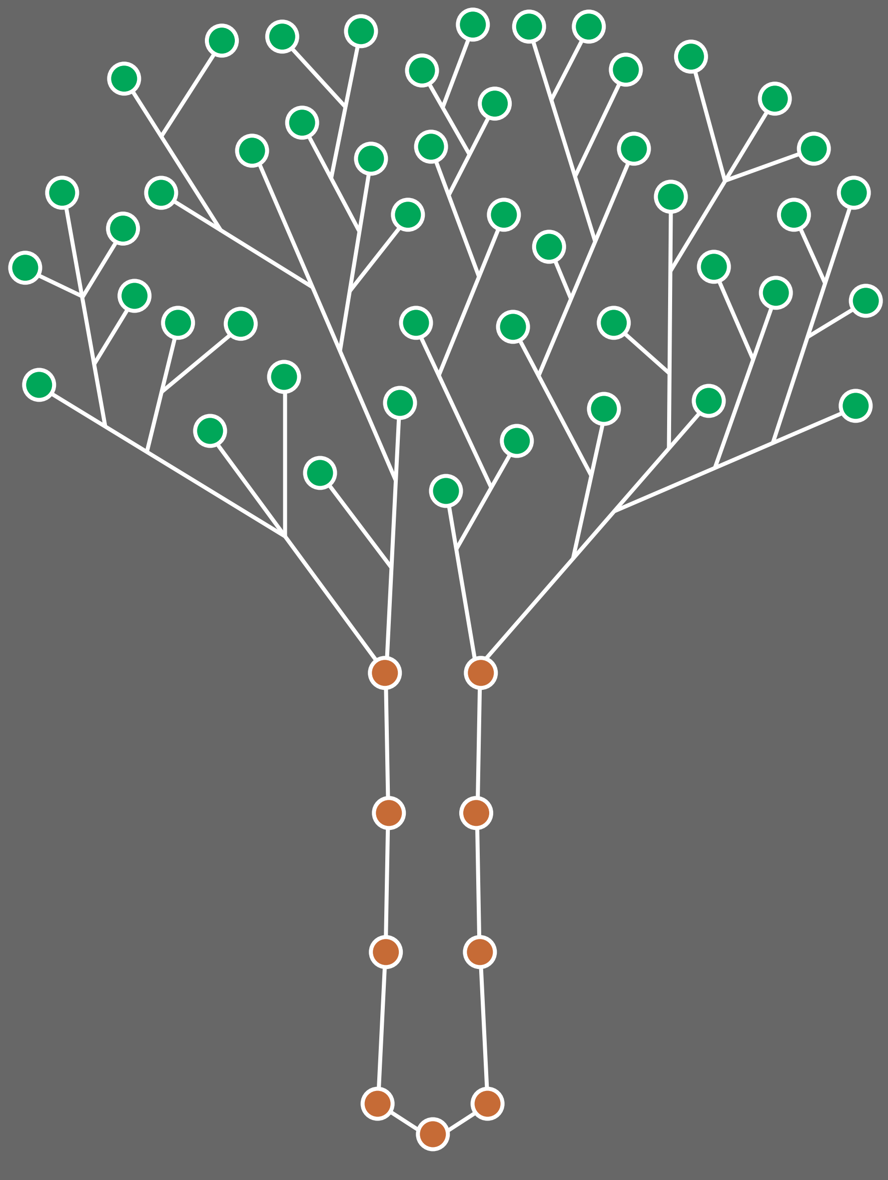
پرسش و پاسخ داده ساختارها



تالیف

مسعود فلاح پور

به نام یزدان نیک اندیش

پرسش و پاسخ داده ساختارها

تالیف
مسعود فلاح پور

مهر ۱۳۹۳
نگارش ۱.۰

مجوز

تقدیم به

پدر و مادرم؛ که همواره یار و یاورم بوده‌اند

فهرست مطالب

فهرست الگوریتم‌ها

پیش‌گفتار

در علم کامپیوتر درس‌هایی وجود دارند که از آنها به عنوان درس‌های بنیادین این علم یاد می‌شود. برخی از این درس‌ها عبارت‌اند از: داده‌ساختارها، تجزیه و تحلیل الگوریتم‌ها، طراحی کامپایلر و نظریه‌ی زبان‌ها و ماشین‌ها.

از میان درس‌های بنیادین علم کامپیوتر یکی از مهمترین آنها، درس داده‌ساختارها است که به عنوان پیشنیازی برای درس‌هایی همچون تجزیه و تحلیل الگوریتم‌ها و سیستم‌های عامل نیز مطرح است. اگر داده‌ساختار را به این صورت تعریف کنیم که «یک داده‌ساختار روشی برای ذخیره و سازماندهی داده‌ها است به طوریکه بازیابی و/یا تغییر داده‌ها به سادگی و با کارایی بالا انجام شود» آنگاه می‌توان به تعریفی از درس داده‌ساختارها نیز رسید. در درس داده‌ساختارها به بررسی دقیق و موشکافانه‌ی انواع داده‌ساختارها و چگونگی پیاده‌سازی آنها در یک زبان برنامه‌نویسی پرداخته می‌شود.

به دلیل اهمیت درس داده‌ساختارها کتاب‌های مختلفی در مورد آن نوشته شده است که بسیاری از آنها دارای قالب کم و بیش یکسانی هستند. قالب کلی این کتاب‌ها به این شکل است که در هر فصل از کتاب ابتدا به معرفی و بررسی یک داده‌ساختار خاص پرداخته شده و در انتهای فصل تمریناتی مرتبط با آن داده‌ساختار ارائه می‌شود. در کتاب حاضر سعی شده است از قالبی متفاوت استفاده شود.

در این کتاب فرض بر این است که خواننده با مباحث مختلف داده‌ساختارها آشنایی نسبی دارد و در نتیجه هر فصل از این کتاب دارای بخش نخست کتابهای معمول، یعنی معرفی و بررسی یک داده‌ساختار، نیست. تمرکز این کتاب بر روی مطرح کردن تعدادی سوال در مورد هر یک از انواع داده‌ساختارها و دادن پاسخ گام به گام و تشریحی به هر یک از سوالات است. به بیانی دیگر می‌توان قالب این کتاب را به صورت پرسش و پاسخ در نظر گرفت که به خواننده کمک می‌کند تا فهم عمیقتری از داده‌ساختارهای مختلف به دست آورد.

در ویرایش حاضر، تنها فصل‌های اول و دوم در کتاب گنجانده شده‌اند و سایر فصول پس از آماده‌سازی و کسب اطمینان از کیفیت علمی و ظاهری آنها در ویرایش‌های بعدی به کتاب اضافه خواهند شد.

متن کتاب با استفاده از سیستم حروفچینی لاتک ، بسته‌ی زی‌پرشین و ویرایشگر bidiTexmaker آماده شده است. برای متن پارسی از قلم XB Niloofar و برای کلمات انگلیسی و شبه‌کدها از قلم Computer Modern استفاده شده است. برای طراحی جلد کتاب از نرم‌افزار Corel DRAW و برای رسم شکل‌ها از بسته‌ی PSTricks و نرم‌افزار LaTeXDraw استفاده شده است. برای دسترسی به متن خام کتاب می‌توانید به نشانی https://github.com/MasoodFallahpoor/DS-Book مراجعه کنید.

در آماده‌سازی این کتاب تلاش شده است تا چه از نظر علمی و چه از نظر ظاهری کتابی شایسته و خالی از خطا به خوانندگان تقدیم شود. اما از آنجایی که هیچ کتابی نمی‌تواند به طور کامل از خطا در امان باشد از این رو از شما خواننده‌ی گرامی خواهشمندم در صورت مشاهده هرگونه خطای املایی، نگارشی و یا علمی به نشانی masood.fallahpoor@gmail.com اطلاع دهید تا خطای موجود در ویرایش‌های بعدی کتاب رفع شود.

مسعود فلاح‌پور
مهر ۱۳۹۳

درباره مولف

مسعود فلاح‌پور در سال ۱۳۶۷ در تهران متولد شد و تحصیلات اولیه خود را در همین شهر پشت سر گذاشت. او در سال ۱۳۸۷ مدرک کاردانی و در سال ۱۳۹۰ مدرک کارشناسی ناپیوسته خود را از دانشکده فنی شماره دو تهران (شهید شمسی‌پور) دریافت کرد. مسعود دارای مدرک کارشناسی ارشد مهندسی کامپیوتر در گرایش مهندسی نرم‌افزار از دانشکده مهندسی برق و کامپیوتر دانشگاه شهید بهشتی است.

مسعود به هر دو جنبه‌ی نظری و عملی علم کامپیوتر علاقه‌مند است. از جمله علاقه‌مندی‌های او در بخش نظری می‌توان به سیستم‌های عامل، داده‌ساختارها، طراحی الگوریتم‌ها و طراحی کامپایلر اشاره کرد. علاقه به سیستم عامل گنو/لینوکس، برنامه‌نویسی به زبان‌های جاوا و سی و همچنین برنامه‌نویسی اندروید از جمله علاقه‌مندی‌های او در بخش عملی است.

قدردانی

قدردانی و نام بردن از تمام افرادی که در به ثمر رسیدن این کتاب نقش داشته‌اند کاری است بس دشوار. به همین جهت فقط از برخی افراد نام برده خواهد شد.

بر خود لازم می‌دانم از آقای مهدی جوانمرد که ایده اولیه نوشتن این کتاب را مطرح کردند و همچنین جمع‌آوری بخشی از سوالات هر فصل را بر عهده داشتند صمیمانه سپاسگزاری کنم.

همچنین قدردانی می‌کنم از استاد گرامی، دکتر محسن ابراهیمی مقدم، که فهم دقیق و عمیق بسیاری از مفاهیم داده‌ساختارها را مدیون ایشان هستم.

در نهایت نیز از تلاش‌های چندین و چند ساله‌ی آقای وفا کارن پَهلَو برای توسعه‌ی بسته‌ی زی‌پرشین کمال قدردانی را دارم زیرا با خلق این بسته کمک شایانی به جامعه‌ی دانشگاهی ایران کرده‌اند.

قواعد شبه‌کد

برای بیان الگوریتم‌های بیان شده در کتاب، به جای استفاده از یک زبان برنامه‌نویسی خاص، از شبه‌کد استفاده شده است. با استفاده از شبه‌کد می‌توان الگوریتم‌ها را به شکلی ساده بیان کرد و از بیان جزئیات غیر ضروری خودداری کرد. در ادامه توضیحاتی در مورد کلیات شبه‌کد استفاده شده در کتاب بیان خواهد شد.

