* **FP (frequent pattern)-Growth Algorithm**

**Overview**

An efficient algorithm for mining **frequent item sets**. It constructs a compact data structure called the **FP-tree** to represent the dataset and recursively mines frequent item sets from this structure. FP-Growth avoids the generation of candidate item sets and is **known for its scalability**.

Dataset

|  |  |
| --- | --- |
| TID | items |
| 01 | Shrimp, almonds, avocado, honey, Eggs |
| 02 | Eggs, Shrimp |
| 03 | Honey, almonds, avocado, Eggs |
| 04 | Shrimp, honey, Eggs, avocado |
| 05 | Eggs, Shrimp, avocado |

We encode items for simplicity

|  |  |
| --- | --- |
| TID | items |
| 01 | A, B, C, D,E |
| 02 | E, A |
| 03 | D, B, C,E |
| 04 | A, D, E, C |
| 05 | E, A, C |

**Steps**

1. **Scan the dataset**

In this step, the algorithm scans the input dataset to determine the frequency of each item. This determines the order in which items are added to the FP tree, with the most frequent items added first.

The result of this step is a table with distinct items and their frequencies.

|  |  |
| --- | --- |
| Item | Frequency |
| A | 4 |
| B | 2 |
| C | 4 |
| D | 3 |
| E | 3 |

1. **Sort & prone items**

In this step, the items in the dataset are sorted in acescending order of frequency. The infrequent items that do not meet the minimum support threshold are removed from the dataset. This helps to reduce the dataset's size and improve the algorithm's efficiency.

The result of this step is an order list of frequent items.

