BITS Pilani

BITS Pilani
Hyderabad Campus

Prof.Aruna Malapati
Department of CSIS

innovate achieve lead

BITS Pilani
Hyderabad Campus

ECLAT and FP Growth Algorithm

Today's Agenda

- Challenges of Apriori Algorithm
- Horizontal/Vertical representation of Databases
- ECLAT Algorithm
- Frequent Pattern Growth Tree
- Construct FP-tree from a Transaction DB

Challenges of Apriori Algorithm

- Challenges
  - Multiple scans of the transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: the general ideas
  - Reduce the number of transaction database scans
  - Shrink the number of candidates
  - Facilitate support counting of candidates

Horizontal representation of Databases

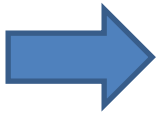
- Transactions are generally stored in horizontal (Row) format .

TID	Items
1	A B E
2	B C D
3	C E
4	A C D
5	A B C D
6	A E
7	A B
8	A B C
9	A C D
10	B

Vertical representation of Databases

- Column representation is beneficial for depth-first search
- Join operations to intersect columns and $k-1$-itemsets to k-itemsets
- Columns can be compressed well (c.f., column store databases, inverted indexes)

A	B	C	D	E
1	1	2	2	1
4	2	3	4	3
5	5	4	5	6
6	7	8	9	
7	8	9		
8	10			
9				



A	B	C	D	E
1-1	1-2	2-4	2-2	1-1
4-9	5-5	8-9	4-5	3-3
	7-8		9-9	6-6
	10-10			

Eclat Algorithm

- Apriori used a breath-first search, and stored the data in rows; Eclat uses a depth-first search.
- Eclat uses a columnar database $C=\{(X,T_X)\}$ of items with TID lists, and recursion on partitions with a shared itemset prefix (initially: empty prefix).

Eclat Algorithm (Contd..)

innovate achieve lead

Algorithm: Eclat($I, C, minsupp$)

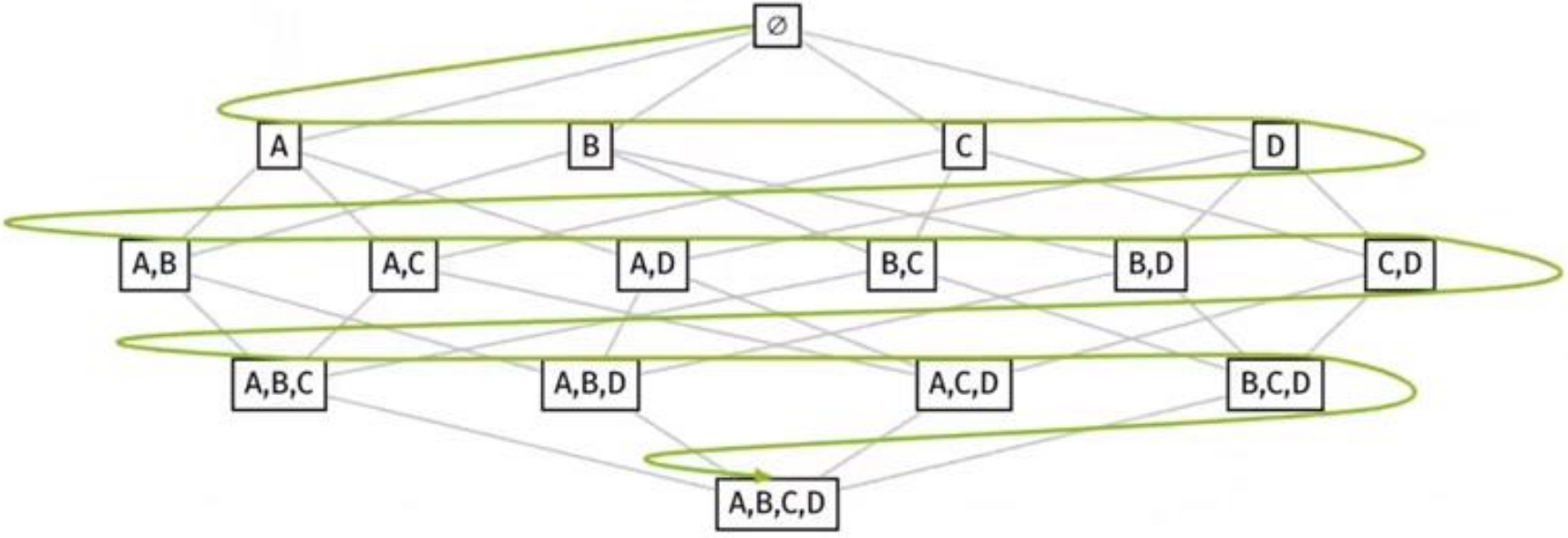
```
1 ℐ ← {(X, T_X) ∈ C | |T_X| ≥ minsupp}        // columns with minimum support
2 EclatRecursive(ℐ, minsupp)
```

Algorithm: EclatRecursive($\mathcal{I}, minsupp$)

```
1 foreach (X, T_X) ∈ ℐ do                          // every tid-list in the partition
2     ℐ_X ← ∅
3     foreach (Y, T_Y) ∈ ℐ with Y > X do             // and every later tid-list
4         if |T_X ∩ T_Y| ≥ minsupp then             // check support of intersection
5             ℐ_X ← ℐ_X + (X ∪ Y, T_X ∩ T_Y)       // add to the new partition
6     if ℐ_X ≠ ∅ then EclatRecursive(ℐ, minsupp)   // recurse into new partition
```

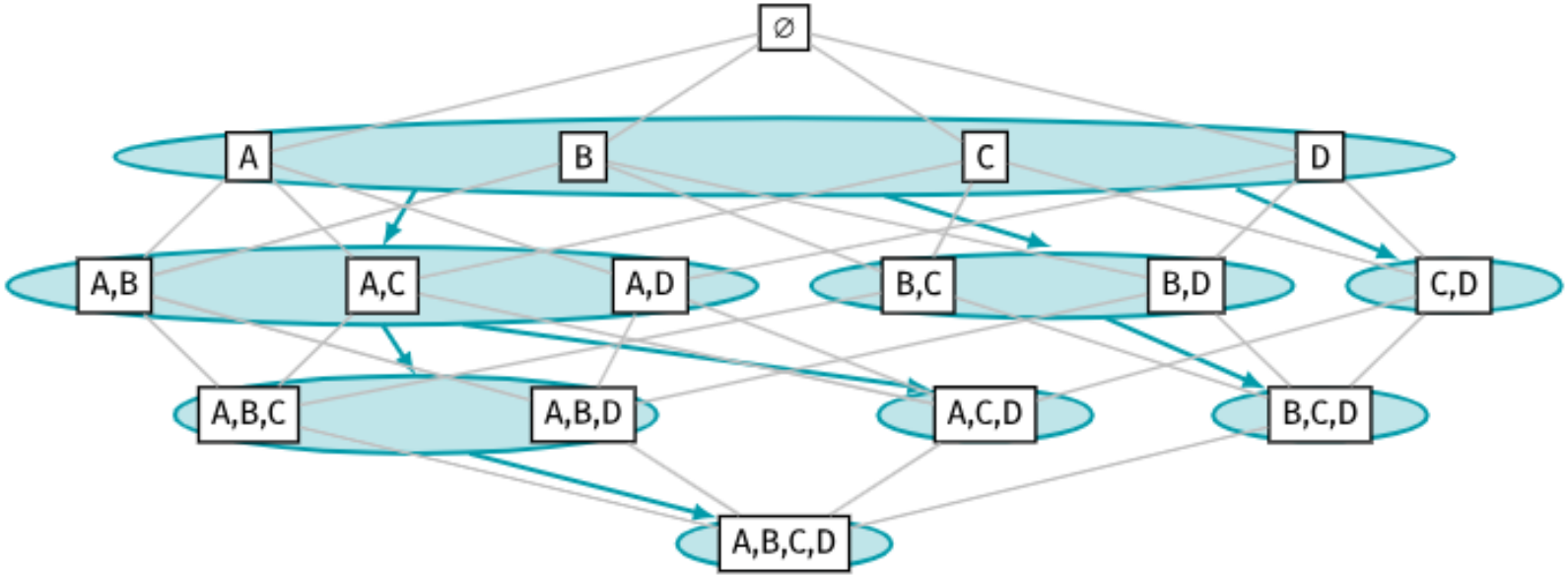
Candidate Processing Order

- Apriori uses a breadth-first search.



Candidate Processing Order(Contd..)

Eclat uses a prefix-partitioned depth-first.



Eclat Example with *minsupp*=50%

Item	TIDlist
A	1 2 3 4 7 9 10
B	1 2 3 4 5 6 7 9 10
C	1 5 7 8
D	1 3 4 5 6 8 9 10
E	1 2 3 5 6 7

C is eliminated.

