Indexing and Query of check-ins in the New York City

Lok Yee Joey CHEUNG

I. INTRODUCTION

In this technological era, location-based services have been prominent. Enormous amount of spatio-temporal data has been collected which provides insightful trends and patterns. However, manual searching and organization of spatial data objects could be problematic given such huge volum, velocity and variety of sources. Effective search and information retrieval of objects in multi-dimensions are vital.

Indexing algorithms are powerful means to build data structures to speed up searching, reduce computational resources and improve accuracy. In this paper, six indexing algorithms for high-dimensional data which are Quad tree, K-d tree and R-tree, Geo-hash, Grid-based, Linear scan were introduced. Query tasks ranging from range query, distance query to nearest neighbour query etc. based on the check-in data points in New York City were designed and efficiently solved with the algorithms.

II. METHODOLOGY

Six effective indexing algorithms are introduced. To examine their effectiveness in searching and information retrieval, each of them are tested on queries based on real world scenarios.

A. Algorithms

1) Quad Tree

Quad tree is a tree data structure used for spatial indexing and searching of two-dimensional data, point quad tree is an example. It sub-divides the space into four quadrants in Z-order recursively depending on the insertion order and distribution of points, until termination is reached. Quad tree is commonly used in image processing and spatial indexing.

Point quad tree construction is described in detail by the following flow-chart:

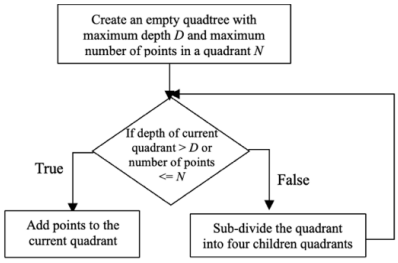


Figure 1. Flow chart of Quad Tree

To perform searching, we start with the root and confirm if the query region intersects the bounding box of current node. If it does, check the data points in the node as well as its child nodes until leaf nodes is reached.

2) K-d Tree

K-d tree is a binary search tree which recursively divide k-dimensional space into two partitions. It is widely used for searching namely range and nearest-neighbour search.

K-d tree is constructed as follows (Skrodzki, 2013):

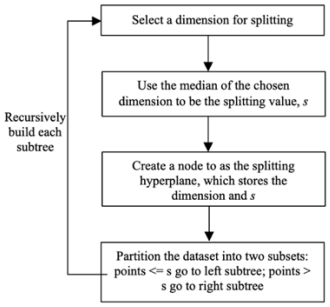


Figure 2. Flow chart of K-d Tree

During searching, the process is similar to binary search tree but only one dimension is compared at each level of the tree starting from the root. If the query point is smaller, go to left sub-tree, go to the right otherwise. Check the data points in the leaf node when it is reached.

3) R-Tree

R-tree is a height-balanced tree structure which organizes points in a multi-dimensional space (Guttman, 1984). It is similar to B-tree due to the fact that its leaf nodes contain records pointing to data objects. As for the non-leaf nodes, each of them records a minimum bounding rectangle (*MBR*). Below shows the procedures of constructing a R-tree:

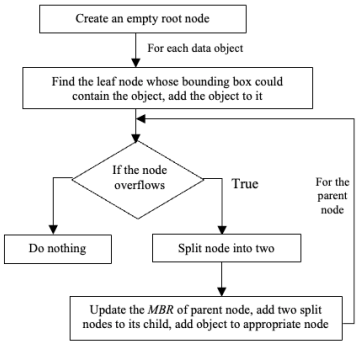


Figure 3. Flow chart of R-tree

During insertion of points, *S* is the whole set of points. When dividing *S* into two sets of data points *S1, S2*, the perimeter of *MBR(S1)* and *MBR(S2)* is minimized (Huang, 2023). The query searching process starts from the root and it will be checked if the query region intersects the MBR of the

current node. If there is, search along the child nodes to see if there is any more intersections.

4) Geo-hash

The Earth's surface is divided into rectangular regions by Geohash. It is a base 32 spatial indexing algorithm that assigns unique value to each region via encoding latitude and longitude of the area. It is an implementation of Z-order curve which maps two-dimension data to one dimension. To perform spatial query, the proximity of the prefix of geohash string is calculated. To compute geo-hash string for each location, the geo-hash algorithm is used as below (Geohash, 2023):

1. Determine the length, *n* of the geohash string
2. Encode the location coordinates into binary representations
3. Divide the latitude and longitude range into intervals respectively
4. Iterate through the intervals for *n* times and append 1 to geohash if specified location lies between the interval
5. Translate the bits into integers and encode them into base 32 alphabets

5) Grid-based indexing

Grid-based indexing is a spatial indexing method which partition spatial data into grids. Equal sizes of cells are formed in each space and they contain data points of the corresponding latitude and longitude. Unique index is assigned to each grid for efficient indexing. To construct index:

Algorithm 1: Grid-based indexing
Input: *location dataset, grid cell size*
Output: *Index*
index = {}
For each item in location dataset **do**
  cell_x = latitude of item / grid cell size
  cell_y = longitude of item / grid cell size
  if (cell_x, cell_y) is **not in** index **then**
    (cell_x, cell_y) as key with an empty list
  append item to key (cell_x, cell_y)
return index

Figure 4. Pseudocode of grid-based indexing

6) Linear Scan

Linear scanning is a basic indexing algorithm which search through a list of data linearly. By examining each item, matched ones are selected.

