**Spring 2017 CSC 244 Homework assignment 2**

Description

Karan Mitra

Sac ID : 219032620

**1. Algorithm -**

The algorithm used to join these two given tuples will be a **index-based simple sort join** using a sorted index. There are a few variations of the simple sort join that need to be applied here since the relation S and the index are sorted on Y. This algorithm is a two-pass algorithm which is required because of the given condition that neither the blocks of R or S can all completely fit into memory. In this algorithm,

* We first load the sorted secondary index of R whose size in much smaller than S and can thus fit into the memory without taking up significant space.
* We then load the number of blocks of S from disk that can fit into M-1 blocks of the remaining space in the memory.
* Once the blocks have been loaded, we loop through the Y attribute in the R index, comparing them to the Y attribute of the tuples of S.
* If a match is found, the corresponding block required to extract the tuple of R is brought in from disk using the pointer of that tuple in the index. This block is stored in the single empty block in the memory. At this point, the memory is completely utilized.
* Once this block has been brought in, the appropriate and required tuple of R is extracted from this block, we join the remaining tuples of R and S to form a joined tuple.
* This joined tuple is fed to the output buffer. If the output buffer is full, it is emptied out to the disk.
* Once all the tuples in memory of S are done being checked, the next required tuples of S are brought in till the time all are done.
* The output buffer is now flushed to output any joined tuples which may have remained in the buffer.
* The final output containing all the tuples is now present in the disk.

**2. Description of Simulation of memory and disk storage -**

Disk storage has been simulated by using the disk storage of the running machine. Text files, containing the required tuples of R and S are placed at the location of the code. These files are read and stored in an in-memory class, each object of which represents a tuple in the corresponding relation. There are also classes which represent the blocks of these tuples. They simply contain arrays of the above mentioned tuples to simulate blocked loading of tuples. I have assumed 3 tuples per block for both R and S.

Memory has been simulated by a java class. It has a single instance which simulates the required memory structure. The object of this memory structure is capable of holding the index (represented by a java util Map), one block of R and 2 blocks of S. This size restriction ensures the given memory condition of B(R) >M and B(S) >M since all block of R or S will not fit into memory.

Index has been simulated as a java util Map which is built and is present in the disk simulation, at the same level where the relations R and S lie after their respective files have been parsed. This index is later brought into memory before the tuple matching begins.

**3. Example used to test the program -**

**R (A,B,Y)**

1 R1 456

2 R2 345

3 R3 234

4 R4 123

5 R5 789

6 R6 567

7 R7 901

8 R8 123

9 R9 910

10 R10 902

11 R11 091

12 R12 798

**S(Y,Z,X)**

123 S1 1.1

132 S5 5.5

234 S2 2.2

234 S4 10.1

234 S4 11.1

243 S6 6.6

345 S3 3.3

345 S7 12.1

456 S4 4.4

456 S8 8.8

798 S7 7.7

**Output**

Y A B Z X

------------------------------------------

123 8 R8 S1 1.1

234 3 R3 S2 2.2

234 3 R3 S4 10.1

234 3 R3 S4 11.1

345 2 R2 S3 3.3

345 2 R2 S7 12.1

456 1 R1 S4 4.4

456 1 R1 S8 8.8

798 12 R12 S7 7.7

CSC 244

HW 2

How\_to\_run (Tested on personal ubuntu and atoz, java 7 or higher required to run)

Karan Mitra

219032620

The code can be run by ensuring the following four files are in the same directory -

1. NaturalJoin.java

2. rTuple.java

3. rBlock.java

4. sTuple.java

5. sBlock.java

6. Memory.java

7. JoinTuple.java

Run the following commands on a linux terminal or dos terminal -

javac NaturalJoin.java

java NaturalJoin

If the code is run through eclipse, the the input files need to be in the projet directory.

If the code is run through the terminal or command prompt, then the input files need to be present in the same directory as the java files.

Final output is present in a file called output.txt which will be generated in the same directory.

The example data, files rData.txt and sData.txt are included in the submission.

/\*\*

\* **@author** Karan Mitra

\* Subject: CSC 244, Spring 2017

\* Assignment: 2 - Natural Join

\*/

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** java.util.HashMap;

**import** java.util.List;

**import** java.util.Map;

/\*\*

\* This is the main class that contains the business logic

\* All the required objects for this are defined in separate classes.s

\* A file - rData.txt and sData.txt are required for this to work.

\* The final output is generated in a output.txt file in the same directory.

\*/

**public** **class** NaturalJoin {

**static** BufferedReader *inputBuffer* = **null**; //Input file reader

**static** BufferedReader *inputBufferS* = **null**;

**static** BufferedWriter *outputFileBuffer* = **null**; //Output file writer

/\*\*

\* Function - Used to set the reader to the file from which

\* data is supposed to be read.

