Generating frequent itemsets

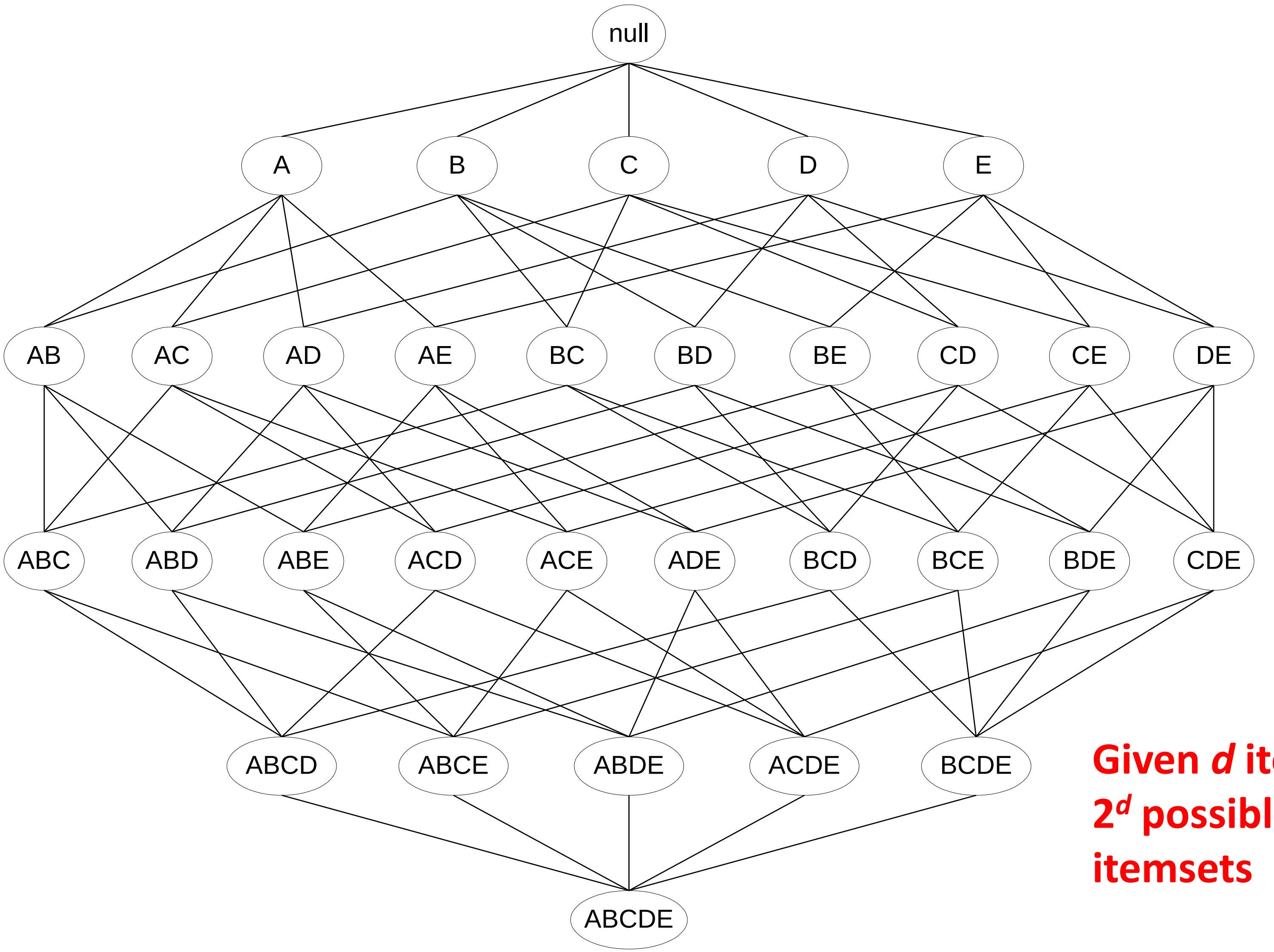
Lecture 13

Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets whose support $\geq$ minsup (these itemsets are called *frequent itemset*)
  2. Rule Generation
     - Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset (these rules are called *strong rules*)

We focus first on **frequent itemset generation**.

Candidates for frequent itemsets



Given *d* items, there are 2^d possible candidate itemsets

Frequent Itemset Generation: brute force

- Each itemset in the lattice is a candidate frequent itemset
- Count the support of each candidate by scanning the database
- Match each transaction against every candidate
- Complexity ~ $O(NMw)$ => Expensive since $M = 2^d$!!!
  - w is max transaction width.

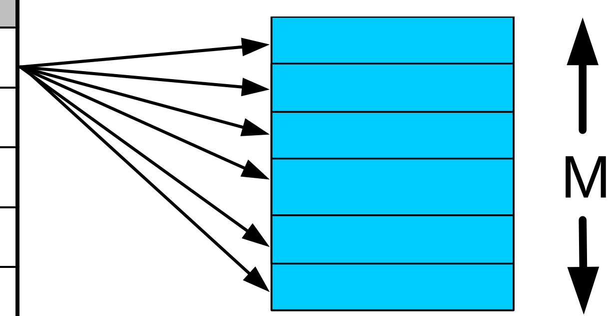
Transactions

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

N

w

List of Candidates



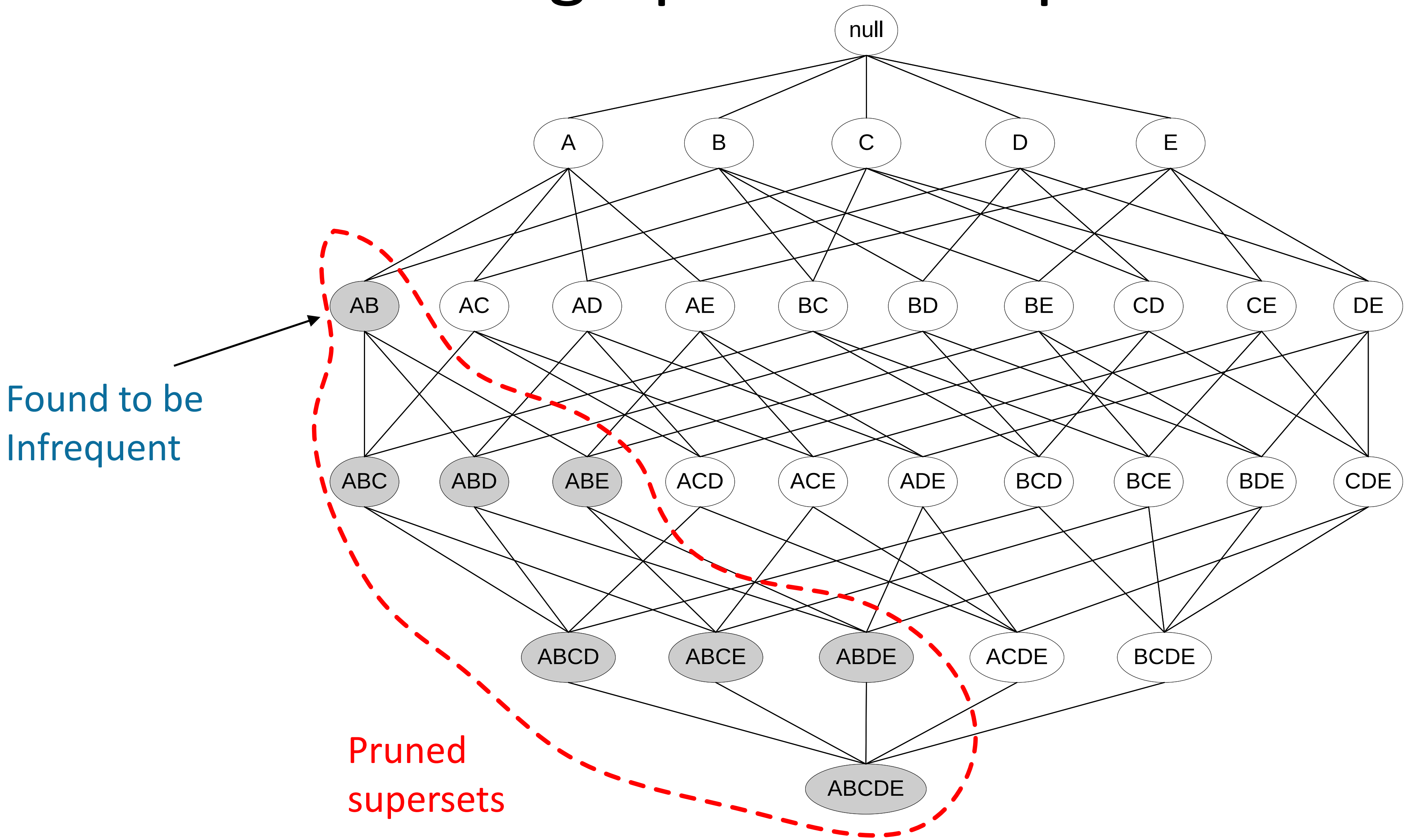
Frequent itemset generation: Apriori algorithm

