

PHẦN MỞ ĐẦU

Việc tăng số lượng các phần tử trong một hệ phân tán điều này cũng có nghĩa là tăng nguy cơ có một vài phần tử gặp lỗi trong quá trình thực thi một thuật toán phân tán. Các lỗi có thể là máy tính trong một mạng có thể hỏng, các process trong một hệ thống có thể ngừng thực hiện do các máy trạm bị tắt, hoặc máy tính có thể cho ra các kết quả không đúng, ví dụ như bộ nhớ bị lỗi. Những máy tính hiện đại càng ngày càng đáng tin cậy, do đó giảm đi sự xảy ra lỗi trong từng máy tính cá nhân. Tuy nhiên, cơ hội xảy ra lỗi ở một vài nơi trong một hệ thống phân tán là lớn bất kỳ khi số lượng các phần tử tăng. Để tránh phải khởi động lại một thuật toán mỗi lần lỗi xuất hiện, các thuật toán phải được thiết kế để chịu những lỗi như thế.

Sự hư hại do lỗi còn liên quan đến việc tính toán tuần tự, đến các ứng dụng không chú trọng đến sự an toàn, hoặc một tính toán chạy trong một thời gian dài và sinh ra những kết quả không thể kiểm tra được. Việc kiểm tra nội bộ có thể xử lý một vài loại lỗi (ví dụ kiểm tra chia cho không, kiểm tra giá trị nhập vào có là số hợp lệ không,...), nhưng không thể bảo vệ cho việc mất toàn bộ chương trình (máy tính bị tắt nguồn) hoặc do sự thay đổi trong chính các lệnh của chương trình. Do đó khả năng chịu lỗi của các thuật toán tuần tự là hạn chế.

Trong luận văn này sẽ trình bày các thuật toán giải quyết lỗi dạng như sau: trong khi thực hiện tính toán có một vài process ngưng hoạt động, thuật toán vẫn hoạt động. Khi thuật toán kết thúc kết quả phải bảo đảm thỏa một số điều kiện cho trước. Các thuật toán được xây dựng cho các bài toán cụ thể sau : bài toán quyết định, bài toán tô sáu màu cho đồ thị phẳng, bài toán giải gần đúng hệ phương trình tuyến tính bằng phương pháp lặp đơn.

CHƯƠNG 1: THUẬT TOÁN PHÂN TÁN

1.1 Hệ thống phân tán:

Một hệ thống phân tán bao gồm một tập tất cả các trạng thái có thể có của hệ thống (system), hệ thống sẽ thay đổi trạng thái của nó trên tập trạng thái này, và một tập các trạng thái ban đầu. Ta sẽ gọi trạng thái của một process trong hệ thống là *trạng thái*, và trạng thái của hệ thống *cấu hình*.

Định nghĩa 1.1.1 Một hệ thống phân tán là một bộ $S=(C, \rightarrow, I)$, với C là tập các cấu hình của hệ thống (sau này gọi tắt là cấu hình), \rightarrow là một quan hệ nhị phân trên C , và I là tập các cấu hình ban đầu.

Thay vì viết $(\gamma, \delta) \in \rightarrow$, $\gamma, \delta \in C$, ta sẽ viết là $\gamma \rightarrow \delta$.

Định nghĩa 1.1.2 Cho $S=(C, \rightarrow, I)$. Một thực thi của S (execution of S) là dãy tối đại $E=(\gamma_0, \gamma_1, \dots)$ với $\gamma_i \rightarrow \gamma_{i+1}$, $\forall i=0, 1, 2, \dots$, và $\gamma_0 \in I$.

Một cấu hình kết thúc γ là cấu hình mà không tồn tại cấu hình δ sao cho $\gamma \rightarrow \delta$. Vậy $E=(\gamma_0, \gamma_1, \dots)$ là dãy vô hạn hoặc là dãy được kết thúc bởi cấu hình kết thúc.

Định nghĩa 1.1.3 Một cấu hình δ được nói là có thể đạt được từ γ , ký hiệu $\gamma \mapsto \delta$, nếu tồn tại một dãy $\gamma = \gamma_0, \gamma_1, \gamma_2, \dots, \gamma_k = \delta$ với $\gamma_i \rightarrow \gamma_{i+1}$, $\forall i=0, 1, 2, \dots, k$. Cấu hình δ được nói là có thể đạt được nếu nó có thể đạt được từ một cấu hình thuộc I .

1.2 Hệ thống phân tán với sự trao đổi thông điệp không đồng bộ :

Ta xét một hệ phân tán bao gồm một tập các process, ký hiệu \mathbf{P} , và một hệ thống trao đổi thông điệp giữa các process (communication subsystem). Mỗi một process là một *máy tính*, là một automata. Để tránh nhầm lẫn ta sẽ sử dụng “dịch chuyển” (transition) và “cấu hình” (configuration) cho cả hệ thống và “sự kiện” (event) và “trạng thái” (state) cho process. Gọi M là tập các thông điệp (message) được truyền giữa các process.

Định nghĩa 1.2.1 Thuật toán địa phương của một process p là một bộ $(Z_p, I_p, \rightarrow_p^i, \rightarrow_p^s, \rightarrow_p^r)$, với Z_p là tập các trạng thái, $I_p \subseteq Z_p$ là tập các trạng thái ban đầu,

\rightarrow_p^i là quan hệ trên Z^*Z , \rightarrow_p^s và \rightarrow_p^r là các quan hệ trên Z^*M^*Z . Một quan hệ nhị phân \rightarrow_p trên Z được định nghĩa bởi

$$c \rightarrow_p d \Leftrightarrow (c, d) \in \rightarrow_p^i \vee (\exists m \in M : (c, m, d) \in \rightarrow_p^s \cup \rightarrow_p^r).$$

Các quan hệ \rightarrow_p^i , \rightarrow_p^s , và \rightarrow_p^r là những chuyển đổi trạng thái trong process p với \rightarrow_p^i là chuyển đổi trạng thái nội bộ, không nhận hay gửi thông điệp, \rightarrow_p^s chuyển đổi trạng thái và gửi thông điệp, và cuối cùng là \rightarrow_p^r chuyển đổi trạng thái và nhận thông điệp.

Định nghĩa 1.2.2 Một thuật toán phân tán (distributed algorithm) trên $P=\{p_1, \dots, p_N\}$ là tập các thuật toán địa phương của các p_i .

Mỗi cấu hình của hệ thống là một vector có dạng $\gamma = (s_1, \dots, s_N)$, với s_i là trạng thái của process p_i .

Định nghĩa 1.2.3 Một sự dịch chuyển cấu hình theo phương thức truyền thông tin không đồng bộ dựa trên một thuật toán phân tán trên p_1, \dots, p_N (với thuật toán địa phương của một process p_i là một bộ $(Z_{p_i}, I_{p_i}, \rightarrow_{p_i}^i, \rightarrow_{p_i}^s, \rightarrow_{p_i}^r)$), là $S = (C, \rightarrow, I)$ với

- 1) $C = \{(c_{p1}, \dots, c_{pN}, M) : (\forall p \in P : c_p \in Z_p) \text{ và } M \subseteq M\}$,
- 2) $\rightarrow = (\cup_{p \in P} \rightarrow_p)$, với \rightarrow_p là hàm chuyển đổi trạng thái của process p ; \rightarrow_{p_i} là tập các cặp

$$(c_{p1}, \dots, c_{p_i}, \dots, c_{pN}, M_1), (c_{p1}, \dots, c'_{p_i}, \dots, c_{pN}, M_2)$$

chúng thỏa một trong ba điều sau

- $(c_{p_i}, c'_{p_i}) \in \rightarrow_{p_i}^i$ và $M_1 = M_2$
- có $m \in M$, $(c_{p_i}, m, c'_{p_i}) \in \rightarrow_{p_i}^s$ và $M_2 = M_1 \cup \{m\}$,
- có $m \in M$, $(c_{p_i}, m, c'_{p_i}) \in \rightarrow_{p_i}^r$ và $M_1 = M_2 \cup \{m\}$,

- 3) $I = \{(c_{p1}, \dots, c_{pN}, M) : (\forall p \in P : c_p \in I_p) \text{ và } M = \emptyset\}$.

Một thực thi của một thuật toán phân tán là sự chuyển đổi cấu hình dựa trên S . Các cặp $(c, d) \in \rightarrow_p^i$ được gọi là các sự kiện nội bộ (internal event) của p , và (c, m, d) thuộc \rightarrow_p^s và \rightarrow_p^r được gọi là các sự kiện gửi (send event) và sự kiện nhận (receive event) của p .

- Một sự kiện nội bộ e được cho bởi $e=(c, d)$ của p được nói là *áp dụng được* trong cấu hình $\gamma=(c_{p1}, \dots, c_p, \dots, c_{pN}, M)$ nếu $c_p=c$. Trong trường hợp này, $e(\gamma)$ được định nghĩa là cấu hình $(c_{p1}, \dots, d, \dots, c_{pN}, M)$.
- Một sự kiện gọi e được cho bởi $e=(c, m, d)$ của p được nói là *áp dụng được* trong cấu hình $\gamma=(c_{p1}, \dots, c_p, \dots, c_{pN}, M)$ nếu $c_p=c$. Trong trường hợp này, $e(\gamma)$ được định nghĩa là cấu hình $(c_{p1}, \dots, d, \dots, c_{pN}, M \cup \{m\})$.
- Một sự kiện nhận e được cho bởi $e=(c, m, d)$ của p được nói là *áp dụng được* trong cấu hình $\gamma=(c_{p1}, \dots, c_p, \dots, c_{pN}, M)$ nếu $c_p=c$. Trong trường hợp này, $e(\gamma)$ được định nghĩa là cấu hình $(c_{p1}, \dots, d, \dots, c_{pN}, M \setminus \{m\})$.

Ta luôn giả thiết rằng mỗi một thông điệp chỉ có một process là có thể nhận và mỗi thông điệp được gọi cho một process thì sau một khoảng thời gian hữu hạn process đó sẽ nhận được nếu process còn hoạt động.

1.3 Hệ thống phân tán với sự trao đổi thông điệp đồng bộ :

Truyền thông điệp được nói là đồng bộ nếu sự kiện gọi và biến cố nhận phối hợp với nhau một cách thống nhất. Một process p không được gọi thông điệp cho process q trừ khi q sẵn sàng nhận thông điệp. Ta có định nghĩa hình thức sau.

Định nghĩa 1.3.1 Một sự thay đổi cấu hình (transition system) theo phương thức truyền thông tin đồng bộ dựa trên một thuật toán phân tán trên p_1, \dots, p_N , là $S = (C, \rightarrow, I)$ với

1) $C = \{(c_{p1}, \dots, c_{pN}, M) : \forall p \in P : c_p \in Z_p\}$,

2) $\rightarrow = (\cup_{p \in P} \rightarrow_p) \cup (\cup_{p, q \in P : p \neq q} \rightarrow_{pq})$, với

- \rightarrow_{pi} là tập các cặp

$$(c_{p1}, \dots, c_{pi}, \dots, c_{pN}), (c_{p1}, \dots, c'_{pi}, \dots, c_{pN}) \text{ thỏa } (c_{pi}, c'_{pi}) \in \rightarrow_{pi}^i,$$

- $\rightarrow_{pi,qj}$ là tập các cặp

$$(\dots, c_{pi}, \dots, c_{pj}, \dots), (\dots, c'_{pi}, \dots, c'_{pj}, \dots)$$

thỏa có một thông điệp $m \in M$ sao cho

$$(c_{pi}, m, c'_{pi}) \in \rightarrow_{pi}^s \text{ và } (c_{pj}, m, c'_{pj}) \in \rightarrow_{pj}^r.$$

$$3) \ I = \{(c_{p1}, \dots, c_{pN}) : (\forall p \in P : c_p \in I_p)\}.$$

CHƯƠNG 2: SỰ CHỊU LỖI TRONG HỆ THỐNG PHÂN TÁN

2.1 Bài toán quyết định (Decision Problem) :

Bài toán quyết định được phát biểu như sau: là bài toán trong đó đòi hỏi mỗi process sau một thời gian hoạt động phải dừng lại và cho ra một quyết định bằng cách gán một giá trị vào một biến, ta gọi biến và giá trị này lần lượt là *biến quyết định* và *giá trị quyết định*, tính chất dừng này được gọi là tính chất *kết thúc* (termination). Biến nhận giá trị quyết định không được thay đổi giá trị khi đã được gán giá trị quyết định. Các giá trị quyết định giữa các process thường có một mối liên hệ nào đó, tính chất này được gọi là tính *ổn định* (consistency). Ngoài ra để tránh các bài toán tầm thường bài toán quyết định còn đòi hỏi tính chất *không tầm thường* (non-trivial).

Một ví dụ về bài toán quyết định tầm thường là sau khi nhận song giá trị vào ban đầu nào đó thì các process đồng loạt gán giá trị quyết định vào biến quyết định và kết thúc.

