Frequent Pattern (FP) Growth

Renato R. Maaliw III, *DIT*
College of Engineering
Southern Luzon State University
Lucban, Quezon, Philippines

FP Growth

- An algorithm for mining frequent itemsets without need for generating candidate itemsets.

- Particularly useful in application like market basket analysis, where it helps identify sets of products that frequency co-occur in transactions (Han, Peri, Yin; 2000)

Key Objectives of FP Growth

- Identify frequent itemset (combinations of items that frequently appear together)

- Optimize the process by reducing the computational complexity and memory requirements compared to Apriori

Core Concepts of FP Growth

- Uses a data structure called the FP-Tree (Frequent Pattern Tree) to represent transactions compactly, which facilitates efficient mining of frequent itemsets.

A. FP-Tree Structure

Compact Representation:

- The FP-Tree is a compressed structure that stores frequency of each item within the data, making it easy to locate patterns

A. FP-Tree Structure

Hierarchical Structure:

- The tree organizes items in a hierarchy based on their frequency. Each branch of the tree represents items co-occurring in transactions.

A. FP-Tree Structure

Node Paths:

- Each node in the FP-Tree represents an item and a counter indicating the frequency of that item within transactions along that path.

B. Steps for Building the FP-Tree

1. **Count item frequency** in the data.
2. **Sort items by frequency** (descending order) to prioritize frequent items at the root, which helps compress the tree
3. **Insert transactions** into the FP-Tree, following the sorted order and incrementing counters for items that already exist along the paths.
4. **Create header tables** that link to each unique item in the tree. These headers serve as entry points for efficiency accessing and traversing nodes associated with each item.

C. Pattern Extraction Using Conditional FP-Trees

1. Once the FP-Tree is constructed, FP-Growth extracts patterns by constructing conditional FP-Trees for each item.

2. For each item at the bottom of the FP-Tree, the algorithm collects prefix paths (paths leading to that item) and generates new trees to find frequent patterns recursively.

D. Recursive Mining Process

1. The FP-Growth algorithm mines the tree recursively by iterating over items in the header table.
2. By examining each item's conditional tree, FP-Growth generates frequent itemsets without ever generating large candidate sets explicitly.

Advantages of FP-Growth over Apriori

a. Avoids Candidate Generation
   - unlike Apriori, which generates and tests candidate itemsets at each level (1-itemset, 2-itemset, etc.). FP-Growth avoids this step altogether, significantly reduces the computational load for large data.

Advantages of FP-Growth over Apriori

b. Compact Representation of Data

- FP-Growth compresses the transaction database into an FP-Tree, allowing it to store the same data in a much smaller format. This is particularly helpful for dense datasets, where items frequently co-occur.

Advantages of FP-Growth over Apriori

c. Recursive, Divide-and-Conquer Approach

- The divide-and-conquer strategy used by FP-Growth helps in breaking down complex problems into smaller, more manageable parts, improving speed and memory efficiency.

Feature	FP-Growth	Apriori
Methodology	Uses a compact FP-Tree to mine patterns without candidate generation.	Uses candidate generation with a join-and-prune approach.
Database Scans	Requires only two database scans.	Requires multiple scans (one for each k-itemset).
Efficiency	Faster, especially with dense data.	Slower due to candidate generation, especially with large datasets.
Memory Usage	More memory-efficient, stores data compactly in a tree.	Memory-intensive due to candidate sets and multiple passes.
Best Use Cases	Works well for dense, large databases with many frequent patterns.	Suitable for small to medium-sized, sparse datasets.

1. Count Item Frequencies

Consider a database of five transactions with minimum support of 2:

Transaction ID	Items
T1	{A, B, D, E}
T2	{B, C, E}
T3	{A, B, C, E}
T4	{B, C}
T5	{A, C, D}

FP-Tree Construction:

1. Count frequencies**:** *{A: 3, B: 4, C: 4, D: 2, E: 3}*
2. Sort by frequency
3. Insert each transaction into the FP-Tree following the sorted order

2. Sort Items in Each Transactions

Consider a database of five transactions with minimum support of 2:

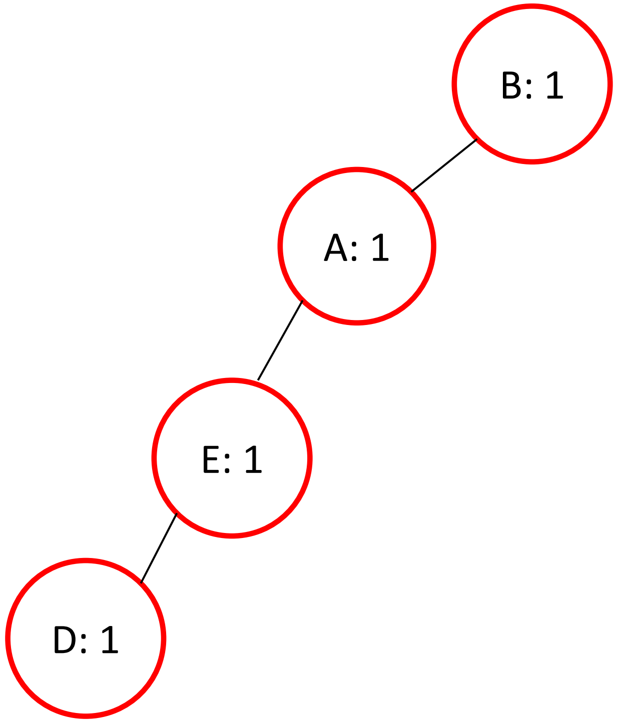
Transaction ID	**Items (Sorted by Frequency)**
T1	{B, A, E, D}
T2	{B, C, E}
T3	{B, C, A, E}
T4	{B, C}
T5	{C, A, D}

Count frequencies: *{A: 3, B: 4, C: 4, D: 2, E: 3}* (For reference only)

3. Insert Transactions into the FP-Tree

Transaction T1: ({B, A, E, D})

