ساختمان داده
دانشکده ریاضی. دانشگاه صنعتی خواجه نصیرالدین طوسی. پاییز ۱۴۰۲

۱ صف اولویت با ادغام meld

می‌خواهیم یک ساختار داده انتزاعی برای صف اولویت تعریف کنیم که علاوه بر اعمال اصلی درج عنصر جدید، حذف عنصر کمینه و پیدا کردن عنصر کمینه، عمل ادغام دو صف را هم پشتیبانی کند. لذا اینجا فرض ما بر این است که مجموعه‌ای از صفها داریم $P_1, P_2, \cdots, P_k$ و می‌خواهیم عمل ادغام $\text{meld}(P_i, P_j)$ را پشتیبابی کنیم. در ادغام دو صف، عناصر دو صف در یک صف تجمیع می‌شوند. به مثال زیر توجه کنید.

P1 = { 3, 4, 4, 5, 6, 12, 13, 13}

P2 = {5, 5, 12, 14}

P3 = meld(P1,P2)

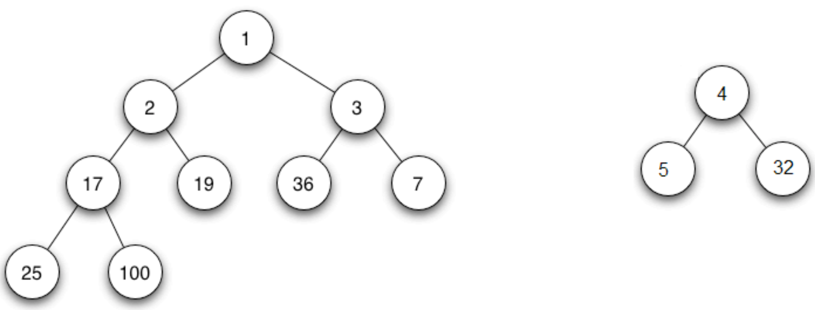
P3 = {3, 4, 4, 5, 5, 5, 6, 12, 13, 13, 14}

۱.۱ اعمال اصلی ساختار داده انتزاعی صف اولویت با ادغام

- $\text{Enqueue}(P_i, v)$
  درج عنصر v در صف P_i
- $\text{Find-Min}(P_i)$
  برگرداندن عنصر کمینه صف P_i
- $\text{Extract-Min}(P_i)$
  حذف عنصر کمینه صف P_i
- $\text{Meld}(P_i, P_j)$ ادغام دو صف اولویت P_i و P_j

۲.۱ هرم باینری و مسئله ادغام

قبلا صف اولویت را با استفاده از هرم باینری Binary Heap پیاده‌سازی کرده ایم. برای یک هرم باینری با n عنصر درخت مربوطه شکل کاملا مشخصی دارد و می‌دانیم که زمان درج در صف و حذف عنصر کمینه از مرتبه $\Theta(\log n)$ است. پیدا کردن عنصر کمینه در زمان $O(1)$ انجام می‌شود. متاسفانه، با توجه به ساختار غیر منعطفی که هرم باینری دارد، مشخص نیست که چگونه می‌توان دو هرم باینری با اندازه‌های مختلف را در زمان سریع ادغام کرد. بنظر می‌رسد اگر بخواهیم دو هرم باینری با اندازه n را ادغام کنیم باید زمان $O(n)$ صرف این کار کنیم.

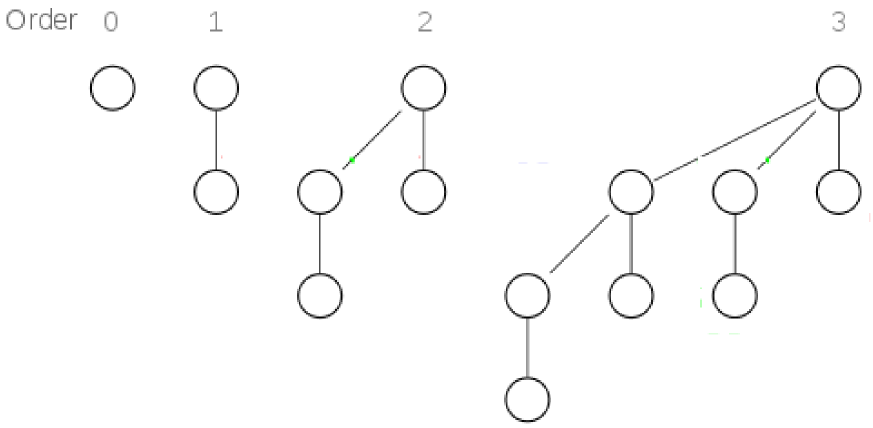


۲ درخت دوجمله‌ای

با هدف پیاده‌سازی سریع ادغام، هرم باینری را کنار می‌گذاریم و از یک ساختار دیگر استفاده می‌کنیم. درخت دو جمله‌ای Binomial Tree یک گزینه مناسب برای این منظور است. درخت دوجمله‌ای از مرتبه k یک تعریف بازگشتی دارد.

