Mining Complex Data

COMP 790-90 Seminar

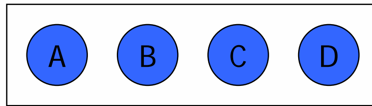
Spring 2009

Mining Complex Patterns

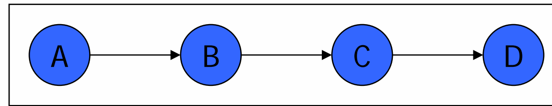
- Common Pattern Mining Tasks:
  - **Itemsets** (transactional, unordered data)
  - **Sequences** (temporal/positional: text, bioseqs)
  - **Tree patterns** (semi-structured/XML data, web mining)
  - **Graph patterns** (protein structure, web data, social network)

Example Pattern Types

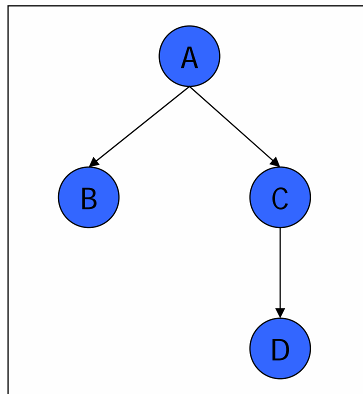
Itemset



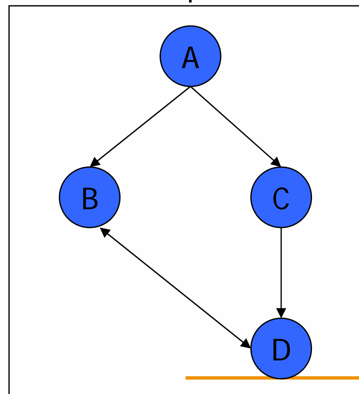
Sequence



Tree



Graph



- Can add attributes
  - To nodes
  - To edges
- **Attributes**
  - Labels
  - Type (directed or undirected)
  - Set-valued

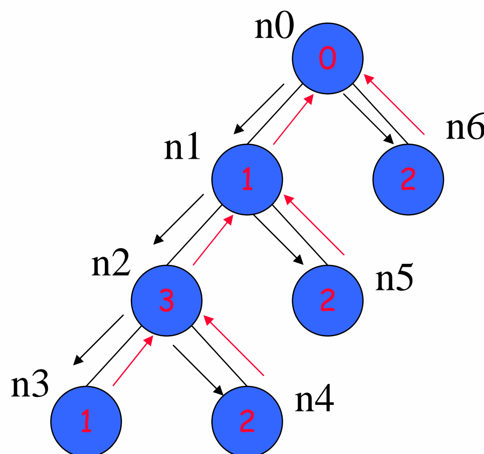
Induced vs Embedded Sub-trees

- Induced Sub-trees: $S = (V_s, E_s)$ is a sub-tree of $T = (V,E)$ if and only if
  - $V_s \subseteq V$
  - $e = (n_x, n_y) \in E_s$ iff $(n_x, n_y) \in E$ (n_x directly connected to n_y)
- Embedded Sub-trees: $S = (V_s, E_s)$ is a sub-tree of $T = (V,E)$ if and only if
  - $V_s \subseteq V$
  - $e = (n_x, n_y) \in E_s$ iff $n_x \leq_l n_y$ in T (n_x connected to n_y)
- An induced sub-tree is a special case of embedded sub-tree.
- We say S *occurs* in T and T *contains* S if S is an embedded sub-tree of T
- If S has *k* nodes, we call it a *k*-sub-tree

Mining Frequent Trees

- Support: the *support* of a subtree in a database of trees, is the number of trees containing the subtree.
- A subtree is frequent if its support is at least the minimum support.
- TreeMiner: Given a database of trees (a forest) and a minimum support, find all frequent subtrees.

