

**COMP - 6521**

**Advanced Database Technology and Applications**

**Project Report**

**On**

**Lab Assignment 2: Bitmap Indexing**

**Professor**

**Dr. Nematollaah Shiri**

**Team Members**

|  |  |  |
| --- | --- | --- |
| **Student Name** | **Student Id** | **Email Id** |
| Yash Pandya | 40119272 | y\_pandy@encs.concordia.ca |
| Himen Sidhpura | 40091993 | h\_sidhpu@encs.concordia.ca |
| Sucheta Sudhakumari | 40080543 | s\_ijaya@encs.concordia.ca |

**Index**

1. Bitmap indexing 3
2. Program Description 4
3. Steps to run the program 4
4. Algorithm 5
5. Experiment Results 7
6. Comparison of results to LA1(TPPMS) 8
7. Coding Standard 9
8. Class Description 9
9. Group Member Contribution 9
10. Results 10
11. **Bitmap Indexing**

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records.  
  
Indexes are primarily built on specific keys /fields of the data base even though some databases extend the power of indexing by letting developers create indexes on functions or expressions.A bitmap index is built on table attributes such that bitmap index for a field F is a collection of bit-vectors of length n, one for each possible value that may appear in the field F. The vector for iralue u

has 1 in position i if the ith record has v in field F, and it ha5 0 there if not.   
  
The number of bit vectors created for a field will be equal to the number of distinct values for that particular field and the length of the bit vector will be equal to the number of rows/tuples in a table.

1. **Program Description**

**Phase 1 –Program Initilization:**  Program execution is triggered by the ProgramController class. Constant values including block size and file I/O paths are initialized .The input files are ready block by block into the program , starting with T1 followed by T2.

**Phase 2 – Creation of bitmap index and sorting index:**  A 2D array of data type:long and size : (number of tuples )\*(number of tuples +1 ) is created to store the bit vectors .The first column holds the distinct values of the field while the remaining columns signify the bits corresponding to each tuple. The row size of the array is set to be equal to the number of tuples, assuming the maximum row numbers possible in case no field values are repeated across the block. Everytime a unique value is read, the 1st column in the most recent row is set to that value after which the corresponding bit vector is set to 1. The key values are then sorted using quicksort and the bitmap index for each block is then written into a sublist.

**Phase 3 –** **Merging index blocks:**  The blockwise bitmap index sublists , corresponding to each file are merged recursively into a single file using TPPMS technique. At the end of this point , we get 6 separate index files corresponding to the 3 fields and 2 files , . For each distinct value of the index, the columns with bit vector 1 indicate duplicates tuples.

**Phase 4 – Compression of Index :** Each of the index files are then compressed using run length encoding algorithm. **Phase 5 –** **Merging the files:**  The bitmap indices of employee file are read into the memory using TPPMS algorithm . For each unique employee number , corresponding tuples are located from data file and checked for duplicates. In case of duplicates , the data is sorted using date and the latest tuple is then written into the output file.

1. **Steps to run the program**

1. Import the project Bitmap-Indexing to the IDE.

2. Set constant values including the block size , memory allocation size , I?O file paths in the constants.java class

3. Run the program from ProgramController.java class.

4. The execution will be performed phase by phase and results will be printed on the console.

1. **Algorithms**

**PHASE - 1 : Program initialization :**

* Start.
* Allocate main memory.
* Allocate block size , file I/O paths and other utility constants.

**PHASE - 2 : Bitmap Index Creation :**

* Start
* Read the file block by block.
* For each block , create a 2D array of size equal to the number of tuples in a block , to store the bitmap indices for the corresponding field.
* For each unique field value read , add a row to the array .
* For each field value read(new or existing in the array ) , set the corresponding bit vector to 1 in the array.
* Add the key value into a list of key values already read.
* Sort the ley list .
* Write the bit vectors for each block into a file , fetching tuples in sorted order(This creates a sorted index).
* Continue till all blocks are read.
* Repeat the steps for indexes on other fields on the same file and also on the other file T2.
* Stop

**PHASE – 3 : Merging Bitmap Index blocks:**

* Start.
* Read bitmap index sublists of the same type, file by file..
* Taking 2 indices at a time , merge them into a single file .
* Repeat the process until a single index file remains for each index type.
* Identify duplicates using the index and eliminate them from the data file.
* Repeat the steps for indexes on other fields and other file T2.
* Stop