توضیحات

اگر در قسمتی از شبه کد نیاز به توضیح وجود داشته باشد، مانند زبان C++ از دو علامت اسلش پشت سرهم برای شروع توضیح استفاده می‌شود. در ادامه نمونه‌ای از یک توضیح آورده شده است.

```
// This is a comment
```

زیربرنامه‌ها

تمامی الگوریتم‌های کتاب به صورت زیربرنامه تعریف می‌شوند. یک زیربرنامه دارای دو نوع است: تابع و رویه. اگر زیربرنامه بخواهد مقداری را به عنوان خروجی بازگرداند آنگاه از تابع استفاده می‌کنیم و اگر مقداری را برنگرداند از رویه استفاده خواهیم کرد.

تعریف یک تابع با کلمه کلیدی function آغاز می‌شود. سپس نام تابع بیان می‌شود و در صورتی که تابع دارای ورودی باشد، ورودی‌های تابع در داخل پرانتز آورده می‌شوند. در ادامه‌ی تعریف تابع، بدنه تابع شروع می‌شود و در انتها مقداری به عنوان خروجی تابع توسط دستور return برگشت داده می‌شود. عبارت end function نیز خاتمه تعریف تابع را نشان می‌دهد. شکل کلی تعریف یک تابع در ادامه نشان داده شده است.

```
function FunctionName(param1, param2, ... , paramN)
    // body of function
    return result
end function
```

شکل کلی تعریف یک رویه هم مانند یک تابع است با این تفاوت‌ها که تعریف یک رویه با کلمه کلیدی procedure آغاز می‌شود، مقداری توسط رویه بازگردانده نمی‌شود و همچنین خاتمه رویه توسط عبارت end procedure مشخص می‌شود.

متغیرها

نوع متغیرهای مورد استفاده در شبه‌کد به صورت صریح بیان نمی‌شود زیرا با توجه به زیربرنامه‌ای که متغیر در آن استفاده شده است به راحتی می‌توان به نوع متغیرها پی برد. همچنین نیازی به تعریف متغیرها قبل از استفاده از آنها نیست و فرض بر این است که با اولین استفاده از یک متغیر، آن متغیر به صورت ضمنی تعریف نیز می‌شود.

آرایه‌ها

اندیس تمامی آرایه‌ها از عدد یک آغاز می‌شود مگر اینکه در یک شبه‌کد صراحتاً چیز دیگری بیان شود. برای دسترسی به خانه‌ی i ام آرایه یک بعدی A از قالب $A[i]$ و برای دسترسی به عنصر سطر i ام و ستون j ام آرایه دو بعدی B از قالب $B[i, j]$ استفاده می‌شود.

اگر متغیر A نشان دهنده یک آرایه یک بعدی باشد آنگاه طول این آرایه در خصیصه $length$ آن قرار دارد و برای دسترسی به آن از قالب $A.length$ استفاده می‌شود. اگر A یک آرایه دو بعدی باشد تعدادی سطرهای آن در خصیصه row و تعداد ستون‌های آن در خصیصه $column$ قرار دارد و برای دسترسی به آنها به ترتیب از قالب $A.rows$ و $A.columns$ استفاده می‌شود.

جهت اشاره به بازه‌ای از یک آرایه از قالب $A[i \,..\, j]$ استفاده می‌شود که در آن i اندیس شروع بازه و j اندیس پایان بازه است.

حلقه‌ها

برای تکرار یک تا تعدادی دستور از دو نوع حلقه استفاده خواهد شد: حلقه for و حلقه while. از ساختار حلقه for زمانی استفاده می‌شود که تعداد تکرار بدنه حلقه از قبل مشخص باشد و از حلقه while زمانی استفاده می‌شود که تعداد تکرار بدنه حلقه از قبل معلوم نباشد.

تعریف حلقه for با کلمه کلیدی for آغاز می‌شود. سپس مقدار اولیه شمارنده حلقه به متغیر شمارنده حلقه انتساب داده می‌شود و پس از کلمه کلیدی to مقدار نهایی شمارنده حلقه مشخص می‌شود. بعد از تعریف سرآیند حلقه، بدنه حلقه تعریف می‌دهد و در نهایت عبارت end for پایان حلقه را نشان می‌شود. در ادامه شکل کلی تعریف حلقه for نشان داده شده است.

for $counter = startValue$ **to** $endValue$
    // body of for loop
end for

با هر بار اجرای این حلقه یک واحد به متغیر شمارنده حلقه افزوده می‌شود و بدنه حلقه تا زمانی اجرا می‌شود که شرط $counter \leqslant endValue$ برقرار باشد. اگر بخواهیم شمارنده حلقه به جای افزایش، کاهش یابد آنگاه به جای کلمه کلیدی to از کلمه کلیدی downto استفاده می‌شود.

تعریف حلقه while با کلمه کلیدی while آغاز می‌شود. سپس یک عبارت منطقی قرار می‌گیرد و تا زمانی که عبارت منطقی برقرار باشد بدنه حلقه اجرا می‌شود. خاتمه حلقه while نیز با عبارت end while نشان داده می‌شود.

while $booleanExpression$
    // body of while loop
end while

دستورات شرطی

برای شروع یک دستور شرطی از کلمه کلیدی if استفاده می‌شود و در ادامه یک عبارت منطقی آورده می‌شود. اگر عبارت منطقی درست باشد دستورات بخش اول و در غیر این صورت دستورات بخش دوم اجرا می‌شوند. خاتمه تعریف دستور شرطی نیز با عبارت end if نشان داده می‌شود. برای تعریف یک دستور شرطی از قالب کلی زیر استفاده می‌شود.

if $booleanExpression$
    // statements to be executed when $booleanExpression$ is TRUE
else
    // statements to be executed when $booleanExpression$ is FALSE
end if

وجود بخش else اجباری نیست و این یعنی اگر این بخش وجود نداشته باشد و شرط دستور شرطی برقرار نباشد آنگاه بدنه دستور شرطی اجرا نخواهد شد.

دستور return

با اجرای دستور return اجرای زیربرنامه بلافاصله پایان می‌یابد. از این دستور در صورت نیاز برای بازگرداندن مقدار یا مقادیری به عنوان خروجی یک تابع نیز می‌توان استفاده کرد. در ادامه شکل‌های مختلف دستور return آورده شده است.

return
return result
return (result1, result2, ... , resultN)

در صورتی که شکل خروجی تابعی مانند $\textsc{Func}(A, B)$ مانند حالت سوم دستور return باشد آنگاه از شکل زیر برای دریافت تمامی خروجی‌های آن استفاده خواهد شد. با اجرای دستور زیر مقدار result1 در var1 قرار می‌گیرد، result2 در var2 قرار می‌گیرد و به همین ترتیب تا resultN که در varN قرار می‌گیرد.

var1, var2, ... , varN $=\textsc{Func}(A, B)$

عملگرها

عملگرهای منطقی مورد استفاده در الگوریتم‌ها عبارت‌اند از $>$، $<$، $\leq$، $\geq$ و == که از عملگر آخر برای بررسی تساوی دو مقدار استفاده می‌شود.

برای ترکیب عبارات منطقی از عملگرهای and، or و not استفاده می‌شود. جهت انتساب مقداری به یک متغیر از عملگر $=$ استفاده می‌شود.

برای انجام چهار عمل اصلی ریاضی از عملگرهای $+$، $-$، $\times$ و $/$ استفاده می‌شود. همچنین از عملگر mod به عنوان عملگر باقیمانده استفاده می‌شود.

فصل ۴

درخت‌های عمومی، دودویی و هیپ

۱.۴ مقدمه

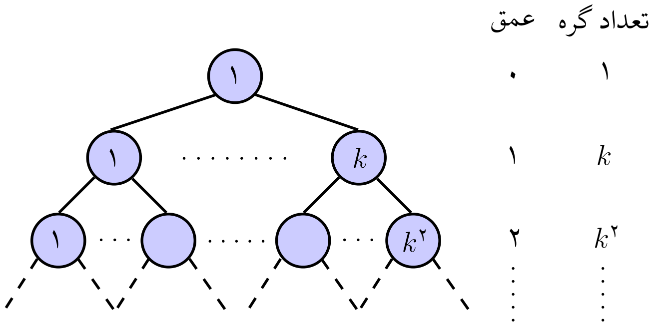
۲.۴ منابع مطالعاتی

۳.۴ فرمول‌های درخت

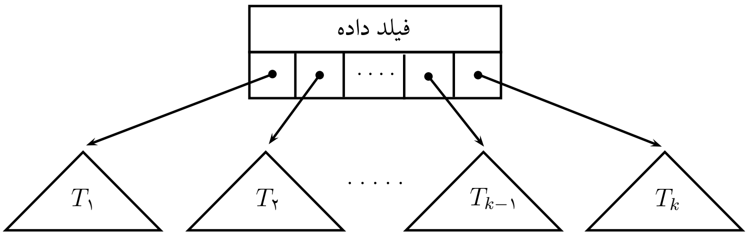
◂ **سوال ۱.** ثابت کنید عمق یک درخت kتایی پر با n گره، برابر با $\lfloor \log_k n \rfloor$ است (عمق گره‌ی ریشه را صفر در نظر بگیرید).

▹ **پاسخ سوال ۱.**

در عمق صفر یک درخت kتایی پر دارای تنها یک گره هستیم که همان گره‌ی ریشه است. در عمق یک چنین درختی دارای k گره خواهیم بود زیرا درخت پر است و گره ریشه باید دارای دقیقاً k فرزند باشد. در عمق دو دارای $k^۲$ گره خواهیم بود و به همین ترتیب. به عبارت بهتر تعداد گره‌ها در عمق d ($d \neq ۰$)، k برابر تعداد گره‌های عمق $d - ۱$ است. برای درک بهتر این موضوع به شکل ۱.۴ توجه کنید.



