دانشکده‌ی علوم ریاضی

داده‌ساختارها و الگوریتم‌ها — ۲۸ آبان ۹۲

جلسه‌ی ۱۵: مدل‌های محاسباتی، لیست‌های زنجیره‌ای و درخت‌ها

مدرّس: دکتر شهرام خزائی — *نگارنده: شفیع قلیزاده*

۱ مدل‌های محاسباتی

برای نمایش برنامه‌های رایانه‌ای از زبان‌های برنامه‌نویسی استفاده می‌شود. به طور معادل برای نمایش الگوریتم‌ها از شبه‌کد استفاده می‌گردد. چنان که می‌دانیم، زبان‌های برنامه‌نویسی در رایانه اجرا می‌شوند؛ اما به طور معادل برای اجرا و آزمایش شبه‌کدها چه می‌توان گفت؟
شبه‌کدها در مدل‌های محاسباتی مورد آزمایش قرار می‌گیرند. در حقیقت مدل‌های محاسباتی همانند نسخه‌های نظری رایانه‌ها عمل می‌کنند؛ درست همان گونه که شبه‌کدها نسخه‌های نظری زبان‌های برنامه‌نویسی هستند و نیز همان گونه که الگوریتم‌ها نسخه‌های نظری برنامه‌های رایانه‌ای هستند. در هر مدل محاسباتی مشخص می‌گردد که الگوریتم‌ها شامل چه نوع عملیاتی می‌توانند باشند و نیز هزینه هر نوع عملیات معیّن می‌گردد. مدل RAM[۱] و ماشین‌های اشاره‌گر[۲] دو گونه متفاوت از مدل‌های محاسباتی هستند که احتمالا با آنها زیاد سر و کار داشته‌ایم.
مدل محاسباتی RAM مانند یك حافظه بزرگ اما محدود از کلمات عمل می‌کند که در آن هر کلمه طولی برابر با w بیت دارد. این مدل محاسباتی می‌تواند در یك مدت زمان محدود، تعداد معدودی کلمه را بخواند، تعداد معدودی محاسبه انجام دهد و تعداد معدودی کلمه را ذخیره کند.
در ماشین اشاره‌گر، اشیاء می‌توانند طول‌های متفاوتی داشته باشند. هر شیء تعداد محدودی مؤلفه[۳] دارد. هر مؤلفه می‌تواند یك کلمه، یك اشاره‌گر[۴] به شیء دیگر و یا پوچ[۵] باشد.

۲ پیاده‌سازی لیست‌های زنجیره‌ای دوسویه با استفاده از آرایه

در جلسه گذشته با لیست‌های زنجیره‌ای دوسویه آشنا شدیم. در اینجا می‌خواهیم به نحوه پیاده‌سازی این لیست‌ها در در مدل RAM با استفاده از آرایه بپردازیم. برای پیاده‌سازی می‌توانیم از آرایه‌های یك بعدی و یا چندبعدی استفاده کنیم. استفاده از آرایه چندبعدی برای پیاده‌سازی ساده‌تر است، اما در مدل RAM آرایه چندبعدی نیز نهایتاً به آرایه یك بعدی تبدیل می‌شود.

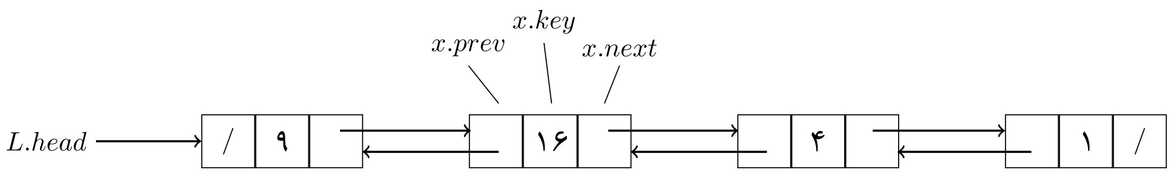
[۱] Random Access Machine
[۲] Pointer Machine
[۳] Field
[۴] Pointer
[۵] null

