Data Mining
Assignment 2

September 24, 2025

Viet Huy Duong
811433146

---

1. Implementing Apriori algorithm (using C++, Java, or any programming language). Your program should be able to accept two parameters with input: filename and a minimal support level. For instance, "myapriori filename 15", where "myapriori" is the execution file, and 15 means a frequent itemset has frequency of 15% of the entire transactions in "filename". The file format is as follows: each line corresponds to a transaction (no transaction id) and each item in the transaction is separated by a space. Your program should output all the frequent itemsets in the input file with the specified minimal support level. (10 pts)

   (a) A detailed Pseudo code including the necessary data structure for implementation.

   (b) Source code.

   (c) The running results (screen captures) of the following input file and minimal support (10%, 20%, 30%, 50%).

(a) A detailed Pseudo code including the necessary data structure for imple- mentation.

Algorithm 1 Support Function

```
function FindFrequentOne(transactions, min_support_count)
    item_counts ← DefaultDict(int)
    for each transaction in transactions do
        for each item in transaction do
            item_counts[item] ← item_counts[item] + 1
        end for
    end for
    return {frozenset([item]) : count | count $\geq$ min_support_count}
end function
function GenerateCandidates(prev_frequent, k)
    candidates ← $\emptyset$
    for each pair (set1, set2) in prev_frequent do
        union ← set1 $\cup$ set2
        if len(union) = k then
            candidates ← candidates $\cup$ {union}
        end if
    end for
    return candidates
end function
function CountSupports(candidates, transactions)
    supports ← DefaultDict(int)
    for each transaction in transactions do
        for each candidate in candidates do
            if candidate $\subseteq$ transaction then
                supports[candidate] ← supports[candidate] + 1
            end if
        end for
    end for
    return supports
end function
```

Algorithm 2 Apriori

```
procedure APRIORI(transactions, min_support_perc)
    total_transactions ← len(transactions)
    min_support_count ← min_support_perc/100 × total_transactions
    frequent_itemsets ← FindFrequentOne(transactions, min_support_count)
    all_frequent ← frequent_itemsets
    k ← 2
    while frequent_itemsets is not empty do
        candidates ← GenerateCandidates(frequent_itemsets, k)
        if candidates is empty then
            break
        end if
        supports ← CountSupports(candidates, transactions)
        frequent_itemsets ← $\{\text{itemset:count} | \text{count} \geq \text{min\_support\_count}\}$
        all_frequent ← all_frequent $\cup$ frequent_itemsets
        k ← k + 1
    end while
    return all_frequent
end procedure
```

The main idea to solve above problem is base on dynamic programming, first of all, we count and get the set of one item have frequent greater than the threashold, then, the core idea is defination the candidate of k length items set by combine the k-1 length items set, that is because if a set of k elements satisfies the condition, then it proves that every set of k-1 elements in it must satisfy the condition.

For example:

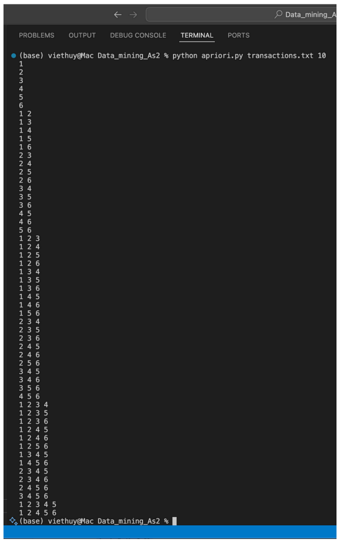
We have 2 set $\{butter, cake, icecream\}$ and $\{icecream, meat, cake\}$, the combination of them is $\{butter, cake, icecream, meat\}$ and that combination is not insure the set 4 will satisfies the condition, that just be a candidate for the next support count check progress.

After that, we going to recheck the fequent of candidate set, if satisfies, candidate set will be save into the frequent itemsets, which will be used for the next $GenerateCandidate$ k value

(b) Source code.

