Mining data graph

FREQUENT GRAPH PATTERN MINING

Lecturer: Le Ngoc Thanh

Email: lnthanh@fit.hcmus.edu.vn

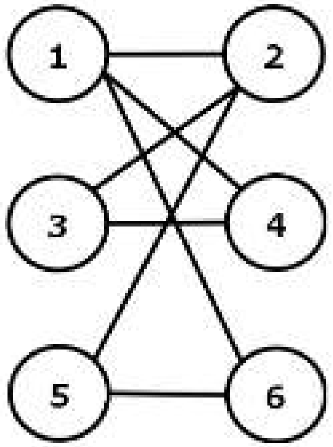
fit@hcmus

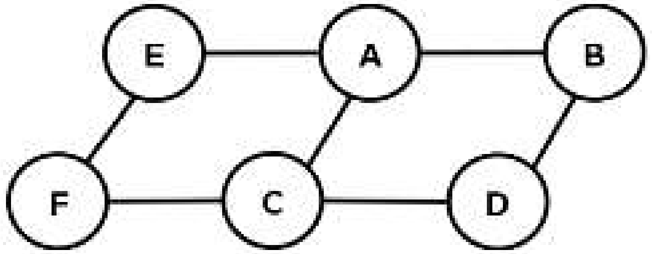
Content

- **Frequent subgraph pattern**
- Common pattern finding method based on Apriori
- Depth-based common pattern finding method
- Greedy frequent pattern finding method

Subgraph and graph isomorphism

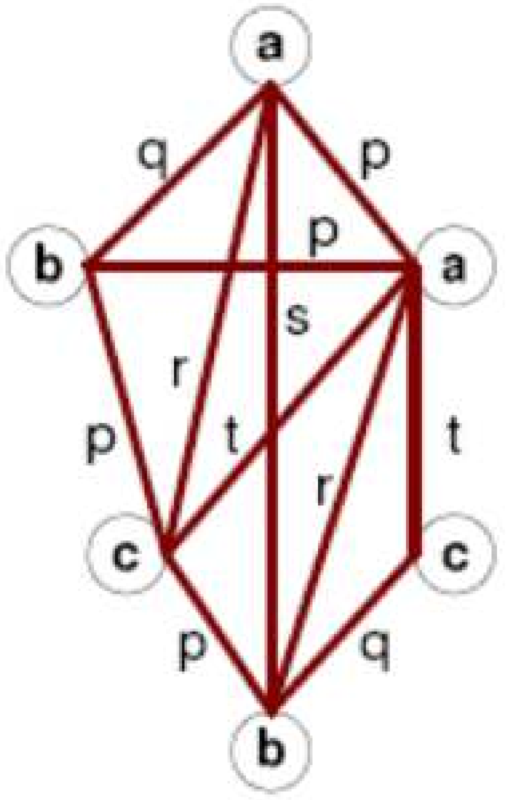
- A graph $G_S(V_S, E_S)$ is a subgraph of the graph G(V, E) if:
  - V_S is the vertex set of V and E_S is a subset of E
- Two graphs $G_1(V_1, E_1)$ and $G_2(V_2, E_2)$ will be isomorphic if they are the same cartouche
  - That is, there is a way of mapping from V_1 to V_2 such that each edge in E_1 corresponds to an edge in E_2 and vice versa.



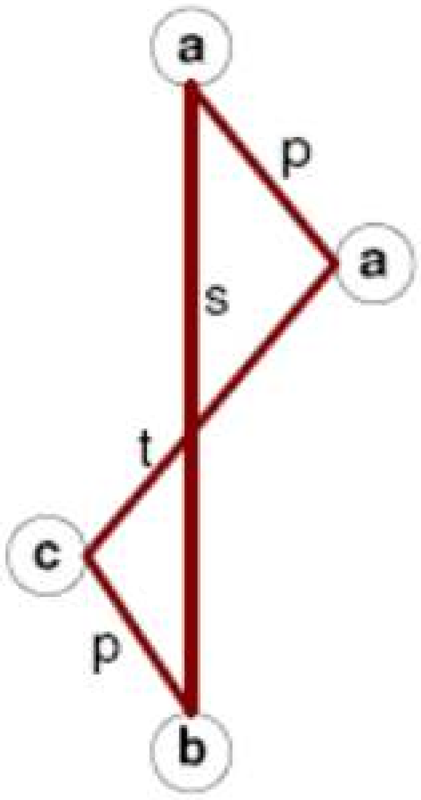


$f(1) = A \quad f(2) = C \quad f(3) = D \quad f(4) = B \quad f(5) = F \quad f(6) = E$

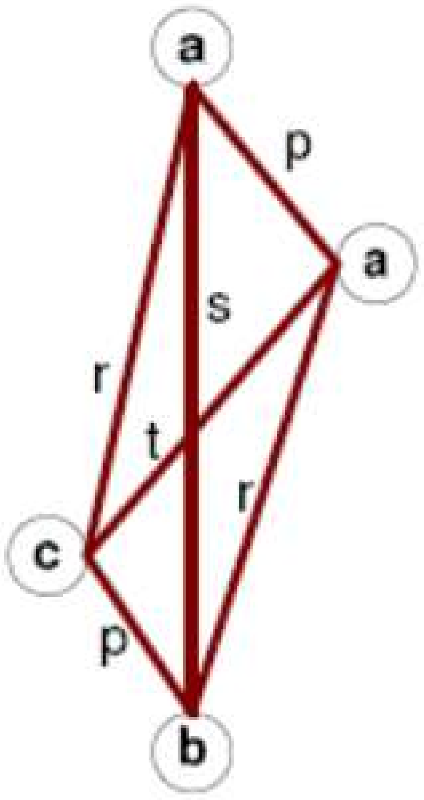
Subgraph example



(a) Labeled Graph



(b) Subgraph



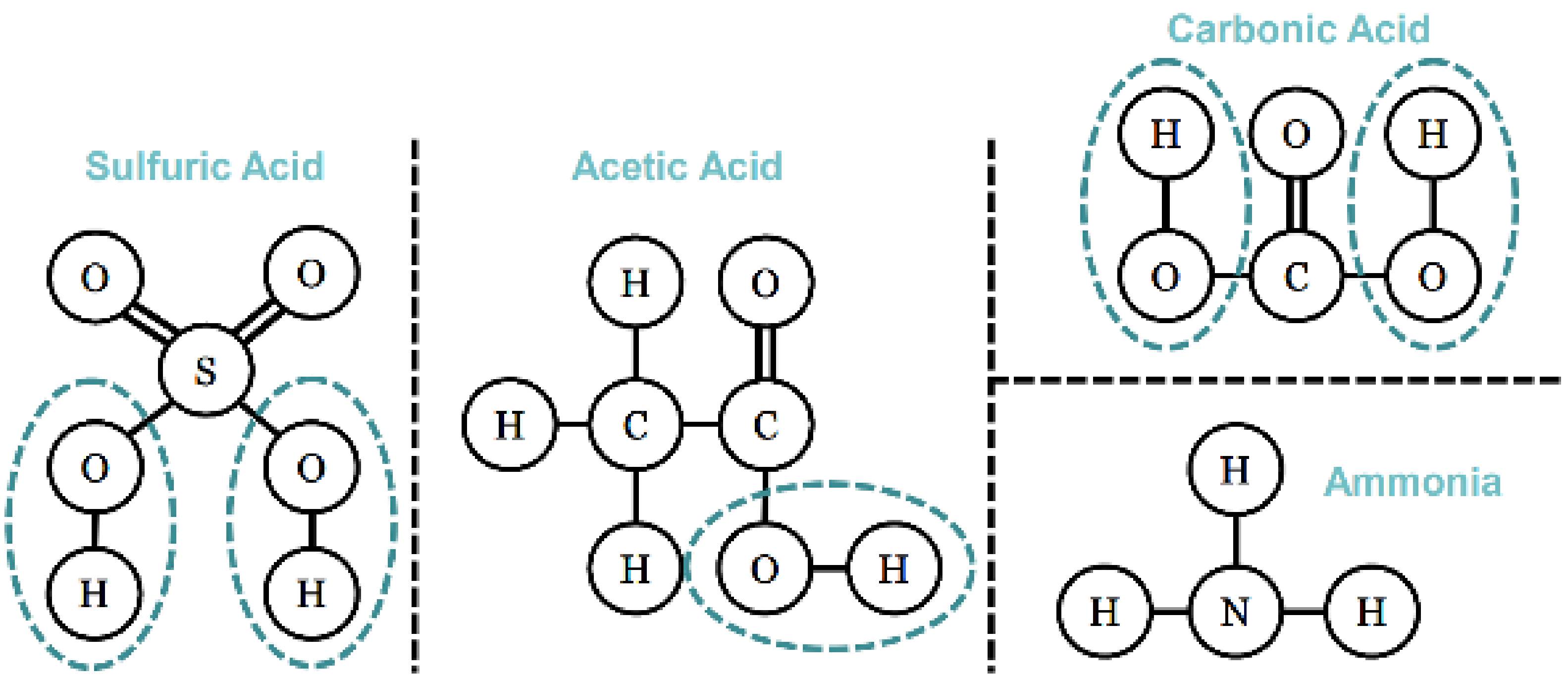
(c) **Subgraph**

Frequent subgraph pattern

- Frequent subgraph pattern: a subgraph structure that occurs frequently in a given set of graphs.

$$\text{sup(subgraph)} \geq minsup$$

  – For example, find connections that frequently appear in acid types



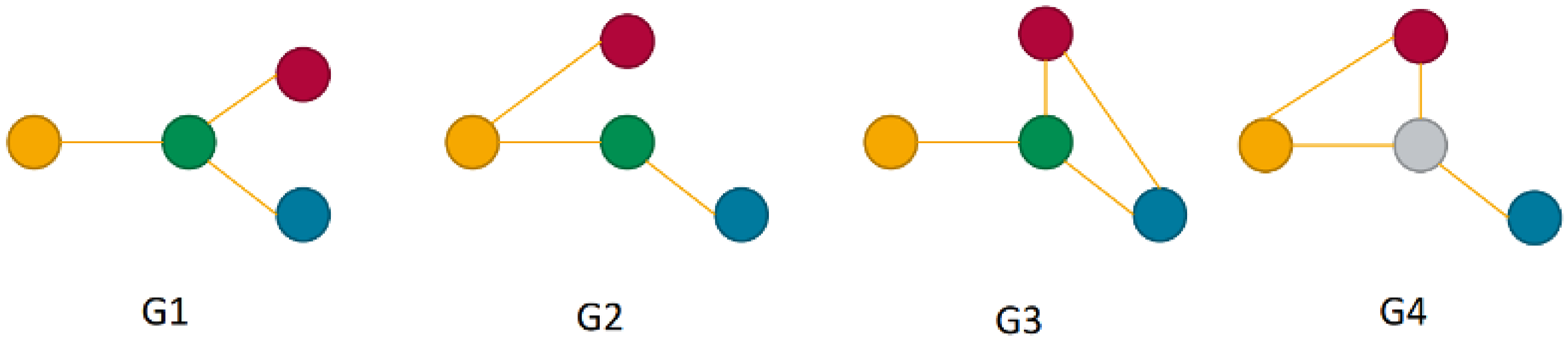
O-H xảy ra 3 trong số 4 dữ liệu → phổ biến nếu độ trợ <= 3

Frequent graph pattern

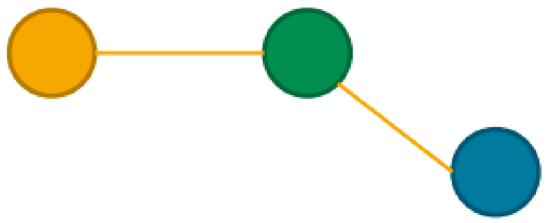
- Frequent graph pattern mining applications:
  - Find common biological pathways between species.
  - Community analysis in social networks
  - Build blocks for graph layering, grouping, compression, comparison or correlation analysis.

Graph sample mining example

- Define subgraphs of G that appear at least minsup =3 times



Đồ thị con phổ biến với minsup=3:



Frequent graph pattern

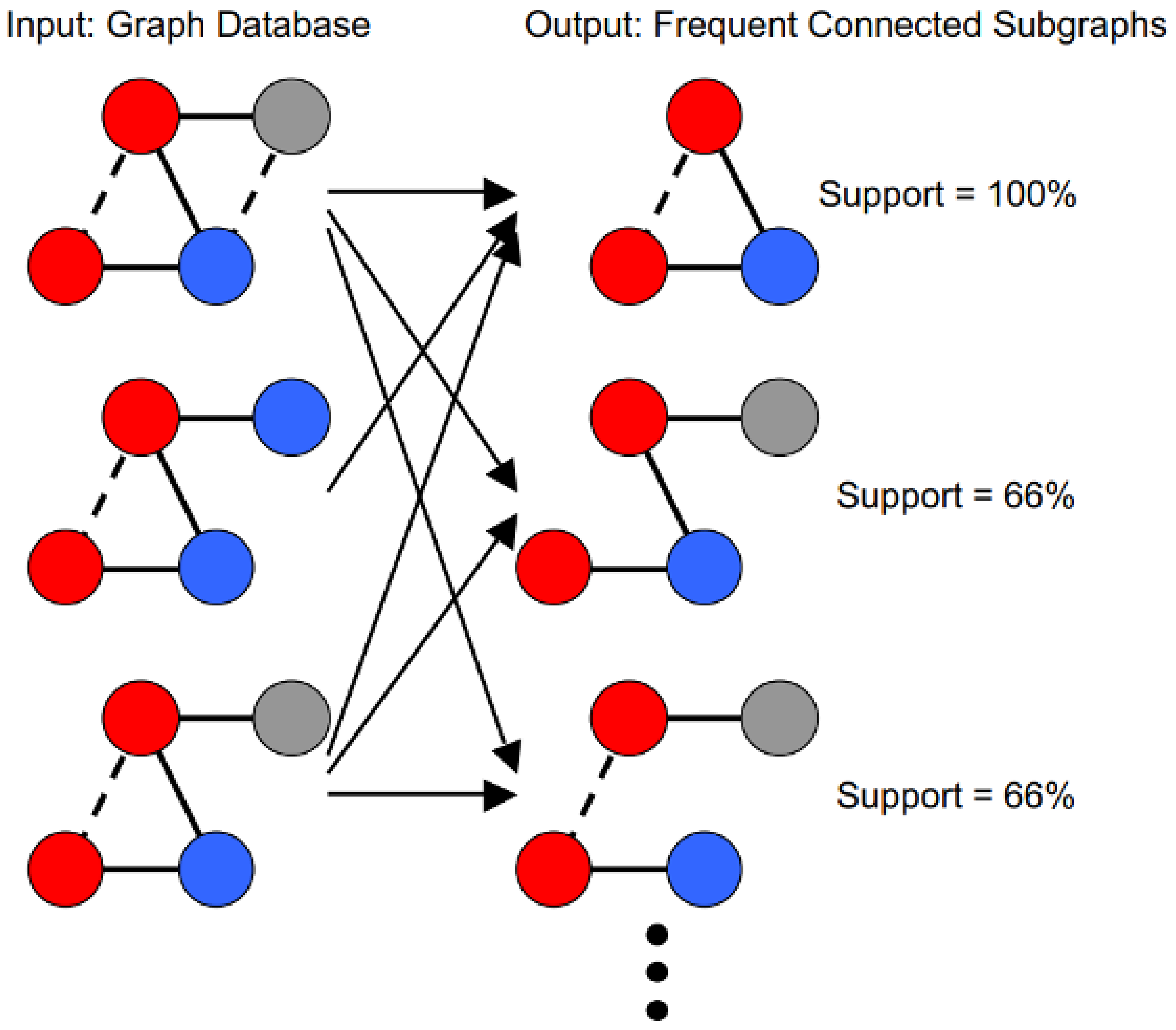
- When considering a graph pattern, we do not consider the number of times this pattern occurs repeatedly in the graph (like in frequent pattern mining, in which each transaction items are counted only once).
- However, another problem is finding a pattern that occurs frequently in a graph.

→ In the content of this section, we do not consider the above common pattern.

Statement of the problem

- Give a set of graphs labeled D={G_1,G_2, ...,G_n} and a subgraph G:
  - G's support set is $D_G = \{G_i | G \sqsubseteq G_i, G_i \in D\}$ where the symbol $G \sqsubseteq G_i$ implies G is a submorphism of G_i
  - Assists are calculated $\sigma(G) = \frac{|D_G|}{|D|}$
- Input:
  - Set of graphs D
  - Minsup value
- Output:
  - Common interconnected subgraphs

Statement of the problem



Challenges

- Finding common graph patterns often involves the identification of isomorphic subgraphs.
  - That is, define a graph that contains a subgraph that is isomorphic to another graph.
- This is classified as an NP-Complete problem
  - Requires running with exponential time
- The algorithm that exploits common subgraphs must effectively reduce the search space by reducing the number of subgraph isomorphism checks.

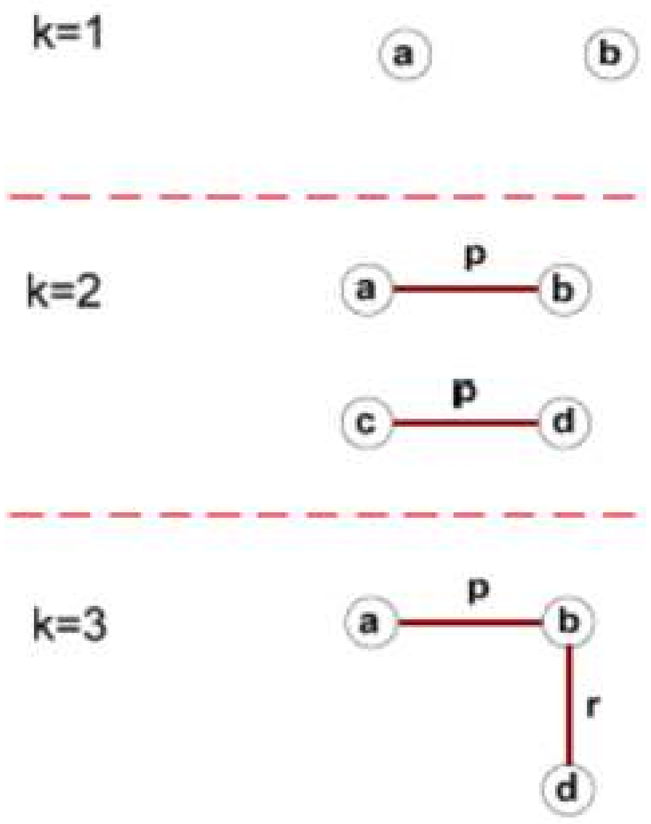
Mining algorithms

- Popular subgraph mining algorithms are divided into several groups:
  - Methods based on graph theory:
    - Width-based method or Apriori: AGM/AcGM, FSG, gFSG, PATH#, FFSM
    - Depth-based or pattern-growth methods: MoFa, gSpan, Gaston
    -
  - Greedy method: Subdue
  - Methods based on logical deduction: WARMR

Nội dung

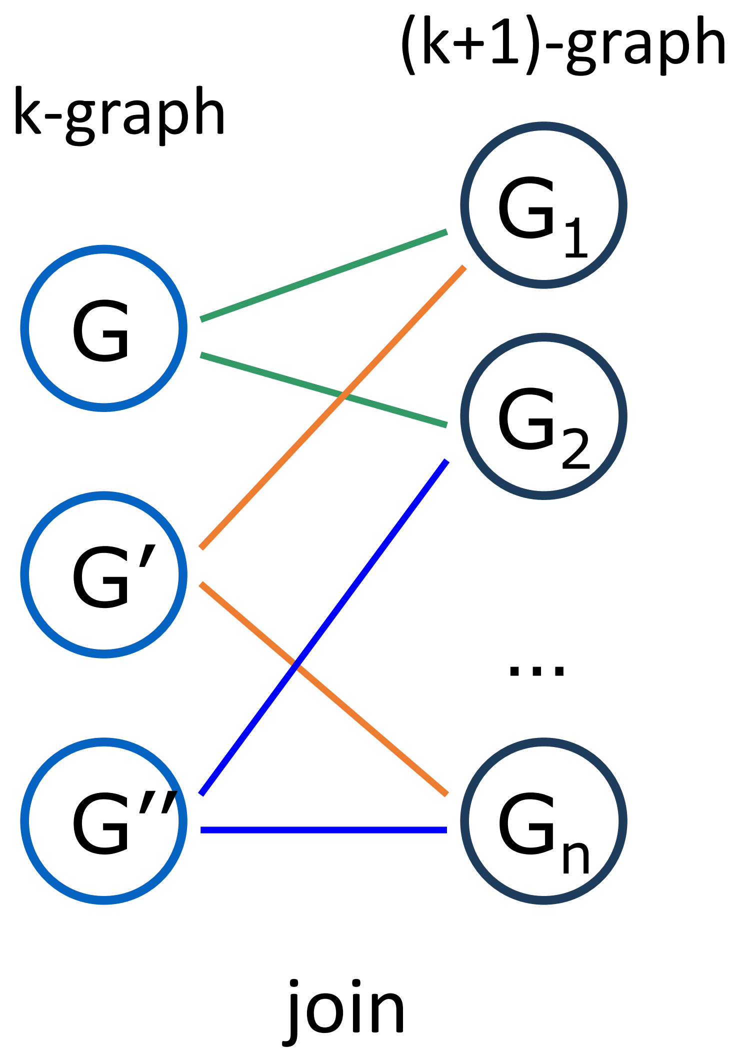
Methods based on Apriori

- Concept:
  - A k-subgraph is a subgraph with k vertices or k edges



Methods based on Apriori

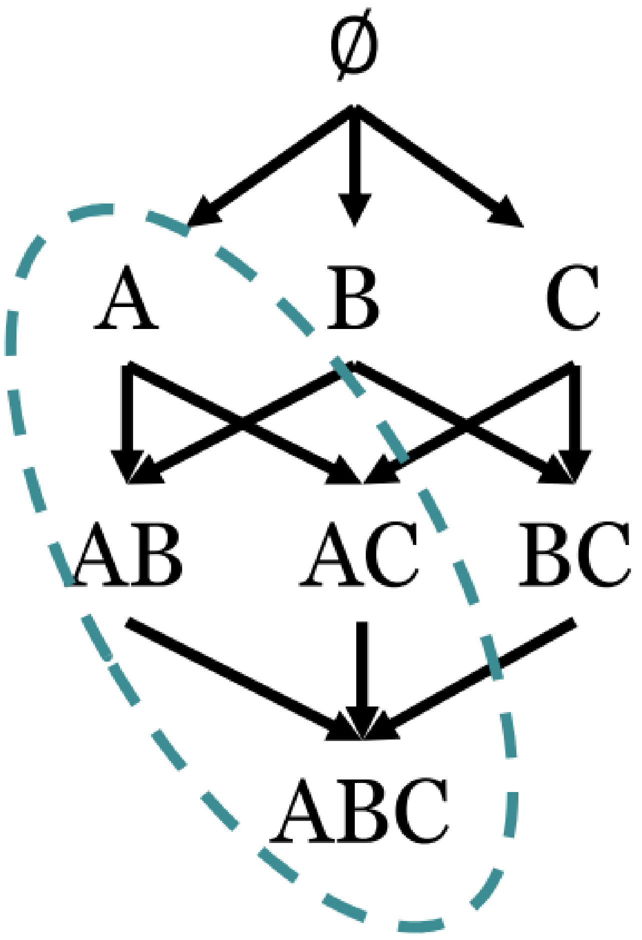
- The idea of the Apriori-based method is use common k-subgraphs to generate (k+1)-common subgraphs.



