FP-Growth Algorithm

Apriori는 매번 디스크를 스캔해야함 → 느리다

FP-Growth는 메모리에서 모든걸 해결 (메모리를 아주 많이 사용)

↳ 메모리에 담을 수 없으면 극도로 느려짐 , 데이터가 소량이면 압도적

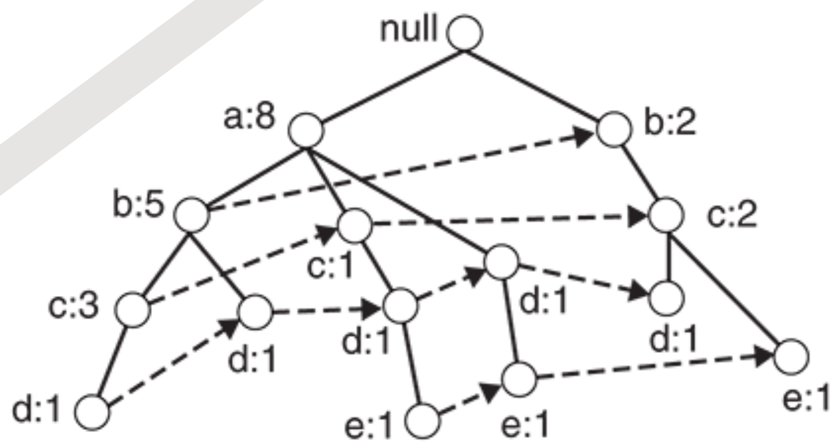
이번학기에 수업하지 않음. 학부레벨 X

FP-Growth Algorithm

- A ***radically*** different approach to discovering frequent itemsets
  - Does not subscribe to the generate-and-test paradigm of Apriori
  - Instead, it encodes the data set using a ***compact data structure***
    - Called an ***FP-tree***
  - Then, it extracts frequent itemsets directly from this structure

Transaction Data Set

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	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



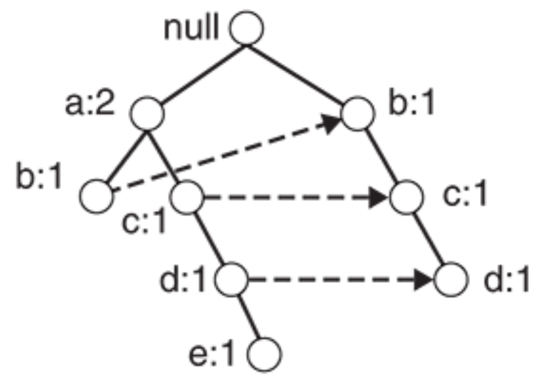
FP-tree

Frequent Itemsets

FP-Tree

- A ***compressed*** representation of the input data set
  - Constructed by mapping each transaction onto a ***path*** in the FP-tree
  - As different transactions have common items, their paths might ***overlap***
  - The more the paths overlap with one another, the more compression we can achieve using the FP-tree structure

TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}



- If the size of the FP-tree is small enough to fit into main memory, we can extract frequent itemsets ***directly from FP-tree in memory***
  - Instead of making repeated passes over the data stored on disk

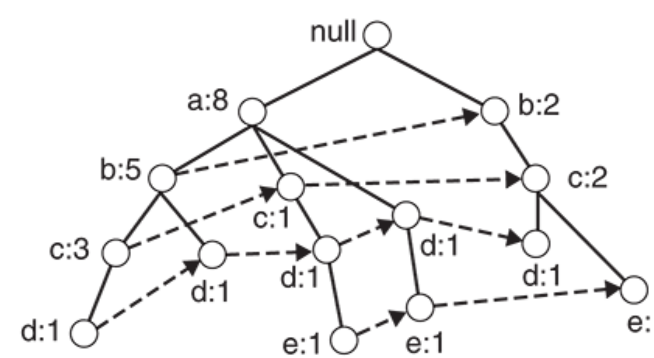
Construction of an FP-Tree

- Initially, the FP-tree contains only the root node   null

- **Step 1**:
  - Scan the data to count each item
  - Discard infrequent items
  - ***Sort*** frequent items in decreasing support counts inside every transaction of the data set
    - In this example, $\sigma(a) > \sigma(b) > \sigma(c) > \sigma(d) > \sigma(e)$

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	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

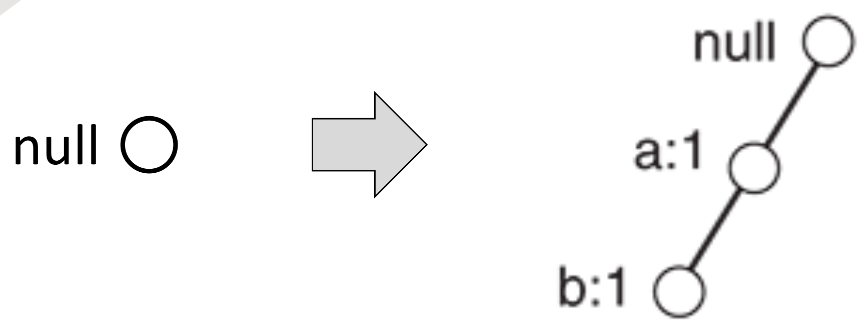
- **Step 2**:
  - For each transaction, ***extend*** the FP-tree
    - Map each transaction onto a path in the FP-tree
  - We illustrate this step from the next slide



(Ex) Construction of an FP-Tree (1/4)

- After reading the first transaction $\{a, b\}$
  - Create a path a → b to encode this transaction
  - Add this path to the root of the FP-tree
    - Every node along the path has a frequency count of 1

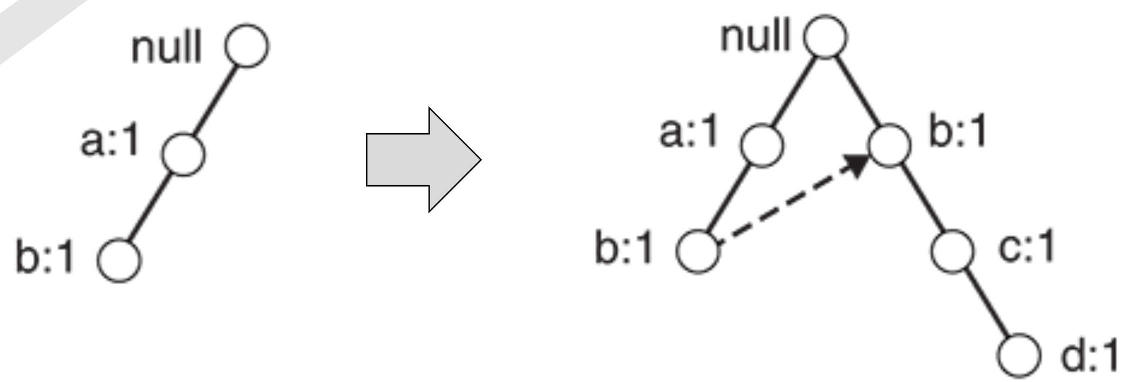
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



