

November 10, 2017



Inter-University Research Institute Corporation /
Research Organization of Information and Systems

National Institute of Informatics

Graph Mining

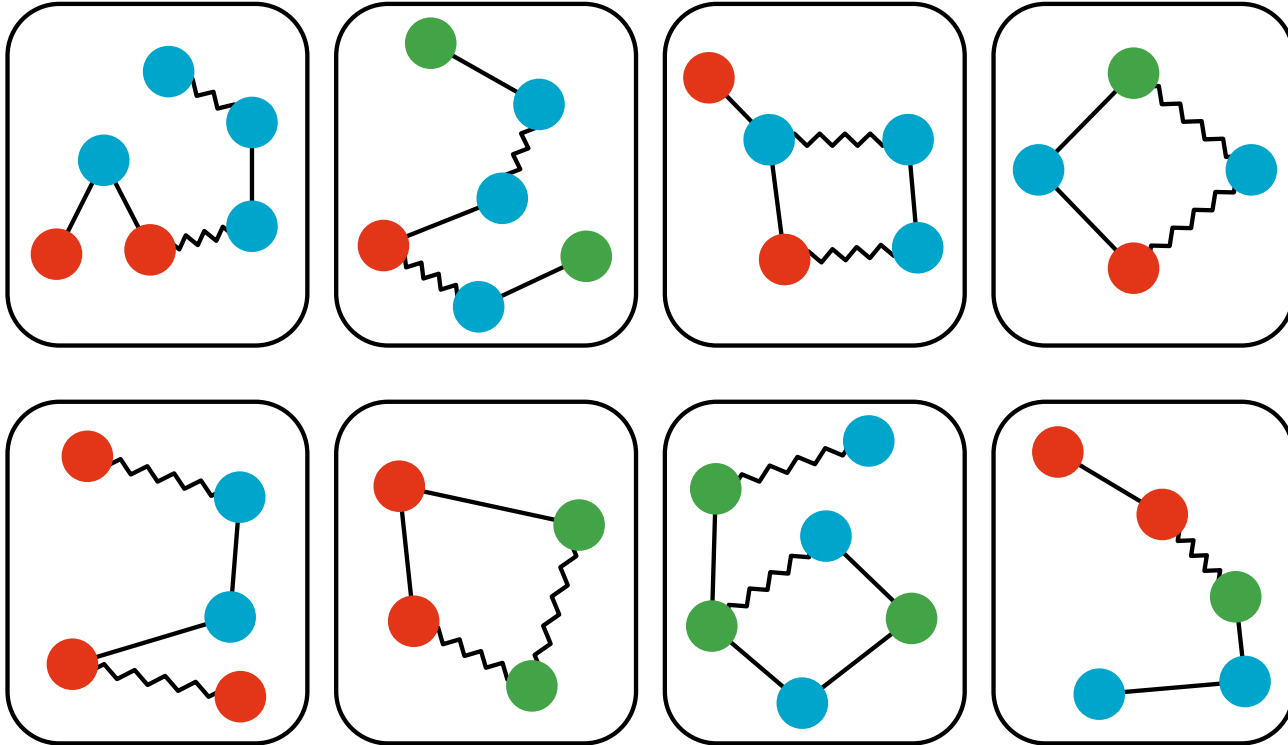
Data Mining 03 (データマイニング)

Mahito Sugiyama (杉山磨人)

