Apriori implementation using Hadoop

Apache Hadoop (pronunciation: /həˈduːp/) is an open-source software framework used for distributed storage and processing of big data sets using the MapReduce programming model.
MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

One of our tasks was to implement mapper and reducer methods of a hadoop framework. According to these methods process will be parallelized and run on parallel machines and so proccess will be much faster.
We have 2 map methods.
One of them is base case. 1mapper. We simply iterated through all elements of transaction and saved every element as a key with a value 1.

```
for(Integer element : txn) {
    String individualItem="["+String.valueOf(element)+"]";
    item.set(individualItem);
    context.write(item, one);
}
```

Our reduce step collects all elements with same case, summing their values up, so that at the end we will be left with unique keys and values holding total information of all previous values assigned to key.

```
public void reduce(Text itemSet, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException {
    //int itemCount=
    /** COMPLETE **/
    IntWritable result = new IntWritable();
    Double minSup = Double.parseDouble(context.getConfiguration().get("minSup"));
    Integer numTxns = context.getConfiguration().getInt("numTxns", 2);
    int itemCount = 0;
    for (IntWritable val : values) {
        itemCount+= val.get();
    }
    if( AprioriUtils.hasMinSupport( minSup, numTxns, itemCount)){
         result.set(itemCount);
         context.write(itemSet, result);
    }
```

One main point to catch our attention is the line
if(AprioriUtils.hasMinSupport(minSup, numTxns, itemCount))
saying that if itemCount which summed up all values accoring to current key has min support we will write this itemset as key to our output and itemCount as its value. However what will happen to itemsets do not have min support, will we write their data somewhere? Actually not, we stop to hold all data which does not have min support and this is called prunning. It makes this algorithm really efficient because we continue keeping track of only that records which are matter. For d items there are 2^d different itemset what is really huge number if we gonna hold data for all, but cutting unnecessary data on each step leads us to hold only neccessary data and decreasing space to the minimum needed. Although if our min support equals 0, pruning will bring no use and our space will be filled with 2^d elements.

So far, I have told about one mapper method and reducer, another mapper method is left. This method can be called only of k-1 map exists. It accesses the content of k-1 map and self joins it with itself. Generated itemsets are called candidateItemsets containing all possible candidates and are useful for generating a trie. My trie implementation can be seen from previous assignment.

```
public void map(LongWritable key, Text txnRecord, Context context)
        throws IOException, InterruptedException {
    Transaction txn = AprioriUtils.getTransaction((int) key.get(), txnRecord.toString());
    /** COMPLETE **/
    Collections.sort(txn);
    ArrayList<ItemSet> ItemSetRight= new ArrayList<>();
    trie.findItemSets(ItemSetRight, txn);
    for(ItemSet val : ItemSetRight) {
        System.out.println("itemSetRight: " + val.toString());
    }
    for(ItemSet element:ItemSetRight){
        String individualItem=element.toString();
        item.set(individualItem);
        context.write(item, one);
    }
```

Our trie.findItemSets line returns all matched itemsets from transaction and we we write each itemset from matched itemsets with again value one. Which later will be also collected and filtered by our reducer method.

Talking more about self joining and frequently raised question as why do we need to compute k-1 candidates before computing candidates of step k. Why could not we complete step k+1 from data of step k-1. The answer is simple and can be found if we look deeper how a function getCandidates works.

```
public static List<ItemSet> getCandidateItemSets(List<ItemSet> prevPassItemSets, int itemSetSize)
    List<ItemSet> candidateItemSets = new ArrayList<>();
    Map<Integer, ItemSet> itemSetMap = generateItemSetMap(prevPassItemSets);
    Collections.sort(prevPassItemSets);
    int prevPassItemSetsSize = prevPassItemSets.size();

    /** COMPLETE **/


    for(int index1 = 0; index1 < prevPassItemSetsSize; index1++){
        for(int index2 = index1 + 1; index2 < prevPassItemSetsSize; index2++) {
            boolean not_equal = false;
            for(int c = 0; c < itemSetSize  - 1; c++) {
                if(prevPassItemSets.get(index1).get(c) != prevPassItemSets.get(index2).get(c)) {
                    not_equal = true;
                    break;
                }
            }
            if(not_equal == false) {
                ItemSet newCreateItemSet = new ItemSet();
                if(itemSetSize > 1){
                    for(int i1=0;i1<itemSetSize-1;i1++){
                        newCreateItemSet.add(prevPassItemSets.get(index1).get(i1));
                    }
                }
                newCreateItemSet.add(prevPassItemSets.get(index1).get(itemSetSize - 1));
                newCreateItemSet.add(prevPassItemSets.get(index2).get(itemSetSize - 1));
                if(prune(itemSetMap, newCreateItemSet) == false) continue;
                candidateItemSets.add(newCreateItemSet);
            }
            else break;
        }
    }

    for(ItemSet val : candidateItemSets) {
        System.out.println("Candidates: " + val.toString());
    }
    return candidateItemSets;
}
```

This function takes List of itemsets and returns list of itemsets with the length bigger than length of input list. So is it possible to create list with length 4 by adding 1 item to a list with length 2? Of course, not, so as it is impossible to generate list of size k+1 by adding 1 to list of size k-1, it is impossible to generate candidateItemSets not having a list of prevousItemSets with a length differing only by 1.