(Ex) Construction of an FP-Tree (2/4)

- After reading the second transaction $\{b, c, d\}$
  - Create a path $b \rightarrow c \rightarrow d$ to encode this transaction
  - Add this path to the root of the FP-tree
    - Although the first two transactions have an item b in common, their paths are ***disjoint*** because the transactions do not share a common prefix

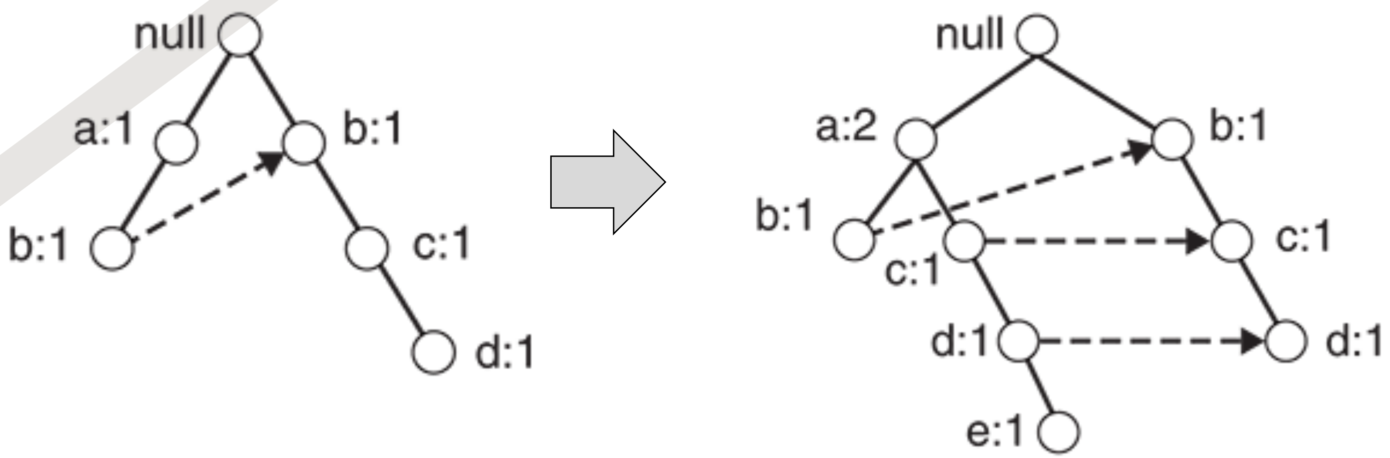
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



(Ex) Construction of an FP-Tree (3/4)

- After reading the third transaction $\{a, c, d, e\}$
  - Create a path $a \rightarrow c \rightarrow d \rightarrow e$ to encode this transaction
  - Add this path to the root of the FP-tree
    - Because this path shares a common prefix a with the first transaction, the frequency count for a is incremented to ***two***

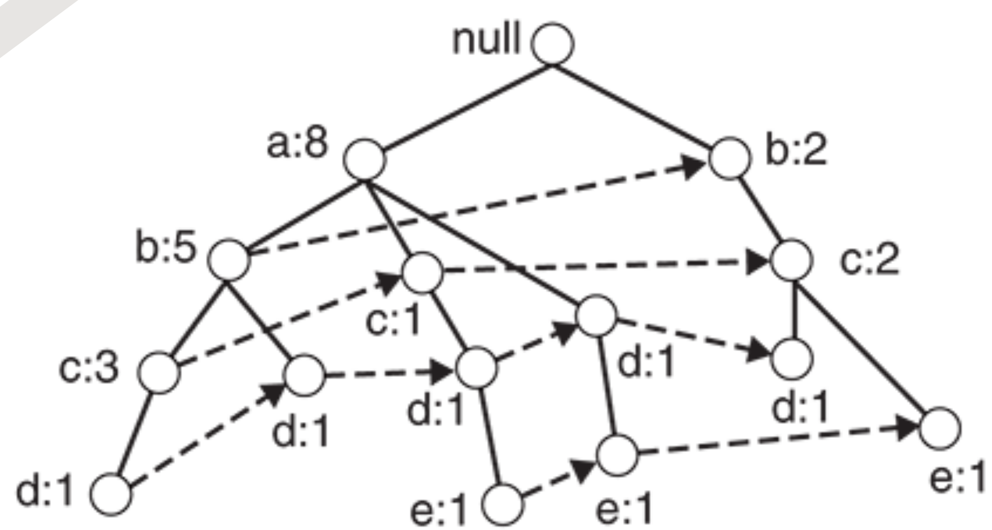
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



(Ex) Construction of an FP-Tree (4/4)

- Continue this process until
  - Every transaction has been mapped onto one of the paths in the FP-tree
    - If different transactions share a common prefix, their paths ***share*** the common nodes and the frequency counts for the common nodes ***increase***
  - The following is the resulting FP-tree after reading all the transaction

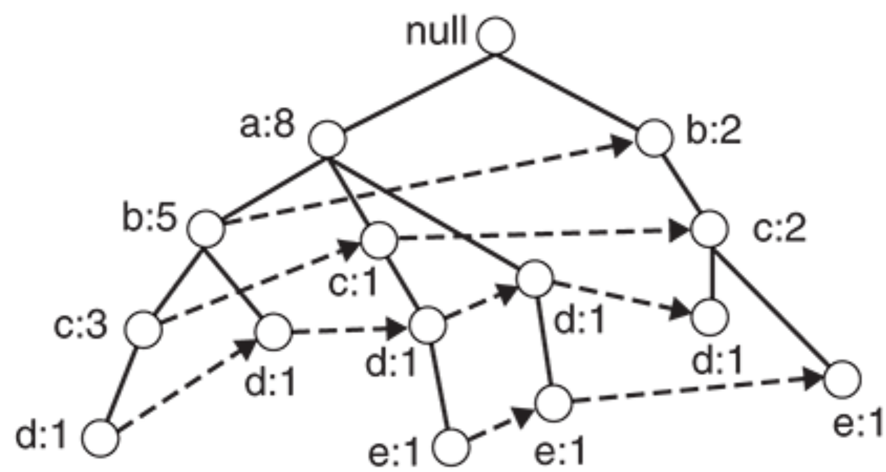
TID	Items
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}