B. Task queries

Five query tasks from the real world were proposed:

1) Find all check-ins in a specific venue category within a rectangular area during a time window

Major example: Find all check-ins in bars within Brooklyn during New Year's Eve and New Year's Day in 2012-13

To solve query (1), R-tree, Quad tree, K-d tree were performed. For each algorithm, index was built by inserting bounding rectangle of all check-in objects. We then performed spatial and temporal search by selecting all data points whose *MBR* intersected with the *MBR* that encompassed the Borough. Lastly, timestamp and venues were checked for each selected points to filter out the data not in the specified time.

2) Find all check-ins within a maximum distance to a location in a particular time window

Major example: Find all check-ins within 5 km to Museum of the City of New York in the afternoon of Independence Day (4th July 2012)

To solve this distance query, we have used R-tree, Quad tree and grid-based indexing. For each method, index was built similarly as the previous tasks. Spatial search was performed by selecting all data points whose *MBR* intersected with the location within the time-window. Finally, only points with distance no larger than the maximum distance were selected.

3) Find K-nearest neighbours of the given location within a specified timestamp window

Major example: Find 5 nearest neighbours (5-nn) of the given check-in at [40.78430372, -73.97968281] within the same check-in day which is 4th of April 2012

To solve the nearest neighbour query, k-d tree, R-tree, geohash was used. For the first two, we built the index by inserting all check-in objects' locations on the given date and queried the tree to find the k nearest neighbours. Euclidean distance was used for k-d tree while MBR distance were measured for R-tree. As for geohash, specified location is hashed to a string and similarity of neighbours were compared using geo-hash values.

4) Find the number of data points in a rectangular area in grouped time windows

Major example: Find the number of people travelling from John F. Kennedy International Airport in each season in NYC during 2012-13

To solve query (4), we deployed quad tree, R-tree and linear scan. Index was built and queried similarly as task 1. Last, data points were grouped by seasons based on their time stamps. Linear scanning was performed intuitively.

5) Find top k types of locations with highest number of data points in a specified time window

Major example: Find top 5 most popular venues that people love to go to during Thanksgiving in Manhattan in 2012-13

To solve query (5), quad tree, R-tree and linear scan were implemented. Index was built and queried similarly as task 1. Finally, we grouped the data based on the types of locations and performed ranking. Linear scanning was performed intuitively.

More test examples for each query are shown in attached code file and in *Appendix B*.

III. EXPERIMENTS

A. Data source

The dataset used in the experiment is the user activity of citizens in the New York City (NYC) during 2012 and 2013. The data are the check-ins collected from FourSquare. Each check-in is associated with the information of the user, venue, venue's category, GPS coordinates and time stamp (chetan, 2017).

Since the data is stored in a csv file, Python was used to extract data from the file and implement the algorithms in Visual Studio Code. Additionally, SQL was used to construct database in PostgreSQL and validate the implementation.

B. Implementation

After defining the data source and tools needed, the *'pandas'* library was selected for extracting data from csv file into data frame to facilitate cleaning and analysis. For example, timestamp was converted to datetime for easier comparison in each task. Below illustration are based on the major example created for each query.

i) Task 1

R-tree: The '*rtree'* packages was imported and class *rtee.index.Index() was* implemented to create the index. First, all check-in data points was passed into *rtree.index.Index.insert()* function for index insertion. After the insertion, bounding box of Borough was also passed into *rtee.index.Index.intersection()* function in the form of *(min_lat, min_long, max_lat, max_long)*[1] for spatial query.

Quad tree*:* We applied the '*pyqtree*' library. Bounding box of New York was inputted into class *Index()* to set spatial extent of the index. *Index.insert()* and *Index.intersection()* performed similarly as *'rtree'* packages for insertion of all data points in NYC and spatial query execution respectively.

K-d tree*:* Library '*scipy.spatial'* was imported. Center points of Brooklyn was found with given bounding box. After index was constructed using *KDTree()*, we implemented *KDTree.query_ball_point()* to find all points within Brooklyn with radius of 0.1 degree.

For each of the above, the last step was to filter based on timestamps and venues after data points were retrieved from spatial query.

ii) Task 2

R-tree, Quad tree*:* Index construction was same as task 1. Great-circle distance in the *'geopy'* library was helpful for the computation of the shortest distance between two points on a sphere's surface for setting the distance constraint. After the data points within 5 km from the Museum were retrieved, results were obtained by filtering out data points not on 4th July.

Grid-based: Grid size was set as 0.1 and check-in points were inserted into associated grid cells by dividing each data's latitude and longitude with grid size respectively. We performed query by iterating over neighbour cells and retrieved data that satisfy the distance and time constraints.

iii) Task 3

K-d tree: '*scipy.spatial'* library was imported. Time window was first filtered before creating a K-d tree object using *KDTree()*. We then used *query()* method to query the tree for 5 nearest neighbours based on the location provided.

R-tree: Insertion of index was same as above. We then perform searching using *rtee.index.Index.nearest()* by inputting location and number of nearest neighbours.

