Data Mining

Study Assignment Set #6

Reference Books:
Introduction to Data Mining by Tan P. N., Steinbach M and Kumar V.
Pearson Education, 2006

Data Mining: Concepts and Techniques, Second Edition by Jiawei Han and Micheline Kamber
Morgan Kaufmann Publishers, 2006

Topic: FP Growth for generating Frequent Patterns

Question 1	FP Tree

Learning objectives:

- **In this study assignment we focus on mining frequent itemsets without candidate generation.**

Prerequisites:

- **Itemsets, Frequent Itemsets, Support, Confidence, Apriori Algorithm etc.**

Reference:

- **Chapter 6, Section 6.2.4 of Data Mining: Concepts and Techniques, Second Edition by Jiawei Han and Micheline Kamber**

About FP-Growth technique for mining Frequent Items.

The FPGrowth Approach (J. Han, J. Pei, and Y. Yin, SIGMOD' 00)

- **Depth-first search**
- **Avoid explicit candidate generation**

Major philosophy: Grow long patterns from short ones using local frequent items only

- **"abc" is a frequent pattern**
- **Get all transactions having "abc", i.e., project DB on abc: DB|abc**
- **"d" is a local frequent item in DB|abc abcd is a frequent pattern**

Example: Chapter 6, Section 6.2, Table 6.1
Let's take an example to understand the FP-Growth, construction of FP-Trees.

Transaction Database: (Usually called DB)

TID (Transaction ID)	List of Item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Step 1: Select the minimum support count.

In this example, let the minimum support count be 2.

Step 2: Scan the DB for frequent items (1-itemsets) and their support counts (frequencies)

Item_ID	Support Count
I1	6
I2	7
I3	6
I4	2
I5	2

Step 3: Sort the set of frequent items in the order of descending support count. (Usually, the sorted list is denoted by L.)

Item_ID	Support Count
I2	7
I1	6
I3	6
I4	2

I5	2

Step 4: Scan the DB to rearrange the itemsets in the L order. We will be using this DB for constructing the FP-Tree.

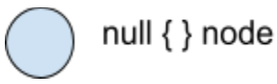
Transaction Database: Arrange the items in the L-order. Refer Step 3.

TID (Transaction ID)	List of Item_IDs	List of Item_IDs (L Order)
T100	I1, I2, I5	I2, I1, I5
T200	I2, I4	I2, I4
T300	I2, I3	I2, I3
T400	I1, I2, I4	I2, I1, I4
T500	I1, I3	I1, I3
T600	I2, I3	I2, I3
T700	I1, I3	I1, I3
T800	I1, I2, I3, I5	I2, I1, I3, I5
T900	I1, I2, I3	I2, I1, I3

Step 5: Construction of the FP-Tree.

The FP-Tree construction is simple.
While constructing the FP-Tree we refer to the L-Order DB (List of Item_IDs (L Order) in Step 4).

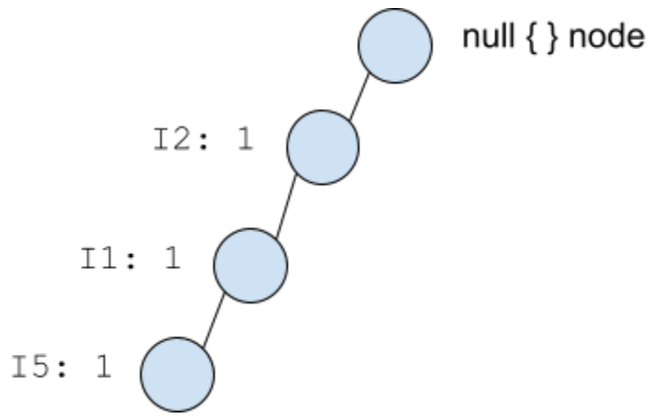
Step 5.0: Create a "null" node. This is root node of the FT-Tree



Step 5.1: Construct the branches considering the first transaction
T100 - I2, I1, I5

Tree Traversal: Root -> I2 -> I1 -> I5

Notation of each node: <Node Item: Node count>
T100 - <I2: 1>, <I1: 1>, <I5: 1>



Do not get confused by the support count! In the T100, there are just 3 items. `<I2: 1>, <I1: 1>, <I5: 1>`

Step 5.2: Construct the branches considering the 2nd transaction

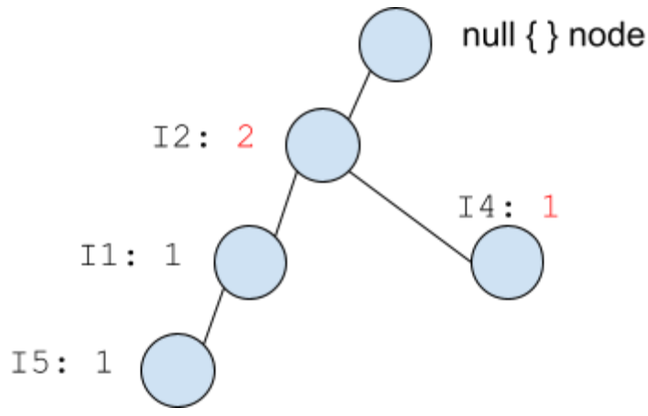
`T200 - I2, I4`

Now, we add the second transaction to the tree above.

