Chapter 7: Frequent Itemsets and Association Rules

Information Retrieval & Data Mining
Universität des Saarlandes, Saarbrücken
Winter Semester 2013/14

Motivational Example

- Assume you run an on-line store and you want to increase your sales
  - You want to show visitors ads of your products before they search the products

- This is easy if you know the left-hand side
  - But if you don't…

Chapter VII: Frequent Itemsets and Association Rules*

1. Definitions: Frequent Itemsets and Association Rules
2. Algorithms for Frequent Itemset Mining
   - Monotonicity and candidate pruning, Apriori, ECLAT, FPGrowth
3. Association Rules
   - Measures of interestingness
4. Summarizing Itemsets
   - Closed, maximal, and non-derivable itemsets

*Zaki & Meira, Chapters 10 and 11; Tan, Steinbach & Kumar, Chapter 6

Chapter VII.1: Definitions

1. The transaction data model
    1.1. Data as subsets
    1.2. Data as binary matrix
2. Itemsets, support, and frequency
3. Association rules
4. Applications of association analysis

The transaction data model

- Data mining considers larger variety of data types than typical IR
- Methods usually work on any data that can be expressed in certain type
  - Graphs, points in metric space, vectors, ...
- The data type used in itemset mining is the **transaction data**
  - Data contains transactions over some set of items

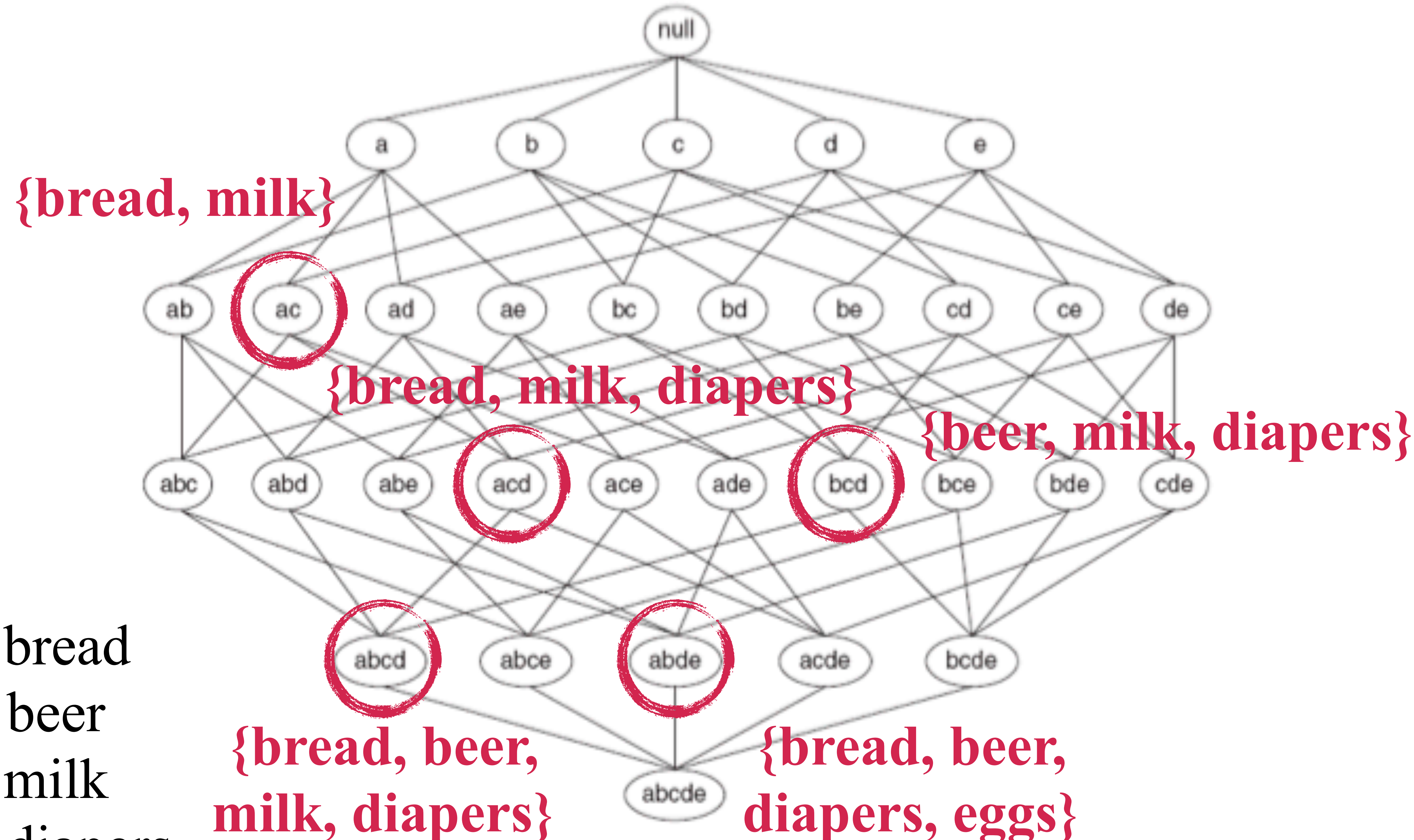
The market basket data

Items are: bread, milk, diapers, beer, and eggs
Transactions are: 1:{bread, milk}, 2:{bread, diapers, beer, eggs}, 3:{milk, diapers, beer}, 4:{bread, milk, diapers, beer}, and 5:{bread, milk, diapers}

