COSC 3337 : Data Science I

N. Rizk

College of Natural and Applied Sciences

Department of Computer Science

University of Houston

Fast Algorithm for Mining Association Rule

`Basket data'

Association Analysis

- Consider shopping cart filled with several items
- Market basket analysis tries to answer the following questions:
  - Who makes purchases?
  - What do customers buy together?
  - In what order do customers purchase items?
  - When do customers purchase the most and what?

Market Basket Analysis (Contd.)

1. Coocurrences
   - 80% of all customers purchase items X, Y and Z together.
2. Association rules
   - 60% of all customers who purchase X and Y also buy Z.
3. Sequential patterns
   - 60% of customers who first buy X also purchase Y within three weeks.

Definitions !

Confidence and Support

We prune the set of all possible association rules using two interestingness measures:

- Support of a rule:
  - X → Y has support s if P(XY) = s
  - Represents percentage of the transactions that contain all these items
- Confidence of a rule:
  - X → Y has confidence c if P(sup(LHS U RHS) | sup (LHS)) = c
  - Confidence for a rule X → Y is the percentage of such transactions that also contain all items in Y

We can also define

- Support of an itemset (a coocurrence) XY:
  - XY has support s if P(XY) = s

$$Rule:\ X \Rightarrow Y$$

$$Support = \frac{frq(X,Y)}{N}$$

$$Confidence = \frac{frq(X,Y)}{frq(X)}$$

$$Lift = \frac{Support}{Supp(X) \times Supp(Y)}$$

Discovering Rules

A common and useful application of data mining

A `rule' is something like this:

If a basket contains apples and cheese, then it also contains beer

Any such rule has two associated measures:

1. *confidence* – when the `if' part is true, how often is the `then' bit true? This is the same as *accuracy*.
2. *coverage* or *support* – how much of the database contains the `if' part?

supp(x)= number of transactions in which x appears/total number of transactions
conf(x->y)= supp(xU y)/ supp(x) confidence
Lift(x->y)= supp(xU y)/ supp(x)*sup(y)
Conv(x->y) =1-supp(y)/(1- conf(x->y)) conviction

Support threshold
(when x starts affecting the result)

Confidence threshold

$$\text{Rule: } X \Rightarrow Y$$

$$Support = \frac{frq(X,Y)}{N}$$

$$Confidence = \frac{frq(X,Y)}{frq(X)}$$

$$Lift = \frac{Support}{Supp(X) \times Supp(Y)}$$

Example:

Rule	Support	Confidence	Lift
$A \Rightarrow D$	2/5	2/3	10/9
$C \Rightarrow A$	2/5	2/4	5/6
$A \Rightarrow C$	2/5	2/3	5/6
$B \& C \Rightarrow D$	1/5	1/3	5/9

Frequent Itemsets

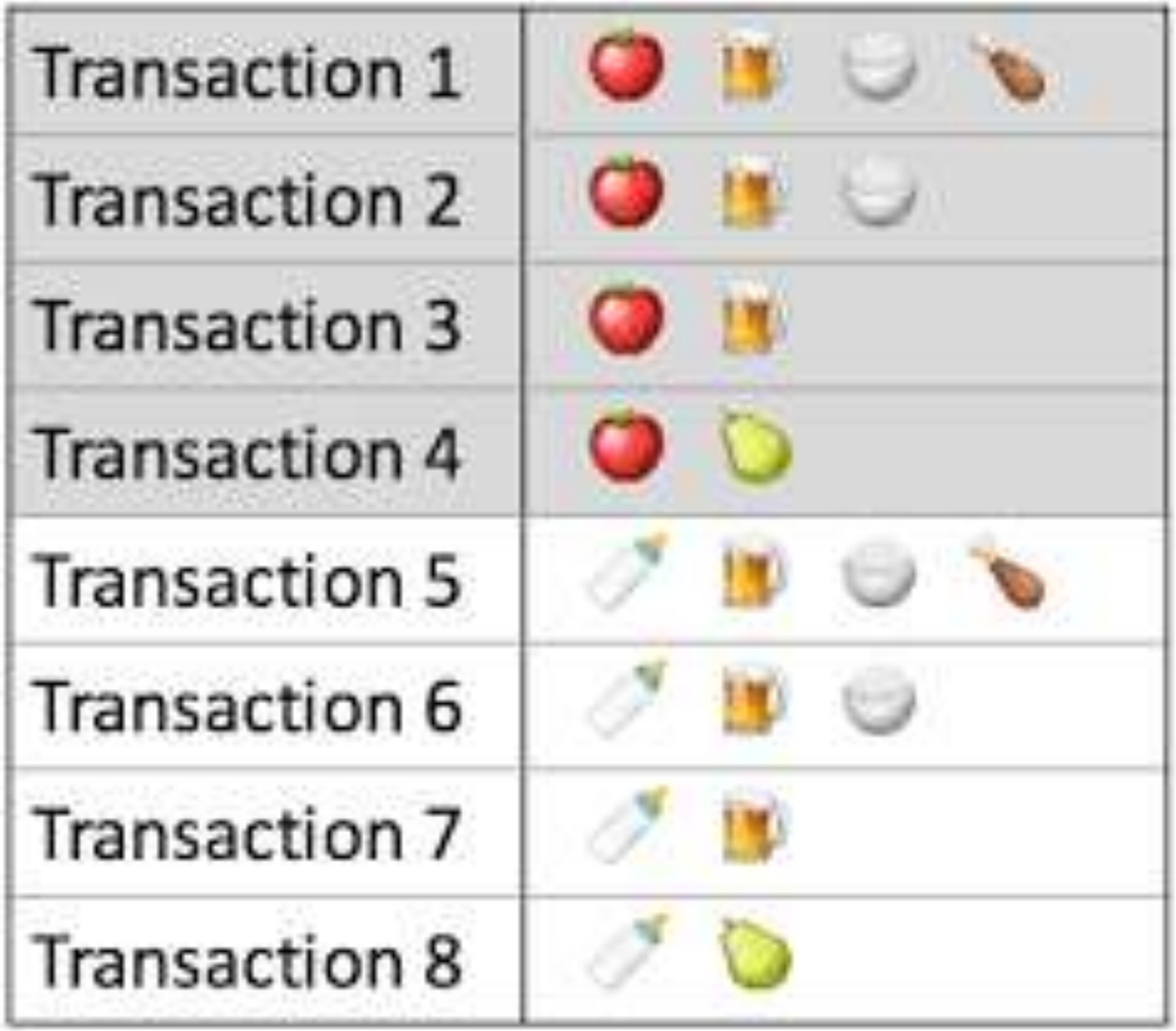
Applications

- Items = products; baskets = sets of products someone bought in one trip to the store.
- Example application: given that many people buy beer and diapers together:
  - Run a sale on diapers; raise price of beer.
- Baskets = Web pages; items = words.
- Example application: Unusual words appearing together in a large number of documents, e.g., “Brad” and “Angelina,” may indicate an interesting relationship.
- Baskets = sentences; items = documents containing those sentences.
- Example application: Items that appear together too often could represent plagiarism.

Market-Basket Data

- A large set of items, e.g., things sold in a supermarket.
- A large set of baskets, each of which is a small set of the items, e.g., the things one customer buys on one day.

Connections among “items,” not “baskets.”


Transaction 1
Transaction 2
Transaction 3
Transaction 4
Transaction 5
Transaction 6
Transaction 7
Transaction 8

Support

Support {🍎} = $\frac{4}{8}$

Transaction 1	🍎 🍺 🍚 🍗
Transaction 2	🍎 🍺 🍚
Transaction 3	🍎 🍺
Transaction 4	🍎 🍐
Transaction 5	🍼 🍺 🍚 🍗
Transaction 6	🍼 🍺 🍚
Transaction 7	🍼 🍺
Transaction 8	🍼 🍐

Confidence

How likely item Y is purchased when item X is purchased, expressed as {X -> Y}

$$\text{Confidence}\ \{\text{🍎} \rightarrow \text{🍺}\} = \frac{\text{Support}\ \{\text{🍎}, \text{🍺}\}}{\text{Support}\ \{\text{🍎}\}}$$

Lift

$$\text{Lift } \{🍎 \rightarrow 🍺\} = \frac{\text{Support } \{🍎, 🍺\}}{\text{Support } \{🍎\} \times \text{Support } \{🍺\}}$$

How likely item Y is purchased when item X is purchased, while controlling for how popular item Y is.

If the lift of {apple -> beer} is 1➔no association
If lift value >1 ➔that item Y is *likely* to be bought if item X is bought.
If lift value <1➔ that item Y is *unlikely* to be bought if item X is bought

Frequent Itemsets

- Given a set of transactions, find combinations of items (itemsets) that occur frequently

Support s I : number of transactions that contain

- **k-itemset**
  - An itemset that contains k items

Market-Basket transactions **Items**: {Bread, Milk, Diaper, Beer, Eggs, Coke}

TID	*Items*
1	**Bread, Milk**
2	**Bread, Diaper, Beer, Eggs**
3	**Milk, Diaper, Beer, Coke**
4	**Bread, Milk, Diaper, Beer**
5	**Bread, Milk, Diaper, Coke**

σ({Milk, Bread,Diaper}) = 2

s({Milk, Bread, Diaper}) = 40%

Examples of frequent itemsets $s(I)$ ≥ 3
itemset I

{Bread}: 4
{Milk} : 4 **1-itemset**
{Diaper} : 4
{Beer}: 3
{Diaper, Beer} : 3 **2-itemset**
{Milk, Bread} : 3

Lift

In data **mining** and **association rule** learning, **lift** is a measure of the performance of a targeting model (**association rule**) at predicting or classifying cases as having an enhanced response (with respect to the population as a whole), measured against a random choice targeting model.

	Coffee	$\overline{\text{Coffee}}$	
Tea	15	5	20
$\overline{\text{Tea}}$	75	5	80
	90	10	100

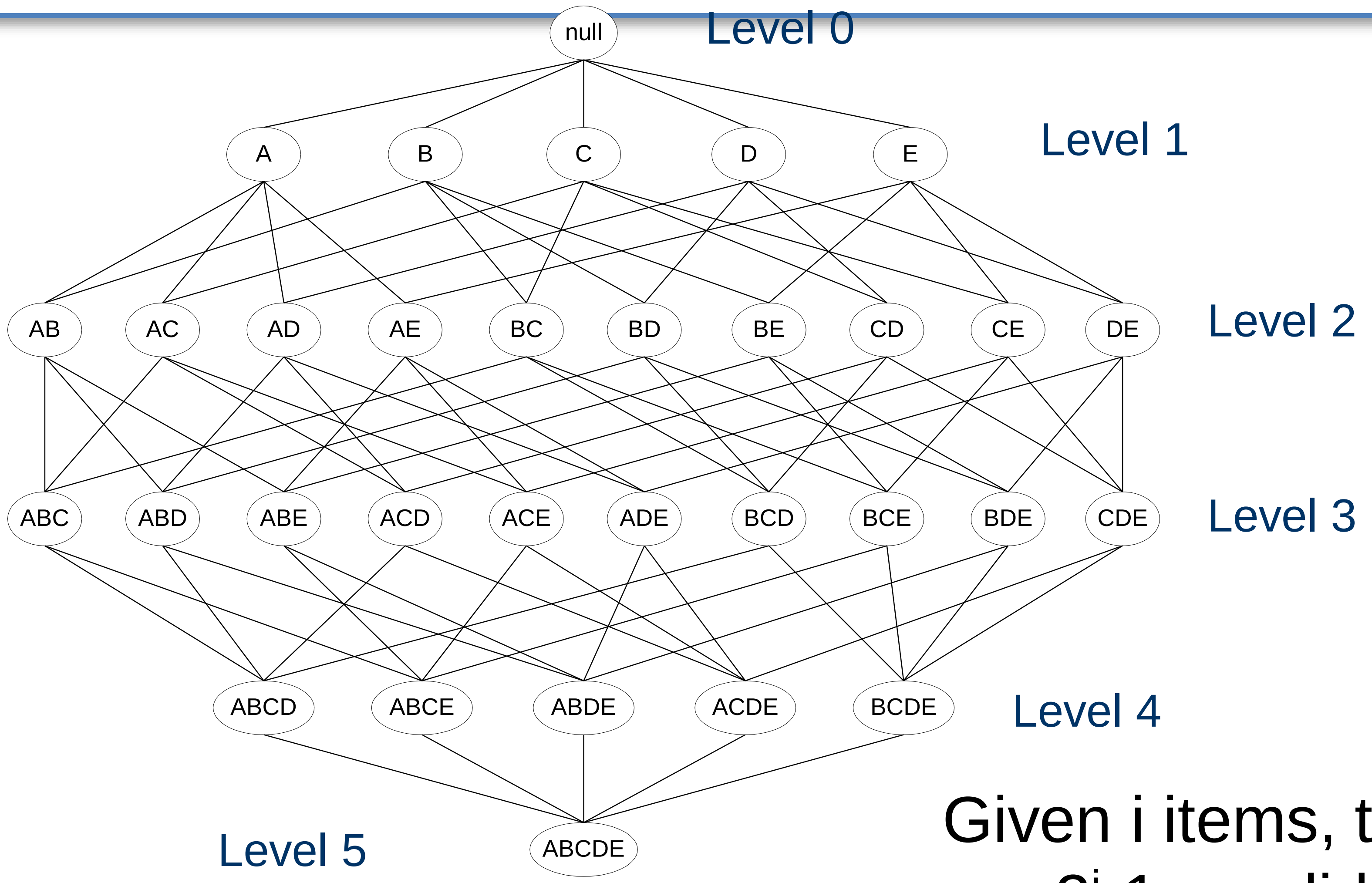
Association Rule: Tea $\rightarrow$ Coffee

