تمرین– سری صفر

سوالاتی که در این متن مطرح شده است لزوما جواب یکتا و مشخصی ندارد و تنها هدف فکر کردن به این مسائل، تحلیل و ارائه‌ی راه‌حل‌هایی است که مشکلات را از سر راه بردارد. برای فکر کردن می‌توانید با یکدیگر هم فکری کنید و یکدیگر را به چالش بکشید. اما لطفا نوشتن را به تنهایی انجام دهید.

مساله ۱. شام فیلسوفان[1]

پنج فیلسوف یونانی دور میزی نشسته‌اند. هر کدام دارای یک پیاله برنج و یک چوب غذاخوری[2] *در هر طرف آن هستند. چوب سمت راست هر فیلسوف، چوب سمت چپ فیلسوف کناری‌اش می‌باشد. حال، قانون غذاخوری را به صورت زیر در نظر بگیرید:*

- *هر فیلسوف مدتی فکر می‌کند، مدتی غذا می‌خورد، و مدتی منتظر می‌ماند.*
- *هر فیلسوف برای غذا خوردن باید هر دو چوب سمت چپ و راستش را در اختیار داشته باشد.*
- *تنها ارتباط فیلسوفان با یکدیگر از طریق برداشتن و گذاشتن چوبهاست (فرض کنید آن‌ها نه حرف می‌زنند و نه می‌نویسند).*

اینک فرض کنید هر فیلسوف از الگوریتم زیر برای غذا خوردن استفاده کند:

- *چوب سمت راست را بردار و اگر در اختیار فرد دست راستی است منتظر بمان.*
- *چوب سمت چپ را بردار و اگر در اختیار فرد دست چپی است منتظر بمان.*
- *غذا بخور.*

چند چیز ممکن است اتفاق بیفتد. چنانچه تمام فیلسوفان به طور همزمان تصمیم به غذا خوردن بگیرند، همگی مرحله‌ی اول الگوریتم را با موفقیت پشت سر می‌گذارند و در مرحله‌ی دوم به طور نامحدود منتظر خواهند ماند. به این وضعیت بن‌بست[3] *گفته می‌شود.*

[1] این مساله را دایکسترا در پاییز سال ۱۹۶۵ به عنوان سوال امتحانی در دانشگاه فنی آیندهوون مطرح کرد.

[2] chopstick

[3] deadlock

حال فرض کنید یکی از فیلسوفان که یک چوب به دست آورده و در مرحله‌ی دوم الگوریتم در انتظار مانده است، چوبش را زمین بگذارد و مدتی ساکت بنشیند و تماشاگر غذا خوردن فرد بغل دستی‌اش بماند. در این حالت این احتمال وجود خواهد آمد که فیلسوف فداکار هیچگاه فرصت غذا خوردن نیابد. به این وضعیت قحطی[۴] گفته می‌شود.

حتی اگر تمام فیلسوفان نیز بتوانند غذا بخورند، ممکن است بعضی بیشتر از بعضی دیگر فرصت غذا خوردن بیابند. این وضعیت نیز بی‌عدالتی خوانده می‌شود.

برای مشکلات مطرح شده در الگوریتم فوق سعی کنید راه‌حل‌هایی ارائه کنید و الگوریتم را اصلاح کنید. آیا مشکلات دیگری ممکن است پیش آید؟

مشابه این‌گونه مسائل که در مساله‌ی شام فیلسوفان پیش می‌آید، غالبا در شبکه‌های رایانه‌ای بروز می‌کند. به عنوان مثال، رایانه‌های یک شبکه‌ی محلی معمولا از یک سیم یا مجرای ارتباطی به صورت اشتراکی استفاده می‌کنند و در هر لحظه بر روی آن تنها یک پیام قابل ارسال می‌باشد. اگر تمام پایگاه‌ها بخواهند در یک لحظه پیام ارسال کنند، همگی ناموفق خواهند ماند. اگر همگی بلافاصله دوباره سعی کنند، باز هم موفق نخواهند شد. این مشابه مساله‌ی بن‌بست است. از سوی دیگر، اگر یک پایگاه همیشه امکان ارسال پیام پیدا کند، دیگران ممکن است با قحطی روبه‌رو شوند که در این صورت پروتکل، ناعادلانه خواهد بود. مسائل مشابهی از این دست، در تخصیص منابع در سیستم عامل نیز مطرح است.

