

# Data Mining

## Association Analysis: Basic Concepts and Algorithms

---

### Lecture Notes for Chapter 6



# Contents

## Association Rule Mining

# 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

## Market-Basket transactions

<i>TID</i>	<i>Items</i>
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

$\{\text{Diaper}\} \rightarrow \{\text{Beer}\},$   
 $\{\text{Milk, Bread}\} \rightarrow \{\text{Eggs, Coke}\},$   
 $\{\text{Beer, Bread}\} \rightarrow \{\text{Milk}\},$

**Implication means co-occurrence,  
not causality!**

# Definition: Frequent Itemset

## 항목집합(Itemset)

- A collection of one or more items
  - ◆ Example: {Milk, Bread, Diaper}
- k-itemset
  - ◆ An itemset that contains k items

## 지지도 카운트(Support count) ( $\sigma$ )

- 특정 항목 집합을 포함하는 transaction 수
- 예:  $\sigma(\{\text{Eggs}\}) = 1$ ,  $\sigma(\{\text{Milk, Bread, Diaper}\}) = 2$

## 지지도(Support) $s$

- 항목집합이 나타나는 트랜잭션의 비율
- 예:  $s\{\text{Eggs}\} = 1/5 = 0.2$   
 $s(\{\text{Milk, Bread, Diaper}\}) = 2/5$

## 빈발항목집합(Frequent Itemset)

- 지지도가 주어진 임계치  $\text{minsup}$ 보다 큰

### 항목집합

2-itemset

4-itemset

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

만약  $\text{minsup} = 0.3$ 이라면, {Eggs}은 빈발하지 않으며, {Milk, Bread, Diaper}은 빈발함

# Definition: Association Rule

## 연관규칙이란?

- X와 Y가 항목집합 일때,  $X \rightarrow Y$  형태로 나타나는 함축 표현(implication expression)
- 예: {Milk, Diaper}  $\rightarrow$  {Bread}

## 연관규칙의 평가척도

- **지지도(Support): s**

$$s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$
  - X와 Y를 함께 포함하는 트랜잭션 비율
  - 규칙이 얼마나 중요하며, 유용한지를 알 수 있음 (낮은 지지도 갖는 규칙은 우연으로 볼 수 있음)

- **신뢰도(Confidence): c**

$$c(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$
  - X를 포함한 트랜잭션 중에 Y가 나타나는 비율
  - 규칙이 얼마나 믿을 만 한가?  
(가정과 결론이 얼마나 타이트한 관련성 있는지를 나타냄)

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

Support Count:  $\sigma(X) = |\{t_i \mid X \subseteq t_i, t_i \in T\}|$

$T = \{t_1, t_2, t_3, \dots, t_N\}$  는 모든 transaction 집합

**Example:**

{Milk, Diaper}  $\Rightarrow$  Beer

$$s = \frac{\sigma(\text{Milk, Diaper, Beer})}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk, Diaper, Beer})}{\sigma(\text{Milk, Diaper})} = \frac{2}{3} = 0.67$$

# Association Rule Mining Task

Given a set of transactions  $T$ , the goal of association rule mining is to find all rules having

- Support  $\geq$  *minsup* threshold
  - Confidence  $\geq$  *minconf* threshold
- 최소지지도 (Minsup: minimum support)
  - 최소신뢰도 (Minconf: minimum confidence)
  - 연관규칙 마이닝 = 왼쪽 조건을 만족하는 모든 규칙을 찾는 작업

Brute-force approach:

- List all possible association rules
  - Compute the support and confidence for each rule
  - Prune rules that fail the *minsup* and *minconf* thresholds
- ⇒ **Computationally prohibitive!**

# Mining Association Rules

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:

$\{\text{Milk, Diaper}\} \rightarrow \{\text{Beer}\}$  (s=0.4, c=0.67)  
 $\{\text{Milk, Beer}\} \rightarrow \{\text{Diaper}\}$  (s=0.4, c=1.0)  
 $\{\text{Diaper, Beer}\} \rightarrow \{\text{Milk}\}$  (s=0.4, c=0.67)  
 $\{\text{Beer}\} \rightarrow \{\text{Milk, Diaper}\}$  (s=0.4, c=0.67)  
 $\{\text{Diaper}\} \rightarrow \{\text{Milk, Beer}\}$  (s=0.4, c=0.5)  
 $\{\text{Milk}\} \rightarrow \{\text{Diaper, Beer}\}$  (s=0.4, c=0.5)

## Observations:

- All the above rules are binary partitions of the same itemset:  
 $\{\text{Milk, Diaper, Beer}\}$  // 위의 rule들은 모두 이 3개의 항목을 가짐
- Rules originating from the same itemset have identical support **but can have different confidence**

➤ 이 때문에, 지지도(support)와 신뢰도(confidence)를 분리하여 마이닝 할 필요 있음

# Contents

## Frequent Itemset Generation & Apriori Algorithm



# Mining Association Rules

Transaction DB에서 연관 규칙을 찾아야 함

연관규칙을 만들기 위한 2단계 접근법(Two-step approach):

## 1. 빈발 항목집합 생성(Frequent Itemset Generation)

- Transaction DB에서 많이 존재하는 빈발 항목집합을 찾아야 함
- 빈발 항목집합:  $\text{support} \geq \text{minsup}$ 을 만족하는 항목집합

## 2. 연관 규칙 생성(Rule Generation)

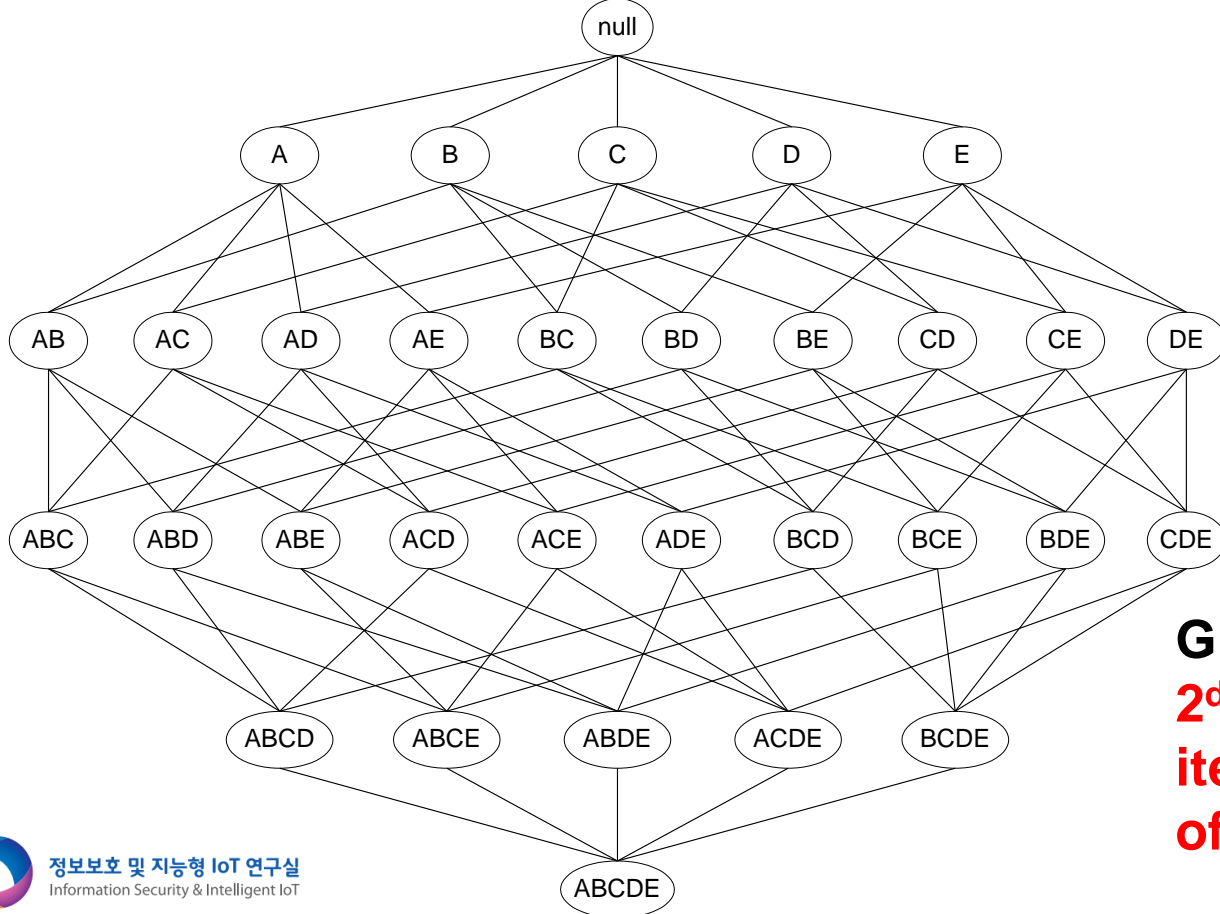
- 각 빈발 항목집합을 두 개의 항목집합으로 분리하여  $\text{confidence} \geq \text{minconf}$ 를 만족하는 연관규칙들을 도출함
- (Generate high confidence rules from each frequent itemset, where each rule is a binary partitioning of a frequent itemset)

