In [1]:

```python
from pyspark import SparkConf, SparkContext
import math
import itertools as it
from operator import add
```

In [2]:

```python
sc = SparkContext('local[*]')
```

In [3]:

```python
def parse_line(line):
    return [x for x in line.strip().split(' ')]


file = sc.textFile('4.txt')
sessions = file.map(parse_line)
# sessions.take(5)
```

In [4]:

```python
def session_common_items(session, common_items):
    if (len(common_items) > 0):
        return [item for item in session if (item in common_items)]
    else:
        return session

def common_combinations_tuple(session, common_items, size):
    sess_common_items = sorted(session_common_items(session, common_items))
    return [(comb, 1) for comb in it.combinations(sess_common_items, size)]

def common_combinations_dict(sessions, common_items, size, threshold):
    tuples = sessions.flatMap(lambda ses: common_combinations_tuple(ses, common_items, size)) \
                .reduceByKey(add) \
                .filter(lambda x: x[1] >= threshold) \
                .collect()

    return dict(tuples)

def get_items(common_dict):
    item_list = sc.parallelize(common_dict.keys()) \
            .flatMap(lambda x: x) \
            .collect()

    return item_list


singles = common_combinations_dict(sessions, [], 1, 100)
doubles = common_combinations_dict(sessions, get_items(singles), 2, 100)
triples = common_combinations_dict(sessions, get_items(doubles), 3, 100)
# print(len(singles))
# print(len(doubles))
# print(len(triples))
```

In [5]:

```python
def rule(items, combination, confidence):
    rule_if = list(combination)
    rule_then = list(set(items) - set(combination))
    return ((rule_if, rule_then), confidence)

def combination_rules(items, support, common_items):
    rules = []
    for comb in it.combinations(items, len(items) - 1):
        if comb in common_items:
            confidence = support / common_items[comb]
            rules.append(rule(items, comb, confidence))
    return rules

def rules(combinations, common_items):
    rules_lists = [combination_rules(comb, supp, common_items) for comb, supp in combinations.items
()]
    rules_list = list(it.chain(*rules_lists))

    rules_list.sort()
    rules_list.sort(key=lambda r: r[1], reverse=True)

    return rules_list


rules_doubles = rules(doubles, singles)
rules_triples = rules(triples, doubles)
```

In [6]:

```
def parse_rule(rule_tuple):
    condition, confidence = rule_tuple
    rule_if, rule_then = condition

    return "{} [{}] {}".format(" ".join(rule_if), " ".join(rule_then), confidence)

def parse_rules(rules_tuple):
    return [parse_rule(tup) for tup in rules_tuple]


doubles_output = parse_rules(rules_doubles)
triples_output = parse_rules(rules_triples)
# print(doubles_output)
# print(triples_output)
```

In [7]:

```
with open("result_doubles.txt", "w") as outfile:
    outfile.write("\n".join(doubles_output))

with open("result_triples.txt", "w") as outfile:
    outfile.write("\n".join(triples_output))
```

In [8]:

```
sc.stop()
```