DATA MINING

Result Post-Processing
Alternative Algorithms

Slides originally by Panayiotis Tsaparas, modified by Stephanie Schwartz

RESULT POST-PROCESSING

Reducing the # of frequent itemsets

Reducing the number of rules

Compact Representation of Frequent Itemsets

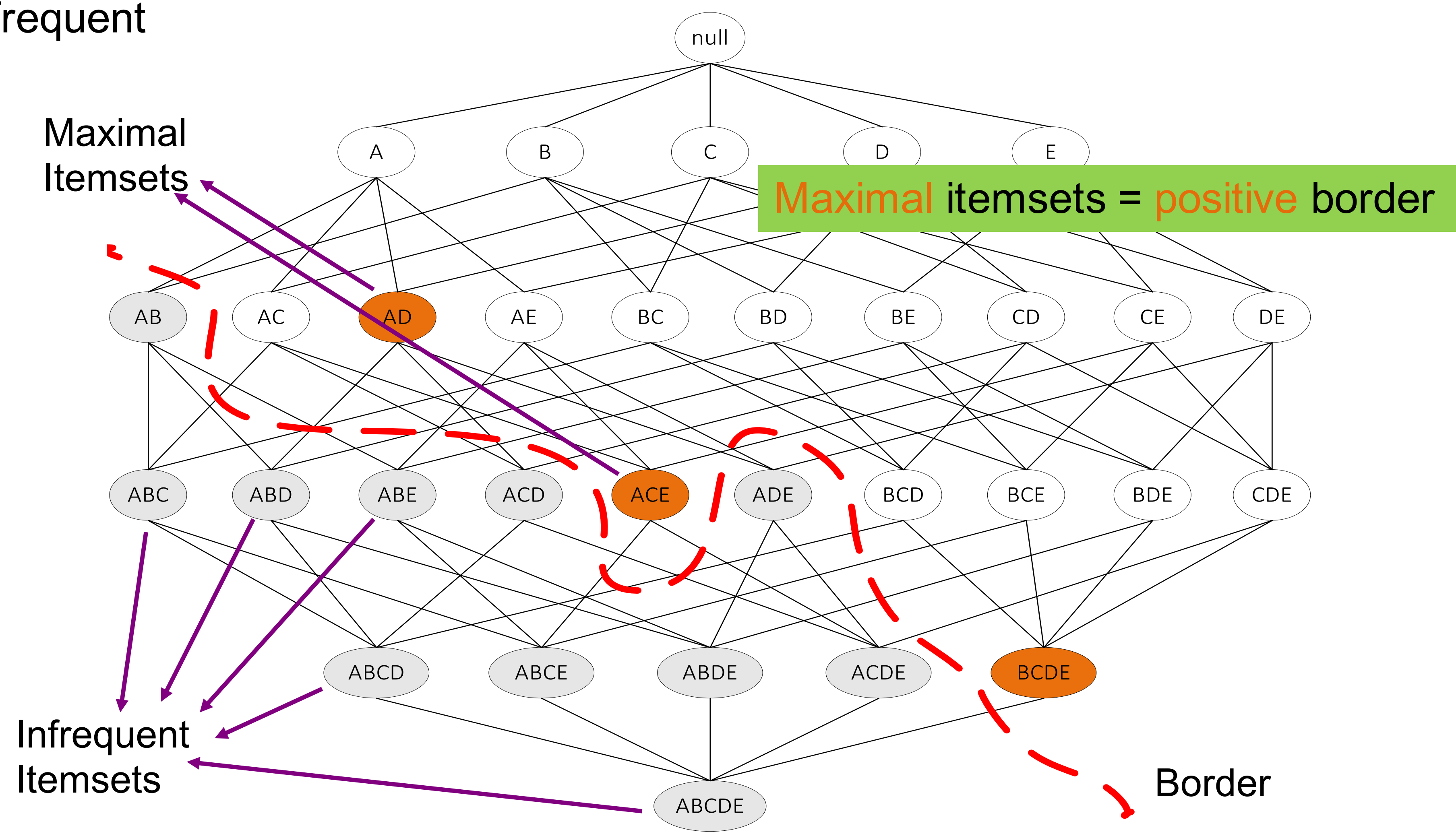
- Some itemsets are redundant because they have identical support as their supersets

TID	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	B1	B2	B3	B4	B5	B6	B7	B8	B9	B10	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10
1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1

- Number of frequent itemsets $= 3 \times \sum_{k=1}^{10} \binom{10}{k}$
- Need a compact representation

Maximal Frequent Itemset

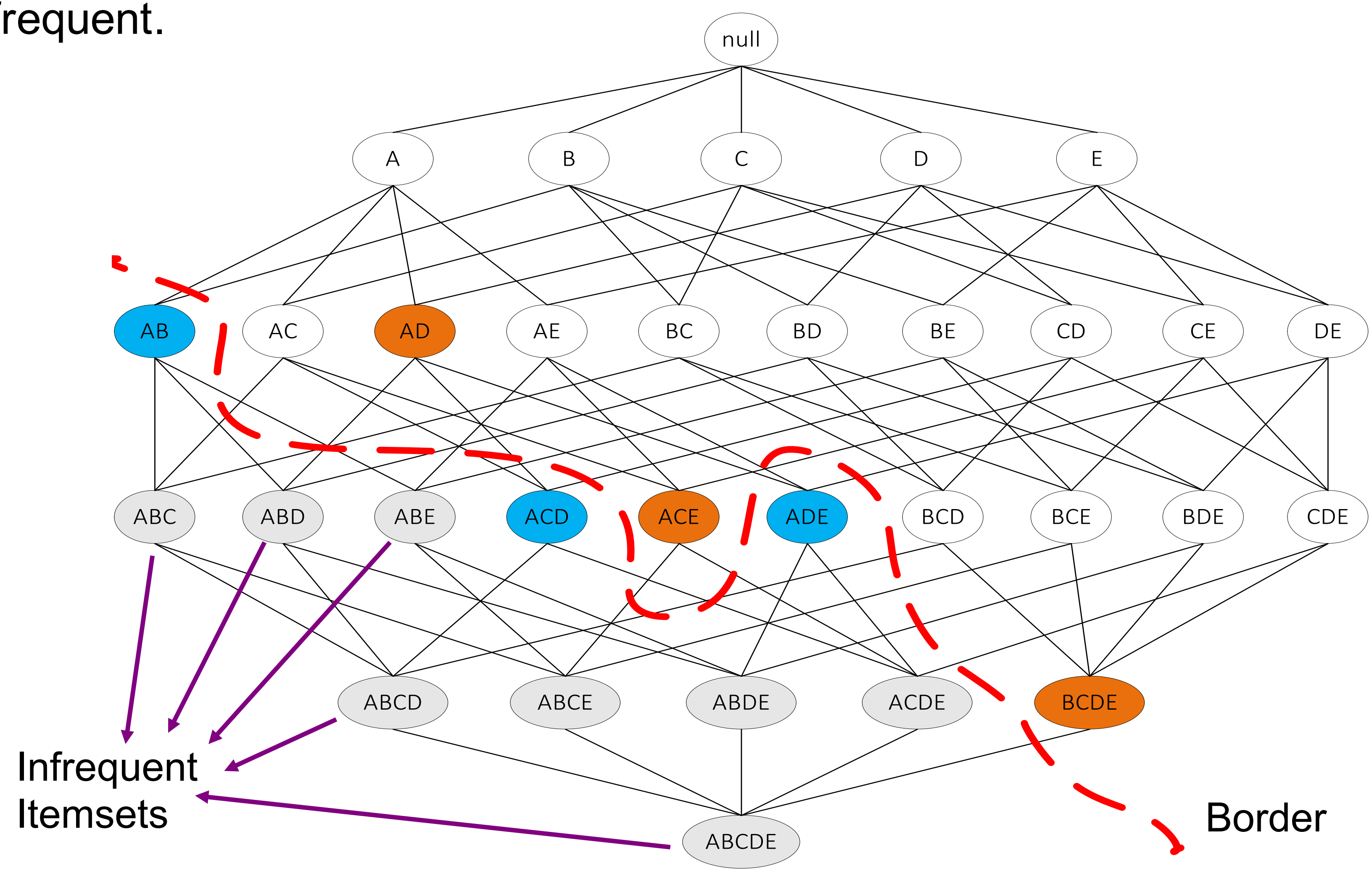
An itemset is maximal frequent if none of its immediate supersets is frequent



Maximal: no superset has this property

Negative Border

Itemsets that are not frequent, but all their immediate subsets are frequent.



Minimal: no subset has this property

Border

- Border = Positive Border + Negative Border
  - Itemsets such that all their immediate subsets are frequent and all their immediate supersets are infrequent.

- Either the positive, or the negative border is sufficient to summarize all frequent itemsets.

Closed Itemset

- An itemset is closed if none of its immediate supersets has the same support as the itemset

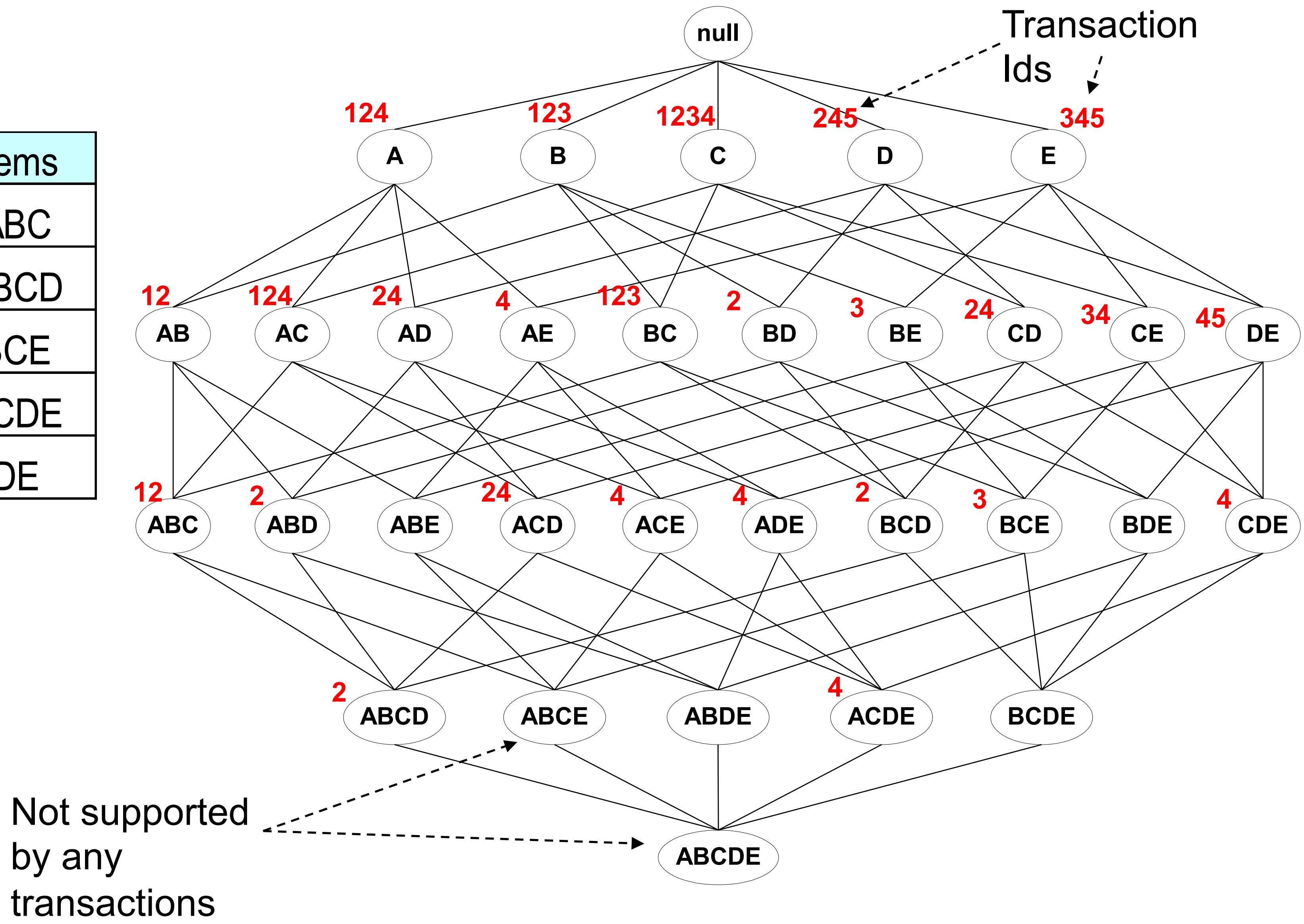
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,B,C,D}
4	{A,B,D}
5	{A,B,C,D}

Itemset	Support
{A}	4
{B}	5
{C}	3
{D}	4
{A,B}	4
{A,C}	2
{A,D}	3
{B,C}	3
{B,D}	4
{C,D}	3

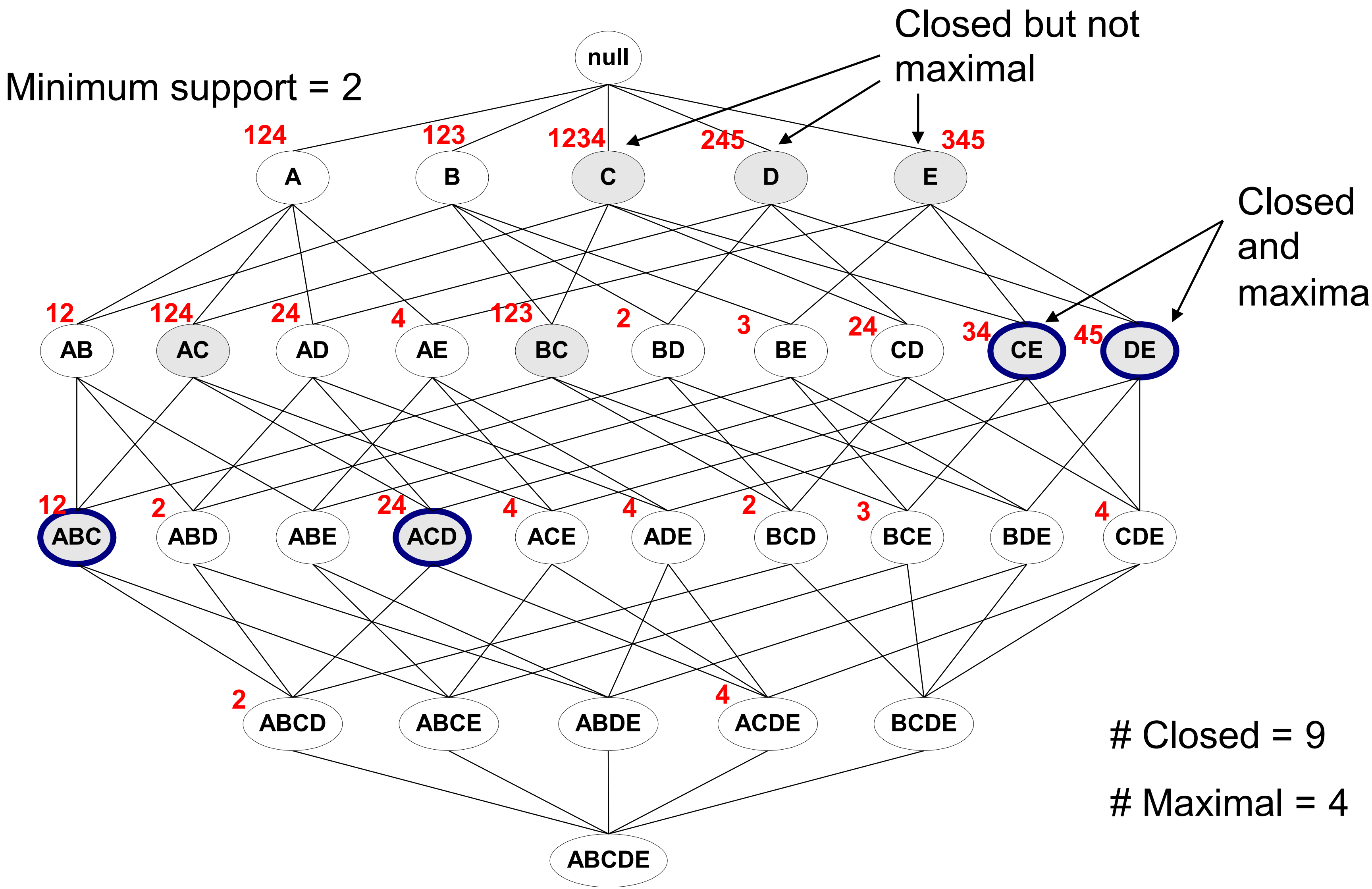
Itemset	Support
{A,B,C}	2
{A,B,D}	3
{A,C,D}	2
{B,C,D}	3
{A,B,C,D}	2

Maximal vs Closed Itemsets

TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



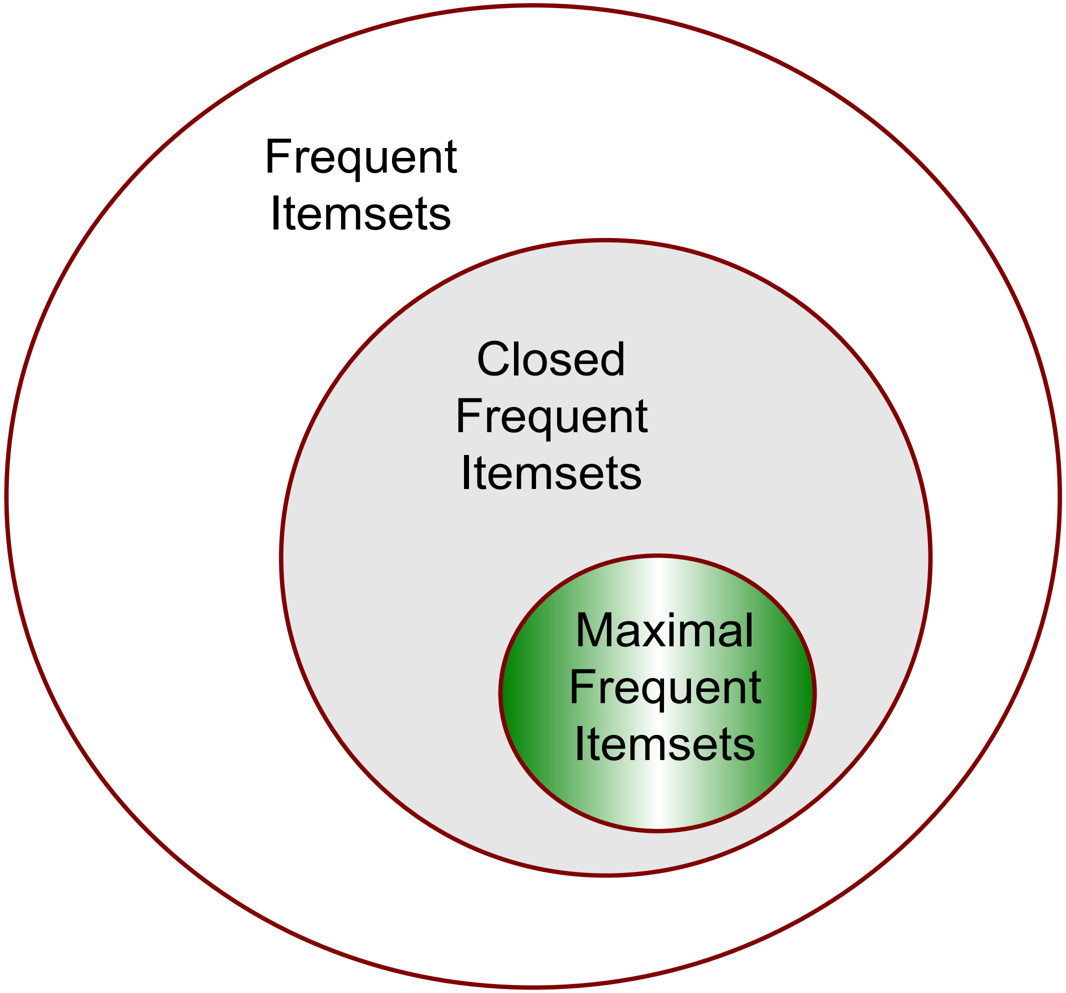
Maximal vs Closed Frequent Itemsets



Closed = 9

Maximal = 4

Maximal vs Closed Itemsets



Pattern Evaluation

- Association rule algorithms tend to produce too many rules but many of them are uninteresting or redundant
  - Redundant if {A,B,C} → {D} and {A,B} → {D} have same support & confidence
    - Summarization techniques
  - Uninteresting, if the pattern that is revealed does not offer useful information.
    - Interestingness measures: a hard problem to define
- Interestingness measures can be used to prune/rank the derived patterns
  - Subjective measures: require human analyst
  - Objective measures: rely on the data.
- In the original formulation of association rules, support & confidence are the only measures used

Computing Interestingness Measure

- Given a rule X → Y, information needed to compute rule interestingness can be obtained from a contingency table

Contingency table for X → Y

	Y	$\bar{Y}$	
X	f_{11}	f_{10}	f_{1+}
$\bar{X}$	f_{01}	f_{00}	f_{o+}
	f_{+1}	f_{+0}	N

f_{11}: support of X and Y
f_{10}: support of X and $\bar{Y}$
f_{01}: support of $\bar{X}$ and Y
f_{00}: support of $\bar{X}$ and $\bar{Y}$

X: itemset X appears in tuple
Y: itemset Y appears in tuple
$\bar{X}$: itemset X does not appear in tuple
$\bar{Y}$: itemset Y does not appear in tuple

Used to define various measures

- support, confidence, lift, Gini, J-measure, etc.

Drawback of Confidence

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

Number of people that drink tea

Number of people that drink coffee and tea

Number of people that drink coffee but not tea

Number of people that drink coffee

Association Rule: Tea → Coffee

Confidence= P(Coffee|Tea) = $\frac{15}{20} = 0.75$

but P(Coffee) = $\frac{90}{100} = 0.9$

- Although confidence is high, rule is misleading
- P(Coffee|$\overline{\text{Tea}}$) = 0.9375

Statistical Independence

- Population of 1000 students
  - 600 students know how to swim (S)
  - 700 students know how to bike (B)
  - 420 students know how to swim and bike (S,B)

  - $P(S \wedge B) = 420/1000 = 0.42$
  - $P(S) \times P(B) = 0.6 \times 0.7 = 0.42$

  - $P(S \wedge B) = P(S) \times P(B)$ => Statistical independence

Statistical Independence

- Population of 1000 students
  - 600 students know how to swim (S)
  - 700 students know how to bike (B)
  - 500 students know how to swim and bike (S,B)

  - $P(S \wedge B) = 500/1000 = 0.5$
  - $P(S) \times P(B) = 0.6 \times 0.7 = 0.42$

  - $P(S \wedge B) > P(S) \times P(B)$ => Positively correlated

Statistical Independence

- Population of 1000 students
  - 600 students know how to swim (S)
  - 700 students know how to bike (B)
  - 300 students know how to swim and bike (S,B)

  - $P(S \wedge B) = 300/1000 = 0.3$
  - $P(S) \times P(B) = 0.6 \times 0.7 = 0.42$

  - $P(S \wedge B) < P(S) \times P(B)$ => Negatively correlated

Statistical-based Measures

- Measures that take into account statistical dependence
  - Lift/Interest/PMI

$$\text{Lift} = \frac{P(Y|X)}{P(Y)} = \frac{P(X,Y)}{P(X)P(Y)} = \text{Interest}$$

In text mining it is called: Pointwise Mutual Information

  - Piatesky-Shapiro

$$\text{PS} = P(X,Y) - P(X)P(Y)$$

- All these measures measure deviation from independence
  - The higher, the better (why?)

