

# Announcements

- Homework-1: out Tuesday, please start early
- Please don't post any material from this class in any form, including your homework and related materials, to any public places, such as GitHub or others, on the Internet !

# Finding Frequent Itemsets (Chapter 6)

Professor Wei-Min Shen

University of Southern California

Thanks for source slides and material to:  
J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets  
<http://www.mmds.org>

# Frequent Itemsets and Association Rules

- Family of techniques for characterizing data: **discovery of frequent itemsets**
  - e.g., identify sets of items that are frequently purchased together

## Outline:

- Introduce market-basket model of data
- Define frequent itemsets
- Discover association rules
  - Confidence and interest of rules
- A-Priori Algorithm and variations

# THE MARKET-BASKET MODEL

# Association Rule Discovery

## Supermarket shelf management – Market-basket model:

- **Goal:** Identify items that are bought together by sufficiently many customers
- **Approach:** Process the sales data to find dependencies among items
  - Brick and mortar stores: data collected with barcode scanners
  - Online retailers: transaction records for sales
- **A classic rule:**
  - If someone buys diaper and milk, then he/she is likely to buy beer.      // really ☺    do you know why?
  - Don't be surprised if you find six-packs next to diapers!

# The Market-Basket Model

- A large set of **items**
  - e.g., things sold in a supermarket
- A large set of **baskets**
- Each basket is a **small subset of items**
  - e.g., the things one customer buys on one day
- **Want to discover Association Rules**
  - People who bought  $\{x,y,z\}$  tend to buy  $\{v,w\}$ 
    - **Brick and mortar stores:** Influences setting of prices, what to put on sale when, product placement on store shelves
    - **Recommender systems:** Amazon, Netflix, etc.

## Input:

<i>TID</i>	<i>Items</i>
1	Bread, Coke, Milk
2	Beer, Bread
3	Beer, Coke, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Coke, Diaper, Milk

## Output:

### Rules Discovered:

$\{\text{Milk}\} \rightarrow \{\text{Coke}\}$

$\{\text{Diaper, Milk}\} \rightarrow \{\text{Beer}\}$

# Market-Baskets

- Really a **general many-many mapping** (association) between two kinds of things: items and baskets
  - But we ask about connections among “items,” not “baskets.”
- The technology focuses on **common events**, **not rare events**
  - Don't need to focus on identifying **\*all\*** association rules
  - Want to focus on **common events**, focus pricing strategies or product recommendations on those items or association rules

# Market Basket Applications (1):

## Identify items bought together

- **Items** = products
- **Baskets** = sets of products someone bought in one trip to the store
- **Real market baskets:** Stores (Walmart, Target, Ralphs, etc.) keep terabytes of data about what items customers buy together
  - Tells how typical customers navigate stores
  - Lets them position tempting items
  - Suggests tie-in “tricks”, e.g., run sale on diapers and raise the price of beer
  - **Need the rule to occur frequently, or no profits!**
- **Amazon’s people who bought X also bought Y**
  - Recommendation Systems



# Market Basket Applications (2): Plagiarism detection

- **Baskets**

- = Sentences?

- = Documents containing those sentences?

- **Items**

- = Sentences?

- = Documents containing those sentences?

- Question: Baskets=?, Items=?

# Market Basket Applications (2): Plagiarism detection

- **Baskets** = sentences
- **Items** = documents containing those sentences
  - Item/document is “in” a basket if sentence is in the document
  - May seem backward, but relationship between baskets and items is many-to-many
- Look for items that appear together in several baskets
  - Multiple documents share sentence(s)
- **Items (documents) that appear together too often could represent plagiarism.**
- **Please don't copy anyone's code for your homework!**
  - It will be detected easily by our Agent!

## Market Basket Applications (3): Identify related “concepts” in web documents

- **Baskets** = words? Web pages?
- **Items** = words? Web pages?

# Market Basket Applications (3):

## Identify related “concepts” in web documents

- **Baskets** = Web pages
- **Items** = words
- Baskets/documents contain items/words in the document
- Look for sets of words (items) that appear together in many documents (baskets)
- Ignore most common words
- Unusual words appearing together in a large number of documents, e.g., “World” and “Cup,” may indicate an interesting relationship or joint concept
  - Can you think of such examples: Word-X, Word-Y ?

## Market Basket Applications (4): Drug Interactions

- **Baskets** = patients
- **Items** = drugs and side effects
- Has been used to **detect combinations of drugs that result in particular side-effects**
- **But requires extension:** Absence of an item needs to be observed as well as presence!!
  - Drinking milk and oil together: BAD
  - Drinking milk alone: OK
  - Drinking oil alone: OK

## Scale of the Problem

- WalMart sells 100,000 items and can store billions of baskets.
- The Web has billions of words and many billions of pages.

