Week 05 Questions

سوالات تکالیف هفته ۰۵

سوال اول : خواندن داستان های زیر و تبدیل آن به درخت تصمیم (Decision Tree)

داستان های زیر را خوانده و درخت تصمیم (Decision Tree) آنها را رسم کنید

الف) ساعت 1:30 بعد از ظهر شده است و علی پشت میز اش نشسته است و به صفحه مانیتور نگاه میکند و چشمانش سنگین شده است و تمرکز اش کم شده و بازدهی او در کارش کم شده و افت کرده و یهویی یه سوالی ذهنش رو دگیر میکنه :

آیا خسته‌ام؟

بله ← چرا؟

دیشب خوب نخوابیده‌ام ← قهوه بنوش

مدت زیادی بدون وقفه کار کرده‌ام ← یک استراحت کوتاه

خیر ← پس چرا حواسم پرت است؟

حوصله‌ام سر رفته ← کارم را عوض کنم

بدنم کم‌آب است ← آب بنوشم

ب) سارا یک دانشجو مهندسی کامپیوتر است که میخواهد برای پروژه نهایی خود تصمیم بگیرد و استادش به او می گوید که تصمیم های هاش رو مثل یک درخت ببیند و سارا روی کاغذ نوشت:

سؤال اول:

آیا پروژه باید **کاربردی** باشد؟

اگر بله ← به سؤال بعدی برو

اگر خیر ← پروژه‌ی تئوریک انتخاب کن

سؤال دوم:

آیا پروژه باید مبتنی بر داده باشد؟

اگر بله ← سراغ یادگیری ماشین برو

اگر خیر ← یک سیستم نرم‌افزاری طراحی کن

سؤال سوم:

آیا داده‌ها برچسب‌دار هستند؟

اگر بله ← از الگوریتم‌های نظارت‌شده استفاده کن

اگر خیر ← خوشه‌بندی را انتخاب کن

ج) سارا یک دانشجوی مهندسی کامپیوتر و در حال طراحی یک سیستم هوشمند برای پذیرش وام بانکی است. سیستم باید بر اساس اطلاعات متقاضی تصمیم بگیرد که وام داده شود یا نه و او سه ویژگی را برای سیستم تعیین میکند :

۱. درآمد ماهانه

۲. سابقه اعتباری

۳. وضعیت شغلی

و حالا سیستمی که برای پذیرش یا رد وام بانکی طراحی شده :

آیا درآمد ماهانه متقاضی حداقل تعیین‌شده است؟

بله ← برو به سؤال ۲

خیر ← رد وام (پایان)

سؤال ۲:

آیا سابقه اعتباری متقاضی خوب است؟

بله → تأیید وام

خیر ← برو به سؤال ۳

سؤال ۳:

آیا متقاضی شغل ثابت دارد؟

بله ← بررسی بیشتر (پایان)

خیر ← رد وام (پایان)

سوال دوم : پروتکل تریاژ: داستانی از ناو فضایی اتلگارد

بدنه ناو سنگین *اِتِلگارد* زیر رگبار کمربند سیارکی به شدت می‌لرزید. در عرشه فرماندهی، چراغ‌های قرمز با ریتمی آشفته چشمک می‌زدند و ترمینال ثبت خطاها با سرعتی سرگیجه‌آور پر از متن می‌شد.

کاپیتان ونس دسته‌های صندلی‌اش را محکم گرفت و فریاد زد: «گزارش بده! اول چی رو تعمیر کنیم؟»

ناوبان کِل با وحشت در حال بالا و پایین کردن لیست هشدارهای روی صفحه بود: «نمی‌دونم قربان! ما ۵۰۰۰ تا هشدار داریم! "خرابی سیستم پشتیبانی حیات" جایی بین "خرابی تستر نان در آشپزخانه" و "داغ کردن سپر محافظ" گم شده. این لیست کاملاً نامرتب و به‌هم‌ریخته است!»

برای پیدا کردن بدترین مشکل، کِل داشت لیست را به صورت خطی (یکی‌یکی) بررسی می‌کرد. برای Nهشدار، پیدا کردن بالاترین اولویت $O(N)$زمان می‌برد. برای رفع همه آن‌ها به ترتیب، او به زمانی معادل $O(N^2)$نیاز داشت. کشتی فضایی خیلی پیش از آنکه کار او تمام شود، منفجر می‌شد.

