Übung 4

Frequent Itemset Mining und FP-Tree

Frequent Itemset Mining

Motivation:

- Es existiert eine Datenbank mit vielen Einträgen
- Man möchte wissen, welche Einträge oft zusammen vorkommen

Frequent Itemset Mining

Mathematische Definitionen:

- Sei *Items* $I = \{i_1, ..., i{-}m\}$ eine Menge von Literalen
- Sei *Itemset* $X : X \subseteq I$ eine Menge von Items
- $T = (t_1, ..., t_n)$ ist ein Tupel von Transaktionen, wobei gilt $\forall k, 1 \leq k \leq n : t_k \subseteq I$. T ist die Transaktionsdatenbank
- $cover(X) = \{t | t \in T, X \subseteq t\}$ ist das *Cover* eines Itemsets X
- Sei $support(X) = |cover(X)|$ der Anteil der Transaktionen in T, die X enthalten.
- Ein frequent Itemset (oder frequent pattern) ist ein Itemset X in T, das häufiger vorkommt, als ein Grenzwert s: $support(X) \geq s$, wobei $(0 \leq s \leq |T|)$

Frequent Itemset Mining: Beispiel Warenkorb

Es gibt einen Supermarkt bei dem Kunden verschiedene Waren kaufen können. Alle Einkäufe werden beim Bezahlen gespeichert.

- Waren im Supermarkt sind die Items
- Ein Einkauf ist ein Itemset
- Transaktionsdatenbank ist die Menge aller Einkäufe

Frequent Itemset Mining: Beispiel Warenkorb

Einkäufe:

- {Butter, Brot, Milch, Zucker}
- {Brot, Mehl, Milch, Wurst}
- {Bier, Chips, Käse}
- {Bier, Butter, Chips, Mehl}

Also:

- $cover(Milch) = \{\{Butter, Brot, Milch, Zucker\}$ $, \{Brot, Mehl, MilchWurst\}\}$
- $support(Milch) = 2$
- Bei Grenzwert 2 sind frequent Itemsets {Brot,Milch} und {Bier, Chips}

FP-Tree

- Der Frequent Pattern Tree (FP-Tree) ist eine Datenstruktur um frequent Itemsets zu speichern und zu erkennen
- Besteht aus Baum und Tabelle Header table
- Hat zwei wichtige Algorithmen: Konstruktion und Finden von frequent Pattern

FP-Tree: Konstruktion

```
FP-Tree-Construction(Datenbank DB, Integer threshold):
List F = emptyList()
1. Scan Database
1.1     Collect Set of frequent Items in F
        their  supports
1.2     Sort F in support descending order as L,
```

FP-Tree: Konstruktion

```
2.   Create the root of FP-tree, T, and label it
     as "null".
     For each transaction trans in DB do:
2.1   Select and Sort the Items in Trans
      according to the Order of L
2.2      p = first Element in Trans
2.3      P = remaining List
2.4      insert_Tree([p|P],T)
```

FP-Tree: Konstruktion

```
insertTree([p|P],T):
1.    If T hasChild N and N.name==p.item-name
N.Count=N.Count+1
2.    Else:
2.1       Node N = new Node
2.2       N.count = 1
2.3       N.parent = T
2.4       HeaderTable.add(N)
3.    If P != EmptyList
          insertTree(P[0]|P[1:P.length])
```

FP-Tree: Konstruktion: Beispiel

Transaktionsdatenbank:

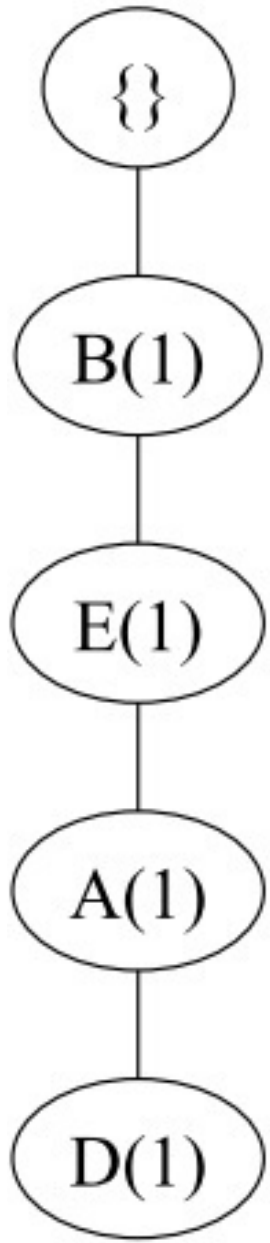
	i(t)
1	ABDE
2	BCE
3	ABDE
4	ABCE
5	ABCDE
6	BCD