Confidence= P(Coffee|Tea) = 0.75

but P(Coffee) = 0.9

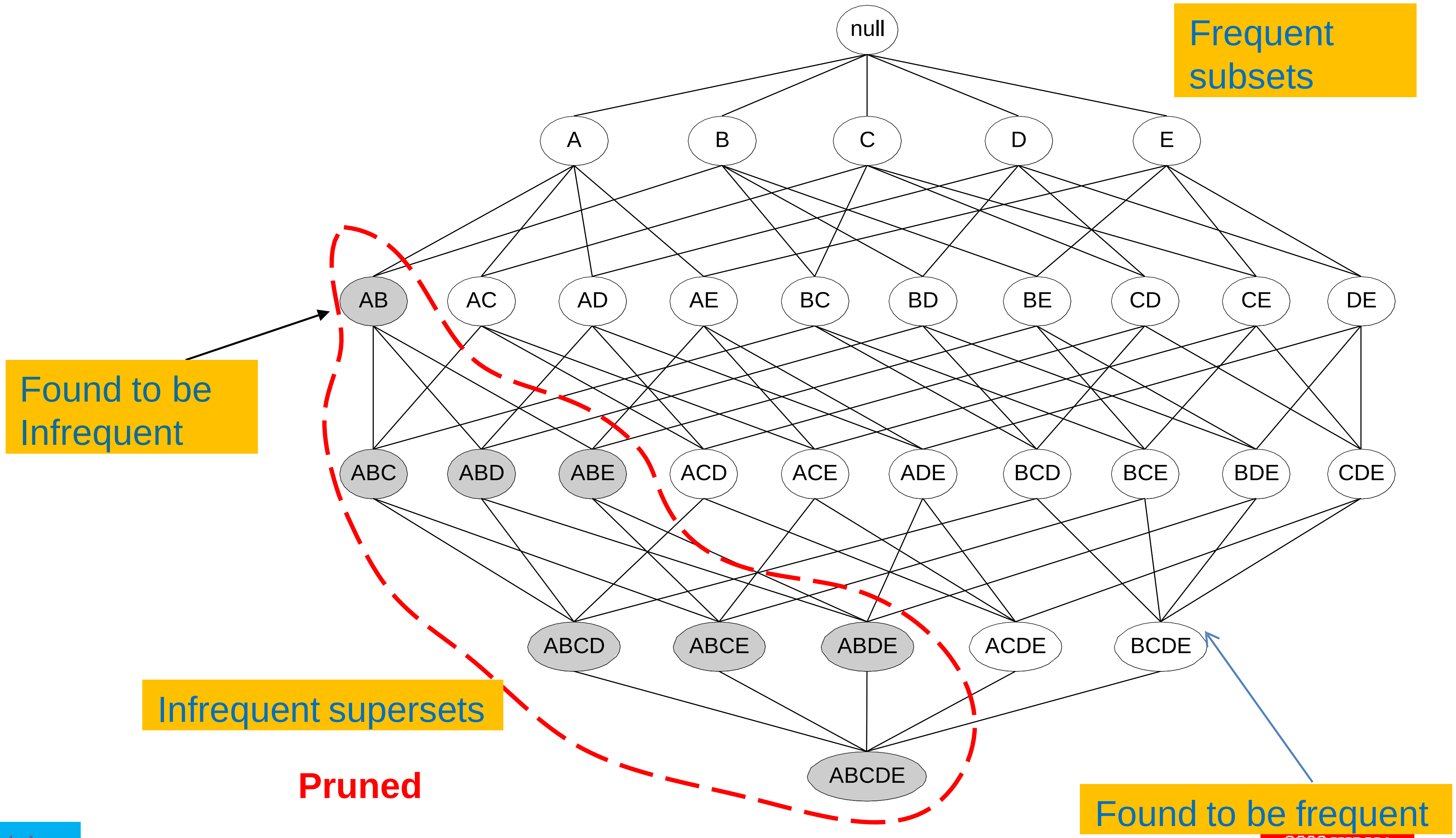
$\Rightarrow$Lift = 0.75/0.9= 0.8333 (< 1, therefore is negatively associated)

Frequent Itemset Identification: the Itemset Lattice

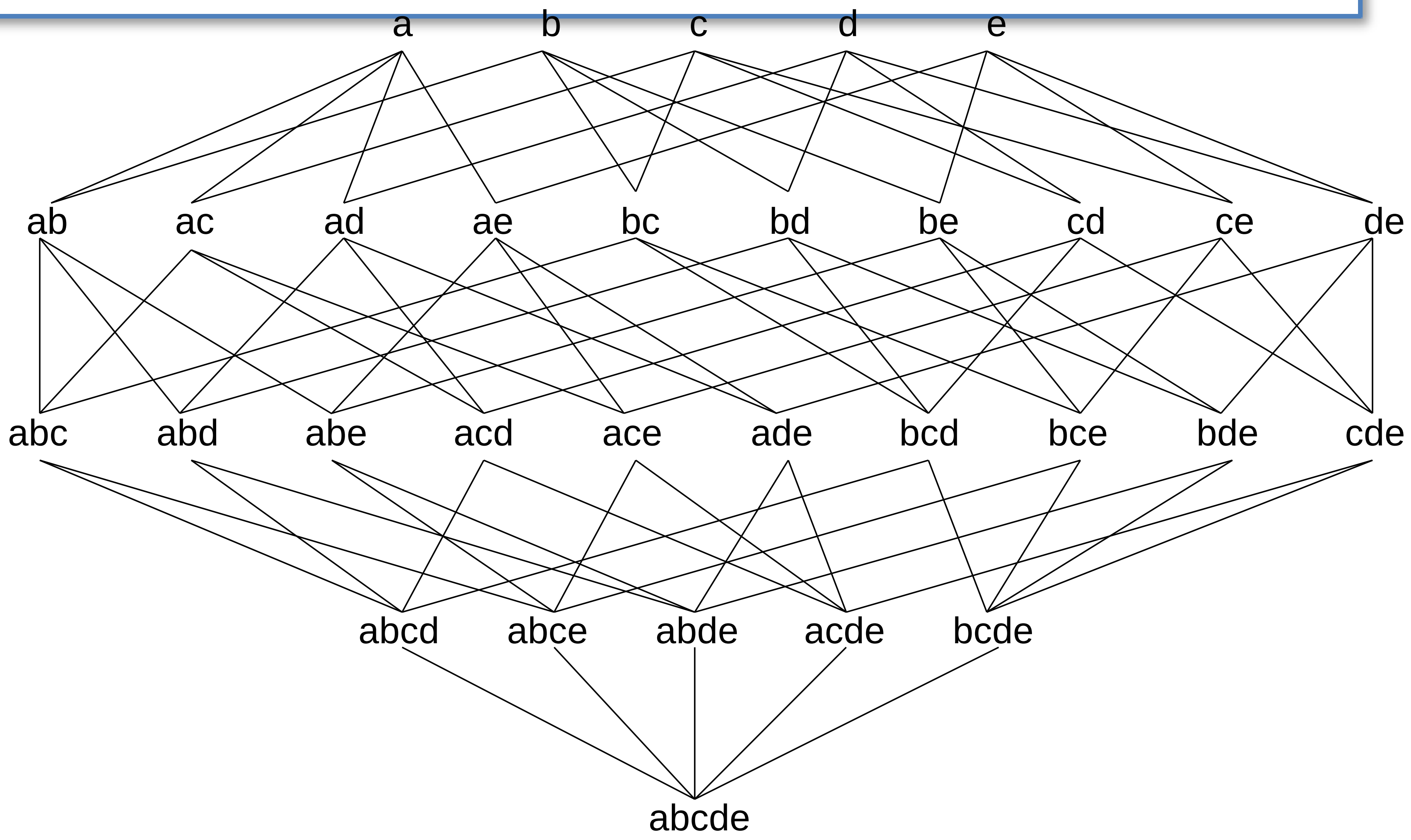


Given i items, there are 2^i-1 candidate itemsets!