**DEFINE FREQUENT ITEMSETS**

## “Support” and “Frequent Itemsets”

- Simplest question: Find sets of items that appear “frequently” in the baskets
- Support for itemset  $I$  = the number of baskets containing all items in  $I$ 
  - Sometimes given as a percentage
- Given a support threshold  $s$ , sets of items that appear in at least  $s$  baskets are called “*Frequent Itemsets*”



## Example: Frequent Itemsets

- Items = {milk, coke, pepsi, beer, juice}.
- Support = 3 baskets.**

$$B_1 = \{m, c, b\}$$

$$B_2 = \{m, p, j\}$$

$$B_3 = \{m, b\}$$

$$B_4 = \{c, j\}$$

$$B_5 = \{m, p, b\}$$

$$B_6 = \{m, c, b, j\}$$

$$B_7 = \{c, b, j\}$$

$$B_8 = \{b, c\}$$

- Frequent itemsets of size 1: {m}, {c}, {b}, {j}

{m,b}, {b,c}, {c,j}.

# ASSOCIATION RULES

## “Association Rules” and “Confidence”

- If-then rules about the contents of baskets
- Basket  $I$  contains  $\{i_1, i_2, \dots, i_k\}$
- Rule  $\{i_1, i_2, \dots, i_k\} \rightarrow j$  means: “if a basket contains all of  $i_1, \dots, i_k$  then it is *likely* to contain  $j$ .”
- **Confidence** of this association rule is  
the probability of  $j$  given  $i_1, \dots, i_k$ 
  - $$\frac{\text{Ratio of support for } I \cup \{j\} \text{ with support for } I}{\text{support for } I}$$
  - Support for  $I$ : number of baskets containing  $I$

## Example: Confidence of a Rule

$$B_1 = \{m, c, b\}$$

$$B_2 = \{m, p, j\}$$

$$B_3 = \{m, b\}$$

$$B_4 = \{c, j\}$$

$$B_5 = \{m, p, b\}$$

$$B_6 = \{m, c, b, j\}$$

$$B_7 = \{c, b, j\}$$

$$B_8 = \{b, c\}$$

- An association rule:  $\{m, b\} \rightarrow c$ 
  - Confidence: Ratio of support for  $I \cup \{j\}$  with support for  $I$
  - Ratio of support for  $\{m, b\} \cup \{c\}$  to support for  $\{m, b\}$
  - Confidence =  $2/4 = 50\%$
- Want to identify association rules with high confidence

## Interesting Association Rules

- **Not all high-confidence rules are interesting**
  - The rule  $X \rightarrow \textit{milk}$  may have high confidence for many itemsets  $X$ 
    - because milk is just purchased very often (independent of  $X$ )
- **Interest of an association rule  $I \rightarrow j$ : difference between its confidence and the fraction of baskets that contain  $j$**

$$\text{Interest}(I \rightarrow j) = \text{conf}(I \rightarrow j) - \text{Pr}[j]$$

- **Interesting rules are those with high positive or negative interest values (usually above 0.5)**
- High **positive/negative** interest means presence of  $I$  **encourages** or **discourages** presence of  $j$
- Example: {coke}  $\rightarrow$  pepsi should have high negative interest

## Example: Confidence and Interest

$$\begin{array}{ll} B_1 = \{m, c, b\} & B_2 = \{m, p, j\} \\ B_3 = \{m, b\} & B_4 = \{c, j\} \\ B_5 = \{m, p, b\} & B_6 = \{m, c, b, j\} \\ B_7 = \{c, b, j\} & B_8 = \{b, c\} \end{array}$$

- **Association rule:  $\{m, b\} \rightarrow c$** 
  - Confidence: Ratio of support for  $I \cup \{j\}$  with support for  $I$
  - **Confidence** =  $2/4 = 0.5$
  - Interest:  $\text{Interest}(I \rightarrow j) = \text{conf}(I \rightarrow j) - \text{Pr}[j]$
  - **Difference between its confidence and the fraction of baskets that contain  $j$**
  - **Interest** =  $|0.5 - 5/8| = 1/8$ 
    - Item  $c$  appears in  $5/8$  of the baskets
    - Rule is not very interesting!

