جزوه جلسه بیست و پنجم داده ساختارها و الگوریتم

۲۸ آذر ۱۴۰۰

فهرست مطالب

۱ داده ساختارهای افزوده یا Augmented Data Structures

در این داده ساختارها، به منظور نگه داری اطلاعات اضافی و همچنین انجام عملیات جدید، تغییراتی در داده ساختار انجام می شود. منظور از نگه داری اطلاعات، ذخیره سازی $f(v)$ از زیردرخت v در راس v می باشد که تابع f نیز یک تابع ساده است.
تابع f برای راس v را اگر بتوان از روی f فرزندان راس v و مقدار خود v محاسبه کرد، عملیات تعریف شده برای درخت نیز در همان زمان قبلی خود انجام می گیرند (معادلا یعنی محاسبه $f(v)$ در $O(1)$ انجام می شود)
توابعی مانند جمع، ضرب، XOR، Min و Max این ویژگی را دارند اما تابعی مانند میانه، چنین ویژگی را ندارد.
اگر ویژگی ذکر شده برای f برقرار باشد، درج، حذف و یا تغییر کلید یک عنصر، تنها روی f راس های مسیر آن راس تا ریشه تاثیر میگذارد. پس به زمان اجرای قبلی عملیات داده ساختار، $\Theta(h)$ نیز اضافه می شود.
یک رویکرد دیگر برای اعمال تغییرات، انجام تغییرات در برگ ها و آپدیت کردن f برای بقیه رئوس است. درخت های AVL و R-B به دلیل کم بودن ارتفاعشان، برای اینکار مناسب هستند.

۱.۱ مسئله جمع زیربازه ها

آرایه A را داریم که در هر Query، جمع عناصر اندیس i تا j را خروجی میدهیم.
ساده ترین راه حل، استفاده از آرایه است. با انجام یک پیش پردازش $O(n)$، آرایه B را می سازیم به نحوی که:

$B[i] = \sum_{j=1}^{i} A[j]$

حال در هر کوئری با دریافت i و j برای پاسخ دادن به مسئله در $O(1)$ داریم:

$B[j] - B[i-1] = A[i] + ... + A[j]$

اگر در حین اجرای الگوریتم، مقادیر آرایه تغییر کند، اجرای الگوریتم در زمان گفته شده انجام نمی شود و کمی پیچیده می شود.
برای رفع این مشکل میتوان آرایه A را در یک درخت ذخیره کرد و $f(v)$ را مجموع کلیدهای زیر درخت v در نظر گرفت. در این پیاده سازی باید قسمت های جدید به کد مربوط به درخت AVL یا R-B اضافه کرد که در عمل چنین چیزی اتفاق نمی افتد. راه حل این مسئله با این رویکرد در جلسه بعد ذکر می شود.

۲ درخت مرتبه آماری یا Order Statistic Tree

در واسط مربوط به این درخت، عملیات زیر قابل تعریف است:

- درج یک کلید
- حذف یک کلید
- پیدا کردن عنصر بعدی: با گرفتن یک کلید، عنصر بعد کلید را خروجی میدهد.
- مرتبه عنصر داده شده: یک کلید ورودی می گیرد و مرتبه آن (چندمین عنصر در آرایه مرتب شده کلیدها) کلید را بر میگرداند.
- انتخاب عنصر با مرتبه خاص: با ورودی گرفتن یک عدد، عنصر با مرتبه ورودی را بر میگرداند.

در این داده ساختار نیاز داریم تا مرتبه عناصر را نیز ذخیره کنیم. بدین منظور تابع f را برابر با تعداد عناصر زیر درخت هر راس تعریف می کنیم.
تنها چالش ما در این داده ساختار، به روز نگه داشتن f بعد از هر درج و حذف است و بقیه عملیات به سادگی انجام می شوند.

۱.۲ پیدا کردن مرتبه عنصر داده شده

از ریشه شروع به حرکت می کنیم و یک متغیر را به صورت $order = 0$ تعریف میکنیم که در نهایت خروجی ماست. با حرکت به سمت راست $order$ را با f زیر درخت چپ بعلاوه یک جمع می کنیم و با حرکت به سمت چپ هیچ کاری نمیکنیم. در نهایت هنگام رسیدن به خود عنصر $order$ را با یک جمع می کنیم و $order$ را خروجی می دهیم.