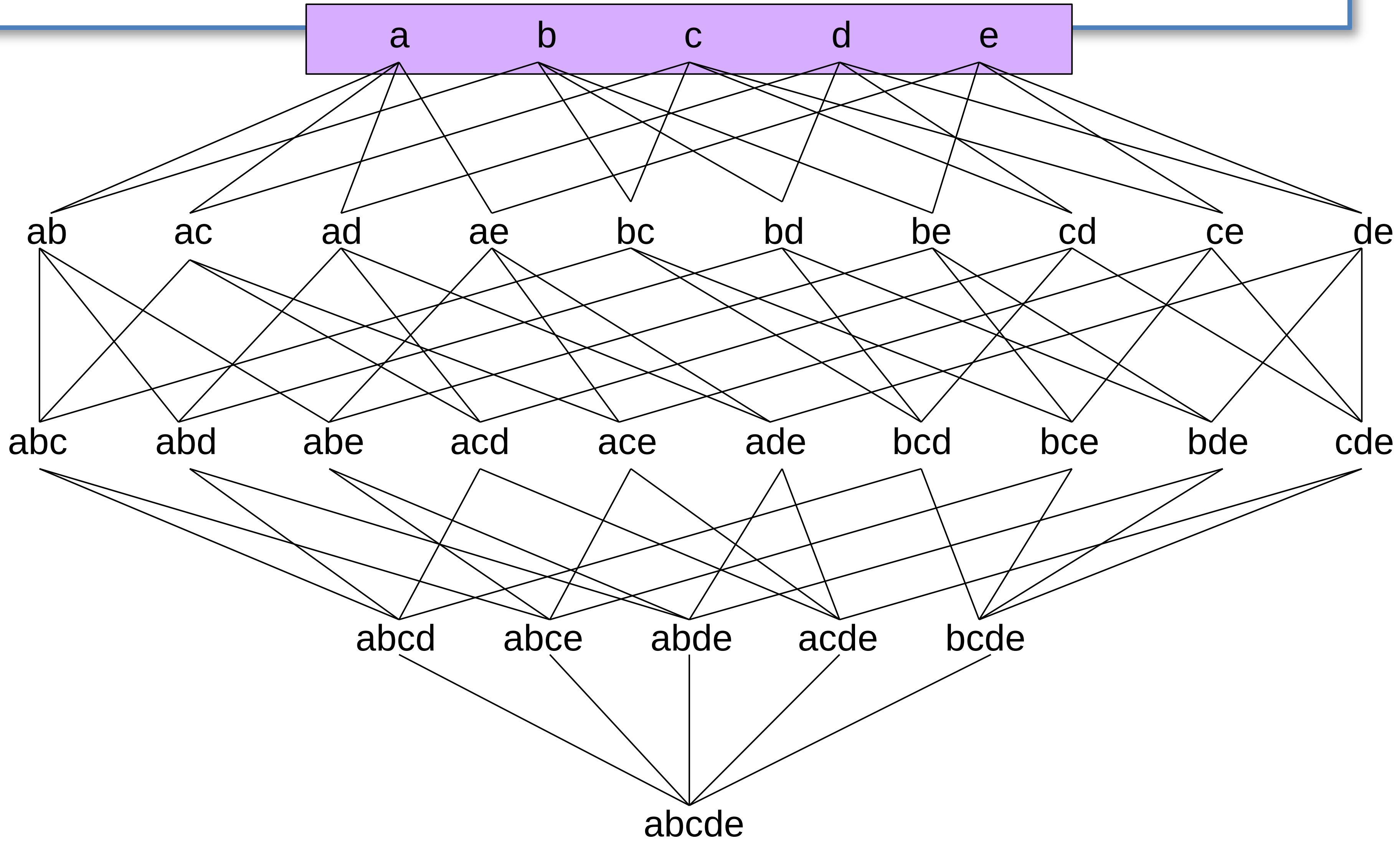
Illustration of the Aprioriprinciple



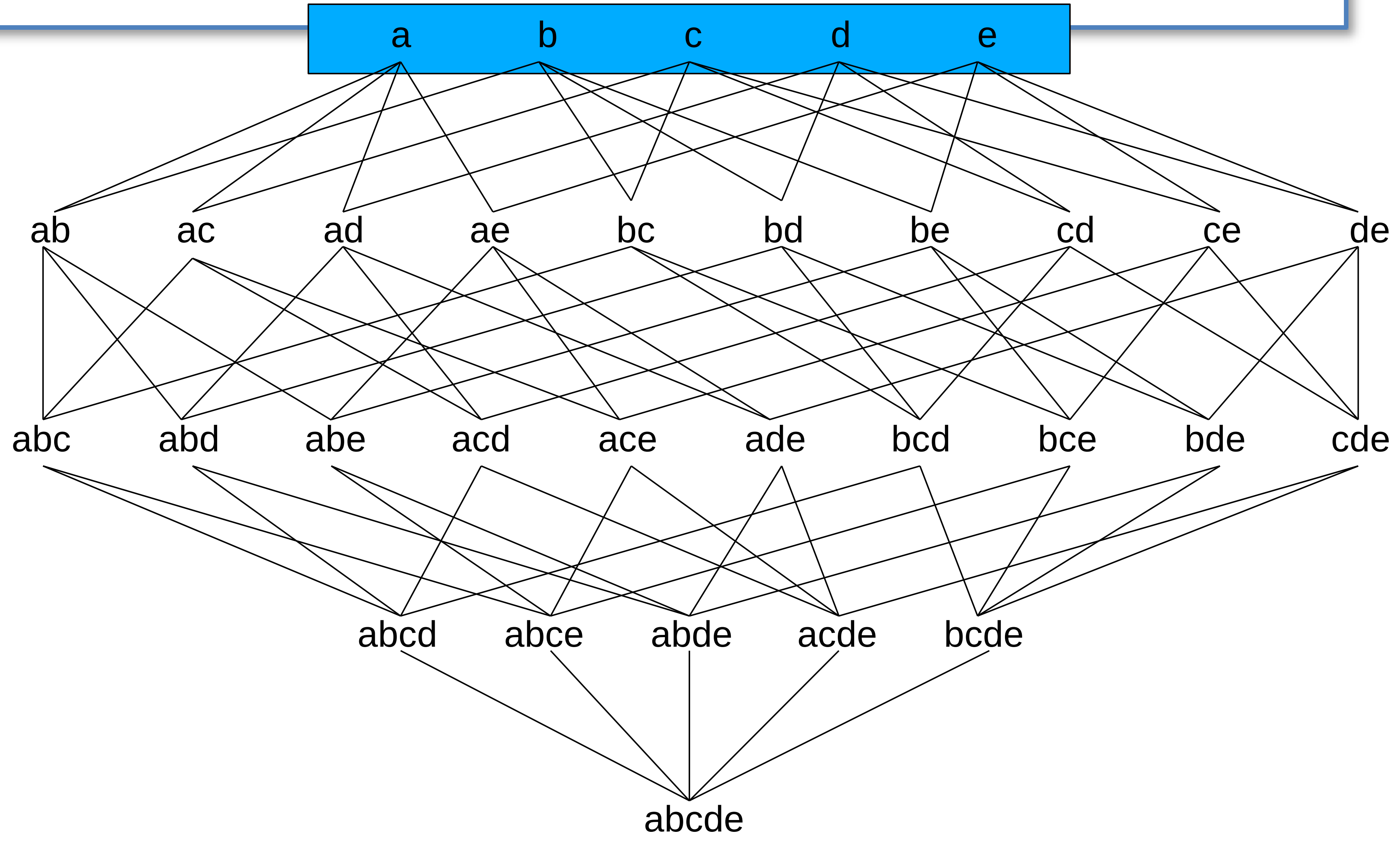
Apriori: *Breadth first* visit of the lattice



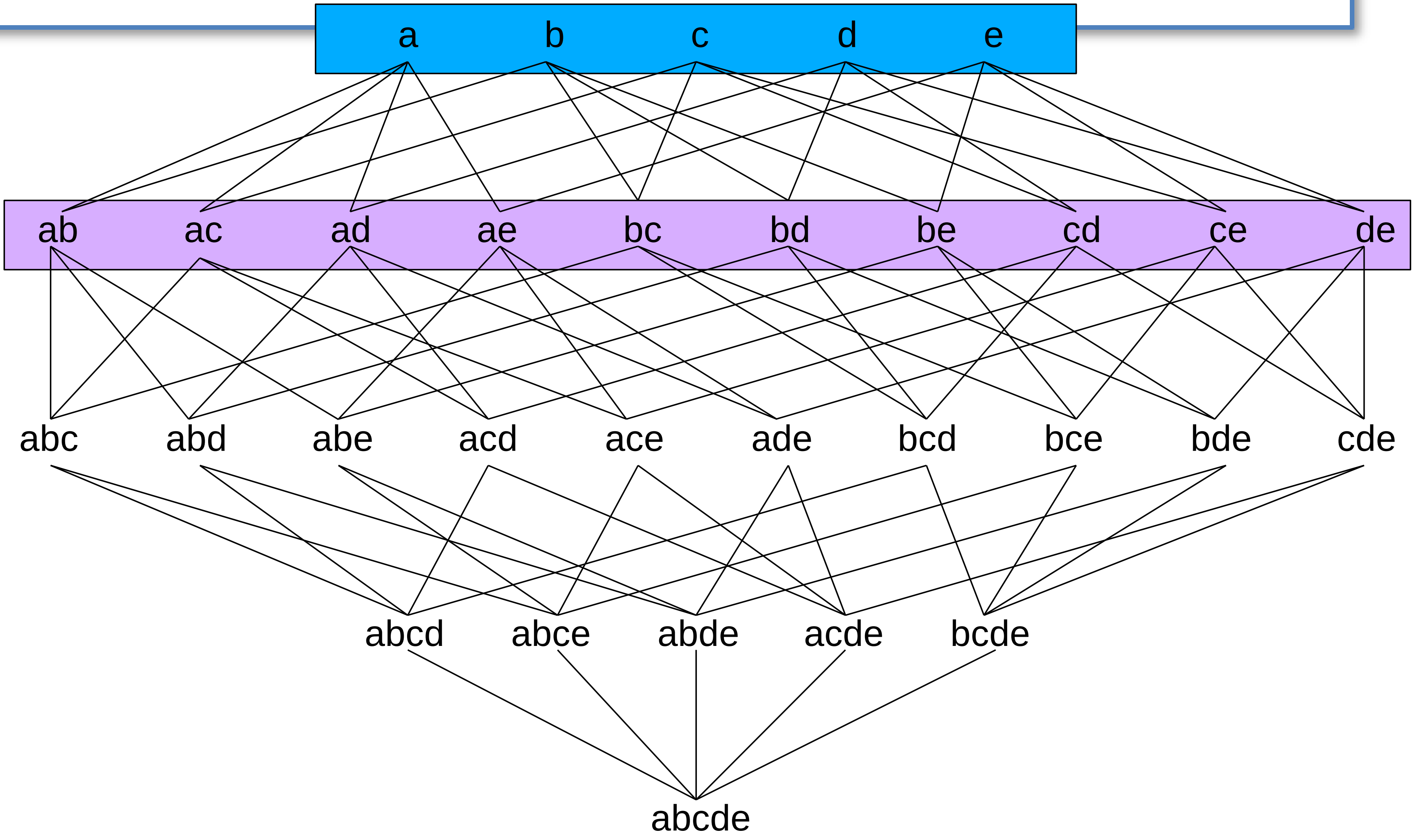
Generate the candidates of dimension 1



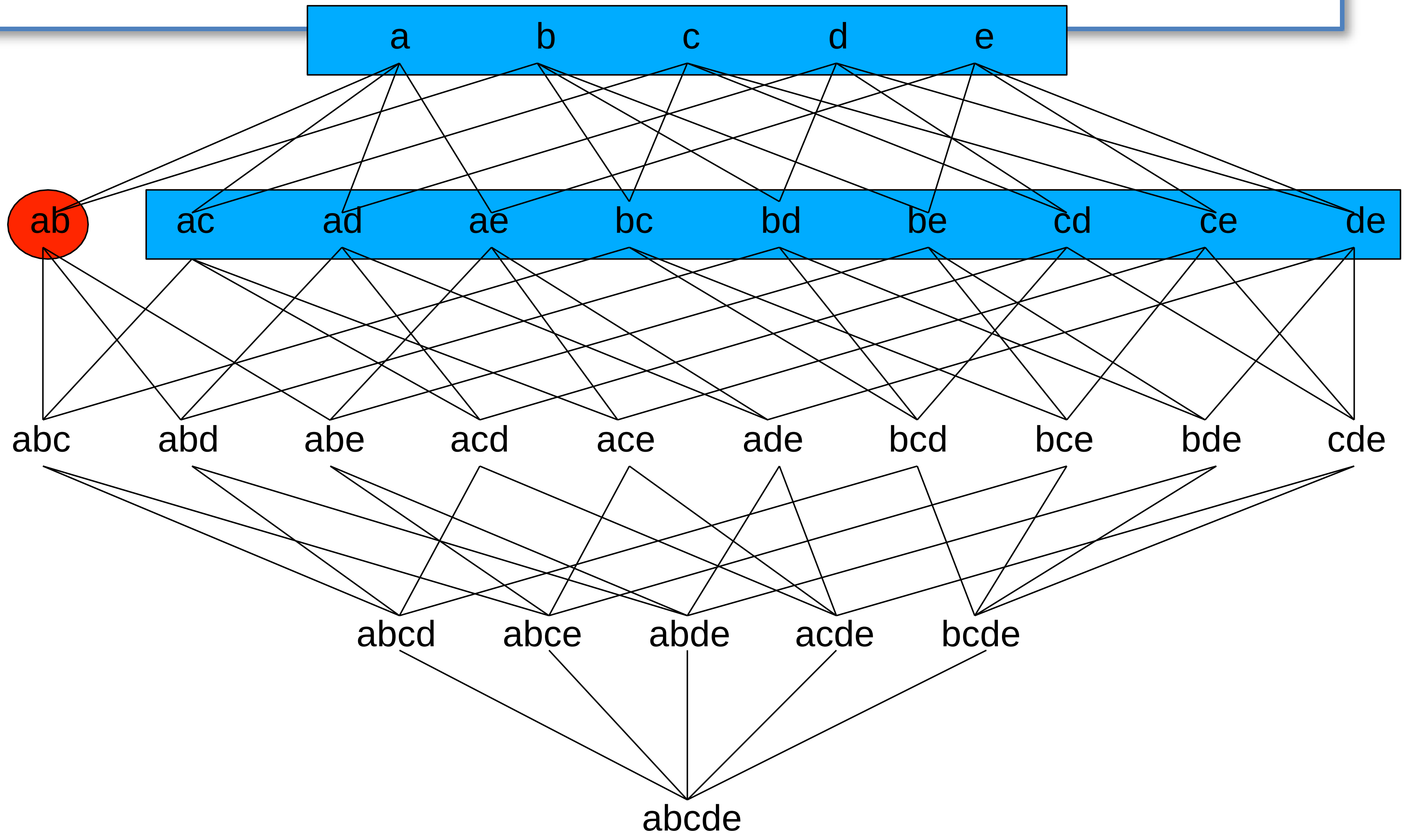
Compute the supports of the Candidates of dim. 1



