Data Mining

Association Rules Discovery
Part one: Apriori

Market basket problem

- Given:
  - Large number of items : bread, milk, banana, cereal, …
  - Customer fill their basket (or, online shopping carts) with a subset of these items
  - Available recordings of the content of the baskets
  - Goal:
    - Derive "What items do people buy together?"
- Purpose:
  - Marketers use the information to position items, and control the way a typical customer traverse the store
  - Targeted advertisement during online shopping/browsing

Related Problems

- Similar problems
  - Information gathering from text documents
    - Baskets : documents
    - Items : words
    - Goal: documents share groups of words may indicate the resemblance of their content → information retrieval and intelligence gathering
  - Finding mirror sites
    - Baskets : documents
    - Items : sentences
    - Goal : web page containing groups of the same sentences may indicate they are mirror sites on the web

Goal of Market-basket analysis

- Association rule discovery
- Causality analysis

What is Association Rule ?

- Assume :
  - J={i_1, i_2, …, i_m} is a set of items;
  - D, be a set of database transactions where each transaction T is a set of items, such that T is a subset of J;

T_1 : i_2, i_5, i_6, i_{12}, i_9, i_{30}, i_{55}, …

T_2 : i_1, i_2, i_5, i_{38}, i_{39}, i_{100}, i_{121}, …

T_3: i_4, i_6, i_{23}, i_{29}, i_{59}, i_{44}, …

….

T_N : i_6, i_9, i_{35}, i_{26}, i_{40}, …

Association Rule

- An **association rule** is an implication of the form

  X → Y,

  where X and Y are subsets of J, and X and Y do not share common item.

  For example:

  buys canned beer → buys chips & buys coke

  buys laptop & buys external hard drive → buys power supply

X and Y are sets of items.

A transaction T is said to **contain** X (or Y) if and only if X is a subset of T.

Support and Confidence

- <u>Support of the rule</u>:

  The rule X ➔ Y holds in the transaction set D with **support** s, where s is the percentage of transactions in D that contain both X and Y. (Computed as P(X and Y))

- <u>Confidence of the rule</u>:

  The rule X ➔ Y has **confidence** c in the transaction set D if c is the percentage of transactions in D containing X that also contain Y. (Computed as P(Y|X))

Lift

- <u>Lift of the rule:</u>

The rule X ➔ Y has **lift** l in the transaction set D if l is the percentage of transactions in D containing X that also contain Y, while controlling for how popular item Y is. (Computed as $\frac{P(X,Y)}{P(X)P(Y)}$)

Example Association Rules

- Examples:
  1. Computer → financial_management_software
     [support = 2%, confidence = 60%, lift =0.3]
  2. Diaper → beer
     [support =3.5%, confidence = 45%, lift =2.6]
  3. Milk, butter → bread
     [support = 6%, confidence = 60%, lift=1.2]

Rules from a real database

circle size → support
red color → lift

grocery transactions

Practice problem

- Given transaction database, D:

T1	bread, milk, banana, cereal, apple, sugar, flour, butter
T2	cereal, pear, sugar, salt, egg, flour, milk
T3	bread, milk, potato, onion, apple
T4	potato chip, orange juice, coke, ice cream
T5	coke, potato chip, sugar, flour, milk

Assume that : milk → apple is an association rule discovered:

- What is the support, confidence, and lift for this rule?

- how about {sugar, flour} → egg ?

Causality Analysis

- Causality analysis:
  - Does the presence of X actually causes Y to be bought?
  - Test method:
    - Question: does diaper causes beer to be bought? Or
      does beer causes diaper to be bought?
    - Approach 1: lower the price of diaper, raise the price of beer
    - Approach 2: lower the price of beer, raise the price of diaper
    - Result

Terminologies

- **itemset :** a set of items
- **k-itemset** : an itemset that contains k items
- **occurrence frequency of an item**: the number of transactions that contain the itemset
- **frequent k-itemset:** a k-itemset whose occurrence frequency is greater than or equal to a pre-defined minimum support count
- **minimum support (count)**
- **minimum confidence (count)**
- **strong association rules :** association rules that satisfy both the minimum support and the minimum confidence threshold

Apriori based association rule discovery methods

- Finding frequent item sets
  - Frequent item set : set of items appearing in at least fraction **s** of the baskets
  - Why do we need to find frequent item sets?
  - What do we mean by frequent?
  - Frequent item sets can be found efficiently using the **Apriori** property

- What is the Apriori property?

  Aprori property : If a set of items **S** is frequent, then every subset of **S** is also frequent

Apriori property

- How does Apriori property help in finding the frequent item sets efficiently?
  - Proceeds level wise, start from frequent single items, then find the frequent pairs, the frequent triples, …

The Basic Process

- The algorithm proceeds level-wise:
  - Given minimum support count s, in the first pass find the 1-itemsets (sets with single items) that appear in at least s number of baskets. (L1)
  - Pairs of items in L1 become the candidate pairs C2 for the second pass. The pairs in C2 whose count reaches s are the frequent pairs, L2.

The basic process (cont.)

- The candidate triples, C3 are those sets such that all of the 2-itemsets are frequent. E.g., for {A, B,C} to be candidate 3-itemset, {A, B}, {B,C}, and {A,C} should all be frequent 2-itemset. Count the occurrences of triples in C3, those with a count of at least s are the frequent triples, L3.
- Proceed as this until the ith frequent item set becomes empty.

  L_i : frequent item set of size i

  C_i : candidate item set of size i

The Apriori Algorithm

Identify frequent 1-itemset by scanning the database

For each k value (starting with k=2, ends when L_{k-1} becomes empty):

1. Applying candidate generation to generate all possible k-itemsets based on the frequent 1-itemsets

- the join step :
  - L_k is found by joining L_{k-1} with itself
  - Apriori assumes that all items within a transaction or itemset are sorted in lexicographic order.
  - Two items may be joined if they share the first k-2 items (Why?)

The Apriori Algorithm

- <u>the pruning step</u>:

  C_k is a superset of $L_{k,}$ that is, its members may or may not be frequent, but all of the frequent k-itemsets are included in C_k.