- The name *Apriori* is based on the fact that we use *prior* knowledge about *k*-itemsets in order to prune candidate *k*+1-itemsets
- The idea: level-wise processing
  - find frequent 1-itemsets: F_1
  - F_1 is used to find F_2
  - F_k is used to find F_{k+1}
- The efficiency is based on *anti-monotone* property of support: if a set cannot pass the test, all its supersets will fail the same test

Apriori principle

- All subsets of a frequent itemset A must also be frequent
- If itemset A appears in less than *minsup* fraction of transactions, then itemset A with one more item added cannot occur more frequently than A. Therefore, if A is not frequent, all its supersets are not frequent as well

Illustrating Apriori Principle



Illustrating Apriori Principle

Item	Count
Bread	**4**
Coke	**2**
Milk	**4**
Beer	**3**
Diaper	**4**
Eggs	**1**

Items (1-itemsets)

Minimum support count = 3

Itemset	Count
{Bread,Milk}	**3**
{Bread,Beer}	**2**
{Bread,Diaper}	**3**
{Milk,Beer}	**2**
{Milk,Diaper}	**3**
{Beer,Diaper}	**3**

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	**3**

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 41$
With support-based pruning,
$6 + 6 + 1 = 13$

With the Apriori principle we need to keep only this triplet, because it's the only one whose subsets are all frequent.

Apriori Algorithm

- Let $k=1$
- Generate set F_1 of frequent 1-itemsets
- Repeat until F_k is empty
    - $k=k+1$
    - **Generate** length-k candidate itemsets C_k from length-k-1 frequent itemsets F_{k-1}
    - **Prune** candidate itemsets containing subsets of length-k-1 that are infrequent
    - **Count** the support of each candidate in C_k by scanning the DB and eliminate candidates that are infrequent, leaving only those that are frequent - F_k

Candidate generation and prunning

Many ways to generate candidate itemsets.
An effective candidate generation procedure:

1. Should avoid generating too many unnecessary candidates.
   - A candidate itemset is unnecessary if at least one of its subsets is infrequent.
2. Must ensure that the candidate set is complete,
   - i.e., no frequent itemsets are left out by the candidate generation procedure.
3. Should not generate the same candidate itemset more than once.
   - E.g., the candidate itemset {a, b, c, d} can be generated in many ways---
     - by merging {a, b, c} with {d},
     - {c} with {a, b, d}, etc.

Generating C_{k+1} from F_k: brute force

- A bruteforce method considers every frequent *k*-itemset as a potential candidate and then applies the candidate pruning step to remove any unnecessary candidates.

Items

Item
Beer
Bread
Cola
Diapers
Milk
Eggs

→

Candidate Generation

Itemset
{Beer, Bread, Cola}
{Beer, Bread, Diapers}
{Beer, Bread, Milk}
{Beer, Bread, Eggs}
{Beer, Cola, Diapers}
{Beer, Cola, Milk}
{Beer, Cola, Eggs}
{Beer, Diapers, Milk}
{Beer, Diapers, Eggs}
{Beer, Milk, Eggs}
{Bread, Cola, Diapers}
{Bread, Cola, Milk}
{Bread, Cola, Eggs}
{Bread, Diapers, Milk}
{Bread, Diapers, Eggs}
{Bread, Milk, Eggs}
{Cola, Diapers, Milk}
{Cola, Diapers, Eggs}
{Cola, Milk, Eggs}
{Diapers, Milk, Eggs}

→

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

$F_{k-1} \times F_1$ Method

- Extend each frequent $(k - 1)$itemset with a frequent 1-itemset.

- Is it complete?

The procedure is complete because every frequent k--itemset is composed of a frequent $(k - 1)$itemset and a frequent 1-itemset.

- However, it doesn't prevent the same candidate itemset from being generated more than once.

E.g., {Bread, Diapers, Milk} can be generated by merging

  - {Bread, Diapers} with {Milk},
  - {Bread, Milk} with {Diapers}, or
  - {Diapers, Milk} with {Bread}.

Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Frequent 1-itemset

Item
Beer
Bread
Diapers
Milk

Lexicographic Order

- Avoid generating duplicate candidates by ensuring that the items in each frequent itemset are kept sorted in their lexicographic order.
- Each frequent (k-1)-itemset X is then extended with frequent items that are lexicographically larger than the items in X.
- For example, the itemset {Bread, Diapers} can be augmented with {Milk} since Milk is lexicographically larger than Bread and Diapers.
- However, we don't augment {Diapers, Milk} with {Bread} nor {Bread, Milk} with {Diapers} because they violate the lexicographic ordering condition.
- Is it complete?

Lexicographic Order - Completeness

- Is it complete?

Let $(i_1,\ldots, i_{k-1}, i_k)$ be a frequent k-itemset sorted in lexicographic order.

Since it is frequent, by the Apriori principle, $(i_1,\ldots, i_{k-1})$ and (i_k) are frequent as well.

$(i_1,\ldots, i_{k-1}) \in F_{k-1}$ and $(i_k) \in \mathrm{F}_1$.

Since, (i_k) is lexicographically bigger than $i_1,\ldots, i_{k-1}$, we have that $(i_1,\ldots, i_{k-1})$ would be joined with (i_k) for giving $(i_1,\ldots, i_{k-1}, i_k)$ as a candidate k-itemset.

Still too many candidates…

- E.g. merging {Beer, Diapers} with {Milk} is unnecessary because one of its subsets, {Beer, Milk}, is infrequent.

- For a candidate *k*-itemset to be worthy,
  - every item in the candidate must be contained in at least *k*-1 of the frequent (*k*-1)-itemsets.
  - {Beer, Diapers, Milk} is a viable candidate 3-itemset only if every item in the candidate, including Beer, is contained in at least 2 frequent 2itemsets.

    Since there is only one frequent 2-itemset containing Beer, all candidate 3-itemsets involving Beer must be infrequent.
- Why?

Because each of *k*-1-subsets containing an item must be frequent.

$F_{k\text{-}1} \times F_1$

Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Frequent 1-itemset

Item
Beer
Bread
Diapers
Milk

Candidate Generation

Itemset
{Beer, Diapers, Bread}
{Beer, Diapers, Milk}
{Bread, Diapers, Milk}
{Bread, Milk, Beer}

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

$F_{k-1} \times F_{k-1}$ Method

- Merge a pair of frequent (k-1)-itemsets only if their first k-2 items are identical.
  - E.g. frequent itemsets {Bread, Diapers} and {Bread, Milk} are merged to form a candidate 3itemset {Bread, Diapers, Milk}.
  - We don't merge {Beer, Diapers} with {Diapers, Milk} because the first item in both itemsets is different.
  - Indeed, if {Beer, Diapers, Milk} is a viable candidate, it would have been obtained by merging {Beer, Diapers} with {Beer, Milk} instead.

- This illustrates both the completeness of the candidate generation procedure and the advantages of using lexicographic ordering to prevent duplicate candidates.

Pruning?

- Because each candidate is obtained by merging a pair of frequent (k-1) -itemsets, an additional candidate pruning step is needed to ensure that the remaining k-2 subsets of k-1 elements are frequent.