۱.۲ پیاده‌سازی لیست با استفاده از آرایه چندبعدی

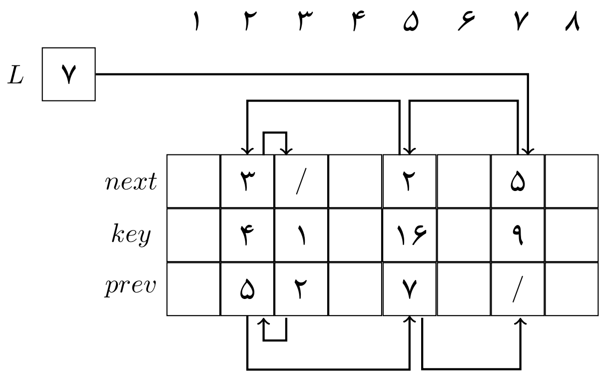
در اینجا می‌خواهیم پیاده‌سازی لیست دوسویه را در یك آرایه چندبعدی بررسی کنیم. برای سادگی فرض می‌کنیم که هر شیء موجود در لیست علاوه بر مؤلفه‌هایی که به اشیاء قبلی و بعدی آن اشاره می‌کنند، تنها یک مؤلفه دیگر برای ذخیره کلید آن داشته باشد. به طور دقیق‌تر اگر x اشاره‌گری به یک شیء موجود در لیست باشد، سه مؤلفه آن $x.key$، $x.next$ و $x.prev$ هستند. برای پیاده‌سازی چنین لیستی به یک آرایه سه‌بعدی نیاز داریم. برای این منظور به ازای هر شیء موجود در لیست، مقدار key را در آدرسی از سطر دوم آرایه، مقدار اشاره‌گر $next$ را در همان آدرس از سطر اول و مقدار اشاره‌گر $prev$ را در همان آدرس از سطر سوم آرایه وارد می‌کنیم. همچنین باید آدرس نخستین شیء موجود در لیست را به طور جداگانه در اشاره‌گری مانند L ذخیره کنیم.

قرارداد ۱ *برای نمایش شکلی لیست‌ها هر شیء را به صورت تعدادی خانه نشان می‌دهیم. به عنوان مثال اشیاء ۲مؤلفه‌ای، ۳مؤلفه‌ای و ۴مؤلفه‌ای معمولا به ترتیب با مستطیل‌های ۱ × ۲، ۱ × ۳ و ۲ × ۲ نمایش داده می‌شوند. هر خانه هر شیء بیانگر یکی از مؤلفه‌های شیء است. اگر مؤلفه‌ای دارای ارزش مقداری باشد، مقدار آن را در خانه متناظر می‌نویسیم. اگر مؤلفه‌ای از شیء x محتوی یک اشاره‌گر به شیء y باشد، یک پیکان از خانه متناظر در شیء x به شیء y رسم می‌کنیم. اگر مؤلفه‌ای از یک شیء محتوی اشاره‌گر تهی باشد، نماد / را در خانه متناظر می‌نویسیم.*

مثال ۱ *شکل زیر بیانگر یک لیست شامل چهار شیء ۳مؤلفه‌ای است.*

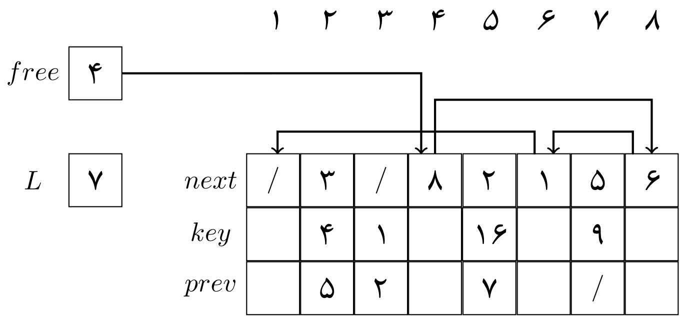


مثال ۲ *می‌خواهیم لیست زنجیره‌ای دوسویه مثال فوق را در یک آرایه سه‌بعدی پیاده‌سازی نماییم. نمونه‌ای از پیاده سازی را می‌توانیم در آرایه زیر ببینیم.*



در عمل برای اضافه کردن اشیاء جدید، نیاز است تا به آدرس خانه‌های آزاد آرایه دسترسی داشته باشیم. موقعیت خانه‌های آزاد یك آرایه را می‌توانیم در همان آرایه ثبت کنیم. برای این کار، آدرس اولین خانه آزاد را به طور جداگانه در اشاره‌گری مانند $free$ نگاه می‌داریم و در سطر اول آرایه، در هر خانه آزاد، آدرس خانه آزاد بعدی را ذخیره می‌کنیم.