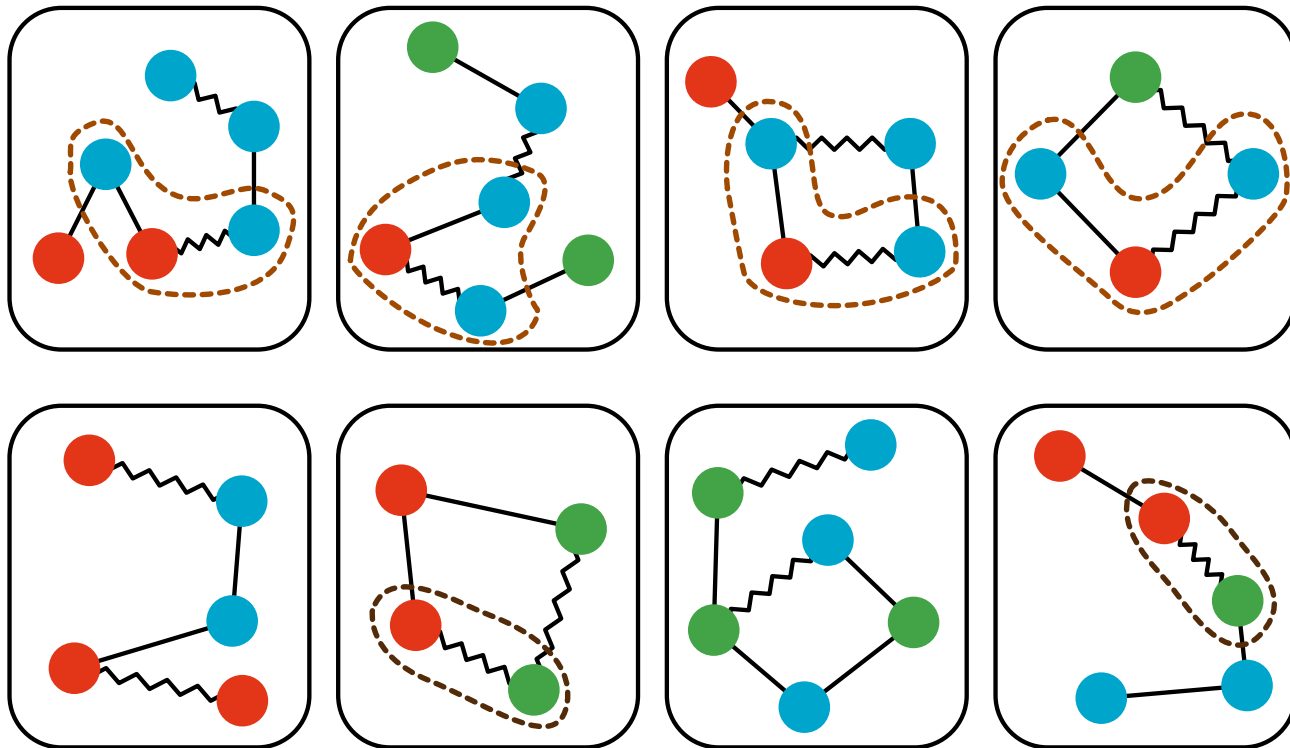
Today's Outline

- A primer of graphs
 - Subgraph isomorphism
- Graph mining
 - How to find (sub)graphs from graph databases?
 - Revisiting the Apriori principle to avoid combinatorial explosion
 - The canonical DFS code for graph representation

