FP-Growth Algorithm – Overview

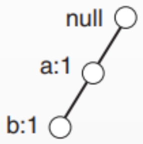
- Apriori requires one pass for each *k* (2+ on first pass for PCY variants)
- Can we find ***all*** frequent item sets in fewer passes over the data?

FP-Growth Algorithm:

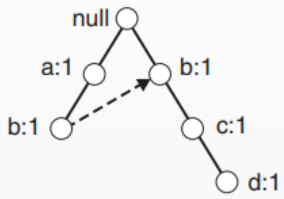
- *Pass 1*: Count items with support ≥ s
- Sort frequent items in descending order according to count
- *Pass 2*: Store all frequent itemsets in a frequent pattern tree (FP-tree)
- Mine patterns from FP-Tree

FP-Tree Construction

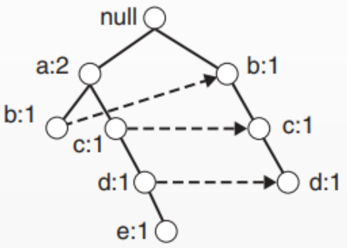
TID = 1



TID = 2

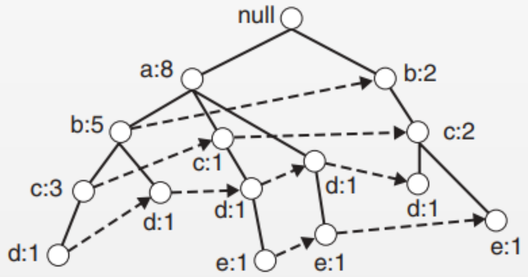


TID = 3



TID	Items Bought	Frequent Items
1	{a,b,f}	{a,b}
2	{b,g,c,d}	{b,c,d}
3	{h, a,c,d,e}	{a,c,d,e}
4	{a,d, p,e}	{a,d,e}
5	{a,b,c}	{a,b,c}
6	{a,b,q,c,d}	{a,b,c,d}
7	{a}	{a}
8	{a,m,b,c}	{a,b,c}
9	{a,b,n,d}	{a,b,d}
10	{b,c,e}	{b,c,e}

