نام و نام خانوادگی: حامد فتحی شماره دانشجویی: 980172075

سوالات میان ترم

1. در ریاضیات و علوم کامپیوتر ، الگوریتم دنباله ای محدود از دستورالعمل های کاملاً تعریف شده و قابل پیاده سازی رایانه است که معمولاً برای حل یک دسته از مسائل خاص یا انجام یک محاسبه است. الگوریتم ها همیشه بدون ابهام هستند و به عنوان مشخصات برای انجام محاسبات ، پردازش داده ها ، استدلال خودکار و سایر کارها استفاده می شوند. در مقابل ، اکتشافی تکنیکی است که در حل مسئله استفاده می شود و از روش های عملی و / یا برآوردهای مختلف برای تولید راه حل هایی استفاده می کند که ممکن است بهینه نباشند اما با توجه به شرایط کافی باشند. به عنوان یک روش موثر ، یک الگوریتم را می توان در مقدار محدودی از فضا و زمان و با یک زبان رسمی کاملاً مشخص برای محاسبه یک تابع بیان کرد. با شروع از حالت اولیه و ورودی اولیه (شاید خالی) ، دستورالعمل ها یک محاسبه را توصیف می کنند که ، اگر اجرا شود ، از طریق تعداد محدودی از حالت های متوالی کاملاً مشخص پیش می رود ، در نهایت "خروجی" تولید می شود و در یک حالت پایان نهایی خاتمه می یابد. انتقال از یک حالت به حالت دیگر لزوماً قطعی نیست. برخی از الگوریتم ها ، معروف به الگوریتم های تصادفی ، ورودی تصادفی را در خود دارند. دانستن الگوریتم به صورت کامل باعث میشود که مهندسین نرم افزار تمام مسائلی را که ممکن است با آن ها برخورد داشته باشند به صورت کامل بتوانند تجزیه تحلیل کنند و بهترین راه حل یا الگوریتم که در حل مسئله به کمک آن ها می آید را انتخاب کنند. داشتن تفکر الگوریتمی به مهندس نرم افزار قابلیت تفکر منطقی و داشتن حل مسائل به صورت مرحله به مرحله را می دهد.
2. یک تقسیم کننده بیتونیک از چندین مرحله تشکیل شده است که به هریک از مراحل نیمه پاک کننده گفته می شود. هر نیمه پاک کننده یک شبکه مقایسه عمق 1 است که در آن خط ورودی i با خط i + n / 2 برای i = 1 ، 2 ، مقایسه می شود. . . ، n / 2 (ما فرض می کنیم که n برابر است.)
هنگامی که یک توالی bitonic از 0 و 1 به عنوان ورودی به یک نیمه پاک کننده اعمال می شود ، نیمه پاک کننده یک توالی خروجی تولید می کند که در آن مقادیر کوچکتر در نیمه بالا ، مقادیر بزرگتر در نیمه پایین و هر دو نیمه bitonic هستند. در حقیقت ، حداقل یکی از نیمه ها تمیز است - متشکل از هر 0 یا همه 1 - و از این خاصیت است که ما نام "نیمه پاک کننده" را گرفته ایم. (توجه داشته باشید که تمام توالی های تمیز بیتونیک هستند.)
اگر ورودی یک نیمه تمیز کننده یک توالی بیتونیک از 0 و 1 باشد ، در این صورت خروجی خصوصیات زیر را برآورده می کند: هر دو نیمه بالا و نیمه پایین بیتونیک هستند ، هر عنصر در نیمه بالا حداقل به اندازه هر عنصر کوچک است از نیمه پایین ، و حداقل یک نیمه پاک است.

