(ilişki kuralları =))

* What is Association Rule mining?

Given a set of transactions, find rules that will predict occurange of an item based on occurances of other items in a transaction.

TID	Items
1	Bread, Peanuts, Milk, Fruit, Jam
2	Bread, Jam, Soda, Chips, Milk, Fruit
3	Steak, Jam, Soda, Chips, Bread
4	Jam, Soda, Peanuts, Milk, Fruit
5	Jam, Soda, Chips, Milk, Bread
6	Fruit, Soda, Chips, Milk
7	Fruit, Soda, Peanuts, Milk
8	Fruit, Peanuts, Cheese, Yogurt

→ ex of ass. rules;

{Bread} ⇒ {Milk}

{Soda} ⇒ {Chips}

* {Bread} (X) and {Milk} (Y) are itemsets

* Two evaluation Metrics;

① Support: fraction of transactions that contain both X and Y

$$s = \frac{\sigma(\{Bread, Milk\})}{\#\text{ of transactions}} = \frac{3}{8}$$

* Sırası önemli değil Milk ⇒ Bread de olur.

② Confidence: Measures how often items in Y appear in transaction that contain X

$$c = \frac{\sigma(\{Bread, Milk\})}{\sigma(\{Bread\})} = \frac{3}{4}$$

* Association Rule Mining

* What are Frequent Itemset?

- An itemset whose support ≥ Minsup threshold

* Given a set of transactions, goal of association rule mining is to find all rules having; **support ≥ minsup** and **confidence ≥ minconf**

* Brute-Force Algorithm → computationally prohibitive =(

- List all possible association rules
- Compute support and confidence for each rule
- Prune (budamak) rules that fail the minsup and minconf thresholds.
- Pseudocode:

✡Apriori Algorithm ⇝ level-wise approach

- If an itemset is frequent, all its subsets must be frequent also
- $\forall X, Y: (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$ → *anti-monotone property

Given the following database and a min support of 3, generate all the frequent itemsets

D	A	B	C	D	E
1	1	1	0	1	1
2	0	1	1	0	1
3	1	1	0	1	1
4	1	1	1	0	1
5	1	1	1	1	1
6	0	1	1	1	0

(a) Binary database

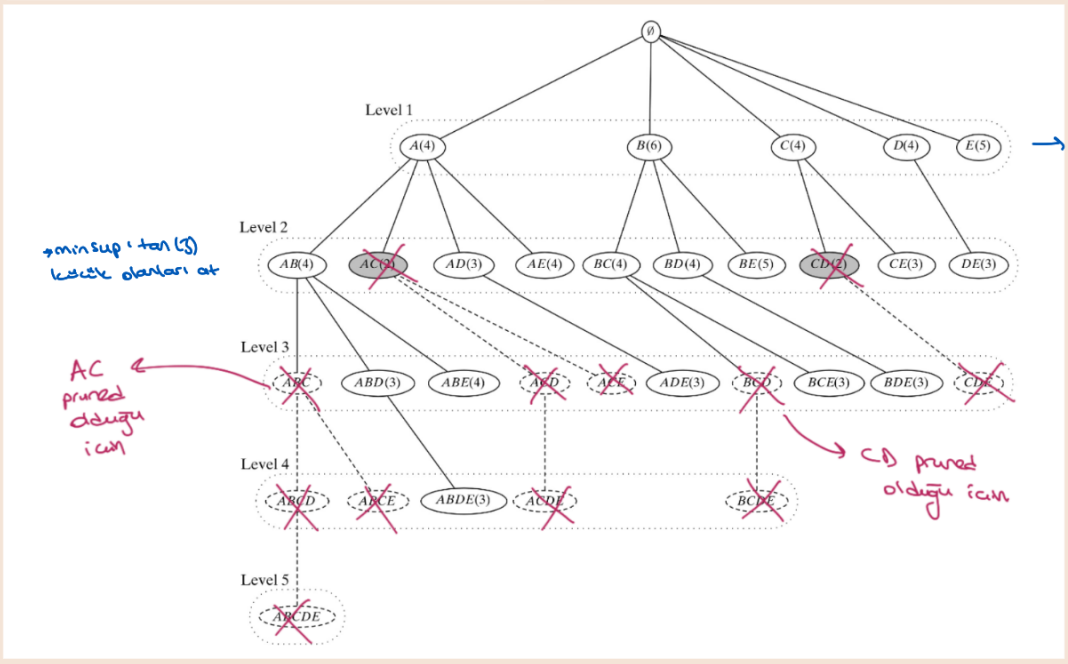
t	i(t)
1	ABDE
2	BCE
3	ABDE
4	ABCE
5	ABCDE
6	BCD

