Khả tuần tự View (View Serializability)

Mục tiêu thực sự của chúng ta trong thiết kế một trình lập lịch là *chỉ cho phép các lịch có thể tuần tự hóa.* Chúng ta cũng thấy cách các khác biệt trong các giao dịch hoạt động áp dụng cho dữ liệu có thể ảnh hưởng đến việc một lịch nhất định có thể tuần tự hóa hay không. Các trình lập lịch thường áp dụng lịch **khả tuần tự xung đột** (conflict serializability), đảm bảo khả năng tuần tự hóa bất kể các giao dịch làm gì với dữ liệu của chúng.

Tuy nhiên, có những điều kiện yếu hơn conflict-serializability cũng đảm bảo khả năng tuần tự hóa, được gọi là **khả tuần tự View (View-serializability).** View-serializability xem xét tất cả các kết nối giữa các giao tác T và U sao cho T ghi (write) một phần tử cơ sở dữ liệu *(đơn vị dữ liệu)* có giá trị U đọc được.

Sự khác biệt chính giữa View-serializability và conflict serializability xuất hiện khi *một giao tác T **ghi** một giá trị A mà không có giao tác nào khác đọc được* (vì một số giao tác khác sau đó ghi giá trị của riêng nó cho A) *(~ ghi mù – blind write).* Trong trường hợp đó, hành động $w_T(A)$ có thể được đặt ở một số vị trí khác của lịch (nơi A cũng không bao giờ được đọc) mà sẽ không được phép theo định nghĩa về khả năng tuần tự hóa xung đột.

Trong nội dung này, chúng ta sẽ định nghĩa khả tuần tự View một cách đúng nhất và kiểm tra nó.

1. Tương đương View (View Equivalence)

Giả sử chúng ta có hai lịch S_1 và S_2 của cùng một tập hợp các giao dịch.

Có một giao dịch giả định T_0 (T_b) đã *ghi các giá trị khởi tạo cho mỗi phần tử* cơ sở dữ liệu *(đơn vị dữ liệu)* được *đọc bởi bất kỳ giao dịch nào* trong các lịch và

Một giao dịch giả định khác T_f *đọc mọi phần tử được ghi bởi một hoặc nhiều giao dịch* sau khi mỗi lịch kết thúc.

Sau đó, đối với mọi hành động đọc $r_i(A)$ trong một trong các lịch trình, chúng ta có thể tìm thấy hành động ghi w_j (A) **gần nhất trước** hành động đọc đang đề cập. Ta nói **T_j là nguồn của hành động đọc $r_i(A)$.**

Lưu ý rằng giao dịch T_j có thể là giao dịch giả định ban đầu T_0 và Ti có thể là T_f.

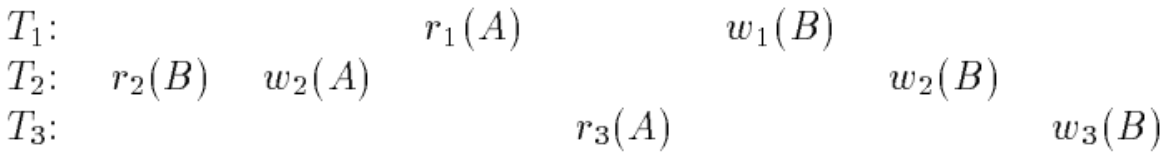
Nếu đối với mọi hành động đọc trong một trong các lịch trình, nguồn của nó giống nhau trong lịch trình kia, chúng ta nói rằng **S_1 và S_2 tương đương View.** Chắc chắn, các lịch tương đương View thực sự tương đương; mỗi lịch trình đều thực hiện giống nhau khi được thực hiện trên bất kỳ trạng thái cơ sở dữ liệu nào.

Nếu một lịch S tương đương View với một lịch tuần tự, chúng ta nói S là khả tuần tự View.

Ví dụ 1. Giả sử lịch S được định nghĩa như sau:

T_1	T_2	T_3
	r(B)	
	w(A)	
r(A)		
		r(A)
w(B)		
	w(B)	
		w(B)

Hoặc trình bày như dưới đây:



Ta tách các hành động của từng giao dịch theo chiều dọc, để *chỉ ra rõ hơn giao dịch nào thực hiện gì*; bạn nên đọc lịch trình từ trái sang phải.