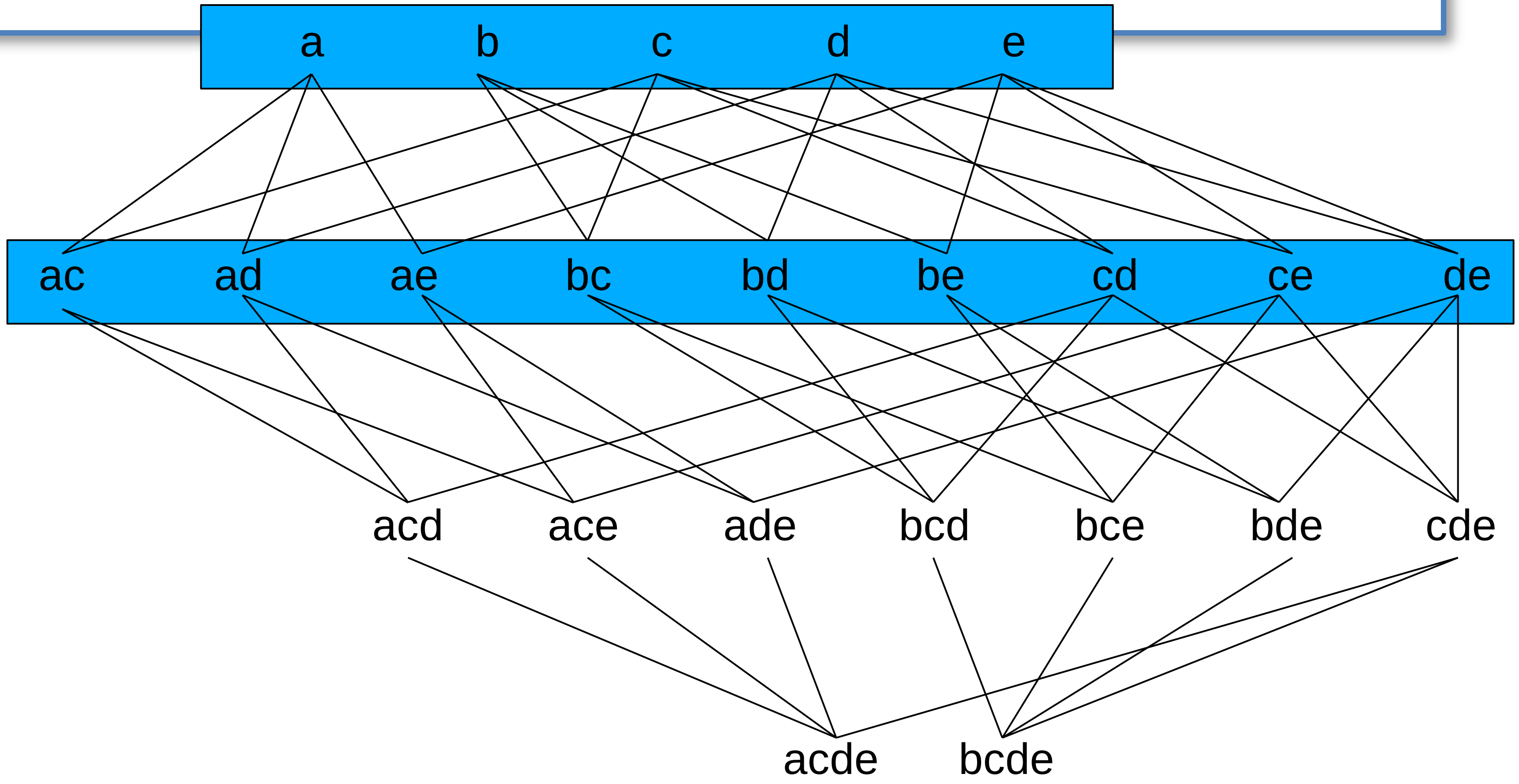
Generate the candidates of dimension 2



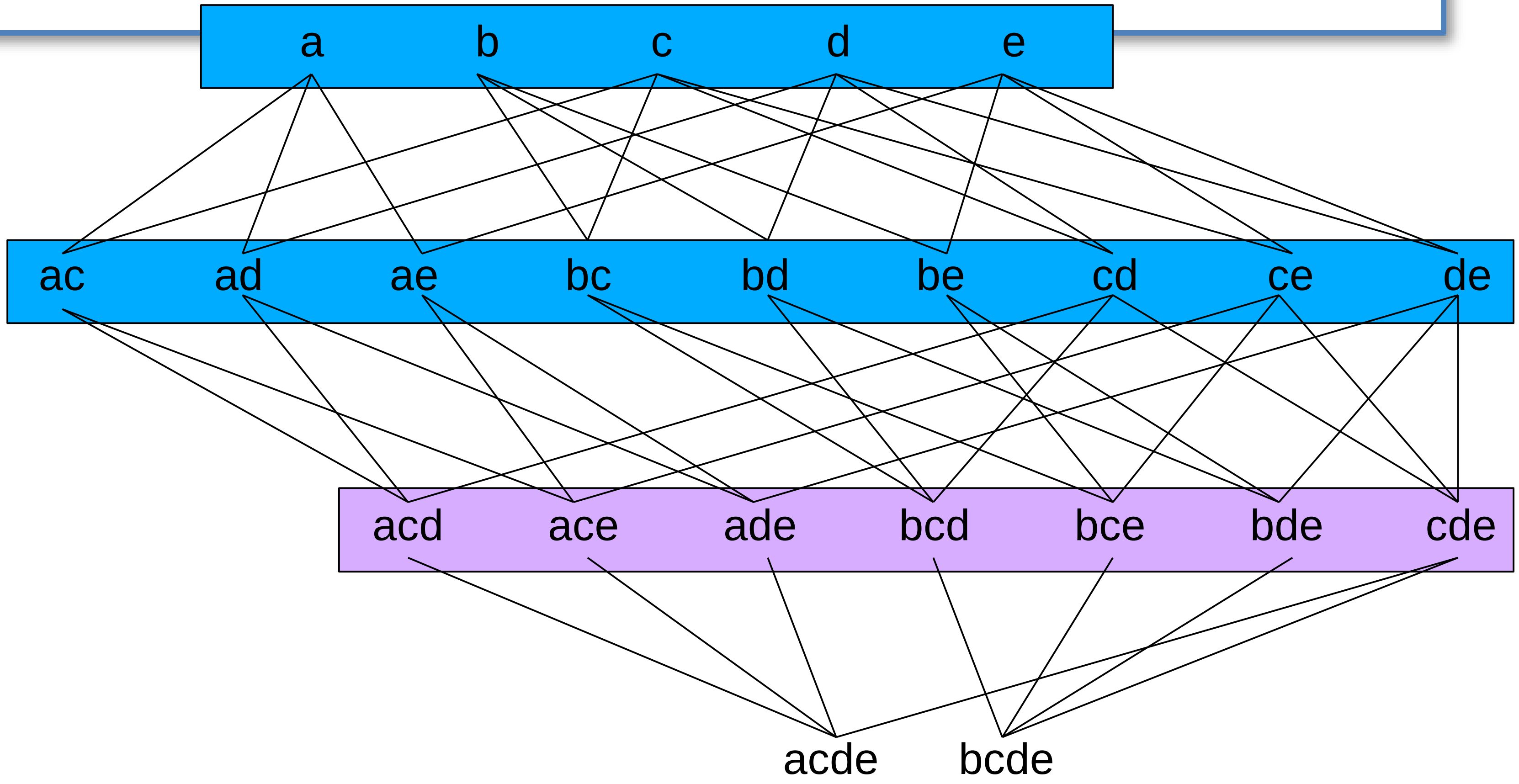
Compute the supports of the Candidates of dim. 2