Consider the minimum support to be 3

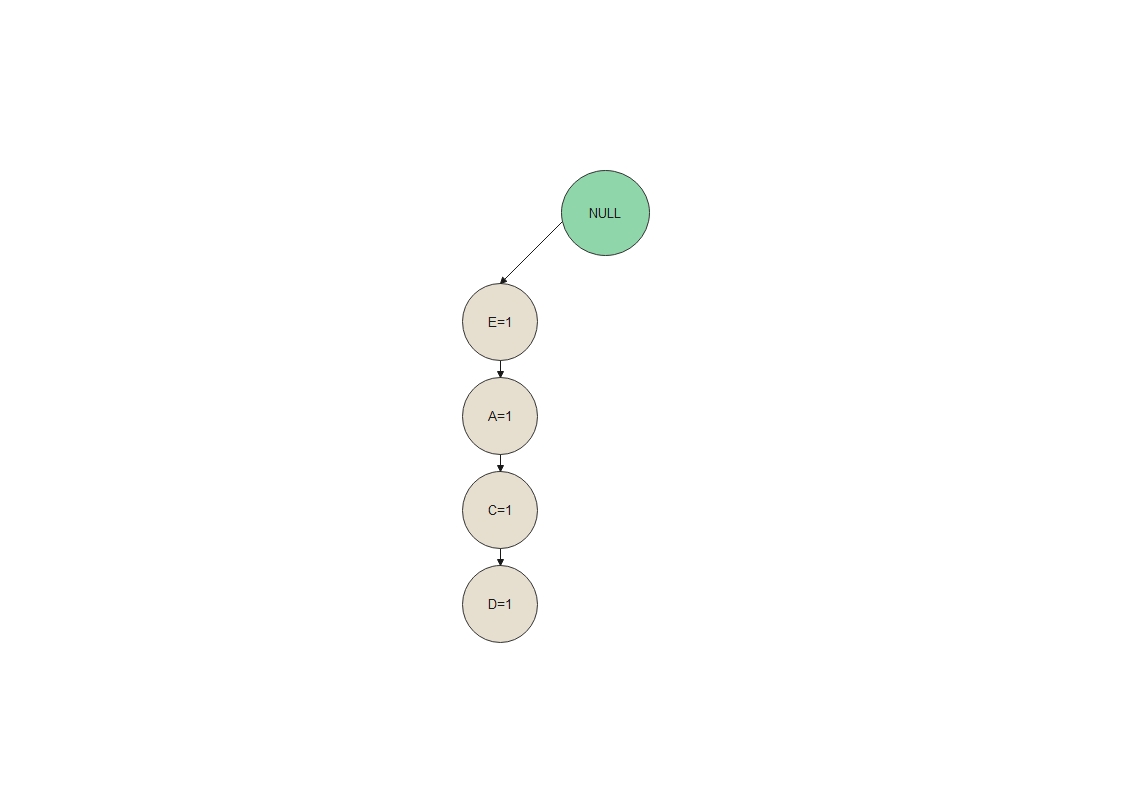
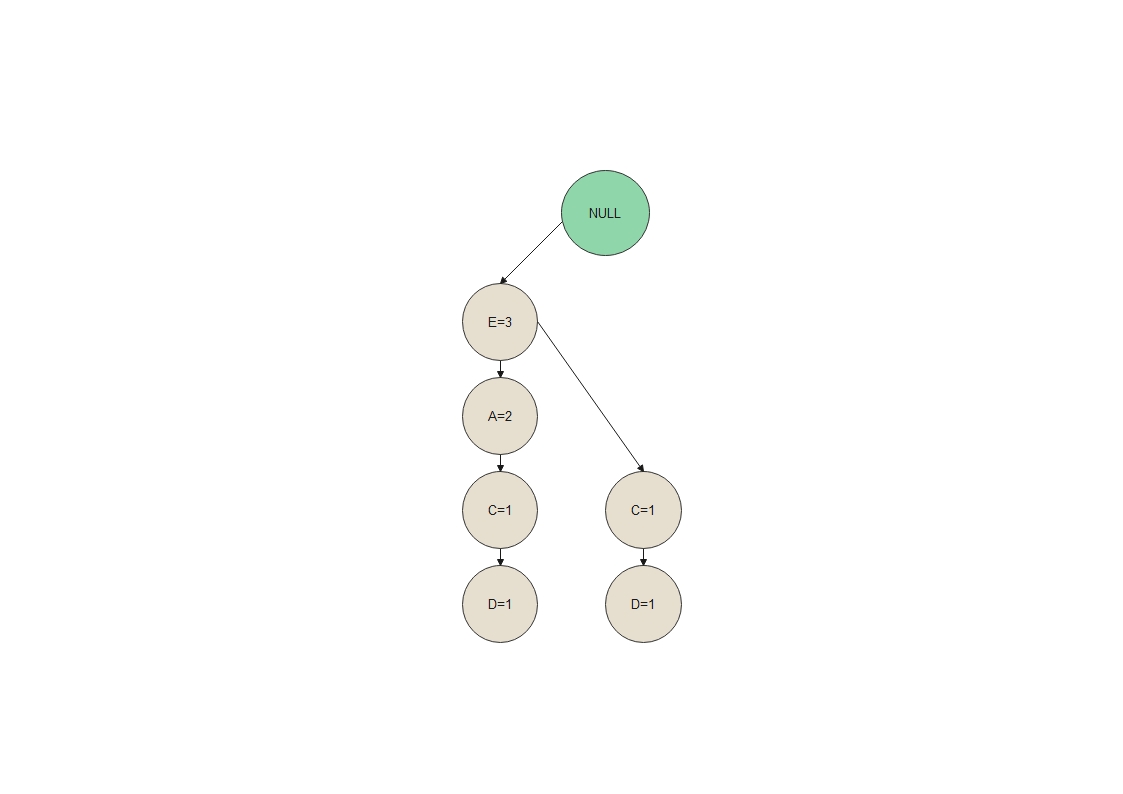
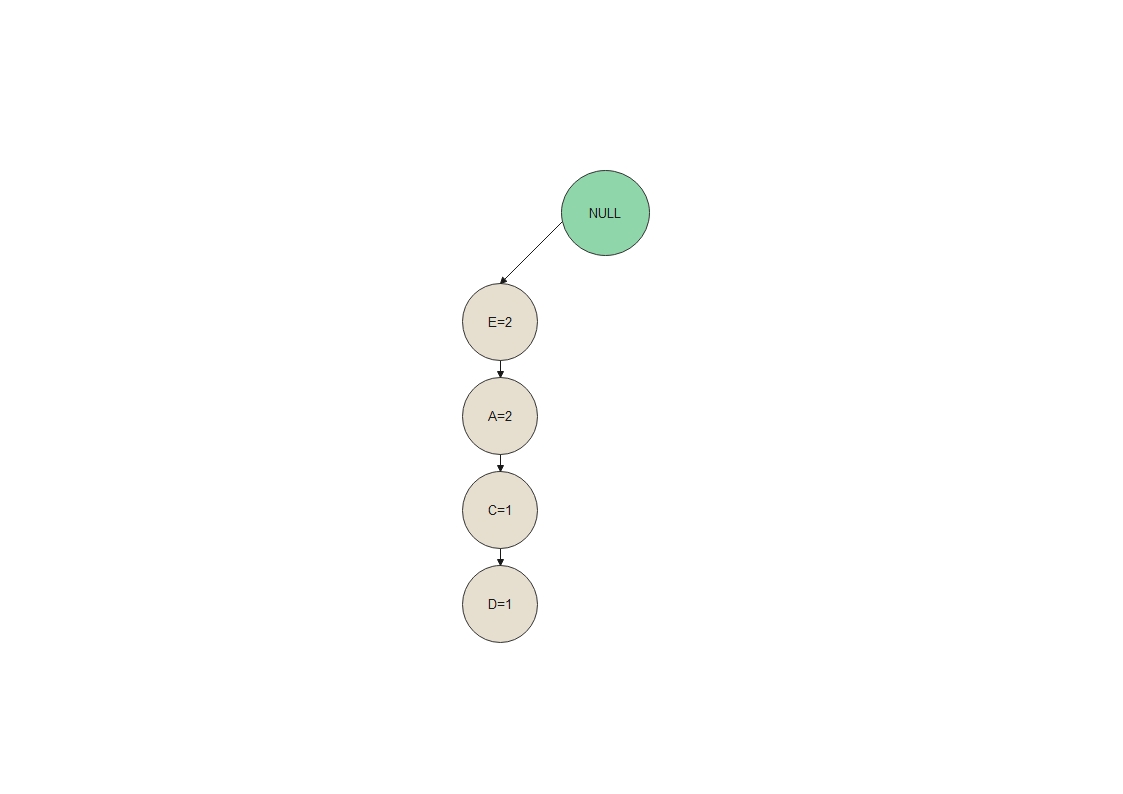
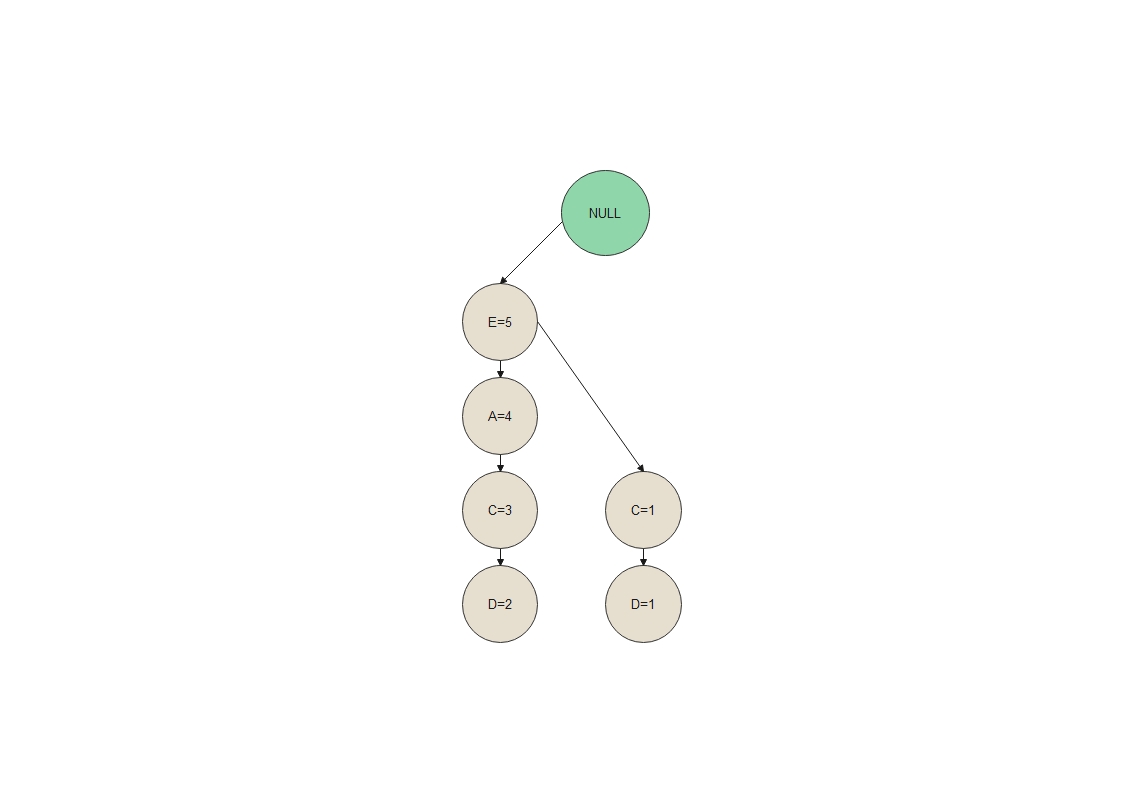
|  |  |
| --- | --- |
| Item | Frequency |
| E | 5 |
| A | 4 |
| C | 4 |
| D | 3 |

