

ساختمان داده‌ها

مؤلف: فرشید شیرافکن

دانشجوی دکترای بیوانفورماتیک دانشگاه تهران

ناشر: سازمان علمی آموزش فرادرس

بزرگترین پلتفرم آموزش آنلاین ایران

وب: www.faradars.org

تقدیم به:

روح پاک پدرم

- فرشید شیر/فکن



سخن ناشر

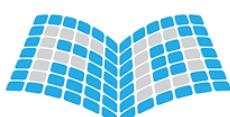
در عین تمام نقدهای وارد شده به کنکور، هنوز راه حلی عملی که در جمیع جوانب، بهتر از سبک کوتاه و چند گزینه‌ای سوالات باشد؛ ارائه نشده است. همین موضوع، کنکور را به ویژه کنکور کارشناسی ارشد به عنوان یک آزمون متمرکز و سراسری، از اهمیت دو چندانی برخوردار می‌کند.

یکی از آسیب‌های همراه با این آزمون سراسری این است که فضای رقابتی آن با ایجاد موسسات گوناگون، به سرعت از فضای یک رقابت علمی تبدیل به فضای رقابت اقتصادی می‌شود؛ به گونه‌ای که هزینه سرسام آور کلاس‌ها، دوره‌ها و منابع مرتبط با آزمون، از عهده بسیاری از دانشجویان خارج می‌شود. دانشجویانی که در عین استعداد تحصیلی بالا، در میدان رقابت مالی تحمیلی، در عین تمام شایستگی‌های خود، قدرت ادامه مسیر را از دست می‌دهند یا به نتیجه‌ای که در فضای مساوی مالی برای همه باید به آن می‌رسیدند، دست نمی‌یابند.

یکی از اهداف و آرمان‌های فرادرس به عنوان بزرگترین پروژه آموزش دانشگاهی اجرا شده بر پسترو وب کشور، ایجاد دسترسی همگانی و یکسان به آموزش و دانش؛ مستقل از جغرافیا، زمان و سطح مالی دانشجویان بوده است. سیاست کاری فرادرس در راستای این آرمان، انتشار آموزش‌های ویدئویی تخصصی و دانشگاهی رایگان و یا بسیار کم هزینه، با تدریس محرب‌ترین اساتید داخل و خارج کشور بوده است.

ما با انتشار رایگان این کتاب (به همراه نزدیک به ده کتاب رایگان دیگر) یکی از گام‌های دیگر خود را در راستای آرمان فرادرس برداشتیم. کتاب حاضر که حاصل نزدیک به دو دهه تدریس و پژوهش و تألیف مؤلف و مدرس فرادرس می‌باشد؛ در عین هزینه‌های بالای تألیف و آماده‌سازی، به جای انتشار و فروش؛ با تأمین مالی و سرمایه‌گذاری فرادرس به عنوان ناشر، به صورت کاملاً رایگان منتشر می‌شود. ما در گام‌های بعدی نیز تلاش خواهیم کرد تا هر جا بتوانیم، حتی شده یک کتاب مرجع دیگر و بیشتر را با پرداخت هزینه، آزادسازی کرده و به صورت رایگان منتشر کنیم.

مؤلفین و ناشرینی که تمایل به واگذاری حق انتشار کتاب خود به فرادرس را دارند، می‌توانند با ایمیل ebooks@faradars.org مکاتبه نمایند. ما این کتاب‌ها را با پرداخت هزینه تألیف به مدرس و ناشر، به صورت رایگان منتشر خواهیم کرد تا همه دانشجویان مستقل از سطح مالی، به منابع مفید آزمون دسترسی داشته باشند. همچنین اگر ایده و نظری در خصوص کتاب‌های رایگان فرادرس داشته باشید، خوشحال می‌شویم که آن را با ایمیل ebooks@faradars.org مطرح نمایید.



فرادرس

فرادرس از یک کلاس درس
www.faradars.org

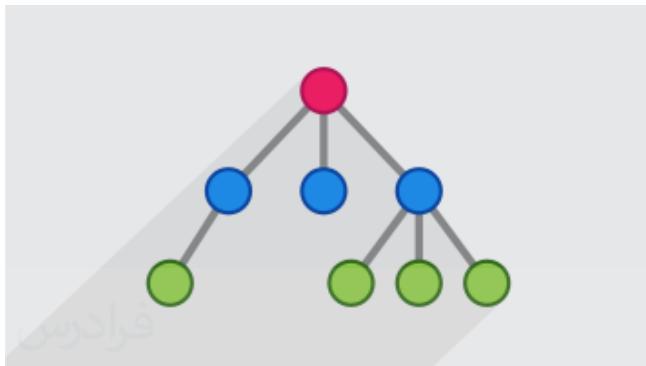
سازمان علمی آموزش فرادرس

بزرگترین پلتفرم آموزش آنلاین ایران

وب: www.faradars.org

منبع مطالعاتی تکمیلی مرتبه با این کتاب

آموزش ویدئویی ساختمان داده‌ها



ساختمان داده‌ها، یکی از دروس مهم و پایه‌ای دانشگاهی است که پیش نیاز دروس مختلف رشته کامپیوتر است و به عنوان مبحثی که نکات فراوانی دارد در کنکور کارشناسی ارشد کامپیوتر و دکترای هوش مصنوعی و نرم افزار از دروس با ضرایب بالا می‌باشد.

آموزش ساختمان داده‌ها، توسط مهندس فرشید شیرافکن، یکی از بهترین مدرسین مسلط به مباحث ساختمان داده‌ها، ارائه شده است.

مدرس: مهندس فرشید شیرافکن

مدت زمان: ۱۰ ساعت

faradars.org/fvds9402

جهت مشاهده آموزش ویدئویی این آموزش [کلیک](#) کنید.

درباره مدرس



مهند فرشيد شيرافكن كارشناس ارشد مهندسي کامپيوتر گرایش نرم افزا است و در حال حاضر دانشجوی دکتراي بیوانفورماتیک دانشگاه تهران هستند. ایشان از مدرسین نمونه در زمینه ارائه و آموزش دروس دانشگاهی انتخاب شده‌اند. ایشان مشاور کنکور هستند و بیش از ۳۰ کتاب در زمینه کنکور رشته کامپیوتر تألیف نموده‌اند. ایشان در حال حاضر به عنوان یکی از برترین مدرسین فرادرس از جهت کمیت و کیفیت دروس ارائه شده، نزدیک به ۲۰ عنوان درسی را در قالب آموزش ویدئویی از طریق فرادرس منتشر کرده‌اند. این مجموعه دروس تا کنون مورد استفاده دهها هزار دانشجوی سراسر کشور قرار گرفته‌اند.

مشاهده همه آموزش‌های تدریسی و تألیفی توسط مؤلف کتاب [کلیک](#) کنید.

Email: fshirafkan51@gmail.com

شماره همراه مدرس: ۰۹۱۲۱۹۷۲۰۲۸

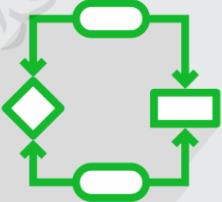
کتب رایگان دیگر از این مجموعه آموزشی

۱. آموزش نظریه زبان ها و ماشین – [کلیک کنید \(+\)](#)
۲. آموزش برنامه‌نویسی سی پلاس پلاس – [کلیک کنید \(+\)](#)
۳. آموزش پایگاه داده ها – [کلیک کنید \(+\)](#)
۴. آموزش شیء‌گرایی در سی پلاس پلاس – [کلیک کنید \(+\)](#)
۵. آموزش سیستم عامل – [کلیک کنید \(+\)](#)
۶. آموزش ذخیره و بازیابی اطلاعات – [کلیک کنید \(+\)](#)

برای دانلود رایگان این مجموعه کتب، به لینک زیر مراجعه کنید:

<http://faradars.org/computer-engineering-exam>

دسته بندی موضوعی آموزش‌های فرادرس، در ادامه آمده است:

 <p>مهندسی برق الکترونیک و روباتیک</p> <p>مهندسي برق الکترونيک و روباتيك - کلیک (+)</p>	 <p>هوش مصنوعی</p> <p>هوش مصنوعی - کلیک (+)</p>	 <p>آموزش‌های دانشگاهی و تخصصی</p> <p>آموزش‌های دانشگاهی و تخصصی - کلیک (+)</p>	 <p>برنامه‌نویسی</p> <p>برنامه‌نویسی - کلیک (+)</p>
 <p>مهارت‌های دانشگاهی</p> <p>مهارت های دانشگاهی - کلیک (+)</p>	 <p>مباحث مشترک</p> <p>مباحث مشترک - کلیک (+)</p>	 <p>دروس دانشگاهی</p> <p>دروس دانشگاهی - کلیک (+)</p>	 <p>مهندسي نرم‌افزار</p> <p>مهندسي نرم‌افزار - کلیک (+)</p>
 <p>آموزش‌های عمومی</p> <p>آموزش‌های عمومي - کلیک (+)</p>	 <p>طراحی و توسعه وب</p> <p>طراحی و توسعه وب - کلیک (+)</p>	 <p>نرم‌افزارهای عمومی</p> <p>نرم‌افزارهای عمومي - کلیک (+)</p>	 <p>نرم‌افزارهای تخصصی</p> <p>نرم‌افزارهای تخصصي - کلیک (+)</p>

فهرست مطالب

فصل ۱ : مرتبه اجرایی

O نشان گذاری

مرتبه اجرایی حلقه ها

خواص سیگما

نمادهای Ω و θ

فصل ۲ : زیربرنامه های بازگشتی و مرتبه زمانی آنها

زیر برنامه بازگشتی

زیر برنامه های بازگشتی معروف

مرتبه اجرایی توابع بازگشتی

قضیه اصلی برای حل روابط بازگشتی

فصل ۳ : آرایه

آرایه

نحوه ذخیره عناصر آرایه در حافظه

جستجو در آرایه (خطی - دودویی - سه تایی)

اضافه و حذف در آرایه

پیدا کردن عنصر کمینه در آرایه

ماتریس

انواع ماتریس

ماتریس اسپارس

ماتریس مثلثی

ماتریس ۳ قطری

فصل ۴ : صفحه و پشته

صف

درج و حذف در صفحه ساده

صف حلقوی

درج و حذف در صفحه حلقوی

pop و push

کاربردهای پشته

ارزشیابی عبارات (تبدیل infix و prefix و postfix به یکدیگر)

الگوریتم تبدیل عبارت infix به postfix توسط پشته
الگوریتم محاسبه یک عبارت به فرم Postfix توسط پشته
کاربرد پشته در زیر برنامه های بازگشتی

فصل ۵ : لیست پیوندی

لیست پیوندی یک طرفه
الگوریتم های کار بر روی لیست پیوندی یک طرفه (اضافه و حذف گره - اتصال - وارون و....)
پیاده سازی پشته با لیست پیوندی
پیاده سازی صف با لیست پیوندی
لیست پیوندی دو طرفه
لیست پیوندی حلقوی

فصل ۶ : درخت

تعاریف اولیه
درخت دودویی(کامل، پر)
درخت k تایی
درخت دودویی کامل شماره گذاری شده
روشهای ذخیره درخت دودویی
تعداد درخت های دودویی
الگوریتم های کار بر روی درخت دودویی
پیمایش درخت دودویی
درخت نخی دودویی
درخت عمومی
جنگل
درخت دودویی گسترش یافته (2-Tree)

فصل ۷ : درخت های جستجو (BST , AVL , 2-3 , Btree,...)

درخت جستجوی دودویی(BST)
عملیات بر روی یک BST
درخت AVL
درخت قرمز-سیاه
درخت آماری
درخت ۳-۲
درخت بی (B-Tree)

فصل ۸ : درخت های هیپ

(Heap) هرم

هیپ d تایی

صف اولویت

Deap

Treap

هیپ دو جمله‌ای

هیپ فیبوناچی

فصل ۹ : گراف

گراف

انواع گراف

نمایش گراف

پیمایش گراف

درخت پوشان

الگوریتم کراسکال

الگوریتم پریم

الگوریتم سولین

فصل ۱۰ : مرتب سازی

مرتب سازی

الگوریتم مرتب سازی حبابی

الگوریتم مرتب سازی انتخابی

الگوریتم مرتب سازی درجی

الگوریتم مرتب سازی ادغامی

الگوریتم مرتب سازی سریع

الگوریتم مرتب سازی هرمی

الگوریتم مرتب سازی درختی

الگوریتم مرتب سازی شل

الگوریتم های پایدار

درخت تصمیم گیری

الگوریتم مرتب سازی مبنا

فصل ۱۱ : درهم سازی

جدول آدرس دهی مستقیم

جدول های درهم سازی

برخورد

توابع درهم سازی

روش زنجیره ای برای حل برخورد

آدرس دهی باز

درهم سازی پویا



فصل ۱

مرتبه اجرایی

مجموعه محدودی از دستورالعملها که با دنبال کردن آنها هدف خاصی برآورده می‌شود را الگوریتم می‌نامند. در هر الگوریتم مواردی همچون ورودی، خروجی، قطعیت، محدودیت و کارایی قبل بررسی می‌باشند. به علت وجود بیش از یک راه حل برای یک مسئله، الگوریتم‌های متفاوتی مورد توجه قرار می‌گیرند که برای انتخاب یکی از آنها فاکتورهایی چون ملزومات برنامه‌نویسی، ملزومات عملیاتی (تأثیر زمان اجرا) و ملزومات حافظه‌ای را باید در نظر گرفت. به علت مشکل بودن تجزیه و تحلیل دقیق ملزومات برنامه‌نویسی، تئوری پیچیدگی روی ملزومات حافظه‌ای و عملیاتی متمرکز می‌شود. ملزومات عملیاتی حساس‌تر از ملزومات حافظه‌ای بوده و معمولاً رکن کار می‌باشند. پس می‌توان عامل فضا را فدای زمان کرد، یعنی الگوریتمی پر حجم‌تر اما سریع نوشت.

O نشان‌گذاری

روشی است که اندازه‌گیری کمیتهای ملزومات عملیاتی را به گونه‌ای عمومی ممکن می‌سازد. پیچیدگی یک الگوریتم، تابعی است که مدت زمان اجرای استفاده شده توسط الگوریتم را بر حسب تعداد داده‌های ورودی n اندازه می‌گیرد. به طور نمونه $O(1)$ معرف زمان اجرای ثابت، $O(n)$ معرف زمان اجرای خطی و $O(n^2)$ معرف زمان اجرای مربعی است.

رابطه زیر برقرار است:

$$O(1) < O(\log n) < O(n) < O(n \cdot \log n) < O(n^2) < O(2^n) < O(n!) < O(n^n)$$

این رابطه به طور نمونه مشخص می‌کند که الگوریتمی با مرتبه اجرایی $O(n \cdot \log n)$ سریع‌تر از الگوریتمی با مرتبه اجرایی $O(n^2)$ است.

مرتبه اجرایی تابع $f(n)$ برابر $O(n^m)$ می‌باشد:

$$f(n) = a_m n^m + a_{m-1} n^{m-1} + \dots + a_2 n^2 + a_1 n + a_0$$

مثال

مرتبه اجرایی تابع $f(n) = 5n^2 - 3n + 4$ کدام است؟

حل: مرتبه اجرایی تابع f برابر $O(n^2)$ می‌باشد، چون موثرترین عبارت n^2 می‌باشد یعنی از همه جملات سریع‌تر رشد می‌کند.

تذکر: تعداد تکرار حلقه زیر برابر $\left\lceil \frac{b-a+1}{k} \right\rceil$ می‌باشد:

for (i=a ; i<= b; i=i+k)

....

مثال

مرتبه اجرایی دستور $x=x+1$ برابر است با $O(n)$:

```
for (i=l ; i<=n ; i=i+1)
    x=x+1;
```

**مثال**

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
for(i=2 ; i<= n-1; i=i+2 )
    x=x+1;
```

حل: حلقه $\left\lceil \frac{n-2}{2} \right\rceil$ بار اجرا می شود، بنابراین مرتبه $O(n)$ می باشد.

**مثال**

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
for ( i=2 ; i<= n; i=i+4 )
    x=x+1;
```

حل: حلقه $\left\lceil \frac{n-1}{4} \right\rceil$ بار اجرا می شود، بنابراین مرتبه $O(n)$ می باشد.

مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
for( i=0 ; i< n2 ; i=i+2)
    x=x+1;
```

حل: حلقه $\frac{n^2}{2}$ بار اجرا می شود، بنابراین مرتبه $O(n^2)$ می باشد.

مثال

مرتبه اجرایی دستور $x=x+1$ را بدست آورید؟

```
i= n;
while (i >=1){
    x = x+1;
    i = i % 2;
}
```

حل:

حلقه به ازای هر n بزرگتر یا مساوی ۱، یک یا دو مرتبه اجرا می شود. چون باقیمانده تقسیم صحیح هر عدد بر دو، یا صفر است و یا یک. بنابراین مرتبه اجرایی $O(1)$ می باشد.

تذکر: اگر به جای عدد ۲ در دستور $i \% 2$ ، از هر عدد دیگری نیز استفاده شود، مرتبه اجرایی باز هم $O(1)$ خواهد بود. چون تعداد اجرای حلقه به پارامتر n وابسته نمی باشد.



مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
for ( i=1 ; i<=n ; i=i*2 )
    x=x+1;
```

حل:

چون i مقادیر ..., 1, 2, 4, 16,... را می‌گیرد. بنابراین مرتبه اجرایی برابر $O(\log_2^n)$ یا $O(\lg n)$ می‌باشد. تذکر: می‌توان به جای \log_2^n از واژه خلاصه $\lg n$ و یا $\lg n$ استفاده کرد.

مثال

مرتبه اجرایی دستور $x=x+1$ برابر است را مشخص کنید.

```
for( i=n ; i>1; i=i/2 )
    x=x+1;
```

حل: شمارنده i در هر بار اجرا، نصف می‌شود، بنابراین مرتبه $O(\lg n)$ است. این حلقه را با دستور while نیز می‌توان نوشت:

```
i = n;
while( i>1) {
    x = x+1;
    i = i /2;
}
```

مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
for( i=n ; i>1; i=i/3 )
    x=x+1;
```

حل: شمارنده i در هر بار اجرا، تقسیم بر 3 می‌شود، بنابراین مرتبه \log_3^n است.

تذکر: اگر به جای $i > 1$ از $i > 0$ در حلقه بالا استفاده می‌شد، حلقه هیچگاه تمام نمی‌شود، بنابراین زمان اجرا، بی‌نهایت می‌شود.

مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
for (i=1 ; i<=n ; i++)
    x=x+1;
for (j=1 ; j<=m ; j++)
    x=x+1;
```

حل:

حلقه اول n مرتبه و حلقه دوم m مرتبه اجرا می‌شود. مرتبه برابر $O(n + m)$ می‌باشد. تذکر: این حلقه‌ها تودرتو نمی‌باشند.

مرتبه اجرایی حلقه های تودر تو

اگر حلقه ها تو در تو باشند، مرتبه اجرایی از ضرب مرتبه اجرایی حلقه ها بدست می آید.

مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
for( i=l ; i<=n ; i++)
    for( j=l ; j<=n ; j++)
        x=x+1;
```

حل: مرتبه اجرایی $O(n^2)$ می باشد.

مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید. n زوج است

```
for ( i=1 ; i<=n ; i++)
{
    for ( j=1 ; j<=n ; j++)
    {
        x=x+1;
    }
    n=n-1;
}
```

حل:

به ازای $i=1$ ، دستور $x=x+1$ ، n بار اجرا شده و سپس یک واحد از n کم می شود.

به ازای $i=2$ ، دستور $x=x+1$ ، $n-1$ بار اجرا شده و از n مجدداً یک واحد کم می شود.

به ازای $i=\frac{n}{2}+1$ ، دستور $x=x+1$ ، $n-\frac{n}{2}+1$ بار اجرا شده و از n یک واحد کم می شود.

به ازای $i=\frac{n}{2}+1$ ، شرط برقرار نمی باشد و اجرای حلقه پایان می یابد.

بنابراین تعداد اجرای $x=x+1$ برابر است با:

$$n + (n - 1) + (n - 2) + \dots + \left(n - \frac{n}{2} + 1\right) = \frac{3n^2 + 2n}{8}$$

در نتیجه مرتبه زمانی برابر $O(n^2)$ می باشد.

مثال

مرتبه اجرایی دستور $x=x+1$ کدام است؟

```
for (i=1; i<= n ; i++)
    for (j=1; j<=7 i ; j++)
        x=x+1;
```

حل:

تعداد دفعات تکرار دستور $x=x+1$ برابر است با: $7 \times \frac{n(n+1)}{2}$. بنابراین مرتبه $O(n^2)$ است.

مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
for (i=l ; i<=n ; i++){
    k=n;
    while (k >1)
    {
        k=k / 2;
        x=x+1;
    }
}
```

حل: حلقه for، n مرتبه و حلقه while، log_n مرتبه تکرار می‌شود و چون حلقه‌ها متداخل هستند، مرتبه اجرایی $O(n \log n)$ می‌باشد.



مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
for( i=l ; i<=n; i++)
{
    for ( j=l ; j<=m; j++)
        x =x+1;
    for ( k=l ; i<=p; k++)
        x =x+1;
}
```

حل: مرتبه اجرایی for با شمارنده i برابر n، for با شمارنده j برابر m و for با شمارنده k برابر p می‌باشد. چون دو حلقه داخلی پشت سر هم می‌باشند (نه متداخل)، مرتبه انها با هم جمع شده و در $n(m+p)$ ضرب می‌شوند، چون داخل حلقه با مرتبه n می‌باشند. بنابراین مرتبه اجرایی $O(n(m+p))$ است.



مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
for (i=l ; i<=n; i++){
    for (j=l ; j<=m ; j++)
        x =x+1;
    j =1;
    while( j<=n )    {
        x= x+1;
        j=3*j;
    }
}
```

حل: حلقه for اول از $O(n)$ و حلقه for دوم از $O(m)$ و حلقه while از مرتبه $(O(\log_3^n))$ است.
مرتبه دو حلقه داخلی چون تو در تو نیستند، با هم جمع شده و در مرتبه حلقه for خارجی ضرب می‌شوند:

$$O(n(m + \log_3^n))$$

مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید.

```
i = n;
while(i>1){
    i =i/2;
    j =n;
    while (j>1) {
        j=j/3;
        x=x+1;
    }
}
```

حل: حلقه داخلی با تقسیم مقدار شمارنده بر 3، به صورت لگاریتمی با $(O(\log_3^n))$ اجرا شده و حلقه بیرونی با تقسیم شمارنده بر 2 به صورت $(O(\log_2^n))$ اجرا می‌شود و چون دو حلقه متداخل هستند، جواب برابر حاصل ضرب این دو مقدار یعنی $O(\log_3^n \times \log_2^n)$ می‌باشد.

مثال

مرتبه اجرایی دستور $x=x+1$ را مشخص کنید. ($n = 2^k$)

```
for ( i = n ; i>0 ; i=i/2)
    for ( j = i ; j <= n ; j=j+1)
        x=x+1;
```

i	2^k	2^{k-1}	2^{k-2}	2^1	2^0
تعداد تکرار حلقه داخلی	1	$2^k - 2^{k-1}$	$2^k - 2^{k-2}$	$2^k - 2^1$	$2^k - 2^0$

بنابراین داریم:

$$\begin{aligned} & 1 + 2^k - 2^{k-1} + 2^k - 2^{k-2} + \dots + 2^k - 2^1 + 2^k - 2^0 \\ & = 1 + k \cdot 2^k - (2^{k-1} + 2^{k-2} + \dots + 2^1 + 2^0) = 1 + k \cdot 2^k - (2^k - 1) = k \cdot 2^k - 2^k + 2 \\ & = n \lg n - \lg n + 2 = O(n \lg n) \end{aligned}$$

تذکر: اگر شمارنده j به صورت $j=j+a$ تغییر کند داریم:

خواص سیگما

در بعضی از مسائل نیاز است که خواص را بدانید. این خواص در زیر آورده شده است:

$$\sum_{i=a}^b c = (b - a + 1) \times c$$

$$\sum_{i=1}^n n = n^2$$

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{n-1} i = \frac{n(n-1)}{2}$$

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^n i^3 = \left[\frac{n(n+1)}{2} \right]^2$$

$$\sum_{i=0}^n 2^i = 2^{n+1} - 1$$

$$\sum_{i=1}^n i 2^i = (n-1)2^{n+1} + 2$$

$$\sum_{i=0}^n k^i = \frac{k^{n+1} - 1}{k - 1}$$

$$\sum_{i=1}^n i^k \approx \frac{n^{k+1}}{k+1}$$

$$\sum_{i=1}^{n-1} k^i = \frac{k^n - 1}{k - 1} - 1$$

$$\sum_{i=1}^{n-1} i k^{i-1} = \frac{n k^{n-1}}{k - 1} + \frac{1 - k^n}{(k - 1)^2}$$

$$\sum_{i=0}^n i(i+1) = \frac{n(n+1)(n+2)}{3}$$

$$\sum_{i=0}^{n-1} \frac{1}{2^i} = 2 - \frac{1}{2^{n-1}}$$

$$\sum_{i=1}^{n-1} (i+1)(n-i) = \frac{n(n-1)(n+4)}{6}$$

$$\sum_{i=1}^{n-1} i(n-i) = \frac{n(n-1)(n+1)}{6}$$

مثال

مرتبه اجرایی دستور $x=x+1$ کدام است؟

```
for ( t=1 ; t <= n-1 ; t++)
{
    for ( i=1 ; i<= n-t ; i++)
        { j= i+t;
            for ( k= i ; k<= j-1; k++)
                x=x+1}
}
```

حل:

$$\begin{aligned} \sum_{t=1}^{n-1} \sum_{i=1}^{n-t} \sum_{k=i}^{j-1} 1 &= \sum_{t=1}^{n-1} \sum_{i=1}^{n-t} (j-1-i+1) = \sum_{t=1}^{n-1} \sum_{i=1}^{n-t} (i+t-1-i+1) \\ &= \sum_{t=1}^{n-1} \sum_{i=1}^{n-t} t = \sum_{t=1}^{n-1} t(n-t) = \frac{n(n-1)(n+1)}{6} \end{aligned}$$

بنابراین مرتبه $O(n^3)$ است.

بررسی چند مثال از کاربرد سیگما

for (i=1; i<= n ; i++) for (j=1; j<= n ; j++){ x=x+1; x=x+1; }	$\sum_{i=1}^n \sum_{j=1}^n 2 = 2 \sum_{i=1}^n \sum_{j=1}^n 1 = 2 \sum_{i=1}^n n = 2n \sum_{i=1}^n 1 = 2n \times n = 2n^2$
for (i=1; i<= n ; i++) for (j=1; j<= i ; j++) x=x+1;	$\sum_{i=1}^n \sum_{j=1}^i 1 = \sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n(n+1)}{2}$
for (k=0; k<= n-1 ; k++) for (i=1; i<= n-k ; i++) x=x+1;	$\sum_{k=0}^{n-1} \sum_{i=1}^{n-k} 1 = \sum_{k=0}^{n-1} (n-k) = \sum_{k=0}^{n-1} n - \sum_{k=0}^{n-1} k = n^2 - \frac{n(n-1)}{2} = \frac{n(n+1)}{2}$
for (i=1; i<= n ³ ; i++) for (j=1; j<= i ; j++) x=x+1;	$\sum_{i=1}^{n^3} \sum_{j=1}^i 1 = \sum_{i=1}^{n^3} i = \frac{n^3(n^3+1)}{2} \in O(n^6)$
for (i=1; i<= n ² ; i++) for (j=1; j<= log n ; j++) x=x+1;	$\sum_{i=1}^{n^2} \sum_{j=1}^{\log n} 1 = \sum_{i=1}^{n^2} \log n = \log n \sum_{i=1}^{n^2} 1 = n^2 \times \log n$

 تعداد اجرای دستور $x=x+1$ در حلقه وابسته برابر n^t می باشد.

مثال

تعداد اجرای دستور $x=x+1$ کدام است؟

```
for (i=1; i<=3 ; i++)
    for (j=1; j<= i ; j++)
        for (k=1; k<= j ; k++)
            for (m=1; m<=k ; m++)
                x=x+1;
```

حل: دستور در داخل چهار حلقه وابسته قرار دارد، بنابراین تعداد اجرای آن برابر است با:

$$\binom{3+4-1}{4} = \binom{6}{4} = \frac{6!}{2! \times 4!} = 15$$

 تذکر بسیار مهم: در موقعی که تغییر شمارنده به اندازه یک واحد نباشد، در صورت استفاده از \sum باید یک متغیر جدید تعریف کرد.

مثال

مرتبه اجرایی شبه دستور زیر کدام است؟

```
for (i=1; i<=n ; i=i*2)
    for (j=1; j<= i ; j++)
        x=x+1;
```

حل: چون در حلقه اول متغیر i هر بار دو برابر می شود، قرار می دهیم: $i = 2^t$. چون $0 \leq t \leq \log n$ ، بنابراین $1 \leq i \leq n$

در حلقه دوم، متغیر j از ۱ تا i یعنی 2^t تغییر می کند:

$$\sum_{t=0}^{\log n} \sum_{j=1}^{2^t} 1 = \sum_{t=0}^{\log n} 2^t = 2^{\log n + 1} - 1 = 2^{\log n} \times 2 - 1 = 2n - 1$$

بنابراین از مرتبه $O(n)$ می باشد.

مثال

مرتبه اجرایی دستور $x=x+1$ کدام است؟

```
for ( i=1 ; i <= n ; i++)
    for ( j=1 ; j<= n ; j=2*j)
        for ( k= 1 ; k<= j ; k++)
            x=x+1;
```

حل:

حلقه اول n مرتبه تکرار می شود. در حلقه دوم چون متغیر j هر بار دو برابر می شود، به جای j از 2^t استفاده می کنیم که چون $0 \leq t \leq \log n$ داریم: $1 \leq j \leq n$.

در حلقه سوم متغیر k از ۱ تا j یعنی 2^t تغییر می‌کند:

$$\sum_{i=1}^n \sum_{t=0}^{\log n} \sum_{k=1}^{2^t} 1 = \sum_{i=1}^n \sum_{t=0}^{\log n} 2^t = \sum_{i=1}^n (2^{\log n+1} - 1) = (2^{\log n+1} - 1) \sum_{i=1}^n 1 = (2n - 1)n$$

بنابراین از مرتبه $O(n^2)$ می‌باشد.



θ, Ω نمادهای

جهت نمایش مرتبه اجرایی می‌توان از نمادهای اومگای بزرگ (Ω) و تا (θ) نیز استفاده کرد. این نمادها در کتاب طراحی الگوریتم اینجانب به طور مفصل بررسی شده‌اند. در اینجا کافی است بدانید که برای بررسی صحت یک عبارت می‌توان به جای Ω از \geq ، به جای θ از $=$ و به جای O از \leq استفاده کرد.

مثال

عبارت‌های زیر برقرار است:

$$5n^2 \in \theta(n^2) \quad 5n^2 \in \Omega(n^2) \quad 5n^2 \in O(n^2)$$

$$n \in O(n^2) \quad n^2 + 10n \in \Omega(n^2) \quad n^3 \in \Omega(n^2)$$

$$n! = o(n^n) \quad \lg(n!) = \theta(n \lg n) \quad n^2 \sin n \in \Omega(n)$$

$$5n + 31g \quad n + 10n \lg n + 7n^2 \in \theta(n^2) \quad \log_2^n \in \theta(\log_{10}^n)$$



مثال

رشد مجانبی تابع $\sum_{i=1}^n i^3$ و $\sum_{i=1}^n i^2$ را به دست آورید.

حل:

$$\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6} = \theta(n^3) \quad \sum_{i=1}^n i^3 = \left(\frac{n(n+1)}{2} \right)^2 = \theta(n^4) \quad \text{و}$$



رشد مجانبی تابع $\sum_{i=1}^n i^k$ برابر $\theta(n^{k+1})$ می‌باشد.

مثال

رشد مجانبی تابع $\sum_{i=0}^n \sum_{j=0}^n i^2 j^3$

$$\sum_{i=0}^n \sum_{j=0}^n i^2 j^3 = \sum_{i=0}^n i^2 \sum_{j=0}^n j^3 = \theta(n^3) \theta(n^4) = \theta(n^7)$$



مثال

رشد مجانبی تابع $\sum_{i=0}^n 3^i \times i^8$ را به دست آورید.

حل:

$$\sum_{i=0}^n 3^i \times i^8 \leq \sum_{i=0}^n 3^i \times n^8 = n^8 \sum_{i=0}^n 3^i = n^8 \times \frac{3^{n+1} - 1}{3 - 1} \leq n^8 3^n \Rightarrow \sum_{i=0}^n 3^i \times i^8 = O(3^n n^8) \blacksquare$$

مثال

رشد مجانبی تابع $\sum_{i=0}^n 2^i$

$$\sum_{i=0}^n 2^i = 2^0 + 2^1 + \dots + 2^n = \frac{2^{n+1} - 1}{2 - 1} = 2^{n+1} - 1 = \theta(2^n)$$

■

رشد مجانبی تابع $\sum_{i=0}^n k^i$ برابر $\theta(k^n)$ می‌باشد.

مثال

رشد مجانبی تابع $\sum_{i=0}^{\lg n} \lg(\frac{n}{2^i})$ را به دست آورید.

حل:

$$\sum_{i=0}^{\lg n} \lg n - \sum_{i=0}^{\lg n} \lg 2^i = \lg n \sum_{i=0}^{\lg n} 1 - \sum_{i=0}^{\lg n} i = \lg n (\lg n + 1) - \frac{\lg n (\lg n + 1)}{2} = \frac{\lg n (\lg n + 1)}{2} = \theta(\lg^2 n) \blacksquare$$

کنکور ارشد

۱- پیچیدگی زمانی الگوریتم خاصی به صورت چند جمله‌ای زیر معین شده است. در این صورت زمان اجرای آن کدام است؟ (فرض کنید n بسیار بزرگ است) (ارشد IT - آزاد ۸۹)

$$P(n) = 2^n + 100n^3 + n \log_2^n$$

$$O(n \log_2^n) \quad (۲)$$

$$O(100n^3) \quad (۱)$$

$$O(2^n) \quad (۴)$$

$$O(2^n + n^3) \quad (۳)$$

حل: گزینه ۴ درست است.

بزرگترین جمله در عبارات داده شده 2^n می‌باشد.

۲- رتبه زمانی شبه کد زیر چیست؟ (ارشد IT - دولتی ۸۶)

```
for ( i=1 ; i<=n ; i++)
    for ( j=1 ; j<=n ; j++){
        x=x+1;
        n--;
    }
```

$$n \log n \quad (۴)$$

$$\log n \quad (۳)$$

$$n^2 \quad (۲)$$

$$n \quad (۱)$$

حل: گزینه ۱ درست است.

به ازای $i=1$ ، دستور $x=x+1$ $\frac{n}{4}$ بار اجرا می‌شود. به ازای $i=2$ ، دستور $x=x+1$ $\frac{n}{2}$ بار اجرا می‌شود. بنابراین داریم:

$$\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \dots = n \left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots \right) = n \times \frac{\frac{1}{2}}{1 - \frac{1}{2}} = n$$

در نتیجه مرتبه زمانی برابر $O(n)$ می‌باشد. جمله داخل پرانتز، مجموع جملات یک تصاعد هندسی با جمله اول $1/2$ و قدرنسبت $1/2$ است. مجموع n جمله اول یک تصاعد هندسی با جمله اول a و قدر نسبت q که $|q| < 1$ ، برابر است با:

$$a + aq + aq^2 + \dots = \frac{a}{1-q}$$

۳- مرتبه زمانی شبه کد زیر کدام گزینه است؟ (ارشد IT - دولتی ۸۹)

```
for ( i=1; i<=n; i++)
    for ( j=1; j<=n; j=j+i)
        x=x+1;
```

$$n^2 \quad (۴)$$

$$\log n \quad (۳)$$

$$n \log n \quad (۲)$$

$$n \quad (۱)$$

حل: گزینه ۲ درست است.

دستور $x=x+1$ ، به ازای $i=1$ ، n بار اجرا می‌شود. به ازای $i=2$ ، $\frac{n}{2}$ بار اجرا می‌شود و بنابراین تعداد اجرای جمله برابر است با:

$$n + \frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \dots + \frac{n}{n} = n\left(1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}\right) \approx n \cdot \ln n \Rightarrow O(n \cdot \log n)$$

۴- هزینه‌ی زمانی تکه برنامه زیر کدام است؟ (ارشد IT - دولتی ۸۹)

```
int i=n;
while ( i>1){
    i /= 2;
    j = i;
    while(j>1)
    {
        j /= 3;
    }
}
```

$$O(n^2) \quad (4) \qquad O(n) \quad (3) \qquad O(\lg^2 n) \quad (2) \qquad O(\lg n) \quad (1)$$

حل: گزینه ۲ درست است.

حلقه اول از مرتبه $O(\log_2^n)$ و حلقه دوم از مرتبه $O(\log_3^n)$ است و چون دو حلقه تو در تو است، جواب ضرب آنها است. $O(\log_2^n \times \log_3^n)$

لازم به ذکر است که لگاریتم‌ها در هر پایه‌ای از مرتبه یکدیگر هستند و می‌توان به جای $O(\log_3^n)$ از $O(\log_2^n)$ استفاده کرد. بنابراین داریم:

$$O(\log_2^n \times \log_3^n) = O(\log_2^n \times \log_2^n) = O(\log_2^n)^2 = O(\lg^2 n)$$

۵- پس از اجرای قطعه کد زیر، مقدار نهایی x چه مقداری خواهد بود؟ (ارشد IT - دولتی ۸۵)

```
x=0;
for( i=1 ; i <=n ; i++) {
    for( j=1 ; j<=n ; j++)
        x++;
    j=1;
    while( j< n ) {
        x++;
        j=j*2;
    }
}
```

$$n(1 + \lfloor \log n \rfloor) \quad (4) \qquad n^2 + n \lceil \log n \rceil \quad (5) \qquad n^2 + \lceil \log n \rceil \quad (2) \qquad n \lceil \log n \rceil \quad (1)$$

حل: گزینه ۳ درست است.

حلقه اول n مرتبه و دو حلقه داخل آن $\lceil \log n \rceil$ مرتبه اجرا می‌شوند. بنابراین تعداد اجرا برابر حاصل ضرب این دو مقدار خواهد بود:

$$n(n + \lceil \log n \rceil) = n^2 + n\lceil \log n \rceil$$

۶- کدام یک از موارد زیر نادرست است؟ (ارشد کامپیوتر- آزاد ۸۶)

$$6 * 2^n + n^2 = \theta(2^n) \quad (2)$$

$$6 * 2^n + n^2 = \theta(n^2) \quad (1)$$

$$6 * 2^n + n^2 = \Omega(n^2) \quad (4)$$

$$6 * 2^n + n^2 = \Omega(2^n) \quad (3)$$

حل: جواب گزینه ۱ است.

به جای عبارت $6 * 2^n + n^2$ از عبارت 2^n استفاده می‌کنیم. (چون رشد آن بیشتر از n^2 است). همچنین به جای θ از علامت $=$ و به جای Ω از علامت \geq استفاده شود. در این صورت گزینه‌ها به صورت زیر خواهند بود:

$$2^n \geq n^2 \quad (4)$$

$$2^n \geq 2^n \quad (3)$$

$$2^n = 2^n \quad (2)$$

$$2^n = n^2 \quad (1)$$

که در این صورت نادرستی گزینه ۱ واضح است.

۷- کدام یک از عبارات زیر درست‌اند؟ (ارشد کامپیوتر- دولتی ۸۵)

الف - $e^{c\sqrt{n}} = O(e^{\sqrt{n}}) \quad c \geq 1$

ب - $n^2 = O(n \log n)$

ج - $n = O(n \log n)$

۴) الف و ج

۳) الف و ب

۲) فقط ج

۱) فقط ب

حل: گزینه ۲ درست است.

گزاره الف نادرست است، چون به ازای $c \geq 1$ داریم:

$$e^{c\sqrt{n}} \geq e^{\sqrt{n}}$$

$$. n^2 \geq n \lg n$$

فصل ۲:

زیر برنامه‌های بازگشتی

زیر برنامه بازگشتی (Recursive)، زیر برنامه‌ای است که حاوی حداقل یک دستور باشد که خود زیر برنامه را صدا بزنند. این زیر برنامه به تعداد مراحل محدودی اجرا می‌شود و پس از آن متوقف می‌شود.

به طور نمونه زیر برنامه بازگشتی زیر را در نظر بگیرید:

$$f(n) = \begin{cases} n \times f(n-1) & n > 1 \\ 1 & n = 1 \end{cases}$$

این زیر برنامه، فاکتوریل عدد n را محاسبه می‌کند. به طور نمونه برای محاسبه $f(3)$ داریم:

$$f(3) = 3 * f(2)$$

$$f(2) = 2 * f(1)$$

که با جایگذاری داریم:

$$f(1) = 1$$

$$f(2) = 2 * 1 = 2$$

$$f(3) = 3 * 2 = 6$$



زیر برنامه‌های بازگشتی معروف

۱- محاسبه n امین جمله سری فیبوناچی

$$f(n) = \begin{cases} n & n = 0, n = 1 \\ f(n-1) + f(n-2) & n > 1 \end{cases}$$

دو جمله اول سری فیبوناچی 0 و 1 می‌باشند و جملات بعدی از جمع دو جمله قبلی محاسبه می‌شوند:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

۲- مجموع اعداد 1 تا n

$$f(n) = \begin{cases} 1 & n = 1 \\ n + f(n-1) & n > 1 \end{cases}$$

۳- توان (a^b)

$$f(a, b) = \begin{cases} a & b = 1 \\ a * f(a, b-1) & b > 1 \end{cases}$$

۴- خارج قسمت تقسیم صحیح ($a \text{ div } b$)

$$f(a,b) = \begin{cases} 0 & a < b \\ f(a-b,b)+1 & a \geq b \end{cases}$$

۵- باقیمانده تقسیم صحیح ($a \text{ mod } b$)

$$f(a,b) = \begin{cases} a & a < b \\ f(a-b,b) & a \geq b \end{cases}$$

۶- ترکیب $\binom{a}{b}$

$$f(a,b) = \begin{cases} 1 & a = b \text{ or } b = 0 \\ f(a-1,b) + f(a-1,b-1) & a \neq b \text{ and } b \neq 0 \end{cases}$$

۷- بزرگترین مقسوم علیه مشترک (ب.م.م)

$$f(a,b) = \begin{cases} b & a \geq b, a \text{ mod } b = 0 \\ f(b,a) & a < b \\ f(b, b \text{ mod } a) & a \neq b \text{ and } b \neq 0 \end{cases}$$

۸- آکرمان

$$f(a,b) = \begin{cases} b+1 & a=0 \\ f(a-1,1) & b=0 \\ f(a-1, f(a,b-1)) & a \neq 0, b \neq 0 \end{cases}$$

مثال

با توجه به تابع فیبوناچی، خروجی (4) f را بدست آورید؟

حل:

$$f(4) = f(3) + f(2)$$

$$f(3) = f(2) + f(1)$$

$$f(2) = f(1) + f(0)$$

که با جایگذاری داریم:

$$f(0)=0$$

$$f(1)=1$$

$$f(2)=1+0=1$$

$$f(3)=1+1=2$$

$$f(4)=2+1=3$$



مثال

با توجه بهتابع مجموع اعداد ۱ تا n ، مقدار $f(3)$ چه خواهد بود؟

حل:

$$f(3) = 3 + f(2)$$

$$f(2) = 2 + f(1)$$

که با جایگذاری داریم:

$$f(1) = 1$$

$$f(2) = 2 + 1 = 3$$

$$f(3) = 3 + 3 = 6$$

بنابراین $f(3)$ برابر 6 خواهد شد.

تذکر: البته می‌دانیم که مجموع اعداد ۱ تا n برابر $\frac{n(n+1)}{2}$ می‌باشد و می‌توان مستقیماً در این رابطه به جای n عدد ۳ را

قرار داد.



مثال

با توجه بهتابع توان، خروجی $f(2,3)$ را بدست آورید؟

حل:

$$f(2,3) = 2 * f(2,2)$$

$$f(2,2) = 2 * f(2,1)$$

که با جایگذاری داریم:

$$f(2,1) = 2$$

$$f(2,2) = 2 * 2 = 4$$

$$f(2,3) = 2 * 4 = 8$$

بنابراین $f(2,3) = 8$. در واقع تابع $f(a,b)$ را بتوان b می‌رساند.



مثال

با توجه بهتابع خارج قسمت تقسیم صحیح ، خروجی $f(5,2)$ را بدست آورید؟

حل:

$$f(5,2) = f(3,2) + 1$$

$$f(3,2) = f(1,2) + 1$$

که با جایگذاری داریم:

$$f(1,2) = 0$$

$$f(3,2) = f(1,2) + 1 = 0 + 1 = 1$$

$$f(5,2) = f(3,2) + 1 = 1 + 1 = 2$$

در واقع تابع $f(a,b)$ مقدار a div b را محاسبه می‌کند. (خارج قسمت تقسیم صحیح)



مثال

با توجه به تابع باقیمانده تقسیم صحیح، خروجی $f(5,2)$ را بدست آورید؟

حل:

$$f(5,2) = f(3,2)$$

$$f(3,2) = f(1,2)$$

با جایگذاری داریم:

$$f(1,2) = 1 \quad \text{و} \quad f(3,2) = 1 \quad \text{و} \quad f(5,2) = 1$$



مثال

با توجه به تابع ترکیب، مقدار $f(4,2)$ کدام است؟

حل:

$$f(4,2) = f(3,2) + f(3,1)$$

$$f(3,2) = f(2,2) + f(2,1)$$

$$f(3,1) = f(2,1) + f(2,0)$$

$$f(2,1) = f(1,1) + f(1,0)$$

که با جایگذاری داریم:

$$f(2,1) = 1+1=2$$

$$f(3,1) = 2+1=3$$

$$f(3,2) = 1+2=3$$

$$f(4,2) = 3+3=6$$



عمل جمع در الگوریتم محاسبه $\binom{n}{m}$ به تعداد $n-m$ مرتبه انجام می شود.

مثال

با توجه به تابع بزرگترین مقسوم علیه مشترک (ب.م.م)، حاصل $f(18,4)$ را بدست آورید؟

حل:

چهار مرتبه تابع f برای محاسبه $f(18,4)$ صدای زده می شود:

$$f(18,4) = f(4,2) = f(2,0) = f(2,0) = 2$$

در واقع تابع، بزرگترین مقسوم علیه مشترک دو عدد 18 و 4 را که برابر 2 می باشد را محاسبه می کند.



مثال

با توجه به تابع آکرمان، مقدار $f(1,1)$ کدام است؟

حل:

$$f(1,1) = f(0,f(1,0))$$

$$f(1,0) = f(0,1)$$

که با جایگذاری داریم:

$$f(0,1)=2$$

$$f(1,0)=2$$

$$f(1,1)=f(0,2)=3$$



رابطه مقابل در تابع آکرمان برقرار است: $f(1,n)=n+2$

مثال

با توجه به تابع آکرمان، حاصل $f(1,2)$ کدام است؟

حل: با توجه به نکته قبل داریم : $f(1,2)=4$

روش دوم:

$$f(1,2)=f(0,f(1,1))=f(0,3)=4$$



در تابع آکرمان روابط زیر برقرار است:

$$f(1,n) = 2 + (n + 3) - 3$$

$$f(2,n) = 2(n + 3) - 3$$

$$f(3,n) = 2^{n+3} - 3$$

$$f(4,n) = \underbrace{2^{2^2}}_{n+3} - 3$$

$$f(4,1) = 2^{2^2} - 3 , f(4,3) = 2^{65536} - 3$$

مثال :

مثال

با توجه به تابع زیر، حاصل $f(18)$ را بدست آورید؟

$$f(a) = \begin{cases} 0 & a = 1 \\ f\left(\left\lfloor \frac{a}{2} \right\rfloor\right) + 1 & a > 1 \end{cases}$$

حل:

$$f(18)=f(9)+1$$

$$f(9)=f(4)+1$$

$$f(4)=f(2)+1$$

$$f(2)=f(1)+1$$

که با جایگذاری داریم:

$$f(2)=0+1=1 , f(4)=1+1=2 , f(9)=2+1=3 , f(18)=3+1=4$$

در واقع تابع داده شده ، کف لگاریتم n در پایه 2 را محاسبه می کند. ($\lfloor \log_2^{18} \rfloor = 4$)



مثال

مقدار $f(1,1)$ را با توجه به رابطه های بازگشتی زیر بدست آورید.

$$F(x,0)=F(x+1,0) + F(x+1,1) \quad , \text{ if } x < 3$$

$$F(x,1)=2F(x+1,0) + F(x+1,1) \quad , \text{ if } x < 3$$

$$F(3,0)=1$$

$$F(3,1)=0$$

حل:

$$f(1,1) = 2f(2,0) + f(2,1) = 2+2=4$$

$$f(2,0) = f(3,0) + f(3,1) = 1+0=1$$

$$f(2,1) = 2f(3,0) + f(3,1) = 2+0=2$$



مرتبه اجرایی توابع بازگشتی

برای پیدا کردن مرتبه اجرایی توابع بازگشتی بهتر است روش های بررسی شده در کتاب طراحی الگوریتم ها همین مولف را مطالعه کنید. در اینجا اشاره مختصری می کنیم.

مثال

یک رابطه بازگشتی برای تعداد ضرب ها در تابع فاکتوریل بنویسید.

```
int fact (int n){
    if (n==0)
        return 1;
    else
        return n*fact(n-1);
}
```

حل: برای یک n معین تعداد ضرب هایی که انجام می شود برابر است با تعداد ضرب های انجام شده در فراخوانی $(n-1)$ به اضافه عمل ضرب n در $fact(n-1)$. بنابراین اگر تعداد ضرب های انجام شده برای یک مقدار معین n را با $T(n)$ نمایش دهیم، داریم:

$$T(n) = 1 + T(n-1)$$

$$T(0) = 0$$

چنانی معادله ای را معادله بازگشتی می گویند. در این الگوریتم وقتی $n=0$ باشد هیچ ضربی صورت نمی گیرد. لذا $T(0)=0$ شرط اولیه است.

برای حل این معادله چند مقدار اولیه می دهیم تا به عملکرد آن پی ببریم:

$$T(1) = 1 + T(0) = 1 + 0 = 1$$

$$T(2) = 1 + T(1) = 1 + 1 = 2$$

$$T(3) = 1 + T(2) = 1 + 2 = 3$$

نتیجه می شود که: $T(n) = n$. بنابراین $T(n)$ از مرتبه $O(n)$ است.



مثال

بازگشتی $T(n) = T(n-2) + 1$ با فرض $T(0) = 0$ حل کنید. (n مضربی از 2 است)

حل: چند مقدار اولیه عبارتند از:

$$T(2) = T(0) + 1 = 0 + 1 = 1$$

$$T(4) = T(2) + 1 = 1 + 1 = 2$$

$$T(6) = T(4) + 1 = 2 + 1 = 3$$

نتیجه می‌شود که: $T(n) = \frac{n}{2}$



مثال

بازگشتی $T(n) = n + T(n-1)$ با فرض $T(1) = 1$ حل کنید.

حل:

$$T(1) = 1$$

$$T(2) = 2 + T(1) = 2 + 1$$

$$T(3) = 3 + T(2) = 3 + 2 + 1$$

...

$$T(n) = n + (n-1) + (n-2) + \dots + 2 + 1 = \frac{n(n+1)}{2}$$



تابع مجموع اعداد 1 تا n را محاسبه و از مرتبه $O(n^2)$ می‌باشد.



قضیه اصلی

می‌توان بعضی از معادله‌های بازگشتی را به کمک قضیه زیر که به نام قضیه اصلی شناخته می‌شود، حل کرد.

قضیه اصلی (Master)

اگر داشته باشیم: $T(n) = aT(\frac{n}{b}) + f(n)$ و با فرض $a \geq 1, b > 1$, آنگاه :

$$T(n) = \begin{cases} \Theta(n^{\log_b^a}) & f(n) < n^{\log_b^a} \\ \Theta(f(n) \cdot \lg n) & f(n) = n^{\log_b^a} \\ \Theta(f(n)) & f(n) > n^{\log_b^a} \end{cases}$$

تذکر: رابطه بالا به صورت دقیق در کتاب طراحی الگوریتم همین مولف بررسی شده است.

مثال

مرتبه اجرایی رابطه بازگشتی $T(n) = 9T\left(\frac{n}{3}\right) + n$ کدام است؟

حل: داریم $n^{\log_3^9} = n^2$. پس $f(n) = n, b = 3, a = 9$

از آنجا که $T(n) = \theta(n^2)$ ، با توجه به حالت اول قضیه اصلی، داریم:



مثال

مرتبه اجرایی رابطه بازگشتی $T(n) = T\left(\frac{2n}{3}\right) + 1$ کدام است؟

حل: داریم $n^{\log_{3/2}^1} = n^0 = 1$ و بنابراین داریم: $f(n) = 1, b = \frac{3}{2}, a = 1$

از آنجا که $T(n) = \theta(\lg n)$ ، با توجه به حالت دوم قضیه اصلی، داریم:



مثال

مرتبه اجرایی رابطه بازگشتی $T(n) = 3T\left(\frac{n}{4}\right) + n \lg n$ کدام است؟

حل: داریم $n^{\log_4^3} = O(n^{0.793})$ و بنابراین داریم: $f(n) = n \lg n, a = 3, b = 4$

از آنجا که $T(n) = \theta(n \lg n)$ ، با توجه به حالت سوم قضیه اصلی، داریم:



بررسی چند مثال

رابطه بازگشته	مرتبه اجرایی
$T(n) = 4T\left(\frac{n}{2}\right) + \lg n$	$f(n) = \lg n < n^{\log_2^4} \Rightarrow T(n) = \theta(n^2)$
$T(n) = 8T\left(\frac{n}{9}\right) + n \lg n$	$f(n) = n \lg n > n^{\log_9^8} \Rightarrow T(n) = \theta(n \lg n)$
$T(n) = T\left(\frac{2n}{3}\right) + 1$	$f(n) = 1 = n^{\log_{3/2}^1} \Rightarrow T(n) = \theta(\lg n)$
$T(n) = 9T\left(\frac{n}{3}\right) + n$	$f(n) = n < n^{\log_3^9} \Rightarrow T(n) = \theta(n^2)$
$T(n) = 3T\left(\frac{n}{4}\right) + n \lg n$	$f(n) = n \lg n > n^{\log_4^3} \Rightarrow T(n) = \theta(n \lg n)$
$T(n) = 8T\left(\frac{n}{4}\right) + 5n^2$	$f(n) = 5n^2 > n^{\log_4^8} \Rightarrow T(n) = \theta(n^2)$
$T(n) = 2T\left(\frac{n}{2}\right) + n^3$	$f(n) = n^3 > n^{\log_2^2} \Rightarrow T(n) = \theta(n^3)$
$T(n) = 7T\left(\frac{n}{3}\right) + n^2$	$f(n) = n^2 > n^{\log_3^7} \Rightarrow T(n) = \theta(n^2)$
$T(n) = 7T\left(\frac{n}{2}\right) + n^2$	$f(n) = n^2 < n^{\log_2^7} \Rightarrow T(n) = \theta(n^{\lg 7})$
$T(n) = 16T\left(\frac{n}{4}\right) + n^2$	$f(n) = n^2 = n^{\log_4^{16}} \Rightarrow T(n) = \theta(n^2 \lg n)$

تبصره قضیه اصلی

در قضیه اصلی، اگر $\frac{f(n)}{n^{\log_a b}} < n^\varepsilon$ باشد، (یعنی $f(n)$ به صورت چند جمله‌ای از $n^{\log_a^b}$ بزرگتر نباشد) آنگاه نمی‌توان از قضیه اصلی استفاده کرد. در این حالت اگر $f(n)$ از مرتبه $T(n)$ برابر $n^{\log_b^a} \cdot \lg^{k+1} n$ است.

در زیر چند مثال آورده شده است:

مرتبه اجرایی	رابطه بازگشتی
$T(n) = 2T\left(\frac{n}{2}\right) + n \lg n$	$\frac{n \lg n}{n} = \lg n < n^\varepsilon \Rightarrow T(n) = \theta(n \lg^2 n)$
$T(n) = 4T\left(\frac{n}{2}\right) + n^2 \lg n$	$\frac{n^2 \lg n}{n^2} = \lg n < n^\varepsilon \Rightarrow T(n) = \theta(n^2 \lg^2 n)$
$T(n) = 4T\left(\frac{n}{2}\right) + n^2 \lg^3 n$	$\frac{n^2 \lg^3 n}{n^2} = \lg^3 n < n^\varepsilon \Rightarrow T(n) = \theta(n^2 \lg^4 n)$

کنکور ارشد

۱- فرض می‌گردد n و m مقادیری مثبت و بزرگتر از یک را دارا می‌باشند. تابع زیر چه عملی را انجام می‌دهد؟

(مهندسی کامپیوتر - آزاد ۸۹)

```
int f(int n, int m){
    if (m==2)
        return n;
    else
        return n*f(n,m-1);
}
```

$$m^n \quad (4)$$

$$n^{m+1} \quad (3)$$

$$n^{m-1} \quad (2)$$

$$n^m \quad (1)$$

حل: گزینه ۲ درست است.

در واقع همان تابع توان است، با این تفاوت که شرط اولیه آن تغییر کرده است. اگر شرط توقف به جای $n = 2$ برابر $f(1) = n$ بود، آنگاه گزینه ۴ درست بود.

راه حل کلی:
می‌توان از روش تکرار و جایگذاری استفاده کرد:

$$\begin{aligned} f(n, m) &= n * f(n, m-1) \\ &= n * n * f(n, m-2) \\ &= n * n * n * f(n, m-3) \\ &= \dots \\ &= n * n * n * \dots * n * f(n, 2) \\ &= n * n * n * \dots * n * n \\ &= n^{m-1} \end{aligned}$$

از آنجا که وقتی که $m=2$ می‌شود، به جای $f(n, 2)$ ، مقدار n را قرار می‌دهیم، پس تابع به تعداد $m-1$ مرتبه خود را فراخوانی می‌کند.



