نُه: هرم (Heap)

ساختمان داده ها و الگوریتم

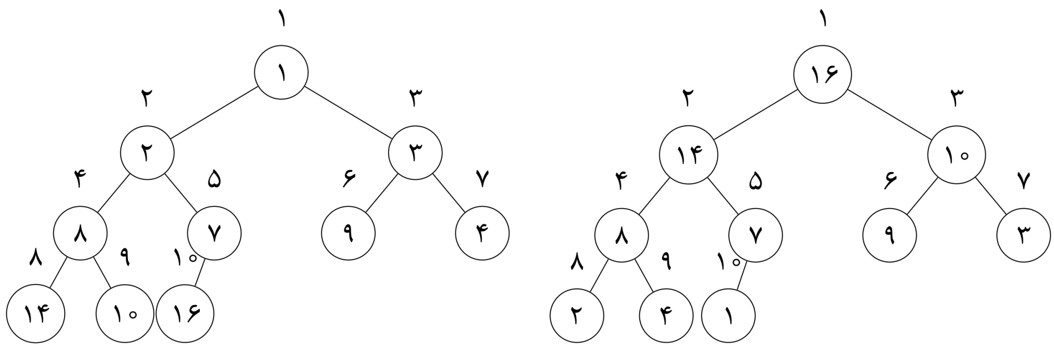
مدرس: دکتر نجمه منصوری

نگارنده: سجاد هاشمیان

۱. تعریف

هرم یک داده‌ساختار بر مبنای درخت دودویی تقریبا کامل است که دارای این خاصیت است که رابطه کوچکتر یا بزرگتری بین کلید گره‌های پدر و فرزند در طول درخت ثابت است. به این خاصیت، ویژگی هرم می‌گویند. بسته به رابطه بزرگتری و یا کوچکتری، هرم بیشینه یا کمینه خوانده می‌شود.

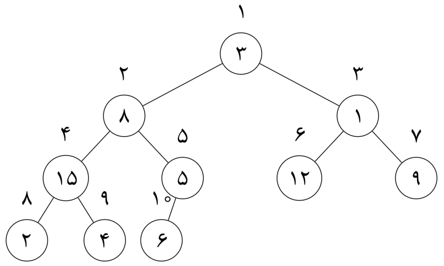
به طور دقیق‌تر هرم بیشینه و هرم کمینه به این‌صورت‌اند که اگر در یک درخت کلید هر گره از کلید فر زندانش بزرگتر باشد درخت دارای ویژگی هرم بیشینه و اگر کلید هر گره از کلید فرزندانش کوچکتر باشد درخت دارای ویژگی هرم کمینه است. به طور خلاصه $A[Parent(i)] \geq A[i]$ نشان‌دهنده ویژگی هرم بیشینه و $A[Parent(i)] \leq A[i]$ برای یک هرم کمینه است.



(آ) هرم بیشینه (ب) هرم کمینه

۱.۱ نمایش هرم‌ها

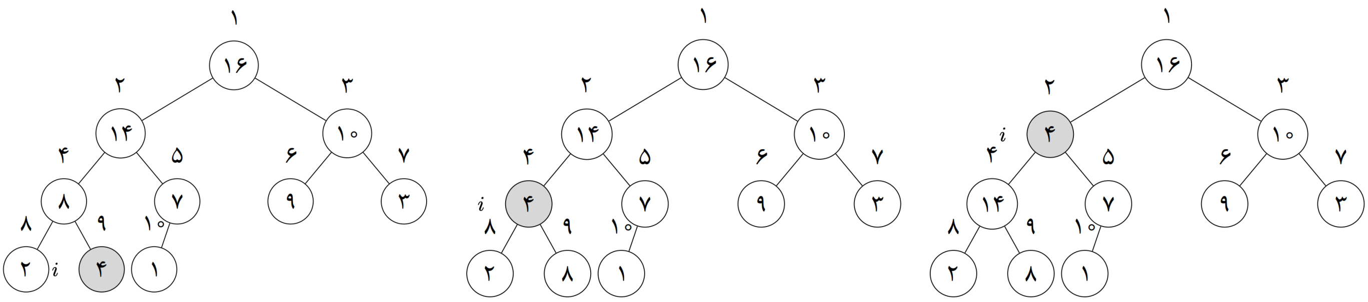
نمایش با استفاده از لیست پیوندی و آرایه دو شکل مشهور نمایش درخت دودویی در ساختمان داده‌ها است. در حالت عادی انتخاب یکی از این دو روش برای نمایش بهینه و با مصرف حافظه‌ی کمتر بسته به چیدمان عناصر درخت دارد. به عنوان مثال، در درخت‌های مورب روش نمایش با آرایه بدترین بازدهی و بیشترین مصرف حافظه را دارد. اما در درخت دودویی کامل این روش در مقایسه با روش لیست پیوندی بسیار بهینه‌تر است.



