Khai Thác Dữ Liệu Đồ Thị

KHAI THÁC MẪU ĐỒ THỊ PHỔ BIẾN

Giảng viên: Lê Ngọc Thành

Email: lnthanh@fit.hcmus.edu.vn

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP. HỒ CHÍ MINH

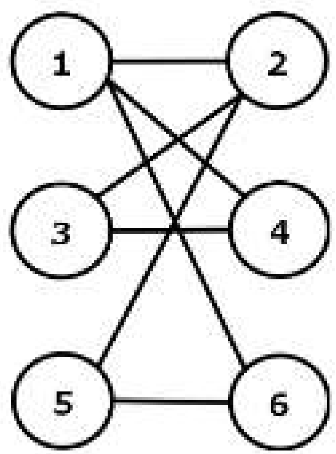
fit@hcmus

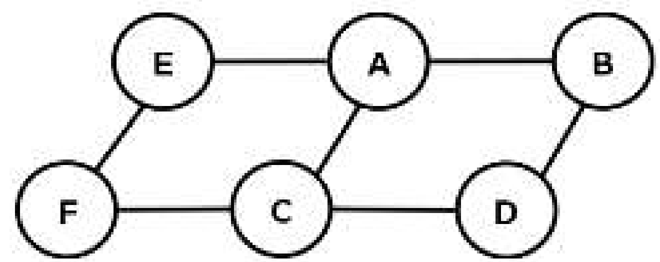
Nội dung

- **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

Đồ thị con và đẳng cấu đồ thị

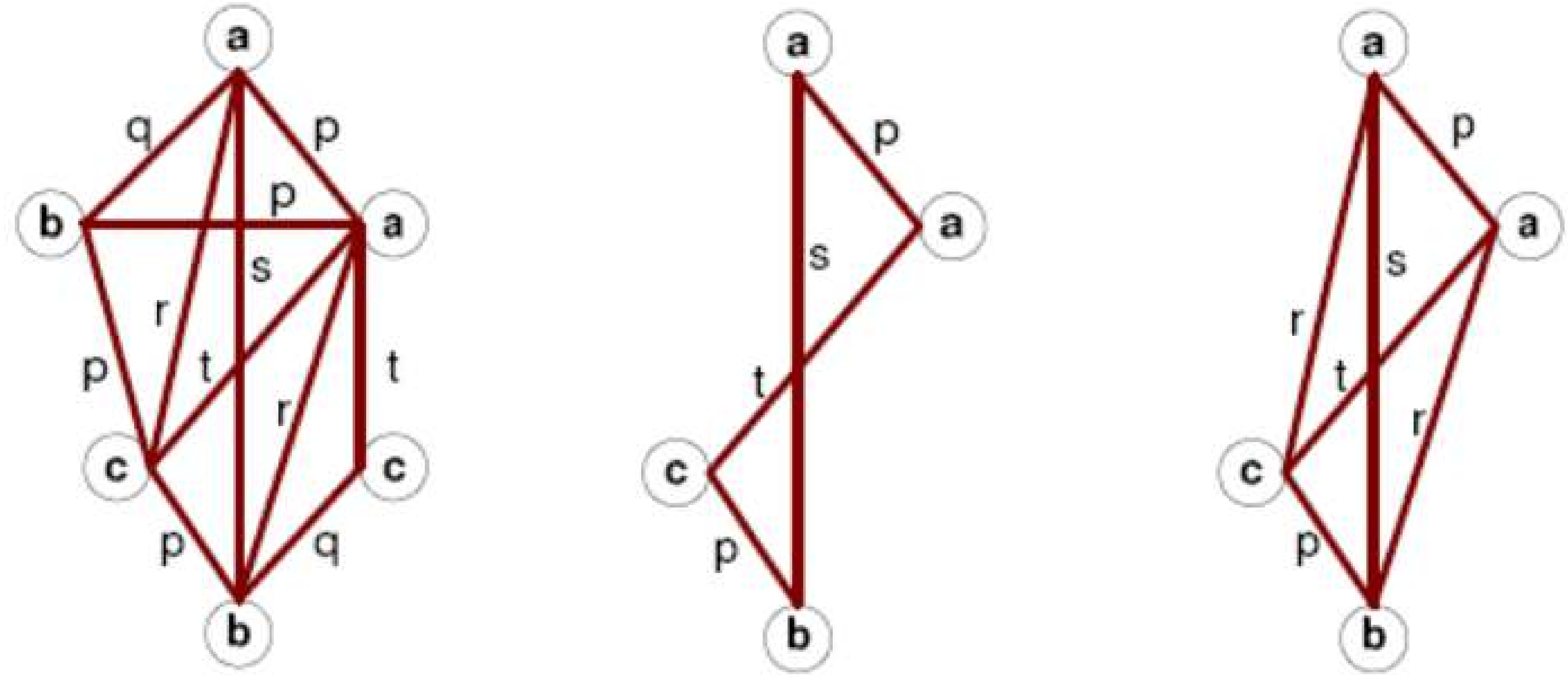
- Một đồ thị $G_S(V_S, E_S)$ là một đồ thị con của đồ thị $G(V, E)$ nếu:
  - V_S là tập đỉnh của V và E_S là tập cạnh con của E
- Hai đồ thị $G_1(V_1, E_1)$ và $G_2(V_2, E_2)$ được gọi là đẳng cấu (isomorphic) nếu chúng giống nhau đồ hình
  - Nghĩa là có một cách ánh xạ từ V_1 đến V_2 sao cho mỗi cạnh trong E_1 tương ứng với một cạnh trong E_2 và ngược lại.





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

Ví dụ đồ thị con



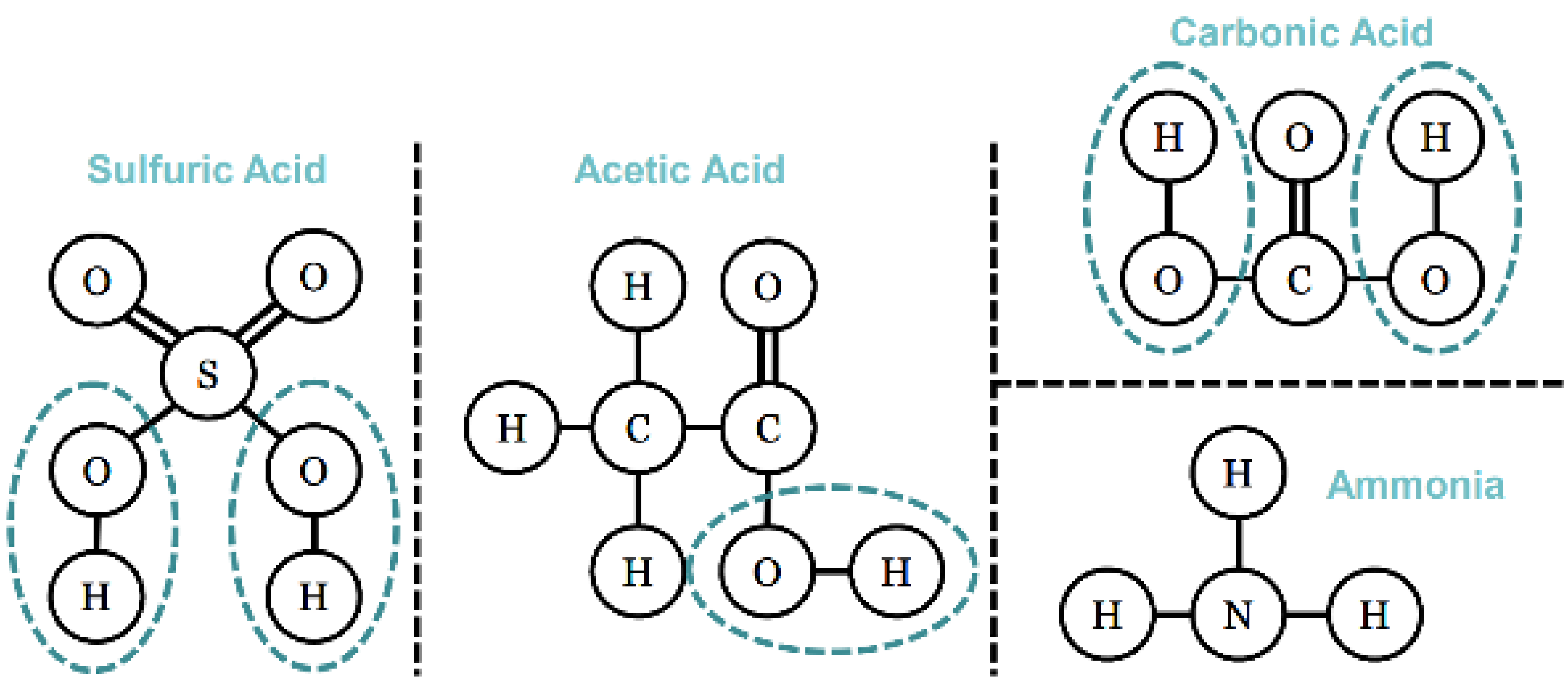
(a) Labeled Graph (b) Subgraph (c) **Subgraph**

Mẫu đồ thị con phổ biến

- Mẫu đồ thị con phổ biến (frequent subgraph pattern): một cấu trúc đồ thị con xảy ra thường xuyên trong một tập các đồ thị cho trước.

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

  – Ví dụ: tìm các kết nối thường xuyên xuất hiện trong các loại axit



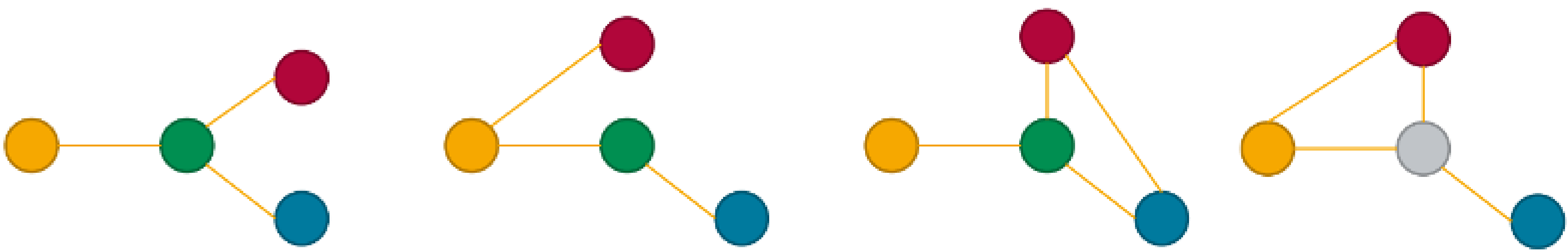
O-H xảy ra 3 trong số 4 dữ liệu → phổ biến nếu độ trợ <= 3

Mẫu đồ thị phổ biến

- Ứng dụng khai thác mẫu đồ thị phổ biến:
  - Tìm con đường sinh học (biological pathway) chung giữa các loài.
  - Phân tích cộng đồng trong mạng xã hội
  - Xây dựng các khối cho phân lớp đồ thị, gom nhóm, nén, so sánh hay phân tích tương quan.

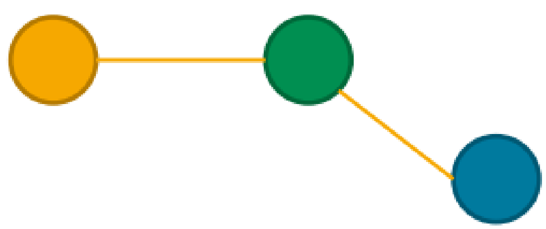
Ví dụ khai thác mẫu đồ thị

- Xác định đồ thị con của G xuất hiện ít nhất $\text{minsup} = 3$ lần



G1 G2 G3 G4

Đồ thị con phổ biến với minsup=3: => Mỗi đồ thị là 1 Transaction



Mẫu đồ thị phổ biến

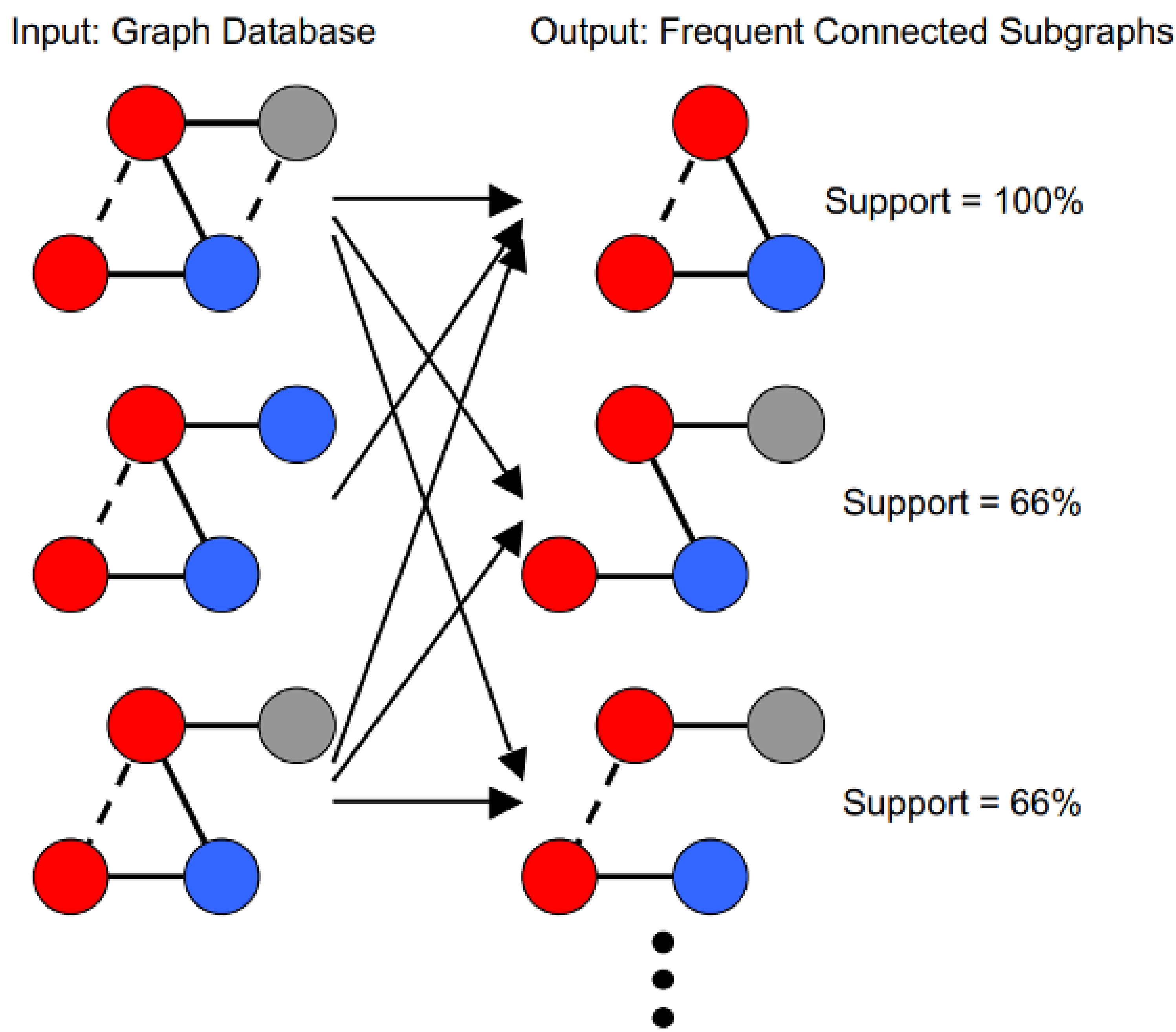
- Khi xem xét một mẫu đồ thị, ta không xem xét số lần mẫu này xuất hiện lặp lại trong đồ thị (giống trong khai thác mẫu phổ biến, trong mỗi giao tác, các hạng mục chỉ được tính một lần).
- Tuy nhiên, một bài toán khác là tìm mẫu xuất hiện phổ biến trong một đồ thị.

→ Trong nội dung phần này, ta không xem xét dạng mẫu phổ biến trên.

Phát biểu bài toán

- Cho trước một tập các đồ thị được đánh nhãn $D = \{G_1, G_2, \dots, G_n\}$ và một đồ thị con G:
  - Tập trợ của G là $D_G = \{G_i | G \sqsubseteq G_i, G_i \in D\}$ trong đó ký hiệu $G \sqsubseteq G_i$ ám chỉ G là đẳng cấu con của G_i
  - Độ trợ được tính $\sigma(G) = \frac{|D_G|}{|D|}$ Số đồ thị chứa G / Tổng số đồ thị
- Đầu vào:
  - Tập các đồ thị D
  - Giá trị minsup
- Đầu ra:
  - Các đồ thị con liên thông phổ biến

Phát biểu bài toán



Các thách thức

- Việc tìm mẫu đồ thị phổ biến thường phải trải qua bước xác định đồ thị con đẳng cấu.
  - Nghĩa là xác định một đồ thị có chứa một đồ thị con đẳng cấu với đồ thị khác.
- Đây được xếp vào loại bài toán NP-Complete
  - Đòi hỏi chạy với thời gian mũ
- Thuật toán khai thác đồ thị con phổ biến hiệu quả phải giảm không gian tìm kiếm bằng cách giảm số kiểm tra đẳng cấu đồ thị con.