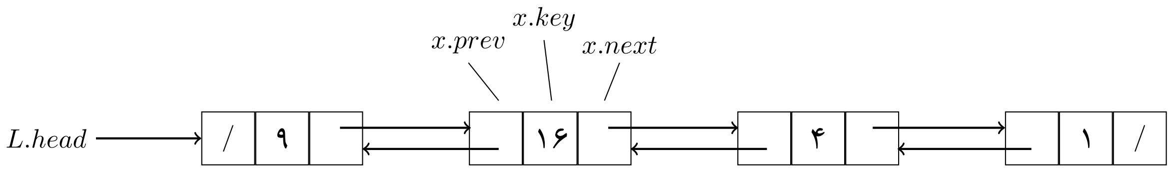
مثال ۳ *نحوه ثبت خانه‌های آزاد در آرایه به دست آمده از مثال قبلی را می‌توانیم در زیر ببینیم.*



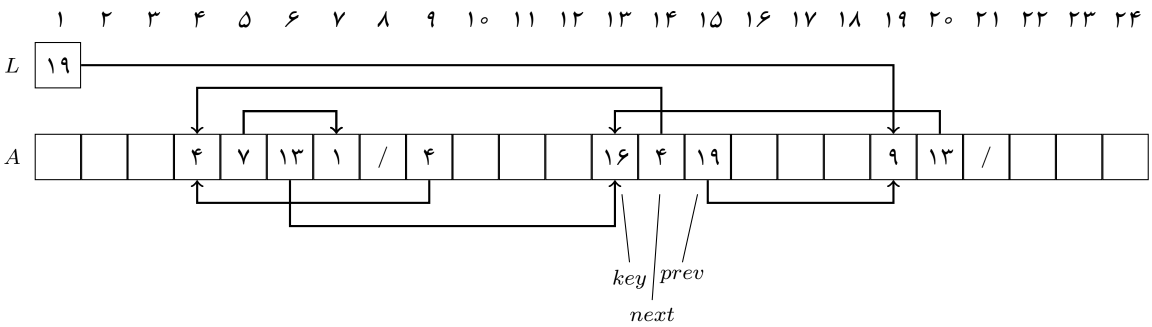
۲.۲ پیاده‌سازی لیست دوسویه با استفاده از آرایه یک‌بعدی

برای پیاده‌سازی لیست در آرایه یک‌بعدی، به ازای هر شیء موجود در لیست، در قسمتی از آرایه، به ترتیب مقادیر key، $next$ و $prev$ را وارد می‌کنیم. این مقادیر به ترتیب نشان‌دهنده مقدار کلید، اشاره‌گر به شیء بعدی و اشاره‌گر به شیء قبلی هستند. همچون مورد قبل، در اینجا نیز آدرس نخستین شیء موجود در لیست را به طور جداگانه در اشاره‌گر L ذخیره می‌کنیم.

مثال ۴ *می‌خواهیم لیست زنجیره‌ای دوسویه موجود در مثال ۱ را در یک آرایه یک‌بعدی پیاده‌سازی نماییم.*



نمونه‌ای از پیاده سازی را می‌توانیم در آرایه مشخص‌شده ببینیم.



۳.۲ جستجوی لیست زنجیره‌ای

روال List-Search اولین عنصر با کلید k را در لیست L با استفاده از یک جستجوی خطی ساده پیدا می‌کند و اشاره‌گر x به این عنصر را برمی‌گرداند. اگر هیچ شیئی با کلید k موجود نباشد، $NILL$ برگردانده می‌شود. این الگوریتم در

بدترین حالت زمان $\Theta(n)$ را صرف می‌کند.

Algorithm 1 Algorithm: LIST-SEARCH

```
function LIST-SEARCH(L, k)
    [assumes L is a list and k is a value]
    x ← L.head
    while x ≠ NILL and x.key ≠ k do
        x ← x.next
    return x
```

۴.۲ درج شیء جدید در لیست زنجیره‌ای

برای درج هر شیء جدید از آدرس اولین خانه آزاد استفاده می‌کنیم که در اشاره‌گر $free$ ذخیره شده است. سپس مقدار موجود در اشاره‌گر $free$ را با آدرس خانه آزاد بعدی به روز می‌کنیم. در الگوریتم ALLOCATE-OBJECT این روند به آسانی قابل تشخیص است. این الگوریتم در زمان $O(۱)$ اجرا می‌شود.

Algorithm 2 Algorithm: ALLOCATE-OBJECT

```
function ALLOCATE-OBJECT(L, x)
    [assumes L is a list and x is the new object]
    if free = NILL then
        "ERROR! Out of space."
    else
        x ← free
        free ← x.next
    return x
```

۵.۲ حذف شیء از لیست زنجیره‌ای

برای حذف شیء x از لیست، می‌توانیم از الگوریتم FREE-OBJECT استفاده کنیم. این الگوریتم نیز در زمان $O(۱)$ اجرا می‌شود.

