Data Mining and Analysis

Market Basket Analysis: 2

CSCE 676 :: Fall 2019
Texas A&M University
Department of Computer Science & Engineering
Prof. James Caverlee

Resources

MMDS: Mining of Massive Datasets [http://www.mmds.org/mmds/v2.1/ch06-assocrules.pdf]

Tan, Steinbach, Karpatne, Kumar. Introduction to Data Mining [https://www-users.cs.umn.edu/~kumar001/dmbook/slides/chap5-association_analysis.pdf]

Carlos Castillo course on Data Mining [https://github.com/chatox/data-mining-course]

Vagelis Papalexakis course on Data Mining [https://www.cs.ucr.edu/~epapalex/teaching/235_S19/index.html]

Finding Association Rules

Finding Association Rules

Problem: Find all association rules with support $\geq$ minsup and confidence $\geq$ minconf

Step 1: Find all frequent item sets I (with support $\geq$ minsup)

expensive!

Step 2: Generate association rules (with confidence $\geq$ minconf)

easy-ish!

Step 2: Generating the Rules

Step 2: Generating the Rules

Given frequent item sets, how do we actually generate the rules?

For each frequent item set I   // sup(I) ≥ minsup

For each possible partition X and Y, where Y = I – X

Check if conf(X⇒Y) ≥ minconf

Use the confidence monotonicity property (next slide) to reduce search space

Confidence Monotonicity

In general, confidence does not have a monotonicity property

conf(ABC $\rightarrow$ D) can be larger or smaller than conf(AB $\rightarrow$ D)

But confidence of rules generated from the same itemset has a monotonicity property!

Confidence Monotonicity

Confidence monotonicity property:

Let X_S, X_L, I be itemsets;
assume $X_S \subset X_L \subset I$

Then:

$$\mathrm{conf}(X_L \Rightarrow I - X_L) \geq \mathrm{conf}(X_S \Rightarrow I - X_S)$$

Example: Confidence Monotonicity

$$\mathrm{conf}(X_L \Rightarrow I - X_L) \geq \mathrm{conf}(X_S \Rightarrow I - X_S)$$

Suppose {A,B,C,D} is a frequent itemset, then:

c(ABC → D) ≥ c(AB → CD)

and

c(AB → CD) ≥ c(A → BCD)

Example: Generating the Rules

Suppose {A,B,C,D} is a frequent itemset, then the candidate rules are:

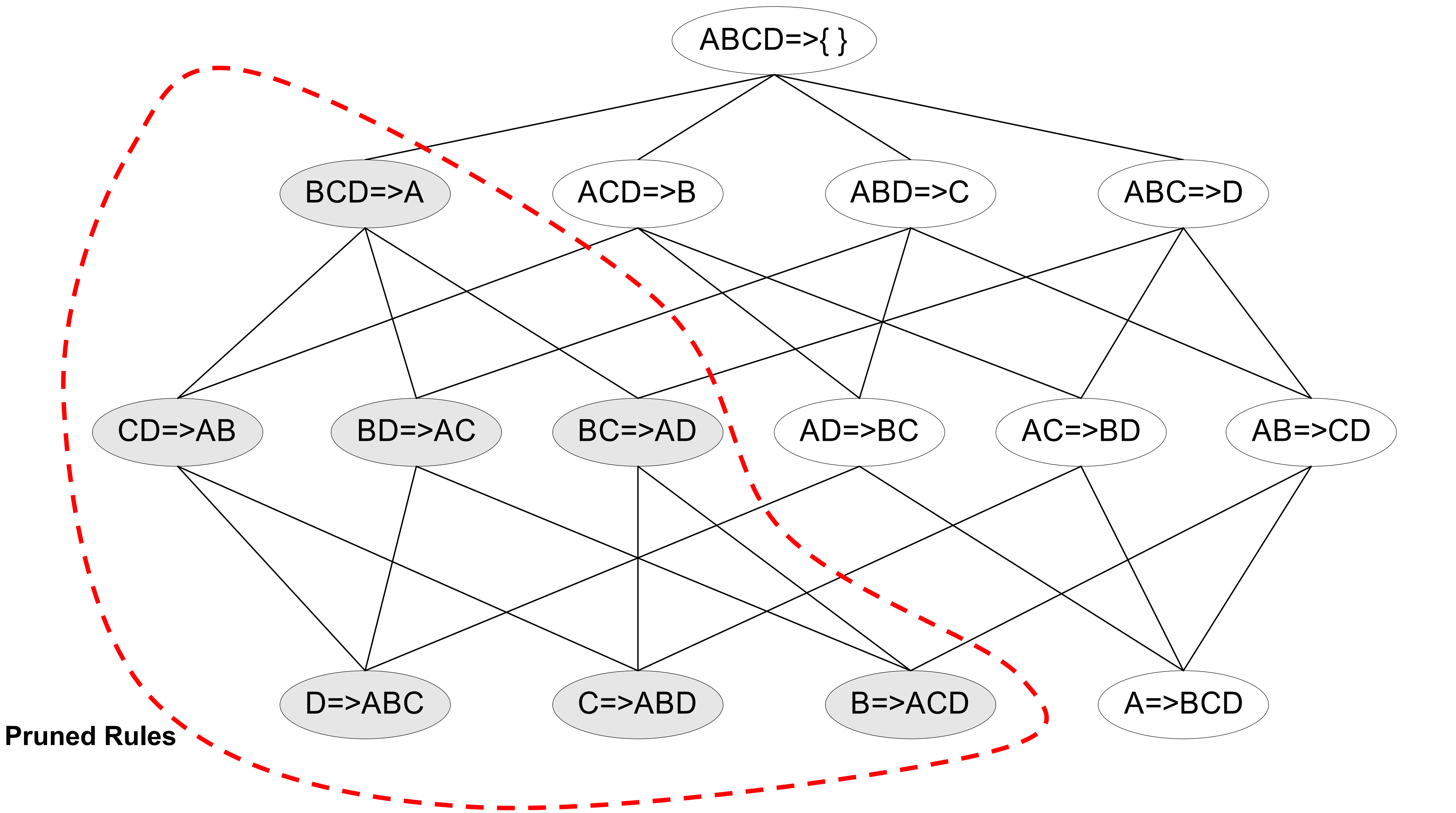
A →BCD, B →ACD, C →ABD, D →ABC,

AB →CD, AC → BD, AD → BC, BC →AD, BD →AC, CD →AB,

ABC →D, ABD →C, ACD →B, BCD →A,

If |itemset| = k, then there are $2^k - 2$ candidate association rules

Example: Generating the Rules

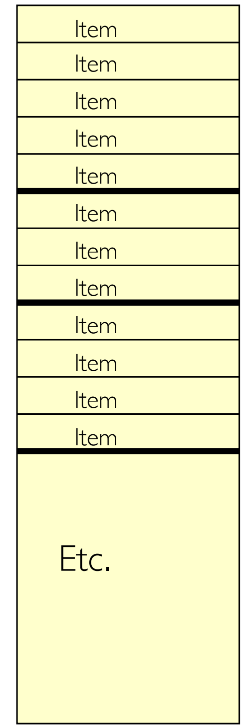


Step 1: Find Frequent Itemsets

Itemsets: Computation Model

Typically, data is kept in flat files rather than in a database system:

- Stored on disk
- Stored basket-by-basket
- Baskets are small but we have many baskets and many items
  - Expand baskets into pairs, triples, etc. as you read baskets
  - Use k nested loops to generate all sets of size k



Computation Model

The true cost of mining disk-resident data is usually the number of disk I/Os

In practice, association-rule algorithms read the data in passes – all baskets read in turn

We measure the cost by the number of passes an algorithm makes over the data

Main-Memory Bottleneck

For many frequent-itemset algorithms, main-memory is the critical resource

- As we read baskets, we need to count something, e.g., occurrences of pairs of items
- The number of different things we can count is limited by main memory
- Swapping counts in/out is a disaster (why?)