تعریف: درخت دوجمله‌ای از مرتبه k درختی ریشه‌دار و مرتب است که فرزندان ریشه آن به ترتیب (از راست به چپ) خود درختان دوجمله‌ای از مرتبه 0 و 1 و 2 تا $k-1$ هستند. درخت دوجمله‌ای از مرتبه 0 تنها یک راس دارد که ریشه درخت است.

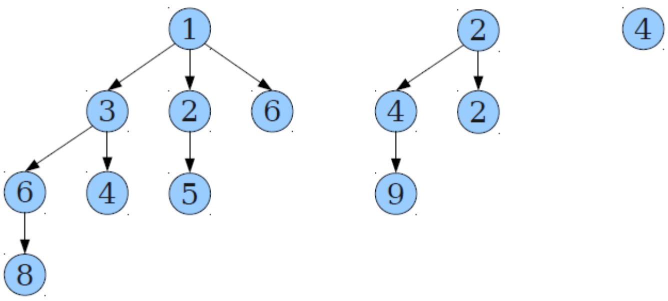
در زیر چند درخت دوجمله‌ای از مرتبه های مختلف نشان داده شده است.



مشاهده: یک درخت دوجمله‌ای از مرتبه k به تعداد 2^k راس دارد.

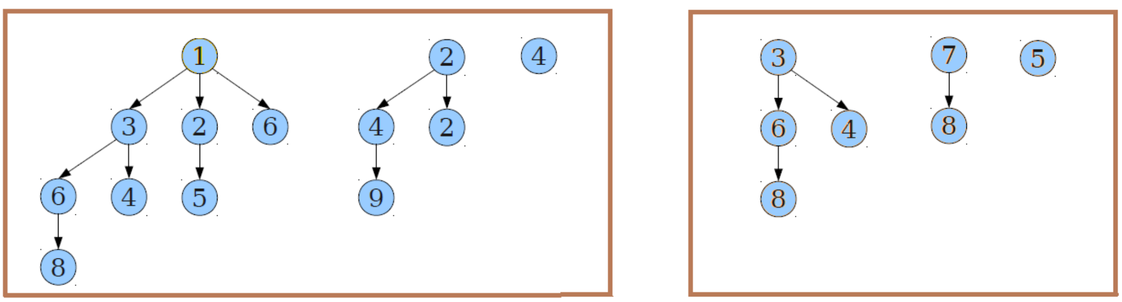
۳ هرم دوجمله‌ای: ادغام حریصانه

۱. ساختار داده هرم دوجمله‌ای از مجموعه‌ای از درختان دوجمله‌ای تشکیل شده است بطوریکه عناصر هر درخت از ساختار هرم پیروی می‌کنند. به عبارت دیگر، مقدار ذخیره شده در هر راس از فرزندان کوچکتر است. یک مثال در شکل زیر نشان داده شده است.



۲. در هرم دوجمله‌ای این محدودیت را اعمال می‌کنیم که از هر مرتبه فقط یک درخت دوجمله‌ای داشته باشیم.

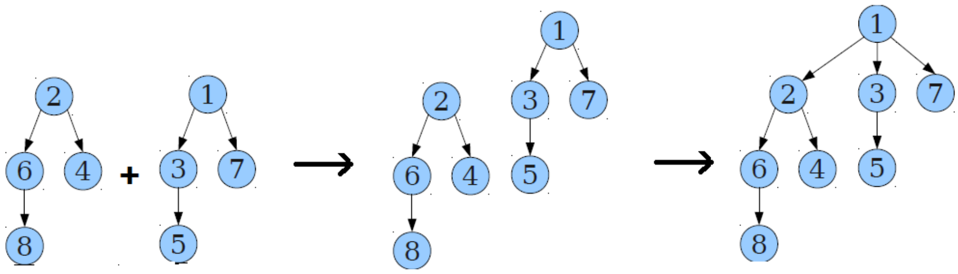
۳. می‌توانیم مجموعه‌ای از صفها داشته باشیم. هر صف عنصر کمینه خود را دارد.



