درخت عملی

1- یک درخت جستجوی دودویی با مقادیر زیر بسازید و آن را با پیمایش **(in-order)** و **(pre-order)** و **(post-order)** نمایش دهید:(چپ به راست)

```
[100, 50, 150, 25, 75, 125, 175, 10, 30, 60, 90, 110, 140, 160, 190]
```

2- یک درخت جستجوی دودویی با مقادیر زیر بسازید مراحل و درخت نهایی رو رسم کنید.(چپ به راست)

`[50, 30, 70, 20, 40, 60, 80, 15, 25]` سپس گره های 20 و 50 و 30 را به ترتیب گفته شده حذف نموده و مراحل حذف را توضیح و نمایش دهید.

راهنمایی:

- برای حذف گره با دو فرزند، از گره جانشین (successor) یا پیشین (predecessor) استفاده کنید و توضیح دهید که چرا یکی از این دو را انتخاب کردید.

درخت جذاب

آرایه ای از کدهای مشتریان یک فروشگاه به شما داده شده است `[10, 20, 30, 40, 50, 60]`

- الف) این کد ها را از چپ به راست به ترتیب در یک درخت دودویی جستجو درج کنید و درخت نهایی را رسم کنید.
- ب) توضیح دهید که چرا این درخت به این شکل درآمده است و چه ویژگی دارد که آن را از یک درخت BST نرمال و کارآمد متمایز میکند؟
- ج) عنصر `60` را در این درخت جستجو کنید و پیچیدگی زمانی عملیات جستجو در این نوع درخت را تحلیل کنید؟

شما برای یک اپلیکیشن تحلیل داده‌های فروش، سیستمی طراحی می‌کنید که قیمت محصولات (عدد صحیح منحصربه‌فرد) در یک BST ذخیره می‌شود. مدیر فروش می‌خواهد سریعاً ارزان‌ترین و گران‌ترین محصول را پیدا کند تا برای تنظیم تخفیف‌ها تصمیم‌گیری کند.

آرایه قیمت‌ها: `[45, 20, 70, 10, 30, 60, 80]` (چپ به راست)

- الف) کوچکترین و بزرگترین گره در این درخت را پیدا کرده و گام های پیمایش را خلاصه توضیح دهید؟
- ب) چرا کوچکترین و بزرگترین گره ها در این جایگاه ها قرار دارند؟

پوی برنامه نویس

در یک روستای دورافتاده، در بالای کوهستانی سرسبز، **پو**، پاندای کونگفوکار، در حال تمرین بود که استاد **شیفو** او را صدا زد.

شیفو: "پو! امروز باید یک مهارت جدید یاد بگیری. پیدا کردن چیزهای ارزشمند در میان آشفتگی‌ها، مانند پیدا کردن سوزن در انبار کاه!"

پو که همیشه عاشق یادگیری چیزهای جدید (و البته عاشق غذا!) بود، با اشتیاق گفت: "عالیه استاد! چی یاد می‌گیرم؟ یه حرکت جدید تو کونگفو؟!"

استاد شیفو سری تکان داد و لبخند زد. "نه، امروز یاد می‌گیری چطور در **درخت دودویی جستجو** به‌دنبال یک مقدار بگردی."

پو متعجب شد. "درخت دودویی جستجو؟ یعنی یه درخت پر از دامپلینگ؟!"

استاد شیفو آهی کشید. "نه پو! این یک درخت خاص است که هر مقدار کوچک‌تر از یک مقدار مشخص، در سمت چپ و هر مقدار بزرگ‌تر در سمت راست آن قرار می‌گیرد. اگر بخواهی بفهمی که یک مقدار خاص در این درخت هست یا نه، باید جستجو کنی، ولی نه به‌صورت تصادفی!"

"اوه! یعنی اول مقدار وسط رو نگاه کنم، بعد اگه کوچیک‌تر بود سمت چپ برم و اگه بزرگ‌تر بود سمت راست؟"

استاد شیفو سر تکان داد. "آفرین پو! دقیقاً همین‌طور است."

حالا که پو اصول کار را یاد گرفت، وقت آن بود که این مهارت را در کدنویسی پیاده کند! 🐼💻

مثال‌ها

ورودی نمونه 1

```
10 5 15 2 7 12 18
6
```

خروجی نمونه 1

```
False
```

ورودی نمونه 2

```
100 50 200 25 75 150 250
150
```

خروجی نمونه 2

```
True
```

ورودی نمونه 3

```
15 10 20 5 12 17 25
17
```

خروجی نمونه 3

```
True
```

تام و درخت دودویی

یک روز، تام تصمیم گرفت که خانه‌اش را به‌طور مرتب‌تر و منظم‌تر کند. او همیشه به‌دنبال راه‌هایی بود که بتواند به‌سرعت هرچیزی که می‌خواهد را پیدا کند. جری، موش بامزه‌ای که همیشه در خانه تام حضور داشت، پیشنهاد کرد که تام می‌تواند از یک **درخت دودویی جستجو** برای مرتب کردن و ذخیره‌سازی اشیاء استفاده کند.

