HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

582364 Data mining, 4 cu
Lecture 4: Finding frequent itemsets - concepts and algorithms

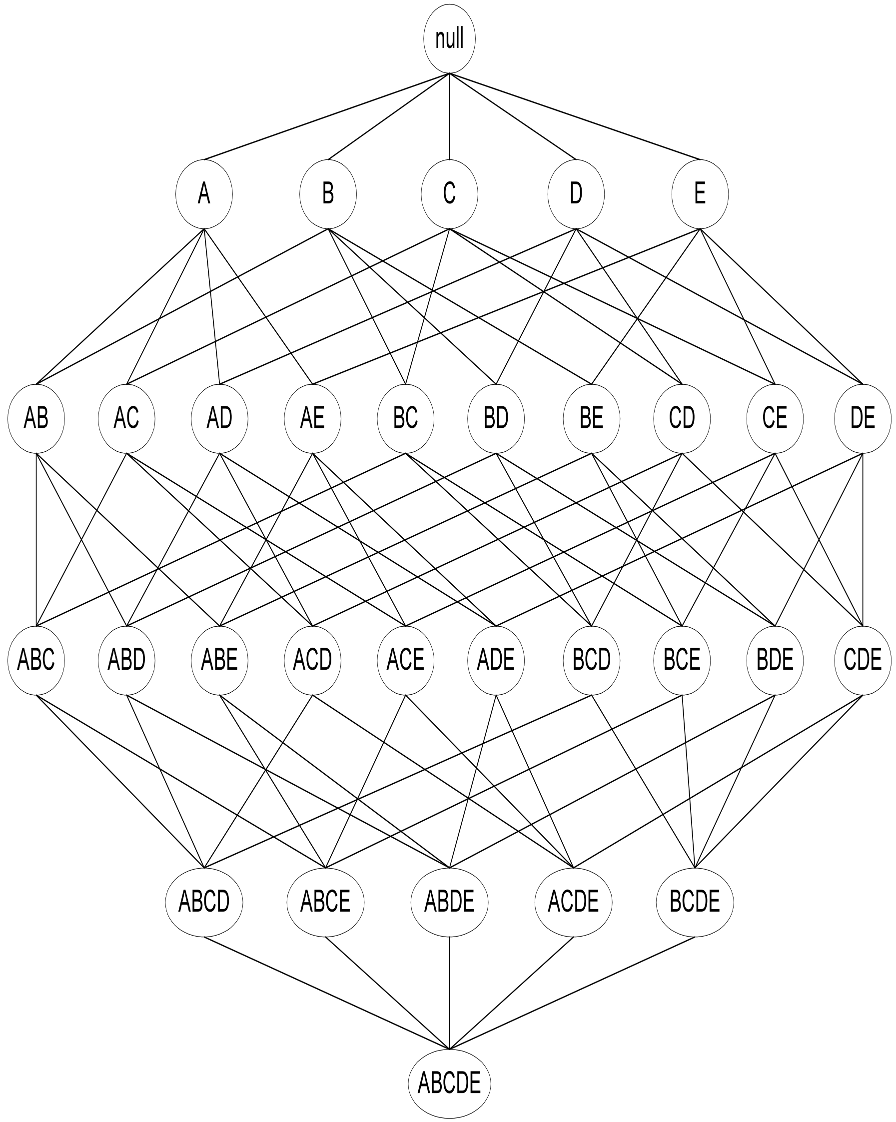
Spring 2010

Lecturer: Juho Rousu

Teaching assistant: Taru Itäpelto

Itemset lattice

- Itemsets that can be constructed from a set of items have a *partial order* with respect to the subset operator
  - i.e. a set is larger than its proper subsets
- This induces a lattice where nodes correspond to itemsets and arcs correspond to the subset relation
- The lattice is called the *itemset lattice*
- For d items, the size of the lattice is 2^d



Frequent itemsets on the itemset lattice

- The Apriori principle is illustrated on the Itemset lattice
  - The subsets of a frequent itemset are frequent
  - They span a sublattice of the original lattice (the grey area)

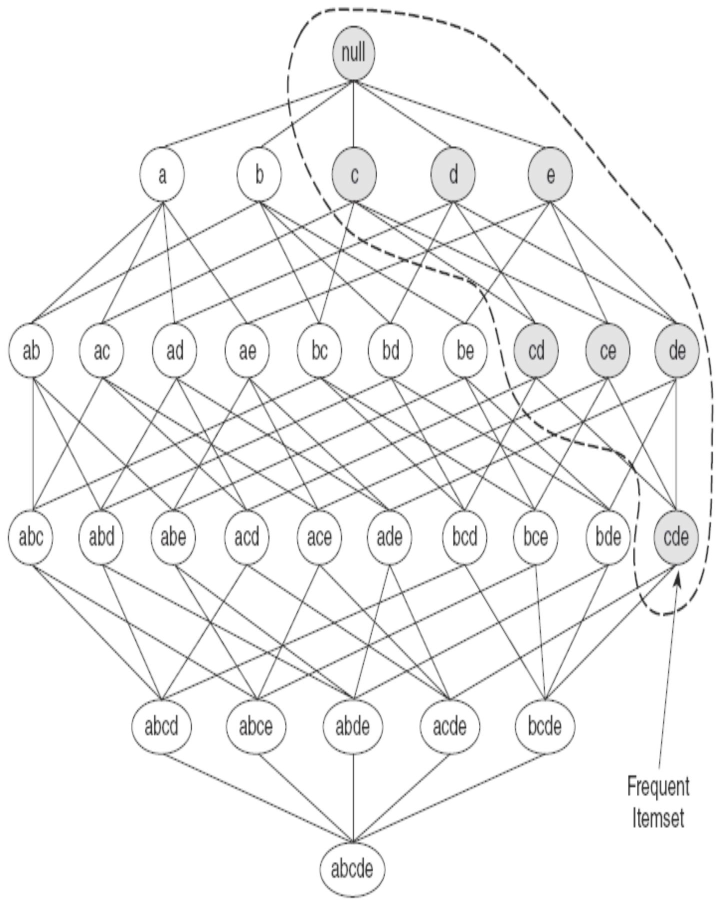


Figure 6.3. An illustration of the *Apriori* principle. If $\{c, d, e\}$ is frequent, then all subsets of this itemset are frequent.

Frequent itemsets on the itemset lattice

- Conversely
  - The supersets of an infrequent itemset are infrequent
  - They also span a sublattice of the original lattice (the crossed out nodes)
- If we know that {a,b} is infrequent, we never need to check any of the supersets
  - This fact is used in *support-based pruning*

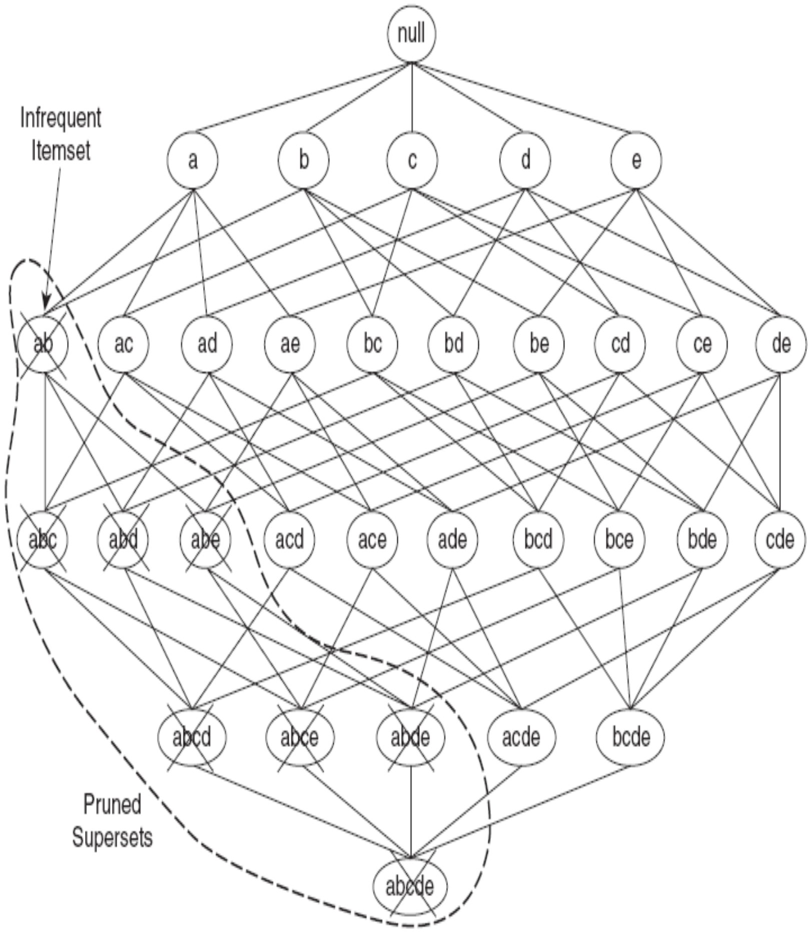


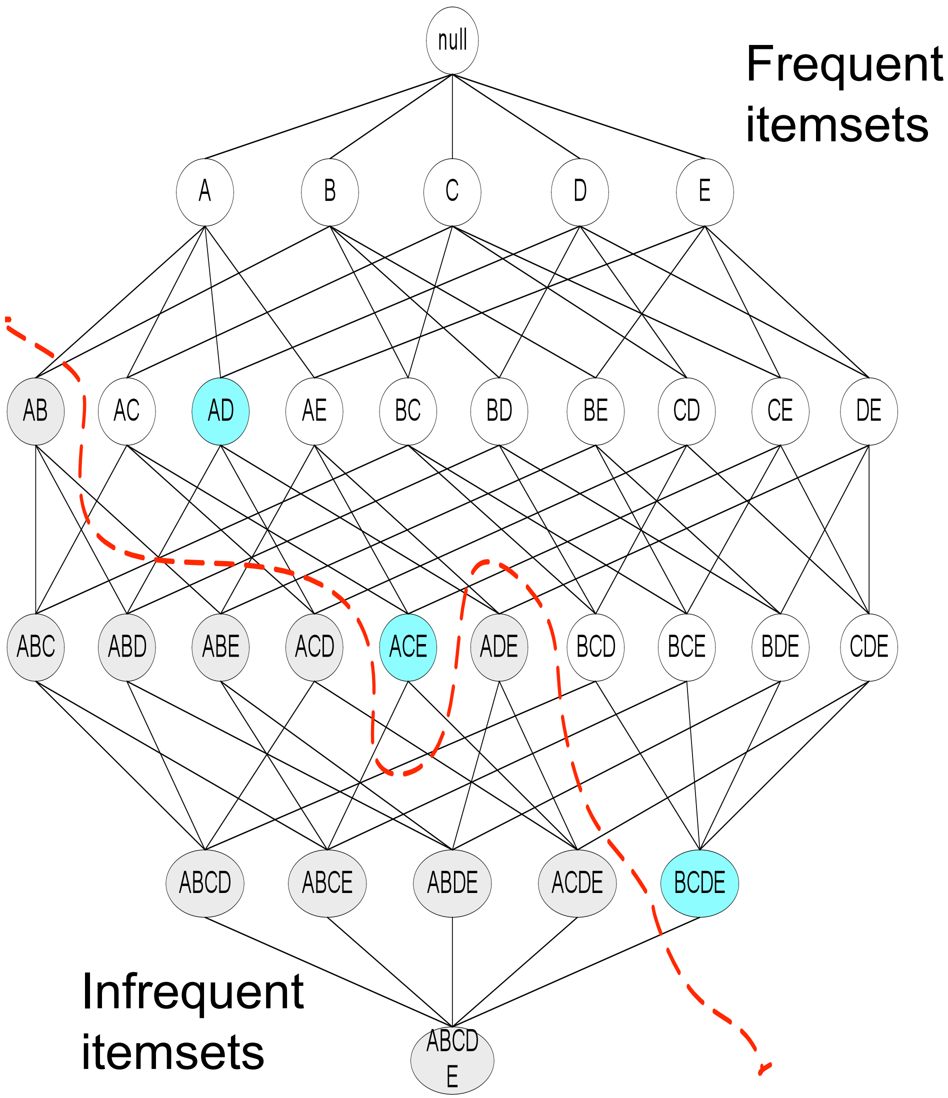
Figure 6.4. An illustration of support-based pruning. If $\{a, b\}$ is infrequent, then all supersets of $\{a, b\}$ are infrequent.