Geo-hash: We used geohash.encode() from library *'geohash'* to encode specified location into binary representation. After prefix length was set, we filtered by matching geohash strings with the same prefix as the given location. *geodesic()* from *'geopy.distance'* was used for the calculation of distances and retrieve nearest neighbours.

iv) Task 4 and 5

Quad tree: The *'csv'* library was also used for reading and processing csv file as a dictionary using *csv.DictReader()*. Building of index and execution of query were same as Task 1 with different input parameters.

R-tree: Building of index and execution of query were same as Task 1 with different input parameters.

Linear scan*:* By iterating through items, data within the specified bounding box were selected.

For each of the above, the last step was to filter and count based on seasons for task 4, time and venues for task 5 were done afterwards.

C. Results

To better visualize the results, geometry is added to the database as an individual column. Tools in pgAdmin 4 namely 'Geometry Viewer' and 'Graph Visualizer' were used for visualizing the results in a user-friendly way (See *Appendix A* for detailed algorithms' results). Results of query tasks are illustrated as follows:

Query task (1):

Find all check-ins in bars within Brooklyn during New Year's Eve and New Year's Day in 2012-13

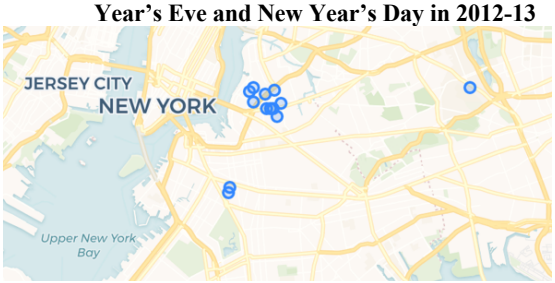


Figure 5. Map visualization of Query Task (1)

Query task (2):

All check-ins within 5 km to Museum of the City of New York in the afternoon of Independence Day

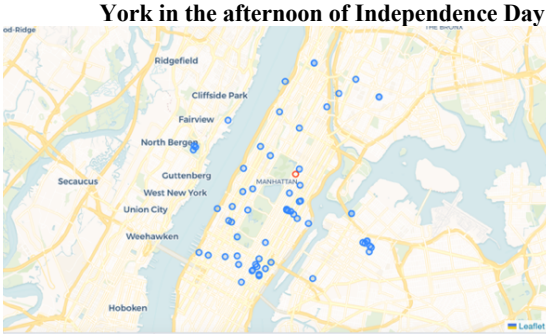


Figure 6. Map visualization of Query Task (2)

Query task (3):

5 nearest neighbours of the red spot on 4/4/2012

5 nearest neighbour using K-d tree				
	userID	**Latitude**	**Longitude**	**Distance (m)**
1	260	40.784537	-73.979763	2.753
2	1066	40.781680	-73.979321	29.504
3	804	40.780589	-73.980739	43.016
4	983	40.780335	-73.981364	48.014
5	1	40.781558	-73.975792	53.045

Figure 7. Table visualization of Query Task (3)

5 nearest neighbour using SQL				
	userID	**Latitude**	**Longitude**	**Distance (m)**
1	260	40.784537	-73.979763	11.458
2	1066	40.781680	-73.979321	90.000
3	804	40.780589	-73.980739	163.650
4	983	40.780335	-73.981364	223.149
5	415	40.778852	-73.981225	239.616

[1] 'min_lat', 'max_lat' refers to the minimum and maximum latitude of the rectangular area respectively. 'min_long', 'max_long' refers to the minimum and maximum longitude respectively.

Figure 8. Table visualization of Query Task (3)

Query task (4):

Number of people travelling from John F. Kennedy International Airport in each season in 2012-13

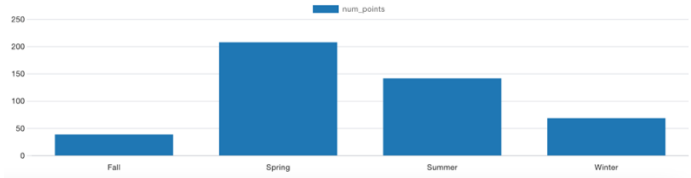


Figure 9. Bar chart visualization of Query Task (4)

Query task (5):

Top 5 most popular venues that people love to go to during Thanksgiving in Manhattan in 2012-13

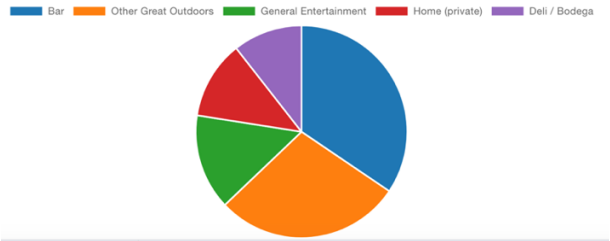


Figure 10. Pie visualization of Query Task (5)

D. Evaluation and Analysis

i) Results analysis

In Figure 5, check-ins in bars during New Year in Brooklyn were concentrated in Williamsburg. The result is reasonable as Williamsburg has been well-known for its nightlife.

In Figure 6, plenty of check-ins (blue spots) within 5 km to the given Museum (red spot) were presented in the map. Most were distributed in Manhattan.