Initial class:

A	1 2 3 4 7 9 10	OK
B	1 2 3 4 5 6 7 9 10	OK
D	1 3 4 5 6 8 9 10	OK
E	1 2 3 5 6 7	OK

1-Partition A:

A B	1 2 3 4 7 9 10	OK
A D	1 3 4 9 10	OK
A E	1 2 3 7	X

2-Partition A B:

A B D	1 3 4 9 10	OK

1-Partition B:

B D	1 3 4 5 6 9 10	OK
B E	1 2 3 5 6 7	OK

2-Partition B D:

B D E	1 3 5 6	X

D E	1 3 5 6	X

Note: Eclat also uses the prefix join idea of Apriori, but cannot use pruning (c.f., BDE).

Bottleneck of Frequent-pattern Mining

- Multiple database scans are costly
- Mining long patterns needs many passes of scanning and generates lots of candidates
  - To find frequent itemset $i_1 i_2 \ldots i_{100}$
    - # of scans: 100
    - # of Candidates: $\binom{100}{1} + \binom{100}{2} + \ldots + \binom{1}{1}\binom{0}{0}\binom{0}{0} = 2^{100}-1 = 1.27*10^{30}$!
- Bottleneck: candidate-generation-and-test
- Can we avoid candidate generation?

Mining Freq Patterns w/o Candidate Generation

- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  - Highly condensed, but complete for frequent pattern mining
  - Avoid costly database scans
- Develop an efficient, FP-tree-based frequent pattern mining method
  - A divide-and-conquer methodology: decompose mining tasks into smaller ones
  - Avoid candidate generation: examine sub-database (conditional pattern base) only!

Frequent Pattern Growth Tree

- Many itemsets are duplicate or similar
- A prefix-tree-like aggregation can exploit redundancy to conserve memory
- The tree can often be held in main memory even when the database cannot
- Find frequent patterns from the aggregated tree via projection
- Find association rules without generating all frequent itemsets

Construction of an FP-tree

The FP-tree is constructed in two phases:

First database scan:

- determine frequent 1-itemsets (based on a given $minsupp$ parameter)
- sort items by descending frequency (to get smaller trees later)

Second database scan:

- read one transaction at a time
- omit non-frequent items from the transaction
- sort items based on support counts in decreasing order
- insert into the FP-tree

Construct FP-tree from a Transaction DB

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

min_sup= 3

Steps:

1. Scan DB once, find frequent 1-itemset (single item pattern)
2. Order frequent items in frequency descending order: f, c, a, b, m, p (L-order)
3. Process DB based on L-order

a	3	~~i~~	~~1~~
b	3	~~j~~	~~1~~
c	4	~~k~~	~~1~~
~~d~~	~~1~~	~~l~~	~~2~~
~~e~~	~~1~~	m	3
f	4	~~n~~	~~1~~
~~g~~	~~1~~	o	~~2~~
~~h~~	~~1~~	p	3

Construct FP-tree from a Transaction DB (Contd..)

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

{}

Header Table

Item	frequency	head
f	0	nil
c	0	nil
a	0	nil
b	0	nil
m	0	nil
p	0	nil

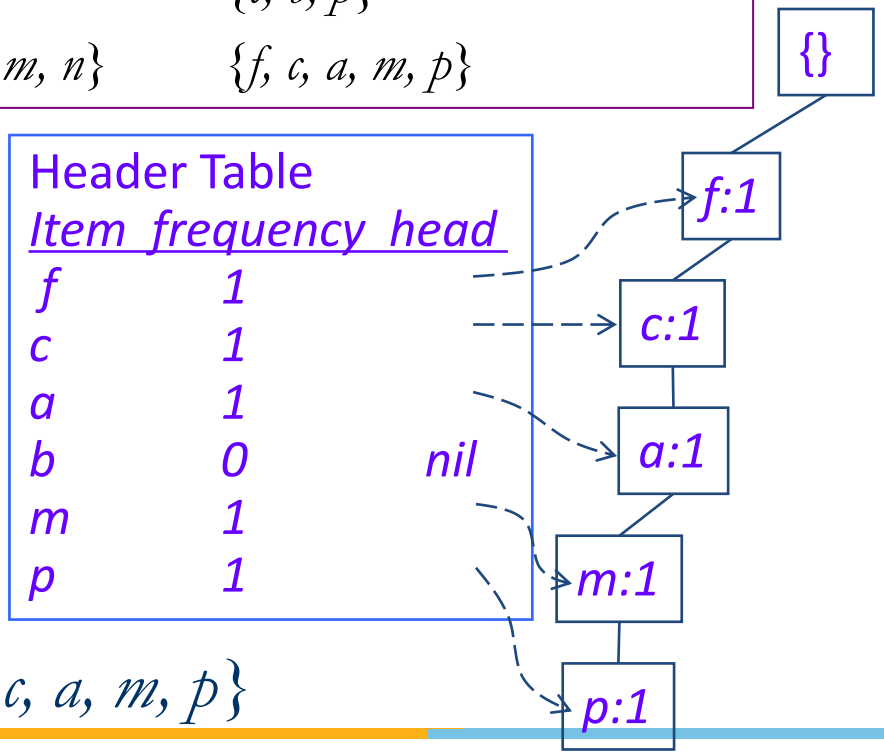
Initial FP-tree

Construct FP-tree from a Transaction DB (Contd..)

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

Header Table

Item	frequency	head
f	1	
c	1	
a	1	
b	0	nil
m	1	
p	1	



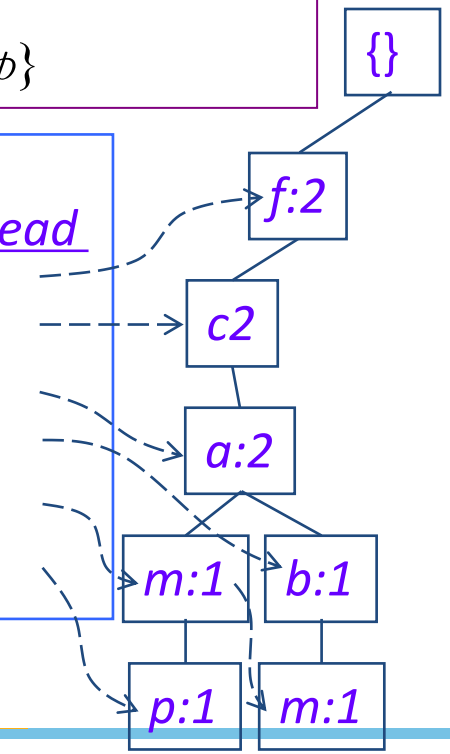
Insert {f, c, a, m, p}

Construct FP-tree from a Transaction DB (Contd..)

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

Header Table

Item	frequency	head
f	2	
c	2	
a	2	
b	1	
m	2	
p	1	



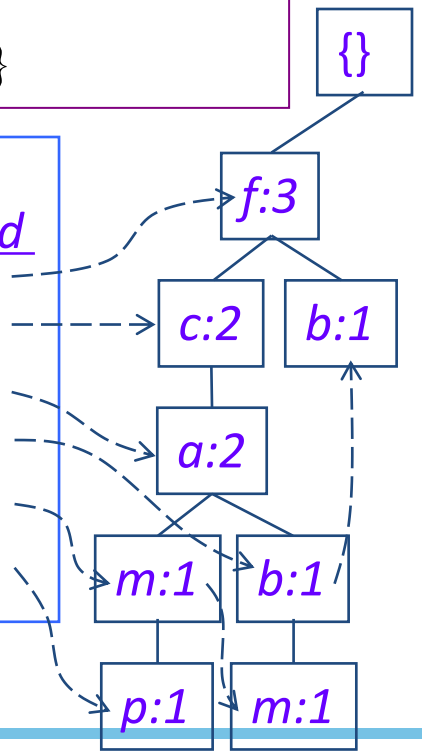
Insert {f, c, a, b, m}

Construct FP-tree from a Transaction DB (Contd..)

