اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

1-1- درخت Heap

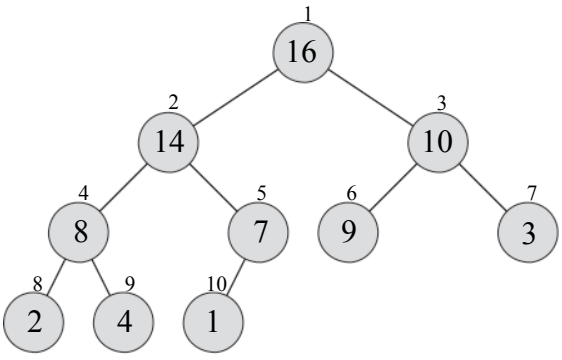
Heap نوعی درخت است که در آن فقط ارتباط میان پدران و فرزندان با هم اهمیت دارد و همزاد ها هیچگونه ارتباط خاصی با هم ندارند در ادامه با ویژگی های Heap آشنا میشویم.

ساختمان داده Heap (دوتایی) یک آرایه است (در پیاده سازی درخت Heap توسط آرایه هیچ خانه ای از آرایه خالی نمی ماند) که میتواند بصورت یک درخت دودویی تقریبا کامل دیده شود، همانطور که در شکل 2 نشان داده شده است.

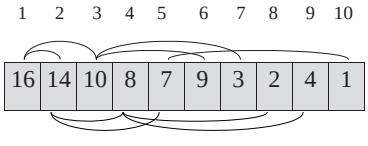
هرگره درخت به یک عنصر از آرایه ای که مقادیر گره ها را نگهداری می کند، اشاره میکند.یک آرایه مرتب همیشه میتواند یک آرایه Heap باشد اما برعکس آن لزوما برقرار نیست مانند شکل 2

درخت به طور کامل در همه سطح ها پر می شود (متوازن است) مگر در آخرین سطح که ممکن است به صورت کامل پر نشود و در آن از سمت چپ پر می شود در نتیجه درخت های Heap با تعداد مشخص گره دارای شکل یکسان هستند.

درخت Heapدودویی به دو نوع Max-Heap و Min-Heap تقسیم میشود. در max- / min-heap ارزش هرگره از فرزندانش کمتر / بیشتر و ریشه maximum / minimum است.



(ب)



(الف)

شکل 1- یکmax-heapکه بصورتآرایه ای که ریشه در اندیس 1 قرار دارد (شکل الف) و درخت دودویی با ارتفاع سه (شکل ب) نشان داده شده است. عددی که در هر دایره گره درخت است مقدار ذخیره شده در آن گره است و عددی که بالای هر دایره است نمایانگر اندیس مربوط به مقدار آن گره در آرایه است. خط های نشان داده شده بالا و پایین آرایه شکل (الف) رابطه پدر فرزندی را نشان می دهند. پدر ها همیشه سمت چپ فرزندانشان هستند.

رابطه پدر فرزندی در آرایه ای که بیانگر درخت heap است و مقادیر گره های درخت در آن ذخیره شده اند با در نظر گرفتن " i " به عنوان اندیس آرایه به صورت زیر است:

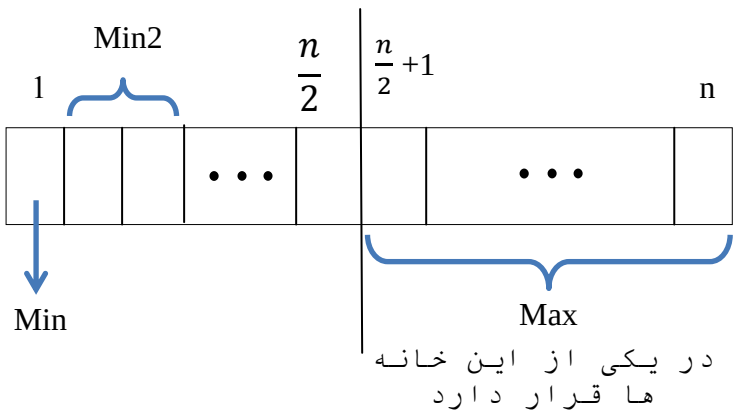
```
PARENT(i)
    return [ i/2 ]

LEFT(i)
    return 2i

RIGHT(i)
    return 2i+1
```

در درخت Heap ، ارتفاع هر گره برابر با تعداد یال ها در بلندترین مسیر رو به پایین از آن گره به برگ و ارتفاع درخت برابر با ارتفاع ریشه است.