Tree Traversal: `Root -> I2 -> I4`

Let's take `I2` from T200 and traverse from the root node. The tree has `I2 node`. Now increment its count.

Let's take `I4` from T200 and traverse from the I2 node. Currently there is no I4 branch. So, construct a new branch and node. `<I4: 1>`



Note: Observe the `I2 node` count is incremented, and a new node `I4 with count 1` is added.

Step 5.3: Construct the branches considering the 3rd transaction

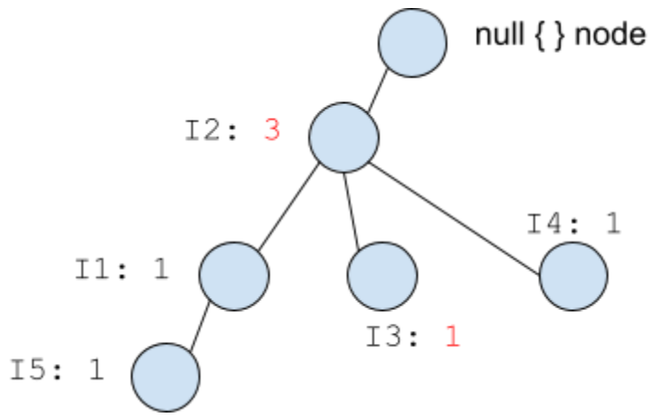
`T300 - I2, I3`

Now, we add the 3rd transaction to the tree above.

Tree Traversal: `Root -> I2 -> I3`

Let's take `I2` from T300 and traverse from the root node. The tree has `I2 node`. Now increment its count.

Let's take `I3` from T300 and traverse from the I2 node. Currently there is no I3 branch. So, construct a new branch and node. `<I3: 1>`



Note: Observe the `I2 node` count is incremented, and a new node `I3 with count 1` is added.

Step 5.4: Construct the branches considering the 4th transaction

`T400 - I2, I1, I4`

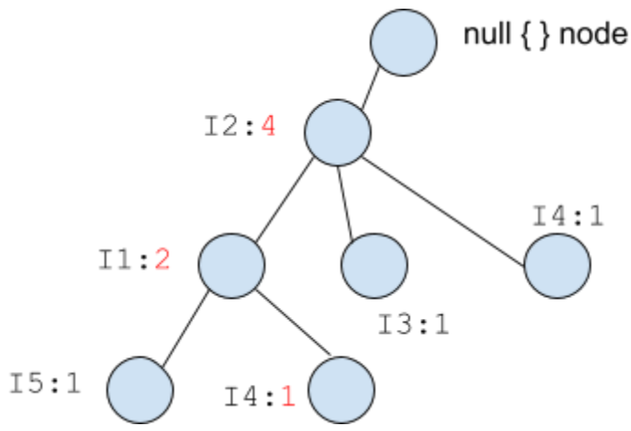
Now, we add the 4th transaction to the tree above.

Tree Traversal: `Root -> I2 -> I1 -> I4`

Let's take `I2` from T400 and traverse from the root node. The tree has `I2 node`. Now increment its count.

Let's take `I1` from T400 and traverse from the `I2` node. Let's take `I3` from T300 and traverse from the `I1` node.

Let's take `I4` from T400 and traverse from the `I2 -> I1` node. Currently there is no `I4` branch. So, construct a new branch and node. `<I4: 1>`



Note: Observe the `I2, I1 node` counts are incremented, and a new node `I4 with count 1` is added. `Root -> I2 -> I1 -> I4`

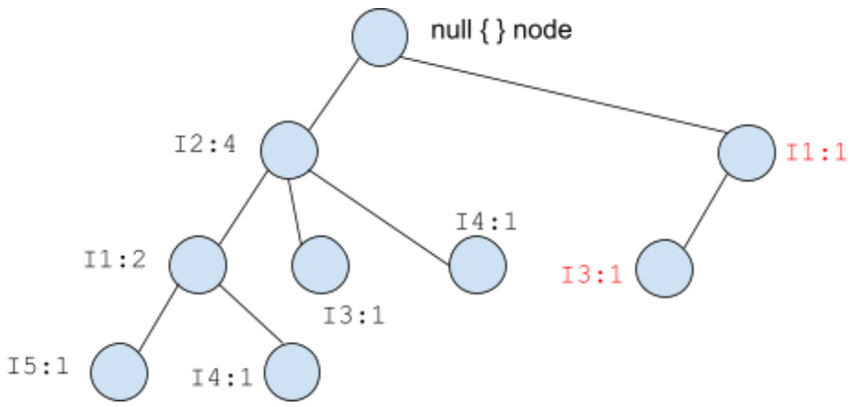
Step 5.5: Construct the branches considering the 5th transaction

`T500 - I1, I3`

Now, we add the 5th transaction to the tree above.

