به نام خدا

پاسخ تمارین سری اول

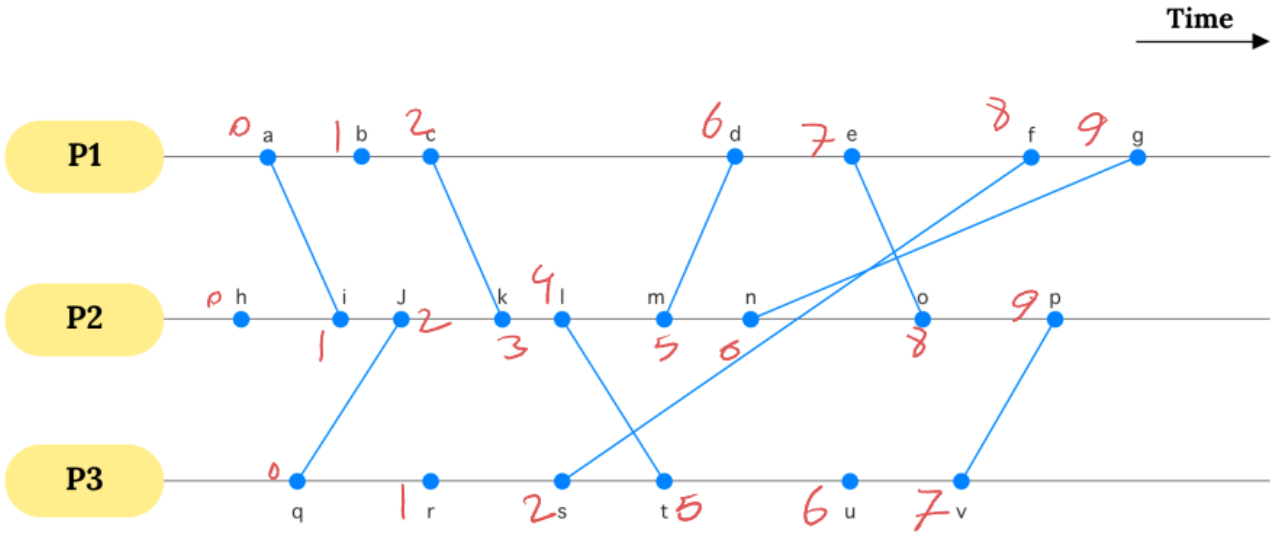
حسام مومیوند فرد

۸۱۰۸۰۳۰۶۳

فهرست

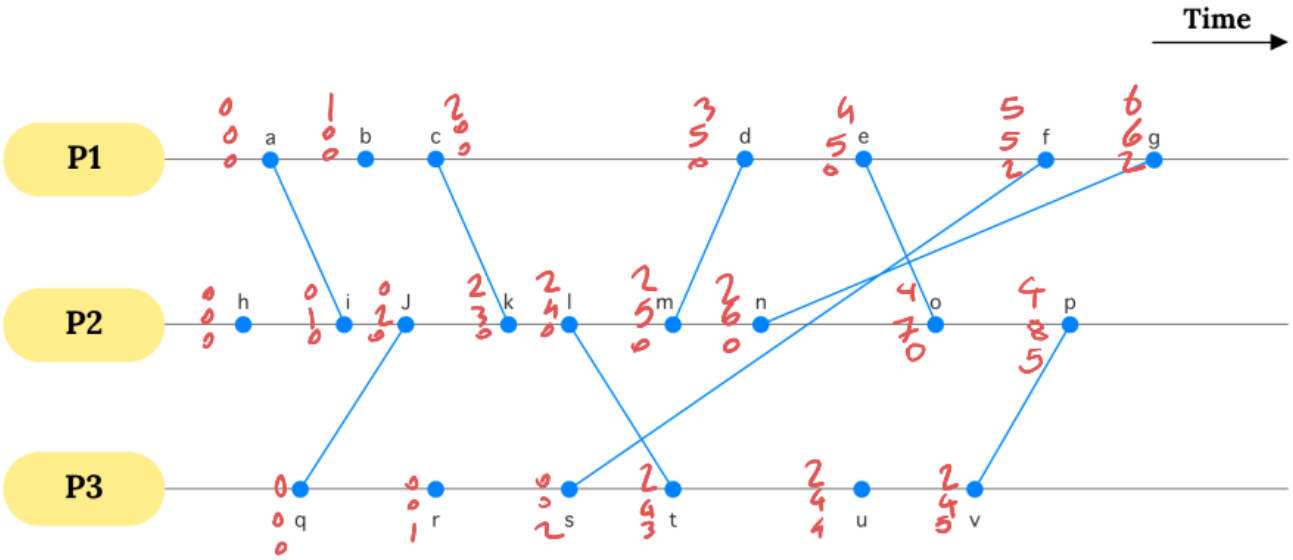
سوال اول

بخش اول



تصویر 1

بخش دوم



تصویر 2

سوال دوم

بخش اول

می‌توان این مورد را به حالتی تشبیه کرد که هرکدام از فرایند ها به مقدار عدد ست شده رویداد داخلی داشته اند و سپس مراحل دریافت و ارسال پیام بین فرآیندها شروع شده است و چون ادامه‌ی کار مطابق الگوریتم لمپورت پیش می‌رود پس تمام ویژگی های آن را خواهد داشت.

بخش دوم

به علت اینکه در طی مراحل آپدیت کردن ساعت برداری همواره به ازای تک تک درایه‌ها ماکسیمم گرفته می‌شود مشکلی در سازگاری به وجود نخواهد آمد. اما در صورتی که یک فرآیند ایندکس ساعت مربوط به فرآیند دیگری را به صورت رندوم به مقدار بزرگتری از ایندکس آن فرآیند در بردار ساعت خودش ست کند آنگاه سازگاری قوی نقض خواهد شد.

برای مثال اگر در لحظه‌ی اولیه بردار ساعت پردازش شماره 1 برابر با [10,20] باشد در صورتی که بردار پردازش شماره ۲ برابر با [10,5] باشد آنگاه به علت بزرگتر بودن بردار پردازش شماره 1 باید علت پردازش شماره 2 باشد اما چنین نیست و هر دو اولین رویداد های پردازش ها هستند پس سازگاری قوی نقض شده است.

سوال سوم

اگر ما برای n فرآیند n-1 ساعت رو نگه داریم اون موقع دو حالت خواهیم داشت:

۱. تمام فرآیندها ساعت مربوط به یک فرآیند خاص مثل m رو نگه نمیدارن

۲. هر فرآیند ساعت مربوط به یک فرآیند (به صورت رندوم) نگه نمیداره.

در حالت اول خب اگه دوتا رویداد از طریق m با هم دیگه رابطه علی داشته باشن چون ما نمیتونیم زمان m رو اندازه گیری کنیم پس نمیتونیم رابطه علی رو نشون بدیم و سازگاری کلا نقض میشه و کار نمیکنه

در حالت دوم هم خب قطعا وجود خواهند داشت دو رویدادی که ساعت فرآیند مشترکی رو نگه ندارن (طبق اصل لانه کبوتری اگه ما n تا ایونت داشته باشیم و n-1 ایندکس برای ساعت پس حتما دوتا ایونت وجود دارن که ساعت یک پردازش خاص رو نگه نمیدارن) و اگه از طریق اون فرآیند با هم دیگه رابطی علی داشته باشن باز هم سازگاری نقض میشه چه برسه به سازگاری قوی و ...

سوال چهارم

هدف ما اینه که ثابت کنیم در هر زمانی هیچ فرآیندی مثل k وجود نداره که در بردار ساعتش برای فرآیند دیگه ای مثل i زمانی رو ذخیره کرده باشه که از زمان ذخیره شده توسط خود فرآیند i برای خودش در بردار ساعتش بزرگتر باشه.

ما از اینجا شروع میکنیم که در لحظه‌ی اول تنها فرآیندی که ایندکس i ام در بردار ساعتش برابر ۱ هست خود i هست و مابقی فرآیند ها ۰ رو نگه داشتن توی اون ایندکس.

فرض استقرا رو لحظه‌ی n قرار میدیم و میگیم توی لحظه‌ی n ایندکس i در بردار ساعت فرآیند i از ایندکس i در بردار ساعت تمامی فرآیندهای دیگه بزرگتره.