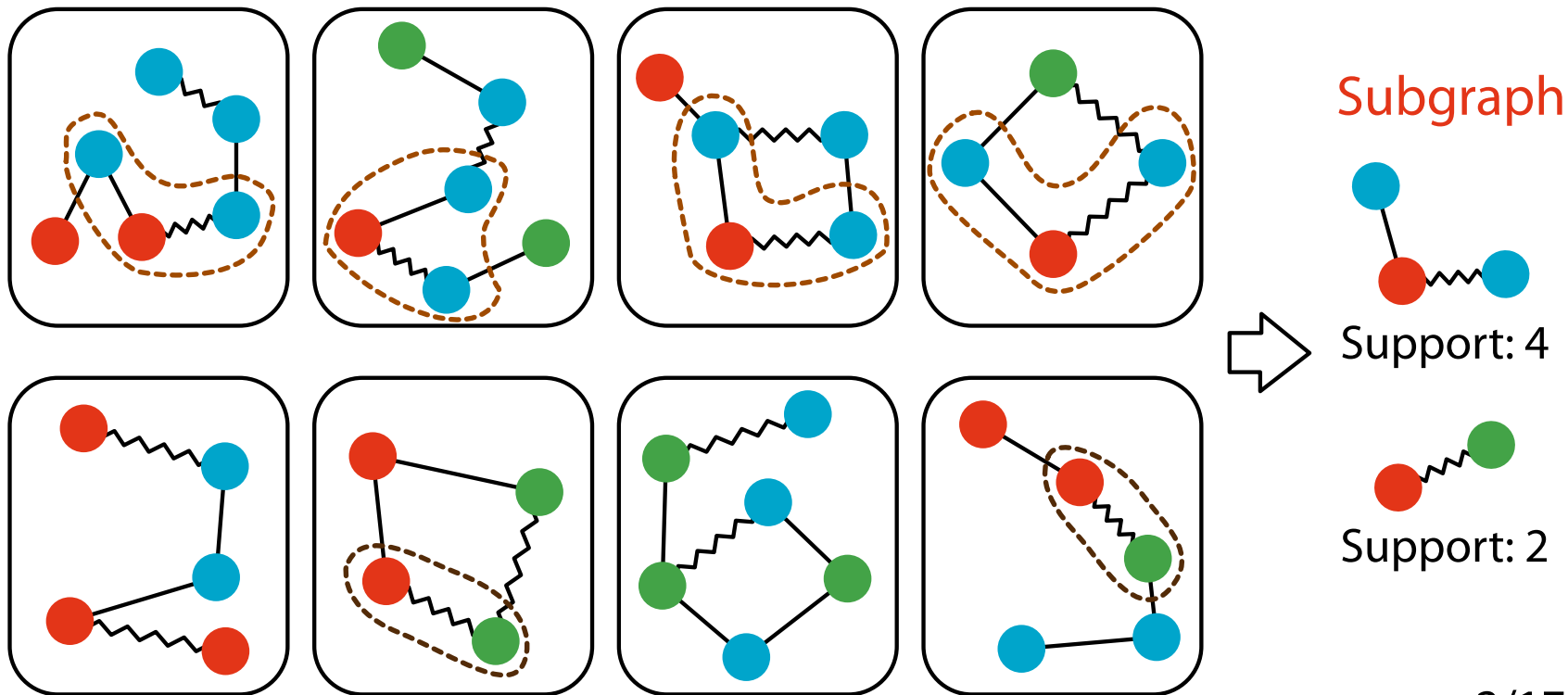
Graph Mining: Overview



Graph Mining: Overview



Graph Mining: Overview



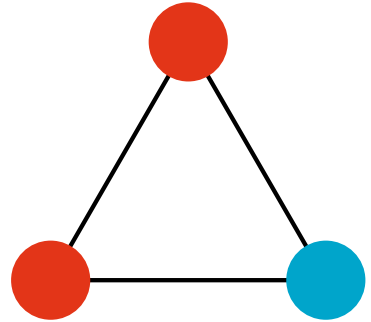
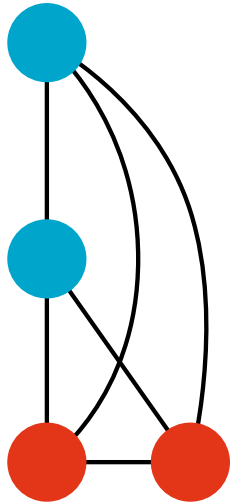
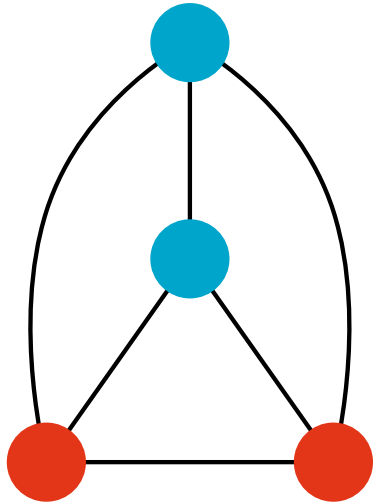
Graphs

- An (unlabeled) graph $G = (V, E)$
 - V : a vertex set, $E \subseteq V \times V$: an edge set
 - For $(u, v) \in E$, u, v are **adjacent**, v is a **neighbor** of u
 - (u, v) and (v, u) are identified if the graph is undirected
 - $N(v) = \{u \in V \mid (v, u) \in E\}$, the set of all neighbors
- A labeled graph $G = (V, E, \varphi)$
 - $\varphi : V \cup E \rightarrow \Sigma$, where Σ is the set of vertex and edge labels

