FP Tree (Frequent Pattern tree)

Using the database below, we will calculate the support for all single items. After calculating the support for each item in the database, we sort these items by support in decreasing order.

1 ACTW
2 CDW
3 ACTW
4 ACDW
5 ACDTW
6 CDT

C(6), W(5), A(4), D(4), T(4)
X (support) denotes the support for an itemset X
Ordering: CWADT
Putting it in this order is to maximize the common prefixes.
All transactions should take on the same ordering of their itemsets
For example: transaction 5 = ACDTW = CWADT (Do this for all transactions in the database.)

Reorder all transactions results in the following:

1 CWAT
2 CWD
3 CWAT
4 CWAD
5 CWADT
6 CDT

Assumption: minsup = 3/6

An FP –tree is a projection based approach

The FP-tree is constructed by using the new ordering of each transaction. We use the common prefixes to construct the tree and add branches wherever needed. Our result is the tree below:

```
NULL
 |
 C(6)---D(1)----T(1)
 |
W(5)--D(1)
 |
A(4)---D(2)---T(1)
 |
T(2)
```

The support of each node in the above tree indicates how many times that prefix (from root) is traversed in the corresponding database.

Now, we have to consider all items from T to C, in the reverse order of support, from smallest support to largest support. We have to generate all subsets that end in the particular item that we are processing before moving to the next item. All occurrences of each item are kept in a linked list. Example: a linked list for T that keeps track of T's occurrence: T: T(2) — T(1)—T(1) . The same thing for D's, D: D(2) — D(1) — D(1), and other items.

<u>Conditional FP-tree for T</u>

We are going to extract all paths that end with T. First, we count for T's occurrence. To do this, we can trace the linked list and add up all of its support. T has a support of 4 because 4 is the sum of T's linked list (T: T(2)—T(1)—T(1)). T is frequent because its support > 3.

We start with drawing tree whose end-nodes are Ts, keeping only the support of Ts.

```
    NULL
      |
 C(0)---D(0)---T(1)
      |
 W(0)
      |
  A(0)---D(0)---T(1)
      |
 T(2)
```

Then, we take out T one by one, and as we do so, we push its support to every node up the chain to the root that was a part of the same transaction in which T was.

```
    NULL                          NULL                     NULL
      |                             |                        |
 C(2)---D(0)---T(1)           C(3)---D(0)---T(1)          C(4)---D(1)
      |                    ➔        |                  ➔     |
 W(2)                         W(3)                        W(3)
      |                             |                        |
  A(2)---D(0)---T(1)           A(3)---D(1)              A(3)---D(1)
```

For example: C is part of 4 transactions in which T was. W is a part of 3 transactions in which was and so on.

```
    NULL
      |
 C(4)---D(1)
      |
 W(3)
      |
 A(3)---D(1)
```

With T being the last item, support(D) = 2.
Therefore, D is infrequent (less than minsup = 3) and will be removed from the FP-tree
We are now going to recursively enumerate all the subset that

ends up with item T, so we need to apply the same principle of FP-tree. We consider each item that appears in the tree and create new FP-trees for C, W, A, and D (note we are now within the context of the new FP-Tree for T).

The below tree is obtained by removing D, which is infrequent. Since there is only one path, now we can enumerate all subsets of that path and add T at the end of each subset because T is the particular item that we are enumerating subsets for. The support of each subset is then set to be the supported of the last item before T. For example, CWT has a support of three because W (the last item before T in CWT) only has a support of three (from the subtree below, i.e. W(3)), even though C and T have a support of four.

```
NULL
  |
 C(4)
  |
 W(3)
  |
 A(3)
```

Now, we construct all subsets of CWA and the note the support

CT(4)..CWT(3)..CWAT(3)
WT(3)..CAT(3)..T(4)
AT(3)..WAT(3)

8 possible subsets for T

Now, we can remove T from the entire original tree because we have enumerated all output subsets that contain T in the database.

This process is then repeated for D, A, W, and C, in that particular order.