Trong S, cả T_1 và T_2 đều ghi các giá trị của B bị mất; chỉ giá trị của B do T_3 ghi mới tồn tại đến cuối lịch trình và được đọc bởi giao dịch giả định T_f.

S không thể khả tuần tự xung đột. Để xem lý do, trước tiên hãy lưu ý rằng

T_2 ghi A trước khi T_1 đọc A, do đó **T_2 phải đứng trước T_1** trong một lịch tuần tự tương đương xung đột giả định.

Thực tế là hành động $w_1(B)$ đứng trước $w_2(B)$ cũng buộc **T_1 phải đứng trước T_2** trong bất kỳ lịch tuần tự tương đương xung đột nào.

(Hai điều kiện trên mâu thuẫn với nhau, vì không thể có thứ tự tuần tự tương đương vừa thỏa mãn T_2 đứng trước T_1 vừa T_1 đứng trước T_2. Do đó, S không khả tuần tự xung đột).

Tuy nhiên, *cả $w_1(B)$ và $w_2(B)$ đều không có tác động dài hạn nào đến cơ sở dữ liệu.* **Những loại ghi không liên quan này khả tuần tự View có thể bỏ qua** khi xác định các ràng buộc thực sự trên một lịch trình tuần tự tương đương.

Chính xác hơn, chúng ta hãy **xem xét các nguồn của tất cả các lần đọc** trong S:

1. Nguồn của $r_2(B)$ là T_0, vì không có lần ghi trước nào của B trong S.
2. Nguồn của $r_1(A)$ là T_2, vì T_2 gần đây nhất đã ghi A trước khi đọc.
3. Tương tự, nguồn của $r_3(A)$ là T_2.
4. Nguồn của lần đọc giả định của A bởi T_f là T_2.
5. Nguồn của lần đọc giả định của B bởi T_f là T_3, giao tác thực hiện thao tác ghi B cuối cùng.

Tất nhiên, *T_0 xuất hiện trước tất cả các giao dịch thực trong bất kỳ lịch trình nào và T_f xuất hiện sau tất cả các giao dịch.*

Nếu chúng ta sắp xếp các giao dịch thực (T_2, T_1, T_3), thì nguồn của tất cả các lần đọc đều giống như trong lịch S. *(Giả sử có một lịch tuần tự S' tuân theo thứ tự thực hiện các giao dịch như trên, ta sẽ tìm xem lịch S' có tương đương với S không, nếu có, thì S và S' là tương đương View, dẫn đến S là lịch khả tuần tự).*

Nghĩa là, T_2 đọc B và chắc chắn T_0 thực hiện thao tác ghi trước đó. T_1 đọc A, nhưng T_2 đã ghi A, do đó nguồn của $r_1(A)$ là T_2, như trong S.

T_3 cũng đọc A, nhưng vì T_2 trước đó đã ghi A, đó là nguồn của $r_3(A)$, như trong S.

Cuối cùng, T_f giả định đọc A và B, nhưng giao tác thực hiện thao tác ghi cuối cùng của A và B trong lịch (T_2; T_1; T_3) lần lượt là T_2 và T_3, cũng như trong S.

Ta kết luận rằng S là một lịch **khả tuần tự View** và lịch được biểu diễn bằng thứ tự (T_2; T_1; T_3) là một **lịch tương đương View.** □

2. Đồ thị phức cho lịch khả tuần tự View (Polygraphs for View-Serializability)

Có một phương pháp khái quát của **đồ thị thứ tự ưu tiên (precedence graph)** *(phương pháp mà ta đã sử dụng để kiểm tra khả tuần tự xung đột)* phản ánh tất cả các ràng buộc thứ tự ưu tiên được yêu cầu theo định nghĩa về khả tuần tự View. Ta định nghĩa **đồ thị phức (polygraph)** cho một lịch bao gồm những điều sau:

1. **Một nút (node) cho mỗi giao tác** và các nút bổ sung **cho các giao dịch giả định T_0 (T_b) và T_f.**

