B.V.S.S.Vidya Charan (2020CS10334) Monu (2020CS50432) E.Sriharshitha (2020CS10345)

COL761 Assignment - 1

Problem Statement:

Given a large transactional dataset D, where each row of the dataset, i.e., a transaction, contains a list of items purchased jointly by some users. We have to compress this dataset D by reducing the repetition of the item sets. Here, the order of the occurrence of items in the transactions doesn't matter until there is no data loss.

We can do this by using frequent itemset mining. This can be done in two approaches: either using Apriori Algorithm or Frequent Pattern Tree Growth. In this assignment, we have used FP Tree Growth approach (which is previously implemented onto which we have added our own modifications).

FP Growth Algorithm:

We have used an implementation of the FP Growth algorithm, which is a popular algorithm used for frequent item set mining. Steps followed in the algorithm for FP Tree construction and growth are as follows:

1. Data Reading:
    - The whole transaction dataset stored in a file is taken as input and stores this into a vector of transactions (say, the database).
    - Each transaction is a vector of items, whereas each item is a string storing the value.
2. Counting Frequencies:
    - The code scans through the transaction database to count the frequency of each single item set. This information is in turn used to find the frequent item sets.
    - These are usually defined on the basis of minimum support threshold value which is to be finetuned and fixed.
3. Building the FP Tree:
    - Here, a tree-like data structure is created which stores the frequency and conditional patterns for each item in the dataset.

- For each transaction, items are sorted in decreasing order of frequency.
- Starting with an empty FP Tree, each transaction is used to grow the tree. For each item in a transaction, the following is done:
    i. If the item is not in the current node's children, a new node is created for it, and a link is added to the node in the FP Tree.
    ii. If the item already exists in the current node's children, the node is incremented to reflect the frequency of the item.
- For each item, conditional FP-Trees are constructed, which are effectively subtrees containing only the item and its ancestors.
- These conditional FP-Trees are used to build the whole FP-Tree iteratively.

4. Mining Frequent Item sets:
    - The FP Tree constructed is now used to mine frequent item sets.
    - These are obtained by recursively exploring the FP Tree and the item sets in a depth-first search manner.
5. Pattern Growth:
    - Find all frequent single item sets, i.e., with size 1.
    - Recursively, build conditional FP Trees on top of it, i.e., do this for each frequent item and mine more frequent item sets on those conditional FP Trees.
    - Do this recursively until no itemset is found.
6. Output:
    - The frequent patterns are thus returned along with their frequency values.

This implementation of FP Growth algorithm helps us to mine the frequent patterns efficiently.

Data Compression Algorithm:

The algorithm we used contains a compressor class, which has the functions to compress and decompress the given files. It works as follows:

1. Private Variables stored:
    - Map – this contains mapping used to compress, i.e., a mapping from the item sets to the value with which it is replaced.

- reverseMap – this is used for decompression. This is the reverse of the previous map stored.
- Set – this contains frequent item sets.

2. Constructor:
   - This takes 2 inputs, frequent item sets and a threshold value. It assigns frequent item sets to the “Set” variable.
   - Associates each set with an integer value ‘thres + 1 + i’ in the Map data member, where ‘I’ is the index of the set in ‘freqItems’. This mapping is used to compress the frequent item sets into integer values.
   - Associates the integer value with the corresponding set in the reverseMap data member, allowing for reverse lookup.
3. Compression:
   - This method takes two parameters: a reference to a vector of Transaction objects called transactions and a string filename.
   - It compresses the transactional data based on the item sets in Set.
   - For each transaction in transactions, it iterates through the sets in Set and replaces any subset of the transaction with the corresponding integer code from Map.
   - It maintains a Hash map to keep track of the frequency of compressed item sets.
   - The method writes the compressed data to the specified filename and calculates compression statistics.
4. Decompression:
   - This method takes two string parameters: compressed_file and decompressed_file.
   - It is responsible for decompressing the data that was previously compressed using the compress method.
   - It reads the compressed data from compressed_file, reconstructs the item sets and transactions, and stores them in a vector called transactions.
   - Then, it writes the decompressed data to decompressed_file.

This class is designed to compress and decompress transactional data using a mapping mechanism that converts sets of items into integer codes and vice versa.

Time and Compression Analysis:

For small file:

```
==============================
3593540.pbshpc
Laregst Num  : 75
-------------Compression Details-----------
Input Size:              118252
Compressed Size:         73133
Diff:                    45119
Percentage Compression: 38
-----------------------------------------------
```

Time taken: 06:58

For medium file:

Support percentage: 1%

```
==============================
3593713.pbshpc
Laregst Num  : 41270
-------------Compression Details-----------
Input Size:              8019015
Compressed Size:         6753205
Diff:                    1265810
Percentage Compression: 15
-----------------------------------------------
```

Time taken: 06:49

Support percentage: 0.5%

```
==============================
3593723.pbshpc
Laregst Num  : 41270
-------------Compression Details-----------
Input Size:              8019015
Compressed Size:         6541608
Diff:                    1477407
Percentage Compression: 18
-----------------------------------------------
```

Time taken: 34:21

Medium2 file:

Support percentage: 5%

```
==============================
3593675.pbshpc
Laregst Num  : 999
------------Compression Details----------
Input Size:             3960507
Compressed Size:        3925309
Diff:                   35198
Percentage Compression: 0.8
-----------------------------------------------
```

Time taken: 24:42

Large file:

Support percentage: 40%

File Edit Format View Help

```
==============================
3593878.pbshpc
Laregst Num  : 2635326
------------Compression Details----------
Input Size:             109360594
Compressed Size:        108074852
Diff:                   1285742
Percentage Compression: 1
-----------------------------------------------
```

Time taken: 02:43

References:

For FP Tree implementation:

integeruser/FP-growth: A C++ implementation of the FP-growth algorithm (github.com)