دانشکده‌ی علوم ریاضی

ساختمان داده — ۵ آبان ۹۲

جلسه‌ی ۱۱: درخت دودویی، هرم

مدرّس: دکتر شهرام خزائی — *نگارنده: احمدرضا رحیمی*

۱ مقدمه

الگوریتم مرتب‌سازی هرمی[۱] یکی دیگر از الگوریتم‌های مرتب‌سازی است که دارای برخی از بهترین ویژگی‌های دو الگوریتم مرتب‌سازی درجی[۲] و مرتب‌سازی ادغامی[۳] که قبلاً آنها را مورد بررسی قرار دادیم است. برخی از ویژگی‌های خوب الگوریتم مرتب‌سازی هرمی در زیر آورده شده:

- زمان اجرای $O(n \lg n)$ مثل مرتب‌سازی ادغامی
- مرتب‌سازی درجا مثل مرتب‌سازی درجی

الگوریتم مرتب‌سازی هرمی را در جلسه بعد مطرح خواهیم کرد. برای معرفی این الگوریتم لازم است مقدماتی را در باره‌ی داده ساختار هرم[۴]، انواع درخت‌ها و درخت‌های دودویی[۵] بدانیم، که در این جلسه به آن می‌پردازیم.

۲ معرفی درخت‌ها

تعریف ۱ (درخت) *درخت یک گراف همبند است که دور ندارد.*

تعریف ۲ (درخت دودویی) *درخت دودویی به طور بازگشتی تعریف می‌شود: یک درخت دودویی مجموعه‌ای از گره‌هاست که ممکن است تهی باشد. اگر تهی نباشد، شامل یک گره به‌نام ریشه و دو درخت دودویی مجزا که زیردرخت‌های چپ و راست نامیده می‌شوند، است.*

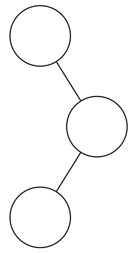
تعریف ۳ (اجزای درخت دودویی) *فرض کنید T یک درخت دودویی غیر تهی با ریشه r و زیردرخت‌های چپ و راست L و R باشد.*

- *(**فرزند چپ**) اگر L تهی نباشد، ریشه آن فرزند چپ r است.*
- *(**فرزند راست**) اگر R تهی نباشد، ریشه آن فرزند راست r است.*
- *(**پدر و فرزند**) هریک از فرزندان چپ و راست r، فرزند آن است. همچنین r پدر فرزندان خود است.*

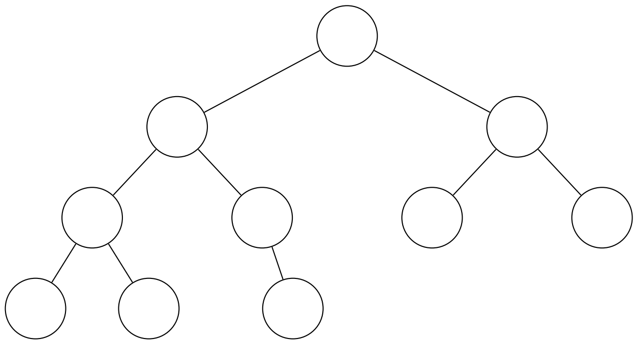
---
[۱] HeapSort
[۲] InsertionSort
[۳] MergeSort
[۴] Heap
[۵] Binary Trees

- **(همزاد)** *فرزندان چپ و راست یک گره همزاد یکدیگرند.*
- **(برگ)** *گره‌ای که هیچ فرزندی نداشته باشد، برگ نامیده می‌شود.*
- **(گره میانی)** *گره‌ای که ریشه یا برگ نباشد، گره میانی نامیده می‌شود.*

می‌توان یک درخت دودویی را با یک نمودار با رسم یالی بین پدران و فرزندان و قرار دادن گره‌ها در سطوح مختلف نشان داد. شکل ۱ درخت‌های دودویی را نشان می‌دهد. دقت کنید تنها گره‌ای که پدر ندارد ریشه درخت است.