در ادامه می‌خواهیم که الگوریتم خود را شبیه‌سازی کنید. برای این کار شما باید یک $server$ و یک $client$ طراحی کنید. کلاینت‌ها در واقع فیلسوفان خواهند بود و سرور در واقع پیکربندی و مدیریت مساله را به عهده دارد. قبل از شروع، کلاینت‌ها به سرور درخواست می‌دهند و خود را به عنوان دریافت کننده‌ی غذا رجیستر می‌کنند. سرور به هر کدام عددی نسبت می‌دهد از ۱ تا n و این عدد به عنوان مشخصه‌ی کلاینت به فیلسوف باز گردانده می‌شود. در نهایت همه‌ی آن‌ها به صورت دایره‌ای قرار خواهند گرفت. غذا خوردن زمانی شروع می‌شود که یکی از کلاینت‌ها به سرور درخواست شروع می‌دهد و سرور به تمامی کلاینت‌ها این را اعلام می‌کند. و در نهایت هر کدام از فیلسوفان براساس الگوریتم شما، درخواست‌های چوب چپ و راست و غذا خوردن را خواهند داشت. تمامی مواردی که در صورت مساله مطرح شده است باید در برنامه‌ی شما رعایت شود. در ادامه هر کلاینت تنها دستورات زیر را می‌تواند داشته باشد:

- درخواست چوب سمت راست را بدهد. در این حالت اگر چوب سمت راست آزاد باشد، سرور چوب سمت راست فیلسوف را در اختیار او قرار می‌دهد و آن را به حالت غیرآزاد در می‌آورد. در غیراین صورت به فیلسوف پیغام مشغول بودن چوب برگردانده می‌شود.

[4] starvation

- *درخواست چوب سمت چپ داده شود. در این حالت اگر چوب سمت چپ فیلسوف آزاد باشد و فیلسوف چوب سمت راستش را در اختیار قرار داشته باشد، چوب سمت چپ در اختیار او قرار می‌گیرد و چوب به حالت غیرآزاد درآورده می‌شود. در غیر این صورت پیغام مشغول بودن چوب برگردانده می‌شود یا اینکه ابتدا باید چوب سمت راست را در دسترس داشته باشد.*
- *کلاینت درخواست غذا خوردن به مدت t ثانیه را می‌دهد و سرور بسته به الگوریتم شما این دسترسی را به او می‌دهد یا خیر.*

به جز دستور شروع، هیچ‌یک از اطلاعات فیلسوفی توسط سرور در اختیار دیگر فیلسوفان قرار نمی‌گیرد. هدف نهایی این است که بعد از زمان T مجموع غذاخوردن فیلسوفان بیشترین زمان ممکن باشد و اختلاف زمان غذا خوردن فیلسوفان کمینه باشد.

مساله ۲. مساله تناقض دو ژنرال[5]

دو ارتش، به رهبری آشیل و پاتروکلوس در حال آماده‌سازی برای حمله به شهر مستحکم تروآ هستند. ارتش‌ها در نزدیکی شهر، یکی در بخش شمالی جزیره و یکی در بخش جنوبی جزیره قرار گرفته‌اند و تنها راه ارتباطی دو ژنرال ارسال پیام توسط افراد پیام‌رسان از طریق خط ساحلی است که اکثر آن توسط ارتش هکتور زیر نظر است. بنابراین ممکن است افراد پیام‌رسان توسط نیروهای هکتور اسیر شوند و پیغام‌رسان به مقصد نرسد.

با توجه به استحکام تروآ، برای تصرف شهر لازم است که آشیل و پاتروکلوس همزمان به شهر حمله کنند. در حالی که هر دو ژنرال برای حمله توافق کرده‌اند، زمانی را برای حمله مشخص نکرده‌اند. بنابراین نیاز است که در مورد زمان حمله با یکدیگر ارتباط برقرار کنند. مساله اینجاست که پیغام‌رسان‌ها از هر طرف ممکن است توسط نیروهای هکتور اسیر شوند و پیغام به مقصد نرسد.

در مورد این مساله فکر کنید و اینکه آیا می‌توان راه‌حلی برای آن یافت؟ چه پیشنهاداتی برای پیروزی آشیل و پاتروکلوس بر تروآ می‌توانید داشته باشید.

این مساله به سادگی در دنیای نرم‌افزار می‌تواند اتفاق بیفتد. مثلا در یک خرید آنلاین ارتباط بین کلاینت و سرور را در نظر بگیرید که کلاینت می‌خواهد درخواست خرید یک کالا را بدهد و مطمئن باشد که سرور آن را دریافت کرده و در نظر گرفته است. به راحتی هر کدام از پیغام‌ها می‌تواند دچار مشکل شود.

