تمرین سری اول سیستم‌های توزیع شده

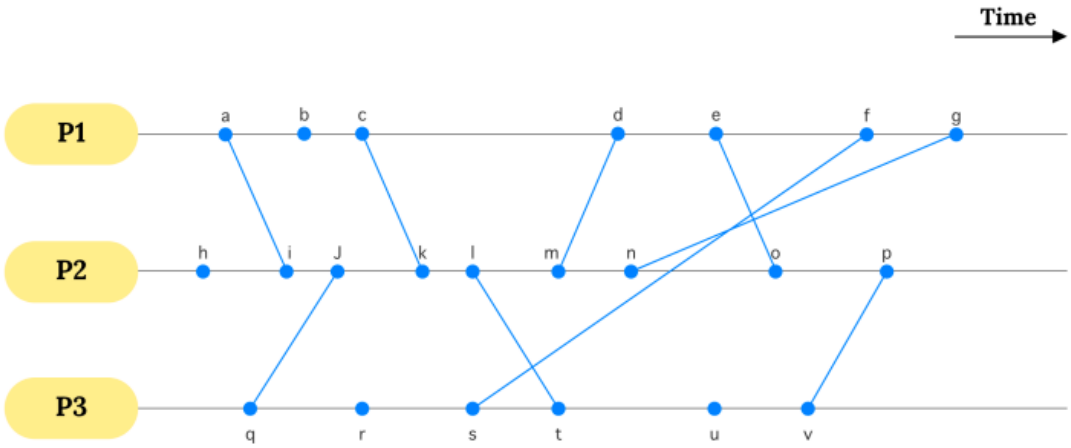
استاد: دکتر کمندی

پارسا محمدپور

پاییز ۱۴۰۳

پرسش ۱

شکل زیر را در نظر بگیرید:



الف) با فرض استفاده از ساعت لمپورت، زمان هر ایونت لیبل‌دار را مشخص کنید.

با توجه به نحوه عملکرد ساعت لمپورت، هر پروسس با انجام هر کار (شامل ارسال و دریافت پیام و انجام یک کار درون خودش) ساعتش را بروزرسانی می‌کند. فرمول این بروزرسانی به صورت زیر است:

- **R1** Before executing an event (send, receive, or internal), process p_i executes the following:

$$C_i := C_i + d \qquad (d > 0).$$

In general, every time **R1** is executed, d can have a different value

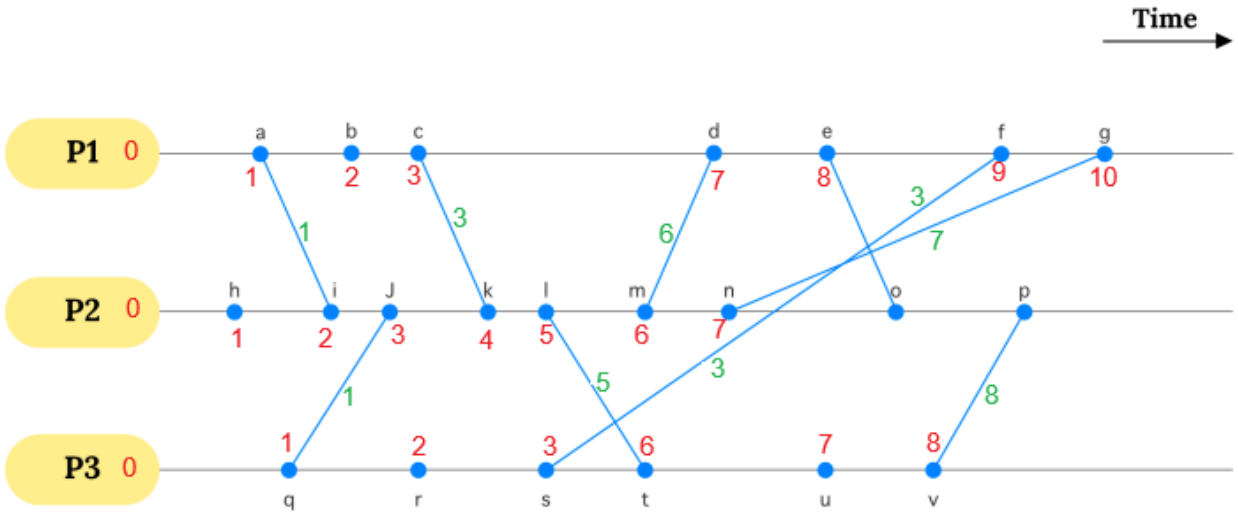
که طبق این فرمول، هر پراسس بعد از انجام یک عمل، ساعت خودش را به مقدار d، که بزرگتر از صفر است، اضافه می‌کند. همچنین هر پیامی که بین دو پراسس ارسال می‌شود، شامل مقدار زمان منطقی[1] پراسس ارسال کننده پیام نیز می‌باشد. سپس پراسس دریافت کننده بین این مقدار ارسال شده و مقدار ساعت منطقی خودش در آن لحظه، ماکزیمم را محاسبه می‌کند سپس مقدار را بعلاوه یک می‌کند و این مقدار جدید را به عنوان ساعت منطقی جدیدش درنظر می‌گیرد. شکل زیر این عملیات را نشان می‌دهد:

- **R2** Each message piggybacks the clock value of its sender at sending time. When a process p_i receives a message with timestamp C_{msg}, it executes the following actions:
  1. $C_i := max(C_i, C_{msg})$;
  2. execute **R1**;
  3. deliver the message.

[1] Logical time

در اکثر مواقع، مقدار d را برابر با یک درنظر می‌گیرند و ما هم برای حل این مثال، مقدار d را برابر با یک در نظر می‌گیریم. حال با این حساب، به سراغ حل مسئله می‌رویم.

شکل زیر، حل شده سوال برای حالتی که از ساعت لمپورت استفاده کنیم، می‌باشد. اعداد روی هر ایونت، مقدار زمان منطقی هر پراسس در آن زمان را نشان می‌دهد و اعداد روی پیام‌ها، مقدار ساعت منطقی حمل شده از پراسس ارسال‌کننده به پراسس دیگر را نشان می‌دهد. همچنین ساعت منطقی هر پراسس با رنگ قرمز و مقدار حمل شده در هر پیام از هر پراسس به پراسس دیگر با رنگ سبز تیره، مشخص شده است.



در این شکل در ابتدا هر پراسس با مقدار زمان منطقی برابر با صفر شروع به کار می‌کند و پس از انجام هر کاری، مقدار ساعت آن به شکل گفته شده بروزرسانی می‌شود. در پیام ارسال شده از s به f، مقدار حمل شده را دوبار بر روی پیام گذاشته‌ایم. این صرفا به دلیل طولانی بودن پیام است و برای اینکه مقدار آن گم نشود و یا با مقدار دیگری اشتباه گرفته نشود، این کار را کردیم؛ وگرنه دلیل دیگری ندارد.