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

Header Table

Item	frequency	head
f	3	
c	2	
a	2	
b	2	
m	2	
p	1	



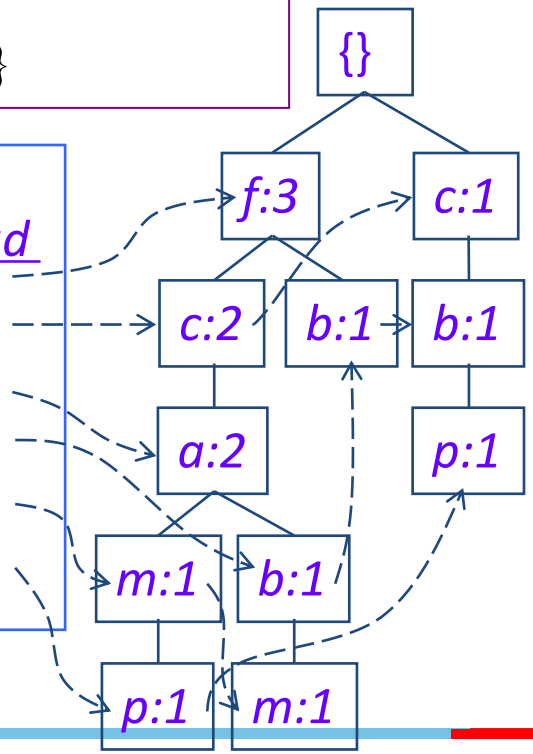
Insert {f, b}

Construct FP-tree from a Transaction DB (Contd..)

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

Header Table

Item	frequency	head
f	3	
c	3	
a	2	
b	3	
m	2	
p	2	



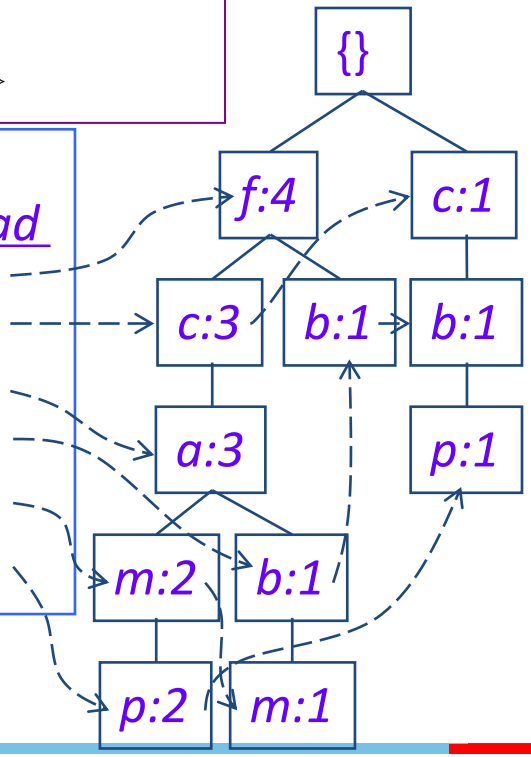
Insert {c, b, p}

Construct FP-tree from a Transaction DB (Contd..)

TID	Items bought	(ordered) frequent items
100	{f, a, c, d, g, i, m, p}	{f, c, a, m, p}
200	{a, b, c, f, l, m, o}	{f, c, a, b, m}
300	{b, f, h, j, o}	{f, b}
400	{b, c, k, s, p}	{c, b, p}
500	{a, f, c, e, l, p, m, n}	{f, c, a, m, p}

Header Table

Item	frequency	head
f	4	
c	4	
a	3	
b	3	
m	3	
p	3	

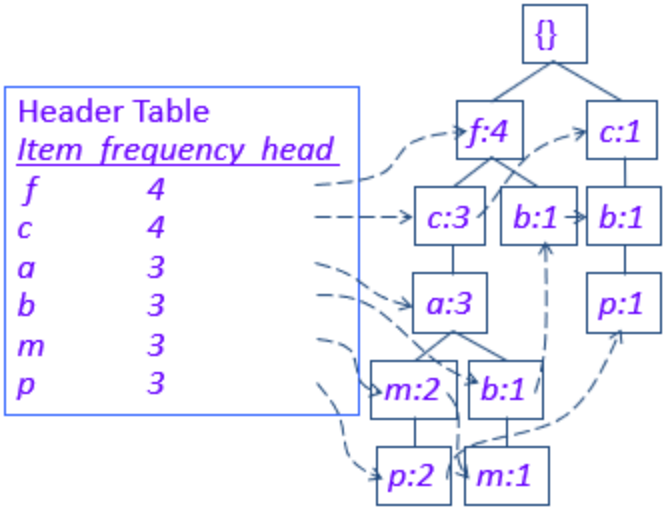


Insert {f, c, a, m, p}

Conditional Pattern Base

- For each item, the **Conditional Pattern Base** is computed which is the path labels of all the paths which lead to any node of the given item in the frequent-pattern tree.

Conditional Pattern Base (Contd..)



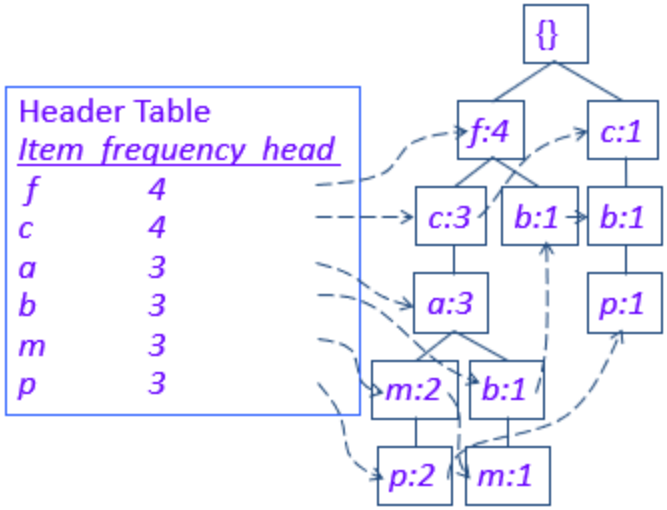
Item	Increasing Order of Frequency	Conditional base pattern
p	3	{{f, c, a, m : 2}, {c, b : 1}}
m	3	{{f, c, a : 2}, {f, c, a, b : 1}}
b	3	{{f, c, a : 1}, {f : 1}, {c : 1}}
a	3	{f, c : 3}
c	4	{f : 3}
f	4	Φ

Conditional Frequent Pattern Tree

- For each item, the Conditional Frequent Pattern Tree is built.

- It is done by taking the set of elements in all the paths of the Conditional Pattern Base of that item and calculating its support count by summing the support counts of all the paths in the Conditional Pattern Base.

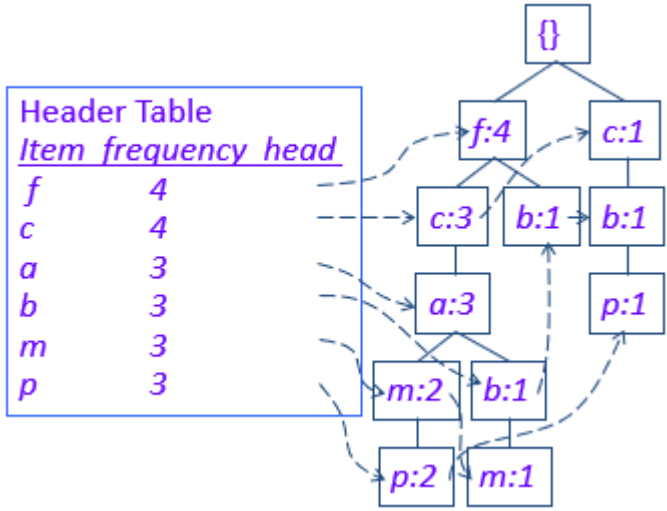
Conditional Frequent Pattern Tree (Contd..)



Item	Increasing Order of Frequency	Conditional base pattern	Conditional FP Tree
p	3	{{f, c, a, m : 2}, {c, b : 1}}	c:3
m	3	{{f, c, a : 2}, {f, c, a, b : 1}}	f:3,c:3,a:3
b	3	{{f, c, a : 1}, {f : 1}, {c : 1}}	Φ
a	3	{f, c : 3}	f:3,c:3
c	4	{f : 3}	f:3
f	4	Φ	

Frequent Patterns Generation

- From the Conditional Frequent Pattern tree, the Frequent Patterns are generated by pairing the Conditional Frequent Pattern Tree items set to the corresponding item.



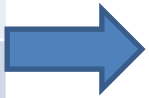
Item	Increasing Order of Frequency	Conditional base pattern	Conditional FP Tree	Frequent Patterns
p	3	{{f, c, a, m : 2}, {c, b : 1}}	c:3	cp:3
m	3	{{f, c, a : 2}, {f, c, a, b : 1}}	f:3,c:3,a:3	fm:3,cm:3,am:3,fcm:3, fam:3,cam:3,fcam:3
b	3	{{f, c, a : 1}, {f : 1}, {c : 1}}	Φ	Φ
a	3	{f, c : 3}	f:3,c:3	f a : 3,c a : 3,fca:3
c	4	{f : 3}	f:3	fc:3
f	4	Φ	Φ	Φ

Association rule Generation

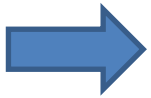
- For each Frequent Pattern generated, extract all possible rules.
- Association rules greater than the confidence will be retained as interesting rules.