In Figure 7 and 8, the 5-nn returned by K-d tree were slightly different from that of Postgres, as shown in the rows marked in orange. K-d tree used Euclidean distance for nearest neighbour search while Postgres calculated the spherical distance on the Earth's surface using *ST_DistanceSphere()* function. This led to the difference in the 5th neighbour resulted. Distances columns were also different due to distinct computation methods and assumptions.

Figure 9 shows spring is the peak season for travel which doubled the total counts of Winter and Fall. Factors like pleasant weather and spring break might explain the finding.

As for Figure 10, 'Bar' and 'outdoors' are the most popular during holidays for New York citizens.

ii) Validation of returned results

A database containing check-in data was created in Postgres by using *'SQLAlchemy'* tookit in Python. We then wrote SQL code for the proposed tasks, regarded as the ground truth. To measure the correctness of the returned results, metrics like precision, recall and F1-score were calculated, refer to *Appendix B* for the screenshots of different cases.

Precision, recall and F-score of all query tasks except task 3 achieved 1.0, which indicated that all the results accurately matched with their ground truths. Since some results included strings, manual check was performed e.g. 'venue' column in resulted data frame of task 5. It is observed that task 3's scores were 0.8 in all 3 metrics in K-d tree and R-tree. As explained in (i) above, K-d tree used Euclidean distance while R-tree used MBR distance to measure the extent of overlapping of bounding boxes which led to different results from Postgres using spherical distance. 0.4 was resulted from using Geo-hash since no distance metrics was used, spatial proximity was considered instead.

iii) Evaluation metrics

During building the index and executing query, runtime cost, memory cost and I/O cost were computed with the use of '*Datetime'* module and '*psutil'* library (see *Appendix C for* detailed metrics table).

As shown in the evaluation metrics tables, it is obvious that the time cost of constructing a R-tree in most range and distance queries was significantly higher than that of the other algorithms (over 9) while having an outstanding performance in fast query execution. This may be due to large amounts of overlapping of bounding rectangles in the dataset. Node may be split and re-organized continuously in R-tree which leads to high overhead and time cost.

When building index, quad-tree took up relatively high space complexity. This may be attributed to recursive spatial sub-division which increases memory usage and disk access. On the contrary, it had fast query ability and close to 0 I/O cost during execution. This may stem from the elimination of quadrants that do not intersect the search area during range search.

It is discovered that the selected three algorithms performed fast and occupy less space for nearest neighbour search. However, accuracy might not be assured.

Surprisingly, grid-based indexing has occupied the least space during the whole process in distance query. It may not require overhead cost to maintain tree-like structure compared to others. Lastly, linear scan's performance could differ a lot with the change of data size and type of query which may not be desirable for big data size query.

IV. CONCLUSION

This report has illustrated that six indexing algorithms which have efficiently solved various queries with high speed and accuracy. Evaluation metrics and validation using SQL played a vital role in assessing the algorithms' performance. It is revealed that that R-tree and Quad tree excelled in executing range and distance query but are relatively expensive in index construction compared to K-d tree and grid-based indexing. Geo-hash and grid-based indexing are capable of searching with less special cost due to its non-tree structured nature. During selection of algorithms, it is crucial to take into consideration their distance methodologies which may lead to distinct results and affect accuracy.

Further research could be done on more complex datasets and optimizations problems. Not only does this project provide comprehensive analysis of indexing algorithms, it also contributes to valuable visions in the enhancement of indexing in the future.

REFERENCES

chetan. (2017). *FourSquare - NYC and Tokyo Check-ins*. Retrieved from Kaggle: https://www.kaggle.com/datasets/chetanism/foursquare-nyc-and-tokyo-checkin-dataset

Skrodzki, M. (2013). *The k-d tree data structure and a proof for neighborhood computation in expected logarithmic time.*

Guttman, A. (1984). *R-TREES. A DYNAMIC INDEX STRUCTURE FOR SPATIAL SEARCHING* . Berkeley: Association for Computer Machinery.

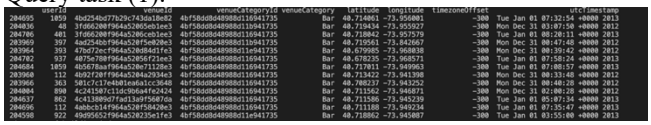
Huang, H. (2023). *INFS4205/7205 Advanced Techniques for High Dimensional Data Lec 4. Spatial Data Indexing 2.* Queensland.

Geohash. (2023, May 2). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Geohash

APPENDIX A

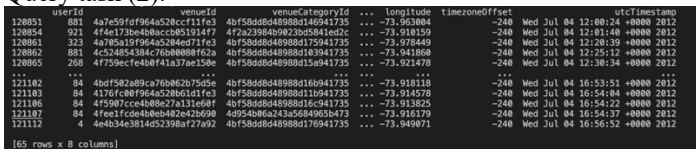
Screenshot of algorithms' results (Major Example)*:*