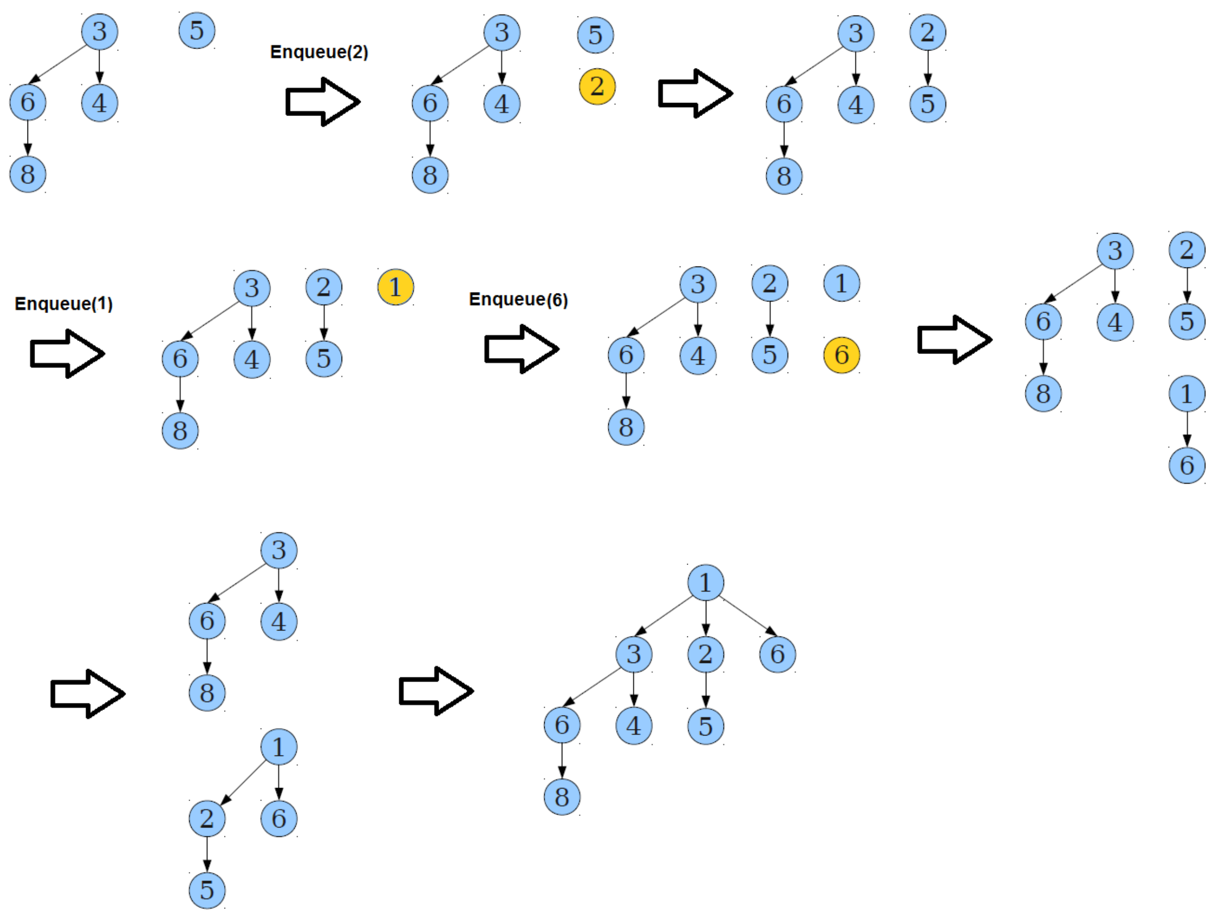
۱.۳ پیاده‌سازی اعمال اصلی

- $\mathrm{Enqueue}(P_i, v)$

عنصر جدید بصورت یک درخت دوجمله‌ای از مرتبه صفر اضافه می‌شود. اگر قبلا یک درخت دوجمله‌ای از مرتبه صفر داشته باشیم باید با عنصر جدید ادغام شوند تا از یک مرتبه دو درخت نداشته باشیم. این خود می‌تواند باعث ایجاد موجی از ادغامهای دیگر شود. ادغام هرم دوجمله‌ای هم مرتبه به راحتی قابل انجام است. به مثال زیر توجه کنید. درخت با ریشه بزرگتر فرزند ریشه درخت با ریشه کوچکتر می‌شود. بدین ترتیب یک درخت دوجمله‌ای از مرتبه بالاتر بدست می‌آید. این کار در زمان $O(1)$ قابل انجام است.



یک نمونه از درج عنصر جدید در هرم دوجمله‌ای در شکل زیر نشان داده شده است. توجه کنید که یک هرم دوجمله‌ای با n عنصر حداکثر $\log n$ درخت دارد. لذا پیرو درج یک عنصر، حداکثر $\log n$ ادغام رخ می‌دهد. پس زمان درج $O(\log n)$ است.