Methods based on Apriori

- Downward closure property in Apriori:

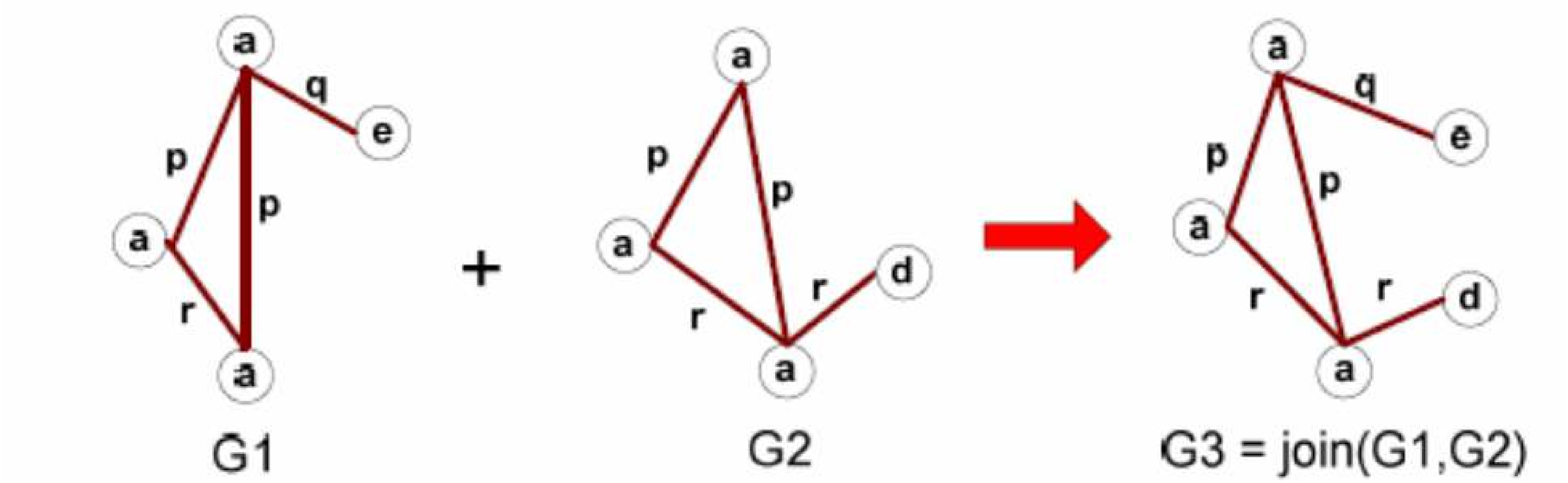
If A is not common, then its parent sets are also uncommon



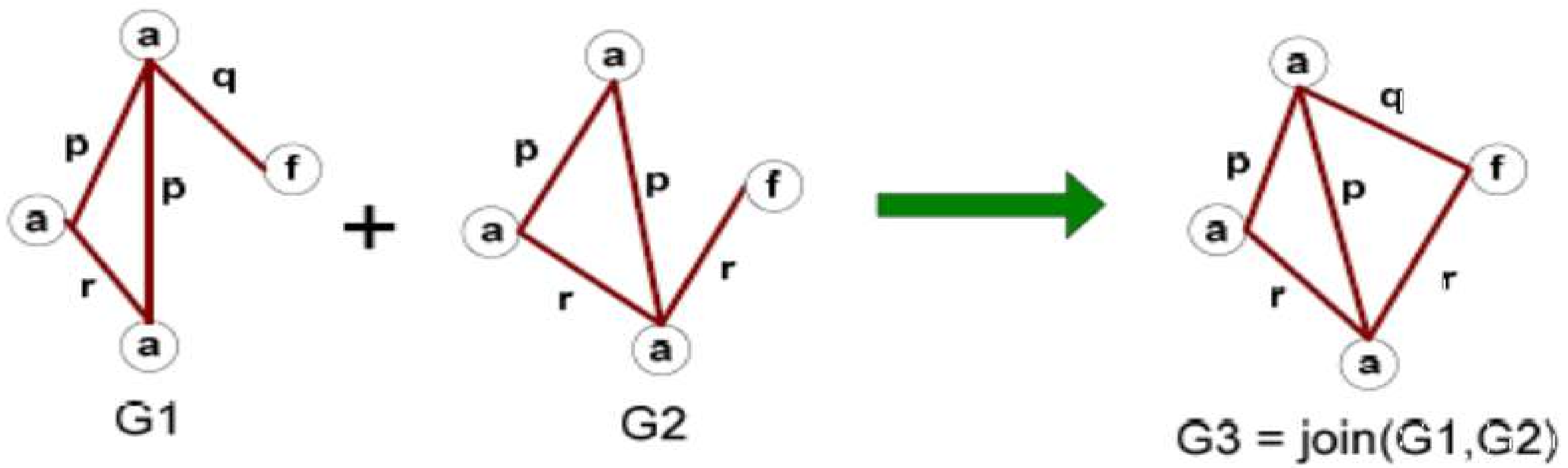
Methods based on Apriori

- There are two branches within the Apriori-based methodology:
  - Vertex growing: AGM
    - The index k will be the peak number
  - Edge growing: FGM
    - The index k will be the number of edges

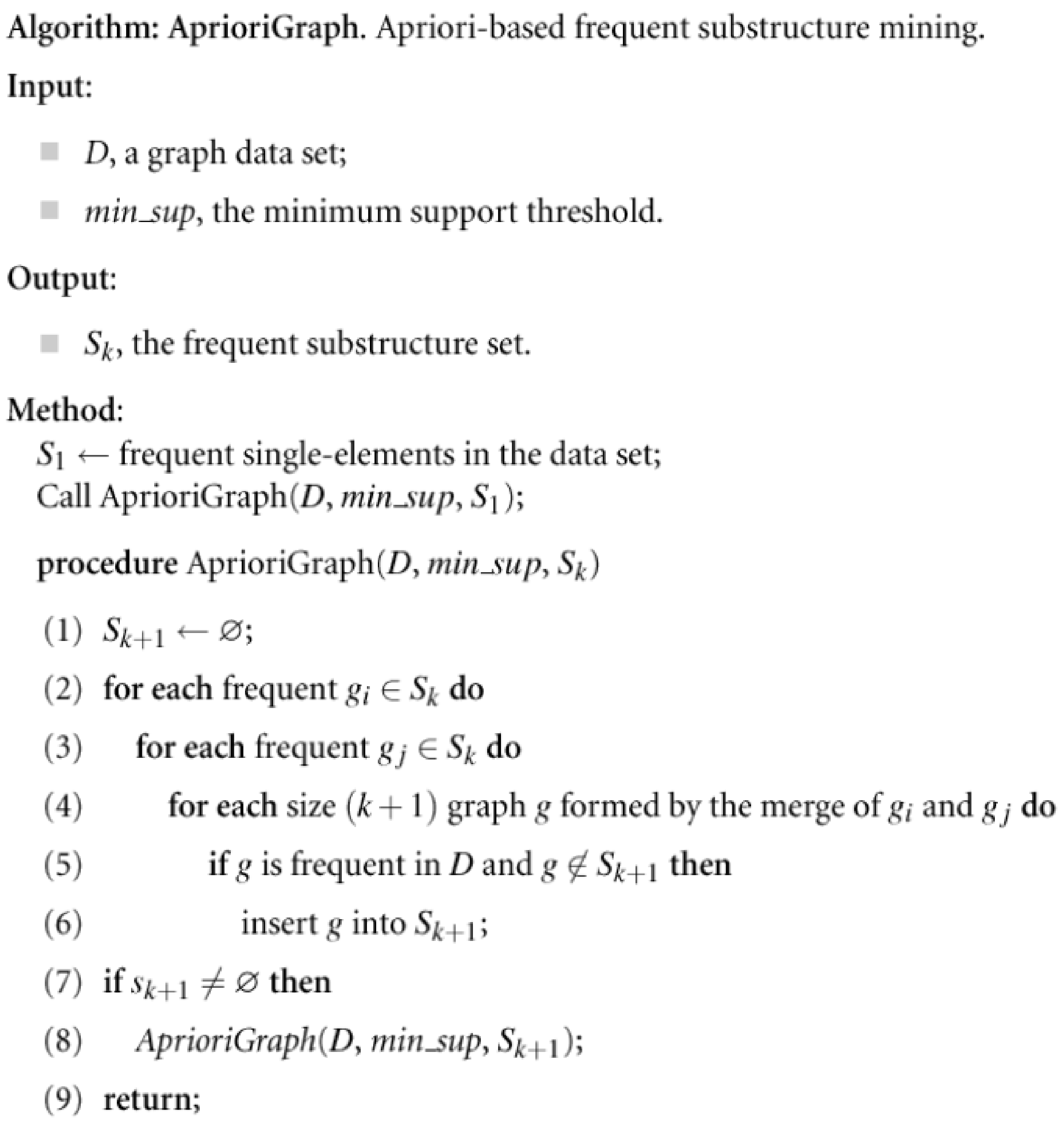
Peak development branch



Edge development branch



AGM (Apriori-based Graph Mining)



Algorithm: AprioriGraph. Apriori-based frequent substructure mining.

Input:

- *D*, a graph data set;
- *min_sup*, the minimum support threshold.

Output:

- S_k, the frequent substructure set.

Method:

```
$S_1 \leftarrow$ frequent single-elements in the data set;
Call AprioriGraph($D$, $min\_sup$, $S_1$);

procedure AprioriGraph($D$, $min\_sup$, $S_k$)

(1) $S_{k+1} \leftarrow \varnothing$;
(2) for each frequent $g_i \in S_k$ do
(3)     for each frequent $g_j \in S_k$ do
(4)         for each size $(k+1)$ graph $g$ formed by the merge of $g_i$ and $g_j$ do
(5)             if $g$ is frequent in $D$ and $g \notin S_{k+1}$ then
(6)                 insert $g$ into $S_{k+1}$;
(7) if $s_{k+1} \neq \varnothing$ then
(8)     $AprioriGraph(D, min\_sup, S_{k+1})$;
(9) return;
```

AGM (Apriori-based Graph Mining)

- AGM algorithm:

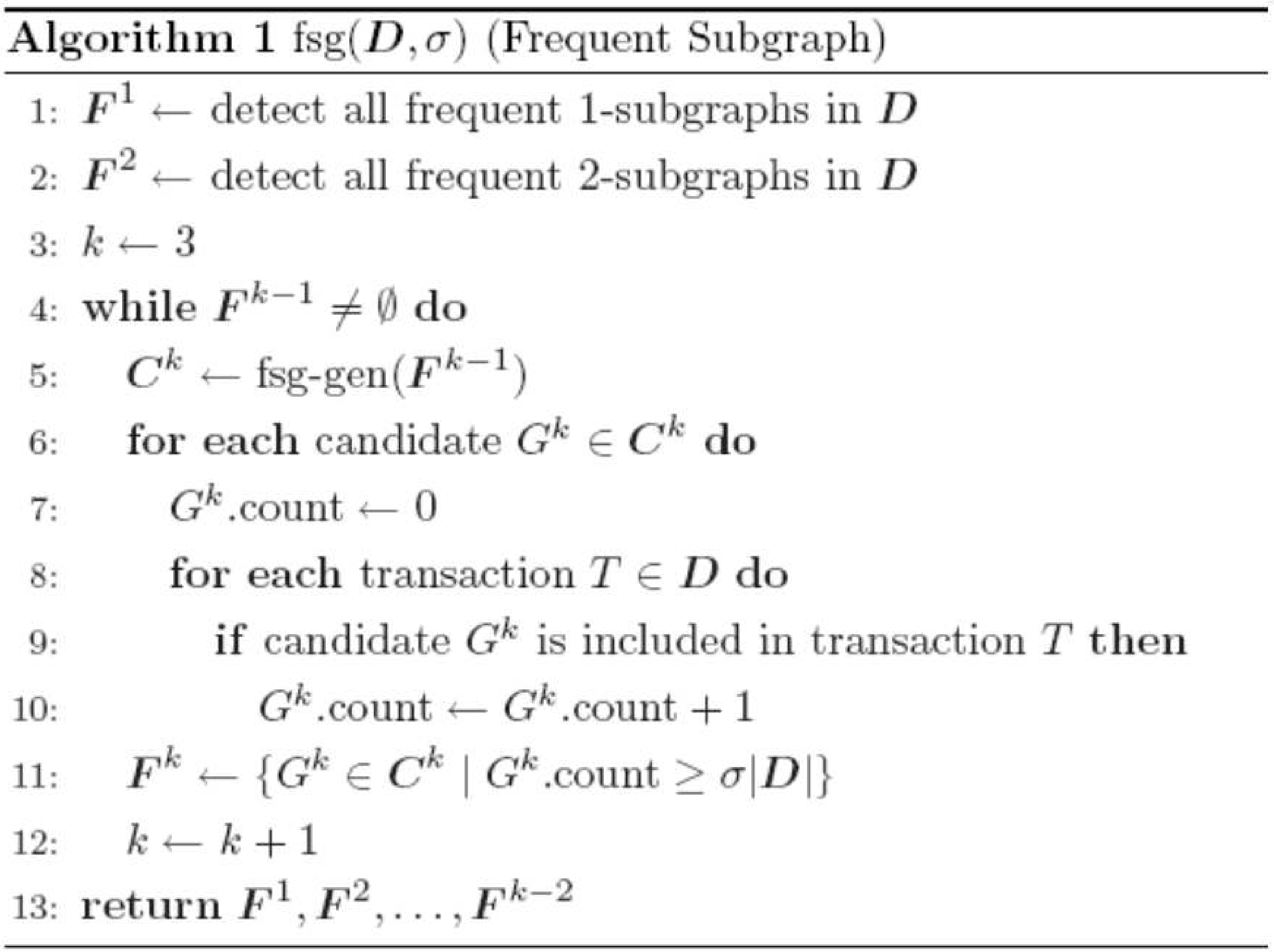
Notation: k-subgraph is a subgraph with k vertices

Initialization: scan the data to find F1, the set of all common 1-subgraphs, along with their support;

For (k=3; $F_{k-1} \neq \emptyset$; k++)

1. **Phát sinh ứng viên** - *Ck, candidate set k-subgraphs, from Fk-1; The candidate will increase by 1 peak compared to the previous graph.*
2. **Tỉa nhánh** – The necessary condition for the candidate to become popular is that each (k-1)-subgraph of it must be popular.
3. **Tính độ phổ biến** – scan the data to count the number of times the Ck subgraph appears.
4. $F_k = \{ c \in C_K \mid c$ has a support not less than *minSup* $\}$
5. Return $F_1 \cup F_2 \cup \ldots\ldots \cup F_k$ (= F)

FGM (Frequent Sub-graph Mining)



Algorithm 1 fsg(D, σ) (Frequent Subgraph)

```
1: $F^1 \leftarrow$ detect all frequent 1-subgraphs in $D$
2: $F^2 \leftarrow$ detect all frequent 2-subgraphs in $D$
3: $k \leftarrow 3$
4: while $F^{k-1} \neq \emptyset$ do
5:     $C^k \leftarrow$ fsg-gen($F^{k-1}$)
6:     for each candidate $G^k \in C^k$ do
7:         $G^k$.count $\leftarrow 0$
8:         for each transaction $T \in D$ do
9:             if candidate $G^k$ is included in transaction $T$ then
10:                $G^k$.count $\leftarrow G^k$.count $+ 1$
11:    $F^k \leftarrow \{G^k \in C^k \mid G^k.\text{count} \geq \sigma|D|\}$
12:    $k \leftarrow k + 1$
13: return $F^1, F^2, \ldots, F^{k-2}$
```

FGM (Frequent Sub-graph Mining)

- Similar to AGM but interested in the edge

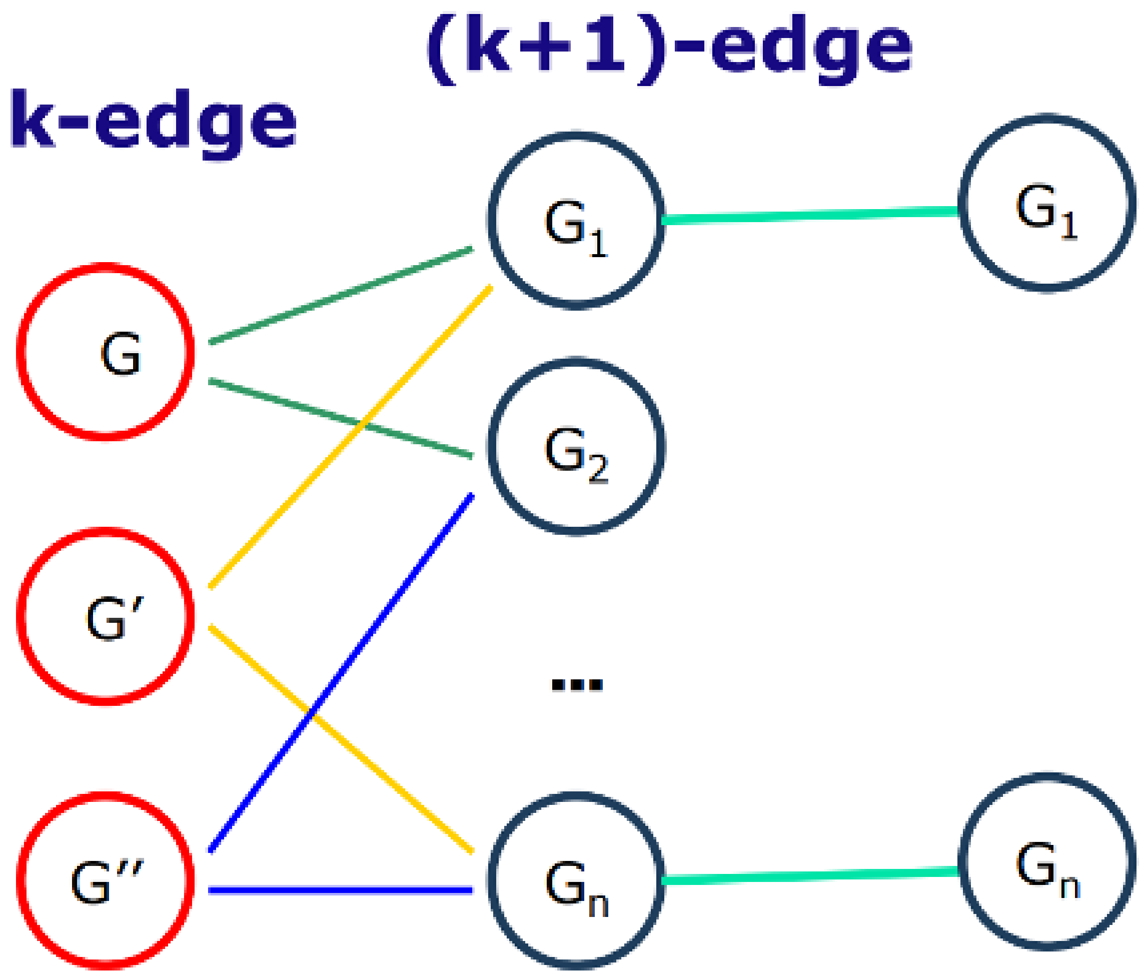
Notation: k-subgraph is a subgraph with k edges

