1 یاسمین آشوری — استاد شکری — فصل اول کامل

Year. ۹۹ Month. ۱۲ Day. ۱۳ — جلسه اول (سرتره) — Subject. طراحی الگوریتم

این درس ادامه درس ساختمان داده و برنامه نویسی هست. برای این که یک برنامه کامپیوتری شکل بگیره

۳ عنصر لازم داریم: اول روش اون مسئله‌ای داریم دوپیدا کنیم یعنی می‌گیم الگوریتم. بعدی ساختمان داده

هست که یعنی داده‌های ما توی چه ساختاری نگهداری و پردازش. سومی هم زبان برنامه نویسی است.

کتاب CLRS کتاب سنگینی هست و مطالب خوبی عمیق و اثبات‌های ریاضی داره. مهم‌ترین

منبع برای کنکور هم همین کتاب (CLRS) هست. (در خاطر این درس) چند چیز است اول چرا الگوریتم کامپیوتری چگونه

کارا شود بوده؟ چرا؟ در طراحی الگوریتم کامپیوتری برای بررسی ارزیابی میزان کارایی و

چگونه ارزیابی آن کارایی، در فاکتور کارایی زمان و حافظه و بحث می‌کنیم

(الگوریتم‌های کلاسیک) ایده‌های اصلی در طراحی الگوریتم — پیچیدگی محاسبات.

تو ایده تقسیم و حل میگه مسئله‌های بزرگ رو به کوچکتر بشکن. ایده ساده هست ولی بسیاری از مسائل

کلاسیک رو با کامپیوتر با این ایده حل شده‌اند. با تقسیم و حل. پس داشتن این توش‌ها بدیها

کمک می‌کنه که روش مسئله خودمون رو انتخاب کنیم و به جزئیات پیاده‌سازی وارد حل الگوریتم می‌شیم.

حالا یه سوال توی تست تعریف الگوریتم معایبش زیاده. می‌گیم الگوریتم کاملاً مستقل از زبان برنامه نویسی هست؟

نمی‌گیم نه چون موقع برنامه نویسی یعنی پیاده‌سازی ما با یک زبان این کار رو می‌کنیم. ولی از ساختمان داده

رو زبان استفاده کنیم. ولی به صورت کلی تعریف الگوریتم نمیتونه مستقل از زبان برنامه نویسی باشه.

الگوریتم باید این ۵ شرط رو برقرار کنه تا بشه بگیم الگوریتم درست.

ورودی: به دو شرط داشتن ورودی و مشخص بودن نقطه شروع

خروجی: دو شرط داشتن خروجی به عنوان نتیجه و داشتن نقطه پایان مشخص شده

Month. Day. Subject.

شرط کارایی: یک کامپیوتر دو منبع برای کامپیوتر بیشتر از یک هست (RAM) یکی دیسک. این دو تا می‌تونند به

یک اندازه استفاده کنند باید به توجه به الگوریتم یک مقداری دور ازش استفاده کرد. شرط کارایی به اندازه مسئله هم ربط داره.

میلیارد کند و ۱۰۰ عدد منا حافظه هاشون باهم فرق داره.

شرط قطعیت: روی کامپیوتر قابل اجرا و فهم باشه، کامپیوتر چند تا کار خودش می‌تونه برای شما انجام بده و توی این

دونه‌ها باشه. (ورودی پیچیده - خروجی پیچیده - محاسبات ریاضیاتی، منطقی، مقایسه‌ای - ذخیره و بازیابی در حافظه)

شرط کمیت: خانه پذیر باشه اجراش و تعداد حالتی کم بشه. فرقش با دومی اینه که اون نقطه خروجی داشته

باشه ولی همگرا نشه و بیافته توی حلقه بی‌کفایت

روش‌های بیان الگوریتم:

زبان طبیعی: مثل انگلیسی / فارسی: سخت به حالت خیلی بهتر است اما برای کسی که گرامر و فلوچارت اون

شکل رو بفهمه و متوجه بشه. شبه کد: جزئیات زبان اش رو رعایت نکنیم ولی خوانایی داشته باشه.

بهتر از حالت فارسی است شبه کد هست ولی ~~چیزی~~ تولید برنامه خوبه.

میخوام برای یک مثال ساده از الگوریتم. مرتب کردن مرتب کردن، یکی از اولین مسائل کلاسیک

دوران کامپیوتر است. در بیشتر چیزها مرتب کردن کاربرد داره. شرطش اینه که n تا عدد مقایسه

پذیر داشته باشیم و قابل مقایسه عدد باشه دیگه مثل رشته string باشه. حداقل n تا عدد.

Year. Month. Day. Subject.

ما با استفاده از متن (زبان طبیعی) و اشکال اینارو توضیح دادیم. اول متن رو توضیح دادم. در هر مرحله ...

خوبه ولی خیلی دقیق نیست ابهاماتی وجود داره. اینجوری بزرگ ترین اعداد از پایین به بالا چیده می شوند.

سوال: چرا اسمش Bubble Sort هست؟ چرا حباب؟ (من گفتم ربطش مرتب سازی انتخابی هست و

گفت در واقع فرق اصلی متدی مقایسه کردن اول با آخر در انتخابی و در حبابی دونه به دونه مقایسه میشه.)

جوابش اینه که مقایسه ها به صورت پی در پی هست یعنی یهو یا جلو می و به شکل حباب میشه.

یه سوال دیگه موقع مقایسه حبابی پیش میاد. چرا توی مرحله اول چند تا مقایسه بیهوده انجام میشه مثل ۲ و ۳؟

چون کامپیوتر تا اونارو مقایسه نکنه باهم که درکی نداره از اون اعداد. ما دیدیم که بزرگه یا کوچیک ولی

کامپیوتر که نمیدونه. هدف بشه که برنامه نویس بیشتر از الگوریتم هست و کلیتی

تو مقالات اصلی ترین روش طراحی هست که بدونید

(جلسه دوم)

جلسه قبل گفتیم تو مرتب سازی حبابی اگه مثلا ۶ آیتم باشه تو مرحله اول که مثلا بزرگ ترین عدد ره

جایگاه آخر قرار بگیره ۵ تا مقایسه انجام میدادیم. نفر اول با دوم و ... مرحله بعد اون عددی که جایگاهشو پیدا کرده و داریم

کنار بقیه رو مقایسه میکنیم. فرمول های محاسبه تعداد مراحل و مقایسه و تعداد کل مقایسه نوشته شده هست. اگر یک کرایه ها

داشته باشیم چند تا مقایسه حبابی انجام خواهیم داد؟ ۴۹۵۰. اگر فرض کنیم کامپیوتر هر ثانیه یک میلیون مقایسه رو در ثانیه

انجام میده این مسئله رو چند ثانیه طول میکشه حل کنه؟

ثانیه	میلیون ۱
۰/۰۰۴۹۵	۴۹۵۰

Day. Subject.

پس تیم پیچیدگی زمانی این الگوریتم ← $T(n) = \frac{n(n-1)}{2}$ که از مرتبه n^2 هستند.

تعداد تعویض‌ها در مرحله i ام و تعداد تعویض کل چقدر است؟ بستگی به این که چه تعداد وروودی‌هایی باشند مسئله

و جواب ثابت نداره و بیشترین و بدترین حالت داریم. بهترین حالت یعنی تعویض اتفاق نیافته.

حالا آرایه‌ی من چطور باید باشد که تعداد کل تعویض‌ها هم صفر بشه و یعنی به تعویض نشه؟ آرایه مرتب.

حالا میشه آرایه‌ای داشته باشیم که تمام مقایسه‌های ما مجزیه تعویض بشه؟ اگر آرایه برعکس مرتب باشه پس همیشه مقایسه‌ها

منجر به تعویض میشه. و عمل اصلی توی این الگوریتم مقایسه هست و تعویض نیست چون مثال‌هایی هست که کلی مقایسه

داریم ولی تعداد کمی تعویض صورت میگیره. توی الگوریتم‌ها خیلی پیش میاد که یه الگوریتم پایه معرفی میشه و یه سری

تغییرات داده میشه و الگوریتم‌های دیگه‌ای ساخته میشه و حتی میشه مثلا الگوریتم جایی رو بهتر کرد. البته مرتبه

اش تغییر نمی‌کنه فقط بهبود داره. با توجه به متن این تغییر داده اگه هم کلیه از مقایسه‌ها مجزیه تعویض نشه چه

نتیجه‌ای میگیریم؟ آرایه مرتب بوده. پس توی اتفاق این مرحله تعداد تعویض اگه صفر بود چه نیازی هست مرحله بیشتر

رفت تو آرایه مرتب هست پس همین کار ساده هم باعث حذف جمعی زمان میشه. حالا باید بیایم الگوریتم رو

تحلیل کنیم. تو چه حالت‌هایی سریع‌تر جواب میده تو چه حالت‌هایی طول میکشه؟

در اینجا فرض تعداد کل مقایسه‌ها یکسری کنید چون ممکن است آرایه‌ها عدد اول مرتب باشند و تغییری صورت نگیرد پس

تعداد کل مقایسه‌ها در بهترین حالت $n-1$ و تو بدترین حالت مثل قبل هست یعنی همه مراحل رو رفت.

SHAHBAZ

Year. Month. Day. Subject.

پس الگوریتم تمرینه چون تا هر جایی که آرایه مرتب شد مراحل بعدی انجام داده نمیشه. شرط عدد کلی مقایسه ها هم سوال

خوبی هست. باید همه حالت هایی که میتونه اتفاق بیافته رو لیست بندی کنیم و مقایسه کنیم. مسئله که حالت تحقیق داره این

ماده طلب میباشد و آخر هم فعل تحویل بدیم. shaker گونه ای هست که بر پایه Bubble شکل گرفته.

اصلی ترین روش همین است و ادامه به شکل شبه کد. کتاب از شبه کد زبان C استفاده کرده. از دیتا تایپ های

زبان C استفاده نکرده بلکه از این نوع استفاده کرده. keyty ← خاصیت این قابل مقایسه بودن ترتیبی

index ← شماره متغیر به حافظه ای استفاده میشه number ← متغیر از نوع عدد bool ← از نوع کره و نه

حدود بعدی در زبان C آرایه ها از 0 شروع میشن ولی اینجا اجازه میده به هر اندیسی تعریف بشه. محدوده اول

متغیر باشه مثل n. در حالی که برخلاف ساختار داده هر که طول مشخص داره

تغییر دوم از دستور repeat end استفاده میکنه و ما C همچین دستوری نداریم. معمولا اگر بخواهیم آرایه درون

یک دوال برگردونده ... که می تونند برگردانده شه و یا شه اگر بدنه از const استفاده میشه.

ما می دونیم تو زبان C هر تکه کدی یه برنامه اصلی هم برگردانده میشه.

مثالی هست که هم شبه کد و تمرین و کدنویسی این الگوریتم. اسم تابع seq se هست. و بیشتر undli چون

نباید spa باشه. کتاب برای هر الگوریتم یه تابع نوشته. و پارامترهای int n که میشه خود na هم نوشت.

یک آرایه از نوع S داریم که Keyty هست و چرا از نوع cons؟ چون می خوام تغییراتی که روی S داده شده برگردانده شه.

CS CamScanner

موضوع .....

Day. Subject.

n عنصری هست که دنبالش هستیم در آرایه S و باید هم نوع با S باشد از نوع keytyp

و ind نشان دهنده اندیس یک مکان.

تو loop می‌خوام چی رو پیدا کنم؟ اگه بزرگتر یعنی پیدا نکردم یا -1 بستگی به این داره که آرایه از ۰ یا ۱ باشه شروع شه

و اگه n رو پیدا کردم اندیس اون جایی که n رو توش پیدا کردم میذارم خروجی که اندیس از اون نوع باشه و زمانی که کوچکتر مساوی n و

کمتر از توکال - 10 تو آرایه S هستیم مازاد n هست که دنبالشیم، اونو پیدا کنه زمانی که شرط حلقه نقض شه یعنی یا اندیس به آخر برسه یا پیدا کنیم n

خب الگوریتم تعویض هم خیلی بیشتر هست به جایی، اونجا مقایسه و تعویض به آخر بود اینجا کوچکترین عنصر رو

ابتدا و برای بقیه اونو میذاریم کنار و همین مرحله دوباره. هر دو هم از کوچک به بزرگ مرتب میشه.

پرمنی هست الگوریتم تعویضی

تفاوت هم مقایسه‌ها نیست بلکه همه رو با یه عنصر مقایسه نیست اول با دومی اول با سوم ... (مرحله ۱ شکل ۳ اصلاح)

تعداد مقایسه‌های تعویضی بیشتر است؟ مساله پایین هم هست. در این شکل برای n تا عنصر میشه n-1

در جایی شه n-1. تو مرحله اول چند تا مقایسه اگه n تا عنصر؟ n-1 در مرحله دوم n-2 مرحله سوم n-3

و مرحله آخر 1 پس تعداد کل مقایسه‌ها میشه $(n-1)+(n-2)+(n-3)\dots+1 = \frac{n(n-1)}{2}$

کدوم الگوریتم برای مسئله ما بهتره؟ خب بستگی به هر مسئله تفاوت هست اما به صورت کلی اون الگوریتمی که دلرای پیچیدگی زمانی کمتری هست.

در جستجوی ترتیبی اگه n تا عنصر باشه چقدر مقایسه؟ در حالت بدترین $T(n)=n$ و بهترین $T(n)=1$

در جستجوی ترتیبی هم در بدترین حالت فرمولش این بوده $\lceil \log_2 n + 1 \rceil$ ← اگه توی شکل هم نباشه

SHAHBAZ

ولی شرط استفاده از الگوریتم جستجوی دودویی این هست که آرایه مرتب باشه.

مثلاً هم است و کنکوری درصورت مفهوم تابع پیچیدگی زمانی. مفهوم $T(n)$ = پیچیدگی یعنی تعداد تا کدام عمل اصلی

به ازای ورودی n ساده‌ترین و خوبی ترش میشه $T(n) = \log_2 n + 1$ یعنی مثلاً روی آرایه‌ای

برای اندازه n حل کنی نیاز به $\log_2 n + 1$ تا عمل اصلی داریم. مسئله مابهست آدرس n امین عدد فیبوناچی

هست. برای فیبو به صورت بازگشتی تعریف میشه اول. بعد ۱ عدد بعدی جمع دو تا قبلی هست. $f(n-1)+f(n-2)$

اگه بخام n امین عدد فیبوناچی رو بدم بگه ۱۳ مثلاً موقع الگوریتم نوشتم که حقت اونا این کار رو انجام میدن. توی

توسط یه تابع بازگشتی انجام میشه (تابعی که داخل خودش، خودش رو صدا میزنه برای یه تابع زیاد)

اینجا مثلاً اگه $n=0$ باشه این عدد فیبو میشه. اگه ۱ باشه میشه ۱ تا اولین اندیس از ۰ شروع میشه.

الگوریتم خودش بازگشتی بود. تو الگوریتم‌های بازگشتی محاسبه $T(n)$ سوراخ سخت و خودش بخش به چند قسمت

بازگشتی میشه برخلاف الگوریتم‌های غیربازگشتی که فقط تابع پیچیدگی هم بود اما اینجا خودش

خودش هم بازگشتی است و عمل اصلی هم نه for و نه چیزی یه عمل جمع پس تابع پیچیدگی زمانی

رو باید به اصل فهم مسئله بازیابی تا ساده بشه. اینجا گفتیم بود که رو در حالت

$fib\ n-1$ و $T(n-1)$ و ــــــ و در نهایت یه عمل جمع

$T(n)$ خودش براساس $T(n)$ های قبلی بدست میاد این نوع اش تقسیم و حل هست که بعداً

به کوچک تقسیم میشه . اول مانند کوچک n-1 و بعد n-2 و

از ایده برنامه نویسی پویا استفاده کرده از ایده های اصلی طراحی الگوریتم ها هست استفاده کرده و الگوریتم پایین کیفیت بازگشتی هست

روش بهتر هست و هر جایی که الگوریتم بازگشتی داریم از الگوریتم غیر بازگشتی استفاده کنیم .

