دانشکده علوم ریاضی و آمار

مدرس: دکتر مجتبی رفیعی نیمسال اول ۱۴۰۰-۱۴۰۱

ساختمان داده‌ها و الگوریتم‌ها

جلسه ۳۷

نگارنده: حانیه ارفع الرفیعی

۱۰ دی ۱۴۰۰

فهرست مطالب

۱ تحلیل پیچیدگی زمانی راه حل ۲

$$T(n) = \sum_{h=0}^{\log\ n} h * 2^{(\log\ n)-h}$$

جایی‌که $h = 0$ منظور سطح اخر است وبه سمت ریشه حرکت می‌کنیم (ریشه $h = logn$) و هر سطح h نیز $2^{(\log\ n)-h}$ نود داریم.

$$
\begin{aligned}
T(n) &= \sum_{h=0}^{\log\ n} h * 2^{(\log\ n)-h} \\
&= \sum_{h=0}^{\log\ n} h * n/2^h = n \sum_{h=0}^{\log\ n} h/2^h = n * A^*
\end{aligned}
$$

ادعا می‌کنیم که A^* یک عدد ثابت است و در نتیجه $T(n) = O(n)$ می‌باشد.

یاد آوری: اگر $|a| < 1$ باشد انگاه رابطه $\sum_{k=0}^{\infty} a^k = \frac{1}{1-a}$ برقرار است.

با توجه به مطلب مذکور داریم:

$$A^* <= 1/2 + 2/2^2 + 3/2^3 + 4/2^4 + ...$$

می دانیم که:

$$1/2 + (1/2)^2 + (1/2)^3 + (1/2)^4 + ... = 1$$

$$(1/2)^2 + (1/2)^3 + (1/2)^4 + ... = 1 - 1/2 = 1/2$$

$$(1/2)^3 + (1/2)^4 + ... = 1/2 - 1/4 = 1/4$$

و روال بالا می‌تواند به همین منوال ادامه داشته باشد.

$$A^* = \sum_{k=0}^{\infty} (1/2)^k = 2$$

در نتیجه، رابطه بازگشتی مذکور از مرتبه $T(n) = 2n = O(n)$ می‌باشد.

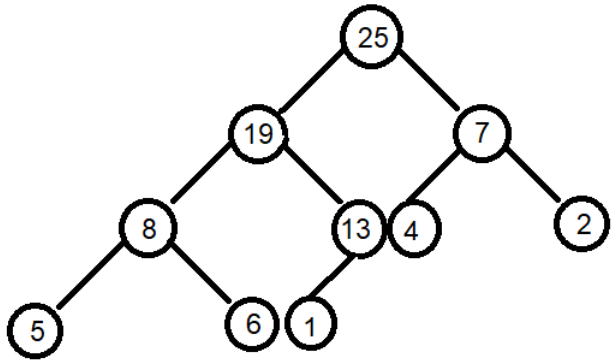
نکته مهم: ایده راه حل دوم به ما کمک کرد jh به ازای هر عنصر، $\log\ n$ هزینه نکنیم، و به عبارت دیگر:

$$
\begin{aligned}
\text{۱عنصر} &\rightarrow \log\ n \\
\text{۲عنصر} &\rightarrow (\log\ n) - 1 \\
\text{۴عنصر} &\rightarrow (\log\ n) - 2 \\
&\vdots\ \vdots
\end{aligned}
$$

۲ عمل افزایش محتوای یک عنصر

در این عملیات می خواهیم اگر گره یا کلید key مقداری کمتر از x دارد، مقدار ان را به x افزایش دهیم.

مثال: درخت هرم بیشینه زیر را در نظر بگیرید:



به سوالات زیر به دقت پاسخ دهید.

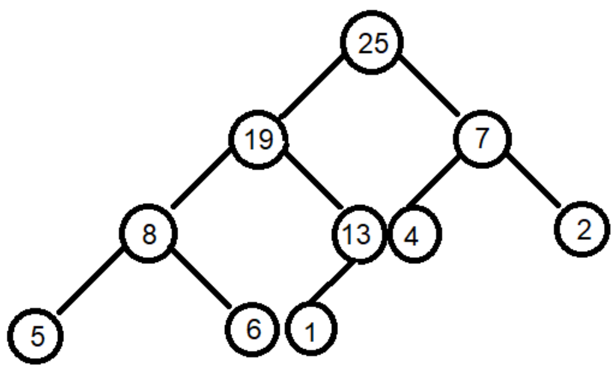
- فراخوانی(۸و۱۰) Increase چه تغییری در درخت Max Heap بالاایجاد می کند؟
- فراخوانی(۸و۷) Increase چه تغییری ایجاد می کند؟
- فراخوانی(۸و۲۷) Increase چه تغییری ایجاد می کند؟

جمع بندی: متناسب با مقدار x از نود حاوی کلید key تا ریشه می‌بایست ساختار هرم بیشینه رابروز رسانی کنیم. پیچیدگی این عملیات از مرتبه $O(\log n)$ است، جایی که n تعداد نودهای درخت هرم بیشینه است.

