FP-Growth algorithm

Introduction

- Apriori: uses a generate-and-test approach – generates candidate itemsets and tests if they are frequent
  - Generation of candidate itemsets is expensive(in both space and time)
  - Support counting is expensive
    - Subset checking (computationally expensive)
    - Multiple Database scans (I/O)
- FP-Growth: allows frequent itemset discovery without candidate itemset generation. Two step approach:
  - Step 1: Build a compact data structure called the FP-tree
    - Built using 2 passes over the data-set.
  - Step 2: Extracts frequent itemsets directly from the FP-tree

Step 1: FP-Tree Construction

- FP-Tree is constructed using 2 passes over the data-set:

Pass 1:

- Scan data and find support for each item.
- Discard infrequent items.
- Sort frequent items in decreasing order based on their support.

Use this order when building the FP-Tree, so common prefixes can be shared.

Step 1: FP-Tree Construction

Pass 2:

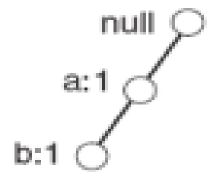
Nodes correspond to items and have a counter

1. FP-Growth reads 1 transaction at a time and maps it to a path
2. Fixed order is used, so paths can overlap when transactions share items (when they have the same prfix).
   - In this case, counters are incremented
3. Pointers are maintained between nodes containing the same item, creating singly linked lists (dotted lines)
   - The more paths that overlap, the higher the compression. FP-tree may fit in memory.
4. Frequent itemsets extracted from the FP-Tree.

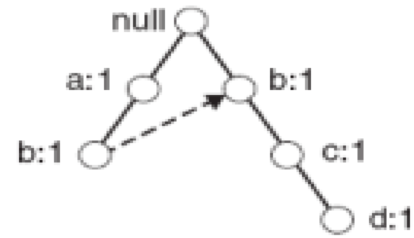
Step 1: FP-Tree Construction (Example)

Transaction
Data Set

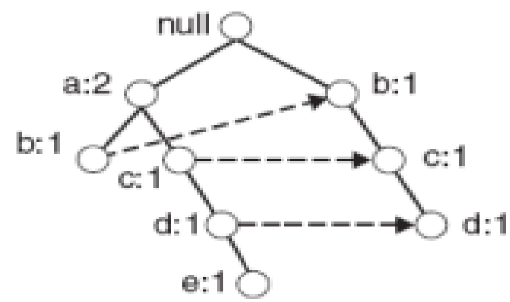
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