ب) با فرض استفاده از ساعت برداری، زمان هر ایونت را مشخص کنید. (مقدار پیش‌فرض هر فرآیند: بردار صفر)

در سیستم ساعت منطقی برداری[1]، هر پراسس یک بردار nتایی (n تعداد پراسس‌های موجود در سیستم است) نگه می‌دارد که در با انجام هر عمل درون خود، مقدار درایه متناظر با عدد خودش در بین پراسس‌ها (مثلا اگر شماره پراسس برابر با یک باشد، درایه شماره یکم از این بردار) را بروزرسانی می‌کند. یعنی مقدار آن درایه را بعلاوه یک مقدار مثبت d می‌کند. فرمول این کار به صورت زیر است:

- **R1** Before executing an event, process p_i updates its local logical time as follows:

$$vt_i[i] := vt_i[i] + d \qquad (d > 0).$$

همچنین مطابق با حالت قبل، هر پراسس با هر ارسال پیام، مقدار بردار خودش را هم در پیام ارسال می‌کند. پراسس دریافت‌کننده، با دریافت پیام، برای هر درایه، مقدار درایه متناظر در بردار خودش را بروزرسانی می‌کند (ماکزیمم می‌گیرد) و سپس درایه متناظر با شماره خودش را d تا اضافه می‌کند. فرمول این کار به صورت زیر است:

- **R2** Each message m is piggybacked with the vector clock vt of the sender process at sending time. On the receipt of such a message (m,vt), process p_i executes the following sequence of actions:
  1. update its global logical time as follows:

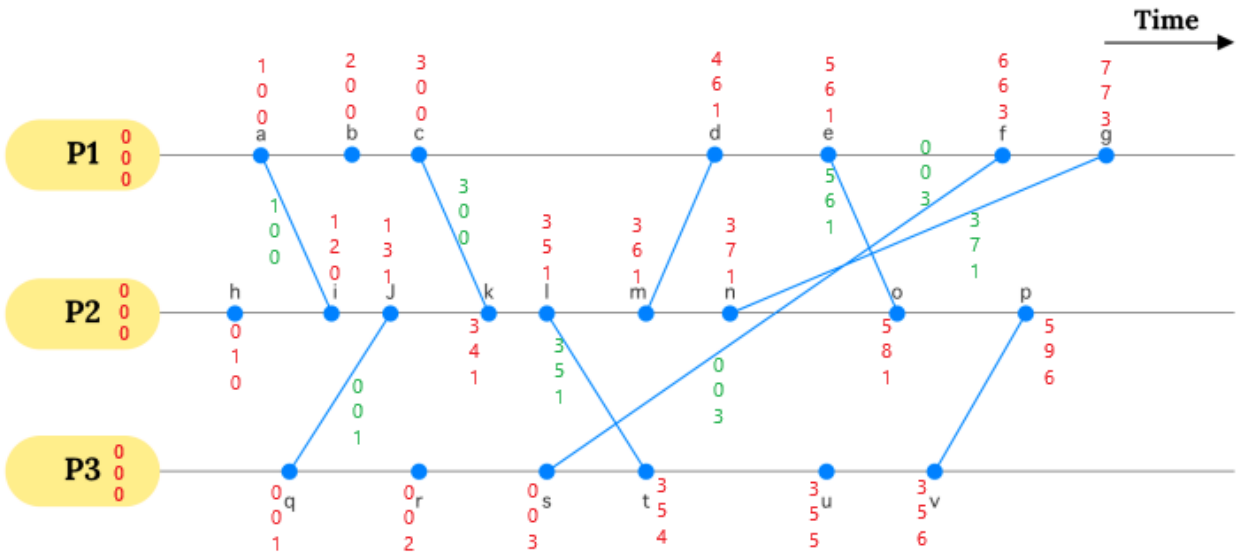
$$1 \le k \le n \ : \ vt_i[k] := max(vt_i[k], vt[k]);$$

  2. execute **R1**;
  3. deliver the message m.

در اکثر مواقع، مقدار d را برابر با یک درنظر می‌گیرند و ما هم برای حل این مثال، مقدار d را برابر با یک در نظر می‌گیریم. حال با این حساب، به سراغ حل مسئله می‌رویم.

به هر پراسس یک بردار اختصاص می‌دهیم و مقدار اولیه آن را برابر با صفر می‌گذاریم. پس همه پراسس‌ها با مقدار اولیه صفر کار را شروع می‌کنند. سپس با توجه به نحوه عملکرد الگوریتم، جلو می‌رویم. شکل زیر، حل شده سوال برای حالتی که از ساعت برداری استفاده کنیم، می‌باشد. بردار روی هر ایونت، مقدار زمان منطقی برداری هر پراسس در آن زمان را نشان می‌دهد و بردار روی پیام‌ها، مقدار ساعت منطقی برداری حمل شده از پراسس ارسال‌کننده به پراسس دیگر را نشان می‌دهد. همچنین ساعت منطقی برداری هر پراسس با رنگ قرمز و مقدار بردار حمل شده در هر پیام از هر پراسس به پراسس دیگر با رنگ سبز تیره، مشخص شده است.

[1] Logical vector time



در این شکل در ابتدا هر پراسس با مقدار زمان منطقی برابر با صفر شروع به کار می‌کند و پس از انجام هر کاری، مقدار ساعت آن به شکل گفته شده بروزرسانی می‌شود. در پیام ارسال شده از s به f، مقدار حمل شده را دوبار بر روی پیام گذاشته‌ایم. این صرفا به دلیل طولانی بودن پیام است و برای اینکه مقدار آن گم نشود و یا با مقدار دیگری اشتباه گرفته نشود، این کار را کردیم؛ وگرنه دلیل دیگری ندارد.

پرسش ۲

الف) یک نسخه اصلاح شده از ساعت لمپورت را در نظر بگیرید:

به‌جای تنظیم مقدار اولیه روی صفر، هر فرآیند مقدار اولیه دلخواه را ثبت می‌کند. سپس فرآیندها به شیوه الگوریتم اصلی عمل می‌کنند.

آیا زمان‌های اختصاص داده شده به ایونت‌ها ویژگی ساعت لمپورت را حفظ می‌کند؟ توضیح دهید.

بله. در این حالت ویژگی‌های ساعت برداری لمپورت حفظ می‌شود.

ساعت لمپورت ویژگی زیر که به ویژگی سازگاری[1] معروف است، را دارد. یعنی اگر بین دو پراسس، ایونتی ردوبدل شده باشد، ساعت گیرنده پیام، حتما بزرگتر از ساعت فرستنده پیام می‌باشد. فرمول زیر نشان‌دهنده این موضوع می‌باشد:

Consistency property

Clearly, scalar clocks satisfy the monotonicity and hence the consistency property:

$$\text{for two events } e_i \text{ and } e_j,\ e_i \rightarrow e_j \Longrightarrow C(e_i) < C(e_j).$$

این ویژگی کماکان حفظ می‌شود. برای مثال اگر پراسس i با مقدار اولیه a کارش را شروع کرده باشد و پراسس j با مقدار اولیه b کارش را شروع کرده باشد، بعد از اینکه i یک پیام به j بدهد، مقدار ساعت منطقی آن‌ها بروزرسانی می‌شود. پس مقدار ساعت منطقی i برابر با a + d می‌شود و مقدار ساعت منطقی b برابر با max(a+d, b)+d می‌شود. حال با توجه به اینکه ما ماکزیمم گرفتیم، مقدار ساعت منطقی پراسس j حتما بزرگتر مقدار ساعت منطقی پراسس i می‌شود. (چون اگر حاصل ماکزیمم برابر با a+d شود، پس مقدار ساعت منطقی j برابر با a + 2d می‌شود و از جاییکه d>0 می‌باشد، پس شرط برقرار می‌شود. اگر هم مقدار b انتخاب شود، با توجه به اینکه از ماکزیمم این مقدار انتخاب شده، پس b بزرگتر از a + d می‌باشد، پس b+d (چون با یک مقدار مثبت جمع شده) هم بزرگتر از a+d می‌شود.)

قابل ذکر است که ساعت لمپورت فقط ویژگی سازگاری را داشت و ویژگی قویا سازگاری[2] را نداشت.

[1] consistency
[2] Strongly consistency

ب) نسخه اصلاح شده دیگر از ساعت برداری را درنظر بگیرید:

به‌جای تنظیم مقدار اولیه روی بردار صفر، هر فرآیند مقادیر اولیه دلخواه را ثبت می‌کند، سپس فرآیندها به شیوه الگوریتم اصلی عمل می‌کنند.

آیا بردارهای اختصاص داده شده به ایونت‌ها، ویژگی ساعت‌های برداری را حفظ می‌کنند؟ توضیح دهید.

هم بله و هم خیر.