Các thuật toán khai thác

- Các thuật toán khai thác đồ thị con phổ biến được phân chia thành nhiều nhóm:
  - Phương pháp dựa trên lý thuyết đồ thị:
    - Phương pháp dựa trên chiều rộng hay Apriori: AGM/AcGM, FSG, gFSG, PATH#, FFSM
    - Phương pháp dựa trên chiều sâu hoặc phát triển mẫu (pattern-growth): MoFa, gSpan, Gaston
  - Phương pháp tham lam (greedy): Subdue
  - Phương pháp dựa trên suy diễn logic: WARMR

Nội dung

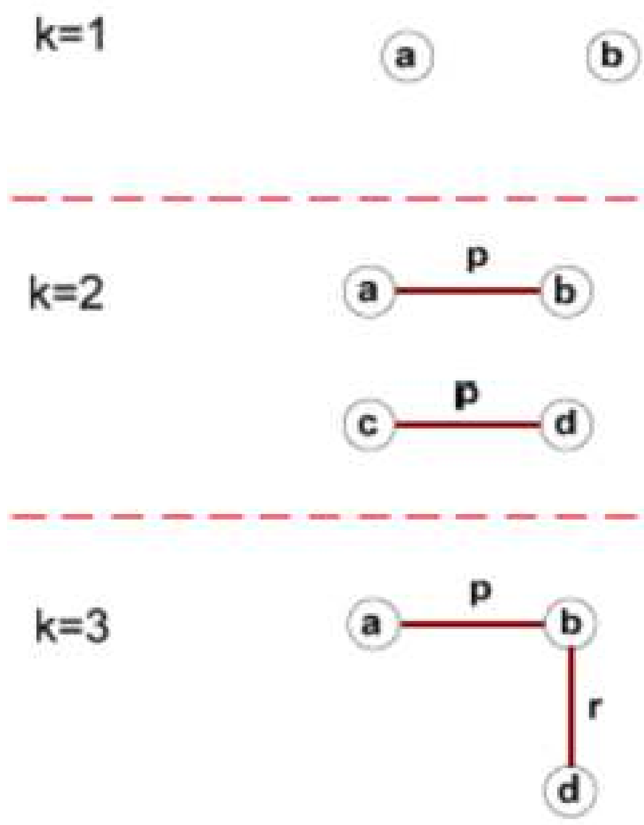
- Mẫu đồ thị con phổ biến
- **Phương pháp tìm mẫu phổ biến dựa trên Apriori**
- Phương pháp tìm mẫu phổ biến dựa trên chiều sâu
- Phương pháp tìm mẫu phổ biến tham lam

Phương pháp dựa trên Apriori

- Khái niệm:
  - Đồ thị con bậc k (k-subgraph) là đồ thị con với k đỉnh hoặc k cạnh

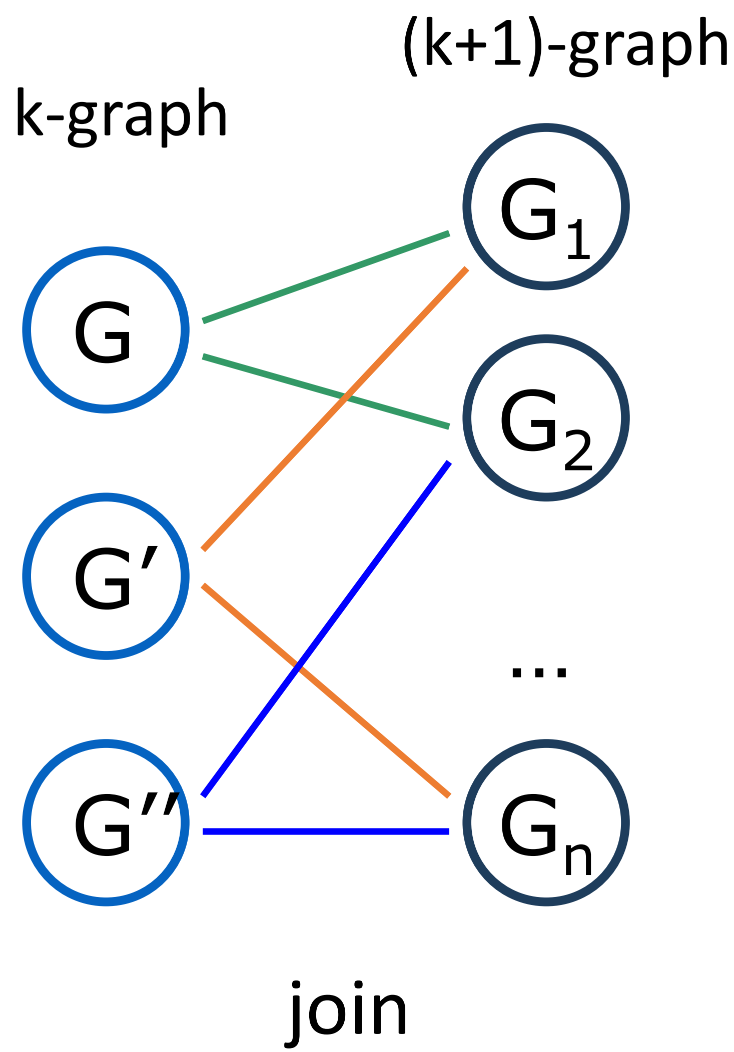
k - itemset

=> Sử dụng k-subgraph phổ biến
để sinh ra (k+1)-subgraph



Phương pháp dựa trên Apriori

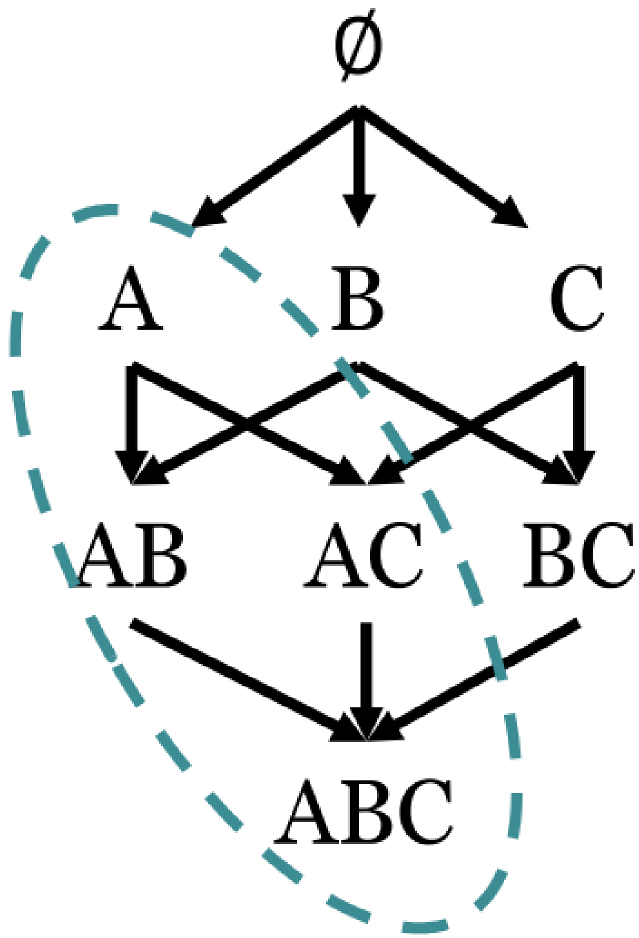
- Ý tưởng của phương pháp dựa trên Apriori là sử dụng các k-subgraph phổ biến để phát sinh (k+1)-subgraph phổ biến.



Phương pháp dựa trên Apriori

- Tính chất bao đóng hướng xuống (downward closure) trong Apriori:

Nếu A không phổ biến thì các tập cha của nó cũng không phổ biến

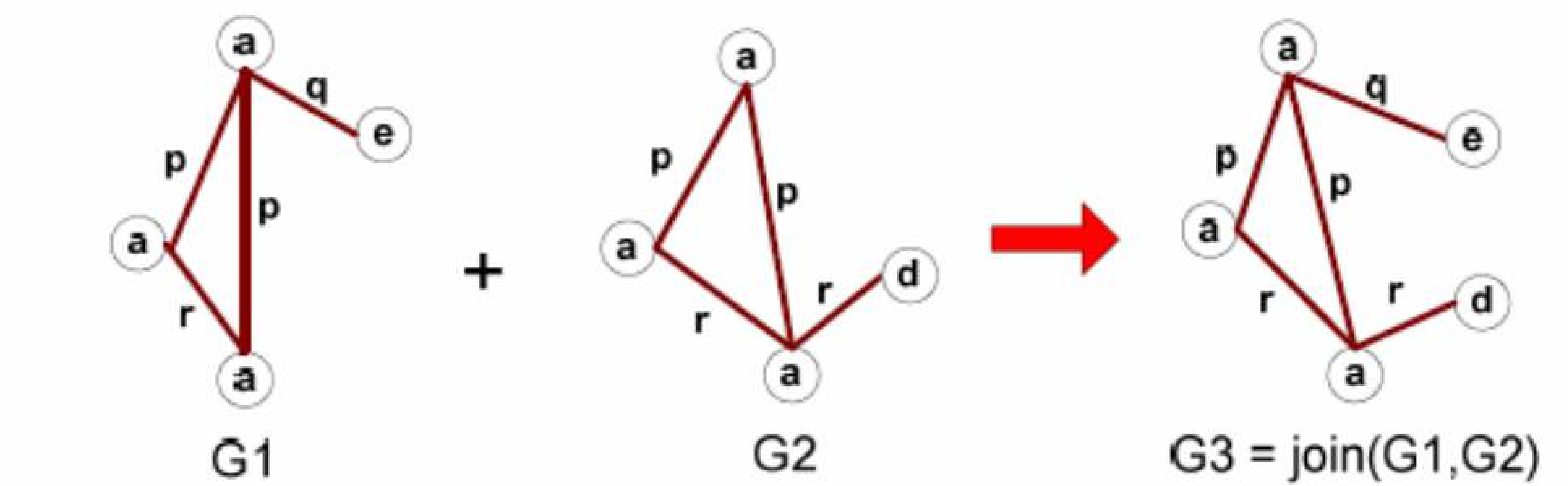


Phương pháp dựa trên Apriori

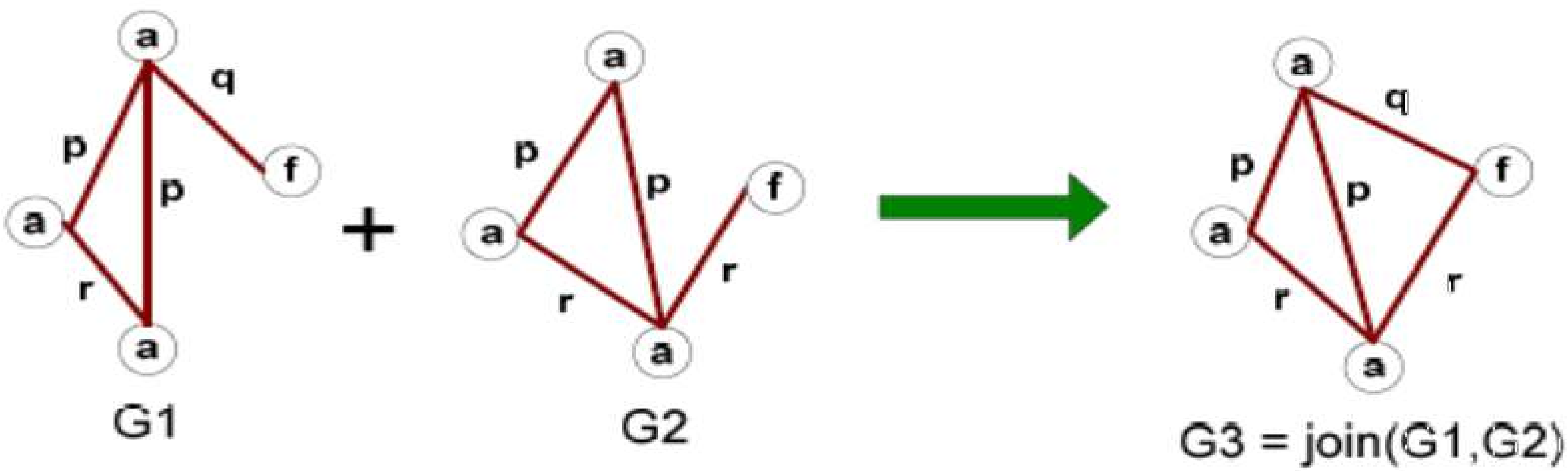
- Có hai nhánh trong phương pháp dựa trên Apriori:
  - Nhánh phát triển đỉnh (vertex growing): AGM
    - Chỉ số k sẽ là số đỉnh
  - Nhánh phát triển cạnh (edge growing): FGM
    - Chỉ số k sẽ là số cạnh

Nhánh phát triển đỉnh

=> 2 đồ thị kết hợp vs nhau đc nếu chúng giống nhau về tiền tố / subgraph (chỉ khác nhau ở 1 đỉnh or cạnh)



Nhánh phát triển cạnh



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)

- Thuật toán AGM:

=> Tất cả các cải tiến trên Apriori đều là cải tiến tốc độ chứ ko thay đổi về kết quả

Ký hiệu: k-subgraph là đồ thị con với k đỉnh

Khởi tạo: Quét dữ liệu để tìm F_1, tập tất cả 1-subgraphs phổ biến, cùng với độ trợ của chúng;

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

1. **Phát sinh ứng viên** - C_k, tập ứng viên k-subgraphs, từ F_{k-1}; ứng viên sẽ tăng 1 đỉnh so với đồ thị trước. ràng buộc để giảm số lượng trùng
2. **Tỉa nhánh** – điều kiện cần để ứng viên trở nên phổ biến là mỗi (k-1)-subgraph của nó phải phổ biến. ab+ac ra abc thì abc bị tỉa khi bc ko xuất hiện trog F (ko phổ biến)
3. **Tính độ phổ biến** – quét dữ liệu để đếm số lần đồ thị con C_k xuất hiện.
4. $F_k = \{ c \in C_K \mid c$ có độ trợ không nhỏ hơn $\#minSup \}$
5. Trả về $F_1 \cup F_2 \cup \ldots\ldots \cup F_k$ (= F)

FGM (Frequent Sub-graph Mining)

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

$F^1 \leftarrow$ detect all frequent 1-subgraphs in D
$F^2 \leftarrow$ detect all frequent 2-subgraphs in D
$k \leftarrow 3$
while $F^{k-1} \neq \emptyset$ **do**
    $C^k \leftarrow$ fsg-gen(F^{k-1})
    for each candidate $G^k \in C^k$ **do**
        G^k.count $\leftarrow 0$
        for each transaction $T \in D$ **do**
            if candidate G^k is included in transaction T **then**
                G^k.count $\leftarrow G^k$.count $+ 1$
    $F^k \leftarrow \{G^k \in C^k \mid G^k.\text{count} \geq \sigma|D|\}$
    $k \leftarrow k + 1$
return $F^1, F^2, \ldots, F^{k-2}$

FGM (Frequent Sub-graph Mining)

- Tương tự như AGM nhưng quan tâm đến cạnh

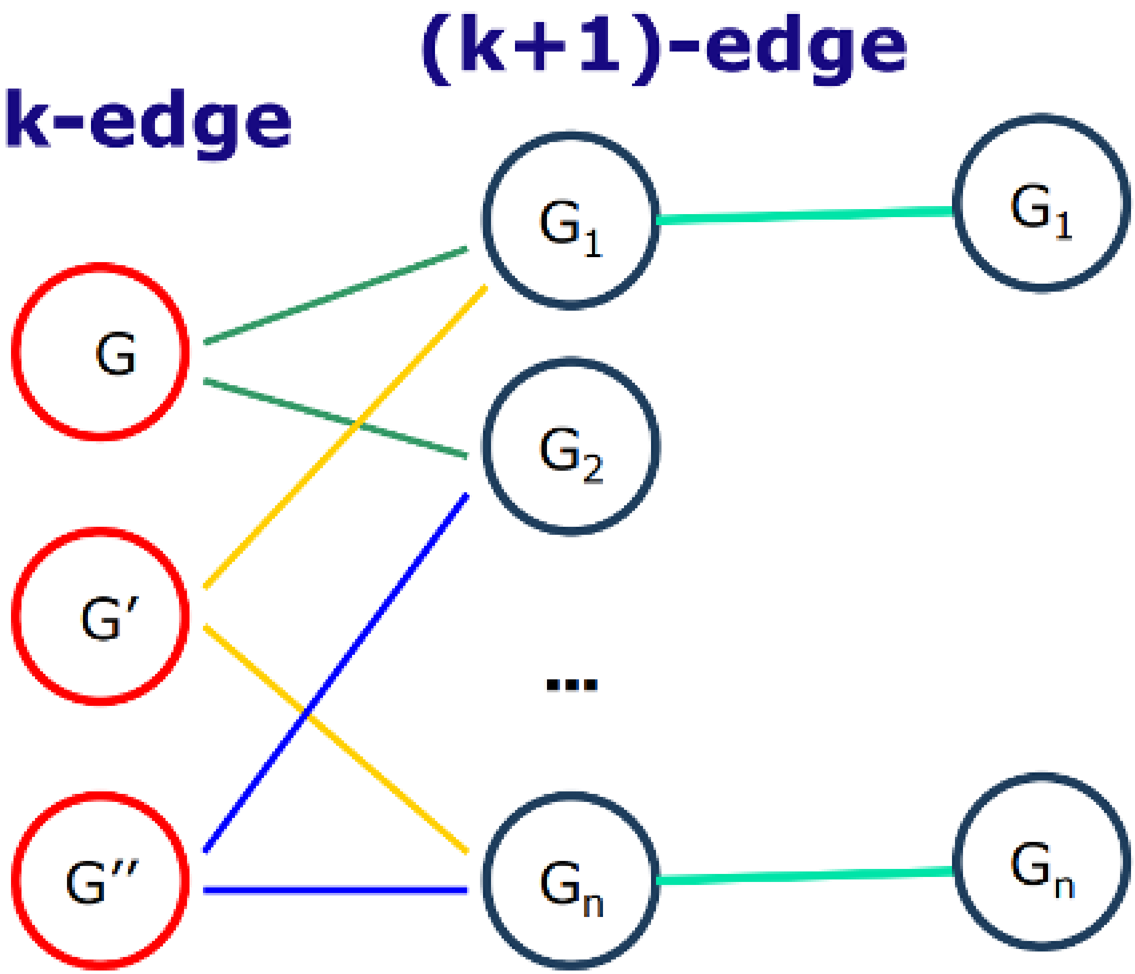
Ký hiệu: k-subgraph là đồ thị con với k cạnh

Khởi tạo: Quét dữ liệu để tìm F_1, tập tất cả 1-subgraphs phổ biến, cùng với độ trợ của chúng;