The Size of an FP-Tree (1/2)

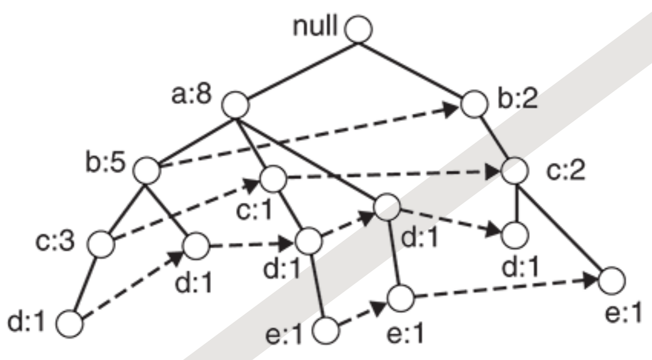
- Typically ***smaller*** than the size of the original data set
  - Because many transaction often ***share*** a few items in common
  - **Best case**: all the transactions have the same items
    - The FP-tree contains only a single branch of nodes
  - **Worst case**: none of the transactions have any items in common
    - The size of the FP-tree is effectively the same as the size of the original data

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	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

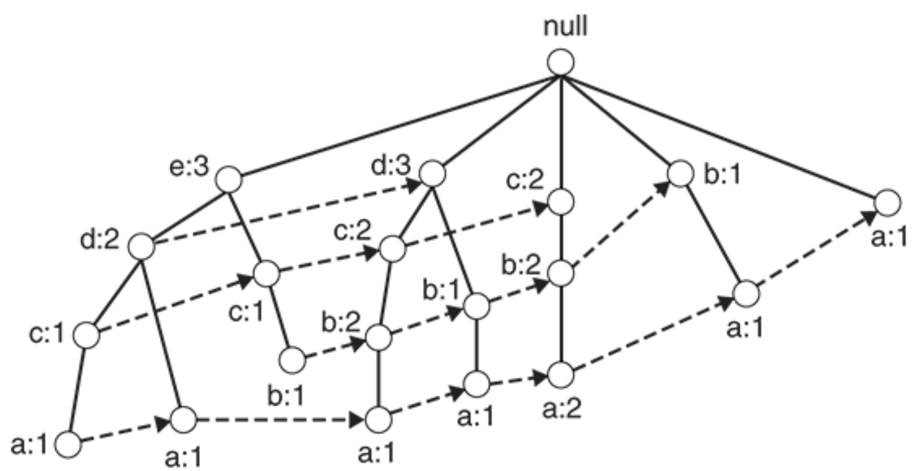


The Size of an FP-Tree (2/2)

- Why do we sort items in ***decreasing*** order of support counts inside every transaction?
  - The high support items occur more frequently across all paths
  - Hence, they must be used as most commonly occurring prefixes
  - However, this strategy does not always lead to the smallest tree
    - Especially when the high support items do not occur frequently together with the other items



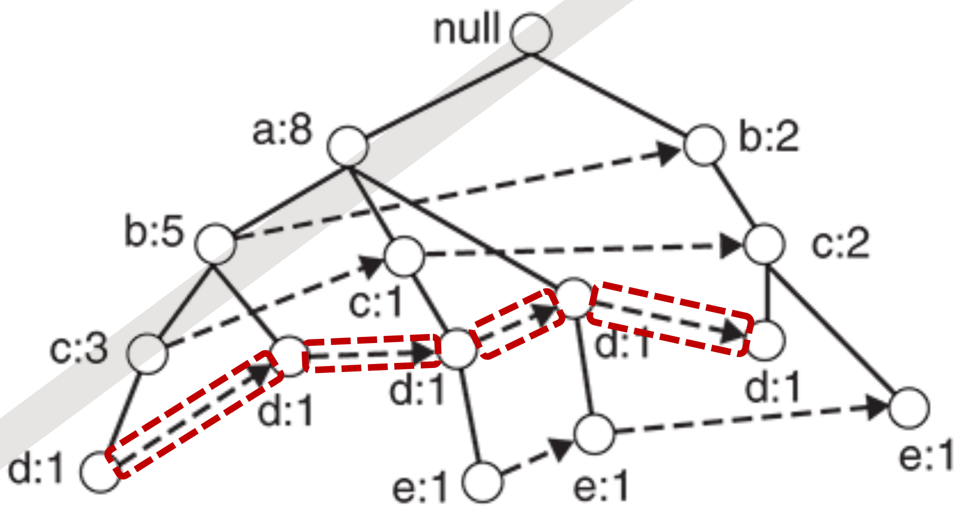
When items are sorted in
decreasing order of support counts



When items are sorted in
increasing order of support counts
(The number of branches are increased)

Pointers in FP-Tree

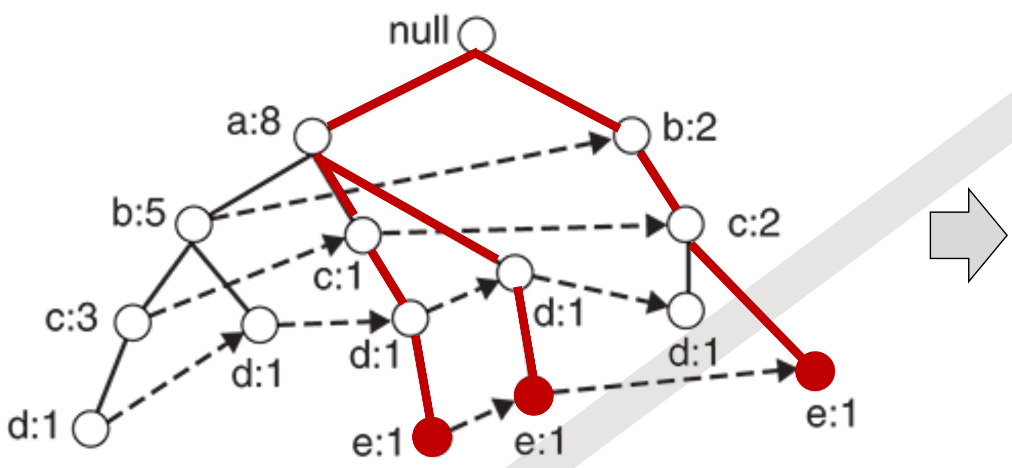
- An FP-tree also contains a list of ***pointers*** connecting nodes that represent the ***same*** items
  - These pointers help the rapid access of individual items in the tree
  - We can follow the pointers connecting d to obtain the ***support count*** for d



- We discuss how to use the FP-tree and its corresponding pointers for frequent itemset generation from the next slide