$A =$									
۳	۸	۱	۱۵	۵	۱۲	۹	۲	۴	۶

۱.۲ روند MAX-HEAPIFY

این روال برای حفظ ویژگی Max Heap به کار می‌رود. ورودی آن آرایه‌ی A و اندیس i در آرایه است.زمانی که این روال فراخوانی می‌شود فرض می‌شود که درخت‌های دودویی مشتق شده از left(i) و right(i) خود به تنهایی یک Max Heap هستند. ولی عنصر A[i] ممکن است کوچک‌تر از فرزندانش باشد. در نتیجه ویژگی Max Heap از بین می‌رود. وظیفه روال MAX-HEAPIFY این است که مقدار موجود در A[i] را به سمت پایین حرکت بدهد تا درخت مشتق شده از i یک Max Heap شود.



(ج) (ب) (آ)

```
def MAX_HEAPIFY(A,i):
        l=i.left
        r=i.right
        max=i
        if(l<=len(A) and A[l]>A[max]):
                max=l
        if(r<=len(A) and A[r]>A[max]):
                        max=r
        if(max!=i):
                swap(max,i)
                return MAX_HEAPIFY(A,max)
```

۱.۳ ساختن هرم

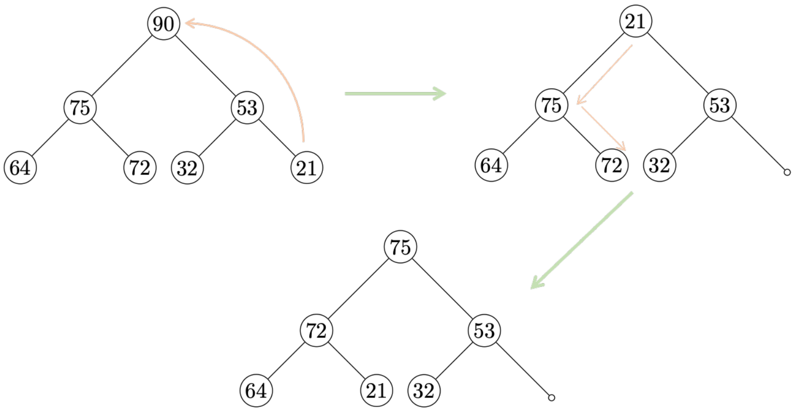
در اینجا می‌خواهیم با استفاده از روال MAX-HEAPIFY یک آرایه‌ی A[1...n] را به یک Max Heap تبدیل کنیم.

روال Build-Max-Heap در درخت حرکت کرده و با استفاده از MAX-HEAPIFY آن‌را به صورت درجا به Max Heap تبدیل می‌کند. این الگوریتم از $O(n)$ زمان مصرف می‌کند.

```
def Build_Max_Heap(A):
        heap.size=len(A)
        st=(heap.size)//2
        for i in [st, st-1, st-2, ..., 1]:
                MAX_HEAPIFY(i)
        return 'Built.'
```

۱.۳ حذف عنصر بیشینه

از آنجا که در Max Heap بزرگ‌ترین عنصر در ریشه قرار دارد. برای این کار ابتدا ریشه را از درخت خارج و سپس آخرین عنصر هرم(آرایه) را در ریشه قرار می‌دهد. با این کار فرزندان ریشه Max Heap باقی خواهند ماند اما عنصر ریشه جدید ممکن است ویژگی Max Heap را نداشته باشد. از این رو برای بازیابی هرم، یک MAX-HEAPIFY به روی عنصر ریشه کافیست.




```
def Extract_MAX(A):
        if(len(A)<1):
                return 'Heap is empty'
        result=A[1]
        A[1]=A[len(A)-1]
        len(A)=len(A)-1
        MAX_HEAPIFY(A,1)
        return result
```

۱.۴ درج در هرم

ابتدا گره‌ای جدید با کلید $-\infty$ به عنوان آخرین گره (در انتهای آرایه) در درخت درج می‌شود سپس با استفاده از روند افزایش کلید، مقدار گره به مقدار مورد نظر افزایش یافته و در مکان صحیح خود قرار می‌گیرد. با توجه به آن که گره درج شونده ممکن است تا ریشه جابه‌جا شود، زمان درج حداکثر $O(h)$ خواهد بود. از طرفی ارتفاع یک هرم حداکثر $h = \lfloor \log n \rfloor + 1$ است. بنابراین زمان درج یک گره در هرم $O(\log n)$ خواهد بود.