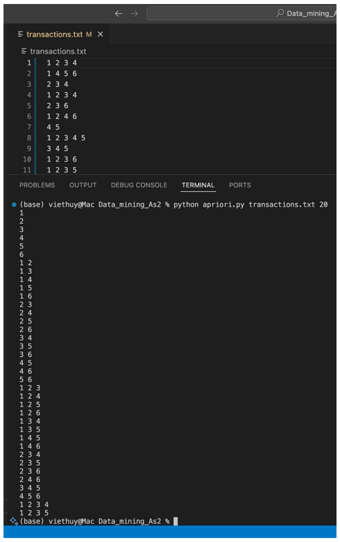
```
import sys
from collections import defaultdict
import math

def find_frequent_1_itemsets(transactions, min_support_count):
    item_counts = defaultdict(int)
    for transaction in transactions:
        for item in transaction:
            item_counts[item]+=1
    frequent_items ={frozenset([item]):count for item,count in
    ↪  item_counts.items() if count>=min_support_count}
    return frequent_items

def generate_candidates(prev_frequent, k):
    candidates= set()
    prev_items= list(prev_frequent.keys())
    for i in range(len(prev_items)):
        for j in range(i+1, len(prev_items)):
            set1= prev_items[i]
            set2 =prev_items[j]
            union =set1|set2
            if len(union)== k:
                candidates.add(union)
    return candidates

def count_supports(candidates,transactions):
    supports =defaultdict(int)
    for transaction in transactions:
        for candidate in candidates:
            if candidate.issubset(transaction):
                supports[candidate] +=1
    return supports

def main():
    filename = sys.argv[1]
    try:
        min_support_perc = float(sys.argv[2])
    except ValueError:
        print("lack of minimum fequent percentage")
        sys.exit(1)
    try:
        with open(filename, 'r') as f:
            transactions = [set(line.strip().split()) for line in
            ↪  f if line.strip()]
    except FileNotFoundError:
        print(f"File {filename} not available")
        sys.exit(1)

    if not transactions:
        print("No transactions")
        return

    total_transactions = len(transactions)
    min_support_count =math.ceil((min_support_perc / 100) *
    ↪  total_transactions)

    frequent_itemsets = find_frequent_1_itemsets(transactions,
    ↪  min_support_count)
    all_frequent = frequent_itemsets.copy()

    k = 2
    while frequent_itemsets:
        candidates = generate_candidates(frequent_itemsets, k)
        if not candidates:
            break
        supports =count_supports(candidates, transactions)
        frequent_itemsets = {itemset: count for itemset, count in
        ↪  supports.items() if count >= min_support_count}
        all_frequent.update(frequent_itemsets)
        k +=1

    if len(sys.argv) > 3 and sys.argv[3] == "1":
        for itemset in sorted(all_frequent.keys(), key=lambda x:
        ↪  (len(x), sorted(x))):
            print(' '.join(sorted(itemset)) + f" (support:
            ↪  {all_frequent[itemset]} with frequent:
            ↪  {all_frequent[itemset]*100/total_transactions}%)"
            ↪  )
    else:
        for itemset in sorted(all_frequent.keys(), key=lambda x:
         ↪  (len(x), sorted(x))):
            print(' '.join(sorted(itemset)))

if __name__ == "__main__":
    main()
```

(c) The running results (screen captures) of the following input file and minimal support (10%, 20%, 30%, 50%)



```
(base) viethuy@Mac Data_mining_As2 % python apriori.py transactions.txt 10
1
2
3
4
5
6
1 2
1 3
1 4
1 5
1 6
2 3
2 4
2 5
2 6
3 4
3 5
3 6
4 5
4 6
5 6
1 2 3
1 2 4
1 2 5
1 2 6
1 3 4
1 3 5
1 3 6
1 4 5
1 4 6
1 5 6
2 3 4
2 3 5
2 3 6
2 4 5
2 4 6
2 5 6
3 4 5
3 4 6
3 5 6
4 5 6
1 2 3 4
1 2 3 5
1 2 3 6
1 2 4 5
1 2 4 6
1 2 5 6
1 3 4 5
1 4 5 6
2 3 4 5
2 3 4 6
2 4 5 6
3 4 5 6
1 2 3 4 5
1 2 4 5 6
(base) viethuy@Mac Data_mining_As2 %
```