FP-Growth

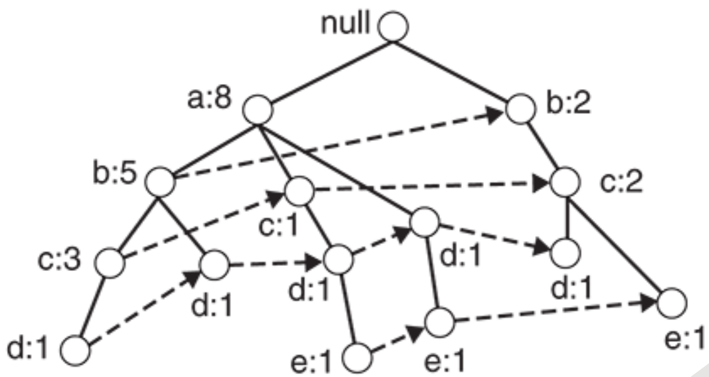
- An algorithm that ***generates frequent itemsets*** from an FP-tree
  - By exploring the tree in a bottom-up fashion
    - (ex) Find frequent itemsets ending in e first, followed by d, c, b, and finally a



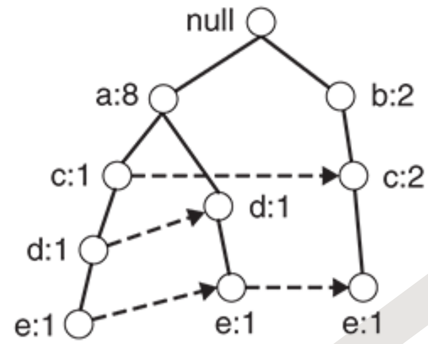
$\{e\}, \{d, e\}, \{c, e\}, \{b, e\}, \{a, e\}, \ldots$
$\{d\}, \{c, d\}, \{b, d\}, \{a, d\}, \ldots$
$\{c\}, \{b, c\}, \{a, c\}, \ldots$
$\{b\}, \{a, b\}, \ldots$
$\{a\}$

- We can derive the frequent itemsets ending with a particular item, say, e, by examining only the paths containing e
  - These paths can be accessed rapidly using the pointers associated with e

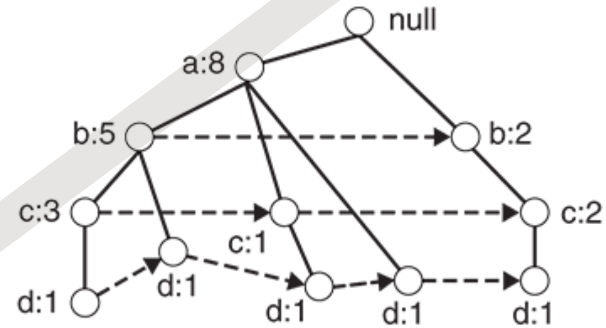
(Ex) Frequent Itemset Generation



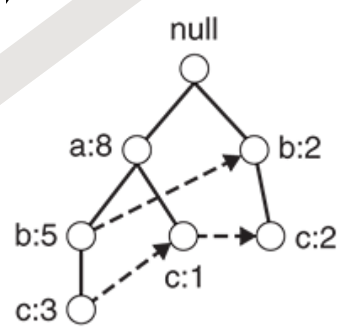
FP-tree



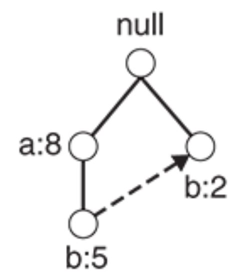
(a) Paths containing e
(used to find frequent itemsets ending with e)



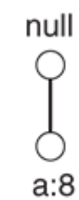
(b) Paths containing d
(used to find frequent itemsets ending with d)



(c) Paths containing c
(used to find frequent itemsets ending with c)



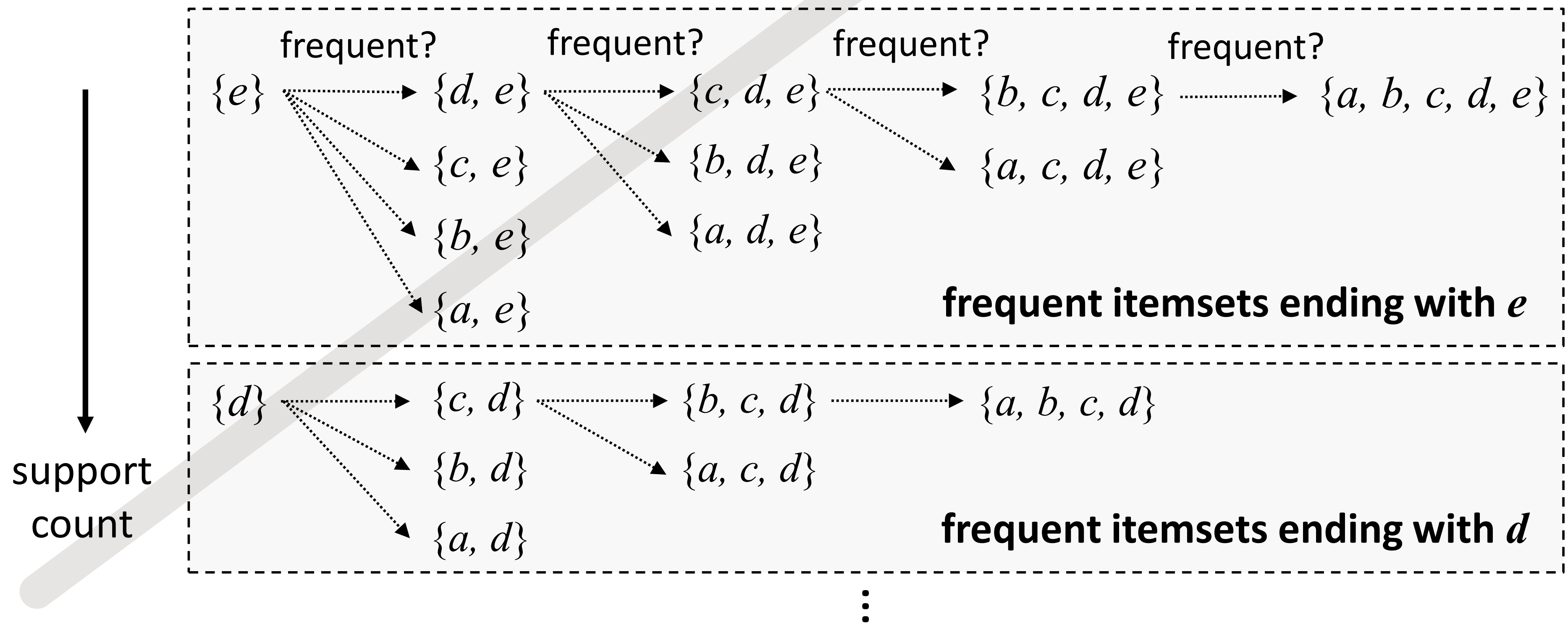
(d) Paths containing b
(used to find frequent itemsets ending with b)