TID = 10

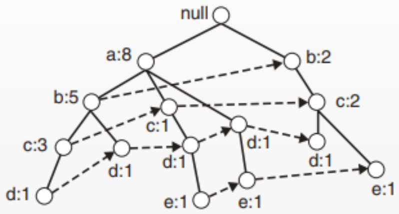


a: 8, b: 7, c: 6, d: 5, e: 3,
~~f: 1~~, ~~g: 1~~, ~~h: 1~~, ~~m: 1~~, ~~n: 1~~

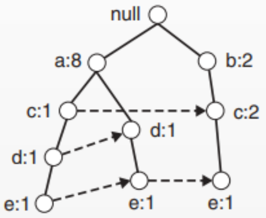
Mining Patterns from the FP-Tree

Step 1: Extract subtrees ending in each item

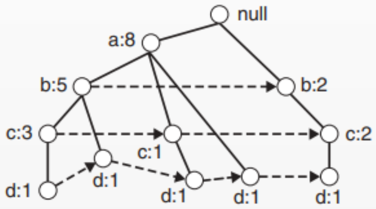
Full Tree



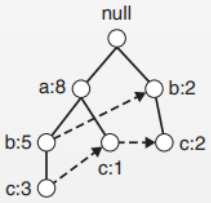
Subtree *e*



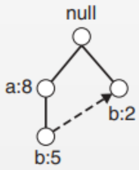
Subtree *d*



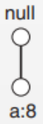
Subtree *c*



Subtree *b*



Subtree *a*

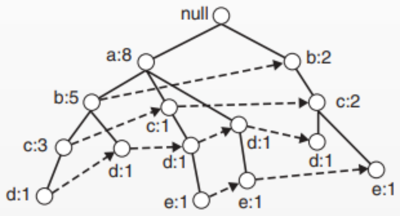


a: 8, b: 7, c: 6, d: 5, e: 3, ~~f: 1~~, ~~g: 1~~, ~~h: 1~~, ~~m: 1~~, ~~n: 1~~

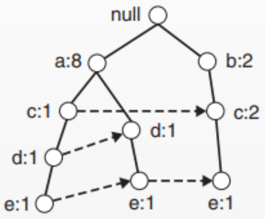
Mining Patterns from the FP-Tree

Step 2: Construct Conditional FP-Tree for each item

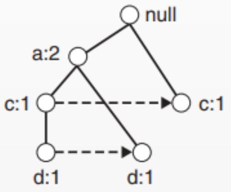
Full Tree



Subtree *e*



Conditional *e*



Conditional Pattern Base for e
acd: 1, ad: 1, bc: 1

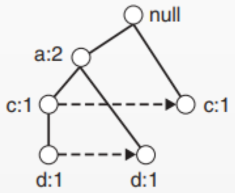
Conditional Node Counts
a: 2, ~~b: 1~~, c: 2, d: 2

- Calculate counts for paths ending in ***e***
- Remove leaf nodes
- Prune nodes with count ≤ ***s***

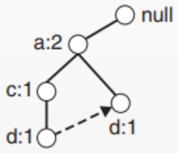
Mining Patterns from the FP-Tree

Step 3: Recursively mine conditional FP-Tree for each item

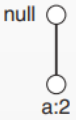
Conditional *e*



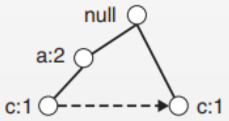
Subtree *de*



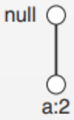
Conditional *de*



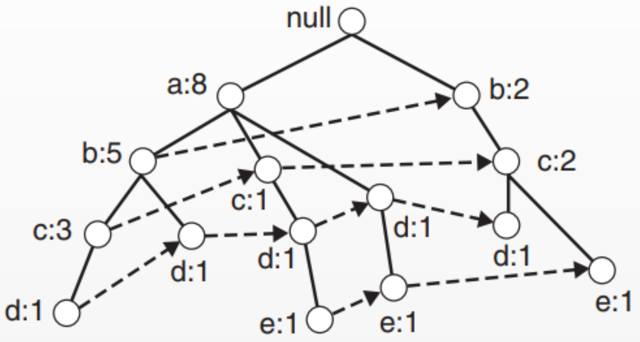
Subtree *ce*



Subtree *ae*



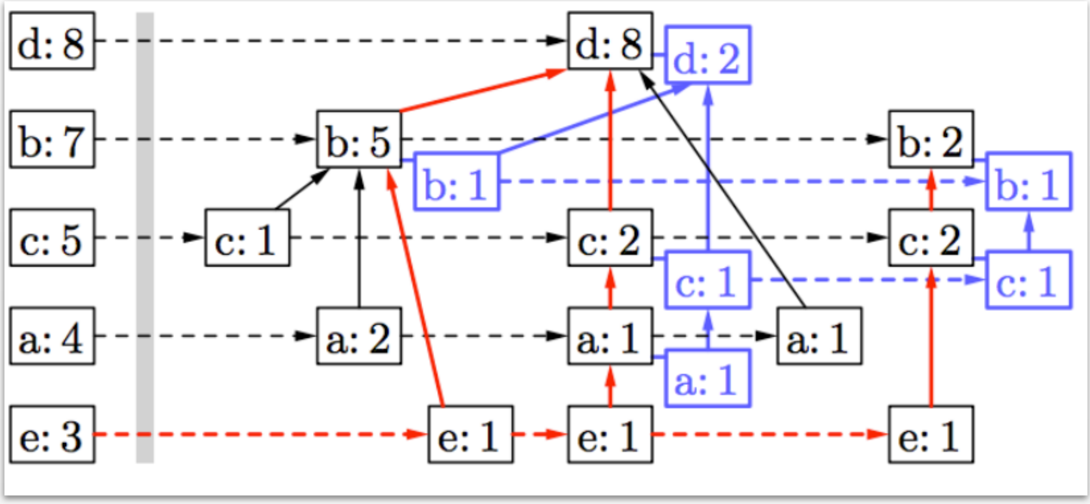
Mining Patterns from the FP-Tree



Suffix	Conditional Pattern Base
e	acd:1; ad:1; bc:1
d	abc:1; ab:1; ac:1; a:1; bc:1
c	ab:3; a:1; b:2
b	a:5
a	ϕ