(b) Transaction database

x	A	B	C	D	E
t(x)	1	1	2	1	1
	3	2	4	3	2
	4	3	5	5	3
	5	4	6	6	4
		5			5
		6			

(c) Vertical database

→ 3 farklı gösterim



→ yan boruda kaçar adet olduğu yazıyor

*minsup'tan (3) küçük olanları at

AC pruned olduğu için

CD pruned olduğu için

- Pseudocode:

k = i(t)'nin uzunluğu ; (2) = 4 örneğin

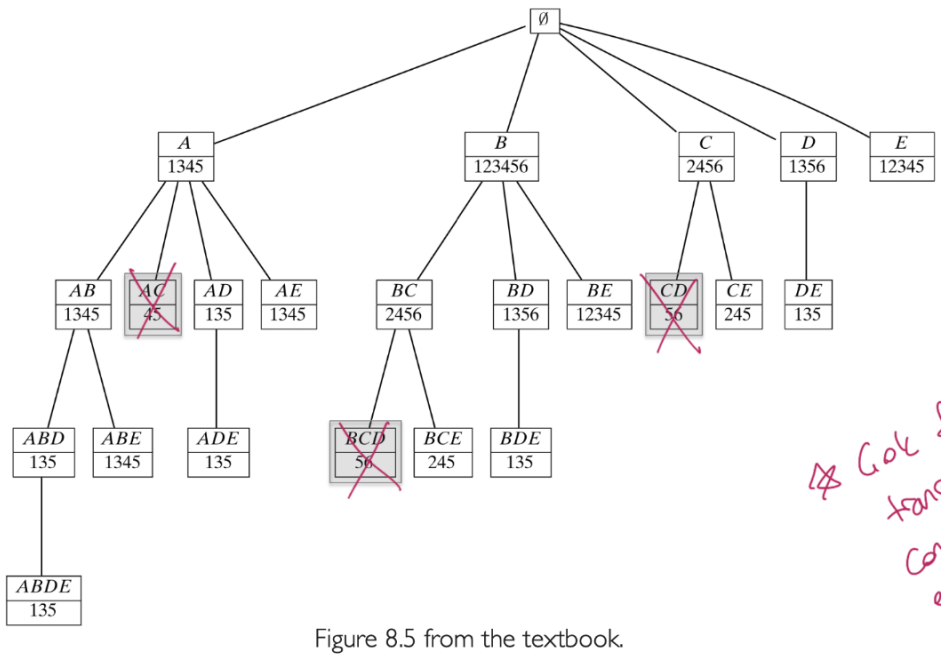
```
APRIORI(D, I, minsup):
F ← ∅                          // set of freq. itemsets
C' ← {∅}                       // initial prefix tree with single item
for each i ∈ I do
    Add i as child of ∅ in C' with sup(i) ← 0
k ← 1
while C^k ≠ ∅ do
    COMPUTESUPPORT(C^k, D)
    for each leaf X ∈ C^k do
        if sup(X) ≥ minsup then F ← F ∪ {(X, sup(X))}
        else remove X from C^k
    C^{k+1} ← EXTENDPREFIXTREE(C^k)
    k ← k+1
return F^k
```

```
COMPUTESUPPORT(X, D):
for each ⟨t, i(t)⟩ ∈ D do
    for each k-subset X ⊆ i(t) do
        if X ∈ C^k then sup(X) ← sup(X) + 1
```

```
EXTENDPREFIXTREE(C^k):
for each leaf X_a ∈ C^k do
    for each leaf X_b ∈ SIBLING(X_a) such that b > a do
        X_ab ← X_a ∪ X_b
        if X_j ∈ C^k for all X_j ⊂ X_ab, such that |X_j| = |X_ab| - 1 then
            add X_ab as child of X_a with sup(X_ab) ← 0
    if no extensions from X_a then
        remove X_a, and all ancestors (varsa) of X_a with no extensions, from C^k
return C^k
```

✱ Eclat Algorithm:

- Leverages the tidsets (subset of itemset) directly for support computation
- Itemset support is computed by intersecting tidsets of suitably chosen subsets.
- Given t(x) and t(y) (for frequent itemsets X and y) $\Rightarrow$ $t(XY) = t(X) \cap t(Y)$, $sup(XY) = |t(XY)|$



✱ Coût facile transaction vers computationally expensive

Figure 8.5 from the textbook.

