Chair of Process and Data Science

RWTH AACHEN UNIVERSITY

Elements of Machine Learning & Data Science

Frequent Itemsets

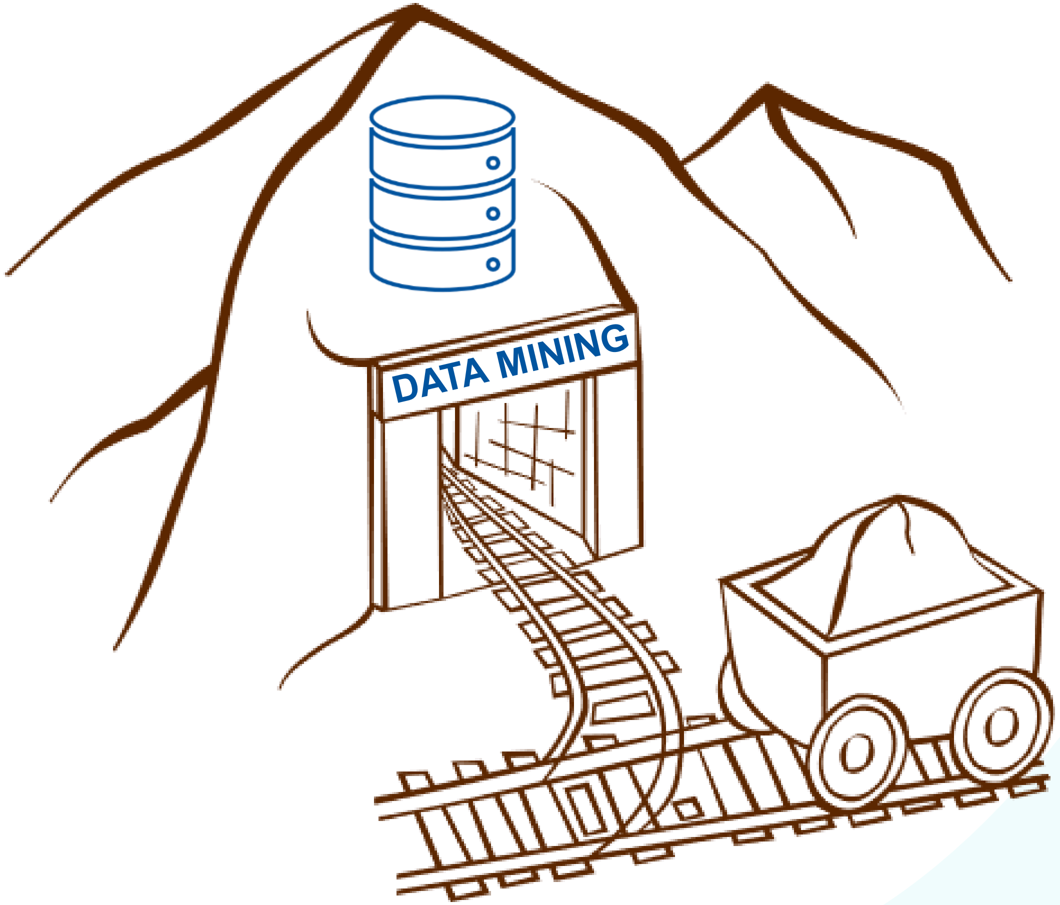
Lecture 9

Prof. Wil van der Aalst

Marco Pegoraro, M.Sc.
Christopher Schwanen, M.Sc.
Tsunghao Huang, M.Sc.

Frequent Itemsets

1. **Introduction**
2. Properties of Frequent Itemsets
3. A-Priori Algorithm
4. FP-Growth Algorithm



Pattern Mining

- Finding surprising patterns in the input data

- Types of patterns:
  - Frequent itemsets
  - Association rules
  - Sequential patterns
  - Partial orders
  - Subgraphs

Itemset Data

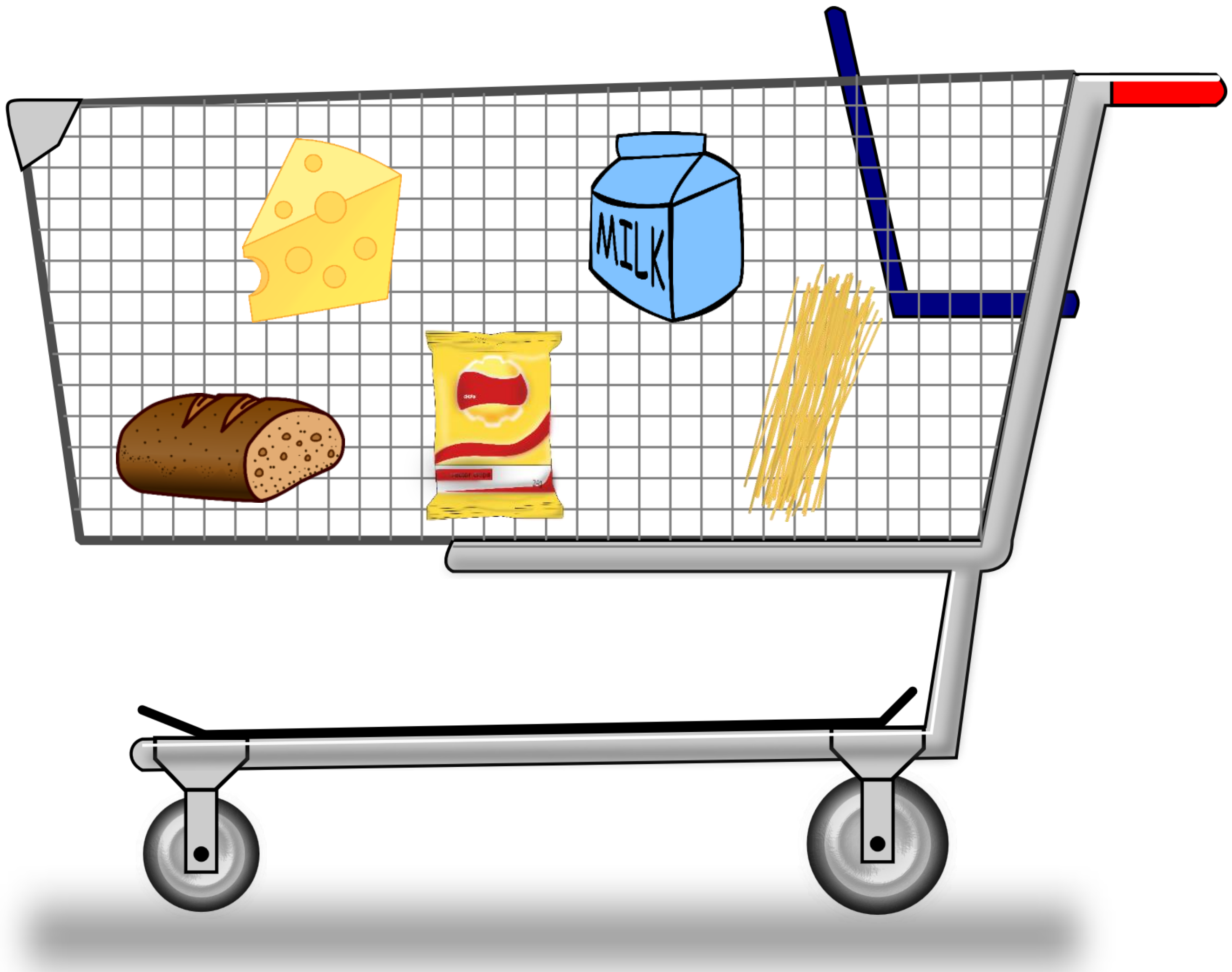
ID	f_1	f_2	f_3	f_4	…	f_D
1						
2						
3						
4						
5						
…						

Each feature refers to an item (e.g., a product, disease, song, course, or error code)

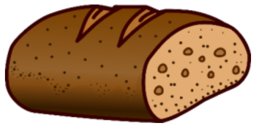
Each cell describes the presence of the item (Boolean 0/1 or Positive Integer Count)

Each instance is a **transaction**

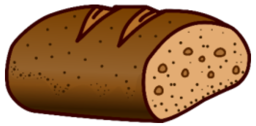
Itemset Data – Example



Itemset Data – Example

ID					…	
1	2	2	0	3		2
2	0	0	1	1		0
3	2	1	0	0		0
4	0	1	0	0		0
5	0	0	0	0		2
…	…	…	…	…		…

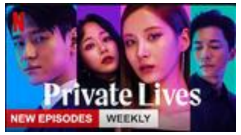
Itemset Data – Example

ID					…	
1	True	True	False	True		True
2	False	False	True	True		False
3	True	True	False	False		False
4	False	True	False	False		False
5	False	False	False	False		True
…	…	…	…	…		…

Other Itemset Data Examples

Rows	Columns
EdX users	Courses taken
Spotify users	Songs Played
Netflix users	Movies Watched
Patients in hospital	Diseases
Repair bills	Components replaced
…	…

Application of Frequent Itemsets

ID					…	
1	True	True	False	True		True
2	False	False	True	True		False
3	True	True	False	True		False
4	False	True	False	True		False
5	False	False	False	False		True
…	…	…	…	…		…

Frequent Itemsets (movies)

NETFLIX

Application of Frequent Itemsets

ID					...	
1	True	True	False	True		True
2	True	True	True	False		False
3	True	True	False	True		False
4	True	False	False	False		False
5	False	False	False	False		True
...	...	...	...			...

Frequent Itemsets

- A notorious success story: the Tesco Clubcard
- Introduced in 1995, it was the first loyalty card with automatic data collection
- Widely regarded as responsible for Tesco's supremacy in the UK
- 1bn£ of increase in sales (4%) in one year
- Today, the Clubcard program is still incredibly profitable, even though Tesco gives away about 1bn£ in rewards and discounts each year!

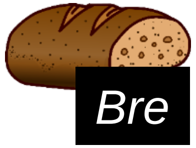
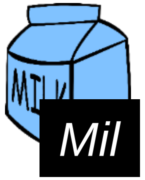
"You know more about my customers after three months than I know after 30 years."

- Lord MacLaurin, chairman for Tesco, talking to the data scientists of the Clubcard program

Frequent Itemsets – Notation

- $\mathcal{I} = \{I_1, I_2, \ldots, I_D\}$ is the set of all possible items
- $\mathcal{A} \subseteq \mathcal{I}$ is an itemset
- A transaction $\mathcal{T}$ is a non-empty itemset
- A dataset $\mathcal{X}$ is a collection of transactions
- Technically $\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$ such that $\emptyset \notin \mathcal{X}$
  ($\mathbb{M}$ is the multiset and $\mathbb{P}$ is the powerset operator)

Frequent Itemsets – Notation Example

ID	 *Che*	 *Bre*	 *Chi*	 *Mil*	...	 *Pas*
1	2 (true)	0 (false)	0 (false)	3 (true)	0 (false)	2 (true)
2	0 (false)	0 (false)	1 (true)	1 (true)	0 (false)	0 (false)
3	2 (true)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)
4	0 (false)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)

$\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$

- Set of all items $\mathcal{I} = \{Che, Bre, Chi, Mil, \ldots, Pas\}$
- Transaction $\mathcal{T}_1 = \{Che, Mil, Pas\} \subseteq \mathcal{I}$
- Dataset with four transactions $\mathcal{X} = [\{Che, Mil, Pas\}, \{Chi, Mil\}, \{Che, Bre\}, \{Bre\}]$
- Dataset with ten transactions $\mathcal{X} = [\{Che, Mil, Pas\}^4, \{Chi, Mil\}^3, \{Che, Bre\}^2, \{Bre\}^1]$

Frequent Itemsets – Notation Generalization

ID	*Che*	*Bre*	*Chi*	*Mil*	…	*Pas*
1	2 (true)	0 (false)	0 (false)	3 (true)	0 (false)	2 (true)
2	0 (false)	0 (false)	1 (true)	1 (true)	0 (false)	0 (false)
3	2 (true)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)
4	0 (false)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)

$\mathcal{X} \in \mathbb{M}(\mathbb{M}(\mathcal{I}))$

- Set of all items $\mathcal{I} = \{Che, Bre, Chi, Mil, \ldots, Pas\}$
- Transaction $\mathcal{T}_1 = [Che^2, Mil^3, Pas^2] \in \mathbb{M}(\mathcal{I})$
- Dataset with four transactions $\mathcal{X} = [[Che^2, Mil^3, Pas^2], [Chi, Mil], [Che^2, Bre], [Bre]]$

We will consider only itemsets that are proper sets (not multisets). However, generalization is trivial.

Frequent Itemsets – Support

$$\text{support}(\mathcal{A}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{A} \subseteq \mathcal{T}]|}{|\mathcal{X}|}$$

(relative)

Fraction of transactions $\mathcal{T}$ in dataset $\mathcal{X}$ that cover the itemset $\mathcal{A}$

$$\text{support\_count}(\mathcal{A}) = |[\mathcal{T} \in \mathcal{X} \mid \mathcal{A} \subseteq \mathcal{T}]|$$

(absolute, also called frequency or count)

Frequent Itemsets – Support

$$\mathrm{support}(\mathcal{A}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{A} \subseteq \mathcal{T}]|}{|\mathcal{X}|}$$

(relative)

Fraction of transactions $\mathcal{T}$ in dataset $\mathcal{X}$ that cover the itemset $\mathcal{A}$

$$\mathrm{support\_count}(\mathcal{A}) = |[\mathcal{T} \in \mathcal{X} \mid \mathcal{A} \subseteq \mathcal{T}]|$$

(absolute, also called frequency or count)

- Minimum **support threshold** $\boldsymbol{min\_sup}$: lower bound for $\boldsymbol{support}(A)$
- An itemset is **frequent** if its support is higher than $\boldsymbol{min\_sup}$
- Frequent itemsets are used to find **association rules**

Support – Example

ID	 *Che*	 *Bre*	 *Chi*	 *Mil*	…	 *Pas*
1	2 (true)	0 (false)	0 (false)	3 (true)	0 (false)	2 (true)
2	0 (false)	0 (false)	1 (true)	1 (true)	0 (false)	0 (false)
3	2 (true)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)
4	1 (true)	1 (true)	0 (false)	1 (true)	0 (false)	0 (false)

$\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$

Dataset $\mathcal{X} = [\{Che, Mil, Pas\}, \{Chi, Mil\}, \{Che, Bre\}, \{Che, Bre, Mil\}]$

Support – Example

ID	 Che	 Bre	 Chi	 Mil	...	 Pas
1	2 (true)	0 (false)	0 (false)	3 (true)	0 (false)	2 (true)
2	0 (false)	0 (false)	1 (true)	1 (true)	0 (false)	0 (false)
3	2 (true)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)
4	1 (true)	1 (true)	0 (false)	1 (true)	0 (false)	0 (false)

$\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$

Dataset $\mathcal{X} = [\{Che, Mil, Pas\}, \{Chi, Mil\}, \{Che, Bre\}, \{Che, Bre, Mil\}]$

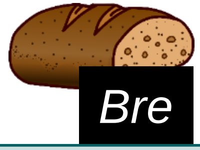
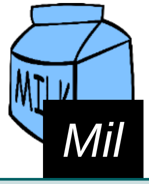
Itemset $\mathcal{A} = \{Che, Mil\} \subseteq \mathcal{I}$

$\text{support\_count}(\mathcal{A}) = |[\mathcal{T} \in \mathcal{X} \mid \mathcal{A} \subseteq \mathcal{T}]| = |[\mathcal{T}_1, \mathcal{T}_4]| = 2$

$\text{support}(\mathcal{A}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{A} \subseteq \mathcal{T}]|}{|\mathcal{X}|} = \frac{|[\mathcal{T}_1, \mathcal{T}_4]|}{4} = \frac{2}{4}$

$\mathcal{A}$ is **frequent** if $\text{min\_sup} \leq 0.5$

Support – Example

ID	 Che	 Bre	 Chi	 Mil	...	 Pas
1	2 (true)	0 (false)	0 (false)	3 (true)	0 (false)	2 (true)
2	0 (false)	0 (false)	1 (true)	1 (true)	0 (false)	0 (false)
3	2 (true)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)
4	1 (true)	1 (true)	0 (false)	1 (true)	0 (false)	0 (false)

$\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$

Dataset $\mathcal{X} = [\{Che, Mil, Pas\}, \{Chi, Mil\}, \{Che, Bre\}, \{Che, Bre, Mil\}]$

Itemset $\mathcal{A} = \{Che, Mil\} \subseteq \mathcal{I}$

$\text{support\_count}(\mathcal{A}) = |[\mathcal{T} \in \mathcal{X} \mid \mathcal{A} \subseteq \mathcal{T}]| = |[\mathcal{T}_1, \mathcal{T}_4]| = 2$

$\text{support}(\mathcal{A}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{A} \subseteq \mathcal{T}]|}{|\mathcal{X}|} = \frac{|[\mathcal{T}_1, \mathcal{T}_4]|}{4} = \frac{2}{4}$

$\mathcal{A}$ is **frequent** if $\text{min\_sup} \leq 0.5$

Itemset $\mathcal{B} = \{Mil\} \subseteq \mathcal{I}$

$\text{support\_count}(\mathcal{B}) = |[\mathcal{T} \in \mathcal{X} \mid \mathcal{A} \subseteq \mathcal{T}]| = |[\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_4]| = 3$

$\text{support}(\mathcal{B}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{B} \subseteq \mathcal{T}]|}{|\mathcal{X}|} = \frac{|[\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_4]|}{4} = \frac{3}{4}$

$\mathcal{B}$ is **frequent** if $\text{min\_sup} \leq 0.75$

Support – Example

ID	 Che	 Bre	 Chi	 Mil	...	 Pas
1	2 (true)	0 (false)	0 (false)	3 (true)	0 (false)	2 (true)
2	0 (false)	0 (false)	1 (true)	1 (true)	0 (false)	0 (false)
3	2 (true)	1 (true)	0 (false)	0 (false)	0 (false)	0 (false)
4	1 (true)	1 (true)	0 (false)	1 (true)	0 (false)	0 (false)

$\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$

Itemset $\mathcal{A} = \{Che, Mil\} \subseteq \mathcal{I}$

$\text{support}(\mathcal{A}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{A} \subseteq \mathcal{T}]|}{|\mathcal{X}|} = \frac{|[\mathcal{T}_1, \mathcal{T}_4]|}{4} = \frac{2}{4}$

Itemset $\mathcal{B} = \{Mil\} \subseteq \mathcal{I}$

$\text{support}(\mathcal{B}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{B} \subseteq \mathcal{T}]|}{|\mathcal{X}|} = \frac{|[\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_4]|}{4} = \frac{3}{4}$

$$\mathcal{B} \subseteq \mathcal{A} \implies \text{support}(\mathcal{B}) \geq \text{support}(\mathcal{A})$$

General rule:
Subset of an itemset has higher or equal support than this itemset

Support – Summary

Support

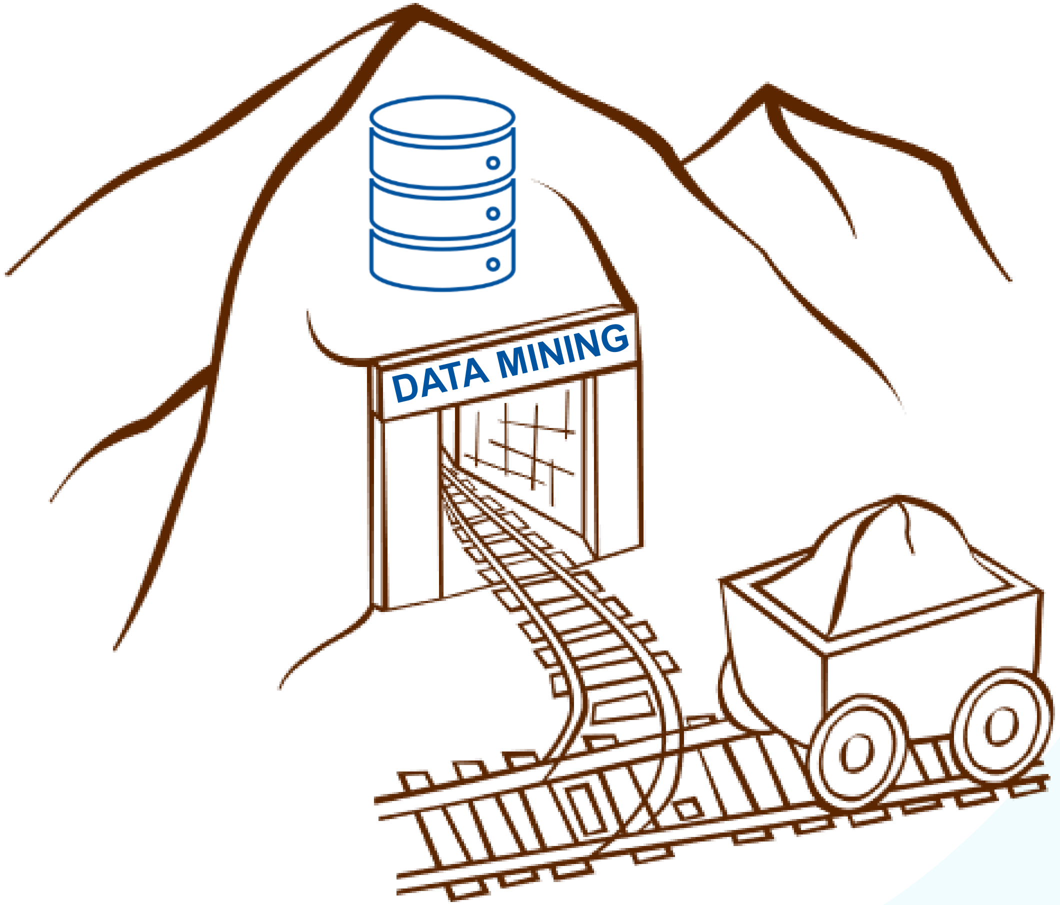
- A measure of the popularity (frequency) of an itemset.
- Calculated as the fraction of transactions in a dataset that contain the itemset.

$$\mathrm{support}(\mathcal{A}) = \frac{|[\mathcal{T} \in \mathcal{X} | \mathcal{A} \subseteq \mathcal{T}]|}{|\mathcal{X}|}$$

- Any itemset with a support below the threshold is considered to be infrequent.
- Support is also used to find association rules

Frequent Itemsets

1. Introduction
2. **Properties of Frequent Itemsets**
3. A-Priori Algorithm
4. FP-Growth Algorithm



Problem Statement

Given dataset $\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$ and minimum support threshold $\boldsymbol{min\_sup}$,
find all frequent non-empty itemsets:

$$\{\mathcal{A} \subseteq \mathcal{I} \mid \mathrm{support}(\mathcal{A}) \geq \mathrm{min\_sup}\}$$

Naïve Approach

- Given $\mathcal{A} \subseteq \mathcal{I}$, it is possible to check whether $\boldsymbol{support}(\mathcal{A}) \geq \boldsymbol{min\_sup}$ by testing all transactions
- If there are $\boldsymbol{D}$ unique items, then there are $\mathbf{2}^{\boldsymbol{D}} - \mathbf{1}$ candidate itemsets that can all be tested individually
- However, this can be very time consuming…