(آ) یک درخت دودویی با ۳ گره



(ب) یک درخت دودویی با ۱۲ گره

شکل ۱: درخت‌های دودویی

تعریف ۴ (ارتفاع درخت) *طول طولانی‌ترین مسیر از ریشه به یک برگ را ارتفاع درخت می‌گوییم.*

تعریف ۵ (ارتفاع گره) *طول طولانی‌ترین مسیر از یک گره به یک برگ را ارتفاع آن گره می‌گوییم.*

تعریف ۶ *(سطح گره) به هر یک از گره‌های درخت با ارتفاع h یک شماره سطح در بازه $[0, h-1]$ نسبت می‌دهیم. ریشه در سطح 0 قرار دارد؛ اگر پدر گرهی در سطح d قرار گرفته باشد، خود گره در سطح $d+1$ جای دارد.*

تعریف ۷ (درخت دودویی کامل) *نوعی درخت دودویی است که در آن هر گره دقیقا دو فرزند دارد.*

لم ۱ *برای هر درخت دودویی کامل با n گره و ارتفاع h داریم $n = 2^{h+1} - 1$ یا معادلاً $h = \lg(n+1) - 1$.*

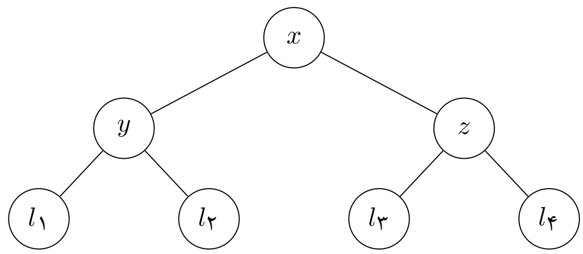
مثال ۱ *شکل ۲ یک درخت دودویی کامل به ارتفاع $h = 2$ که x ریشه آن است و y و z به ترتیب فرزندان چپ و راست x هستند و l_i ها برگ‌های درخت هستند. ریشه در سطح 0، گره‌های y و z در سطح ۱ و برگ‌ها همگی در سطح ۲ قرار گرفته‌اند.*

تعریف ۸ (درخت دودویی تقریباً کامل) *یک درخت دودویی تقریباً کامل با ارتفاع h نوعی درخت دودویی با ارتفاع h است که دو شرط زیر در آن صدق می‌کنند:*

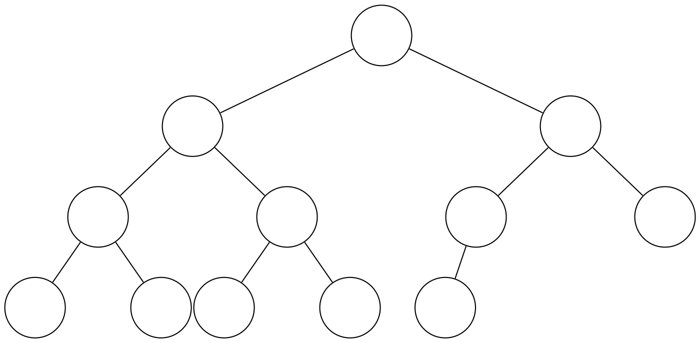
- *در سطح dام درخت، $d = 0, 1, \cdots h-1$، دقیقاً 2^d گره وجود دارند.*
- *اگر در سطح hام، برگی موجود باشد، تمام برگ‌های سمت چپ آن نیز حضور دارند.*

لم ۲ *برای هر درخت دودویی تقریباً کامل با n گره و ارتفاع h داریم $2^h < n \le 2^{h+1} - 1$ یا معادلاً $h = \lfloor \lg n \rfloor$.*

مثال ۲ *شکل ۳ یک درخت دودویی تقریباً کامل با $n = 12$ گره و به ارتفاع $h = 3$ را نشان می‌دهد. دقت کنید که درخت شکل ۱ب با این درخت متفاوت است و درخت تقریباً کامل محسوب نمی‌شود.*



شکل ۲: درخت دودویی کامل به ارتفاع ۲



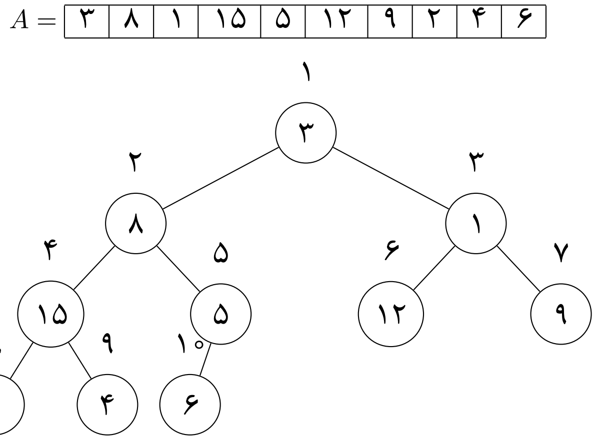
شکل ۳: یک درخت دودویی تقریباً کامل با ۱۲ گره

