Association rules mining

Mining Massive Datasets

Carlos Castillo

Topic 06

Universitat Pompeu Fabra Barcelona

Sources

- Data Mining, The Textbook (2015) by Charu Aggarwal (Chapters 4, 5) – slides by Lijun Zhang
- Mining of Massive Datasets 2nd edition (2014) by Leskovec et al. (Chapter 6) - slides
- Data Mining Concepts and Techniques, 3rd edition (2011) by Han et al. (Chapter 6)
- Introduction to Data Mining 2nd edition (2019) by Tan et al. (Chapters 5, 6) – slides ch5, slides ch6

Association rule

- Let X, Y be two itemsets; the rule X⇒Y is an **association rule** of minimum support **minsup** and minimum confidence **minconf** if:

$$\text{sup}(X \Rightarrow Y) \geq \text{minsup}$$

and

$$\text{conf}(X \Rightarrow Y) \geq \text{minconf}$$

Association rule mining framework

- In the first phase, all the frequent itemsets are generated at the minimum support of minsup
  - The most difficult (computationally expensive) step
- In the second phase, the association rules are generated from the frequent itemsets at the minimum confidence level of minconf
  - Relatively straightforward

A straightforward implementation of the second phase

For each frequent itemset I // sup(I) ≥ minsup

For each possible partition X, Y = I – X

Check if conf(X⇒Y) ≥ minconf

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

Confidence monotonicity property

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

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

Prove this, remember:

$$\mathrm{conf}(X \Rightarrow Y) = \frac{\sup(X \cup Y)}{\sup(X)}$$

Brute-force itemset mining algorithms

Naïve approach

- Generate all candidate itemsets ($2^{|U|}$ of them)
  - Not practical, U=1000 ⇒ more than 10^{300} itemsets
- Calculate sup(I) for every itemset
- Key observation
  - If no k-itemsets are frequent
  - No (k+1)-itemsets are frequent

Improved approach

Start with k=1

- Generate all k-itemsets
- Determine sup(I)
- If no k-itemset has sup(I) ≥ minsup, stop
- Otherwise, k ← k+1 and repeat

Improved approach is a significant improvement

- Let l be the final value of k

$$\sum_{i=1}^{l} \binom{|U|}{i} \ll 2^{|U|}$$

- For $|U| = 1000$, $l=10$, this is $\simeq 10^{23}$

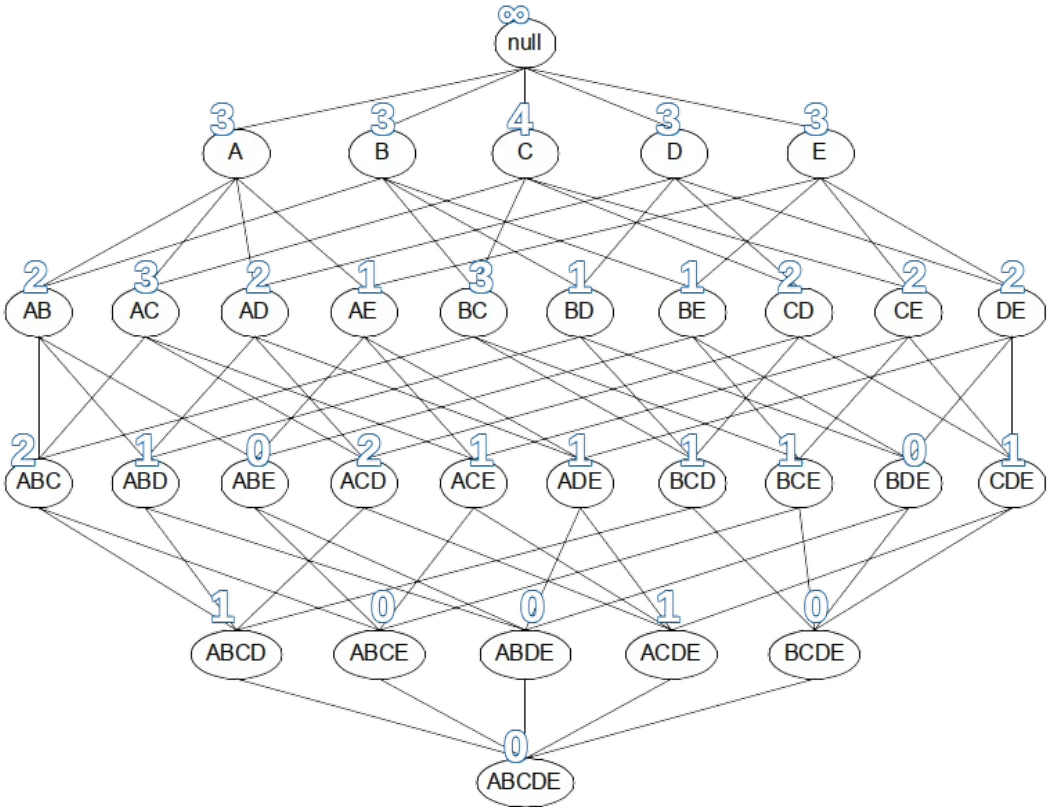
Further improvements to brute-force method

1. Reducing the size of the explored search space (lattice) by pruning candidate itemsets (lattice nodes) using tricks, such as the downward closure property
2. Counting the support of each candidate more efficiently by pruning transactions that are known to be irrelevant for counting a candidate itemset
3. Using compact data structures to represent either candidates or transaction databases that support efficient counting

The Apriori Algorithm

Apriori algorithm principle

- **Downward closure property**: every subset of a frequent itemset is also frequent
- Conversely, if an itemset has a subset that is not frequent, the itemset cannot be frequent
- **What are subsets in the lattice?**



Example

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	X
Milk	4	
Beer	3	
Diaper	4	
Eggs	1	X

Minimum Support = 3

Example

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~~	X
Milk	4	
Beer	3	
Diaper	4	
~~Eggs~~	~~1~~	X

Minimum Support = 3

Example (cont.)

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~~	X
Milk	4	
Beer	3	
Diaper	4	
~~Eggs~~	~~1~~	X

Pairs (2-itemsets)

Item	Count	
{Bread, Milk}	3	
{Beer, Bread}	2	X
{Bread, Diaper}	3	
{Beer, Milk}	2	X
{Diaper, Milk}	3	
{Beer, Diaper}	3	

Minimum Support = 3

Example (cont.)

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~~	X
Milk	4	
Beer	3	
Diaper	4	
~~Eggs~~	~~1~~	X

Pairs (2-itemsets)

Item	Count	
{Bread, Milk}	3	
~~{Beer, Bread}~~	~~2~~	X
{Bread, Diaper}	3	
~~{Beer, Milk}~~	~~2~~	X
{Diaper, Milk}	3	
{Beer, Diaper}	3	

Minimum Support = 3

Example (cont.)

