3. Spatial Queries:
    a) Range Query: Developing a function to retrieve all POIs within a specified distance from a given point.
    b) Nearest Neighbor Query: Creating a function to find the nearest POI of a specific type relative to a given location.
4. Efficiency Comparison: Comparing the performance of spatially indexed queries against brute-force approaches.
5. Integration with Haversine Formula: Utilizing the Haversine formula to calculate real-world

distances between geographic coordinates.

Implementation

The code was structured into several functions, each responsible for a specific aspect of the spatial querying process:

- IndexBuilding: Constructs a spatial index for the provided POI data.

```
# 1. Index-building function
def IndexBuilding(file_path):
    poi_data = pd.read_csv(file_path)
    poi_data['geometry'] = poi_data.apply(lambda row: Point(row['wgs_lng'], row['wgs_lat']), axis=1)
    gdf = gpd.GeoDataFrame(poi_data, geometry='geometry')

    idx = rt_index.Index()
    for poi_id, row in gdf.iterrows():
        idx.insert(int(poi_id), (row.geometry.x, row.geometry.y, row.geometry.x, row.geometry.y), obj=row)

    return idx, gdf
```

- RangeQuery: Executes spatial range queries utilizing the spatial index.

```
# 2. Range query function
def RangeQuery(query_range, type_regex_str, idx, gdf):
    results = []
    regex = re.compile(type_regex_str)

    if isinstance(query_range, tuple) and len(query_range) == 3:  # Circle range
        center_point = (query_range[0], query_range[1])  # (lat, lon)
        radius = query_range[2]  # Radius in kilometers

        for id in idx.intersection((center_point[1] - radius/111, center_point[0] - radius/111,
                                    center_point[1] + radius/111, center_point[0] + radius/111), objects=True):
            row = gdf.iloc[id.id]
            # Ensure type_code is treated as a string
            type_code_str = str(row['type_code'])
            if regex.match(type_code_str):  # Use the string version for regex matching
                # Note: haversine() expects (lat, lon)
                dist = haversine(center_point, (row.geometry.y, row.geometry.x))
                if dist <= radius:
                    results.append(row)
    else:
        raise ValueError("Invalid query_range format. Must be a circle (x, y, radius).")
    return pd.DataFrame(results)
```

- NNQuery: Performs nearest neighbor searches using the spatial index.

```python
def NNQuery(query_point, type_regex_str, idx, gdf):
    regex = re.compile(type_regex_str)
    nearest_poi = None
    min_dist = float('inf')  # Set to a very high number initially

    for id in idx.intersection((query_point[1] - 0.05, query_point[0] - 0.05, query_point[1] + 0.05, query_point[0]
        row = gdf.iloc[id.id]
        # Ensure type_code is treated as a string
        type_code_str = str(row['type_code'])
        if regex.match(type_code_str):
            # Note: haversine() expects (lat, lon)
            dist = haversine(query_point, (row.geometry.y, row.geometry.x))
            if dist < min_dist:
                nearest_poi = row
                min_dist = dist

    return nearest_poi
```

- RangeScan and NNScan: Implements brute-force approaches for range and nearest neighbor queries, respectively, for comparison purposes.

```python
# 4. Brute-force range query function
def RangeScan(query_range, type_regex_str, file_path):
    poi_data = pd.read_csv(file_path)
    results = []
    regex = re.compile(type_regex_str)
    for _, row in poi_data.iterrows():
        if regex.match(row['type_code']):
            point = Point(row['wgs_lng'], row['wgs_lat'])
            if isinstance(query_range, tuple) and len(query_range) == 2:  # Rectangle range
                if box(query_range[0][0], query_range[0][1], query_range[1][0], query_range[1][1]).contains(point):
                    results.append(row)
            elif isinstance(query_range, tuple) and len(query_range) == 3:  # Circle range
                if point.distance(Point(query_range[0], query_range[1])) <= query_range[2]:
                    results.append(row)
    return pd.DataFrame(results)
```

```python
# 5. Brute-force nearest neighbor query function
def NNScan(query_point, type_regex_str, file_path):
    poi_data = pd.read_csv(file_path)
    nearest_poi = None
    min_dist = float('inf')
    regex = re.compile(type_regex_str)
    for _, row in poi_data.iterrows():
        if regex.match(row['type_code']):
            dist = Point(query_point[0], query_point[1]).distance(Point(row['wgs_lng'], row['wgs_lat']))
            if dist < min_dist:
                nearest_poi = row
                min_dist = dist
    return nearest_poi
```

Results

The implemented system was tested with specific queries:

- Nearest ATM Query: Searched for the nearest ATM to the Central Building of BIT, which resulted in the following output

Nearest ATM: 招商银行ＡＴＭ（魏公村路８号院东北）

```python
# Nearest ATM to the Central Building of BIT
nearest_atm = NNQuery((39.958, 116.311), '^1603', index, gdf)
if nearest_atm is not None:
    print("Nearest ATM:", nearest_atm['name'])
else:
    print("No ATM found")
```

```
Nearest ATM: 招商银行ＡＴＭ（魏公村路８号院东北）
```

- Range Query for Restaurants: Counted the number of restaurants within 500 meters of the south door of BIT, resulting in:

Number of restaurants within 500.0 meters: 36

```python
# Check for restaurants within 500 meters of a specific point
sample_point = (39.955, 116.310)  # Adjust as necessary
sample_radius = 0.5  # 500 meters
sample_restaurants = RangeQuery((sample_point[0], sample_point[1], sample_radius), '^5', index, gdf)
print(f"Number of restaurants within {sample_radius * 1000} meters: {len(sample_restaurants)}")
```

```
Number of restaurants within 500.0 meters: 36
```