از آنجا که Heap با n عضو یک درخت دودویی کامل است ارتفاع آن $\theta(\log n)$ است.

شکل Heap برای n عدد همیشه یکسان است.

نکته:پیاده سازی درخت Min-Heapتوسط آرایه (ریشه در اندیس 1 آرایه قرار دارد) :



اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

شکل 2- درخت Min-Heap ، پیدا کردن مینیمم ها و ماکزیمم درخت در آرایه. Min3 (سومین کوچکترین مقدار در درخت) در یکی از اندیس های 2 تا 7 وجود دارد، یعنی شش حالت دارد. اگر Min2 مشخص باشد برای Min3 سه حالت وجود دارد.

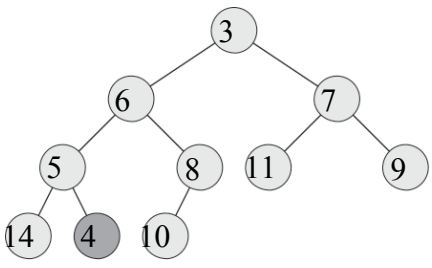
1-1-1- تعریف Heapify:

یعنی در درخت Min-Heap (Min-Heapify) / Max-Heap(Max-Heapify) کاری کنیم که ارزش هر گره از فرزندانش کمتر/ بیشتر باشد.(خواص Binary Heap حفظ شود و درخت همواره Heap باقی بماند) این عمل به دو نوع زیر تقسیم می شود:

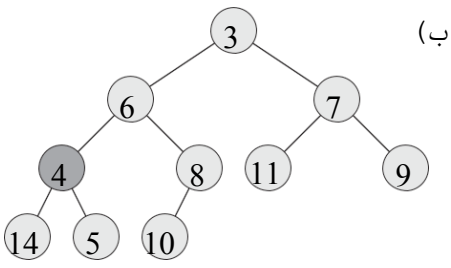
Bubble-Up: به منظور این کار در درخت Min-Heap / Max-Heap با شروع از برگ ها، هر گره را با پدرش مقایسه میکنیم اگر کوچکتر / بزرگتر بود آن ها را با هم Swap (جابجا)میکنیم. شکل 4

Bubble-Down: به منظور این کار درخت Min-Heap / Max-Heap با شروع از ریشه هر گره را با فرزندانش مقایسه میکنیم اگر از یکی از فرزندانش کوچکتر / بزرگتر بود آن ها را با هم Swap (جابجا) میکنیم و اگر از هر دو فرزندش کوچکتر / بزرگتر بود آن را با کوچکترین / بزرگترین فرزندش جابجا میکنیم.

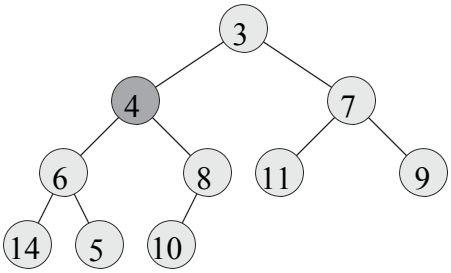
(الف)



(ب)



(ج)



شکل 3- BubbleUp←Min-Heapify: در قسمت (الف) مشاهده میکنید که گره با مقدار 4 از پدرش (گره با مقدار 5) کوچکتر است که این بر خلاف ویژگی های درخت Min-Heap (ارزش هر گره از فرزندانش کمتر است) می باشد. در نتیجه آن را با پدرش جابجا میکنیم و درخت قسمت (ب) حاصل میشود. حال در این شکل دوباره مشاهده میکنید که گره با مقدار 4 از پدرش کوچکتر است پس آن ها را جابجا کرده و شکل قسمت (ج) حاصل میشود. شکل قسمت (ج) یک درخت Min-Heap است.