Compact Representation of Frequent Itemsets

- In practise, the number of frequent itemsets produced from transaction data can be very large
  - when the database is dense i.e. many items per transaction on average
  - when the number of transactions is high
  - when the minimum support level is set too low
- We will look at methods that
  - use the properties of the itemset lattice and the support function...
  - to compress the collection of frequent itemsets in a more manageable size...
  - so that all frequent itemsets can be derived from the compressed representation

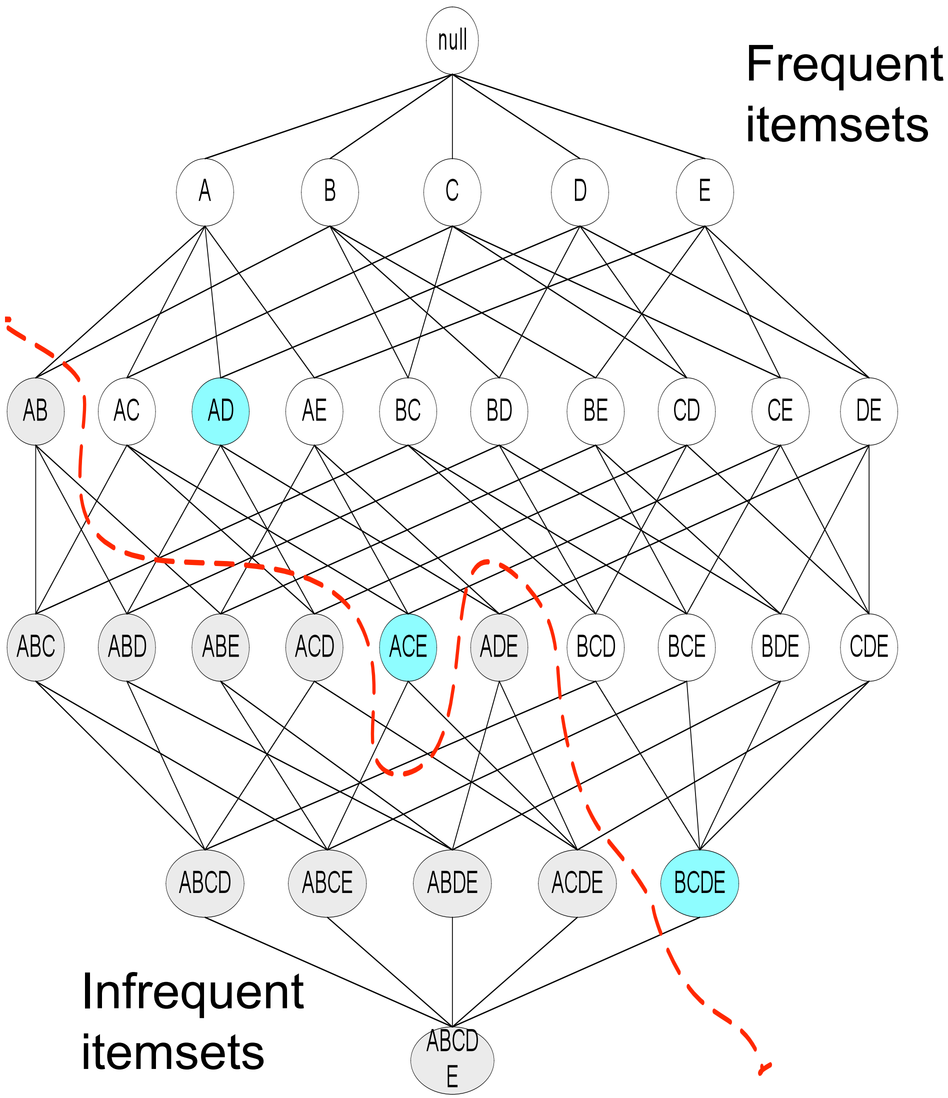
Maximal Frequent Itemsets

- The minimum support threshold induces a partition of the itemset lattice into frequent and infrequent itemsets (grey nodes)
- Frequent itemsets that cannot be extended with any item without making them infrequent are called *maximal frequent itemsets*
- We can derive all frequent itemsets from the set of maximal itemsets
  - Use of the Apriori principle “backwards”



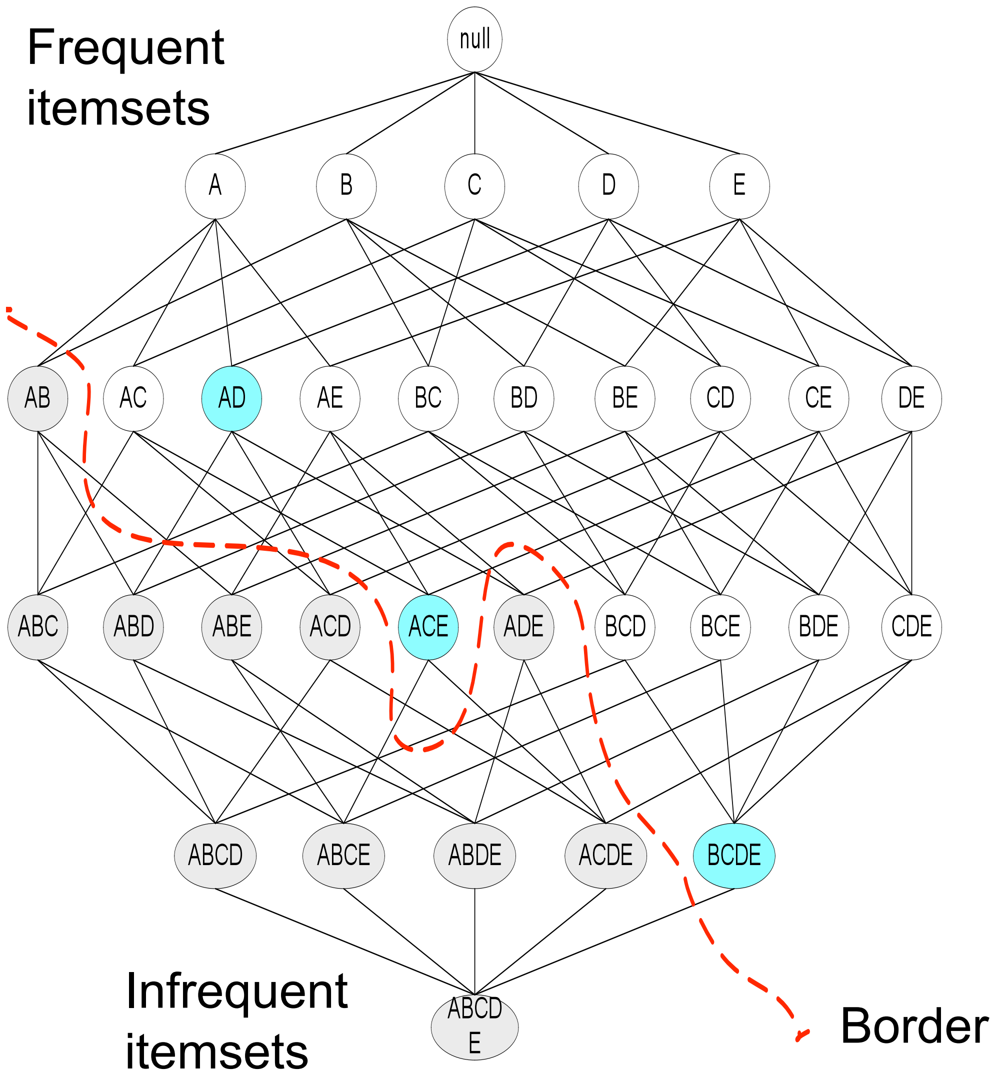
Maximal Frequent Itemsets

- {A,C} is not maximal as it can be extended to frequent itemset {A,C,E} although its supersets {A,B,C}, {A,C,D} are infrequent
- *{A,D}* is maximal as all its immediate supersets {A,B,D}, {A,C,D} and {A,D,E} are infrequent
- {B,D} is not maximal as it can be extended to frequent itemsets {B,C,D} and {B,D,E}



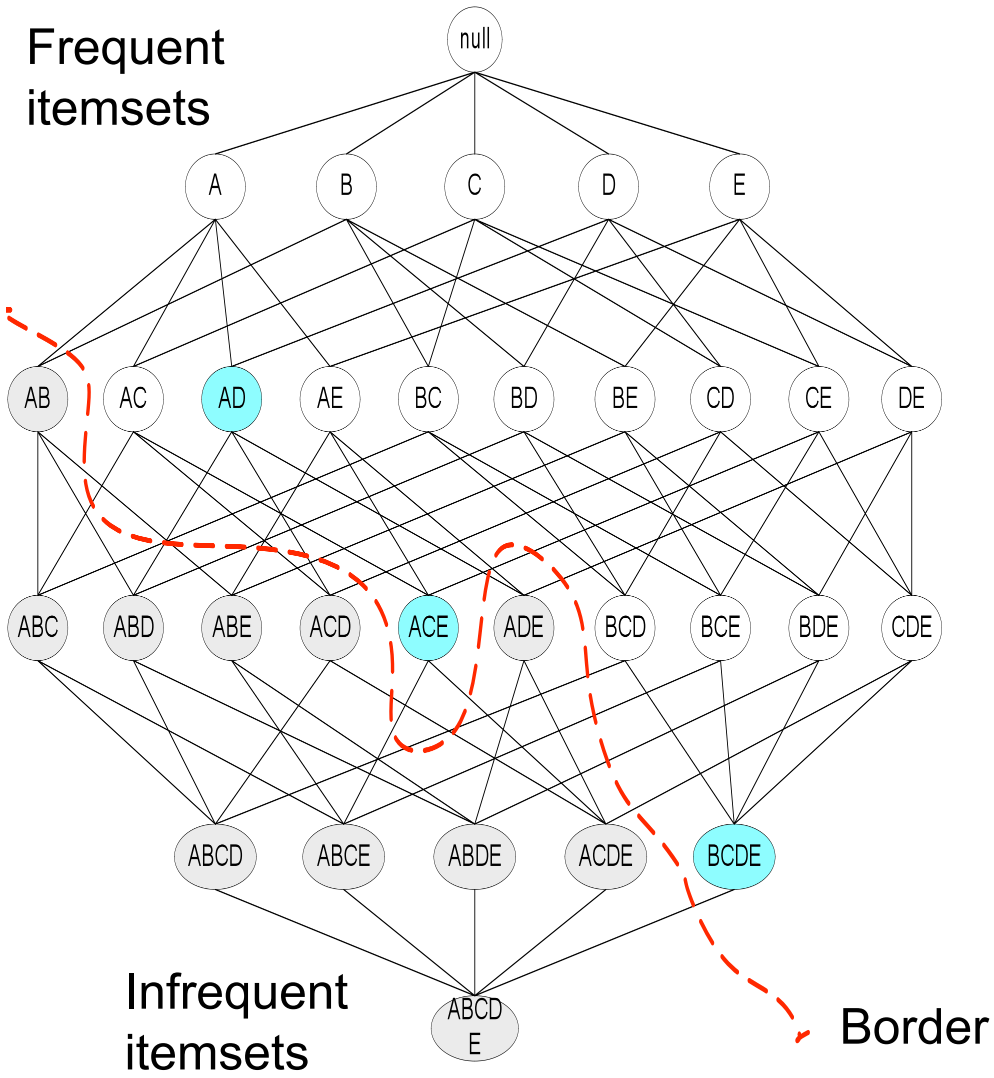
Maximal frequent itemsets

- The number of maximal frequent itemsets is typically considerably smaller than the number of all frequent itemsets
- In worst case, the number can still be exponential in the number of items:
  - e.g. consider the case where all itemsets of size d/2 are frequent and no itemset of size d/2+1 is frequent.
- Still need efficient algorithms