۲- تابع زیر چه فرمولی را محاسبه می‌نماید؟ (مهندسی IT - آزاد ۸۸)

```
Function N(a,b : integer) : integer;
begin
    if a<b then
        N:=a
    else
        N:=N(a-b, b)
end;
```

$$(a-b) \text{ div } b \quad (4)$$

$$a \text{ div } b \quad (3)$$

$$a \bmod b \quad (2)$$

$$a^b \quad (1)$$

حل: گزینه ۲ درست است.



تابع داده شده باقیمانده تقسیم صحیح a بر b را محاسبه می‌کند.

۳- حاصل تابع $Q(5861,7)$ کدام است؟ (مهندسی کامپیوتر - آزاد ۸۴)

$$Q(a, b) = \begin{cases} 0 & \text{if } a < b \\ Q(a - b, b) + 1 & \text{if } a \geq b \end{cases}$$

858 (۴)

850 (۳)

843 (۲)

837 (۱)

حل: گزینه ۱ درست است.

این تابع خارج قسمت صحیح تقسیم a بر b را محاسبه می‌کند. بنابراین داریم:

5861 div 7=837



۴- پیچیدگی زمانی الگوریتم زیر از چه درجه‌ای است؟ (علوم کامپیوتر - دولتی ۸۹)

```
int f(int n, int m){  
    if (n<=1 || m<=1)  
        return 1;  
    return n + m * f( n-1 , m/2 );}
```

logm (۲)

n (۱)

max(n , logm) (۴)

min(n , logm) (۳)

حل: جواب گزینه ۳ است.

تابع f به فرم $f(n-1, m/2)$ خود را فراخوانی می‌کند. بنابراین اگر $n-1$ زودتر به ۱ برسد، از مرتبه $O(n)$ و اگر $\frac{m}{2}$ زودتر به ۱

برسد، از مرتبه $O(\log m)$ می‌باشد. در نتیجه تابع از مرتبه $\min(n, \log m)$ می‌باشد.



فصل ۳:

آرایه

داده‌ها می‌توانند به روش‌های متفاوتی سازماندهی شوند. مدل منطقی یا ریاضی سازماندهی داده‌ها به یک صورت خاص، یک ساختمان داده نامیده می‌شود. انواع ساختمان داده‌ها عبارتند از:

۱- ایستا (مانند آرایه و رکورد)

۲- نیمه ایستا (مانند صف و پشته)

۳- پویا (مانند لیست پیوندی، درخت و گراف)

ساختمان داده‌های ایستا در طول حیات آنها ثابت است ولی در نوع پویا تغییرات مجاز و نامحدود است. عملیات رایج در ساختمان داده‌ها عبارتند از: اضافه کردن (insert)، حذف داده (delete)، جستجوی داده (search) و پیمایش.

آرایه

آرایه مجموعه‌ای از داده‌های هم نوع است که تحت یک نام معرفی شده و برای دسترسی به هر عنصر آن از اندیس مشخصی استفاده می‌شود.

مثال

در زیر نحوه تعریف آرایه‌های یک بعدی و دو بعدی در زبان پاسکال آورده شده است:

الف- تعریف آرایه‌ای یک بعدی به نام `a` شامل 3 عنصر (اندیسهای 2 تا 4) از نوع صحیح:

`a : array [2..4] of integer ;`

ب- تعریف آرایه‌ای دو بعدی به نام `b` شامل 6 عنصر (2 سطر و 3 ستون) از نوع کاراکتری:

`b : array [1..2,4..6] of char ;`

مثال

در زیر نحوه تعریف آرایه‌های یک بعدی و دو بعدی در زبان `c` آورده شده است:

الف- تعریف آرایه‌ای یک بعدی به نام `c` شامل 3 عنصر (اندیس‌های 0 تا 2) از نوع اعشاری:

`float c[3];`

ب- تعریف آرایه‌ای دو بعدی به نام `d` شامل 12 عنصر (3 سطر و 4 ستون) از نوع صحیح:

مثال

تعداد عناصر آرایه چهار بعدی `[A[2..5,3..8,1..3,3..4]` را بدست آورید؟

حل:

$$(5 - 2 + 1)(8 - 3 + 1)(3 - 1 + 1)(4 - 3 + 1) = 4 \times 6 \times 3 \times 2 = 144$$

 تعداد عناصر آرایه n بعدی به شکل $A[l_1..u_1, l_2..u_2, \dots, l_n..u_n]$ برابر است با:

نحوه ذخیره عناصر آرایه در حافظه

عناصر آرایه در حافظه به صورت پشت سر هم قرار می‌گیرند که موجب سریع شدن سرعت دسترسی به عناصر آرایه می‌شود. با فرض اینکه عنصر اول آرایه در آدرس α حافظه ذخیره شود و هر عنصر آرایه به اندازه w بایت فضای اشغال نماید، محل هر عنصر آرایه در حافظه به کمک روابط زیر محاسبه می‌شود:

۱- آدرس عنصر $A[i]$ در آرایه یک بعدی :

$$\alpha + (i - l) \times w$$

۲- آدرس عنصر $A[i,j]$ در آرایه دو بعدی :

$$\alpha + [(i - l_1) \times (u_2 - l_2 + 1) + (j - l_2)] \times w$$

سطری : سنتونی :

۳- آدرس عنصر $A[i,j,k]$ در آرایه سه بعدی :

$$\alpha + [(i - l_1) \times (u_2 - l_2 + 1) \times (u_3 - l_3 + 1) + (j - l_2) \times (u_3 - l_3 + 1) + (k - l_3)] \times w$$

سطری سنتونی :

۴- آدرس عنصر $A[i, j, k, t]$ در آرایه چهار بعدی :

$$(d_i = u_i - l_i + 1)$$

سطری :

$$\alpha + [(i - l_1) \times d_2 \times d_3 \times d_4 + (j - l_2) \times d_3 \times d_4 + (k - l_3) \times d_4 + (t - l_4)] \times w$$

سطری :

$$\alpha + [(i - l_1) + (j - l_2) \times d_1 + (k - l_3) \times d_1 \times d_2 + (t - l_4) \times d_1 \times d_2 \times d_3] \times w$$

مثال

مطلوب است آدرس عنصر $x[7]$ در حافظه؟ (عنصر اول آرایه در آدرس 1000 حافظه قرار دارد)
 $x : \text{array } [-2..30] \text{ of Real;}$

حل: متغیر از نوع real در پاسکال ، 6 بایتی است.

$$1000 + [7 - (-2)] \times 6 = 1054$$



مثال

آرایه دو بعدی x با 3 سطر و 4 ستون مفروض است. اگر خانه اول آرایه در محل 10 حافظه ذخیره شود. محل عنصر $[2,3]$ در حافظه کدام است؟ ($w=1$)

$x : \text{array}[1..3, 1..4] \text{ of char} ;$

حل:

$$10 + [(2-1)(4-1+1)+(3-1)] \times 1 = 16 \quad \text{سطری :}$$

$$10 + [(2-1)+(3-1)(3-1+1)] \times 1 = 17 \quad \text{ستونی :}$$



مثال

آرایه سه بعدی $(A[1..14, 2..13, 3..14])$ از آدرس 100 حافظه قرار گرفته می‌باشد. آدرس عنصر $A(9,7,6)$ چه خواهد بود؟ ($w=1$)

حل:

$$100 + [(9-1) \times (13-2+1) \times (14-3+1) + (7-2) \times (14-3+1) + (6-3)] \times 1 = 1315 \quad \text{سطری :}$$

$$100 + [(9-1) + (7-2) \times (11-1+1) + (6-3) \times (13-2+1) \times (11-1+1)] \times 1 = 559 \quad \text{ستونی :}$$



مثال

آرایه چهار بعدی $[A[1..10, 1..20, 1..10, 1..20]]$ را در نظر بگیرید که به صورت سطری ذخیره شده است. اگر آدرس شروع آرایه صفر باشد، آدرس عنصر $A[1,2,3,4]$ کدام است؟ ($w=1$)

حل:

$$0 + [(1-1) \times 20 \times 10 \times 20 + (2-1) \times 10 \times 20 + (3-1) \times 20 + (4-1)] \times 1 = 243$$



مثال

اگر آرایه $x[-2,-2; 3, -4:-2, 'A':'F', 0:3]$ با طول داده ای 2 و از آدرس 100 در حافظه ذخیره شده باشد، آدرس $2, 'D', 0]$ کدام است؟

$$100 + [(-2+2) \times 3 \times 6 \times 4 + (-2+4) \times 6 \times 4 + ('D'-'A') \times 4 + (0-0)] \times 2 = 220$$



جستجو در آرایه

به دو روش خطی و دودویی می‌توان در یک آرایه جستجو کرد.

جستجوی خطی

تابع زیر مقدار x را در آرایه A ، به روش مقایسه با تک تک عناصر آرایه، جستجو می‌نماید. در صورت پیدا کردن، اندیس خانه حاوی x و در صورت پیدا نکردن، عدد -1 را بر می‌گرداند.

```
int seqsearch (int a[ ] , int n , int x){
    int i;
    a[n] = x;
    for ( i = 0 ; i < n ; i++)
        if (a[i] == x)
            return i;
    return -1;
}
```

میانگین تعداد عمل مقایسه در یک جستجوی موفقیت آمیز برابر $\frac{n+1}{2}$ می‌باشد.

در یک جستجوی ناموفق نیاز به $n+1$ عمل مقایسه داریم که در نتیجه زمان آن $O(n)$ خواهد بود.

جستجوی دودویی

اگر عناصر یک آرایه مرتب شده باشند، بهتر است از روش جستجوی دودویی استفاده کرد. در این روش با فرض اینکه آرایه به طور صعودی مرتب شده باشد، عنصر مورد جستجو با عنصر وسط آرایه مقایسه می‌شود، در صورت برابر بودن، پیدا شده است و در غیر اینصورت اگر از عنصر وسط آرایه بزرگتر باشد، مقایسه به طور بازگشتی در نیمه بالایی آرایه انجام می‌گیرد و در صورت کوچکتر بودن از عنصر وسط، مقایسه به طور بازگشتی در نیمه پایینی آرایه انجام می‌شود.

تابع زیر مقدار x را در آرایه A ، به روش دودویی، جستجو می‌نماید. در صورت پیدا کردن، اندیس خانه حاوی x و در صورت پیدا نکردن، عدد -1 را بر می‌گرداند. (در ابتدا : $low=0$, $high=n-1$)

```
int bsearch( int a[ ] , int x , int low , int high){
    int mid ;
    while(low <=high)
    {
        mid = (low+high)/2;
        if (x < a[mid])
            high = mid-1;
        else
            if (x > a[mid])  low = mid+1;  else  return mid;
    }
    return -1;
}
```

پیاده سازی بازگشته جستجوی دودویی

```

int bsearch (int a[ ] , int x , int low , int high){
    int mid;
    if (low <=high){
        mid = (low+high)/2;
        if (x < a[mid])
            bsearch(a , x , low , mid-1);
        else if (x > a[mid])
            bsearch(a , x , mid+1 , high);
        else return mid;
    }
    return -1;
}

```

رابطه بازگشته تابع بالا $T(n) = T\left(\frac{n}{2}\right) + 1$ بوده که از مرتبه $O(\lg n)$ است.

مثال

برای پیدا کردن عدد 9 در آرایه مرتب زیر با روش جستجوی دودویی، به چند مقایسه نیاز است؟

1	2	3	4	5	6	7	8	9
5	9	12	20	35	50	82	88	97

حل:

ابتدا عدد 9 با عنصر وسط آرایه یعنی 35 مقایسه می شود و چون از آن کوچکتر است مقایسه به طور بازگشته در زیر آرایه $[1..4]x$ انجام می گیرد. بنابراین مقایسه بعدی با عنصر وسط این آرایه یعنی 9 است که برابر است. بنابراین با دو مقایسه به نتیجه می رسیم.



حداکثر تعداد مقایسه‌ها برای پیدا کردن یک عنصر در آرایه مرتب n عنصری به روش جستجوی دودویی برابر $\lfloor \lg n \rfloor + 1$ می باشد.

مثال

حداکثر چند عمل مقایسه برای پیدا کردن یک عدد به روش جستجوی دودویی در آرایه مرتب هزار عنصری انجام می گیرد؟

حل: کافی است در رابطه $\lfloor \lg n \rfloor + 1$ به جای n ، عدد 1000 قرار دهیم که حاصل برابر 10 خواهد شد.

البته می توان به جای عدد هزار، از عدد توان دو و بزرگتر از آن یعنی 1024 استفاده کرده و از یک صرف نظر کرد.

 حداکثر تعداد مقایسه‌ها برای پیدا کردن عنصری به روش جستجوی دودویی در یک آرایه با هزار عنصر برابر ۱۰، در آرایه با ده هزار عنصر برابر ۱۴، در آرایه با صد هزار عنصر برابر ۱۷ و در آرایه با یک میلیون عنصر برابر ۲۰ می‌باشد. بنابراین برتری جستجوی خطی در آرایه با تعداد عناصر زیاد بیشتر خود را نشان می‌دهد.

مثال

میانگین تعداد مقایسه‌ها برای یافتن یک عنصر دلخواه در آرایه مرتب شده با ۱۰ عنصر با استفاده از جستجوی دودویی کدام است؟

حل: در شکل زیر تعداد مقایسه‌های لازم برای هر عنصر در زیر آن نوشته شده است:

۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰
12	20	26	30	45	65	68	70	75	92
3	2	3	4	1	3	4	2	3	4

به طور نمونه برای پیدا کردن عدد ۴۵ نیاز به یک مقایسه است، چون در وسط آرایه قرار دارد. همچنین برای پیدا کردن عدد ۲۰ نیاز به ۲ مقایسه می‌باشد، چون در ابتدا با مقدار ۴۵ مقایسه شده و چون از آن کوچکتر است با وسط آرایه پایین یعنی ۲۰ مقایسه می‌شود. برای محاسبه میانگین تعداد مقایسه‌ها کافی است مجموع این اعداد را بر تعداد آنها تقسیم کرد، یعنی:

$$\frac{29}{10}$$

روش دوم: می‌توان با اعداد داده شده یک درخت دودویی کامل ساخت و سپس مجموع حاصلضرب تعداد گره‌ها در یک سطح و سطح آن گره‌ها را محاسبه کرد و در نهایت حاصل را به تعداد اعداد تقسیم کرد. به طور نمونه برای آرایه ۱۰ عنصری بالا داریم:

$$\frac{1 \times 1 + 2 \times 2 + 4 \times 3 + 3 \times 4}{10} = \frac{29}{10}$$

مثال

میانگین تعداد مقایسه‌ها برای پیدا کردن یک عنصر معلوم به روش جستجوی دودویی در یک آرایه ۲۰۰ عنصری مرتب کدام است؟

حل: با فرض اینکه N_K ، تعداد عناصری در آرایه باشد که برای تعیین مکان آن به K مقایسه نیاز است، داریم:

K	1	2	3	4	5	6	7	8
N_K	1	2	4	8	16	32	64	73

به طور نمونه، ۸ عنصر در آرایه ۲۰۰ عنصری وجود دارد که برای پیدا کردن آنها به ۴ مقایسه نیاز است.

تذکر: عدد ۷۳ از رابطه مقابل بدست می‌آید:

$$200 - (1+2+4+8+16+32+64) = 73$$

بنابراین میانگین تعداد مقایسه‌ها برابر است با:

$$g(n) = \frac{1}{n} \sum_{k=1}^8 k \cdot n_k = \frac{1.1 + 2.2 + 3.4 + 4.8 + 5.16 + 6.32 + 7.64 + 8.73}{200} = \frac{1353}{200} = 6.765$$

مثال

میانگین تعداد مقایسه‌ها برای جستجوی موفق (S) و همچنین میانگین تعداد مقایسه‌ها در جستجوی ناموفق (U) را برای آرایه 4 عنصری مرتب زیر بدست آورید.

حل:

تعداد مقایسه‌ها در جستجوی موفق برای هر یک از عناصر در زیر آن آورده شده است:

5	10	15	20
2	1	2	3

$$S = \frac{2+1+2+3}{4} = \frac{8}{4}$$

تعداد مقایسه‌ها در جستجوی ناموفق برای هر 5 حالت ممکن در شکل زیر آورده شده است:

5	10	15	20
2	2	2	3

به طور نمونه برای جستجوی عدد 4 که در آرایه نیست (جستجوی ناموفق)، ابتدا مقایسه با وسط آرایه یعنی عدد 10 انجام شده و سپس با عدد 5 مقایسه انجام می‌شود. یعنی با 2 مقایسه متوجه می‌شویم که عدد 4 در آرایه نمی‌باشد. همچنین برای جستجوی عددی مابین 10,5 10,5 مانند عدد 7، با 2 مقایسه مشخص می‌شود که در آرایه نیست و یا برای جستجوی عددی بزرگتر از 30، با 3 مقایسه مشخص می‌شود که در آرایه نمی‌باشد.

بنابراین میانگین تعداد مقایسه‌ها در جستجوی ناموفق برابر است با:

$$U = \frac{2+2+2+3+3}{5} = \frac{12}{5}$$

■ بین S و U رابطه $S = (1 + \frac{1}{n})(U - 1)$ برقرار می‌باشد. که می‌توان آن را به صورت $(\frac{n}{n+1})(S+1) = U$ نیز نوشت.

الگوریتم جستجوی سه تایی (ترنری)

در جستجوی سه تایی، آرایه باید به سه قسمت تقسیم شود. در هر تکرار نقطه $\frac{1}{3}$ که با m_1 و نقطه $\frac{2}{3}$ که با m_2 نشان داده

می‌شود، به صورت زیر محاسبه می‌گردد:

(L حد پایین و h حد بالا)

$$m_1 = L + \frac{1}{3}(h - L) = \frac{3L + h - L}{3} = \frac{h + 2L}{3}$$

$$m_2 = L + \frac{2}{3}(h - L) = \frac{3L + 2h - 2L}{3} = \frac{2h + L}{3}$$

اضافه و حذف در آرایه

تابع زیر مقدار صحیح x را در مکان k ام آرایه a با n عنصر که m عنصر آن پر است، ($m < n$) اضافه می‌کند:

```
insert (int a[ ] ,int m , int k , int x)
{
    for( i = m-1 ; i >= k ; i--)
        a[i+1] = a[i];
    a[k] = x;
}
```

تابع زیر k امین عنصر آرایه a را حذف کرده و آن را در متغیر x قرار می‌دهد. ($k <= n$)

```
delete(int a[ ] , int n , int k , int x)
{
    x = a[k];
    for ( i = k ; i<n ; i++)
        a[i] = a[i+1];
    a[i] = 0;
    return(x);
}
```

تعداد شیفت مورد نیاز برای اضافه کردن (insert) در محل k ام آرایه، برابر است با: $m - k$

تعداد شیفت مورد نیاز برای حذف عنصر واقع در محل k ام آرایه، برابر است با: $m - k - 1$

پیدا کردن عنصر کمینه در یک آرایه

الگوریتم زیر کوچکترین عنصر در آرایه $A[1..n]$ را پیدا می‌کند و در \min ذخیره می‌کند:

MINIMUM(A, n)

```
1   min ←  $-\infty$ 
2   for i ← 1 to n do
3       if min > A[i] then
4           min ← A[i]
```

اگر $A[i]$ ، کوچکترین عنصر زیر آرایه $A[1..i]$ باشد، روشن است که $\min A[i]$ مقدار $\min A[1..i]$ را در سطر ۴ تغییر می‌دهد. احتمال

این امر $\frac{1}{i}$ است. بنابراین میانگین تعداد دفعاتی است که سطر ۴ الگوریتم بالا اجرا می‌شود برابر است با: $(n \lg n)$

ماتریس

آرایه دو بعدی را ماتریس می‌گویند که انواع معروف آن عبارتنند از: اسپارس، مثلثی، سه قطری، متقارن و پله‌ای. قبل از پرداختن به این ماتریس‌ها، عملیات رایج بر روی ماتریس را بررسی می‌نماییم.

۱- جمع دو ماتریس $B_{m \times n}$ و $A_{m \times n}$

```
void add(int a[ ][max], int b[ ][max], int c[ ][max], int m, int n){
    for(i=0; i<m ; i++)
        for(j=0; j<n ; j++)
            c[i][j] = a[i][j] + b[i][j];
}
```

۲- ضرب دو ماتریس $B_{p \times n}$ و $A_{m \times p}$

```
void mult(int a[ ][max], int b[ ][max], int c[ ][max], int m, int p, int n){
    for(i=0; i<m ; i++)
        for(j=0; j<n ; j++)
        {
            c[i][j] = 0;
            for(k=0; k<p ; k++)
                c[i][j] = c[i][j] + a[i][k] * b[k][j];
        }
}
```

 ماتریس c دارای ابعاد $m \times n$ است و داریم:

$$c_{ij} = \sum_{k=0}^{p-1} a_{ik} b_{kj} \quad 0 \leq j < n, 0 \leq i < m$$

 مرتبه اجرایی الگوریتم جمع دو ماتریس n^2 برابر $O(n^2)$ و مرتبه ضرب آنها برابر $O(n^3)$ می‌باشد.

۳- ترانهاده کردن ماتریس $A_{n \times m}$

برای پیدا کردن ترانهاده یک ماتریس باید جای سطرها و ستونها را عوض کنیم.

```
for ( j=0 ; j<m ; j++)
    for( i=0 ; i<n ; i++)
        b[j][i] = a[i][j];
```

 در ماتریس $A_{n \times n}$ ، عنصر $A_{i,j}$ روی قطر فرعی قرار دارد، اگر $i+j = n+1$.

ضرب بهینه ماتریس‌ها

در ضرب چند ماتریس در یکدیگر باید ضربها به نحوی انجام شود که تعداد آنها حداقل باشد. بنابراین ابتدا دو ماتریسی را ضرب می‌کنیم که عدد حذف شده بزرگتر باشد.

تذکر مهم: اگر فاصله ابعاد ماتریس‌ها بسیار کم باشد، روش گفته شده همیشه جواب نمی‌دهد و از روشی که در کتاب طراحی الگوریتم اینجانب در فصل پویا توضیح داده شده، باید استفاده کرد.

 در ضرب دو ماتریس $A_{m \times n}, B_{n \times k}$ به $m \times n \times k$ عمل ضرب نیاز می‌باشد.

مثال

حداقل تعداد ضرب‌های لازم برای ضرب ۴ ماتریس زیر کدام است؟

$$A_{30 \times 1}, B_{1 \times 40}, C_{40 \times 10}, D_{10 \times 25}$$

حل: ابتدا دو ماتریس B و C را ضرب می‌کنیم، چون بعد مشترک آنها یعنی 40 از همه بیشتر است:

$$A_{30 \times 1} (BC)_{1 \times 10} D_{10 \times 25}$$

سپس ماتریس بدست آمده از مرحله قبل را در ماتریس D ضرب می‌کنیم:

$$A_{30 \times 1} (BCD)_{1 \times 25}$$

و در نهایت ماتریس A را در ماتریس بدست آمده ضرب می‌کنیم:

$$(ABCD)_{30 \times 25}$$

تعداد ضرب‌ها برابر است با:

$$1 \times 40 \times 10 + 1 \times 10 \times 25 + 30 \times 1 \times 25 = 1400$$

 تعداد حالات شرکت پذیری ضرب $n+1$ ماتریس برابر است با: $\frac{\binom{2n}{n}}{(n+1)}$

مثال

تعداد حالات شرکت پذیری ضرب 4 ماتریس چند می‌باشد؟

حل: با قرار دادن عدد 3 در رابطه $\frac{\binom{2n}{n}}{(n+1)}$ جواب برابر 5 خواهد شد. این 5 حالت به صورت زیر می‌باشد:

$$(AB)(CD), ((A(BC))D), (A((BC)D)), (((AB)C)D), (A(B(CD)))$$

انواع ماتریس

ماتریسهای معروف عبارتند از:

- ۱- اسپارس
- ۲- ماتریس مثلثی (پایین مثلثی و بالا مثلثی)
- ۳- ماتریس قطری (سه قطری ، پنج قطری و ...)

ماتریس اسپارس (خلوت)

ماتریسی که دارای تعداد نسبتاً زیادی عنصر صفر باشد را ماتریس اسپارس(خلوت یا تنک)می نامند. در این ماتریس، برای کاهش حافظه مصرفی و زمان اجرا، فقط عناصر غیرصفر ماتریس ذخیره می‌شوند.

ذخیره ماتریس اسپارس به کمک آرایه دو بعدی

ماتریس اسپارس را می‌توان به صورت آرایه دو بعدی ذخیره کرد:

```
var a : array[0..count , 1..3] of integer;
```

`a[0,3]` تعداد عناصر غیر صفر ماتریس اسپارس است. در خانه `[0,1]` ، `a[0,2]` تعداد سطرها در `[0,3]` و در `[1,3]` تعداد عناصر غیر صفر ماتریس قرار می‌گیرد. در خانه‌های `[1,1]` و `[1,2]` و `[1,3]` به ترتیب سطر و ستون و مقدار اولین عنصر غیرصفر قرار می‌گیرد و به همین ترتیب عناصر غیر صفر دیگر ذخیره می‌شود.

مثال

ذخیره ماتریس اسپارس S در ماتریس دو بعدی a :

$$S = \begin{pmatrix} 0 & 0 & 2 & 0 \\ 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow a = \begin{pmatrix} 3 & 4 & 2 \\ 1 & 3 & 2 \\ 2 & 2 & -6 \end{pmatrix}$$

سطر اول ماتریس a ، نشان می‌دهد که ماتریس اسپارس دارای 3 سطر و 4 ستون و 2 عنصر غیر صفر است و سطر دوم ماتریس a نشان می‌دهد که در سطر 1 و ستون 3 ماتریس اسپارس مقدار 2 قرار دارد. و سطر سوم نشان می‌دهد که در سطر دوم و ستون دوم ماتریس اسپارس، مقدار 6- قرار دارد.

ترانهاده کردن ماتریس اسپارس

برای ترانهاده کردن یک ماتریس کافی است که سطر اول با ستون اول ، سطر دوم با ستون دوم و تعویض شود. برای ترانهاده کردن یک ماتریس اسپارس که به صورت آرایه دو بعدی ذخیره شده ، کافی است که محل ستون اول و ستون دوم در آرایه A را با یکدیگر جابجا کرده و سپس از سطر دوم تا سطر آخر را بطور مرتب نوشت.

مثال

ترانهاده ماتریس اسپارس S که به کمک ماتریس a نمایش داده شده را بدست آورید.

$$S = \begin{pmatrix} 0 & 0 & 2 & 0 \\ 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \Rightarrow a = \begin{pmatrix} 3 & 4 & 2 \\ 1 & 3 & 2 \\ 2 & 2 & -6 \end{pmatrix}$$

حل:

در ماتریس a، محل ستون اول و دوم را با یکدیگر جابجا کرده:

4	3	2
3	1	2
2	2	-6

و سپس سطر 2 و سطر 3 را بطور مرتب می نویسیم:

4	3	2
2	2	-6
3	1	2



الگوریتم سریع ترانهاده کردن ماتریس اسپارس $m \times n$ از مرتبه $O(n + t)$ می باشد.

(t : تعداد عناصر غیر صفر)

ماتریس مثلثی

ماتریسی که تمام عناصر بالای قطر اصلی آن صفر باشد را ماتریس پایین مثلثی می گویند. ماتریسی که تمام عناصر پایین قطر اصلی آن صفر باشد را ماتریس بالا مثلثی می گویند. برای ذخیره این ماتریس، فقط عناصر غیر صفر ذخیره می شوند.

مثال

ماتریس پایین مثلثی A را به ترتیب سطحی در آرایه یک بعدی B ذخیره نمایید.

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 3 & 6 & 0 \\ 5 & 4 & 1 \end{bmatrix}$$

حل: کافی است فقط عناصر غیر صفر ماتریس A را در آرایه یک بعدی B ذخیره نماییم:

1	2	3	4	5	6
B	2	3	6	5	4

در این حالت عنصر $B[k]$ در $A[i,j]$ ذخیره می‌شود به طوری که:

$$K = \frac{i(i-1)}{2} + j$$

به طور نمونه $A[2,2]$ در $B[5]$ و $A[3,2]$ در $B[3]$ ذخیره می‌شود.



در ماتریس مثلثی، تعداد عناصر غیر صفر برابر $\frac{n(n+1)}{2}$ و تعداد عناصر صفر برابر $\frac{n(n-1)}{2}$ می‌باشد.

مثال

ماتریس پایین مثلثی A را به ترتیب ستونی در آرایه یک بعدی C ذخیره نمایید.

$$A = \begin{bmatrix} 2 & 0 & 0 \\ 3 & 6 & 0 \\ 5 & 4 & 1 \end{bmatrix}$$

حل: کافی است فقط عناصر غیر صفر ماتریس A را در آرایه یک بعدی B ذخیره نماییم:

$$C \quad \boxed{\begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 3 & 5 & 6 & 4 & 1 \end{array}}$$

در این حالت عنصر $B[k]$ در $A[i,j]$ ذخیره می‌شود به طوری که:

$$k = i + (j-1)(n - \frac{j}{2})$$

به طور نمونه $A[2,2]$ در $B[5]$ و $A[3,2]$ در $B[3]$ ذخیره می‌شود.



به طور کلی داریم:

ستونی	سطری	
$k = n \times (j-1) - \frac{j(j-1)}{2} + i$	$k = \frac{i(i-1)}{2} + j$	پایین مثلثی
$k = \frac{j(j-1)}{2} + i$	$k = n \times (i-1) - \frac{i(i-1)}{2} + j$	بالا مثلثی

تذکر: می‌توان با اضافه کردن $1 - \alpha$ به فرمولهای بالا، به یک رابطه کلی رسید.

مثال

ماتریس پایین مثلثی مثال قبل را به ترتیب قطری در آرایه یک بعدی B ذخیره نمایید.(ابتدا قطر اصلی)
حل:

نحوه ذخیره به صورت زیر است:

	1	2	3	4	5	6
B	2	6	1	3	4	5

در این حالت عنصر $A[i,j]$ در $B[k]$ ذخیره می شود به طوری که k برابر است با:

$$[(3+2+\dots+(3-(i-j))]-3+i$$

این رابطه برای عناصر قطر اصلی برابر است با:

$$k = 3 - 3 + i = i$$

و برای قطر زیر قطر اصلی برابر است با:

$$k = [3+2] - 3 + i = 2 + i$$

و برای قطر آخر برابر است با:

$$k = [3+2+1] - 3 + i = 3 + i$$

این رابطه برای یک ماتریس پایین مثلثی $n \times n$ به صورت زیر می باشد:

$$[n + (n-1) + \dots + (n-(i-j))] - n + i$$

که می توان با اضافه کردن $1 - \alpha$ به یک رابطه کلی رسید.



ماتریس سه قطری

ماتریس سه قطری یک ماتریس مربعی $n \times n$ می باشد که درایه های غیر صفر آن روی قطر اصلی و بلافقله بالا و پائین قطر اصلی ظاهر می شوند. تعداد عناصر غیر صفر در این ماتریس برابر $2 - 3n$ می باشد.

مثال

ماتریس سه قطری 4×4 زیر را به صورت سطری در یک آرایه یک بعدی ذخیره نمایید.

$$\begin{bmatrix} 6 & 23 & 0 & 0 \\ 1 & 3 & 7 & 0 \\ 0 & 2 & 4 & 83 \\ 0 & 0 & 9 & 5 \end{bmatrix}$$

حل:

فقط عناصر غیر صفر ماتریس را سطر به سطر در آرایه یک بعدی ذخیره می کنیم:

1	2	3	4	5	6	7	8	9	10
B	6	23	1	3	7	2	4	83	9



مثال

ماتریس سه قطری 5×5 زیر را به صورت ستونی در یک آرایه یک بعدی ذخیره نمایید.

$$\begin{vmatrix} \mathbf{10} & 5 & 0 & 0 & 0 \\ 4 & \mathbf{15} & 2 & 0 & 0 \\ 0 & 16 & \mathbf{35} & 1 & 0 \\ 0 & 0 & 36 & \mathbf{72} & 78 \\ 0 & 0 & 0 & 90 & \mathbf{60} \end{vmatrix}$$

حل:

فقط عناصر غیر صفر ماتریس را ستون به ستون در آرایه یک بعدی ذخیره می‌کنیم:

1	2	3	4	5	6	7	8	9	10	11	12	13	
B	10	4	5	15	16	2	35	36	1	72	90	78	60

اگر عناصر غیر صفر ماتریس سه قطری A به ابعاد $n \times n$ را به صورت سطری در آرایه یک بعدی B ذخیره نماییم، عنصر $[A[i,j]]$ در خانه $2i + j - 2$ آرایه B و اگر به صورت ستونی ذخیره نماییم، در خانه $i + 2j - 2$ آرایه B قرار می‌گیرد.

تذکر: می‌توان با اضافه کردن $1 - \alpha$ به یک رابطه کلی رسید.

کنکور ارشد

۱- اگر آرایه A به صورت $A[20][10][5]$ تعریف شده باشد (یعنی محدوده اندیس‌ها به صورت $A[0..19][0..9][0..4]$ باشد). با فرض این که هر عدد int چهار بایت لازم دارد و آدرس شروع آرایه صفر باشد، آدرس عنصر $A[3][4][2]$ را در حالت ستون محور (column major) عبارت است از:

(مهندسی IT - دولتی ۸۵)

۴۸۳ (۴) ۴۸۴ (۳) ۱۹۳۲ (۲) ۱۹۳۴ (۱)

حل: جواب گزینه ۲ است.

$$0 + [(3 - 0) + (4 - 0) \times 20 + (2 - 0) \times 20 \times 10] \times 4 = 1932$$



۲- آرایه $S[2][3]=\{2,4,6,8,10,12\}$ را در نظر بگیرید. با فرض آنکه زبان مربوطه آرایه‌ها را به صورت سطري ذخیره نموده، اگر آدرس شروع آرایه $0X\ 0012ff80$ باشد، آدرس عضو $S[1][1]$ کدام گزینه است؟ (برای هر متغیر صحیح دو بایت اختصاص می‌دهد)

(مهندسی کامپیوتر - آزاد ۹۰)

۰X 0012ff86 (۲) ۰X 0012ff88 (۱)
۰X 0012ff82 (۴) ۰X 0012ff0c (۳)

(استفاده از $0X$ در اول هر عدد، نشان دهنده این است که عدد در مبنای ۱۶ است).

حل: گزینه ۱ جواب است.

فاصله عنصر $S[1][1]$ از شروع آرایه برابر است با:

$$[(1 - 0) \times 3 + (1 - 0)] \times 2 = 8$$

بنابراین با اضافه کردن عدد حاصله (یعنی ۸) به آدرس شروع یعنی $0012ff80$ ، آدرس $0012ff88$ به دست می‌آید.
روش دوم: آرایه داده شده به صورت زیر می‌باشد.

	0	1	2
0	2	4	6
1	8	10	12

بدون استفاده از فرمول نیز مشخص است که قبل از عنصر $S[1][1]$ که طوسی رنگ شده، چهار عنصر ۲ بایتی ذخیره شده است (سطری). بنابراین باید ۸ بایت به آدرس شروع اضافه کرد تا آدرس این عنصر مشخص شود.



۳- دستور حذف شده در زیر برنامه جستجوی دودویی کدام است؟ (علوم کامپیوتر - دولتی ۸۰)

```
procedure binsearch (a: elementarray; x:element; var left , right , j:integer)
```

```
    var middle : integer;
```

```
begin
```

```
    if (left<=right) then
```

```
        begin
```

```
            middle:=(left+right) div 2;
```

```
            case compare (x,a [middle]) of
```

```
                '>' : binsearch (a,x,middle+1, right,j);
```

```
                '=<' : دستور حذف شده
```

```
                '=': j:=middle;
```

```
            end;
```

```
        end;
```

```
        j:=-1;
```

```
    end;
```

binsearch (a,x,middle+1, right,j); (۱)

binsearh (a,x, middle-1, left,j); (۲)

binsearch (a,x,left, middle-1, j); (۳)

binsearch (a,x,left,middle,righth); (۴)

حل: گزینه ۳ درست است. اگر عنصر مورد جستجو از عنصر وسط کوچکتر باشد، جستجو باید در نیمه اول آرایه، یعنی از تا -1 middle ، به صورت بازگشتی انجام شود.

■ ۴- الگوریتم زیر را برای جستجوی x در آرایه A شامل n عنصر درنظر بگیرد. پیچیدگی زمانی این الگوریتم چیست؟ (فرض کنیم در ابتدا x ، top =n ، down=1 ورودی باشد)

(مهندسی کامپیوتر - آزاد ۸۶)

```
int search (int down , int top){  
    int tmp;  
    if (down > top) return 0;  
    else{  
        tmp= (down+top) div 2;  
        if (x== A[tmp])  
            return tmp;  
        else if (x<A[tmp])  
            return search (down,tmp-1);  
        else  
            return search (tmp+1,top);  
    }  
}
```

$O(\log_2 n)$ (۴)

$O(\frac{n}{2})$ (۳)

$O(n \log n)$ (۲)

$O(\log_{10} n)$ (۱)

حل: جواب گزینه ۴ است. الگوریتم داده شده، جستجوی دودویی می‌باشد که از مرتبه $O(\log_2 n)$ است.

۵- اگر $s(n)$ متوسط تعداد مقایسه‌ها برای جستجوی موفق در یک آرایه مرتب با طول n و $u(n)$ متوسط تعداد مقایسه‌ها برای جستجوی ناموفق در این آرایه با استفاده از روش جستجوی دودویی باشد. کدام یک از گزینه‌های زیر نادرست است؟ (مهندسی IT - دولتی ۸۳)

$$s(n) = u(n) \quad (۲)$$

$$s(n) = \theta(u(n)) \quad (۱)$$

۴) موارد ۱ و ۳ صحیح است.

$$s(n) = \left(1 + \frac{1}{n}\right)u(n) - 1 \quad (۳)$$

حل: جواب تست گزینه ۲ است.

گزینه ۱ و ۳ درست می‌باشند.

$$s = \left(1 + \frac{1}{n}\right)u - 1 = \left(\frac{n+1}{n}\right)u - 1$$

$$s = \theta(u) \quad . \quad s \approx \left(\frac{n}{n}\right)u \quad \text{و در نتیجه: } s \approx u$$

۶- اگر l حد پایین آرایه و h حد بالای یک آرایه باشد، در الگوریتم جستجوی ترنری (سه تایی) در هر تکرار نقطه $\frac{1}{3}$ که با m_1 و نقطه $\frac{2}{3}$ که با m_2 نشان داده می‌شود، به چه صورت محاسبه می‌گردد؟ (علوم کامپیوتر - دولتی ۹۱)

$$m_1 = \frac{h+2l}{3}, m_2 = \frac{2h+l}{3} \quad (۲)$$

$$m_1 = \frac{h+2l}{3}, m_2 = \frac{h-2l}{3} \quad (۱)$$

$$m_1 = \frac{l+h}{3}, m_2 = \frac{2(h-l)}{3} \quad (۴)$$

$$m_1 = \frac{l+h}{3}, m_2 = \frac{2(h+l)}{3} \quad (۳)$$

حل: گزینه ۲ درست است.

۷- برای سه ماتریس $(N_1 \times N_2 \times N_3)$ ، $N_3(p \times q)$ و $N_2(n \times p)$ ، $N_1(m \times n)$ با یکسان باشد، باید: (علوم کامپیوتر - دولتی ۸۶)

$$m = n \quad \text{یا} \quad p = q \quad (۲)$$

۱) فقط $m = n = p = q$ باشد.

$$\frac{1}{m} - \frac{1}{n} = \frac{1}{q} - \frac{1}{p} \quad (۴)$$

$$\frac{1}{m} + \frac{1}{n} = \frac{1}{p} + \frac{1}{q} \quad (۳)$$

حل: جواب گزینه ۴ است.

تعداد ضرب های $mnp + mpq$ می باشد و داریم:

$$npq + mnq = mnp + mpq$$

با تقسیم طرفین رابطه به $mnpq$ داریم:

$$\frac{1}{m} + \frac{1}{p} = \frac{1}{q} + \frac{1}{n}$$

که می توان آن را به صورت زیر نوشت.

$$\frac{1}{m} - \frac{1}{n} = \frac{1}{q} - \frac{1}{p}$$



-۸- فرض کنید A یک ماتریس $m \times n$ خلوت باشد، عناصر غیر صفر آن را به ترتیب سط्रی در یک آرایه $t \times 3$ ذخیره می کنیم.تابع زمان عمل ترانهاده گیری کدام است؟ (تعداد عناصر غیر صفر ماتریس است)

(علوم کامپیوتر- دولتی ۸۰)

$$\theta(t+n) \quad (4)$$

$$\theta(tn) \quad (3)$$

$$\theta(tm) \quad (2)$$

$$\theta(t+m) \quad (1)$$

حل: گزینه ۴ درست است.

مرتبه زمانی الگوریتم ترانهاده گیری از این ماتریس برابر $\theta(t+n)$ می باشد.

-۹- ماتریس پایین مثلثی ماتریس مربعی است $n \times n$ که عناصر بالای قطر اصلی آن برابر صفر می باشند. اگر عناصر غیر صفر را به صورت قطری با شروع از قطر اصلی در یک آرایه یک بعدی ذخیره کنیم، فرمول ذخیره سازی $loc[A[i,j]]$ چیست؟ (آدرس شروع آرایه در حافظه است و e اندازه هر عنصر آرایه می باشد). (مهندسی کامپیوتر- آزاد ۸۶)

$$\alpha + \left[\frac{i(i-1)}{2} + j-1 \right] * e \quad (1)$$

$$\alpha + [n + (n-1) + \dots + (n-(i-j))] - n + (i-1) * e \quad (2)$$

$$\alpha + \left[\frac{j(j-1)}{2} + i-1 \right] * e \quad (3)$$

$$\alpha + \left[\frac{i}{2} + j-1 \right] * e \quad (4)$$

حل: گزینه ۲ جواب است.

به طور نمونه اگر $i=2, j=1$ را بررسی کنیم، تنها گزینه ۲ جواب صحیح را می دهد. ($e = n \times e$)



-۱۰- تعداد عناصر غیر صفر یک آرایه اسپارس سه قطری $n \times n$ چقدر است؟ (مهندسی IT- آزاد ۸۸)

$$3n+3 \quad (4)$$

$$3n-2 \quad (3)$$

$$3n+2 \quad (2)$$

$$3n-3 \quad (1)$$

حل: گزینه ۳ درست است.

n عنصر در قطر اصلی و $n-1$ عنصر در بالای قطر اصلی و $n-1$ عنصر در پایین قطر اصلی.

$$n + (n - 1) + (n - 1) = 3n - 2$$



۱۱- ماتریس سه قطري، ماتریس مربعی است که در آن به جز عناصر قطر اصلی و قطر بالا و پائین آن، سایر عناصر برابر صفر می باشند. عناصر غیر صفر را به صورت سطري در يك آرایه يك بعدی ذخیره نماییم، فرمول محل ذخیره سازی چیست؟ (بفرض ابعاد ماتریس $n \times n$ باشد و آدرس شروع آرایه در حافظه α است).

(مهندسی کامپیوتر - آزاد ۸۵)

$$\alpha + 2j + i - 3 \quad (2)$$

$$\alpha + 2i + j - 3 \quad (1)$$

$$\alpha + 2i + j + 3 \quad (4)$$

$$\alpha + i + 2j + 3 \quad (3)$$

حل: گزینه ۱ درست است.

اگر عناصر غیر صفر يك ماتریس سه قطري را به صورت سطري در يك آرایه يك بعدی ذخیره نماییم، فرمول محل ذخیره سازی برابر $\alpha + 2i + j - 3$ می باشد.

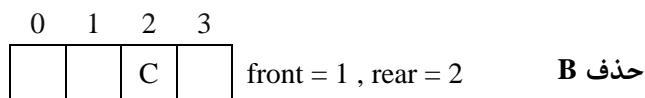
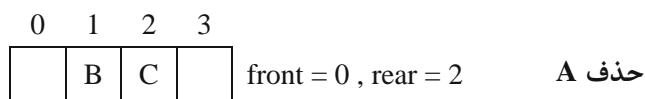
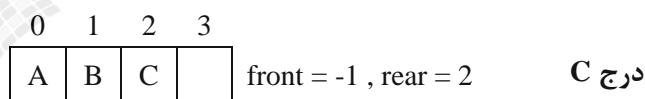
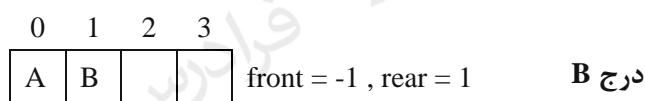
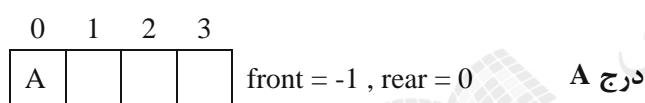
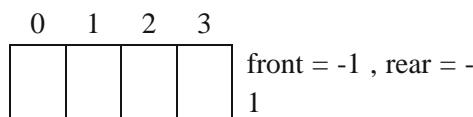


فصل ۴: صف و پشته

صف (queue)، ساختمان داده‌ای است که عمل حذف از ابتدای آن و درج به انتهای آن انجام می‌شود. صف را لیست (First In First Out) FIFO می‌نامند، زیرا اولین عنصر وارد شده به صف، اولین عنصری است که خارج می‌شود. برای نمایش صف، از آرایه یک بعدی $[0..n-1]$ queue و دو متغیر front و rear استفاده می‌شود. در متغیر front یکی کمتر از مقدار محل عنصر اول صف و در متغیر rear محل آخرین عنصر صف ذخیره می‌شود.

مثال

یک صف با 4 خانه که در ابتدای خالی است مفروض می‌باشد. ($front = -1$, $rear = -1$)



در صف خالی مقدار front با rear برابر است و در صف پر مقدار rear برابر $n-1$ می‌باشد.

درج و حذف در صف

تابع‌های درج و حذف در صف در زیر آورده شده است:

اضافه کردن عنصر به صف

```
void addq (int *rear , int item){
    if (*rear == max -1)
    {
        queue-full( );
        return;
    }
    queue[++*rear] = item;
}
```

(max : حداکثر اندازه صف می باشد).

حذف عنصر از صف

```
int deleteq (int *front , int rear){
    if (*front == rear)
        return queue-empty( );
    return
        q[++*front];
}
```

این نوع نمایش ترتیبی صف، دارای نقاط ابهامی است. با ورودی و خروج داده‌ها، صف بتدريج بطرف راست تغییر مکان می‌دهد. به نحوی که rear برابر max-1 می‌شود و به نظر می‌رسد که صف پر است. در این حالت، queue-full() باید تمام صف را به سمت چپ شيفت دهد و اولين عنصر دوباره در موقعیت queue[0] قرار گرفته و front برابر 1- شود. rear نيز باید تنظیم شود. ولی شيفت عملی زمان گير است.

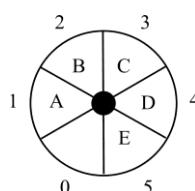
صف حلقوی

در صف ساده مشکلاتی داشتیم. اگر آرایه را حلقوی فرض کنیم، اندیس ابتدا همیشه به یک موقعیت عقب تر (در خلاف حرکت عقربه‌های ساعت) از اولین عنصر موجود در صف اشاره می‌کند. اندیس انتهای (rear) به انتهای فعلی صف اشاره می‌کند. اگر front=rear باشد، صف خالی خواهد بود.

در یک صف حلقوی به اندازه max حداکثر 1 عنصر می‌تواند، قرار گیرد. اگر از آن یک خانه نیز استفاده شود، front=rear می‌شود و نمی‌توانیم یک صف پر و خالی را از هم تشخیص دهیم.

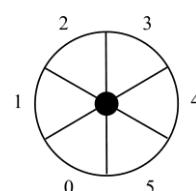
مثال

حالتهای خالی و پر در یک صف حلقوی در زیر نمایش داده شده است:



صف پر

front = 0
rear = 5



صف خالی

front = 0
rear = 0

 اگر شرط $\text{front} = \text{rear}$ در صف حلقوی برقرار باشد، آنگاه صف حلقوی خالی است.

 اگر شرط $(\text{rear}+1) \bmod n = \text{front}$ در صف حلقوی برقرار باشد، آنگاه صف حلقوی پر است.

 اگر شرط $(n-\text{front}+\text{rear}) \bmod n = n-1$ در صف حلقوی برقرار باشد، آنگاه صف حلقوی پر است.

 تعداد عناصر صف حلقوی برابر $n-(\text{front}+\text{rear}) \bmod n$ است که می‌توان به صورت زیر نیز بیان کرد:

اگر $\text{front} \leq \text{rear}$

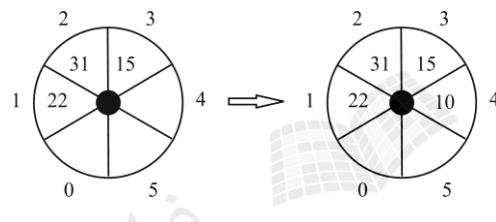
و اگر $\text{front} > \text{rear}$

درج و حذف در صف حلقوی

برای اضافه کردن یک عنصر لازم است که rear در جهت عقربه‌های ساعت حرکت کند و اگر با front برابر شد آنگاه صف پر است و توسط تابع full مقدار rear به مقدار قبلی برگردانده می‌شود و پیغام خطای چاپ می‌شود.

مثال

اضافه کردن عدد 10 به صف حلقوی زیر:



$\text{front} = 0$
 $\text{rear} = 3$

$\text{front} = 0$
 $\text{rear} = 4$

```
addq ( $\text{front}$  ,  $\text{*rear}$  , item)
{
     $\text{*rear} = (\text{*rear}+1) \% \text{max};$ 
    if ( $\text{*rear} == \text{front}$ )
    {
        queue-full( $\text{rear}$ );
        return;
    }
    queue[ $\text{*rear}$ ] = item ;
}
```

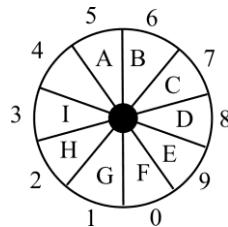
```
deleteq ( $\text{*front}$  ,  $\text{rear}$ )
{
    if ( $\text{*front} == \text{rear}$ )
        return queue-empty();
     $\text{*front} = (\text{*front}+1) \% \text{max};$ 
    return queue[ $\text{*front}$ ];
}
```

تذکر مهم: پیاده سازی توابع queue-full و queue-empty بسته به کاربردهای خاص دارد. اگر هدف ادامه پردازش باشد و سپس یک عنصر را حذف کنیم، queue-full باید اشاره گر rear را در موقعیت قبلی ذخیره کند. queue-empty باید یک عنصر با یک کلید خطای توسعه برنامه اصلی آن را تست می‌کند، برگرداند.

مثال

یک صف حلقوی با ۱۰ خانه و $front=3$ ، $rear=4$ ، چه وضعیتی دارد؟

حل: صف پر است، چون مقدار $n \% (rear+1)$ برابر $front$ است. شکل زیر مثالی از این حالت است:



پشته (STACK)

In First Out به معنی LIFO می‌نامند، چون آخرین عنصر وارد شده در آن، اولین عنصری است که از آن برداشته می‌شود. ساده‌ترین روش پیاده سازی پشته، استفاده از یک آرایه یک بعدی به نام stack[MAX] است که MAX، حداقل تعداد عناصر آرایه می‌باشد. همچنین از یک متغیر به نام top که به عنصر بالایی پشته اشاره می‌کند، نیاز است. اولین یا پایین‌ترین عنصر پشته در stack[0] ذخیره می‌شود. در ابتدا به top مقدار -1- داده می‌شود که نشان‌دهنده یک پشته خالی است.

عملیات درج در پشته

```
void push(int item, int *top){
    if (*top >= max-1)
    {
        stack-full();
        return;
    }
    stack[++*top] = item;
}
```

در این تابع اگر پشته پر نباشد، top را افزایش داده و item در آرایه stack اضافه می‌شود.

عملیات حذف از پشته

```
int pop(int *top){
    if (*top == -1)
        return stack-empty();
    return stack[(*top)--];
}
```

در این تابع اگر پشته خالی نباشد، عنصر بالای پشته برگردانده شده و مقدار top یک واحد کاهش می‌یابد.

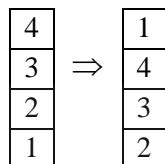
برای معکوس کردن محتویات صف از یک پشته کمک می‌گیریم. ابتدا تمامی عناصر صف را حذف کرده و به پشته اضافه می‌کنیم، سپس تمامی عناصر پشته را حذف کرده و به صف بر می‌گردانیم. برای معکوس کردن ترتیب عناصر یک پشته از یک صف اضافی کمک می‌گیریم.

برای معکوس کردن محتویات صف از یک پشته کمک می‌گیریم. ابتدا تمامی عناصر صف را حذف کرده و به پشته اضافه می‌کنیم، سپس تمامی عناصر پشته را حذف کرده و به صف بر می‌گردانیم. برای معکوس کردن ترتیب عناصر یک پشته از یک صف اضافی کمک می‌گیریم.



مثال

اگر بخواهیم عنصر پایین پشته زیر را بر روی عناصر پشته قرار دهیم، به چند push نیاز داریم؟



پاسخ:

مراحل کار به صورت زیر است:

- ۱- سه عدد بالای پشته را pop کرده و در یک پشته کمکی push می‌کنیم.
- ۲- عدد ۱ را pop کرده و در یک متغیر کمکی ذخیره می‌کنیم.
- ۳- سه عدد موجود در پشته کمکی را pop کرده و در پشته اصلی push می‌کنیم.
- ۴- مقدار موجود در متغیر کمکی (یعنی ۱) را در پشته اصلی push می‌کنیم.

بنابراین به هفت عدد push نیاز است.

مثال

برای انتقال عناصر از پشته ۱، به همان ترتیب، به پشته ۲ نیاز است؟

پاسخ:

ابتدا با n تا pop عناصر را از پشته اول برداشته و با n تا push در پشته کمکی قرار می‌دهیم. سپس عناصر از پشته کمکی را با n تا pop برداشته و با n تا push آنها را در پشته دوم درج می‌کنیم. بنابراین در کل به $4n$ عمل push و pop نیاز داریم.



تعداد خروجی‌های مجاز از یک پشته n تایی برابر است با $\frac{\binom{2n}{n}}{(n+1)}$.



مثال

در ورودی یک پشته اعداد ۱ تا n به ترتیب قرار دارند(۱ در ابتدای ورودی است). عمل push و pop به صورت زیر تعریف شده اند.

Push : اولین عدد ورودی را برداشته و در بالای پشته قرار می‌دهد.

Pop : عدد بالای پشته را برداشته و در انتهای دنباله خروجی می‌نویسد.

با ترکیب مناسبی از n عدد pop می‌توان جایگشتی از اعداد ۱ تا n را در خروجی تولید کرد که به آن جایگشت قابل قبول می‌گوییم. آیا برای $n=8$ ، جایگشت $<4,3,7,8,6,2,5,1>$ قابل قبول است؟ پاسخ: خیر - چون نمی‌توان ۲ را قبل از ۵، pop کرد. (بالای ۲ در پشته قرار دارد).

راه تشخیص سریع: برای هر عدد x ، اعداد کوچکتر از x که بعد از آن قرار دارند، باید یک دنباله نزولی باشند. در دنباله داده شده، اعداد بعد از ۶ که $<2,5,1>$ هستند، یک دنباله نزولی نمی‌باشند.



مثال

در صورتی که اعداد ۱ و ۲ و ۳ به ترتیب از راست به چپ وارد یک پشته شوند، چند حالت خروجی می‌تواند ایجاد شود؟

$$\text{حل: با قرار دادن } n=3 \text{ در رابطه } \frac{\binom{2n}{n}}{(n+1)} \text{ جواب برابر 5 خواهد شد.}$$

این پنج حالت به صورت زیر می‌باشد:

123 (ابتدا ۱ را در پشته قرار داده و سپس آن را خارج کرده و همین کار را برای ۲ و ۳ نیز انجام می‌دهیم.)

132 (ابتدا ۱ را در پشته قرار داده و سپس آن را بر می‌داریم، بعد عدد ۲ و ۳ را در پشته قرار داده و سپس هر دو را خارج می‌کنیم.)

213 (عدد ۱ و ۲ را در پشته قرار داده و سپس هر دو را خارج کرده و ۳ را در پشته قرار داده و خارج می‌کنیم.)

231 (عدد ۱ و ۲ را در پشته قرار داده، عدد ۲ را خارج کرده و عدد ۳ را قرار داده و عدد ۱ را خارج می‌کنیم.)

321 (عدد ۱ و ۲ را وارد کرده و سپس هر سه عدد را خارج می‌کنیم.)

تذکر: توجه کنید که خروجی 312 ممکن نمی‌باشد. چون ابتدا ۱ و ۲ و ۳ را در پشته قرار داده و سپس ۳ را خارج کرده و بعد از آن نمی‌توان ۱ را خارج کرد چون قبل از آن باید عدد ۲ را خارج کرد.



کاربردهای پشته

از کاربردهای پشته می‌توان موارد زیر را نام برد:

۱- ارزشیابی عبارات

۲- زیر برنامه‌های بازگشته

۳- نگهداری آدرس برگشت زیر برنامه‌ها

ارزشیابی عبارات

یکی از کاربردهای پشته، ارزشیابی عبارات می‌باشد. یک عبارت از عملوندها و عملگرها ساخته شده که می‌تواند به سه شکل نمایش داده شود:

(۱) prefix (پیشوندی) (۲) infix (میانندی) (۳) postfix (پسوندی)

در فرم infix، عملگرها بین عملوندها قرار می‌گیرند. در فرم postfix، عملگرها بعد از عملوندها قرار می‌گیرند و در فرم prefix، عملگرها قبل از عملوندها قرار می‌گیرند. به طور نمونه عبارت A^*B/C میانندی، عبارت AB^*C پسوندی و عبارت $ABC/*$ پیشوندی می‌باشد.

تبدیل فرم‌ها به یکدیگر

حالتهای ممکن تبدیل فرم به صورت زیر می‌باشد:

postfix به infix -۱

prefix به infix -۲

infix به postfix -۳

infix به prefix -۴

prefix به postfix -۵

postfix به prefix -۶

تبدیل از فرم postfix به infix

ابتدا عبارت را بر حسب اولویت عملگرهایش در نظر گرفته و سپس هر عملگر را بعد از عملوند‌هایش می‌نویسیم. به طور نمونه عبارت $A+B$ را به صورت $AB+$ نمایش می‌دهیم.