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}

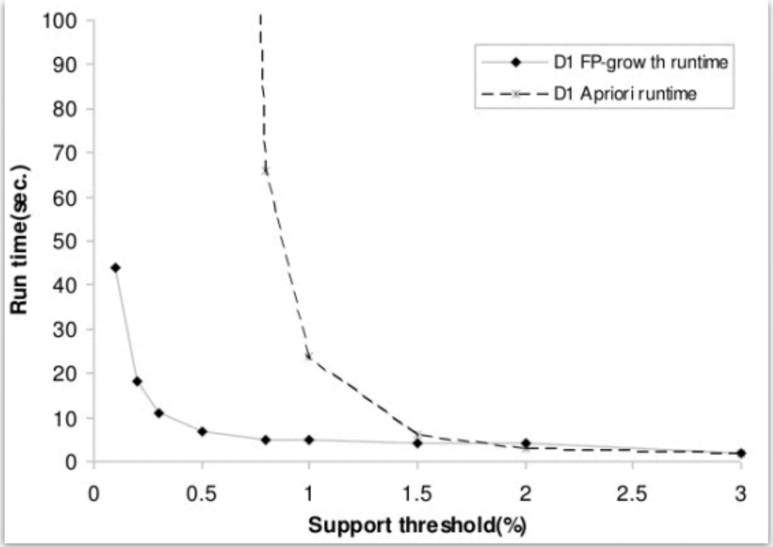
Projecting Sub-trees



- “Cutting” and “pruning” trees requires that we create copies/mirrors of the subtrees
- Mining patterns requires additional memory

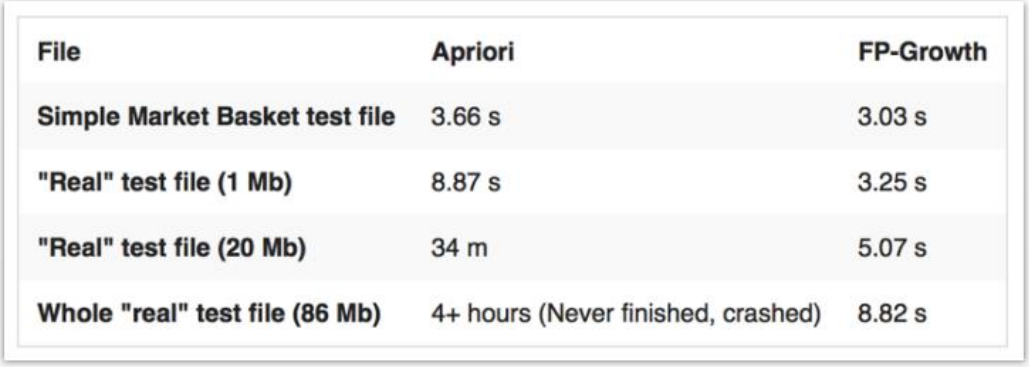
FP-Growth vs Apriori

Simulated data 10k baskets, 25 items on average



(from: Han, Kamber & Pei, Chapter 6)

FP-Growth vs Apriori



File	Apriori	FP-Growth
Simple Market Basket test file	3.66 s	3.03 s
"Real" test file (1 Mb)	8.87 s	3.25 s
"Real" test file (20 Mb)	34 m	5.07 s
Whole "real" test file (86 Mb)	4+ hours (Never finished, crashed)	8.82 s

FP-Growth vs Apriori

Advantages of FP-Growth

- Only 2 passes over dataset
- Stores "compact" version of dataset
- No candidate generation
- Faster than A-priori

Disadvantages of FP-Growth

- The FP-Tree may not be "compact" enough to fit in memory
- Even more memory required to construct subtrees in mining phase