ساعت برداری هم ویژگی سازگاری (که فرمول و توضیح آن در قسمت قبل داده شد) را دارد و هم ویژگی قویا سازگاری را دارد. ویژگی قویا سازگار به این صورت است که به نوعی جهت عکس ویژگی سازگار را هم برقرارمی‌کند. یعنی اگر ما ببینیم که یک بردار از دیگری بزرگتر مساوی است (با توجه به تعریف بزرگتر مساوی بودن در اینجا که به صورت مقابل است: در مقایسه درایه‌های نظیر به نظیرشان یکی از دیگری همواره بزرگتر یا مساوی باشد) بین بردار دو تا ایونت وجود دارد، پس می‌توانیم بگوییم که یکی از آن‌ها قبل از دیگری اتفاق افتاده است و به نوعی بینشان ترتیب کلی[1] برقرار است.

The total order relation $\prec$ on two events x and y with timestamps (h,i) and (k,j), respectively, is defined as follows:

$$x \prec y \Leftrightarrow (h < k \text{ or } (h = k \text{ and } i < j)).$$

در این شکل زمان ایونت x را با (h, i) نشان دادیم که یعنی در ساعت پراسس شماره i برابر با h است. این فرمول می‌گوید ما می‌توانیم بگوییم یک ترتیب کلی بین دو تا ایونت داریم اگر و تنها اگر زمان یک پراسس از دیگری کوچکتر باشد یا اگر زمان‌شان یکسان است، اندیس مطلق به پراسس ایونت اول کوچک‌تر باشد. به عبارتی هم سازگار باشد و هم قویا سازگار باشد.

حال ابتدا به سراغ بررسی ویژگی سازگاری می‌رویم. برای این حالت فرض می‌کنیم که پراسس a با بردار اولیه h پیامی برای پراسس b با بردار اولیه g ارسال کند. طبق الگوریتم زمان منطقی برداری، ابتدا a مقدار موجود در اندیس متناظر با خودش در بردار را یکی اضافه می‌کند و به این ترتیب، بردار h’ بوجود می‌آید. سپس طبق الگوریتم، a این بردار جدید رابه همراه پیام، برای b می‌فرستد. طبق الگوریتم، b می‌آید و بردار خودش را بروزرسانی می‌کند به طوری که درایه‌های متناظر با هم را بین دو مقدار ماکزیمم میگیرد (این دومقدار، یکیش مقدار موجود فعلی در بردار خودش است و دیگری مقدار مجمود در بردار ارسال شده از سمت a که همان h’ است) و به جای بردار خودش قرار می‌دهد. سپس پس از اجرای الگوریتم و پردازش‌های مورد نیاز روی پیام، مقدار موجود با درایه متناظر با خودش در بردار را یکی زیاد می‌کند. پس با این حساب با توجه به اینکه بین درایه‌های قبلی b با درایه‌های موجود در h’، ماکزیمم گرفتیم، پس مقادیر موجود در بردار جدید v، g’ بزرگتر یا مساوی هستند با مقادیر (درایه‌های متناظر باهم) موجود در بردار h’. ب توجه به این مورد و همچنین باتوجه به اینکه در g’ حداقل یک درایه هم وجود دارد که از درایه متناظرش در بردار h’ بزرگتر باشد (مقدار موجود در درایه متناظر با شماره خود پراسس b قطعا بزرگتر است و نمی‌تواند مساوی باشد، چون حتی اگر بعد از ماکزیمم گرفتن هم مساوی میشد، باز هم در انتها بعلاوه یک شده است و بزرگتر می‌شود) پس قطعا این دو بردار نمی‌توانند موازی باشند و مقدار بردار پراسس b با توجه به تعریف بزرگتر بودن برداری(مقایسه درایه‌های متناظر)، بیشتر می‌شود از مقدار موجود در پراسس a. پس ویژگی سازگاری برقرار است.

حال به سراغ ویژگی قویا سازگار می‌رویم. در اینجا این ویژگی را ممکن است نداشته باشم. یعنی در اصل تا وقتی که پراسس‌ها با هم همگام[2] نشوند یا مقادیر اولیه‌شان حالت خاصی نباشد، برقرار نمی‌شود. برای دیدن این مورد، یک مثال می‌آوریم که در آن این ویژگی برقرار نیست.

فرض کنید مقادیر اولیه برای دو تا پراسس به صورتی مشخص شوند که یکی از دیگری چندین واحد بیشتر باشد، به طوریکه پراسس با اندیس کوچک‌تر مقادیر بزرگتری از پراسس دوم که اندیس بزرگتری دارد، داشته باشد. سپس در نظر میگیریم که هر کدام از اینها، یک عملیات داخلی انجام می‌دهند و بدین ترتیب مقدار موجود در درایه‌متناظر با شماره هر کدام، بروزرسانی می‌شود. اما با توجه به فرض، همچنان این دو بردار خاصیه اولیه‌شان را حفظ کرده‌اند؛ یعنی همچنان بردار پراسس با شماره پراسس کوچک‌تر، بزرگتر از بردار دیگر است (درایه‌های متناظر آن از درایه‌های بردار پراسس با شماره پراسس بزرگتر، بزرگتر است.) پس با این حساب

[1] Total ordering
[2] sync

می‌دانیمک که ساعت برداری پراسس با شناسه (شماره پراسس‌ها به ترتیب از یک تا n) کوچک‌تر بزرگتر است، پس اگر ویژگی قویا سازگاری را داشته باشیم، باید بگوییم ایونت پراسس دوم رابطه علیت[1] دارد با ایونت پراسس اولف در حالی که می‌دانیم اینطور نیست و در حالت عادی اتفاقا به خاطر وجود ترتیب کلی، حتی باید در نظر می‌گرفتیم که ایونت پراسس اول اتفاق بیفتد و سپس ایونت پراسس دوم.(این در اینجا مهم نیست ولی وجود آن به خاظر شرط موجود در رابطه ترتیب کلی است و اصلا برای ما در اینجا مهم نیست). پس حتی اگر ترتیب کلی را لحاظ نکنیم و صرفا به سراغ قویا سازگاری برویم، میبینیم که این دو تا ایونت که هیچ‌ربطی بهم ندارند، با یکدیگر رابطه علیت دارند که خب میدونیم چون اصلا این ایونت‌ها در ابتدا اتفاق افتاده است و هر پراسس در خودش کاری انجام داده‌است، ربطی به یکدیگر ندارند. پس رابطه قویا سازگاری برقررا نیست. ممکن است البته رابطه قویا سازگاری هم به ازای مقادیر اولیه خاصی و همچنین حالات خاصی (مثلا ابتدا همه باهم یک دور با هم بردارهایشان را به اشتراک بگذارند و یا مقادیر اولیه همه‌آن‌ها یکسان انتخاب شود) برابر باشند ولی به صورت کلی نمی‌توان این را گفت که برقرار است. پس رابطه قویا سازگار به طور کلی برقرار نیست.

[1] Causality relation

پرسش ۳

نشان دهید که در ساعت برداری، باید بعد بردارها برابر با تعداد فرآیندها باشد، در غیر این صورت ساعت از حالت strongly consistency خارج می‌شود.

برای این اثبات از برهان خلف استفاده می‌کنیم.

فرض می‌کنیم این کار با بردار با طول n-1 قابل انجام باشد (اگر با هر طول کمتر از n ای قابل انجام باشد، با برداری با طول n-1 هم قطعا قابل انجام است برای مثال می‌توان بقیه درایه‌های بردار را با صفر پر کرد و صرفا آن ها را قرار داد تا به بردار n-1 تایی برسیم.) که n در آن تعداد پراسس‌هاست. برای این کار پس باید یک پراسس j وجود داشته باشد که مقدار زمان آن را نگه نمی‌داریم.

حال با این تفاسیر، در نظر می‌گیریم که i و i’ پراسس‌های هستند که برای آن‌ها مقدار زمان را در بردار زمانی نگه می‌داریم. حال فرض میکنیم که یک از پراسس i یک ایونت برای هر کدام از پراسس‌های i' و j (به ترتیب) ارسال می‌شود که این اولین ایونتی است که این دو پراسس در کل دریافت می‌کنند.(میشد به جای این شرط هم بگوییم هر حالتی که بردار نهاییشان حاصل از ماکزیمم گرفتن با بردار زمانی فرستاده شده از طرف i برابر شود ولی خب در حالتی که مثال زدیم، خیلی واضح و سر راست است. همچنین در این حالت هیچ شرطی را هم نقض نکردیم؛ کماکان شروع پراسس‌ها با بردار صفر است و قوانین بروزرسانی و ... هم کاملا مطابق با الگوریتم اصلی است.)