کاپیتان دستور داد: «سیستم P.Q (صف اولویت) را فعال کن. تمام هشدارها را بفرست داخل هیپ (Heap).»

تجسم هیپ (The Heap)

ناگهان، لیست متنی که دیوانه‌وار حرکت می‌کرد ناپدید شد. به جای آن، یک ساختار درختی سه بعدی و درخشان ظاهر شد. این همان صف اولویت بود که با ساختار Max-Heap (هیپ ماکزیمم) پیاده‌سازی شده بود.

به جای یک لیست تخت، هشدارها درون گره‌هایی شناور بودند که با پرتوهای نور به هم متصل می‌شدند.

- قانون هیپ: سیستم هر هشدار ورودی را می‌گرفت و درون درخت می‌انداخت. آن‌ها بلافاصله بر اساس یک عدد صحیح به نام «سطح خطر» مرتب می‌شدند. قانون مطلق بود: اولویت یک گره پدر همیشه باید بیشتر از فرزندانش باشد.

- غربال رو به بالا (Sift Up): وقتی گزارش‌های خرابی جدید می‌رسیدند، از پایین درخت وارد می‌شدند و اگر خطرناک بودند، مثل حباب به سمت بالا می‌رفتند و جای خود را با مسائل کم‌اهمیت‌تر عوض می‌کردند تا به سطح مناسب خود برسند

کشتی *اِتِلگارد* از کمربند سیارکی جان سالم به در برد، اما یک مشکل ظریف در جلسه بررسی پس از عملیات مطرح شد.

فرض کنید دو سیستم مجزا دقیقاً در یک زمان و با سطح اولویت کاملاً یکسان دچار خرابی شدند.

- **هشدار الف:** «خرابی پشتیبانی حیات طبقه ۴» (اولویت: ۸۰) — زمان دریافت: ۱۲:۰۰:۰۱
- **هشدار ب:** «خرابی پشتیبانی حیات طبقه ۵» (اولویت: ۸۰) — زمان دریافت: ۱۲:۰۰:۰۵

در پیاده‌سازی استاندارد Heap Sort، ترتیب نسبی عناصری که کلیدهای برابر دارند **تضمین نمی‌شود**. این یعنی الگوریتم ناپایدار (Unstable) است. هیپ ممکن است آن‌ها را طوری جابجا کند که طبقه ۵ زودتر از طبقه ۴ تعمیر شود، صرفاً بر اساس نحوه حرکت آن‌ها در ساختار درخت طی فرآیند غربالگری.

سوال:

شما میخواهید به عنوان طراح نسخه بعد سیستم عامل اتلگارد باشید و این سیستم را طوری طراحی کنید که :

- هشدارهای با اولویت برابر حتماً به ترتیب **ورود-اول-خروج-اول (FIFO)** حل شوند (یعنی هشدار الف *باید* قبل از هشدار ب پردازش شود)
- جوری داده های ذخیره شده در صف اولویت (Priority Queue) را تغییر دهید که به **پایداری** برسد
- پیچدگی زمانی O(nlogn) خراب نشود

سوال سوم : میانه جریان

در شهر هوشمند **داده‌سرا**، هر ثانیه هزاران عدد از حسگرها وارد یک جریان بی‌پایان می‌شد. این اعداد نشان‌دهنده‌ی وضعیت لحظه‌ای شهر بودند: دما، فشار، مصرف انرژی.

مدیران شهر فقط به یک چیز اهمیت می‌دادند: **میانه (Median)**.

نه بزرگ‌ترین عدد مهم بود، نه کوچک‌ترین؛ بلکه عددی که تعادل را حفظ می‌کرد.

در ابتدا، مهندسان همه‌ی داده‌ها را جمع می‌کردند، با **Heap Sort** مرتب می‌کردند و عنصر وسط را برمی‌داشتند. اما خیلی زود فهمیدند که این روش کند است؛ جریان داده متوقف نمی‌شد تا آن‌ها مرتب‌سازی را تمام کنند , در این میان، **سارا** یک مهندس الگوریتم راه‌حلی پیشنهاد داد:

«چرا کل داده‌ها را مرتب کنیم؟ بیایید آن‌ها را به دو نیمه تقسیم کنیم.»