2. Đối với **mỗi hành động $r_i(X)$ với nguồn T_j, vẽ một cung từ T_j đến T_i.**

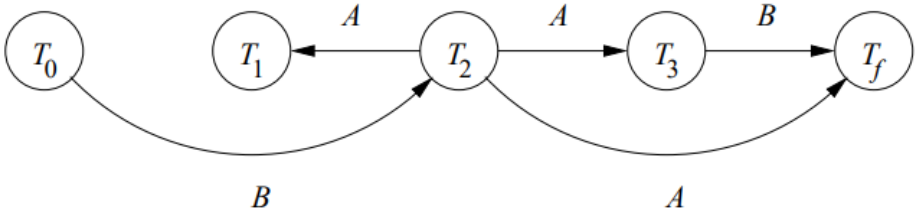
3. Giả sử **T_j là nguồn của lệnh đọc $r_i(X)$,** và **T_k là một giao tác khác thực hiện thao tác ghi của X. T_k không được phép chen** *(can thiệp – intervene)* **giữa T_j và T_i,** vì vậy nó phải xuất hiện trước T_j hoặc sau T_i. Ta biểu diễn điều kiện này bằng **một cặp cung** (được vẽ bằng nét đứt) từ **T_k đến T_j và từ T_i đến T_k.**

Theo trực giác (Intuitively), **một trong hai cung là "thực",** nhưng ta không quan tâm là cung nào, và khi **ta cố gắng làm cho đồ thị phức trở thành phi chu trình (không có chu trình),** chúng ta có thể **chọn bất kỳ cặp nào giúp làm cho nó phi chu trình (loại bỏ một trong hai cạnh tạo chu trình).**

Tuy nhiên, có những **trường hợp đặc biệt** quan trọng mà cặp cung trở thành một cung duy nhất:

(a) Nếu **T_j là T_0 (T_b),** *thì T_k không thể xuất hiện trước T_j,* vì vậy chúng ta sử dụng cung Ti → Tk thay cho cặp cung.

(b) Nếu **T_i là T_f,** *thì T_k không thể theo sau T_i,* vì vậy chúng ta sử dụng cung Tk → Tj thay cho cặp cung.



Hình 1. *Bước đầu tiên của phương pháp đồ thị phức minh họa Ví dụ 1.*

Ví dụ 2. Xem xét lịch trình S từ **Ví dụ 1**. Hình 1 biểu diễn phần đầu của đồ thị phức cho S, trong đó chỉ có **các nút và các cung từ quy tắc (2) được đặt.** Đồng thời cũng đã chỉ ra phần tử cơ sở dữ liệu *(đơn vị dữ liệu)* lập nên mỗi cung.

Nghĩa là, A được truyền từ T_2 đến T_1, T_3 và T_f, trong khi B được truyền từ T_0 đến T_2 và từ T_3 đến T_f.

Bây giờ, ta phải xem xét những giao tác nào có thể can thiệp vào từng "kết nối" này bằng cách *ghi cùng một đơn vị dữ liệu (element)* giữa chúng. Những can thiệp tiềm ẩn này bị loại trừ bởi các cặp cung từ quy tắc (3), mặc dù như chúng ta sẽ thấy, trong ví dụ này, *mỗi cặp cung liên quan đến một trường hợp đặc biệt và trở thành một cung duy nhất.*

Hãy xem xét cung **$T_2 \rightarrow T_1$** dựa trên đơn vị dữ liệu A. *Các giao tác thực hiện duy nhất việc ghi A là T_0 và T_2,* và không giao tác nào trong số chúng có thể chen giữa cung này *(theo quy tắc 3)*, vì T_0 không thể di chuyển vị trí của nó và T_2 đã là điểm kết thúc của cung. Do đó, không cần thêm cung (đỉnh) nào.

Một lập luận tương tự cho chúng ta biết, không cần thêm bất kỳ cung (đỉnh) nào để đảm bảo rằng **những giao tác khác thực hiện việc ghi A nằm ngoài các cung $T_2 \rightarrow T_3$ và $T_2 \rightarrow T_f$.**

(T_3 đọc A, và T_2 là giao tác gần nhất ghi A trước đó. T_f, giao tác giả định cuối cùng, đọc A, và T_2 là giao tác cuối cùng ghi A trước khi T_f đọc.

Giao tác khác thực hiện việc ghi A chỉ là T_0, thực hiện ghi A trước T_2.