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

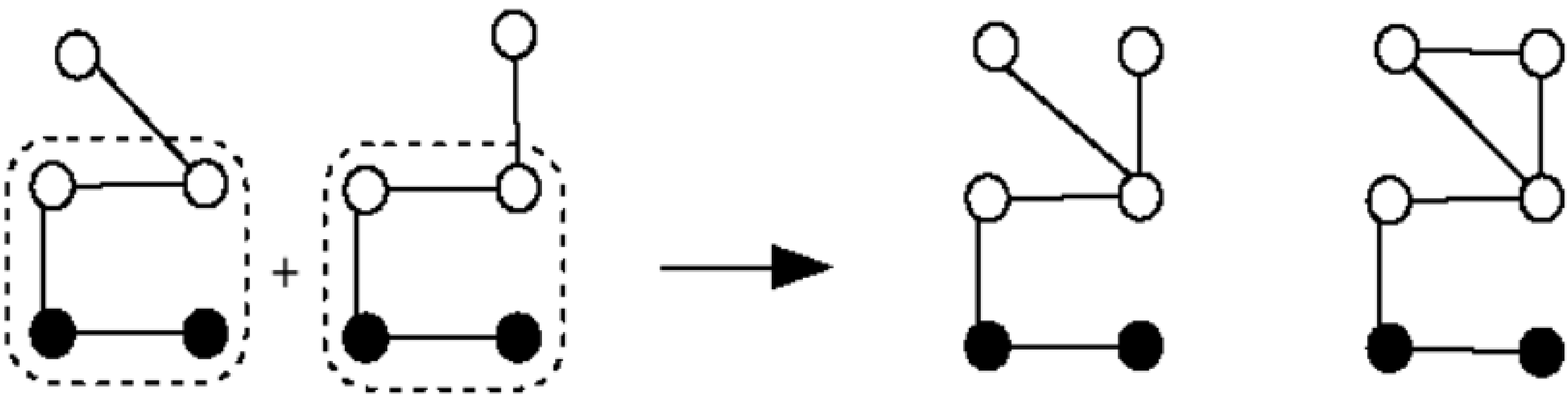
1. **Phát sinh ứng viên** - C_k, tập ứng viên k-subgraphs, từ F_{k-1}; ứng viên sẽ tăng 1 cạnh so với đồ thị trước.
2. **Tỉa nhánh** – điều kiện cần để ứng viên trở nên phổ biến là mỗi (k-1)-subgraph của nó phải phổ biến.
3. **Tính độ phổ biến** – quét dữ liệu để đếm số lần đồ thị con C_k xuất hiện.
4. $F_k = \{ c \in C_K \mid c$ có độ trợ không nhỏ hơn *#minSup* $\}$
5. Trả về $F_1 \cup F_2 \cup \ldots\ldots \cup F_k$ (= F)

AGM và FGM



AGM và FGM

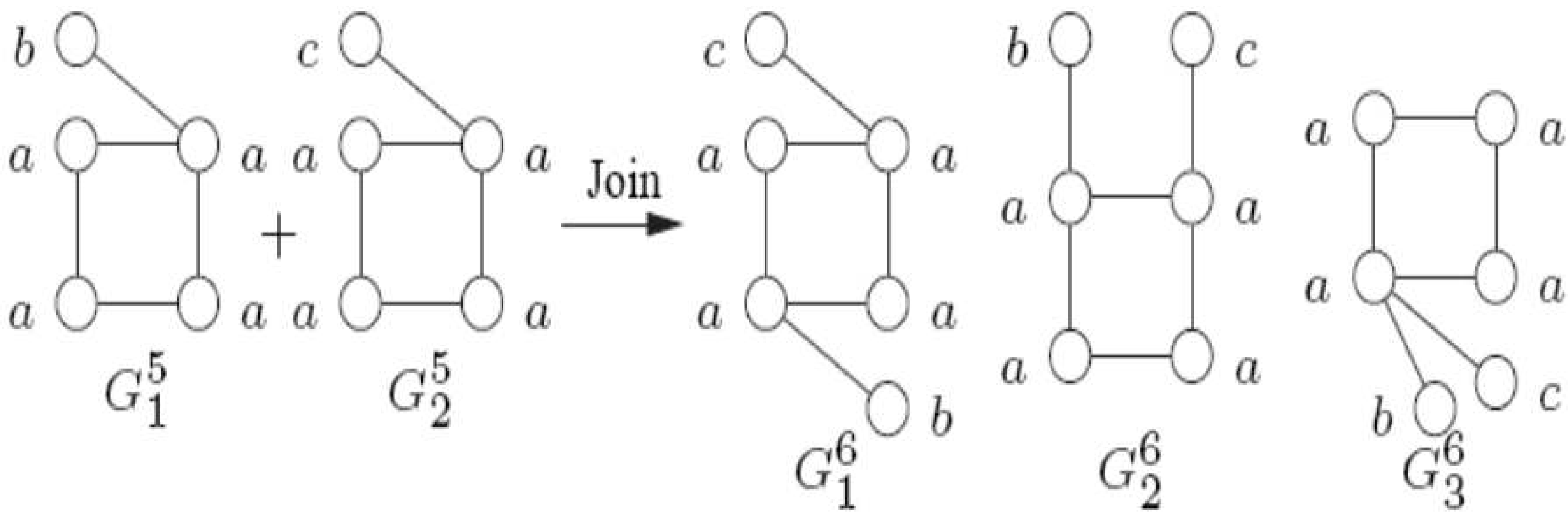
- Hàm phát sinh ứng viên:
  - Hai đồ thị phổ biến kích thước k được gia nhập chỉ nếu chúng có cùng đồ thị con kích thước k-1.
  - Điểm khác so với Apriori khi làm trên đồ thị là việc gia nhập hai đồ thị có thể tạo ra nhiều hơn hai ứng viên.



Đồ thị được gia nhập trong AGM

AGM và FGM

- Hàm phát sinh ứng viên:
  - Hai đồ thị phổ biến kích thước k được gia nhập chỉ nếu chúng có cùng đồ thị con kích thước k-1 (đồ thị lõi (core)).
  - Điểm khác so với Apriori khi làm trên đồ thị là việc gia nhập hai đồ thị có thể tạo ra nhiều hơn hai ứng viên.



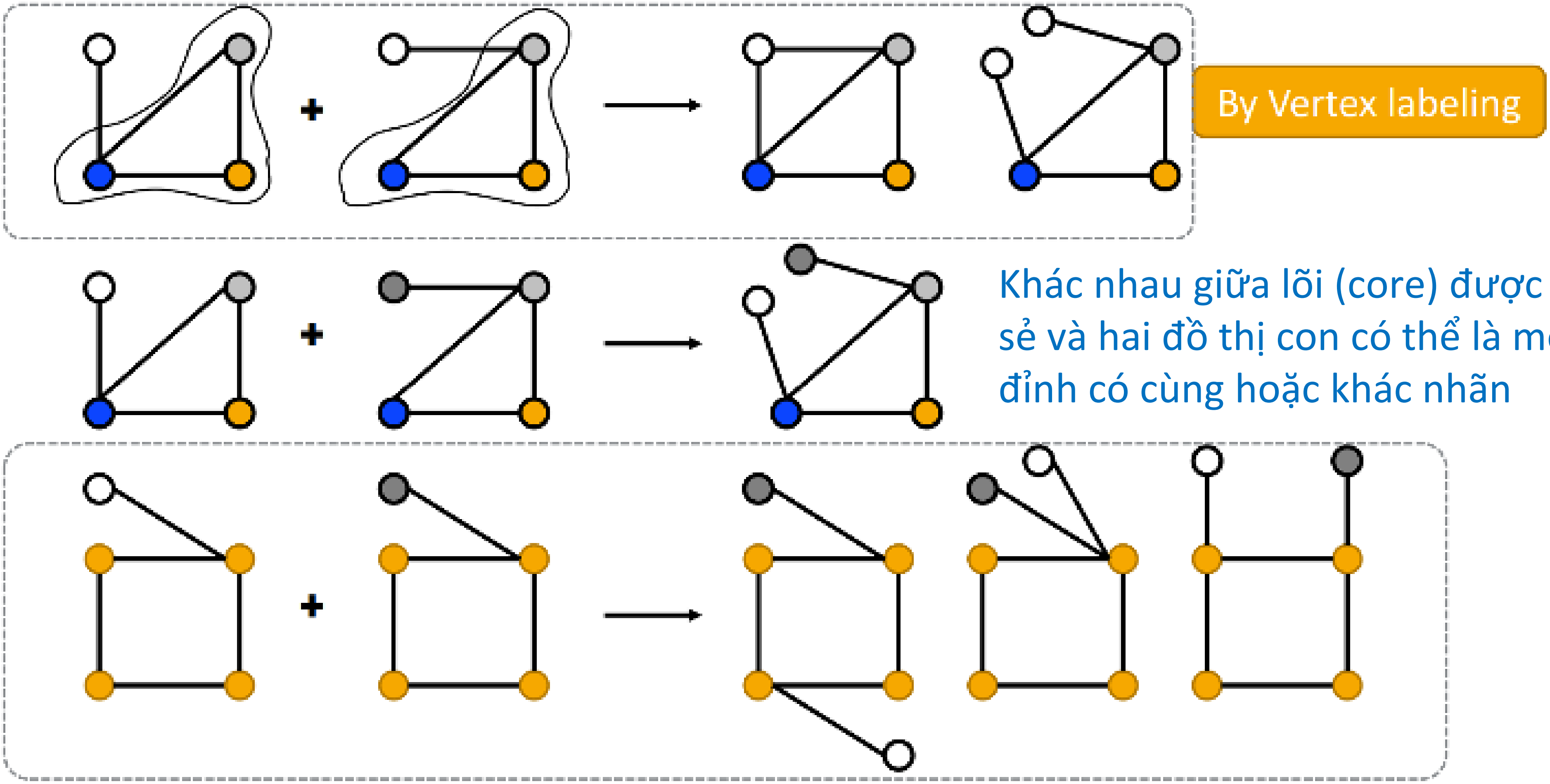
Đồ thị được gia nhập trong FGM

Xác định lõi

=> Là xác định tiền tố

- Lõi giữa hai đồ thị G_i^k và G_j^k được xác định bằng cách:
  - Tạo đồ thị con (k-1)-subgraph của đồ thị G_i^k bằng cách bỏ đi một cạnh
  - Kiểm tra đồ thị con này có phải là đồ thị con của G_j^k không
  - Lặp lại quá trình trên để có những lõi khác (nếu có)

Phát sinh ứng viên dựa trên phát hiện lõi

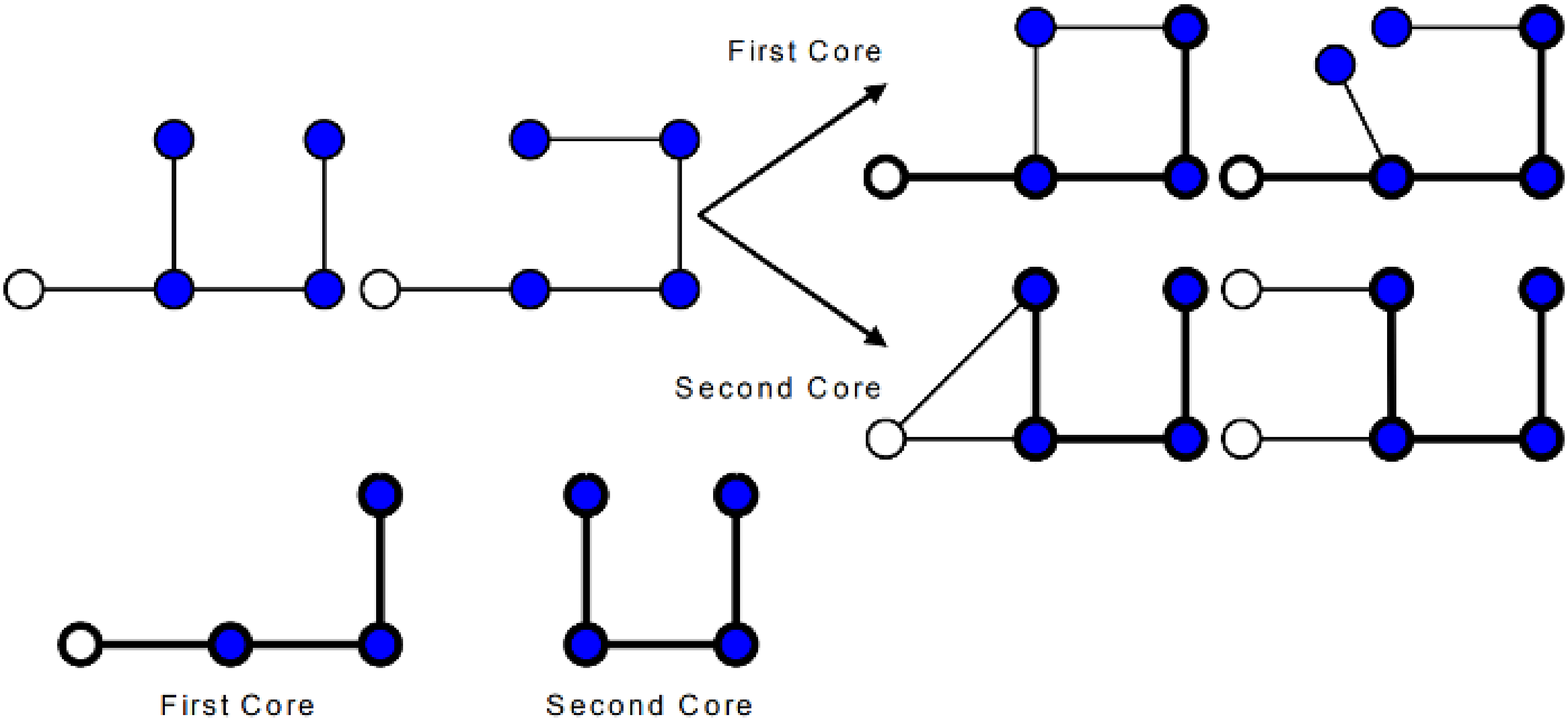


Khác nhau giữa lõi (core) được chia sẻ và hai đồ thị con có thể là một đỉnh có cùng hoặc khác nhãn

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

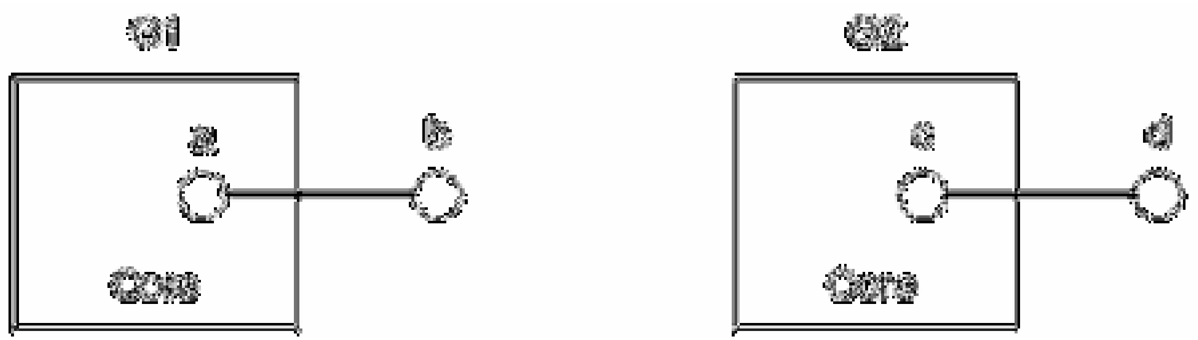
Phát sinh ứng viên dựa trên phát hiện lõi

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

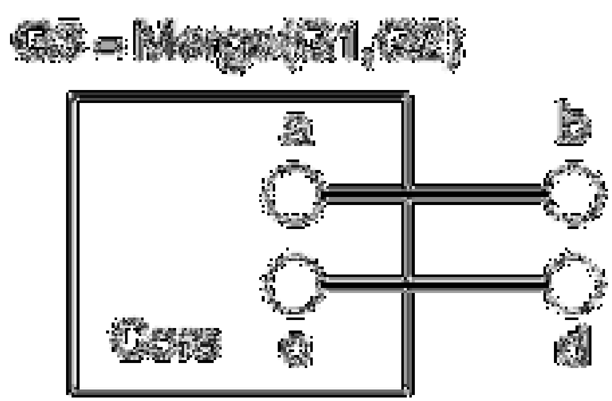


Tổng quát cho hàm phát sinh trong FGM

- Cho hai đồ thị:

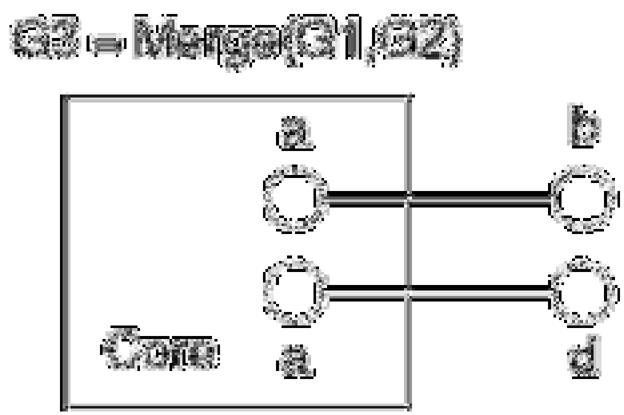


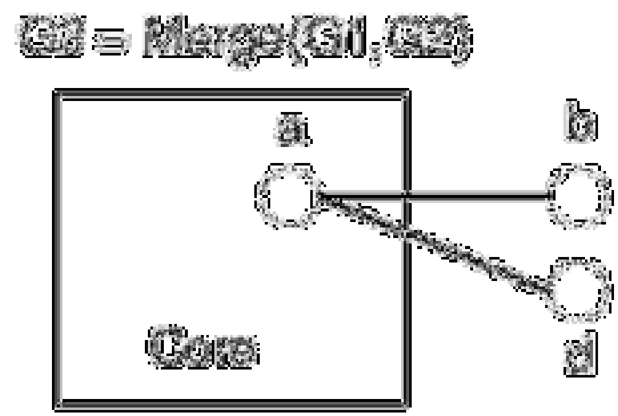
- Trường hợp 1: $a \neq c$ và b $\neq d$