2. Eliminate candidate k-itemsets that have support smaller than the minimum support count

Return the union of all L_k

Algorithm Apriori

```
L1 = find_frequent_1-itemsets (D);
for (k=2; L k-1  != ϕ; k++)  {
   Ck = apriori_gen (Lk-1);
   for  each transaction  t  in D     {
       Ct = subset(Ck, t);
       for each candidate c in Ct
           c.count ++;
    }
    Lk = {c in Ck | c.count >= minimum_support _count}
     L=L  union  Lk;
}
return  L;
```

Algorithm Apriori (cont.)

```
procedure apriori_gen(L_{k-1})        // frequent (k-1)-itemset
    for each itemset l_1 in L_{k-1}
        for each itemset l_2 in L_{k-1}    {
            if ((l_1[1] = l_2[1]) ^ (l_1[2]=l_2[2])^ …^(l_1[k-2]=l_2[k-2])^(l_1[k-1]<l_2[k-1]))
            {
                c = l_1 join l_2;   // join step
                if  has_infrequent_subset(c, L_{k-1})   then
                    delete c;    // pruning step
                else
                    add c to C_k;
            }
        }
    return C_k;
```

Algorithm Apriori (cont.)

```
procedure has_infrequent_subset (c_k, ,   L_k-1 )  {
   for each (k-1)-subset   s  of  c_k  {
        if  s not in L_k-1  then
             return  TRUE;
   }

   return FALSE;
}
```

One Example

Set minimum support count to be 2

Database

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

$\overline{C}_1$

TID	Set-of-Itemsets
100	{ {1}, {3}, {4} }
200	{ {2}, {3}, {5} }
300	{ {1}, {2}, {3}, {5} }
400	{ {2}, {5} }

L_1

Itemset	Support
{1}	2
{2}	3
{3}	3
{5}	3

C_2

Itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

$\overline{C}_2$

TID	Set-of-Itemsets
100	{ {1 3} }
200	{ {2 3}, {2 5}, {3 5} }
300	{ {1 2}, {1 3}, {1 5}, {2 3}, {2 5}, {3 5} }
400	{ {2 5} }