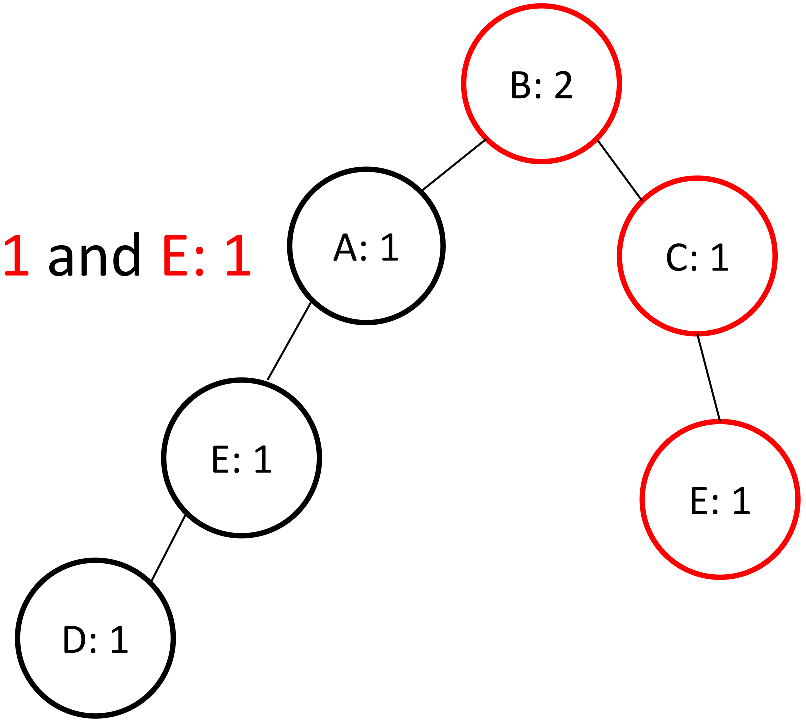
Insert path B → A → E → D
Set counters: B: 1, A: 1, E: 1, D: 1



3. Insert Transactions into the FP-Tree

Transaction T2: ({B, C, E})

Insert path B → C → E
B exists already, so increment B to 2
C and E are new in this branch, add C: 1 and E: 1

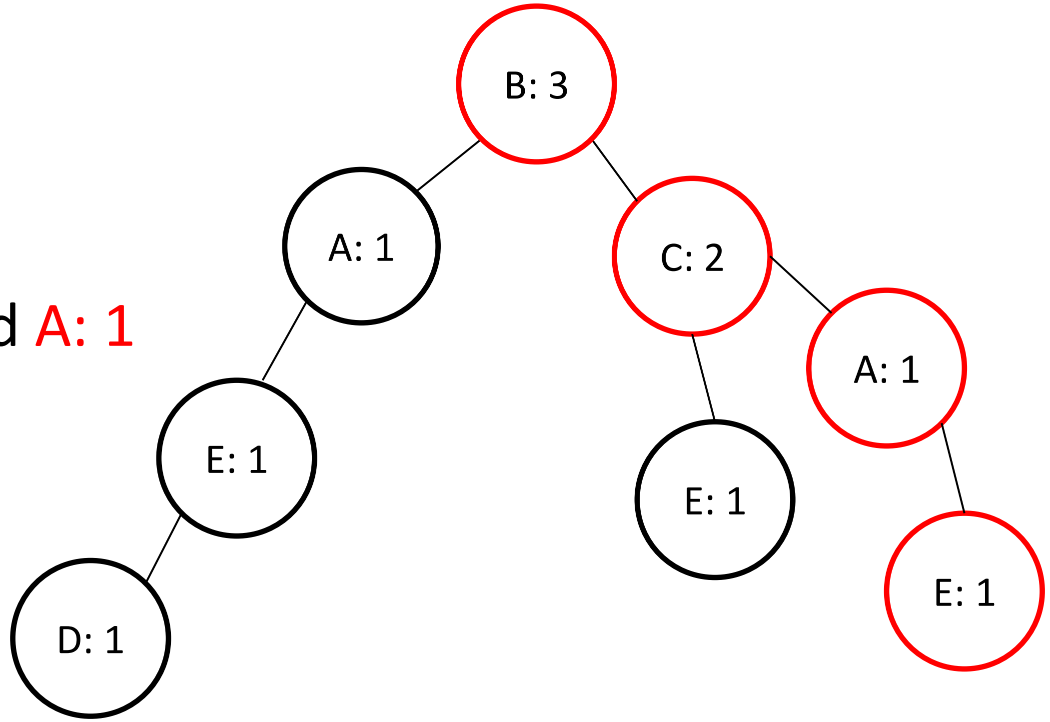


3. Insert Transactions into the FP-Tree

Transaction T3**:** ({B, C, A, E})

B & C already exists in this branch, so increment B to 3 and C to 2

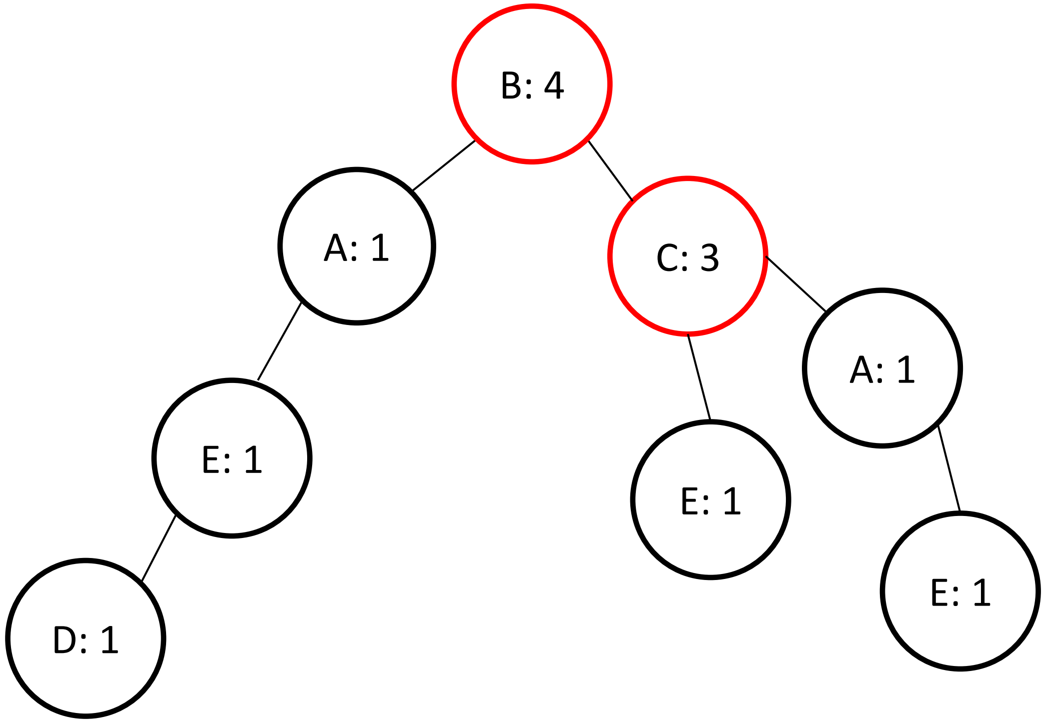
A and E are new in this branch, so add A: 1 and E: 1