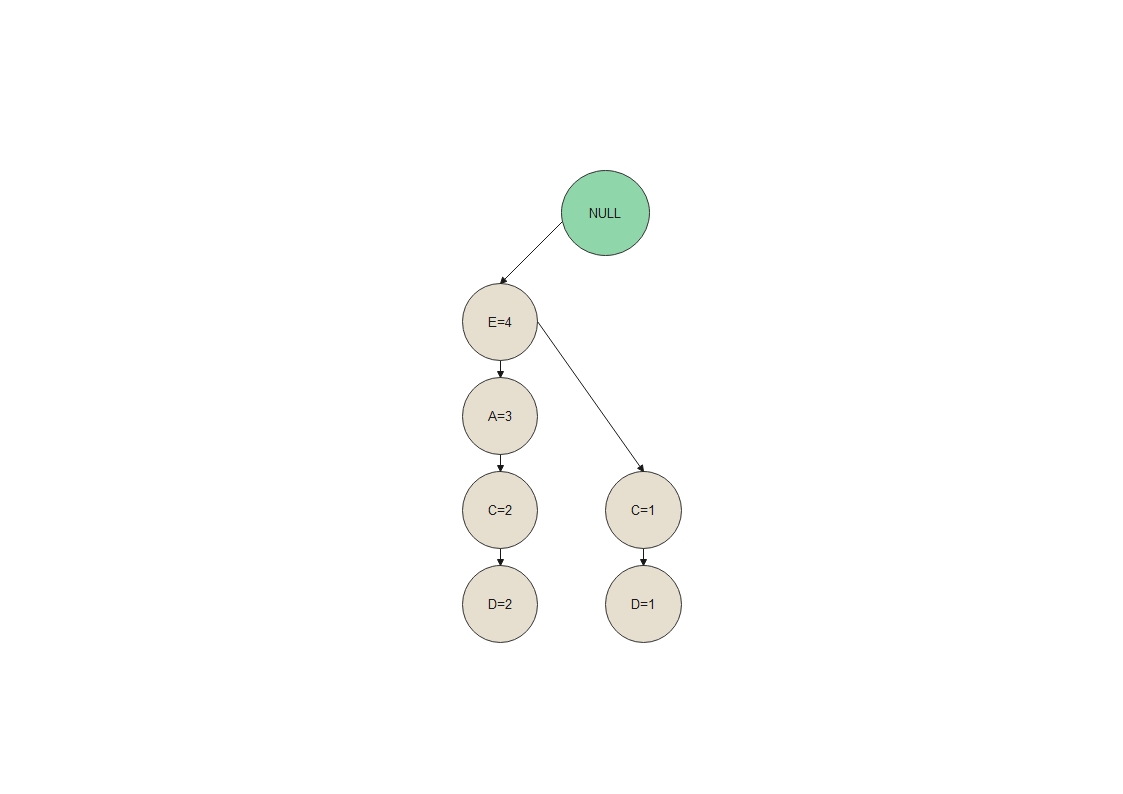
1. **Re-order item sets in the dataset**