L_2

Itemset	Support
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

Itemset
{2 3 5}

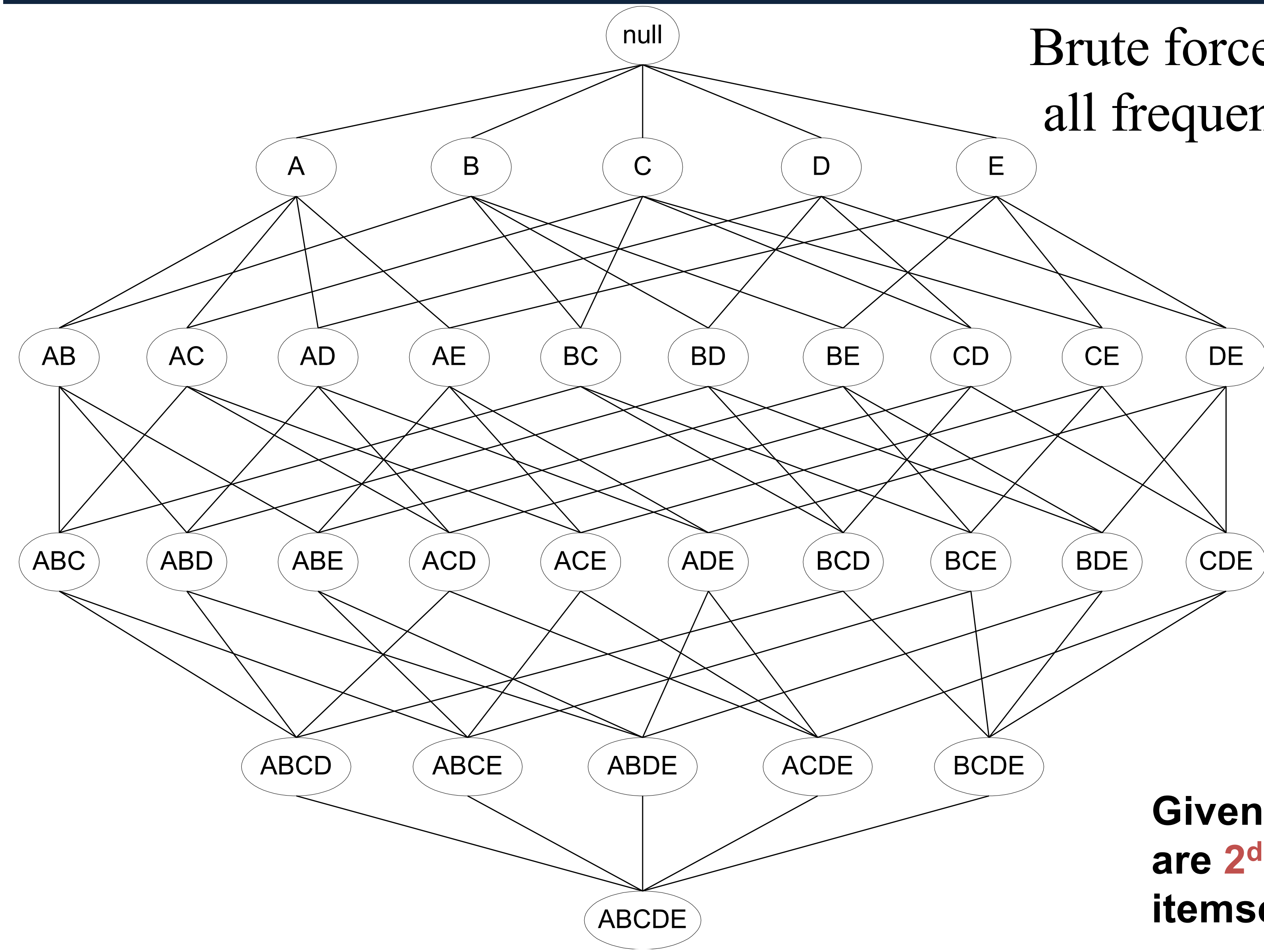
$\overline{C}_3$

TID	Set-of-Itemsets
200	{ {2 3 5} }
300	{ {2 3 5} }

L_3

Itemset	Support
{2 3 5}	2

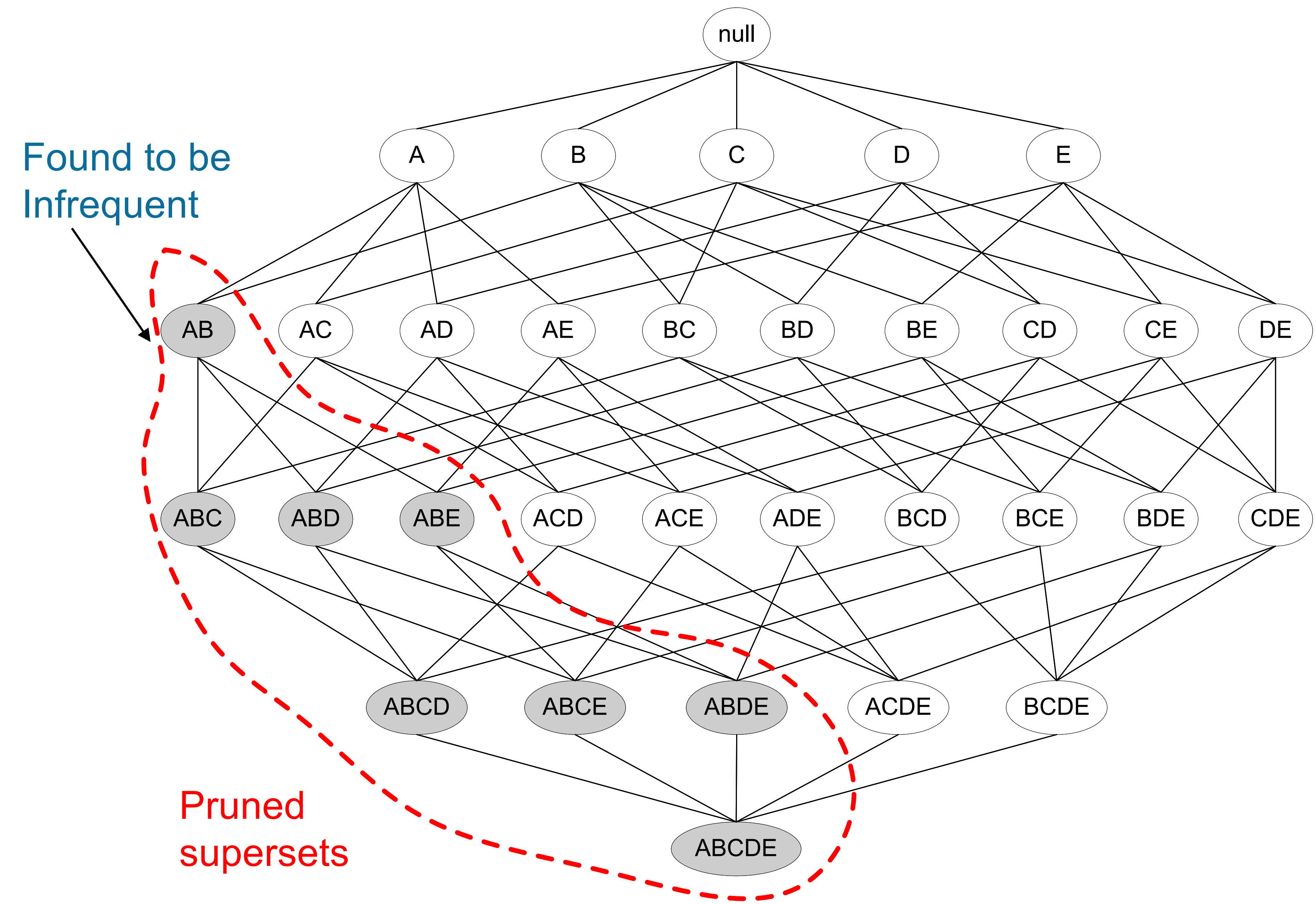
Recap



Brute force way of finding all frequent itemsets ??

Given d items, there are 2^d possible itemsets

Effects of the Apriori property



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

Set minimum Support count to be 3

Items (1-itemsets)

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

Pairs (2-itemsets)

(No need to generate candidates involving Coke or Eggs)

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

Min sup = 3/5

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	3

If every subset is considered,
$^6C_1 + {}^6C_2 + {}^6C_3 = 6 + 12 + 20 = 38$
With support-based pruning,
6 + 6 + 1 = 13

Generate Candidates

- Assume the items in L_k are listed in an order (e.g., alphabetical)
- **Step 1:** ***self-joining*** L_k *(IN SQL)*

  insert into C_{k+1}

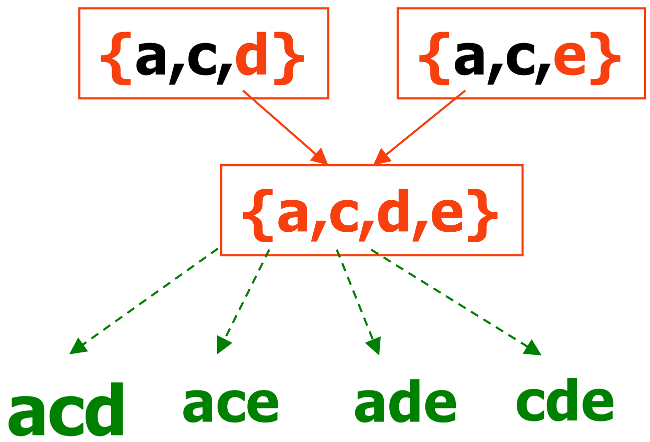
  select $p.item_1, p.item_2, \ldots, p.item_k, q.item_k$

  from L_k p, L_k q

  where $p.item_1=q.item_1, \ldots, p.item_{k-1}=q.item_{k-1}, p.item_k < q.item_k$

Example of Candidates Generation

- L_3={*abc, abd, acd, ace, bcd*}
- ***Self-joining***: $L_3 * L_3$
  - ***abcd*** from ***abc*** and ***abd***
  - ***acde*** from ***acd*** and ***ace***



Generate Candidates

- Assume the items in L_k are listed in an order (e.g., alphabetical)
- **Step 1:** ***self-joining*** L_k

  insert into C_{k+1}

  select $p.item_1, p.item_2, \ldots, p.item_k, q.item_k$

  from $L_k\ p, L_k\ q$

  where $p.item_1=q.item_1, \ldots, p.item_{k-1}=q.item_{k-1}, p.item_k < q.item_k$
- **Step 2:** ***pruning***

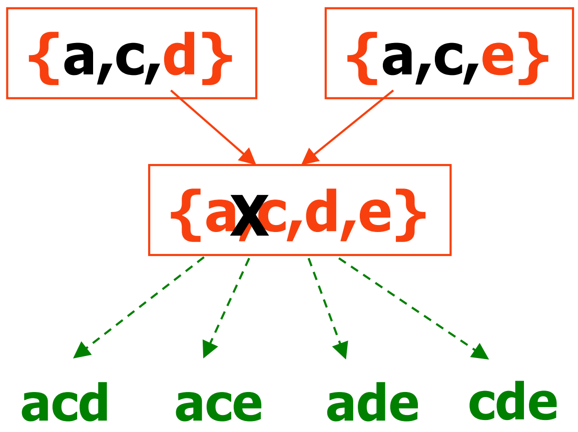
  forall ***itemsets c in*** C_{k+1} do

  forall ***k-subsets s of c*** do

  if *(s is **not** in* L_k*)* **then delete** ***c*** **from** C_{k+1}

Example of Candidates Generation

- $L_3=\{abc, abd, acd, ace, bcd\}$
- ***Self-joining***: L_3*L_3
  - ***abcd*** from ***abc*** and ***abd***
  - ***acde*** from ***acd*** and ***ace***
- ***Pruning:***
  - ***acde*** is removed because ***ade*** is not in L_3
- $C_4=\{abcd\}$