\* **@param** fileName - The full name of the file

\*/

**public** **static** **void** readFromDisk(String fileName) {

**try** {

*inputBuffer* = **new** BufferedReader(**new** FileReader(fileName));

// System.out.println("Input buffer set to - " + fileName);

} **catch** (FileNotFoundException e) {

System.***out***.println("File not found : " + fileName);

e.printStackTrace();

}

}

/\*\*

\* Used when a block of R needs to be fetched after a Y from the index matches

\* with a Y from S.

\* **@param** position - This is the pointer which is stored in the second column of the index.

\* **@return** rBlock - This function fetches and returns the block in which the required tuple resides.

\*/

**public** **static** rBlock fetchBlockofTuple(**int** position) {

*readFromDisk*("rData.txt");

String line = **null**;

String tokens[] = **null**;

**int** place = position % 3;

rBlock r = **new** rBlock();

position -= place;

//Fetch the block

//Find first tuple of required block

**for**(**int** i = 0; i <= position-1 ; i++) {

**try** {

line = *inputBuffer*.readLine();

} **catch** (IOException e) {

e.printStackTrace();

}

}

//Build the required block

**for**(**int** i=0; i <= 2; i++) {

**try** {

line = *inputBuffer*.readLine();

} **catch** (IOException e) {

e.printStackTrace();

}

**if**(line != **null**) {

tokens = line.split(" ");

r.rBlock[i] = **new** rTuple(Integer.*parseInt*(tokens[0]), tokens[1], Integer.*parseInt*(tokens[2]));

}

}

// System.out.println("The block fetched is : " + r);

**return** r;

}

/\*\*

\* This function is used at two places in the code -

\* 1. When the outputBuffer is full and needs to be emptied out to disk.

\* 2. At the end of the join, when the remaining elements of the outputBuffer

\* needs to be put to the disk.

\* **@param** outputBuffer - The memory structure used to simulate the outputBuffer.

\*/

**public** **static** **void** flushOutputBuffer(JoinTuple[] outputBuffer) {

**try** {

**for**(**int** i=0; i <= 2; i++) {

**if**(outputBuffer[i] != **null**)

*outputFileBuffer*.write(outputBuffer[i] + System.*lineSeparator*());

}

} **catch** (IOException e) {

System.***out***.println("File not found : output.txt");

e.printStackTrace();

}

}

**public** **static** **void** main(String[] args) {

String line = **null**;

String[] tokens = **null**;

**try** {

File outputFile = **new** File("output.txt");

*outputFileBuffer* = **new** BufferedWriter(**new** FileWriter(outputFile));

} **catch** (IOException e1) {

// **TODO** Auto-generated catch block

e1.printStackTrace();

}

Map<String, Integer> rIndex = **new** HashMap<String, Integer>();

JoinTuple[] outputBuffer = **new** JoinTuple[3];

**int** opBufTracker = 0;

**try** {

System.***out***.println("Reading data file of R");

*readFromDisk*("rData.txt");

line = *inputBuffer*.readLine();

**int** lineNo = 0;

System.***out***.println("Building the index on R");

**while**(line != **null**) {

tokens = line.split(" ");

rIndex.put(tokens[2], lineNo++);

line = *inputBuffer*.readLine();

}

System.***out***.println(System.*lineSeparator*() + "R index");

System.***out***.println("Y \tPtr");

System.***out***.println("-----------");

**for**(String s : rIndex.keySet())

System.***out***.println(s + "\t" + rIndex.get(s));

System.***out***.println(System.*lineSeparator*());

System.***out***.println("Initializing memory");

Memory memory = **new** Memory();

System.***out***.println("Loading the index into memory");

memory.rIndex = rIndex;

System.***out***.println("Starting the natural join by loading S, 2 blocks at a time into memory");

**try** {

*inputBufferS* = **new** BufferedReader(**new** FileReader("sData.txt"));

} **catch** (FileNotFoundException e) {

e.printStackTrace();

}

line = *inputBufferS*.readLine();

tokens = **null**;

*outputFileBuffer*.write("Y\tA\tB\tZ\tX");

*outputFileBuffer*.write(System.*lineSeparator*() + "-------------------------------------"+System.*lineSeparator*());

**while**(line != **null**) {

**for**(**int** j=0; j <= 1; j++) {

memory.sBlockMem[j] = **new** sBlock();

//Reading the required tuples of r and putting into memory

**for**(**int** i=0; i <= 2; i++) {

**if**(line != **null**) {

tokens = line.split(" ");

memory.sBlockMem[j].sBlock[i] = **new** sTuple(Integer.*parseInt*(tokens[0]), tokens[1], Float.*parseFloat*(tokens[2]));

line = *inputBufferS*.readLine();

