Data Mining

Chapter 6: Recommender Systems

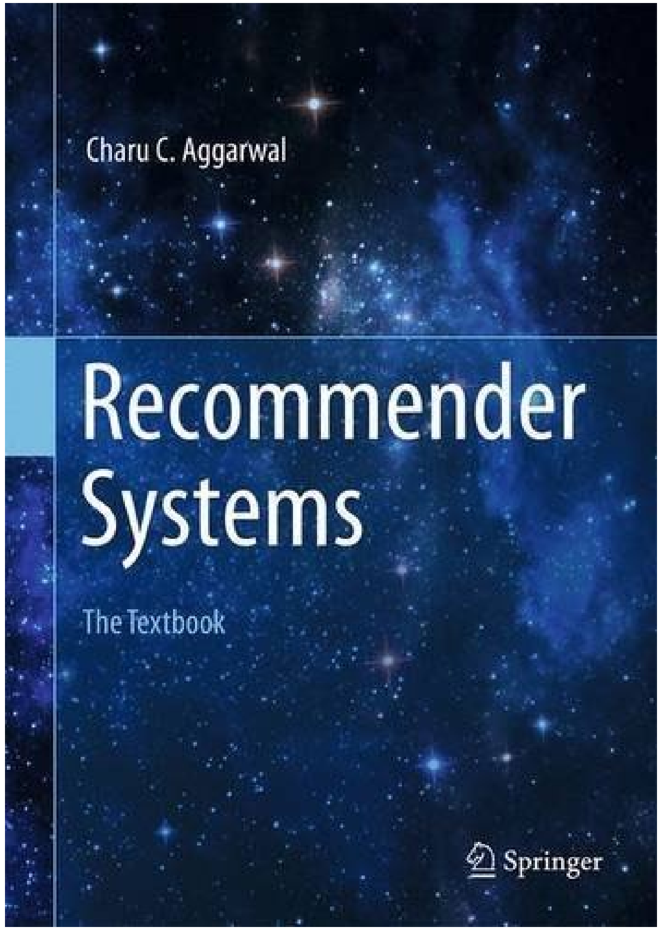
Yunming Ye, Baoquan Zhang

School of Computer Science

Harbin Institute of Technology, Shenzhen

Agenda

- Basic Concepts of Recommender Systems
- Collaborative Filtering
  - Neighborhood-based Collaborative Filtering
  - Model-based Collaborative Filtering
- Recommendation with Association Rules



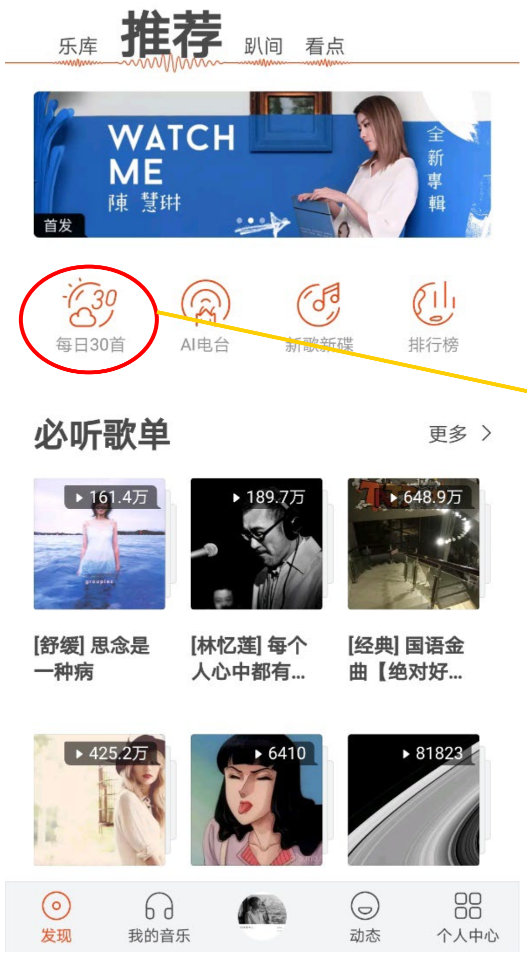
9.1 Basic Concepts of Recommender Systems

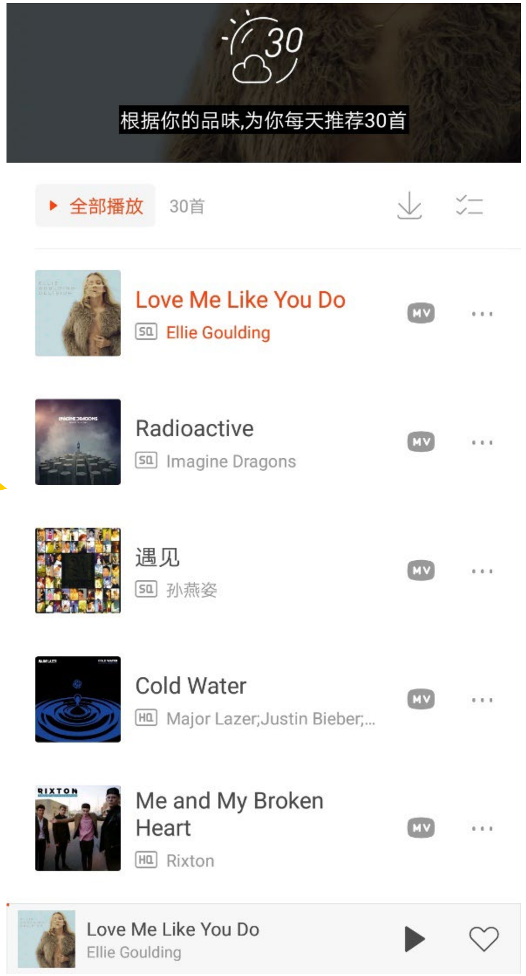
Example of Recommender Systems

- **根据浏览记录推荐商品**

Example of Recommender Systems

- **根据听歌历史推荐歌单**





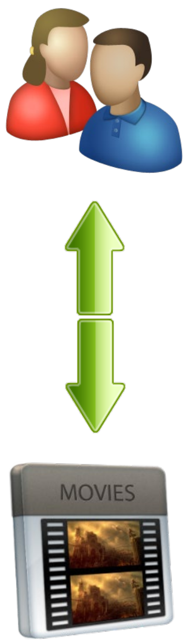
Example of Recommender Systems

- **根据浏览历史推荐新闻**

A recommender system is a subclass of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item. ——*Wiki*

Definition

- **Recommendation systems (RS) help to match users with items**
  - Ease information overload
  - Sales assistance (guidance, advisory, persuasion,…)



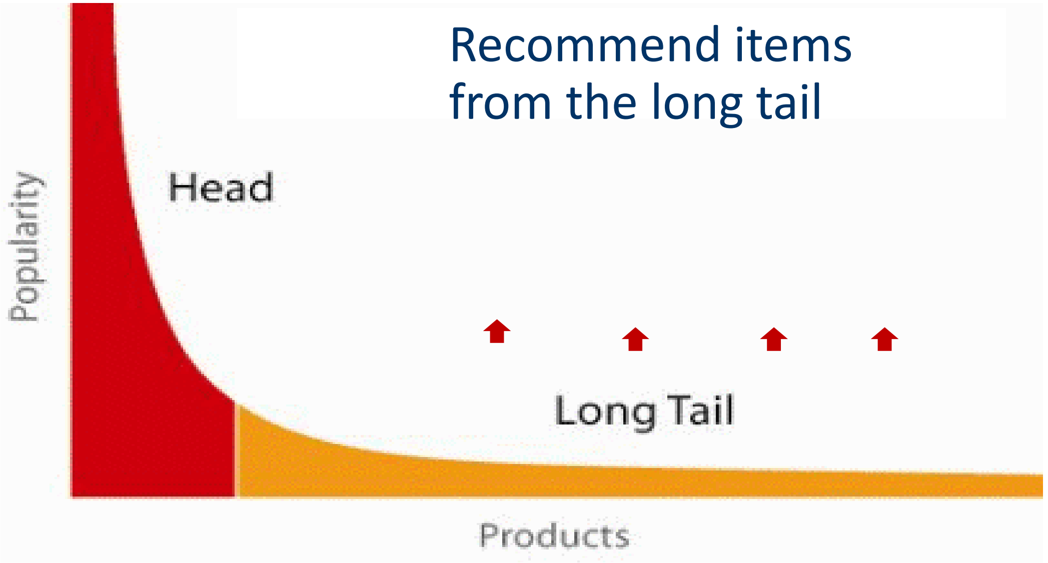
> *RS are software agents that elicit the interests and preferences of individual consumers […] and make recommendations accordingly. They have the potential to support and improve the quality of the decisions consumers make while searching for and selecting products online.*
>
> *»(Xiao & Benbasat 2007[1])*

- **Different system designs / paradigms**
  - Based on availability of exploitable data
  - Implicit and explicit user feedback
  - Domain characteristics

Purpose and success criteria

- **Different perspectives/aspects**
  - Depends on domain and purpose
  - No holistic evaluation scenario exists
- **Retrieval perspective**
  - Reduce search costs
  - Provide "correct" proposals
  - Users know in advance what they want
- **Recommendation perspective**
  - Serendipity – identify items from the Long Tail
  - Users did not know about existence

When does a RS do its job well?



- “Recommend widely unknown items that users might actually like!”
- 20% of items accumulate 74% of all positive ratings
- Items rated > 3 in MovieLens 100K dataset

Paradigms of recommender systems

Recommender systems reduce information overload by estimating relevance

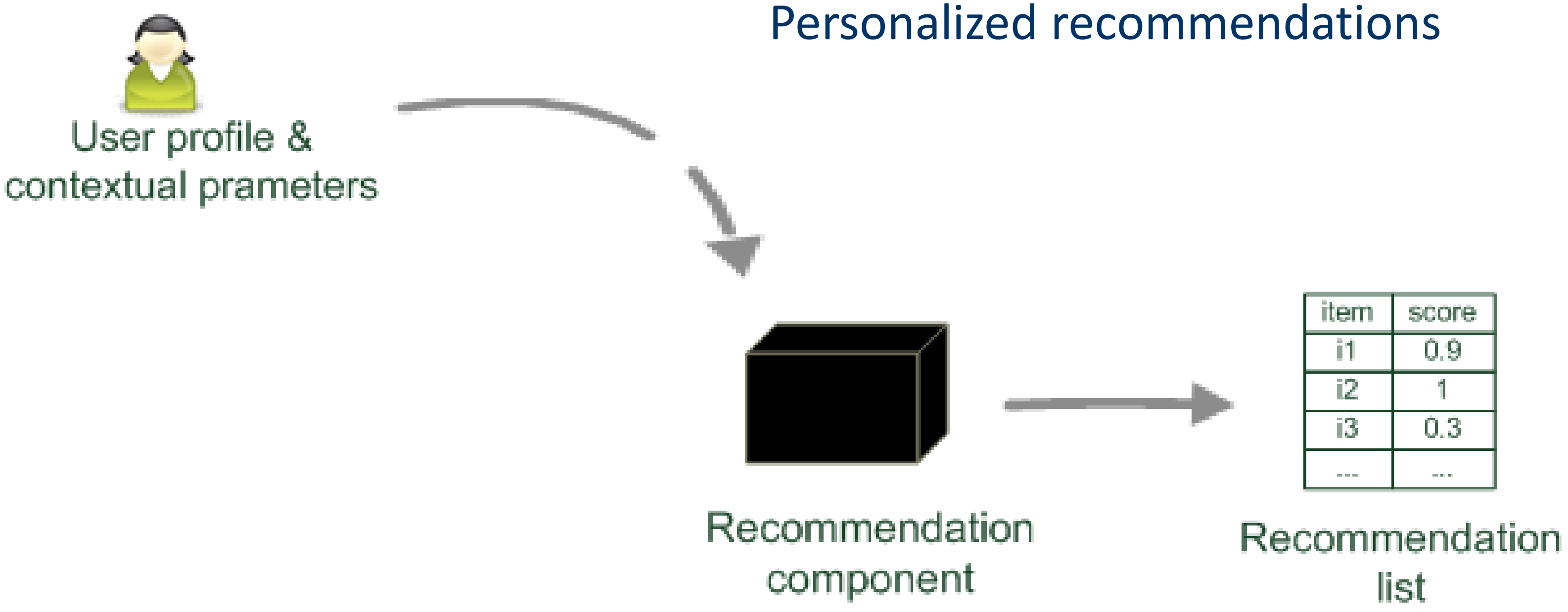
Recommendation component

item	score
i1	0.9
i2	1
i3	0.3
...	...

Recommendation list

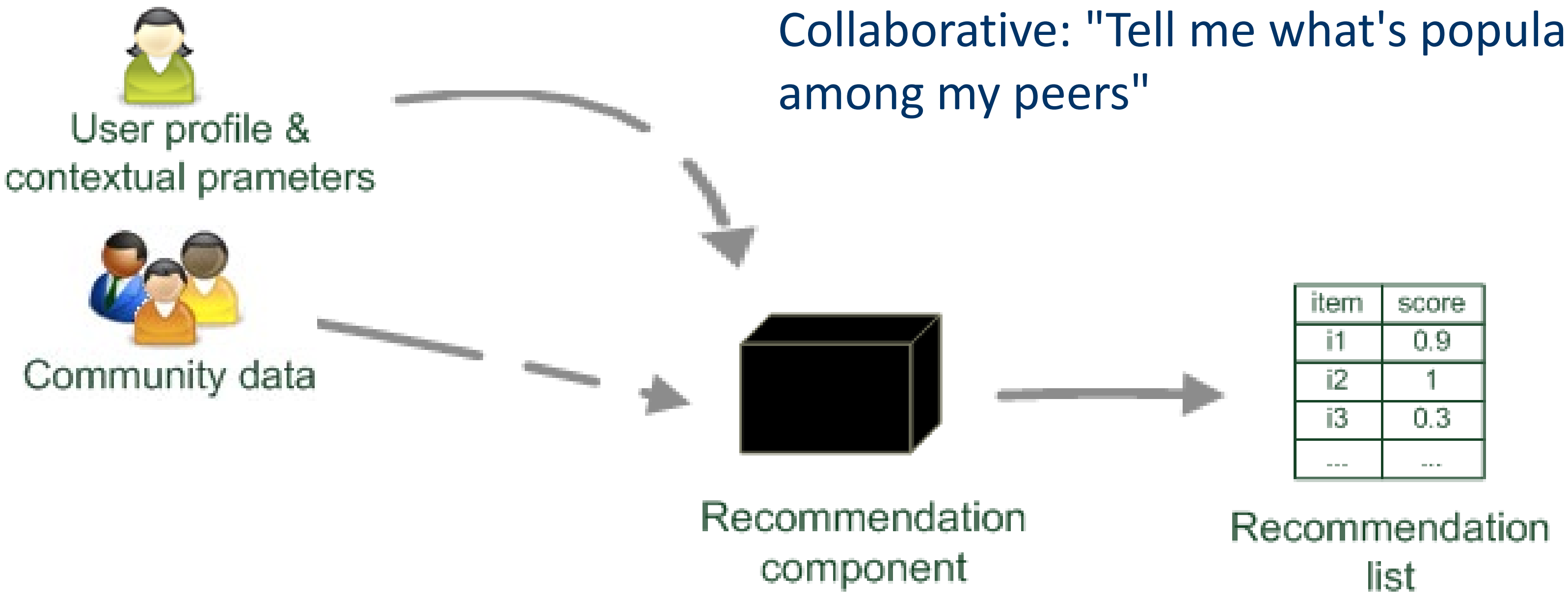
Paradigms of recommender systems

Personalized recommendations



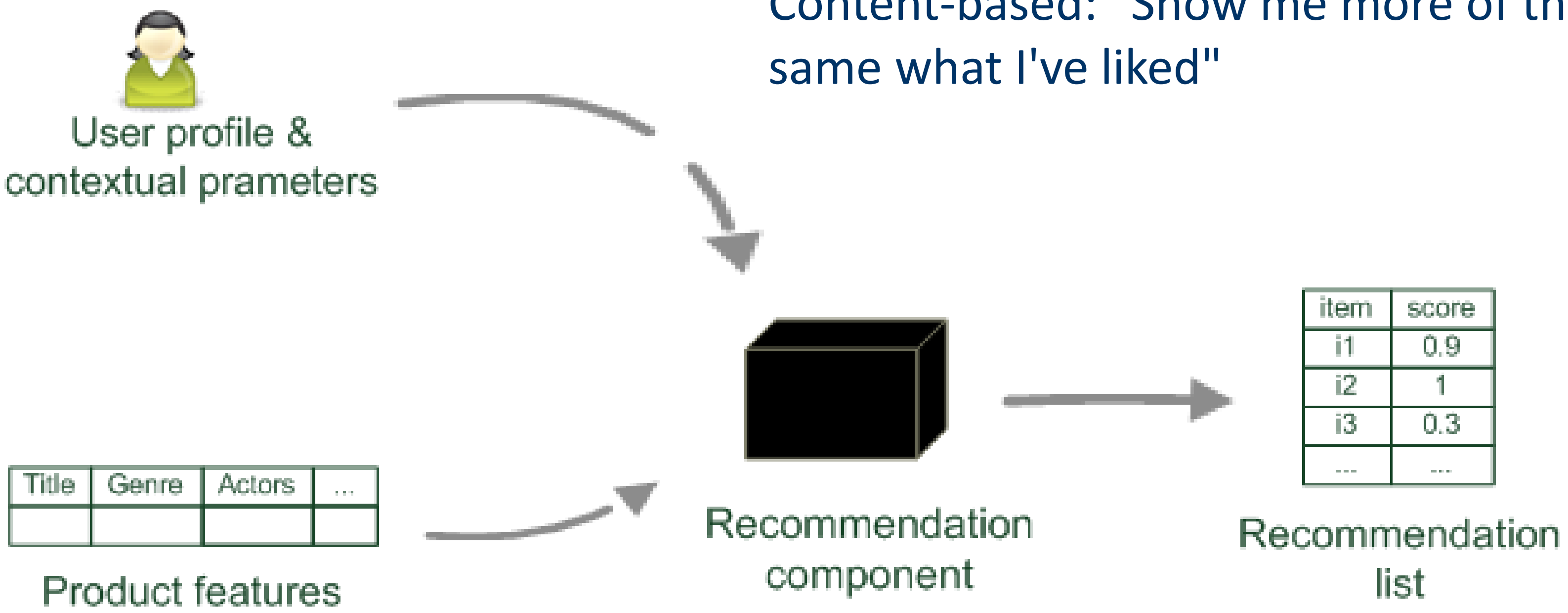
Paradigms of recommender systems

Collaborative: "Tell me what's popular among my peers"