# Finding Useful Association Rules

- **Question:** “find all association rules with support  $\geq s$  and confidence  $\geq c$ ”
- **Hard part:** finding the frequent itemsets
  - **Note:** if  $\{i_1, i_2, \dots, i_k\} \rightarrow j$  has high support and confidence, then both  $\{i_1, i_2, \dots, i_k\}$  and  $\{i_1, i_2, \dots, i_k, j\}$  will be “frequent”
- **Assume:** not too many frequent itemsets or candidates for high support, high confidence association rules
  - Not so many that they can't be acted upon
  - Adjust support threshold to avoid too many frequent itemsets

## Example: Find Association Rules with support $\geq s$ and confidence $\geq c$

$$\begin{array}{ll} B_1 = \{m, c, b\} & B_2 = \{m, p, j\} \\ B_3 = \{m, c, b, n\} & B_4 = \{c, j\} \\ B_5 = \{m, p, b\} & B_6 = \{m, c, b, j\} \\ B_7 = \{c, b, j\} & B_8 = \{b, c\} \end{array}$$

- Support threshold  $s = 3$ , confidence  $c = 0.75$
- 1) Frequent itemsets:
  - $\{b\} \{c\} \{j\} \{m\} \{b,m\} \{b,c\} \{c,m\} \{c,j\} \{m,c,b\}$
- 2) Generate rules:
  - $b \rightarrow m: \text{conf}=4/6$        $b \rightarrow c: \text{conf}=5/6$        $b,c \rightarrow m: \text{conf}=3/6$
  - $m \rightarrow b: \text{conf}=4/5$       ...       $b,m \rightarrow c: \text{conf}=3/4$
  - $b \rightarrow c,m: \text{conf}=3/6$

**Difficult part is identifying frequent itemsets: algorithms to find them are the focus of this chapter**

$$\text{conf}(I \rightarrow j) = \frac{\text{support}(I \cup j)}{\text{support}(I)}$$

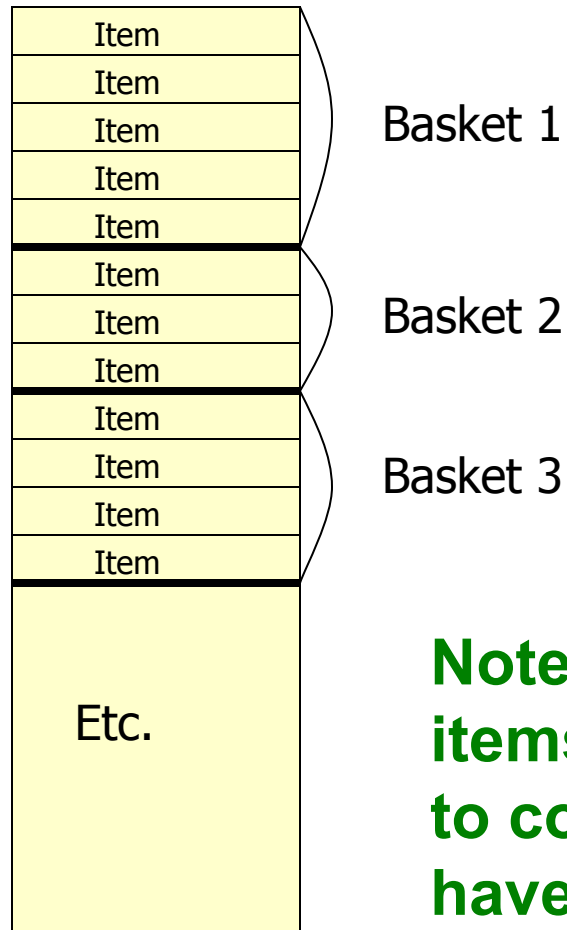


# **FIND FREQUENT ITEMSETS**

# Computation Model

- Typically, market basket data are kept in **flat files** rather than in a database system
  - Stored **on disk because they are very large files**
  - Stored **basket-by-basket**
  - **Goal: Expand baskets into pairs, triples, etc. as you read baskets**
    - Use  $k$  nested loops to generate all sets of size  $k$

# File Organization



**Example:** items are positive integers, and boundaries between baskets are  $-1$

**Note:** We want to find frequent itemsets. To find them, we have to count them. To count them, we have to generate them.

## Computation Model – (2)

- The true cost of mining disk-resident data is usually the **number of disk I/O's**
- In practice, association-rule algorithms read the data in *passes* – all baskets read in turn
- Thus, we measure the cost by the **number of passes** an algorithm takes

# Main-Memory Bottleneck

- **For many frequent-itemset algorithms, main memory is the critical resource**
  - As we read baskets, **we need to count something, e.g., occurrences of pairs**
  - **The number of different things we can count is limited by main memory**
  - Swapping counts in/out is a disaster
  - **Algorithms are designed so that counts can fit into main memory**

## Finding Frequent Pairs

- The hardest problem often turns out to be finding the frequent pairs
  - Why? Often frequent pairs are common, frequent triples are rare
    - Why? Probability of being frequent drops exponentially with size; number of sets grows more slowly with size
- We'll concentrate on pairs, then extend to larger itemsets

a
b
c
y
a
x
y
b
z
b
x
c
z
c
x
b
y

## Baskets

### Naïve Algorithm

- **Read file once, counting in main memory the occurrences of each pair**

- Number of pairs in a basket of  $n$  items:  $n$  choose 2

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

- From each basket of  $n$  items, generate its  $n*(n-1)/2$  pairs using **two nested loops**, add to the count for each pair

- First basket: (a,b), (a,c), (a,y), (b,c), (b,y), (c,y)

- Second basket: (a,b), (a,x), (a,y), (a,z), (b,x), (b,y), (b,z), ...