Algorithm 3 Algorithm: FREE-OBJECT

```
function FREE-OBJECT(L, x)
    [assumes L is a list and x is an object]
    x.next ← free
    free ← x
```

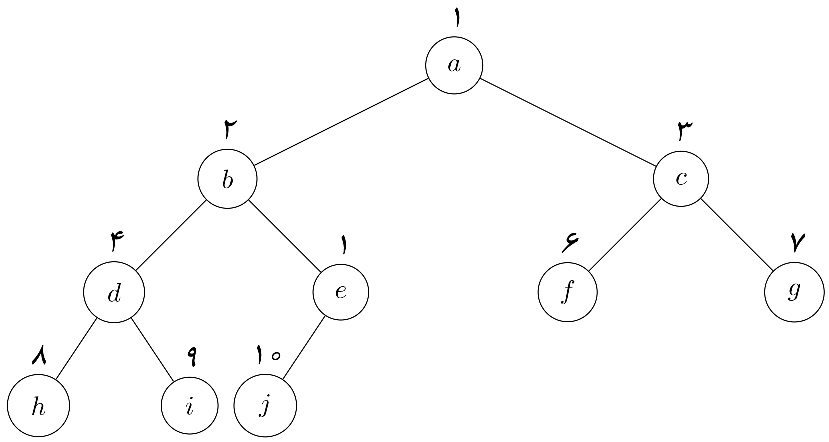
۳ نمایش درخت‌ها با استفاده از آرایه

برای پیاده‌سازی درخت‌ها می‌توانیم از آرایه‌های یک‌بعدی یا چندبعدی استفاده کنیم. عملا شیوه‌ای که برای پیاده‌سازی انتخاب کنیم، به نوع درختی بستگی دارد که می‌خواهیم آن را پیاده‌سازی کنیم. در ادامه با نحوه نمایش انواع درخت آشنا خواهیم شد.

۱.۳ نمایش درخت دودویی کامل

برای نمایش درخت دودویی کامل در یک آرایه، کافی است اندازه درخت را بدانیم و نیز مطمئن شویم که این اندازه از یک مقدار بیشیه تجاوز نخواهد کرد. این مقدار بیشینه (max) در حقیقت طول آرایه مورد استفاده برای پیاده‌سازی است. برای پیاده‌سازی ابتدا رئوس درخت را از بالا به پایین و از چپ به راست، شماره‌گذاری می‌کنیم. اندازه درخت را نیز در اشاره‌گر $size$ نگاه می‌داریم. در نهایت، مقادیر کلیدها در آرایه key ذخیره می‌شوند. آدرسی از این آرایه که برای ذخیره‌سازی مقدار کلید هر رأس درخت استفاده می‌کنیم برابر شماره‌ای است که پیش از این به آن رأس اختصاص داده‌ایم.

مثال ۵ *درخت زیر را در نظر بگیرید. می‌خواهیم این درخت را در یک آرایه یک‌بعدی نمایش بدهیم.*



نتیجه پیروی از الگوریتم مورد اشاره در آرایه زیر نمایش داده شده است.

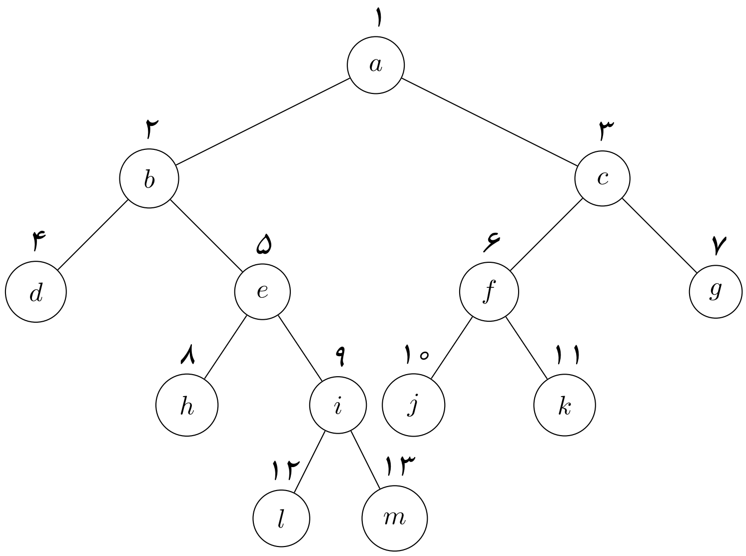
$size = \boxed{۱۰}$

	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	...	max
key	a	b	c	d	e	f	g	h	i	j		

