جزوه جلسه نهم داده ساختارها و الگوریتم

۲۳ مهر ۱۴۰۰

فهرست مطالب

۱ مقدمات

همانگونه که در جلسه قبل گفته شد، اردر زمانی عملیات مختلف برابر $O(height)$ میباشد که ارتفاع میتواند تا n-1 بزرگتر شود. اما هدف در درخت دودویی جست و جو این است که ارتفاع را کمتر کرده و به یک درخت متوازن برسیم که در این حالت ارتفاع به $O(logn)$ میرسد.
به طور کلی، هر درخت با ارتفاع لگاریتمی متوازن است و بالعکس.

۲ درخت دودویی جست و جوی AVL

این ساختار که در سال ۱۹۶۲ معرفی شده است، درختی متوازن است که اصلی ترین ویژگی آن، اختلاف ارتفاع فرزندان هر راس است؛ به این صورت که اختلاف ارتفاع فرزند چپ و راست هر راس، حداکثر یک واحد اختلاف دارد.
هر راس که یک فرزند داشته باشد، فرزند غایب آن none نامیده میشود و ارتفاع آن -1 در نظر گرفته میشود.

۱.۲ ارتفاع درخت AVL

ادعا میکنیم ارتفاع درخت حداکثر برابر $2log_2n$ میباشد. برای اثبات این ادعا N_h را حداقل تعداد رئوس در ارتفاع h مینامیم. اگر ارتفاع فرزند چپ و راست راس مورد نظر را با h_1 و h_2 نشان دهیم برای زوج مرتب (h_1, h_2) داریم:

1. $(h_1, h_2) = (h-1, h-1)$
2. $(h_1, h_2) = (h-1, h-2)$
3. $(h_1, h_2) = (h-2, h-1)$

بدیهی است که در حالت اول نمیتوان به دنبال مینیمم تعداد رئوس گشت. حالت های دوم و سوم معادل هم هستند پس با درنظر گرفتن ارتفاع های $(h-1, h-2)$ داریم:

$$N_h = N_{h-1} + N_{h-2} + 1$$

بدیهی است $N_{h-1} \geq N_{h-2}$ پس:

$$N_h \geq 2N_{h-2} \geq 4N_{h-4} \geq 8N_{h-8} \geq ... \implies N_h \geq 2^{h/2}$$

میدانیم:

$$n \geq N_h \implies n \geq 2^{h/2} \implies log_2n \geq h/2 \implies h \leq 2log_2n$$

پس ارتفاع درخت متوارن AVL حداکثر برابر $2log_2n$ میباشد.

۱.۱.۲ محاسبه دقیق تر ارتفاع درخت!

اگر جمله iام دنباله فیبوناتچی را با f_i نشان دهیم، ادعا میکنیم:

$$N_h = f_{n+3}-1$$

درستی رابطه فوق را نیز میتوان تحقیق کرد:

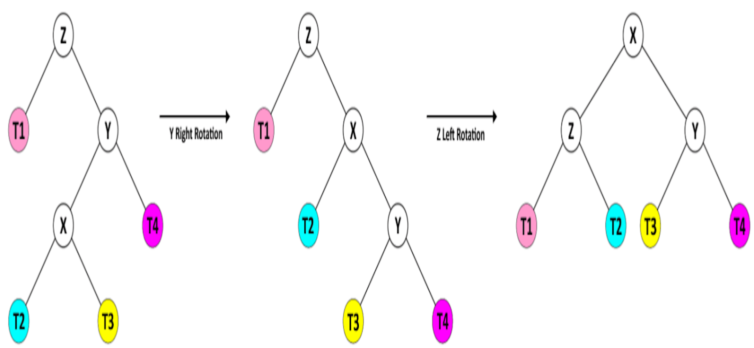
$$N_h = N_{h-1}+N_{h-2}+1 = (f_{n+2}-1)+(f_{n+1}-1)+1 = f_{n+3}-1$$

با انجام محاسبات، میتوان کران بالای بهتری برای ارتفاع درخت ارائه داد:

$$h \leq 1.440*log_2(n+1)$$

۲.۲ متوازن نگه داشتن درخت **AVL**

بعد از عملیاتی مانند درج و حذف یک کلید، ممکن است ارتفاع بعضی رئوس تغییر کرده و ویژگی درخت AVL یعنی اختلاف ارتفاع فرزندان چپ و راست برقرار نباشد. در این حالت، بسته به حالت درخت، با یک یا چند دوران میتوان شرط ارتفاع هارا برقرار کرد.
دو نوع دوران وجود دارد که در شکل زیر هردوی آنها آمده است.



شکل ۱: از چپ به راست: دوران راست روی راس Y، دوران چپ روی راس Z

۳.۲ عملیات تعریف شده برای درخت AVL

کد مربوط به find _next() و find _prev() مانند یک درخت دودویی جست و جو معمولی پیاده سازی میشود.
اما برای عملیات insert() و delete() نیاز است بعد از درج یا حذف یک راس، در صورت لزوم با انجام دوران ارتفاع رئوس به هم ریخته را درست کنیم. برای این کار، از پایین (برگ ها)، اولین راسی را درنظر میگیریم که ارتفاع آن به هم ریخته است(X). پس بعد از انجام عملیات درج یا حذف، اگر ارتفاعش را k+2 در نظر بگیریم، ارتفاع فرزند راست را k+1 و ارتفاع فرزند چپ را k-1 فرض میکنیم.
حال اگر فرزند راست (Y) را درنظر بگیریم، برای ارتفاع فرزند راست(h_1) و فرزند چپ(h_2) آن سه حالت میتوان متصور شد:

1. $(h_1, h_2) = (k, k)$

این حالت بعد از درج یک راس ناممکن است و تنها بعد از حذف یک راس میتواند اتفاق بیافتد.

2. $(h_1, h_2) = (k, k-1)$

در این حالت، با یک دوران چپ گرد روی راس Y میتوان مشکل را حل کرد و بعد از دوران نیز ارتفاع چپ و راست باهم برابر میشوند. در این حالت، بعد از انجام دوران، نیاز به بررسی رئوس بالاتر نیست.

3. $(h_1, h_2) = (k-1, k)$

در این حالت، ابتدا یک دوران راست گرد روی Y انجام میدهیم و سپس، یک دوران چپ گرد روی X انجام میدهیم.
بعد این مراحل، مجدد به بالا حرکت میکنیم و اگر راس دیگری وجود داشت که ارتفاع آن به هم ریخته بود، عملیات بالا را با توجه به ارتفاع چپ و راست آن انجام میدهیم.

۴.۲ مرتب سازی با درخت AVL

میدانیم in-order درخت AVL مرتب شده است. با توجه به این نکته میتوان کد زیر را برای مرتب سازی ارائه داد:

```
def AVl_sort(A):
    T = AVL()
    for i in A:
        T.insert(i)
    return in _order _traversal(T)
```