Assume $D = 50\,000$ products

$\mathbf{2^D - 1 =}$

3,160,699,436,856,317,896,135,924,659,945,691,788,984,676,387,834,935,666,847,743,155,564,943,937,902,009,550,651,067,144,922,529,420,974,282,690,343,798,061,622,891,650,247,060,091,533,595,130,170,365,868,108,099,970,116,531,087,467,047,583,722,093,787,639,674,649,765,662,074,366,466,883,324,927,932,743,926,222,226,256,325,646,619,479,597,070,853,065,410,126,319,556,64
5,095,487,584,255,731,625,229,939,513,738,335,892,649,026,005,867,435,951,184,963,615,454,162,198,009,836,540,553,902,746,189,836,926,616,803,054,602,986,713,771,300,764,236,103,912,584,316,984,387,046,421,891,042,940,862,409,285,760,387,607,642,178,660,447,084,278,582,343,741,975,788,782,575,397,255,679,835,385,509,255,617,699,530,378,773,285,612,149,746,612,966,192,65
1,253,693,114,313,273,858,994,605,409,771,445,163,688,490,715,356,137,720,041,811,636,018,280,718,733,780,759,203,810,695,576,005,864,810,237,900,789,639,882,550,370,233,732,760,551,157,423,186,221,777,437,962,235,113,685,057,004,875,812,625,837,741,639,074,014,635,869,254,467,870,669,671,658,811,749,944,572,950,947,120,522,955,828,413,262,134,950,123,343,450,944,893,38
1,086,445,704,327,516,608,478,853,295,735,214,794,279,795,422,828,861,127,478,241,396,081,419,738,759,639,852,247,634,596,698,774,287,547,984,270,270,829,223,110,217,158,518,692,621,481,994,384,293,987,162,186,350,943,113,710,209,892,772,971,716,110,929,927,731,636,487,052,992,987,749,952,582,075,199,999,823,623,336,378,201,206,536,182,409,135,673,260,858,434,397,680,32
4,294,431,695,532,313,988,094,705,825,127,170,281,671,342,756,132,130,834,630,626,735,161,518,877,001,479,353,099,675,421,568,669,161,387,971,545,385,350,702,911,535,042,214,105,270,064,184,049,195,321,693,402,645,502,131,017,185,839,794,236,528,392,026,118,916,801,534,722,139,870,222,746,741,395,902,492,128,586,493,084,122,499,921,428,423,167,659,386,991,199,053,145,44
0,232,918,025,337,294,792,121,480,817,320,557,424,817,134,341,166,560,131,328,380,547,089,162,451,960,863,019,896,841,277,608,267,473,071,032,610,361,884,348,741,832,609,100,033,764,267,372,951,754,668,207,732,140,976,344,881,559,366,912,679,236,124,697,823,644,196,584,984,332,119,466,217,601,083,183,663,311,742,410,337,555,621,104,734,292,489,903,235,192,160,821,330,89
8,029,535,914,395,120,861,478,951,157,725,966,695,019,256,259,017,910,724,870,747,305,048,741,631,683,422,158,829,592,626,998,675,935,495,968,355,706,034,794,377,249,704,826,237,385,353,428,041,477,614,977,159,200,366,168,819,484,882,369,473,577,673,820,234,304,059,320,536,368,181,780,663,852,770,156,176,045,997,023,854,188,210,931,505,764,603,497,418,111,116,833,045,59
4,683,460,761,810,060,347,938,666,107,376,407,376,538,161,433,794,082,201,152,452,755,038,680,915,248,667,576,037,816,448,107,114,928,454,957,358,973,769,979,101,775,081,280,563,858,425,659,985,551,853,002,945,198,136,847,407,803,098,731,016,476,002,188,721,167,044,725,016,609,620,417,517,264,461,388,623,721,099,795,827,561,842,233,211,027,265,707,652,869,227,549,971,36
0,421,138,523,837,935,337,610,829,930,530,940,245,818,717,397,767,877,474,556,856,115,648,381,712,850,949,057,429,834,689,786,923,900,238,199,668,462,420,124,975,014,460,176,937,757,543,706,037,088,013,809,808,763,009,055,350,430,278,490,509,069,412,005,546,309,455,095,193,839,024,869,890,529,164,588,776,426,256,822,355,723,140,640,821,450,894,163,789,455,395,044,742,35
8,602,248,211,119,402,554,274,364,607,815,242,974,440,790,954,476,462,235,421,343,042,061,888,188,431,752,632,527,921,982,019,539,829,676,482,175,124,356,602,558,024,358,281,709,710,521,718,828,251,944,407,816,772,714,544,977,673,460,983,558,935,168,237,819,654,683,418,376,457,904,080,539,202,530,627,669,869,881,215,398,947,957,922,545,005,739,842,763,519,814,852,704,76
5,560,828,463,466,036,149,792,281,182,326,943,206,886,419,926,850,135,063,162,179,241,871,274,796,024,802,742,686,225,024,907,605,632,447,471,332,642,008,409,940,104,913,983,385,874,283,001,746,786,647,879,747,994,089,213,330,239,015,474,018,013,137,685,271,043,254,385,416,804,411,944,936,017,477,858,365,910,618,800,586,138,894,786,098,262,095,903,664,034,479,708,143,59
1,509,590,289,319,757,152,692,646,380,447,474,820,059,221,354,059,445,093,575,442,716,973,331,098,176,716,707,848,689,462,376,523,334,556,192,193,952,316,116,218,228,442,266,069,078,290,033,668,168,882,789,937,930,106,606,528,021,679,497,563,218,701,433,041,385,362,553,384,155,785,144,110,504,433,121,892,992,899,207,988,096,988,077,083,065,532,889,768,442,011,243,43
6,599,859,452,840,889,686,495,279,989,903,477,096,078,575,541,951,328,806,322,851,950,252,552,592,344,168,743,316,523,924,091,677,776,555,035,300,546,896,564,711,246,599,096,501,538,015,732,460,568,505,159,309,026,876,942,822,257,942,805,338,777,947,949,083,385,366,853,887,931,563,547,342,972,750,038,735,441,012,362,088,807,826,002,678,323,209,075,341,664,682,039,970,24
4,371,752,439,627,477,589,922,704,036,287,386,433,079,400,082,357,329,563,904,616,267,987,268,587,201,847,267,650,982,070,274,204,118,716,145,673,577,347,087,894,405,829,670,915,892,444,598,008,311,679,006,927,556,585,863,235,062,177,393,323,753,925,769,530,080,667,314,748,109,769,340,705,616,829,158,713,742,366,956,323,811,658,939,980,376,490,362,206,536,509,163,262,89
5,994,035,528,548,104,714,311,529,907,097,590,979,874,192,495,481,909,859,345,140,169,036,819,397,692,571,708,295,085,608,035,848,878,511,103,780,211,394,150,671,490,415,999,523,917,903,105,532,567,495,107,879,712,588,387,210,084,586,358,044,694,457,753,354,516,325,347,437,062,681,260,366,814,109,779,422,050,191,032,581,851,452,046,060,881,359,376,620,137,966,167,740,23
9,764,843,152,438,225,370,822,153,538,376,483,436,617,110,334,960,092,927,481,817,805,369,608,748,651,839,786,120,991,198,073,428,704,485,016,027,868,995,877,751,620,428,289,400,641,132,459,288,145,560,902,751,347,098,644,695,563,477,637,451,093,696,302,793,057,489,201,058,088,342,606,342,922,981,190,034,432,698,490,915,886,271,311,961,857,652,026,361,053,628,705,171,60
0,303,150,123,538,685,001,308,779,479,681,857,312,254,943,518,449,704,275,479,642,437,275,140,137,984,669,941,944,836,642,040,223,878,486,538,067,789,693,348,083,730,346,825,273,845,460,061,706,852,290,775,466,985,780,968,592,555,948,972,413,863,120,063,221,143,185,629,834,828,284,332,040,441,845,171,583,418,684,231,660,145,392,992,409,922,598,320,558,427,635,546,169,24
0,259,499,303,896,687,655,889,853,235,173,231,563,594,894,331,355,085,387,107,697,230,905,654,705,259,068,621,648,397,387,329,097,578,047,593,092,846,447,686,613,216,298,865,047,063,582,525,798,691,383,901,918,628,632,995,284,123,553,924,531,341,699,613,651,503,475,557,720,432,902,819,498,996,966,702,546,551,722,665,864,293,765,800,457,458,150,514,318,283,415,015,125,84
4,481,860,456,154,735,333,547,800,438,057,205,778,685,809,314,280,011,156,961,657,480,785,672,481,976,603,269,568,321,288,690,659,192,303,413,177,475,131,231,965,528,530,942,225,372,500,539,202,363,473,718,122,984,588,259,769,774,091,345,713,910,425,974,018,911,236,180,570,088,580,544,088,658,248,204,579,751,148,634,022,325,873,940,217,022,225,699,314,067,337,213,094,08
6,024,904,721,125,159,205,464,697,477,837,951,501,543,061,510,253,460,227,330,486,801,150,033,856,688,083,604,379,[illegible]564,999,371,994,208,371,160,152,723,047,097,310,816,[illegible]441,163,452,[illegible]069,854,700,418,024,650,230,525,9[illegible]19,301,515,72[illegible]567,531,657,567,580,163,596,380,227,833,565,239,370,333,047,582,063,688,793,679,363,684,486,168,061,587,575,930,547,105,424,04
0,975,094,918,638,314,255,917,502,944,267,332,343,175,376,082,638,428,085,697,931,895,281,841,412,624,618,678,[illegible]829,064,[illegible]827,990,052,308,716,078,511,409,520,429,551,751,0[illegible]805,106,260,[illegible]324,886,752,157,711,131,176,942,09[illegible]75,766,121,1[illegible]24,754,507,114,500,952,202,347,645,155,880,155,885,278,014,954,372,029,516,401,063,040,466,885,470,145,534,802,886,825,482,77
3,843,551,988,621,361,739,581,412,576,048,062,774,658,251,424,538,120,236,005,887,704,051,240,777,805,903,13[illegible]1,154,130,098,955,699,[illegible]759,909,5[illegible]071,859,6[illegible]202,737,485,[illegible]88[illegible]64,561,2[illegible]218,9[illegible]879,16[illegible]38,363[illegible]0[illegible]802,354,991,176,417,279,141,163,114,186,466,600,052,094,711,863,705,684,900,098,579,523,355,710,253,485,631,035,686,550,033,97
4,037,903,520,541,402,450,242,719,240,797,326,025,484,993,704,399,679,794,871,910,927,023,664,828,239,225,17[illegible]572,246,226,765,050,6[illegible]233,552[illegible]9,66[illegible]91,901,[illegible]458,[illegible]29,069,[illegible]662,215,341,[illegible]0,61[illegible]20,483,[illegible]457,[illegible]0,124,[illegible]01,[illegible]22,[illegible]211,64[illegible]494,149,867,849,258,369,271,568,806,893,320,386,405,191,564,343,234,607,244,674,387,924,106,898,339,326,025,526,674,659,009,78
4,143,464,823,162,427,053,006,177,894,952,451,951,244,958,914,648,418,596,424,889,789,008,679,263,496,096,28[illegible]78,850,[illegible]4,1[illegible]514,767,[illegible]91,[illegible]21,389,[illegible]27[illegible]47,705,7[illegible]009,882,939,[illegible]06[illegible]41,476,9[illegible]34[illegible]05,500,574,33[illegible]99,819,38[illegible]16,844,803,259,941,917,998,483,757,110,114,029,135,092,405,018,779,285,891,809,345,880,290,584,450,371,392,611,286,191,050,79
2,529,551,802,851,823,990,309,632,705,346,023,254,826,875,476,129,983,877,945,558,650,786,101,744,970,453,29[illegible]64,067,487,078[illegible]3,9[illegible]396,706,7[illegible]46[illegible]098,573,1[illegible]12[illegible]82,116,8[illegible]648,576,088,[illegible]1,25[illegible]34,151,0[illegible]15[illegible]47,756,385,37[illegible]0,008,89[illegible]443,649,234,393,079,979,131,039,365,729,356,796,938,706,763,568,193,651,332,269,863,240,182,237,146,533,295,405,342,240,355,24
2,109,979,044,395,569,527,874,520,774,012,788,166,968,677,276,493,457,207,047,533,055,279,088,933,682,692,014,[illegible]040,504,82[illegible]3,7[illegible]58,961,[illegible]4,24[illegible]5,308,[illegible]19[illegible]0,190,[illegible]789,613,998,[illegible]39[illegible]0,414,[illegible]254[illegible]5,465,[illegible]31[illegible]94,1[illegible]752,896,493,772,038,215,056,824,512,514,878,469,301,317,017,063,082,134,512,404,070,211,623,281,097,074,165,789,178,376,568,103,254,80
0,488,918,480,236,589,450,225,691,849,471,266,440,724,147,764,862,569,534,948,241,458,843,759,034,565,370,282,54[illegible]25,689,180,[illegible]692,190,0[illegible]63,260,[illegible]446,021,890,[illegible]1,085,[illegible]037,0[illegible]46,96[illegible]52,255[illegible]10[illegible]03,242,852,550,678,733,531,247,684,088,423,360,784,794,424,939,667,547,678,023,939,008,044,711,347,695,593,952,595,645,764,20
0,085,119,099,979,544,055,147,487,323,341,215,334,555,709,196,780,139,933,682,500,741,839,812,932,156,351,120,796,247,468,214,657,244,618,195,095,030,506,657,808,740,291,107,993,818,182,376,397,882,737,632,741,745,747,195,351,545,715,871,008,258,681,361,684,966,548,660,619,181,449,534,458,489,173,580,647,791,296,269,246,341,186,055,766,932,761,559,150,521,309,231,891,73
8,476,118,424,685,199,962,976,418,495,044,193,590,453,686,856,782,078,504,306,975,602,130,322,378,117,418,749,641,128,299,141,724,823,021,053,565,651,713,404,997,730,697,438,119,519,620,081,631,942,450,323,286,536,535,509,426,261,805,775,933,903,494,172,951,093,268,252,243,461,370,693,657,946,027,197,493,080,914,272,267,737,573,180,566,941,791,396,738,773,682,690,483,99
8,697,262,776,008,005,259,823,852,668,363,210,327,895,409,895,710,606,975,967,654,501,181,761,916,387,719,452,260,299,629,849,433,609,758,473,948,413,349,455,194,564,403,806,274,966,798,391,602,464,531,940,912,243,207,581,482,784,616,291,015,255,687,740,576,936,388,024,757,675,016,888,156,274,288,245,856,121,912,221,021,943,453,864,780,323,759,118,283,507,272,635,996,50
5,745,462,224,497,094,163,590,997,074,700,818,017,606,283,851,984,670,403,315,773,016,215,268,600,152,326,505,929,064,680,061,985,889,686,166,593,526,608,260,345,504,672,816,451,020,882,755,976,378,512,567,192,578,233,745,494,174,487,961,334,187,784,916,308,168,713,658,989,300,284,134,750,439,153,982,921,779,323,332,543,834,027,743,015,241,922,957,054,075,061,365,948,32
6,552,811,778,580,199,775,305,520,304,113,656,699,963,224,042,243,414,282,607,459,803,128,399,687,495,472,309,439,378,003,887,795,387,400,170,190,820,290,583,561,307,179,115,229,188,565,097,204,042,602,940,124,326,205,683,789,470,721,022,643,993,908,246,960,833,228,077,215,237,703,927,490,179,886,175,843,896,484,778,643,410,817,269,690,317,213,721,645,512,394,083,995,44
7,392,760,535,159,680,365,360,118,842,021,109,089,391,140,333,773,312,517,258,060,363,877,298,255,941,817,162,718,561,438,303,458,401,089,912,150,516,256,341,442,449,652,054,770,023,414,946,313,724,050,167,525,187,041,208,646,911,228,350,990,403,368,199,576,064,233,819,946,029,869,949,279,451,686,312,076,828,179,937,708,739,936,387,134,174,085,385,333,643,463,257,874,58
0,597,240,792,690,571,190,641,405,631,195,239,721,208,985,551,093,571,538,370,792,573,232,757,096,258,466,454,855,642,311,617,374,673,942,354,602,528,830,612,341,852,932,686,815,306,450,185,341,484,849,093,491,762,837,620,237,386,842,287,913,381,520,626,080,117,262,315,168,261,863,461,661,340,918,705,577,375,417,670,023,366,454,025,183,582,024,337,301,164,714,138,474,62
4,112,398,744,475,799,226,379,647,228,101,617,058,084,705,988,475,800,655,053,094,825,820,504,783,647,736,671,118,625,783,709,327,991,030,041,227,664,594,263,923,246,692,075,724,835,347,919,569,762,240,576,413,561,949,173,457,011,336,100,957,011,363,477,637,411,765,767,203,240,425,103,928,180,229,797,422,209,486,261,121,508,346,184,699,801,431,283,827,835,411,003,800,85
7,328,920,405,6[illegible]759,72[illegible]5,238,599,898,679,291,911,988,037,144,249,012,939,408,701,938,679,052,231,893,846,650,292,083,009,022,858,162,0[illegible]0,527,628,730,034,80[illegible]119,82[illegible]073[illegible]45,480,621,201,400,108,131,349,518,751,111,706,142,512,239,976,281,395,176,137,249,051,218,501,377,9[illegible]2,897,652,779,649,236,179,128,863,610,683,268,961,841,620,291,117,325,813,307,342,243,86
2,125,126,394[illegible]71,504,8[illegible]3[illegible]9,028,90[illegible]081,4[illegible]61,94[illegible]515,0[illegible]65[illegible]3,1[illegible]425,3[illegible]740,464,694,8[illegible]074,4[illegible]086,[illegible]900,[illegible]282,635,2[illegible]7,21[illegible]010,2[illegible]454,[illegible]660,618,4[illegible]951,[illegible]486,[illegible]186,096,09[illegible]9,261,[illegible]090,617,613,837,956,389,467,907,896,797,311,184,827,619,479,537,682,406,670,665,041,15
0,801,077,40[illegible]778,622,85[illegible]242,530,744,8[illegible]158,557[illegible]29,65[illegible]752,[illegible]5,[illegible]184,[illegible]847,[illegible]1,629,5[illegible]933,8[illegible]064,475,[illegible]37,1[illegible]143[illegible]14[illegible]10,31[illegible]87[illegible]352,336,7[illegible]7,828,471[illegible]06,03[illegible]504,[illegible]08,524,06[illegible]049[illegible]34,40[illegible]85[illegible]314,6[illegible]1[illegible]706,[illegible]10,1[illegible]949,735,053,09[illegible]523[illegible]15,2[illegible]261,5[illegible]8,361,08[illegible]622,5[illegible]0,[illegible]646,0[illegible]7,590,496,303,679,729,545,203,050,908,730,605,290,907,810,258,967,044,283,091,528,451,89
5,818,870,34[illegible]00,587,5[illegible]3[illegible]24,658,8[illegible]7,206,47[illegible]22,69[illegible]820,5[illegible]79,974,924,0[illegible]95[illegible]922,853,364,79[illegible]505,8[illegible]2,982[illegible]50[illegible]44,86[illegible]596,93[illegible]1,[illegible]4,150,563[illegible]50,58[illegible]059[illegible]69,170,4[illegible]968[illegible]53,51[illegible]22[illegible]933,88[illegible]539,[illegible]7,93[illegible]270,687[illegible]20,27[illegible]995[illegible]71,3[illegible]193,937,679,49[illegible]128,8[illegible]09[illegible]769,[illegible]5,9[illegible]8,354,112,403,200,778,056,635,299,274,033,456,320,537,498,356,054,366,136,573,398,71
5,082,620,546,4[illegible]186[illegible]2,305[illegible]792,690,73[illegible]229[illegible]00,[illegible]5,45[illegible]639[illegible]17,[illegible]70[illegible]639,[illegible]7,67[illegible]585[illegible]82,[illegible]3,645,177[illegible]10,4[illegible]3,301,[illegible]45,[illegible]9,69[illegible]32,[illegible]936,[illegible]5,5[illegible]170,369,[illegible]4,87[illegible]070[illegible]09,603,53[illegible]978[illegible]3,93[illegible]120[illegible]28,0[illegible]3,31[illegible]387,[illegible]8,05[illegible]616,339[illegible]76,6[illegible]188[illegible]00,1[illegible]7,978,16[illegible]409,2[illegible]0,91[illegible]03[illegible]55,3[illegible]562,505,363,147,926,142,387,716,810,598,488,508,143,536,876,145,510,573,301,401,934,62
6,741,183,440,087,486,797,914,444,089,643,836,461,081,374,200,075,670,768,923,426,371,348,078,887,063,230,377,992,083,702,521,930,485,045,630,111,341,754,744,795,119,913,700,945,053,214,862,237,119,548,160,083,718,839,166,153,809,005,875,205,409,810,444,764,595,305,207,226,204,919,916,851,026,595,808,781,071,177,399,516,703,007,441,268,128,388,251,027,806,978,756,780,96
5,863,282,079,114,981,098,532,902,812,630,569,600,865,339,409,792,542,901,056,179,018,773,276,890,596,101,452,759,462,289,523,165,998,928,649,697,014,127,541,189,050,489,365,806,221,059,002,546,651,669,413,543,808,486,226,831,386,921,663,452,816,224,122,639,817,917,125,581,419,125,143,616,542,010,433,801,514,207,135,317,250,107,335,576,675,600,961,875,533,373,968,231,97
7,609,530,585,561,723,093,869,718,973,501,662,405,376,134,561,821,455,699,409,638,672,137,368,629,934,434,904,135,243,616,290,939,496,392,213,529,132,465,038,070,381,597,121,554,593,301,891,351,813,831,284,779,390,595,784,060,424,586,912,828,116,601,132,494,255,062,973,159,800,567,641,836,498,957,014,054,942,954,711,894,710,082,738,273,171,530,521,637,467,348,697,452,46
6,979,485,951,0[illegible]7,251,467,882,839,811,425,221,840,317,955,114,236,128,886,094,171,414,113,610,394,646,355,770,173,692,127,111,713,323,245,274,928,734,486,777,208,288,057,391,240,912,508,657,708,984,431,709,954,063,118,992,225,204,215,214,782,260,900,326,628,193,205,980,828,392,377,722,350,075,636,649,265,242,636,777,301,520,038,207,771,647,186,571,232,202,354,949,299,30
9,536,851,491[illegible]55,187,903,038,387,627,856,713,703,659,184,219,746,396,337,480,40[illegible]616,195,395,474,061,986,947,525,036,188,279,119,416,469,635,112,795,244,440,267,844,485,260,032,847,241,459,514,923,450,833,935,808,678,103,311,534,444,263,158,100,167,092,569,520,823,729,697,095,097,220,555,929,207,430,809,463,272,459,605,535,199,920,221,147,327,782,617,721,315,661,289,01
4,745,577,69[illegible]73,6[illegible]996[illegible]0,1[illegible]03[illegible]742,[illegible]9,6[illegible]2,040,[illegible]31,[illegible]3,67[illegible]847[illegible]27,1[illegible]578,596,290,273,945,780,131,899,615,049,297,151,882,080,212,613,618,218,835,189,615,546,146,490,618,786,229,788,621,159,478,409,062,855,522,518,401,969,800,317,868,267,878,459,364,052,275,698,949,560,834,684,003,729,553,507,462,896,883,801,001,785,393,523,528,693,189,808,998,634,228,530,098,11
2,820,657,163[illegible]19,6[illegible]6,32[illegible]6[illegible]9,031,9[illegible]5,2[illegible]2,118,[illegible]19,[illegible]698,[illegible]98,[illegible]63,89[illegible]23[illegible]551,901,772,401,695,505,788,408,841,293,397,249,025,198,635,937,668,398,411,035,193,162,459,824,589,919,298,447,926,993,308,316,131,770,771,810,396,869,544,550,114,424,232,839,264,186,286,312,815,374,965,713,953,477,456,197,606,082,891,701,044,812,561,651,660,31
9,138,187,936[illegible]72,0[illegible]7,50[illegible]6[illegible]99,754,26[illegible]527,[illegible]7,8[illegible]2,174[illegible]61,[illegible]0,767,366[illegible]10,10[illegible]938[illegible]150,863,634,759,902,606,141,578,632,188,751,943,281,741,709,639,780,096,523,879,596,418,113,007,481,292,983,895,437,069,705,044,558,380,617,028,625,715,666,636,690,157,525,182,398,256,117,199,445,403,241,023,559,015,298,807,601,851,330,547,178,795,577,843,149,795,933,566,060,376,644,011,17
2,155,058,399,062,530,769,179,375,903,286,1[illegible]9,027,612,075,806,315,776,377,707,195,825,601,868,759,450,088,050,733,579,314,903,189,062,524,181,137,056,132,917,203,202,803,431,388,444,967,344,457,607,414,912,712,286,136,586,662,334,942,533,483,472,775,769,916,860,739,822,410,605,139,889,887,622,933,231,942,596,545,836,342,388,849,977,051,318,874,585,752,685,553,439,222,48
0,788,285,263,696,111,065,583,479,158,521,6[illegible]5,599,011,752,852,282,847,768,809,742,787,921,241,434,089,317,310,409,167,342,417,736,279,172,947,292,772,969,327,153,366,897,236,428,973,617,809,658,910,838,076,604,238,454,065,661,670,923,252,421,041,624,103,843,924,047,529,979,248,232,963,083,731,950,440,348,856,678,553,094,106,548,043,172,702,289,199,899,275,318,362,905,37
8,408,712,415,159,269,923,344,094,958,681,016,353,398,306,725,904,407,358,220,769,186,194,606,870,015,032,484,591,003,096,383,698,104,591,712,625,702,867,260,453,854,415,173,761,377,387,219,612,874,764,117,226,958,751,591,406,925,850,280,213,167,967,358,855,839,013,668,351,812,162,347,016,911,885,289,247,085,576,542,162,727,707,634,043,698,542,226,997,665,877,092,869,91
2,956,662,095,277,615,147,881,880,580,839,497,663,044,456,735,668,074,084,606,055,509,121,324,098,162,577,700,495,623,250,626,576,140,392,118,793,726,598,300,895,151,612,936,669,369,981,218,841,802,508,569,563,506,749,092,758,361,711,504,078,242,899,013,061,396,198,412,853,703,118,010,073,765,571,276,002,967,623,291,654,506,202,798,544,591,136,168,371,274,314,046,289,06
2,299,081,496,153,033,412,112,641,387,870,580,908,582,351,970,894,304,880,328,771,879,450,411,873,869,041,904,178,677,230,778,206,368,284,438,333,429,760,436,040,150,912,799,518,037,357,212,354,374,733,844,905,651,600,813,021,957,809,144,377,631,479,571,636,638,095,648,273,971,176,393,911,636,202,338,522,338,349,855,986,239,822,559,626,678,516,947,067,545,491,059,640,36
3,164,772,253,282,891,079,994,101,376,152,087,416,182,773,952,840,881,445,035,913,235,822,373,550,516,126,262,884,445,234,405,811,592,466,055,571,627,450,168,569,925,389,721,039,944,218,176,868,921,181,944,425,304,096,132,312,721,538,680,757,455,121,594,473,310,802,579,385,393,574,059,209,602,443,968,201,967,760,613,102,289,456,131,085,235,835,109,375

Good luck!

Of course most of them are not frequent…

Subsets of Frequent Itemsets Are Also Frequent

- Assume $\mathcal{A} = \{I_1, I_2, \ldots, I_{100}\}$ and $\boldsymbol{support}(\mathcal{A}) \geq \boldsymbol{min\_sup}$
- All subsets of $\mathcal{A}$ are also frequent
- There are $\binom{\mathbf{100}}{\mathbf{1}} = \mathbf{100}$ frequent itemsets having 1 item
- There are $\binom{\mathbf{100}}{\boldsymbol{k}}$ frequent itemsets having $\boldsymbol{k}$ items ("100 choose k")
- There are $\binom{\mathbf{100}}{\mathbf{1}} + \binom{\mathbf{100}}{\mathbf{2}} + \cdots \binom{\mathbf{100}}{\mathbf{99}} = \mathbf{2^{100} - 2 = 1.27} \times \mathbf{10^{30}}$ smaller frequent itemsets contained in $\mathcal{A}$

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n(n-1)\ldots(n-k+1)}{k(k-1)\ldots1}$$

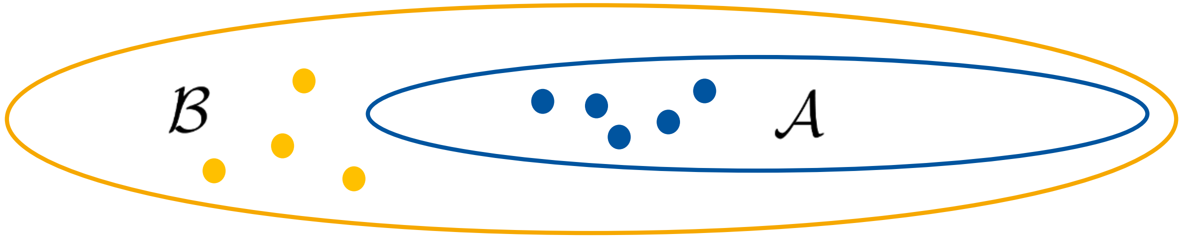
Summary

- We should avoid exhaustively testing all candidate itemsets
- We need to focus on the “interesting” ones

→ Closed itemsets

Closed Itemsets

- An itemset $\mathcal{A}$ is closed if there is no proper superset $\mathcal{B} \supset \mathcal{A}$ that has the same support
- If $\mathcal{A}$ is closed, then $\boldsymbol{support}(\mathcal{A}) > \boldsymbol{support}(\mathcal{B})$ for any $\mathcal{B} \supset \mathcal{A}$

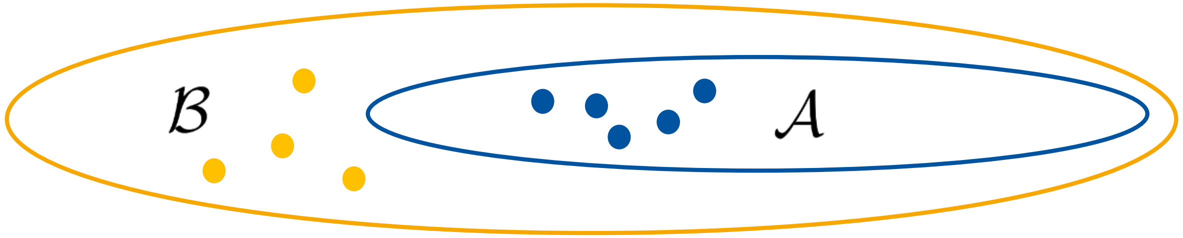


adding any item to $\mathcal{A}$ will always **reduce support**

Closed Frequent Itemsets

- An itemset $\mathcal{A}$ is closed if there is no proper superset $\mathcal{B} \supset \mathcal{A}$ that has the same support
- If $\mathcal{A}$ is closed, then $\boldsymbol{support}(\mathcal{A}) > \boldsymbol{support}(\mathcal{B})$ for any $\mathcal{B} \supset \mathcal{A}$

- $\mathcal{A}$ is frequent if its support is higher than threshold $\boldsymbol{min\_sup}$

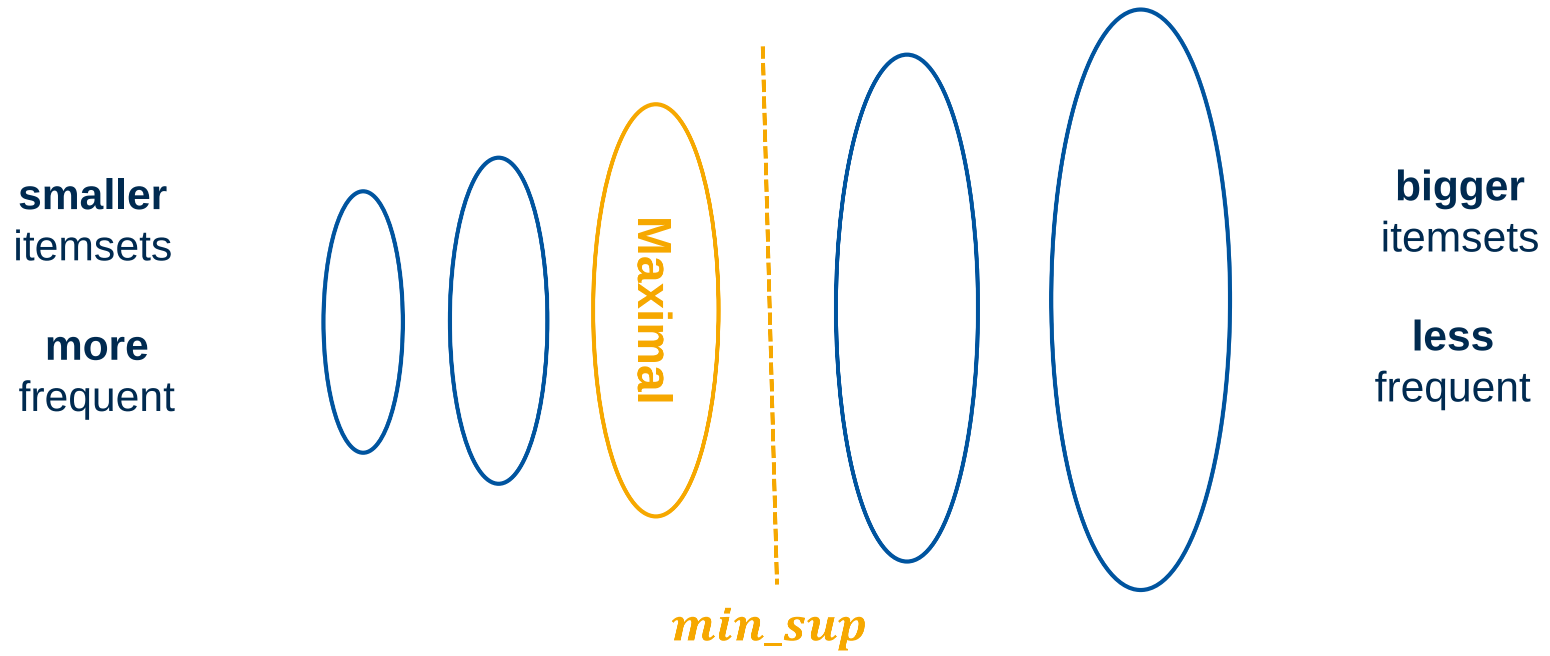


closed frequent itemsets are **closed and frequent**

Maximal Frequent Itemsets

An itemset $\mathcal{A}$ is a maximal frequent itemset if:

- $\mathcal{A}$ is frequent
- there is no proper superset $\mathcal{B} \supset \mathcal{A}$ that is also frequent



Relationships

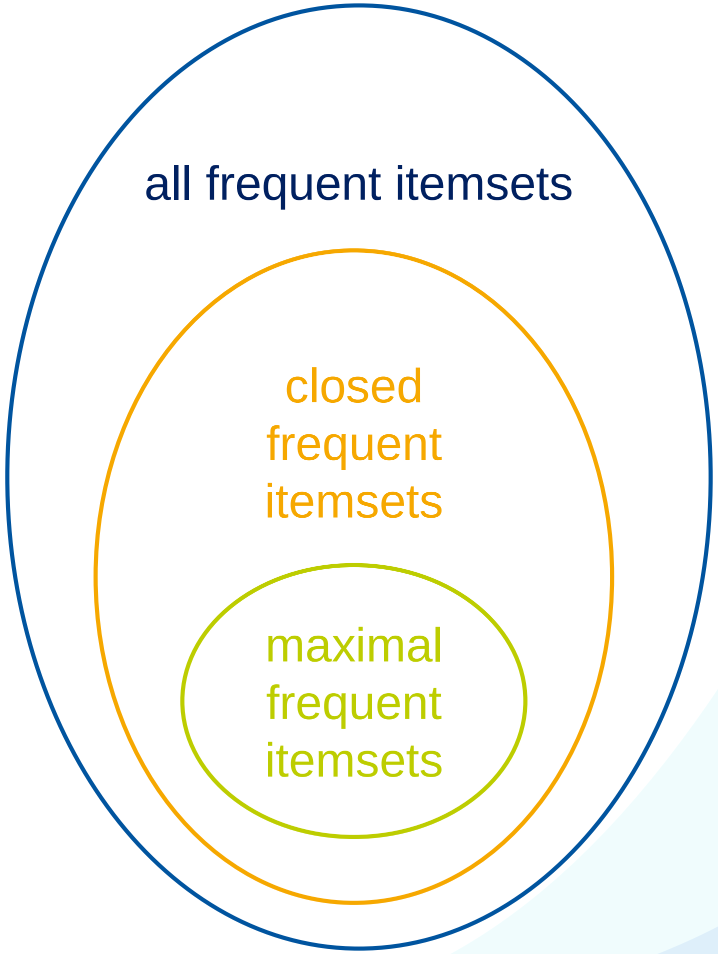
An itemset $\mathcal{A}$ is a closed frequent itemset if:

- $\mathcal{A}$ is frequent
- there is no proper superset $\mathcal{B} \supset \mathcal{A}$ that has the same support

An itemset $\mathcal{A}$ is a maximal frequent itemset if:

- $\mathcal{A}$ is frequent
- there is no proper superset $\mathcal{B} \supset \mathcal{A}$ that is also frequent

Hence, maximal frequent itemsets are closed by definition.