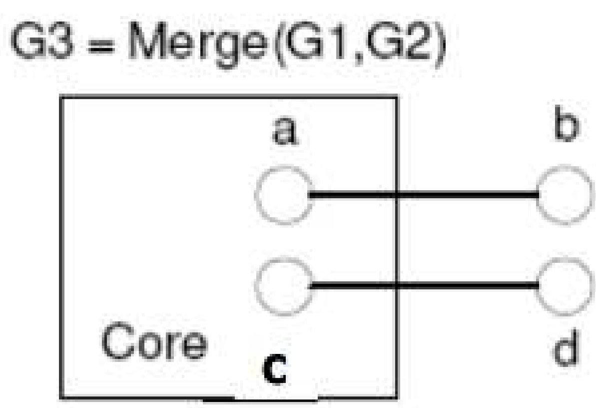
Tổng quát cho hàm phát sinh trong FGM

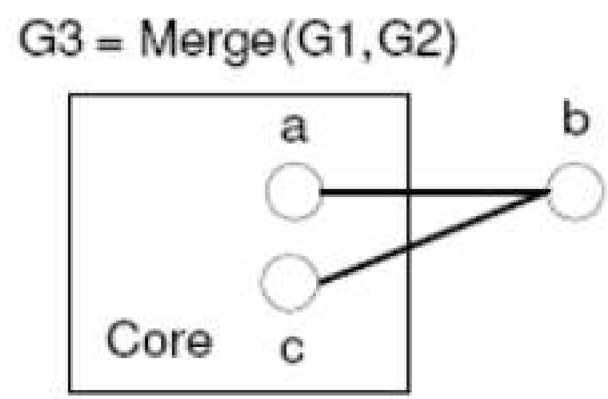
- Trường hợp 2: $a = c$ và b $\neq d$





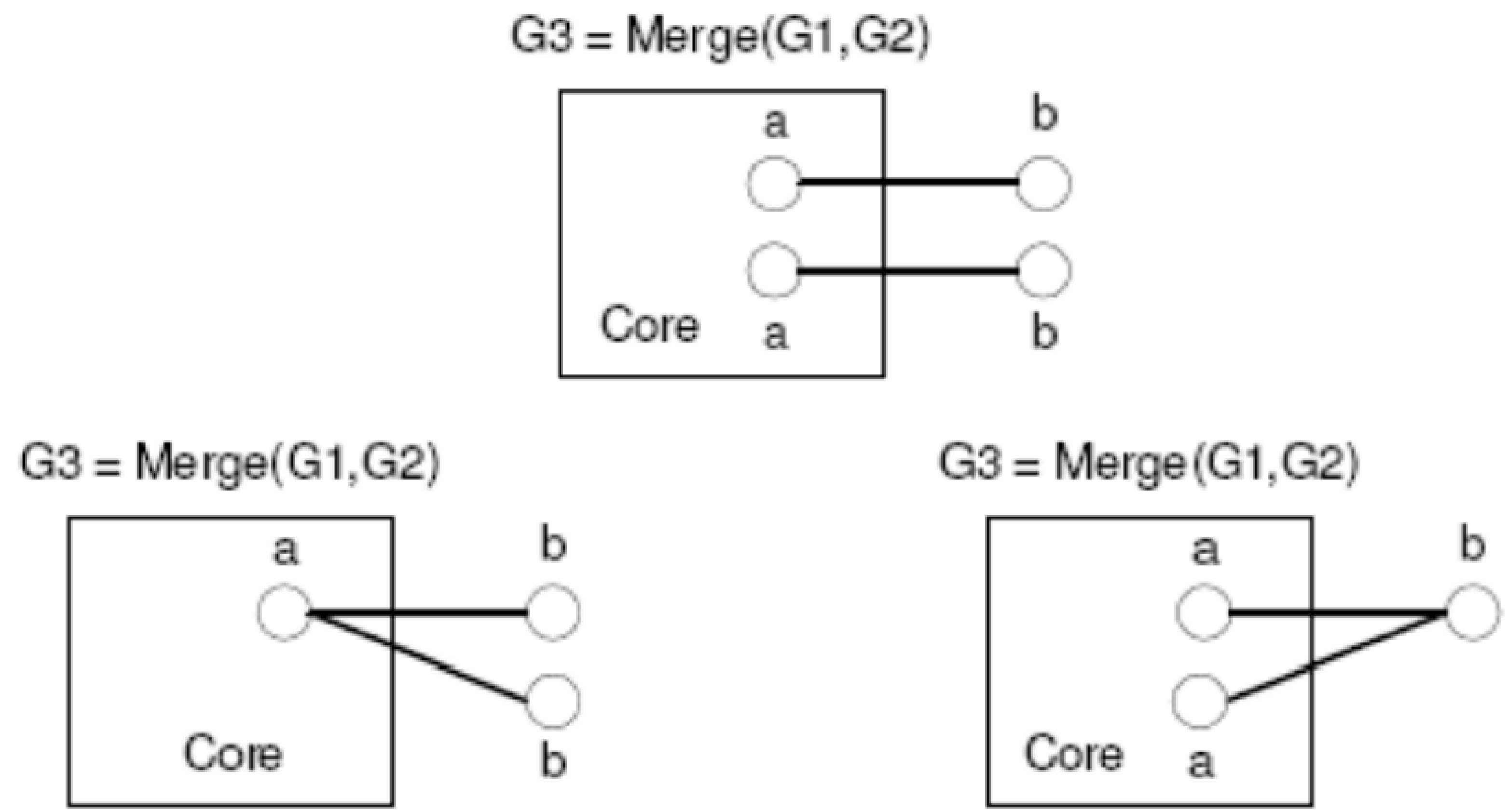
- Trường hợp 3: $a \neq c$ và b $= d$





Tổng quát cho hàm phát sinh trong FGM

- Trường hợp 4: $a = c$ và $\mathrm{b} = d$

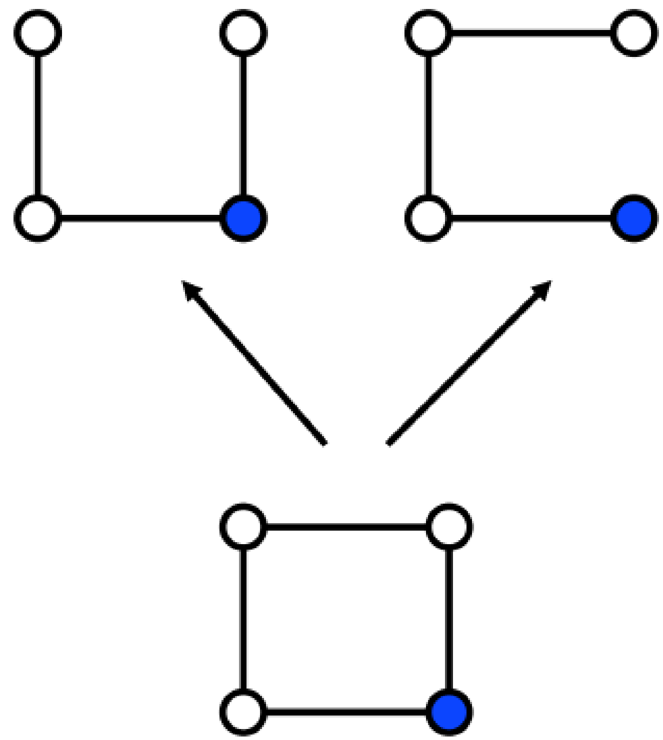


Tỉa nhánh

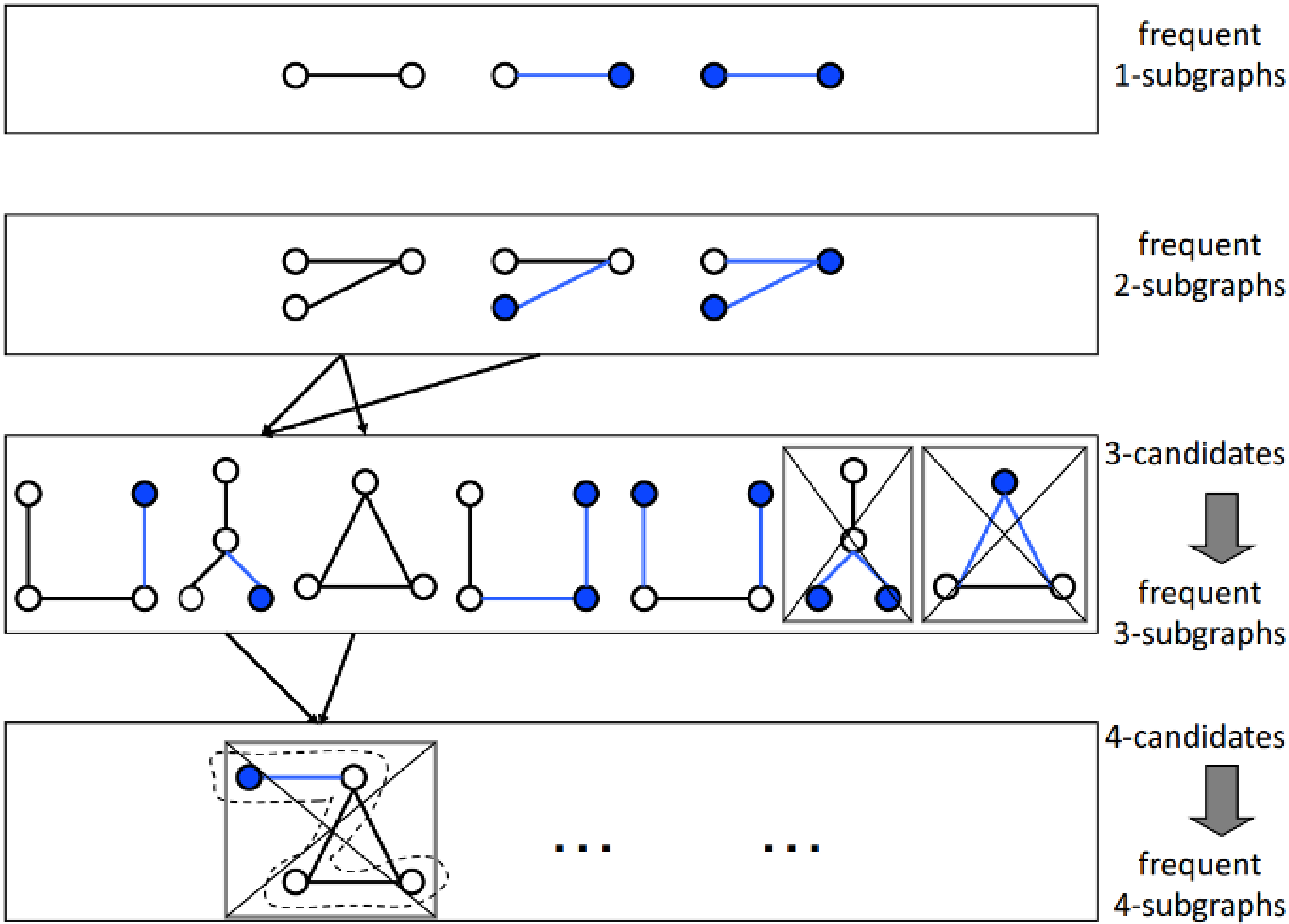
- Trong bước tỉa nhánh, mọi đồ thị con k-1 phải phổ biến.

3-candidates:

4-candidates:



Ví dụ toàn bộ quá trình trong FGM



Nhận xét

- Quá trình phát sinh ứng viên:
  - Để xác định hai ứng viên cần để gia nhập, ta cần kiểm tra đồ thị con đẳng cấu → chi phí cao
- Tỉa ứng viên:
  - Để kiểm tra tính chất bao đóng hướng xuống, ta cũng cần kiểm tra đồ thị đẳng cấu → chi phí cao
- Đếm độ phổ biến:
  - Đồ thị con đẳng cấu cần thực hiện để kiểm tra đồ thị có chứa đồ thị con này không → chi phí cao

Tối ưu hóa FGM

- Để giảm chi phí tính toán cho FGM:
  - Áp dụng nhãn chính tắc (canonical labeling) cho xác định đẳng cấu
  - Sử dụng thuật toán phát sinh ứng viên cải tiến để giảm số lần mỗi ứng viên được phát sinh
  - Áp dụng phương pháp dựa trên TID-list tăng cường (augmented TID-list) để tăng tốc đếm độ phổ biến
- Mặc dù quá trình tối ưu làm cho FGM tốt hơn nhưng nhìn chung vẫn còn thuộc bài toán NP-Complete.

Hàm phát sinh ứng viên FGM

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

$C^{k+1} \leftarrow \emptyset$

for each pair of $G_i^k, G_j^k \in F^k, i \leq j$ such that $\mathrm{cl}(G_i^k) \leq \mathrm{cl}(G_j^k)$ **do**

$\quad H^{k-1} \leftarrow \{H^{k-1} \mid$ a core H^{k-1} shared by G_i^k and $G_j^k\}$

$\quad$ **for each** core $H^{k-1} \in H^{k-1}$ **do**

$\quad\quad$ {B^{k+1} is a set of tentative candidates}

$\quad\quad B^{k+1} \leftarrow$ fsg-join(G_i^k, G_j^k, H^{k-1})

$\quad\quad$ **for each** $G_j^{k+1} \in B^{k+1}$ **do**

$\quad\quad\quad$ {test if the downward closure property holds}

$\quad\quad\quad$ flag $\leftarrow$ true

$\quad\quad\quad$ **for each** edge $e_l \in G_j^{k+1}$ **do**

$\quad\quad\quad\quad H_l^k \leftarrow G_j^{k+1} - e_l$

$\quad\quad\quad\quad$ **if** H_l^k is connected and $H_l^k \notin F^k$ **then**

$\quad\quad\quad\quad\quad$ flag $\leftarrow$ false

$\quad\quad\quad\quad\quad$ **break**

$\quad\quad\quad$ **if** flag = true **then**

$\quad\quad\quad\quad C^{k+1} \leftarrow C^{k+1} \cup \{G_j^{k+1}\}$

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

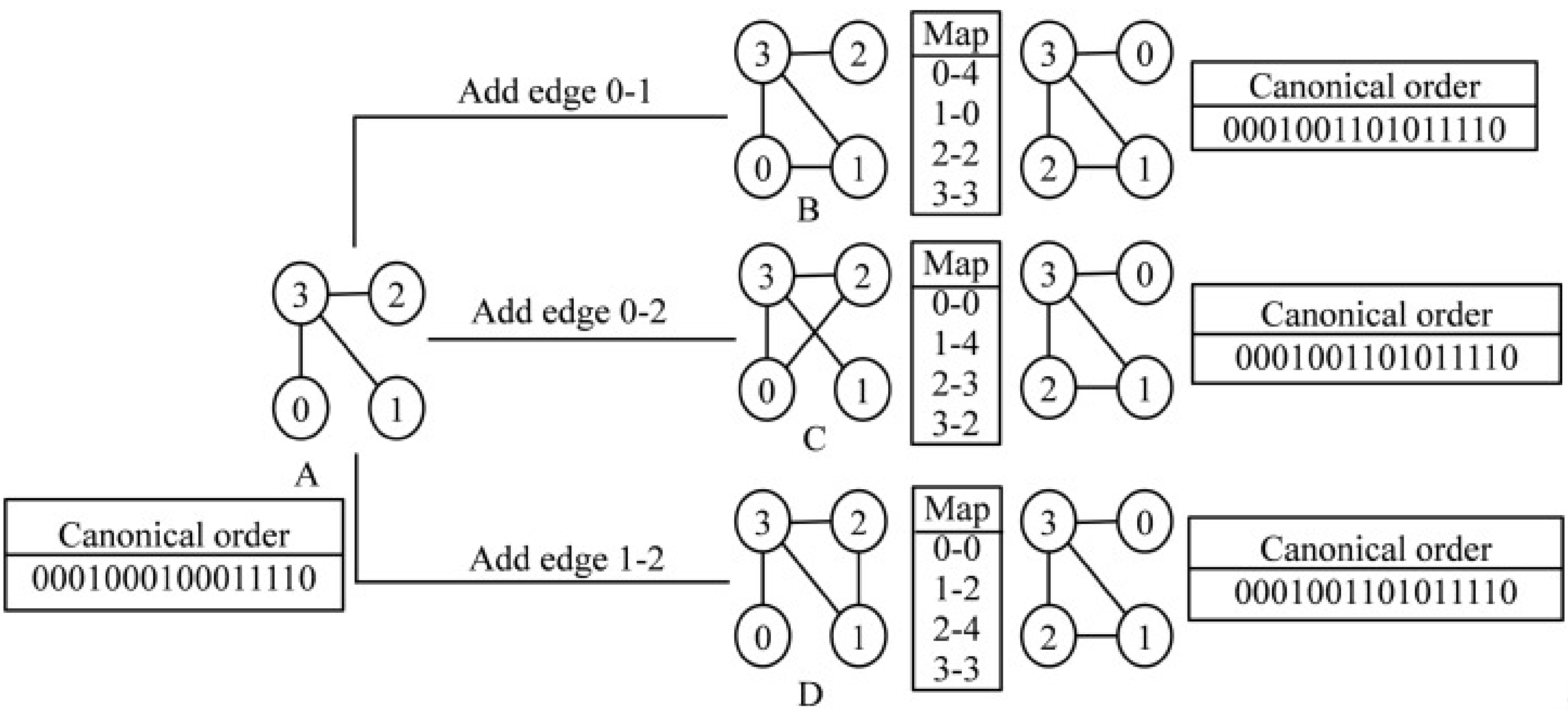
Hàm phát sinh ứng viên FGM

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: $\quad \boldsymbol{B}^{k+1} \leftarrow \boldsymbol{B}^{k+1} \cup \{$all possible candidates of size $k+1$ created from a set of e_1, e_2, H^{k-1} and $\phi\}$
7: **return** $\boldsymbol{B}^{k+1}$

Nhãn chính tắc

- Nhãn chính tắc (canonical label) của một đồ thị là một mã (code) đơn nhất xác định đồ thị
  - Hai đồ thị đẳng cấu với nhau nếu chúng được gắn cùng mã.
  - Có nhiều cách để xác định nhãn chính tắc cho đồ thị



Xác định nhãn chính tắc

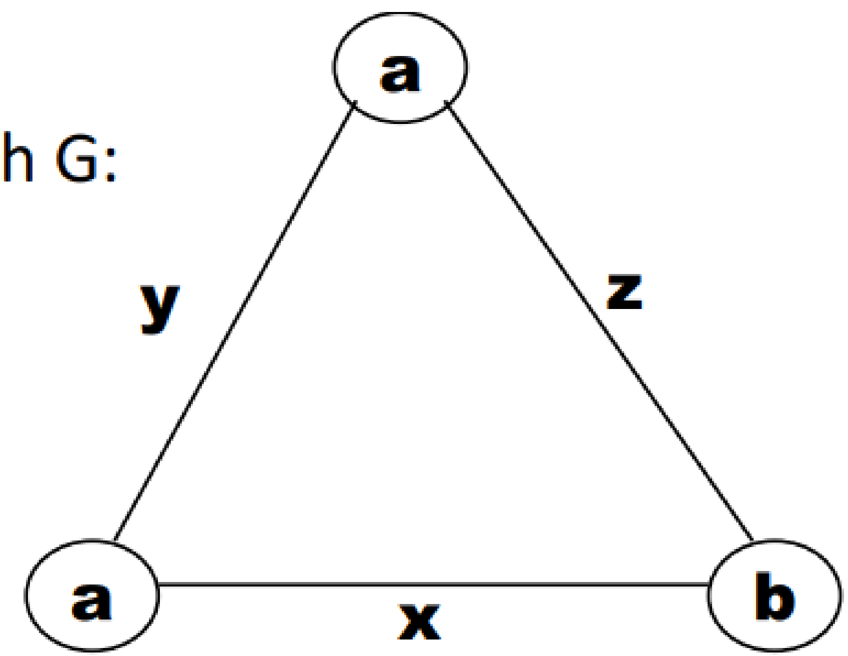
- Một cách đơn giản để gán mã cho đồ thị là chuyển biểu diễn ma trận kề của nó thành một chuỗi ký hiệu tuyến tính.