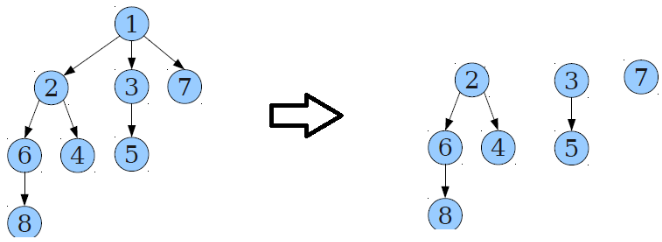


- $\mathrm{Find\text{-}Min}(P_i, v)$

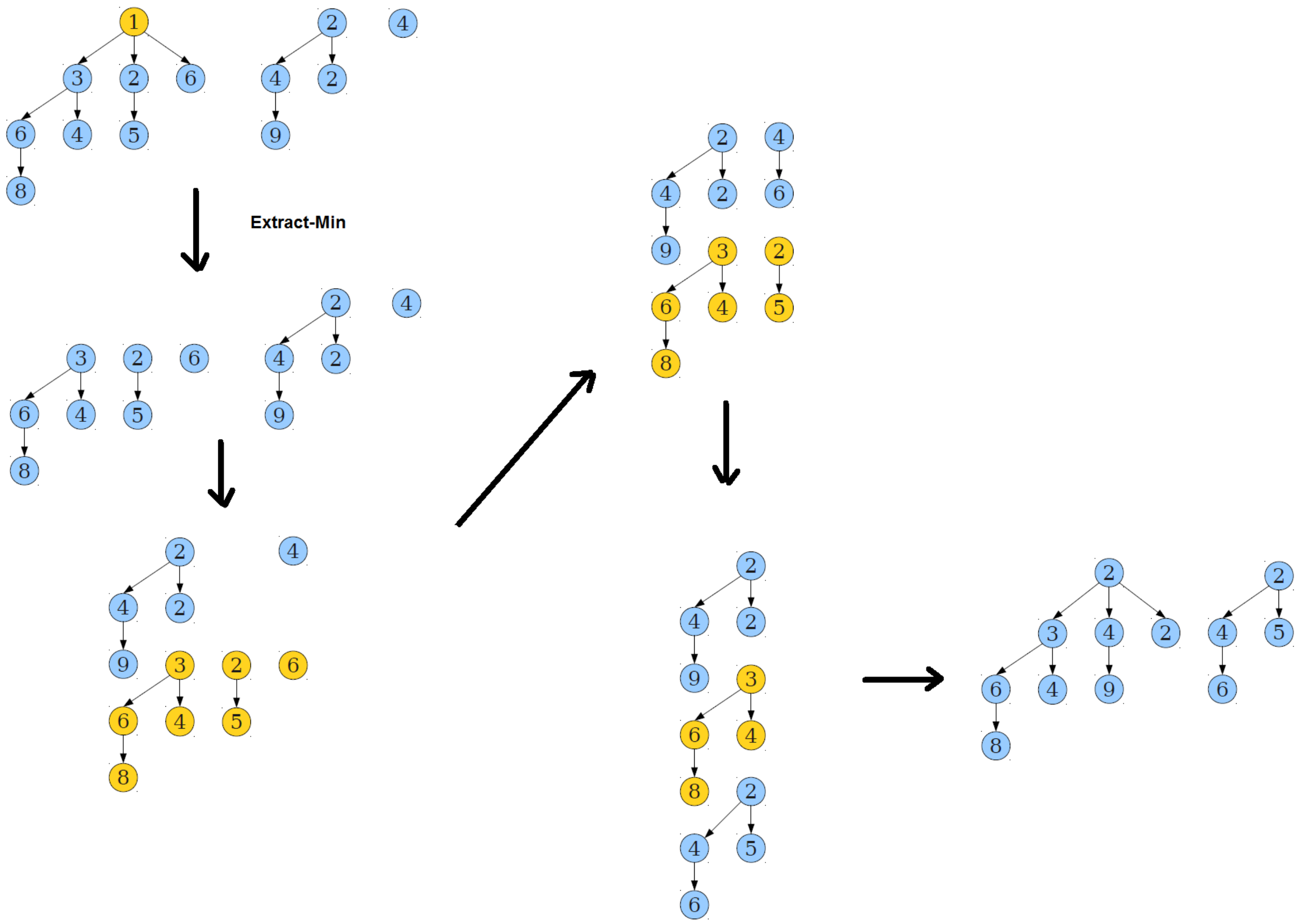
برای پیدا کردن عنصر کمینه باید ریشه همه درختها را چک کنیم. از آنجا که حداکثر $\log n$ درخت داریم، عنصر کمینه در زمان $O(\log n)$ قابل اجرا است.

- Extract-Min(P_i, v)

با حذف ریشه درخت دو جمله‌ای از مرتبه k، درختان دوجمله‌ای مجزا از مرتبه‌های صفر، یک، دو ... تا $k-1$ ایجاد می‌شوند.



درختان دو جمله‌ای حاصل از حذف ریشه، در صورت لزوم، باید با درختان دیگر ادغام شوند. اگر n عنصر داشته باشیم، با توجه به اینکه حداکثر $\log n$ درخت دوجمله‌ای داریم، ادغامهای بعد از حذف ریشه حداکثر $O(\log n)$ عدد خواهد بود. این مانند این است که دو عدد باینری با $\log n$ رقم را با هم جمع کنیم. لذا زمان حذف عنصر کمینه در زمان $O(\log n)$ قابل پیاده‌سازی است.