Example

Assume:
$\mathcal{I} = \{I_1, I_2, \ldots, I_{100}\}, \boldsymbol{min\_sup} = \frac{5}{20} = 0.25$
$\mathcal{X} = [\{I_1, I_2, \ldots, I_{50}\}^{10}, \{I_1, I_2, \ldots, I_{100}\}^{10}]$

- There are $\mathbf{2^{100} - 1 = 1.27 \times 10^{30}}$ itemsets; all are frequent.
- There are two closed frequent itemsets:
  - $\mathcal{A} = \{I_1, I_2, \ldots, I_{50}\}$ $with\ \boldsymbol{support}(\mathcal{A}) = \frac{\mathbf{20}}{\mathbf{20}}$
  - $\mathcal{B} = \{I_1, I_2, \ldots, I_{100}\}$ $with\ \boldsymbol{support}(\mathcal{B}) = \frac{\mathbf{10}}{\mathbf{20}}$
- There is only one maximal frequent itemset: $\mathcal{B} = \{I_1, I_2, \ldots, I_{100}\}$

Example

Assume:

$\mathcal{I} = \{I_1, I_2, \ldots, I_{100}\}, \boldsymbol{min\_sup} = \frac{15}{20} = 0.75$

$\mathcal{X} = [\{I_1, I_2, \ldots, I_{50}\}^{10}, \{I_1, I_2, \ldots, I_{100}\}^{10}]$

- There are $\mathbf{2^{50} - 1 = 3.17 \times 10^{15}}$ itemsets that are frequent.
- There is one closed frequent itemsets:
  - $\mathcal{A} = \{I_1, I_2, \ldots, I_{50}\}\, with\ \boldsymbol{support}(\mathcal{A}) = \mathbf{\frac{20}{20}}$
- There is only one maximal frequent itemset: $\mathcal{A} = \{I_1, I_2, \ldots, I_{50}\}$

Example

Assume:
$\mathcal{I} = \{I_1, I_2, \ldots, I_{100}\}, \boldsymbol{min\_sup} = \frac{15}{20}$
$\mathcal{X} = [\{I_1, I_2, \ldots, I_{99}\}^{10}, \{I_2, I_3, \ldots, I_{100}\}^{10}]$

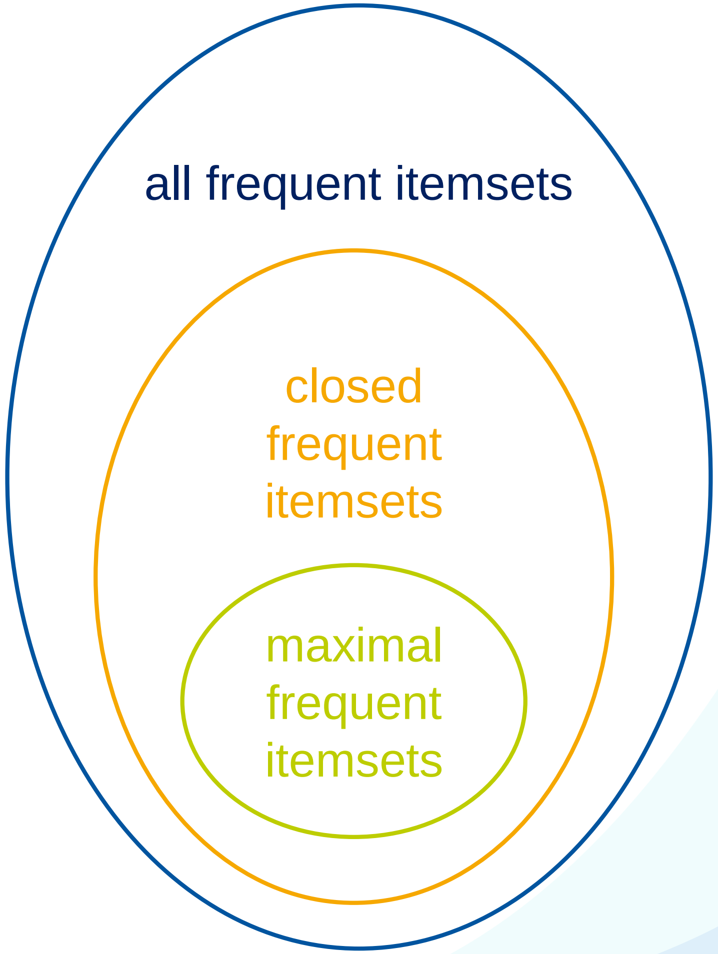
- There are $\mathbf{2^{98} - 1 = 1.17 \times 10^{29}}$ itemsets that are frequent.
- There is one closed frequent itemset:
  - $\mathcal{A} = \{I_2, I_3, \ldots, I_{99}\}$ *with* $\boldsymbol{support}(\mathcal{A}) = \frac{\mathbf{20}}{\mathbf{20}}$
- There is only one maximal frequent itemset: $\mathcal{A} = \{I_2, I_3, \ldots, I_{99}\}$

Observations

- The supports of the closed frequent itemsets provide complete information about the supports of all frequent item sets
- Formally, assume:
  - $\mathcal{A} \subset \mathcal{B}$,
  - $\mathcal{B}$ is a closed frequent itemset, and
  - there is no closed frequent itemset $\mathcal{B}'$ such that $\mathcal{A} \subseteq \mathcal{B}' \subset \mathcal{B}$.

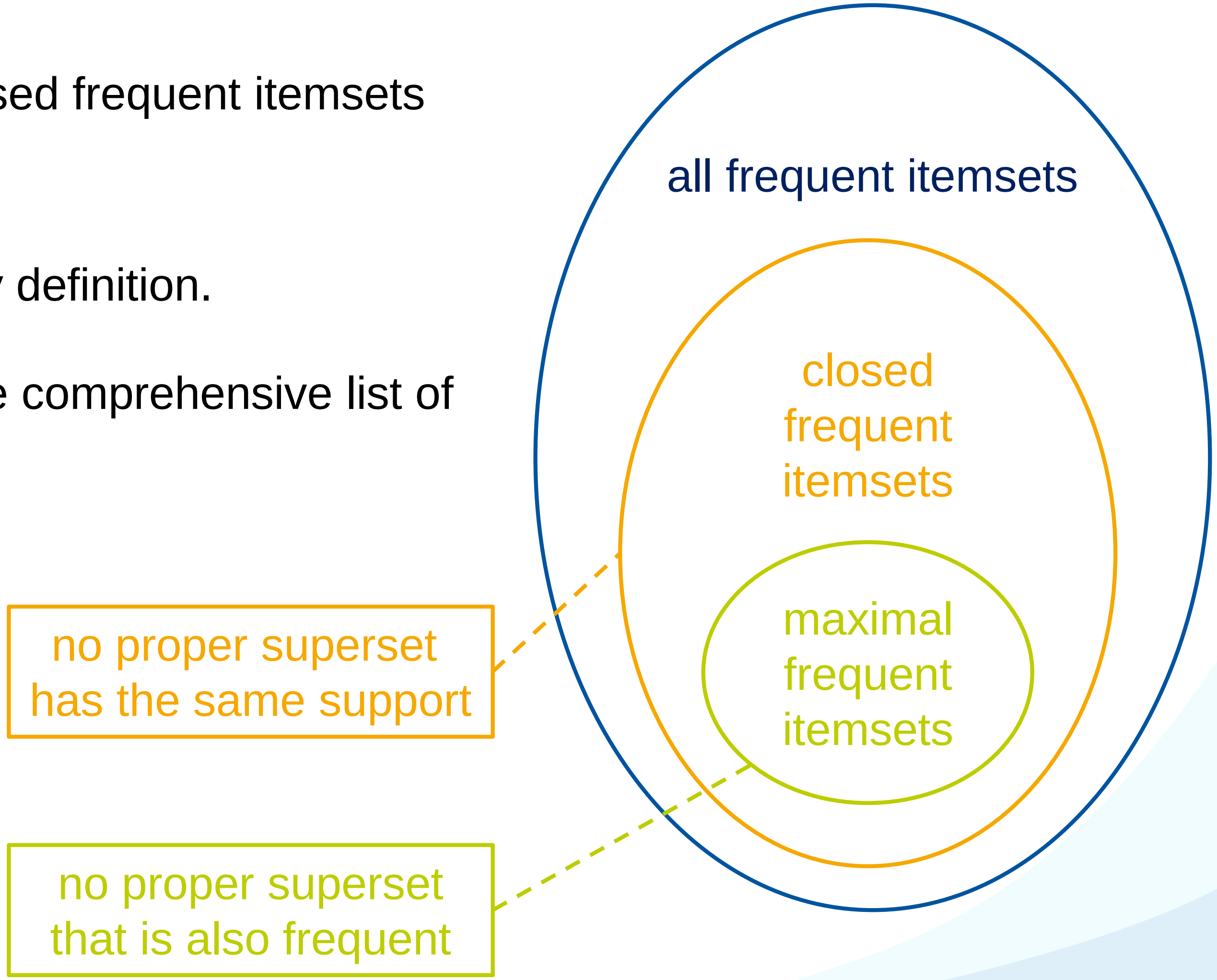
  Then $\boldsymbol{support}(\mathcal{A}) = \boldsymbol{support}(\mathcal{B})$.

→ It suffices to store closed frequent itemsets
(maximal frequent itemsets provide less information)



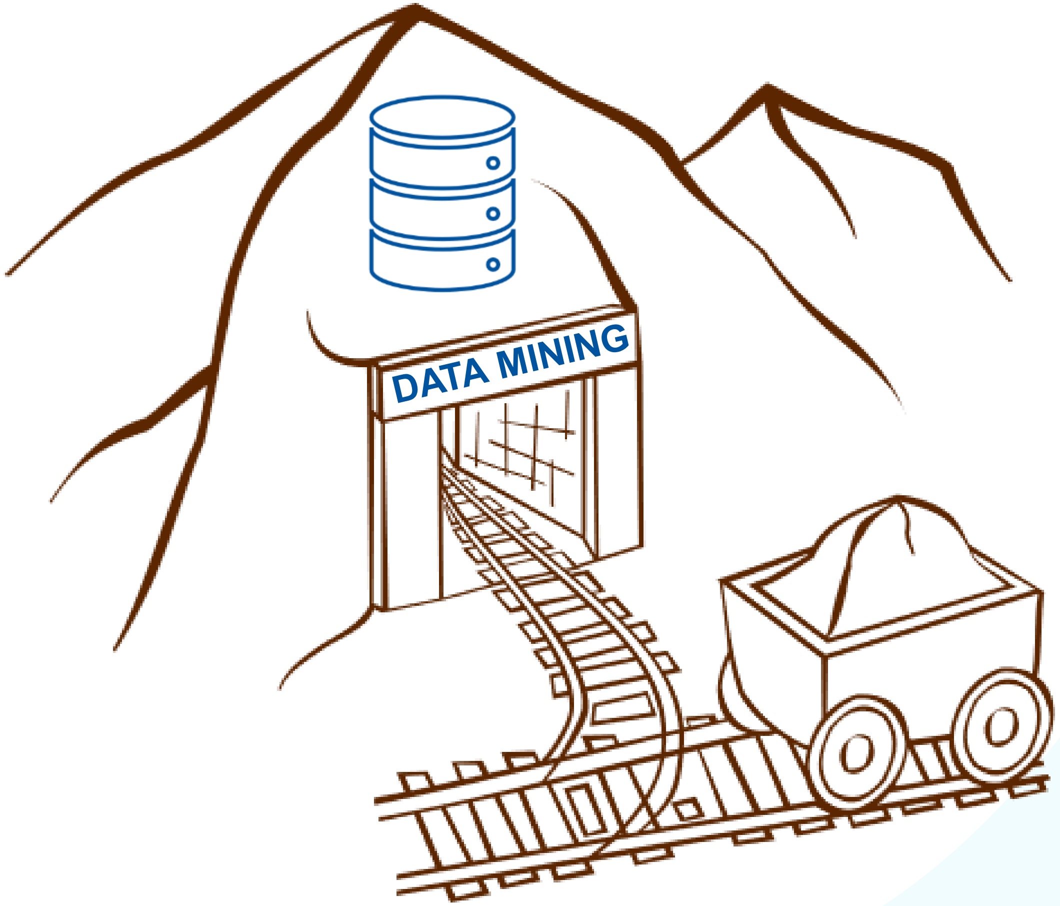
Summary

- Both maximal frequent itemsets and closed frequent itemsets are subsets of frequent itemsets.
- Maximal frequent itemsets are closed by definition.
- Closed frequent itemsets provide a more comprehensive list of frequent patterns



Frequent Itemsets

1. Introduction
2. Properties of Frequent Itemsets
3. **Apriori Algorithm**
4. FP-Growth Algorithm



Apriori Algorithm

- Introduced by Rakesh Agrawal and Ramakrishnan Srikant in “Fast Algorithms for Mining Association Rules in Large Databases. VLDB 1994: 487-499”
- Computes frequent itemsets / association rules in a dataset
- Uses a “bottom up” approach (starts with candidate itemsets of size one)
- Extends frequent subsets one item at a time (candidate generation)
- Avoids unnecessary checks by re-using information from smaller subsets and exploiting frequent itemsets’ properties

Apriori Algorithm – Basic Idea

$$\mathcal{L}_k = \{\mathcal{A} \subseteq \mathcal{I} \ \mid \boldsymbol{support}(\mathcal{A}) \geq \mathbf{min\_sup} \wedge |\mathcal{A}| = k\}$$

frequent itemsets of length *k*

1. Candidate generation: use the set $\mathcal{L}_k$ of frequent itemsets of length k to generate the candidate set $\mathcal{C}_{k+1}$ of candidate itemsets with length *k+1*

Apriori Algorithm – Basic Idea

$$\mathcal{L}_k = \{\mathcal{A} \subseteq \mathcal{I} \mid \boldsymbol{support}(\mathcal{A}) \geq \mathbf{min\_sup} \wedge |\mathcal{A}| = k\}$$

frequent itemsets of length *k*

1. Candidate generation: use the set $\mathcal{L}_k$ of frequent itemsets of length k to generate the candidate set $\mathcal{C}_{k+1}$ of candidate itemsets with length *k+1*

does not need the input data (efficient)

Apriori Algorithm – Basic Idea

$$\mathcal{L}_k = \{\mathcal{A} \subseteq \mathcal{I} \mid \boldsymbol{support}(\mathcal{A}) \geq \mathbf{min\_sup} \wedge |\mathcal{A}| = k\}$$

frequent itemsets of length *k*

1. Candidate generation: use the set $\mathcal{L}_k$ of frequent itemsets of length k to generate the candidate set $\mathcal{C}_{k+1}$ of candidate itemsets with length *k+1*

2. Pruning (antimonotonicity): all nonempty subsets of a frequent itemset must also be frequent → superset of an infrequent itemset cannot be frequent

does not need the input data (efficient)

Apriori Algorithm – Basic Idea

$$\mathcal{L}_k = \{\mathcal{A} \subseteq \mathcal{I} \mid \textbf{\textit{support}}(\mathcal{A}) \geq \textbf{min\_sup} \wedge |\mathcal{A}| = k\}$$

frequent itemsets of length *k*

1. Candidate generation: use the set $\mathcal{L}_k$ of frequent itemsets of length k to generate the candidate set $\mathcal{C}_{k+1}$ of candidate itemsets with length *k+1*

2. Pruning (antimonotonicity): all nonempty subsets of a frequent itemset must also be frequent → superset of an infrequent itemset cannot be frequent

does not need the input data (efficient)

3. Testing candidates: use the dataset to filter the infrequent itemsets from $\mathcal{C}_{k+1}$ and obtain $\mathcal{L}_{k+1}$

needs the input data (inefficient)

Apriori Algorithm – Basic Idea

$$\mathcal{L}_k = \{\mathcal{A} \subseteq \mathcal{I} \mid \boldsymbol{support}(\mathcal{A}) \geq \mathbf{min\_sup} \wedge |\mathcal{A}| = k\}$$

frequent itemsets of length *k*

1. Candidate generation: use the set $\mathcal{L}_k$ of frequent itemsets of length k to generate the candidate set $\mathcal{C}_{k+1}$ of candidate itemsets with length *k+1*
2. Pruning (antimonotonicity): all nonempty subsets of a frequent itemset must also be frequent → superset of an infrequent itemset cannot be frequent

does not need the input data (efficient)

3. Testing candidates: use the dataset to filter the infrequent itemsets from $\mathcal{C}_{k+1}$ and obtain $\mathcal{L}_{k+1}$

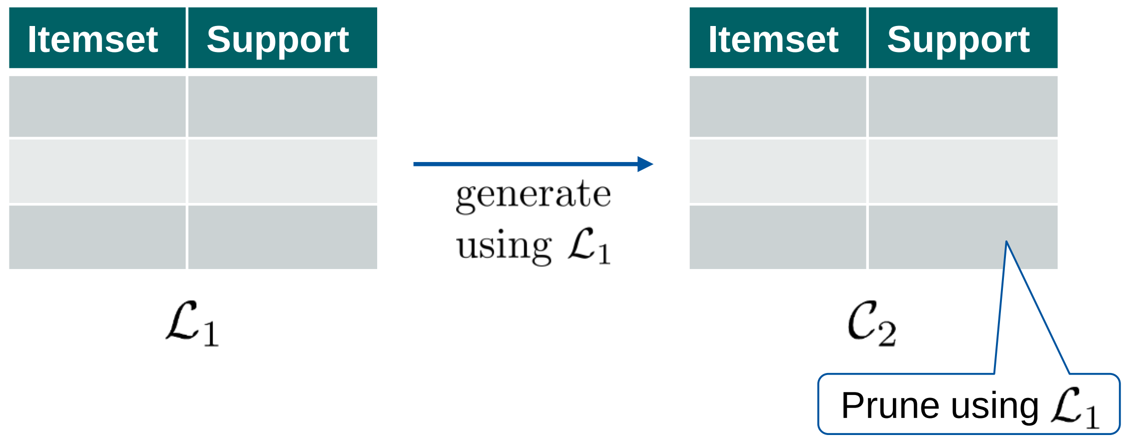
needs the input data (inefficient)

Apriori Algorithm – Basic Idea

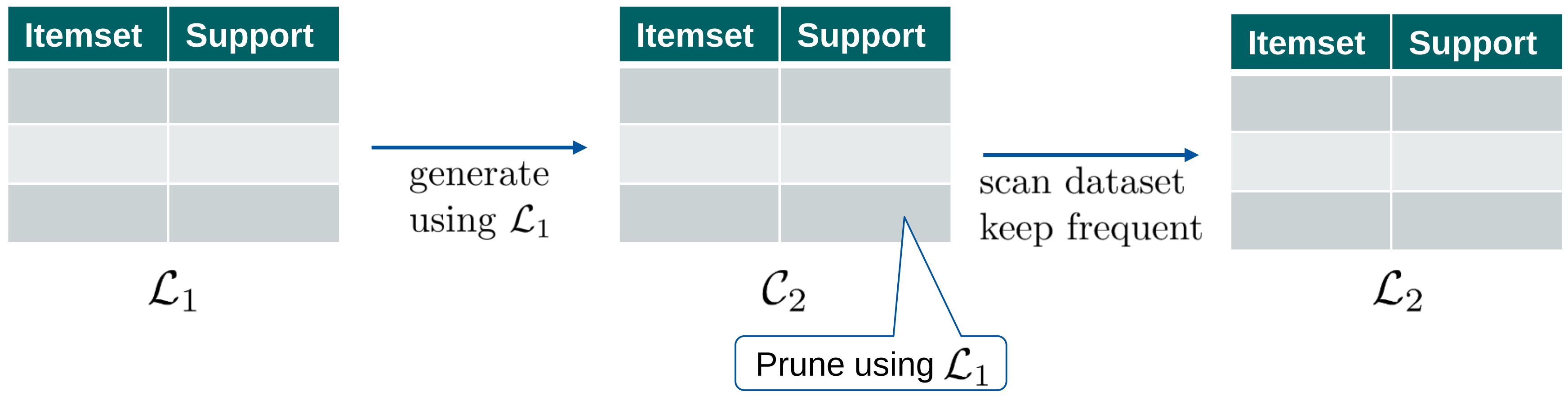
Itemset	Support

$\mathcal{L}_1$

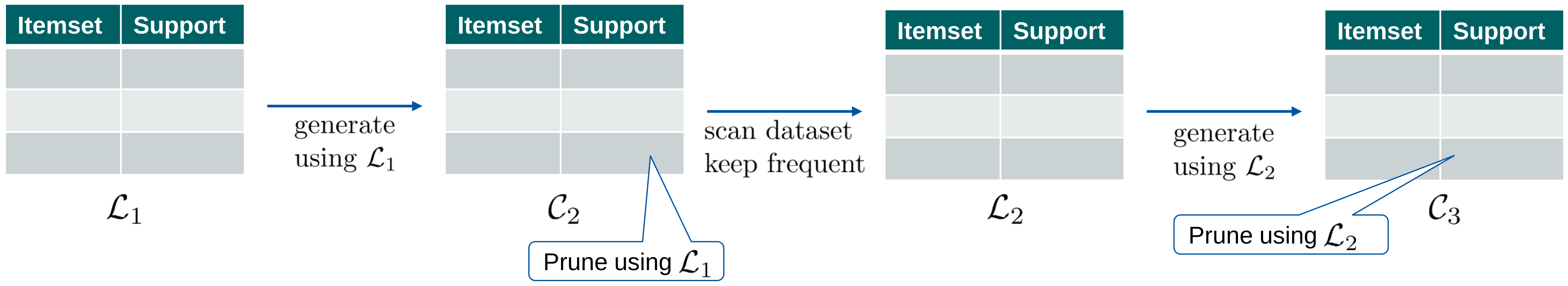
Apriori Algorithm – Basic Idea



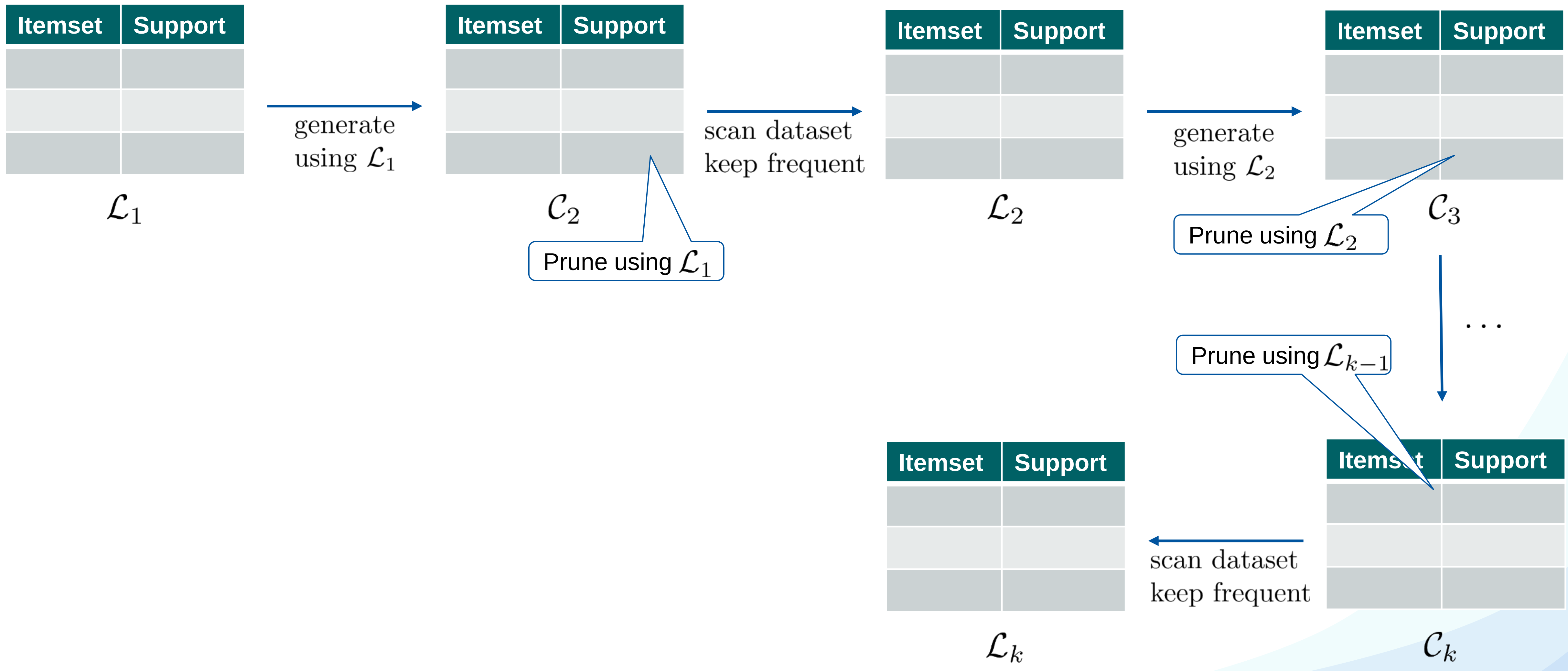
Apriori Algorithm – Basic Idea



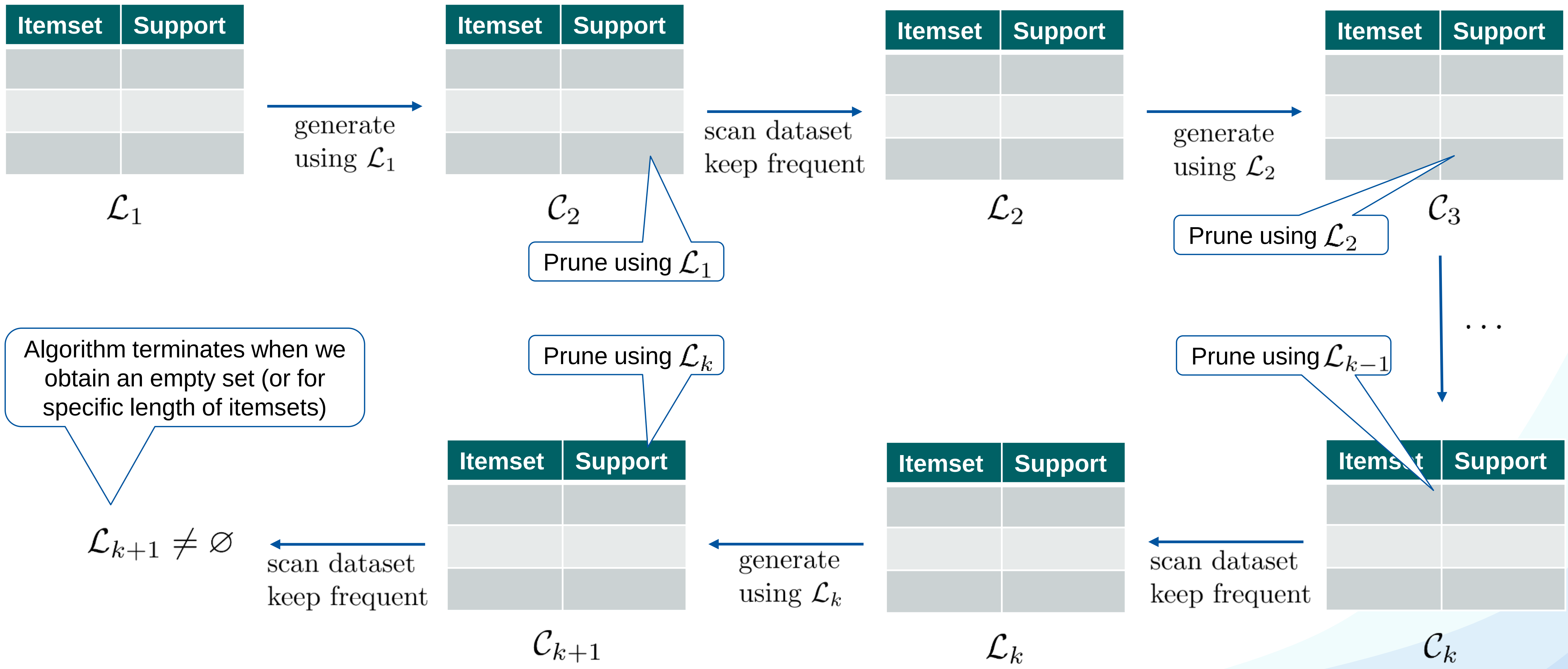
Apriori Algorithm – Basic Idea



Apriori Algorithm – Basic Idea



Apriori Algorithm – Basic Idea

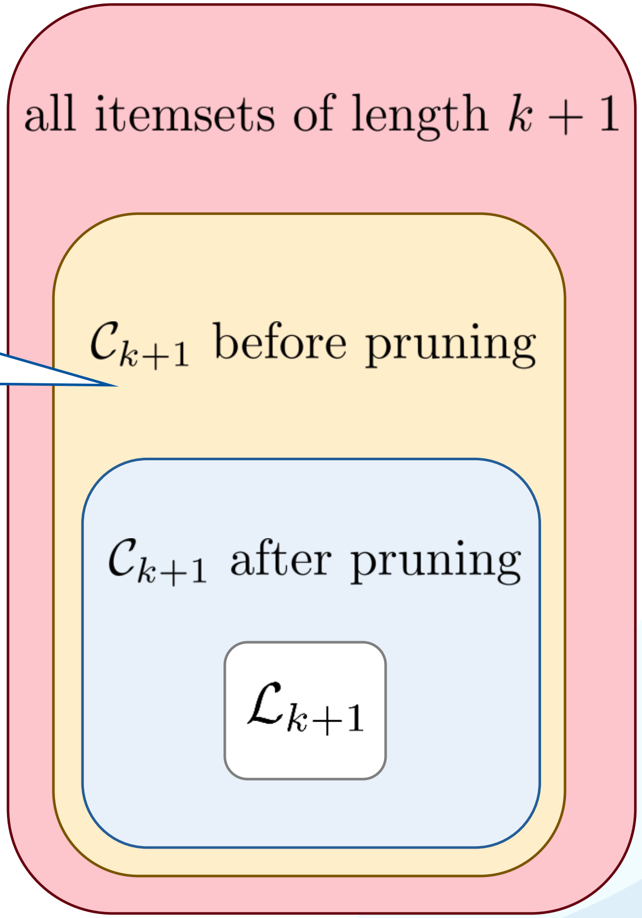


Candidate Generation – Leveling

Leveling is used to generate candidate itemset $\mathcal{C}_{k+1}$ from $\mathcal{L}_k$:

For any $\mathcal{A} \in \mathcal{L}_{k+1}$ there exist $\mathcal{A}', \mathcal{A}'' \in \mathcal{L}_k$ such that $\mathcal{A} = \mathcal{A}' \cup \mathcal{A}''$

Hence, we can obtain the candidate itemsets $\mathcal{C}_{k+1} \supseteq \mathcal{L}_{k+1}$ by joining suitable $\mathcal{A}', \mathcal{A}'' \in \mathcal{L}_k$!