$F_{k-1} \times F_{k-1}$

Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Frequent 2-itemset

Itemset
{Beer, Diapers}
{Bread, Diapers}
{Bread, Milk}
{Diapers, Milk}

Candidate Generation

Itemset
{Bread, Diapers, Milk}

Candidate Pruning

Itemset
{Bread, Diapers, Milk}

Example: Apriori candidate generation

Find all frequent itemsets from the following data.
Min support count threshold=2

Pizza toppings dataset

TID	Extra cheese	Onions	Peppers	Mushrooms	Olives	Anchovy
1	1	1			1	
2			1	1		
3		1				1
4	1			1		
5	1	1		1	1	
6	1	1		1		

Binary data format

2. Count 1-item frequent itemsets

TID	A	B	C	D	E	F
1	1	1			1	
2			1	1		
3		1				1
4	1			1		
5	1	1		1	1	
6	1	1		1		
σ	4	4	1	4	2	1

Support count

Frequent 1-itemsets: {A}, {B}, {D}, {E}

3. Generate candidate 2-itemsets

	A	B	D	E
A				
B				
D				
E				

Candidate 2-itemsets C_2
{A,B} {A,D} {A,E}
{B,D} {B,E}
{D,E}

4. Scan DB, count candidates

TID	A	B	C	D	E	F
1	1	1			1	
2			1	1		
3		1				1
4	1			1		
5	1	1		1	1	
6	1	1		1		

	A	B	D	E
A		3	3	2
B			2	2
D				1
E				

Frequent 2-itemsets F_2

{A,B} {A,D} {A,E}

{B,D} {B,E}

~~{D,E}~~

2 ways of candidate generation

a) $C_k = F_k \times F_1$
b) $C_k = F_{k-1} \times F_{k-1}$

In both cases itemsets are lexicographically sorted: we may extend existing itemset only with an item which is lexicographically largest among all items in F_{k-1}

5a. Generate $C_3=F_2xF_1$

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

Frequent 1-itemsets:
{A}, {B}, {D}, {E}

$F_2\backslash F_1$	A	B	D	E
A,B				
A,D				
A,E				
B,D				
B,E				

5a. Generate $C_3=F_2 x F_1$

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

Frequent 1-itemsets:
{A}, {B}, {D}, {E}

$F_2 \backslash F_1$	A	B	D	E
A,B				
A,D				
A,E				
B,D				
B,E				

Candidate 3-itemsets C_3
{A,B,D} {A,B,E} {A,D,E} {B,D,E}

5b. Generate $C_3=F_2xF_2$

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

The first item should be identical in order to join

$F_2\backslash F_2$	A,B	A,D	A,E	B,D	B,E
A,B					
A,D					
A,E					
B,D					
B,E					

5b. Generate $C_3=F_2xF_2$

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

The first item should be identical in order to join

$F_2\backslash F_2$	A,B	A,D	A,E	B,D	B,E
A,B					
A,D					
A,E					
B,D					
B,E					

Candidate 3-itemsets C_3
{A,B,D} {A,B,E} {A,D,E} {B,D,E}

6a. Prune C_3 before counting

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

Frequent 1-itemsets:
{A}, {B}, {D}, {E}

$F_2 \backslash F_1$	A	B	D	E
A,B				
A,D				
A,E				
B,D				
B,E				

Candidate 3-itemsets C_3
{A,B,D} {A,B,E} {A,D,E} {B,D,E}

6. Prune C_3 before counting

Frequent 2-itemsets F_2
{A,B} {A,D} {A,E}
{B,D} {B,E}

Frequent 1-itemsets:
{A}, {B}, {D}, {E}

$F_2 \backslash F_1$	A	B	D	E
A,B				
A,D				
A,E				
B,D				
B,E				

Candidate 3-itemsets C_3
{A,B,D} {A,B,E} ~~{A,D,E} {B,D,E}~~

7. Count candidates

TID	A	B	C	D	E	F
1	1	1			1	
2			1	1		
3		1				1
4	1			1		
5	1	1		1	1	
6	1	1		1		

$F_2 \backslash F_1$	A	B	D	E
A,B			2	2
A,D				
A,E				
B,D				
B,E				

Frequent 3-itemsets F_3
{A,B,D} {A,B,E}

8a. Generate candidates $C_4=F_3xF_1$

$F_3 \backslash F_1$	A	B	D	E
A,B,D				
A,B,E				

The only candidate 4-itemset:
{A,B,D,E}
Do we need to count its support?
Can it be pruned?

Solution: all frequent *k*-itemsets, *k*>=2

- {A,B} {A,D} {A,E} {B,D} {B,E}
- {A,B,D} {A,B,E}

Apriori Algorithm. Summary

- Generate F_1
- Let k=1
- Repeat until F_k is empty
    - k=k+1
    - **Generate** C_k from F_{k-1}
    - **Prune** C_k containing subsets that are not in F_{k-1}
    - **Count** support of each candidate in C_k by scanning DB
    - **Eliminate** infrequent candidates, leaving F_k

Reduces the number of candidates to be counted against the database

Counting candidates

- Generate F_1
- Let k=1
- Repeat until F_k is empty
  - $k=k+1$
  - **Generate** C_k from F_{k-1}
  - **Prune** C_k containing subsets that are not in F_{k-1}
  - **Count** support of each candidate in C_k by scanning DB
  - **Eliminate** infrequent candidates, leaving F_k

Goal: to reduce the number of comparisons by avoiding matching each candidate against each transaction

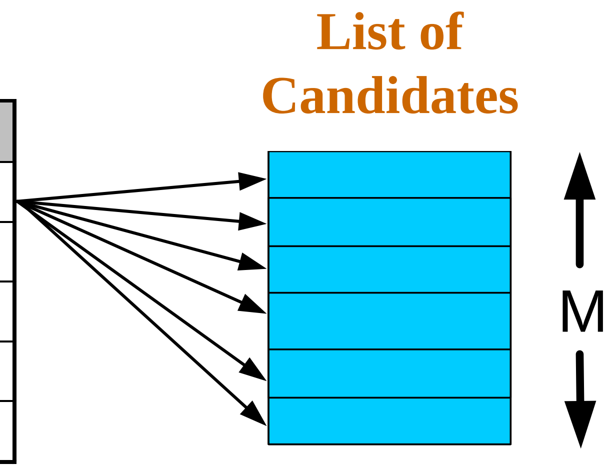
Counting candidates: brute-force

- For each transaction: loop through all candidates and increment count if a candidate is found in the transaction

Transactions

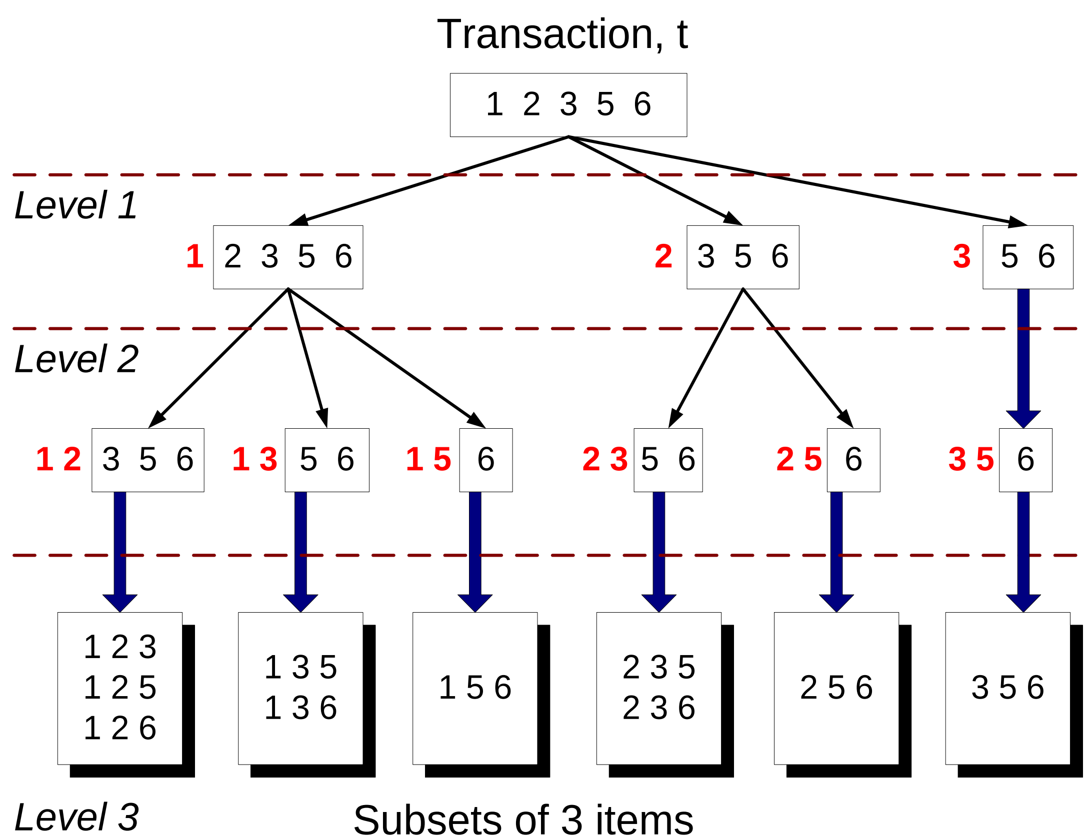
TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