Query task (1):



	userID	venueId	venueCategoryId	venueCategory	latitude	longitude	timezoneOffset	utcTimestamp
204695	1059	4bd254bd77b29c743da18e82	4bf58dd8d48988d116941735	Bar	40.714061	-73.956001	-300	Tue Jan 01 07:32:54 +0000 2013
204036	40	3fd66200f964a52065eb1ee3	4bf58dd8d48988d116941735	Bar	40.715434	-73.955927	-300	Mon Dec 31 03:07:50 +0000 2012
204706	401	3fd66200f964a5206ceb1ee3	4bf58dd8d48988d116941735	Bar	40.718042	-73.957579	-300	Tue Jan 01 08:20:11 +0000 2013
203969	397	4ad2545bf964a520e8d41fe3	4bf58dd8d48988d116941735	Bar	40.719561	-73.842667	-300	Mon Dec 31 00:47:48 +0000 2012
203964	393	47bd72ecf964a520f84d1fe3	4bf58dd8d48988d1e941735	Bar	40.679985	-73.968838	-300	Mon Dec 31 00:39:42 +0000 2012
204702	937	4075e780f964a52058f21ee3	4bf58dd8d48988d116941735	Bar	40.678235	-73.968571	-300	Tue Jan 01 07:58:24 +0000 2013
204684	1059	4b5678acf964a520e7122de3	4bf58dd8d48988d116941735	Bar	40.717911	-73.949963	-300	Tue Jan 01 07:08:57 +0000 2013
203960	112	4b9272ff964a5204a2934e3	4bf58dd8d48988d116941735	Bar	40.713422	-73.941398	-300	Mon Dec 31 00:33:48 +0000 2012
203966	363	501c7c174f401e6a1cc3640	4bf58dd8d48988d116941735	Bar	40.708237	-73.943252	-300	Mon Dec 31 00:40:28 +0000 2012
204004	890	4c241507c11dc9b6a4fe2424	4bf58dd8d48988d116941735	Bar	40.711562	-73.946871	-300	Mon Dec 31 02:00:28 +0000 2012
204637	862	4c413809d7fad13a97507d0a	4bf58dd8d48988d116941735	Bar	40.711586	-73.945239	-300	Tue Jan 01 05:07:34 +0000 2013
204698	112	4a8bcb14f964a520f5842de3	4bf58dd8d48988d116941735	Bar	40.711188	-73.949234	-300	Tue Jan 01 07:35:47 +0000 2013
204598	922	49d95652f964a520235e1fe3	4bf58dd8d48988d11e941735	Bar	40.710862	-73.945087	-300	Tue Jan 01 03:55:00 +0000 2013

Query task (2):



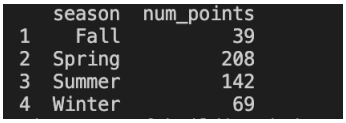
	userId	venueId	venueCategoryId	...	longitude	timezoneOffset	utcTimestamp
120851	881	4a7e59fdf964a520ccf11fe3	4bf58dd8d48988d146941735	...	-73.963004	-240	Wed Jul 04 12:00:24 +0000 2012
120854	921	4f4e173be4b0acc0b5934f7	4f2a23984b9023bd5841ed2c	...	-73.918159	-240	Wed Jul 04 12:01:40 +0000 2012
120861	323	4a705a19f964a5204ed71fe3	4bf58dd8d48988d175941735	...	-73.978449	-240	Wed Jul 04 12:20:39 +0000 2012
120862	881	4c524854384c76b00080f62a	4bf58dd8d48988d103941735	...	-73.941860	-240	Wed Jul 04 12:25:12 +0000 2012
120865	268	4f759ecfe4b0f41a37ae150e	4bf58dd8d48988d15a941735	...	-73.921478	-240	Wed Jul 04 12:30:34 +0000 2012
...	...	...	...	...	...	...	...
121102	84	4bdf502a89ca76b862b75d5e	4bf58dd8d48988d16b941735	...	-73.918118	-240	Wed Jul 04 16:53:51 +0000 2012
121103	84	4176fc00f964a520b61d1fe3	4bf58dd8d48988d11b941735	...	-73.914578	-240	Wed Jul 04 16:54:04 +0000 2012
121106	84	4f590fcce4b08e27a131e60f	4bf58dd8d48988d16c941735	...	-73.913825	-240	Wed Jul 04 16:54:22 +0000 2012
121107	84	4fee1fcde4b08eb482e62b698	4d954b06a243a5684965b473	...	-73.916179	-240	Wed Jul 04 16:54:37 +0000 2012
121112	4	4e4b34e3814d52398af27a92	4bf58dd8d48988d176941735	...	-73.949071	-240	Wed Jul 04 16:56:52 +0000 2012

[65 rows x 8 columns]

Query task (3):

	userId	venueId	venueCategoryId	venueCategory	latitude	longitude	timezoneOffset	utcTimestamp
1254	260	4a9787c3f964a52065292b0e	4bf58dd8d48988d176941735	Gym / Fitness Center	40.784537	-73.979763	-240	Wed Apr 04 10:29:36 +0000 2012
1397	1060	4e734ad0980f94a582a5f5c	4bf58dd8d48988d176941735	Gym / Fitness Center	40.781680	-73.979321	-240	Wed Apr 04 11:16:38 +0000 2012
830	804	3fd66200f964a5202e01ee3	503279209104c4b30a586d5c	Concert Hall	40.780589	-73.980739	-240	Wed Apr 04 08:51:43 +0000 2012
1165	983	4ae2iadef964a520c48a21e3	4bf58dd8d48988d119951735	Food & Drink Shop	40.780335	-73.981364	-240	Wed Apr 04 09:57:27 +0000 2012
2454	1	4abc1f51f964a52079862be3	4bf58dd8d48988d1ce941735	Seafood Restaurant	40.781558	-73.975792	-240	Wed Apr 04 21:31:31 +0000 2012