Tuy nhiên, T_0 luôn đứng trước tất cả các giao tác khác trong lịch. Vì thế, T_0 không thể can thiệp hay làm ảnh hưởng đến thứ tự của các cung $T_2 \rightarrow T_3$ và $T_2 \rightarrow T_f$.)

Bây giờ hãy xem xét các cung dựa trên đơn vị dữ liệu B. Lưu ý rằng T_0, T_1, T_2 và T_3 đều ghi B.

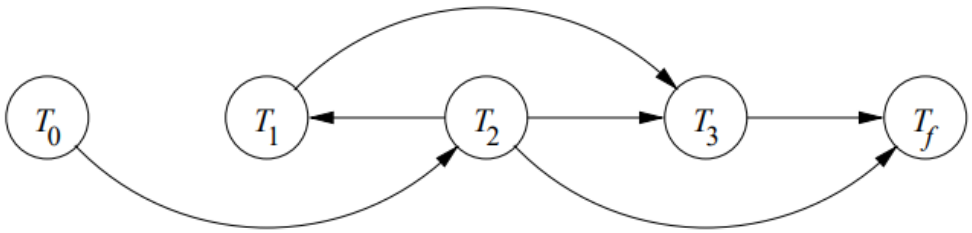
Trước tiên hãy xem xét cung **$T_0 \rightarrow T_2$.** T_1 và T_3 là các giao tác khác thực hiện ghi B; T_0 và T_2 cũng ghi B, nhưng như chúng ta đã thấy, các đỉnh của cung không thể gây "nhiễu" (interference), vì vậy chúng ta không cần xem xét chúng.

Vì chúng ta không thể đặt T_1 giữa T_0 và T_2 theo nguyên tắc (3), chúng ta cần cặp cung ($T_1 \rightarrow T_0$; $T_2 \rightarrow T_1$). Tuy nhiên, không có gì có thể đứng trước T_0, vì vậy lựa chọn **$T_1 \rightarrow T_0$ là không khả thi.** Trong trường hợp đặc biệt này, chúng ta chỉ cần thêm cung $T_2 \rightarrow T_1$ vào đồ thị phức. Nhưng cung này đã có sẵn của đơn vị dữ liệu A, vì vậy, chúng ta không thực hiện bất kỳ thay đổi nào đối với đồ thị phức để giữ T_1 bên ngoài cung $T_0 \rightarrow T_2$.

Chúng ta cũng không thể đặt T_3 giữa T_0 và T_2. Lý luận tương tự cho chúng ta biết rằng thêm cung $T_2 \rightarrow T_3$, thay vì một cặp cung. Tuy nhiên, cung này cũng đã có trong đồ thị do thuộc đơn vị dữ liệu A, vì vậy chúng ta không thực hiện bất kỳ thay đổi nào.

Tiếp theo, hãy xem xét cung **$T_3 \rightarrow T_f$.** Vì T_0, T_1 và T_2 là những giao tác khác thực hiện viết B, chúng ta phải giữ chúng bên ngoài cung này. T_0 không thể nằm giữa T_3 và T_f, nhưng T_1 hoặc T_2 thì có thể. Vì không ai trong số chúng có thể nằm sau T_f, chúng ta phải buộc T_1 và T_2 xuất hiện trước T_3.

Đã có một cung $T_2 \rightarrow T_3$, nhưng chúng ta phải **thêm cung $T_1 \rightarrow T_3$** vào đồ thị. Sự thay đổi này là cung duy nhất chúng ta phải thêm vào đồ thị, và tập hợp các cung của đồ thị được biểu diễn trong Hình 2. □



Hình 2. *Đồ thị phức hoàn chỉnh cho ví dụ 2.*

Ví dụ 3. Trong Ví dụ 2, tất cả các cặp cung đều là các cung đơn lẻ như một trường hợp đặc biệt. Hình 3 là một ví dụ về lịch trình gồm bốn giao dịch trong đó có một cặp cung thực sự trong đồ thị phức.

T_1	T_2	T_3	T_4
	$r_2(A);$		
$r_1(A);\ w_1(C);$			
		$r_3(C);$	
$w_1(B);$			
			$r_4(B);$
		$w_3(A);$	
			$r_4(C);$
	$w_2(D);\ r_2(B);$		
			$w_4(A);\ w_4(B);$

Hình 3. *Ví dụ về các giao tác làm cho đồ thị phức cần một cặp cung.*

Hình 4 cho thấy đồ thị phức chỉ có các cung xuất phát từ các kết nối từ nguồn đến thao tác đọc tương ứng. Như trong Hình 1, chúng ta gắn nhãn *(label)* mỗi cung theo (các) đơn vị dữ liệu thuộc cung đó.