N W



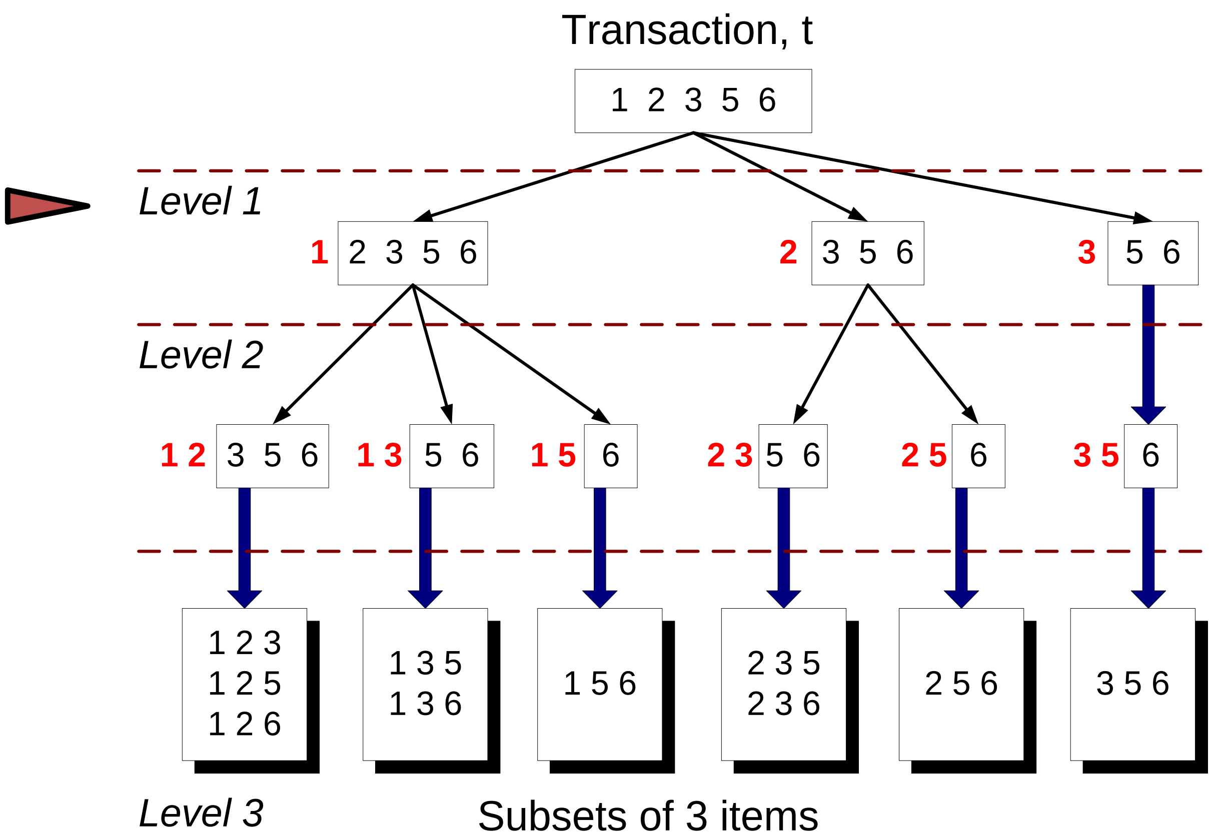
Counting candidates: enumerating items in transaction

- For a transaction of 6 items the number of possible 3-itemsets is $C_{3,5}=10$. If the number of candidates is significantly larger than transaction width, we enumerate all possible k-itemsets in each transaction and increment support count only for the corresponding candidates



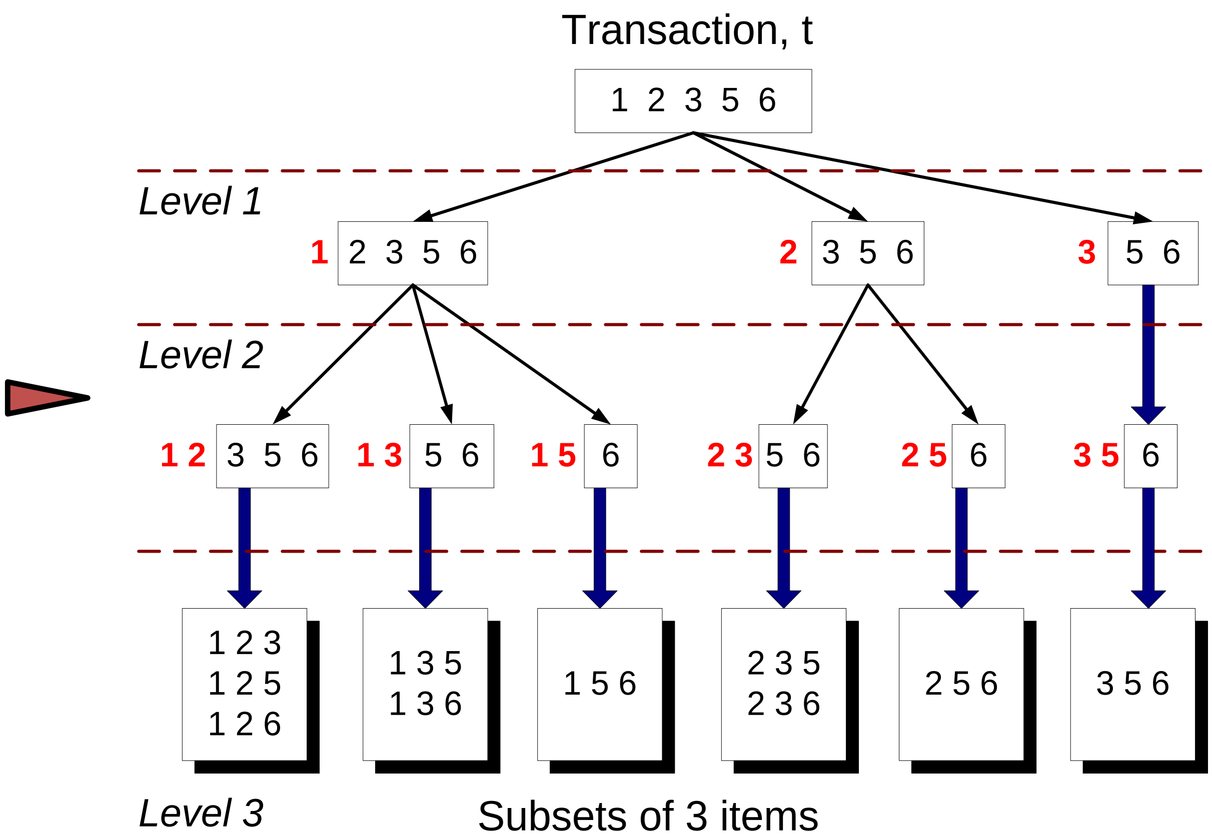
Counting candidates: enumerating items in transaction

- All 3-itemsets must begin with 1,2, or 3. Why?