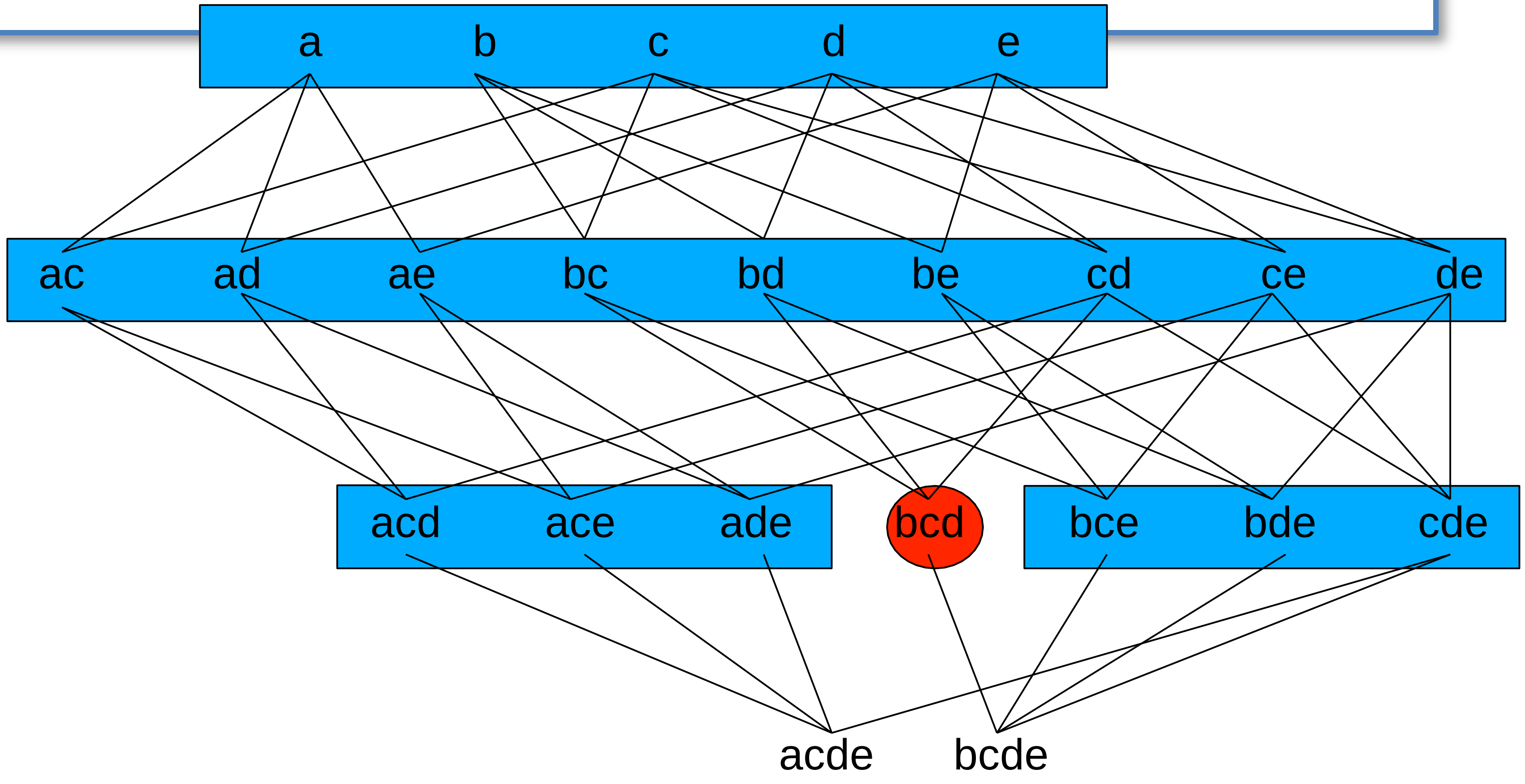
Prune the infrequent itemsets



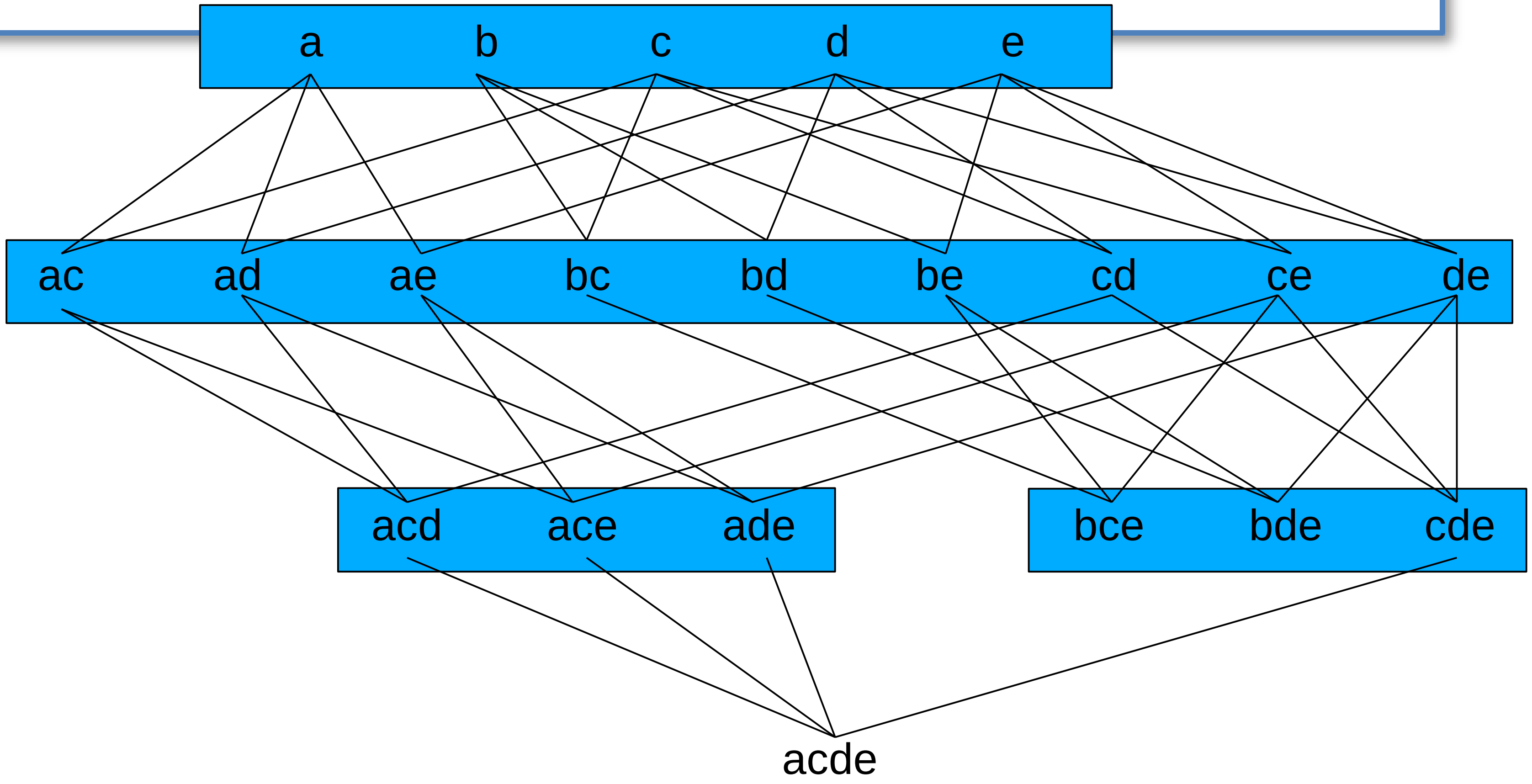
Generate the candidates of dimension 3



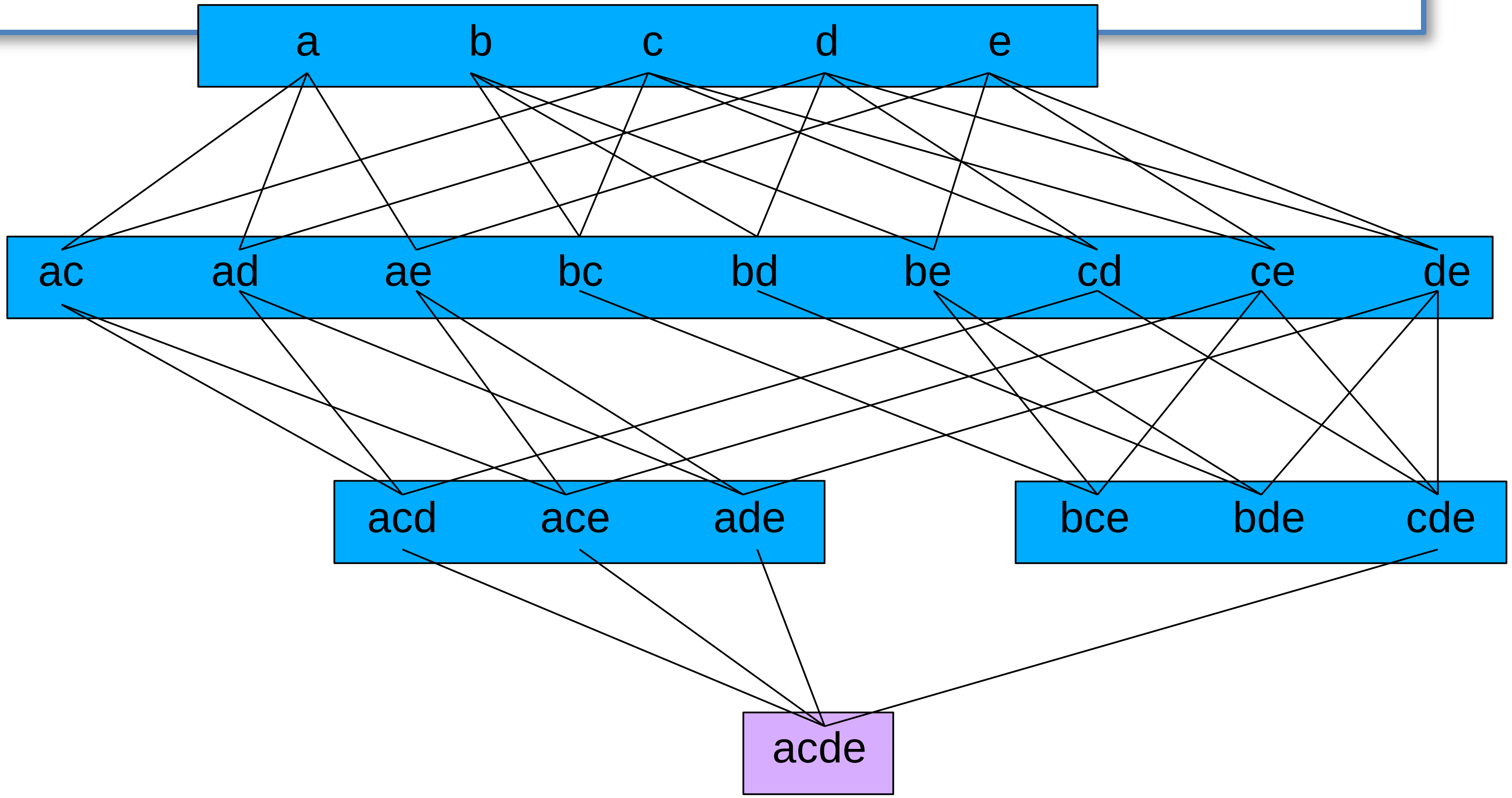
Compute the supports of the Candidates of dim. 3



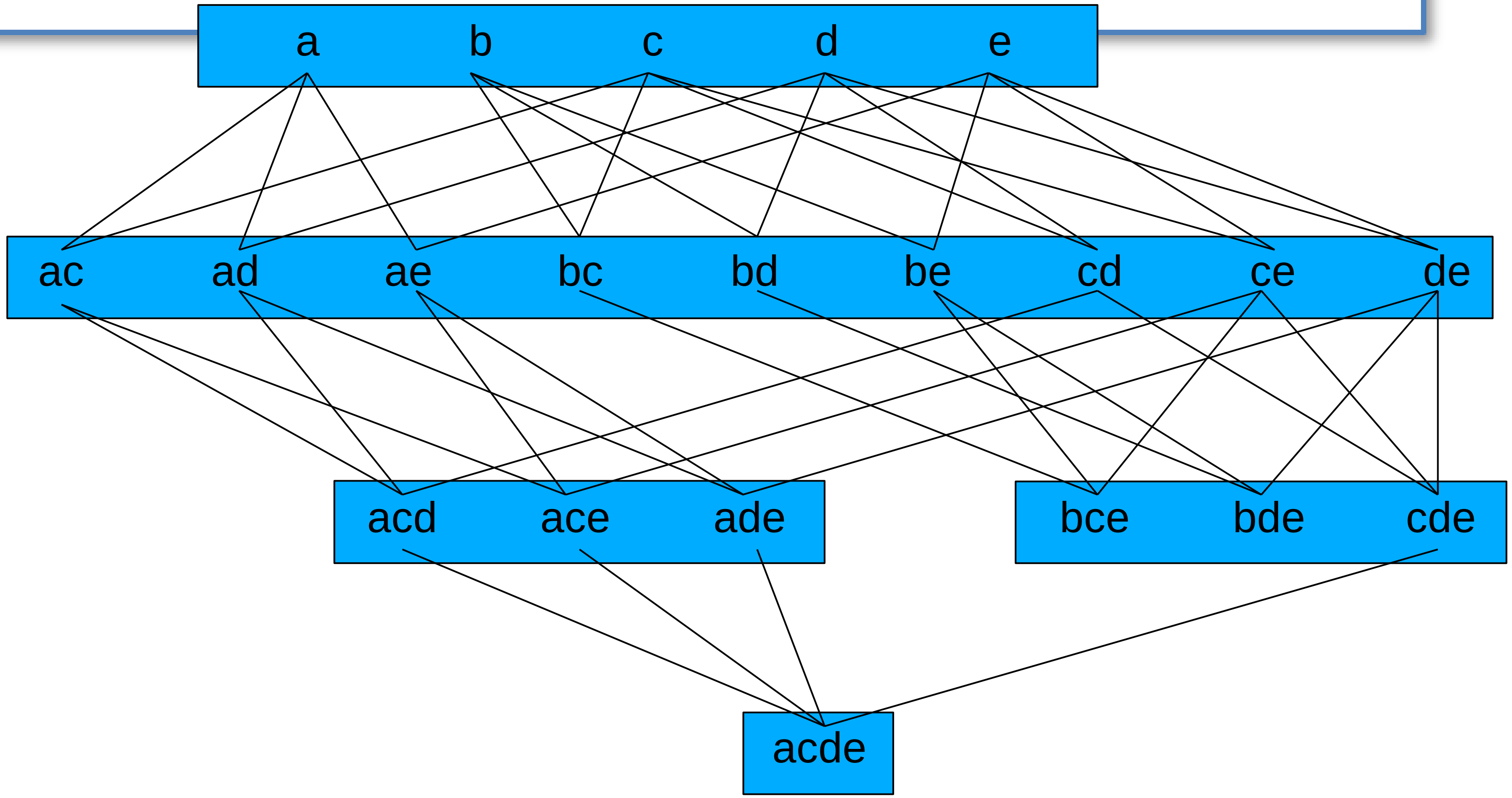
Prune the infrequent itemsets



Generate the candidates of dimension 3



Compute the supports of the Candidates of dim. 4



Mining Frequent Itemsets task

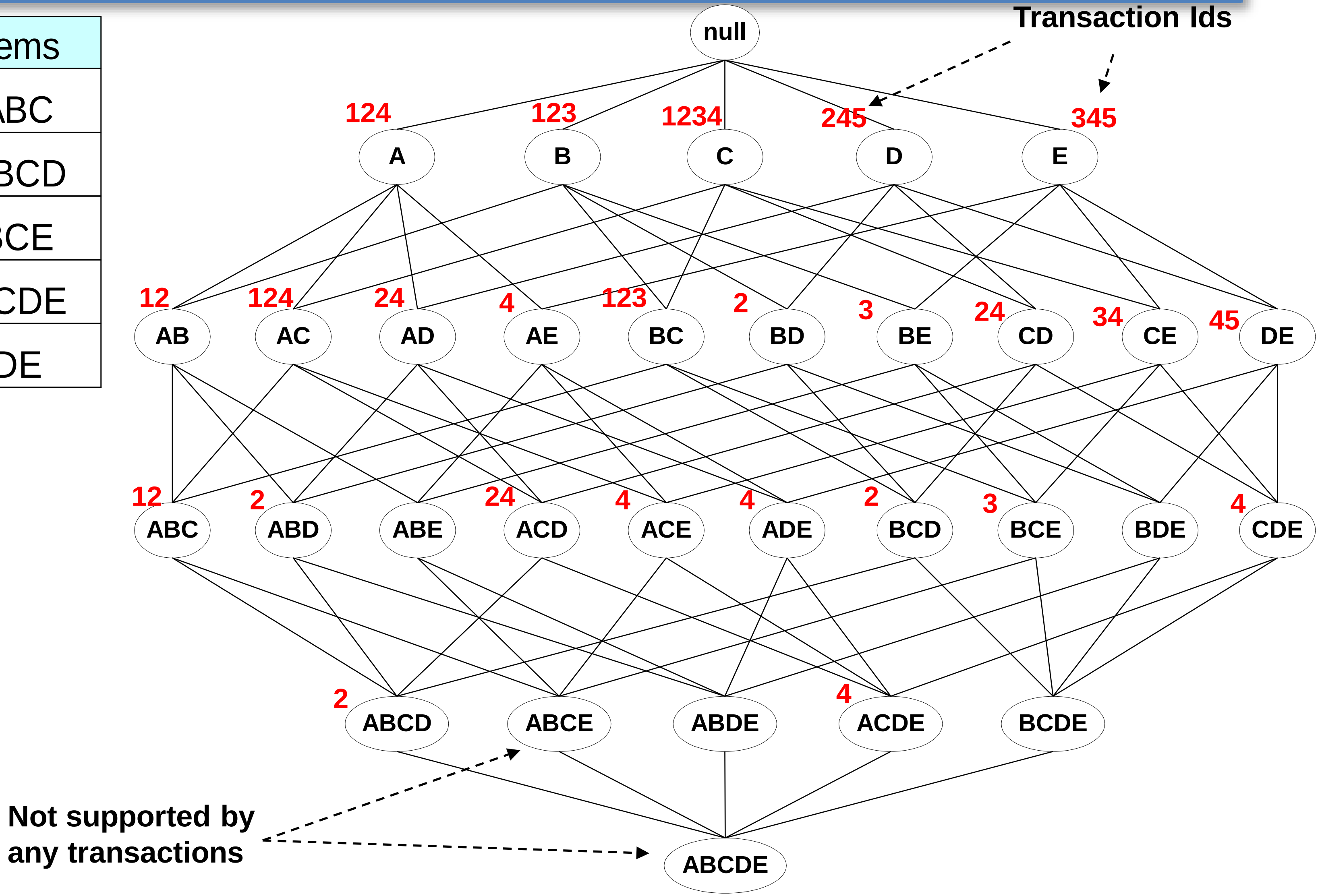
- Input: A set of transactions T, over a set of items I
- Output: All itemsets with items in I having
  - support ≥ *minsup* threshold

- Problem parameters:
  - N = |T|: number of transactions
  - d = |I|: number of (distinct) items
  - w: max width of a transaction
  - Number of possible itemsets? $M = 2^d$

- 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.