여기서, Transaction DB에서 빈발 항목집합을 찾아내는 과정은 computationally expensive 함

# Frequent Itemset Generation

- Lattice 기반 항목집합 열거

- 아래의 lattice(격자) 구조는 모든 가능한 항목집합 목록 열거에 도움됨
- $I=\{a,b,c,d,e\}$ 에 대한 항목 집합 격자. D개의 항목을 포함하는 데이터 집합은  $2^d-1$ 개의 빈발 항목 집합 생성 가능

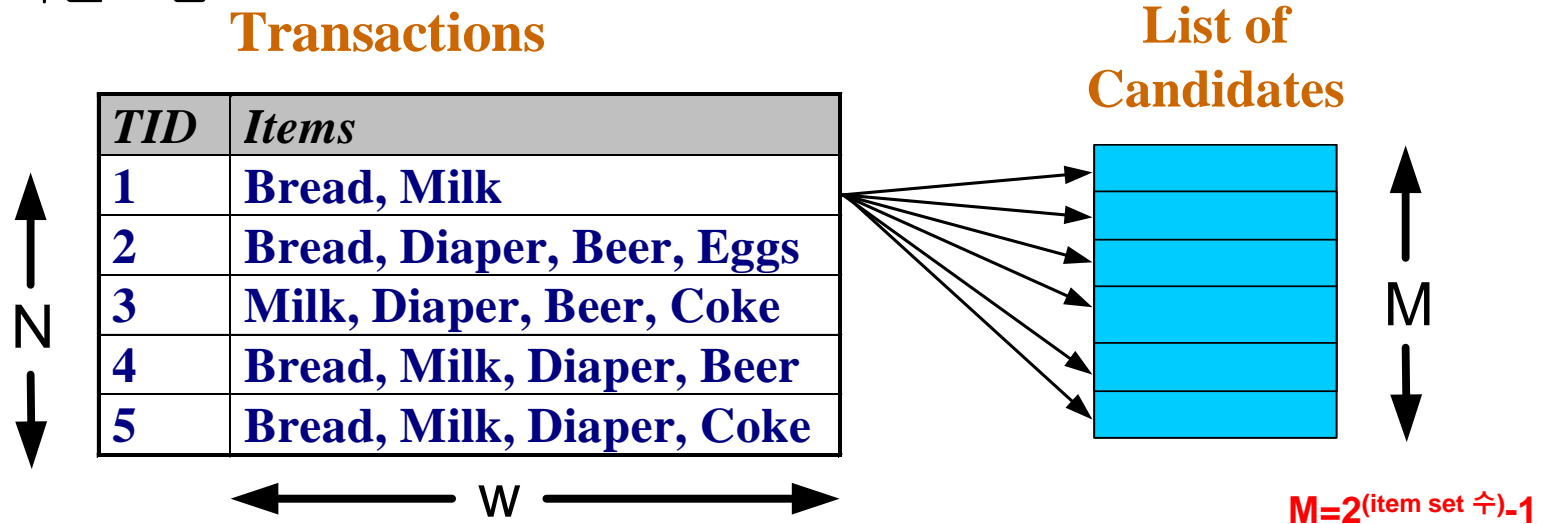


Given  $d$  items, there are  $2^d$  possible candidate itemsets (The number of subsets)

# Frequent Itemset Generation

Brute-force approach:

- Lattice의 모든 itemset은 candidate frequent itemset이 됨
- Transaction 데이터베이스를 스캔하면서, 각 후보에 대해 support 계산 및 카운트함



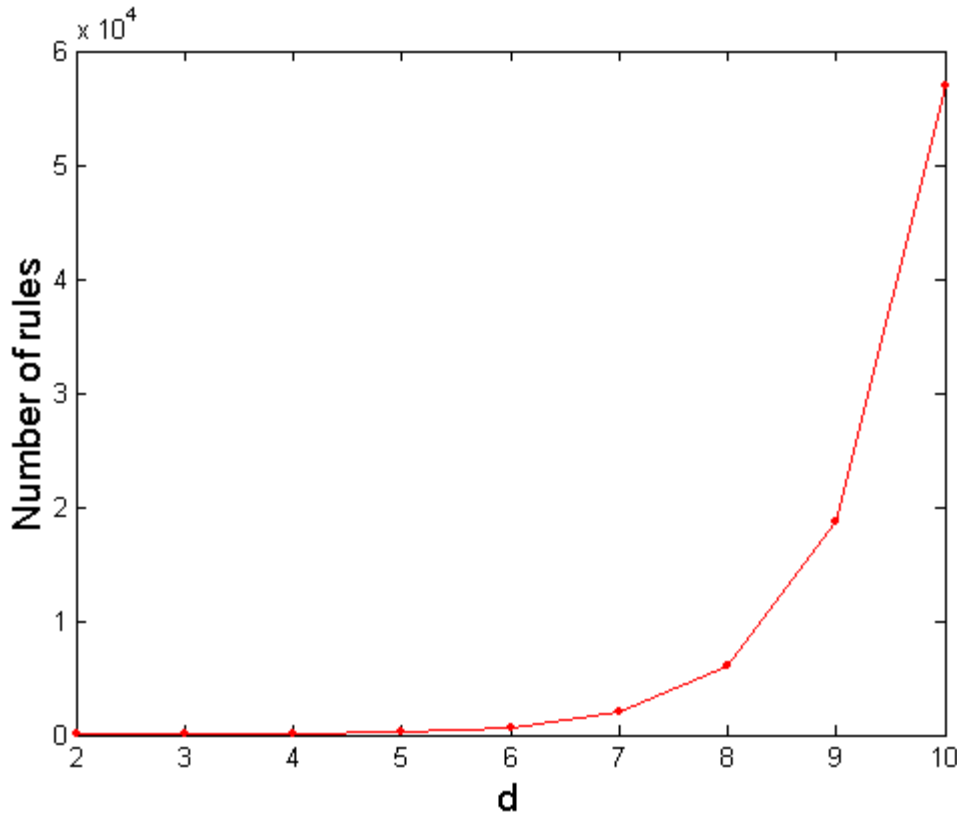
- 모든 후보에 대해 각 transaction을 매치함
- 복잡도 ~  $O(NMw) \Rightarrow$  Expensive since  $M = 2^d$  !!!