Transaction IDs

TID	Bread	Milk	Diapers	Beer	Eggs
1	✔	✔			
2	✔		✔	✔	✔
3		✔	✔	✔	
4	✔	✔	✔	✔	
5	✔	✔	✔		

Transaction data as subsets



a: bread
b: beer
c: milk
d: diapers
e: eggs

2^n subsets of n items. Layer k has $\binom{n}{k}$ subsets.

Transaction data as binary matrix

TID	Bread	Milk	Diapers	Beer	Eggs
1	1	1	0	0	0
2	1	0	1	1	1
3	0	1	1	1	0
4	1	1	1	1	0
5	1	1	1	0	0

Any data that can be expressed as a binary matrix can be used.

Itemsets, support, and frequency

- An **itemset** is a set of items
  - A transaction t is an itemset with associated transaction ID, $t = (tid, I)$, where I is the set of items of the transaction
- A transaction $t = (tid, I)$ contains itemset X if $X \subseteq I$
- The **support** of itemset X in database D is the number of transactions in D that contain it:

$$supp(X, D) = |\{t \in D : t \text{ contains } X\}|$$

- The **frequency** of itemset X in database D is its support relative to the database size, $supp(X, D) / |D|$
- Itemset is **frequent** if its frequency is above user-defined threshold **minfreq**

Frequent itemset example

TID	Bread	Milk	Diapers	Beer	Eggs
1	1	1	0	0	0
2	1	0	1	1	1
3	0	1	1	1	0
4	1	1	1	1	0
5	1	1	1	0	0

Itemset {Bread, Milk} has support 3 and frequency 3/5
Itemset {Bread, Milk, Eggs} has support and frequency 0
For **minfreq** = 1/2, frequent itemsets are:
{Bread}, {Milk}, {Diapers}, {Beer}, {Bread, Milk}, {Bread, Diapers}, {Milk, Diapers}, and {Diapers, Beer}

Association rules and confidence

- An **association rule** is a rule of type $X \to Y$, where X and Y are disjoint itemsets ($X \cap Y = \emptyset$)
  - If transaction contains itemset X, it (probably) also contains itemset Y
- The **support** of rule $X \to Y$ in data D is

$$supp(X \to Y, D) = supp(X \cup Y, D)$$

  - Tan et al. (and other authors) divide this value by $|D|$
- The **confidence** of rule $X \to Y$ in data D is

$$c(X \to Y, D) = supp(X \cup Y, D)/supp(X, D)$$

  - The confidence is the empirical conditional probability that transaction contains Y given that it contains X

Association rule examples

TID	Bread	Milk	Diapers	Beer	Eggs
1	1	1	0	0	0
2	1	0	1	1	1
3	0	1	1	1	0
4	1	1	1	1	0
5	1	1	1	0	0

{Bread, Milk} → {Diapers} has support 2 and confidence 2/3
{Diapers} → {Bread, Milk} has support 2 and confidence 1/2
{Eggs} → {Bread, Diapers, Beer} has support 1 and confidence 1

Applications

- Frequent itemset mining
  - Which items appear together often?
    - What products people by together?
    - What web pages people visit in some web site?
  - Later we learn better concepts for this
- Association rule mining
  - Implication analysis: If X is bought/observed, what else will probably be bought/observed
    - If people who buy milk and cereal also buy bananas, we can locate bananas close to milk or cereal to improve their sales
    - If people who search for swimsuits and cameras also search for holidays, we should show holiday advertisements for those who've searched swimsuits and cameras