- Total possible number of pairs in all baskets:

$$(\#items)(\#items - 1)/2$$

- **Fails if  $(\#items)^2$  exceeds main memory**

- **Remember:** #items can be 100K (Wal-Mart) or 10B (Web pages)

## Example: Counting Pairs

- Suppose  $10^5$  items
- Suppose counts are 4-byte integers
- Number of pairs of items:  $10^5(10^5-1)/2 = 5*10^9$   
(approximately)
- Therefore,  $2*10^{10}$  (20 gigabytes) of main memory needed



## Details of Main-Memory Counting

- **Two approaches:**

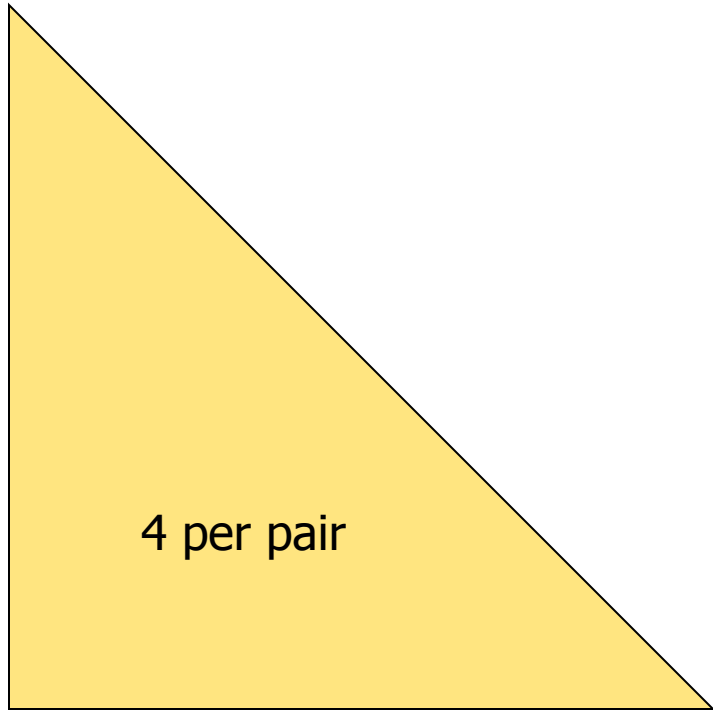
1. Count all pairs, using a **triangular matrix**
2. Keep a **table of triples**  $[i, j, c]$  = “the count of the pair of items  $\{i, j\}$  is  $c$ ”