Counting candidates: enumerating items in transaction

- The number of ways to select a second item: 1 can be followed by 2,3, or 5. Why not 6?



Matching enumerated itemsets to candidates: hash tree

- At each level of Apriori algorithm, candidates are hashed into separate buckets. The enumerated itemsets in each transaction are also hashed using the same hashing function. The comparison is only within several buckets, instead of the entire candidate set.

Matching enumerated itemsets to candidates: hash tree

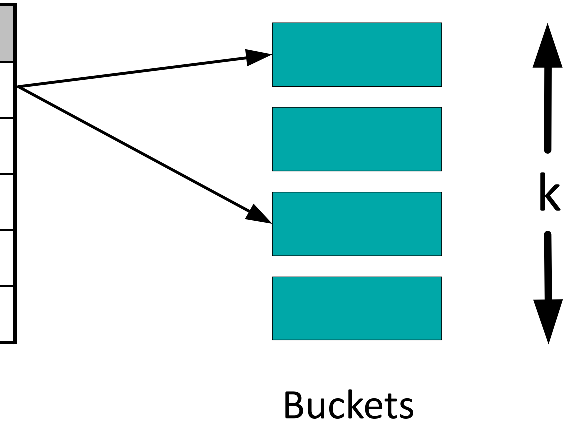
- At each level of Apriori algorithm, candidates are hashed into separate buckets. The enumerated itemsets in each transaction are also hashed using the same hashing function. The comparison is only within several buckets, instead of the entire candidate set.

Transactions

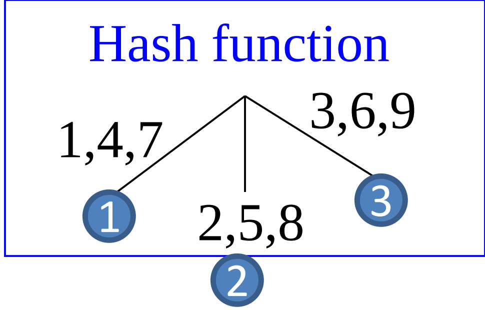
N

TID	*Items*
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Hash Structure



Generate Hash Tree

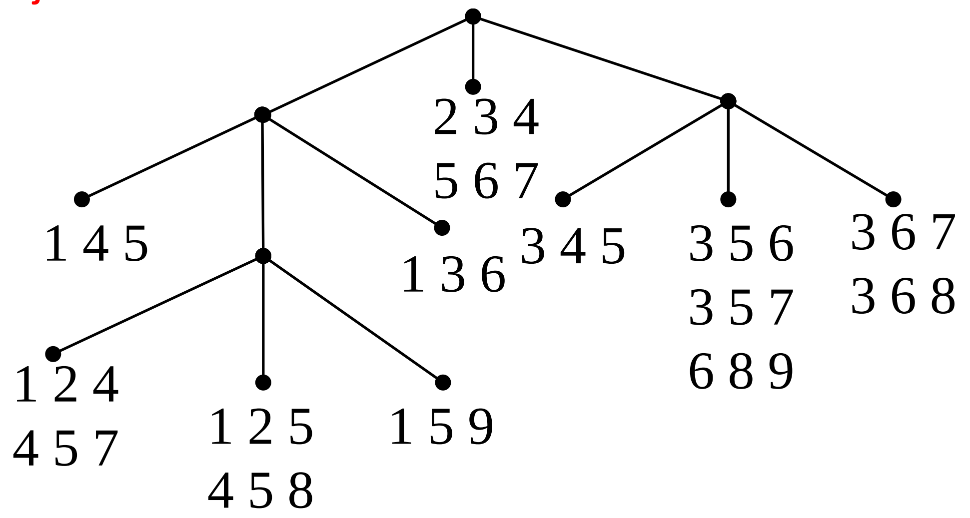


You need:

- A hash function (e.g. *h(p)=p mod 3*)
- Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)

Suppose you have 15 candidate itemsets of length 3 and leaf size is 3:

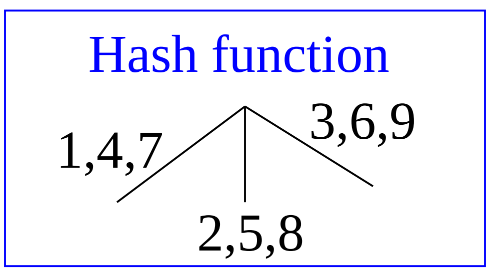
{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

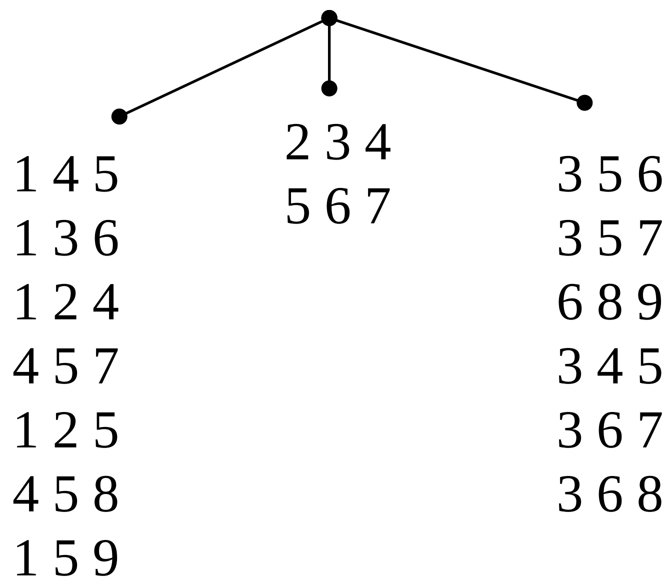


Generate Hash Tree

Suppose you have 15 candidate itemsets of length 3 and leaf size is 3:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}



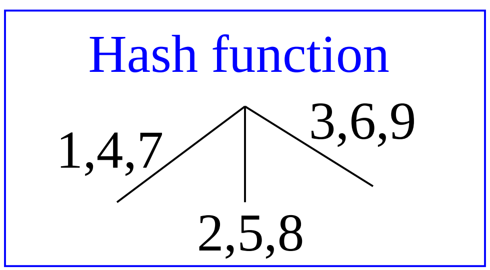


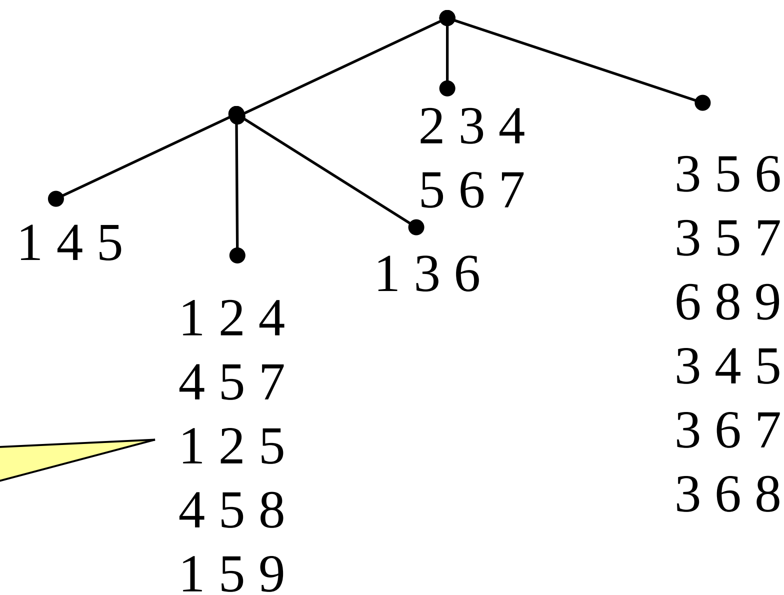
Split nodes with more than 3 candidates using the second item

Generate Hash Tree

Suppose you have 15 candidate itemsets of length 3 and leaf size is 3:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}