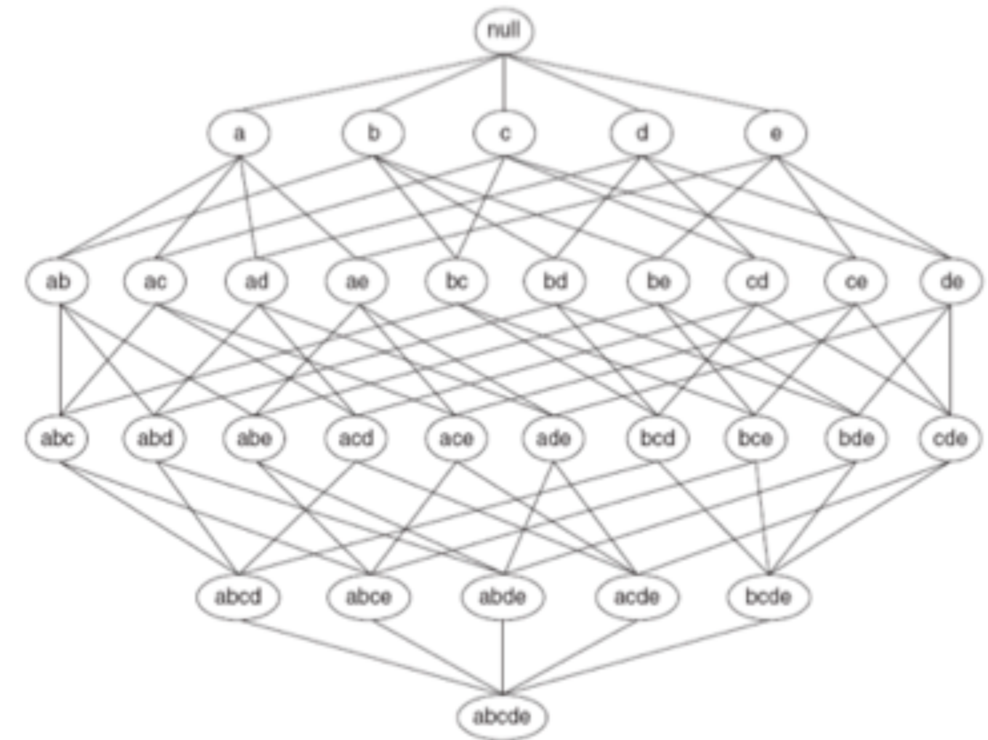
Chapter VII.2: Algorithms

1. The Naïve Algorithm
2. The Apriori Algorithm
    2.1. Key observation: monotonicity of support
3. Improving Apriori: Eclat
4. The FP-Growth Algorithm

Zaki & Meira, Chapter 10; Tan, Steinbach & Kumar, Chapter 6

The Naïve Algorithm

- Try every possible itemset and check is it frequent
- How to try the itemsets?
  - Breath-first in subset lattice
  - Depth-first in subset lattice
- How to compute the support?
  - Check for every transaction is the itemset included



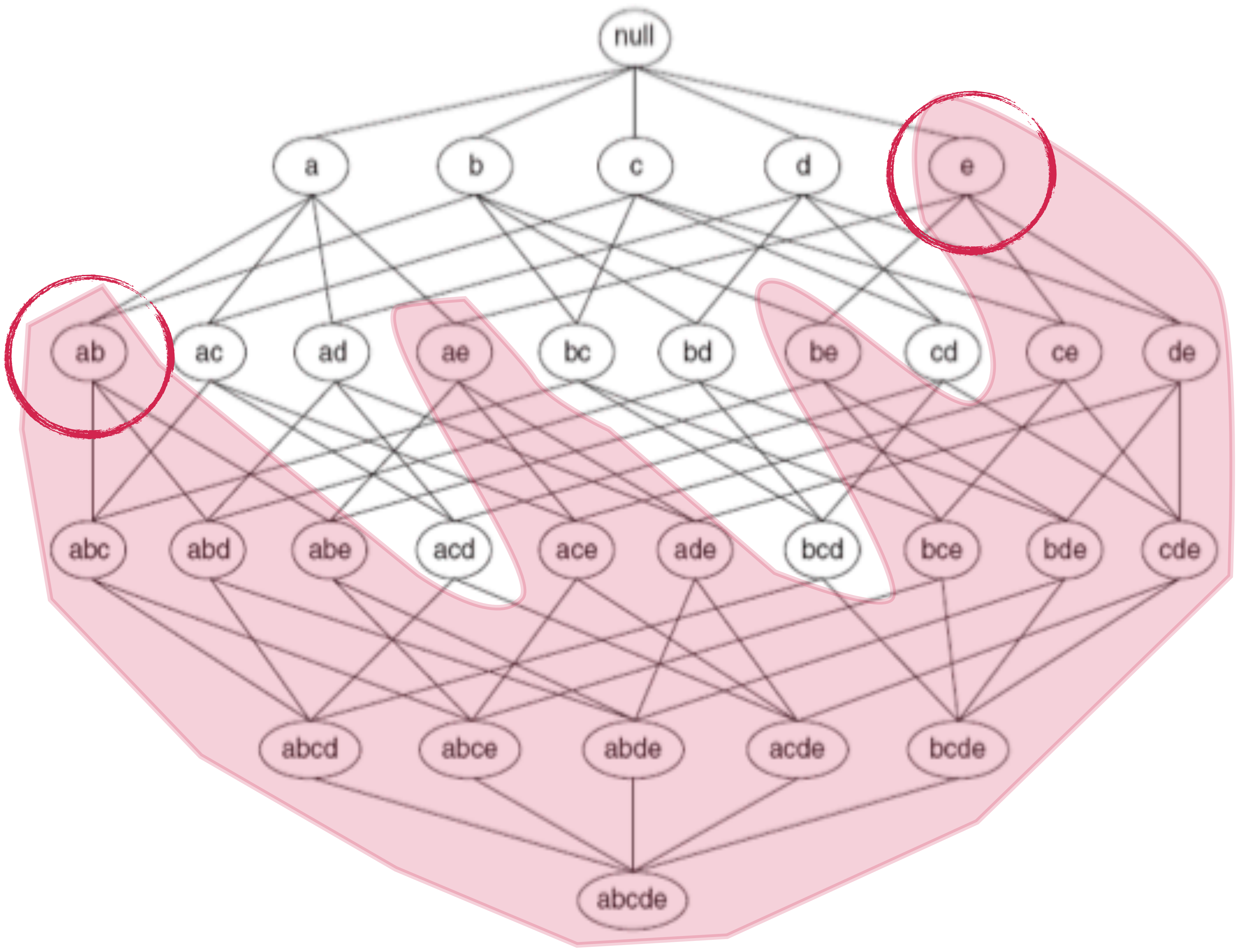
- Time complexity:
  - Computing the support takes $O(|\mathcal{I}|\times|D|)$ and there are $2^{|\mathcal{I}|}$ possible itemsets: worst-case: $O(|\mathcal{I}|\times|D|\times 2^{|\mathcal{I}|})$
  - I/O complexity is $O(2^{|\mathcal{I}|})$ database accesses