Bài toán quyết định là sự trừu tượng hóa một lớp các bài toán như:

- 1) *Commit-Abort*. Trong cơ sở dữ liệu phân tán khi một giao dịch cần được thực hiện trên các site liên quan thì một là thực hiện trên tất cả các site đó hoặc không site nào trong các site đó. Do đó sau khi được thông báo về một giao dịch thì các site sẽ xác định xem việc cập nhật trên nó có được không, nếu được thì gửi cho các site khác “yes”, ngược lại là “no”. Sau khi nhận được các tín hiệu từ các site khác , nó phải *quyết định* là có thực hiện cập nhật hay không.
- 2) *Độ tin cậy của tín hiệu đầu vào*. Giả sử tín hiệu vào là 0 hay 1. Để biết một tín hiệu vào có phải đúng là tín hiệu 1 hay không thì ta có thể nhận tín hiệu đó từ nhiều sensor. Sau đó gửi tín hiệu nhận được cho các process. Các process phải ra cùng quyết định nếu chúng nhận được cùng một tín hiệu, hoặc các tín hiệu phải thỏa một tiêu chí nào đó, ví dụ như về số lượng của một loại tín hiệu.
- 3) *Election*. Đây là một lớp các bài toán. Election được phát biểu chung như sau: Trong các process phải có một process quyết định và trở thành *leader* và các process khác cũng quyết định nhưng trở thành non-leader. Ví dụ (đây cũng là

một lớp chứa nhiều bài toán khác) giả sử mỗi process chứa một giá trị là số nguyên, tất cả đều khác nhau. Hãy tìm process mang giá trị lớn nhất.

Trong luận văn sẽ trình bày bài toán quyết định với các ý chính sau:

- Mỗi process nhận giá trị vào là 0 hoặc 1,
- Giá trị quyết định là 0 hoặc 1 tùy vào số lượng 0 hoặc 1 chiếm ưu thế,
- Các process chỉ nhận được một số lượng nhất định các tín hiệu từ các process khác. Ví dụ trong hệ thống mạng ta cần biết tính chất nào đó có phải là tính chất A hay không. Sau khi phát tín hiệu lên mạng mỗi máy trả lời một ý, yes/no. Vì không biết số máy tham gia là bao nhiêu (N) và máy tính phát tín hiệu chỉ có thể chờ trong một khoảng thời gian nào đó. Do đó số lượng tín hiệu nhận được chỉ là $d (\leq N)$. d sẽ được xác định trong phần lý thuyết để sự quyết định là đáng tin cậy.

Các giả thiết :

- Mỗi process thực hiện ít nhất một lần gửi thông điệp cho các process khác vẫn còn hoạt động.
- Một process vẫn còn hoạt động thì luôn nhận thông điệp khi thông điệp được gửi đến nó.

2.2 Khả năng chịu lỗi của các thuật toán quyết định :

Trong mục này ta sẽ nghiên cứu về khả năng chịu lỗi của các thuật toán phân tán dùng để giải quyết bài toán quyết định. Hai loại thuật toán sẽ được xem xét là đơn định (deterministic) và xác suất (Probabilistic) . Trong mục này cũng sẽ trình bày các chứng minh cho thấy các thuật toán đơn định, phân tán không có khả năng chịu lỗi. Thuật toán xác suất của Bracha và Toueg sẽ được đưa ra trong mục này để giải quyết bài toán đã trình bày ở trên.

2.2.1 Các khái niệm và các kết quả cơ bản:

Cho dãy các sự kiện $\sigma = (e_1, \dots, e_k)$ được nói là áp dụng được trong cấu hình γ nếu e_1 áp dụng được trong γ , e_2 áp dụng được trong $e_1(\gamma)$, và ...v.v. Nếu cấu hình kết quả khi áp dụng σ trong γ là δ , ta viết $\gamma \rightarrow^\sigma \delta$ hay $\sigma(\gamma)=\delta$. Nếu $S \subseteq \mathbb{D}$ và σ chỉ chứa các sự kiện của các process thuộc S thì ta viết $\gamma \rightarrow_S \delta$.

Mệnh đề 2.2.1.0 [3] *Gọi γ là cấu hình mà hai sự kiện e_p , và e_q của hai process khác nhau có thể áp dụng được. Khi đó e_p áp dụng được trong $e_q(\gamma)$, e_q áp dụng được trong $e_p(\gamma)$, và $e_p(e_q(\gamma))=e_q(e_p(\gamma))$ (e_p và e_q giao hoán).*

Mệnh đề 2.2.1.1 [4] *Gọi σ_1 và σ_2 là hai dãy áp dụng được trong cấu hình γ thỏa không tồn tại process chung (process p có sự kiện e_i trong σ_1 và có sự kiện e_j trong σ_2 là process chung). Khi đó σ_2 áp dụng được trong $\sigma_1(\gamma)$, σ_1 áp dụng được trong $\sigma_2(\gamma)$ và $\sigma_2(\sigma_1(\gamma))=\sigma_1(\sigma_2(\gamma))$.*

Định nghĩa 2.2.1.2 *Một thực thi được nói là t -crash fair nếu có ít nhất $N-t$ process có thể thực thi nhiều vô hạn các sự kiện (các process như vậy được gọi là các process đúng) và mỗi thông điệp được gửi tới một process đúng luôn được nhận.*

Định nghĩa 2.2.1.3 *Một thuật toán được nói là t -crash-robust consensus nếu nó thỏa các đòi hỏi sau:*

- 1) **Kết thúc (Termination).** *Với mọi thực thi t -crash fair, mọi process đúng đều quyết định (decide).*
- 2) **Đồng ý (Agreement).** *Nếu, trong một cấu hình đạt được, $y_p \neq b$ và $y_q \neq b$ trong các process đúng p và q thì $y_p=y_q$, với y_p, y_q là các biến quyết định.*
- 3) **Không tầm thường (Non-triviality).** *Với $v=0$ và $v=1$ tồn tại những cấu hình đạt được mà trong đó có ít nhất một process p sao cho $y_p=v$.*

Với $v=0, 1$ một cấu hình được nói là v -quyết định nếu có process p sao cho $y_p=v$; một cấu hình được nói là quyết định nếu nó là 0-quyết định hay 1-quyết định. Một cấu hình được nói là v -valent nếu mọi cấu hình quyết định đạt được từ nó là những v -quyết định. Một cấu hình được nói là bivalent nếu cả hai loại cấu hình 0-quyết định và 1-

quyết định đều đạt được từ nó, và được nói là *univalent* nếu nó là một trong 0-quyết định hoặc 1-quyết định.

Một cấu hình γ của một thuật toán t -robust được nói là một *fork* nếu tồn tại tập T có nhiều nhất t process, và hai cấu hình γ_0 và γ_1 , sao cho $\gamma \rightarrow_T \gamma_0$, $\gamma \rightarrow_T \gamma_1$, và γ_v là v -valent.

Mệnh đề 2.2.1.4 [5] *Với mỗi cấu hình đạt được γ của một thuật toán t -robust và mỗi tập con S có ít nhất $N-t$ process, tồn tại một cấu hình quyết định δ sao cho $\gamma \rightarrow_S \delta$.*

Chứng minh. Xét thực thi đạt tới cấu hình γ và chứa nhiều tùy ý các sự kiện trong mỗi process p của S (không chứa các sự kiện của các process ngoài S). Rõ ràng đây là một thực thi t -crash fair, và các process trong S là đúng nên các process này sẽ đạt tới trạng thái quyết định. \square

Bổ đề 2.2.1.5 [6] *Không tồn tại cấu hình fork.*

Chứng minh. Gọi γ là cấu hình đạt được và T là tập con có nhiều nhất t process.

Đặt $S = \mathbf{P} \setminus T$; S có ít nhất $N-t$ process, do đó tồn tại cấu hình quyết định δ sao cho $\gamma \rightarrow_S \delta$ (mệnh đề 2.2.1.3). Không mất tính tổng quát ta giả sử δ là 0-quyết định.

Gọi γ' là cấu hình sao cho $\gamma \rightarrow_T \gamma'$. Các bước trong T và S là loại trừ nhau nên theo mệnh đề 2.2.1.1, tồn tại cấu hình δ' đạt được từ cả hai δ và γ' . Vì δ là 0-quyết định nên δ' cũng là 0-quyết định. Vậy ta có γ' phải là 0-quyết định.

\square

2.2.2 Khả năng chịu lỗi của thuật toán phân tán đơn định:

Ta gọi A là một thuật toán 1-crash-robust. Trong mục này ta sẽ chỉ ra thuật toán phân tán đơn định không có khả năng chịu lỗi đối với bài toán quyết định.

Bổ đề 2.2.2.1 [7] *Tồn tại một cấu hình bắt đầu bivalent cho A .*

Chứng minh. Vì A là không tầm thường nên có các cấu hình 0- và 1-quyết định đạt được. Gọi δ_0 và δ_1 là các cấu hình bắt đầu sao cho cấu hình v -quyết định là đạt được từ δ_v .

Nếu $\delta_0 = \delta_1$ thì ta có điều cần chứng minh. Giả sử $\delta_0 \neq \delta_1$. Ta xét dãy các cấu hình bắt đầu, phần tử đầu là δ_0 và phần tử cuối là δ_1 , hai phần tử liên tiếp trong dãy chỉ khác

nhau ở một process (dãy như vậy được xây dựng bằng cách lần lượt đổi các bit từ δ_0 đến δ_1 , vd : $\delta_0=100110$ và $\delta_1=110101$, ta có $\delta_0=100110 \rightarrow 110110 \rightarrow 110110 \rightarrow \delta_1=110100$). Gọi γ_0 và γ_1 là hai cấu hình liên tiếp trong dãy sao cho v-quyết định đạt được từ γ_v . Gọi p là process trong γ_0 và γ_1 .

Xét một thực thi với cấu hình bắt đầu là γ_0 trong thực thi này p không chuyển đổi trạng thái (không bước nào được thực hiện trong p). Ta có thực thi này là 1-crash fair, do đó có cấu hình quyết định β đạt được từ γ_0 . nếu β là 1-quyết định thì γ_0 là bivalent. Nếu β là 0-quyết định, vì γ_1 chỉ khác γ_0 ở process p , và p không thực hiện bước nào nên β đạt được từ γ_1 , điều này cho thấy γ_1 là bivalent.

□

Ta gọi *bước* s là để chỉ sự nhận và xử lý một thông điệp hoặc một chuyển trạng thái (sự kiện nội bộ hay sự kiện gửi). Sự nhận thông điệp chỉ áp dụng được nếu thông điệp đang trên đường dẫn và một chuyển trạng thái thì luôn có thể được áp dụng.

Bổ đề 2.2.2.2 [8] *Gọi γ là một cấu hình bivalent đạt được và s là một bước có thể áp dụng được trong process p trong cấu hình γ . Khi đó tồn tại một dãy σ các sự kiện sao cho s có thể áp dụng được trong $\sigma(\gamma)$, và $s(\sigma(\gamma))$ là bivalent.*

Chứng minh. Gọi C là tập tất cả các cấu hình đạt được từ γ chưa áp dụng bước s , i.e., $C=\{\sigma(\gamma) : s \text{ không xảy ra trong } \sigma(\gamma)\}$, vậy ta có s có thể áp dụng được trong mọi cấu hình thuộc C .

Ta chứng minh tồn tại các cấu hình α_0 và α_1 trong C sao cho có một cấu hình v-quyết định có thể đạt được từ $s(\alpha_v)$. Nếu chứng minh được điều này thì có hai trường hợp xảy ra

- (1) $\alpha_0 = \alpha_1$, đây là điều mà ta cần chứng minh,
- (2) $\alpha_0 \neq \alpha_1$, xét tất cả các cấu hình trên đường dẫn từ γ đến α_0 và α_1 . Hai cấu hình trên đường dẫn là kề nhau nếu cấu hình này thu được từ cấu hình kia chỉ bởi một bước. Vì 0-quyết định có thể đạt được từ $s(\alpha_0)$ và 1-quyết định có thể đạt được từ $s(\alpha_1)$ nên tồn tại hai cấu hình kề nhau γ_0 và γ_1 sao cho $s(\gamma_0)$ là 0-valent

và $s(\gamma_1)$ là 1-valent. Ta chứng minh một trong γ_0 và γ_1 là một fork. Đây là điều trái với Bổ đề 2.2.1.5. Vậy ta phải có $\alpha_0 = \alpha_1$.

→ Chứng minh tồn tại các cấu hình α_0 và α_1 trong C sao cho có một cấu hình v -quyết định có thể đạt được từ $s(\alpha_v)$.