حکم استقرا اینه که ثابت کنیم در لحظه‌ی n+1 هم این شرط برقرار هست.

با حالت بندی میریم جلو:

در لحظه n+1 دو حالت داریم:

۱. رویدادی از i به یکی از فرآیند های دیگه رخ میده پس مقدار ایندکس i ام بردار ساعت فرآیند مذکور برابر میشه با مقداری که روی بردار ساعت i ذخیره شده که خب درسته و اثبات ما کامل میشه

۲. رویداد درونی برای i به وجود میاد و مقدار ایندکس i ام بردار ساعت این فرآیند یک واحد میره بالاتر که چون برای مقدار قبلی طبق فرض استقرا بزرگترین مقدار در بیان تمام بردارهای ساعت رو دارا بود پس در این لحظه هم کماکان بزرگترین مقدار هست و باز هم درسته و اثبات ما کامل میشه.

در هر دو حالت استقرا جواب داد و اثبات کامل شد.

سوال پنجم

برای حل این سوال ما باید هم → و هم ← رو ثابت کنیم

بخش اول →

در صورتی که بدونیم $e_i \rightarrow e_j$ اون موقع میخوایم ثابت کنیم که : $vt_{e_i} < vt_{e_j}$

با داشتن فرض و اینکه میدونیم برای آپدیت بردار ساعت از رابطه‌ی R_2 به صورت زیر استفاده میشه:

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

آنگاه لزوما:

$$vt_{e_i} \leq vt_{e_j}$$

و چون میدونیم که در هر لحظه بزرگترین مقدار در ایندکس j تمام بردار های ساعت در بردار ساعت فرآیند j ام قرار داره پس نتیجه میگیریم که :

$$vt_{e_i} < vt_{e_j}$$

بخش دوم ←

در صورتی که بدونیم $vt_{e_i} < vt_{e_j}$ به دنبال اثبات $e_i \rightarrow e_j$ هستیم.

با داشتن فرض و اینکه میدونیم هرفرآیند با بردار ساعتی معادل با تماما صفر به جز ایندکس مربوط به شماره خودش که برابر یک بوده شروع کرده و اینکه طبق فرض الان درایه‌ی i ام از بردار e_j بزرگتر یا مساویه با درایه i ام از بردار ساعت e_i پس حتما یه رابطه علی به صورت $e_i \rightarrow e_j$ وجود داشته که تونسته صفر رو تغییر بده.