The Apriori Algorithm

- **The downward closedness of support:**
  - If X and Y are itemsets s.t. $X \subseteq Y$, then $supp(X) \geq supp(Y)$
    $\Rightarrow$ If X is infrequent, so are all its supersets
- The **Apriori** algorithm uses this feature to significantly reduce the search space
  - Apriori never generates a candidate that has an infrequent subset
- Worst-case time complexity is still the same $O(|\mathcal{I}| \times |D| \times 2^{|\mathcal{I}|})$
  - In practice the time complexity can be much less

Example of pruning itemsets

If {e} and {ab} are infrequent



Improving I/O

- The Naïve algorithm computed the frequency of every candidate itemset
  - Exponential number of database scans
- It's better to loop over the transactions:
  - Collect all candidate *k*-itemsets
  - Iterate over every transaction
    - For every *k*-subitemset of the transaction, if the itemset is a candidate, increase the candidate's support by 1
- This way we only need to sweep thru the data once per level
  - At most $O(|\mathcal{I}|)$ database scans

Example of Apriori (on blackboard)

	A	B	C	D	E
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0
∑	4	6	4	4	5

Improving Apriori: Eclat

- In Apriori, the support computation requires creating all k-subitemsets of all transactions
  - Many of them might not be in the candidate set
- Way to speed up things: index the data base so that we can compute the support directly
  - A **tidset** of itemset X, $\mathbf{t}(X)$, is the set of transaction IDs that contain X, i.e. $\mathbf{t}(X) = \{tid : (tid, I) \in D \text{ is such that } X \subseteq I\}$
    - $supp(X) = |\mathbf{t}(X)|$
    - $\mathbf{t}(XY) = \mathbf{t}(X) \cap \mathbf{t}(Y)$
      - XY is a shorthand notation for $X \cup Y$
- We can compute the support by intersecting the tidsets

The Eclat algorithm

- The **Eclat** algorithm uses tidsets to compute the support
- A **prefix equivalence class** (PEC) is a set of all itemsets that share the same prefix
  - We assume there's some (arbitrary) order of items
  - E.g. all itemsets that contain items A and B
- Eclat merges two itemsets from the same PEC and intersects their tidsets to compute the support
  - If the result is frequent, it is moved down to a PEC with prefix matching the first itemset
- Eclat traverses the prefix tree on DFS-like manner