Compression of Itemset Information

TID	Items
1	ABC
2	ABCD
3	BCE
4	ACDE
5	DE



Example file: retail

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29
30 31 32
33 34 35
36 37 38 39 40 41 42 43 44 45 46
38 39 47 48
38 39 48 49 50 51 52 53 54 55 56 57 58
32 41 59 60 61 62
3 39 48
63 64 65 66 67 68
32 69
48 70 71 72
39 73 74 75 76 77 78 79
36 38 39 41 48 79 80 81
82 83 84
41 85 86 87 88
39 48 89 90 91 92 93 94 95 96 97 98 99 100 101
36 38 39 48 89
39 41 102 103 104 105 106 107 108
38 39 41 109 110
39 111 112 113 114 115 116 117 118
119 120 121 122 123 124 125 126 127 128 129 130 131 132 133
48 134 135 136
39 48 137 138 139 140 141 142 143 144 145 146 147 148 149
39 150 151 152
38 39 56 153 154 155
```

Example: items are positive integers, and each basket corresponds to a line in the file of space separated integers

Association Rules

- If-then rules about the contents of baskets.
- $\{i_1, i_2,\ldots,i_k\} \rightarrow j$ means: "if a basket contains all of $i_1,\ldots,i_k$ then it is likely to contain j.
- *Confidence* of this association rule is the probability of j given $i_1,\ldots,i_k$.

A typical question:

"find all association rules with support $\geq$ *s* and confidence $\geq$ c."

- B1 = {m, c, b}
- B3 = {m, b}
- B5 = {m, p, b}

B7 = {c, b, j}

B2 = {m, p, j}

B4 = {c, j}

- B6 = {m, c, b, j}

B8 = {b, c}

An association rule: {m, b} → c.

- Confidence = 2/4 = 50%.

Why Apriori?

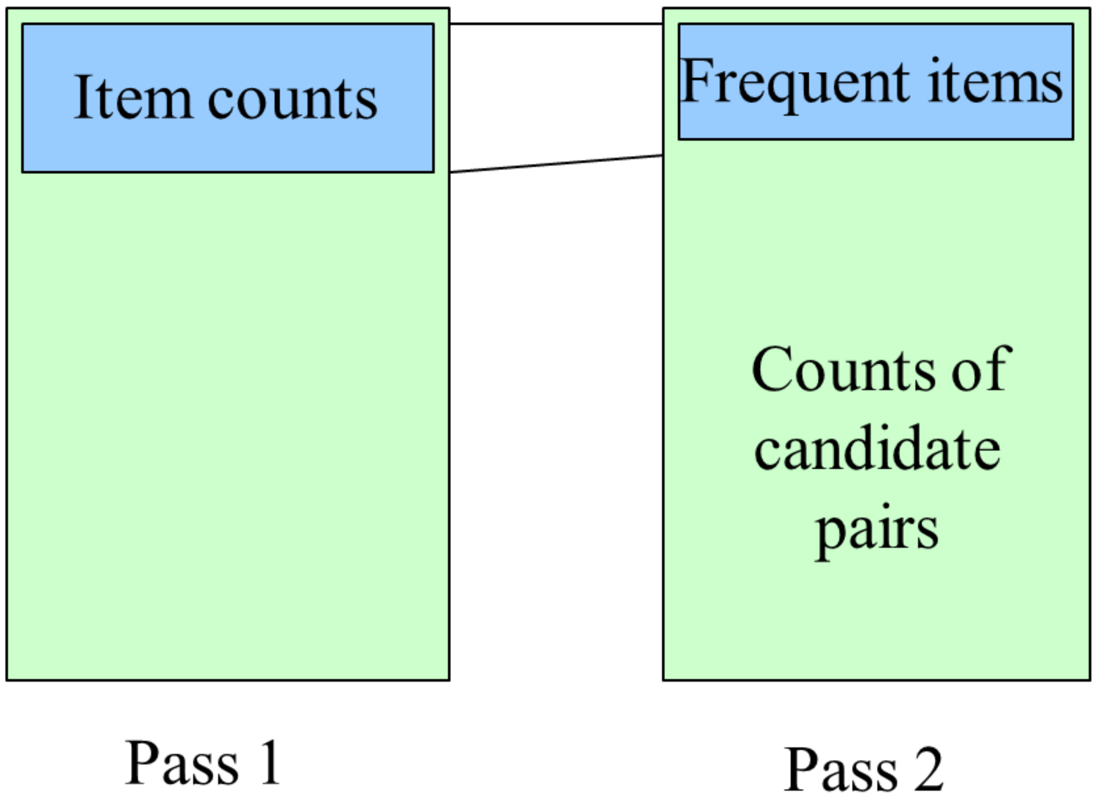
Number items 1,2,…

Keep pairs in the order {1,2}, {1,3},…, {1,n}, {2,3}, {2,4},…,{2,n}, {3,4},…, {3,n},…{n-1,n}.

Find pair {i, j} at the position $(i-1)(n-i/2)+j-i$.

Total number of pairs $n(n-1)/2$; total bytes about $2n^2$.

Picture of A-Priori



27

Frequent Itemsets

- C_1 = all items
- L_1 = those counted on first pass to be frequent.
- C_2 = pairs, both chosen from L_1.
- In general, $C_k = k$–tuples each $k-1$ of which is in $L_{k\text{-}1}$.
- L_k = those candidates with support>=s.

Illustration of the Aprioriprinciple

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

minsup = 3

Items (1-itemsets)

Item	Count
Bread	**4**
Coke	**2**
Milk Beer	**4**
Diaper	**3**
	4
Eggs	**1**

Pairs (2-itemsets)

Itemset	Count
{Bread,Milk}	**3**
{Bread,Beer}	**2**
{Bread,Diaper}	**3**
{Milk,Beer}	**2**
{Milk,Diaper}	**3**
{Beer,Diaper}	**3**

(No need to generate candidates involving Coke or Eggs)

Triplets (3-itemsets)

Itemset	Count
{Bread,Milk,Diaper}	**2**

If every subset is considered,

$$\binom{6}{1} + \binom{6}{2} + \binom{6}{3} = 6 + 15 + 20 = 41$$

With support-based pruning,

$$\binom{6}{1} + \binom{4}{2} + 1 = 6 + 6 + 1 = 13$$

Only this triplet has all subsets to be frequent
But it is below the minsup threshold

What is C (candidate) ? L(list of items with s)?

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

C_1'

TID	Sets of itemsets
100	{{1},{3},{4}}
200	{{2},{3},{5}}
300	{{1},{2},{3},{5}}
400	{{2},{5}}

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

C_2'

TID	Sets of itemsets
100	{{1 3}}
200	{{2 3},{2 5},{3 5}}
300	{{1 2},{1 3},{1 5}, {2 3},{2 5},{3 5}}
400	{{2 5}}

L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

TID	Sets of itemsets
200	{{2 3 5}}
300	{{2 3 5}}

C_3'

L_3

itemset	sup
{2 3 5}	2

minsup=2

Discussion on the AprioriTID algorithm

- L_1 = {frequent 1-itemsets}
- C_1' = database **D**
- **for** (k=2, $L_{k-1}' \neq$ empty; k++)

    C_k = GenerateCandidates(L_{k-1})

    C_k' = {}

    for all entries **t** є C_{k-1}'

        C_t= {cє C_k|t[c-c[k]]=1 and t[c-c[k-1]]=1}

        for all **cє C_t** {c.count++}

        if (**$C_t \neq$ {}**)

            append C_t to C_k'

        endif

    endfor

    L_k= {cє C_k|c.count >= minsup}

  endfor
- **return** $\mathbf{U_k}$ L_k

- One single pass over the data
- C_k' is generated from C_{k-1}'
- For small values of **k**, C_k' could be larger than the database!
- For large values of **k**, C_k' can be very small

The Apriori Principle

- Apriori principle (Main observation):
  - If an itemset is frequent, then all of its subsets must also be frequent
  - If an itemset is not frequent, then all of its supersets cannot be frequent

$$\forall X, Y : (X \subseteq Y) \Rightarrow s(X) \geq s(Y)$$

– The support of an itemset never exceeds the support of its subsets

– This is known as the anti-monotone property of support

Anti-Monotone Property

- Any subset of a *frequent* itemset must be also *frequent* — an anti-monotone property
  - Any transaction containing {beer, diaper, milk} also contains {beer, diaper}
  - {beer, diaper, milk} is frequent → {beer, diaper} must also be frequent
- In other words, any superset of an *infrequent* itemset must also be *infrequent*
  - No superset of any infrequent itemset should be generated or tested
  - Many item combinations can be pruned!

The Apriori Algorithm

- C_k: Candidate itemset of size k
- L_k : frequent itemset of size k

- L_1 = {frequent items};
- for (k = 1; L_k !=$\emptyset$; k++) do
  - *Candidate Generation:* C_{k+1} = candidates generated from L_k;
  - *Candidate Counting:* for each transaction *t* in database do increment the count of all candidates in C_{k+1} that are contained in *t*
  - L_{k+1} = candidates in C_{k+1} with min_sup
- return $\cup_k L_k$;

Candidate-generation: Self-joining

- Given L_k, how to generate C_{k+1}?

  Step 1: self-joining L_k

  INSERT INTO C_{k+1}

  SELECT $p.item_1, p.item_2, \ldots, p.item_k, q.item_k$

  FROM $L_k\ p, L_k\ q$

  WHERE $p.item_1=q.item_1, \ldots, p.item_{k-1}=q.item_{k-1}, p.item_k < q.item_k$

- Example

  $L_3=\{abc, abd, acd, ace, bcd\}$

  Self-joining: L_3*L_3

  - $abcd \leftarrow abc * abd$
  - $acde \leftarrow acd * ace$

  $C_4=\{abcd, acde\}$

Candidate Generation: Pruning

- Can we further reduce the candidates in C_{k+1}?

For each itemset *c* in C_{k+1} do

For each *k*-subsets *s* of *c* do

If (*s* is not in L_k) Then delete *c* from C_{k+1}

End For

End For

- Example

L_3={*abc, abd, acd, ace, bcd*}, C_4={*abcd, acde*}

acde cannot be frequent since *ade* (and also *cde*) is not in L_3, so *acde* can be pruned from C_4.

The Apriori algorithm

Level-wise approach

C_k = candidate itemsets of size k
L_k = frequent itemsets of size k

1. k = 1, C_1 = all items
2. While C_k not empty

Frequent itemset generation

3. Scan the database to find which itemsets in C_k are frequent and put them into L_k

Candidate generation

4. Use L_k to generate a collection of candidate itemsets C_{k+1} of size k+1
5. k = k+1

R. Agrawal, R. Srikant: "Fast Algorithms for Mining Association Rules", *Proc. of the 20th Int'l Conference on Very Large Databases*, 1994.

Candidate Generation

- Basic principle (Apriori):
  - An itemset of size k+1 is candidate to be frequent only if all of its subsets of size k are known to be frequent
- Main idea:
  - Construct a candidate of size k+1 by combining frequent itemsets of size k
    - If k = 1, take the all pairs of frequent items
    - If k > 1, join pairs of itemsets that differ by just one item
    - For each generated candidate itemset ensure that all subsets of size k are frequent.

Generate Candidates C_{k+1}

- Assumption: The items in an itemset are ordered
  - E.g., if integers ordered in increasing order, if strings ordered in lexicographic order
    - The order ensures that if item y > x appears before x, then x is not in the itemset
- The items in L_k are also listed in an order

Create a candidate itemset of size k+1, by joining two itemsets of size k, that share the first k-1 items

Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

Generate Candidates C_{k+1}

- Assumption: The items in an itemset are ordered
  - E.g., if integers ordered in increasing order, if strings ordered in lexicographic order
    - The order ensures that if item y > x appears before x, then x is not in the itemset
- The items in L_k are also listed in an order

Create a candidate itemset of size k+1, by joining two itemsets of size k, that share the first k-1 items

Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

1	2	3	5

Generate Candidates C_{k+1}

- Assumption: The items in an itemset are ordered
  - E.g., if integers ordered in increasing order, if strings ordered in lexicographic order
    - The order ensures that if item y > x appears before x, then x is not in the itemset