Vì γ là bivalent nên có v -quyết định β_v có thể đạt được từ γ , với $v=0,1$. Nếu $\beta_v \in C$ thì chọn $\alpha_v = \beta_v$ (ta lưu ý $s(\beta_v)$ vẫn là v -quyết định). Nếu $\beta_v \notin C$, khi này s đã được áp dụng vào một cấu hình $\delta_v \in C$ và β_v đạt được từ δ_v . Ta chọn α_v là δ_v .

→ Chứng minh một trong γ_0 và γ_1 là một fork.

Ta xem γ_1 là cấu hình thu được từ γ_0 bởi một bước e của process q , $\gamma_1 = e(\gamma_0)$. Ta có $s(e(\gamma_0))$ là $s(\gamma_1)$ nên là 1-valent, nhưng ta lại có $e(s(\gamma_0))$ không là 1-valent vì $s(\gamma_0)$ là 0-valent. Vậy e và s không giao hoán (mệnh đề 2.2.1.0), suy ra $p=q$. Ta có $\gamma_0 \rightarrow_p s(\gamma_0)$ và $\gamma_0 \rightarrow_p s(e(\gamma_0))$ lần lượt là 0-valent và 1-valent. Vậy γ_0 là một fork.

□

Mệnh đề 2.2.2.3 [9] *Không tồn tại thuật toán 1-crash-robust consensus đơn định, phân tán, không đồng bộ.*

Chứng minh. Giả sử thuật toán trên tồn tại, khi đó tồn tại γ_0 ban đầu là bivalent (Bổ đề 2.2.2.1). Ta chọn bước s_0 có thể áp dụng trong γ_0 của process p . Tồn tại dãy sự kiện σ sao cho $\gamma_1 = s_0(\sigma(\gamma))$ là bivalent. Ta lại thực hiện như trên cho γ_1 . Vậy ta đã xây dựng được một thực thi fair nhưng không quyết định.

□

Mệnh đề sau đây sẽ cho thấy không thể xây dựng thuật toán cho bài toán quyết định với số process bị hỏng $\geq N/2$, với N là số process.

Mệnh đề 2.2.2.4 [10] *Không tồn tại thuật toán t -crash-robust consensus với $t \geq N/2$.*

Chứng minh. Nếu tồn tại một thuật toán P như thế, gọi thuật toán ấy là P , thì sẽ dẫn đến các khẳng định sau:

Khẳng định 1 *P có một cấu hình bắt đầu là bivalent.*

Chứng minh. Bổ đề 2.2.2.1.

□

Cho tập con S các process, cấu hình γ được nói là S -bivalent nếu cả hai 0- và 1-quyết định đều có thể đạt được từ γ với các bước chỉ thuộc các process trong S . Cấu hình γ được nói là S -0-valent, nếu chỉ có 0-quyết định có thể đạt được từ γ với các bước chỉ thuộc các process trong S , và S -1-valent được định nghĩa tương tự.

Chia N process thành hai tập S và T có số phần tử là $\lfloor N/2 \rfloor$ và $\lceil N/2 \rceil$.

Khẳng định 2 Với mọi cấu hình γ đạt được, γ đồng thời là S -0-valent và T -0-valent hoặc đồng thời là S -1-valent và T -1-valent.

Chứng minh. Vì $t \geq N/2$ nên ta có S và T đều có thể đạt đến sự quyết định một cách độc lập. Nếu các quyết định này khác nhau thì trái với định nghĩa về t -crash-robust consensus (tính chất đồng ý).

□

Khẳng định 3 P không có cấu hình đạt được là bivalent.

Chứng minh. Giả sử P có cấu hình đạt được là γ , không mất tính tổng quát ta có thể xem γ là S -1-valent và T -1-valent (Khẳng định 2). T lại có γ là bivalent nên sẽ có một cấu hình 0-quyết định δ_0 đạt được từ γ . Trong dãy các cấu hình từ γ đến δ_0 có hai cấu hình liên tiếp nhau γ_1 và γ_0 , với γ_v đồng thời là S - v -valent và T - v -valent. Gọi p là process gây ra sự chuyển dịch từ cấu hình γ_1 sang γ_0 . Process p không thể thuộc S vì γ_1 là S -1-valent và γ_0 là S -0valent. Tương tự p không thể thuộc T .

□

Ta thấy rằng khẳng định 1 và 3 là mâu thuẫn nhau. Mệnh đề 2.2.2.4 đã được chứng minh. □

2.2.3 Khả năng chịu lỗi của thuật toán consensus xác suất:

Mệnh đề 2.2.2.3 cho ta thấy rằng mọi thuật toán consensus đơn định, phân tán đều có một tính toán vô hạn mà không quyết định. Trong mục này sẽ trình bày một thuật toán xác suất của Bracha và Toueg để giải quyết bài toán trên. Thuật toán của Bracha

và Toueg vận hành dựa trên sự lặp đi lặp lại (ta sẽ gọi là các round) hai thao tác đó là gửi thông điệp cho tất cả các process (kể cả chính nó) và chờ nhận các thông điệp từ các process khác. Thuật toán được chứng minh là quyết định, với số process hằng < N/2.

Định nghĩa 2.2.3.1 Một thuật toán được nói là *Probabilistic, t-crash-robust consensus* nếu nó thỏa các điều kiện sau:

1) **Hội tụ (Convergence).** Với mọi cấu hình ban đầu , mọi thực thi cho trước,

$$\lim_{k \rightarrow \infty} Pr[\text{một process chưa quyết định sau } k \text{ bước}] = 0.$$

2) **Đồng ý (Agreement).** Nếu , trong một cấu hình đạt được, $y_p \neq b$ và $y_q \neq b$ trong các process đúng p và q thì $y_p = y_q$.

3) **Không tầm thường (Non-triviality).** Với $v=0$ và $v=1$ tồn tại những cấu hình đạt được mà trong đó có ít nhất một process p sao cho $y_p = v$.

Ở lần lặp thứ k ta sẽ gọi là *round* k . Ta định nghĩa $R(p, q, k)$ là sự kiện , ở round k , process q nhận thông tin của p trong $N-t$ thông tin đến đầu tiên ở round k . Giả sử ta có :

(1) $\exists \varepsilon > 0 \forall p, q, k : Pr[R(p, q, k)] \geq \varepsilon$.

(2) Với mọi k , và với mọi p, q, r các process khác nhau, sự kiện $R(q, p, k)$ và $R(q, r, k)$ là độc lập.

Hai giả thiết trên được gọi là giả thiết Fair Scheduling.

Thuật toán 2.2.3.2 [11] (thuật toán của process p)

```

var value          : {0, 1}    init  $x_p$     // giá trị  $p$  sẽ bầu (  $p$ 's vote)
      round         : integer    init 0      // giá trị round
      weight        : integer    init 1      // trọng số  $p$ 's vote
      msg[0..1]     : integer    init (0,0) // đếm các vote đã nhận
      witness[0..1] : integer    init (0,0) // đếm các witness đã nhận

begin
  while  $y \neq b$  do
    begin

```

```

witness[0] = witness[1] = 0;
msg[0] = msg[1] = 0;
shout ( round, value, weight);
while (msg[0]+msg[1] < N-t) do
begin
    receive (r, v, w); // nhận thông tin từ một process q
    if r > round then
        send (r, v, w) to p; // chờ round kế
    else if r = round then
        begin
            msg[v] = msg[v] + 1;
            if w > N/2 then witness[v] = witness[v] + 1;
        end
    else skip;
end;
// Chọn giá trị mới : vote và weight cho vòng lặp kế
if witness[0] > 0 then value = 0
else if witness[1] > 0 then value = 1
else if msg[0] > msg[1] then value = 0
else value = 1;

weight = msg[v];

// Quyết định nếu lớn hơn t witness
if witness[value] > t then y = value; // kết thúc
round = round + 1;
end;
shout( round, value, N-t);
shout( round+1, value, N-t);

```

end // Kết thúc thuật toán

Mô tả thuật toán :

Ở mỗi round process p sẽ gửi giá trị round (biến round), một phiếu bầu cho 0 hoặc 1 (ta gọi là vote, biến value), cùng với trọng số của lá phiếu ấy (biến weight). Trọng số của lá phiếu là số lượng lá phiếu đó nhận được ở round trước. Process p chờ nhận đủ $N-t$ thông điệp (r, v, w) từ các process q , với r là round, v là value và w là weight của q . Nếu w trọng số $> N/2$ thì tăng số lượng v -witness một đơn vị (biến $witness[v]$), được gọi là một v -witness. Nếu p có $witness[v] > 0$ ở round k , thì p sẽ bầu cho v ở round $k+1$. Nếu $witness[v]=0$, $v=0,1$, thì nếu số thông điệp 0 nhận được $>$ số lượng thông điệp 1 nhận được thì p sẽ bầu cho 0 ở round $k+1$ ngược lại bầu cho 1. Process p sẽ quyết định với giá trị $v=0,1$ nếu p có số lượng v -witness $> t$. Các mệnh đề sau sẽ chứng minh rằng tất cả các process quyết định cùng một giá trị nếu $t < N/2$.

Mệnh đề 2.2.3.3 [12] *Với mọi round, không có hai process có v -witness, w -witness mà $v \neq w$ khác nhau.*

Chứng minh. Giả sử ở round k process p có v -witness và q có w -witness; $k > 1$ vì ở round 1 không có process nào có witness. Ta có ở round $k-1$, p nhận nhiều hơn $N/2$ vote cho v và q nhận nhiều hơn $N/2$ vote cho w . Số process là N nên có một process r gửi v -vote cho p và gửi w -vote cho q . Điều này dẫn đến $v = w$.

□

Mệnh đề 2.2.3.4 [13] *Nếu một process quyết định với giá trị v thì tất cả các process đúng sẽ cùng quyết định với giá trị v ở nhiều nhất là hai round kế tiếp.*

Chứng minh. Gọi k là round đầu tiên p quyết định với giá trị v . Sự quyết định này dẫn đến là có những v -witness trong round k . Theo mệnh đề 2.2.3.3 không có quyết định với giá trị khác v trong round k .

Trong round k có nhiều hơn t v -witness (do quyết định của p), do đó tất cả các process đúng nhận ít nhất là một v -witness ở round k . Vậy tất cả các process sẽ cùng

bầu cho v ở round $k+1$ (ở round $k+1$ p vẫn gửi v -vote). Điều này dẫn đến là nếu có một quyết định xảy ra trong một process trong round $k+1$ thì quyết định đó là cho giá trị v .

Trong round $k+1$ chỉ có v -vote là được gửi ở tất cả các process đúng. Vậy trong round $k+1$ tất cả các process đúng đều có v -witness. Từ đó ta thấy rằng ở round $k+2$ tất cả các process đúng chưa quyết định sẽ nhận $N-t$ v -witness ($N-t > t$), nên sẽ quyết định với giá trị v . \square

Mệnh đề 2.2.3.5 [14] $\lim_{k \rightarrow \infty} Pr[\text{Không có quyết định trong round } \leq k] = 0$.

Chứng minh. Gọi S là tập có $N-t$ process và giả sử rằng không có quyết định nào cho đến round k_0 . Giả thiết Fair Scheduling dẫn đến tồn tại $\rho > 0$ là xác suất nhỏ nhất mà trong bất kỳ round nào mọi process trong S đều nhận đúng các vote của $N-t$ process trong S . Ta có xác suất để điều này xảy ra trong các round $k_0, k_0 + 1$ và $k_0 + 2$ là $\psi = \rho^3$.

Nếu điều trên xảy ra thì dễ dàng ta có tất cả các process trong S sẽ quyết định ở round k_0+2 . Nếu mệnh đề 2.2.3.5 không đúng thì có một tính toán mà trong đó không có quyết định, i.e, sẽ không xảy ra điều sau: với mọi k_0 , trong các round $k_0, k_0 + 1$ và $k_0 + 2$ mọi process trong S đều nhận đúng các vote của $N-t$ process trong S . Điều này chỉ xảy ra khi $\rho=0$. \square

Mệnh đề 2.2.3.6 [15] *Nếu tất cả các process cùng giá trị ban đầu là v thì tất cả process sẽ quyết định cho v ở round 2.*

Mệnh đề 2.2.3.7 [16] *Thuật toán 2.2.3.2 là thuật toán t -crash-robust consensus xác suất với $t < N/2$.*

Chứng minh. Hội tụ do mệnh đề 2.2.3.5, không tầm thường do mệnh đề 2.2.3.6, đồng ý do mệnh đề 2.2.3.4.

2.2.4 Thuật toán Master:

Trong phần cài đặt sẽ sử dụng mô hình Master và Slave. Các Slave là các process như đã trình bày ở các mục trên. Master là một process có nhiệm vụ khởi tạo các process, gửi giá trị $x = 0/1$ cho các process, kiểm tra xem process nào hỏng, gửi số lượng process hỏng cho các process, và nhận các quyết định từ các process vẫn còn hoạt động (process đúng).