او دو ساختار ساخت:

- یک **Max-Heap** برای نیمه‌ی کوچک‌تر اعداد
- یک **Min-Heap** برای نیمه‌ی بزرگ‌تر اعداد

هر عدد جدید که وارد می‌شد:

- اگر از بیشینه‌ی نیمه‌ی پایین کوچک‌تر بود، به Max-Heap می‌رفت؛
- وگرنه وارد Min-Heap می‌شد.

سپس، اگر یکی از heapها بیش از حد بزرگ می‌شد، سارا تعادل را با جابه‌جایی یک عنصر برقرار می‌کرد.

حالا میانه همیشه آماده بود:

یا رأس یکی از heapها، یا میانگین دو رأس, بدون مرتب‌سازی کامل سریع پایدار

اما ناگهان قانون شهر عوض شد:

از این پس فقط **k داده‌ی آخر** اهمیت داشتند. با ورود هر عدد جدید، قدیمی‌ترین عدد باید حذف می‌شد.

سارا مکث کرد.

درج در heap آسان بود، اما **حذف یک عنصر دلخواه** از heap، ساده و سریع نبود.

heapها اعدادی را نگه می‌داشتند که دیگر جزو جریان معتبر نبودند.

او به صفحه‌ی داده‌ها خیره شد و با خود گفت:

«اگر نتوانم حذف را به‌صرفه انجام دهم، تمام این تعادل از بین می‌رود...»

وظیفه شما :

با پایتون برای کمک به سارا برنامه ای بنویسید برای محاسبه میانه در یک جریان داده با پنجره لغزان (Sliding window) ، می توان ساختار دو heap را طوری باز طراحی کرد که درج و حذف هر دو در زمان نزدیک به $O(\log k)$ انجام شوند.

سوال چهارم : تورنمنت خرده بلورها

در پادشاهی دیتاون، هر سال یک تورنمنت بزرگ برگزار می‌شد تا سریع‌ترین مرتب‌ساز سرزمین مشخص شود.

این رقابت درباره‌ی قدرت نبود، بلکه درباره‌ی استراتژی بود و امسال چالش بسیار سختی پیشِ‌رو بود: مرتب‌سازی کوهی از خرده‌بلورهای کریستالی، که هرکدام اندازه‌ای منحصربه‌فرد داشتند.

در میان شرکت‌کنندگان لیرا، جادوگر جوان الگوریتم‌ها، با زیرکی‌اش شناخته می‌شد. در حالی که دیگران در حال ساخت ماشین‌های عظیم مرتب‌سازی بودند، لیرا آرام دایره‌ای روی زمین کشید و گفت:

«چرا همه‌چیز را یک‌جا مرتب کنیم، وقتی می‌توان از خودِ بی‌نظمی برای رسیدن به نظم استفاده کرد؟»

او یک بلور را انتخاب کرد نه بزرگ‌ترین و نه کوچک‌ترین و آن را در مرکز قرار داد.

سپس جادویش را فعال کرد: تمام بلورهای کوچک‌تر به سمت چپ شناور شدند و تمام بلورهای بزرگ‌تر به سمت راست رفتند.

هر سمت به کوهی کوچک‌تر تبدیل شد، و لیرا همین فرایند را بارها و بارها تکرار کرد؛ تقسیم، سپس مرتب‌سازی، تا جایی که همه‌ی بلورها به شکلی کاملاً صعودی کنار هم ایستادند.

جمعیت شگفت‌زده شد؛ در زمانی هم‌مرتبه با O(n log n) ، کوه بی‌نظم به هرمی درخشان از نظم تبدیل شد. داوران سر تکان دادند لیرا از افسانه‌ی کوییک‌سورت (Quick Sort) استفاده کرده بود؛ هنر «تقسیم و حل».