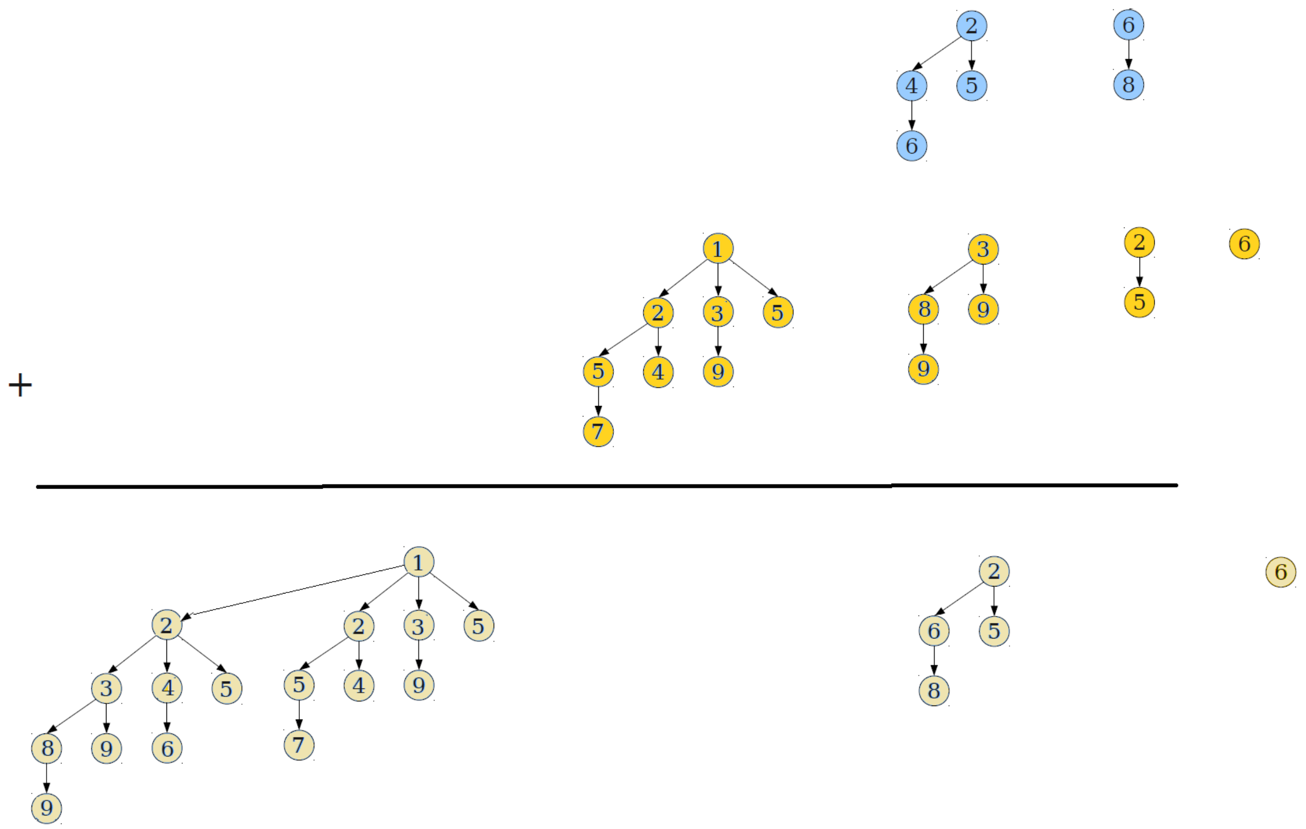


$$
\begin{array}{cccccc}
 & & 1 & 1 & 1 & \\
 & & 0 & 1 & 0 & 1 \\
+ & & 0 & 1 & 1 & 1 \\
\hline
 & & 1 & 1 & 0 & 0
\end{array}
$$

- Meld(P_i, P_j)

بطور مشابه، ادغام دو هرم دوجمله‌ای که هر کدام حداکثر n عنصر دارند مانند جمع دو عدد باینری با $\log n$ رقم است. لذا اینجا هم حداکثر $O(\log n)$ درخت دوجمله‌ای ادغام می‌شوند و لذا در بدترین حالت زمان اجرا $O(\log n)$ است.

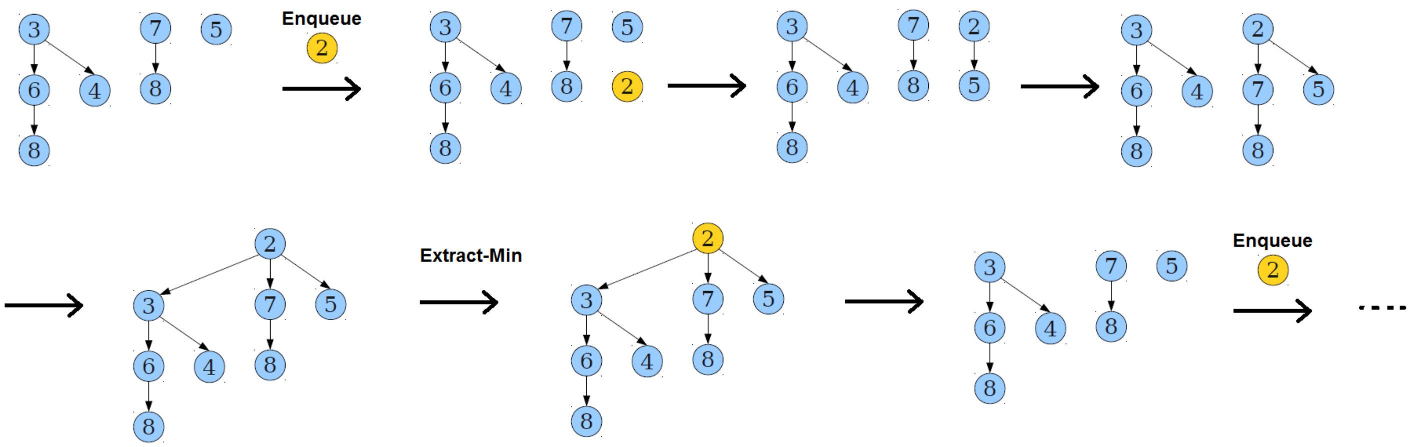
$$
\begin{array}{cccccc}
 & 1 & 1 & 1 & & \\
 & & 0 & 1 & 1 & 0 \\
+ & & 1 & 1 & 1 & 1 \\
\hline
 & 1 & 0 & 1 & 0 & 1
\end{array}
$$



