Assignment #1
Big Data Management COSC2636/2632
Submission Deadline: Week 5, April 5 2019 11:59pm
Student Name: Syed Hariz
Student ID: S3701799

Introduction

This report contains a brief explanation of algorithms and data structure used to match spatial keyword queries and retrieve matching objects in the spatial region. JAVA language is used to program the algorithm to achieve this task. The algorithm proposed is to process different variations of spatial keyword query relaxations called INSPIRE (Incremental Spatial Prefix query Relaxation). The goal of this assignment is to demonstrate the INSPIRE framework that uses multiple types of relaxation including, spatial region expansion and exact/approximate prefix/substring matching. The idea of this framework is to enhance the efficiency and effectiveness of spatial keyword search.

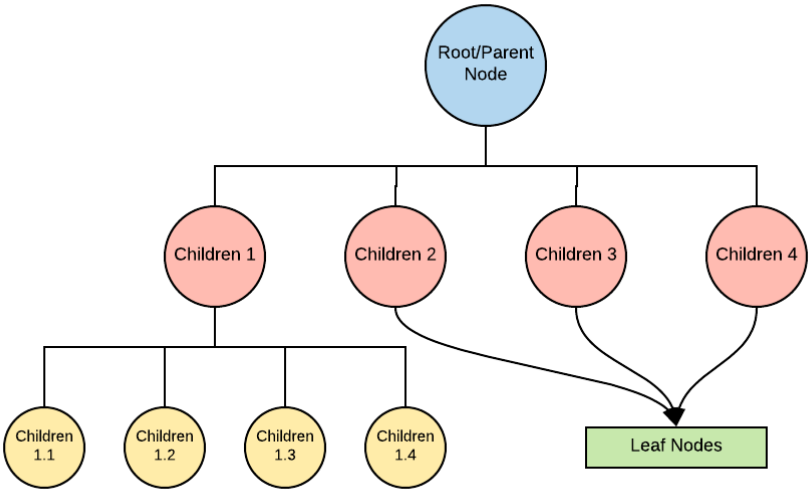
The outline of this report will include, type of data structure used to store objects, how data is being stored through the JAVA program, an explanation of arguments passed in IndexBuilding() method parameter and explanation of how the parameter is being used to create Map and Tree in the program. This report contains illustrations to further improve the understanding of the concepts listed.

Index structure

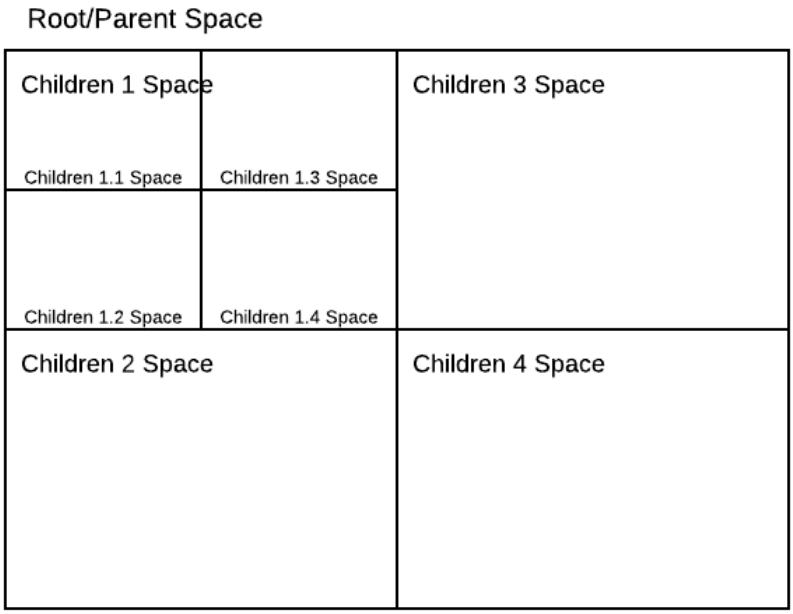
Quad-tree

A quad tree is a data structure or a tree data structure. It is best for fast information retrieval in multi-dimensional space. It is appropriate for information storing which can be retrieved by composite keys. A composite key is merging two columns of a table which can be used to identify each row in the table uniquely. Each node in a quadtree can hold a maximum of 4 other nodes. In two dimensional quad-tree, each node represents a box that covers a part of the space. The root node covers the whole space for itself, the space will be decomposed into equal quadrants with each leaf nodes containing data of its subregion. A node with children will then divide its own subregion space into equal quadrants. The subdivided region can be a square, rectangular or unspecified shape. Top of the tree node can be referred to as the parent or root node which has other nodes extended from it, the children node. A leaf node is a node with no children. A children node can have nodes extended from it as well.