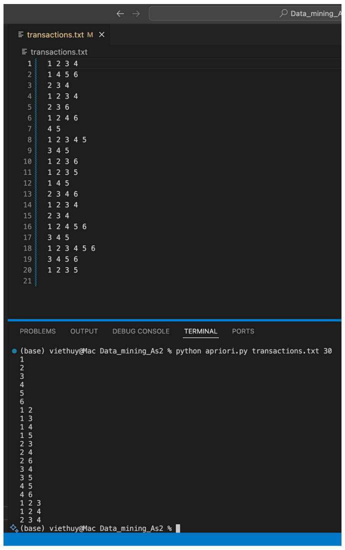
10%



```
1 2 3 4
1 4 5 6
2 3 4
1 2 3 4
2 3 6
1 2 4 6
4 5
1 2 3 4 5
3 4 5
1 2 3 6
1 2 3 5
```

```
(base) viethuy@Mac Data_mining_As2 % python apriori.py transactions.txt 20
1
2
3
4
5
6
1 2
1 3
1 4
1 5
1 6
2 3
2 4
2 5
2 6
3 4
3 5
3 6
4 5
4 6
5 6
1 2 3
1 2 4
1 2 5
1 2 6
1 3 4
1 3 5
1 4 5
1 4 6
2 3 4
2 3 5
2 3 6
2 4 6
3 4 5
4 5 6
1 2 3 4
1 2 3 5
(base) viethuy@Mac Data_mining_As2 %
```

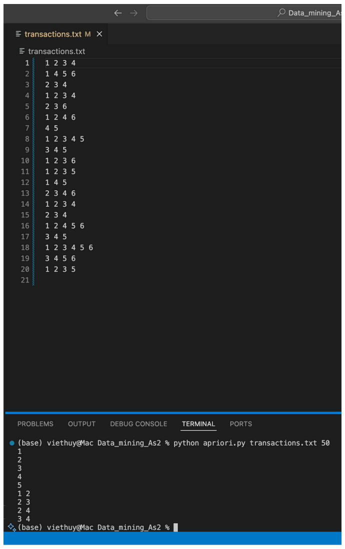
20%



```
1 2 3 4
1 4 5 6
2 3 4
1 2 3 4
2 3 6
1 2 4 6
4 5
1 2 3 4 5
3 4 5
1 2 3 6
1 2 3 5
1 4 5
2 3 4 6
1 2 3 4
2 3 4
1 2 4 5 6
3 4 5
1 2 3 4 5 6
3 4 5 6
1 2 3 5

```

```
(base) viethuy@Mac Data_mining_As2 % python apriori.py transactions.txt 30
1
2
3
4
5
6
1 2
1 3
1 4
1 5
2 3
2 4
2 6
3 4
3 5
4 5
4 6
1 2 3
1 2 4
2 3 4
(base) viethuy@Mac Data_mining_As2 %
```

30%



```
1 2 3 4
1 4 5 6
2 3 4
1 2 3 4
2 3 6
1 2 4 6
4 5
1 2 3 4 5
3 4 5
1 2 3 6
1 2 3 5
1 4 5
2 3 4 6
1 2 3 4
2 3 4
1 2 4 5 6
3 4 5
1 2 3 4 5 6
3 4 5 6
1 2 3 5

```

```
(base) viethuy@Mac Data_mining_As2 % python apriori.py transactions.txt 50
1
2
3
4
5
1 2
2 3
2 4
3 4
(base) viethuy@Mac Data_mining_As2 %
```

50%

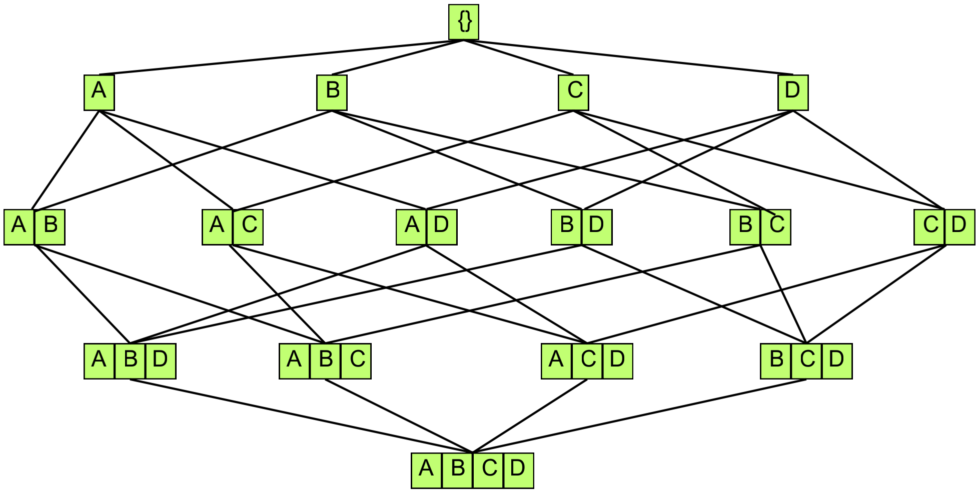
Note: I am sorry if the output is too small, i considered to trade off between the clearly output and the evident that i am actually running on my program. It is a pdf embedded, so please zoom in to see clearly. I am very appreciate *for that.*