Initialization: Scan the data to find F1, the set of all common 1-subgraphs, along with their support;

For (k=3; $F_{k-1} \neq \emptyset$; k++)

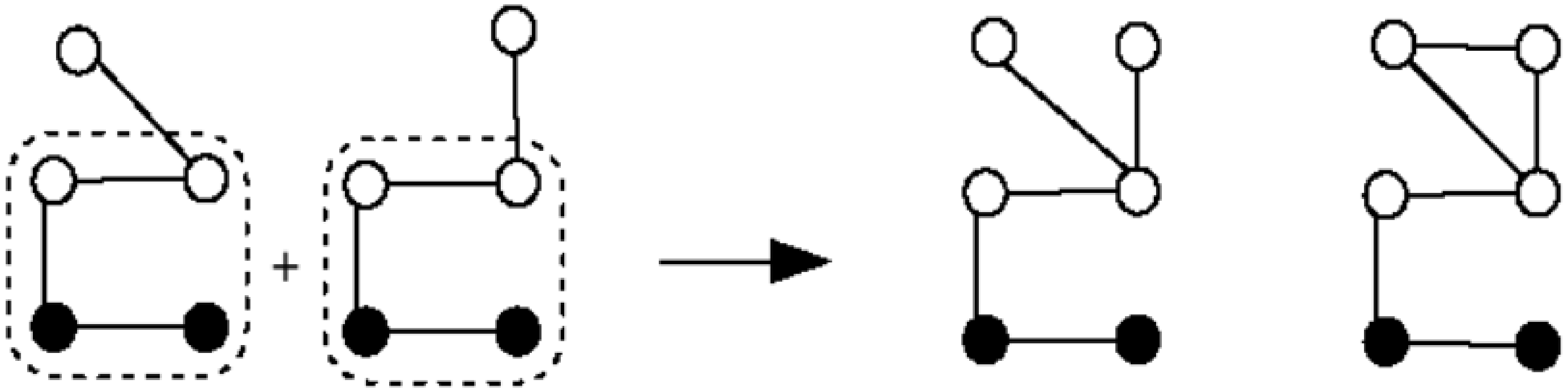
1. **Candidate generation** - C_k, Candidate set K-Subgraphs, from F_{k-1}; candidates will increase by 1 edge compared to the previous graph.
2. **Pruning branches** – The necessary condition for the candidate to become popular is that each (k-1)-subgraph of it must be popular.
3. **Calculate popularity**– scan data to count the number of times the Ck subgraph appears.
4. $F_k = \{ c \in C_K \mid c$ has a support not less than *#minSup* }
5. Return $F_1 \cup F_2 \cup \ldots\ldots \cup F_k$ (= F)

AGM and FGM



AGM and FGM

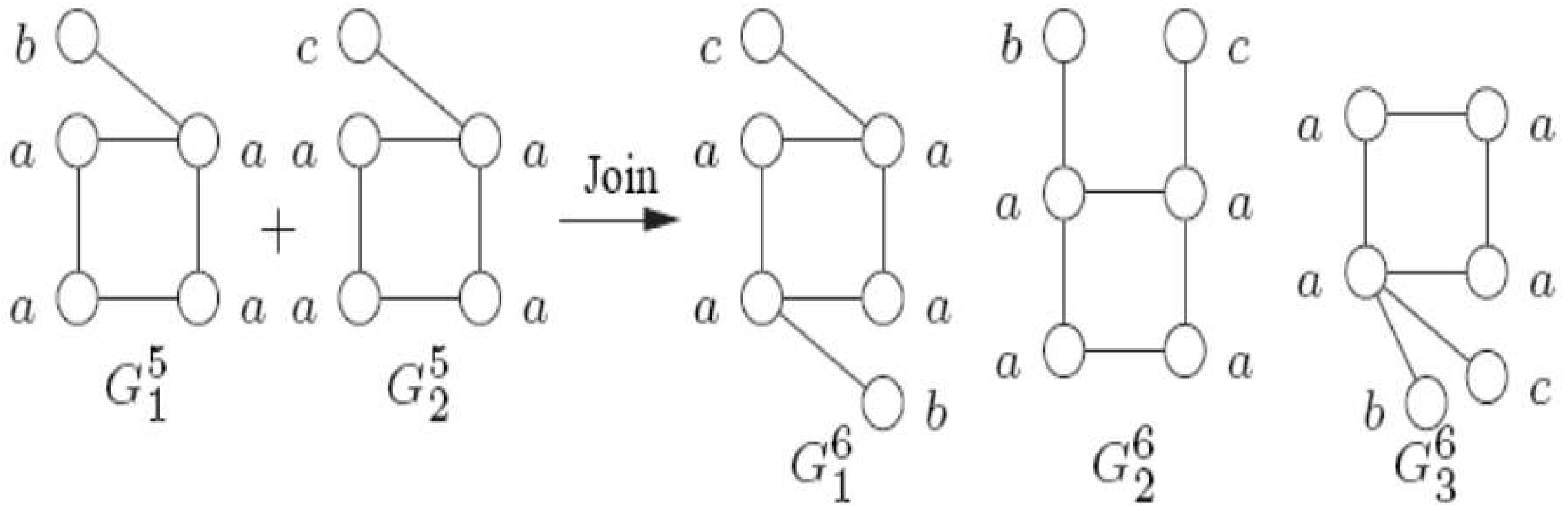
- Candidate generation function:
  - Two common graphs of size k are joined only if they have the same subgraph of size k-1.
  - The difference compared to Apriori when working on graphs is that joining two graphs can generate more than two candidates.



Graphs joined in AGMs

AGM and FGM

- Candidate generation function:
  - Two common graphs of size k are joined only if they have the same size subgraph k-1 (core graph).
  - The difference from Apriori when doing on graphs is that joining two graphs can create more than two candidates.

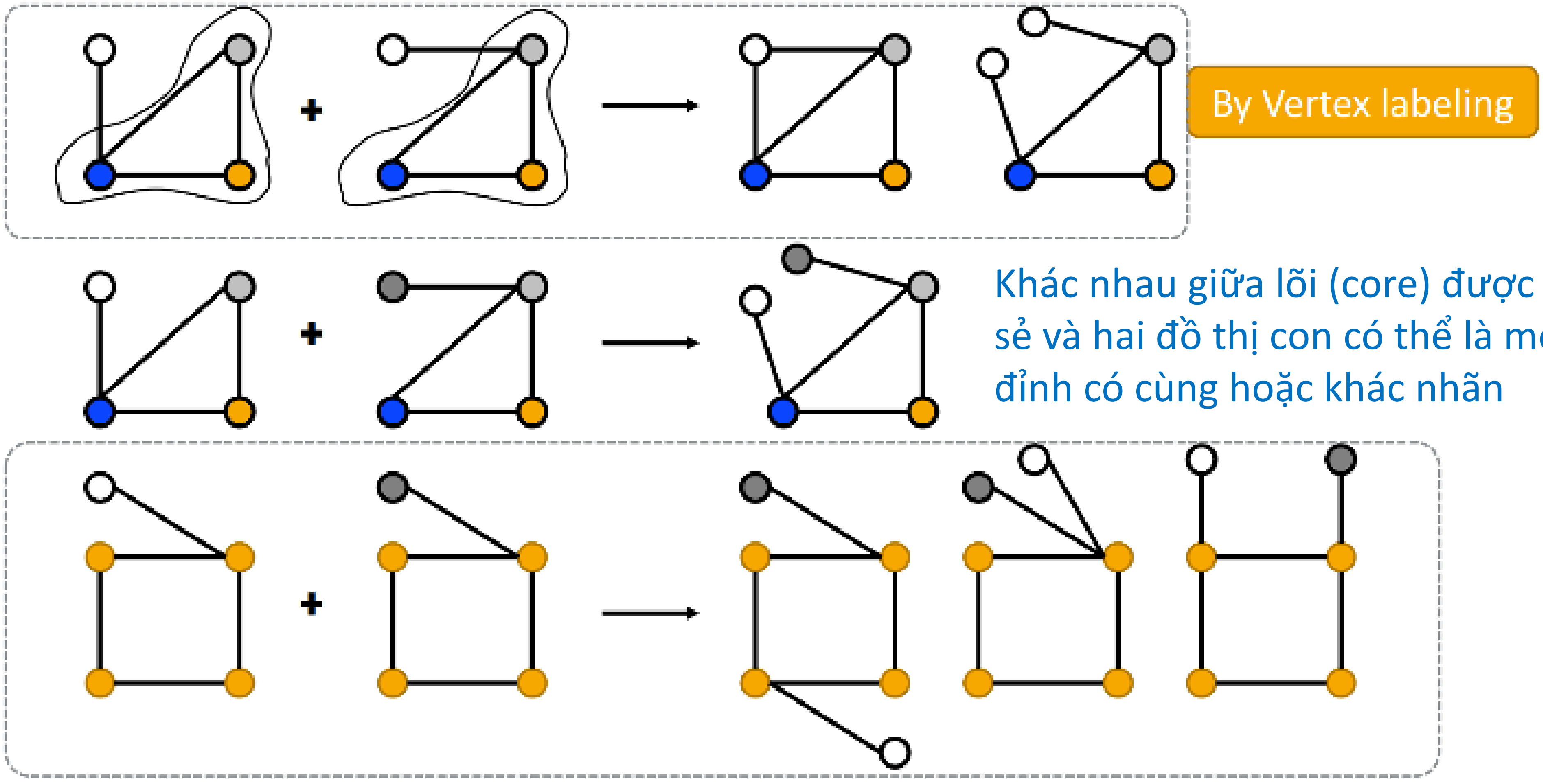


Đồ thị được gia nhập trong FGM

Core determination

- Core between two graphs G_i^k and G_j^k is determined by:
  - Create a (k-1)-subgraph of a graph G_i^k by removing an edge
  - Check if this subgraph is a subgraph of G_j^k.
  - Repeat the process above to have other cores (if any)

Candidate generation based on core detection

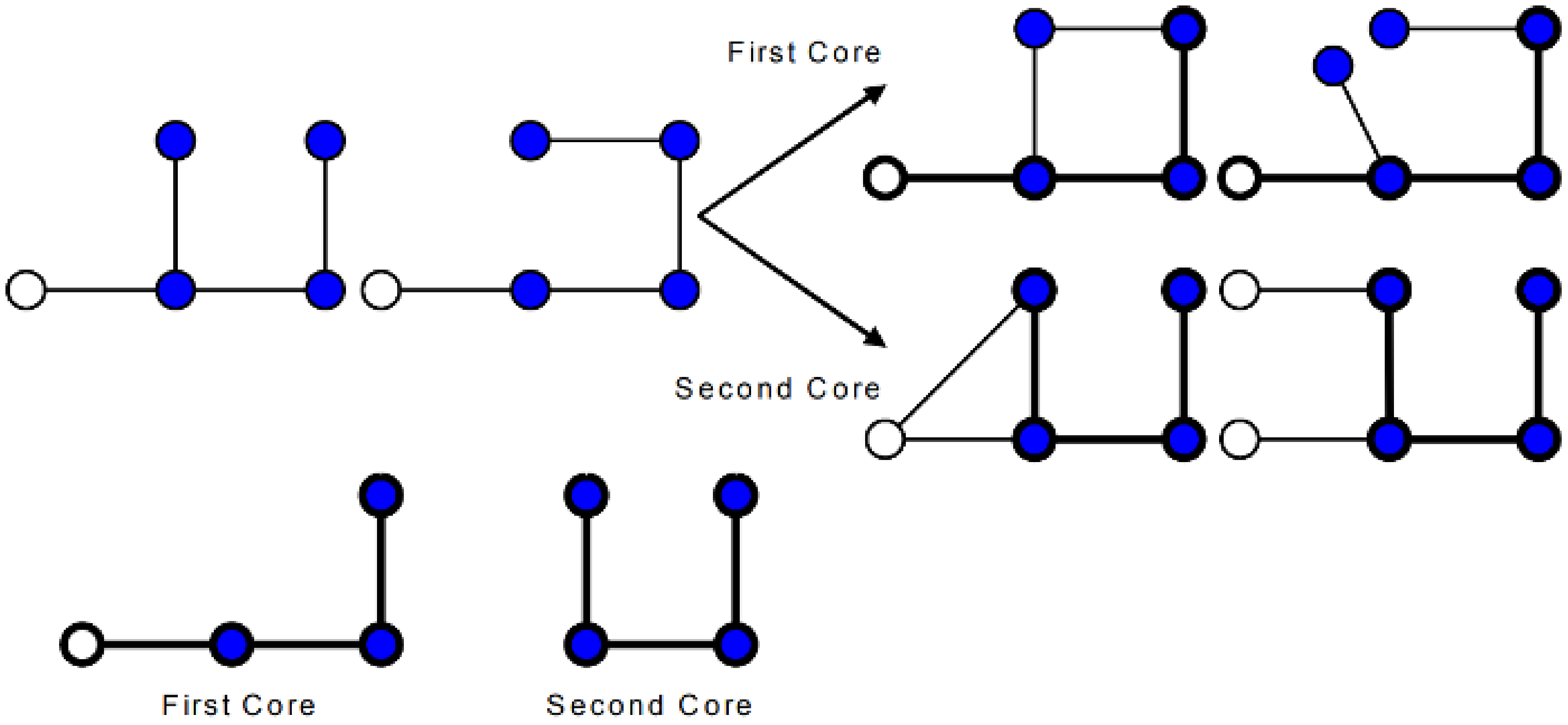


Khác nhau giữa lõi (core) được chia sẻ và hai đồ thị con có thể là một đỉnh có cùng hoặc khác nhãn

Bản thân lõi cũng có nhiều dạng tự đẳng cấu (automorphism).
Mỗi lõi có thể tạo ra các ứng viên k+1 khác nhau.

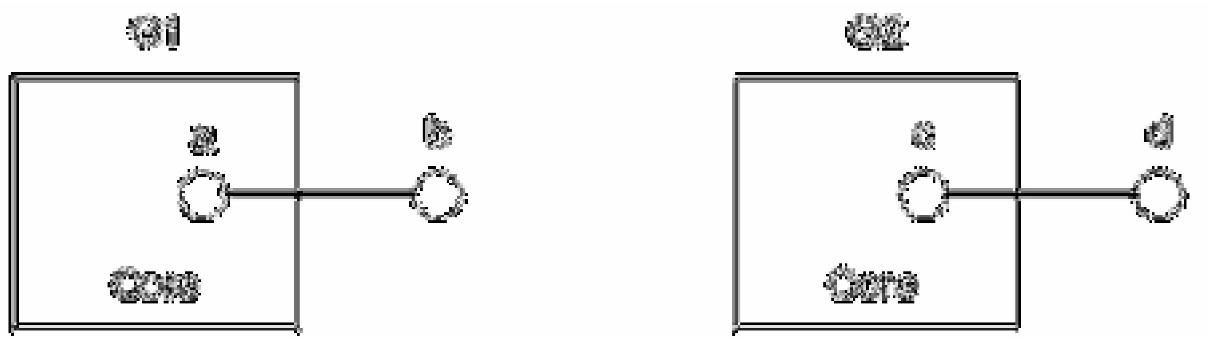
Candidate generation based on core detection

Hai đồ thị con có thể có nhiều lõi chung khác nhau

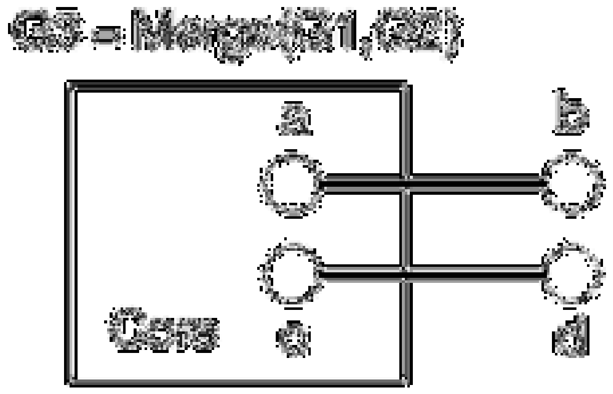


General for generating functions in FGM

- Cho hai đồ thị:

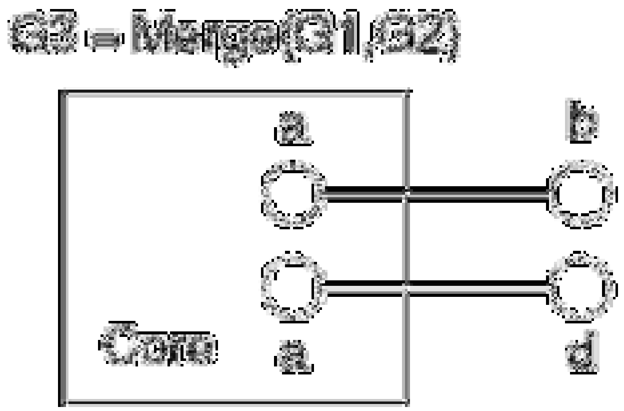


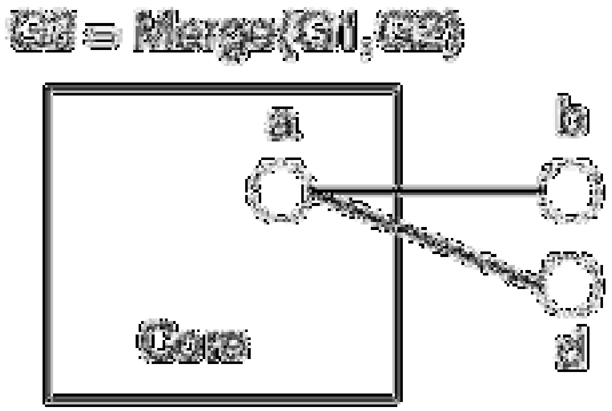
- Trường hợp 1: $a \neq c$ và b $\neq d$