Trong phần cài đặt Master còn giữ vai trò làm hỏng một số process được chỉ định. Thủ tục *kill* trong thuật toán Master dùng để ra lệnh cho một số process ngừng hoạt động. Thủ tục này đặt ở đâu cũng được miễn sao các process đã gửi vote ít nhất một lần cho ít nhất một process đúng (do giả thiết ban đầu).

Dữ liệu được dùng trong Master :

- livetids : mảng các *tid* của các process vẫn làm việc,
- t : số process hỏng,
- count : đếm số quyết định được gửi về từ các process.

Thuật toán :

begin

t = 0;

count = 0;

Khởi tạo N process;

Gửi $x[q] = 0/1$ cho process có tid là livetids[q], $q = 0, 1, \dots, N-1$;

while (1)

begin

If (process có tid là livetids[q] bị hỏng)

begin

t++;

Cập nhật lại livetids;

end

If (có thông điệp *quyết định* đến)

```

begin
    Nhận thông điệp;
    count++;
end
If (count==N-t)
begin
    Kết thúc while(1);
end
end // while (1)
end.

```

2.2.4 Một ứng dụng của bài toán consensus xác suất :

Với các mệnh đề trên trong mục này sẽ trình bày những lập luận để cho thấy giá trị quyết định là 0 hay là 1 tùy vào số lượng 0 hoặc 1 chiếm ưu thế và sự quyết định có tính xác suất. Xác suất quyết định là 0 sẽ lớn hơn xác suất quyết định là 1 nếu số lượng 0 lớn hơn số lượng 1, i.e, số process nhận giá trị 0 ban đầu lớn hơn số process nhận giá trị 1 ban đầu.

Gọi m_0 là tập hợp N-t phần tử với số lượng giá trị 0 lớn hơn số lượng giá trị 1, m_1 là tập hợp N-t phần tử với số lượng giá trị 1 lớn hơn số lượng giá trị 0. Gọi S_0 là tập các phần tử m_0 , S_1 là tập các phần tử m_1 . Nếu số lượng giá trị 0 lớn hơn số lượng giá trị 1 thì ta có $|S_0| > |S_1|$.

Giả thiết rằng xác suất nhận được tín hiệu của process p được gửi bởi các process q là như nhau với mọi q. Với giả thiết này, nếu số lượng giá trị 0 lớn hơn số lượng giá trị 1 thì ta có $|S_0| > |S_1|$, ở *round thứ hai* xác suất để số lượng *process nhận m_0* lớn hơn xác suất để số lượng *process nhận m_1* . Khi đó ở *round thứ ba* xác suất để số lượng số 0 được chọn để bầu (vote) là lớn hơn xác suất để số lượng số 1 được chọn để bầu. Từ nhận xét trên suy ra xác suất 0 là giá trị quyết định sẽ lớn hơn 1.

Phần thực nghiệm được trình bày ở chương 5.

CHƯƠNG 3 : GIẢI GẦN ĐÚNG HỆ PHƯƠNG TRÌNH TUYẾN TÍNH VÀ BÀI TOÁN TÔ MÀU ĐỒ THỊ PHẪNG

3.1 Giới thiệu :

Tổng quan về các thuật toán ổn định là vào một giai đoạn nào đó các process đúng có thể bị xáo trộn hành vi (trong chương này được xem có thể là do một vài process hỏng) nhưng sau một khoảng thời gian hữu hạn các process đúng sẽ trở lại hành vi đúng như khi chưa xảy ra có process hỏng.

Trong chương trước ta định nghĩa một hệ thống phân tán là một bộ $S=(C, \rightarrow, I)$, trong chương này S là bộ $S=(C, \rightarrow)$.

Định nghĩa 3.1.1 Cho hệ thống phân tán $S=(C, \rightarrow)$ được nói là ổn định đối với điều kiện P nếu tồn tại một tập con $L \subseteq C$, là tập các cấu hình hợp lệ có các tính chất sau:

- 1) **Đúng (Correctness).** Mọi thực thi bắt đầu bởi một cấu hình trong L thì thỏa P .
- 2) **Hội tụ (Convergence).** Mọi thực thi đều chứa một cấu hình thuộc L .

Định nghĩa 3.1.2 Một chương trình được định nghĩa như là một tập các biến và các bước nguyên tố trên các biến đó.

Sự kiện gán một giá trị vào một biến, sự kiện đọc giá trị của biến là các bước nguyên tố trên biến.

Cho một thực thi $E=(\gamma_0, \gamma_1, ..)$ ta ký hiệu $E_i=(\gamma_i, \gamma_{i+1}, ...)$.

Định nghĩa 3.1.3 Gọi S_1 và S_2 là hai chương trình sao cho không có biến nào được gán giá trị bởi S_2 mà xuất hiện trong S_1 . Ta định nghĩa $S_1 \Theta S_2$ là chương trình chứa tất cả các biến và tất cả các thao tác (của các bước) của S_1 và S_2 .

Định nghĩa 3.1.4 Một thực thi E của $S_1 \Theta S_2$ là fair đối với S_i nếu nó chứa vô hạn các bước của S_i , hoặc có một E_j của E mà trong đó không có bước nào của S_i là áp dụng được.

Mệnh đề 3.1.5 [17] *Nếu các điều kiện sau thỏa:*

- 1) *chương trình S_1 ổn định đối với θ ,*
- 2) *chương trình S_2 ổn định đối với ψ nếu θ thỏa,*
- 3) *chương trình S_1 không thay đổi giá trị của các biến đọc bởi S_2 khi θ thỏa,*
- 4) *tất cả các thực thi đều fair đối với cả hai S_1 và S_2 ,*

thì $S_1 \Theta S_2$ ổn định đối với ψ .

Chứng minh. Cho một cấu hình của $S_1 \Theta S_2$, gọi $\gamma^{(i)}$ là tập giá trị của các biến của S_i . Cho một thực thi $E = (\gamma_0, \gamma_1, \dots)$ của $S_1 \Theta S_2$, đặt $E^{(i)} = (\gamma_0^{(i)}, \gamma_1^{(i)}, \dots)$, các $\gamma_j^{(i)}$ giống nhau thì chỉ giữ lại một.

Xét $E^{(1)}$. Vì S_2 không viết vào các biến của S_1 , nên tất cả các thay đổi trong $\gamma^{(1)}$ là bởi các bước của S_1 , điều này dẫn đến $E^{(1)}$ là một thực thi của S_1 . Theo 4) ta có E là fair đối với S_1 dẫn đến nếu $E^{(1)}$ là hữu hạn thì cấu hình cuối trong dãy E sẽ là cấu hình kết thúc cho S_1 . Vì S_1 là ổn định đối với θ nên có i sao cho $\theta(\gamma_j)$ đúng với mọi $j \geq i$.

Với i ở trên ta xét E_i . θ đúng với mọi cấu hình trong $E^{(i)}$ dẫn đến không có biến nào được đọc bởi S_2 mà bị thay đổi bởi S_1 . Do đó, chuỗi $(\gamma_i^{(2)}, \gamma_{i+1}^{(2)}, \dots)$ là một thực thi của S_2 . Nếu $(\gamma_i^{(2)}, \gamma_{i+1}^{(2)}, \dots)$ là hữu hạn thì trạng thái cuối sẽ là trạng thái kết thúc. Vì S_2 là ổn định đối với ψ nếu θ đúng, vậy ta sẽ có ψ đúng ở một còn lại nào đó của E .

□

3.2 Bài toán tô sáu màu cho đồ thị phẳng:

3.2.1 Cơ sở lý thuyết:

Cho $G=(V, E)$ là đồ thị phẳng, với mỗi đỉnh là một process, cạnh nối giữa p và q được xem là đường truyền giữa p và q . Process p có biến c_p có giá trị thuộc $\{1, 2, 3, 4, 5, 6\}$ là màu tô của đỉnh p . Các đỉnh sẽ được tô màu sao cho hai đỉnh kề nhau thì có màu khác nhau. Nếu tân từ ψ được định nghĩa

$$\psi \equiv (\forall pq \in E : c_p \neq c_q),$$

thì G tô được sáu màu như phát biểu ta phải có ψ là đúng.

Mệnh 3.2.1.1 [18] *Một đồ thị phẳng có ít nhất một đỉnh có bậc nhỏ hơn hay bằng 5.*

Một đồ thị G không hướng nếu mỗi cạnh của G ta xác định một hướng trên đó thì ta bảo rằng G được định hướng. Cho G là đồ thị có hướng. Một chu trình là một dãy các cạnh (có hướng) từ đỉnh v_0 đến đến đỉnh v_0 .

Mệnh 3.2.1.2 [19] *Cho G là đồ thị phẳng khi đó có một định hướng, không chu trình của G sao cho mọi đỉnh đều có bậc ra lớn nhất là 5.*

Chứng minh. Ta chứng minh bằng qui nạp trên các đỉnh. Xét đồ thị với số đỉnh lớn hơn hay bằng 2. Theo mệnh đề 3.2.1 có đỉnh v với bậc ≤ 5 . Đồ thị $G' = G - \{v\}$ là đồ thị có ít hơn G một đỉnh, theo giả thiết qui nạp G' có một định hướng, không chu trình có các đỉnh với bậc ra lớn nhất là 5. Ta xây dựng định hướng của G bằng cách thêm v vào định hướng của G' và các cạnh của v sẽ hướng ra.

□

Mệnh 3.2.1.3 [20] *Một đồ thị phẳng là tô được sáu màu.*

Chứng minh. Cho đồ thị phẳng G , và gọi G' là một định hướng, không chu trình, không có đỉnh nào có bậc ra lớn hơn 5 của G (mệnh đề 3.2.2). Do G' là không chu trình nên ta có thể đánh nhãn cho các đỉnh của G' bởi các nhãn $v_1, v_2, v_3, \dots, v_n$ sao cho nếu $\vec{v_j v_i}$ thì $j > i$. Các đỉnh sẽ được tô theo thứ tự từ v_1 đến v_n . Tính chất của các nhãn cho ta thấy rằng đỉnh v chỉ được tô chỉ khi các đỉnh kề trên cạnh hướng ra từ v đã được tô. Vì v chỉ có tối đa là 5 cạnh hướng ra nên màu của v sẽ khác các đỉnh kề trên các cạnh hướng ra của nó.

□

Giải thuật tô màu dựa vào ý tưởng chứng minh mệnh đề 3.2.3. Giải thuật được chia là hai pha. Pha đầu là xây dựng G' . Pha thứ hai tô màu theo thứ tự như đã chỉ. Để thể hiện một định hướng, không chu trình đồ thị mà các đỉnh là các process, trong mỗi

process p , p cũng là một số nguyên, có biến x_p có kiểu nguyên, và ta định nghĩa cạnh pq là hướng từ p đến q , nếu

$$x_p < x_q \vee (x_p = x_q \wedge p < q).$$

Rõ ràng mỗi chỉ có thể theo một hướng và đồ thị là định hướng, không chu trình. Vấn đề còn lại là thay đổi x_p để có được mỗi p có số cạnh ra nhiều nhất là 5. Gọi $out(p)$ là số cạnh ra của p ,

$$out(p) = \#\{q \in Neigh_p : \vec{pq}\}.$$

Tân từ cho pha đầu tiên là $\theta \equiv (\forall p : out(p) \leq 5)$.

Chương trình S_1 , dùng để thiết lập θ , chứa một thao tác cho mỗi process, định hướng lại cho tất cả các cạnh hướng ra từ p nếu số cạnh này > 5 :

D_p : if ($out(p) > 5$)

$$x_p = \max \{ x_q : q \in Neigh_p \} + 1$$

Mệnh đề 3.2.1.4 [21] *Chương trình S_1 ổn định đối với θ , và nếu θ được thỏa thì giá trị của x_p là hằng số trong tất cả các cấu hình sau đó.*

Chứng minh. Để chứng minh θ đúng ta sẽ chứng minh S_1 dừng.