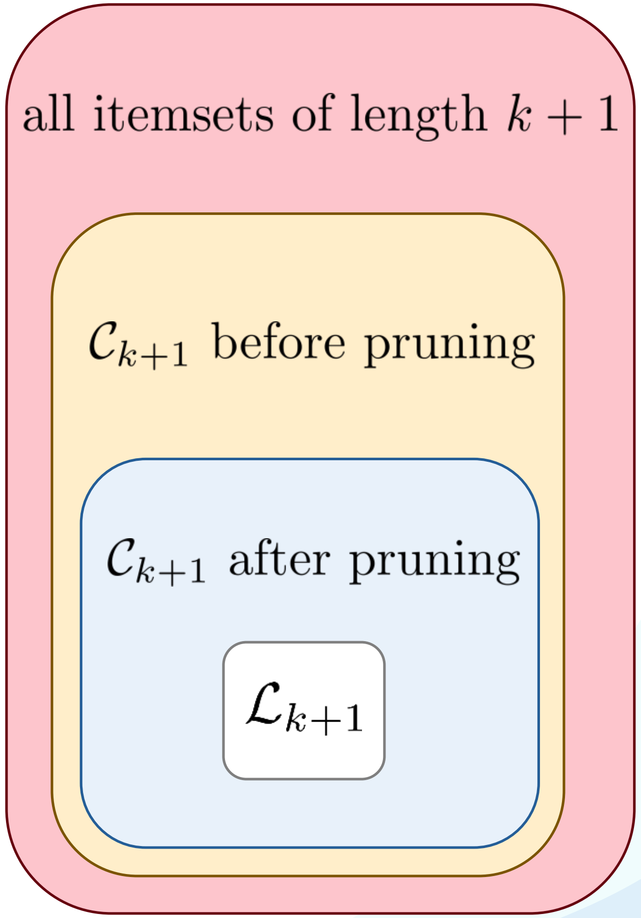
Candidate Generation – Leveling

Leveling is used to generate candidate itemset $\mathcal{C}_{k+1}$ from $\mathcal{L}_k$:

$$\boxed{\text{For any } \mathcal{A} \in \mathcal{L}_{k+1} \text{ there exist } \mathcal{A}', \mathcal{A}'' \in \mathcal{L}_k \text{ such that } \mathcal{A} = \mathcal{A}' \cup \mathcal{A}''}$$

Assume that the items are ordered $(I_1, I_2, \dots)$ and that
$\mathcal{A} = \{I_1, I_2, \dots, I_{k-1}, I_k, I_{k+1}\} \in \mathcal{L}_{k+1}$

If $\mathcal{A}$ is frequent, its subsets must be frequent, in particular:
$\mathcal{A}' = \{I_1, I_2, \dots, I_{k-1}, I_k\} \in \mathcal{L}_k$
$\mathcal{A}'' = \{I_1, I_2, \dots, I_{k-1}, I_{k+1}\} \in \mathcal{L}_k$
$\mathcal{A}' \cup \mathcal{A}'' = \{I_1, I_2, \dots, I_{k-1}, I_k, I_{k+1}\} \in \mathcal{L}_{k+1} = \mathcal{A}$



Candidate Generation – Leveling

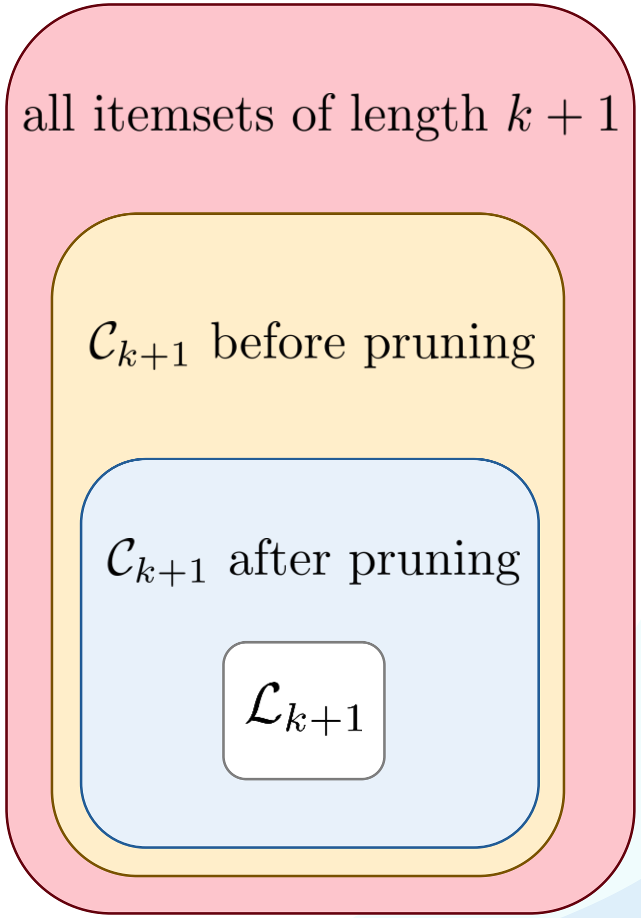
Leveling is used to generate candidate itemset $\mathcal{C}_{k+1}$ from $\mathcal{L}_k$:

$$\boxed{\text{For any } \mathcal{A} \in \mathcal{L}_{k+1} \text{ there exist } \mathcal{A}', \mathcal{A}'' \in \mathcal{L}_k \text{ such that } \mathcal{A} = \mathcal{A}' \cup \mathcal{A}''}$$

Assume that the items are ordered $(I_1, I_2, \dots)$ and that
$\mathcal{A} = \{I_1, I_2, \dots, I_{k-1}, I_k, I_{k+1}\} \in \mathcal{L}_{k+1}$

If $\mathcal{A}$ is frequent, its subsets must be frequent, in particular:
$\mathcal{A}' = \{I_1, I_2, \dots, I_{k-1}, I_k\} \in \mathcal{L}_k$
$\mathcal{A}'' = \{I_1, I_2, \dots, I_{k-1}, I_{k+1}\} \in \mathcal{L}_k$
$\mathcal{A}' \cup \mathcal{A}'' = \{I_1, I_2, \dots, I_{k-1}, I_k, I_{k+1}\} \in \mathcal{L}_{k+1} = \mathcal{A}$

$\Rightarrow$ We can generate $\mathcal{C}_{k+1}$ by joining itemsets $\mathcal{A}', \mathcal{A}'' \in \mathcal{L}_k$ which differ in one item



Candidate Generation

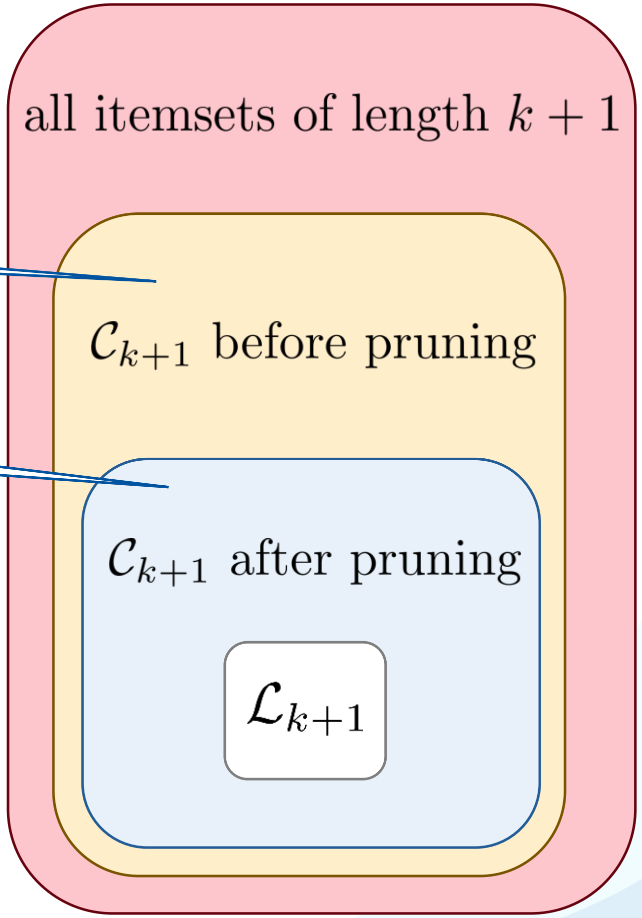
Thanks to leveling:

- Apriori creates the set of candidate itemsets of length k+1, $\mathcal{C}_{k+1}$, by joining two frequent itemsets of length *k*
- This can be done efficiently without creating duplicates
- Next, we prune the set $\mathcal{C}_{k+1}$ based on infrequent subsets

If $\mathcal{A}$ is frequent, its subsets must be frequent, in particular:

$$\mathcal{A}' = \{I_1, I_2, \ldots, I_{k-1}, I_k\} \in \mathcal{L}_k$$
$$\mathcal{A}'' = \{I_1, I_2, \ldots, I_{k-1}, I_{k+1}\} \in \mathcal{L}_k$$
$$\mathcal{A}' \cup \mathcal{A}'' = \{I_1, I_2, \ldots, I_{k-1}, I_k, I_{k+1}\} \in \mathcal{L}_{k+1} = \mathcal{A}$$

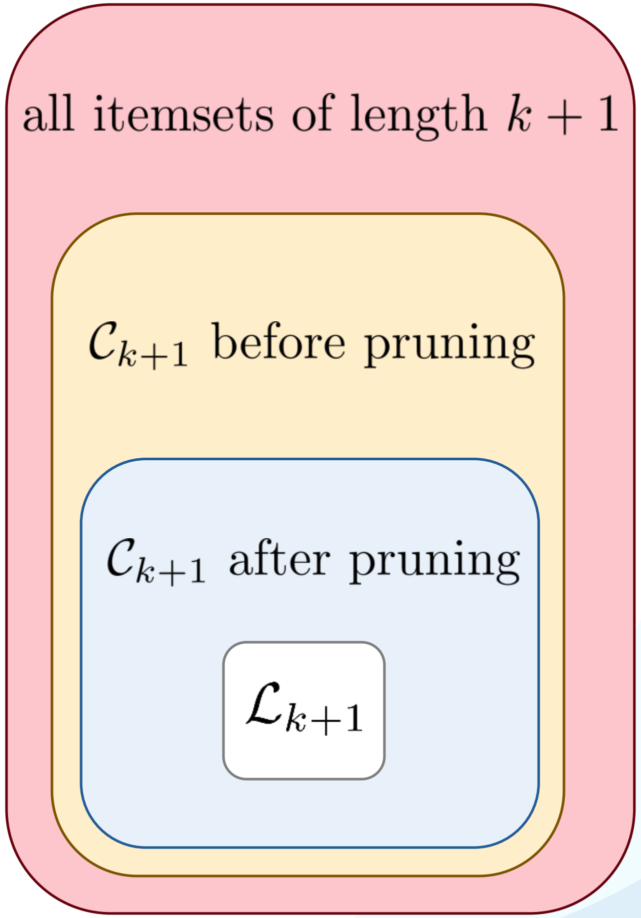
$\Rightarrow$ We can generate $\mathcal{C}_{k+1}$ by joining itemsets $\mathcal{A}', \mathcal{A}'' \in \mathcal{L}_k$ which differ in one item



Pruning – Antimonotonicity

Antimonotonicity is used to prune the candidate set:

If $\mathcal{B}$ is a frequent itemset, any subset $\mathcal{A} \subseteq \mathcal{B}$ must be frequent
$\Rightarrow$ If a subset $\mathcal{A} \subseteq \mathcal{B}$ is infrequent, then $\mathcal{B}$ is infrequent



For any $\mathcal{A} \subseteq \mathcal{I}$ and $\mathcal{B} \subseteq \mathcal{I}$:

1. If $\mathcal{A} \subseteq \mathcal{B}$, then $\textbf{support}(\mathcal{A}) \geq \textbf{support}(\mathcal{B})$
2. If $\mathcal{A} \subseteq \mathcal{B}$ and $\textbf{support}(\mathcal{B}) \geq \textbf{min\_sup}$, then $\textbf{support}(\mathcal{A}) \geq \textbf{min\_sup}$
3. If $\mathcal{A} \subseteq \mathcal{B}$ and $\textbf{support}(\mathcal{A}) < \textbf{min\_sup}$, then $\textbf{support}(\mathcal{B}) < \textbf{min\_sup}$

Pruning – Antimonotonicity

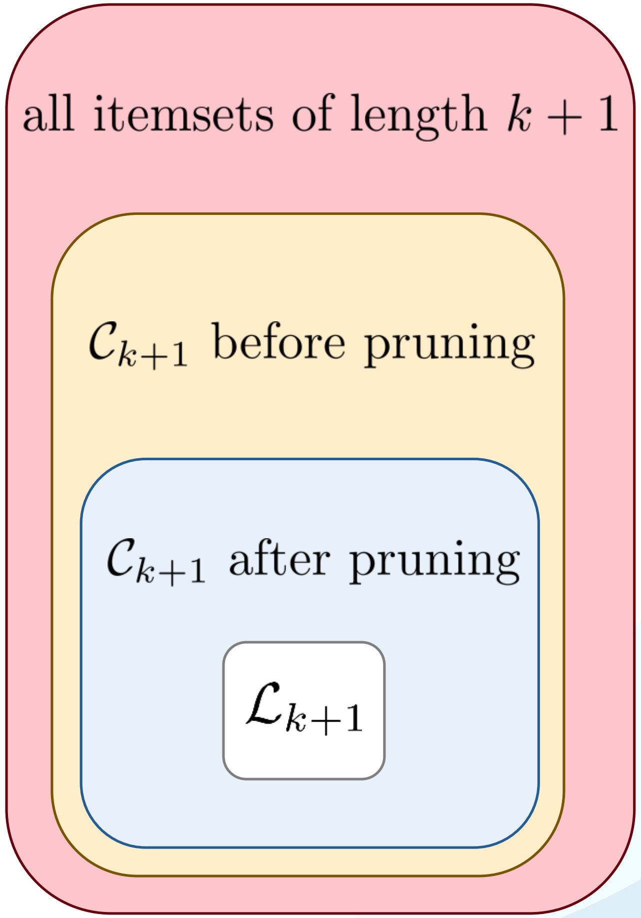
Antimonotonicity is used to prune the candidate set:

If $\mathcal{B}$ is a frequent itemset, any subset $\mathcal{A} \subseteq \mathcal{B}$ must be frequent
$\Rightarrow$ If a subset $\mathcal{A} \subseteq \mathcal{B}$ is infrequent, then $\mathcal{B}$ is infrequent

1. If {Apple} or {Banana} is infrequent, then {Apple, Banana} is infrequent

2. If {Grapes} or {Banana} is infrequent, then {Grapes, Banana} is infrequent

3. If {Apples} or {Grapes} is infrequent, then {Apples, Grapes} is infrequent



Pruning – Antimonotonicity

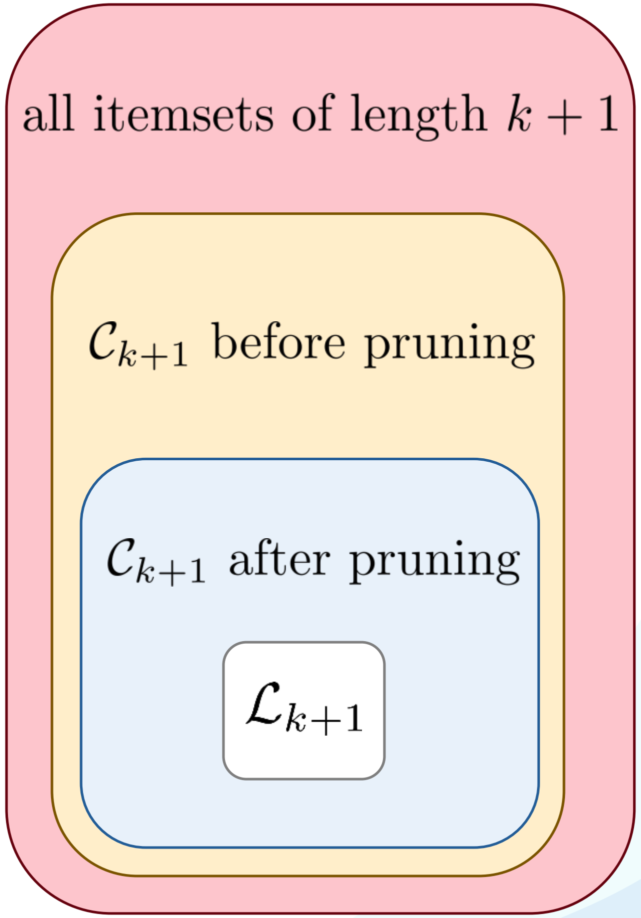
Antimonotonicity is used to prune the candidate set:

If $\mathcal{B}$ is a frequent itemset, any subset $\mathcal{A} \subseteq \mathcal{B}$ must be frequent
$\Rightarrow$ If a subset $\mathcal{A} \subseteq \mathcal{B}$ is infrequent, then $\mathcal{B}$ is infrequent

1. If {Apple} or {Banana} is infrequent, then {Apple, Banana} is infrequent
2. If {Grapes} or {Banana} is infrequent, then {Grapes, Banana} is infrequent
3. If {Apples} or {Grapes} is infrequent, then {Apples, Grapes} is infrequent

Bought Fruits	Support _count
{Banana}	4
{Grapes}	1
{Apple}	2

$\mathbf{min\_sup\_count} = 2$



Pruning – Antimonotonicity

Antimonotonicity is used to prune the candidate set:

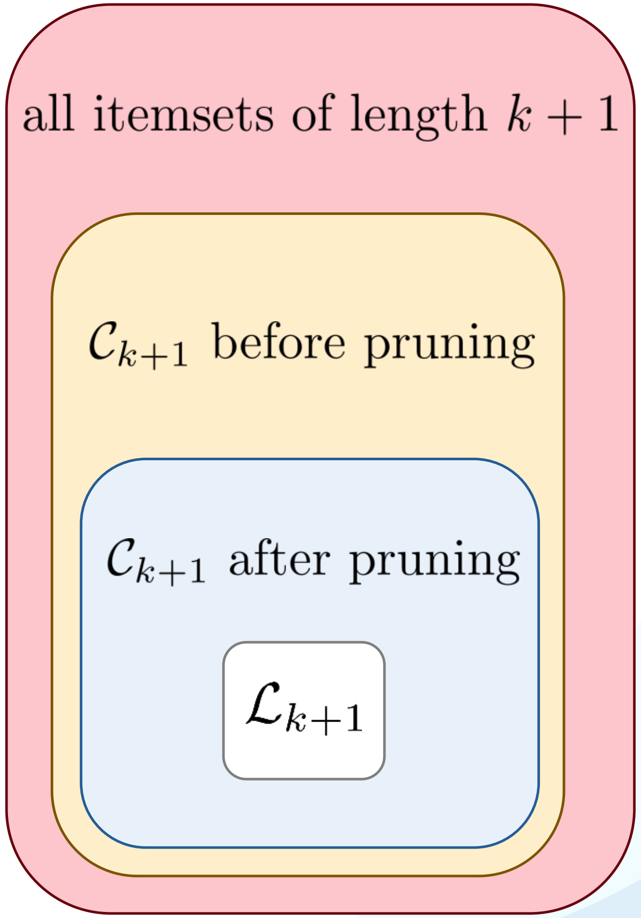
If $\mathcal{B}$ is a frequent itemset, any subset $\mathcal{A} \subseteq \mathcal{B}$ must be frequent

$\Rightarrow$ If a subset $\mathcal{A} \subseteq \mathcal{B}$ is infrequent, then $\mathcal{B}$ is infrequent

ID	Bought Fruits
1	{Banana, Apple}
2	{Grapes, Banana}
3	{Apple, Grapes}

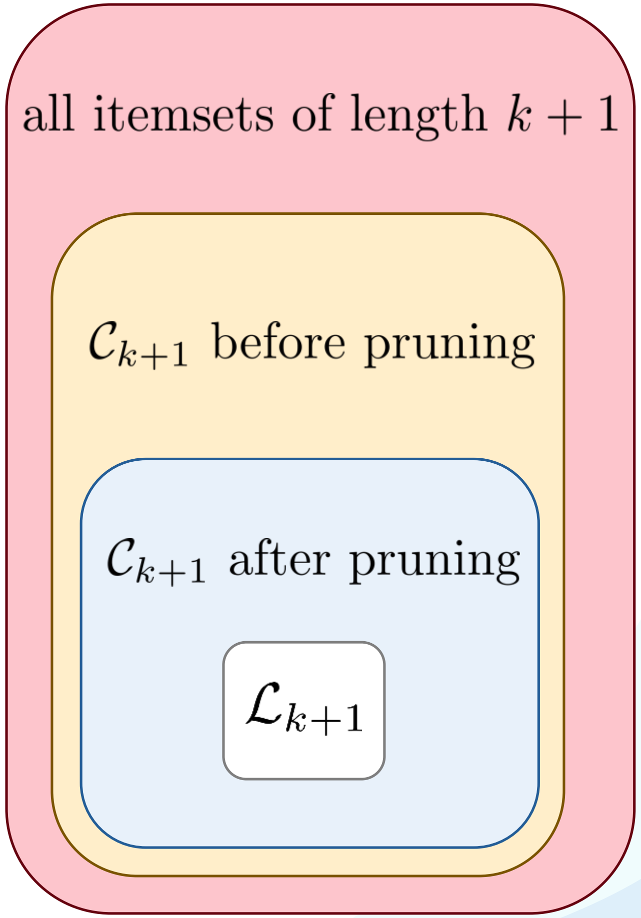
$\xrightarrow{\mathbf{min\_sup\_count} = 2}$

ID	Bought Fruits
1	{Banana, Apple}
~~2~~	~~{Grapes, Banana}~~
~~3~~	~~{Apple, Grapes}~~



Testing Candidates

- After candidate generation and pruning test the remaining candidate itemsets
- We scan the dataset $\mathcal{X}$ and remove all infrequent candidate itemsets from $\mathcal{C}_{k+1}$ to obtain $\mathcal{L}_{k+1}$

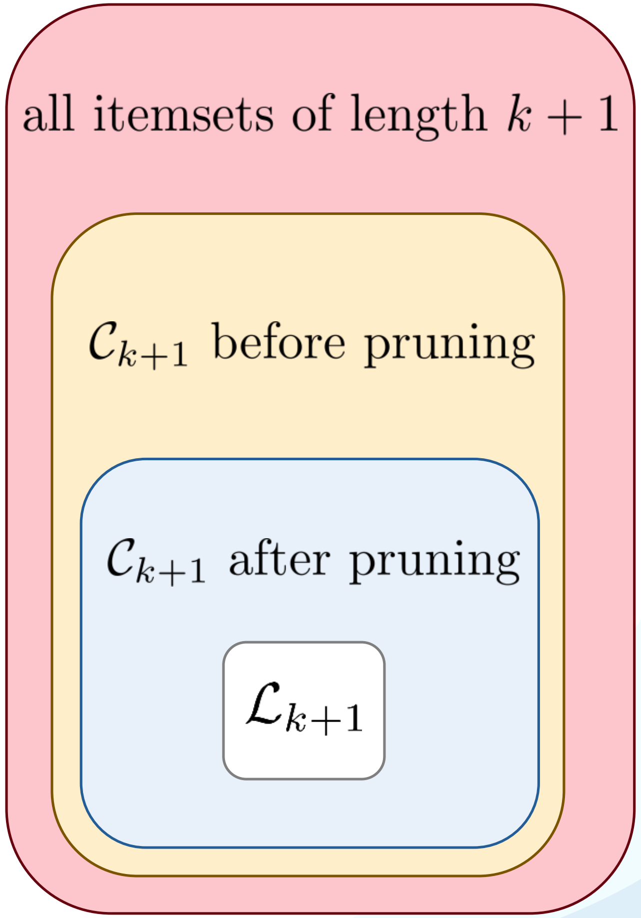


Testing Candidates

- After candidate generation and pruning test the remaining candidate itemsets
- We scan the dataset $\mathcal{X}$ and remove all infrequent candidate itemsets from $\mathcal{C}_{k+1}$ to obtain $\mathcal{L}_{k+1}$

- Consider all transactions $\mathcal{T} \in \mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{I}))$
- For each candidate itemset $\mathcal{A} \in \mathcal{C}_k$ increment the corresponding counter if $\mathcal{A} \subseteq \mathcal{T}_k$
- This returns the frequencies (**support_count**) of the candidate itemsets and we can compute $\mathcal{L}_{k+1}$ from $\mathcal{C}_{k+1}$

$$\mathcal{L}_{k+1} = \{\mathcal{A} \in \mathcal{C}_{k+1} \,|\, \mathbf{support}(\mathcal{A}) \geq \mathbf{min\_sup}\}$$



Algorithm

Apriori algorithm:

1. Find the frequent itemsets of length 1 ($\mathcal{L}_1$)
2. Let $k \leftarrow 1$
3. **Repeat until** $\mathcal{L}_k \neq \varnothing$:

