تکالیف جلسه هفتم ، دانشجو: کیمیا کریمی ، سکشن دوشنبه 10:15

سوال یک :

استفاده از درخت‌ها به جای لیست‌ها و آرایه‌ها در برخی مواقع می‌تواند مزایای قابل توجهی داشته باشد. در اینجا به چند دلیل و شرایطی که استفاده از درخت‌ها مفیدتر است:

جستجوی سریع‌تر:درخت‌ها به ویژه درختان جستجوی دودویی این امکان را می‌دهند که داده‌ها را به صورت مرتب ذخیره کرده و جستجو، درج و حذف را با زمان پیچیدگی O(log n) انجام دهند. در حالی که جستجو در یک لیست یا آرایه به طور متوسط به زمان O(n) نیاز دارد.

مثال: فرض کنید شما یک دیکشنری دارید و می‌خواهید کلمه‌ای را جستجو کنید. اگر داده‌ها در یک آرایه باشند، باید همه عناصر را بررسی کنید تا کلمه مورد نظر را پیدا کنید. اما اگر از یک درخت جستجوی دودویی استفاده کنید، می‌توانید با مقایسه‌های کمتر به نتیجه برسید.

ساختار سلسله‌مراتبی:درخت‌ها برای نمایش داده‌هایی با ساختار سلسله‌مراتبی بسیار مناسب هستند. مثلاً برای نمایش فایل‌ها و پوشه‌ها در یک سیستم فایل، یا برای نمایش روابط والد-فرزند.

مثال: اگر بخواهید یک ساختار سازمانی را مدل‌سازی کنید، استفاده از درخت بسیار مناسب است زیرا هر کارمند می‌تواند زیرمجموعه‌ای از کارمندان دیگر داشته باشد.

حافظه بهینه‌تر:درخت‌ها می‌توانند به شما کمک کنند تا داده‌ها را به صورت دینامیک و بدون نیاز به تخصیص مجدد حافظه ذخیره کنید. در آرایه‌ها، اندازه ثابت است و اگر بخواهید داده‌های بیشتری اضافه کنید، نیاز به ایجاد یک آرایه جدید دارید.

مثال: فرض کنید که شما یک برنامه دارید که به طور مداوم داده‌های جدیدی دریافت می‌کند. با استفاده از یک درخت، می‌توانید به راحتی داده‌های جدید را اضافه کنید بدون اینکه نیاز باشد فضای حافظه جدیدی تخصیص دهید.

عملیات پیچیده‌تر:درخت‌ها امکان انجام عملیات پیچیده‌تری مانند پیمایش و فیلتر کردن داده‌ها را با کارایی بالا فراهم می‌کنند. این عملیات در لیست‌ها و آرایه‌ها ممکن است پیچیده‌تر و زمان‌بر باشد.

مثال: اگر بخواهید تمام عناصر یک مجموعه خاص را پیدا کنید که شرایط خاصی را برآورده می‌کنند، استفاده از درخت می‌تواند این کار را سریع‌تر و آسان‌تر کند.

به طور کلی، انتخاب بین درخت‌ها و لیست‌ها یا آرایه‌ها بستگی به نوع داده‌ها و نیازهای خاص برنامه شما دارد. اگر نیاز به جستجو، درج و حذف سریع دارید یا داده‌ها ساختار سلسله‌مراتبی دارند، درخت‌ها گزینه مناسبی هستند. اما اگر فقط نیاز به ذخیره‌سازی ساده و دسترسی سریع به عناصر دارید، لیست‌ها یا آرایه‌ها ممکن است کافی باشند.

سوال دوم :