Example: Lift/Interest

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

Association Rule: Tea $\rightarrow$ Coffee

Confidence= P(Coffee|Tea) = 0.75

but P(Coffee) = 0.9

$\Rightarrow$ Lift = 0.75/0.9= 0.8333 (< 1, therefore is negatively associated)

= 0.15/(0.9*0.2)

Another Example

	of	the	of, the
Fraction of documents	0.9	0.9	0.8

$$P(\text{of}, \text{the}) \approx P(\text{of})P(\text{the})$$

If I was creating a document by picking words randomly, (of, the) have more or less the same probability of appearing together by chance

No correlation

	hong	kong	hong, kong
Fraction of documents	0.2	0.2	0.19

$$P(\text{hong}, \text{kong}) \gg P(\text{hong})P(\text{kong})$$

(hong, kong) have much lower probability to appear together by chance. The two words appear almost always only together

Positive correlation

	obama	karagounis	obama, karagounis
Fraction of documents	0.2	0.2	0.001

$$P(\text{obama}, \text{karagounis}) \ll P(\text{obama})P(\text{karagounis})$$

(obama, karagounis) have much higher probability to appear together by chance. The two words appear almost never together

Negative correlation

Drawbacks of Lift/Interest/Mutual Information

	honk	konk	honk, konk
Fraction of documents	0.0001	0.0001	0.0001

$$MI(honk, konk) = \frac{0.0001}{0.0001 * 0.0001} = 10000$$

	hong	kong	hong, kong
Fraction of documents	0.2	0.2	0.19

$$MI(hong, kong) = \frac{0.19}{0.2 * 0.2} = 4.75$$

Rare co-occurrences are deemed more interesting.
But this is not always what we want

THE FP-TREE AND THE FP-GROWTH ALGORITHM

Slides from course lecture of E. Pitoura

Overview

- The FP-tree contains a compressed representation of the transaction database.
  - A trie (prefix-tree) data structure is used
  - Each transaction is a path in the tree – paths can overlap.

- Once the FP-tree is constructed the recursive, divide-and-conquer FP-Growth algorithm is used to enumerate all frequent itemsets.

FP-tree Construction

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

- The FP-tree is a trie (prefix tree)
- Since transactions are sets of items, we need to transform them into ordered sequences so that we can have prefixes
  - Otherwise, there is no common prefix between sets {A,B} and {B,C,A}
- We need to impose an order to the items
  - Initially, assume a lexicographic order.

FP-tree Construction

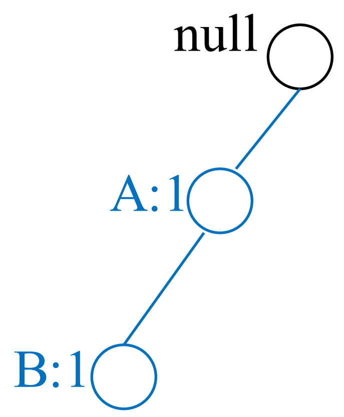
- Initially the tree is empty

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

FP-tree Construction

- Reading transaction TID = 1

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}



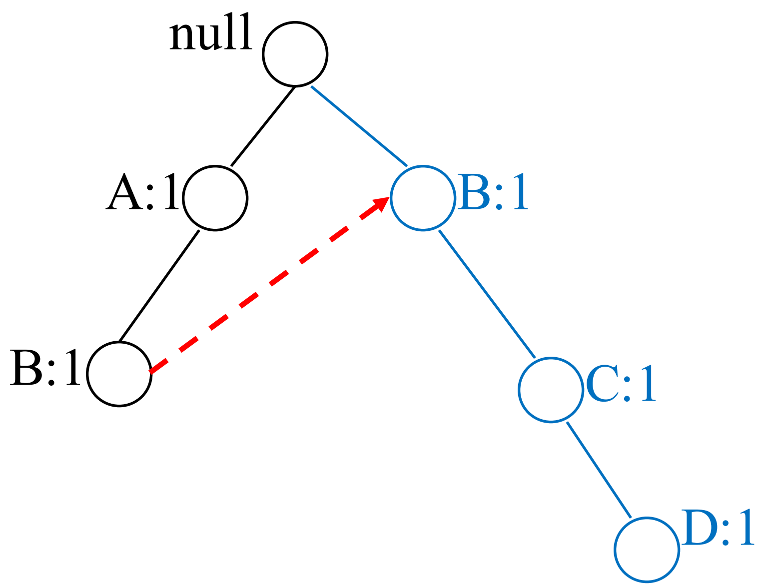
Node label = item:support

- Each node in the tree has a label consisting of the item and the support (number of transactions that reach that node, i.e. follow that path)

FP-tree Construction

- Reading transaction TID = 2

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}



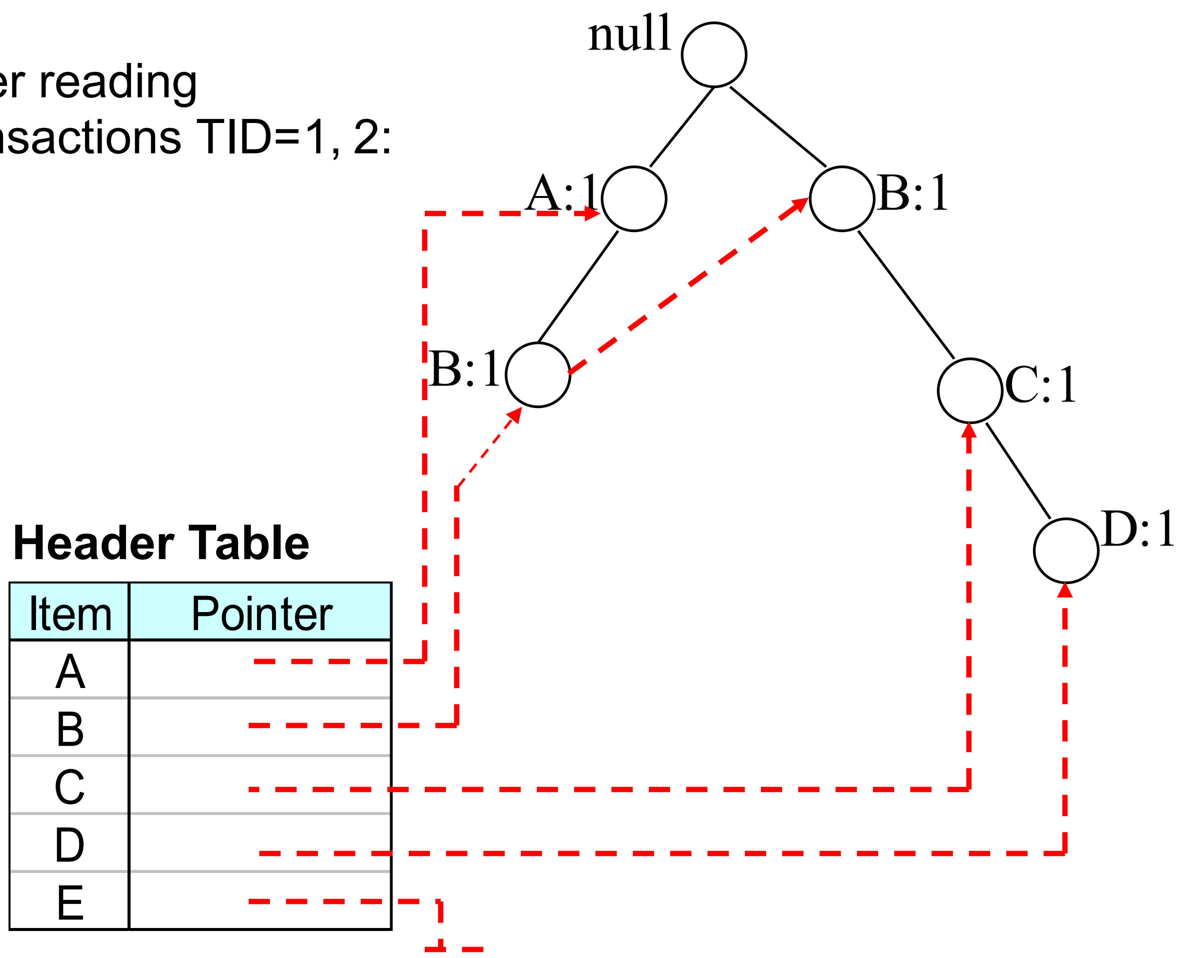
Each transaction is a path in the tree

- We add pointers between nodes that refer to the same item

FP-tree Construction

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

After reading transactions TID=1, 2:

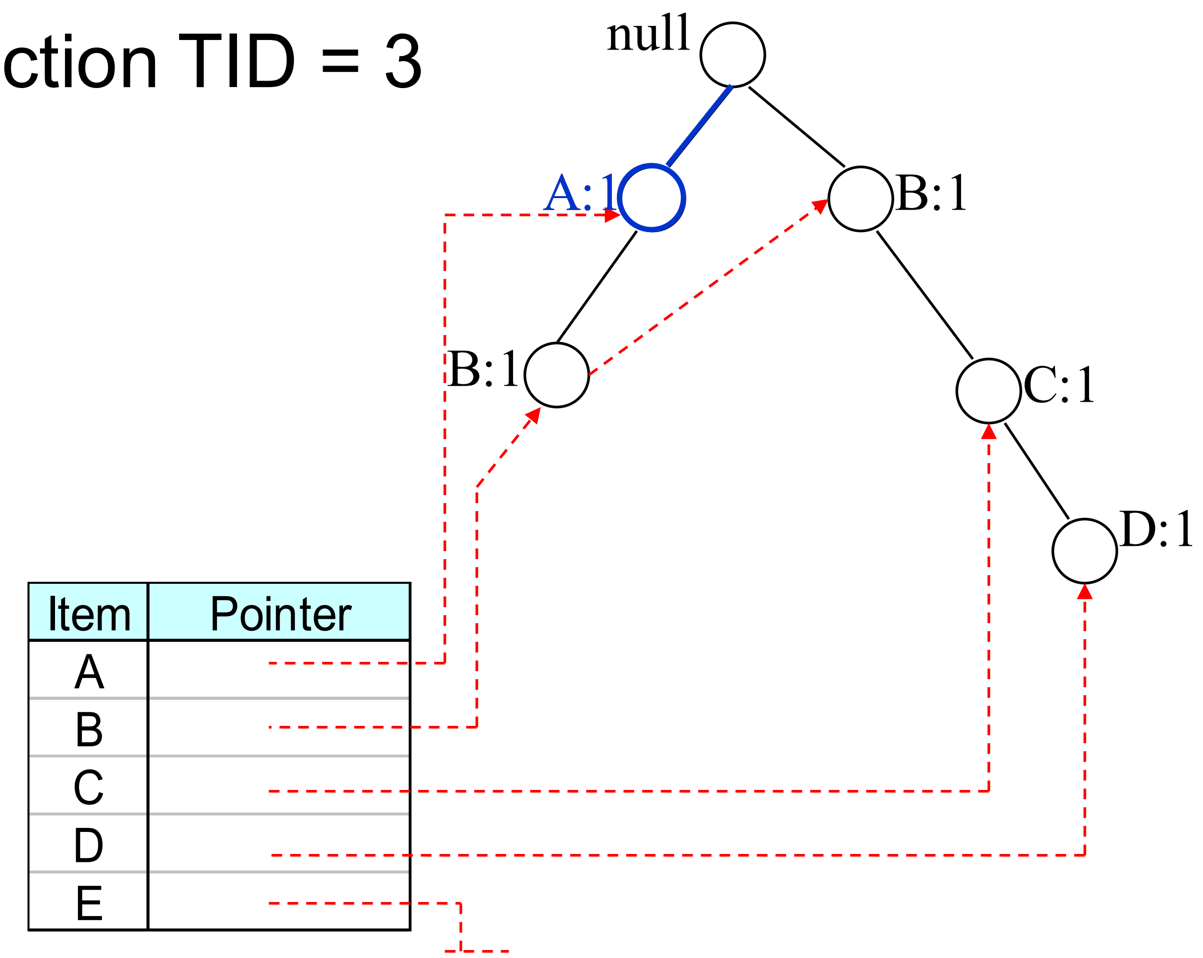


The Header Table and the pointers assist in computing the itemset support

FP-tree Construction

- Reading transaction TID = 3

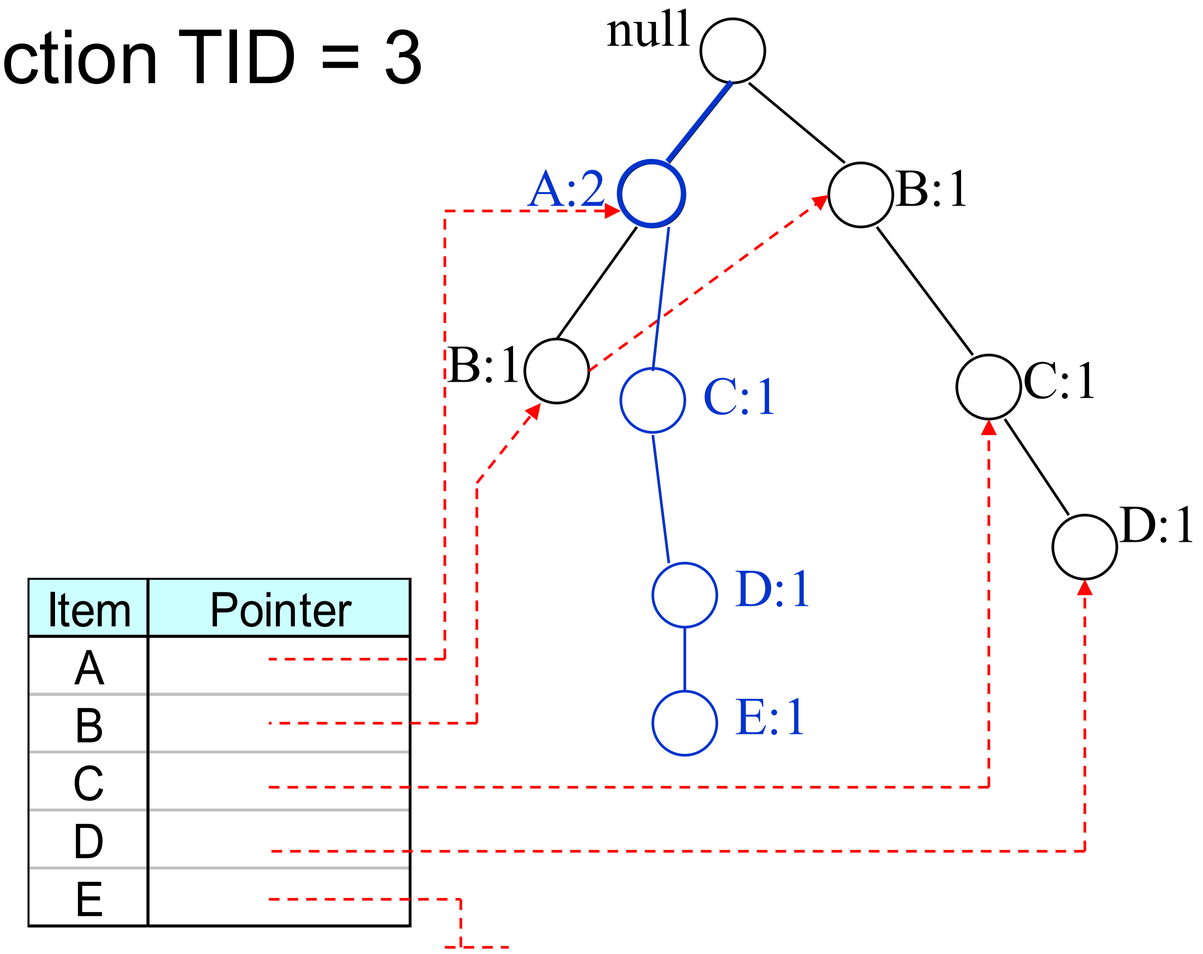
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}



FP-tree Construction

- Reading transaction TID = 3

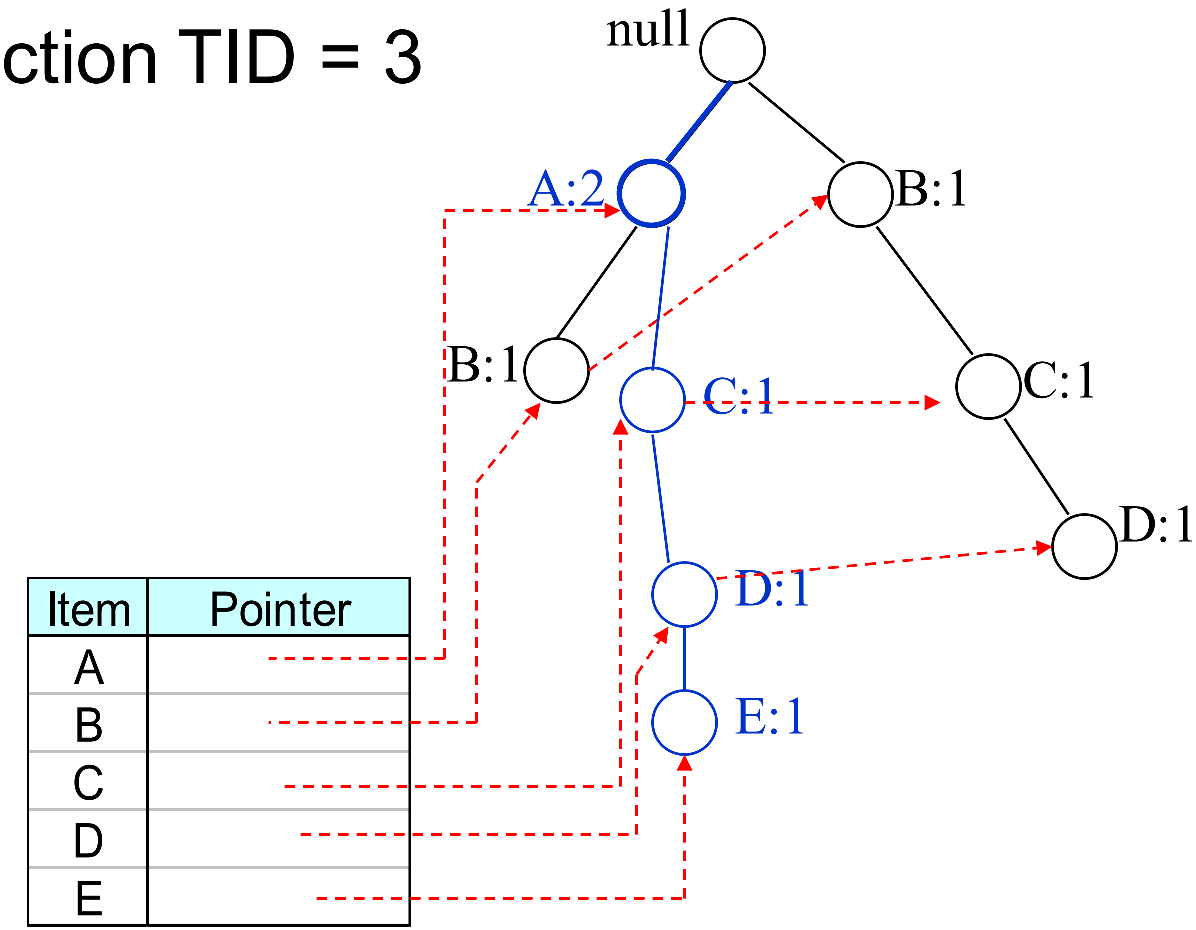
TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}



FP-tree Construction

- Reading transaction TID = 3

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}



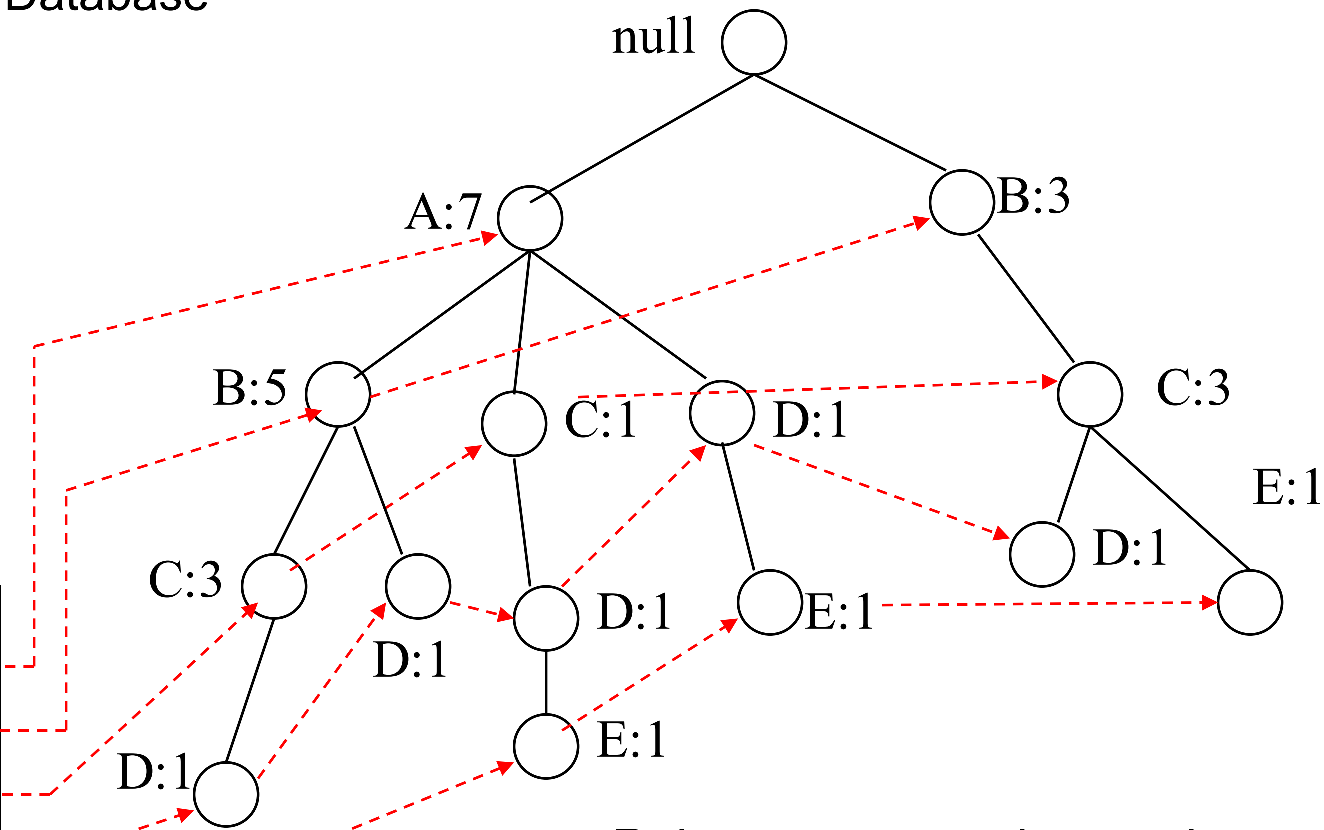
Each transaction is a path in the tree

FP-Tree Construction

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Transaction Database

Each transaction is a path in the tree



Header table

Item	Pointer
A	
B	
C	
D	
E	

Pointers are used to assist frequent itemset generation

FP-tree size

- Every transaction is a path in the FP-tree
- The size of the tree depends on the compressibility of the data
  - Extreme case: All transactions are the same, the FP-tree is a single branch
  - Extreme case: All transactions are different the size of the tree is the same as that of the database (bigger actually since we need additional pointers)