(e) Paths containing a
(used to find frequent itemsets ending with a)

Divide-and-Conquer Strategy of FP-Growth

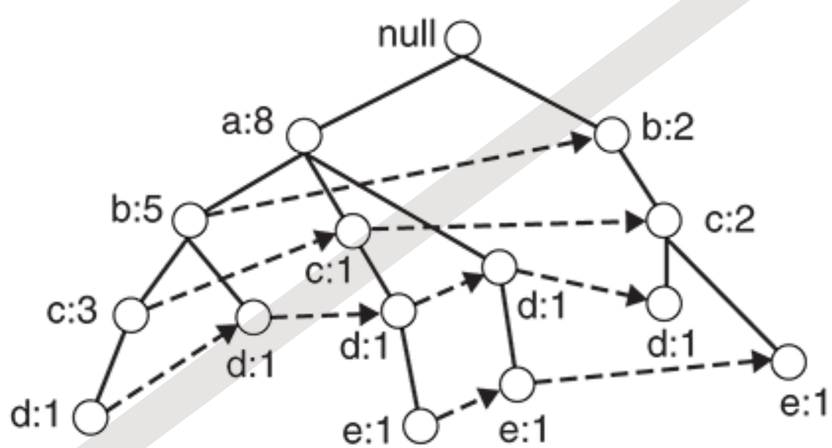
- FP-growth finds all the frequent itemsets by ***splitting*** the problem into smaller sub-problems
  - Finally, FP-growth ***merges*** the solutions to the sub-problems
- Example (suppose $\sigma(a) > \sigma(b) > \sigma(c) > \sigma(d) > \sigma(e) > minsup$)
  - FP-growth finds frequent itemsets in the following order



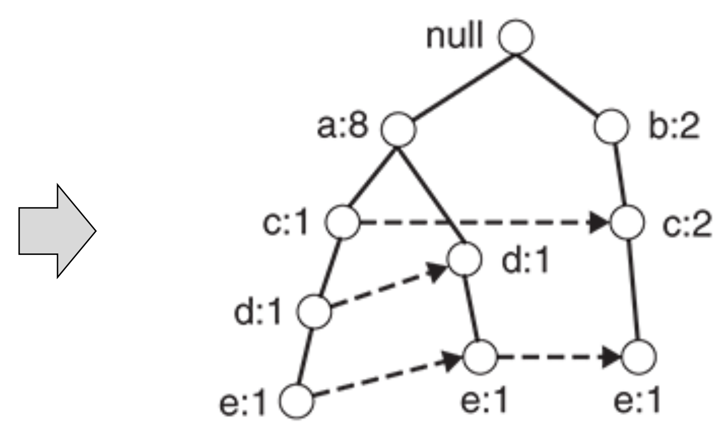
(Ex) FP-Growth (1/7)

- Consider the task of finding frequent itemsets ending with e
  - (ex) $\{e\}$, $\{d, e\}$, $\{c, e\}$, $\{b, e\}$, $\{a, e\}$, $\{c, d, e\}$, $\{b, d, e\}$, $\{a, d, e\}$, …
  - We assume that $minsup = 2$

- **Step 1**: From the FP-tree, gather all the paths containing e
  - These paths are called the ***prefix paths*** ending in e



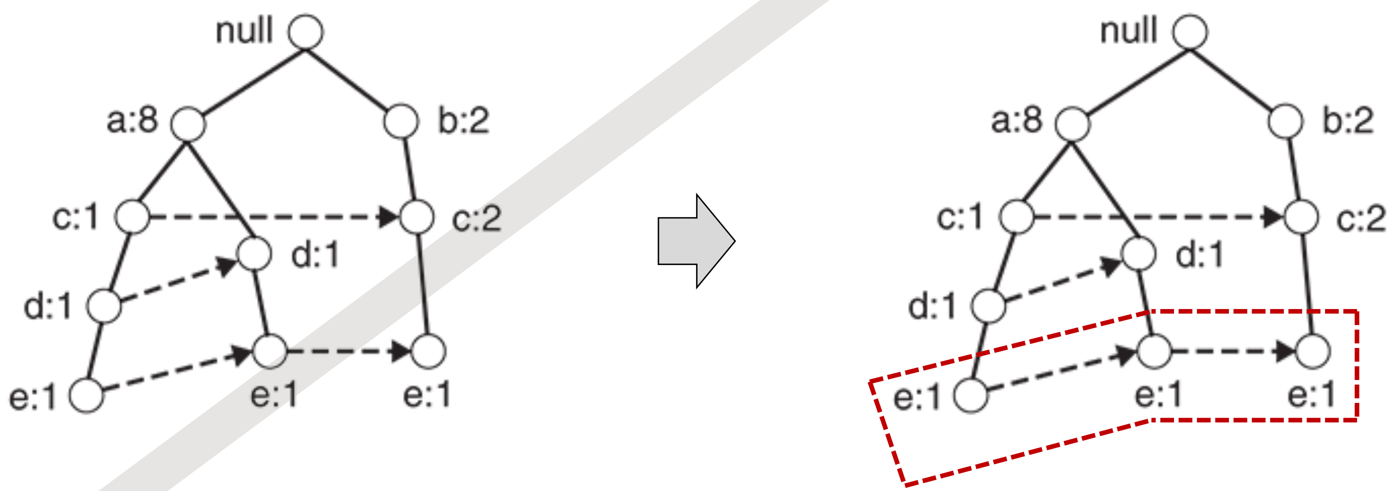
FP-tree



Prefix paths ending in e

(Ex) FP-Growth (1/7)

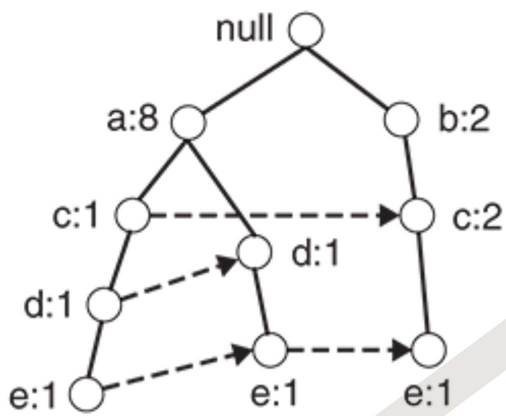
- **Step 2**: From the prefix paths, obtain the support count for e
  - By adding the support counts associated with e
  - Assuming that $minsup = 2$, $\{e\}$ is declared a frequent itemset because its support count is **3**