CamScanner

date :

سوم تابع پیچیدگی زمانی رو بگیم. وقتی میگیم $T(n)$ برابر با یه چیزی مثل مثلا $\log_2^{n+1}$

یعنی تعداد عمل اصلی به ازای ورودی n ساده است یا ...

حالا متوجه ترسش. این بدست یعنی اگه مسئله رو بخوایم برای اندازه n حل کنیم نیاز به $\log_2^{n+1}$ عمل اصلی داریم.

این خوبش بود. چقدر الگوریتم برای حل یک مسئله به دست آوردن n امین عدد فیبوناچی

n امین عدد فیبوناچی چی به صورت بازگشتی تعریف میشه روی فیبوناچی.

و جمع قبلی ها میشه. حالا مسئله اینه یه عدد بگیره مثل n و n امین عدد فیبوناچی رو برگردونه. مثلا اگه بگیره ۸

باید ۸ امین عدد فیبوناچی رو برگردونه مثلا جواب بده ۱۳، اگه بگیم ۹، ۲۳ و ... ۲۳ و ۱۳ و ۸ و ۵ و ۳ و ۲ و ۱ و ۱ و ۰

حالا چقدر الگوریتم ها خوب کار میکنن. اولی توسط یه تابع بازگشتی نوشته شده که داخلش خودشون، خودشون

رو صدا میزنن برای یه تابع دیگه.

اگه n کوچکتر مساوی n باشه n رو برگردون. اگه ۰ باشه ۰ امین عدد فیبوناچی میشه ۰ فکر کنم اندیس از

۰ شروع میشه. این عدد ۱ دومین عدد ۱ هم بعدی. اگه $n \leq 1$ یعنی اگه n یا ۰ یا

۱ باشه پس ۰ امین عدد ۰ و ۱ امین میشه ۱ کلیه خودش رو اجرا کنم در غیر این صورت باید فیبوناچی

هر تا عدد قبلی رو بدست بیاره و با هم جمع کنه یعنی خط این تابع رو برای $n-1$ و $n-2$ صدا بزنه

تو کدش یکی هم تابع fib1 رو صدا میزنه برای $n-2$ تا برای $n-1$ عدد فیبوناچی رو بدست بیاره.

یعنی اگه fib1 رو برای ۱۰ صدا بزنیم داخل خودش fib1 رو برای ۸ و ۹ هم صدا میزنه داخل خودش

۷ هم ۶ و ۷ رو صدا میزنه و به همین ترتیب تا خر الگوی تقسیمش اگه درخت اش رو بکشید.

maxnote

CamScanner

حالا در مورد تابع بازگشتی این الگوریتم، مشخصاً با هم مسأله حل میشه. برای این که مسئله در حالت n حل بشه چی میخوایم

یکی اول برای n-1 حل میشه بعد n-2 و بعد جواب رو جمع میکنه. حال مفهوم Tn، تعداد عمل اصلی برای حل

مسئله توی حالت n. خب برای fib1 برای n چند تا عمل جمع می‌خوام؟ به ظاهر فقط 1 جمع می‌خوام که همین

1 تو فرموله. اما واقعاً اول باید برم n-1 و n-2 و حالا خودتون لیاقتم هم اتفاق میافته و یه جمع سر آخر زدنم

اگه T(n) تعداد جمع‌ها برای حالت fibn برای fibn-1 چند تا می‌خواد؟ طبق همین تعریف Tn-1

و برای حل مسئله

بعد مسئله رو برای n-2 حل کنم چند تا عمل اصلی می‌خواد T(n-2) بلکه به صورت

برای الگوریتم اول که بازگشتی هست، تابع زنجیره‌ای صریح نیست بلکه فرمولش خودش یه تابع بازگشتی هست.

تو مسئله‌های قبلی T(n) یه تعداد صریح نداشت ولی اینجا Tn به صورت صریح قابل محاسبه نیست بلکه Tn

برابر خواهد بود با این فرمول. اینجوری مفهوم Tn می‌فهمید؟ یعنی اینکه مسئله رو اگه بخوام تو حالت n حل کنم

اول باید برم مسئله رو تو حالت n-1 و n-2 حل کنم و یه جمع هم نیاز دارم که جمع این نتایج

هست که برای Tn-1 و Tn-2 به دست اومده تا جواب آخر به دست بیاد.

تو الگوریتم اول هم به همین ...

توی این مثال مفهوم تابع زمانی رو و فواید حدود تابعی رو توضیح میدم. برای یه مسئله مشترک دو تا الگوریتم نوشته و توابع زمانی اش رو مقایسه کردم. الگوریتم اول یک الگوریتم بازگشتی هست که اگرچه نوشتن شون خیلی ساده هست ولی یه مشکلی دارن و اون زمان طولانی اجراشون هست و این به خاطر فراخوانی overhead هست و فراخوانی های متعدد و در نتیجه تابع زمانی پرزحمتی دارند پس به الگوریتمی که یا تابع غیربازگشتی حل میشه.

یک حد سهم بیلد رو هم گفتم یا به تعریف مفهوم تابع زمانی، مهمه برای الگوریتم هایی که بازگشتی نوشته شدن مثل اولی، فرمولش میره و مشخص میشه. و به صورت یک فرمول برحسب n بدست میاد و فرمولش به یه فرمول بازگشتی میرسه. الان بگم T10 چند میشه نمیتونید بگید یعنی اول T9 بعد T8 بعد T9 رو چطور بدست بیارم؟ فرمولش دوباره... توی الگوریتم اول هم که خلاصه پیغام بگم اینطور میشه که اگه n بود کوچکتر مساوی 1 خود n میشه. چون fib0 و 1 میشه خودشون و در غیر این صورت هم از اون فرمول استفاده میشه. و مزیت اش این هست که خیلی راحت نوشته شده و خیلی خطا نمیشه. و عمل اصلی هم جمع هست ولی طفرا با دقت بیشتر بدحساب کردن fibn-1 فرمول دقیقش چی داره...

ما خیلی جاها چون الگوریتم هاشون بازگشتی هست پس فرمولش هم بازگشتی هست. الان می تونیم بگیم n=100 چند میشه چون فرمول بدست میاد و براساس آدمای قبلی بدست میاد. fib2 همون مسئله رو حل کرده ولی به صورت غیربازگشتی. گفتم من یه آرایه میگیرم از عددهای کوچک شروع اول fib0، 1، 2، 3 و... حساب می کنم هر بار باید حساب توجه میذارم که روی عددهای قبلی دیگه محاسبه نکنم چون قبلا محاسبه اش کردم. یه آرایه f تعریف کرده f[0] = 0 و f[1] = 1 و گفتم i = 2...n برو تو آرایه زمین عدد توی آرایه f بزار i-1+i-2

میخاریم یکی تو مکان ۰ توی آرایه f = ۰ توی مکان ۱ هم ۱ و توی حلقه از 2 تا n پر کنه. مکان قبلی رو میذاریم

جمع دو تا مکان قبلی. باید بیرون برای n، جمع دو تای قبلی رو میریزه تو 3 تا برسه به n و تو n میرسه

محاسبه شده. قبلی خواسته Top down حل کنه اگه مثلا n=10 بوده گفته بود اول f8 و داخلش خودش

ولی روش دوم bottom up هست اول f0 و f1 و ... هر بار هم بعد محاسبه نتیجه اش رو تو

آرایه میذاره و میتا چون مثلا f7 رو محاسبه کردم میدم از تو آرایه برمیدارمش. الگوریتم اول بازگشتی دوم غیربازگشتی

و یه راه دوم هم بالا به پایین و بالعکس هست. یه منظر هم دیگه. پس الگوریتم بالا از نوع Devide & Conquer

یعنی تقسیم و حل یعنی حل خونه های بزرگتر به کمک حل خونه های کوچیکتر. و با ترکیب خونه های کوچیک

تر به جواب خونه های بزرگ رو بدست بیاریم. ولی الگوریتم دوم از نوع برنامه نویسی پویا هست، اول خونه های کوچیکتر رو

حل کن، شروع کن تا خودت رو به خونه های بزرگتر برسونی. اگر قبلی هارو نیاز داشت جوابش رو از

توی آرایه برمیداره.

خب این الگوریتم کلا چند بار اجرا شده. جمع. به اندازه ی اجرای این for ها. از ۲ رفته

تا n چند بار اجرا شده؟ n-1 (یعنی ما گفتیم از جمله دوم n-2 دوباره اگه بگیم از 1 تا n میشه n بار پس این میشه n-1)

خود ۲ هم اجرا میشه دیگه یه بار i = ۳ و ... پس فرمول کلی میشه عدد اولی - عدد آخری + 1 پس میشه n-1

و این فرمول اصلاح شود T(n) = n-1 یعنی اگه بخوای 70 عدد فیبوناچی رو محاسبه کنیم چند تا عمل جمع

نیاز داریم ۹. حالا کدوم T(n) بهتر هست؟ من تا فرمول اولی رو به یه فرمول دیگه تبدیل نکنم نمی تونم مقایسه

شون کنم مثلا اگه حل کردم شد 2n. این نکته اساسی هست اینکه حل معادلات بازگشتی، کریک مسئله

subject: date:

ریاضیاتی هست روی بش نیاز داریم. و کتاب نیولیان در آخر کتاب ریاضیات مورد نیاز را آورده است.

پس باید بتونم اینو تبدیل کنم حالا پایین یه استدلالی کردم، خرخول صرح نه ولی یه نامساوی درست آورده براش.

خب اگه ۱ رو برداریم از اینجا سمت راست ها کوچیک تر میشه و به جای $T(n-1)$ گذاشتیم $T(n-2)$ و باعث شد

سمت راست کوچیک تر شده. پس $T(n) > 2T(n-2)$ و با استفاده از یک نامساوی $T(n-4)$ بزرگتر هست از دوتا T_{n-6}

و تو سمت راست دوتا دوتا کم میشه یه ازاش توان ۲ اضافه میشه. اگه n^2 بار این کار و انجام بدیم n^2 بار بوجود

میاد و میشه $2^{n/2}$ پس مبنایی برای مقایسه بوجود آوردیم. برای تایم اول این رو دوی $T(n)=n-1$

حالا کدوم بزرگتر هست؟ اولی میشه 2^{50} و دومی میشه 99 پس اولی بزرگتر هست.

تازه $T(n)$ الگوریتم اول خیلی بزرگتر از دوم هست چون طبق نامساوی گفتیم $T(n) > 2^{n/2}$

و پارامترش میتونه n هم نباشه و عدد باشه و بستگی به اندازه ورودی داره مثلا اگه عددی برای آرایه باشه اما اگه ماتریس

باشه نمیشه چون ماتریس به تعدادی سطر و ستون داره و مینویسیم $T(n,m)$ و فقط $T(n)$ دیگه نیست

گراف هم به بزرگی یه گراف دوتا پارامتر رأس داره و یال ها متغیر اش میکنه.

اگه یه گراف وزن دار باشه ۳ تا پارامتر. بعضی موقع ها تعداد عمل اصلی به خروجی ورودی بستگی نداره و ثابت

هست ولی بعضی وقت ها به تعداد داده ها بستگی داره و اینکه چه مدل ورودی باشه روی تعداد عمل اصلی تأثیر داره و

در این صورت می تونیم یه خرخول صرح بگیم و یا متوسط و بد و خوب سؤال ببین T, W, A

و یه خود $T(n)$ ولی بعضی جاها هم مینویسن $T(n)$ در بدترین حالت و ...

خب بریم سراغ مثال ها گفتم جمع عنصر آرایه. عمل اصلی جمع هست، اندازه ورودی چون یه پارامتر داره n هست و برای

مثال های پیچیدگی زمانی

date :

چند تا جمع نیاز داریم؟ یه for می‌ذاریم از عدد اول تا آخر $i=1$ تا n و هر بار یک دونه از عددها رو جمع می‌زنیم.

از G استفاده کنیم میشه for $i=0$ تا کمتر از $n-1$ و هر بار $I_i + Sum$ Sum و n تا عمل جمع داریم.

توی مثال دوم هم مرتبه‌ی الگوریتم جلوی n^2 هست. حالا اگر ضرب دو ماتریس مربعی باشه چند تا پارامتر می‌خوام؟

شاید فکر کنید هر ماتریس دوتا و مربعی هست پس میشه ۴ تا اما برای هر ماتریس یکی کافیه. چون ماتریس مربعی

سطر و ستون برابره و هر ماتریس یکی کافیه هر چند تا سطر همون قدر ستون. حالا اگر دو تا ماتریس مربعی

ضرب بشه باید تعداد سطر و ستون ها برابر باشه پس مثل ماتریس اول میشه پس برای ضرب دو تا

ماتریس مربعی $n \times n$ فقط یه دونه پارامتر n کافیه. عمل اصلی رو هم میتونیم عمل ضرب اعداد در نظر بگیریم

و حساب کنیم n^3 ضرب نیاز داریم پس میشه تابع زمانی n^3. چرا n^3 ضرب عددی؟ اگر یه ماتریس

$n \times n$ باشه ماتریس دوم که مثل اول هست و یه پارامتر بیشتر نمی‌خواد. دو تا ماتریس $n \times n$ ضرب بشن

حاصل یه ماتریس n^2 بوده. یادمون هست اگه مربعی باشه و هر دو $n \times n$ ماتریس G که حاصل ضرب

این دو تا شه اون هم $n \times n$ خواهد بود. ماتریس $n \times n$ چند تا عضو داره؟ n^2

مثل جدول 10×10 چند تا عدد توش بود؟ 100. حالا ماتریس $n \times n$ چند تا عضو داره؟ n^2 و هر عضو توی ماتریس

حاصل چطوری بدست میاد؟ یه سطر از ماتریس اول با سطر از ماتریس دوم هر کدوم n تا پس هر عضو نیاز به ضرب یک

سطر و یه ستون داره یعنی نیاز به n تا ضرب داره. پس کلا n^2 عضو $\times$ n برای محاسبه هر عضو میشه n^3

مثال بعدی ضرب دو ماتریس به صورت کلی، یعنی دیگه مربعی نیست چند تا پارامتر میخواد؟ میدونم چند تا بخوای باشه

چند میشه؟ دو تا ماتریس داریم هر کدوم هم سطر و ستون. بدون دقت می‌کنیم اما اینکه تو ضرب ماتریس تعداد ستون اول

subject :

date :

تعداد سطر دوم برابر باشد یعنی p و Q ، np هست . پس ۳ تا پارامتر می‌خوایم . چون دو تا ماتریس دو بعدی بود

هر کدوم ۴ تا اما چون تعداد سطرهای A با ستون‌های B برابره پس ۳ تا کافیه . و ماتریس حاصل ابعادش qn

هست n سطر Q ستون . از لحاظ اندازه ورودی ۳ تا کافیه پس تابع زمانی اش هم میشه این . عمل اصلی

ضرب عددی هست تعداد ضرب‌ها . حالا تابع زمانی باید تعداد این ضرب‌ها رو بشمارع . و تعدادش

درمیاد npq . استدلالش مثل قبل هست اما اونجا خاص بود حالتش اینجا به طور کلی . ماتریس حاصل که ابعادش

nq گفتیم چند تا عنصر یا درایه داره ؟ nq عنصر داره . برای محاسبه هر کدوم باید یه سطر از A × ستون از B

چند تا عنصر ؟ p تا عنصر دو تا هم در ضرب میشن . nq تا عنصر داریم برای هر عنصر هم سطر × ستون یعنی p

تا ضرب عددی نیاز داریم یعنی $n \times q \times p$ نیاز داریم که میشه خودش npq مثل مثال بالا و اون رو هم

پوشش میده .