مساله ۳. ژنرال‌های بیزانسی[6]

امپراتوری روم شرقی که با عنوان‌های امپراتوری بیزانس و بیزانتیوم نیز شناخته می‌شود، ادامهٔ امپراتوری روم در سرزمین‌های شرقی این امپراتوری (بالکان، آناتولی، شامات، مصر) در طول دوران باستان متأخر و قرون وسطی بود و پایتخت آن کُنستانتینوپل (استانبول امروزی، که با نام بیزانتیوم تأسیس شد) بود.

[5] Two General's Paradox
[6] The Byzantine Generals Problem

گروهی از فرماندهان بیزانس شهری را محاصره کرده‌اند و هر کدام بخشی از ارتش بیزانس را رهبری می‌کنند و هر یک در مکانی مختلف مستقرند. راه دسترسی بین آن‌ها از طریق پیک و چاپار است. فرماندهان قصد دارند در مورد حمله به شهر یا عقب‌نشینی تصمیم بگیرند. گروهی ممکن است میل به حمله و گروهی ممکن است میل به عقب‌نشینی داشته باشند. اگر بخشی از ارتش حمله کند، شکست خواهند خورد و از بین خواهند رفت. پس مهم است که همه‌ی فرماندهان تصمیمی واحد اتخاذ کنند. حمله بخشی از فرماندهان و نیروهای تحت امرشان خطری بیش از حمله مشترک یا عقب‌نشینی سازمان‌یافته دارد.

مشکل اول وجود فرماندهان خائن است که قصدشان برهم زدن این هماهنگیست. آن‌ها ممکن است به گروهی از فرماندهان اطلاع حمله دهند و به گروهی دیگر اطلاع عقب‌نشینی. یا اینکه زمان متفاوتی برای حمله به فرماندهان دیگر ابلاغ کنند. با این خدعه هماهنگی از بین می‌رود. مشکل دوم دوری فرماندهان از یکدیگر و قابل اعتماد نبودن پیک‌هاست. یعنی ممکن است که پیک‌ها به مقصد نرسند یا پیغام را اشتباه برسانند.

با فرض وجود تنها مشکل اول آیا می‌توانید راه‌حلی ارائه کنید که اگر ۴ فرمانده وجود داشته باشد که تنها ۱ فرمانده خائن باشد همه‌چیز به درستی انجام شود؟ اگر سه فرمانده باشند و یک فرمانده‌ی خائن چطور؟

اگر مشکل دوم نیز اضافه شود چه راه‌حل‌هایی برای اجماع در مورد حمله یا عقب‌نشینی می‌توان در نظر گرفت؟

این مساله نیز در دنیای نرم‌افزار نمونه‌های زیادی دارد. به طور مثال در مساله‌ی خرید از فروشگاه، فرض کنید که این‌بار فروشگاه، خریدار و بانک هر سه درگیر هستند و همگی می‌خواند از پرداخت و خرید مطمئن باشند و از طرفی هر کدام از خریدار و فروشنده ممکن است بخواهند در پروسه تقلبی ایجاد کنند، چطور می‌توان جلوی آن را گرفت مشابه مساله‌ای است که ژنرال‌های ما درگیر آن بودند.

مساله ۴. استفاده از شبکه‌ی ریلی

شبکه‌ی ریلی کشور را در نظر بگیرید. این شبکه در بخشی از مسیرها، یک‌طرفه است و در بخشی از مسیرها دوطرفه. یعنی بخشی از مسیرها از دو طرف مورد استفاده قرار می‌گیرند. حال فرض کنید دو قطار از دو طرف مسیر همزمان به یکی از این بخش‌های مسیر برسد، چه اتفاقی می‌افتد؟ اگر درست مدیریت نشود ممکن است فاجعه‌ای رخ دهد. در این مساله می‌خواهیم به این مساله بپردازیم و جلوی فاجعه را بگیریم. مهندسان برای جلوگیری از برخورد قطارها و حصول اطمینان از این‌که در هر زمان فقط یک قطار در بخش مشترک راه‌آهن وجود دارد، از نشانبرها استفاده می‌کنند. نشان‌برها مثل چراغ راهنمایی عمل می‌کنند و تا وقتی که یک قطار در ناحیه‌ی مشترک وجود دارد، چراغ را برای دیگری قرمز نگاه می‌دارد و بدین ترتیت این اطمینان به وجود می‌آید که در هر لحظه، تنها یک قطار در ناحیه‌ی مشترک یا ناحیه‌ی بحرانی وجود خواهد داشت. به این پدیده دو به دو ناسازگاری می‌گویند.