Daraus ergibt sich als Sortierreihenfolge: (B,E,A,C,D)

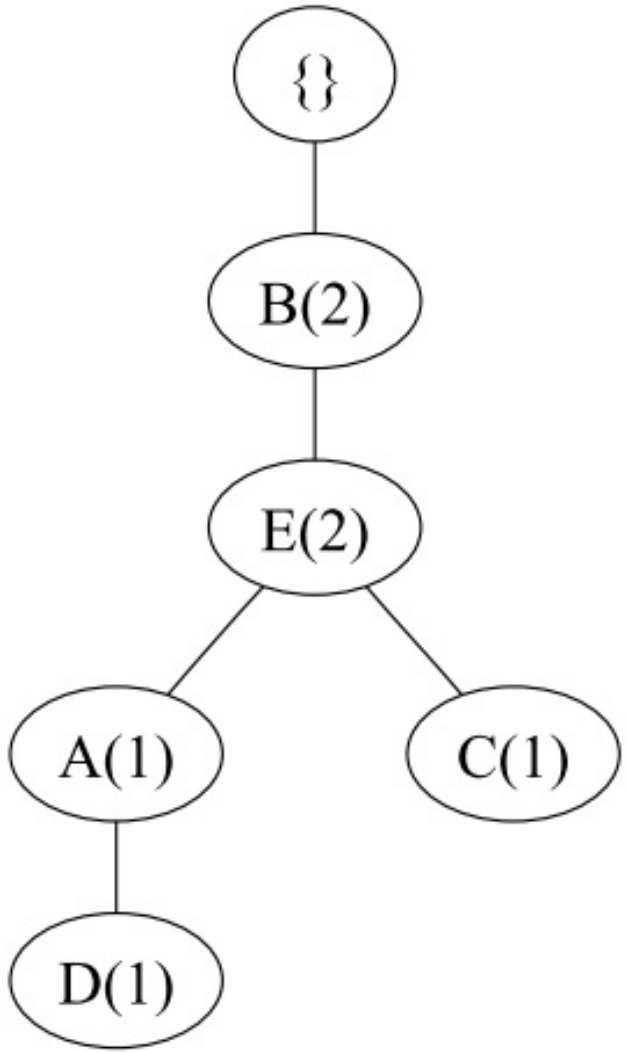
Transaktionsdatenbank nach Sortierung

	i(t)
1	BEAD
2	BEC
3	BEAD
4	BEAC
5	BEACD
6	BCD

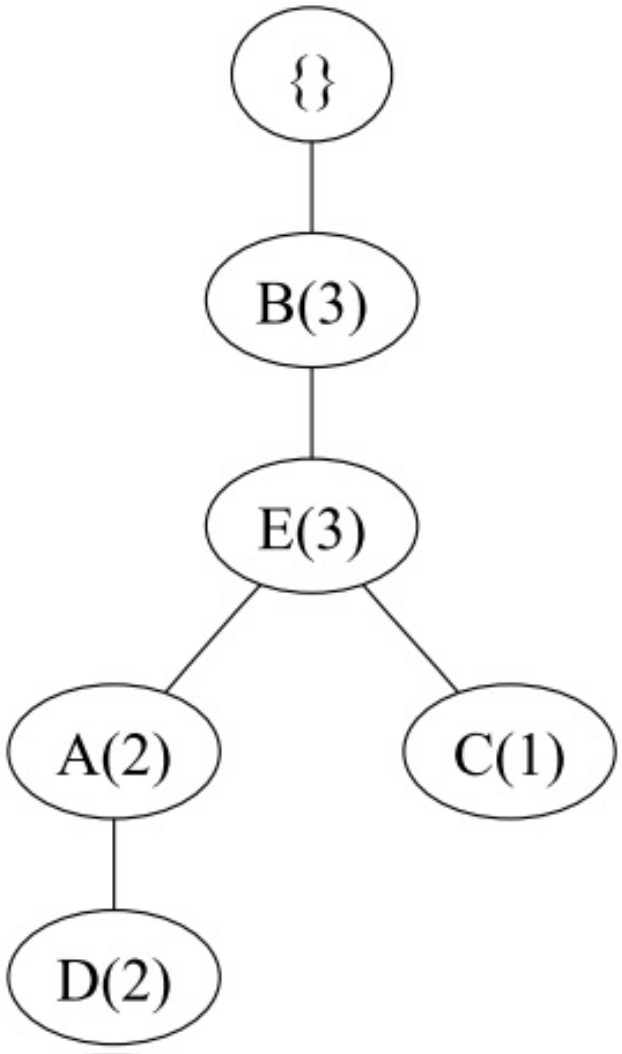
FP-Tree nach Transaktion 1