۲.۳ تحلیل سرشکنی هرم دو جمله‌ای، ادغام حریصانه

از آنجا که تناظر مستقیمی میان عمل درج در هرم دوجمله‌ای Enqueue و عمل افزایش Increment در شمارنده باینری وجود دارد، از آنجا که زمان n عمل متوالی Increment در شمارنده باینری $O(n)$ است پس می‌توان گفت که زمان انجام n عمل متوالی درج در هرم دوجمله‌ای نیز $O(n)$ است. این را با هرم باینری مقایسه کنید که آنجا زمان n عمل متوالی درج $O(n \log n)$ بود.

از طرف دیگر، اگر بخواهیم زمان اجرای n عمل متوالی که ترکیبی از Enqueue و Extract-Min است تحلیل کنیم، نمی‌توان گفت که این از مرتبه $O(n)$ است. دقیقا مشابه شمارنده باینری است جایی که اجرای متوالی n عمل Increment و Decrement زمان اجرایش $\Theta(nk)$ شد و نه $O(n)$. حالتی را تصور کنید که Extract-Min و Enqueue پشت سر هم اجرا می‌شوند. زمان اجرا می‌تواند $O(n \log n)$ باشد. به مثال زیر توجه کنید.



در این مثال هر عمل Enqueue بطور موج وار باعث رخداد $\log n$ ادغام می‌شود له لذا زمانش $O(\log n)$ است. هر عمل Extract-Min ساختار داده را به حالت قبل از Enqueue برمی‌گرداند.

جدول زیر زمان اجرای اعمال اصلی ساختار داده هرم دوجمله‌ای (با ادغام حریصانه) را نشان می‌دهد.

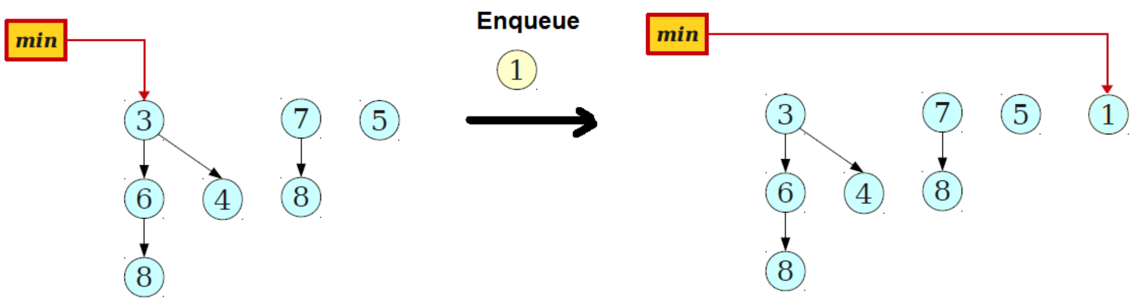
	Enqueue	Find-Min	Extract-Min	Meld
Worst-Case	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$
Amortized	$\Theta(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

پرسش: آیا می‌توان تغییراتی در ساختار داده ایجاد کرد بطوریکه زمان سرشکنی Enqueue کم باشد (مثلا $O(1)$ باشد) فارغ از اینکه عمل Extract-Min اجرا بشود یا نشود؟ جواب این سوال مثبت است که در قسمت بعد به آن می‌پردازیم. البته برای بدست آوردن این نتیجه هزینه‌ای پرداخت می‌کنیم. زمان Extract-Min در بدترین حالت به $O(n)$ افزایش پیدا می‌کند.

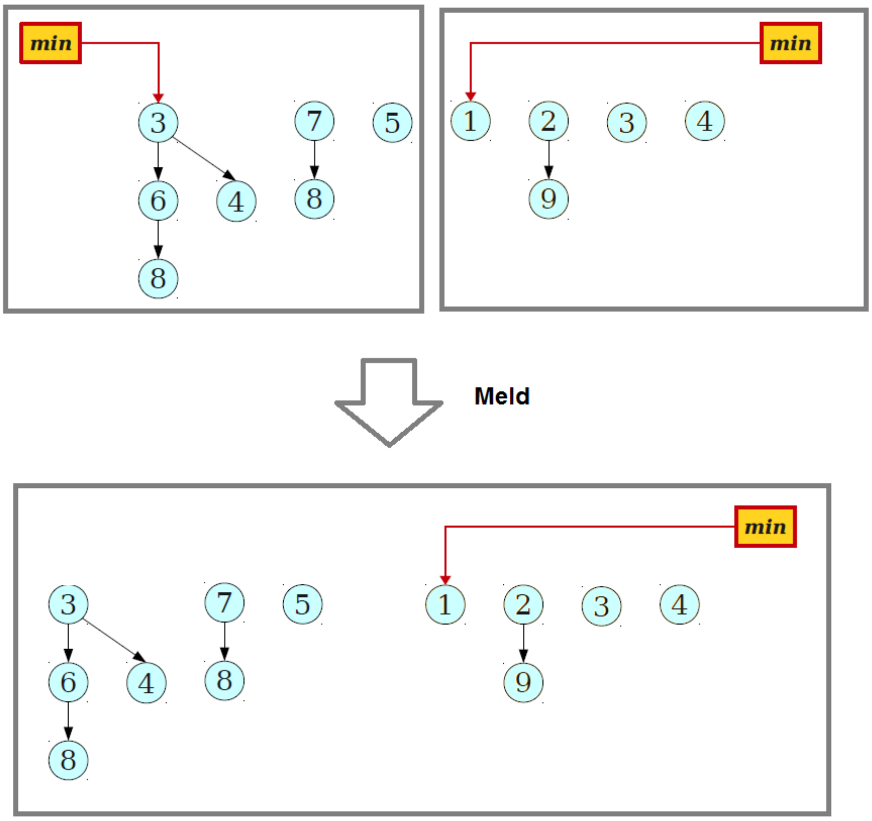
۴ هرم دوجمله‌ای: استراتژی ادغام با تاخیر