۲.۳ نمایش درخت دودویی غیرکامل

برای نمایش درخت دودویی غیرکامل می‌توانیم از آرایه دوبعدی استفاده کنیم. در اینجا نیز درخت را از بالا به پایین و از چپ به راست شماره‌گذاری کرده و اندازه درخت را در اشاره‌گر $size$ ذخیره می‌کنیم. همچون قبل، مقدار کلید هر رأس را در سطر اول آرایه که با key نمایش داده می‌شود در خانه‌ای قرار می‌دهیم که آدرسی متناظر با شماره آن رأس دارد. در سطر دوم که با p نمایش داده می‌شود نیز آدرس محل ذخیره سازی پدر رأس را وارد می‌کنیم.

مثال ۶ *درخت زیر را در نظر بگیرید. می‌خواهیم این درخت را در یک آرایه چندبعدی نمایش بدهیم.*



اگر با الگوریتم مورد اشاره پیش برویم، نهایتا به آرایه زیر خواهیم رسید.

$size = \boxed{۱۳}$

	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰	۱۱	۱۲	۱۳	...	max
key	a	b	c	d	e	f	g	h	i	j	k	l	m		
p	/	۱	۱	۲	۲	۳	۳	۵	۵	۶	۶	۹	۹		

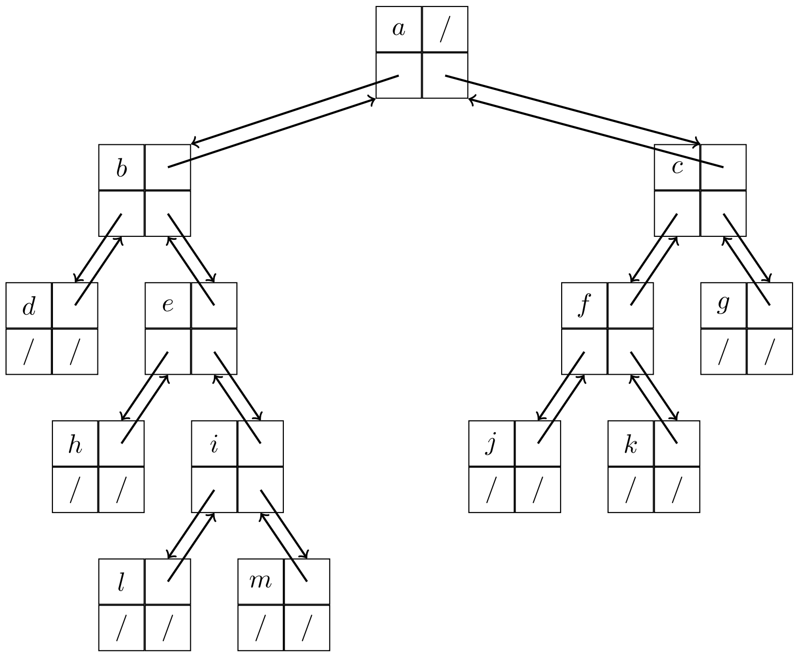
۴ نمایش درخت‌ها با استفاده از اشاره‌گر

برای نمایش درخت در شیوه‌هایی که تاکنون استفاده کردیم، چنان که مشخص است نیاز داشتیم حداکثر اندازه درخت را بدانیم. اما اگر از اشاره‌گرها برای نمایش درخت استفاده کنیم، دیگر این مشکل وجود نخواهد داشت.

۱.۴ نمایش درخت دودویی غیرکامل

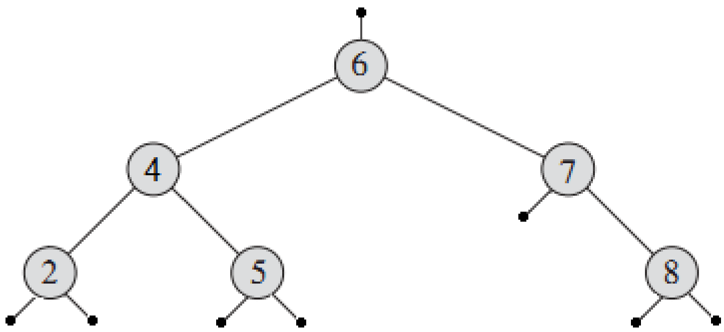
می‌توانیم اطلاعات درخت را در یك اشاره‌گر چهارمؤلفه‌ای ذخیره کنیم. مؤلفه اول اشاره‌گر را به مقادیر کلید رئوس یا key اختصاص می‌دهیم. مؤلفه دوم را به اشاره‌گر به رأس پدر، مؤلفه سوم را به اشاره‌گر به رأس فرزند چپ و مؤلفه چهارم را به اشاره‌گر به فرزند راست اختصاص می‌دهیم. یعنی به ازای هر راس x مقادیر $x.key$، $x.p$، $x.left$ و $x.right$ را ذخیره می‌کنیم.