**PHASE – 4 : Compression of Bitmap indices:**

* Start.
* Read bitmap index blocks one by one into the program.
* Using Run Length Encoding technique perform index compression.
* Repeat the steps for indexes on other fields and other file T2.
* Stop

**PHASE – 5 : Merging the data files:**

* Start.
* Using TPPMS technique, read the index for employee field into the program.
* For each employee IDs read,, identify the indices with bit vector 1 which correspond to duplicate tuples.
* Find the corresponding block of data from the data blocks.
* Scan the data block and locate all tuples with the same employee ID, adding them to a list .
* Sort the list according to date and write the latest one to the output file.
* Repeat the steps until the end of both index files.
* Stop

1. **Experiment Results:**

The experiment was conducted by changing the input file size , the result of which has been consolidated below:

|  |  |  |  |
| --- | --- | --- | --- |
| File T1 | | | |
| **Tuple Count** | **20,000** | **50,000** | **1,000,00** |
| T1 |  |  |  |
| **Gender Bitmap Index** |  |  |  |
| Time taken to generate index blocks | 0. 230 s | 0.371 s | 0.806 s |
| Time taken for Merge index blocks | 0.022 s | 0.036 s | 0.096 s |
| Time taken for compression of index | 0.004 s | 0.008 s | .016 s |
|  |  |  |  |
| **Department Bitmap Index** |  |  |  |
| Time taken to generate index blocks | 0.279 s | 0.654 s | 1.062 s |
| Time taken for Merge index blocks | .075 s | 0.087 s | 0.172 s |
| Time taken for compression of index | .009 s | 0.019 s | 0.046s |
|  |  |  |  |
| **Employee ID Bitmap Index** |  |  |  |
| Time taken to generate index blocks | 0.526 s | 0.915 s | 1.617 s |
| Time taken for Merge index blocks | 8.677 s | 47.179 s | 206 s |
| Time taken for compression of index | 8.581 s | 53.762 s | 215 s |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| File T2 | | | |
| **Tuple Count** | **20,000** | **50,000** | **1,000,00** |
| T2 |  |  |  |
| **Gender Bitmap Index** |  |  |  |
| Time taken to generate index blocks | 352 ms | 421 ms | 0.498 s |
| Time taken for Merge index blocks | 40 ms | 36 ms | 0.070 s |
| Time taken for compression of index | 4 ms | 10 ms | 0.016 s |
|  |  |  |  |
| **Department Bitmap Index** |  |  |  |
| Time taken to generate index blocks | 254 ms | 349 ms | 0.568 s |
| Time taken for Merge index blocks | 43 ms | 61 ms | 0.183 s |
| Time taken for compression of index | 9 ms | 19 ms | 0.050 s |
|  |  |  |  |
| **Employee ID Bitmap Index** |  |  |  |
| Time taken to generate index blocks | 310 ms | 986 ms | 1.67 s |
| Time taken for Merge index blocks | 7911 ms | 46749 ms | 170 s |
| Time taken for compression of index | 8617 ms | 53650 ms | 217 s |
|  |  |  |  |

**Time taken to output the list of unique tuples : 138.799 seconds**

1. **Comparison to TPPMS**

|  |  |  |
| --- | --- | --- |
|  | Bitmap Indexing | TPPMS |
| Total Number of Disk IOs |  |  |
| Total Time taken |  |  |

1. **Coding Standards**

The most general coding conventions were followed while the codes were developed as follows,

* The class name begins with an uppercase word.
* E.g.: ProgramController.java
* Constants are called with characters in the upper case
* The variable name is descriptive and is rendered in lower case including a capital letter to separate words.
* The procedure name begins with a lowercase character and uses the uppercase characters to separate words.

1. **Class Description:**

**Constants.java:**  This class stores the constant values required for program execution like file IO paths, block size etc.

**ProgramController.java:**  This class has the methods for reading tuples into main memory, sorting them and creating the sublists. It also handles reading the indices and producing final merged output.

**BuildIndex.java:** This class creates bit vectors and bitmap index subsequently from the input files.

**CompressedBitmap.java:** This creates the compressed bitmaps.

**QuickSort.java:**  This class performs the quicksort algorithm on the bitmap index for blocks read into memory.

**MergeData.java:**  This class merges the bitmap for blocks into a single bitmap index.

1. **Group Member Contribution:**

Member participation was uniform across all stages of the project development. We had meetings once a week to discuss on design changes and individual progress. This ensured that everyone is on the same page. We also adopted pair programming strategy which let us help each other with our areas of expertise, thereby developing efficient code. The documentation part was split into sections and assigned to each team mate as a part of even work distribution.