مثال

عبارت میانندی $A/B-C+D^*E$ را به فرم پسوندی تبدیل کنید.(اولویت: / ، * ، - ، + و)

$A/B-C+D^*E$

$AB/- C + D^*E$ تقسیم :

$AB/- C + DE^*$ ضرب :

$AB/C- + DE^*$ منها :

$AB/C-DE^*+$ جمع :

تذکر: تقسیم و ضرب دارای اولویت یکسانی می‌باشند و هر کدام که از چپ به راست، زودتر استفاده شود، قدم بیشتری خواهد داشت.(برای جمع و تفریق هم چنین است)

مثال

عبارت $((A+B)^*D)^(E-F)$ را به فرم پسوندی تبدیل کنید.

پاسخ:

$$\begin{aligned}
 & (\mathbf{AB}+ *D) ^\wedge (\mathbf{E}-\mathbf{F}) \\
 \Rightarrow & \mathbf{AB}+\mathbf{D}* ^\wedge (\mathbf{E}-\mathbf{F}) \\
 \Rightarrow & \mathbf{AB}+\mathbf{D}^*^\wedge \mathbf{E}\mathbf{F}- \\
 \Rightarrow & \mathbf{AB}+\mathbf{D}^*\mathbf{E}\mathbf{F}-^\wedge
 \end{aligned}$$



تبدیل از فرم infix به prefix

در این تبدیل عملگرها را با توجه به اولویت آنها به سمت چپ منتقل می‌کنیم. به طور نمونه عبارت میانوندی $A+B$ را به فرم $+AB$ نمایش می‌دهیم.

مثال

عبارت $x+y^*z-w$ را به فرم پیشوندی تبدیل نمایید.

$$x+ *yz -w \Rightarrow +x*yz - w \Rightarrow -+x*yzw$$

مثال

عبارت $a+(b-c*d)^e-f^g(h/ i*k)$ را به فرم پیشوندی و پسوندی تبدیل نمایید.

پاسخ: عملگر توان سمت راست، زودتر اجرا می‌شود.

$$-+a^-b*cde^f^g*/hik \quad \text{و} \quad abcd*-e^fghi/k^{^88-}$$



تبدیل از فرم infix به postfix

این تبدیل به کمک پشته و یا روش استفاده شده در مثال زیر قابل انجام می‌باشد.

مثال

تبدیل عبارت پسوندی ABC^*+ به فرم میانوندی :

$$\mathbf{AB}^*\mathbf{C}+ \Rightarrow \mathbf{A}+\mathbf{B}^*\mathbf{C}$$

**مثال**

حاصل عبارت محاسباتی $-,* ,/ ,8,3,9,-,14,6,1+$ کدام است؟

پاسخ:

14 , 6 , 1 , + , / , 8 , 3 , 9 , - , * , -

14 , 7 , / , 8 , 3 , 9 , - , * , -

2 , 8 , 3 , 9 , - , * , -

2 , 8 , -6 , * , -

2 , -48 , -

50

**مثال**

اگر عبارتی، $-AB/C-DE^*+AC^*$ باشد، چند جفت پرانتز در عبارت infix آن قرار دهیم تا postfix آن به صورت، $ABC-D+/EA-*C^*$ شود؟

پاسخ: حاصل infix عبارت داده شده برابر $A/B-C+D^*E-A^*C$ است که با قرار دادن ۲ پرانتز به صورت $\boxed{ABC-D+/EA-*C^*}$ در می‌آید. postfix آن به صورت $C+D)^*(E-A)^*C$ است.

تبدیل از فرم infix به prefix

این تبدیل را نیز می‌توان به کمک پشته و یا روش استفاده شده در مثال زیر انجام داد.

مثال

عبارت infix زیر را به prefix تبدیل کنید؟

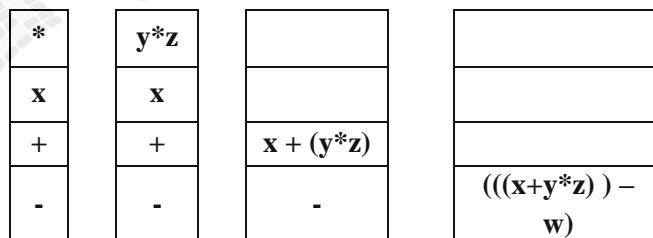
$+-*\uparrow ABCD / E / F + GH$

پاسخ:

$+ - * A^B CD/E/F +GH$
 $+ - * A^B CD/E/F G+H$
 $+- A^B*C D/E/F G+H$
 $+-A^B*C D F/(G+H)$
 $+-A^B*C D E/(F/(G+H))$
 $+ A^B*C-D E/(F/(G+H))$
 $\boxed{A^B*C-D + E/(F/(G+H))}$

مثال

عبارت پیشوندی $-+xyzw$ به فرم میانوندی تبدیل نمایید. (به کمک پشته)



پاسخ:

برای تبدیل به کمک پشته، عملگرهای عبارت را از چپ به راست در پشته قرار داده تا به دو عملوند برسیم . در این حالت عملگر بالای پشته را بر روی آنها اعمال کرده و نتیجه را در پشته قرار می‌دهیم. اگر بعد از عملوندی، یک عملگر بود آن را در پشته قرار می‌دهیم. اگر در حالتی در بالای پشته دو عبارت قرار داشت که قبل از آنها عملگری بود، ابتدا عملگر را بر روی آن دو عبارت اعمال کرده و سپس ورودی‌های بعدی را پردازش می‌کنیم.

تبدیل از فرم Prefix به Postfix

برای این تبدیل ابتدا عبارت postfix را به infix تبدیل کرده و سپس حاصل را به prefix تبدیل می‌کنیم. البته می‌توان مستقیماً نیز این تبدیل را انجام داد.

مثال

عبارت پسوندی $ABC \wedge D^*E^+$ را به فرم پیشوندی تبدیل نمایید.

پاسخ: ابتدا عبارت را به فرم میانوندی تبدیل می‌کنیم:

$$ABC \wedge D^*E^+ \Rightarrow AB^*C/D^*E^+ \Rightarrow A/B^*C D^*E^+ \Rightarrow A/B^*C^* DE^+ \Rightarrow A/B^*C^* D+E$$

حال عبارت میانوندی حاصل را به فرم پیشوندی تبدیل می‌کنیم:

$$A/B^*C^*D+E \Rightarrow A/\wedge BC^*D+E \Rightarrow /A^*BC^*D+E \Rightarrow +*/A^*BCDE$$



مثال

معادل پیشوندی عبارت AB/C^*D^+ چیست؟ (بدون تبدیل به عبارت میانوندی)

پاسخ: می‌توان عبارت پسوندی را مستقیماً به عبارت پیشوندی تبدیل کرد. یعنی عملگر بعد از عملوندها را به قبل از آنها برد.

به طور نمونه عبارت $/AB$ را به فرم AB نمایش داد.

$$AB/C^*D^+ \Rightarrow /AB C^*D^+ D^+ */ABC \Rightarrow \Rightarrow +*/ABCD$$



تبدیل از فرم Postfix به Prefix

ابتدا عبارت infix را به prefix تبدیل کرده و سپس عبارت میانوندی حاصل را به Postfix تبدیل می‌کنیم.

مثال

عبارت پیشوندی $/*+ABC-DE$ به فرم پسوندی تبدیل نمایید.

پاسخ: ابتدا عبارت را به فرم میانوندی تبدیل می‌کنیم:

$$\begin{aligned} /*+ABC-DE &\Rightarrow /*(A+B)C-DE \Rightarrow /((A+B)^*C)-DE \\ &\Rightarrow /((A+B)^*C)(D-E) \Rightarrow ((A+B)^*C)/(D-E) \end{aligned}$$

حال عبارت میانوندی حاصل را به فرم پسوندی تبدیل می‌کنیم:

$$\begin{aligned} ((A+B)^*C) / (D-E) E - *C) / (DAB+(\Rightarrow \\ \Rightarrow AB+C^* / (D-E) \Rightarrow AB+C^* / DE- /-AB+C^*DE \Rightarrow \end{aligned}$$



مثال

عبارت پیشوندی $ABCD \wedge EF-GH$ را به فرم پسوندی تبدیل نمایید.

پاسخ: می‌توان مستقیماً و بدون استفاده از فرم میانوندی، این تبدیل را انجام داد. کافی است عملگر قبل از عملوندها را به

بعد از آنها برد. به طور نمونه عبارت AB را به فرم AB نمایش داد.

$$\begin{aligned} - + */ABCD \wedge EF-GH &\Rightarrow - + *AB/CD \wedge EF-GH \Rightarrow - + AB/C^*D \wedge EF-GH \Rightarrow \\ \Rightarrow - AB/C^*D^+ \wedge EF-GH &\Rightarrow - AB/C^*D^+ \wedge EF/GH \Rightarrow \\ \Rightarrow - AB/C^*D^+ EF/GH^- &\Rightarrow AB/C^*D^+ EF/GH^- \end{aligned}$$



الگوریتم تبدیل عبارت infix به postfix توسط پشته

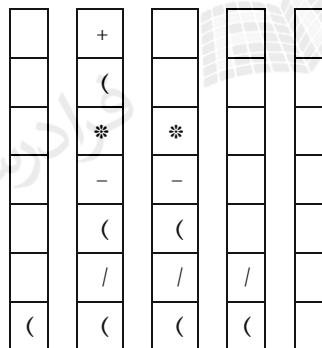
عبارت داده شده میانوندی را Q و عبارت حاصل در فرم پسوندی را P در نظر می‌گیریم. ابتدا یک پرانتز باز در پشته push کرده و یک پرانتز بسته به انتهای عبارت Q اضافه می‌کنیم و عبارت Q را از چپ به راست پیمایش کرده و عملیات زیر را انجام می‌دهیم:

- ۱- با برخورد به عملوند، آن را به عبارت P اضافه می‌کنیم.
- ۲- با برخورد به پرانتز باز، آن را در پشته push می‌نماییم.
- ۳- با برخورد به عملگر، آن را در پشته push می‌نماییم. البته اگر به عملگری برسیم که اولویت عملگر بالای پشته از این عملگر بیشتر یا مساوی بود، ابتدا عملگر بالای پشته را pop کرده و به P اضافه می‌کنیم و سپس عملگر مورد نظر را در پشته push می‌نماییم.
- ۴- با برخورد به پرانتز بسته، عملگرهای بالای پشته را pop کرده و به P اضافه می‌نماییم و پرانتز باز داخل پشته را حذف می‌نماییم.
- ۵- در نهایت با برخورد به پرانتزی که در ابتدا به انتهای Q اضافه کرده بودیم، عملگر و پرانتز موجود در پشته را pop کرده و عملیات پایان می‌یابد.(پشته خالی است)

مثال

عبارت میانوندی $(B-C^*(D+E)) / A$ را به کمک پشته به معادل پسوندی تبدیل نمایید.

پاسخ: ابتدا یک پرانتز باز در پشته push کرده و یک پرانتز بسته به آخر عبارت میانوندی اضافه می‌کنیم و طبق الگوریتم بالا عملیات را انجام می‌دهیم:



مقدار عبارت P در هر یک از حالت‌ها در زیر نشان داده شده است:

$$P = ABCDE$$

$$P = ABCDE+$$

$$P = ABCDE+*-$$

$$P = ABCDE+*-/$$

الگوریتم محاسبه یک عبارت به فرم Postfix توسط پشته

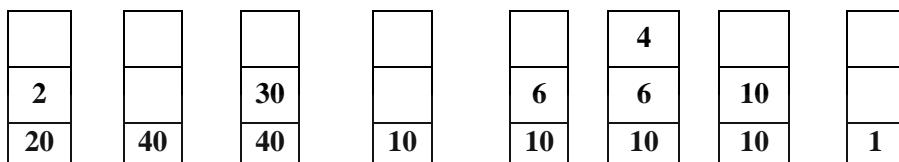
برای محاسبه یک عبارت که به فرم postfix داده شده، ابتدا یک پرانتز بسته نگهبان در انتهای عبارت اضافه می‌کنیم و به کمک پشته ارزیابی را انجام می‌دهیم: هنگامی که به پرانتز بسته در عبارت برسیم، نتیجه نهایی در پشته موجود است.

مثال

حاصل عبارت $/20,2,*30,-6,4,+$ را بدست آورید. (به کمک پشته)

پاسخ:

در ابتدای یک پرانتز بسته به انتهای عبارت داده شده اضافه می‌کنیم. سپس مقدار 20 و 2 را در پشته push کرده و با رسیدن به عملگر ضرب، آن دو را از پشته pop کرده و نتیجه اعمال ضرب بر روی آنها، یعنی 40 را در پشته push می‌کنیم و عملیات را به همین نحوه ادامه داده تا به پرانتز بسته در انتهای عبارت برسیم. در این حالت نتیجه در پشته خواهد بود.



در نهایت با 8 عدد push و 8 عدد pop به نتیجه نهایی رسیدیم. (البته push نتیجه نهایی در پشته را در نظر نگرفتیم.)

تعداد pop ها برای تبدیل یک عبارت infix به فرم postfix، دو برابر تعداد عملگرها می‌باشد.

تعداد push ها برای تبدیل یک عبارت postfix به فرم infix، دو برابر تعداد عملگرها می‌باشد.

کاربرد پشته در زیر برنامه های بازگشتی

توابع بازگشتی قبلاب بررسی شد. در این قسمت نحوه استفاده توابع بازگشتی از پشته را نشان می‌دهیم.

مثال

خروجی زیر برنامه زیر، به ازای فراخوانی `f(1)` چیست؟

```
f (int x){  
    if (x==3) exit();  
    else{  
        x=x+1;  
        f(x);  
        cout<<x;  
        cout<< 'a';  
    }  
}
```

پاسخ: ترتیب فراخوانی ها به صورت $f(1) \Rightarrow f(2) \Rightarrow f(3)$ بوده و از آنجا که دستورات بعد از فراخوانی بازگشتی (دستورات چاپ در این مثال)، در پشته قرار می‌گیرند داریم:

cout<<3
cout<<'a'
cout<<2
cout<<'a'

بنابراین خروجی برابر $3a2a$ خواهد بود.

مثال

خروجی زیر برنامه زیر، به ازای فراخوانی $f(1,4,7)$ چیست؟

```
f( int x , int y , int z ){
    if (x==3) exit( );
    else{
        x++; y++; z++;
        f(x,y,z);
        cout<< x;
        cout<< y;
        cout<< z;
    }
}
```

پاسخ: ترتیب فراخوانی‌ها به صورت مقابل است: $f(1,4,7) \Rightarrow f(2,5,8) \Rightarrow f(3,6,9)$

دستورات بعد از فراخوانی بازگشتی (دستورات چاپ در این مثال)، در پشتیه قرار می‌گیرند:

cout<< 3
cout<< 6
cout<< 9
cout<< 2
cout<< 5
cout<< 8

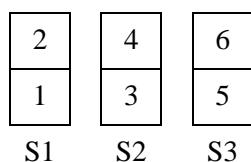
بنابراین خروجی برابر 369258 خواهد بود.



کنکور ارشد

(مهندسی کامپیوتر - آزاد ۸۰)

۱- سه پشته S_1 و S_2 و S_3 هر یک حاوی دو عدد بشكل زیر می باشند.



دو عملگر $\text{Pop}(i, j)$ و $\text{PopPush}(i, j)$ بصورت زیر تعریف شده اند.

$\text{PopPush}(i, j)$: یک قلم از پشته S_i حذف و به پشته S_j اضافه می کند.

$\text{Pop}(i)$: یک قلم از پشته S_i حذف و سپس آن را چاپ می کند.

برای چاپ اعداد ۱ تا ۶ بصورت ۱, ۳, ۵, ۲, ۴, ۶ با استفاده از PopPush بایستی حداقل چند بار مورد استفاده قرار گیرد؟

۴ (۴)

۶ (۳)

۵ (۲)

۳ (۱)

حل: گزینه ۴ درست است. عملیات لازم عبارتند از:

۱- $\text{PopPush}(1, 3)$

۲- $\text{Pop}(1)$

۳- $\text{PopPush}(2, 1)$

۴- $\text{Pop}(2)$

۵- $\text{PopPush}(3, 1)$

۶- $\text{PopPush}(3, 2)$

۷- $\text{Pop}(3)$

۸- $\text{Pop}(1)$

۹- $\text{Pop}(1)$

10- $\text{Pop}(2)$

(مهندسی کامپیوتر - آزاد ۹۰)

۲- تعداد دنباله های مجاز خروجی با k ورودی از یک پشته (stack) برابر با کدام گزینه است؟

$$\frac{1}{k-1} \binom{2k}{k} \quad (4) \qquad 2k+1 \quad (3) \qquad \frac{1}{k} \binom{2k}{k} \quad (2) \qquad \frac{1}{k+1} \binom{2k}{k} \quad (1)$$

حل: جواب گزینه ۱ است.



(مهندسی کامپیوتر - دولتی ۷۸)

۳- بر روی پشته S اعمال زیر قابل انجام است. اگر n عمل از اعمال فوق به ترتیب دلخواه، بر روی پشته S که در ابتدای تهی است انجام شود، مجموع هزینه این n عمل در بدترین حالت کدام است؟

: درج x در بالای پشته با هزینه $O(1)$.

: حذف عنصر بالای پشته با هزینه $O(1)$.

: حذف k عنصر بالای پشته با هزینه $O(k)$.

(k) حداکثر برابر تعداد عناصر موجود پشته است).

$$o(n) \quad (4)$$

$$o(nk) \quad (3)$$

$$o(n^2) \quad (2)$$

$$o(n \cdot \log n) \quad (1)$$

حل: جواب گزینه ۴ است. چون هر عنصر به طور دقیق یکبار درج و مستقل از نحوه حذف شدنش توسط pop یا multipop می‌شود، به ازای هر عنصر، ۲ واحد هزینه پرداخت می‌شود. در نتیجه، در مجموع هزینه کل اعمال حداکثر برابر $2n$ خواهد بود.



(علوم کامپیوتر - دولتی ۸۰)

- عبارت psotfix معادل عبارت $(A + B)^* D + E / (F + A^* D)$ با.....

$$AB + D^* EFAD^* +/+ \quad (2)$$

$$AB + DE + FAD^* +/+ \quad (4)$$

$$AB + D^* E + F / A + D^* \quad (1)$$

$$ABDEFAD^* +^* / +^* \quad (3)$$

حل: جواب گزینه ۲ است.

$$(A+B)^* D + E / (F + A^* D)$$

$$\Rightarrow AB +^* D + E / (F + A^* D)$$

$$\Rightarrow AB +^* D + E / FAD^* +$$

$$\Rightarrow AB + D^* + E / FAD^* +$$

$$\Rightarrow AB + D^* + EFAD^* + /$$

$$\Rightarrow AB + D^* EFAD^* + / +$$



(مهندسی IT - دولتی ۸۶)

- معادل infix عبارت prefix روبرو چیست؟

$$/*A-B/*C-+DEFG-HI$$

$$((C+E-F)^* C / G - B)^* A / H - I \quad (2)$$

$$A^* (B / (((C^* (D+E)) - F - G)) / (H-I)) \quad (4)$$

$$A^* B - C^* D + E - F / G / H - I \quad (1)$$

$$A^* (B - C^* (D+E-F) / G) / (H-I) \quad (3)$$

حل: گزینه ۳ درست است.

$$/*A-B/*C-+DEFG-HI$$

$$/*A-B/*C-(D+E)FG(H-I)$$

$$/*A-B/*C((D+E)-F)G(H-I)$$

$$/*A-B/(C*((D+E)-F))G(H-I)$$

$$/*A-B(((C*((D+E)-F))/G)(H-I)$$

$$/*A(B-((C*((D+E)-F))/G)(H-I)$$

$$/A^* ((B-((C*((D+E)-F))/G))(H-I)$$

$$(A^* ((B-((C*((D+E)-F))/G))) / (H-I)$$

روش سریع: قبل از DE عملگر + قرار دارد و به (D+E) تبدیل می‌شود که تنها در گزینه ۴ این عبارت وجود دارد.



(مهندسی کامپیوتر - آزاد ۸۶)

- مقدار ارزیابی عبارت پسوندی $2 \$ 3 + - 3 8 2 / + ^* 2 \$ 3 + 6 2 3 +$ چیست؟ (\$: توان)

$$54 \quad (4)$$

$$48 \quad (3)$$

$$52 \quad (2)$$

$$50 \quad (1)$$

حل: جواب گزینه ۲ است.

$$\begin{aligned}
 & 6\ 2\ 3\ +\ -\ 3\ 8\ 2\ /+\ * 2\$3+ \\
 \Rightarrow & 65 - 34 + * 2\$3+ \\
 \Rightarrow & 1\ 7 * 2\$3+ \\
 \Rightarrow & 72\$3+ \\
 \Rightarrow & 49\ 3+ \\
 \Rightarrow & 52
 \end{aligned}$$



(مهندسی کامپیووتر - دولتی ۸۲)

۷- با توجه به دوتابع زیر، خروجی $f1(4)$ چیست؟

```

void f1(int x){
    if (x)
        f2(x-1);
    printf(x);
}

void f2(int y){
    if (y) {
        printf(y + 1);
        f1(y - 1);
    }
}

```

42024(۴)

42204(۳)

20244 (۲)

44220 (۱)

حل: گزینه ۴ درست است.

نحوه فراخوانی‌ها به صورت زیر است:

$$f1(4) \Rightarrow f2(3) \Rightarrow f1(2) \Rightarrow f2(1) \Rightarrow f1(0)$$

در ابتدا با صدازدن $f1(4)$ ، تابع $f2$ با مقدار ۳ فراخوانی شده و دستور بعد از فراخوانی $\text{printf}(4)$ در پشته ذخیره می‌شود. سپس در $f2$ ابتدا مقدار ۴ چاپ شده و تابع $f1$ با مقدار ۲ صدا زده می‌شود. در این حالت $f2$ با مقدار یک صدا زده شده و سپس دستور $\text{printf}(2)$ در پشته ذخیره می‌شود. در این حالت $f2$ ابتدا مقدار ۲ را چاپ کرده و سپس $f1$ را با مقدار صفر صدا می‌زند. در $f1$ چون شرط if برقرار نیست، مقدار صفر توسط دستور printf چاپ می‌شود و اجرای برنامه پایان می‌یابد و مقادیر موجود در پشته چاپ می‌شوند.

فصل ۵: لیست پیوندی

لیست پیوندی، ساختمان داده ای پویا است که اشیاء با یک ترتیب خطی در آن قرار گرفته اند. بر خلاف آرایه، که در آن ترتیب خطی توسط اندیسه‌های آرایه تعیین می‌شود، ترتیب در لیست پیوندی بوسیله یک اشاره گر در هر شیء تعیین می‌گردد. لیست پیوندی بر دو نوع یک طرفه و دو طرفه می‌باشد.

لیست پیوندی یک طرفه(یک سویه)

لیستی که در آن، هر عنصر فقط آدرس عنصر بعدی را نگهداری می‌کند. هر یک از گرههای این لیست پیوندی از دو قسمت داده و آدرس تشکیل شده که در زبان C به صورت زیر تعریف می‌شود:

```
typedef struct list-node *list-pointer;
typedef struct list-node{
    int      data;
    list-pointer next;
};
```

در next، آدرس گره بعدی ذخیره می‌شود. (به جای next از link نیز استفاده می‌شود)

یادآوری: تعریف ساختار یک گره لیست پیوندی یک طرفه در زبان پاسکال:

```
type pointer=^ node;
node = record
    data : char;
    next : ^ pointer;
end;
```

تذکر: برای دسترسی به فیلد داده گره اول لیست پیوندی در زبان پاسکال از دستور first^.data و در زبان C از دستور first->data استفاده می‌کنیم.

درج و حذف در لیست پیوندی نسبت به آرایه ساده‌تر است. چون نیاز به جابجایی فیزیکی عناصر در حافظه نمی‌باشد.

دسترسی به عناصر آرایه نسبت به لیست پیوندی سریع‌تر است، چون اندیس‌های هر عنصر در آرایه مشخص است.

امکان جستجوی دودویی در لیست پیوندی وجود ندارد.

در زبان پاسکال برای تخصیص حافظه پویا از دستور new استفاده می‌شود و برای آزاد سازی فضای دستور dispose استفاده می‌شود. معادل این دستورات در زبان C عبارتند از malloc و free.

الگوریتم های کار بر روی لیست پیوندی یکطرفه

درج یک گره

تابع زیر یک گره با داده d بعد از یک گره با آدرس مشخص n درج می کند:

```
void insert( list-pointer *first , list-pointer n){
    list-pointer t;
    t = malloc(sizeof(list-node));
    t->data = d;
    if (*first) {
        tnext;->next = n ->
        n-> next = t;
    }
    else{
        t ->next = NULL;
        *first = t;
    }
}
```

اگر لیست تهی باشد، گره اضافه شده، تنها گره لیست خواهد بود.

تذکر: قبل از p ، از ستاره استفاده نشده است چون این آدرس تغییری نمی کند.

حذف یک گره

تابع delete گره واقع در قبل از گره با آدرس مشخص n را حذف می کند. (t : آدرس گره قبل از گره)

```
void delete( list-pointer *first , n , t )
{
    if (t)
        t -> next = n ->next;
    else
        *first = (*first) ->next;
    free(n);}
```

تذکر: اگر t برابر NULL باشد، آنگاه n اولین گره لیست می باشد.

چاپ داده های لیست

تابع زیر داده های یک لیست با آدرس شروع p را چاپ می کند.

```
void print-list( list-pointer *p )
{
    for( ; p ; p = p ->next )
        cout << p ->data;
}
```

اتصال دو لیست پیوندی به یکدیگر

تابع زیر لیست p2 را به انتهای لیست p1 متصل می‌کند:

```
list-pointer concatenate (list-pointer p1 , p2)
{
    list-pointer t;
    if (p1==NULL) return p2;
    else{
        if (p2 !=NULL)
        {
            for(t = p1 ; t -> next ; t = t -> next);
            t -> next = p2;
        }
        return p1;
    }
}
```

اگر شرط $p1=NULL$ برقرار باشد، یعنی لیست p1 موجود نیست و نتیجه همان p2 است و در صورت موجود بودن لیست p2 ، لیست p1 را توسط حلقه تا آخر پیمایش کرده و سپس p2 را به انتهای آن توسط دستور $t->next=p2;$ ، اضافه می‌کنیم.

 الگوریتم اتصال دو لیست x و y (x به انتهای y و یا y به انتهای x)، از مرتبه $O(\max(m, n))$ می‌باشد، زیرا در بدترین حالت لیست بزرگتر باید تا انتهای پیمایش شود. ($n \geq m$) طول لیست‌ها می‌باشند)

حذف عناصر تکراری در یک لیست (با شبه کد CLRS)

الگوریتم زیر عناصر تکراری در یک لیست را طوری حذف می کند که ترتیب عناصر باقی مانده تغییر نکند. مثلا لیست $<1,2,3,4>$ به $<1,2,3,2,1,3,4>$ تبدیل می شود.

PurgeList(L)

```

1 p ← First(L)
2 while p<>null
3   do q ← p
4     while next[q]<>null
5       do if element[p]=element[next[q]]
6         then delete-after(L,q)
7       else q ← next[q]
8   p ← next[p]
```

تذکر: در الگوریتم بالا، تابع **delete-after(L,q)**، عنصر بعد از **q** را در لیست **L** حذف می کند. منظور از **next** همان **next** است.

 الگوریتم بالا از مرتبه $O(n^2)$ است. می توان لیست را مرتب کرد و سپس با $O(n)$ کار اضافی، عناصر تکراری را در لیست مرتب حذف کرد. اما لیست نهایی این الگوریتم که عناصرش مرتب هستند، با لیستی که الگوریتم فوق تولید می کند، متفاوت است.

وارون کردن یک لیست پیوندی

می توان یک لیست n عضوی را تنها با تغییر اشاره گرهای آن وارون کرد، به طوری که عنصر i ام لیست بعد از وارون سازی عنصر $n-i+1$ ام لیست جدید شود. این کار را به صورت بازگشتی و غیر بازگشتی می توان انجام داد.

الف - روش غیر بازگشتی

در روش غیر بازگشتی از سه اشاره گر **p**، **q** و **r** استفاده می کنیم، که به سه عنصر متوالی اشاره می کنند. در زمان اجرا **p** و **q** به اولین و دومین عنصر لیستی که تا این مرحله وارون شده اشاره می کنند و **r** به عنصر بعدی در لیست که هنوز وارون نشده است. ابتدا **p** تهی است و **q** به عنصر اول و **r** به عنصر دوم لیست اشاره می کنند. در هر مرحله، مولفه **next** عنصر **r** را به عنصر **q** تغییر داده و هر سه اشاره گر را به جلو می بزیم. با تهی شدن **r**، لیست معکوس شده است. این کار در زمان خطی انجام می شود.

تذکر: الگوریتم زیر با توجه به شبه کد CLRS می‌باشد. منظور از $next[k] > next[c]$ در زبان C می‌باشد.

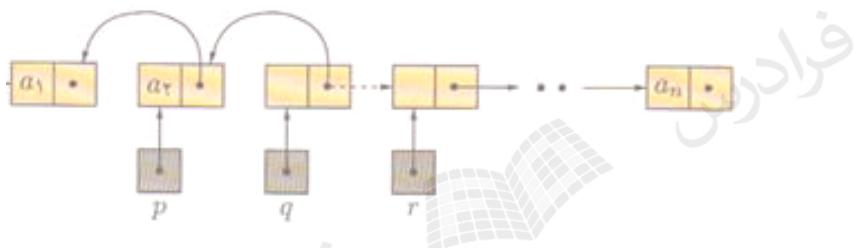
reverse(L)

```

1 if size(L) ≤ 1
2   then return first[L]
3 p ← null
4 q ← First(L)
5 r ← next[q]
6 while r ≠ null
7   do next[q] ← p
8     p ← q
9     q r ←
10    r ← next[r]
11 next[q] ← p
12 return q

```

تذکر: رویه معکوس کردن لیست(بازگشتی یا غیر بازگشتی) از مرتبه $O(n)$ است.



ب- روش بازگشتی

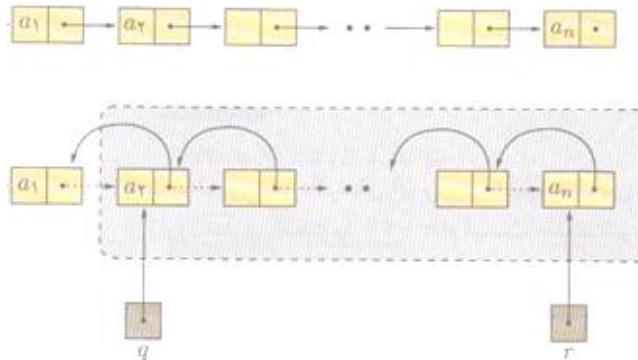
رویه بازگشتی $reverse(L, p)$ ، زیر لیست شامل عناصر p به بعد را در لیست L ، وارون شده بر می‌گرداند. برای وارون کل لیست، p باید برابر $first[L]$ باشد. با فرض اینکه تعداد عناصر از p به بعد در لیست L برابر n باشد، الگوریتم ابتداء عنصر p را کتار می‌گذارد و بقیه لیست با $n-1$ عنصر که با q شروع می‌شود را به صورت بازگشتی وارون می‌کند و از همان عناصر یک لیست r ایجاد می‌کند. در نهایت، عنصر p را به انتهای لیست r وصل می‌کند.

reverse(L,p)

```

1 if p=null or next[p]=null
2   then return p
3 q ← next[p]
4 r ← reverse(L,q)
5 next[q] ← p
6 next[p] ← null
6 return r

```

**مثال**

الگوریتم زیر، لیست L را وارون می‌کند. اگر در دستور if ، قسمت دوم شرط or نباشد، آنگاه خروجی چه خواهد بود؟

reverse(L)

```

1 if L=null or next[L]=null
2   then return L
3 else r ← reverse(next[L])
4   next[next[L]] ← L
5   next[L] ← null
6 return r
    
```

پاسخ: آنگاه return اول همیشه مقدار null بر می‌گرداند و چون این مقدار در r ذخیره شده و دوباره برگردانده می‌شود، بنابراین خروجی همواره null خواهد بود.

پیاده‌سازی پشته با لیست پیوندی

می‌توان پشته را با لیست پیوندی پیاده‌سازی کرد. درج و حذف هر عنصر فقط در ابتدای لیست انجام می‌شود و محدودیتی برای تعداد عناصر پشته نداریم.

اضافه کردن به پشته پیوندی

اضافه کردن یک گره به پشته پیوندی، مشابه اضافه کردن یک گره به اول لیست پیوندی است. آدرس شروع پشته پیوندی برابر top می‌باشد:

```

void add(stack-pointer *top , int a){
    stack-pointer p;
    p = malloc(sizeof(stack) );
    p-> item =a;
    p-> next = *top;
    *top = p;
}
    
```

حذف از پشته پیوندی

حذف یک گره از پشته پیوندی مشابه حذف اولین گره لیست پیوندی می‌باشد:

```
int delete(stack-pointer *top ){
    stack-pointer t;
    int a;
    t =*top;
    if (t ==NULL) exit ( );
    else{
        a = t -> item;
        *top = t -> next;
        free(t);
        return a;
    }
}
```

(CLRS پیادسازی پشته با لیست پیوندی یکطرفه) توسط شبه کد

Push(S,x) 1 n ← Allocate-Node() 2 element[n] ← x 3 next[n] ← top[S] 4 top[S] ← n 5 size[s] ← size[s]+1	Pop(S) 1 if IsEmpty(S) 2 then error STACK IS EMPTY 3 n ← top[S] 4 temp ← element[top[s]] 5 top[s] ← next[top[s]] 6 size[s] ← size[s]-1 7 Free-Object(n) 8 return temp
---	--

پیاده‌سازی صف با لیست پیوندی

برای پیاده سازی صف Q می‌توان از یک لیست پیوندی یک طرفه با دو اشاره گر front[Q] (اشاره گر به عنصر اول لیست) و rear[Q] (اشاره گر به عنصر آخر لیست) استفاده کرد. عمل EnQueue یک عنصر به انتهای لیست درج می‌کند. و عمل DeQueue عنصر اول لیست را حذف می‌کند و در اختیار قرار می‌دهد.

EnQueue (Q,x)	DeQueue(Q,x)
1 next[rear[Q]] ← Allocate-	1 if IsEmpty(Q)
Node(x,null)	2 then error Queue IS EMPTY
2 rear[Q] ← next[rear[Q]]	3 n ← front[Q]
3 size[Q] ← size[Q]+1	4 front[Q] ← next[front[Q]]
	5 temp ← element[n]
	6 Free-Node(n)
	7 size[Q] ← size[Q]-1
	8 return temp

پیاده‌سازی صف با لیست پیوندی یکطرفه (توسط دستورات زبان C)

اولین گره لیست، عنصر ابتدای صف می‌باشد. در واقع صف پیوندی لیستی است که اضافه به انتهای آن و حذف از ابتدای آن انجام می‌گیرد.

addq(*front , *rear, a)	int deleteq(queue-pointer *front)
<pre> { t = malloc(sizeof(queue)); t -> item = a; t -> next = NULL; if (*front) (*rear) -> next = t; else *front = t; *rear = t; } </pre>	<pre> { int a; queue-pointer t; t = *front; if ((*front) == NULL) exit(); a = t -> item; *front = t -> next; free(t); return a; } </pre>

لیست پیوندی دوطرفه

در لیستهای پیوندی دو طرفه (دو گانه یا دو سویه)، هر عنصر علاوه بر مولفه‌های لیست یک طرفه، مولفه prev هم دارد که به عنصر قبلی اش در لیست اشاره می‌کند.

تذکر: به جای Prev از Llink و به جای next از Rlink نیز استفاده می‌شود.
با استفاده از این اشاره‌گرها امکان حرکت به هر دو طرف وجود دارد و بنابراین با داشتن آدرس یک گره، به کلیه گره‌های لیست می‌توان دسترسی پیدا کرد.

شكل زیر یک لیست دو طرفه با چهار گره را نشان می‌دهد:



تعریف یک نود لیست دو طرفه در زبان C :

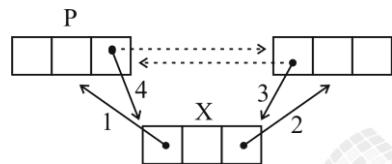
```
typedef struct node *node-pointer;
typedef struct node{
    int      data;      node-pointer next;      node-pointer prev;
};
```

اضافه کردن گره

اضافه کردن گره X به سمت راست گره با آدرس مشخص p

```
void insert(node-pointer p , node-pointer x){
    x -> prev = p;          (1)
    x -> next = p -> next; (2)
    p -> next -> prev = x; (3)
    p -> next = x;         (4)
}
```

مراحل کار:



تذکر: می توان ۸ حالت مجاز برای ترتیب شماره ها در نظر گرفت:

1234 , 2134 , 2314 , 2341 , 1324 , 3124 , 3214 , 3241

باید ابتدا خط ۲ و ۳ تنظیم شوند و سپس خط ۴.

حذف گره

تابع زیر گره با آدرس مشخص p را از لیست پیوندی دو طرفه حذف می کند. برای این کار کافی است اشاره گر next گره قبل از p ، به گره بعد از p اشاره کند و اشاره گر prev گره بعد از p ، به گره قبل از p اشاره کند.

```
delete(node-pointer p)
{
    p -> prev -> next = p -> next;
    p -> next -> prev = p -> prev;
    free(p);
}
```

تذکر: دو دستور زیر معادل هستند. اولی به زبان C و دومی شبیه کد:

- 1) $p -> prev -> next = p -> next;$
- 2) $next[prev[p]] \leftarrow next[p]$

 برای حذف یک گره از یک لیست پیوندی دو طرفه به ۲ تغییر اشاره گر نیاز است.

 برای اضافه کردن یک گره به یک لیست پیوندی دو طرفه به ۴ تغییر اشاره گر نیاز است.

 از مزایای لیست پیوندی دو طرفه نسبت به یک طرفه، می‌توان ساده شدن عملیات زیر را نام برد:

- ۱- حذف یک گره با معلوم بودن مکان آن گره.
- ۲- اضافه کردن یک گره قبل از گره‌ای با مکان معلوم.

کپی گرفتن از لیست

الگوریتم copy از لیست پیوندی دو طرفه first کپی گرفته و لیست پیوندی دو طرفه P را ایجاد می‌کند:

copy (first)

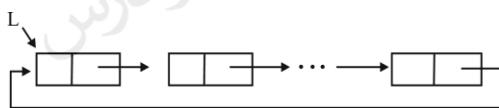
```

1 if (first = null)
2   p ← null;
3 else p←allocatenode (element[first], null , null );
4   next[p] ← copy (next[first]);
5   if (next[first] !=null) prev[next[p]] ← p;
6   return p;
```

لیست پیوندی حلقوی

لیست حلقوی (چرخشی)، نوعی لیست یک طرفه است که در آن اشاره گر آخرین گره، به جای مقدار دهی با NULL به ابتدای لیست اشاره می‌کند.

شکل زیر یک لیست حلقوی یک طرفه را نشان می‌دهد:



محاسبه طول یک لیست حلقوی

```

int length (list *p){
    int c=0;
    list-pointer *t;
    if (p) {
        t = p;
        do{
            c++;
            t = t ->next;
        } while(t != p);
    }
    return c;
}
```

درج در ابتدای یک لیست حلقوی (p : آدرس گره آخر)

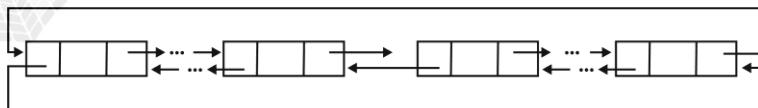
```
void insert(list-pointer *p, *n){
    if (*p==NULL){
        *p = n;
        n->next = n;
    }
    else{
        n->next = (*p)->next;
        (*p)->next = n;
    }
}
```

 مزیت لیست یکطرفه چرخشی نسبت به لیست یکطرفه غیرچرخشی در این است که به راحتی می‌توان به گره قبلی رفت.

 در لیست چرخشی با داشتن آدرس هر گره می‌توان به کلیه گره‌ها دسترسی داشت.

لیست دو طرفه چرخشی

نوعی لیست دو طرفه است که در آن اشاره‌گر راست آخرین گره به ابتدای لیست و اشاره‌گر چپ اولین گره به انتهای لیست اشاره می‌کند. به عبارتی next آخرین گره به گره اول و Prev اولین گره به گره آخر اشاره می‌کند. شکل زیر یک لیست چرخشی دو طرفه را نشان می‌دهد:



حذف گره x از یک لیست پیوندی دو طرفه چرخشی L با گره head:

```
if (x==L) Halt;
x->prev->next = x->next;
x->next->prev = x->prev;
free(x);
```

اضافه کردن گره p بعد از گره x در یک لیست پیوندی دو طرفه چرخشی:

```
p->prev = x;
p->next = x->next;
x->next->prev = p;
x->next = p;
```

تذکر: اگر آخرین خط یعنی; $p->next = null$ را در ابتداء اجرا کنیم، آن گاه دیگر به گره های واقع در سمت راست گره x در لیست اولیه، دسترسی نخواهیم داشت و به آنها زباله (Garbage) می گوییم.

مثال

یک لیست پیوندی یک طرفه بزرگ داریم. می خواهیم بررسی کنیم که آیا آخرین اشاره گر این لیست تهی است یا این که به یکی دیگر از عناصر لیست اشاره می کند. با فرض اینکه تعداد عناصر لیست n باشد و فقط به عنصر اول لیست دسترسی داشته باشیم، الگوریتم تشخیص این موضوع چگونه کار می کند؟
پاسخ:

دو اشاره گر p و q در نظر گرفته که سرعت حرکت p دو برابر q باشد. یعنی به ازای هر دو گام اشاره گر p ای که p می روید، q تنها یک واحد به جلو می روید. دو حالت رخ می دهد:

الف : اگر p به $null$ برسد، پس انتهای لیست $null$ است و زمان اجرای این کار $O(n)$ می باشد.

ب : اگر q به انتهای لیست برسد، p ناگزیر در آن حلقه قرار دارد. چون طول حلقه حداقل n است، حداقل پس از n مرحله، p و q به هم می رسند. در این حالت نیز زمان اجرا از $O(n)$ می باشد.



مسئله ی ژوزفوس

ژوزفوس یکی از ۴۱ یهودی ای بود که به وسیلهٔ رومیان در یک غار محاصره شده بودند. به جای تسلیم، این گروه تصمیم گرفتند که همگی به این صورت خودکشی کنند: آن‌ها قرار گذاشتند تا با شروع از نفر اول و به صورت حلقوی، هر بار نفر دوم زنده‌ها خود را بکشد و نوبت به نفر زنده‌ی بعدی برسد، تا این که هیچکس باقی نماند. ولی ژوزفوس زرنگ تر از آنها بود و مکان نشستن آخرین فردی را که قاعده‌تا باید خود را به تنهایی می کشت محاسبه کرد و از ابتداء در آن مکان نشست و جان سالم در برداشت.

مسئله در حالت کلی: اگر n نفر با شماره‌های ۱ تا n دور دایره‌ای قرار بگیرند و با شروع از شماره ۱ و در جهت ساعت گرد، هر بار دومین نفر زنده خودش را بکشد، آخرین نفر چه شماره ای دارد؟

مثال

ترتیب خودکشی برای $n=5$:

1, 2, 3, 4, 5 \Rightarrow 1, 3, 4, 5 \Rightarrow 1, 3, 5 \Rightarrow 3, 5 \Rightarrow 3

ترتیب خودکشی برای $n=6$:

1, 2, 3, 4, 5, 6 \Rightarrow 1, 3, 4, 5, 6 \Rightarrow 1, 3, 5, 6 \Rightarrow 1, 3, 5 \Rightarrow 1, 5 \Rightarrow 5

رابطه بازگشتی مسئله ژوزفوس

اگر تعداد افراد اولیه زوج باشند، فردی که در دور دوم در مکان x قرار گرفته، ابتداء در مکان $-2x-1$ بوده است. به طور نمونه اگر تعداد افراد اولیه 10 باشد، در دور دوم فردی که در مکان سوم قرار دارد، در ابتداء در مکان پنجم بوده است و یا فردی که در مکان چهارم قرار دارد، در ابتداء در مکان هفتم بوده است:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10 \Rightarrow 1, 3, 5, 7, 9

به طور نمونه اگر تعداد افراد اولیه 13 باشد، در دور دوم فردی که در مکان سوم قرار دارد، در ابتداء در مکان پنجم بوده است:
1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13 \Rightarrow 1, 3, 5, 7, 9, 11, 13

با چنین تحلیلی به رابطه بازگشتی زیر می‌رسیم:

$$f(1) = 1$$

$$f(2n) = 2f(n) - 1 \quad n \geq 1$$

$$f(2n+1) = 2f(n) + 1 \quad n \geq 1$$

جواب این رابطه بازگشتی برابر است با :

که k بزرگترین عدد توان 2 کوچکتر از n می‌باشد. (یعنی $\lfloor k = 2^{\lfloor \log n \rfloor}$)
تذکر: اگر n توانی از 2 باشد، جواب این رابطه برابر 1 می‌باشد.

مثال

برای $n=10$ فرد با چه شماره‌ای زنده می‌ماند؟

پاسخ: اگر در رابطه $2(n-k)+1$ به ازای n مقدار 10 و به ازای k مقدار 8 (بزرگترین عدد توان دو کوچکتر از 10) را قرار دهیم جواب 5 خواهد شد.

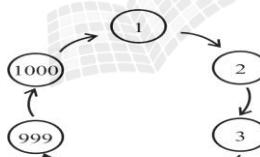
روش دوم: عدد 10 را یکبار به چپ شیفت دورانی دهیم $0101 \Rightarrow 1010$ نیز به عدد 5 می‌رسیم.

مثال

با فرض اجرای تابع بر روی لیست پیوندی حلقوی شکل زیر، مقدار خروجی چقدر است؟

(اشاره گر در ابتدا بر روی گره یا داده 1 می‌باشد).

```
int f(List *L){
    if (L->next == L)
        return L->data;
    L->next = L->next->next; (*)
    return f(L->next);
}
```



پاسخ: با هر بار اجرای دستور ستاره دار، یک گره از لیست حذف می‌شود. در هر بار اجرا اشاره گر یک گره به جلو می‌رود و تابع آنقدر اجرا می‌شود تا یک گره در لیست باقی بماند. در واقع این همان مسئله ژوفوس است. با قرار دادن 1000 به جای n و 512 (بزرگترین عدد توان دو کوچکتر از 1000) به جای k در رابطه $2(n-k)+1$ ، مقدار 977 حاصل می‌شود.

روش دوم:

عدد 1000 را به صورت یک عدد دودویی نشان داده و سپس آن را به چپ شیفت دورانی داد:

1111101000	1000
1111010001	977

تذکر: می‌توان برای محاسبه $f(n)$ ، $f(n) = 2t + 1$ که $t = 2^m + t$ و $n = 2^m + t$. آنگاه 1 .

مثال

مقدار $f(1000)$ را محاسبه کنید.

پاسخ: مقدار m را برابر 9 و مقدار t را برابر 488 در نظر می‌گیریم. $1000 = 2^9 + 488$. بنابراین :

$$f(1000) = 2 \times 488 + 1 = 977$$

الگوریتم مسئله ژوزفوس

JoeSePhous(n)

```

1  create(L)
2  first[L] ← q ← Allocate-node(1, null )
3  p ← q
4  for i ← 2 to n
5    do next[p]Node( i , next[p] )-Allocate ←
6      p ← next[p]
7  next[p]n← q; size[L] ←
8  p ← first(L)
9  while next[p] ≠ p
10   do Delete-After(L,p)
11     p← next[p]
12 print element[p]
```

مسئله ی ژوزفوس در حالت کلی

در مسئله ژوزفوس اگر هر بار k امین نفر زنده خودکشی کند، خواهیم داشت:

$$f(n, k) = ((f(n - 1, k) + k - 1) \bmod n) + 1$$

$$f(1, k) = 1$$

که در حالت k=2 خواهیم داشت:

$$f(n) = ((f(n - 1) + 1) \bmod n) + 1$$

مثال

مقدار f(4) را با فرض k=2 مشخص کنید.

$$f(4) = ((f(3) + 1) \bmod 4) + 1 = 4 \bmod 4 + 1 = 0 + 1 = 1$$

$$f(3) = ((f(2) + 1) \bmod 3) + 1 = 2 \bmod 3 + 1 = 2 + 1 = 3$$

$$f(2) = ((f(1) + 1) \bmod 2) + 1 = 2 \bmod 2 + 1 = 0 + 1 = 1$$

بررسی چند نمونه در حالت k=2 :

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
f(n)	1	1	3	1	3	5	7	1	3	5	7	9	11	13	15	1

مثال

مقدار f(4) را با فرض k=3 مشخص کنید.

اگر در رابطه k را برابر 3 قرار دهیم، خواهیم داشت:

$$f(n) = ((f(n - 1) + 2) \bmod n) + 1$$

$$f(4) = ((f(3) + 2) \bmod 4) + 1 = 4 \bmod 4 + 1 = 0 + 1 = 1$$

$$f(3) = ((f(2) + 2) \bmod 3) + 1 = 4 \bmod 3 + 1 = 1 + 1 = 2$$

$$f(2) = ((f(1) + 2) \bmod 2) + 1 = 3 \bmod 2 + 1 = 1 + 1 = 2$$

ترتیب خودکشی ها به صورت: 1,2,3,4 \Rightarrow 1,2,4 \Rightarrow 1,4 \Rightarrow 1

تذکر: اگر موقعیت ها از ۰ تا $n-1$ باشد، خواهیم داشت:

$$f(n, k) = (f(n-1, k) + k) \bmod n \quad f(1, k) = 0$$

کنکور ارشد

(مهندسی کامپیوتر - آزاد ۸۵)

- ۱- فرض کنید $y = (y_1, y_2, \dots, y_m)$ و $x = (x_1, x_2, \dots, x_n)$ دو لیست پیوندی خطی ساده باشند. مرتبه زمانی الگوریتمی که دو لیست را در لیست z ترکیب می‌کند، چیست؟

$$Z = \begin{cases} (x_1, y_1, x_2, \dots, x_m, y_m, x_{m+1}, \dots, x_n) & , \text{ if } m \leq n \\ (x_1, y_1, x_2, \dots, x_n, y_n, x_{n+1}, \dots, x_m) & , \text{ if } m > n \end{cases}$$

$O(\min(m, n))$ (۴) $O(mn)$ (۳) $O(m + n)$ (۲) $O(\max(m, n))$ (۱)

پاسخ: جواب گزینه ۴ است.

- با توجه به شرایط داده شده، دو لیست را به صورت یک درمیان ادغام کرده تا لیست کوچکتر تمام شود. سپس بقیه لیست بزرگتر را به انتهای لیست بدست آمده، اضافه می‌کنیم. بنابراین همواره به اندازه لیست کوچکتر، عمل پیمایش انجام می‌شود. بنابراین برای دو لیست به طول های n و m ، مرتبه الگوریتم برابر است با :

(مهندسی IT - دولتی ۸۳)

- ۲- فرض کنید بخواهیم ترتیب اشاره گرها را در یک لیست تک پیوندی وارون نمائیم. به عبارت دیگر، هر اشاره گر به جای اشاره به عنصر بعدی به عنصر قبلی اشاره نماید. برای انجام این کار کدام یک از گزاره‌های ذیل درست است؟ (الگوریتم غیر بازگشتی)

(۱) ۳ اشاره گر به جز نام لیست مورد نیاز است.

(۲) ۲ اشاره گر به جز نام لیست مورد نیاز است.

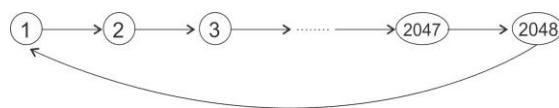
(۳) اشاره گر دیگری غیر از نام لیست، مورد نیاز نیست.

(۴) کافی است از یک استک استفاده کنیم، لذا اشاره گر بالای استک و نام لیست، مورد نیاز است.

حل: جواب گزینه ۳ است. برای معکوس کردن یک لیست یکطرفه به ۳ اشاره گر نیاز است.

(مهندسی کامپیوتر - دولتی ۷۶)

- ۳- با وجود بودن لیست پیوندی حلقوی به شکل زیر، خروجی قطعه کد داده شده چیست؟



همان مسئله ژوزفوس است و چون n توانی از دو می باشد، جواب 1 است.

روش دوم: عدد 2048 را به صورت یک عدد دودویی نشان داده و سپس آن را به چه شیفت دورانی داده که عدد 1 حاصل می شود:

$$100000000000 \Rightarrow 000000000001$$

روش سوم: می توان از رابطه $2^{\lfloor \log n \rfloor} + 1 = 2(n - 2)$ نیز استفاده کرد. خروجیتابع به ازای $n=2048$ برابر 1 خواهد بود. لگاریتم 2048 در پایه 2 برابر 11 می باشد).



فصل ۶:

درخت

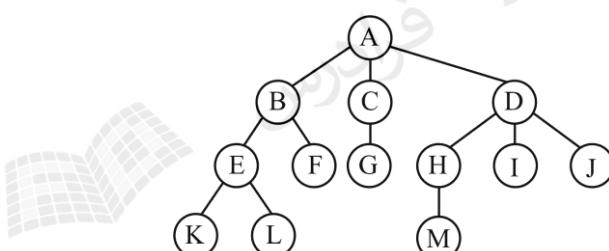
تعاریف اولیه

یک درخت مجموعه محدودی از یک یا چند گره می‌باشد که دارای گره خاصی به نام ریشه است و بقیه گره‌ها به n مجموعه مجزا تقسیم می‌شوند که هر یک از مجموعه‌ها خود نیز یک درخت می‌باشد.

تعداد زیردرخت‌های یک گره. (تعداد یالهایی که در گره برخورد دارند)	درجه گره
گره بدون فرزند (گره با درجه صفر)	برگ
طول بزرگترین مسیر از W به برگ W به طوری که W گره‌ای از زیر درختی به ریشه W باشد.	ارتفاع گره W
ارتفاع ریشه	ارتفاع درخت
طول مسیر از آن گره تا ریشه	سطح(یا عمق) یک گره
درختی که سطح برگهای آن حداقل یک واحد اختلاف داشته باشد.	درخت متوازن
درختی که سطح برگهای آن برابر است.	درخت کاملاً متوازن

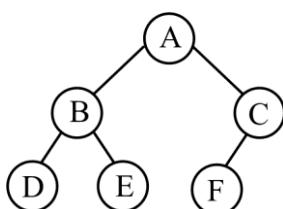
مثال

ارتفاع درخت زیر ۴ است: (ریشه را در سطح یک فرض کردہ‌ایم)



درخت دودویی

درختی که تعداد فرزندان هر گره در آن حداقل برابر دو باشد را درخت دودویی می‌نامند. به عبارتی یک درخت دودویی یا تهی است و یا حاوی مجموعه‌ای محدود از گره‌ها، که هر گره حداقل دو فرزند دارد. مانند درخت زیر:



بیشترین تعداد گره‌ها روی سطح i ام یک درخت دودویی برابر $2^{(i-1)}$ می‌باشد.

بیشترین تعداد گره‌ها در یک درخت دودویی به ارتفاع h ، برابر $2^h - 1$ می‌باشد.

مثال

بیشترین تعداد گره‌ها روی سطح پنجم یک درخت دودویی چند است؟ (ریشه در سطح یک می‌باشد).

پاسخ: بیشترین تعداد گره‌ها روی سطح پنجم یک درخت دودویی، طبق رابطه $2^{(i-1)}$ برابر ۱۶ است.

مثال

بیشترین تعداد گره‌ها در یک درخت دودویی به ارتفاع ۴ چند است؟

پاسخ: بیشترین تعداد گره‌ها در یک درخت دودویی به ارتفاع ۴، طبق رابطه $1 - 2^h$ برابر ۱۵ است.

 در هر درخت دودویی رابطه $n_0 = n_2 + 1$ برقرار است. (n_0 تعداد برگها و n_2 تعداد گره‌های دو فرزندی)

مثال

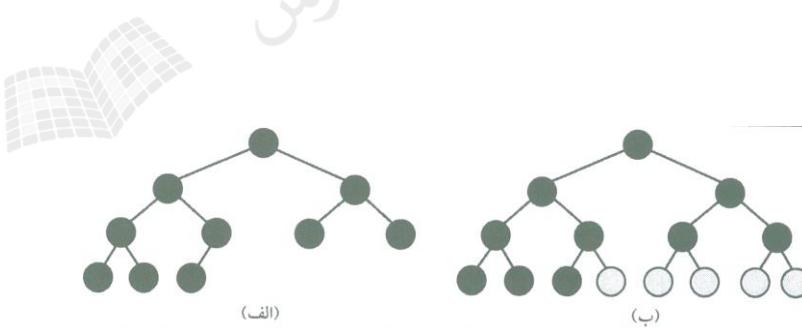
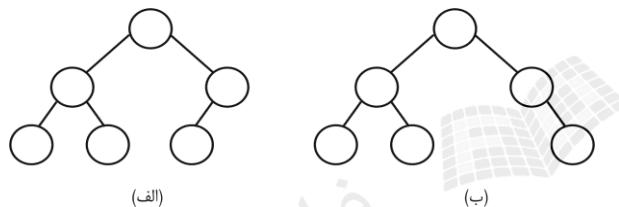
تعداد برگها در یک درخت دودویی شامل ۱۰ گره با درجه دو را بدست آورید؟

پاسخ: با توجه به رابطه $1 + n_0 = n_2$ ، تعداد برگها برابر ۱۱ می‌باشد.

درخت کامل

درختی که تمام سطح‌های آن بجز احتمالاً آخرین سطح، حداقل تعداد گره‌های ممکن را داشته باشد و گره‌های سطح آخر تا حد امکان در سمت چپ باشند را کامل می‌نامند. به طور نمونه درخت شکل الف کامل می‌باشد ولی درخت شکل ب کامل نمی‌باشد:

نمونه‌ای از درخت دودویی کامل و تبدیل آن به درخت پر:



 ارتفاع درخت دودویی کامل برابر $\log_2^{(n+1)} \lfloor \log_2^n \rfloor + 1$ می‌باشد. (همان حد بالای $\lfloor \log_2^n \rfloor + 1$ می‌باشد).