3. Insert Transactions into the FP-Tree

Transaction T4**:** ({B, C})

Insert path B → C
B already exists, so increment **B to 4**,
increment **C to 3**

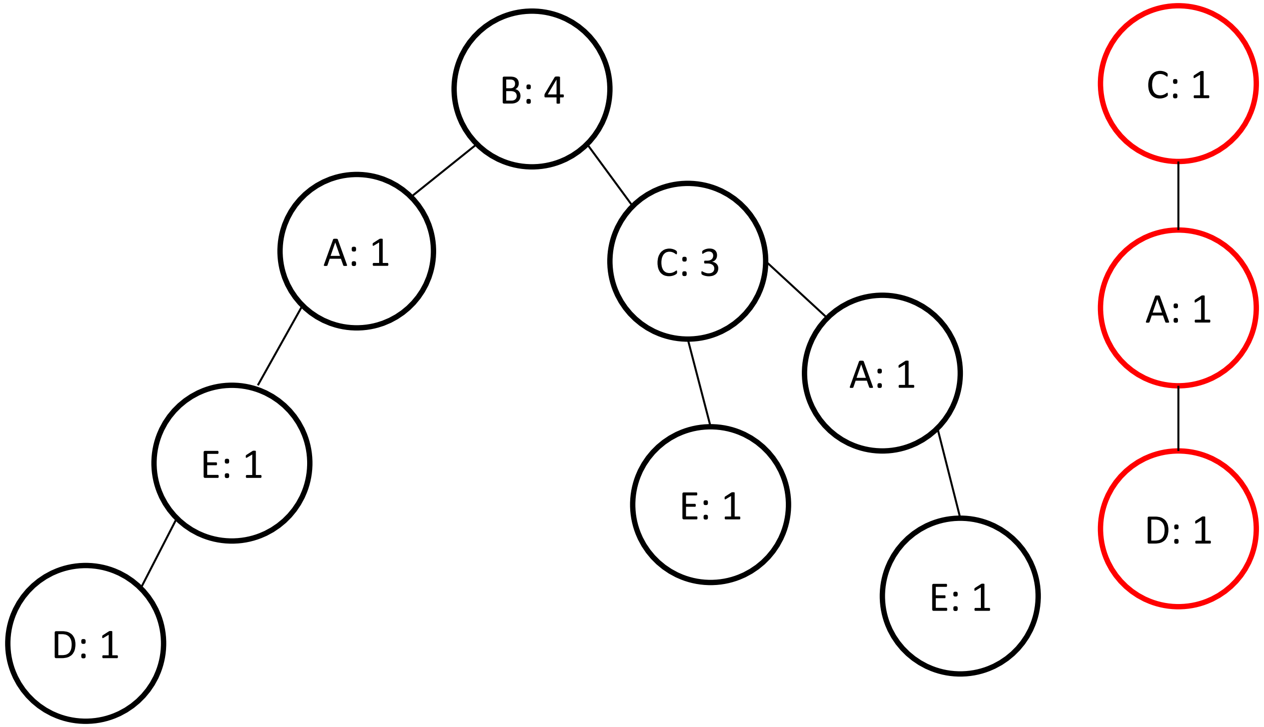


3. Insert Transactions into the FP-Tree

Transaction T5: ({C, A, D})

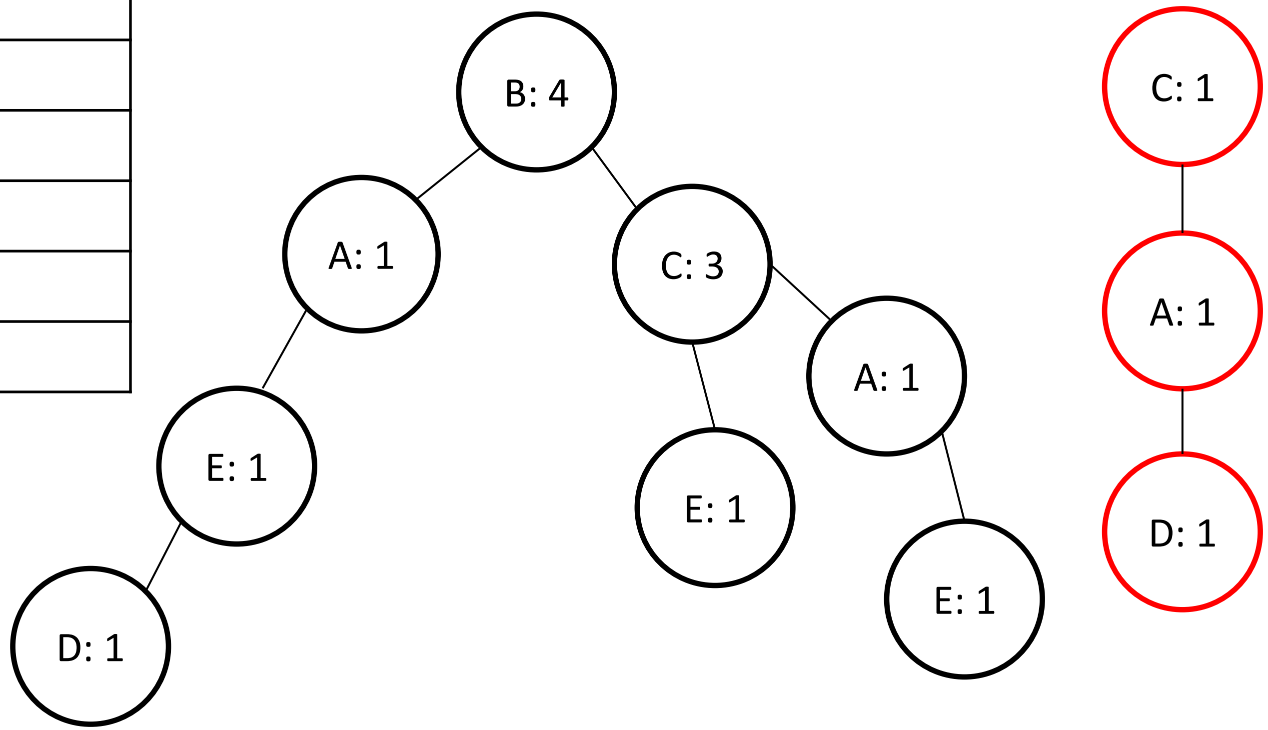
Start a new branch C as the first item

Create path C → A → D



4. Header Table

Item	Frequency	Pointer to First Node
B	4	First B node in tree
C	4	First C node in tree
A	3	First A node in tree
E	3	First E node in tree
D	2	First D node in tree