شکل (۱.۴): بخشی از یک درخت kتایی پر



شکل (۲.۴): ساختار یک گره در یک درخت kتایی

با توجه به شکل تعداد کل گره‌ها، که آن را با n نمایش می‌دهیم، در یک درخت kتایی پر به عمق m به صورت زیر به دست می‌آید:

$$\begin{aligned} n &= 1 + k + k^2 + k^3 + \cdots + k^m \\ &= \frac{k^{m+1} - 1}{k - 1} \end{aligned} \tag{۱.۴}$$

اگر در عبارت $\lfloor \log_k n \rfloor$ به جای n، مقدار به دست آمده از ۱.۴ را قرار دهیم باید داشته باشیم:

$$\left\lfloor \log_k \frac{k^{m+1} - 1}{k - 1} \right\rfloor = m \tag{۲.۴}$$

به این ترتیب باید ثابت کنیم که تساوی ۲.۴ برقرار است. در ادامه، به اثبات برقراری این تساوی پرداخته می‌شود.

با در نظر گرفتن خاصیت تابع جزء صحیح می‌توان تساوی ۲.۴ را به صورت زیر نوشت:

$$m \leq \log_k \frac{k^{m+1} - 1}{k - 1} < m + 1 \tag{۳.۴}$$

اگر طرفین نامعادله ۳.۴ را به عنوان توان عدد k در نظر بگیریم خواهیم داشت:

$$k^m \leq k^{\log_k \frac{k^{m+1} - 1}{k - 1}} < k^{m+1} \Rightarrow k^m \leq \frac{k^{m+1} - 1}{k - 1} < k^{m+1} \tag{۴.۴}$$

با ضرب طرفین نامعادله ۴.۴ در $k - 1$ داریم:

$$k^m(k - 1) \leq k^{m+1} - 1 < k^{m+1}(k - 1) \tag{۵.۴}$$

با ساده‌سازی نامعادله ۵.۴ به عبارت زیر می‌رسیم:

$$k^{m+1} - k^m \leq k^{m+1} - 1 < k^{m+1}(k - 1) \tag{۶.۴}$$

درستی نامعادله ۶.۴ را می‌توان با استقرا بر روی m نشان داد و به این ترتیب اثبات کامل است.

◄ **سوال ۲.** فرض کنید برای پیاده‌سازی درخت‌های kتایی از گره‌هایی با قالب نشان داده شده در شکل ۲.۴ استفاده کرده‌ایم. در چنین ساختاری دارای یک فیلد داده‌ای و k فیلد اشاره‌گر برای اشاره به هر یک از k زیردرخت یک گره هستیم. اگر دارای n گره در چنین درختی باشیم آنگاه تعداد اشاره‌گرهای تهی چه تعداد خواهد بود؟

◁ **پاسخ سوال ۲.**

در چنین ساختاری هر گره دارای k اشاره‌گر است. در نتیجه هنگامی که n گره در درخت وجود دارد دارای nk اشاره‌گر هستیم. از این تعداد اشاره‌گر فقط $n-1$ اشاره‌گر مورد استفاده قرار می‌گیرند زیرا برای اشاره به هر یک از گره‌ها به جز گره ریشه به یک اشاره‌گر نیاز است. اگر تعداد کل اشاره‌گرها را از تعداد اشاره‌گرهای مورد استفاده کم کنیم خواهیم داشت:

$$nk-(n-1)=nk-n+1=n(k-1)+1$$

به این ترتیب تعداد اشاره‌گرهای تهی برابر با $n(k-1)+1$ است.

۴.۴ درخت‌های عمومی و دودویی

◂ **سوال ۳.** با فرض پیاده‌سازی درخت عمومی به کمک اشاره‌گرها و اینکه هر گره موظف است آدرس فرزندان خود را در یک لیست پیوندی نگاه دارد، تابعی بازگشتی بنویسید که بررسی کند آیا عنصری با مقدار x در درخت داده شده وجود دارد یا خیر؟ اگر x موجود بود مقدار TRUE و در غیر اینصورت مقدار FALSE را به عنوان خروجی تابع برگردانده شود.

◁ **پاسخ سوال ۳.**

شبه کد تابعی بازگشتی که در درخت t به دنبال عنصری با مقدار x می‌گردد در الگوریتم ۱.۴ آمده است.

الگوریتم ۱.۴ یافتن عنصری با مقدار مشخص در یک درخت عمومی

```
function FIND(t, x)
    if t == NULL
        return
    end if
    if t.data == x
        return TRUE
    end if
    found = FALSE
    p = t.children
    while p ≠ NULL and found == FALSE
        found = FIND(p.data, x)
        p = p.next
    end while
    return found
end function
```

در ابتدا بررسی می‌شود که آیا درخت تهی است یا خیر. اگر اینگونه بود مقدار FALSE به عنوان خروجی برگشت داده می‌شود. اگر درخت تهی نبود باید بررسی کرد که آیا مقدار عنصر ریشه برابر با x است یا خیر. اگر برابر بود آنگاه مقدار TRUE برگشت داده می‌شود. اگر هیچ یک از دو شرط ابتدایی برقرار نبودند آنگاه باید به دنبال مقدار x در فرزندان گره ریشه و در صورت لزوم در فرزندانِ فرزندان گره ریشه و به همین ترتیب بگردیم تا یا عنصر مورد نظر یافته شود و یا از عدم وجود آن در درخت اطمینان حاصل نماییم. جستجو در فرزندان ریشه، فرزندانِ فرزندان ریشه و ... توسط حلقه موجود در تابع به صورت بازگشتی انجام می‌شود. برای درک بهتر این الگوریتم آن را بر روی یک درخت عمومی ساده امتحان کنید.

◂ **سوال ۴.** با در نظر گرفتن پیاده‌سازی درخت عمومی به کمک اشاره‌گرها و اینکه هر گره موظف است آدرس فرزندان خود را در یک لیست پیوندی نگاه دارد، تابعی بازگشتی بنویسید که اشاره‌گر به یک گره را دریافت کرده و اشاره‌گر به پدر گره‌ی ورودی را بازگرداند. (تابع شما باید دارای تنها دو ورودی باشد. ورودی اول درختی است که باید جستجو در آن انجام شود و ورودی دوم گره‌ای است که قصد یافتن پدر آن را داریم).

▹ **پاسخ سوال ۴.**

شبه کد تابعی بازگشتی که در درخت t به دنبال پدر گره‌ای می‌گردد که n به آن اشاره دارد در الگوریتم ۲.۴ آورده شده است.

اگر درخت t تهی باشد آنگاه عمل خاصی لازم نیست و مقدار تهی برگشت داده می‌شود. در صورتی که درخت t تهی نباشد آنگاه باید در فرزندان گره ریشه به دنبال عنصری بگردیم که n به آن اشاره دارد. اگر چنین عنصری یافت شد آنگاه مقدار t به عنوان خروجی بازگردانده می‌شود و به این معنی است که گره‌ای که n به آن اشاره دارد یکی از فرزندان گره‌ی ریشه است و در نتیجه گره‌ی ریشه پدر آن است (خطوط ۵ تا ۱۱). اگر عنصری که n به آن اشاره دارد یکی از فرزندان گره‌ی ریشه نباشد آنگاه باید به صورت بازگشتی در فرزندان گره‌ی ریشه به دنبال عنصر n بگردیم (خطوط ۱۲ تا ۱۹). اگر به خط انتهایی تابع برسیم بدین معنی است که قادر به یافتن عنصری که n به آن اشاره دارد در درخت t نبوده‌ایم و در نتیجه مقدار تهی به عنوان خروجی تابع برگردانده می‌شود.

◂ **سوال ۵.** تابعی بازگشتی بنویسید که با دریافت یک درخت عمومی، پیاده‌سازی شده توسط لیست عمومی، درخت دودویی معادلش را برگرداند.

▹ **پاسخ سوال ۵.**

تابع بازگشتی مورد نظر در قالب الگوریتم ۳.۴ آورده شده است.

◂ **سوال ۶.** پیاده‌سازی درخت دودویی با استفاده از آرایه را در نظر بگیرید. برای ذخیره‌سازی یک درخت دودویی با n گره، در بدترین حالت به آرایه‌ای چند خانه‌ای نیاز داریم؟ چه تعداد از این خانه‌ها بدون استفاده باقی می‌مانند؟

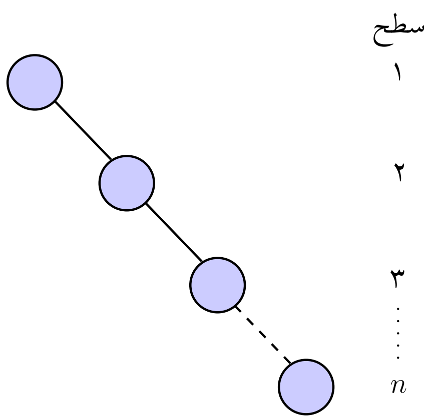
▹ **پاسخ سوال ۶.**

در صورتی که درخت دودویی مورب به راست و هر سطح از درخت دارای تنها یک گره باشد آنگاه بیشترین تعداد خانه مورد نیاز خواهد بود. شکل ۳.۴ نشان دهنده‌ی یک درخت دودویی مورب به راست است.

در چنین درختی، که دارای n سطح است، دارای تنها n گره هستیم اما تعداد خانه‌های لازم برای ذخیره‌سازی

الگوریتم ۲.۴ یافتن پدر یک گره‌ی خاص در یک درخت عمومی

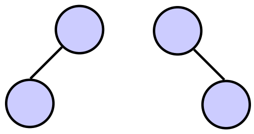
```
function FindParent(t, n)
    if t == NULL
        return NULL
    end if
    p = t.children
    while p ≠ NULL
        if p.data == n
            return t
        end if
        p = p.next
    end while
    p = t.children
    while p ≠ NULL
        q = FindParent(p.data, n)
        if q ≠ NULL
            return q
        end if
        p = p.next
    end while
    return NULL
end function
```



شکل (۳.۴): درخت دودویی مورب به راست

الگوریتم ۳.۴ تبدیل درخت عمومی به درخت دودویی معادل

```
function GenToBin(genTree)
    if genTree == NULL
        return NULL
    end if
    New(q)
    q.data = genTree.data
    m = genTree.link
    if m == NULL
        q.leftChild = NULL
        q.rightChild = NULL
    else
        q.leftChild = GenToBin(m.ChildrenList)
        p = q.leftChild
        p.rightChild = NULL
        m = m.link
        while m ≠ NULL
            p.rightChild = GenToBin(m.ChildrenList)
            p = p.rightChild
            p.rightChild = NULL
            m = m.link
        end while
    end if
    return q
end function
```



شکل (۴.۴): درخت‌های دودویی مختلفی که با دو گره می‌توان ساخت

چنین درختی برابر با تعداد خانه‌های لازم برای یک درخت دودویی پر با n سطح است. تعداد خانه‌های لازم برای ذخیره‌سازی یک درخت دودویی پر با n سطح برابر است با:

$$۱ + ۲ + ۲^۲ + ۲^۳ + \cdots + ۲^n = \frac{۲^n - ۱}{۲ - ۱} = ۲^n - ۱$$

در نتیجه، به $۲^n - ۱$ خانه احتیاج خواهیم داشت که از این تعداد فقط n خانه مورد استفاده قرار می‌گیرد و $۲^n - ۱ - n$ خانه از آرایه بدون استفاده باقی می‌ماند.