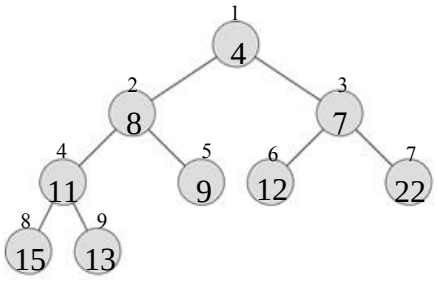
کد Heapify برای Max Heap به شکل زیر است:

همان طور که میبینید این کار به صورت بازگشتی ادامه پیدا میکند.

```
MAX-HEAPIFY(A, i, n)
    l ← LEFT(i )
    r ← RIGHT(i )
    if l ≤ n and A[l] > A[i ]
    then largest ←l
    else largest ←i
    if r ≤ n and A[r ] > A[largest]
    then largest ←r
    if largest _= i
    then exchange A[i ] ↔ A[largest]

     MAX-HEAPIFY(A, largest, n)
```

1-1-2- اضافه کردن یک گره به درخت Heap (Insert Heap):

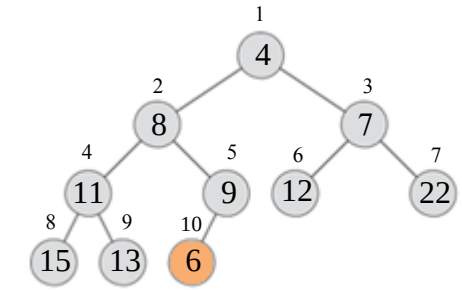
در اضافه کردن یک گره به درخت دودویی Heap همیشه این خاصیت درخت Heap که همیشه برگ ها از چپ به راست پر میشوند باید حفظ شود. پس درنتیجه برای این کار همیشه تنها یک جا برای اضافه کردن گره وجود دارد. پس از اضافه کردن گره به درخت آن را Heapify میکنیم تا درخت کماکان درخت Heap باقی بماند. هزینه آن به علت انجام عمل Heapify حداکثر به اندازه ارتفاع درخت، برابر با ارتفاع آن یعنی $O(\log n)$ است. به مثال زیر توجه کنید:



اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

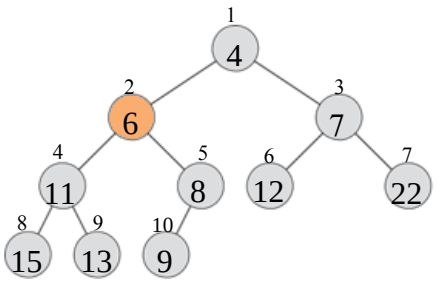
1	2	3	4	5	6	7	8	9
4	8	7	11	9	12	22	15	13

شکل 4- دردرخت Min-Heapبالا میخواهیم یک گره با مقدار 6 را به آن اضافه کنیم. با توجه به Heap بودن این درخت چون باید برگ ها از چپ به راست پر شوند تنها جای ممکن برای این گره فرزند چپ گره با ارزش 9 (اندیس 5) است. پس آن را به درخت اضافه میکنیم. همچنین در آرایه متناظر با درخت همواره گره اضافه شده به درخت به آخر آرایه اضافه میشود. یعنی در اینجا گره اضافه شده در اندیس 10 آرایه قرار خواهد گرفت.

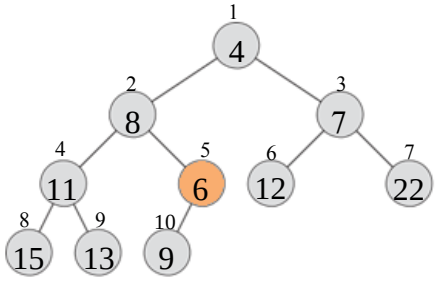


1	2	3	4	5	6	7	8	9	10
4	8	7	11	9	12	22	15	13	6

شکل 5- حال در این درخت که گره با مقدار 6 اضافه شده است (در درخت به عنوان آخرین برگ و در آرایه بعد از آخرین خانه آرایه در اندیس 10) . مشاهده می شود که گره اضافه شده از پدرش کوچکتر است که این بر خلاف Min-Heap بودن درخت است. در نتیجه لازم است که با Heapify کردن گره اضافه شده درخت را دوباره به Min-Heap تبدیل کنیم.



1	2	3	4	5	6	7	8	9	10
4	6	7	11	8	12	22	15	13	9



1	2	3	4	5	6	7	8	9	10
4	8	7	11	6	12	22	15	13	9

(الف) (ب)

شکل 6- در شکل قسمت (ب) مشاهده می شود که پس از انجام Heapify(Bubble Up) روی درخت شکل 6 و سپس روی درخت قسمت (الف)، درخت اولیه با اضافه شدن گره جدید (با مقدار 6) باز هم یک Min-Heap باقی مانده است.