مثال ۷ *می‌خواهیم درخت مورد استفاده در مثال قبلی را با استفاده از اشاره‌گرها پیاده‌سازی نماییم. اگر به ازای هر رأس درخت، مقادیر مورد نیاز برای ذخیره‌سازی را جایگزین نماییم، به درخت زیر خواهیم رسید.*



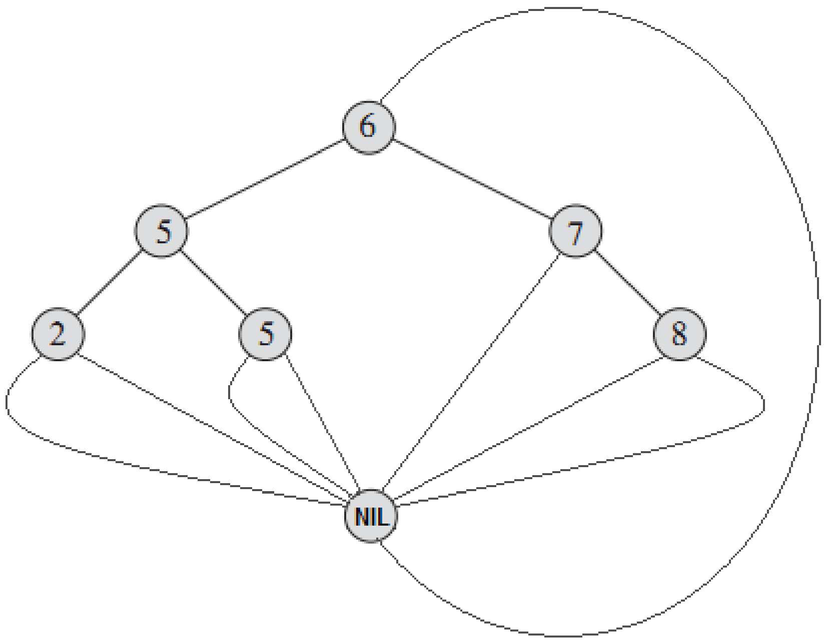
شکل ۱: هر مربع ۴ × ۴ بیانگر یک عنصر لیست زنجیری است که اطلاعات مربوط به یک گره از درخت را نگه می‌دارد. خانه بالا سمت چپ بیانگر مقدار کلید، خانه بالا سمت راست بیانگر مؤلفه اشاره‌گر به پدر، خانه پایین سمت چپ بیانگر مؤلفه اشاره‌گر به فرزند سمت چپ و خانه پایین سمت راست بیانگر مؤلفه اشاره‌گر به فرزند سمت راست است.

همانگونه مشاهده می‌شود، به جای پدر ریشه درخت یعنی راس با کلید و همچنین رئوسی که فرزند چپ یا راست ندارند، از مقدار null استفاده شده است. اشاره‌گر null راهنمای خوبی برای کمک به تشخیص ریشه یا برگ بودن یک گره است. شکل زیر یک درخت را به همراه اشاره‌گرهای null نشان می‌دهد.



شکل ۲: استفاده از مقدار null

همانند پیاده‌سازی لیست برای راحتی، از یک راس NIL به جای مقدار اشاره‌گر null استفاده خواهیم کرد:



شکل ۳: راس NIL

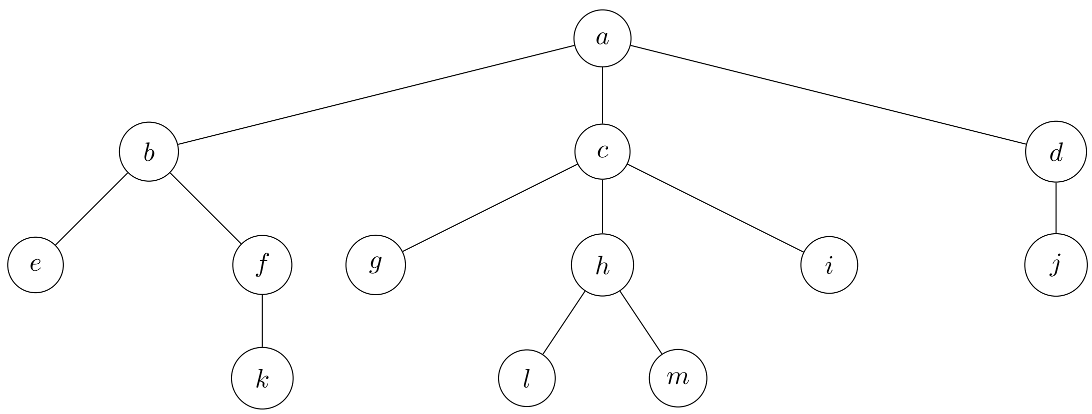
در ادامه برای سادگی راس NIL را رسم نخواهیم کرد.

۲.۴ نمایش درخت غیر دودویی

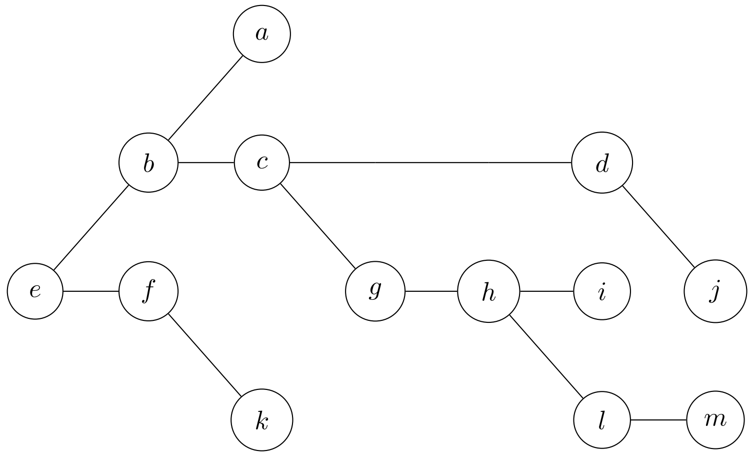
برای نمایش درخت غیردودویی، اگر حداکثر تعداد انشعاب‌های مشتق‌شده از رئوس درخت را بدانیم، می‌توانیم از شیوه‌هایی مشابه حالت‌های قبلی برای پیاده‌سازی استفاده کنیم. اما در حالت کلی، روش بهتر آن است که برای هر درخت غیردودویی، یك درخت دودویی معادل در نظر بگیریم. بدین منظور کافی است برای هر رأس، فرزند سمت

چپ و برادر سمت راست در درخت غیر دودویی را به عنوان فرزند سمت چپ و فرزند سمت راست در درخت دودویی متناظر قرار دهیم. چنان که مشخص است، برای پیاده‌سازی باید به ازای هر راس x مقادیر $x.key$، $x.p$، $x.left-child$ و $x.right-sibling$ را ذخیره کنیم.

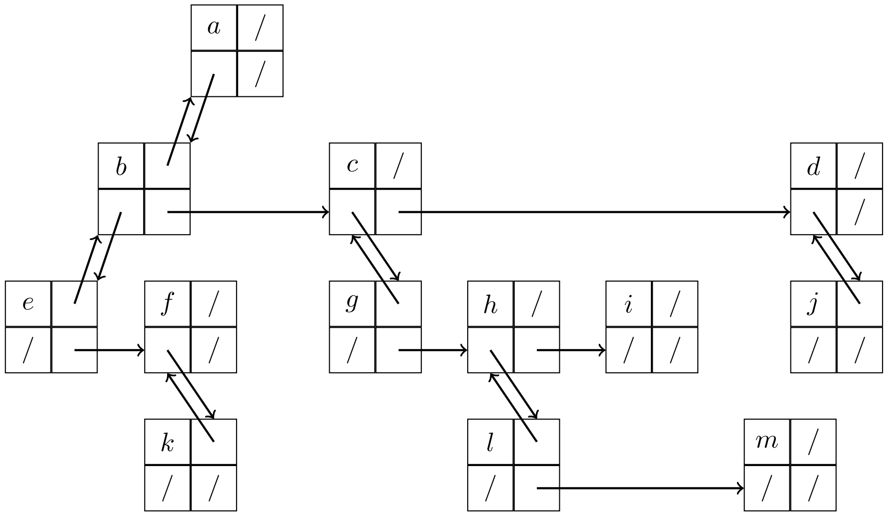
مثال ۸ *درخت زیر را در نظر بگیرید. می‌خواهیم درخت دودویی متناظر با آن را پیدا کنیم.*



با پیروی از الگوریتم مورد اشاره، به درخت دودویی معادل زیر خواهیم رسید.