◀ **سوال ۷.** تعداد درخت‌های دودویی مختلفی که با صفر گره می‌توان ساخت برابر با یک است (درخت تهی). تعداد درخت‌های دودویی مختلفی که با یک گره می‌توان ساخت نیز برابر با یک است و چنین درختی فقط دارای عنصر ریشه است. درخت‌های دودویی مختلفی که می‌توان با دو گره ساخت در شکل ۴.۴ نشان داده شده است. تعداد درخت‌های دودویی مختلفی که با n گره می‌توان ساخت چه تعداد است؟

◁ **پاسخ سوال ۷.**

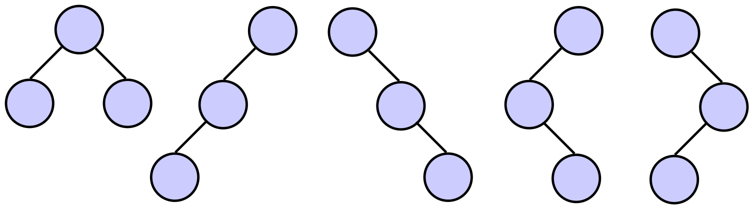
فرض کنید دارای n گره هستیم که آنها را از ۱ تا n شماره‌گذاری کرده‌ایم و این گره‌ها را به صورت زیر در کنار هم قرار داده‌ایم:

$$۱, ۲, ۳, \cdots, n$$

با در نظر گرفتن این شماره‌گذاری، مجدداً فرض کنید که گره شماره‌ی i $(۱ \leqslant i \leqslant n)$ ریشه‌ی یک درخت دودویی باشد و تمام گره‌های با شماره کمتر از i در زیردرخت چپ i و تمام گره‌های با شماره بیشتر از i در زیردرخت راست i قرار بگیرند. در این‌صورت زیردرخت چپ ریشه دارای $i - ۱$ گره و زیردرخت راست دارای $n - i$ گره خواهد بود.

اگر $T(n)$ را برابر با تعداد درختهایی که می‌توان با n گره ساخت در نظر بگیریم آنگاه تعداد زیردرخت‌های چپ و راست درخت با ریشه i به ترتیب برابر با $T(i - ۱)$ و $T(n - i)$ خواهند بود. با توجه به اصل شماره دو شمارش، یعنی اصل ضرب، حاصل ضرب $T(i - ۱) \cdot T(n - i)$ برابر با تعداد کل درخت‌های ممکن با ریشه i است. i می‌تواند هر یک از مقادیر ۱ تا n را اختیار کند. در نتیجه به منظور در نظر گرفتن کلیه‌ی حالات گره‌ی ریشه، باید مقدار عبارت $T(i - ۱) \cdot T(n - i)$ را برای همه‌ی مقادیر i با یکدیگر جمع کنیم. بدین ترتیب به رابطه بازگشتی ۷.۴ می‌رسیم.

$$T(n) = \begin{cases} T(۰) = T(۱) = ۱ & i = ۰, ۱ \\ \sum_{i=۱}^{n} T(i - ۱) \cdot T(n - i) & i > ۱ \end{cases} \tag{۷.۴}$$



شکل (۵.۴): درخت‌های دودویی مختلفی که با سه گره می‌توان ساخت

با حل رابطه‌ی بازگشتی ۷.۴ به رابطه‌ی ۸.۴ می‌رسیم.

$$T(n) = \frac{\binom{2n}{n}}{n+1} \tag{۸.۴}$$

به این ترتیب و با استفاده از رابطه‌ی ۸.۴ می‌توان تعداد درخت‌های دودویی مختلفی که با n گره می‌توان ساخت را به دست آورد. برای مثال تعداد درخت‌های دودویی مختلفی که با ۳ گره می‌توان ساخت برابر است با:

$$T(3) = \frac{\binom{6}{3}}{4} = \frac{20}{4} = 5$$

این درخت‌ها در شکل ۵.۴ نشان داده شده‌اند. به دنباله‌ی اعداد حاصل از رابطه‌ی ۸.۴، دنباله اعداد کاتالان گفته می‌شود.

◄ **سوال ۸.** تابعی بازگشتی بنویسید که با دریافت درخت دودوییِ معادلِ یک درخت عمومی، درخت عمومی را بازسازی کند (تنها ورودی تابع شما باید درخت دودویی باشد).

◁ **پاسخ سوال ۸.**

شبه کد تابع تبدیل یک درخت دودویی به درخت عمومی معادل در الگوریتم ۴.۴ نشان داده شده است.

◄ **سوال ۹.** تابعی بازگشتی بنویسید که اشاره‌گر به ریشه‌ی یک درخت دودویی را دریافت کرده و تعداد سطوح درخت را بازگرداند.

◁ **پاسخ سوال ۹.**

◄ **سوال ۱۰.** پهنای یک درخت برابر است با تعداد گره‌های سطحی که دارای بیشترین تعداد گره در میان تمام سطوح درخت است. برای مثال درخت نشان داده شده در شکل (۶.۴) دارای پهنای چهار است زیرا سطح اول دارای یک گره، سطح دوم دارای دو گره، سطح سوم دارای چهار گره و سطح چهارم دارای یک گره است و در نتیجه حداکثر تعداد گره در یک سطح از این درخت برابر با چهار است و این یعنی پهنای درخت نیز برابر با چهار است. تابعی بازگشتی بنویسید که اشاره‌گر به ریشه‌ی یک درخت دودویی را به عنوان ورودی دریافت کرده و پهنای درخت را به عنوان خروجی بازگرداند.

◁ **پاسخ سوال ۱۰.**

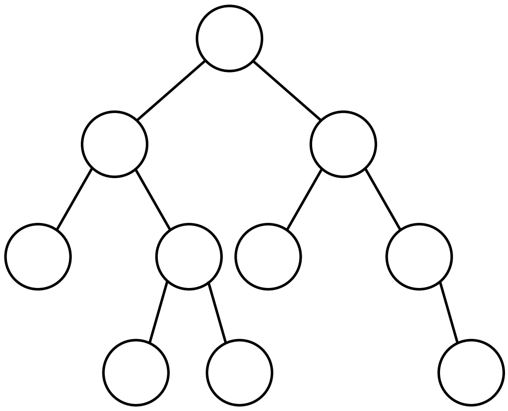
شبه کد تابع به دست آوردن پهنای یک درخت دودویی در الگوریتم (۶.۴) نشان داده شده است. این تابع

الگوریتم ۴.۴ تبدیل درخت دودویی به درخت عمومی معادل

```
function BinToGen(binTree)
    if binTree == NULL
        return NULL
    end if
    New(t)
    t.tag = TRUE
    t.data = binTree.data
    t.link = NULL
    q = t
    p = binTree.leftChild
    while p ≠ NULL
        New(q.link)
        q = q.link
        q.tag = FALSE
        q.dlink = BinToGen(p)
        q.link = NULL
        p = p.rightChild
    end while
    return t
end function
```

الگوریتم ۵.۴ به دست آوردن تعداد سطوح یک درخت دودویی

```
function TreeLevel(t)
    if t == NULL
        return 0
    end if
    leftLevel = TreeLevel(t.leftChild)
    rightLevel = TreeLevel(t.rightChild)
    if leftLevel > rightLevel
        return (leftLevel + 1)
    else
        return (rightLevel + 1)
    end if
end function
```



شکل (۶.۴): درخت دودویی با پهنای چهار

علاوه بر پهنای درخت، شماره سطحی را که تعیین کننده پهنای درخت هست نیز برمی‌گرداند. روش کار این تابع در ادامه بیان می‌شود.

زمانی که درخت ورودی تهی است مقدار صفر هم برای سطح و هم برای پهنای درخت بازگردانده می‌شود. اگر درخت ورودی دارای تنها یک عنصر باشد، که همان عنصر ریشه است، مقدار یک هم برای سطح و هم برای پهنای درخت بازگردانده می‌شود و این یعنی درخت دارای پهنای یک است و همچنین سطحی که دارای این پهنا است سطح شماره یک است. زمانی که درخت ورودی دارای بیش از یک عنصر است از فراخوانی بازگشتی استفاده می‌کنیم تا پهنای زیردرخت چپ و راست گره ریشه را به دست بیاوریم. چون پهنای زیر درخت چپ و راست گره ریشه به صورت جداگانه محاسبه می‌شوند باید بررسی کنیم تا در صورتی که $leftLevel$ برابر با $rightLevel$ باشد، مقادیر $leftWidth$ و $rightWidth$ با یکدیگر جمع شوند تا پهنای کلی درخت به دست آید. در صورتی که $leftLevel$ برابر با $rightLevel$ نباشد آنگاه مقدار بزرگتر از میان دو مقدار $leftWidth$ و $rightWidth$ به عنوان پهنای درخت و سطح متناظر با مقدار بزرگتر به عنوان سطحی که تعیین کننده پهنای درخت است به عنوان خروجی تابع بازگردانده می‌شوند.

◄ **سوال ۱۱.** ساده‌ترین الگوریتم برای یافتن دومین بزرگترین عنصر در یک آرایه‌ی n عنصری نیاز به تقریباً $۲n$ مقایسه دارد. گام‌های کلی الگوریتمی را شرح دهید که دومین بزرگترین مقدار در یک آرایه را حداکثر $n + ۲\lfloor \lg n \rfloor - ۲$ مقایسه پیدا می‌کند (راهنمایی: فرض کنید تعداد عناصر آرایه زوج است و از درخت برنده‌ـ بازنده[1] استفاده کنید).

