COMP 4334 - Assignment 1

Michael Ghattas - May 17, 2025

Assignment Description

This assignment implements the **Grid-Based Close Pairs Problem** using **Spark RDDs** with optimized partitioning, error handling, and efficient distance calculation for spatial data. The goal is to identify pairs of geographic points within a specified distance threshold, leveraging Spark's distributed computation capabilities.

Algorithm Overview

1. Data Loading and Cleaning

- **File Paths:**
  - geoPoints0.csv
  - geoPoints1.csv
- **Header Removal:**
  - Skips the header line starting with **"PointID"** to avoid parsing errors.
- **Early Partitioning:**
  - **Repartitioning** is applied early to balance the load across workers, reducing data skew.

```
geoPoints_files = [
    "dbfs:/FileStore/shared_uploads/michael.ghattas@du.edu/geoPoints0.csv",
    "dbfs:/FileStore/shared_uploads/michael.ghattas@du.edu/geoPoints1.csv"
]

points_rdd = sc.union([
    sc.textFile(file)
    .filter(lambda line: line.strip() and not line.lower().startswith("pointid"))
    .map(parse_point)
    .filter(lambda point: point is not None)
    .repartition(128)  # Early partitioning to balance load
    for file in geoPoints_files
]).cache()
```

---

2. Point Parsing and Grid Cell Assignment

- **Point Parsing:**
  - Each line is parsed into a **(pointID, x, y)** tuple, with robust error handling for empty or malformed lines.

- **Grid Cell Calculation:**
  - Each point is assigned to a grid cell based on integer division.

```
def parse_point(line):
    try:
        line = line.strip()
        if not line or line.lower().startswith("pointid"):
            return None
        parts = line.split(",")
        if len(parts) != 3:
            return None
        point_id, x, y = parts
        try:
            x = float(x.strip())
            y = float(y.strip())
        except ValueError:
            return None
        point_id = point_id.strip()
        if not point_id:
            return None
        return (point_id, x, y)
    except Exception as e:
        return None
```

---

3. Neighbor Pushing and Grid Mapping

- **Efficient Neighbor Generation:**
  - Points are pushed to nearby cells within the distance threshold to capture boundary cases.

```
def get_grid_cell(x, y, cell_size):
    return (math.floor(x / cell_size), math.floor(y / cell_size))

def generate_relevant_neighbors(i, j, distance=0.75, cell_size=0.75):
    max_offset = math.floor(distance / cell_size)
    for dx in range(-max_offset, max_offset + 1):
        for dy in range(-max_offset, max_offset + 1):
            if dx == 0 and dy == 0:
                continue  # Skip the center cell itself
            if math.sqrt((dx * cell_size) ** 2 + (dy * cell_size) ** 2) <= distance:
                yield (i + dx, j + dy)

cells_rdd = points_rdd.flatMap(
```

```
    lambda p: [
        (cell, p) for cell in generate_relevant_neighbors(*get_grid_cell(p[1], p[2], 0.75))
    ]
).partitionBy(128).cache()
```

---

4. True Balanced Pair Extraction

- **Efficient Pair Checking:**
  - The **find_close_pairs()** function is optimized to avoid redundant distance calculations.
- **Consistent Pair Ordering:**
  - Pairs are consistently ordered to avoid duplicates.

```
def distance(p1, p2):
    x1, y1 = p1[1], p1[2]
    x2, y2 = p2[1], p2[2]
    return math.sqrt((x1 - x2) ** 2 + (y1 - y2) ** 2)

def find_close_pairs(points_list):
    n = len(points_list)
    pairs = set()
    for i in range(n):
        p1 = points_list[i]
        for j in range(i + 1, n):
            p2 = points_list[j]
            if len(p1) == 3 and len(p2) == 3 and distance(p1, p2) <= 0.75:
                pair = (p1[0], p2[0]) if p1[0] < p2[0] else (p2[0], p1[0])
                pairs.add(pair)
    return pairs

close_pairs_rdd = cells_rdd.partitionBy(128).groupByKey().flatMap(
    lambda cell_points: find_close_pairs(list(cell_points[1]))
).distinct().cache()
```

---

5. Collecting and Printing Results

```
close_pairs = close_pairs_rdd.collect()
print(f"Dist:0.75")
print(f"{len(close_pairs)} {close_pairs}")
```

Sample Output:

```
Dist:0.75
4 [('Pt06', 'Pt09'), ('Pt05', 'Pt07'), ('Pt03', 'Pt10'), ('Pt01', 'Pt15')]
```