اما ناگهان خردمندترین داور پرسید:

«لیرا، اگر همیشه بزرگ‌ترین بلور را به‌عنوان محور (pivot) انتخاب می‌کردی، چه سرنوشتی در انتظار جادویت بود؟»لیرا مکث کرد. اگر هر بار بزرگ‌ترین عنصر انتخاب می‌شد، در هر مرحله یک بخش شامل (n-1) عنصر و بخش دیگر تهی می‌ماند؛ عمق بازگشت (recursion) افزایش می‌یافت و الگوریتم به‌طرز دردناکی کند می‌شد.

زیبایی الگوریتمش به جای O(n log n) به هرج‌ومرج $O(n^2)$ سقوط می‌کرد.

داور لبخند زد و آرام گفت:

«و حالا سؤال واقعی این است اگر اجازه‌ی استفاده از تصادفی‌سازی در انتخاب محور را نداشتی،

اما مجبور بودی حتی برای داده‌های تقریباً مرتب‌شده نیز کارایی الگوریتم را تضمین کنی...

چه راهبردی برای انتخاب محور طراحی می‌کردی؟»

وظیفه شما:

یک الگوریتم پایتون خالص طراحی کنید که روش انتخاب محور در Quick Sort را بهبود دهد، به‌گونه‌ای که روی آرایه‌های مرتب یا تقریباً مرتب، دچار حالت بدترین حالت نشود.

الگوریتم شما باید:

1. فقط از پایتون خالص استفاده کند (بدون import)

2. از یک روش قطعی (deterministic) برای انتخاب محور بهره ببرد که از رفتار بدترین حالت جلوگیری کند

3. پیچیدگی زمانی متوسط را نزدیک به O(n log n) نگه دارد

سوال پنجم : زنجیره اعداد و قاضی سریع

در شهر **نودهِیوِن (Nodehaven)**، اعداد مثل آرایه‌ها منظم و کنار هم زندگی نمی‌کردند.

آن‌ها به‌صورت یک **زنجیره** بودند؛ هر عدد فقط دستِ عدد بعدی را گرفته بود.

شکستن زنجیره ممنوع بود، اما **تغییر اتصال‌ها مجاز** بود.

روزی از قاضیِ سریع، **کوئیک (Quick)**، خواستند تا زنجیره‌ای آشفته را مرتب کند:

$5 \rightarrow 1 \rightarrow 4 \rightarrow 9 \rightarrow 2 \rightarrow 7$

کوئیک نمی‌توانست به‌دلخواه به وسط زنجیره بپرد؛

او مجبور بود **گره‌به‌گره** حرکت کند.

پس تصمیم گرفت **اولین گره را به‌عنوان Pivot (محور)** انتخاب کند:

عدد **۷**.

او یک بار کل زنجیره را پیمایش کرد:

- اعداد **کوچک‌تر از ۷** به زنجیره‌ی چپ منتقل شدند
- اعداد **بزرگ‌تر یا مساوی ۷** به زنجیره‌ی راست رفتند

هیچ مقداری کپی نشد

هیچ گره‌ی جدیدی ساخته نشد

فقط **پیوندها (links)** دوباره تنظیم شدند

نتیجه:

چپ: $2 \rightarrow 4 \rightarrow 1 \rightarrow 5$

Pivot: 7

راست: 9

کوئیک همین قضاوت را برای زنجیره‌ی چپ و راست هم تکرار کرد؛

هر بار Pivot جدید، هر بار تقسیم، هر بار بازآرایی پیوندها.

روش او **سریع و زیبا** بود...

اما فقط وقتی Pivotها هوشمندانه انتخاب می‌شدند.

وقتی زنجیره تقریباً مرتب بود،

Pivotها نامتوازن شدند،

عمق بازگشت (Recursion) زیاد شد،

و سرعت افسانه‌ای کوئیک کاهش یافت.

پیش از ترک شهر، کوئیک رو به تو کرد و پرسید

می توانی Quick Sort را روی یک Linked List یک طرفه در پایتون پیاده سازی کنی؟

- بدون تبدیل آن به لیست پایتون
- بدون استفاده از کتابخانه‌ها
- و فقط با تغییر پیوند گره‌ها

در ادامه، اسکلت برنامه‌ی چالش آمده است.

وظیفه‌ی تو کامل و صحیح‌کردن آن است.