( M:candidate frequent itemset의 개수, N:Transaction 수, w:최대 transaction 폭 )

# Computational Complexity

항목이  $d$ 개 주어졌을 때,

- 가능한 항목집합의 개수 =  $2^d$
- 가능한 연관규칙의 개수 =  $3^d - 2^{d+1} + 1$



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

If  $d=6$ ,  $R = 602$  rules

# Frequent Itemset 생성 전략

Reduce the **number of candidates** (M)

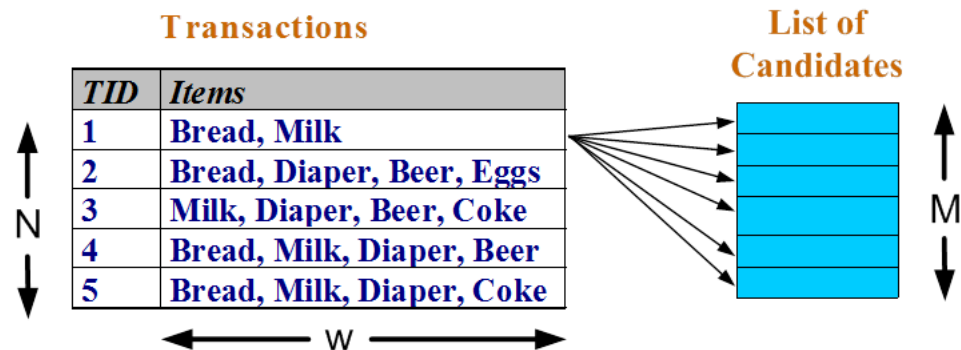
- Complete search:  $M=2^d$
- **Use pruning techniques to reduce M**

Reduce the **number of transactions** (N)

- Reduce size of N as the size of itemset increases
- Using the **DHP(Direct Hashing & Pruning)** and **vertical-based mining algorithms**

Reduce the **number of comparisons** (NM)

- Use **efficient data structures** to store the candidates or transactions
- No need to match every candidate against every transaction



# Reducing Number of Candidates

## Apriori principle:

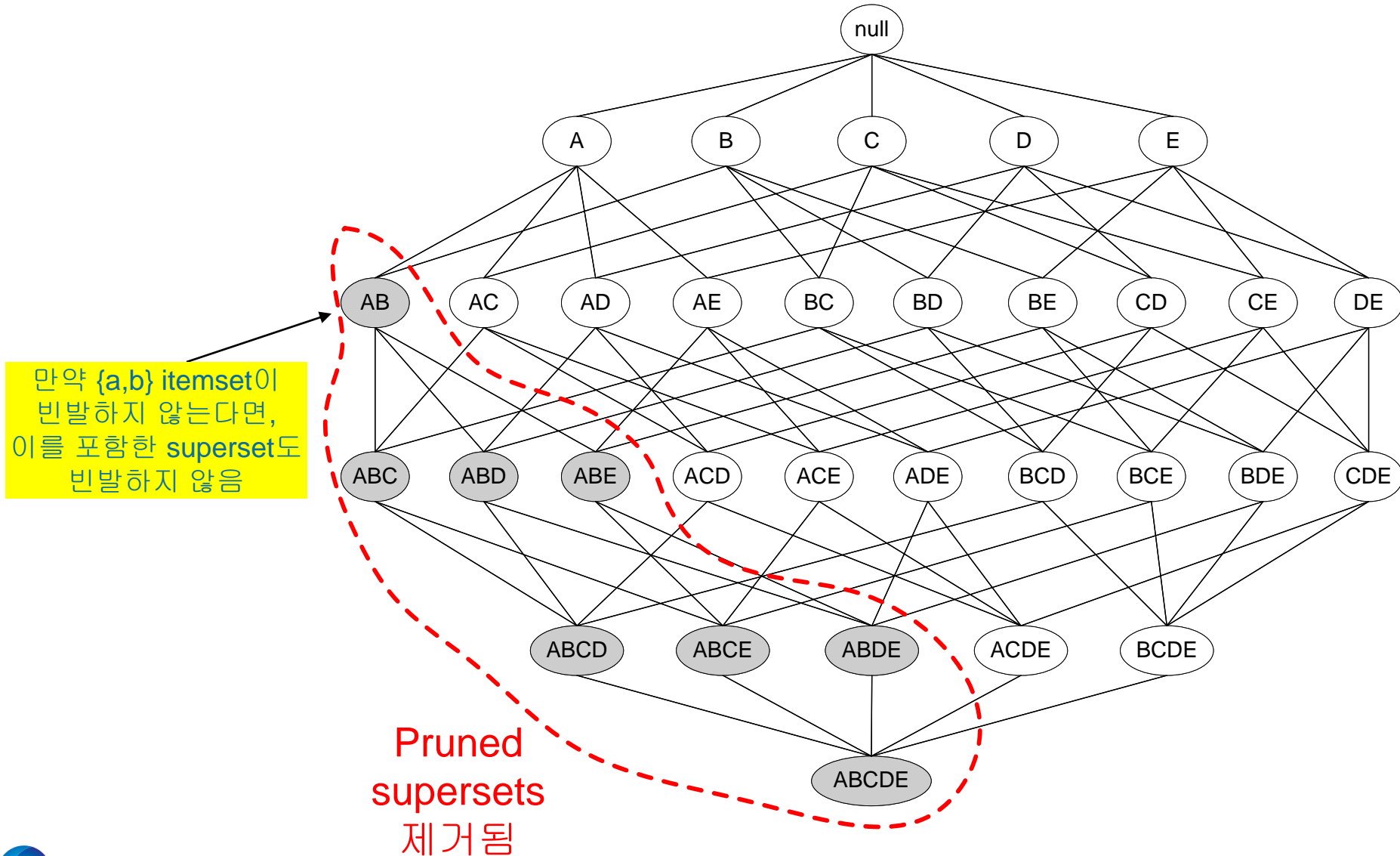
- 어떤 항목집합이 빈발하다면, 그 항목집합의 모든 부분집합도 빈발함
- 예: {Milk, Bread, Diaper}가 빈발 항목집합이면, 이의 부분집합인 {Milk, Bread}, {Bread, Diaper} 등도 빈발 항목집합이다.

Apriori principle holds **due to the following property of the support measure:**

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

- 어떤 항목집합의 지지도는 그 부분집합들의 지지도를 넘을 수 없다!
  - ◆ 즉, 어떤 짧은 서브 패턴이 자주 나오지 않는다는 것을 알고 있다면, 이 서브패턴에서 아이템이 더 붙어서 나오는 슈퍼 패턴은 절대 더 자주 나오지 않음
- 이는 지지도가 anti-monotone 성질을 가지기 때문이다. ( $a > b \rightarrow f(a) < f(b)$ )

# Illustrating Apriori Principle



# Illustrating Apriori Principle

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

Items (1-itemsets)

(1) Coke과 Eggs는 상대적으로 빈도가 적으므로 drop됨 (3이하이면 drop)

Minimum Support = 3

(2) Frequent itemset의 항목이 (1)에서 4개이므로, 여기서 2 항목 itemset을 만들.  $4C_2 = 6$

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

Pairs (2-itemsets)

(3) 다시 3이하인, 두개 itemset을 drop한 후, 3 itemset을 만들

Triplets (3-itemsets)

Itemset	Count
{Bread, Milk, Diaper}	3

If every subset is considered,  
 ${}^6C_1 + {}^6C_2 + {}^6C_3 = 41$  (Brute-force)

With support-based pruning,  
 $6 + 6 + 1 = 13$ 로 줄어듦 (A Priori)



# Illustrating Apriori Principle

**Sup<sub>min</sub> = 2**

Database

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

Candidate itemset

$C_1$

1<sup>st</sup> scan

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

Support 값이 2보다 작은 D itemset 탈락시킴

Frequent itemset

$L_1$

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

$C_2$

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

2<sup>nd</sup> scan

$C_2$

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

$L_2$

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

$C_3$

Itemset
{B, C, E}

3<sup>rd</sup> scan

$L_3$

Itemset	sup
{B, C, E}	2

Frequent 3-itemset  
생성

- {A, C, E}는 없음 Why? {A,E}가 L2에 없으므로
- {B, C, E}는 있음. {B,C} {C,E} {B,E} 모두 L2에 있으므로

# The Apriori Algorithm (Pseudo-Code)

$C_k$ : Candidate itemset of size  $k$

$L_k$ : frequent itemset of size  $k$

$L_1 = \{\text{frequent items}\};$

**for** ( $k = 1; L_k \neq \emptyset; k++$ ) **do begin**

$C_{k+1}$  = candidates generated from  $L_k$ ;