Example of ECLAT

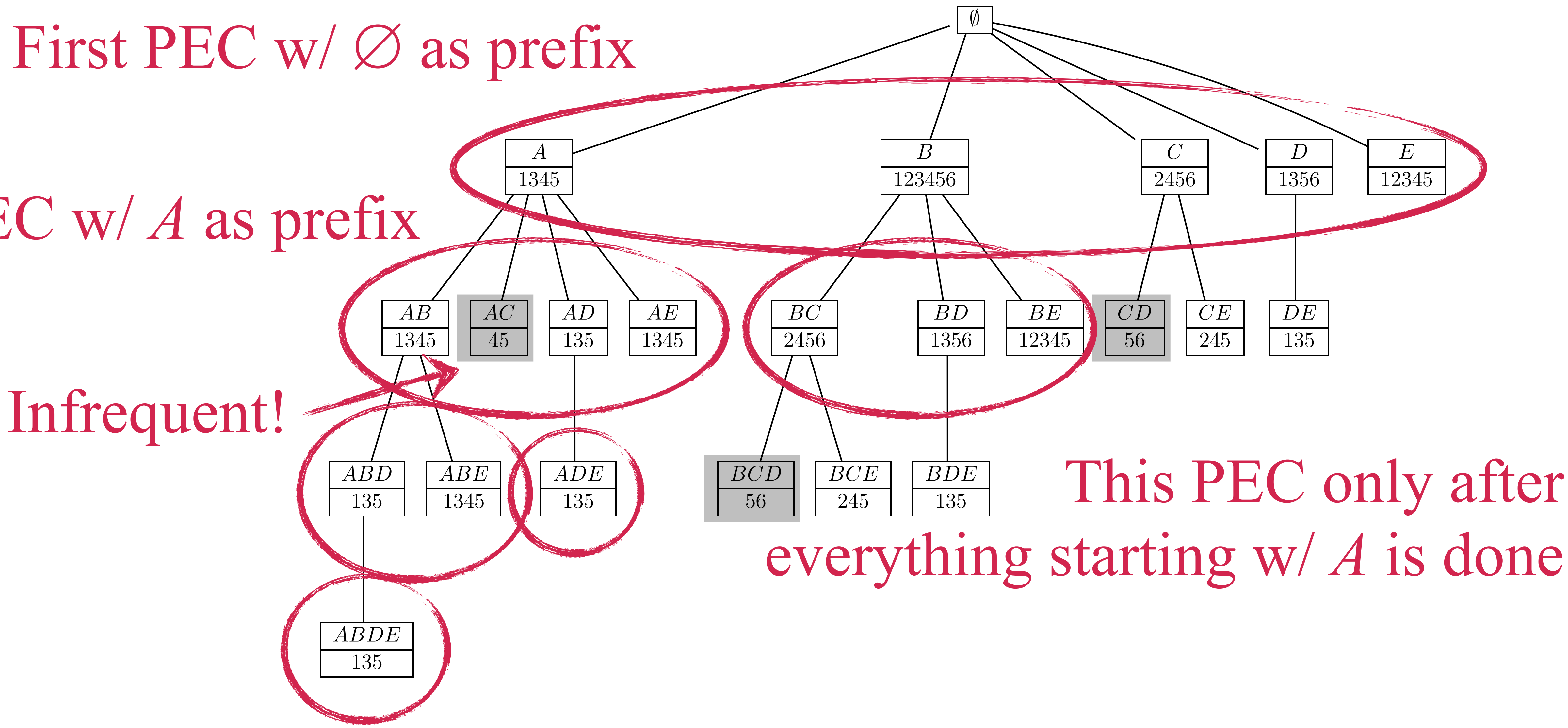


Figure 8.5 of Zaki & Meira

dEclat: Differences of tidsets

- Long tidsets slow down Eclat
- A **diffset** stores the differences of the tidsets
  - The diffset of ABC, $\mathbf{d}(ABC)$, is $\mathbf{t}(AB) \setminus \mathbf{t}(ABC)$
    - E.g. all tids that contain the prefix AB but not ABC
- Updates: $\mathbf{d}(ABC) = \mathbf{d}(C) \setminus \mathbf{d}(AB)$
- Support: $supp(ABC) = supp(AB) - |\mathbf{d}(ABC)|$
- We can replace tidsets with diffsets if they are shorter
  - This replacement can happen at any move to a new PEC in Eclat

The FPGrowth algorithm

- The **FPGrowth** algorithm preprocesses the data to build an **FP-tree** data structure
  - Mining the frequent itemsets is done using this data structure
- An FP-tree is a condensed prefix representation of the data
  - The smaller, the more effective the mining

