Image source: http://socialmedialab.upenn.edu/sites/default/files/uploads/images/DataMining.jpg

CS 422-04: Data Mining

Vijay K. Gurbani, Ph.D., Illinois Institute of Technology

Lecture 6: **Association Analysis (Rules)**

Association Rule Mining

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction. (Note: Implication means co-occurrence, not causality!)

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**

Example of Association Rules

{Diaper} → {Beer}
{Milk, Bread} → {Eggs,Coke}

{Beer, Bread} → {Milk}

Antecedent → Consequent

Association Rule Mining

Binary representation of market basket data

Image source: https://goo.gl/images/mQ3zZz

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

	Beer	Bread	Milk	Diaper	Eggs	Coke
T_1	0	1	1	0	0	0
T_2	1	1	0	1	1	0
T_3	1	0	1	1	0	1
T_4	1	1	1	1	0	0
T_5	0	1	1	1	0	1

Association Rule Mining

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**

Example of Rules:

{Milk,Diaper} → {Beer} (s=0.4, c=0.67)
{Milk,Beer} → {Diaper} (s=0.4, c=1.0)
{Diaper,Beer} → {Milk} (s=0.4, c=0.67)
{Beer} → {Milk,Diaper} (s=0.4, c=0.67)
{Diaper} → {Milk,Beer} (s=0.4, c=0.5)
{Milk} → {Diaper,Beer} (s=0.4, c=0.5)

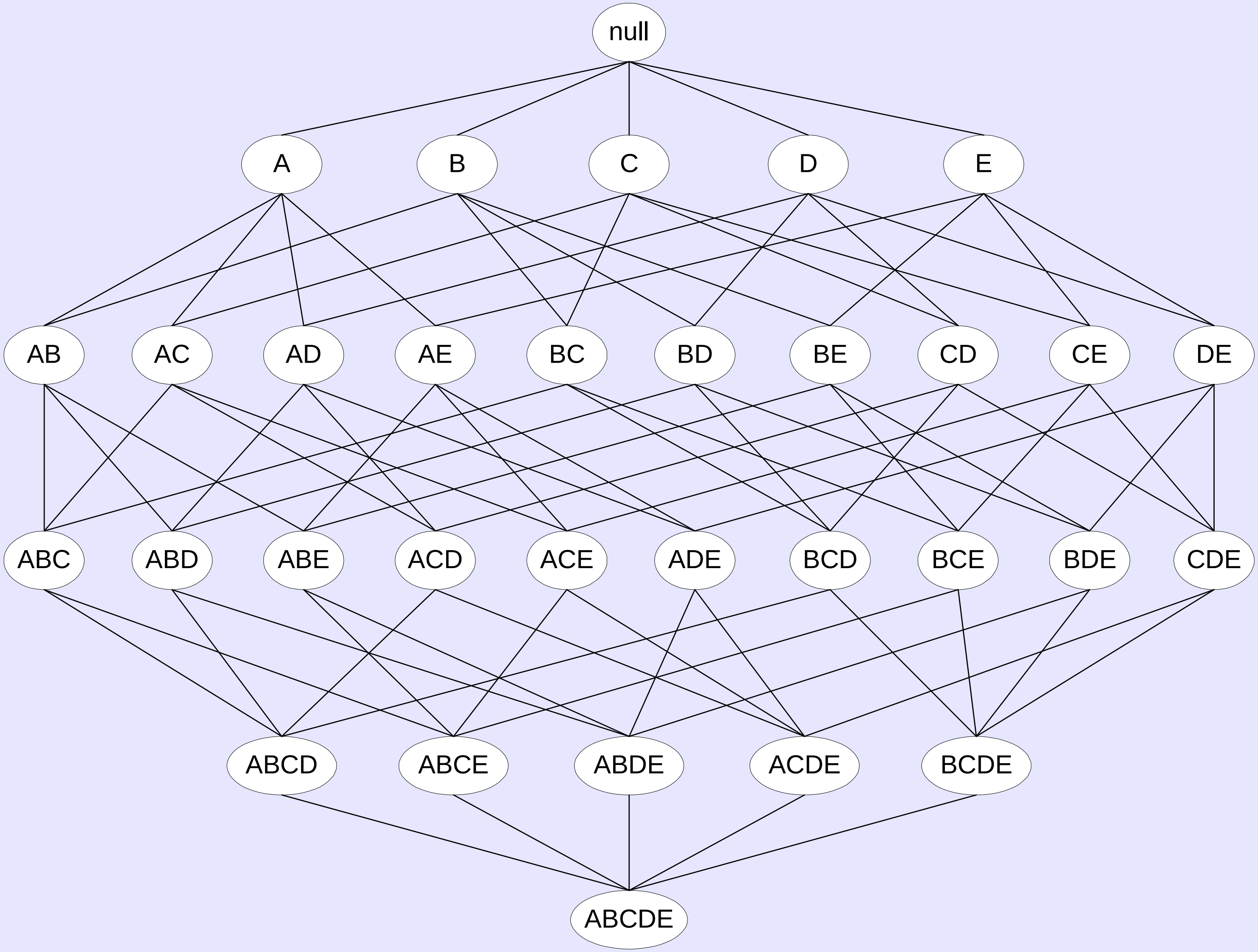
Observations:

- All the above rules are binary partitions of the same itemset:
  {Milk, Diaper, Beer}
- Rules originating from the same itemset have identical support but can have different confidence
- Thus, we may decouple the support and confidence requirements

Association Rule Mining

- Goal of Association Rule Mining: Given a set of transactions, T, find all rules having:
  - support >= *minsup*
  - confidence >= *minconf*
- How do we get there?
- Two steps:
  - Frequent itemset generation: find all items that satisfy *minsup* threshold (frequent itemsets). (Is computationally expensive!!)
  - Rule generation: extract all high-confidence rules from the frequent itemsets (strong rules).

Frequent Itemset Generation



Lattice structure to enumerate all possible itemsets.
A = Bread, B = Milk, C = Diaper, …

k items generate up to 2^k*-1* frequent itemsets.

Frequent Itemset Generation

How many item-sets?

k items generate up to 2^k*-1* frequent itemsets.