مرتبه اللگوریتم : تعریف تقریبی هست نه دقیق و ریاضیاتی . و اینجا می‌خواد به طور ریاضیاتی مرتبه اللگوریتم رو تعریف

کنه . اولین نماد O بزرگ هست برای بیان حد بالای مرتبه هست . یعنی مرتبه اللگوریتم لااقل کوچک‌تر هست از

چیزی که توی داخل نماد O مشخص هست . به صورت می‌گوییم اگر بگیم $T(n)$ عضو بزرگ $F(n)$ هست به این صورت‌ها نوشته

میشه $T(n)$ از مرتبه بزرگ fn هست . اگه gn عضو $O(fn)$ باشه میگیم رشد تابع gn کوچکتر یا مساوی

gn هست یا به زبان دیگه fn حد بالای gn هست . از لحاظ ریاضیاتی $g(n)$ حد بالاش $f(n)$ هست وقتی

یه n پیدا کنیم یه ضریب پیدا کنیم که gn همیشه کوچکتر از حاصل ضرب c در fn باشه . یعنی وقتی گفتیم

رشد تابع gn کوچکتر مساوی fn هست که یه ضریب c و یه عدد مثل n_0 بزرگ باشه که برای

maxnote

date:

همه اعداد جدول به طور دلخواه.

اصلاح جزوه $h(n)$. اصلاح جزوه ... در n^2 باشد

تابع امگا، حد پایین، برعکس اوها تابع هست. اگه یه c و N پیدا شود که تابع g(n) بیشتر بالای f(n) رشد تابع g(n) و پس

بیشتره. حد تابع

یه مثال یا سوال میگم. اگه

موجه دلت باشه توی هر فرض مساوی داره پس اگه مساوی باشه هم میتونم احتمالا رو پایین بنویسم هم O(n)

پس اگه g(n) ≥ f(n) باشه می‌تونیم بگیم g(n) ∈ Ω f(n)

آیا میتونم بنویسم n^2+10n حد پایین اش n^2 هست؟ بیم یه c پیدا کردم که در f n ضرب کردم زیر

g n قرار میگیره. پس می‌تونم بگم. از لحاظ تعریف هم g(n) ≥ n^2 هست که مساوی هست.

آیا میتونم بگم حد پایین اش n^2 Ω هست؟ f(n) = n^2 ، g(n) = n^2+10n

بله میتونم بگم Ω n هست، پس هم n هست هم log n هم خیلی چیزهای دیگه. کدوم جواب مناسب تره؟ هم

امگا n^2 ، n و ... ولی دقیق تر n^2 هست.

هم تونستیم ببریم بالای تابع n^2 با ضریب مثل 1 هم پایین تابع n^2 با ضریب مثل 2. پس نزدیک طبق تعریف

نمیتونه g n حد پایین اش n^2 هست نزدیک هم n و حد بالاش هم n^2. و میرسوند تا رو به تابعی به نام

تتا دقیق، که توش فاکتور تتا و ... بهاش کار کردیم، می‌گفتیم کدوم جواب درست تره. پس میگن تتا

حد دقیق ره. به معنی که هم حد بالا هم حد پایین اش n^2 باشه. پس حد دقیق اش n^2 و هم چیز

دیگه‌ای نمیتونه باشه. بله n^2+10n میتونم بگم حد بالاش n^3 ، میتونم بگم حد پایین اش n هست ولی نمیتونم دیگه بگم حد بالاش هم n

فقط n^2 هست که میشه حد دقیق اش. که مثل قبل پیدا میکردم با نصف ضریب بزرگترین جمله.

یعنی g و f هر دو هم رشد هستن. جفت شون درجه ۲ هستن اگه بیوفتم باید ضریبا با هاش ببریمشون

چون تابع زمانی، چند تعداد تکرار عمل اصلی هست. پس میتونه تابع مثلثاتی باشد پس یا چند جمله‌ای میبینم یا ادیکالی

یا لگاریتمی یا نمایی. پس میتونه تابع مثلثاتی بدیا باشد.

سوال مهم: برای نماد O میگم ولی جلوش برای Ω و Θ هم صدق هست. اسمش گذاشتیم تابع O. ولی این یه اشیاء

مصطلح هست. O از لحاظ ریاضیاتی جنسش چیه؟ تابع نیست قطعاً. عدد؟ مجموعه هست؟ چند تا عدد؟

توی نمادها نشون میده که جنسش چیه. مجموعه هست. چون g(n) عضو O(n) هست پس اینورش مجموعه

هست که باید این عمل کند. حالا چرا O n یه مجموعه هست؟ $O n^2$ و $O n^3$ و ... ؟

$O n^2$ چه مجموعه‌ای هست؟ همه توابعی که حد بالاشون n^2 هست. پس $5n^2+6n$ اینم O n هست.

5n لوغم حد بالاش n^2 هست. هر توابعی که رشدشون n^2 یا کوچکتر از n^2 هستن عضو مجموعه $O n^2$ هستن.

که میتونه حد بالاشون n^2

جنس شون مجموعه‌ای از توابع هست که رشدشون برای $O n^2$ یا n^2 یا کمتر از n

Ωn^2 یعنی شامل همه توابعی که n^2 و بالاتر از n^2 هست. Θn^2 شامل مجموعه تمام توابعی که حد دقیقشون

n^2 باشه. Ωn^3 نیست چون حد دقیقش n^2 نیست. پس Θ زیر مجموعه O و Ω توی اون تابع

هست. $O n^2$ زیر مجموعه Ωn^2 و Θn^2 هست. فقط شامل اونایی که حد دقیقشون n^2 هست ولی هر دو توابعی

هم درخوشون میاد. $O(n^2)$ یا $\Theta(n^2)$.

مثلا تابعی مثل 5n+10 آیا حد بالاش n^2 میتونه باشد ولی حد دقیقش n^2 هست؟ خیره که حد دقیق با حرف

ضریب دیگه میاد. یعنی هم حد بالاش هم حد پایینش یکی باشد. یعنی هر تابعی که حد دقیقش n^2 باشد حد بالاش

subject:

date:

هم عضو $O n^2$ هست. شما برعکسش رو فکر کردید. $\Theta(n \ln n) = O(n \ln n)$ این اشتراک مشکل هست.

و باز تصدیق می‌کند زیر مجموعه بودن را. یعنی شما این دو تا رو باکشید به جا اشتراک دارن که شامل توابعی مثل Θ هست.

چرا Ω کوچیکه و O بزرگ دلالت می‌کند؟ دلیل حضور دارد:

می‌خواهیم که یک تابع در دقت زیاد از محدوده‌ای باشد؟ چه شرایطی مناسب است؟

با توجه به الگوریتم‌هایی که تابع پیچیدگی زمانی رو در بدترین و بهترین حالت بدست می‌آوریم پس در واقع اول تابع‌ها

حد مجانبی بالا و پایین O و Ω بودن؟ که حد دقیق Θ برایشان وجود ندارد؟

یک چند جمله‌ای درجه ۲ هست مرتبه زمانی تو مرتبه n^2 قرار می‌گیرد و جملات ناچیزی ندارد.

۱۸:۲۵

هر تابعی که عضو Θ n^2 باشد عضو $O n^2$ هم هست ولی برعکسش نیست

Θn^2 زیر مجموعه $O(n^2)$ هست.

date :

الگوریتم به رشد تابع و چونکی رشد تابع توجه میکند به جای اینکه به مقدار دقیقش توجه کند چون ممکن که ضابط متفاوت باشند ولی

یک دسته و مرتبه رشدی قرار بگیرند، و بدریم همه توابعی که توی جمله‌ی دهه ۲ هستن مرتبه اشون n^2 و مرتبه دیگر هستن و

علامت و اعداد ثابت نقشی ندارند (دیگر). سه تا تعریف حد بالا، پایین و دقیق رو مطرح کردیم. مثلا $O(n^2)$ شامل

همه توابعی میشه که حد تابع بالایی شون n^2 هست و مرتبه دقیق اونا n^2 یا پایین‌تر از n^2 هست و یا دقیق ضریب

بزرگترین ضریب یا n^2 هست یا کوچکتر از n^2 هست و به طور موجود همگاه میشه همه توابعی که مرتبه دقیق اونا n^2 یا

بالاتر از n^2 هست و مرتبه دقیق که با Θ نشون میدیم همه توابعی که مرتبه اشون برابر با n^2 هست. و Ω و ... همه شامل

مجموعه‌ای از توابع میشن که توی اون تعریف گنجونده میشن. ولی در کل گفته تابع اون متحقق میشه به صورت دقیق یا دقت

ضریب بدست بیاد و زمان‌هایی که می‌شه برای T_n مرتبه دقیق به دست آمد اون قسمت از غاده‌های سالووه استفاده می‌کنیم.

سوال من: چرا ما وقتی برای یک تابع بهترین و بدترین حالت بدست میاریم اول خودم به نوعی مرتبه حد پایین و بالا رو

بیان می‌کنم. ولی حد دقیق برایش وجود داره اما توی اون اول نمی‌هستی که گفتیم میخواستیم حد بالا یا پایین رو بگیریم

که الگوریتم چطور کار میکنه مثلا به این صورت هست که الگوریتم‌ها توی حالت بدترین شرایط رو چون به دقت ضابط طبعا

بدست میاد اونو به عنوان مرتبه دقیق بیان می‌کنن مثلا جستجوی خطی توی حالت متوسط هم $\log n$ بدست میاد

و در کل دقیق موقع‌ها، ما وقتی که یک الگوریتم رو به بدترین و بهترین (از ارزش کمتر برخوردار هست) دارم تحلیل می‌کنم یا

به یک رفتار میرسیم که میتونیم بدترین رو بگیریم یا به مرزهای دقیق نداریم پس میاریم تقریب بزنیم.

ولی اگه توجه کردم از حالت بد و بهتر و ... اگه خواستم دقیق بدست بیارم میتونیم مرتبه دقیق رو استفاده

کنیم و مثلا بگیریم توی بهترین حالت از مرتبه n^3 هست یا متوسط n^2 هست این سالووه برای بدترین حالت سالووه نیست

subject :

بلکه تخمینی برای یه تابع هست که اصلا میتونه پیچیدگی زمانی نباشه. مثلا تو بدترین حالت کلیل برای یه الگوریتم

نتونستیم یه فرمول دقیق براش بدست بیاریم و چیزا عمل مکرر میکنه ولی نشد یه تقریب براش بدست بیاریم.

و در کل بدترین حالت الگوریتم برای ما اهمیت ویژه‌ای داره چون برای محاسبه ریسورس ها و Qn و تحلیل حافظه

پس برای ما مهم هست حالا توی تحلیل بدترین حالت یا فرمول صریح بدست میاد یا باید از حد تخمینی استفاده کنیم مثلا O

و اینطوری نیست که همیشه تو بدترین حالت O باشه میتونه θ باشه فقط باید فرمول صریحش رو بدست بیاریم.

حالا توی خیلی از کتاب ها می‌خوان مرتبه الگوریتم رو بگن از O استفاده کنن همینه که * گفتم یه اشتباه مصطلح

هست و پذیرفته شده. یعنی با اینکه فرمول تابع زمانی رو دقیق بدست میارن O میذارن به جای θ. و همونطور که

گفتیم اگه تابعی باشه حد بالاش n^3 باشه حد بالاش n^3+n^2 و $\cos n^3+n^2$ هست ولی دقیق ترینش همون n^2 هست

مثلا میگن $O(n^2)$ هست مرتبه دقیق تابع مزبور همون θ هست. **اتمه درس جدید:**

خاصیت اول پیچیدگی اگه یه چند جمله باشه به اسم $g(n)$ اینا میگه اول تابع یا دقیق بزرگترین جزئیش هم حد دقیقش

هست، هم بالا، هم پایین. اگه دقیق رو بدونیم میتونیم حد بالا و پایین اش رو هم بگیم که فرقش چیه و

و زمانی حد دقیق یه تابع مشخص نیست که هم حد بالاش اون باشه هم پایینش. که اینا رو آورده

ازش بذاریم.

مورد بعدی: اگه $g(n)$ حد بالاش $f(n)$ باشه یعنی حتما O باشه میتونیم بگیم $g(n)$ حد $f(n)$ هست.

اگه $g(n)$ ریشش کوچکتر مساوی $f(n)$ باشه میتونیم بگیم $f(n)$ ریشش بزرگتر مساوی $g(n)$ هست

و جفتش یکی هست و اینا هم ساده است و هر دوش یه چیز رو تو O و Ω میگن.

maxnote

مرزدهی Θ اگه $g(n)$ و حد تیش $f(n)$ باشه و

چ اگرچه مطلع نیست ولی هست ولی جلد دوم برای این دست که آیا رشد تابع n^3 با n^2+n^3 یکی نهستا؟ بله جشنون

توابع درجه ۳ هستن برای همین مسئله.

؟ اگه دوتا مرتبه داشت باهم کدوم بالاتره؟ یعنی که تابع کدوم مرتبه بالاتری داشته باشه.

پس بخوای بینی کدوم بالاتره؟ لطفا که تابع بالاتری داره MAX.

خیلی مهمه؟ یعنی فرض کنید الگوریتم دو قسمت داره یکی لول یه حلقه‌ای یکی دوم حلقه n^2 پا اجرا میشه. در مجموع

وقت مرتبه الگوریتم و $T(n)$ اش چی میشه؟ $T(n)$ میشه تکرار عمل اصلی توی لول n بار دومی n^2 بار پس

$T(n)$ میشه n^2+n و مرتبه‌اش هم میشه $O(n^2)$. یه سوال اگه این الگوریتم بخش لول رو نداشت مرتبه‌اش

چی میشه؟ $T(n)=n^2$ و مرتبه هم $O(n^2)$ پس مهمه؟ اگه یک الگوریتم چند تیکه پشت سرهم داشته باشه مرتبه الگوریتم

میشه الگوریتم مرتبه‌اش بیشتر از الگوریتم که بیشتر اجرا میشه. چرا؟ چون گفتیم با حذف ضریب، مرتبه چون اون تیکه پس اگه

کوچکه باشه نقشی بازی نمیکنه و بزرگترین رو انتخاب میکنیم. مثلا ورودی هایون رو گذاشتیم تو بخش لول $T(n)=n$

دومی باید یه کاری انجام بدم میشه $T(n)\log^n$ و یعنی هر بار مرتبه بزرگتر میشه یه طور میشه $T(n)=n^2$

برای همین بخش های ساده رو نمیذاریم کنار موقع تحلیل الگوریتم. مثلا تو مرتب سازی جایی اول قسمتی که برای گرفتن

ورودی و چاپ خروجی هست وارد نمیکنیم چون نقشی بازی نمیکرده.

(مثلا یه تابع خطی باشه هرچی توهم ضرب بشه باز هم خطی میشه و حالتش بدتر نمیشه.)

سوال: اگر مرتبه الگوریتم $O(1)$ باشه یعنی چی؟ وقتی $O(1)$ هست یعنی بزرگترین جمله‌اش عدد ثابته و خودش از n نداره و مثلا

$T(n)$ هست مثلاً ۲ یا ۵ یا یه همچین چیزی و به اندازه ورودی حساس نیست و ورودی هرچی باشه تعداد عملیات

اصلی ثابت هست اما ما چنین الگوریتمی نداریم امکان نداره یه بخشی اش این طوری باشه مثلاً هر بخش رو جداگانه

تحلیل میکنیم و مثلاً $O(1)$ میشه ۰ مورد دوم هم توی حلقه‌ها ۵ اگه اولی $f(n)$ بار دومی $g(f(n))$ بار بشه ضرب میشه.