- The items in L_k are also listed in an order

Create a candidate itemset of size k+1, by joining two itemsets of size k, that share the first k-1 items

Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

1	2	4	5

Are we missing something?
What about this candidate?

Generating Candidates C_{k+1} in SQL

- ***self-join*** L_k

  insert into $\boldsymbol{C_{k+1}}$

  select $\boldsymbol{p.item_1, p.item_2, \ldots, p.item_k, q.item_k}$

  from $\boldsymbol{L_k\ p, L_k\ q}$

  where $\boldsymbol{p.item_1{=}q.item_1, \ldots, p.item_{k-1}{=}q.item_{k-1}, p.item_k < q.item_k}$

Example I

- L_3={abc, abd, acd, ace, bcd}
- ***Self-join***: $L_3 * L_3$
  - abcd from abc and abd
  - acde from acd and ace

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

$p.item_1=q.item_1, p.item_2=q.item_2, p.item_3 < q.item_3$

Example I

- L_3={abc, abd, acd, ace, bcd}
- ***Self-joining***: $L_3 * L_3$
  - abcd from abc and abd
  - acde from acd and ace

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

$p.item_1=q.item_1, p.item_2=q.item_2, p.item_3 < q.item_3$

Example I

- L_3={abc, abd, acd, ace, bcd}
- ***Self-joining***: L_3*L_3
  - abcd from abc and abd
  - acde from acd and ace

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

{a,b,c} {a,b,d}

{a,b,c,d}

p.item$_1$=q.item$_1$,p.item$_2$=q.item$_2$, p.item$_3$< q.item$_3$

Example I

- L_3={abc, abd, acd, ace, bcd}
- ***Self-joining***: L_3*L_3
  - abcd from abc and abd
  - acde from acd and ace

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

item1	item2	item3
a	b	c
a	b	d
a	c	d
a	c	e
b	c	d

{a,c,d} {a,c,e}

{a,c,d,e}

p.item$_1$=q.item$_1$,p.item$_2$=q.item$_2$, p.item$_3$< q.item$_3$

Example II

Itemset	Count
{Beer,Diaper}	**3**
{Bread,Diaper}	**3**
{Bread,Milk}	**3**
{Diaper, Milk}	**3**

Itemset	Count
{Beer,Diaper}	**3**
{Bread,Diaper}	**3**
{Bread,Milk}	**3**
{Diaper, Milk}	**3**

Itemset
{Bread,Diaper,Milk}

{Bread,Diaper} √

{Bread,Milk} √

{Diaper, Milk} √

Generate Candidates C_{k+1}

- Are we done? Are all the candidates valid?

Item 1	Item 2	Item 3
1	2	3
1	2	5
1	4	5

1	2	3	5

Is this a valid candidate?

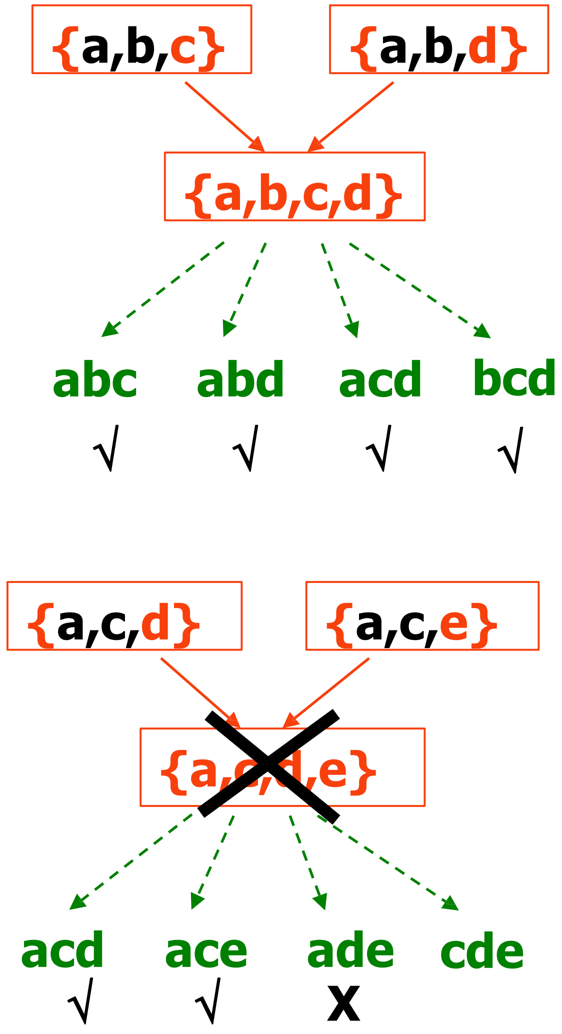
No. Subsets (1,3,5) and (2,3,5) should also be frequent

Apriori principle

- Pruning step:
  - For each candidate (k+1)-itemset create all subset k-itemsets
  - Remove a candidate if it contains a subset k-itemset that is not frequent

Example I

- *L_3={abc, abd, acd, ace, bcd}*
- ***Self-joining***: *L_3*L_3*
  - ***abcd*** from ***abc*** and ***abd***
  - ***acde*** from ***acd*** and ***ace***
- ***Pruning:***
  - ***abcd*** is kept since all subset itemsets are in ***L_3***
  - ***acde*** is removed because ***ade*** is not in ***L_3***
- *C_4={abcd}*



Generate Candidates C_{k+1}

- We have all frequent k-itemsets L_k
- **Step 1: self-join** L_k
  - Create set C_{k+1} by joining frequent k-itemsets that share the first k-1 items
- **Step 2: prune**
  - Remove from C_{k+1} the itemsets that contain a subset k-itemset that is not frequent

Summary of Apriori Algorithm

- Basic idea of Apriori
  - Using anti-monotone property to reduce candidate itemsets
  - Any subset of a *frequent* itemset must be also *frequent*
  - In other words, any superset of an *infrequent* itemset must also be *infrequent*
- Basic operations of Apriori
  - Candidate generation
  - Candidate counting
- How to generate the candidate itemsets?
  - Self-joining
  - Pruning infrequent candidates

The Apriori Algorithm — Example

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D → C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

→ L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D → C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

→ L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

Scan D → L_3

itemset	sup
{2 3 5}	2

Naïve Algorithm: Apriori + Constraint

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
~~{5}~~	~~3~~

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

L_2

itemset	sup
{1 3}	2
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

C_3

itemset
{2 3 5}

Scan D

L_3

itemset	sup
~~{2 3 5}~~	~~2~~

Constraint:

Sum{S.price < 5}

Pushing the constraint deep into the process

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
~~{5}~~	~~3~~

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
~~{5}~~	~~3~~

C_2

itemset
{1 2}
{1 3}
~~{1 5}~~
{2 3}
~~{2 5}~~
~~{3 5}~~

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
~~{1 5}~~	~~1~~
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

L_2

itemset	sup
{1 3}	2
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

C_3

itemset
~~{2 3 5}~~

Scan D

L_3

itemset	sup
~~{2 3 5}~~	~~2~~

Constraint:

Sum{S.price < 5}

Push a Succinct Constraint Deep

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D

C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
~~{2 3}~~
~~{2 5}~~
~~{3 5}~~

Scan D

C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

L_2

itemset	sup
{1 3}	2
~~{2 3}~~	~~2~~
~~{2 5}~~	~~3~~
~~{3 5}~~	~~2~~

C_3

itemset
~~{2 3 5}~~

Scan D

L_3

itemset	sup
~~{2 3 5}~~	~~2~~

Constraint:

min{S.price <= 1 }

The Apriori Algorithm — Example

Database D

TID	Items
100	1 3 4
200	2 3 5
300	1 2 3 5
400	2 5

Scan D → C_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{4}	1
{5}	3

→ L_1

itemset	sup.
{1}	2
{2}	3
{3}	3
{5}	3

C_2

itemset
{1 2}
{1 3}
{1 5}
{2 3}
{2 5}
{3 5}

Scan D → C_2

itemset	sup
{1 2}	1
{1 3}	2
{1 5}	1
{2 3}	2
{2 5}	3
{3 5}	2

→ L_2

itemset	sup
{1 3}	2
{2 3}	2
{2 5}	3
{3 5}	2

C_3

itemset
{2 3 5}

Scan D → L_3

itemset	sup
{2 3 5}	2

Apriori

Apriori-based Mining

Data base D

TID	Items
10	a, c, d
20	b, c, e
30	a, b, c, e
40	b, e

Min_sup=0.5

Scan D

1-candidates

Itemset	Sup
a	2
b	3
c	3
d	1
e	3

Freq 1-itemsets

Itemset	Sup
a	2
b	3
c	3
e	3

2-candidates

Itemset
ab
ac
ae
bc
be
ce

Scan D

Counting

Itemset	Sup
ab	1
ac	2
ae	1
bc	2
be	3
ce	2

Freq 2-itemsets

Itemset	Sup
ac	2
bc	2
be	3
ce	2

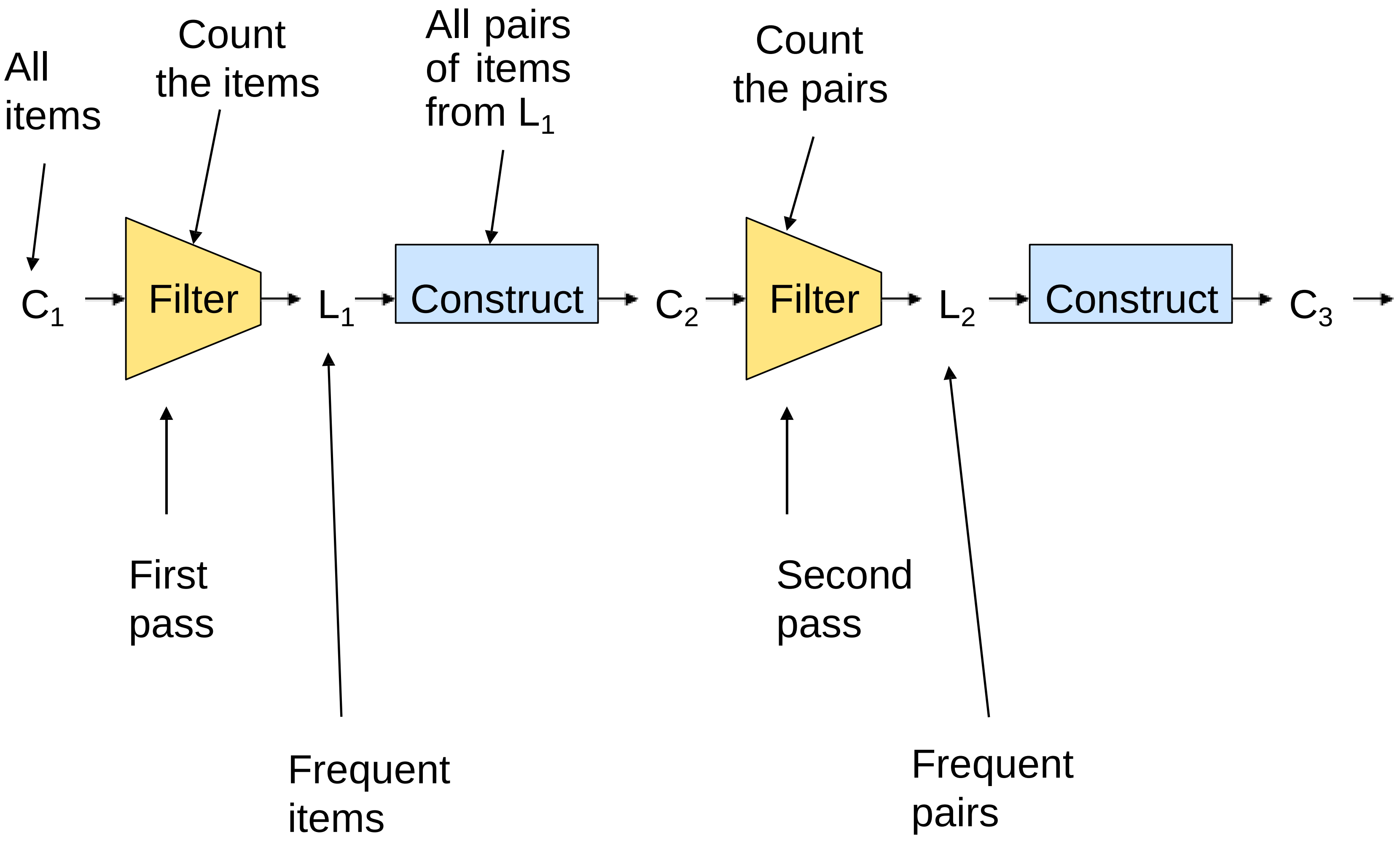
3-candidates

Itemset
bce

Scan D

Freq 3-itemsets

Itemset	Sup
bce	2


All items
Count the items
All pairs of items from L_1
Count the pairs
C_1
Filter
L_1
Construct
C_2
Filter
L_2
Construct
C_3
First pass
Frequent items
Second pass
Frequent pairs

ASSOCIATION RULES

- 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.

Association Rule Mining

- Given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction

Market-Basket transactions
Example of Association Rules

{Diaper} $\rightarrow$ {Beer},
{Milk, Bread} $\rightarrow$ {Eggs,Coke},
{Beer, Bread} $\rightarrow$ {Milk},

Implication means co-occurrence, not causality!

TID	*Items*
1	**Bread, Milk**
2	**Bread, Diaper, Beer, Eggs**
3	**Milk, Diaper, Beer, Coke**
4	**Bread, Milk, Diaper, Beer**
5	**Bread, Milk, Diaper, Coke**

Definition: Association Rule

- **Association Rule**
  - An implication expression of the form $X \rightarrow Y$, where X and Y are itemsets
  - Example:
    - {Milk, Diaper} $\rightarrow$ {Beer}
- **Rule Evaluation Metrics**
  - Support (s)
    - Fraction of transactions that contain both X and Y
    - the probability P(X,Y) that X and Y occur together
  - Confidence (c)
    - Measures how often items in Y appear in transactions that contain X
    - the conditional probability P(X|Y) that X occurs given that Y has occurred.

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

Example:

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

$$s = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{|T|} = \frac{2}{5} = 0.4$$

$$c = \frac{\sigma(\text{Milk}, \text{Diaper}, \text{Beer})}{\sigma(\text{Milk}, \text{Diaper})} = \frac{2}{3} = 0.67$$

Association Rule Mining Task

- Input: A set of transactions T, over a set of items I
- Output: All rules with items in I having
  - support ≥ *minsup* threshold
  - confidence ≥ *minconf* threshold

Mining Association Rules

- Two-step approach:
  1. Frequent Itemset Generation
     - Generate all itemsets whose support ≥ minsup
  2. Rule Generation
     - Generate high confidence rules from each frequent itemset, where each rule is a partitioning of a frequent itemset into Left-Hand-Side (LHS) and Right-Hand-Side (RHS)

Frequent itemset: {A,B,C,D} Rule: AB→CD

Rule Generation

- We have all frequent itemsets, how do we get the rules?