String Representation of Trees



0 1 3 1 -1 2 -1 -1 2 -1 -1 2 -1

With N nodes, M branches, F max fanout

Adjacency Matrix requires: N(F+1) space

Adjacency List requires: 4N-2 space

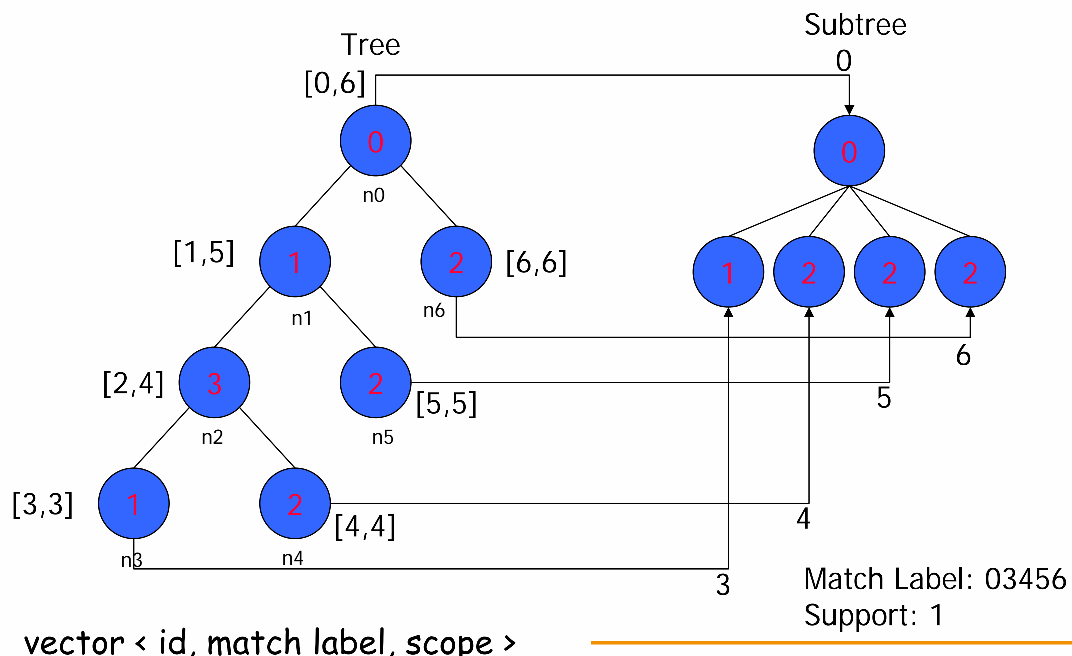
Tree requires (node, child, sibling): 3N space

String representation requires: 2N-1 space

Tree: String Representation

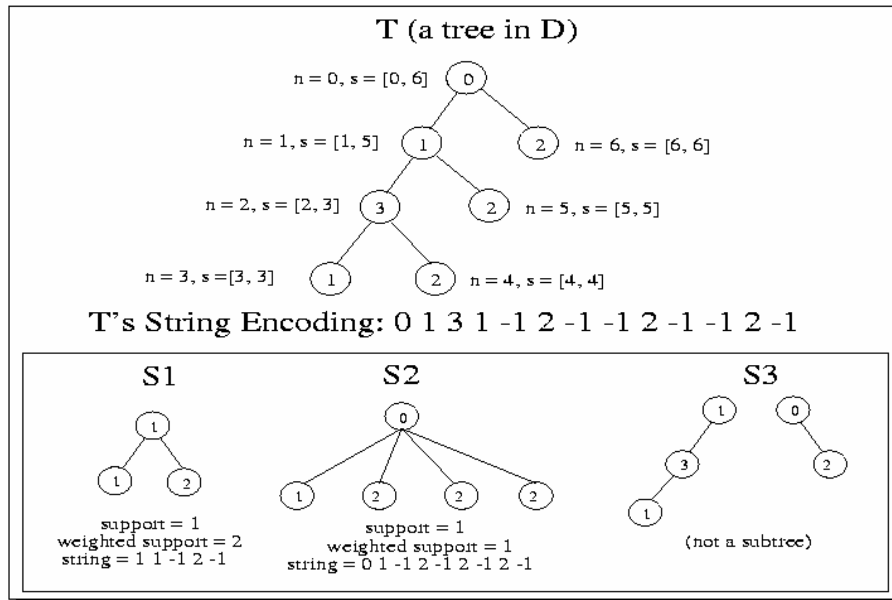
- Like an itemset
- -1 as the backtrack item
- Assuming only labels on nodes
- For trees labels on edges can be treated as labels on nodes:
  - edge-label+node-label = new label!