Query task (4):



	season	num_points
1	Fall	39
2	Spring	208
3	Summer	142
4	Winter	69

Query task (5):

	venue	num_points
1	Bar	52
2	Other Great Outdoors	43
3	General Entertainment	22
4	Home (private)	18
5	Deli / Bodega	16

APPENDIX B

Tested cases for each query and screenshot of correctness:

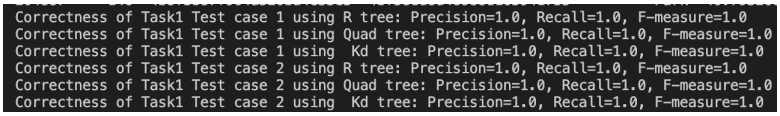
Task 1

Test Case 1 (Major Example):

Find all check-ins in bars within Brooklyn during New Year's Eve and New Year's Day in 2012-13

Test Case 2:

Find all check-ins in parks within Manhattan during the first two days of 2013



```
Correctness of Task1 Test case 1 using R tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task1 Test case 1 using Quad tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task1 Test case 1 using  Kd tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task1 Test case 2 using R tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task1 Test case 2 using Quad tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task1 Test case 2 using  Kd tree: Precision=1.0, Recall=1.0, F-measure=1.0
```

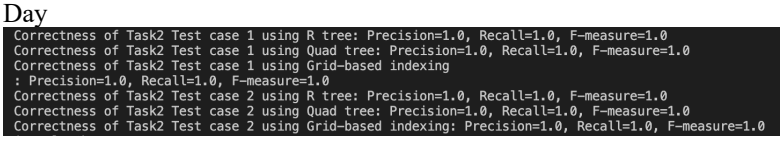
Task 2

Test Case 1 (Major Example):

Find all check-ins within 5 km to Museum of the City of New York in the afternoon of Independence Day

Test Case 2:

Find all check-ins within 3 km to Central Park in Valentines' Day



```
Correctness of Task2 Test case 1 using R tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task2 Test case 1 using Quad tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task2 Test case 1 using Grid-based indexing
: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task2 Test case 2 using R tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task2 Test case 2 using Quad tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task2 Test case 2 using Grid-based indexing: Precision=1.0, Recall=1.0, F-measure=1.0
```

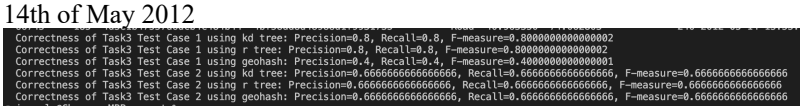
Task 3

Test Case 1 (Major Example):

Find 5 nearest neighbours (5-nn) of the given check-in at [40.78430372, -73.97968281] within the same check-in day which is 4th of April 2012

Test Case 2:

Find 3 nearest neighbours (3-nn) of the given check-in at [40.96537, -74.06281] within the same check-in day which is 14th of May 2012



```
Correctness of Task3 Test Case 1 using kd tree: Precision=0.8, Recall=0.8, F-measure=0.8000000000000002
Correctness of Task3 Test Case 1 using r tree: Precision=0.8, Recall=0.8, F-measure=0.8000000000000002
Correctness of Task3 Test Case 1 using geohash: Precision=0.4, Recall=0.4, F-measure=0.4000000000000001
Correctness of Task3 Test Case 2 using kd tree: Precision=0.6666666666666666, Recall=0.6666666666666666, F-measure=0.6666666666666666
Correctness of Task3 Test Case 2 using r tree: Precision=0.6666666666666666, Recall=0.6666666666666666, F-measure=0.6666666666666666
Correctness of Task3 Test Case 2 using geohash: Precision=0.6666666666666666, Recall=0.6666666666666666, F-measure=0.6666666666666666
```

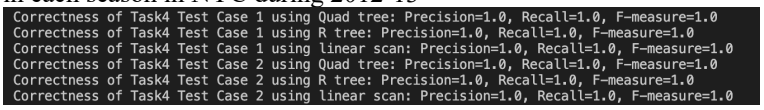
Task 4

Test Case 1 (Major Example):

Find the number of people travelling from John F. Kennedy International Airport in each season in NYC during 2012-13

Test Case 2:

Find the number of people travelling from LaGuardia Airport in each season in NYC during 2012-13



```
Correctness of Task4 Test Case 1 using Quad tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task4 Test Case 1 using R tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task4 Test Case 1 using linear scan: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task4 Test Case 2 using Quad tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task4 Test Case 2 using R tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task4 Test Case 2 using linear scan: Precision=1.0, Recall=1.0, F-measure=1.0
```

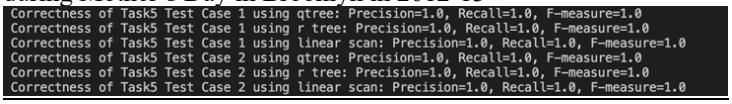
Task 5

Test Case 1 (Major Example):