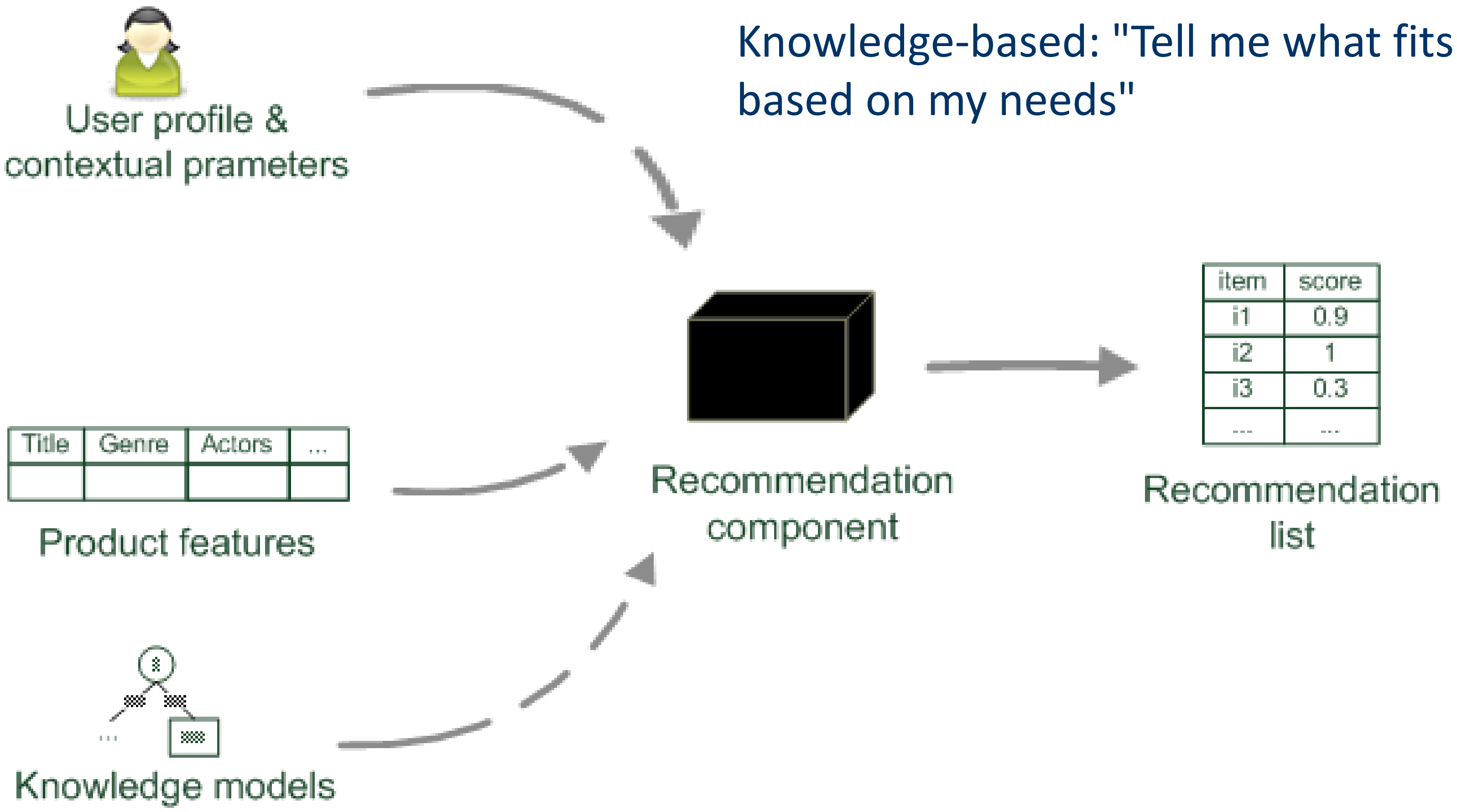
Paradigms of recommender systems

Content-based: "Show me more of the same what I've liked"



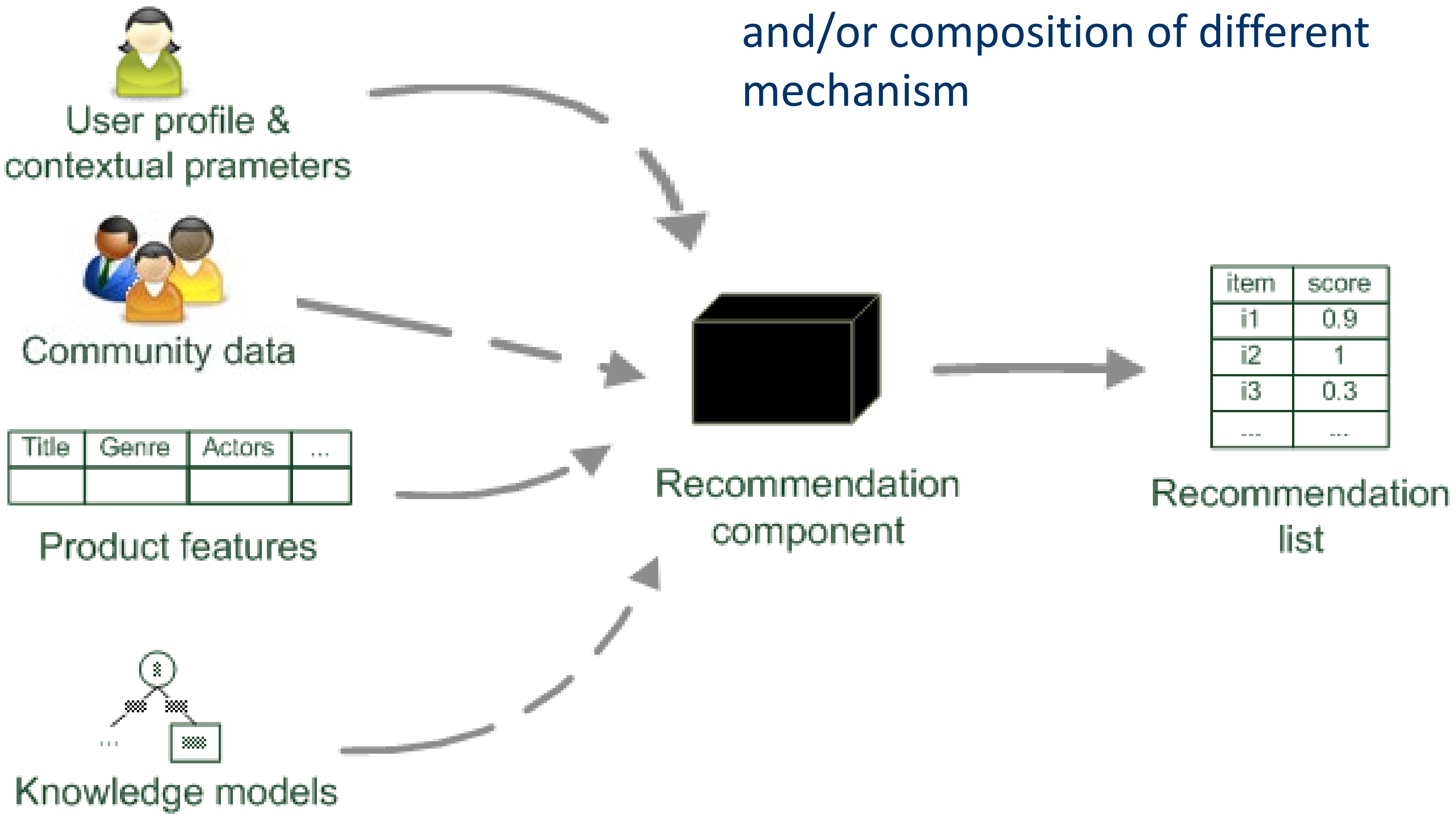
Paradigms of recommender systems

Knowledge-based: "Tell me what fits based on my needs"



Paradigms of recommender systems

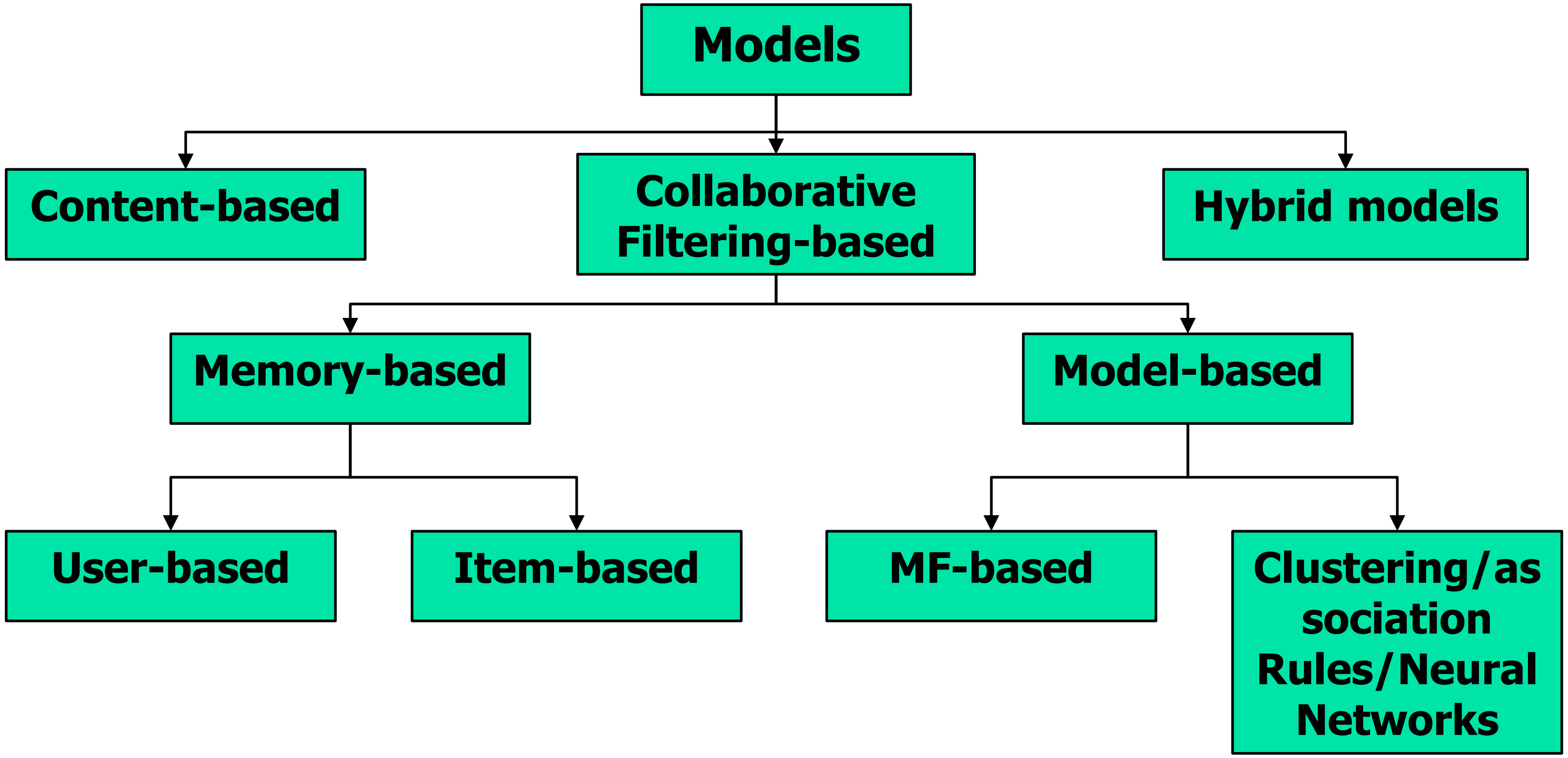
Hybrid: combinations of various inputs and/or composition of different mechanism



Development of recommender systems

2000	KNN Collaborative Filtering
2007-2015	Matrix Factorization, for example, GMF, SVD++ and FM.
2015-2017	Deep Learning Methods, for example, FNN, AutoEncoder, CDL, DeepFM.
2017-Now	Reinforcement Learning Methods

Recommendation Models



9.2 Neighborhood-based Collaborative Filtering

Recap: Collaborative Filtering(CF)

- Given database of user preferences, predict preference of new user
- Example: predict what new movies you will like based on
  - your past preferences
  - others with similar past preferences
  - their preferences for the new movies
- Example: predict what books/CDs a person may want to buy
  - (and suggest it, or give discounts to tempt customer)

	Book1	Book2	Book3	Book4	Book5	Book6	…...
User1							
User2							
User3							
User4							
User5							
User6	?	?		?	**?**	?	?

Neighborhood-based Collaborative Filtering

- **The most prominent approach to generate recommendations**
  - Used by large, commercial e-commerce sites
  - Well-understood, various algorithms and variations exist
  - Applicable in many domains (book, movies, DVDs, ..)
- **Approach**
  - Use the "wisdom of the crowd" to recommend items
- **Basic assumption and idea**
  - Users give ratings to catalog items (implicitly or explicitly)
  - Customers who had similar tastes in the past, will have similar tastes in the future

Neighborhood-based Collaborative Filtering

- **Also referred to as memory-based algorithms**
- **Based on the fact that similar users display similar patterns of rating behavior and similar items receive similar ratings**
  - User-based collaborative filtering
  - Item-based collaborative filtering

Rating matrix

- An incomplete $m \times n$ matrix $R = [r_{uj}]$ containing m users and n items
- Only a small subset of the ratings matrix is specified or observed

User	Item	Rating
1	1	5
1	4	4
…	…	…
u	**j**	**r**
…	…	…
…	…	…

5			4	
	1	2	3	3
4		4		
	3			
			2	1

- u, j: index for u_{th} user and j_{th} item
- r_{uj} : u_{th} user gives a rating r_{uj} to j_{th} item

User-based Collaborative Filtering

- **The basic technique**
  - Given an "active user" (Alice) and an item i not yet seen by Alice
    - find a set of users (peers/nearest neighbors) who liked the same items as Alice in the past and who have rated item i
    - use, e.g. the average of their ratings to predict if Alice will like item i
    - do this for all items Alice has not seen and recommend the best-rated
- **Basic assumption and idea**
  - If users had similar tastes in the past they will have similar tastes in the future
  - User preferences remain stable and consistent over time

User-based Collaborative Filtering

- **Example**
  - A database of ratings of the current user, Alice, and some other users is given:

					
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

  - Determine whether Alice will like or dislike *Cartoon5*, which Alice has not yet rated or seen

User-based Collaborative Filtering

- **Some questions:**
  - How do we measure similarity?
  - How many neighbors should we consider?
  - How do we generate a prediction from the neighbors' ratings?

					
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

Measuring user similarity

- **A popular similarity measure in user-based CF: Pearson correlation**

  u, v : users

  r_{uk} : rating of user u for item k

  I_u : denote the set of item indices for which ratings have been specified by user u

  μ_u : the mean rating for each user u using her specified ratings

  – The first step is to compute the mean rating μ_u

$$\mu_u = \frac{\sum_{k \in I_u} r_{uk}}{|I_u|} \quad \forall u \in \{1 \dots m\}$$

  – Then, the Pearson correlation coefficient between the u and v is defined

$$\mathrm{Sim}(u,v) = \mathrm{Pearson}(u,v) = \frac{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u) \cdot (r_{vk} - \mu_v)}{\sqrt{\sum_{k \in I_u \cap I_v} (r_{uk} - \mu_u)^2} \cdot \sqrt{\sum_{k \in I_u \cap I_v} (r_{vk} - \mu_v)^2}}$$

Measuring user similarity

- **A popular similarity measure in user-based CF: Pearson correlation**

  u, v : users

  r_{uk} : rating of user u for item k

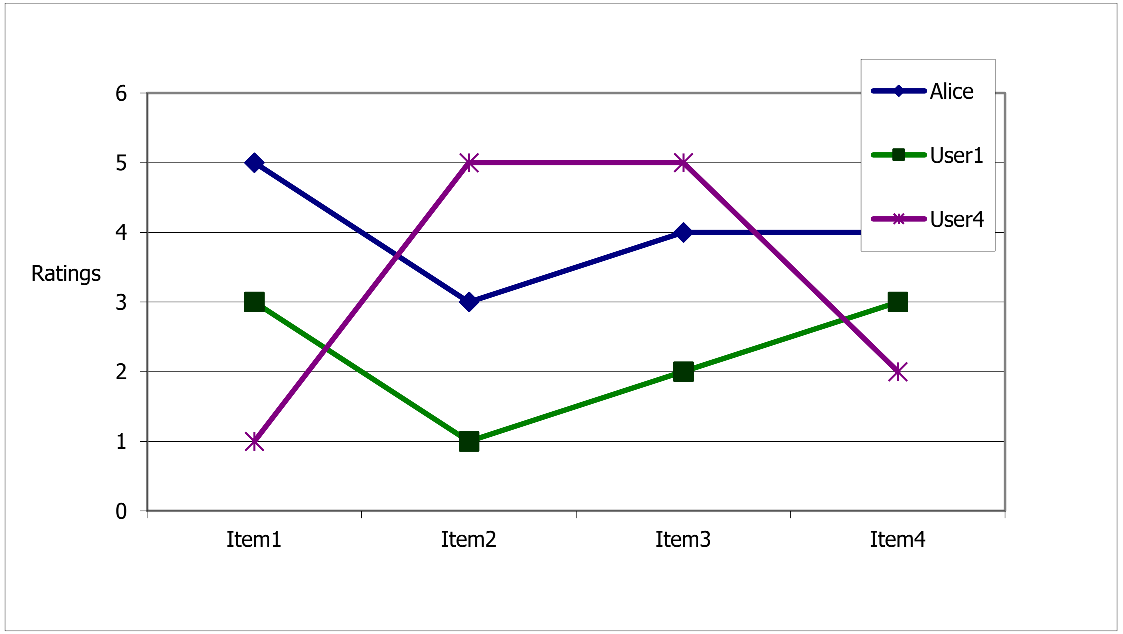
  I_u : denote the set of item indices for which ratings have been specified by user u

  μ_u : the mean rating for each user u using her specified ratings

Alice	5	3	4	4	?	
User1	3	1	2	3	3	sim = 0.85
User2	4	3	4	3	5	sim = 0.00
User3	3	3	1	5	4	sim = 0.70
User4	1	5	5	2	1	sim = -0.79

Pearson correlation

- **Takes differences in rating behavior into account**



- **Works well in usual domains, compared with alternative measures**
  - such as cosine similarity

Making predictions

- Use the set of k users with the highest Pearson coefficient to define the peer group of the target user
- Different users may provide ratings on different scales
- The rating need to be mean-centered in row-wise fashion, the mean-centered rating s_{uj} of a user u for item j is defined

$$s_{uj} = r_{uj} - \mu_u \quad \forall u \in \{1 \dots m\}$$

- Let $P_u(j)$ be the set of k closest users to target user u w.r.t item j
- Then, the overall neighborhood-based prediction function is as follows

$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u,v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u,v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u,v)|}$$

Making predictions

- Then, the overall neighborhood-based prediction function is as follows

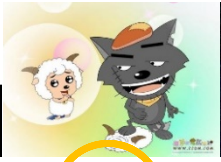
$$\hat{r}_{uj} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u,v) \cdot s_{vj}}{\sum_{v \in P_u(j)} |\text{Sim}(u,v)|} = \mu_u + \frac{\sum_{v \in P_u(j)} \text{Sim}(u,v) \cdot (r_{vj} - \mu_v)}{\sum_{v \in P_u(j)} |\text{Sim}(u,v)|}$$

Alice	5	3	4	4	?	
User1	3	1	2	3	3	sim = 0.85
User2	4	3	4	3	5	sim = 0.00
User3	3	3	1	5	4	sim = 0.70
User4	1	5	5	2	1	sim = -0.79

$$pred(Alice, Cartoon5) = 4 + \frac{0.85 * (3 - 2.4) + 0.7 * (4 - 3.2) + (-0.79) * (1 - 2.8)}{0.85 + 0.7 - 0.79} = 3.28$$

Item-based collaborative filtering

- **Basic idea:**
  - Use the similarity between items (and not users) to make predictions
- **Example:**
  - Look for items that are similar to *Cartoon5*
  - Take Alice's ratings for these items to predict the rating for *Cartoon5*

					
Alice	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

The cosine similarity measure

- **As in the case of user-based ratings, the average rating of each item in the ratings matrix is subtracted from each rating to create a mean-centered matrix**
- **The adjusted cosine similarity between the items i and j**
  - U_i: indices of the set of users who have specified ratings for item i

$$\text{AdjustedCosine}(i, j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} \cdot s_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} s_{ui}^2} \cdot \sqrt{\sum_{u \in U_i \cap U_j} s_{uj}^2}}$$

Making predictions

- The first step is to determine the top-k most similar items to item t based on the adjusted cosine similarity
- Let $Q_t(u)$ be the top-k matching items to item t, for which the user u has specified ratings
- Therefore, the predicted rating $\hat{r}_{ut}$ of user u for target item t is as follows

$$\hat{r}_{ut} = \frac{\sum_{j \in Q_t(u)} \text{AdjustedCosine}(j,t) \cdot r_{uj}}{\sum_{j \in Q_t(u)} |\text{AdjustedCosine}(j,t)|}$$

Comparing User-Based and Item-Based

- **Item-based**
  - More relevant recommendations, better accuracy
  - Might sometimes recommend obvious items, or items which are not novel
  - Item similarities are supposed to be more stable
- **User-based**
  - Diversity, may discover surprising and interesting items
  - Addition of a few ratings can change the similarity values drastically

Strengths and Weaknesses of Neighborhood-Based

- **Advantages**
  - Simplicity and intuitive, easy to implement and debug
  - Easy to justify why a specific item is recommended, and good interpretability
  - Stable with the addition of new items and users
- **Disadvantage**
  - Offline phase can sometimes be impractical in large-scale settings
  - Limited coverage because of sparsity. For example, if none of John’s nearest neighbors have rated *Doraemon*, it is not possible to provide a rating prediction of *Doraemon* for John.