در استراتژی ادغام حریصانه، دلیل اصلی اینکه عمل Enqueue زمان $O(\log n)$ دارد، این است که ما اصرار داریم از یک اندازه حداکثر یک درختان دوجمله‌ای داشته باشیم. به عبارت دیگر، درختان دوجمله‌ای موجود باید اندازه‌های متفاوت داشته باشند. مزیت این محدودیت این است که (وقتی تعداد عناصر n باشد) باعث می‌شود که حداکثر $\log n$ درخت دوجمله‌ای داشته باشیم و در نتیجه Find-Min در زمان $O(\log n)$ قابل انجام باشد

حال فرض کنید محدودیت از هر درخت دوجمله‌ای از مرتبه k حداکثر ۱ عدد را برداریم. چقدر خوب! موقع Enqueue اصلا نیازی به انجام ادغامهای متوالی نداریم. متاسفانه این باعث می‌شود که تعداد زیادی درخت با اندازه 1 ایجاد شود و در نتیجه عمل Find-Min خیلی کند شود. اما جای نگرانی نیست. برای حل این مسئله، می‌توانیم یک اشاره‌گر به عنصر کمینه نگه‌داری کنیم. هر زمان که عمل Enqueue انجام می‌شود، اشاره‌گر بروزرسانی می‌شود و عمل Find-Min هم در زمان $O(1)$ قابل انجام خواهد بود. حتی بهتر از حالت قبل که $O(\log n)$ بود.



حتی موقع انجام Meld می‌توانیم همین کار را بکنیم. ادغامی انجام نمی‌دهیم. فقط اشاره‌گر به عنصر کمینه بروزرسانی می‌شود.

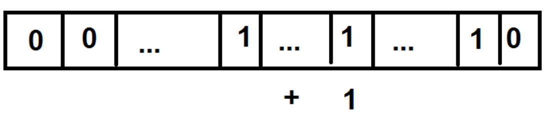


Extract-Mind را فراموش کردیم! وقتی عنصر کمینه حذف می‌شود، کمینه بعدی می‌تواند ریشه هر درختی باشد. تعداد زیادی درخت کوچک داریم. پس ناچاریم همه را چک کنیم. در بدترین حالت این عمل $O(n)$ زمان می‌برد. حالا که زمان $O(n)$ صرف این کار می‌کنیم چرا یک کار صواب هم انجام ندهیم و درختان هم اندازه را ادغام نکنیم؟ همین کار را انجام می‌دهیم. موقع Extract-Min وضعیت هرم دوجمله‌ای را سر و سامان می‌دهیم. درختهای هم اندازه را ادغام می‌کنیم بطوریکه از هر اندازه حداکثر یک درخت موجود باشد.

لم: فرض کنید k درخت دوجمله‌ای با اندازه‌های دلخواه داریم. با فرض اینکه اندازه هر درخت مشخص است، می‌توانیم t درخت را در زمان $O(k)$ در هم ادغام کنیم بطوریکه در نهایت از هر اندازه حداکثر یک درخت موجود باشد.

اثبات: فرض کنید $T_1, T_2, \cdots, T_k$ درختهای دوجمله‌ای داده شده باشد. در گذر اول، مقدار $n = \sum_{i=1}^{k} \text{size}(T_i)$ را محاسبه می‌کنیم. اینجا n تعداد کل عناصر موجود در همه درختهاست. درختهای داده شده را می‌توانیم در $\lceil \log n \rceil$ درخت دوجمله‌ای با اندازه‌های متفاوت ادغام کنیم. آرایه خالی A با $\lceil \log n \rceil$ را اندیس می‌سازیم. برای انجام این کار یک گذر دیگر روی درختها انجام می‌دهیم. فرض کنید $\text{order}(T)$ مرتبه درخت دوجمله‌ای T باشد. از $i = 1$ شروع می‌کنیم، T_i را در اندیس $\text{order}(T_i)$ از آرایه A قرار می‌دهیم. اگر اندیس مورد نظر قبلا توسط درخت دیگری مثلا T' اشغال شده باشد، باید یک عمل ادغام انجام دهیم. حاصل $T'' = \text{meld}(T_i, T')$ را در اندیس $\text{order}(T'')$ قرار می‌دهیم. اگر قبلا اشغال شده باشد باز هم ادغام انجام می‌دهیم. این پروسه ادامه پیدا می‌کند تا زمانیکه نیازی به ادغام وجود نداشته باشد.