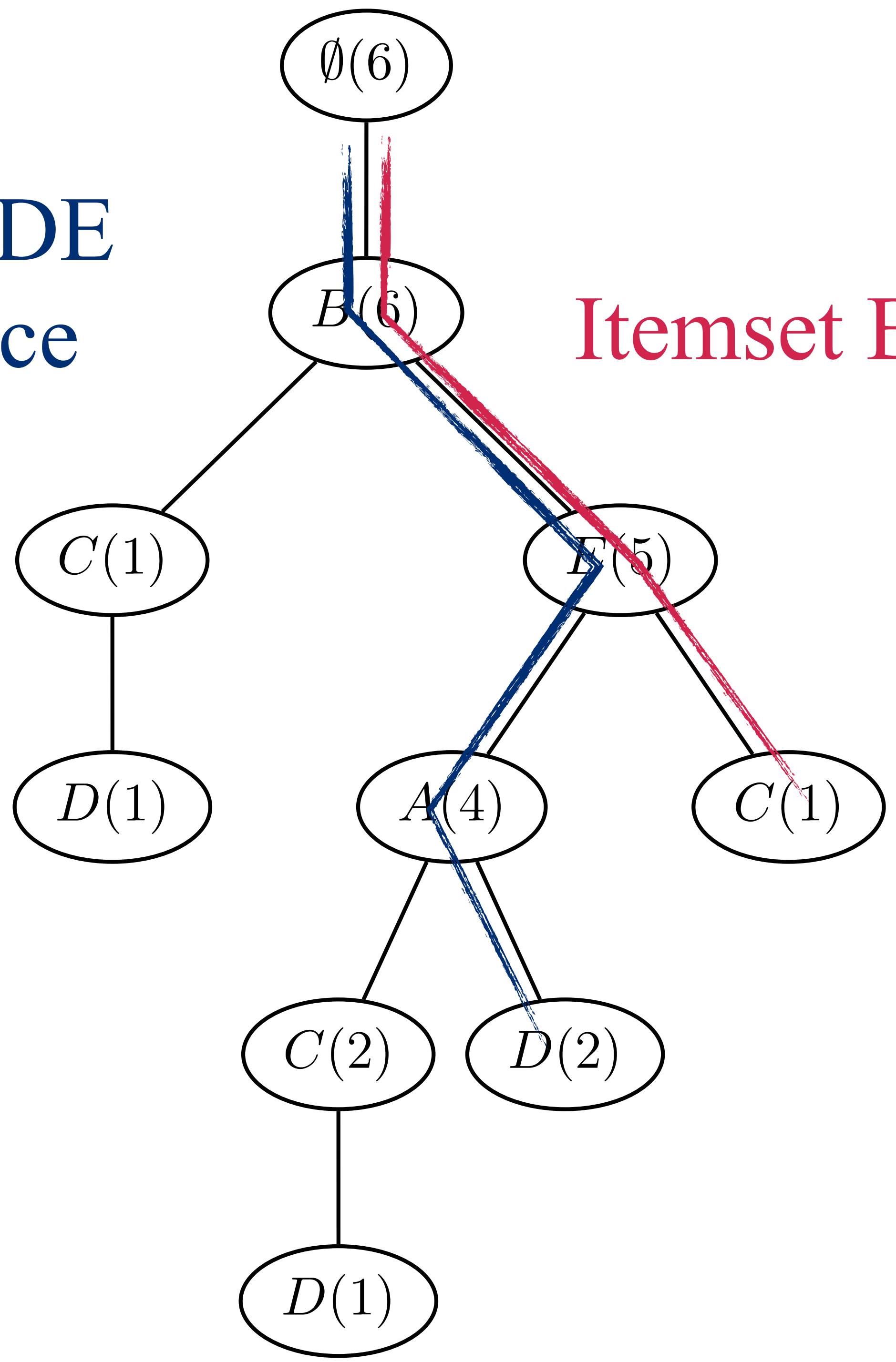
Building an FP-tree

- Initially the tree contains the empty set as a root
- For each transaction, we add a branch that contains one node for each item in the transaction
  - If a prefix of the transaction is already in the tree, we increase the count of the nodes corresponding to the prefix and add only the suffix
    $\Rightarrow$ Every transaction is in a path from the root to a leaf
    - Transactions that are proper subitemsets of other transactions do not reach the leaf
- The items in transactions are added in a decreasing order on support
  - As small tree as possible