تام کنجکاو شد و از جری پرسید: "درخت جستجوی دودویی چطور کار می‌کنه؟"

جری با لبخند جواب داد: "خیلی ساده است! فکر کن به یک درخت که هر گره‌اش یک شیء با یک مقدار خاص داره. اگر مقداری که می‌خواهی وارد کنی بزرگ‌تر از مقدار گره کنونی باشه، باید به سمت شاخه‌ی راست بری، اگر کوچیک‌تر باشه، به سمت شاخه‌ی چپ!"

تام با دقت به جری گوش می‌داد و جری ادامه داد: "حالا برای اینکه بفهمی چی داخل درخت گذاشتی، می‌تونی از روشی به اسم **پیش‌نویسی** (preorder) استفاده کنی. یعنی اول خود گره رو چک می‌کنی، بعد به شاخه‌های چپ و راست می‌ری."

تام سریع شروع به یاد گرفتن کرد و جری هم بهش کمک کرد تا بتونه یک درخت جستجوی دودویی بسازه. حالا هر وقت که تام چیزی رو می‌خواست، فقط کافی بود به درخت نگاه کنه و اون رو به‌سرعت پیدا کنه!

لطفا تام و جری رو در پیاده سازی یک درخت جستجوی دودویی کمک کنید.

مثال‌ها

ورودی نمونه ۱

```
5 3 8 2 4 7 9
```

خروجی نمونه ۱

```
5 3 2 4 8 7 9
```

ورودی نمونه ۲

```
15 10 20 5 12 17 25
```

خروجی نمونه ۲

```
15 10 5 12 20 17 25
```

درخت جذاب میشود

AVL Tree

سوال اول

- الف) در یک درخت AVL اصطلاح balancing را توضیح دهید.
- ب) آیا یک درخت AVL میتواند غیرکامل اما متعادل باشد؟
- ج) تفاوت ساختاری درخت دودویی جستجو رو با درخت AVL بررسی کرده و به صورت خلاصه بنویسید.

سوال دوم

لیست زیر از اعداد صحیح را به ترتیب در یک درخت AVL درج کنید:

[45, 30, 50, 25, 35, 20, 27, 10, 28, 60, 32, 31, 34, 33]

۱. پس از **هر درج**، تراز (Balance Factor) گره‌ها را بررسی کرده و اگر درخت نامتوازن شد، آن را **متعادل‌سازی (Rebalance)** کنید.

۲. در هر مرحله‌ای که متعادل‌سازی نیاز است، نوع چرخش مورد استفاده را مشخص کنید:

- LL (چرخش ساده به راست)
- RR (چرخش ساده به چپ)
- LR (چرخش دوگانه: چپ-راست)
- RL (چرخش دوگانه: راست-چپ)

۳. **درخت نهایی** را رسم کرده و تراز هر گره را مشخص کنید.

درخت جذاب تر میشود

RED-BLACK Tree

صورت سؤال:

فرض کنید یک درخت Red-Black تهی در اختیار داریم. به ترتیب زیر عناصر را در آن درج می‌کنیم. در هنگام درج، تمام قوانین Red-Black Tree را رعایت کرده و در صورت نیاز **چرخش** (Rotation) و **تغییر رنگ (Recoloring)** انجام دهید.

ترتیب درج:(چپ به راست)

```
M, B, Q, A, C, N, R, P, D, O, F, E
```

سوالات:

۱. **رسم درخت نهایی**: درخت نهایی Red-Black را پس از درج همه‌ی عناصر رسم کنید. در هر گره رنگ آن را مشخص کنید.
۲. **تجزیه‌وتحلیل قوانین**: بررسی کنید که آیا در درخت نهایی تمام قوانین Red-Black برقرار هستند یا خیر:
   - ریشه سیاه است؟
   - هیچ دو گره قرمز متوالی وجود ندارد؟
   - همه مسیرهای ریشه تا برگ، تعداد مساوی گره‌های سیاه دارند (Black Height)؟
۳. **محاسبه Black Height**: Black Height درخت را محاسبه کرده و آن را برای مسیر چپ‌ترین و راست‌ترین برگ بنویسید.

پیمایش‌ها:

- ترتیب **Inorder** درخت را بنویسید.
- ترتیب **Postorder** درخت را بنویسید.

مقایسه مقایسه

به سوالات زیر به صورت مختصر و مفید پاسخ دهید.