سپس هر کدام از این پراسس‌ها ساعت خودشان را بروزرسانی می‌کنند به طوریکه در i'، مقدار بردار ماکزیمم گرفته می‌شود و همچنین اندیس متناظر با آن هم مقدار مثبتی اضافه می‌شود، اما در پراسس j ، تنها مقدار بردار زمان ماکزیمم گرفته می‌شود و چون هیچ درایه‌ای متناظر با آن وجود ندارد، پس هیچ مقداری را اضافه نمی‌کند. حال اگر به این دو بردار دقت کنیم، میبینیم که بردار موجود در پراسس j کوچکتر است. (در مقایسه نظیر به نظیر درایه‌ها، کوچکتر مساوی هستند تمام درایه‌های آن از درایه‌های موجود بردار پراسس دیگر) پس طبق قویا سازگاری باید رابطه علیت بین این دو تا ایونت در پراسس‌های متخلف برقرار باشد و باید اول ایونت پراسس j اتفاق بیفتد. این درحالیست که میدانیم با توجه به تعریف علیت، هیچکدام از این پراسس‌ها با یکدیگر رابطه علیت ندارند. پس به تناقض می‌رسیم. پس فرض خلف باطل است و شرط قویا سازگاری را برای حالتی که تعداد درایه‌های آرایه کمتر از تعداد پراسس‌ها باشد، از دست می‌رود.

■

پرسش ۴

If events corresponding to vector timestamp $Vt_1, Vt_2, \dots Vt_n$ are mutually concurrent, then prove that

$$Vt_1[1], Vt_2[2], \dots, Vt_n[n] = \max(Vt_1, Vt_2, \dots, Vt_n)$$

برای اثبات این مورد ابتدا به فرمول الگوریتم ساعت منطقی برداری نگاه می‌کنیم. در این تعریف، اگر یک ایونت، درونی باشد، پس صرفا درایه متناظر با آن عنصر یکی زیاد می‌شود و اگر ارسال پیام باشد، زمان برداری را در آن پیام قرار می‌دهیم. اگر هم گیرنده پیام باشد، بردار خودش را با بردار فرستاده شده از پراسس فرستنده مقایسه می‌کند (درایه‌های نظیر به نظیر را) سپس در هر درایه ماکزیمم را قرار می‌دهد. سپس پس از انجام عملیات درایه متناظر با شماره خود پراسس (پراسس گیرنده) را هم یکی اضافه می‌کند. عکس‌های زیر برای این الگوریتم می‌باشند:

- **R1** Before executing an event, process p_i updates its local logical time as follows:

$$vt_i[i] := vt_i[i] + d \qquad (d > 0).$$

- **R2** Each message m is piggybacked with the vector clock vt of the sender process at sending time. On the receipt of such a message (m,vt), process p_i executes the following sequence of actions:
  1. update its global logical time as follows:

$$1 \le k \le n \ : \ vt_i[k] := max(vt_i[k], vt[k]);$$

نکته: طبق این الگوریتم، هر پراسس فقط می‌تواند زمان خودش را در بردار عوض کند (اضافه کند) و زمان بقیه پراسس‌ها را صرفا می‌تواند نگه‌دارد و ارسال کند. چون در الگوریتم، هر پراسس زمان کل بردار را (درایه‌ها را نظیر به نظیر) ماکزیمم میگرفت و صرفا عدد موجود در بردار که مربوط به خودش بود را مقداری (یکی) اضافه می‌کرد. پس به جز خود پراسس، هیچ پراسس دیگری نمی‌توانست، زمان آن پراسس را عوض کند.

حال با توجه به این مورد، برای هر پراسس هر سه حالت ممکن برای آخرین ایونت آن پراسس را در نظر می‌گیریم. این حالت‌ها به صورت زیر هستند:

۱- ایونت درونی:

در این حالت، طبق تعریف، هر پراسس درایه موجود در شماره خودش (شماره پراسس که بین یک تا n بود) را یکی اضافه می‌کند. همچنین با توجه به اینکه تا قبل از آن، بقیه پراسس‌ها صرفا عدد قبلی را می‌توانستند در بهترین حالت داشته باشند، و هر پراسس مقدار زمان متناظر با پراسس‌های دیگر را فقط از روی ورودی ماکزیمم می‌گرفت و فقط تایم خودش را اضافه می‌کرد، پس هیچکسی به تایم این پراسس جز خودش نمی‌تواند اضافه کند طبق الگوریتم. پس در بهترین حالت بقیه ممکن است آخرین مقدار زمان (زمان محلی یا زمان همین پراسس) قبل از اضافه شدن و انجام ایونت درونی را می‌توانستند داشته باشند و اما الآن در بردار خود این پراسس، این عدد یکی اضافه شده است، پس مقدار ماکزیمم زمان برای این پراسس، **فقط دست بردار خودش** است.

۲- ایونت ارسال پیام:

در این حالت، قبل از ارسال پیام، مقدار زمان مربوط به پراسس فرستنده، در بردار فرستنده بروزرسانی می‌شود و سپس در پیام برای ارسال قرار می‌گیرد. پس باز با توجه به اینکه هر پراسس دیگر، صرفا می‌توانست تا قبل از انجام این عمل، مقدار ماکزیمم زمان ثبت شده در بردارها برای این پراسس را داشته باشد، پس بعد از این اضافه شدن، این مقدار دست این پراسس می‌باشد فقط؛ اما با توجه به اینکه هنوز گیرنده پیام را دریافت نکرده است و و اجرای الگوریتم تموم نشده برای این ایونت، پس نمیتوانیم بگوییم فقط دست خودش است. حال گیرنده پیام این بردار را دریافت می‌کند و سپس روند بروزرسانی را برای بردار زمانش انجام می‌دهد. حال با توجه به اینکه طبق الگوریتم، در پیام مقدار جدید زمان پراسس فرستنده قرار داشت، در هنگام ماکزیمم گرفتن، مقدار موجود در بردار این پراسس (پراسس گیرنده) که مربوط به پراسس فرستنده بود، ماکزیمم گرفته می‌شود که مقدار جدید جایگزین می‌شود.(طبق نکته و مشابه استدلال قسمت بالا، هر پراسس، فقط می‌تواند مقدار زمان موجود در بردارش را که مربوط به خودش است تغییر دهد و برای بقیه را فقط می‌تواند

نگه‌دارد، پس در بهترین حالت مقدار قدیمی زمان پراسس فرستنده را داشته یا مقداری کمتر از آن را داشته که در هر دو حالت در هنگام ماکزیمم گرفتن، جتیگزین می‌شود مقدار جدید پراس فرستنده با مقدار قبلیش در بردار زمانی پراسس گیرنده) پس بدین ترتیب، مقدار ماکزیمم زمان پراسس فرستنده دست خود پراسس فرستنده و دست خود پراسس فرستنده و این پراسس گیرنده می‌باشد. (حالا ممکن است پراسس گیرنده کارش همچنان تمام نشود و این مقدار را برای پراسس‌های دیگر بفرستند ،**کار پراسس فرستنده تمام شده است، نه گیرنده**، پس این مقدار ماکزیمم ممکن است دست چندین پراسس باشد. ولی میدانیم قطعا دست پراسس فرستنده و گیرنده هست، بقیه هم ممکن است داشته باشند و ممکن است نداشته باشند)

۳- ایونت دریافت پیام:

در این حالت، پس از دریافت پیام، این پراسس مقدار بردار زمانی خودش را با بردار ورودی ماکزیمم می‌گیرد. پراسس فرستنده هم نمی‌تواند بیشتر از زمان عددی پراسس گیرنده را برای خودش ارسال کند، چون هیچکسی غیر از خود پراسس نمیتواند زمان آن پراسس را بروزرسانی کند و تغییر دهد، باز هم مقدار ماکزیمم زمان موجود برای پراسس دست خودش است فقط. چون مثلا گر تا آن زمان ماکزیمم مقدار زمان نگه‌داشته شده برای پراسس a بود، بعد از اینکه در انتهای دریافت پیام، مقدار زمان خودش را با مقدار d جمع می‌کند و زمان پراسس گیرنده برابر با a+d می‌شود. پس ماکزیمم زمان موجود برای هر پراسس، **فقط دست خود آن پراسس** می‌باشد در این حالت.