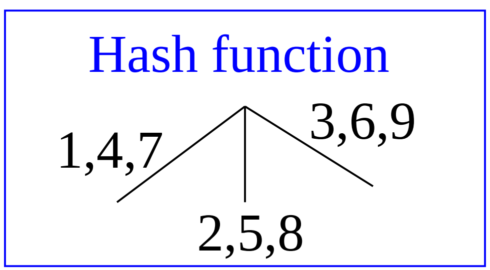


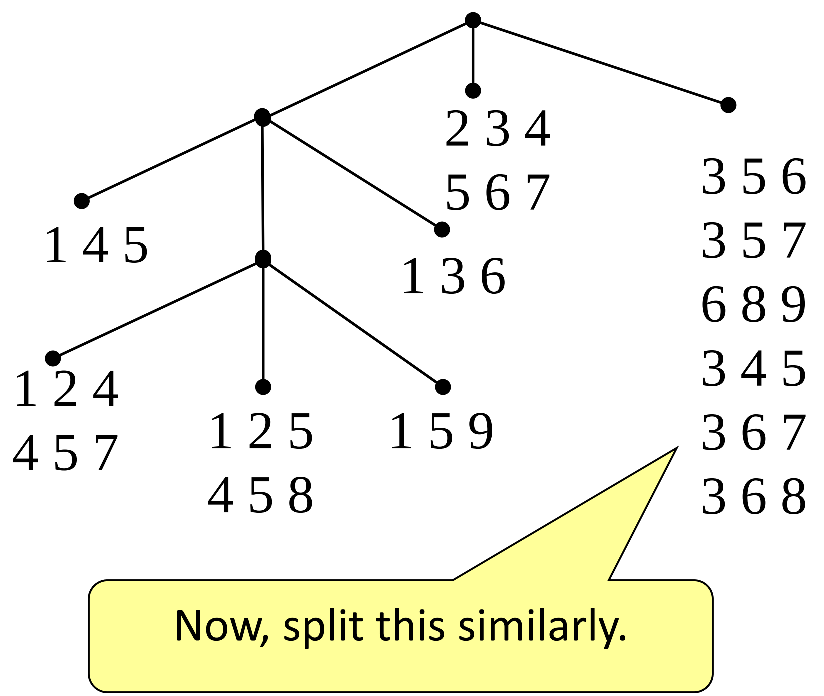
Now split nodes using the third item

Generate Hash Tree

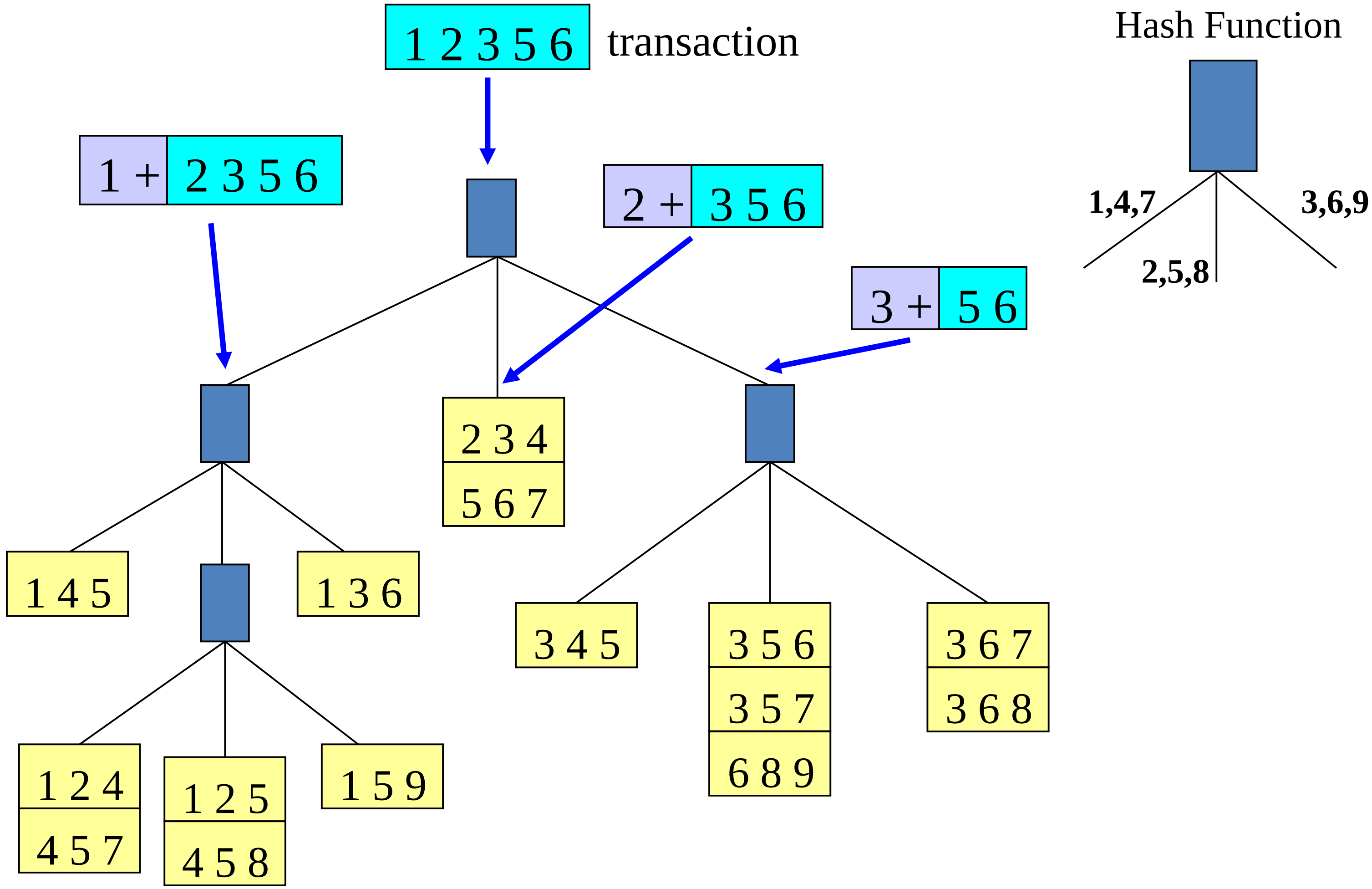
Suppose you have 15 candidate itemsets of length 3 and leaf size is 3:

{1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

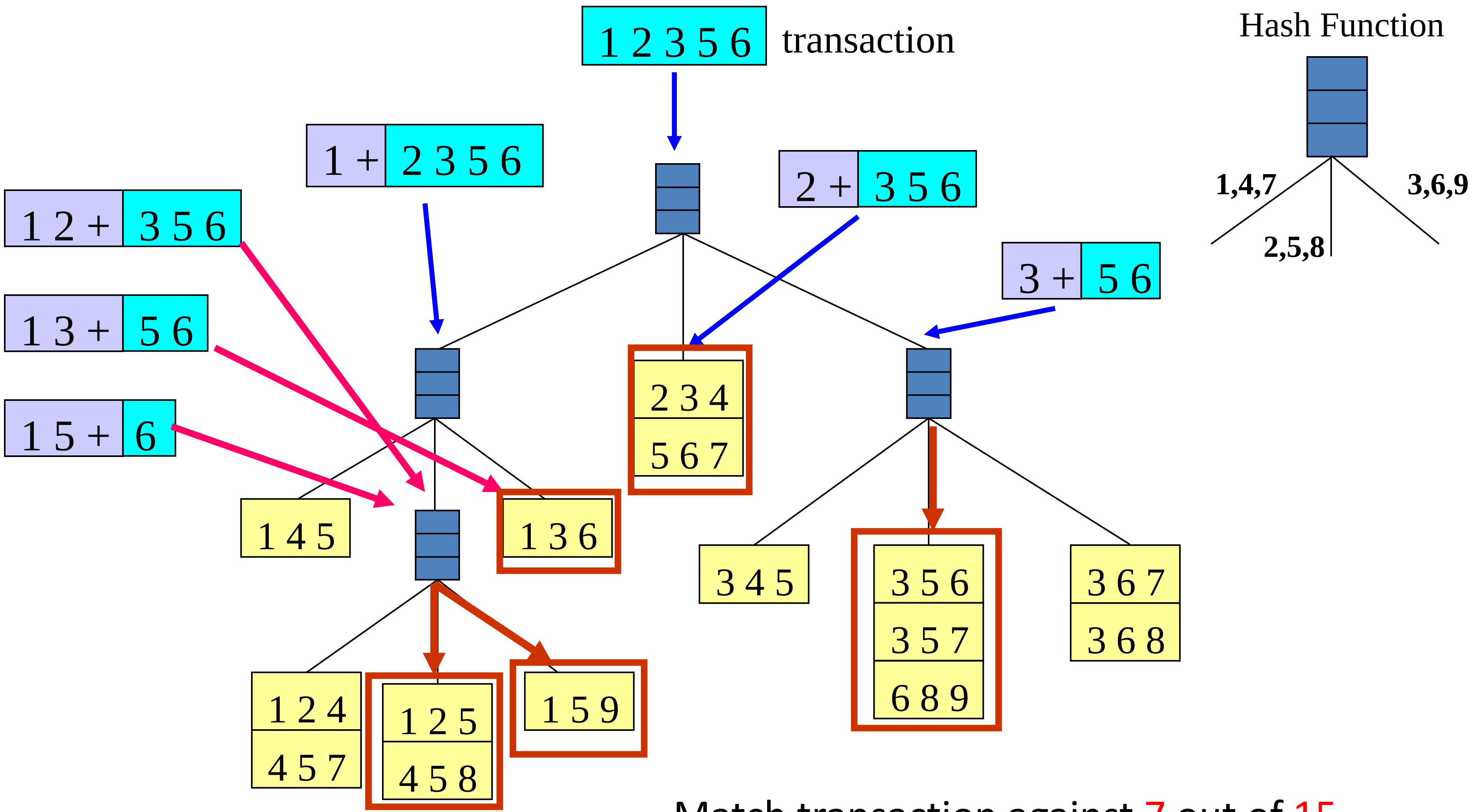




Matching transaction items to the hash tree



Matching transaction items to the hash tree



Match transaction against 7 out of 15 candidates

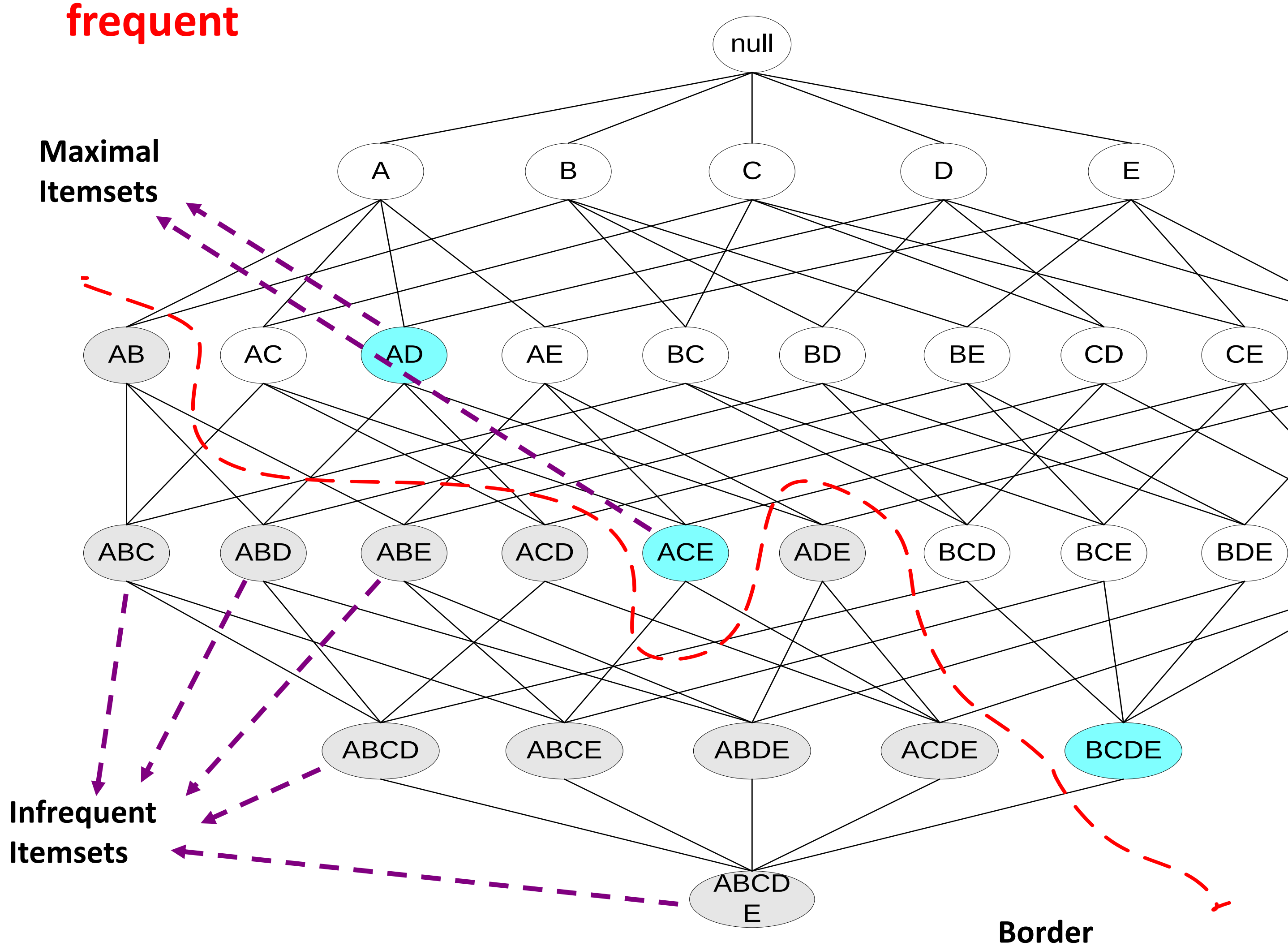
Compact Representation of Frequent Itemsets

- Representative set of frequent itemsets, from which all other frequent itemsets can be derived

  - *Maximal* frequent itemsets
  - *Closed* frequent itemsets

Maximal Frequent Itemsets

An itemset is maximal frequent if none of its immediate supersets is frequent



Maximal frequent itemsets form the **smallest set** of itemsets from which all frequent itemsets can be derived.

Maximal Frequent Itemsets

- Despite providing a compact representation, maximal frequent itemsets do not contain the support information of their subsets.
  - For example, the support of the maximal frequent itemsets {a, c, e}, {a, d}, and {b,c,d,e} do not provide any hint about the support of their subsets.

- An additional pass over the data set is therefore needed to determine the support counts of the nonmaximal frequent itemsets.

- It might be desirable to have a minimal representation of frequent itemsets that preserves the support information.