Data sparsity problems

- **Cold start problem**
  - How to recommend new items? What to recommend to new users?
- **Straightforward approaches**
  - Ask/force users to rate a set of items
  - Use another method (e.g., content-based or simply non-personalized) in the initial phase

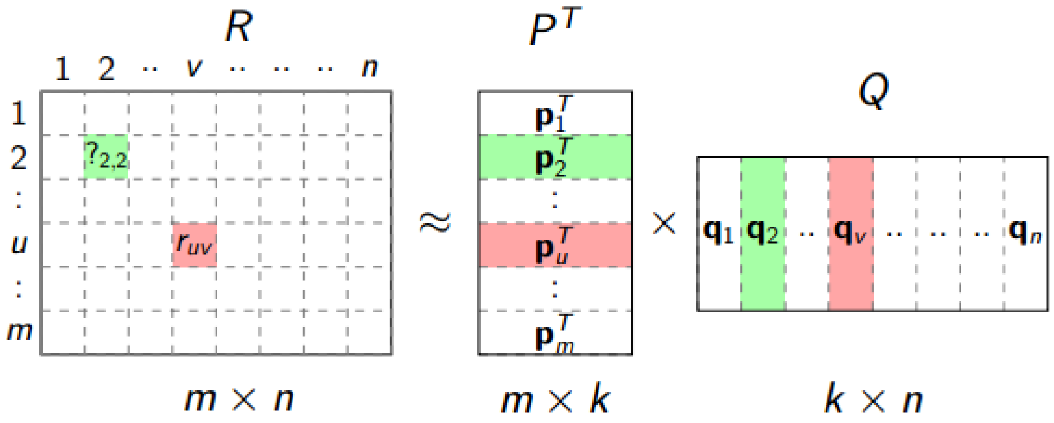
9.3 Model-based Collaborative Filtering

Introduction

- **Latent factor models**
  - try to explain the ratings by characterizing both items and users on latent factors.
  - for example, user-movie rating matrix.
  - for movies, latent factors might measure obvious dimensions such as comedy, drama, action; less well-defined dimensions such as “quirkiness”; or completely uninterpretable dimensions.
  - for users, each factor measures how much the user likes movies that score high on the corresponding movie factor.

Matrix Factorization

- Matrix Factorization [2] is an effective method for recommender systems (e.g., Netflix Prize and KDD Cup 2011) .



  - k: number of latent dimensions
  - $r_{u,v} = p_u^T q_v$
  - $?_{2,2} = p_2^T q_2$

Matrix Factorization

A
B
C
D
E

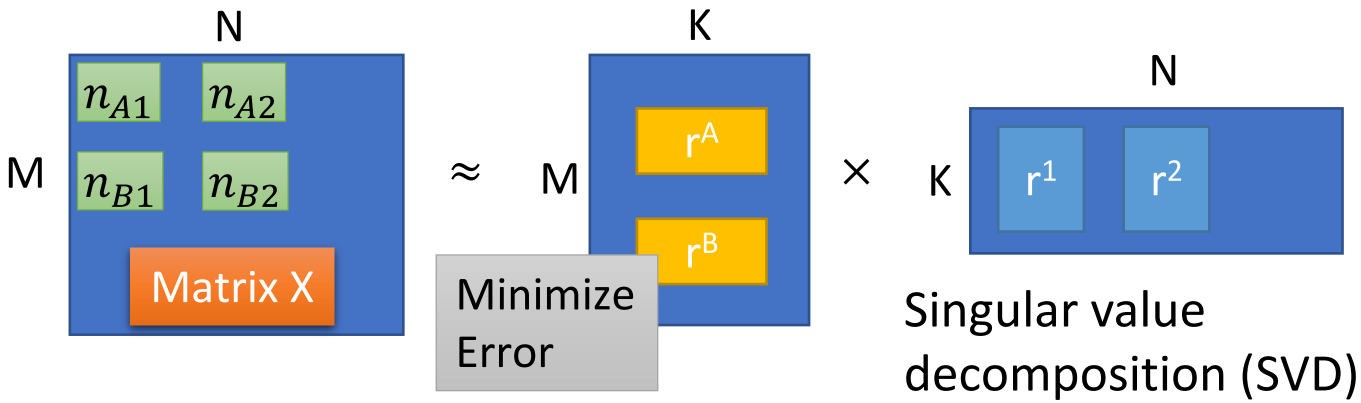
There are some common *factors* behind users and Cartoons.

Matrix Factorization

		r^1	r^2	r^3	r^4
r^A	A	5	3	0	1
r^B	B	4	3	0	1
r^C	C	1	Matrix X		5
r^D	D	1	0	4	4
r^E	E	0	1	5	4

No. of User = M　　No. of Cartoon = N　　No. of latent factor = K

$$r^A \cdot r^1 \approx 5$$
$$r^B \cdot r^1 \approx 4$$
$$r^C \cdot r^1 \approx 1$$
$$\vdots$$



Singular value decomposition (SVD)

Matrix Factorization



r^i \ r^j		r^1	r^2	r^3	r^4
r^A	A	5 n_{A1}	3	?	1
r^B	B	4	3	?	1
r^C	C	1	1	?	5
r^D	D	1	?	4	4
r^E	E	?	1	5	4

$r^A \cdot r^1 \approx 5$

$r^B \cdot r^1 \approx 4$

$r^C \cdot r^1 \approx 1$

⋮

Minimizing

$$L = \sum_{(i,j)} \left(r^i \cdot r^j - \boxed{n_{ij}}\right)^2$$

Only considering the defined value

Find r^i and r^j by gradient descent

Matrix Factorization

		r^1	r^2	r^3	r^4
					
r^A	A	5	3	5.49	1
r^B	B	4	3	4.84	1
r^C	C	1	1	5.19	5
r^D	D	1	0.70	4	4
r^E	E	1.59	1	5	4

Assume the dimensions of r are all 2 (there are two factors)

A	2.38	0.40
B	2.04	0.41
C	0.32	2.19
D	0.27	1.72
E	0.62	1.78

1	1.99	0.21
2	1.31	0.20
3	1.96	2.08
4	0.03	2.27

Learning algorithms

- A non-convex optimization problem:

$$\min_{P,Q} \sum_{(u,v)\in R} \left((r_{u,v} - \mathbf{p}_u^T \mathbf{q}_v)^2 + \lambda_P \|\mathbf{p}_u\|_F^2 + \lambda_Q \|\mathbf{q}_v\|_F^2 \right)$$

- λ_P and λ_Q are regularization parameters.
- SVD is a natural approach that approximates the original rating matrix R by the product of two rank-k matrices $\mathrm{R} = P^T \times Q$. However, as there are many missing elements in the rating matrix R, standard SVD algorithms cannot find P and Q.
  - ✓ SVD:

$$M_{m\times n} = U_{m\times k}\Sigma_{k\times k}V_{k\times n}^T$$

Learning algorithms

$$\min_{P,Q} \sum_{(u,v)\in R} \left((r_{u,v} - \mathbf{p}_u^T \mathbf{q}_v)^2 + \lambda_P \|\mathbf{p}_u\|_F^2 + \lambda_Q \|\mathbf{q}_v\|_F^2 \right)$$

- Stochastic gradient descent
- For each given training case, the system predicts r_{ui} and computes the associated prediction error:

$$e_{ui} \stackrel{def}{=} r_{ui} - q_i^T p_u.$$

- Update the parameters by a magnitude proportional to γ in the opposite direction of the gradient:

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$
$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

Learning algorithms: Alternating least squares

$$\min_{P,Q} \sum_{(u,v)\in R} \left((r_{u,v} - \mathbf{p}_u^T \mathbf{q}_v)^2 + \lambda_P \|\mathbf{p}_u\|_F^2 + \lambda_Q \|\mathbf{q}_v\|_F^2 \right)$$

- Step 1: Initialize matrix M by assigning the average rating for that movie as the first row, and small random numbers for the remaining entries
- Step 2: Fix M, solve U by minimizing the objective function (the sum of squared errors)
- Step 3: Fix U, solve M by minimizing the objective function similarly
- Step 4: Repeat Steps 2 and 3 until convergence

Matrix Factorization

- **Challenges**
  - Matrix Factorization model is poor interpretability. The latent features are hard to be explained, it can not explain the result of recommendation very well
  - Implicit ratings are not considered

Extended MF (Adding Biases)

- **Biases**
  - Much of the variation in ratings is due to effects associated with either users or items, independently of their interactions
    - i.e., some users tend to give higher ratings than others
    - i.e., some items tend to receive higher ratings than others
  - A prediction for an unknown rating r_{ui} is denoted by b_{ui}

$$b_{ui} = \mu + b_i + b_u$$

    - μ : the overall average rating over all items
    - b_u and b_i : the observed deviations of user u and item i

Extended MF (Adding Biases)

- **Suppose that the average rating over all movies, μ, is 3.9 stars**

- Joe tends to rate 0.2 stars lower than the average

- Cartoon1 tends to be rated 0.5 stars above the average

- Cartoon1's predicted rating by Joe:

$$b_{ui} = \mu + b_i + b_u = 3.9 - 0.2 + 0.5 = 4.2$$

Extended MF (Adding Biases)

- **Adding biases**
  - A rating is created by adding biases

$$\hat{r}_{ui} = b_{ui} + p_u^T q_i$$

- **Objective Function**
  - In order to learn parameters (b_i, b_u, q_i and p_u) we minimize the regularized squared error

$$min_{b,q,p} \sum_{(u,i)\in\mathcal{K}} (r_{ui} - (\mu + b_i + b_u + q_i^T p_u))^2 + \lambda(b_i^2 + b_u^2 + \|q_i\|^2 + \|p_u\|^2)$$

  - Minimization is typically performed by either stochastic gradient descent or alternating least squares

Extended MF (Incorporating Implicit Feedback)

- Explicit feedback: user rating
- Implicit feedback: user behaviors
  - indirectly reflect opinion through observing user behavior
  - e.g. purchase history, browsing history, search patterns, or even mouse movements
- Prediction accuracy can be improved by considering both explicit feedback and implicit feedback
- Most famous model: SVD++

Implicit feedback matrix

- Derive implicit feedback matrix from explicit rating(optional)

$$\underbrace{\begin{pmatrix} 1 & -1 & 1 & ? & 1 & 2 \\ ? & ? & -2 & ? & -1 & ? \\ 0 & ? & ? & ? & ? & ? \\ -1 & 2 & -2 & ? & ? & ? \end{pmatrix}}_{R} \Rightarrow \underbrace{\begin{pmatrix} 1/\sqrt{5} & 1/\sqrt{5} & 1/\sqrt{5} & 0 & 1/\sqrt{5} & 1/\sqrt{5} \\ 0 & 0 & 1/\sqrt{2} & 0 & 1/\sqrt{2} & 0 \\ 1/\sqrt{1} & 0 & 0 & 0 & 0 & 0 \\ 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} & 0 & 0 & 0 \end{pmatrix}}_{F}$$

- Construct implicit feedback matrix from user behaviors

SVD++

- **SVD++:** Factorization Meets the Neighborhood: a Multifaceted Collaborative Filtering Model.
  - Neighborhood (Item-based CF):
    - ✓ Estimate unknown ratings by using known ratings made by user for similar movies
    - ✓ Good at capturing localized information
    - ✓ Intuitive and simple to implement
  - Latent Factor (MF):
    - ✓ Estimate unknown ratings by uncover latent features that explain known ratings
    - ✓ Efficient at capturing global information

SVD++

- Integrate
  - ✓ capture both global and localized information
  - ✓ consideration of implicit feedback
- SVD++ is proposed to incorporate implicit feedback and capture all information:

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

- a user u is modeled as $p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j$, N(u) contains all items for which u provided an implicit preference.

SVD++

- **Remember:** $N(u)$ denotes the set of items for which user u expressed an implicit preference
- y_j **is a vector, denotes the implicit preference(factor) mined from the behavior that users** u **browse item j**
  - Similar to p_u, which denotes explicit preference with regard to item-factor
- So, a user can be characterized by normalizing the sum of factor vectors according to implicit information:

$$|N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$$

SVD++

- **SVD++ is proposed to incorporate implicit feedback:**

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

- **a user u is modeled as $p_u + |N(u)|^{-\frac{1}{2}} \sum_{j \in N(u)} y_j$, $N(u)$ contains all items for which u provided an implicit preference**

SVD++

$$\hat{r}_{ui} = b_{ui} + q_i^T \left(p_u + |\mathrm{N}(u)|^{-\frac{1}{2}} \sum_{j \in \mathrm{N}(u)} y_j \right)$$

- ***Another perspective of SVD++:***
- $F(m \times n)$ - Implicit Matrix
- $Y(n \times k)$ - the item-factor matrix of F
- U- the user-factor of explicit rating matrix R
- V – the item-factor of explicit rating matrix R
- *For explicit rating:* $R \approx UV^T$
- Consider SVD++ as the approximation(*explicit*+*implicit*):

$$R \approx (U+FY)V^T$$

9.4 Recommendation with Association Rules (Optional)

什么是关联规则（Association Rule）？

牛奶 → 面包

尿布 → 啤酒

{牛奶, 面包} → {啤酒，尿布}

$X \rightarrow Y$

TID（事务ID）	商品列表
T001	牛奶、啤酒、尿布
T002	鸡蛋、牛奶、面包、啤酒、尿布
T003	鸡蛋、牛奶、面包
T004	面包、啤酒
T005	牛奶、面包、啤酒、尿布

age("25-35") ∧ buy("华为手机") ⟶ buy("格力空调")

关联规则有什么用?

- 零售行业：优化货架
- 电商行业：商品推荐
- 库存优化
- ……

关联规则的形式化定义

- 基本概念
  - 事务(transaction)
  - 项（item）
  - **项集(item set)**
    - k-项集
  - **关联规则：**
    - 形如 $X \rightarrow Y$ 的**蕴涵表达式**
      - X 和 Y 为非空集合
      - X 和 Y 是**不相交**的两个项集

TID（事务ID）	商品列表
T001	牛奶、啤酒、尿布
T002	鸡蛋、牛奶、面包、啤酒、尿布
T003	鸡蛋、牛奶、面包
T004	面包、啤酒
T005	牛奶、面包、啤酒、尿布

TID（事务ID）	商品列表
T001	尿布、牛奶、啤酒
T002	尿布、鸡蛋、牛奶、面包、啤酒
T003	鸡蛋、牛奶、面包
T004	面包、啤酒
T005	尿布、牛奶、面包、啤酒

→

TID（事务ID）	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}

$X \rightarrow Y$ **很多！**

如何衡量关联规则的“质量”?