حال اگر به این حالت‌ها نگاه کنیم، در همه این حالت‌ها، مجموعه پراسس‌هایی که مقدار ماکزییم زمانی (زمان محلی) پراسس را دارند، در همه حالات، شامل خود پراسس هم هست،(بقیه ممکن است این مقدار ماکزیمم را داشته باشند یا نداشته باشند ولی خود پراسس این مقدار را همیشه دارد) پس مقدار ماکزیمم زمان پراسس دست خودش است.

این مورد برای همه پراسس‌ها صدق می‌کند. پس بنابراین اگر بردارها را ماکزیمم بگیریم، مقدار هر درایه هم ماکزیممش در بردار همان پراسس قرار دارد، پس به رابطه بالا می‌رسیم.

راه دیگر هم این بود که بگوییم طبق الگوریتم، بقیه صرفا مقدار زمان پراسس‌های دیگر را دریافت می‌کنند و فقط خود پراسس زمان خودش را زیاد می‌کند. پس با این حساب مقدار ماکزیمم زمانی هر پراسس، دست خود آن پراسس نیز می‌باشد.

یک راه دیگر هم این بود که روش اول را به صورت استقرا جلو برویم به جای اینکه بازگشتی طور جلو برویم. اینطوری که در ابتدا همه پراسس‌ها ماکزیمم خودشونو دارن (چون ابتدا همه با بردار صفر شروع می‌کنند درسته)و بعد فرض کنیم که تا مرحله k ام درسته.(می‌دانیم که برای ساعت برداری، ما یک ترتیب کلی هم داریم، پس اینوت‌ها را به این ترتیب میچینیم، قبل از شروعف مرحله صفر (همون در ابتدا ما) بعدش دونه دونه ایونت‌ها رو میریم جلو، مرحله یک و دو و ... و فرض استقرا هم مرحله صفر که شروع است در نظر می‌گیریم) بعد با توجه به هر ایونت بگیم که سه نوع داره، شامل دریافت، ارسال و درونی، بگیم که برای مرحله k+1 هم درسته. که باز باید تقسیم بندی می‌کردیم و مطابق بالا استدلال می‌آوردیم.(اگر درونی باشه که خب مقدار ماکزیمم قبلیش که دست خودش و یه سریای دیگه (احتمالا) بودهف این سری عوض میشه و فقط میفته دست خودش. توی حال دریافت هم باز ماکزیمم دست خودش بوده واحتمالا یه سریای دیگه، بعدش بازم اضافه میشه و دیگه فقط دست خودشه توی این مرحله. توی حالت ارسال هم که باز مقدار قبلی که ماکزیمم بوده را جمع میکنه و مقدار جدید میشه ماکزیمم و پراسس گیرنده هم مکقدار جدیدش را می‌گیرد ولی چون آن را اضافه نمی‌کند (فقط حق دارد مال خودش را اضافه بکند طبق الگوریتم) پس باز هم مقدار ماکزیمم را خودش هم دارد(در این حالت پراسس گیرنده هم این مقدار ماکزیمم را دارد ولیبرای ما فقط این مهم است که خود فرستنده هم داشته باشد که دارد))

پرسش ۵

If events e_i and e_j respectively occurred at process p_i and p_j and are assigned vector timestamps VT_{e_i} and VT_{e_j}, respectively, then show that

$$e_i \rightarrow e_j \Leftrightarrow VT_{e_i} < VT_{e_j}$$

برای این اثبات، هر دو طرف را باید اثبات کنیم. این اثبات‌ها به این شکل هستند:

- ابتدا حالت سازگاری را بررسی می‌کنیم. یعنی حالت:

$$e_i \rightarrow e_j \Rightarrow VT_{e_i} < VT_{e_j}$$

این حالت به راحتی از روی تعریف الگوریتم قابل اثبات است. چون از e_i به e_j پیام ارسال شده است، پس می‌دانیم بردار زمان e_j طبق تعریف الگوریتم، برابر است با حاصل ماکزیمم گرفته شدم درایه‌های متناظر در بردار ارسال شده در پیام و بردار زمانی موجود در پراسس j قبل از آن. پس بنابراین، تا اینجای کار هر درایه موجود در بردار زمانی e_j بزرگتر مساوی درایه‌های موجود در بردار ارسال شده در پیام از سمت پراسس i می‌باشد(هنور ممکن است که اگرتمام ورودی‌های موجود در پیام ارسالی از سمت i بزرگتر یا مساوی باشند مقدار بردار زمانی حاصل مساوی بردار زمانی ارسالی باشد). اما طبق تعریف الگوریتم میدانیم که باید در این مرحله، پس از انجام پردازش‌ها، مقدار موجود در درایه متناظر با شماره پراسس j را یکی (یک یا یه عدد مثبت) افزایش داد. پس مقدار موجود و متناظر به پراسس j در بردار زمانی مربوط به پراسس j، مقداری بیشتر می‌شود. پس اگر در قبل از آن هم مقدار بردارهایشان مساوی بود، الآن دیگر بردار زمانی پراسس j به خاطر اضافه شدن این مقدار، بزرگتر می‌شود. اما بقیه درایه ها همچنان همان نتیجه ماکزیمم می‌مانند. پس دیگر این دو بردار با یکدیگر مساوی نیستند و مقادیر موجود در بردار پراسس j، بزرگتر مساوی با مقادیر متناظر در بردار پراسس i هستند. پس رابطه زیر برقرار است.

$$e_i \rightarrow e_j \Rightarrow VT_{e_i} < VT_{e_j}$$

■

- حالا حالت قویا سازگار را بررسی می‌کنیم. یعنی حالت زیر:

$$VT_{e_i} < VT_{e_j} \Rightarrow e_i \rightarrow e_j$$

حالا این قسمت را از طریق برهان خلف اثبات می‌کنیم. فرض می‌کنیم که این حالت برقرار نیست، یعنی هیچ رابطه علیتی بین ایونت‌های i و J نداریم. حال با توجه به نحوه مقایسه VT برای ایونت‌های i و j، میدانیم که مقادیر موجود در بردار ایونت j بزرگتر مساوی مقادیر متناظر در بردار زمانی ایونت i هستند. حال اگر بین این دو رابطه علیت برقرار نباشد، پس باید قبل از این دو تا ایونت، یک ایونت،e'، (آخرین ایونت با این شرایط اگر چند تا ایونت بود) در پراسس i وجود داشته باشد که با ایونت پراسس j رابطه علیت داشته باشد و زمان مربوط به پراسس i در بردار j از آن آمده باشد و این شرط برقرار باشد. (اگر هیچی نبود، همان بردار صفر این قابلیت را دارد) و از جایی که ما می‌دانیم، تمام ایونت‌های یک پراسس با یکدیگر ربابطه علیت دارند، پس این دو تا ایونت (e و e' با) هم روی پراسس i با یکدیگر رابطه علیت دارند. حالا دو تا راه داریم:

۱- یا این ایونت، برابر با همان ایونت اولیه باشد($e'=e$):

در این حالت شرط اثبات می‌شود. چون در نظر گرفتیم این دوتا ایونت با هم رابطه علیت دارند و در اصل باید در نظر بگیریم که نمی‌تواند اتفاق بیفتد چون اگر اتفاق بیفتد، یعنی در اصل اینا باهم رابطه علیت داشتند از اول.