مثالش میشه یه حلقه‌ی تودرتو ، حلقه‌ی داخلی $O(n^2)$ و $O\,n$ چون به اندازه‌ی هر حلقه بیرونی حلقه درونی یکبار کامل اجرا

میشه ، میشه n ضرب در n^2 بار میشه n^3 ۰ مثلاً یه حلقه تودرتو داشته باشیم ؟ بیرونی 1 تا n و درونی $\frac{n}{2}$ O

این دو تا حلقه روی هم مرتبه اش چند میشه ؟ که توی ساختمان داده $f\,o\,r$ می‌گفتیم ، چون حلقه بیرونی n بار درونی

$n/2$ و $n\times n/2$ هم باز مرتبه اش n هست هم بیرونی هم درونی مرتبه اش n هست میشه n^2 ۰ مرحله حذف میشه $\frac{n}{2}$

که میشه $\frac{1}{2}n^2$ و مرتبه اش میشه n^2 ۰

۸ - مرتبه لگاریتمی حساس به پایه اش نیست ۰ یکسان هست ۰ اثباتش کرده و گفته یکی از قوانین لگاریتم هست که گفته بین

شکل پنهانی $\log_b^n = \log_a^n \times \log_b^a$ که a ساده میشه و جزء توی این حالت b یک عدد ثابت میشه

مثلاً $\log_2^n = \log_8^n \times \log_2^8$ ، $\log_8^2 = 3$ پس حذف میشه $\log_2^n = \log_8^n \times 3$

که دیدیم عدد ثابت هم نقشی بازی نمیکنه تو مرتبه پس لوگ با هم برابر میشه ، و اینجوری تو کتاب می‌نویسن این

الگوریتم مرتبه اش $\log^n$ یا $n\log^n$ و نباشه اصلاً نمیگن چون مثلاً $\log_2^8$ با هر عددی برابره پس یه ضریب ثابت

هر عدد ثابتی نقش بازی نکنه بستگی نیست مثلاً مرتبه n^2 یا n^3 فرق میکنه ما می‌تونیم باهم نقش بازی بکنند و هر عدد ثابتی

نمیشه بی‌تاثیر نیست اونایی که عدد ثابت هستن فرقی ندارن ضریب هستن ۰ ولی چیزی n^2 ، n^3 اونا درست

عدد ثابت هستن ولی ضریب نیستن و مرتبه رو تعیین می‌کنن . خاصیت آخر هم میگه اگه یه تابعی مثل $\log(n)$

maxnote

CamScanner

باشد که در پایین $f(n)$ باشد و مثل $h(n)$ در دقیقش $f(n)$ باشد حالا این دو تا رو با هم ترکیب کنیم و به ضریبی

از $f(n)$ و $h(n)$ باشد و جمع کنیم رتبه‌اش $\Theta f(n)$ میشه چون $c_1 \alpha$ که ضریب ثابت هستن حالا

باید ببینیم کدوم بزرگتره g یا h خب معلوم اون در بالای $f(n)$ هست ولی اون در دقیقش $f(n)$ هست

پس در حقیقت اینا به کدوم بزرگتره؟ $h(n)$ چون دقیقاً رتبه‌اش $f(n)$ هست اما اون در بالاش هست و هلکذا

کوچیکتر باشد و $h(n)$ مرتبه رو تعیین می‌کنه.

این قسمت درسمون نه خصوص پیدا کردن مرتبه الگوریتم هست که توضیح دادیم چطوری پیدا میاد و یه بخش هم اینکه ما تابع زمانی

رو خودمون رو تعیین کردیم رابطه بازگشتی اش رو بدستیم و حالا مرتبه اش رو میخوایم. توی مثال های قبلی دیدیم اگه

یه الگوریتم رو به صورت بازگشتی بنویسیم می‌تونیم یه خط هم برایش بنویسیم و $T(n)$ اش هم یه خط بازگشتی شد

و احتمالاً حل شد یعنی پایه خط همین رو از روی بازگشتی بدست بیاریم و این کار یه کار ریاضیاتی هست به نام

حل معادلات بازگشتی و ربطی به تحلیل الگوریتم نداره. این است که به صحبت حل معادلات بازگشتی نیاز داریم.

کتابمون یه مقدمه داره و چند تا کتاب شامل اون قسمتی از ریاضیات هست که بهش نیاز داریم.

خب یه الگوریتمی داشتیم احتمالاً بازگشتی بوده $T(n)$ اش هم بازگشتی شده که بدست آوردیم و $T(0)$ هم شده یه عدد ثابت

مثلاً برابر بود $T(1)$ میشه مثلاً شده یه عدد ثابت مثل ۲، ۳۰. حالا تابع زمانی رو حل کنیم از حالت بازگشتی به

حالت تبدیل کنیم. بعد مرتبه اش رو تعیین میشه. روش های مختلفی داریم یکی از پایه‌ای ترین و مهم ترین‌ها

روش جایگذاری هست. یعنی خط خط بازگشتی رو برای n های کوچکتر تکرار کنیم. یکی رو بهمین دیدیم اش

مثلاً هست. طبق خط داریم $T(n) = T(n-1) + 1$ حالا هم باشه. حل اینطوری بود؟ آدرس نمی‌شه. آیدی‌ها نمی‌کنن

به علاوه 6. الان بهم بگید. مثلا $T(10)$ میشه $T(9)$ به علاوه 6 حالا که اینطوری حوزه‌ش یکیه $T(n-1)$ چی میشه؟

میشه $T(n-1)=T(n-2)+1$

$$T(n) = T(n-1) + 1$$
$$T(n-1) = T(n-2) + 1$$

جایگذاری

تا اینکه ادامش بدیم بعد روش جایگذاری دیگه چند بار انجام بدیم تا نهایت این جواب چند بار تکرار پیدا میکنه. (چطور)

ما باید حالا خودمون رو به مجله آخر برسونیم اینجا هم گفته انتهاش رو به جا مسئله $T(0)=1$ توجه کنید هر جا در شما مسئله بازگشتی

میره باید نقطه توقفش رو هم بگیم و گفته یه دری هست مثل 0. n بار تکرارش میکنیم تا برسه به آخره. اگه n بار تکرار

کنیم به $T(n-n)$ میرسیم یا $T(0)$. بعد اینجا توی جزوه به جای ... نوشته n بار تکرار و به جای $T(0)$

هم نوشته شود $T(n-n)$. که چون $T(0)=1$ و $1\times n$ داریم که میشه $1\times n$ و در کل میشه $C+n$

و فعلا صرفنظر بدیت اومد حالا بگیم $T(100)$ چند میشه نیاز نداریم اون پایینی رو انجام بدیم و خودش رو داریم راحت حساب

می‌کنیم. حالا فرض کنید n یه عدد هست دیگه حالا هر عددی باشه میشه $100+n$. مرتبه الگوریتم هم مشخص هست.

این یک مثال هم روش کار هست فقط یه ذره پیچیده‌تره. قبل حل مسئله. این رابطه رو چند بوجود میاریم. $T(10)$ چقدر هست؟

میشه $2T(5)+1$ فقط دقت کنید نباید 2×5 کنید! این اشتباه هست. (فرقش رو میدانید یا درک میکنید؟ پس خواب بالا)

فهمیدی چرا نباید 2×5 کرد؟ اون عدد 5 داخل خودش که همش مثل مثال قبلی کم میشه ازش! تو بعضی از درخت

خارجش بکنی! اشتباه! حالا فهمیدی؟ حالا فرض کنید $T(1)=10$ یعنی n برابر 10 هست حالا

$T(8)$ چند شد؟

$T(1)=10$

$T(2)=2T(1)$

$T(4)=2\times2T(\frac{4}{2})+1$

$T(4)=$

$T_2=2T(\frac{4}{2})+1$ $T_4=2T(\quad)+1$

$T(6)=2T(\frac{6}{2})$

یعنی نظام بهتر است علاوه این کم ۵ و نظام ابتدا به روش جایگذاری حل کنیم منظورش هم مثلاً مثال قبلی. من برای اینکه

بیاورم و توضیحم سادگی تر بشه $n = 2^k$ پس توی این فرمول به جای $n = 2^k$ و چون اگر $n = 2^k$

2^k به اندازه از طرف دو اینو کم میشه $2^{k-1} \leftarrow 2^k \times \frac{1}{2} = 2^{k-1} \rightarrow 2^{k-1}$ $n \times \frac{1}{2}$

استفاده کردیم چون با جایگذاری کردن متفاوت میشه حالا قطعاً اینو بپذیرید باید بیاید

استفاده می‌کنند تا T و ۲ به تعداد دفعات میشه دوتا T به توان یکی کمتر به اضافه ۱

$$T(2^k) = 2T(2^{k-1}) + 1$$

حالا همین رو برای $T(2^{k-1})$ رو هم همینطور بنویسم:

$$T(2^{k-1}) = 2T(2^{k-2}) + 1$$

مثلاً $T(2^2) = 2T(2^1) + 1$ $T(2^3) = 2T(2^2) + 1$

حالا بگید $T(2^{100})$ چند میشه؟ $T(2^{100}) = 2T(2^{99}) + 1$

$$T(2^k) = 2T(2^{k-1}) + 1$$

$$\begin{cases} T(2^{k-1}) = 2T(2^{k-2}) + 1 \\ \text{پس} \\ T(2^k) = 2(2T(2^{k-2}) + 1) + 1 \rightarrow 2^2 T(2^{k-2}) + 2 + 1 \end{cases}$$

$$\begin{cases} T(2^{k-2}) = 2T(2^{k-3}) + 1 \\ \text{پس} \\ T(2^k) = 2 \end{cases}$$

$$T(2^{k-3}) = 2T(2^{k-4}) + 1$$

جمع $k-1$ تا $k-2$ هست.

سه تا عضو k بار تکرار

2^{k-k} به ضریب 2^k بار تکرار بالای ها - مجموع میشه 2^k یعنی 2^{k-1} و 2^{k-2} و 1000

$1 + 2^2 + 2^3 \dots 2^{100} \rightarrow 2^{101} + 1 = 2^{n+1} - 1$ به شرط اینکه k

subject :

date :

$T(1) = 5$

$T(n) = T(\frac{n}{2}) + n \rightarrow T(2^k) = T(2^{k-1}) + 2^k$

$T(2^{k-1}) = T(2^{k-2}) + 2^k \rightarrow T(2^k) = T(2^{k-2}) + 2^k + 2^k$

$T(2^{k-2}) = T(2^{k-3}) + 2^k \rightarrow T(2^k) = T(2^{k-3}) + 2^k + 2^k + 2^k$

$T(2^{k-k}) + \overbrace{2^k + 2^k + \dots}^{2^k \text{ تا}}$

$\text{۵ تا } k$

$T(2^{k-1}) = T(2^{k-2}) + 2^k$

$T(n) = T(2^k) = T(2^{k-1}) + 2^k$

$$\begin{cases} T(n) = T(n-1) + 1 \\ T(0) = 0 \end{cases}$$

$T(n-1) = T(n-2) + 1 \rightarrow T(n) = T(n-2) + 1 + 1$

$T(n-2) = T(n-3) + 1 \rightarrow T(n) = T(n-3) + 1 + 1 + 1$

$$\begin{cases} T(2^k) = 2T(2^{k-1}) + 1 \\ T(1) = 0 \end{cases}$$

$T(2^{k-1}) = 2T(2^{k-2}) + 1 \rightarrow T(2^k) = 2(2T(2^{k-2}) + 1) + 1$

$T(2^{k-2}) = 2T(2^{k-3}) + 1$

$T(2^k) = 2^2 T(2^{k-2}) + 2 + 1$

$\rightarrow T(2^k) = 2^2(2T(2^{k-3}) + 1) + 2 + 1$

$2^3 T(2^{k-3}) + 2^2 + 2 + 1$

ثابت کنید نامساوی های زیر رو: date:

نامساوی بود عددی بزرگتر از 2^n و $n!$ هست و بعدش n^n هست پس این جمله ها هم هر دو درست هستند

تو تقریب پیچیدگی زمانی اگه بیشتر ثابت کرد که $n!$ از یه ضریبی از 2^n بزرگتره از یه ضریب ثابتی از n^2

کوچیک تر هست. خب اینجا بحثی هست مثلا جایش ۲ بذاریم و بیشتر ۱، ۲، ۳ و اینطوری سمت چپی بزرگتره

پس $n!$ بزرگتره برای این هست که جلوش ۲ بذاریم و یعنی این $n-1$ و $n-2$ ها بزرگتر از ۲ هستن پس اگه همه رو ۲

بذاریم پس سمت چپی بزرگتره مثلا چی هست ۵! و سمت راستی ۲ تا ۵ میشه پس در کل چپی بزرگتر میشه

از ۱ تا $\frac{n}{2}$ → 2^{n-1} پس → یا $\frac{1}{2}2^n$

خوبم ساده است رفتم ثابت کم اومد. اینجا جمله اول n هست جمله های بعدی کوچکتر از n هستن پس به جای جملات

n و $n-1$ و ۲ و ۱ اگه بیاییم n بذاریم پس سمت راستی رو بزرگتر می کنیم مثل این میمونه که برای فاکتوریل ۵، بجای ۵، ۴

۳، ۲، ۱ همه رو ۵ بذاریم خب بزرگتر میشه دیگه.

مثال جالب روش جایگذاری حل میشه و عنوانش میشه: اگه ما بیاییم پیچیدگی زمانی الگوریتمی به صورت زیر با شرایط زیر بدست آوریم. که خب خود این تابع بازگشتی هست پس اول باید حل شه بعدش بدست آورد. ما تا الان

روش جایگذاری و قضیه اصلی و درخت بازگشتی رو میاریم. ما فقط روش جایگذاری رو گفتیم و امروز قضیه اصلی رو میگیم

یادآوری: اگه بخوایم بیشتر مفهومی نگاه کنیم: $T(n)$ تعداد کل اصلی برای حل مسئله در حالت n - $T(n/2)$ تعداد کل اصلی

یادآوری هست بیشتر

برای حل مسئله در حالت $\frac{n}{2}$ نصف n و به علاوه ۱ ← در کل: برای حل مسئله در حالت n اول مسئله رو در حالت $\frac{n}{2}$

حل کنیم و بعد کل اصلی دیگه هم می خوام. این تابع پیچیدگی زمانی تو این مثال برای الگوریتم جستجوی دودویی مقدار یا در نظر گیریم

مثال بعدی: گفتیم هر جا n داریم یکی از تغییر متغیرهای میشه 2^k و یا اگه $\frac{n}{2^k}$ و $\frac{n}{3^k}$ و 4^k بود.

(برای این $\frac{n}{2^k}$ که یعنی تو تری جستجوی دودویی از وسط تقسیم میکرد؟ از نصف عددهای آرایه)

← اینجا از روش جایگذاری ها با استفاده کردیم. — ۱۰۰

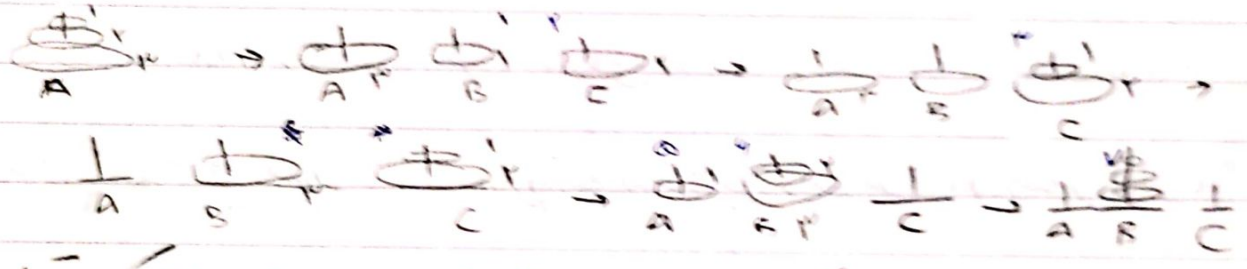
subject :

بحث هانوی: فرض کنید n دیسک روی میله اول داریم به ترتیب اندازه (بزرگترین دیسک در پایین) و ۳ میله داریم.

مسئله این است که دیسک‌ها را از میله اول به میله سوم ببریم، با قوانین داده شده حرکت یک دیسک و هیچ دیسکی نباید روی دیسک بزرگتر قرار بگیرد. برای این ادامه بدهید و بعد بیشتر فکر کنید.

بیت بیدا: اگر ۱۰ میله ۹، ۸، ۷ و ۶ دیسک می‌ذاریم برای حالت ۲ دیسک ۳ حرکت نیاز داریم.

مسئله برای ۳ دیسک چطور؟ ۴، ۵، ۶؟



الان تفکر کن به روی میله A برای آوردن دیسک B وسطی کار کنیم!
من برای کل رفتم ولی واقعا حل درست.