 تعداد گره‌های سطح آخر یک درخت دودویی کامل با n گره برابر است با :

$$n - (2^{(h-1)} - 1)$$

درخت پر

درختی که هم کامل و هم کاملاً متوازن باشد را درخت پر می‌نامند. در این درخت:

۱- ارتفاع درخت برابر $\log_2^{(n+1)}$ می‌باشد.

۲- تعداد گرهها برابر $2^h - 1$ می‌باشد.

۳- تعداد برگها، برابر $\frac{n+1}{2}$ و یا برابر $2^{(h-1)}$ می‌باشد.

۴- تعداد گره‌ها در سطح i ، برابر $2^{(i-1)}$ می‌باشد.

۵- تعداد لینک‌ها یکی کمتر از تعداد گره‌ها می‌باشد.

کران پایین و کران بالای تعداد گره‌ها

در یک درخت دودویی، کران بالای n برابر $2^h - 1$ (درخت پر) و کران پایین n برابر h می‌باشد. (درخت اریب). جدول زیر این کران‌ها را برای درخت دودویی کامل و پر هم نشان داده است:

$h \leq n \leq 2^h - 1$	درخت دودویی
$2^{(h-1)} \leq n \leq 2^h - 1$	درخت دودویی کامل
$2^h - 1 \leq n \leq 2^h - 1$	درخت دودویی پر

کران بالای ارتفاع درخت برابر n و کران پایین آن برابر $\lfloor \log n \rfloor + 1$ می‌باشد. بنابراین می‌توان نوشت:

با n گره می‌توان 2^{n-1} درخت دودویی متمایز (از لحاظ توبولوژی) به ارتفاع n ساخت.

درخت k تایی

درختی که حداقل تعداد فرزندان هر گره آن برابر k باشد را درخت k تایی می‌نامند. معروفترین درخت k تایی، درخت دوتایی (دودویی) است.

در درخت k تایی با n گره تعداد nk اتصال وجود دارد که $n-1$ اتصال استفاده شده و $(n-1)(n-k)$ اتصال تهی می‌باشد.

تعداد برگ‌ها در یک درخت k تایی:

$$n_0 = (k-1)n_k + (k-2)n_{k-1} + \dots + n_2 + 1$$

(n_k : تعداد برگها) (n_0 : تعداد گره‌های k فرزندی)

تذکر: تعداد برگ‌ها مستقل از تعداد گره‌های تک فرزندی است.

مثال

در یک درخت سه تایی، اگر ۵ گره دو فرزندی و ۱۰ برگ داشته باشیم، آنگاه چند گره ۳ فرزندی خواهیم داشت؟

پاسخ: در یک درخت سه تایی، رابطه زیر برقرار است:

$$n_0 = 2n_3 + n_2 + 1 \quad n_3 = 2 \Rightarrow 10 = 2n_3 + 5 + 1 \Rightarrow$$



تعداد برگ‌ها در یک درخت k تایی کامل با n گره، برابر است با:

$$\left\lfloor \frac{(k-1)n+1}{k} \right\rfloor$$

(یا $(n - \left\lceil \frac{n-1}{k} \right\rceil)$)

تعداد گره‌های غیر برگ در یک درخت k تایی کامل با n گره، برابر است با :

$$\left\lceil \frac{n-1}{k} \right\rceil$$

با دادن مقادیر ۲ و ۳ و ۴ به پارامتر k ، جدول زیر به دست می‌آید:

$n_0 = \left\lfloor \frac{n+1}{2} \right\rfloor$	تعداد برگها در درخت دو تایی کامل
$n_0 = \left\lfloor \frac{2n+1}{3} \right\rfloor$	تعداد برگها در درخت سه تایی کامل
$n_0 = \left\lfloor \frac{3n+1}{4} \right\rfloor$	تعداد برگها در درخت چهار تایی کامل

مثال

یک درخت چهار تایی کامل با ۲۷ گره، دارای چند گره برگ می‌باشد؟

$$n_0 = \left\lfloor \frac{3n+1}{4} \right\rfloor = \left\lfloor \frac{3 \times 27 + 1}{4} \right\rfloor = \left\lfloor 20.5 \right\rfloor = 20$$



مثال

حداکثر تعداد گره‌ها در یک درخت دودویی با b برگ چقدر است؟

پاسخ: کران بالا ندارد. چون یک درخت دودویی می‌تواند یک مسیر باشد و بی‌نهایت گره و تنها یک برگ داشته باشد.



بیشترین تعداد گره‌ها روی سطح i ام یک درخت k تایی برابر است با :

$$k^{(i-1)}$$

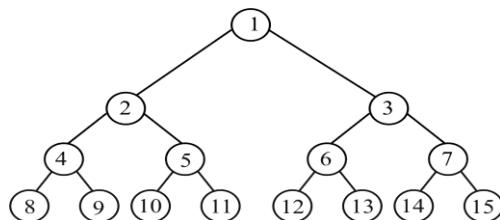
بیشترین تعداد گره‌ها در یک درخت k تایی به ارتفاع h ، برابر است با :

$$\frac{k^h - 1}{k - 1}$$

 تعداد برگ‌ها در درخت k تایی پر به ارتفاع h برابر است با :

درخت دودویی کامل شماره گذاری شده

به هر یک از گره‌های یک درخت دودویی کامل می‌توان یک شماره نسبت داد. در شکل زیر یک درخت دودویی کامل شماره گذاری شده با ارتفاع ۴ نمایش داده شده است:



در هر درخت دودویی کامل شماره گذاری شده، برای هر گره با اندیس i قواعد زیر برقرار است:

(۱) اگر $i=1$ باشد، آنگاه ریشه است.

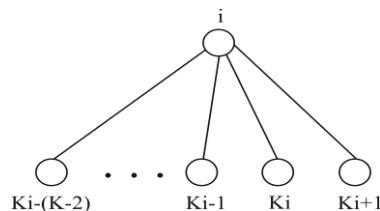
(۲) اگر $1 < i < n$ ، آنگاه والد آن یعنی $\text{parent}(i)$ ، در $\left\lfloor \frac{i}{2} \right\rfloor$ است.

(۳) اگر $2i \leq n$ ، آنگاه فرزند چپ آن یعنی $\text{Leftchild}(i)$ ، در $2i$ است.

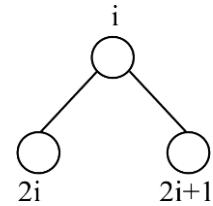
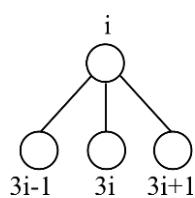
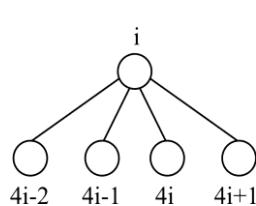
(۴) اگر $2i+1 \leq n$ ، آنگاه فرزند راست آن یعنی $\text{Rightchild}(i)$ ، در $2i+1$ است.

تذکر: اگر $2i > n$ ، آنگاه i فرزند چپ ندارد و اگر $2i+1 > n$ باشد، آنگاه i فرزند راست ندارد.

 یک گره با شماره i در یک درخت k تایی دارای فرزندان با شماره‌های زیر می‌باشد:

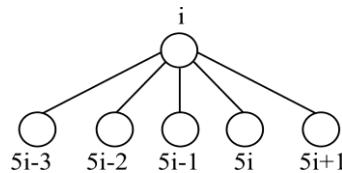


به طور نمونه در درخت‌های ۲ تایی، ۳ تایی و ۴ تایی داریم:



مثال

گره با شماره ۱۵، در یک درخت ۵ تایی کامل، پدر کدام گره‌ها می‌باشد؟ ($n=95$)
پاسخ: گره i ام پدر گره‌های $5i-3, 5i-2, 5i-1, 5i, 5i+1$ می‌باشد:



بنابراین گره پانزدهم پدر گره‌های ۷۶, ۷۳, ۷۴, ۷۵, ۷۲ می‌باشد.

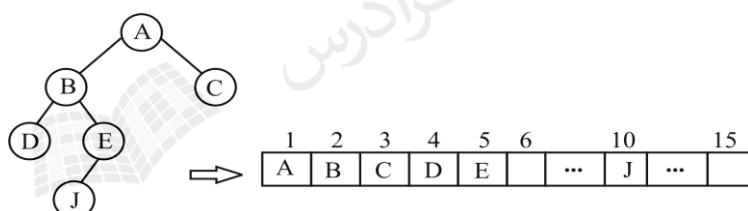


روشهای ذخیره درخت دودویی

یک درخت دودویی را می‌توان به کمک آرایه و یا لیست پیوندی در حافظه ذخیره کرد.

ذخیره درخت دودویی با استفاده از آرایه

برای ذخیره یک درخت دودویی به کمک آرایه، گره i ام با توجه به درخت دودویی کامل شماره گذاری شده، در خانه i ام در آرایه ذخیره می‌شود. در شکل زیر ذخیره درخت داده شده در آرایه نمایش داده شده است:



گره با شماره ۱ (با مقدار A) در خانه اول و گره با شماره ۵ (با مقدار E) در خانه پنجم آرایه ذخیره می‌شود

ذخیره درخت دودویی با استفاده از لیست پیوندی

هر گره در درخت دودویی شامل سه قسمت "داده، اشاره گر به فرزند چپ و اشاره گر به فرزند راست"
می‌باشد. تعریف یک گره درخت دودویی در زبان C به صورت زیر است:

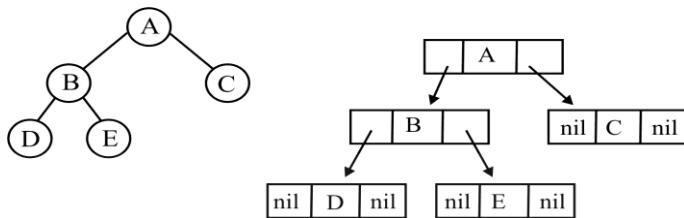
```

typedef struct node *tree-pointer;
typedef struct node{
    int           data;
    tree-pointer  left;
    tree-pointer  right;
} ;
  
```

درخت دودویی با n گره دارای $2n$ اشاره گر است که تعداد $n-1$ اشاره گر استفاده شده و $n+1$ اشاره گر استفاده نشده (تهی) می‌باشد.

مثال

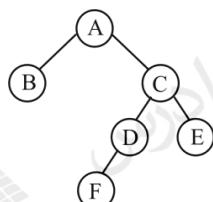
نمایش درخت دودویی با استفاده از لیست پیوندی:



مشاهده می‌شود که درخت دودویی بالا با ۵ گره، شامل ۱۰ اشاره گر است که ۴ اشاره گر آن استفاده شده و ۶ اشاره گر استفاده نشده (`nil`) می‌باشد.

مثال

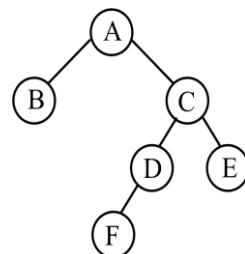
درخت زیر را به روش پرانتری نمایش دهید.



پاسخ: ■ $A(B,C(D(F),E)))$

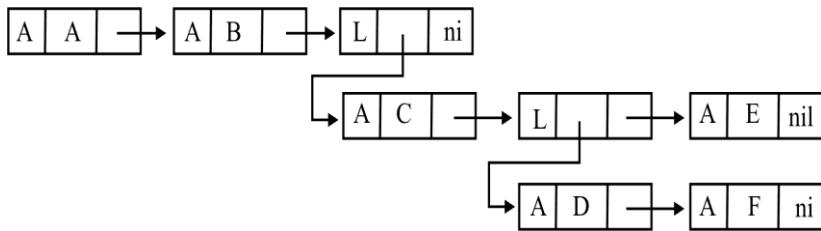
مثال

درخت زیر را با استفاده از لیست عمومی نمایش دهید.



پاسخ:

در این روش فرزندان هر گره در سمت راست آن ذکر می‌شوند و هر فرزندی که خود برگ نباشد به صورت یک زیر لیست نمایش داده می‌شود. گره‌های لیست با حرف L و گره‌های اتم با حرف A مشخص می‌شوند.



تعداد درخت‌های دودویی

اگر بخواهیم تعداد درخت‌های دودویی متفاوت را که می‌توان با n گره ساخت را بدست آوریم، فرض می‌کنیم هر یک از گره‌ها را به روش inorder از 1 تا n شماره گذاری کرده‌ایم. با فرض اینکه گره شماره i ($1 \leq i \leq n$) ریشه باشد، در آن صورت $i-1$ گره در زیر درخت چپ ریشه و $n-i$ گره در زیر درخت راست ریشه باید قرار داشته باشد. بنابراین اگر $T(n)$ را تعداد درخت‌های دودویی با n گره باشد، تعداد زیر درخت‌های چپ ریشه برابر $T(i-1)$ و تعداد زیر درخت‌های راست ریشه برابر $T(n-i)$ خواهد بود. بنابراین اگر گره i ریشه باشد، حاصل ضرب $T(i)T(n-i)$ برابر تعداد کل درخت‌ها می‌باشد. با در نظر گرفتن همه حالتها داریم:

$$T(n) = \sum_{i=1}^n T(i-1)T(n-i)$$

$$T(0) = T(1) = 1$$

تذکر: این رابطه را می‌توان به صورت $b_n = \sum_{k=1}^n b_{k-1}b_{n-k}$ نیز نشان داد. مقدار b_n برابر $\binom{2n}{n}$ می‌باشد که همان

n امین عدد کاتالان است. می‌توان نشان داد که :

$$b_n = \frac{4^n}{\sqrt{\pi}} \frac{1}{n^{3/2}} (1 + O(1/n)) \in \Omega(2^n)$$

مثال

تعداد درختهای دودویی متفاوت که با ۳ گره می‌توان ساخت را مشخص نماید.

پاسخ: یکی را ریشه قرار داده و حالت‌های زیر را ایجاد می‌کنیم:

الف- هر دو گره در راست ریشه

ب- یک گره در چپ ریشه و یک گره در راست ریشه

ج- هر دو گره در چپ ریشه

این حالتها با رابطه $b_0 = 1, b_1 = 1, b_2 = 2, b_3 = b_0b_2 + b_1b_1 + b_2b_0$ قابل بیان است، که از آنجا که

$$b_3 = 1 \times 2 + 1 \times 1 + 2 \times 1 = 5$$

یعنی ۵ درخت متفاوت می‌توان با ۳ گره ساخت.



مثال

تعداد درختهای دودویی متفاوت که با ۳ گره می‌توان ساخت را مشخص نماید.

پاسخ: تعداد درخت‌های دودویی مجزا با n گره برابر $\frac{1}{n+1} \binom{2n}{n}$ می‌باشد.

$$\frac{\binom{6}{3}}{4} = \frac{\frac{6!}{3!3!}}{4} = \frac{\frac{6 \times 5 \times 4 \times 3!}{3!3!}}{4} = \frac{5 \times 4}{4} = 5$$



الگوریتم‌های کار بر روی درخت دودویی

الگوریتم‌های زیادی برای کار بر روی درخت دودویی می‌توان نوشت، از جمله الگوریتم کپی گرفتن، محاسبه ارتفاع، شمارش تعداد گره‌ها و شمارش تعداد برگها.

کپی گرفتن از درخت دودویی

```
tree-pointer copy (treepointer *p){
    tree-pointer temp;
    if (p){
        t = malloc(sizeof(node));
        t -> data = p -> data;
        t -> lchild = copy (p -> lchild );
        t -> rchild = copy (p -> rchild);
        return t ;
    }
    return (NULL);
}
```

محاسبه ارتفاع درخت دودویی

```
int depth(tree-pointer root) {
    int d;
    if (root == NULL) return 0;
    x = depth(root -> left);
    y = depth(root -> right);
    if (x > y)
        d=x+1;
    else
        d=y+1;
    return d;
}
```

شمارش تعداد نودهای درخت دودویی

```
int count(tree-pointer root){
    int c;
    if (root==NULL) return 0;
    a = count (root -> left) ;
    b = count (root -> right);
    c = a+b+1;
    return c;
}
```

این تابع به طور بازگشتی تعداد گره های سمت چپ را شمرده و در NUML قرار داده و همچنین تعداد گره های سمت راست را در NUMR قرار می دهد و با افزایش یک واحد (شمارش ریشه) به مجموع این دو مقدار، تعداد گره های درخت را محاسبه می کند.

شمارش تعداد برگهای درخت دودویی

```
int f(treepointer *t){
    if (t==NULL) return 0;
    else if (t->lchild==NULL) && (t->rchild==NULL)
        return 1;
    else
        return ( f(t->lchild) + f(t->rchild) );
}
```

این تابع به طور بازگشتی به تک تک گره های درخت سر می زند و به ازای هر گره برگ، مقدار 1 را برمی گرداند و به ازای سایر گره ها، تابع را با اشاره گره های چپ و راست آن گره صدا می زند و مقادیر بازگشتی را با هم جمع می کند و بعنوان خروجی به ازای آن گره برمی گرداند.

می توان این تابع را به صورت زیر نشان داد:

$$f(t) = f(\text{left}(t)) + f(\text{right}[t]) + \text{IsLeaf}(t)$$

که تابع $f(t)$ اگر t برگ باشد مقدار 1 و گرنه مقدار 0 را برمی گرداند.

پیمایش درخت دودویی

پیمایش درخت، یعنی حرکت روی یالهای درخت و ملاقات همه گره‌های آن دقیقاً یکبار. اگر انشعاب به چپ در هر گره درخت دودویی را با L، انشعاب به راست را با R و ملاقات گره را با V نمایش دهیم، امکان تولید شش ترکیب وجود دارد.

در سه پیمایش همواره انشعاب به چپ قبل از انشعاب به راست صورت گرفته که روش‌های معمول می‌باشند و عبارتند از:

(۱) پیشوندی (VLR) (ریشه - چپ - راست) (preorder)

(۲) میانوندی (LVR) (چپ - ریشه - راست) (inorder)

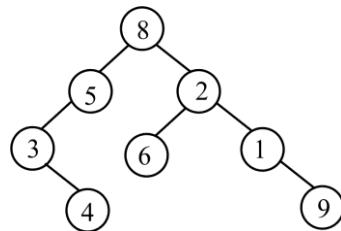
(۳) پسوندی (LRV) (چپ - راست - ریشه) (postorder)

در پیمایش پیشوندی، ابتدا ریشه ملاقات می‌شود. سپس زیر درخت سمت چپ به روش پیشوندی پیمایش می‌شود. در نهایت زیر درخت سمت راست به روش پیشوندی پیمایش می‌شود.

پیشوندی	میانوندی	پسوندی
<pre>void f(tree-pointer p) { if (p){ cout << p-> data; f(p -> left); f(p -> right); } }</pre>	<pre>void f(tree-pointer p) { if (p){ f(p -> left); cout<<p-> data; f(p -> right); } }</pre>	<pre>void f(tree-pointer p) { if (p){ f(p -> left); f(p -> right); cout<<p-> data; } }</pre>

مثال

حاصل پیمایش های معمول درخت زیر را بدست آورید.



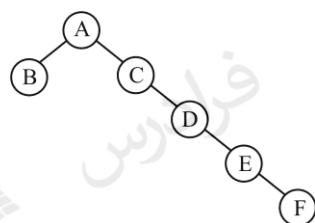
inorder : 34586219

preorder : 85342619

postorder : 43569128

مثال

حاصل پیمایش های معمول درخت زیر را بدست آورید.



inorder : BACDEF

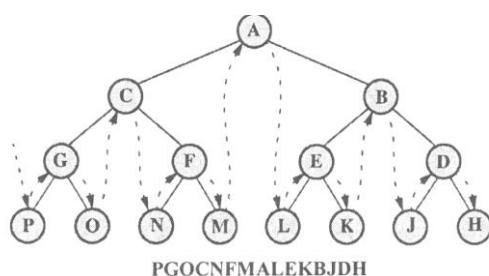
preorder : ABCDEF

postorder : BFEDCA

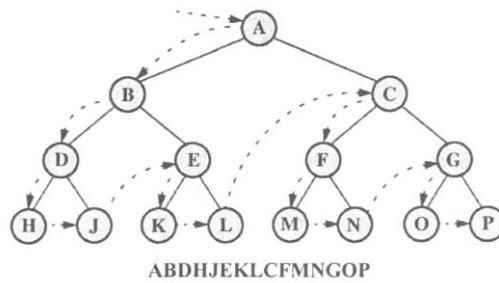
مثال

شكل های زیر شیوه پیمایش درخت دودویی را نشان می دهد:

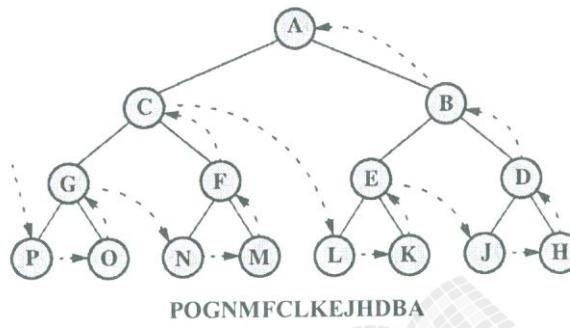
الف : **inorder**



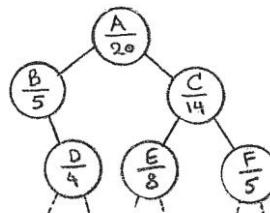
ب : preorder



ج : postorder

**مثال**

در شکل زیر قسمتی از یک درخت دودویی نشان داده شده است. در زیر برچسب هر گره عددی نوشته شده است که تعداد کل گره های موجود در زیر درخت آن گره بعلاوه خود آن گره را نشان می دهد. در پیمایش preorder گره F چندمین گره خواهد بود؟



پاسخ:

برای ملاقات F در پیمایش preorder (ریشه-چپ-راست)، مراحل زیر انجام می شود:

الف- پیمایش گره A (ریشه)

ب- پیمایش 5 گره واقع در زیر درخت چپ A

ج- پیمایش گره C

د- پیمایش 8 گره واقع در زیر درخت چپ C

ه- پیمایش گره F

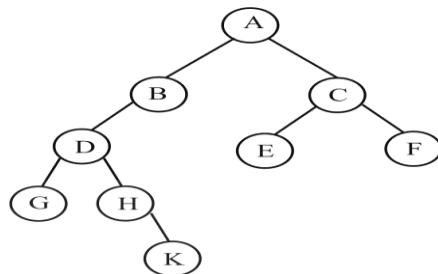
بنابراین گره F، 16 امین گره در پیمایش preorder می‌باشد.

$$1+5+1+8+1=16$$



مثال

تعداد push و pop های مورد نیاز برای پیمایش preorder درخت دودویی زیر را بدست آورید؟



پاسخ:

تعداد چهار push و چهار pop داریم. در پیمایش preorder (ریشه-چپ-راست)، بچه‌های راست (C,H,K,F) در پشته push می‌شوند. مراحل کار به صورت زیر است:

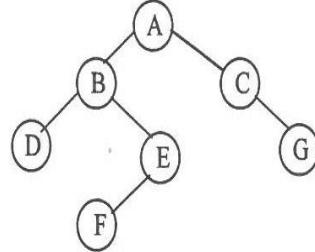
- ۱- قرار دادن آدرس گره با مقدار A در PTR
- ۲- پردازش A و push بچه راست آن یعنی C
- ۳- پردازش B
- ۴- پردازش D و push بچه راست آن یعنی H
- ۵- پردازش G
- ۶- pop کردن عنصر بالای پشته یعنی H و قرار دادن آن در PTR
- ۷- پردازش H و push کردن بچه راست آن یعنی K
- ۸- pop کردن K از پشته و قرار دادن PTR=K
- ۹- پردازش K
- ۱۰- POP کردن C از پشته و قرار دادن C در PTR
- ۱۱- پردازش C، push کردن بچه راست F
- ۱۲- پردازش E
- ۱۳- pop کردن F و قرار دادن F در PTR
- ۱۴- پردازش F



مثال

خروجی الگوریتم زیر برای درخت باینری داده شده کدام است؟

```
f( t ){
    if (t==null) return;
    if (t->left != null ) cout<< '(' ;
    f(t->left);
    if (t->left != null ) cout<< ')';
    cout<<t->data;
    if (t->right != null ) cout<< '(' ;
    f( t->right);
    if (t->right != null ) cout<< ')';
}
}
```



پاسخ:

تابع داده شده، دنباله inorder درخت را به صورت پرانتر گذاری شده چاپ می کند.(قبل و بعد از ریشه درخت و ریشه زیر درخت ها، پرانتر نمی گذارد.)

((D) B ((F) E)) A (C (G))

پیمایش inorder غیر بازگشتی

برای شبیه سازی بازگشت پذیری نیاز به پشته داریم. گره های چپ داخل پشته قرار می گیرند تا هنگامیکه به یک گره تهی برسیم و سپس گره از پشته خارج و فرزند راست آن گره در پشته قرار می گیرد. هر زمان پشته تهی شود، پیمایش کامل می گردد.

```
void inorder(tree-pointer node){
    int top = -1;
    tree-pointer stack[MAX];
    for( ; ; ) {
        for ( ; node ; node= node->left)
            add(&top, node);
        node = delete(&top);
        if (!node) break;
        cout << node->data;
        node = node->right;
    }
}
```

تذکر: تابع add با آنچه که در فصل پشته تعریف شده، متفاوت بوده زیرا نوع عناصر پشته متفاوت می باشند. تابع delete، بجای بازگشت نوع عنصر(tree-pointer element) مقدار NULL برگشت داده می شود.

 هر گره درخت فقط یکبار در پشته قرار گرفته و یا از آن خارج می‌گردد. بنابراین، اگر تعداد گره‌های درخت برابر با n باشد، پیچیدگی زمان تابع برابر با $O(n)$ می‌باشد. حافظه مورد نیاز برابر با ارتفاع درخت است که مساوی با $O(n)$ می‌باشد.

مشخص کردن گره‌های تک فرزندی با داشتن دو پیمایش پیشوندی و پسوندی

می‌توان به کمک دو پیمایش preorder و postorder یک درخت دودویی، گره‌های تک فرزندی را مشخص کرد. برای این کار در پیمایش Preorder از چپ به راست حرکت کرده و زوج پشت سرهمی که معکوس آن در پیمایش postorder باشد را پیدا می‌کنیم. اولین گره در این زوج‌ها، گره‌های تک فرزندی می‌باشند.

مثال

گره‌های تک فرزندی درخت دودویی با پیمایش‌های زیر را مشخص کنید.

Preorder : 85342619

Postorder : 43569128

پاسخ:

زوج‌های پشت سرهمی که در preorder وجود دارند و معکوس آنها در postorder موجود است، عبارتند از: (19) و (34) و (53). بنابراین اولین گره این زوج‌ها یعنی 1, 3, 5 گره‌های تک فرزندی می‌باشند.

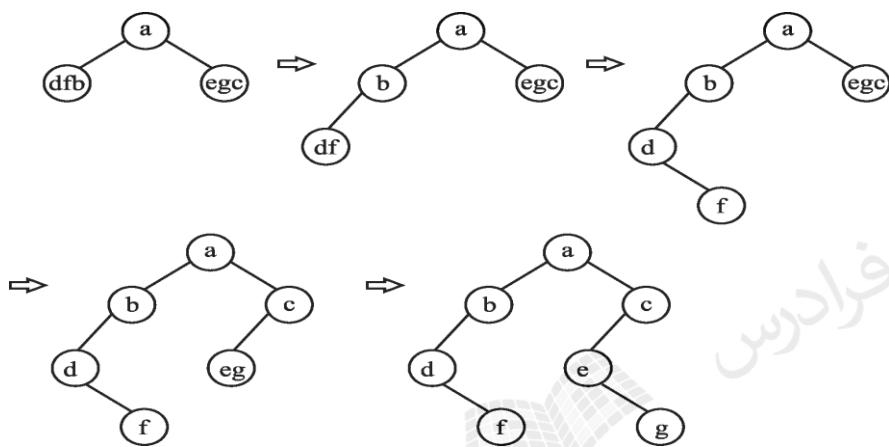


رسم درخت دودویی با داشتن دو پیمایش "inorder" و "postorder" و یا "preorder" و "inorder" می‌توان با داشتن دو پیمایش از یک درخت دودویی، درخت را ترسیم کرد، به شرط آنکه یکی از آنها میانوندی باشد.

مثال

درخت دودویی که پیمایش پیشوندی آن abdfceg و پیمایش میانوندی آن dfbaegc باشد، را رسم نمایید.
پاسخ:

گره a ریشه است، چون ریشه اولین گره از پیمایش پیشوندی است و با توجه به موقعیت a در پیمایش میانوندی، مشخص می‌شود که گره dfb در سمت چپ ریشه و گره‌های egc در سمت راست ریشه قرار دارند. حال در پیمایش پیشوندی به جلو حرکت کرده و به گره b می‌رسیم و با نگاه به موقعیت b در پیمایش میانوندی مشخص می‌شود که گره‌های df در چپ b قرار دارند و به همین روال ادامه می‌دهیم و درخت را می‌سازیم:



با داشتن پیمایش postorder یک درخت دودویی کامل با برچسب‌های متفاوت و مشخص بودن برگهای برگهای درخت، می‌توان درخت را ساخت و حاصل درختی واحد است.

رسم درخت دودویی با داشتن دو پیمایش postorder و preorder

اگر پیمایش‌های پیشوندی و پسوندی یک درخت دودویی با n گره در دسترس باشند، ریشه و گره‌های تک فرزندی را می‌توان تعیین کرد، اما محل گره‌های تک فرزندی را نمی‌توان مشخص کرد. بنابراین در صورت وجود گره‌های تک فرزندی، چندین درخت می‌توان ایجاد کرد که پیمایش postorder و preorder آنها با هم برابر باشند. تعداد این درخت‌ها برابر است با 2^k که k تعداد گره‌های تک فرزندی است.

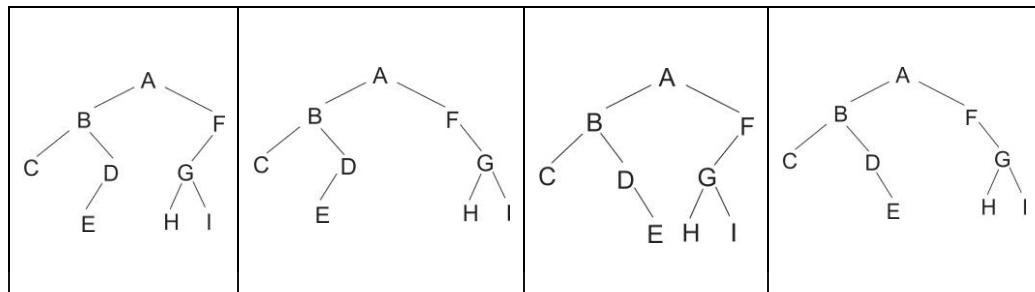
مثال

درخت دودویی که پیمایش پیشوندی و پسوندی آن به صورت زیر است را رسم نمایید.

Preorder : A B C DE FG H I

Postorder : C ED B H I GF A

پاسخ: در پیمایش Preorder از چپ به راست حرکت کرده و زوج پشت سرهمی که معکوس آن در پیمایش postorder باشد را پیدا می‌کنیم. در این مثال دو زوج داریم : (DE) و (FG) . اولین گره در این زوج ها، یعنی D و گره های تک فرزندی می‌باشند. با توجه به اینکه E می‌تواند در چپ یا راست D قرار بگیرد و یا G در چپ یا راست F قرار بگیرد، چهار حالت رخ می‌دهد که در تمامی آنها، پیمایش preorder و postorder یکسان است. این درخت ها در شکل زیر نشان داده شده اند:



مثال

درخت دودویی که پیمایش پیشوندی و پسوندی آن به صورت زیر است را رسم نمایید. (در این درخت ، هر گره دارای دو فرزند است)

Preorder : A B D J E O P F C G M H I K L

Postorder : D O P E F J B G H K L I M C A

پاسخ: واضح است که A ریشه است. برای پیدا کردن کاراکترهای زیر درخت چپ A ، در ترتیب Preorder و پس از A به دنبال رشته ای می‌گردیم که کاراکترهای آن به طور کامل در رشته ای از ابتدای ترتیب Postorder هم ظاهر شود. بدیهی است که حرف اول preorder با حرف آخر postorder باید یکی باشند. این کاراکترها عبارتند از: BDJEOPF . بنابراین کاراکترهای C G M H I K L در راست A قرار دارند. همین روال را به صورت بازگشته بر روی هر یک از زیر درخت ها اعمال کرده تا درخت اصلی را به دست آورد. این روال از مرتبه $O(n^2)$ می‌باشد.

در درخت دودویی با داشتن هر سه شرط زیر می‌توان درخت را ساخت و حاصل نیز یکتا می‌باشد:

۱- همه گره ها برگ یا دو فرزندی باشند.

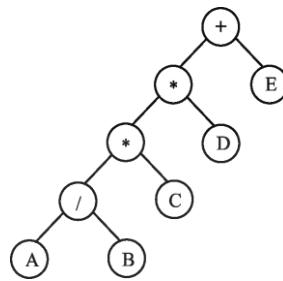
۲- برچسب برگ ها در پیمایش مشخص باشد.

۳- یکی از پیمایش های levelorder preorder یا postorder وجود داشته باشد.

پیمایش درخت به ترتیب سطح

پیمایش‌های میانوندی، پیشوندی و پسوندی به هر دو صورت بازگشتی یا تکرار مراحل، به پشته نیاز دارند. در پیمایش درخت به ترتیب سطح از صف استفاده می‌شود. در این پیمایش اول ریشه، بعد فرزند چپ ریشه و سپس فرزند راست ریشه ملاقات می‌شوند. این فرآیند تکرار می‌شود، بررسی گره‌ها در هر سطح جدید از سمت چپ به راست، صورت می‌گیرد.

پیمایش به ترتیب سطح درخت زیر $+ * E * D / C A B$ می‌باشد:



زیربرنامه پیمایش به ترتیب سطح یک درخت دودویی

```

void levelorder(tree-pointer ptr){
    int front = rear = 0;
    tree-pointer queue[MAX];
    if (!ptr) return;
    addq(front ,&rear , ptr);
    for( ; ; ) {
        ptr = deleteq(&front , rear);
        if (ptr){
            cout << ptr ->data;
            if ( ptr ->left ) addq( front , &rear , ptr ->left);
            if ( ptr ->right ) addq( front , &rear , ptr ->right);
            r = q.delete( );
        }
    }
}
  
```

درخت نخی دودویی

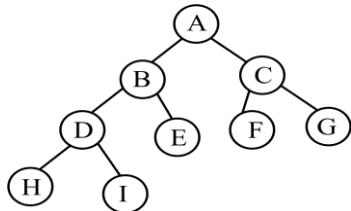
از $2n$ اشاره‌گر موجود در یک درخت دودویی با n گره، $n-1$ اشاره‌گر استفاده شده و $n+1$ اشاره‌گر استفاده نشده است. با استفاده از این اشاره‌گرهای بدون استفاده می‌توان به عناصر قبلی یا بعدی در یک پیمایش اشاره کرد که باعث بالا رفتن سرعت پیمایش درخت می‌شود. به درختی که از اشاره‌گرهای بدون استفاده آن این چنین استفاده شود درخت نخی می‌گویند.

قوانین ایجاد اتصالات نخی

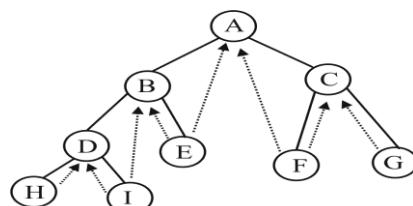
- ۱- اگر $ptr->left$ تهی باشد، آن را طوری تغییر می‌دهیم که به گره ای که در پیمایش inorder قرار دارد، اشاره کند.
- ۲- اگر $ptr->right$ تهی باشد، آن را به گونه‌ای تغییر می‌دهیم که به گره ای که در پیمایش inorder بعد از ptr قرار دارد، اشاره کند.

مثال

درخت نخ کشی شده حاصل از پیمایش inorder درخت زیر را بدست آورید؟



پاسخ: پیمایش میانوندی درخت به صورت $H \ D \ I \ B \ E \ A \ F \ C \ G$ می‌باشد. درخت نخ کشی به صورت زیر است:



به طور مثال با توجه به پیمایش inorder، در سمت چپ گره F، گره A قرار دارد و در سمت راست آن گره C قرار دارد. و یا قبل و بعد از گره I در پیمایش inorder گره‌های D و B می‌باشند که اتصال‌های تهی گره I را به آن‌ها تنظیم می‌کنیم.

تذکر: چون می‌خواهیم که هیچ اتصالی تهی نباشد، یک گره head برای هر درخت دودویی نخی در نظر می‌گیریم. گره H اولین گره است و اشاره گر سمت چپ آن را به گره head تنظیم می‌کنیم. همچنان گره G آخرین گره پیمایش است و اشاره گر سمت راست آن را به گره head تنظیم می‌کنیم.

ساختار یک گره در درخت نخی دودویی به صورت زیر است:

```

typedef struct threaded-tree *tp;
typedef struct threaded-tree{
    char     data;
    tp      left;
    tp      right;
    short int lf;
    short int rf;
} ;
  
```

از دو فیلد منطقی Lf, Rf برای امکان تشخیص اشاره گرهای واقعی از اشاره گرهای نخی استفاده می‌شود. اگر Lf=1 باشد، یک اشاره گر نخی است و در غیر این صورت اشاره گر عادی به فرزند چپ است. اگر Rf=1 باشد، یک اشاره گر نخی است و در غیر این صورت اشاره گر عادی به فرزند راست است.

پیمایش inorder یک درخت نخی دودویی

به کمک اشاره گرهای نخی می‌توان الگوریتم inorder را ساده‌تر نوشت. برای این کار می‌توانیم با فراخوانی مکررتابع insucc تمام گره‌ها را بازیابی کنیم.تابع insucc بدون استفاده از پشته، گره بعدی در پیمایش inorder را در یک درخت نخی دودویی پیدا می‌کند.

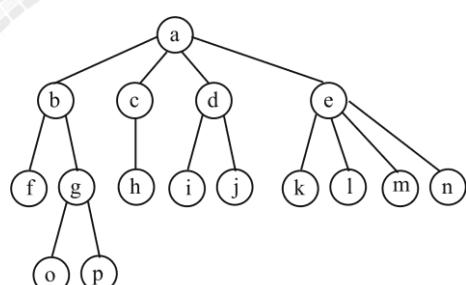
<pre>void inorder(tp tree) { for(; ;) { temp = insucc(temp); if (temp == tree) break; cout << temp->data; } }</pre>	<pre>tp insucc(tp tree) { tp temp; temp = tree -> right; if (!tree-> lf) while(! temp -> lf) temp = temp -> left; return temp; }</pre>
--	--

تذکر: زمان محاسباتی این پیمایش نیز $O(n)$ است.

تذکر: اتصالات نخی، الگوریتم preorder و postorder را نیز ساده می‌کند.

درخت عمومی

یک درخت k تایی که در آن فقط یک گره به نام ریشه با درجه ورودی صفر وجود دارد و سایر گره‌ها دارای یک کمان ورودی می‌باشند را درخت عمومی می‌گویند. شکل زیر، یک درخت عمومی ۴ تایی را نشان می‌دهد:



 تفاوت‌های درخت عمومی با درخت دودویی عبارتند از :

- ۱- درخت دودویی می‌تواند تهی باشد، اما درخت عمومی نمی‌تواند تهی باشد.
- ۲- با فرض اینکه یک گره دارای یک فرزند باشد، آنگاه این فرزند در یک درخت دودویی با عنوان بچه چپ یا راست از هم متمایز می‌شوند، اما در یک درخت عمومی تفاوتی بین آنها وجود ندارد.

تبدیل درخت عمومی به درخت دودویی

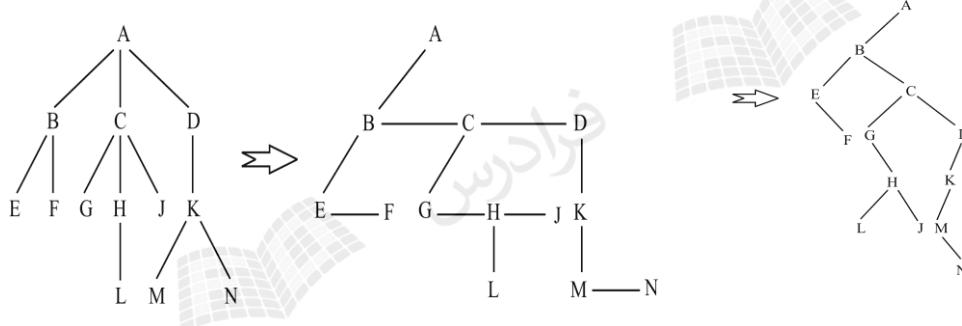
به دلیل اتلاف حافظه توسط اشاره‌گرهای تهی در درخت‌های عمومی، درخت عمومی را به درخت دودویی تبدیل می‌کنیم.

مراحل تبدیل عبارتند از:

- ۱- در هر سطح کلیه گره‌های کنار هم (فرزندان یک پدر) را به یکدیگر متصل می‌کنیم.
- ۲- کلیه اتصالات گره‌ها به گره پدر، بجز اتصال سمت چپ‌ترین فرزند را قطع می‌کنیم.
- ۳- فرزند هر گره در سمت چپ و برادر هر گره در سمت راست آن گره قرار می‌گیرد. (برادر راست-فرزند چپ)

مثال

تبدیل درخت عمومی به درخت دودویی

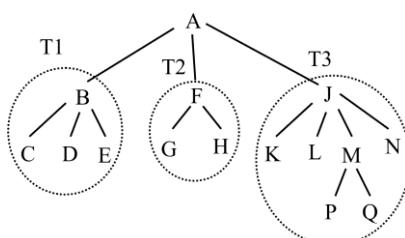


 درخت معادل دودویی، در ریشه زیر درخت راست ندارد.

 هیچ پیمایشی در درخت عمومی منتظر با پیمایش postorder درخت دودویی معادل آن نمی‌باشد.

مثال

پیمایش preorder درخت عمومی زیر را پیدا کنید.



پاسخ: نحوه پیمایش preorder برابر است با :

۱- پردازش ریشه A

۲- پیمایش preorder درخت T1

۳- پیمایش preorder درخت T2

۴- پیمایش preorder درخت T3

نتیجه: A,B,C,D,E,F,G,H,J,K,L,M,P,Q,N

 البته می توان برای بدست آوردن پیمایش preorder یک درخت عمومی، آن را به درخت دودویی تبدیل کرده و سپس درخت دودویی حاصل را به صورت preorder پیمایش می کنیم.

مثال

پیمایش postorder درخت عمومی مثال قبل را پیدا کنید.

پاسخ: نحوه پیمایش postorder برابر است با:

۱- پیمایش postorder درخت T1

۲- پیمایش postorder درخت T2

۳- پیمایش postorder درخت T3

۴- پردازش ریشه A

نتیجه : C,D,E,B,G,H,F,K,L,P,Q,M,N,J,A

 البته می توان برای بدست آوردن پیمایش postorder یک درخت عمومی، آن را به درخت دودویی تبدیل کرده و سپس درخت دودویی حاصل را به صورت inorder پیمایش می کنیم.

 اگر T' درخت دودویی معادل درخت T باشد، آنگاه :

$$\text{preorder}(T') = \text{preorder}(T), \quad \text{inorder}(T') = \text{postorder}(T)$$

جنگل

جنگل شامل n درخت مجزا است. به عبارتی مجموعه‌ای مرتب از صفر یا چند درخت متمایز است. همچنین می توان گفت با

برداشتن ریشه یک درخت دودویی T، جنگل F به دست می‌آید. مراحل تبدیل جنگل به یک درخت دودویی عبارت است از:

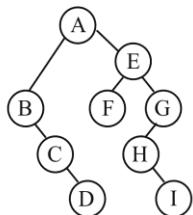
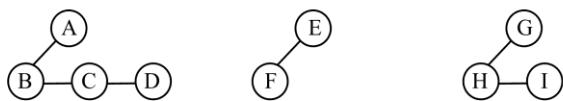
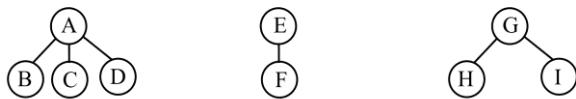
۱- هر درخت جنگل را به یک درخت دودویی تبدیل می کنیم.

۲- درختهای دودویی را از طریق فرزنده راست گره‌های ریشه به هم متصل می نماییم.

مثال

جنگل زیر که از ۳ درخت عمومی تشکیل شده را به یک درخت دودویی تبدیل کنید.

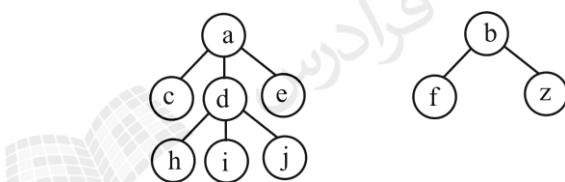
پاسخ: ابتدا هر یک از درختان عمومی را به دودویی تبدیل کرده و سپس درختهای دودویی را از طریق فرزند راست به یکدیگر متصل می‌نماییم.



درخت دودویی حاصل از جنگل، در ریشه زیر درخت راست دارد.

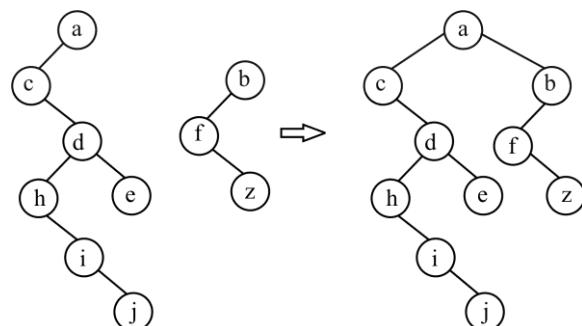
مثال

پیمایش inorder جنگل کدام است؟



پاسخ:

ابتدا جنگل را به درخت دودویی تبدیل می‌کنیم:



سپس پیمایش inorder درخت دودویی حاصل را بدست می‌آوریم:

chijdeafzb

پیمایش inorder درخت دودویی و جنگل نتایج یکسانی دارند.

 پیمایش preorder درخت دودویی و جنگل نتایج یکسانی دارند.

 پیمایش postorder درخت دودویی و جنگل همیشه نتایج یکسانی ندارند.

درخت دودویی گسترش یافته (2-Tree)

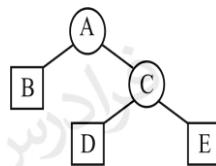
درخت دودویی که در آن هر گره ۰ یا ۲ فرزند دارد ۲-کامل) می‌نامند. گره‌هایی که فرزند ندارند را خارجی و گره‌هایی که ۲ فرزند دارند را داخلی می‌نامیم.

 در 2-Tree ، تعداد گره‌های خارجی را با E و تعداد گره‌های داخلی را با I نشان می‌دهیم.

 در 2-Tree مجموع تمام طولها از ریشه تا هر گره خارجی را طول مسیر خارجی (L_E) و مجموع تمام طولها از ریشه تا هر گره داخلی را طول مسیر داخلی (L_I) می‌نامیم.

مثال

با توجه به 2-TREE زیر ، طول مسیر خارجی (L_E) و طول مسیر داخلی را (L_I) بدست آورید.



پاسخ:

درخت داده شده شامل دو گره داخلی (A,C) و سه گره خارجی (B,D,E) می‌باشد، بنابراین $I=2$ و $E=3$ می‌باشد. برای محاسبه L_I ، فاصله گره داخلی A از ریشه (یعنی صفر) را با فاصله گره داخلی C از ریشه (یعنی 1) جمع می‌کنیم. همچنین برای محاسبه L_E ، فاصله گره خارجی B از ریشه (یعنی یک) و فاصله گره خارجی D از ریشه (یعنی 2) و فاصله گره خارجی E از ریشه (یعنی 2) را با هم جمع می‌کنیم:

$$L_I = 0 + 1 = 1 , L_E = 1 + 2 + 2 = 5$$

 در 2-Tree رابطه $L_E = L_I + 2I$ و $E = I + 1$ برقرار است.

 یک 2-Tree با کمترین طول مسیر داخلی، یک درخت دودویی کامل است.

 در یک درخت ۲-کامل با ارتفاع h ، کمترین تعداد گره‌ها برابر $2^{h+1} - 1$ است. در این حالت، درخت شبیه یک مسیر است که از هر گره آن یک فرزند اضافه نیز خارج شده است.

 در یک درخت ۲-کامل با ارتفاع h ، بیشترین تعداد گره‌ها برابر 2^{h+1} است.(درخت کائند هرم است).

مثال

یک درخت دودویی گسترش یافته شامل 6 گره خارجی می‌باشد. تعداد گره‌های داخلی آن چیست؟
پاسخ: در یک درخت دودویی گسترش یافته، تعداد گره‌های خارجی، یکی بیشتر از تعداد گره‌های داخلی است.
بنابراین اگر تعداد گره‌های خارجی 6 باشد، تعداد گره‌های داخلی 5 است.

مثال

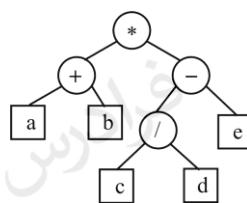
طول مسیر خارجی در یک درخت دودویی گسترش یافته شامل 5 گره داخلی، با طول مسیر داخلی 8، چند است؟

حل:

$$L_E = L_I + 2n = 8 + 2 \times 5 = 18$$

مثال

عبارت جبری متناظر با 2-TREE زیر کدام است؟



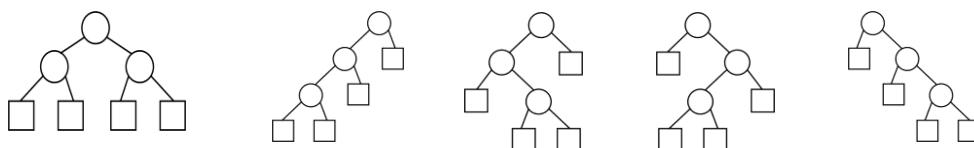
پاسخ: عبارت جبری متناظر با درخت داده شده برابر است با :

$$E = (a+b)*((c/d)-e)$$

مثال

چند 2-Tree غیر متشابه با 4 گره خارجی می‌توان ایجاد کرد؟

پاسخ: تعداد ۵ درخت دودویی توسعه یافته می‌توان به چهار گره خارجی به صورت زیر ساخت:



در واقع تعداد 2-Tree هایی که می‌توان با 4 گره خارجی ساخت برابر است با تعداد درختهای دودویی که می‌توان با 3 گره ساخت.

تعداد 2-Tree هایی که می‌توان با n گره خارجی ساخت برابر است با تعداد درختهای دودویی که می‌توان با 1 گره ساخت.

تذکر: درخت متناظر با یک عبارت جبری دلخواه که تنها از عملگرهای دوتایی استفاده می‌کند، نمونه‌ای مهم از 2-TREE می‌باشد که عملوندها گره‌های خارجی و عملگرها، گره‌های داخلی می‌باشند.

مثال

در یک درخت ۲-کامل، اگر $n(T)$ تعداد گره‌های غیر برگ در T باشد و $h(T)$ ارتفاع T (فاصله دورترین برگ از ریشه T) باشد، آنگاه $n(T) - h(T)$ را محاسبه کنید.
پاسخ:

با فرض اینکه T_R و T_L به ترتیب زیر درخت‌های راست و چپ T باشند، برای T که برگ نباشد، داریم:

$$n(T) = n(T_L) + n(T_R) + 1$$

$$h(T) = 1 + \max\{h(T_L), h(T_R)\}$$

بنابراین با تفاضل این دو خواهیم داشت:

$$\begin{aligned} n(T) - h(T) &= n(T_L) + n(T_R) + 1 - (1 + \max\{h(T_L), h(T_R)\}) \\ &= n(T_L) + n(T_R) - (h(T_L) + h(T_R) - \min\{h(T_L), h(T_R)\}) \\ &= (n(T_L) - h(T_L)) + (n(T_R) - h(T_R)) + \min\{h(T_L), h(T_R)\} \end{aligned}$$

کنکور ارشد

(علوم کامپیوتر - دولتی ۹۰)

۱- در یک درخت دودویی غیر تهی، تعداد برگ‌های درخت برابر L می‌باشد. تعداد گره‌های با درجه ۲ برابر کدام است؟

$$\left\lfloor \log_2^{(L+1)} \right\rfloor \quad (4) \qquad 2^L \quad (3) \qquad L+1 \quad (2) \qquad L-1 \quad (1)$$

پاسخ: جواب گزینه ۱ است.

$$n_0 = n_2 + 1 \Rightarrow n_2 = n_0 - 1 = L - 1$$



(مهندسی کامپیوتر - دولتی ۸۶)

۲- کدام یک از گزاره‌های زیر در مورد یک درخت ۵ تائی کامل با ۹۵ گره صحیح است؟

(ریشه گره اول است و در هر عمق گره‌ها به ترتیب از چپ به راست در نظر گرفته شوند)

(۱) این درخت ۷۶ برگ دارد و گره پانزدهم آن پدر گره ۷۲ ام آن است.

(۲) این درخت ۷۶ برگ دارد و گره چهاردهم آن پدر گره ۷۲ ام آن است.

(۳) این درخت ۷۷ برگ دارد و گره چهاردهم آن پدر گره ۷۲ ام آن است.

(۴) این درخت ۷۷ برگ دارد و گره پانزدهم آن پدر گره ۷۲ ام آن است.

پاسخ: جواب گزینه ۱ است.

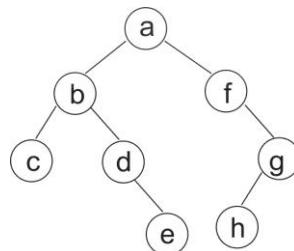
در یک درخت ۵ تائی کامل، تعداد برگ‌ها در درخت با ۹۵ گره، برابر است با:

$$n_0 = n - \left\lceil \frac{n-1}{k} \right\rceil = 95 - \left\lceil \frac{95-1}{5} \right\rceil = 95 - 19 = 76$$

شماره پدر گره ۷۲ ام، برابر است با:

(علوم کامپیوتر - دولتی ۸۷)

۳- نمایش درخت زیر به کدام صورت درست می‌باشد؟



(a(b(c,d),e),f(g(h))) (۲)

(a,b(c,d),e,f(g(h))) (۱)

(a(b(c,d,e)),f(g(h),)) (۴)

(a(b(c,d(e),f(g(h)))) (۳)

پاسخ: جواب گزینه ۴ است.

عبارت $a(b)$ بیانگر این است که b فرزند چپ a است و عبارات $a(b)$ معرف این است که b فرزند راست a می‌باشد. با توجه به این موضوع عبارت $(a(b(c,d,e)),f,g(h))$ معادل درخت داده شده است.

(مهندسی کامپیووتر - آزاد ۸۷)

۴- با ۴ گره چند درخت دودویی متفاوت (از لحاظ فرم درخت و بدون توجه به داده‌های آن) می‌توان ساخت؟

14 (۴)

18 (۳)

42 (۲)

5 (۱)

پاسخ: جواب گزینه ۴ است.

$$\frac{\binom{8}{4}}{5} = \frac{\frac{8!}{4!4!}}{5} = \frac{\frac{8 \times 7 \times 6 \times 5 \times 4!}{4!4!}}{5} = \frac{\frac{8 \times 7 \times 6 \times 5}{4 \times 3 \times 2 \times 1}}{5} = \frac{70}{5} = 14$$

(علوم کامپیووتر - دولتی ۸۶)

۵- خروجی الگوریتم زیر برای یک درخت باینری معادل کدام یک از پیمایش‌های زیر می‌باشد؟

```
void rco(ptr p){
    if ( p !=NULL ) {
        printf( p-> data );
        rco( p -> right );
        rco( p -> left );
    }
}
```

۴) هیچکدام

reverse postorder (۳)

reverse preorder (۲)

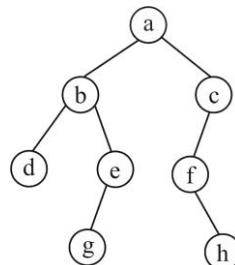
inorder (۱)

پاسخ: جواب گزینه ۳ است.

درخت داده شده به صورت "ریشه-راست-چپ" پیمایش می‌شود که معکوس postorder است.

(علوم کامپیووتر - دولتی ۸۰)

۶- کدام گزینه پیمایش preorder درخت زیر را مشخص می‌کند؟



d b g e a f h c (۲)

a b c d d e f g h (۱)

d g e b h f c a (۴)

a b d e g c f h (۳)

پاسخ: جواب گزینه ۳ است. ترتیب پیمایش در preorder , "ریشه-چپ-راست" می‌باشد.

(مهندسی کامپیوتر - آزاد ۸۷)

۷- درخت دودوئی T در شکل زیر به صورت آرایه نمایش داده شده است. پیمایش inorder , postorder آن کدام است؟

1	2	3	...	6	7	...	12	13	...
A		B		D	E		F	G	

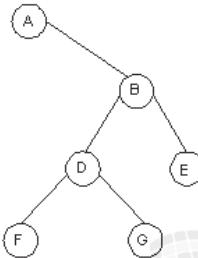
$$\text{inorder} = \text{AFDGBE}, \text{ postorder} = \text{FGDEBA} \quad (1)$$

$$\text{inorder} = \text{ADFGEB}, \text{ postorder} = \text{FGDEBA} \quad (2)$$

$$\text{inorder} = \text{AFDGBE}, \text{ postorder} = \text{FGDBEA} \quad (3)$$

$$\text{inorder} = \text{AFDBEG}, \text{ postorder} = \text{FGDBEA} \quad (4)$$

پاسخ: گزینه ۱ جواب است. با توجه به آرایه داده شده درخت به صورت زیر است:



$$\text{postorder} = \text{FGDEBA} \text{ و } \text{inorder} = \text{AFDGBE}$$

پیمایش میانندی و پسوندی آن برابر است با:

(مهندسی IT - دولتی ۹۰)

۸- نمایش Preorder یک درخت دودوئی به صورت (از چپ به راست) XYZABVFOD است. این درخت گره تک فرزندی ندارد. همچنین گره های DOVBZ مجموعه برگهای درخت را تشکیل می دهند. نمایش Postorder درخت کدام گزینه است؟

$$\text{ZAYVODFBX} \quad (2)$$

$$\text{ZBVAYODFX} \quad (1)$$

$$\text{BVZAYOFDX} \quad (4)$$

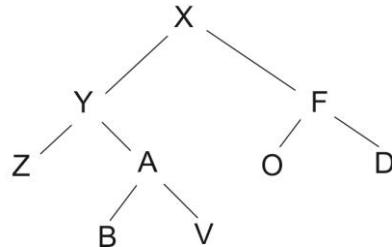
$$\text{ZAYOFVDBX} \quad (3)$$

پاسخ: جواب گزینه ۱ است.