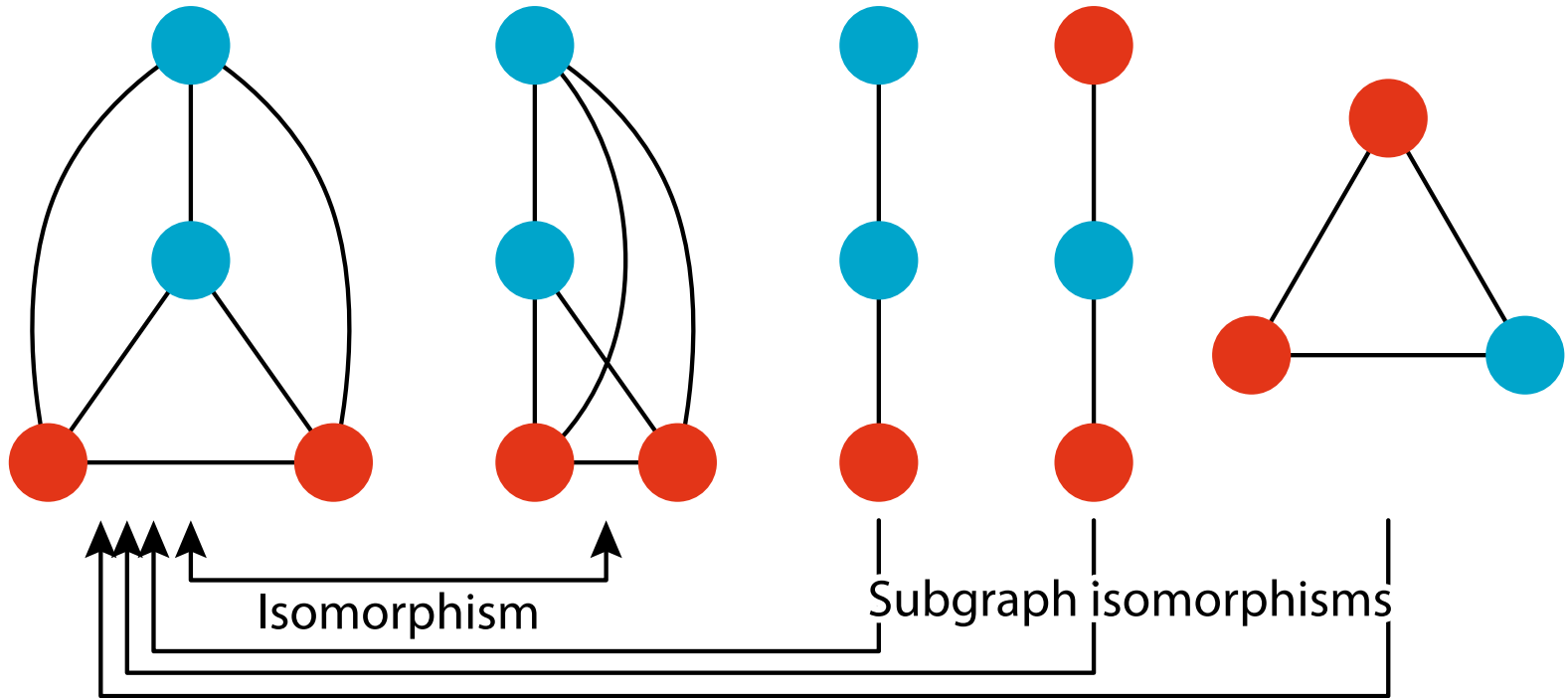
Subgraph Isomorphism

- A graph $G' = (V', E')$ is a **subgraph** of $G = (V, E)$, denoted by $G' \sqsubseteq G$, if $V' \subseteq V$ and $E' \subseteq (V' \times V') \cap E$
- A graph G' is **isomorphic** to G if there exists a **bijective function** $\pi : V' \rightarrow V$ such that
 - (i) $(u, v) \in E' \iff (\pi(u), \pi(v)) \in E$
 - (ii) $\forall v \in V', \varphi(v) = \varphi(\pi(v))$
 - (iii) $\forall (u, v) \in E', \varphi(u, v) = \varphi(\pi(u), \pi(v))$
- If π is injective but not surjective: $G \setminus \text{range}(\pi) \neq \emptyset$, G' is **subgraph isomorphic** to G , denoted by $G' \sqsubseteq G$
 - Testing whether $G' \sqsubseteq G$ is **NP-complete** (computationally heavy!) 4/17

Subgraph Isomorphism



Subgraph Isomorphism



Subgraph Mining

- In graph mining, **pattern** \iff (sub)graph
- S : the set of graphs (can be infinite), a dataset D is a multiset of S
 - D is a collection of graphs: $D = \{G_1, G_2, \dots, G_n\}$

- The **frequency** $\eta(G)$ of a graph G is obtained as

$$\eta(G) = \frac{|\{G_i \in D \mid G \sqsubseteq G_i\}|}{|D|} = \frac{1}{|D|} \sum_{H \ni G} \mathbf{1}_D(H)$$