**for each** transaction  $t$  in database **do**

increment the count of all candidates in  $C_{k+1}$  that are contained in  $t$

$L_{k+1}$  = candidates in  $C_{k+1}$  with min\_support

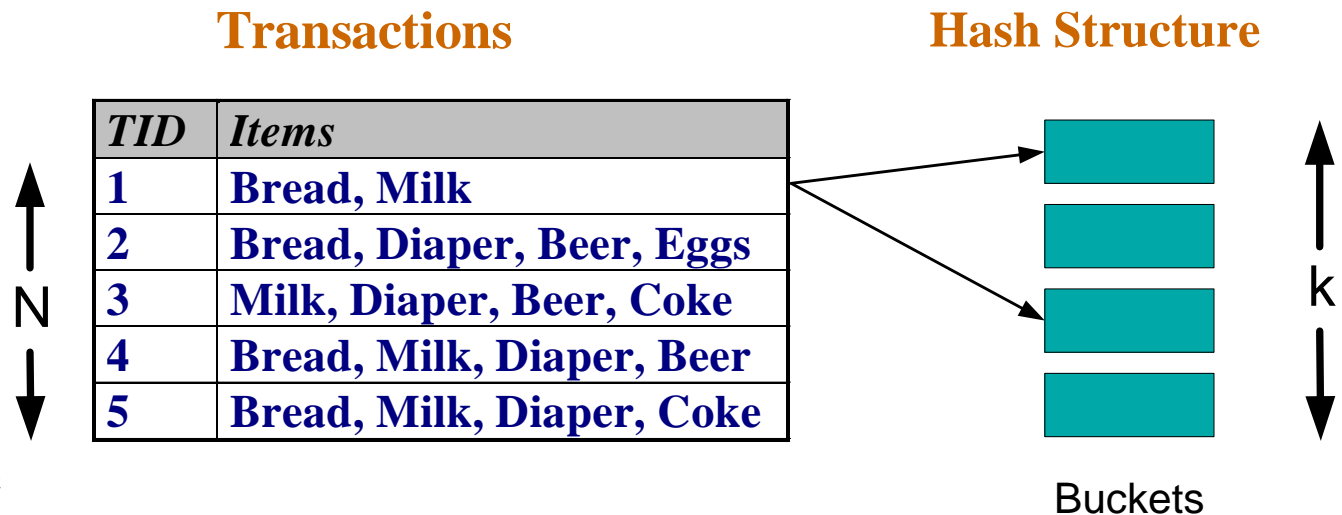
**end**

**return**  $\cup_k L_k$ ;

# Reducing Number of Comparisons

## Candidate counting:

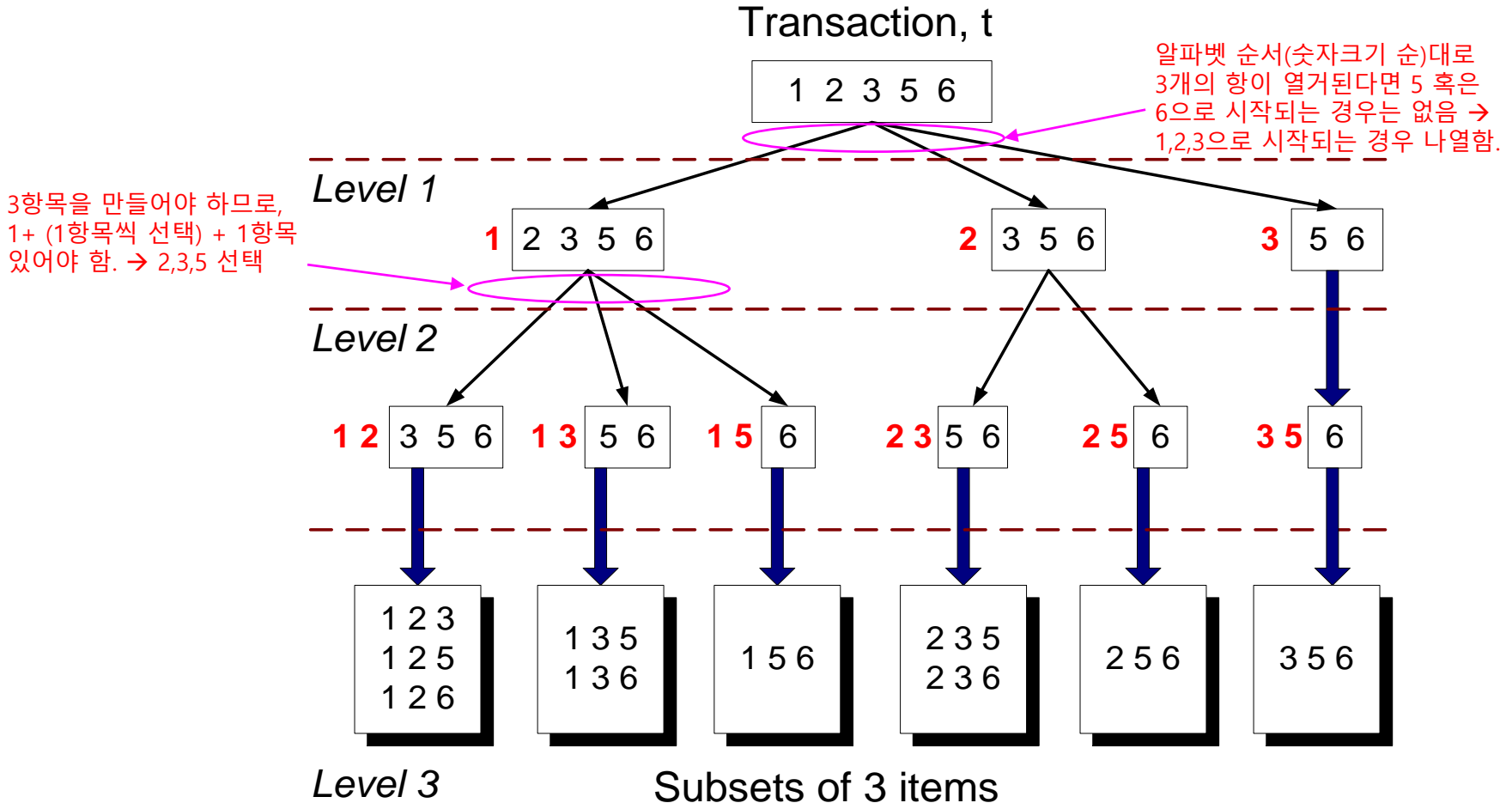
- 빈발항목이라고 결정내리기 위해선 먼저 candidate itemset에서 support 값을 계산해야 함
- Support 값 계산을 위해선 Transaction DB와 모두 비교 필요 → 높은 계산 복잡도 → Candidate를 hash structure에 저장하여 효율적 비교
- ◆ Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets
- ◆ That is, we use a hash tree structure to reduce the number of candidates in  $C$  that are checked for a data-sequence



# Support Counting을 위한 부분집합 열거

Transaction  $t = \{1, 2, 3, 5, 6\}$ 에 속한 3-itemset을 모두 열거하는 체계적 방법

Transaction으로부터 3-itemset을 열거한 후, 후보항목(Candidate list) 각각과 비교하여, 존재하면 해당 후보 항목 지지도 카운트는 증가됨



# Support Counting using a Hash Tree

Candidate itemset이 주어지면 → Hash tree에서 서로 다른 **bucket**에 나눠 저장됨

- 이때 지지도 계산시, 각 **transaction**에 포함된 **itemset**도 그것들에 적합한 **bucket**들로 해쉬됨
- 즉, **transaction**으로부터 **itemset**을 **hash function** 규칙에 따라 **hash tree**의 각 **leaf** 노드로 할당함
- 이로써, **transaction**에 속한 각 **itemset**을 각 **itemset**마다 비교하는 대신, 아래 그림처럼 오직 같은 **bucket**에 **candidate itemset**과 비교함