<u>Construct FP-tree for D</u>

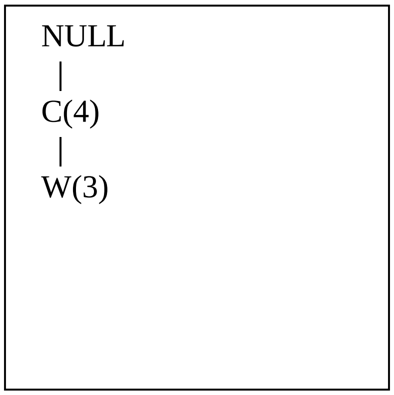
```
NULL
  |
 C(0)---D(1)
  |
W(0)--D(1)
  |
 A(0)---D(2)
```

D's support is then pushed up onto the FP-tree
The support of D, which is 4, is added to everything in its path above it

```
NULL
  |
 C(4)
  |
 W(3)
  |
 A(2)
```

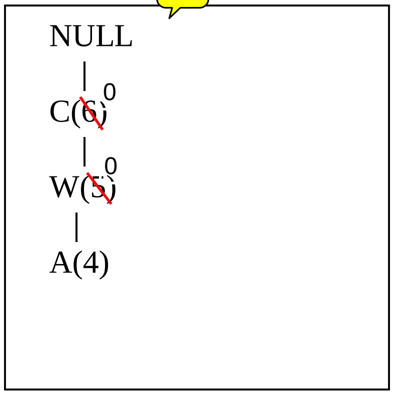
A(2) is not frequent because it is less than the minsup = 3, so we don't need to enumerate any subsets that end with an A. So we can take A out from this subtree.

.

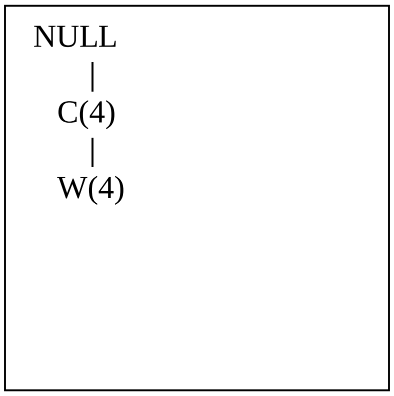


Now, that everything is frequent in this subtree, we construct all possible subsets of items that ends with D:
CD(4)..WD(3)
CWD(3)..D(4)

Construct FP-tree for A



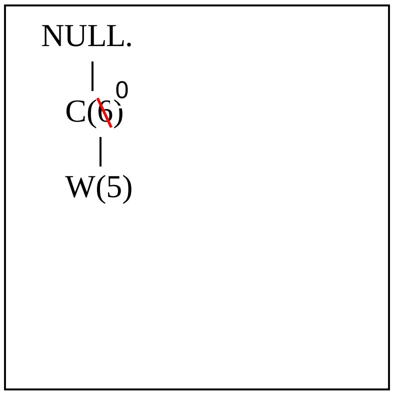
Push A onto FP-tree



W is frequent, so it will be part of A possible subsets

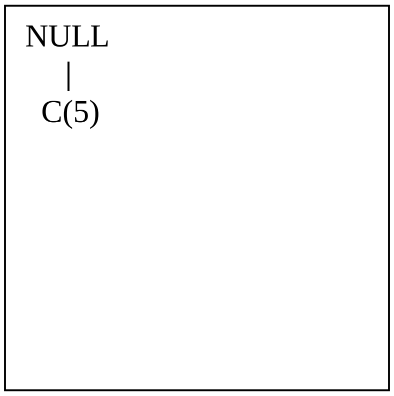
CA(4)..CWA(4)
WA(4)..A(4)

Construct FP-tree for W



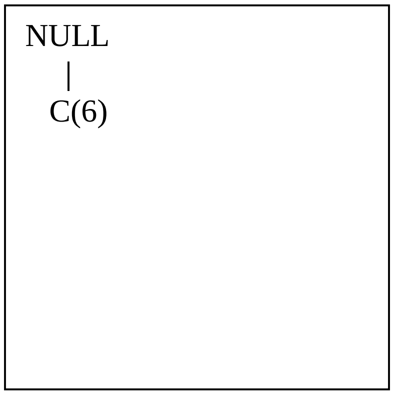
Push W up onto FP-tree

.