Latency numbers every programmer should know

```
L1 cache reference ......................... 0.5 ns
Branch mispredict .......................... 5 ns
L2 cache reference ......................... 7 ns
Mutex lock/unlock .......................... 25 ns
Main memory reference ...................... 100 ns
Compress 1K bytes with Zippy ............. 3,000 ns  =   3 µs
Send 2K bytes over 1 Gbps network ....... 20,000 ns  =  20 µs
SSD random read ........................ 150,000 ns  = 150 µs
Read 1 MB sequentially from memory ..... 250,000 ns  = 250 µs
Round trip within same datacenter ...... 500,000 ns  = 0.5 ms
Read 1 MB sequentially from SSD* ..... 1,000,000 ns  =   1 ms
Disk seek ........................... 10,000,000 ns  =  10 ms
Read 1 MB sequentially from disk .... 20,000,000 ns  =  20 ms
Send packet CA->Netherlands->CA .... 150,000,000 ns  = 150 ms
```

Finding Frequent Pairs

The hardest problem often turns out to be finding the frequent pairs of items {i1, i2}

> Why? Freq. pairs are common, freq. triples are rare
>
> Why? Probability of being frequent drops exponentially with size; number of sets grows more slowly with size

Let's first concentrate on pairs, then extend to larger sets

The approach:

> We always need to generate all the itemsets
>
> But we would only like to count (keep track) of those itemsets that in the end turn out to be frequent

Apriori

Apriori

Pass 1: Read baskets and count in main memory the occurrences of each individual item

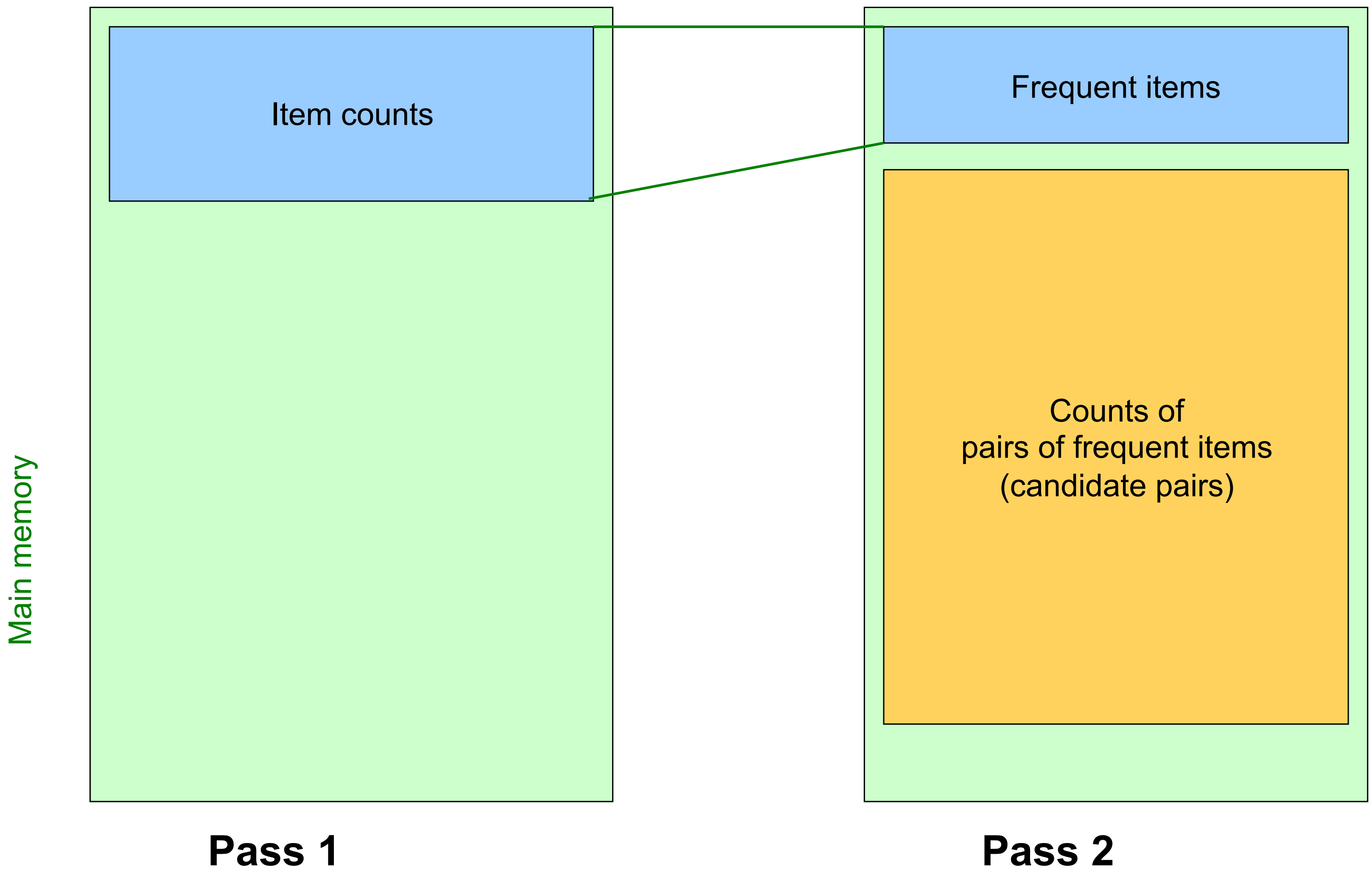
Requires only memory proportional to #items

Pass 2: Read baskets again and count in main memory only those pairs where both elements are frequent (from Pass 1)

Requires memory proportional to square of frequent items only (for counts)

Plus a list of the frequent items (so you know what must be counted)