Match labels



vector < id, match label, scope >

An example



Generic Mining Algorithms

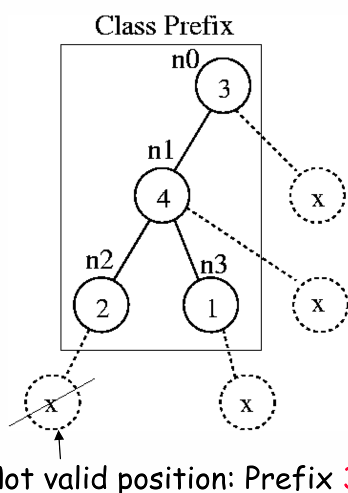
- Horizontal pattern matching based
- Vertical intersection based
- BFS or DFS

Candidate Generation & Support Counting

- Candidate Generation
  - Extend by a node or an edge
  - Avoid duplicates as far as possible

Trees: Systematic Candidate Generation

Two subtrees are in the same class iff they share a common prefix string *P* up to the (k-1)th node



Not valid position: Prefix 3 4 2 x

Equivalence Class

Prefix String: 3 4 2 −1 1

Element List: (label, attached to position)

(x, 0) // attached to n0: 3 4 2 −1 1 −1 −1 x −1

(x, 1) // attached to n1: 3 4 2 −1 1 −1 x −1 −1

(x, 3) // attached to n3: 3 4 2 −1 1 x −1 −1 −1

A valid element *x* attached to only the nodes lying on the path from root to **rightmost leaf** in prefix *P*

Candidate generation

- Given an equivalence class of k-subtrees, how do we generate candidate (k+1)-subtrees?
- Main idea: consider each ordered pair of elements in the class for extension, including self extension
  - Sort elements by node label and position

Class extension

Theorem 1 (Class Extension) *Let P be a prefix class with encoding $\mathcal{P}$, and let (x, i) and (y, j) denote any two elements in the class. Let $\mathcal{X}$ denote the subtree encoding for element x, and let $\mathcal{Y}$ denote the encoding for y. Define a* join operator $\otimes$ *on the two elements, given as $(x, i) \otimes (y, j)$:*

case I – $(i = j)$: *If $\mathcal{P} \neq \emptyset$, add (y, j) and $(y, j+1)$ to the new class $[\mathcal{X}]$.*

If $\mathcal{P} = \emptyset$, add $(y, j+1)$ to the new class $[\mathcal{X}]$.

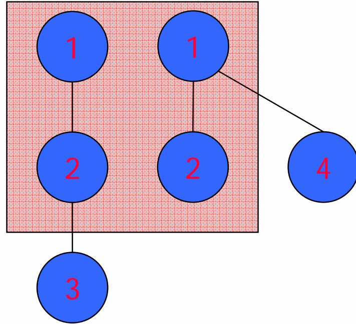
case II – $(i > j)$: *add (y, j) to the new class $[\mathcal{X}]$.*

case III – $(i < j)$: *no new candidate is possible in this case.*

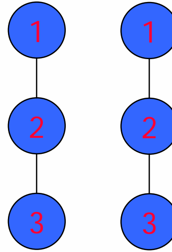
Then all possible $(k+1)$-subtrees with the prefix P of size $k-1$ will be enumerated by applying the join operator to each unordered pair of elements (x, i) and (y, j).

Candidate Generation (Join operator)

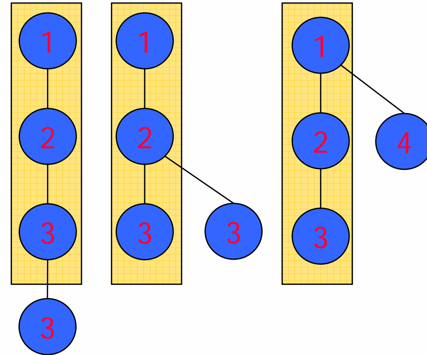
Equivalence Class
Prefix: 1 2, Elements: (3,1) (4,0)



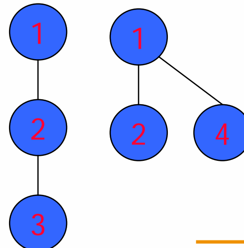
Self Join



New Candidates



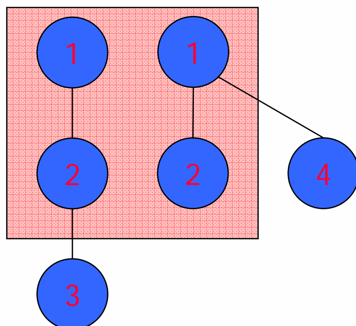
Join



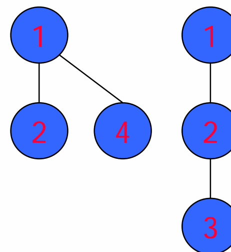
New Equivalence Class
Prefix: 1 2 3
Elements: (3,1) (3,2) (4,0)