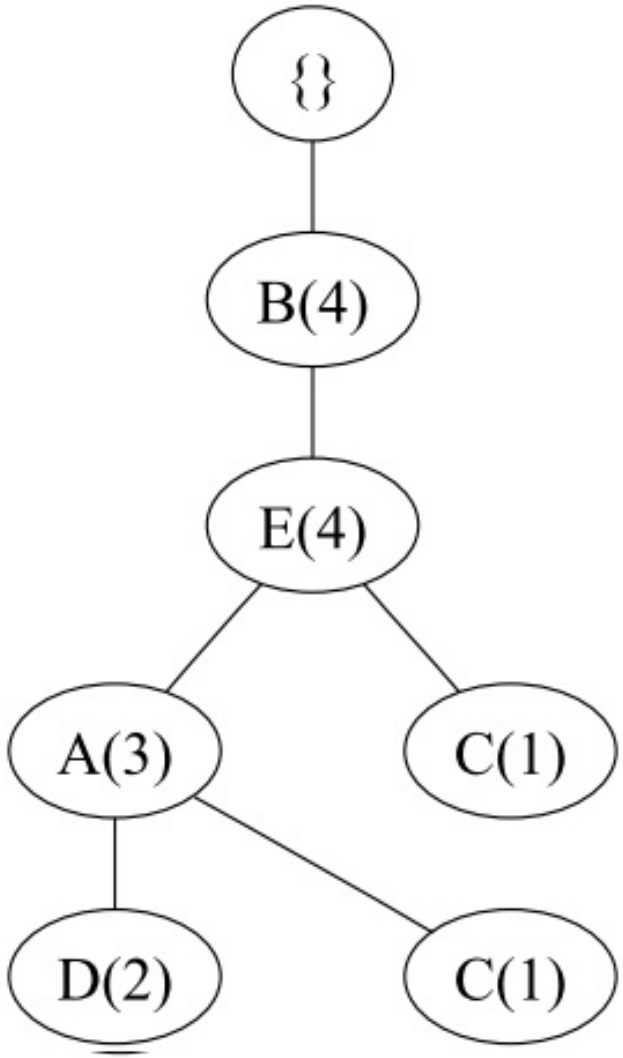
FP-Tree nach Transaktion 2



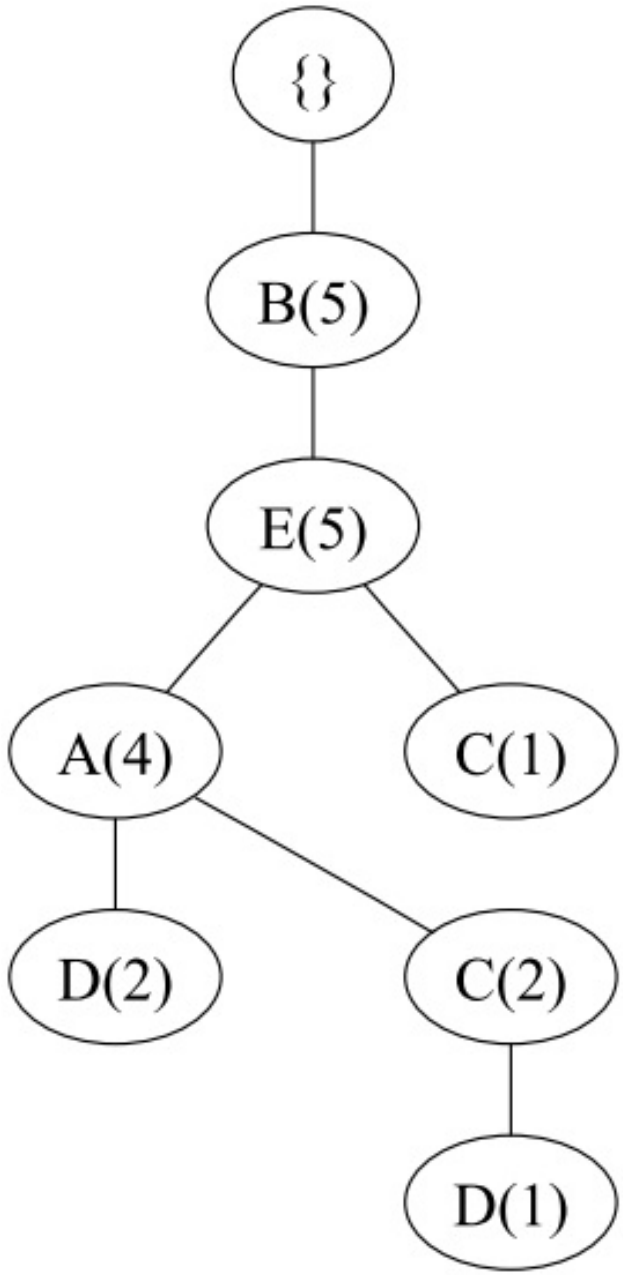
FP-Tree nach Transaktion 3



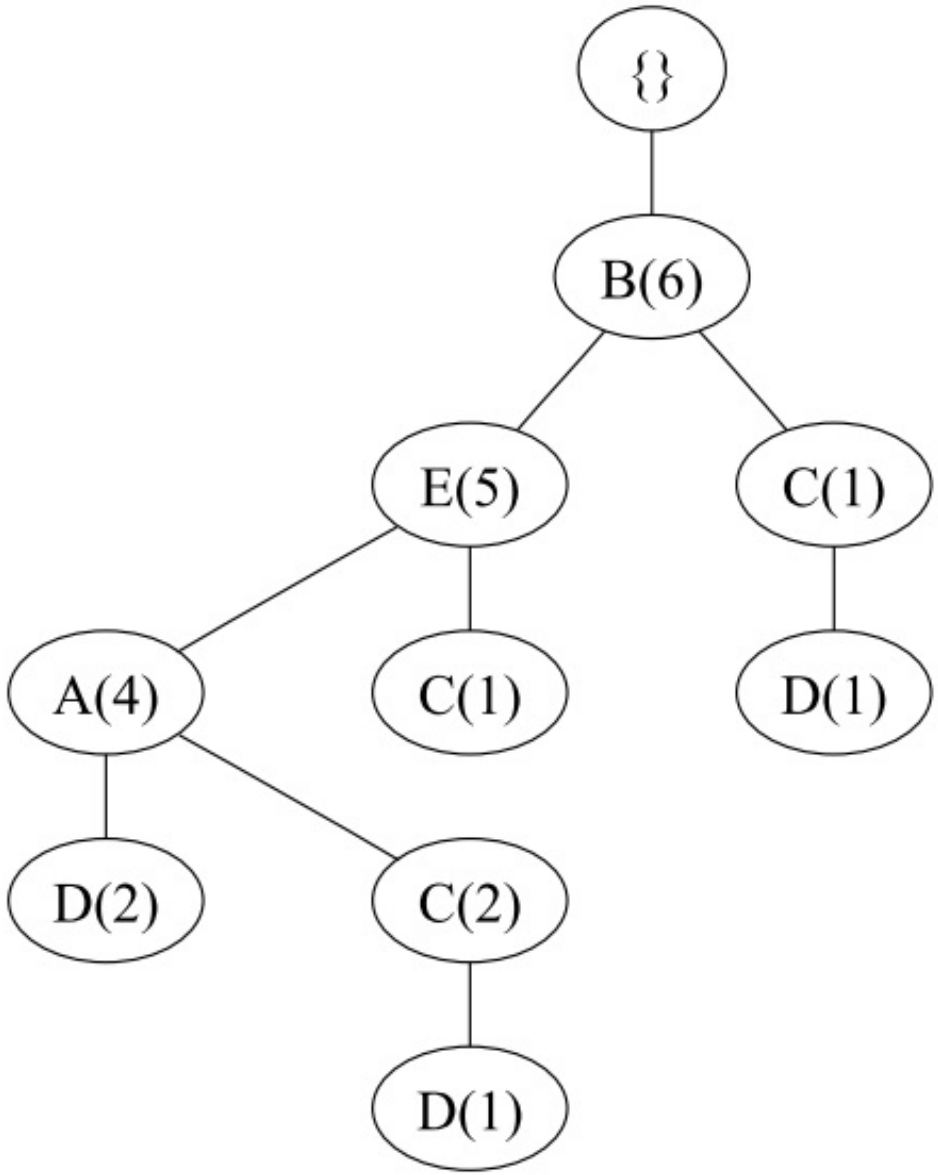
FP-Tree nach Transaktion 4



FP-Tree nach Transaktion 5



FP-Tree nach Transaktion 6



Wichtige Definitionen zum FP-Growth Algorithmus

transformed prefix path

Ein Teilpfad eines FP-Trees, bei dem der Zähler in jedem Knoten, auf den Wert des Zählers des letzten Knotens in dem Pfad gesetzt wird.