شکل ۱: شبکه‌ی ریلی ایران

در این تمرین می‌خواهیم به این مساله بپردازیم. این مساله در سیستم‌های نرم‌افزاری بسیار رخ می‌دهد. فرض کنید برنامه‌ای می‌خواهیم بنویسیم که در آن تعداد N قطار با شماره‌های ۱ تا n در اول مشخص می‌شوند و تعداد M تقاطع (ناحیه‌ی ریلی مشترک) با شماره‌های ۱ تا m در نظر گرفته می‌شوند. در زمان t هر کدام از قطارها به صورت تصادفی درخواست استفاده از ریل x را به مدت زمان t. می‌دهد. اگر این ریل آزاد باشد به این درخواست پاسخ داده می‌شود و ریل x به مدت زمان t. اشغال می‌شود. و اگر ریل آزاد نباشد درخواست بی‌پاسخ می‌ماند و قطار باید منتظر بماند. در ابتدا تمامی مسیرها باز هستند و قطارها شروع به درخواست می‌دهند. در خواست ۰ معادل درخواست ریل‌های یکطرفه است و بنابراین همیشه پاسخ درست به آن‌ها داده خواهد شد. برنامه‌ای بنویسید که با استفاده از برنامه‌نویسی همروند این مساله را شبیه‌سازی کند و در هر ثانیه وضعیت هر خط راه‌آهن و وضعیت هر قطار را نمایش دهد.

برای شفاف شدن مساله، ورودی مساله را به شکل زیر در نظر بگیرید:

$n \quad m$
$id۱ \quad l۱ \quad label$
$id۲ \quad l۲ \quad label$
...
$id۱ \quad v۱$
$id۲ \quad v۲$
...

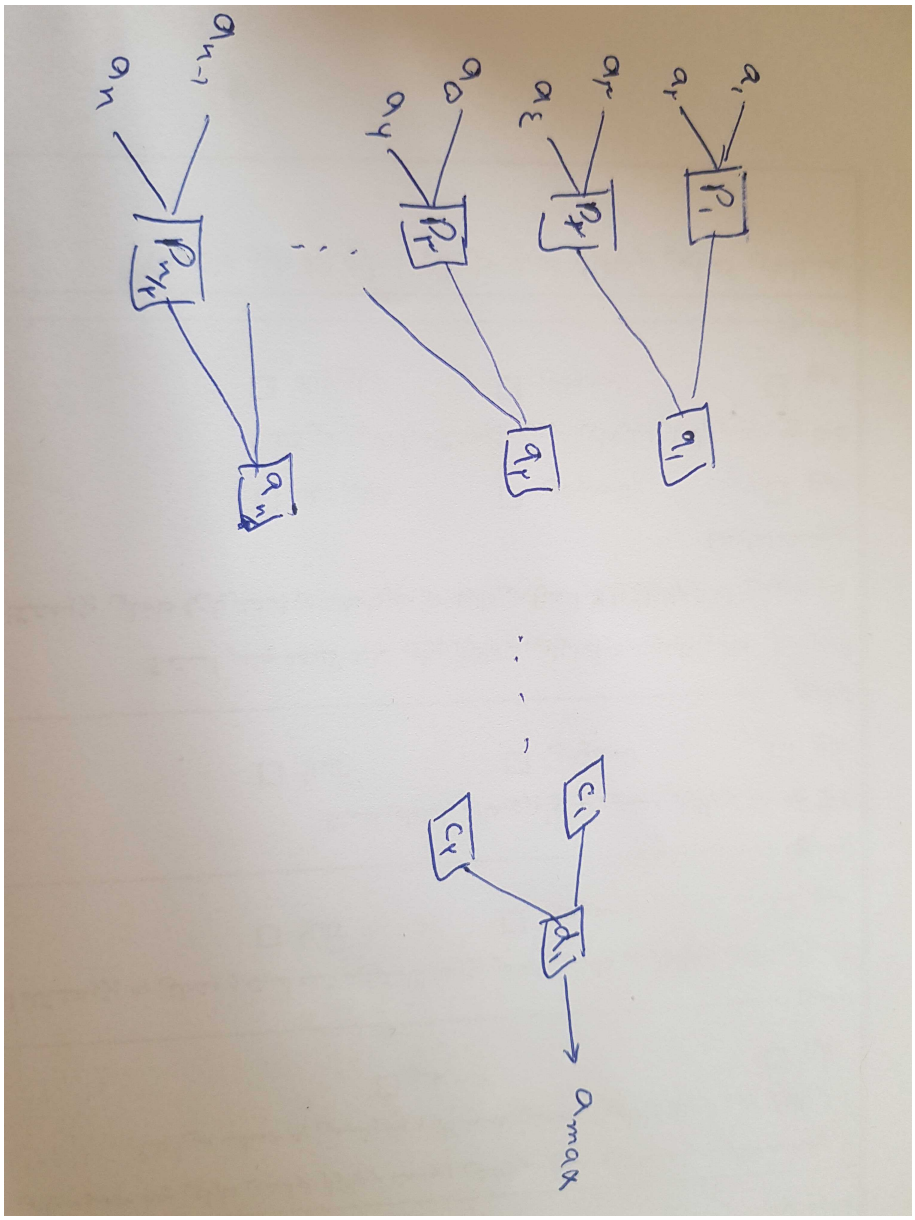
در خط اول n تعداد ریل‌ها و m تعداد قطارها خواهد بود. در n خط بعدی مشخصات ریل‌ها داده می‌شود. در هر خط آن شناسه‌ی ریل می‌باشد که قطارها برای درخواست از آن استفاده می‌کنند. مولفه‌ی بعدی طول آن بخش از ریل است و در نهایت لیبل که مشخص کننده‌ی مشترک بودن ریل یا غیر مشترک بودن آن است. در ادامه m خط ورودی مشخصات قطارها را نمایش می‌دهد که شامل مشخصه و سرعت قطار است، فرض شده است قطارها با سرعت ثابت در حرکت هستند. برای نواحی غیر مشترک، برای ساده‌سازی مساله، فرض می‌کنیم حتی اگر دو قطار با سرعت ثابت در حرکت هستند تصادفی رخ نمی‌دهد. در نهایت هدف کمینه کردن زمان انتظار قطارها برای استفاده از ریل‌های مشترک است.