Example - 2

TID	List of Items
T100	I1,I2,I5
T200	I2,I4
T300	I2,I3
T400	I1,I2,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I1,I2,I3,I5
T900	I1,I2,I3



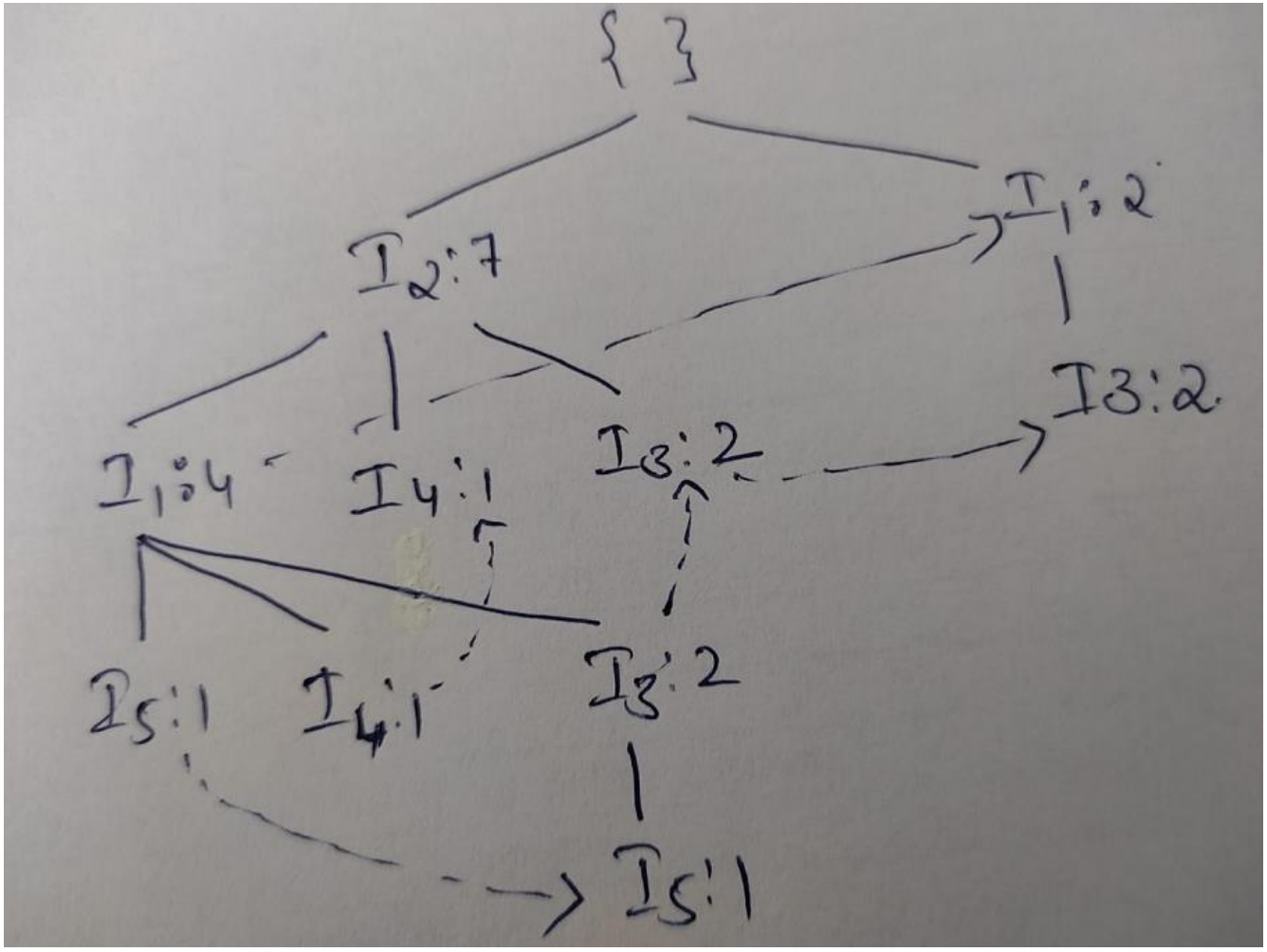
Item	Support Count
I2	7
I1	6
I3	6
I4	2
I5	2

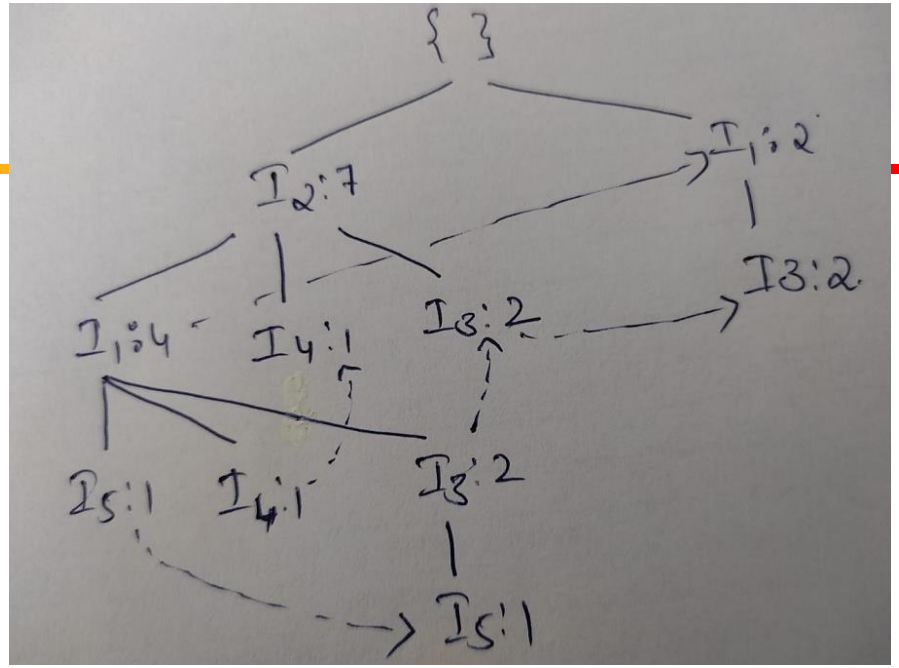


TID	List of Items
T100	I2,I1,I5
T200	I2,I4
T300	I2,I3
T400	I2,I1,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I2,I1,I3,I5
T900	I2,I1,I3

min_sup= 2

TID	List of Items
T100	I2,I1,I5
T200	I2,I4
T300	I2,I3
T400	I2,I1,I4
T500	I1,I3
T600	I2,I3
T700	I1,I3
T800	I2,I1,I3,I5
T900	I2,I1,I3





Item	Conditional Pattern Base	Conditional FP-Tree	Frequent Patterns
I5	{I2 I1:1} ,{I2 I1 I3:1}	I2:2,I1:2	{I2 I5:2} {I1 I5:2} {I2 I1 I5:2}
I4	{I2 I1:1}, {I2:1}	I2:2	{I2 I4:2}
I3	{I2 I1:2}, {I2:2} {I1:2}	I2:4,I1:2 I1:2	{I2 I3:4} {I1 I3:4} {I2 I1 I3:2}
I1	{I2:4}	I2:4	{I2 I1:4}
I2	Φ	Φ	Φ

Why Is Frequent Pattern Growth Fast?

- Performance study shows
  - FP-growth is an order of magnitude faster than Apriori
- Reasoning
  - No candidate generation, no candidate test
  - Use a compact data structure
  - Eliminate repeated database scan
  - Basic operations are counting and FP-tree building