۱.۲ نمایش آرایه در درخت دودویی تقریباً کامل

می‌توان اعضای یک آرایه را به‌جای نمایش برداری-سطری در یک درخت دودویی تقریباً کامل تصور کرد. فرض می‌کنیم که اندیس عناصر آرایه از ۱ شروع می‌شود. برای این کار یک درخت دودویی تقریباً کامل که تعداد گره‌های آن برابر تعداد اعضای آرایه است در نظر می‌گیریم. سپس گره‌های درخت را با شروع از سطح صفر و از سمت چپ به راست با شروع از ۱ شماره‌گذاری می‌کنیم. بدین ترتیب به ریشه اندیس ۱، فرزند چپ آن اندیس ۲ و به همین ترتیب ادامه می‌دهیم. سپس عضو iام دنباله را در داخل گره با اندیس i در درخت قرار می‌دهیم که کلید آن گره خوانده می‌شود. چنین درختی هرم نامیده می‌شود.

مثال ۳ *شکل ۴ یک آرایه و درخت باینری (هرم) متناظر با آن را نشان می‌دهد.*

بنابراین می‌توانیم به آرایه $A = [۱ \cdots n]$ به عنوان یک درخت دودویی نگاه کنیم که هر عضو آن متناظر با یک گره از درخت است. در این درخت برای دستیابی به گره پدر و یا فرزندان یک گره، توابع $\textsc{Parent}(i)$، $\textsc{RightChild}(i)$ و $\textsc{LeftChild}(i)$ که بر روی گره i اجرا می‌شوند را به این صورت تعریف می‌کنیم:



شکل ۴: یک آرایه و هرم متناظر با آن

الگوریتم ۱ توابع $\textsc{Parent}(i)$، $\textsc{RightChild}(i)$ و $\textsc{LeftChild}(i)$

---

function $\textsc{Parent}(i)$
    return $\left\lfloor \frac{i}{2} \right\rfloor$
function $\textsc{LeftChild}(i)$
    return $2i$
function $\textsc{RightChild}(i)$
    return $2i + 1$

---

۳ ساختار هرم

در علوم کامپیوتر، 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[۶] می‌گویند. بسته به رابطه بزرگ‌تری و یا کوچک‌تری، هرم بیشینه یا کمینه خوانده می‌شود.

تعریف ۹ *(هرم بیشینه[۷] و هرم کمینه[۸]) اگر در یک درخت کلید هر گره از کلید فرزندانش 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 در هرم بیشینه* $A[\textsc{Parent}(i)] \geq A[i]$ *و در هرم کمینه* $A[\textsc{Parent}(i)] \leq A[i]$ *است.*

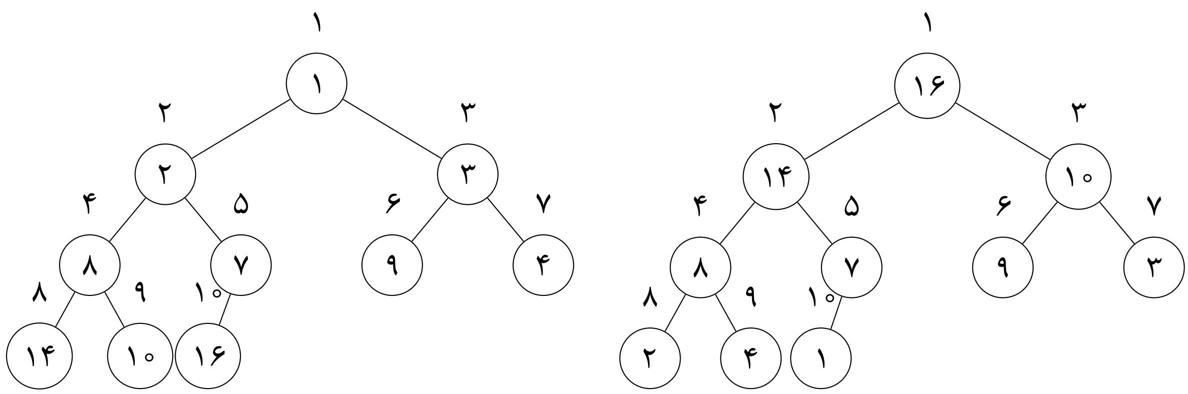
مثال ۴ *شکل ۵ یک هرم بیشینه و یک هرم کمینه را نشان می‌دهد.*