درخت اول

(Post-order): 4, 5, 2, 3, 1

(In-order): 4, 2, 5, 1, 3

(Pre-order): 1, 2, 4, 5, 3

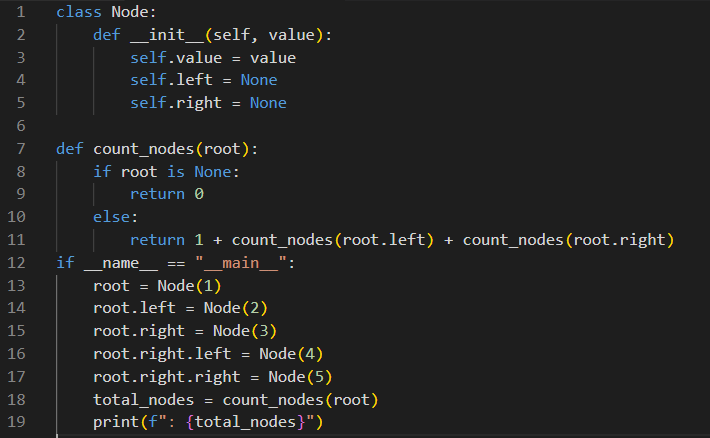
درخت دوم

(Post-order): 2, 1, 12, 3, 9, 7, 11, 5, 4, 8

(In-order): 1, 2, 5, 9, 7, 11, 3, 4, 8

(Pre-order): 8, 5, 9, 1, 2, 7, 11, 3, 4

سوال سوم:



سوال چهارم :

▎1. تفاوت بین دو تابع add\_new\_node\_1 و add\_new\_node\_2

هر دو تابع برای اضافه کردن یک گره جدید به درخت استفاده می‌شوند، اما نحوه پیاده‌سازی و رفتار آنها متفاوت است:

• add\_new\_node\_1:

• این تابع به سادگی یک گره جدید به عنوان فرزند چپ والد اضافه می‌کند.

• اگر والد قبلاً فرزندی داشته باشد، گره جدید به عنوان فرزند چپ قرار می‌گیرد و فرزند قبلی به عنوان برادر راست گره جدید قرار می‌گیرد.

• به این ترتیب، فقط یک فرزند چپ برای هر والد وجود دارد و سایر فرزندان به صورت برادران راست مرتب می‌شوند.

• add\_new\_node\_2:

• این تابع نیز یک گره جدید به والد اضافه می‌کند، اما با یک رفتار متفاوت.

• اگر والد هیچ فرزندی نداشته باشد، گره جدید به عنوان فرزند چپ قرار می‌گیرد.

• اگر والد فرزندی داشته باشد، این تابع به دنبال آخرین برادر راست می‌گردد و گره جدید را به عنوان برادر راست آخرین فرزند موجود اضافه می‌کند.

• به این ترتیب، می‌توان چندین فرزند برای هر والد داشت که به صورت زنجیره‌ای به هم متصل هستند.

▎2. نحوه کارکرد تابع find\_in\_subtree

تابع find\_in\_subtree برای جستجوی یک گره با برچسب مشخص در زیر درختی که از یک گره خاص شروع می‌شود، طراحی شده است. مراحل کار این تابع به شرح زیر است:

1. بررسی برچسب: ابتدا بررسی می‌کند که آیا برچسب گره جاری (node) با برچسب مورد نظر مطابقت دارد یا خیر. اگر مطابقت داشت، آن گره را بازمی‌گرداند.

2. جستجوی فرزندان: اگر گره جاری برچسب مورد نظر را نداشت، به فرزندان آن گره (که از طریق left\_child و right\_sibling مشخص شده‌اند) مراجعه می‌کند.

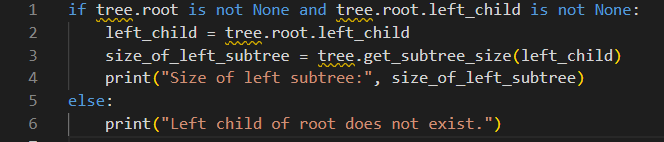
3. حلقه جستجو: با استفاده از یک حلقه، از فرزند چپ شروع کرده و به ترتیب به سمت راست (برادران راست) حرکت می‌کند و برای هر فرزند از خود تابع find\_in\_subtree را فراخوانی می‌کند.

4. نتیجه نهایی: اگر هیچ گره‌ای پیدا نشود، در نهایت None بازمی‌گرداند.

▎3. بدست آوردن اندازه زیر درخت فرزند چپ ریشه

برای بدست آوردن اندازه زیر درخت فرزند چپ ریشه، می‌توانید از تابع get\_subtree\_size استفاده کنید. برای این کار کافی است

# فرض کنید tree یک شیء از کلاس Tree است و ریشه آن مقداردهی شده است



این کد ابتدا بررسی می‌کند که آیا ریشه و فرزند چپ آن وجود دارند یا خیر و سپس اندازه زیر درخت فرزند چپ را با استفاده از تابع get\_subtree\_size محاسبه می‌کند.