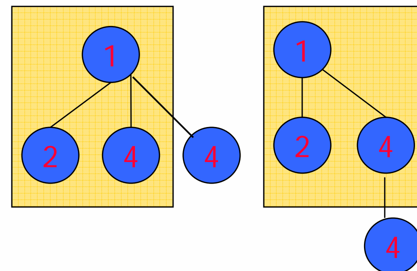
Candidate Generation (Join operator)

Equivalence Class
Prefix: 1 2, Elements: (3,1) (4,0)

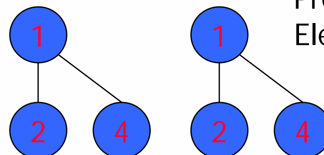


Join



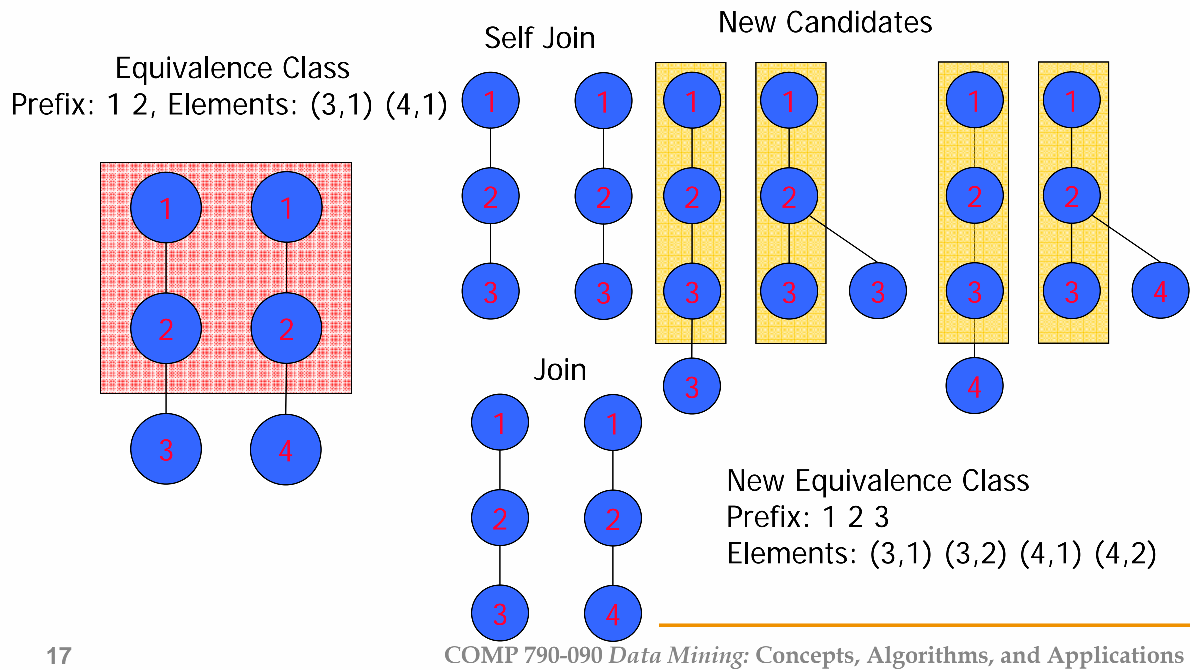


Self Join

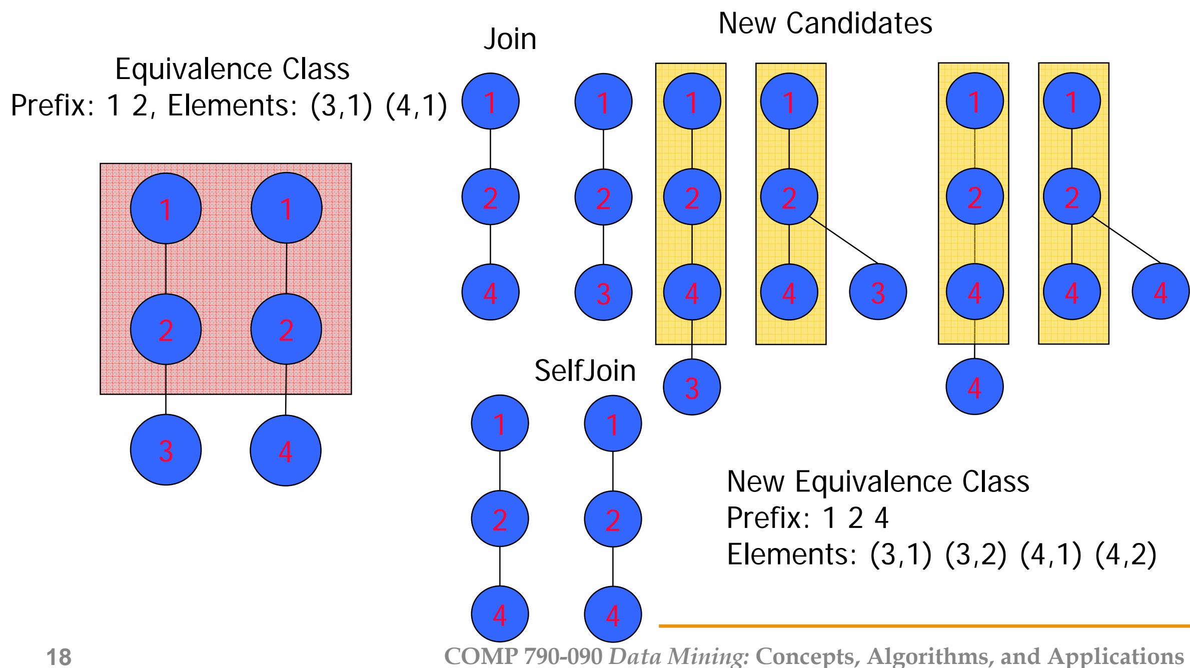


New Equivalence Class
Prefix: 1 2 4
Elements: (4,0) (4,1)

Candidate Generation (Join operator)



Candidate Generation (Join operator)

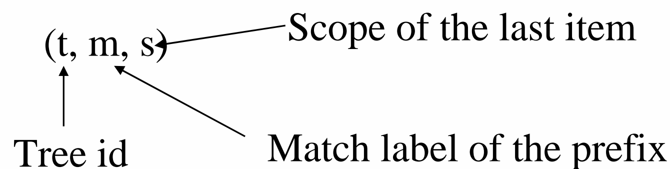


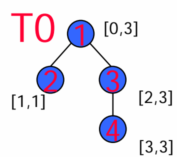
Apriori Style TreeMiner

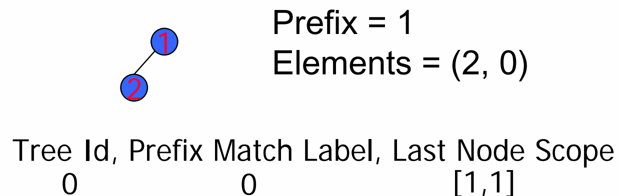
TreeMinerH (D, min_sup):
$F_1 = \{$ frequent 1-subtrees $\}$;
$F_2 = \{$ classes of frequent 2-subtrees $\}$;
for ($k = 3$; $F_{k-1} \neq \emptyset$; $k = k + 1$) do
  $C_k = \{$ classes $[P]_{k-1}$ of candidate k-subtrees $\}$;
  for all trees T in D do
    Increment count of all $S \preceq T$, $S \in [P]_{k-1}$
  $F_k = \{$ classes of frequent k-subtrees $\}$;
Set of all frequent subtrees $= \bigcup_k F_k$;