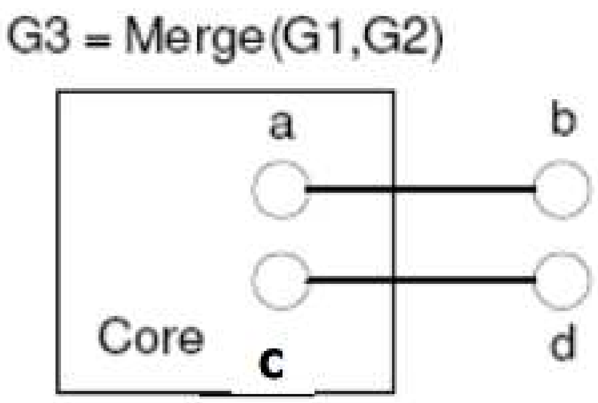
General for generating functions in FGM

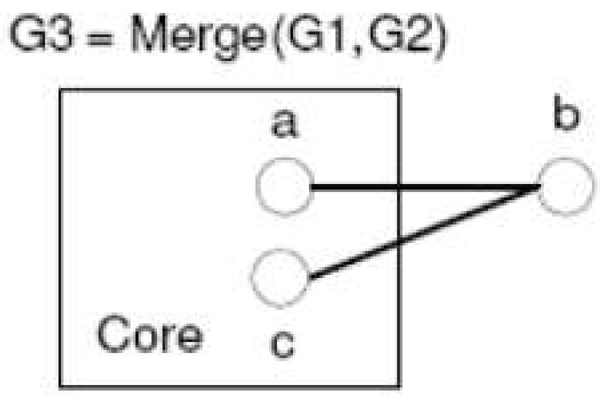
- Trường hợp 2: $a = c$ và b $\neq d$





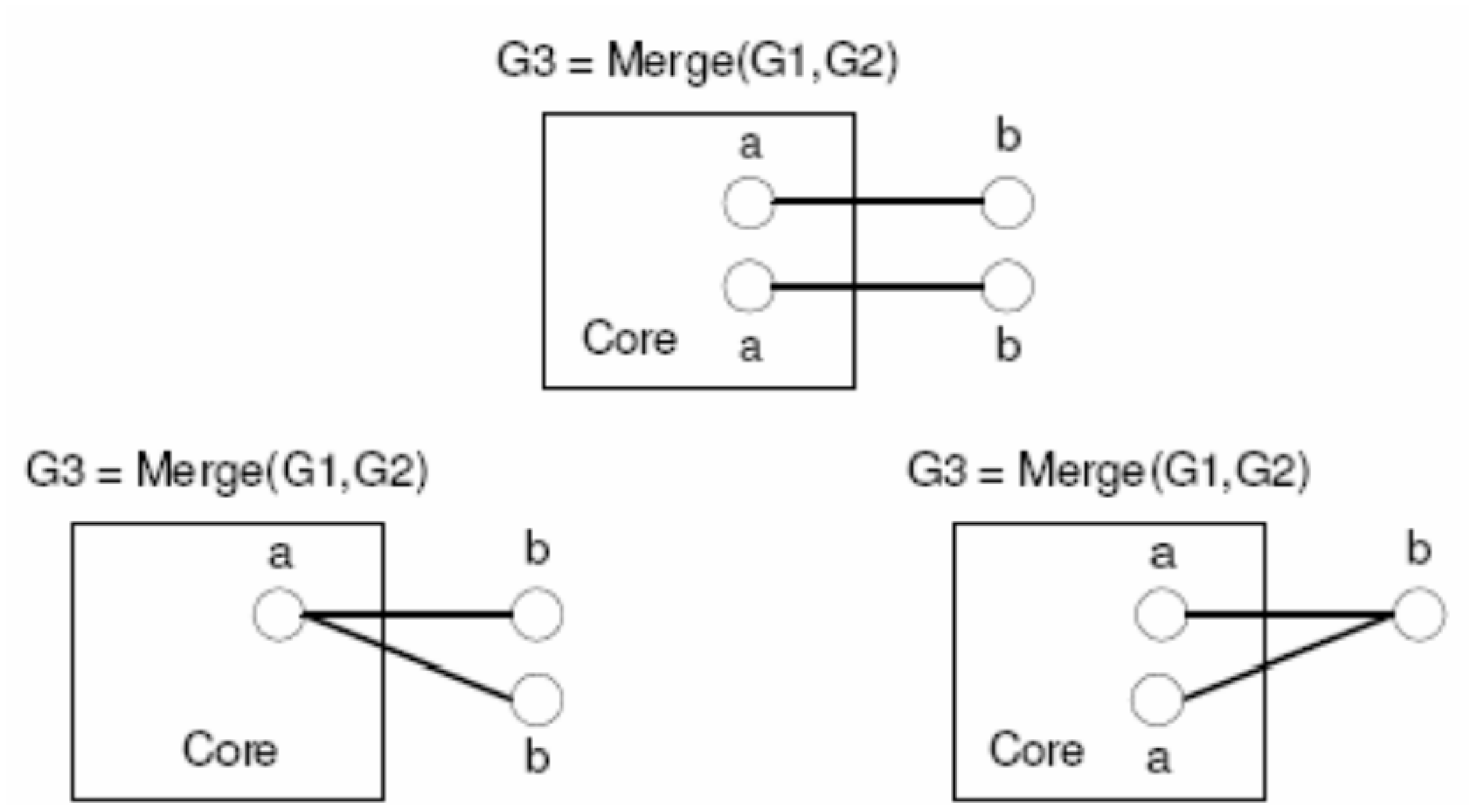
- Trường hợp 3: $a \neq c$ và b $= d$





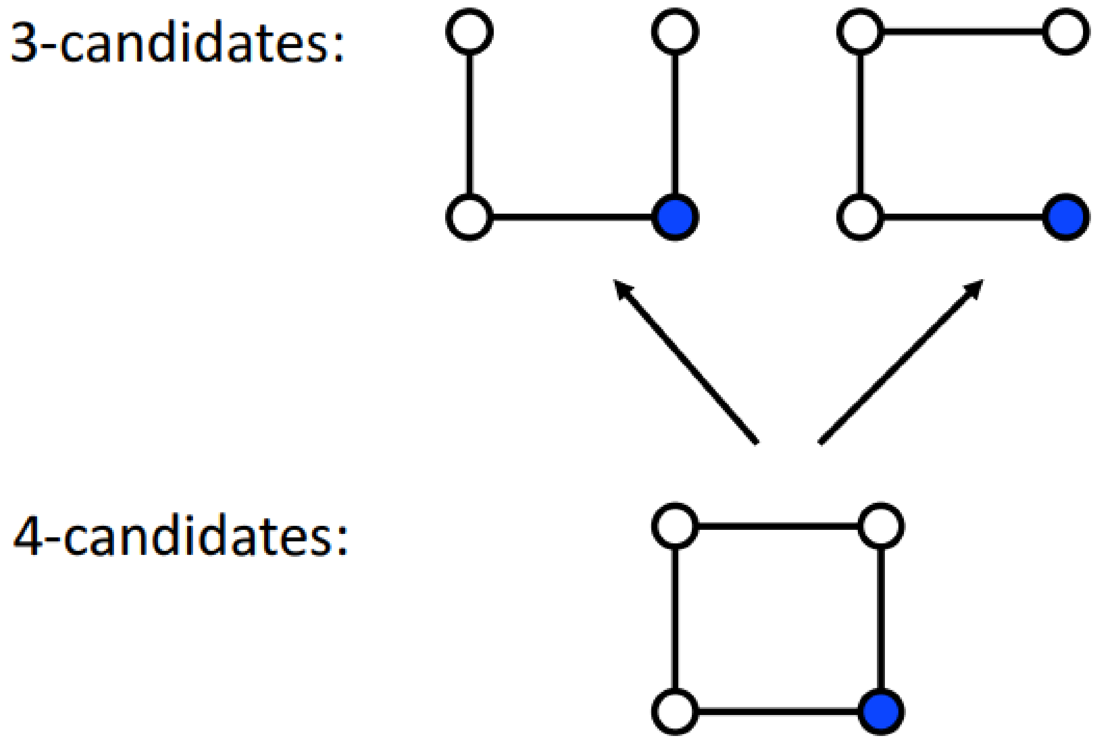
General for generating functions in FGM

- Trường hợp 4: $a = c$ và b $= d$

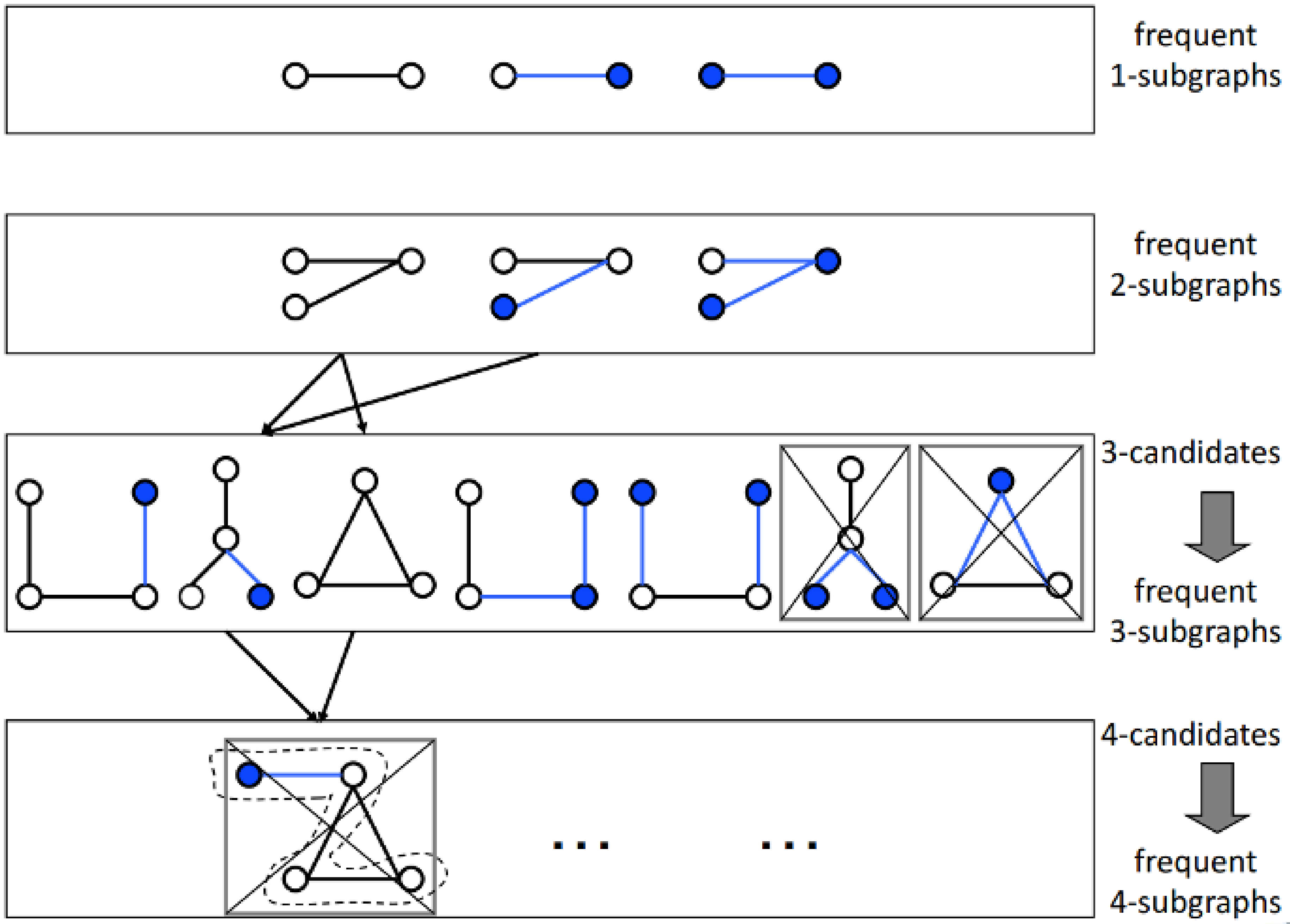


Pruning branches

- During branching, every k-1 subgraph must be universal.



Full process example in FGM



Comment

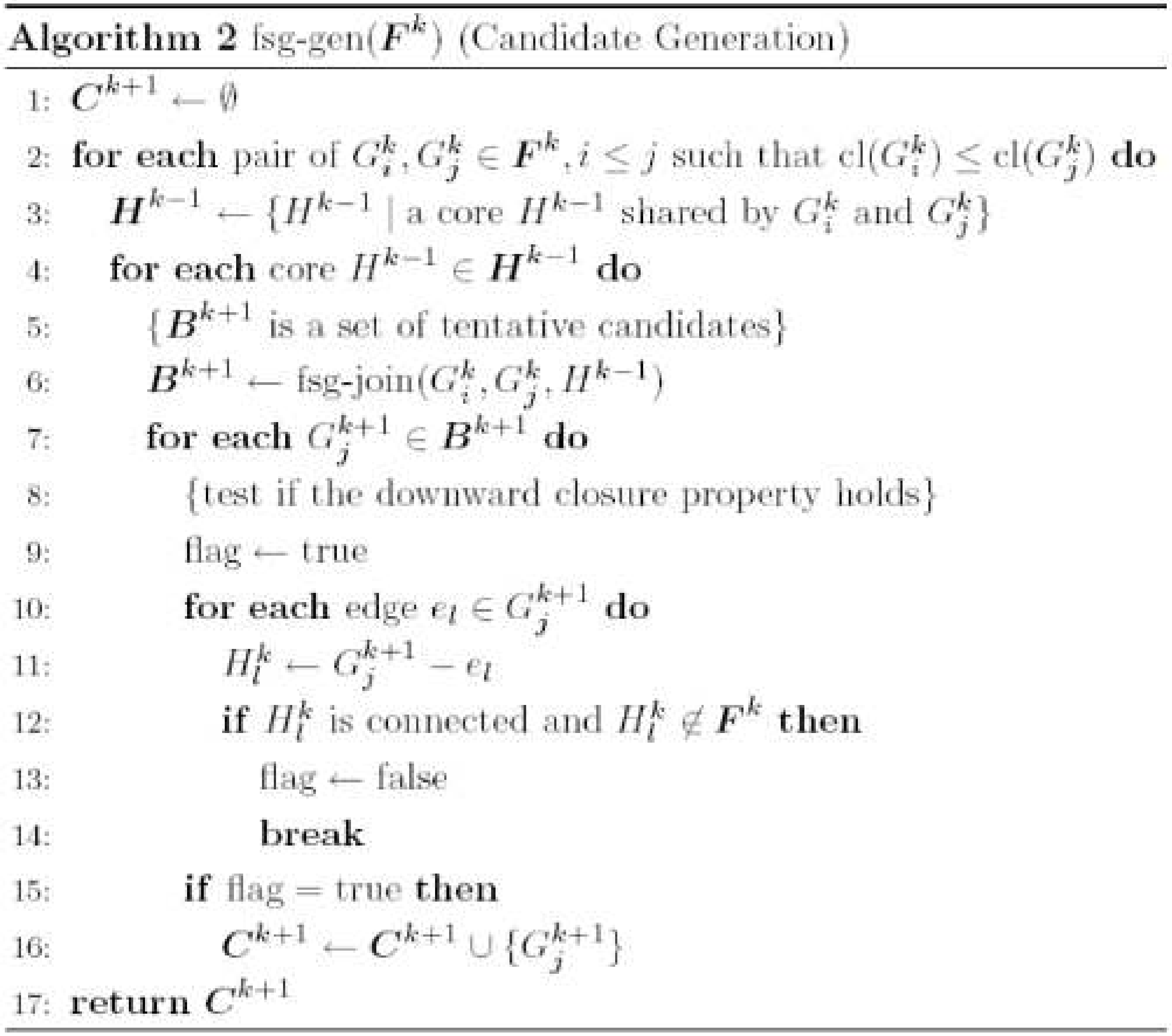
- Candidate generation process:
  - To determine the two candidates needed to join, we need to examine the isomorphic subgraph → high cost
- Pruning candidates:
  - To check the properties of the downward closure, we also need to check the isomorphic graph → high cost
- Popularity Count:
  - The isomorphic subgraph needs to be performed to check if the graph contains this subgraph → high cost

NP-Complete

FGM optimization

- To reduce computational costs for FGM:
  - Apply canonical labeling to isomorphism
  - Use improved candidate generation algorithm to reduce the number of times each candidate is inflated
  - Apply augmented TID-list based method to speed up popularity counting
- Although the optimization process makes FGM better, it still generally falls under the NP-Complete problem.

FGM candidate generation function



Algorithm 2 fsg-gen(F^k) (Candidate Generation)

```
1:  $C^{k+1} \leftarrow \emptyset$
2:  for each pair of $G_i^k, G_j^k \in F^k, i \le j$ such that $\mathrm{cl}(G_i^k) \le \mathrm{cl}(G_j^k)$ do
3:      $H^{k-1} \leftarrow \{H^{k-1} \mid$ a core $H^{k-1}$ shared by $G_i^k$ and $G_j^k\}$
4:      for each core $H^{k-1} \in H^{k-1}$ do
5:          {$B^{k+1}$ is a set of tentative candidates}
6:          $B^{k+1} \leftarrow$ fsg-join($G_i^k, G_j^k, H^{k-1}$)
7:          for each $G_j^{k+1} \in B^{k+1}$ do
8:              {test if the downward closure property holds}
9:              flag $\leftarrow$ true
10:             for each edge $e_l \in G_j^{k+1}$ do
11:                 $H_l^k \leftarrow G_j^{k+1} - e_l$
12:                 if $H_l^k$ is connected and $H_l^k \notin F^k$ then
13:                     flag $\leftarrow$ false
14:                     break
15:             if flag = true then
16:                 $C^{k+1} \leftarrow C^{k+1} \cup \{G_j^{k+1}\}$
17: return $C^{k+1}$
```

← For each pair of frequent - subgraph(canonical labeling -cl)

← Detect shared core

← Generates all possible candidates of size *k*+1

← Test downward closure property

← Add to candidate set

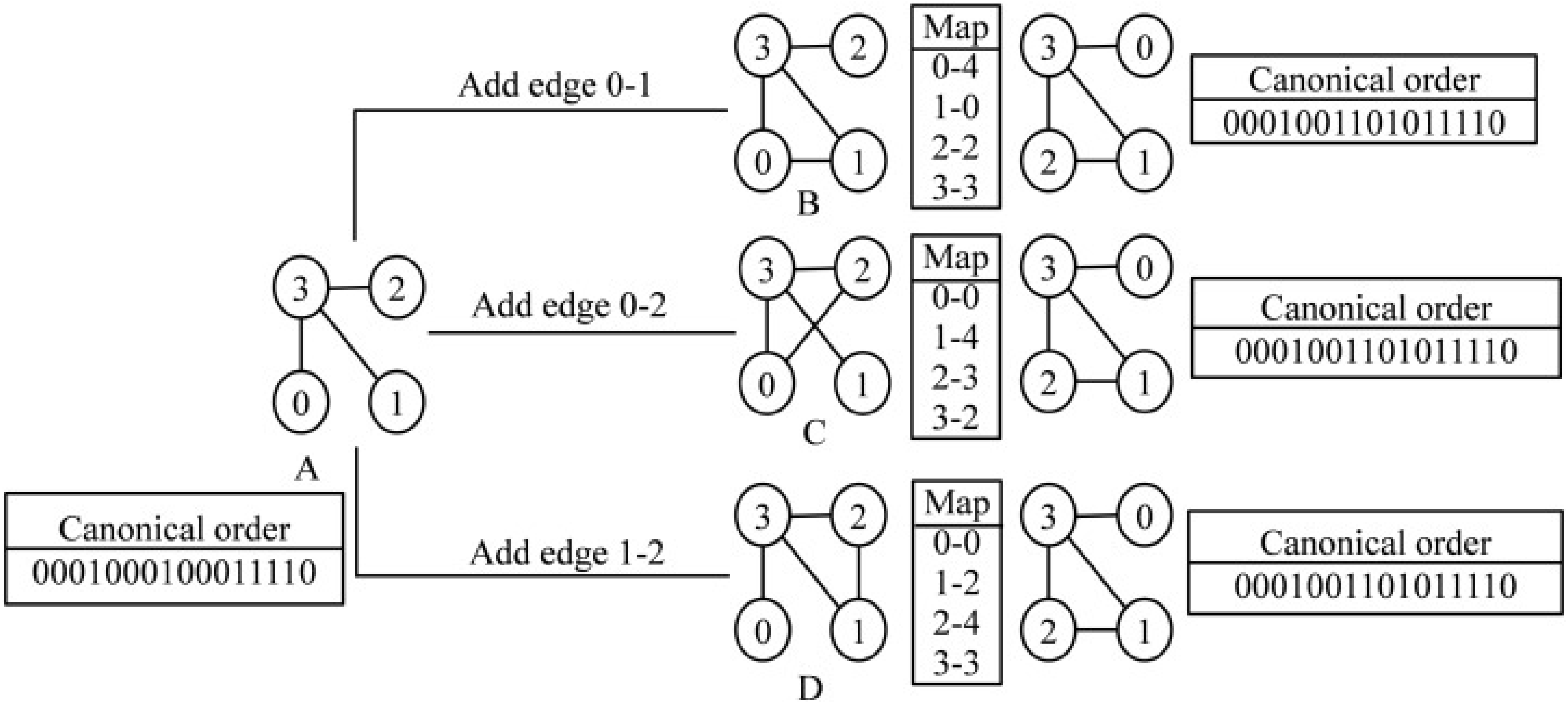
FGM candidate generation function

Algorithm 3 fsg-join(G_1^k, G_2^k, H^{k-1}) (Join)

```
1: $e_1 \leftarrow$ the edge appears only in $G_1^k$, not in $H^{k-1}$
2: $e_2 \leftarrow$ the edge appears only in $G_2^k$, not in $H^{k-1}$
3: $M \leftarrow$ generate all automorphisms of $H^{k-1}$
4: $\boldsymbol{B}^{k+1} = \emptyset$
5: for each automorphism $\phi \in M$ do
6:     $\boldsymbol{B}^{k+1} \leftarrow \boldsymbol{B}^{k+1} \cup$ {all possible candidates of size $k+1$ created from a set of $e_1$, $e_2$, $H^{k-1}$ and $\phi$}
7: return $\boldsymbol{B}^{k+1}$
```