5.1 Mining the FP-Tree for Frequent Patterns

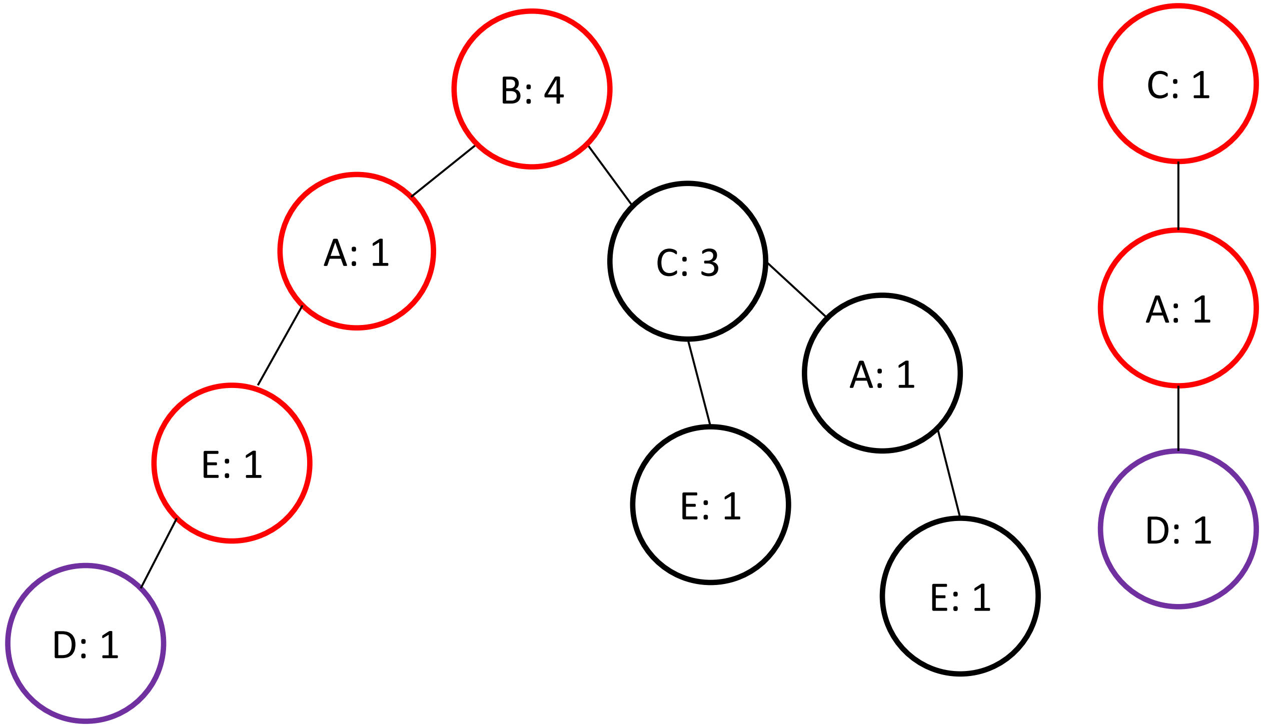
Paths Leading to D Nodes:

1. Path 1 containing D:
   - **B → A → E → D (support count = 1)**

2. Path 2 containing D:
   - **C → A → D (support count = 1)**

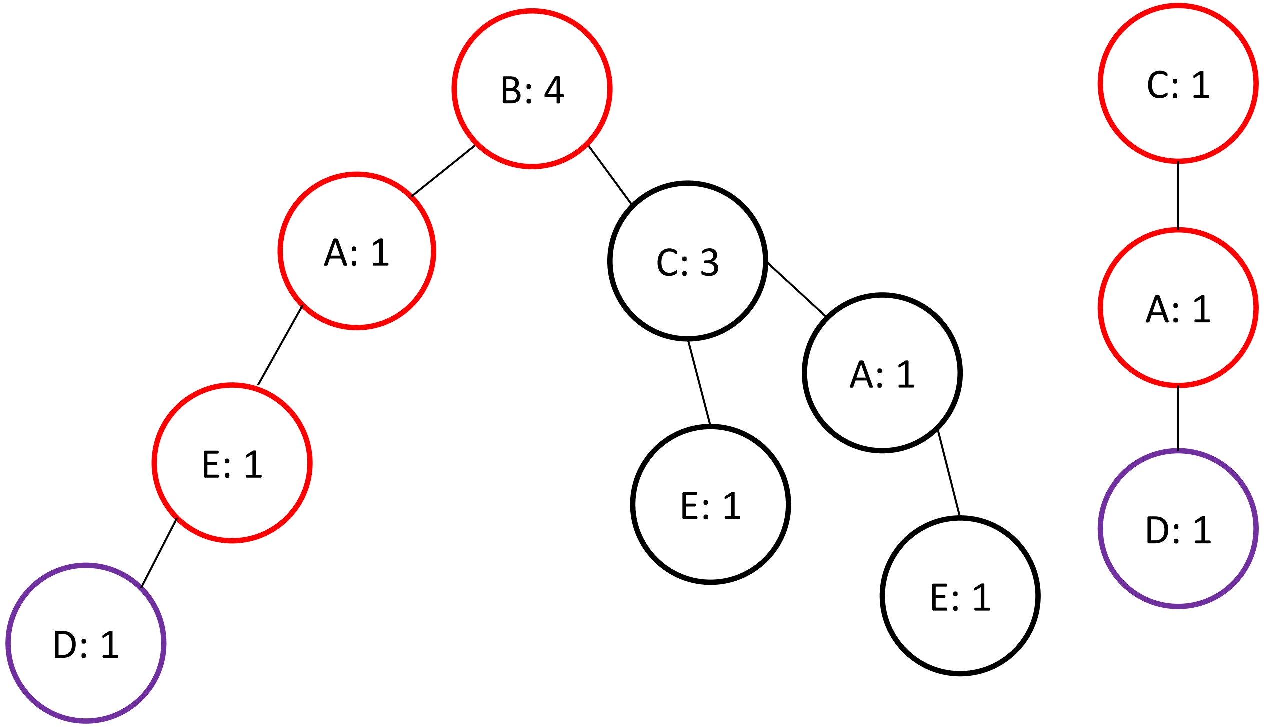
Conditional Pattern Base on D:

B, A, E	(1)
C, A	(1)