Main Memory: Picture of Apriori



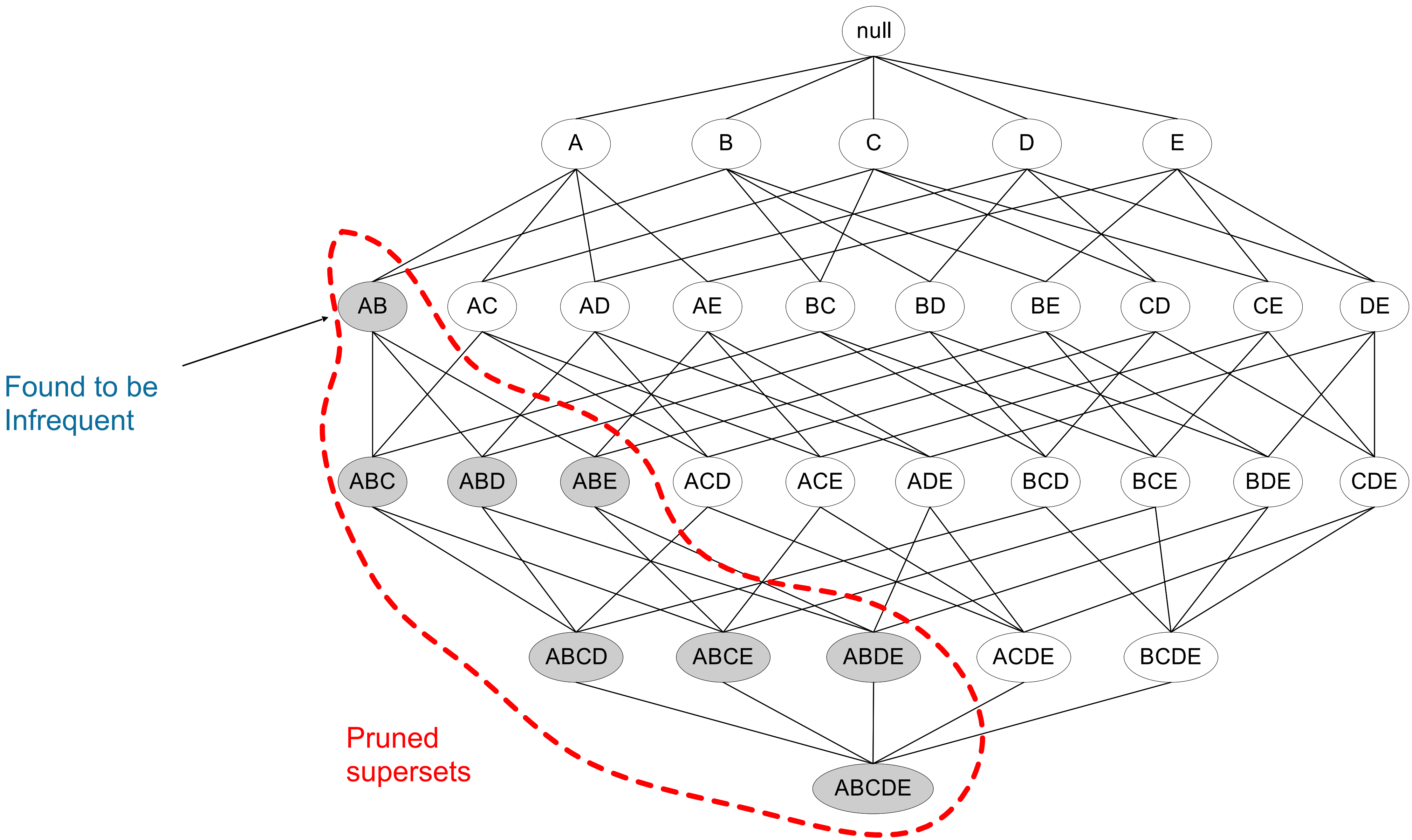
Recall: Downward Closure

The smaller the support threshold is, the larger the number of frequent item sets

Support monotonicity property:
if $J \subseteq I$, $\sup(J) \geq \sup(I)$

Downward closure property: every subset of a frequent itemset is also frequent

Apriori Principle



Illustrating Apriori

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

Items (1-itemsets)

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

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori

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

Items (1-itemsets)

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

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori

Items (1-itemsets)

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

Pairs (2-itemsets)

Itemset
{Bread,Milk}
{Bread, Beer }
{Bread,Diaper}
{Beer, Milk}
{Diaper, Milk}
{Beer,Diaper}

(No need to generate candidates involving Coke or Eggs)

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori

Items (1-itemsets)

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

Pairs (2-itemsets)

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

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori

Items (1-itemsets)

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

Pairs (2-itemsets)

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

Triplets (3-itemsets)

Itemset
{ Beer, Diaper, Milk}
{ Beer,Bread,Diaper}
{Bread, Diaper, Milk}
{ Beer, Bread, Milk}

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

Illustrating Apriori

Items (1-itemsets)

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

Pairs (2-itemsets)

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

Minimum Support = 3

If every subset is considered,

$^6C_1 + {}^6C_2 + {}^6C_3$

$6 + 15 + 20 = 41$

With support-based pruning,

$6 + 6 + 4 = 16$

$6 + 6 + 1 = 13$

Triplets (3-itemsets)

Itemset	Count
{ Beer, Diaper, Milk}	**2**
{ Beer,Bread, Diaper}	**2**
{Bread, Diaper, Milk}	**2**
{Beer, Bread, Milk}	**1**

Apriori Pseudocode

Algorithm *Apriori*(Transactions: $\mathcal{T}$, Minimum Support: $minsup$)
begin
  $k = 1$;
  $\mathcal{F}_1 = \{$ All Frequent 1-itemsets $\}$;
  while $\mathcal{F}_k$ is not empty **do begin**
    Generate $\mathcal{C}_{k+1}$ by joining itemset-pairs in $\mathcal{F}_k$;
    Prune itemsets from $\mathcal{C}_{k+1}$ that violate downward closure;
    Determine $\mathcal{F}_{k+1}$ by support counting on $(\mathcal{C}_{k+1}, \mathcal{T})$ and retaining
        itemsets from $\mathcal{C}_{k+1}$ with support at least $minsup$;
    $k = k + 1$;
  end;
  return$(\cup_{i=1}^{k} \mathcal{F}_i)$;
end

Improving Computation of Support

Naive Counting

Must match every candidate itemset against every transaction, which is an expensive operation

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

Itemset
{ Beer, Diaper, Milk}
{ Beer,Bread,Diaper}
{Bread, Diaper, Milk}
{ Beer, Bread, Milk}

Naive Counting

Naïve counting:

For each candidate $I_i \in C_{k+1}$

For each transaction T_j in T

Check whether I_i appears in T_j

Limitation

Inefficient if both $|C_{k+1}|$ and $|T|$ are large

Support Counting with a Data Structure

A Better Approach

Organize the candidate patterns in C_{k+1} in a data structure

Use the data structure to accelerate counting

Each transaction in T_i examined against the subset of candidates in C_{k+1} that might contain T_i

Hash-based Approach

Naïve counting:
For each $I_i \in C_{k+1}$
  For all $T_j \in T$
    If $I_i \subseteq T_j$
      Add to $sup(I_i)$

Hashed counting:
For each $T_j \in T$
  For $I_i \in hashbucket(T_j, C_{k+1})$
    If $I_i \subseteq T_j$
      Add to $sup(I_i)$

Hash-based Approach

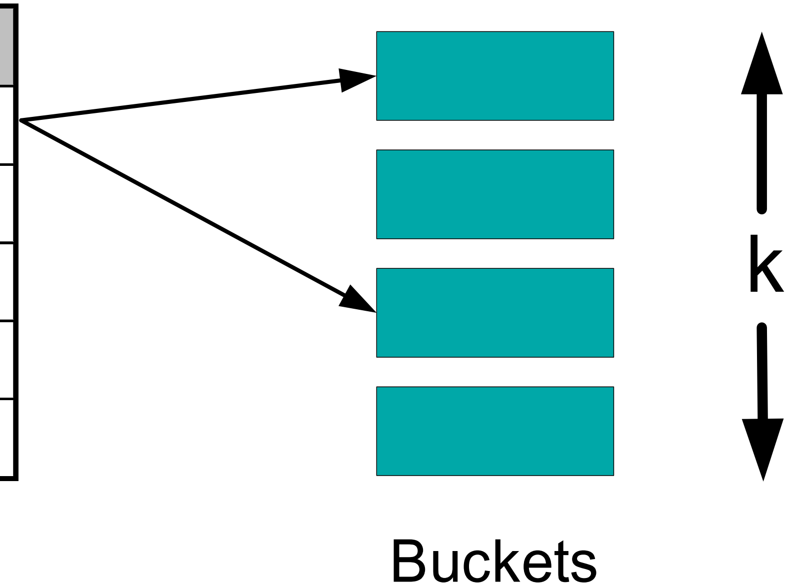
Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets

Transactions

Hash Structure

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

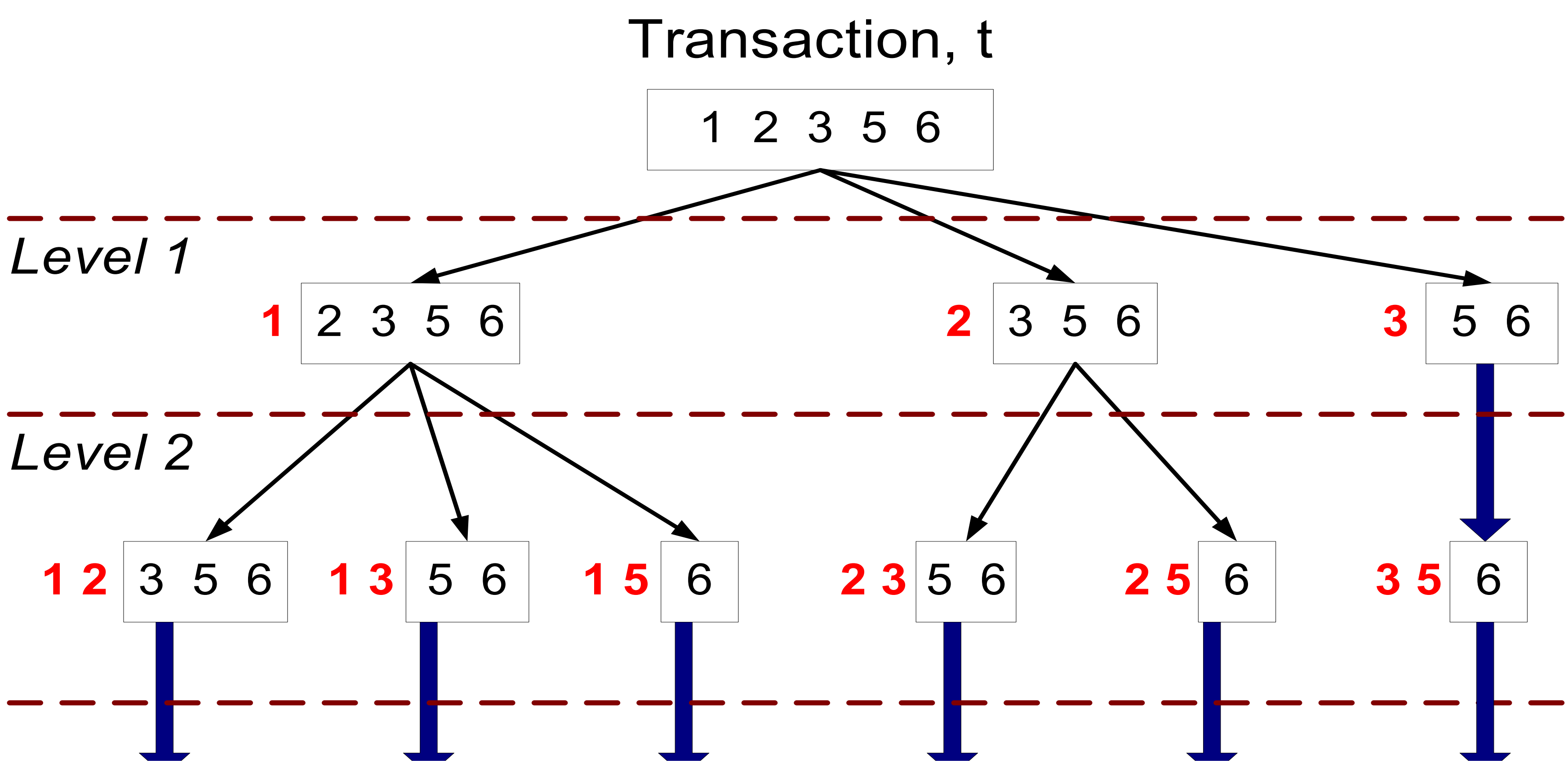


Support Counting: Example

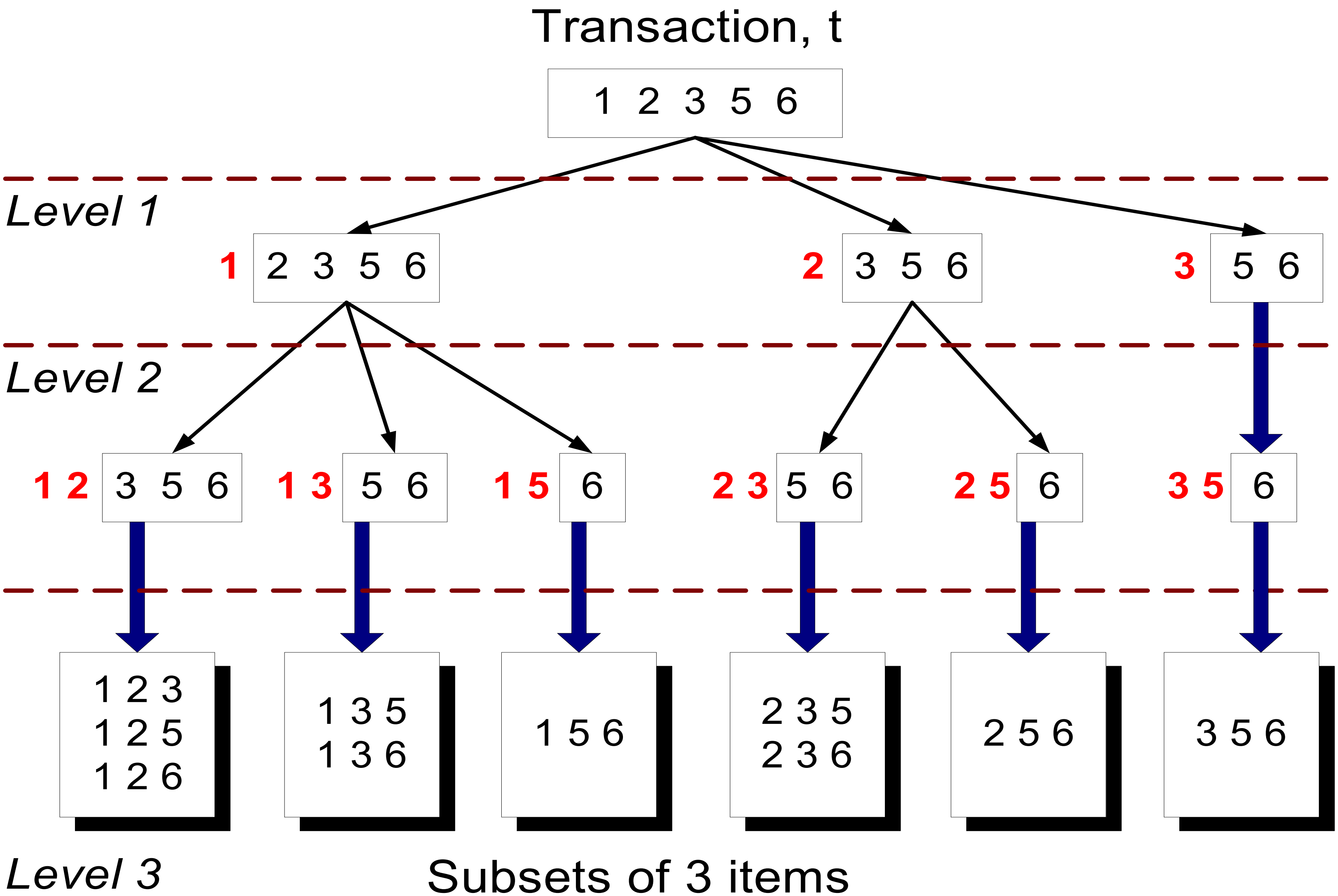
Suppose you have 15 candidate itemsets of length 3:

{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}

How many of these itemsets are supported by transaction (1,2,3,5,6)?