من بخام برای ۳ دیسک کل کنم سه بار اول رو برای ۲ دیسک انجام میدم و بعد کل انتهایی رو هم انجام میدم

(و قرار نظمی برای طبق ۱-۲^n در نظر گرفتم یعنی همون ۷ ← که آخرش بیست لعنت!

برای ۴ تا هم حلش رو بنویسید برای تفاوت بنویسید. بعد از ۴ حالت بالایی رو اصلا نوشته نمیشود در ذهن ما.

قاعده تمام یعنی کلا فرمول دیگه ۲^n - ۱ ؟

نکته: هر وقت دیدی چیزی رو ... کم کردن ... اطلاعات رو نمی‌تونم به آخر بریم

محاسبه نکنی و بنویسی، هر وقت ساده نمیشه تو ذهن حل کنی.

۱- ابتدا تلاش کنیم بفهمیم به ۱ چطور میشه ۲^n - ۱ به دو دسته از ۰ به بالا میشه که از ابتدا

subject: date:

و مباحث هم بدست میاد ۱- 2^n که یه فرمول بازگشتی هم میخوره.

هم چنین تجربه کنید مرتبه تابعی به بالا خیلی مرتبه های بدی هستن چون نامیدور فعل میشه حل کنند.

استفاده از حد برای مرتبه:

اگه دقیقا تابع رو تقسیم کنید بر هم و حدش رو حساب کنید هستی به سمت Θ میره حدش رو حساب میکنه چون همگرا

یعنی توابع اینطوری فرق کنه ولی رشدشون نزدیک میشه و مرتبه بالای تابع هیچ.

پس می خواد تو nهای بزرگ حساب کنه وقتی حد یگه ۰ شد حالت بدست میاد. اگه ۰ بشه

مرتبه چپ میشه؟ یعنی صورت نیست یه جزء کوچیک شده پس تو این حالت gn رشدش کمتر از fn

پس حد بالای اون شده 0 و اگه ∞ بشه؟ جزء کوچیک شده. یعنی gn رشدش بیشتر

بوده پس ω میشه و اگه یه حد ثابت هم بشه رشدهاشون نسبتا هم بوده ولی خیلی نمیتونه

بیشتر شده و خیلی از هم بودن نیست Θ شده هم مرتبه هستن.

بخش مهم و کاربردی درس (هم امتحان هم کنکور) - قضیه اصلی

وقتی یعنی چطوری جایگذاری حلش نشه به ما کمک میکنه. و برای همه معادلات بازگشتی بخشه ازش استفاده کرد.

برای اون هایی که میشه خیلی راحت هست و اثباتش رو میگذریم. قضیه اینو میگه: اگه تابع زمانی به این صورت باشه

$T(n)$ آماری است با یه ضریب و تقسیم بر یه عدد + cn^k در این صورت

که a و b و c و k یه سری عدد هستن که باید توی این شرایط باشن. زیدش نوشته. مهم ترین

b هست. T هم هر چی باشه مهم نیست. تو این حالت سه وضعیت بدست میاد که اینطوری خلاصه میشه اگه a b^k

مرتبه الگوریتم n^k هست. و خودش قضیه مجبوری اثبات میشه هم لازمی نداره.

که همه این حالت های کامل کردیم با این هاش هم بدست میاد.

مثال ب - ۲۶: معادله ۹، ۲ و ۱ رو جایگذاری کرده حالا مرتبه هارو بدست میاریم و $a=1$

$b=3$ ، $c=0$ ، $k=2$ حالا مرتبه اش رو بدست میاریم کسم حالت بدست میاد: $8<4^2$

که میشه از مرتبه n^2 و حالت اول اتفاق افتاده اگه $T(n)$ توی این قالب بگنجه میشه از این روش

استفاده کرد اما مثلا بعضی ها مثل برج هانوی توی این قانون نمیگنجه.

و سوال بعدی هم آخرش اگه حساب کنی $\log_3 9$ میشه ۲ پس میشه n^2. البته دقت کنید این روش مرتبه

رو میده، تایم رنگی رو بدست نمیاره! آشنوع هم راه حل هست توی قضیه اصلی و روی مرتبه تاثیری نداره.

حالا برای مثال قبلی مثلا: $2T(\frac{n}{2})+n$ ← $a=2$ $b=2$ $k=1$

$\Downarrow$

$Cn^k \to n^{\log_b k}$

کمیته $n\log_n$ و $2<2^1$

و توجه کنید خود قضیه اصلی با روش جایگذاری اثبات شده و فقط ساده شده به این حالت.

خب حالا بیاین برای این معادله بیشه قضیه اصلی حل کرد و بگید مقادیرش رو؟

$T(n)=T(\frac{n}{2})+1$

$a=1$

$b=2$

$c=1$ → چون هر دو یکسان

$k=0$ → کدوم تعریف $k=0$ باشه

$1=2^0 \to n\log n \to \log n$

این یکی هم حل کنید ← $T(n)=2T(\frac{n}{2})+1$

maxnote

CamScanner

subject: (مثال‌های قبل از بچه‌ها نوشتی) جا مونده بود date:

نکته: تعجب نکنید چرا و چطور الگوریتمش کار کند که تابعش این باشد؟ $T(n) = 2T(n/2) + n$

بله این باید برای حل اصلی برای حل مسئله توی حالت $n/2 + n$ عمل اصلی و دوباره باید مسئله رو توی حالت $n/2$

حل کنم و n تا هم عمل اصلی می‌خوام تا بتونم نتایج رو بدارم با هم جمع کنم تا مسئله رو برای ورودی n حل کنم.

و نکته مهم روش تقسیم و حل هست چون n رو تبدیل کرده به دوبار حل مسئله توی حالت نصف اون (n)

مثال: $f(n) = 1$ $f(n) = n$ $f(n) = n^2$ این رو از روش جایگذاری حل کردیم خیلی راحت.

معادله زوج: گفت اگه این به یک که برای یه الگوریتم باشد مرتبه الگوریتم رو به دست بیارید.

اینجا گفته که a و b که یه حالت تو درختش داخل حالت هم عمل اصلی خواسته. و اول مرتبه قالی و بعد مرتبه از روی اون. اول حساب کنیم عمل اصلی چند بار اجرا میشه که تعداد تکرار عمل اصلی بدست بیاد. هر بار که اجرا میشه یه بار عمل اصلی اجرا میشه اگه تعداد تکرار عمل اصلی رو این خانه بدست بیاد لازم حساب میشه. اگه برعکس بود کار راحت بود. n بار اجرا میشه ولی این جا از اول از حالت n پس تست می‌خواد. اگه مسئله از روش جایگذاری کنیم بیشتر طول داخلی چند بار اجرا میشه. $a = 1$ یک بار حالت تلفی = یعنی یک بار عمل اصلی و میره تا n یعنی n بار. که خب تعداد خیلی هستن و فرمولش این نیست و تا خودش.

و اینو به صورت دیگه هم نوشتیم. و از این فرمول توی مثال بعدی هم میشه استفاده کرد. مثال بعدی: در حالت تو درختت

یعنی n بار عدد یک رو جمع کنم میشه خود n، دیگه اون رو حذف می‌کنم n میشه. یعنی داخلی ترین حالت n بار. $\sum_{i=1}^{n}$ هم که داشتیم و در کل فرمولش خیلی به دست میاد از بالا هم فرمول $\sum_{i=1}^{n}$ رو داشتیم پس جواب این میشه.

subject: فصل دوم — طراحی الگوریتم — date:

ما مقایسه‌ی الگوریتم های کامپیوتری داریم و به ازای هر مسأله یک یا چند الگوریتم ارائه شده ولی از منظر اینکه از چه ایده اصلی استفاده کردن خیلی متفاوت هستن و به چند دسته قابل تقسیم هستن. از فصل ۲ به بعد هر کدوم از ایده‌ها و معرفی چند الگوریتم که از این ایده عنوان شده استفاده کرده.

کلیات

(تقسیم و حل): که تقسیم و غلبه هم بهش میگن این ایده از یه رویکرد قدیمی در جنگ گرفته شده مثل فتح و پیروزی به قطر عجیب اسمش میگیند. ایده توی قدیم این بود لشکر رو به ۲ بخش از نظر تقسیم می‌کنند و اول برای بخش اول میرن و بعد بخش دوم. و به این صورت لشکر مقابل رو از بین میبرن. و هر کدوم در واقع از جنس مسئله بزرگتر هست. حالا یا اگر مسئله کوچک حل شد بزرگ هم حل شده است نیاز به کار دیگه‌ای نیست ولی بعضی موقع‌ها باید روی نتایج مسئله‌های کوچک هم عملیاتی انجام بدیم تا حل مسئله اصلی بدست بیاد مثل الگوریتم بازگشتی فیبوناچی.

یه سوال هم که وجود داره اینه که اون مسائل کوچیک حل میشه؟ جوابش اینه با تکرار همون روشی که برای حل مسئله‌های کوچک استفاده کردیم. لزوماً حل میشه یعنی کوچیک‌تر دیوار شکسته میشه و اصلاً روش تقسیم و حل ذاتاً بازگشتی هست. از ایده روش تقسیم و حل یک ایده و روش بازگشتی هست پس موقع پیاده‌سازی تابع اش یک تابع، دوال بازگشتی هست. نکته: برای مسئله‌های بزرگتر اون رو نمیکنم به کوچیک. حالا دوباره اون ویژگی‌ها ... تا کی باید ادامه بدیم؟ جواب: مسائل کوچیک وقتی رسیدن به مسائلی میرسن که حلش سادس و آسون هست مثل مرتب کردن یک آرایه. اونقدر کوچیک که بدیه کرایه برسیم که دیگه مرتب هست توش. ذهن انسان، تجزیه‌گرا هست، روش Top down هست و خوب بلدیم اینجا ایده اصلی همین روش قابل درکه.

روش‌های استقرایی مثل Bottom top هم داریم. ذهن‌های قوی ذهن‌هایی هستن که Bottom top هم بتونند هر دو روش

مخصوصاً در حوزه علوم تجربی و دانشمندان.

روش جستجوی دودویی

در واقع به لیستی یا آرایه ای از عناصر داریم و دنبال عنصری مثل x میگردیم که ببینیم توی آرایه وجود داره یا نه. روش کار هم ساده بود و مفهومش این بود که آرایه از عنصر کوچک به بزرگ مرتب باشه و روش این بود وسط عنصر رو پیدا می کردیم و یا از سمت راست یا سمت چپ تا برسیم به x یا اینکه به ته برسیم.

تو این مثال یه آرایه داریم که مرتب هست. عنصر وسط رو پیدا کرده 25 و مقایسه و کوچکتر پس زیر آرایه سمت چپ. 13 باقی میشه 4 باقی پس تک هست و مرحله آخر رو داره. و به این نوع میگن tail recursion که به حالت غیربازگشتی راحت تبدیل میشه که تو درس ساختمان داده ایزیادی اونو تمرین رو نوشته بودیم. (تقسیم و حل هست binary search و از یک عنوانی کوچکتر تشکیل شده)

حل: فرض می کنیم یه آرایه ی مرتب داریم که وظایفم این مسئله رو پارس حل کنیم و برای همین که دو متغیر یه L و H آورده اندیس پایین و بالا محدوده جستجو آرایه. و بعد هر مرحله آرایه کوچک تر باشه اگه چپ یا راست باشه میتونه H و L یکی کمتر و کوچک بشه.

خط بعد: یه آرایه ی مرتب داریم و وظایفم تو محدوده L و H جستجو بکنه x رو پیدا میکنه و گفت چون low > high هست پس عنصری پیدا نشده و محدوده جستجوی وجود نداره و x رو پیدا نکردیم و صدا زده (اول این چاپ چک کنه)

محدوده چک هم اول mid رو پایین مرحله قرار مقایسه میکنه اگه مساوی بود خودش رو برمیگردونه $h=6$ و $7/2=3$ $L=1$

x میگیره x رو باید مقایسه کنه و اگه کوچکتر بود بره تو این محدوده low, mid-1 سرچ بکنه. و اگه بزرگتر هم این یکی محدوده.

پایه ی کوچک آرایه ی توی سمت چپ قرار داره. زیر آرایه سمت چپ

همین پایه ی سرچ رو اجرا کن و برای محدوده دیگه. این کار تا زمانی ادامه پیدا میکنه که x پیدا شه یا $L>h$ بشه و x پیدا نشه

تمرین: (برای فصل بعد که باید ترجیحاً C++ یا Plot رو استفاده کنیم)

{

else {

چون بزرگتر نیست الگوریتم پس تابع زمانی هم بازگشتی میشه

}

برای پیچیدگی زمانی جستجوی دودویی بدست آورده - حل اصلی مسئله هست.

}

پیچیدگی زمانی که قبلاً هم بدست آوردیم: این روش هر بار روش جایگذاری هم قبله اصلی حل میشه و طبق قضیه master بدست میاد

ادغامی (Mergesort)

تا حالا یکی دو روش گفتیم مثل جایی، تقسیمی. که از این روش حل میشن. یک الگوریتم با روش تقسیم و حل هست

فرض کنید ۲ تا دسته کارت داریم روی هر کدام یه عددی و می‌خوام روی هم بچینید و مرتب کنید. مثلاً دو تا دسته کارت

۲ تایی میگم اینا رو مرتب بهم بچین کدوم راحت‌تره؟ تو هر مرحله نگاه می‌کنید کدوم عدد کوچیکتره اول رو میذارید بیرون

بعد که مرتب بود مقایسه‌های بیشتری انجام میشه. تو هر مرحله فقط باید نگاه کنم کدوم کوچیکه اونو بذارم بیرون

کسی که اینو طراحی کرده تو ذهنش این بوده که دو تا دسته کارت مرتب داشته باشه ریختن روی هم

اینا و مرتب کردنش راحت‌تره نسبت به حالت دیگه. پس میاد یه آرایه n تایی داریم و $\frac{n}{2}$ میشه و هر کدوم

رو مرتب می‌کنیم و اونا رو ادغام می‌کنم تا به یه آرایه مرتب n تایی برسم. و فرقش با تقسیم و حل در واقع

Divide & Conquer هست و به کمک بازگشتی قسمت تبدیل میشه و مرحله ادغام هم داره. و هر کدوم از $\frac{n}{2}$ ها را هم دوباره با کمک همین روش

تقسیم اول. تا برسه به ۱ که دیگه نمی‌شه.

مثال: یه آرایه ۸ تایی داریم می‌خوام اینا رو مرتب کنم اول تقسیم می‌کنم تا دو آرایه بشه و اتفاقی روی محتوایات نیفتاده

حالا باید این‌ها رو مرتب کنم. اول ۴ تایی سمت چپ و تا به ۲ برسه و اونو دو تا رو مرتب می‌کنیم و ترتیب می‌کنیم

خود این ۲ تا رو انقدر کوچیک هست که بازی نداره دیگه فرقش مرتب هست یک تفاوتی مرتبه.

دقت کنید منظور ما نشانه divid و ↗ ادغام است و این ها رو پاکسازی ها ادغام می کنیم. تا به ادغام هر

۸ عدد برسیم. اینکه با چه روشی و چطور! ادغام همیشه عددا ...

حالا دوتا آرایه ۴ تایی داریم مرتب هم هستن ادغامشون میکنیم بعد بالاتری ها نگاه می کنیم اول روی 10 تایی 93

اول میرسیم رو 92 دومی رو 13 اولی 20 دومی 13 دوباره همینطوری تا آخر. هرچی آخرش نمونده رو میریم می نویسیم از چپ یا راست هر چی خواستی

حواست باشه در نظر میگیریم و این روش ادغام رو براش انجام میدیم. دقت کنید این ۳۰ مرحله کلی، داخل هر کدوم از اونها هم اتفاق میافته برای

ساده کردن من فقط دادم ۸ تایی نشونش بدم ولی این ۳۰ مرحله داخل هر کدوم اتفاق میافته.