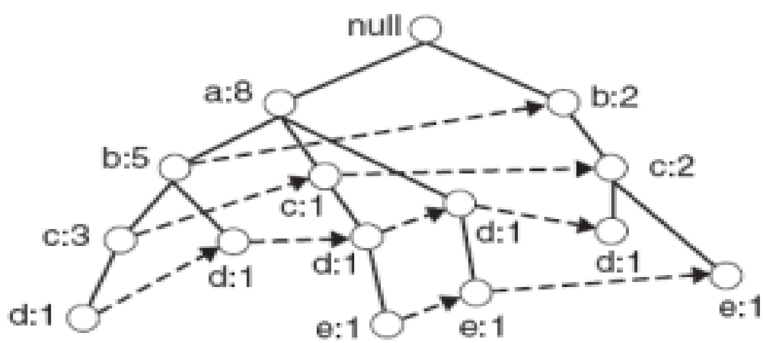
(i) After reading TID=1



(ii) After reading TID=2



(iii) After reading TID=3



(iv) After reading TID=10

FP-Tree size

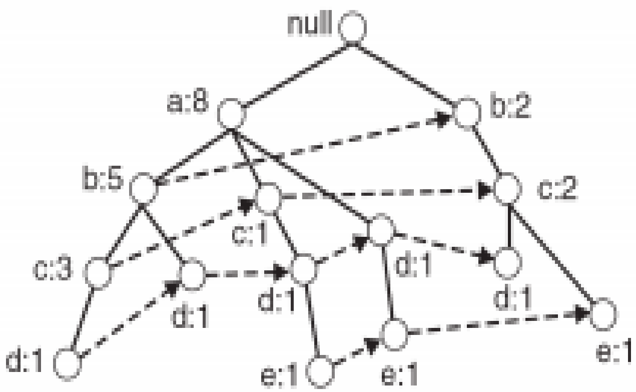
- The FP-Tree usually has a smaller size than the uncompressed data - typically many transactions share items (and hence prefixes).
  - Best case scenario: all transactions contain the same set of items.
    - 1 path in the FP-tree
  - Worst case scenario: every transaction has a unique set of items (no items in common)
    - Size of the FP-tree is at least as large as the original data.
    - Storage requirements for the FP-tree are higher - need to store the pointers between the nodes and the counters.

- The size of the FP-tree depends on how the items are ordered
- Ordering by decreasing support is typically used but it does not always lead to the smallest tree (it's a heuristic).

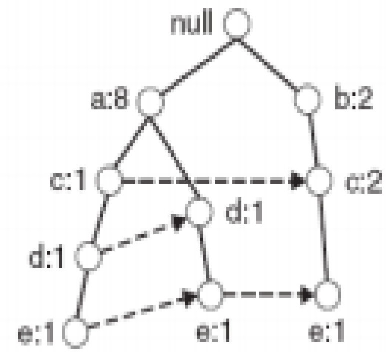
Step 2: Frequent Itemset Generation

- FP-Growth extracts frequent itemsets from the FP-tree.
- Bottom-up algorithm - from the leaves towards the root
- Divide and conquer: first look for frequent itemsets ending in e, then de, etc. . . then d, then cd, etc. . .
- First, extract prefix path sub-trees ending in an item(set). (hint: use the linked lists)

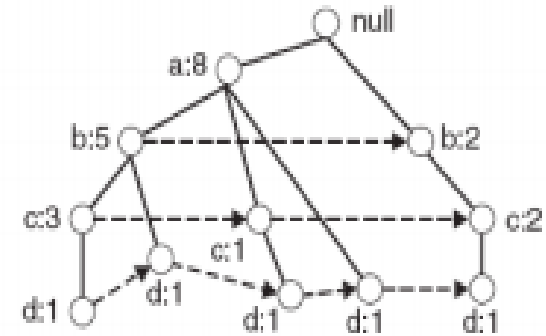
Prefix path sub-trees (Example)



