Speeding up Association rules

Dynamic Hashing and Pruning technique

Borrowed from software.ssu.ac.kr/AI08_page/class8-Association%20Rules.ppt

DHP: Reduce the Number of Candidates

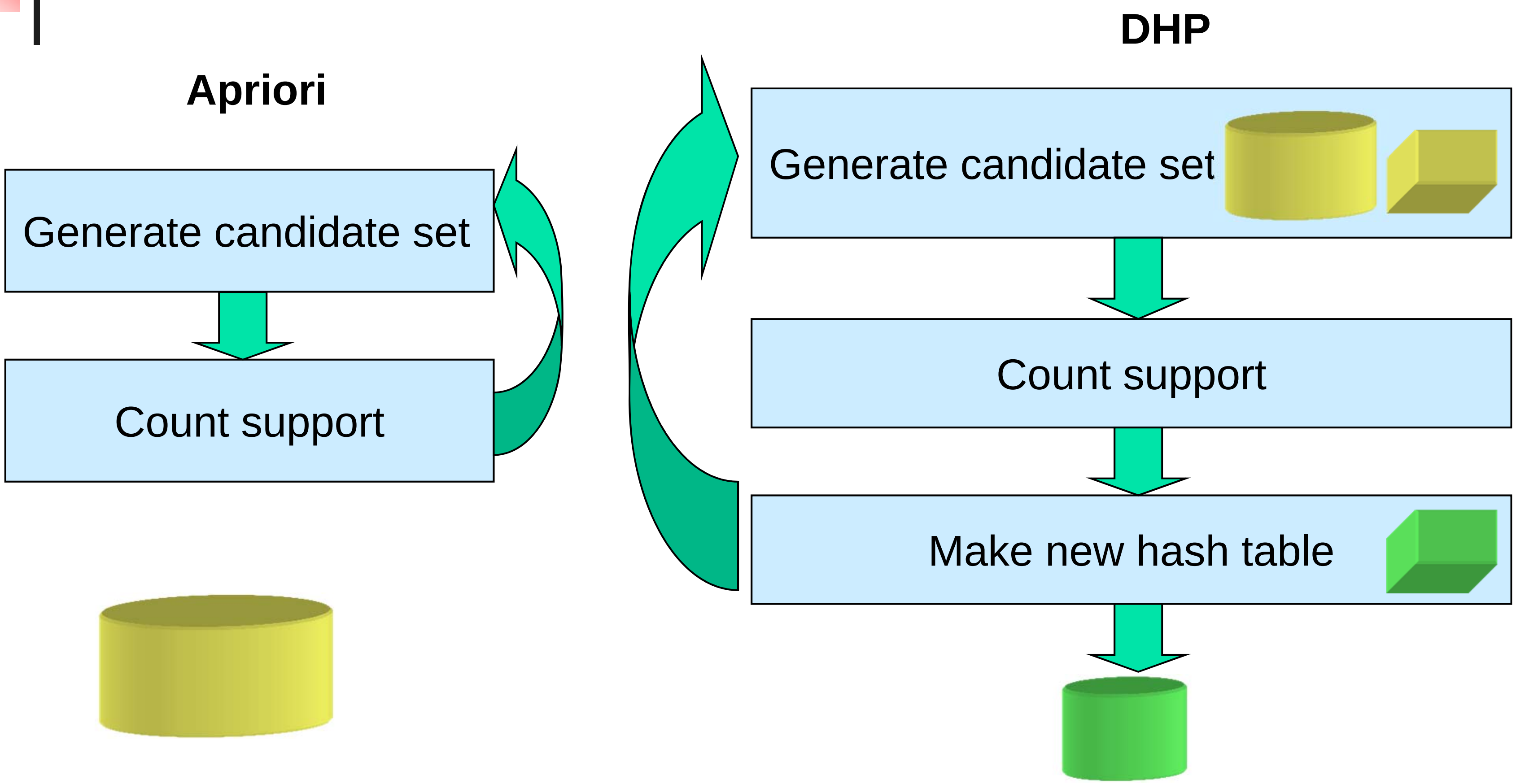
- A *k*-itemset whose corresponding hashing bucket count is below the threshold cannot be frequent
  - Candidates: a, b, c, d, e
  - Hash entries: {ab, ad, ae} {bd, be, de} ...
  - Frequent 1-itemset: a, b, d, e
  - ab is not a candidate 2-itemset if the sum of count of {ab, ad, ae} is below support threshold
- *J. Park, M. Chen, and P. Yu. An effective hash-based algorithm for mining association rules. In SIGMOD'95*

Still challenging, the niche for DHP

- DHP (Park '95): Dynamic Hashing and Pruning

- Candidate large 2-itemsets are huge.
  - DHP: trim them using hashing
- Transaction database is huge that one scan per iteration is costly
  - DHP: prune both number of transactions and number of items in each transaction after each iteration

How does it look like?