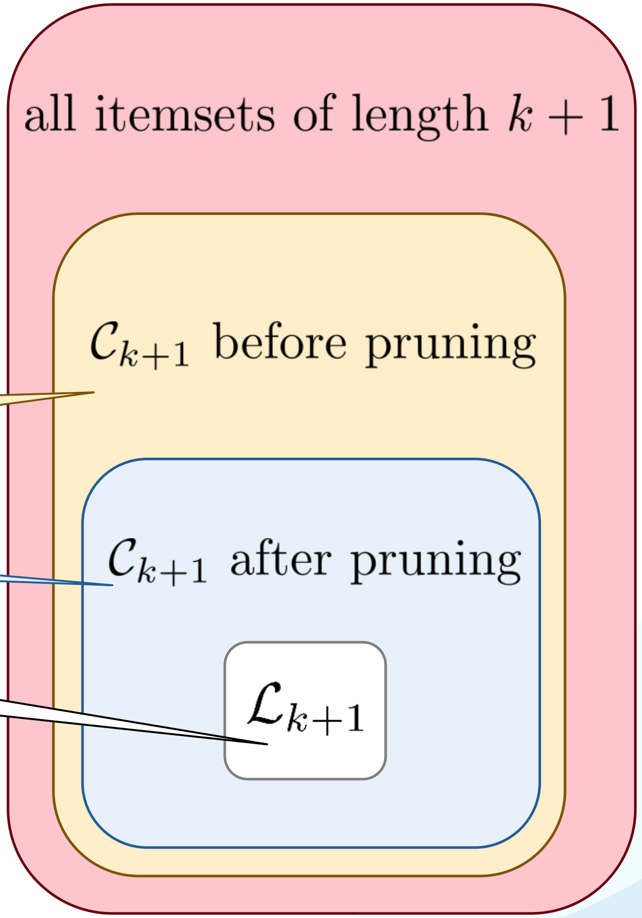
   (Or until we find frequent itemsets of pre-defined length)

   (a) Generate set of candidate itemsets of length k+1 ($\mathcal{C}_{k+1}$) based on $\mathcal{L}_k$

   (b) Prune $\mathcal{C}_{k+1}$ from itemsets that have infrequent subsets

   (c) Test all remaining candidates from $\mathcal{C}_{k+1}$ and remove infrequent itemsets to obtain $\mathcal{L}_{k+1}$

   (d) Assign $k \leftarrow k + 1$



Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

$\longrightarrow$

Itemset	Count
{Grapes}	3
{Apple}	4
{Pineapple}	1
{Orange}	3
{Banana}	4

$\mathcal{C}_1$

Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

Itemset	Count
{Grapes}	3
{Apple}	4
{Pineapple}	1
{Orange}	3
{Banana}	4

$\mathcal{C}_1$

$\mathbf{min\_sup\_count} = 2$

Itemset	Count
{Grapes}	3
{Apple}	4
~~{Pineapple}~~	~~1~~
{Orange}	3
{Banana}	4

$\mathcal{L}_1$

Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

Itemset	Count
{Grapes}	3
{Apple}	4
{Pineapple}	1
{Orange}	3
{Banana}	4

$\mathcal{C}_1$

$\mathbf{min\_sup\_count} = 2$

Itemset	Count
{Grapes}	3
{Apple}	4
~~{Pineapple}~~	~~1~~
{Orange}	3
{Banana}	4

$\mathcal{L}_1$

generate candidates from $\mathcal{L}_1$

Itemset
{Grapes, Apple}
{Grapes, Orange}
{Grapes, Banana}
{Apple, Orange}
{Apple, Banana}
{Orange, Banana}

$\mathcal{C}_2$

pruning based on $\mathcal{L}_1$

Itemset	Count
{Grapes, Apple}	3
{Grapes, Orange}	1
{Grapes, Banana}	2
{Apple, Orange}	2
{Apple, Banana}	3
{Orange, Banana}	3

$\mathcal{C}_2$

Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

Itemset	Count
{Grapes}	3
{Apple}	4
{Pineapple}	1
{Orange}	3
{Banana}	4

$\mathcal{C}_1$

$\mathbf{min\_sup\_count} = 2$

Itemset	Count
{Grapes}	3
{Apple}	4
~~{Pineapple}~~	~~1~~
{Orange}	3
{Banana}	4

$\mathcal{L}_1$

generate candidates from $\mathcal{L}_1$

Itemset
{Grapes, Apple}
{Grapes, Orange}
{Grapes, Banana}
{Apple, Orange}
{Apple, Banana}
{Orange, Banana}

$\mathcal{C}_2$

pruning based on $\mathcal{L}_1$

Itemset	Count
{Grapes, Apple}	3
{Grapes, Orange}	1
{Grapes, Banana}	2
{Apple, Orange}	2
{Apple, Banana}	3
{Orange, Banana}	3

$\mathcal{C}_2$

$\mathbf{min\_sup\_count} = 2$

scan dataset test candidates

Itemset	Count
{Grapes, Apple}	3
~~{Grapes, Orange}~~	~~1~~
{Grapes, Banana}	2
{Apple, Orange}	2
{Apple, Banana}	3
{Orange, Banana}	3

$\mathcal{L}_2$

Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

$\longrightarrow \quad \cdots \quad \longrightarrow$

$\mathbf{min\_sup\_count} = 2$

Itemset	Count
{Grapes, Apple}	3
{Grapes, Banana}	2
{Apple, Orange}	2
{Apple, Banana}	3
{Orange, Banana}	3

$\mathcal{L}_2$

Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

$\cdots$

$\mathbf{min\_sup\_count} = 2$

Itemset	Count
{Grapes, Apple}	3
{Grapes, Banana}	2
{Apple, Orange}	2
{Apple, Banana}	3
{Orange, Banana}	3

$\mathcal{L}_2$

generate candidates from $\mathcal{L}_2$

Itemset
{Grapes, Apple, Banana}
{Grapes, Apple, Orange}
{Grapes, Banana, Orange}
{Apple, Banana, Orange}

$\mathcal{C}_3$

pruning based on $\mathcal{L}_2$

Itemset	Subsets of Length 2
{Grapes, Apple, Banana}	{Grapes, Apple}, {Grapes, Banana}, {Apple, Banana}
~~{Grapes, Apple, Orange}~~	~~{Grapes, Apple}, **{Grapes, Orange},** {Apple, Orange}~~
~~{Grapes, Banana, Orange}~~	~~{Grapes, Banana}, **{Grapes, Orange},** {Banana, Orange}~~
{Apple, Banana, Orange}	{Apple, Banana}, {Apple, Orange}, {Banana, Orange}

$\mathcal{C}_3$

$\mathbf{min\_sup\_count} = 2$

scan dataset
test candidates

Itemset	Count
{Grapes, Apple, Banana}	2
{Apple, Banana, Orange}	2

$\mathcal{L}_3$

Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

$\longrightarrow \ \cdots \ \longrightarrow$

$\mathbf{min\_sup\_count} = 2$

Itemset	Count
{Grapes, Apple, Banana}	2
{Apple, Banana, Orange}	2

$\mathcal{L}_3$

Example

TID	Bought Fruits
1	{Grapes, Apple, Pineapple}
2	{Orange, Apple, Banana}
3	{Grapes, Orange, Apple, Banana}
4	{Orange, Banana}
5	{Grapes, Apple, Banana}

$\mathcal{X}$

$\ldots$

$\mathbf{min\_sup\_count} = 2$

Itemset	Count
{Grapes, Apple, Banana}	2
{Apple, Banana, Orange}	2

$\mathcal{L}_3$

generate candidates from $\mathcal{L}_3$

Itemset
{Grapes, Apple, Banana, Orange}

$\mathcal{C}_4$

pruning based on $\mathcal{L}_3$

Itemset	Subsets of length 3
~~{Grapes, Apple, Banana, Orange}~~	~~{Grapes, Apple, Banana}, **{Grapes, Apple, Orange}, {Grapes, Banana, Orange},** {Apple, Banana, Orange}~~

No more candidates of length 4 – algorithm terminates

Optimizations

Further optimizations

- Distributing the data (can be done in various ways)
- Gradually removing transactions not containing any frequent itemset of length $\boldsymbol{k}$
- Sampling

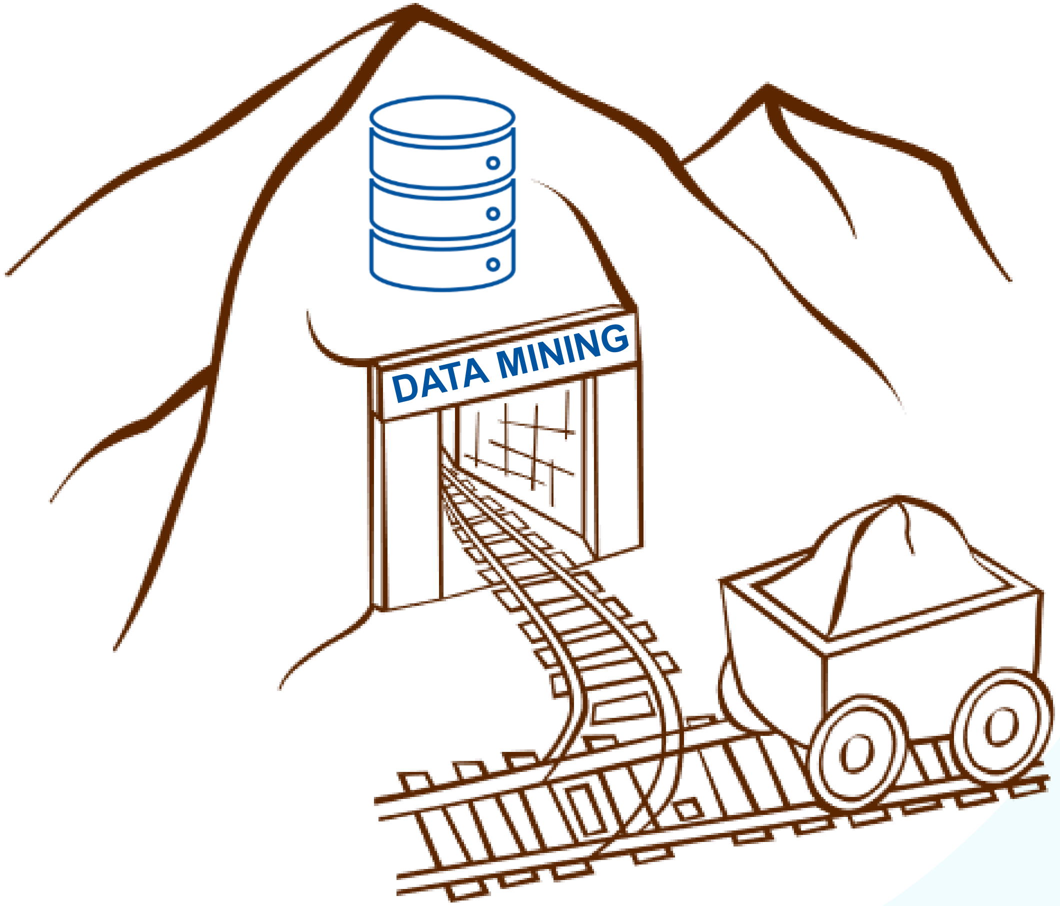
Limitations

- It may remain challenging to generate the candidate sets (may be huge)
- Each candidate needs to be tested against the whole dataset

→ FP-Growth is an approach that aims to overcome these limitations

Frequent Itemsets

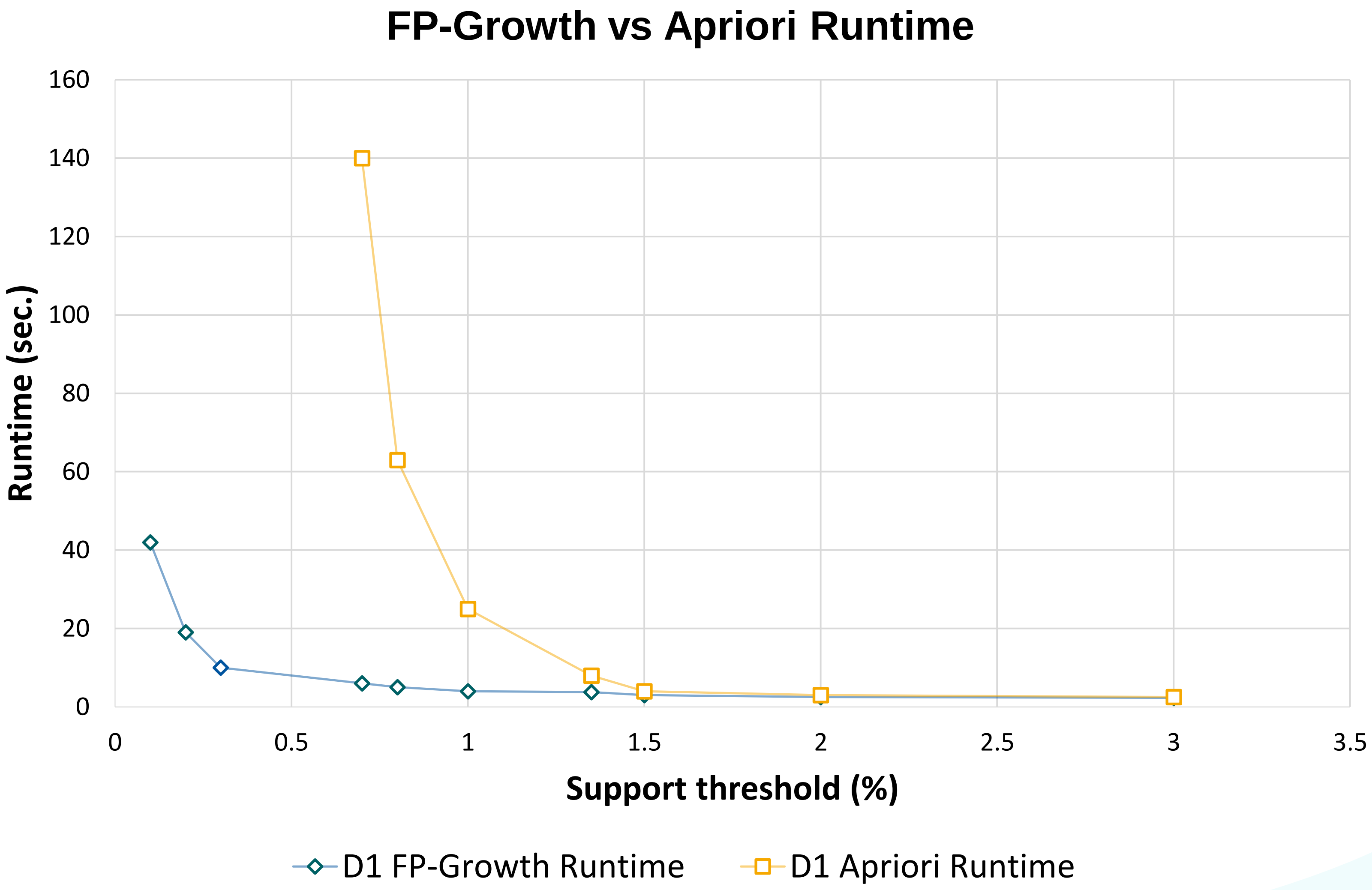
1. Introduction
2. Properties of Frequent Itemsets
3. Apriori Algorithm
4. **FP-Growth Algorithm**



Frequent Pattern Growth Algorithm

- Introduced by Jiawei Han, Jian Pei, Yiwen Yin in “Mining Frequent Patterns without Candidate Generation. SIGMOD Conference 2000: 1-12”
- Based on constructing the Frequent Pattern Tree (FP-Tree)
- Avoids generation of many candidates
- Depth-first rather than breadth-first
- Requires only two passes over the (potentially huge) dataset

Motivation



Graph is based on Jiawei Han, Jian Pei, Yiwen Yin in “Mining Frequent Patterns without Candidate Generation. SIGMOD Conference 2000: 1-12”

FP-Growth Steps

1. Determine the frequency of each item (first pass through the dataset)
2. Sort $\mathcal{I} = \{I_1, \ldots, I_D\}$ based on their frequencies (I_1 is most frequent, I_D is the least frequent)
3. Remove the non-frequent items
4. The remaining items in each transactions are ordered by frequency (same as above)
5. This can be used to build a so-called prefix tree (second pass trough the dataset)

FP-Growth Steps

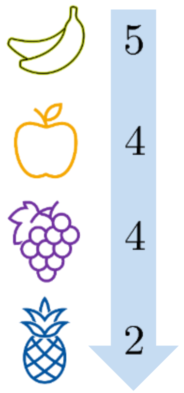
1. Determine the frequency of each item (first pass through the dataset)
2. Sort $\mathcal{I} = \{I_1, \ldots, I_D\}$ based on their frequencies (I_1 is most frequent, I_D is the least frequent)
3. Remove the non-frequent items
4. The remaining items in each transactions are ordered by frequency (same as above)
5. This can be used to build a so-called prefix tree (second pass trough the dataset)

6. The resulting FP-tree contains all information needed to find the frequent itemsets of any length (no need to traverse the dataset again)

Constructing FP-Tree – Example

TID	Bought Fruits
1	{Banana, Apple}
2	{Grapes, Banana, Pineapple}
3	{Apple, Banana}
4	{Apple, Grapes, Pineapple}
5	{Grapes, Banana}
6	{Apple, Banana, Grapes}

Dataset $\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{J}))$

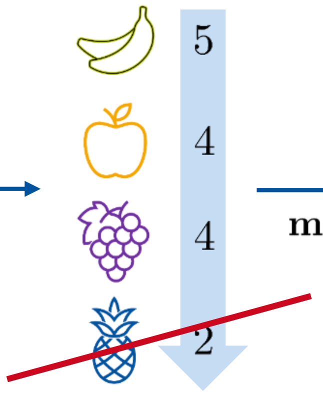


1. Determine the frequencies of items
2. Order the items based on frequency

Constructing FP-Tree – Example

TID	Bought Fruits
1	{Banana, Apple}
2	{Grapes, Banana, Pineapple}
3	{Apple, Banana}
4	{Apple, Grapes, Pineapple}
5	{Grapes, Banana}
6	{Apple, Banana, Grapes}

Dataset $\mathcal{X} \in \mathbb{M}(\mathbb{P}(\mathcal{J}))$



$\text{min\_sup\_count} = 3$

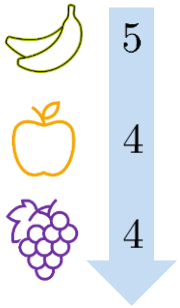
TID	Bought Fruits
1	{Banana, Apple}
2	{Banana, Grapes}
3	{Banana, Apple}
4	{Apple, Grapes}
5	{Banana, Grapes}
6	{Banana, Apple, Grapes}

1. Determine the frequencies of items
2. Order the items based on frequency

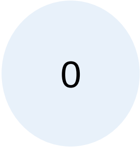
3. Remove non-frequent items
4. Sort the items in the transactions

Constructing FP-Tree – Example

TID	Bought Fruits
1	{Banana, Apple}
2	{Banana, Grapes}
3	{Banana, Apple}
4	{Apple, Grapes}
5	{Banana, Grapes}
6	{Banana, Apple, Grapes}

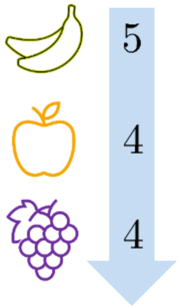


5. Build the FP-tree going through each transaction

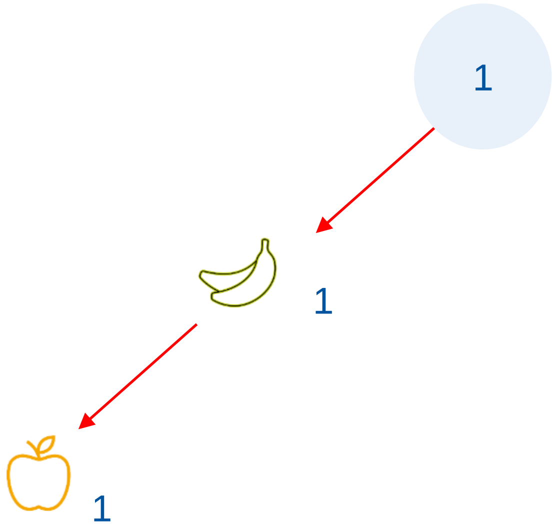


Constructing FP-Tree – Example

TID	Bought Fruits
1	**{Banana, Apple}**
2	{Banana, Grapes}
3	{Banana, Apple}
4	{Apple, Grapes}
5	{Banana, Grapes}
6	{Banana, Apple, Grapes}

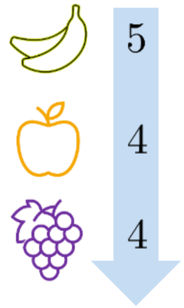


5. Build the FP-tree going through each transaction

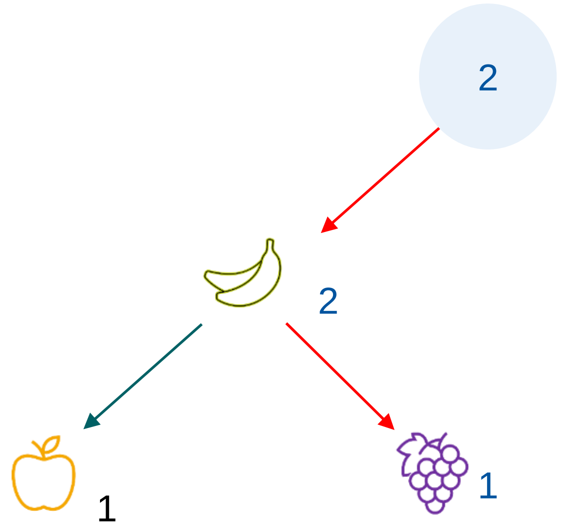


Constructing FP-Tree – Example

TID	Bought Fruits
1	{Banana, Apple}
2	**{Banana, Grapes}**
3	{Banana, Apple}
4	{Apple, Grapes}
5	{Banana, Grapes}
6	{Banana, Apple, Grapes}

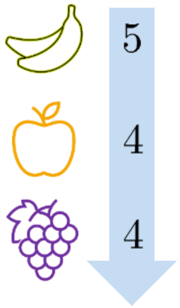


5. Build the FP-tree going through each transaction

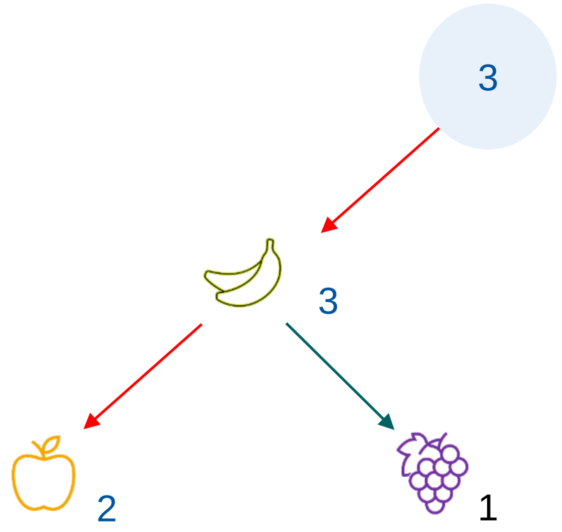


Constructing FP-Tree – Example

TID	Bought Fruits
1	{Banana, Apple}
2	{Banana, Grapes}
3	**{Banana, Apple}**
4	{Apple, Grapes}
5	{Banana, Grapes}
6	{Banana, Apple, Grapes}

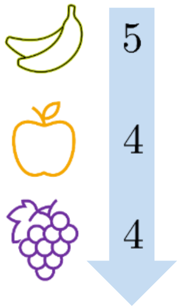


5. Build the FP-tree going through each transaction

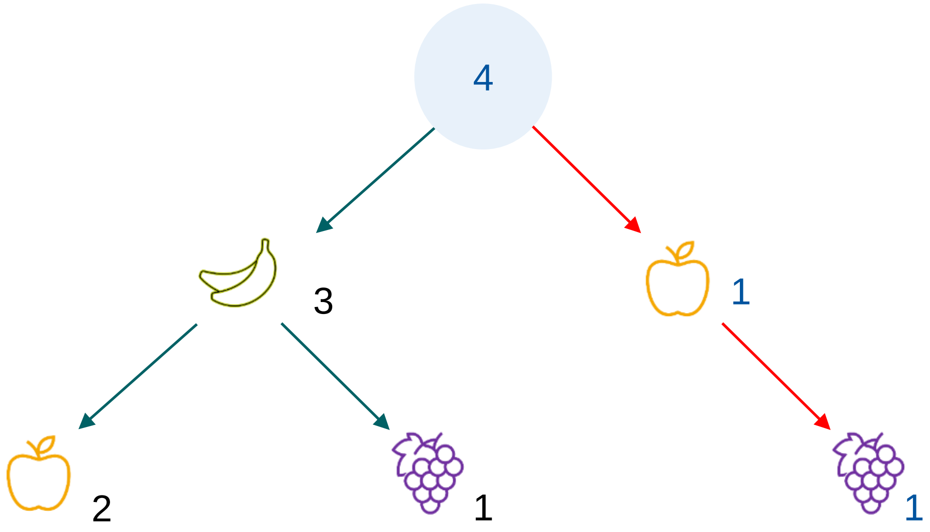


Constructing FP-Tree – Example

TID	Bought Fruits
1	{Banana, Apple}
2	{Banana, Grapes}
3	{Banana, Apple}
4	**{Apple, Grapes}**
5	{Banana, Grapes}
6	{Banana, Apple, Grapes}

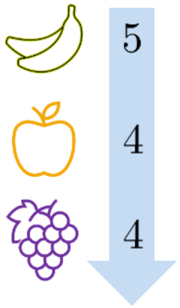


5. Build the FP-tree going through each transaction

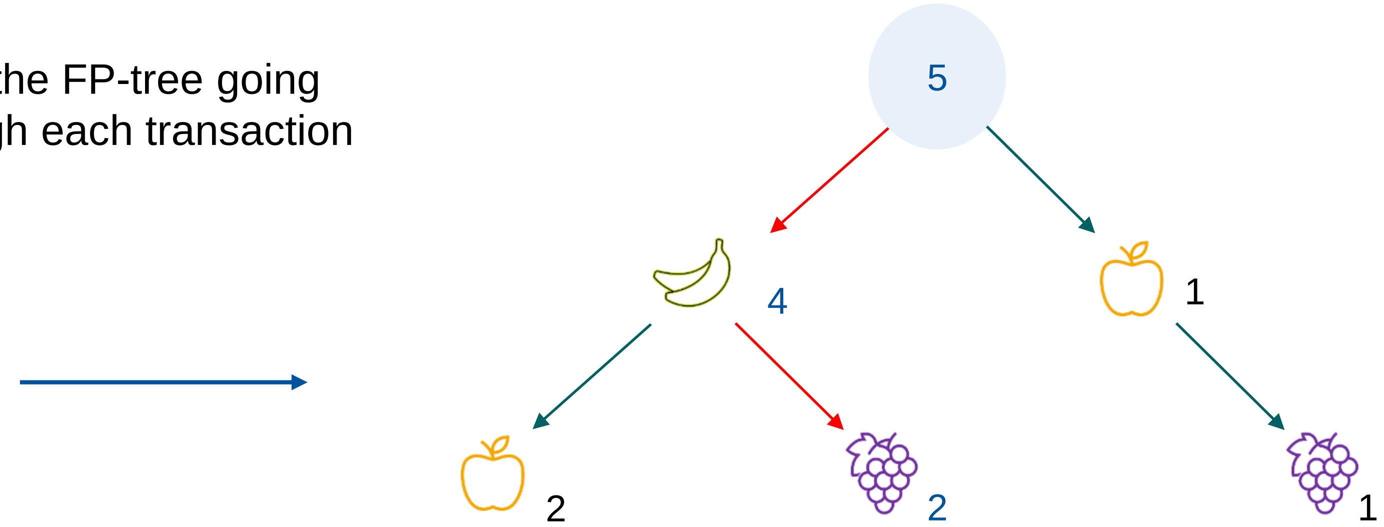


Constructing FP-Tree – Example

TID	Bought Fruits
1	{Banana, Apple}
2	{Banana, Grapes}
3	{Banana, Apple}
4	{Apple, Grapes}
5	**{Banana, Grapes}**
6	{Banana, Apple, Grapes}