$X \longrightarrow Y$

- **项集的支持度计数:**
  - 在数据库中包含项集W的事务个数
  - $\sigma(W) = |\{t_i | W \sqsubseteq t_i, t_i \in \mathrm{T}\}|$
- **支持度(support)**
  - 给定数据集中X和Y的共现频度
  - 令 $W = X \cup Y$
  - $s(X \rightarrow Y) = \dfrac{\sigma(W)}{|T|}$
- **置信度(confidence)**
  - 在包含X的事务子集中Y出现的频繁程度
  - $c(X \rightarrow Y) = \dfrac{\sigma(W)}{\sigma(X)}$

TID(事务ID)	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}

R_1: **I5 → I1**

$$s(R_1) = \frac{\sigma(\{I5, I1\})}{|T|} = \frac{3}{5} = 0.6$$

$$c(R_1) = \frac{\sigma(\{I5, I1\})}{\sigma(\{I5\})} = \frac{3}{4} = 0.75$$

强关联规则（strong association rule）

- 定义
  - 最小支持度阈值：*min-sup*，　最小置信度阈值：*min-conf*
  - 关联规则*R*是强关联规则，当且仅当：

$$(1).\ s(R) \geq \mathit{min{-}sup} \qquad (2).\ c(R) \geq \mathit{min\text{-}conf}$$

- **关联规则挖掘任务：找到所有强关联规则！**

关联规则挖掘任务

- 输入：
  - 事务数据库、
  - 阈值参数：*min-sup*，*min-conf*
- 输出：
  - 所有满足*min-sup*、*min-conf*的强关联规则
  - 包括每条强关联规则的支持度和置信度取值

TID（事务ID）	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}

最简单的关联规则挖掘算法——枚举法

- 假设： *min-sup* = 0.6， *min-conf* = 0.8
- 枚举法:
  - Step 1: 列出全部可能的关联规则
  - Step 2: 计算每条关联规则的支持度和置信度
  - Step 3: 筛选出满足min-sup和min-conf条件的规则

TID（事务ID）	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}

TID	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}

关联规则	支持度	置信度
{I1}→{I2}	0.2	0.33
{I2}→{I1}	0.2	0.5
{I1}→{I3}	0.6	1.0
{I3}→{I1}	0.6	0.75
……	…	…
{I1, I2}→{I3}	0.2	1.0
{I3}→{I1, I2}	0.2	0.25
……	…	…
{I1, I2, I3}→{I4}	0.2	1.0
{I4}→{I1, I2, I3}	0.2	0.25
……	…	…
{I1, I2, I3, I4}→{I5}	0.2	1.0
{I5}→{I1, I2, I3, I4}	0.2	0.25
……	…	…

min-sup=0.6

min-conf=0.8

关联规则	支持度	置信度
{I1}→{I3}	0.6	1.0
{I1}→{I5}	0.6	1.0
{I5}→{I3}	0.6	1.0
{I1}→{I3, I5}	0.6	1.0
{I3, I5}→{I1}	0.6	1.0
{I1, I5}→{I3}	0.6	1.0
{I1, I3}→{I5}	0.6	1.0

枚举法的特点分析

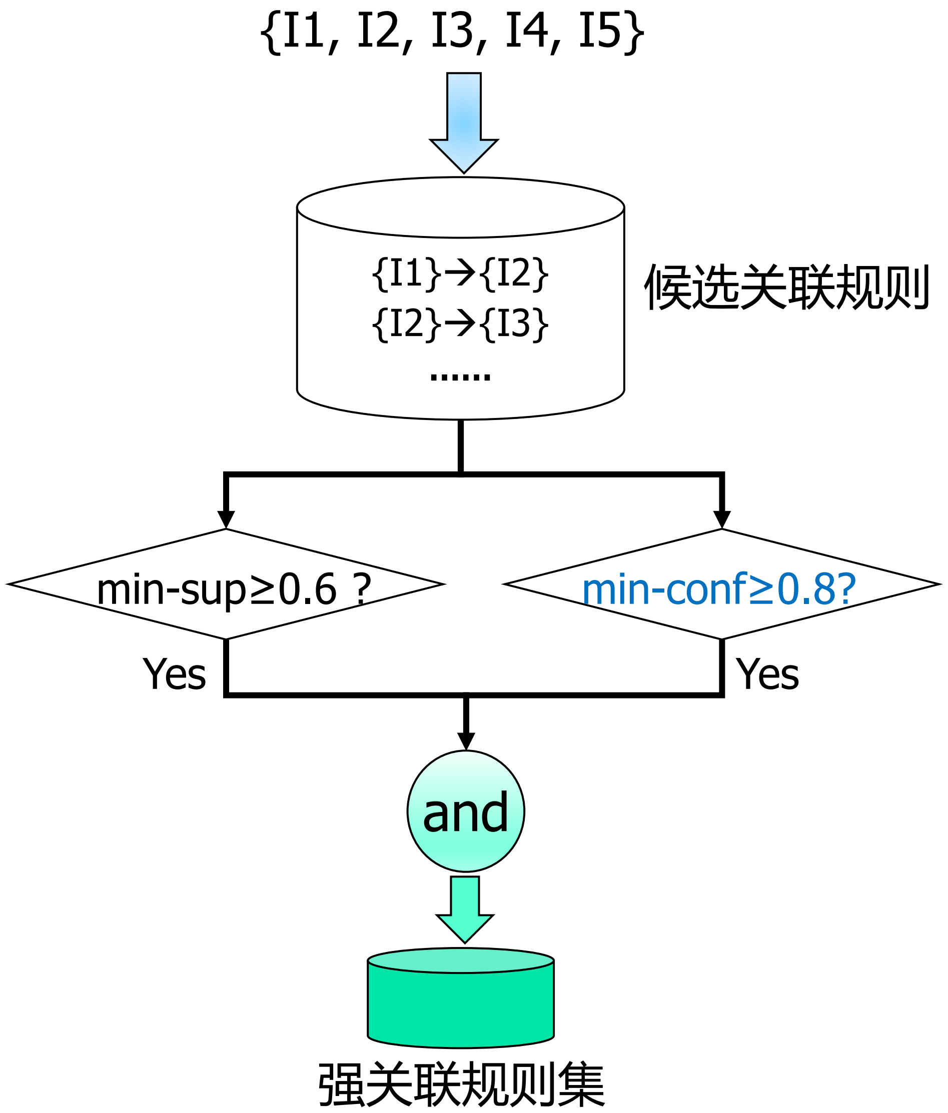
- 优点：实现简单，容易理解
- 缺点：待计算的关联规则数量随数据集增大呈指数增长
  - 包含n个项的数据集可提取的关联规则总数M为：$\mathrm{M}=3^{n}-2^{n+1}+1$

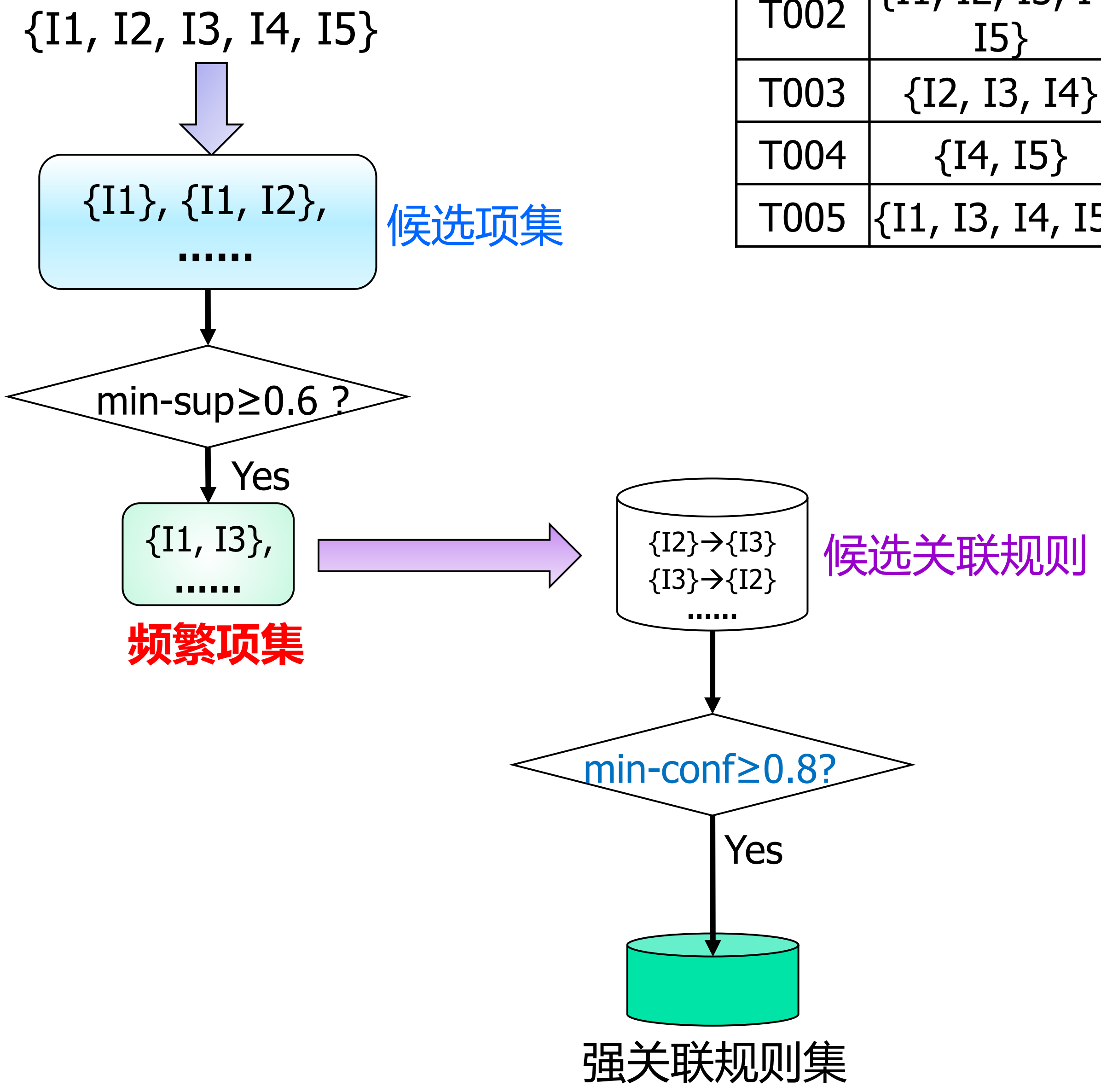
TID	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}

n=5, 可产生180条关联规则！

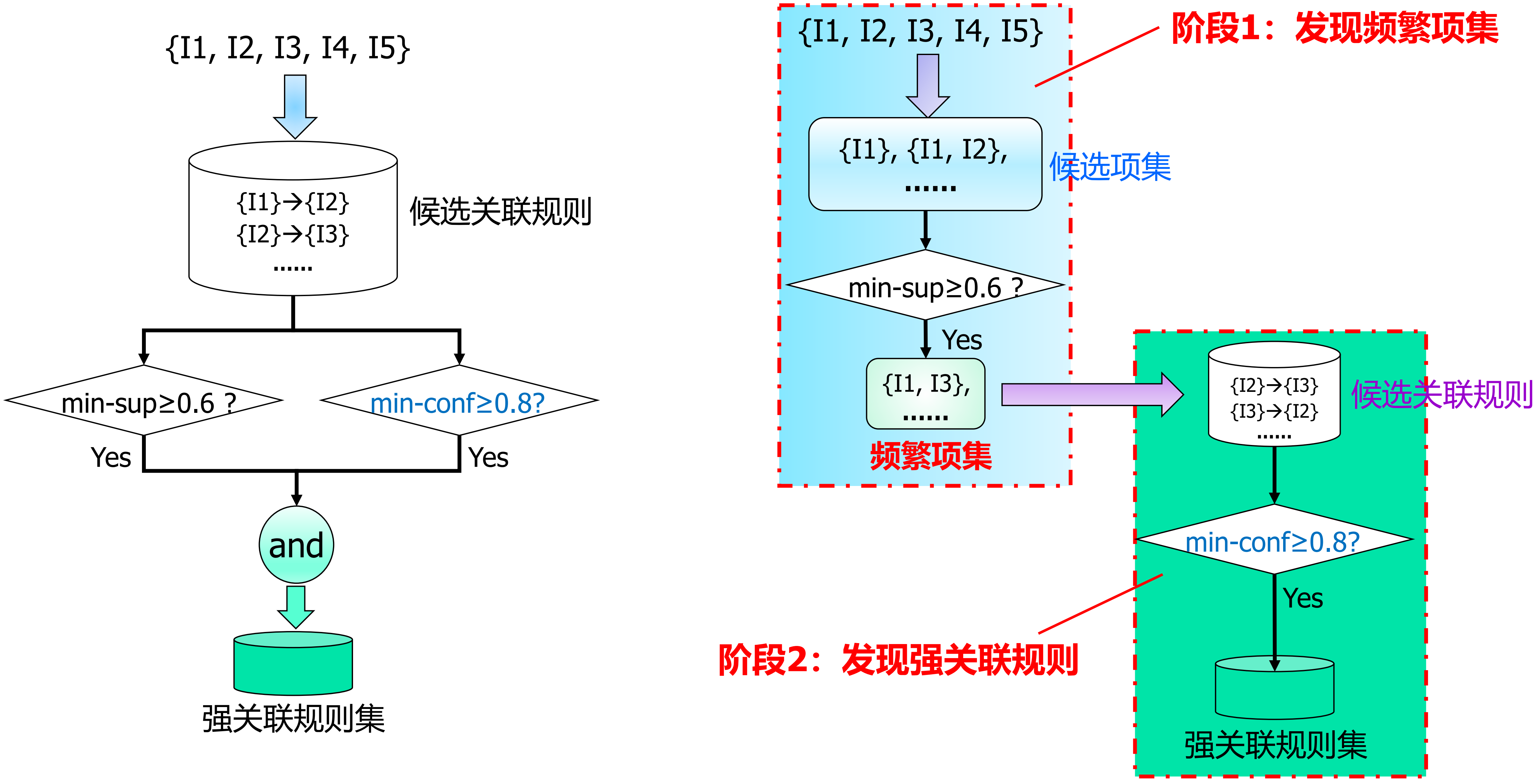
- 改进思路：从单步筛选变为“**两阶段**”筛选

TID	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}





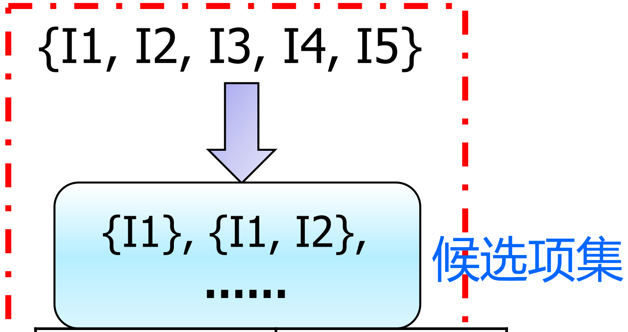
关联规则挖掘的两阶段算法框架



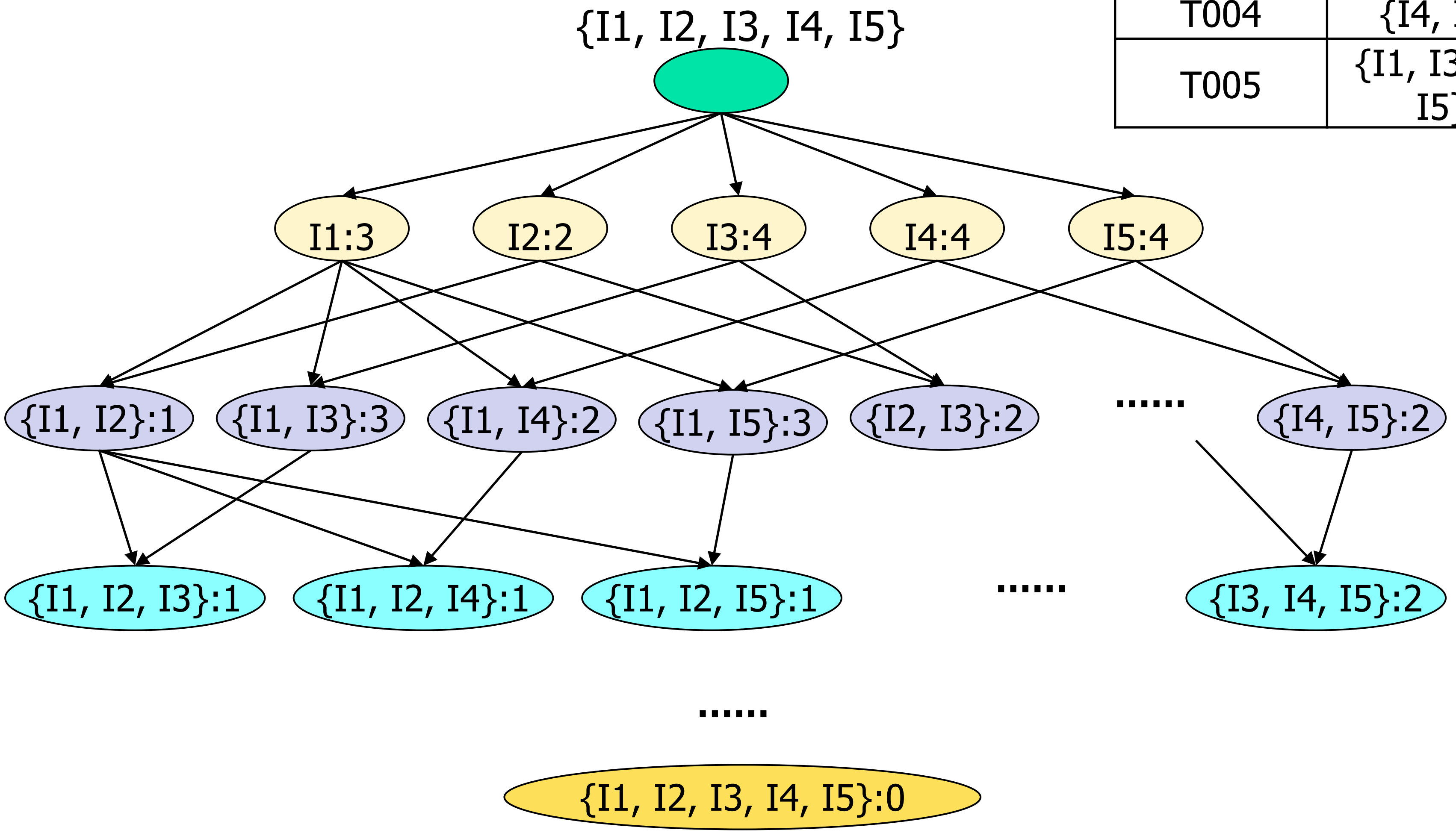
阶段1：发现频繁项集

min-sup = 0.6

TID	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}



频繁项集	支持度计数
{I1}	3
{I3}	4
{I4}	4
{I5}	4
{I1, I3}	3
{I1, I5}	3
{I3, I4}	3
{I3, I5}	3
{I4, I5}	3
{I1, I3, I5}	3



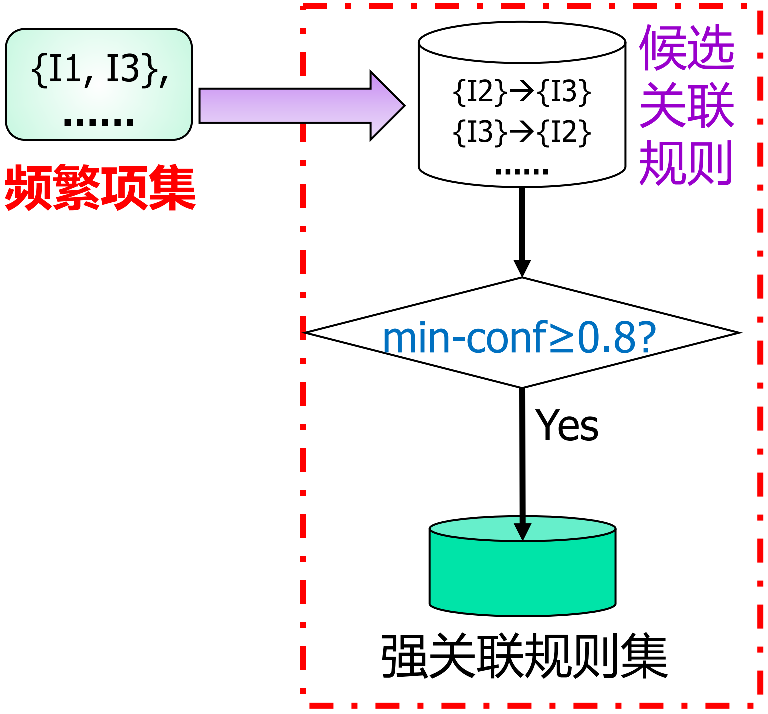
阶段2：发现强关联规则

min-conf = 0.8

TID	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}

频繁项集	支持度计数
{I1}	3
{I3}	4
{I4}	4
{I5}	4
{I1, I3}	3
{I1, I5}	3
{I3, I4}	3
{I3, I5}	3
{I4, I5}	3
{I1, I3, I5}	3