Canonical label

- The canonical label of a graph is a unique code that identifies the graph
- Two graphs are isomorphic to each other if they are attached to the same code.
  - There are many ways to define canonical labels for graphs

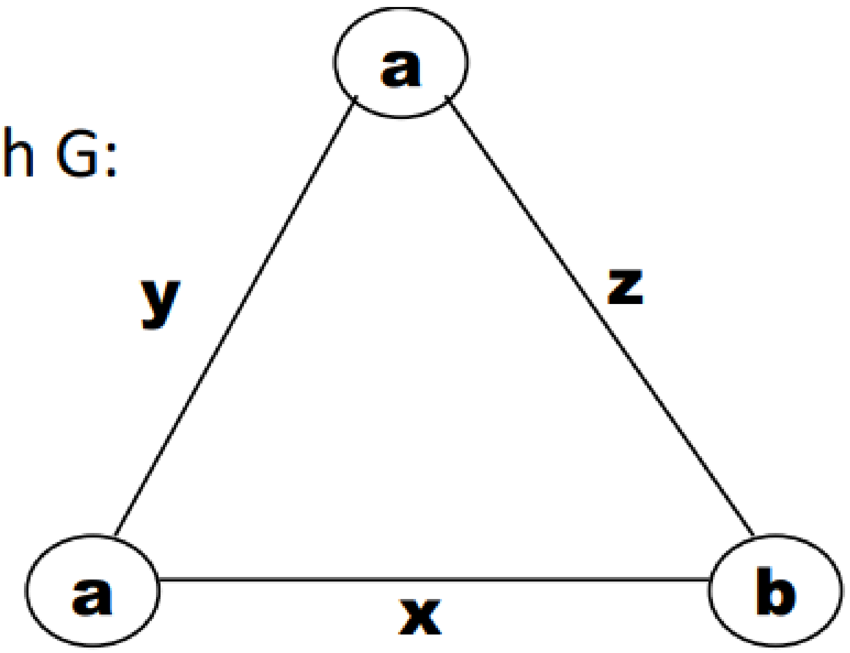


Define the canonical label

- A simple way to assign code to a graph is to convert its adjacent matrix representation into a linear symbol sequence.

$$\text{Code}(M_1) = \text{“}\boxed{\text{aab}}\boxed{\text{yzx}}\text{”}$$

Graph G:



$$M_1: \begin{array}{c|c|c|c|}
 & \mathbf{a} & \mathbf{a} & \mathbf{b} \\
\hline
\mathbf{a} & & \boxed{y} & Z \\
\hline
\mathbf{a} & y & & X \\
\hline
\mathbf{b} & Z & X & \\
\hline
\end{array}$$

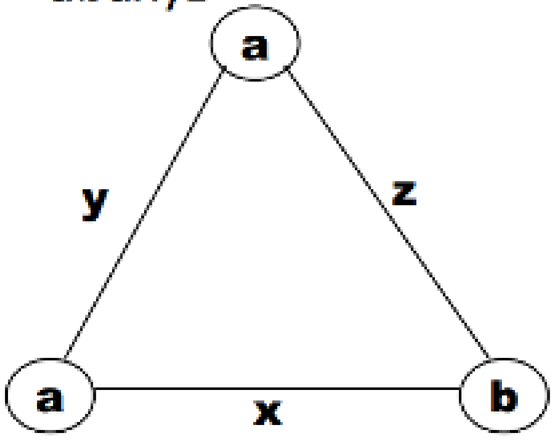
Define the canonical label

- Because each graph can form many types of adjacent matrices depending on the order of vertices.
  - To get an isomorphism-invariant code is to try every possible permutation of vertices.
  - Choose the order in which for its largest or smallest code.

$\text{Code}(M_1)$ = "aabyzx"

$\text{Code}(M_2)$ = "abaxyz"

Graph G:



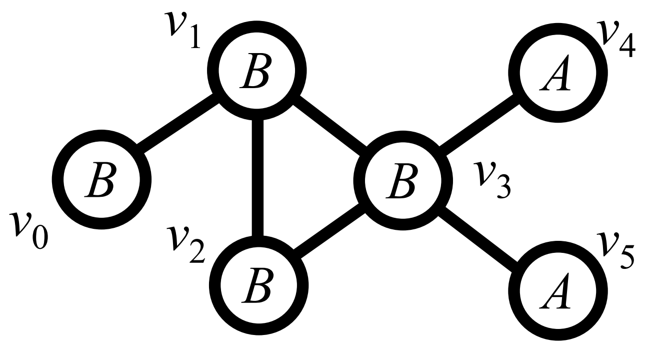
$$M_1: \begin{array}{c|ccc} & a & a & b \\ \hline a & & y & z \\ a & y & & x \\ b & z & x & \end{array}$$

$$M_2: \begin{array}{c|ccc} & a & b & a \\ \hline a & & x & y \\ b & x & & z \\ a & y & z & \end{array}$$

Canonical-Code(G) = min{ code(M) | M is adj. Matrix}

String comparison!

Define the canonical label



$$\left[\begin{array}{cc|cccccc} & & v_3 & v_1 & v_2 & v_4 & v_5 & v_0 \\ & & B & B & B & A & A & B \\ \hline v_3 & B & & 1 & 1 & 1 & 1 & \\ v_1 & B & 1 & & 1 & & & 1 \\ v_2 & B & 1 & 1 & & & & \\ v_4 & A & 1 & & & & & \\ v_5 & A & 1 & & & & & \\ v_0 & B & & 1 & & & & \end{array}\right]$$

Label = “1 11 100 1000 01000”

$$\left[\begin{array}{cc|cccccc} & & v_0 & v_1 & v_2 & v_3 & v_4 & v_5 \\ & & B & B & B & B & A & A \\ \hline v_0 & B & & 1 & & & & \\ v_1 & B & 1 & & 1 & 1 & & \\ v_2 & B & & 1 & & 1 & & \\ v_3 & B & & 1 & 1 & & 1 & 1 \\ v_4 & A & & & & 1 & & \\ v_5 & A & & & & 1 & & \end{array}\right]$$

Label = “1 01 011 0001 00010”

Define the canonical label

- The problem of finding canonical labels is as complicated as the isomorphism of a graph because all possible combinations must be checked.
- FSG proposes several heuristic ways to accelerate:
  - Based on vertex invariants (e.g. orders)
  - List of neighbors
  - Iterative partitioning
- The main idea of these methods is to help alleviate the same cases.

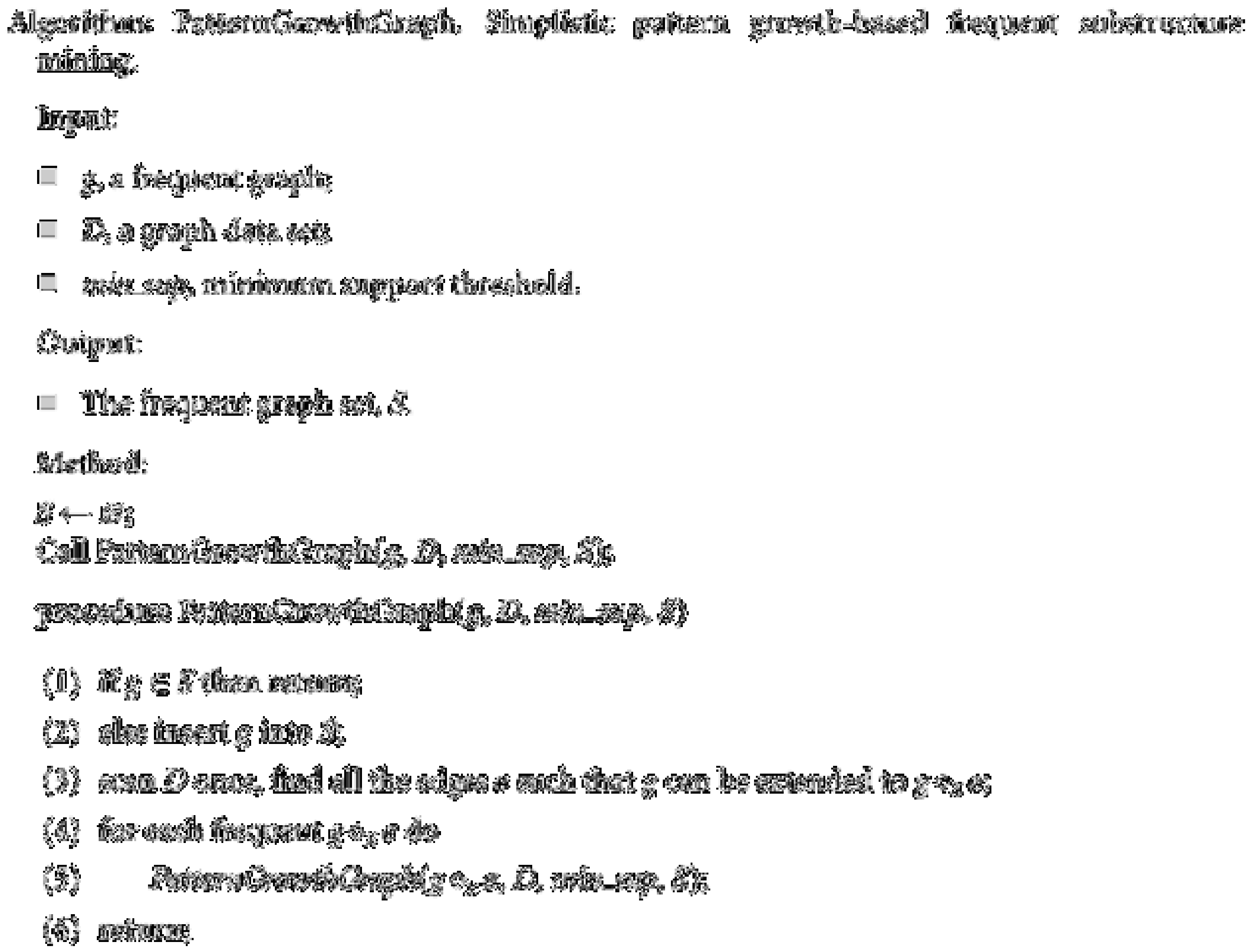
Nội dung

- Mẫu đồ thị con phổ biến
- Phương pháp tìm mẫu phổ biến dựa trên Apriori
- **Phương pháp tìm mẫu phổ biến dựa trên chiều sâu**
- Phương pháp tìm mẫu phổ biến tham lam

Phương pháp phát triển mẫu

- Phương pháp phát triển mẫu được thực hiện:
  - Một đồ thị g được mở rộng bằng cách thêm một cạnh mới e.
  - Kiểm tra độ phổ biến của đồ thị mới
    - Nếu thỏa, phát triển tiếp mẫu mới
  - Lặp lại cho đến khi tất cả đồ thị con phổ biến của đồ thị đã được tìm ra.

Phương pháp phát triển mẫu



Algorithm: PatternGrowthGraph. Simplistic pattern growth-based frequent substructure mining.

Input:

- g, a frequent graph;
- D, a graph data set;
- min_sup, minimum support threshold.

Output:

- The frequent graph set, S.

Method:

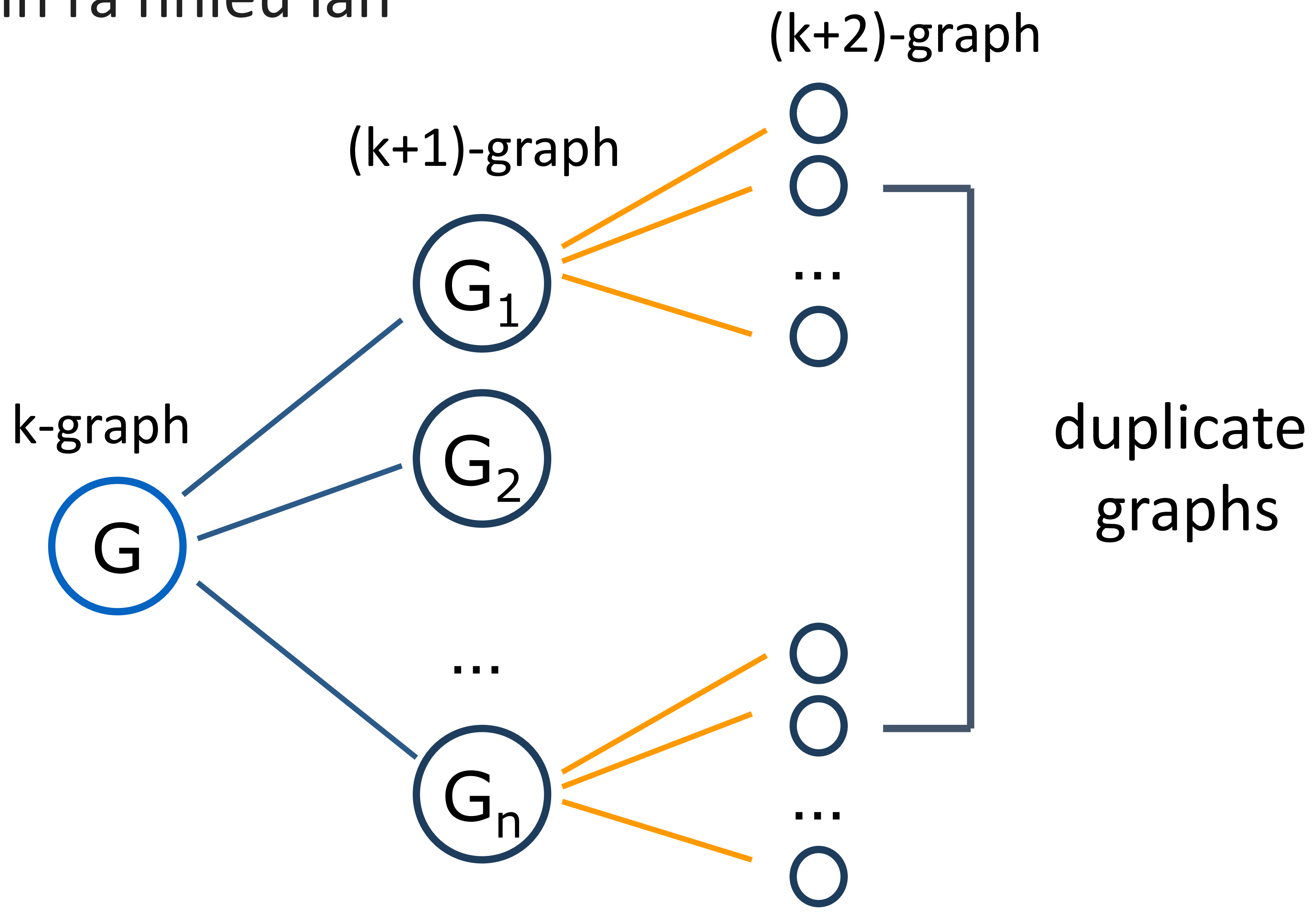
```
$S \leftarrow \emptyset$;
Call PatternGrowthGraph($g$, $D$, $min\_sup$, $S$);

procedure PatternGrowthGraph($g$, $D$, $min\_sup$, $S$)

(1) if $g \in S$ then return;
(2) else insert $g$ into $S$;
(3) scan $D$ once, find all the edges $e$ such that $g$ can be extended to $g \diamond_x e$;
(4) for each frequent $g \diamond_x e$ do
(5)         PatternGrowthGraph($g \diamond_x e$, $D$, $min\_sup$, $S$);
(6) return;
```

Phương pháp phát triển mẫu

- Một vấn đề xảy ra là các đồ thị giống nhau có thể sinh ra nhiều lần



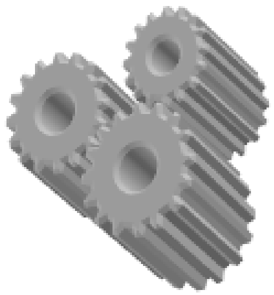
gSpan algorithm

- gSpan (Graph-based Substructure Pattern Mining) dựa trên phương pháp phát triển mẫu nhưng:
  - Giảm việc phát sinh ra các đồ thị trùng lắp
  - Cải thiện việc kiểm tra đẳng cấu
- Ý tưởng dựa trên:
  - Chuyển đồ thị (2-chiều) thành tuần tự cạnh được mã hóa theo cách duyệt chiều sâu (DFS)
  - Chọn mã DFS nhỏ nhất

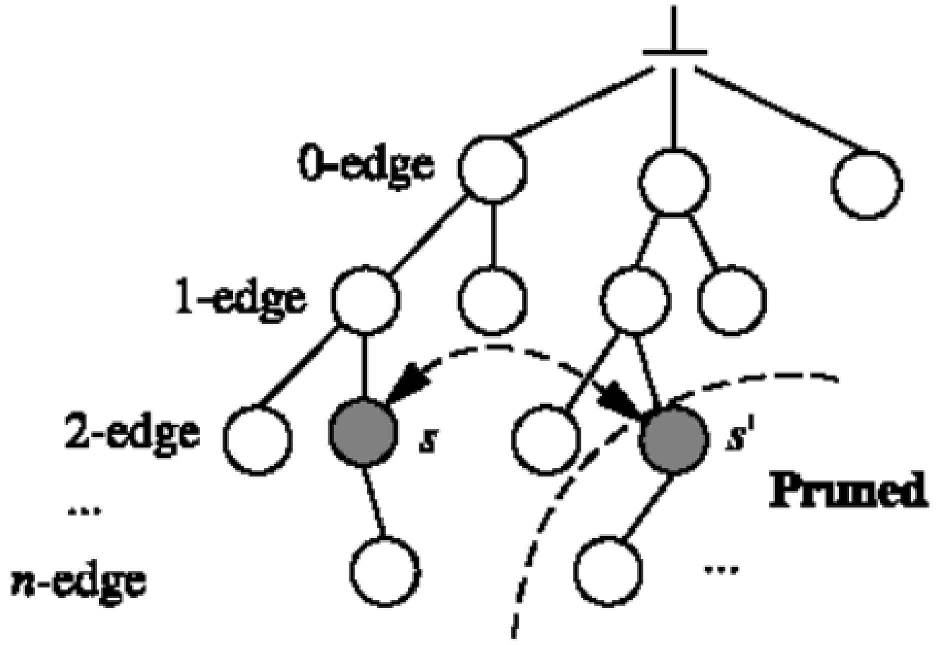
gSpan algorithm

- Thuật toán trải qua hai bước chính:
  - Bước 1: Xây dựng không gian tìm kiếm dựa trên cây (TSS)

  - Bước 2: Tìm tất cả các đồ thị phổ biến thông qua TSS

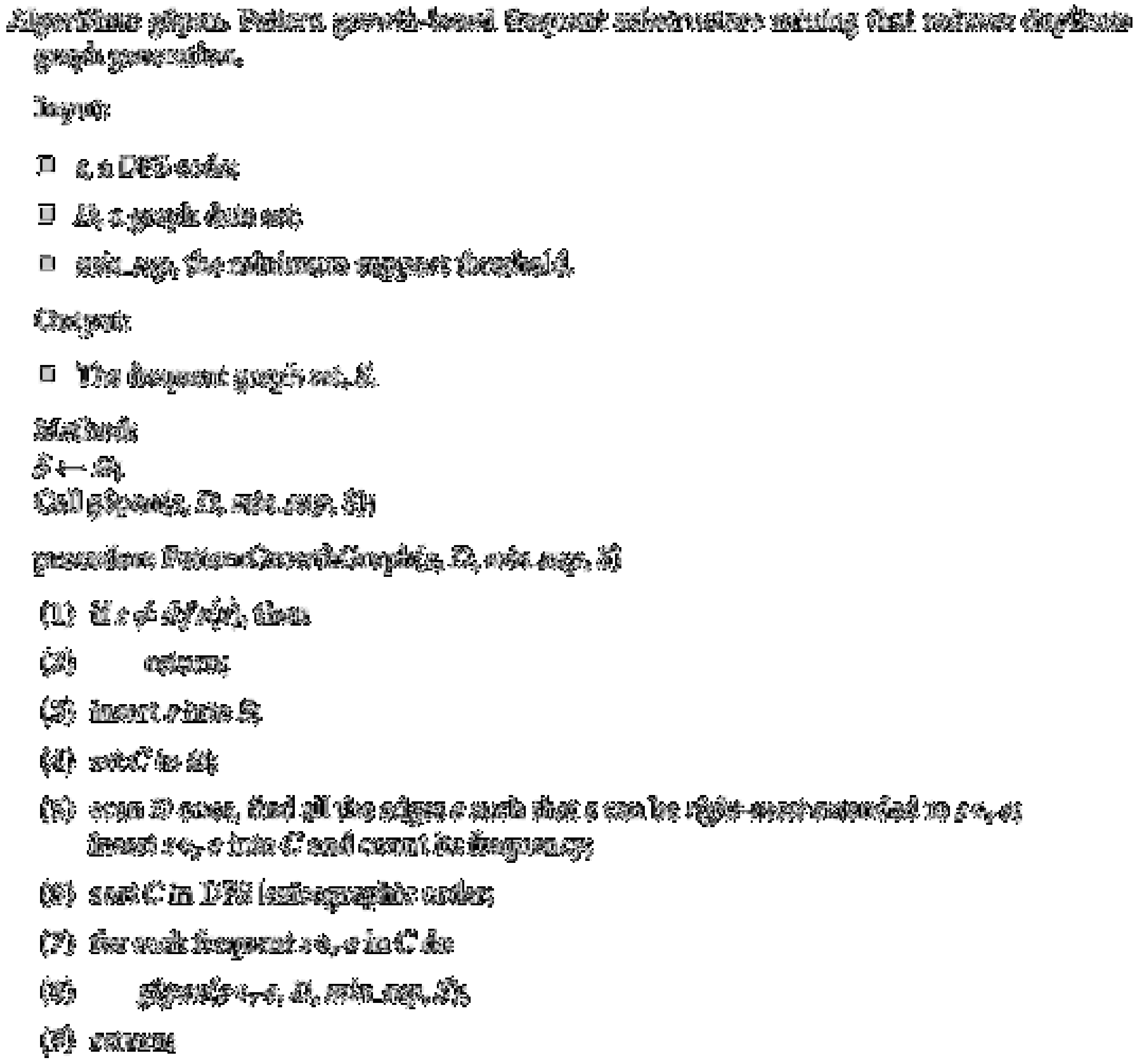


gSpan algorithm



- Root: đỉnh rỗng
- Mỗi đỉnh là một mã DFS ứng với một đồ thị
- Mỗi cạnh: được mở rộng theo hướng ngoài cùng bên phải (right most extension) từ mã DFS chiều dài k-1 đến một mã DFS chiều dài k
  - Nếu mã s và s’ mã hóa cùng một đồ thị, không gian tìm kiếm có thể bị tỉa tại s’.

gSpan algorithm



Algorithm: gSpan. Pattern growth-based frequent substructure mining that reduces duplicate graph generation.