حال اگر مقادیر مورد نیاز برای پیاده‌سازی را در هر رأس جایگزین کنیم، به درخت زیر خواهیم رسید.



چنان که مشخص است، اطلاعات درخت حاصل را می‌توان به سادگی در یك آرایه چهاربعدی ذخیره کرد.

۵ روش نمایش داده ساختاری گراف

تعریف ۱ *گراف[۶] متشکل از زوج مرتب $G = (V, E)$ است که*

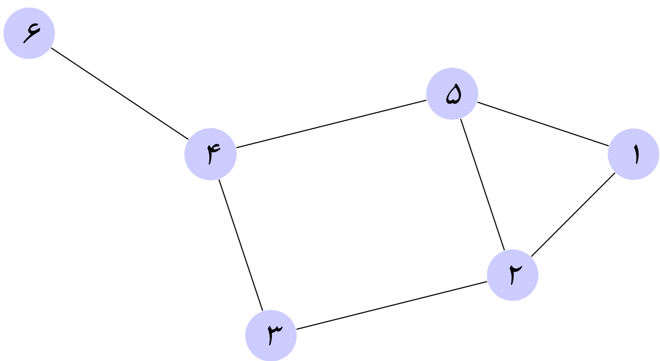
V: مجموعه‌ای غیرتهی و متناهی از رأس‌ها است؛
E: مجموعه‌ی یال‌ها که متشکل از زوج‌های (نامرتب) (u, v) است که $u, v \in V$.

نمادگذاری. تعداد رأس‌های یک گراف را با n و تعداد یال‌های آن را با $|E|$ نمایش می‌دهیم.

مثال ۹ گراف $G = (V, E)$ که

$$
\begin{aligned}
V &= \{۱, ۲, ۳, ۴, ۵, ۶\} \\
E &= \{(۱, ۲), (۱, ۵), (۲, ۵), (۲, ۳), (۴, ۳), (۴, ۵), (۴, ۶)\}
\end{aligned}
$$

دارای $n = ۶$ رأس و $|E| = ۷$ یال است.



[۶] graph

۱.۵ ماتریس مجاورت

تعریف ۲ *ماتریس مجاورت*[۷] *گراف* $G=(V,E)$ *با مجموعه رئوس* $V=\{v_1,\ldots,v_n\}$ *یک ماتریس* $n\times n$ *با درایه‌های صفر و یک به نام* A *است که در آن درایه* $a_{i,j}$ *برابر با ۱ است اگر و فقط اگر* $(v_i,v_j)\in E$.

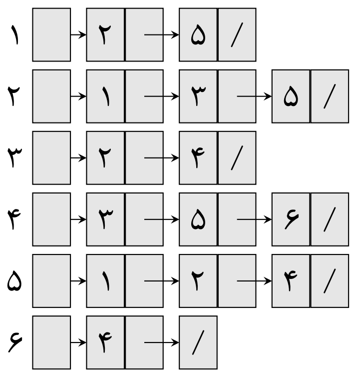
مثال ۱۰ *ماتریس مجاورت گراف مثال ۹ است.*

$$A_{6\times 6}=\begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

۲.۵ لیست مجاورت

لیست مجاورت[۸] فرم دیگر نمایش گراف در کامپیوتر است. این ساختمان داده شامل لیستی از کلیه رئوس گراف است. برای هر رأس یك لیست پیوندی وجود دارد که گره‌های آن رئوس مجاور رأس را دربر می‌گیرند. به عبارت دیگر لیست i حاوی رئوسی است که مجاور رأس v_i است.

مثال. لیست زیر، لیست مجاورت رأس‌های ۱،۲و۳ گراف مثال قبل است.



۳.۵ مقایسه‌ی پیچیدگی‌ها

داده ساختار	حافظه	تعیین $(u,v)\in E$	تعیین رأس‌های مجاور یک رأس	تعیین همه یال‌ها
ماتریس مجاورت	$\Theta(n^2)$	$\Theta(1)$	$\Theta(n)$	$\Theta(n^2)$
لیست مجاورت	$O(n+\|E\|)$	$O(\deg(v))$	$O(\deg(u))$	$O(\|E\|+n)$

[۷]adjacency matrix
[۸]adjacency list