نمونه سوال تمرین شماره ۳

دانشکده مهندسی برق و کامپیوتر

استاد : **دکتر فقیه**

ساختمان داده - بهار ۱۳۹۹

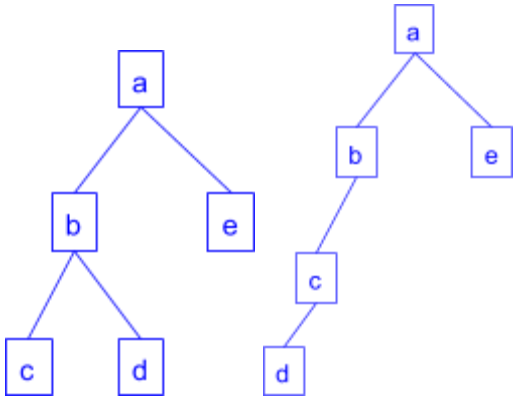
مسئول تمرین : **محمدهادی امیدی**

mh.omidi@ut.ac.ir

---

۱. جملات زیر را بررسی کرده و درستی و غلطی هر یک را اعلام کنید و در صورت درست بودن جمله، برای پاسخ خود دلیل مختصری بیاورید و در صورت غلط بودن مثال نقض بزنید..

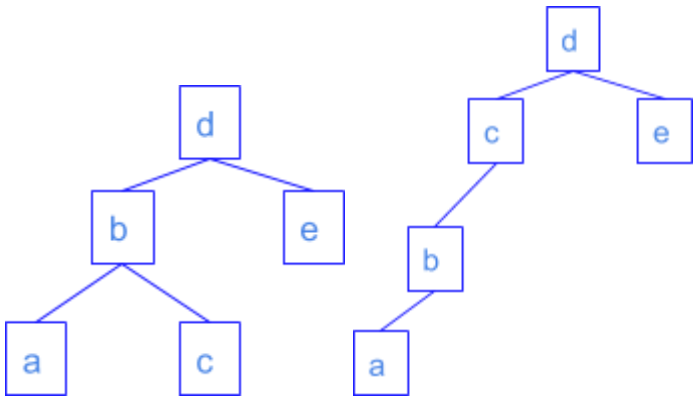
الف) از پیمایش پیش ترتیب یک درخت دودویی می توان پیمایش پس ترتیب آن را به دست آورد. (غ) چون برای یک پیمایش دو درخت می توان کشید، پس درخت به صورت یکتا قابل بازیابی از روی پیمایش نیست. برای مثال:



نمایش پیش ترتیب دو درخت بالا: abcde

ب) از پیمایش میان ترتیب یک درخت دودویی، می توان پیمایش پس ترتیب آن را پیدا کرد. (غ)

در این سوال نیز همانند بخش الف، چون برای یک پیمایش دو درخت می توان کشید، پس درخت به صورت یکتا قابل بازیابی از روی پیمایش نیست. پیمایش میانترتیب دو درخت زیر نیز یکسان است.



پیمایش میان ترتیب این دو شکل: abcde

ج) اگر پیمایش میانترتیب و پس ترتیب یک درخت دودویی را داشته باشیم، می توان درخت یکتای آن را به دست آورد. (ص)
آخرین عنصر در پس ترتیب همان ریشه در درخت است. بنابراین در میانترتیب می چرخیم و ریشه را پیدا می کنیم و میان‌ترتیب را به دو زیر درخت چپ و راست تقسیم می کنیم که هر یک از آن ها نیز پیمایش میانترتیب است. حال عنصر بعدی در نمایش پس ترتیب، ریشه زیر درخت سمت راست است و در نمایش میان ترتیب نیز آن را، پیمایش سمت راست ریشه را به دو بخش تقسیم می کند. همین کار را آن‌قدر ادامه می‌دهیم تا زیر درخت سمت راست به طور کامل ساخته شود (یعنی هیچ عنصری در سمت راست ریشه درخت اصلی در نمایش میانترتیب نمانده باشد که مکان آن در زیر درخت سمت راست ریشه درخت اصلی مشخص نشده باشد). پس از این، همین کار را بر روی زیر درخت ایجاد شده در سمت چپ ریشه درخت اصلی انجام می‌دهیم. عنصری که در پیمایش پس ترتیب پس از آخرین عنصر زیر درخت سمت راست می آید، ریشه زیر درخت سمت چپ ریشه اصلی درخت خواهد بود. این کار را تا زمانی انجام می‌دهیم تا جایگاه تمام عناصر در پیمایش پس ترتیب در درخت اصلی مشخص شود. طبق این الگوریتم همواره زیر درخت‌های سمت چپ و راست در تمام عناصر یکتا هستند و می توان گفت که درخت مورد نظر یکتاست.

۲. الگوریتمی غیر بازگشتی برای پیمایش **inorder** یک درخت دودویی در زمان خطی ارایه دهید.