Practice Question

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

Given the transaction data, find all frequent item sets having minimum support count = 2

Step Two: Generate strong association rules from the frequent itemsets

- **Compute**:

$$\text{confidence}(X \rightarrow Y) = p(Y|X)$$

$$= \frac{\text{support\_count}(X \wedge Y)}{\text{support\_count}(X)}$$

- *support_count(X ^ Y) : number of transactions containing both itemsets X and Y*
- *support_count(X) is the number of transactions containing the itemset X.*

Generate strong association rules

- **The basic idea**: given a frequent itemset l, generate all strong associate rules based on l
- **Direct approach:**
  - for each frequent itemset l, generate all nonempty subsets of l,
  - for every nonempty subset s of l, output the rule :

    s → (l-s)

    if support_count(l) / support_count(s)

    >= minimum confidence

Properties of the confidence measure

Given :

a is a subset of l, a' is a subset of a

support (l) <= support (a) <= support(a')

Then:

confidence (a → (l - a)) >= confidence (a' → (l - a'))

Why?

Properties of the confidence measure

Similarly,

confidence of ((l - a) → a) <= confidence of ((l-a') → a')

Therefore,

If (l-a') →a' is not a strong rule, then none of the rules of the form (1-a) →a can be strong, where a is a superset of a'

Rule Generation

- **The apriori approach for rule generation**
  - Basis: If (l-a') →a' is not a strong rule, then none of the rules of the form (1-a) →a can be strong, (a' is a subset of a)
  - Approach :
    - Start by generating rules that have a single consequent
    - Increase size of consequent to 2 using the "ap_gen" function, based on only the successful single consequents
    - Continue to increase the number of consequents, (equivalently, decreasing the size of the antecedent)…, one item at a time, until the antecedent becomes empty

Example

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

Given the transaction data, find all association rules having minimum support count = 2, and minimum confidence of 70%.

Example

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

Given the transaction data, find all association rules having minimum support count = 2, and minimum confidence of 70%.

We already derived frequent itemsets for this TD as:
{{I1}, {I2}, {I3}, {I4}, {I5},
{I1, I2}, {I1, I3}, {I1, I5},
{I2, I3}, {I2, I4}, {I2, I5}
{I1, I2, I3}, {I1, I2, I5}}

Suppose l_k = {I1, I2, I5}

Second Step: Rule Generation

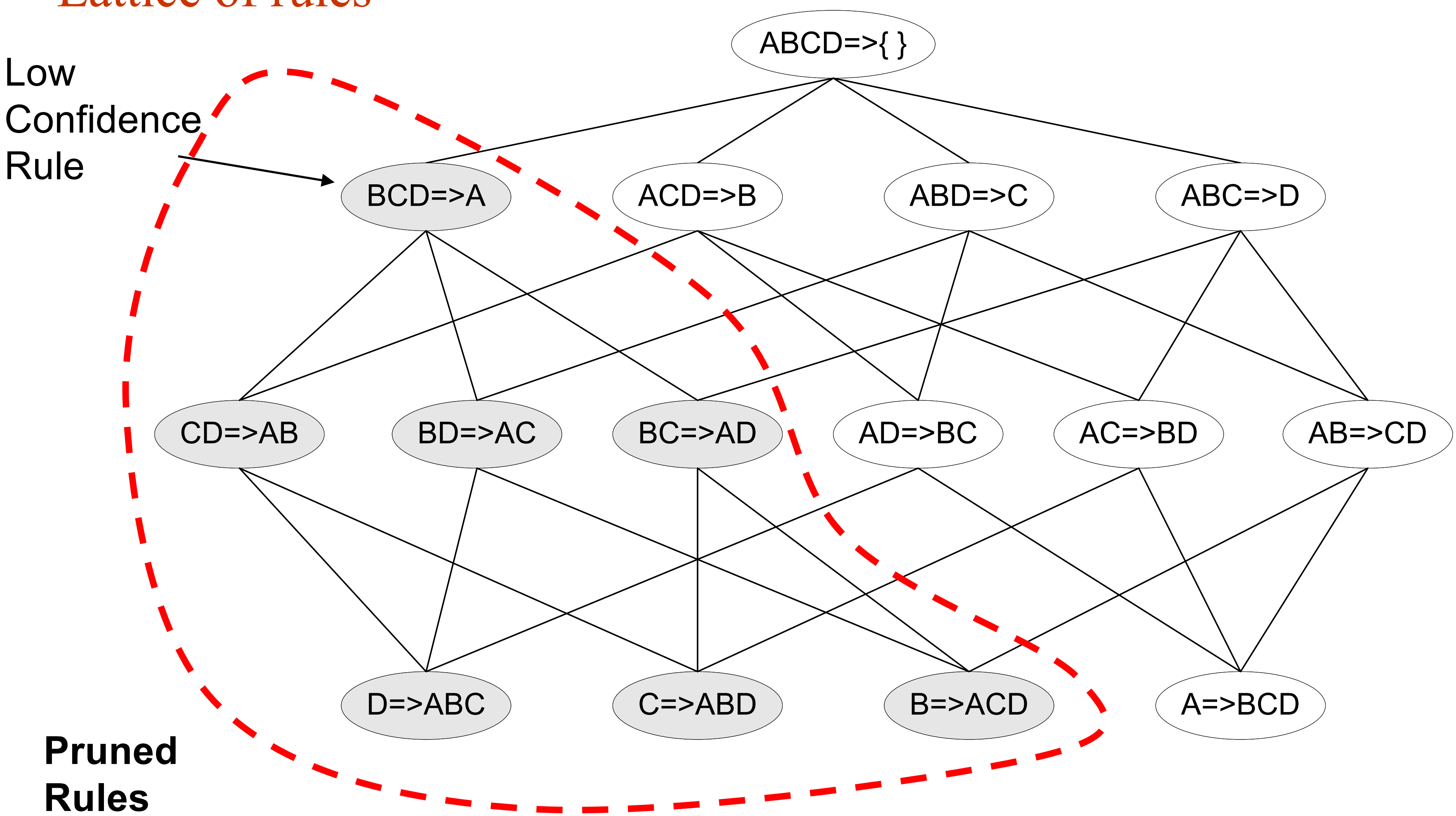
- Given a frequent itemset **X**, find all non-empty subsets $y \subset X$ such that $y \rightarrow X - y$ satisfies the minimum confidence requirement
  - If **{A,B,C,D}** is a frequent itemset, candidate rules:

    $ABC \rightarrow D$, $ABD \rightarrow C$, $ACD \rightarrow B$, $BCD \rightarrow A$,
    $A \rightarrow BCD$, $B \rightarrow ACD$, $C \rightarrow ABD$, $D \rightarrow ABC$
    $AB \rightarrow CD$, $AC \rightarrow BD$, $AD \rightarrow BC$, $BC \rightarrow AD$,
    $BD \rightarrow AC$, $CD \rightarrow AB$,

- If $|X| = k$, then there are $2^k - 2$ candidate association rules (ignoring $L \rightarrow \varnothing$ and $\varnothing \rightarrow L$)

Efficient Rule Generation

Lattice of rules



Efficient Rule Generation

- Candidate rule is generated by merging two rules that share the same prefix in the rule consequent

- **join(CD→AB,BD→AC)** would produce the candidate rule **D →ABC**

- **Prune** rule **D→ABC** if there exists a subset (e.g., **AD→BC**) that does not have high confidence

Rule generation algorithm

H_1 = {consequences of rules from l_k with one item in the consequent };

Given l_k:

for each item i in l_k,

(1) Treat it as consequent,

(2) form rule $l_k - i \rightarrow i$

(3) compute confidence = support(l_k)/support(l_k-i)

if confidence > min_conf

output rule $l_k - i \rightarrow i$

add i to H1

Rule generation algorithm (Cont.)

ap-genrules (l_k: frequent k-itemset, H_m: set of m-item consequents)

if (k > m+1) then begin

H_{m+1} = **apriori-gen**(H_m);

forall h_{m+1} belongs to H_{m+1} do

Conf = support(l_k)/support(l_k - h_{m+1});

if (Conf >= minimum_confidence) then

Output the rule (l_k - h_{m+1}) → h_{m+1} with

confidence = Conf, support = support(l_k);

else // pruning

Delete h_{m+1} from H_{m+1}

Call ap-genrules(l_k, H_{m+1});

Discussion

- Why use apriori-gen to create the consequents?
  - Join: form the consequent part of the rule, with successively larger sizes
  - Prune: eliminate no-hope candidates

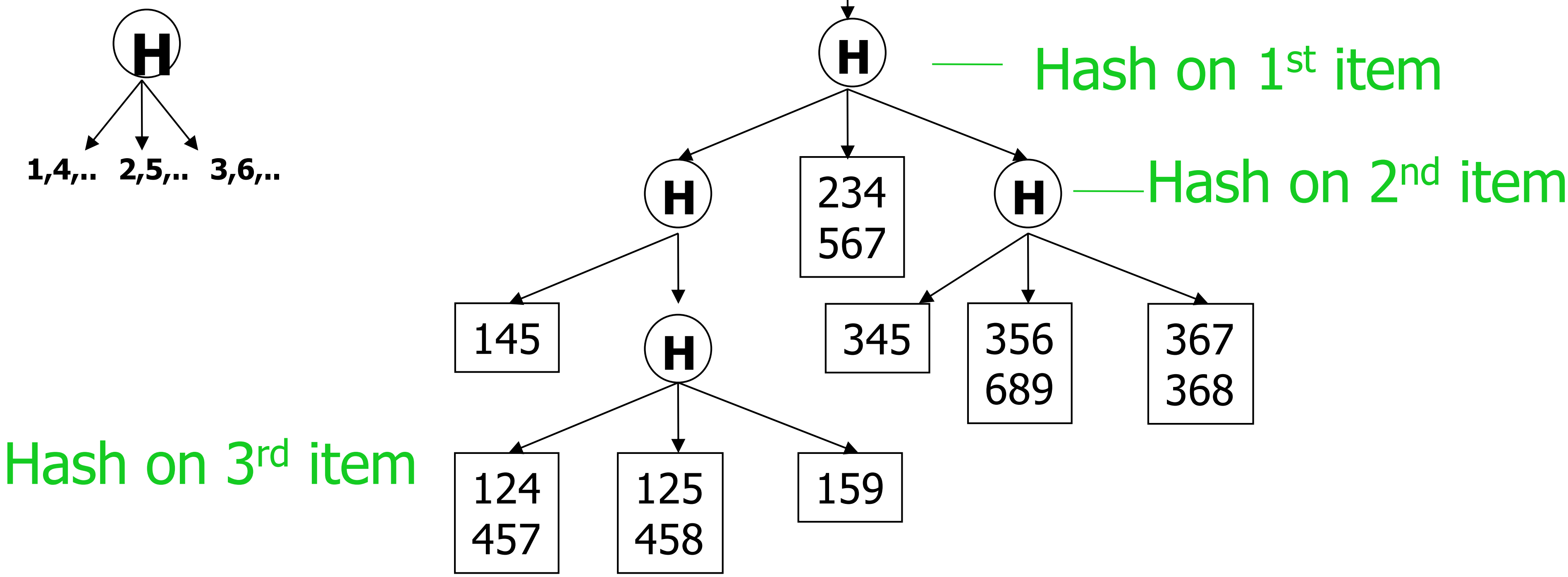
- Why is it necessary to delete h_{m+1} from H_{m+1} ?

Implementation of Apriori

- Hash tree is used for two steps of Apriori
  - Test whether "subset s of candidate itemset is in L_{k-1}" → pruning step
  - Test whether "a candidate itemset C_k is in a transaction t" → for support count
- *hash-tree*
  - *Leaf* node of hash-tree contains a list of itemsets and counts
  - *Interior* node contains a hash table
  - *Subset function*: finds all the candidates contained in a transaction

