COMP7107 Management of complex data types
Assignment 1
Spatial Aggregate Queries
Due Date: October 4, 2024, 5:00pm

Summary

The goal of this assignment is the implementation and testing of spatial indexing techniques and spatial query aggregation algorithms. For this assignment, you are going to use yellow-cab trip data from NYC.

Navigate to:
https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page
Click on 2009 and then click on **February**

- **Yellow Taxi Trip Records**

to download the required data for this assignment.

The file includes information about all yellow taxi trips that took place in February 2009 in New York.

Data Loading and preparation

Note that the downloaded file is in parquet format (a compressed text format). To load parquet files into your program you can use the corresponding libraries of the programming language that you will use. For example, the Pandas read_parquet() method can be used to load the data into a Pandas data frame in Python. If you prefer not to use Pandas, you can export the data to a CSV file using to_csv() method of data frames and then load the data from the CSV file.

Example of Python script to convert your data to CSV:

```
import pandas as pd
df = pd.read_parquet('yellow_tripdata_2009-02.parquet')
df.to_csv('yellow_tripdata_2009-02.csv')
```

The first line of the uncompressed file specifies the attributes of each taxi trip:
vendor_name,Trip_Pickup_DateTime,Trip_Dropoff_DateTime,Passenger_Count,Trip_Distance,Start_Lon,Start_Lat,Rate_Code,store_and_forward,End_Lon,End_Lat,Payment_Type,Fare_Amt,surcharge,mta_tax,Tip_Amt,Tolls_Amt,Total_Amt

In each of the following lines you will find the attribute values of each taxi trip. Each line starts with a taxi trip number (0, 1, 2, etc.) followed by the attribute values of that trip.

The attributes that we are interested in are:
Trip_Pickup_DateTime = date and time when the taxi was picked up
Start_Lon = longitude (that is the same as x-coordinate) of the location where the taxi was picked up
Start_Lat = latitude (that is the same as y-coordinate) of the location where the taxi was picked up
Total_Amt = total amount paid by customer(s) for this trip

Task 1: Data preprocessing and indexing

For the first task you are asked to write a program, which organizes the data that we are interested in (the four attributes only for each trip) into a 3-dimensional grid data structure (index), which can be used for query evaluation. The first dimension of the grid is longitude (x-coordinate), the second dimension is latitude (y-coordinate) and the third dimension is datetime. To construct the grid, you will have to load the data and assign each trip to a grid cell. In the end you will have to produce a text file with filename grid.txt, which includes all information about the grid. This file can then be used by a program (task 2) to load all information about the grid into memory and to be able to evaluate spatial aggregate queries using the grid.

Step 1.1: compute min/max values.

The first step is to compute the minimum and maximum value for each of the three dimensions. For the longitude and latitude dimensions (x,y) this is easy because each value of longitude and latitude is a real number. So, you will have to compute the minimum x-dimension of any record in the file, the maximum x-dimension in the file, minimum y-dimension in the file, the maximum y-dimension in the file. This is a relatively easy task. However, finding the minimum and maximum datetime, as well as dividing the time dimension into ranges to be used by the grid can be challenging because each datetime is specified by a string such as "2009-02-03 08:25:00". Your goal is to convert each of these strings to a **timestamp**, which represents the number of seconds passed from the **epoch datetime** "1970-01-01 00:00:00". This conversion is supported by libraries in programming languages; for example, the following Python script converts a datetime string to the corresponding timestamp.

```
from datetime import datetime
datetime_str = "2009-02-03 08:25:00"
datetime_object = datetime.strptime(datetime_str, "%Y-%m-%d %H:%M:%S")
print(datetime.timestamp(datetime_object))
```

After the conversion of datetimes to timestamps, you can compute the minimum and maximum timestamps in your data, as you do for the minimum/maximum time dimensions. Each trip record is described by four values (x,y,timestamp,Total_Amt).

Step 1.2: set the grid ranges.

Define a 100*100*100 grid by dividing each min-max range in each of the three dimensions to 100 partitions. For example, for the x-dimension, all trip records for which the x-value is from min_x to min_x+(max_x-min_x)/100 goes to partition 0, all trip records for which the x-value is from min_x+(max_x-min_x)/100 to min_x+2*(max_x-min_x)/100 goes to partition 1, etc. A given trip record goes to exactly one grid cell described by three partitions in each of the three dimensions (x, y, time). For example, grid cell (0,0,1) contains all trip records for which the x-value is from min_x to min_x+(max_x-min_x)/100, the y-value is from min_x to min_x+(max_x-min_x)/100, and the timestamp is from min_t+(max_t-min_t)/100 to min_t+2*(max_t-min_t)/100.

