Data Structures and algorithms
Instructor : **Dr.Amin Eskandari**
Head-Ta's : Hamid Namjoo , Amir Hossein Hemati , Ali Mojahed
Ta's : Amir Mohammad Asadjoo , Padina razmjooi , Armin Janfaza , Alireza Heydari , Mahdi Torabi ,
Maryam Ghorbani , Fatemeh Keshavarz , MohammadReza Fasli , Abolfazl Tadrisi

Week 04 Answers

پاسخنامه تکالیف هفته ۰۴

پاسخ سوال اول :

۱ .ساختار گره

هر پهپاد = یک گره درBST

id : شناسه پهپاد (کلید)

Mission Tier : اطلاعات Tier

left و right: زیر درخت‌ها

۲ .اضافه کردن پهپاد(INSERT)

اگر درخت خالی است ← گره جدید را قرار بده.

اگر id > گره فعلی ← برو سمت چپ

اگر id < گره فعلی ← برو سمت راست

اگر id = گره فعلی ← نادیده بگیر

وقتی به جای خالی رسیدی ← گره را اضافه کن

۳ .پیدا کردن پهپاد(FIND)

از ریشه شروع کن

اگر id = گره Mission Tier را چاپ کن

اگر id >گره ← برو چپ

اگر id < گره ← برو راست

اگر رسیدی به خالی ← چاپ کنNOT FOUND

۴ .چاپ شناسه‌ها به ترتیب(INORDER)

اول زیر درخت چپ را چاپ کن

بعد شناسه گره فعلی

بعد زیر درخت راست

نتیجه: شناسه‌ها به ترتیب صعودی چاپ می‌شوند

پاسخ سوال دوم :

فایلA۲.py را اجرا کنید

برای حل این مسئله باید یک **درخت جستجوی دودویی** پیاده‌سازی کنیم. هر گره شامل سه بخش است:

- مقدار (value)
- اشاره‌گر به فرزند چپ
- اشاره‌گر به فرزند راست

سه عملیات اصلی نیاز داریم:

۱. درج (Insert)

۲. جستجو (Search)

۳. پیمایش inorder برای چاپ مرتب

در درخت BST اگر مقدار جدید از گره فعلی کوچک‌تر باشد به سمت چپ می‌رویم و اگر بزرگ‌تر باشد به سمت راست می‌رویم.

برای چاپ مرتب از **پیمایش inorder** استفاده می‌کنیم که ترتیب آن به صورت زیر است:

چپ → ریشه → راست

این پیمایش در BST باعث می‌شود اعداد به صورت **مرتب صعودی** چاپ شوند.

مراحل حل

۱. یک کلاس برای گره درخت تعریف می‌کنیم.
۲. تابع insert برای اضافه کردن مقدار جدید می‌نویسیم.
۳. تابع search برای بررسی وجود مقدار می‌نویسیم.
۴. تابع inorder برای چاپ مرتب عناصر می‌نویسیم.
۵. دستورات ورودی را خوانده و عملیات مناسب را اجرا می‌کنیم.

تحلیل زمانی

در حالت متوسط:

درج و جستجو:

O(log n)

در بدترین حالت (درخت کاملاً خطی):

O(n)

پیمایش inorder:

O(n)

پاسخ سوال سوم :

فایل A۳.py را اجرا کنید

Top K Trending Postsدر شبکه اجتماعی

برای حل این مسئله باید بتوانیم **پست‌ها را بر اساس تعداد لایک مرتب نگه داریم** تا سریع بتوانیم پست‌های ترند را پیدا کنیم.

ساختار مناسب برای این کار **Heap (هرم)** است.

ما از **Max Heap** استفاده می‌کنیم تا همیشه پستی که بیشترین لایک دارد در رأس هرم قرار بگیرد.

اما در پایتون heapq یک **Min Heap** است. بنابراین یک ترفند ساده استفاده می‌کنیم:

به جای مقدار لایک، **منفی لایک** را در heap ذخیره می‌کنیم.

پس هر عنصر heap به شکل زیر خواهد بود:

(-likes , id)

چرا id را هم ذخیره می‌کنیم؟

چون اگر لایک برابر باشد، باید id کوچکتر اول بیاید.

مشکل افزایش لایک

در heap نمی‌توان به راحتی مقدار یک عنصر را تغییر داد.

راه‌حل رایج این است که:

- مقدار جدید را دوباره داخل heap اضافه کنیم
- مقدار واقعی لایک را در یک **دیکشنری** نگه داریم
- هنگام خواندن از heap بررسی کنیم آیا مقدار معتبر است یا نه

مراحل حل

۱. یک دیکشنری برای نگهداری لایک واقعی هر پست می‌سازیم.

۲. یک heap برای نگهداری پست‌ها استفاده می‌کنیم.

۳. هنگام درج:
   - مقدار در دیکشنری ذخیره می‌شود
   - در heap اضافه می‌شود

۴. هنگام افزایش لایک:
   - مقدار در دیکشنری به‌روزرسانی می‌شود
   - مقدار جدید در heap اضافه می‌شود

۵. هنگام گرفتن Top K:
   - از heap عناصر را موقتاً خارج می‌کنیم
   - فقط مقادیر معتبر را نگه می‌داریم
   - سپس دوباره به heap برمی‌گردانیم

تحلیل زمانی

درج در heap:

$O(\log n)$

افزایش لایک:

$O(\log n)$

گرفتن k پست برتر:

$O(k \log n)$

که برای n تا ۵^۱۰ مناسب است.