- List of Restaurants: list the number of restaurants within 500 meters of the south door of BIT, resulting in:

	name	type_code	wgs_lat	wgs_lng
249	参差咖啡（北京魏公村店）	50500	39.955897	116.312532
70	大象空间	50500	39.955902	116.312352
28	桥咖啡	50500	39.957699	116.306435
181	贝果西饼店（韦伯豪家园西）	50800	39.953295	116.312585
259	稻香村	50800	39.953351	116.313283
65	大才子面馆（魏公村店）	50100	39.953606	116.312426
71	风波庄（魏公村分舵）	50100	39.953619	116.312465
260	咕咕派（北外店）	50000	39.953622	116.312385
247	六和烤鸡（魏公村店）	50000	39.953636	116.312961
180	东北骨头庄（韦伯豪家园西北）	50100	39.953643	116.313156
18	北京晋南建梅主食店	50000	39.953668	116.313368
33	浩日沁蒙古餐厅	50100	39.953719	116.312066
178	肥羊王（魏公村店）	50117	39.953719	116.312066
63	九亿（魏公村店）	50100	39.953846	116.313390
27	花舞陕一边	50115	39.955785	116.314059
254	乡村啤酒屋	50100	39.953008	116.310855
253	渝州家厨（魏公村店）	50102	39.953116	116.311681
248	阿曼尼萨汗美食城	50121	39.953130	116.311558
67	周黑鸭（中友大厦北）	50000	39.953135	116.311408
135	北京外国语大学学生食堂	50100	39.953233	116.309386
257	北京外国语大学教工餐厅	50100	39.953913	116.310195

136	北京外国语大学清真餐厅	50121	39.953943	116.310765
252	胶东海鲜大排档（魏公村店）	50119	39.954357	116.310987
8	老自行车咖啡馆	50500	39.954419	116.310982
73	江依林韩式快餐	50300	39.954467	116.311336
62	万记麻辣烫（魏公村店）	50100	39.954578	116.310908
194	一志日本料理（魏公村店）	50202	39.954673	116.310894
251	南门烤翅	50118	39.955509	116.309072
258	巫山烤全鱼	50118	39.955509	116.308983
195	麻里麻里香锅（魏公村店）	50117	39.955512	116.309156
130	金榜缘食府	50100	39.955878	116.307034
132	第七食堂	50100	39.955991	116.306872
200	Ｈｅｌｅｎ′ｓ	50500	39.956000	116.311081
184	富翔鸡煲	50100	39.956008	116.311210
183	小福乐菜馆	50111	39.956008	116.311166
41	Ａ８	50400	39.956014	116.311414

```
# Display some of the found restaurants, if any
if len(sample_restaurants) > 0:
    print(sample_restaurants[['name', 'type_code', 'wgs_lat', 'wgs_lng']])
```

```
              name  type_code    wgs_lat     wgs_lng
249    参差咖啡（北京魏公村店）      50500  39.955897  116.312532
70             大象空间      50500  39.955902  116.312352
28              桥咖啡      50500  39.957699  116.306435
181   贝果西饼店（韦伯豪家园西）      50800  39.953295  116.312585
259             稻香村      50800  39.953351  116.313283
65      大才子面馆（魏公村店）      50100  39.953606  116.312426
71       风波庄（魏公村分舵）      50100  39.953619  116.312465
260        咕咕派（北外店）      50000  39.953622  116.312385
247       六和烤鸡（魏公村店）      50000  39.953636  116.312961
180  东北骨头庄（韦伯豪家园西北）      50100  39.953643  116.313156
18        北京晋南建梅主食店      50000  39.953668  116.313368
33          浩日沁蒙古餐厅      50100  39.953719  116.312066
178        肥羊王（魏公村店）      50117  39.953719  116.312066
63         九亿（魏公村店）      50100  39.953846  116.313390
27            花舞陕一边      50115  39.955785  116.314059
254           乡村啤酒屋      50100  39.953008  116.310855
253      渝州家厨（魏公村店）      50102  39.953116  116.311681
248        阿曼尼萨汗美食城      50121  39.953130  116.311558
67       周黑鸭（中友大厦北）      50000  39.953135  116.311408
135     北京外国语大学学生食堂      50100  39.953233  116.309386
257     北京外国语大学教工餐厅      50100  39.953913  116.310195
136     北京外国语大学清真餐厅      50121  39.953943  116.310765
252   胶东海鲜大排档（魏公村店）      50119  39.954357  116.310987
8           老自行车咖啡馆      50500  39.954419  116.310982
73          江依林韩式快餐      50300  39.954467  116.311336
62      万记麻辣烫（魏公村店）      50100  39.954578  116.310908
194    一志日本料理（魏公村店）      50202  39.954673  116.310894
251            南门烤翅      50118  39.955509  116.309072
258           巫山烤全鱼      50118  39.955509  116.308983
195    麻里麻里香锅（魏公村店）      50117  39.955512  116.309156
130           金榜缘食府      50100  39.955878  116.307034
132            第七食堂      50100  39.955991  116.306872
200          Ｈｅｌｅｎ＇ｓ      50500  39.956000  116.311081
184            富翔鸡煲      50100  39.956008  116.311210
183           小福乐菜馆      50111  39.956008  116.311166
41               Ａ８      50400  39.956014  116.311414
```

These results demonstrate the system's ability to process spatial queries efficiently and accurately.

Conclusion

The developed Python-based system effectively utilizes spatial indexing and querying techniques to efficiently identify and analyze Points of Interest within geographic data. The integration of the Haversine formula enhances the accuracy of distance calculations, ensuring the results are practical for real-world applications.