Reducing Number of Comparisons

Candidate counting:

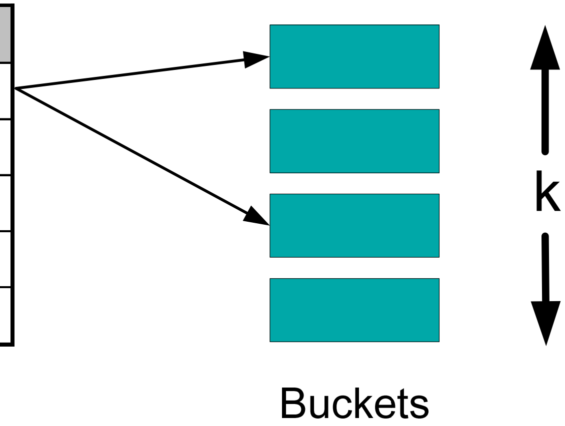
- Scan the database of transactions to determine the support of each candidate itemset
- To reduce the number of comparisons, store the candidates in a hash structure
  - Instead of matching each transaction against every candidate, match it against candidates contained in the hashed buckets

Transactions

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**

Hash Structure



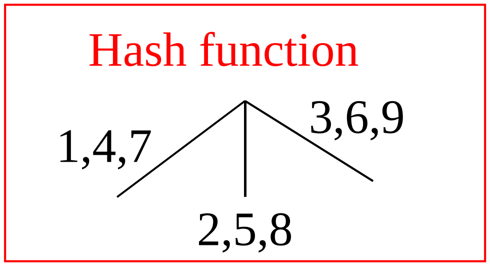
Generate Hash Tree

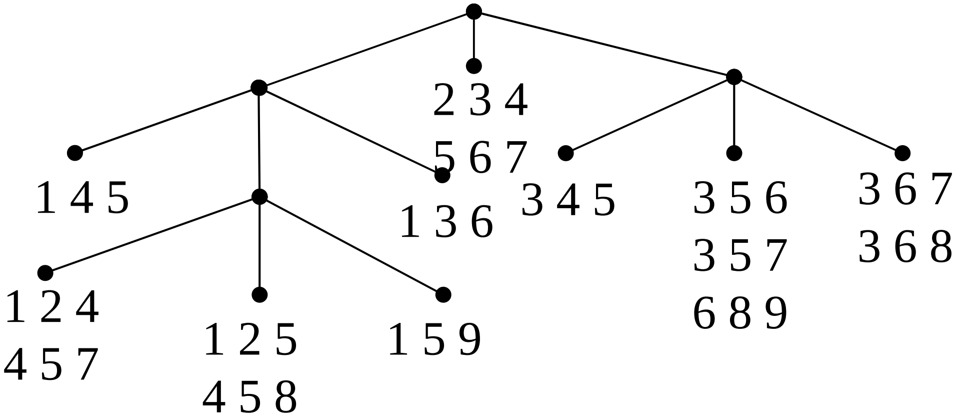
Suppose you have 15 candidate itemsets of length 3:

{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}

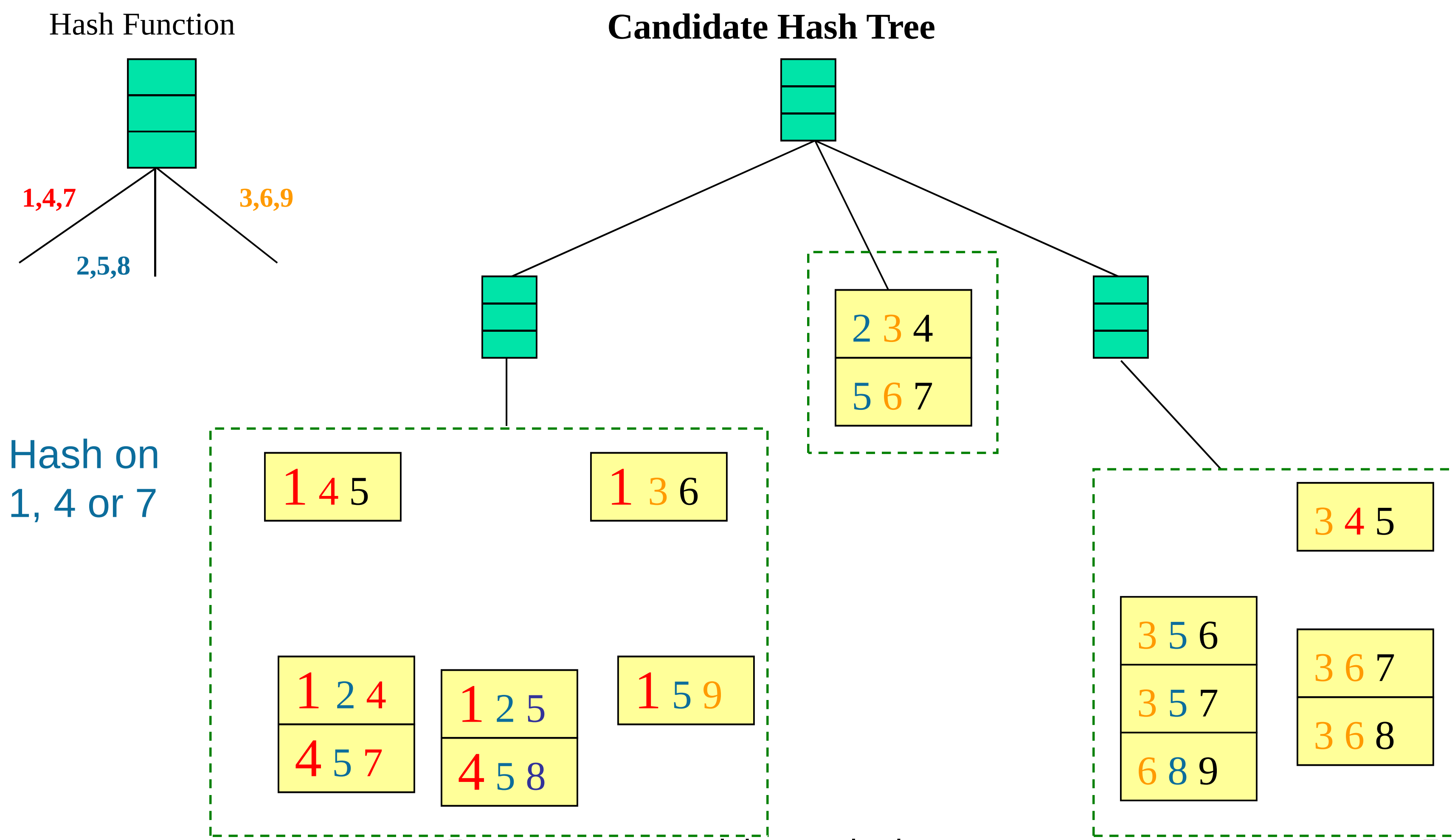
You need:

- 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)