می دانیم که ترتیب ملاقات برگ ها در هر سه پیمایش درخت با هم برابر هستند. بنابراین با توجه اینکه برگ ها مشخص هستند، در پیمایش Preorder داده شده، یعنی XYZABVFOD ، ترتیب ملاقات برگها به صورت ZBVOD می باشد که فقط در گزینه یک این ترتیب رعایت شده است.(به طور مثال در گزینه ۲ و ۳ گره V زودتر از B ملاقات شده و یا در گزینه ۴، گره B زودتر از Z ملاقات شده است).

تذکر: اگر پیمایش preorder یک درخت دودوئی مشخص باشد و درخت دارای گره های تک فرزندی نباشد، پیمایش postorder آن منحصر به فرد می باشد.

روش دوم: با توجه به پیمایش داده شده، X ریشه است. حال چون درخت گره تک فرزندی ندارد، بنابراین Y در چپ X قرار دارد. (چون اگر Y در راست X باشد، آن وقت X تک فرزندی می‌شود). به همین روال درخت را رسم می‌کنیم و شکل زیر حاصل می‌شود:



حال پیمایش Postorder این درخت برابر است با: **Z B V A Y O D F X**

(مهندسی IT - دولتی ۸۸)

۹- تعداد درخت‌های دودویی که پیمایش‌های پیش ترتیب (Postorder) و پس ترتیب (Preorder) آنها برابر با رشته‌های زیر باشد، چقدر است؟

Preorder : a b d e f g c h i j k l

Postorder : e d g f b i h k l j c a

۴) بی‌نهایت درخت

۳) ۸ درخت

۴) ۴ درخت

۱) ۱ درخت

پاسخ: جواب گزینه ۳ است.

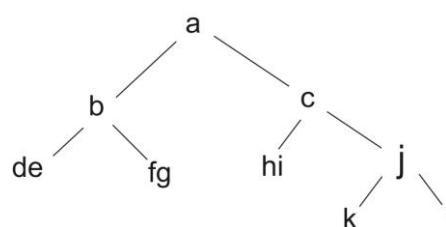
با نگاه به پیمایش‌های داده شده زیر، سه ترکیب دودویی پشت سر هم (de) و (fg) و (hi) در پیمایش preorder وجود دارد که ترکیب عکس آنها یعنی (ed) و (gf) و (ih) در پیمایش postorder موجود است. بنابراین درخت دارای ۳ گره تک فرزندی است. بنابراین تعداد درخت‌های دودویی که دارای پیمایش‌های یکسان برابر $2^3 = 8$ می‌باشد.

Preorder (VLR) : a b **d** e **f** g c **h** i j k l

Postorder (LRV): **e** **d** **g** f b i **h** k l j c a

روش دوم:

با توجه به پیمایش‌های داده شده درخت زیر حاصل می‌شود که وضعیت گره‌های (de) و (fg) و (hi) نسبت به هم مشخص نمی‌باشد.



در این حالت ۸ درخت مختلف می‌توان رسم کرد:

- ۱- گره e در چپ گره d، گره g در چپ گره f و گره i در چپ گره h قرار بگیرد.
- ۲- گره e در چپ گره d، گره g در چپ گره f و گره i در راست گره h قرار بگیرد.
- ۳- گره e در چپ گره d، گره g در راست گره f و گره i در چپ گره h قرار بگیرد.
- ۴- گره e در چپ گره d، گره g در راست گره f و گره i در راست گره h قرار بگیرد.
- ۵- گره e در راست گره d، گره g در چپ گره f و گره i در چپ گره h قرار بگیرد.
- ۶- گره e در راست گره d، گره g در چپ گره f و گره i در راست گره h قرار بگیرد.
- ۷- گره e در راست گره d، گره g در راست گره f و گره i در چپ گره h قرار بگیرد.
- ۸- گره e در راست گره d، گره g در راست گره f و گره i در راست گره h قرار بگیرد.



منتخبی از عناوین آموزشی منتشر شده بر روی فرادرس

عنوان آموزش	برنامه نویسی	مدت زمان تقریبی
آموزش اصول و مبانی برنامه نویسی – کلیک کنید (+)		۲ ساعت
آموزش برنامه نویسی جاوا – کلیک کنید (+)		۲۲ ساعت
آموزش برنامه نویسی PHP – کلیک کنید (+)		۲۹ ساعت
آموزش برنامه نویسی C++ – کلیک کنید (+)		۲۰ ساعت
آموزش پیشرفته C++ (شی گرایی در سی پلاس پلاس) – کلیک کنید (+)		۸ ساعت
آموزش مقدماتی برنامه نویسی سی شارپ (C#) – کلیک کنید (+)		۸ ساعت
مجموعه آموزش های کاربردی برنامه نویسی C# (سی شارپ) – کلیک کنید (+)		۱۴ ساعت
آموزش شی گرایی در سی شارپ (C#) – کلیک کنید (+)		۱۴ ساعت
آموزش برنامه نویسی پایتون – مقدماتی – کلیک کنید (+)		۱۸ ساعت
آموزش تکمیلی برنامه نویسی پایتون – کلیک کنید (+)		۵ ساعت
آموزش برنامه نویسی C – کلیک کنید (+)		۱۳ ساعت
آموزش SQL Server – مقدماتی – کلیک کنید (+)		۱۱ ساعت
آموزش SQL Server – تکمیلی – کلیک کنید (+)		۶ ساعت
آموزش کار با دستورات پایگاه داده در SQL Server – کلیک کنید (+)		۳ ساعت
مجموعه آموزش های برنامه نویسی متلب (MATLAB) – کلیک کنید (+)		۹ ساعت
مجموعه آموزش های برنامه نویسی متلب پیشرفته – کلیک کنید (+)		۷ ساعت
آموزش برنامه نویسی اندروید (Android) – مقدماتی – کلیک کنید (+)		۱۶ ساعت
آموزش برنامه نویسی اندروید (Android) – تکمیلی – کلیک کنید (+)		۲۰ ساعت
آموزش برنامه نویسی اندروید (Android) – پیشرفته – کلیک کنید (+)		۲۶ ساعت
آموزش پروژه محور برنامه نویسی دلفی (Delphi) – کلیک کنید (+)		۳ ساعت
آموزش زبان برنامه نویسی فرتون (Fortran) – مقدماتی – کلیک کنید (+)		۴ ساعت
آموزش مروری زبان برنامه نویسی FORTRAN با مثال های کاربردی – کلیک کنید (+)		۲ ساعت
آموزش پروژه محور اینترنت اشیا (IoT) – کنترل لوازم منزل با شبکه و پیامک و ماژول های SIM و ESP8266 – کلیک کنید (+)		۱۰ ساعت
آموزش برنامه نویسی Swift (سوئیفت) برای برنامه نویسی iOS – کلیک کنید (+)		۵ ساعت
آموزش برنامه نویسی پاسکال (Pascal) – کلیک کنید (+)		۶ ساعت
آموزش برنامه نویسی برای دانش آموزان با اسمال بیسیک (Small Basic) – کلیک کنید (+)		۱۶ ساعت
آموزش برنامه نویسی تصویری به کودکان با زبان اسکرچ (Scratch) – کلیک کنید (+)		۶ ساعت
آموزش محافظت از کدهای نرم افزاری با SmartAssembly (اسمارت اسembly) – کلیک کنید (+)		۱ ساعت

برنامه نویسی (ادامه از صفحه قبل)

عنوان آموزش	مدت زمان تقریبی
آموزش مبانی برنامه نویسی (الگوریتم و فلوچارت) با رویکرد حل مسأله – کلیک کنید (+)	۱۰ ساعت
آموزش پروژه محور Python (پایتون) - ساخت نرم افزار برای Windows و Linux – کلیک کنید (+)	۱۰ ساعت
آموزش برنامه نویسی آردوینو (Arduino) با پایتون (Python) – کلیک کنید (+)	۵ ساعت
آموزش ساخت اپلیکیشن اندروید (Android) با MySQL و PHP – کلیک کنید (+)	۲ ساعت
آموزش مقدماتی زبان برنامه نویسی کاتلین (Kotlin) برای توسعه اندروید (Android) – کلیک کنید (+)	۳ ساعت
آموزش داکر (Docker) برای توسعه دهندگان – کلیک کنید (+)	۱ ساعت
آموزش توسعه وب با زبان برنامه نویسی Go – کلیک کنید (+)	۱ ساعت
آموزش ویژوال بیسیک دات نت (Visual Basic.NET) – مقدماتی – کلیک کنید (+)	۱۳ ساعت
آموزش ویژوال بیسیک دات نت (Visual Basic.NET) – تکمیلی – کلیک کنید (+)	۱۶ ساعت
آموزش تاریخچه گیت (Git) و مقدمات آن – کلیک کنید (+)	۱ ساعت
آموزش گیت (Git) برای مدیریت نسخه توزیع شده – کلیک کنید (+)	۱ ساعت
آموزش زبان برنامه نویسی AWK در لینوکس - پردازش و آنالیز فایل های متنی – کلیک کنید (+)	۵ ساعت
آموزش برنامه نویسی R و نرم افزار RStudio – کلیک کنید (+)	۷ ساعت
آموزش تکمیلی برنامه نویسی R و نرم افزار RStudio – کلیک کنید (+)	۵ ساعت
آموزش مقدماتی ساخت ربات تلگرام با PHP – کلیک کنید (+)	۱ ساعت
آموزش کتابخانه قالب استاندارد (STL) در C++ برای ساده نویسی برنامه ها – کلیک کنید (+)	۹ ساعت
آموزش برنامه نویسی با روش سه لایه به زبان VB.Net – کلیک کنید (+)	۴ ساعت
آموزش بازی سازی در پایتون با کتابخانه Pygame – کلیک کنید (+)	۴ ساعت
آموزش پایتون گرافیکی (رابط های گرافیکی پایتون) – کلیک کنید (+)	۵ ساعت
آموزش ساخت ربات تلگرام با پایتون (Python) – کلیک کنید (+)	۱ ساعت
مجموعه آموزش های طراحی رابط های گرافیکی (GUI) در متلب	۷ ساعت
آموزش تکمیلی طراحی رابط گرافیکی کاربر (GUI) با متلب (MATLAB) – کلیک کنید (+)	۱۴ ساعت
آموزش ساخت ربات تلگرام با متلب (MATLAB) – کلیک کنید (+)	۱ ساعت
آموزش ساخت ربات تلگرام با جاوا (Java) – کلیک کنید (+)	۱ ساعت
آموزش پایگاه داده ها در جاوا – کلیک کنید (+)	۲ ساعت
آموزش کار با فایل ها در جاوا با پکیج Java/IO (به همراه پروژه دفترچه یادداشت) – کلیک کنید (+)	۳ ساعت
آموزش پروژه محور جاوا - تولید QR code و ایجاد و اسکن بارکد (Barcode) – کلیک کنید (+)	۱ ساعت

برنامه نویسی (ادامه از صفحه قبل)

عنوان آموزش	مدت زمان تقریبی
آموزش پروژه محور جاوا (Java) - طراحی و ساخت شبکه اجتماعی - کلیک کنید (+)	۶ ساعت
آموزش طراحی رابط کاربری مدرن در .NET با کامپوننت های جانوس (Janus) - کلیک کنید (+)	۴ ساعت
آموزش ساخت بازی ساده در ویژوال بیسیک - کلیک کنید (+)	۲ ساعت
آموزش پیاده سازی سیستم انبارداری و فروش با ویژوال بیسیک دات نت (VB.NET) - مقدماتی - کلیک کنید (+)	۱۰ ساعت
آموزش پیاده سازی سیستم انبارداری و فروش با ویژوال بیسیک دات نت (VB.NET) - تکمیلی - کلیک کنید (+)	۱۲ ساعت
آموزش مقدماتی پیاده سازی مسائل بهینه سازی در پایتون (Python) - کلیک کنید (+)	۴ ساعت
آموزش الگوهای طراحی (Design Patterns) در پایتون (Python) - کلیک کنید (+)	۱۲ ساعت
آموزش رایگان نکات و ترفندهای متلب - کلیک کنید (+)	۲ ساعت
مجموعه آموزش های برنامه نویسی متلب برای علوم و مهندسی - کلیک کنید (+)	۱۴ ساعت
آموزش فریم ورک Spring در جاوا - کلیک کنید (+)	۶ ساعت
آموزش ORM هایبرنیت (Hibernate) جاوا - کلیک کنید (+)	۱۸ ساعت
آموزش آشنایی با LINQ to SQL در C# - کلیک کنید (+)	۲ ساعت
آموزش برنامه نویسی تحت شبکه با سی شارپ در قالب پروژه - کلیک کنید (+)	۱ ساعت
آموزش برنامه نویسی با روش سه لایه به زبان سی شارپ - کلیک کنید (+)	۴ ساعت
آموزش Cryptography در دات نت - کلیک کنید (+)	۳ ساعت
آموزش قفل نرم افزاری در C# از طریق رجیستری - کلیک کنید (+)	۴ ساعت
آموزش پروژه محور سی شارپ (C#) - سیستم مدیریت آموزشگاه - کلیک کنید (+)	۱۹ ساعت
آموزش پروژه محور سی شارپ (C#) - سیستم مدیریت حسابداری و انبارداری - کلیک کنید (+)	۱۴ ساعت
آموزش برنامه نویسی سورس کدهای الگوریتمی در سی شارپ - کلیک کنید (+)	۷ ساعت
آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم هتلداری - کلیک کنید (+)	۱۰ ساعت
آموزش ساخت سیستم آزمون گیر در سی شارپ - کلیک کنید (+)	۴ ساعت
آموزش ارسال و دریافت پیامک با مودم در سی شارپ (C#) - کلیک کنید (+)	۳ ساعت
آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم مدیریت آتلیه - کلیک کنید (+)	۱۱ ساعت
آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم نوبت دهی - کلیک کنید (+)	۷ ساعت
آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم صدور بلیط - کلیک کنید (+)	۹ ساعت
آموزش رسم نمودار در سی شارپ (C#) - کلیک کنید (+)	۳ ساعت
آموزش پروژه محور سی شارپ (C#) - سیستم مدیریت بانک - کلیک کنید (+)	۷ ساعت
آموزش ساخت بازی با سی شارپ (C#) - کلیک کنید (+)	۵ ساعت
آموزش Metaprogramming (برنامه نویسی متا) در پایتون - کلیک کنید (+)	۲ ساعت

برنامه نویسی (ادامه از صفحه قبل)

مدت زمان تقریبی	عنوان آموزش
۱۲ ساعت	آموزش کاربردی Entity Framework در سی شارپ (C#) - کلیک کنید (+)
۶ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم مدیریت کارمندان - کلیک کنید (+)
۶ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم مدیریت چک های بانکی - کلیک کنید (+)
۴ ساعت	آموزش پروژه محور سی شارپ (C#) - شبیه سازی دستگاه عابر بانک - کلیک کنید (+)
۲ ساعت	آموزش گزارش گیری با کریستال ریپورت و استیمول سافت (Stimulsoft) در سی شارپ (C#) - کلیک کنید (+)
۲ ساعت	آموزش پایگاه داده اس کیو لایت (SQLite) در سی شارپ (C#) - کلیک کنید (+)
۶ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی نرم افزار حسابداری شخصی - کلیک کنید (+)
۱۵ ساعت	آموزش پروژه محور سی شارپ (C#) - پیاده سازی سیستم رستوران آنلاین - کلیک کنید (+)
۳ ساعت	آموزش کار با بانک اطلاعاتی اکسس (Access) در سی شارپ (C#) - کلیک کنید (+)
۲ ساعت	آموزش کار با فایل ها و دایرکتوری در # C# (سی شارپ) - کلیک کنید (+)
۳ ساعت	آموزش پروژه محور سی شارپ C# - پیاده سازی نرم افزار FileReader (فیدریدر) - کلیک کنید (+)
۴ ساعت	آموزش کار با دستورات گرافیکی در سی شارپ (C#) - کلیک کنید (+)
۳ ساعت	آموزش مقدماتی WPF در سی شارپ (C#) برای ساخت و اجرای اینترفیس - کلیک کنید (+)
۴ ساعت	آموزش WPF در سی شارپ برای ساخت و اجرای اینترفیس - پیشرفته - کلیک کنید (+)
۲ ساعت	آموزش WPF در سی شارپ (C#) برای ساخت و اجرای اینترفیس - تکمیلی - کلیک کنید (+)
۱ ساعت	آموزش ساخت ربات تلگرام با سی شارپ (C#) - کلیک کنید (+)
۱۲ ساعت	آموزش پروژه محور پایگاه داده LocalDB در سی شارپ (C#) - سیستم مدیریت کارمندان - کلیک کنید (+)
۵ ساعت	آموزش کتابخانه Spire.Doc در سی شارپ (C#) برای ایجاد و ویرایش فایل Word - کلیک کنید (+)
۱ ساعت	آموزش پروژه محور سی شارپ (C#) - ساخت برنامه دانلودگر - کلیک کنید (+)

علوم کامپیوتر

عنوان آموزش	مدت زمان تقریبی
آموزش ساختمان داده‌ها – کلیک کنید (+)	۱۰ ساعت
آموزش پیشرفته ساختمان داده (همراه با حل نمونه سوالات کنکور ارشد و دکتری) – کلیک کنید (+)	۸ ساعت
آموزش ساختمان داده‌ها (مرور – تست کنکور ارشد) – کلیک کنید (+)	۲۰ ساعت
آموزش سیستم‌های عامل – کلیک کنید (+)	۱۱ ساعت
آموزش سیستم عامل (مرور – تست کنکور ارشد) – کلیک کنید (+)	۱۲ ساعت
آموزش پایگاه داده‌ها – کلیک کنید (+)	۸ ساعت
آموزش پایگاه داده‌ها (مرور – تست کنکور ارشد) – کلیک کنید (+)	۸ ساعت
آموزش طراحی و پیاده‌سازی زبان‌های برنامه‌سازی – کلیک کنید (+)	۱۰ ساعت
آموزش طراحی و پیاده‌سازی زبان‌های برنامه‌سازی (مرور – تست کنکور ارشد) – کلیک کنید (+)	۹ ساعت
آموزش طراحی الگوریتم – کلیک کنید (+)	۱۵ ساعت
آموزش طراحی الگوریتم به همراه حل مثال‌های عملی – کلیک کنید (+)	۸ ساعت
آموزش روش‌های حل روابط بازگشتی – کلیک کنید (+)	۴ ساعت
آموزش رابطه‌های بازگشتی در طراحی الگوریتم و ساختمان گسسته (مرور – تست کنکور ارشد) – کلیک کنید (+)	۲ ساعت
آموزش طراحی الگوریتم (مرور – تست کنکور ارشد) – کلیک کنید (+)	۱۵ ساعت
آموزش روش تقسیم و حل در طراحی الگوریتم (مرور – تست کنکور کارشناسی ارشد) – کلیک کنید (+)	۲ ساعت
آموزش مبانی منطق و نظریه مجموعه‌ها – کلیک کنید (+)	۱۸ ساعت
آموزش مدار منطقی (مرور – تست کنکور ارشد) – کلیک کنید (+)	۱۵ ساعت
آموزش مبانی الکترونیک دیجیتال - بخش اول – کلیک کنید (+)	۱۲ ساعت
آموزش مبانی الکترونیک دیجیتال - بخش دوم – کلیک کنید (+)	۱۱ ساعت
آموزش معماری کامپیوتر با رویکرد حل مساله - مقدماتی - کلیک کنید (+)	۲۰ ساعت
آموزش معماری کامپیوتر با رویکرد حل مساله - پیشرفته - کلیک کنید (+)	۱۴ ساعت
آموزش نظریه زبان‌ها و ماشین‌ها – کلیک کنید (+)	۹ ساعت
آموزش نظریه زبان‌ها و ماشین‌ها (مرور – تست کنکور ارشد) – کلیک کنید (+)	۸ ساعت
آموزش شبیه سازی و تست نظریه زبان‌ها و ماشین در JFLAP – کلیک کنید (+)	۳ ساعت
آموزش یادگیری عمیق در پردازش کلام – کلیک کنید (+)	۴ ساعت
آموزش ساختمان گسسته با رویکرد حل مساله – کلیک کنید (+)	۱۶ ساعت
آموزش ساختمان گسسته (مرور و حل تست‌های کنکور کارشناسی ارشد) – کلیک کنید (+)	۱۲ ساعت
آموزش آمار و احتمال مهندسی – کلیک کنید (+)	۱۳ ساعت
آموزش آمار و احتمال مهندسی (حل تمرین و تست کنکور ارشد) – کلیک کنید (+)	۱۴ ساعت

علوم کامپیوتر (ادامه از صفحه قبل)

عنوان آموزش	مدت زمان تقریبی
آموزش شبکه های کامپیوتری ۱ - کلیک کنید (+)	۱۹ ساعت
آموزش شبکه های کامپیوتری ۲ - همراه با حل مسئله - کلیک کنید (+)	۲۳ ساعت
آموزش شبیه سازی شبکه های مبتنی بر سیسکو (Cisco) در نرم افزار Boson NetSim - کلیک کنید (+)	۱ ساعت
آموزش طراحی کامپایلر - کلیک کنید (+)	۱۵ ساعت
آموزش طراحی کامپایلر (مرور و حل تست های کنکور کارشناسی ارشد) - کلیک کنید (+)	۱۵ ساعت
آموزش ذخیره و بازیابی اطلاعات - کلیک کنید (+)	۸ ساعت
آموزش ساختمن داده ها همراه با پیاده سازی در C++ - کلیک کنید (+)	۲۲ ساعت
آموزش انتقال داده ها در شبکه های کامپیوتری و اینترنت به همراه حل مسائل - کلیک کنید (+)	۹ ساعت
آموزش مقدماتی Hadoop (هدوپ) برای تجزیه و تحلیل کلان داده - کلیک کنید (+)	۵ ساعت
آموزش نظریه صف (Queueing theory) - کلیک کنید (+)	۲۲ ساعت
آموزش گرافیک کامپیوتری - کلیک کنید (+)	۴ ساعت
آموزش گرافیک کامپیوتری با OpenGL - کلیک کنید (+)	۱۶ ساعت
آموزش مهندسی نرم افزار ۱ - کلیک کنید (+)	۱۲ ساعت
آموزش زبان تخصصی مهندسی کامپیوتر - کلیک کنید (+)	۱۰ ساعت
آموزش مروری بر پیچیدگی محاسبات (Computational Complexity) - کلیک کنید (+)	۱ ساعت
آموزش تبدیل فلوچارت به کد با Flowgorithm - کلیک کنید (+)	۱ ساعت
آموزش نرم افزار RAPTOR برای ترسیم فلوچارت - کلیک کنید (+)	۱ ساعت
آموزش حل سوالات کنکور کارشناسی ارشد کامپیوتر - ۹۴ - کلیک کنید (+)	۲ ساعت
آموزش مرور و حل تست های کنکور کارشناسی ارشد معماری کامپیوتر - کلیک کنید (+)	۱۷ ساعت
آموزش برنامه نویسی اسمبلي - کلیک کنید (+)	۱۰ ساعت
آموزش نظریه گراف و کاربردها - کلیک کنید (+)	۱۵ ساعت
آموزش ترکیبیات و کاربردها - کلیک کنید (+)	۱۵ ساعت
آموزش نظریه بازی ها و کاربرد آن در اقتصاد - کلیک کنید (+)	۸ ساعت
آموزش مدل سازی UML با نرم افزار Rational Rose - کلیک کنید (+)	۴ ساعت
آموزش الگوریتم موازی و پردازش موازی - کلیک کنید (+)	۱۲ ساعت
آموزش ترکیب اطلاعات (Information Fusion) - کلیک کنید (+)	۲۰ ساعت
آموزش نهان نگاری دیجیتال (Digital watermarking) - کلیک کنید (+)	۳ ساعت
آموزش نهان نگاری و پیاده سازی آن در سی شارپ (C#) - کلیک کنید (+)	۳ ساعت
آموزش واترمارکینگ در نهان نگاری با سی شارپ (C#) - کلیک کنید (+)	۱ ساعت
آموزش شبکه های پیچیده پویا (Complex Dynamical Networks) - کلیک کنید (+)	۱ ساعت

علوم کامپیوتر (ادامه از صفحه قبل)

عنوان آموزش	مدت زمان تقریبی
آموزش تخمین تلاش لازم برای توسعه نرم افزارها با متناب - کلیک کنید (+)	۳ ساعت
آموزش توسعه نرم افزار با متد ICONIX و زبان مدل سازی UML - کلیک کنید (+)	۴ ساعت
آموزش نگارش پروپوزال، پایان نامه و ارائه - ویژه پردازش تصویر و بینایی کامپیوتر - کلیک کنید (+)	۱ ساعت

هوش مصنوعی

عنوان آموزش	مدت زمان تقریبی
آموزش هوش مصنوعی - کلیک کنید (+)	۴ ساعت
آموزش هوش مصنوعی - تکمیلی - کلیک کنید (+)	۱۹ ساعت
آموزش هوش مصنوعی توزیع شده - کلیک کنید (+)	۲۴ ساعت
آموزش هوش مصنوعی (مرور و حل تست کنکور ارشد) - کلیک کنید (+)	۱۵ ساعت
مجموعه آموزش های شبکه های عصبی مصنوعی در مطلب - کلیک کنید (+)	۲۸ ساعت
مجموعه آموزش های کاربردی شبکه های عصبی مصنوعی - کلیک کنید (+)	۱۰ ساعت
آموزش طراحی سیستم های فازی عصبی یا ANFIS با استفاده از الگوریتم های فرآیندکاری و تکاملی - کلیک کنید (+)	۲ ساعت
آموزش کاربردی مکاترونیک - محاسبات پایه طراحی درب های خودکار - کلیک کنید (+)	۵ ساعت
آموزش شبکه عصبی GMDH به همراه پیاده سازی عملی در مطلب - کلیک کنید (+)	۴ ساعت
آموزش شبکه های عصبی گازی به همراه پیاده سازی عملی در مطلب - کلیک کنید (+)	۳ ساعت
آموزش طبقه بندی و بازناسی الگو با شبکه های عصبی LVQ در مطلب - کلیک کنید (+)	۳ ساعت
آموزش یادگیری عمیق (Deep learning) - کلیک کنید (+)	۳ ساعت
آموزش برنامه نویسی یادگیری عمیق با پایتون (TensorFlow و Keras) - کلیک کنید (+)	۳ ساعت
مجموعه آموزش های سیستم های فازی در مطلب - کلیک کنید (+)	۲۱ ساعت
مجموعه آموزش های تئوری و عملی الگوریتم ژنتیک - کلیک کنید (+)	۱۴ ساعت
آموزش پیاده سازی ترکیب الگوریتم ژنتیک و PSO در مطلب - کلیک کنید (+)	۱ ساعت
مجموعه آموزش های بهینه سازی چند هدفه در مطلب - کلیک کنید (+)	۱۹ ساعت
آموزش حل مساله فروشنده دوره گرد با استفاده از الگوریتم ژنتیک - کلیک کنید (+)	۲ ساعت
آموزش انتخاب ویژگی یا Feature Selection - کلیک کنید (+)	۲ ساعت
آموزش انتخاب ویژگی با استفاده از الگوریتم های فرآیندکاری و تکاملی - کلیک کنید (+)	۴ ساعت
آموزش تخمین خطای طبقه بندی یا Classifier Error Estimation - کلیک کنید (+)	۲ ساعت
آموزش کاهش تعداد رنگ تصاویر با استفاده از روش های خوش بندی هوشمند - کلیک کنید (+)	۱ ساعت

مهندسی برق

عنوان آموزش	مدت زمان تقریبی
آموزش الکترونیک ۱ - کلیک کنید (+)	۱۱ ساعت
آموزش حل تمرین الکترونیک ۱ - کلیک کنید (+)	۲۵ ساعت
آموزش الکترونیک ۱ (مرور و حل تست) - کلیک کنید (+)	۱۱ ساعت
آموزش الکترونیک ۲ - کلیک کنید (+)	۸ ساعت
آموزش حل تمرین الکترونیک ۲ - کلیک کنید (+)	۲۵ ساعت
آموزش الکترونیک ۲ (مرور و حل تست) - کلیک کنید (+)	۱۲ ساعت
آموزش الکترونیک ۳ - کلیک کنید (+)	۱۵ ساعت
آموزش الکترونیک ۳ (مرور و حل مساله) - کلیک کنید (+)	۱۲ ساعت
آموزش مبانی مهندسی برق ۱ - کلیک کنید (+)	۱۴ ساعت
آموزش مبانی مهندسی برق ۱ (مرور و حل مساله) - کلیک کنید (+)	۱۴ ساعت
آموزش مبانی مهندسی برق ۲ - کلیک کنید (+)	۱۶ ساعت
آموزش مدارهای الکتریکی ۱ - کلیک کنید (+)	۲۴ ساعت
آموزش حل تمرین مدارهای الکتریکی ۱ - کلیک کنید (+)	۱۷ ساعت
آموزش مدارهای الکتریکی ۱ (مرور و حل تست) - کلیک کنید (+)	۱۴ ساعت
آموزش مدارهای الکتریکی ۲ - کلیک کنید (+)	۱۱ ساعت
آموزش مدارهای الکتریکی ۲ (حل تمرین) - کلیک کنید (+)	۱۸ ساعت
آموزش مدارهای الکتریکی ۲ (مرور و حل تست) - کلیک کنید (+)	۱۳ ساعت
آموزش هندسه فینسلر (Finsler Geometry) - کلیک کنید (+)	۹ ساعت
آموزش زبان تخصصی مهندسی برق (همراه با حل نمونه سوالات کنکور ارشد) - کلیک کنید (+)	۱۶ ساعت
آموزش مروری ماشین های الکتریکی ۱ - کلیک کنید (+)	۲ ساعت
آموزش ماشین های الکتریکی ۱ - کلیک کنید (+)	۱۴ ساعت
آموزش ماشین های الکتریکی ۱ (مرور و حل تست) - کلیک کنید (+)	۱۶ ساعت
آموزش ماشین های الکتریکی ۲ - کلیک کنید (+)	۱۷ ساعت
آموزش ماشین های الکتریکی ۲ (مرور و حل تست کنکور ارشد) - کلیک کنید (+)	۱۲ ساعت
آموزش ماشین های الکتریکی ۳ - کلیک کنید (+)	۱۱ ساعت
آموزش مدارهای مخابراتی - کلیک کنید (+)	۹ ساعت
آموزش ماشین های الکتریکی مخصوص - کلیک کنید (+)	۹ ساعت
آموزش کنترل موتورهای الکتریکی صنعتی ۱ - کلیک کنید (+)	۱۴ ساعت
آموزش کنترل موتورهای الکتریکی صنعتی ۲ - کلیک کنید (+)	۱۷ ساعت
آموزش کنترل صنعتی - حل مساله - کلیک کنید (+)	۱۵ ساعت

مهندسی برق (ادامه از صفحه قبل)

عنوان آموزش	مدت زمان تقریبی
مجموعه آموزش های مدارهای منطقی (طراحی دیجیتال) – کلیک کنید (+)	۹ ساعت
آموزش مبانی الکترونیک دیجیتال - بخش اول – کلیک کنید (+)	۱۲ ساعت
آموزش مبانی الکترونیک دیجیتال - بخش دوم – کلیک کنید (+)	۱۱ ساعت
آموزش الکترومغناطیس مهندسی – کلیک کنید (+)	۲۰ ساعت
آموزش الکترومغناطیس مهندسی (مروار و حل تست کنکور ارشد) – کلیک کنید (+)	۱۸ ساعت
آموزش الکترومغناطیس ۱ – کلیک کنید (+)	۹ ساعت
آموزش مقدماتی شبکه های هوشمند (Smart grid) انرژی الکتریکی – کلیک کنید (+)	۳ ساعت
آموزش کاربردی مکاترونیک - محاسبات پایه طراحی درب های خودکار – کلیک کنید (+)	۵ ساعت
آموزش کامسول مولتی فیزیکس - COMSOL Multiphysics (مباحث منتخب) – کلیک کنید (+)	۲ ساعت
آموزش نرم افزار COMSOL Multiphysics برای پدیده های الکترومغناطیس – کلیک کنید (+)	۴ ساعت
آموزش شبیه سازی مدولاسیون دیجیتال در متلب (MATLAB) – کلیک کنید (+)	۸ ساعت
آموزش نرم افزار سیلواکو (SILVACO) برای شبیه سازی افزارهای نیمه رسانا – کلیک کنید (+)	۴ ساعت

ابزار مدیریت وبسایت

عنوان آموزش	مدت زمان تقریبی
آموزش کار با cPanel – کلیک کنید (+)	۱ ساعت
آموزش کار با کنترل پنل plesk برای ارائه خدمات وب – کلیک کنید (+)	۱ ساعت
آموزش مدیریت هاست با دایرکت ادمین (DirectAdmin) – کلیک کنید (+)	۱ ساعت
آموزش جوملا (Joomla) برای راه اندازی و مدیریت وب سایت – کلیک کنید (+)	۳ ساعت
آموزش نرم افزار Mobirise برای ساخت وب سایت بدون کدنویسی – کلیک کنید (+)	۱ ساعت
آموزش مقدماتی Oracle APEX برای تولید و توسعه نرم افزارهای تحت وب – کلیک کنید (+)	۸ ساعت
آموزش وردپرس (WordPress) – مقدماتی – کلیک کنید (+)	۱۱ ساعت
مجموعه آموزش های راه اندازی سایت و کار با وردپرس – تکمیلی – کلیک کنید (+)	۶ ساعت
آموزش گوگل آنالیتیکس (Google Analytics) برای تحلیل آمار وب سایت – کلیک کنید (+)	۳ ساعت
آموزش آنالیز و بهینه سازی سایت با GTmetrix – کلیک کنید (+)	۲ ساعت
آموزش تست نفوذ در وب و راه های مقابله با آن – کلیک کنید (+)	۱۰ ساعت

طراحی و گرافیک

عنوان آموزش	مدت زمان تقریبی
آموزش فتوشاپ (Photoshop) – مقدماتی – کلیک کنید (+)	۱۲ ساعت
مجموعه آموزش های کاربردی طراحی و گرافیک با نرم افزار کورل (CorelDRAW) – کلیک کنید (+)	۱۴ ساعت
آموزش کورل پیشرفته – کلیک کنید (+)	۱۰ ساعت
آموزش نرم افزار طراحی گرافیکی Adobe Illustrator – کلیک کنید (+)	۱۰ ساعت
آموزش نرم افزار طراحی گرافیکی Adobe Illustrator – تکمیلی – کلیک کنید (+)	۵ ساعت
آموزش مقایسه نرم افزارهای گرافیکی Photoshop , CorelDRAW , Illustrator – کلیک کنید (+)	۱ ساعت
آموزش مقدماتی طراحی کاراکتر در Adobe Illustrator (کاراکتر پلیس) – کلیک کنید (+)	۲ ساعت
آموزش نرم افزار Blender برای طراحی و مدل سازی سه بعدی – کلیک کنید (+)	۸ ساعت
آموزش مدل سازی کاراکتر در بلندر - ایده پردازی تاریگ – کلیک کنید (+)	۱۴ ساعت
آموزش طراحی فلت (Flat design) با کورل (CorelDRAW) – کلیک کنید (+)	۱ ساعت
آموزش فتوشاپ برای استفاده در صنعت چاپ و تبلیغات – کلیک کنید (+)	۳ ساعت
آموزش تایپوگرافی (Typography) - سایه زدن حروف در فتوشاپ (Photoshop) – کلیک کنید (+)	۳ ساعت
آموزش پروژه محور فتوشاپ (Photoshop) - ساخت ماسک (Mask) فانتزی – کلیک کنید (+)	۱ ساعت
آموزش آشنایی با پنل Adjustments در فتوشاپ – کلیک کنید (+)	۲ ساعت
آموزش ساخت جلوه های ویژه در فتوشاپ – کلیک کنید (+)	۳ ساعت
آموزش Adobe Lightroom (ادوی لایت روم) برای ویرایش و روتوش تصاویر – کلیک کنید (+)	۶ ساعت
آموزش Adobe Dimension برای ایجاد تصاویر سه بعدی محصولات و تبلیغات – کلیک کنید (+)	۲ ساعت
آموزش Adobe Bridge (ادوبی بربج) برای مدیریت فایل های چند رسانه ای – کلیک کنید (+)	۲ ساعت
آموزش ایندیزاین (InDesign) برای چاپ و صفحه آرایی کتاب و مجلات – کلیک کنید (+)	۴ ساعت
آموزش نرم افزار SketchBook Pro (اسکچ بوک) برای طراحی آناتومی حیوانات – کلیک کنید (+)	۲۲ ساعت
آموزش طراحی مفهومی (Conceptual) خودرو با SketchBook – کلیک کنید (+)	۳ ساعت
آموزش نقاشی سه بعدی با نرم افزار Paint 3D ویندوز – کلیک کنید (+)	۳ ساعت
آموزش مبانی هنرهای تجسمی ۱ – کلیک کنید (+)	۳ ساعت
آموزش اصول طراحی نشانه (Sign Design) – کلیک کنید (+)	۱ ساعت
آموزش نقاشی دیجیتال با Photoshop و Mischief – کلیک کنید (+)	۹ ساعت
آموزش طراحی لوگو با Adobe Illustrator – کلیک کنید (+)	۲ ساعت
آموزش طراحی محیط بازی های دو بعدی با نرم افزار Adobe Illustrator و Photoshop – کلیک کنید (+)	۳ ساعت

طراحی و گرافیک (ادامه از صفحه قبل)

عنوان آموزش	مدت زمان تقریبی
آموزش طراحی لوگو - آشنایی با مفاهیم و ایده پردازی (همراه با مثال عملی) - کلیک کنید (+)	۱ ساعت
آموزش طراحی برچسب و کاور DVD و CD - کلیک کنید (+)	۲ ساعت
آموزش ابزار سه بعدی در فتوشاپ (Photoshop) - کلیک کنید (+)	۵ ساعت
آموزش فتوشاپ سه بعدی - طراحی و متحرک سازی - کلیک کنید (+)	۲ ساعت
آموزش موتور رندر آرنولد (Arnold Renderer) برای ایجاد جلوه های ویژه در ۳DS Max - کلیک کنید (+)	۲ ساعت
آموزش کاربرد فتوشاپ در معماری - کلیک کنید (+)	۵ ساعت

روباتیک

عنوان آموزش	مدت زمان تقریبی
آموزش رباتیک و ربات های سری - کلیک کنید (+)	۲ ساعت
آموزش برنامه نویسی آردوینو (Arduino) با محوریت پروژه های رباتیک - مقدماتی - کلیک کنید (+)	۹ ساعت
آموزش برنامه نویسی آردوینو (Arduino) - تکمیلی - کلیک کنید (+)	۷ ساعت
آموزش سینماتیک مستقیم و معکوس روبات ها - کلیک کنید (+)	۳ ساعت
آموزش مبانی ربات های برنامه پذیر - کلیک کنید (+)	۵ ساعت
آموزش شبیه سازی و کنترل ربات با اندروید (Android) - کلیک کنید (+)	۱۶ ساعت
آموزش مکاترونیک کاربردی ۱ - کلیک کنید (+)	۱۳ ساعت
آموزش مکاترونیک کاربردی ۲ - کلیک کنید (+)	۷ ساعت
آموزش کاربردی مکاترونیک - محاسبات پایه طراحی درب های خودکار - کلیک کنید (+)	۵ ساعت
آموزش برد آردوینو (Arduino) با انجام پروژه های عملی - کلیک کنید (+)	۱۹ ساعت
آموزش پروژه محور آردوینو با سی شارپ و اندروید - کنترل موتور پله ای - کلیک کنید (+)	۵ ساعت
آموزش پیاده سازی سیستم های کنترلی با آردوینو - کلیک کنید (+)	۵ ساعت

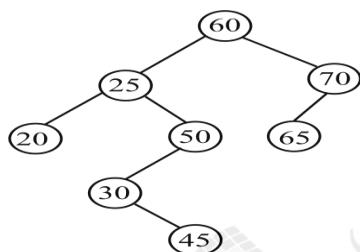
فصل ۷: درخت های جستجو

درخت هایی که در این فصل مطالعه می کنیم، عبارتند از:
B-Tree ، قرمز- سیاه، مرتبه آماری ، AVL و 2-3 ،

درخت جستجوی دودویی (BST)

درختی را جستجوی دودویی(Binary Search Tree) می نامند که :

- ۱- عناصر زیر درخت چپ، کوچکتر از ریشه باشند.
- ۲- عناصر زیر درخت راست، بزرگ‌تر از ریشه باشند.
- ۳- زیر درختان چپ و راست ، درختان جستجوی دودویی باشند.
- ۴- گره با عنصر تکراری نداشته باشد.



مثال

آیا درخت داده شده یک BST را نشان می دهد؟

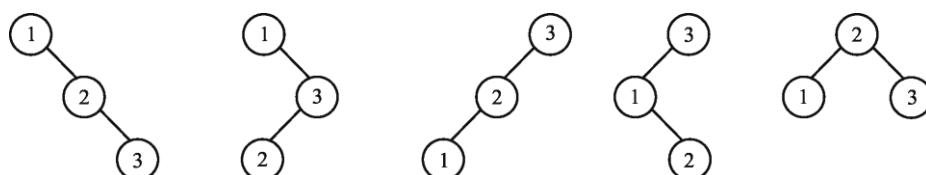
پاسخ:

بله، چون تمامی داده های واقع در سمت چپ ریشه، دارای مقدار کمتر از 60 و تمامی داده های واقع در سمت راست ریشه، دارای مقداری بزرگ‌تر از 60 می باشند. همچنین زیر درختان چپ و راست ریشه خود یک BST می باشند.

مثال

به چند حالت می توان با وارد کردن مقادیر 1,2,3 به هر ترتیب دلخواه در یک BST تهی، یک BST با سه گره ساخت؟ پاسخ:

ترتیب های ممکن برای ورود سه عدد برابر 1, 2, 3 می باشند که BST ساخته شده به ازای هر ترتیب برابر است با:



ترتیب 213 و 231 منجر به یک درخت متمایز می‌شوند. بنابراین 5 درخت متمایز می‌توان ایجاد کرد. البته می‌توان با قرار دادن

$$\text{مقدار } 3 \text{ در رابطه } \frac{1}{n+1} \binom{2n}{n} \text{ به جواب رسید.}$$

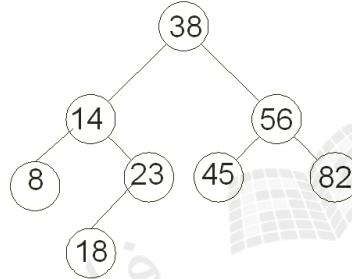
 ارتفاع درخت جستجوی دودویی به طور متوسط برابر $\log_2 n$ و در بدترین حالت برابر n می‌باشد.

 در یک BST، با n گره متمایز که خاصیت MaxHeap هم دارد، داریم: $h = o(n)$

مثال

در BST که به صورت آرایه نمایش داده شده، آیا می‌توان به جای عدد 56 عدد 35 قرار داد؟
38, 14, 56, 8, 23, 45, 82, -, -, 18

پاسخ: درخت BST به صورت زیر است:



بنابراین مشخص است که اگر به جای عدد 56 عدد 35 قرار گیرد، درخت BST نخواهد بود، چون عدد 35 از ریشه کوچکتر است و نمی‌تواند در سمت راست ریشه قرار بگیرد.

عملیات بر روی یک BST

می‌توان عملیاتی چون جستجو، درج، حذف و مرتب سازی را بر روی یک درخت جستجوی دودویی انجام داد.

جستجوی یک عنصر در BST

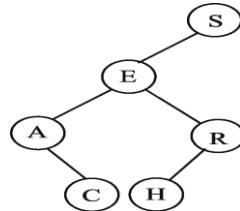
برای جستجوی عنصری در درخت جستجوی دودویی، ابتدا مقدار کلید عنصر مورد نظر با ریشه مقایسه می‌شود، اگر برابر باشد به نتیجه رسیده‌ایم، در غیر این صورت اگر کمتر از مقدار ریشه باشد، زیردرخت چپ و اگر بزرگتر از مقدار ریشه باشد، زیردرخت راست را به صورت بازگشتی جستجو می‌نماییم.

مثال

یک درخت جستجوی باینری با کلمه "SEARCH" بسازید. تعداد متوسط مقایسه برای پیدا کردن یک کاراکتر در این درخت کدام است؟

پاسخ:

درخت جستجوی ساخته شده با کلمه SEARCH با توجه به ترتیب حروف الفبا به صورت زیر است. ابتدا S در ریشه قرار گرفته و سپس E چون کوچکتر است، در سمت چپ قرار گرفته و به همین روال درخت ساخته می‌شود.



برای پیدا کردن S به یک مقایسه نیاز داریم و برای پیدا کردن E به دو مقایسه نیاز داریم(ابتدا با S مقایسه شده و چون کوچکتر است به چپ S می‌رویم و با E مقایسه می‌کنیم)، برای پیدا کردن A و یا R به سه مقایسه و برای C و یا H به چهار مقایسه نیاز داریم. بنابراین متوسط تعداد مقایسه‌ها برابر است با:

$$\frac{1+2+3+3+4+4}{6} = \frac{17}{6} = 2.83$$

جستجوی موفق(یا ناموفق) در هر درخت دودویی به طور میانگین به اندازه $O(\lg n)$ طول می‌کشد.

اما جستجو در بدترین حالت، به اندازه $O(n)$ طول می‌کشد. (اگر درخت یک مسیر باشد).

جستجوی در BST

```

search(t , key){
  while( t != NULL and key != t->data)
  {
    if ( key < t->data)
      t = t->left;
    else
      t = t->right;
  }
  return t;
}
  
```

جستجوی بازگشتی در BST

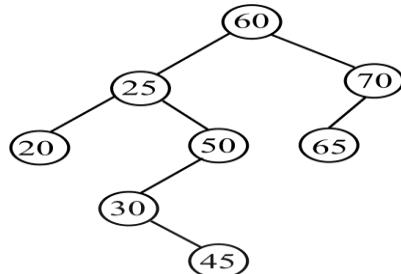
```

search( t , key){
  if (t==NULL or key == t->data)
    return t;
  if (key < t->data)
    return search(t->left , key);
  else
    return search(t->right , key);
}
  
```

دنباله جستجو

در یک BST یک "دنباله جستجوی کلید x "، دنباله ای از برچسب‌های گره‌های آن درخت است که در یافتن موفق(یا ناموفق) x ملاقات می‌شوند. برای مثال، در درخت زیر، دنباله جستجو 45 برابر است با:

$<60, 25, 50, 30, 45>$



شرط لازم و کافی برای این که یک دنباله از کلیدها، یک دنباله جستجوی درست در یک BST با کلیدهای متمایز باشد، این است که برای هر عنصر این دنباله با کلید x ، همه عناصر بعدی یا از x کمتر باشند و یا بزرگتر.



مثال

فرض کنید اعداد 1 تا 1000 در یک BST ذخیره شده‌اند. آیا دنباله زیر، می‌تواند دنباله جستجوی عدد 363 باشد؟

$<925, 202, 911, 240, 912, 245, 363>$

پاسخ: خیر - بعد از ملاقات گره 911 به گره 240 رفته ایم یعنی از 911 به فرزند (و زیر درخت) سمت چپ رفته ایم، اما در ادامه دنباله به 912 بر می‌خوریم که باید در زیر درخت سمت چپ 911 می‌بود که تناقض است. با توجه به نکته قبل، شرایط برای 925 برقرار است، چون همه عناصر بعدی از آن کوچکترند. شرایط برای 202 نیز برقرار است، چون همه عناصر بعدی از آن بزرگترند. اما برای 911 برقرار نیست. (بعضی بزرگتر و بعضی کوچکترند)

دنباله زیر، یک دنباله جستجوی درست برای 363 می‌باشد:

$<2, 252, 401, 398, 330, 344, 397, 363>$

الگوریتم تشخیص درست بودن یک دنباله جستجو از مرتبه $O(n)$ می‌باشد.

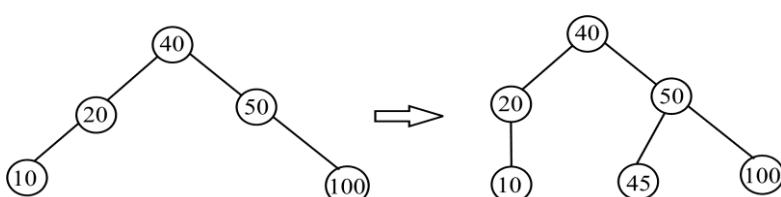
درج یک عنصر به BST

برای اضافه کردن گره ای به درخت جستجوی دودویی، کلید مورد نظر ابتدا با ریشه مقایسه شده و در صورت کوچکتر بودن، به زیر درخت چپ و در صورت بزرگتر بودن به زیر درخت راست اضافه می‌شود.



مثال

اضافه کردن گره 45 به درخت زیر:



چون 45 از 40 بزرگتر است به سمت راست ریشه رفته و چون از 50 کوچکتر است در سمت چپ گره 50 درج شود.

 درج (یا حذف) در BST با n گره و به ارتفاع h در زمان ($O(h)$) انجام می‌گیرد.

 برای حذف عناصر تکراری یک لیست، یک درخت جستجوی دودویی با آن عناصر می‌سازیم. به عبارتی BST بهترین ساختمان داده‌ها برای حذف داده‌های تکراری می‌باشد.

مثال

فرض کنید n عنصر A_1, A_2, \dots, A_n از قبل مرتب شده‌اند ($A_1 < A_2 < \dots < A_n$). این عناصر را به ترتیب در درخت BST خالی اضافه می‌کنیم. ارتفاع درخت حاصل کدام است؟

پاسخ: در صورت ساختن یک BST با n عدد که به صورت صعودی مرتب می‌باشند، درخت اریب به راست با ارتفاع n حاصل می‌شود.



تابع درج در BST

```
insert( t , k ){
    p = search(t , k);
    if ( p or !t ){
        n = malloc(sizeof(t));
        n -> data = k;
        n -> left = n -> right = NULL;
        if (*t)
            if ( k < p -> data) p -> left = n; else p -> right = n;
        else *t = n;
    }
}
```

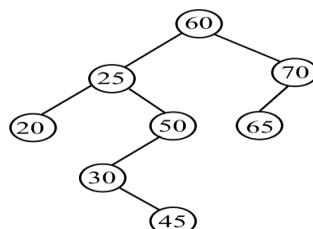
تابع `search`، گره درخت دودویی را برای کلید k جستجو می‌کند. اگر درخت تهی یا k والد باشد، مقدار `NULL` را برگشت می‌دهد. در غیر اینصورت، اشاره گری به گره آخر درخت که در طی جستجو مورد استفاده قرار گرفته، برگشت می‌دهد. عنصر جدید باید به عنوان این گره درج شود.

مرتب سازی به کمک BST

با ایجاد یک درخت جستجوی دودویی با n عدد و سپس پیمایش میانوندی (inorder) درخت، می‌توان یک لیست مرتب شده از اعداد را بدست آورد. به این روش مرتب سازی، Tree Sort می‌گویند.

مثال

پیمایش میانوندی BST زیر را بدست آورید؟



پاسخ: در صورت پیمایش LVR درخت داده شده، لیست مرتب شده زیر بدست می‌آید:
 $20, 25, 30, 45, 50, 60, 65, 70$

■ مرتبه اجرایی الگوریتم مرتب سازی درختی در بهترین حالت برابر $O(n \lg n)$ و در بدترین حالت (BST اریب)، برابر $O(n^2)$ می‌باشد.

■ با فرض اینکه a_1, a_2, \dots, a_n دنباله BST inorder یک باشد:

- اگر a_i برگ نباشد، یا a_{i-1} در زیر درخت چپ a_i است یا a_{i+1} در زیر درخت راست آن (یا هردو)
- اگر a_i دو فرزند داشته باشد، a_{i+1} فرزند چپ ندارد. ($i < n$)
- اگر a_i دو فرزند داشته باشد، a_{i-1} فرزند راست ندارد. ($i > 1$)

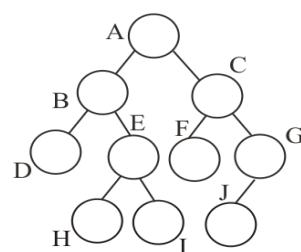
مثال

اگر عناصر یک BST را به صورت inorder پیمایش کنیم و در داخل یک استک قرار دهیم و سپس عناصر استک را خارج کنیم و یک BST بسازیم، درخت حاصل چه درختی خواهد بود؟ پاسخ:

پیمایش میانوندی BST اعداد را به صورت صعودی مرتب می‌کند و در صورتی که آنها را در پشته ریخته و سپس آنها را pop کنیم، حاصل نزولی خواهد شد. حال در صورتی که با داده‌های نزولی یک BST بسازیم، حاصل اریب به چپ خواهد بود.

مثال

اگر T یک درخت جستجو باینری به صورت زیر باشد که در هر گره آن یک عدد صحیح ذخیره شده است، چهارمین کوچکترین عنصر آن در کدام گره قرار دارد؟



پاسخ:

نتیجه پیمایش میانوندی BST داده شده DBHEIAFCJG می‌باشد. به کمک این پیمایش که همواره صعودی است، مشخص می‌شود که گره E حاوی چهارمین کوچکترین عنصر است.

مثال ۸-۱۵ (مهندسی IT - دولتی ۸۹)

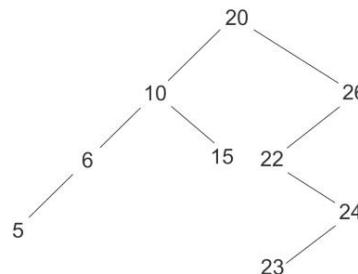
هفتمین کلید در پیمایش preorder یک درخت جستجوی دودویی(BST) که پیمایش postorder آن به شکل زیر است(از چپ به راست)، کدام گزینه است؟

Postorder : 5 , 6 , 15 , 10 , 23 , 24 , 22 , 26 , 20

24 (۴) 22 (۳) 15 (۲) 26 (۱)

پاسخ: گزینه ۳ جواب است.

می دانیم که پیمایش میانوندی BST ، کلیدها را به صورت صعودی مرتب می کند. بنابراین پیمایش میانوندی این درخت برابر است با: 26 , 24 , 22 , 23 , 15 , 10 , 6 , 20 . همچنین می دانیم که با داشتن دو پیمایش از یک درخت دودویی که یکی از آنها میانوندی است، می توان درخت را رسم کرد. این درخت به صورت زیر می باشد:



پیمایش preorder این درخت برابر است با: 20 , 10 , 6 , 5 , 15 , 26 , 22 , 24 , 23

مشاهده می کنید که هفتمین کلید در این پیمایش برابر 22 می باشد.

روش دوم:

در BST ، می توان به کمک پیمایش postorder درخت را رسم کرد. با توجه به پیمایش postorder داده شده یعنی 5,6,15,10,23,24,22,26,20

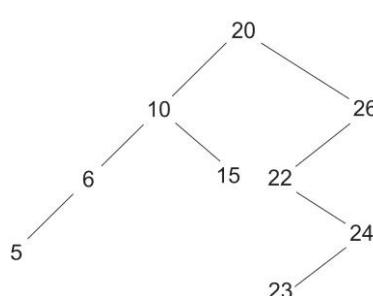
الف: اعداد کوچکتر از 20 یعنی 5,6,15,10 که در سمت چپ 20 قرار می گیرند.

ب: اعداد بزرگتر از 20 یعنی 23,24,22,26 که در سمت راست 20 قرار می گیرند.

این روال را برای این دو نیمه تکرار می کنیم. در بین اعداد سمت چپ، آخرین گره یعنی 10 در ریشه قرار گرفته و اعداد 6

در چپ آن و عدد 15 در راست آن قرار می گیرد. در سمت راست ریشه نیز عدد 26 در ریشه و اعداد 23,24,22 در چپ آن

قرار می گیرند. به همین روال ادامه داده تا درخت زیر حاصل شود:



پیمایش preorder این درخت برابر است با:

20 , 10 , 6 , 5 , 15 , 26 , 22 , 24 , 23



 اگر عناصر دنباله پیمایش سطح ترتیب حاصل از یک BST را در یک درخت خالی درج کنیم، درخت اولیه ایجاد می‌شود.

 تعداد BST های متفاوت با n گره و برچسب های ۱ تا n که دارای ترتیب یکسانی در هر دو روش postorder و inorder هستند، برابر عدد $n!$ است کاتالان می‌باشد.

 در یک BST، با داشتن دنباله پیمایش های preorder یا level order یا postorder یا inorder، می‌توان آن را به طور یکتا ساخت. (تذکر: با داشتن پیمایش inorder یک BST نمی‌توان درخت را به صوت یکتا ساخت، چون BST های متفاوتی می‌توان مثلای داشت که پیمایش inorder آنها با هم برابر باشد.)

پیدا کردن عنصر کمینه در BST

عنصری که کلید آن در BST، مینیمم است را می‌توان با دنبال کردن اشاره گرهای فرزندان چپ از ریشه تا رسیدن به NULL پیدا کرد. روال زیر اشاره گری به عنصر مینیمم در زیر درخت مشتق شده از t را بر می‌گرداند:

```

min( t ){
    while ( t -> left !=NULL )
        t = t -> left;
    return t;
}
max( t ){
    while ( t -> right !=NULL )
        t = t -> right;
    return t;
}

```

 مرتبه اجرایی روال های min یا max روی یک درخت BST با ارتفاع h برابر $O(h)$ می‌باشد.