Maximal frequent itemsets

- *Exact support counts* of the subsets cannot be directly derived from support of the maximal frequent itemset
- From Apriori principle we only know that the subsets must be frequent, but not how frequent
- Need to do support counting for the subsets of the maximal frequent itemset to create association rules



Closed itemsets

- An alternative approach is to try to retain some of the support information in the compacted representation
- A *closed itemset* is an itemset whose all immediate supersets have different support count
- A *closed frequent itemset* is a closed itemset that satisfies the minimum support threshold
- Maximal frequent itemsets are closed by definition

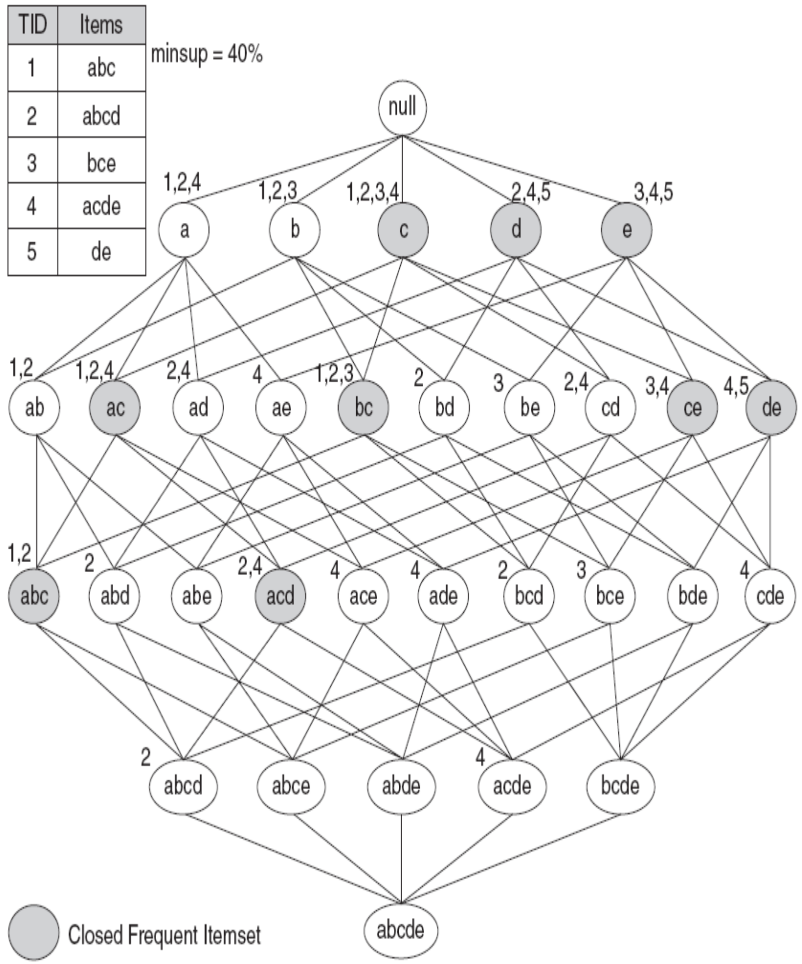


Figure 6.17. An example of the closed frequent itemsets (with minimum support count equal to 40%).

Example: Closed frequent itemsets

- Assume minimum support threshold 40%
- {b} is frequent: σ({b})=3, but not closed: σ({b}) = σ({b,c}) = 3
- {b,c} is frequent: σ({b,c})= 3, and closed: σ({a,b,c}) = 2, σ({b,c,d})=1,σ({b,c,e})=1
- {b,c,d} is not frequent: σ({b,c,d}) = 1, and not closed : σ({a,b,c,d}) = 1

TID	Items
1	abc
2	abcd
3	bce
4	acde
5	de

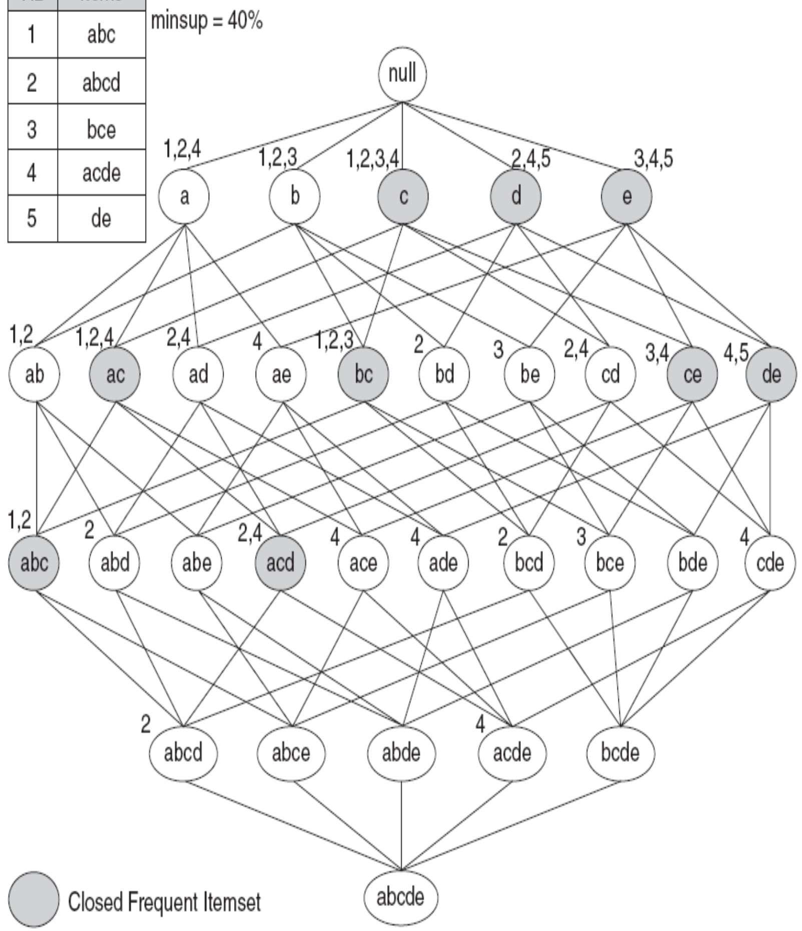
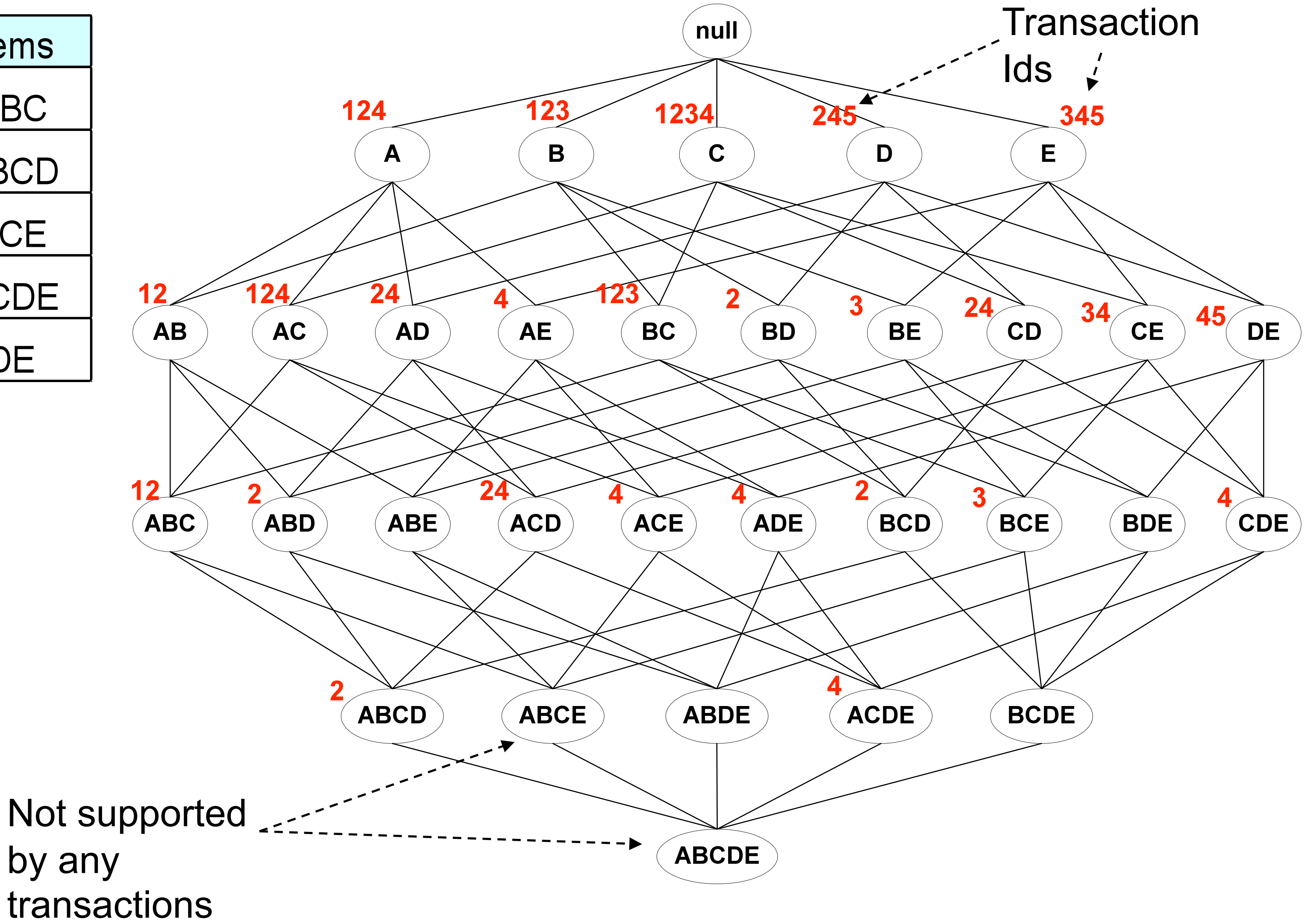


Figure 6.17. An example of the closed frequent itemsets (with minimum support count equal to 40%).