Mệnh đề 3.2.2 suy ra tồn tại một cách đánh nhãn cho các đỉnh với các nhãn v_1, v_2, \dots sao cho mọi đỉnh p , có nhãn v_i , có nhiều nhất là 5 đỉnh kề có nhãn v_j mà $j > i$. Ta sẽ gọi một cạnh là *sai* nếu v_i hướng tới v_j mà $i > j$. Xét hàm $f(\gamma) = (n_1, n_2, \dots)$, với n_i số cạnh sai kề với v_i . Giả sử D_p áp dụng trên v_j , gọi i là chỉ số bé nhất sao cho $\vec{v_j v_i}$. Vì v_j có ít nhất sáu cạnh ra nhưng chỉ có nhiều nhất 5 đỉnh trong số đó là có nhãn có chỉ số lớn hơn j . Suy ra $j > i$. Vậy khi D_p thực hiện thì n_i giảm, n_k không bị ảnh hưởng, với $k < i$. Vậy Mỗi lần áp dụng D_p sẽ là giảm f (xét theo thứ tự chuỗi ký tự, thứ tự lexicographic). Vậy S_1 dừng. \square

Chương trình S_2 , thiết lập ψ , cũng chứa một thao tác cho mỗi process. Process p gán giá trị $b \in \{1, 2, 3, 4, 5, 6\}$ cho c_p nếu có q sao cho p hướng đến q , $c_p = c_q$ và b chưa được dùng. Ta có S_2 như sau:

$$\mathbf{C}_p: \text{If } (\exists q : \overrightarrow{pq} \wedge c_p = c_q) \wedge (\forall r \text{ sao cho } : c_r \neq b) \text{ then} \\ c_p = b$$

Mệnh đề 3.2.1.5 [22] *Chương trình S_2 dừng, và nếu θ được thỏa thì cấu hình kết thúc của S_2 thỏa ψ .*

Chứng minh. Trong định một định hướng, không chu trình của G tồn tại một đánh nhãn các đỉnh bằng v_1, v_2, \dots mỗi đỉnh chỉ có các đỉnh kề ở cạnh ra có nhãn với chỉ số nhỏ hơn. Định nghĩa $g(\gamma) = (m_1, m_2, \dots)$, với m_i là 0 nếu màu c_i khác với tất cả các màu của các đỉnh kề nói trên và là 1 nếu ngược lại. Một áp dụng C_p ở đỉnh v_i sẽ thay đổi m_i từ 1 thành 0, và có thể sẽ chuyển m_j từ 0 thành 1 với $j > i$, vậy g giảm theo thứ tự lexicographic.

Giả sử θ thỏa, xét cấu hình kết thúc. C_p không áp dụng trong cấu hình kết thúc, điều kiện If của C_p sai. Process p có tối đa là 5 đỉnh kề ở các cạnh hướng ra vậy sẽ có $b \in \{1, 2, 3, 4, 5, 6\}$ chưa được dùng bởi các đỉnh này. Vậy ta phải có $(\exists q : \overrightarrow{pq} \wedge c_p = c_q)$ sai, i.e, c_p khác tất các c_q .

Với mỗi cạnh qr , ta có q hướng tới r hoặc r hướng tới q . Suy ra $c_r \neq c_q$.

□

Thuật toán 3.2.1.6

var x_p : integer;

$c_p : \{1, 2, 3, 4, 5, 6\};$

begin

\mathbf{D}_p : if (out(p) > 5)

$$x_p = \max \{ x_q : q \in \text{Neigh}_p \} + 1$$

C_p : If $(\exists q : \overrightarrow{pq} \wedge c_p = c_q) \wedge (\forall r \text{ sao cho } : c_r \neq b)$ then

$$c_p = b$$

end.

Mệnh đề 3.2.1.7 [23] *Thuật toán 3.2.6 ổn định đối với ψ .*

Chứng minh. Thuật toán 3.2.6 là $S_1 \Theta S_2$ vì các biến được gán bởi S_2 (c_p) không xuất hiện ở S_1 . Mệnh đề 3.2.4 cho S_1 ổn định đối với θ và không thay đổi x_p sau khi θ thỏa. Mệnh đề 3.2.5 cho S_2 ổn định đối với ψ nếu θ thỏa. Cuối cùng ta có, theo mệnh đề 3.2.5 chỉ có một số hữu hạn các bước của S_2 giữa các hai bước của S_1 , suy ra $S_1 \Theta S_2$ fair đối với S_1 . Theo mệnh đề 3.2.4 S_1 dừng cho thấy $S_1 \Theta S_2$ fair với S_2 . Mệnh đề 3.1.5 cho ta điều cần chứng minh. \square

3.2.2 Cài đặt:

process dừng hoạt động. Ta cũng giả định rằng Master luôn hoạt động. Ta cũng có thể khắc phục hạn chế này bằng cách mỗi một process cũng là một Master kiểm soát các process kề với mình (đỉnh kề trong đồ thị).

Thuật toán cho Master sẽ làm việc theo cách thức sau :

- Khởi tạo các process,
- Gửi các thông tin x_p , c_p , các đỉnh liên kết (lưu trong mảng $G[p]$) cho process p ,
- Kiểm tra có process q nào hỏng không. Nếu có thực hiện :
 - Tạo process mới p_t và cho process này đảm nhận công việc của q . Gửi x_q , c_q , các đỉnh liên kết cho process t .
 - Nếu không tạo được thì tìm một process p có số nhiệm vụ (đang thực hiện giúp cho các process khác) ít nhất và gán nhiệm vụ của q cho p .

Thuật toán cho Master:

Dữ liệu:

- Mảng $G[N][N]$ có giá trị 0, 1 biểu diễn đồ thị G . Nếu $G[i][j]=1$ có cạnh giữa đỉnh i và j , và $G[i][j]=0$ nếu ngược lại.
- Mảng $F[N][N+1]$ có giá nguyên.
 - Nếu $F[i][j]=1$, $0 \leq j \leq N-1$ thì process i đang thực hiện nhiệm vụ của process j .
 - $F[i][N]$: số process process i đang đảm nhận, có giá trị đầu là 1.
- Mảng $x[N]$: giá trị x_p của các đỉnh, có giá trị nguyên dương, có giá trị đầu là bất kỳ.
- Mảng $c[N]$: giá trị c_p của các đỉnh, có giá trị $\in \{1,2,3,4,5,6\}$, có giá trị đầu là bất kỳ, trong thuật toán là giá trị 1.

begin //1

Khởi tạo N process ;

Gửi $\langle p, x[p], G[p] \rangle$ cho process p , $p=0,1,\dots, N-1$;

while (1)

begin //2

if(có process q hỏng) **then**

begin //3

for ($t=0$; $t < N$; $t++$)

if ($F[q][t]==1$) // q đang đảm nhận công việc của process k

begin //4

$F[q][t]=0$; // Không còn đảm nhận công việc t

$F[q][N]--$; // Giảm số lượng công việc

if (sinh được process mới t) **then**

begin //5

$F[t][t]=1$; // Task t nhận công việc t .

$F[t][N]=1$; // Số nhiệm vụ là 1

Gửi $\langle t, x[t], c[t], G[t] \rangle$ process t ;

end //5

else

```

begin //6
    Tìm process min_q có số nhiệm vụ ít nhất;
    F[min_q][N]++; //Số lượng nhiệm vụ tăng 1
    F[min_q][q]=1; // Nhận thêm nhiệm vụ của q
    Gởi <t, x[t], c[t], G[t]> process min_q;
end //6
end //4
end //3
If ( có cp tới ) then Nhận cp;
If ( đã nhận hết cp) then Kết thúc;
end // 2
end. //1

```

Thuật toán của process p ngoài thực hiện **D_p** và **C_p** cho chính nó thì p còn có thể nhận thêm các đỉnh q mà các process đảm nhận q không còn hoạt động.

Khi thực hiện xong **C_p** nếu c[p] không đổi thì gửi về Master c[p] và giá trị v=1. Nhờ giá trị này mà Master biết khi nào kết thúc. Ta dễ dàng thấy rằng chỉ kết thúc khi mọi đỉnh đều gửi giá trị v=1.

Thuật toán cho process p:

Dữ liệu:

- Mảng x[N] : giá trị nguyên, xác định chiều đỉnh p và đỉnh q,
- Mảng c[N] : màu của các đỉnh,
- Mảng G[N][N] : Nếu G[q] ≠ Null thì G[q] là mảng chỉ các đỉnh j liên kết với q, G[q][j]=1, và cũng cho biết p đang đảm nhận công việc của q,

begin

Nhận <p, x_p, c_p, G[p]> từ Master;

do

begin // 1

```

for(q=0;q<N;q++)
if (G[q]!=NULL) then // process p thực hiện nhiệm vụ của đỉnh q
    begin
        Gởi <q, x[q], c[q]> cho tất cả các process;
    end
for (q=0;q<N;q++) x[q]=0,với q≠p;
do
    begin //2
        Nhận <q, x[q], c[q]> q hoặc process k đang đảm nhận nhiệm vụ của q;
        If (có nhiệm vụ mới , nhiệm vụ của q được gởi từ Master) then
            begin
                Nhận <q, x[q], c[q], G[q]>;
                Gởi <q, x[q], c[q]> cho tất cả các process;
            end
        If (đã nhận đủ các x[q]) then kết thúc nhận; // x[q] ≠ 0, q=0, 1, ..., N-1
    end // 2
    for( k =0; k < N;k++)
        if (G[k]!=NULL) then // đỉnh k được đảm nhận bởi p
            begin //3
                D_p :
                    if (out_p(k)>5) then
                        x[k]=max_x(k)+1;
                C_p:
                    If ( có q kề với k sao cho: (k→q) && (c[k]==c[q])) &&
                        (∃ b ∈ { 1,2,3,4,5,6} sao cho: (∀ r :(k→r)) && c[r] ≠ b)
                        then c[k]=b;
                    v=0;
                    if (c[k] không đổi giá trị) then v=1; //
                    Gởi <c[k], v> cho Master;
            end
    end

```

end //3

if (có tín hiệu kết thúc từ Master) **then** Kết thúc;

end //1

end.

Mệnh đề 3.2.2.1 *Nếu tồn tại n sao cho cấu hình γ_j không còn process hỏng, $j \geq n$, thì thuật toán cho Master và cho p thỏa mệnh đề 3.2.1.7.*

Chứng minh. Rõ ràng nếu không process nào ngừng hoạt động thì ta có ngay mệnh đề 3.2.1.7 thỏa.

Giả sử có process q ngừng hoạt động. Vậy sẽ có một vài process chưa nhận được $x[q]$, $c[q]$ từ q . Thuật toán cho Master cho ta thấy khi phát hiện ra q ngừng hoạt động thì có hai trường hợp:

1. Master sinh được process mới đảm nhận đỉnh q . Thuật toán cho process p cho ta thấy khi hoạt động nó gửi ngay $x[p]$, $c[p]$ cho tất cả các process.
2. Master không sinh được process mới khi đó một process sẽ đảm nhận q . Thuật toán cho process cho ta thấy khi nhận thêm một nhiệm vụ thì nó gửi ngay $x[q]$, $c[q]$ cho tất cả các process.

Vậy sau khoảng một thời gian hữu hạn $x[q]$, $c[q]$ sẽ được gửi cho tất cả các process, có giá trị có thể không là giá trị process ngừng hoạt động đang lưu trữ. Các mệnh đề trên cho ta thấy chỉ cần đủ các $x[p]$ và $c[p]$ thì cuối cùng ta cũng có mệnh đề 3.2.1.7 thỏa.

3.3 Bài toán giải gần đúng hệ phương trình tuyến tính:

3.3.1 Cơ sở lý thuyết:

Cho hệ phương trình tuyến tính $AX = B$, với A là ma trận vuông cấp n có các hệ số a_{ij} , X là vector nghiệm cần tìm có nghiệm thành phần x_i , B vector n thành phần có hệ số là b_i , $i=1,..,n$, $j=1..n$. Ta chuyển hệ phương trình đã cho về dạng:

$$X = FX + G,$$

F là ma trận vuông cấp n có các hệ số là f_{ij} , G là vector có các hệ số g_i . f_{ij} được tính theo công thức:

$$f_{ij} = -\frac{a_{ij}}{a_{ii}}, a_{ii} \neq 0.$$

Mỗi g_i được tính theo công thức:

$$g_i = \frac{b_i}{a_{ii}}.$$

Định nghĩa 3.3.1.1 Một bộ $(S, +, *)$ được gọi là không gian tuyến tính, nếu :

- a) Với mọi cặp x, y của S , α là số thực, ta có $x+y \in S$ và $\alpha*x \in S$.
- b) Phép toán $+, *$ thỏa:
 - 1) $x+y = y+x$,
 - 2) $(x+y)+z = x+(y+z)$,
 - 3) Có phần tử $0 \in X$ sao cho $x+0 = x$,
 - 4) Với mọi x có phần tử đối $x' \in S$ sao cho $x+x' = 0$, ký hiệu $-x$ cho x' ,
 - 5) $1*x = x$,
 - 6) $\alpha*(\beta*x) = (\alpha\beta)*x$, $\alpha, \beta \in R$,
 - 7) $(\alpha+\beta)*x = (\alpha*x) + (\beta*x) = \alpha*x + \beta*x$,
 - 8) $\alpha*(x+y) = (\alpha*x) + (\alpha*y) = \alpha*x + \alpha*y$.

Mệnh đề 3.3.1.2 Với S là tập các ma trận vuông cấp n , với các phép toán $+, *$ được định nghĩa như sau:

$$A = (a_{ij})_{n \times n}, B = (b_{ij})_{n \times n} \in S, A+B = (a_{ij}+b_{ij}) \text{ và } \alpha \in R, \alpha*A = (\alpha a_{ij})$$

thì $(S, +, *)$ là một không gian tuyến tính.