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~~	X
Milk	4	
Beer	3	
Diaper	4	
~~Eggs~~	~~1~~	X

Pairs (2-itemsets)

Item	Count	
{Bread, Milk}	3	
~~{Beer, Bread}~~	~~2~~	X
{Bread, Diaper}	3	
~~{Beer, Milk}~~	~~2~~	X
{Diaper, Milk}	3	
{Beer, Diaper}	3	

Triplets (3-itemsets)

Item	Count	
{Bread, Diaper, Milk}	2	X
{Beer, Bread, Diaper}	2	X
{Bread, Diaper, Milk}	2	X
{Beer, Bread, Milk}	1	X

Minimum Support = 3

Example (cont.)

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~~	X
Milk	4	
Beer	3	
Diaper	4	
~~Eggs~~	~~1~~	X

Pairs (2-itemsets)

Item	Count	
{Bread, Milk}	3	
~~{Beer, Bread}~~	~~2~~	X
{Bread, Diaper}	3	
~~{Beer, Milk}~~	~~2~~	X
{Diaper, Milk}	3	
{Beer, Diaper}	3	

Triplets (3-itemsets)

Item	Count	
~~{Bread, Diaper, Milk}~~	~~2~~	X
~~{Beer, Bread, Diaper}~~	~~2~~	X
~~{Bread, Diaper, Milk}~~	~~2~~	X
~~{Beer, Bread, Milk}~~	~~1~~	X

Minimum Support = 3, **found 8 frequent itemsets**

Pseudocode of Apriori

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

(1) Generation
(2) Pruning
(3) Support counting

Try it!

Use the Apriori algorithm to obtain all rules of the form {a,b}→{c} having
minimum support = 2
 and
confidence >= 50%

Note: to generate only rules of the form {a,b}→{c}, use only the itemsets of size 3

TID	items
T1	I1, I2 , I5
T2	I2,I4
T3	I2,I3
T4	I1,I2,I4
T5	I1,I3
T6	I2,I3
T7	I1,I3
T8	I1,I2,I3,I5
T9	I1,I2,I3

Speeding up candidate generation

Level-wise pruning trick

- Let F_k be the set of frequent k-itemsets
- Let C_{k+1} be the set of (k+1)-candidates
- $I \in C_{k+1}$ is frequent only if all the k-subsets of I are frequent
- Pruning
  - Generate all the k-subsets of I
  - If any one of them does not belong to F_k, then remove I

Candidates generation

- A Naïve Approach
  - Check all the possible combinations of frequent itemsets
- An Example of the Naïve Approach
  - itemsets: {abc} {bcd} {abd} {cde}
  - {abc} + {bcd} = {abcd}
  - {bcd} + {abd} = {abcd}
  - {abd} + {cde} = {abcde}
  - ….

Candidates generation (cont.)

- Introduction of Ordering
  - Items in U have a lexicographic ordering
  - Itemsets can be order as strings
- A Better Approach
  - Order the frequent k-itemsets
  - Merge two itemset if the first k-1 items of them are the same

Candidates generation (cont.)

- Example
  - k-itemsets: {abc} {abd} {bcd}
    - {abc} + {abd} = {abcd}
- k-itemsets: {abc} {acd} {bcd}
  - No (k+1) -candidates
- Early stop is possible
  - Do not need to check {abc} +{bcd} after checking {abc} + {acd}
- Do we miss {abcd}?
  - No, due to the Downward Closure Property

Improving computation of support

Naïve support 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

Data structured for support counting based on hashing

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)

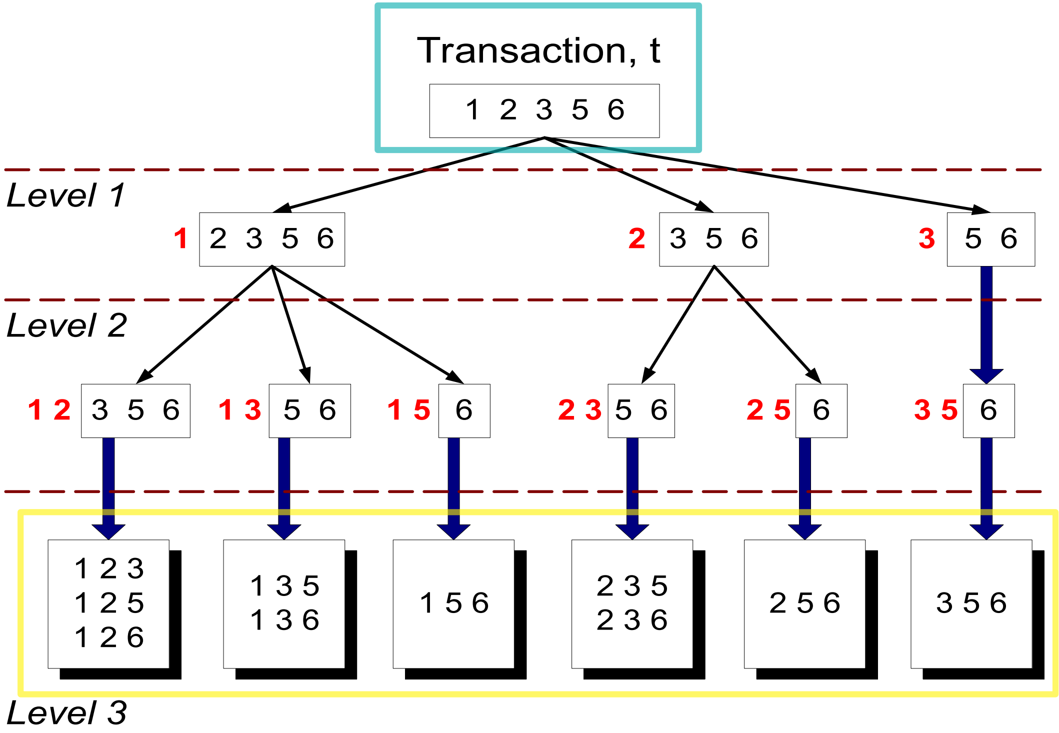
Which candidates are relevant?

Imagine 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}
```

Now, suppose we look for this transaction:

```
{1 2 3 5 6}
```



Here we depict only the candidates that appear in the transaction (10 out of 15)

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

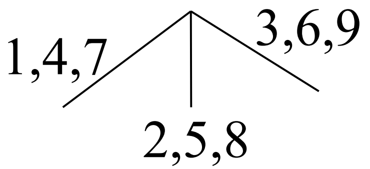
Example hash tree

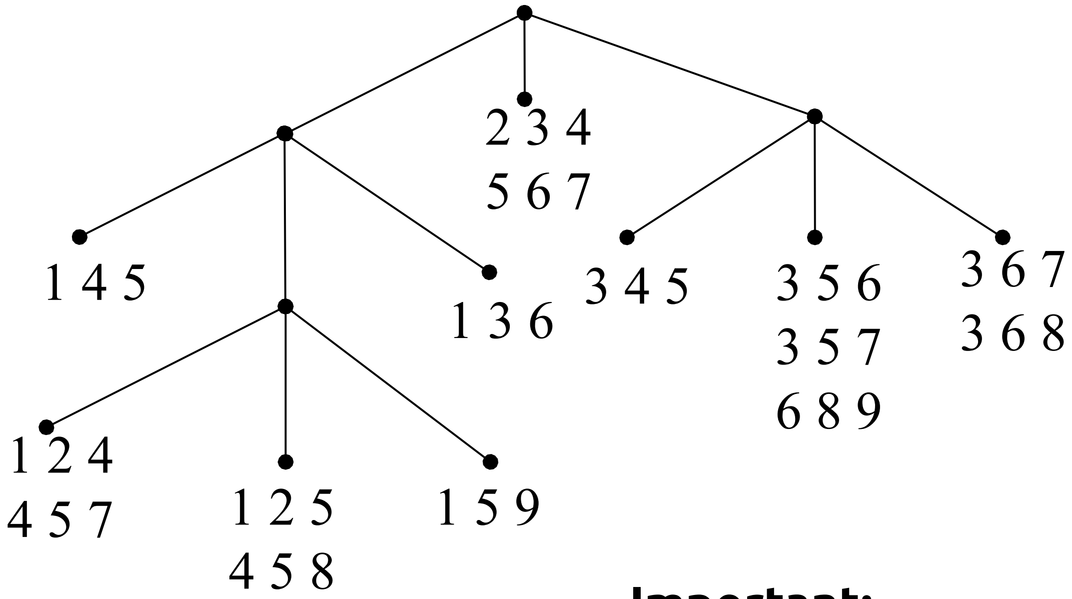
`r=3  max_leaf_size=3`

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

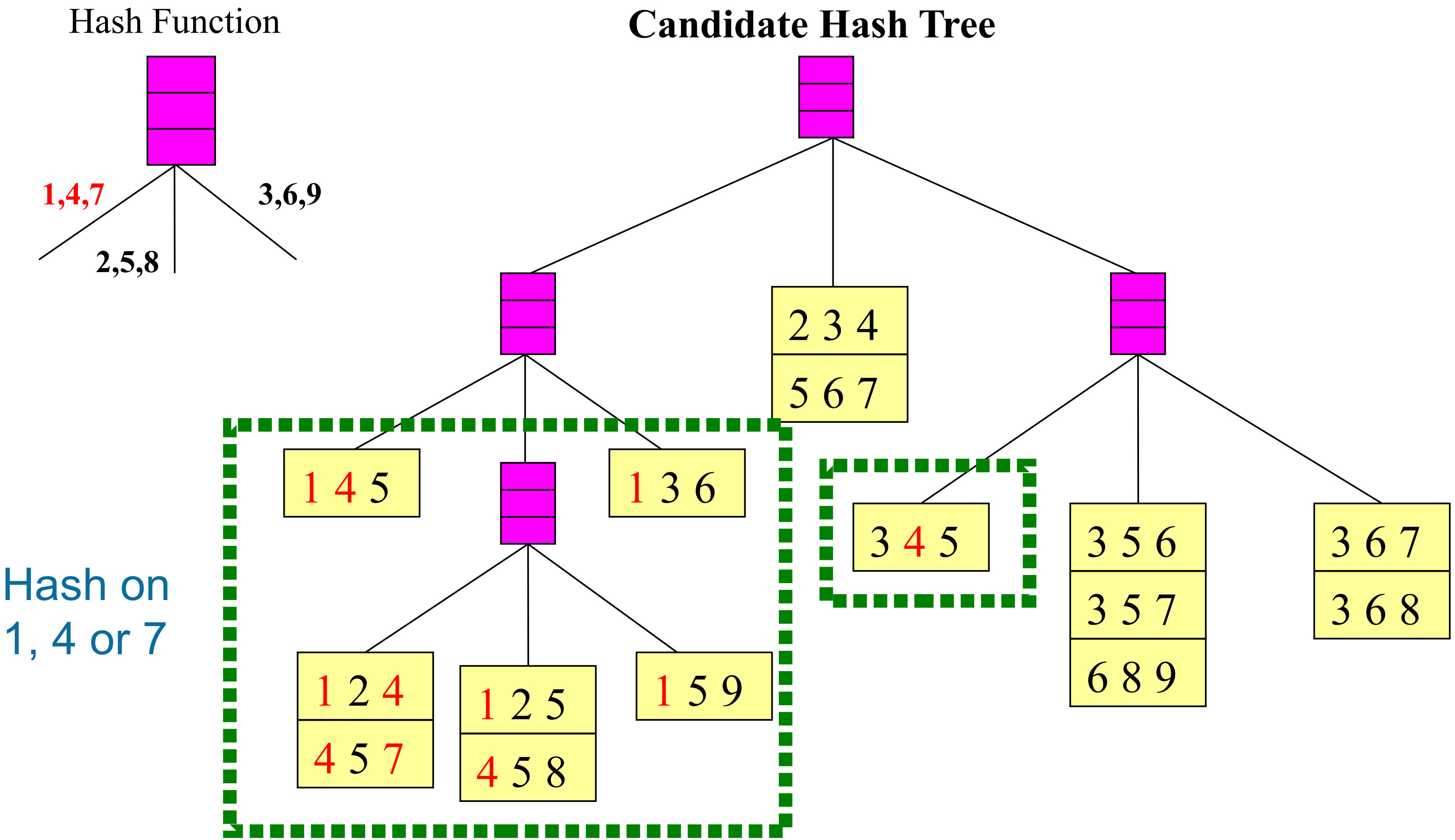




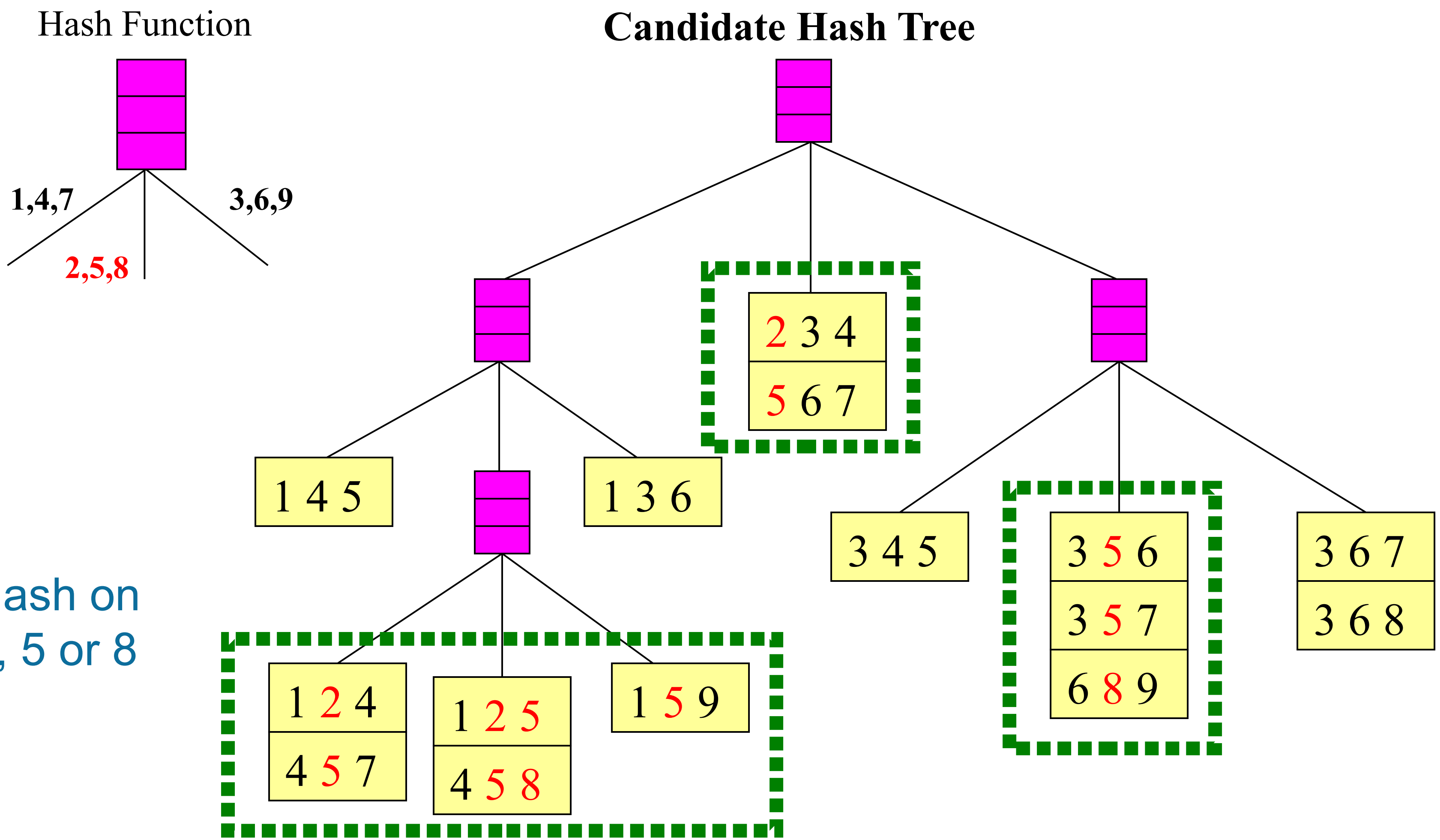
Important: itemsets are sorted!

This example from: Introduction to Data Mining 2nd edition (2019) by Tan et al. (Chapter 5) – slides ch5

