ساختمان داده و الگوریتم‌ها

بهار ۱۴۰۱
استاد: مهدی صفرنژاد
گردآورندگان: محمدرضا دویران، زهره عباسی، کسری امانی
بررسی و بازبینی: کهبد آیینی

SHARIF UNIVERSITY OF TECHNOLOGY
دانشگاه صنعتی شریف
دانشکده‌ی مهندسی کامپیوتر

پاسخنامه امتحان میانترم

سوالات (۱۰۰ نمره)

۱. • الف) فرض کنید $n = ۲^m$

$$T(۲^m) = ۸T(۲^{\frac{m}{۲}}) + m^۳$$

$$S(m) = T(۲^m) \Rightarrow S(m) = ۸S(\tfrac{m}{۲}) + m^۳$$

مطابق قضیه‌ی اصلی چون $\log_۲ ۸ = ۳$:

$$S(m) = \theta(m^۳ \log_۲ m) \Rightarrow m = \log_۲ n \Rightarrow T(n) = \theta((\log_۲ n)^۳ \log_۲ \log_۲ n)$$

• ب) حدس می‌زنیم $T(n) = \mathcal{O}(n)$. از استقرا استفاده می‌کنیم. می‌دانیم برای همه‌ی $c \geq ۱$ پایه‌ی $T(n) = ۱ \leq cn$ برای همه‌ی $n \leq ۵$ برقرار است. داریم:

$$T(n) = T(\tfrac{n}{۲}) + T(\tfrac{n}{۴}) + n \leq \tfrac{cn}{۲} + \tfrac{cn}{۴} + n = (\tfrac{۳c}{۴} + ۱)n$$

اگر $c = ۴$ را انتخاب کنیم آنگاه $T(n) \leq ۴n = cn$ می‌شود. پس برای همه‌ی $c \geq ۴$ و $n \geq ۱$ $\Leftarrow$ $T(n) \leq cn$ $\Leftarrow$ $T(n) = \mathcal{O}(n)$

• ج)

$$nT(n) = \sum_{i=۱}^{n-۱} T(i) + ۳n^۲$$

$$(n+۱)T(n+۱) = \sum_{i=۱}^{n} T(i) + ۳(n+۱)^۲$$

دو عبارت را از هم کم می‌کنیم :

$$(n+۱)T(n+۱) - nT(n) = T(n) + ۶n + ۳$$
$$\Rightarrow T(n+۱) = \frac{nT(n)+T(n)}{n+۱} + \frac{۶n+۳}{n+۱} = T(n) - \frac{۳}{n+۱} + ۶$$
$$\Rightarrow T(n) = T(n-۱) - \frac{۳}{n} + ۶ \Rightarrow T(n) = T(n-۲) + \frac{۳}{n-۱} + ۶ + \frac{۳}{n} + ۶$$
$$\Rightarrow T(n) = ۶(n-۲) + ۳(\frac{۱}{n} + \frac{۱}{n-۱} + ... + \frac{۱}{۲})$$
$$\Rightarrow T(n) = \mathcal{O}(\max(n, \log_۲ n))$$
$$\Rightarrow T(n) = \mathcal{O}(n)$$

۲. ابتدا توپ‌ها را مرتبا به دو بخش تقسیم می‌کنیم تا به آرایه‌های دوتایی برسیم. سپس به صورت بازگشتی آن‌ها را با هم مقایسه می‌کنیم. اگر بخش‌های تقسیم شده را کیسه‌های حاوی توپ در نظر بگیریم می‌دانیم در صورت وجود داشتن عضو اکثریت کیسه‌ای که حداقل $\frac{n}{۲}$ عضو داشته باشد شامل مجموعه‌ی جواب ما خواهد بود. چون در هنگام بازگشت فقط یک عضو از هر دو کیسه را با هم مقایسه کردیم دیگر نیاز به مقایسه‌ی تک به تک عضوها با هم نخواهد بود. رابطه‌ی بازگشتی سوال به صورت زیر است که $\mathcal{O}(n)$ برای مرحله‌ی ادغام است:

$T(n) = ۲T(\frac{n}{۲}) + O(n)$

مطابق با قضیه‌ی اصلی پیچیدگی الگوریتم از $\mathcal{O}(n \log n)$ می‌شود.

۳. •

Algorithm ۱: MULTIPOP(S، k)

```
۱ while not stack EMPTY and k > ۰ do
۲ |   POP(S)؛
۳ |   k ⟵ k − ۱؛
۴ end
```

• می‌دانیم که تعداد POPهای اجرا شده (شامل آن‌هایی که در فرایند اجرای MULTIPOP فراخوانی شده‌اند) حداکثر برابر تعداد PUSHها خواهد بود (زیرا استک در ابتدا خالی بوده) بنابراین اگر هزینه اجرا PUSH و POP از $O(۱)$ باشد، حداکثر هزینه یک دنباله nتایی از روال‌های PUSH, POP, MULTIPOP برابر با $O(n)$ خواهد بود و بدیهی است که هزینه سرشکن هر روال نیز $O(۱)$ می‌باشد.