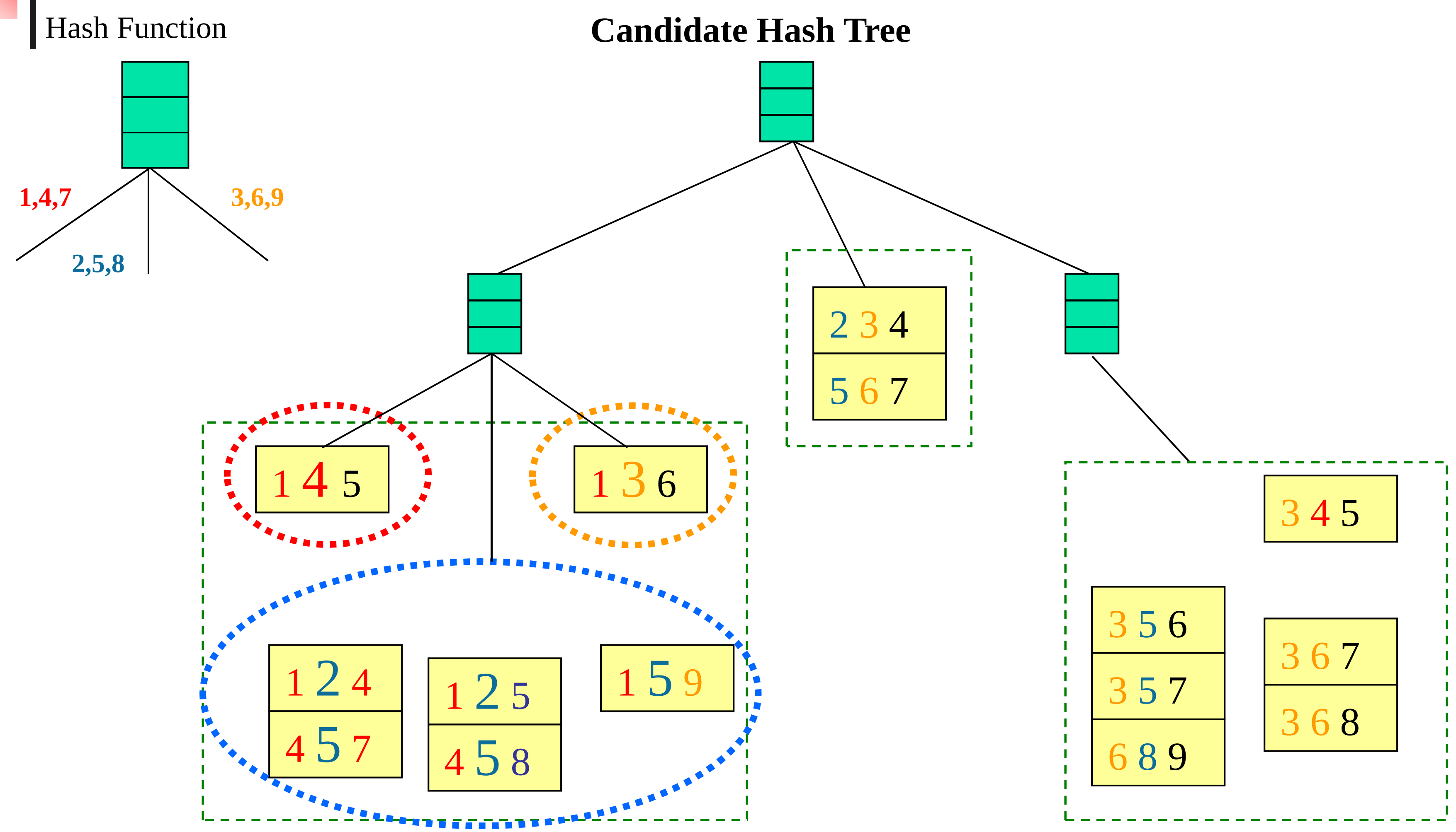




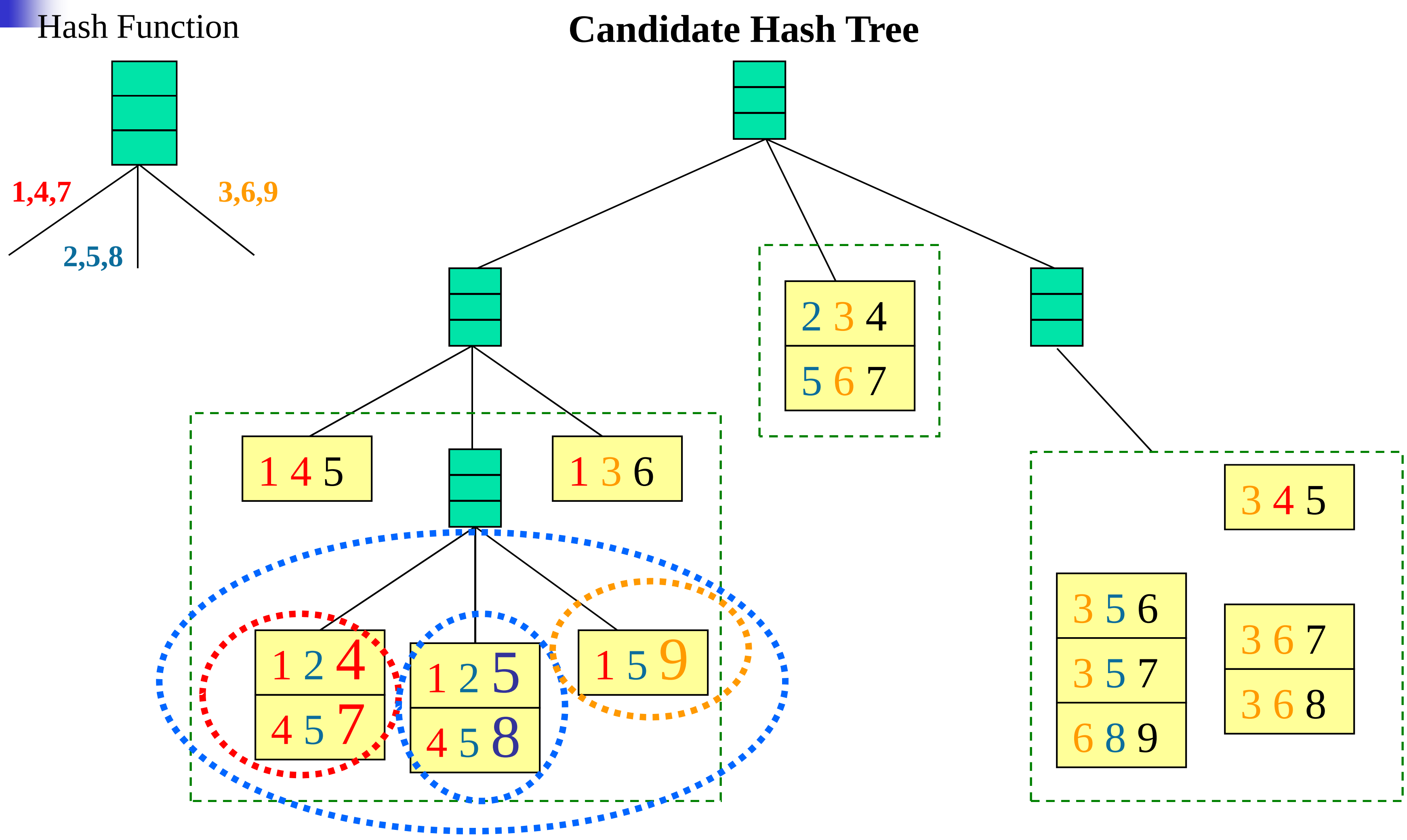
Association Rule Discovery: Hash tree



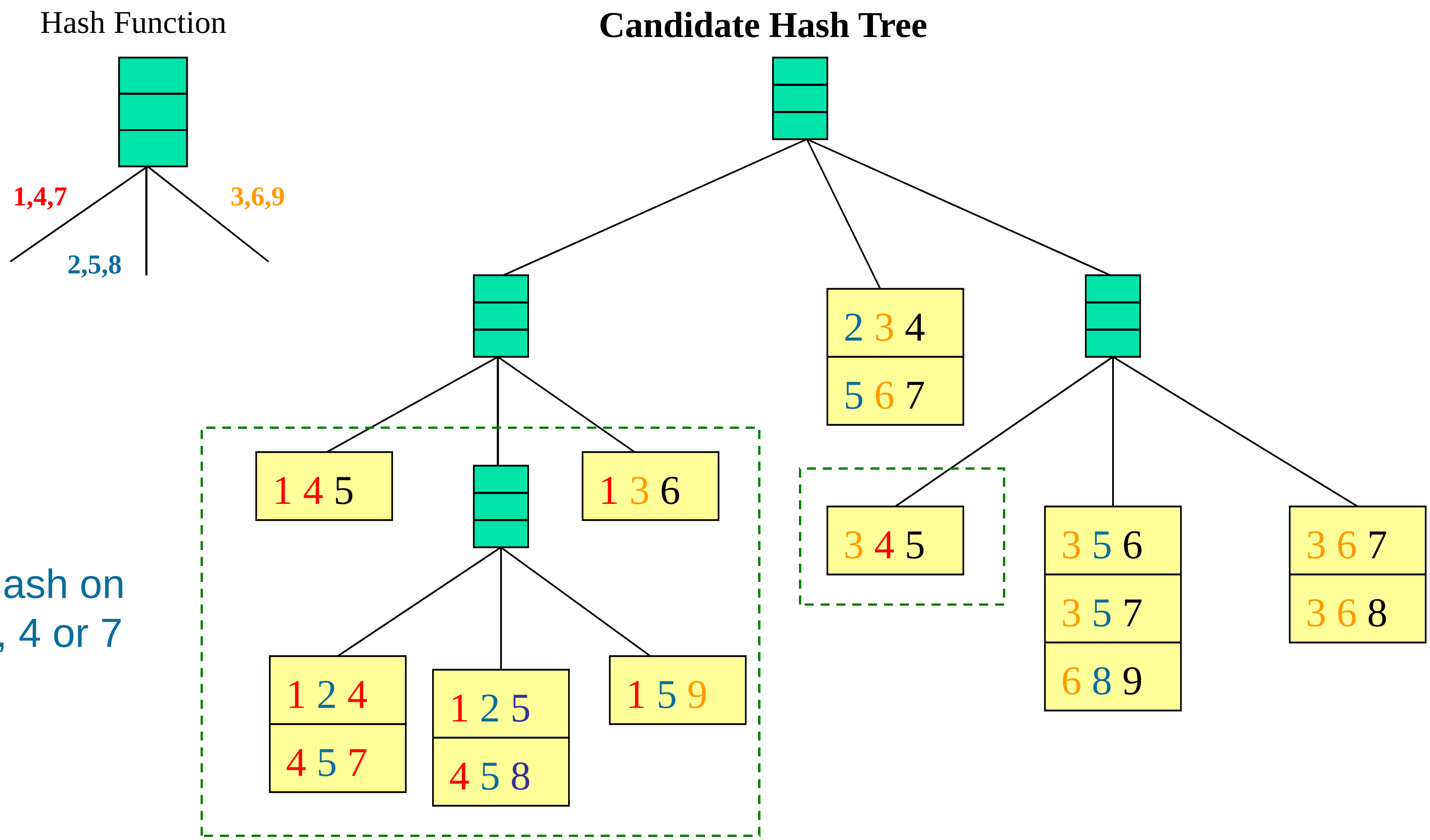
Association Rule Discovery: Hash tree



Association Rule Discovery: Hash tree

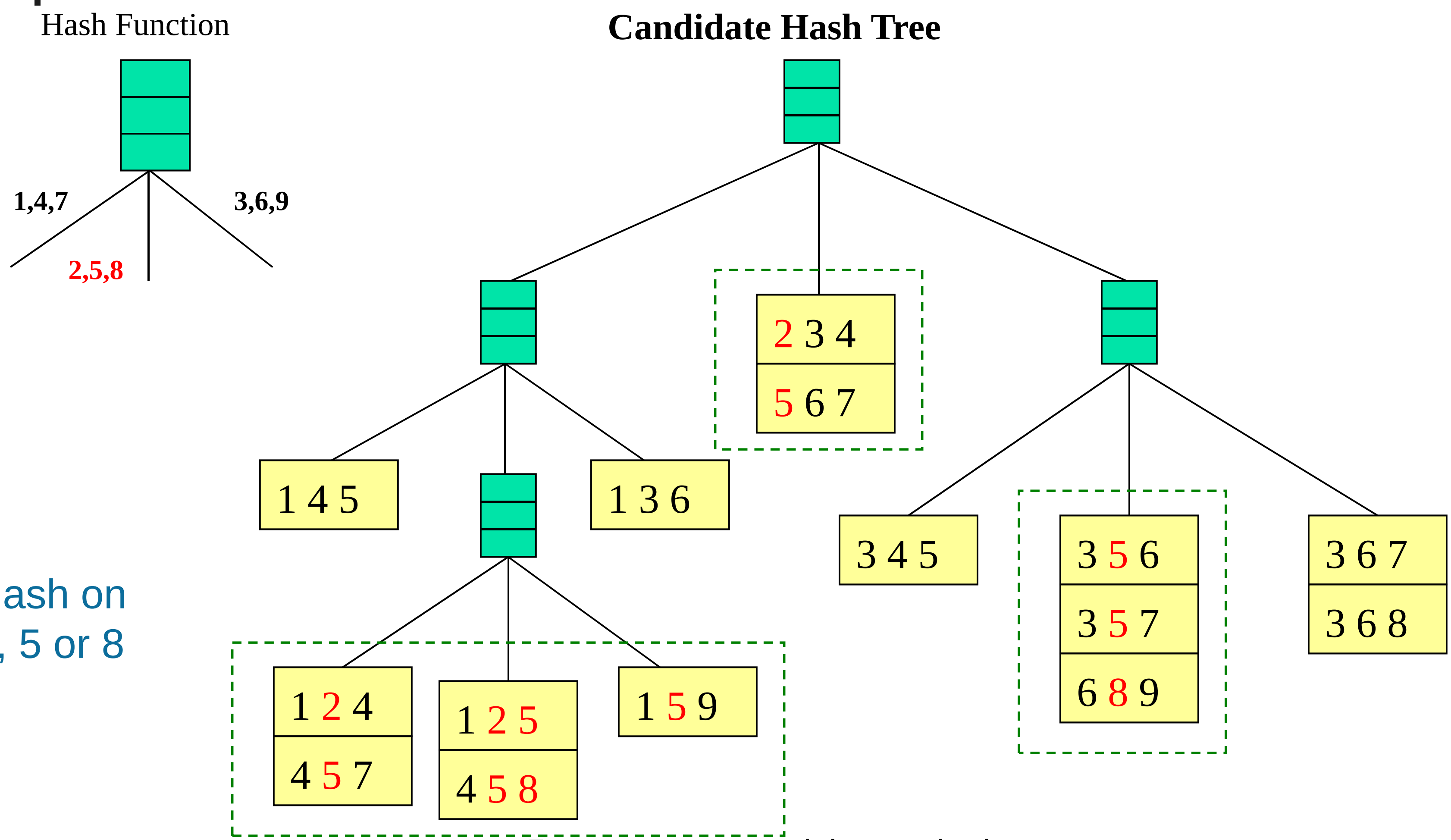