Tree Traversal: `Root -> I1 -> I3`

There is no branch `Root -> I1 -> I3`. So, we have to add a new branch to the tree as below.



Note: Observe the `I1, I3 nodes` are added to the roor.
`Root -> I1 -> I3`

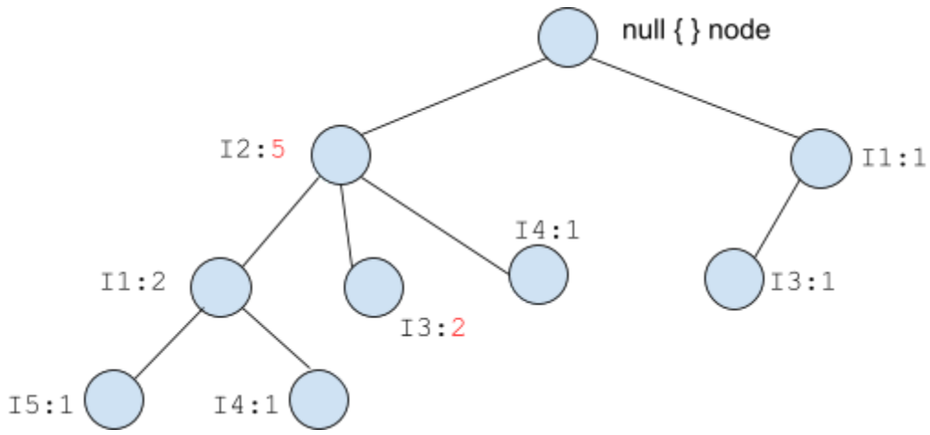
Step 5.6: Construct the branches considering the 6th transaction

`T600 - I2, I3`

Now, we add the 6th transaction to the tree above.

Tree Traversal: `Root -> I2 -> I3`

`You can observe that the tree has I2 -> I3 branch. So, we need to just increment the node count.`



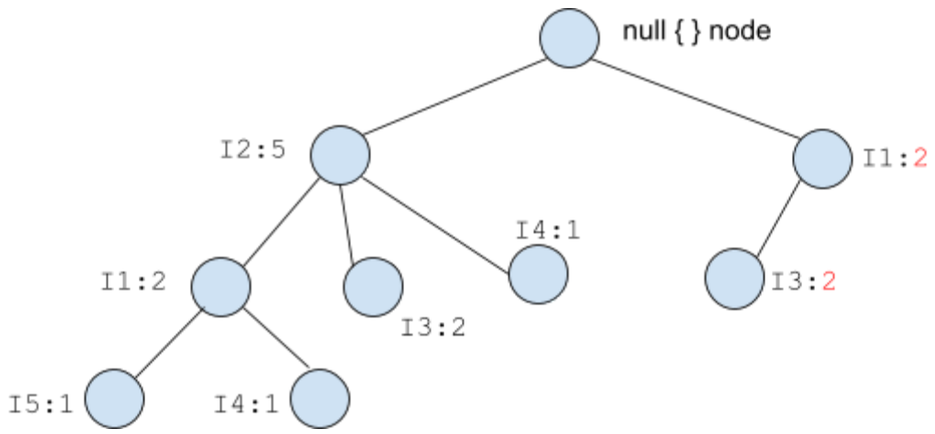
Step 5.7: Construct the branches considering the 7th transaction

`T700 - I1, I3`

Now, we add the 6th transaction to the tree above.