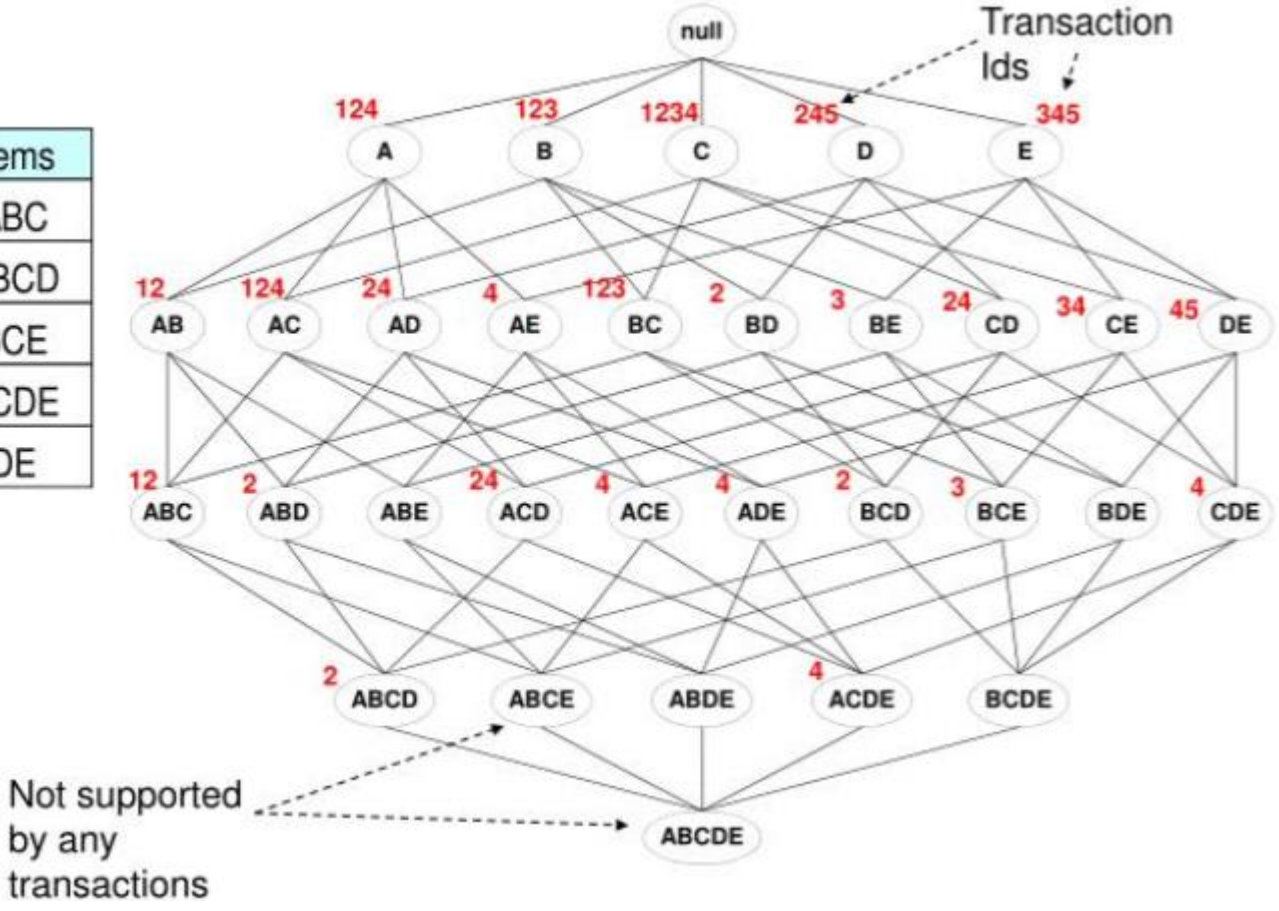
Compact Representation of Frequent Itemsets

- The number of frequent itemsets produced from a transaction data set can be very large.
- It is useful to identify a small representation set of itemsets from which all other frequent itemsets can be derived.
- Maximal frequent itemsets
  - No immediate superset is frequent
- Closed itemsets
  - No immediate superset has the same count
  - Stores not only frequent information but exact counts

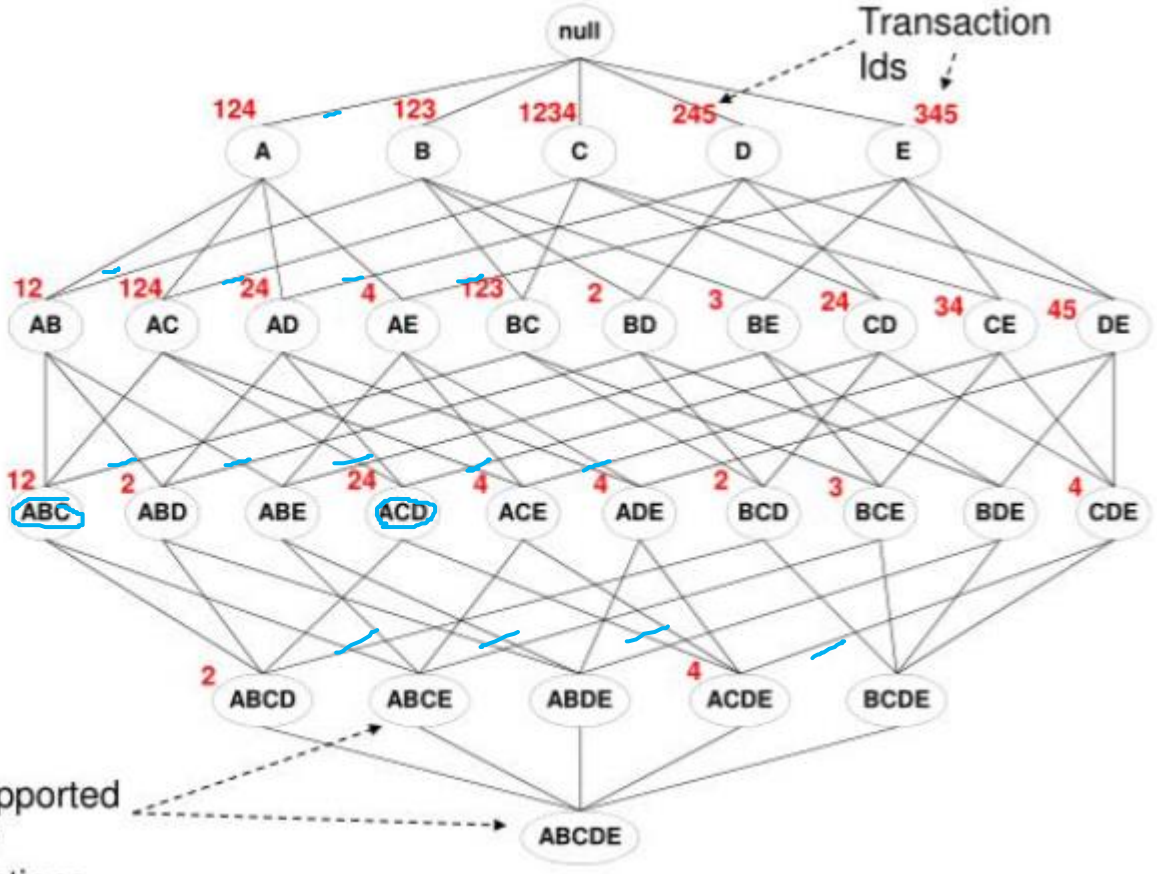
Maximal Frequent Itemsets

- The Apriori and FP growth tree algorithms generate very large Frequent Item sets.
- A frequent itemset X is Maximal if there is no frequent itemset Y such that $X \subseteq Y$. i.e frequent itemset X is *maximal* if none of its supersets is frequent.
- Maximal frequent itemsets do not contain the support information of their subsets.

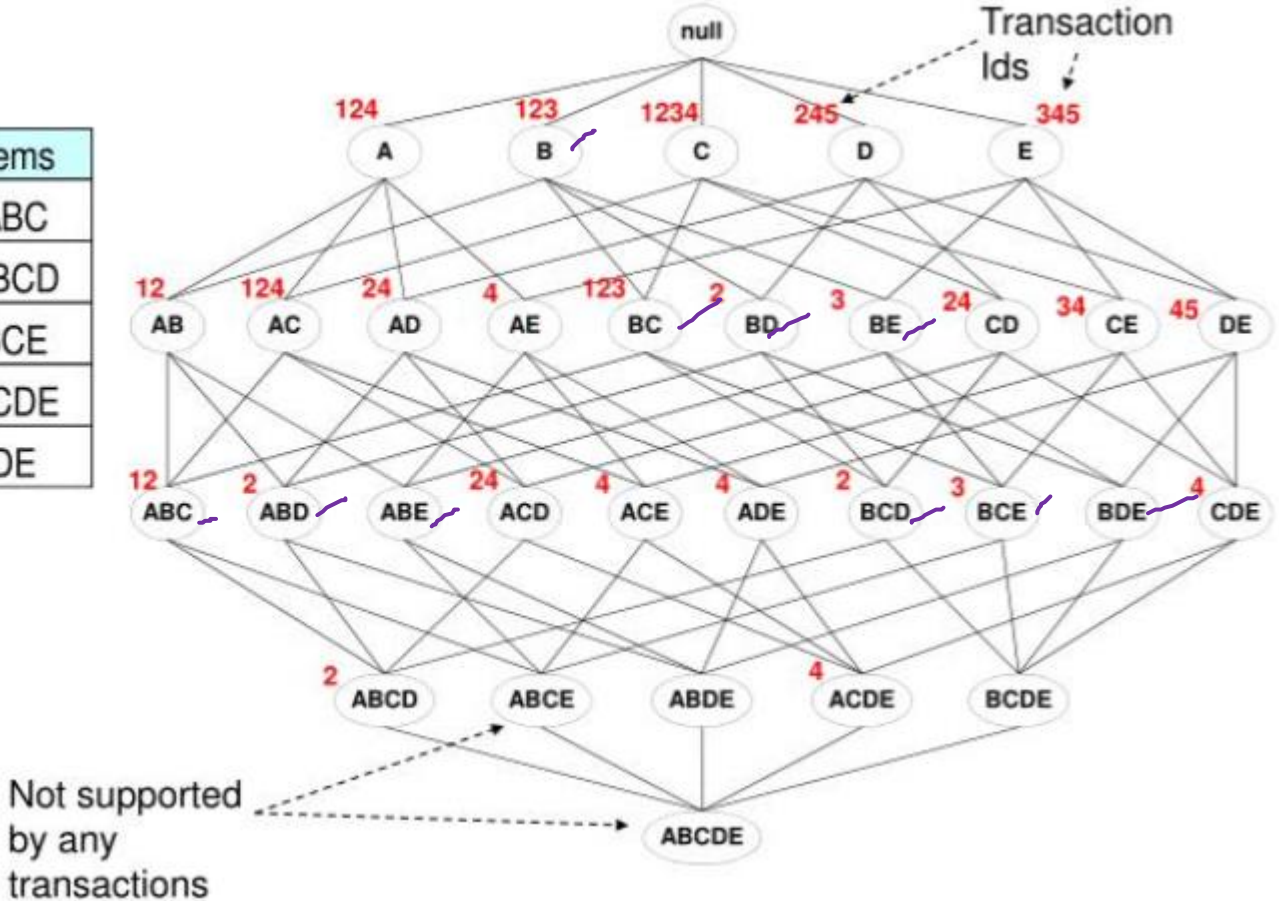
TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