۲- این ایونت، برابر با ایونت اولیه نباشد*(e -> e′)*:

در این حالت به تناقض می‌خوریم چون وقتی این دو تا با یکدیگر برابر نیستند، پس مقدار درایه متناظر با زمان ایونت *e* پراسس *i* در بردارش، نسبت به قبلی (*e′*) باید اضافه شده باشد. حال از آن طرف با توجه به اینکه بین *e′* در پراسس *i* و *e* در پراسس *j*، رابطه علیت داشتیم و آخرین ایونت از پراسس *i* بوده که این حالت را داشته، پس مقدار متناظر با پراسس *i* در بردار VT_{e_j} برابر است با $VT_{e\prime_i}$ و همچنین می‌دانیم چون $VT_{e\prime_i} < VT_{e_i}$ هست(چون باهم رابطه علیت دارند). حال با فرض مسئله به تناقض رسیدیم چون با این تفاسیر الآن باید $VT_{e_j} < VT_{e_i}$ در حالی که با توجه به فرض صورت مسئله برعکس این را داشتیم. پس به تناقض رسیدیم در این حالت.

حال با توجه به اینکه در حالت دوم به تناقض رسیدیم، پس باید همان حالت اول درست باشد(در همانجا هم گفته شد که اگر این را در مظر بگیریم پس فرض مسئله اصلی تایید شده است)، پس یا باید رابطه علیت داشته باشند، یا به تناقض می‌خویم. پس باید رابطه علیت داشته باشند.

(یک راه دیگر هم این بود که کلا برهان خلف نزنیم. صرفا بیایم . بگیم این دو حالت را داریم و توی حالت دوم به تناقض می‌رسیم پس نمیتونه اتفاق بیفته و در حالت اول درسته همه چی که خب فرض مسئله تایید میشه)

■

حال هر دو طرف شرط اثبات شد. پس می‌توانیم بگوییم که شرط اثبات شده است.

■

پرسش ۶

الگوریتم floodMax برای انتخاب لیدر را مطالعه کرده و روند الگوریتم را با استفاده از مثال به طور کامل شرح دهید. (منبع: کتاب Distributed algorithms از Lynch)

الگوریتم فلودمکس[1] یکی از انواع الگوریتم‌های انتخاب لیدر است[2] است که در آن برخلاف الگوریتم‌های قبلی که دیدم، نیازی به داشتن گرافی (شبکه‌ای) به صورت دایره‌ای نداریم. در اینجا فقط نیاز به این داریم که گراف جهت‌دار (بی‌جهت هم باشد اوکی است چون در اصل همانن جهت‌داری است که تمام یال‌هایش دوطرفه هستند) قویا متصل[3] (یعنی از هر پراسسی بتوان به پراسس دیگر راه پیدا کرد) باشد. توضیحات این الگوریتم به صورت زیر است:

پراسس‌ها در یک گراف قرار دارند و به آن‌ها شناسه‌هایی که دارای ترتیب هستند، اختصاص داده شده است. هر کدام از این پراسس‌ها فقط یک شناسه دارند و هیچ دو پراسسی شناسه تکراری ندارند و گراف قویا متصل است.

نیازمندی‌ها[4]:

در این الگوریتم هم همانند دیگر الگوریتم‌های انتخاب لیدر، باید فقط و فقط یک پراسس ادعای رهبری بکند و و این کار را باید با تغییر یک استاتوس در خودش به لیدر انجام دهد.

در این الگوریتم باید ما مقدار قطر گراف[5] را هم بدانیم.

الگوریتم:

در این الگوریتم، هر پراسس، مقدار شناسه ماکزیممی را که دیده است در گراف پخش می‌کند[6] (به این معنی که آن را به تمام یال‌های خروجی‌اش می‌فرستد) و آن را نیز در متغیری ذخیره می‌کند. در ابتدا مقدار این متغیر برابر است با مقدار شناسه خود آن گره است. بعد از تعداد قطر گراف مرحله، آن پراسسی که مقدار ماکزیمم درون آن برابر با مقدار شناسه خودش باشد، می‌شود لیدر و بقیه پراسس‌ها غیر لیدر نیستند.

[1] Flood max algorithm
[2] Leader election
[3] Strongly connected
[4] requirements
[5] Graph diameter
[6] Broadcast

تعریف دقیق این الگوریتم در کتاب "سیستم‌های توزیع شده - نانسی-لینچ"[1] به صورت زیر است:

- متغیرها و مقدارهای اولیه‌های موجود در پراسس

$states_i$ consists of components:
u, a UID, initially i's UID
max-uid, a UID, initially i's UID
$status \in \{unknown, leader, non\text{-}leader\}$, initially *unknown*
rounds, an integer, initially 0

همانطور که در تصویر مشخص است، هر پراسس شامل تعدادی متغیر است. این متغیرها عبارتند از:

- u: شناسه هر پراسس می‌باشد
- max-uid: ماکزیمم شناسه‌ای است که هر پراسس تا به حال دیده است
- status: شامل استاتس هر پراسس است که یکی از مقادیر نامشخص، لیدر و غیر لیدر است. در ابتدا برابر با نامشخص است.
- rounds: شماره مرحله‌ای است که در حال حاضر در آن هستیم که مقدار اولیه آن صفر است.

- پیام‌های ارسالی هر پراسس:

$msgs_i$:
if $rounds < diam$ then
  send *max-uid* to all $j \in out\text{-}nbrs$

همانطور که مشخص است، هر پراسس در هر مرحله، ماکزیمم شناسه‌ای را که دیده است (همان متغیر max-uid) را برای تمام پراسس‌هایی که به آن‌ها می‌تواند پیام بفرستد، می‌فرستد.

- تابع تغییرات (الگوریتم یا تابع مورد اجرا هر پراسس در هر مرحله یا همان trans):

[1] Nancy Lynch – distributed systems

$trans_i$:
$rounds := rounds + 1$
let U be the set of UIDs that arrive from processes in $in\text{-}nbrs$
$max\text{-}uid := \max(\{max\text{-}uid\} \cup U)$
if $rounds = diam$ then
    if $max\text{-}uid = u$ then $status := leader$
    else $status := non\text{-}leader$

توضیح این الگوریتم (تابع) همانطور که در تصویر مشخص است، به این صورت است که در هر مرحله، تعداد مراحل (متغیری که در آن مرحله حاضر را نگه‌می‌دارد را) یکی اضافه می‌کند. سپس تمام شناسه‌های دریافتی را در یک مجموعه می‌ریزد و سپس مقدار متغیر max-uid که در اصل همان ماکزیمم شناسه‌ای است که تا حالا دیده‌است را برابر با بزرگترین عضو مجموعه شناسه‌های دریافتی و ماکزیمم شناسه‌ای که دیده است قرار می‌دهد (بین تمام شناسه‌های ورودی جدید و ماکزیممی که تا مرحله قبل دیده بود، ماکزیمم میگیرد). سپس اگر مرحله ما مرحله انتهایی بود(تعداد مراحل برابر با قطر گراف)، مقدار متغیر status را تعیین می‌کنیم به این صورت که اگر بزرگترین شناسه‌ای که تاحالا دیده بود، برابر با شناسه خودش بود، پس مقدار status را برابر با لیدر بگذار؛ در غیر این صورت(اگر ماکزیمم شناسه مشاهده شده که در متغیر max-uid است برابر با شناسه خودش نبود)، مقدار status را برابر با غیر لیدر بگذار.

تحلیل صحت عملکرد الگوریتم فلادمکس:

میدانیم که اگر گراف و یا همان شبکه ارتباطی ما، یک شبکه جهت‌دار قویا متصل باشد، یعنی از هر گره‌ای مسیری به گره دیگر وجود دارد. از طرفی اگر هم گراف ما جهت دار نباشد هم می‌توان آن را گراف جهت داری در نظر گرفت که هر یال بی‌جهت، به دو تا یال در خلاف جهت تبدیل شده است. پس فرض میگیریم که ما یک گراف جهت‌دار قویا متصل داریم و از هر پراسسی می‌توان به پراسس‌های دیگر در تعدادی مرحله پیام فرستاد.

