A transaction database

|  |  |  |
| --- | --- | --- |
| **ID** | **items** | **ordered frequent items** |
| 1 | a, c, g, f | c, f, a |
| 2 | e, a, c, b | b, c, e, a |
| 3 | e, c, b, i | b, c, e |
| 4 | b, f, h | b, f |
| 5 | b, f, e, c, d | b, c, e, f |

Minsupport: 40%

(0,4 \* 5) = 2

|  |  |
| --- | --- |
| ITEM | SUPPORT COUNT |
| a | 2 |
| b | 4 |
| c | 4 |
| d | 1 |
| e | 3 |
| f | 3 |
| g | 1 |
| h | 1 |
| i | 1 |

F1= {b,c,e,f,a}

**PPC-tree**

**PP-code:** each node of PPC-tree is a PP-code

**N-list:** a sequence of all the PP-codes of nodes registering the item in the PPC-tree.

The N-list of frequent items in Example 1:

b 🡪 < (4,8):4>

c 🡪 < (1,2):1> --- < (5,6):3>

e 🡪 < (6,5):3>

f 🡪 < (2,1):1> --- < (8,4):1> --- < (9,7):1>

a 🡪 < (4,8):4> --- < (7,3):1>

b 🡪 < (4,8):4>

c 🡪 < (1,2):1> --- < (5,6):3>

e 🡪 < (6,5):3>

f 🡪 < (2,1):1> --- < (8,4):1> --- < (9,7):1>

a 🡪 < (4,8):4> --- < (7,3):1>

bf 🡪 < (4,8):2>

**PrePost algorithm**

1. Construct PPC-tree and identify all frequent 1-itemsets.
2. Based on PPC-tree, construct the N-list of each frequent 1-itemsets.
3. Scan PPC-tree to find all frequent 2-itemsets.
4. Mine all frequent k-itemsets.