Item ordering

- The size of the tree also depends on the ordering of the items.
- Heuristic: order the items in according to their frequency from larger to smaller.
  - We would need to do an extra pass over the dataset to count frequencies
- Example:

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

σ(A)=7, σ(B)=8, σ(C)=7, σ(D)=5, σ(E)=3

Ordering : B,A,C,D,E

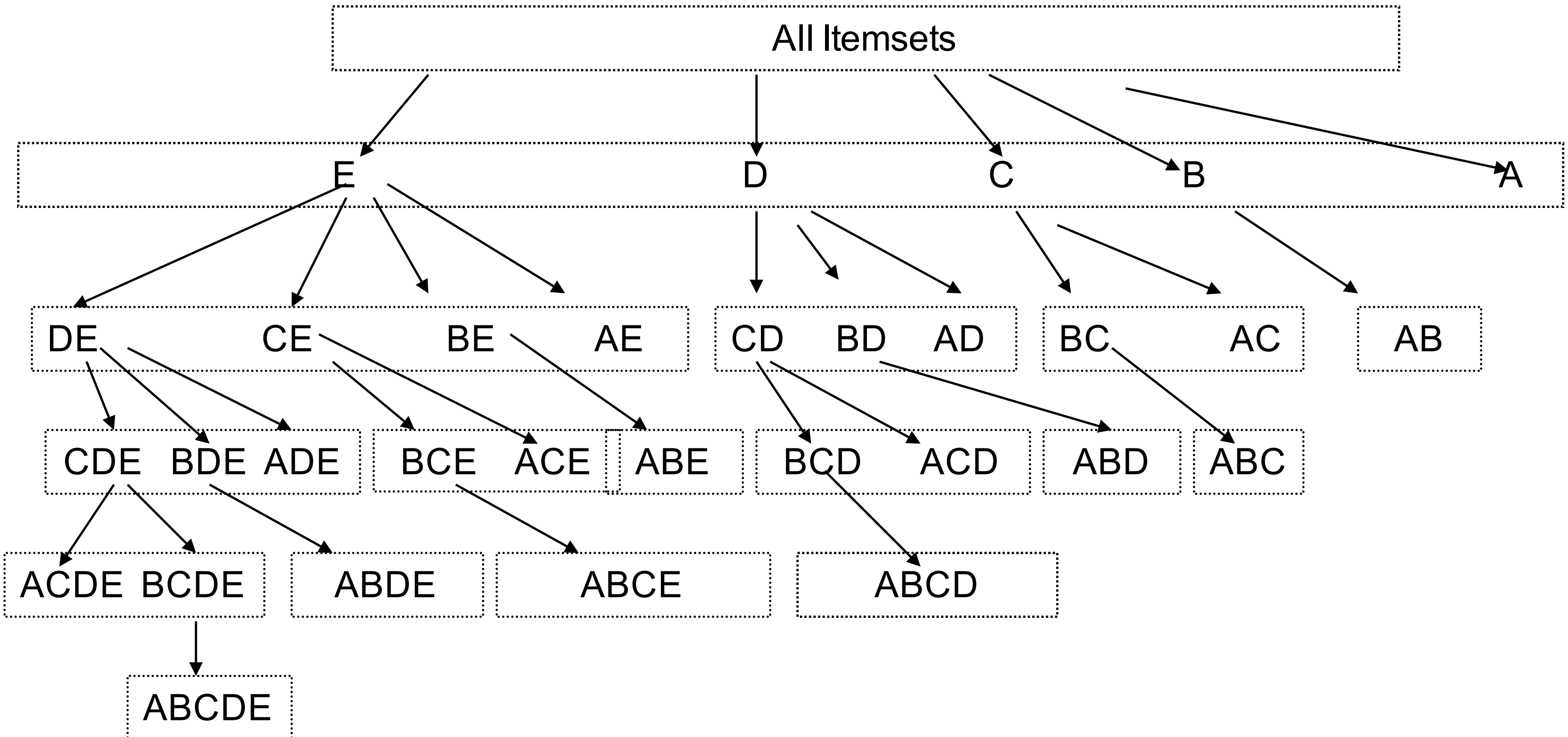
→

TID	Items
1	{B,A}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{B,A,C}
6	{B,A,C,D}
7	{B,C}
8	{B,A,C}
9	{B,A,D}
10	{B,C,E}

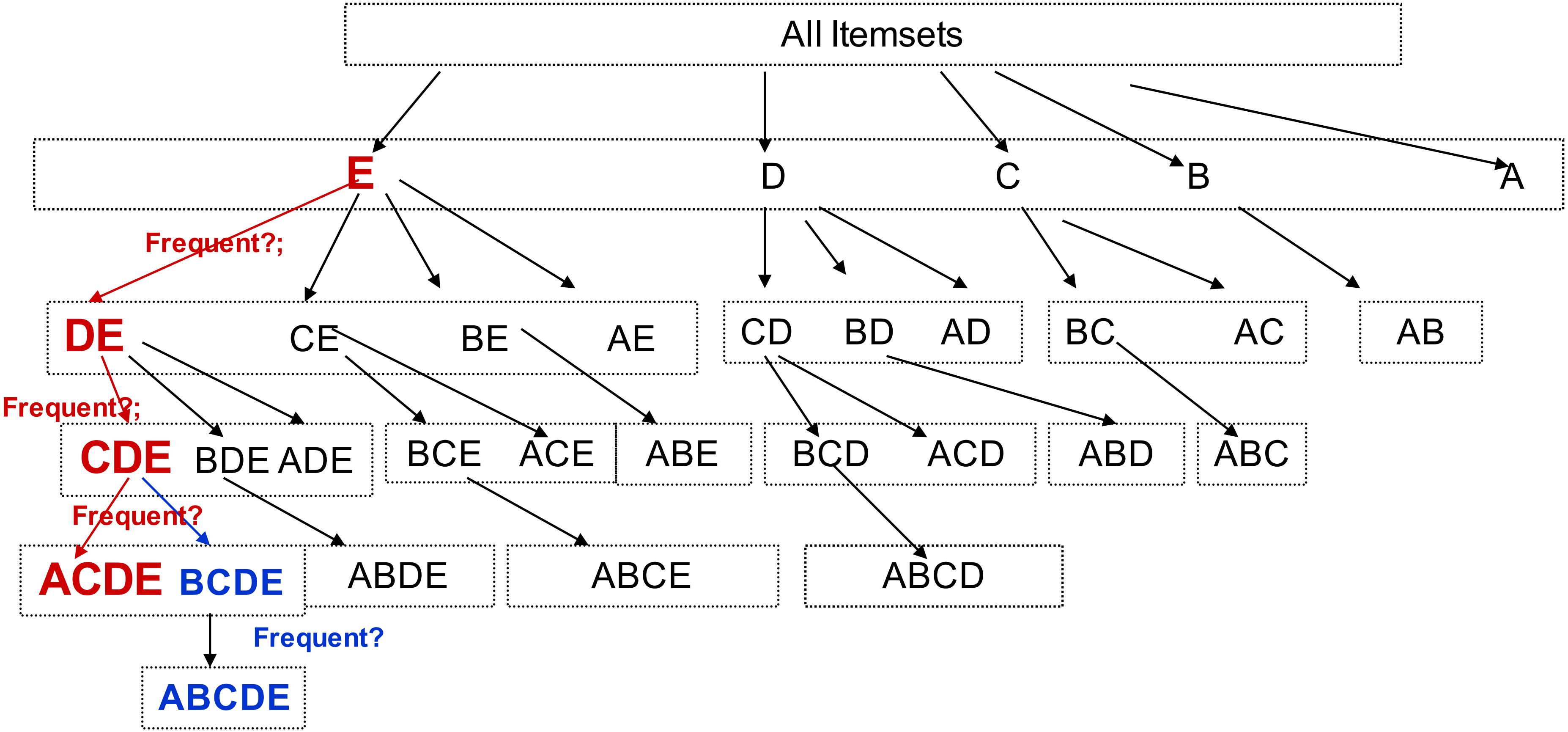
Finding Frequent Itemsets

- Input: The FP-tree
- Output: All Frequent Itemsets and their support
- Method:
  - Divide and Conquer:
  - Consider all itemsets that end in: E, D, C, B, A
    - For each possible ending item, consider the itemsets with last items one of items preceding it in the ordering
    - E.g, for E, consider all itemsets with last item D, C, B, A. This way we get all the itesets ending at DE, CE, BE, AE
    - Proceed recursively this way.
    - Do this for all items.

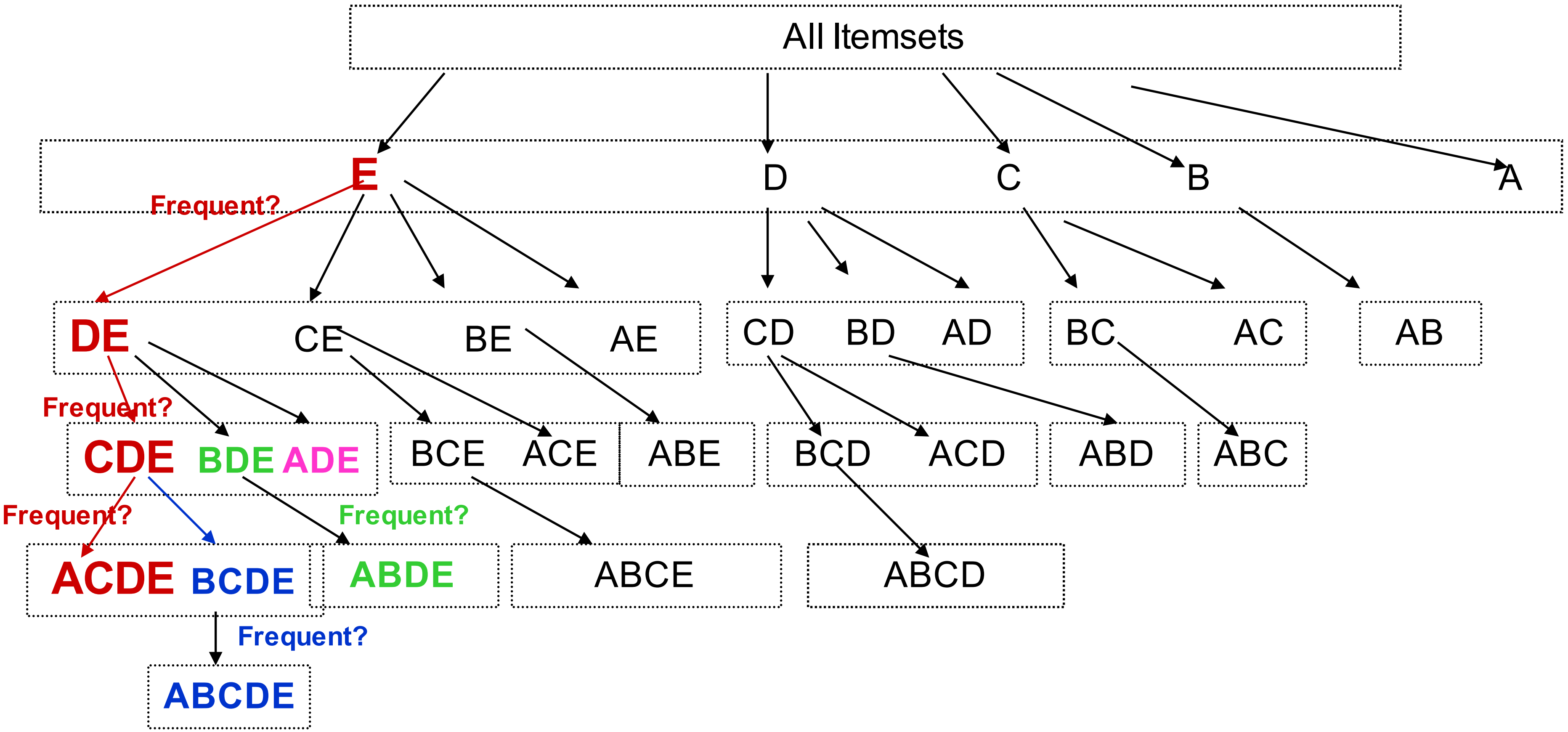
Frequent itemsets



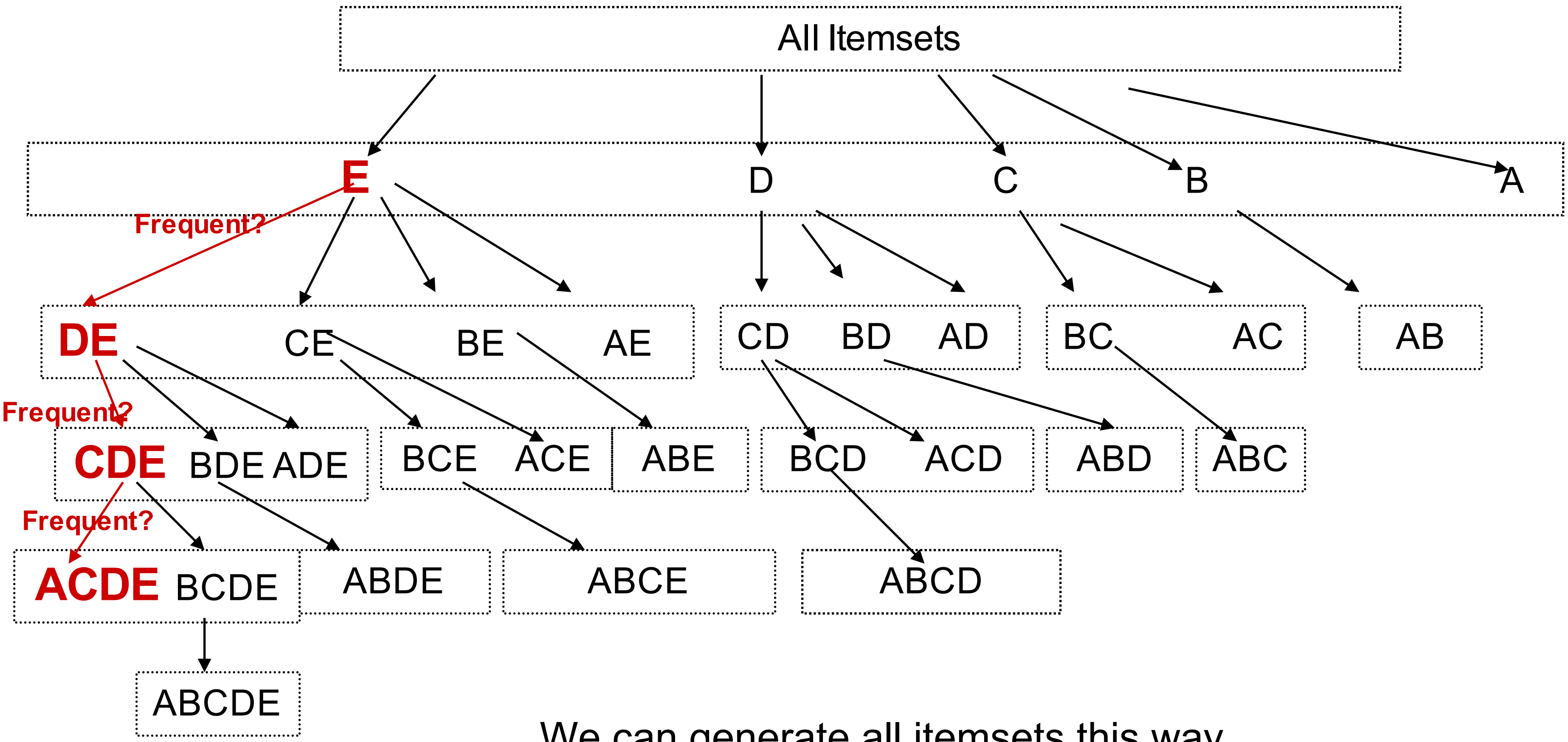
Frequent Itemsets



Frequent Itemsets



Frequent Itemsets



We can generate all itemsets this way
We expect the FP-tree to contain a lot less

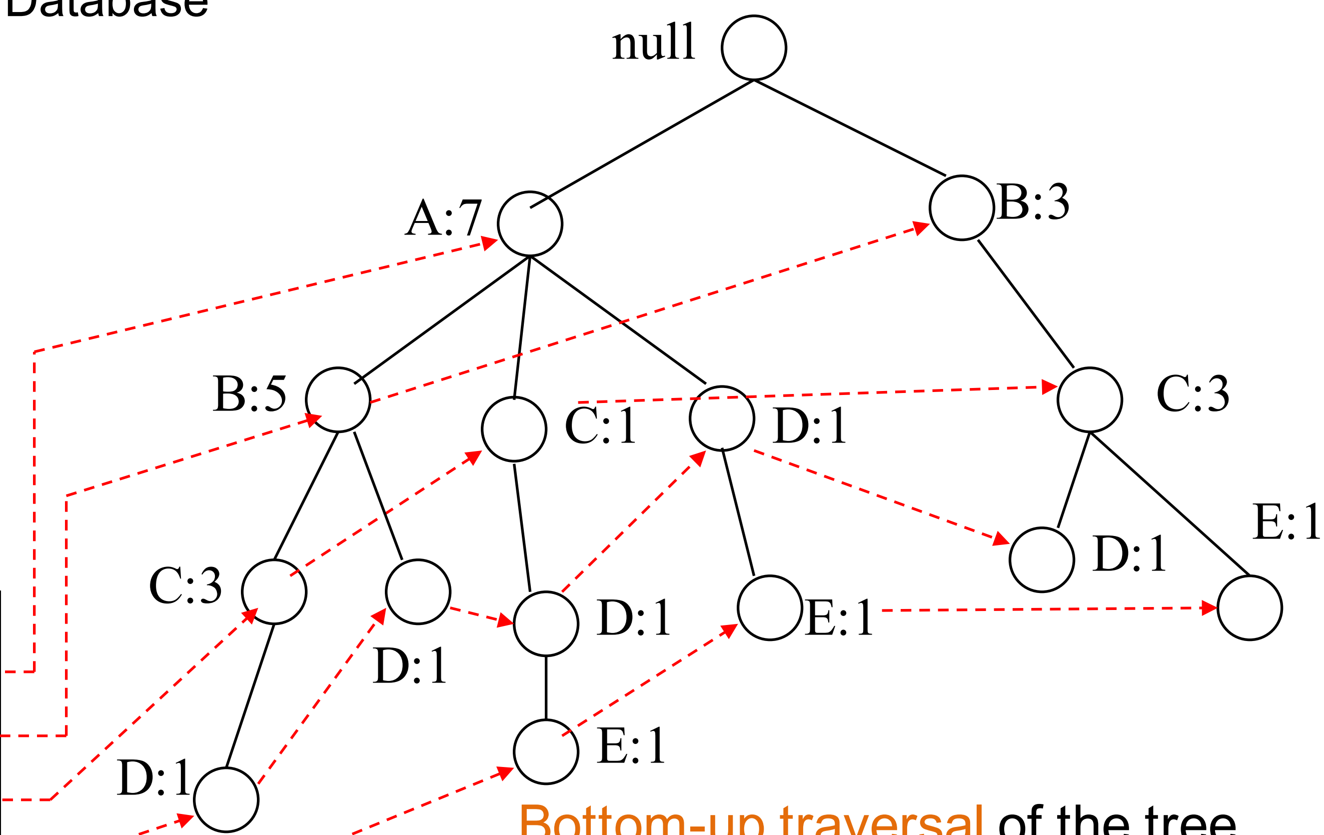
Using the FP-tree to find frequent itemsets

TID	Items
1	{A,B}
2	{B,C,D}
3	{A,C,D,E}
4	{A,D,E}
5	{A,B,C}
6	{A,B,C,D}
7	{B,C}
8	{A,B,C}
9	{A,B,D}
10	{B,C,E}

Transaction Database

Header table

Item	Pointer
A	
B	
C	
D	
E	

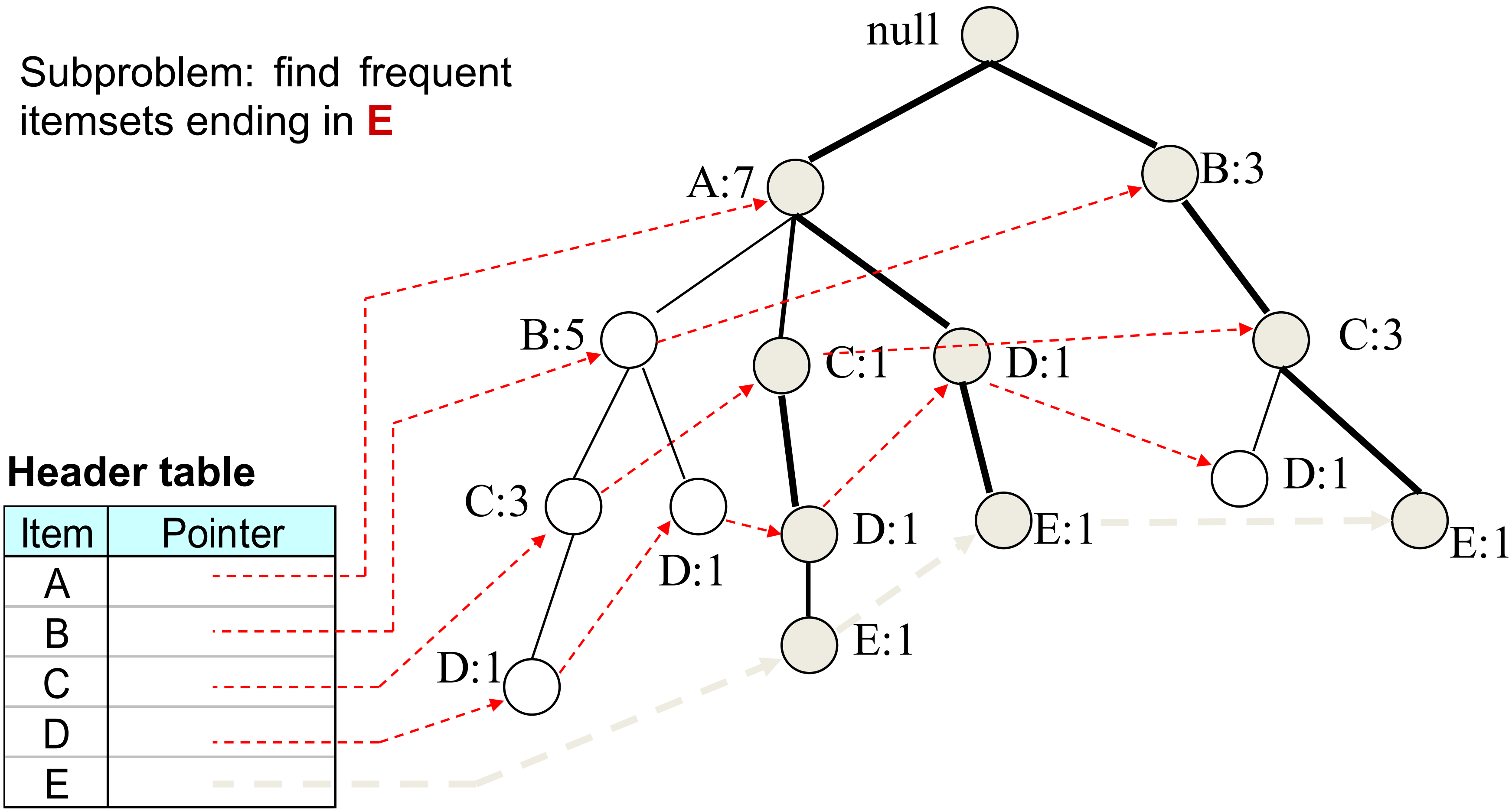


Bottom-up traversal of the tree.