conditional Pattern base

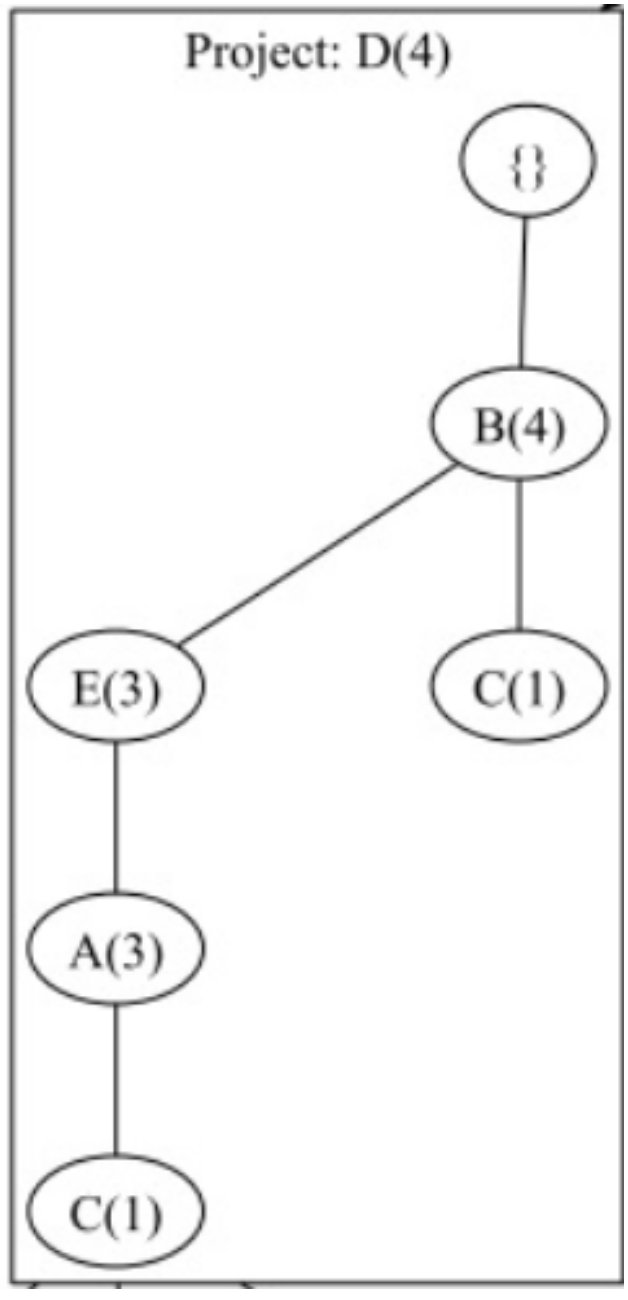
Menge von transformationen, die sich aus einer Menge von transformed prefix paths ableiten lässt.