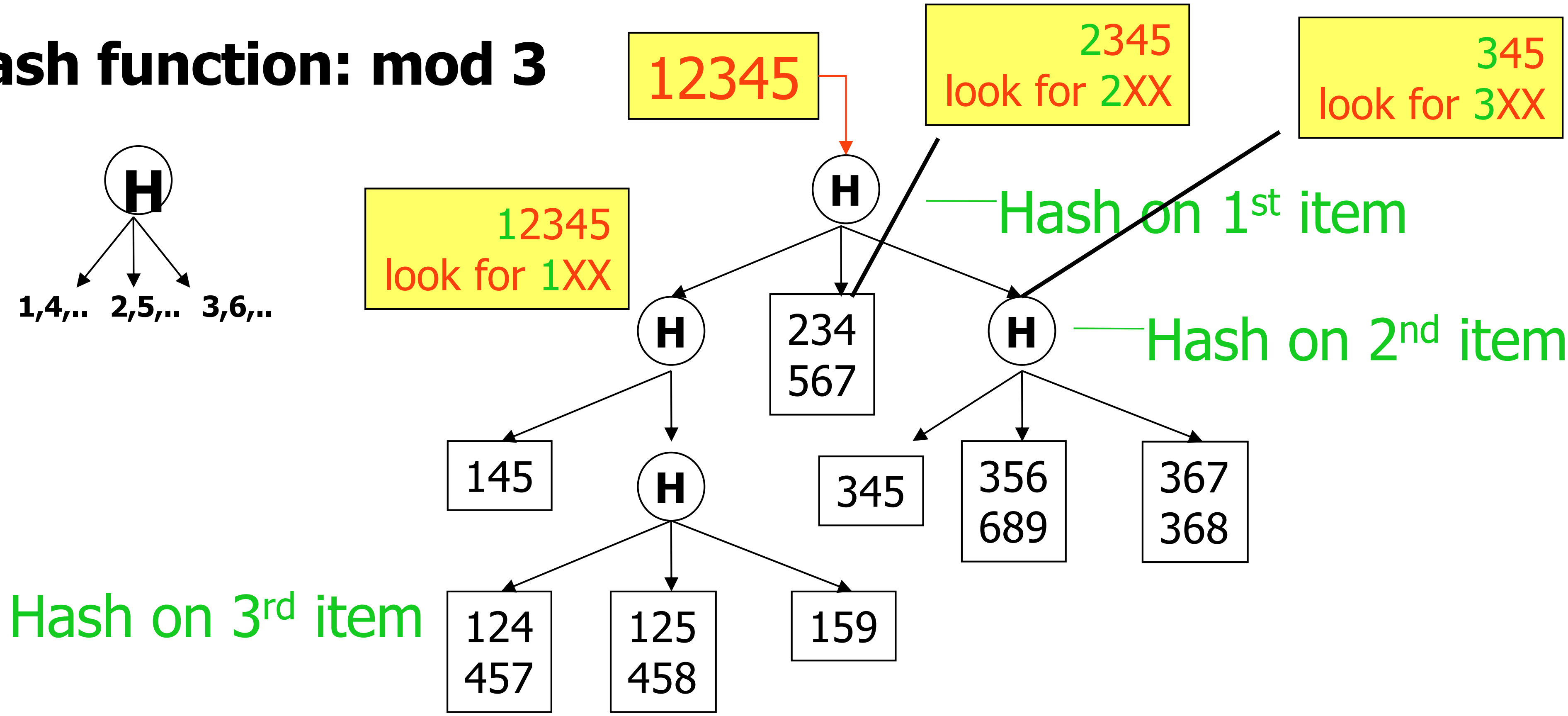
Example of the hash-tree for C_3

Hash function: mod 3



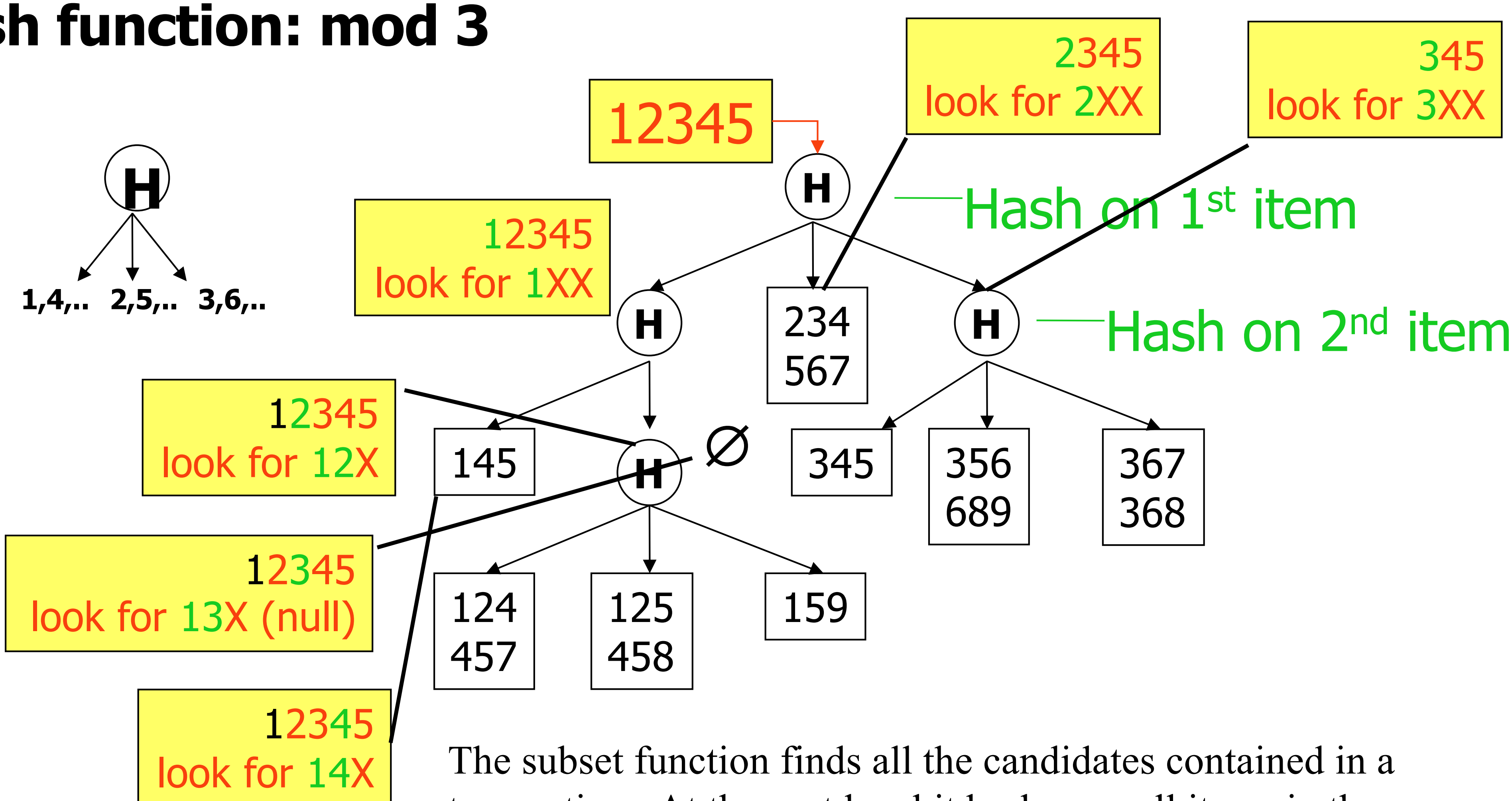
Example of the hash-tree for C_3

Hash function: mod 3



Example of the hash-tree for C_3

Hash function: mod 3



The subset function finds all the candidates contained in a transaction: At the root level it hashes on all items in the transaction; At level i it hashes on all items in the transaction that come after the i-th item