2. Given 4 items, A, B, C, and D, list all possible itemsets in a lattice. (10 pts)

As i understand, lattice is the hierarchical representation of subsets of a set included the emty set. For set $\{A, B, C, D\}$ with 4 items, the lattice includes all possible subsets, which total $2^4 = 16$, include:

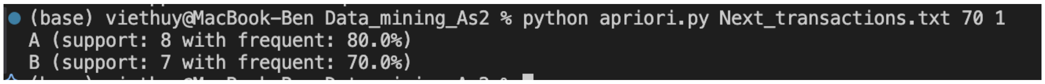
- Level 0 (size 0): $\{\emptyset\}$ (empty set)
- Level 1 (size 1): $\{A\}, \{B\}, \{C\}, \{D\}$
- Level 2 (size 2): $\{A, B\}, \{A, C\}, \{A, D\}, \{B, C\}, \{B, D\}, \{C, D\}$
- Level 3 (size 3): $\{A, B, C\}, \{A, B, D\}, \{A, C, D\}, \{B, C, D\}$
- Level 4 (size 4): $\{A, B, C, D\}$

And the chart bellow show the hierarchical of them:



3. In the following transaction database, if the minimum support is 7, please list all frequent itemsets and their support counts. (10 pts)

This is the output of Apriori with above transaction and minimum support is 7:

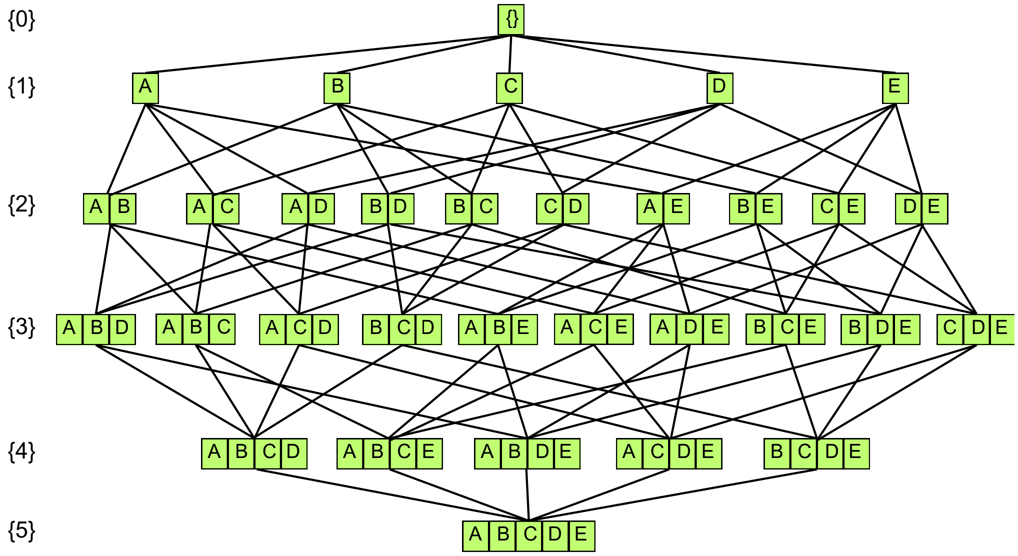


```
● (base) viethuy@MacBook-Ben Data_mining_As2 % python apriori.py Next_transactions.txt 70 1
  A (support: 8 with frequent: 80.0%)
  B (support: 7 with frequent: 70.0%)
```

Note: Because there are 10 transaction so i will use 70 stand for 70% ,parameter 1 at the end of terminal line is for show support count and frequent: 1 shows, None or other value is ignore the output)

4. In Question 3 above, please answer the following questions: (10 pts)
   (a) draw the lattice of all itemsets (with their support counts).
   (b) Given minimal support 5, list closed and maximal frequent itemset(s).