مساله ۵. معماری ریزپردازنده‌ها

فرض کنید تعداد دلخواهی پردازنده‌ی کوچک دارید که قابلیت انجام عملیات مقایسه بین دو عدد را در زمان ۱ انجام می‌دهند، همچنین گنجایش نگهداری ۲ عدد در لحظه‌را دارد، در واقع همان ۲ عددی که مقایسه می‌کند، که می‌تواند آن‌هارا در حافظه‌ی خود ذخیره کند. شما می‌توانید این پردازنده‌ها را به یکدیگر به هر شکلی که دوست دارید متصل کنید. ارتباط این پردازنده‌ها و انتقال اطلاعات بین آن‌ها یک واحد زمانی زمان لازم دارد. همچنین پردازنده‌ها به یک حافظه‌ی مشترک دسترسی دارند که همگی می‌توانند از آن بخوانند و روی آن بنویسند که در واقع از این حافظه برای ورودی و خروجی برنامه استفاده می‌شود.

برای مرتب‌سازی n عدد در حالت عادی و استفاده از یک پردازنده حداقل زمان $n \log n$ را لازم داریم. می‌خواهیم به کمک این پردازنده‌ها، معماری بسازیم که بتوانیم این کار را در زمان بهتری انجام دهیم. روی این مساله فکر کنید و معماری‌هایی که به کمک آن بتوان این کار را در زمان کمتری انجام داد ارائه دهید. بهترین زمان ممکن چقدر است؟

برای شفاف شدن مساله، بیایید یک مثال را با هم حل کنیم. فرض کنید یک آرایه داریم و می‌خواهیم بزرگترین عدد آن را بیابیم. اگر آرایه مرتب شده نباشد حداقل زمان $O(n)$ برای اینکار لازم است. حال بیایید با فرض وجود پردازنده‌ی دلخواه این کار را انجام دهیم. در لایه‌ی ابتدایی فرض کنید که $n/۲$ پردازنده وجود دارد که هر کدام دو عدد از آرایه را به عنوان ورودی می‌گیرند. خروجی هر کدام از این پردازنده‌ها عدد بزرگتر خواهد بود که آن را به پردازنده‌ی بعدی ارسال می‌کند که در لایه‌ی بعدی قرار دارد. لایه‌ی بعدی شامل $n/۴$ پردازنده است که هر کدام به دو پردازنده‌ی لایه‌ی اول متصل است. با همین رویکرد $\log n$ لایه و n پردازنده خواهیم داشت و در زمان $\log n$ بزرگترین عدد آرایه را می‌یابیم. در تصویر زیر می‌توانید این معماری را ببینید:



شکل ۲: معماری پردازنده‌های برای پیدا کردن بزرگترین عدد آرایه

مراجع

[۱] ترجمه ابراهیم نقیب‌زاده مشایخ، *بزرگان دانش رایانه*. انجمن انفورماتیک ایران

[۲] ویکی‌پدیای فارسی

تاریخ تحویل ۱ اسفندماه ۱۴۰۰

موفق باشید.