Quad-tree illustration;



Quad-tree space illustration;

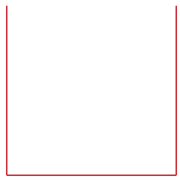


Data structure

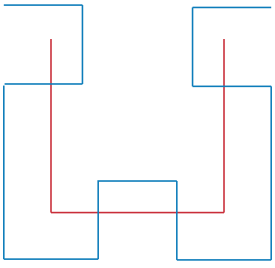
The data structure used is a mixture of Hilbert curve in the Quad-tree space, we can refer to this as Hilbert-encoded Quad-tree. The usage of applying a Hilbert curve onto the Quad-tree space is to convert the multi-dimensional space into one dimension where the curve fills out the space. Each cell (space) is divided into four subcells into each Hilbert curve levels and the order of the subcells in the curve depends on its Hilbert Code. The advantage of applying this method is to efficiently retrieve places that is located in the spatial region since space is broken down into much detail pieces.

Hilbert curve patterns;

1st Order:

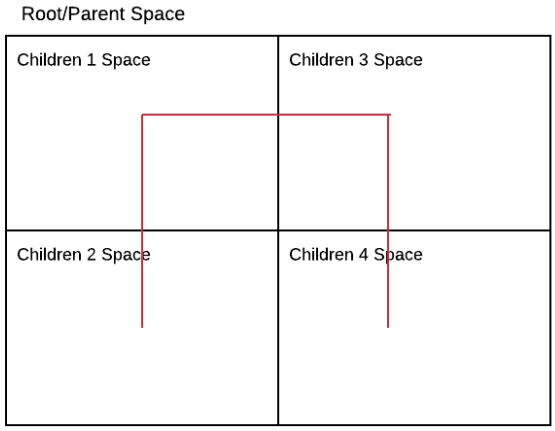


1st order & 2nd Order:

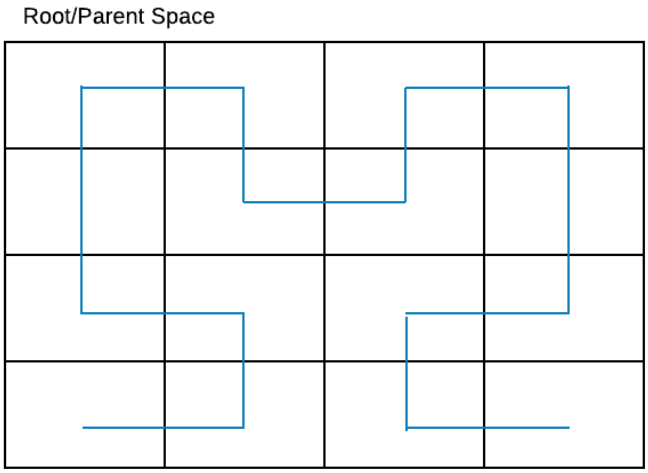


Hilbert code in Quad-tree space illustration;

1st Order:



2nd Order:



Incremental relaxation

The idea of this framework is to efficiently expand the search gap of a query. An incremental relaxation approach starts with Spatial Prefix query. If there are no results returned or a few results are returned, the framework will then triggers relaxation according to its relaxation hierarchy. Depending on just one query method to return results would lower the chances of searching for the right or desired results. Therefore, this framework includes various types of relaxation to return the most accurate and various results of a query.

Spatial Prefix Query (SP)

Spatial Prefix Query contains 2 parts, the spatial region according to the user's viewport and the query string.

SP = (user's spatial region, user's query string)

An object in the Quad-tree spatial region will answer SP query if Object Location is in User's Spatial Region and Object String is a prefix of Object String.