Note that the upper bounds of cells are open intervals and not closed ones. Therefore, for example, for the x-dimension, when you compute the partition, you may need to add a very small number (e.g., 0.000001) to (max_x-min_x). For example, in Python, you may determine the partition for an x-value as follows:

```
int(100 * ((x-min_x) / (max_x-min_x+0.000001))
```

, where x is the value in the x-dimension of an object. This is done for avoiding having an object falling outside the grid if it has the maximum value in one dimension.

Step 1.3: assign the records to cells.

Construct a data structure in memory, where for each grid cell you allocate all the trip records in that cell. After assigning all records to the grid cells, you should create file grid.txt and write for each non-empty cell its contents. The file should be of the following **format**:

min_x, max_x
min_y, max_y
min_t, max_t
(0,0,0)
Total_Amt_in_cell
x,y,timestamp,Total_Amt

x,y,timestamp,Total_Amt
x,y,timestamp,Total_Amt
…
(0,0,1)
Total_Amt_in_cell
x,y,timestamp,Total_Amt
x,y,timestamp,Total_Amt
x,y,timestamp,Total_Amt
…
…

Note that empty cells (those with no records in them) should not be printed to the output file. Note also that for each non-empty cell, the first line after the cell coordinates is the total amount for all trips in that cell.

Task 2: Query evaluation

Write a program that first reads grid.txt and loads all its data in memory in an in-memory data structure. Then, the program reads file queries.txt with range queries of the form
low_x, up_x, low_y, up_y, low_datetime, up_datetime

Each line of the file corresponds to a query that asks the total amount of all trips within the rectangle area described by low_x, up_x, low_y, up_y, that took place within the low_datetime, up_datetime time range; the low_datetime, up_datetime range is described in string format (as in the original data read from the file in task 2, e.g., 2009-02-03 08:25:00).

The first four lines of queries.txt:
-102,-79.934,38,56.57,2009-02-25 08:38:24,2009-02-27 11:25:56
-85.8,-82.1,-39.1,44.4,2009-02-14 07:59:52,2009-02-15 22:32:16
-78.7,-73.969,31.808,40.761,2009-02-14 02:57:22,2009-02-27 11:08:06
-73.9983,-73.994,40.7356,40.745,2009-02-01 02:53:37,2009-02-03 23:03:00

Each the datetime low/up bounds in each query should be converted to timestamp low/up bounds before query processing.

Step 2.1: Exact query processing

Write a function, which computes the exact result for each query. Your function should identify the grid cells which include the query results and access the data from these cells only. If a cell is included in the query range, then you should directly use the Total_Amt_in_cell for that cell. If a cell is partially included in the query, you should read the trips in them one-by-one, verify the query range for each trip and if the trip is inside the range, add its Total_Amt to the query result.

Step 2.2: Approximate query processing

Write a function, which computes the approximate result for each query. Your function should identify the grid cells which include the query results and access these cells only. If a cell is included in the query range, then you should add the Total_Amt_in_cell for that cell to the query result. If a cell is partially included in the query, you should compute the **fraction** f of space in the cell inside the query and then add f* Total_Amt_in_cell to the query result.

Program output

Your program should evaluate all queries and generate an output file, where for each query you show the result. Your program should take a command line argument <exact> which is 1 or 0. If the argument is 1 then you should run the exact processing algorithm, if the argument is 0 then you should run the approximate algorithm. Moreover, include the contents of your output file for both <exact> values (0 and 1) in your report.

Time comparison

Report the time needed from the start of processing the first query to the end of processing the last query. Run your program for exact processing and for approximate processing and compare the runtimes. You should observe that approximate processing is faster compared to exact processing.

Deliverables: You should submit your program(s) and a single PDF file which **documents** the programs, reports your **results** and observations, and includes any special instructions for compiling and running your programs. Your programs should run in **reasonable** time. Please submit a single **ZIP** file with all requested programs and documents to Moodle on or before 5:00pm, October 4th, 2024. Make sure all contents are readable. **Please do not submit any data files**. Please feel free to post your questions on **Moodle forum** or contact the TA of the course if you encounter any difficulty in this assignment. We would be happy to help.