۲.۲ انتخاب عنصر با مرتبه خاص

به f زیر درخت چپ نگاه می کنیم؛ اگر بیشتر از مرتبه ورودی بود در زیر درخت چپ به دنبال کلید می کردیم و در غیر اینصورت به مقدار f زیر درخت چپ بعلاوه یک را از مرتبه ورودی کم میکنیم و در زیر درخت راست به دنبال کلید می گردیم.

۳ درخت پاره خطی یا Segment Tree

فرض می کنیم که کلیدها اعداد صحیح یک تا m هستند. (لزومی بر برابر m و تعداد عناصر (n) نیست)
برای ایجاد درخت پاره خطی نیز یک درخت دودویی تقریبا کامل ایجاد میکنیم و f را مانند درخت مرتبه آماری برای هر راس نگه داری می کنیم.
حافظه کل مورد نیاز برابر با $4m$ می باشد.

۱.۳ گام های تشکیل درخت و تحلیل آن

برای تشکیل دادن درخت نیز m را تا اولین توان ۲ ادامه میدهیم و آنها را برگ های درخت قرار می دهیم. در گام بعدی با شروع از برگ ها، جمع هر دو راس را به عنوان parent ۲ راس ذکر شده قرار می دهیم.
مقدار f نیز برای هر راس برابر با جمع رئوس متصل به آن می شود.

- نکته قوت: ساختار درخت ثابت است و نیازی به دروان ندارد
- نکته ضعف: زمان هر عملیات زیربازه ای برابر است با $\Theta(logm)$ که برای m های بزرگ بهینه نیست و مطلوب ما $\Theta(logn)$ می باشد. (اگر$m = \Theta(n)$ الگوریتم بهینه است و در غیر این صورت خیر)

۲.۳ پیدا کردن جمع زیربازه

با داشتن a و b به عنوان ابتدا و انتهای بازه، از ریشه شروع به حرکت می کنیم. در حرکت به سمت راست، تمام عناصر سمت چپ عنصر فعلی و در حرکت به چپ تمام عناصر سمت راست راس فعلی در جمع نهایی ظاهر می شوند.
پس تعداد اعدادی که جمع می کنیم برابر است با $2log(m)$ (یک بار حرکت برای پیدا کردن a در $log(m)$ و تکرار عملیات برای b)

۳.۳ آپدیت کردن یک عنصر

برای آپدیت یک عنصر، ۲ رویکرد داریم:

- f همه رئوس از برگ آپدیت شده تا ریشه را تغییر دهیم
- رگ یا راس مدنظر را تغییر دهیم و f ها را مجدد از برگ ها تا ریشه محاسبه کنیم.

برای آپدیت بازه ای نیز برای کاهش مرتبه زمانی از تکنیک Lazy Propagation استفاده می کنیم؛ بدین صورت که هنگام رسیدن به یک راس که تمام فرزاندش تغییر کرده اند، با یک flag این تغییر را نشان می دهیم و به هنگام پایین رفتن از آن راس، در صورت لزوم شروع به تغییر فرزندان می کنیم. در این رویکرد تا حد امکان از تغییر بی مورد f رئوس دوری می کنیم.
با این تکنیک عملیات تعریف شده برای Segment Tree در $\Theta(logn)$ انجام می شود.

۴ درخت فنویک

این درخت برای f های جمع، ضرب و XOR کار میکند و مثلا برای Min و Max کار نمیکند.
برای بازه های ۱ تا یک جای خاص را می توان از این درخت استخراج کرد. به همین دلیل جمع، ضرب و XOR پشتیبانی می شود.
کد پایتون این درخت با فرض اندیس گذاری از صفر در تکه کد زیر آمده است:

```
1.  def sum(index):
2.      result = 0
3.      whild index != 0:
4.          result += array[index]
5.          index -= index & -index
6.      return result

7.  def update(index, add):
8.      while index < len(self.array):
9.          array[index] =+ add
10.         index += index & -index
```

حافظه مصرفی در این حالت نیز از مرتبه n است که به نسبت درخت پاره خطی مقدار کمتری است.