Find top 5 most popular venues that people love to go to during Thanksgiving in Manhattan in 2012-13

Test Case 2:

Find top 3 most popular venues that people love to go to during Mother's Day in Brooklyn in 2012-13



```
Correctness of Task5 Test Case 1 using qtree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task5 Test Case 1 using r tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task5 Test Case 1 using linear scan: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task5 Test Case 2 using qtree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task5 Test Case 2 using r tree: Precision=1.0, Recall=1.0, F-measure=1.0
Correctness of Task5 Test Case 2 using linear scan: Precision=1.0, Recall=1.0, F-measure=1.0
```

APPENDIX C

Evaluation metrics (Major example):

Task 1

Query Task (1) using R-tree		
	Building the index	Executing the query
Time cost	9.751	0.118
Memory cost	27901952	4882432
I/O cost	46.78 MB	0.35 MB

Query Task (1) using Quad tree		
	Building the index	Executing the query
Time cost	1.020	0.133
Memory cost	47497216	3670016
I/O cost	0.10 MB	0.00 MB

Query Task (1) using K-d tree		
	Building the index	Executing the query
Time cost	0.050	0.263
Memory cost	7258112	245768
I/O cost	0.19 MB	0.14 MB

Task 2

Query Task (2) using R-tree		
	Building the index	Executing the query
Time cost	9.547	1.477
Memory cost	100,859,904	23,003,136
I/O cost	1.94 MB	0.40 MB

Query Task (2) using Quad tree		
	Building the index	Executing the query
Time cost	1.068	1.774
Memory cost	10305536	4997120
I/O cost	25.35 MB	8.91 MB

Query Task (2) using Grid based indexing		
	Building the index	Executing the query
Time cost	0.0445	0.7558
Memory cost	1687552	163840
I/O cost	0.00 MB	0.02 MB

Task 3

Query Task (3) using K-d tree		
	Building the index	Executing the query
Time cost	0.00119	0.000452
Memory cost	622592	81920
I/O cost	0.11 MB	0.11 MB

Query Task (3) using R-tree		
	Building the index	Executing the query
Time cost	0.0550	0.000608
Memory cost	425984	16384
I/O cost	0.11 MB	0.02 MB

Query Task (3) using Geo-hash		
	Building the index	Executing the query
Time cost	0.00191	0.001482
Memory cost	671744	65536
I/O cost	0.03 MB	0.00 MB

Task 4

Query Task (4) using Quad tree		
	Building the index	Executing the query
Time cost	2.412	0.000719
Memory cost	201162752	49152
I/O cost	28.86 MB	0.00 MB

Query Task (4) using R-tree		
	Building the index	Executing the query
Time cost	9.672	0.00303
Memory cost	5455872	1884160
I/O cost	5.95 MB	0.02 MB

Query Task (4) using linear scan		
	Building the index	Executing the query
Time cost	N/A	0.323
Memory cost	N/A	0
I/O cost	N/A	0.64 MB

Task 5

Query Task (5) using Quad tree		
	Building the index	Executing the query
Time cost	2.437	0.132
Memory cost	220446720	8372224
I/O cost	2.44 MB	0.00 MB

Query Task (5) using R-tree		
	Building the index	Executing the query
Time cost	9.374	0.389
Memory cost	26787840	3211264
I/O cost	3.92 MB	1.14 MB

Query Task (5) using linear scan		
	Building the index	Executing the query
Time cost	N/A	0.378
Memory cost	N/A	320896
I/O cost	N/A	28.19 MB

Query task (1) using R-tree

```
Time cost of building index: 9.751414
Memory cost of building index: 27901952
Disk I/O cost of building index: 46.78 MB
Time cost of executing query: 0.118304
Memory cost of executing query: 4882432
Disk I/O cost of executing query: 0.35 MB
```

Query task (1) using Quad tree

```
Time cost of building index: 1.019676
Memory cost of building index: 47497216
Disk I/O cost of building index: 0.10 MB
Time cost of executing query: 0.133026
Memory cost of executing query: 3670016
Disk I/O cost of executing query: 0.00 MB
```

Query task (1) using Kd-tree

```
Time cost of building index: 0.050046
Memory cost of building index: 7258112
Disk I/O cost of building index: 0.19 MB
Time cost of executing query: 0.263053
Memory cost of executing query: 245760
Disk I/O cost of executing query: 0.14 MB
```

Query task (2) using R-tree

```
Time cost of building index: 9.546589
Memory cost of building index: 100859904
Disk I/O cost of building index: 1.94 MB
Time cost of executing query: 1.477081
Memory cost of executing query: 23003136
Disk I/O cost of executing query: 0.40 MB
```

Query task (2) using Quad-tree

```
Time cost of building index: 1.067711
Memory cost of building index: 10305536
Disk I/O cost of building index: 25.35 MB
Time cost of executing query: 1.773729
Memory cost of executing query: 4997120
Disk I/O cost of executing query: 8.91 MB
```