Tree Traversal: `Root -> I1 -> I3`

`You can observe that the tree has a Root-> I1 -> I3 branch. So, we need to just increment the node count.`



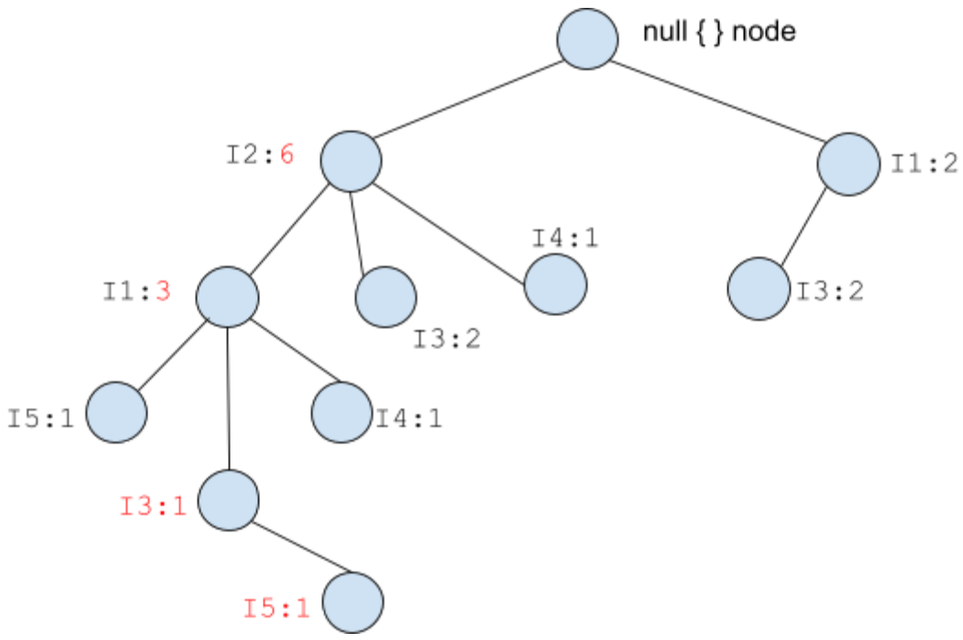
Step 5.8: Construct the branches considering the 8th transaction

T800 - I2, I1, I3, I5

Now, we add the 6th transaction to the tree above.

Tree Traversal: Root -> I2 -> I1 -> I3 -> I5

You can observe Root -> I2 -> I1 branch exists. We need to just increment the node count of I2 and I1. However, we need to create new nodes I3 -> I5 linked to I1 (Root -> I2 -> I1)



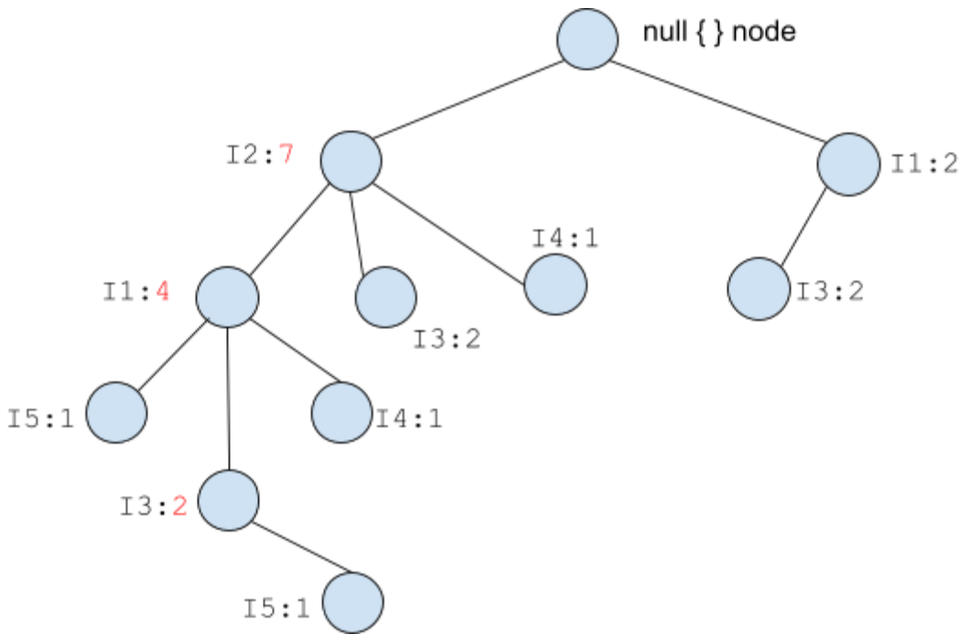
Step 5.9: Construct the branches considering the 9th transaction

T900 - I2, I1, I3

Now, we add the 6th transaction to the tree above.

Tree Traversal: Root -> I2 -> I1 -> I3

You can observe the Root -> I2 -> I1 -> I3 branch exists. We need to just increment the node count of I2, I1 and I3.