1. **Results :**

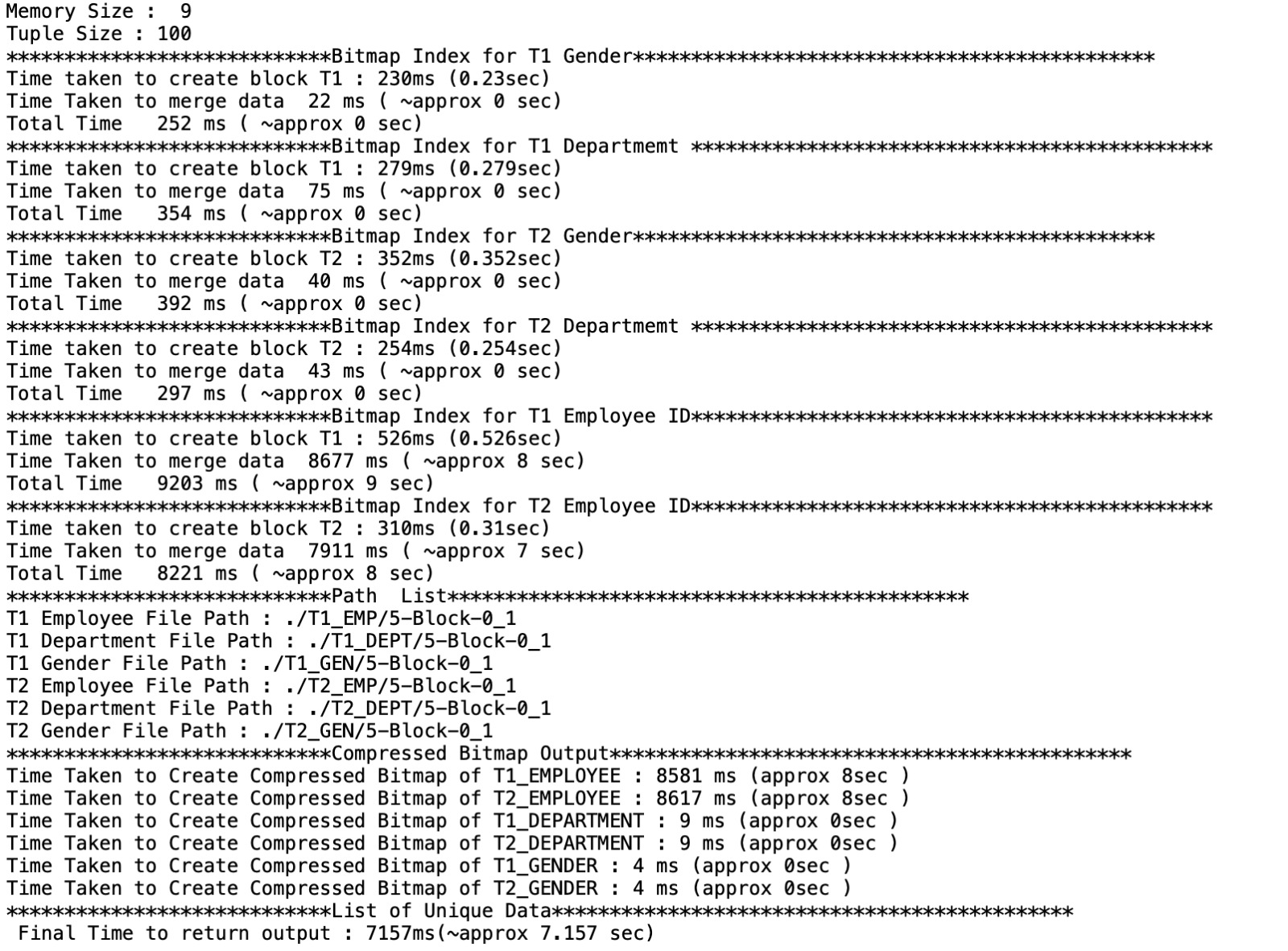