- **Frequent subgraph mining problem:**

Given a threshold σ , enumerate the set $F = \{G \in S \mid \eta(G) \geq \sigma\}$

Two Problems in Graph Mining

1. Combinatorial explosion of the search space

- More massive than itemset mining
- The number of subgraphs with m vertices: $O(2^{m^2})$
 - $O(m^2)$ possible edges
- The number of subgraphs with m vertices and s labels: $O(s^{m^2})$

2. Subgraph isomorphism checking

- When we obtain a subgraph G' , computing $\eta(G')$ is heavy as we need to repeat subgraph isomorphism checking for every $G_i \in D$
- **Solution:** Use the Apriori principle and the (canonical) DFS code

Graph Mining Algorithms

- The first algorithm that achieves graph mining is **AGM**
 - Inokuchi, A. and Washio, T. and Motoda, H., **An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data**, PKDD 2000
- The standard method is **gSpan**
 - Yan, X. and Han, J., **gSpan: Graph-based substructure pattern mining**, ICDM 2002
- The state-of-the-art is **GASTON**
 - Nijssen, S. and Kok, J. N., **A Quickstart in Frequent Structure Mining Can Make a Difference**, SIGKDD 2004

DFS Code (1/3)

- The **DFS code** represents a graph G as a sequence of tuples based on depth first search (DFS)
 - There can be multiple DFS codes for a single graph
- Perform DFS traversal on a graph G and index each vertex according to the order of discovery in the DFS
 - Edges included in the DFS are **forward edges**, other edges are **backward edges**
- Each edge (i, j) is represented as a tuple $(i, j, \varphi(i), \varphi(j), \varphi(i, j))$
 - $i < j$ if it is a forward edge and $i > j$ if backward

DFS Code (2/3)

- Introduce the (total) order " $<_t$ " between two tuples $t_1 = (i_1, j_1, \varphi(i_1), \varphi(j_1), \varphi(i_1, j_1))$ and $t_2 = (i_2, j_2, \varphi(i_2), \varphi(j_2), \varphi(i_2, j_2))$
- First, introduce the order $<_e$ between $e_1 = (i_1, j_1)$ and $e_2 = (i_2, j_2)$:
 $e_1 <_e e_2 \iff$
 - If both e_1 and e_2 are forward edges, (a) $j_1 < j_2$ or (b) $j_1 = j_2$ and $i_1 > i_2$
 - If both e_1 and e_2 are backward edges, (a) $i_1 < i_2$ or (b) $i_1 = i_2$ and $j_1 < j_2$
 - If e_1 and e_2 are forward and backward edges, $j_1 \leq i_2$
 - If e_1 and e_2 are backward and forward edges, $i_1 < j_2$
- Introduce some total order $<_l$ into triples of labels $(\varphi(i_1), \varphi(j_1), \varphi(i_1, j_1))$

DFS Code (3/3)

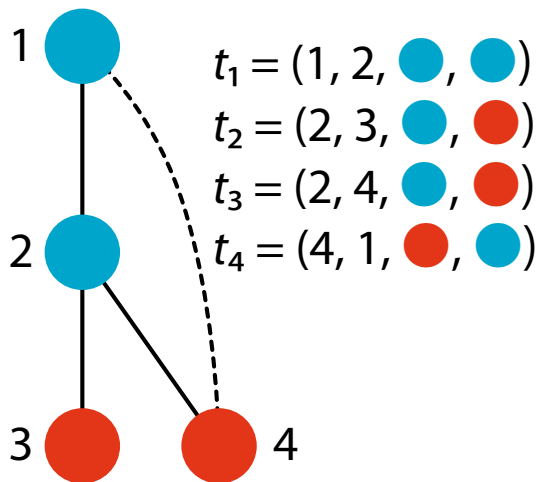
- $t_1 = (i_1, j_1, \varphi(i_1), \varphi(j_1), \varphi(i_1, j_1)) <_t t_2 = (i_2, j_2, \varphi(i_2), \varphi(j_2), \varphi(i_2, j_2)) \iff$
 - (i) $(i_1, j_1) <_e (i_2, j_2)$, or
 - (ii) $(i_1, j_1) = (i_2, j_2)$ and $(\varphi(i_1), \varphi(j_1), \varphi(i_1, j_1)) <_l (\varphi(i_2), \varphi(j_2), \varphi(i_2, j_2))$
- The DFS code of a graph is a sequence of tuples sorted according to the order " $<$ "