C is frequent, so it is part of all possible subsets of W
CW(5)..W(5)

Construct the FP-tree for C



C is the root, so we cannot push it up the tree
Therefore, we can list all subset of C

C(6)

In our conclusion, we can see that we have 19 possible output subsets

Complexity

Worst Case: We could have as many as 2^n leaves because every transaction in our DB could be different (no overlapping sub-items).

Best Case: We could have as few as 1 path if every transaction in our DB is the same.

Sequence Mining

With Rule Mining / Set Mining, we try to find relationship between items.

Sequence mining adds temporal or positional dimension, i.e., a "happens after" relation.

Example for Sequence Mining:

1) Biosequences

Domain space $\Sigma = \{A, T, C, G\}$

Example of possible DNA sequences:
 a) ACCTGCTA …
 b) CCATCGTA….

Goal : to mine frequent sequences (gene finding)

2) Hypertext documents (Web domain).

By keeping track of the word position and number of its occurrences we can extract most frequent occurring words and sequences from the documents.

Suppose these documents fall into some categories such as music, art, etc. If two documents are in the same category, we are interested to find if there are some patterns that keep repeating.

3) Grocery store example

Cust ID	Time (day of the year)	Items
1	10	ACD
1	11	D
1	13	CD

The transactions above for customer ID 1 can be rewritten as follow:

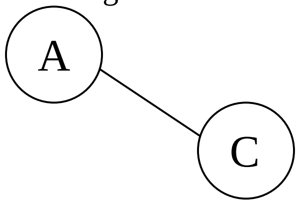
ACD → D → CD

(The arrows indicate the "happen after" relations, i.e.: D happens after ACD)

4. Web log mining
   Keep track of the IP address of the users and the web pages they visited.

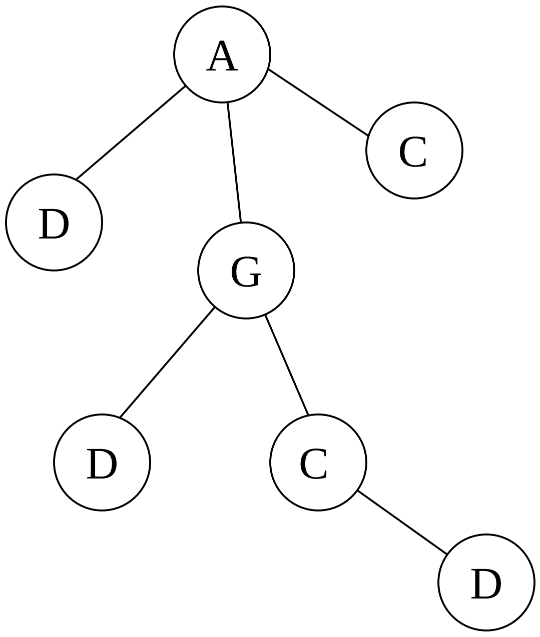
User	Web log
1	A → C → A → G
2	A → G → C → D
3	E → F → A

Web log can also be represented by tree. For example, for user 2, his/her web log can be represented as follow. The edge in the tree represents the 'happens after' relationship, i.e.



This means, C happens sometimes after A.

So web log sequence for user 2 can be represented by the following tree, whereas is one forward sequence supported by the user A → G → C → D



For Set Mining, the search space is all combinations of lengths 1 to n, where n is the number of items. The sets ABC, BCA, and CBA were all equivalent.

For Sequence Mining, the order matters. The search space is all permutations of lengths 1 to n.

Finding Frequent Sequences
<u>Example Database</u>
The example DB below records the activity of 4 customers (CID) and the time (Time) in which they purchased items (Items).