First, itemsets ending in E, then D, etc, each time a suffix-based class

Finding Frequent Itemsets

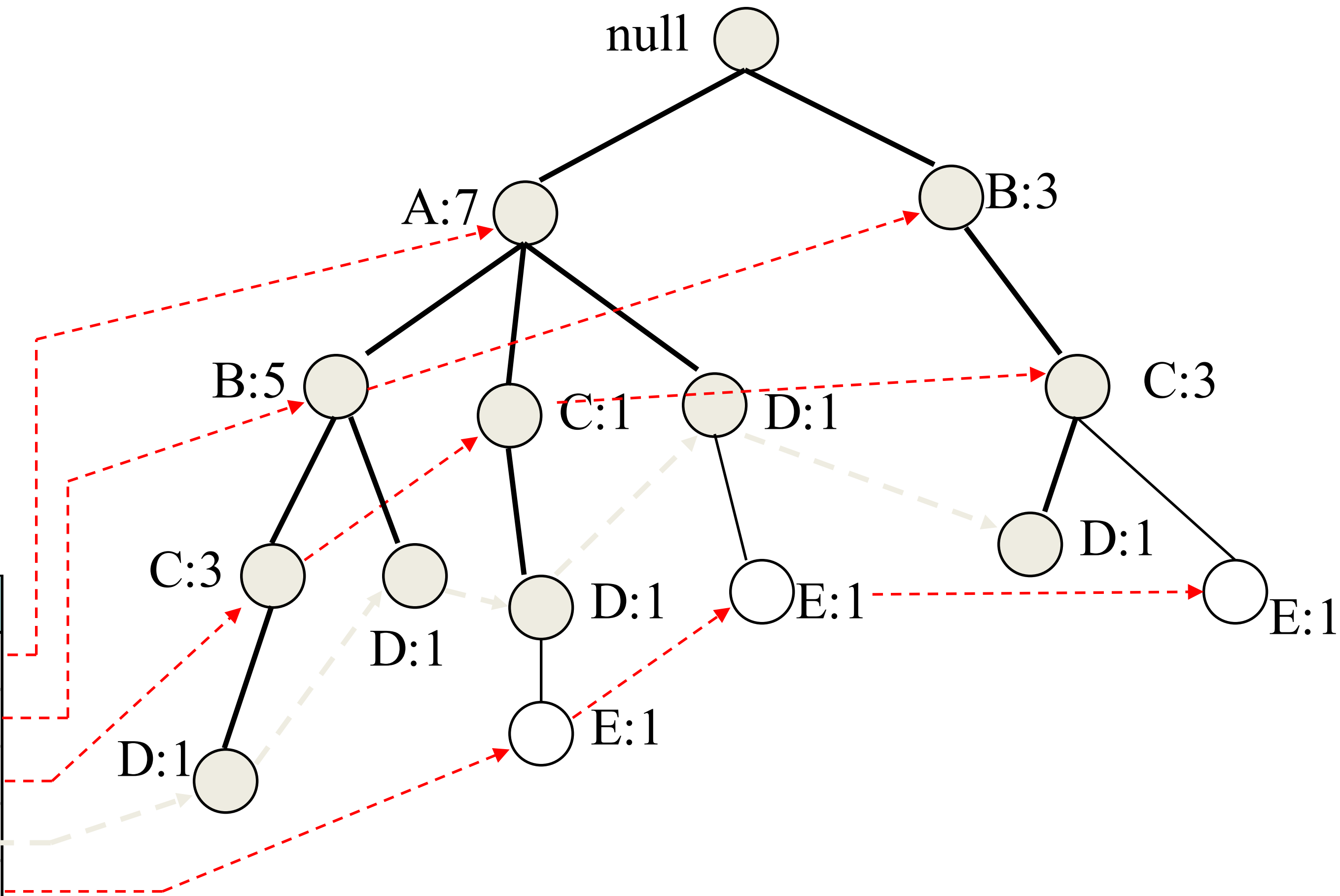
Subproblem: find frequent itemsets ending in **E**



- We will then see how to compute the support for the possible itemsets

Finding Frequent Itemsets

Ending in **D**

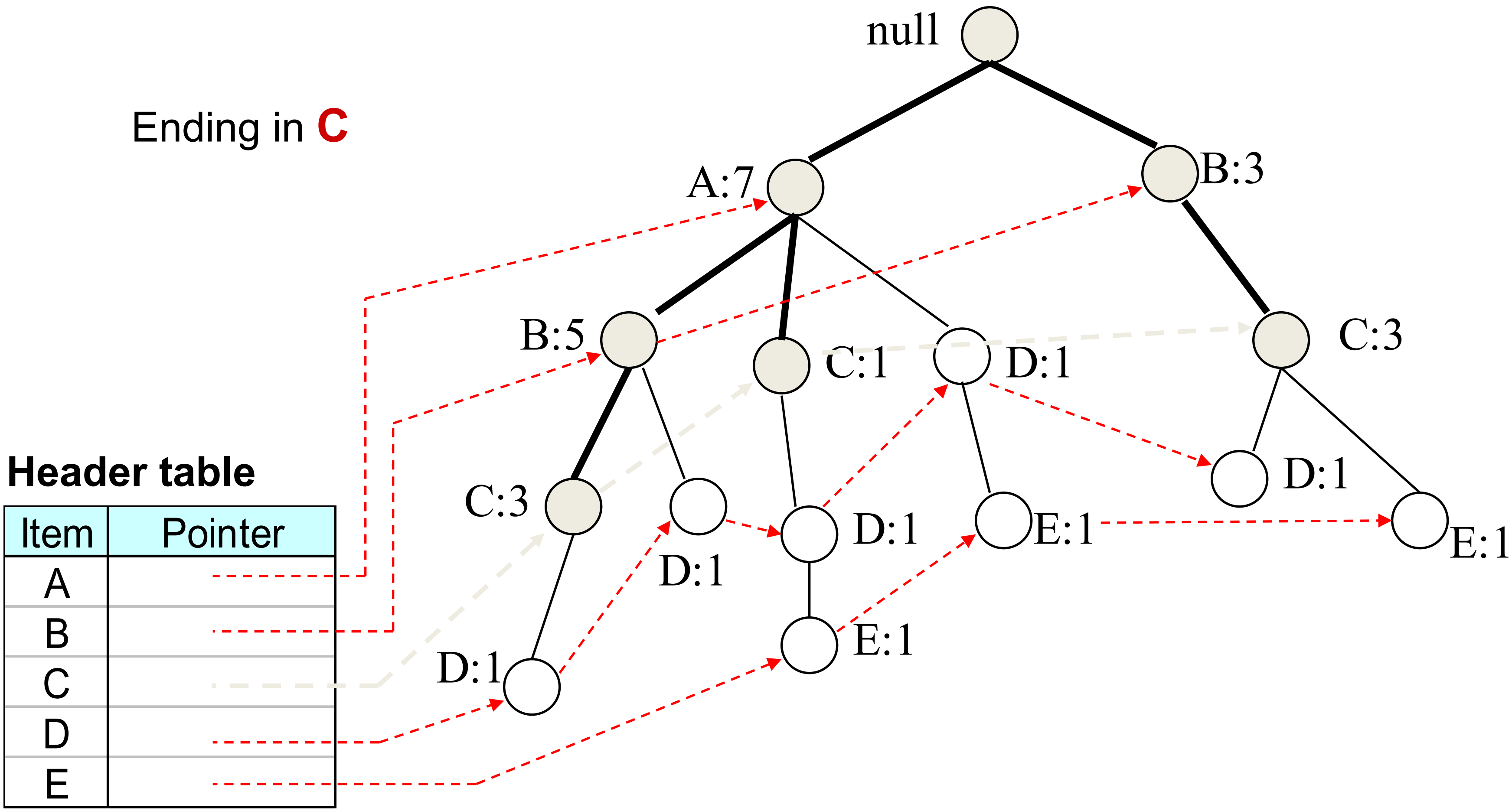


Header table

Item	Pointer
A	
B	
C	
D	
E	

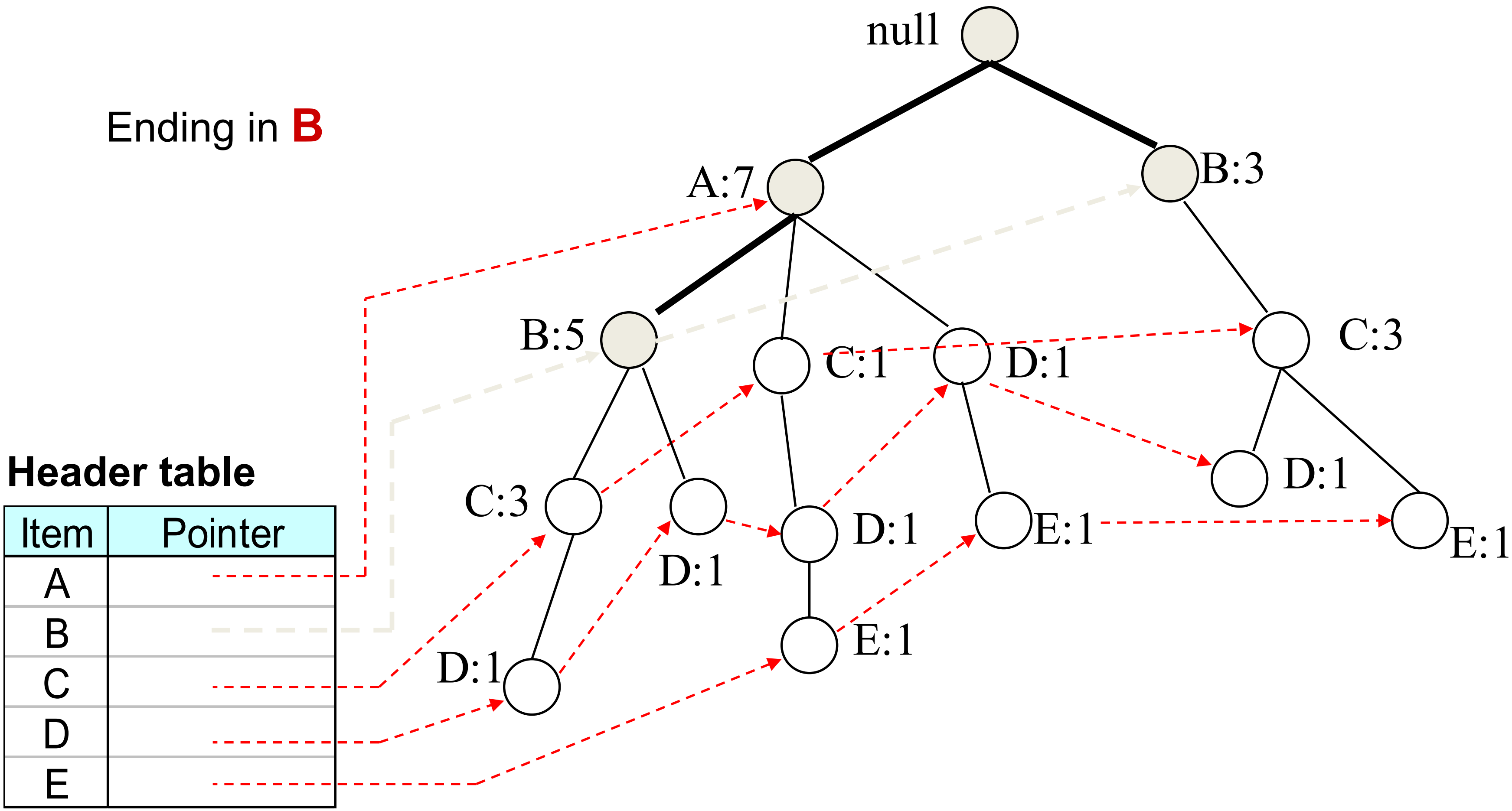
Finding Frequent Itemsets

Ending in **C**



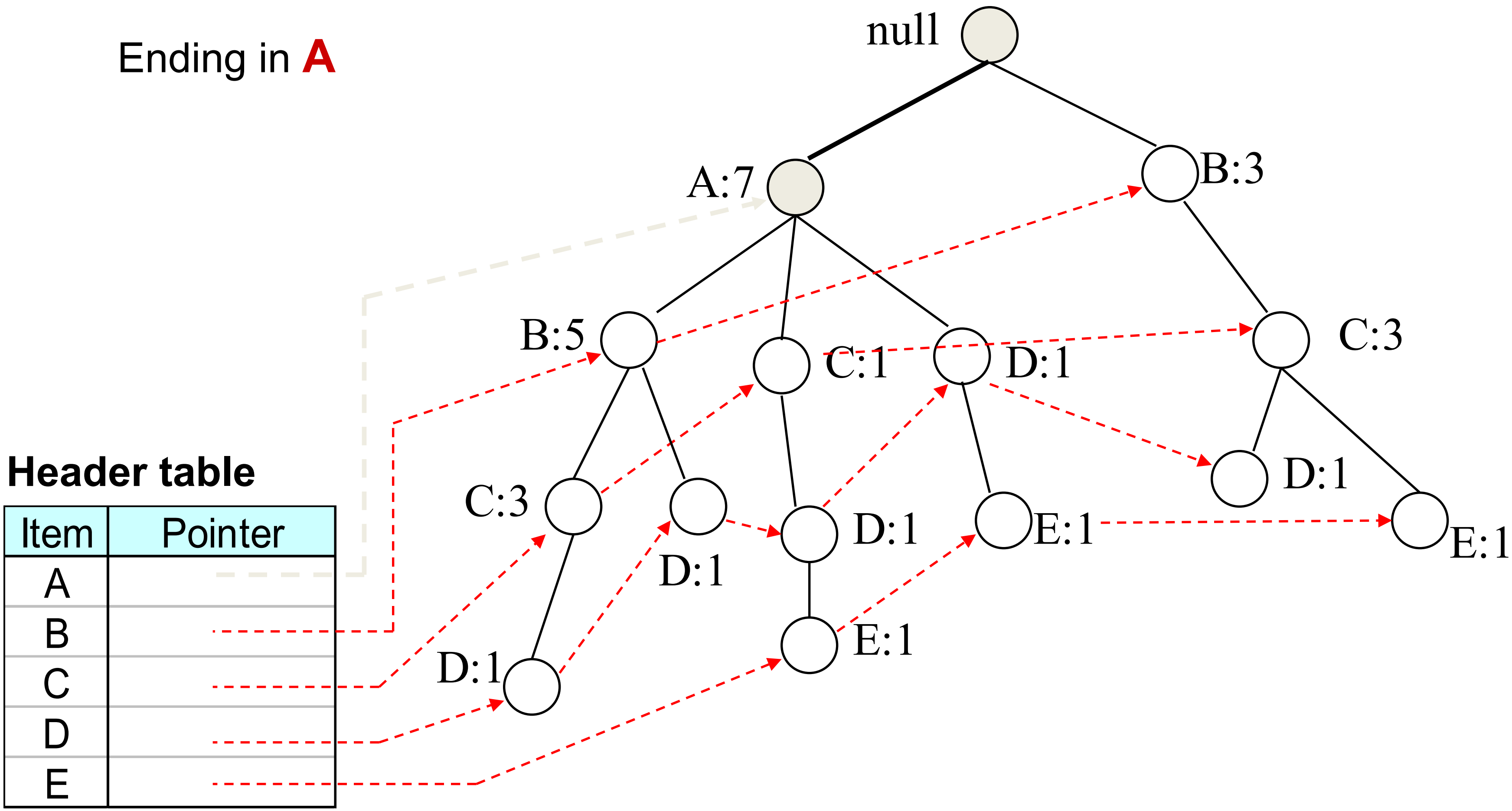
Finding Frequent Itemsets

Ending in **B**



Finding Frequent Itemsets

Ending in **A**



Algorithm

- For each suffix X
- Phase 1
  - Construct the prefix tree for X as shown before, and compute the support using the header table and the pointers

- Phase 2
  - If X is frequent, construct the conditional FP-tree for X in the following steps
    1. Recompute support
    2. Prune infrequent items
    3. Prune leaves and recurse

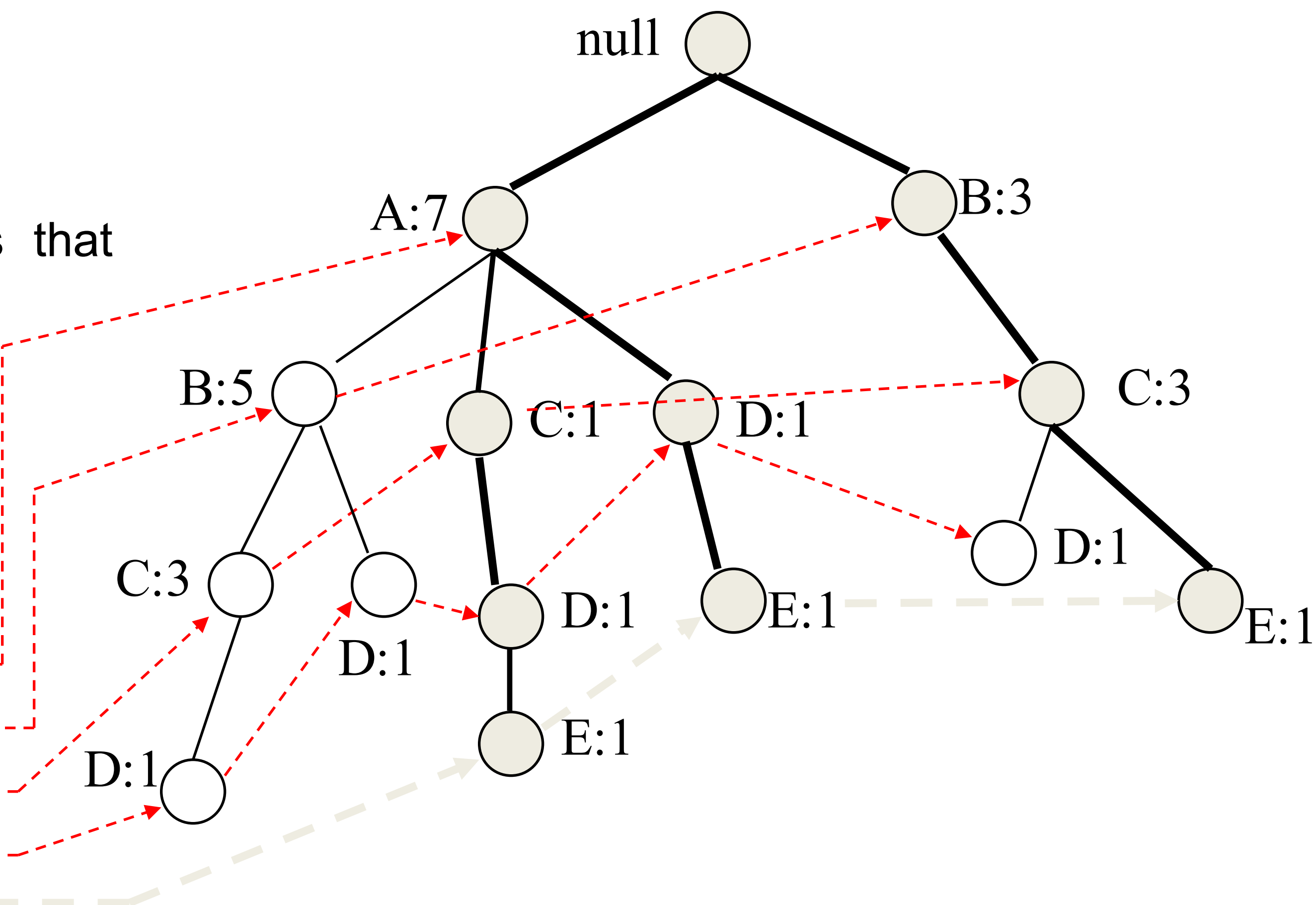
Example

Phase 1 – construct prefix tree

Find all prefix paths that contain E

Header table

Item	Pointer
A	
B	
C	
D	
E	



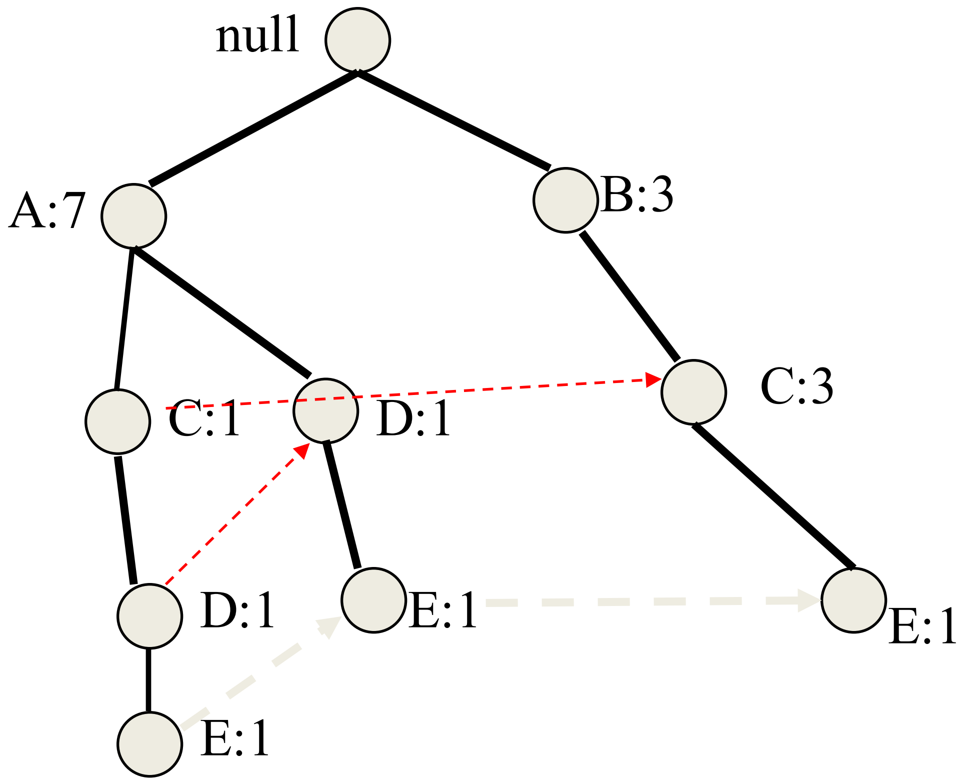
Suffix Paths for E:

{A,C,D,E}, {A,D,E}, {B,C,E}

Example

Phase 1 – construct prefix tree

Find all prefix paths that contain E



Prefix Paths for E:

{A,C,D,E}, {A,D,E}, {B,C,E}

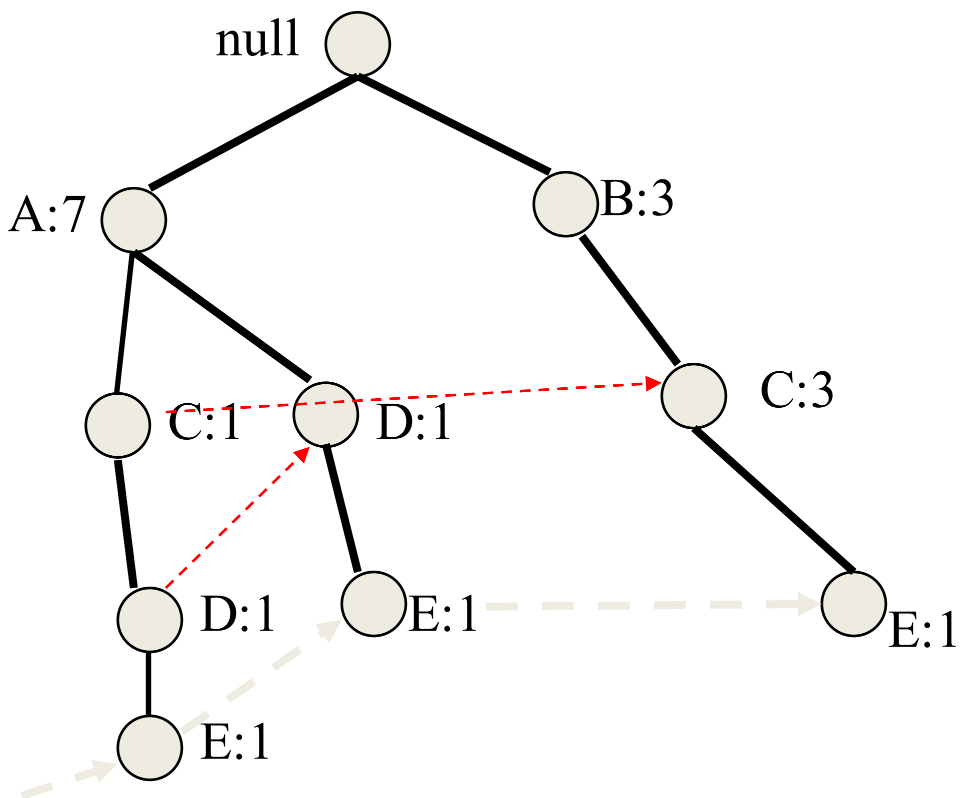
Example

Compute Support for E (minsup = 2)

How?