Relaxation of Spatial Prefix Query (SP)

Spatial Prefix Query can be relaxed in two ways; Relaxation of Spatial Region & Relaxation of Prefix Constraint. Relaxation of Spatial Region (SPR) includes expanding the width of the user's search region to return matching query back to the user. There are three types of Relaxation of Prefix Constraint; Spatial Substring (SS) query, Spatial Approximate Prefix (SAP) query and Spatial Approximate Substring (SAS) query. Spatial Substring (SS) query is matching the query from the user with the string of an object that contains the keyword. Spatial Approximate Prefix (SAP) query is returning an object/objects which has the closest match between the object's string and the user's query. Spatial Approximate Substring (SAS) query is the combination of Spatial Substring (SS) query and Spatial Approximate Prefix (SAP) query where it uses substring method to return objects that match similar keyword with an object's string and return objects that have the closest match to user's query.

Relaxation Hierarchy

Spatial Prefixed query in a Relaxed region ⟹ Spatial Substring query ⟹ Spatial Approximate Prefix query ⟹ Spatial Approximate Substring query

Two level nested inverted index

The INSPIRE framework requires an index that is adequate to support the spatial prefix query and its various types of relaxation. Based on the concept, the two-level inverted index is stored on disk.

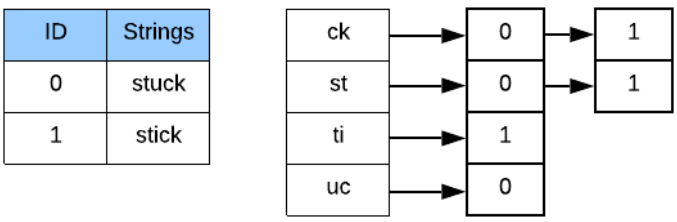
Spatial-q gram

A spatial q-gram is used to capture the spatial and textual information of an object by linking the Hilbert code and q-gram. Matching spatial q-gram is used to remove objects that cannot satisfy the user's query. Since spatial positional q-gram contains Hilbert code and q-gram. The formula for matching spatial q-gram is when; HilbertCode1 = HilbertCode2 & q-gram1 = q-gram2 and Spatial positional q-gram match a threshold; HilberCode1 = HilbertCode2, q-gram1 = q-gram2 and | StringPosition1 – StringPosition2 | <= Threshold.

Object-level & Node-level inverted index

An object-level inverted index is built to store index on spatial q-grams where the value of spatial q-gram is the key and objects of the spatial q-gram as the value. The node-level inverted index is used to return the number of objects having a particular q-gram by setting the q-gram as the key and a list of the Hilbert code and count pairs as a value.

Inverted-index illustration by character sequence with 2 q-grams;



Storing Data

JDBM

This program uses JDBM as its database storage. JDBM provides functionality such as TreeMap, HashMap and other collections which are backed up by disk storage. JDBM is a Java Database. JDBM supports single transaction per store, it does not have multiple transactions with rows/columns. All classes of JDBM are contained in `org.apache.jdbm.DB` package. The two important classes in JDBM are; DBMaker a builder that opens the database, DB is the database which opens the collections.

Code explanation

```
//Line 257
public static void  indexBuilding(String indexFile, int qgramLength, int
largerQgramLength, int positionUpperBound, int maxElement, int
infrequentThreshold, int sparseThreshold, QuadTree quadtree, HashSet<String>
infrequentPositionalQgramSet, HashSet<String> infrequentQgramTokenSet,
SpatialObjectDatabase objectDatabase, InfrequentPositionalQgramInvertedIndex
infrequentQgramIndex, InfrequentQgramTokenInvertedIndex infrequentQgramTokenIndex)
{

FirstLevelInvertedIndex firstLevelInvertedIndex = new
FirstLevelInvertedIndex(indexFile, qgramLength);
firstLevelInvertedIndex.createTree();

// create second level inverted index
SecondLevelInvertedIndex secondLevelInvertedIndex = new
SecondLevelInvertedIndex(indexFile, qgramLength);
secondLevelInvertedIndex.createMap();

QgramTokenCountPairInvertedIndex qgramTokenCountPairInvertedIndex = new
QgramTokenCountPairInvertedIndex(indexFile);
qgramTokenCountPairInvertedIndex.createTree();

HilbertQgramTokenInvertedIndex hilbertQgramTokenInvertedIndex = new
HilbertQgramTokenInvertedIndex(indexFile);
hilbertQgramTokenInvertedIndex.createTree();

// build inverted database
Long startTime = System.currentTimeMillis();



quadtree.buildInvertedIndexNew(firstLevelInvertedIndex,
qgramTokenCountPairInvertedIndex, secondLevelInvertedIndex,
hilbertQgramTokenInvertedIndex, objectDatabase, infrequentPositionalQgramSet,
infrequentQgramTokenSet, sparseThreshold);

// save inverted database
firstLevelInvertedIndex.flush();
secondLevelInvertedIndex.flush();
qgramTokenCountPairInvertedIndex.flush();
hilbertQgramTokenInvertedIndex.flush();

// print the index built time
Long totalTime = System.currentTimeMillis() - startTime;
```

```
System.out.println("node-level and object-level inverted index build time: " +
totalTime / 1000 + " seconds");
System.out.println(" ");

System.out.println("number of sparse nodes in Quadtree: " +
quadtree.getSparseNodeNumber(sparseThreshold));
System.out.println("number of leaf nodes in Quadtree: " +
quadtree.getNumberOfLeaves());
System.out.println("number of level in Quadtree: " + quadtree.getMaxDepth());
System.out.println("infrequent positional q-gram inverted index size: " +
infrequentQgramIndex._map.size());
System.out.println("first-level positional q-gram inverted index size: " +
firstLevelInvertedIndex._map.size());
System.out.println("second-level positional q-gram inverted index size: " +
secondLevelInvertedIndex._map.size());
System.out.println("infrequent q-gram token inverted index size: " +
infrequentQgramTokenIndex._map.size());
System.out.println("first-level q-gram token inverted index size: " +
qgramTokenCountPairInvertedIndex._map.size());
System.out.println("second-level q-gram token inverted index size: " +
hilbertQgramTokenInvertedIndex._map.size());

}
```