CID	Time	Items
1	10	CD
	15	ABC
	20	ABF
	25	A
2	15	ABF
3	10	ABF
4	10	D
	20	BF
	25	A

The transactions can be rewritten as follows:

CD -> ABC ->ABF

ABF
ABF

D-> BF -> A

*CID = Customer ID

To find the frequent sets, first count the single items.

Minsup = 2 /4

C_1 = { A_4, B_4, C_1, D_2, F_4 } *Items are counted once per customer
-Discard C

F_1 = { A_4, B_4, D_2, F_4 }

How are candidates of size 2 generated?
First find sets of size 2 (*set extension)*

C_2 = { AB_3, AD_0, AF_3, BD_0, BF_4, DF_0}
Now consider all size 2 sequences starting with A (*sequence extension*)
A->A_1, A->B_1, A->D_0, A->F_1

Now sequences starting with B:
B->A_2, B->B_1, B->D_0, B->F_1

Now D:
D->A_2, D->B_2, D->D_0, D->F_2

Finally F:
F->A_2, F->B_0, F->D_0, F->F_0

F_2 = { $\boxed{AB_3, AF_3}$ $\boxed{BF_4, B\text{->}A_2}$ $\boxed{D\text{-}A_2, D\text{->}B_2, D\text{->}F_2}$ $\boxed{F\text{->}A_2}$ }

Some General Rules for Generating Candidates

1) Only combine items within a single class (denoted by $\boxed{\text{boxes}}$). In these classes, all members contain the same prefix. Classes contain both sets (such as BF_4), and sequences (B->A_2). Here, sets are similar to the sets generated with Set Mining. However, in Sequence Mining the order matters, so we are interested in X->Y as well as Y->X.

   For example, what can we consider for C_3 by extending $\boxed{BF_4, B\text{->}A_2}$?

B->AF	No
B->A->F	No
B->A->A	√ **Yes!**

   For B->AF, from F_2 there is no evidence that B->AF is frequent.
   Similarly for B->A->F there is no evidence that B->F is frequent.
   B->A->A is valid because we know B->A is frequent. It is possible that within B->A, there are intermediate items. In this class however, the only possibility is another A.

2) How are sets and sequences extended?
   Generate candidates, prune infrequent subsequences

   Combine:

1	Sets with Sets
2	Sets with Sequences
3	Sequences with Sequences
4	But not Sequences with Sets

Using these rules, let's generate C_3 from
F_2 ={$\boxed{AB_3, AF_3}$ $\boxed{BF_4, B\text{->}A_2}$ $\boxed{D\text{-}A_2, D\text{->}B_2, D\text{->}F_2}$ $\boxed{F\text{->}A_2}$}

1) Start with class $\boxed{AB_3, AF_3}$
   There is only one possible candidate, ABF (rule 1)

2) Now class $\boxed{BF_4, B\text{->}A_2}$
   BF->A (rule 2)
   B->A->A (rule 3)
   ~~B->BF~~ (rule 4) We do not generate sequences like this.

3) Now class $\boxed{D\text{-}A_2,\ D\text{->}B_2,\ D\text{->}F_2}$

D->A->A	(rule 3)	D->AB	(rule 3)
D->AF	(rule 3)	D->A->F	(rule 3)
D->B->B	(rule 3)	D->BF	(rule 3)
D->B->F	(rule 3)	D->B->A	(rule 3)
D->F->F	(rule 3)	D->F->A	(rule 3)
D->F->B	(rule 3)		

4) Finally class $\boxed{F\text{->}A_2}$
F->A->A (rule 3)

5) Infrequent sequences such as D->B->F can be eliminated since there is no evidence from [f2] that its subsequence B->F is frequent.

$F_3 = \{\boxed{ABF_3}, \boxed{BF\text{-}A_2}, \boxed{D\text{->}F\text{->}A_2}, \boxed{D\text{->}BF_2,\ D\text{->}B\text{->}A_2}\}$

$C_4 = \{D\text{->}BF\text{->}A_2\}$
At this point, no possible extension can be generated. Since D->BF->A >= minsup, so we include it in F_4.