Exercise 3.2

Let *freq_count* be a minimal amount of support count needed for an itemset to call it frequent. Having min_sup equal to 60% and cardinality of database D equal to 4 we can say that an itemset's support count must be at least 2.4 to call him frequent. Since support count it's the natural number we have to round it up. Therefore our *freq_count* is equal to 3. We will be using this number during the limitation of candidate itemsets.
While the column “date” in the database D is irrelevant for us, we're not going to include it in this solution.

(a)

scan →

D:

TID	**itemset**
T1	{K,A,D,B}
T2	{D,A,C,E,B}
T3	{C,A,B,E}
T4	{B,A,D}

→ *C1:*

itemset	**count**
{A}	4
{B}	4
{C}	2
{D}	3
{E}	2
{K}	1

join(L1,L1) + prune → scan →

L1:

itemset	**count**
{A}	4
{B}	4
{D}	3

C2:

itemset
{A,B}
{A,D}
{B,D}

C2:

itemset
{A,B}
{A,D}
{B,D}

Since each itemset held in C2 has count bigger than *freq_count* or equal to it we are safe to say that L2 is equal to C2.

join(L2,L2) →

C3:

itemset
{A,B,D}

prune → *C3:*

itemset
{A,B,D}

scan →

C3:

itemset	**Count**
{A,B,D}	3

→ *L3:*

itemset	**count**
{A,B,D}	3

As C4 would be obtained from joining L3 with itself and L3's cardinality is equal to 1 then we know that C4 is in fact an empty set which is condition that ends our algorithm.

The frequent itemsets are the union of L1, L2 and L3, thus frequent itemsets = {{A}, {B}, {D}, {A,B}, {A,D}, {B,D}, {A,B,D}}.

(b) The first step we have to do is the same as the one in the Apriori Algorithm: we need to count occurrence of every item in the database and then, basing on the *freq_count*, discard the ones that have lower count. What are we left with is the table called “header table” which will be useful afterwards.

D:

TID	**itemset**
T1	{K,A,D,B}
T2	{D,A,C,E,B}
T3	{C,A,B,E}
T4	{B,A,D}

item	**count**
A	4
B	4
C	2
D	3
E	2
K	1

Header table:

item	**count**
A	4
B	4
D	3

After we have established which items are frequent, we need to order the content of itemsets stored in the database D so we can then build a fp-tree.

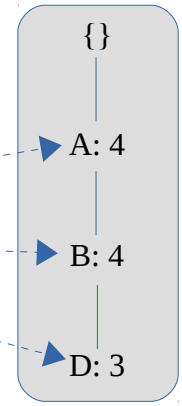
D:

TID	itemset	Ordered itemsets (excluding not frequent items)
T1	{K,A,D,B}	{A,B,D}
T2	{D,A,C,E,B}	{A,B,D}
T3	{C,A,B,E}	{A,B}
T4	{B,A,D}	{A,B,D}

Now, using ordered itemsets and header table, we are able to build a proper fp-tree.

Header table:

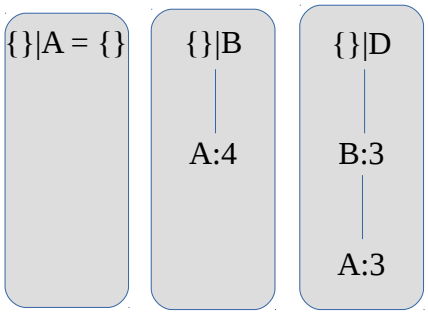
item	count	head
A	4	•
B	4	•
D	3	•



After we've built our fp-tree we can compose conditional pattern base table.

item	Conditional pattern base
A	{}
B	A: 4
D	AB: 3

Now, from each conditional pattern base we have to construct an fp-tree:



When the fp-trees are made we can recursively mine them and obtain frequent patterns concerning each item.

All frequent patterns concerning…

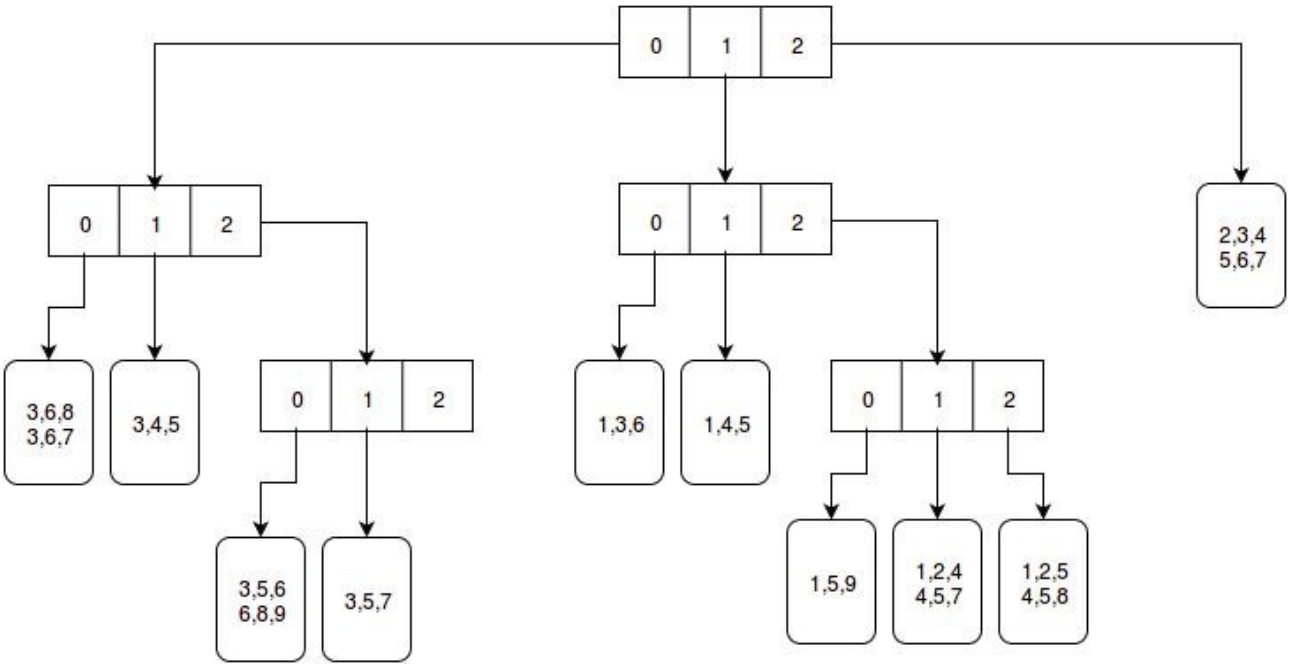
- A : {A}
- B : {B}, {A,B}
- C : {D}, {B,D}. {A,D}, {A,B,D}

The frequent itemsets listed above are the result of applying the fp-growth algorithm.

At the end: frequent itemsets = {{A}, {B}, {D}, {A,B}, {A,D}, {B,D}, {A,B,D}}.

Exercise 3.3

(a) The following picture presents the hash tree we've constructed from the given candidates and the hash function h. Although the cost of building such tree may be big, it is amortized by possibility of checking multiple transactions.



(b) When we search for 3-itemsets that appear in (1,3,7,8,9) we have to visit five nodes and check six candidates. The following picture presents the searching process.
The numbers written in italic show the current item we are looking for. Nodes that were visited have blue background.
It's worth mentioning that without a tree we would have to check 15 candidates which is almost 3 times more than we have checked with the help of a hash tree.