Chứng minh. Dễ dàng kiểm tra các tính chất của định nghĩa 3.3.1.1.

□

Định nghĩa 3.3.1.3 Cho $(S, +, *)$ là không gian tuyến tính. Với $x \in S$ ta định nghĩa chuẩn của x , ký hiệu $\|x\|$, là một giá trị thực thỏa các tính chất sau:

- 1) $\|x\| > 0$ nếu $x \neq 0$ và $\|x\| = 0$ nếu $x = 0$,
- 2) $\|\alpha*x\| = |\alpha|.\|x\|$, α là số thực,

$$3) \|x+y\| \leq \|x\| + \|y\|.$$

Mệnh đề 3.3.1.4 Cho $(S, +, *)$ như trong mệnh đề 3.3.1.2. Cho $A \in S$, gọi $\|A\|_1, \|A\|_2$ lần lượt là chuẩn 1, chuẩn 2 của A được định nghĩa như sau:

$$1) \|A\|_1 = \max_i \left\{ \sum_j |a_{ij}| \right\},$$

$$2) \|A\|_2 = \max_j \left\{ \sum_i |a_{ij}| \right\},$$

thì $\|A\|_1$ và $\|A\|_2$ là các chuẩn.

Chứng minh. Chứng minh cho $\|A\|_1$.

Tính chất 1. Nếu $A \neq 0 \Rightarrow \exists i, j$ sao cho $a_{ij} \neq 0$. Vậy $\|A\|_1 \neq 0$.

Tính chất 2. Ta có $\alpha * A = (\alpha a_{ij}) \Rightarrow \|\alpha * A\|_1 = \max_i \left\{ \sum_j |\alpha a_{ij}| \right\} = \alpha \max_i \left\{ \sum_j |a_{ij}| \right\}.$

Tính chất 3. $\|A+B\|_1 = \max_i \left\{ \sum_j |a_{ij} + b_{ij}| \right\} \leq \max_i \left\{ \sum_j (|a_{ij}| + |b_{ij}|) \right\} \leq \max_i \left\{ \sum_j |a_{ij}| \right\} + \max_i \left\{ \sum_j |b_{ij}| \right\}$

Mệnh đề 3.3.1.5 $(R^n, +, \cdot)$, với $X, Y \in R^n, \alpha \in R$. Ta định nghĩa:

$$1) X+Y = (x_1+y_1, \dots, x_n+y_n),$$

$$2) \alpha X = (\alpha x_1, \dots, \alpha x_n),$$

$$3) \|X\| = \max \{ |x_i| \},$$

thì $(R^n, +, \cdot)$ là không gian tuyến tính và $\|X\|$ là chuẩn.

Mệnh đề 3.3.1.6 Cho $(S, +, *)$ như trong mệnh đề 3.3.1.2 và $(R^n, +, \cdot)$ như trên ta có $\|AX\| \leq \|A\|_1 \|X\|.$

Định nghĩa 3.3.1.7 $(S, +, *)$ là không gian tuyến tính với chuẩn $\|\cdot\|$ và $(S', +, *)$ là không gian tuyến tính với chuẩn $\|\cdot\|'$. Ánh xạ $f: S \rightarrow S'$ là liên tục nếu

$$\forall \varepsilon, \exists \eta > 0 \text{ sao cho: } \|x-y\| < \eta \Rightarrow \|f(x)-f(y)\|' < \varepsilon.$$

Định nghĩa 3.3.1.8 $(S, +, *)$ là không gian tuyến tính với chuẩn $\|\cdot\|$. Dãy $\{x_n\} \in S$ được nói là hội tụ về $a \in S$ nếu $\|x_n - a\| \rightarrow 0$, khi $n \rightarrow \infty$.

Mệnh đề 3.3.1.9 Cho $(R^n, +, \cdot)$ và chuẩn $\|\cdot\|$ như mệnh đề 3.3.1.5. Nếu $\|X_n - X_m\| \rightarrow 0$, khi $n, m \rightarrow \infty$ thì có a để $X_n \rightarrow a$ (dãy X_n được gọi là dãy Cauchy).

Mệnh đề 3.3.1.10 $(S, +, *)$ là không gian tuyến tính với chuẩn $\|\cdot\|$, $(S', +, *)$ là không gian tuyến tính với chuẩn $\|\cdot\|'$ và $f: S \rightarrow S'$ liên tục. Khi đó nếu dãy $\{x_n\} \in S$ hội tụ về $a \in S$ thì dãy $\{f(x_n)\}$ hội tụ về $f(a)$.

Mệnh đề 3.3.1.11 Cho $(S, +, *)$ như trong mệnh đề 3.3.1.2. $f: (S, +, *) \rightarrow (R^n, +, \cdot)$ với $f(X) = FX + G$ là liên tục.

Mệnh đề 3.3.1.12 [24] Cho phương trình $X = FX + G$. Nếu $\|F\|_1 \leq \alpha < 1$ hay $\|F\|_2 \leq \beta < 1$ thì với mọi X_0 ban đầu, $X_{i+1} = FX_i + G$, ta có X_n hội tụ về nghiệm đúng duy nhất X^* của hệ phương trình tuyến tính đã cho. Hơn nữa ta có $\|X^* - X_n\| \leq \frac{\alpha}{1-\alpha} \|X_n - X_{n-1}\|$, với chuẩn $\|X\|$ là chuẩn trong R^n ở trên (tương tự được phát biểu cho β).
chứng minh.

Chúng minh X_n là dãy Cauchy:

Với $k \geq 2$ ta có

$$\begin{aligned} \|X_k - X_{k-1}\| &= \|FX_{k-1} - FX_{k-2}\| = \|F(X_{k-1} - X_{k-2})\| \leq \|F\|_1 \|X_{k-1} - X_{k-2}\| \quad (\text{mệnh đề 3.3.1.6}) \\ &\leq (\|F\|_1)^2 \|X_{k-2} - X_{k-3}\| \leq \dots \leq (\|F\|_1)^{k-1} \|X_1 - X_0\| \\ &< \alpha^{k-1} \|X_1 - X_0\|, \end{aligned}$$

Suy ra với $m > n$ ta có

$$\begin{aligned} \|X_m - X_n\| &= \|X_m - X_{m-1} + X_{m-1} - \dots + X_{n+1} - X_n\| \\ &\leq \|X_m - X_{m-1}\| + \|X_{m-1} - X_{m-2}\| + \dots + \|X_{n+1} - X_n\| \quad (\text{tính chất 3}) \\ &\leq (\alpha^{m-1} + \alpha^{m-2} + \dots + \alpha^n) \|X_1 - X_0\| \\ &\leq \frac{\alpha}{1-\alpha} \|X_1 - X_0\| \end{aligned}$$

Vậy khi $m, n \rightarrow \infty$ ta có $\|X_m - X_n\| \rightarrow 0$.

Theo mệnh đề 3.3.1.9 tồn tại a sao cho $X_n \rightarrow a$, khi $n \rightarrow \infty$. Do $f(X) = FX + G$ là liên tục nên ta có $F(X_n) \rightarrow F(a)$, khi $X_n \rightarrow a$ và do đó $a = Fa + G$. Suy ra a là nghiệm của hệ phương trình tuyến tính đã cho. Đặt $X^* = a$.

X^* là duy nhất:

Giả sử có $X' \neq X^*$. Ta có $\|X^* - X'\| = \|FX^* - FX'\| \leq \|F\|_1 \|X^* - X'\|$. Suy ra $\|F\|_1 \geq 1$.

$$\begin{aligned} \text{Ta có } \|X_n - X^*\| &= \|FX_{n-1} - FX^*\| \leq \|F\|_1 \|X_{n-1} - X^*\| \\ &\leq \alpha \|X_{n-1} - X_n + X_n - X^*\| \\ &\leq \alpha \|X_{n-1} - X_n\| + \alpha \|X_n - X^*\|, \end{aligned}$$

$$\text{suy ra } \|X^* - X_n\| \leq \frac{\alpha}{1-\alpha} \|X_n - X_{n-1}\|.$$

□

3.3.2 Thuật toán :

Ta xây dựng thuật toán giải hệ phương trình tuyến tính với $\|F\|_1 \leq \alpha < 1$. Giá trị nhập vào là một vector X_0 bất kỳ. Thuật toán sẽ kết thúc khi $\|X_n - X^*\| \leq \varepsilon$. Tuy nhiên từ mệnh đề 3.3.1.12 ta sử dụng điều kiện dừng là $\|X_n - X_{n-1}\| \leq \varepsilon$.

Đặt $\psi \equiv (\exists p \in P, \exists n \geq 1 : \|X_n - X_{n-1}\| \leq \varepsilon, X \text{ của } p)$. Ta cần xây dựng thuật toán ổn định đối với ψ , i.e, nếu mỗi process p_i đảm nhận việc tính x_i thì với cấu hình γ_0 ban đầu (ứng với X_0) thì có k sao cho cấu hình γ_j thỏa ψ , với mọi $j \geq k$.

Ký hiệu $X_n = (x_0^n, x_1^n, \dots, x_{N-1}^n)$.

Thuật toán cho Master:

Dữ liệu:

- $F[N][N+2]$: $F[p][i]$ là f_{pi} của ma trận F trong công thức $X = FX + G$. $F[p][N]$ là g_p .
 $F[p][N+1]$ là $1/a_{pp}$.
- $G[N][N+1]$: $G[p][t]=1$, process p đảm nhận tính $x[t]$. $G[p][N]$ số lượng công việc hiện có của p , dùng để tìm process có số công việc ít nhất.

- $x[N]$: vector nghiệm.

begin

Tạo N process;

for($p=0;p<N;p++$)

Gửi x , $F[p]$ cho process p ;

while(1)

begin

if(process q ngừng hoạt động) **then**

begin

for($t=0;t<N;t++$)

if($G[q][t]==1$) **then** // q có tính $x[t]$

begin

$G[q][t]=0$; // không còn đảm nhận công việc tính $x[t]$

$G[q][N]--$; // Số lượng công việc giảm 1

if(Tạo được process mới) **then**

begin

$G[t][t]=1$;

$G[t][N]=1$;

Gửi x , $F[t]$ cho process;

end

else

begin

Tìm process có số công việc ít nhất, \min_q ;

$G[\min_q][N]++$;

$G[\min_q][t]=1$;

Gửi $x[t]$, $F[t]$ cho process \min_q ;

end

end

end

```

    Nhận x[p] từ process p;
    if (có tín hiệu kết thúc từ p) then kết thúc;
    end
end.

```

Thuật toán của p, đảm nhận tính x[p], có các ý chính sau : nhận X_0 ban đầu, tính x[p], gửi x[p] cho các process khác. Kiểm tra điều kiện dừng , $\|X_k - X_{k-1}\| \leq \varepsilon$. Trong quá trình có tiếp nhận thêm nhiệm vụ tính cho các x[q] khác nếu các process tính x[q] ngừng hoạt động và không tạo process mới cho q được.

Thuật toán cho process p:

Dữ liệu:

- $F[N][N]$, $G[N]$: ma trận F và vector G trong công thức $X = F * X + G$,
- $x[N]$, $y[N]$: vector nghiệm.

begin

Nhận y, F[p] , G[p] từ master //nhận X_0

repeat

$x = y$;

for(q=0;q<N;q++)

if (F[q]!=NULL) **then**

begin

 Gửi y[q] cho Master;

 Tính $y[q] = F[q][0]*x[0] + F[q][1]*x[1] + \dots + F[q][9]*x[9] + G[q]$;

end

 Gửi y[p] cho các process khác;

repeat

 Nhận các y[q] từ các process

if (có nhiệm vụ tính x[q]) **then**

begin

```

    Nhận  $y[q]$ ,  $F[q]$ ;
    Gởi  $y[q]$  cho các process khác;
    end
    until (đã nhận đủ);
    until  $\|x-y\| \leq \varepsilon$ ; //  $\|X_k - X_{k-1}\| \leq \varepsilon$ 
    Gởi thông điệp kết thúc cho Master;

```

end.

Mệnh đề 3.3.2.1 *Nếu tồn tại n sao cho cấu hình γ_j không còn process hỏng, $j \geq n$, thì thuật toán cho Master và cho p thỏa ψ .*

Chứng minh. Gọi p_i là process tính x_i . Mỗi process đều lưu trữ vector nghiệm $X = [x_0, x_1, \dots, x_{N-1}]$.

Trường hợp 1: Các p_i không bị hỏng trong suốt quá trình tính toán. Theo thuật toán, các p_i đều nhận cùng X_0 , được lưu vào mảng x , lúc khởi tạo, bước kế là tính y_i từ mảng x , bước kế là gởi x_i cho tất cả các process, bước kế chờ nhận các x_j từ các process khác, nếu nhận đủ quay lại tính x_i . Vậy trước khi quay lại tính x_i mảng x p_i có giá trị là X_1 . Quá trình được lặp lại ta trong p_i lần lượt lưu trữ X_0, X_1, X_2, \dots , với $X_{i+1} = FX_i + G$. Từ mệnh đề 3.3.1.12 ta có điều cần chứng minh.