۴. به جای پیدا کردن k امین عنصر و پس از آن مقایسه آن با x سعی می کنیم که با پیمایش هرم k عنصر کوچکتر از عدد x را بیابیم. اگر موفق به این کار بشویم نتیجه میگیریم که عنصر k امین کوچکترین عنصر از x کوچکتر است. اگر نتوانیم در پیمایش خود k عنصر را بیابیم و همه عناصر کوچکتر از x را ویزیت کرده باشیم که تعدادشان کمتر از k است، در این صورت k امین کوچکترین عنصر از x بزرگتر است.

$def\ \ function(node, x, k):$
    $if\ node.value >= x:$
        $return\ \ ۰$
    $counter = ۱$
    $if\ counter == k:$
        $return\ \ k$
    $if\ node.left:$
        $counter += \ function(node.left,\ x,\ k - counter)$
    $if\ counter == k:$
        $return\ \ k$
    $if\ node.right:$
        $counter += \ function(node.right,\ x,\ k - counter)$
    $return\ \ counter$

حال اگر خروجی این تابع نهایتا k باشد پس x از k امین عنصر کوچک هرم بزرگتر است و اما در غیر این صورت این مورد برقرار نیست. برای هر یک از k راس کوچکتر از x حداکثر دو فرزند داریم که تابع بازگشتی برای آن ها اجرا می شود اما جز رئوس کوچکتر از x نیستند. پس در نهایت حداکثر برای پیدا کردن k راس k۳ بار تابع صدا زده می شود که هرکدام از $O(۱)$ می باشد پس در نهایت پیچیدگی زمانی نهایی ما از $O(k)$ خواهد بود.

۵. ابتدا کوچکترین جد مشترک LCA را پیدا می‌کنیم و برای این کار از ریشه شروع میکنیم. اگر از هر دو مقدار a،b بیشتر بود تابع LCA را به صورت بازگشتی بین فرزند چپ $root$ و مقادیر a و b صدا زده می‌شود در غیر این صورت اگر $root$ از هر دو مقدار کوچکتر بود تابع مورد نظر به صورت بازگشتی بین فرزند راست و اعداد

a و b صدا زده می‌شود و در صورتی که $root$ بین a و b باشد کوچکترین جد مشترک ما همین است. بدیهی است که این کار از پیچیدگی زمانی $O(h)$ است. حال تابع $max\_element\_in\_path$ را تعریف می‌کنیم که یکبار برای (b, LCA) پیدا شده انجام می‌شود و یکبار برای (a, LCA) و با استفاده از این تابع ماکزیمم بین یک گره و گرهی از جدش پیدا می‌شود و این دو مقدار را مقایسه کرده و ماکسیمم این دو مقدار پاسخ نهایی ما است که پیاده سازی آن به صورت زیر است:

```
max_element_in_path(x, node)
    maximum = max(value, node.value)
    while(node.value! = x)
        if(node.value > x)
            maximum = max(x, node.value)
            node = node.left
        else
            maximum = max(x, node.value)
            node = node.right
    return max(maximum, x)
```

بدیهی است که این تابع برای $max\_element\_in\_path(a, lca)$ نهایتا از پیچیدگی زمانی $height(a) - height(lca)$ و برای $max\_element\_in\_path(b, lca)$ نهایتا از پیچیدگی زمانی $height(b) - height(lca)$ می‌باشد. پس حل این مسئله با پیچیدگی زمانی $O(h)$ امکان پذیر است.

۶. در هر رو صورت متود RANDOM تعداد $\Theta(n)$ بار فراخوانی خواهد شد اما تعداد فراخوانی‌های متود PARTITION در بهترین حالت کمتر خواهد بود زیرا اندازه ورودی کوچک‌تر است. درک این مطلب با مشاهده pseudocode الگوریتم ساده‌تر است:

Algorithm ۲: RANDOMIZED-PARTITION(A،p،r)

$i \longleftarrow RANDOM(p, r)$؛
exchange A[r] with A[i]؛
return PARTITION(A،p،r)؛

Algorithm ۳: RANDOMIZED-QUICKSORT(A،p،r)

if $p < r$ **then**
    q =RANDOMIZED-PARTITION(A،p،r)
    RANDOMIZED-QUICKSORT(A،p،q-۱(
    RANDOMIZED-QUICKSORT(A،q+۱،r)
end

موفق باشید.