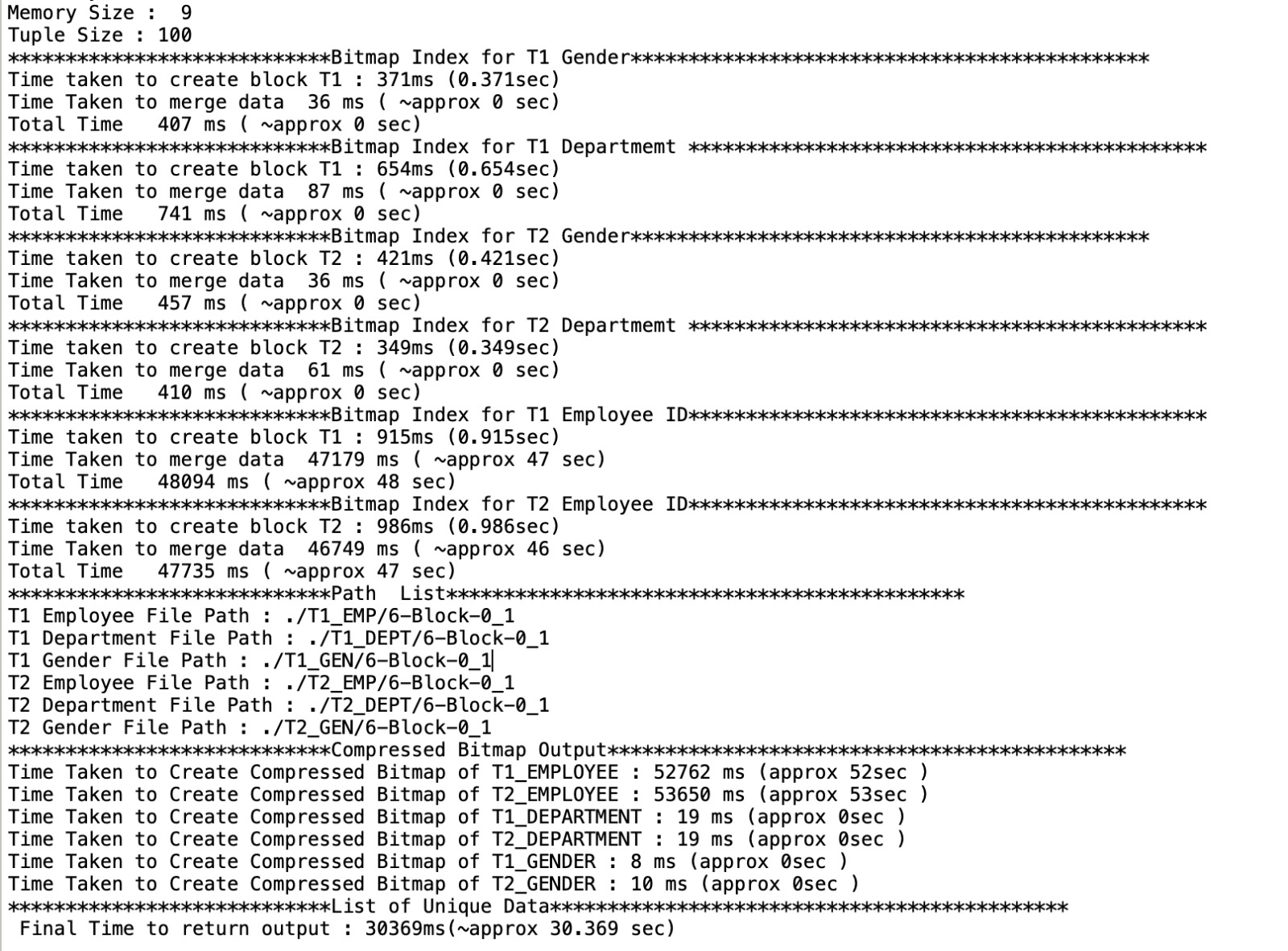
****