الگوریتم مرتب‌سازی هرمی نیاز به الگوریتمی دارد که ابتدا با استفاده از آن بتوان یک هرم را به یک هرم بیشینه تبدیل کرد. پس ابتدا لازم است با الگوریتم‌های زیر آشنا شویم:

- $\textsc{MaxHeapify}(\text{array } A, \text{ index } i)$: در زمان $O(\lg n)$ اجرا می‌شود و با فرض این که زیر درخت‌های چپ و راست گره i هرم بیشینه‌اند با حفظ ویژگی هرم بیشینه کلید گره i را به مکان مناسب می‌برد و زیر درخت با ریشه i را به یک هرم بیشینه تبدیل می‌کند.

---

[۶] heap property
[۷] MaxHeap
[۸] MinHeap



(آ) هرم بیشینه

(ب) هرم کمینه

شکل ۵: هرم‌های کمینه و بیشینه

- BuildMaxHeap(array A): در زمان $O(n)$ اجرا می‌شود و از آرایه ورودی یک هرم بیشینه می‌سازد.

۱.۳ الگوریتم MaxHeapify

الگوریتم MaxHeapify یک آرایه A و اندیس i را به عنوان ورودی می‌گیرد و فرض می‌کنیم زیر درخت‌های چپ و راست گره i خود هرم بیشینه باشند. اگر $A[i] \leq A[\text{LeftChild}(i)]$ یا $A[i] \leq A[\text{RightChild}(i)]$ باشد به $A[i]$ اجازه می‌دهد که در هرم بیشینه به سمت پایین حرکت کند تا ویژگی هرم بیشینه حفظ شود.

Algorithm 2 Algorithm: Max-Heapify

```
function MaxHeapify(array A, i)
    l ← LeftChild(i)
    r ← RightChild(i)
    if l ≤ n and A[l] > A[i] then
        largest ← l
    else
        largest ← i
    if r ≤ n and A[r] > A[i] then
        largest ← r
    if largest ≠ i then
        Exchange A[i] with A[largest]
        MaxHeapify(A,largest)
```