یک تحلیل ساده می‌گوید چون پردازش هر درخت باعث ادغام شدن حداکثر $\log n$ درخت دوجمله‌ای می‌شود پس زمان الگوریتم حداکثر $O(k \log n)$ است. اما یک تحلیل بهتر وجود دارد. توجه کنید که در ابتدا آرایه A خالی است. این مانند شمارنده باینری با $\lceil \log n \rceil$ بیت است که در ابتدا صفر است. اضافه شدن درخت T_i با مرتبه $d = \text{order}(T_i)$ مانند اضافه کردن 2^d به مقدار شمارنده است.



چون شمارنده از مقدار صفر شروع می‌شود، اگر k بار یک توان دو را به شمارنده اضافه کنیم، زمان اجرا $O(k)$ خواهد بود.

این وضعیت تعمیمی از مسئله Increment در شمارنده باینری است. آنجا هر دفعه 2^0 را به مقدار شمارنده اضافه می‌کردیم. همان تحلیل تابع پتانسیل را می‌توان اینجا هم بکار برد و نتیجه مشابه گرفت. پس نتیجه می‌گیریم پردازش k درخت در زمان $O(k)$ قابل انجام است. $\square$

۱.۴ تحلیل اعمال اصلی هرم دوجمله‌ای: استراتژی ادغام با تاخیر

برای تحلیل سرشکنی از تابع پتانسیل زیر استفاده می‌کنیم.

$$\Phi(D) = \text{تعداد کل درختها در صفهای اولویت}$$

- $\text{Enqueue}(P_i, v)$

موقع درج عنصر جدید تنها درخت دوجمله‌ای دو جمله‌ای از مرتبه صفر با ریشه v ایجاد می‌شود. هیچ ادغامی انجام نمی‌شود. اشاره‌گر به عنصر کمینه در صورت لزوم بروزرسانی می‌شود. لذا زمان بدترین حالت $O(1)$ است. برای تحلیل سرشکنی داریم

$$a_{\text{Enquque}} = c_{\text{Enqueue}} + \Delta\Phi$$

$$a_{\text{Enquque}} \leq O(1) + 1 = O(1)$$

زمان سرشکنی Enqueue برابر با $O(1)$ است.

- $\text{Find-Min}(P_i)$ پیدا کردن عنصر کمینه تغییری در ساختار داده ایجاد نمی‌کند. با استفاده از اشاره‌گر min عنصر کمینه در زمان $O(1)$ انجام می‌شود. چون $\Delta\Phi = 0$ پس

$$a_{\text{Find-Min}} = c_{\text{Find-Min}} = O(1)$$

- $\text{Meld}(P_i, P_j)$

مشابه حالت Find-Min است. هیچ درختی ادغام نمی‌شود. مقدار پتانسیل تغییری نمی‌کند چون تعداد درختها ثابت باقی می‌ماند. فقط اشاره‌گر به عنصر کمینه بروز می‌شود که آن هم مقدار پتانسیل را تغییر نمی‌دهد. لذا داریم

$$a_{\text{Meld}} = c_{\text{Meld}} = O(1)$$

- $\text{Extract-Mind}(P_i)$

اینجا تنها جایی است که ادغام اتفاق می‌افتد. همانطور که گفته شد، اجرای Extract-Min روی صف اولویت P_i باعث می‌شود که درختان داخل صف با استفاده از رویه‌ای که در اثبات لم قبلی توضیح داده شده ادغام شده و تعدادشان به حداکثر $\log n$ برسد. اگر t تعداد درختان قبل از اجرای عمل باشد، داریم

$$\Delta\Phi \leq \log n - t$$

زمان اجرا $c_{\text{Extract-Min}}$ اینجا حداکثر bt است وقتی b یک عدد ثابت است (پیرو لم قبلی). لذا داریم

$$a_{\text{Extract-Min}} = c_{\text{Extract-Min}} + \Delta\Phi$$
$$a_{\text{Extract-Min}} \leq bt + \log n - t$$

برای کنسل شدن bt، اینجا یک تغییر کوچک در تابع پتانسیل می‌دهیم. تعریف می‌کنیم

$$\Phi'(D) = b \times (\text{تعداد کل درختها})$$

با این تابع پتانسیل جدید داریم

$$a_{\text{Extract-Min}} \leq bt + b(\log n - t) \leq b \log n = O(\log n)$$

دقت کنید با فرض تابع پتانسیل جدید، تحلیلهای قبلی کماکان درست است. یعنی زمان سرشکنی Enqueue و Find-Min و Meld کماکان $O(1)$ است.

در جدول زمان اعمال اصلی استراتژی ادغام با تاخیر آمده است.

	Enqueue	Find-Min	Extract-Min	Meld
Worst-Case	$O(1)$	$O(1)$	$O(n)$	$O(1)$
Amortized	$O(1)$	$O(1)$	$O(\log n)$	$O(1)$