FP-tree example

Itemset ABDE appears twice

Itemset BCE



From Figure 8.9 of Zaki & Meira

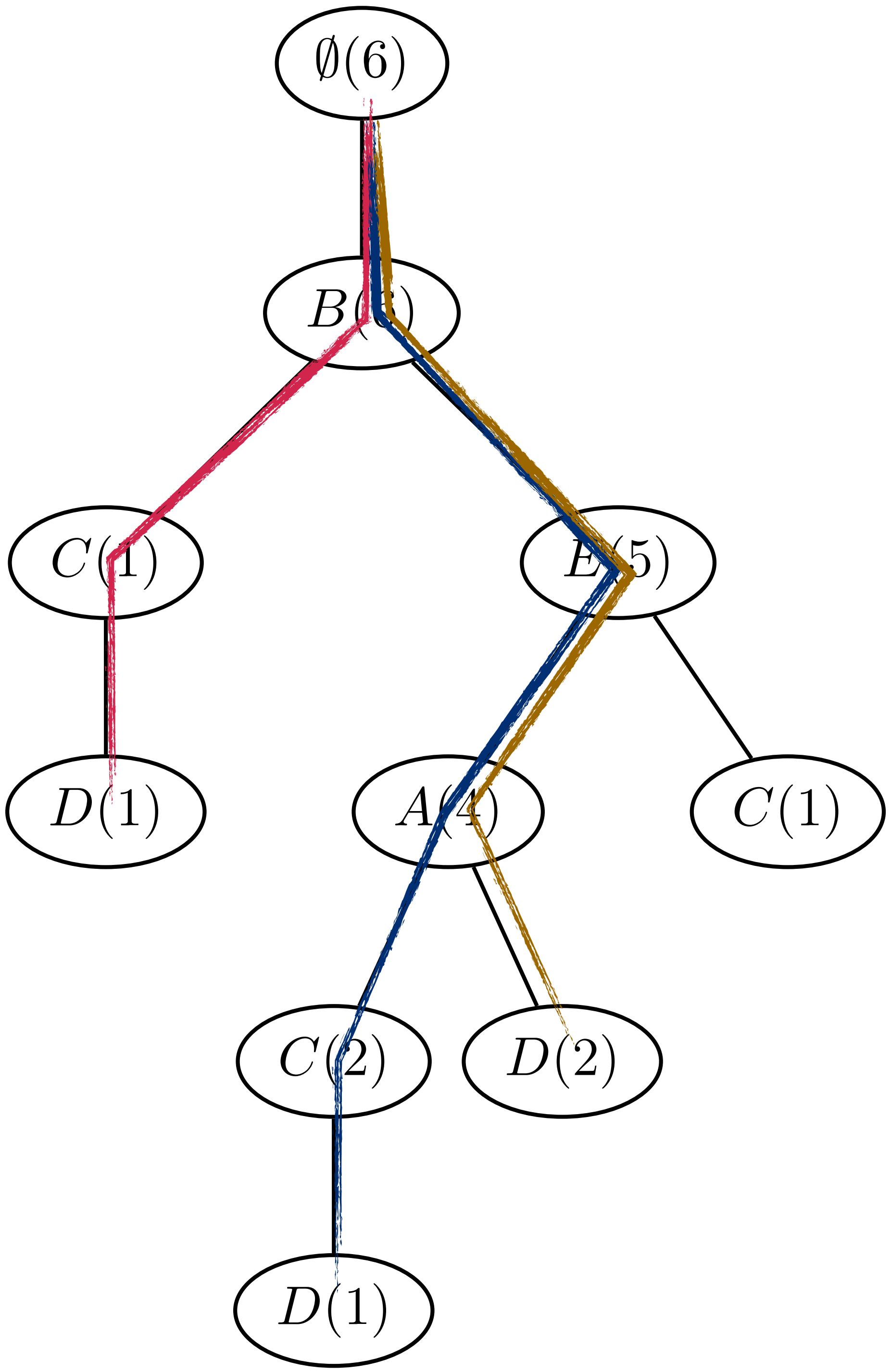
Mining the frequent itemsets

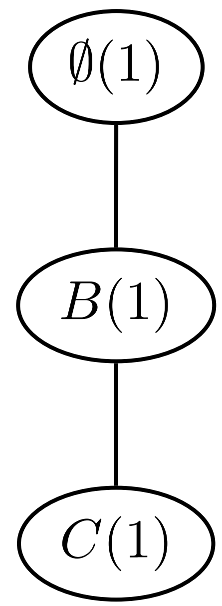
- To mine the itemsets, we *project* the FP-tree onto an itemset prefix
  - Initially these prefixes contain single items in order of increasing support
  - The result is another FP-tree
- If the projected tree is a path, we add all subsets of nodes together with the prefix as frequent itemsets
  - The support is the smallest count
  - If the projected tree is not a path, we call FPGrowth recursively

How to project?

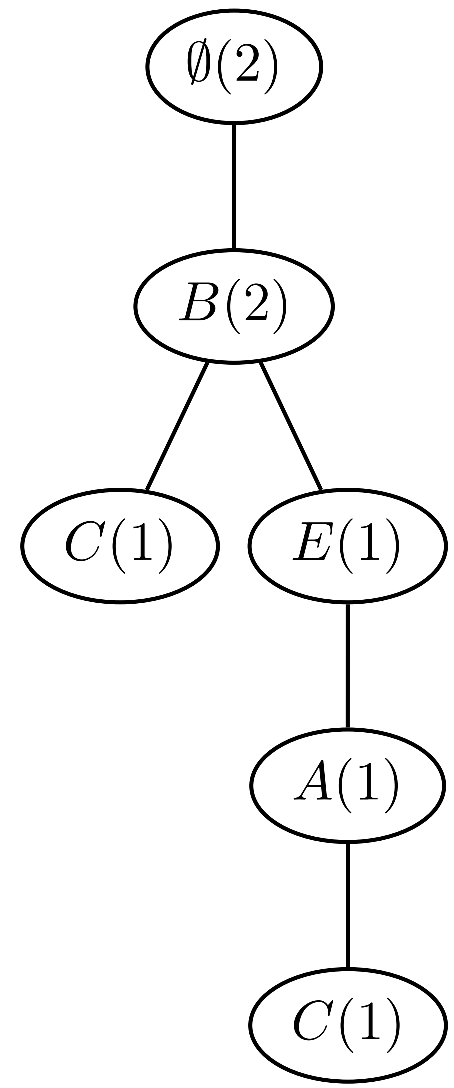
- To project tree *T* to item *i*, we first find all occurrences of *i* from *T*
  - For each occurrence, find the path from the root to the node
  - Copy this path to the projected tree without the node corresponding to *i*
  - Increase the count of every node in the copied path by the count of the node corresponding to *i*
- Item *i* is added to the prefix
- Nodes corresponding to elements with support less than the **minsup** are removed
  - Element's support is the sum of counts in the nodes corresponding to it
- Either call FPGrowth recursively or list the frequent items if the resulting tree is a path
  - If calling FPGrowth, add all itemsets with current prefix and any single item from the tree