۳ عملیات درج عنصر در Max Heap

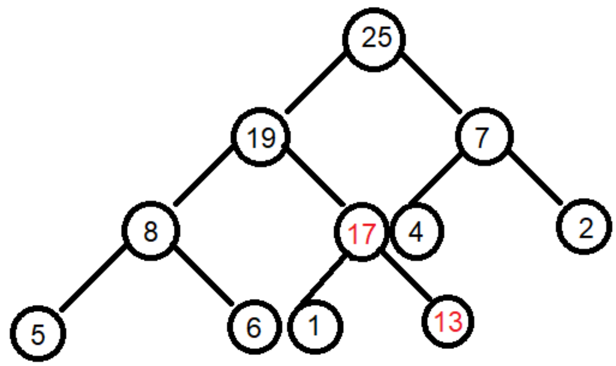
دراین عملیات می‌خواهیم عنصر جدید key را به نحوی به درخت Max Heap اضافه کنیم که درخت حاصل همچنان Max Heap بماند.

مثال: درخت Max Heap زیر را در نظربگیرید:



سوال: فراخوانی (۱۷)Insert چه تغییری در درخت Max Heap بالا ایجاد می‌کند؟

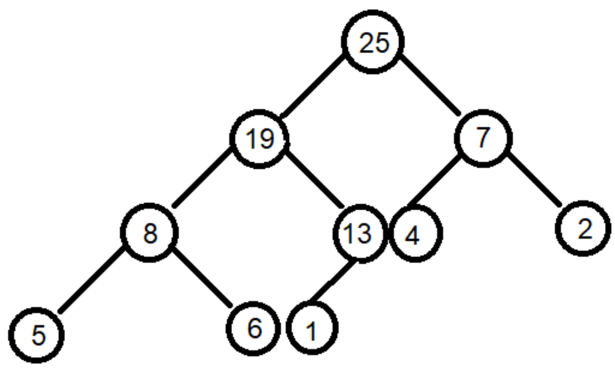
ایده: یک نود با کلید $-\infty$ درسطح اخر و به عنوان سمت راست‌ترین نود ایجاد می‌کنیم و اشاره‌گر پدر ان را بطور مناسبی بروز می‌کنیم. سپس با فراخوانی Increase (17، $-\infty$) ، درخت Max Heap نهایی ساخته می‌شود. درخت حاصل از Insert (17) بصورت زیر می‌شود:



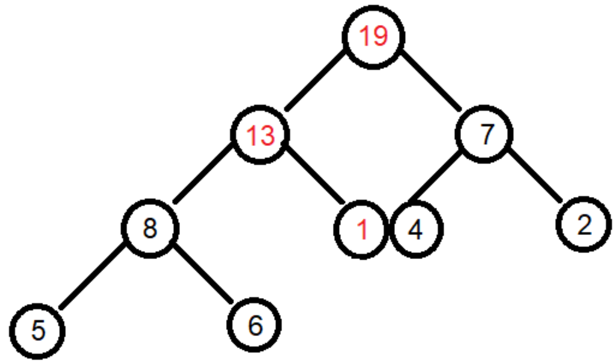
۴ عملیات حذف عنصر Max از درخت Max Heap

در این عملیات می خواهیم عنصر بیشینه را به نحوی حذف کنیم که درخت حاصل همچنان Max Heap بماند.

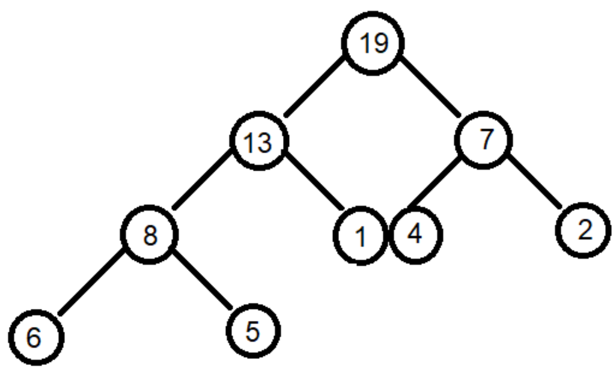
مثال: درخت Max Heap زیر را در نظر بگیرید:

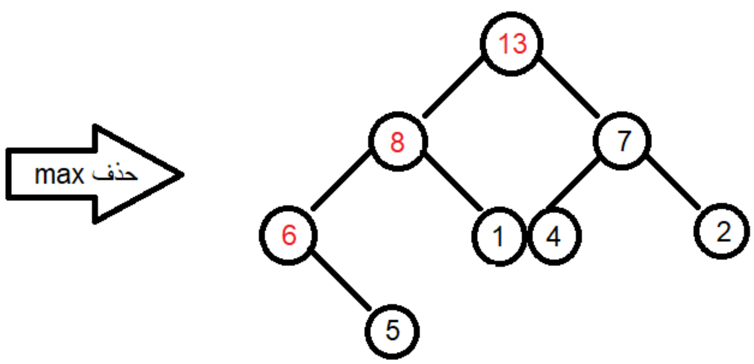


راه حل پیشنهادی ۱: ابتدا ریشه را حذف می‌کنیم و سپس از ریشه به سمت برگ‌ها حرکت کرده و در صورت نیاز برای حفظ ویژگی Heap Max بودن، عنصر ماکزیمم رابه سطح بالاتر انتقال می‌دهیم.