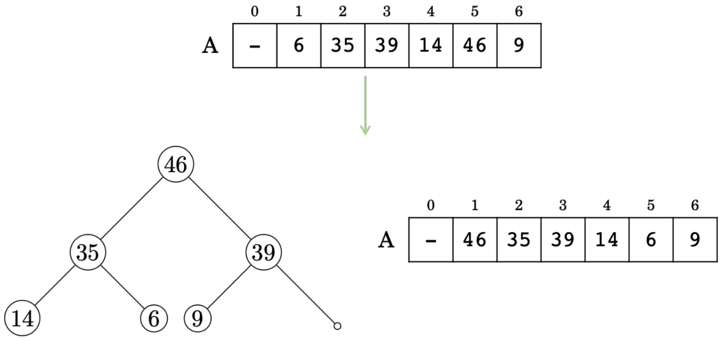
تابع افزایش کلید مسیر این گره تا ریشه را برای یافتن مکان مناسب برای این کلید می‌پیماید. در طی این مسیر، کلید افزایش یافته مکررا با پدرش مقایسه می‌شود. اگر این کلید از پدرش بزرگ‌تر باشد. با آن جابه‌جا می‌شود. در غیر این صورت روال به پایان می‌دهد.

```
def HEAP_INCREASE_KEY(A,i,key):
        A[i]=key
        par=i//2 # parent(i)
        while(i>0 and A[par]<A[i]):
                swap(A[i],A[par])
                i=i//2 # i = parent(i)
        return 'done.'

def insert(A,key):
        A.append(-∞)
        HEAP_INCREASE_KEY(A,len(A),key)
        return 'done.'
```

۲. مرتب‌سازی بر مبنای هرم

بنظرم این یک چیز واضح است، کافیست تا برا اساس لیست اولیه از اعداد یک هرم ایجاد شود، سپس استخراج عناصر از هرم را انجام داد؛ بر مبنای آنچه که گفتیم این استخراج به ترتیب کمینه یا بیشینه(وابسته به نوع هرم) انجام می‌شود و این یعنی با استخراج همه عناصر از هرم می‌توان لیست مرتب شده را تشکیل داد. از آنجا که هرم را از مرتبه $O(n)$ می‌توان ایجاد کرد و استخراج از آن به $O(\log n)$ عملیات نیاز دارد، به طور متوسط این مرتب سازی از زمان $O(n + \log n) = O(\log n)$ زمان می‌برد.



۳. سوالات برنامه نویسی

1. HackerEarth, Chandu and chandni's secret chat
2. HackerEarth, Divide Apples
3. HackerEarth, Seating Arrangement
4. HackerEarth, Special Array Operation
5. Timus, 1306. Sequence Median

۴. برای مطالعه بیشتر

1. Suchenek, Marek A. "Elementary yet precise worst-case analysis of Floyd's heap-construction program." Fundamenta Informaticae 120.1 (2012): 75-92.
2. Edelkamp, Stefan, Amr Elmasry, and Jyrki Katajainen. "Heap Construction—50 Years Later." The Computer Journal 60.5 (2017): 657-674.
3. Frederickson, Greg N. "An optimal algorithm for selection in a min-heap." Information and Computation 104.2 (1993): 197-214.
4. Brodal, Gerth Stølting. "Worst-case efficient priority queues." Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms. 1996.
5. Goodrich, Michael T.; Tamassia, Roberto (2004). "7.3.6. Bottom-Up Heap Construction". Data Structures and Algorithms in Java (3rd ed.). pp. 338-341.
6. Fredman, Michael L., and Robert Endre Tarjan. "Fibonacci heaps and their uses in improved network optimization algorithms." Journal of the ACM (JACM) 34.3 (1987): 596-615.
7. Takaoka, Tadao. "Theory of 2-3 heaps." Discrete Applied Mathematics 126.1 (2003): 115-128.