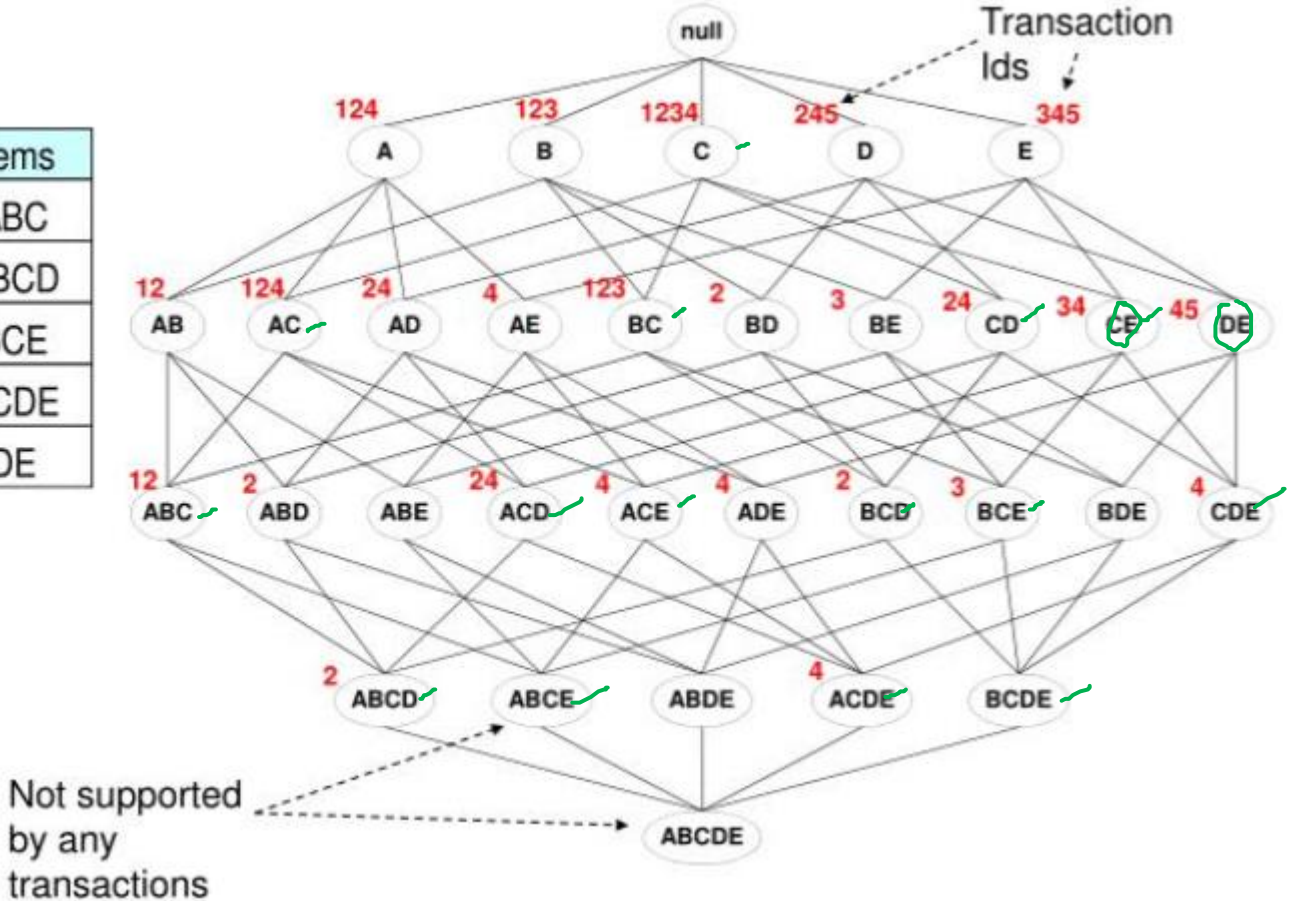
TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



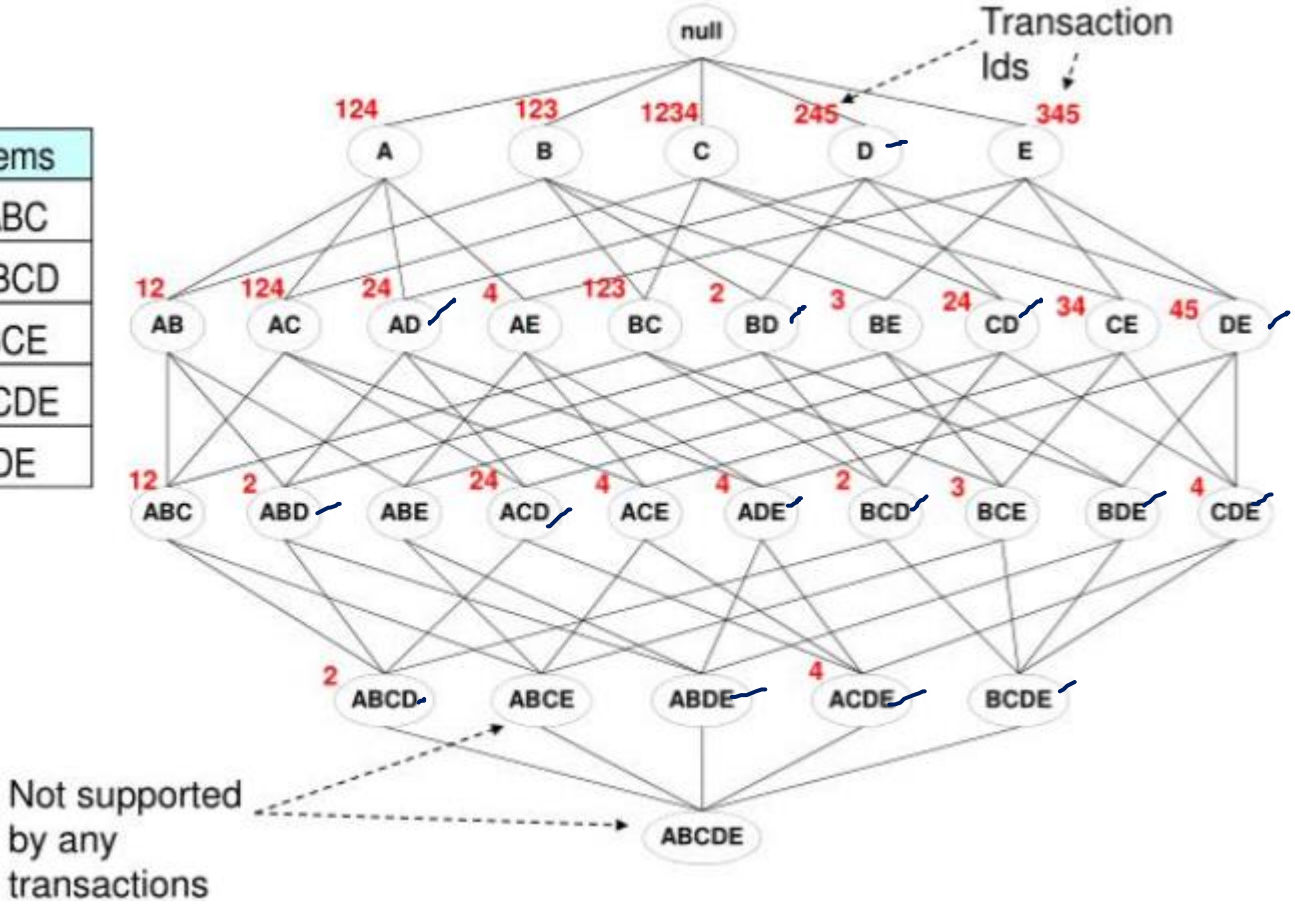
TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