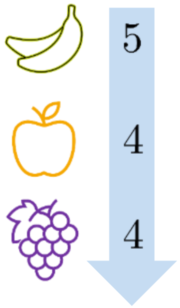


5. Build the FP-tree going through each transaction

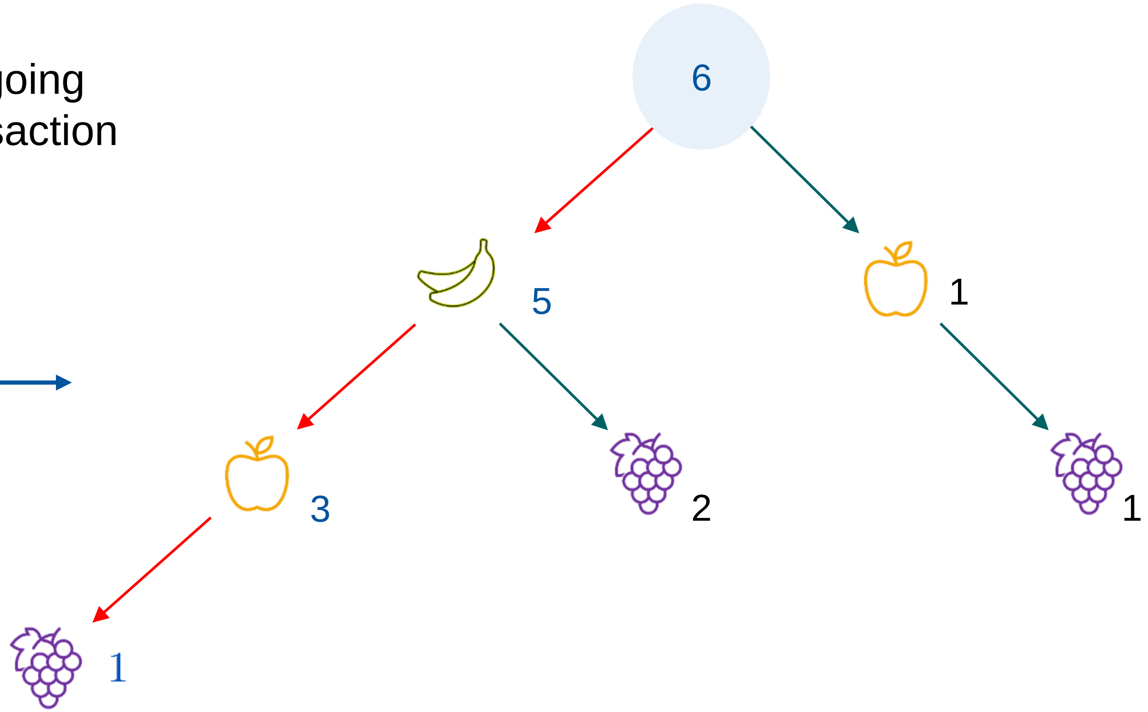


Constructing FP-Tree – Example

TID	Bought Fruits
1	{Banana, Apple}
2	{Banana, Grapes}
3	{Banana, Apple}
4	{Apple, Grapes}
5	{Banana, Grapes}
6	**{Banana, Apple, Grapes}**

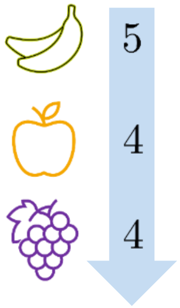


5. Build the FP-tree going through each transaction

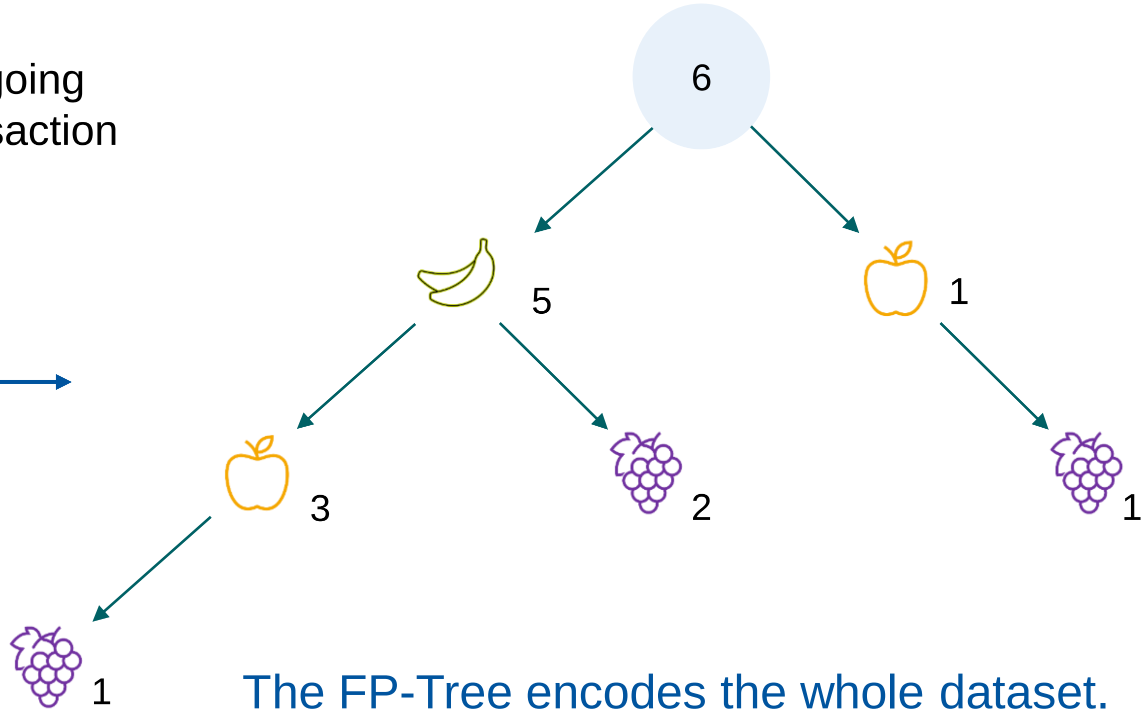


Constructing FP-Tree – Example

TID	Bought Fruits
1	{Banana, Apple}
2	{Banana, Grapes}
3	{Banana, Apple}
4	{Apple, Grapes}
5	{Banana, Grapes}
6	{Banana, Apple, Grapes}



5. Build the FP-tree going through each transaction

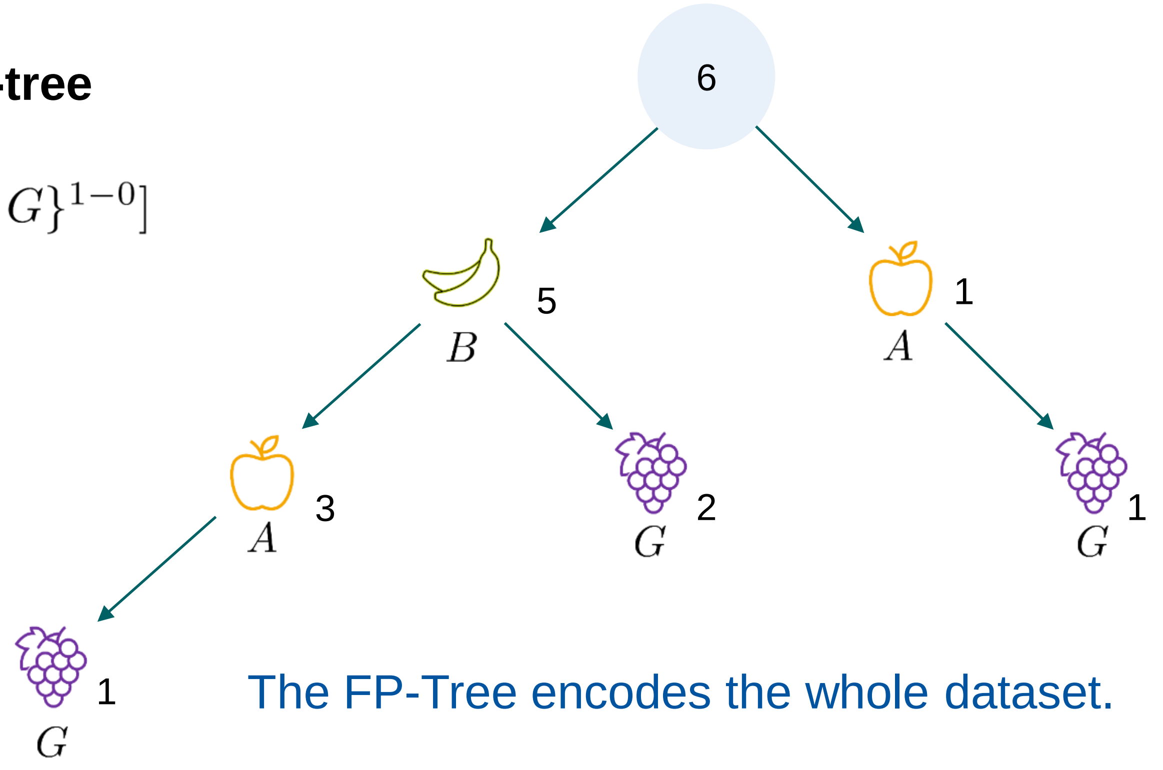


The FP-Tree encodes the whole dataset.

FP-Tree – Encodes The Dataset

We can read the transactions from the FP-tree

$$\begin{aligned}\mathcal{X} &= [\{B,A,G\}^{1-0}, \{B,A\}^{3-1}, \{B,G\}^{2-0}, \{A,G\}^{1-0}] \\ &= [\{B,A,G\}, \{B,A\}^2, \{B,G\}^2, \{A,G\}]\end{aligned}$$

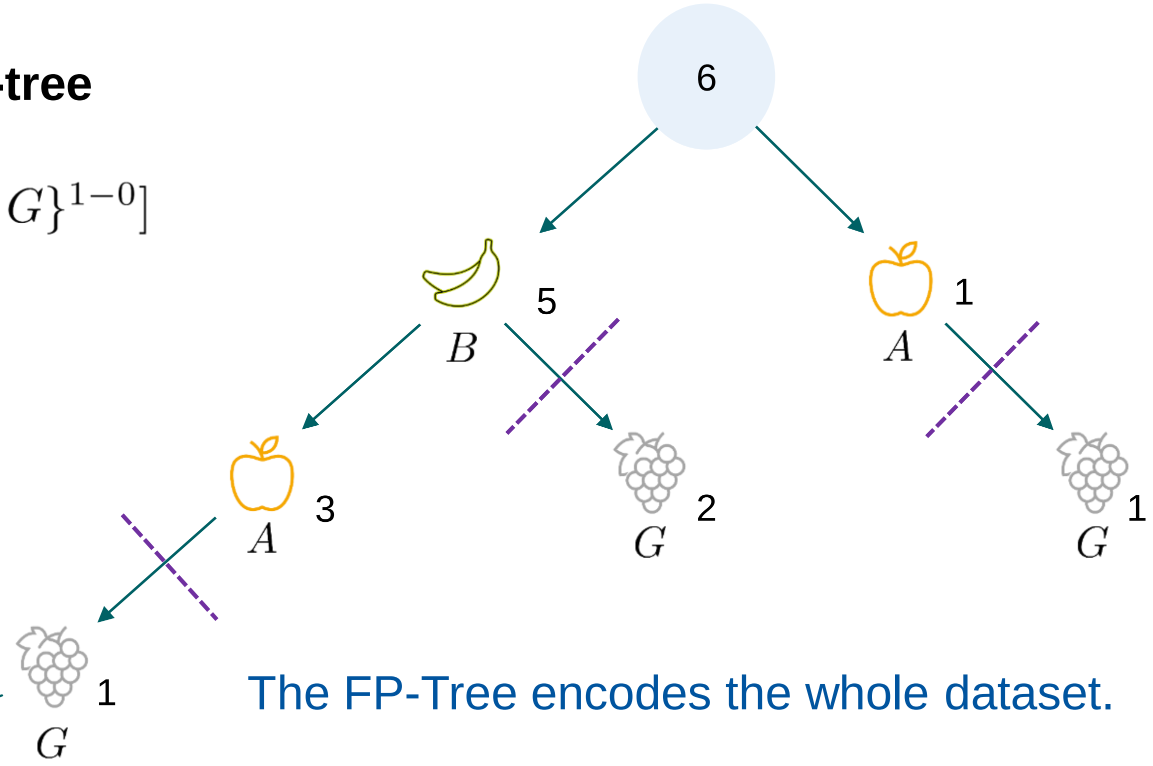


The FP-Tree encodes the whole dataset.

FP-Tree – Cannot Cut Naïvely

We can read the transactions from the FP-tree

$$\mathcal{X} = [\{B, A, G\}^{1-0}, \{B, A\}^{3-1}, \{B, G\}^{2-0}, \{A, G\}^{1-0}]$$
$$= [\{B, A, G\}, \{B, A\}^2, \{B, G\}^2, \{A, G\}]$$



The FP-Tree encodes the whole dataset.

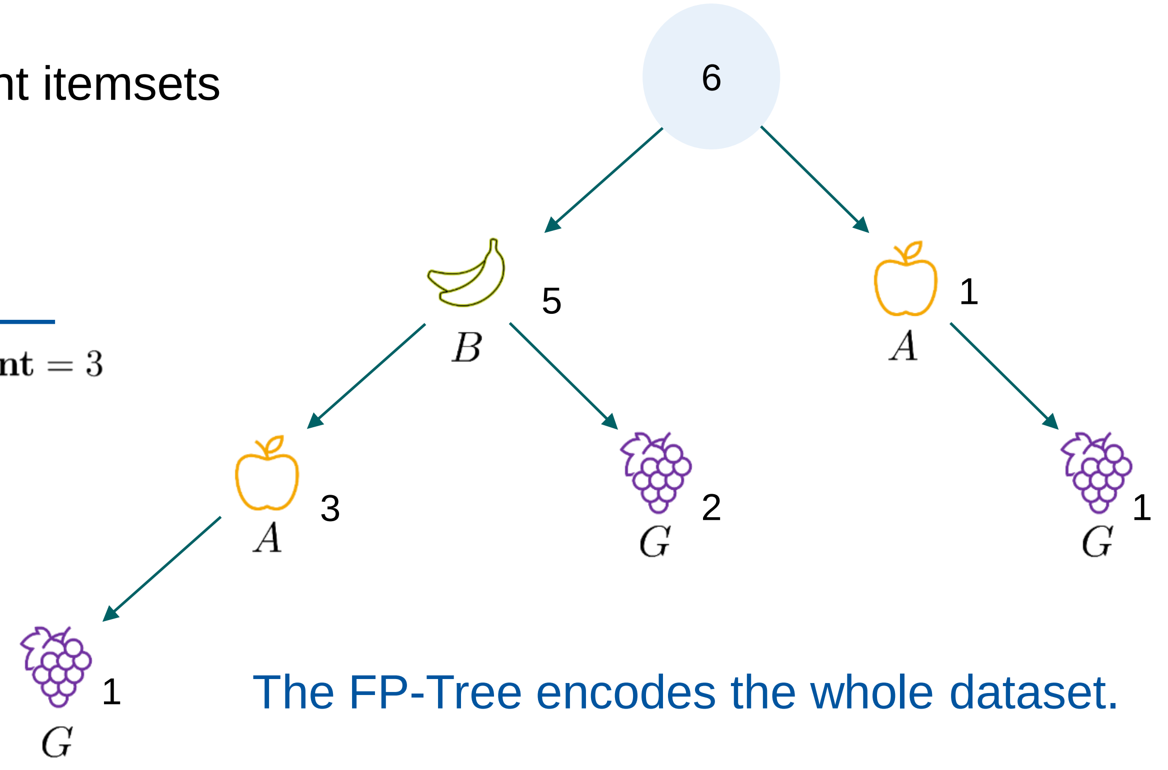
Even though *G* exists only once in this subtree, we can't cut it naively, because its support may be higher than the threshold (3)

FP-Tree – Frequent Itemsets

Next: 6. Mining the FP-tree to obtain frequent itemsets

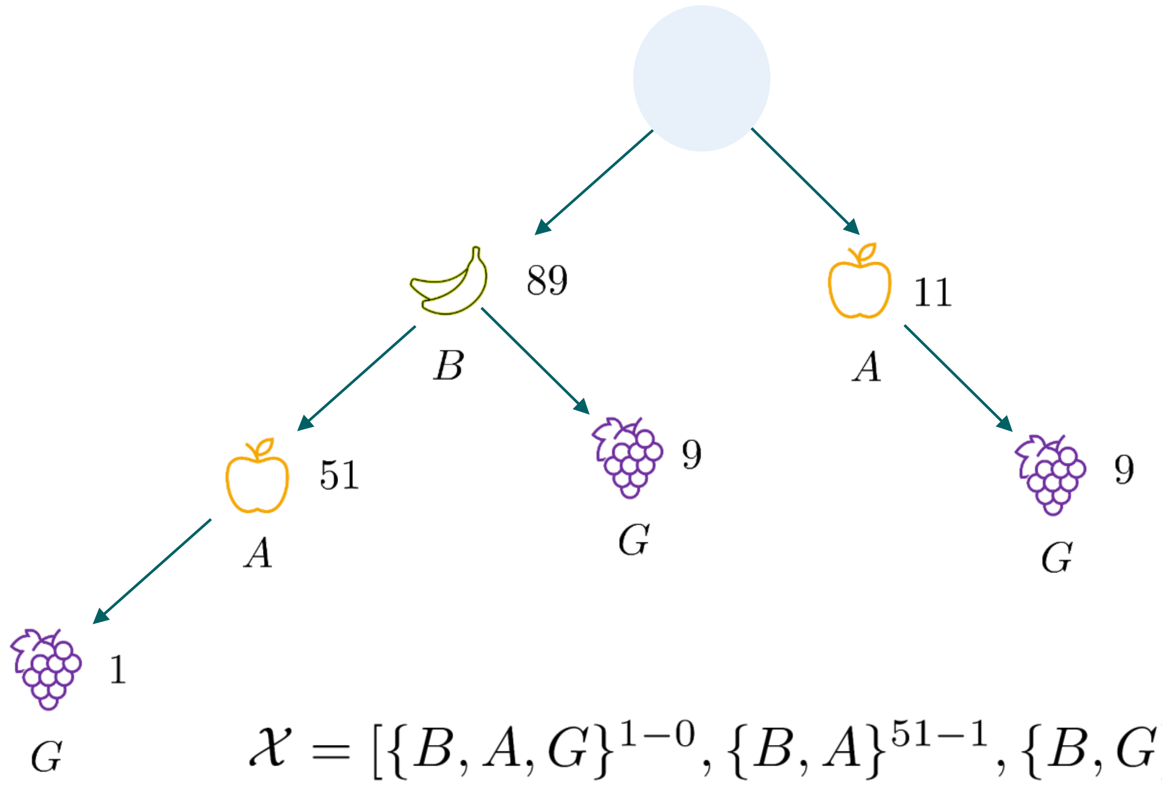
Frequent Itemsets	Support Count
{B}	5
{A}	4
{G}	4
{B, A}	3
{B, G}	3

$\mathbf{min\_sup\_count} = 3$



The FP-Tree encodes the whole dataset.

FP-Tree Encodes Dataset – Another Example



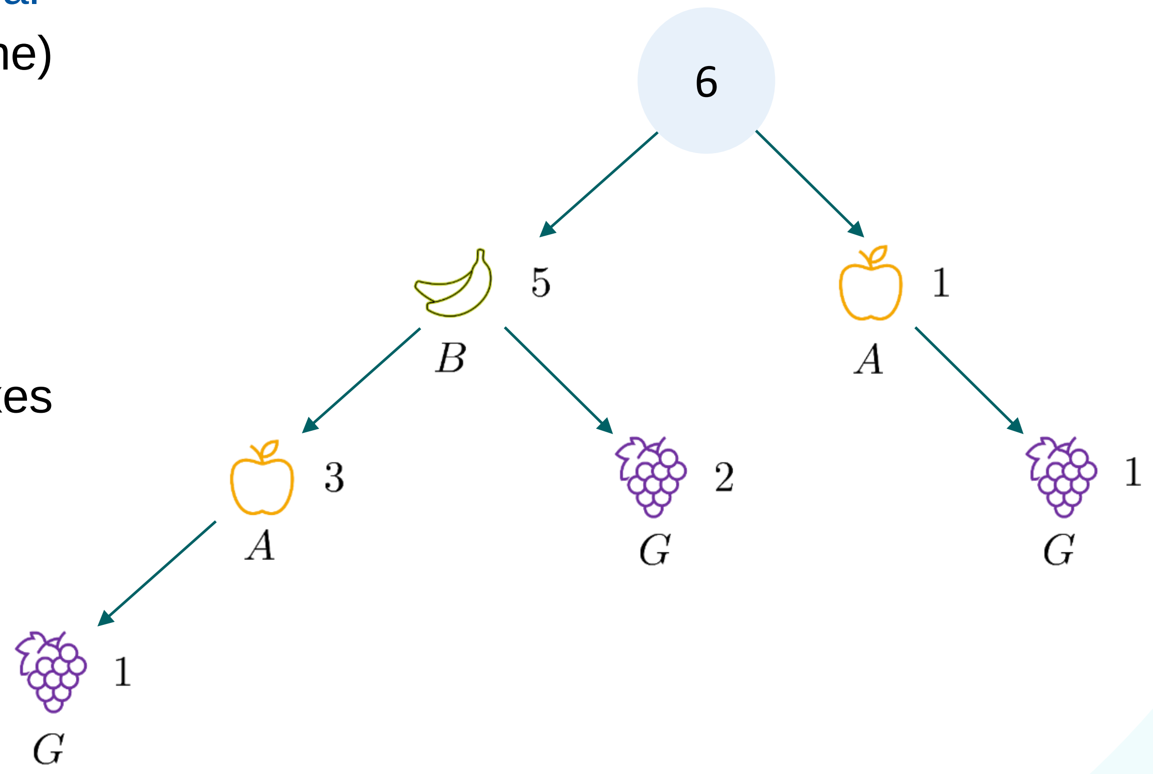
We can read the transactions from the FP-tree

$$\mathcal{X} = [\{B, A, G\}^{1-0}, \{B, A\}^{51-1}, \{B, G\}^{9-0}, \{B\}^{89-(51+9)}, \{A, G\}^{9-0}, \{A\}^{11-9}]$$

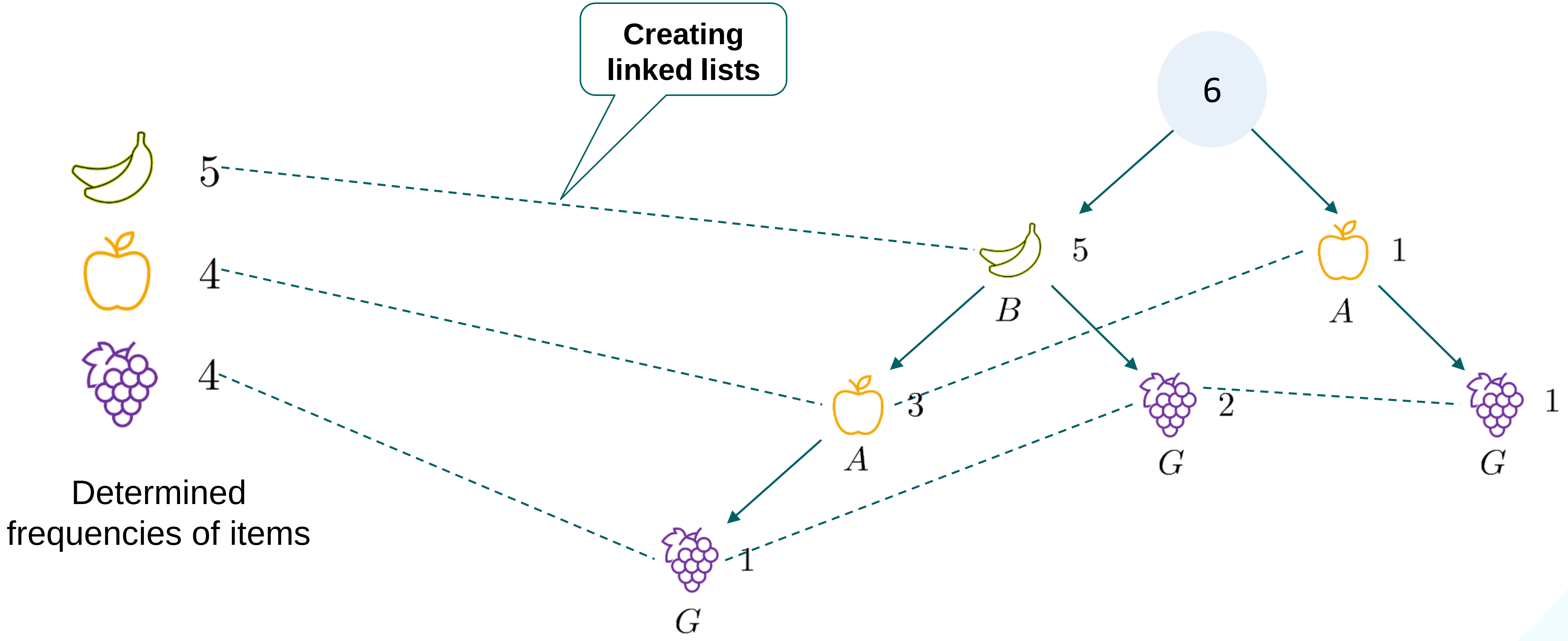
$$\mathcal{X} = [\{B, A, G\}^{1}, \{B, A\}^{50}, \{B, G\}^{9}, \{B\}^{29}, \{A, G\}^{9}, \{A\}^{2}]$$

Mining the FP-Tree – Overview

- For each frequent item, create a conditional FP-tree (starting with the least frequent one)
- The conditional FP-tree considers all transactions ending with this item
- Apply this recursively
- Due to recursion, we also consider postfixes that contain multiple elements
- The ordering ensures that postfixes are considered only once

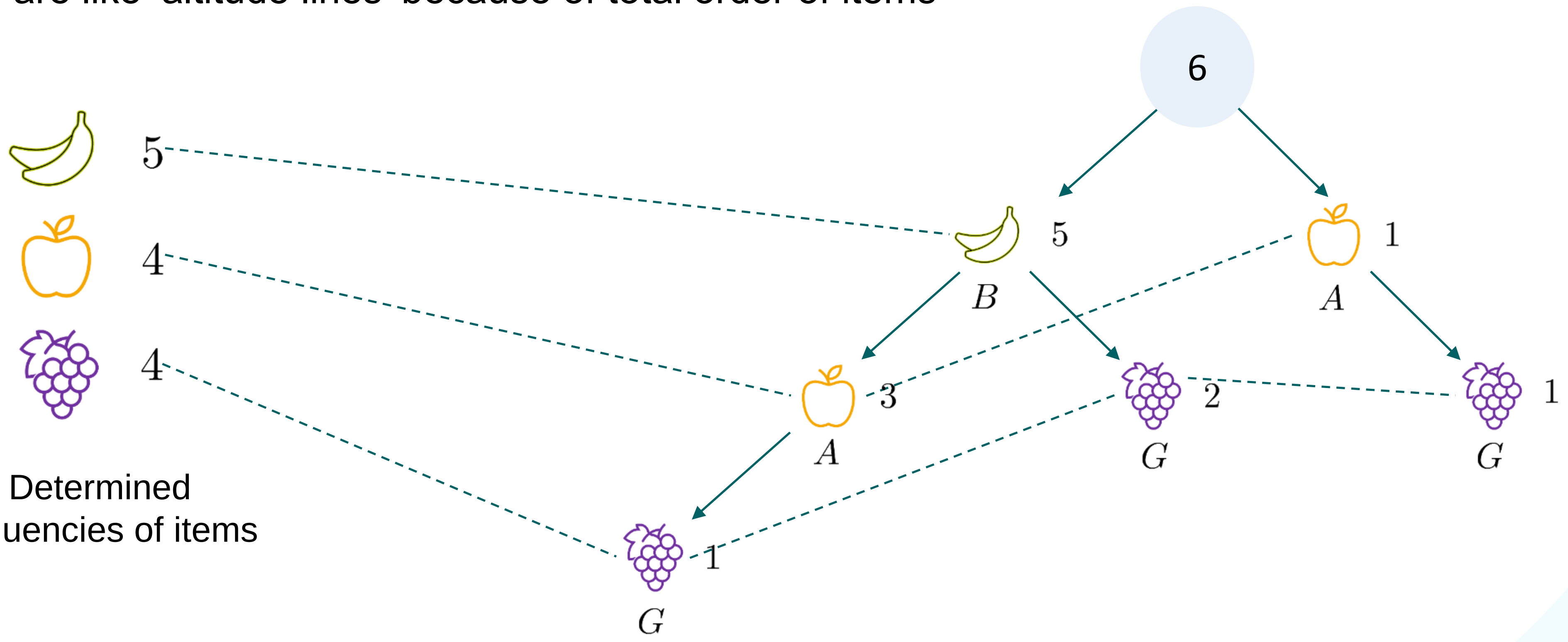


Node Links


Creating linked lists
6
5
4
4
5
B
1
A
3
A
2
G
1
G
1
G
Determined frequencies of items