}

}

}

List<JoinTuple> joinList = **new** ArrayList<JoinTuple>();

**for**(String rY : rIndex.keySet()) {

**for**(sBlock s : memory.sBlockMem) {

**for**(sTuple sTuple : s.sBlock) {

**if**(sTuple != **null**)

**if**(Integer.*parseInt*(rY) == sTuple.y) { //Match y entry in R to y attribute in S

// System.out.println("Matched " + rY + " with " + sTuple.y);

JoinTuple jt = **new** JoinTuple();

memory.rBlockMem = *fetchBlockofTuple*(rIndex.get(rY)); //Fetch required block

jt.rJoinTuple = memory.rBlockMem.rBlock[rIndex.get(rY) % 3]; //Fetch required tuple

jt.sJoinTuple = sTuple;

//Add joined tuple to outputBuffer

outputBuffer[opBufTracker++] = jt;

**if**(opBufTracker == 2) {

*flushOutputBuffer*(outputBuffer);

outputBuffer = **new** JoinTuple[3];

opBufTracker = 0;

}

// joinList.add(jt);

}

}

}

}

// System.out.println("Joined tuples: " + joinList);

}

*flushOutputBuffer*(outputBuffer);

System.***out***.println("Find output in file - output.txt");

// rTuple rTuple = fetchBlockofTuple(3);

// System.out.println("The tuple is : " + rTuple);

} **catch** (FileNotFoundException e) {

e.printStackTrace();

} **catch** (IOException e) {

e.printStackTrace();

} **finally** {

**try** {

*inputBuffer*.close();

*outputFileBuffer*.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

}

}

/\*\*

\*

\* **@author** karanm1992

\* This data structure represents a tuple formed after joining a tuple of R & S

\*/

**public** **class** JoinTuple {

rTuple rJoinTuple;

sTuple sJoinTuple;

**public** JoinTuple() {

// **TODO** Auto-generated constructor stub

}

@Override

**public** String toString() {

**return** rJoinTuple.y

+ "\t" + rJoinTuple.a + "\t" + rJoinTuple.b

+ "\t" + sJoinTuple.z + "\t" + sJoinTuple.x;

}

}

import java.util.Arrays;

import java.util.HashMap;

import java.util.Map;

public class Memory {

Map<String, Integer> rIndex = new HashMap<String, Integer>();

rBlock rBlockMem;

sBlock[] sBlockMem = new sBlock[2];

public Memory() {

// TODO Auto-generated constructor stub

}

@Override

public String toString() {

return "Memory [rIndex=" + rIndex + System.lineSeparator()

+ " rBlockMem=" + rBlockMem + System.lineSeparator()

+ " sBlockMem=" + Arrays.toString(sBlockMem)

+ "]";

}

}

**public** **class** sTuple {

**int** y;

String z;

**float** x;

**public** sTuple() {

// **TODO** Auto-generated constructor stub

}

**public** sTuple(**int** y, String z, **float** x) {

**super**();

**this**.y = y;

**this**.z = z;

**this**.x = x;

}

@Override

**public** String toString() {

**return** "sTuple [y=" + y + ", z=" + z + ", x=" + x + "]";

}

}

**import** java.util.Arrays;

**public** **class** rBlock {

rTuple[] rBlock = **new** rTuple[3];

**public** rBlock() {

// **TODO** Auto-generated constructor stub

}

@Override

**public** String toString() {

**return** "rBlock [rBlock=" + Arrays.*toString*(rBlock) + "]";

}

}

**public** **class** rTuple {

**int** a;

String b;

**int** y;

**public** rTuple() {

// **TODO** Auto-generated constructor stub

}

**public** rTuple(**int** a, String b, **int** y) {

**super**();

**this**.a = a;

**this**.b = b;

**this**.y = y;

}

@Override

**public** String toString() {

**return** "rTuple [a=" + a + ", b=" + b + ", y=" + y + "]";

}

}

**import** java.util.Arrays;

**public** **class** sBlock {

sTuple[] sBlock = **new** sTuple[3];

**public** sBlock() {

// **TODO** Auto-generated constructor stub

}

@Override

**public** String toString() {

**return** "sBlock [sBlock=" + Arrays.*toString*(sBlock) + "]";

}

}

/\*\*

\*

\* **@author** karanm1992

\* This data structure represents a tuple formed after joining a tuple of R & S

\*/

**public** **class** JoinTuple {

rTuple rJoinTuple;

sTuple sJoinTuple;

**public** JoinTuple() {

// **TODO** Auto-generated constructor stub

}

@Override

**public** String toString() {

**return** rJoinTuple.y

+ "\t" + rJoinTuple.a + "\t" + rJoinTuple.b

+ "\t" + sJoinTuple.z + "\t" + sJoinTuple.x;

}

}