COL761 (Session 2023-2024)
Assignment 1

Kshitij Singh(⅓) Darshan Rakhewar(⅓) Harshit Rastogi(⅓)

2020CS10353 2020CS10340 2020AM11021

Data Compression Algorithm:

Input:

- Transactions: Lists of (positive integers) items.

Encryption:

Modified FP Growth algorithm:(Reference code - FP-growth)

- Build the initial header table with the frequency of every unique transaction item in the transactions.
- Sort elements in the header table according to their decreasing frequency.
- Build FP-tree:
    - For each transaction, insert items from the transaction into the tree, following the order specified by the sorted header table.
    - Generate conditional FP-Trees for each item, starting from the least frequent item.
- Mined the FP-Tree for frequent item sets by recursively traversing the tree only for the specific number of the most frequent items. Store these frequent item sets.

Encrypting the Transactions:

- Sort the frequent item sets in decreasing order of **frequency*size**.
- For each transaction:
    - Check if any frequent item set is a subset of the transaction. If found, replace it with a mapping for the frequent item set. This mapping is a new negative integer.
    - Repeat.
- Generate the encryption mapping table only for the used frequent item sets.

Repeat:

- Repeat the compression with the encrypted transactions.

Output:

- Compressed transactions: Encrypted list of transactions.
- Recursive Encryption mapping: Map from encryption keys to frequent item sets.

Data Decompression Algorithm:

Input:

- Compressed transactions: Encrypted list of transactions.
- Recursive Encryption mapping: Map from encryption keys to frequent item sets.

Decryption:

- Recursively for every encryption:
    - For each encrypted transaction in the list, replace the unique encryption keywords with their corresponding mapping to decrypt it.

Output:

- Original transactions: List of the original transactions.

Results:

Dataset	Compression	Time
Small (342KB)	81%	30sec
Medium(15.5MB)	42%	418sec
Medium2(32MB)	50%	418sec
Large(539.5MB)	14%	3600sec

The small data size shows better compression in less time, due to more iterations running, and as the data sizes increases the time increases exponentially as the number of transactions increase exponentially thus growing way large.