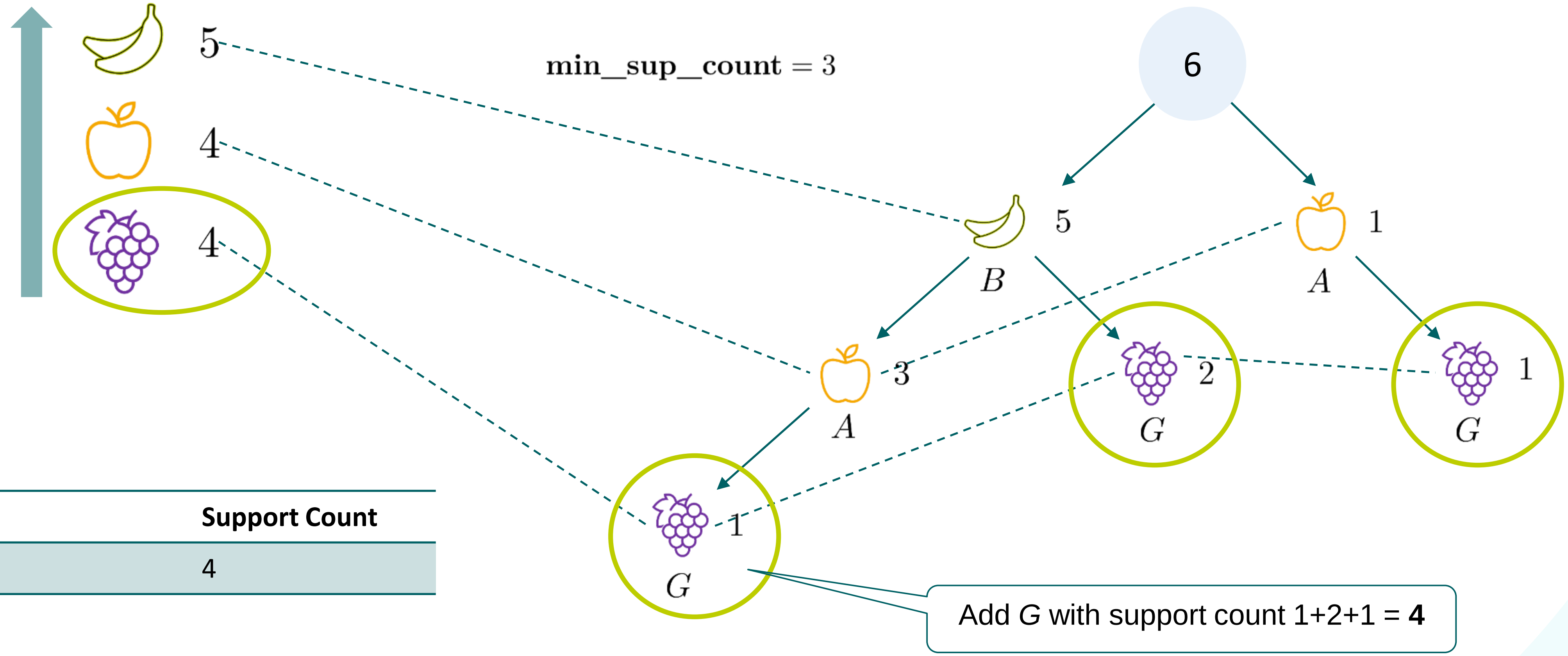
Node Links

Node links are like ‘altitude lines’ because of total order of items



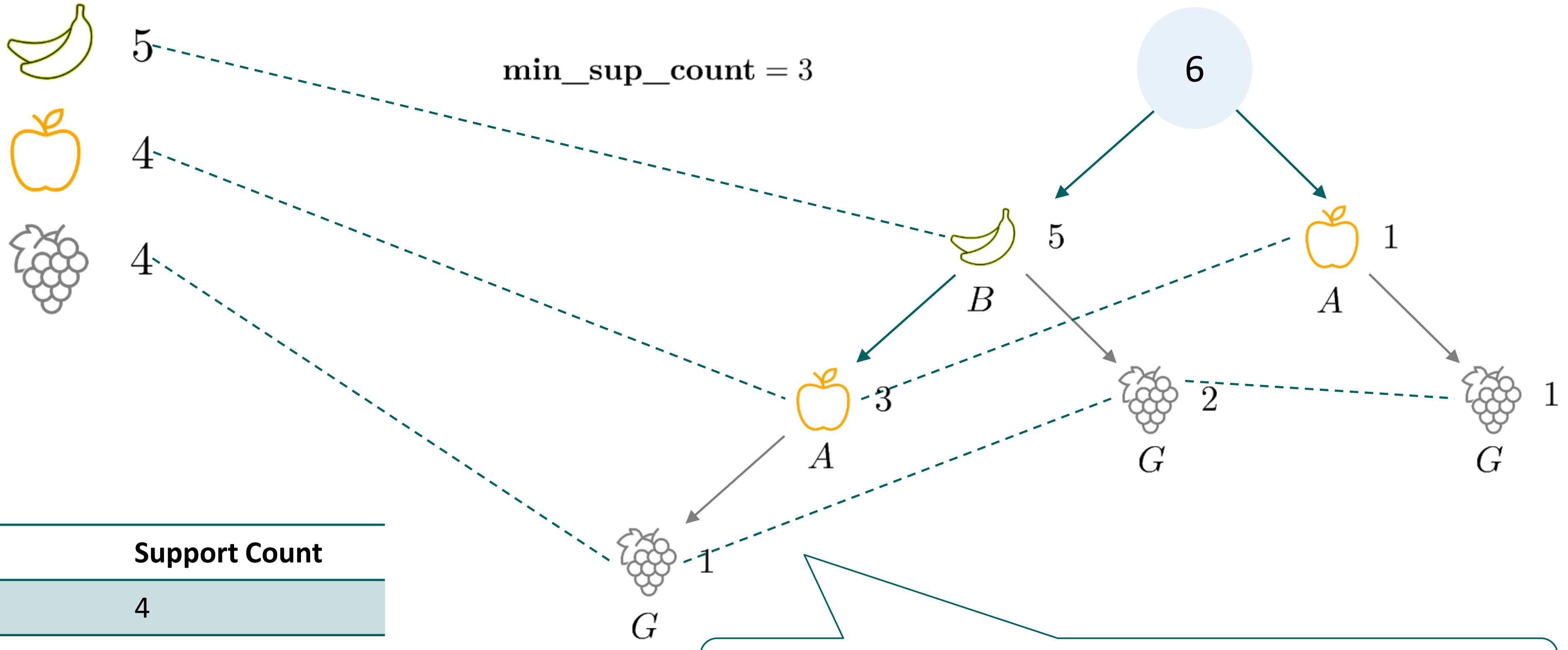
Determined frequencies of items

Consider Postfix G



Itemsets	Support Count
{G}	4

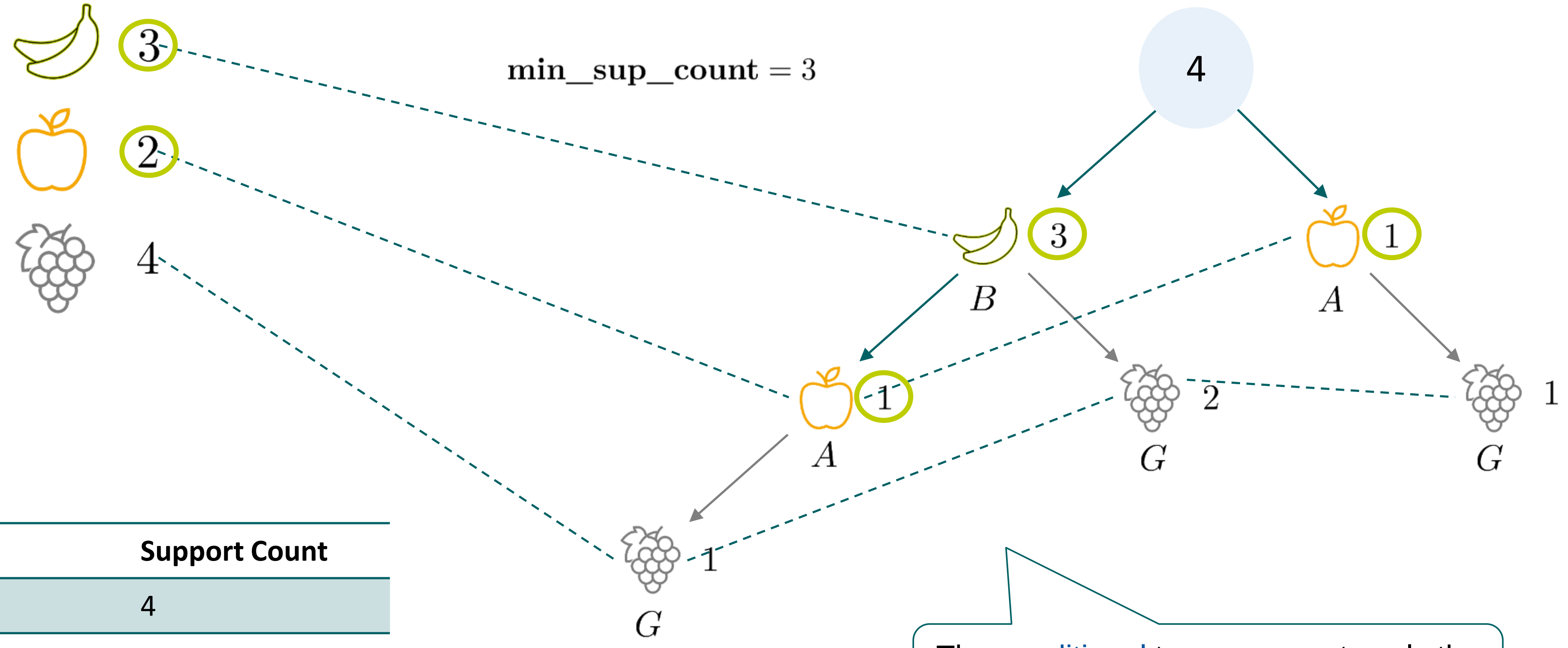
Consider Postfix G



Itemsets	Support Count
{G}	4

Now only consider **all the paths** leading from the root to *G* (representing transactions that include *G*)

Towards the Conditional FP-Tree for Postfix G



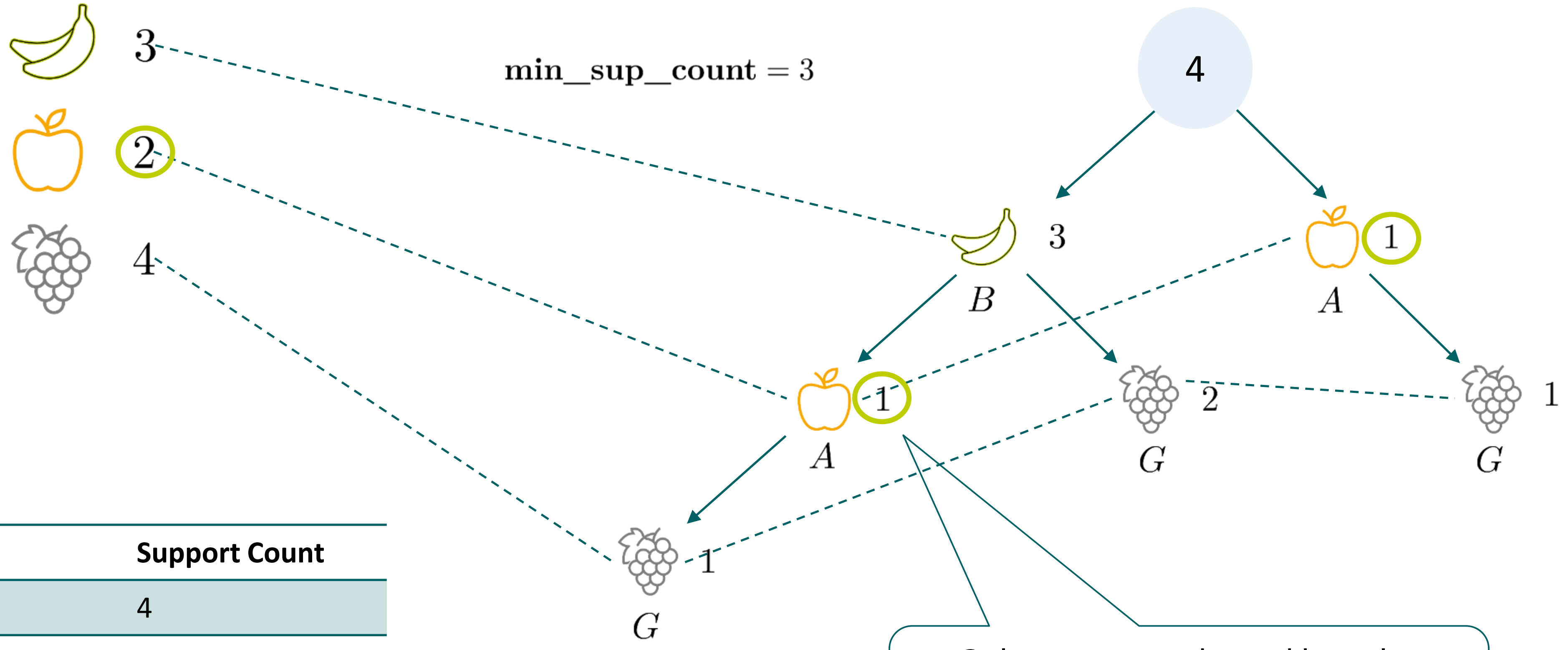
recomputed using linked lists

Itemsets	Support Count
{G}	4

The conditional tree represents only the transactions that contain postfix *G*

Towards the Conditional FP-Tree for Postfix G

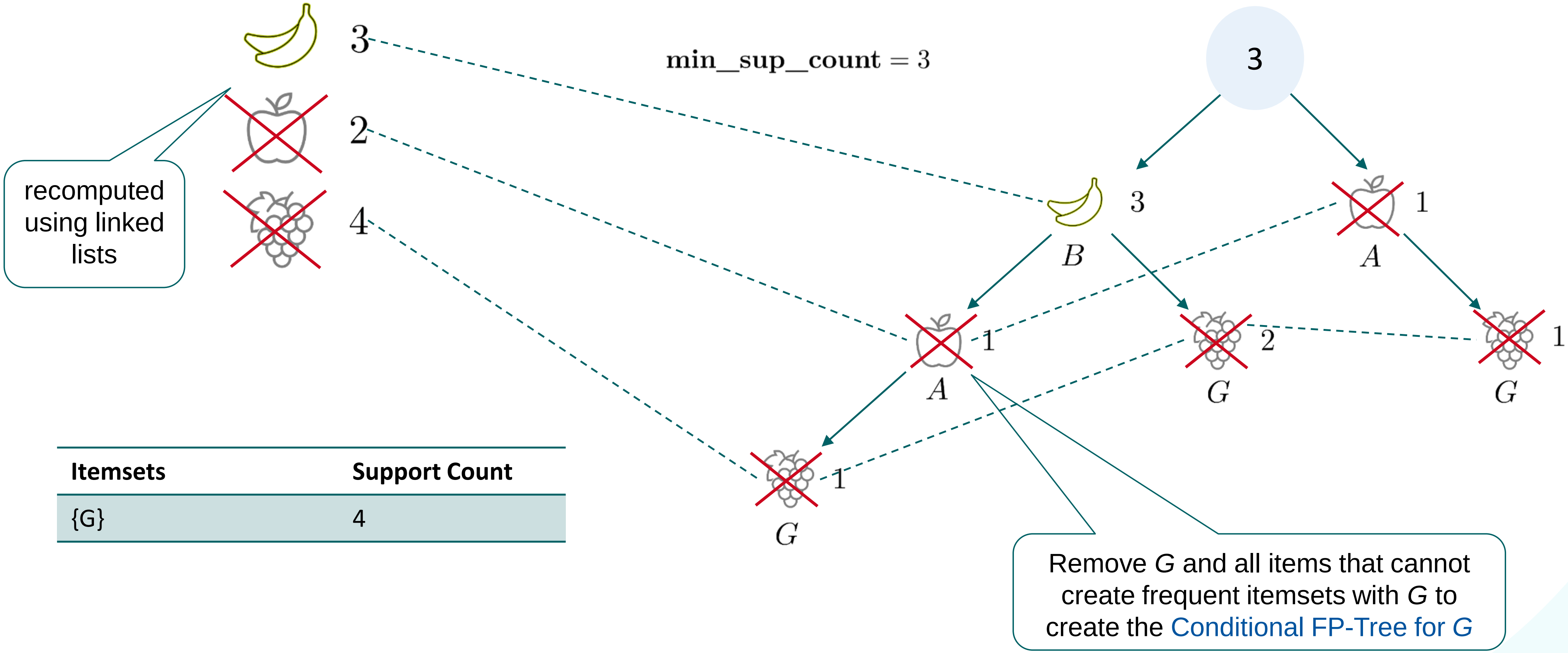
recomputed using linked lists



Itemsets	Support Count
{G}	4

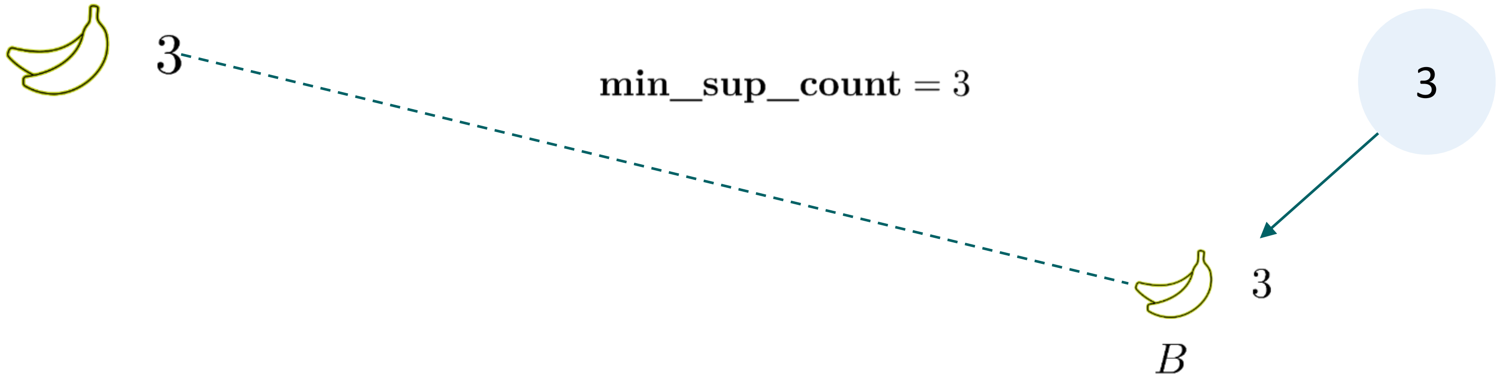
Only two transactions with apples contain *G* (less than **min_sup_count**), so {*A*,*G*} cannot be frequent

Towards the Conditional FP-Tree for Postfix G



Itemsets	Support Count
{G}	4

Conditional FP-Tree for Postfix G

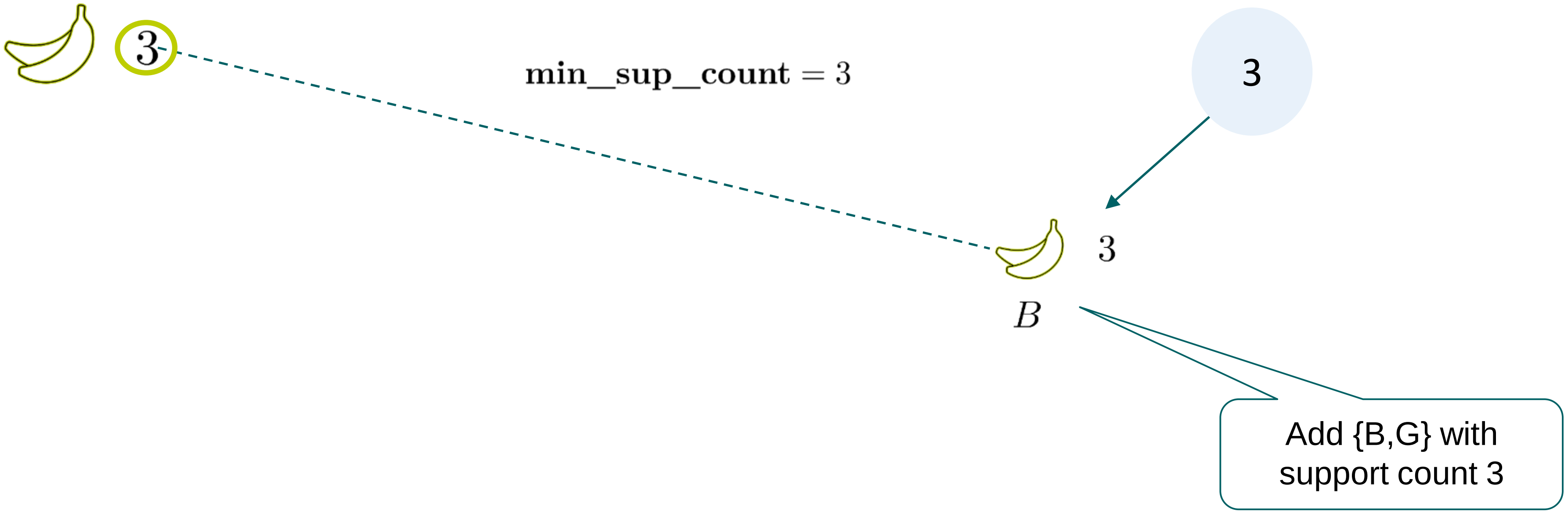


Itemsets	Support Count
{G}	4
{ … , G }	

recurse

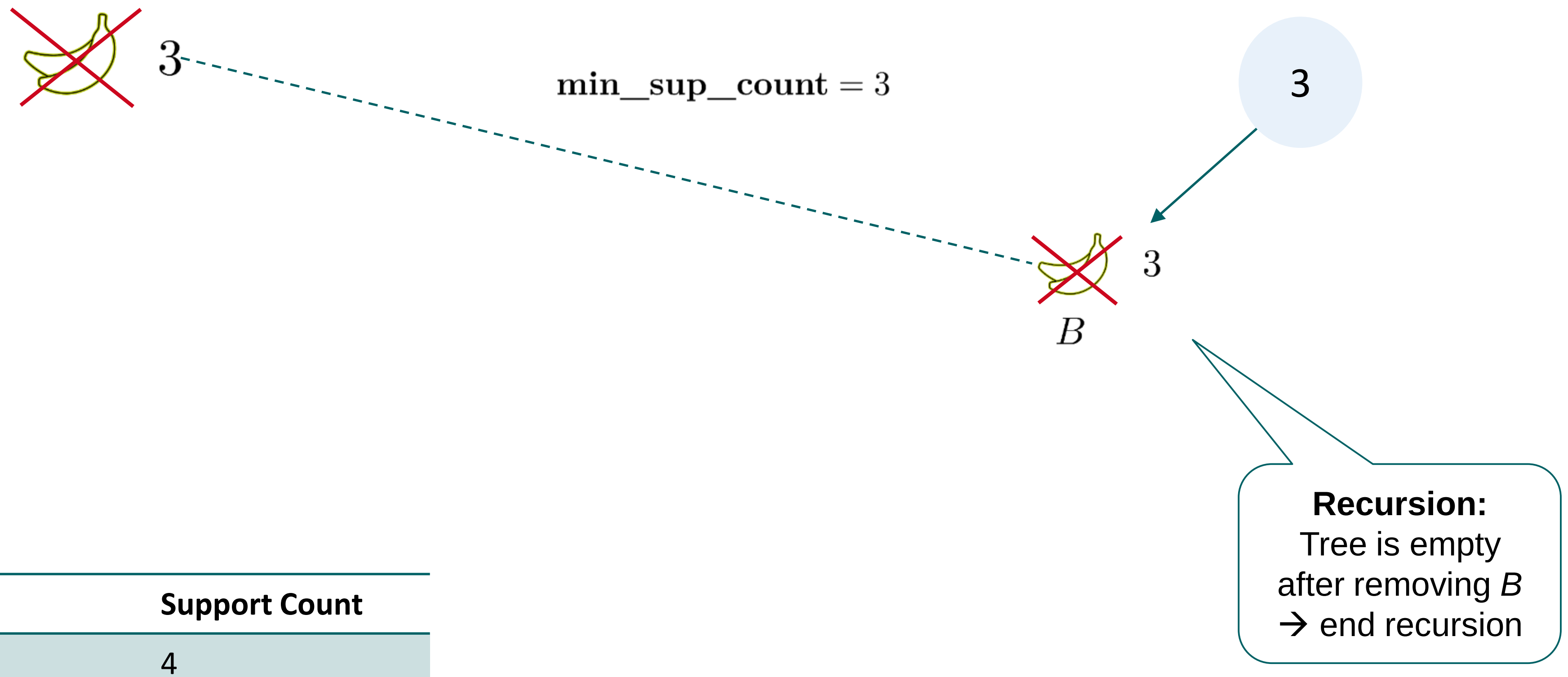
Mine the Conditional FP-Tree for G to find frequent itemsets that contain *G* (Recursion)