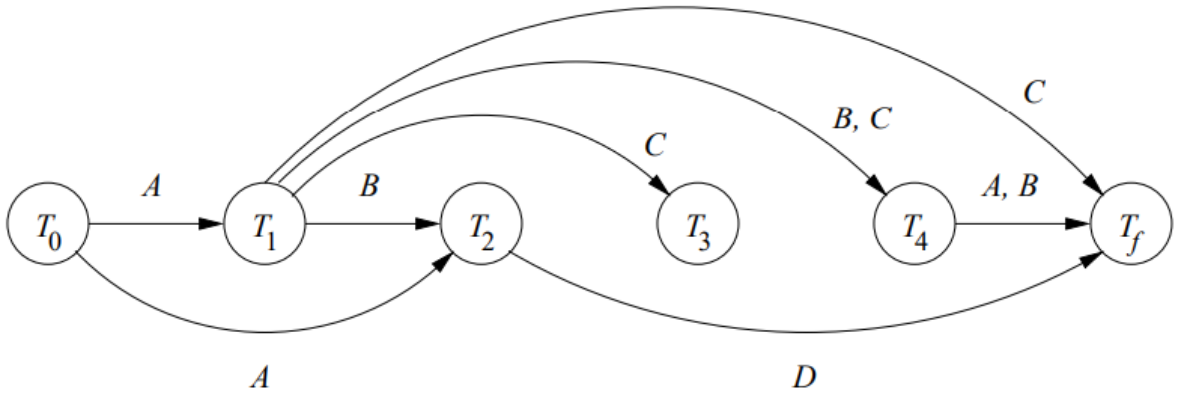
Sau đó, chúng ta phải xem xét các cách có thể để thêm cặp cung. Như chúng ta đã thấy trong **Ví dụ 2**, có một số cách đơn giản hóa mà chúng ta có thể thực hiện. Khi tránh sự can thiệp vào cung $T_j \rightarrow T_i$, duy nhất các giao tác cần được coi là T_k (giao dịch không thể ở giữa) là những giao tác thỏa:

- **Thao tác ghi của một đơn vị dữ liệu** đã tạo ra cung này Tj → Ti,
- **Nhưng không phải T_0** (T_b) **hoặc T_f** – những giao tác không bao giờ có thể là Tk, và
- **Không phải T_i hoặc T_j,** là các đỉnh của chính cung đó.

Với các quy tắc này, chúng ta hãy xem xét các cung thực hiện trên đơn vị dữ liệu A, được viết bởi T_0, T_3 và T_4.

Chúng ta không cần phải xem xét T_0. T_3 không được nằm giữa $T_4 \rightarrow T_f$, vì vậy chúng ta **thêm cung $T_3 \rightarrow T_4$;** hãy nhớ rằng cung còn lại trong cặp, *$T_f \rightarrow T_3$ không phải là một lựa chọn đúng.*

Tương tự như vậy, T_3 không được nằm giữa $T_0 \rightarrow T_1$ hoặc $T_0 \rightarrow T_2$, dẫn đến các cung **$T_1 \rightarrow T_3$ và $T_2 \rightarrow T_3$.**



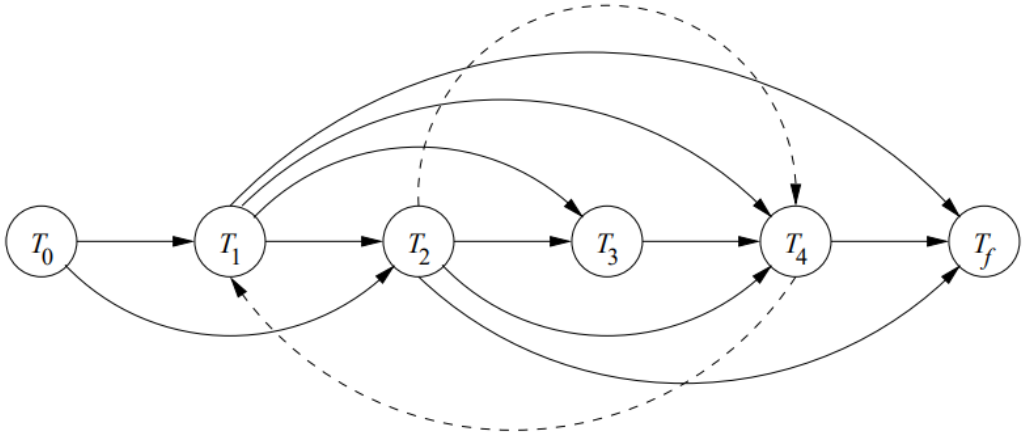
Hình 4. *Bước đầu cho việc vẽ đồ thị phức minh họa Ví dụ 3.*