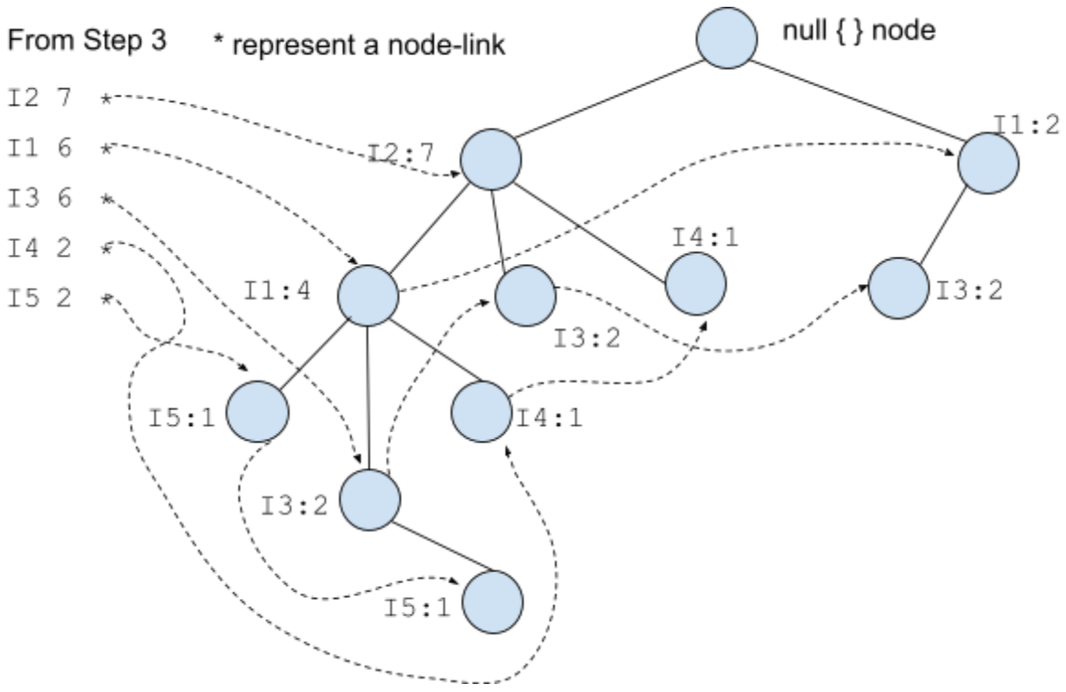


FP-Tree was created for the 9 transactions.

STEP 6: Mining the FP-Tree.

Now we need to mine the FT-Tree created above for Frequent Patterns.

Step 6.0:

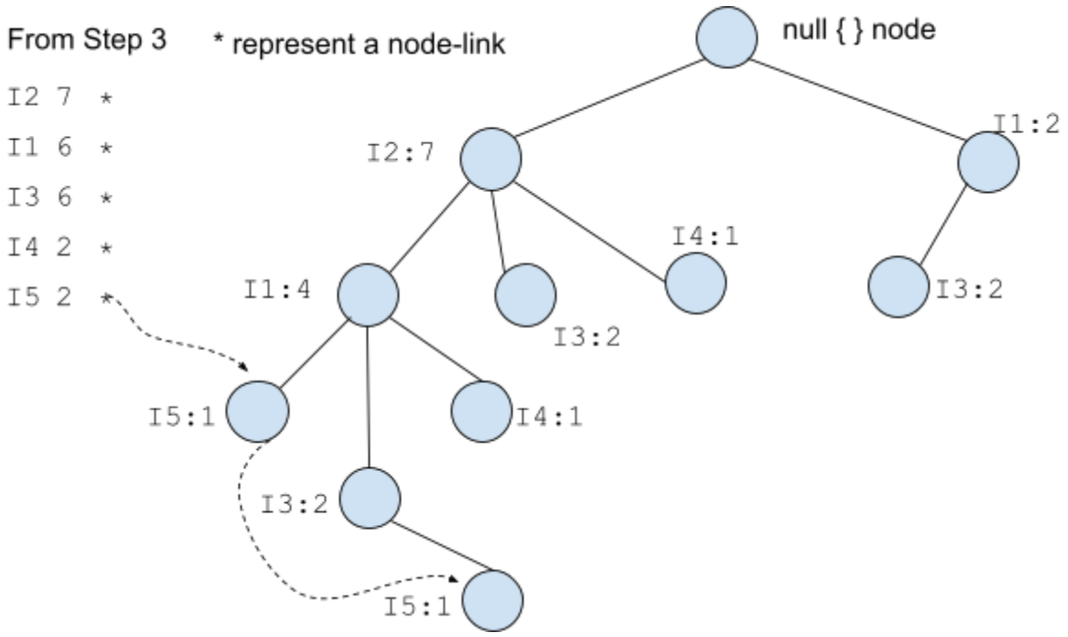


Please observe the L-Order list has a link to the node.

Step 6.1: Construct Conditional FP-Tree with the conditional node I5

Note: Start from the last item in the L-Order List.