Canonical DFS Code

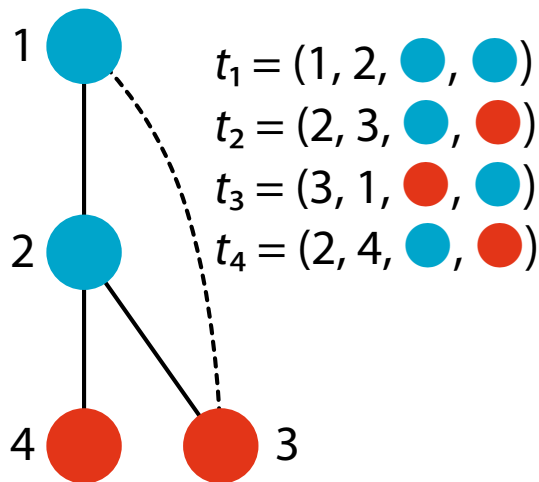
- Finally, introduce the order $<$ between two DFS codes $\mathbf{t} = (t_1, t_2, \dots, t_m)$ and $\mathbf{t}' = (t'_1, t'_2, \dots, t'_n)$
- $\mathbf{t} < \mathbf{s} \iff$ (i) or (ii)
 - (i) $\exists k$ s.t. $0 \leq k \leq \min(m, n)$, $t_1 = t'_1, t_2 = t'_2, \dots, t_{k-1} = t'_{k-1}, t_k < t'_k$
 - (ii) $m \leq n$ and $t_1 = t'_1, t_2 = t'_2, \dots, t_m = t'_m$
- The **canonical DFS code** of a graph G is the smallest DFS code of G according to the order " $<$ "

Canonical DFS Code

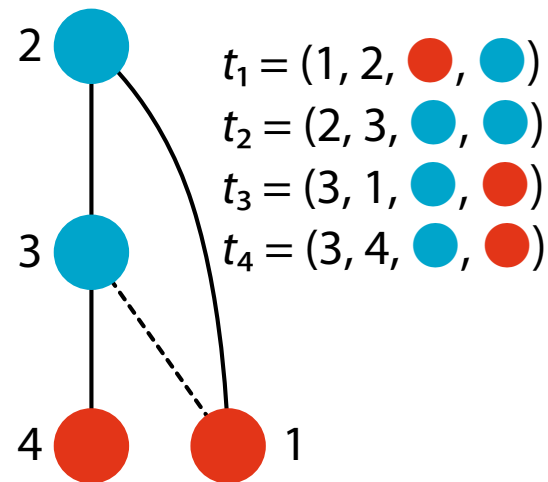
Noncanonical



Canonical



Noncanonical



Rightmost Path Extension

- During the DFS traversal on a graph G , the **rightmost path** is the path from the root to the rightmost leaf (leaf with the largest index)
- **Rightmost path extension** achieves systematic candidate graph generation from an existing graph G by either
 - (i) adding a backward edge from the **rightmost vertex** to other vertex on the rightmost path, or
 - (ii) adding a forward edge from a vertex on the rightmost path

The gSpan Algorithm

Algorithm 1: Algorithm gSpan

// $C \leftarrow \emptyset$ for the initial call

```
1 gSpan( $C, D, \sigma$ )
2    $\mathcal{E} \leftarrow \text{RightmostPathExtension}(C, D)$ 
3   foreach  $(t, \eta_t) \in \mathcal{E}$  do
4      $C \leftarrow C \cup \{t\}$ 
5      $\eta(C) \leftarrow \eta_t$ 
6     if  $\eta(C) > \sigma$  and  $\text{isCanonical}(C)$  then
7        $\text{gSpan}(C, D, \sigma)$ 
```

Subprocesses in gSpan

- $\text{RightmostPathExtension}(C, D)$
 - Receive a graph G represented by its DFS code C and a dataset D
 - Return all possible rightmost path extensions of G
 - A set of pairs of tuples and frequencies
$$\mathcal{E} = \{(t_1, \eta_{t_1}), (t_2, \eta_{t_2}), \dots, (t_m, \eta_{t_m})\}$$
- $\text{isCanonical}(C)$
 - Receive a DFS code C
 - Return TRUE if C is canonical and FALSE otherwise

Conclusion

- gSpan achieves graph mining
- The keys are:
 - Canonical DFS codes
 - Rightmost path extension
 - Combine them with the Apriori principle