سوالات پایان ترم

1. الگوریتم روشی گام به گام برای حل مسئله است. سال تولید، پشتیبانی ها، مانع زدایی ها دقیقا جایی است که الگوریتم می تواند به کمک آن بیاید. وظیفه مهندسان نرم افزار در واقع پیدا کردن چالش ها و نقاط مورد اشکال در یک سیستم و تبدیل آن به برنامه ای است که توان حل آن مشکل را دارد و دقیقا پیدا کردن راه حل آن مشکل با الگوریتم های مختلف است. به طور مثال کاری که می توان انجام داد استفاده از الگوریتم divide and conquer برای حل مشکل تولید است. به این صورت که می توان مشکل را حل کرد که اگر ما میخواهیم مشکل تولید و مانع زدایی را در سطح کشور حل کنیم باید این مشکل را تقسیم به مشکل تولید در هر استان کشور کنیم. سپس همین پروسه را برای هر استان تکرار کنیم و تا آنجا پیش میرویم که به یک تولیدی مستقل به عنوان واحدی در این مسئله برسیم. سپس با حل مشکل در سطح هر تولیدی ما می توانیم مشکل تولید را در سطح اون شهرستان و سپس شهر و استان و کشور حل کنیم. در واقع نقش مهندس نرم افزار در اینجا این است که بتواند برنامه ای طراحی کند که دقیقا تمام این مراحل را به طور کامل پیاده کرده و دقیقا با استفاده از آن الگوریتم میتواند این مشکل را حل کند. اگر این برنامه به نحوی طراحی شود که بتواند دسته بندی ها را به صورت دقیق انجام داده و به همین ترتیب پیش برود مدیران می توانند با مشاهده مشکلات و دیدن اینکه هر استان چه مشکل به خصوصی دارد یا در واقع اینکه هر استان یا هر بخش بیشتر با چه مشکلاتی دست و پنجه نرم میکند میتواند برای رفع آن تصمیم درستی بگیرد. نقش مهندس نرم افزار این است که برنامه ای که طراحی میکند به بهترین نحو این الگوریتم را پیاده کرده و به قابل فهم ترین راه ممکن آن را نمایش دهد که به راحتی قابل پیگیری و درک باشد. کار دیگری که مهندس نرم افزار میتواند انجام دهد جمع تمام مشکلات در یک دیتابیس و استفاده از آن برای مدیران کارخانه به عنوان مشکلات مشابه

و دادن راه حل به عنوان پیشنهاد به مدیر آن کارخانه است زیرا قطعا خیلی از مشکلات تولیدی ها همپوشانی با هم داشته و میتوان از راه حل هایی که برای رفع مشکلات مشابه در گذشته برای آن استفاده شده است استفاده کرد یا حداقل میتواند سرنخی از این باشد که حل مشکل را از کجا میتوان شروع کرد. در واقع اینجا هم باز الگوریتم divide and conquer به کمک ما می آید زیرا که اگر آن مشکل را با این الگوریتم به کوچکترین اعضای آن تقسیم کنیم میتواند از یک سری از راه حل های موجود به صورتی که بالا بیان شد استفاده کرد و مشکل کلی را راحت تر حل کرد.

2. مشکل کارآمدی دانشجویان نرم افزار را من میخواهم با استفاده از الگوریتم dynamic programming توضیح بدهم.
در واقع مشکل اصلی دانشجویان نرم افزار 3 موضوع کلی: 1. نداشتن دانش کافی 2. نداشتن تجربه و مهارت کافی 3. نداشتن ارتباط قوی با شرکت های نرم افزاری است. حال اگر ما الگوریتم dynamic programming را در نظر بگیریم چیزی که این الگوریتم به ما پیشنهاد میکند این است که راه حل کلی یک مشکل در حل زیر مشکل های آن است. به این صورت که راه حل برای زیر مشکل ها در نهایت به حل آن مشکل کلی می انجامد. در مورد مسئله اول راه حل این مشکل این است که دانشجو با جستجو در اینترنت و پیدا کردن منابع خوب و کلاس های خوب دانش خود را ارتقا دهد. در واقع این مشکل به پشتکار و تلاش دانشجوی نرم افزار نیاز دارد. وقتی که دانشجو در مسیر درست و با تلاش درست افتاد میتواند به مشکل دوم یعنی نداشتن مهارت و تجربه کافی بپردازد. این مشکل هم میتوان با انجام تمرین های کلاس ها و کتاب هایی که دانشجو در مدت دانش اندوزی با آن مواجه می شود تا حدودی جبران کرد. گرچه نباید از این واقعیت گذشت که تعریف پروژه به عنوان سمپل به دانشجو کمک میکند با مسائل واقعی که در مدت توسعه نرم افزار با آن مواجه می شود آشنا شود. پس مشکل دوم هم با تلاش و کوشش قابل حل است. بعد از حل این دو مشکل دانشجو باید سعی کند که با آماده کردن یک رزومه استاندارد و جستجو در لینکدین سعی در پیدا کردن آدم هایی که مسئول استخدام شرکت های نرم افزاری هستند را پیدا کرده و رزومه خودشون رو در اختیار آن افراد قرار دهند. سپس با تجربه کردن چند مصاحبه می توانند تجربه لازم را کسب کرده و سپس به استخدام یک شرکت نرم افزاری در بیایند و پس از به دست آوردن تجربه کاری آن ها قادر خواهند بود در روند توسعه نرم افزار های مختلف به صورت بنیاد آزاد شرکت کنند. در واقع در این مشکل ما مشاهده کردیم که دانستن مسیر درست و تلاش و پشتکار دانشجو قادر است که مشکل کلی تر که شرکت نداشتن در روند توسعه نرم افزار حال چه به صورت خصوصی برای شرکت ها یا چه به صورت عمومی به صورت بنیاد آزاد فائق آمده و آن را حل کنند.