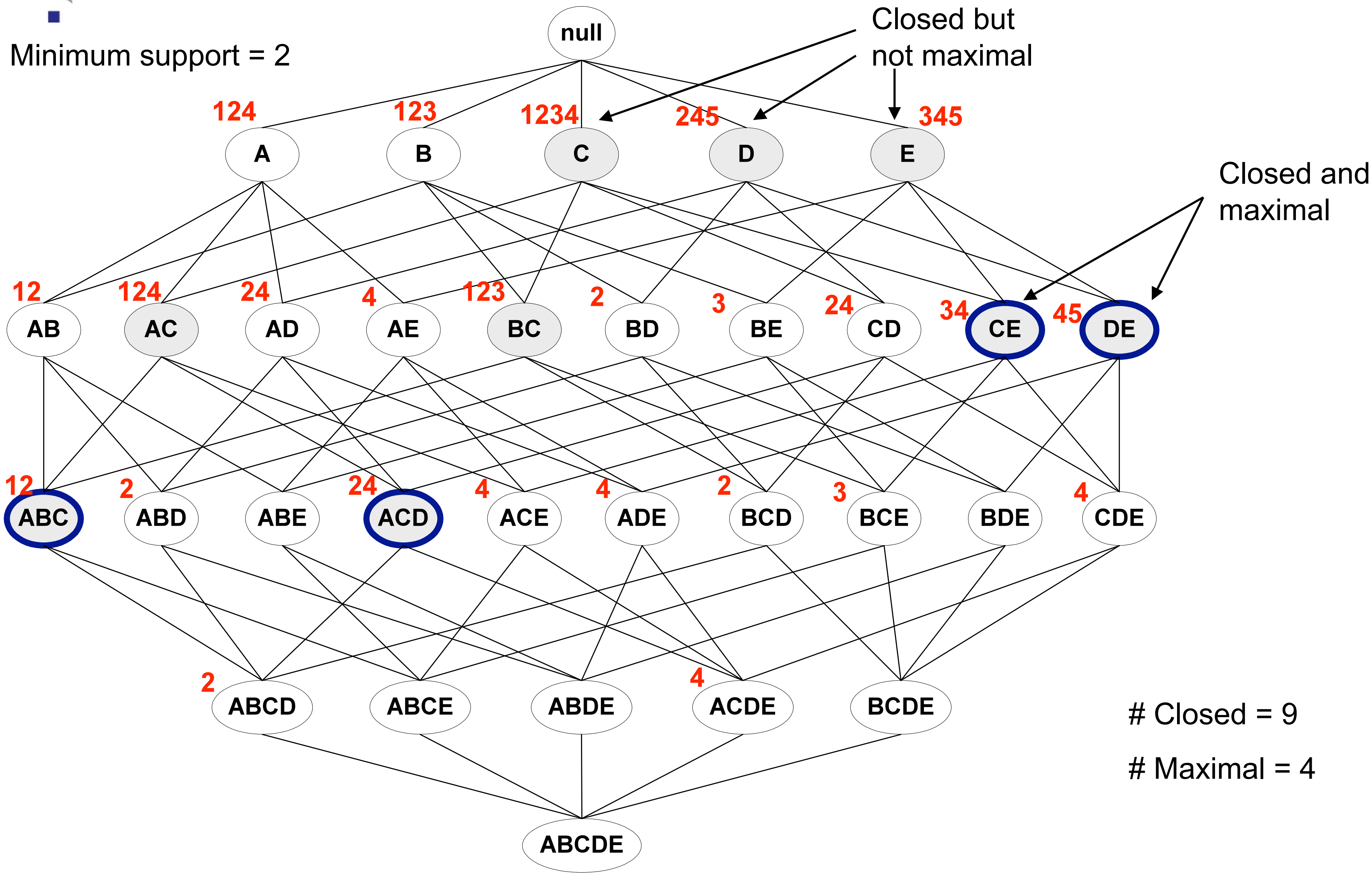
Maximal vs Closed Itemsets

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



Maximal vs Closed Frequent Itemsets

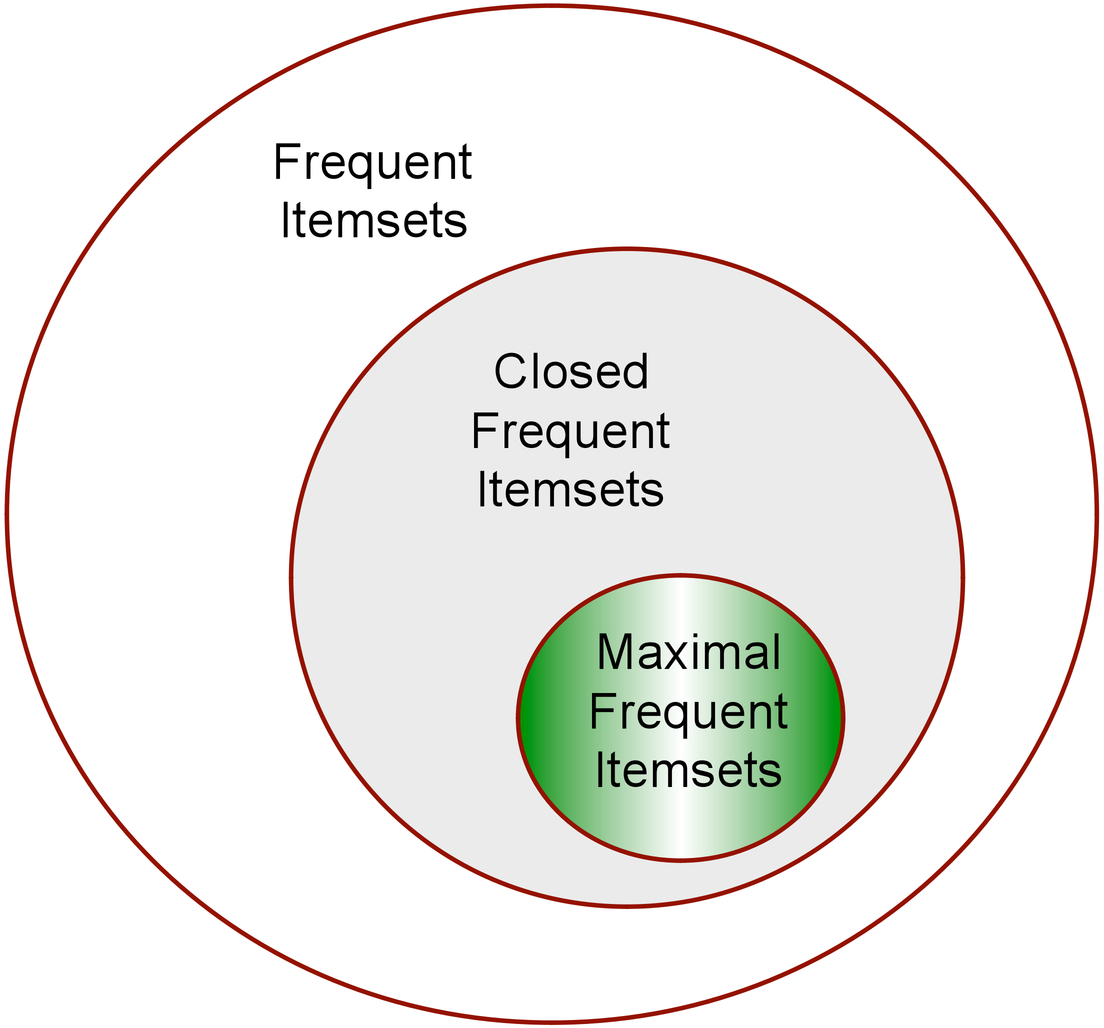
Minimum support = 2



Closed = 9

Maximal = 4

Maximal vs Closed Itemsets



Determining the support of non-closed frequent itemsets

- Consider a non-closed frequent itemset {a,d}
  - assume we have not stored its support count
- By definition, there must be at least one immediate superset that has the same support count
- It must be that σ({a,d}) = σ(X) for some immediate superset X of {a,d}

TID	Items
1	abc
2	abcd
3	bce
4	acde
5	de

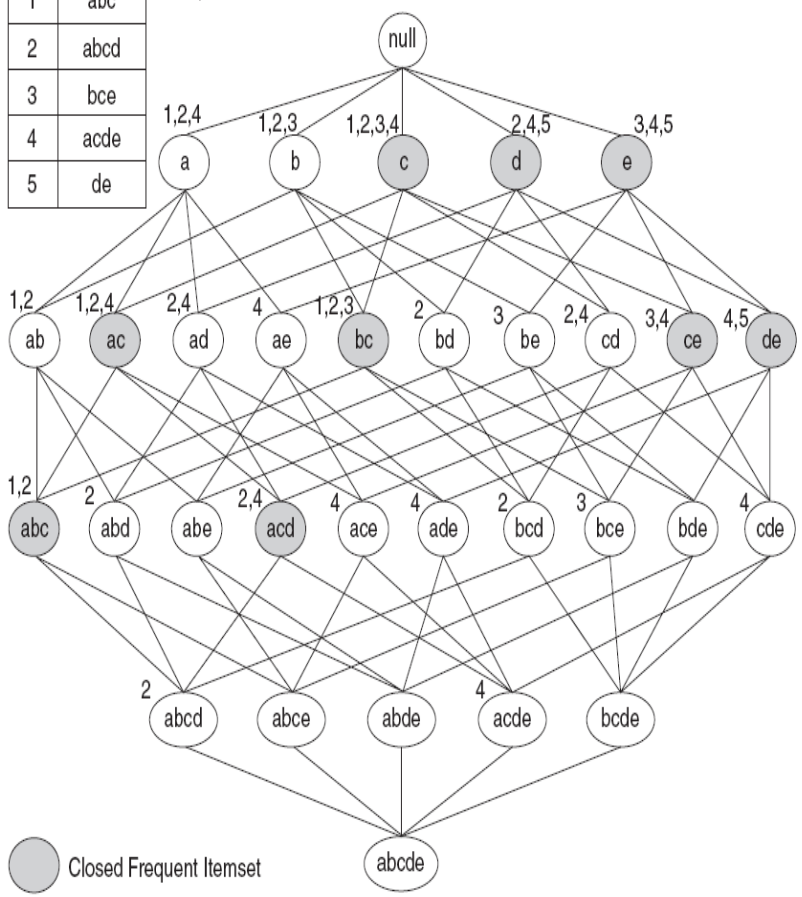


Figure 6.17. An example of the closed frequent itemsets (with minimum support count equal to 40%).

Determining the support of non-closed frequent itemsets

- From the Apriori principle we know that no superset can have higher support than {a,d}
- It must be that the support equals the support of the most frequent superset

σ({a,d}) =
max(σ(abd),σ(acd),σ(ade))