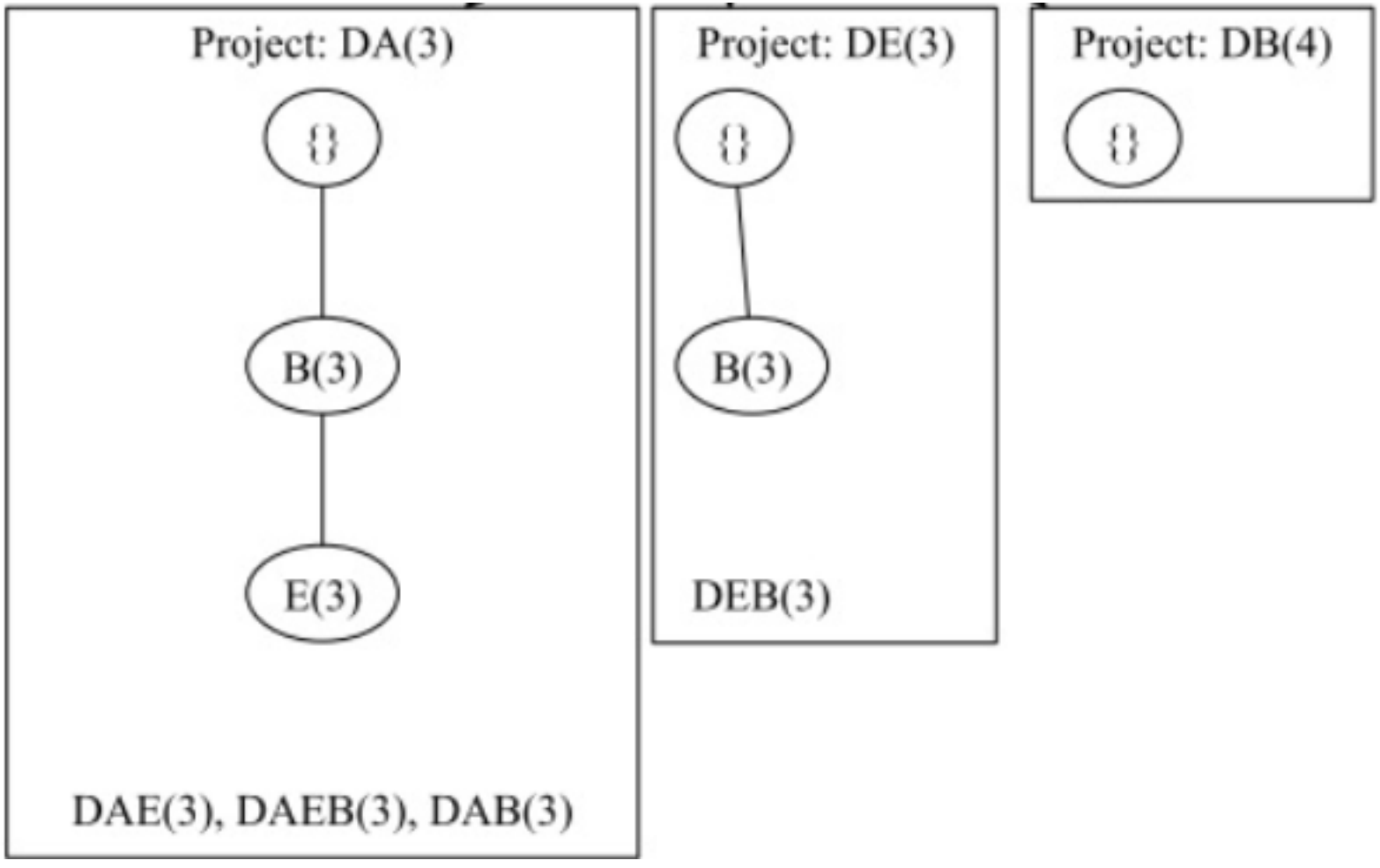
FP-Growth

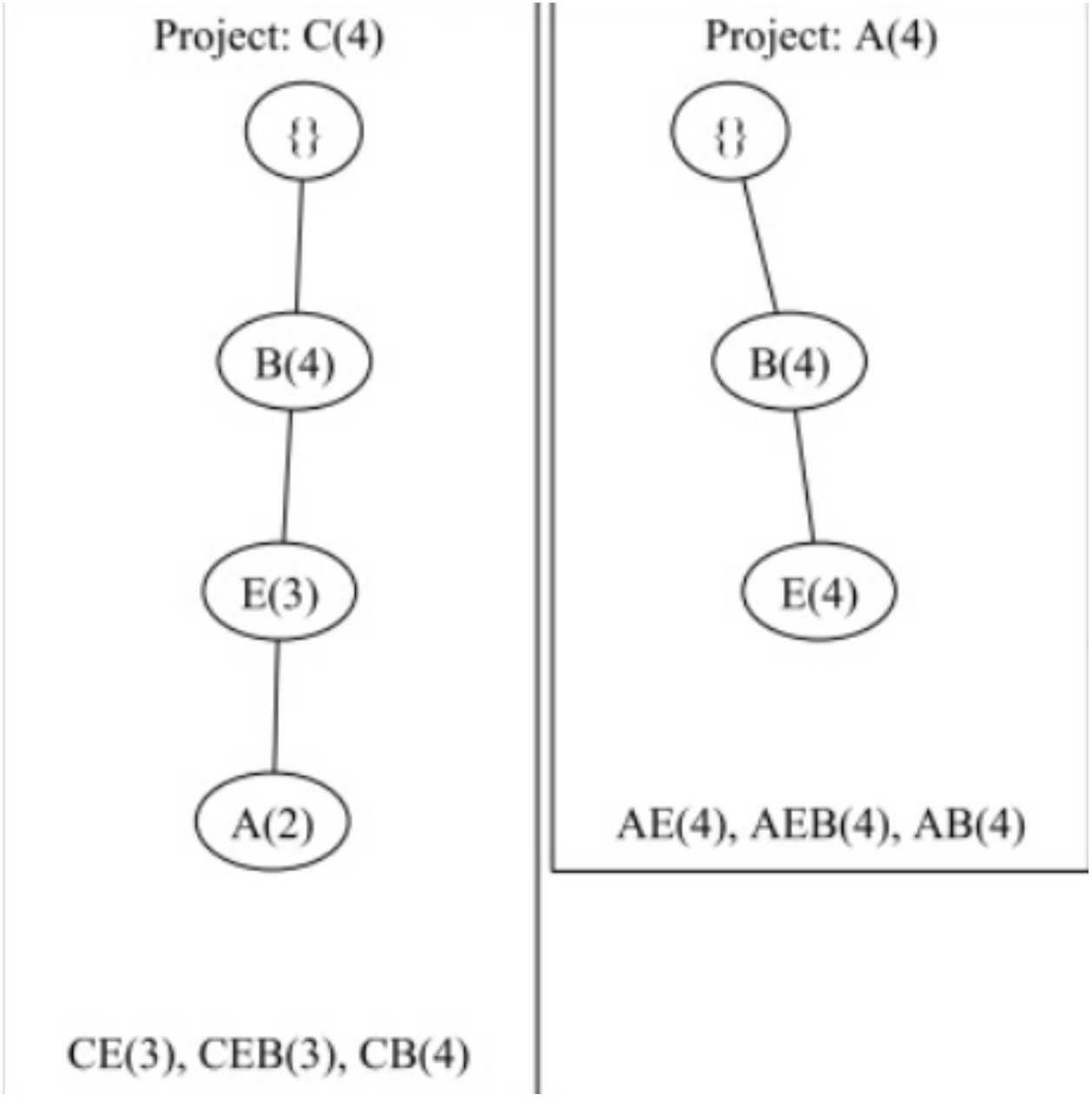
```
call FP-Growth(FP-tree,null)

FP-Growth(Tree, alpha):
1.  If(Tree has only one Path P):
1.1     for each combination beta of the nodes P:
1.1.1     generate Pattern beta U alpha
1.1.2     (beta U alpha).support =
          min(alpha.support, beta.support)
```