1-1-3- حذف کردن یک گره از درخت Heap (Delete Heap):

برای حذف کردن گره x از درخت Heap با توجه به مکان قرار گیری گره دو حالت زیر وجود دارد:

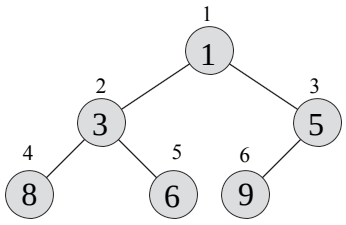
1) گره x ریشه باشد (Delete Min در Min-Heap یا Delete Max در Max-Heap) : در این حالت مثلا در درخت Min-Heap آخرین برگ را حذف کرده و مقدار آن را بجای مقدار ریشه میگذاریم، سپس تا زمانی که همه گره ها از هر دو فرزندشان کوچکتر شوند با شروع از ریشه هر گره را با کوچکترین فرزندش جابجا میکنیم. (Heapify -> Bubble Down) شکل 8

2) گره x ریشه نباشد : در این حالت هم دوباره آخرین برگ را حذف کرده و مقدار آن را به جای عنصر x قرار می دهیمو بعد Heapify میکنیم.

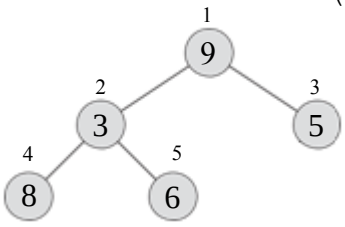
نکته: در حالت 2 معلوم نیست که باید Bubble Up انجام دهیم یا Bubble Down. این انتخاب بستگی به داده های درخت دارد. مثلا در حذف یک گره میانی در درخت Min-Heap اگر عنصر x که مقدار آن با آخرین برگ جابجا شده از پدرش کوچکتر بود Bubble Up و اگر از فرزندش بزرگتر بود Bubble Down انجام میدهیم تا درخت Min-Heap باقی بماند.

نکته:هزینه حذف یک گره از درخت دودویی Heapبه علت انجام عمل Heapify حداکثر به اندازه ارتفاع درخت، برابر با ارتفاع آن یعنی $O(\log n)$ است.

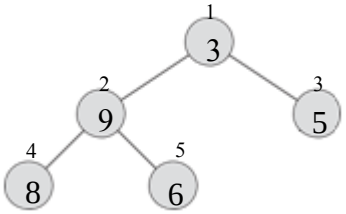
(الف)



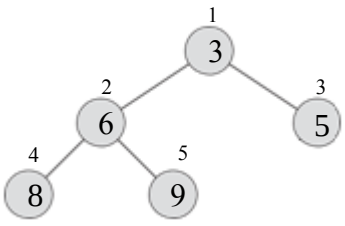
(ب)



(ج)



(د)



شکل 7- (الف) در این درخت Min-Heap میخواهیم ریشه (گره با ارزش 1) را حذف کنیم. (ب) حال آخرین برگ درخت یعنی گره با ارزش 9 را حذف کرده و مقدار آن را با مقدار ریشه جابجا میکنیم. در اینجا میبینیم که ریشه از فرزندانش بزرگتر است (ج) پس آن را با کوچکترین فرزندش یعنی گره با ارزش 3 جابجا میکنیم. (د) در این شکل گره با ارزش 9 را با کوچکترین فرزندش یعنی گره با ارزش 6 را جابجا میکنیم و درخت نهایی باز هم یک Min-Heap است.

Increase/Decrease Key

در Heap ها دو عمل رایج دیگر با نام های Decrease / Increase Key وجود دارد که عنصری از درخت حذف یا به آن اضافه نمی کند بلکه یک مقدار که در درخت وجود دارد را کم / زیاد میکند که بعد از انجام این کار باید یک دور Heapify هم انجام دهیم تا مطمئن شویم درخت ما همچنان Heap است در زیر نمونه کدی که در آن مقدار خانه I ام آرایه را به مقدار key افزایش میدهد آورده شده برای کاهش هم به همین ترتیب عمل میکنیم :

```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i ]
        then error .new key is smaller than current key.
A[i ] ← key
while i > 1 and A[PARENT(i )] < A[i ]
do exchange A[i ] ↔ A[PARENT(i )]

i ← PARENT(i )
```