Depth first version of TreeMiner

Scope-list for each element of a class



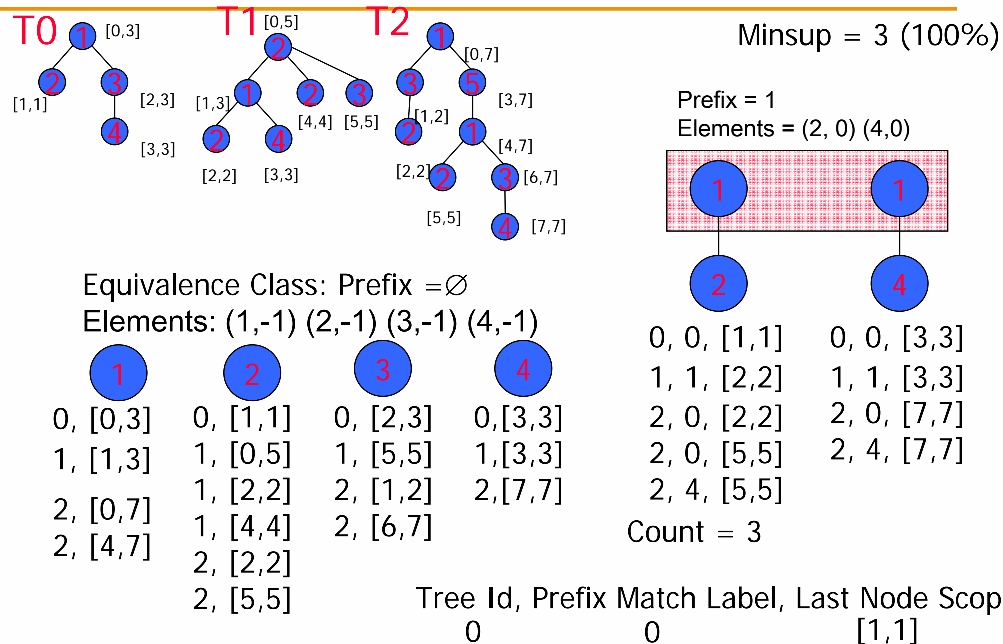




Scope-List Joins

- Join elements (x, i) and (y, j)
  - In-scope test
    - Add (y, j+1) when i = j
      - Add y as a child of x
    - Check whether there exist (t_x, m_x, s_x) and (t_y, m_y, s_y) s.t.
      - $t_x = t_y$
      - $m_x = m_y$
      - $s_y \subset s_x$,

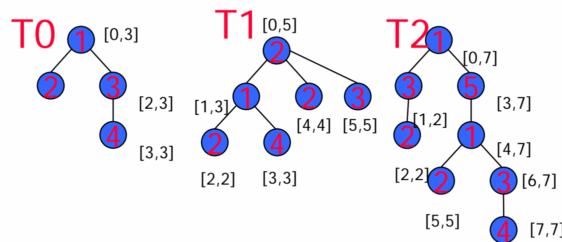
Frequency Computation: Scope List Joins - In Scope



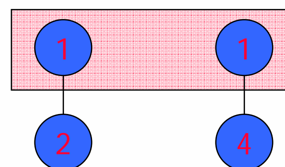
Scope-List Joins

- Join elements (x, i) and (y, j)
  - out-scope test
    - Add (y, j)
    - Add y as a sibling of x
    - Check whether there exist (t_x, m_x, s_x) and (t_y, m_y, s_y) s.t.
      - $t_x = t_y$
      - $m_x = m_y$
      - $s_y > s_x$,

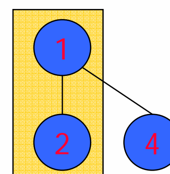
Scope List Joins: Out Scope



Prefix = 1
Elements = (2, 0) (4,0)



0, 0, [1,1]	0, 0, [3,3]
1, 1, [2,2]	1, 1, [3,3]
2, 0, [2,2]	2, 0, [7,7]
2, 0, [5,5]	2, 4, [7,7]
2, 4, [5,5]	

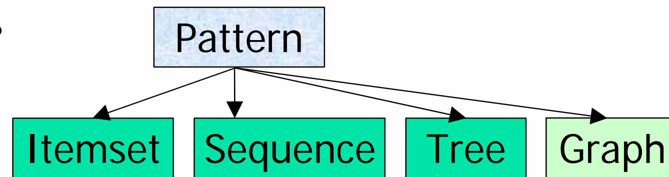


Prefix = 12
Elements = (4,0)

0, 01, [3,3]
1, 12, [3,3]
2, 02, [7,7]
2, 05, [7,7]
2, 45, [7,7]

Generic Pattern Class

- Types of patterns
  - Itemset
  - Sequence
  - Tree & Graphs
  - Define your own!



- Apriori property
  - Every sub-pattern of a frequent pattern is frequent
  - Every super-pattern of an infrequent pattern is infrequent
- Order to explore the solution space
  - Breadth-first level-wise
  - Depth-first projection based