حذف یک گره از BST

پس از یافتن گره مورد نظر برای حذف، یکی از سه حالت زیر ممکن است رخ دهد:

۱- اگر گره فاقد فرزند باشد، به سادگی حذف شده و نیازی به تنظیم درخت نمی‌باشد.

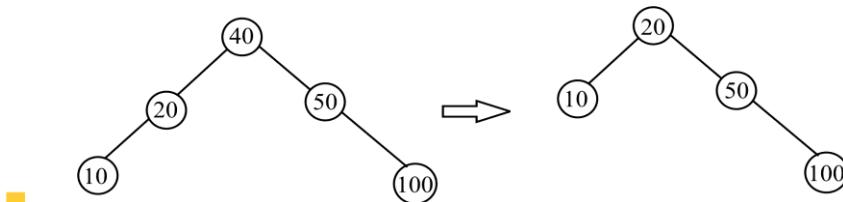
۲- اگر گره فقط دارای یک فرزند باشد، فرزند آن به طرف بالا در درخت منتقل می‌شود.

۳- اگر گره دارای دو فرزند باشد، گره بعدی یا قبلی آن در پیمایش میانوندی جای آن را می‌گیرد. به عبارتی عنصر حذف شده، با مقدار بزرگترین عنصر زیر درخت چپ یا کوچکترین عنصر زیر درخت راست جایگزین می‌شود.

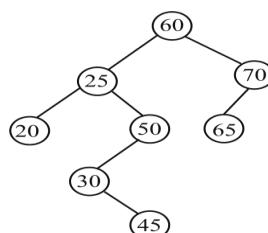
 در BST غیر اریب، می‌توان کوچکترین عنصر را با مرتبه $O(\log n)$ حذف کرد.

مثال

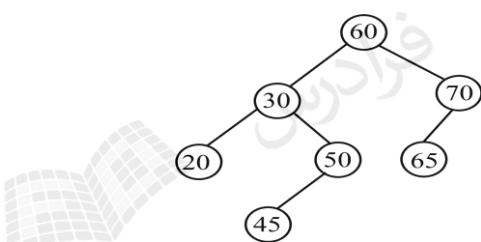
برای حذف گره با داده 40 از درخت جستجوی زیر، گره 20 یا 50 می‌توانند جایگزین شوند، که ما گره 20 را جایگزین کرده‌ایم:

**مثال**

گره با داده 25 را از درخت زیر حذف کنید.



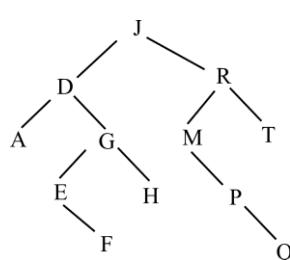
پاسخ: برای حذف گره با داده 25، می‌توان گره 20 یا 30 را جایگزین کرد. (در این مثال گره 30)



تذکر: در واقع گره 20 یا 30، گره‌های قبل و بعد از گره 25 در پیمایش میانوندی درخت می‌باشند.

مثال

بعد از حذف گره D در درخت BST زیر، کدام گره جایگزین آن می‌شود؟



پاسخ: گره E یا گره F

بررسی BST بودن

تابع زیر بررسی می‌کند که آیا درخت دودویی t با کلیدهای صحیح و متمایز یک BST است:
 $(\min = -32768, \max = 32767)$

```
f(tree *t , int min , int max){
    if (t == NULL)
        return TRUE;
    if ( t->key < min || t->key > max )
        return FALSE;
    return f(t->left , min, t->key-1) && f(t->right , t->key+1, max)
}
```

چاپ داده‌های در یک محدوده خاص

تابع زیر، کلیه کلیدهای واقع در محدوده خاص $a \leq k \leq b$ در یک BST با ریشه t را چاپ می‌کند:

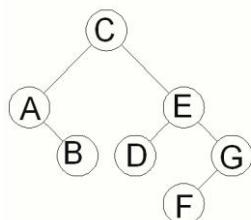
```
f(t , a , b){
    if ( t->data >= a && t->data <= b)
        cout << t->data;
    if ( t->data >= a && t->left != null )
        f( t->left , a , b);
    if ( t->data <= b && t->right != null)
        f( t->right , a , b);
}
```

اگر h ارتفاع درخت و m تعداد جواب باشد، زمان اجرای این الگوریتم $O(m + h)$ است. چون حداکثر تعداد دفعاتی که این تابع صدای زده می‌شود و چیزی چاپ نمی‌شود از $O(h)$ است.

روش دوم: می‌توان دو عنصر a و b را در درخت پیدا کرد. سپس با شروع از a و تاریخیدن به b (مرحله)، عمل Tree-Successor (a) را انجام دهیم و آن را به مجموعه جواب نیز اضافه کنیم.

مثال

چه تعداد از حالت‌های درج عناصر A تا F در یک BST تهی، درخت مشابه زیر را تولید می‌کنند؟



پاسخ: دنباله دارای هفت خانه است که باید با C شروع شود. سپس دو خانه از شش خانه باقی مانده به $\binom{6}{2}$ حالت برای A و

B در نظر گرفته می‌شود. پس از آن، عناصر باقی مانده به یکی از سه ترتیب زیر در چهار خانه باقی مانده، قرار می‌گیرند.
 $\{E,D,G,F\}$ ، $\{E,G,D,F\}$ ، $\{E,G,F,D\}$
در نتیجه تعداد کل حالت‌ها برابر است با:

$$\binom{6}{2} \times 3 = 15 \times 3 = 45$$



مثال

تعداد درخت‌های دودویی جست و جویی که می‌توان با 10 کلید داده شده مجزا از هم ساخت به طوری که اختلاف عمق برگ‌های آن درخت حداقل 1 باشد، چند تاست؟
پاسخ:

چون اختلاف عمق برگ‌ها باید حداقل یک باشد، درخت تا سطح سوم پر است و 7 گره دارد و 3 گره دیگر آن باید در سطح چهارم قرار گیرد. در سطح چهارم 8 مکان وجود دارد و به $\binom{8}{3}$ یعنی 56 روش می‌توان این 3 گره را در این محل‌ها قرار داد.



تعداد BST که می‌توان با n کلید داده شده مجزا از هم ساخت به طوری که اختلاف عمق برگ‌های درخت حداقل 1 باشد، برابر $\binom{k}{n+1-k}$ می‌باشد به طوری که $k = 2^{\lfloor \log n \rfloor}$.

در واقع k بزرگترین عدد توان دو است که از n کوچکتر است.

با فرض اینکه $P(T)$ ، مجموع طول مسیرهای BST به نام T باشد، آنگاه:

$$P(T) = P(T_R) + P(T_L) + n - 1$$

همچنین می‌توان نشان داد که میانگین عمق یک گره در T برابر $\frac{1}{n} P(T)$ می‌باشد.

با فرض اینکه $P(n)$ بیانگر مجموع طول مسیرهای تمام BST‌های ساخته شده با n گره باشد، خواهیم داشت:

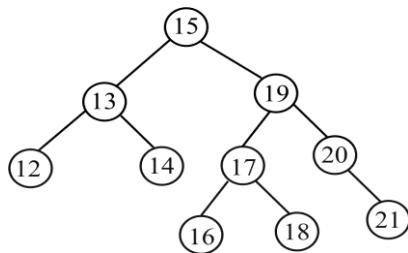
$$P(n) = \frac{1}{n} \sum_{i=0}^{n-1} (P(i) + P(n-i-1) + n - 1)$$

می‌توان $P(n)$ را به صورت $P(n) = \frac{2}{n} \sum_{k=1}^{n-1} P(k) + \theta(n)$ نیز نوشت.

spin چپ یک BST ، مسیری از ریشه به گره ای با کوچکترین کلید می‌باشد. بنابراین تنها شامل یالهای چپ می‌باشد.

درخت AVL

درخت AVL یک درخت جستجوی دودویی است که حداقل اختلاف ارتفاع دو زیر درخت چپ و راست آن برابر یک بوده ($|h_R - h_L| \leq 1$) و این خاصیت در هر یک از زیر درختان نیز برقرار است. درخت زیر یک AVL می‌باشد:



تذکر: در درخت AVL ریشه را در سطح صفر فرض می‌کنیم. بنابراین ارتفاع درخت بالا ۳ است.

حداقل تعداد گره‌های مورد نیاز برای ساختن AVL

اگر تعداد عناصر یک درخت AVL کمینه باشد، تعداد عناصر زیر درخت‌های آن هم باید کمینه باشد. بنابراین حداقل تعداد گره مورد نیاز برای ساختن یک درخت AVL با ارتفاع h برابر است با:

$$G_h = G_{h-1} + G_{h-2} + 1$$

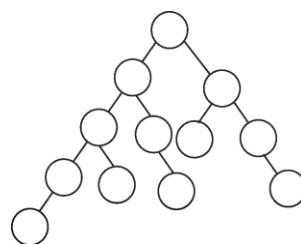
$$G_0 = 1, G_1 = 2$$

به طور نمونه برای ساختن یک AVL با ارتفاع دو، حداقل نیاز به چهار گره می‌باشد:

$$G_2 = G_1 + G_0 + 1 = 1 + 2 + 1 = 4$$

مثال

شکل زیر یک درخت AVL با ارتفاع چهار، با حداقل تعداد گره‌ها یعنی 12 را نشان می‌دهد:



حداکثر ارتفاع یک درخت AVL با n گره از $O(\log n)$ است.

مرتبه اجرایی هر یک از اعمال "جستجو، حذف، درج" در AVL، برابر $O(\log n)$ می‌باشد.

در سری فیبوناچی داریم $F_n = F_{n-1} + F_{n-2}$ ، پس با توجه به جدول زیر مشخص می‌شود که:

$$G_n = F_{n+3} - 1$$

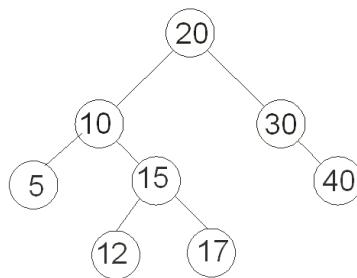
n	0	1	2	3	4	5	6	7	8
F	0	1	1	2	3	5	8	13	21	
G	1	2	4	7	12	20	33	54	88	

مثال

با کلیدهای 3 و 1 چه تعداد درخت AVL می‌توان ساخت؟
پاسخ: فقط یک درخت که 2 در ریشه و 1 در چپ آن و 3 در راست آن باشد.

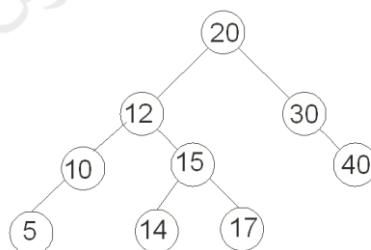
مثال

عنصر با کلید 14 را به درخت AVL زیر اضافه کنید.



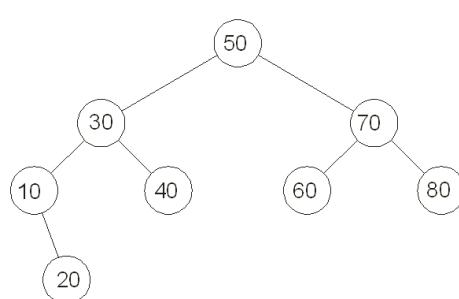
پاسخ:

با توجه به اینکه 14 از 20 کوچکتر و از 10 بزرگتر و از 15 کوچکتر است، در سمت راست 12 اضافه می‌شود. از آنجا که در AVL حداقل اختلاف ارتفاع زیر شاخه‌های چپ و راست برای هر گره باید یک باشد، بعد از اضافه شدن 14، ارتفاع زیر شاخه چپ کلید 10 برابر 1 و ارتفاع زیر شاخه راست برابر 3 می‌شود که توازن آن بهم می‌خورد. بنابراین با توجه به قوانین چرخش، کلید 12 به جای 10 می‌رود و 10 در چپ 12 قرار می‌گیرد. سپس 14 به چپ 15 می‌رود.
درخت نهایی به صورت زیر است:



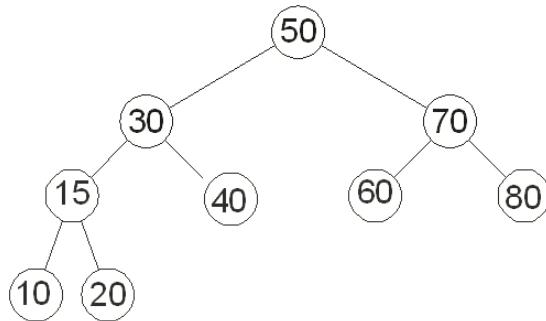
مثال

عنصر با کلید 15 را به درخت AVL زیر اضافه کنید.



پاسخ:

در درخت AVL باید حداقل اختلاف ارتفاع چپ و راست برابر یک باشد. عدد 15 به سمت چپ 20 اضافه می‌شود و چون اختلاف عمق گره 10 بیش از یک می‌شود، 15 به جای 10 رفته و 10 در چپ 15 و 20 در راست 15 قرار می‌گیرد. شکل نهایی به صورت زیر است:

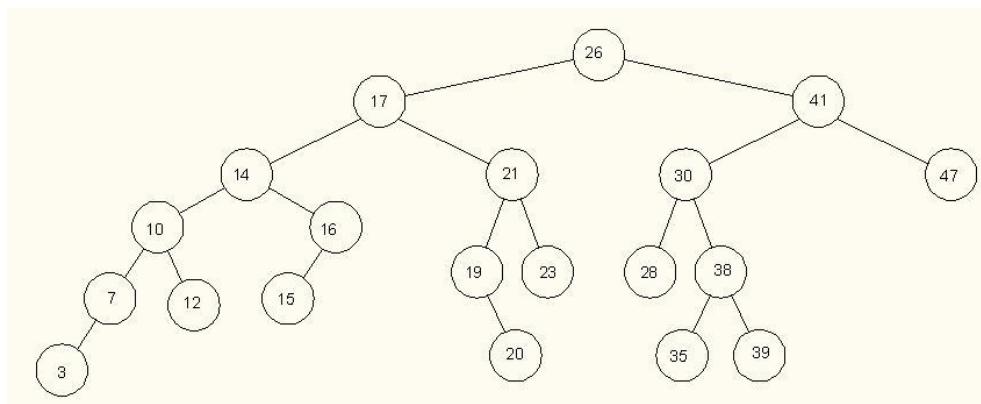


درخت قرمز-سیاه

درخت قرمز-سیاه یک BST است که با اضافه کردن یک بیت به هر گره (معرف رنگ قرمز یا سیاه)، کاری می‌کنیم که همیشه ارتفاع درخت حداقل $\log n$ شود. در این درخت، هر یک از اعمال درج، حذف، جستجو و اعمال دیگری که در BST قابل انجام است، می‌توان در زمان لگاریتمی انجام داد. در یک درخت قرمز-سیاه:

- ۱- رنگ ریشه سیاه است.(این شرط ضروری نیست)
- ۲- رنگ برگ‌ها سیاه است.
- ۳- پدر یک گره قرمز، حتما سیاه است.
- ۴- به ازای هر گره داخلی x ، تعداد گره‌های سیاه(بجز خود x) موجود که از x تا هر یک از نواده برگ تهی آن برود، برابر است. (به این تعداد، ارتفاع سیاه x می‌گوییم و با $bh(x)$ نشان می‌دهیم.)

شکل زیر یک درخت قرمز-سیاه را نشان می‌دهد:



حداقل ارتفاع یک درخت قرمز-سیاه که دارای n گره داخلی باشد، برابر $2\log(n+1)$ است.



 ارتفاع یک گره درخت قرمز-سیاه با n گره داخلی $O(\log n)$ است.

 با فرض اینکه $p[x]$ معرف پدر گره x باشد، $\text{uncle}[x]$ یعنی عموی گره x ، به صورت زیر قابل بیان است:

```
if p[x] = right[p[p[x]]] then
    uncle[x] ← left[p[p[x]]]
else
    uncle[x] ← right[p[p[x]]]
```

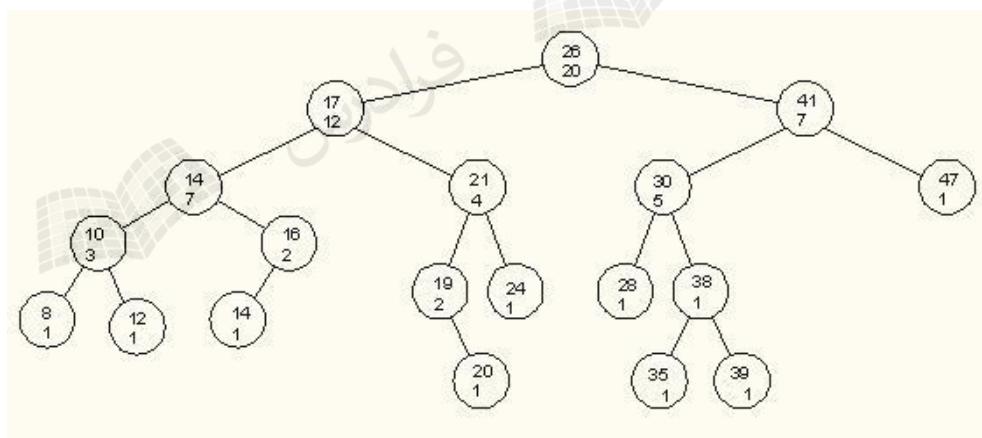
درخت مرتبه آماری

درخت مرتبه آماری، یک درخت قرمز-سیاه است که مجموعه ای از n عنصر را که به صورت پویا در آن درج یا از آن حذف می شوند، طوری پیاده سازی می کند که اعمال مرتبه آماری را در هر زمان بتوان در $O(\log n)$ انجام داد. (منظور از اعمال مرتبه آماری، یافتن مرتبه یک عنصر یا یافتن عنصری با مرتبه خاص می باشد).

هر گره در درخت مرتبه آماری، علاوه بر اطلاعات مربوط به خاصیت قرمز-سیاه، یک مولفه $\text{size}[x]$ هم دارد که تعداد عناصر موجود در زیر درختی به ریشه x را نشان می دهد. یعنی:

$$\text{size}[x] = \text{size}[\text{left}[x]] + \text{size}[\text{right}[x]] + 1$$

شکل زیر یک درخت مرتبه آماری را نشان می دهد:



دو رویه استفاده شده بر روی درخت مرتبه آماری

۱- رویه select(x,i) ، عنصری با i امین کلید بین عناصر زیر درختی به ریشه x را به دست می آورد.

۲- رویه rank(T,x) ، مرتبه عنصر x را در درخت مرتبه آماری T به دست می آورد.

<pre>select (x , i) { r = size[left[x]]+1; if (r == i) return x; else if (r > i) return select (left[x] , i); else return select(right[x] , i-r); }</pre>	<pre>rank(T,x) { r = size[left[x]]+1; y = x; while (y != root[x]) if (y == right[p[y]]) r = r + size[left[p[y]]] + 1; y = p[y]; return r; }</pre>
--	---

تحقیق: یکی دیگر از گسترش های درخت قرمز- سیاه، درخت بازه است. درباره این درخت تحقیق کنید.

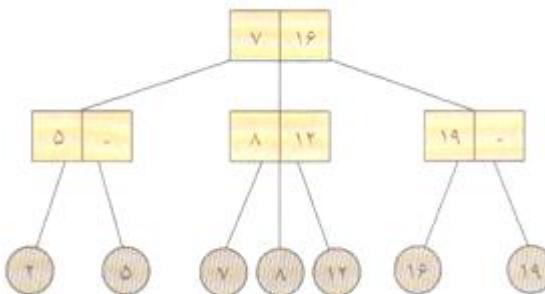
درخت ۲-۳

یک درخت ۲-۳، درختی است که هر گره داخلی یا ۲ یا ۳ فرزند دارد و همه برگها در عمق یکسان هستند. (درختی کاملاً متوازن). عناصر در این درخت، به ترتیب کلیدشان از چپ به راست فقط در برگها قرار دارند و فرض می شود که کلید هر عنصر تک است. یک درخت ۲-۳ با یک عنصر، تنها یک برگ است.

در درخت ۲-۳، در هر گره داخلی x ، دو مقدار ذخیره می شود:

۱- کلید کوچکترین عنصر در زیر درخت دوم x

۲- کلید کوچکترین عنصر در زیر درخت سوم x (البته در صورت وجود)



یک درخت ۲-۳ به ارتفاع h ، حداقل 2^h و حداقل 3^h برگ دارد و همین تعداد عنصر را می تواند در خود ذخیره کند. به عبارتی اگر h ارتفاع یک درخت ۲-۳ با n عنصر باشد، داریم:

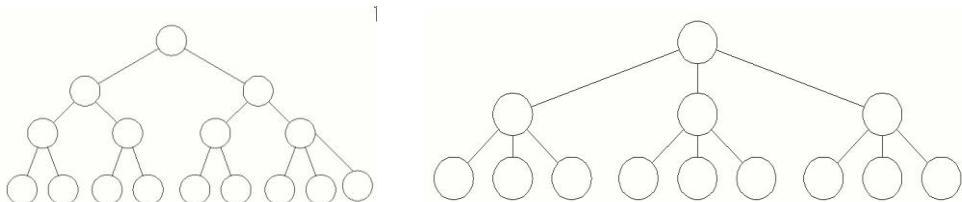
$$\left\lfloor \log_3^n \right\rfloor \leq h \leq \left\lfloor \log_2^n \right\rfloor$$

هر یک از اعمال "درج، حذف و جستجو" در درخت ۲-۳ از $O(\log n)$ می باشند.

مثال

یک درخت ۲-۳ با ۹ برگ، چه تعداد گره داخلی دارد؟

حل: این درخت دو حالت می‌تواند داشته باشد. با ۴ گره داخلی و با ۷ گره داخلی:



درخت بی (B-Tree)

B-Tree ها، درخت های جستجوی متوازنی هستند که برای کار روی رسانه ها با دسترسی مستقیم طراحی شده اند. این درخت به منظور بازیابی، درج و حذف رکوردها از یک فایل بزرگ طراحی شده است. ارتفاع این درخت برابر $O(\log n)$ می‌باشد. درخت بی عمدتاً برای ذخیره داده ها بر روی حافظه خارجی مورد استفاده قرار می‌گیرد. B-tree ها شبیه درختهای قرمز-سیاه می‌باشند، اما در مینیمم کردن اعمال I/O دیسک بهتر عمل می‌کنند. تفاوت B-tree ها با درخت های قرمز-سیاه در این است که گره های B-tree ممکن است فرزندان زیادی داشته باشند. درخت بی را می‌توان مبتنی بر دو درخت زیر طراحی کرد:

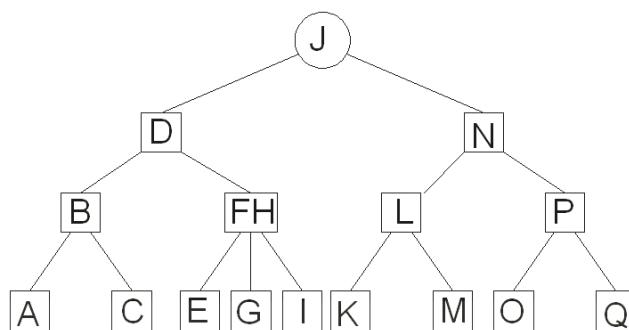
۱ - درخت دودویی جستجو ۲ - درخت ۲-۳

یک B-TREE با درجه t یک درخت جستجو با ویژگی های زیر است:

- ۱- برگها همه در یک سطح قرار دارند و رکوردهای ذخیره شده در هر برگ به ترتیب کلیدشان از چپ به راست قرار دارند.
 - ۲- هر گره به غیر از ریشه باید حداقل $t-1$ و حداکثر $2t-1$ کلید داشته باشد.
 - ۳- هر گره داخلی به غیر از ریشه حداقل t و حداکثر $2t$ فرزند دارد.
 - ۴- در درخت غیر تهی، ریشه حداقل یک کلید دارد.
 - ۵- گره پر دقیقاً شامل $t-1$ کلید است.
 - ۶- هر گره داخلی شامل $k+1$ اشاره گر به فرزندانش است. (k تعداد کلیدهای گره)
- تذکر: کمترین مقدار برای t برابر ۲ می‌باشد.

مثال

درخت زیر یک B-Tree با درجه مینیمم $t=2$ است. هر گره به غیر از ریشه حداقل ۱ و حدکثر ۳ کلید دارد.
هر گره داخلی، ۲، ۳ و یا ۴ فرزند می‌تواند داشته باشد.



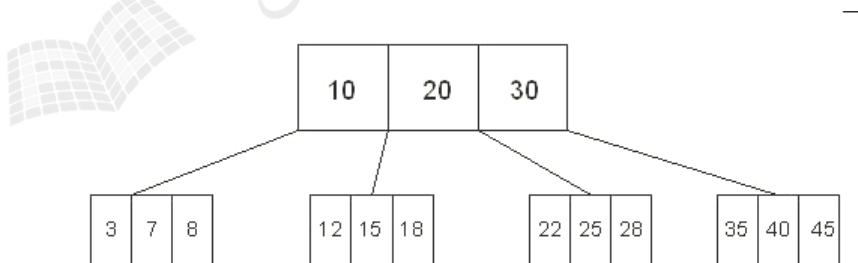
حداقل تعداد کلیدها در B-Tree با درجه t و ارتفاع h برابر است با:

$$[1 + (2t) + (2t)^2 + \dots + (2t)^h] \times (2t - 1) = \frac{(2t)^{h+1} - 1}{2t - 1} \times (2t - 1) = (2t)^{h+1} - 1$$

تعداد کلیدها در B-Tree با درجه 2 و ارتفاع h حداقل $4^{h+1} - 1$ می‌باشد.

مثال

شکل زیر یک B-Tree با $t=2$ و $h=1$ را نشان می‌دهد که دارای حداقل تعداد کلید یعنی 15 است.



حداقل تعداد کلیدها در B-Tree با درجه t و ارتفاع h برابر است با:

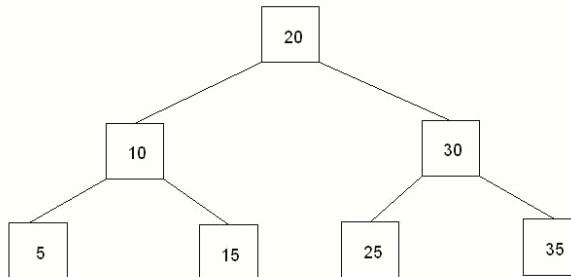
$$1 + (2 + 2t + 2t^2 + \dots + 2t^{h-1})(t - 1) = 1 + 2\left(\frac{t^h - 1}{t - 1}\right)(t - 1) = 2t^h - 1$$

ریشه حداقل یک کلید دارد و بقیه گره‌ها حداقل $t-1$ کلید دارند. بنابراین حداقل 2 گره در عمق 1 ، $2t$ گره در عمق 2 ، $2t^2$ گره در عمق 3 و ... و $2t^{h-1}$ گره در عمق h وجود دارند.

تعداد کلیدها در B-Tree با درجه 2 و ارتفاع h حداقل $2^{h+1} - 1$ می‌باشد.

مثال

شکل زیر یک B-Tree با $t=2$ و $h=2$ را نشان می‌دهد که دارای حداقل تعداد کلید یعنی 7 است.



$$h \leq \log_t^{(n+1)/2} \quad \text{در یک B-Tree با } n \text{ کلید و ارتفاع } h \text{ و درجه } t \text{ داریم:}$$



پیاده سازی درخت بی مبتنی بر درخت ۲-۳

یک درخت بی از مرتبه m یک درخت جستجوی m تایی با ویژگی های زیر است:

۱ - ریشه، یا برگ است یا حداقل دو فرزند دارد.

۲ - هر گره، غیر از ریشه و برگ ها، حداقل $\left\lceil \frac{m}{2} \right\rceil$ و حداکثر m فرزند دارد.

۳ - یک گره داخلی دارای $m-1$ کلید و m اشاره گر است.

۴ - برگ ها همه در یک سطح قرار دارند و رکوردهای ذخیره شده در هر برگ به ترتیب کلیدشان از چپ به راست قرار دارند.

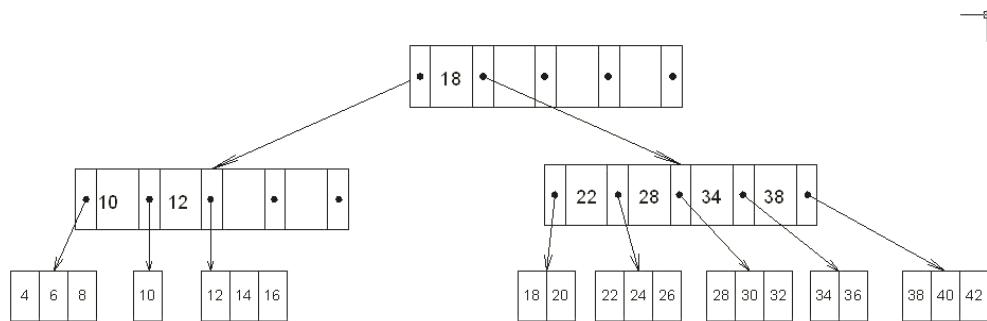
۵ - در هر برگ با توجه به اندازه هر رکورد، از 1 تا حداکثر k رکورد قرار می گیرد.

تذکر: معمولاً مقادیر m و k به گونه ای تعیین می شوند که اندازه هر گره داخلی و هر برگ برابر یک بلوک از دیسک باشد و بتوان با یکبار دسترسی آن را به حافظه اصلی خواند.

تذکر: اشاره گرها آدرس بلوک بر روی دیسک هستند.

مثال

شکل زیر یک درخت بی از مرتبه پنج را نشان می‌دهد که در هر بلوک برگ، حداکثر سه رکورد جای می‌گیرد.



درخت ۳-۲ یک درخت بی از مرتبه ۳ است.

اگر j تعداد گره‌های موجود در مسیر بین ریشه و برگ‌ها باشد، اعمال جستجو و حذف حداکثر j بار و عمل درج حداکثر $1+j$ بار نیاز به دسترسی به دیسک دارد.

کنکور ارشد

(مهندسی IT - آزاد ۸۶)

۱- می خواهیم با استفاده از اعداد ۱۴, ۱۲, ۲۰, ۱۵ به هر ترتیب دلخواه در یک درخت نهی دودویی جستجو، یک درخت دودویی جستجو با ۴ گره بسازیم. در این صورت تعداد درخت‌های دودویی جستجوی ممکن برابر است با:

8 (۴)

14 (۳)

12 (۲)

16 (۱)

پاسخ: گزینه ۳ جواب است.

می توان با قرار دادن مقدار ۴ به جای n در رابطه

$$\frac{1}{n+1} \binom{2n}{n}$$

به جواب رسید.

$$\frac{1}{5} \times \binom{8}{4} = \frac{1}{5} \times \frac{8!}{4! \times 4!} = \frac{1}{5} \times \frac{8 \times 7 \times 6 \times 5 \times 4!}{4! \times 4!} = \frac{1}{5} \times \frac{8 \times 7 \times 6 \times 5}{4!} = 14$$



(علوم کامپیوتر - دولتی ۹۰)

۲- فرض کنید n عدد در بازه $[1, \log n]$ داده شده باشند. ارتفاع درخت جستجوی دودویی (h)، برای این اعداد چه وضعیتی دارد؟ (از هر عدد حداقل یک نمونه وجود دارد).

$$\log^2 n \leq h \leq n \quad (۲)$$

$$\log n \leq h \leq n \quad (۱)$$

$$\log(\log n) \leq h \leq \log n \quad (۴)$$

$$\log n \leq h \leq \log^2 n \quad (۳)$$

پاسخ: جواب گزینه ۴ است.

ارتفاع BST برای یک درخت اریب برابر با تعداد اعداد و برای یک درخت متوزان برابر لگاریتم تعداد اعداد است. بنابراین اگر $\log n$ عدد داشته باشیم، داریم:

$$\log(\log n) \leq h \leq \log n$$

(علوم کامپیوتر - دولتی ۸۴)

۳- در یک درخت جستجوی باینری با n گره، تعداد مقایسه برای جستجوی یک عنصر حداکثر برابر است با :

$$O(\log n) \quad (۴)$$

$$O(\sqrt{n}) \quad (۳)$$

$$O(n) \quad (۲)$$

$$O\left(\frac{n}{2}\right) \quad (۱)$$

پاسخ: جواب گزینه ۲ است.

یک درخت BST اگر اریب باشد، در بدترین حالت عنصر مورد جستجو در برگ است و برای پیدا کردن آن به n مقایسه نیاز می باشد.



(مهندسی IT - دولتی ۸۹)

۴- فرض کنید ۱۰۰ کلید ۱ تا ۱۰۰ را به یک درخت جستجوی دودویی اضافه کرده ایم. ترتیب اضافه شدن کلیدها مشخص نیست، اما احتمال تمام حالات ترتیب‌های اضافه شدن کلیدها (تمام Permutation های ممکن یک تا صد) با هم برابر است. با چه احتمالی در روند اضافه شدن کلیدها به درخت، کلید ۴ و کلید ۹ با هم مقایسه می شوند، در صورتی که اولین کلید اضافه شده کلید ۴۳ باشد؟

$$\frac{1}{3} \quad (4)$$

$$\frac{1}{4} \quad (3)$$

$$\frac{43}{100} \times \frac{1}{4} \quad (2)$$

$$\frac{1}{2} \quad (1)$$

پاسخ: جواب گزینه ۴ است.

چون طبق صورت سؤال، ابتدا 43 وارد شده است، بنابراین 43 در ریشه و اعداد 4 و 9 هر دو در سمت چپ درخت قرار می‌گیرند. حال اگر دو عدد 4 و 9 زودتر از چهار عدد بین آنها (یعنی 5,6,7,8) به درخت اضافه شوند، حتماً با یکدیگر مقایسه می‌شوند. احتمال این حالت برابر $\frac{2}{6}$ می‌باشد. عدد 2 در صورت به علت تعداد اعداد 4 و 9 و عدد 6 در مخرج به علت کل اعداد از 4 تا 9 می‌باشد).

تذکر: وقتی 4 و 9 با یکدیگر مقایسه نمی‌شوند، که یکی از اعداد بین آنها، قبل از آنها به درخت اضافه شود.

(مهندسی IT - دولتی) (۸۸)

۵- تابع زیر بررسی می‌کند که آیا یک درخت دودویی با کلیدهای صحیح (int) و متمایز یک درخت جستجوی دودویی است یا خیر؟ محل های خالی A تا D گزینه است؟

```
bool ISBST (tree *t){ return ISBST (t, MININT, MAXINT); }
int ISBST (tree *t, int min , int max){
    if (t==null)
        return TRUE;
    if (t->key < min || t->key> max)
        return FALSE;
    return ISBST (t->leftchild, A, B) && ISBST (t->rightchild , C,D)
}
```

$$A = \min, B = t \rightarrow \text{key}-1, C = t \rightarrow \text{key}+1, D = \max \quad (1)$$

$$A = \min, B = t \rightarrow \text{key}+1, C = t \rightarrow \text{key}-1, D = \max \quad (2)$$

$$A = t \rightarrow \text{key}+1, B = \min, C = \max, D = t \rightarrow \text{key}-1 \quad (3)$$

$$A = t \rightarrow \text{key}-1, B = \min, C = \max, D = t \rightarrow \text{key}+1 \quad (4)$$

پاسخ: جواب گزینه ۱ است.

در BST ، کلید هر گره از همه کلیدهای گره های زیر درخت چپ بزرگتر و از همه کلیدهای زیر درخت راست کوچکتر است. الگوریتم ISBST ، چک می‌کند که آیا درخت با آدرس ریشه t یک BST است یا خیر. برای این کار تابع ISBST دیگری با سه ورودی را صدا می‌کند که کمترین و بیشترین مقدار ممکن برای یک متغیر از نوع int را علاوه بر t با تابع ارسال می‌کند. (min=-32768, max=32767)

این تابع به صورت بازگشتی سمت چپ و سمت راست را بررسی می‌کند:

ISBST(t->left , min , t->key-1) && ISBST(t->right , t->key+1 , max)

قسمت اول، این موضوع را بررسی می‌کند که در سمت چپ گره t ، مقدار کلیدها حداقل برابر min و حداکثر یک واحد کمتر از مقدار کلید گره t باشند.

قسمت دوم، این موضوع را بررسی می‌کند که در سمت راست گره t ، مقدار کلیدها حداقل یک واحد بیشتر از مقدار کلید گره t و حداکثر برابر max باشند.

(مهندسی IT - دولتی ۸۸)

۶- حداقل تعداد کلید در یک Tree.B با مینیمم درجه i نود t و ارتفاع h چقدر است؟

$$2t^{h+1} - 1 \quad (4) \quad 2t^h - 1 \quad (3) \quad (2t)^{h+1} - 1 \quad (2) \quad (2t)^{h+1} \quad (1)$$

حل: گزینه ۲ درست است.

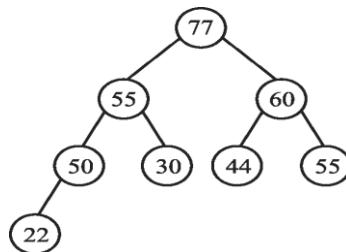
فصل ۸: درخت های Heap

در این فصل درختهای heap، treap، Heap، Deap، Fibonaci را بررسی می‌کنیم.

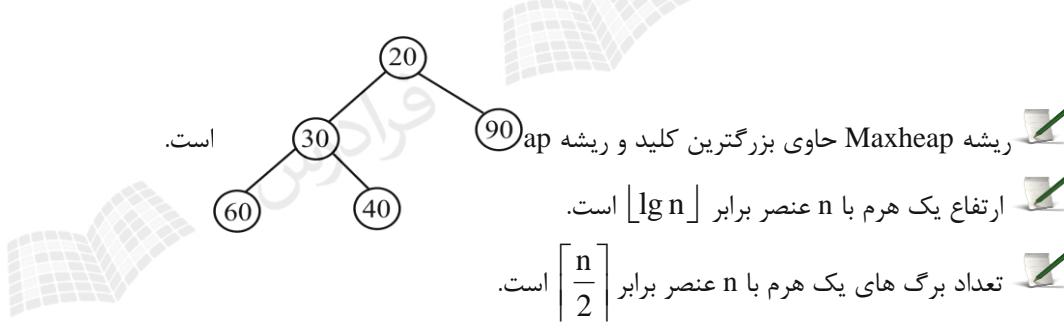
هرم ها (HEAPS)

دو نوع Heap دودویی وجود دارد:

-۱ - **Max-Heap** : یک درخت دودویی کامل که مقدار کلید هر گره آن بزرگتر یا مساوی مقدار کلیدهای فرزندانش باشد.



-۲ - **Min-Heap** : یک درخت دودویی کامل که مقدار کلید هر گره آن کوچکتر یا مساوی مقدار کلیدهای فرزندانش باشد.



در یک هرم با n گره، عمق یک گره در درایه i ، برابر $\lfloor \lg i \rfloor$ است.

بیشترین تعداد گره های با ارتفاع h در یک هرم با n عنصر برابر است با:

کوچکترین عنصر در یک MaxHeap با عناصر متمایز، برگ است. (زیرا در غیر اینصورت دارای فرزند است و باید از آنها بزرگتر باشد، که با "کوچکترین" بودن در تنافض است).

کوچکترین عنصر در یک MaxHeap با عناصر متمایز را می توان با $O(n)$ پیدا کرد.

(چون این عنصر در برگ قرار دارد و تعداد برگها نیز برابر $\left\lceil \frac{n}{2} \right\rceil$ می باشد).

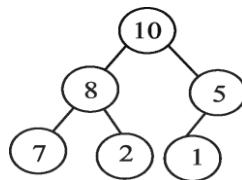
بزرگترین عنصر در یک MinHeap با عناصر متمایز را می توان با $O(n)$ پیدا کرد.

 امین بزرگ ترین عنصر در یک MaxHeap (یا m امین کوچک ترین عنصر در MinHeap)، می‌تواند در یکی از خانه‌های $A[2^m - 1]$ قرار بگیرد. ($m > 1$)

مثال

یک N عنصر متمایز را در نظر بگیرید که با یک آرایه پیاده سازی شده است. چهارمین بزرگترین عنصر در کدام یک از درایه‌ها می‌تواند قرار بگیرد؟

حل: چهارمین بزرگترین عنصر می‌تواند در هر یک از موقعیت‌های 2 تا 15 قرار بگیرد. به طور مثال در درخت MaxHeap زیر، چهارمین بزرگترین عنصر (مقدار 5) در موقعیت 3 قرار دارد:



 کوچکترین عددی که می‌تواند در آخرین سطح یک MinHeap با اعداد متمایز 1 تا n قرار بگیرد، برابر $\lfloor \log n \rfloor + 1$ می‌باشد.

 بزرگترین عددی که می‌تواند در آخرین سطح یک MaxHeap با اعداد متمایز 1 تا n قرار بگیرد، برابر $\lfloor \log n \rfloor - n$ می‌باشد.

 در heap حاوی n گره به عمق d ، به ازای هر j کوچک‌تر از d ، تعداد 2^j گره به عمق j وجود دارد. (n توانی از 2 است)

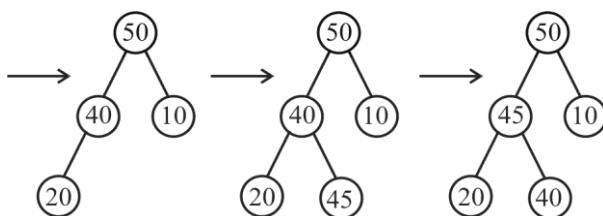
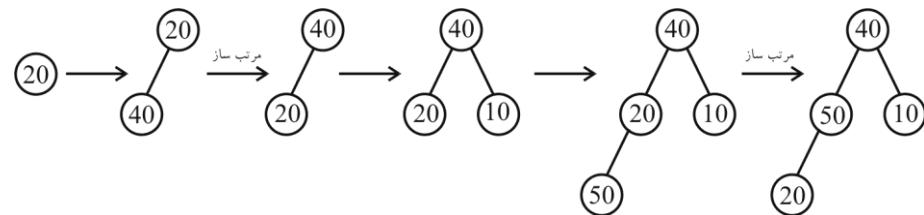
 رابطه بازگشتی $H(n) \leq H(\frac{2n}{3}) + 1$ ، ارتفاع بک هرم با n گره را نشان می‌دهد. چون زیر درخت چپ هرم حداقل $\frac{2n}{3}$ گره دارد.

درج در درخت Heap

در heap، درخت از چپ به راست در هر سطح پرشده، سپس سطح بعدی پر می‌شود. وقتی عنصر جدیدی در درخت وارد می‌شود، ابتدا در سطحی که هنوز به طور کامل پرنشهده در چپ ترین جای خالی قرار می‌گیرد، سپس عمل مرتب‌سازی درخت (ReHeap) صورت می‌پذیرد و گره در پایین‌ترین سطح تا حد امکان با گره پدرس جا به جا می‌شود.

مثال

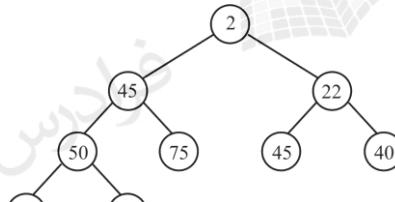
ساختن یک MaxHeap با ورود مقادیر 20 , 40 , 10 , 50 , 45 (از چپ به راست) :

**مثال**

با ورود مقادیر زیر یک درخت MinHeap بسازید.

2 , 50 , 45 , 70 , 75 , 22 , 40 , 55 , 45

حل:

**تابع درج یک عنصر به maxheap**

```

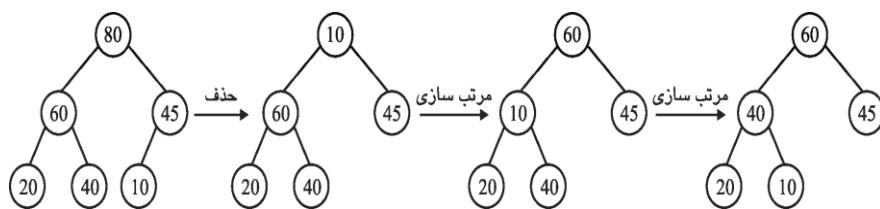
insert (A , n , key){
  if (key<A[i] ) exit( );
  A[i]=key;
  while( (i >1) and ( A[i/2] <A[i] ) ) {
    exchange( A[i] , A[i/2])
    i = i/2;
  }
}
  
```

پیچیدگی تابع درج برابر $O(\lg n)$ می باشد.

 عمل درج یا حذف در درخت Heap به گونه‌ای انجام می‌شود که درخت به صورت Heap باقی بماند. بنابراین بعد از حذف و درج نیاز به عملیات اضافی برای تنظیم درخت می‌باشد.

حذف از درخت Heap

در عمل حذف از Heap، همواره مقدار ریشه حذف شده و سمت راست‌ترین عنصر موجود در پایین‌ترین سطح، در ریشه قرار می‌گیرد و درخت مجدداً تنظیم می‌شود. در شکل زیر یک بار عمل حذف انجام گرفته است:



 اگر همه گره‌های یک MaxHeap را حذف کرده و به همین ترتیب در یک BST خالی درج کنیم، اریب به چپ خواهد شد، چون مقادیر MaxHeap از بزرگ به کوچک بیرون می‌آیند.

 اگر همه گره‌های یک MaxHeap را حذف کنیم، لیست خروجی نزولی خواهد بود و اگر همه گره‌های یک MinHeap را حذف کنیم، لیست خروجی صعودی خواهد بود. به این روش مرتب‌سازی، HeapSort می‌گویند.

رابطه بازگشتی برای تعداد MinHeap‌های متقاوی که می‌توان با $n = 2^k - 1$ عنصر مجزا ساخت، برابر است با:

$$T(n) = \left\lfloor \frac{n-1}{2} \right\rfloor \times T\left(\left\lfloor \frac{n}{2} \right\rfloor\right)^2$$

مثال

با سه عدد متفاوت چند تا MinHeap می‌توان ساخت؟

حل: با ۳ عنصر می‌توان به ۲ صورت یک MinHeap ساخت، کافی است کوچک‌ترین عنصر را در ریشه و دو عنصر دیگر را به ۲ صورت به عنوان فرزندان آن قرار داد. پس $T(3) = 2$.

مثال

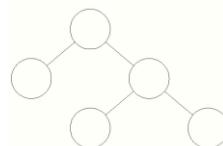
با ۱۵ عدد متفاوت چند تا MinHeap می‌توان ساخت؟

حل:

$$T(15) = \binom{14}{7} \times T(7) \times T(7) = \binom{14}{7} \times 80 \times 80$$

مثال

به چند طریق می‌توان اعداد ۱ تا ۵ را در گره‌های درخت زیر برچسب گذاری کرد تا عدد هر گره از اعداد فرزندان آن بزرگ‌تر باشد؟ (از هر ۵ عدد باید استفاده شود و تکرار مجاز نیست.)



حل:

تعداد حالات را با $T(5)$ نشان می‌دهیم. می‌دانیم که عدد ۵ باید ریشه باشد. از بین چهار عدد باقی مانده یعنی ۱, ۲, ۳, ۴

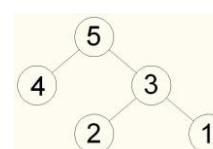
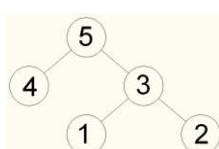
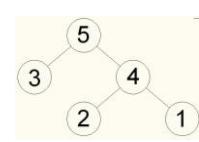
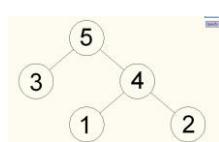
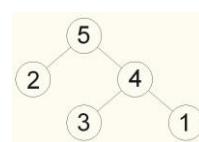
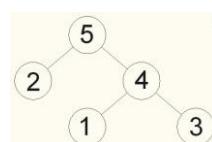
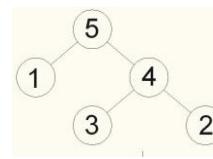
به $\binom{4}{3}$ حالت می‌توان سه عنصر زیر درخت راست را انتخاب کرد:

(1,2,3), (1,2,4), (1,3,4), (2,3,4)

بدیهی است که برای هر انتخاب در زیر درخت راست، فقط یک عنصر برای زیر درخت چپ باقی می‌ماند. به طور نمونه اگر ۳, ۲, ۱ را برای زیر درخت راست انتخاب کنیم، در زیر درخت چپ مقدار ۴ باید انتخاب شود. از طرفی با توجه به شکل، هر سه عنصر را می‌توان به دو حالت در زیر درخت راست نوشت، عدد بزرگ در ریشه و دو عدد دیگر به عنوان فرزندان آن، که جای فرزندان را می‌توان عوض کرد. یعنی $2 = T(3)$. پس تعداد کل حالات برابر است با :

$$T(5) = \binom{4}{3} \times T(3) \times 1 = 4 \times 2 \times 1 = 8$$

این 8 حالت در زیر نشان داده شده است:



 درختی که مقدار کلید هر گره آن بزرگتر یا مساوی کلیدهای فرزندانش باشد را Maxtree و اگر کوچکتر یا مساوی باشد را Mintree می‌گویند.

صف اولویت (Priority Queue)

یکی از رایج ترین کاربردهای heap، استفاده از آن به عنوان یک صف اولویت کارآمد می‌باشد. همانند heap، دو نوع صف اولویت وجود دارد: صف اولویت ماکزیمم و صف اولویت مینیمم.

یک صف اولویت، ساختمان داده‌ای برای نگهداری مجموعه S از عناصری است که هر یک دارای یک مقدار مربوطه بنام key است.

یک صف اولویت ماکزیمم (max-priority queue) اعمال زیر را پشتیبانی می‌کند:

$O(1)$	برگرداندن عنصر با بزرگترین کلید	MAXIMUM(S)
$O(\lg n)$	درج عنصر x	INSERT(S, x)
$O(\lg n)$	حذف و برگرداندن عنصر با بزرگترین کلید	EXTRACT-MAX(S)
$O(\lg n)$	افزایش مقدار کلید عنصر x به مقدار جدید k	INCREASE-KEY(S, x, k)

یک صف اولویت مینیمم (min-priority queue) اعمال زیر را پشتیبانی می‌کند:

برگرداندن عنصر با کوچکترین کلید	MINIMUM(S)
درج عنصر x	INSERT(S, x)
حذف و برگرداندن عنصر با کوچکترین کلید	EXTRACT-MIN(S)
کاهش مقدار کلید عنصر x به مقدار جدید k	DECREASE-KEY(S, x, k)

یکی از کاربردهای صف اولویت ماکزیمم، دسته بندی کارها در یک کامپیوتر مشترک می‌باشد. وقتی کاری پایان یافت، کاری که اولویتی بالاتر از بقیه دارد، توسط EXTRACT-MAX انتخاب می‌شود. توسط INSERT نیز می‌توان کار جدیدی را اضافه کرد.

 هر یک از اعمال، "درج یک عنصر، حذف کوچکترین عنصر، کاهش اولویت یک عنصر" را می‌توان به صورت کارا در یک صف اولویت انجام داد، اما یافتن یک عنصر در صف اولویت را نمی‌توان به صورت کارا انجام داد، چون یک عنصر خاص جای ثابتی ندارد و باید تمام عناصر جستجو شوند.

روشهای نمایش صف اولویت:

حذف	درج	نحوه نمایش صف اولویت
$O(n)$	$O(1)$	آرایه
$O(n)$	$O(1)$	لیست پیوندی
$O(\log n)$	$O(\log n)$	(heap)

تذکر: در جدول بالا، اگر صفت اولویت با آرایه یا لیست پیوندی مرتب شده پیاده سازی شده باشد، آنگاه درج در $O(n)$ و حذف در $O(1)$ انجام می شود.

درخت Deap

یک درخت دودویی کامل است که یا تهی است و یا دارای خواص زیر می باشد:

۱- در ریشه عنصری وجود ندارد.

۲- زیر درخت سمت چپ یک minheap است.

۳- زیر درخت سمت راست یک maxheap است.

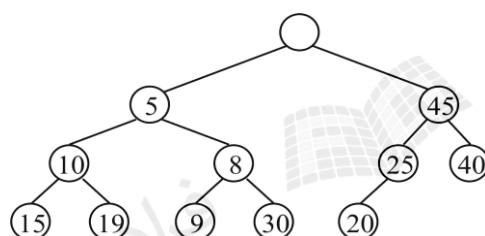
۴- کلید گره زیر درخت چپ کوچکتر یا مساوی کلید گره متناظر در زیر درخت راست می باشد.

 اگر i گرهی در minheap باشد، آنگاه گره متناظر با آن در maxheap $i + 2^{\lfloor \log_2 i \rfloor - 1}$ است.

تذکر: اگر شرط و خاصیت^(۴) تعريف deap، توسط تمام گره های برگ i در minheap برقرار باشد، سپس نتیجه می گیریم که این خاصیت برای کلیه گره های باقیمانده در minheap نیز برقرار است.

مثال

شکل زیر یک Deap یا زده عنصری را نشان می دهد:



مثال

در یک Deap بادوازده گره، شماره گره متناظر با گره با ۴ چه می باشد؟

 حل: با قرار دادن عدد ۴ به جای i در رابطه $i + 2^{\lfloor \log_2 i \rfloor - 1}$ ، مقدار j برابر ۶ خواهد شد.

 ارتفاع Deap $O(\log n)$ می باشد.

 حذف یا درج در Deap $O(\log n)$ از مرتبه $O(n)$ می باشد.

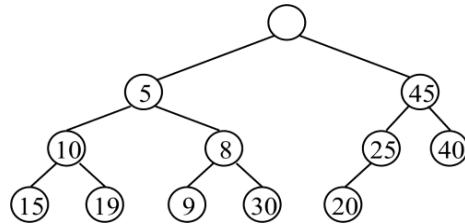
 یک Deap را می توان در یک آرایه ذخیره کرد. در خانه اول آرایه چیزی قرار نمی دهیم و تعداد عناصر آرایه برابر $n-1$ می باشد.

 یک Deap یک heap با دو انتها است که در آن اعمال درج، حذف عنصر مینیمم و حذف عنصر ماکزیمم روی یک صفت اولویت با دو انتها انجام می شود و دارای سرعت بیشتری از heap می باشد.

درج در یک Deap

در زیر :

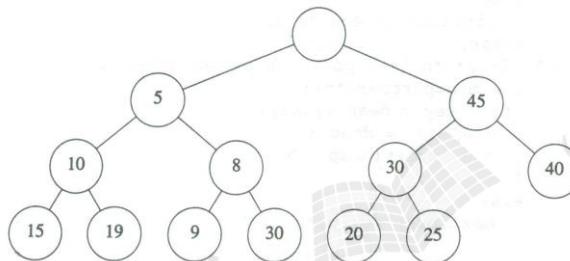
الف- عنصری با کلید 30 را در Deap زیر درج کنید.



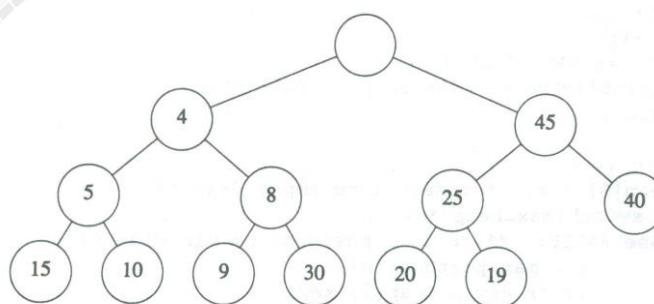
ب- عنصری با کلید 4 را در Deap زیر درج کنید.

حل:

الف- کلید 30 در راست 25 درج می شود و چون 30 از گره متناظرش یعنی 19 بزرگتر است، درج با همان روش در maxheap انجام می شود. یعنی 30 با 25 جایجا می شود.



ب- کلید 4 در راست 25 درج می شود. ولی چون 4 از گره متناظر آن یعنی 19 بزرگتر نیست، 4 و 19 با هم تعویض می شوند. سپس 4 در درخت Minheap به بالا می رود و 5 را با پایین می کشاند.



درخت Treap

درخت Treap از ترکیب ویژگی‌های دو درخت BST و HEAP ساخته می‌شود. در این درخت هر گره x دارای کلید و اولویت می‌باشد. گره‌های treap طوری مرتب می‌شوند که کلیدها از ویژگی BST و اولویت‌ها از ویژگی MinHeap پیروی کنند.

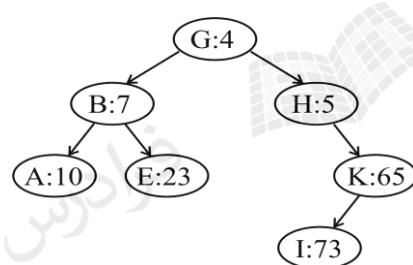
در Treap ، کلید تکراری نداریم. در ، اولویت تکراری نداریم. 