Follow the node-link path - `I5` as below.



How to generate a conditional pattern base?

Consider `I5` as suffix, generate the Conditional Pattern Base and Conditional FP-Tree

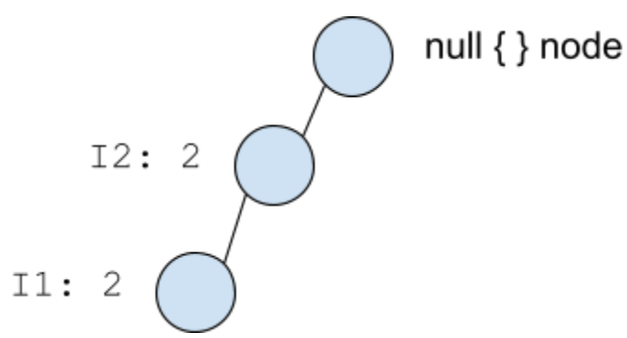
```
I5:1 linked to I2 -> I1
I5:1 linked to I2 -> I1 -> I3
```

Item	Conditional Pattern Base	Conditional FP-Tree	Frequent Patterns generated
I5	{ {I2, I1: 1}, {I2, I1, I3: 1} }	<I2: 2, I1: 2>	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5:2}

How to generate a conditional FP-Tree? Prefix Paths.

Use the Conditional Pattern Base and generate Conditional FP-Tree as below.

Since the support count is 2, `I3` is ignored.



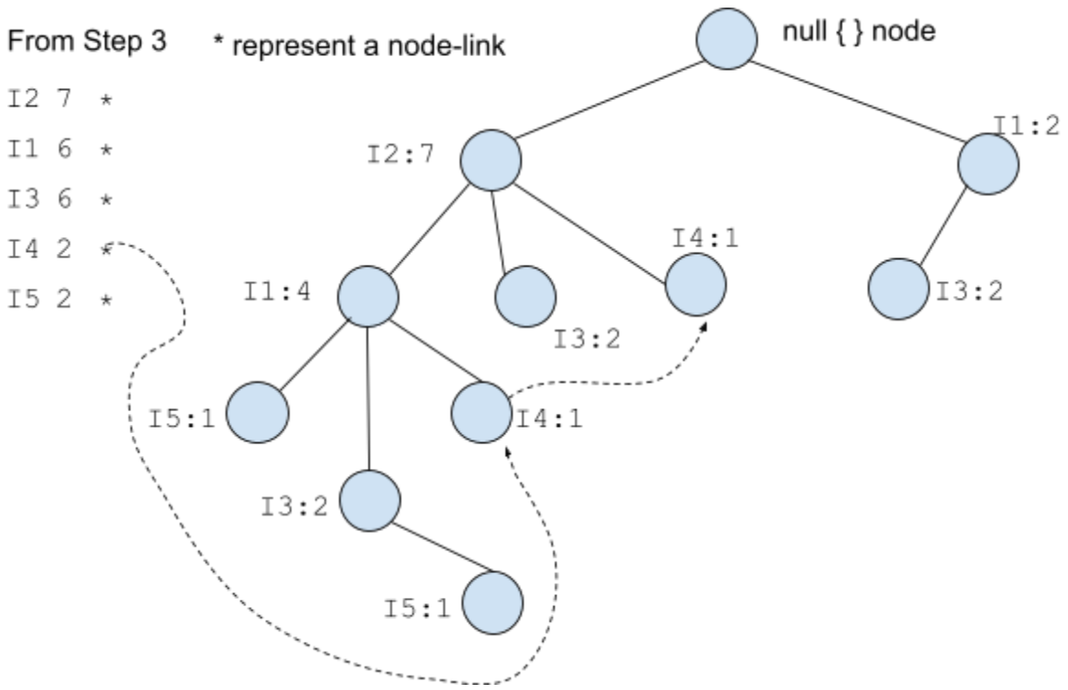
How to generate Frequent Patterns?

Generate all combinations with suffix `(I5)` using the prefix path (the conditional FP-Tree).

`{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5: 2}`

Step 6.2: Construct Conditional FP-Tree with the conditional node `I4`

Follow the node-link path - `I4` as below.



How to generate a conditional pattern base?

Consider `I4` as suffix, generate the Conditional Pattern Base and Conditional FP-Tree

```
I4:1 linked to I2 -> I1
I4:1 linked to I2
```

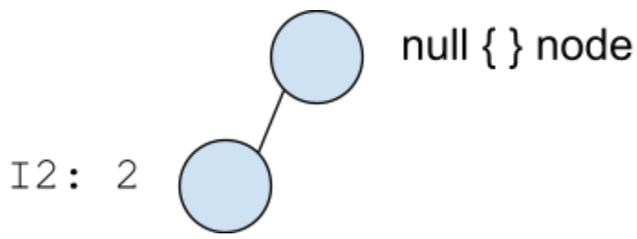
Item	Conditional Pattern Base	Conditional FP-Tree	Frequent Patterns generated

I4	{
	{I2, I1: 1},
	{I2: 1}
	}
		<I2: 2>	{I2, I4: 2}

How to generate a conditional FP-Tree? Prefix Paths.