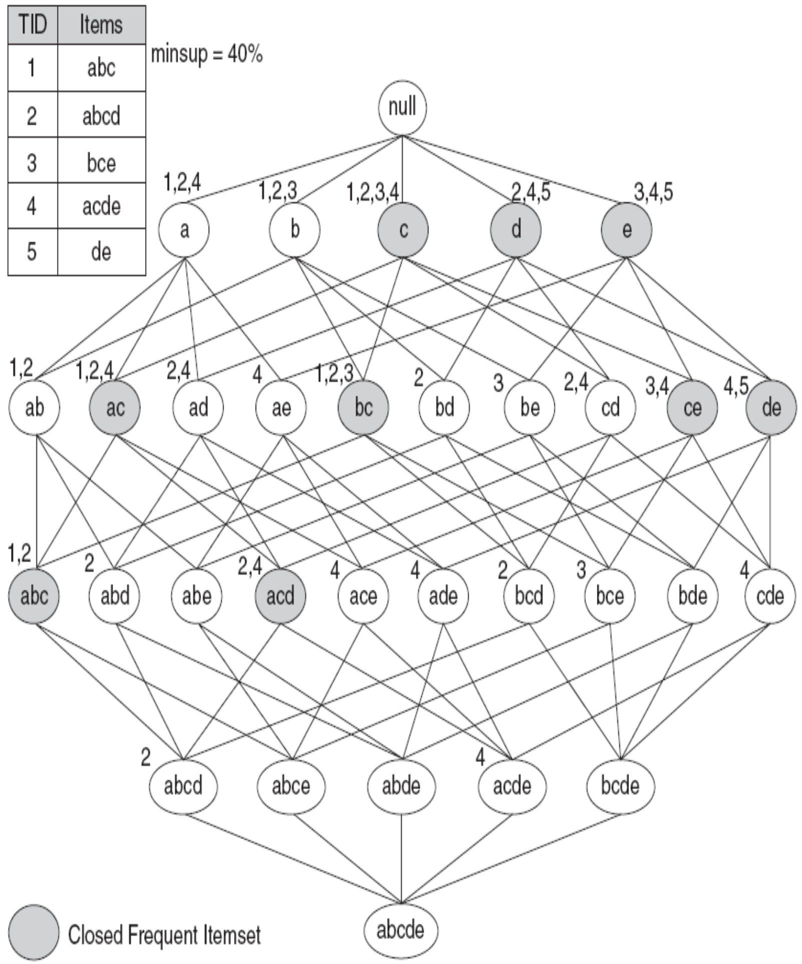


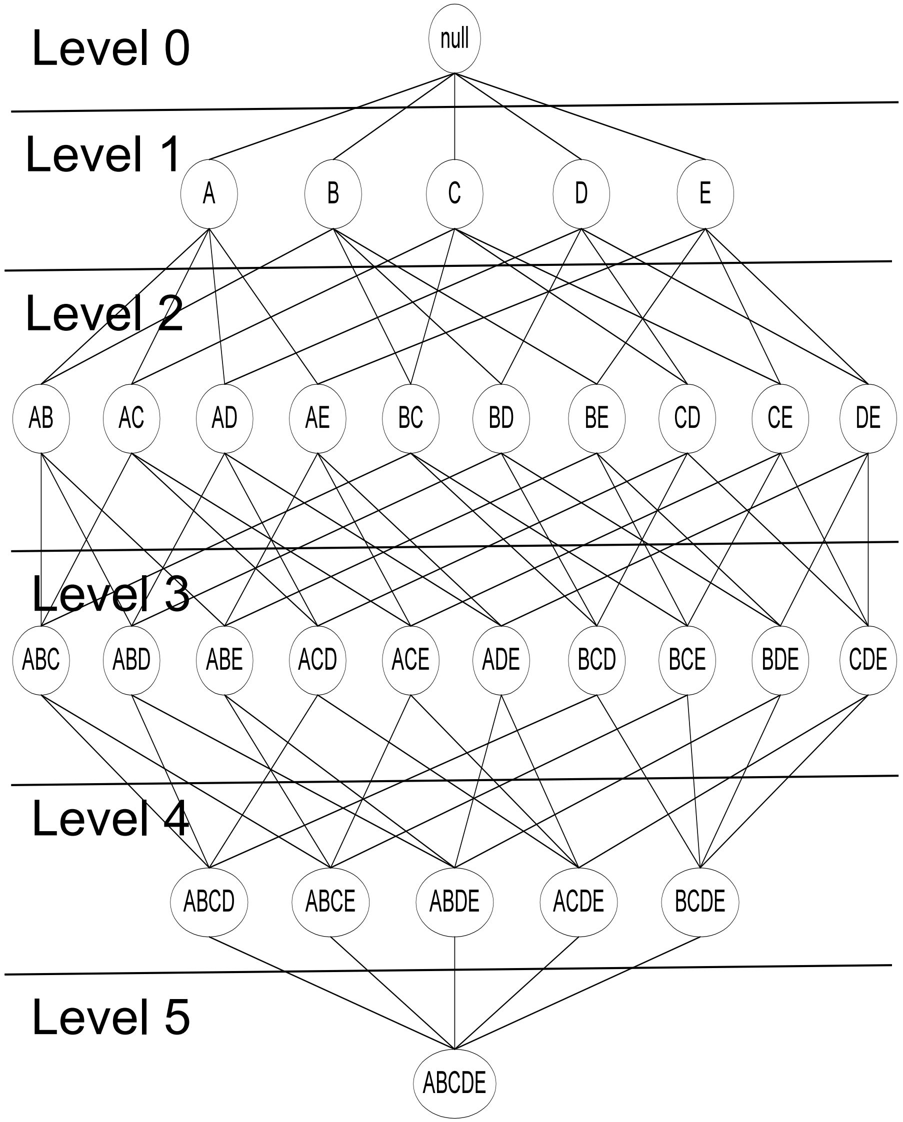
Figure 6.17. An example of the closed frequent itemsets (with minimum support count equal to 40%).

Determining the support of non-closed frequent itemsets

- Algorithm sketch:
  1. kmax = size of largest closed frequent itemset
  2. F_{kmax} = closed frequent itemsets of size kmax
  3. for k = kmax-1 downto 1 do
  4. F_k = {f | f immediate subset of f' in F_{k+1} or f is closed, |f|=k }
  5. for every f in F_k do
  6. if f is not closed
  7. f.support = max(f'.support | f' in F_{k+1}, f' is a superset of f)
  8. endif
  9. endfor
  10. endfor

Characteristics of Apriori algorithm

- Breadth-first search algorithm:
  - all frequent itemsets of given size are kept in the algorithms processing queue
- General-to-specific search:
  - start with itemsets with large support, work towards lower-support region
- Generate-and-test strategy:
  - generate candidates, test by support counting

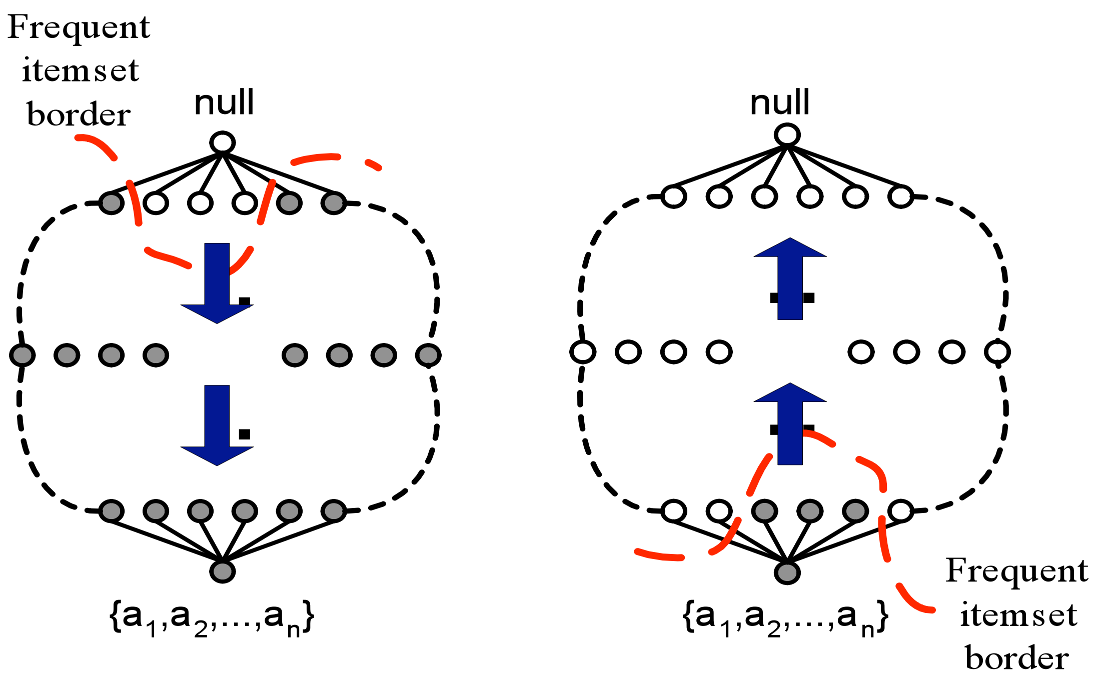


Weaknesses of Apriori

- Apriori is one of the first algorithms that succesfully tackled the exponential size of the frequent itemset space
- Nevertheless the Apriori suffers from two main weaknesses:
  - High I/O overhead from the generate-and-test strategy: several passes are required over the database to find the frequent itemsets
  - The performance can degrade significantly on dense databases, as large portion of the itemset lattice becomes frequent

Alternative methods for generating frequent itemsets: Traversal of itemset lattice

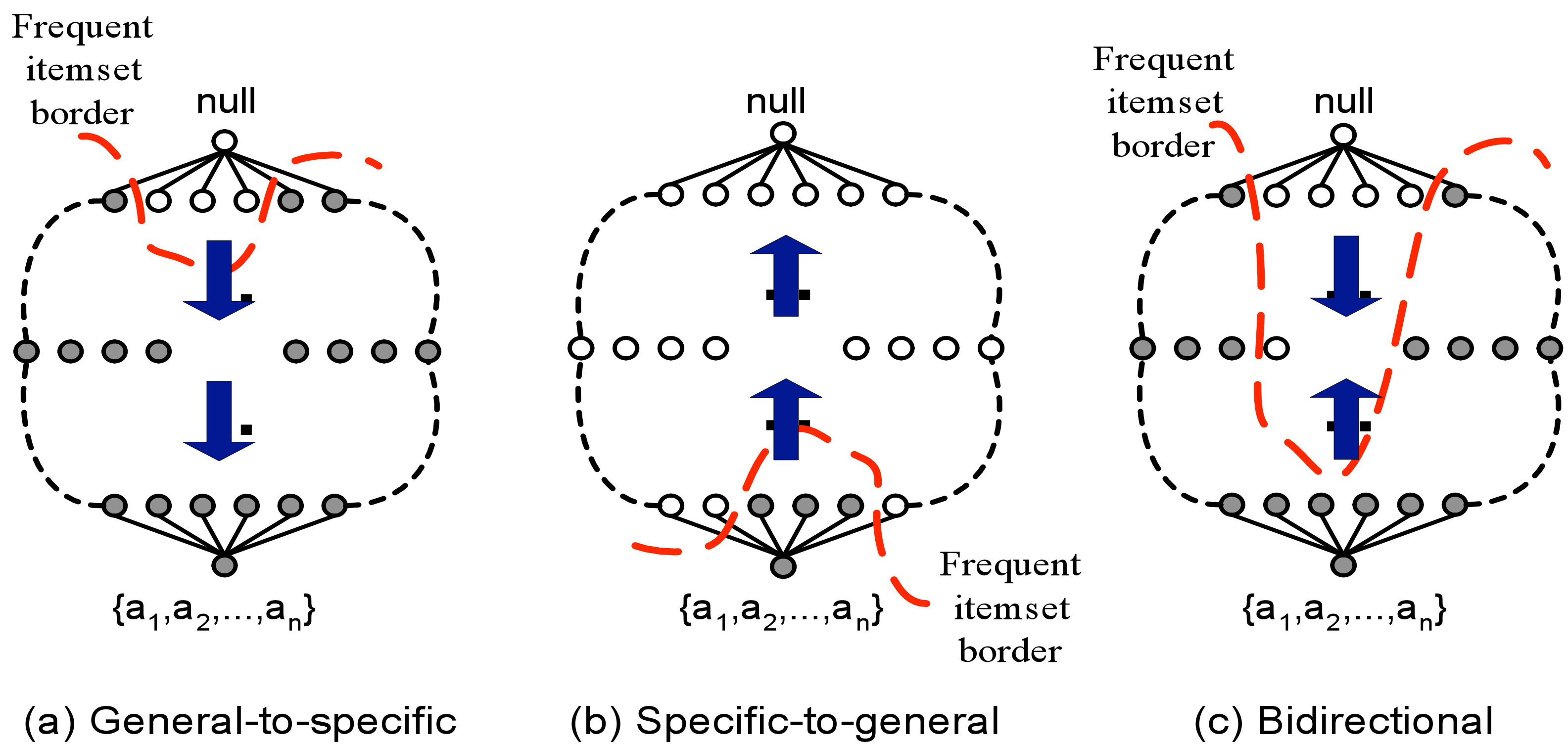
- Apriori uses general-to-specific search: start from most highly supported itemsets, work towards lower support region
  - Works well if the frequent itemset border is close to the top of the lattice