شبه کد: سوال پایانی: یه آرایه گرفته با اسم S و طول n اول n/2 رو یه بازه برداشت و n/2 رو هم n-n/2

باهاش همین صورت رو تب کن.

و حواست آرایه میاد U و V یکی سمت چپ یکی راست. برای n>1 این کارها رو انجام میده. اگه n=1 باشه که یعنی آرایه تک عضوی و تمیز

و توی اینپیس S از i تا n رو کپی کن توی U و برای V (سمت راست) هم همین کار رو کن. (اندیس ها رو چک کنید)

(برای این کپی دو تا شرط نوشتن استفاده می شود در واقع) (آرایه ۸ تایی S، ۴ تایی های U و V)

حالا تا اینجا مرحله ۱ بود تقسیم شدن بزرگ به دو تا کوچیک. و خط های پایین مرحله ۲. میگه برو merge کن

برای آرایه U، محاسبه بزن و همین رو برای آرایه V که n/2 عضوی هست. و دو خط پایانی مرتب

هستن برای U و V و دوباره داخل این چهار سورت ها باید مرتب بشن و ادغام این ها رو برای merge (مرحله ۳) نوشته

و معنی این اینه: آرایه ی U، ۱ عضوی رو با آرایه V، ۱ عضوی ادغام کن بریز تو آرایه S نتیجه رو

برای همین بچه ها: ۵ به خانه برای سمت چپ یعنی U (i) و راست V (j) و بعدش روی (K)

subject :

که هر ۳ توی ابتدای کار ۱ هستند. روی شروع هر کدوم ۰ (while میاییم)

تو هر مرحله i و j رو با V مقایسه می‌کنه یعنی عنصر i از آرایه U رو با j از V مقایسه کردن و اگه یکی فهمید کدوم کوچیکه بود پس اونو منتقل می‌کنیم پایین. و بعدش عنصر i از آرایه چپ رفته میشه و به عدد بعدیه بالا

و تا وقتی یکی از زیر آرایه‌ها تموم شد اینا رو میدارم تا زمانی که $i<n$ و $j<m$ باشه. از اینجا به بعد نمیدونم باشه.

(تعداد M و N یکی هست و یه متغیر از هر کدوم هر کدوم)

else → آخر آخرش } → برای while

if (i<n)

حالا که یکی از شماره‌ها به انتها رسیده. اون شماره‌ای که روی اون یکی عدد ها توی S می‌ریم و باید بفهمیم کدوم بوده

تموم شده $n<i$ و در غیر این صورت $m<j$ یعنی راست به انتها رسیده و یکی شماره‌ای هنوز که باید به S منتقل بشه.

(نکته: درجا با in-place آرایه‌ای اضافه تعریف نشده روی همون کار انجام میشه و اینجا U و V رو گرفتیم پس یک دونه بیشتر)

(هر جایی in-place بود تا الان.)

جمع‌بندی کلی اینه: می‌خواهیم آرایه U رو با آرایه V مرتب کنیم بدیم تو آرایه S که n عنصری و M عنصری هست

۳ متغیر میاریم i که می‌خواد تو آرایه U چپ باشه (میگیم همینم رو زیر آرایه U چپ به راست)

یه V و یه K که روی S می‌خواد حرکت کنه. و تو ابتدا برای همه ۱ هست. تو هر مرحله i U رو با j V

مقایسه میکنه یعنی عنصر i رو از آرایه چپ U و ... V مقایسه میکنه هر کدوم کوچیکتر بود اونو منتقل میکنیم تو آرایه

S تو خانه K اش این رو میذاریم پایین و اون یکی رو میذاریم سر جاش. و در غیر این صورت else اولی بزرگتر

و اونو میذاریم پایین تو K میذاریم و j رو یه واحد اضافه کنیم. تا وقتی که یکی از زیر آرایه‌ها تموم نشه اینا رو این کار

و تا زمانی که $i<n$ و $j<m$ هست اینا رو. (اون بالا نوشته این)

هر موقع تموم شد بیاد از حلقه while

اون شماره‌ای که مونده میریم روی پس چک می‌کنیم کدوم تموم نشده اگه $n<i$ پس چپ تموم نشده و راست شماره‌ای هست که میریم پایین

و در غیر این صورت اون یکی حالت

maxnote

(پیچیدگی زمانی Merge sort)

عمل اصلی ما مقایسه هست. که توی تابع Merge اتفاق میافته مقایسه من حالا سوال اینه اگه آرایه ما عضو 10 باشه

مقدار دفعه توی این Merge چند تا مقایسه اتفاق میافته؟ یا دو تا مقایسه به عنوان میانه پایین دل بندیم از چپ میره پایین یا راستی

و الان من عدد ثابت بیشترین و بهترین حالت تلقی. فکر کنید همه عناصر سمت چپی کوچکتر از عناصر سمت

راستی اولش باشن پس همه از چپ میرن پایین تمام میشه و بعدش انگار روی سمت راستی فقط دارن مارک کنن کار تموم

پس تو بهترین حالت 10 مقایسه لازم میشه تا وقتی که میشه یه کار کنم این خانه‌ها دیگر بدون مقایسه، تا مقایسه‌های دیگر انجام

میشه. استوری یکی میومن، 10 راستی و 9 تا راستی رفتن پایین پس به جز آخر مقایسه انجام میشه تا آخر

و چون 10 تا تو خانه‌ها خارج میشه پس 20 میشه.

بدترین $m+n-1$ و بهترین $m+n$.

اول بدترین حالت: به جای T، W گذاشتم. پس پیچیدگی زمانی برای n میشه زمانی $n+m$ —

برای n می‌تونیم حل کنیم اول جمله، برای سمت چپ، بعد سمت راست حل میکنیم و بعد ادغام میکردیم دیگه.

$n+m-1$ که خود $n+m$ همون n آرایه گذاشته شده. که خود $m+n$ میشه $\frac{n}{2}$ تقریباً و دست میاد.

تقسیم و حل هم به صورت بازگشتی میشه پیچیدگی‌اش، و برای ادغام هم تقسیم و حل هست پس رابطه استاندارد میشه.

که با روش جایگذاری هم حل میشه و قضیه اصلی (اگه $n-1$ رو در نظر نگیریم) میشه $n\log n$

که بهترین حالت: یعنی مقایسات لازم برای ادغام چپ و راست کمترین مقایسه رو بخواد که زمان میشه که کلاً یا یکی سمت

راست و اون یکی کپی میشه که با اون آرایه پس مقایسه کنید با دو تا هم برابره! tail است. و مرتبه‌اش همه $n\log n$

هست پس نسبت به جبری و تقریبی برتره چون n^2 بود اونا.

subject :

date :

یاسمین آسوده‌ای ۰۰/۲/۱۵

سوال ۱:

$T(n) = T(n/2) + n$

$T(1) = 5$

$n = 2^k \rightarrow T(2^k) = T(2^{k-1}) + 2^k$

$T(2^{k-1}) = T(2^{k-2}) + 2^{k-1} \rightarrow T(2^k) = T(2^{k-2}) + 2^{k-1} + 2^k$

$T(2^{k-2}) = T(2^{k-3}) + 2^{k-2} \rightarrow T(2^k) = T(2^{k-3}) + 2^k + 2^{k-1} + 2^{k-2}$

$\vdots$

$= \underbrace{T(2^{k-k})}_{T(2^0)=5} + 2^{k-1} + \dots + 1 = \underbrace{5}_{c} + 2^k - 1 \in O(2^k) = O(n)$

سوال ۲: گزینه ج

$a = 8 \quad b = 2 \quad k = 2$

$a > b^k \rightarrow 8 > 2^2$

$n^{\log_2^8} \rightarrow$ که با توجه به قانون های log در پیچیدگی زمانی $\log^n$ داریم هست

$\rightarrow \Theta n^3$

n^3

سوال ۳: گزینه ب به خاطر مرتبه‌های آنها

$n < \log n \log n$

$n > \log n^2$

الف مشکل غلط اینجا بود که مثال تزدی با برات راحت باشه!

یعنی اگه به جای هول سئل میگفتی $\log_2^8 = 3$

اونطوری راحت متوجه میشدی! $27 > 9 > 8$

اینم نتیجه به اندازه کافی عربت نگرفتن!!!!!

mohnote

فصل دوم روش تقسیم و حل

از فصل دو به بعد توی هر فصل یه رویکرد اصلی در طراحی مسائل الگوریتم کامپیوتری ارائه میشه. ما تعداد زیادی الگوریتم های کامپیوتری داریم. به ازای هر ایده مسئله ولی در کل خیلی متنوع نیستن از نظر ایده و کلا چند دسته تقسیم میشن.

ایده های اصلی چند تا چیز محدود هستن و بعد هر ایده اصلی چند الگوریتم که از اون ایده اصلی استفاده می کنن رو میگه.

خیلی از مسائل که تا حالا حل کردیم به طور ناخودآگاه یا خودآگاه از این ایده استفاده کردیم یعنی غلبه و فتح که توی چند جا هم استفاده میشه. مثلا یا یه مرتفند لشکر رو به چند بخش تقسیم می کنه مثل مونگولها، بعد حالا ایده غلبه رو تکنیک میده و غلبه میکنه و بعد یه دراز تقسیم پیدا میشه این ایده هم از اینجا میاد و هر مسئله کوچیک جنس این از اون مسئله اصلی هست که فقط کوچیک شده. با حل مسئله های کوچیک حل مسئله بزرگتر و ایده اون تبدیل میاد و حالا این کار یا با کار اضافی همراه خواهد بود یا چند مثل همین مثالی که الان زدم بگذر ولی بعضی وقتها بعد حل اون مسائل کوچیک باید یه عملیات اضافه ای انجام بدیم تا حل مسئله اصلی به دست بیاد. و نکته اینکه اون مسائل کوچیک چطور حل میشن؟

جوابش باید با تکرار همین روشی که برای مسائل کوچیک استفاده کردیم. و روش تقسیم و حل اصلا ذاتا روش این بازگشتی هست یعنی تابع حل تابع بازگشتی، پس ایده و روش تقسیم و حل یک ایده و روش بازگشتی هست در ذات خودش، وقتی پیاده سازی می کنیم، میشه یه تابع بازگشتی یه روال بازگشتی.

حالا نکته: ما همش با تکرار همون روش مسئله های کوچیک رو حل می کنیم تا کی این اتفاق میفته؟ جواب اینه مسائل کوچیک وقتی رسیدن به یه جایی میرسن که حلش خیلی آسون و راحت هست.

مثلا یه آرایه تک عنصری به شما بدم بگم مرتبش کن شما چی میگی؟ میگی مرتبه. پس وقتی اونقدر ساده میشه دیگه

خودش راحت حل میشه ذهن انسان فقط ذهن تجزیه گرا است و روش بالا به پایین مثل روش top down
رو خوب بلده انجام بده و این ذهن تجزیه گرا است در مقابل هم داریم روش ترکیب گرا هست مثلا یه ذهن
شناس، از یه نمونه سنگ و سنگ های کوچیک یه نمونه بزرگتر رو حدس بزنه، ذهن ذره بینی، مثلا انسان های
دانشمند اون قسمتی که خصوصا میتونن نتایج بزرگتر رو از نتایج کوچیک دربیاد بیارن علاوه بر تجزیه کردن.
روش جستجوی دودویی: یکی دیگه از روش ها که از این ایده استفاده میکنه البته تا حالا به این شکل بهش نگاه نکرده بودیم
که این یک الگوریتم تقسیم و حل هست. میاییم یه آرایه ای از اعداد و دنبال عددی مثل x میگردیم که یکم محل
اش رو پیدا کنیم. روش کار خیلی ساده بود. البته فرض ما این بود که آرایه مرتب باشه.
مثال: یه آرایه مرتب هست برای n=18 اجرا کنه. عنصر وسط رو پیدا میکنه مثلا 25، اگه x بزرگتر بود یکی یا چند تا را
مقایسه می کنه اگه کوچکتر بود میره سراغ زیر آرایه سمت چپ، که چپ آخرش چون 16 بزرگتر هست پس وجود نداره.
که اینجا این چند مسئله به یک مسئله برای حل کردن تبدیل شده. اینجا چون بزرگتر فقط به یک نمونه کوچک تبدیل
شده درسته تو ذهن ما چند تا هست ولی در واقع یه نمونه هست و مرحله ترکیب نداریم اینجا. جستجوی دودویی tail recur
هست. اگرچه میشه به صورت بازگشتی نوشت اما به راحتی هم میشه غیربازگشتی نوشت که مثلا تو ساختمان داده گفتیم.
توی این الگوریتم فرض میشه یه آرایه سراسری داریم پس تو ورودی ها نیاورده یه L و H آورده و اندیس
پایین و بالای محدوده جستجو میشه مثلا اینجا L=1 و H=13 یعنی محدوده آرایه رو کلش رو شامل میشه.
ولی مثلا اینجا که مقایسه انجام میدیم محدوده جستجو کوچکتر میشه L=7 H=6 میشه چون عنصر وسط محدوده اش 7 بود
و میدیم سمت چپ L دست نمیخوره ولی H تغییر میکنه. میدیم تمرین که H با بودن در محدوده حتی

شده که ۰ برمیگرده ۰ else هم اول نصف وسط رو پیدا میکنه حالا n رو مقایسه میکنه با عنصری که توی آرایه‌ی

توی مکان وسط قرار داره اگه مساوی بود که پیداش کرده و برمیگردونه اگه کوچکتر باشه میره سراغ سمت چپ ۲ که همونه

اسمش میشه low, mid-1 که گفتیم توضیح اش رو ۰ یعنی همون تابع رو صدا میزنه ولی برای H-1 چیه ۰ و اون قسمت کوچکتر دوباره

داره همینطوری حل میشه که توشه ۰ اگه مساوی نبود که کوچکتر هم نبود یعنی بزرگتره پس برای اون حلش میکنم ۰ این کار تا جایی

ادامه پیدا میکنه که S=mid یا کوچکتر بشه L<H تو همین ها همش اینطوری بودن ولی هی دارن

کوچکتر ترمیش به هم نزدیک میشن و اگه نزدیک شن از هم میگذرند که یعنی دیگه عنصری نیست پیداش

نکردیم ! یعنی ۰ برمیگردونم ۰ از روی کدهای Binary هم مطلوبه یه الگوریتم بازگشتی هست چون خودش داره تابع خودش صدا زده

میشه ۰ خودش هی کوچکتر رو پیدا میده و نتیجه هر چی بود کارامو انجام میده ۰ همینو xor tiny میگه پس tail هست ۰

تابع زمانی : خب گفتیم وقتی بازگشتی هست پس تابع اش هم بازگشتی میشه دیگه ۰ T(1) = 1 هست و اگه

یک عنصر داشته باشه پیدا میشه و نیاز به عمل خاصی نداره ۰ و تابع زمانی یا اندازه n با یک مقایسه که انجام میده + 1 درجه

میشه که خب نصف هست برای زیرآرایه چپ یا راست چون تعداد اون حدوداً n/2 هست پس T(n/2)

مقایسه نیاز داریم که به صورت بازگشتی رو میاد که میشه حل خودش اصلی + 1 مقایسه‌ای که نیاز داریم ۰ و اینو با هر

دو روش جایگذاری و قضیه اصلی حل کردیم ۰ که O(log n) میشه و مرتبه پایین و کارایی هست ۰

الگوریتم بعدی : لوغاریتمی (جایی تقریبی ادامه‌ی تا الان)

یه الگوریتم یا روش تقسیم و حل هست ۰ از یه نکته مهم استفاده میکنه ۰ مثلا فرض کنید دو تا دسته کارت روی هر کدام یه

دسته کارت و میگه لیاقت رو هم بریزید و مرتب اش کنید ۰ حالا اگه خودتون مرتب باشن راحتتره یا برعکس ؟

یادت نرفته

subject :

مرتب باشه راحت تره چون تو مرحله توی هر دست تون نگاه می کنید و هر کدوم کوچیکتر بود اونو میذارید و تو هر مرحله کافیه نگاه کنید کدوم دومی کوچیک هست و خیلی راحت تره تا اینکه مرتب نباشه و مقایسه هاش زیاد شه

پس کسی که اینو طراحی کرده توی ذهنش همین بوده که اگر مرتب باشه خیلی راحت تره تا نباشه.