(1) requires only 4 bytes/pair, but requires a count for each pair

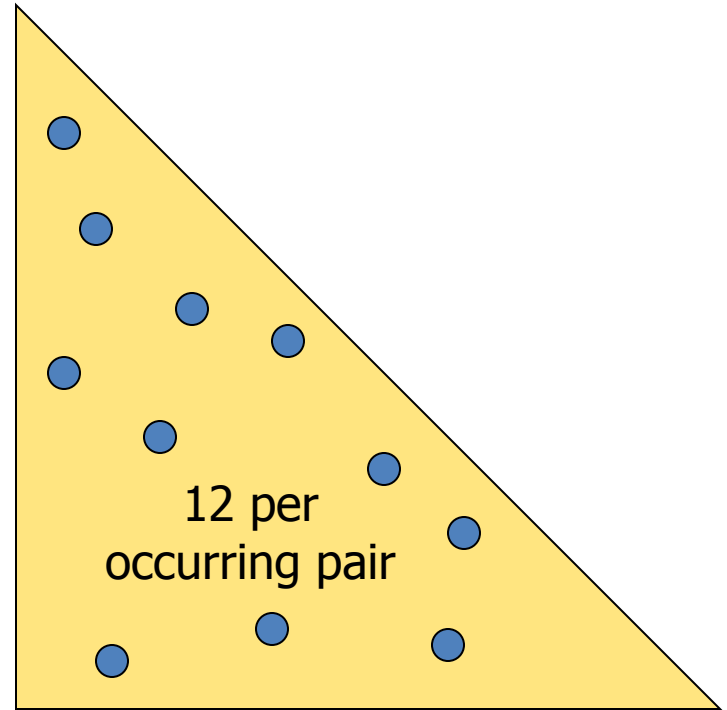
**Note:** assume integers are 4 bytes

(2) requires 12 bytes, but only for those pairs with count > 0

Plus some additional overhead for a hashtable

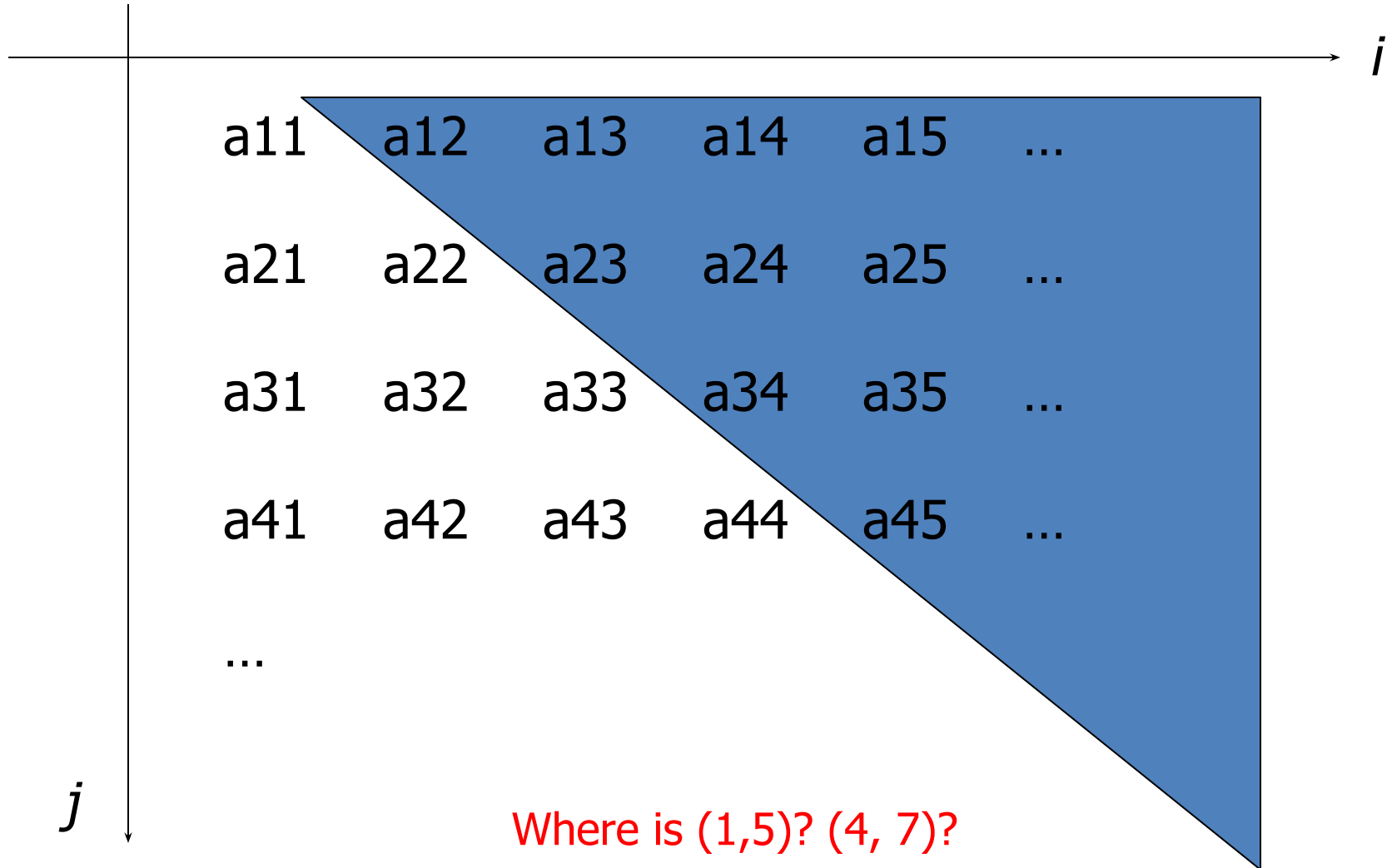


Method (1):  
It is a long list of "c"



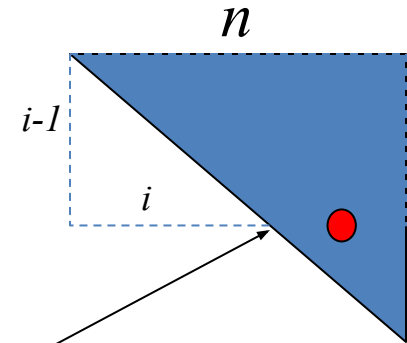
Method (2)  
It is a long list of  
● = (i,j,c)

## Triangular Matrix: (i,j) is index, c is count



# Triangular-Matrix Approach – (1)

- $n$  = total number of items
- Order each pair of items  $\{i, j\}$  so that  $i < j$
- Keep pair counts in lexicographic order:
  - $\{1,2\}, \{1,3\}, \dots, \{1,n\}, \{2,3\}, \{2,4\}, \dots, \{2,n\}, \{3,4\}, \dots$
- Pair  $\{i, j\}$  is at position  $(i-1)(n-i/2) + j - i$ 
  - *Every time you see a pair  $\{i,j\}$  from a basket, increment the count at the corresponding position in triangular matrix*
- Total number of pairs  $n(n-1)/2$ ; total bytes =  $2n^2$
- **Triangular Matrix** requires 4 bytes (1 integer) per pair