- Pseudocode :

Algorithm 8.3: Algorithm ECLAT

```
// Initial Call: F ← ∅, P ← {⟨i, t(i)⟩ | i ∈ I, |t(i)| ≥ minsup}
ECLAT (P, minsup, F):
1 foreach ⟨X_a, t(X_a)⟩ ∈ P do
2     F ← F ∪ {(X_a, sup(X_a))}
3     P_a ← ∅
4     foreach ⟨X_b, t(X_b)⟩ ∈ P, with X_b > X_a do
5         X_ab = X_a ∪ X_b
6         t(X_ab) = t(X_a) ∩ t(X_b)
7         if sup(X_ab) ≥ minsup then
8             P_a ← P_a ∪ {⟨X_ab, t(X_ab)⟩}
9     if P_a ≠ ∅ then ECLAT (P_a, minsup, F)
```

☆ FP Growth Algorithm (Frequent Patterns Tree)

- Core of Apriori Algorithm
  - use frequent (k-1)-itemsets to generate candidate freq. k-itemsets
  - use database scan and pattern matching to collect counts for candidate itemsets

it may need to generate huge candidate sets

(burada k=level, yani 10^4 frequent 1-itemset, 10^7 tane candidate 2-itemset generate edecek!)

solution

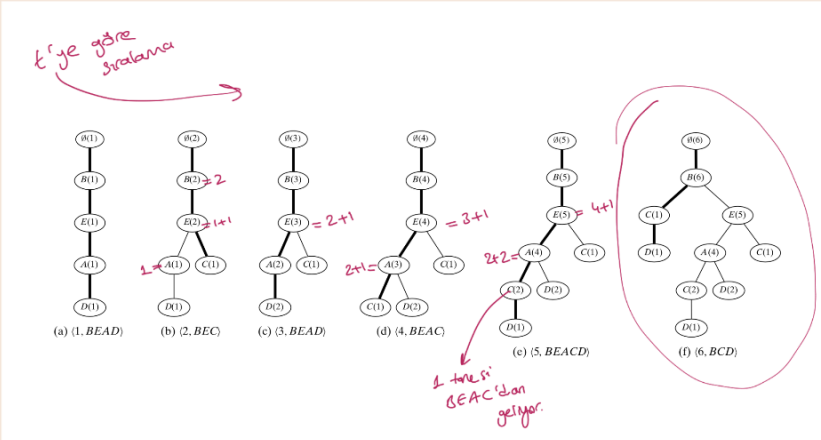
- Compress a large database into a compact, FP-tree structure

- Major Steps to mine FP-tree
  - Construct the FP-tree
  - For each frequent item i, compute projected FP-tree
  - Recursively mine conditional FP-trees and grow frequent patterns obtained
  - If the conditional FP-tree contains a single path, enumerate all patterns

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

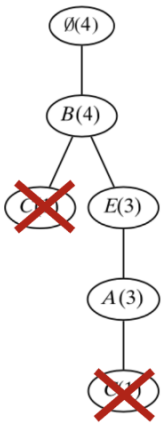
Transactions
BEAD
BEC
BEAD
BEAC
BEACD
BCD

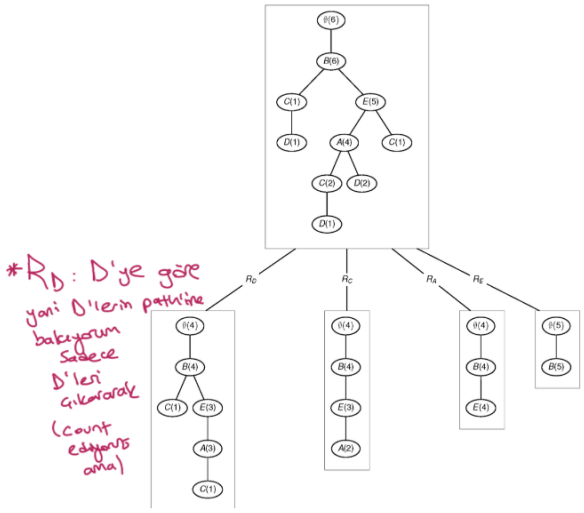
→ sort all transactions based on their frequency (B en çok bulunan)



After removing the infrequent item C (which has support 2), the resulting FP-Tree has only one path B(4)-E(3)-A(3)

We enumerate all the subsets of the path and prefix them with D to obtain the frequent itemsets DB(4), DE(4), DA(3), DBE(3), DBA(3), DEA(3), DBEA(3)





All frequent pattern tree projections. Figure 8.9 of the textbook.