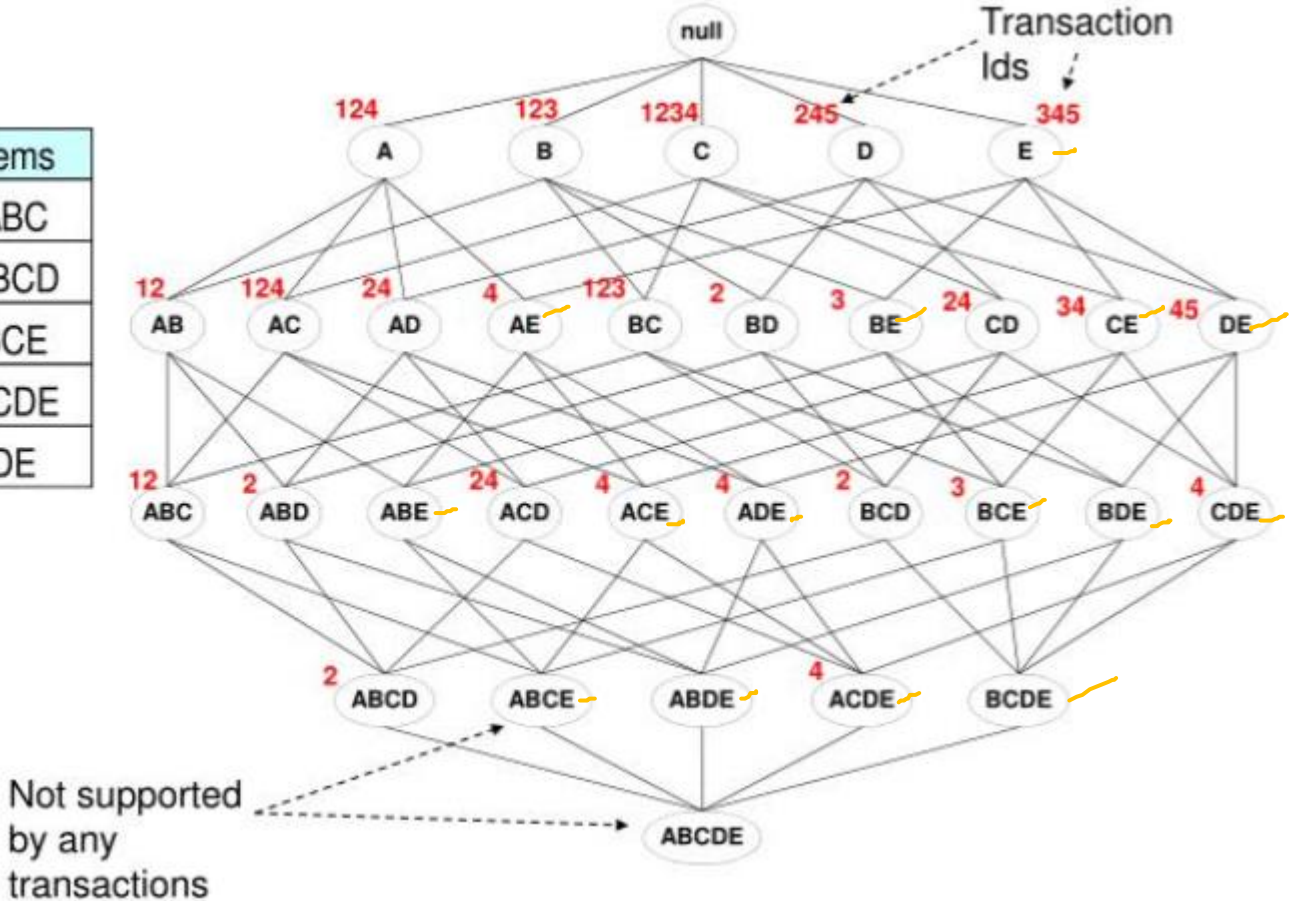
TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



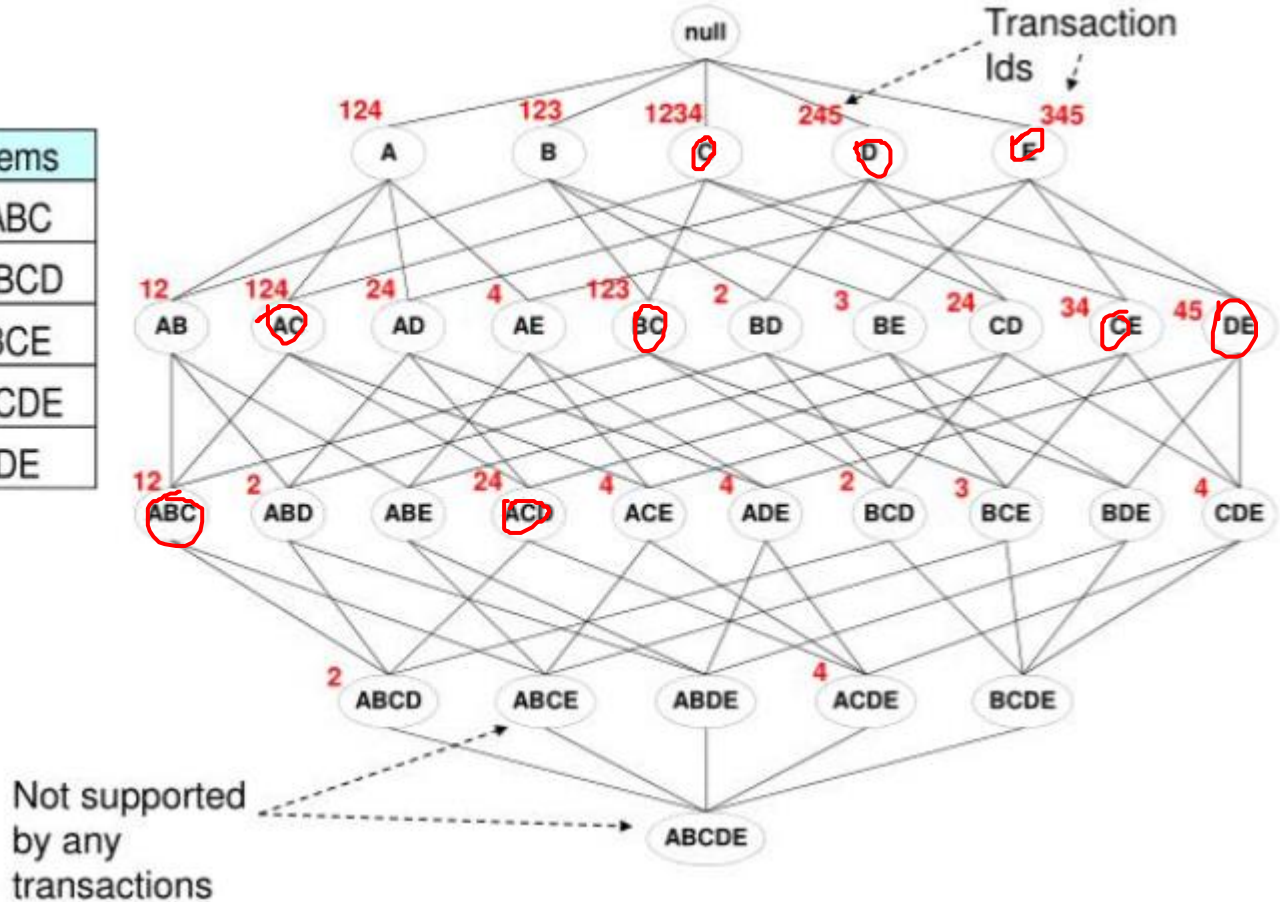
TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