Example of projection

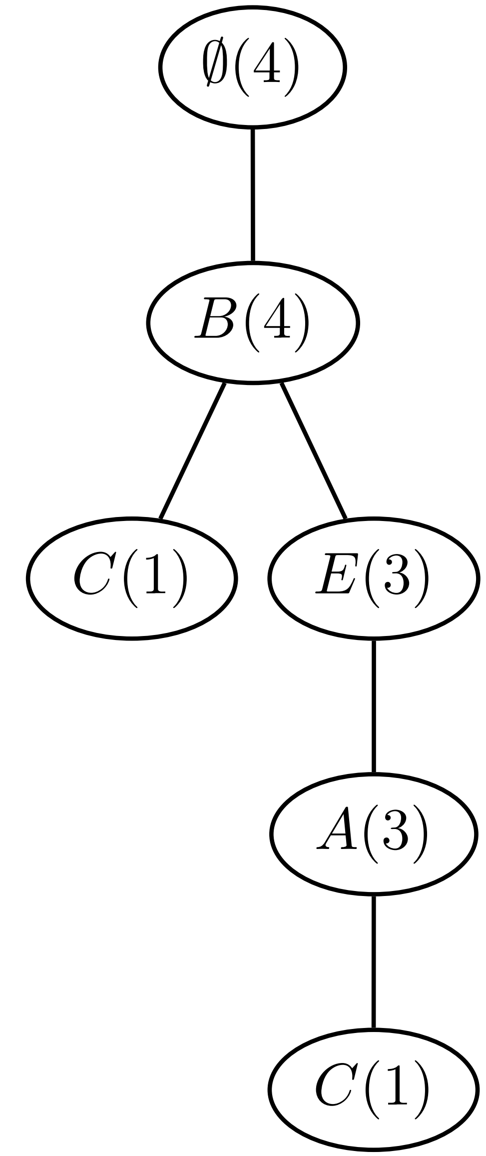




Add BCD
count = 1



Add BEACD
count = 1

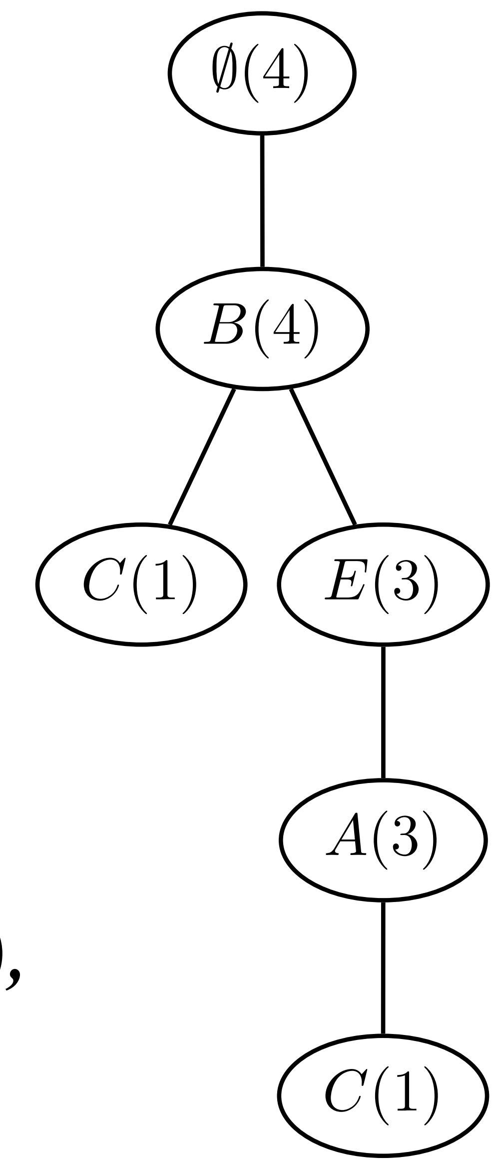


Add BEAD
count = 2

From Figures 8.8 & 8.9 of Zaki & Meira

Example of finding frequent itemsets

- The tree projected onto prefix *D*
- Nodes with *C* are infrequent
  - Can be removed
- The result is a path
  ⇒ Frequent itemsets are all subsets of nodes with prefix *D*
  - Support is the smallest count
  - *DB* (4), *DE* (3), *DA* (3), *DBE* (3), *DBA* (3), *DEA* (3), and *DBEA* (3)
- Similar process is done to other prefixes, with possibly recursive calls



From Figure 8.8 of Zaki & Meira