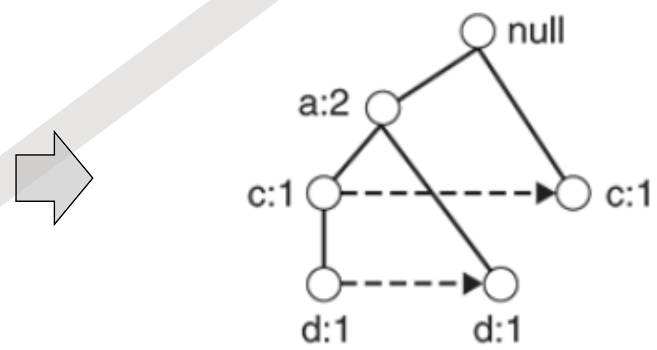
Prefix paths ending in e

(Ex) FP-Growth (2/7)

- **Step 3**: Because $\{e\}$ is frequent, we continue to find frequent itemsets ending in de, ce, be, and ae
  - To do so, we first convert the prefix paths into a ***conditional FP-tree*** for e



Prefix paths ending in e



Conditional FP-tree for e

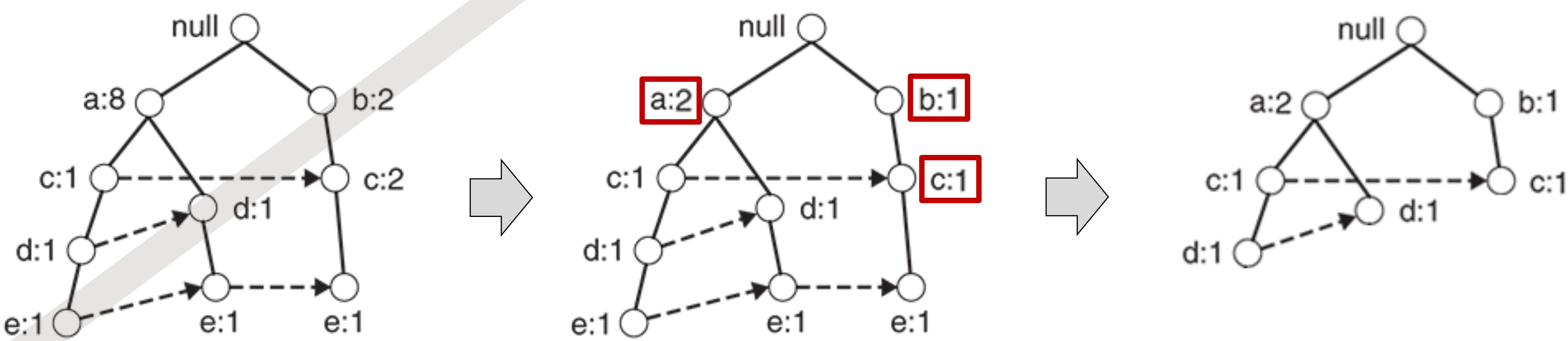
✓ Conditional FP-tree for e

- An "small" FP-tree representing ***only*** those transactions that contain e

Construction of a Conditional FP-Tree (1/2)

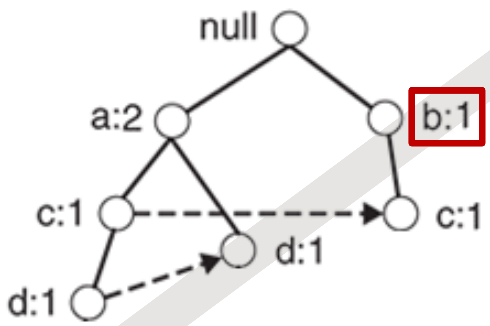
- **Step 3.a**: Update the support counts along the prefix paths
  - Because some of the counts include transactions that do not contain e
  - The updated counts reflect the actual number of transactions containing e

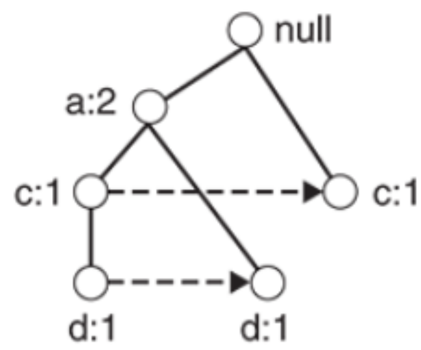
- **Step 3.b**: Remove the nodes for e
  - Because the support counts reflect only transactions that contain e
  - Hence, we no longer need information about node e



Construction of a Conditional FP-Tree (2/2)

- **Step 3.c**: Remove infrequent items
  - After updating the support counts, some items may no longer frequent
    - (ex) b has a support count equal to 1, which means there is only one transaction that contains both b and e
  - Those items can be safely ignored because all itemsets containing those items must be infrequent





The conditional FP-tree for *e*

(Ex) FP-Growth (3/7)

- **Step 4**: Using the conditional FP-tree for e, find frequent itemsets ending in de, ce, and ae
  - Note that b has been eliminated in the conditional FP-tree for e
  - In other words, we ***grow*** the frequent patterns (∴ FP-growth)

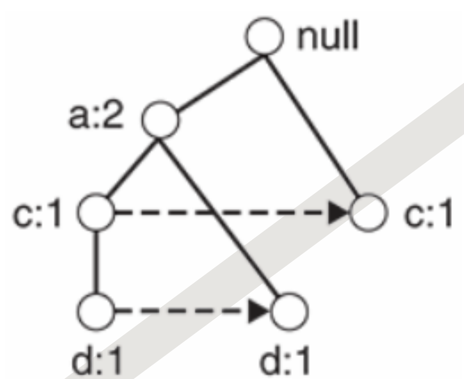
$$\{e\} \Longrightarrow \{d,\ e\},\ \{c,\ e\},\ \cancel{\{b,\ e\}},\ \{a,\ e\}$$

✓ In detail, FP-growth performs the following in Step 4

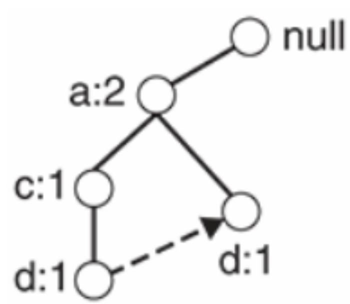
- Step 4.1: Finding the frequent itemsets ending in de
- Step 4.2: Finding the frequent itemsets ending in ce
- Step 4.3: Finding the frequent itemsets ending in ae