◁ **پاسخ سوال ۱۱.**

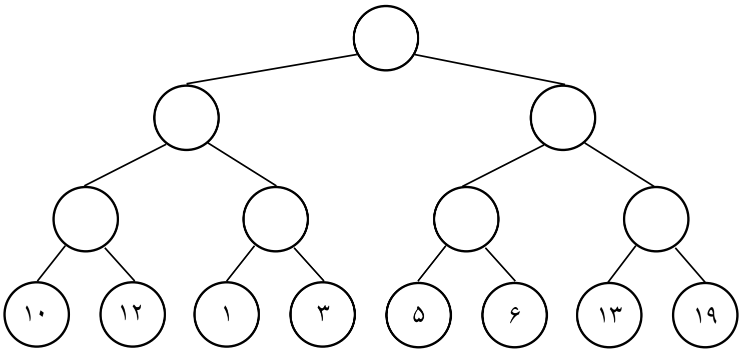
این الگوریتم دارای دو مرحله است و در هر مرحله درختی دودویی با نام درخت برنده‌ـ بازنده ساخته می‌شود.

در یک درخت برنده‌ـ بازنده تمامی عناصر آرایه ورودی به عنوان برگ‌های درخت قرار می‌گیرند. اگر فرض کنیم آرایه ورودی دارای هشت خانه است و حاوی اعداد ۱۹، ۱۳، ۶، ۵، ۳، ۱، ۱۲، ۱۰ است آنگاه درخت برنده‌ـ بازنده‌ی ابتدایی در مرحله اول اجرای الگوریتم به صورت شکل (۷.۴) خواهد بود. در هر سطح، عناصری که دارای پدر یکسان هستند با یکدیگر مقایسه شده و مقدار بزرگتر به سطح بالاتر یعنی به گره‌ی پدر آن دو گره‌ای که با یکدیگر مقایسه شده‌اند کپی می‌شود. این روند به همین ترتیب ادامه می‌یابد تا در نهایت

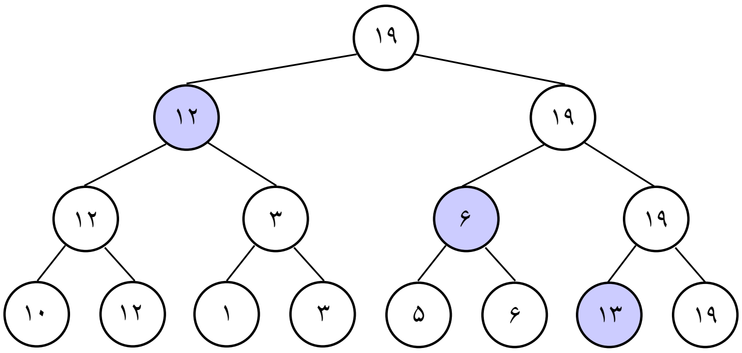
[1] winner-loser tree

الگوریتم ۶.۴ به دست آوردن پهنای یک درخت دودویی

```
function TreeWidth(t)
    if t == NULL
        return (0,0)
    end if
    if t.leftChild == NULL and t.rightChild == NULL
        return (1,1)
    end if
    leftLevel, leftWidth = TreeWidth(t.leftChild)
    rightLevel, rightWidth = TreeWidth(t.rightChild)
    if leftLevel == rightLevel
        width = leftWidth + rightWidth
        level = leftLevel
    else
        width = Max(leftWidth, rightWidth)
        if leftWidth > rightWidth
            width = leftWidth
            level = leftLevel
        else
            width = rightWidth
            level = rightLevel
        end if
    end if
    return (level, width)
end function
```



شکل (۷.۴): درخت برنده‌ـ بازنده ابتدایی در مرحله اول از اجرای الگوریتم



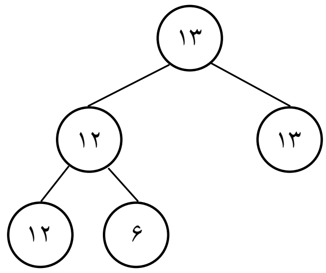
شکل (۸.۴): درخت برنده‌ـ بازنده نهایی در مرحله اول از اجرای الگوریتم

گره ریشه بدست آید که همان بزرگترین مقدار آرایه است. درخت نهایی مرحله اول اجرای الگوریتم در شکل (۸.۴) به نمایش در آمده است.

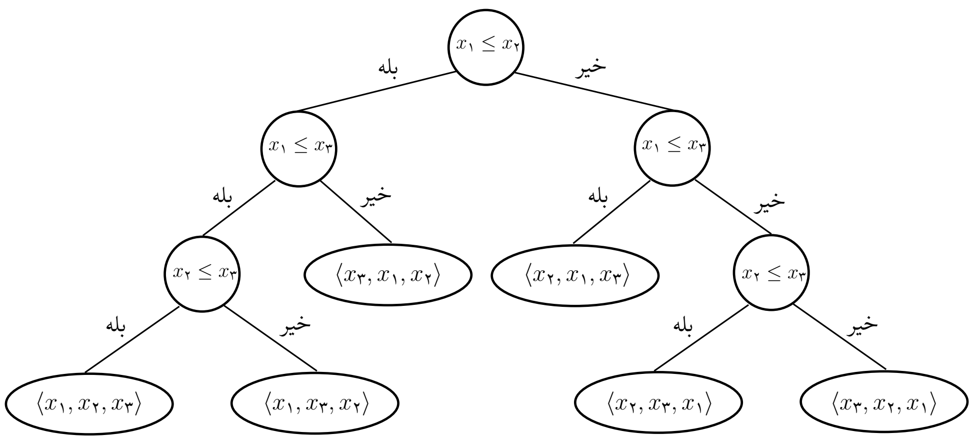
در این درخت با هر مقایسه‌ای که انجام می‌شود یک برنده، یعنی عنصر با مقدار بزرگتر، و یک بازنده، یعنی عنصر با مقدار کوچکتر، مشخص می‌شوند و مقدار عنصر برنده به سطح بالاتر منتقل می‌شود. با توجه به اینکه هر عنصری غیر از بزرگترین عنصر، دقیقاً یک بار باید بازنده شده باشد می‌توان گفت که تعداد مقایسات لازم برای ساخت چنین درختی برابر با $n - ۱$ است.

در مرحله دوم اجرای الگوریتم می‌توان گفت دومین بزرگترین عنصر، عنصری است که تنها به بزرگترین عنصر، یعنی عنصر ریشه، باخته است. در نتیجه کافی است در مسیر حرکت از گره‌ی ریشه به سمت پایین، در مسیری که بزرگترین عنصر برای رسیدن به گره ریشه طی کرده است حرکت کرده و مقادیری که در مقایسه با گره‌ی ریشه بازنده بوده‌اند را مشخص کنیم (گره‌های خاکستری در شکل (۸.۴)). با توجه به دودویی بودن درخت و تعداد سطوح چنین درختی، تعداد عناصر بازنده به بزرگترین عنصر حداکثر برابر با $\lceil \lg n \rceil$ خواهد بود. لذا $\lceil \lg n \rceil$ مقایسه نیاز خواهیم داشت تا تمام عناصر بازنده به بزرگترین عنصر را به دست بیاوریم. حال برای این $\lceil \lg n \rceil$ عنصر نیز یک درخت برنده‌ـ بازنده می‌سازیم (شکل (۹.۴)) و سپس با $\lceil \lg n \rceil - ۱$ مقایسه بزرگترین عنصر این درخت که همان دومین بزرگترین عنصر آرایه است را می‌یابیم. بدین ترتیب می‌توان گفت حداکثر تعداد مقایسات برای یافتن دومین بزرگترین عنصر برابر با $n + ۲\lceil \lg n \rceil - ۲$ است.

◂ **سوال ۱۲.** ثابت کنید هر الگوریتم مرتب‌سازی که از مقایسه‌ی عناصر برای تعیین ترتیب صحیح عناصر استفاده می‌کند در بهینه‌ترین حالت از مرتبه $\Omega(n \lg n)$ است.



شکل (۹.۴): درخت برنده– بازنده نهایی در مرحله دوم از اجرای الگوریتم



شکل (۱۰.۴): درخت تصمیم دودویی برای مرتب‌سازی سه عنصر

◁ **پاسخ سوال ۱۲.**

در ابتدا یک مثال ساده را بررسی می‌کنیم. فرض کنید قصد مرتب‌سازی آرایه‌ای سه عنصری را داریم و هر سه عنصر متمایز از یکدیگر هستند. برای نمایش روند مرتب‌سازی عناصر می‌توان از یک درخت دودویی استفاده کرد. در چنین درختی برچسب هر گره نمایش‌دهنده یک مقایسه میان دو عنصر آرایه است و برچسب هر یال نمایش دهنده‌ی نتیجه مقایسه دو عنصر است. ترتیب مرتب‌شده عناصر را می‌توان با حرکت از ریشه‌ی درخت تا یکی از برگ‌ها بدست آورد زیرا هر برگ بیانگر یکی از جایگشت‌های ممکن عناصر آرایه است. به چنین درختی، یک درخت تصمیم دودویی[۲] گفته می‌شود. شکل (۱۰.۴) نشان دهنده‌ی درخت تصمیم دودویی برای یک آرایه سه عنصری است (در این شکل x_i به معنی مقدار خانه iام آرایه است).

هر الگوریتم مرتب‌سازی مبتنی بر مقایسه، با توجه به مقایساتی که انجام می‌دهد، یک درخت تصمیم دودویی را ایجاد می‌کند. در چنین درختی، طولانی‌ترین مسیر از گره ریشه به یک گره برگ بیانگر بدترین حالت ممکن در اجرای الگوریتم است. یعنی در چنین حالتی الگوریتم برای مرتب‌سازی عناصر به بیشترین تعداد مقایسات نیاز دارد. همچنین بهترین حالت در اجرای الگوریتم معادل با کوتاه‌ترین مسیر از گره ریشه به یک گره برگ است. حالت متوسط نیز از تقسیم تعداد یال‌های موجود در درخت بر تعداد برگ‌های موجود بدست می‌آید. در حالت متوسط در حقیقت مشخص می‌کنیم به طور متوسط تعداد یال‌هایی که باید طی شوند تا به یک برگ برسیم چه تعداد است.

اگرچه ممکن است در نگاه اول بتوان با رسم درخت تصمیم برای هر الگوریتم، به تعیین طول کوتاهترین و

[۲] Binary decision tree

طولانی‌ترین مسیر پرداخت امّا حالتی را در نظر بگیرید که قصد مرتب‌سازی آرایه‌ای با ۱۰ عنصر را داریم. درخت تصمیم برای چنین آرایه‌ای دارای حداقل !۱۰ برگ است (چون ممکن است برخی از جایگشت‌ها بیش از یکبار ظاهر شوند دارای حداقل این تعداد برگ هستیم) و دارای حداقل ۲۲ سطح است. در نتیجه رسم چنین درختی با این ابعاد منطقی به نظر نمی‌رسد. پس باید دید چگونه می‌توان با استفاده از ایده‌ی درخت تصمیم دودویی به مرتبه زمانی $\Omega(n \lg n)$ رسید.