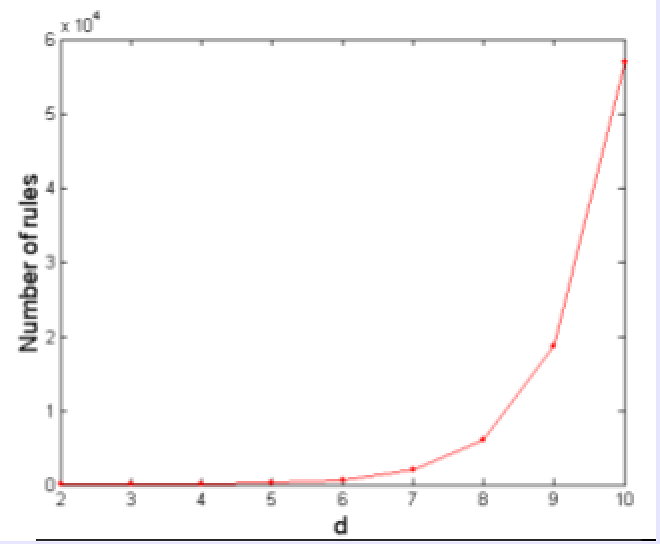
$$\text{Number of itemsets for k items} = \binom{k}{1} + \binom{k}{2} + \ldots + \binom{k}{k} = 2^k - 1$$

For a 3-itemset {a,b,c} the candidate rules will be:
ab → c, ac → b, a → bc, b → ac, …, abc → 0 and 0 → abc

How many rules?

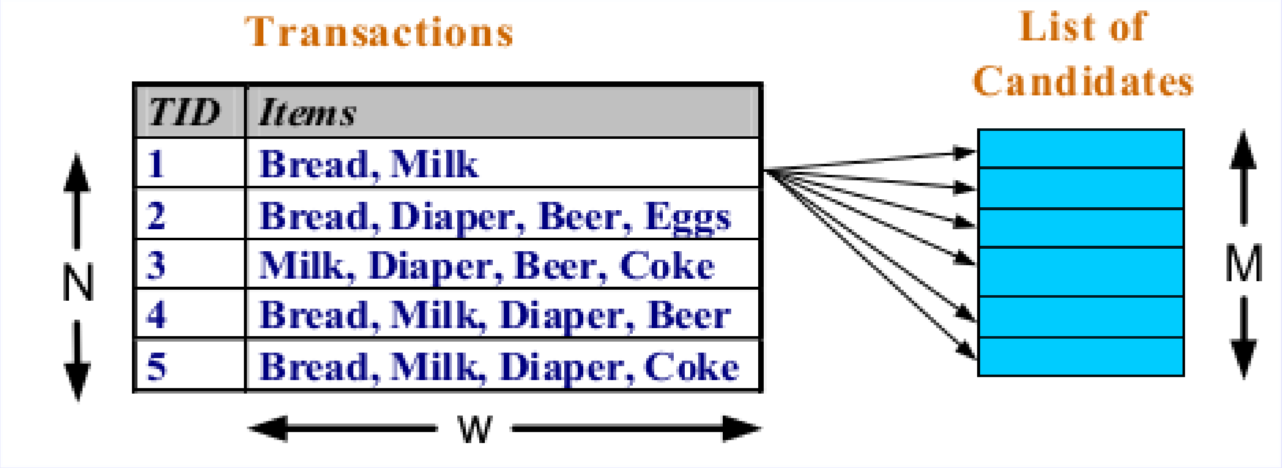
$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right] = 3^d - 2^{d+1} + 1$$

d = No. of items
For d = 3, R = 12
For d = 6, R = 602



Frequent Itemset Generation

- Brute force approach:
  - Each itemset in the lattice is a candidate frequent itemset. Store it in a database.
  - If candidate is contained in a transaction, support_count++.
  - Requires matching each transaction against every candidate.



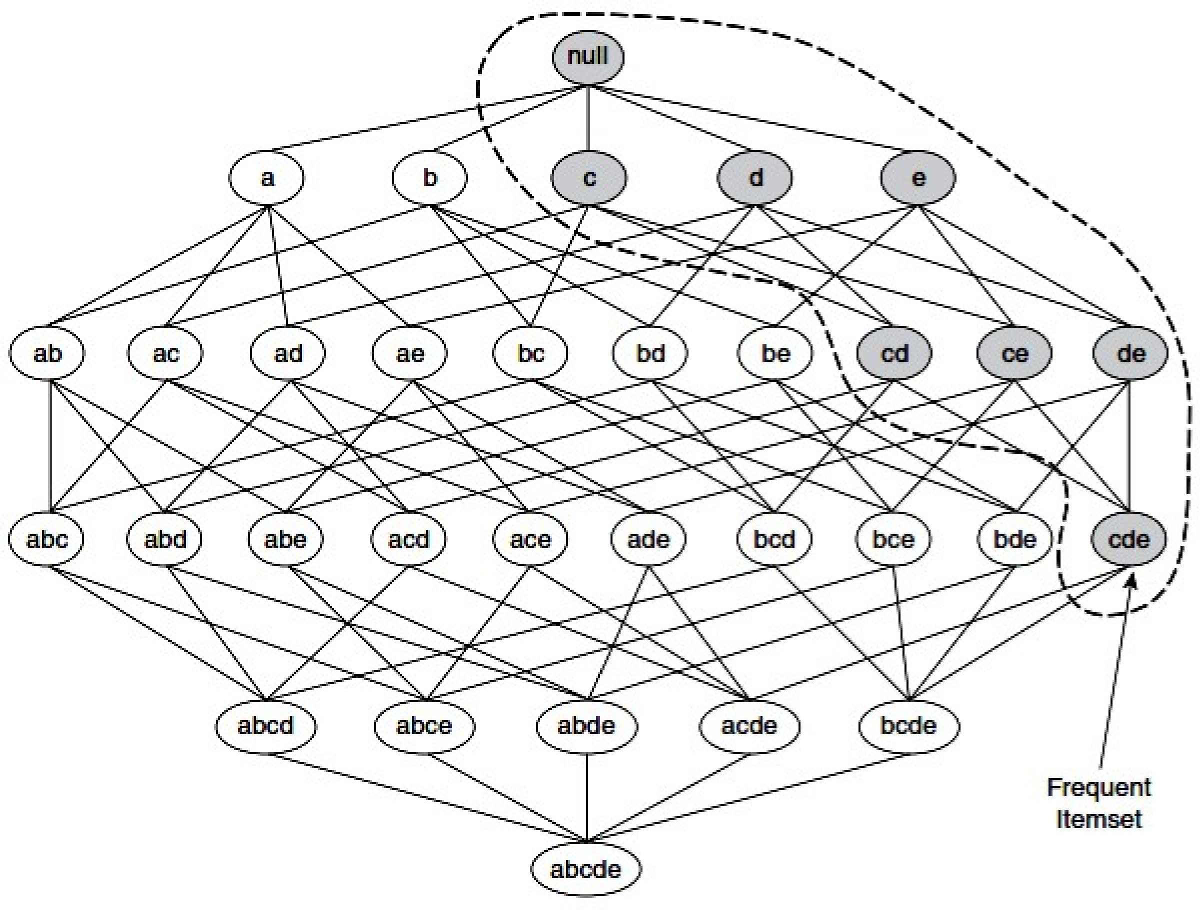
Complexity: $O(NMw)$ is exponential since $M = 2^d$.