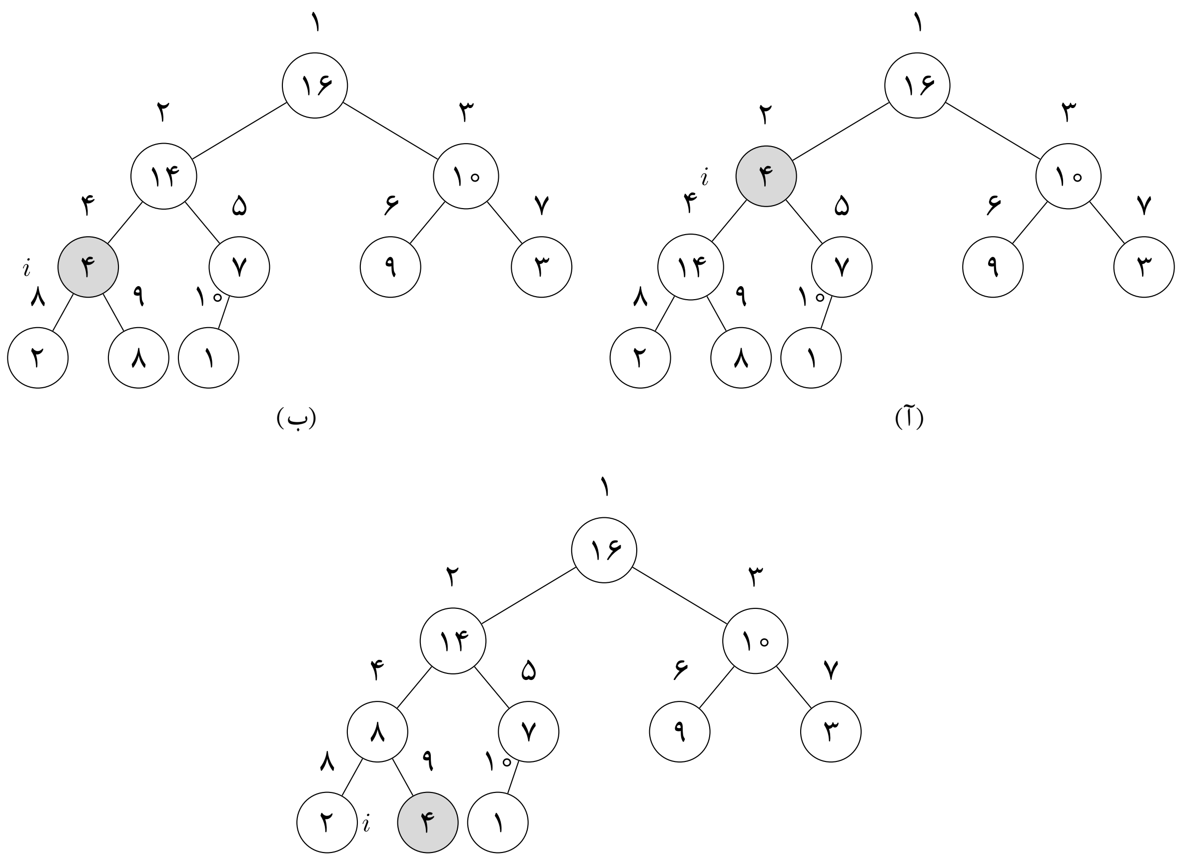
در هر مرحله الگوریتم اندیسی متناظر با خانه‌ی حاوی بیشینه سه عدد $A[i]$, $A[\text{LeftChild}(i)]$, $A[\text{RightChild}(i)]$ را پیدا و آن‌را در $largest$ ذخیره می‌کند. اگر $A[i]$ بزرگترین عدد باشد، یعنی $largest = i$، زیر درخت با ریشه i خود یک هرم بیشینه است و نیاز به تغییر ندارد. اما اگر $A[i]$ بزرگترین نباشد، جای آن با $A[largest]$ عوض می‌شود. دقت کنید که ویژگی هرم بیشینه ممکن است برای زیر درخت جدید با ریشه $largest$ از بین برود. بنابراین MaxHeapify را بر روی زیر درخت با ریشه $largest$ به صورت بازگشتی اجرا می‌کنیم. زمان یک‌بار اجرای MaxHeapify بر روی درخت با اندازه n با ریشه در گره i برابر $\Theta(۱)$ است. علاوه بر این باید زمان اجرای روال بازگشتی MaxHeapify

بر روی زیر درخت‌هایی که فرزندان i ریشه‌های آن هستند (همه فراخوانی‌های MaxHeapify) را حساب کنیم، چون زیر درخت های فرزندان i حداکثر $\frac{2n}{3}$ گره دارند-بدترین حالت زمانی رخ میدهد که پایین‌ترین سطح درخت نیمه پر باشد- زمان اجرای MaxHeapify بر روی درخت از رابطه بازگشتی زیر بدست می آید.

$$T(n) \leq T(\frac{2n}{3}) + \Theta(1)$$

که با توجه به رابطه‌ی بازگشتی بالا زمان اجرای این الگوریتم $O(\lg n)$ است.

مثال ۵ *شکل ۶ صفحه بعد اجرای الگوریتم* MaxHeapify *را روی زیردرخت با ریشه ۲ در درخت شکل الف نشان می‌دهد. با توجه به اینکه کلید گره ۲ از کلید فرزندانش (گره‌های ۴ و ۵) کوچک‌تر است، اندیس ۴ که فرزند حاوی کلید بزرگ‌تر است در* $largest$ *قرار می‌گیرد و جای کلید گره‌های ۲ و ۴ با هم عوض می‌شوند. فراخوان بعدی، روی زیردرخت با ریشه ۴ درخت شکل ب اجرا می‌شود و جای کلید گره‌های ۴ و ۹ عوض می‌شود و به درخت شکل ج تبدیل می‌شود.*



شکل ۶: اجرای MaxHeapify

۲.۳ ساختن هرم

با استفاده از MaxHeapify می‌توانیم آرایه $A[۱ \cdots n]$ را به یک هرم بیشینه تبدیل کنیم. الگوریتم BuildMaxHeap از گره‌های داخلی (غیر برگ) درخت می‌گذرد و برروی هر گره MaxHeapify را اجرا می‌کند. دقت کنید که عنصرهای زیر آرایه $A[\lfloor \frac{n}{۲} \rfloor + ۱ \cdots n]$ تماماً برگ‌های درخت هستند که خود هرم بیشینه هستند. بنابراین نیاز به اجرای الگوریتم MaxHeapify روی آن‌ها نیست.