Input:

- s, a DFS code;
- D, a graph data set;
- min_sup, the minimum support threshold.

Output:

- The frequent graph set, S.

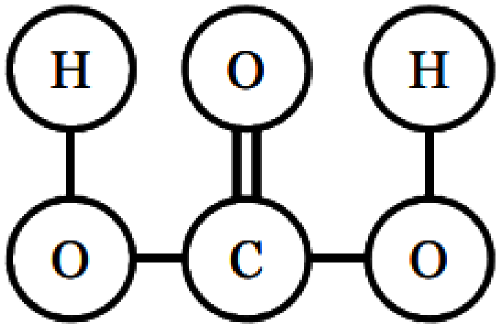
Method:

```
S ← ∅;
Call gSpan(s, D, min_sup, S);

procedure PatternGrowthGraph(s, D, min_sup, S)
(1) if s ≠ dfs(s), then
(2)         return;
(3) insert s into S;
(4) set C to ∅;
(5) scan D once, find all the edges e such that s can be right-most extended to s ◇r e;
    insert s ◇r e into C and count its frequency;
(6) sort C in DFS lexicographic order;
(7) for each frequent s ◇r e in C do
(8)         gSpan(s ◇r e, D, min_sup, S);
(9) return;
```

gSpan algorithm

- Input:
  - A set of graphs and thresholds
  - Each graph takes the form $G = (V, E, L_V, L_E)$
    - V is the vertex set
    - E is the edge set
    - L_V is the label of the vertices
    - L_E is the label of the edges
    - Unique optional label

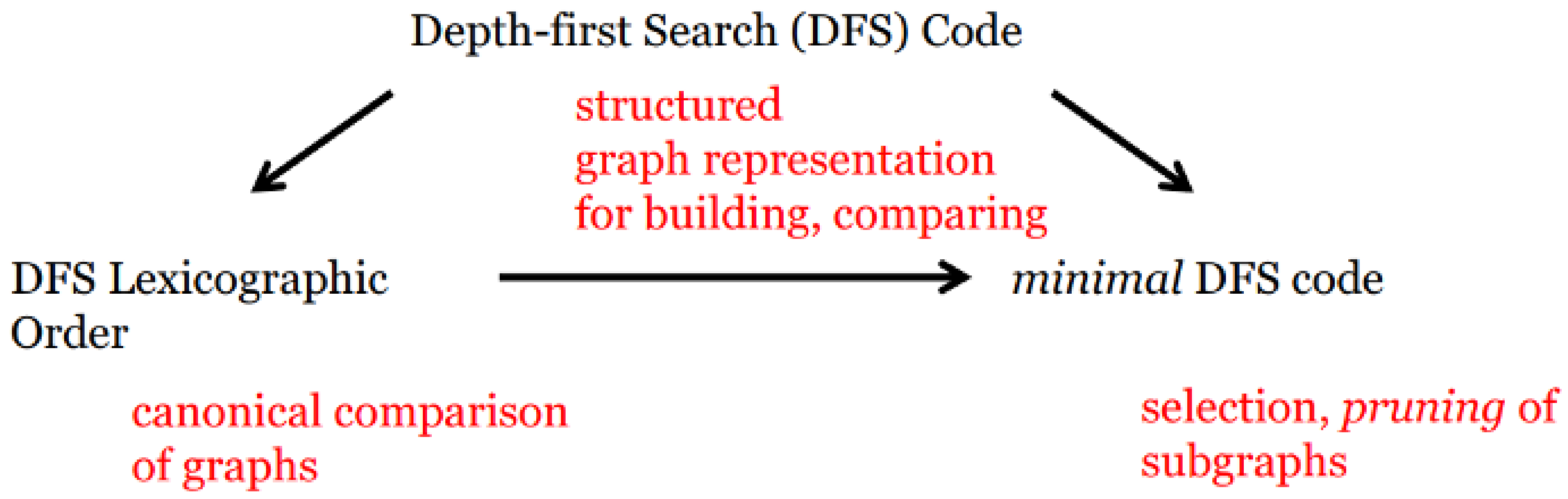


$$L_V = \{ H, O, C \}$$

$$L_E = \{ \text{single}-\text{bond}, \text{double}-\text{bond} \}$$

gSpan algorithm

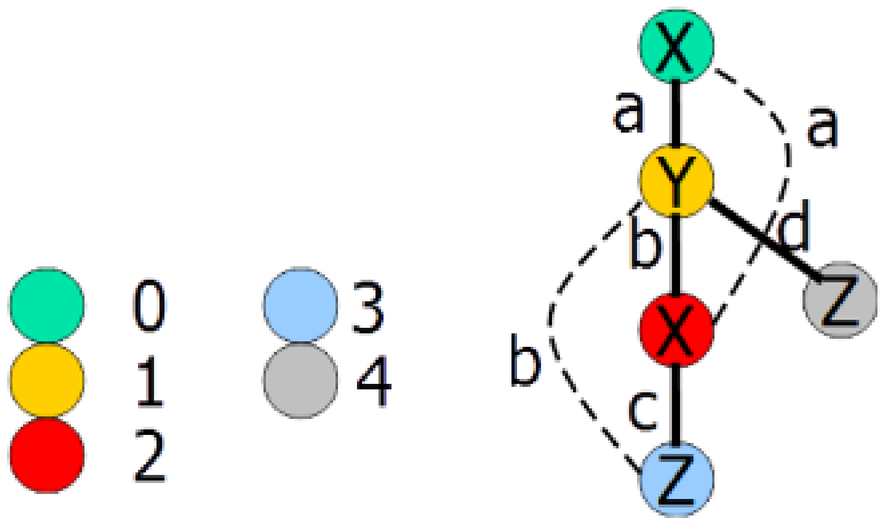
- Strategy:
  - Build common bottom-up subgraphs, using DFS code as the canonical representation for the graph.
  - Remove duplicates through minimal DFS code based on the dictionary order of the code.



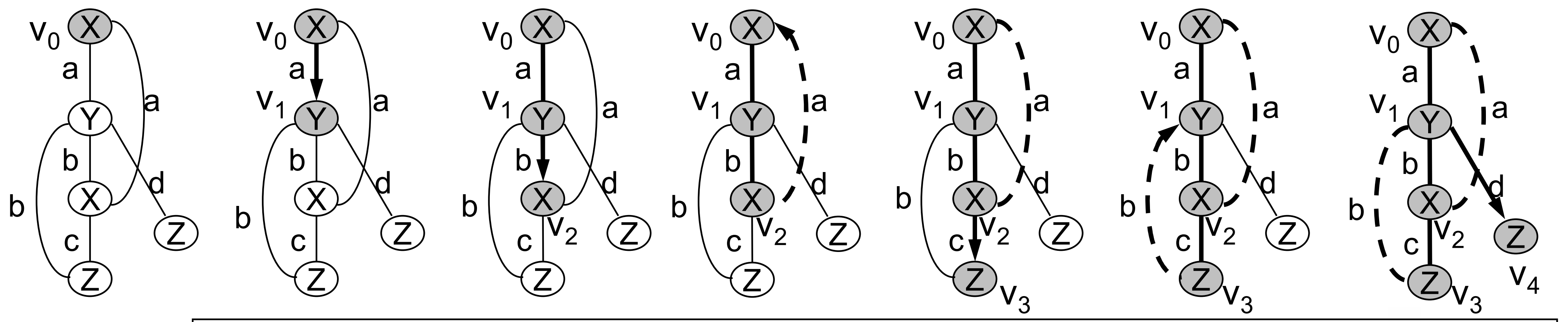
DFS code in gSpan

- DFS code is the sequential of edges browsed using DFS
  - Each edge is encoded as $(i, j, L_i, L_{(i\ j)}, L_j)$
    - i, j: are vertices
    - L_i, L_j: Top and end labels of edges
    - $L_{(i\ j)}$: edge labels
    - $i < j$: called the forward edge
    - $i > j$: called the back edge

DFS code in gSpan



Cạnh #	Mã cạnh
1	(0,1,X,a,Y)
2	(1,2,Y,b,X)
3	(2,0,X,a,X)
4	(2,3,X,c,Z)
5	(3,1,Z,b,Y)
6	(1,4,Y,d,Z)

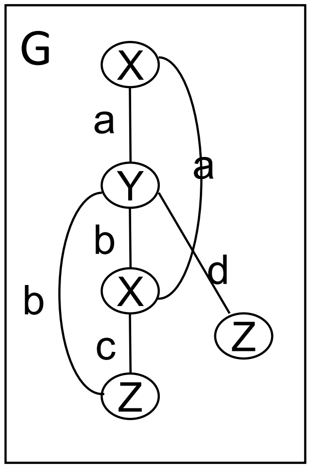


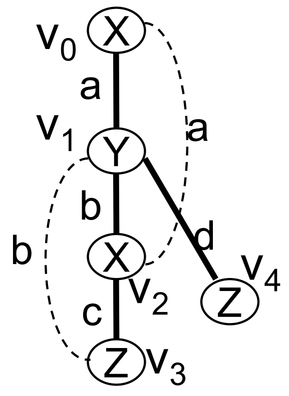
Mã DFS	(0,1,X,a,Y)	(1,2,Y,b,X)	(2,0,X,a,X)	(2,3,X,c,Z)	(3,1,Z,b,Y)	(1,4,Y,d,Z)

DFS code in gSpan

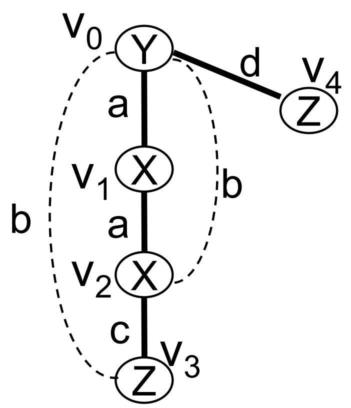
- With a given graph, there can be multiple DFS codes depends on graph traversal way và choosing a starting point.

	(a)	(b)	(c)
1	(0, 1, X, a, Y)	(0, 1, Y, a, X)	(0, 1, X, a, X)
2	(1, 2, Y, b, X)	(1, 2, X, a, X)	(1, 2, X, a, Y)
3	(2, 0, X, a, X)	(2, 0, X, b, Y)	(2, 0, Y, b, X)
4	(2, 3, X, c, Z)	(2, 3, X, c, Z)	(2, 3, Y, b, Z)
5	(3, 1, Z, b, Y)	(3, 0, Z, b, Y)	(3, 0, Z, c, X)
6	(1, 4, Y, d, Z)	(0, 4, Y, d, Z)	(2, 4, Y, d, Z)

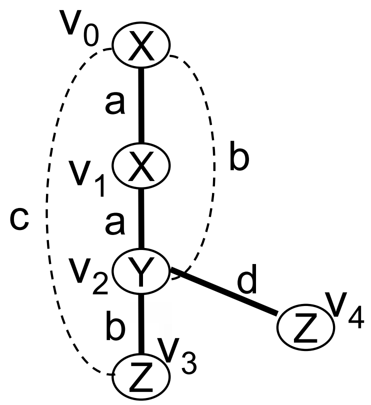




(a)



(b)



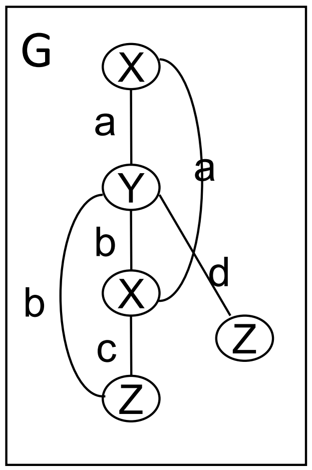
(c)

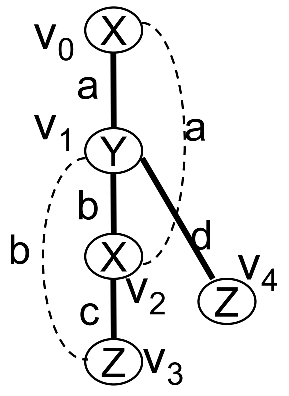
DFS code in gSpan

- It is necessary to select the smallest DFS code

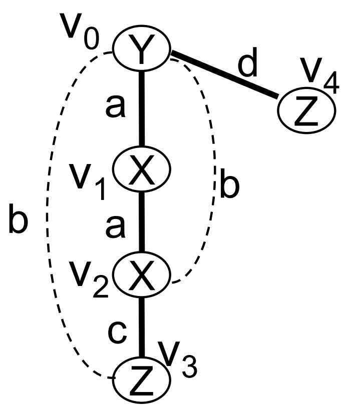
Min DFS-Code

	(a)	(b)	(c)
1	(0, 1, X, a, Y)	(0, 1, Y, a, X)	**(0, 1, X, a, X)**
2	(1, 2, Y, b, X)	(1, 2, X, a, X)	**(1, 2, X, a, Y)**
3	(2, 0, X, a, X)	(2, 0, X, b, Y)	**(2, 0, Y, b, X)**
4	(2, 3, X, c, Z)	(2, 3, X, c, Z)	**(2, 3, Y, b, Z)**
5	(3, 1, Z, b, Y)	(3, 0, Z, b, Y)	**(3, 0, Z, c, X)**
6	(1, 4, Y, d, Z)	(0, 4, Y, d, Z)	**(2, 4, Y, d, Z)**

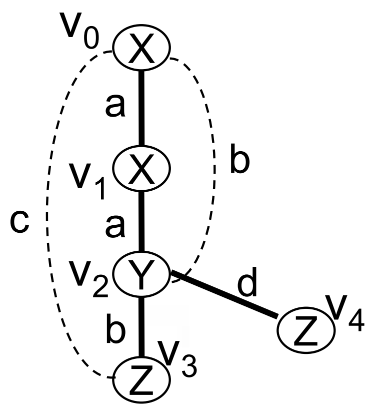




(a)



(b)

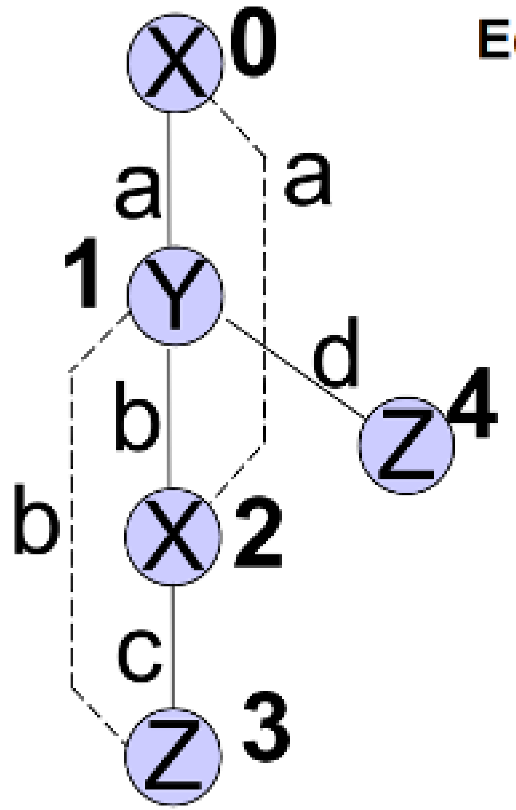


(c)

Valid edge order in DFS code

- To determine the minimum DFS code, we need to agree on an edge order so that DFS codes can be compared.
- Symbol:
  - $e_1 = (i_1, j_1), e_2 = (i_2, j_2)$
  - $e_1 \prec e_2$: mean e_1 appeared first e_2 in code
- Ordinal Law:
  - Rule 1: if $i_1 = i_2$ and $j_1 < j_2$ then $e_1 \prec e_2$
    - From the same source vertex, e_1 pre-approved e_2 in DFS
  - Rule 2: if $i_1 < j_1$ và $j_1 = i_2$ then $e_1 \prec e_2$ (next edge)
    - e_1 is an incoming edge and e_2 traversed as a result of traversal e_1
  - Rule 3: if $e_1 \prec e_2$ và $e_2 \prec e_3$ then $e_1 \prec e_3$ (bridge)

Valid edge order in DFS code



Edge #	Code
0	(0,1,X,a,Y)
1	(1,2,Y,b,X)
2	(2,0,X,a,X)
3	(2,3,X,c,Z)
4	(3,1,Z,b,Y)
5	(1,4,Y,d,Z)

- $0 \prec 1$ (Rule 2)
- $1 \prec 2$ (Rule 2)
- $0 \prec 2$ (Rule 3)
- $2 \prec 3$ (Rule 1)
- • • •

DFS code and DFS dictionary order

- Note to distinguish between DFS code and DFS code dictionary order
  - DFS Code: is the order of the edges of a particular DFS.
  - DFS dictionary order: is the order between the different DFS codes.