(a) General-to-specific (b) Specific-to-general (c) Bidirectional

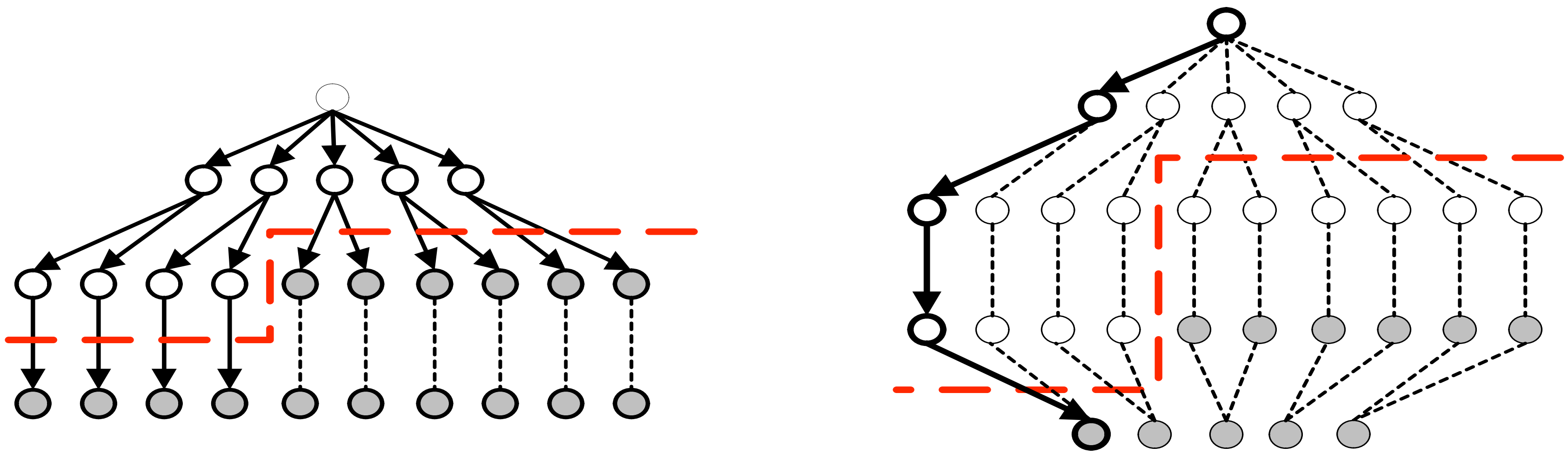
Alternative methods for generating frequent itemsets: Traversal of itemset lattice

- Specific-to-general search: look first for the most specific frequent itemsets, work towards higher support region
- Works well if the border is close to the bottom of the lattice
  - Dense databases



Alternative Methods for Frequent Itemset Generation: Breadth-first vs Depth-first

- Apriori traverses the itemset lattice in breadth-first manner
- Alternatively, the lattice can be searched in depth-first manner: extend single itemset until it cannot be extended
  - often used to find maximal frequent itemsets
  - hits the border of frequent itemsets quickly



(a) Breadth first (b) Depth first

Alternative Methods for Frequent Itemset Generation: Breadth-first vs Depth-first

- Depth-first search allows different kind of pruning of the search space
- Example: if {b,c,d,e} is found maximal frequent by the search algorithm, the region of the lattice consisting of subsets of {b,c,d,e} does not need to be traversed
  - known to be frequent non-maximal

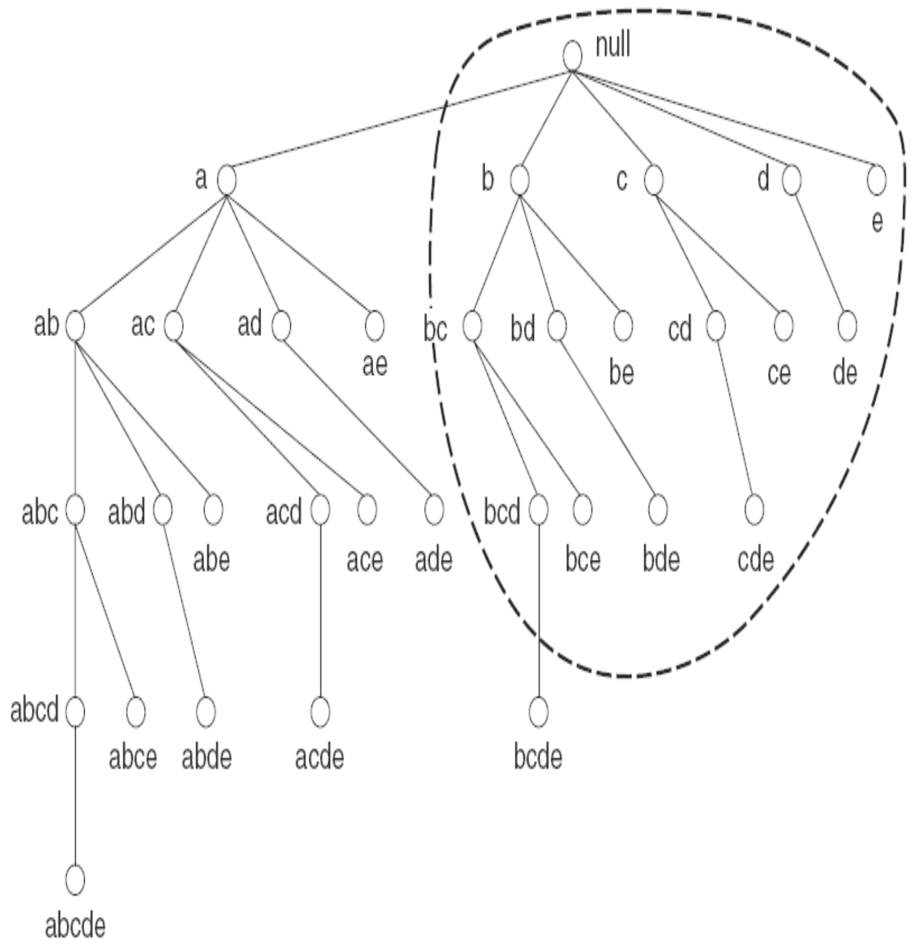


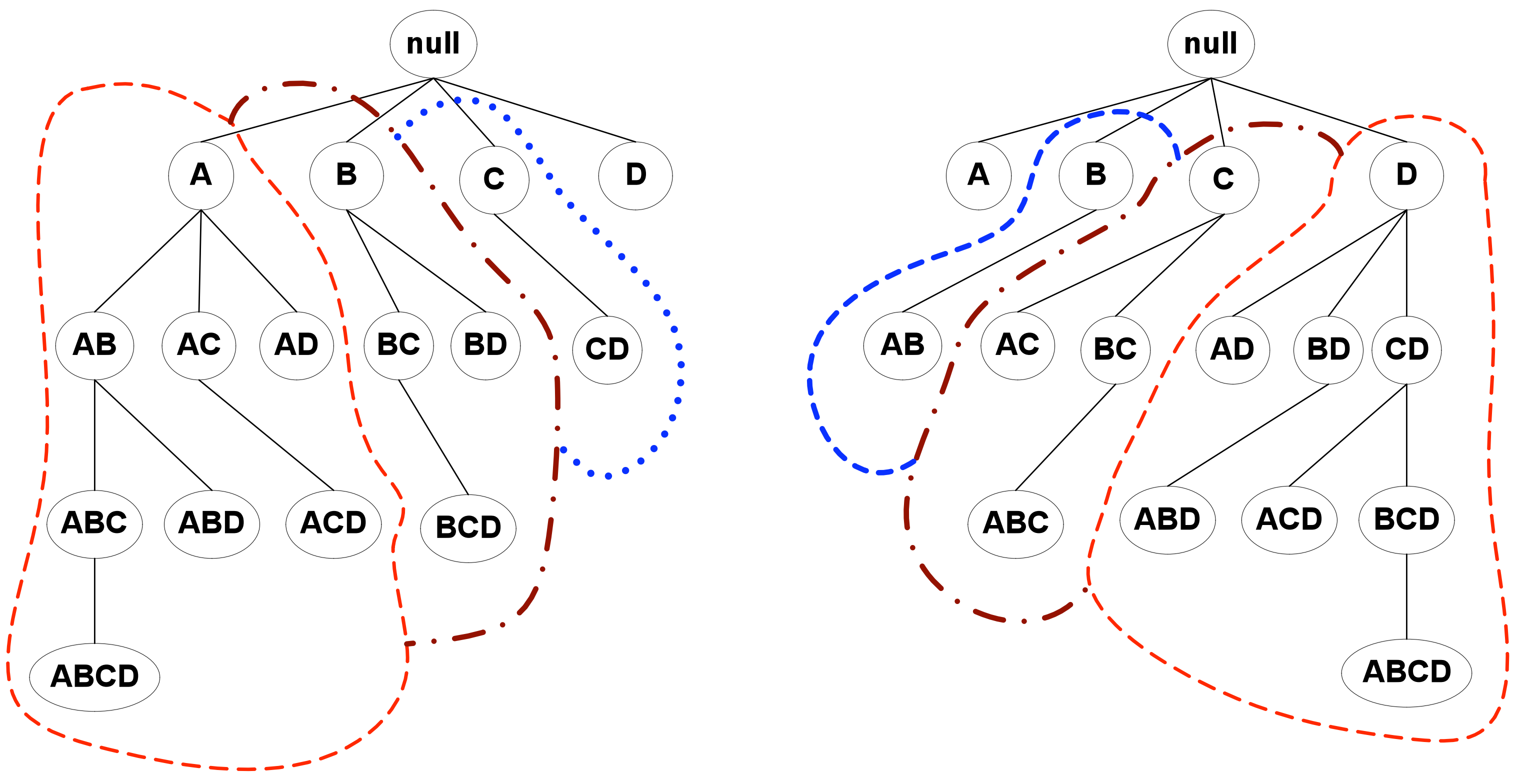
Figure 6.22. Generating candidate itemsets using the depth-first approach.

Alternative methods for generating frequent itemsets: Equivalence classes

- Many search algorithms can be seen to conceptually partition the itemset lattice into equivalence classes
  - The itemsets in one equivalence class are processed before moving into the next
- Several ways of defining equivalence classes
  - Levels defined by itemset size (used by Apriori)
  - Prefix labels: two itemsets that share a prefix of length k belong to the same class e.g. {a,c,d}, {a,c,e} if k <= 2
  - Suffix labels: two itemsets that share a suffix of length k

Prefix and suffix trees

- Left: prefix tree and equivalence classes defined by for prefixes of length k=1
- Right: suffix tree and equivalence classes defined by for prefixes of length k=1



(a) Prefix tree (b) Suffix tree

FP-growth algorithm

- FP-growth avoids the repeated scans of the database of Apriori by using a compressed representation of the transaction database using a data structure called FP-tree
- Once an FP-tree has been constructed, it uses a recursive divide-and-conquer approach to mine the frequent itemsets

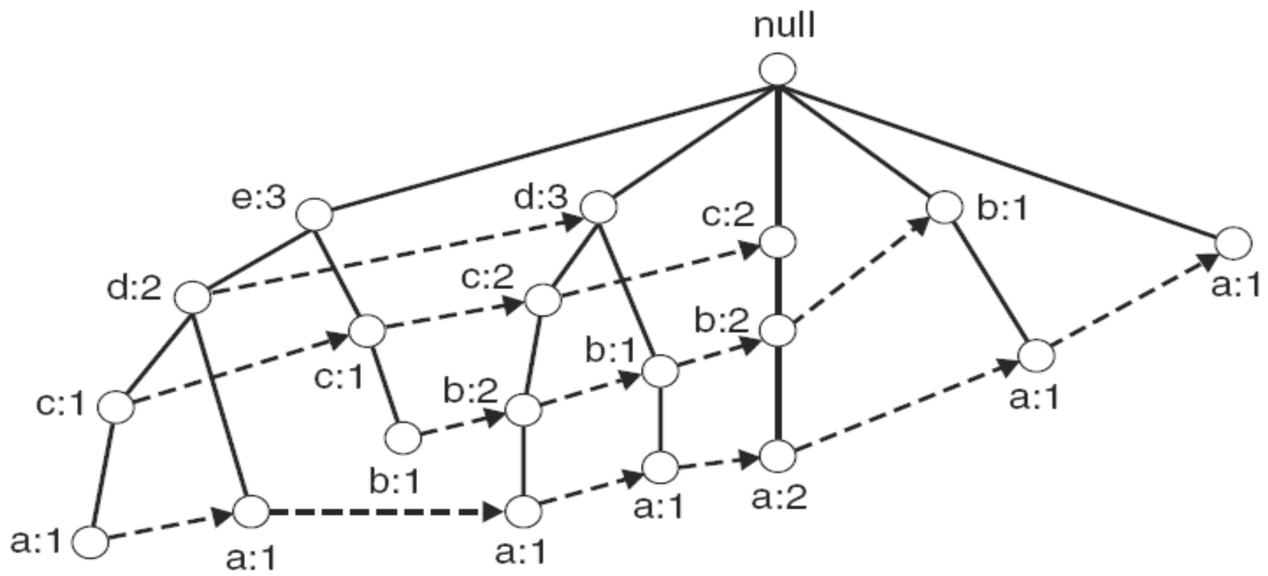


Figure 6.25. An FP-tree representation for the data set shown in Figure 6.24 with a different item ordering scheme.