Follow pointers while summing up counts: 1+1+1 = 3 > 2

E is frequent



{E} is frequent so we can now consider suffixes DE, CE, BE, AE

Example

E is frequent so we proceed with Phase 2

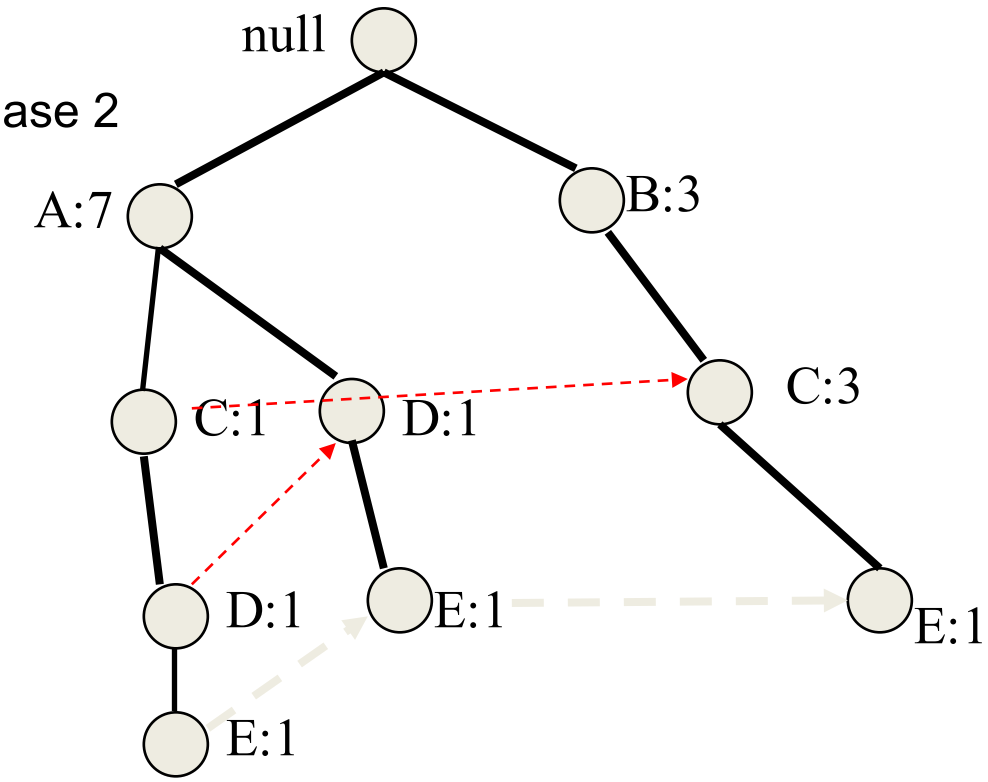
Phase 2

Convert the prefix tree of E into a conditional FP-tree

Two changes

(1) Recompute support

(2) Prune infrequent



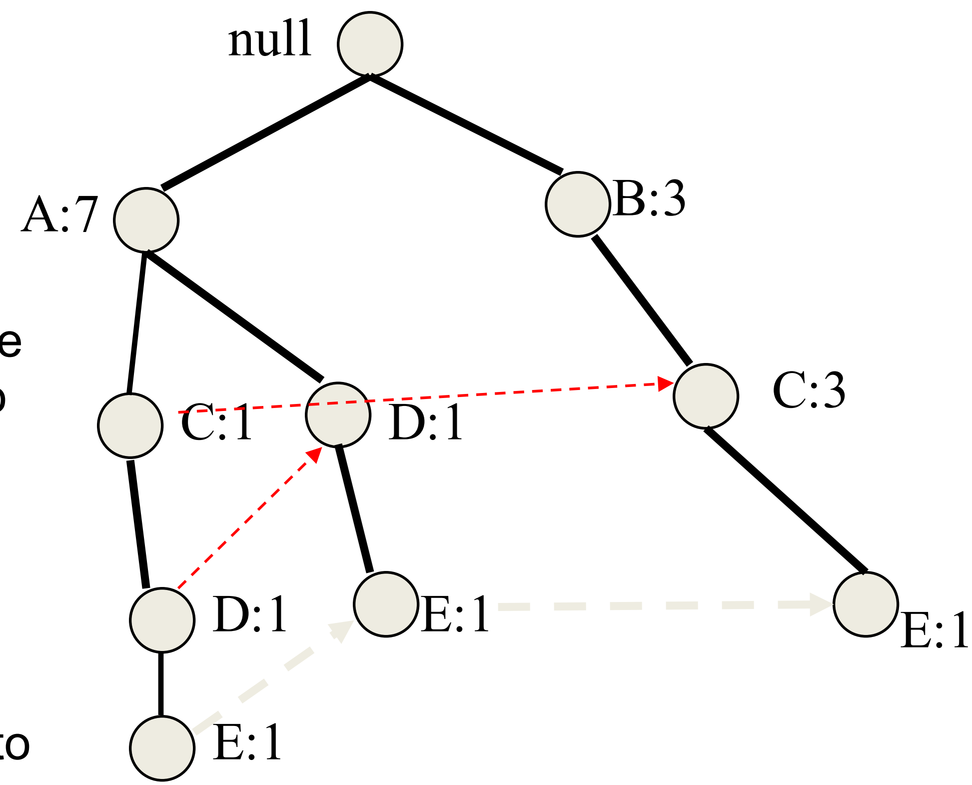
Example

Recompute Support

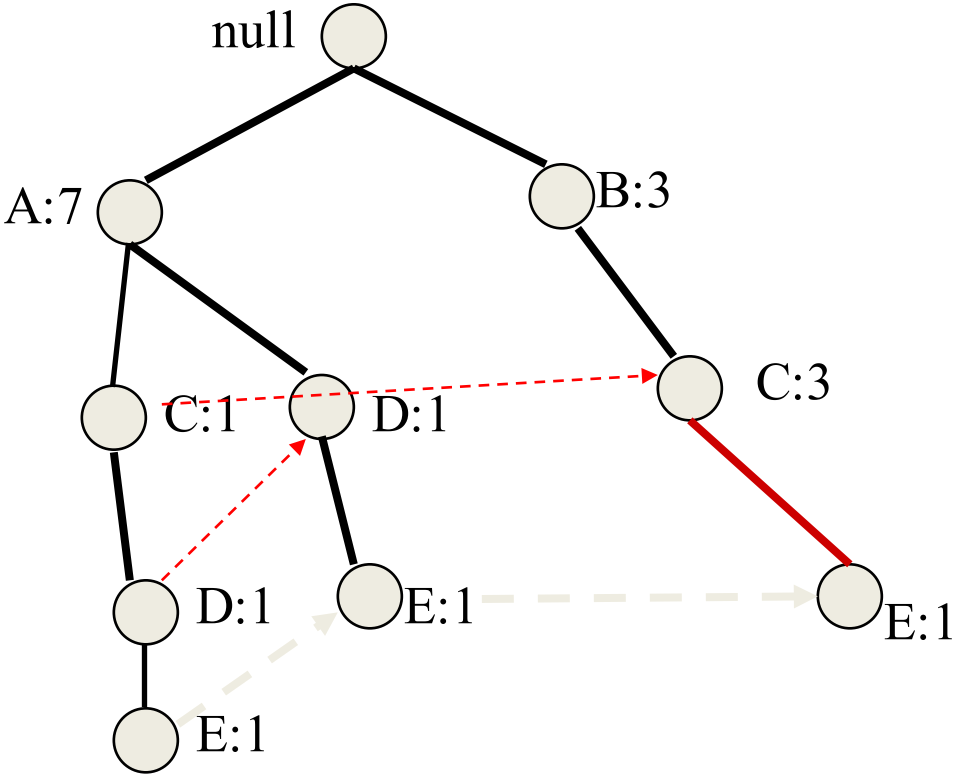
The support counts for some of the nodes include transactions that do not end in E

For example in null->B->C->E we count {B, C}

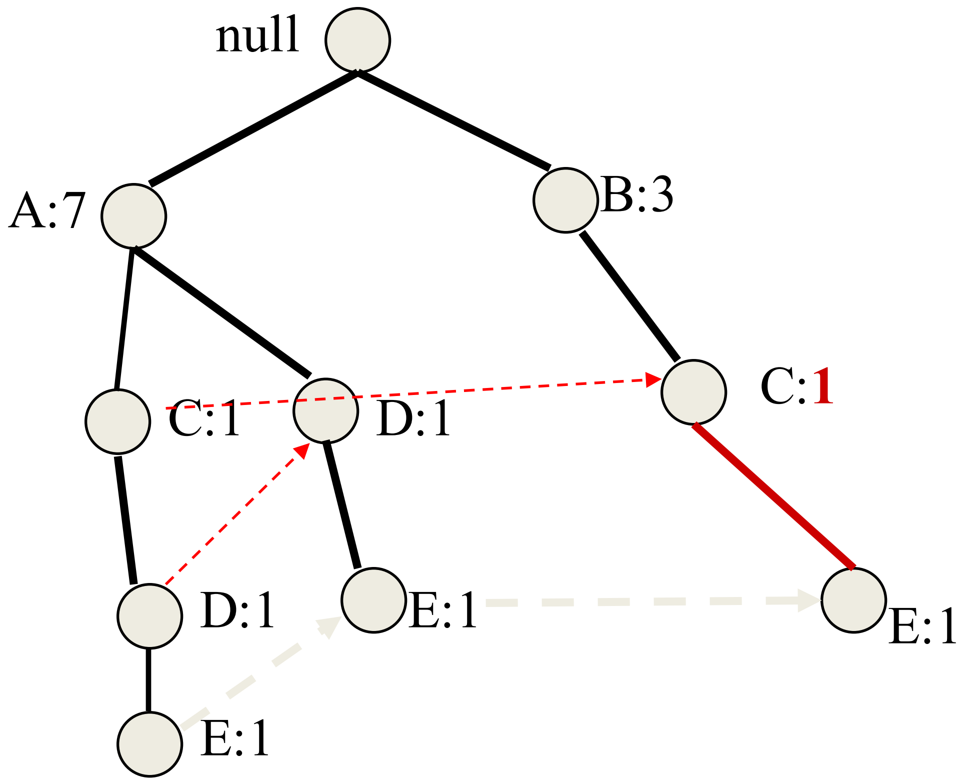
The support of any node is equal to the sum of the support of leaves with label E in its subtree