频繁项集	支持度计数
{I1, I3}	3
{I1, I5}	3
{I3, I4}	3
{I3, I5}	3
{I4, I5}	3
{I1, I3, I5}	3



两阶段算法与单步枚举法的计算量比较

TID	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}

频繁项集	支持度计数
{I1}	3
{I3}	4
{I4}	4
{I5}	4
{I1, I3}	3
{I1, I5}	3
{I3, I4}	3
{I3, I5}	3
{I4, I5}	3
{I1, I3, I5}	3

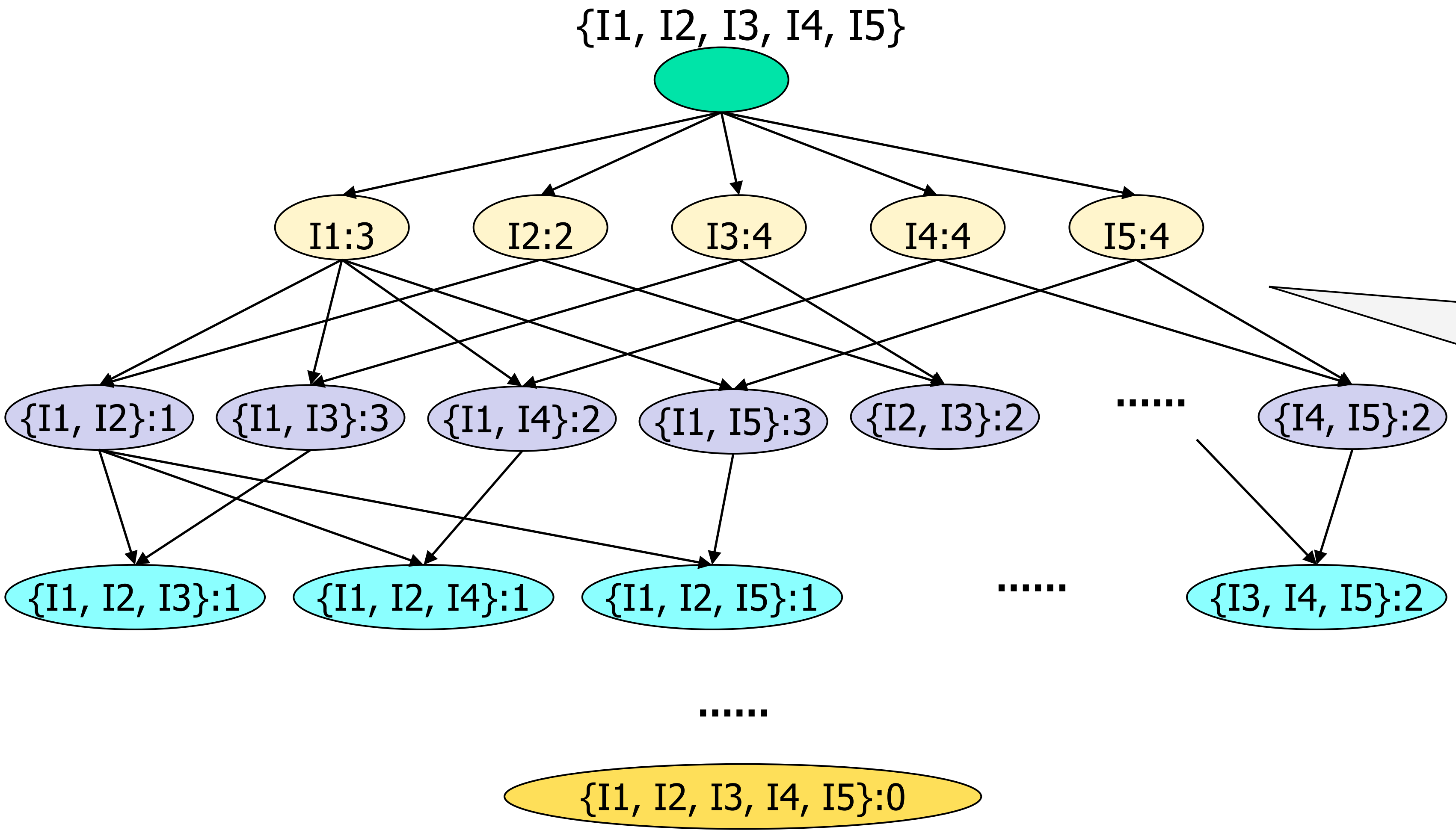
关联规则	置信度
{I1}→{I3}	1.0
{I1}→{I5}	1.0
{I5}→{I3}	1.0
{I1}→{I3, I5}	1.0
{I3, I5}→{I1}	1.0
{I1, I5}→{I3}	1.0
{I1, I3}→{I5}	1.0

- 阶段一：频繁项集发现
  - 计算31个项集的支持度
  - 筛选得到12个频繁项集
- 阶段二：关联规则生成
  - 计算12条候选关联规则的置信度
  - 筛选得到7条强关联规则
- 单步枚举法需要计算180个关联规则的支持度和置信度
- **计算量对比：360 → 43**

9.5 频繁项集发现：Apriori算法与FP-Growth

用枚举法发现频繁项集的问题

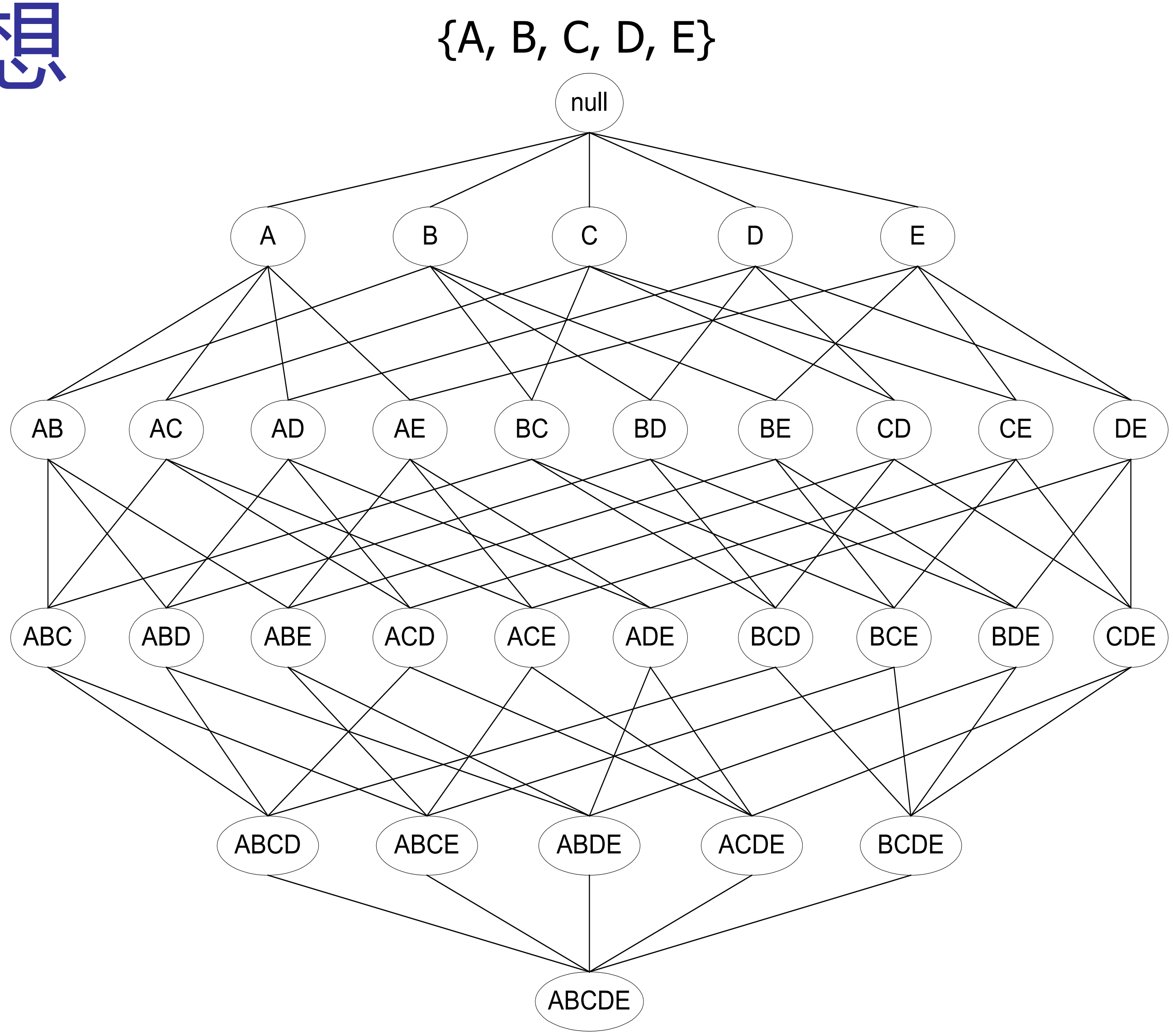
TID	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}



k个项
有2^k-1个可能的候选项集
计算开销大!

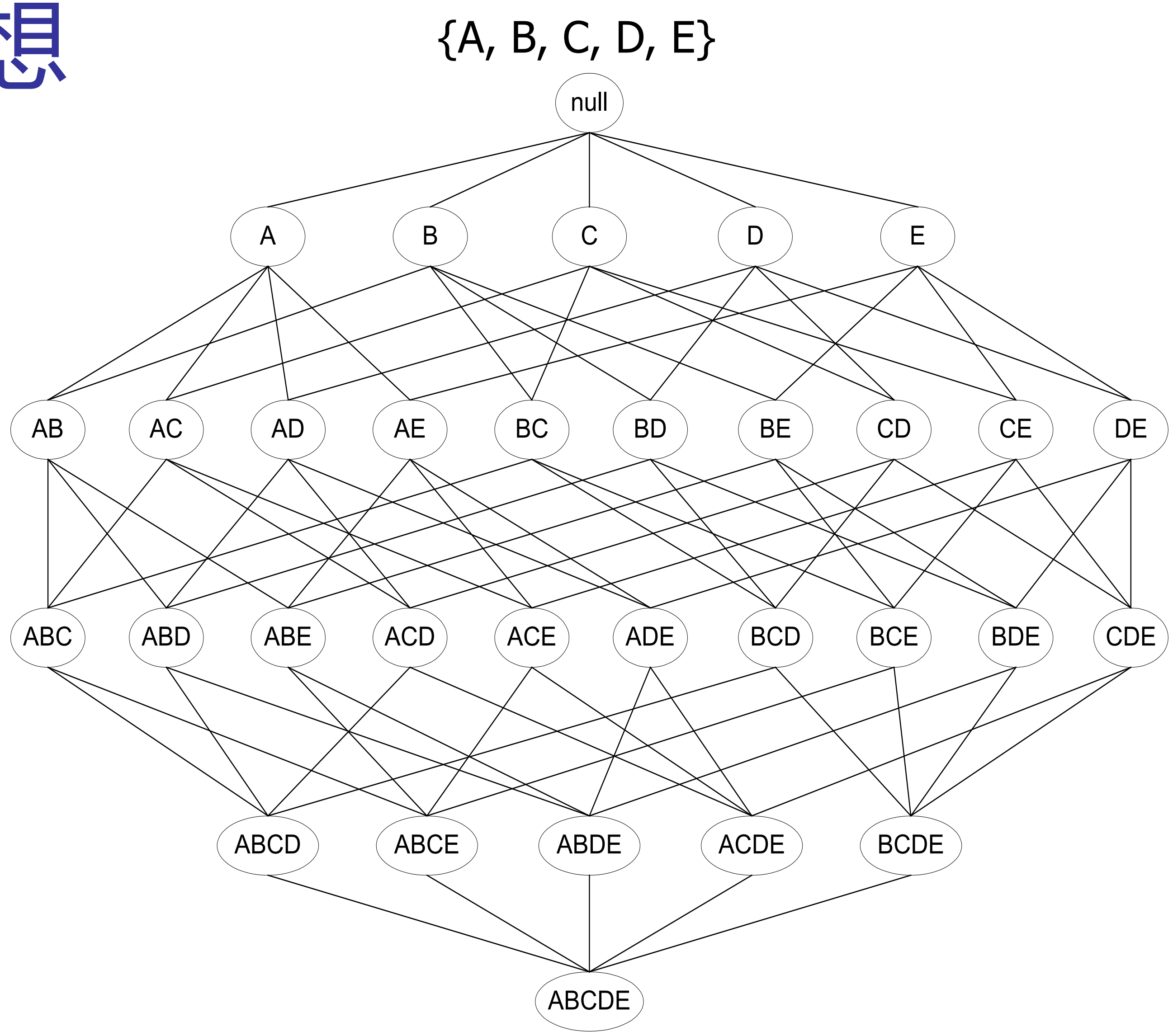
Aprior算法的基本思想

- **减少候选项集的数量！**
- **剪枝！**



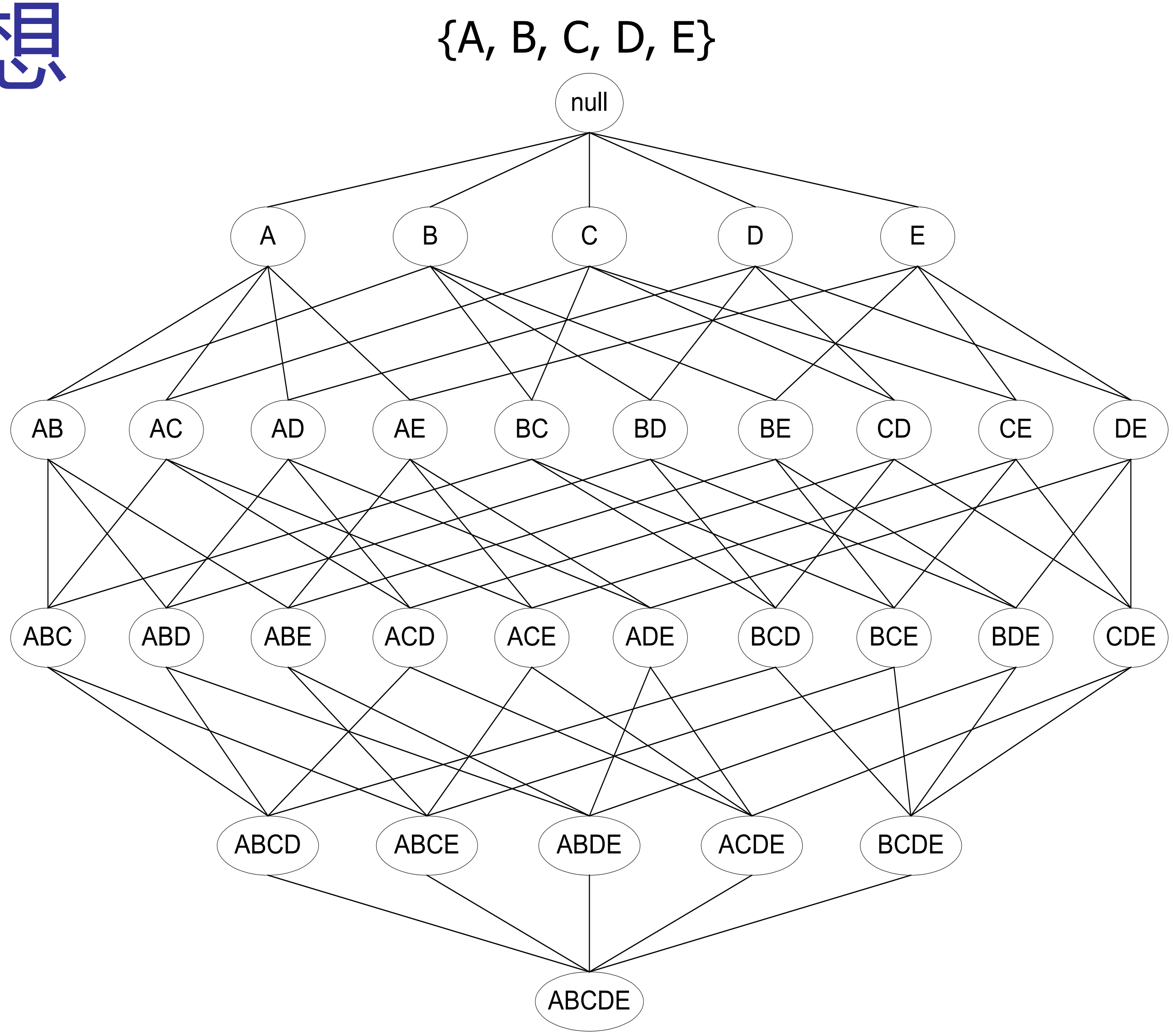
Aprior算法的基本思想

- **先验原理**
  - 频繁项集的子集一定也是频繁的！
- **反单调性**
  - 一个项集的支持度不超过它的子集的支持度
  - 如果对于项集Y的每个真子集X（即X⊏Y），有f(Y)≤f(X)，那么称度量f具有反单调性。



Aprior算法的基本思想

- 基本思想:
  - 逐层搜索迭代
  - 用上一轮迭代得到的k项集来探索下一轮迭代的（k+1）项集



Aprior算法的主要步骤

(1). 扫描数据库，得到所有频繁1项集

(2). 用k-频繁项集生成(k+1)-候选项集

(3). 扫描数据库计算每个(k+1)-候选项集的支持度，如果超过*min-sup*则为(k+1)-频繁项集

(4). 重复(2)、(3)直到无法生成新的候选项集

min-sup = 0.6

TID	商品ID列表
T001	{I1, I3, I5}
T002	{I1, I2, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I4, I5}
T005	{I1, I3, I4, I5}

TID	商品ID列表
T001	{I1, I2, I3, I4, I5}
T002	{I1, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I1, I3, I5}
T005	{I4, I5}

min-sup = 0.6

min-sup计数=3

1st scan

Itemset	sup
{I1}	
{I2}	
{I3}	
{I4}	
{I5}	

C1

Itemset	sup

L1

Itemset

C2

2nd scan

Itemset	sup

C2

Itemset	sup

L2

Itemset

C3

3rd scan

Itemset	sup

C3

Itemset	sup

L3

TID	商品ID列表
T001	{I1, I2, I3, I4, I5}
T002	{I1, I3, I4, I5}
T003	{I2, I3, I4}
T004	{I1, I3, I5}
T005	{I4, I5}

Database
min-sup=3

1^{st} scan →

Itemset	sup
{I1}	3
{I2}	2
{I3}	4
{I4}	4
{I5}	4

C1

Itemset	sup
{I1}	3
{I3}	4
{I4}	4
{I5}	4

L1

Itemset
{I1, I3}
{I1, I4}
{I1, I5}
{I3, I4}
{I3, I5}
{I4, I5}

C2

2^{nd} scan →

Itemset	sup
{I1, I3}	3
{I1, I4}	2
{I1, I5}	3
{I3, I4}	3
{I3, I5}	3
{I4, I5}	3

C2

Itemset	sup
{I1, I3}	3
{I1, I5}	3
{I3, I4}	3
{I3, I5}	3
{I4, I5}	3

L2

Itemset
~~{I1, I3, I4}~~
{I1, I3, I5}
{I3, I4, I5}

C3

3^{rd} scan ←

Itemset	sup
~~{I1, I3, I4}~~	~~2~~
{I1, I3, I5}	3
{I3, I4, I5}	2

C3

Itemset	sup
{I1, I3, I5}	3

L3

Aprior算法的特点分析

- 优点
  - 原理简单，易于实现
  - 适合于稀疏数据集中的频繁模式挖掘
- 缺点
  - 候选项集数量可能很大
    - 假设L_1有10^4项，则C_2将包含10^7项
    - 假设要挖掘L_{100}，需要产生$2^{100} \approx 10^{30}$个候选项
  - 重复扫描数据库
    - 每轮迭代对候选项集进行支持度计数时，都需要扫描一遍数据库，从而产生不可忽视的I/O开销

FP-tree：不需要生成候选项集的算法

- 裴健等人于2000年提出另一种算法——频繁模式生长（Frequent-Pattern Growth），也称为FP-tree算法
- 基本思想
  - 假设 “abc” 是频繁项集
  - 找到数据集中包含“abc”的记录：

    DB|abc —— 条件数据库
  - “d” 是DB|abc中的频繁项

    → “abcd” 也是频繁项集

FP-tree算法的基本框架

- 使用一种紧凑的数据结构——FP树（FP-tree）来组织数据，并从中发现频繁项集
- 主要步骤：
  - 构建FP-tree
  - 从FP-tree中发现（生成）频繁项集
- 特点：
  - 只需要扫描两次原始数据库
  - 在数据库较大、记录较长时，可比Apriori算法快多个数量级

FP-tree的构建

(1) 扫描数据库找到频繁1-项集

(2) 将频繁1-项集按降序排序

(3) 再次扫描数据库，构建FP-tree

最小支持度计数设为3

TID（事务ID）	商品ID列表
T001	{I5, I1, I2, I3, I6}
T002	{I4, I5, I1, I3}
T003	{I1, I4, I7}
T004	{I5, I1, I3, I6, I7}
T005	{I4, I3, I6}

项集	支持度计数
{I5}	3
{I1}	4
{I2}	1
{I3}	4
{I6}	3
{I4}	3
{I7}	2

→

项集	支持度计数
{I5}	3
{I1}	4
{I3}	4
{I4}	3
{I6}	3

→

项集	支持度计数
{I1}	4
{I3}	4
{I4}	3
{I5}	3
{I6}	3

(3) 再次扫描数据库，构建FP-tree

项集	支持度计数	节点链
{I1}	**4**	
{I3}	**4**	
{I4}	**3**	
{I5}	**3**	
{I6}	**3**	

项集	支持度计数
{I1}	4
{I3}	4
{I4}	3
{I5}	3
{I6}	3

原事务数据库

TID	商品ID列表
T001	{I5, I1, I2, I3, I6}
T002	{I4, I5, I1, I3}
T003	{I1, I4, I7}
T004	{I5, I1, I3, I6, I7}
T005	{I4, I3, I6}

剔除非频繁项、并排序!