FP-tree

- FP-tree is a compressed representation of the transaction database
- Each transaction is mapped onto a path in the tree
- Each node contains an item and the support count corresponding to the number of transactions with the prefix corresponding to the path from root
- Nodes having the same item label are cross-linked: this helps finding the frequent itemsets ending with a particular item

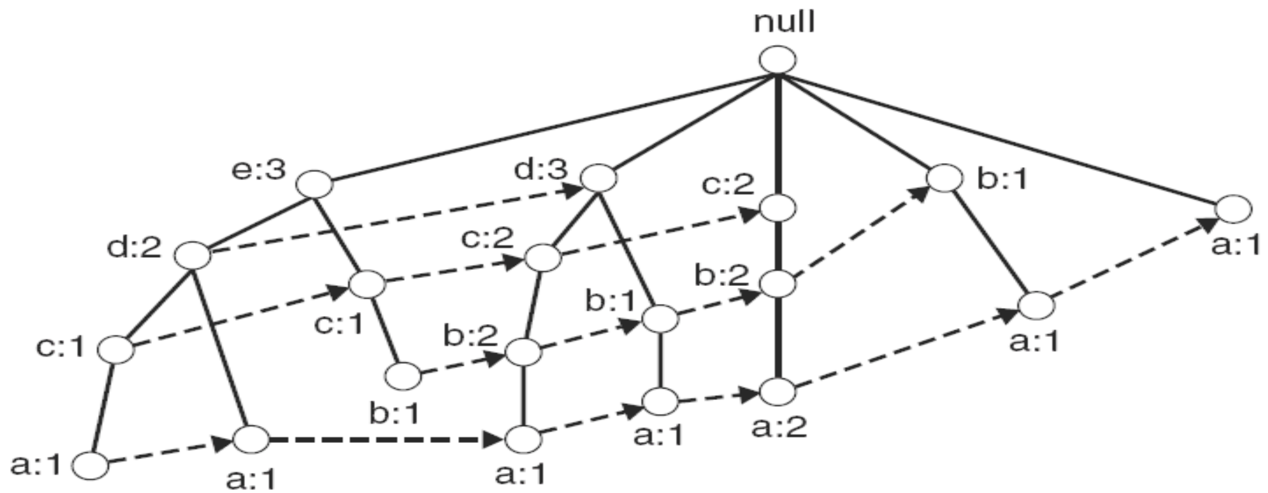
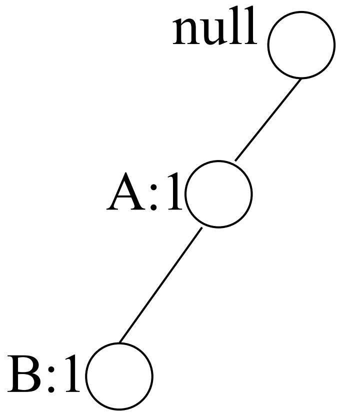


Figure 6.25. An FP-tree representation for the data set shown in Figure 6.24 with a different item ordering scheme.

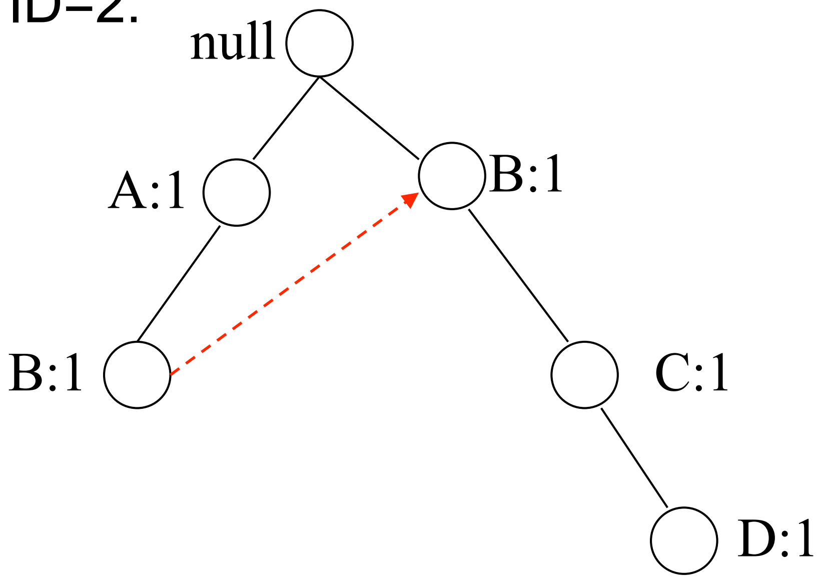
FP-tree construction

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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

After reading TID=1:



After reading TID=2:



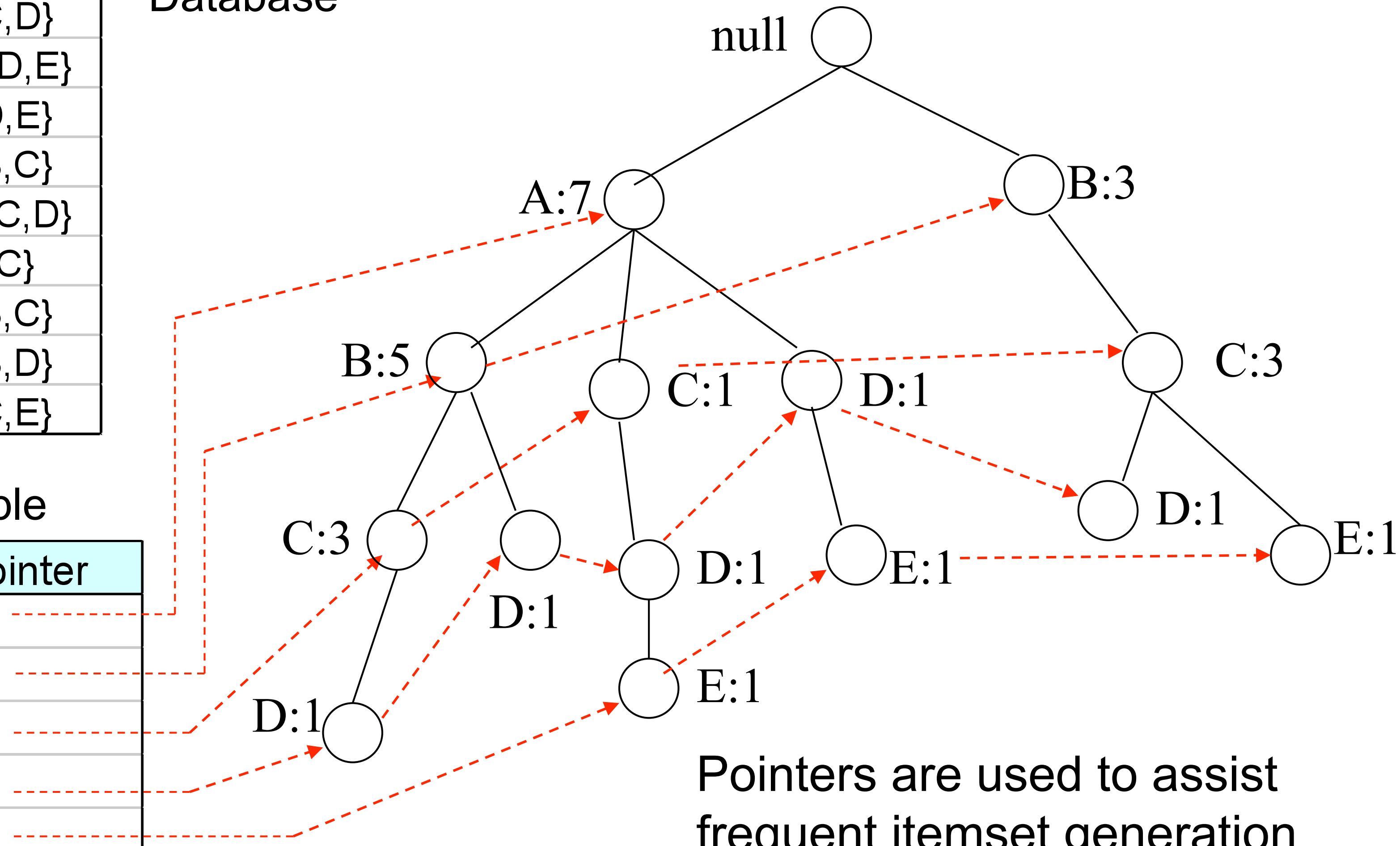
FP-Tree Construction

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	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Transaction Database

Header table

Item	Pointer
A	
B	
C	
D	
E	



Pointers are used to assist frequent itemset generation

FP-Tree vs. original database

- If the transactions share a significant number of items, FP-tree can be considerably smaller as the common subset of the items is likely to share paths
- There is a storage overhead from the links as well from the support counts, so in worst case may even be larger than original

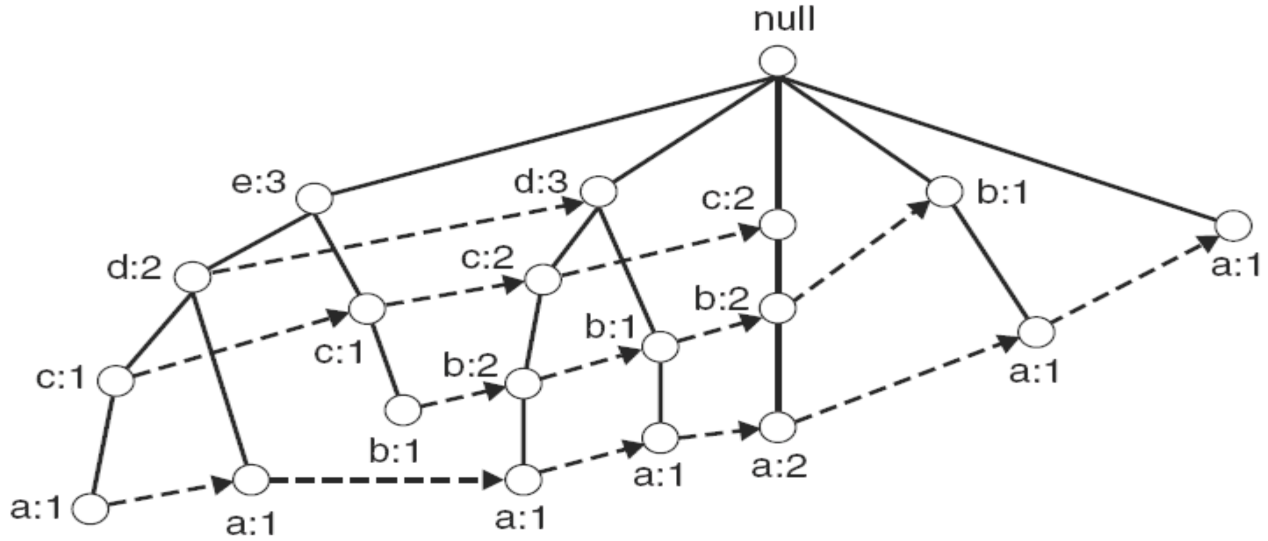
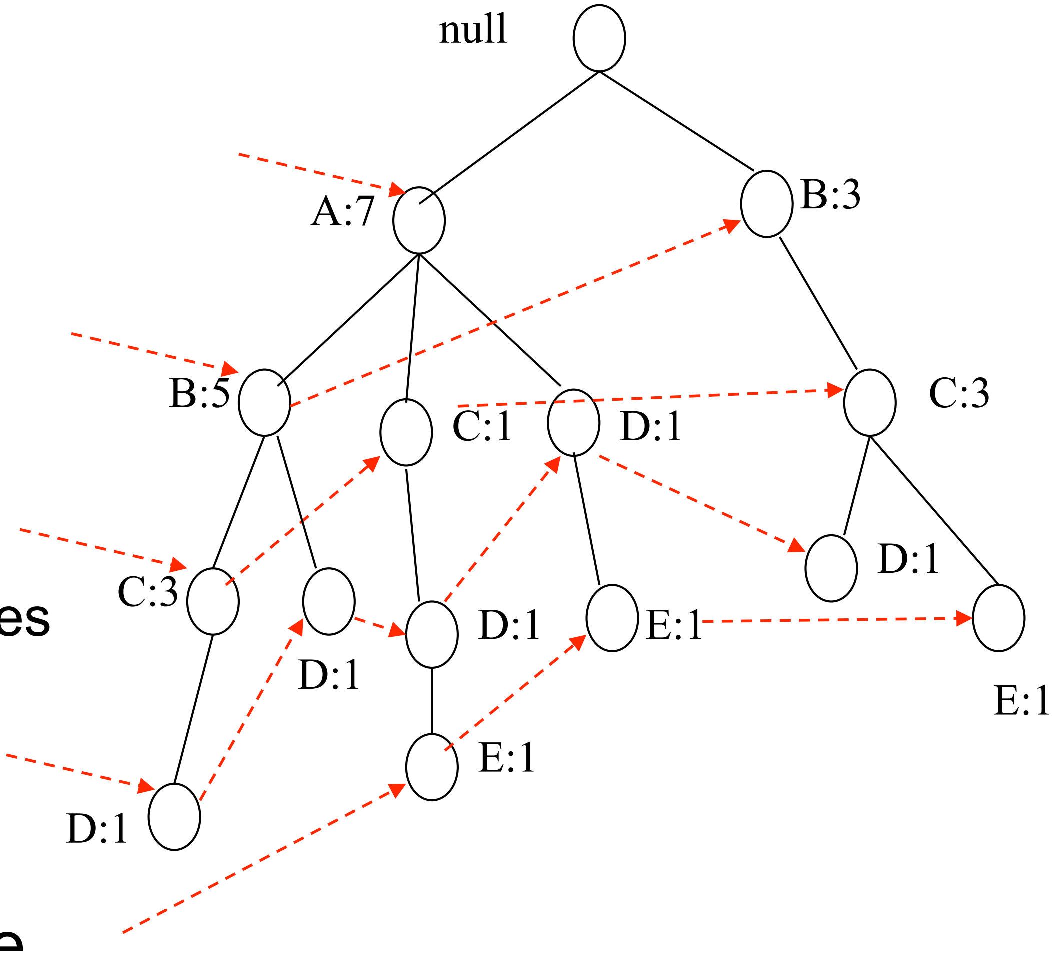