Frequent Pattern for Node D

Pattern	Support Count
{D}	2
{A, D}	2
{E, D}	1
{A, E, D}	1
{B, A, E, D}	1
{C, A, D}	1



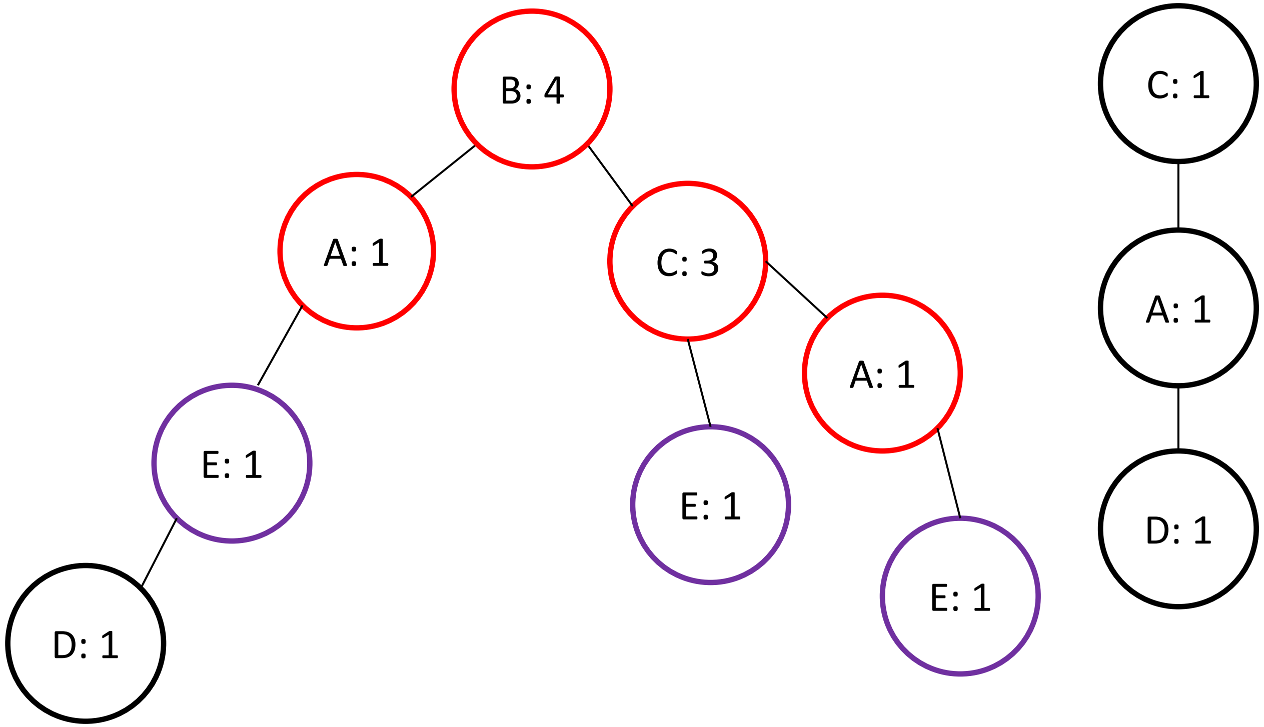
5.2 Mining the FP-Tree for Frequent Patterns

Paths Leading to E Nodes:

1. Path 1 containing E:
   - **B → A → E (support count = 1)**
2. Path 2 containing E:
   - **B → C → E (support count = 1)**
3. Path 2 containing E:
   - **B → C → A → E (support count = 1)**

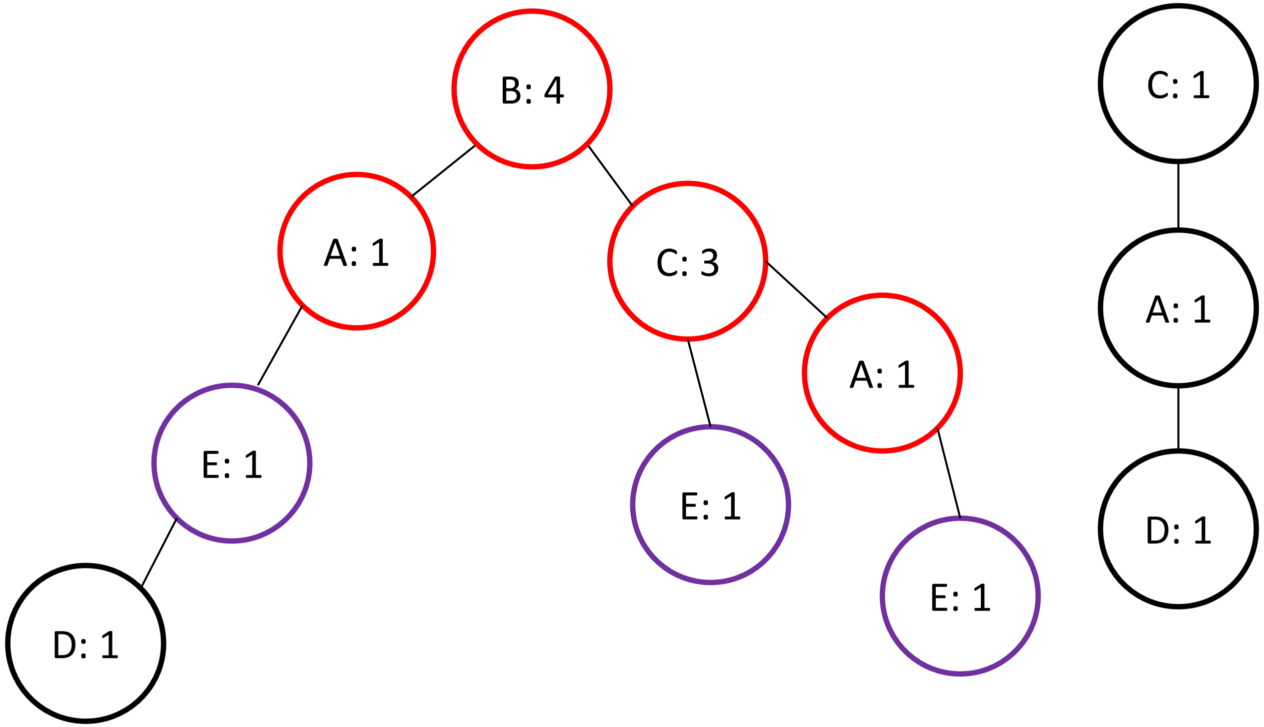
Conditional Pattern Base on E:

B, A	(1)
B, C	(1)
B, C, A	(1)



Frequent Pattern for Node E

Pattern	Support Count
{E}	3
{B, E}	3
{B, A, E}	2
{B, C, E}	2
{B, C, A, E}	1



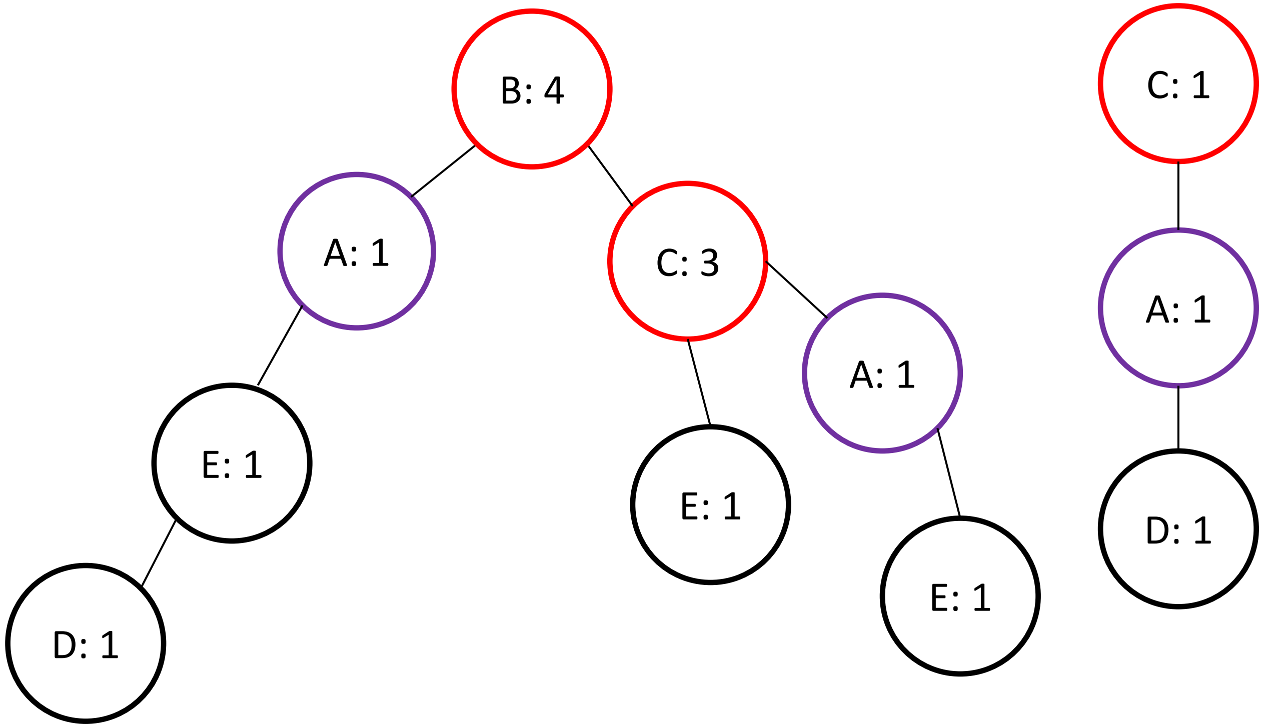
5.3 Mining the FP-Tree for Frequent Patterns

Paths Leading to A Nodes:

1. Path 1 containing A:
   - **B → A (support count = 1)**
2. Path 2 containing A:
   - **B → C → A (support count = 1)**
3. Path 2 containing A:
   - **C → A (support count = 1)**

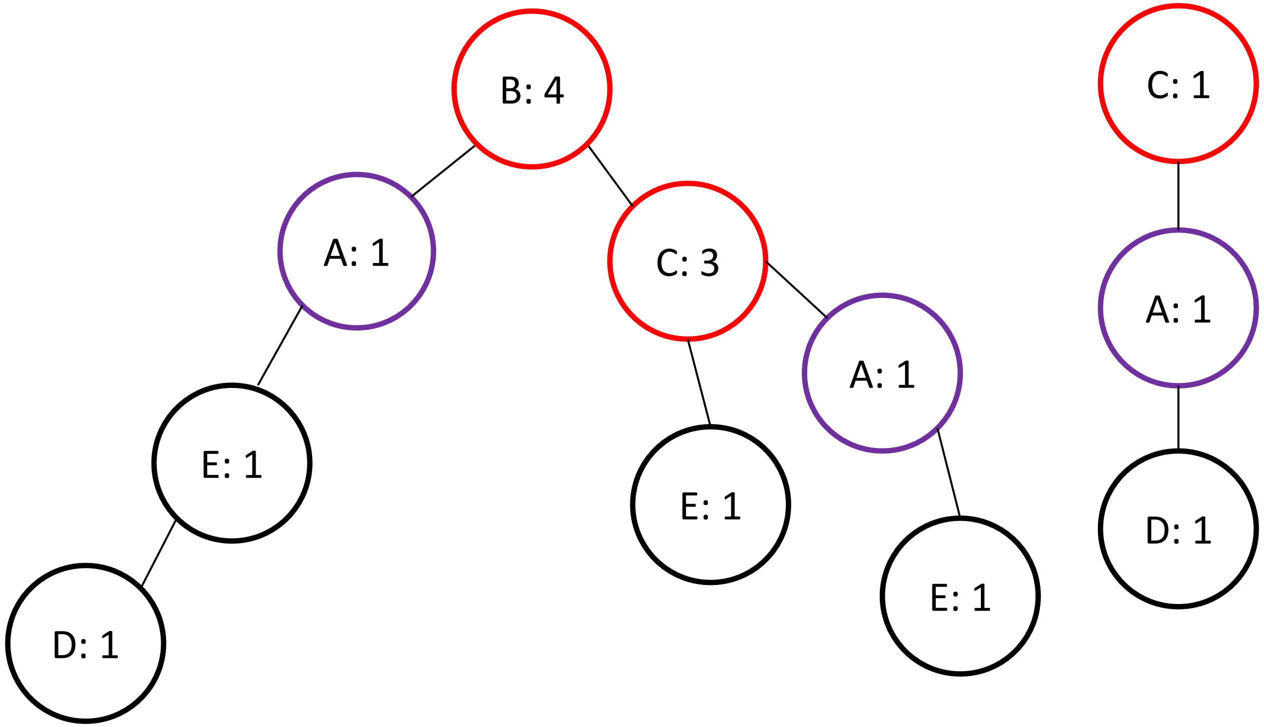
Conditional Pattern Base on A:

B	(1)
B, C	(1)
C	(1)



Frequent Pattern for Node A

Pattern	Support Count
{A}	3
{B, A}	2
{C, A}	2
{B, C, A}	1



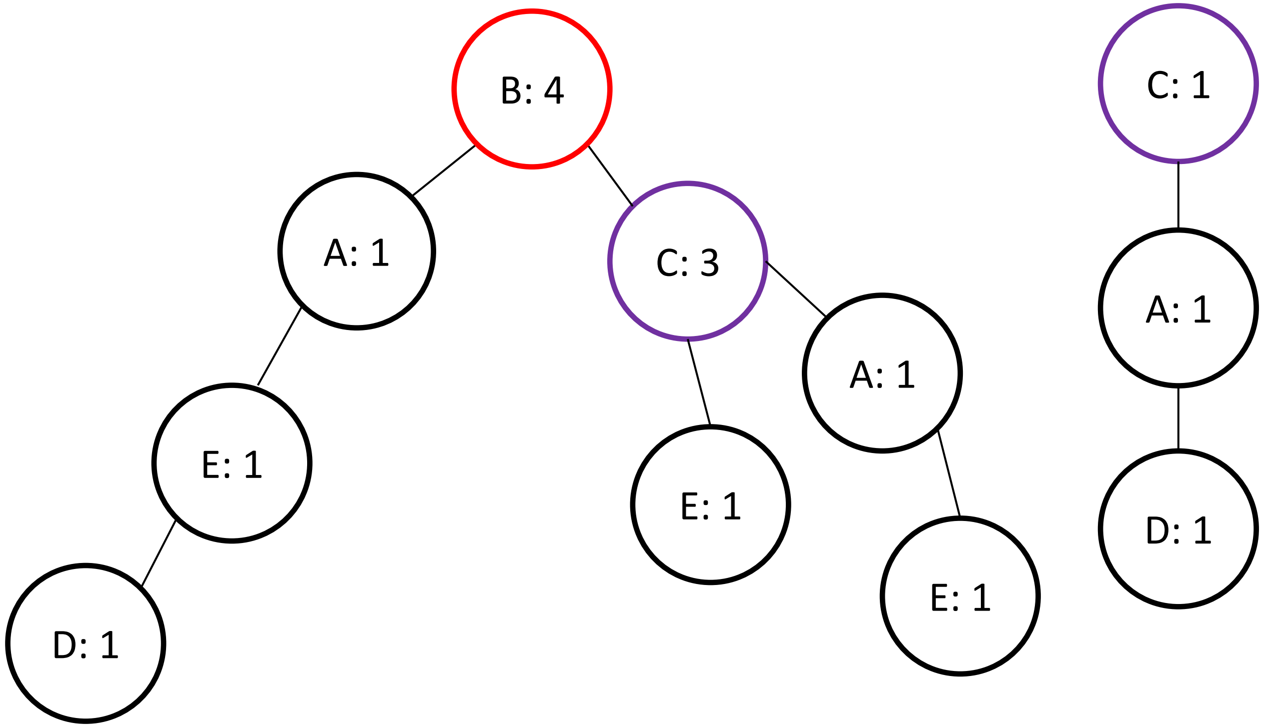
5.4 Mining the FP-Tree for Frequent Patterns

Paths Leading to C Nodes:

1. Path 1 containing C:
   - **B → C (support count = 1)**
2. Path 2 containing C:
   - **Root C (support count = 1)**

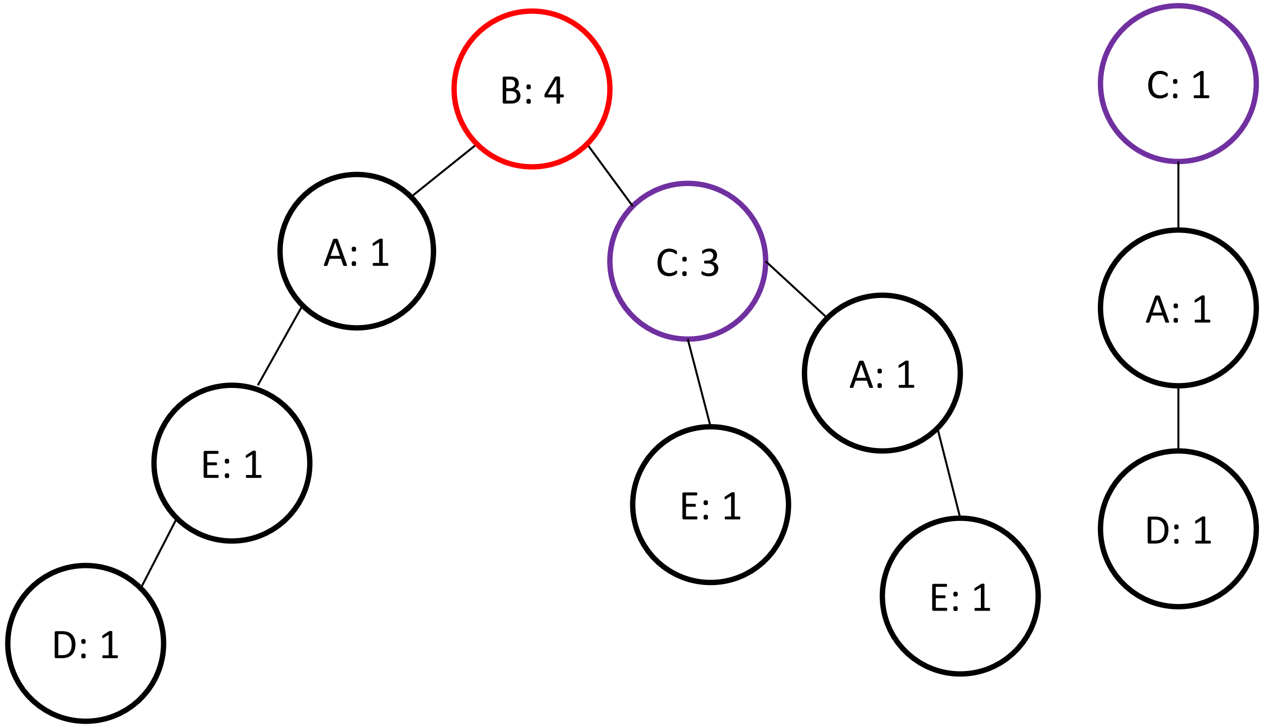
Conditional Pattern Base on C:

B	(3)
C (root)	(1)