(a) draw the lattice of all itemsets (with their support counts). Because the image is too large, so i just represent the lattice with index level, and the support counts will be describe bellow:

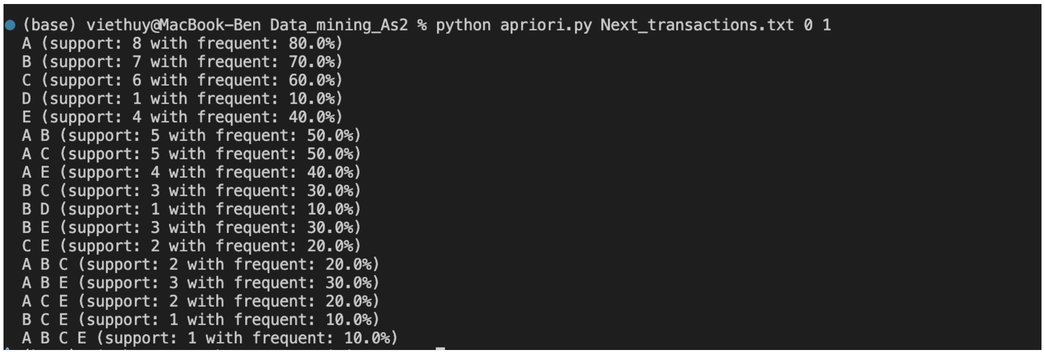


The support counts of each itemset:

- Level 0 (size 0): $\{\emptyset : 10\}$

- Level 1 (size 1): $\{A:8\},\{B:7\},\{C:6\},\{D:1\},\{E:4\}$
- Level 2 (size 2): $\{A,B:5\},\{A,C:5\},\{A,E:4\},\{B,C:3\},\{B,D:1\},\{B,E:3\},\{C,E:2\}$, $\{A,D:0\},\{C,D:0\},\{D,E:0\}$
- Level 3 (size 3): $\{A,B,C:2\},\{A,B,E:3\},\{A,C,E:2\},\{B,C,E:1\}$, $\{A,B,D:0\},\{A,C,D:0\},\{A,D,E:0\},\{B,C,D:0\},\{B,D,E:0\},\{C,D,E:0\}$
- Level 4 (size 4): $\{A,B,C,E:1\}$, $\{A,B,C,D:0\},\{A,B,D,E:0\},\{A,C,D,E:0\},\{B,C,D,E:0\}$
- Level 5 (size 5): $\{A,B,C,D,E:0\}$

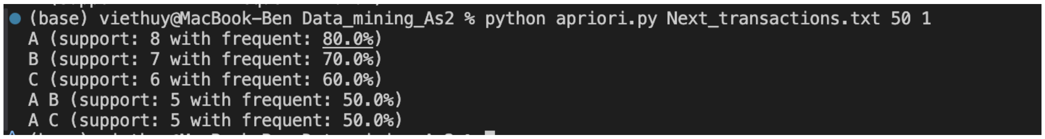
To have that information, i just run the apriori algorithm with minimum support is 0, and the output will show all the itemsets with their support counts.



```
● (base) viethuy@MacBook-Ben Data_mining_As2 % python apriori.py Next_transactions.txt 0 1
  A (support: 8 with frequent: 80.0%)
  B (support: 7 with frequent: 70.0%)
  C (support: 6 with frequent: 60.0%)
  D (support: 1 with frequent: 10.0%)
  E (support: 4 with frequent: 40.0%)
  A B (support: 5 with frequent: 50.0%)
  A C (support: 5 with frequent: 50.0%)
  A E (support: 4 with frequent: 40.0%)
  B C (support: 3 with frequent: 30.0%)
  B D (support: 1 with frequent: 10.0%)
  B E (support: 3 with frequent: 30.0%)
  C E (support: 2 with frequent: 20.0%)
  A B C (support: 2 with frequent: 20.0%)
  A B E (support: 3 with frequent: 30.0%)
  A C E (support: 2 with frequent: 20.0%)
  B C E (support: 1 with frequent: 10.0%)
  A B C E (support: 1 with frequent: 10.0%)
```

(b) Given minimal support 5, list closed and maximal frequent itemset(s).

With minimum support is 5, the frequent itemsets are:



```
● (base) viethuy@MacBook-Ben Data_mining_As2 % python apriori.py Next_transactions.txt 50 1
  A (support: 8 with frequent: 80.0%)
  B (support: 7 with frequent: 70.0%)
  C (support: 6 with frequent: 60.0%)
  A B (support: 5 with frequent: 50.0%)
  A C (support: 5 with frequent: 50.0%)
```

And the closed itemsets:

- $\{A : 8\}$
- $\{B : 7\}$
- $\{C : 6\}$
- $\{A, B : 5\}$
- $\{A, C : 5\}$

And the maximal itemsets:

- $\{A, B : 5\}$
- $\{A, C : 5\}$