Example hash tree (cont.)

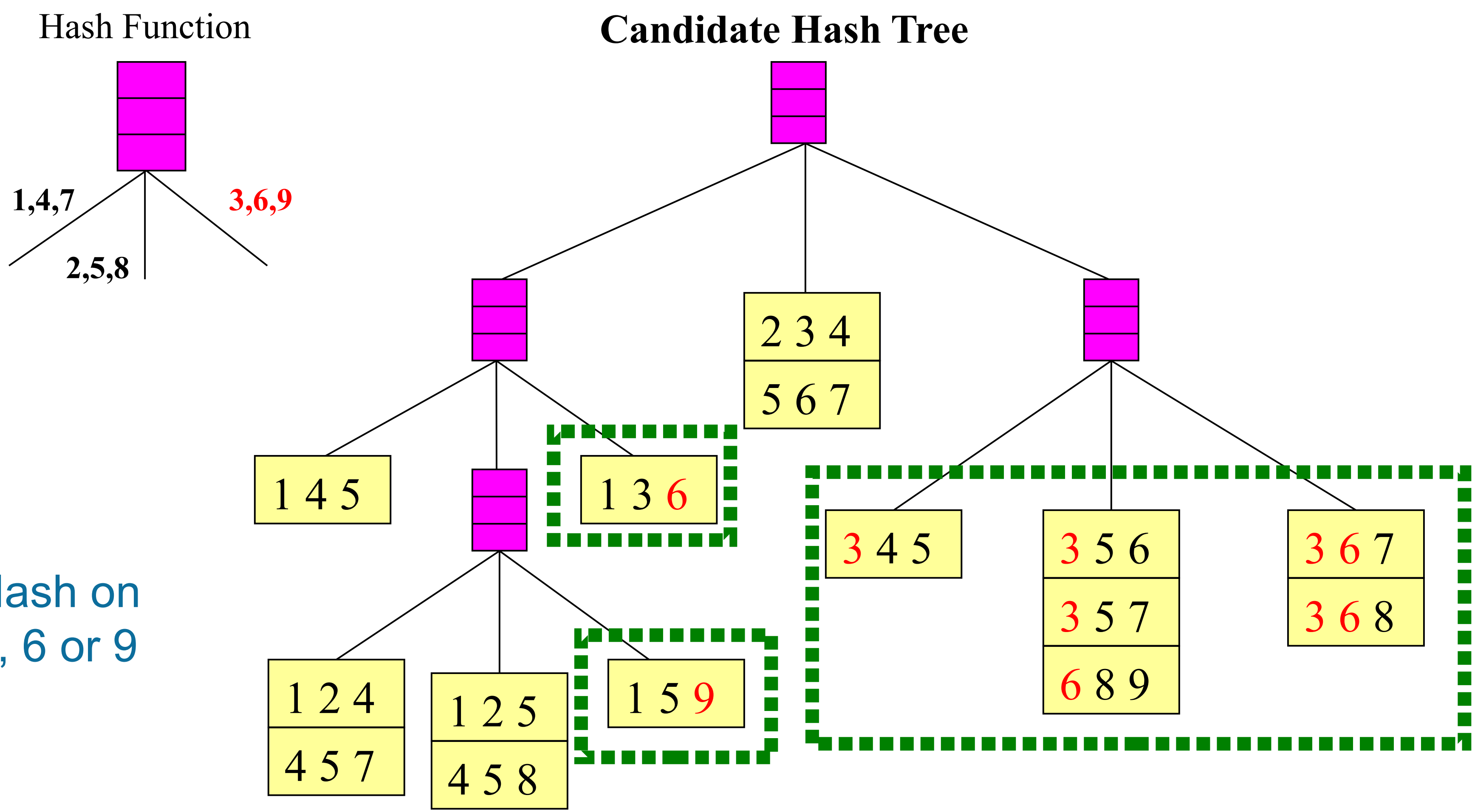


Example hash tree (cont.)



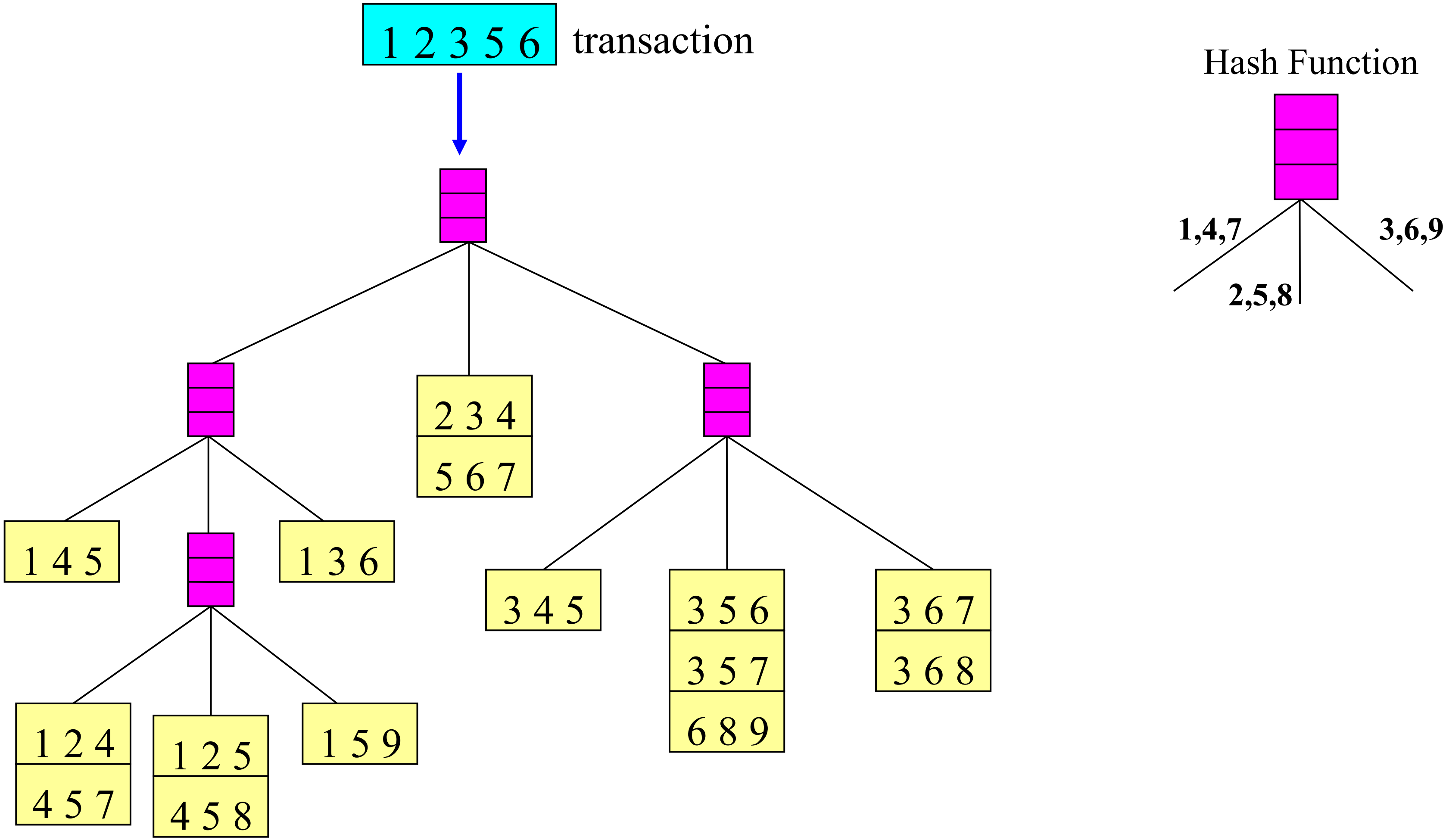
Hash on
2, 5 or 8

Example hash tree (cont.)

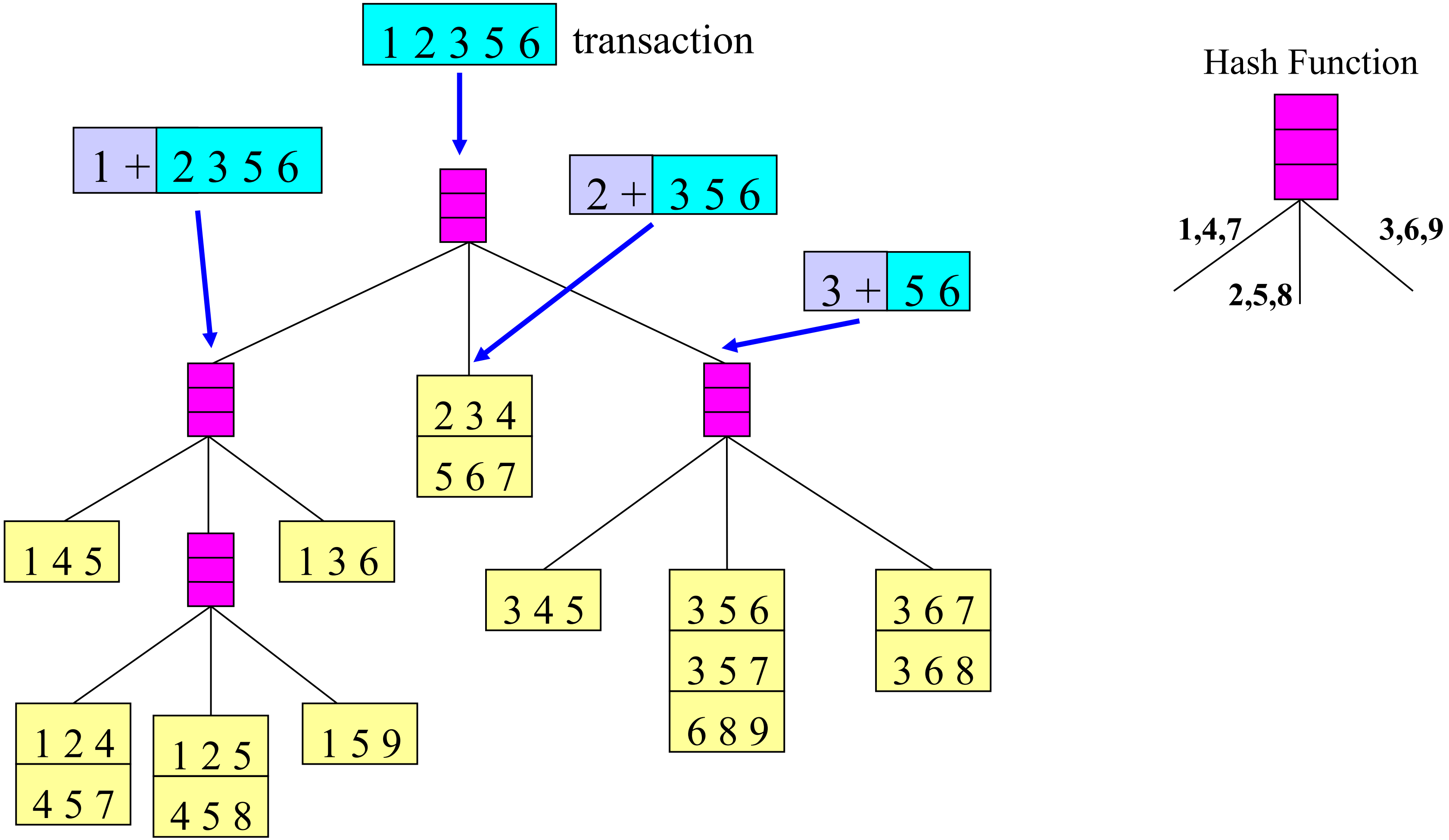


Hash on
3, 6 or 9

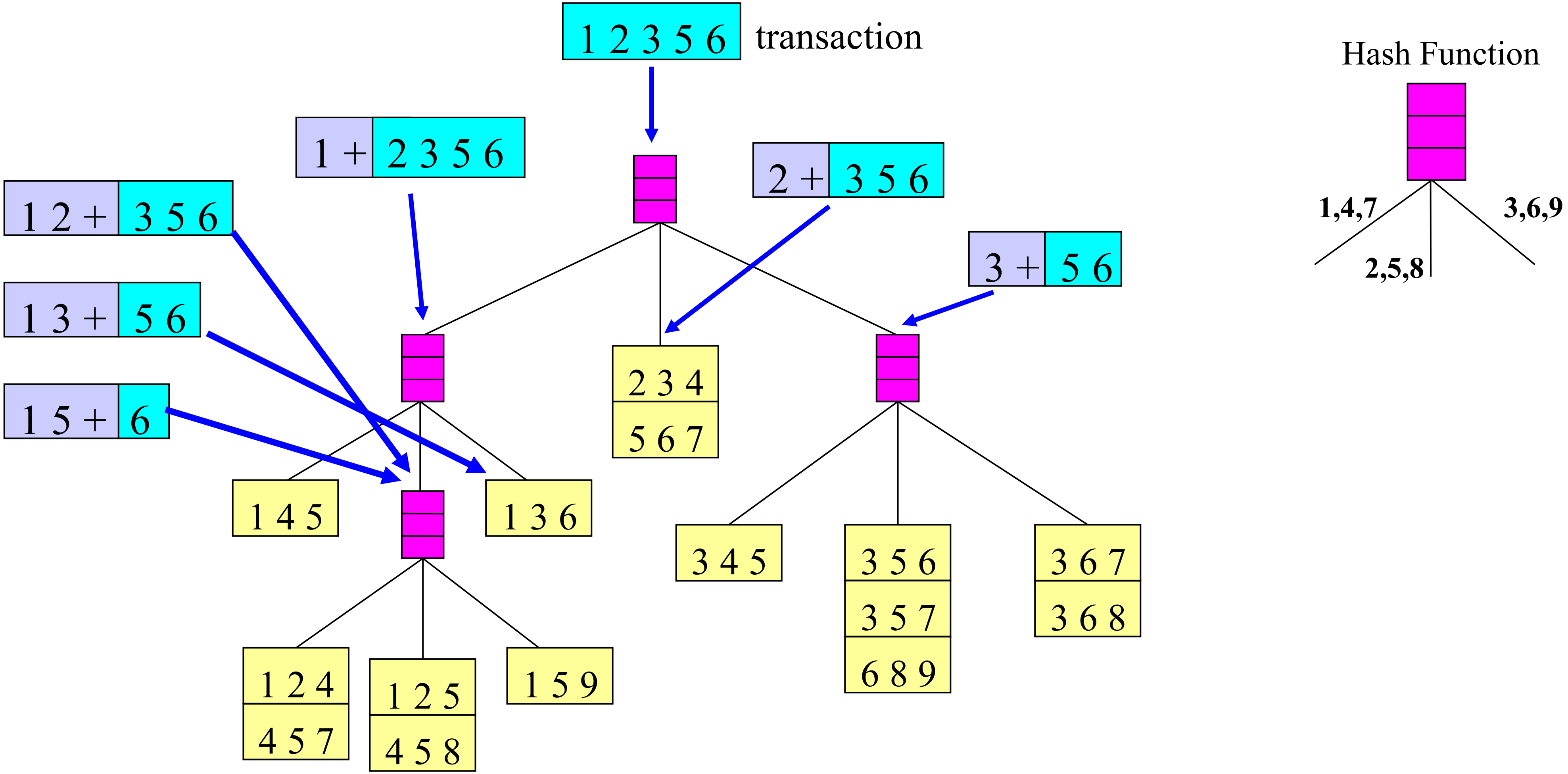
Checking which candidates might be in a transaction



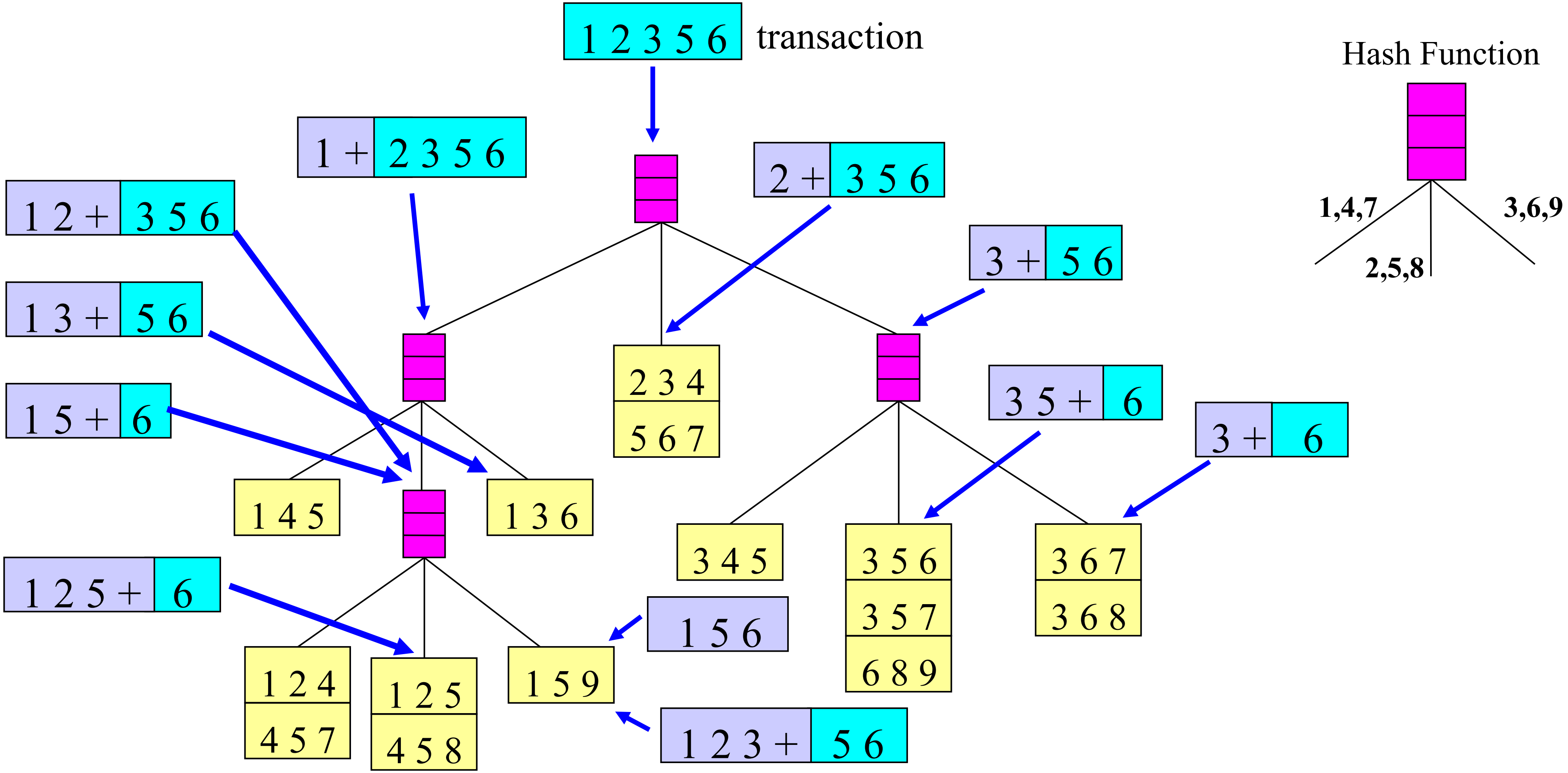
Checking which candidates might be in a transaction



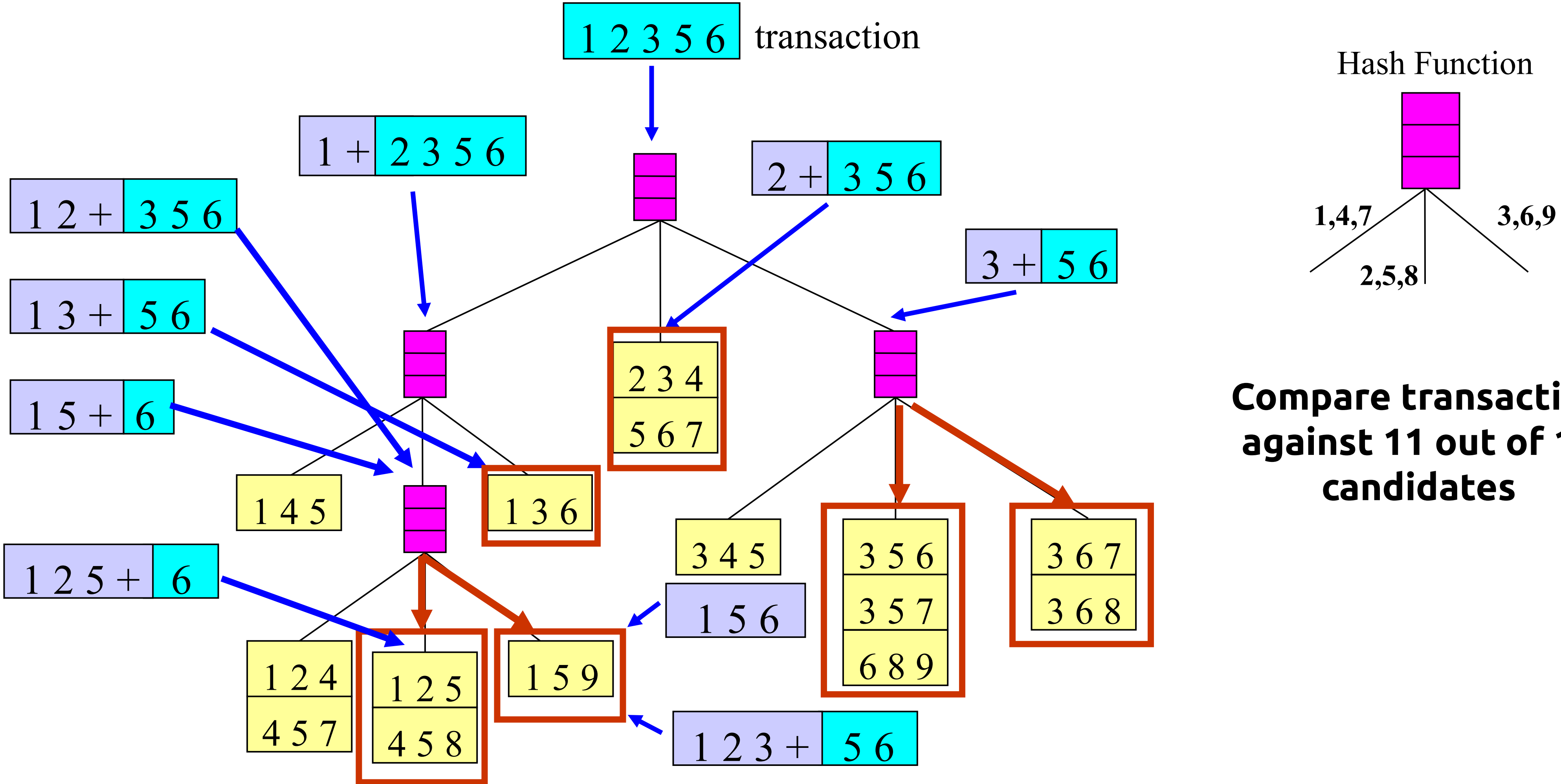
Checking which candidates might be in a transaction



