In [1]:

```
from itertools import combinations
import pandas as pd
```

1. ACLOSE algorithm

In [2]:

```
def prune_itemsets(itemsets, transactions, min_support):
    item_counts = {}
    for transaction in transactions:
        for itemset in itemsets:
            if itemset.issubset(transaction):
                item_counts[itemset] = item_counts.get(itemset, 0) + 1
    num_transactions = float(len(transactions))

    pruned_item_counts = {}
    for itemset, count in item_counts.items():
        support = count / num_transactions
        if support >= min_support:
            pruned_item_counts[itemset] = count

    return pruned_item_counts
```

In [3]:

```
def is_closed(itemset, all_frequent_itemsets, support_lookup):
    for frequent_itemset in all_frequent_itemsets:
        if itemset != frequent_itemset and set(itemset).issubset(set(frequent_itemset)):
            if support_lookup[itemset] == support_lookup[frequent_itemset]:
                return False

    return True
```

In [4]:

```
def aclose(transactions, min_support):
    all_frequent_itemsets = []
    closed_itemsets = []
    support_lookup = {}

    k = 1
    while True:
        candidate_itemsets = set()
        if k == 1:
            for transaction in transactions:
                for item in transaction:
                    candidate_itemsets.add(frozenset([item]))
        else:
            for itemset1 in frequent_itemsets:
                for itemset2 in frequent_itemsets:
                    union = itemset1.union(itemset2)
                    if len(union) == k and union not in candidate_itemsets:
                        candidate_itemsets.add(union)

        item_counts = prune_itemsets(candidate_itemsets, transactions, min_support)
        frequent_itemsets = list(item_counts.keys())
        if len(frequent_itemsets) == 0:
            break
        all_frequent_itemsets.extend(frequent_itemsets)
        support_lookup = support_lookup | item_counts

        if k > 2:
            for itemset in frequent_itemsets:
                if is_closed(itemset, all_frequent_itemsets, support_lookup):
                    closed_itemsets.append(itemset)
        k += 1

    return closed_itemsets, support_lookup
```

In [5]:

```
transactions = [
     ['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
     ['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],
     ['Milk', 'Apple', 'Kidney Beans', 'Eggs'],
     ['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],
     ['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']
]
min_support = 0.4
closed_itemsets, support_lookup = aclose(transactions, min_support)
closed_itemsets
```

Out[5]:

```
[frozenset({'Eggs', 'Nutmeg', 'Onion'}),
 frozenset({'Eggs', 'Nutmeg', 'Yogurt'}),
 frozenset({'Kidney Beans', 'Nutmeg', 'Onion'}),
 frozenset({'Eggs', 'Kidney Beans', 'Milk'}),
 frozenset({'Eggs', 'Onion', 'Yogurt'}),
 frozenset({'Kidney Beans', 'Onion', 'Yogurt'}),
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg'}),
 frozenset({'Nutmeg', 'Onion', 'Yogurt'}),
 frozenset({'Eggs', 'Kidney Beans', 'Onion'}),
 frozenset({'Kidney Beans', 'Nutmeg', 'Yogurt'}),
 frozenset({'Eggs', 'Kidney Beans', 'Yogurt'}),
 frozenset({'Kidney Beans', 'Milk', 'Yogurt'}),
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg', 'Yogurt'}),
 frozenset({'Eggs', 'Kidney Beans', 'Onion', 'Yogurt'}),
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion'}),
 frozenset({'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'}),
 frozenset({'Eggs', 'Nutmeg', 'Onion', 'Yogurt'}),
 frozenset({'Eggs', 'Kidney Beans', 'Nutmeg', 'Onion', 'Yogurt'})]
```

2. Pincer Search

In [6]:

```python
def mfcs_prune(old_ckplus1,curr_mfcs):
    new_ckplus1=[]
    for c in old_ckplus1:
        for itemset in curr_mfcs:
            if set(c).issubset(set(itemset)):
                new_ckplus1.append(c)
    return new_ckplus1
```

In [7]:

```python
def mfs_prune(old_ck,curr_mfs):
    new_ck=old_ck.copy()
    for c in old_ck.copy():
        for itemset in curr_mfs:
            if set(c).issubset(set(itemset)):
                new_ck.remove(c)
    return new_ck
```