Frequent Itemset Generation

- So, how to reduce this complexity?
  - **Reduce M,** the number of candidate itemsets (the *Apriori* principle).
  - **Reduce the number of comparisons**, using better data structures to store the candidate itemsets (Support Counting) or to compress the dataset (FP-Growth).

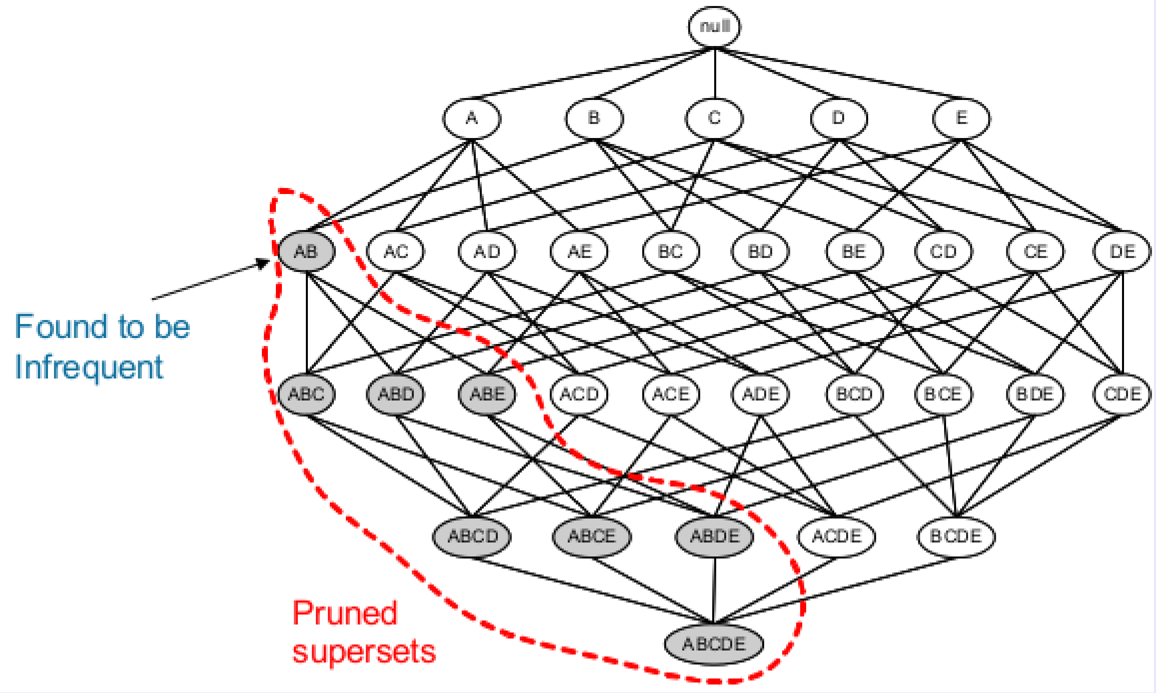
Frequent Itemset Generation: Reducing the number of candidate itemsets

- The *Apriori* principle:
  - If an itemset is frequent, then all of its subsets must be frequent as well.



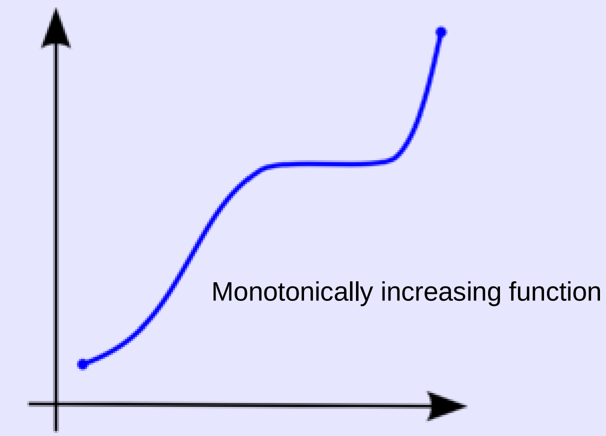
Frequent Itemset Generation: Reducing the number of candidate itemsets

- The *Apriori* principle:
  - Conversely, if an itemset is infrequent, then all of its supersets must be infrequent as well.

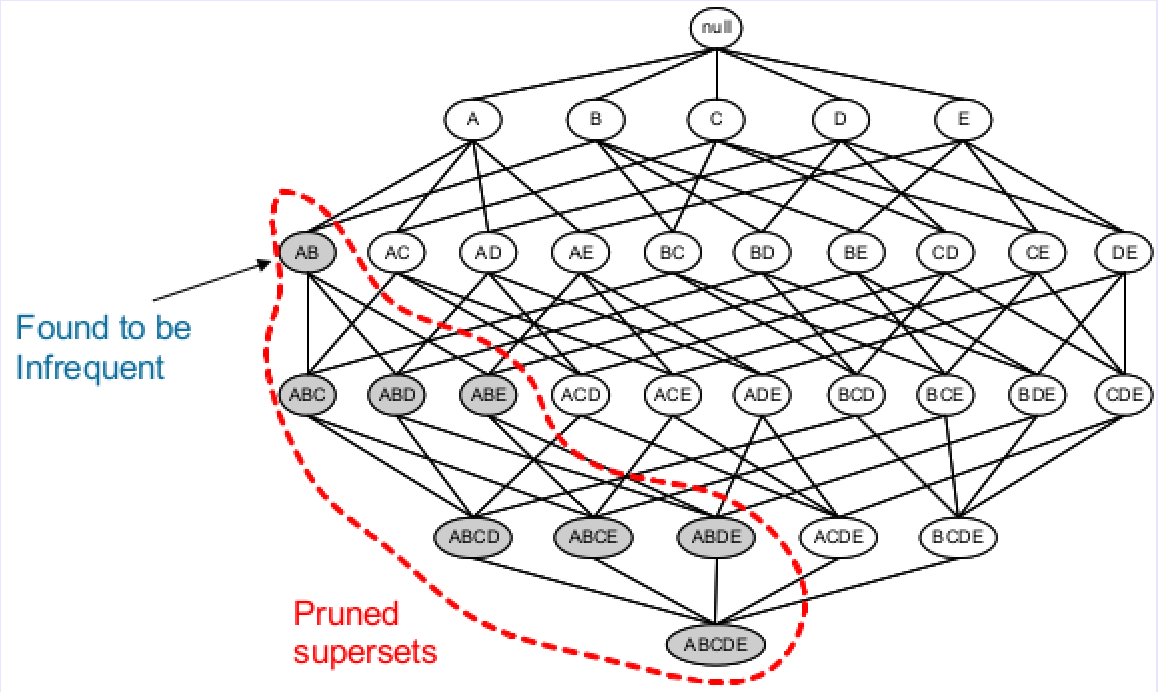


Frequent Itemset Generation: Reducing the number of candidate itemsets

- The anti-monotone property



Source: https://en.wikipedia.org/wiki/File:Monotonicity_example1.png



- Apriori holds due to the following property of support measure:

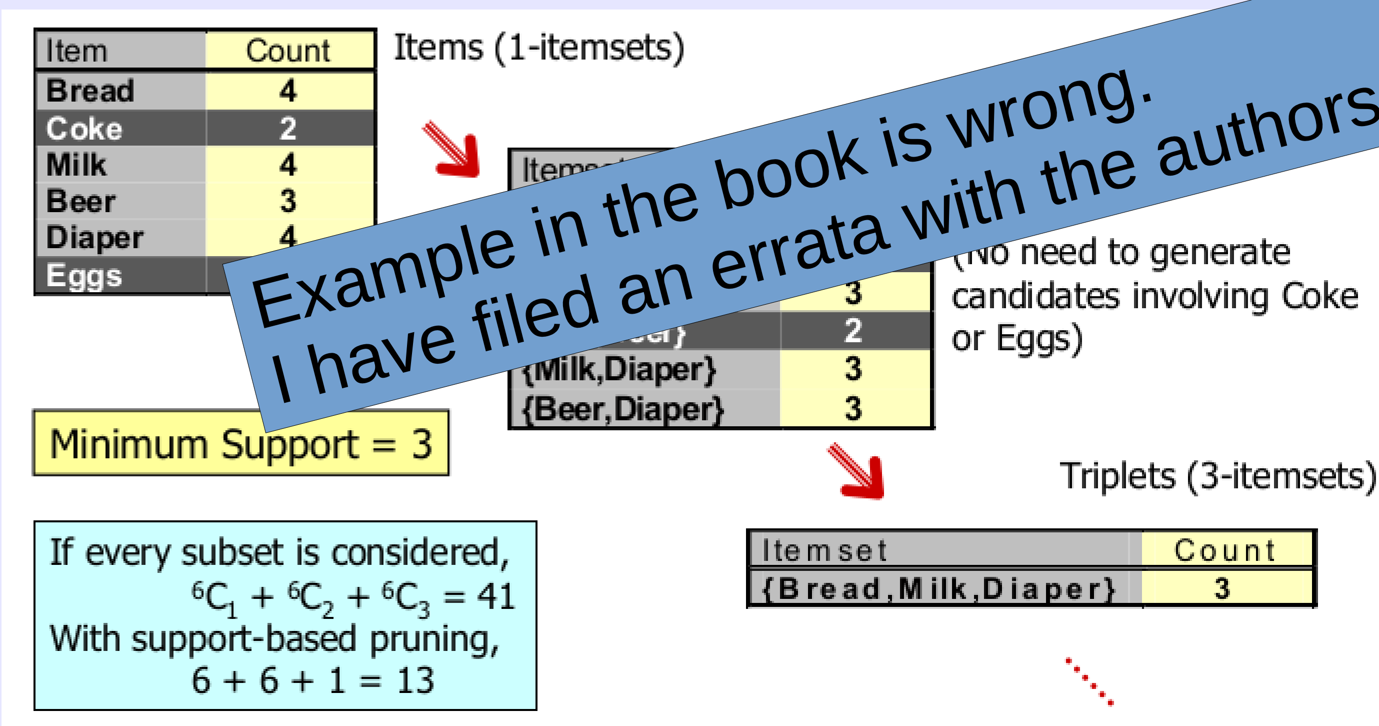
$$\forall X, Y : (X \subseteq Y) \Rightarrow s(Y) \leq s(X)$$

E.g. X=ABDE, Y=ABCDE, then s(ABCDE) <= s(ABDE), if we can prune ABCDE, we can prune ABDE.

Frequent Itemset Generation: Reducing the number of candidate itemsets

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, E
	Milk, Diaper, Beer, Co

Example in the book is wrong.
I have filed an errata with the authors.



Frequent Itemset Generation: Reducing the number of candidate itemsets

Min. Support Count = 3 (minsup = 0.50)

Tid	Beer	Bread	Cola	Diaper	Eggs	Milk
T1	0	1	0	0	0	1
T2	1	1	0	1	1	0
T3	1	0	1	1	0	1
T4	1	1	0	1	0	1
T5	0	1	0	1	0	1
T6	0	1	0	1	1	1

→

Itemset	Count
Beer	3
Bread	5
Cola	1
Diaper	5
Eggs	2
Milk	5

Candidate 1-itemsets

Frequent Itemset Generation: Reducing the number of candidate itemsets

Min. Support Count = 3 (minsup = 0.50)

Tid	Beer	Bread	Cola	Diaper	Eggs	Milk
T1	0	1	0	0	0	1
T2	1	1	0	1	1	0
T3	1	0	1	1	0	1
T4	1	1	0	1	0	1
T5	0	1	0	1	0	1
T6	0	1	0	1	1	1

Itemset	Count
Beer	3
Bread	5
~~Cola~~	~~1~~
Diaper	5
~~Eggs~~	~~2~~
Milk	5

Candidate 1-itemsets

Itemset	Count
Beer,Bread	2
Beer,Diaper	3
Beer, Milk	2
Bread,Diaper	4
Bread,Milk	4
Diaper,Milk	4

Candidate 2-itemsets

Frequent Itemset Generation: Reducing the number of candidate itemsets

Min. Support Count = 3 (minsup = 0.50)

Tid	Beer	Bread	Cola	Diaper	Eggs	Milk
T1	0	1	0	0	0	1
T2	1	1	0	1	1	0
T3	1	0	1	1	0	1
T4	1	1	0	1	0	1
T5	0	1	0	1	0	1
T6	0	1	0	1	1	1

Itemset	Count
Beer	3
Bread	5
~~Cola~~	~~1~~
Diaper	5
~~Eggs~~	~~2~~
Milk	5

Candidate 1-itemsets

Itemset	Count
~~Beer,Bread~~	~~2~~
Beer,Diaper	3
~~Beer, Milk~~	~~2~~
Bread,Diaper	4
Bread,Milk	4
Diaper,Milk	4