Bây giờ, hãy xem xét, thực tế là T_4 cũng không được nằm giữa một cung do A. Đó là đỉnh của $T_4 \rightarrow T_f$, do đó cung đó không liên quan. T_4 không được nằm giữa $T_0 \rightarrow T_1$ hoặc $T_0 \rightarrow T_2$, do đó tạo ra các **cung $T_1 \rightarrow T_4$ và $T_2 \rightarrow T_4$.**

Tiếp theo, chúng ta hãy xem xét các cung thực hiện đơn vị dữ liệu B, được viết bởi T_0, T_1 và T_4.

Một lần nữa, chúng ta không cần xem xét T_0. Các cung duy nhất do B là $T_1 \rightarrow T_2$, $T_1 \rightarrow T_4$, và $T_4 \rightarrow T_f$. **T_1** không thể nằm giữa hai cung đầu tiên, *nhưng cung thứ ba yêu cầu thêm* ***cung $T_1 \rightarrow T_4$.***

T_4 chỉ có thể nằm giữa $T_1 \rightarrow T_2$. Cung này không có đỉnh nào ở T_0 hoặc T_f, do đó thực sự cần một cặp cung **($T_4 \rightarrow T_1$, $T_2 \rightarrow T_4$)**. Ta vẽ cặp cung này, cũng như tất cả các cung khác được thêm vào, như trong Hình 5.



Hình 5. *Đồ thị phức hoàn chỉnh của Ví dụ 3*

Tiếp theo, hãy xem xét các giao tác thực hiện ghi C: T_0 và T_1.

T_0 ta không cần quan tâm như giải thích trên. Ngoài ra, T_1 là một phần của mọi cung thuộc đơn vị dữ liệu C, vì vậy nó *không thể nằm ở giữa.*

Tương tự, D chỉ được ghi bởi T_0 và T_2, vì vậy chúng ta có thể xác định rằng *không cần thêm cung nào nữa.*

Do đó, đồ thị phức cuối cùng là máy trong Hình 5. □

3. Kiểm tra khả tuần tự View

Vì chúng ta chỉ phải chọn một cung trong mỗi cặp cung, nên chúng ta có thể tìm thấy một thứ tự tuần tự tương đương cho lịch trình S nếu và chỉ nếu có một số lựa chọn từ mỗi cặp cung biến đồ thị phức của S thành một **đồ thị phi chu trình.**

Để xem lý do tại sao, hãy lưu ý rằng nếu có một đồ thị phi chu trình như vậy, thì bất kỳ phép sắp xếp tôpô nào của đồ thị đều đưa ra một thứ tự mà *không có thao tác ghi nào có thể xuất hiện giữa thao tác đọc và nguồn của nó, và mọi thao ghi xuất hiện trước các thao tác đọc tương ứng của nó.* Do đó, các kết nối thao tác đọc-nguồn trong thứ tự tuần tự hoàn toàn giống như trong S; hai lịch tương đương View, và do đó **S có thể khả tuần tự View.**

Ngược lại (Conversely), nếu S có thể khả tuần tự View, thì có một thứ tự tuần tự tương đương View S'. Mỗi cặp cung ($T_k \rightarrow T_j$, $T_i \rightarrow T_k$) trong đồ thị phức của S phải có T_k trước T_j hoặc sau T_i trong S'; nếu không, việc viết bởi T_k sẽ phá vỡ kết nối từ T_j đến T_i, nghĩa là **S và S' không tương đương View.**

Tương tự như vậy, mọi cung trong đồ thị phức phải tuân theo thứ tự giao tác của S'. Chúng ta kết luận rằng có một lựa chọn cung từ mỗi cặp cung khiến đồ thị phức thành một đồ thị mà thứ tự tuần tự S' nhất quán với mỗi cung của đồ thị. Do đó, đồ thị này là phi chu trình.