Algorithm 3 Algorithm: BuildMaxHeap

```
function BuildMaxHeap(array A[1···n])
    for i = ⌊n/2⌋ downto 1 do
        MaxHeapify(A, i)
```

مثال ۶ *اجرای الگوریتم* BuildMaxHeapify *بر روی آرایه زیر در شکل ۷ قابل مشاهده است.*

۷	۸	۱۴	۱۰	۹	۱۶	۲	۳	۱	۴

۳.۳ اثبات درستی BuildMaxHeap

قضیه ۳ BuildMaxHeap *یک هرم بیشینه را تولید می‌کند.*

برهان. برای اثبات درستی الگوریتم ابتدا یک ناوردای حلقه مناسب به صورت زیر در نظر می‌گیریم:

- **ناوردای حلقه:** درست پس از مقداردهی متغیر حلقه‌ی **for** با مقدار i، همه گره‌های $i+۱, i+۲ \cdots n$ ریشه‌های یک هرم بیشینه هستند.

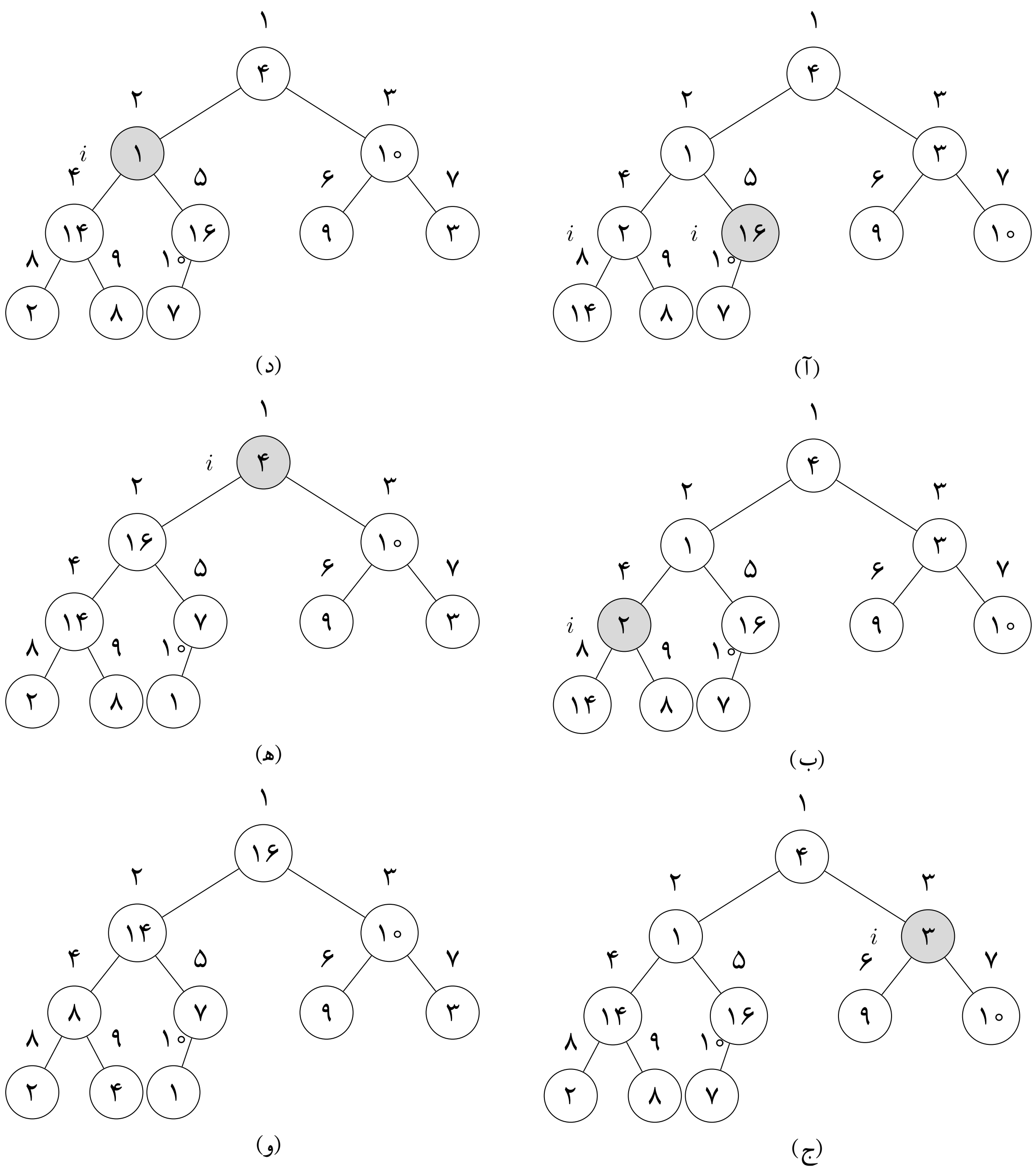
حال صحت مراحل آغاز، نگهداری و پایان را نشان می‌دهیم:

- **آغاز:** قبل از اولین تکرار **for**، هر کدام از گره‌های $\lfloor \frac{n}{۲} \rfloor + ۱ \cdots n$ یک برگ است و بنابراین ریشه یک هرم بیشینه‌ی بدیهی است.

- **نگهداری:** فرض کنید درست پس از مقداردهی متغیر حلقه‌ی **for** با مقدار i ناوردایی حلقه برقرار باشد. یعنی همه‌ی گره‌های $i+۱, i+۲, \cdots, n$ ریشه‌های یک هرم بیشینه باشند. در تکرار بعدی مقدار متغیر حلقه به $i-۱$ کاهش می‌یابد. باید نشان دهیم با اجرای MaxHeapify(A, i) همه‌ی گره‌های $i, i+۱, i+۲, \cdots, n$ ریشه‌های یک هرم بیشینه هستند. توجه کنید که فرزندان گره i دارای اندیسی هستند که از i بزرگترند و طبق فرض ناوردایی، خود ریشه‌های هرم‌هایی بیشینه هستند. بنابراین شرط لازم برای اجرای صحیح MaxHeapify(A, i) برقرار است و با اجرای آن i نیز ریشه‌ی یک هرم بیشینه می‌شود. بقیه گره‌های $i+۱, i+۲, \cdots, n$ که قبلاً ریشه‌های یک هرم بیشینه بوده‌اند، باز هم دارای این ویژگی خواهند بود (چرا؟).

  پایان: در پایان متغیر حلقه‌ی برابر $i = ۰$ است و با توجه به ناوردای حلقه همه گره‌های $۱, \cdots, n$ ریشه یک هرم بیشینه است. بالاخص، خود درخت نیز یک هرم بیشینه است.

■



شکل ۷: الگوریتم BuildMaxHeap از گره ۵ کار خود را آغاز می‌کند و $\textsc{MaxHeapify}(A,۵)$ را اجرا می‌کند چون زیر درخت‌های این گره هرم بیشینه هستند درخت بدون تغییر باقی می‌ماند و متغیر حلقه‌ی **for** به گره ۴ می‌رود و $\textsc{MaxHeapify}(A,۴)$ اجرا می‌شود و جای گره ۴ با ۵ عوض می‌شود و زیر درخت‌های گره ۴ هرم بیشینه می‌شوند. به همین ترتیب وقتی متغیر حلقه‌ی **for** به ۱ می‌رسد $\textsc{MaxHeapify}(A,۱)$ اجرا می‌شود و یک هرم بیشینه درست می‌شود.

۴.۳ زمان اجرای BuildMaxHeap

زمان اجرای MaxHeapify برای یک گره با ارتفاع h برابر $O(h)$ است. بنابراین می‌توانیم یک کران بالای ساده برای زمان کل BuildMaxHeap را با توجه به اینکه زمان اجرای MaxHeapify برابر $O(\lg n)$ است و BuildMaxHeap را $\frac{n}{2}$ بار اجرا می‌کند، پیدا کنیم. بنابراین زمان اجرای BuildMaxHeap برابر $O(n \lg n)$ است؛ اما این کران یک کران خوب نیست برای محاسبه یک کران بالای بهتر می‌توانیم به صورت زیر عمل کنیم:

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right)$$

می‌دانیم:

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = 2$$

بنابراین زمان اجرای الگوریتم BuildMaxHeap برابر است با:

$$O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) = O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) = O\left(n\right)$$

به وضوح زمان اجرای الگوریتم روی هر آرایه دلخواه از $\Omega(n)$ است. بنابراین زمان اجرای الگوریتم $\Theta(n)$ است.