# Comparing the two approaches

- **Approach 1: Triangular Matrix**

- $n$  = total number items
- Count pair of items  $\{i, j\}$  only if  $i < j$
- Keep pair counts in lexicographic order:
  - $\{1,2\}, \{1,3\}, \dots, \{1,n\}, \{2,3\}, \{2,4\}, \dots, \{2,n\}, \{3,4\}, \dots$
- Pair  $\{i, j\}$  is at position  $(i-1)(n-i/2) + j-i$
- Total number of pairs  $n(n-1)/2$ ; total bytes =  $2n^2$
- **Triangular Matrix** requires 4 bytes (1 integer for c) per pair

- **Approach 2:** uses **12 bytes** (i, j, c) per occurring pair  
(but only for pairs with count > 0)

- **Beats Approach 1 if fewer than 1/3 of possible pairs actually occur in the market basket data**

## Comparing the two approaches

- **Approach 1: Triangular Matrix**

- $n$  = total number items

- Count pairs  $(i, j)$  such that  $i < j$

- E

- E

- T

- T

- **Approach 2: Hash Table**

- (but)

- E

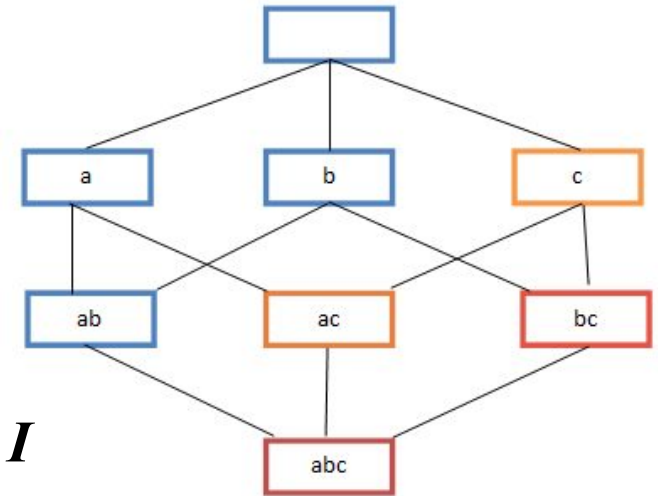
- possible pairs actually occur

**Problem is if we have  
too many items so the  
pairs  
do not fit into memory.  
Can we do better?**

# A-Priori Algorithm

## A-Priori Algorithm – (1)

- A **two-pass** approach called *A-Priori* limits the need for main memory
- **Key idea: *monotonicity***
  - If a set of items  $I$  appears at least  $s$  times, so does every **subset**  $J$  of  $I$
- **Contrapositive for pairs:**  
If item  $i$  does not appear in  $s$  baskets, then no pair including  $i$  can appear in  $s$  baskets
- **So, how does A-Priori find freq. pairs?**