در : Treap

- ۱- اگر v فرزند چپ u باشد، آنگاه: $\text{key}[v] < \text{key}[u]$
- ۲- اگر v فرزند راست u باشد، آنگاه : $\text{key}[v] > \text{key}[u]$
- ۳- اگر u پدر v باشد، آنگاه: $\text{priority}[u] < \text{priority}[v]$

مثال

شکل زیر یک Treap را نشان می‌دهد که کلیدها حرفی و اولویت‌ها به صورت عددی می‌باشند:

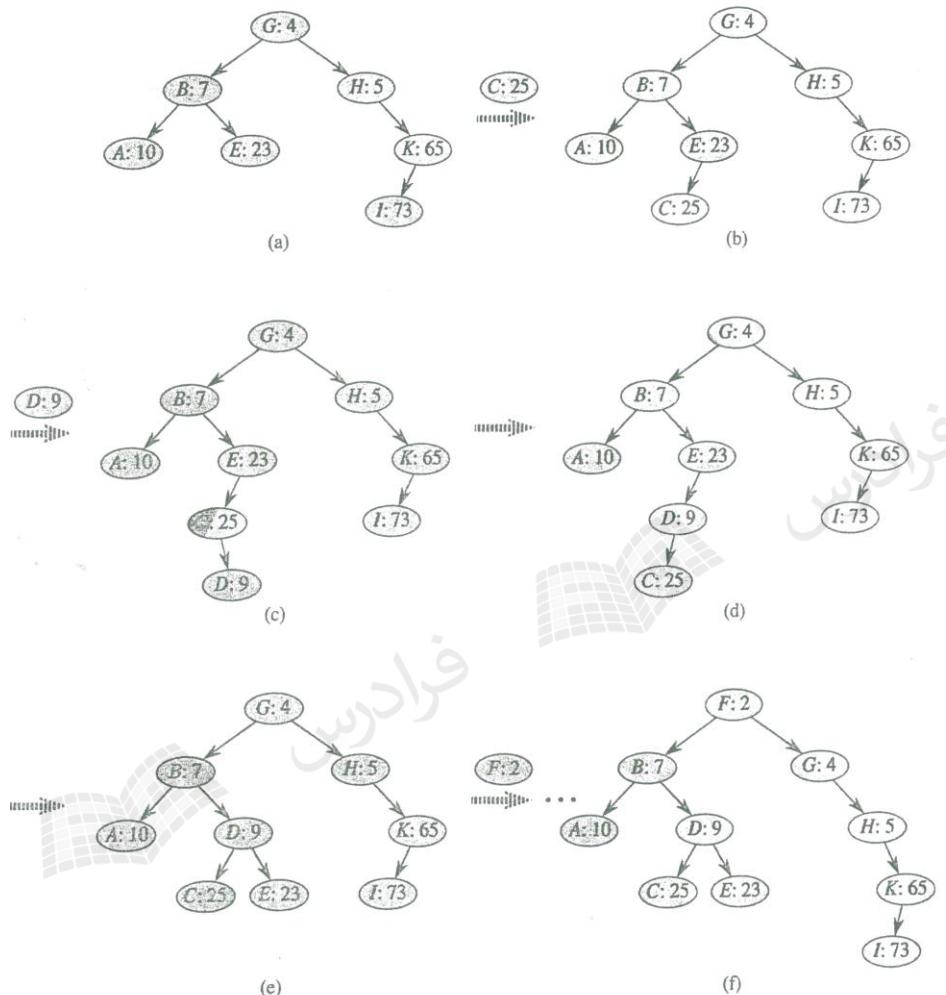


ارتفاع treap برابر $\theta(\log n)$ است و بنابراین زمان جستجوی یک مقدار در treap برابر $\theta(\log n)$ است. 

اگر کلیدها و امتیاز عناصر متمایز باشند، فقط یک treap به صورت یکتا ساخته می‌شود. 

مثال

شکل (a) یک treap را نشان می‌دهد. شکل (b) بعد از درج گره با کلید C و اولویت 25 است. شکل (c) تا (e) مراحل درج کلید D با اولویت 9 را مشخص می‌کند. شکل (f) بعد از درج یک گره با کلید F و اولویت 2 را نشان می‌دهد.



هیپ دو جمله‌ای

برای ادغام دو هیپ دودویی، با متصل کردن دو آرایه که هیپ دودویی را نگهداری می‌کنند و سپس اجرا کردن Min-Heapify در بدترین حالت، زمان $\Theta(n)$ را صرف می‌کند. می‌توان با استفاده از هیپ دو جمله‌ای، عمل ادغام دو هیپ دو جمله‌ای با تعداد کل n عنصر را در زمان $O(\log n)$ انجام داد. هیپ‌های فیبوناچی زمان بهتری برای برخی از اعمال نسبت به هیپ دو جمله‌ای دارند.

هیپ دو جمله‌ای مجموعه‌ای از درخت‌های دو جمله‌ای است. درخت دو جمله‌ای، یک درخت مرتب شده است که به طور بازگشتی تعریف می‌شود. ویژگی‌های درخت دو جمله‌ای B_k

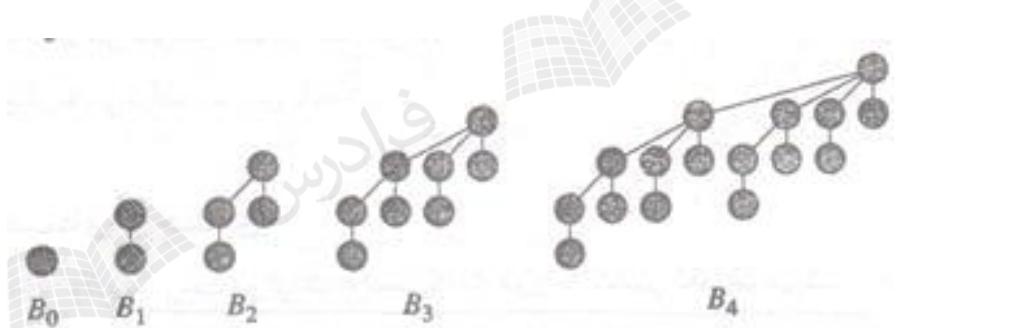
۱- دارای 2^k گره است.

۲- ارتفاع درخت برابر k است.

۳- در عمق i ، دقیقاً $\binom{k}{i}$ گره وجود دارد. ($i = 0, 1, \dots, k$)

۴- درجه ریشه k است که از درجه گره‌های دیگر بزرگتر است. بعلاوه اگر فرزندان ریشه از چپ به راست با شماره گذاری شوند، آنگاه i ریشه زیر درخت B_i است.

شکل زیر درخت‌های B_0 تا B_4 را نشان می‌دهد:



بیشترین درجه هر گره در درخت دو جمله‌ای با n گره برابر $\log n$ است.

هیپ دو جمله‌ای H ، مجموعه‌ای از درخت‌های دو جمله‌ای است که ویژگی‌های زیر را دارا می‌باشد:

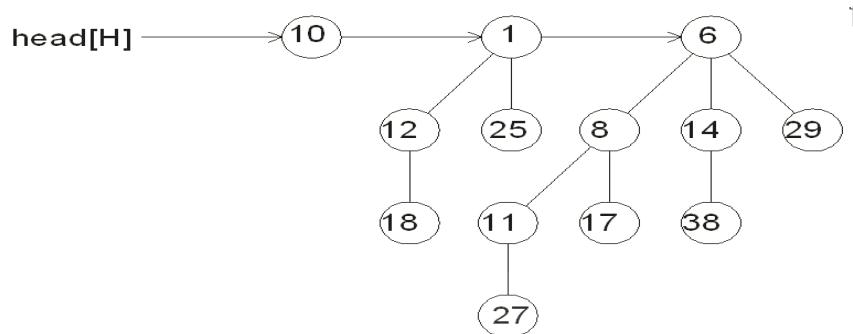
۱- هر درخت دو جمله‌ای در H ، از ویژگی minheap پیروی می‌کند. کلید یک گره، بزرگتر یا مساوی کلید پدرش است.

۲- می‌گوییم چنین درختی minheap مرتب شده است.

۳- برای عدد صحیح غیرمنفی k ، حداکثر یک درخت دو جمله‌ای در H که ریشه‌ای با درجه k دارد وجود دارد.

هیپ دو جمله‌ای با n گره شامل حداکثر $\lfloor \lg n \rfloor + 1$ درخت دو جمله‌ای است.

شکل زیر یک هیپ دو جمله‌ای با ۱۳ گره را نشان می‌دهد که شامل درخت‌های دو جمله‌ای B_0 , B_2 و B_4 است که به ترتیب ۱، ۴ و ۸ گره دارند. همچنین لیست ریشه که یک لیست پیوندی از ریشه‌ها به ترتیب افزایش درجه می‌باشد، نشان داده شده است.

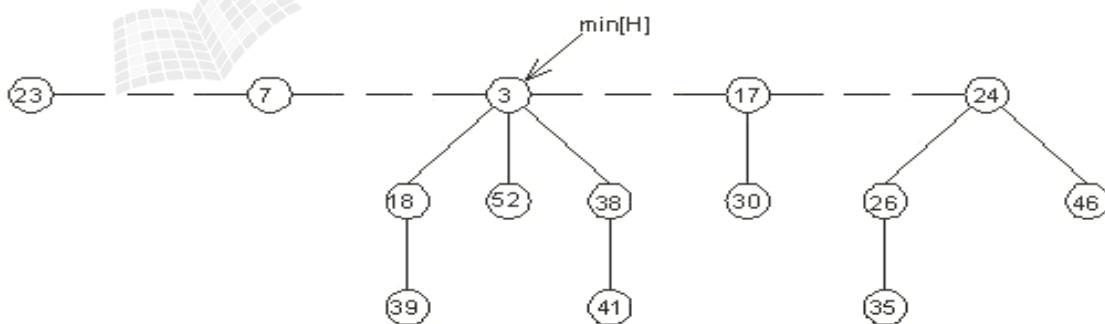


هیپ فیبوناچی

هیپ فیبوناچی مانند هیپ دو جمله‌ای یک مجموعه از درخت‌های minheap مرتب شده است، اما درخت‌ها لزوماً درخت‌های دو جمله‌ای نیستند. به عبارتی هیپ فیبوناچی، مجموعه‌ای از درخت‌های دو جمله‌ای نامرتب است. یک درخت دو جمله‌ای نامرتب، شبیه یک درخت دو جمله‌ای است و به صورت بازگشتی تعریف می‌شود. ویژگی‌های این درخت مانند درخت دو جمله‌ای مرتب استاماً با تفاوت زیر در ویژگی ۴ :

- ۴- برای درخت دو جمله‌ای نامرتب k ، ریشه دارای درجه k است، که بزرگتر از درجه هر گره دیگر است. فرزندان ریشه، ریشه‌های زیر درخت‌های U_0, U_1, \dots, U_{k-1} هستند.

شکل زیر یک هیپ فیبوناچی شامل پنج درخت minheap مرتب شده با ۱۴ گره را نشان می‌دهد:



خط نقطه چین، لیست ریشه را مشخص می‌کند. گره مینیمم heap گرهی است که شامل ۳ کلید می‌باشد. در لیست ریشه از لیست پیوندی دو طرفه چرخشی استفاده می‌شود، تا حذف یک گره و یا اتصال دو لیست در زمان $O(1)$ قابل انجام باشد.

در جدول زیر زمان اجرا در بدترین حالت برای هفت عمل بر روی سه پیاده سازی هیپ های قابل ادغام نشان داده شده است.
برای هیپ فیبوناچی، زمان سرکشن شده نشان داده شده است، نه بدترین حالت.

	هیپ دودویی	هیپ دو جمله‌ای	هیپ فیبوناچی
ساختن هیپ جدید	$\theta(1)$	$\theta(1)$	$\theta(1)$
ادغام دو هیپ	$\theta(n)$	$O(\lg n)$	$\theta(1)$
درج یک گره به هیپ	$\theta(\lg n)$	$O(\lg n)$	$\theta(1)$
حذف گره از هیپ	$\theta(\lg n)$	$\theta(\lg n)$	$O(\lg n)$
حذف گره با کلید مینیمم	$\theta(\lg n)$	$\theta(\lg n)$	$O(\lg n)$
کاهش مقدار یک کلید	$\theta(\lg n)$	$\theta(\lg n)$	$\theta(1)$
پیدا کردن گره با کلید مینیمم	$\theta(1)$	$O(\lg n)$	$\theta(1)$

کنکور ارشد

(مهندسی کامپیوتر - دولتی ۸۵)

- ۱- آرایه T که در آن تعدادی از خانه‌ها هنوز مقداردهی نشده اند را در نظر می‌گیریم. با قرار دادن مقادیر کدام یک از موارد زیر، این آرایه به یک هیپ تبدیل خواهد شد؟

	1	2	3	4	5	6	7	8	9	10
T	75		30		17	28	20	7		

$$T[2]=30 ; T[4]=9 ; T[9]=10 ; T[10]=6 \quad (1)$$

$$T[2]=15 ; T[4]=15 ; T[9]=17 ; T[10]=10 \quad (2)$$

$$T[2]=39 ; T[4]=30 ; T[9]=32 ; T[10]=16 \quad (3)$$

$$T[2]=21 ; T[4]=14 ; T[9]=0 ; T[10]=16 \quad (4)$$

پاسخ: جواب گزینه ۴ است.

- در یک MaxHeap، کلید هر گره از فرزندانش بزرگتر یا مساوی است. به عبارتی باید $T[i]$ از $T[2i]$ و $T[2i+1]$ بیشتر یا مساوی باشد. بنابراین گزینه ۱ و ۲ نادرست است، چون $T[4]$ بزرگتر از $T[9]$ است. البته می‌توان درخت هر گزینه را رسم کرد. فقط درخت گزینه ۴، Maxheap است:

	1	2	3	4	5	6	7	8	9	10
T	75	21	30	14	17	28	20	7	0	16

(مهندسی IT - دولتی ۸۶)

- ۲- سومین کوچکترین کلید در یک MinHeap با کلیدهای متمایز در درایه‌هایی با چه اندیس‌هایی می‌تواند باشد؟

$$1,2,3,4,5,6,7 \quad (4) \qquad 4,5,6,7 \quad (3) \qquad 2,3,4,5,6,7 \quad (2) \qquad 1,2,3 \quad (1)$$

پاسخ: جواب گزینه ۲ است.

- با توجه به اینکه m امین کوچک ترین عنصر در MinHeap، می‌تواند در یکی از خانه‌های $A[2^m - 1]$ تا $A[2^m - 1]$ قرار بگیرد، سومین کوچکترین عنصر می‌تواند از اندیس ۲ تا اندیس ۷ قرار بگیرد.

(مهندسی کامپیوتر - دولتی ۸۰)

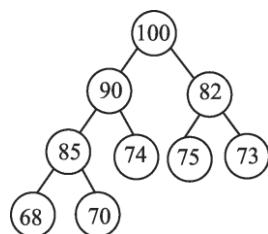
-۳- آرایه زیر یک Heap است. برای درج عدد ۹۵ در آرایه به گونه‌ای که آرایه نهایی نیز وضعیت Heap داشته باشد، چند عمل exchange (تعویض دو کمیت) لازم است؟

100	90	82	85	74	75	73	68	70
-----	----	----	----	----	----	----	----	----

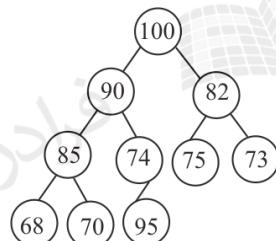
8 (۴) 6 (۳) 4 (۲) 2 (۱)

پاسخ: جواب گزینه ۱ است.

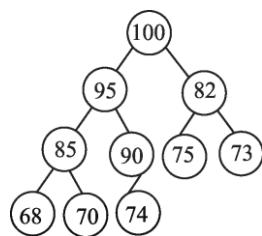
با توجه به آرایه داده شده، درخت MaxHeap به صورت زیر خواهد بود.



عدد ۹۵ در آخرین سطح درج می‌شود و در صورت نیاز درخت تنظیم می‌شود:



بعد از درج ، محل ۹۵ با ۷۴ تعویض می‌شود و سپس ۹۵ با ۹۰ تعویض می‌شود و درخت زیر حاصل می‌شود:



(مهندسی IT - دولتی ۹۰)

-۴- یک MaxHeap با n عنصر را که در آرایه $A[1..n]$ قرار دارد، در نظر بگیرید. مرتبه زمانی الگوریتم حذف عنصر i ام از این Max Heap به گونه‌ای که ساختار Max Heap را حفظ کند، چیست؟

n (۴) $\log n$ (۳) $n \log n$ (۲) ۱ (۱)

حل : جواب گزینه ۳ است. مرتبه زمانی الگوریتم حذف یک عنصر از درخت Heap به ارتفاع درخت که برابر $\log n$ است، بستگی دارد و برابر $O(\log n)$ می‌باشد.

(علوم کامپیوتر - دولتی) ۸۹

۵- یک درخت کدام از عملیات زیر همچنان MaxHeap می‌ماند. فرض کنید که عمل خواسته شده روی یک یا چند گره دلخواه درخت اعمال شود؟

(۱) left Rotate (چرخش به چپ)

(۲) Mirror (تعویض بجهه‌های چپ و راست گره)

(۳) Right rotate (چرخش به راست)

(۴) Exchange (تعویض مقدار موجود در گره با بزرگترین فرزند)

پاسخ: گزینه ۲ جواب است.

بچه‌های چپ و راست یک MaxHeap هیچ ارتباطی با هم ندارند و اگر آنها را تعویض کنیم، درخت باز هم می‌ماند. (در گره از فرزندانش بزرگتر یا مساوی است و بین فرزندان یک گره، ارتباطی وجود ندارد).



(مهندسی IT - دولتی) ۹۰

۶- چند Min Heap با هفت عنصر که حاوی کلیدهای متمایز یک تا هفت (با هر ترتیب دلخواه) است، می‌توان ساخت؟

(۱) ۲۰ (۲) ۸۰ (۳) ۶۰ (۴) ۴۰

حل: جواب گزینه ۲ است.

کوچکترین عنصر را در ریشه قرار می‌دهیم. از بین ۶ عنصر باقی مانده، ۳ عنصر را به $\binom{6}{3}$ حالت انتخاب می‌کنیم و با آن‌ها

به ۲ صورت، زیر درخت سمت چپ و با یقیه نیز به ۲ دو صورت، زیر درخت سمت راست می‌سازیم. پس تعداد کل حالت‌ها برابر است با:

$$T(7) = \binom{6}{3} \times T(3) \times T(3) = 20 \times 2 \times 2 = 80$$

فصل ۹:

گراف

گراف

گراف G ، شامل دو مجموعه V و E است. V مجموعه محدود و غیرتنهی از رئوس و E مجموعه‌ای از زوج رئوس (که یال نامیده می‌شود) است. گراف با عبارت $(V, E) = G$ مشخص می‌شود.

گراف را می‌توان به دو نوع تقسیم کرد:

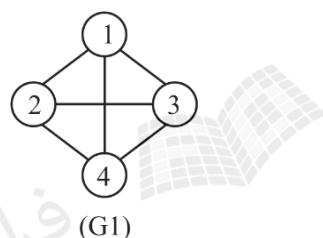
۱- گراف جهت‌دار

گرافی که در آن ترتیب گره‌ها در زوج مرتب اهمیت داشته باشد، یعنی زوجهای $\langle v_1, v_2 \rangle$ و $\langle v_2, v_1 \rangle$ دو یال متفاوت را نمایش دهند.

۲- گراف غیر جهت‌دار: گرافی که در آن جای دو رأس در مجموعه یالها اهمیت نداشته باشد.

مثال

برای گراف بدون جهت G_1 ، مجموعه V و E را مشخص نمایید.



پاسخ:

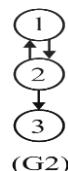
$$V(G_1) = \{1, 2, 3, 4\}$$

$$E(G_1) = \{(1,2), (1,3), (1,4), (2,3), (2,4), (3,4)\}$$



مثال

برای گراف جهت دار G_2 ، مجموعه V و E را مشخص نمایید.



پاسخ:

$$V(G_2) = \{1, 2, 3\}, E(G_2) = \{\langle 1, 2 \rangle, \langle 2, 1 \rangle, \langle 2, 3 \rangle\}$$



درجه یک گره، تعداد یالهای گذرنده از آن می‌باشد.

تعداد یالها در هر گرافی برابر $\frac{1}{2} \sum_{i=1}^n d_i$ می‌باشد. (d_i : درجه رأس i) (n: تعداد رئوس)

تعداد گره‌ها با درجه فرد در یک گراف بدون جهت، همواره عددی زوج می‌باشد.

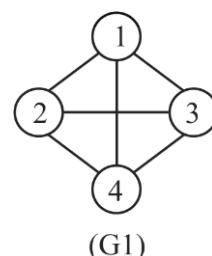
در یک گراف جهت دار، گره با درجه خروجی مثبت و درجه ورودی صفر را گره منبع و گره با درجه ورودی مثبت درجه خروجی صفر را گره چاه می‌نامند.

مسیری که همه رئوس آن مجزا باشند، بجز (احتمالاً) اولی و آخری را مسیر ساده می‌گویند.

یک مسیر ساده که اولین و آخرین راس آن یکسان باشد را حلقه می‌گویند.

مثال

چند نمونه از مسیر ساده و حلقه در گراف زیر را مشخص کنید.



پاسخ: مسیر $3 \rightarrow 4 \rightarrow 1 \rightarrow 2 \rightarrow 3$ در G_1 می‌باشد. مسیر $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$ یک حلقه به طول 3 در G_1 است. ■

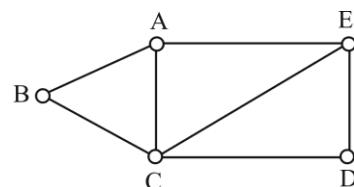
مثال

با توجه به گراف زیر موارد زیر مشخص نمایید:

۱- از B به E چند مسیر ساده به طول دو وجود دارد؟

۲- از B به D چند مسیر ساده به طول دو وجود دارد؟

۳- گراف زیر دارای چند حلقه به طول چهار می‌باشد؟



پاسخ:

از B به E مسیرهای ساده BAE , BCE به طول 2 وجود دارد. از B به D یک مسیر ساده BCD به طول 2 وجود دارد و تعداد دو حلقه $AECBA$ و $AEDCA$ به طول 4 وجود دارد. ■

انواع گراف

انواع گراف در جدول زیر آورده شده است:

گراف ساده	گرافی که طوقه و یال موازی نداشته باشد.
گراف چند گانه (Multi Graph)	گرافی که در آن یالهای چندگانه (Multi Edge) مجاز باشد.
گراف کامل	گراف بدون جهتی که همه یالهای آن رسم شده باشد.
گراف وزن دار(شبکه)	گرافی که به هر یال آن یک مقدار عددی نسبت داده شده باشد.
گراف درختی	گرافی که حلقه نداشته باشد.
گراف همبند (متصل)	گرافی که یک مسیر بین هر دو گره آن وجود داشته باشد.
گراف همبند قوی	گراف جهتداری که برای هر زوج گره v, u هم یک مسیر از u به v و هم یک مسیر از v به u وجود داشته باشد.

چند نمونه :

	گراف چند گانه		گراف کامل
	گراف ناهمبند		گراف درختی

یالهای متمايز که نقاط پایانی یکسانی را به هم وصل می کنند، را یالهای چند گانه می گویند.

در گراف کامل، درجه هر یک از گره ها برابر $n-1$ و تعداد یالهای آن برابر $\frac{n(n-1)}{2}$ می باشد.

 گراف مکمل یک گراف، هیچ یال مشترکی با گراف ندارد و مجموع دو گراف، یک گراف کامل خواهد

بود. گرافهای زیر مکمل یکدیگرند:



نمایش گراف

یک گراف را می‌توان به روشهای زیر نمایش داد:

۱- ماتریس مجاورتی

۲- لیستهای مجاورتی

۳- لیستهای مجاورتی معکوس

۴- لیستهای مجاورتی چند گانه

تذکر: به جای اصطلاح مجاورتی از همجاواری یا همسایگی نیز استفاده می‌شود.

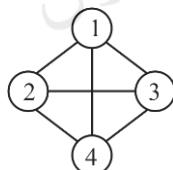
ماتریس مجاورتی

ماتریس مجاورتی گراف $G=(V,E)$ با n راس، آرایه‌ای $n \times n$ به نام A می‌باشد که بصورت زیر تعریف می‌شود.

$$\begin{cases} A[i, j] = 1 & (Vi, Vj) \in E(G) \\ A[i, j] = 0 & (Vi, Vj) \notin E(G) \end{cases}$$

مثال

ماتریس مجاورتی گراف زیر را بدست آورید.



پاسخ: درایه‌های سطر اول به غیر از اولین درایه، همه برابر یک می‌باشند، چون گره ۱ با بقیه گره‌ها مجاور است و فقط با خودش مجاور نمی‌باشد. چنین تحلیلی نیز برای سطوحی دیگر می‌توان انجام داد.

	1	2	3	4
1	0	1	1	1
2	1	0	1	1
3	1	1	0	1
4	1	1	1	0

 درجه هر رأس مانند ۳ در گراف بدون جهت، مجموع تعداد یکهای موجود در سطر ۳ ماتریس است

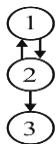
 تعداد یکهای ماتریس مجاورتی گراف بدون جهت، همواره دو برابر تعداد یال‌ها می‌باشد.

 ماتریس هم جواری گراف بدون جهت، متقارن است ولی برای گراف جهت‌دار، لزوماً متقارن نمی‌باشد.

 تمامی درایه‌های ماتریس هم‌جواری گراف کامل به غیر از قطر اصلی برابر یک می‌باشند.

مثال

ماتریس مجاورتی گراف جهت دار زیر را بدست آورید.



پاسخ:

	1	2	3
1	0	1	0
2	1	0	1
3	0	0	0

تعداد یک های ماتریس مجاورتی گراف جهت دار، همواره برابر تعداد یال ها می باشد.

در یک گراف جهت دار، درجه خروجی هر رأس، مجموع عناصر سطری آن و درجه ورودی هر رأس، مجموع عناصر ستونی آن رأس می باشد.

اگر در ماتریس هم جواری، درایه واقع در سطر و ستون یک گره برابر یک باشد، یک طوقه در آن گره وجود دارد.(طوقه یعنی حلقه به طول یک)

در یک گراف جهت دار، اگر ماتریس هم جواری مثلثی باشد، گراف بدون دور است.

اگر گرافی Acyclic (فاقد دور و حلقه) باشد، آنگاه ماتریس هم جواری آن مثلثی می باشد.

لیستهای مجاورتی (Adjacency List)

در این روش نمایش برای هر گره در گراف یک لیست وجود دارد و اشاره‌گر شروع این لیستها در یک آرایه قرار دارند. لیست سطر i حاوی رئوسی است که از گره i به آنها مسیری به طول یک وجود دارد.

مثال

لیست مجاورتی گراف داده شده رارسم نمایید.

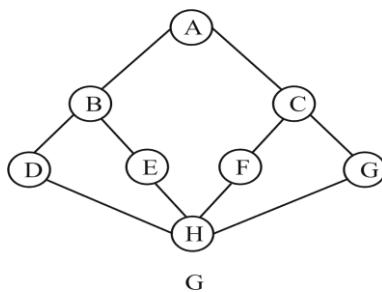


پاسخ:

[1]	• → [2 nil]
[2]	• → [3 •] → [1 nil]
[3]	nil

مثال

لیست مجاورتی گراف داده شده را رسم نمایید.



پاسخ:

[A]	
[B]	
[C]	
[D]	
[E]	
[F]	
[G]	
[H]	

درجه هر راس در یک گراف غیر جهتدار را می‌توان با شمارش تعداد گره‌های متناظر آن در لیست مجاورتی تعیین کرد.

اگر تعداد یال‌ها در یک گراف زیاد باشد، از روش ماتریس مجاورتی و اگر تعداد یال‌ها کم باشد از روش لیست مجاورتی جهت نمایش گراف استفاده می‌شود. بنابراین برای نمایش گراف‌های پراکنده از لیست هم‌جاواری استفاده می‌شود.

برای نمایش گراف $G = (V, E)$ به روش لیست هم‌جاواری، فضای مصرفی برابر با $O(|E| + |V|)$ می‌باشد.

لیستهای مجاورتی معکوس

لیست مجاورتی معکوس مشابه لیست مجاورتی است، با این تفاوت که در لیست سطر i ، رؤوسی قرار می‌گیرند که از آنها به گره i ، مسیری به طول یک وجود دارد. توسط لیست مجاورتی معکوس می‌توان به سادگی درجه ورودی یک راس را مشخص کرد.

مثال ۱۰-۱۰

لیست مجاورتی معکوس گراف زیر رارسم نمایید.



پاسخ:

[1]	
[2]	
[3]	



در گراف بدون جهت، لیست مجاورتی و لیست مجاورتی معکوس یکسان می‌باشند.

معکوس یک گراف

برای بدست آوردن معکوس گراف جهت دار G که آن را با \tilde{G} نمایش می‌دهند، کافی است که جهت یال‌ها را در G معکوس می‌کنیم.

اگر A ماتریس همسایگی گراف جهت دار G باشد، آنگاه ترانهاده A یعنی A^T ، ماتریس همسایگی گراف \tilde{G} می‌باشد.



پیمایش گراف

پیمایش یک گراف به منظور ملاقات کلیه گره‌های آن انجام می‌گیرد و به دو طریق ممکن می‌باشد:

۱- عمقی (پیمایش اول - عمق) (DFS : Depth First Search)

۲- سطحی (پیمایش اول - عرض) (BFS : Breadth First Search)

در پیمایش DFS از پشته و در پیمایش BFS از صف استفاده می‌شود.



پیمایش سطحی (BFS)

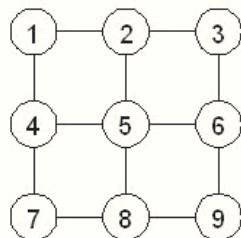
پیمایش سطحی منحصر به فرد نیست. در یک روش با شروع از یک گره، ابتدا گره را وارد صف کرده و سپس کلیه گره‌های مجاور با آن (فرزندان گره از چپ به راست) را در صف درج می‌کنیم. حال عنصر بعدی از صف را حذف کرده و گره‌های مجاور آن را درج می‌کنیم و این عمل را ادامه داده تا همه گره‌ها پیمایش شوند.

پیمایش عمقی (DFS)

پیمایش عمقی منحصر به فرد نیست. در یک روش با شروع از یک گره، آن را ملاقات کرده و سپس کلیه گره‌های سمت چپ را تا آخرین عمق پیمایش می‌کنیم. اگر در پایین رفتهای متوالی، گره‌ای ملاقات شود که هیچ گره هم‌جواری نداشته باشد یا تمام گره‌های هم‌جوار آن ملاقات شده باشد، برگشت به یک سطح بالا انجام می‌گیرند و روند فوق تکرار می‌شود. این الگوریتم از پشته استفاده می‌کند.

مثال ۱۱-۱۰

سه پیمایش DFS برای گراف زیر مشخص کنید.



پاسخ:

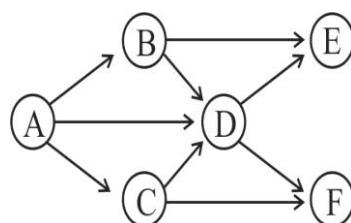
1 , 4 , 7 , 8 , 9 , 6 , 5 , 2 , 3

1 , 2 , 3 , 6 , 9 , 8 , 7 , 4 , 5

1 , 2 , 3 , 6 , 5 , 4 , 7 , 8 , 9

مثال ۱۲-۱۰

چند پیمایش BFS و DFS برای گراف زیر مشخص کنید.



پاسخ:

چند پیمایش BFS به صورت زیر است:

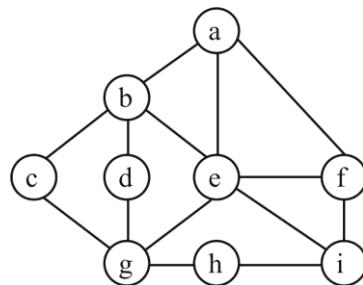
ABCDEF , ADBCEF , ADBCFC

و چند پیمایش DFS :

ABEDFC , ABDFEC , ADEFBC

مثال ۱۳-۱۰

یکی از پیمایش‌های BFS و DFS گراف زیر را مشخص کنید.

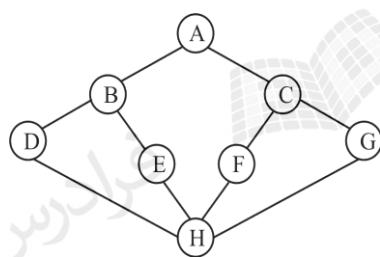


پاسخ:

BFS : a b e f c d g i h
DFS : a b c g d e f i h

**مثال ۱۴-۱۰**

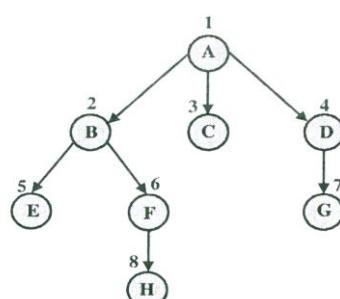
یکی از پیمایش‌های BFS گراف زیر را مشخص کنید.



پاسخ: ABCDEFGH

مثال ۱۵-۱۰

پیمایش DFS گراف زیر را به دست آورید؟

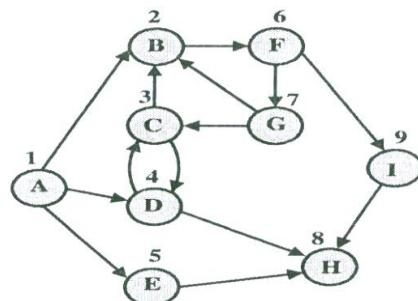


پاسخ: ABEFHCDG



مثال ۱۶-۱۰

پیمایش BFS و DFS گراف زیر را به دست آورید؟



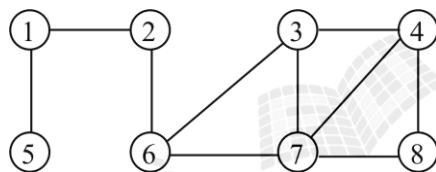
پاسخ:

BFS : ABDEFCHGI

DFS : ABFIHGCDE

**مثال ۱۷-۱۰**

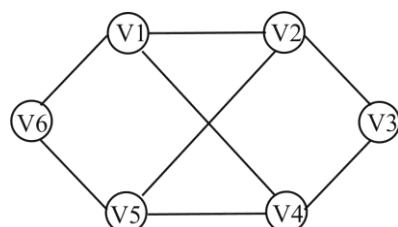
یکی از پیمایش‌های BFS گراف زیر با شروع از گره شماره ۲ را مشخص کنید.



حل : 2 , 6 , 1 , 3 , 7 , 5 , 4 , 8

**مثال ۱۸-۱۰**

یکی از پیمایش‌های BFS گراف زیر را به کمک صف مشخص کنید.



پاسخ:

پیمایش BFS از صف استفاده کرده و در صف درج به انتهای حذف از ابتداء انجام می‌گیرد.

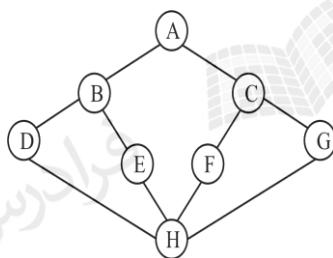
- ۱- درج V1 در ابتدای صف
 - ۲- حذف و چاپ V1
 - ۳- درج فرزندان V1 یعنی V6,V4,V2
 - ۴- حذف و چاپ اولین فرزند V1 که درج کرده ایم (یعنی V6)
 - ۵- درج فرزند V6 (یعنی V5)
 - ۶- حذف و چاپ گره ابتدای صف(یعنی V4)
 - ۷- درج فرزند V4 (یعنی V3)
 - ۸- حذف و چاپ گره ابتدای صف(یعنی V2)
 - ۹- حذف و چاپ گره ابتدای صف(یعنی V5)
 - ۱۰- حذف و چاپ گره ابتدای صف(یعنی V3)
- بنابراین حاصل پیمایش BFS گراف برابر است با:

V1 , V6 , V4 , V2 , V5 , V3



مثال ۱۹-۱۰

یکی از پیمایش‌های DFS گراف زیر را به کمک پشته مشخص کنید.



پاسخ:

حاصل پیمایش عمقی گراف داده شده برابر است با: ABDHEFCG

الگوریتم‌های BFS، DFS، اگر در مورد یک گراف همبند بکار برد شدن، از $(n-1)$ ضلع گراف که درز هم ندارند، استفاده می‌کنند. (n : تعداد گره‌ها)

در الگوریتم‌های BFS، DFS، یالهای (اضلاع) مورد استفاده، یک درخت را می‌سازند.

هزینه زمانی هر یک از پیمایش‌های BFS و DFS، گراف $G=(V,E)$ که به صورت ماتریس مجاورت بیان شده است، برابر $|V|^2$ است و اگر به صورت لیست مجاورت بیان شده است، برابر $|E|$ است.

 تعداد یالهای گراف کامل برابر $\frac{n(n-1)}{2}$ و تعداد یالهای انتخاب شده در پیمایش DFS برابر $(n-1)$ است.

می باشد. بنابراین تعداد یالهای ملاقات نشده در پیمایش DFS برابر است با :

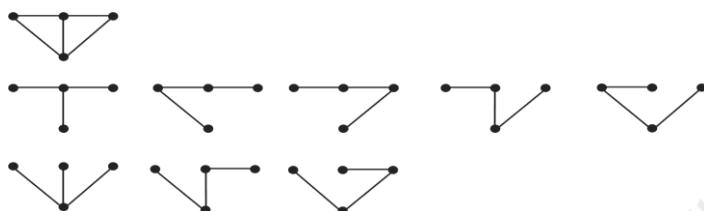
$$\frac{n(n-1)}{2} - (n-1) = \frac{(n-1)(n-2)}{2}$$

درخت پوشای (Spanning Tree)

درختی شامل تمامی رئوس گراف و تعدادی از یالهای گراف را درخت پوشای گراف می نامند. درخت پوشای یک گراف با n گره، دارای حداقل $n-1$ یال است.

مثال ۲۰-۱۰

برای گراف زیر ۸ درخت پوشای رسم شده است:

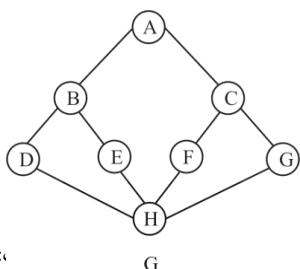


هر درخت فرآگیر را می‌توان با حذف ۲ یال از ۵ یال گراف داده شده به دست آورد. این کار با ۱۰ روش انجام می‌شود اما ۲ تا از آنها منتهی به گراف ناهمبند می‌شوند.

 یک گراف کامل با n راس، حداقل دارای $1 - 2^{(n-1)}$ درخت پوشای می باشد.

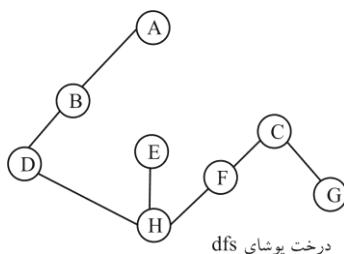
مثال ۲۱-۱۰

درخت پوشای حاصل از جستجوی عمقی گراف G را بدست آورید؟



∴

پاسخ: پیمایش DFS برای G می باشد، بنابراین درخت پوشای

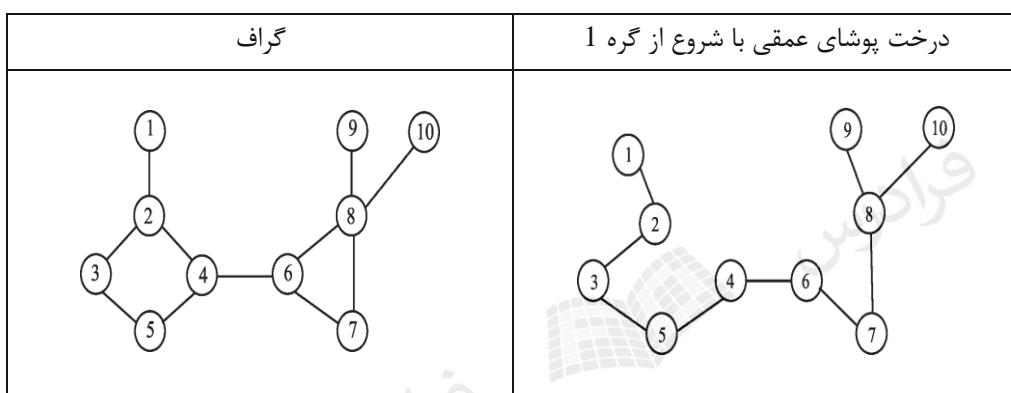


 درخت پوشای حاصل از پیمایش عمقی گراف را درخت پوشای عمقی و درخت پوشای حاصل از پیمایش سطحی گراف را درخت پوشای سطحی، می نامند.

 اگر G یک گراف متصل و T یکی از درختهای پوشای عمقی آن باشد، آنگاه G هیچ لبه متقاطع با T ندارد.

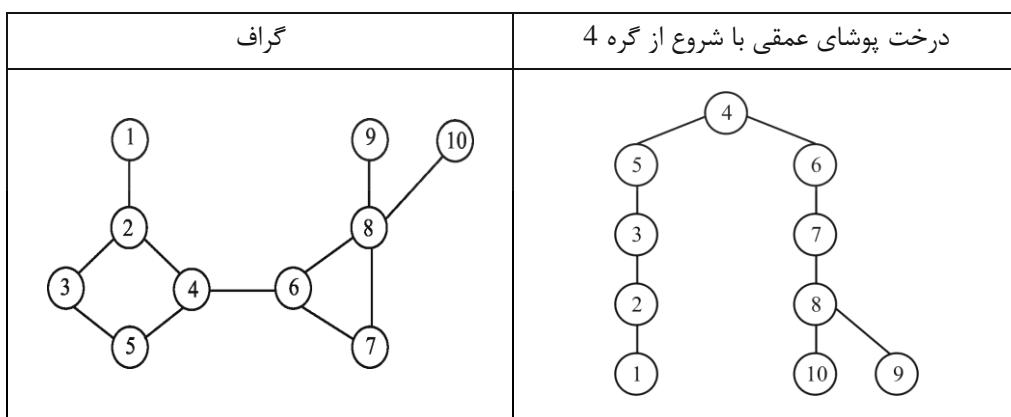
مثال ۲۲-۱۰

رسم درخت پوشای عمقی گراف با شروع از گره ۱ :



مثال ۲۳-۱۰

درخت پوشای عمقی گراف با شروع از گره ۴ :



درخت پوشای حداقل (MST)

درخت پوشایی که در بین تمام درختهای پوشای حداقل وزن باشد را MST می‌نامند. برای به دست آوردن MST توان از الگوریتمهای زیر استفاده کرد: ۱- کراسکال(راشال) ۲- پریم ۳- سولین

 در این روشها باید دقیقاً از $n-1$ یال استفاده شود و نباید از یال‌هایی که حلقه تولید می‌کنند، استفاده کرد.

الگوریتم کراسکال

مراحل اجرای این الگوریتم به صورت زیر می‌باشد:

۱- مرتب کردن صعودی تمام یال‌ها.

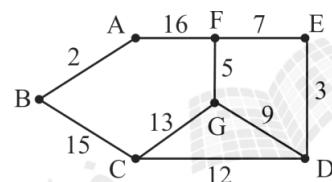
۲- مقدار دهی اولیه T ، به طوری که یک گراف مشکل از همان گره‌های G و بدون یال باشد.

۳- تکرار عملیات زیر به تعداد $n-1$ مرتبه:

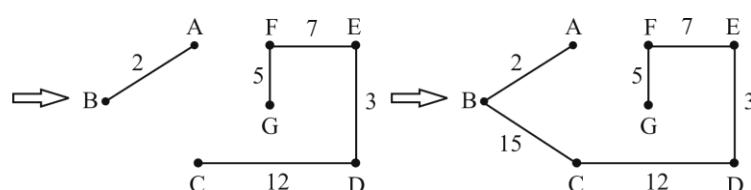
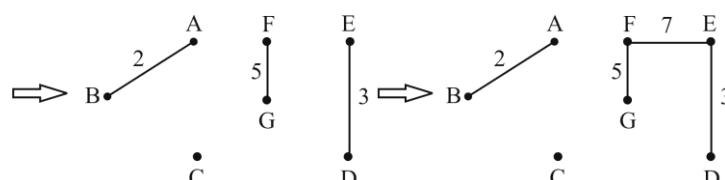
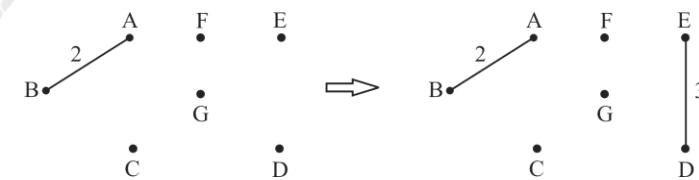
"به T یک یال از G با حداقل وزن اضافه کن به طوری که در T تشکیل حلقه ندهد."

مثال ۲۴-۱۰

به کمک الگوریتم کراسکال، درخت پوشای حداقل گراف زیر را بدست آورید.

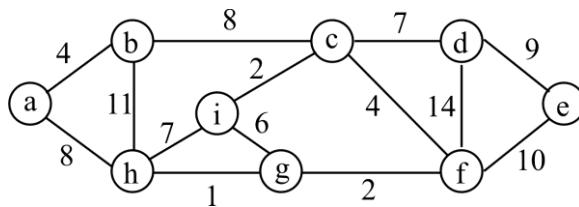


پاسخ:



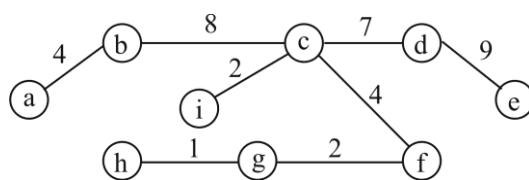
مثال ۲۵-۱۰

وزن درخت پوشای مینیمم گراف در صورت استفاده از الگوریتم کراسکال چه می‌باشد؟



حل:

ترتیب اضافه شدن یال‌ها در روش کراسکال برابر است با: 1,2,2,4,4,7,8,9 ، بنابراین وزن یال‌ها در درخت پوشای مینیمم برابر 37 می‌باشد.



درخت پوشای بدست آمده، منحصر به فرد نیست، چون می‌توان به جای یال (a,h) از یال (b,c) استفاده کرد.

در گرافی که یال‌هایی با وزن مساوی دارد، احتمال دارد بتوان بیش از یک درخت پوشای حداقل رسم کرد، اما وزن همه آنها برابر است.

الگوریتم پریم

الگوریتم پریم نیز مشابه کراسکال، درخت پوشای کمترین هزینه را لبه به لبه می‌سازد با این تفاوت که یالی که در روش کراسکال اضافه می‌شود، می‌تواند به یال‌هایی که قبلاً اضافه شده متصل نباشد، اما در پریم باید متصل باشد. در روش پریم از یک گره شروع کرده و گره ای را به آن وصل می‌کنیم که نزدیکتر است. حال گره ای را که به یکی از این دو گره نزدیکتر است را وصل می‌کنیم.

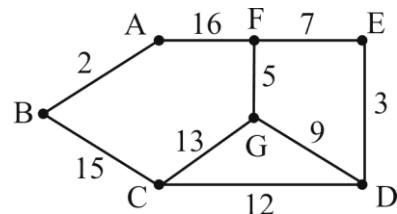
مرتبه اجرایی پریم برابر $O(n^2)$ و مرتبه اجرایی کراسکال برابر $O(e \times \log e)$ می‌باشد.

وزن درخت پوشای حداقل حاصل از هر یک الگوریتم‌های پریم یا کراسکال با هم برابر می‌باشد.

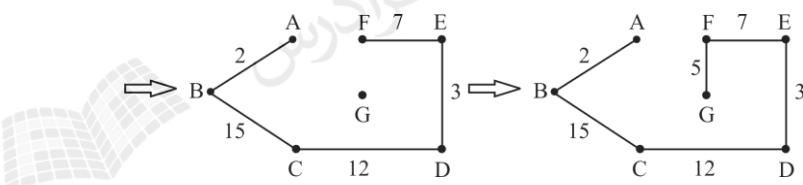
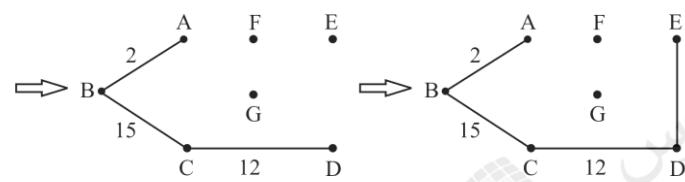
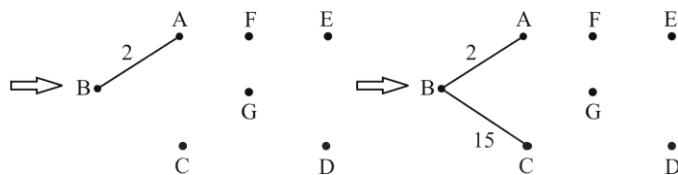
در هر یک از مراحل اجرای الگوریتم پریم، مجموعه لبه‌های انتخابی یک درخت را می‌سازند در حالی که در کراسکال در هر لحظه، یک جنگل را می‌سازد.

مثال ۲۶-۱۰

به کمک الگوریتم پریم، درخت پوشای حداقل گراف زیر را بدست آورید.(با شروع از گره a)

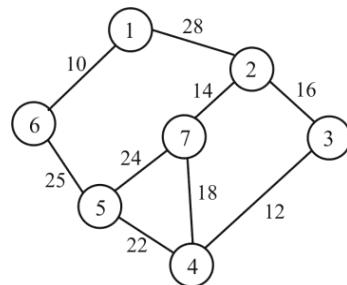


پاسخ:



مثال ۲۷-۱۰

ترتیب اضافه شدن یالها را در صورت استفاده از الگوریتم پریم و کراسکال مشخص نمایید.



حل:

ترتیب اضافه شدن یالها در روش کراسکال برابر است با: 10,12,14,16,22,25

و در روش پریم برابر است با : 10,25,22,12,16,14

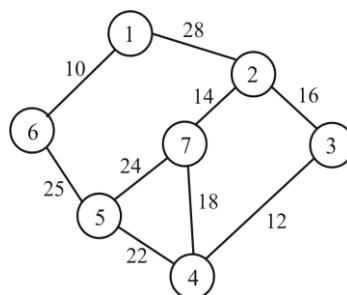
برای یافتن درخت پوشای حداقل یک گراف خلوت از الگوریتم کراسکال استفاده می شود. چون این الگوریتم بر پایه یالها کار می کند.(گراف خلوت، گرافی است که تعداد یالهای آن کم باشد.)

الگوریتم سولین

در این الگوریتم در هر مرحله چندین لبه انتخاب می شود. در شروع هر مرحله، لبه های انتخابی، به همراه همه گراف برداری، یک جنگل پوشای تشکیل می دهند. در طی هر مرحله، یک لبه برای هر درخت در این جنگل انتخاب می شود و این لبه، لبه ای با کمترین هزینه است که دقیقاً یک بردار در درخت دارد.

مثال ۳۰-۱۰

درخت پوشای حداقل گراف زیر را با استفاده از الگوریتم سولین بدست آورید.



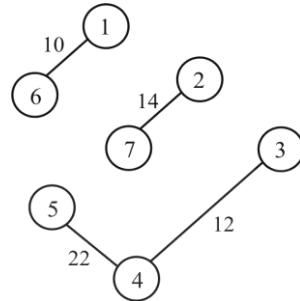
پاسخ:

هر درخت در این جنگل پوشان، یک بردار منفرد است. لبه‌های انتخابی با رئوس ۱ تا ۷، به ترتیب برابر است با: $(1,6), (2,7), (3,4), (4,3), (5,4), (6,1), (7,2)$

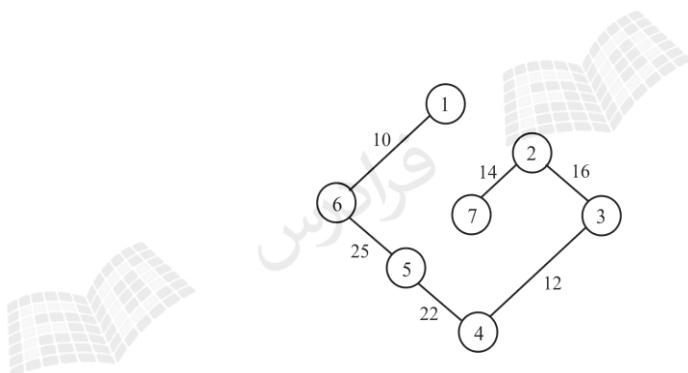
که لبه‌های مجازی این انتساب برابر است با:

$(1,6), (2,7), (3,4), (5,4)$

با افزودن اینها به مجموعه لبه‌های انتخابی، شکل زیر حاصل می‌شود:



در مرحله بعدی، درخت با مجموعه رئوس $\{1,6\}$ ، لبه $(6,5)$ را انتخاب می‌کند و دو درخت باقیمانده، لبه $(2,3)$ را انتخاب می‌کند و درخت پوشای نهایی حاصل می‌شود:



فرض کنیم گراف $G=(V,E)$ یک گراف بی‌جهت و M یک درخت فرآگیر مینیمم برای G باشد، مسیر M بین هر جفت رأس V_1 و V_2 لزوماً کوتاهترین مسیر نمی‌باشد.

کنکور ارشد

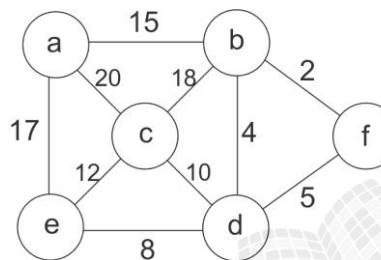
(علوم کامپیوتر - دولتی ۸۹)

- ۱- در نمایش یالی (Adjacency Multilist) یک گراف $G=(V,E)$ با $|E|=m, |V|=n$ و چه تعداد node وجود دارد (بدون در نظر گرفتن head node) (گزینه‌ها از چپ به راست)
- $|V|, |V|+|E|$ (۴) $|E|, |E|$ (۳) $|V|, |E|$ (۲) $|V|+|E|, |V|$ (۱)
- حل: جواب گزینه ۲ است.

در نمایش یالی گراف $G=(V,E)$ ، به تعداد گره‌ها یعنی $|V|$ ، لیست پیوندی و به تعداد دو برابر یال‌ها که از مرتبه $|E|$ می‌باشد، node وجود دارد.

(مهندسی IT - دولتی ۸۵)

- ۲- ترتیب انتخاب یال‌ها در الگوریتم Prim روی گراف زیر با شروع از راس a کدام مورد است؟



- (a,b),(b,f),(f,d),(d,e),(e,c) (۲) (b,f),(b,d),(e,d),(c,d),(a,b) (۱)
 (a,b),(b,f),(b,d),(d,e),(d,c) (۳)
 ۴) هیچکدام

حل: جواب گزینه ۳ است.

با شروع از گره a، در مرحله اول یال (a,b) با وزن 15 انتخاب می‌شود. در مرحله دوم یال (b,f) با وزن 2، در مرحله سوم یال (b,d) با وزن 4، در مرحله چهارم یال (e,d) با وزن 8 و در مرحله پنجم یال (d,c) با وزن 10 انتخاب خواهد شد.

(مهندسی کامپیوتر - آزاد ۸۶)

- ۳- یک گراف بسیار متصل شامل n گره مفروض است. می‌خواهیم با استفاده از الگوریتم Kruskal کوچکترین درخت پوشای این گراف را بدست آوریم. در این صورت پیچیدگی زمانی این الگوریتم برابر است با:

$$O\left(\frac{n^2}{2}\right) \quad (۴) \quad O(n^2) \quad (۳) \quad O(n \log n) \quad (۲) \quad O(n^2 \log n) \quad (۱)$$

حل: گزینه ۱ جواب است.

مرتبه اجرایی الگوریتم کراسکال برابر $\theta(e \log e)$ می‌باشد که در یک گراف کامل که تعداد یال‌ها برابر $\binom{n}{2}$ (از مرتبه n^2) می‌باشد، خواهیم داشت:

$$\theta(e \log e) = \theta(n^2 \log n^2) = \theta(2n^2 \log n) = \theta(n^2 \log n)$$

فصل ۱۰: مرتب سازی

الگوریتم‌های مرتب سازی را از نظر نحوه مرتب سازی داده‌ها می‌توان به صورت زیر دسته بندی کرد:

۱- مرتب سازی مقایسه‌ای

در مرتب سازی مقایسه‌ای، اطلاعات دیگری از داده‌های ورودی وجود ندارد و فقط با مقایسه بین کلیدهای عناصر، ترتیب نسبی آنها پیدا می‌شود.

۲- مرتب سازی غیر مقایسه‌ای (خطی)

در مرتب سازی غیر مقایسه‌ای، بدون مقایسه کلیدهای عناصر با هم، عمل مرتب سازی انجام می‌شود. این الگوریتم‌ها با استفاده از اطلاعاتی که از قبل در خصوص نوع کلیدها موجود است و در شرایط خاص، از روش‌های سریعی استفاده می‌کند که در آنها کلیدهای عناصر با هم مقایسه نمی‌شوند.

در این فصل مرتب سازی‌های مقایسه‌ای را بررسی می‌کنیم. از مرتب سازی‌های غیر مقایسه‌ای مانند مرتب سازی "مبنای"، "شمارشی و سطلی"، مرتب سازی مبنای را بررسی می‌کنیم و مرتب سازی شمارشی و سطلی در کتاب طراحی الگوریتم اینجانب بررسی شده است.

مرتب سازی

مرتبه اجرایی مرتب سازی‌های مقایسه‌ای در جدول زیر نشان داده شده است.

بدترین	میانگین	بهترین	نام روش
$o(n^2)$	$o(n^2)$	حبابی اصلاح شده	Bubble
$o(n^2)$	$o(n^2)$	$o(n^2)$	Selection
$o(n^2)$	$o(n^2)$	$o(n)$	Insertion
$o(n^2)$	$o(n \log n)$	$o(n \log n)$	Quick
$o(n \log n)$	$o(n \log n)$	$o(n \log n)$	Merge
$o(n \log n)$	$o(n \log n)$	$o(n \log n)$	Heap
$o(n^2)$	$o(n \log n)$	$o(n \log n)$	Tree

مرتب سازی حبابی (Bubble Sort)

در این روش با n بار حرکت در طول آرایه، یک عنصر با عنصر بعدی مقایسه شده و در صورت لزوم جا به جا می‌شوند. بنابراین در مرتبه اول طی کردن آرایه، بزرگترین (یا کوچکترین) عنصر در انتهای آرایه قرار می‌گیرد. در این مرتب سازی، در i امین مرتبه، عناصر i تا n به طور صحیح قرار می‌گیرند.

مثال

اعداد ۲، ۴، ۳، ۸ را به روش حبابی، صعودی نمایید.

حل: گذر اول به صورت زیر می‌باشد:

مقایسه ۴ با ۸ :

مقایسه ۳ با ۸ و تعویض آنها :

مقایسه ۸ با ۲ و تعویض آنها :

در پایان گذر اول، بزرگترین عنصر در خانه آخر قرار گرفته است. گذر دوم به صورت زیر است:

مقایسه ۴ با ۳ و تعویض آنها :

مقایسه ۴ با ۲ و تعویض آنها :

مقایسه ۳ با ۲ و تعویض آنها :

عناصر داده شده بعد از ۳ گذر با ۶ مقایسه که ۵ تا از آنها منجر به تعویض شد، مرتب شدند.