از یک استک و ارایه کمکی (mark) به طول بزرگترین عنصر درختمان کمک می گیریم ابتدا ریشه را در درون استک push می کنیم آرایه mark را برای این گرفتیم که چک کنیم آیا گره i را قبلا پیمایش کردیم یا خیر. در صورت پیمایش (یا true بودن) دیگر به پیمایش آن نمی پردازیم . در هر مرحله عنصر بالای استک را در نظر می گیریم مثلا V سپس اگر بچه سمت چپ این گره false بود آن را در استک push می کنیم در غیر این صورت خود v را بررسی می کنیم[v]mark را true می کنیم اگر هم از قبل true بود سراغ بچه سمت راست گره v می رویم اگر v.right نیز true بود V را POP می کنیم در غیر این صورت v.right را push در استک می کنیم .در این روش inorder درخت را غیر بازگشتی طی کردیم. (این الگوریتم دارای پیچیدگی زمانی خطی است اما از نظر پیچیدگی فضای حافظه خطی نیست.)

۳. تابع پیدا کردن parent را در درخت دودویی معادل یک درخت عمومی پیاده سازی کنید.
در پیاده سازی درخت عمومی با استفاده از درخت دودویی، فرزند چپ هر گره، چپ ترین فرزند آن، و فرزند راست آن، برادر راست آن می‌باشد. برای پیدا کردن parent یک گره به روش بازگشتی به روش زیر عمل می‌کنیم. هدف این است که یک بار در زیر درخت راست به دنبال گره و در نهایت parent آن بگردیم و یک بار در زیردرخت چپ. تنها تفاوتی که در این بین وجود دارد، این است که وقتی به گره راست مراجعه می‌کنیم، اگر گره راست با گره مربوطه یکسان بود، parent آن با parent گره فعلی برابر است، چون این دو گره در درخت اصلی برادر هستند. در حالی که وقتی به گره چپ مراجعه می‌کنیم، اگر گره چپ با گره مورد نظر یکی بود، parent آن در واقع گره فعلی می

باشد، چون گره چپ در درخت اصلی هم فرزند گره فعلی می باشد. این کار با استفاده از پارامتر parent انجام می‌گیرد. قطعه کد زیر برای پیدا کردن parent یک درخت دودویی معادل درخت عمومی است.

```
def get_parent(root, node, parent=None):
    if node == root:
        return parent
    if root.right is not None:
        p = get_parent(root.right, node, parent)
        if p is not None:
            return p
    if root.left is not None:
        p = get_parent(root.left, node, root)
        if p is not None:
            return p
    return None
```

۴. آرایه‌ای از اعداد غیر مرتب به ما داده شده است. می خواهیم k عنصر نزدیک به عدد x را در آن بیابیم. الگوریتمی بهینه ارائه دهید که بتوان این کار را در $O(n\ lg\ k)$ انجام دهد.

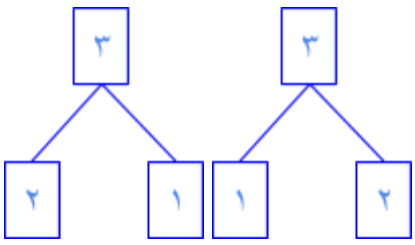
برای این کار ابتدا با اختلاف k عنصر اول آرایه با عدد x، یک Max heap می‌کنیم.سپس برای عناصر از k+1 به بعد مراحل زیر را طی می‌کنیم.

1. اگر اختلاف این عنصر با عدد x از ریشه درخت هیپ بزرگ تر بود، آن را کنار می‌گذاریم و سراغ عنصر بعدی می‌رویم
2. اگر اختلاف این عنصر از x از ریشه درخت کمتر بود ابتدا ریشه را حذف کرده و سپس این عنصر را به هیپ اضافه می‌کنیم و سراغ عنصر بعدی می‌رویم.

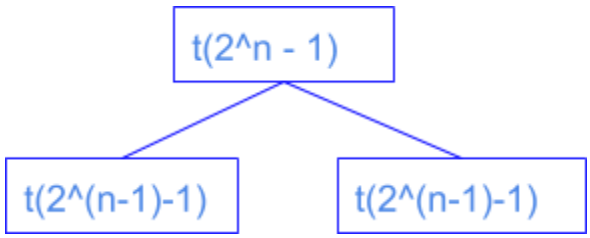
این کار را تا جایی ادامه می دهیم که عنصری از عناصر آرایه باقی نماند. باید برای این کار کل آرایه طی کنیم و در هر مرحله در بدترین حالت $O(lg\ k)$ هزینه داریم. بنابراین هزینه‌ی کلی برابر $O(n\ lg\ k)$ است.

۵. فرض کنید $2^n - 1$ عدد داریم و می‌خواهیم آن‌ها را به صورت یک درخت هیپ در بیاوریم. به چند حالت می توان آن‌ها را max heap کرد؟

برای حل این سوال ابتدا با ۳ گره تعداد حالات کلی را رسم می کنیم. فرض کنید عناصر ما اعداد ۱ تا ۳ هستند. آنگاه داریم:



بنابراین برای ۳ گره ۲ حالت داریم. حال برای حالت $2^n - 1$ به صورت زیر است:



به عبارت دیگر برای $2^n - 1$ عنصر داریم:

$$T(2^n - 1) = C(2^n - 2,\ 2^{n-1} - 1) * T(2^{n-1} - 1)^2$$

با دانستن این مسئله می توان از $T(3)$ شروع کرد و T عدد مورد نظر را به صورت بازگشتی محاسبه کرد.