- 1) پیچیدگی زمانی عملیات درج , حذف و جستجو در درخت های `BST` , `AVL` , `RED-BLACK` را باهم مقایسه کنید.
- 2) در چه شرایطی AVL Tree کارایی بهتری نسبت به Red-Black Tree دارد؟ مثال بزنید.
- 3) در چه مواقعی در درخت دودویی جستجو با مشکل مواجه میشدیم که بخاطر آن به فکر طراحی درخت های بالانس افتادیم؟(**راهنمایی :** به دنبال ارتفاع و پیچیدگی باشید!)
- 4) اگر داده‌ها تقریباً مرتب باشند، کدام ساختار درخت عملکرد بهتری خواهد داشت؟
- 5) آیا Red-Black Tree همیشه تعداد چرخش‌های کمتری نسبت به AVL دارد؟ تحلیل کنید.

تبدیل پر حاشیه

درخت‌های **قرمز-مشکی (Red-Black Tree)** و **AVL (Adelson-Velsky and Landis)** هر دو درخت‌های دودویی جستجو (BST) هستند که ویژگی‌های خاصی برای حفظ تعادل دارند. درخت‌های قرمز-مشکی محدودیت‌های خاصی دارند، از جمله رنگ‌دهی به گره‌ها و حفظ تعداد یکسان گره‌های سیاه در هر مسیر از ریشه به برگ، در حالی که درخت‌های AVL به گونه‌ای طراحی شده‌اند که تفاوت ارتفاع زیر درخت‌های چپ و راست هر گره نباید از 1 بیشتر باشد.

در این مسئله، هدف شما این است که یک درخت قرمز-مشکی داده‌شده را به یک درخت AVL تبدیل کنید. با انجام این کار، ویژگی‌های خاص درخت قرمز-مشکی (مانند رنگ‌دهی و قوانین خاص) باید رعایت شده و در عین حال، تعادل درخت به‌صورت AVL حفظ شود.

ورودی:

یک درخت قرمز-مشکی داده‌شده، که دارای ویژگی‌های زیر است:

۱. هر گره می‌تواند یا قرمز یا سیاه باشد.
۲. هر مسیر از ریشه تا برگ باید تعداد یکسانی از گره‌های سیاه داشته باشد.
۳. هیچ‌گاه دو گره قرمز متوالی در یک مسیر وجود ندارد.
۴. درخت یک درخت دودویی جستجو (BST) است.

خروجی:

یک درخت AVL که همان درخت قرمز-مشکی ورودی را نمایش می‌دهد، به‌طوری که:

۱. پس از تبدیل، درخت باید ویژگی‌های درخت AVL را رعایت کند (یعنی تفاوت ارتفاع دو زیر درخت در هر گره نمی‌تواند بیشتر از 1 باشد).
۲. درخت جدید باید به‌طور صحیح تعادل خود را حفظ کرده و تمامی ویژگی‌های خاص خود را که در درخت AVL باید وجود داشته باشد، برآورده کند.

۳. ویژگی‌های قرمز-مشکی نباید در درخت نهایی تغییر کنند.

راهی به نزدیک ترین

درخت جستجوی دودویی (Binary Search Tree - BST) داده شده است که هر گره آن شامل یک عدد صحیح یکتا است.

عدد صحیح `target` نیز داده شده است.

الگوریتمی طراحی کنید که **عنصری از درخت را بیابد که مقدار آن نزدیک‌ترین مقدار به `target` باشد**.

نکات:

- اگر دو مقدار به یک فاصله از `target` باشند، مقدار کوچکتر را انتخاب کنید.
- الگوریتم باید از ویژگی‌های BST برای بهینه‌سازی جستجو استفاده کند.
- تحلیل زمانی الگوریتم را نیز بنویسید.

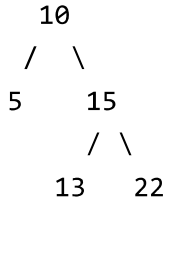
ورودی:

- ریشه‌ی درخت BST
- عدد صحیح `target`

خروجی:

- عددی از درخت که نزدیک‌ترین مقدار به `target` است.

مثال



`target = 12` باشد.

نزدیک‌ترین مقدار به 12 در این درخت، عدد **13** است. بنابراین خروجی:

k-امین بزرگترین

درخت جستجوی دودویی (BST) داده شده است. عدد صحیح مثبت k نیز داده شده است. الگوریتمی طراحی کنید که **kامین عنصر بزرگ‌تر** (یعنی بزرگ‌ترین، دومین بزرگ‌ترین، ... تا kام) را در زمان بهینه بیابد.

نکات:

- منظور از "kامین بزرگ‌تر"، kامین عدد از انتهای ترتیب مرتب‌شده (Descending order) است.
- الگوریتم باید از خاصیت BST و پیمایش معکوس (Right → Root → Left) inorder برای جستجوی بهینه استفاده کند.

ورودی:

- ریشه درخت BST
- عدد صحیح k

خروجی:

- kامین عنصر بزرگ‌تر در BST

```
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;

    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};
```