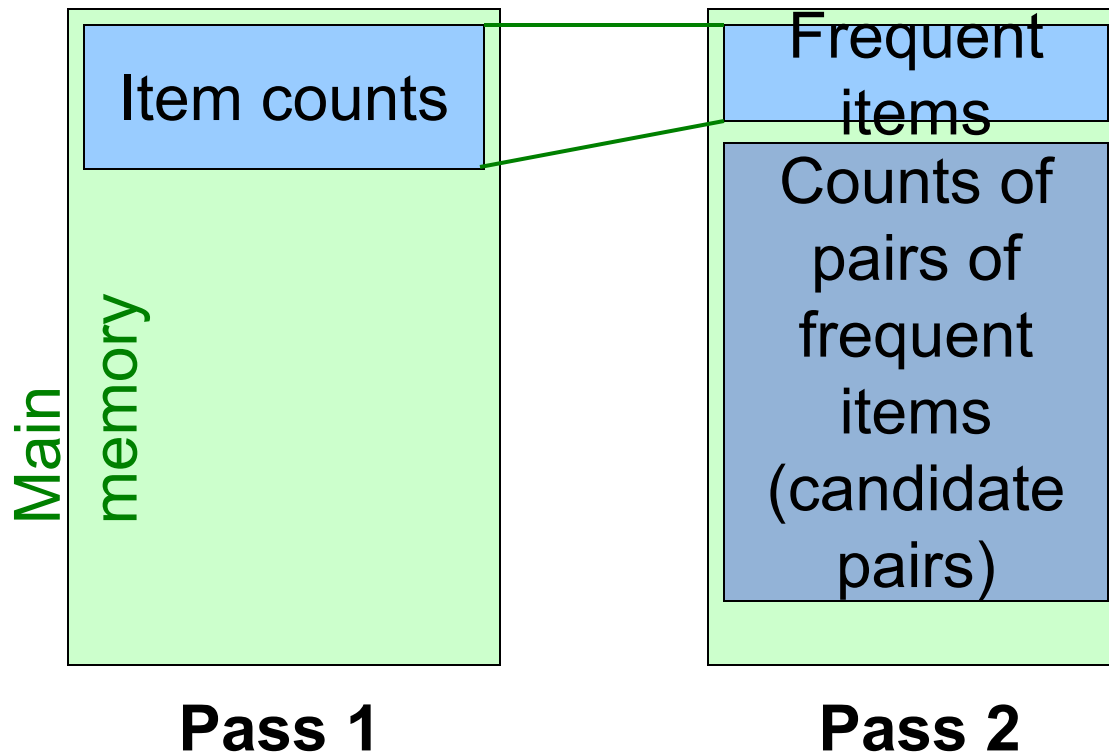
# A-Priori Algorithm

- **Pass 1: Read baskets and count in main memory the occurrences of each single item**
  - Requires only memory proportional to #items
- **Items that appear at least  $s$  times are the *frequent items***
  - At the end of pass 1, after the complete input file has been processed, check the count for each item
  - If  $\text{count} > s$ , then that item is frequent: saved for the next pass
- **Pass 1 identifies frequent itemsets (support  $> s$ ) of size 1**

# A-Priori Algorithm

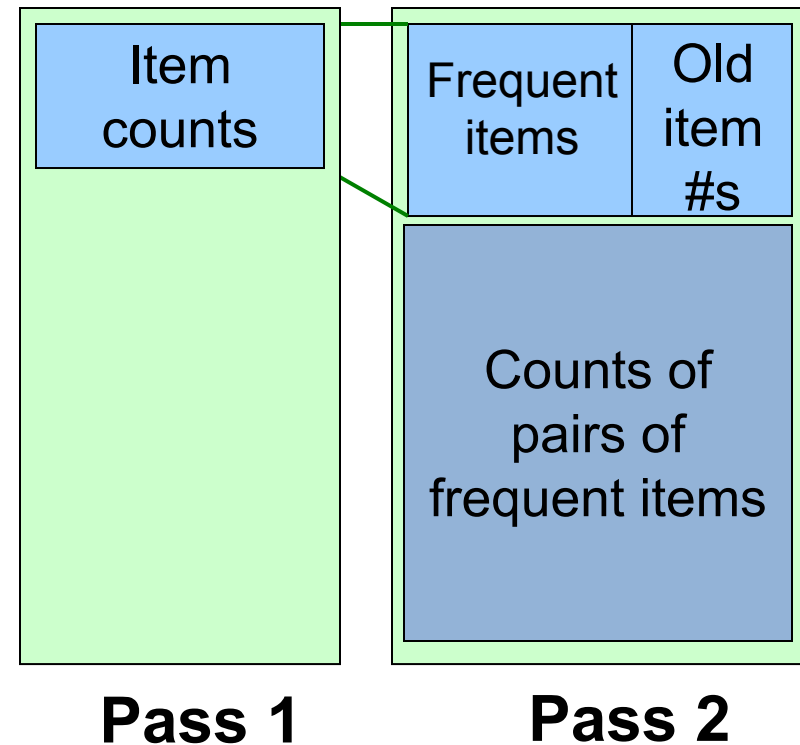
- **Pass 2: Read baskets again and count in main memory only those pairs of items where both were found in Pass 1 to be frequent**
- **Requires:**
  - **Memory proportional to square of *frequent* items only** (to hold counts of pairs)
  - **List of the frequent items from the first pass** (so you know what must be counted)
- **Pairs of items that appear at least  $s$  times are the *frequent pairs of size 2***
  - **At the end of pass 2, check the count for each pair**
  - **If  $\text{count} > s$ , then that pair is frequent**
- **Pass 2 identifies frequent pairs: itemsets of size 2**