Closed frequent itemsets

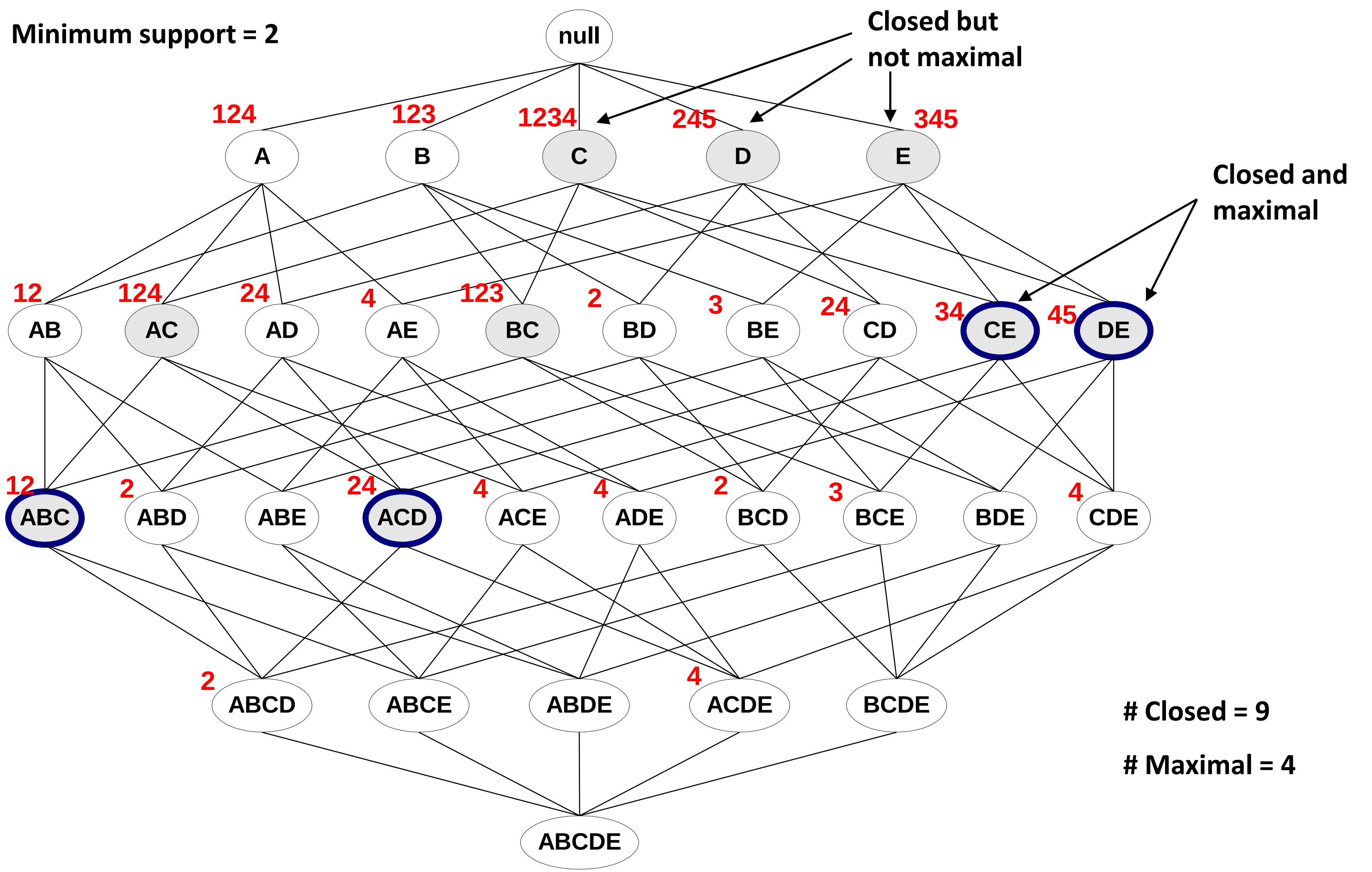
- An itemset *Y* is **closed** if none of its immediate supersets has the same support count as *Y*.
  - Put another way, an itemset *X* is not closed if at least one of its immediate supersets has the same support count as *X*.
- An itemset is a **closed frequent itemset** if it is closed and its support is greater than or equal to minsup count.

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	3
{A,B,C,D}	2

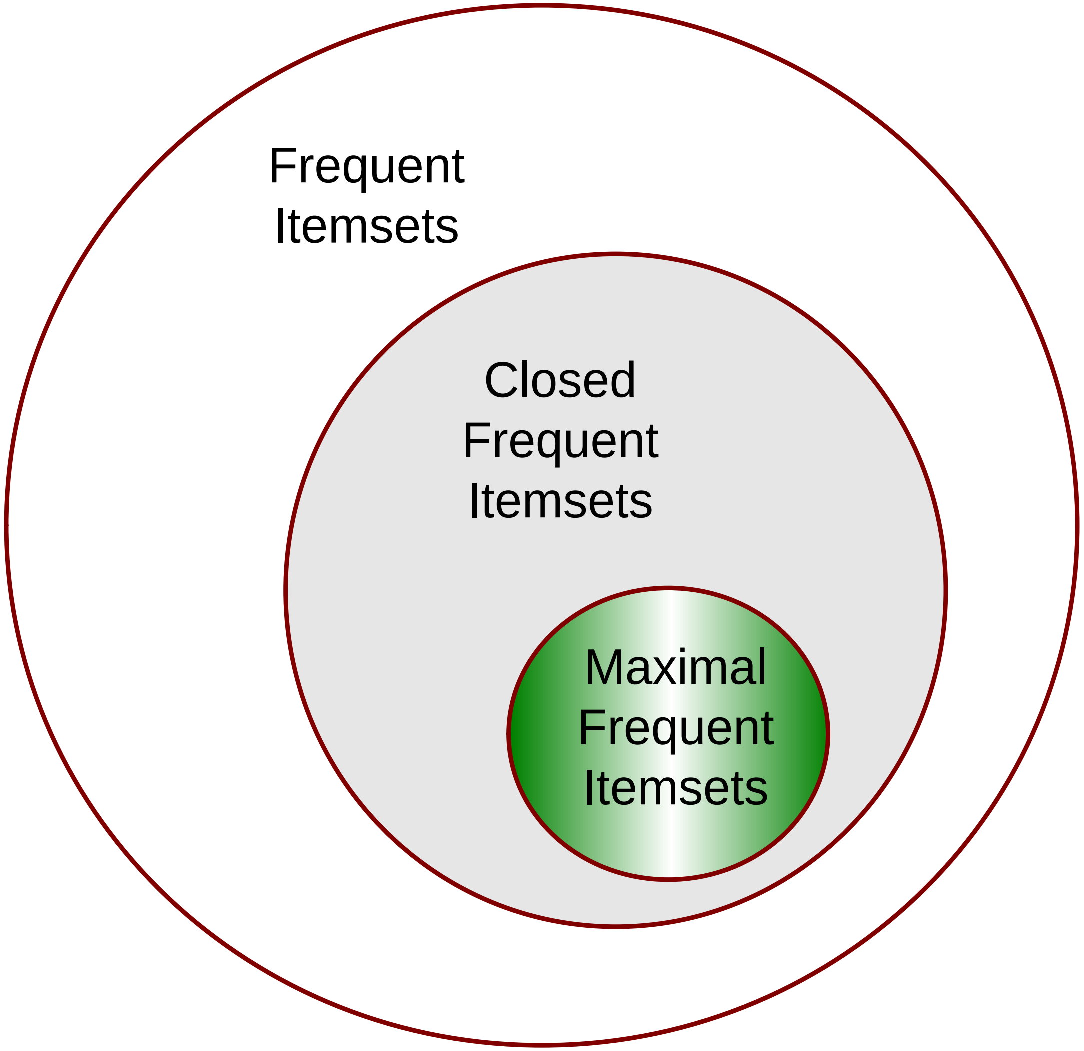
Maximal vs. Closed Itemsets



Closed = 9

Maximal = 4

Maximal vs Closed Itemsets



All maximal frequent itemsets are closed because none of the maximal frequent itemsets can have the same support count as their immediate supersets.

Deriving Frequent Itemsets From Closed Frequent Itemsets

- Consider {a, d}.
  - It is frequent because {a, b, d} is.
  - Since it isn't closed, its support count must be identical to one of its immediate supersets.
  - The key is to determine which superset among {a, b, d}, {a, c, d}, or {a, d, e} has exactly the same support count as {a, d}.

- The Apriori principle states that:
  - Any transaction that contains the superset of {a, d} must also contain {a, d}.
  - However, any transaction that contains {a, d} does not have to contain the supersets of {a, d}.
  - So, the support for {a, d} must be equal to the largest support among its supersets.
  - Since {a, c, d} has a larger support than both {a, b, d} and {a, d, e}, the support for {a, d} must be identical to the support for {a, c, d}.

Example

C = {ABC:3, ACD:4, CE:6, DE:7}

k_{max}=3

F3 = {ABC:3, ACD:4}

F2 = {AB:3, AC:4, BC:3, AD:4, CD:4, CE:6, DE:7}

F1 = {A:4, B:3, C:6, D:7, E:7}

Computing Frequent Closed Itemsets

During the Apriori Algorithm:

- After computing, say F_k and F_{k+1}, check whether there is some itemset in F_k which has a support equal to the support of one of its supersets in F_{k+1}. Purge all such itemsets from F_k.