Prefix tree enumerating all 3-itemsets in transaction t



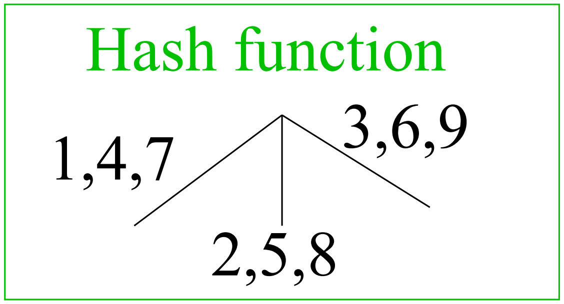
Hash tree for itemsets in C_{k+1}

A tree with fixed degree r

Each itemset in C_{k+1} is stored in a leaf node

All internal nodes use a hash function to map items to one of the r branches (can be the same for all internal nodes)

All leaf nodes contain a lexicographically sorted list of up to max_leaf_size itemsets



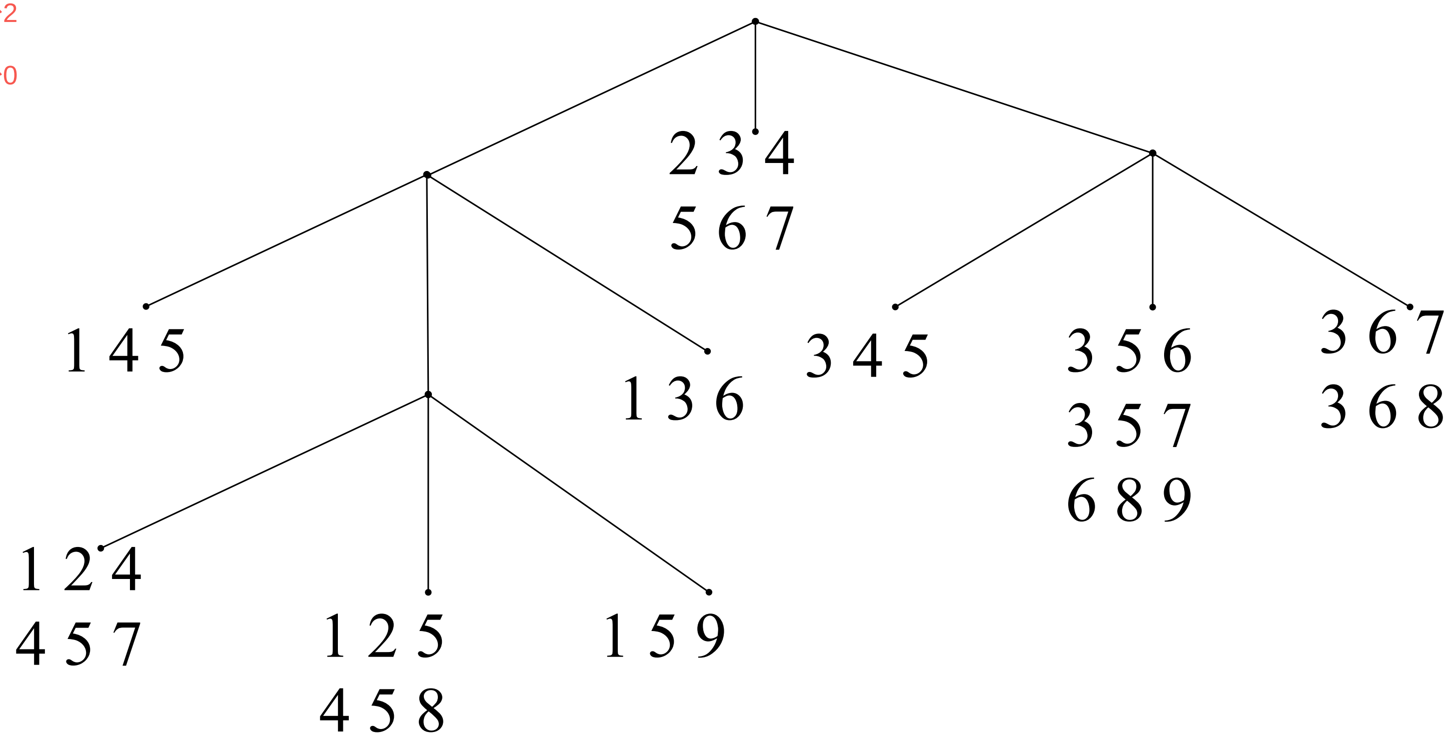
有道是，一層一層的hash！！！！

Example

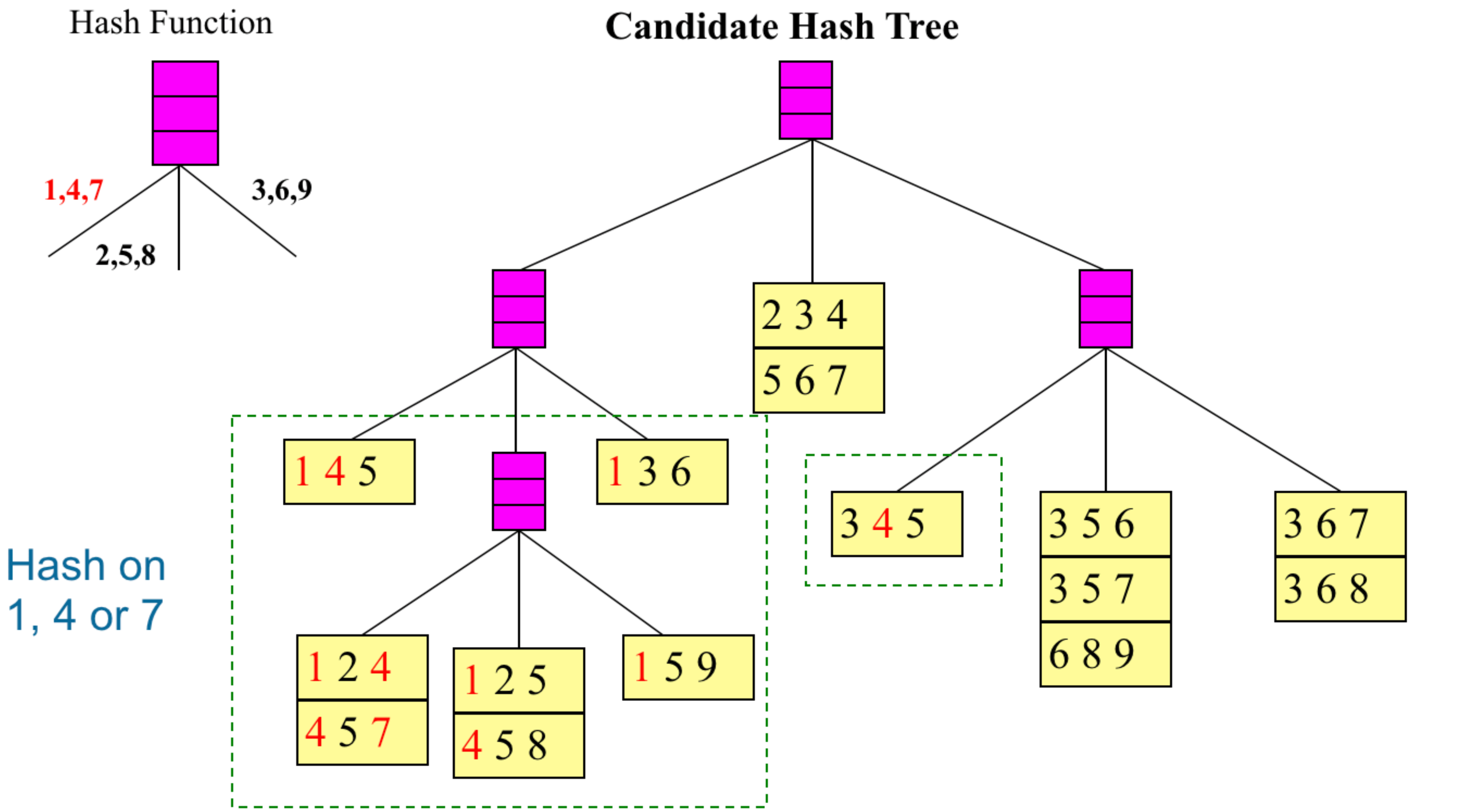
1，4，7除以3，都等于1

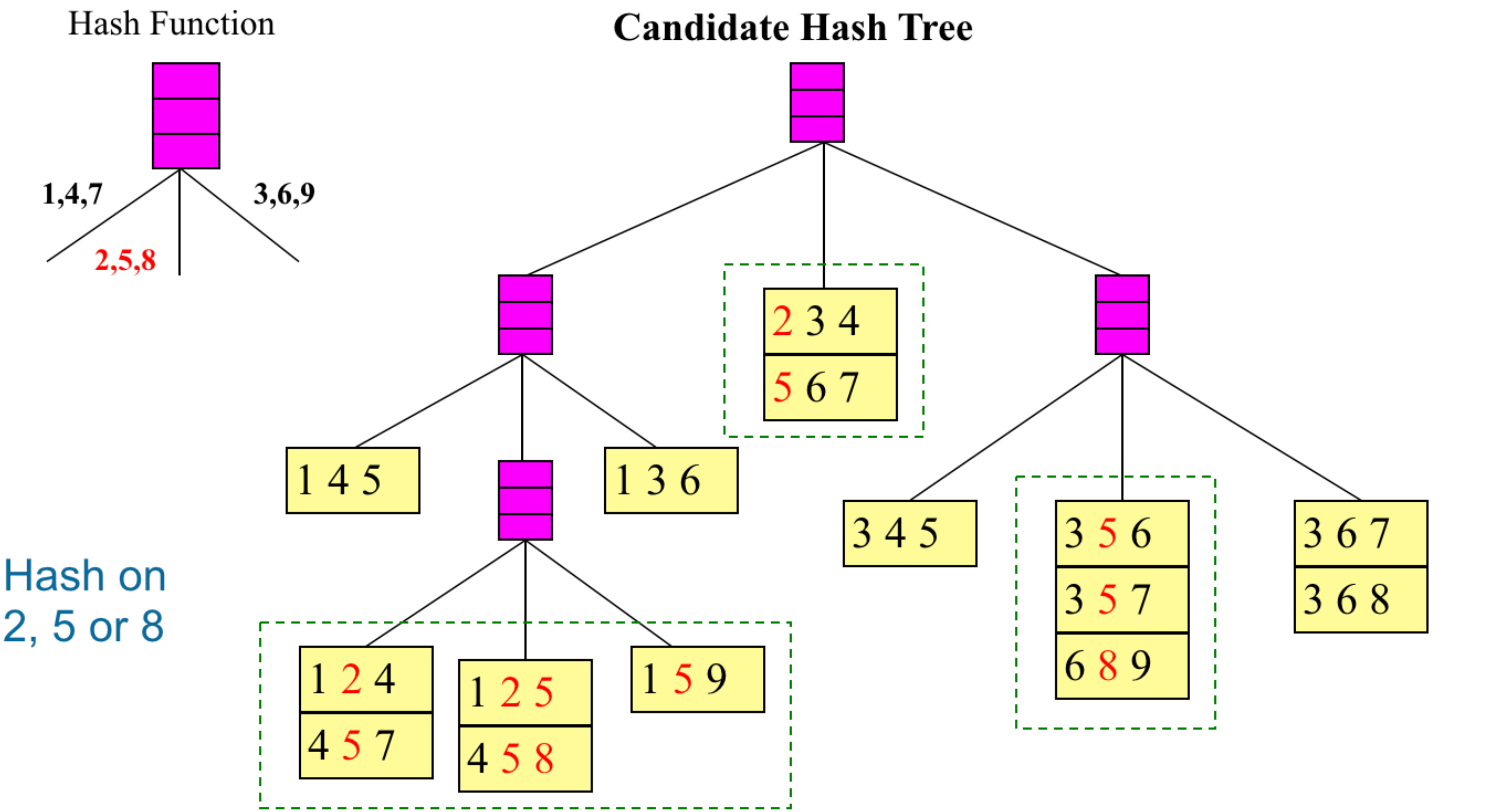
2，5，8除以3，都等於2