3. کلاس P

هر مسئله‌ای که بتوان در زمان چندجمله‌ای حل کرد، به کلاس P تعلق دارد. پس مرتب‌سازی سریع مسئله‌ایست در کلاس P چون زمانش حداکثر مربعی است.

کلاس NP

مسایلی که خودش شاید در زمان چندجمله‌ای حل نشود، ولی اگر یک راه‌حلش را داشته باشیم، می‌توانیم درستی آن را در زمان چندجمله‌ای وارسی کنیم، NP است. پس مسئله یافتن یک تور کمینه بین چند شهر NP است. علتش این است که اگرچه خودش در زمان نمایی حل می‌شود ولی اگر یک تور داشته باشیم می‌توانیم در زمان مثلا مربعی وارسی کنیم که درست است. پس مرتب‌سازی سریع هم علاوه بر کلاس P، در کلاس NP هم هست: هم خودش در زمان مربعی حل می‌شود، هم راه‌حلش در زمان مربعی وارسی می‌شود.

کلاس NP-Complete

می‌بینیم که دو دسته مسائل در کلاس NP هست که یکی سخت‌تر از دیگری است. دسته سخت‌تر را مسائل NP-Complete می‌گویند. یعنی پیدا کردن تور بهینه NP-Complete است ولی مرتب‌سازی سریع فقط NP است و نیز P.

کلاس Np-Hard

سپس محققان مسائل حتی خیلی سخت‌تری را هم پیدا کردند. این‌ها مسائلی هستند که نه تنها خودشان در چندجمله‌ای حل نمی‌شوند، بلکه راه‌حلشان هم شاید در چندجمله‌ای قابل وارسی نباشد. به این مسائل NP-Hard می‌گویند. مسائل NP-Complete، اِن‌پی سخت هم هستند ولی مسائلی در NP-Hard است که سخت‌تر از بقیه است و NP-Complete نیست. سختی یعنی خیلی طول می‌کشد که مسئله حل شود. راه‌حل را می‌دانیم، ولی انبوهی از حالت‌های مختلف را باید بررسی کنیم تا به جواب برسیم.

مسائل NP-Hard خانواده‌ایست از چند هزار مسئله با کاربردهای روزمره و واقعی. مثلا پیدا کردن کم‌ترین میزان سیم‌کشی و اتصالات در یک مدار الکترونیک. اما بخاطر سختی حل آن‌ها از روش‌های دیگری استفاده می‌شود. یکی از این‌ها الگوریتم‌های تقریبی هستند. دوم الگوریتم‌های ابتکاری و تصادفی هستند. همه این روش‌ها یک مسئله NP-Hard را دقیق حل نمی‌کنند، ولی مزیتشان اینست که جواب تقریبی نزدیک به دقیق را خیلی زود و تند برای ما تولید می‌کنند.