باید در حالت کلی بررسی کنیم که حداقل عمق یک درخت تصمیم برای مرتب‌سازی n عنصر چه مقداری است. با تعیین این مقدار در حقیقت حداقل تعداد مقایسات در یک الگوریتم مرتب‌سازی مبتنی بر مقایسه را به دست آورده‌ایم.

می‌دانیم که یک درخت تصمیم دودویی برای آرایه‌ای n عنصری دارای حداقل $n!$ برگ است. در ادامه حداقل عمق ممکن برای درخت تصمیمی با $n!$ برگ را محاسبه می‌کنیم. اگر عمق ریشه را برابر با یک در نظر بگیریم می‌توان گفت در عمق فرضی k تعداد گره‌ها برابر است با $۲^{k-۱}$. لذا برای داشتن درختی با حداقل $n!$ برگ، اگر k را کمترین عمق ممکن برای درخت در نظر بگیریم آنگاه باید داشته باشیم:

$$n! \leq ۲^{k-۱} \qquad (۹.۴)$$

با لگاریتم گرفتن از طرفین نامعادله (۹.۴) به نامعادله (۱۰.۴) می‌رسیم.

$$\lg n! \leq k - ۱ \qquad (۱۰.۴)$$

با درنظر گرفتن این که $\lg n!$ تقریباً برابر با $n \lg n - ۱/۵$ است (چرا؟) می‌توان گفت حداقل مقدار k تقریباً برابر با $n \lg n$ است. بدین ترتیب حداقل عمق یک درخت تصمیم دودویی با $n!$ برگ برابر با $n \lg n$ است و این یعنی تعداد مقایسات در یک الگوریتم مرتب‌سازی مبتنی بر مقایسه در بهینه‌ترین حالت از مرتبه $\Omega(n \lg n)$ است. در نتیجه می‌توان گفت هیچ الگوریتم مرتب‌سازی مبتنی بر مقایسه نمی‌تواند در زمانی بهتر از $\Omega(n \lg n)$ آرایه‌ای n عنصری را مرتب کند.

۵.۴ درخت‌های هیپ

◄ **سوال ۱۳.** در یک درخت هیپ بیشینه با n عنصر متمایز، اعمال زیر را با چه مرتبه‌ای می‌توان انجام داد؟ توضیح دهید.

- به دست آوردن مجموع همه‌ی اعداد موجود در هیپ
- به دست آوردن مجموع $\lg n$ بزرگترین اعداد موجود در هیپ
- به دست آوردن مجموع ۱۰ عدد بزرگ موجود در هیپ

◁ **پاسخ سوال ۱۳.**

برای به دست آوردن مجموع اعداد موجود در درخت هیپ کافیست درخت را با استفاده از یک پیمایش دلخواه، مثلا میان‌ترتیب، پیمایش کرده و در حین پیمایش با ملاقات هر گره مقدار آن را به مقدار مجموع

اضافه کنیم. با توجه به اینکه مرتبه پیمایش میان‌ترتیب برابر با $O(n)$ است پس مرتبه اجرایی این الگوریتم نیز برابر با $O(n)$ است.

برای به دست آوردن مجموع $\lg n$ بزرگترین اعداد موجود در هیپ می‌توان به تعداد $\lg n$ بار عمل حذف از ریشه را انجام داد و مقدار عدد حذف شده را به مقدار مجموع اضافه کرد. به این ترتیب در بار اول بزرگترین عدد، که در ریشه هیپ است، حذف شده و مقدار آن به مقدار مجموع اضافه می‌شود سپس دومین بزرگترین عنصر حذف شده و به مقدار مجموع اضافه شده و به همین ترتیب. با توجه به اینکه هر عمل حذف از هیپ از مرتبه $O(\lg n)$ است پس مرتبه کلی الگوریتم برابر خواهد بود با $\lg n \times O(\lg n) = O(\lg^۲ n)$.

به دست آوردن مجموع ۱۰ عدد بزرگتر هیپ حالت خاصی از حالت قبلی است. یعنی کافیست ۱۰ بار عنصر ریشه را حذف کرده و هربار مقدار عنصر حذف شده را به یک متغیر که حاصل جمع را نگه می‌دارد اضافه کنیم. مرتبه زمانی انجام چنین کاری برابر است با $O(۱۰ \times \lg n) = O(\lg n)$

◄ **سوال ۱۴.** در یک درخت هیپ بیشینه حاوی n عدد متمایز، چهارمین بزرگترین عدد ممکن است در کدامیک از خانه‌های آرایه‌ی حاوی هیپ بیشینه قرار بگیرد؟

◁ **پاسخ سوال ۱۴.**

برای تعیین اینکه چهارمین بزرگترین مقدار می‌تواند در کدامیک از خانه‌های آرایه قرار بگیرد باید به طور کلی به دنبال این باشیم که kامین بزرگترین مقدار در چه سطوحی از درخت هیپ می‌تواند ظاهر شود. سپس شماره‌ی خانه‌های متناظر این سطوح در آرایه را یافته و به این ترتیب محدوده‌ای از خانه‌های آرایه که می‌تواند حاوی kامین بزرگترین مقدار باشد مشخص می‌شود.

بزرگترین مقدار همواره در سطح یک درخت که تنها شامل گره‌ی ریشه است قرار می‌گیرد. دومین بزرگترین مقدار می‌تواند در یکی از گره‌های سطح دوم قرار بگیرد. سومین بزرگترین مقدار می‌تواند در یکی از گره‌های سطوح دو یا سه قرار بگیرد. می‌توان اثبات کرد که kامین بزرگترین مقدار می‌تواند در یکی از سطوح ۲ تا k قرار بگیرد (طبق آنچه گفته شد برای حالت خاص $k = ۱$ این رابطه برقرار نیست و بزرگترین مقدار در سطح یک قرار خواهد گرفت).

بزرگترین مقدار در خانه‌ی شماره یک آرایه قرار می‌گیرد (با فرض شروع شماره‌ی اندیس خانه‌های آرایه از یک). دومین بزرگترین مقدار می‌تواند در سطح دو (خانه‌های دو یا سه) یا سطح سه (خانه‌های چهار تا هفت) قرار بگیرد (در مجموع خانه‌های دو تا هفت). به همین ترتیب می‌توان گفت kامین بزرگترین مقدار می‌تواند در یکی از خانه‌های ۲ تا $۲^k - ۱$ قرار بگیرد.

با توجه به اینکه یک فرمول کلی به دست آوردیم در نتیجه می‌توان گفت چهارمین بزرگترین مقدار می‌تواند در یکی از خانه‌های دو تا پانزده قرار بگیرد.

◄ **سوال ۱۵.** فرض کنید یک درخت هیپ بیشینه حاوی اعداد ۱ تا ۱۰۲۳ موجود است. چه تعداد از اعداد بزرگتر از ۱۰۰۰ می‌توانند در گره‌های برگ چنین درختی قرار بگیرند؟ حداکثر چه تعداد از این اعداد می‌توانند همزمان برگ باشند؟

◁ **پاسخ سوال ۱۵.**

برای پاسخ به این سوال به دو نکته‌ی زیر توجه می‌کنیم:

۱. در یک درخت هیپ kامین بزرگترین عنصر می‌تواند در یکی از سطوح دوم، سوم، ...، kام قرار بگیرد.

۲. یک درخت هیپ که حاوی اعداد ۱ تا ۱۰۲۳ است دارای ده سطح است.

قسمت اول:

با در نظر گرفتن دو نکته‌ی مطرح شده می‌توان گفت عدد ۱۰۲۳ در سطح اول، عدد ۱۰۲۲ در سطح دوم، عدد ۱۰۲۱ در یکی از سطوح دوم یا سوم و به همین ترتیب تا عدد ۱۰۱۵ که در یکی از سطوح دوم تا نهم قرار می‌گیرند. در نتیجه می‌توان گفت اعداد ۱۰۱۵ تا ۱۰۲۳ نمی‌توانند در سطح دهم قرار بگیرند و این یعنی این اعداد نمی‌توانند به عنوان برگ درخت هیپ ظاهر شوند. به بیان دیگر چهارده عدد بزرگتر از ۱۰۰۰ این امکان را دارند که به عنوان برگ درخت هیپ ظاهر شوند و این اعداد عبارتند از ۱۰۰۱ تا ۱۰۱۴.

قسمت دوم:

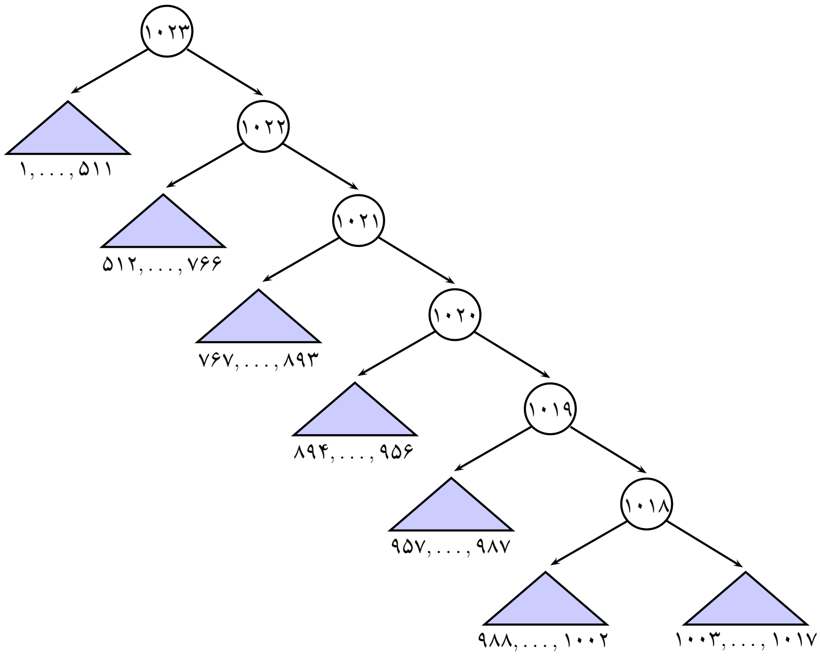
برای پاسخ به این قسمت باید طوری اعداد را در هیپ بیشینه قرار دهیم که تا حد ممکن بیشترین اعداد در بازه ۱۰۰۱ تا ۱۰۲۳ در گره‌های برگ ظاهر شوند.