(1) Candidate itemset은 hash tree에 나눠 저장됨

(2) Transaction ID:2,3,4는 Hash Tree의 왼쪽 leaf node (bucket)에만 비교됨

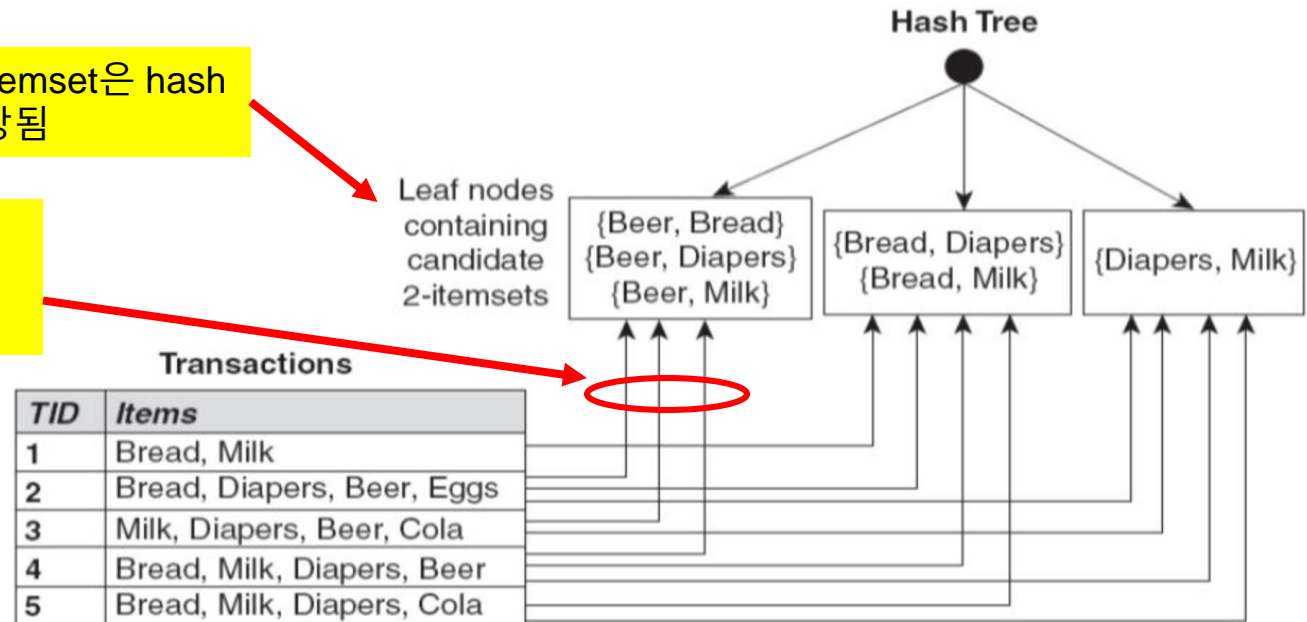


Figure 6.10. Counting the support of itemsets using hash structure

# Support Counting using a Hash Tree

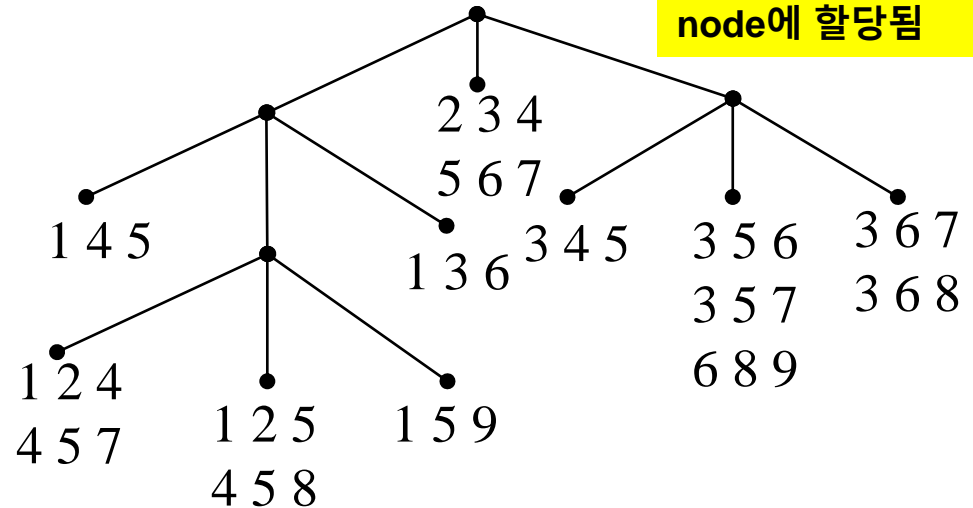
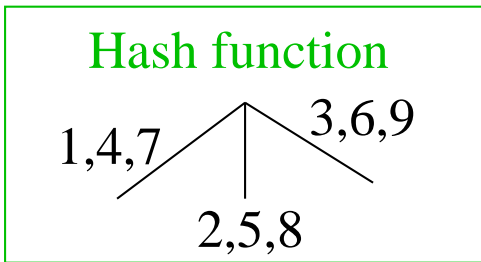
Suppose you have 15 candidate itemsets of length 3:

{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)

15 개의 candidate itemsets이 leaf node에 할당됨

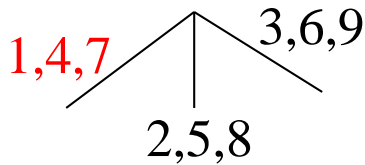


# Support Counting using a Hash Tree

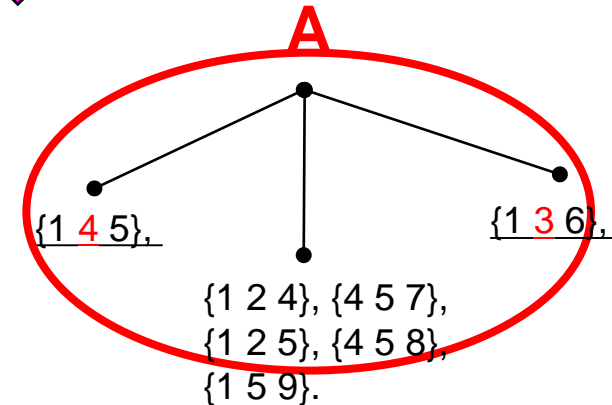
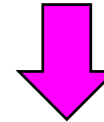
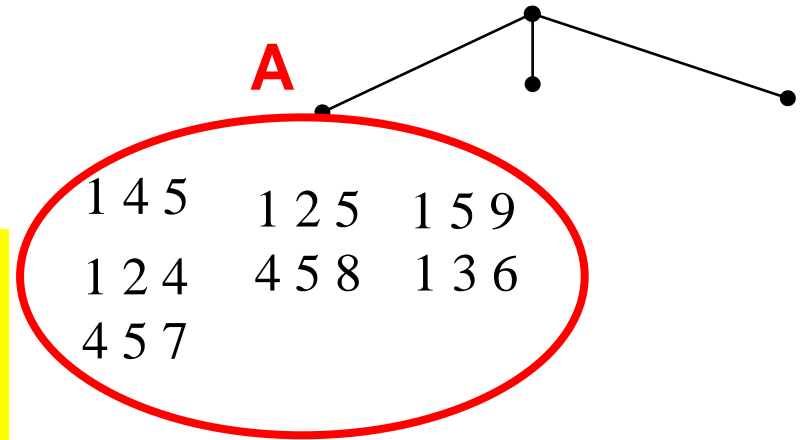
Suppose you have 15 candidate itemsets of length 3:

{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}

## Hash function



- Hash function(해쉬 규칙)에 의해,
- Candidate itemsets에서 1,4,7로 시작되는 것은 A 가지에 모두 할당함
- 즉, 7개를 왼쪽 가지로 할당함
- {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}