Association Rule Discovery: Hash tree



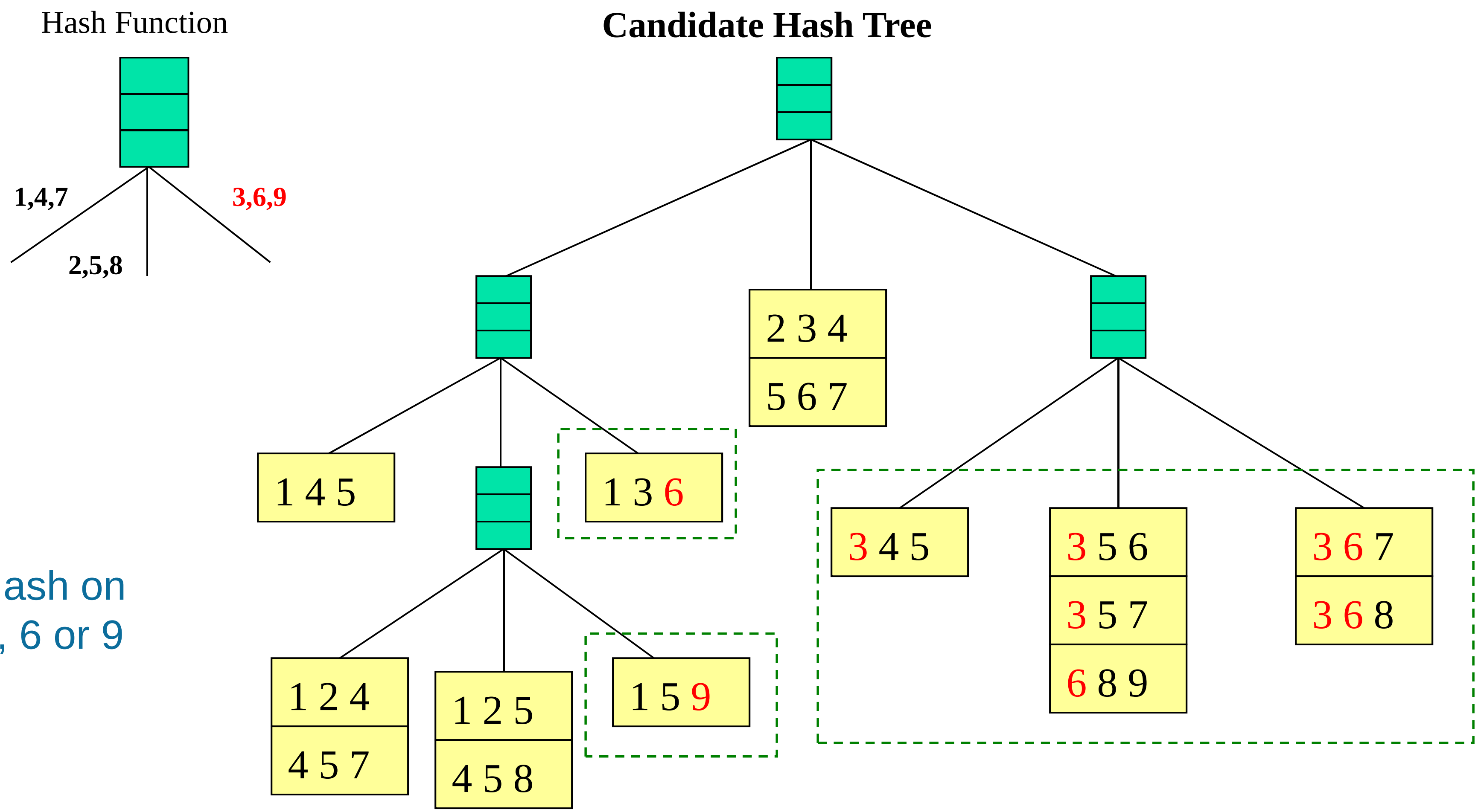
Hash on
1, 4 or 7

Association Rule Discovery: Hash tree



Hash on
2, 5 or 8

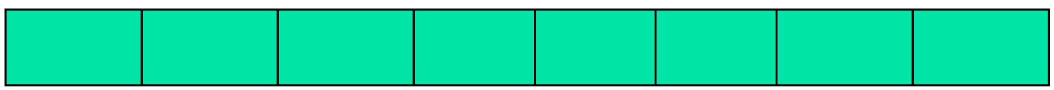
Association Rule Discovery: Hash tree



Hash on
3, 6 or 9

How to trim candidate itemsets

- In k-iteration, hash all “appearing” k+1 itemsets in a hashtable, count all the occurrences of an itemset in the correspondent bucket.



- In k+1 iteration, examine each of the candidate itemset to see if its correspondent bucket value is above the support (necessary condition)

Hash Table Construction

- Consider two items sets, all itesms are numbered as i1, i2, ...in. For any any pair (x, y), has according to
  - Hash function bucket #=

    h({x y}) = ((order of x)*10+(order of y)) % 7
- Example:
  - Items = A, B, C, D, E,
  - Order = 1, 2, 3 4, 5,
  - H({C, E})= (3*10 + 5) % 7 = 0
  - Thus, {C, E} belong to bucket 0.