Use the Conditional Pattern Base and generate Conditional FP-Tree as below.

Since the support count is 2, `I1` is ignored.



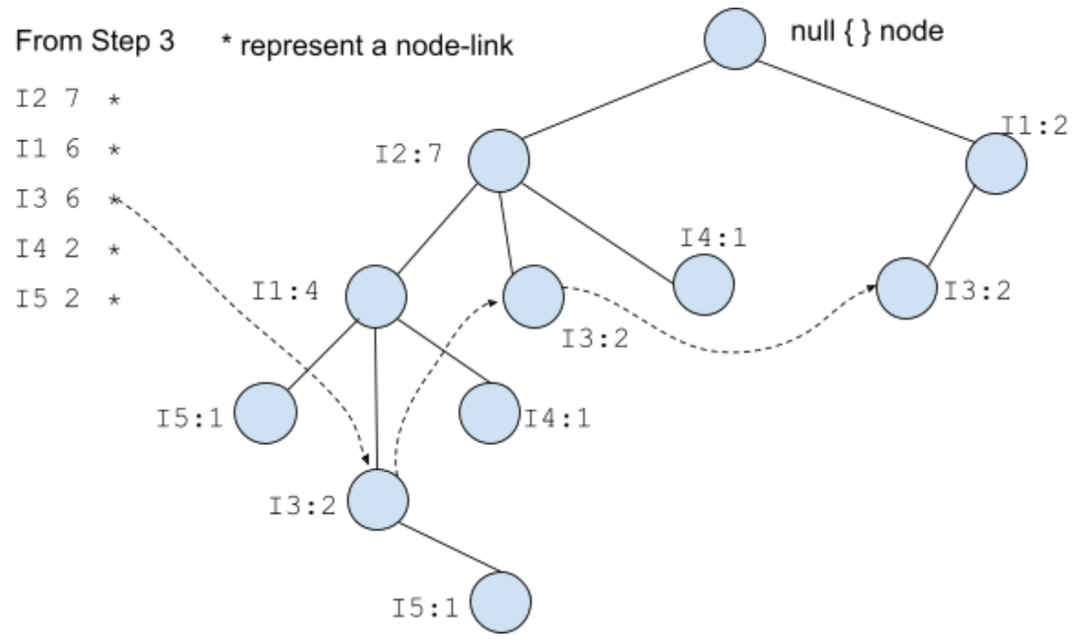
How to generate Frequent Patterns?

Generate all combinations with suffix `(I4)` using the prefix path (the conditional FP-Tree).

`{I2, I4: 2}`

Step 6.3: Construct Conditional FP-Tree with the conditional node `I3`

Follow the node-link path - `I3` as below.



How to generate a conditional pattern base?

Consider `I3` as suffix, generate the Conditional Pattern Base and Conditional FP-Tree

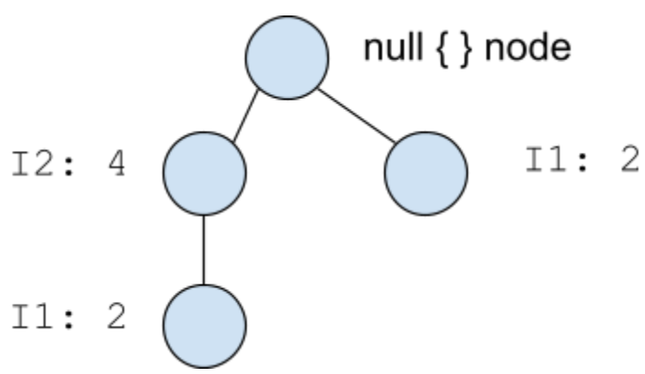
```
I3:2 linked to I2 -> I1
I3:2 linked to I2
I3:2 linked to I1
```

Item	Conditional Pattern Base	Conditional FP-Tree	Frequent Patterns generated

I3	{
	{I2, I1: 2},
	{I2: 2},
	{I1: 2}
	}
		<I2: 4, I1: 2>,
		<I1: 2>
			{I2, I3: 4},
			{I1, I3: 2},
			{I2, I1, I3: 2}

How to generate a conditional FP-Tree? Prefix Paths.

Use the Conditional Pattern Base and generate Conditional FP-Tree as below.