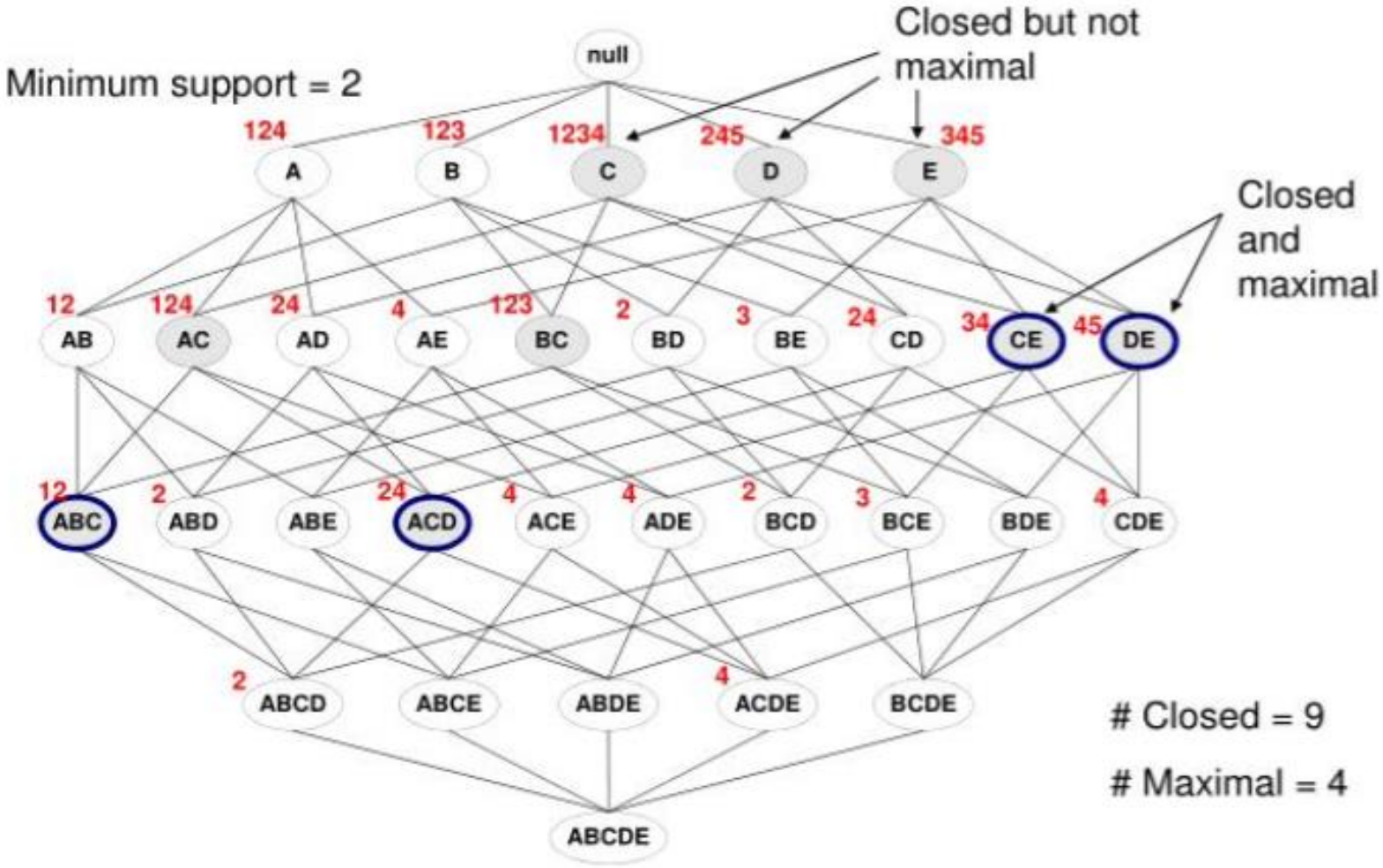
Closed Frequent Itemsets

- An itemset is closed frequent if it is closed and its support is greater than or equal to minsup.

TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



Maximal VS Closed Frequent itemsets



How to find support of frequent items from closed itemset?

- To find support for any other frequent itemset X from the closed itemset

  1. Find closure(X) : The closure of an itemset X is the smallest closed itemset Y such that X is a subset of Y. denoted as c(X).

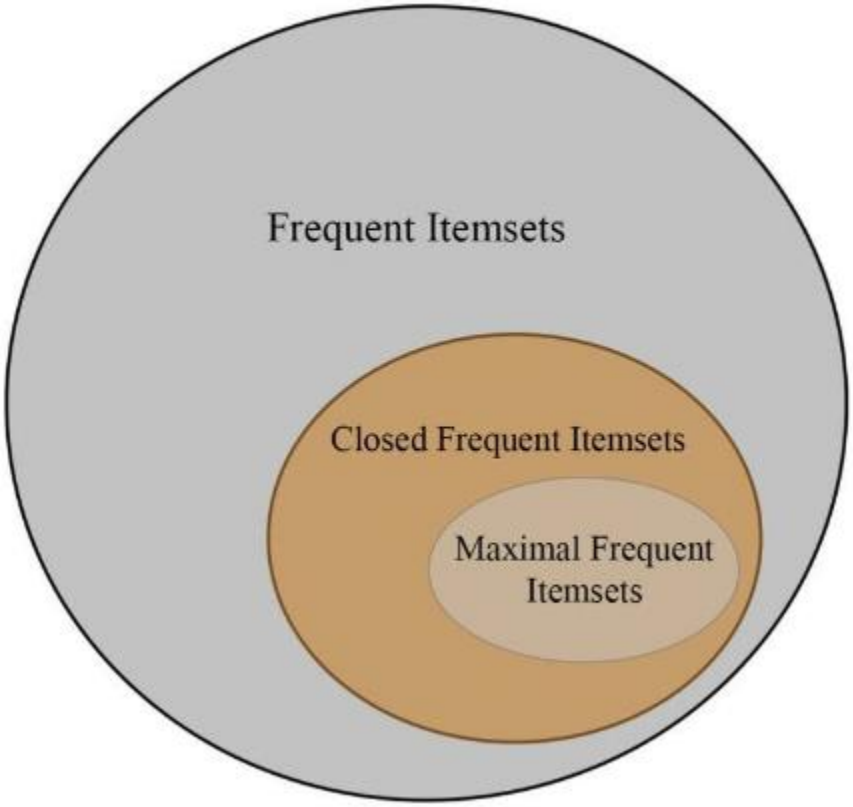
  2. Return the support of the closure of X.

How to find support of frequent items from closed itemset?

- For example, using the closed itemsets, you want to find support for CD, which is a frequent item.
- The closure of C(CD) = {ACD} = 2

List of Closed itemset and their support
C:4
D:3
E:3
AC:3
BC:3
CE:2
DE:2
ABC:2
ACD:2

Frequent, closed frequent itemsets and maximal frequent itemsets



Evaluation of Association Patterns

- Association rule algorithms tend to produce too many rules
  - Many of them are uninteresting or redundant.
  - Redundant if {A,B,C} → {D} and {A,B} → {D} have same support & confidence.
- In the original formulation of association rules, support & and confidence are the only measures used.

Objective Vs Subjective evaluation of Association rules

- Objective measures:
  - Support
  - Confidence
  - Correlation
- Subjective
  - It is unexpected (surprising to the user); and/or
  - actionable (the user can do something with it)

Criticism to Support and Confidence

Example 1:

- Among 5000 students
  - 3000 play basketball
  - 3750 eat cereal
  - 2000 both play basket ball and eat cereal
- *play basketball* $\Rightarrow$ *eat cereal* [40%, 66.7%]

  support=2000/5000,confidence=2000/3000
- *Not play basketball* $\Rightarrow$ *eat cereal* [35%, 87.5%]

  support=1750/5000,confidence=1750/2000

	basketball	not basketball	sum(row)
cereal	2000	1750	3750
not cereal	1000	250	1250
sum(col.)	3000	2000	5000

Criticism to Support and Confidence (Cont.)

	basketball	not basketball	sum(row)
cereal	2000	1750	3750
not cereal	1000	250	1250
sum(col.)	3000	2000	5000

- Measure of dependent/correlated events:
- Lift(B,C) = C(B -> C) / S(C) =S(B U C)/S(B) x S(C)
- Lift(B,C) may tell how B and C are Correlated
- Lift(B,C) =1: B and C are independent

  >1 Positively correlated

  <1 negatively correlated

Lift(play basketball , eat cereal) = (2000/5000)/(3000/5000)(3750/5000) =0.89
Lift(Not play basketball , eat cereal) = (1750/5000)/(2000/5000)(3750/5000) =1.16

Take home message

- ECLAT uses a column database to reduce the the effort of support counting.
- Compress a large database into a compact, Frequent-Pattern tree (FP-tree) structure
  - Highly condensed, but complete for frequent pattern mining
  - Avoid costly database scans

References

- Zaki, M.J. 2000. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.* 12, 3 (2000), 372–390. DOI:10.1109/69.846291
- Borgelt, C. 2005. An implementation of the FP-growth algorithm. *Proceedings of the 1st international workshop on open source data mining: Frequent pattern mining implementations* (2005), 1–5.
- Han, J., Pei, J. and Yin, Y. 2000. Mining frequent patterns without candidate generation. *SIGMOD* (2000), 1–12.