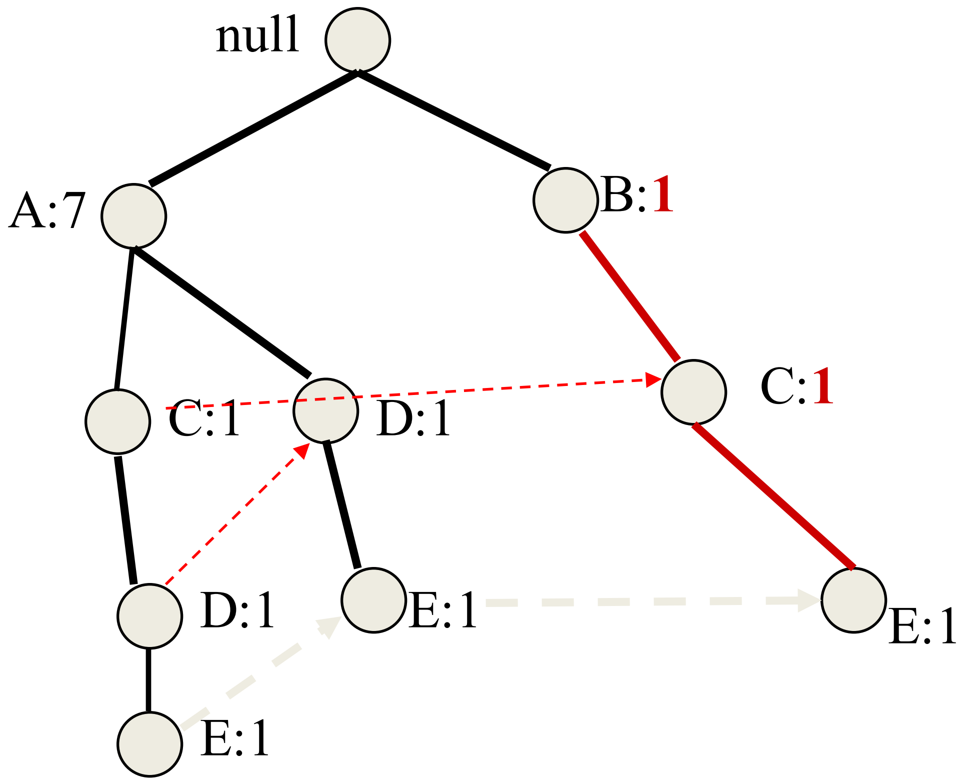
Example



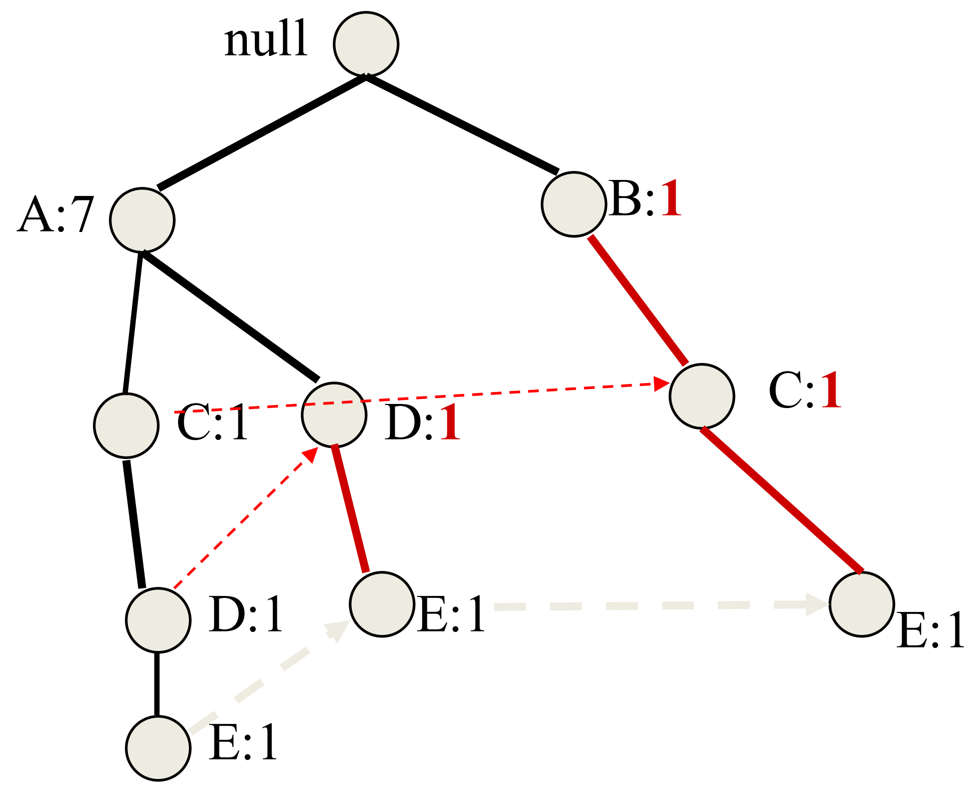
Example



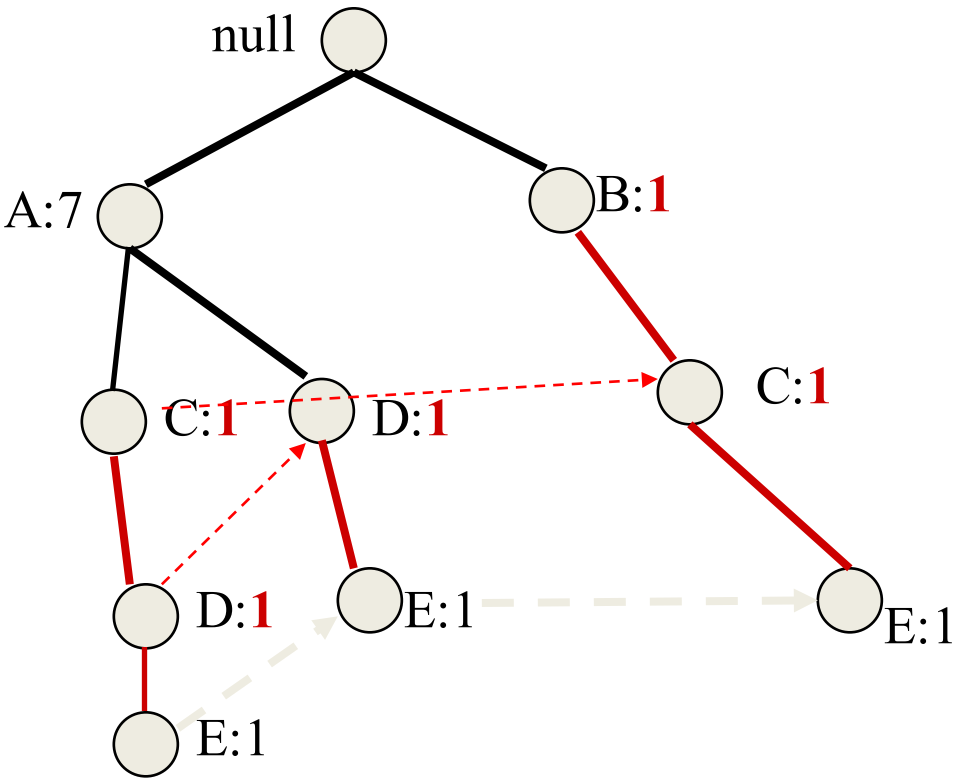
Example



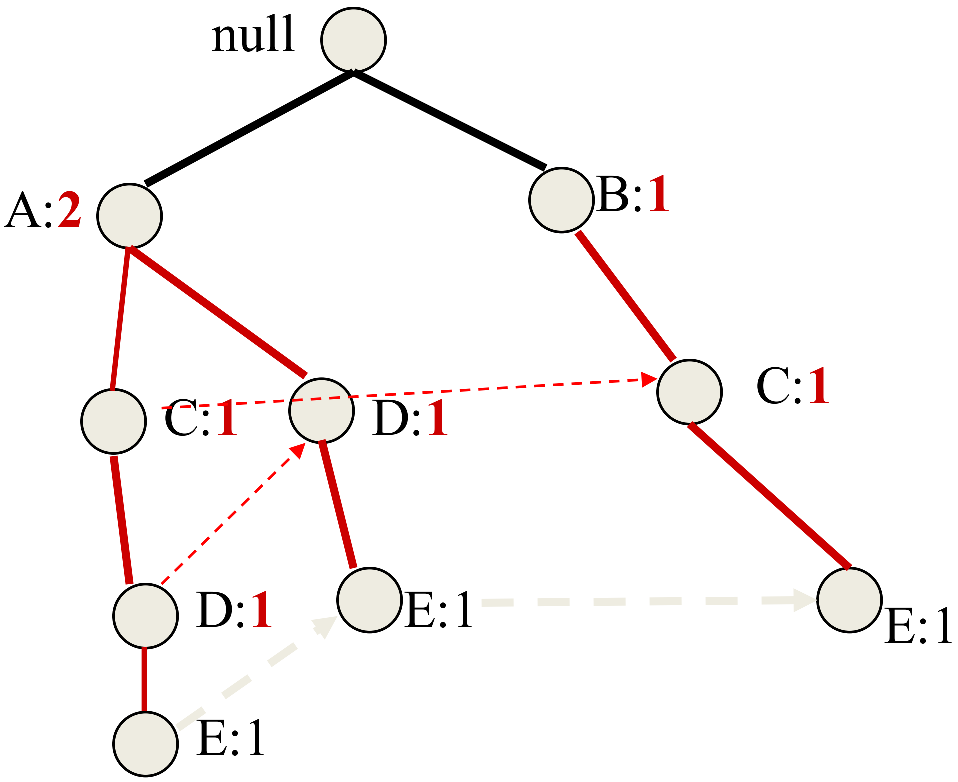
Example



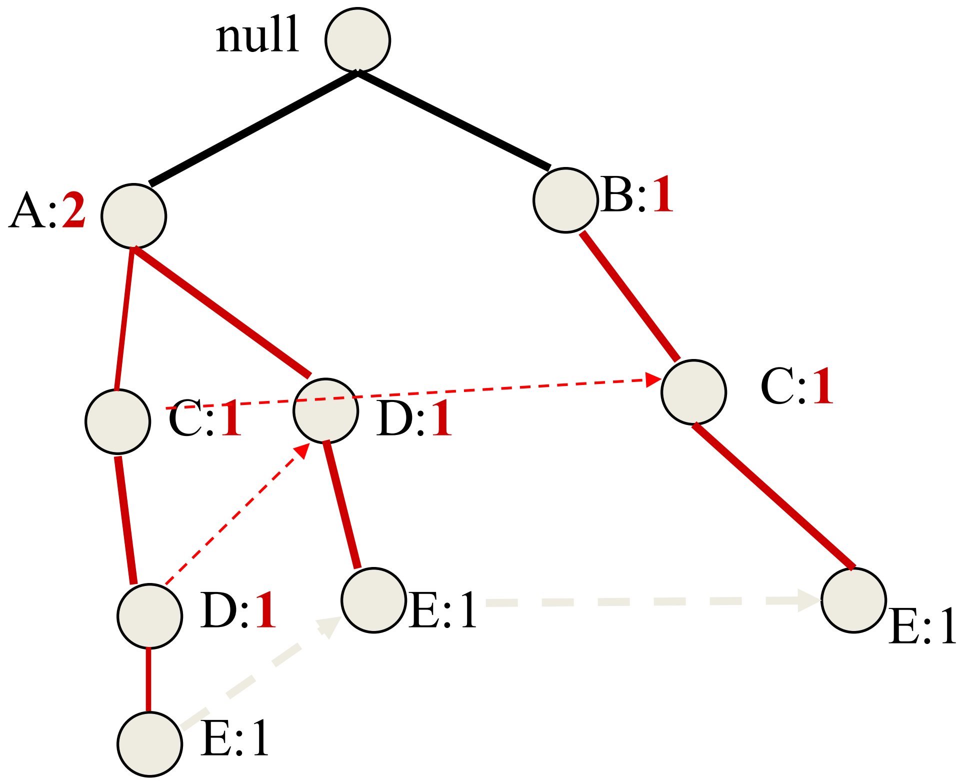
Example



Example



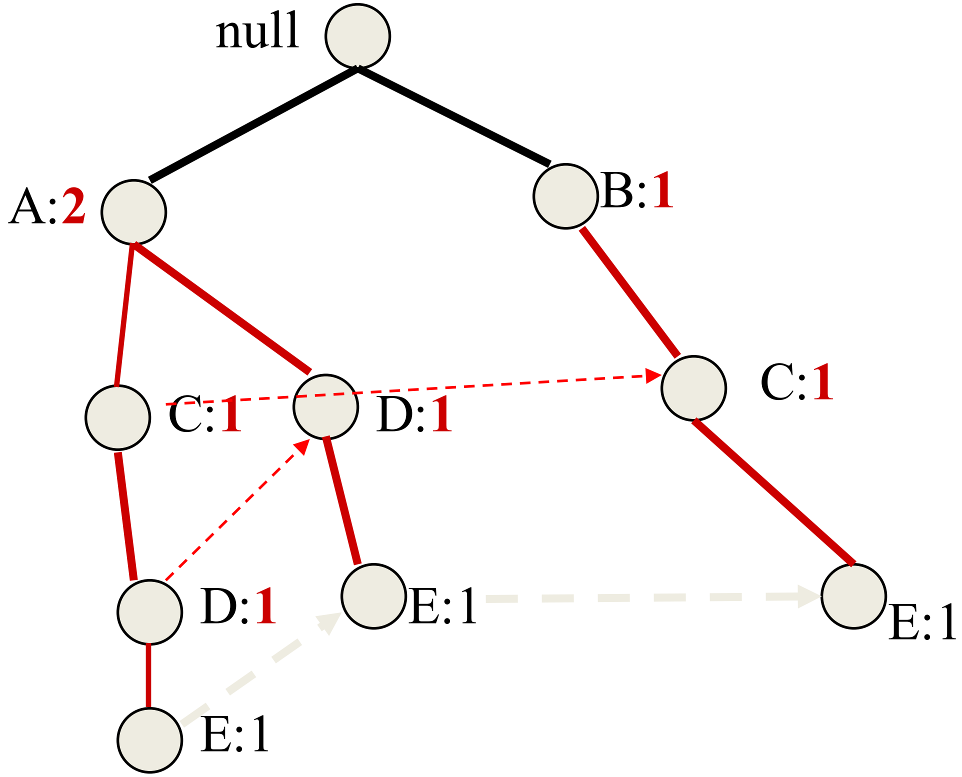
Example



Example

Truncate

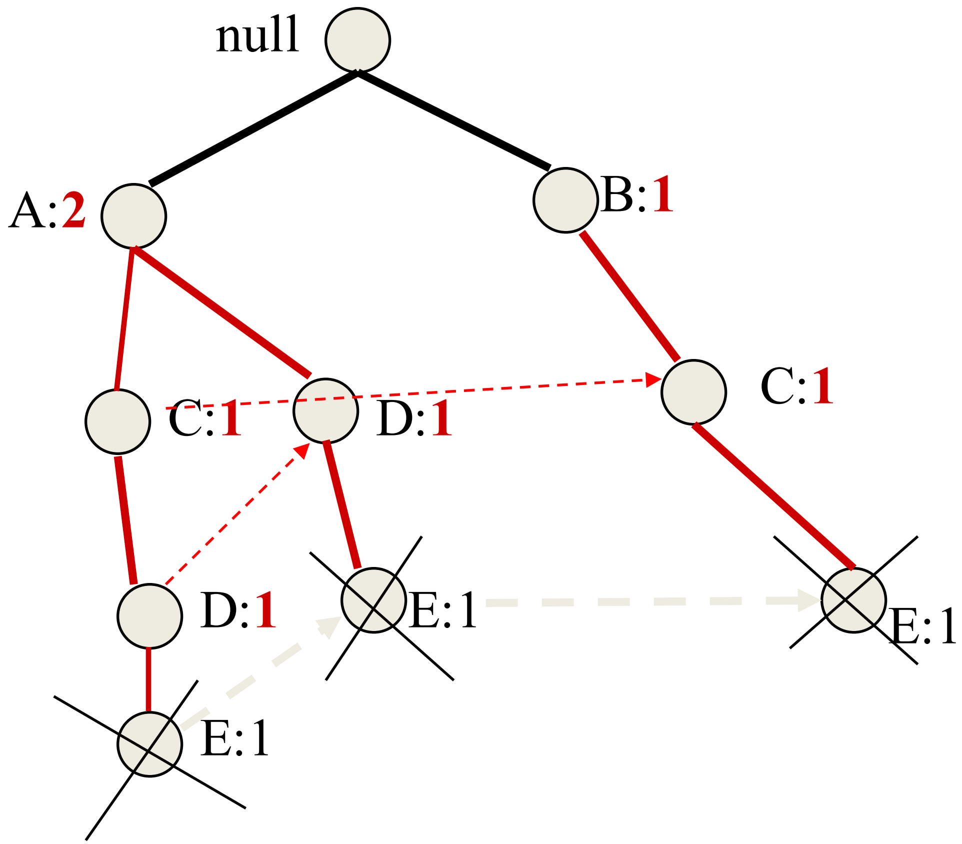
Delete the nodes of E



Example

Truncate

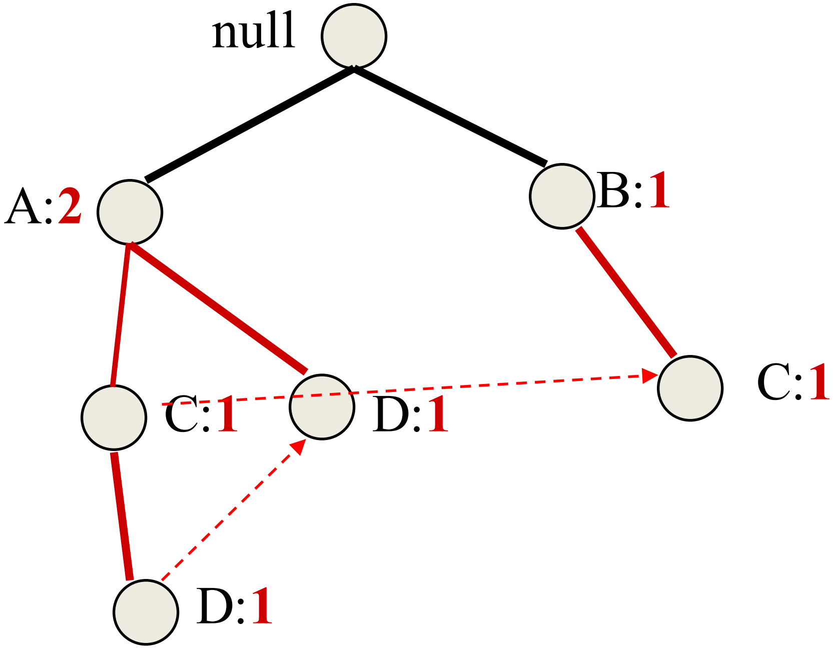
Delete the nodes of E



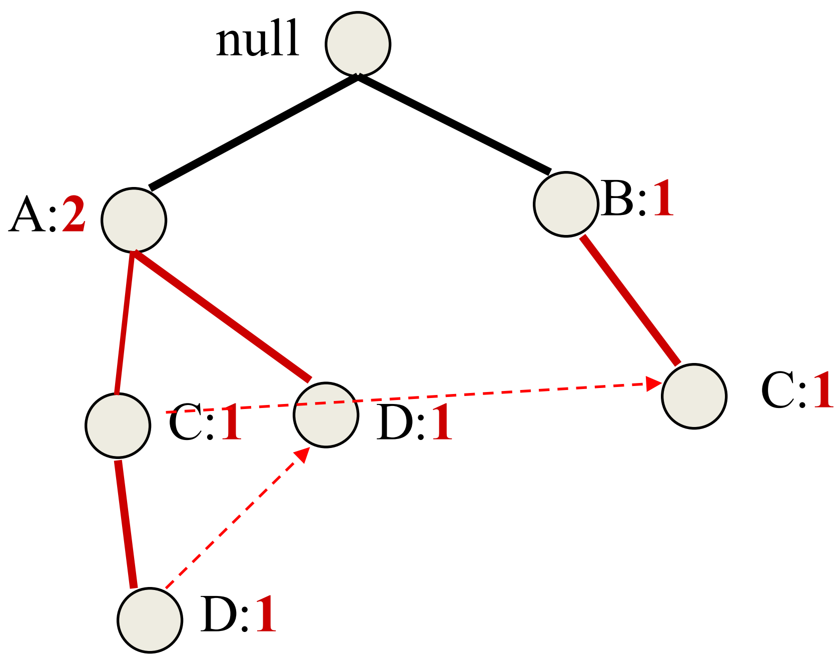
Example

Truncate

Delete the nodes of E



Example



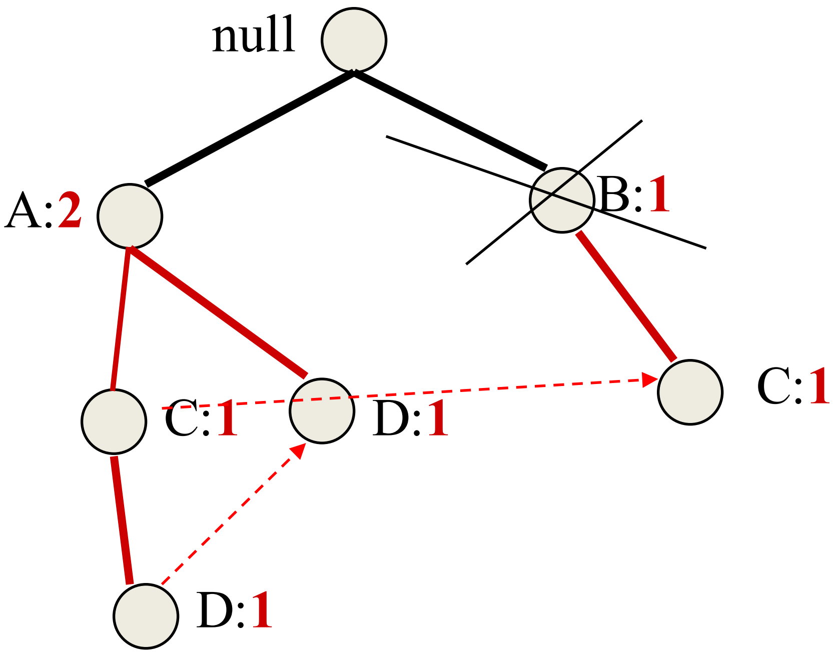
Prune infrequent

In the conditional FP-tree some nodes may have support less than minsup

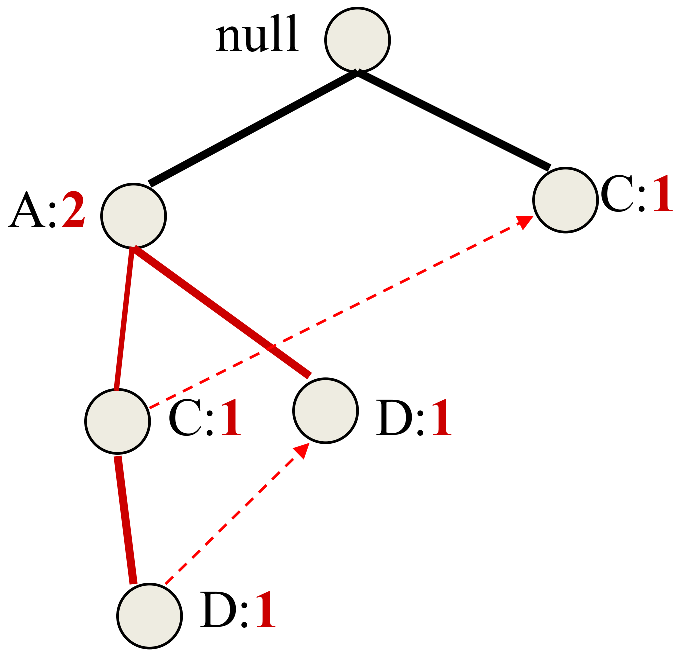
e.g., B needs to be pruned

This means that B appears with E less than minsup times

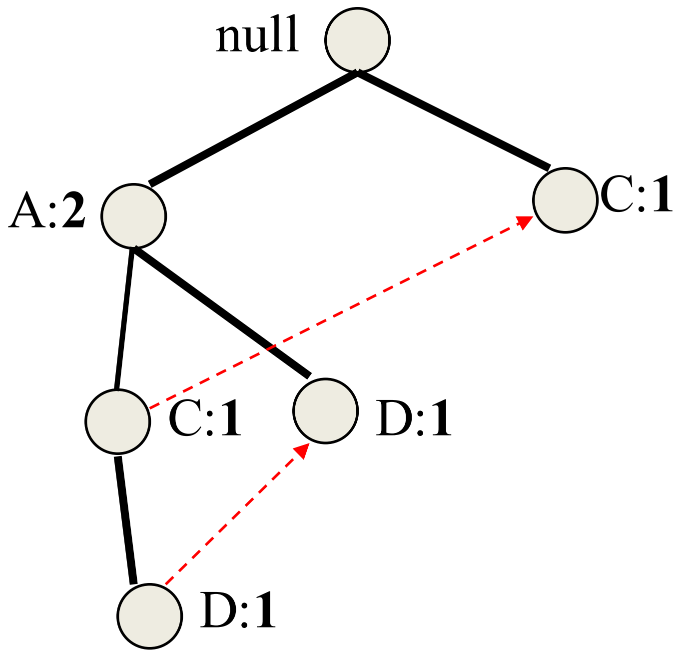
Example



Example



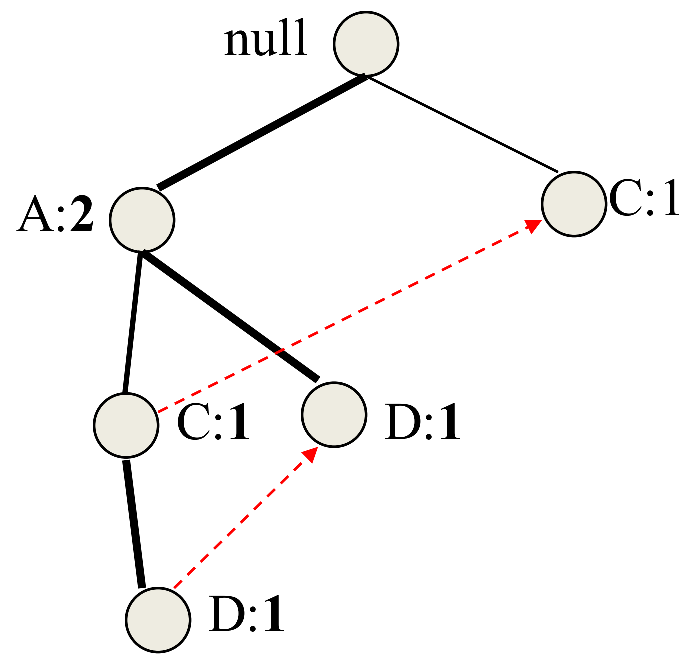
Example



The conditional FP-tree for E

Repeat the algorithm for {D, E}, {C, E}, {A, E}

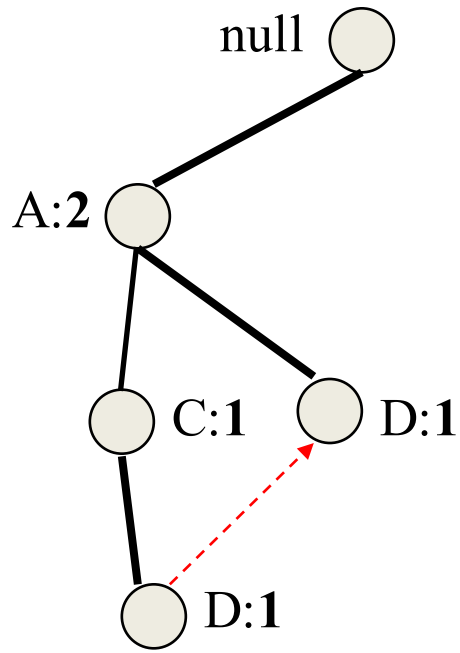
Example



Phase 1

Find all prefix paths that contain D (DE) in the conditional FP-tree

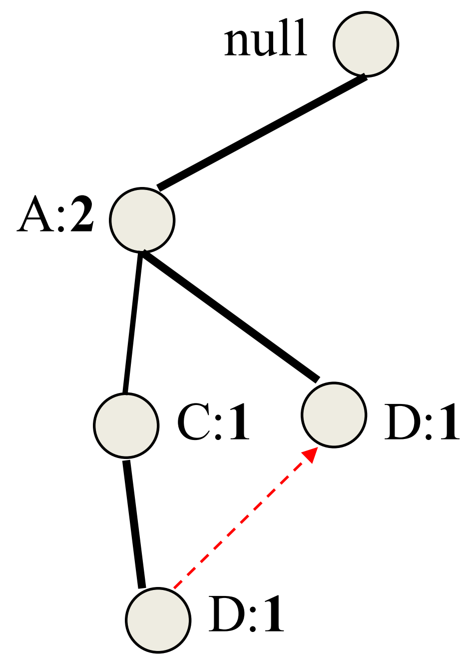
Example



Phase 1

Find all prefix paths that contain D (DE) in the conditional FP-tree

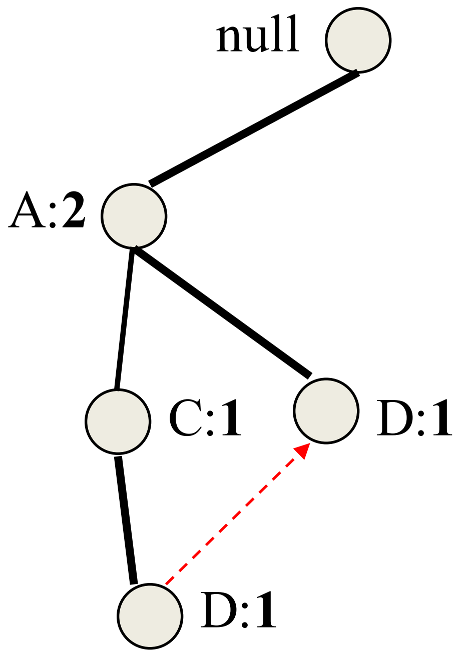
Example



Compute the support of {D,E} by following the pointers in the tree
$1+1 = 2 \geq 2 = minsup$

{D,E} is frequent

Example



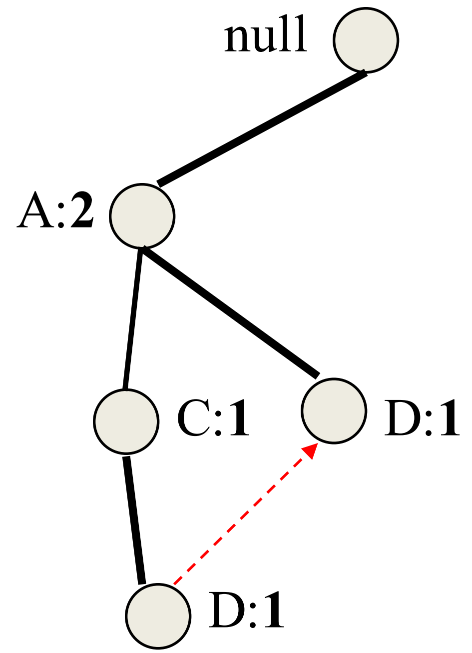
Phase 2

Construct the conditional FP-tree

1. Recompute Support
2. Prune nodes

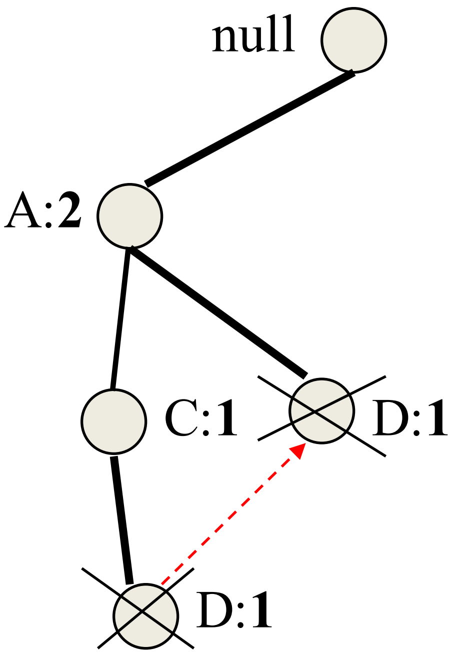
Example

Recompute support

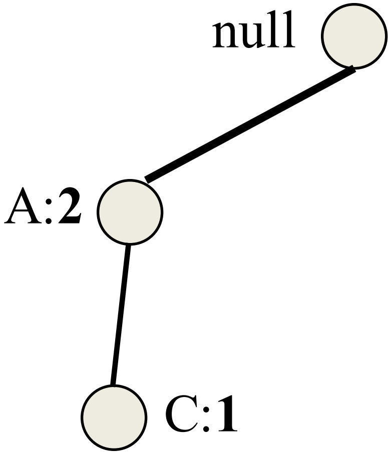


Example

Prune nodes



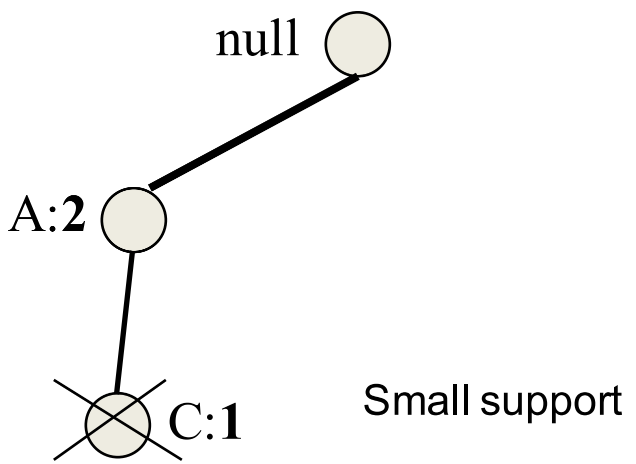
Example



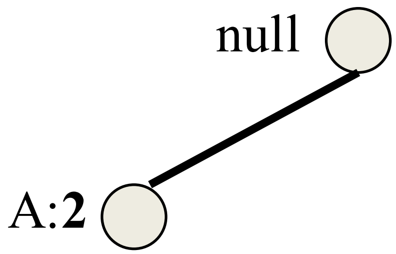
Prune nodes

Example

Prune nodes



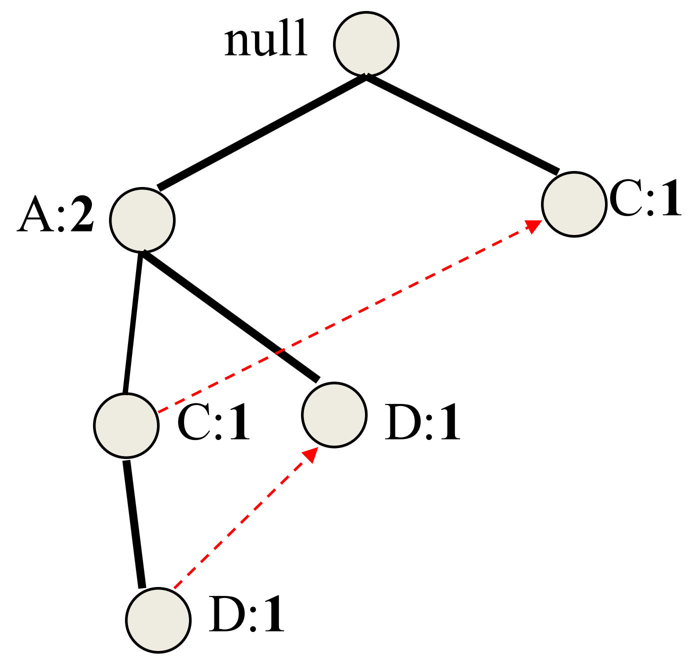
Example



Final condition FP-tree for {D,E}

The support of A is ≥ minsup so {A,D,E} is frequent
Since the tree has a single node we return to the next subproblem