1. String indexFile

indexFile is a variable that stores the reference to the sg.txt file. In the main parameter (String args[]) indexFile is declared as `String file = args[0];` whereby the first element of the args array is the path to sg.txt stored in the program. Declaration example; C:\Users\HP\eclipse-workspace\Inspire_assignment\sg.txt index

2. qgramLength

qgramLength is an int variable which holds the value of q-gram. This variable determines how many q-grams characters of a string to be stored in the inverted index. In this assignment, 2-grams is being used.

3.
```
SecondLevelInvertedIndex secondLevelInvertedIndex = new
SecondLevelInvertedIndex(indexFile, qgramLength);
secondLevelInvertedIndex.createMap();
```

To create the second level inverted index, indexFile and qgramLength arguments must be passed in its parameter according to the class's constructor. There are two constructors in SecondLevelInvertedIndex.java class. Another constructor only accepts indexFile as its parameter argument. Since the two-level inverted index is being stored on disk, we use the constructor that accepts two such arguments in its parameter. createMap() function is used to create a set of TreeMap in the database. The arguments that were passed to the class, will be used to pass in the database's createTreeMap() parameter that accepts createTreeMap(String name, Comparator<K> keyComparator, Serializer<k> keySerializer, Serializer<V> valueSerializer). Arguments that were passed to createTreeMap() parameters for database;

```
_map = _database.createTreeMap(Integer.toString(0), new
SecondLevelKeyStringComparator(_qgramLength), null, null );
```

```
4. QgramTokenCountPairInvertedIndex qgramTokenCountPairInvertedIndex = new
   QgramTokenCountPairInvertedIndex(indexFile);
   qgramTokenCountPairInvertedIndex.createTree();
```

This function is called to count the total objects with the same matching q-grams in the first level of the Quad-tree. Once the total number of objects is count, it will create a Tree map in the database containing the objects with matching q-grams data.

```
5. HilbertQgramTokenInvertedIndex hilbertQgramTokenInvertedIndex = new
   HilbertQgramTokenInvertedIndex(indexFile);
   hilbertQgramTokenInvertedIndex.createTree();
```

This function is to count objects in the second level of the Quad-tree with matching amount of q-grams that was declared, in this case, 2 q-grams. It will create a Treemap of its own in the database containing the object's information.

```
6. quadtree.buildInvertedIndexNew(firstLevelInvertedIndex,
   qgramTokenCountPairInvertedIndex, secondLevelInvertedIndex,
   hilbertQgramTokenInvertedIndex, objectDatabase,
   infrequentPositionalQgramSet, infrequentQgramTokenSet, sparseThreshold);
```