Code(M_1) = "aabyzx"

M_1 :

	a	a	b
a		y	z
a	y		x
b	z	x	

Graph G:



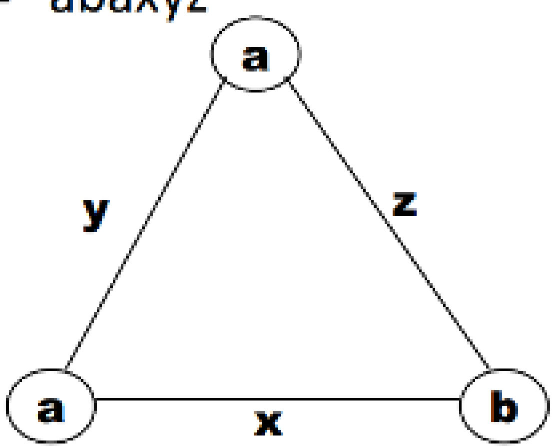
Xác định nhãn chính tắc

- Do mỗi đồ thị có thể hình thành ra nhiều dạng ma trận kề phụ thuộc vào thứ tự các đỉnh.
  - Để có được mã bất biến cho đẳng cấu (isomorphism-invariant code) là thử mọi hoán vị có thể có của các đỉnh.
  - Chọn thứ tự mà cho mã lớn nhất hoặc nhỏ nhất của nó.

$\text{Code}(M_1)$ = “aabyzx”

$\text{Code}(M_2)$ = “abaxyz”

Graph G:



M_1:

	a	a	b
a		y	z
a	y		x
b	z	x	

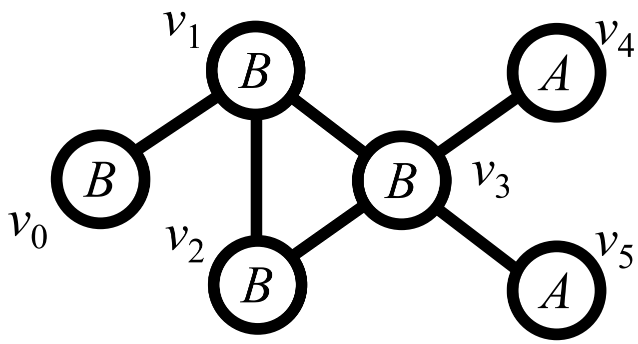
M_2:

	a	b	a
a		x	y
b	x		z
a	y	z	

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

String comparison!

Xác định nhãn chính tắc



		v_3	v_1	v_2	v_4	v_5	v_0
		B	*B*	*B*	*A*	*A*	*B*
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				

Label = “1 11 100 1000 01000”

		v_0	v_1	v_2	v_3	v_4	v_5
		B	*B*	*B*	*B*	*A*	*A*
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		

Label = “1 01 011 0001 00010”

Xác định nhãn chính tắc

- Vấn đề tìm nhãn chính tắc cũng phức tạp như đẳng cấu của đồ thị do phải kiểm tra tất cả các tổ hợp có thể có.
- FSG đề xuất một số cách heuristic để tăng tốc:
  – Dựa trên bất biến đỉnh (ví dụ: bậc)
  – Danh sách láng giềng
  – Phân hoạch lặp (iterative partitioning)
- Ý tưởng chính của các phương pháp này để giúp bỏ bớt các trường hợp giống nhau.

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);

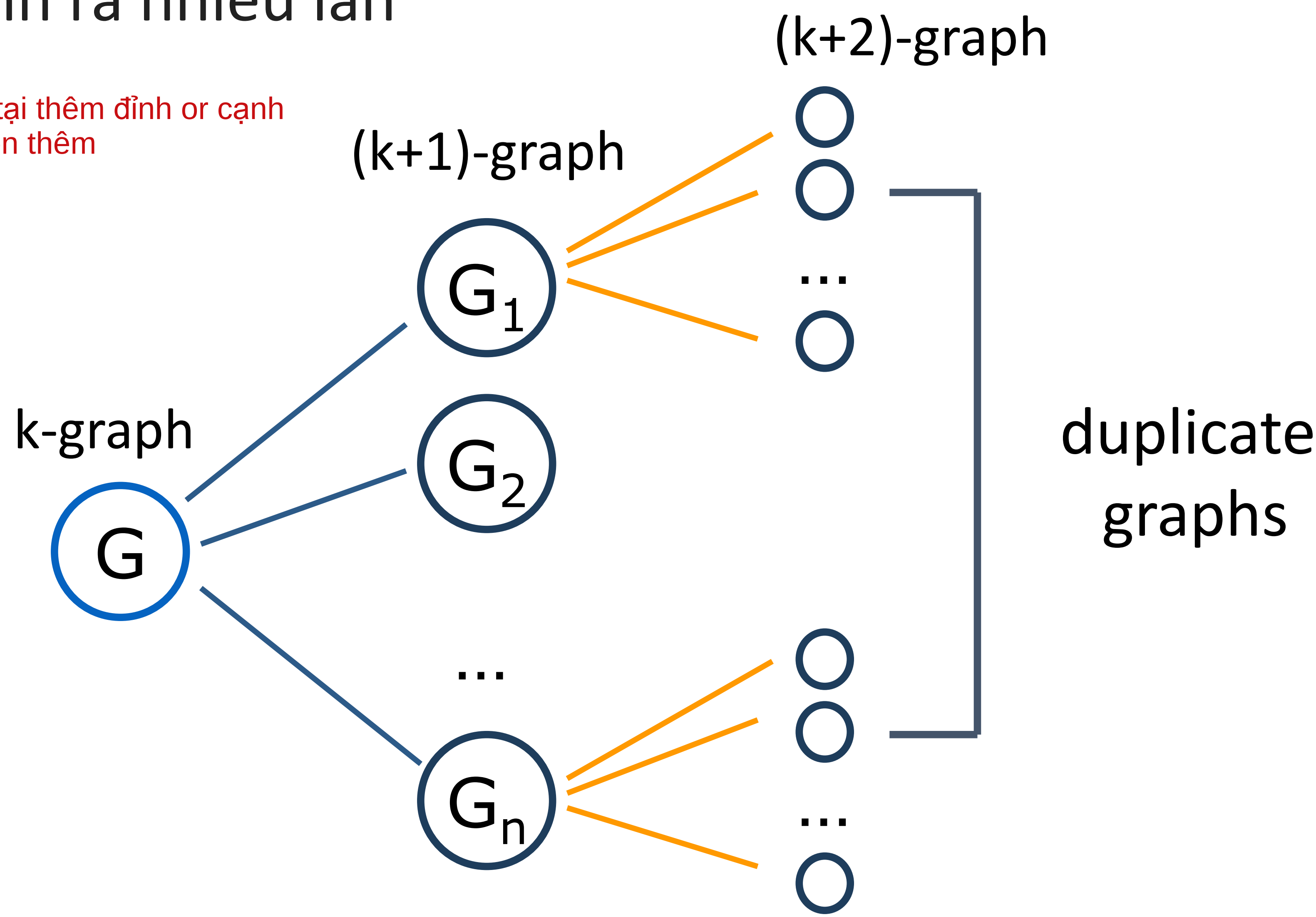
procedure PatternGrowthGraph(g, D, min_sup, S)

(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

Lấy k hiện tại thêm đỉnh or cạnh
=> phát triển thêm



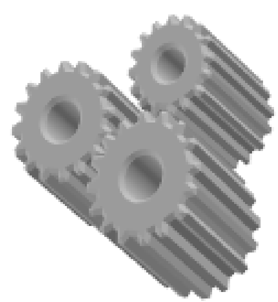
Thuật toán gSpan

- 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

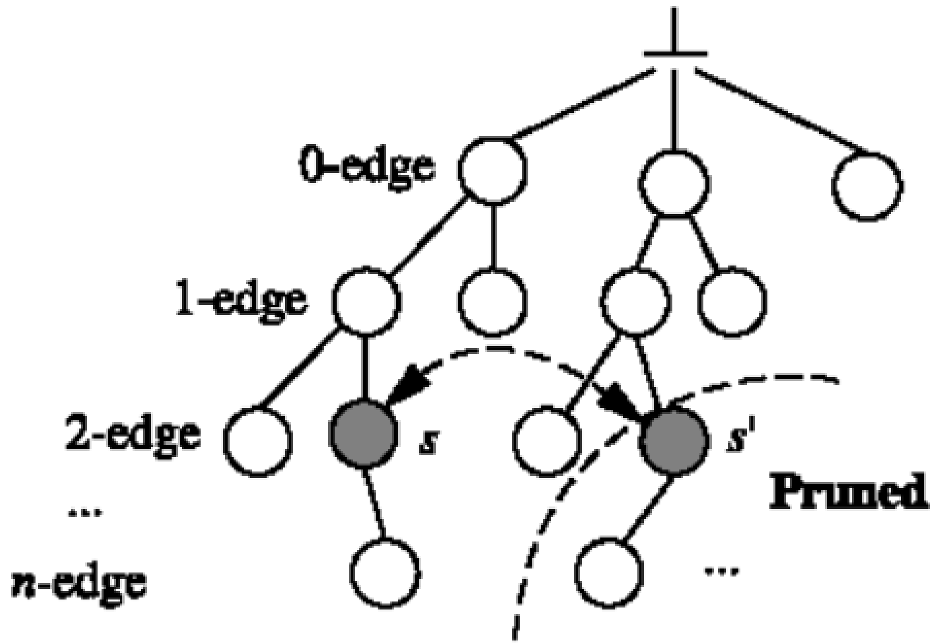
Thuật toán gSpan

- 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



Thuật toán gSpan



- 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’.

Thuật toán gSpan

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:

$S \leftarrow \emptyset$;

Call gSpan(s, D, *min_sup*, S);

procedure PatternGrowthGraph(s, D, *min_sup*, S)

(1) **if** $s \neq dfs(s)$, **then**

(2)   **return**;

(3) insert s into S;

(4) set C to $\emptyset$;

(5) scan D once, find all the edges e such that s can be right-most extended to $s \diamond_r e$; insert $s \diamond_r e$ into C and count its frequency;

(6) sort C in DFS lexicographic order;

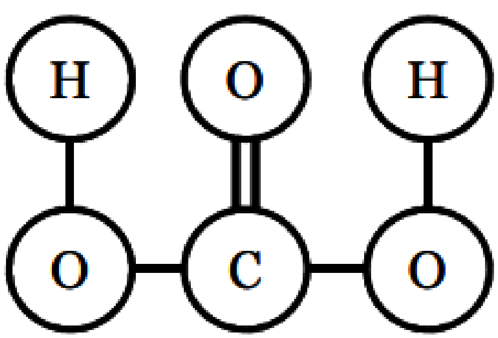
(7) **for each** frequent $s \diamond_r e$ in C **do**

(8)   gSpan($s \diamond_r e$, D, *min_sup*, S);

(9) **return**;

Thuật toán gSpan

- Đầu vào:
  - Một tập các đồ thị và ngưỡng trợ
  - Mỗi đồ thị có dạng $G = (V, E, L_V, L_E)$
    - V là tập đỉnh
    - E là tập cạnh
    - L_V là nhãn của các đỉnh
    - L_E là nhãn của các cạnh
    - Nhãn không bắt buộc duy nhất

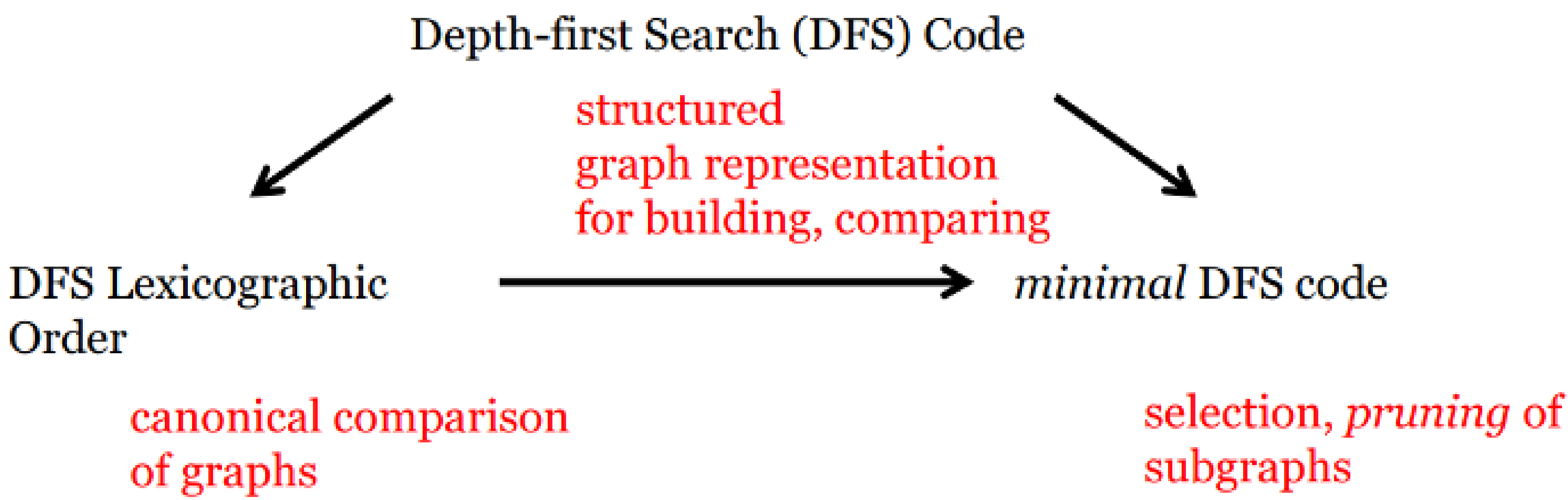


$$L_V = \{H, O, C\}$$

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

Thuật toán gSpan

- Chiến lược:
  - Xây dựng đồ thị con phổ biến từ dưới lên (bottom-up), sử dụng mã DFS như thể hiện chính tắc cho đồ thị.
  - Bỏ đi trùng lắp thông qua mã DFS tối thiểu dựa trên thứ tự từ điển của mã.