Candidate 2-itemsets

Itemset	Count
Bread,Diaper,Milk	3

Candidate 3-itemsets

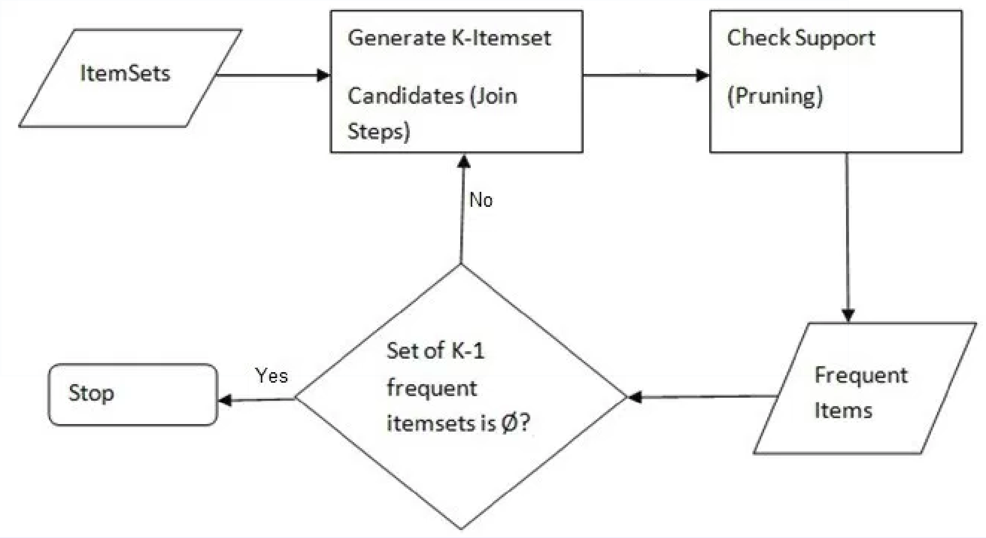
$$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41$$

$$\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13$$

Reduction of 68% in no. of candidate itemsets

Frequent Itemset Generation: Apriori algorithm

- Let k=1
- Generate frequent itemsets of length 1
- Repeat until no new frequent itemsets are identified
  - Generate length (k+1) candidate itemsets from length k frequent itemsets
  - Prune candidate itemsets containing subsets of length k that are infrequent
  - Count the support of each candidate by scanning the DB
  - Eliminate candidates that are infrequent, leaving only those that are frequent



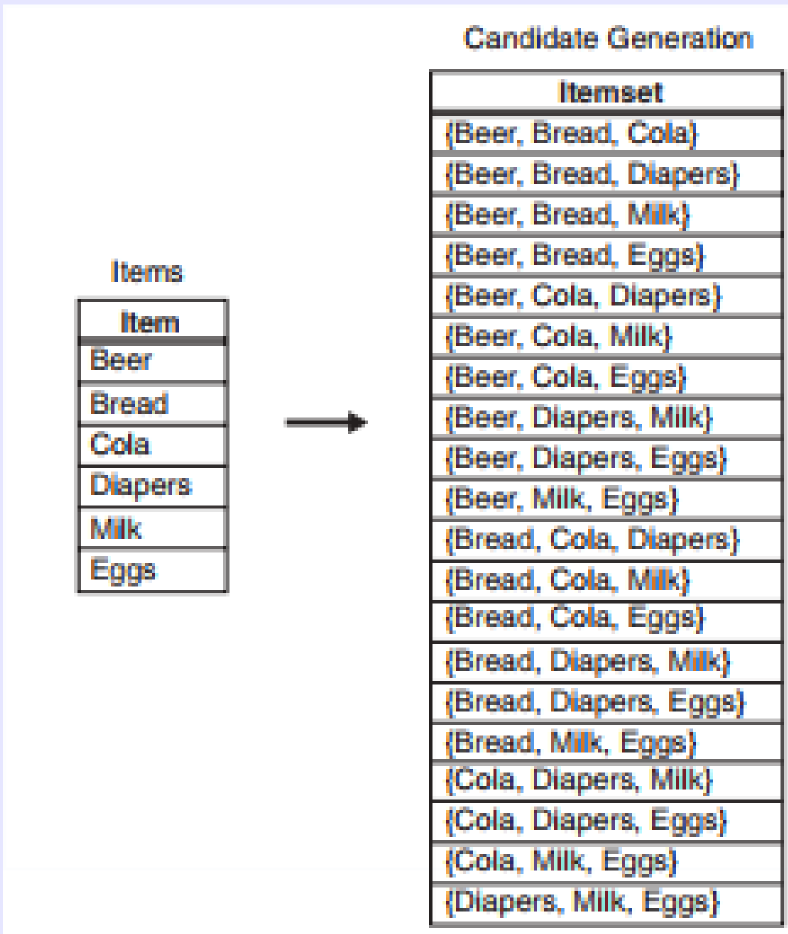
Slide courtesy: http://blog.hackerearth.com/beginners-tutorial-apriori-algorithm-data-mining-r-implementation

Frequent Itemset Generation: Generate candidate itemsets

- Many ways to generate candidate itemsets.
  - We study two: Brute-force method and F_{k-1} X F_{k-1} method.
- Requirements:
  - Do not generate too many unnecessary candidates. (Remember the anti-monotone property: supersets of infrequent itemsets are themselves infrequent.)
  - Candidate set is complete. No frequent itemset is left out.
  - Should not generate the same candidate more than once. {milk,diaper,beer} = {diaper,milk,beer} = {beer,milk,diaper} = …
    - Generation of duplicate candidates leads to wasted compute cycles.
    - How to avoid duplicate candidates? Lexicographic ordering.

Frequent Itemset Generation: Generate candidate itemsets

- Generate candidate itemsets using brute-force.