In [8]:

```
def mfcs_gen(sk,mfcs):
    mfcs = mfcs.copy()

    for infrequent_itemset in sk:

        for mfcs_itemset in mfcs.copy():

            # If infrequent itemset is a subset of mfcs itemset
            if all(_item in mfcs_itemset for _item in infrequent_itemset):
                mfcs.remove(mfcs_itemset)

                for item in infrequent_itemset:
                    updated_mfcs_itemset = mfcs_itemset.copy()
                    updated_mfcs_itemset.remove(item)

                    if not any(all(item in _mfcs_itemset for item in updated_mfcs_itemset) for _mfcs_itemset in mfcs):
                        mfcs.append(updated_mfcs_itemset)

    return mfcs
```

In [9]:

```
def gen_next_ck(ck,k):
    num_freq=len(ck)
    newck=[]
    for i in range(num_freq):
        j=i+1
        while((j<num_freq) and (ck[i][:k-1]==ck[j][:k-1])):
            new_itemset=ck[i][:k-1]+[ck[i][k-1]]+[ck[j][k-1]]
            insert_in_new=False
            if k==1:
                insert_in_new=True
            elif k==2 and (new_itemset[-2:] in ck):
                insert_in_new=True
            else:
                for a in combinations(ck[:-2],k-2):
                    if(list(a)+ck[-2:] not in ck):
                        insert_in_new=False
            if insert_in_new:
                newck.append(new_itemset)
            j+=1
    return newck
```

In [10]:

```
def pincerSearch(txn,min_supp):
    items = set()
    for transaction in txn:
        items.update(transaction)
    level_k=1
    cand_freq_itemsets=[[item] for item in items]
    level_freq_itemsets=[]
    level_infreq_itemsets=[]

    mfcs=[items.copy()]
    mfs=[]

    print(f"MFCS={mfcs}\n")
    print(f"MFS={mfs}\n")

    while len(cand_freq_itemsets)!=0:
        print(f"Level {level_k}")
        print(f"C{level_k} = {cand_freq_itemsets}")


        cand_freq_itemsets_cnt=[0]*len(cand_freq_itemsets)
        mfcs_itemsets_cnt=[0]*len(mfcs)

        # step 1- read txn from db and get support for ck and mfcs
        for each in txn:
            for i,itemset in enumerate(cand_freq_itemsets):
                if set(itemset).issubset(each):
                    cand_freq_itemsets_cnt[i]+=1
            for i,itemset in enumerate(mfcs):
                if set(itemset).issubset(each):
                    mfcs_itemsets_cnt[i]+=1

        for itemset,supp in zip(cand_freq_itemsets,cand_freq_itemsets_cnt):
            print(f"{itemset} - {supp}")
        print('\n')
        for itemset,supp in zip(mfcs,mfcs_itemsets_cnt):
            print(f"{itemset} - {supp}")
        print('\n')

        # step 2 - add freq itemsets from mfcs to mfs
        for itemset,supp in zip(mfcs,mfcs_itemsets_cnt):
            if (itemset not in mfs) and supp>=min_supp:
                mfs.append(itemset)
        print(f"MFS - {mfs}")
        level_freq_itemsets=[]
        level_infreq_itemsets=[]
        # step 3 - infreq itemsets in ck makes sk
        for itemset,supp in zip(cand_freq_itemsets,cand_freq_itemsets_cnt):
```

```
        for itemset,supp in zip(cand_freq_itemsets,cand_freq_itemsets_cnt):
            if supp>=min_supp:
                level_freq_itemsets.append(itemset)
            if supp<min_supp:
                level_infreq_itemsets.append(itemset)
        print(f"L{level_k} - {level_freq_itemsets}")
        print(f"S{level_k} - {level_infreq_itemsets}")

        # step 4 - mfcs-gen if sk is non empty
        mfcs=mfcs_gen(level_infreq_itemsets,mfcs)
        print(f"MFCS - {mfcs}")

        # step 5 - pruning cand using mfs
        print(f"C{level_k} was - {level_freq_itemsets}")
        level_freq_itemsets=mfs_prune(level_freq_itemsets,mfs)
        print(f"After pruning,L{level_k} - {level_freq_itemsets}")

        # step 6 - gen next ck from old ck
        cand_freq_itemsets=gen_next_ck(cand_freq_itemsets,level_k)

        # step 7 - prune new ck with mfcs
        cand_freq_itemsets=mfcs_prune(cand_freq_itemsets,mfcs)

        level_k+=1
    return mfs
```