Ví dụ 4. Xem xét đồ thị phức của Hình 2. Nó đã là một đồ thị, và nó là phi chu trình. Thứ tự tôpô duy nhất là (T_2; T_1; T_3), do đó là thứ tự tuần tự tương đương View cho lịch của Ví dụ 2.

Bây giờ hãy xem xét đồ thị phức của Hình 5. Chúng ta phải xem xét từng cung từ một cặp cung để lựa chọn xem có nên giữ lại không.

- Nếu chúng ta chọn $T_4 \rightarrow T_1$, thì sẽ có một chu trình.
- Tuy nhiên, nếu chúng ta chọn **$T_2 \rightarrow T_4$, thì kết quả là một đồ thị phi chu trình.**

Thứ tự tôpô duy nhất cho đồ thị này là (T_1; T_2; T_3; T_4). Thứ tự này tạo ra thứ tự tuần tự tương đương View và cho thấy lịch ban đầu có thể khả tuần tự View.

4. Bài tập cho phần 2

2.1 Vẽ đồ thị phức và tìm tất cả thứ tự tuần tự tương đương View cho các lịch sau:

* a) $r_1(A)$; $r_2(A)$; $r_3(A)$; $w_1(B)$; $w_2(B)$; $w_3(B)$.

b) $r_1(A)$; $r_2(A)$; $r_3(A)$; $r_4(A)$; $w_1(B)$; $w_2(B)$; $w_3(B)$; $w_4(B)$.

2.2 Dưới đây là một vài lịch tuần tự. Hãy tìm xem có bao nhiêu lịch *(i) khả tuần tự xung đột, (ii) khả tuần tự View* với mỗi lịch dưới đây.

* a) $r_1(A)$; $w_1(B)$; $r_2(A)$; $w_2(B)$; $r_3(A)$; $w_3(B)$; đây là 3 giao tác, mỗi giao tác đọc A rồi ghi B.

b) $r_1(A)$; $w_1(B)$; $w_1(C)$; $r_2(A)$; $w_2(B)$; $w_2(C)$; đây là 2 giao tác, mỗi giao tác đọc A rồi ghi B và C..

Tại sao tồn tại chu trình trong đồ thị thì không khả tuần tự View và cũng không khả tuần tự xung đột?

Khi tồn tại **chu trình trong đồ thị**, không thể tìm được một thứ tự tuần tự hóa S' tương đương với S.

- **Chu trình biểu thị mâu thuẫn thứ tự:**
  - Ví dụ: Nếu có chu trình $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_1$:
    - T_1 phải xảy ra trước T_2 (cung $T_1 \rightarrow T_2$).
    - T_2 phải xảy ra trước T_3 (cung $T_2 \rightarrow T_3$).
    - T_3 phải xảy ra trước T_1 (cung $T_3 \rightarrow T_1$).
  - Điều này là **mâu thuẫn logic**, vì không thể sắp xếp tuần tự hóa sao cho T_1, T_2, và T_3 thỏa mãn tất cả các cung.
- **Vi phạm View Serializable:**
  - Khi không thể sắp xếp thứ tự giao tác, sẽ xảy ra tình trạng thao tác ghi hoặc đọc trong S không tương đương với bất kỳ lịch trình tuần tự hóa nào S'.
  - **Ví dụ:** Một thao tác đọc có thể đọc giá trị từ một giao tác bị xung đột trong chu trình (đọc từ một giá trị "chưa ổn định"), gây ra kết quả không tương thích với bất kỳ lịch trình tuần tự nào.
- **Vi phạm Conflict Serializable:**
  - Khi không thể giải quyết các xung đột theo thứ tự hợp lệ, lịch trình không thể tương đương với bất kỳ lịch trình tuần tự nào.
  - Chu trình trong đồ thị xung đột là minh chứng cho việc không có thứ tự hợp lệ.