Checking which candidates might be in a transaction

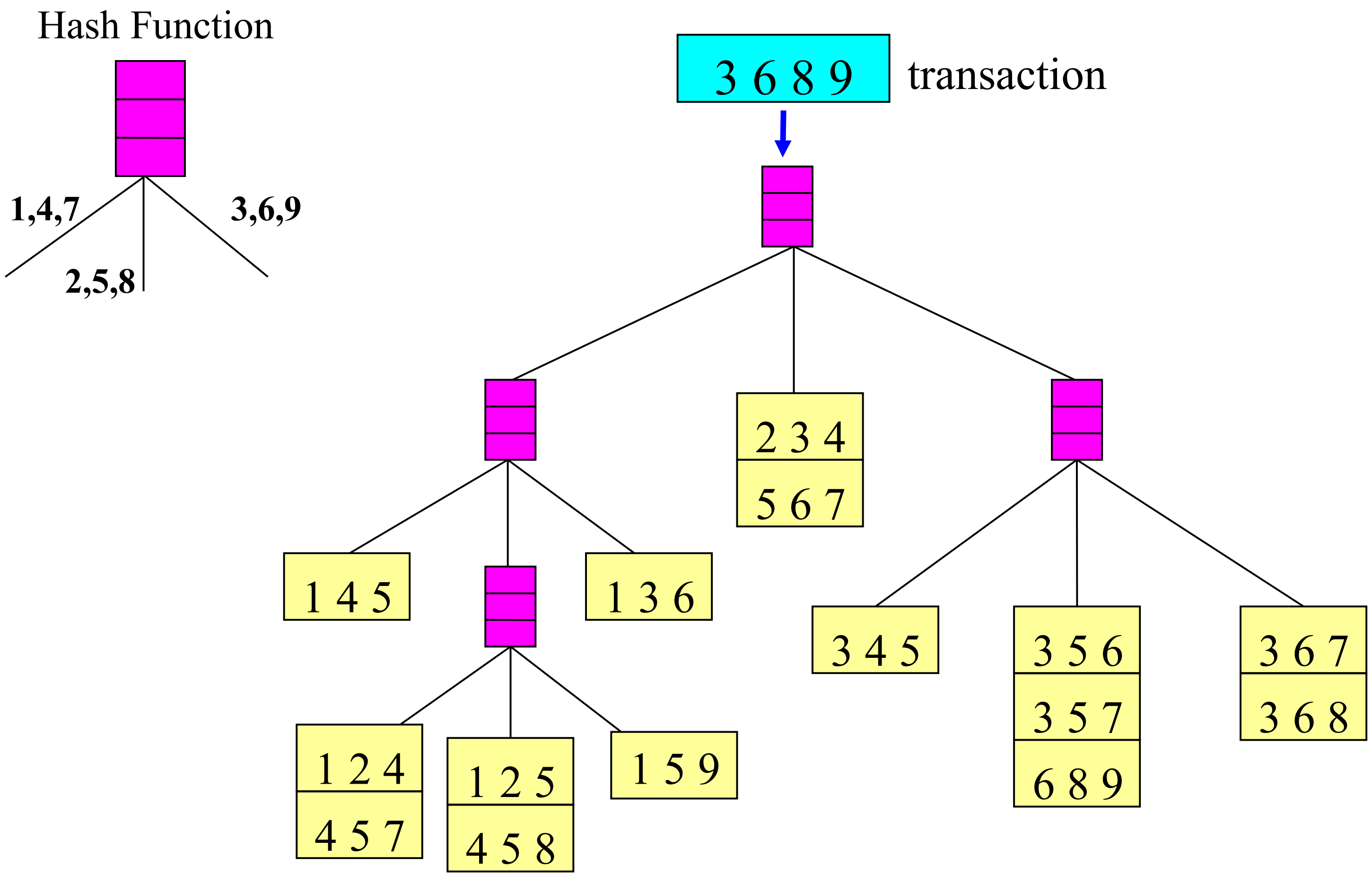


Checking which candidates might be in a transaction



Compare transaction against 11 out of 15 candidates

Try it! Use the hash tree to determine which candidates might be in this transaction

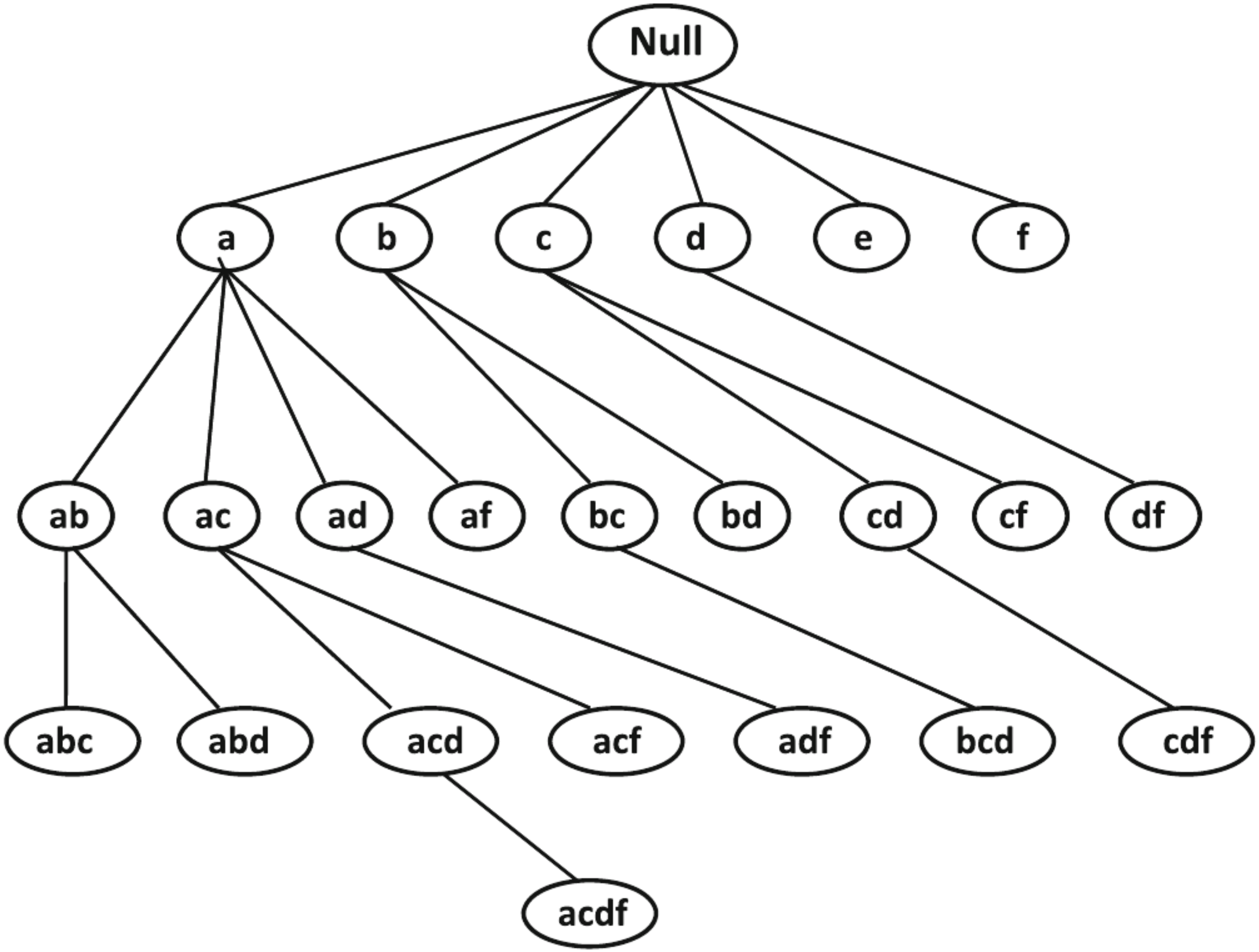


Enumeration-tree algorithms: Lexicographic tree

- There is a node in the tree for each frequent itemset
- The root of the tree contains the null itemset
- If I = $\{i_1, i_2, \ldots, i_k\}$ then the parent of I in the tree is $\{i_1, i_2, \ldots, i_{k-1}\}$