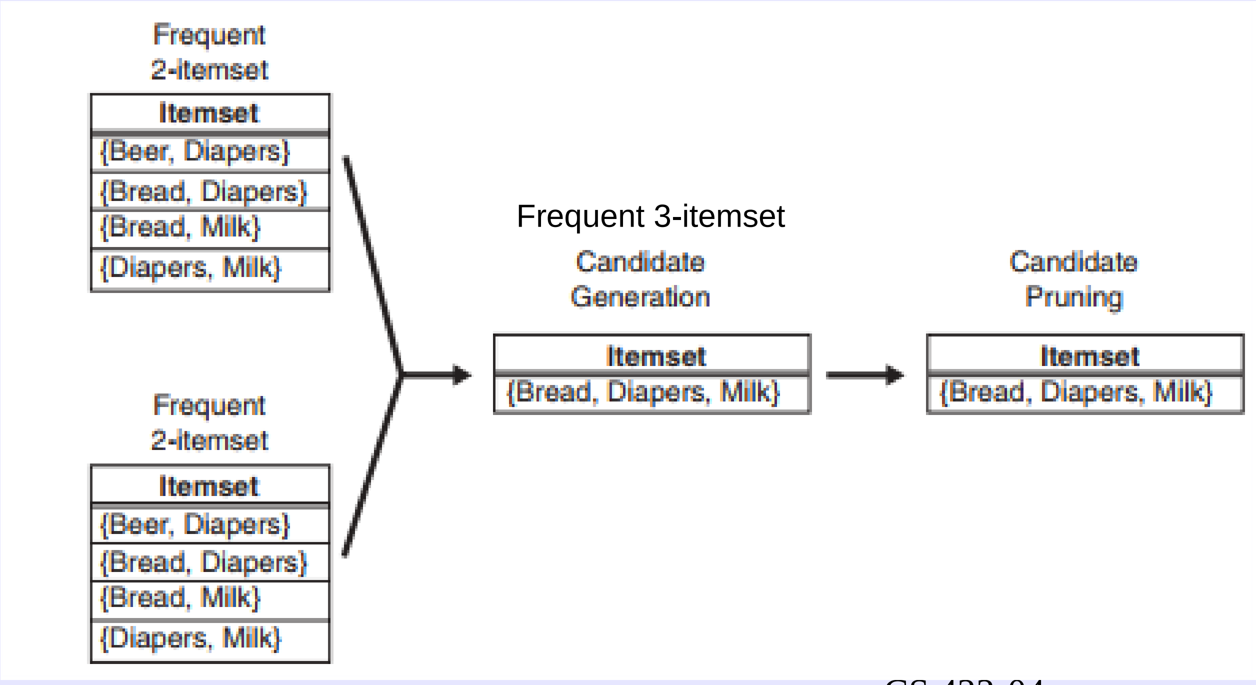


Most of these are not frequent!

$O(d*2^{d-1})$, where d is total number of items.

Frequent Itemset Generation: Generate candidate itemsets

- Generate candidate itemsets using F_{k-1} X F_{k-1} method: *merge a pair of frequent (k-1) itemsets IFF their first k-2 items are identical.* To generate k=3-itemset, first k-2 = 3-1 = 1 items must be similar.

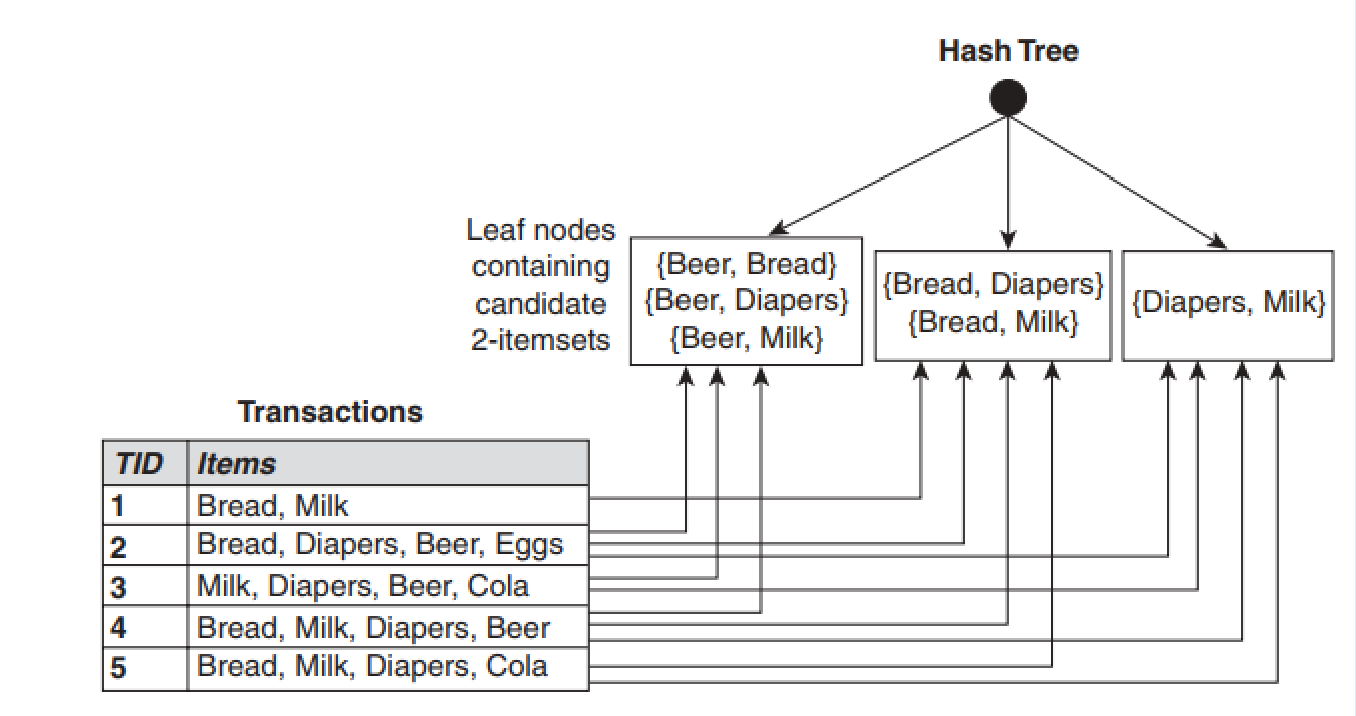


Frequent Itemset Generation: Support counting

- Support counting: determine frequency of occurrence of each candidate itemset that survives after pruning.
  - How? Compare each transaction against every candidate itemset and update support count of the candidates contained in the transaction.
  - Computationally expensive when candidate itemsets and number of transactions are large.