In [11]:

```
transactions = [
                {1, 5, 6, 8},
                {2, 4, 8},
                {4, 5, 7},
                {2, 3},
                {5, 6, 7},
                {2, 3, 4},
                {2, 6, 7, 9},
                {5},
                {8},
                {3, 5, 7},
                {3, 5, 7},
                {5, 6, 8},
                {2, 4, 6, 7},
                {1, 3, 5, 7},
                {2, 3, 9},
        ]

min_support_count = 3

MFS = pincerSearch(transactions, min_support_count)
print("MFS = {}".format(MFS))
```

```
MFCS=[{1, 2, 3, 4, 5, 6, 7, 8, 9}]

MFS=[]

Level 1
C1 = [[1], [2], [3], [4], [5], [6], [7], [8], [9]]
[1] - 2
[2] - 6
[3] - 6
[4] - 4
[5] - 8
[6] - 5
[7] - 7
[8] - 4
[9] - 2


{1, 2, 3, 4, 5, 6, 7, 8, 9} - 0


MFS - []
L1 - [[2], [3], [4], [5], [6], [7], [8]]
S1 - [[1], [9]]
MFCS - [{2, 3, 4, 5, 6, 7, 8}]
C1 was - [[2], [3], [4], [5], [6], [7], [8]]
After pruning,L1 - [[2], [3], [4], [5], [6], [7], [8]]
Level 2
C2 = [[2, 3], [2, 4], [2, 5], [2, 6], [2, 7], [2, 8], [3, 4], [3, 5], [3, 6], [3, 7], [3, 8], [4, 5], [4, 6], [4, 7], [4, 8], [5, 6], [5,
7], [5, 8], [6, 7], [6, 8], [7, 8]]
[2, 3] - 3
[2, 4] - 3
[2, 5] - 0
[2, 6] - 2
[2, 7] - 2
[2, 8] - 1
[3, 4] - 1
[3, 5] - 3
[3, 6] - 0
[3, 7] - 3
[3, 8] - 0
[4, 5] - 1
[4, 6] - 1
[4, 7] - 2
[4, 8] - 1
[5, 6] - 3
[5, 7] - 5
[5, 8] - 2
[6, 7] - 3
```

```
[6, 8] - 2
[7, 8] - 0


{2, 3, 4, 5, 6, 7, 8} - 0


MFS - []
L2 - [[2, 3], [2, 4], [3, 5], [3, 7], [5, 6], [5, 7], [6, 7]]
S2 - [[2, 5], [2, 6], [2, 7], [2, 8], [3, 4], [3, 6], [3, 8], [4, 5], [4, 6], [4, 7], [4, 8], [5, 8], [6, 8], [7, 8]]
MFCS - [{2, 4}, {2, 3}, {3, 5, 7}, {5, 6, 7}, {8}]
C2 was - [[2, 3], [2, 4], [3, 5], [3, 7], [5, 6], [5, 7], [6, 7]]
After pruning,L2 - [[2, 3], [2, 4], [3, 5], [3, 7], [5, 6], [5, 7], [6, 7]]
Level 3
C3 = [[3, 5, 7], [5, 6, 7]]
[3, 5, 7] - 3
[5, 6, 7] - 1


{2, 4} - 3
{2, 3} - 3
{3, 5, 7} - 3
{5, 6, 7} - 1
{8} - 4


MFS - [{2, 4}, {2, 3}, {3, 5, 7}, {8}]
L3 - [[3, 5, 7]]
S3 - [[5, 6, 7]]
MFCS - [{2, 4}, {2, 3}, {3, 5, 7}, {8}, {6, 7}, {5, 6}]
C3 was - [[3, 5, 7]]
After pruning,L3 - []
MFS = [{2, 4}, {2, 3}, {3, 5, 7}, {8}]
```