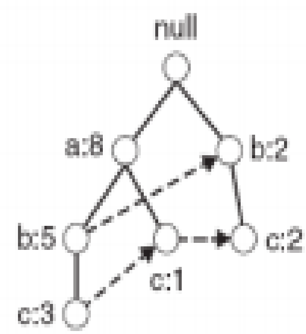
↑ Complete FP-tree



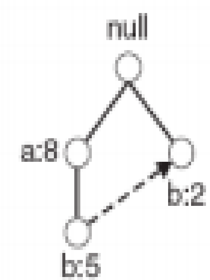
(a) Paths containing node e



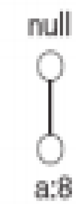
(b) Paths containing node d



(c) Paths containing node c



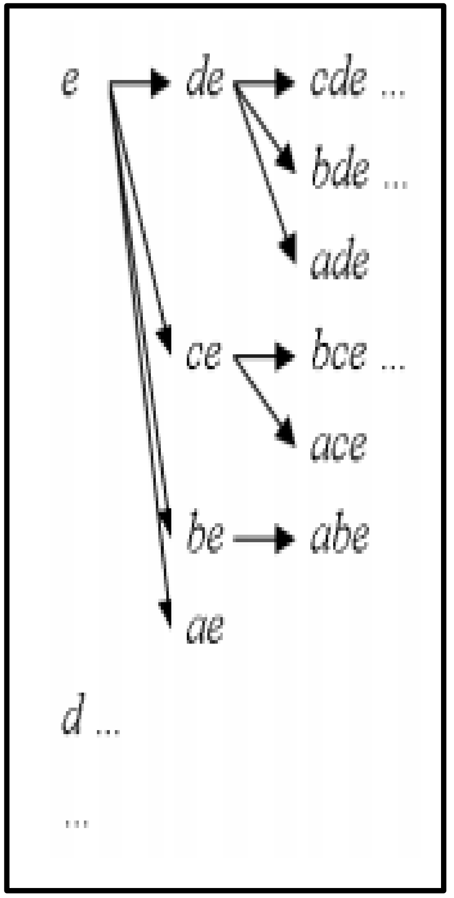
(d) Paths containing node b



(e) Paths containing node a

Step 2: Frequent Itemset Generation

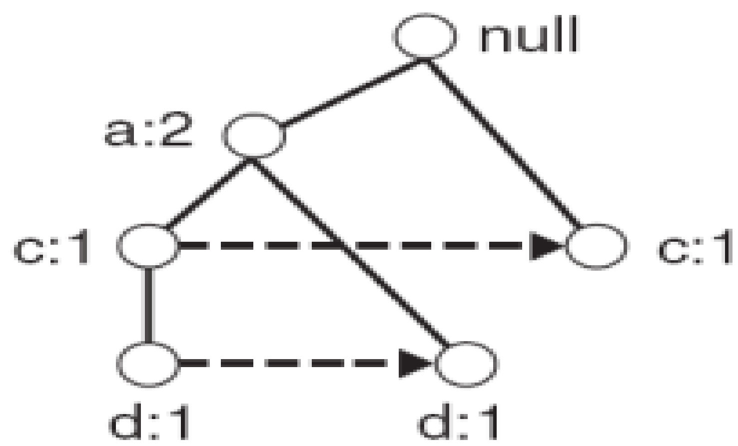
- Each prefix path sub-tree is processed recursively to extract the frequent itemsets. Solutions are then merged.
  - E.g. the prefix path sub-tree for *e* will be used to extract frequent itemsets ending in e, then in de, ce, be and ae, then in cde, bde, cde, etc.
  - Divide and conquer approach



Conditional FP-Tree

- The FP-Tree that would be built if we only consider transactions containing a particular itemset (and then removing that itemset from all transactions).
- I Example: FP-Tree conditional on e.

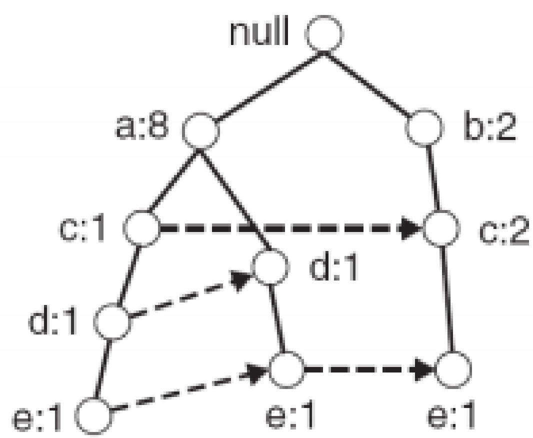
TID	Items
~~1~~	~~{a,b}~~
~~2~~	~~{b,c,d}~~
3	{a,c,d,~~e~~}
4	{a,d,~~e~~}
~~5~~	~~{a,b,c}~~
~~6~~	~~{a,b,c,d}~~
~~7~~	~~{a}~~
~~8~~	~~{a,b,c}~~
~~9~~	~~{a,b,d}~~
10	{b,c,~~e~~}



Example

Let minSup = 2 and extract all frequent itemsets containing e.

- 1. Obtain the prefix path sub-tree for e:



Example

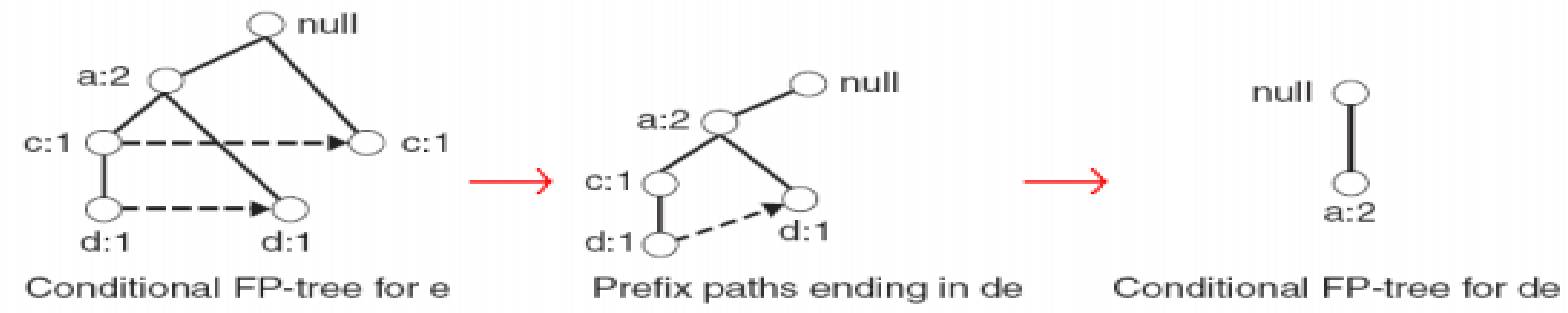
- 2. Check if e is a frequent item by adding the counts along the linked list (dotted line). If so, extract it.
  - Yes, count =3 so {e} is extracted as a frequent itemset.
- 3. As e is frequent, find frequent itemsets ending in e. i.e. de, ce, be and ae.

Example

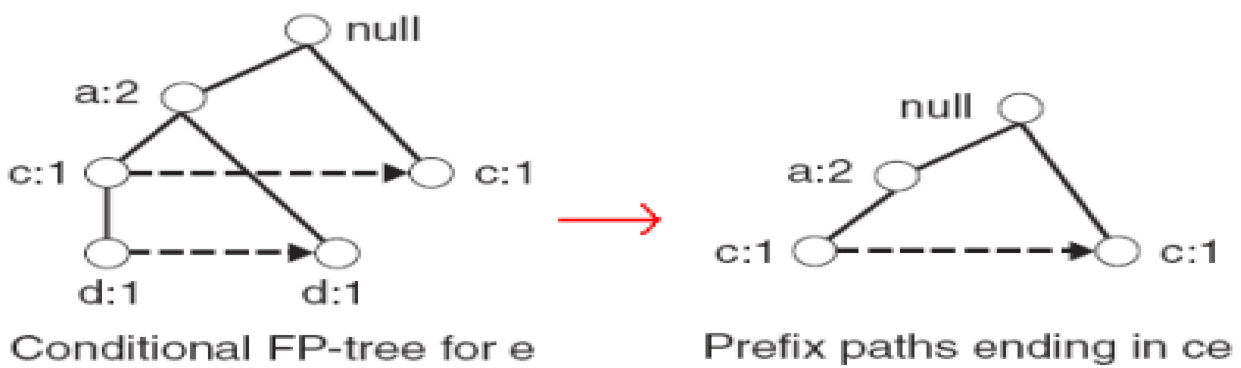
- 4. Use the the conditional FP-tree for e to find frequent itemsets ending in de, ce and ae
  - Note that be is not considered as b is not in the conditional FP-tree for e.
- I For each of them (e.g. de), find the prefix paths from the conditional tree for e, extract frequent itemsets, generate conditional FP-tree, etc... (recursive)

Example

- Example: e -> de -> ade ({d,e}, {a,d,e} are found to be frequent)



- Example: e -> ce ({c,e} is found to be frequent)



Result

Frequent itemsets found (ordered by sufix and order in which they are found):

Transaction Data Set

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

Suffix	Frequent Itemsets
e	{e}, {d,e}, {a,d,e}, {c,e},{a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}

Discusion

- Advantages of FP-Growth
  – only 2 passes over data-set
  – “compresses” data-set
  – no candidate generation
  – much faster than Apriori
- Disadvantages of FP-Growth
  – FP-Tree may not fit in memory!!
  – FP-Tree is expensive to build

References

- [1] Pang-Ning Tan, Michael Steinbach, Vipin Kumar:*Introduction to Data Mining*, Addison-Wesley
- www.wikipedia.org