Query task (2) using Grid-based indexing

```
Time cost of building index: 0.044537
Memory cost of building index: 1687552
Disk I/O cost of building index: 0.00 MB
Time cost of executing query: 0.755876
Memory cost of executing query: 163840
Disk I/O cost of executing query: 0.02 MB
```

Query task (3) using k-d tree

```
Time cost of building index: 0.001189
Memory cost of building index: 622592
Disk I/O cost of building index: 0.11 MB
Time cost of executing query: 0.000452
Memory cost of executing query: 81920
Disk I/O cost of executing query: 0.11 MB
```

Query Task (3) using Geo-hash		
	Building the index	**Executing the query**
Time cost	0.00191	0.001482
Memory cost	671744	65536
I/O cost	0.03 MB	0.00 MB

Query task (3) using R tree

```
Time cost of building index: 0.054978
Memory cost of building index: 425984
Disk I/O cost of building index: 0.42 MB
Time cost of executing query: 0.000608
Memory cost of executing query: 16384
Disk I/O cost of executing query: 0.02 MB
```

Query task (3) using Geohash

```
Time cost of building index: 0.001918
Memory cost of building index: 671744
Disk I/O cost of building index: 0.03 MB
Time cost of executing query: 0.001483
Memory cost of executing query: 65536
Disk I/O cost of executing query: 0.00 MB
```

Query task (4) using Quad tree

```
Time cost of building index: 2.412261
Memory cost of building index: 201162752
Disk I/O cost of building index: 28.86 MB
Time cost of executing query: 0.000719
Memory cost of executing query: 49152
Disk I/O cost of executing query: 0.00 MB
```

Query task (4) using R-tree

```
Time cost of building index: 9.672251
Memory cost of building index: 5455872
Disk I/O cost of building index: 5.95 MB
Time cost of executing query: 0.003025
Memory cost of executing query: 1884160
Disk I/O cost of executing query: 0.02 MB
```

Query task (4) using linear scan

```
Time cost of executing query: 0.322527
Memory cost of executing query: 0
Disk I/O cost of executing query: 0.64 MB
```

Query task (5) using Quad tree

```
Time cost of building index: 2.436975
Memory cost of building index: 220446720
Disk I/O cost of building index: 2.44 MB
Time cost of executing query: 0.132186
Memory cost of executing query: 8372224
Disk I/O cost of executing query: 0.00 MB
```

Query task (5) using R tree

```
Time cost of building index: 9.373613
Memory cost of building index: 26787840
Disk I/O cost of building index: 3.92 MB
Time cost of executing query: 0.389203
Memory cost of executing query: 3211264
Disk I/O cost of executing query: 1.14 MB
```

Query task (5) using linear scan

```
Time cost of executing query: 0.377682
Memory cost of executing query: 720896
Disk I/O cost of executing query: 28.19 MB
```

Let Q be the parent quadrant, a rectangular bounding box with *(x_min, y_min, x_max, y_max)* coordinates. It is sub-divided in vertically and horizontally through the center point into Q1, Q2, Q3, Q4. The coordinates are as follows:

Q1: (x_min, y_min, (x_min + x_max)/2, (y_min + y_max)/2)
Q2: ((x_min + x_max)/2, y_min, x_max, (y_min + y_max)/2)
Q3: (x_min, (y_min + y_max)/2, (x_min + x_max)/2, y_max)
Q4: ((x_min + x_max)/2, (y_min + y_max)/2, x_max, y_max)

Methodology.B *Task queries.b*
Three constraints were set: 1. Time-window was set between 5am to 12pm on a Sunday 2. Bounding rectangle that encompassed Museum of the City of New York was defined 3. Maximum distance from the bounding rectangle was set as 5 km.

***Abstract*—Indexing algorithms play an important role in searching and information retrieval of data. When high-dimensional data is encountered, advanced indexing namely Quad tree, K-d tree and R-tree could be useful means. This article demonstrated the above algorithms and its implementation on real world queries. Evaluation and analysis were performed to compare and provide insights.**

For the implementation of task 1, *task1()* function was created with csv file, bounding box of a specified area (i.e. Brooklyn), specified start time and end time (i.e. time window of New Year) and venue category (i.e. 'bar') as the parameters. *task2()* function was created for task 2 which included

To split a leaf node *u*, the following steps are illustrated (Huang, 2023):

1. Let *n* be the number of data points in *u*
2. Sort the data points in *u* on one selected dimension
3. For $i = \lceil 0.4B \rceil \; to \; n - \lceil 0.4B \rceil$
4. The set of first *i* points will go to *S1*
5. The rest of the points will go to *S2*
6. Find the perimeter sum of *MBR(S1)* and *MBR(S2)*
7. For other dimensions, repeat step 2-6
8. Get the best split with minimum perimeter

Algorithm 1: Grid-based indexing

Input: *location dataset, grid cell size*
Output: *Index*

```
index = {}
For each item in location dataset do
    cell_x = latitude of item / grid cell size
    cell_y = longitude of item / grid cell size
    if (cell_x, cell_y) is not in index then
        (cell_x, cell_y) as key with an empty list
    append item to key (cell_x, cell_y)
return index
```

Query Task (5) using R-tree		
	Building the	Executing the

	index	query
Time cost	9.374	0.389
Memory cost	26787840	3211264
I/O cost	3.92 MB	1.14 MB