DFS dictionary order

- Given in advance :
  - Dictionary order of the set of vertex and edge labels L (symbol $\prec_L$)
  - GraphG_α, G_β (with the same set of labels)
  - DFS codes:
    - $\alpha = code(G_\alpha, T_\alpha) = (a_0, a_1, \dots, a_m)$
    - $\beta = code(G_\beta, T_\beta) = (b_0, b_1, \dots, b_n)$
    - Suppose $m \leq n$
- $\alpha \leq \beta$ if and only if one of the following conditions is met::
  - $\exists t, 0 \leq t \leq \min(m, n)$ so that
    - $a_k = b_k$ with $k < t$ and
    - $a_t \prec_e b_t$
  - $a_k = b_k$ for every k in the paragraph $0 \leq k \leq m$

Edge comparison $a_t \prec_e b_t$

- Call:
  - $a_t = (i_a, j_a, L_{i_a}, L_{i_a, j_a}, L_{j_a})$
  - $b_t = (i_b, j_b, L_{i_b}, L_{i_b, j_b}, L_{j_b})$
- $a_t \prec_e b_t$ if one of the following cases is satisfied::
  - TH1: Both sides are the incident edge (j_a = j_b because the front edges are equal) and ...
  - Q2: Both sides are backward edges (i_a = i_b for the same reason) and...
  - TH3: a_t is the backward edge and b_t is the forward edge

Edge comparison $a_t \prec_e b_t$

- Call:
  - $a_t = (i_a, j_a, L_{i_a}, L_{i_a, j_a}, L_{j_a})$
  - $b_t = (i_b, j_b, L_{i_b}, L_{i_b, j_b}, L_{j_b})$
- $a_t \prec_e b_t$ if one of the following cases is satisfied:
  - TH1: Both edges are incoming edges ($j_a = j_b$ since the anterior edges are equal) and satisfying one in the condition:
    - $i_b < i_a$ (The edge starting from a vertex is visited later)
    - $i_a = i_b$: and the labels of A are in order of less than the labels of B in triplets.
      - Example: $a_t = (-, -, m, e, x)$, $b_t = (-, -, m, u, x)$

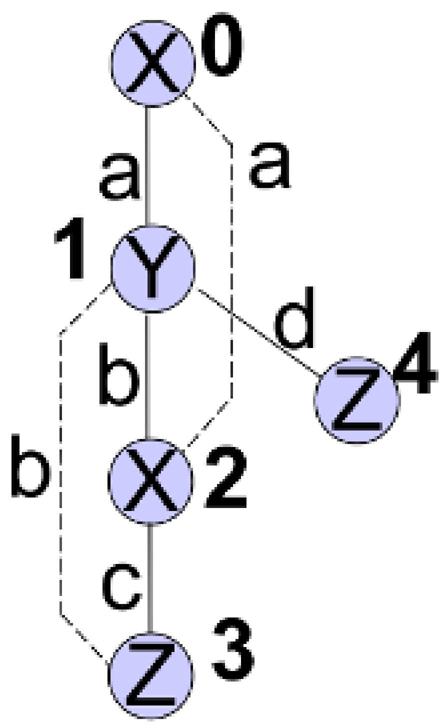
        $m = m, e < u \rightarrow a_t \prec_e b_t$

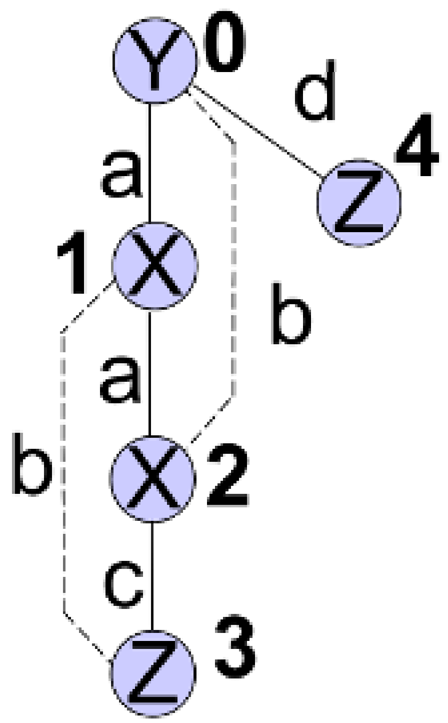
Edge comparison $a_t \prec_e b_t$

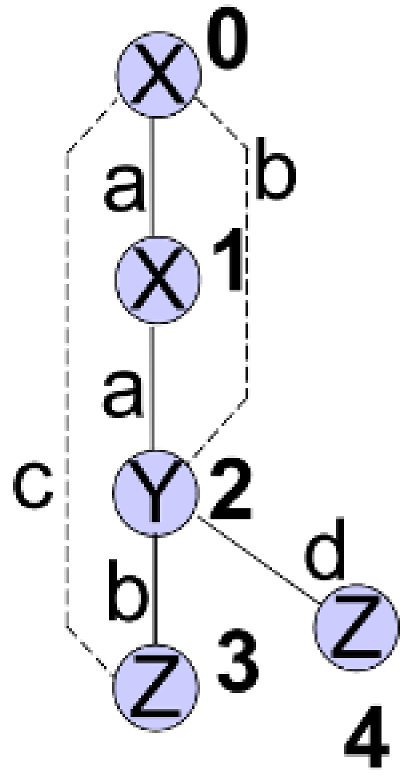
- Call:
  - $a_t = (i_a, j_a, L_{i_a}, L_{i_a,j_a}, L_{j_a})$
  - $b_t = (i_b, j_b, L_{i_b}, L_{i_b,j_b}, L_{j_b})$
- $a_t \prec_e b_t$ if one of the following cases is satisfied:
  - TH1: …
  - TH2: Both sides are backward edges (i_a=i_b for the same reason) and meet one of the following conditions:
    - $j_a < j_b$ (The connecting edge to a pre-visited vertex)
    - $j_a = j_b$ and edge labels of a has an order of smaller than the edge label of b.
      - ➢ Do not consider the vertex label because all the front edges are equal, so the vertex labels are already equal.

Examples of DFS codes

Edge #	Code (A)	Code (B)	Code (C)
0	(0,1,X,a,Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	(1,2,X,a,X)	(1,2,X,a,Y)
2	(2,0,X,a,X)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	(3,1,Z,b,X)	(3,0,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)



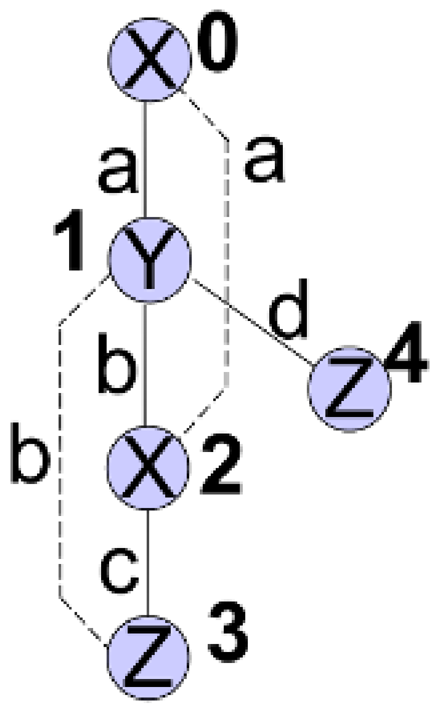


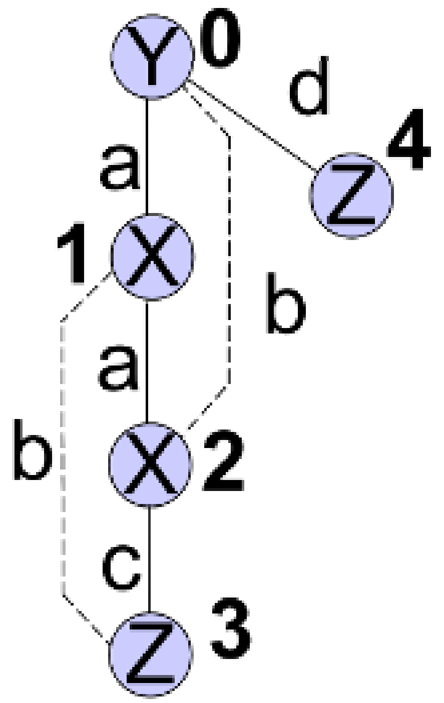


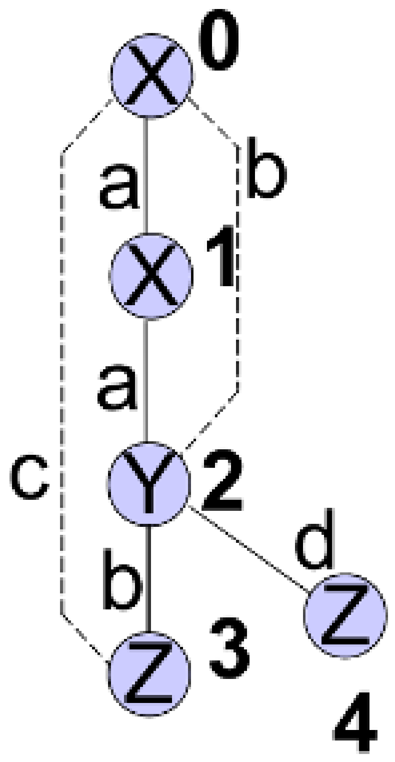
Examples of DFS codes

$<_L = \{X < Y < Z : a < b < c\}$ C < A < B

0	(0,1,X,a,Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	(1,2,X,a,X)	(1,2,X,a,Y)
2	(2,0,X,a,X)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	(3,1,Z,b,X)	(3,0,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)



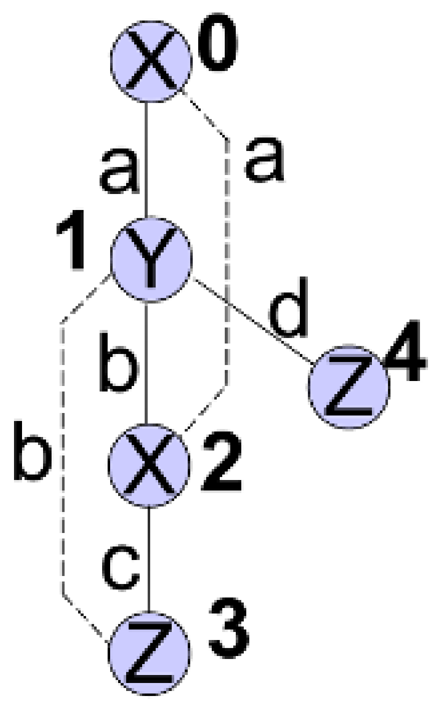


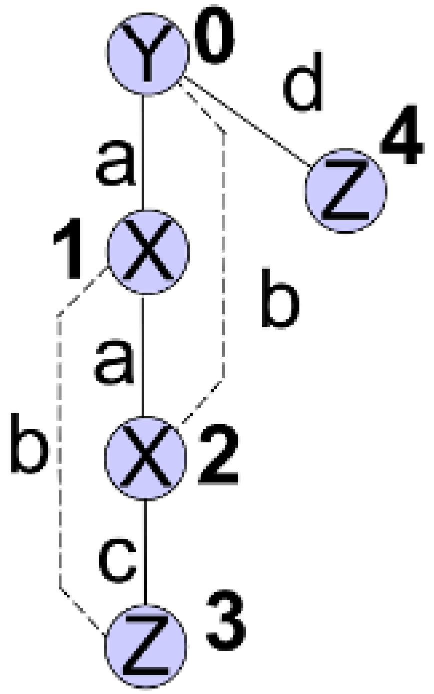


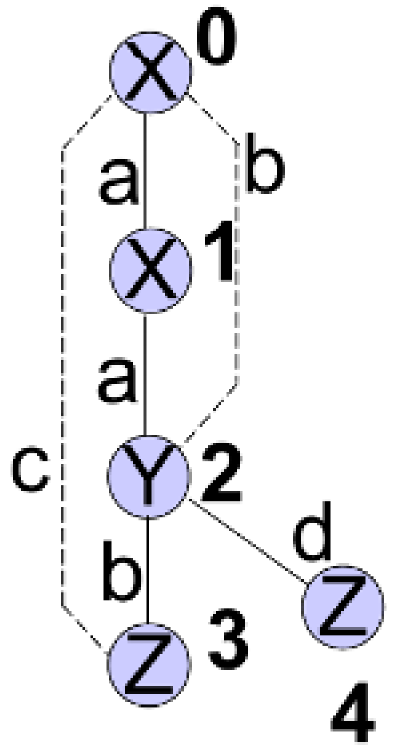
Examples of DFS codes

$<_L = \{X = Y = Z : b < c < a\}$ A < C < B

0	(0,1,X,a,Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	(1,2,X,a,X)	(1,2,X,a,Y)
2	(2,0,X,a,X)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	(3,1,Z,b,X)	(3,0,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)



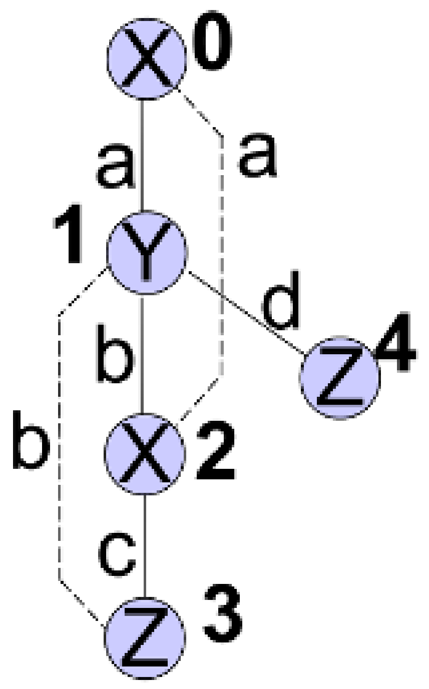


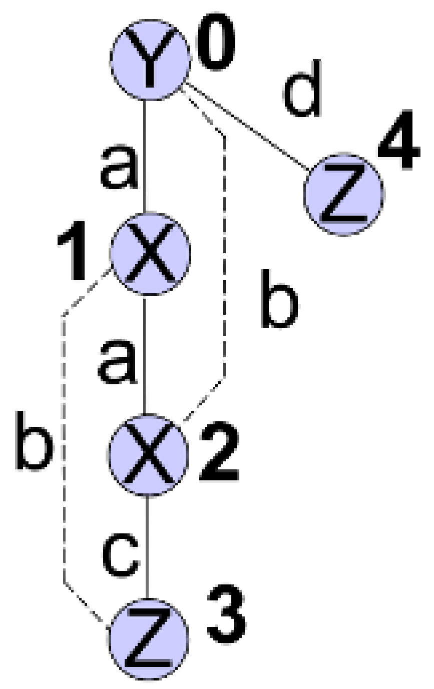


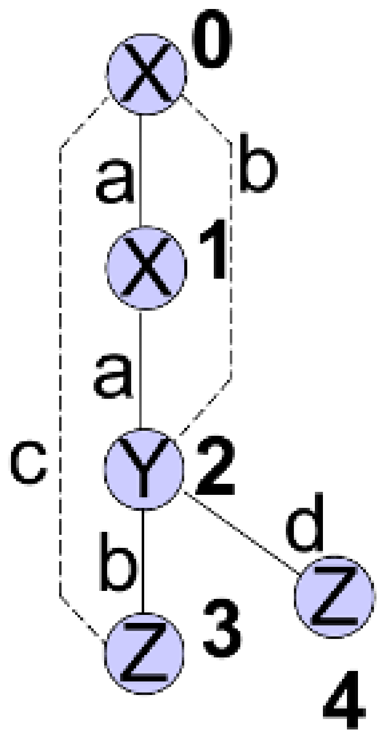
Example of DFS codes

$<_L = \{X = Y = Z : b = c = a\}$ C < A < B

0	(0,1,X,a,Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	(1,2,X,a,X)	(1,2,X,a,Y)
2	(2,0,X,a,X)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	(3,1,Z,b,X)	(3,0,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)







Tính chất mã DFS tối thiểu

- Gọi mã DFS tối thiểu của một đồ thị là $\min\_DFS(G)$, hai đồ thị A và B là đẳng cấu nếu và chỉ nếu thỏa:

$$\min\_DFS(A) = \min\_DFS(B)$$

Parenthood in the DFS code tree

- If $\min\_DFS(G_1) = \{a_0, a_1, \ldots, a_n\}$

  and $\min\_DFS(G_2) = \{a_0, a_1, \ldots, a_n, b\}$ then

  - $G_1 is\ called\ the\ father\ of\ G_2$
  - G_2 is called the child of G_1

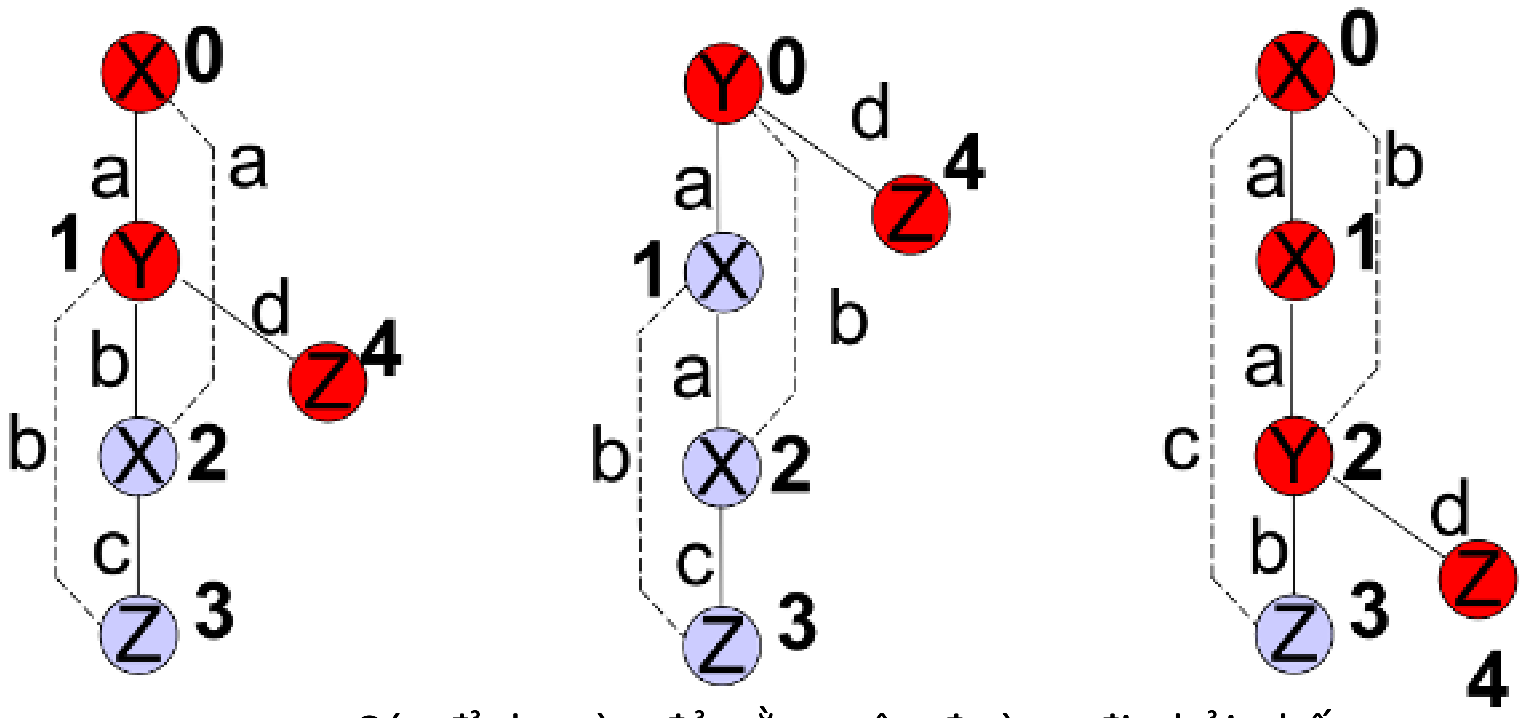
$$\begin{tikzcd}
\begin{array}{l}(0,1,X,a,Y)\\(1,2,Y,b,X)\end{array} \arrow[r] & \begin{array}{l}(0,1,X,a,Y)\\(1,2,Y,b,X)\\(2,0,X,a,X)\end{array} \arrow[r] & \begin{array}{l}(0,1,X,a,Y)\\(1,2,Y,b,X)\\(2,0,X,a,X)\\(2,3,X,c,Z)\end{array}
\end{tikzcd}$$

Parenthood in the DFS code tree

- Since b needs to be greater than all a_i, a valid DFS code requires b to grow from a vertex on the right path (rightmost path)
  - If extending the forward edge to a DFS code, it must occur from a vertex on the most right path.
  - If the backward edge is extended to a DFS code, it must occur from the rightmost vertex.

Rightmost path

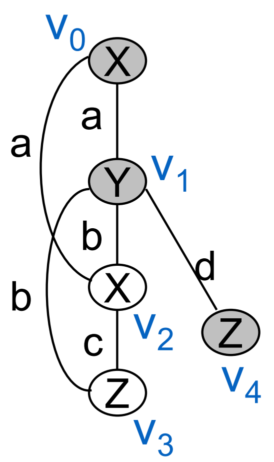
- For a vertex order to be visited: $(v_0, v_1, \ldots, v_n)$
  - v_0: is the first peak to be visited
  - v_n: is the last peak visited (the most right-hand peak).
- Rightmost path is the shortest path between v_0 and v_n using only the incoming edges.