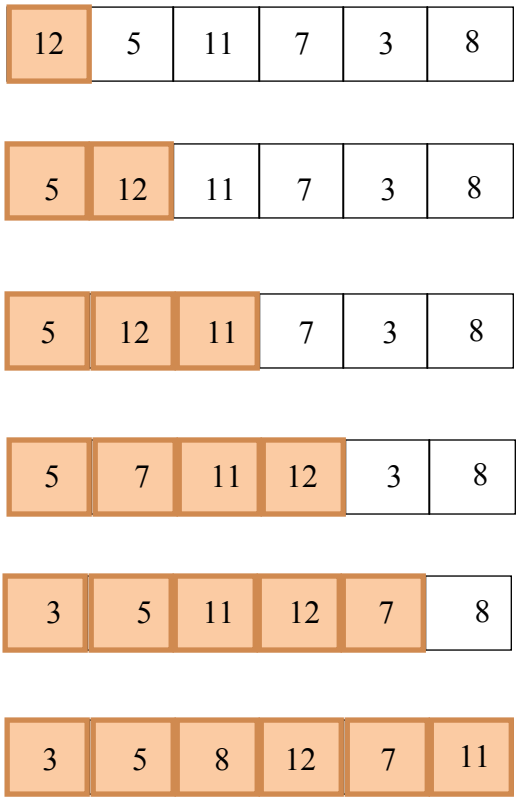
اگر ندانیم عنصری که میخواهیم مقدار آن را تغییر دهیم در چه خانه ای قرار دارد باید کل خانه های heap را چک کنیم به عبارت دیگر پیچیدگی زمانی جستجو روی درخت heap ، O(n) است.

1-1-4- ساختن درخت دودویی Heap (Make Heap):

برای ساختن درخت Heap از روی آرایه ای به طول n بدون حافظه کمکی ($O(1)$ حافظه) سه روش وجود دارد:

1) میتوان با هزینه $O(n \log n)$ آرایه را مرتب کرد. آرایه مرتب یک Heap است.

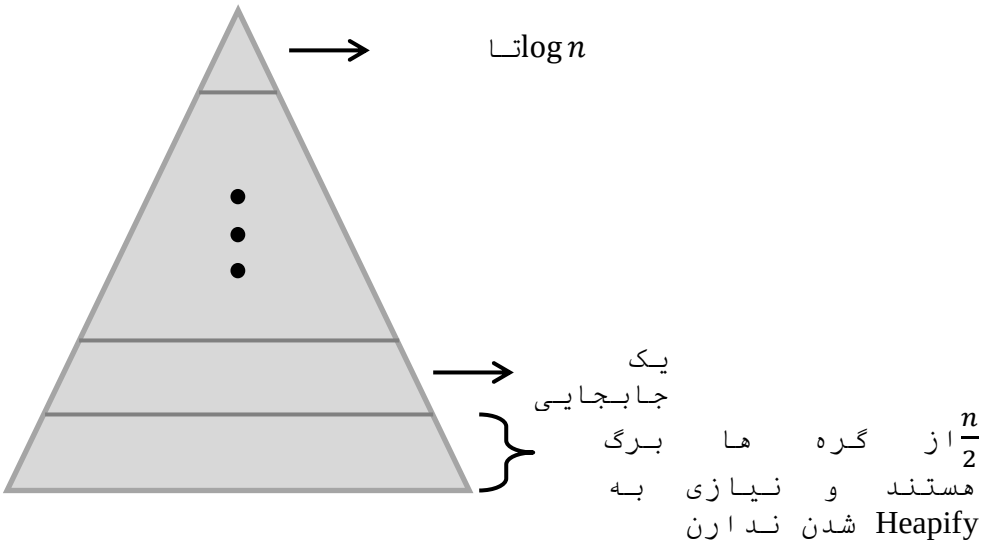
2) با n بار Insert Heap میتوانبا هزینه $O(n \log n)$ یک Heap ساخت. مانند شکل 9.



شکل 8- ساختن درخت Min-Heap. در اینجا برای ساخت درخت از حافظه کمکی استفاده نکرده ایم بلکه در هر مرحله از خود آرایه استفاده میکنیم و فقط چند سری خانه های آرایه را باهم جابجا (swap) کرده ایم. در واقع بخش های رنگی هر آرایه نمایانگر درخت Min-Heap ایجاد شده پس از هر بار اضافه کردن یک گره است. وقتی گره ای را اضافه میکنیم باید آن را Heapify کنیم. تا درخت کماکان Min-Heap باقی بماند. مثلا وقتی گره با ارزش 3 را به درخت اضافه میکنیم مشاهده میشود که از پدرش یعنی گره با ارزش 7 کوچکتر است که این بر خلاف Min-Heap بودن است پس آن را با پدرش جابجا میکنیم یعنی در آرایه خانه با مقدار 3 را با خانه با مقدار 7 که پدرش در درخت است جابجا (swap) میکنیم. و سپس به دلیل ذکر شده خانه با مقدار 3 را با خانه با مقدار 5 جابجا کرده و خواهیم دید که درخت با ریشه 3 یک Min-Heap می باشد.