دو تیکه اش می کنم هر نصفشو مرتب می کنم و بعد ادغام می کنم تا به یه آرایه مرتب n تایی برسونم.

مرحله ۳ دفعه کلا. و اسم همین اینهمش ادغام هست. مرحله ۳، همون ترکیب هست.

تو مرحله دوم، هر کدوم از اون آرایه های $n/2$ رو چطوری مرتب می کنم؟ با تکرار همین روش. و کوچیکتر میشه و میرسه به آرایه تکی و مرتب هست.

مثال: این شکل برای جالفاتین این الگوریتم است. یک آرایه ۸ تایی داریم و قرارم مراحل مرتب سازی ادغامی رو براش بکنیم. مرحله اول کسرات تره لحاظ به خط بکشیم و تقسیم اش کنیم. حالا باید این ۴ تایی ها رو حل کنیم هر کدومو و بعدش ادغام کنیم. خودشو به دو تا زیر آرایه تقسیم کنم که کردم. (M↑ , d↓) و اونارو مرتب کنم و ادغام کنم. حالا 27 ,10 رو چطور مرتب کنم؟ با تکرار همون روش. که تکی ها دیگه خودشون مرتب هستن آرایه تک عضوی خودش مرتبه. بعدش باید برسیم به مرحله ادغام که میایم 27, 10 رو ادغام می کنیم، بیانه چطوری! مرحله بعد این که این آرایه دو تایی 20 , 72 رو هم مرتب کنم و دوباره با همین روش حالا ادغامشون میکنم تا آرایه مرتب ۴ تایی به دست بیاد این 10, 72, 20, 27 همون مرتب شده آرایه ۴ تایی بالا هست حالا یادتونه ۸ تایی رو کردیم ۲ تا چهار تایی و بعد دوباره اونو مرتب کردیم حالا میریم سراغ آرایه ۴ تایی دوم که اونم با همین روش دوباره مرتب میشه برای اون نمونه بزرگتر که مرتب شد حالا مرحله ۳ باید براشون انجام بشه. حالا

چطور ادغام کنم؟ دو لول رو 10 دومی رو 13 کوچکتره رو میارم پایین نیم 12 لول 12 دومی 13 میارم پایین

12 ، لول 13 دومی 20 (برعکس میچسبانیم) و 13 رو میارم پایین لول چهارم رو 20 دومی 22 و بعدش

27 و دومی 22 رو میدم پایین و 27 و 25 و دومی تمام شد لول هرچی باقی مونده بود میاوردم

پایین ، یه شمارنده رو آرایه لول یه شمارنده رو آرایه دوم و هر بار حقایه رو میارم پایین و میبرم جلو

نکته :

(این سه مرحله فقط برای اون n تایی هست یعنی برای n تایی مرحله 1 مرتب شدن لول 0 تا مرحله 2 مرتب شدن

و ادغام شدن لول 2 تا مرحله 3)

شبکه

خب سلیقه کتاب این هست برای هر الگوریتم یک تابع بنویسیم که ورودی هارو بگیره اینجا شبکه خروجی هارو بده اینجا یه تابع داریم

به اسم Merge Sort این عدد طول این آرایه رو نشون میده n و میخواد sort کنه ، لول h رو میاره

h بعد $h[\frac{n}{2}]$ $m=n-h$ مثلا $m=10$ برابر $h[5]$ و $n=15$ و که نوشته h و M برابر سمت چپ

و راست هستن که تقریبا هم $\frac{n}{2}$ هست اگر زوج باشه دو آرایه U و V میاره که V سمت راست و U سمت چپ

که اندازه U ، h تایی و لول m تایی هست که بیگ اگر بزرگتر از 1 باشد که بیگ نیازی نیست مرتبه اگر کوچک یا

مساوی 1 باشد و بیگ زیر آرایه کمتر چیزی از آرایه S از اندیس 1 تا h رو کپی کن تو آرایه U و در لول بعدش

از h+1 تا n کپی کن تو آرایه V که (تو تو نقل هم لول n تایی میشه و لول تو تا U و V) و با حلقه for

انجام میشه در واقع نه copy حالا میگه این دو تا خط تا اینجا مرحله 1 بود تا کپی و مرحله دو میشه دو خط بعد بیگ برو

همین تابع رو مجموعه کاهش کن برای آرایه U و همین رو مدام بزن برای آرایه V که U ، h صفری

subject :

و V ، m عنصری) این خط که آرایه U هست که n عنصری هست مرتب میشه و اون هم که آرایه V هست

و m عنصری مرتب میشه . (مرتب این تقسیم و حل طبیعی که اون قسمت های کوچیکه هم چین و قاره و بلدن

و نوشتن این رابطه) حالا باید اینارو ادغام کنیم و توی تابعی به نام merge بنویسیم که شکلش نوشته شده میگه آرایه U

n عنصری رو با آرایه V ، m عنصری ادغام کن و نتیجه اش رو بریز تو آرایه S .

(برای اینکه void merge آخرش } پایین [S[k بذارش) (برای while هم بالای if بخواد }

بشه که تابع : می خوایم آرایه U و V رو مرتب کنیم بریزیم توی S نگاه کنید سه متغیر میگیره یکی i که میخواد روی آرایه سمت

چپ باشد که میگفتم چشم روی سمت چپ باشه و یه j میگیره برای حرکت روی آرایه V و یه k که روی S

حرکت میکنه که چون میخوام بریزه پایین باید توی S بریزه پس لوپ می خواد که هر کدوم شروعش روی 1 هست

و پایین S رو هم ابتدا یعنی 1 هست که 1 . تو هر مرحله U[i] رو مقایسه میکنه با V[j] هر کدوم

کوچیکتر بود توی آرایه که توی محل k این عنصر i از آرایه U رو میریم پایین یعنی i رو میدیم میریم جلو

و k رو هم میریم جلو چون یه عضوشو ریختیم باید بریزه بعدی در غیر این صورت آرایه j رو میریم پایین

توی k ام و بعدش j رو میدیم افتاد کنم این کار تا لحظه ای که یکی از زیر آرایه ها تموم بشه لوپ میدا میکنه

پس میگه پس کارو تا وقتی که شمارنده آرایه چپ به انتهاش نرسیده باشد و آرایه سمت چپ از انتهاش نگذشته

باشد یعنی از n و M نگذشته باشد . حالا که یکی از شمارنده ها به انتها رسید پس هنوز شمارنده دیگه عناصری داره که نیامده رو S اول

چک می کنیم کدوم اول تموم شده اگه n > i یعنی سمت چپ به انتهاش رسیده پس کافی است بقیه پایین بریزیم

صورت M-j شده و همه پایین . پس تو ماتریس V عناصر باقی مانده ریخته میشه جلوی خانه S که میشه k اش

پیشرفته پس $m+n$ صفر میشه. اگه برعکس بشه $m > n$ پس آرایه مقابله باقی مونده پس از ادغام با

اضافه میره تو باقی مانده ماتریس S قرار میگیره.

پیچیدگی زمانی Merge

ما توی این الگوریتم چل اولی؟ مقایسه هست. مقایسه من توی تابع Merge اتفاق میافته که باید هی مقایسه کنم حالا سوال اینه

آرایه U، n عنصر و V، m عنصر داره توی این Merge چند بار اتفاق میافته؟ با هر مقایسه یه عنصر یا از چپ یا از راست

میره پایین ولی نمیشه یه عدد ثابت گفت چون نمیدونیم از کدوم ورمیده پایین پس بدترین و بهترین حالت رو فرض

کنید همه عناصر سمت چپ کوچیکتر از عنصر اول سمت راستی باشن پس همه از سمت چپ میرن پایین و ده تایی

چپ میرن پایین و سمت راستی هم میره پایین پس در بهترین حالت 10 تا مقایسه نیاز داشتم

توی بدترین حالت یعنی شمارنده ها دیرتر به انتها برسن تا مقایسه ها کمتر باشه پس از یکی از چپی یکی از راستی

بره پایین یکی درمیون اینجوری 10 تا از چپی و 9 تا از طرف راستی پس یکی عقب مونده و آخری هم رو نفر

آخر مونده پس میشه 19. به ازای جمع دو تا آرایه ها مقایسه صورت میگیره غیر از آخری که میشه 19. چون

نمیتونه همزمان خاتمه بدن پس 20 نقطه. پس تعداد مقایسه ها تو این تابع چه خیلی بیشتر و چه کمتر و حداکثر

فقط میشه $m+n-1$ و بدترین اش میشه m یا n یعنی نصف اش.

اول بدترین حالت: T لعنت اینجا به جای W. برای حل n میشه حل پیچیدگی زمانی برای سمت

چپ و راست n و m + پیچیدگی ادغام یعنی مقایسه های سمت چپ n و راست m و یه مقدار برای

ادغام کردن مقایسه میخواد که میشه $n+m-1$ که میشه $n-1$ پس چون n و m تقریبا $\frac{n}{2}$ هست میشه $2T(\frac{n}{2})$

subject :

به علاوه $n-1$ که تقسیم و حل هست که تقسیم و حل همیشه بازگشتی بدست میاد. و $T(n)$ هم میشه که نیاز به مقایسه کردن نداریم و میشه با قضیه اصلی (اگه ۱- رو در نظر نگیری) هم میشه با جایگذاری حل کرد.

برای بهترین حالت: این مقایسه برای ادغام چپ و راست کمترین متعادل باشه یعنی کلا یه ورژن آرایه باشه و اول نیکی کمی میشه که میشه $n/2$ و خودش بدست میاد. پس تو حالت متوسط هم $n\log n$ هست که از بهترین و بدترین حالت که $n\log n$ بود نتیجه گرفتیم و مقایسه سازی اضافی با استفاده از لیست تقسیم و حل هست و tail نیست!

پس پشت به جایگاه و تقدیمی برتر هست چون اول n^2 بود. (تقدیم اصلی ۲ اصلی هست و ایا را باید نشان بدهد توجه)

نکته: in-place یعنی درجا. الگوریتمی درجا هست که برای مرتب کردن یک آرایه کلا همه روی همون آرایه انجام بده و آرایه اضافی نگیره. یا اینکه دو تا آرایه جدید U و V تعریف کردیم و علاوه بر S دو تا آرایه U و V که اندازه شون اندازه S بود هم تعریف کردیم و اگه درجا بود حافظه خیلی کمتری مصرف میکرده. توی کتاب درجاش هم گفته.

همه الگوریتم های مکی، بازیری و تقدیمی و جایگاه همه in-place بود.

سوال کنکور ← تعداد عمل Merge چند تا هست؟ اگه n عضو داشته باشیم $n-1$ عمل Merge و گفتیم M و d داریم اگه بیاریم لا تا از هر کدوم هست پس با هم برابر هستن و تعداد هر دو از تعداد عملیات نصف یکی کمتره.

اگه میخواستیم اینایی حل کنیم فقط کنیم $n \le 2^k$ میخوام بیام چند M یا d داره. تو مرحله آخر هم یکی یکی میشن و ادغام میشن و دوتایی میره چند تا تکی؟ 2^k و نصف اونا که ادغام میشن میشه 2^{k-1} تو مرحله بعد دوباره نصف 2^{k-1} میشه 2^{k-2} و بعد دوباره تا برسه 2^0 حالا اینا رو جمع نوشته که مرحله آخر یه جمع میخواد فقط که مقدار ثابت است که میشه 2^k-1 که همون $n-1$ هست.

الگوریتم QS

پیاده‌کد: یه تابع نوشته به نام QuickS و دو متغیر به نام L و H میگیره یه آرایه سراسری به اسم S و L و H اندیس

های محدوده‌ای که میخواد QuickSort کنه یعنی کل این محدوده رو بیاد QuickS کنه، تو ابتدا L باشه اندیس اول

H باشه اندیس آخر و حق به زیرآرایه‌هایی هم بدون اول کوچکتره pivot می‌ده از نوع int، اندیس محل که عنصر محوری قرار

میگیره بعد از عمل پارتیشن، میگه اگه L<H باشه که همیشه همینه یا اصلا مساوی (یه عنصر) باشه بیا این کارو انجام بده یعنی

اول عمل پارتیشن رو تو محدوده L و H انجام بده و عنصر محوری رو بذار جایی که باید باشه و نتیجه این رو تو piv بگذار دوست.

و اینکه خودش تابع part چطوری هست رو بعدش میگه، مثلا اینو قبول میکنیم و حالا محدوده اندیس‌های زیرآرایه چپ و راست

پیدا میشه چپ میشه از L تا یه عدد قبل نقطه شکست و همین از یه عدد بعد از نقطه شکست تا H و این میشه کوئیک

سورت برای هر کدوم از زیرآرایه‌ها. که اینم بازگشتی شد چون اومده تقسیم و حل بوده.

خود تابع partition: مثل پیاده سازی به صورت تنگی توضیح اش دادم، عنصر محوری میذاره اول 15، و نیک عنصر شروع

از دو میگیره بعد عنصر اول یعنی اندیس عنصر دوم. هر بار چک میکنه عددی که اشغال نشده کوچکتر هست یا محوری

اگه عددی که اشغال شده کوچکتر باشه پس خودش رو اضافه میکنه به جای بعدی اون رو عوض میکنه، بعد اون

از روی این داده اشاره میکنه کوچکتر هارو میریزه سمت چپ، و یه خ اضافه میکنه و بعد تو اشغال میکنه

عنصر محوری که عدد وسط هست و خ رو میگیره عدد شروع و از دو میگیره بعد عنصر اول. هر بار چک میکنه عددی که از

pivio کوچکتر عنصر اول

نشغل میده بزرگتر یا عنصر محوری. اگه عددی که اشغال شده کوچکتر باشه، خ رو اضافه میکنه یه دو تا جای اون

عوض میشه چون 22 کوچکتر هست میره بعدی یعنی 15 پس جاشا میشه اول و زیده جلو بیاد

pivot عنصر اول ، x رو عنصر اول ۱ و از عنصر دوم تا n-1 و هر بار عنصر i رو با عنصر اولی مقایسه و اگه کوچیکتر باشد x رو اضافه و جاشون رو عوض میکنه حتی این ۸۷ تم شد ، یعنی عنصر اولی رو با عنصر x عوض میکنه تا عنصر اولی سر جای خودش برسه. و در انتقال x اندیس اولی رو نمیدونیم پس میدیم تو pivot که بعدش اندیس اولی کجا قرار گرفته.

(از نوع inplace هست چون که با همون اسم هاش نوشته میشه ۱ مرتبه partition و فقط باید یکی بزرگتر از عنصر اولی باشه و اینکه به برگردونده pivot رو چون میخواد با عناصر سمت چپ و راست مقایسه کنه)

مثال زمانی

مثال اصلی: مقایسه ۰ پس تعداد مقایسه ها رو میشماریم ۰ سه مرحله بود: ۱- عنصر اولی ۲- پارتیشن کردن ۳- حل زیرآرایه های سمت چپ و راست ۰ هزینه هر کدوم چقدره؟ x عنصر اول عنصر اولی پس مقایسه ای نمیخواد ، پارتیشن کردن چی؟

خط ۸۷ مقایسه اتفاق میافته از عنصر دوم تا آخر میره پس اگه n عنصر داشته باشیم این خط n-1 بار اجرا میشه ۰ مرحله ۳ زیرآرایه ها حل شد ، اگه من T(n) برای تعداد مقایسات یه آرایه n تایی بارش برای سمت چپ میشه تعداد مقایسه عناصر سمت راست و چپ ۰ T(n-1) سمت راست + T تعداد عناصر سمت چپ = T(n)

برای پارتیشن

تعداد عناصر سمت چپ و راست چی میشن؟ ما نمیدونیم بستگی به عنصر اولی داره میتونه هر جایی باشه پس این جا بدترین و بهترین حالت اتفاق میافته ۰ هر خانه دو تا زیرمسئله میشه دیگه یعنی ایده اش اینه میتونه به جای دو تا چهار تایی یا نه تایی یا نه دو تایی شکسته شه این چطوری باشه که عمق زیاد یا نامتوازن ترین حالت رو داشته باشه؟ یعنی هر چی بیافته یکی از زیرآرایه ها یا چپ یا راست پس بدترین حالت زمانی هست که یکی از این دو آرایه بعد پارتیشن یکی شون تهی بشه و بقیه تقریبا نمی باشه حالا یکی ۰ عنصر و بقیه اش تو بقیه قسمت کنیم ۰ n-1 + T(n-1) + ۰

$T(n-1)$ چون زمانی که مرتب است و به صورتی میشه $T(n-1) = T(n-1)$ که این به روش جایگذاری

راحت قابل دیدن هست پیچیدگی n^2 درمیاد و تو بدترین حالت پیچیدگی وجابی محل میکنه.