TID	商品ID列表
T001	
T002	
T003	
T004	
T005	

(3) 再次扫描数据库，构建FP-tree

项集	支持度计数	节点链
{I1}	**4**	
{I3}	**4**	
{I4}	**3**	
{I5}	**3**	
{I6}	**3**	

原事务数据库

TID	商品ID列表
T001	{I5, I1, I2, I3, I6}
T002	{I4, I5, I1, I3}
T003	{I1, I4, I7}
T004	{I5, I1, I3, I6, I7}
T005	{I4, I3, I6}

剔除非频繁项、并排序！

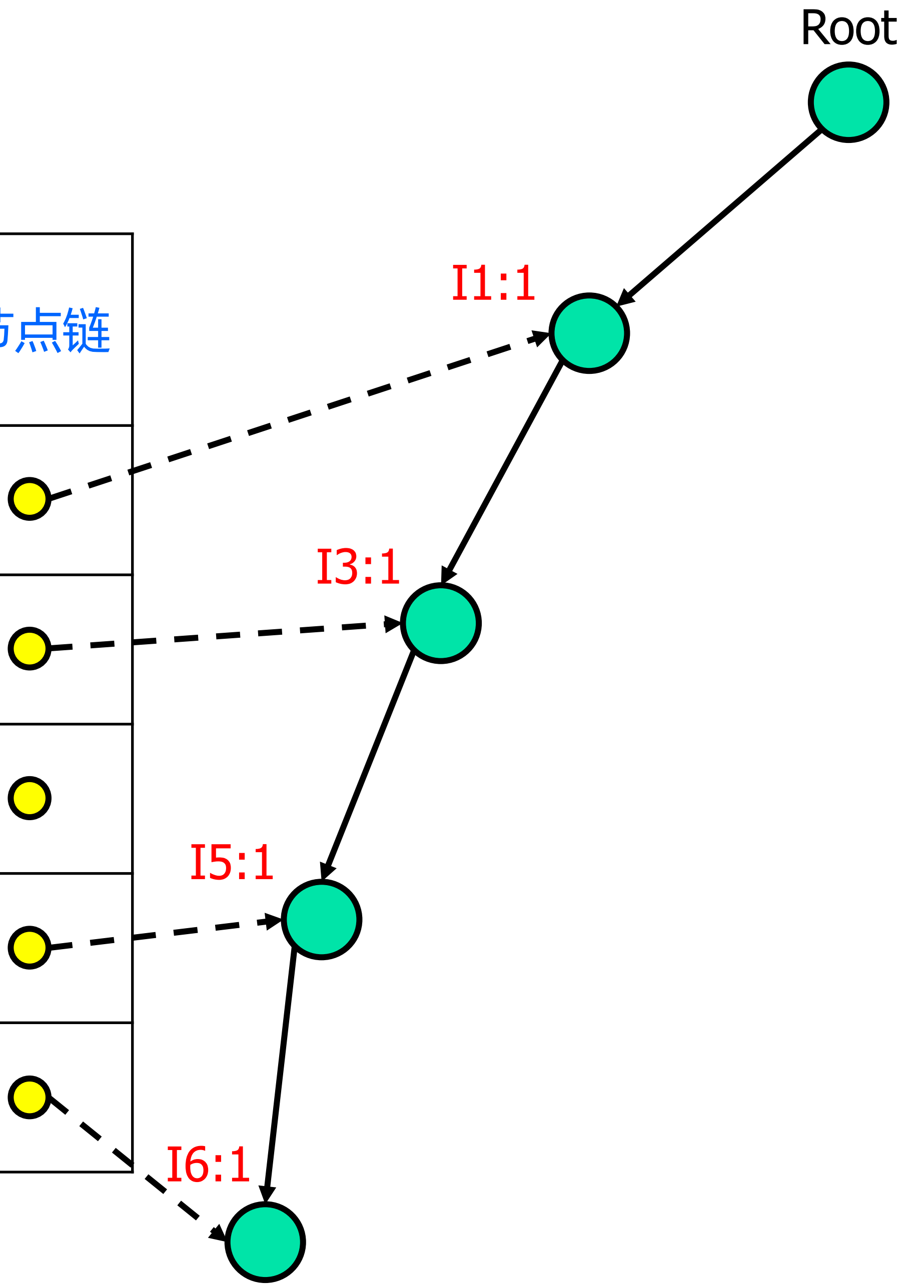
TID	商品ID列表
T001	{I1, I3, I5, I6}
T002	{I1, I3, I4, I5}
T003	{I1, I4}
T004	{I1, I3, I5, I6}
T005	{I3, I4, I6}

(3) 再次扫描数据库，构建FP-tree

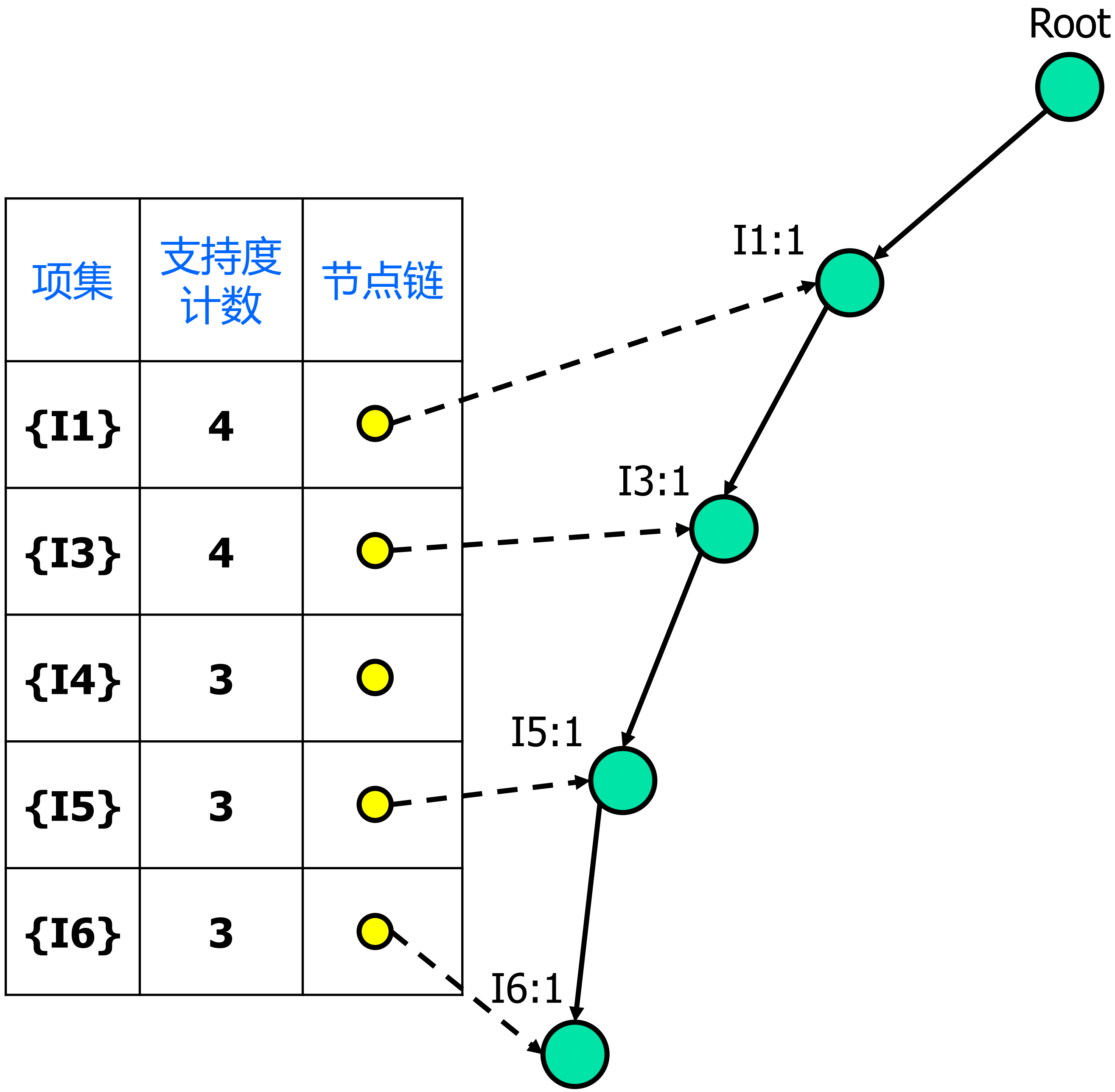
项集	支持度计数	节点链
{I1}	**4**	
{I3}	**4**	
{I4}	**3**	
{I5}	**3**	
{I6}	**3**	

TID（事务ID）	商品ID列表
T001	{I1, I3, I5, I6}
T002	{I1, I3, I4, I5}
T003	{I1, I4}
T004	{I1, I3, I5, I6}
T005	{I3, I4, I6}

项集	支持度计数	节点链
{I1}	**4**	
{I3}	**4**	
{I4}	**3**	
{I5}	**3**	
{I6}	**3**	

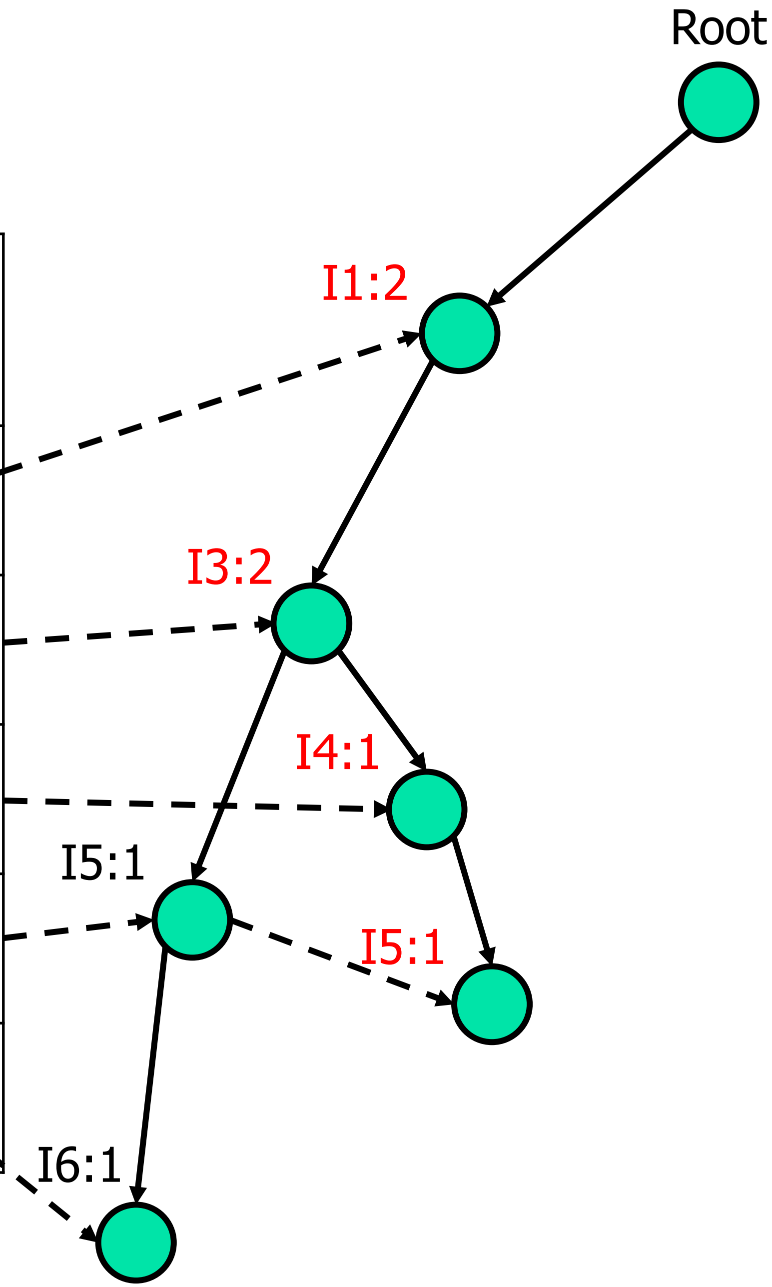


TID（事务ID）	商品ID列表
T001	**{I1, I3, I5, I6}**
T002	{I1, I3, I4, I5}
T003	{I1, I4}
T004	{I1, I3, I5, I6}
T005	{I3, I4, I6}



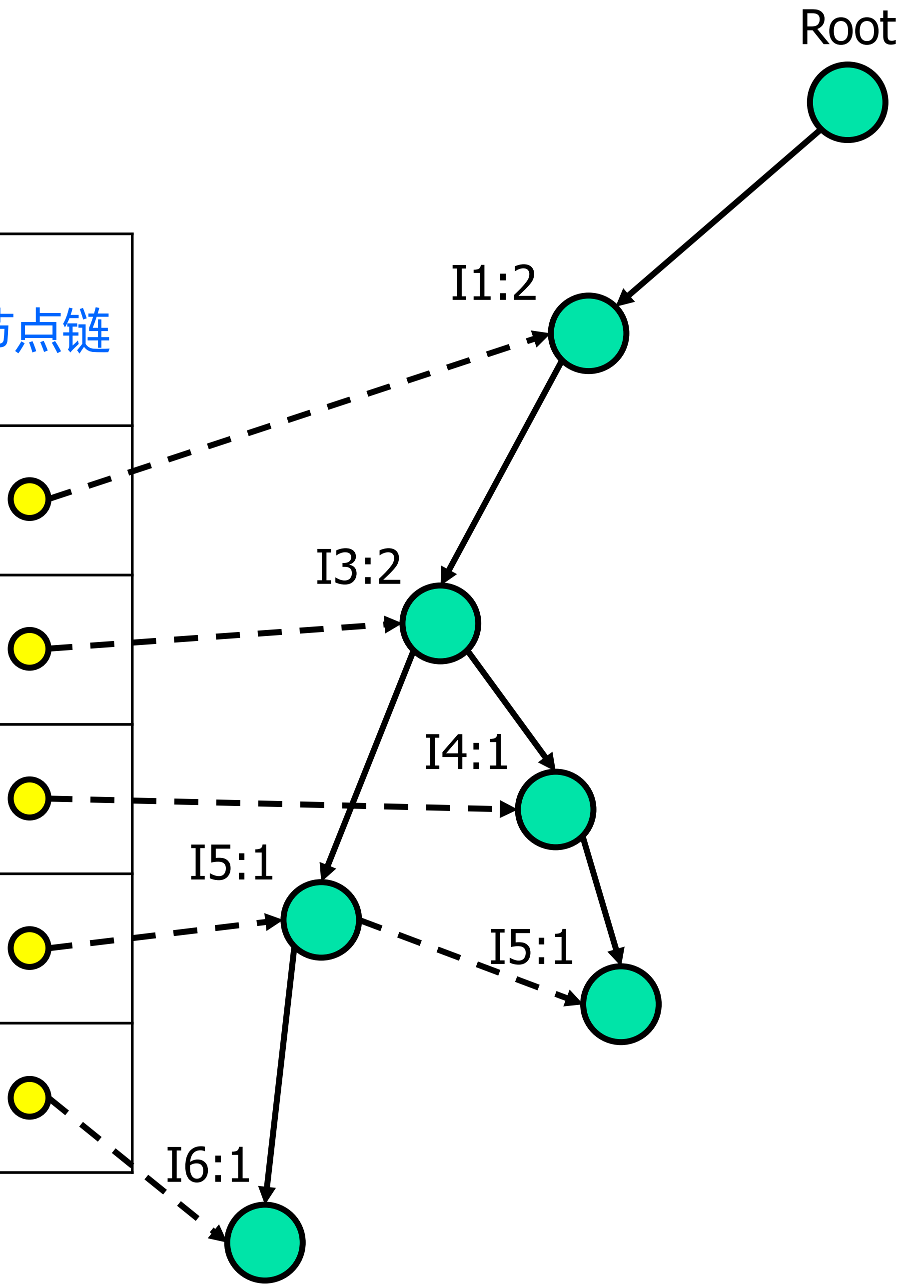
TID（事务ID）	商品ID列表
T001	{I1, I3, I5, I6}
T002	**{I1, I3, I4, I5}**
T003	{I1, I4}
T004	{I1, I3, I5, I6}
T005	{I3, I4, I6}