می‌دانیم که بزرگترین مقدار یعنی ۱۰۲۳ در سطح اول، دومین بزرگترین مقدار یعنی ۱۰۲۲ در سطح دوم، سومین بزرگترین مقدار در یکی از سطوح دوم یا سوم و به همین ترتیب ظاهر می‌شوند. حال برای آنکه بیشترین تعداد از اعداد در بازه ۱۰۰۱ تا ۱۰۲۳ در برگ‌ها ظاهر شوند، سعی می‌کنیم اعداد ۱ تا ۵۱۱ را زیردرخت چپ ریشه و اعداد ۵۱۲ تا ۱۰۲۲ را در زیردرخت راست ریشه قرار دهیم (می‌توانستیم به صورت برعکس نیز عمل کنیم بدین معنی که اعداد ۱ تا ۵۱۱ را زیردرخت راست ریشه و اعداد ۵۱۲ تا ۱۰۲۲ را در زیردرخت چپ ریشه قرار دهیم). با توجه به اینکه اعداد مورد نظر ما یعنی ۱۰۰۱ تا ۱۰۲۲ در زیردرخت راست قرار دارند در نتیجه برای ادامه کار فقط به زیردرخت راست توجه می‌کنیم و زیردرخت چپ ریشه را نادیده می‌گیریم.

در زیردرخت راست، عدد ۱۰۲۲ را به عنوان گره ریشه این زیردرخت در نظر می‌گیریم و در زیردرخت چپ این گره اعداد ۵۱۲ تا ۷۶۶ و در زیردرخت راست آن اعداد ۷۶۷ تا ۱۰۲۱ را قرار می‌دهیم. مجدداً برای زیردرخت با ریشه ۱۰۲۲ سعی می‌کنیم اعداد ۷۶۷ تا ۸۹۳ را در زیردرخت چپ و اعداد ۸۹۴ تا ۱۰۲۱ را در زیردرخت راست قرار دهیم. اگر همین روند را ادامه دهیم در سطح ششم درخت به حالتی می‌رسیم که در آن باید اعداد ۹۸۸ تا ۱۰۱۸ را در درخت قرار دهیم. در این حالت نیز به این صورت عمل می‌کنیم که عدد ۱۰۱۸ را در ریشه، اعداد ۹۸۸ تا ۱۰۰۲ را در زیردرخت چپ و اعداد ۱۰۰۳ تا ۱۰۱۷ را در زیردرخت راست قرار می‌دهیم. درخت هیپ مورد نظر تا این مرحله در شکل ۱۱.۴ نشان داده شده است.

در این حالت می‌توان گفت زیردرخت حاوی اعداد ۱۰۰۳ تا ۱۰۱۷ دارای چهار سطح است و در نتیجه دارای هشت برگ است. به این ترتیب اعداد ۱۱۶، ۱۱۵ و ۱۱۴ با توجه به اینکه اولین، دومین و سومین بزرگترین مقادیر در این زیردرخت هستند در نتیجه نمی‌توانند در سطح چهارم این زیردرخت (یعنی به عنوان گره‌ی برگ) ظاهر شوند. پس می‌توان گفت در زیردرخت راست گره‌ای که دارای مقدار ۱۰۱۸ در ریشه خود است، هشت عنصر بزرگتر از ۱۰۰۰ و کوچکتر از ۱۰۱۴ می‌توانند به طور همزمان در برگهای درخت هیپ ظاهر شوند.

در مورد ۱۰۰۱ و ۱۰۰۲ نیز می‌توان گفت با توجه به اینکه ۱۰۰۱ و ۱۰۰۲ در زیردرخت حاوی این اعداد، به ترتیب اولین و دومین بزرگترین عناصر هستند در نتیجه در زیردرخت خود نمی‌توانند به صورت برگ ظاهر شوند.



شکل (۱۱.۴): درخت هیپ بیشینه حاوی اعداد ۱ تا ۱۰۲۳

به عنوان نتیجه‌گیری می‌توان گفت اعداد ۱۰۰۱ تا ۱۰۱۴ این امکان را دارند که به عنوان گره‌ی برگ در درخت هیپ بیشینه ظاهر شوند اما فقط هشت عدد از میان اعداد ۱۰۰۳ تا ۱۰۱۳ این امکان را داردند که به طور *همزمان* در گره‌های برگ ظاهر شوند.

◄ **سوال ۱۶.** داده‌ساختار صف اولویت میانه[۳]، یک داده‌ساختار شامل n عنصر با مقادیر متمایز است که می‌توان اعمال زیر را بر روی آن انجام داد:

- درج یک عنصر در زمان $O(\lg n)$
- حذف عنصر دارای مقدار میانه در زمان $O(\lg n)$

با استفاده از داده‌ساختار درخت هیپ، داده‌ساختار صف اولویت میانه را طراحی کنید. سپس نحوه‌ی انجام اعمال نامبرده با پیچیدگی زمانی خواسته شده را شرح دهید.

◁ **پاسخ سوال ۱۶.**

عنصر میانه در یک لیست، عنصری است که از نیمی از داده‌های لیست بزرگتر و از نیمی از داده‌ها کوچکتر است. از همین ویژگی برای طراحی داده‌ساختار صف اولویت میانه استفاده خواهیم کرد. در لیستی با تعداد فرد عنصر، عنصر میانه عنصری است که پس از مرتب کردن عناصر لیست در وسط لیست قرار می‌گیرد. به طور مثال لیست $L = \langle ۴, ۲, ۳, ۵, ۱ \rangle$ را در نظر می‌گیریم. پس از مرتب کردن عناصر لیست L، به لیست $L' = \langle ۱, ۲, ۳, ۴, ۵ \rangle$ می‌رسیم. بدین ترتیب میانه لیست D عدد ۳ خواهد بود. به عبارت بهتر عنصر میانه در لیستی با فرد عنصر پس از مرتب‌سازی در مکان $\lfloor n/۲ \rfloor + ۱$ قرار خواهد گرفت. در لیستی با زوج عنصر، با توجه به اینکه دارای عنصر وسط نیستیم، می‌توان پس از مرتب کردن لیست، عنصر موجود در مکان $\lfloor n/۲ \rfloor$ یا $\lfloor n/۲ \rfloor + ۱$ را به عنوان عنصر میانه در نظر گرفت.

[۳] Median priority queue

فرض کنید می‌خواهیم عنصر میانه را از لیست $L = \langle ۴, ۲, ۳, ۵, ۱ \langle$ حذف کنیم. در این صورت باید عدد ۲ (بزرگترین عددی که کوچکتر از میانه است) یا ۴ (کوچکترین عددی که از میانه بزرگتر است) را به عنوان عنصر میانه‌ی جدید برگزینیم. برای این منظور می‌توان دو درخت هیپ در نظر گرفت که یکی از آنها هیپ بیشینه و دیگری هیپ کمینه است. عناصر کمتر از میانه‌ی جاری را در هیپ بیشینه و عناصر بیشتر از میانه‌ی جاری را در هیپ کمینه نگه می‌داریم. شکل داده‌ساختار برای لیستی مانند $L = \langle ۴, ۲, ۳, ۵, ۱ \langle$ به صورت زیر خواهد بود:

می‌توانیم عنصر میانه را در یکی از دو درخت هیپ نیز قرار دهیم. برای مثال اگر عنصر میانه را در هیپ بیشینه قرار دهیم به شکل زیر می‌رسیم:

در ادامه فرض می‌کنیم که از حالت ذخیره‌سازی دوم استفاده شده است. با این روش پیاده‌سازی، برای حذف عنصر میانه، در شرایطی که تعداد عناصر هیپ بیشینه و هیپ کمینه با یکدیگر برابر نباشند، همواره عنصر ریشه‌ی درختی که تعداد عناصرش یکی بیشتر از دیگری است را به عنوان میانه جدید انتخاب می‌کنیم. اما اگر تعداد عناصر هر دو هیپ با یکدیگر برابر باشند می‌توان هر کدام از دو ریشه‌ی هیپ‌های مذکور را به عنوان میانه‌ی جدید انتخاب کرد.

در ادامه بدین صورت قرارداد می‌کنیم که اگر تعداد عناصر هر دو هیپ با یکدیگر برابر بودند آنگاه عنصر ریشه هیپ بیشینه به عنوان میانه جدید در نظر گرفته خواهد شد. در ادامه الگوریتم‌های مربوط به اعمال درج و حذف را شرح داده و نشان می‌دهیم که هر دو از مرتبه $O(\lg n)$ هستند.

برای حذف عنصر میانه در ابتدا باید بررسی کنیم که هر کدام از دو درخت هیپ دارای چه تعداد عنصر هستند. برای این کار می‌توان از دو متغیر استفاده کرد که هر کدام تعداد عناصر یکی از هیپ‌ها را نگه می‌دارند. اگر تعداد عناصر دو هیپ با یکدیگر برابر نبود، کافیست ریشه‌ی درخت هیپی که یک عنصر بیشتر دارد را به عنوان میانه‌ی جدید برگزینیم و آن ریشه را از هیپ حذف کنیم که در این صورت تعداد عناصر در هر دو هیپ با یکدیگر برابر خواهد شد. اما در صورتی که تعداد عناصر در هر دو هیپ با یکدیگر برابر باشد آنگاه عنصر ریشه هیپ بیشینه به عنوان میانه جدید در نظر گرفته می‌شود و این عنصر از درخت هیپ بیشینه حذف می‌شود. همانطور که می‌دانیم مرتبه زمانی عمل حذف یک عنصر از یک هیپ n عنصری برابر با $O(\lg n)$ است. اگر تعداد عناصر هیپ بیشینه را $n_{۱}$ و تعداد عناصر هیپ کمینه را $n_{۲}$ در نظر بگیریم آنگاه حذف ریشه از هیپ بیشینه از مرتبه $O(\lg n_{۱})$ و حذف ریشه از هیپ کمینه از مرتبه $O(\lg n_{۲})$ است. از طرفی می‌دانیم که $n = n_{۱} + n_{۲}$ و اختلاف $n_{۱}$ و $n_{۲}$ حداکثر یک واحد است. پس می‌توان گفت که حذف ریشه از هر کدام از درخت‌ها از مرتبه $O(\lg(n/۲))$ است که این نیز برابر با $O(\lg n)$ است.