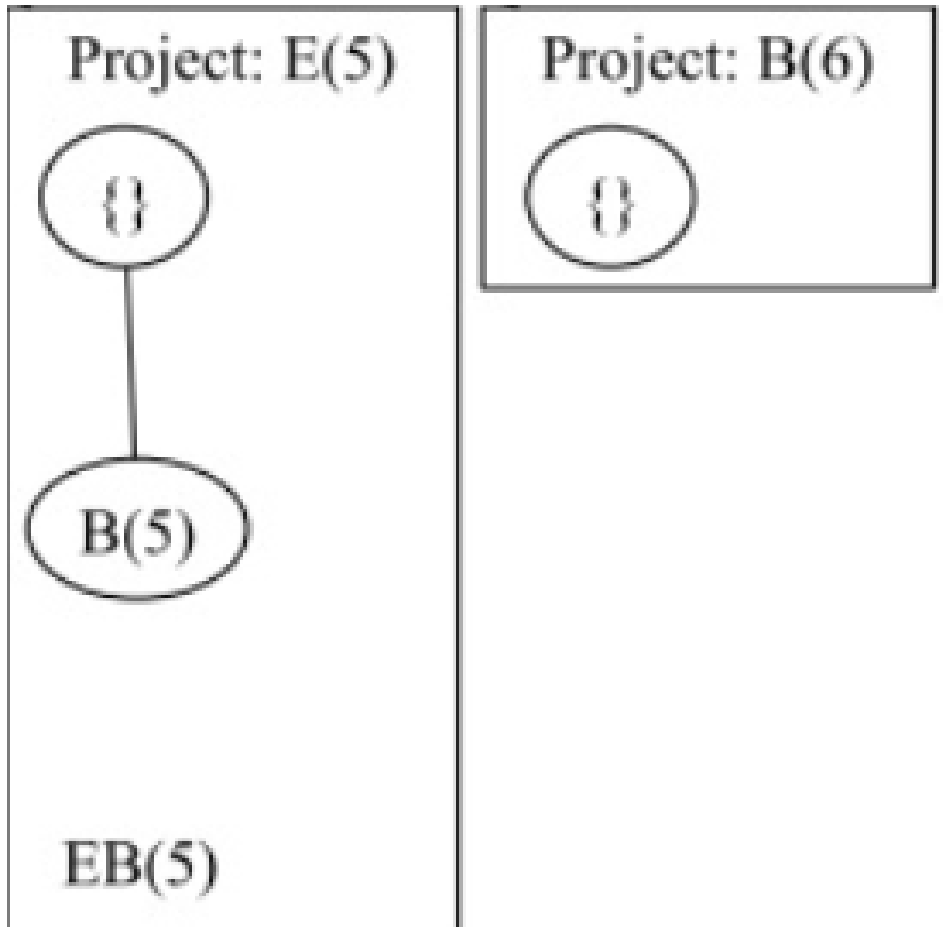
FP-Growth

```
2.  Else:
2.1   for each a in HeaderTable of Tree:
2.1.1     beta = a U alpha
2.1.2     beta.support = a.support
2.1.3     construct beta's conditional pattern base
2.1.4     construct beta's conditional FP-tree betaTree
2.1.5     If betaTree != empty
2.1.5.1     FP-Growth(betaTree, beta)
```