Trường hợp 2: Xét một thực thi như sau:

Bước 1: p_i nhận X_0 lưu vào x , $i=0,1,\dots, N-1$,

Bước 2: p_i tính y_i từ x và gởi y_i cho các process p_j . giá trị y_i là x_i^1 , thành phần thứ i của X_1 của p_i . Giả sử trong bước này p_0 ngừng hoạt động và không kịp gởi y_0 cho p_1 .

Bước 3: p_i nhận các y_j từ các process khác. Trong bước này các p_2, \dots, p_{N-1} đều nhận được y_0 . Ta có các p_2, \dots, p_{N-1} đều có y mang giá trị X_1 . p_1 có y là X_1 thiếu thành phần x_0^1 .

Bước 4: Process p_2, \dots, p_{N-1} kiểm tra điều kiện dừng và quay lại bước 2. Process p_1 vẫn nằm ở bước 3.

Tiếp tục thực hiện đến bước 3 ta có p_2, \dots, p_{N-1} đều có y mang giá trị X_2 với X_2 thiếu thành phần x_0^2, x_1^2 . p_2, \dots, p_{N-1} chờ ở bước này. Ta ctheo thuật toán p_1 có y mang giá trị X_2 cũng thiếu hai thành như đã nói còn các thành phần khác cũng như p_2, \dots, p_{N-1} .

Khi có process mới (hay chung với process nào đó) thực hiện tính y_0 thì y_0 sẽ gửi cho tất cả các process. Các process lại tiếp tục hoạt động và có x là như nhau và vẫn bảo đảm X_{i+1} trong mỗi process là $FX_i + G$. Cũng theo mệnh đề 3.3.1.12 ta có điều cần chứng minh.

Ta có thể tổng quát hóa chứng minh trên bằng cách phân hoạch thành các tập các process ngừng hoạt động và p các process không nhận được y_i từ các process ngừng hoạt động.

□

CHƯƠNG 4: BÀI TOÁN PHÂN CÔNG CÔNG VIỆC

4.1 Bài toán :

Ở các chương trước , process master giữ nhiệm vụ quan sát xem process nào ngưng làm việc rồi sau đó tạo process mới thay thế hoặc gán các công việc của process ngưng hoạt động cho process có số công việc ít nhất. Ta thấy rằng sự ngưng làm việc không loại trừ bất kỳ một process nào. Do đó nếu không may master ngưng hoạt động thì toàn bộ hệ thống sẽ deadlock. Hơn nữa có những bài toán trong đó không có khái niệm master. Ví dụ như bài toán tìm đường đi ngắn nhất giữa hai máy tính nối mạng khi gửi thông điệp, bài toán ngữ pháp đồ thị, Trong chương này xin đề cử một thuật toán phân công công việc của các process ngưng hoạt động được thực hiện bởi các process còn đang hoạt động.

Định nghĩa 4.1.1 Cho (x, y) và $(u, v) \in N \times N$. Ta định nghĩa $(x, y) \prec (u, v) \Leftrightarrow (y < v) \vee ((x < u) \wedge (y = v))$.

Mệnh đề 4.1.2 Phép toán \prec có tính bắc cầu.

Chứng minh. Giả sử ta có $(x, y) \prec (u, v)$ và $(u, v) \prec (t, z)$. theo định nghĩa 4.1.1 ta có các trường hợp :

- Trường hợp 1: $y < v$ và $v < z \Rightarrow y < z \Rightarrow (x, y) \prec (t, z)$.
- Trường hợp 2: $y < v$ và $u < t$ và $v = z \Rightarrow y < z \Rightarrow (x, y) \prec (t, z)$.
- Trường hợp 3: $x < u$, $y = v$, và $v < z \Rightarrow y < z \Rightarrow (x, y) \prec (t, z)$.
- Trường hợp 4: $x < u$, $y = v$, và $u < t$ và $v = z \Rightarrow x < t$ và $y = z \Rightarrow (x, y) \prec (t, z)$.

□

Mệnh đề 4.1.2 Gọi S là tập hữu hạn các phần tử có dạng $\{(0, L_0), (1, L_1), \dots, (n, L_n)\}$, với $L_i \in N$. Khi đó tồn tại duy nhất i sao cho $\forall j \neq i, (i, L_i) \prec (j, L_j)$. Ta có thể nói (i, L_i) là bé nhất.

Chứng minh. Ta có với mọi cặp (i, L_i) và (j, L_j) luôn có $(i, L_i) \prec (j, L_j)$ hoặc $(j, L_j) \prec (i, L_i)$. Mặt khác S là hữu hạn nên tồn tại i sao cho $\forall j \neq i, (i, L_i) \prec (j, L_j)$. do tính bắc

cầu nên không thể có đồng thời $(i, L_i) \prec (j, L_j)$ và $(j, L_j) \prec (i, L_i)$. Vậy i là duy nhất.

□

4.2 Thuật toán phân công công việc:

Các process trao đổi thông tin cho nhau, ngoài các thông tin cần tính toán cho bài toán thì còn kèm theo (p_index, p_L) , với p_index chỉ số công việc p đảm nhận, p_L là số lượng công việc p đảm nhận. Ta xây dựng thuật toán cho p như sau:

- Nếu có một process q ngưng hoạt động, tìm p có (p_index, p_L) nhỏ nhất (theo định nghĩa Định nghĩa 4.1.1) và gán công việc mà q đang đảm nhận cho p .
- Giá trị ban đầu của (v_index, v_L) là (p_index, p_L) . Process p nhận tất cả thông điệp (q_index, L_q) từ q còn đang hoạt động. Nếu $(v_index, v_L) < (q_index, L_q)$ thì gán (q_index, q_L) cho (v_index, v_L) . Khi kết thúc việc nhận, nếu (p_index, p_L) là nhỏ nhất thì ta có $(p_index, p_L) = (v_index, v_L)$.

Thuật toán cho process p :

Dữ liệu:

- p_tid : tid của process,
- p_index : chỉ số công việc process p đảm nhận,
- p_L : số lượng công việc p đang đảm nhận,
- q_index : chỉ số công việc process q đảm nhận,
- q_L : số lượng công việc q đang đảm nhận,
- $danhan_k[N]$: nếu $danhan_k[i]=1$ thì p đã nhận các thông tin về công việc thứ i ,
- $p_karr[N]$: nếu $danhan_k[i]=1$ thì p đang đảm nhận công việc i .

begin

If là process được khôi phục then

begin

Gửi $\langle p_tid, p_index \rangle$ cho các process;

Chờ nhận $\langle q_tid, q_index \rangle$ từ các q ;

end

```

p_L=1;
repeat
  If có một process q vừ khởi tạo then
    begin
      Nhận  $q\_tid, q\_index$  của q;
      Kết nối với p;
      Nếu process p đang đảm nhận  $q\_index$  thì  $p\_L = p\_L - 1$ ;
    end
     $(v\_index, v\_L) = (p\_index, p\_L)$  ;
    For all process q
      begin
        If p đang đảm nhận  $q\_index$  (kể cả process  $p\_index$ ) hoặc process q ngưng
        hoạt động then
           $danhan\_k[q\_index] = 1$ ;
          If p đang đảm nhận  $q\_index$  then
             $p\_karr[q\_index] = 1$ ;
          end
        Gởi  $\langle p\_index, p\_L, p\_karr \rangle$  cho các process q;

        while(nhận chưa đủ  $\langle p\_index, p\_L, p\_karr \rangle$  từ các q còn hoạt động)
          /*  $\exists k : danhan\_k[k] = 0$  */
          begin
            If có process q ngưng hoạt động then
              begin
                Nhận  $q\_index$  của q;
                Ghi nhận q không còn hoạt động ;
                 $danhan\_k[q\_index] = 1$ ; // xem như đã nhận các thông tin từ q
              end
              If có thông điệp từ process q then
                begin
                  Nhận  $q\_tid, q\_index, q\_L, q\_karr, q\_giá\ trị$  của q;
                  If p không đảm nhận  $q\_index$  then
                    begin
                      For  $k = 0$  to  $N - 1$ 
                        If  $q\_karr[k] = 1$  then
                           $danhan\_k[k] = 1$ ; // đã nhận thông điệp k mà q đảm nhận,
                          // k giá trị
                        end
                      If  $(q\_index, q\_L) < (v\_index, v\_L)$  then
                         $(q\_index, q\_L) = (v\_index, v\_L)$ ;
                      end
                    end
                  end
                end
              end
            end // while
          end

```

```

If  $(q\_index, q\_L) = (v\_index, v\_L)$  then
/* p là process có ít công việc nhất và có chỉ số bé nhất */
begin
    Tìm chỉ số k nhỏ nhất chưa có process đảm nhận , gán k cho p;
     $p\_L++$ ; // số công việc tăng lên 1
end
until thỏa điều kiện kết thúc;

```

Gọi ψ là tần từ được phát biểu như sau :

“ Tồn tại $s \in N$ sao cho sau bước *s* hệ thống sẽ dừng phân công công việc, mỗi process có các công việc là khác nhau và số công việc của hai process chênh lệch nhiều nhất là một đơn vị”

Mệnh đề 4.2.1 Thuật toán cho process *p* là ổn định đối với ψ .

Chứng minh. Gọi *S* là $\{(0, L_0), (1, L_1), \dots, (N-1, L_{N-1})\}$. *S* là hữu hạn. do đó ở mỗi đầu vòng lặp nhận luôn tồn tại phần tử bé nhất và là duy nhất. Theo thuật toán ta thấy vòng lặp while chỉ kết thúc khi *p* đã nhận đủ các (q_index, q_L) , với các *q* còn hoạt động. Do đó khi kết thúc while ta có nếu (p_index, p_L) là nhỏ nhất thì $(v_index, v_L) = (p_index, p_L)$, *p* là duy nhất. Khi đó *p* sẽ đảm nhận *k* (chỉ số bé nhất) nếu process đảm nhận *k* đã ngưng hoạt động. Số lượng công việc , p_L , của *p* tăng 1 đơn vị. Vậy nếu *p* vẫn ít công việc nhất thì *p* lại tiếp tục nhận thêm công việc (nếu có). Rõ ràng số công việc *p* và *q* không thể chênh lệch nhau hơn một đơn vị. Các công việc giữa hai process *q* và *p* là khác nhau vì tại mỗi đầu vòng lặp repeat ta chỉ có duy nhất một process là bé nhất. Dễ dàng kiểm tra vòng lặp while luôn kết thúc □

4.3 Áp dụng bài toán phân công công việc cho bài toán tô đồ thị phẳng bằng sáu màu:

Thuật toán này cũng như thuật toán trước , vẫn áp dụng ổn định đã trình bày. Trong thuật này không có Master mà các process tự phân công công việc theo thuật toán ở trên.