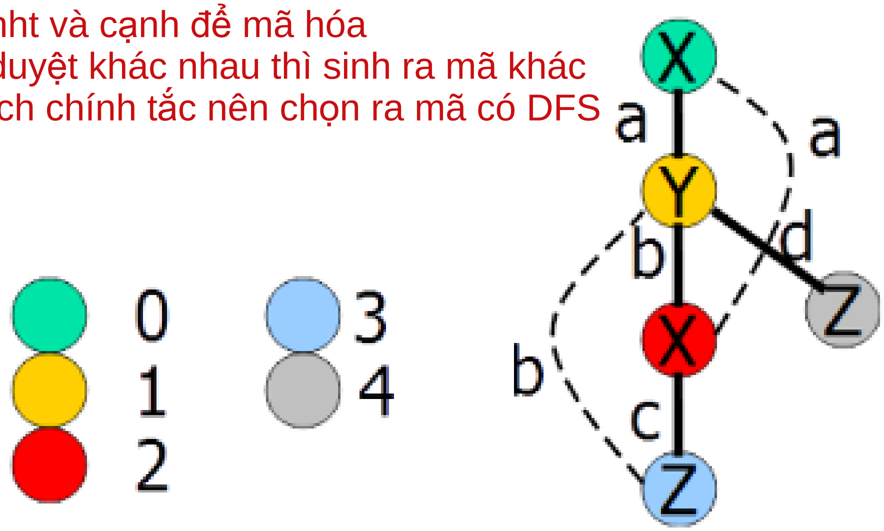


Mã DFS trong gSpan

- Mã DFS là tuần tự của các cạnh được duyệt bằng DFS
  - Mỗi cạnh được mã hóa dưới dạng $(i, j, L_i, L_{(i\ j)}, L_j)$
    - i, j: là các đỉnh
    - L_i, L_j: nhãn đỉnh đầu và cuối của cạnh
    - $L_{(i\ j)}$: nhãn cạnh
    - $i < j$: được gọi là cạnh tới (forward edge)
    - $i > j$: được gọi là cạnh lùi (back edge)

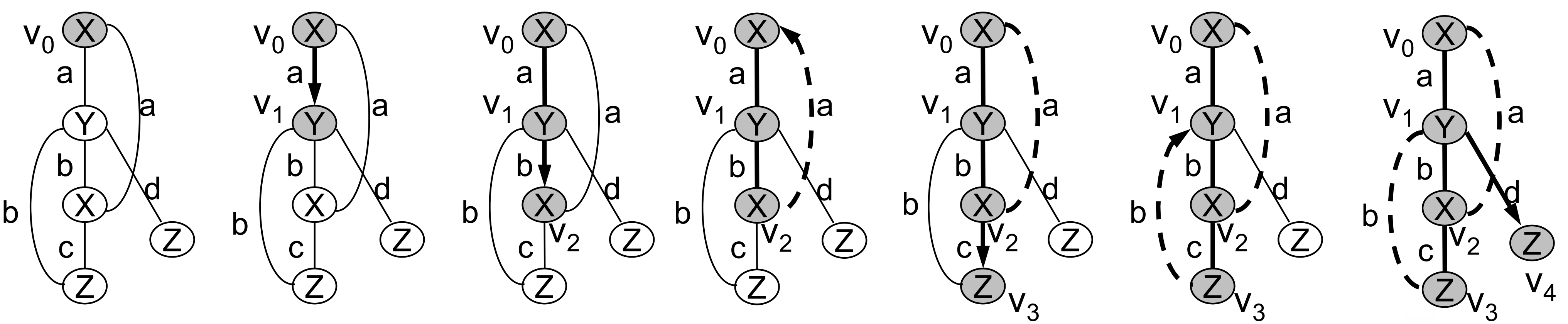
Mã DFS trong gSpan

=> Đi hết đinht và cạnh để mã hóa
=> Với các duyệt khác nhau thì sinh ra mã khác
=> Ko có cách chính tắc nên chọn ra mã có DFS nhỏ nhất



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)

ko tính bước quay lui

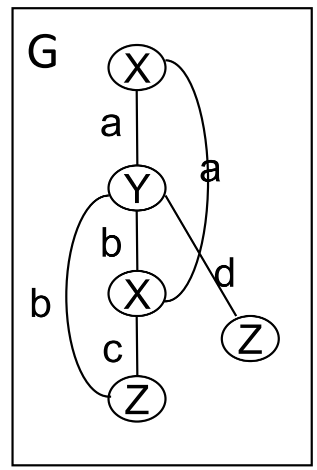


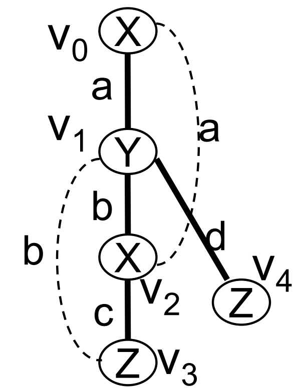
Mã DFS	(0,1,X,a,Y)	(1,2,Y,b,X)	(2,0,X,a,X)	(2,3,X,c,Z)	(3,1,Z,b,Y)	(1,4,Y,d,Z)

Mã DFS trong gSpan

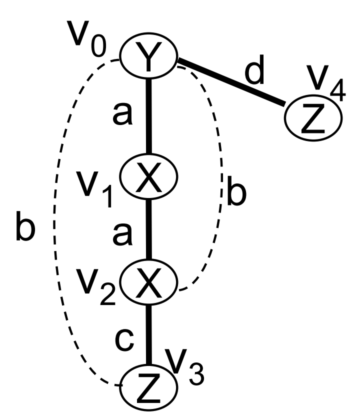
- Với một đồ thị cho trước, có thể có nhiều mã DFS phụ thuộc vào cách duyệt đồ thị và cách chọn điểm bắt đầu.

	(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)

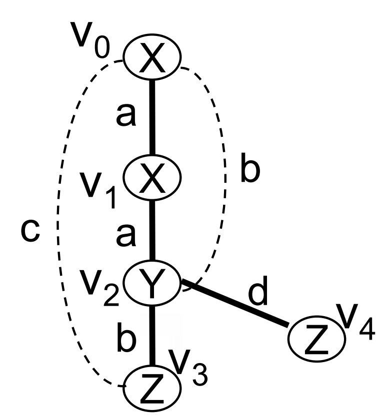




(a)



(b)



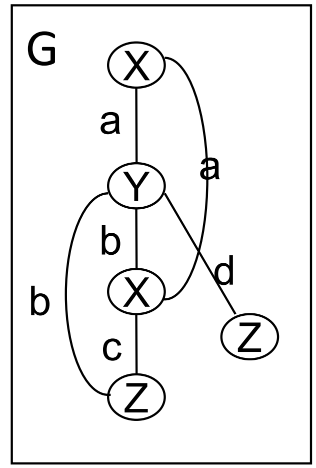
(c)

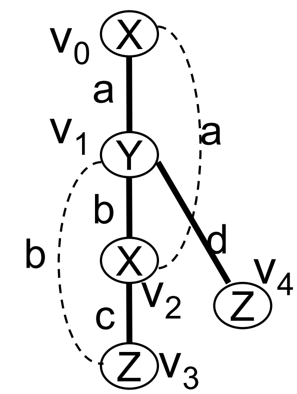
Mã DFS trong gSpan

- Cần chọn ra mã DFS nhỏ nhất

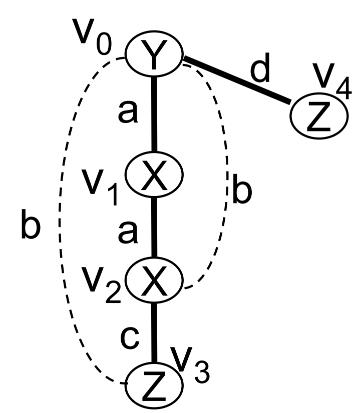
Min DFS-Code

	(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)**

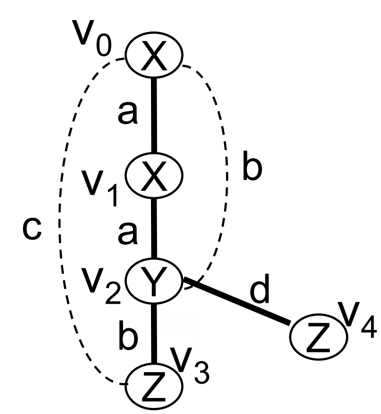




(a)



(b)

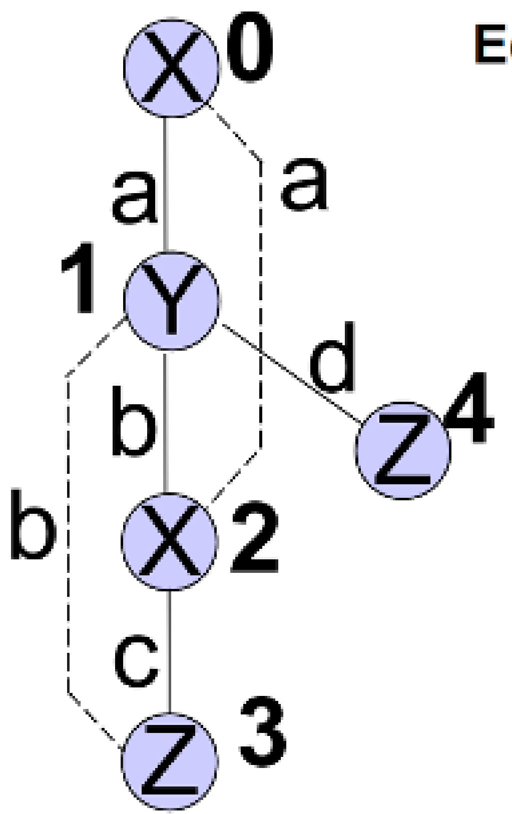


(c)

Thứ tự cạnh hợp lệ trong mã DFS

- Để xác định mã DFS tối thiểu, ta cần thống nhất một thứ tự cạnh để có thể so sánh được các mã DFS.
- Ký hiệu:
  - $e_1 = (i_1, j_1), e_2 = (i_2, j_2)$ => vẫn còn các phần sau nữa, focus vào 2 đỉnh
  - $e_1 \prec e_2$: nghĩa là e_1 xuất hiện trước e_2 trong mã
- Luật thứ tự:
  - Luật 1: Nếu $i_1 = i_2$ và $j_1 < j_2$ thì $e_1 \prec e_2$
    - Từ cùng một đỉnh nguồn, e_1 được duyệt trước e_2 trong DFS
  - Luật 2: Nếu $i_1 < j_1$ và $j_1 = i_2$ thì $e_1 \prec e_2$ (cạnh tới)
    - e_1 là một cạnh tới và e_2 được duyệt như là kết quả của phép duyệt e_1
  - Luật 3: Nếu $e_1 \prec e_2$ và $e_2 \prec e_3$ thì $e_1 \prec e_3$ (bắc cầu)

Thứ tự cạnh hợp lệ trong mã DFS



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)**
- . . .

Mã DFS và thứ tự từ điển DFS

- Lưu ý cần phân biệt mã DFS và thứ tự từ điển mã DFS
  - Mã DFS: là thứ tự các cạnh của một DFS cụ thể.
  - Thứ tự từ điển DFS: là thứ tự giữa các mã DFS khác nhau.

Thứ tự từ điển DFS

- Cho trước:
  - Thứ tự từ điển của tập nhãn đỉnh và cạnh L (ký hiệu $\prec_L$)
  - Đồ thị G_α, G_β (với tập nhãn giống nhau)
  - Các mã DFS:
    - $\alpha = code(G_\alpha, T_\alpha) = (a_0, a_1, \dots, a_m)$
    - $\beta = code(G_\beta, T_\beta) = (b_0, b_1, \dots, b_n)$
    - Giả sử $m \leq n$
- $\alpha \leq \beta$ nếu và chỉ nếu thỏa một trong các điều kiện sau:
  - $\exists t, 0 \leq t \leq \min(m, n)$ sao cho
    - $a_k = b_k$ với $k < t$ và => trước vị trí k thì các cạnh trong alpha = cạnh trog beta bằng nhau
    - $a_t \prec_e b_t$
  - $a_k = b_k$ với mọi k trong đoạn $0 \leq k \leq m$ => mọi cạnh trong alpha đều nằm trong beta

So sánh cạnh $a_t \prec_e b_t$

- Gọi:
  - $a_t = (i_a, j_a, L_{i_a}, L_{i_a,j_a}, L_{j_a})$
  - $b_t = (i_b, j_b, L_{i_b}, L_{i_b,j_b}, L_{j_b})$
- $a_t \prec_e b_t$ nếu thỏa một trong các trường hợp sau:
  - TH1: Cả hai cạnh là cạnh tới ($j_a = j_b$ do các cạnh phía trước đều bằng nhau) và …
  - TH2: Cả hai cạnh là cạnh lui ($i_a = i_b$ với lí do tương tự) và…
  - TH3: a_t là cạnh lui và b_t là cạnh tới

So sánh cạnh $a_t \prec_e b_t$

- Gọi:
  - $a_t = (i_a, j_a, L_{i_a}, L_{i_a,j_a}, L_{j_a})$
  - $b_t = (i_b, j_b, L_{i_b}, L_{i_b,j_b}, L_{j_b})$
- $a_t \prec_e b_t$ nếu thỏa một trong các trường hợp sau:
  - TH1: Cả hai cạnh là cạnh tới ($j_a = j_b$ do các cạnh phía trước đều bằng nhau) và thỏa một trong điều kiện:
    - $i_b < i_a$ (cạnh bắt đầu từ một đỉnh được viếng thăm sau)
    - $i_a = i_b$ và các nhãn của a có thứ tự nhỏ hơn nhãn của b theo bộ ba.
      - Ví dụ: $a_t = (-,-,m,e,x)$, $b_t = (-,-,m,u,x)$

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

=> so sánh m vs m (nếu < thì suy ra at < bt) nếu bằng thì tiếp e vs u nếu bằng thì tiếp x vs x. nếu đều bằng thì suy ra at = bt

So sánh cạnh $a_t \prec_e b_t$

- Gọi:
  - $a_t = (i_a, j_a, L_{i_a}, L_{i_a,j_a}, L_{j_a})$
  - $b_t = (i_b, j_b, L_{i_b}, L_{i_b,j_b}, L_{j_b})$
- $a_t \prec_e b_t$ nếu thỏa một trong các trường hợp sau:
  - TH1: …
  - TH2: Cả hai cạnh là cạnh lui ($i_a = i_b$ với lí do tương tự) và thỏa một trong điều kiện sau:
    - $j_a < j_b$ (cạnh kết nối đến một đỉnh được viếng thăm trước)
    - $j_a = j_b$ và nhãn cạnh của a có thứ tự nhỏ hơn nhãn cạnh của b.
      - ➢ Không xét nhãn đỉnh vì tất cả các cạnh phía trước bằng nhau nên nhãn các đỉnh đều đã bằng nhau.

Ví dụ xét các mã DFS

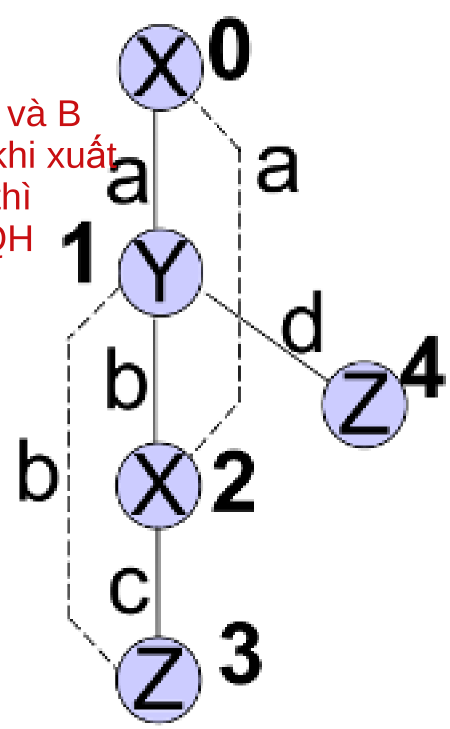
Định nghĩa 1:
X<Y<Z
a<b<c

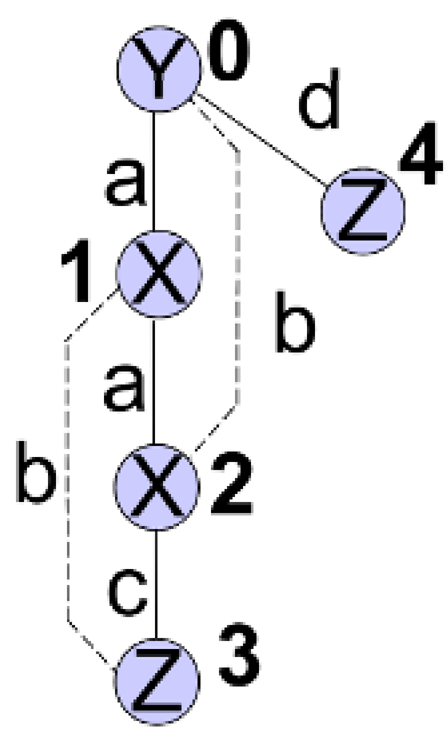
Định nghĩa 2:
X=Y=Z
b<c<a

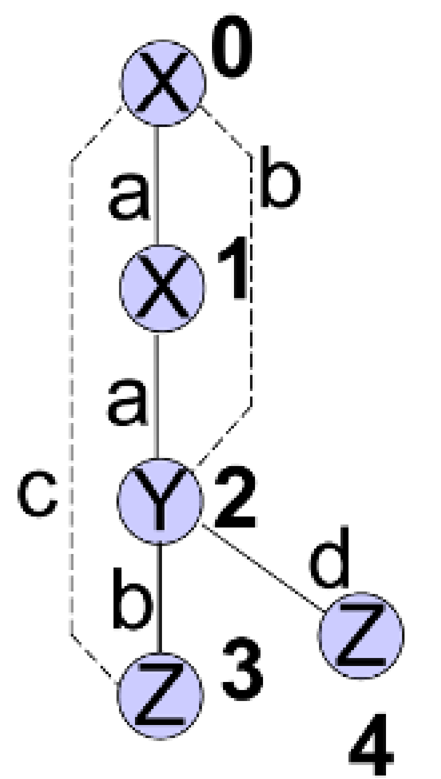
Edge #	Code (A)		Code (B)	Code (C)
	=> xét cạnh tới or cạnh lui trước (trong cùng 1 cạnh trước)			
0 cạnh tới	(0,1,X,a,Y)	< (X<Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	>	(1,2,X,a,X)	(1,2,X,a,Y)
2 cạnh lui	(2,0,X,a,X)	ĐN2: > (a>b)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	=	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	=	(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)

=> Xét QH giữa A và B dựa trên ĐN đến khi xuất hiện 1 cặp < or > thì dừng và xét tiếp QH giữa B và C