نکته: راه حل پیشنهادی ۱ ضمانتی برای ویژگی تقریبا کامل بودن یا رعایت پر بودن از سمت چپ به راست نودها راندارد. در این راستا به مثال زیر دقت کنید:

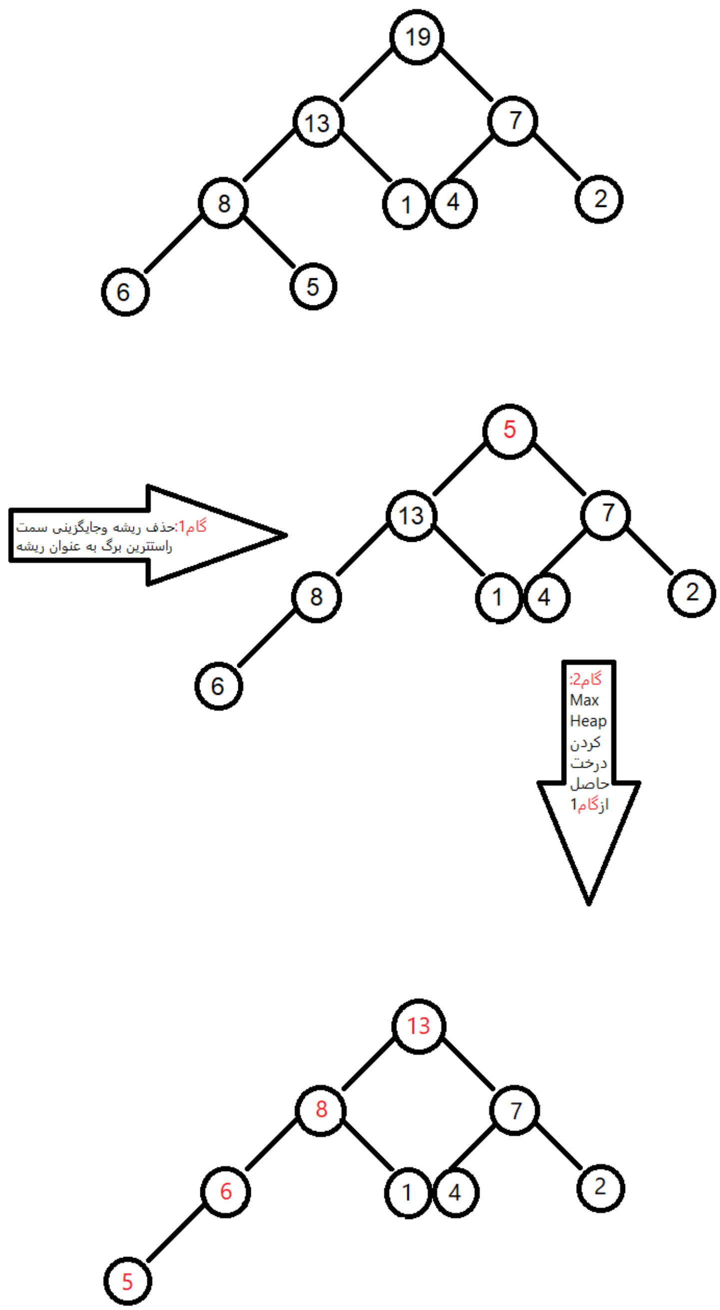




به وضوح درخت بالا، درخت Max Heap نیست چراکه ساختار Max Heap را رعایت نکرده است.

راه حل ۲: ابتدا ریشه را حذف کرده وبا سمت راست ترین برگ جایگزین می‌کنیم، سپس از ریشه تابرگ Max Heap بودن را بررسی و در صورت نیاز جابجایی‌های لازم را انجام می‌دهیم.

مثال: درخت Max Heap زیر را درنظر بگیرید:



نکته: پیچیدگی زمانی الگوریتم مربوط به راه حل ۲ ازمرتبه $O(\log\ n)$ است.

۵ پیاده سازی درخت Max Heap

با توجه به خصیصه‌های ساختاری Max Heap (تقریبا کامل بودن و پر شدن از چپ به راست)، لیست یک گزینه مناسب برای پیاده‌سازی درخت Max Heap است. به عنوان مثال برای هر نود در اندیس i لیست، فرزندان آن در اندیس‌های $2i$ و $2i+1$ قرار می‌گیرند و هدر رفت حافظه‌ای هم اتفاق نمی‌افتد.