项集	支持度计数	节点链
{I1}	**4**	
{I3}	**4**	
{I4}	**3**	
{I5}	**3**	
{I6}	**3**	



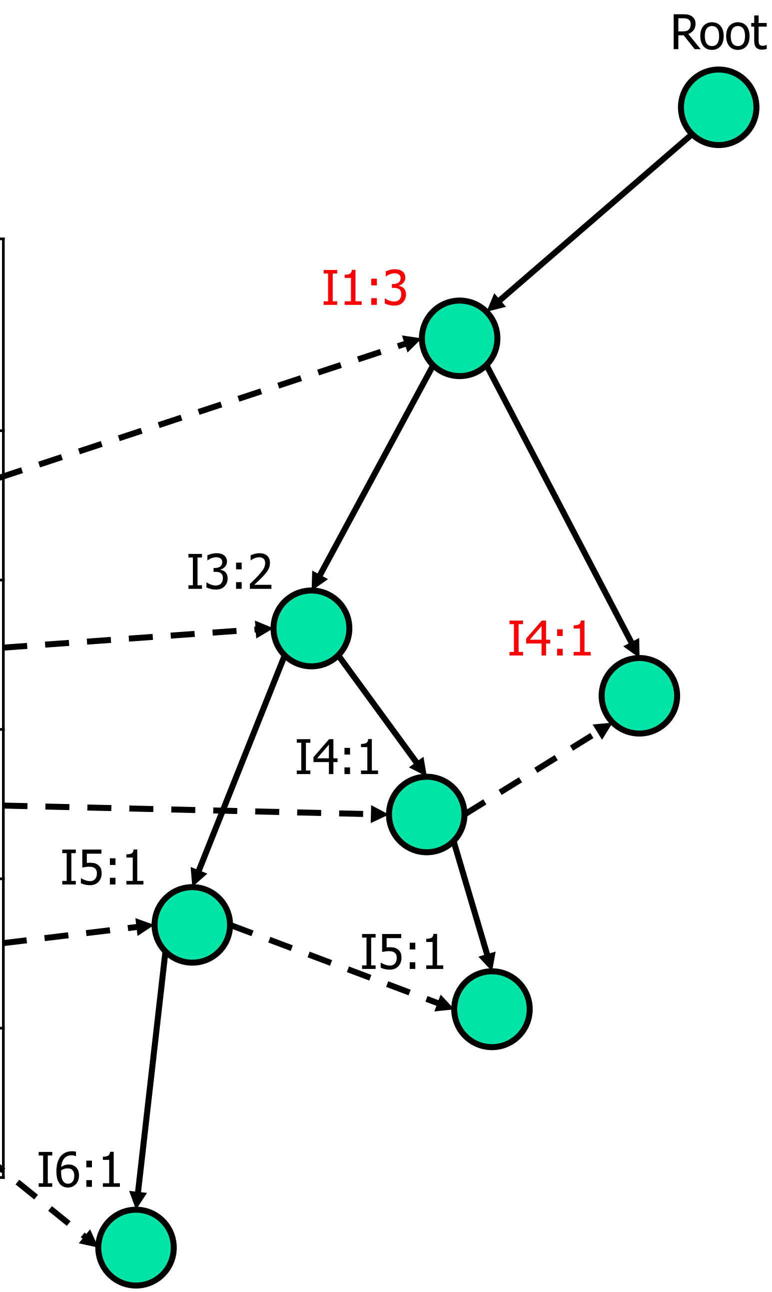
TID（事务ID）	商品ID列表
T001	{I1, I3, I5, I6}
T002	**{I1, I3, I4, I5}**
T003	{I1, I4}
T004	{I1, I3, I5, I6}
T005	{I3, I4, I6}

项集	支持度计数	节点链
{I1}	**4**	
{I3}	**4**	
{I4}	**3**	
{I5}	**3**	
{I6}	**3**	



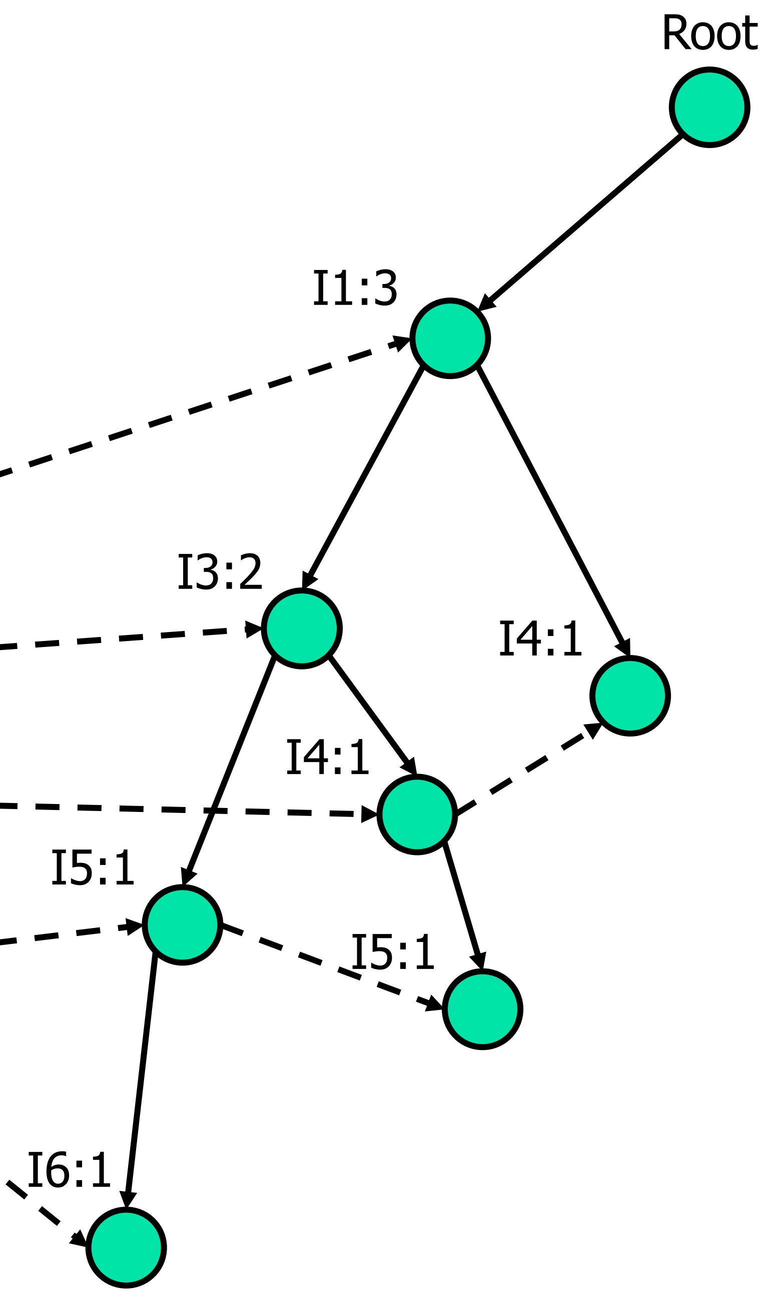
TID（事务ID）	商品ID列表
T001	{I1, I3, I5, I6}
T002	{I1, I3, I4, I5}
T003	**{I1, I4}**
T004	{I1, I3, I5, I6}
T005	{I3, I4, I6}

项集	支持度计数	节点链
{I1}	**4**	
{I3}	**4**	
{I4}	**3**	
{I5}	**3**	
{I6}	**3**	



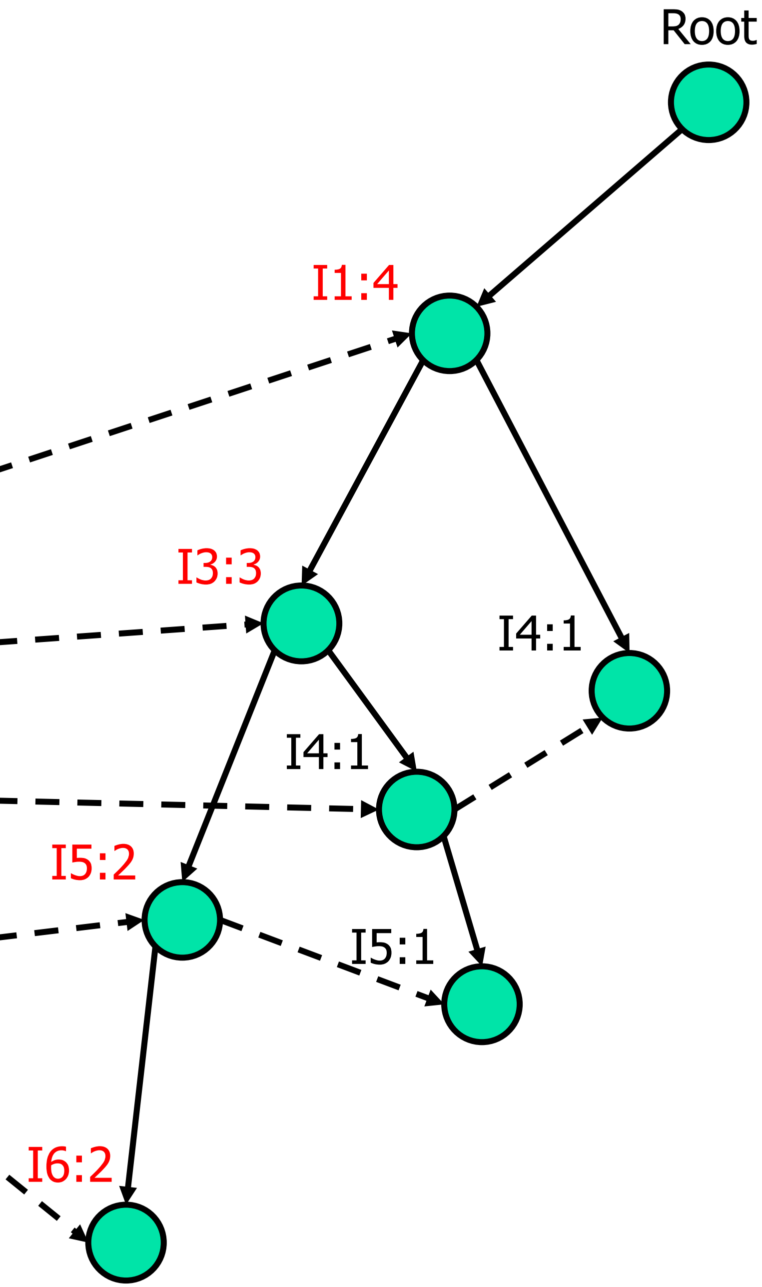
TID（事务ID）	商品ID列表
T001	{I1, I3, I5, I6}
T002	{I1, I3, I4, I5}
T003	**{I1, I4}**
T004	{I1, I3, I5, I6}
T005	{I3, I4, I6}

项集	支持度计数	节点链
{I1}	**4**	
{I3}	**4**	
{I4}	**3**	
{I5}	**3**	
{I6}	**3**	



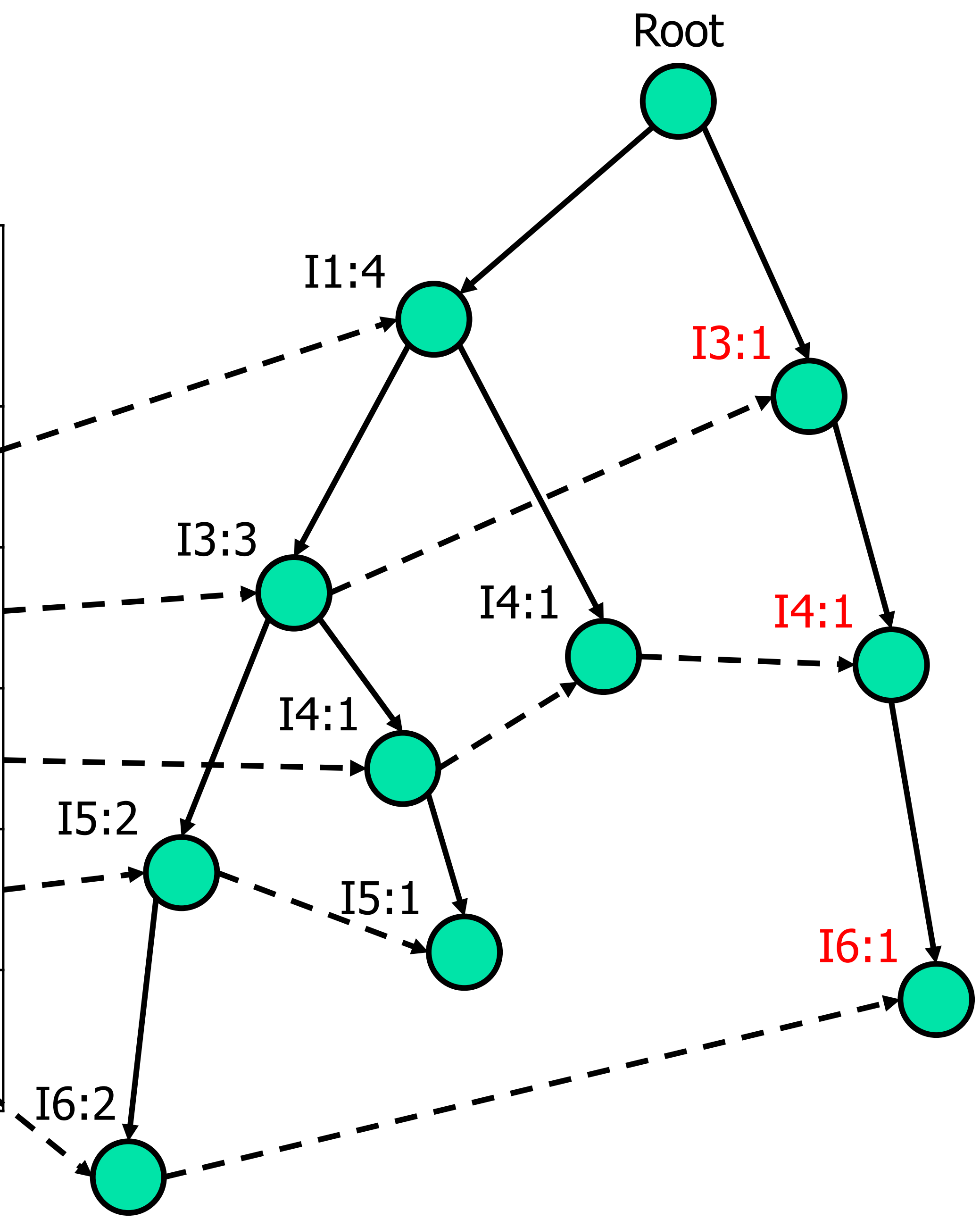
TID（事务ID）	商品ID列表
T001	{I1, I3, I5, I6}
T002	{I1, I3, I4, I5}
T003	{I1, I4}
T004	**{I1, I3, I5, I6}**
T005	{I3, I4, I6}

项集	支持度计数	节点链
{I1}	**4**	
{I3}	**4**	
{I4}	**3**	
{I5}	**3**	
{I6}	**3**	



TID（事务ID）	商品ID列表
T001	{I1, I3, I5, I6}
T002	{I1, I3, I4, I5}
T003	{I1, I4}
T004	**{I1, I3, I5, I6}**
T005	{I3, I4, I6}

项集	支持度计数	节点链
{I1}	**4**	
{I3}	**4**	
{I4}	**3**	
{I5}	**3**	
{I6}	**3**	



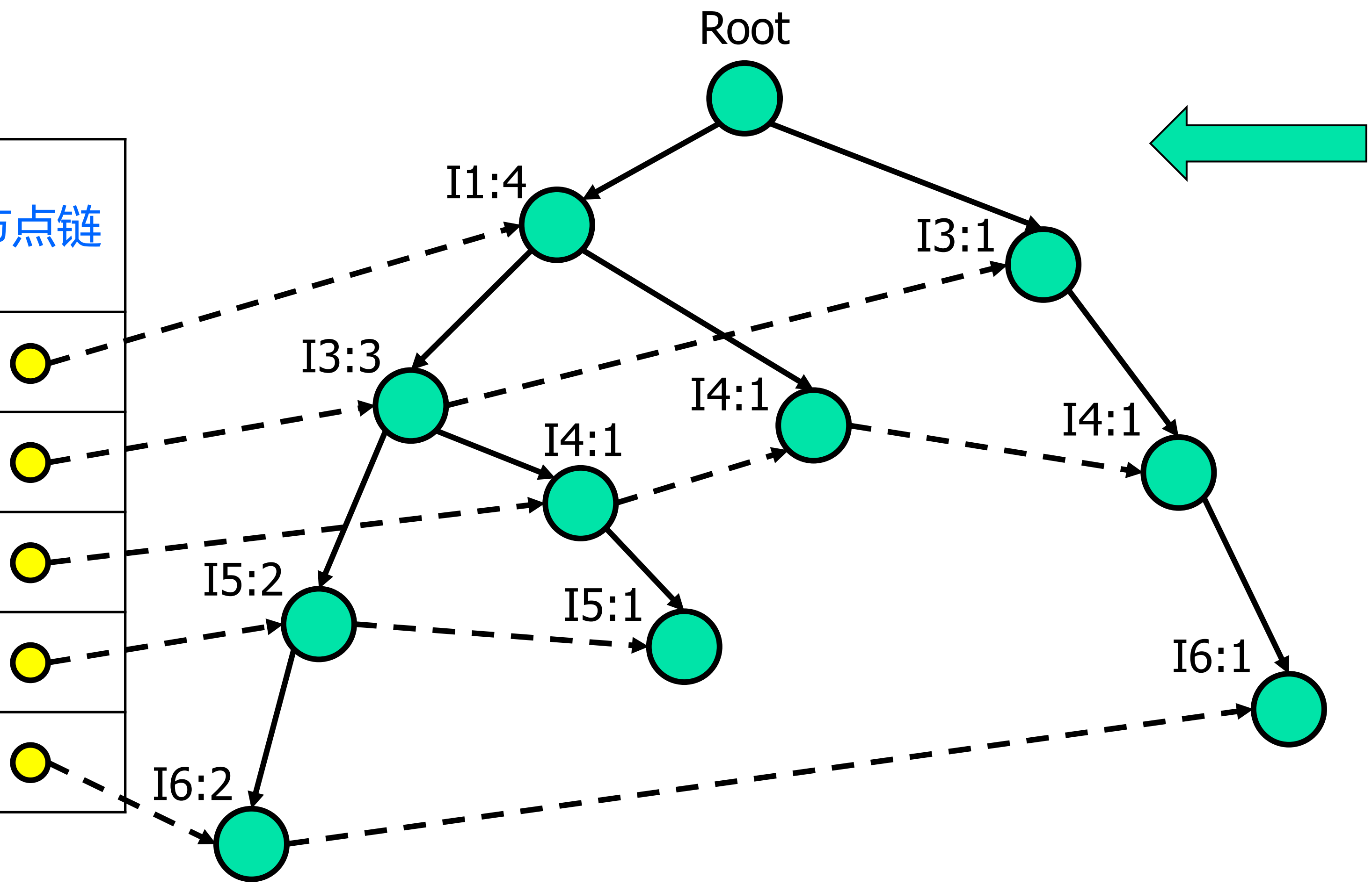
TID（事务ID）	商品ID列表
T001	{I1, I3, I5, I6}
T002	{I1, I3, I4, I5}
T003	{I1, I4}
T004	{I1, I3, I5, I6}
T005	**{I3, I4, I6}**