Example

TID	Items
100	A C D
200	B C E
300	A B C E
400	B E

Figure1. An example transaction database

Generation of C1 & L1(1st iteration)

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

C1

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

L1

Hash Table Construction

- Find all 2-itemset of each transaction

TID	2-itemset
100	{A C} {A D} {C D}
200	{B C} {B E} {C E}
300	{A B} {A C} {A E} {B C} {B E} {C E}
400	{B E}

Hash Table Construction (2)

- Hash function

  h({x y}) = ((order of x)*10+(order of y)) % 7

- Hash table

	{C E}	{A E}	{B C}		{B E}	{A B}	{A C}
	{C E}		{B C}		{B E}		{C D}
	{A D}				{B E}		{A C}
	3	1	2	0	3	1	3
bucket	0	1	2	3	4	5	6

C2 Generation (2nd iteration)

L1*L1	# in the bucket
{A B}	1
{A C}	3
{A E}	1
{B C}	2
{B E}	3
{C E}	3

Resulted C2
{A C}
{B C}
{B E}
{C E}

C2 of Apriori
{A B}
{A C}
{A E}
{B C}
{B E}
{C E}

Effective Database Pruning

- Apriori
- Don't prune database.
- Prune C_k by support counting on the original database.

- DHP
- *More efficient support counting can be achieved on pruned database.*

Performance Comparison

	Apriori	DHP		
	number	number	D_k,	$\lvert D_k \rvert$
L_1	820	820	6,700KB,	100,000
C_2	335,790	338	6,700KB,	100,000
L_2	207	207		
C_3	618	618	659KB,	20,602
L_3	201	201		
C_4	184	184	546KB,	17,417
L_4	98	98		
C_5	30	30	332KB,	10,149
L_5	23	23		
C_6	1	1	24KB,	756
L_6	1	1		
total time	39.39	13.91		

Performance Comparison (2)

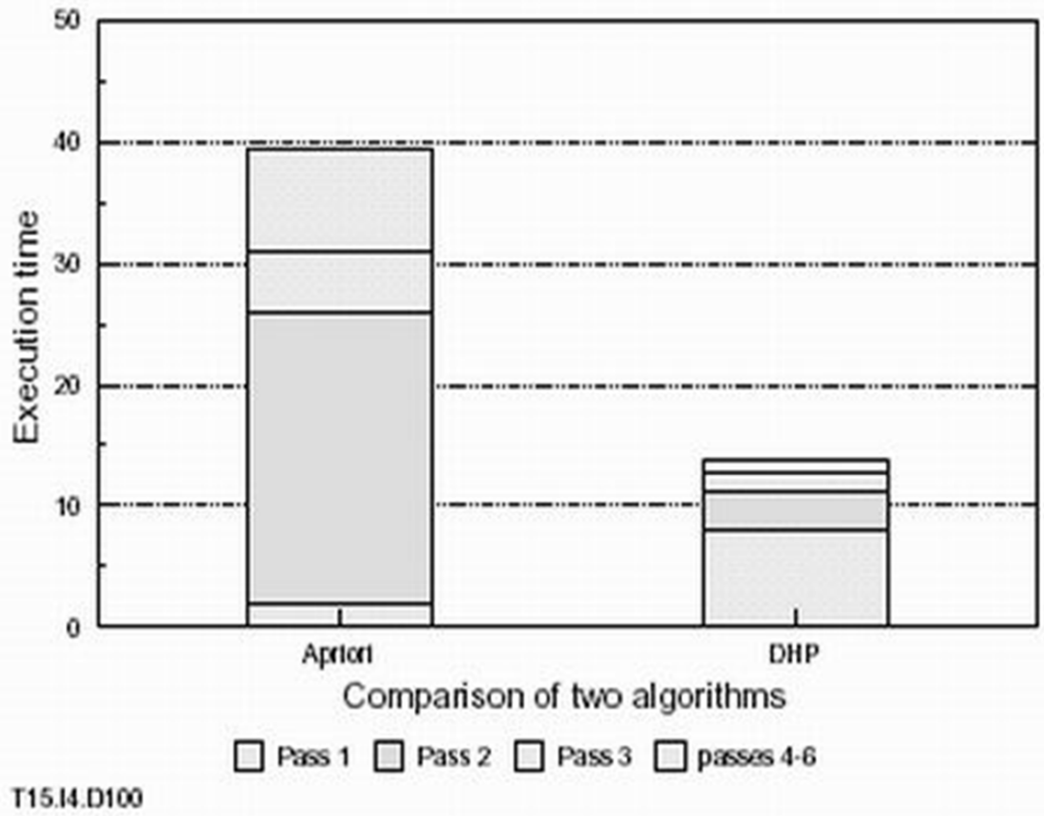


Figure 8: Execution time of Apriori and DHP

Conclusion

- Effective hash-based algorithm for the candidate itemset generation
- Two phase transaction database pruning
- Much more efficient (time & space) than Apriori algorithm