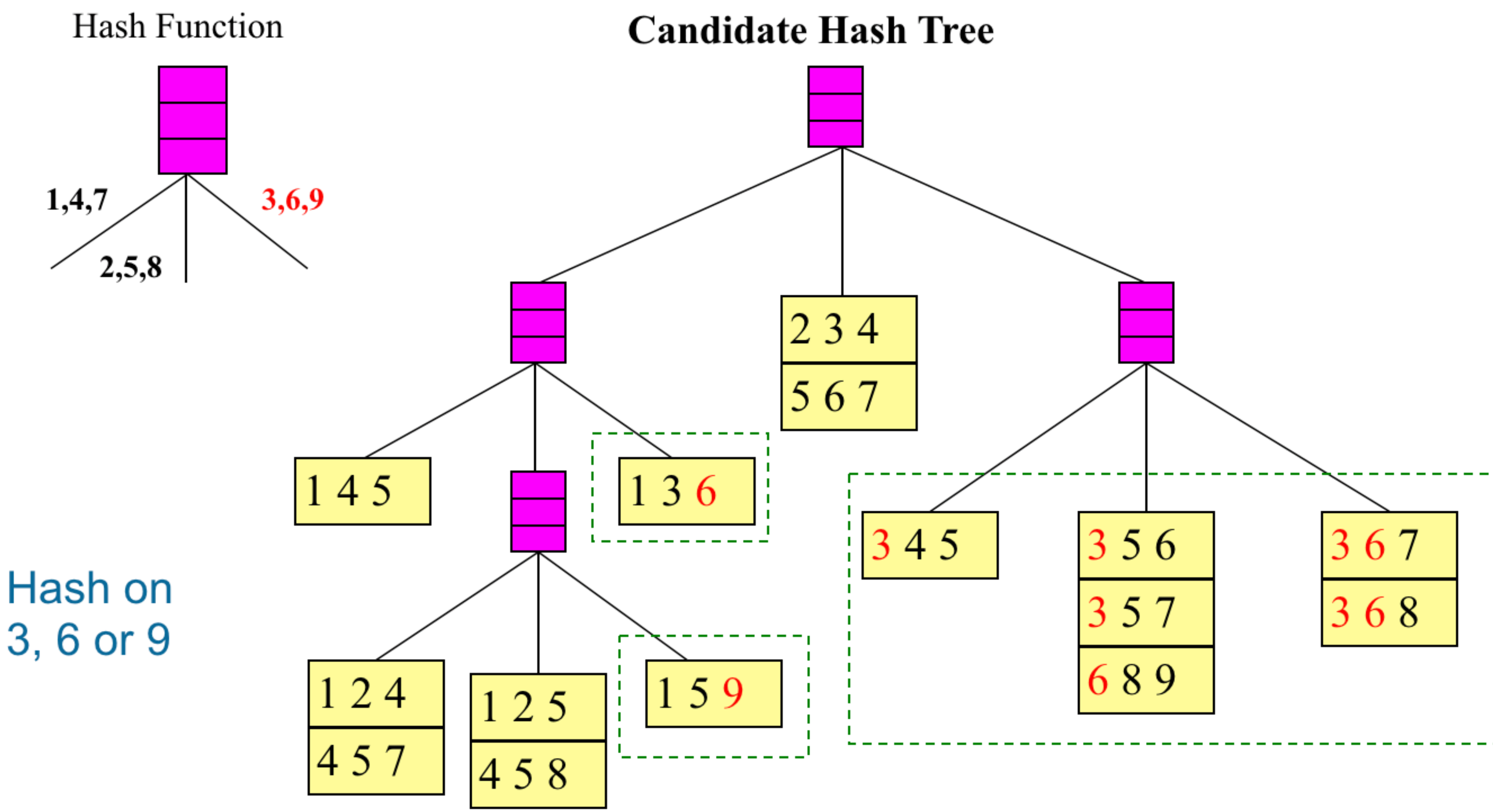
3，6，9除以3，都等於0

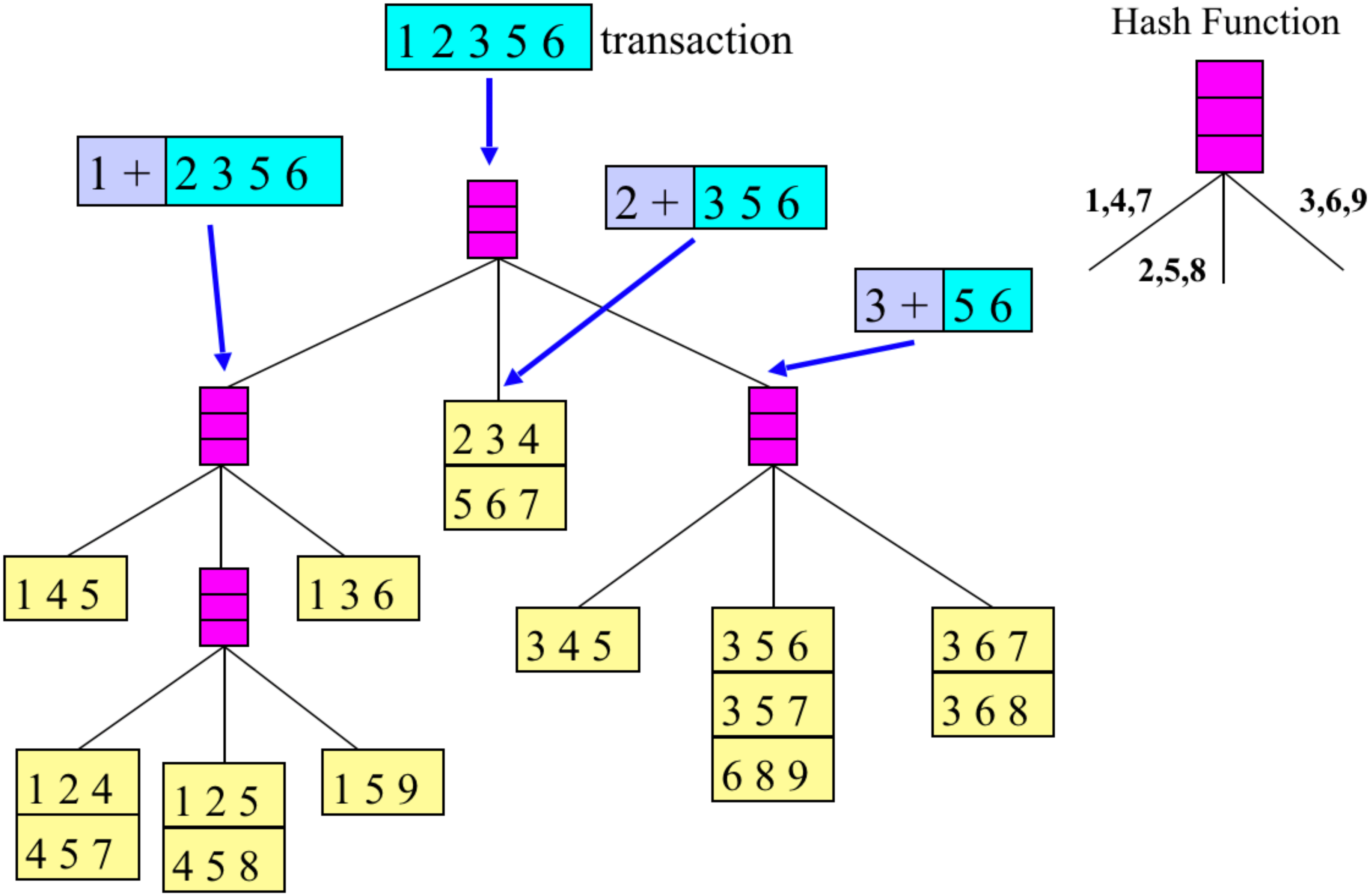


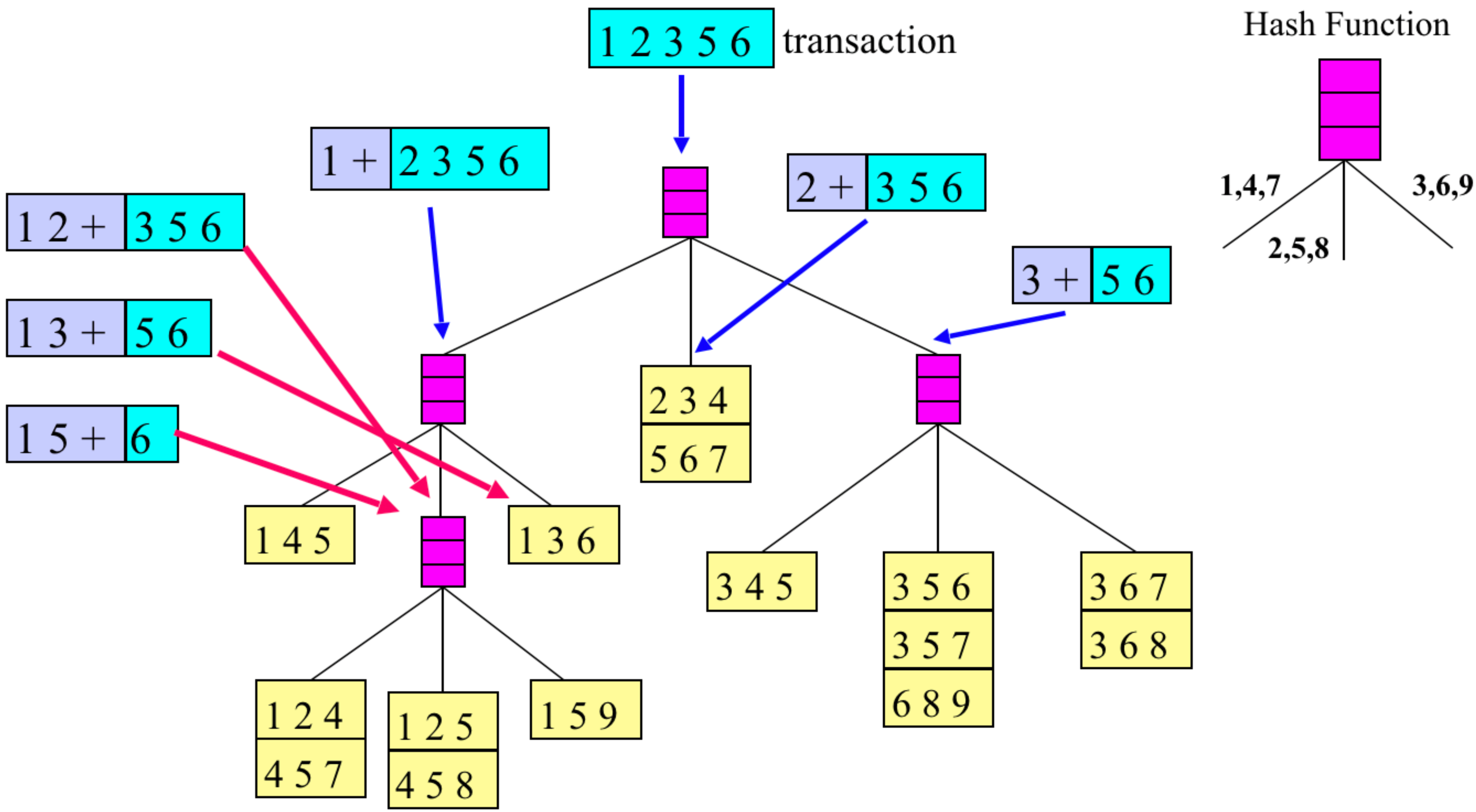
Candidate itemsets: {1 4 5}, {1 2 4}, {4 5 7}, {1 2 5}, {4 5 8}, {1 5 9}, {1 3 6}, {2 3 4}, {5 6 7}, {3 4 5}, {3 5 6}, {3 5 7}, {6 8 9}, {3 6 7}, {3 6 8}