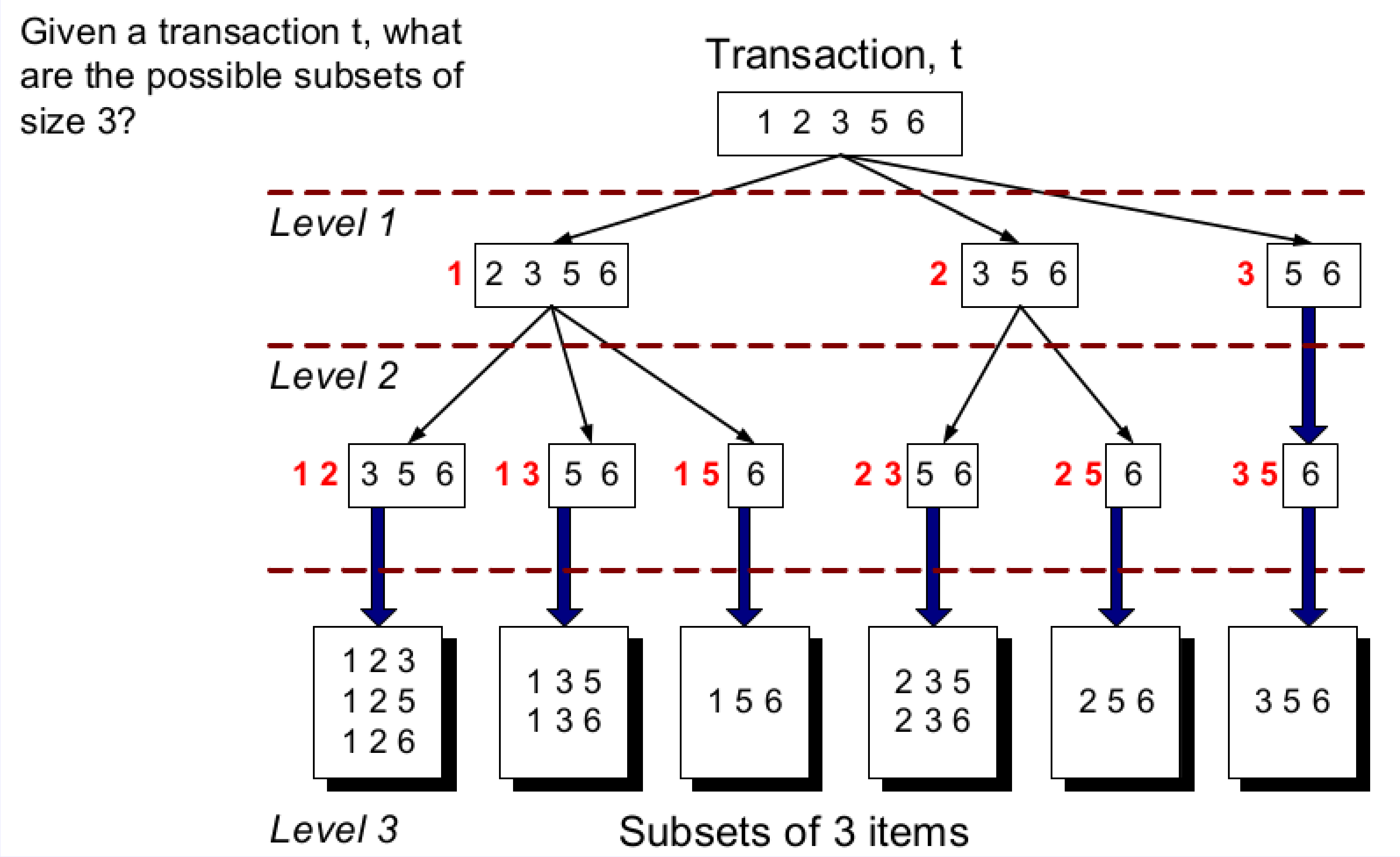
Frequent Itemset Generation: Support counting

- Instead, we want to use efficient data structures for support counting.
  - Hashes! Search time: O(1).



Frequent Itemset Generation: Support counting

- Efficient enumeration of subsets.



Frequent Itemset Generation: Support counting

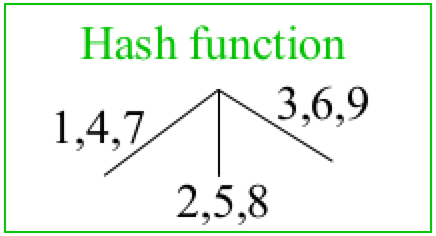
- Generate hash tree. *h(p) = p mod 3.*

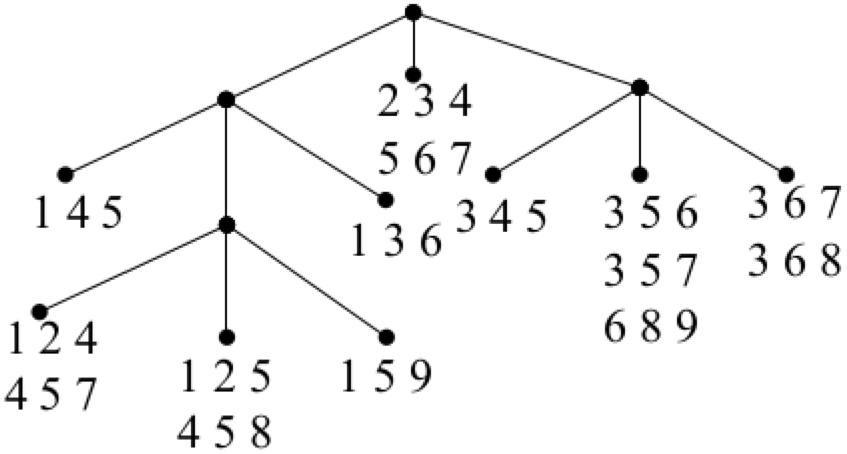
Suppose you have 15 candidate itemsets of length 3:

{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}

You need:

- **Hash function**
- **Max leaf size: max number of itemsets stored in a leaf node (if number of candidate itemsets exceeds max leaf size, split the node)**