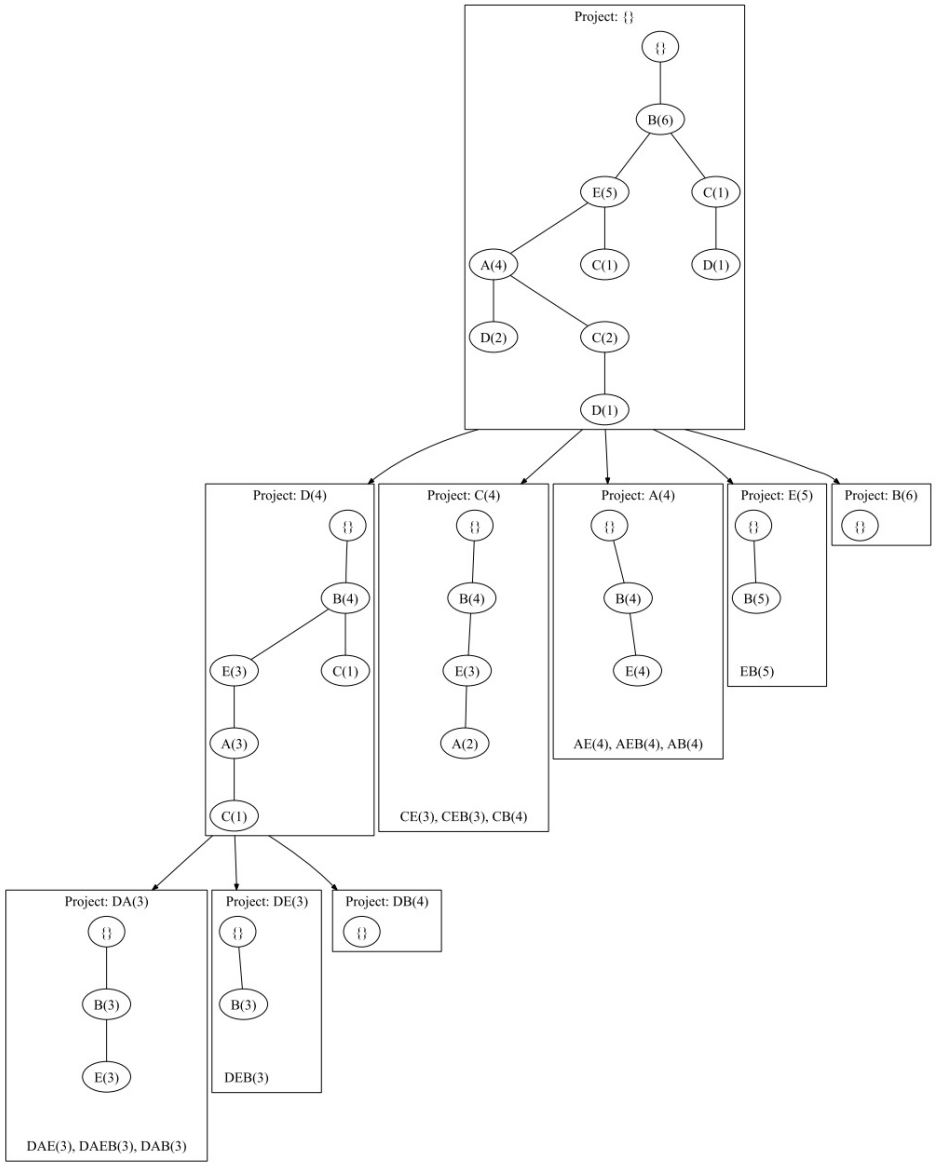

Project: D(4)
{}
B(4)
E(3)
C(1)
A(3)
C(1)


Project: DA(3)
{}
B(3)
E(3)
DAE(3), DAEB(3), DAB(3)
Project: DE(3)
{}
B(3)
DEB(3)
Project: DB(4)
{}


Project: C(4)
{}
B(4)
E(3)
A(2)
CE(3), CEB(3), CB(4)
Project: A(4)
{}
B(4)
E(4)
AE(4), AEB(4), AB(4)


Project: E(5)
{}
B(5)
EB(5)
Project: B(6)
{}

Kompletter Baum



Ergebnis von FP-Growth

Frequent Itemsets: { DAE, DAEB, DAB, DEB, CE, CEB, CB, AE, AEB, AB, EB }

Weitere Hilfen

Ausführliche Erklärung:
https://en.wikibooks.org/wiki/Data_Mining_Algorithms_In_R/Frequent_Pattern_Mining/The_FP-Growth_Algorithm
Simple Implementierung in PySpark:
Wird auf Homepage der Vorlesungsseite hochgeladen
Sehr gute Implementierung (in Scala):
https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/mllib/fpm/FPGrowth.scala