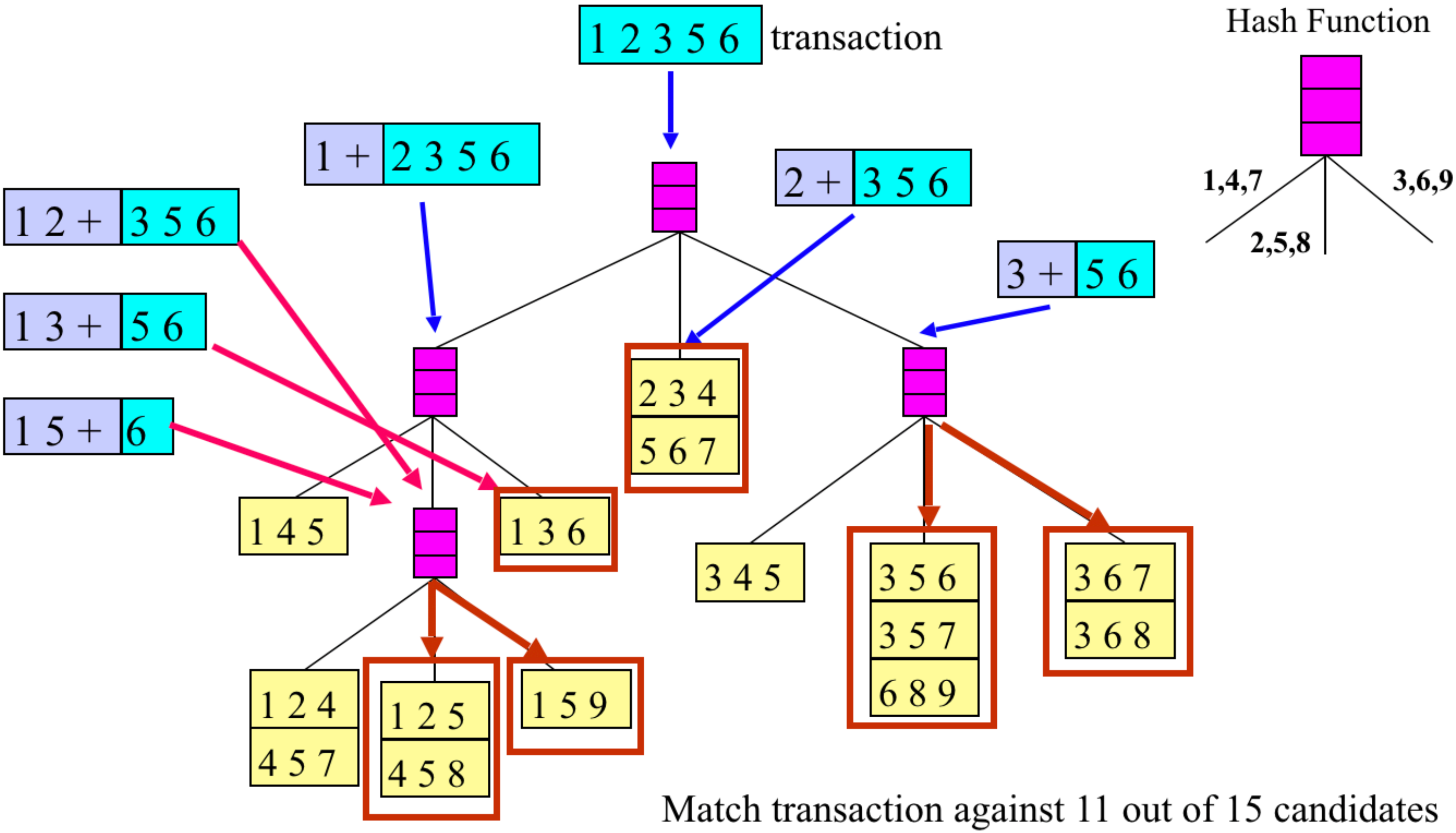

Hash Function
Candidate Hash Tree
1,4,7
3,6,9
2,5,8
2 3 4
5 6 7
1 4 5
1 3 6
3 4 5
3 5 6
3 5 7
6 8 9
3 6 7
3 6 8
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9
Hash on
1, 4 or 7


Hash Function
1,4,7
2,5,8
3,6,9
Candidate Hash Tree
2 3 4
5 6 7
1 4 5
1 3 6
3 4 5
3 5 6
3 5 7
6 8 9
3 6 7
3 6 8
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9
Hash on
2, 5 or 8


Hash Function
Candidate Hash Tree
1,4,7
2,5,8
3,6,9
2 3 4
5 6 7
1 4 5
1 3 6
3 4 5
3 5 6
3 5 7
6 8 9
3 6 7
3 6 8
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9
Hash on
3, 6 or 9


1 2 3 5 6
transaction
Hash Function
1,4,7
2,5,8
3,6,9
1 + 2 3 5 6
2 + 3 5 6
3 + 5 6
2 3 4
5 6 7
1 4 5
1 3 6
3 4 5
3 5 6
3 5 7
6 8 9
3 6 7
3 6 8
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9


1 2 3 5 6
transaction
Hash Function
1,4,7
2,5,8
3,6,9
1 + 2 3 5 6
2 + 3 5 6
3 + 5 6
1 2 + 3 5 6
1 3 + 5 6
1 5 + 6
2 3 4
5 6 7
1 4 5
1 3 6
3 4 5
3 5 6
3 5 7
6 8 9
3 6 7
3 6 8
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9



1 2 3 5 6 transaction
Hash Function
1,4,7
2,5,8
3,6,9
1 + 2 3 5 6
2 + 3 5 6
3 + 5 6
1 2 + 3 5 6
1 3 + 5 6
1 5 + 6
2 3 4
5 6 7
1 4 5
1 3 6
3 4 5
3 5 6
3 5 7
6 8 9
3 6 7
3 6 8
1 2 4
4 5 7
1 2 5
4 5 8
1 5 9
Match transaction against 11 out of 15 candidates