در هنگام درج در این داده‌ساختار باید کاری کرد که اختلاف تعداد عناصر هیپ‌های کمینه و بیشینه حداکثر یک باشد. در ادامه مقداری که قرار است درج شود را a، عنصر ریشه هیپ بیشینه را x و عنصر ریشه هیپ کمینه را y می‌نامیم. a را با x و y مقایسه می‌کنیم. سه حالت ممکن به شرح زیر خواهند بود:

۱. $y \leq a$: در این صورت a باید در هیپ کمینه، که شامل عناصر بزرگتر از میانه است، درج شود. اگر تعداد عناصر هیپ کمینه، کمتر یا مساوی تعداد عناصر هیپ بیشینه باشد، عمل درج را به راحتی انجام می‌دهیم. در چنین حالتی تعداد عناصر هیپ کمینه برابر یا یکی بیشتر از عناصر هیپ بیشینه خواهد شد. اما در صورتی که تعداد عناصر هیپ کمینه یکی بیشتر از هیپ بیشینه باشد، درج عنصر جدید در هیپ کمینه سبب خواهد شد که تعداد عناصر این هیپ دو واحد بیشتر از تعداد عناصر هیپ بیشینه

شود که چنین حالتی قابل قبول نخواهد بود. در نتیجه پیش از درج عنصر جدید در هیپ کمینه، ابتدا ریشه‌ی هیپ کمینه، یعنی y، را حذف کرده و مقدار حذف شده را در هیپ بیشینه درج می‌کنیم. سپس a را در هیپ کمینه درج می‌کنیم که با انجام این کار دوباره اختلاف حداکثر یک واحدی تعداد عناصر دو هیپ نسبت به یکدیگر حفظ خواهد شد.

۲. اگر $x \leq a$: مراحل کلی عمل درج شبیه به حالت قبل است با این تفاوت که درج در هیپ بیشینه انجام خواهد شد.

۳. اگر $y \leq a \leq x$: در این حالت تعداد عناصر هیپ‌های بیشینه و کمینه را با یکدیگر مقایسه می‌کنیم. اگر تعداد عناصر هر دو هیپ برابر بود، می‌توان عمل درج را در هر یک از هیپ‌ها انجام داد. اما اگر تعداد عناصر دو هیپ یکسان نبود، درج را در هیپ با تعداد عناصر کمتر انجام می‌دهیم.

با توجه به توضیحات ارائه شده، برای درج یک عنصر در صف اولویت میانه در بدترین حالت باید یک عمل حذف و دو عمل درج در هیپ انجام داد. در نتیجه مرتبه زمانی عمل درج در بدترین حالت برابر با $O(۳ \times \lg(n/۲))$ است که این نیز از مرتبه‌ی $O(\lg n)$ است.

◂ **سوال ۱۷.** درستی عبارت زیر را اثبات کنید:

تعداد گره‌های با ارتفاع h در یک درخت هیپ n عنصری، حداکثر برابر با $\lceil n/۲^{h+۱} \rceil$ است.

▹ **پاسخ سوال ۱۷.**

اثبات را با استقرا بر روی h انجام می‌دهیم.

پایه‌ی استقرا: باید نشان دهیم هنگامی که $h = ۰$ است تعداد گره‌های با ارتفاع h، کمتر یا مساوی $\lceil n/۲^{h+۱} \rceil$ است. به عبارت دیگر باید نشان دهیم تعداد گره‌های با ارتفاع $h = ۰$ کوچکتر یا مساوی $\lceil n/۲ \rceil$ است.

اگر فرض کنیم درخت هیپ دارای عمق D است آنگاه گره‌های با ارتفاع صفر، که همان گره‌های برگ هستند، می‌توانند در عمق D یا $D - ۱$ باشند و هر یک از این گره‌ها در یکی از دو حالت زیر صدق می‌کنند:

- گره در عمق D قرار دارد.
- گره در عمق $D - ۱$ قرار دارد و این گره پدر هیچ گره‌ای در عمق D نیست.

حال فرض می‌کنیم x تعداد گره‌های موجود در عمق D باشد. اگر n را تعداد کل گره‌های درخت در نظر بگیریم آنگاه $n - x$ عددی فرد است زیرا $n - x$ گره‌ی موجود، تشکیل یک درخت دودویی با حداکثر تعداد گره را می‌دهند. به عبارت دیگر با نادیده گرفتن x گره‌ی موجود در عمق D دارای درختی کامل با $n - x$ گره خواهیم بود. به این ترتیب می‌توان تعداد گره‌های درخت را برابر با $n = (۲^D - ۱) + x$ در نظر گرفت. با توجه به فرد بودن مقدار $n - x$، اگر n فرد باشد آنگاه x زوج است و اگر n زوج باشد x فرد است.

برای اثبات پایه‌ی استقرا باید دو حالت زوج و فرد بودن n را به صورت جداگانه درنظر بگیریم. برای ادامه‌ی اثبات توجه داشته باشید که در عمق d به شرطی که $d < D$ باشد دارای ۲^d گره هستیم زیرا درخت در چنین عمقی دارای حداکثر تعداد گره است.

اگر n فرد باشد آنگاه x زوج است. چون x زوج است پس دقیقاً $x/۲$ گره در عمق $D - ۱$ قرار دارند که والدین x گره‌ی موجود در عمق D هستند. این بدین معنی است که تعداد گره‌های عمق $D - ۱$ که پدر هیچ

گره‌ای در عمق D نیستند برابر است با:

$$۲^{D-۱} - \frac{x}{۲} \tag{۱۱.۴}$$

این یعنی علاوه بر x گره‌ی برگ موجود در عمق D دارای $۲^{D-۱} - x/۲$ گره برگ در عمق $D - ۱$ نیز هستیم. پس تعداد کل گره‌های برگ برابر خواهد بود با $x + (۲^{D-۱} - x/۲)$ که این مقدار معادل با $۲^{D-۱} + x/۲$ است. این مقدار را می‌توان به صورت $(۲^D + x)/۲$ نیز نوشت. با توجه به اینکه x زوج است مقدار $(۲^D + x)/۲$ معادل با $\lceil(۲^D + x - ۱)/۲\rceil$ است. حال اگر به جای عبارت $۲^D + x - ۱$ مقدار معادل آن یعنی n را جایگزین کنیم آنگاه عبارت $\lceil(۲^D + x - ۱)/۲\rceil$ تبدیل به $\lceil n/۲\rceil$ می‌شود. بدین ترتیب نشان دادیم که وقتی n فرد است تعداد گره‌های با عمق صفر برابر با $\lceil n/۲\rceil$ است.

حال اگر فرض کنیم n زوج است آنگاه x فرد خواهد بود و با استدلالی شبیه به استدلال انجام شده در حالت فرد بودن n، تعداد گره‌های با عمق صفر به صورت زیر به دست می‌آید:

$$\begin{aligned} n_۰ &= x + \left(۲^{D-۱} + \frac{x+۱}{۲}\right) \\ &= ۲^{D-۱} + \frac{x-۱}{۲} \\ &= \frac{۲^{D-۱} - ۱ + x}{۲} \\ &= \frac{n}{۲} \end{aligned} \tag{۱۲.۴}$$

چون n زوج است در نتیجه می‌توان تساوی (۱۲.۴) را معادل با $n_۰ = \lceil n/۲\rceil$ در نظر گرفت. به این ترتیب نشان دادیم وقتی n زوج باشد تعداد گره‌های با ارتفاع صفر برابر با $\lceil n/۲\rceil$ است.

با توجه به اینکه هم برای n فرد و هم برای n زوج نشان دادیم که تعداد گره‌های برگ در ارتفاع صفر حداکثر برابر با $\lceil n/۲\rceil$ است در نتیجه می‌توان گفت پایه‌ی استقرا برقرار است.

فرض استقرا: فرض می‌کنیم در یک درخت هیپ حداکثر تعداد گره‌های برگ با ارتفاع $h - ۱$ برابر با $\lceil n/۲^h\rceil$ است.

حکم استقرا: نشان می‌دهیم در یک درخت هیپ حداکثر تعداد گره‌های برگ با ارتفاع h برابر با $\lceil n/۲^{h+۱}\rceil$ است.

فرض می‌کنیم n_h تعداد گره‌های برگ با عمق h در یک درخت هیپ n عنصری به نام T باشد. T' را درختی در نظر می‌گیریم که با حذف برگ‌های درخت T حاصل شده است. به این ترتیب تعداد گره‌های درخت T'، که آن را با n' نشان می‌دهیم، برابر است با $n' = n - n_۰$. با توجه به اثباتی که برای پایه‌ی استقرا انجام دادیم می‌دانیم که $n_۰ = \lceil n/۲\rceil$ است. پس مقدار n' را می‌توان به صورت زیر نوشت:

$$n' = n - n_۰ = n - \left\lceil\frac{n}{۲}\right\rceil = \left\lfloor\frac{n}{۲}\right\rfloor$$

توجه به این نکته ضروری است که اگر ارتفاع گره‌ای در درخت T برابر با h باشد آنگاه چنین گره‌ای در درخت T' دارای ارتفاع $h - ۱$ است. با توجه به این نکته و تعریف $n'_{h-۱}$ به عنوان تعداد گره‌های با ارتفاع $h - ۱$

در درخت T' می‌توان گفت $n_h = n'_{h-1}$. حال با استفاده از فرض استقرا داریم:

$$n_h = n'_{h-1} \leq \left\lceil \frac{n'}{2^h} \right\rceil = \left\lceil \frac{\lfloor n/2 \rfloor}{2^h} \right\rceil \leq \left\lceil \frac{n/2}{2^h} \right\rceil = \left\lceil \frac{n}{2^{h+1}} \right\rceil$$

بدین ترتیب اثبات کامل است.

کتاب‌نامه

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 3rd ed. , 2009.

[۲] محسن ابراهیمی مقدم، آذرخش کی‌پور و حسین عبدی. *ساختمان‌داده‌ها و الگوریتم‌ها.* انتشارات نصیر، ویرایش اول، ۱۳۹۲.

[3] E. Horowitz, S. Sahni, and S. Anderson-Freed. *Fundamentals of Data Structures in C.* Silicon Press, 2nd ed. , 2007.