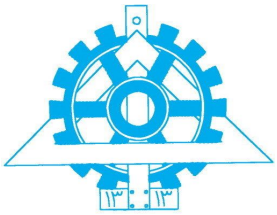
پاسخ تمرین شماره ۳



دانشکده مهندسی برق و کامپیوتر

استاد : **دکتر فقیه**

ساختمان داده - بهار ۱۳۹۹

مسئول تمرین : **محمدهادی امیدی**

mh.omidi@ut.ac.ir

---

۱. آرایه نامرتب زیر را با به یک آرایه min heap از چپ به راست به صورت inplace تبدیل کنید. (در هر مرحله فقط جایگاه عدد را پس از پایان heapify نشان دهید)

{۱، ۵، ۱۲، ۲، ۱۰۰، ۱۷، ۲۲، ۱۸، ۱۲۳، ۸۲، ۲۵}

در هر مرحله آرایه چپ هیپ شده و راست باقی آرایه

{25} {82, 123 ,…}

{25, 82} {123 ,…}

{25, 82, 123} {18, 22 ,…}

{18, 25, 82, 123} {22 ,…}

{18, 25, 22, 82, 123} {17 ,…}

…

{1, 2, 17, 12, 5, 123, 100, 18, 25, 82, 22}

۲. تعداد n دختر و n پسر داریم که هر یک ارث پدری گیرشان آمده است و این افراد مجرد هستند. این افراد طی یک اتفاق عجیب تصمیم گرفته اند که با فردی مانند خود ازدواج کنند (منظور این است که یک دختر دوست دارد با پسری ازدواج کند تا احوال یکدیگر را بهتر درک کنند!). می خواهیم این پسرها را به دخترها برسانیم. اگر پسری با دختری ازدواج کند، میراث آنها با یکدیگر جمع می شود. الگوریتمی ارائه دهید که بتوان k زوج پولدار را پیدا کرد. الگوریتم شما باید بهینه باشد.

ابتدا max heap هر یک از دسته ها را به صورت جداگانه می سازیم که این عمل طبق پیچیدگی زمانی ساخت max heap، زمان $O(n)$ می برد. حال در هر مرحله باید getMax را صدا بزنیم که در عمل برابر $O(lg\ n)$ است. چون باید k تا زوج برداریم. بنابراین پیچیدگی زمانی این الگوریتم $O(n + k\ lg\ n)$ است

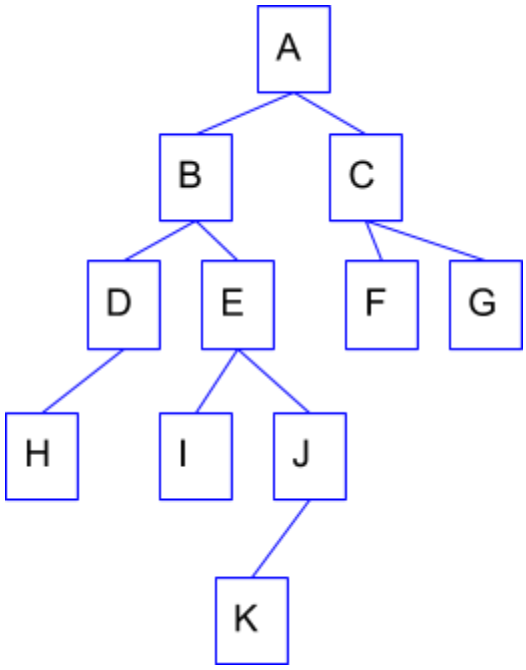
۳. دو min heap با نامهای m1 و m2 را در نظر بگیرید به طوری که تمام عناصر m1 از تمامی عناصر در m2 بزرگتر است. الگوریتمی ارائه دهید که این دو min heap را ادغام کرده و یک min heap بزرگتر بسازد. پیچیدگی این الگوریتم را بیان کنید. (min heap ها درخت دودویی کامل هستند.)

کافیست یک آرایه با طول $m1 + m2$ بسازیم و آرایه‌ی هیپ m2 را در ابتدا و m1 را در انتها بگذاریم. چون باید تمام این عناصر در آرایه کپی شوند بنابراین پیچیدگی زمانی برابر $O(m1 + m2)$ دارد.

۴. الف) درختی با شرایط زیر رسم کنید.

Inorder: HDBIEKJAFCG

Preorder: ABDHEIJKCFG



ب) پیمایش postorder درخت را بنویسید.

Preorder: HDIKJEBFGCA

۵. درخت دودویی A دارای n گره و درخت دودویی B دارای m گره است که هیچ عنصر تکراری در آن‌ها نیست. الگوریتمی از $O(m + n)$ ارائه دهید که نشان دهد این دو درخت یکسان هستند یا خیر؟ درستی الگوریتم خود را توضیح دهید.