Subset(candidate itemset, L_{k-1})

- Goal : check "is there any (k-1) subset of c that is not in L_{k-1} ?
- Approach:
  - All items in L_{k-1} are stored in a hash tree
  - For each non-empty (k-1) subsets of a k-itemset, checking whether a (k-1) subset is in L_{k-1} takes O(1)

Subset (c_k, t)

- Assumption:
  - Items in transactions are ordered
  - Items in candidate set are ordered
  - Candidate c_k are put in a hash tree
- Approach:
  - At root level, hash on every item in the transaction,
  - At level i,

    if it is an interior node, hash on every item following the i^{th} item,

    if it is a leaf node, check if the candidate c is in the list

        if yes, update the count for that candidate

Efficient Support Count

- Goal : Test whether “a candidate itemset C_k is in a transaction t”?
  - All the k subsets of a transaction t are stored in a hash tree
  - Checking whether a candidate itemset C_k is in the transaction t takes O(1) time

Discussion of the Apriori algorithm

- Much faster than the Brute-force algorithm
  - It avoids checking all elements in the lattice
- The running time is in the worst case **$O(2^d)$**
  - Pruning really prunes in practice
- It makes multiple passes over the dataset
  - One pass for every level **k**
- Multiple passes over the dataset is inefficient when we have thousands of candidates and millions of transactions

AproriTid

- Objectives: as the size of the frequent itemsets increases,
  - reduce the length of each transaction
  - reduce the number of transactions necessary to check for support
- Method:
  - Database D is not used for counting support after the first pass.
  - The set $\overline{C}_k$<Tid, $\{X_k\}$> is used for counting support afterwards, where $\{X_k\}$ is a potentially large k-itemset present in the transaction with identifier Tid.

Making a single pass over the data: the AprioriTid algorithm

- The database is **not** used for counting support after the 1st pass!

- Instead information in data structure $\overline{C}_k$ is used for counting support in every step

  - $\overline{C}_k = \{<TID, \{X_k\}> \mid X_k$ **is a potentially frequent** k**-itemset in transaction with** $id=TID\}$

  - $\overline{C}_1$: corresponds to the original database (every item i is replaced by itemset $\{i\}$)

  - The member $\overline{C}_k$ corresponding to transaction t is $<t.TID, \{c \in C_k \mid c \text{ is contained in } t\}>$

Algorithm AprioriTid

1) L_1 = {large 1-itemsets};
2) $\overline{C}_1$ = database $\mathcal{D}$;
3) **for** ($k = 2$; $L_{k-1} \neq \emptyset$; k++) **do begin**
4) $\quad C_k$ = apriori-gen(L_{k-1}); // New candidates
5) $\quad \overline{C}_k = \emptyset$;
6) $\quad$ **forall** entries $t \in \overline{C}_{k-1}$ **do begin**
7) $\quad\quad$ // determine candidate itemsets in C_k contained
$\quad\quad$ // in the transaction with identifier t.TID
$\quad\quad C_t = \{c \in C_k \mid (c - c[k]) \in t.\text{set-of-itemsets} \wedge$
$\quad\quad\quad (c - c[k-1]) \in t.\text{set-of-itemsets}\}$;
8) $\quad\quad$ **forall** candidates $c \in C_t$ **do**
9) $\quad\quad\quad c$.count++;
10) $\quad\quad$ **if** ($C_t \neq \emptyset$) **then** $\overline{C}_k$ += $< t.\text{TID}, C_t >$;
11) $\quad$ **end**
12) $\quad L_k = \{c \in C_k \mid c.\text{count} \geq \text{minsup}\}$
13) **end**
14) Answer = $\bigcup_k L_k$;

AprioriTid Example (minsup=2)

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

$\overline{C}_1$

TID	Sets of itemsets
100	{{1},{3},{4}}
200	{{2},{3},{5}}
300	{{1},{2},{3},{5}}
400	{{2},{5}}

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

$\overline{C}_2$

TID	Sets of itemsets
100	{{1 3}}
200	{{2 3},{2 5},{3 5}}
300	{{1 2},{1 3},{1 5}, {2 3},{2 5},{3 5}}
400	{{2 5}}

L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

$\overline{C}_3$

TID	Sets of itemsets
200	{{2 3 5}}
300	{{2 3 5}}

L_3

itemset	sup
{2 3 5}	2

Discussion on the AprioriTID algorithm

- L_1 = {frequent 1-itemsets}
- $\overline{C}_1$ = database **D**
- **for** ($k=2$, $L_{k-1}' \neq$ empty; k++)

    C_k = GenerateCandidates(L_{k-1})

    $\overline{C}_k$ = {}

    for all entries **t** ϵ $\overline{C}_{k-1}$

    C_t= {c ϵ C_k | t[c-c[k]]=1 and t[c-c[k-1]]=1}

    for all **c ϵ C_t** {c.count++}

    if (**$C_t \neq$ {}**)

    append C_t to $\overline{C}_k$

    endif

    endfor

    L_k= {cϵ C_k | c.count >= minsup}

  endfor
- **return** $\mathbf{U}_k$ L_k

- One single pass over the data
- $\overline{C}_k$ is generated from $\overline{C}_{k-1}$
- For small values of **k**, $\overline{C}_k$ could be larger than the database!
- For large values of **k**, $\overline{C}_k$ can be very small

Apriori vs. AprioriTID

- *Apriori* makes multiple passes over the data while *AprioriTID* makes a single pass over the data