  - For every frequent itemset S, we find rules of the form $L \rightarrow S - L$, where $L \subset S$, that satisfy the minimum confidence requirement
  - Example: $L = \{A,B,C,D\}$
  - Candidate rules:

$A \rightarrow BCD$, $B \rightarrow ACD$, $C \rightarrow ABD$, $D \rightarrow ABC$
$AB \rightarrow CD$, $AC \rightarrow BD$, $AD \rightarrow BC$, $BD \rightarrow AC$, $CD \rightarrow AB$, $ABC \rightarrow D$, $BCD \rightarrow A$, $BC \rightarrow AD$,

- If $|L| = k$, then there are $2^k - 2$ candidate association rules (ignoring $L \rightarrow \varnothing$ and $\varnothing \rightarrow L$)

Rule Generation

- How to efficiently generate rules from frequent itemsets?
  - In general, confidence does not have an anti-monotone property

$c(ABC \rightarrow D)$ can be larger or smaller than $c(AB \rightarrow D)$

  - But confidence of rules generated from the same itemset has an anti-monotone property
  - e.g., $L = \{A,B,C,D\}$:

$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

- Confidence is anti-monotone w.r.t. number of items on the RHS of the rule

Rule Generation for APriori Algorithm

- Candidate rule is generated by merging two rules that share the same prefix
in the RHS
CD->AB BD->AC

- join(CD→AB,BD→AC) would produce the candidate rule D → ABC

- Prune rule D → ABC if its subset AD→BC does not have high confidence

D->ABC

- Essentially we are doing APriori on the RHS

Association Rule Mining Task

- An association rule *r* is *strong* if
  - *Support(r)* $\geq$ *min_sup*
  - *Confidence(r)* $\geq$ *min_conf*
- Given a transactions database D, the goal of association rule mining is to find all *strong* rules
- Two-step approach:
  1. Frequent Itemset Identification
     – Find all itemsets whose support $\geq$ *min_sup*
  2. Rule Generation
     – From each frequent itemset, generate all confident rules whose confidence $\geq$ *min_conf*

Rule Generation

Suppose min_sup=0.3, min_conf=0.6,
Support({Beer, Diaper, Milk})=0.4

TID	Items
1	Bread, Milk
2	Bread, Diaper, Beer, Eggs
3	Milk, Diaper, Beer, Coke
4	Bread, Milk, Diaper, Beer
5	Bread, Milk, Diaper, Coke

All candidate rules:

{Beer} → {Diaper, Milk} (s=0.4, c=0.67)
{Diaper} → {Beer, Milk} (s=0.4, c=0.5)
{Milk} → {Beer, Diaper} (s=0.4, c=0.5)
{Beer, Diaper} → {Milk} (s=0.4, c=0.67)
{Beer, Milk} → {Diaper} (s=0.4, c=0.67)
{Diaper, Milk} → {Beer} (s=0.4, c=0.67)

All non-empty real subsets

{Beer} , {Diaper} , {Milk}, {Beer, Diaper}, {Beer, Milk} , {Diaper, Milk}

Strong rules:

{Beer} → {Diaper, Milk} (s=0.4, c=0.67)
{Beer, Diaper} → {Milk} (s=0.4, c=0.67)
{Beer, Milk} → {Diaper} (s=0.4, c=0.67)
{Diaper, Milk} → {Beer} (s=0.4, c=0.67)

An Example

Transaction ID	Items Bought
2000	A,B,C
1000	A,C
4000	A,D
5000	B,E,F

Min. support 50%
Min. confidence 50%

Frequent Itemset	Support
{A}	75%
{B}	50%
{C}	50%
{A,C}	50%

For rule $A \Rightarrow C$:

support = support($\{A \lambda C\}$) = 50%

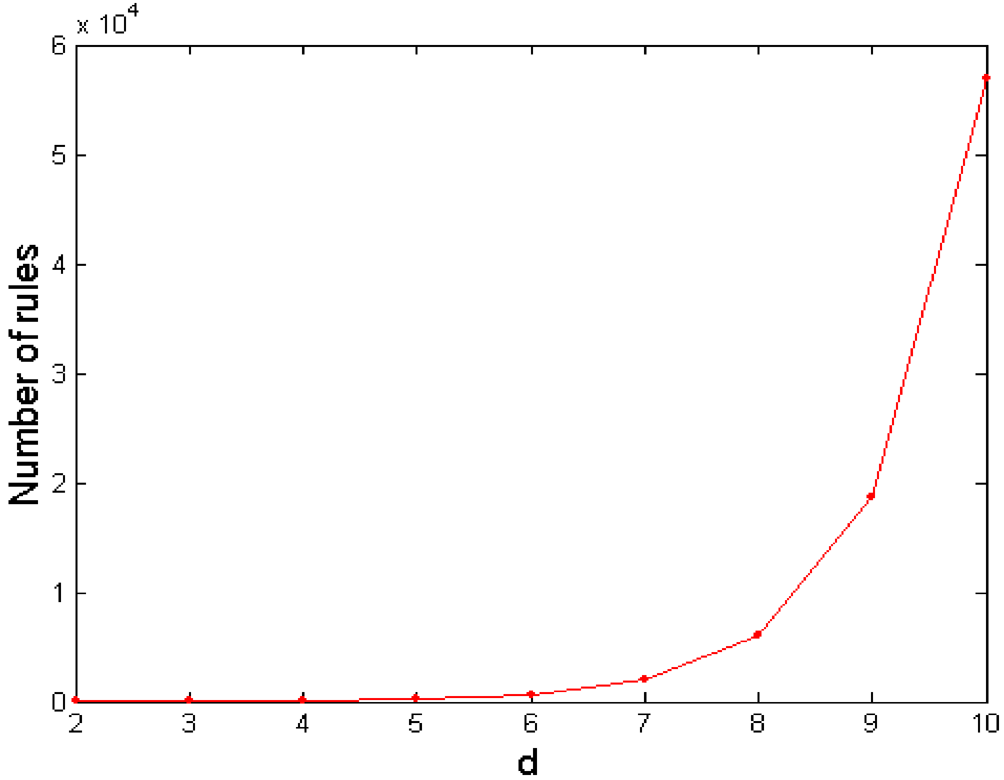
confidence = support($\{A \lambda C\}$)/support($\{A\}$) = 66.6%

The Apriori principle:

Any subset of a frequent itemset must be frequent

Computational Complexity

- Given **d** unique items in ***I***:
  - Total number of itemsets = 2^d
  - Total number of possible association rules:



$$R = \sum_{k=1}^{d-1}\left[\binom{d}{k} \times \sum_{j=1}^{d-k}\binom{d-k}{j}\right]$$

$$= 3^d - 2^{d+1} + 1$$

If d=6, R = 602 rules

Rule Generation – Naive algorithm

- Given a frequent itemset **X**, find all non-empty subsets **y$\subset$ X** such that **y$\rightarrow$ X – y** satisfies the minimum confidence requirement

  – If **{A,B,C,D}** is a frequent itemset, candidate rules:

  ABC $\rightarrow$D, ABD $\rightarrow$C, ACD $\rightarrow$B, BCD $\rightarrow$A,
  A $\rightarrow$BCD, B $\rightarrow$ACD, C $\rightarrow$ABD, D $\rightarrow$ABC
  AB $\rightarrow$CD, AC $\rightarrow$ BD, AD $\rightarrow$ BC, BC $\rightarrow$AD,
  BD $\rightarrow$AC, CD $\rightarrow$AB,

- If **|X| = k**, then there are **2^k – 2** candidate association rules (ignoring **L $\rightarrow$ $\emptyset$** and **$\emptyset$ $\rightarrow$ L**)

Efficient rule generation

- How to efficiently generate rules from frequent itemsets?
  - In general, confidence does not have an anti-monotone property
    c(ABC →D) can be larger or smaller than **c(AB →D)**
  - ***But confidence of rules generated from the same itemset has an anti-monotone property***
  - Example: X = {A,B,C,D}:

$$c(ABC \rightarrow D) \geq c(AB \rightarrow CD) \geq c(A \rightarrow BCD)$$

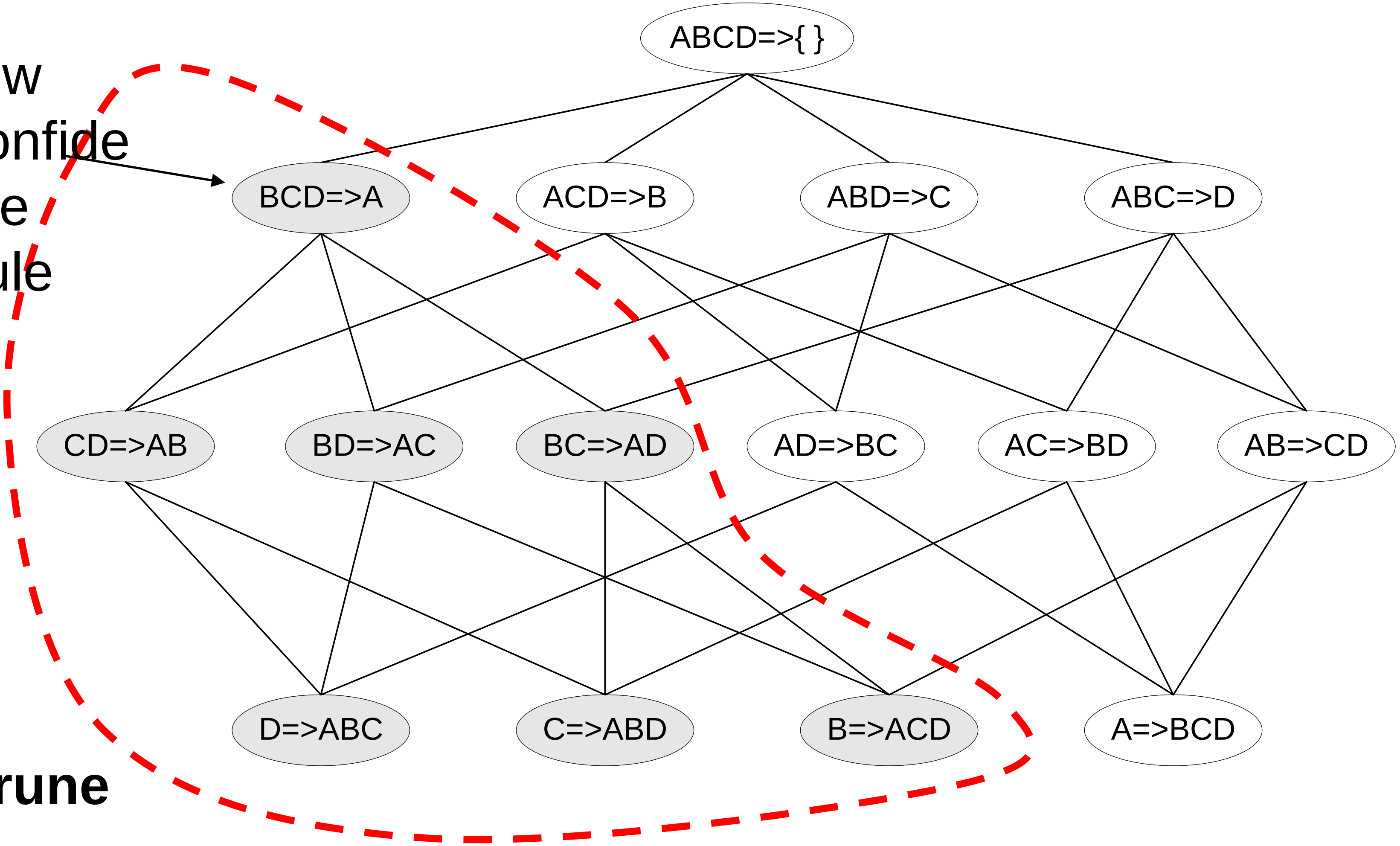
  - **Why?**

Confidence is anti-monotone w.r.t. number of items on the RHS of the rule

Rule Generation for Apriori Algorithm

Lattice of rules

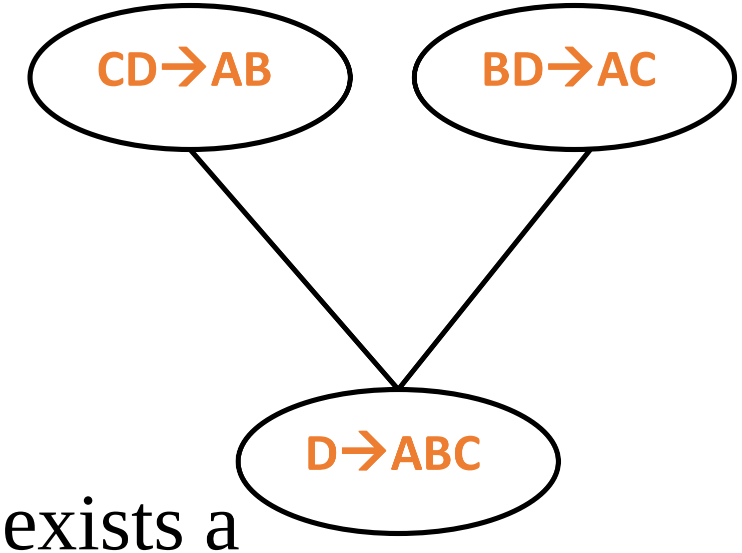
Low Confidence Rule



Pruned Rules

Apriori algorithm for rule generation

- Candidate rule is generated by merging two rules that share the same prefix
in the rule consequent
- **join(CD→AB,BD—>AC)**
would produce the candidate
rule **D →ABC**
- **Prune** rule **D→ABC** if there exists a subset (e.g., **AD→BC**) that does not have high confidence



Challenges of Apriori Algorithm

- Challenges
  - Multiple scans of transaction database
  - Huge number of candidates
  - Tedious workload of support counting for candidates
- Improving Apriori: the general ideas
  - Reduce the number of transaction database scans
  - Shrink the number of candidates
  - Facilitate support counting of candidates

• Imp

Performance Bottlenecks

- The core of the Apriori algorithm:
  - Use frequent (k – 1)-itemsets to generate candidate frequent k-itemsets
  - Use database scan and pattern matching to collect counts for the candidate itemsets
- The bottleneck of *Apriori*: candidate generation
  - Huge candidate sets:
    - 10^4 frequent 1-itemset will generate 10^7 candidate 2-itemsets
    - To discover a frequent pattern of size 100, e.g., $\{a_1, a_2, \ldots, a_{100}\}$, one needs to generate $2^{100} \approx 10^{30}$ candidates.
  - Multiple scans of database:
    - Needs (n +*1*) scans, n is the length of the longest pattern

Factors Affecting Complexity

- Choice of minimum support threshold
  - lowering support threshold results in more frequent itemsets
  - this may increase number of candidates and max length of frequent itemsets
- Dimensionality (number of items) of the data set
  - more space is needed to store support count of each item
  - if number of frequent items also increases, both computation and I/O costs may also increase
- Size of database
  - since Apriori makes multiple passes, run time of algorithm may increase with number of transactions
- Average transaction width
  - transaction width increases with denser data sets
  - This may increase max length of frequent itemsets and traversals of hash tree (number of subsets in a transaction increases with its width)

Using Apriori In class

1-Find the frequent subsets with
s threshold =3

TID	Items
100	ACDFG
200	ABCDF
300	CDE
400	ADF
500	ACDEF
600	BCDEFG

2-From each frequent itemset, generate all confident rules whose confidence ≥ 75%

```
In [ ]: import pandas as pd
        from mlxtend.preprocessing import OnehotTransactions
        from mlxtend.frequent_patterns import apriori
        from mlxtend.frequent_patterns import association_rules

        dataset = [['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']]
```

```
In [ ]: ht_ary = oht.fit(dataset).transform(dataset)
        df = pd.DataFrame(oht_ary, columns=oht.columns_)
        print (df)

        frequent_itemsets = apriori(df, min_support=0.6, use_colnames=True)
        print (frequent_itemsets)
```

```
In [ ]:
        association_rules(frequent_itemsets, metric="confidence", min_threshold=0.7)
        rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.2)
        print (rules)
```

```
In [ ]: support=rules.as_matrix(columns=['support'])
        confidence=rules.as_matrix(columns=['confidence'])
        print(support)
        print(confidence)
```

```
In [ ]: import random
        import matplotlib.pyplot as plt
        for i in range (len(support)):
            support[i] = support[i] + 0.0025 * (random.randint(1,10) - 5)
            confidence[i] = confidence[i] + 0.0025 * (random.randint(1,10) - 5)

        plt.scatter(support, confidence,   alpha=0.5, marker="*")
        plt.xlabel('support')
        plt.ylabel('confidence')
        plt.show()
```