Thuật toán cho process *p*:

Thuật toán cho process p:

Dữ liệu:

- p_tid : tid của process,
- p_index : chỉ số công việc process p đảm nhận,
- p_L : số lượng công việc p đang đảm nhận,
- q_index : chỉ số công việc process q đảm nhận,
- q_L : số lượng công việc q đang đảm nhận,
- danhan_k[N] : nếu danhan_k[i]=1 thì p đã nhận các thông tin về công việc thứ i,
- p_karr[N]: nếu danhan_k[i]=1 thì p đang đảm nhận công việc i,
- x[N] : giá trị đỉnh, xác định hướng cạnh (kết hợp với chỉ số đỉnh),
- c[N] : màu tô đỉnh.

begin

p_L=1;

repeat**If** có một process q vừa khởi tạo **then****begin**

Nhận q_tid, q_index của q;

Kết nối với p;

Nếu process p đang đảm nhận q_index thì p_L = p_L-1;

end

(v_index, v_L) = (p_index, p_L) ;

For all process q

begin**If** p đang đảm nhận q_index (kể cả process p_index) hoặc process q ngưng hoạt động **then**

danhan_k[q_index]=1;

If p đang đảm nhận q_index **then**

p_karr[q_index]=1;

end

Gửi <p_index, p_L, p_karr> cho các process q;

while(nhận chưa đủ <p_index, p_L, p_karr> từ các q còn hoạt động)/* $\exists k : \text{danhan_k}[k]=0$ */**begin****If** có process q ngưng hoạt động **then****begin**

Nhận q_index của q;

Ghi nhận q không còn hoạt động ;

```

        danhan_k[q_index]=1; // xem như đã nhận các thông tin từ q
    end
    If có thông điệp từ process q then
    begin
        Nhận  $q\_tid, q\_index, q\_L, q\_karr, q\_giá\ trị$  của q;
        If p không đảm nhận q_index then
        begin
            For k = 0 to N-1
                If  $q\_karr[k]=1$  then
                    danhan_k[k]=1; // đã nhận thông điệp k mà q đảm nhận,
                    // k_giá trị

                If  $(q\_index, q\_L) < (v\_index, v\_L)$  then
                     $(q\_index, q\_L) = (v\_index, v\_L);$ 
            end
        end

    end // while

    If  $(q\_index, q\_L) = (v\_index, v\_L)$  then
        /* p là process có ít công việc nhất và có chỉ số bé nhất */
    begin
        Tìm chỉ số k nhỏ nhất chưa có process đảm nhận, gán k cho p;
        p_L++; // số công việc tăng lên 1
    end

    If mọi đỉnh k được đảm nhận bởi một process then Tính x ,c ;

until thỏa điều kiện kết thúc;
end. /*kết thúc process*/

```

CHƯƠNG 5 : KẾT QUẢ THỰC NGHIỆM

5.1 Kết quả thực nghiệm của thuật toán Bracha – Toueg :

Mỗi process p sẽ nhận đầu vào là $x[p]$, $0 \leq p \leq N-1$. N trong thực nghiệm là 100 và số lượng số 1 lớn hơn số lượng số 0 là 2 và được cho như sau:

$x[N]=\{1,0,1,0,1,0,1,0,1,0,$
 $1,0,1,0,1,0,1,0,1,0,$
 $1,0,1,0,1,0,1,0,1,0,$
 $1,0,1,0,1,0,1,0,1,0,$
 $1,0,1,0,1,0,1,0,1,0,$
 $1,0,1,0,1,0,1,0,1,0,$
 $1,0,1,0,1,0,1,0,1,0,$
 $1,0,1,0,1,0,1,0,1,0,$
 $1,0,1,0,1,1,1,1,1,0,$
 $1,0,1,0,1,0,1,0,1,0,$
 $1,0,1,0,1,0,1,0,1,0$
 $\};$

Kết quả khi thực hiện chương trình:

49, 27, 20, 20, 42, 49, 22, 29, 0, 19,
 30, 20, 2, 47, 7, 26, 0, 44, 12, 12,
 10, 25, 12, 9, 19, 22, 21, 14, 7, 47,
 18, 8, 26, 40, 28, 20, 41, 2, 1, 41,
 21, 32, 13, 25, 31, 20, 2, 33, 15, 16,
 47, 27, 41, 10, 37, 10, 34, 10, 24, 41,
 9, 44, 0, 35, 34, 30, 5, 25, 33, 7,
 18, 4, 41, 32, 30, 22, 4, 34, 7, 21,
 0, 4, 49, 41, 14, 38, 3, 48, 48, 29,
 42, 7, 23, 44, 44, 9, 24, 49, 34, 7,
 tid[262155] ---> decide=1.
 tid[262159] ---> decide=1.
 tid[262161] ---> decide=1.
 tid[262163] ---> decide=1.
 tid[262170] ---> decide=1.
 tid[262175] ---> decide=1.
 tid[262178] ---> decide=1.
 tid[262184] ---> decide=1.
 tid[262185] ---> decide=1.
 tid[262193] ---> decide=1.
 tid[262207] ---> decide=1.
 tid[262209] ---> decide=1.
 tid[262213] ---> decide=1.
 tid[262216] ---> decide=1.

Process [9] có t=19, giá trị này được lấy ngẫu nhiên.

tid[262218] ---> decide=1.
tid[262223] ---> decide=1.
tid[262225] ---> decide=1.
tid[262227] ---> decide=1.
tid[262228] ---> decide=1.
tid[262233] ---> decide=1.
tid[262238] ---> decide=1.
tid[262246] ---> decide=1.
tid[262148] ---> decide=1.
tid[262149] ---> decide=1.
tid[262150] ---> decide=1.
tid[262153] ---> decide=1.
tid[262154] ---> decide=1.
tid[262156] ---> decide=1.
tid[262157] ---> decide=1.
tid[262158] ---> decide=1.
tid[262162] ---> decide=1.
tid[262165] ---> decide=1.
tid[262166] ---> decide=1.
tid[262167] ---> decide=1.
tid[262168] ---> decide=1.
tid[262169] ---> decide=1.
tid[262171] ---> decide=1.
tid[262172] ---> decide=1.
tid[262173] ---> decide=1.
tid[262174] ---> decide=1.
tid[262177] ---> decide=1.
tid[262179] ---> decide=1.
tid[262181] ---> decide=1.
tid[262182] ---> decide=1.
tid[262187] ---> decide=1.
tid[262188] ---> decide=1.
tid[262189] ---> decide=1.
tid[262190] ---> decide=1.
tid[262191] ---> decide=1.
tid[262192] ---> decide=1.
tid[262194] ---> decide=1.
tid[262195] ---> decide=1.
tid[262196] ---> decide=1.
tid[262198] ---> decide=1.
tid[262200] ---> decide=1.
tid[262202] ---> decide=1.
tid[262203] ---> decide=1.
tid[262204] ---> decide=1.

tid[262205] ---> *decide*=1.
tid[262210] ---> *decide*=1.
tid[262211] ---> *decide*=1.
tid[262212] ---> *decide*=1.
tid[262214] ---> *decide*=1.
tid[262217] ---> *decide*=1.
tid[262220] ---> *decide*=1.
tid[262221] ---> *decide*=1.
tid[262222] ---> *decide*=1.
tid[262224] ---> *decide*=1.
tid[262226] ---> *decide*=1.
tid[262231] ---> *decide*=1.
tid[262236] ---> *decide*=1.
tid[262239] ---> *decide*=1.
tid[262242] ---> *decide*=1.
tid[262243] ---> *decide*=1.
tid[262147] ---> *decide*=1.
tid[262151] ---> *decide*=1.
tid[262152] ---> *decide*=1.
tid[262160] ---> *decide*=1.
tid[262164] ---> *decide*=1.
tid[262176] ---> *decide*=1.
tid[262180] ---> *decide*=1.
tid[262183] ---> *decide*=1.
tid[262186] ---> *decide*=1.
tid[262197] ---> *decide*=1.
tid[262199] ---> *decide*=1.
tid[262201] ---> *decide*=1.
tid[262206] ---> *decide*=1.
tid[262208] ---> *decide*=1.
tid[262215] ---> *decide*=1.
tid[262219] ---> *decide*=1.
tid[262229] ---> *decide*=1.
tid[262230] ---> *decide*=1.
tid[262232] ---> *decide*=1.
tid[262234] ---> *decide*=1.
tid[262235] ---> *decide*=1.
tid[262237] ---> *decide*=1.
tid[262240] ---> *decide*=1.
tid[262241] ---> *decide*=1.
tid[262244] ---> *decide*=1.
tid[262245] ---> *decide*=1.

5.2 Kết quả thực nghiệm của thuật toán tô 6 màu cho đồ thị phẳng:

Đồ thị được cho như sau:

```
int G[N*N]={
    0,1,1,1,1,0,1,0,1,0,
    1,0,1,1,1,0,0,0,0,0,
    1,1,0,1,0,0,0,0,0,0,
    1,1,1,0,1,1,0,1,1,1,
    1,1,0,1,0,1,1,0,0,0,
    0,0,0,1,1,0,1,1,0,0,
    1,0,0,0,1,1,0,1,1,0,
    0,0,0,1,0,1,1,0,1,1,
    1,0,0,1,0,0,1,1,0,1,
    0,0,0,1,0,0,0,1,1,0
};
```

Trong thực nghiệm này process tính đỉnh 0 , được gọi là Task[0], và task[1] sẽ ngưng hoạt động. Task[2] nhận tính x_0 và x_1 . Kế đến Task[2] ngưng hoạt động. Kế đến Task[8] ngưng hoạt động.

Kết quả của thực nghiệm:

N=10

Task[0] ngưng hoạt động thêm vào task[2].

Task[1] ngưng hoạt động thêm vào task[2].

tids:

-20, -20, 262248, 262249, 262250, 262251, 262252, 262253, 262254, 262255,

Task[262248][2] ngưng hoạt động.

Master tạo thêm new Task[262256][0].

Master tạo thêm new Task[262257][1].

Master gọi new_task[2] cho task[262256][0] .

round = 1.

Task[262254][8] ngưng hoạt động.

Master tạo thêm new Task[262258][8].

round = 2.

round = 3.

round = 4.

round = 5.

round = 6.

```

đinh[0]---->mau[1].
đinh[1]---->mau[4].
đinh[2]---->mau[2].
đinh[3]---->mau[3].
đinh[4]---->mau[5].
đinh[5]---->mau[2].
đinh[6]---->mau[3].
đinh[7]---->mau[4].
đinh[8]---->mau[2].
đinh[9]---->mau[1].

```

5.3 Kết quả thực nghiệm của thuật toán giải gần đúng hệ phương trình tuyến tính :

Hệ phương trình có nghiệm đúng là $X=1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ và được cho

$$\begin{aligned}
 10x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} &= 64 \\
 x_1 + 10x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} &= 73 \\
 x_1 + x_2 + 10x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} &= 82 \\
 x_1 + x_2 + x_3 + 10x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} &= 91 \\
 x_1 + x_2 + x_3 + x_4 + 10x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} &= 100 \\
 x_1 + x_2 + x_3 + x_4 + x_5 + 10x_6 + x_7 + x_8 + x_9 + x_{10} &= 109 \\
 x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + 10x_7 + x_8 + x_9 + x_{10} &= 118 \\
 x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + 10x_8 + x_9 + x_{10} &= 127 \\
 x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + 10x_9 + x_{10} &= 136 \\
 x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + 10x_{10} &= 145
 \end{aligned}$$

Trong thực nghiệm này process tính x_0 , được gọi là Task[0], và task[1] sẽ ngưng hoạt động. Task[2] nhận tính x_0 và x_1 . Kế đến Task[2] ngưng hoạt động. Kế đến Task[3] ngưng hoạt động. kế đến Task[7] ngưng hoạt động.

Kết quả của thực nghiệm:

$N=10$

Master->Task[0] ngưng hoạt động.

Master->Goi task[0] cho Process[262260][2] .

Master->Task[1] g hoạt động.

Master->Goi task[1] cho Process[262260][2] .

0---Master->tids:

0, 0, 262260, 262261, 262262, 262263, 262264, 262265, 262266, 262267,

Process->Process[262260][2] bắt đầu.

Process->Process[262261][3] bat dau.
 Process->Process[262262][4] bat dau.
 Process->Process[262263][5] bat dau.
 Process->Process[262264][6] bat dau.
 Process->Process[262265][7] bat dau.
 Process->Process[262267][9] bat dau.
 1---Master->Task[262260][2] ngung hoat dong.
 2---Master->tao New Task[262268] cho x[0].
 3---Master->tao New Task[262269] cho x[1].
 4---Master->Goi task[2] cho Process[262268][0] .
 Process->Process[262266][8] bat dau.
 5---Master->Task[262261][3] ngung hoat dong.
 6---Master->tao New Task[262270] cho x[3].
 Process->Process thay the[262268][0] bat dau.
 Process->Process thay the[262269][1] bat dau.
 Process->Process thay the[262270][3] bat dau.
 Process->Chuan=14.500000.
 Process->Task[262268][0] nhan Task[2].
 Process->Chuan=8.170000.
 Process->Chuan=4.156596.
 Process->Chuan=1.680311.
 Process->Chuan=0.454753.
 Process->Chuan=0.064768.
 Process->Chuan=0.021505.
 Process->Chuan=0.007622.
 Process->Chuan=0.002470.
 Process->Chuan=0.000501.
 7---Master->Task[262265][7] ngung hoat dong.
 3---Nghiem:
 x[0]=0.999985.
 x[1]=1.999987.
 x[2]=2.999985.
 x[3]=3.999985.
 x[4]=5.000028.
 x[5]=6.000028.
 x[6]=7.000028.
 x[7]=8.000028.
 x[8]=9.000028.
 x[9]=10.000028.

Kết luận :

Luận văn đã thực hiện được các vấn đề sau:

- cài đặt các thuật toán bằng các chương trình cụ thể bằng pvm cho C,
- chỉ ra một ứng dụng của bài toán quyết định,
- ứng dụng lý thuyết ổn định để giải hai bài toán giải gần đúng hệ phương trình tuyến tính và tô đồ thị phẳng bằng 6 màu, và trong khi chương trình hoạt động có thể có một vài process ngưng hoạt động, chương trình vẫn đảm bảo cho kết quả đúng,
- đưa ra và cài đặt được bài toán phân công công việc khi có process hỏng.

Một số vấn đề luận văn chưa đạt được là các chương trình chỉ được thực hiện trên một số ít máy , cụ thể là 3 máy. Do đó chưa thể hiện được hết các kịch bản mà các process có thể hỏng. Về mặt viết chương trình thì chương trình chưa được gọn, chưa tối ưu.