(Ex) FP-Growth (4/7)

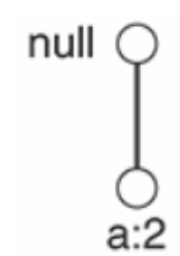
- **Step 4.1**: Finding the frequent itemsets ending in de
  - From the conditional FP-tree for e, gather all the paths containing d
    - i.e., the prefix paths ending in de
  - Obtain the support count for d
    - Since the support count is equal to 2, $\{d, e\}$ is declared a frequent itemset
  - Because $\{d, e\}$ is frequent, we continue to find frequent itemsets ending in cde and ade
    - To do so, we convert the prefix paths into a conditional FP-tree for de



Conditional FP-tree for e



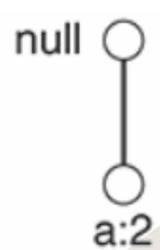
Prefix paths ending in de



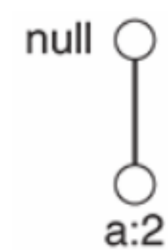
Conditional FP-tree for de
(c is infrequent and removed)

(Ex) FP-Growth (5/7)

- **Step 4.1.1**: Finding the frequent itemsets ending in *ade*
  - From the conditional FP-tree for *de*, gather all the paths containing *a*
    - i.e., the prefix paths ending in *ade*
  - Obtain the support count for *a*
    - Since the support count is equal to 2, $\{a, d, e\}$ is declared a frequent itemset
  - Although $\{a, d, e\}$ is frequent, we stop because there are no more items left



Conditional FP-tree for *de*



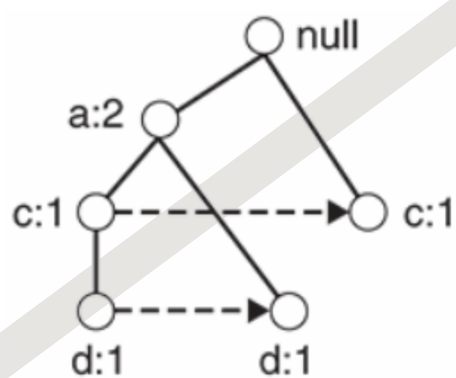
Prefix paths ending in *ade*

$\varnothing$

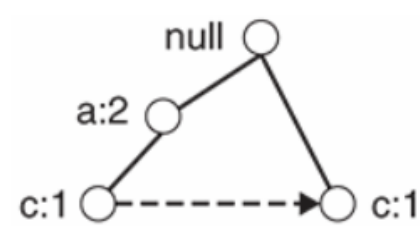
Conditional FP-tree for *ade*

(Ex) FP-Growth (6/7)

- **Step 4.2**: Finding the frequent itemsets ending in ce
  - From the conditional FP-tree for e, gather all the paths containing c
    - i.e., the prefix paths ending in ce
  - Obtain the support count for c
    - Since the support count is equal to 2, $\{c, e\}$ is declared a frequent itemset
  - Because $\{c, e\}$ is frequent, we continue to find frequent itemsets ending in bce and ace
    - To do so, we convert the prefix paths into a conditional FP-tree for ce
    - However, the conditional FP-tree for ce has no frequent items and we stop



Conditional FP-tree for e



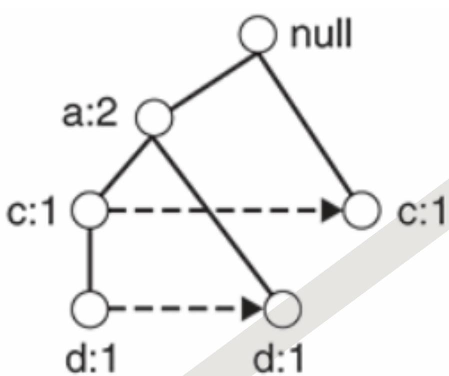
Prefix paths ending in ce

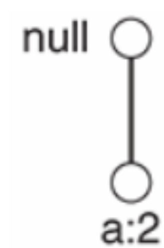
$\varnothing$

Conditional FP-tree for ce

(Ex) FP-Growth (7/7)

- **Step 4.3**: Finding the frequent itemsets ending in ae
  - From the conditional FP-tree for e, gather all the paths containing a
    - i.e., the prefix paths ending in ae
  - Obtain the support count for a
    - Since the support count is equal to 2, $\{a, e\}$ is declared a frequent itemset
  - Although $\{a, e\}$ is frequent, we stop because there are no more items left