راه اول: الگوریتم زیر را به صورت بازگشتی به روی هر یک از گره‌های یکسان درخت و با شروع از ریشه درخت‌ها می‌زنیم:

۱. اگر هر دو ریشه پوچ بودند، مقدار ۱ باز می گردانیم.

۲. اگر پوچ نبودند، سه چیز را چک می کنیم:

۱.۲. آیا ریشه‌ها مقادیر یکسانی دارند.

۲.۲. آیا زیر درخت‌های سمت چپ هر دو ریشه با هم برابرند. (بازگشتی)

۳.۲. آیا زیر درخت‌هاای سمت راست هر دو ریشه با هم برابرند. (بازگشتی)

۳. اگر ۳ شرط مرحله ۲ درست بود ۱ بر می گردانیم و در غیر این صورت صفر.

در این حالت تمام گره‌های هر دو درخت باید چک شود. بنابراین پیچیدگی آن $O(n+m)$ می‌شود.

راه دوم: 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.

بنابراین کافیست پیمایش پس ترتیب و میان ترتیب دو درخت یکسان باشد. از آن‌جا که نوشتن زیر پیمایش‌ها و چک کردن آن‌ها پیچیدگی $O(n)$ و $O(m)$ دارد بنابراین پیچیدگی کلی الگوریتم $O(m+n)$ است.

۶. درخت T را در نظر بگیرید که n گره دارد و هر گره غیر برگ در آن دو فرزند دارد. E نشان‌دهنده مجموع عمق برگ‌ها و I نشان دهنده‌ی مجموع عمق داخلی (عناصر غیر برگ) باشد. ثابت کنید. $E-I=n-1$

کافیست استقرا کنیم در هر مرحله e دو واحد افزایش و i یک واحد پس مجموع یک واحد بالا می رود.

پایه استقرا: برای n=3 این مسئله صدق می کند.

فرض: برای n=k مسئله گفته شده صدق می کند.

حکم: برای n=k+2 صحیح است. تعداد فرزندان باید ۲ تا ۲ تا بالا رود. زیرا در غیر اینصورت شرط مسئله نقض می‌شود.

اثبات: با توجه به این در هر مرحله قرار است یکی از برگ ها برداشته شود و دو تا فرزند به او داده شود بنابراین در هر مرحله یک برگ با ارتفاع h از برگ‌های قبلی کم شده و به برگ‌های داخلی اضافه شده و دو برگ با ارتفاع h+1 را به وجود می آید. بنابراین داریم:

$$E_{new}=E_{old}-h+2(h+1)\ ,\ I_{new}=I_{old}+h$$
$$\rightarrow E_{new}-I_{new}=n-1-h+2(h+1)-h=n+1$$

بنابراین حکم ثابت می‌شود.

۷. تابع غیر بازگشتی طراحی کنید که در یک درخت دودویی، ارتفاع درخت را برگرداند. (فرض کنید اگر تعداد گره‌های درخت n باشد، بزرگترین عدد موجود در درخت 2n است.)

برای حل این بخش، باید از یک پشته و یک آرایه به طول 2n کمک بگیریم. از ریشه شروع می‌کنیم و آن را گره جاری در نظر می‌گیریم. سپس در هر مرحله:

۱. اگر فرزند چپ وجود داشت و چک نشده بود، گره را در پشته می کنیم و به فرزند چپ را گره جاری می کنیم و از مرحله ۱ کارها را برای آن گره انجام می‌دهیم.

۲. اگر فرزند راست داشت و چک نشده بود، گره را در پشته می کنیم و به فرزند راست را گره جاری می کنیم و از مرحله ۱ کارها را برای آن گره انجام می‌دهیم.

۳. اگر فرزندی نداشت و یا فرزند چک نشده نداشت، ارتفاع بیشینه و چک شدن گره مورد نظر را بروزرسانی می‌کنیم و سپس از پشته یک بار pop کرده و گره جاری را برابر گره pop شده از پشته در نظر می‌گیریم و مراحل را از ۱ برای آن تکرار می‌کنیم.

۴. این فرایند را تا جایی انجام می دهیم تا پشته خالی شود.

قطعه کد مربوط به این فرایند در زیر آمده است.

```
def get_height(root, n):
    remained_nodes = stack()
    remained_nodes.push(root)
    checked_nodes = [False for i in range(2 * n)]
    height = 0
    max_height = -1
    curr = root
    while remained_nodes:
        if curr.left and checked_nodes[curr.left.value] is False:
            remained_nodes.push(curr)
            curr = curr.left
            height += 1
            continue
        if curr.right and checked_nodes[curr.right.value] is False:
            remained_nodes.push(curr)
            curr = curr.right
            height += 1
            continue
        if height > max_height:
            max_height = height
        checked_nodes[curr.value] = True
        curr = remained_nodes.pop()
        height -= 1
    return max_height
```