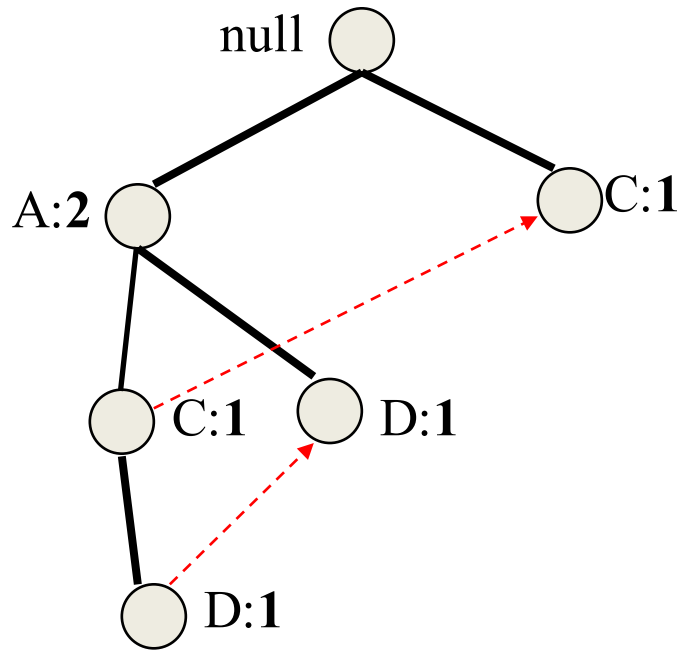
Example



The conditional FP-tree for E

We repeat the algorithm for ~~{D,E},~~ {C,E}, {A,E}

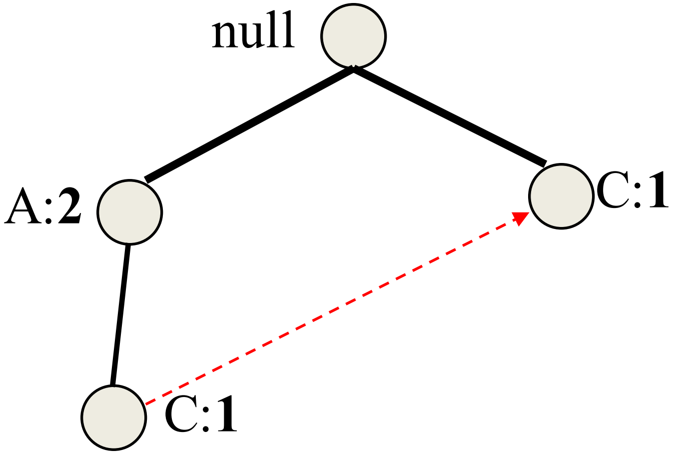
Example



Phase 1

Find all prefix paths that contain C (CE) in the conditional FP-tree

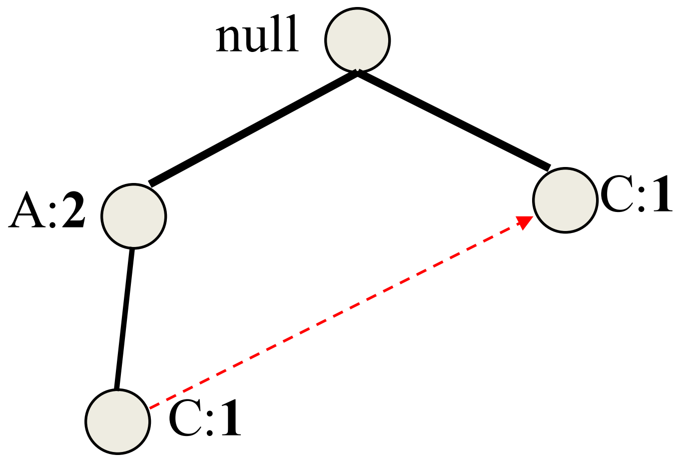
Example



Phase 1

Find all prefix paths that contain C (CE) in the conditional FP-tree

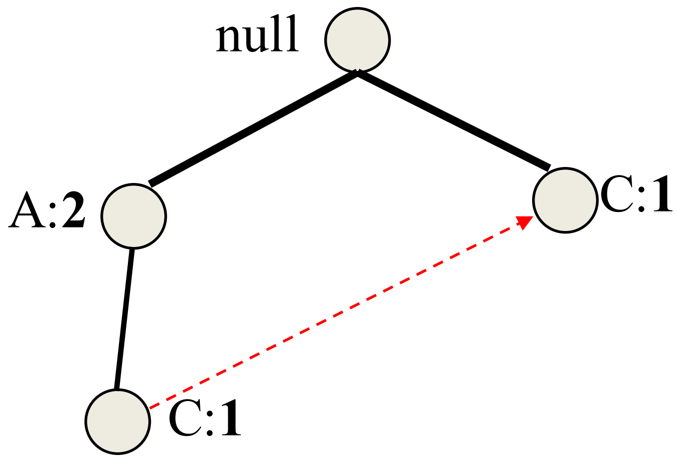
Example



Compute the support of {C,E} by following the pointers in the tree
$1+1 = 2 \geq 2 = minsup$

{C,E} is frequent

Example

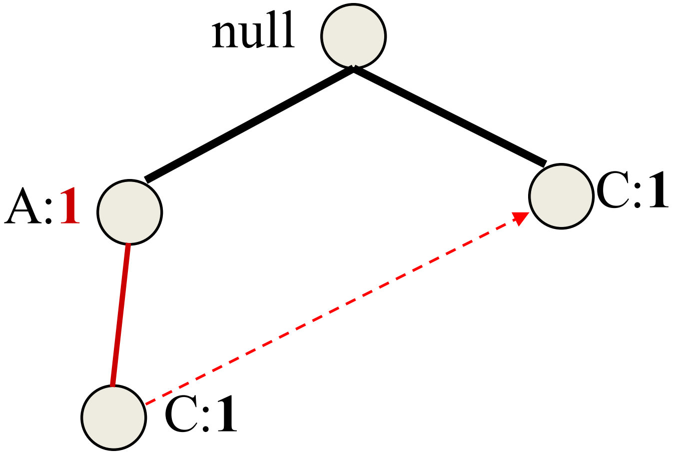


Phase 2

Construct the conditional FP-tree

1. Recompute Support
2. Prune nodes

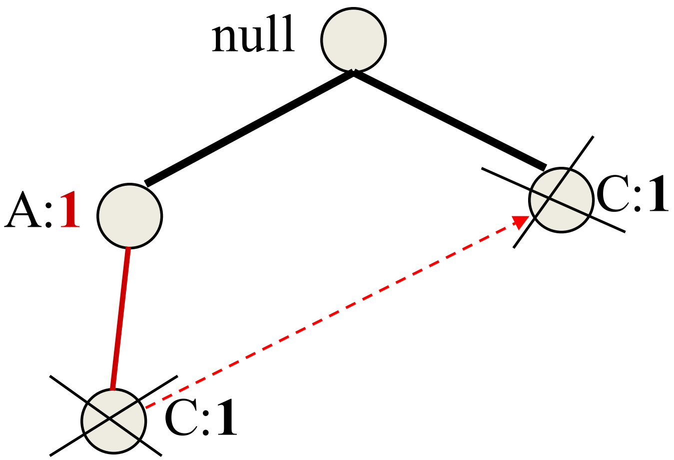
Example



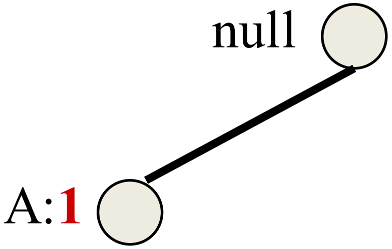
Recompute support

Example

Prune nodes

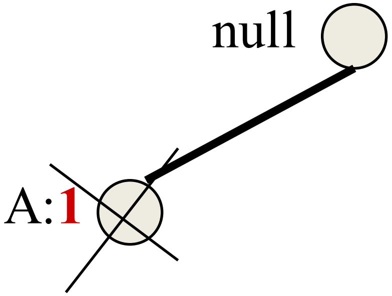


Example



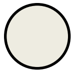
Prune nodes

Example



Prune nodes

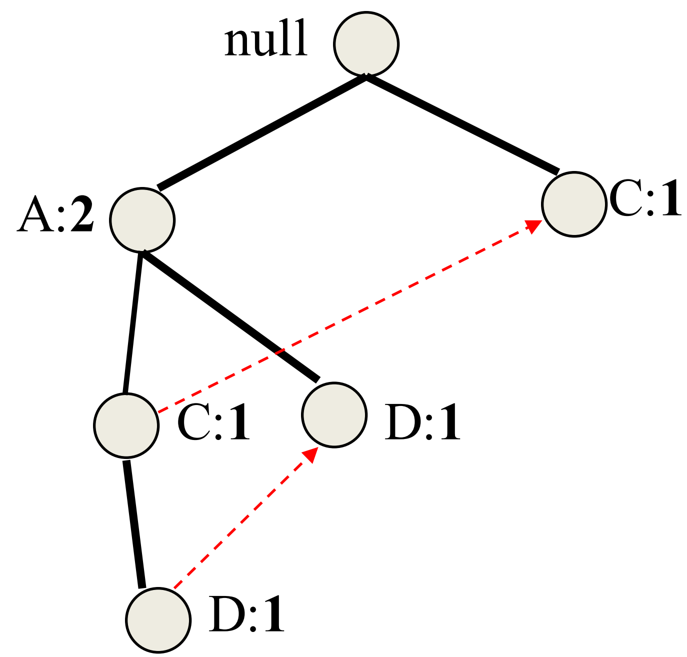
Example

null 

Prune nodes

Return to the previous subproblem

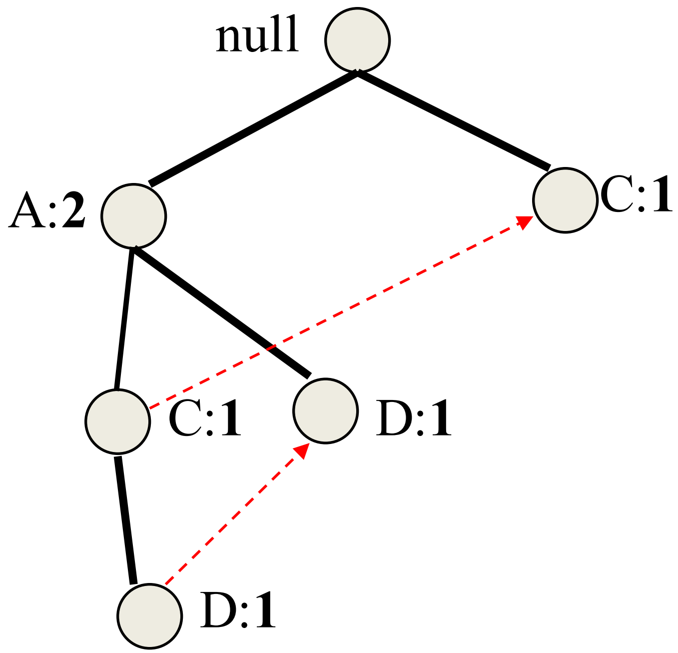
Example



The conditional FP-tree for E

We repeat the algorithm for ~~{D,E}, {C,E},~~ {A,E}

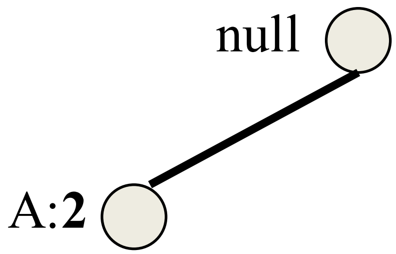
Example



Phase 1

Find all prefix paths that contain A (AE) in the conditional FP-tree

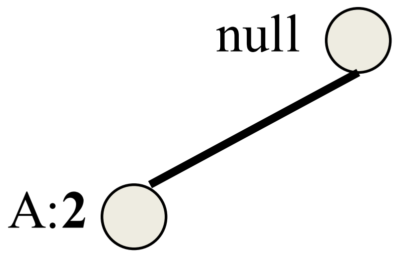
Example



Phase 1

Find all prefix paths that contain A (AE) in the conditional FP-tree

Example



Compute the support of {A,E} by following the pointers in the tree
2 ≥ minsup

{A,E} is frequent

There is no conditional FP-tree for {A,E}

Example

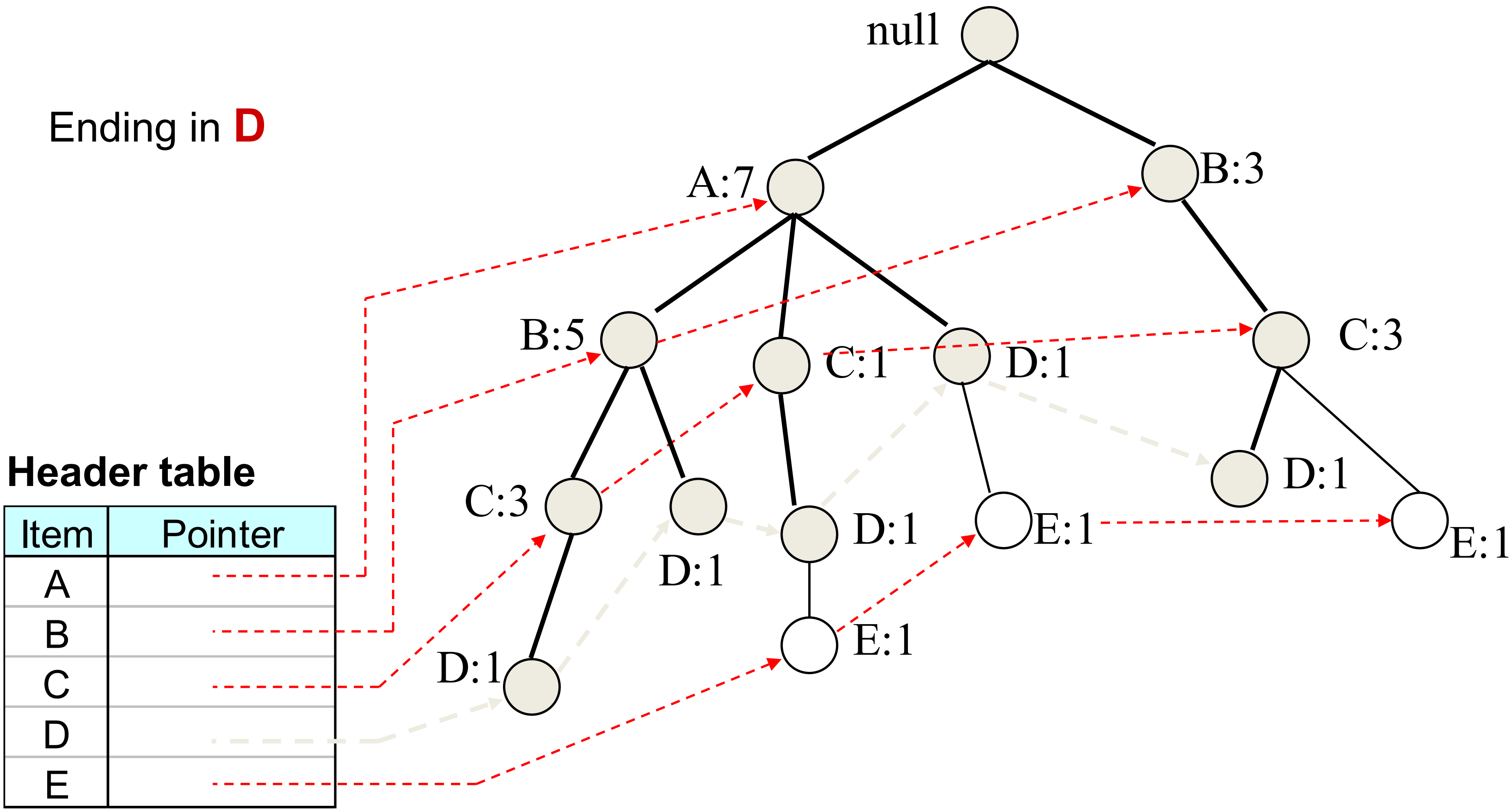
- So for E we have the following frequent itemsets