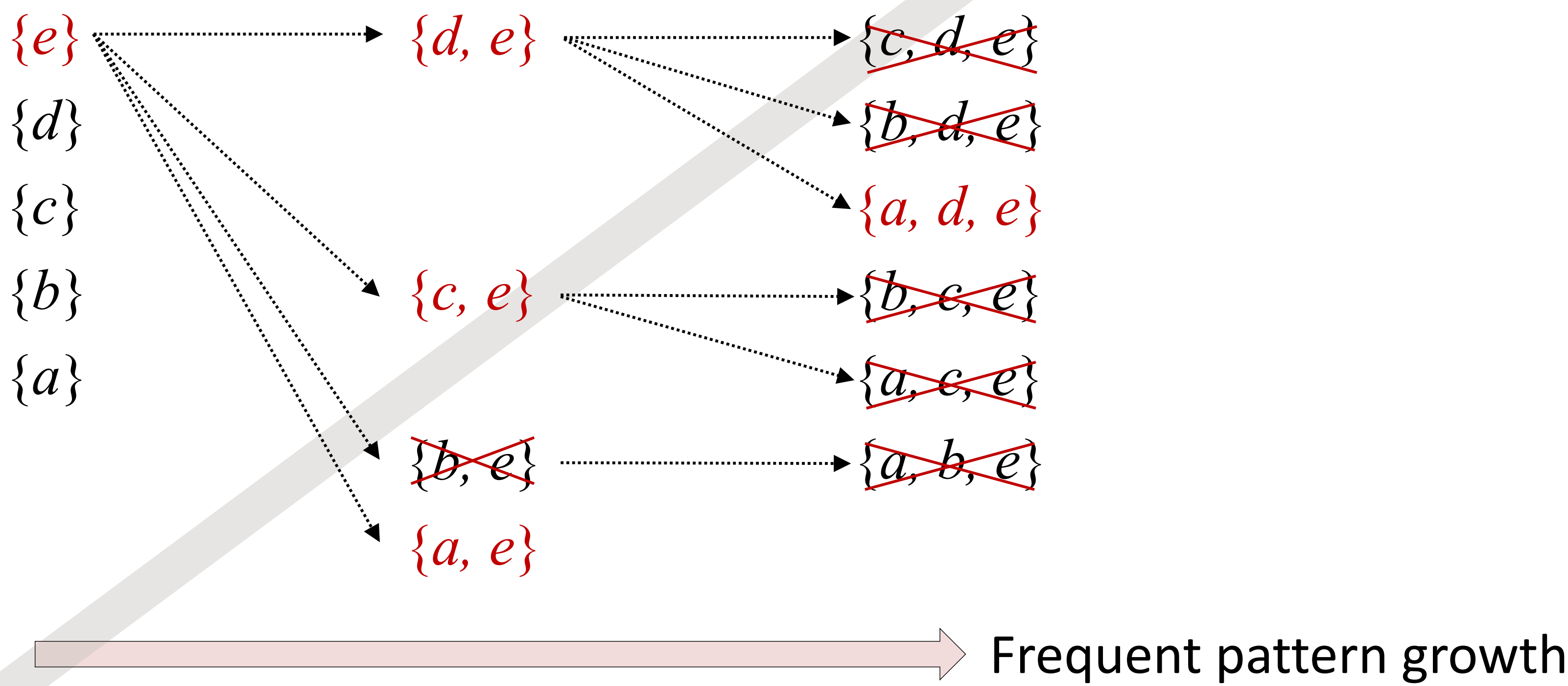


$\varnothing$

Conditional FP-tree for e | Prefix paths ending in ae | Conditional FP-tree for ae

(Ex) FP-Growth (8/8)

- Summary: what have we done up to now?
  - We have found all the frequent itemsets ending in e



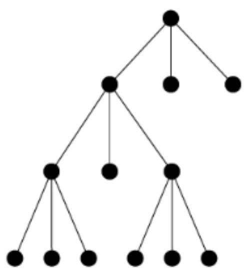
- Next, we continue to find frequent itemsets ending in d
  - Then c, b, and a in turn

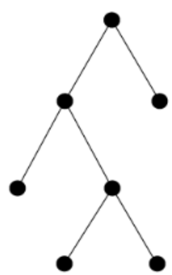
Characteristics of FP-Growth

- FP-growth does ***not*** generate any duplicate itemsets
  - Because the sub-problems are disjoint

- The run-time performance of FP-growth depends on the ***compaction factor*** of the data set
  - If the resulting conditional FP-trees are very ***bushy***
    - Its performance degrades significantly
    - Because it has to generate a large number of sub-problems
  - If the resulting conditional FP-trees are ***shallow***
    - It outperforms the standard Apriori algorithm by several orders of magnitude

Bushy tree





Shallow tree

Evaluation of Association Rules

↳ ① $X \rightarrow Y$: support
② $\{X, Y\}$: confidence

이것만으로 충분하지 않다?!

④ lift (interest factor)

Evaluation of Association Rules

- Association analysis algorithms have the potential to generate a ***large*** number of association rules
  - As the size and dimensionality of real commercial databases can be very large, we can easily end up with thousands or even millions of patterns
  - However, many of them might ***not*** be interesting
    - (ex) $\{Butter\} \rightarrow \{Bread\}$

규칙을 평가하는 객관적인 수치

- It is therefore important to evaluate the ***quality*** of rules
  - For example, we can define ***objective interestingness measures***
  - These measures can be used to rank found association rules

- Here, we study one more measure for association rules
  - i.e., the interest factor or the "lift"

(X와 상관없이) Y를 사는 사람 : $\frac{\sigma(\{Y\})}{N}$

X를 산 사람 中 Y를 산 사람 : $\frac{\sigma(\{X,Y\})}{\sigma(\{X\})}$

① $\frac{\sigma(\{Y\})}{N} < \frac{\sigma(\{X,Y\})}{\sigma(\{X\})}$ 인 경우

⇒ X가 Y의 구매에 영향을 미쳤다는 것

Interest Factor (or Lift)

② $\frac{\sigma(\{Y\})}{N} > \frac{\sigma(\{X,Y\})}{\sigma(\{X\})}$ 인 경우

⇒ X가 Y의 구매에 영향을 미쳤다는 것
(X가 Y의 대체제로 작용, Y를 안 삼)

③ $\frac{\sigma(\{Y\})}{N} \approx \frac{\sigma(\{X,Y\})}{\sigma(\{X\})}$ 인 경우

⇒ X가 Y의 구매에 영향을 미치지 않음
연관성 X

- ***Not*** all high confidence rules are interesting
  - If everyone buys $milk$, $X \rightarrow \{milk\}$ will have 100% confidence for any X
  - A rule $X \rightarrow Y$ is more valuable if it reflects a true relationship
    - i.e., X somehow affects Y

X→Y가 confidence, support를 모두 만족해도
Y가 진짜 X와 연관성이 있는지 알 수 없음
(ex. 이 매장에 오는 사람은 무조건 Y를 산다면? X가 뭐든 confidence가 높게 나옴.)

- Definition
  - The ***interest factor*** (or ***lift***) of a rule $X \rightarrow Y$, $I(X \rightarrow Y)$, is defined as follows:

$$I(X \rightarrow Y) = \frac{\sigma(X \cup Y)/\sigma(X)}{\sigma(Y)/N} = \frac{c(X \rightarrow Y)}{\sigma(Y)/N}$$

→ confidence

  - $I(X \rightarrow Y) = 1 \rightarrow$ X has no influence on Y
  - $I(X \rightarrow Y) > 1 \rightarrow$ the presence of X ***positively*** affects the presence of Y
  - $I(X \rightarrow Y) < 1 \rightarrow$ the presence of X ***discourages*** the presence of Y

(Ex) Interest Factor (or Lift)

- Consider the following data set

TID	Items
1	*milk, coke, beer*
2	*milk, pepsi, juice*
3	*milk, beer*
4	*coke, juice*
5	*milk, pepsi, beer*
6	*milk, coke, beer, juice*
7	*coke, beer, juice*
8	*beer, coke*

- What is the interest factor of the rule $\{milk, beer\} \rightarrow \{coke\}$?
  - $c(\{milk, beer\} \rightarrow \{coke\}) = 2/4 = 0.5$
  - $I(\{milk, beer\} \rightarrow \{coke\}) = 0.5/(\sigma(\{coke\})/8) = 0.5/0.625 = 0.8$
    - Thus, this rule is ***not*** very interesting! → 사람이 판단