=> ĐN3:
X=Y=Z
a=b=c
Suy ra: B>A>C



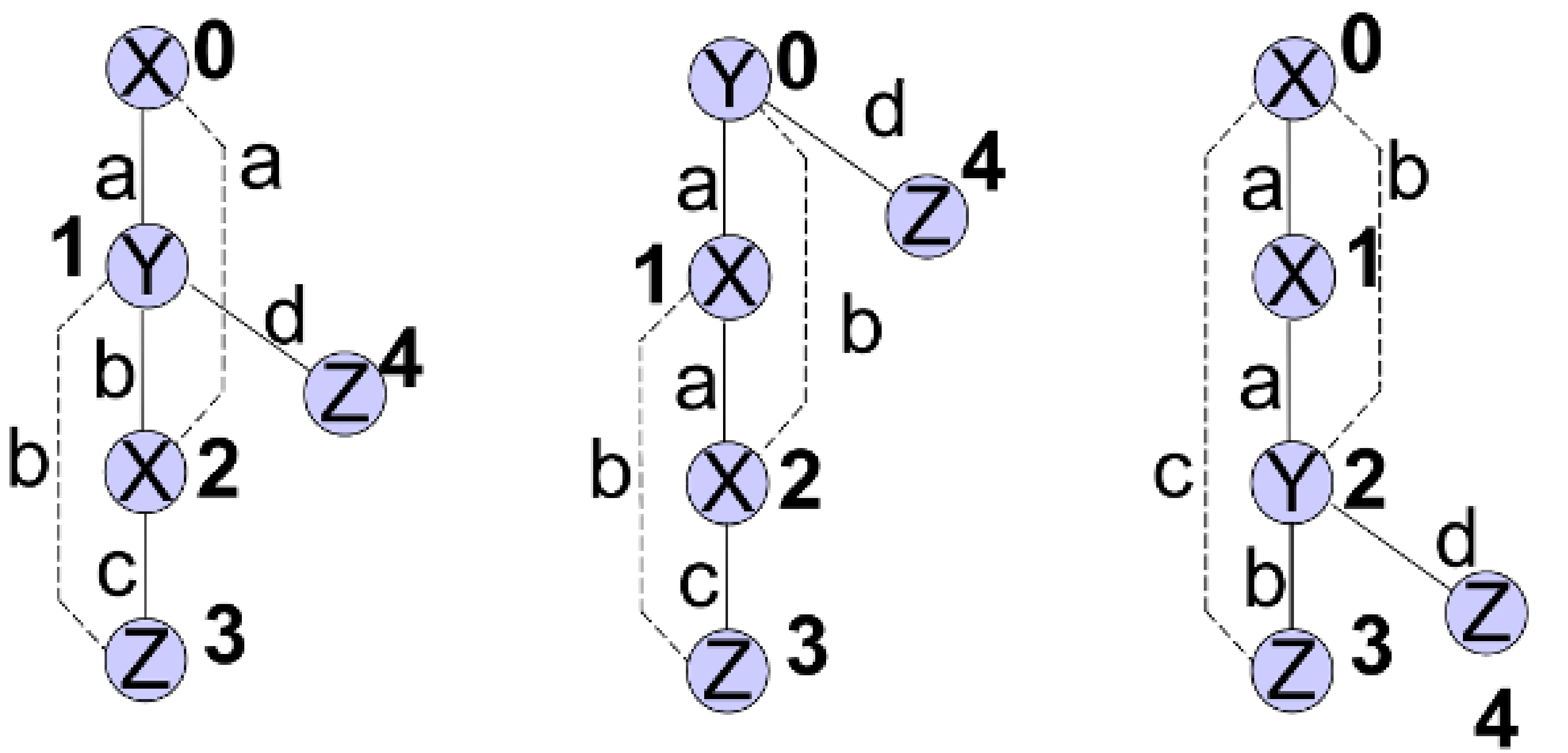




Ví dụ xét các mã DFS

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

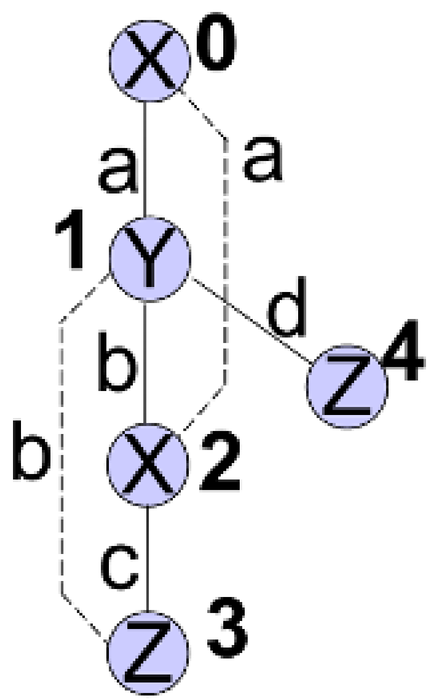
0	(0,1,X,a,Y)	(0,1,Y,a,X)	(0,1,X,a,X)
1	(1,2,Y,b,X)	(1,2,X,a,X)	(1,2,X,a,Y)
2	(2,0,X,a,X)	(2,0,X,b,Y)	(2,0,Y,b,X)
3	(2,3,X,c,Z)	(2,3,X,c,Z)	(2,3,Y,b,Z)
4	(3,1,Z,b,Y)	(3,1,Z,b,X)	(3,0,Z,c,X)
5	(1,4,Y,d,Z)	(0,4,Y,d,Z)	(2,4,Y,d,Z)

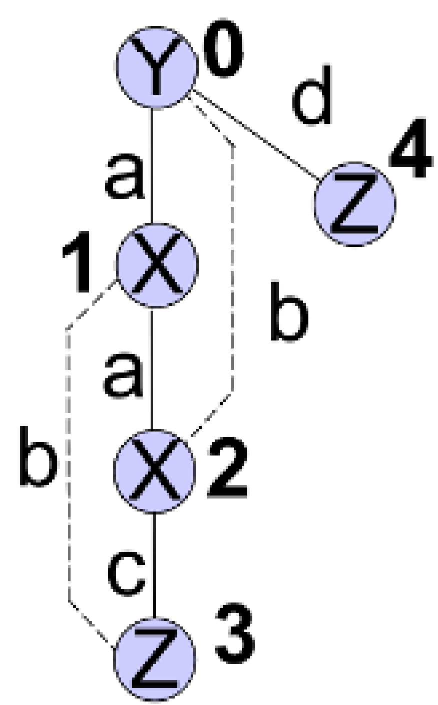


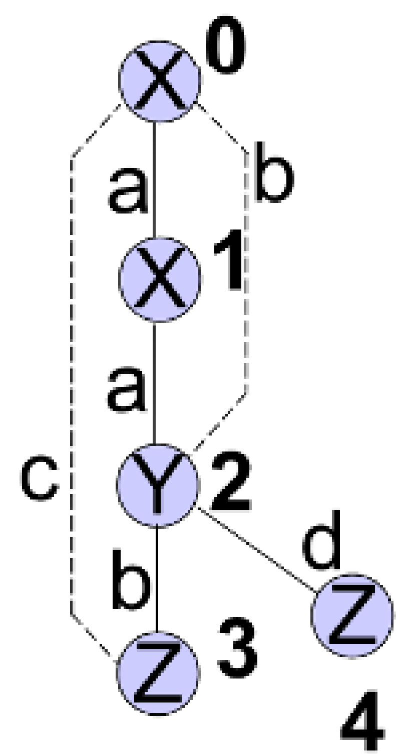
Ví dụ xét các mã DFS

$<_L = \{X = Y = Z : b < c < a\}$ 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)



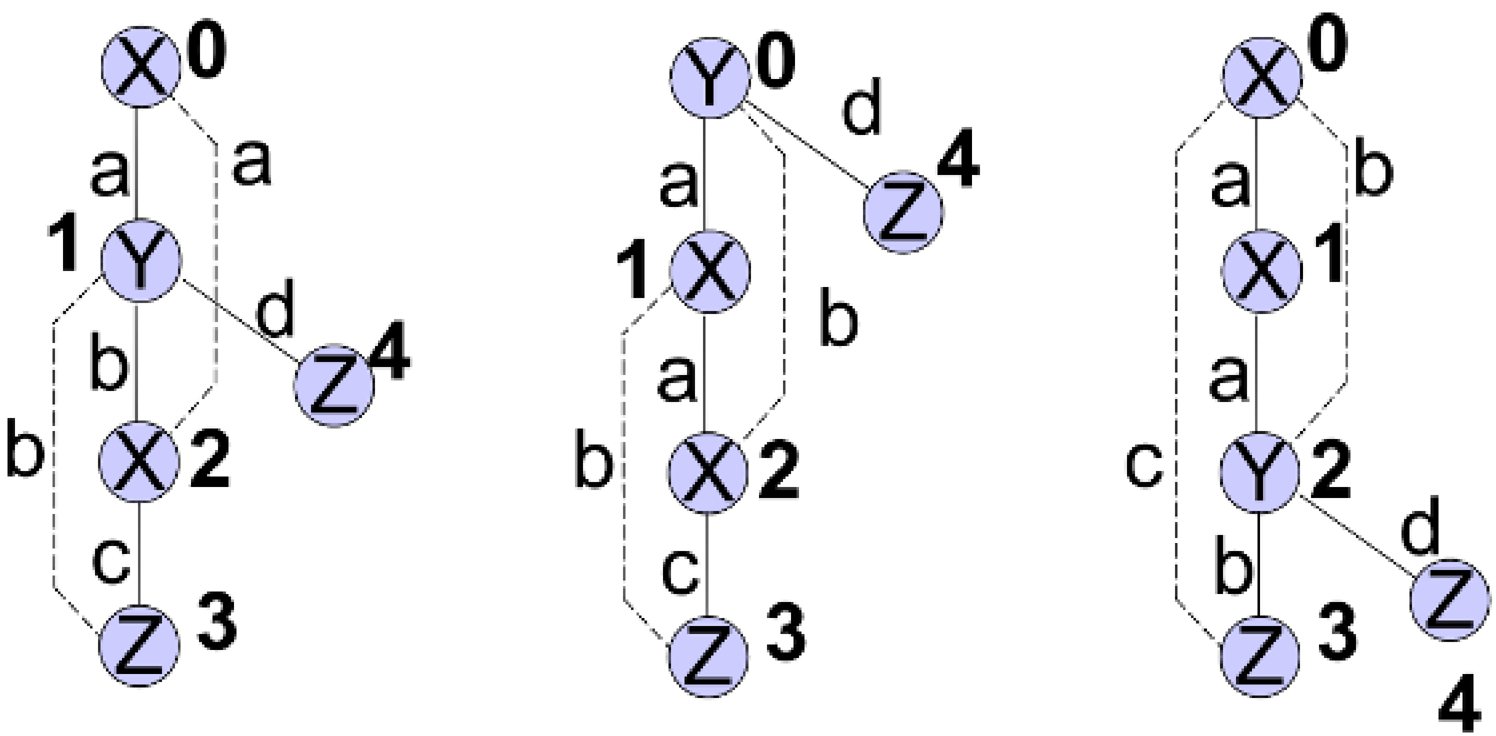




Ví dụ xét các mã DFS

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

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

Mối quan hệ cha con trong cây mã DFS

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

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

  - G_1 được gọi là cha của G_2
  - G_2 được gọi là con của G_1

(0,1,X,a,Y)
(1,2,Y,b,X)

⇒

(0,1,X,a,Y)
(1,2,Y,b,X)
(2,0,X,a,X)

⇒

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

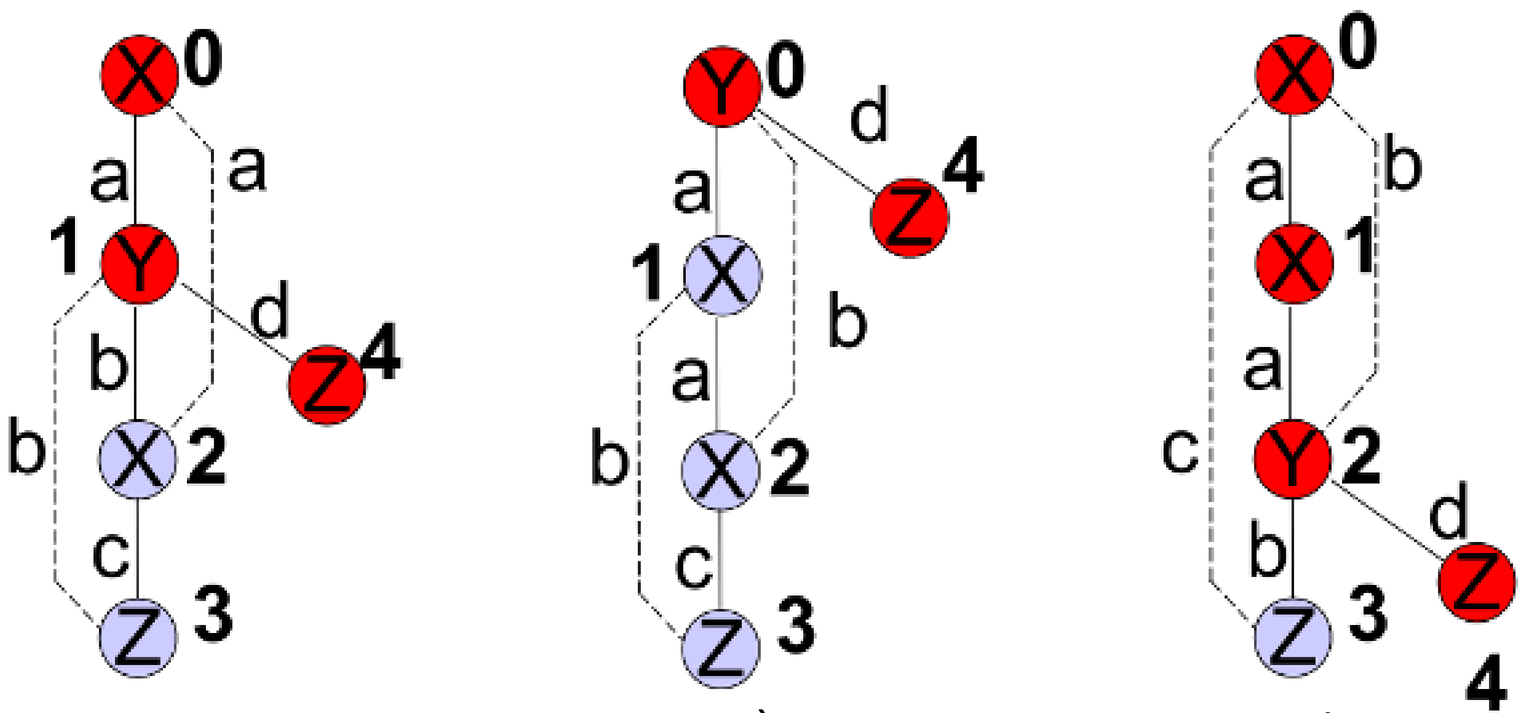
Mối quan hệ cha con trong cây mã DFS

- Do b cần lớn hơn tất các a_i nên một mã DFS hợp lệ đòi hỏi b phát triển từ một đỉnh trên đường đi phải nhất (rightmost path)
  - Nếu mở rộng cạnh tới (forward edge) đến một mã DFS thì nó phải xảy ra từ một đỉnh trên đường đi phải nhất.
  - Nếu mở rộng cạnh lui (backward edge) đến một mã DFS thì nó phải xảy ra từ đỉnh phải nhất (rightmost vertex).

Đường đi phải nhất

T1: BAB$D$$B$C
T2: BAB$D$$C$B
T3: BCBAB$D
T4: BCBAD$B

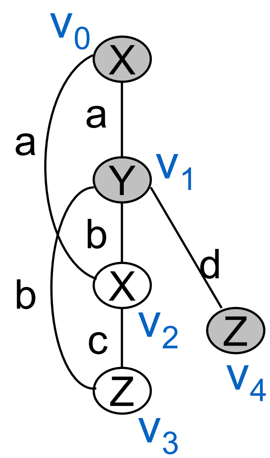
- Cho một thứ tự đỉnh được viếng thăm: $(v_0, v_1, \dots, v_n)$
  - v_0: là đỉnh đầu tiên được viếng thăm
  - v_n: là đỉnh cuối cùng được viếng thăm (đỉnh nằm bên phải nhất).
- Đường đi phải nhất (rightmost path) là đường đi ngắn nhất giữa v_0 và v_n sử dụng chỉ các cạnh tới.



Các đỉnh màu đỏ nằm trên đường đi phải nhất

Đường đi phải nhất

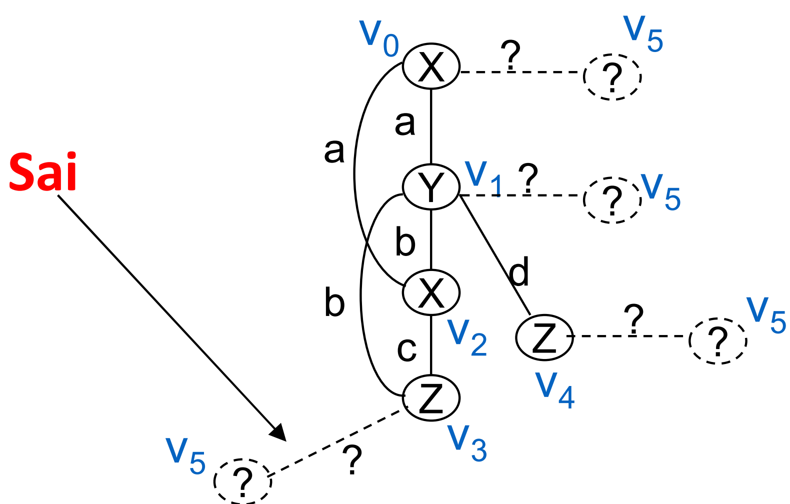
Graph G_1:



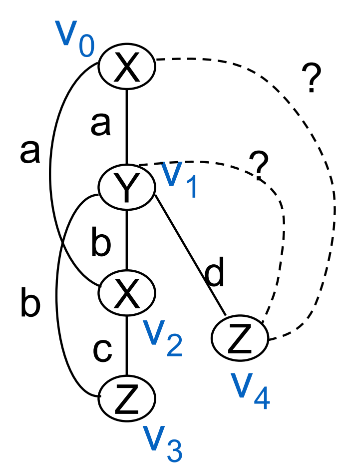
Min(g) = (0,1,X,a,Y) (1,2,Y,b,X) (2,0,X,a,X) (2,3,X,c,Z) (3,1,Z,b,Y) (1,4,Y,d,Z)

Cần phát triển thêm cạnh vào G_1?