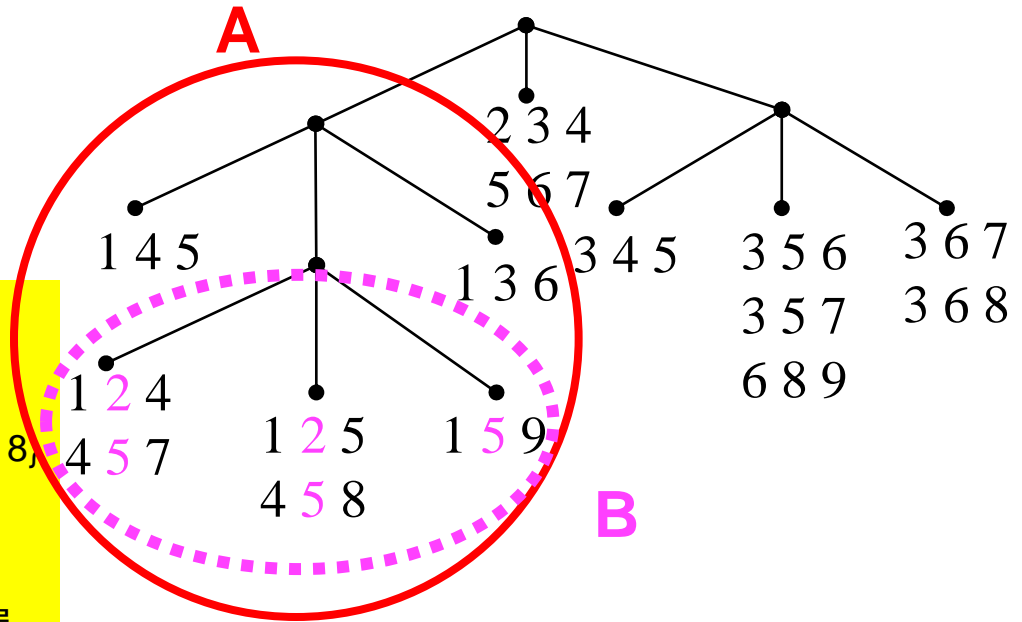
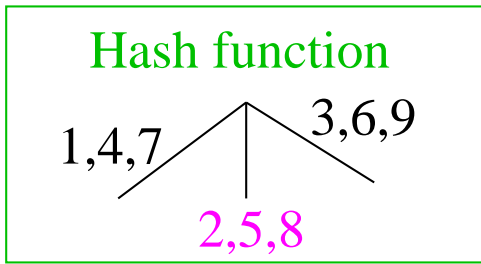


- Hash function에 의해, 즉, 7개중에서 두번째 항이 1,4,7인 {1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, 1개를 왼쪽 노드로
- 오른쪽 노드는 3,6 9에 해당하는 {1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, 1개 즉, {1 3 6}을 배치함
- 나머지, 즉, 중간 노드가 2,5,8은 중간 노드로.. {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}.
- 이를 반복해서 해쉬 트리를 구성함

# Support Counting using a Hash Tree

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}



- candidate itemsets에서 1,4,7로 시작되는 것은 A 가지에 모두 할당되어야 함
- 즉, {1 4 5} {1 2 4} {4 5 7} {1 2 5} {4 5 8}, {1 5 9} {1 3 6}. 총 7개의 itemset이 A 가지에 할당됨
- 각 leaf 노드에 최대 3개만 할당가능하다면 Hash Tree는 depth를 늘림 → B tree 추가됨
- B tree에서는 두번째 값이 2,5 8것을 갖게 되고, 세번째 값에 따라 왼쪽, 가운데, 오른쪽 node로 분배됨

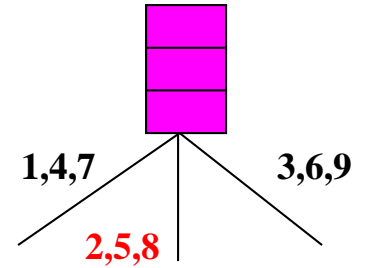


# Hash tree

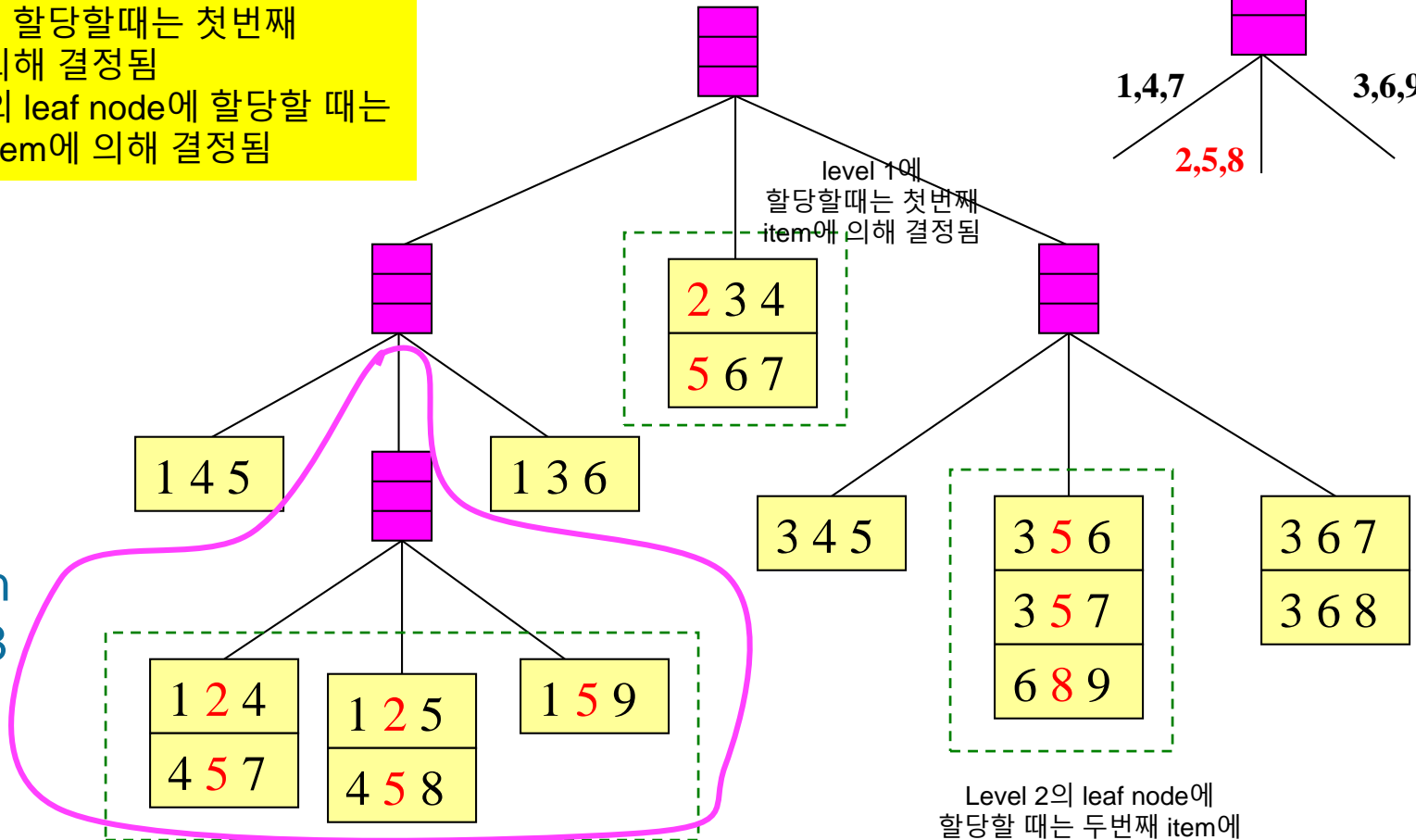
- 주어진 candidate itemset을 hash tree에 할당하는 방법
- Candidate itemset을 Hash tree의 level 1에 할당할 때는 첫번째 item에 의해 결정됨
- Level 2의 leaf node에 할당할 때는 두번째 item에 의해 결정됨