- Pseudo code:

Algorithm 8.5: Algorithm FPGROWTH

```
// Initial Call: R ← FP-tree(D), P ← ∅, F ← ∅
FPGROWTH (R, P, F, minsup):
Remove infrequent items from R
if ISPATH(R) then // insert subsets of R into F
    foreach Y ⊆ R do
        X ← P ∪ Y
        sup(X) ← min_{x∈Y}{cnt(x)}
        F ← F ∪ {(X, sup(X))}
else // process projected FP-trees for each frequent item i
    foreach i ∈ R in increasing order of sup(i) do
        X ← P ∪ {i}
        sup(X) ← sup(i) // sum of cnt(i) for all nodes labeled i
        F ← F ∪ {(X, sup(X))}
        R_X ← ∅ // projected FP-tree for X
        foreach path ∈ PATHFROMROOT(i) do
            cnt(i) ← count of i in path
            Insert path, excluding i, into FP-tree R_X with count cnt(i)
        if R_X ≠ ∅ then FPGROWTH (R_X, X, F, minsup)
```

- Benefits of FP-Structure:

- Presesue complete info for frequent pattern mining
- Reduce irrelevant info; infrequent items are gone
- More frequently occuring items → more likely to be shared
- Never be larger than original database
- No candidate generation, no candidate test, no repeated scan of entire database.

*Rule Generation

- Given a frequent itemset L, find all non-empty subsets f ⊂ L such that L\f satisfies the minimum confidence requirement

ex: {A,B,C,D} is a frequent itemset, candidate rules: k = |L| = 4

ABC→D, ACD→B, BCD→A, ABD→C, AD→BC, BD→AC, AB→CD,
x2 (tersi) x2(tersi) x2(tersi) x2(tersi) x2 (tersi) x2(tersi) x2(tersi)

] $2^4 - 2 = 14$ association rules

or $2^k - 2$

- Efficient rule generation: Confidence have anti-monotone property if rules are generated from same itemset] c(ABC→D) ≥ c(AB→D) ≥ c(A→BCD) hepsi L'in icinde

*Rule Assessment Measures

- How to set minsup?
  - → if minsup too high; we could miss itemsets containing rare items
  - → if minsup too low; # of itemsets very large; computationally expensive!
  - → a single minsup threshold may not be effective

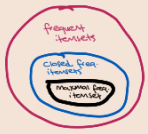
- Lift: Ratio of the observed joint probability of X and Y to the expected joint prob. (if they were statistically independent)

$$lift(X \Rightarrow Y) = \frac{sup(X \cup Y)}{sup(X).sup(Y)} = \frac{conf(X \Rightarrow Y)}{sup(Y)}$$

→ if = 1 then X and Y statistically independent

- Lift is measure of deviation
- Lift measures the surprise of rule (if ≈1; support of rule is expected)
- Should be much larger or much smaller than 1

✡ Summarizing Itemsets



*NOT: AB, A'nın superset'i

- Frequent Itemset: An itemset, whose support is greater than some user-specific minimum support
- Closed Frequent Itemset: An itemset X is closed if all supersets of X have strictly less support

$$sup(X) > sup(Y) \quad \text{for all } X \subset Y$$

- Maximal Frequent Itemset: An itemset X is maximal if has no frequent supersets → en alttakiler yani

$$M = \{X \mid X \in F \text{ and } \nexists Y \supset X \text{ such that } Y \in F\}$$

- Minimal Generators: A frequent itemset is a minimal generator if it has no subsets with same support

✡ Mining Frequent Sequences

• terminology: $s = ACT\,GAACG$

$r_1 = CGAAG$ subsequence of s

$r_2 = CTGA$ consecutive subseq. of s

$r_3 = ACT$ prefix of s

$r_4 = GAACG$ suffix of s

- Support of a sequence; $sup(r) = |\{s_i \in D \mid r \subseteq s_i\}|$
- Relative support; $rsup(r) = sup(r) / N$ → # of seq.
- r is frequent if $sup(r) \geq minsup$

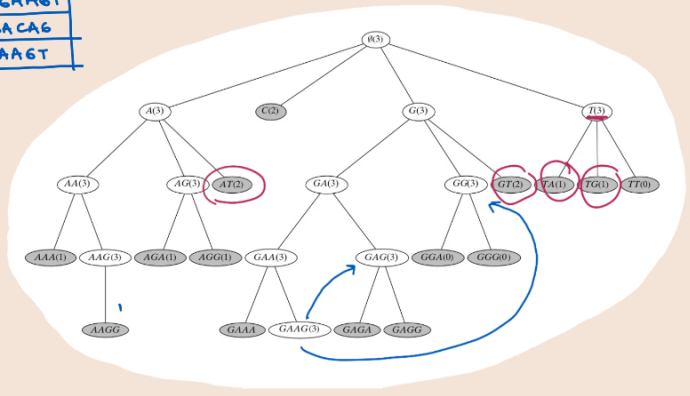
→ GSP Algorithm

- It searches the sequence prefix tree using a level-wise

ex

Id	Sequence
s_1	CAGAAGT
s_2	TGACAG
s_3	GAAGT

minsup=3



» Given set of freq. sequences at level k, generate candidates for level $k+1$, and compute support, prune not frequent ones.