Conditional FP-Tree for Postfix G



Itemsets	Support Count
{G}	4
{B, G}	3

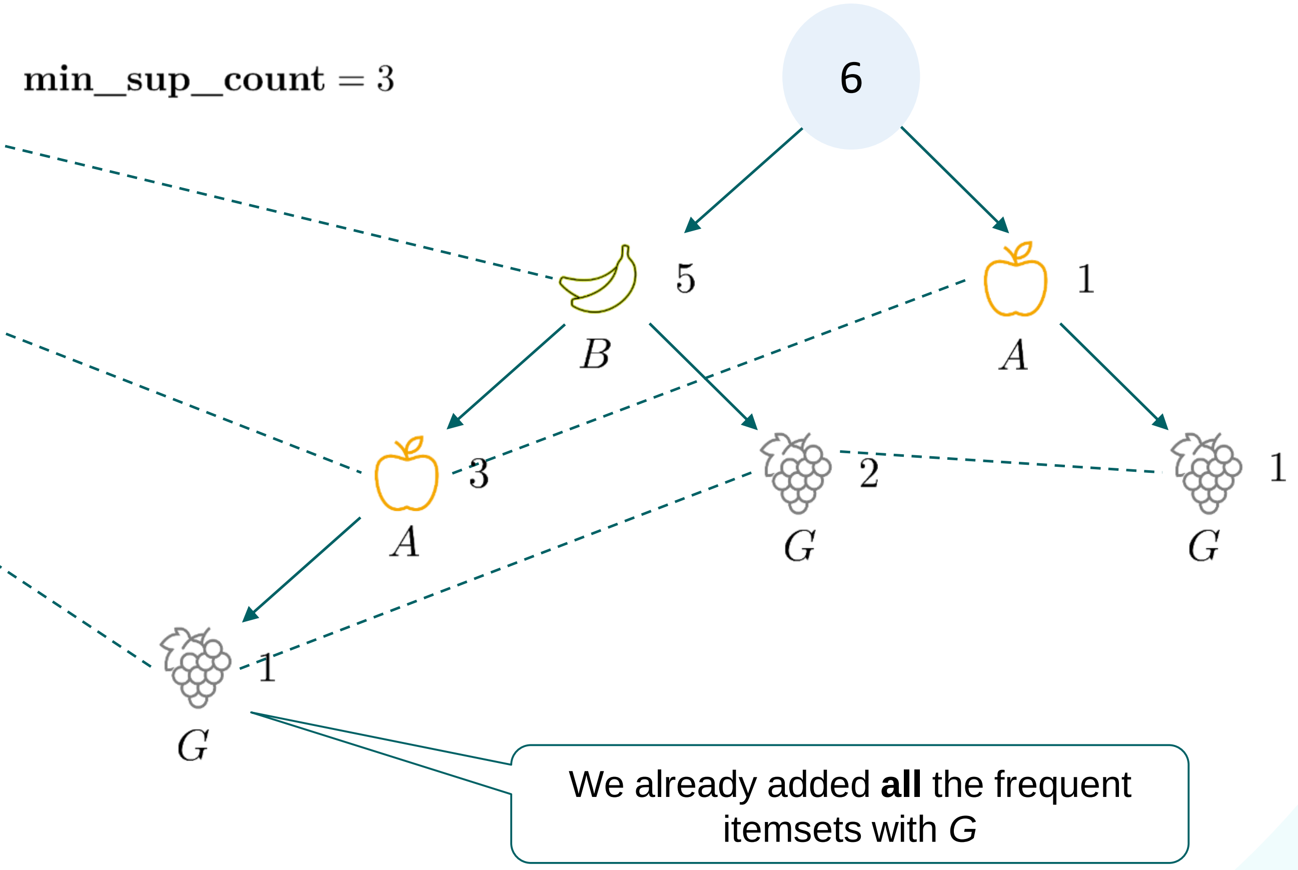
Conditional FP-Tree for Postfix G



Itemsets	Support Count
{G}	4
{B, G}	3

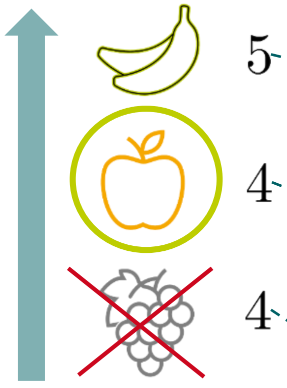
Consider Postfix A

Determined frequencies of items

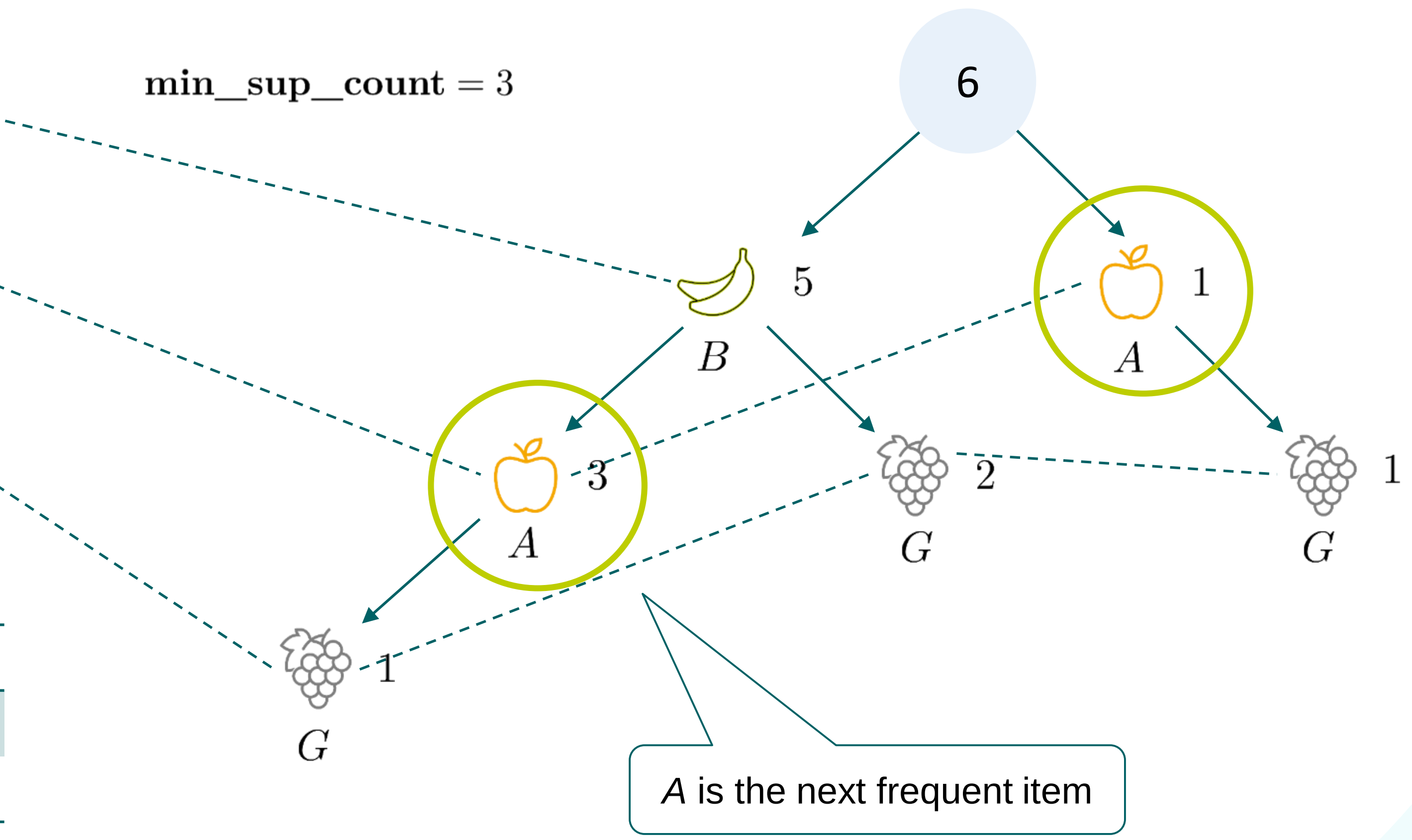


Itemsets	Support Count
{G}	4
{B, G}	3

Consider Postfix A

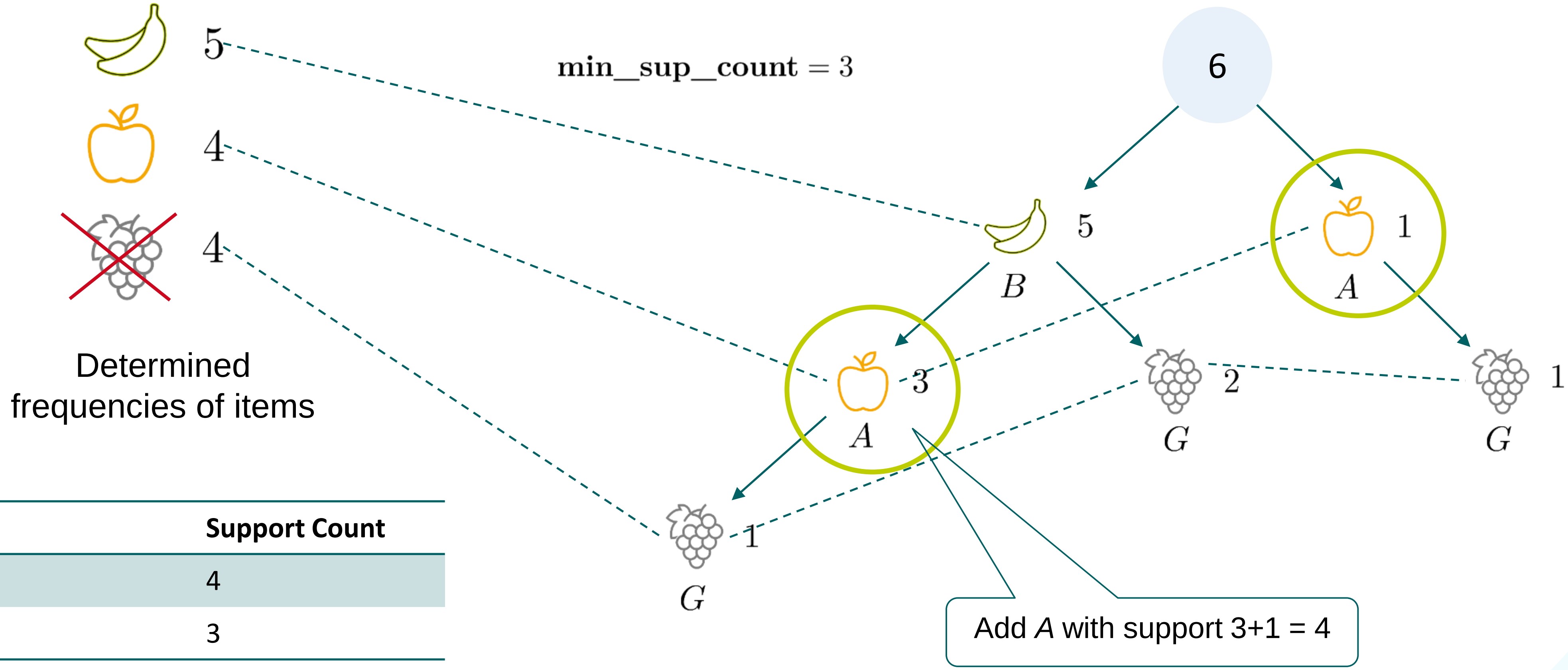


Determined frequencies of items



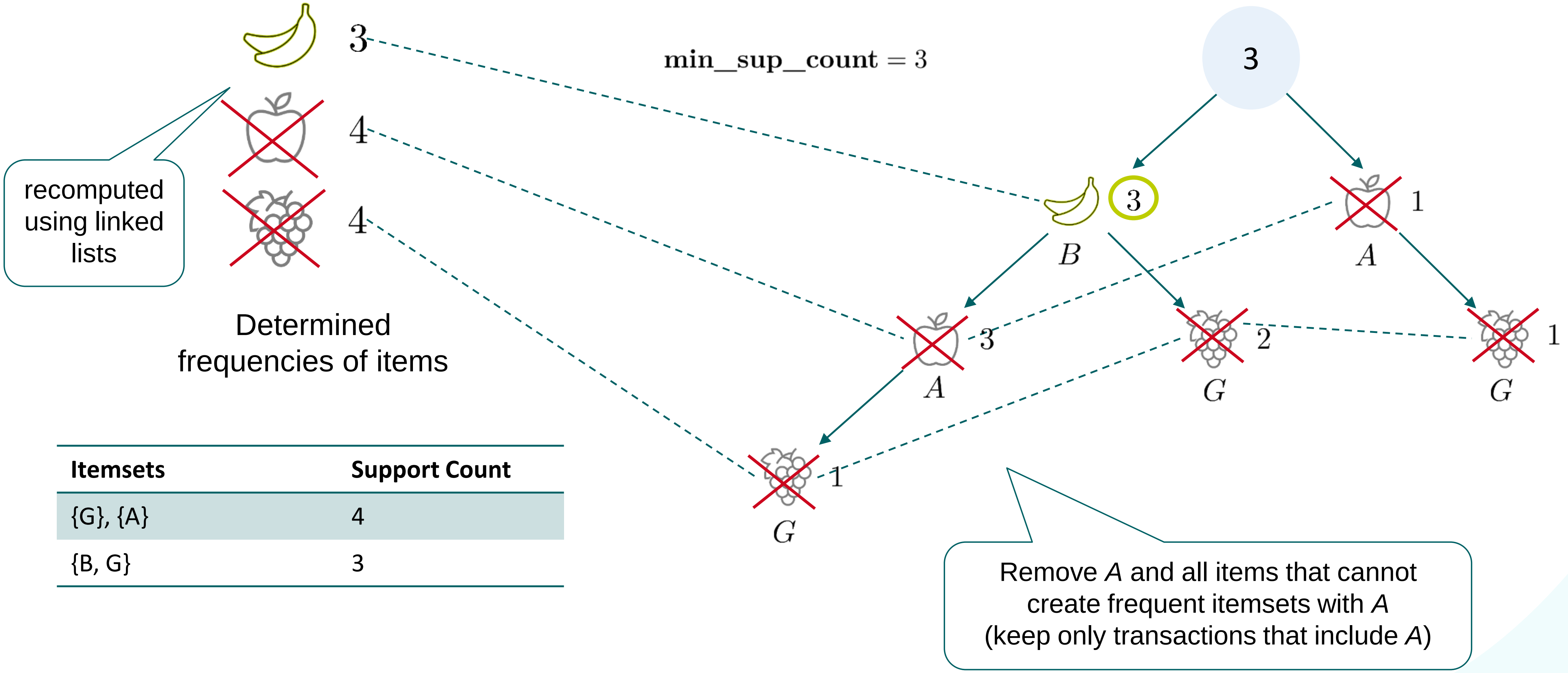
Itemsets	Support Count
{G}	4
{B, G}	3

Consider Postfix A



Itemsets	Support Count
{G}, {A}	4
{B, G}	3

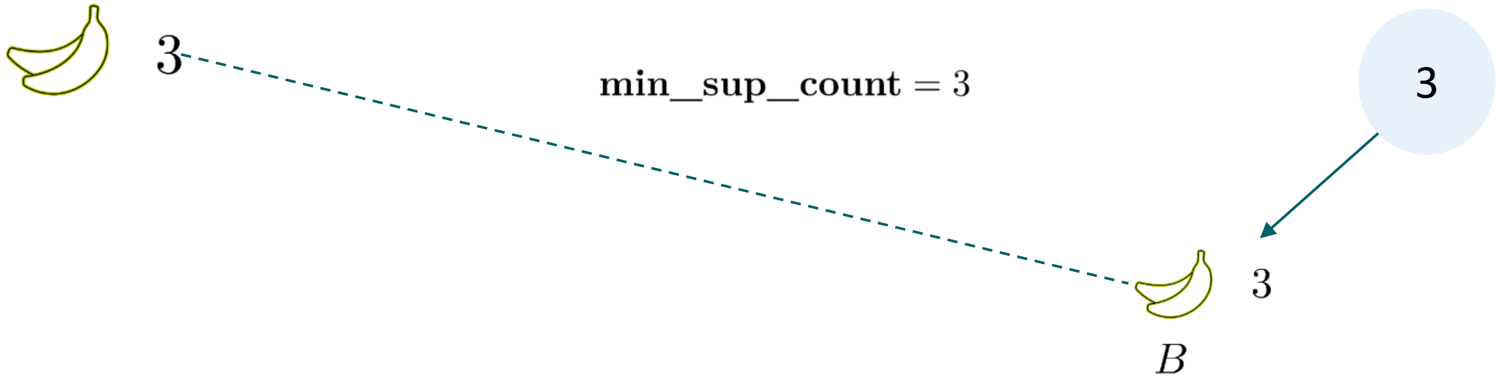
Towards the Conditional FP-Tree for Postfix A



Determined frequencies of items

Itemsets	Support Count
{G}, {A}	4
{B, G}	3

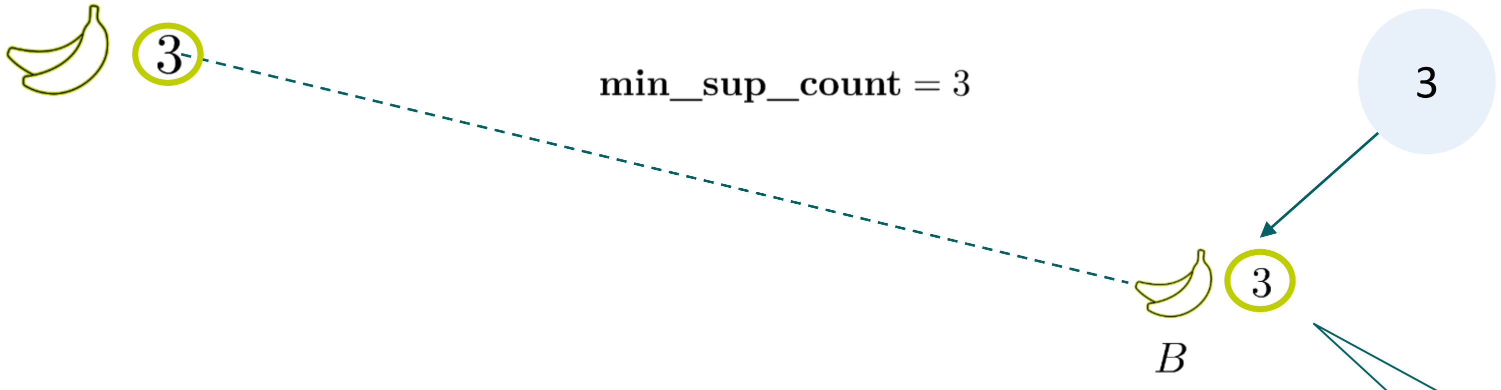
Towards Conditional FP-Tree for Postfix A



Determined
frequencies of items

Itemsets	Support Count
{G}, {A}	4
{B, G}	3

Conditional FP-Tree for Postfix A

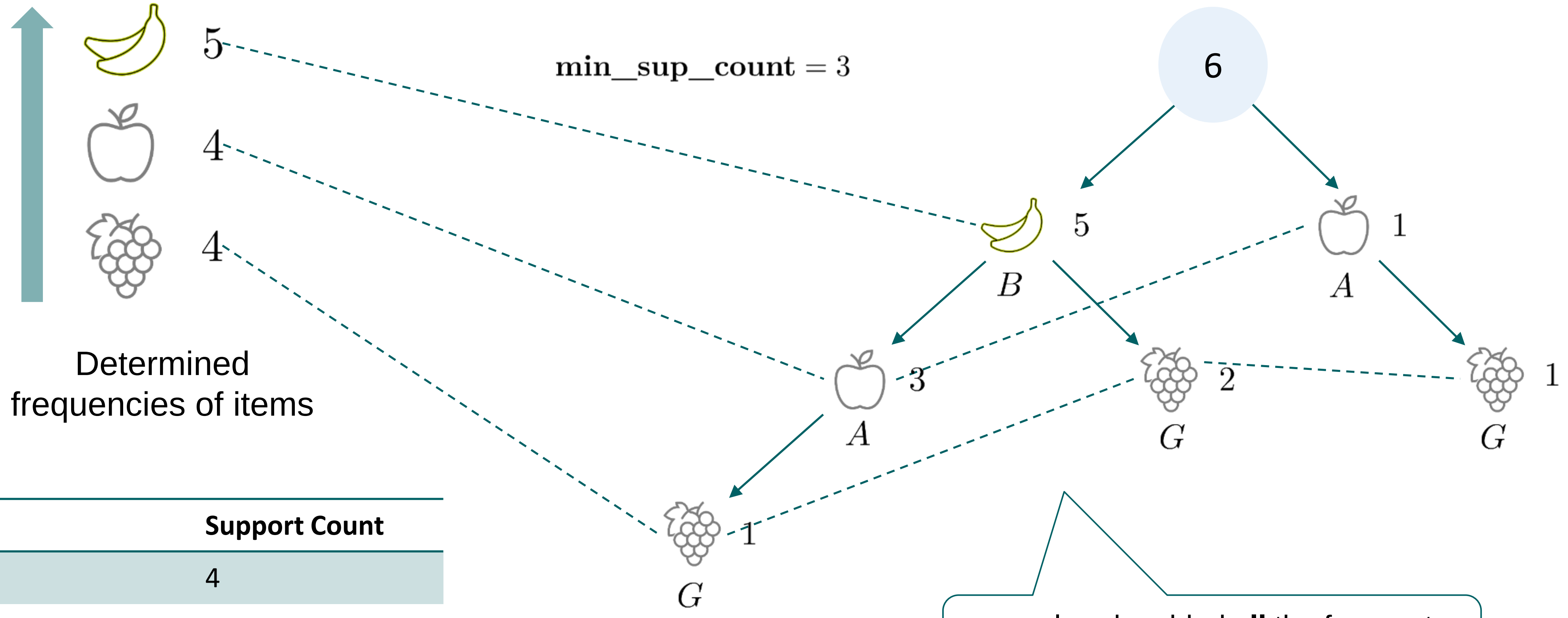


Recursion:
Add {*B*, *A*} with support 3 (then end recursion)

Determined frequencies of items

Itemsets	Support Count
{G}, {A}	4
{B, G}, {B, A}	3

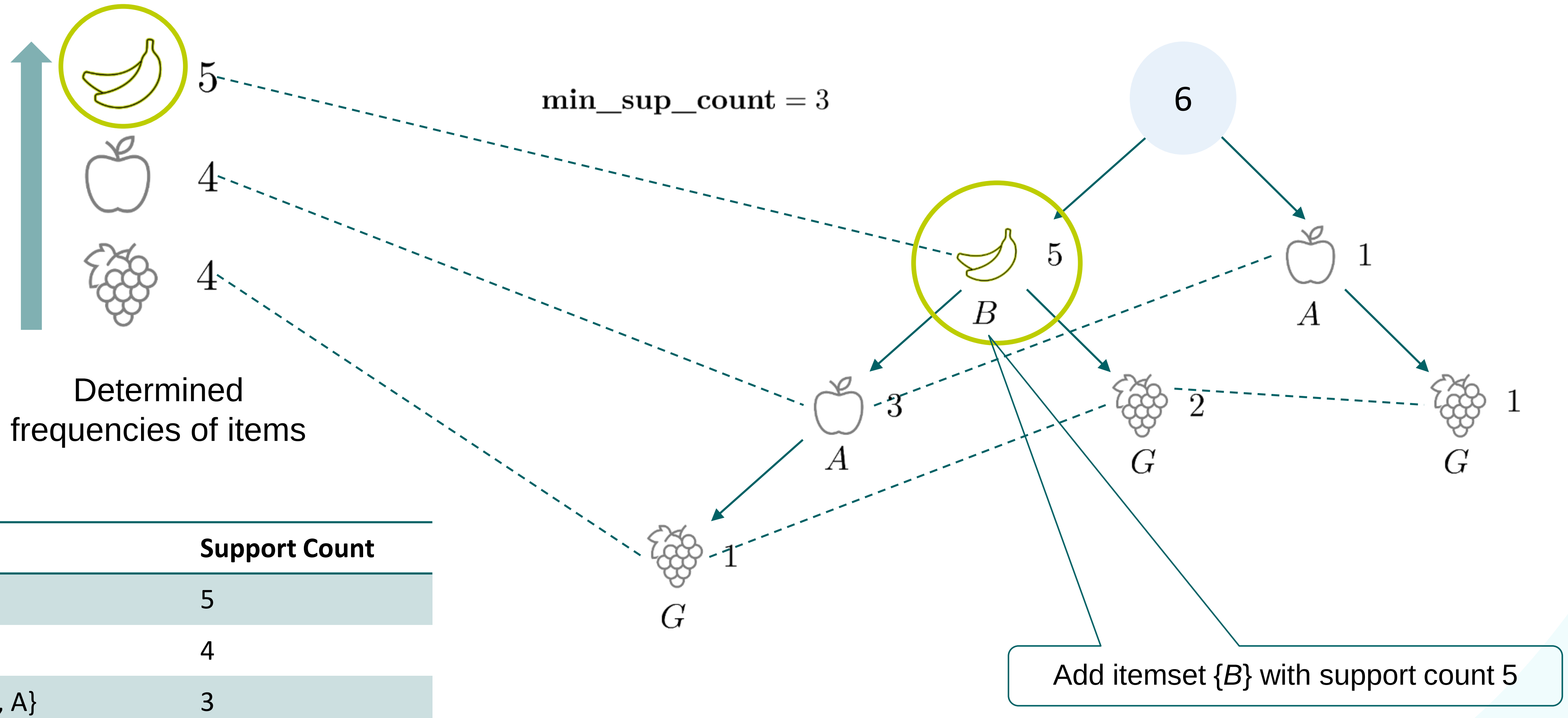
Consider Postfix B



Itemsets	Support Count
{G}, {A}	4
{B, G}, {B, A}	3

we already added **all** the frequent itemsets with *G* and *A*

Consider Postfix B

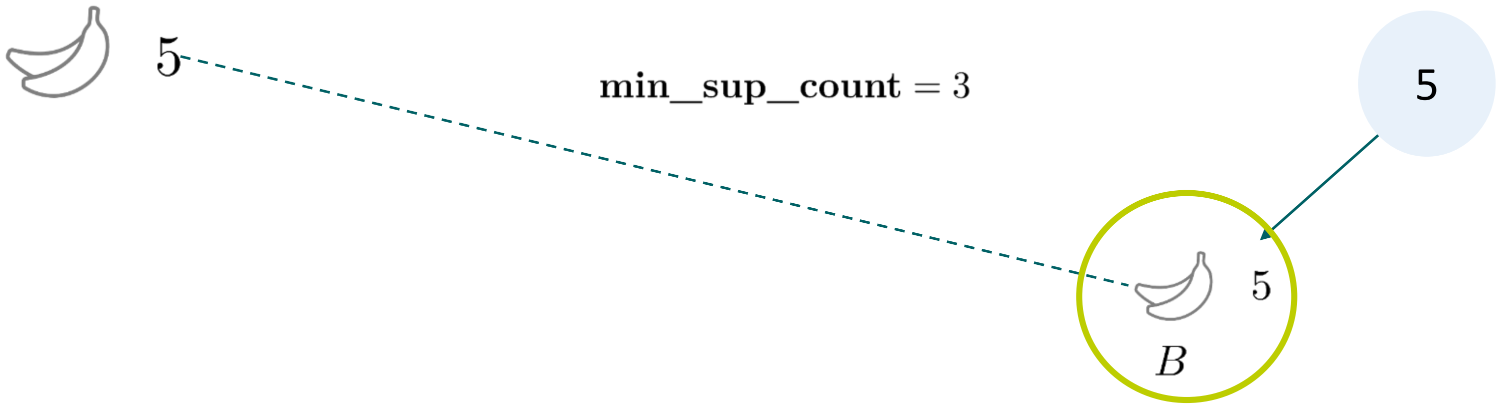


Determined frequencies of items

Itemsets	Support Count
{B}	5
{G}, {A}	4
{B, G}, {B, A}	3

Towards Conditional FP-Tree for Postfix B

$$\mathbf{min\_sup\_count} = 3$$

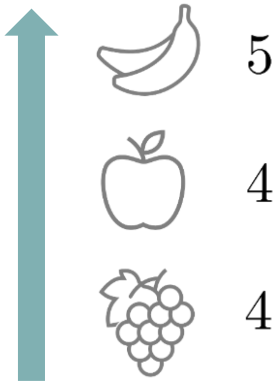


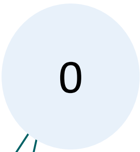
Now only consider **all the paths** leading to *B* (transactions including *B*)

Determined frequencies of items

Itemsets	Support Count
{B}	5
{G}, {A}	4
{B, G}, {B, A}	3

Conditional FP-Tree for Postfix B

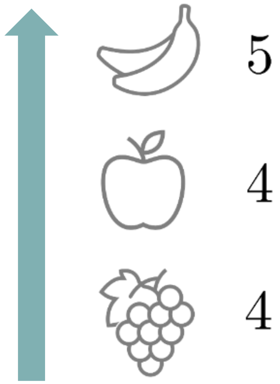


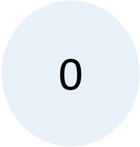


Conditional FP-tree is empty, i.e., no additional frequent itemsets with *B*

Itemsets	Support Count
{B}	5
{G}, {A}	4
{B, G}, {B, A}	3

Conditional FP-Tree for Postfix B

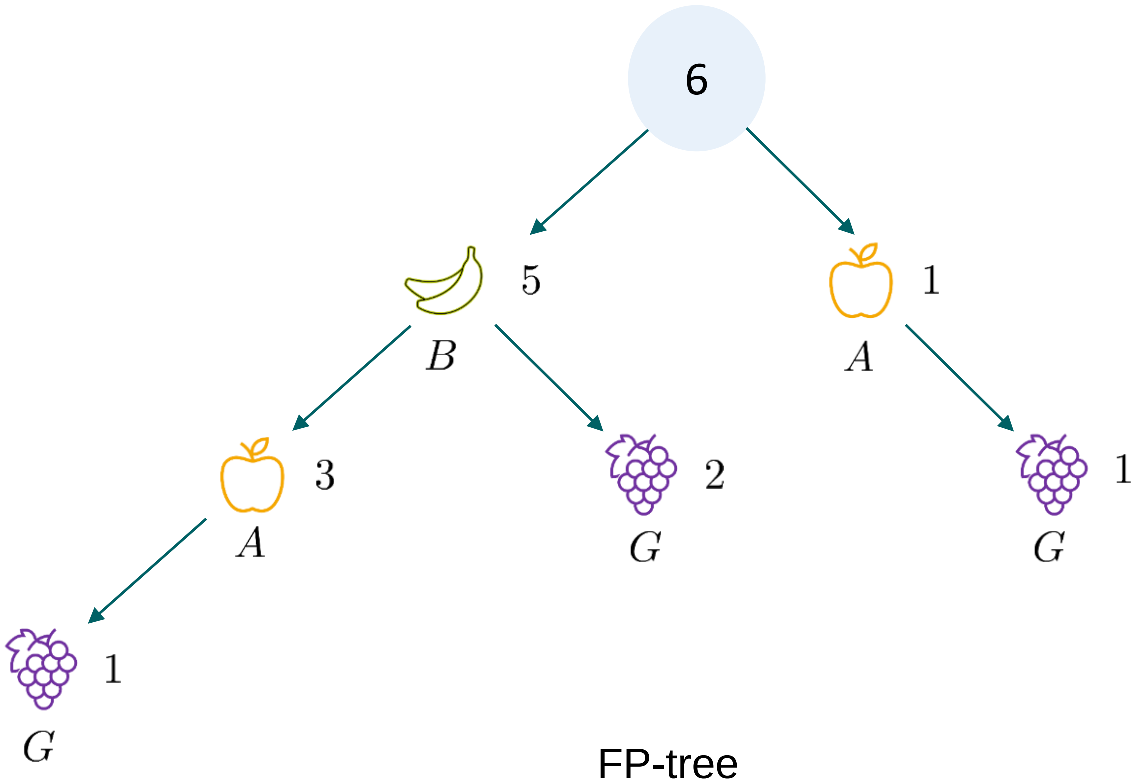




No more frequent items to consider
→algorithm terminates

Itemsets	Support Count
{B}	5
{G}, {A}	4
{B, G}, {B, A}	3

All Frequent Itemsets Generated



FP-tree

Itemsets	Support Count
{B}	5
{G}, {A}	4
{B, G}, {B, A}	3

Frequent itemsets mined

FP-Growth Algorithm – Summary

- Idea: frequent pattern growth based on FP-tree
- Method:
  - Construct the FP-tree from the dataset (previous video)
  - For each frequent item, construct its conditional pattern-base, and then its conditional FP-tree
  - Recursively repeat the process on each newly created conditional FP-tree until the tree is empty

FP-Growth Algorithm – Summary

- Advantages:
  - ✓ Only two passes through the dataset are needed (when constructing the tree)
  - ✓ Avoiding testing many hopeless candidates
  - ✓ Very fast when FP-tree fits in main memory

- However: approach has problems when FP-tree is too large to fit into memory

Frequent Itemsets – Summary

- Pattern mining is a form of unsupervised learning
- Frequent itemsets are the basis for finding patterns (ideas can be transferred to other patterns)
- Two well-known algorithms using generally applicable concepts:
  - Apriori algorithm
  - FP-growth algorithm
- Outlook
  - There may be many frequent “patterns”
  - How to determine which ones are surprising / interesting?

Association Rules – Preview

(one of the topics of the next lecture)

- $\{\text{Cheese, Bread}\} \Rightarrow \{\text{Milk}\}$
  People that buy Cheese and Bread also tend to buy Milk.
- $\{\text{Track1, Track2}\} \Rightarrow \{\text{Track3}\}$
  Students that take the Track 1 and Track 2 modules of BridgingAI also tend to take the Track 3 courses. (We hope you do!)
- $\{\text{Bitburger}\} \Rightarrow \{\text{Heineken, Palm}\}$
  People that buy Bitburger beer tend to buy both Heineken and Palm beer.
- $\{\text{Carbonara, Margherita }\} \Rightarrow \{\text{Espresso, Tiramisu}\}$
  People that buy Carbonara and Margherita also tend to buy Espresso and Tiramisu.
- $\{\text{part-245, part-345, part-456}\} \Rightarrow \{\text{part-372}\}$
  When Parts 245, 345, and 456 are replaced, then often also Part 372 is replaced.