## Candidate Hash Tree

## Hash Function

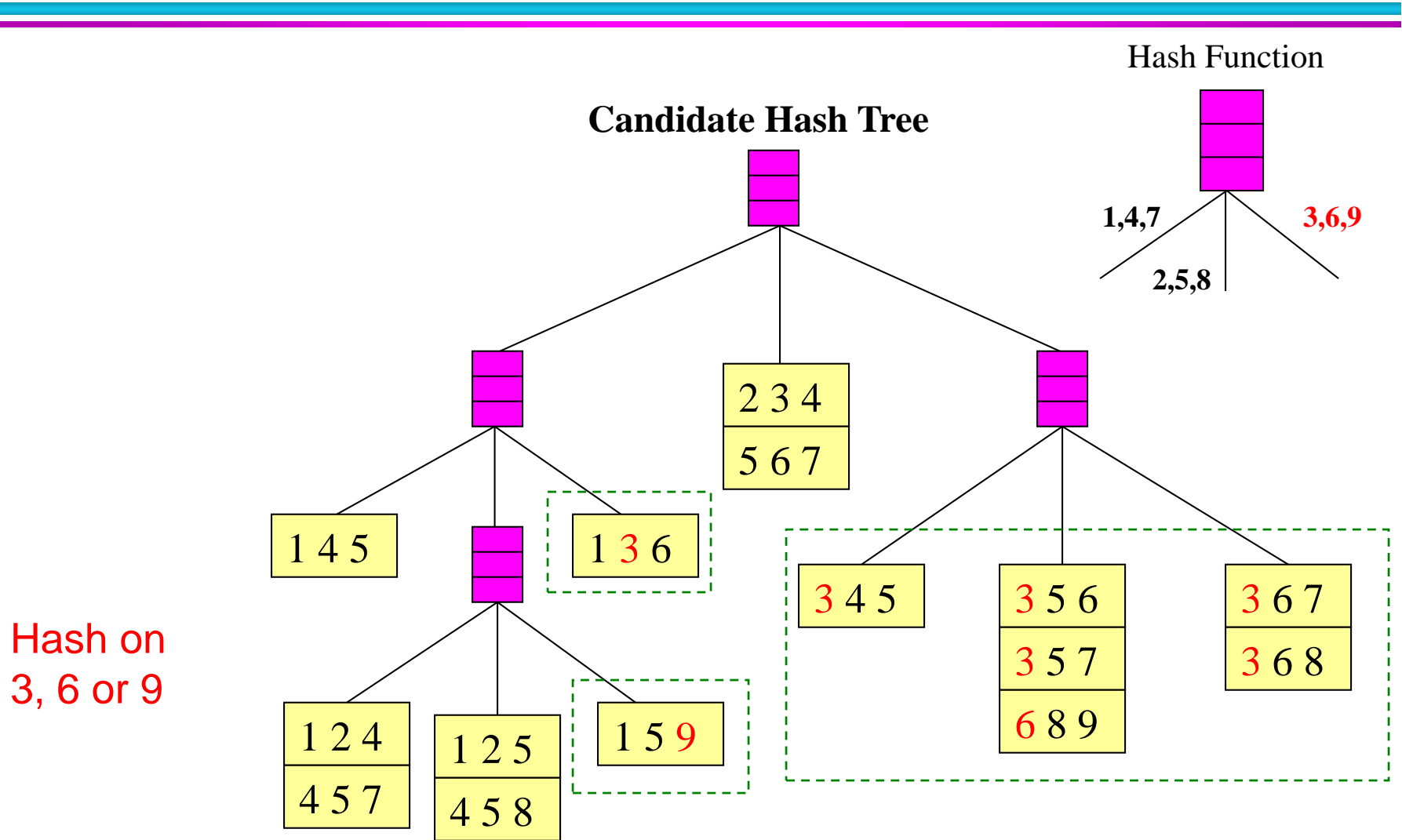


Hash on  
2, 5 or 8

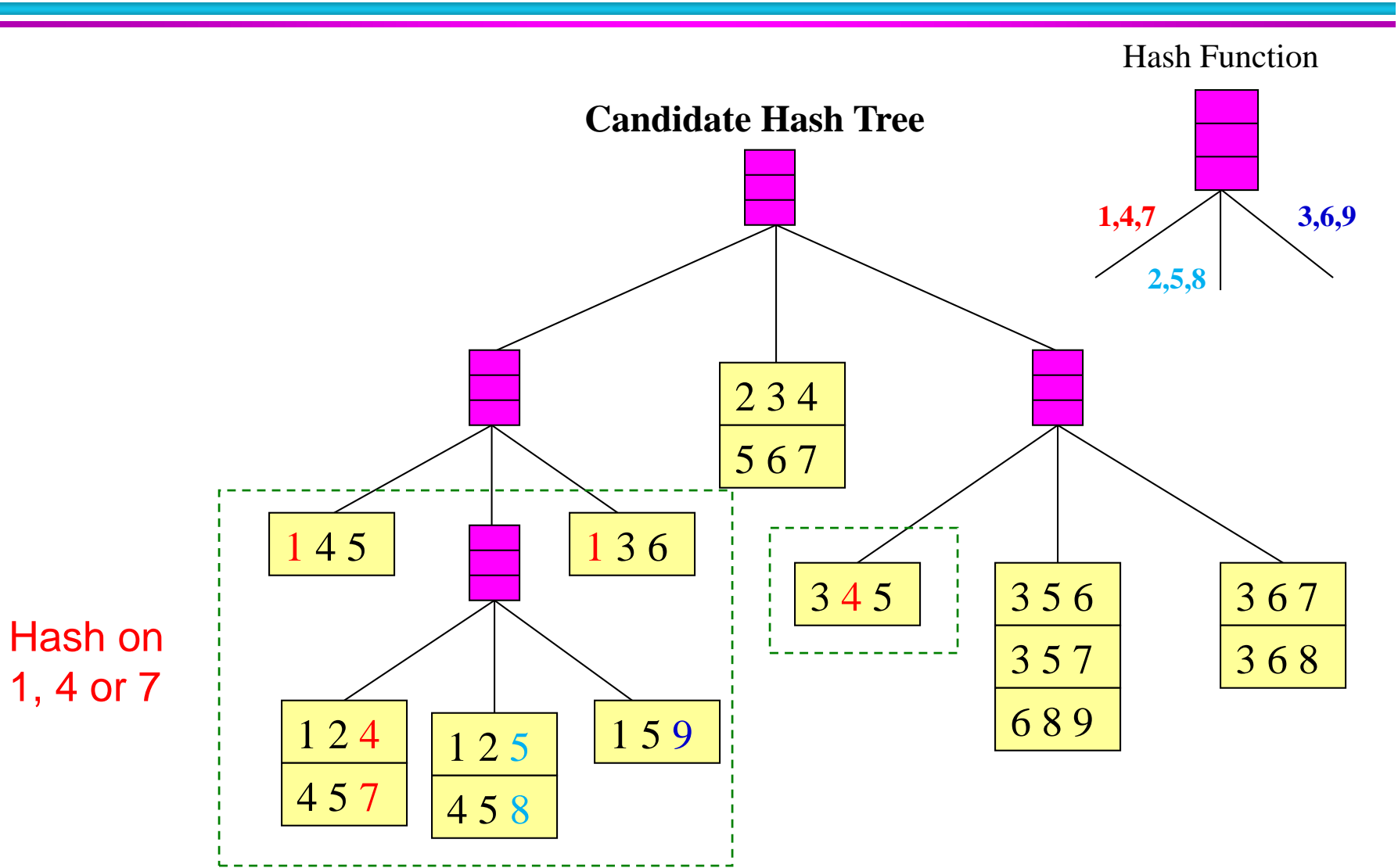


Level 2의 leaf node에 할당할 때는 두번째 item에 의해 결정됨

# Hash tree

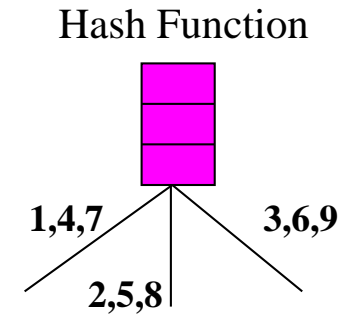
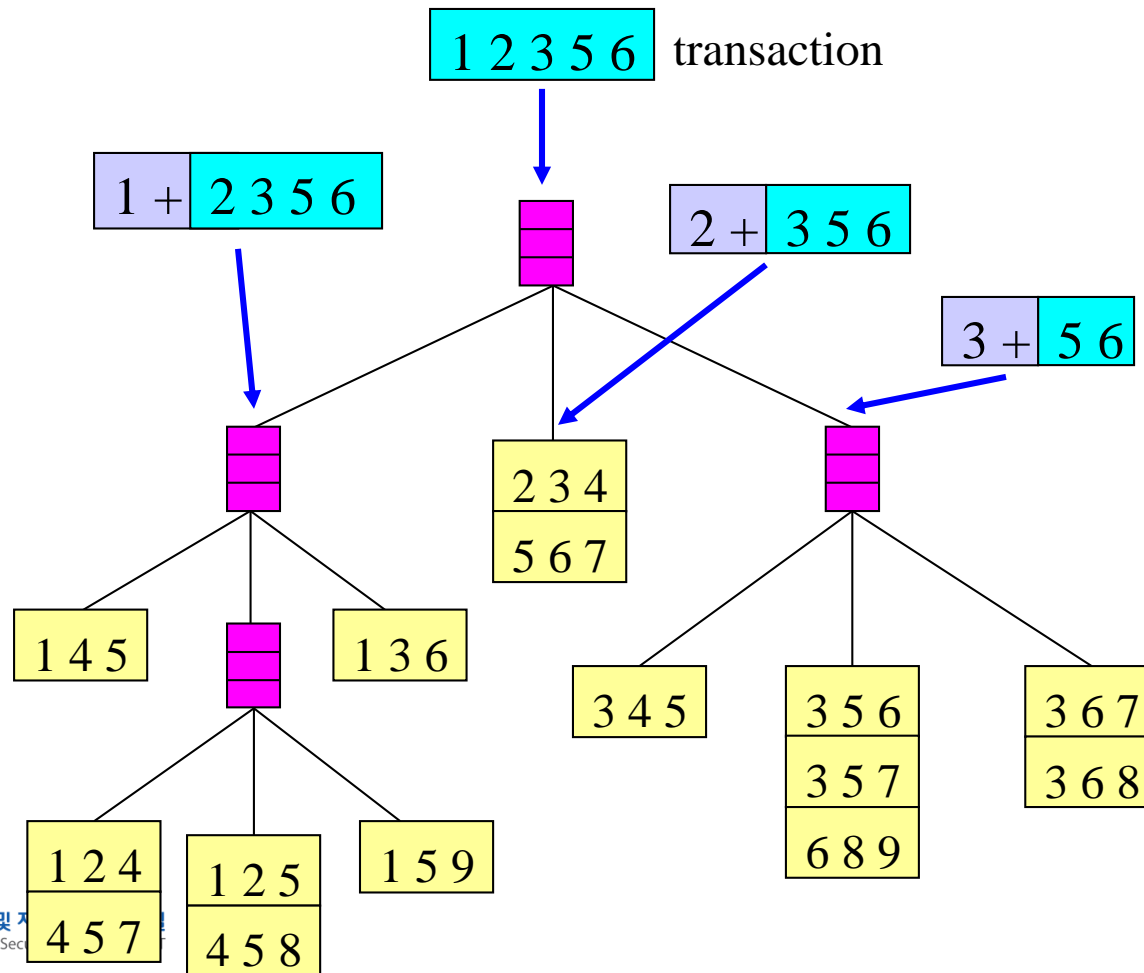


# Hash tree



# Support Counting using a Hash Tree

- Transaction {1 2 3 5 6} 은 아래처럼 됨

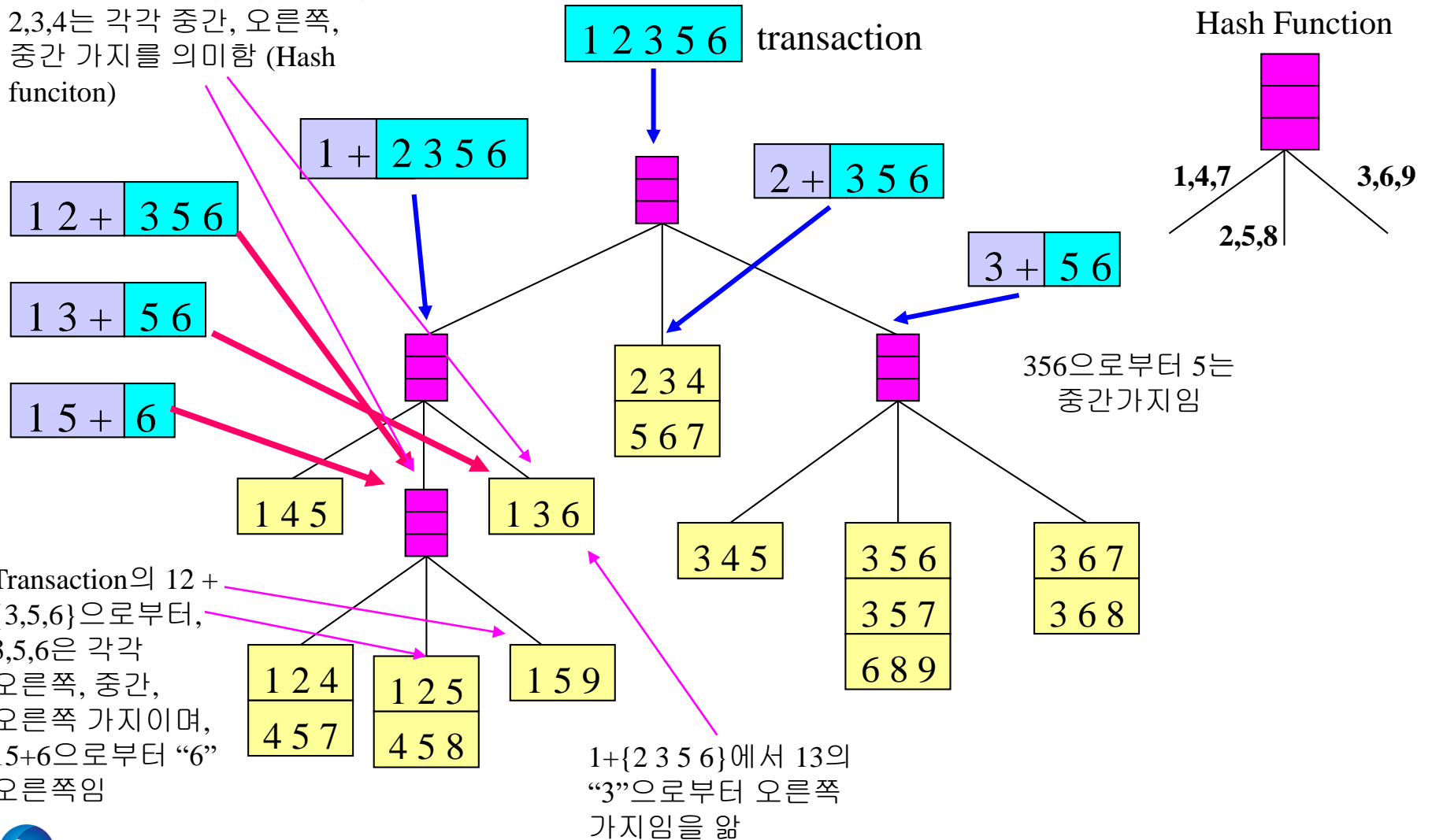


Transaction의 1,2,3은 각각 트리의 왼쪽, 중간, 오른쪽 방문을 의미함(Hash function)

# Support Counting using a Hash Tree

Transaction의 12, 13, 15에서,  
2,3,4는 각각 중간, 오른쪽,  
중간 가지를 의미함 (Hash  
function)

Transaction의 1,2,3은왼쪽, 중간, 오른쪽 가지.



# Support Counting using a Hash Tree

Example) Transaction  $t = \{1,2,3,5,6\}$

15 candidate itemsets of length 3:

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