ولی تو حالت متوسط و بهترین حالت چی؟ تو بهترین حالت $T(n-2) + (n-2) + (n-1)$ &

و متوسط رو هم ازش میگذریم ولی در حالت معادله دیگه و بین بدترین پیچیدگی میکنه و خیلی آخرش

$A(n) = 1.38(n+1)\log n + 1$ یا همون $T(n)$. چون تو متوسط با ارتفاعی یکی ولی سریع

تو بدترین حالت n^2 بود و همین بود ولی تو حالت متوسط برای حقیقی زمانی زیاد $n\log n$ هست و خوب

بهترین $\rightarrow T(n) = T(n-\frac{1}{2}) + T(n-\frac{1}{2}) + \Theta(n-1) \rightarrow n\log n$

متوسط $\rightarrow$ میگذریم $\rightarrow A(n) = 1.38(n+1)\log n + 1$ هست این خیلی کوچکه و

ولی اندازه دقیق رفتار از روی تابع زمانی بدست میاد و مرتب فقط ارتفاع تابع رو نشان میدهد.

(و اینکه تو حالت متوسط برابر با ارتفاعی هست)

نکته: تو طراحی الگوریتم ها از روش تقسیم و حل هم با استفاده نکنید که تابع زمانی این به شکل:

۲- جایی که تعداد اولیه رو به تعدادی تعداد کوچیک می شکنیم که خود این تعدادها نیمی نباشد، n باشد

تا اینجا به دو تا تعداد شکسته شد که عدد نیمی هست و خوبه ولی اگه به n تا زیرمسئله کوچیک تقسیم بشه و

$n^{\log n}$ دیگه فایده نداره. (یعنی تابعی از n باشد این تعداد اولیه)

۱- تعداد بزرگتر رو به تعداد خیلی بشکنید دو یا چند تعداد که این ها هم تقریباً n باشد

۲- اینجا حدوداً نصف بود و می میشکوندیم زیر آرایه ها رو. که مثالش فیبوناچی هست

(اندازه مسئله n فوق تعداد یا تعداد مسئله قبلی n باشد)

غلبه بر شکست حالت

برنامه نویسی پویا

همانطور که گفتیم یکی از ایرادات اصلی روش تقسیم و حل، به الگوریتم غیر کارا منجر میشد و مرتبه زمانی بالا به دست می‌آمد و بالا رفتن که گفتیم اگر نمونه بزرگ به تعداد زیادی نمونه کوچک مثلا به n تا شکسته شد و یا خیلی با اندازه اصلی فرق نکند در این صورت منجر به الگوریتم غیر کارا میشد و روش برنامه نویسی پویا میتوند این مشکل را حل کند. مسئله مشکل اصلی این حالاتی که گفتیم تقسیم و حل جواب نمیده چیه؟ ما وقتی تقسیم و حل به مشکل می‌خوریم که زیر نمونه‌های کوچک به تعداد تکراری منجر میشه و این منجر به غیر کارا شدن میشه ولی تو پویا ایده این هست نمونه کوچک حل شده رو ذخیره و نگه داری بشه که تو مراحل بعدی اگر نیازش داشتیم لازم نباشه محاسبه بکنیم و صرفا از نتیجه‌ای که محاسبه کردیم استفاده کنیم.

مثال: فرض کنید می‌خواهیم n امین عدد فیبوناچی رو با رابطه پایین حساب کنیم تقسیم و حل یعنی گفتیم چطوری رو پشت سر هم یعنی f10 اول f9 و f8 و بعد با هم جمع کنیم خود f8 و f9 باید مجدد حساب رو دوباره بیاریم. یعنی 20 بار f7 و دوبار f6 و 30 بار f5 یعنی چند بار محاسبه میشه و همین الگوریتم رو غیر کارا میکنه و تکراری هست و درخت رو با یک حجم عظیم از محاسباتش غیر تکراری هست و همینه که لوند غیر کارا میکنه. حالا این رو چی محاسبه کردیم یه جا بنویسیم اگه خواستیم دوباره محاسبه کنیم لیست کار و انجام ندیم و از نتیجه لیست استفاده کنیم یعنی استفاده از یه حافظه موقت برای نگهداری نتایج حل زیرمسئله‌های کوچکتر تو تقسیم و حل که اثر بالا به پایین بود مشکل ایجاد شد.

اگر بخوایم به صورت رسمی تو برنامه نویسی پویا رو بگیم تو برنامه نویسی پویا مثل تقسیم و حل به ایده بازگشتی جلو رفته و حل میکنند اولی با استفاده از حل زیرمسئله‌های کوچکتر ولی تفاوت اینه که این روش حل بازگشتی رو به صورت جزء به کل میبریم جلو

مثال ۴ (اصلاح شود $C_n^1 9^{n-1}$)

هدف محاسبه C_K^n مثل C_{20}^4 که K و n بین C_K^n رو محاسبه کنیم.

به هر دو حالت C_n^K نوشته می‌شود.

محاسبه این از فرمول آماری جزوه سخته ولی چرا؟ باید for و تو بیشتر مرکبم از n! و K! هارو محاسبه کنیم اشکالش اینه که

اگه n و K عدد بزرگی باشن چون K<n باشه از ۰ انجام نمیشه. اینی هیچ اتفاقی نمیکرد اگه n بزرگ باشه

n! خیلی بزرگ میشه درست تقسیم میشه ولی برای بدست آوردن خیلی عدد بزرگی میشه و می‌تونه تایپ های ما

هم جا بیفتاده و خیلی محاسبه اش دشواره (نکته فوقش تو جزوه هست)

تو فرمول $C_K^n = C_K^{n-1} + C_{K-1}^{n-1}$ هر دو تا از عددها اصلی C_K^n کوچیکتر هستن و اگه K=n یا K=0 باشه

یعنی تموم میشه یعنی مثلا به چند طریق از ۶ نفر میتیم ۶ نفره؟ یه روش یا به چند طریق از ۶ نفر میتیم ۰ نفره؟ یه راه

دیگه که این دو حالت کمکی پایانی این رابطه بازگشتی هست.

اینجا حالت اول یعنی خانه‌های کوچیکه خیلی فرقی با عددهای بزرگتر نداره اون یه ۱ اضافه شده خیلی فرق

نداره حالت دوم نیست چون فقط دو تا عدد کوچیک ایجاد شد نه n تا.

اون توصیه‌ای بود که اتفاقی فعال کردیم که اگه درخت بازگشتی اش رو بکشیم و ببینیم که کلی محاسبات تکراری پیدا می‌کنیم

باید مثال کوچیکی اینو نشون بدم و گفتم اگه بخوام از این روش کدکوندا استفاده کنم اللازم به صورت خوره چون

محاسبات تکراری میشه ($C_4^6 + C_3^6 = C_4^7$) که درخت اش رو نوشتیم که تا پایین هم نمیره برای

عددها می‌نویسیم. و C_4^9 که میشه ۱ نیازی به اطلاعات نداره یا ۰ بشه که هم شرط پیداره.

subject : date :

کلا هست عدد وسط این درخت تکراری میشه تا وقتی هاش نه وقتی وسط.

مثلا C_{50}^{100} که یه درخت خیلی بزرگ میشه و سطح زیادی خواهد داشت چند تا سطح میره پایین تر؟ 99 یعنی

کلا 100 تا و 2^{100} میشه.

$i=5 \quad j=2$

و خوام C_4^7 رو محاسبه کنیم با استفاده از برنامه نویسی پویا به جای تقسیم حل و این آرایه رو هم برای

نگهداری نتایج حل زیرمسئله ها با هم جمع جلو کنیم C_j^i رو و نتیجه این رو تو B_j^i قرار میدیم.

و با پایتون برای این آرایه تبدیل میشه به اینکه هر خونه خونه بالای و بالای سمت چپ رو

با هم جمع کنیم دربیاد.

C_4^7 که گفتم یه آرایه دوبعدی که سطر و ستون تا n باشه رو بلند بایم و این باشه تو B

یا مثل ذخیره سازی داده ها مون هست. ستون K و سطرش n داره.

و اون $\square$ C_4^7 رو به ما نشون نمیده ما آرایه خالی رو نشان میدیم این پر شده است.

فرض کنید خانه جدید سطر و 7 ستون داره و بعد C_4^7 میخوام

با این رابطه $B[i][j]$ جدید تو خانه ای که $j=0$ $B(i)(j)=1$ هست یعنی

چی؟ این چرا کجا ها 1 بذارم؟ $j=0$ یعنی ستون 0 ام کلا میشه 1.

و وقتی گفته وقتی $j=i$ هست سلول = 1 میشه یعنی پایه 0 و 1 و 2 و ... باشه

یعنی قطر جدول 0 پس تا حالا ستون 0 ام و قطر اش پر شده و 1 شده (قطر اصلی یعنی)

الان بالای قطر اصلی رو کار داریم یا نداریم؟ چرا؟ چون که مثلا نقطه C_2^4 که شده 6 و نقطه C_4^2

به چند طریق میشه از n شیء k شیء انتخاب کرد؟ 0 ! یعنی k بزرگتر از n است یعنی این نقطه

همیشه k حاصل از n حاصل بزرگتره و ما کاری عملش نداریم چون k از n بزرگتر

باشد اولا نیازی به محاسبه نداریم و 0 هست خودش ! اولین جایی که باید محاسبه کنیم کجاست

خالی بودنه باید بریم سراغش؟ ستون 1 و سطر 2 و بقیه پایین این 1 هاست که کاری

باهاش نداریم. حالا ستون 1 و سطر 2 رو میتونیم محاسباش کنیم؟ به جز بالایی و پایین سمت چپ

نیاز داشتیم. بالایی میشه 1 که مثلا بلدیم و 1 که اینم بلدیم 1+1 که میشه 2. منظورمه؟

از بالا بیاید سمت پایین. ستون 1 سطر 3 که اینم میشه محاسبه کرده. دقت کنید ستون 2 و

سطر 3 که اینم 3 میشه حالا سطر 4. سطر 4 ستون 1 اینم میشه 4 بعد میریم سراغ 6 و بقیه

4 و اینم 1 حالا سطر بعدی. ما دنبال C_4^7 بودیم وقتی اینکارو تا سطر 4 انجام بدیم جواب

بدست میاد. یعنی هم C_4^7 و هم C_3^6 رو مثلا بدست میارم همه ی خونه های کوچیکه رو بلدیم

از بالا به سمت پایین حرکت کردیم طبق قوانین بالایی ما خونه های کوچیکه ی جدول

منطقی هم نقشه جدول رعایت شده. از پایین نمیشه برگرد چون بلد نیستیم.

اینجا یعنی برنامه نویسی پویا. یعنی ذخیره سازی نتایج کوچیکه تو یه آرایه و با استفاده از یه

رابطه بازگشتی از خونه کوچیکه شروع کردن به خونه بزرگتر حرکت کردن. در نهایت رسیدیم به

خونه آخر. تو این مثال من C_4^7 رو خواستم آیا خونه 15 یعنی ستون 4 سطر 6 استفاده شد برای

محاسبه این یا نه؟ بله. 15 برای C_4^7 آیا خونه سطر 4 ستون 2 به درد خورد یا نه؟ بله چون

این عنصر رو نیاز داشتیم تا دوتم غلبه شدیم اول بتونیم پایین ها غلبه بشن و دوتم غلبه بشن

بله به صورت مستقیم نه ولی به صورت غیر مستقیم نیاز بود دیگه . از همون ستون 1 سطر 2

هی استفاده شد؟ بله مثل مثل بدست . پس نکته این عناصر قبل ستون 4 باید بلد بودن

تا به 4 کا برسیم ولی نیازی به عناصر قطر اصلی نداشتیم؟ نه چون به عمل کردن و سمت چپ

نیاز نداشتیم اونطور نیاز نداشتیم پس الان به عضوی داریم ستون 4 هست پس اولا ستون های دیگه

5 و 6 و 7 نیازی نداریم و از سطر 4 ام به بعد و فقط 0 تا 4 را محاسبه می کنیم و این

خط ستون هیچ به دیگری ها نیاز نداریم واسه همین مدرج اکثر دو تا رو

راحت تر و سریع تر می کنیم .

(شبیه کد) ← فرض کنیم ام n تا باشه

الان C_k^n . این n یا k کوچکتره یه اکلید B میاد سطر و ستون و بعد از k رو دو نظر می گیره

و از k تا 0 هست ستون هاش با اینکه سطر n تا 0 هست .

i = 0 تا n لول سطر 0 بده 0.0.0.1 و j هم از صفر تا میتیم از 0 تا i می تونی سطر 0 ام

تا ستون 0 هر رقم توسط 1 ؟ ستون 1 و 0.0.0 و از 5 و 6 و 7 از جمله سطر

4 بیشتر نرفتیم این همونه منظورش . اول فرض رو چک می کنیم اگر j = i باشد یا روی

قطر اصلی باشد 1 میشه در غیر این صورت طبق فرمول هستی عضو بالا راست و سمت چپ پایین

میاد که پس از سطر 0 ام شروع کرده و میره جلو پس همه چیز فرمول داشت بعد این تعداد

CamScanner

subject :

date :

میره توی سطر و ستون و اونارو Return میکنه و برمیگردونه و برامون محاسبه اش میکنه.

(پیچیدگی زمانی)

تو این الگوریتم عمل اصلی؟ جمع حالا باید پیچیدگی زمانی رو محاسبه کنیم به ازای K و n و تعداد عضوهارو هم

به ما میده. پس چند بار عمل اصلی انجام میشه برابر با اینکه چند بار حلقه for انجام میشه و تعداد عضوهارو

هم تعیین کن. دو زنتو به ما میده و برابر با تعداد دفعاتی هست که این حلقه انجام میشه.

پس برای اینکه محاسبه ساده بشه لوفو گرفتم بلاک اصلی (عمل جمع رو)

یعنی هم فرمول زمانی دقیقش هم فرمول تقریبی که محاسبه میشه

اصلاح $T(n,k)$ چون هم n و K دخالت دارند و اعداد رو مشخص میکنه.

پارامتر $i=0$ یه دونه عضو حساب میکنه بار بالا $i=1$ دو عضو حساب میکنه حلقه داخلی دو بار

اجرا میشه بعد که $i=2$ حلقه داخلی ۳ بار اجرا میشه و $i=4$ حلقه داخلی 4 بار محاسبه میشه.

و بلاک if اجرا میشه تا $i=K-1$ که میشه K

و از سطر K ام به بعد یعنی 4, 5, 6, 7 از 0 تا K یعنی K+1 عضو محاسبه

میشه و ستون 0 تا K میشه K+1 یعنی: بیشتر از K میشه و ...

تا K طبق فرمول حل میشه $\frac{K(K+1)}{2}$

و تا $i=n$ هم K+1 داریم از K تا n چند تا میشه؟ (مثلا $n=7$ برای $i=5$ و ...

میشه $n-K+1$ چون خود K هم باید حساب کنیم پس چند تا K+1 داریم. این بزرگترین جمله K^2 هست

$K+1$ ؟