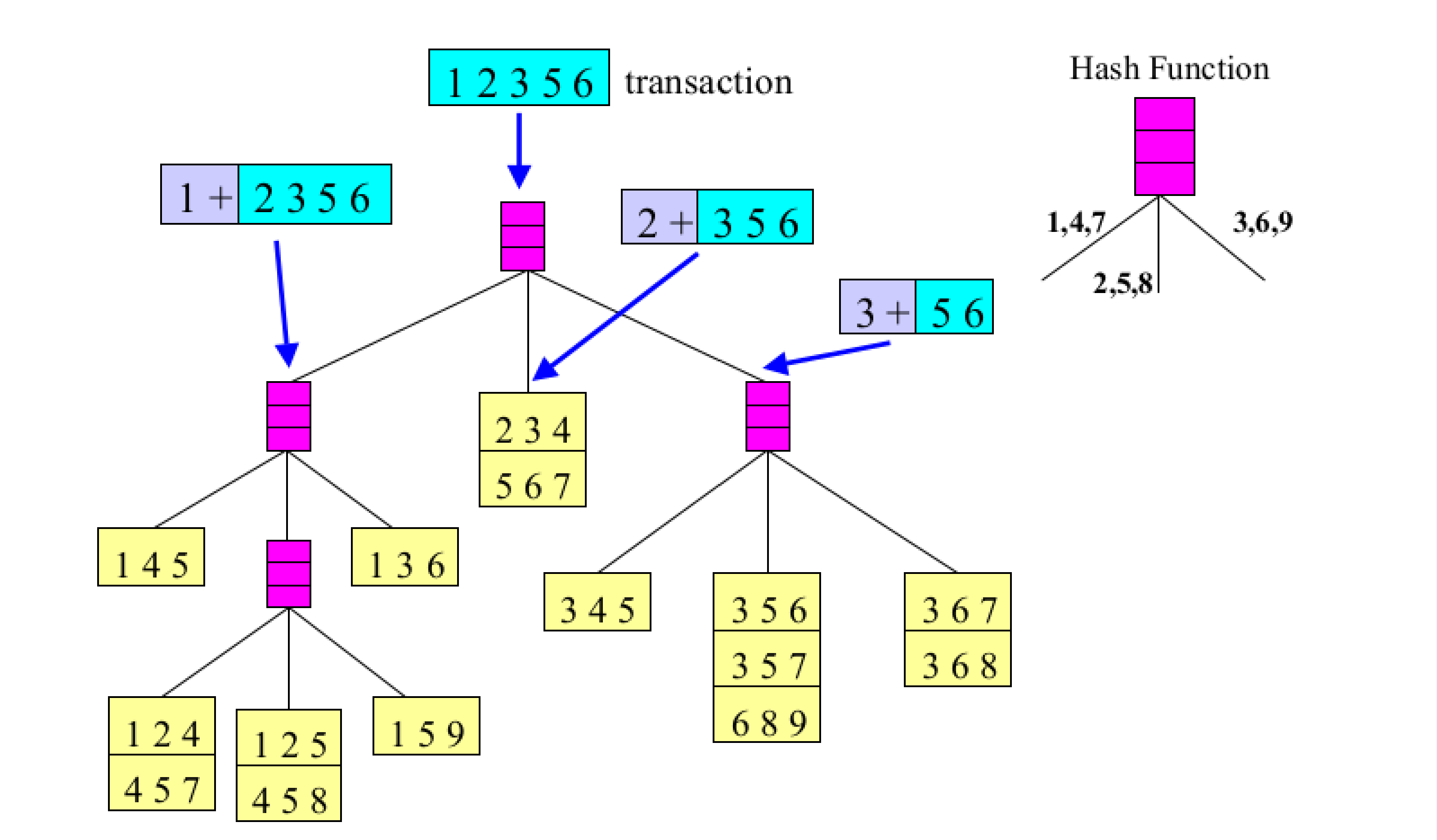




Frequent Itemset Generation: Support counting

h(p) = p mod 3

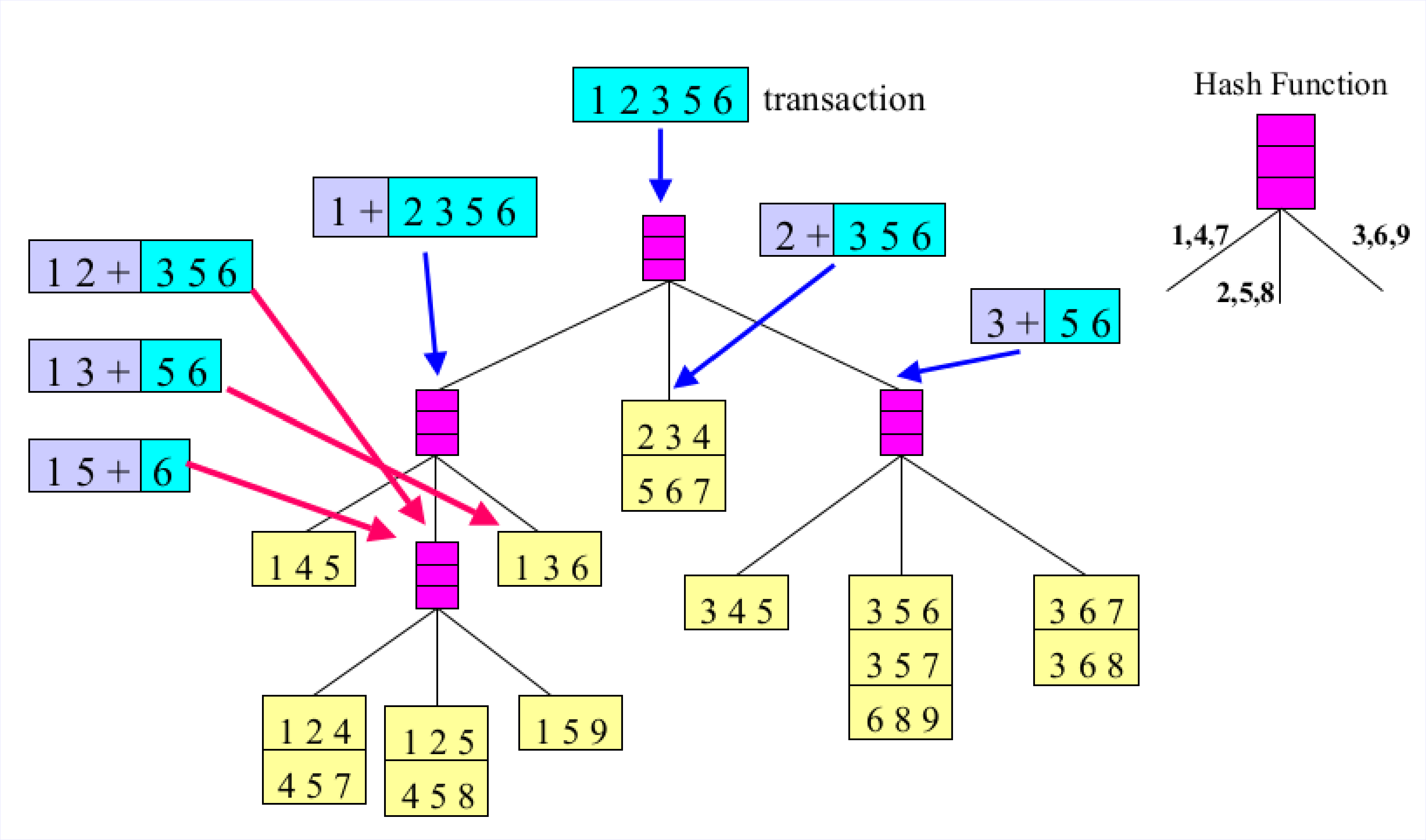
- Subset operation using a hash tree



Frequent Itemset Generation: Support counting

h(p) = p mod 3

- Subset operation using a hash tree



Find 3-itemsets:
{1 2 5}
{2 3 6}
{3 5 6}