Frequent Pattern for Node C

Pattern	Support Count
{C}	4
{B, C}	3



5.4 Mining the FP-Tree for Frequent Patterns

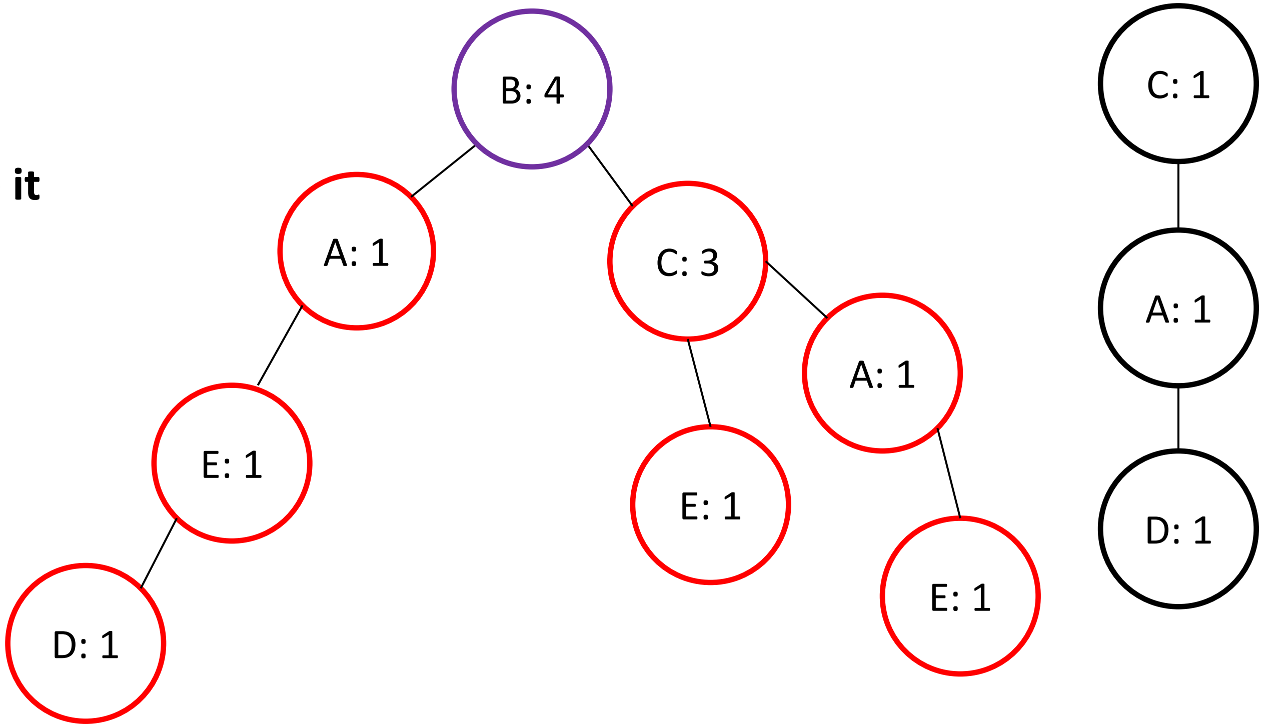
Paths Leading to B Nodes:

1. Path 1 containing B:

   - **B is the root ode, all paths inherently include it**

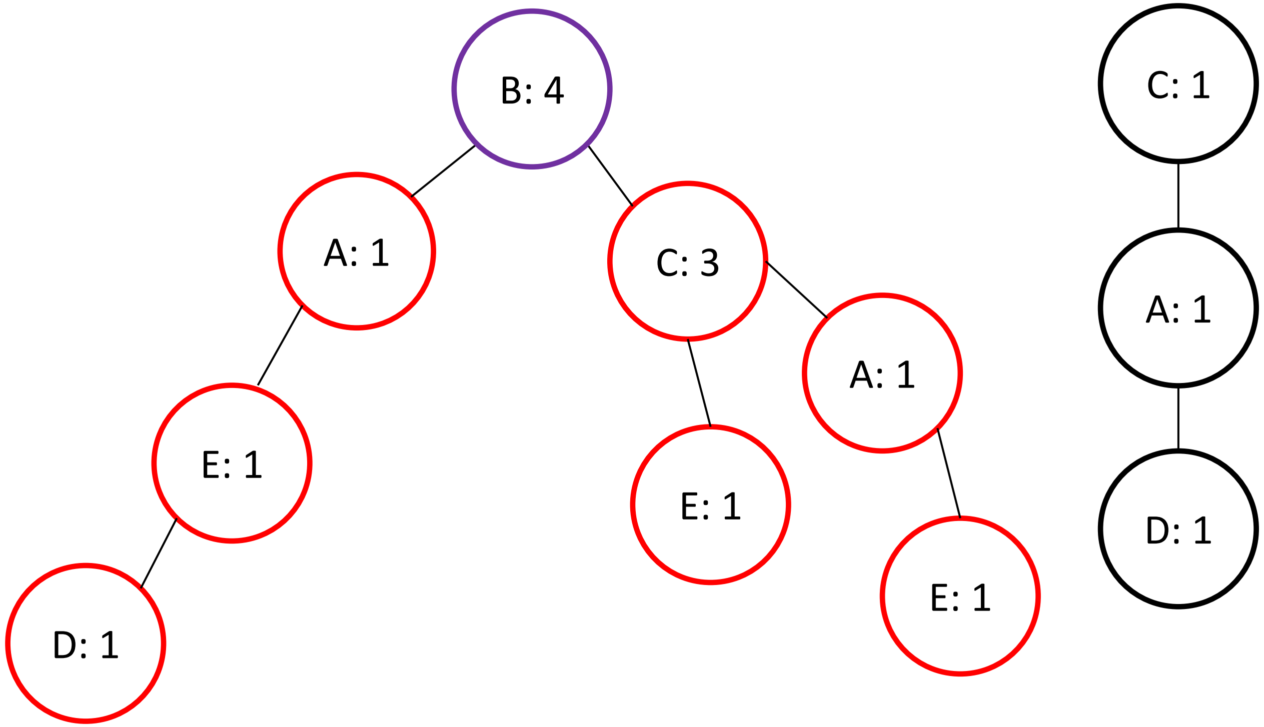
Conditional Pattern Base on B:

A	(1)
C	(3)
A, E, D	(1)
C, E	(1)
C, A, E	(1)



Frequent Pattern for Node B

Pattern	Support Count
{B}	4
{B, A}	2
{B, C}	3
{B, E}	3
{B, C, E}	2
{B, C, A}	1
{B, A, E}	2
{B, A, E, D}	1



Thank you very much for listening.