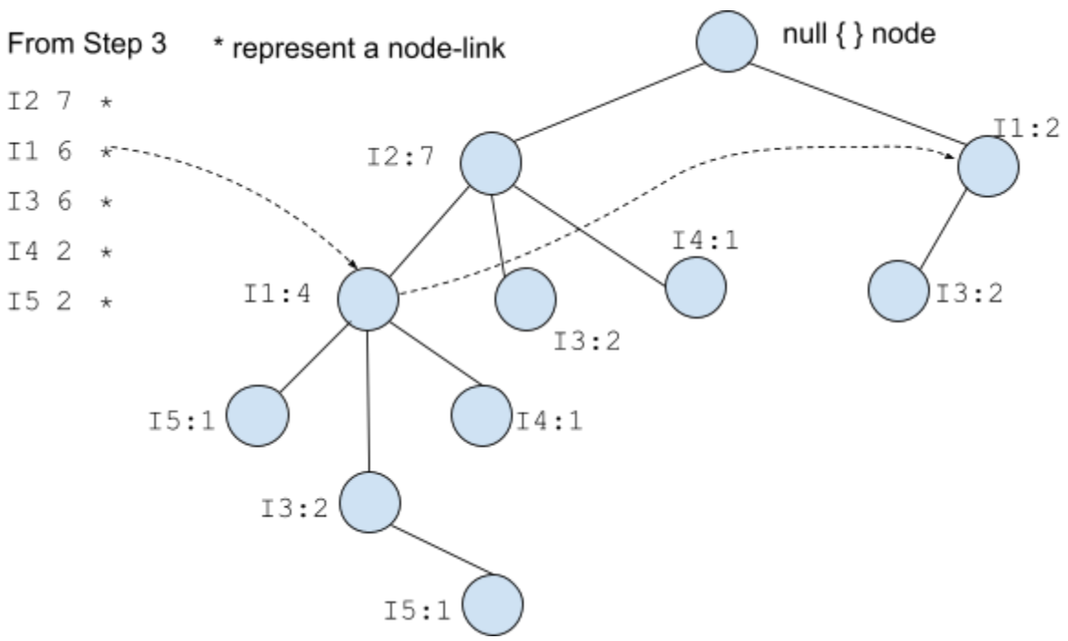
How to generate Frequent Patterns?

Generate all combinations with suffix `(I3)` using the prefix path (the conditional FP-Tree).

`{I2, I3: 4}, {I1, I3: 2},{I2, I1, I3: 2}`

Step 6.3: Construct Conditional FP-Tree with the conditional node `I1`

Follow the node-link path - `I1` as below.



How to generate a conditional pattern base?

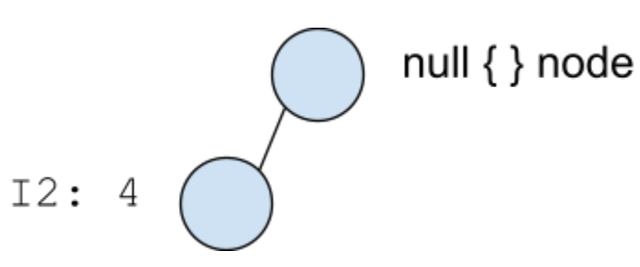
Consider `I1` as suffix, generate the Conditional Pattern Base and Conditional FP-Tree

`I1:4 linked to I2`
`I1:2 linked to null`

Item	Conditional Pattern Base	Conditional FP-Tree	Frequent Patterns generated
`I1`	`{ {I2: 4} }`	`<I2: 4>`	`{I2, I1: 4}`

How to generate a conditional FP-Tree? Prefix Paths.

Use the Conditional Pattern Base and generate Conditional FP-Tree as below.



How to generate Frequent Patterns?

Generate all combinations with suffix `(I1)` using the prefix path (the conditional FP-Tree).

`{I2, I1: 4}`

Summary:

Item	Conditional Pattern Base	Conditional FP-Tree	Frequent Patterns generated
I5	{ {I2, I1: 1}, {I2, I1, I3: 1} }	<I2: 2, I1: 2>	{I2, I5: 2}, {I1, I5: 2}, {I2, I1, I5:2}

I4	{
	{I2, I1: 1},
	{I2: 1}
	}
		<I2: 2>	{I2, I4: 2}

I3	{
	{I2, I1: 2},
	{I2: 2},
	{I1: 2}
	}
		<I2: 4, I1: 2>,
		<I1: 2>
			{I2, I3: 4},
			{I1, I3: 2},
			{I2, I1, I3: 2}

I1	{
	{I2: 4}
	}
		<I2: 4>	{I2, I1: 4}

Questions:

A	**Generate Rules using Frequent Patterns generated.** **It is simple and you can try.**