Example

Note that, unlike the lattice, a parent can only be extended with an item that is lexicographically larger



Enumeration tree algorithm

Algorithm *GenericEnumerationTree*(Transactions: $\mathcal{T}$,
Minimum Support: *minsup*)
begin
Initialize enumeration tree $\mathcal{ET}$ to single $Null$ node;
while any node in $\mathcal{ET}$ has not been examined **do begin**
Select one of more unexamined nodes $\mathcal{P}$ from $\mathcal{ET}$ for examination;
Generate candidates extensions $C(P)$ of each node $P \in \mathcal{P}$;
Determine frequent extensions $F(P) \subseteq C(P)$ for each $P \in \mathcal{P}$ with support counting;
Extend each node $P \in \mathcal{P}$ in $\mathcal{ET}$ with its frequent extensions in $F(P)$;
end
return enumeration tree $\mathcal{ET}$;
end

Enumeration-tree-based implementation of Apriori

- Apriori constructs the enumeration tree in a breadth-first manner
- Apriori generates candidate (k+1)-itemsets by merging two frequent k-itemsets of which the first k-1 items are the same ⇒ extension in the enumeration-tree

Summary

Things to remember

- Support and confidence on a rule
- Downward closure property
  - every subset of a frequent itemset is also frequent
  - hence, if an itemset X has a subset that is not frequent, X cannot be frequent
- Methods for candidate generation, pruning
- Algorithms for fast support computation

Exercises for this topic

- Data Mining, The Textbook (2015) by Charu Aggarwal
  - Exercises 4.9 → 9-10
- Mining of Massive Datasets 2nd edition (2014) by Leskovec et al.
  - Exercises 6.2.7 → 6.2.5 and 6.2.6
- Introduction to Data Mining 2nd edition (2019) by Tan et al.
  - Exercises 5.10 → 9-12