This following function is called to build the Quad-tree, the Quad-tree accepts 8 arguments in its parameter. Where the objects of firstLevelInvertedIndex, qgramTokenCountPairInvertedIndex, seconLevelInvertedIndex, hilbertQgramTokenInvertedIndex, SpatialObjectDatabase are passed to store in the spatial region. The new Quad-tree will build levels of objects with specified q-grams according to their levels. The value sparseThreshold is set to 10.

```
7. // save inverted database
   firstLevelInvertedIndex.flush();
   secondLevelInvertedIndex.flush();
   qgramTokenCountPairInvertedIndex.flush();
   hilbertQgramTokenInvertedIndex.flush();
```

flush() function is called to save the inverted indexes in the database. Each individual reference will call the function commit() of DB class. Once a reference call this method, the database commits the action and will make a persistent changes.

```
8. Long startTime = System.currentTimeMillis();
   Long totalTime = System.currentTimeMillis() - startTime;
```

startTime stores the time before the program builds the Quad-Tree.
totalTime stores the time after the program has finished creating the Quad-Tree in the database. Therefore, to get the total time we have to minus off endTime with startTime.

9.
```
System.out.println("node-level and object-level inverted index build time: " + totalTime / 1000 + " seconds");
System.out.println(" ");

System.out.println("number of sparse nodes in Quadtree: " + quadtree.getSparseNodeNumber(sparseThreshold));
System.out.println("number of leaf nodes in Quadtree: " + quadtree.getNumberOfLeaves());
System.out.println("number of level in Quadtree: " + quadtree.getMaxDepth());
System.out.println("infrequent positional q-gram inverted index size: " + infrequentQgramIndex._map.size());
System.out.println("first-level positional q-gram inverted index size: " + firstLevelInvertedIndex._map.size());
System.out.println("second-level positional q-gram inverted index size: " + secondLevelInvertedIndex._map.size());
System.out.println("infrequent q-gram token inverted index size: " + infrequentQgramTokenIndex._map.size());
System.out.println("first-level q-gram token inverted index size: " + qgramTokenCountPairInvertedIndex._map.size());
System.out.println("second-level q-gram token inverted index size: " + hilbertQgramTokenInvertedIndex._map.size());
```

This line of code will print out the values of each references that were created earlier onto the console. Once the program runs, it will return as shown in Program Output section.

Program Output

BuildAll.java

```
274
275 // build inverted database
276 Long startTime = System.currentTimeMillis();
277
278 quadtree.buildInvertedIndexNew(firstLevelInvertedIr
279         objectDatabase, infrequentPositionalQgramSe
280
281 // save inverted database
282 firstLevelInvertedIndex.flush();
283 secondLevelInvertedIndex.flush();
284 qgramTokenCountPairInvertedIndex.flush();
285 hilbertQgramTokenInvertedIndex.flush();
286
287
288 // print the index built time
289 Long totalTime = System.currentTimeMillis() - start
290
291 System.out.println("node-level and object-level inv
292 totalTime / 1000 + " seconds");
293 System.out.println(" ");
294
295 System.out.println("number of sparse nodes in Quadt
296 System.out.println("number of leaf nodes in Quadtre
297 System.out.println("number of level in Quadtree: "
298 System.out.println("infrequent positional q-gram ir
299 System.out.println("first-level positional q-gram i
300 System.out.println("second-level positional q-gram
301 System.out.println("infrequent q-gram token inverte
302 System.out.println("first-level q-gram token invert
303 System.out.println("second-level q-gram token inver
304
305
306
307
308
309
```

Console

```
<terminated> BuildAll (1) [Java Application] C:\Program Files\Java\jdk1.8.0_181\bin\javaw.exe (3
building object database and Quadtree...
object database and quad tree build time : 2 second
number of sparse nodes in Quadtree: 3013
number of leaf nodes in Quadtree: 5320
number of level in Quadtree: 11

positional q-gram infrequent ratio : 2930 / 7323
q-gram token infrequent ratio : 8234 / 15078
infrequent inverted index building time : 0 seconds

building node-level and object-level inverted index...
node-level and object-level inverted index build time: 7 seconds

number of sparse nodes in Quadtree: 3013
number of leaf nodes in Quadtree: 5320
number of level in Quadtree: 11
infrequent positional q-gram inverted index size: 2930
first-level positional q-gram inverted index size: 7323
second-level positional q-gram inverted index size: 234629
infrequent q-gram token inverted index size: 8234
first-level q-gram token inverted index size: 16725
second-level q-gram token inverted index size: 449632
```