In this step, the original item sets will be filtered and reordered based on the result of the previous step.

|  |  |
| --- | --- |
| TID | items |
| 01 | E,A,C,D |
| 02 | E, A |
| 03 | E,C,D |
| 04 | E,A,C,D |
| 05 | E, A, C |

1. **Construct the FP-tree**

In this step, the FP-tree is constructed. The FP-tree is a compact data structure that stores the frequent item sets and their support counts. It is built based on the result of the previous step.



1. **Generate Conditional Pattern Base** **& Conditional FP tree**

The Conditional Pattern Base is computed which is path labels of all the paths which lead to any node of the given item in the frequent-pattern tree. Note that the items in the below table are arranged in the ascending order of their frequencies.

The Conditional Frequent Pattern Tree is built. It is done by taking the set of elements that is common in all the paths in the Conditional Pattern Base of that item and calculating its support count by summing the support counts of all the paths in the Conditional Pattern Base.

|  |  |  |
| --- | --- | --- |
| Items | Conditional Pattern Base | Conditional FP Tree |
| D | {{E,A,C:2},{E,C:1}} | {E:3} |
| C | {{E,A:3},{E:1}} | {E:4} |
| A | {{E:4}} | {E:4} |
| E |  |  |

1. **Generate Frequent Pattern Rules**

The Frequent Pattern rules are generated by pairing the items of the Conditional Frequent Pattern Tree set to the corresponding to the item as given in the below table.

|  |  |  |
| --- | --- | --- |
| Items | Frequent Pattern Rules | Confidence |
| D | {<E,D:3>} | .6/1=.6 |
| C | {<E,C:4>} | .8/1=.8 |
| A | {<E,A:4>} | .8/1=.8 |
| E |  |  |

FP Growth Algorithm Vs. Apriori Algorithm

|  |  |  |
| --- | --- | --- |
|  | FP Growth | Apriori |
| Memory Usage | less memory as it constructs the FP-Tree | a large amount of memory to store candidate itemset |
| Candidate Generation | Generates frequent itemsets by constructing the FP-Tree and recursively generating conditional pattern bases. | Generates candidate itemsets by joining and pruning. |
| Speed | Faster due to efficient data compression and generation of frequent itemsets. | Slower due to multiple database scans and candidate generation. |