FP-tree的特性

- 完整性（Completeness）：对于频繁模式发现来说
- 紧凑性（Compactness）
  - 能有效压缩事务数据库
  - 压缩比可超100倍

如何从FP-tree中发现频繁项集？

项集	支持度计数	节点链
{I1}	4	
{I3}	4	
{I4}	3	
{I5}	3	
{I6}	3	



TID	商品ID列表
T001	{I5, I1, I2, I3, I6}
T002	{I4, I5, I1, I3}
T003	{I1, I4, I7}
T004	{I5, I1, I3, I6, I7}
T005	{I4, I3, I6}

- 结点链的作用：找到条件数据库！

从FP-tree中生成频繁项集的算法

- **分治法**：将挖掘全体频繁项集的问题分为多个子问题

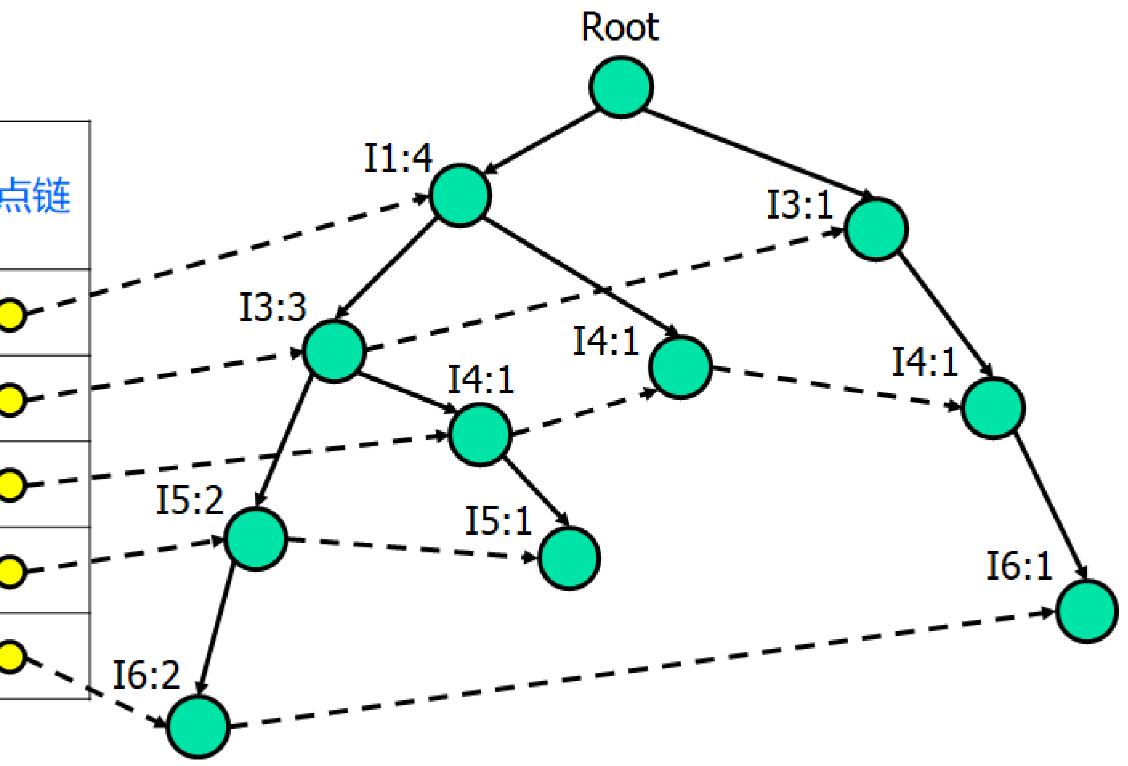
  --挖掘以I6结尾的频繁项集 --挖掘以I5结尾的频繁项集

  --挖掘以I4结尾的频繁项集 --挖掘以I3结尾的频繁项集

  --挖掘以I1结尾的频繁项集

- **递归法：求解每个子问题**

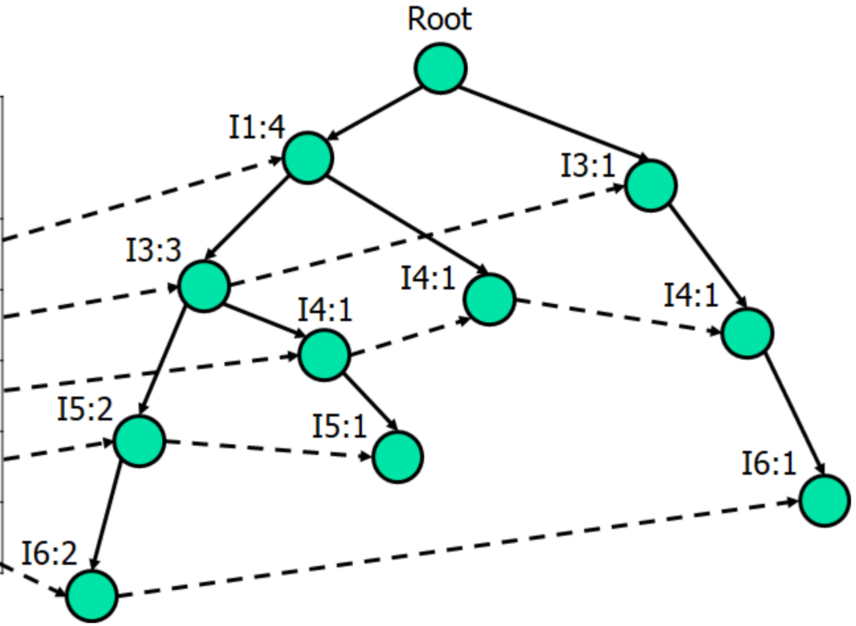
项集	支持度计数	节点链
{I1}	4	
{I3}	4	
{I4}	3	
{I5}	3	
{I6}	3	



Procedure FP-growth($Tree$, α)

- **if** $Tree$ 只包含单个分支 P **then**
  - **for each** θ (节点组合) **in** C_θ (P中节点的全部组合)
    - ✓ 生成新的频繁项集: $\boldsymbol{\beta} = \theta \cup \boldsymbol{\alpha}$
- **else for each entry** e_i **in** head table（头部表）
  - 生成新的频繁项集: $\boldsymbol{\beta} = e_i \cup \boldsymbol{\alpha}$
  - 构建 $\boldsymbol{\beta}$ 的条件数据库: D_β
  - 基于D_β构建 $\boldsymbol{\beta}$ 的条件FP-tree: $Tree_\beta$
  - **If** $Tree_\beta \neq \emptyset$ **then**
    - ✓ 递归调用 **FP_growth($Tree_\beta$, β)**

项集	支持度计数	节点链
{I1}	4	○
{I3}	4	○
{I4}	3	○
{I5}	3	○
{I6}	3	○



➢ **递归的停止条件:**

- ○ **if** $Tree$ 只包含单个分支 P
- ○ **if** $Tree_\beta = \emptyset$

Procedure FP-growth(*Tree*, α)

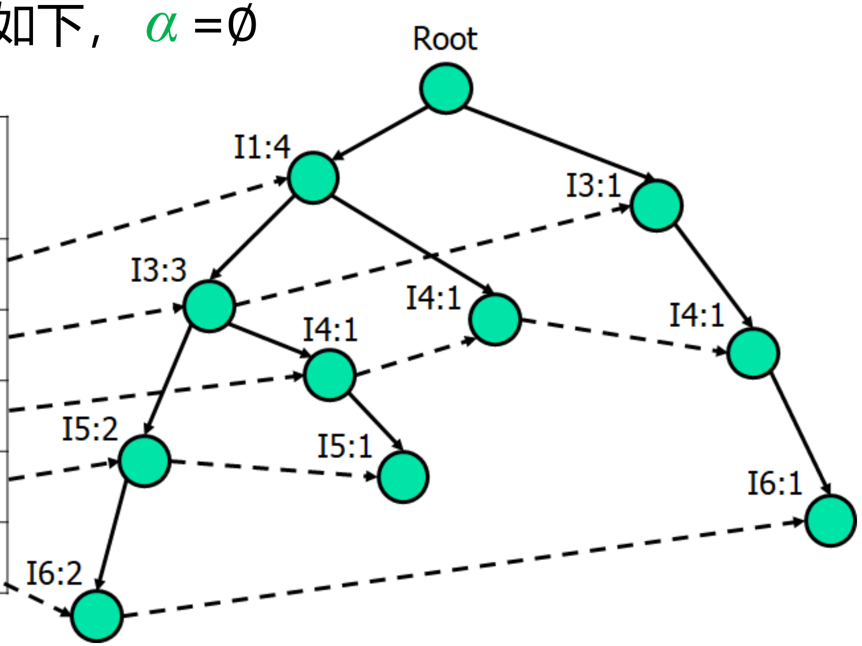
- **if *Tree* 只包含单个分支 P then**
  - **for each θ (节点组合) in C_θ (P中节点的全部组合)**
    - ✓ 生成新的频繁项集: $\boldsymbol{\beta} = \theta \cup \boldsymbol{\alpha}$
- **else for each entry e_i in head table（头部表）**
  - 生成新的频繁项集: $\boldsymbol{\beta} = e_i \cup \boldsymbol{\alpha}$
  - 构建 $\boldsymbol{\beta}$ 的条件数据库：D_β
  - 基于D_β构建 $\boldsymbol{\beta}$ 的条件FP-tree：$Tree_\beta$
  - **If $Tree_\beta \neq \emptyset$ then**
    - ✓ 递归调用 **FP_growth($Tree_\beta, \beta$)**

e_i ={I6} $\boldsymbol{\beta} = e_i \cup \boldsymbol{\alpha}$ = {I6}

- 第1个子问题：挖掘以I6结尾的频繁项集

最初调用，参数Tree如下， $\alpha = \emptyset$

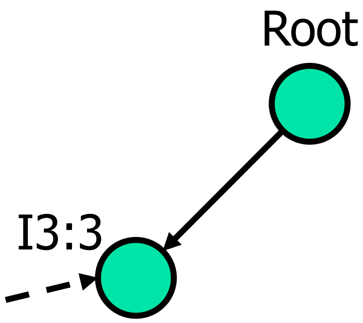
项集	支持度计数	节点链
{I1}	4	
{I3}	4	
{I4}	3	
{I5}	3	
{I6}	3	



D_β: {I1, I3, I5}:2, {I3, I4}:1

$Tree_\beta$:

项集	支持度	节点链
{I3}	3	

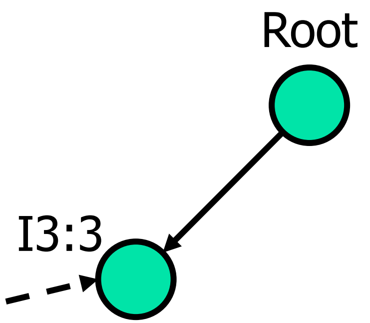


Procedure FP-growth(*Tree*, α)

递归调用，参数Tree如下， α ={I6}

- **if** ***Tree*** 只包含单个分支 P **then**
  - **for each** θ (节点组合) **in** C_θ (P中节点的全部组合)
    - ✓ 生成新的频繁项集: $\boldsymbol{\beta} = \theta \cup \boldsymbol{\alpha}$
- **else for each entry** $\boldsymbol{e_i}$ **in** head table（头部表）
  - 生成新的频繁项集: $\boldsymbol{\beta} = \boldsymbol{e_i} \cup \boldsymbol{\alpha}$
  - 构建 $\boldsymbol{\beta}$ 的条件数据库： D_β
  - 基于D_β构建 $\boldsymbol{\beta}$ 的条件FP-tree： $Tree_\beta$
  - **If** $Tree_\beta \neq \emptyset$ **then**
    - ✓ 递归调用 **FP_growth(**$Tree_\beta, \beta$**)**

项集	支持度	节点链
{I3}	3	



生成新的频繁项集:

$\theta \cup \alpha$ = {I3} ∪ {I6} = **{I3, I6}**

➢ **解决第1个子问题**：挖掘以I6结尾的频繁项集

Procedure FP-growth(*Tree*, α)

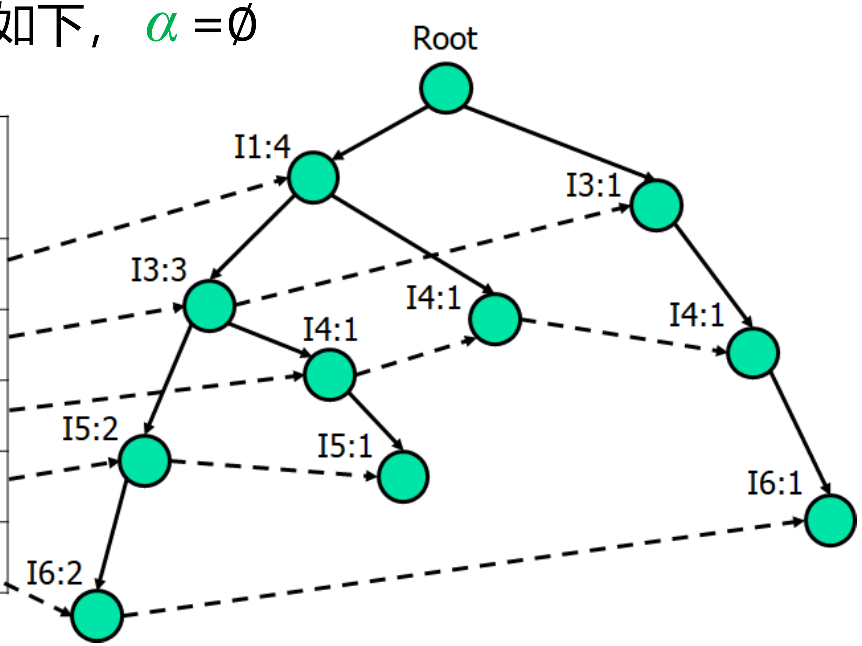
- **if** ***Tree*** **只包含单个分支** P **then**
  - **for each** θ (节点组合) **in** C_θ (P中节点的全部组合)
    - ✓ 生成新的频繁项集: $\boldsymbol{\beta} = \theta \cup \boldsymbol{\alpha}$
- **else for each entry** e_i **in** head table（头部表）
  - **生成新的频繁项集:** $\boldsymbol{\beta} = \boldsymbol{e_i} \cup \boldsymbol{\alpha}$
  - **构建** $\boldsymbol{\beta}$ **的条件数据库：** D_β
  - **基于**D_β**构建** $\boldsymbol{\beta}$ **的条件FP-tree：** $Tree_\beta$
  - **If** $Tree_\beta \neq \emptyset$ **then**
    - ✓ 递归调用 **FP_growth(**$Tree_\beta, \beta$**)**

e_i ={I5}　　　$\boldsymbol{\beta} = \boldsymbol{e_i} \cup \boldsymbol{\alpha}$= {I5}

- 第2个子问题：挖掘以I5结尾的频繁项集

最初调用，参数Tree如下， $\alpha = \emptyset$

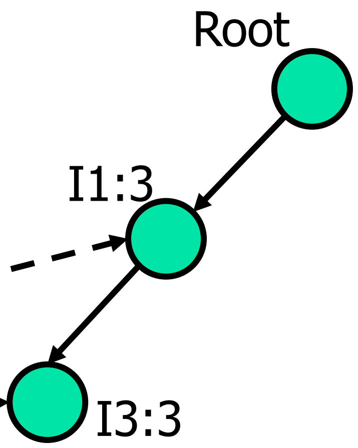
项集	支持度计数	节点链
{I1}	4	
{I3}	4	
{I4}	3	
{I5}	3	
{I6}	3	



D_β: {I1, I3}:2, {I1, I3, I4}:1

$Tree_\beta$:

项集	支持度	节点链
{I1}	3	
{I3}	3	

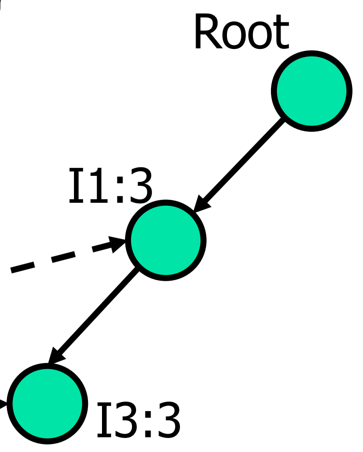


Procedure FP-growth($Tree$, α)

- **if** $Tree$ 只包含单个分支 P **then**
  - **for each** θ (节点组合) **in** C_θ (P中节点的全部组合)
    - ✓ 生成新的频繁项集: $\boldsymbol{\beta} = \theta \cup \boldsymbol{\alpha}$
- **else for each entry** e_i **in** head table（头部表）
  - 生成新的频繁项集: $\boldsymbol{\beta} = e_i \cup \boldsymbol{\alpha}$
  - 构建 $\boldsymbol{\beta}$ 的条件数据库：D_β
  - 基于D_β构建 $\boldsymbol{\beta}$ 的条件FP-tree：$Tree_\beta$
  - **If** $Tree_\beta \neq \emptyset$ **then**
    - ✓ 递归调用 **FP_growth(**$Tree_\beta, \beta$**)**

递归调用，参数Tree如下， α ={I5}

项集	支持度	节点链
{I1}	3	
{I3}	3	



生成新的频繁项集:

$\theta \cup \alpha$ = {I3} ∪ {I5} = **{I3, I5}**

$\theta \cup \alpha$ = {I1} ∪ {I5} = **{I1, I5}**

$\theta \cup \alpha$ = {I1, I3} ∪ {I5} = **{I1, I3, I5}**

- 解决第2个子问题：挖掘以I5结尾的频繁项集

- 接着依次求解以I4、I3、I1结尾的频繁项集集合，得到最终结果

后缀项	条件数据库	挖掘到的频繁项集
{I6}	{I1, I3, I5}:2, {I3, I4}:1	{I3, I6}
{I5}	{I1, I3}:2, {I1, I3, I4}:1	{I1, I5}, {I3, I5}, {I1, I3, I5}
{I4}	{I1}:1, {I3}:1, {I1, I3}:1	\
{I3}	{I1}:3	{I1, I3}
{I1}	\	\

L1	{I1}, {I3}, {I4}, {I5}, {I6}
L2	{I1, I3}, {I1, I5}, {I3, I5}, {I3, I6}
L3	{I1, I3, I5}

FP-tree算法的特点分析

- 优点
  - 使用一个高度压缩的数据结构存储了事务数据库的信息，整个过程只需扫描两次数据集，相关研究表明，在挖掘某些事务数据集时，FP-tree算法比Apriori算法快多个数量级。
- 缺点
  - 由于FP-tree算法在执行过程中需要递归生成条件数据库和条件FP-tree，所以内存开销较大，且当生成的FP-tree十分茂盛时，如满前缀树，算法产生的子问题数量会剧增，导致性能显著下降。

Acknowledgements

- Some text, figures and formulations are from WWW. Thanks for their sharing. If you have copyright claim please contact with me at yym@hit.edu.cn.
- This lecture is distributed for nonprofit purpose.

Thank You for Your Attention

Contact me at: yym@hit.edu.cn

Tel: 26033008, 13760196623

Address: Rm.1402, H# Building