» For each leaf r_a, sequence is extended with last symbol of other leaf r_b that shares same prefix. So $r_{ab} = r_a + r_b[k]$

- Pseudocode:

Algorithm 10.1: Algorithm GSP

```
GSP (D, Σ, minsup):
1  F ← ∅
2  C(1) ← {∅} // Initial prefix tree with single symbols
3  foreach s ∈ Σ do Add s as child of ∅ in C(1) with sup(s) ← 0
4  k ← 1 // k denotes the level
5  while C(k) ≠ ∅ do
6      ComputeSupport (C(k), D)
7      foreach leaf s ∈ C(k) do
8          if sup(r) ≥ minsup then F ← F ∪ {(r, sup(r))}
9          else remove s from C(k)
10     C(k+1) ← ExtendPrefixTree (C(k))
11     k ← k + 1
12 return F(k)

ComputeSupport (C(k), D):
13 foreach s_i ∈ D do
14     foreach r ∈ C(k) do
15         if r ⊆ s_i then sup(r) ← sup(r) + 1

ExtendPrefixTree (C(k)):
16 foreach leaf r_a ∈ C(k) do
17     foreach leaf r_b ∈ Children(Parent(r_a)) do
18         r_ab ← r_a + r_b[k] // extend r_a with last item of r_b
           // prune if there are any infrequent subsequences
19         if r_c ∈ C(k), for all r_c ⊂ r_ab, such that |r_c| = |r_ab| − 1 then
20             Add r_ab as child of r_a with sup(r_ab) ← 0
21     if no extensions from r_a then
22         remove r_a, and all ancestors of r_a with no extensions, from C(k)
23 return C(k)
```

→ Sequential Pattern Mining

- In several application we deal with sequence of itemsets (not items)
- Association rules do not consider order of transaction but in many application, ordering is important. (Buying bed ⇒ buying bed sheets)
- In Sequential Pattern Mining, a sequence is an ordered list of itemsets And it consist of categorical data. When numerical → apply time series

ex: $I = \{1,2,3,4,5,6,7,8,9\}$

Sequence $\langle \{3\},\{4,5\},\{8\} \rangle$ is subseq. of $\langle \{6\},\{3,7\},\{9\},\{4,5,8\},\{3,8\} \rangle$

$\langle \{3\},\{8\} \rangle$ is not contained in $\langle \{3,8\} \rangle$ or vice-versa

Size of sequence $\langle \{3\},\{4,5\},\{8\} \rangle$ is 3, length is 4

- **Goal:** is to find all the sequences that have user-specified min. support (frequent sequences)
- $sup(r) = |\{s : r \text{ is a subsequence of } s\}|$ → $rsup(r) = sup(r) / N$

Customer ID	Transaction Time	Items Bought
0	June 25 '93	30
0	June 30 '93	90
1	June 10 '93	10,20
1	June 15 '93	30
1	June 20 '93	40,60,70
2	June 25 '93	30,50,70
3	June 25 '93	30
3	June 30 '93	40,70
3	July 25 '93	90
4	June 12 '93	90

Customer ID	Customer Sequence
0	< (30) (90) >
1	< (10 20) (30) (40 60 70) >
2	< (30 50 70) >
3	< (30) (40 70) (90) >
4	< (90) >

Frequent Sequence	Support	Customer ID
(30)	4	0,1,2,3
(40)	2	1,3
(70)	3	1,2,3
(90)	3	0,3,4
(30) (40)	2	1,3
(30) (70)	2	1,3
(30) (90)	2	0,3
(40 70)	2	1,3
(30) (40 70)	2	1,3

Maximal seq with support of 40%
< (30) (90) >
< (30) (40 70) >

• Maximal Frequent itemset: An itemset is maximal if none of its supersets is frequent

→ None of supersets is frequent for (40 70)

10 20 30 40 70
30 40 70 90
30 40 70 → frequent superset

- Level-by-level:
  1) Generate candidate sequences
  2) Scan database to collect support counts
  3) Use Apriori property to prune candidates

- Limitations:
  - It can generate massive amount of candidates
  - It requires many scans of the database