Các đỉnh màu đỏ nằm trên đường đi phải nhất

Rightmost path

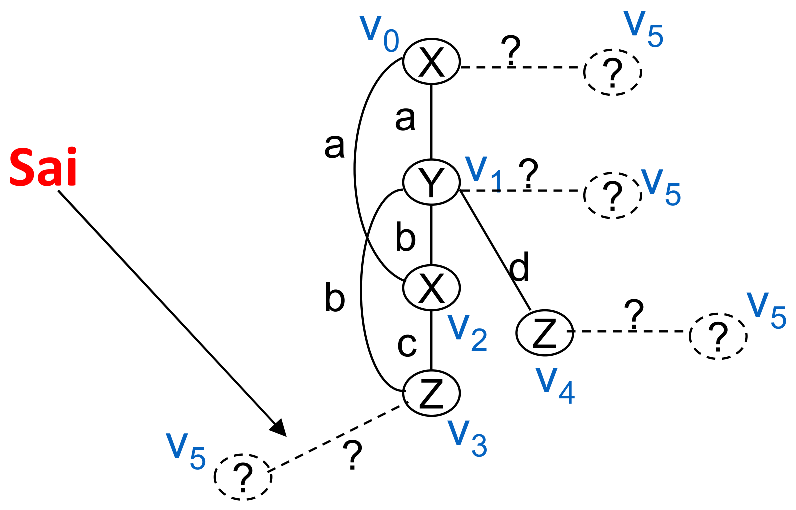
Graph G_1:



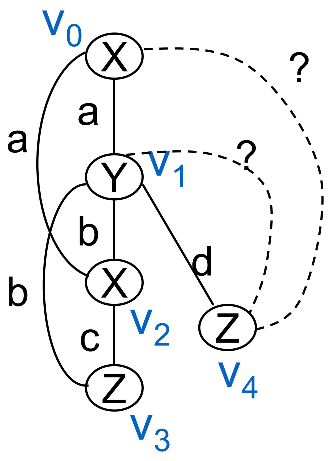
Min(g) = (0,1,X,a,Y) (1,2,Y,b,X) (2,0,X,a,X) (2,3,X,c,Z) (3,1,Z,b,Y) (1,4,Y,d,Z)

Cần phát triển thêm cạnh vào G_1?

Graph G_2:

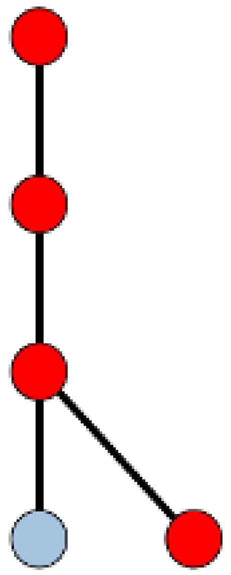


Cạnh tới

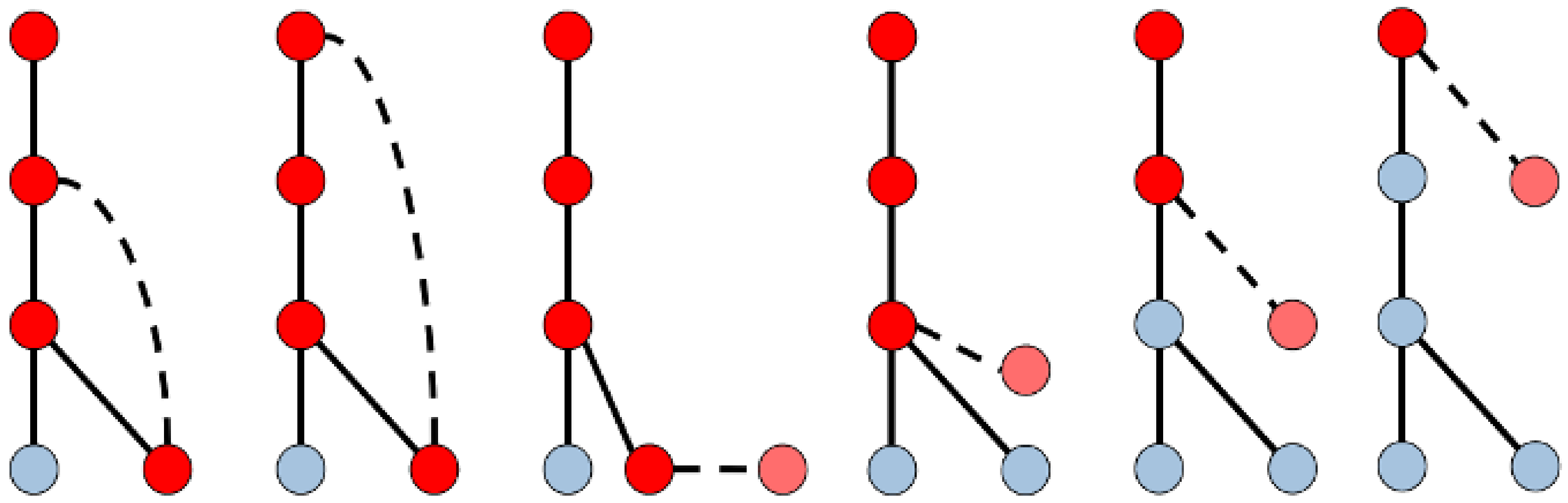


Cạnh lui

Tree development example

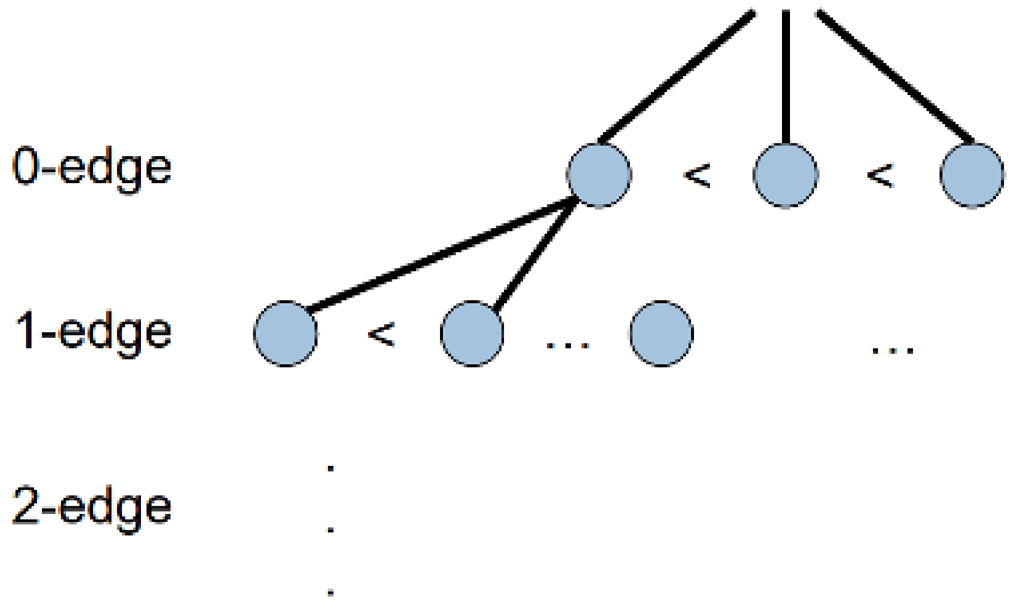


● Rightmost path vertex

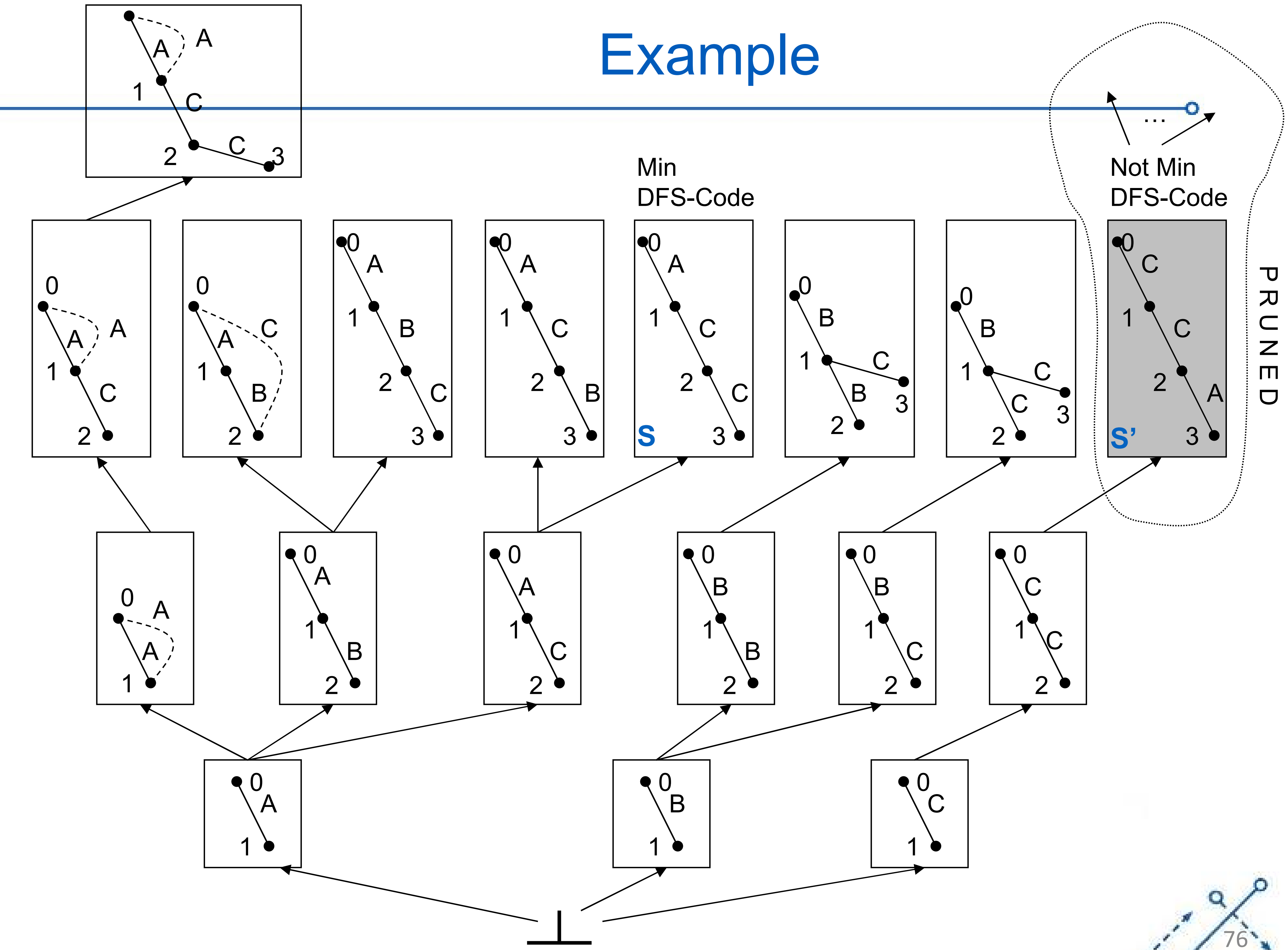


Build a DFS code tree

- Organize DFS code vertices according to the parent-child law.
- The vertices are DFS codes except...
  - The first level of the tree is one peak for each peak label.
- Each level of the tree adds an edge to the DFS code.
- The fraternal vertices are organized in ascending DFS code order.
- Inorder (LNR) tree browsing will follow DFS order.
- The backward edge must be added first.



Example



gSpan algorithm

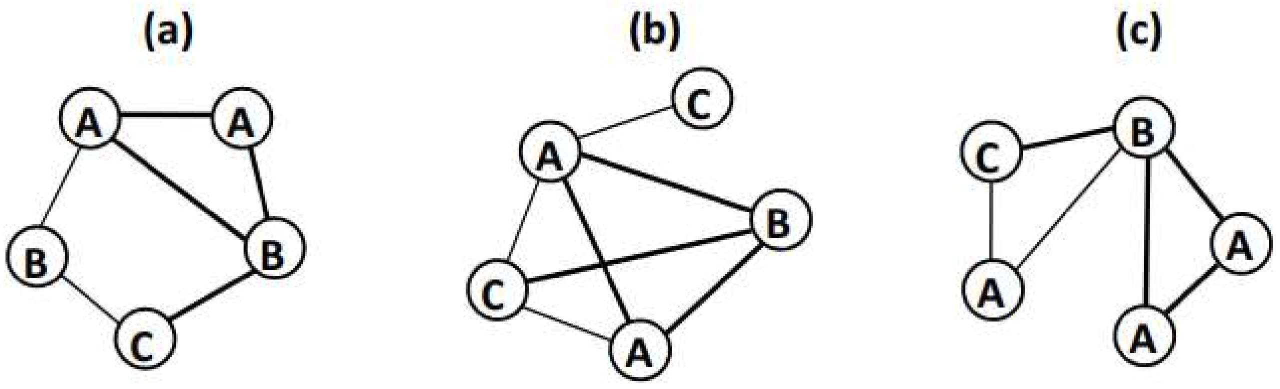
- Input: graph database D, min_sup
- Output: common set of subgraphs S
- Process:
  - Remove uncommon vertices and edges
  - S ← common single-edge subgraphs in D (use DFS code)
  - Sort S in order
  - N← S (because S will be updated)
  - For every n∈N taken:
    - gSpan_Extend(D,n,min_sup,S)
    - Remove n from all graphs in D

gSpan_extend function

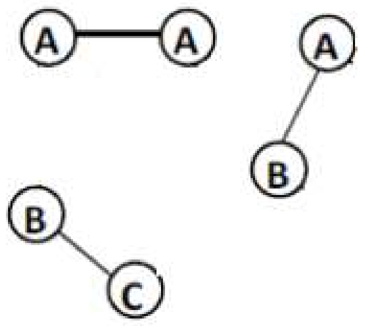
- Input: graph database D, code DFS n, min_sup
- Input/Output: common subgraph set S
- Process:
  - If n is not the smallest then stop
  - Conversely:
    - Add n to S
    - For each extension 1 the most right edge of n(e)
      - If support(e) >= min_sup
        - Call recursively gSpan_extend(D,e, min_sup, S)

Example

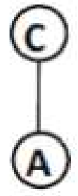
- Input: min_sup = 3
- Graphs without edge labels



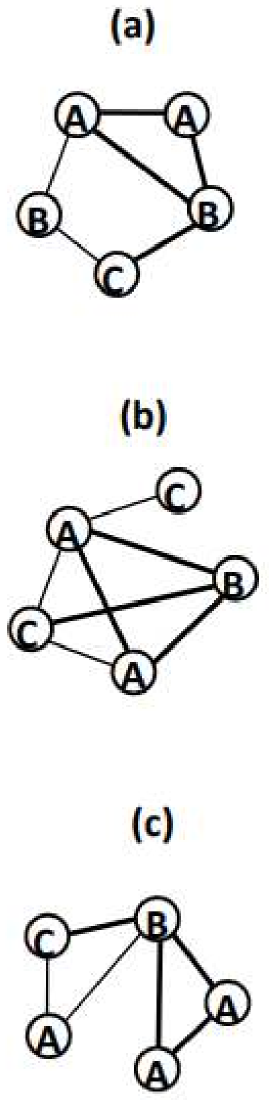
Frequent:



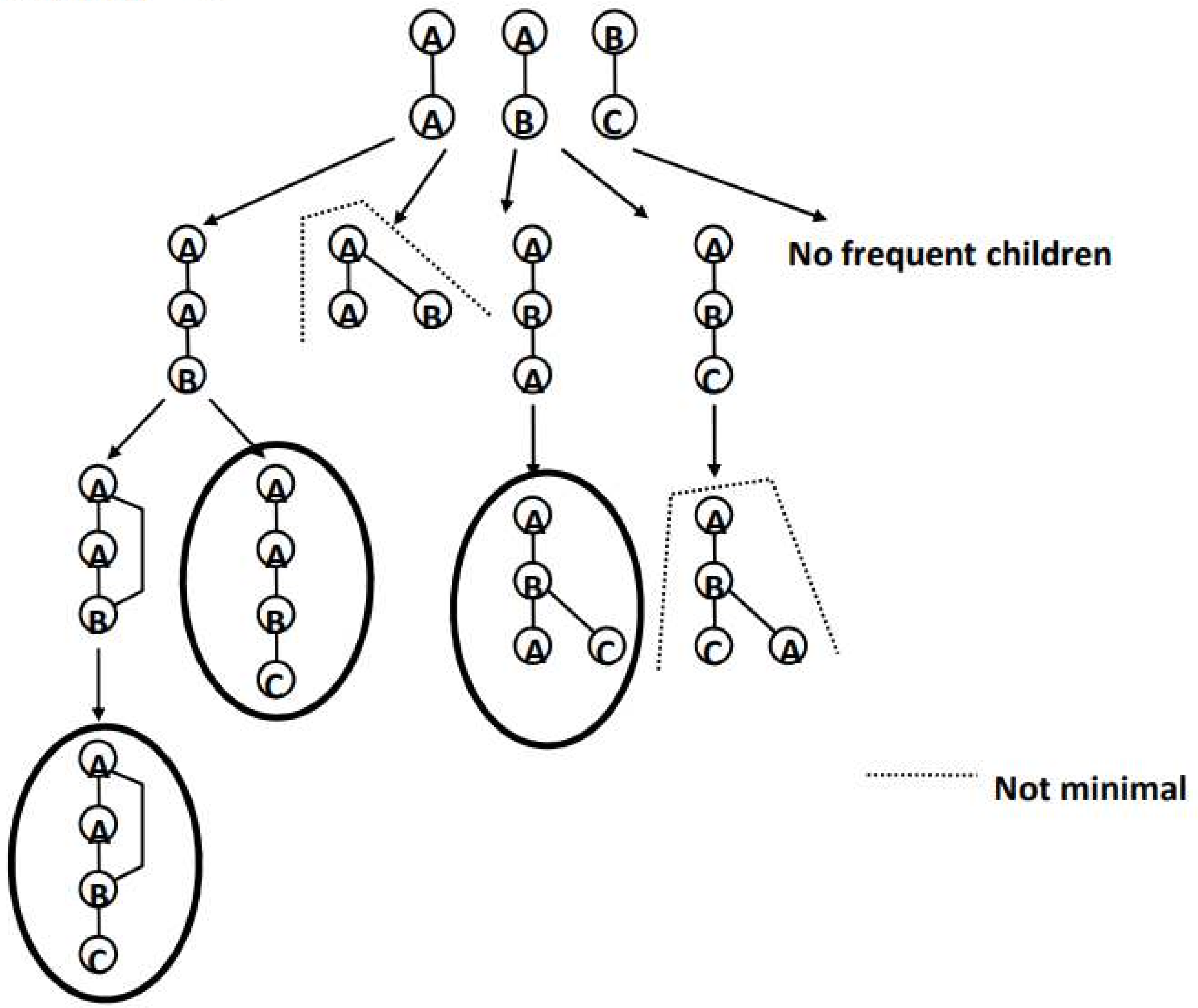
Infrequent:



Example



min support = 3



Nội dung

- Mẫu đồ thị con phổ biến
- Phương pháp tìm mẫu phổ biến dựa trên Apriori
- Phương pháp tìm mẫu phổ biến dựa trên chiều sâu
- **Phương pháp tìm mẫu phổ biến tham lam**

Graph pattern explosion problem

- If a graph is universal, all its subgraphs must be common – Apriori property
- An n-sided common graph can have 2n subgraphs.
- Of the 422 chemical compounds confirmed to be active in the AIDS antiviral screening dataset, 1,000,000 graph samples were common if the aid level was at least 5%.

Subdue algorithm

- Is a greedy algorithm to find some of the most popular subgraphs.
- This method is incomplete, i.e. it may not find all common subgraphs, but the execution process is fast.
- Based on description length: compress graphs using graph templates
  - Use that template for maximum compression.
- Based on Beam Search – like BFS, it progresses in levels. However, unlike BFS, the ray search moves down only through the best node W at each level. Other buttons are ignored.

Reference

- https://hpi.de/fileadmin/user_upload/fachgebiete/mueller/courses/graphmining/GraphMining-04-FrequentSubgraph.pdf
- https://www.cs.helsinki.fi/u/langohr/graphmining/slides/chp2a.pdf
- https://www.csc2.ncsu.edu/faculty/nfsamato/practical-graph-mining-with-R/slides/pdf/Frequent_Subgraph_Mining.pdf
- https://hpi.de/fileadmin/user_upload/fachgebiete/mueller/courses/graphmining/GraphMining-04-FrequentSubgraph.pdf