# Main-Memory: Picture of A-Priori



## Detail for A-Priori

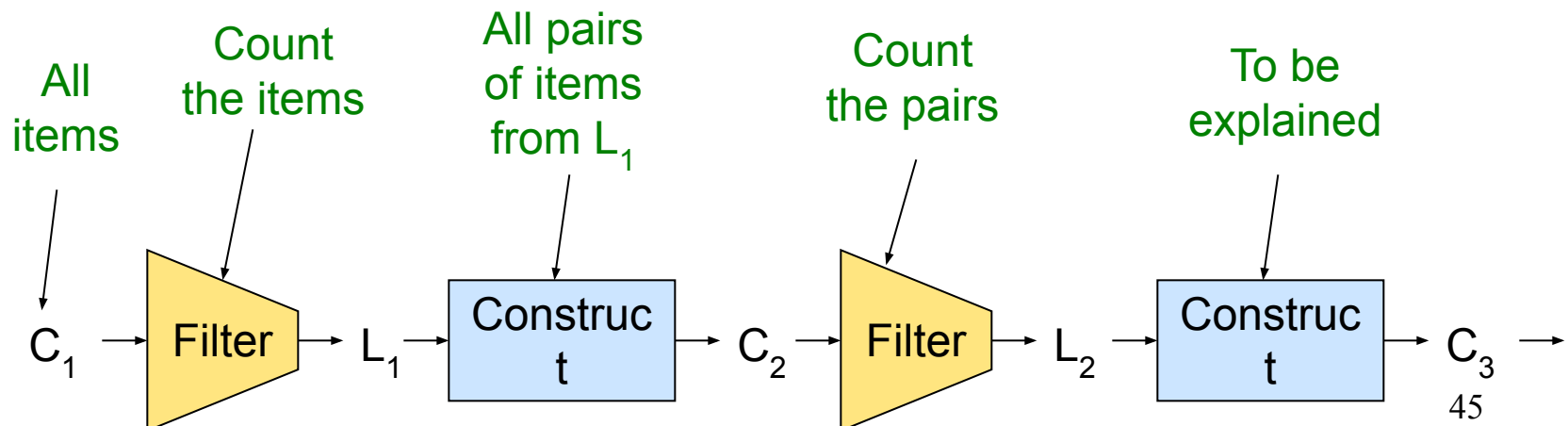
- You can use the triangular matrix method with  $n$  = number of frequent items
  - May save space compared with storing triples
- **Trick:** re-number frequent items 1,2,... and keep a table relating new numbers to original item numbers



# What About Larger Frequent Itemsets?

## Frequent Triples, Etc.

- For each  $k$ , we construct two sets of  $k$ -tuples (sets of size  $k$ ):
  - $C_k$  = *candidate  $k$ -tuples* = those that *might be frequent* sets (support  $\geq s$ ) based on information from the pass for  $k-1$
  - $L_k$  = the set of *truly frequent  $k$ -tuples*



## Recall: Example

$$B_1 = \{m, c, b\}$$

$$B_2 = \{m, p, j\}$$

$$B_3 = \{m, c, b, n\}$$

$$B_4 = \{c, j\}$$

$$B_5 = \{m, p, b\}$$

$$B_6 = \{m, c, b, j\}$$

$$B_7 = \{c, b, j\}$$

$$B_8 = \{b, c\}$$

- **Frequent itemsets (s=3):**

- $\{b\}, \{c\}, \{j\}, \{m\}$

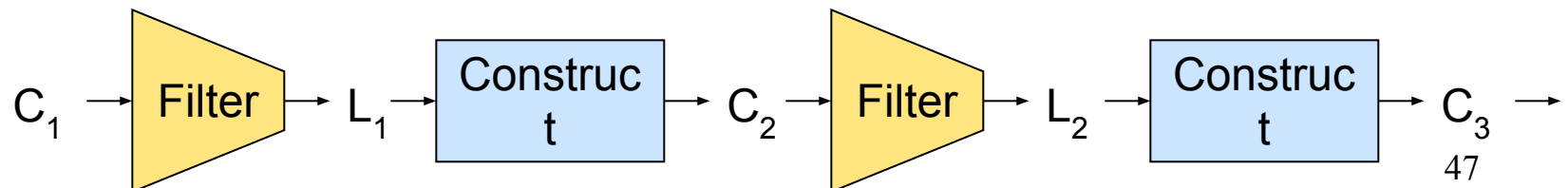
- $\{b,m\} \quad \{b,c\} \quad \{c,m\} \quad \{c,j\}$

- $\{m,c,b\}$

# Example

- **Hypothetical steps of the A-Priori algorithm**

- $C_1 = \{ \{b\} \{c\} \{j\} \{m\} \{n\} \{p\} \}$ : all candidate items
- Count the support of itemsets in  $C_1$
- Prune non-frequent:  $L_1 = \{ b, c, j, m \}$
- Generate  $C_2 = \{ \{b,c\} \{b,j\} \{b,m\} \{c,j\} \{c,m\} \{j,m\} \}$
- Count the support of itemsets in  $C_2$
- Prune non-frequent:  $L_2 = \{ \{b,m\} \{b,c\} \{c,m\} \{c,j\} \}$
- Generate  $C_3 = \{ \{b,c,m\} \}$
- Count the support of itemsets in  $C_3$
- Prune non-frequent:  $L_3 = \{ \{b,c,m\} \}$



## A-Priori for All Frequent Itemsets

- **One pass for each  $k$  (itemset size)**
- Needs room in main memory to count each candidate  $k$ -tuple
- For typical market-basket data and reasonable support (e.g., 1%),  $k = 2$  requires the most memory