Graph G_2:

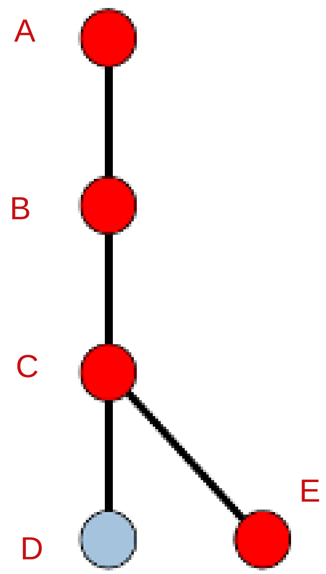


Cạnh tới



Cạnh lui

Ví dụ phát triển cây

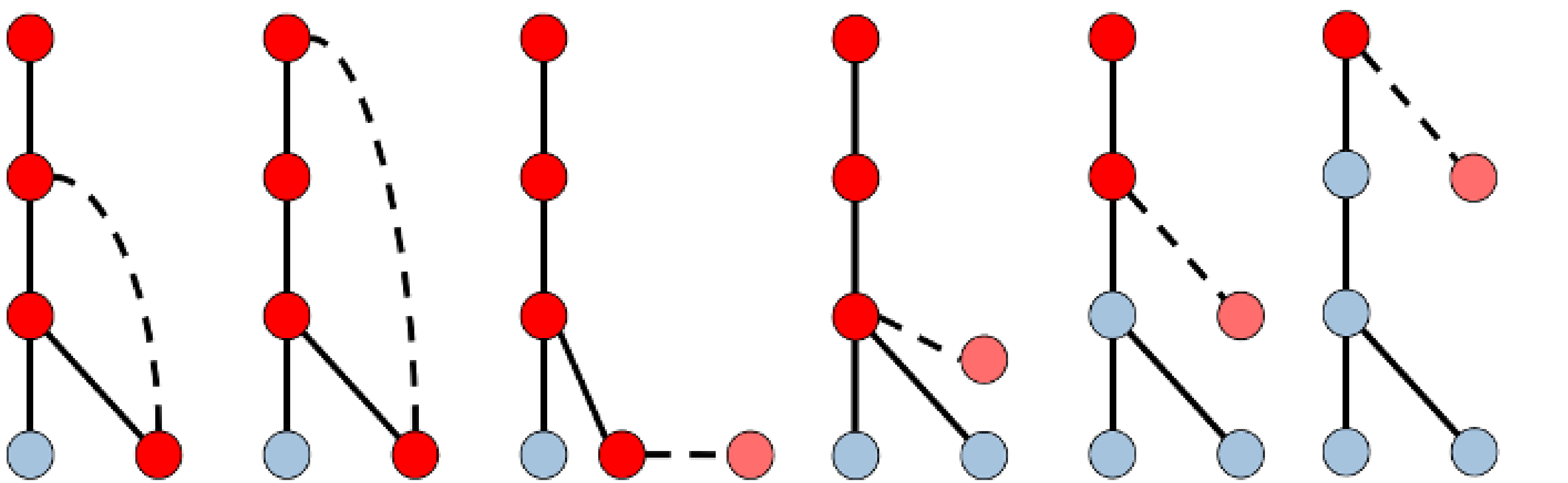


Rightmost path vertex

=> Cạnh lui phải bắt đầu từ đỉnh phải nhất
=> Đg đi phải nhất là đg đi ngắn nhất từ v0 đến v(n) cuối cùng (đg màu đỏ)

=> Dạng chuẩn tắc: ABCD$E

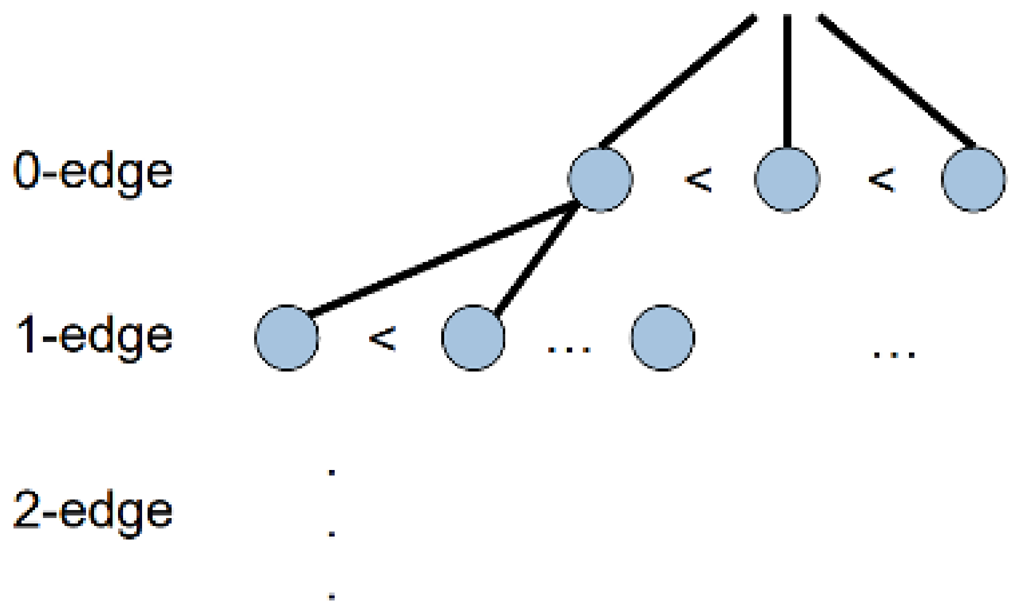
=> Cây nào dài hơn thì nhỏ hơn, cây rỗng nhỏ hơn tất cả cây
Điều kiện: 1 đỉnh mới đc gắn vào mà cây đó vẫn chuẩn tắc (chuẩn tắc thì phổ biến)
+ x phải gắn trên đg đi phải nhất
+ x phải có đk sau khi gắn xong cây đẻ trước phải chuẩn tắc (cây bên trái < cây bên phải)



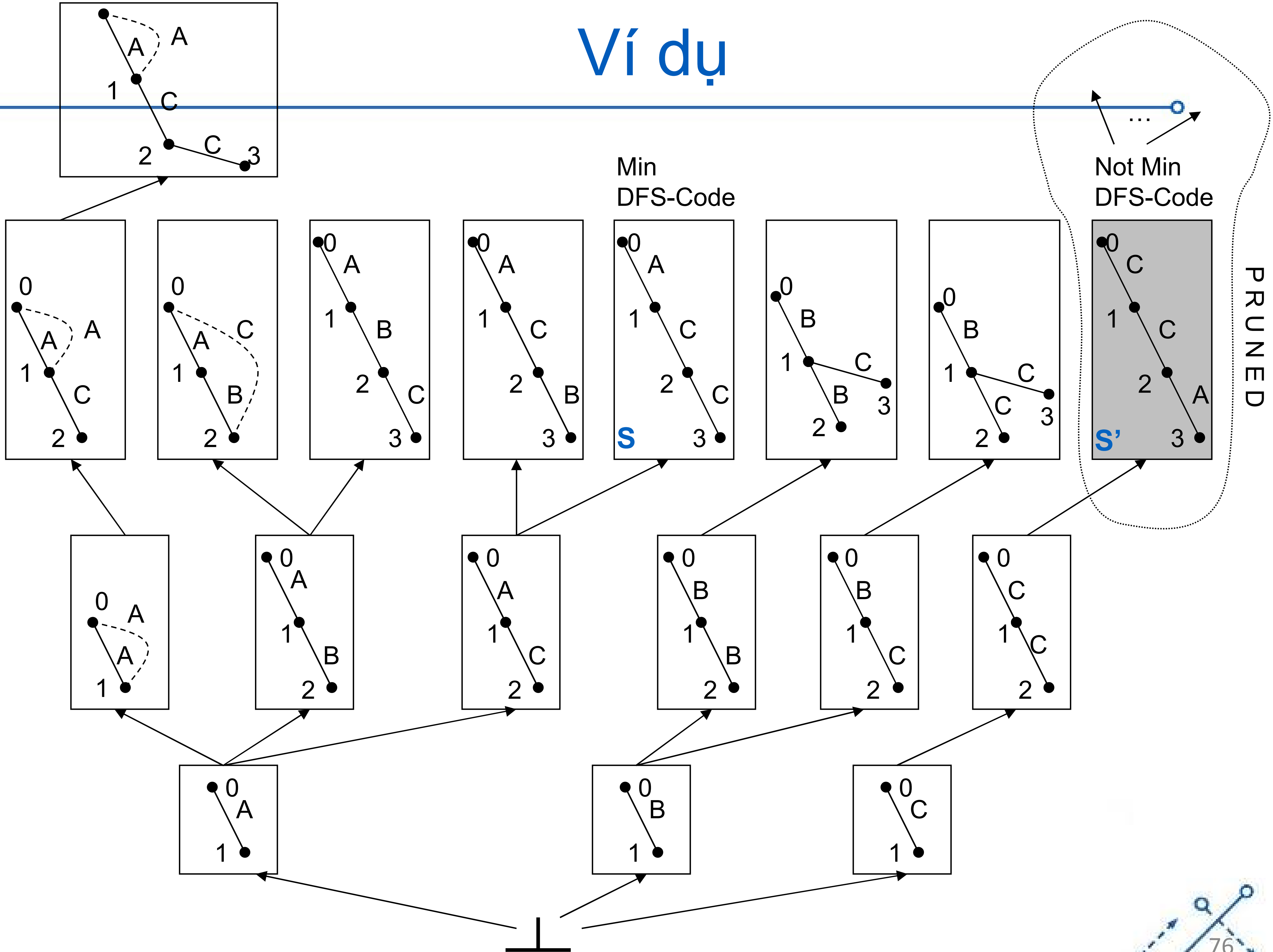
ABCDEB | ABCDEA | ABCD$È | ABCD$E$F | ABCD$E$$F | ABCD$E$$$F

Xây dựng cây mã DFS

- Tổ chức các đỉnh mã DFS theo luật cha-con.
- Các đỉnh là các mã DFS ngoài trừ …
  - mức đầu tiên của cây là một đỉnh cho mỗi nhãn đỉnh.
- Mỗi mức của cây thêm một cạnh đến mã DFS.
- Các đỉnh anh em được tổ chức theo thứ tự mã DFS tăng dần.
- Duyệt cây theo dạng giữa (inorder – LNR) sẽ tuân theo thứ tự DFS.
- Các cạnh lui (backward edge) phải được thêm đầu tiên.



Ví dụ



Thuật toán gSpan

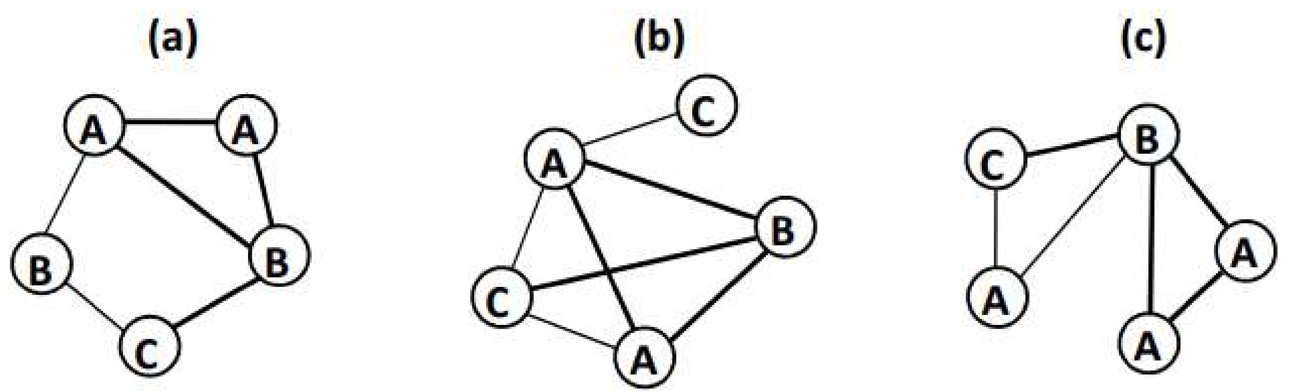
- Input: cơ sở dữ liệu đồ thị D, min_sup
- Output: tập đồ thị con phổ biến S
- Tiến trình:
  - Bỏ đi các đỉnh và cạnh không phổ biến
  - S ← đồ thị con một cạnh phổ biến trong D (sử dụng mã DFS)
  - Sắp xếp S theo thứ tự
  - N← S (vì S sẽ được cập nhật)
  - Cho mỗi $n \in N$ thực hiện:
    - gSpan_Extend(D,n,min_sup,S)
    - Bỏ đi n khỏi tất cả đồ thị trong D

Hàm gSpan_extend

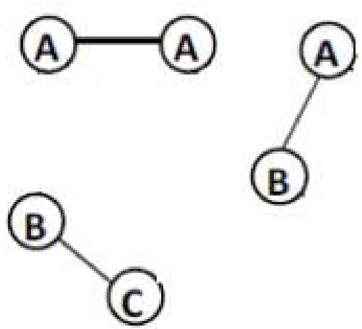
- Input: cơ sở dữ liệu đồ thị D, mã DFS n, min_sup
- Input/Output: tập đồ thị con phổ biến S
- Tiến trình:
  - If n không phải là nhỏ nhất thì dừng
  - Ngược lại:
    - Thêm n vào S
    - Với mỗi mở rộng 1 cạnh phải nhất của n (e)
      - ➢ Nếu support(e) >= min_sup
        - ❖ Gọi đệ quy gSpan_extend(D,e, min_sup, S)

Ví dụ

- Input: min_sup = 3
- Các đồ thị không có nhãn cạnh



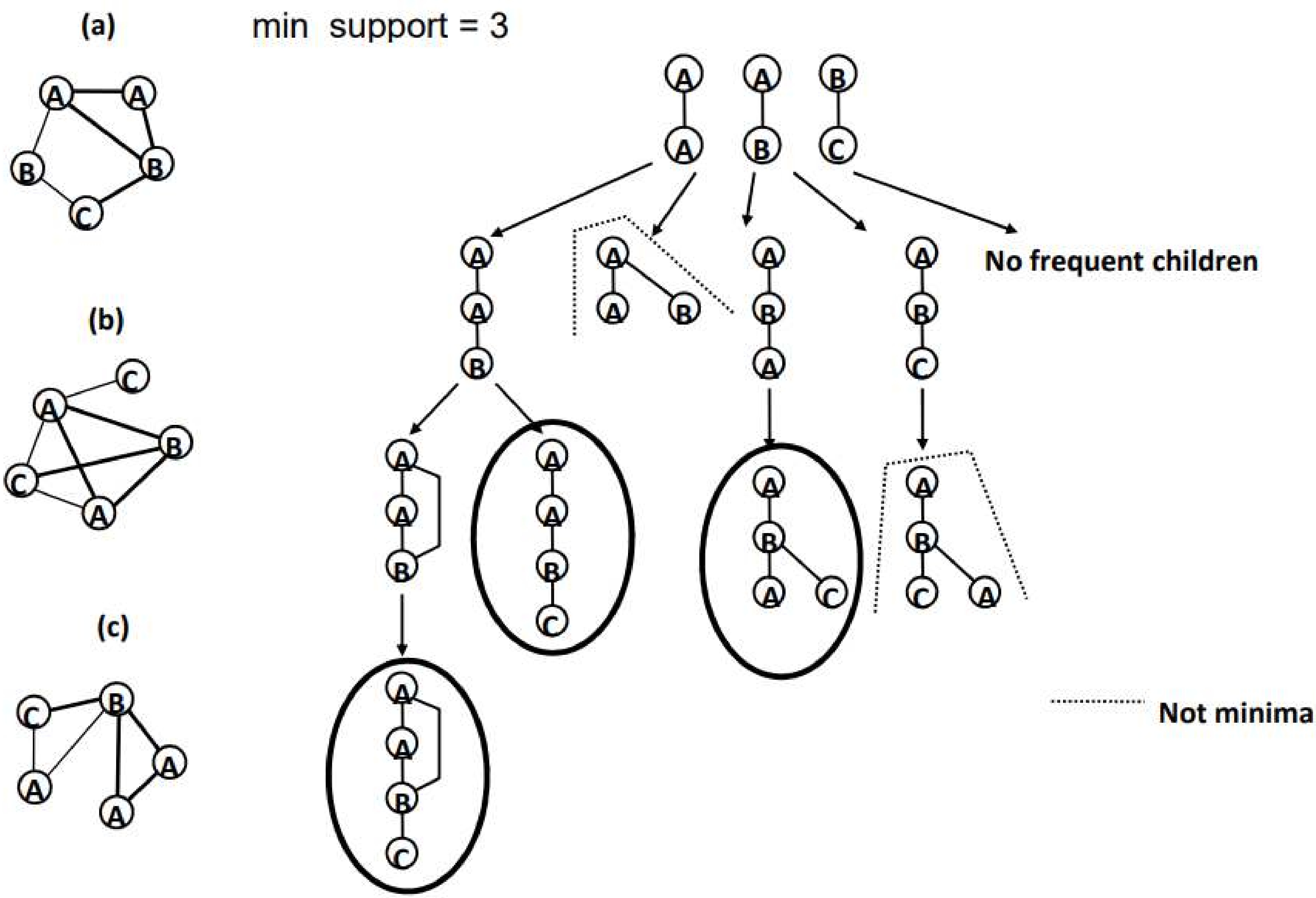
Frequent:



Infrequent:

Ví dụ

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**

Vấn đề bùng nổ mẫu đồ thị

- Nếu một đồ thị là phổ biến, tất cả các đồ thị con của nó phải phổ biến – thuộc tính Apriori
- Một đồ thị phổ biến n cạnh có thể có 2n đồ thị con.
- Trong số 422 hợp chất hóa học được xác nhận là có hoạt tính trong bộ dữ liệu sàng lọc kháng vi-rút AIDS, có 1.000.000 mẫu đồ thị phổ biến nếu mức độ trợ tối thiểu là 5%.

Thuật toán Subdue

- Là một thuật toán tham lam để tìm một số đồ thị con phổ biến nhất.
- Phương pháp này không đầy đủ, tức là nó có thể không tìm được tất cả các đồ thị con phổ biến, nhưng quá trình thực thi nhanh chóng.
- Dựa trên chiều dài mô tả: nén đồ thị sử dụng các mẫu đồ thị
  - Sử dụng mẫu mà cho nén cực đại.
- Dựa trên Beam Search - như BFS, nó tiến triển theo từng mức. Tuy nhiên, nó không giống như BFS, tìm kiếm dạng tia di chuyển xuống chỉ thông qua W nút tốt nhất tại mỗi cấp. Các nút khác bị bỏ qua.

Tài liệu tham khảo

- 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