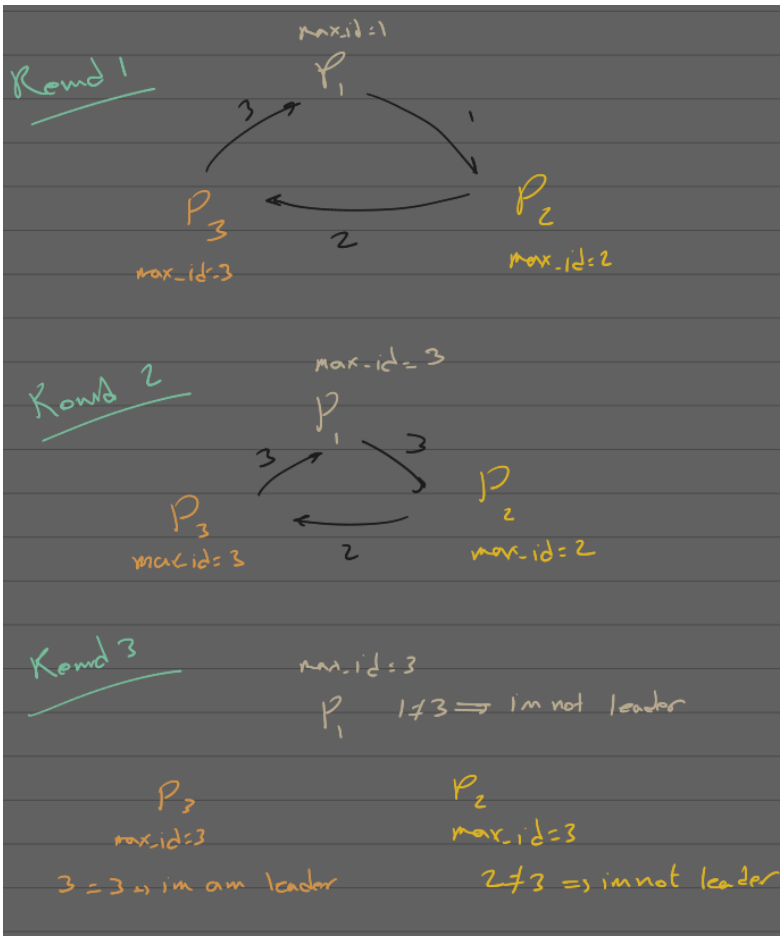
سوال ششم

این یه الگوریتم ساده برای پیدا کردن رهبر توی شبکست که مراحل زیر رو طی میکنه:

1. شروع و مقدار دهی: هر گره در شبکه مقدار حداکثر شناسه ای که تا حالا دیده رو در متغیر max-uid ذخیره میکنه. که خب طبیعتا در ابتدا مقدارش برابر میشه با شناسه خودش
2. پخش شناسه‌ها در هر مرحله: در هر دور از اجرا هر گره مقدار max-uid خودشو به تمامی گره های همسایه خودش (اونایی که مستقیما باهاش در ارتباطن) میفرسته.
3. به روز رسانی حداکثر شناسه: وقتی یک گره پیامی از همسایه‌هاش دریافت میکنه مقدار max-uid خودشو به حداکثر مقدار بین max-uid فعلی و مقدار دریافتی بروز میکنه.
4. تعیین رهبر پس از diam دور: (diam همون قطر شبکست که میشه حداکثر فاصله بین دوتا نود شبکه) هر گره مقدار max-uid رو بررسی میکنه و اگه برابر بود با uid خودش پس لیدره در غیر این صورت لیدر نیست.

مسئله ای هم که وجود داره اینه که هر نود یه لیست از تمام uid هایی که تا حالا به دستش رسیده نگه میداره و max-uid در هر دور رو از بین تمام چیزهایی که تا به حال به دستش رسیده و تو لیست ذخیره کرده انتخاب میکنه.

یه مثال هم تو تصویر زیر هست براش:



نصویر 3

سوال هفتم

این الگوریتم هدفش بهینه سازی انتخاب لیدر در الگوریتم HS معمولیه.

تو بخش مقدمات یه توضیحاتی داده در مورد اهمیت انتخاب لیدر و نقش لیدر توی یه سیستم توزیع شده.

در قسمت کارهای پیشین اشاره کرده به چند تا از الگوریتم هایی که تا حالا برای انتخاب لیدر توسعه داده شدن و الگوریتم های کارایی هستن.

اما میرسیم به خود این الگوریتم:

کلیات کار اینه که این الگوریتم دو دسته نود تعریف میکنه: active , non-active و تو هر مرحله هر نود active بزرگترین UID که تا حالا دیده رو از هر دو طرف منتشر میکنه و در هر مرحله دامنه ارسالش ۲ برابر میشه (۱و۲و۴و ...)

انتخاب رهبر: اگر گره ای UID خودش رو از هر دو سمت دریافت کنه یعنی خودش رهبره و الگوریتم تموم میشه.

حالا نود active , non-active چین؟
گره‌ای که هنوز در رقابت برای رهبر شدنه رو active و کسی که دیگه در رقابت نیست رو non-active مینامیم.

چی میشه که یه نود active تبدیل میشه به non-active ؟ زمانی که UID به دستش برسه که بزرگتر از مال خودشه و اون نود از اون مرحله به بعد به جای انتشار UID خودش ، بزرگترین UID که دیده رو منتشر میکنه. (مفهوم adoption که تقریبا میشه گفت مهم ترین مفهوم این الگوریتم هم هست)

در واقع این کار کردن نود های غیرفعال و اینکه به انتشار بزرگترین UID که دیدن کمک کردنشون باعث شده که الگوریتم کارایی بهتری نسبت به حالت معمول HS داشته باشه.

پیچیدگی زمانی: از مرتبه $O(n)$.

پیچیدگی پیامی: از مرتبه $\theta(n \log n)$.