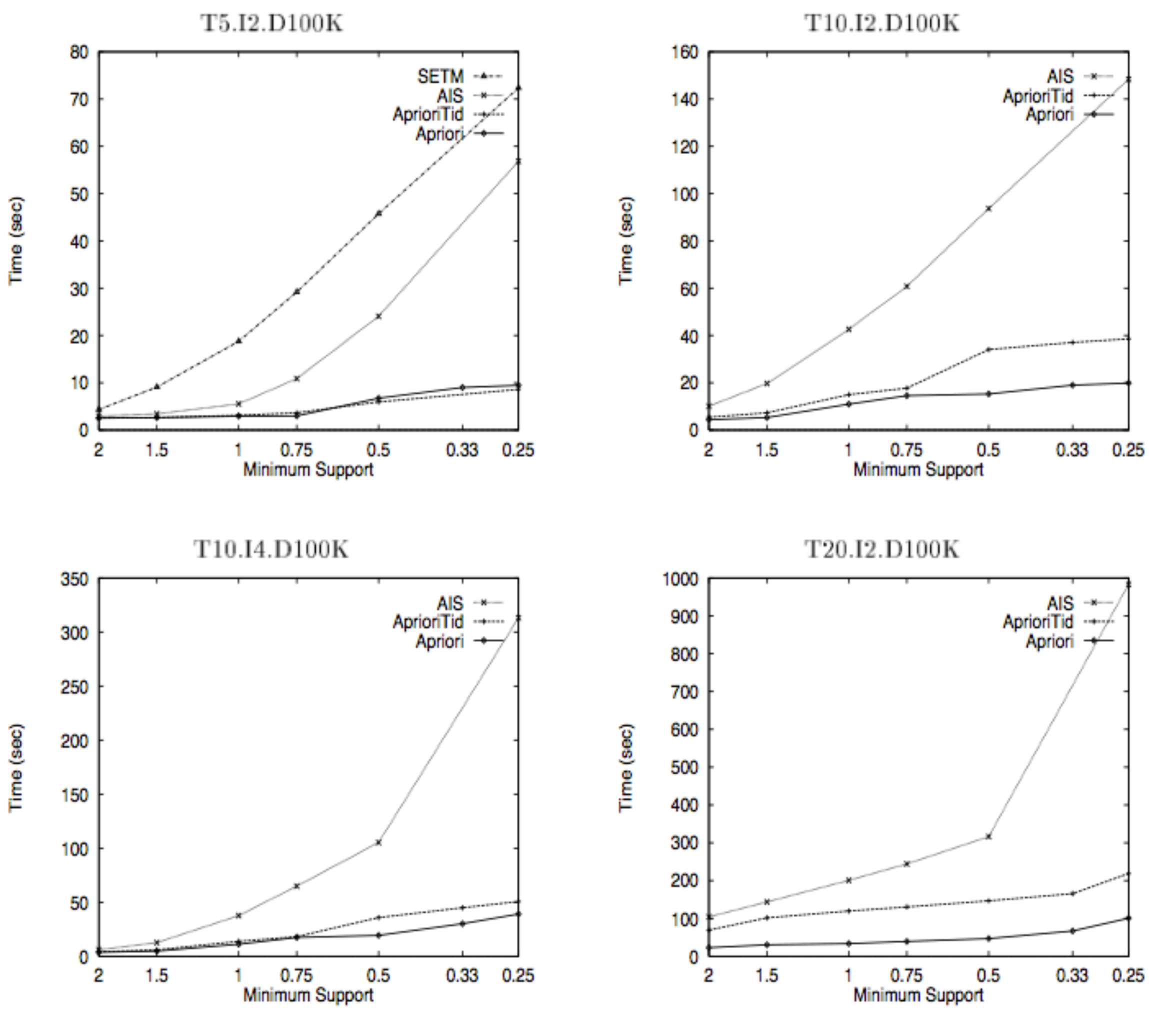
- *AprioriTID* needs to store additional data structures that may require more space than *Apriori*

- Both algorithms need to check all candidates' frequencies in every step

Practice Problem

TID	items
T1	I1, I2, I3, I5
T2	I2, I4
T3	I2, I6
T4	I1, I2, I4, I5
T5	I1, I2
T6	I1, I2, I3, I5
T7	I1, I2, I3

Apriori vs. AprioriTid Performance

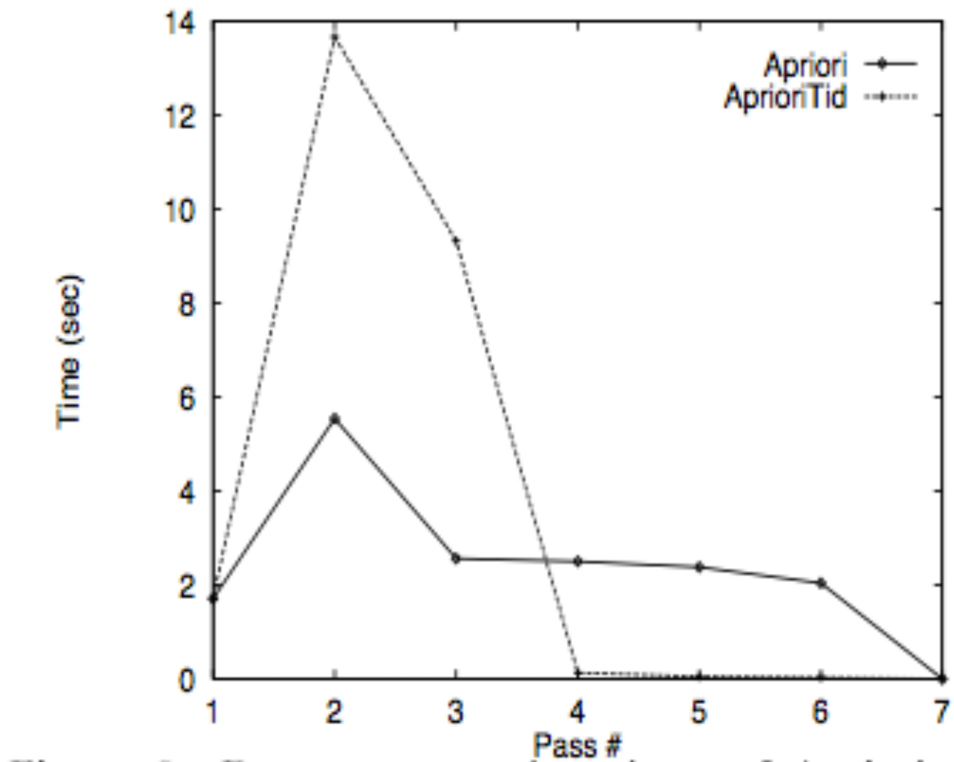


- AprioriTid replaces a pass over the original dataset by A pass over the set $\overline{C}_k$
  ➔ AprioriTid is very effective in later passes when the size of $\overline{C}_k$ becomes small compared to the size of the database.

- AprioriTid beats Apriori when its $\overline{C}_k$ sets can fit in memory.

- When $\overline{C}_k$ does not fit in memory, There is a jump in the execution time for AprioriTid.

Algorithm AprioriHybrid



Use Apriori for the initial passes, and switch to AprioriTid when it expects that the set $\overline{C}_k$ at the end of the pass will fit in memory.

Apriori vs. AprioriTid (T10.I4.D100k, minsup = 0.75%)