Tại sao lại có thể loại bỏ 1 trong 2 cung bất kỳ thuộc một cặp cung mà nó tạo chu trình trong đồ thị phức? Cơ chế nào loại bỏ hay chỉ cần chọn bất kỳ 1 trong 2 cung?

Tại sao có thể loại bỏ một cạnh trong cặp cạnh để phá chu trình?

Trong đồ thị phức (Serialization Graph), các cặp cạnh xuất hiện trong trường hợp có **xung đột về thứ tự giao tác** liên quan đến một thao tác trung gian. Ví dụ:

- Giả sử có hai cạnh $T_k \rightarrow T_j$ và $T_i \rightarrow T_k$, nhưng T_k xuất hiện trong cả hai cạnh, tạo nên một chu trình khi kết hợp với các cạnh khác.
- Loại bỏ **một trong hai cạnh** nghĩa là quyết định "không xem xét một mối quan hệ phụ thuộc cụ thể" để tránh chu trình.

Bằng cách này:

- Chỉ cần **một trong hai mối quan hệ phụ thuộc** được duy trì, thứ tự tuần tự hóa vẫn có thể tồn tại nếu không còn chu trình.
- Miễn là ta chọn cẩn thận, đồ thị kết quả vẫn phản ánh các phụ thuộc cần thiết và có thể xác định được một lịch tuần tự hóa tương đương.

- Việc loại bỏ một cung trong chu trình tương đương với việc *ép buộc* một thứ tự giữa hai giao tác liên quan đến cặp cung đó.
- Ví dụ, nếu ta loại bỏ cung $T_1 \rightarrow T_2$, điều đó có nghĩa là ta đang giả định rằng T_2 *không* phụ thuộc vào T_1 theo cách mà cung đó biểu diễn. Nói cách khác, ta đang xét trường hợp mà T_1 được thực hiện *trước* T_2.
- Bằng cách loại bỏ *từng* cung trong chu trình và kiểm tra xem đồ thị kết quả có còn chu trình hay không, ta đang xét *tất cả* các khả năng thứ tự tương đối giữa các giao tác trong chu trình đó.
- Việc loại bỏ một cung không phải là một "phép màu" loại bỏ xung đột. Nó là một cách để *xét tất cả các khả năng thứ tự* giữa các giao tác trong chu trình. Nếu *tồn tại* một cách loại bỏ cung (tức là tồn tại một thứ tự giao tác) mà đồ thị không còn chu trình, thì lịch đó khả tuần tự view. Nếu *không có* cách nào loại bỏ cung mà đồ thị hết chu trình, thì lịch đó không khả tuần tự view.

Cơ chế loại bỏ và việc chọn cung:

- **Không có cơ chế tự động "loại bỏ" cung nào cả.** Việc này là một bước trong thuật toán kiểm tra tính tuần tự xem.
- **Cần xét tất cả các khả năng:** Để xác định chắc chắn lịch sử có tuần tự xem hay không, ta cần thử loại bỏ *từng* cung trong mỗi chu trình và kiểm tra xem đồ thị kết quả có còn chu trình hay không. Nếu *tồn tại* một cách loại bỏ cung mà đồ thị không còn chu trình, thì lịch sử đó tuần tự xem.
- **Chọn bất kỳ cung nào trong cặp cung:** Về mặt lý thuyết, bạn có thể chọn bất kỳ cung nào trong cặp cung tạo chu trình để loại bỏ. Tuy nhiên, việc này cần được thực hiện một cách có hệ thống để xét *tất cả* các khả năng.

Tính chất của khả tuần tự View và khả tuần tự xung đột

- **Khả tuần tự view:** Một lịch được coi là khả tuần tự view nếu nó tương đương với ***ít nhất một*** thứ tự tuần tự của các giao tác.
- **Khả tuần tự xung đột:** Một lịch được coi là khả tuần tự xung đột nếu nó có thể được chuyển đổi thành một thứ tự tuần tự bằng cách hoán đổi các thao tác *không xung đột*.

Chính xác là, khả tuần tự xung đột (conflict serializability) yêu cầu lịch phải tương đương với ***mọi*** thứ tự tuần tự **có thể có được bằng cách hoán đổi các thao tác không xung đột.**

Điều này mạnh hơn so với yêu cầu của khả tuần tự view.

(Nguồn: ChatGPT, Gemini)