Figure 6.25. An FP-tree representation for the data set shown in Figure 6.24 with a different item ordering scheme.

Frequent itemset generation in FP-growth

- FP-growth uses a divide-and-conquer approach to find frequent itemsets
- It searches frequent itemsets ending with item E first, then itemsets ending with D,C,B,A
  - i.e. uses equivalence classes based on length-1 suffixes
- Paths corresponding to different suffixes are extracted from the FP-tree



Frequent itemset generation in FP-growth

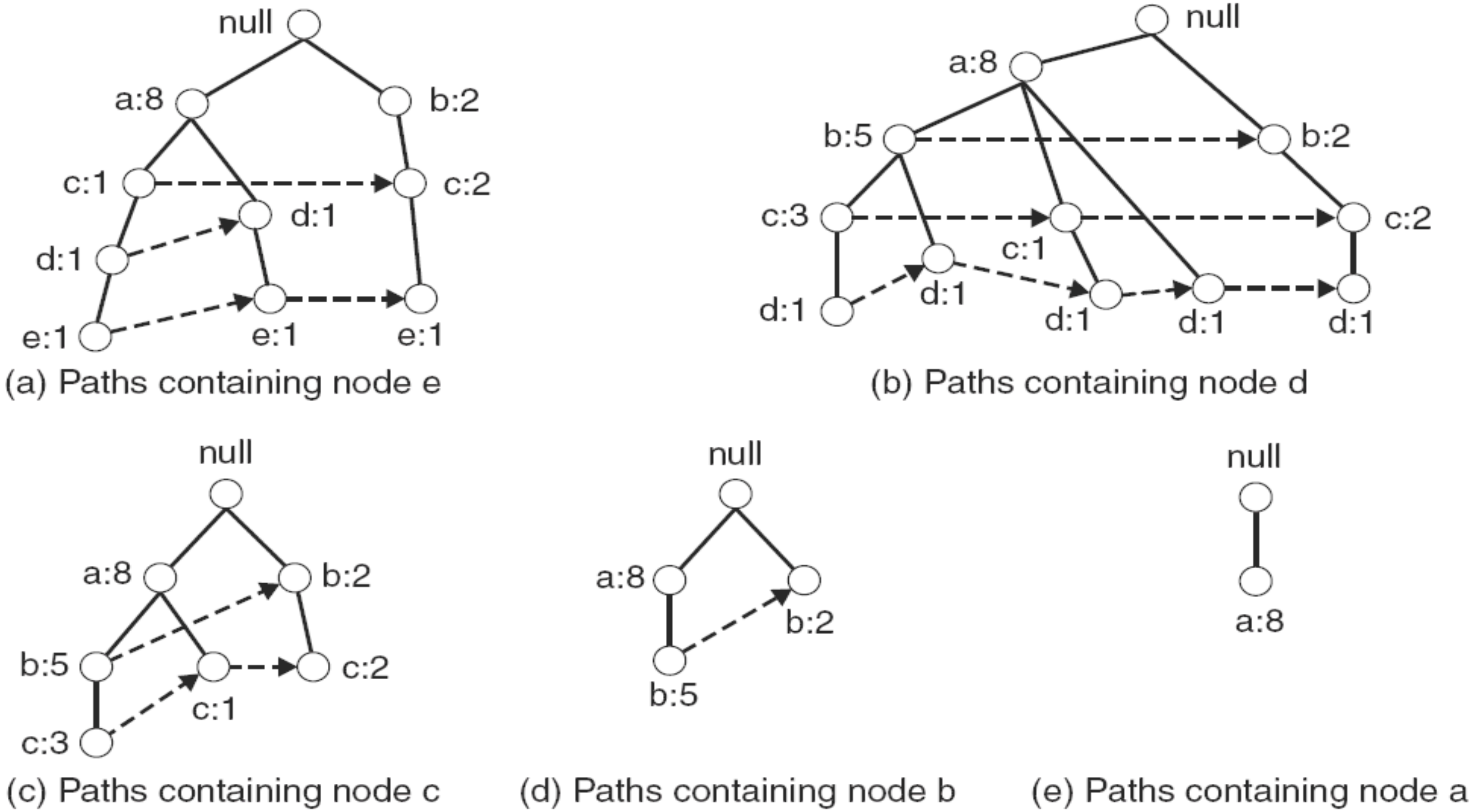
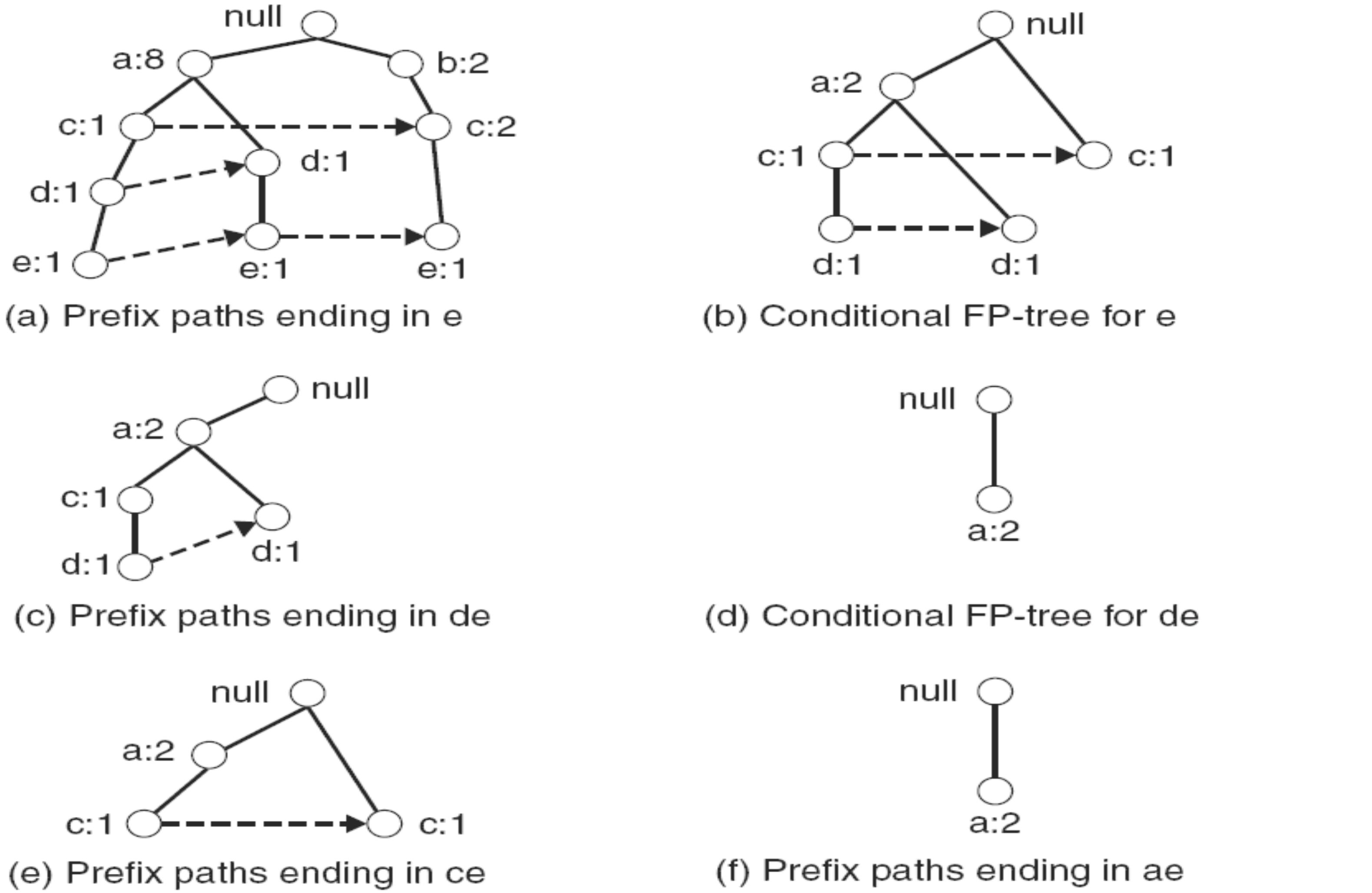


Figure 6.26. Decomposing the frequent itemset generation problem into multiple subproblems, where each subproblem involves finding frequent itemsets ending in e, d, c, b, and a.

Frequent itemset generation in FP-growth

- To find all frequent itemsets ending with given last item (e.g. E), we first need to compute the support of the item
- This is given by the sum of support counts of all nodes labeled with the item (σ(E)=3)
  - found by following the cross-links connecting the nodes with the same item
- If last item is found frequent, FP-growth next iteratively looks for all frequent itemsets ending with given length-2 suffix (DE,CE,BE, and AE),
  - and recursively with length-3 suffix, length-4 suffix until no more frequent itemsets are found
- Conditional FP-tree is constructed for each different suffix to speed up the computation

Frequent itemset generation in FP-growth



(a) Prefix paths ending in e

(b) Conditional FP-tree for e

(c) Prefix paths ending in de

(d) Conditional FP-tree for de

(e) Prefix paths ending in ce

(f) Prefix paths ending in ae

Figure 6.27. Example of applying the FP-growth algorithm to find frequent itemsets ending in e.