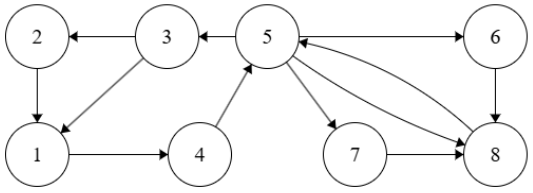
حال ابتدا به سراغ تعریف قطر گراف برویم. قطر گراف برابر است با طول ماکزیمم مسیر کوتاهی که بین دو نود از گراف وجود دارد. یعنی بیایم و تمام نودها را دو به دو انتخاب کنیم (چون جهت‌دار است، پس ترتیب هم مهم است)؛ یعنی دوتایی‌های مرتب از نودها انتخاب کنیم و سپس طول کوتاه‌ترین مسیر بین این نودها را پیدا کنیم. قطر گراف می‌شود بزرگترین عدد بدست آمده در این روند. یعنی اگر مثلا همه آن‌ها را در یک مجموعه بریزیم، همه این طول کوتاه‌ترین مسیر بین هر دو تا نود رو، قطر گراف برابر با بزرگترین عدد در این مجموعه است. پس بنابراین فاصله هر دو گره در گراف، کمتر مساوی قطر گراف می‌باشد.

حال با توجه به تعریف قطر می‌دانیم که اگر یک پراسس، بزرگترین شناسه را داشته باشد که خب همیشه خودش انتخاب می‌شود و بقیه هم بهش برسن یا نرسن مهم نیست چون در نهایت خودش انتخاب می‌شود. ولی برای بقیه نودها که غیر لیدر قرار است بشوند، می‌دانیم که در تعداد مرحله برابر با قطر گراف، آن‌ها همه‌شان مقدار ماکزیمم را گرفته اند. برای اثبات این موضوع، از تعریف قطر گراف استفاده می‌کنیم. در مرحله اول، تمام همسایه‌های بیرونی گره با بزرگترین شناسه، این شناسه را دریافت می‌کنند. در محله دو، تمام گره‌هایی که از گره با بزرگترین شناسه به فاصله حداکثر

دو هستند، این مقدار را دریافت می‌کنند چون پراسس‌هایی که در مرحله قبل آن‌ها را دریافت کرده بودند، آن را برایشان می‌فرستند و ... . همچنین با توجه به تعریف قطر گراف می‌دانیم که تعداد مراحل بین هر دو گره در گراف، کمتر مساوی قطر گراف است، پس حداکثر در تعداد قطر گراف مرحله‌ای بزرگترین شناسه، به همه پراسس‌ها می‌رسد. (در بدترین حالت این است که فاصله یک پراسس از این پراسس با بزرگترین شناسه برابر با قطر گراف باشد، وگرنه زودتر هم ممکن است بفهمد، ولی خب چون نمی‌دانیم کجاست این پراسس با بزرگترین شناسه، تا قطر گراف مرحله پیش می‌رویم تا مطمئن شویم.)

حال در ادامه یک مثال برای این الگوریتم میزنیم:

فرض می‌کنیم که گراف ما جهت‌دار و به صورت زیر است و هر گره برابر با یک پراسس است و همچنین شناسه هر پراسس، در درون دایره‌های زیر می‌باشد. شکل زیر را در نظر بگیرید:



در این شکل قطر گراف برابر با چهار است. در این مثال طول مسیر بین پراسس (گره) هشت و پراسس چهار، برابر است با رفتن از هشت به پنج، بعد به سه، بعد به یک بعد به چهار.

حال جدول زیر که حاصل از اجرای الگوریتم تا پایان مرحله مقداردهی اولیه است، به شکل زیر است:

	P1	P2	P3	P4	P5	P6	P7	P8
max-uid	1	2	3	4	5	6	7	8
status	unknows	unknows	unknows	unknows	unknows	unknows	unknows	unknows

پایان مرحله اول:

	P1	P2	P3	P4	P5	P6	P7	P8
max-uid	3	3	5	4	8	6	7	8
status	unknows	unknows	unknows	unknows	unknows	unknows	unknows	unknows

پایان مرحله دوم:

	P1	P2	P3	P4	P5	P6	P7	P8
max-uid	5	5	8	4	8	8	8	8
status	unknows	unknows	unknows	unknows	unknows	unknows	unknows	unknows

پایان مرحله سوم:

	P1	P2	P3	P4	P5	P6	P7	P8
max-uid	8	8	8	5	8	8	8	8
status	unknows	unknows	unknows	unknows	unknows	unknows	unknows	unknows

پایان مرحله چهارم:

	P1	P2	P3	P4	P5	P6	P7	P8
max-uid	8	8	8	8	8	8	8	8
status	non-leader	non-leader	non-leader	non-leader	non-leader	non-leader	non-leader	leader

جدول‌های بالا وضعیت متغیرهای هر پراسس بود البته متغیرهای تعداد مرحله و شماره پراسس برای کمتر کردن حجم کار حذف شدن و شماره که در اصل با اسم پراسس در بالا آمده است و شماره مرحله هم که در متن بالای هر جدول، گفته شده است.

پرسش ۷

مقاله پیوست شده که نسخه تغییر یافته از الگوریتم HS است را مطالعه کرده، سپس روند الگوریتم و مقایسه با الگوریتم اصلی را به صورت کامل شرح دهید.

ابتدا سراغ توضیح الگوریتم HS می‌رویم. سپس به سراغ توضیح الگوریتم ارائه شده در مقاله می‌رویم. در انتها نیز به سراغ بیان تفاوت این دو الگوریتم و مقایسه آن‌ها می‌پردازیم.

الگوریتم HS: این الگوریتم در اصل یک الگوریتم انتخاب لیدر است. در این الگوریتم، فرض کردیم که پراسس‌ها در یک گراف حلقه‌ای[1] قرار دارند و گراف بدون جهت[2] است. در این الگوریتم هر پراسس در هر مرحله، در صورتی که در مرحله قبل هر دو پیام[3] ارسالی از هر دو طرف را دریافت کرده باشد، برای پراسس‌های اطرافش پیامی حاوی مقدار شناسه[4] خودش را ارسال می‌کند. در غیر اینصورت، هیچ پیامی را برای پراسس‌های اطرافش نمی‌فرستد. در هر مرحله مانند L پیام‌ها انتظار می‌رود که مسافت 2^L را طی کنند. هر پراسس با دریافت پیام، در صورتی که مقدار محتوای پیام (مقدار uid موجود در هر پیام) بزرگتر از مقدار شناسه خودش بود، این مقدار را برای پراسس بعدی ارسال می‌کند یا در صورتی که مسافت آن پیام به اتمام رسیده باشد، آن را برمی‌گرداند؛ اما اگر مقدار شناسه موجود در پیام ورودی کوچک‌تر از مقدار شناسه خودش باشد، این مقدار شناسه پیام را فراموش می‌کند و از بین می‌برد.[5] سپس در مرحله بعد، پراسس در صورتی که در این مرحله گذشته، از هر دو طرف پیام خود را (همان شناسه خود را به نوعی) دریافت کرده بود، این بار پیام را (شناسه خودش را) برای هر دو پراسس اطراف با طول مسافت دو برابر سری قبلی 2^{L+1} می‌فرستد. در این الگوریتم هر گاه پراسسی شناسه ارسال شده از سمت خودش یک دور گراف را پیشمایش کند و به خودش برسد، اجرای الگوریتم تمام می‌شود و آن پراسس خودش را به عنوان لیدر انتخاب می‌کند.