{E}, {D,E}, {C,E}, {A,E}

- We proceed with D

Example

Ending in **D**



Example

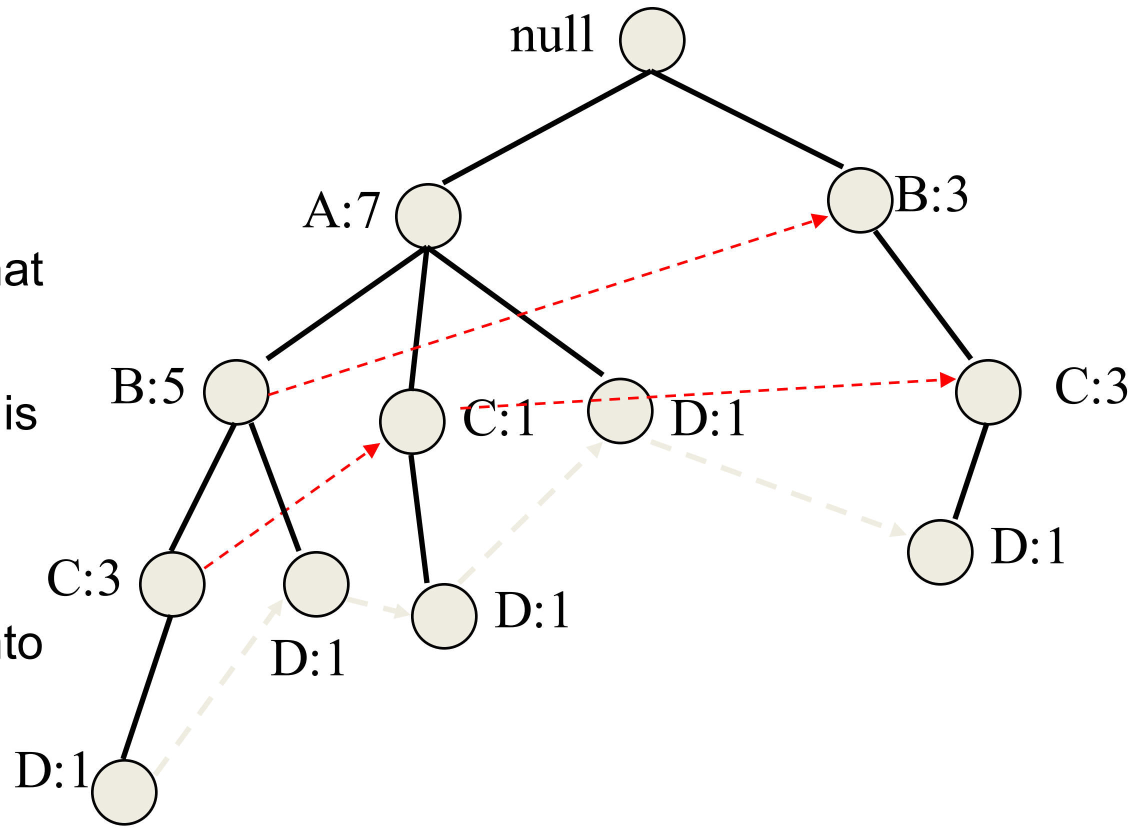
Phase 1 – construct prefix tree

Find all prefix paths that contain D

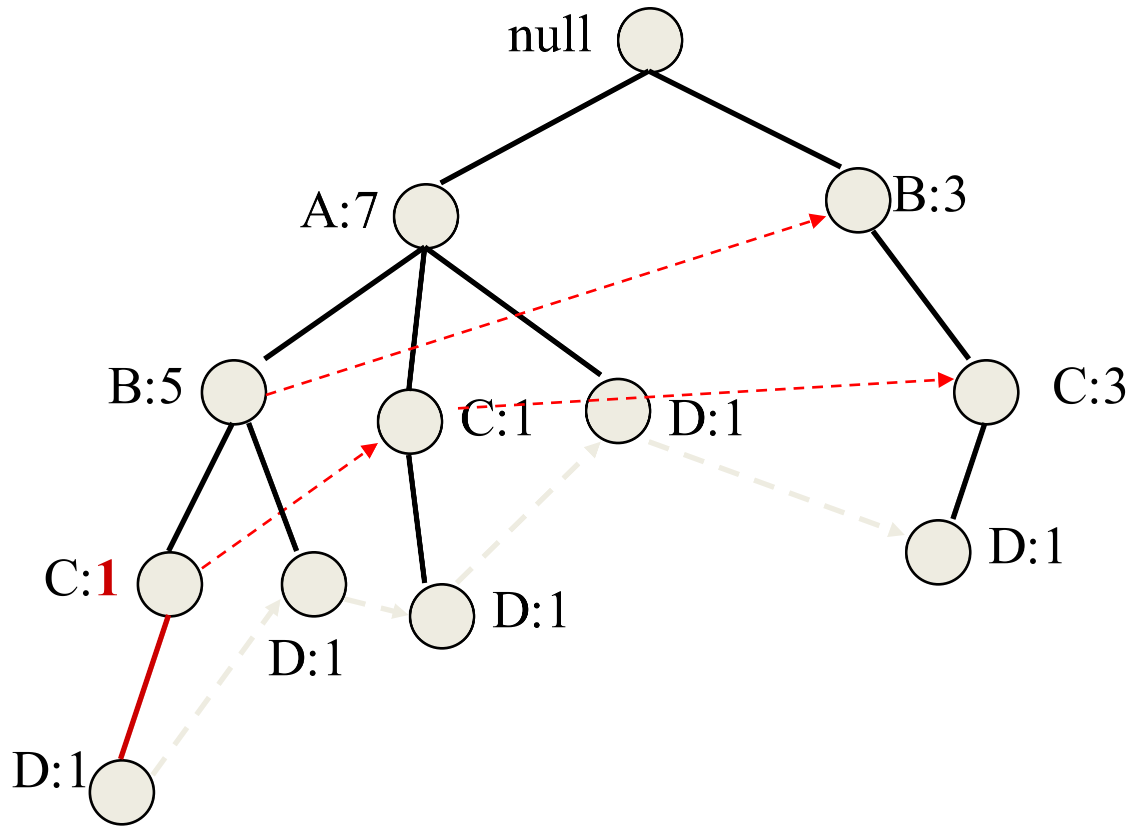
Support 5 > minsup, D is frequent

Phase 2

Convert prefix tree into conditional FP-tree

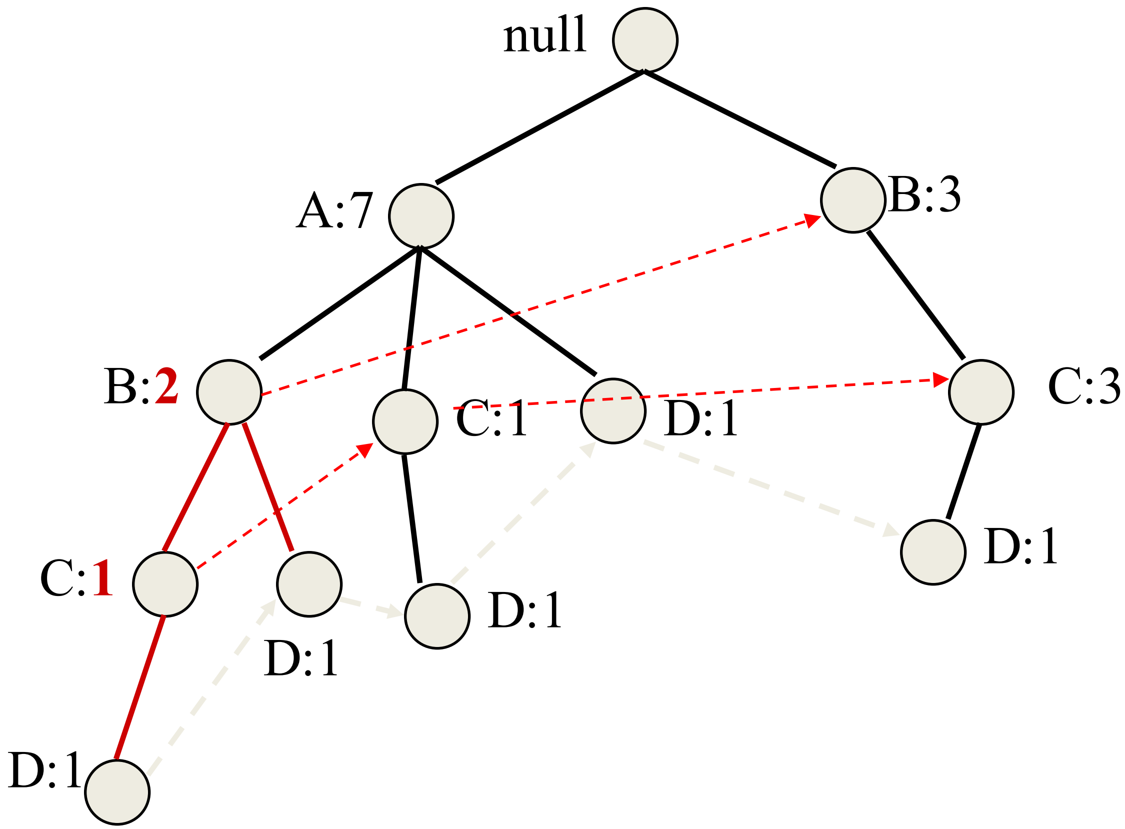


Example



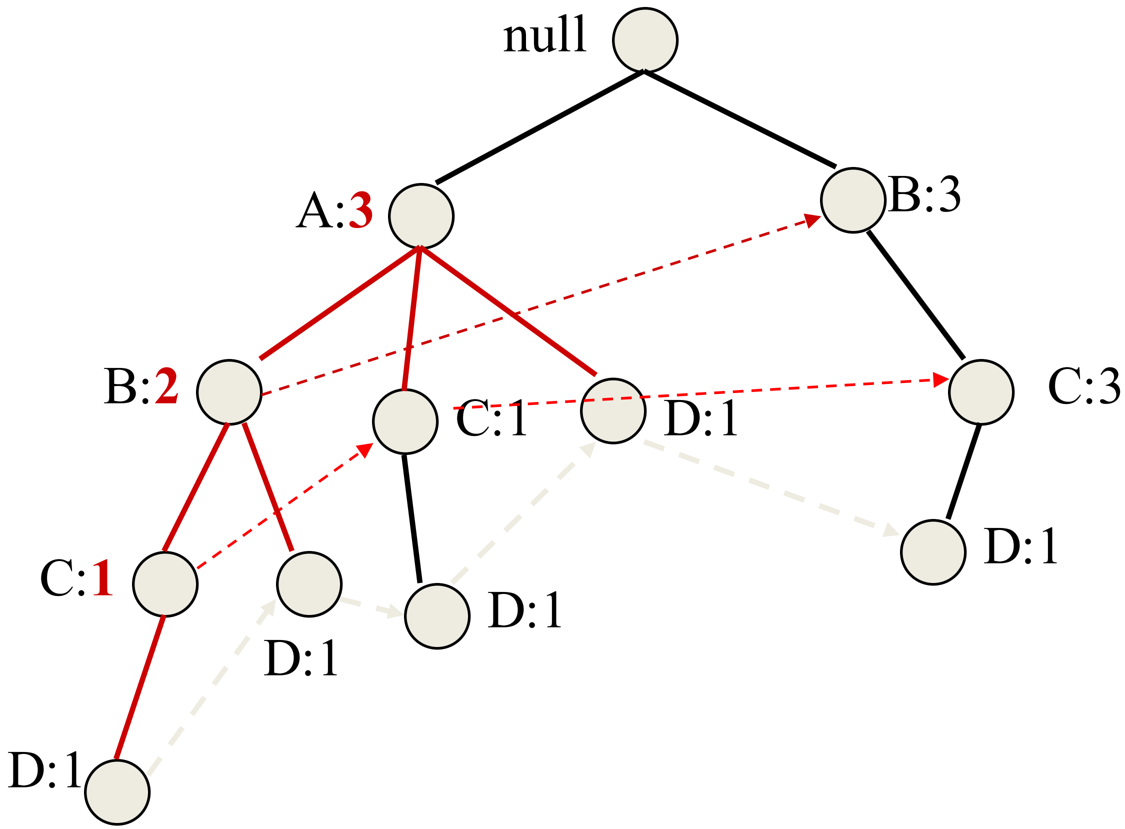
Recompute support

Example



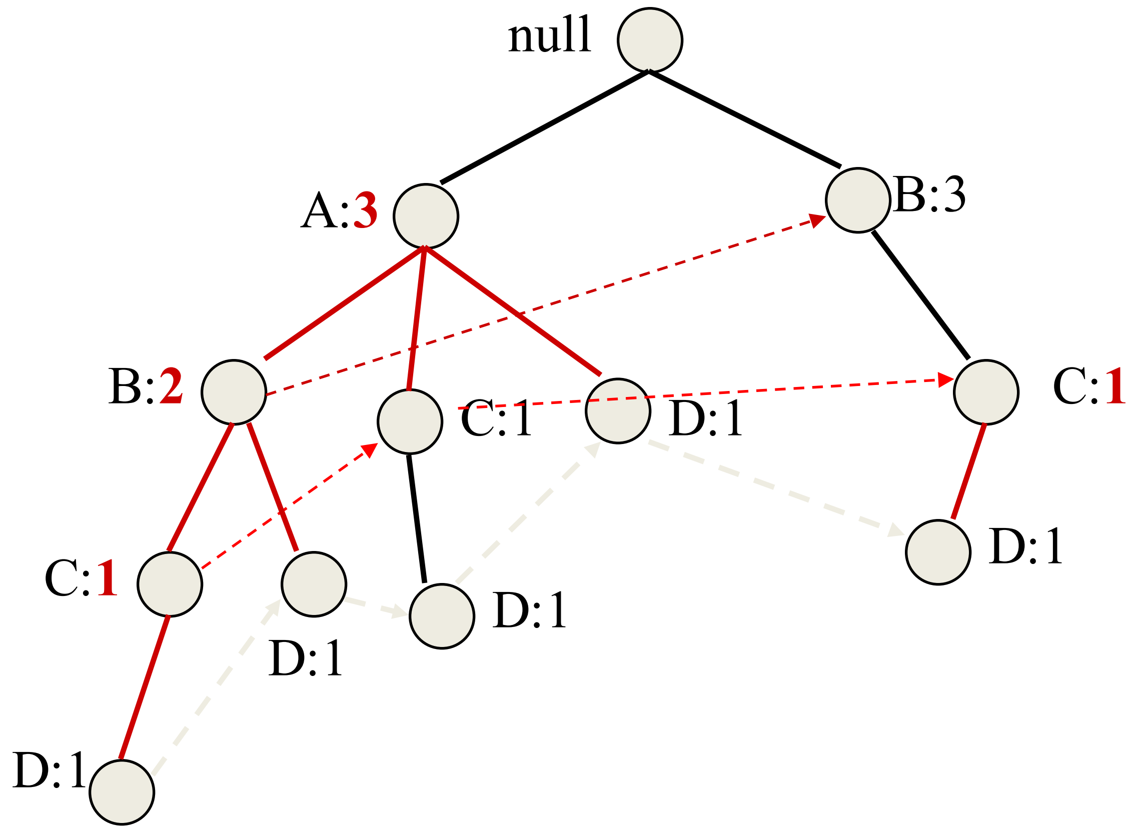
Recompute support

Example



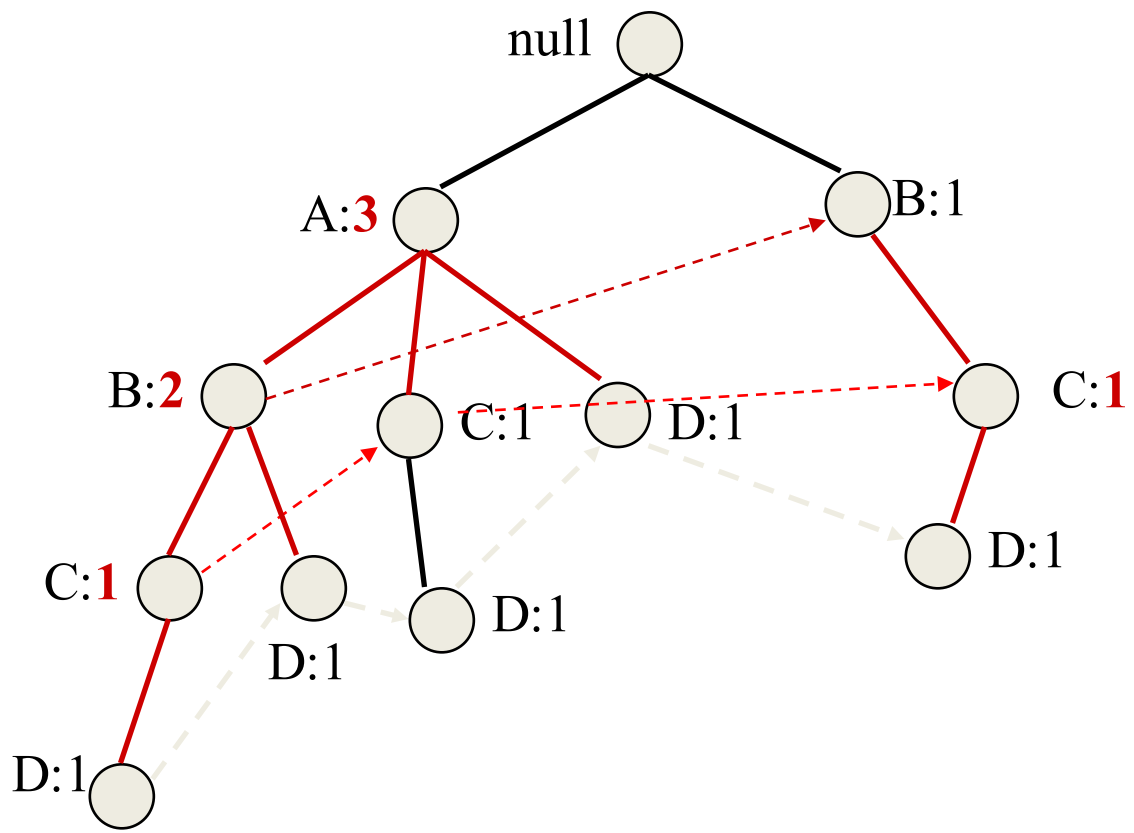
Recompute support

Example



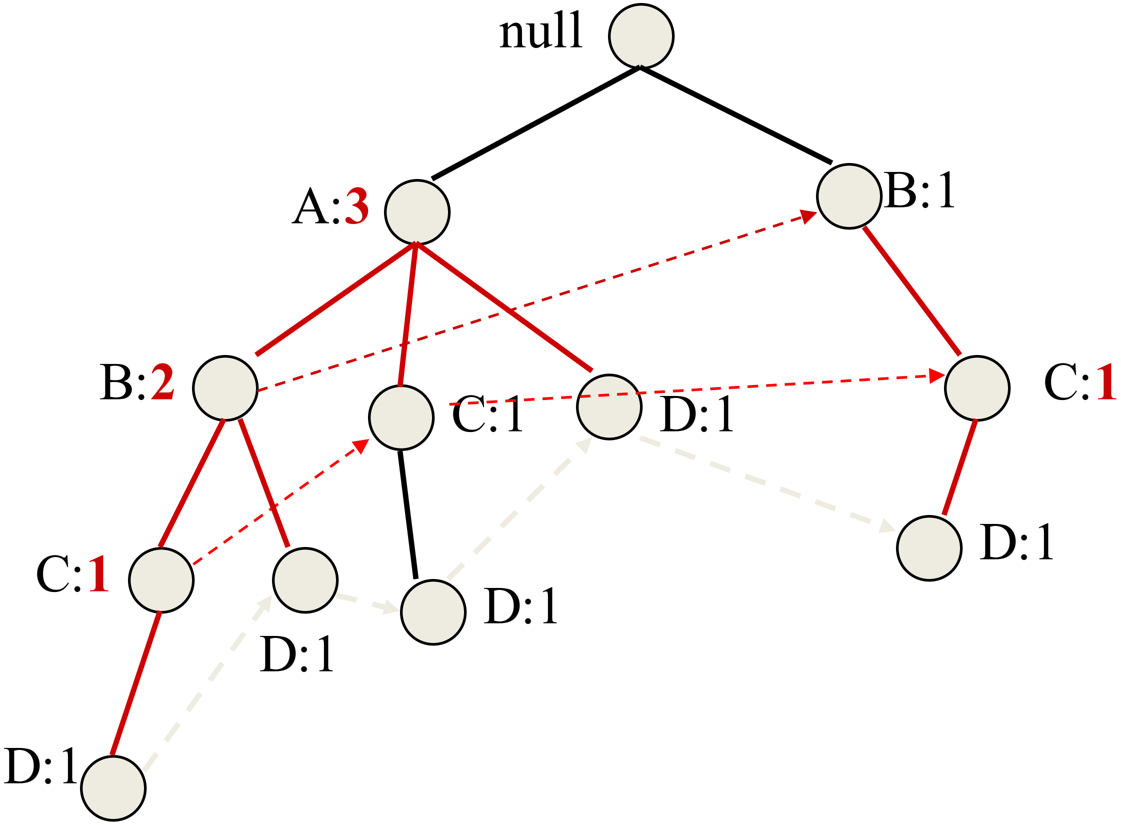
Recompute support

Example



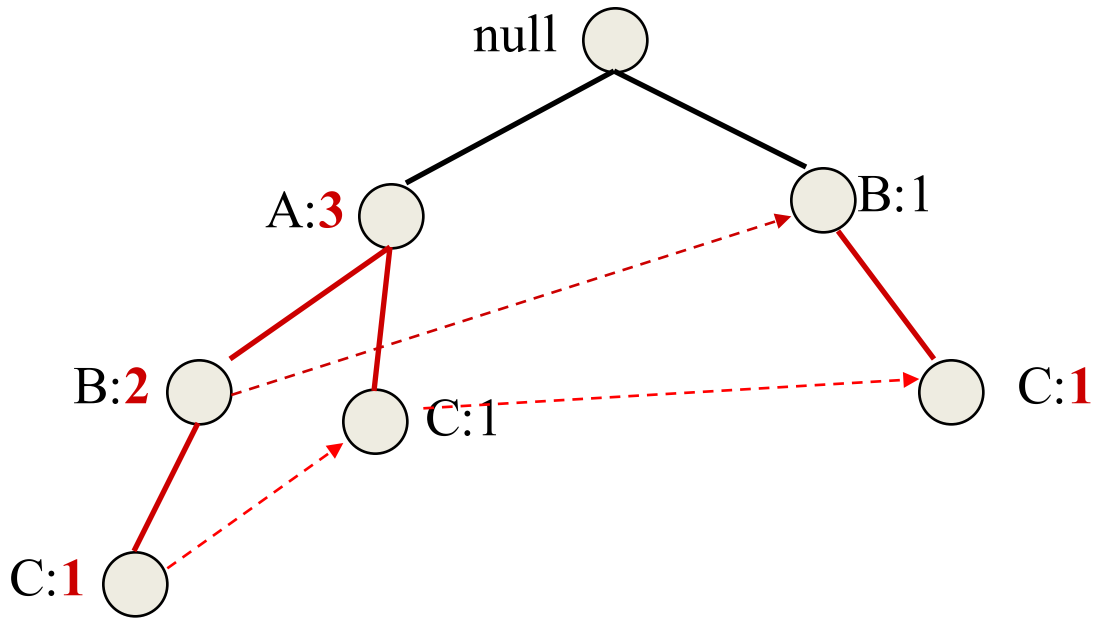
Recompute support

Example



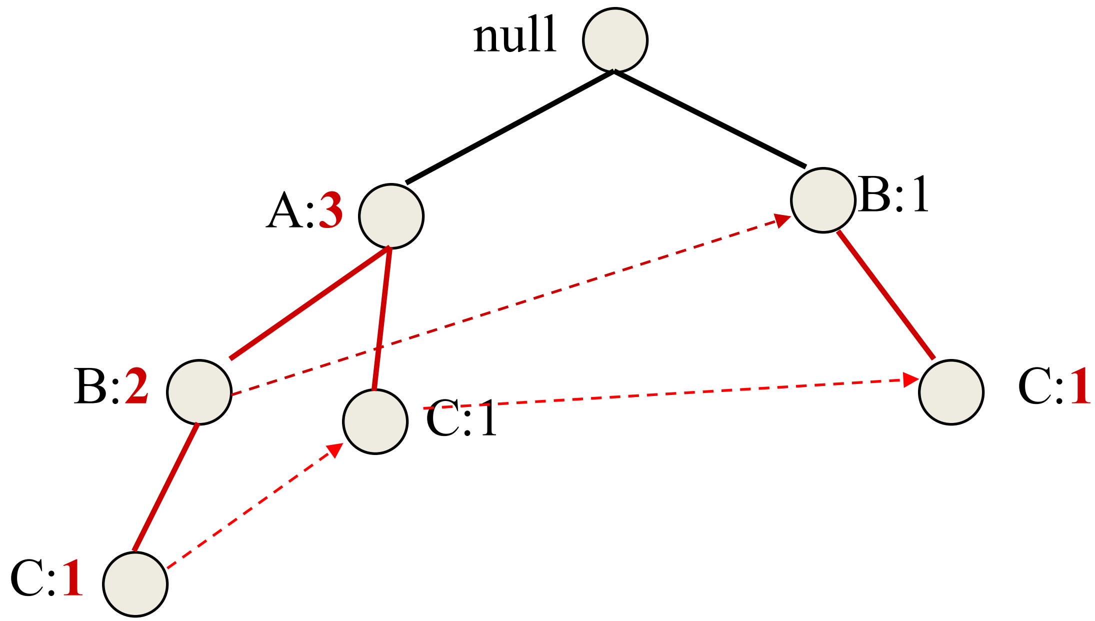
Prune nodes

Example



Prune nodes

Example



Construct conditional FP-trees for {C,D}, {B,D}, {A,D}

And so on….

Observations

- At each recursive step we solve a subproblem
  - Construct the prefix tree
  - Compute the new support
  - Prune nodes
- Subproblems are disjoint so we never consider the same itemset twice

- Support computation is efficient – happens together with the computation of the frequent itemsets.

Observations

- The efficiency of the algorithm depends on the compaction factor of the dataset

- If the tree is bushy then the algorithm does not work well, it increases a lot of number of subproblems that need to be solved.

FREQUENT ITEMSET RESEARCH

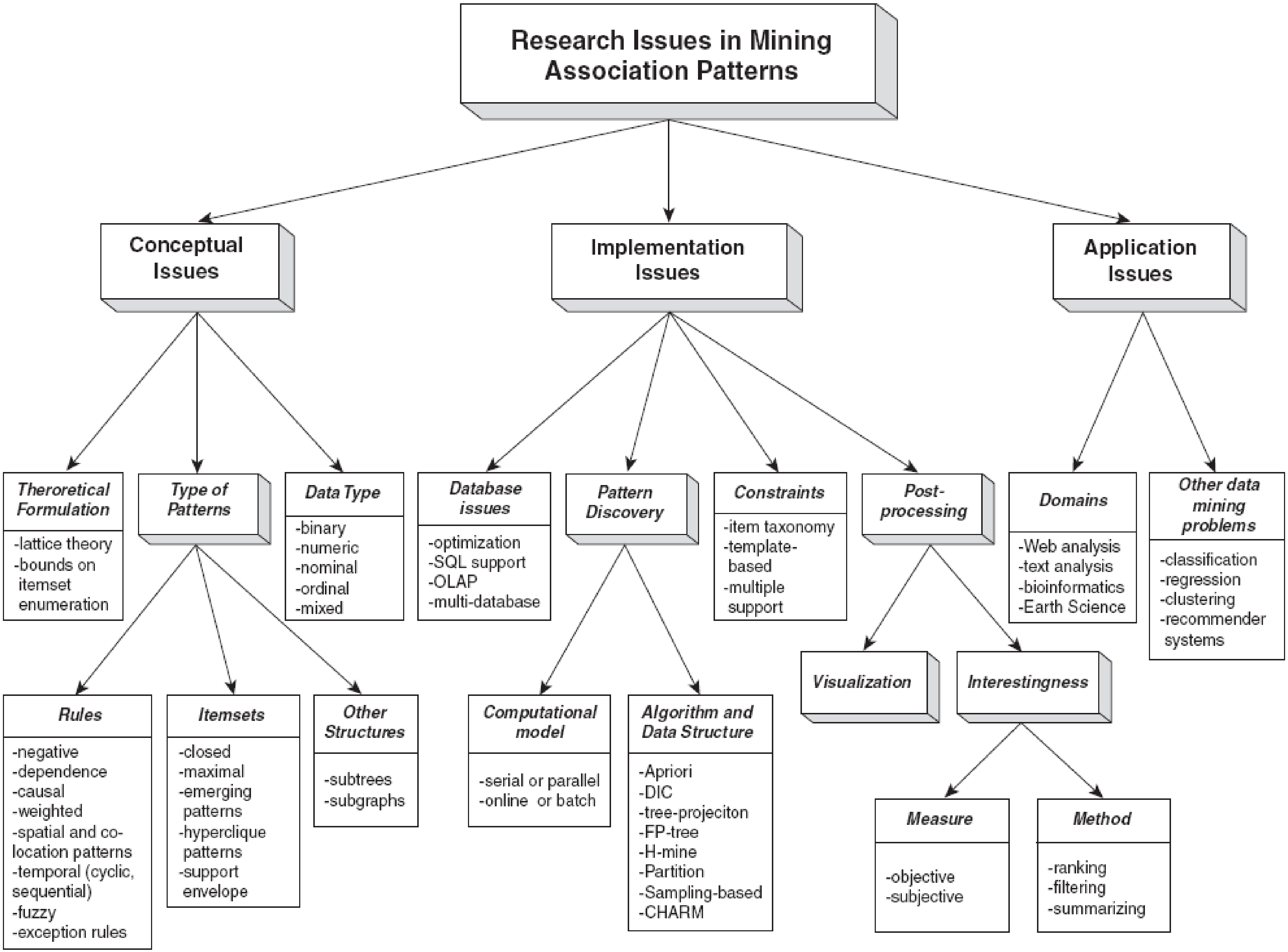


Figure 6.31. A summary of the various research activities in association analysis.