Figure 1: 40,000 tuples

Figure 2: 1,00,000 Tuple Counts 

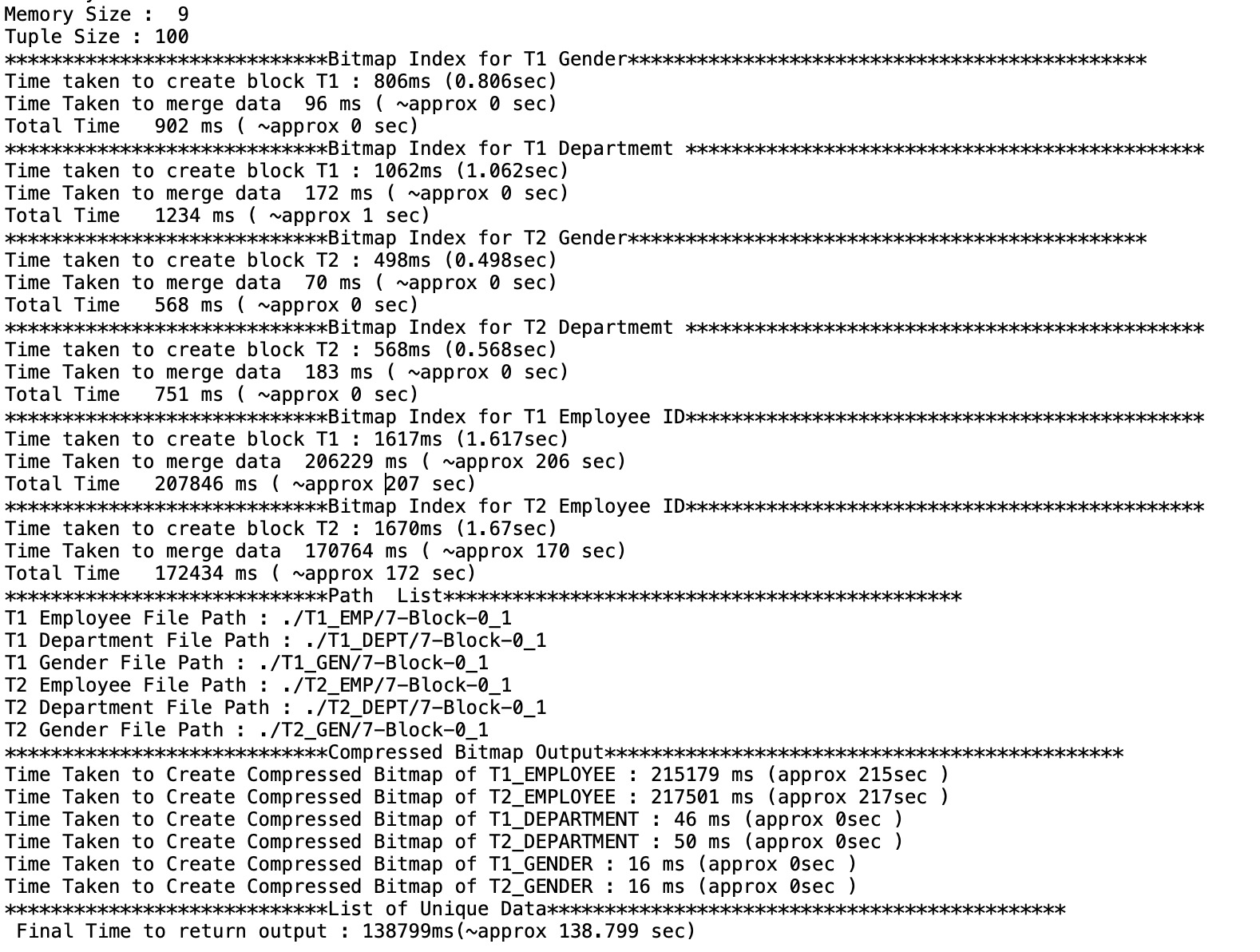
****

Figure *2*: *2,00,000* Tuple Counts

**References(WIP)**

1. <https://www.geeksforgeeks.org/java-program-for-quicksort/>
2. <https://examples.javacodegeeks.com/core-java/nio/bytebuffer/write-append-to-file-with-byte-buffer/>
3. <https://www.geeksforgeeks.org/bytebuffer-get-method-in-java-with-examples/>
4. <http://rosettacode.org/wiki/Binary_search#Java>
5. <https://en.wikipedia.org/wiki/Sorting_algorithm>
6. <https://crunchify.com/increase-eclipse-memory-size-to-avoid-oom-on-startup/>
7. <https://github.com/sagarvetal/ADB_Project_1_TPMMS>
8. <https://howtodoinjava.com/java/io/how-to-check-if-file-exists-in-java/>
9. <https://www.javadevjournal.com/java/java-copy-file-directory/>
10. <https://mkyong.com/java/how-to-delete-directory-in-java/>
11. <https://www.tutorialspoint.com/how-to-create-a-new-directory-by-using-file-object-in-java>
12. <https://www.tutorialspoint.com/java/io/file_isfile.htm>
13. <http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/4-query-exec/2-pass=TPMMS.html>
14. <http://www.mathcs.emory.edu/~cheung/Courses/554/Syllabus/4-query-exec/TPMMS=join2.html>