تحلیل پیچیدگر این الگوریتم به صورت زیر است:

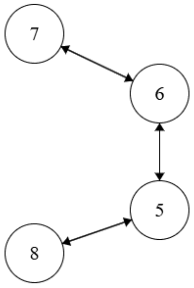
- **پیچیدگی زمانی:** O(n)
- **پیچیدگر پیامی:** O(n log n)

الگوریتم ارائه شده در مقاله: برای توضیح این الگوریتم، باید ابتدا به تعریف مفهوم سرپرستی[6] بپردازیم. این تعریف به این صورت است که هنگامی که پراسسی پیامی دریافت می‌کند مقدار محتوای آن پیام را با شناسه خودش مقایسه می‌کند؛ اگر مقدار موجود در این پیام بزرگتر بود، این مقدار را به سرپرستی می‌گیرد، در غیر این صورت (مقدار شناسه خودش بزرگتر بود از شناسه دریافتی) هیچ کاری نمی‌کند. حالا به سراغ توضیح الگوریتم موجود در این مقاله می‌رویم. در این مقاله گفته شده مطابق با الگوریتم HS اصلی عمل می‌کنیم؛ اما با یک سری تفاوت‌هایی. این تفاوت‌ها به این صورت هستند که در هر مرحله، وقتی پراسسی، پیامی را دریافت می‌کند در صورت نیاز، عملیات جایگزینی را انجام می‌دهد.(اگر نیاز بود در اصل یعنی اگر محتوای پیام ورودی بیشتر از شناسه خودش بود) سپس هر گاه پراسسی پیامی را دید که محتوایش از مقدار شناسه خودش بیشتر بود آن پراسس هم عملیات سرپرستی را انجام می‌دهد و هم اینکه به حالت غیرفعال[7] در می‌آید و از این به بعد دیگر هیچ پیامی را نمی‌فرستد. پس اگر بخواهیم یک بار دیگر از اول بگوییم، اینطوری میشود که: هر پراسس در اجرای این الگوریتم در ابتدا فعال[8] است و پیامی را می‌فرستد به طول یک، سپس در سری‌های بعدی این مقدار طول پیام (مسافت پیام) را دو برابر می‌کند. هر پراسس در صورتی که فعال باشد با دریافت پیامی که محتوای آن بزرگتر از شناسه خودش باشد، به حالت غیر فعال در میاد و مقدار محتوای آن پیام مشاهده شده را به سرپرستی می‌گیرد؛ اما اگر پراسس غیر فعال باشد، مقدار محتوای پیام ورودی را با مقداری که آن پراسس به سرپرستی گرفته مقایسه می‌کند. در صورتی که این مقدار (مقدار موجود در محتوای پیام) بیشتر بود، این مقدار را به سرپرستی می‌گیرد و آن را برای پراسس بعدی می‌فرستد و غیرفعال میماند؛ در غیر اینصورت، (اگر محتوای پیام ورودی از مقداری که آن پراسس به سرپرستی گرفته کمتر بود) این مقدار را (مقدار پیام را) از بین می‌برد و دیگر برای پراسس بعدی ارسال نمی‌کند و غیرفعال میماند. شرط پایان این الگوریتم هم این

[1] Circular graph
[2] Undirected graph
[3] Token
[4] Uid
[5] Discard
[6] Adoption
[7] Inactive
[8] active

است که یک پراسس **فعال**، پیامی را دریافت کند که حاوی شناسه خودش باشد و در جهت فرستاده شدن باشد. (در حالتی نباشد که پراسس در حال دیافت پیام‌های فرستاده شده از سمت خودش در مرحله قبل باشد)

همانطور که در توضیح این دو الگوریتم دیدیم، این دو الگوریتم تقریبا مشابه با یکدیگر هستند، فقط در الگوریتم دوم تعداد پیام‌های ارسالی کاهش می‌یابد. چرا؟ به این دلیل که در الگوریتم اصلی پراسس‌ها تنها غیرفعال میشدند گویا و بعد از غیرفعال شدن، مقدار پیام موجود در هر سری را با مقدار شناسه خودشان مقایسه میکردند ولی در حالت دوم، این مقایسه بین مقدار موجود در پیام ورودی و و مقداری که پراسس به سرپرستی گرفته بود انجام می‌شد. برای همین در این روش تعداد کل پیام‌های رد و بدل شده در کل کمتر می‌شود. با یک مثال این اتفاق را توضیح می‌دهیم. در شکل زیر عدد روی هر پراسس برابر با شماره آن پراسس می‌باشد.



این شکل قسمتی از یک سیستم توزیع شده است که در آن صرفا تعدادی از پراسس‌ها آورده شده‌اند. در اجرای الگوریتم HS برای این تعداد فرآیندها به این صورت می‌شود که در انتهای مرحله اول پراسس‌های پنج و شش مقداری بزرگتر از مقدار خودشان را می‌بینند و دیگر در مرحله بعد پیامی ارسال نمی‌کنند. همچنین فرض کنیم که پراسس‌های هفت و هشت لاقل تا انتهای مرحله اول (مرحله‌ای که در آن پیام‌ها به طول یک ارسال شدند)، بزرگترین مقدار در بین پراسس‌های اطرافشان هستند. سپس در مرحله بعدی پراسس شماره هفت، پیامی به پراسس شماره شش ارسال می‌کند، آن هم چون در با مقدار خودش مقایسه می‌کند، پس اجازه عبور آن را می‌دهد و برای پراسس شماره پنج ارسال می‌کند. سپس پراسس شماره پنج هم آن مقدار را با مقدار موجود در خودش مقایسه می‌کند و چون از مقدار خودش بزرگتر استو چون طول ارسال پیام به پایان رسیده، پیام را برمی‌گرداند و همچنان پراسس شماره هفت اجازه ارسال پیام را در مرحله بعدی، که باید پیام‌هایی به طول چهار ارسال کنند، دارد و همچنین تعداد پیام هم برای بازگرداندن پیام پراسس هفت به خودش ارسال می‌شود. اما در مرحله عدی، که پراسس هفت باید پیام‌هایی به طول چهار بفرستند، پیام آن بعد از رسیدن به پراسس هشت، از بین می‌رود و دیگر ارسال نمی‌شود.

اما در اجرای الگوریتم ارائه شده در مقاله، در انتهای مرحله اول، پراسس‌های پنج و شش به ترتیب مقادیر هشت و هفت را به سرپرستی گرفته‌اند. پس در مرحله دوم، مرحله‌ای که باید پیام‌هایی به طول دو ارسال کنند، پیام پراسس هفت به پراسس شش ارسال می‌شود، چون مقدار پسام ورودی (هفت) بزرگترمساوی از مقداری است که پراسس پنج به سرپرستی گرفته است (مقدار هفت را به سرپرستی گرفته بود) پس این پیام را برای پراسس بعدی ارسال می‌کند. سپس پراسس شش با توجه به اینکه پیام ورودی، مقدار هفت را دارد ولی خودش مقدار هشت را به سرپرستی گرفته بود، با مقدار هشت وقتی مقایسه می‌کند و میبیند این مقدار کمتر است، پس آن را زا بین می‌برد و دیگر پیام پراسس هفت را برای ارسال شدن به سمت خود پراسس هفت (به دلیل تمام شدن مسافت مجاز برای طی شدن در این مرحله باید اگر نیاز بود و شرایط برقرا بود، این پیام را برای پراسس هفت برمی‌گرداند) برنمی‌گرداند. پس تعداد پیام‌های کمتری نسبت به حالت الگوریتم HS عادی ارسال می‌شود.

حال می‌توان همین مثال زده شده با استفاده از اعداد را حذف کرد و به جای آن‌ها متغیر گذاشت. بنابراین اینچنین ثابت می‌شود که الگوریتم پیشنهاد شده تعداد پیام‌های کمتری را در کل ارسال می‌کند. که این به طور شهودی هم مشخص بود، چون انگار مقایسه‌ای که ممکن بود هر جای مسیر حرکت پیام انجام بشه، در این الگوریتم پیشنهادی در ابتدای کار انجام می‌شود به دلیل مفهوم سرپرستی. اما از لحاظ زمانی به نظر می‌رسد که همان مقدار قبلی باقی می‌ماند. چون شرط پایان هم همانند حالت قبل است و با کم شدن تعداد پیام‌های ارسالی تاثییری روی آن ایجاد نمی‌شود.