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

3) با هزینه $O(n)$: از آخر به اول به جز $\frac{n}{2}$گره آخر را Heapify میکنیم.



$$1 \times \log n + 2 \times (\log n - 1) + \cdots + \frac{n}{4} \times 1 = O(n)$$

شکل 9- تعداد جابجایی هر سطح که از بالا به پایین کم میشود در شکل نشان داده شده است. پس برای محاسبه هزینه کل آن داریم : هزینه هر سطح (تعداد گره های آن سطح ضرب در تعداد جابجایی ها) را حساب کرده و سپس با هم جمع میکنیم. هزینه این کار $O(n)$ می باشد.

مثال : فرض کنید دو Max Heapبا نام های A,B داریم و میخواهیم این دو را در قالب یک Heap پیاده سازی کنیم بهترین روش چیست ؟ (Merge Heap)

مسئله را به دو حالت تقسیم میکنیم :

A,B برابر باشند : در این صورت سمت راست ترین برگ B را به عنوان ریشه قرار میدهیم و A را به عنوان فرزند چپ آن و B را به عنوان فرزند سمت راست قرار میدهیم و روی ریشه Heapify را انجام میدهیم .

A,B برابر نباشند : فرض میکنیم A بزرگتر از B باشد در این صورت به اندازه B از پایین سمت چپ A جدا میکنیم و آن را C مینامیم . چون اندازه C و B برابر است طبق روش گفته شده میتوانیم آن هارا Mergeکنیم و درخت حاصل را D مینامیم و آن را در جای C در درخت A قرار میدهیم و ریشه درخت D را E مینامیم باید از E رو به بالا Heapifyکنیم و هرکدام از رئوسی که با E جابجا میشوند را تا پایین heapify کرد چون ممکن است از راس های درخت Bکوچک تر باشند.

پیچیدگی زمانی : O(log(n))

1-1-5- کاربرد ها :

1) مرتب سازی Heap Sort: با هزینه $O(n \log n)$

1. ابتدا Make-Heap را انجام میدهیم که هزینه کل آن $O(n)$ میشود.

2. سپس n بار Delete-Heap(Min/Max) را روی درخت اجرا میکنیم. خروجی ها مرتب هستند و هزینه آن $O(n \log n)$ میشود.

2) صف اولویت:

1. Max-Heap: $O(\log n)$

* بهترین روش از نظر هزینه

خروج از صف: Delete-Max با هزینه $O(\log n)$

ورود به صف: Insert-Heap با هزینه $O(\log n)$

2. آرایه مرتب:O(n)

خروج از صف: O(1)

ورود به صف: O(n)

3. آرایه نامرتب:O(n)

خروج از صف: θ(n)

اخطار : محتویات فایل‌ها تایید شده نیستند و مفاهیم و روابط ممکن است اشتباه باشند

ورود به صف: O(1)

4. لیست پیوندی مرتب:O(n)

خروج از صف: O(1)

ورود به صف: O(n)

5. لیست پیوندی نامرتب:O(n)

خروج از صف: O(n)

ورود به صف: O(1)

فیبوناچی heap

یک نوع از درخت Heap ، فیبوناچی نام دارد. به این نوع از ساختمان داده ها را تنبل (lazy) می نامند که نسبت به درخت Heap معمولی دارای پیچیدگی زمانی بهتری میباشد . پیچیدگی زمانی هریک از دستور های این ساختمان داده به صورت زیر میباشد.

Delete min/max: O(log(n)) Insert : O(1)

همان طور که مشاهده میکنید هزینه اضافه کردن به این نوع Heap بسیار کم است در اصل این نوع Heap در موقع اضافه کردن heapify نمیکند شیوه ی پیاده سازی این نوع Heapجزو مطالب درسی نمی باشد ولی برای اطلاع از شیوه ی پیاده سازی این درخت به آدرس زیر مراجعه کنید :

http://en.wikipedia.org/wiki/Fibonacci_heap