 تعداد مقایسه‌ها در روش مرتب سازی حبابی برابر $\frac{n(n-1)}{2}$ می‌باشد که در بدترین حالت به همین

تعداد، تعویض انجام می‌گیرد.

 بزرگترین عنصر در مرتب سازی حبابی صعودی بعد از $n-1$ مقایسه و حداقل $n-1$ تعویض به انتهای لیست می‌رود.

الگوریتم مرتب سازی حبابی

```
void Bubble-Sort (int a[ ], int n){  
    for( i=0 ; i < n-1 ; i++)  
        for ( j = n-1 ; j > i ; j--)  
            if (a[j] < a[j-1])  
                swap(a[j] , a[j-1]);  
}
```

 در روش حبابی اصلاح شده از یک flag استفاده می‌شود. این flag در ابتدای ورود به حلقه چک شده و در صورت مرتب شدن آرایه از ادامه تکرار حلقه جلوگیری می‌کند. مرتبه اجرایی مرتب سازی حبابی اصلاح شده در بهترین حالت برابر $O(n)$ می‌باشد.

مرتب سازی انتخابی (Selection Sort)

در مرتب سازی صعودی انتخابی، ابتدا کوچکترین عنصر آرایه را پیدا کرده و آن را در مکان اول لیست قرار می‌دهد. آنگاه کوچکترین عنصر دوم داخل لیست را پیدا کرده و آن را در مکان دوم لیست قرار می‌دهد و ...

مثال

تعداد مقایسه‌های لازم برای مرتب کردن لیست زیر به روش انتخابی را محاسبه نمایید.

10, 30, 17, 12, 1, 21, 15

حل: در ابتدا محل کوچک ترین عنصر یعنی ۱ را با عنصر اول تعویض کرده و سپس محل کوچکترین عنصر دوم یعنی ۱۰ با عنصر دوم تعویض می‌شود. به عبارتی در مرحله دوم کوچکترین عنصر آرایه ۲ تا n پیدا شده و محل آن با عنصر اول این آرایه تعویض می‌شود. عملیات به همین روال ادامه می‌یابد.

10	30	17	12	1	21	15
1	30	17	12	10	21	15
1	10	17	12	30	21	15
1	10	12	17	30	21	15
1	10	12	15	30	21	17
1	10	12	15	17	21	30
1	10	12	15	17	21	30

- لیست اولیه
- مرحله اول
- مرحله دوم
- مرحله سوم
- مرحله چهارم
- مرحله پنجم
- مرحله ششم

در مرحله اول برای پیدا کردن کوچکترین عنصر نیاز به شش مقایسه می‌باشد. در مرحله دوم برای پیدا کردن کوچکترین عنصر لیست جدید (از خانه ۲ تا انتهای) به پنج مقایسه نیاز است و به همین ترتیبی تعداد مقایسه‌ها را بدست آورده و بخواهیم داشت: $6+5+4+3+2+1=21$

در مرتب سازی انتخابی، اعداد بعد از $n-1$ مرحله مرتب می‌شوند.

تعداد مقایسه‌ها در مرتب سازی انتخابی برابر $\frac{n(n-1)}{2}$ می‌باشد.

تعداد جابجایی‌ها در مرتب سازی انتخابی در بدترین حالت برابر $\frac{n(n-1)}{2}$ می‌باشد. (بدترین حالت

وقتی رخ می‌دهد که بخواهیم آرایه صعودی را نزولی کنیم و یا بر عکس)

الگوریتم مرتب سازی انتخابی

```
void selectionsort (int s[ ], int n){
    int i , j , min;
    for ( i = 0; i < n-1; i++){
        min = i;
        for (j = i+1; j <= n ; j++)
            if (s[j] < s[min])
                min = j;
        swap(s[i] , s[min]);
    }
}
```

مرتب‌سازی درجی (Insertion Sort)

این الگوریتم بر اساس درج یک عنصر در محل صحیح کار می‌کند. از ابتدای آرایه در بخشی از آن که ابتدا به طول ۱ سپس ۲، ۳... n خواهد بود، آخرین عنصر این بخش و مقادیر قبل از آن تا جایی که این عنصر کوچکتر از آنهاست، به جلو جا به جا شده و این عنصر در محل مناسب درج می‌شود.

به عبارتی برای آرایه A با n عنصر:

۱- [A[1]] مرتب است.

۲- [A[2]] قبل یا بعد از [A[1]] درج می‌شود. طوری که [A[1]] و [A[2]] مرتب باشند.

۳- [A[3]] در مکان صحیح خود نسبت به [A[1]] و [A[2]] درج می‌شود، طوری که [A[1],A[2],A[3]] مرتب باشند.

۴- الی آخر

مثال

آرایه زیر را به روش درجی مرتب سازید.

20 , 5 , 35 , 8 , 2

حل:

20					
5	20				
5	20	35			
5	8	20	35		
2	5	8	20	35	

به عبارتی الگوریتم مرتب‌سازی درجی الگوریتمی است که مرتب‌سازی را با درج رکوردها در یک آرایه مرتب شده موجود مرتب سازی می‌کند.

الگوریتم مرتب‌سازی درجی

می‌خواهیم n کلید موجود در آرایه [s[1..n]] را به ترتیب نزولی مرتب کنیم.

```
void insertionsort (int s[ ] , int n){
    int i,j; int x;
    for (i=1 ; i<n ; i++){
        x = s[i];
        for( j = i-1 ; j>=0 && x < s[j] ; j--)
            s[j+1]=s[j];
        s[j+1]=x;
    }
}
```

تذکر: تنها فضایی که با n افزایش می‌یابد، اندازه آرایه ورودی S است. پس الگوریتم مرتب‌سازی درجی، یک مرتب‌سازی درجا است و فضای اضافی به $\Theta(n)$ تعلق دارد.

 در مرتب سازی درجی، پیچیدگی زمانی تعداد انتساب های رکوردها، در بدترین حالت برابر $\frac{(n-1)(n+4)}{2}$ و در

حالت میانگین برابر $\frac{n(n+7)}{4}$ می باشد.

 مرتب سازی درجی، برای یک آرایه مرتب، بهترین عملکرد و برای یک آرایه مرتب معکوس، بدترین عملکرد را دارد.

 مرتب سازی درجی برای $n=20$ ، سریع ترین روش مرتب سازی است.

 اگر n بزرگ باشد و رکوردها هم بزرگ باشند (زمان انتساب آنها چشمگیر باشد)، مرتب سازی انتخابی باید بهتر از مرتب سازی درجی عمل کند.

 در مرتب سازی درجی، یک آرایه n عنصری را می توان با **n-1** عمل درج مرتب کرد.

 اگر از جستجویی دودویی استفاده کنیم، تعداد مقایسه ها کاهش می باید. این روش را مرتب سازی درجی دودویی می نامند.

وارونگی

وارونگی در یک جایگشت، زوج (k_j, k_i) است به طوری که $j < i$ و $k_j > k_i$ باشد. به طور مثال، در جایگشت $[3,2,4,1,6,5]$ ، زوج $(3,2)$ یک وارونگی است، چون $2 < 3$. وارونگی های این جایگشت عبارتند از: $(3,1), (1, (2,1)), (4,1), (6,5)$.

مرتب سازی n کلید متمایز، به معنی حذف همه وارونگی ها در یک جایگشت است.

مرتب سازی درجی پس از هر بار مقایسه، یا کاری انجام نمی دهد یا کلید موجود در محل j را به محل $(j+1)$ ام منتقل می کند. با منتقل کردن کلید موجود در محل j به یک محل بالاتر، این واقعیت را تصحیح کرده ایم که x باید قبل از آن کلید بیاید. ولی، این تنها کاری است که انجام داده ایم. نشان می دهیم همه الگوریتم های مرتب سازی که فقط از طریق مقایسه کلیدها مرتب سازی را انجام می دهند و چنین مقدار محدودی از باز آرایی را پس از هر بار مقایسه انجام می دهند، حداقل به زمان درجه دوم نیاز دارند. فرض کنیم که کلیدها صرفاً اعداد مثبت و صحیح $n, \dots, 2, 1$ هستند، زیرا می توانیم کوچک ترین کلید را با 1 ، دومی را با 2 و ... جایگزین کنیم.

 یک جایگشت، وارونگی نخواهد داشت اگر و فقط اگر دارای ترتیب مرتب $[1,2,\dots,n]$ باشد.

مرتب سازی ادغام (Merge Sort)

در این روش آرایه n عنصری به n قسمت به طول یک تقسیم و سپس هر دو قسمت مجاور به صورت مرتب شده با هم ادغام می شوند. در این حالت آرایه به طول ۲ ایجاد شده است. روند ادغام تا رسیدن به یک آرایه به طول n ادامه می باید. قبل از توضیح عملکرد این مرتب سازی، نحوه ادغام دو آرایه مرتب را در یک آرایه مرتب جدید بیان می کنیم.

مثال

دو آرایه مرتب A و B را ادغام کرده و حاصل را در آرایه C قرار دهید:

A : 1, 5 , B : 2, 7, 9, 20

حل: برای اینکار از ۳ متغیر اشاره گر pa و pb و pc استفاده می‌شود که ۲ تای خاص از آنها با یکدیگر تغییر می‌کنند. در ابتدا pa به مقدار ۱ در آرایه A و pb به مقدار ۲ در آرایه B و pc به ابتدای آرایه C اشاره می‌کند. محتوای pa و pb با هم مقایسه شده و چون مقدار ۱ کوچکتر است در آرایه C نوشته شده و اشاره گر pa و pc یک واحد اضافه شده و مقدار ۵ با ۲ مقایسه شده و چون ۲ کوچکتر است در آرایه C نوشته شده و pb و pc یک واحد اضافه می‌شوند. حال مقدار ۵ با ۷ مقایسه شده و مقدار ۵ در آرایه C نوشته شده و در این حالت چون به انتهای آرایه A رسیده ایم بقیه عناصر آرایه B را به انتهای آرایه C اضافه می‌کنیم:

$$C : 1, 2, 5, 7, 9, 20$$



حداکثر تعداد مقایسه‌ها برای ادغام دو آرایه مرتب n , m عنصری برابر $n+m-1$ می‌باشد.

مثال

لیست داده شده در روش Merge Sort، بعد از چند گذر مرتب می‌شود؟

2, 3, 1, 7, 6, 5, 4, 0

حل:

$$\begin{aligned} & [2] [3] [1] [7] [6] [5] [4] [0] \\ & [2,3] \quad [1,7] \quad [5,6] \quad [0,4] \\ & [1,2,3,7] \quad [0,4,5,6] \\ & [0,1,2,3,4,5,6,7] \end{aligned}$$

مثال

لیست مقابل را به روش Merge Sort، مرتب نمایید.

25, 4, 70, 2, 63, 11, 57, 12, 45

حل:

$$\begin{aligned} & [25] [4] [70] [2] [63] [11] [57] [12] [45] \\ & [4 25] [2 70] [11 63] [12 57] [45] \\ & [2 4 25 70] [11 12 57 63] [45] \\ & [2 4 11 12 25 57 63 70] [45] \\ & [2 4 11 12 25 45 57 63 70] \end{aligned}$$



تعداد گذرهای مرتب سازی ادغام برابر $\lceil \lg n \rceil$ است.

الگوریتم مرتب سازی ادغامی بازگشتی

در نسخه بازگشتی، ساختار رکورد به صورت زیر تعریف می‌کنیم:

```
typedef struct{ int key; int link; }element;
```

تابع merge یک مقدار صحیح را بر می‌گرداند که به ابتدای لیست مرتب شده اشاره دارد.

```
int merge(element list[ ], int lower, int upper){
    int mid;
    if (lower >= upper) return lower;
    else{
        mid = (lower + upper) /2;
        return listmerge ( list , merge( list , lower , mid) , merge( list , mid+1 , upper));
    }
}
```

تابع listmerge دو زنجیر یا لیست مرتب شده یعنی first و second را دریافت کرده و مقدار صحیح که به ابتدای زنجیر مرتب شده جدید شامل زنجیرهای first و second اشاره دارد را برمی‌گرداند. بر خلاف نسخه تکراری ادغام، نسخه بازگشتی از لحاظ فیزیکی لیست را تغییر یا دوباره مرتب نمی‌کند.

```
int listmerge (element list[ ] , int first , int second){
    int start = n;
    while(first != -1 && second != -1)
        if ( list[first].key <= list[second].key )
    {
        list[start].link = first;
        start = first;
        first = list[first].link;
    }
    else{
        list[start].link = second;
        start = second;
        second = list[second].link;
    }
    if (first == -1)
        list[start].link = second;
    else
        list[start].link = first;
    return list[n].link;
}
```

فراخوانی تابع به صورت $start = merge(list, 0, n-1)$ است.

 در مرتب سازی ادغام ، مراحل کار به صورت زیر می باشد:

۱- تقسیم لیست n عنصری به دو زیر لیست $\frac{n}{2}$ عنصری.

۲- مرتب کردن بازگشتی دو زیر لیست به روش مرتب سازی ادغام در زمان $\frac{n}{2} \cdot 2T\left(\frac{n}{2}\right)$.

۳- ادغام دو زیر لیست مرتب شده در زمان $\theta(n)$

که رابطه بازگشتی آن به صورت زیر می باشد:

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + \theta(n) & n > 1 \\ \theta(1) & n = 1 \end{cases}$$

مرتبه اجرایی این رابطه $O(n \log n)$ می باشد.

 پیچیدگی زمانی تعداد مقایسه های کلیدها در مرتب سازی ادغامی در بدترین حالت، وقتی که n توانی از دو است به صورت $(n-1)\theta(n \lg n) = n \lg n - (n-1)$ تعلق دارد.

 الگوریتم های $\Theta(n \lg n)$ می توانند از عهده ورودی های بسیار بزرگ برآیند، حال آنکه الگوریتم های $\Theta(n^2)$ نمی توانند.

 گاهی مرتب سازی ادغامی پس از هر مقایسه بیش از یک وارونگی را حذف می کند. الگوریتم هایی که حداقل یک وارونگی پس از هر مقایسه انجام می دهند، حداقل $\frac{n(n-1)}{2}$ مقایسه را در هنگامی که ورودی به ترتیب عکس باشد، انجام می دهند.

 مرتب سازی ادغام، معمولاً روی عناصری که در فایل‌ها قرار دارند اجرا می‌شود.

مرتب سازی ادغام از حافظه جانبی استفاده می‌کند و نیاز به بافری به اندازه n دارد. (غیر درجا) در حالت معمول، استفاده از فضای اضافی به $\Theta(n)$ رکورد تعلق دارد.

مرتب سازی ادغامی (غیر بازگشتی)

```
void merge-sort(int a[ ] , int n){
    int len=1;
    int extra[ MAX];
    while(len < n) {
        merge-pass(a , extra , n , len);
        len=len *2;
        merge-pass(extra , a , n , len);
        len = len *2;
    }
}
```

در merge-sort ، در اولین گذر، لیست هایی با اندازه ۱ ، در دومین گذر لیست هایی با اندازه ۲ و در i امین گذر لیست هایی با اندازه 2^{i-1} ادغام خواهند شد.

```
void merge-pass(int a[ ] , int s [ ] , int n , int len){
    int i , j;
    for ( i=0 ; i <= n-2*len ; i = i+2*len )
        merge( a , s , i , i+len-1 , i+2*len-1 );
    if ( i+len < n)
        merge( a , s , i , i+len-1 , n-1 );
    else
        for ( j = i ; j < n; j++) s[j] = a[j];
}

void merge( int a[ ] , int s[ ] , int i , int m , int n){
    int j , k , t;
    j = m+1;
    k = i;
    while( i <= m && j <= n ) {
        if ( a[i] <= a[j] )
            s[k++] = a[i++];
        else
            s[k++] = a[j++];
    }
    if ( i > m)
        for( t = j ; t <= n ; t++)
            s[k+t-j] = a[t];
    else
        for( t = i ; t <= m ; t++)
            s[k+t-i] = a[t];
}
```

تابع merge ، لیستهای مرتب شده ((a[m+1],...,a[n]) و (a[i],...,a[m])) را در یک لیست مرتب شده یعنی (s[i],...,s[n]) ادغام می نماید.

مرتب‌سازی سریع (Quick Sort)

یک عنصر به عنوان محور انتخاب شده(معمولًا عنصر اول) و سایر عناصر در آرایه به صورتی جا به جا می‌شوند که کلیه عناصر کوچکتر از محور در یک طرف و کلیه عناصر بزرگ‌تر از محور در طرف دیگر قرار گیرند، سپس به همین روش دو آرایه واقع در طرفین محور به صورت بازگشتی مرتب می‌شوند.

الگوریتم مرتب سازی سریع

```
QUICKSORT(A , p , r){
    if ( p < r) {
        q = PARTITION(A , p , r);
        QUICKSORT (A , p , q-1);
        QUICKSORT (A , q+1, r);
    }
}
```

برای مرتب سازی کل آرایه A، فرآخوانی اولیه (QUICKSORT(A ,1 , length[A]) است.

```
PARTITION(A , p , r){
    x = A[r];
    i = p-1;
    for (j = p ; j<= r-1 ; j++)
        if (A[j] <= x) {
            i++;
            exchange( A[i] , A[j] );
        }
    exchange ( A[i+1] , A[r] );
    return i+1;
}
```

زمان اجرای PARTITION روی A[p..r] برابر $\theta(n)$ است که $n = r - p + 1$

رابطه بازگشتی مرتب سازی سریع $T(n) = 2T(\frac{n}{2}) + \theta(n)$ می باشد که مرتبه اجرای آن $O(n \log n)$ می باشد.

البته ممکن است در بدترین حالت (آرایه مرتب) آرایه به دو قسمت به طول صفر و $n-1$ تقسیم شده و در این حالت رابطه بازگشتی به صورت $T(n) = T(n-1) + \theta(n)$ است که مرتبه اجرای آن $O(n^2)$ می باشد.

مثال

مراحل اجرای تابع PARTITION بر روی یک آرایه نمونه:

عناصری که به رنگ طوسی می باشند در قسمت اول بوده و مقادیر آنها کوچکتر یا مساوی 4 است. عناصری که پر رنگ تر می باشند در قسمت دوم هستند و مقادیر آنها بزرگ تر از 4 است. عناصری که نه طوسی رنگ هستند و نه پر رنگ، هنوز در هیچ یک از دو قسمت قرار نگرفته اند.

(a) : هیچ یک از عناصر تقسیم بندی نشده اند.

(b) : مقدار 2 با خودش تعویض و در قسمت مقادیر کوچکتر قرار گرفته است.

(c) : مقدار 8 به قسمت مقادیر بزرگتر اضافه شده است.

(d) : مقدار 7 به قسمت مقادیر بزرگتر اضافه شده است.

(e) : مقادیر 1 و 8 تعویض شده اند و قسمت کوچکتر افزایش می یابد.

(f) : مقادیر 3 و 8 عوض شده و قسمت کوچکتر افزایش می یابد.

(g) : مقدار 5 به قسمت بزرگتر اضافه شده است.

(h) : مقدار 6 به قسمت بزرگتر اضافه شده است و حلقه پایان می یابد.

(i) : عنصر محوری عوض شده تا بین دو قسمت قرار بگیرد.

(a)	2	8	7	1	3	5	6	4
-----	---	---	---	---	---	---	---	---

(b)	2	8	7	1	3	5	6	4
-----	---	---	---	---	---	---	---	---

(c)	2	8	7	1	3	5	6	4
-----	---	---	---	---	---	---	---	---

(d)	2	8	7	1	3	5	6	4
-----	---	---	---	---	---	---	---	---

(e)	2	1	7	8	3	5	6	4
-----	---	---	---	---	---	---	---	---

(f)	2	1	3	8	7	5	6	4
-----	---	---	---	---	---	---	---	---

(g)	2	1	3	8	7	5	6	4
-----	---	---	---	---	---	---	---	---

(h)	2	1	3	8	7	5	6	4
-----	---	---	---	---	---	---	---	---

(i)	2	1	3	4	7	5	6	8
-----	---	---	---	---	---	---	---	---

الگوریتم Quick sort برای آرایه مرتب ، بدترین عملکرد را دارد و تعداد مقایسه های در این حالت برابر $\frac{n(n-1)}{2}$ می باشد.

مزیت مرتب سازی سریع نسبت به ادغامی این است که نیازی به آرایه اضافی ندارد. ولی هنوز هم یک مرتب سازی در جایست، زیرا در حالی که الگوریتم، زیر آرایه نخست را مرتب سازی می کند، اندیس اول و آخر زیر آرایه دیگر باید در پشته رکوردهای فعالیت نگهداری شوند.

در مرتب سازی سریع برخلاف ادغامی، تضمینی وجود ندارد که آرایه همواره از وسط تقسیم شود. در بدترین حالت، ممکن است partition مکرراً آرایه را به زیر آرایه تهی در سمت چپ (یا راست) و زیرآرایه ای با یک عنصر کمتر در سمت راست (یا چپ) تقسیم کند. به این ترتیب، $n-1$ جفت اندیس در پشته قرار می گیرند، بنابراین استفاده از فضای اضافی در بدترین حالت به $\Theta(n^2)$ تعلق دارد. می توان مرتب سازی سریع را چنان اصلاح کرد که استفاده از فضای اضافی حداقل در حدود $\lg n$ باشد.

در Randomized-Quicksort ، محور به صورت تصادفی و با احتمال یکسان، یکی از عناصر انتخاب می شود و بقیه می باشد. الگوریتم مانند Quick Sort عادی است.

اگر نمودار میانگین زمانی را رسم کنید، مشاهده می شود که به ازای $n \leq 20$ ، مرتب سازی درجی سریع ترین است. برای مقادیر بین 20 تا 45 ، مرتب سازی سریع بهترین و سریعترین می باشد. برای مقادیر بزرگتر، مرتب سازی ادغام سریع ترین است.

بنابراین، بهتر است در مرتب سازی ادغامی برای زیرلیست‌های کمتر از 45 از مرتب سازی سریع استفاده شود. همچنین در مرتب سازی سریع، زمانی که طول زیر لیست‌ها کمتر از 20 باشند، از مرتب سازی درجی استفاده کند.

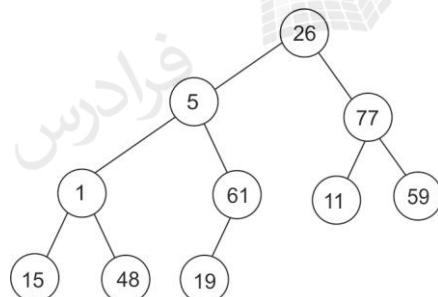
مرتب سازی هیپ (Heap Sort)

در مرتب سازی heap ساختار maxheap مورد استفاده قرار می‌گیرد. یک درخت heap با n رکورد با استفاده از تابع adjust ایجاد می‌شود. سپس رکوردها یکی یکی از heap خارج می‌شوند.

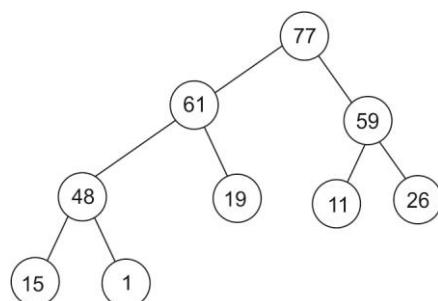
تابع adjust از یک درخت دودویی که زیردرختهای چپ و راست آن خاصیت heap را برآورده می‌کنند، استفاده می‌کند. (به غیر از ریشه) همچنین این تابع درخت را بگونه‌ای تنظیم می‌کند که کل درخت دودویی خاصیت heap را برآورده می‌کند. برای مرتب سازی یک لیست، n-1 گذر بر روی لیست اعمال می‌گردد. در هر گذر، اولین رکورد در heap با آخرین رکورد تعویض می‌شود. چون اولین رکورد، همیشه شامل بزرگترین کلید است، این رکورد با بزرگترین کلید را در محل n ام قرار می‌دهیم. در گذر دوم، رکورد با دومین کلید بزرگ را در موقعیت n-1 قرار می‌دهیم و در نهایت، در n امین گذر، رکورد با n امین کلید بزرگ را در موقعیت n-i+1 قرار می‌دهیم.

مثال

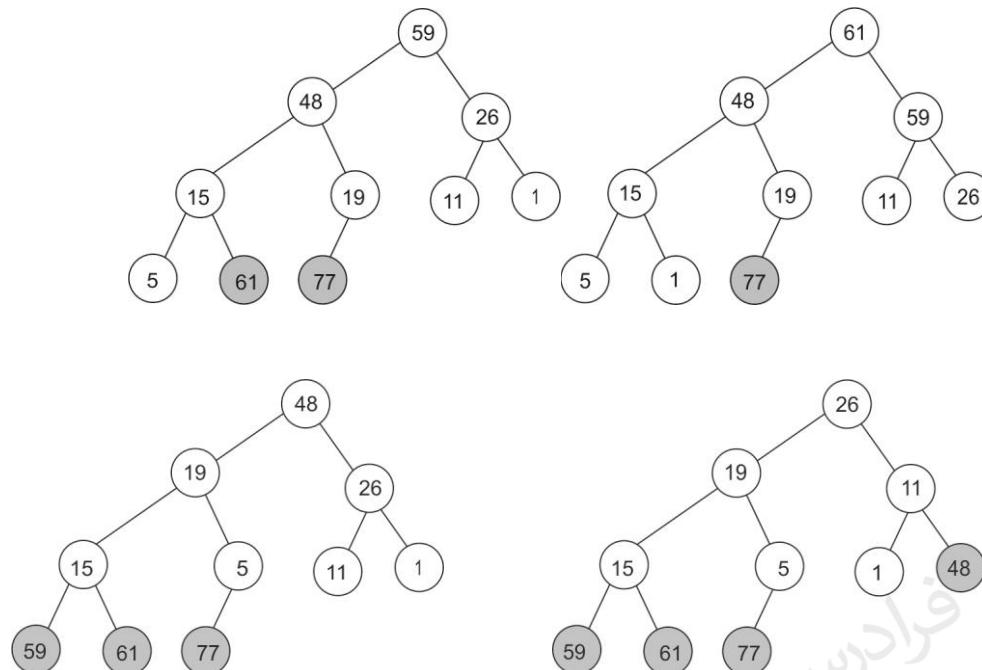
لیست ورودی (19 , 19 , 26 , 5 , 77 , 1 , 61 , 11 , 59 , 15 , 48) را در نظر بگیرید. شکل زیر درخت دودویی اولیه را نشان می‌دهد:



بعد از تبدیل آن به maxheap داریم:



فرایند مرتب سازی در زیر نشان داده شده است. دایره‌های پر رنگ رکوردهایی که در موقعیت مرتب شده خود قرار گرفته اند را نشان می‌دهد.



تابع مرتب سازی Heap

```

void heapsort( int a[ ] , int n ){
    int i , j;
    for ( i = n/2 ; i>0 ; i--)
        adjust( a , i , n);
    for( i = n-1 ; i>0 ; i--)
    {
        swap( a[1] , a[i+1] , temp );
        adjust( a , 1, i );
    }
}
void adjust( int a[ ] , int root , int n )
{
    int child , rootkey, temp;
    temp = a[root];
    rootkey = a[root];
    child = 2 * root;
    while(child <= n)
    {
        if (( child <n) && (a[child] < a[child+1]) )
            child++;
        if ( rootkey > a[child] ) break;
        else{
            a[child / 2] = a[child];
            child = child *2;
        }
    }
    a[child / 2] = temp;
}

```

در حلقه for اول در تابع `heapsort` ، تابع `adjust` یکبار به ازای همه گره ها که دارای یک فرزند هستند، فراخوانی می شود. زمان این حلقه بیشتر از $O(n)$ نمی باشد. در حلقه for دوم، تابع `adjust` به تعداد $(n-1)$ مرتبه با حداقل عمق $\lceil \log(n+1) \rceil$ فراخوانی می شود. بنابراین زمان آن $O(n \log n)$ می باشد. در نتیجه کل زمان محاسباتی $O(n \log n)$ خواهد شد.

در صورتی که بخواهیم k آرایه مرتب صعودی را ادغام کرده و در یک آرایه مرتب ذخیره کنیم، با فرض اینکه کل خانه های آرایه ها برابر n باشد، نیاز به $n(k-1)$ مقایسه می باشد و زمان ادغام برابر $O(n.k)$ است. در این روش عنصر اول آرایه ها با یکدیگر مقایسه شده و با $(k-1)$ مقایسه کوچکترین عنصر خارج شده و آرایه ای که عنصر مینیمم از آن خارج شده را به سمت بالا حرکت می دهیم(رانش) و عملیات را ادامه می دهیم. برای ادغام بهتر است از MinHeap استفاده کرد که به زمان $O(n.\log k)$ نیاز است.

مرتب سازی درختی (Tree Sort)

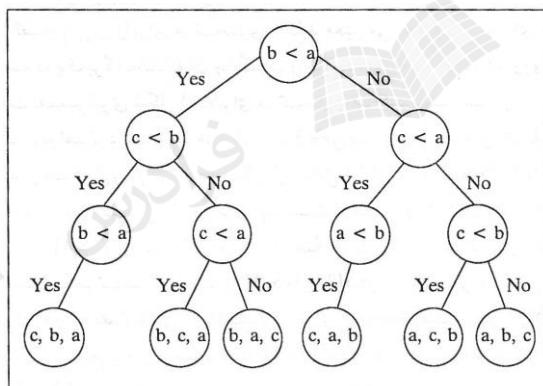
ابتدا با اعداد داده شده یک BST ساخته و سپس با پیمایش میانوندی درخت حاصل، رشته صعودی حاصل می‌شود. این موضوع در مبحث BST بررسی شد.

الگوریتم های پایدار (stable)

الگوریتم‌های مرتب سازی که ترتیب اولیه رکوردهای هم کلید را حفظ می‌کنند، پایدار نام دارند. در این الگوریتم‌ها اعداد با مقدار یکسان در آرایه خروجی به همان ترتیبی که در آرایه ورودی هستند ظاهر می‌شوند. به عبارتی یک الگوریتم مرتب سازی را پایدار گوئیم هر گاه در انتهای روش، عناصر یکسان، به همان ترتیب اولیه در آرایه مرتب نشده، ظاهر بشوند. این ویژگی وقتی مهم است که عناصر مرتب شده همراه با داده‌های وابسته باشند. فرض کنید می‌خواهیم چند نفر را بر اساس سن با یک الگوریتم پایدار مرتب کنیم. اگر دو نفر با نام‌های الف و ب همسن باشند و در لیست اولیه الف جلوتر از ب آمده باشد، در لیست مرتب شده هم الف جلوتر از ب آمده باشد. الگوریتم‌های پایدار عبارتند از: حبابی، درجی، مبنای، ادغام، درختی و شمارشی.

درخت تصمیم‌گیری برای الگوریتم‌های مرتب سازی

منتظر با هر الگوریتم قطعی برای مرتب سازی n کلید، حداقل یک درخت تصمیم‌گیری معتبر وجود دارد. در شکل زیر درخت تصمیم‌گیری منتظر با مرتب سازی تعویضی، در هنگام مرتب سازی سه کلید رسم شده است:



در درخت بالا، گره سطح ۲ حاوی مقایسه " $b < a$ " هیچ فرزند راستی ندارد. چون پاسخ No به آن مقایسه، پاسخ‌هایی را نقض می‌کند که روی مسیر منتهای به آن گره به دست می‌آید.

چند نکته:

- ۱- تعداد مقایسه های انجام شده توسط یک درخت تصمیم گیری در بدترین حالت برابر با عمق درخت است.
- ۲- هر الگوریتم قطعی که n کلید متمایز را فقط با مقایسه کلیدها مرتب سازی می کند، باید در بدترین حالت حداقل $\lceil \lg(n!) \rceil$ مقایسه کلیدها را انجام دهد.
- ۳- درخت تصمیم گیری که فقط از طریق مقایسه کردن عمل مرتب سازی را انجام می دهد، دارای $n!$ برگ است.

مرتب سازی مبنا (Radix Sort)

در مرتب سازی مبنا (توزیعی)، با فرض اینکه اعداد در مبنای 10 باشند، در گذر اول اعداد بر حسب اولین رقم سمت راست مرتب می شوند و در گذر دوم بر طبق دومین رقم از سمت راستشان و به همین روال مرتب سازی ادامه می یابد. (این نوع مرتب سازی مقایسه ای نمی باشد)

مثال

اعداد زیر را به روش مرتب سازی مبنایی، مرتب نمایید.

239 , 234 , 879 , 878 , 123 , 358 , 416 , 317 , 137 , 225

حل: چون مبنا 10 است، ده گروه از 0 تا 9 در نظر می گیریم. ارقام را از راست به چپ بازرسی کرده و هر کلید را در گروه متناظر با رقمی که در حال حاضر بازرسی می شود، قرار می دهیم. بعد از پایان گذر اول، اعداد بر حسب اولین رقم سمت راست مرتب شده اند. بعد از پایان گذر دوم، اعداد بر حسب دومین رقم سمت راست مرتب شده اند و این روال در شکل زیر نشان داده شده است.

اعدادی که باید مرتب شوند.	<table border="1"> <tr><td>239</td><td>234</td><td>879</td><td>878</td><td>123</td><td>358</td><td>416</td><td>317</td><td>137</td><td>225</td></tr> </table>	239	234	879	878	123	358	416	317	137	225				
239	234	879	878	123	358	416	317	137	225						
اعداد با سمت راست ترین رقم توزیع شده اند.	<table border="1"> <tr><td>0</td><td>1</td><td>2</td><td>3</td><td>123</td><td>234</td><td>225</td><td>416</td><td>317</td><td>137</td><td>878</td><td>358</td><td>239</td><td>879</td></tr> </table>	0	1	2	3	123	234	225	416	317	137	878	358	239	879
0	1	2	3	123	234	225	416	317	137	878	358	239	879		
اعداد با دومین رقم از سمت راست توزیع شده اند.	<table border="1"> <tr><td>0</td><td>416</td><td>317</td><td>123</td><td>225</td><td>234</td><td>137</td><td>239</td><td>358</td><td>878</td><td>879</td><td>8</td><td>9</td></tr> </table>	0	416	317	123	225	234	137	239	358	878	879	8	9	
0	416	317	123	225	234	137	239	358	878	879	8	9			
اعداد با سومین رقم از سمت راست توزیع شده اند.	<table border="1"> <tr><td>0</td><td>123</td><td>137</td><td>225</td><td>234</td><td>239</td><td>317</td><td>358</td><td>416</td><td>878</td><td>879</td><td>8</td><td>9</td></tr> </table>	0	123	137	225	234	239	317	358	416	878	879	8	9	
0	123	137	225	234	239	317	358	416	878	879	8	9			

با هر بار گذر، اگر دو کلید باید در یک گروه قرار گیرند، کلیدی که از گروه واقع در منتها الیه سمت چپ در گذر قبلی می آید، در طرف چپ کلید دیگر قرار می گیرد. به عنوان مثال بعد از اولین گذر، کلید 416 در گروه واقع در چپ کلید 317 قرار دارد. بنابراین، هنگامی که در دومین گذر، هر دو آن ها در گروه اول قرار بگیرند، کلید 416 در طرف چپ کلید 317 قرار می گیرد. ولی در گذر سوم، کلید 416 به طرف راست کلید 317 می رود، چون در گروه چهارم قرار داده می شود، سپس کلید 317 در گروه سوم قرار داده می شود.

 مرتب سازی مبنا n عدد d رقمی که در آنها هر رقم می تواند تا k مقدار ممکن را بگیرد در زمان $\Theta(d(k+n))$ مرتب می کند.

کنکور ارشد

(مهندسی کامپیوتر - آزاد ۸۷)

۱- در الگوریتم مرتب سازی حبابی (یا تبادلی) فرض می کنیم $T(n)$ تعداد دستورالعمل مقایسه باشد. در این صورت (n) کدام است؟

$$T(n) = 2n - 1 \quad (۲)$$

$$T(n) = \frac{n^2}{2} - 1 \quad (۴)$$

$$T(n) = \frac{n(n+1)}{2} \quad (۱)$$

$$T(n) = \frac{n(n-1)}{2} \quad (۳)$$

حل: جواب گزینه ۳ است.

تعداد مقایسه‌ها در روش مرتب سازی حبابی برابر $\frac{n(n-1)}{2}$ می‌باشد.

(علوم کامپیوتر - دولتی ۸۹)

۲- اگر برای مرتب سازی آرایه‌ی زیر به صورت صعودی از الگوریتم Bubble Sort استفاده شود، چند عمل مقایسه و چند عمل جابه‌جایی صورت می‌گیرد؟ {3, 5, 2, 1, 4, 7}

(۱) ۳ عمل جابه‌جایی و 30 عمل مقایسه صورت می‌گیرد.

(۲) ۳ عمل جابه‌جایی و 15 عمل مقایسه صورت می‌گیرد.

(۳) 6 عمل جابه‌جایی و 14 عمل مقایسه صورت می‌گیرد.

(۴) 30 عمل جابه‌جایی و 30 عمل مقایسه صورت می‌گیرد.

حل: جواب گزینه ۳ است.

تعداد مقایسه‌ها در روش مرتب سازی حبابی برابر $\frac{n(n-1)}{2}$ می‌باشد که در بدترین حالت به همین تعداد،

تعویض انجام می‌گیرد. بنابراین با قرار دادن 6 به جای n در این رابطه، عدد 15 حاصل می‌شود. بنابراین گزینه‌های ۱ و ۴ که اعداد بزرگتر از 15 دارند، حتماً نادرست می‌باشند. این آرایه در ۳ گذر مرتب می‌شود.

گذر اول:

آرایه اولیه : 3, 5, 2, 1, 4, 7

3, 2, 5, 1, 4, 7

3, 2, 1, 5, 4, 7

3, 2, 1, 4, 5, 7

گذر دوم:

2, 3, 1, 4, 5, 7

2, 1, 3, 4, 5, 7

گذر سوم:

1, 2, 3, 4, 5, 7

مشخص است که 6 جابه‌جایی لازم است. بنابراین گزینه ۳ درست است.

(مهندسی کامپیوتر - دولتی ۸۹)

۳- می‌دانیم که هزینه‌ی الگوریتم مرتب سازی درجی برای مرتب سازی آرایه‌ی A با n عنصر متناسب با تعداد وارونگی (inversion)‌های عناصر آن آرایه است. زوج (i, j) را یک عدد وارونگی می‌گوییم اگر $j < i$ و $A[i] > A[j]$. با فرض احتمال این که یک زوج اندیس دلخواه از A یک وارونگی باشد برابر $\frac{1}{2}$ است، میانگین تعداد وارونگی‌های یک آرایه‌ی A با عناصر متمایز چقدر است؟

$$\frac{n^2 - n}{4} \quad (4)$$

$$\frac{n^2}{4} \quad (3)$$

$$\frac{n^2}{2} \quad (2)$$

$$\frac{n^2 - n}{2} \quad (1)$$

حل: جواب گزینه ۴ است.

تعداد حالت‌هایی که می‌توان دو عنصر را از بین n عنصر انتخاب کرد برابر است با:

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)(n-2)!}{2(n-2)!} = \frac{n(n-1)}{2}$$

بنابراین میانگین تعداد وارونگی‌ها با احتمال $\frac{1}{2}$ برابر است با: $\frac{1}{2} \times \frac{n(n-1)}{2} = \frac{n^2 - n}{4}$

(مهندسی کامپیوتر - آزاد ۸۶)

۴- اگر در مرتب سازی سریع، زیر لیست‌های کوچکتر در ابتدا مرتب شوند، پشته بازگشته دارای چه عمقی خواهد بود؟

$$o(1) \quad (4)$$

$$o(\log n) \quad (3)$$

$$o(n^2) \quad (2)$$

$$o(n) \quad (1)$$

حل: جواب گزینه ۱ است.

اگر در روش Quick Sort، داده‌ها هر بار به طور مساوی تقسیم شوند، حداقل عمق درخت بازگشته $\log n$ بوده و به فضای پشته $o(\log n)$ نیاز خواهد بود. اما در بدترین حالت یعنی وقتی داده‌ها به قسمتهای به طول $1-n$ و صفر تقسیم شده باشد، فضای پشته $o(n)$ خواهد بود.

(مهندسی کامپیوتر - آزاد ۸۹)

۵- الگوریتمی را پایدار گوییم هر گاه ترتیب عناصر مساوی بعد از مرتب سازی عوض نشود. کدام یک از الگوریتم‌های زیر پایدار هستند؟

$$1) \text{ سریع و درجی} \quad 2) \text{ ادغامی و درجی} \quad 3) \text{ سریع و ادغامی} \quad 4) \text{ هرمی و درجی}$$

حل: جواب گزینه ۲ است.

مرتب سازی سریع و هرمی (Heap) پایدار نمی‌باشند، بنابراین گزینه‌های ۱ و ۳ و ۴ نادرست می‌باشند.

(علوم کامپیوتر - دولتی ۸۹)

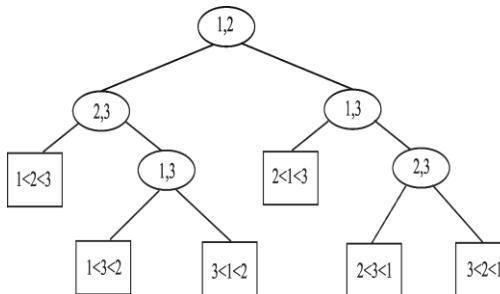
۶- یک الگوریتم مرتب سازی بر مبنای درخت تصمیم‌گیری، حداقل دارای چه تعداد مقایسه خواهد بود؟ فرض کنید $(a = \log_2^{n!}, b = n \log n, c = \log n = d = n)$

b,d (۴)	a,c (۳)	a,b (۲)	a (۱)
---------	---------	---------	-------

حل: جواب گزینه ۲ است. یک درخت تصمیم‌گیری برای مرتب سازی n عنصر، دارای ارتفاع $\log n!$ می‌باشد و از آنجا که $\log n!$ از نظر پیچیدگی با $n \log n$ برابر است، جواب گزینه ۲ می‌باشد. ($n!$ تعداد برگ‌های این درخت است.)

(مهندسی IT - دولتی) ۸۴

۷- یک درخت دودویی تصمیم‌گیری را در نظر بگیرید که از آن برای مرتب کردن عناصر یک مجموعه براساس عمل مقایسه، استفاده می‌شود. برای مثال درخت روبرو، برای مرتب کردن یک فایل با سه عضو، استفاده می‌شود. کدام یک از گزاره‌های ذیل، در مورد این نوع درخت‌ها درست است؟



۱) عمق این نوع درخت‌ها، حداقل $\log_2(n!)$ است و در گروه $O(n \cdot \log n)$ قرار می‌گیرد.

۲) عمق این نوع درخت‌ها، حداکثر $\log_2(n!)$ است و در گروه $O(n \cdot \log n)$ قرار می‌گیرد.

۳) عمق این نوع درخت‌ها، حداکثر $\log_2(n!)$ است و حداقل $O(n \cdot \log n)$ مقایسه در آنها انجام می‌شود.

۴) عمق این نوع درخت‌ها، حداقل $\log_2(n!)$ است و حداکثر $O(n \cdot \log n)$ مقایسه در آنها انجام می‌شود.

حل: جواب گزینه ۱ است.

عمق درخت تصمیم‌گیری، حداقل برابر $\log_2(n!)$ می‌باشد. همچنین می‌دانیم که به جای $n!$ می‌توان n^n نوشت و به

همین علت $\log_2(n!)$ در گروه $O(n \cdot \log n)$ قرار می‌گیرد.

فصل ۱۱: درهم سازی

در کاربردهایی لازم است تا اعمال حستجو، درج و حذف بر روی مجموعه پویایی از داده‌ها انجام شود. هزینه اعمال "جستجو، درج و حذف" بر روی مجموعه ای n عنصری در ساختمان داده‌های لیست یا BST در بدترین حالت $O(n)$ و بسیار کند است. البته با بهبود BST این اعمال در بدترین حالت در $O(\lg n)$ انجام می‌شود. با استفاده از ساختمان داده جدول درهم سازی (hashing table) و روش درهم سازی (hashing)، هزینه هر کدام از این اعمال، در بدترین حالت و نیز در حالت میانگین می‌تواند $O(1)$ شود.

جدول آدرس دهی مستقیم

در آدرس دهی مستقیم (direct address)，هر عنصر را بر حسب مقدار کلیدش مستقیماً در یک آرایه T ذخیره می‌کنیم. برای نمایش مجموعه پویایی از عناصر از جدول آدرس دهی مستقیم ($T[0..m-1]$) استفاده می‌کنیم. در این آرایه داریم:

$$K \subseteq \{0,1,\dots,m-1\}$$

$$T(k) = \begin{cases} x & k \in K, \text{key}[x] = k \\ \text{null} & \text{otherwise} \end{cases}$$

در آدرس دهی مستقیم، اعمال "درج، حذف و جستجو" در $O(1)$ انجام می‌شود.

جدول های درهم سازی

مشکل آدرس دهی مستقیم این است که m می‌تواند بسیار بزرگ باشد و ساخت یک جدول با اندازه بزرگ ناممکن است. برای حل این مشکل از جدول درهم سازی استفاده می‌کنیم. در این روش جدول را به اندازه قابل پیاده سازی m در نظر می‌گیریم و عنصر x با کلید k را در درایه $h(k)$ ذخیره می‌کنیم. که h یک تابع درهم سازی (hash function) است. مقدار $h(k)$ به ازای یک کلید k ، یک عدد صحیح بین ۰ و $m-1$ است.

برخورد (collision)

در جدول درهم سازی، ممکن است بیش از یک عنصر به یک درایه خاص نگاشته شود. به این مشکل برخورد (collision) می‌گویند. به عبارتی برخورد وقتی رخ می‌دهد که پس از اعمال تابع درهم سازی، به ازای دو کلید متفاوت، آدرس یکسانی تولید شود. نمی‌توان تابع درهم سازی انتخاب کرد که اصلاً برخوردی نداشته باشد. بنابراین باید این مشکل را حل کرد. می‌توان به کمک روش‌های زنجیره‌ای یا آدرس دهی باز مشکل برخورد را حل کرد.

توابع درهم سازی

یک تابع درهم ساز یک کلید را به عنوان ورودی گرفته و یک آدرس را به عنوان خروجی بر می‌گرداند. یک تابع درهم سازی خوب، تابعی است که ساده و یکنوا باشد. یعنی هر عنصر با احتمال مساوی در هر یک از درایه‌های جدول قرار می‌گیرد.

یکی از توابع درهم ساز، تابع تقسیم نام دارد. در این روش، تابع درهم سازی برابر باقیمانده‌ی تقسیم یک کلید k بر m در نظر گرفته می‌شود. یعنی، $m \text{ mod } k = h(k) = k \mod m$. که m نباید توانی از دو باشد. یک عدد اول که نزدیک به توانی از دو نباشد، انتخاب مناسبی برای m است.

مثال

اگر در حدود 2000 کلید رشته‌ای داشته باشیم و برای پیدا کردن کلید مورد نظر، به طور میانگین 3 عنصر را بگردیم، بهتر است اندازه جدول را چند بگیریم؟
پاسخ:

بهتر است اندازه جدول را 701 بگیریم، چون یک عدد اول نزدیک به $3/2000$ است. در این صورت، یک رشته را به یک عدد k تبدیل و از تابع درهم سازی $h(k) = k \mod 701$ استفاده می‌کنیم.

مثال

با فرض $i^2 \mod 7$ برای $i = 1..15$ و $h(i) = i^2 \mod 7$ ، تعداد برخوردها را مشخص کنید.
پاسخ:

به عبارتی کلیدها $1^2, 2^2, \dots, 15^2$ و تابع درهم ساز $h(x) = x \mod 7$ است. برای ساده شدن کار می‌توان برای محاسبه $k^2 \mod 7$ ، عبارت $(k \mod 7)^2 \mod 7$ را محاسبه کرد. چون طبق رابطه مقابله این دو با هم معادل اند:
 $ab \mod m = (a \mod m)(b \mod m) \mod m$

بنابراین داریم:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$i \mod 7$	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1
$(i \mod 7)^2 \mod 7$	1	4	2	2	4	1	0	1	4	2	2	4	1	0	1

در نتیجه تعداد برخوردها در درایه‌های مختلف به صورت زیر است:

0	1	2	3	4	5	6
2	5	4	0	4	0	0

اگر در این مثال، کلیدها $1^2, 2^2, \dots, 15^2$ بودند، تعداد برخوردها در هر درایه چگونه بود؟
پاسخ:

خروجی تابع درهم ساز به طور متناوب اعداد 1,0,2,2,4,1,0 می‌باشد. تعداد تکرار هر یک از اعداد 0 تا 6 در این دنباله برابر است با:

0	1	2	3	4	5	6
1	2	2	0	2	0	0

بنابراین اگر این جدول را تا $98^2 = 9604$ ادامه دهیم، 14 مرتبه (98/7 = 14) این دنباله هفت عنصری تکرار می‌شود. در نتیجه تعداد تکرار هر یک از اعداد 0 تا 6، برابر است با:

0	1	2	3	4	5	6
14	28	28	0	28	0	0

ضریب بارگذاری (load factor)

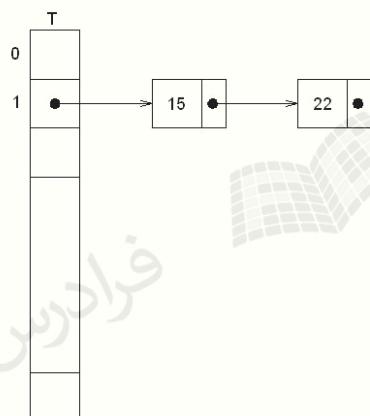
فرض می کنیم که جدول درهم سازی T به اندازه m ، در خود n عنصر را ذخیره می کند. به نسبت $\frac{n}{m}$ ، ضریب بارگذاری می گوییم و با α نشان می دهیم.

روش زنجیره ای برای حل برخورد

در روش زنجیره ای (chaining)، رکوردهای تصادفی به یکدیگر زنجیر می شوند. به عبارتی، عناصری را که به وسیله‌ی تابع درهم سازی به یک درایه از جدول درهم سازی، نگاشته می شوند به صورت لیست خطی یک طرفه در آورده و آدرس شروع آن را در همان درایه، یعنی $T[i]$ قرار می دهیم. در این روش، عناصر با کلیدهای مختلف k که مقدار $h(k)$ آنها برابر است در لیست پیوندی $[T[h(k)]$ قرار می گیرند. درایه های خالی جدول، لیست های تهی هستند.

مثال

اگر تابع درهم سازی h باشد، آنگاه $h(15)=h(22)=1$ است.



عملیات درج، حذف و جستجو در روش زنجیره ای

Insert(T, x)

Insert x at the head of list $T[h(\text{key}[x])]$

Deleted(T, x)

Delete x from the list $T[h(\text{key}[x])]$

Search(T, k)

Search for an element with key k in list $T[h(k)]$

نکاتی در روش زنجیره ای

بدترین حالت زمان اجرا برای درج برابر $O(1)$ است.

اگر تابع درهم سازی به صورت یکنوا عناصر را در جدول درهم سازی توزیع کند، هزینه میانگین جستجو $O(1)$ می باشد.

 بدترین حالت زمان اجرا برای جستجو و حذف متناسب با طول لیست است. چون اول باید با انجام یک جستجو، عنصر مورد نظر را پیدا کرد.

 اگر لیست پیوندی دو طرفه باشد، حذف عنصر می تواند در زمان $O(1)$ انجام شود.

 یک جدول درهم سازی با a درایه را در نظر بگیرد که تصادم در آن به صورت زنجیره ای حل می شود. اگر b عنصر در

جدول باشد، میانگین تعداد عناصر در هر درایه از جدول $\frac{b}{a}$ است.

(تابع درهم ساز با احتمال برابر هر عنصر را به یک درایه از جدول نگاشت می کند).

 می خواهیم n عنصر با کلیدهای مجزا را با استفاده از یک تابع درهم ساز ساده و یکنوا در یک آرایه با اندازه m درج

کنیم. میانگین تعداد برخوردهای دو عنصر برابر $\frac{n(n+1)}{2m}$ است. (از مرتبه $\Theta(\frac{n^2}{m})$ است).

فرض ۱ : با این تابع، احتمال اینکه دو عنصر دلخواه و متفاوت به یک درایه نگاشت شوند برابر است.

فرض ۲ : برخوردها را به روش زنجیره ای حل می کنیم.

 در یک جدول درهم سازی با روش زنجیره ای، با فرض اینکه درهم سازی یکنواخت ساده است، تعداد عناصری که

انتظار می رود بررسی شوند، در یک جستجوی موفق برابر $\frac{\alpha}{2} + \frac{\alpha}{2n}$ و در یک جستجوی ناموفق، برابر α می باشد.

بنابراین میانگین زمان جستجوی موفق با احتساب زمان لازم برای محاسبه تابع درهم سازی برابر است با:

$$\theta\left(2 + \frac{\alpha}{2} - \frac{\alpha}{2n}\right) = \theta(1 + \alpha)$$

آدرس دهی باز

در آدرس دهی باز (open hashing)، از یک آرایه به اندازه m به عنوان جدول درهم سازی استفاده می شود که حداقل m عنصر در آن جای می گیرد و در هر درایه، فقط یک عنصر ذخیره می شود. بنابراین حداقل تعداد عناصر را باید از قبل بدانیم یا از جدول درهم سازی پویا (dynamic hash table) استفاده کنیم. جدول درهم سازی پویا، جدولی است که اندازه آن بر حسب نیاز، کم یا زیاد می شود.(معمولًا نصف یا دو برابر)

با شروع از محل تصادف، جستجوی خطی به سمت انتهای فایل شروع شده و رکورد تصادفی در اولین محل جادار درج می شود. وارسی به روش حلقوی انجام می گیرد.

مثال

جدول درهم سازی $H[0..4]$ مفروض است. اگر عناصر 7 و 15 و 6 و 10 و 11 (از چپ به راست) را وارد جدول کنیم، جدول چگونه خواهد شد؟ (در صورت استفاده از روش آدرس دهی باز با وارسی خطی)(تابع Hash به صورت باقیمانده تقسیم بر 5 تعریف شده است).

پاسخ: خروجی تابع درهم ساز برای هر یک از عناصر برابر است با:

$$11 \% 5 = 1$$

$$10 \% 5 = 0$$

$$6 \% 5 = 1$$

$$15 \% 5 = 0$$

$$7 \% 5 = 2$$

بنابراین 11 در خانه 1 و 10 در خانه 0 درج می‌شود. اما 6 که باید در خانه 1 درج شود، چون این خانه پر است، در اولین خانه خالی بعد از آن یعنی 2 درج می‌شود. سپس 15 که باید در خانه 0 درج شود ولی این خانه قبلاً پر شده است، را در اولین خانه خالی بعد از آن یعنی 3 درج می‌کنیم. در نهایت 7 که باید در خانه 2 درج شود، چون این خانه قبلاً پر شده، را در خانه 4 درج می‌کنیم.

بنابراین جدول به صورت زیر می‌باشد:

0	1	2	3	4
10	11	6	7	15

مثال

جدول درهم سازی H[0..6] مفروض است. اگر عناصر A تا G را با ترتیب {A, C, B, D, E, F, G} وارد جدول کنیم، جدول چگونه خواهد شد؟

فرض: از روش آدرس دهی باز با وارسی خطی وتابع درهم سازی زیر استفاده شود:

Key	A	B	C	D	E	F	G
hash	3	5	3	4	5	6	3

پاسخ: ابتدا A در خانه 3 درج می‌شود. سپس C می‌خواهد در خانه 3 درج شود ولی چون پر است در خانه 4 درج می‌شود. سپس B در خانه 5 درج می‌شود و ...

0	1	2	3	4	5	6
E	F	G	A	C	B	D

تعداد وارسی‌های مورد نیاز(شامل وارسی‌هایی که موجب درج می‌شود) برای درج هر یک از عناصر در زیر آن نوشته شده است:

E	F	G	A	C	B	D
3	3	7	1	2	1	3

که در مجموع برابر 20 می‌باشد.

تذکر: تعداد وارسی‌ها در هر ترتیب مورد قبولی نیز برابر 20 می‌باشد.

مثال

با توجه به مثال قبل، آیا ترتیبی برای ورود A تا G می‌توان مشخص کرد، که باعث جدول زیر شود؟

0	1	2	3	4	5	6
C	E	B	G	F	D	A

پاسخ: خیر - چون ترتیب آمدن A و F متناقض است:

الف - A قبل از F آمده است، چون در جای F نشسته است.

ب - A بعد از F آمده است، چون A در درایه ششم درج شده، پس هنگام ورود آن درایه‌های سوم تا پنجم، شامل درایه چهارم که F در آن قرار دارد، پر بوده است. ■

مثال

خروجی تابع Hash برای هر یک از کلیدهای A تا G در زیر آورده شده است. تعداد حالت‌های مختلف ممکن برای جدول نهایی H، پس از درج هفت عنصر چند تا می‌باشد؟

key	A	B	C	D	E	F	G
hash	3	5	3	4	5	6	3

پاسخ:

تعداد حالات برای هر درایه در زیر آورده شده است:

درایه سوم : ۳ انتخاب. با یکی از عناصر A یا C یا G.

درایه چهارم : ۳ انتخاب. برای هر کدام از ۳ حالت پر شدن درایه سوم، درایه چهارم نیز ۳ انتخاب از بین A، C و D داشت.

درایه پنجم : ۴ انتخاب. شامل B یا A یا A، C و D، منهای عناصر درایه سوم و چهارم دارد.

درایه ششم : ۴ انتخاب. تمامی حروف منهای حروفی که در درایه های سوم و چهارم و پنجم استفاده شده اند.

درایه صفرم : ۳ انتخاب.

درایه اول : ۲ انتخاب.

درایه دوم : ۱ انتخاب.

پس کل حالت ها برابر است با:

$$3 \times 3 \times 4 \times 4 \times 3 \times 2 \times 1 = 864$$

نکاتی در روش آدرس دهی باز (با فرض درهم سازی یکنواخت)

همه عناصر در خود جدول درهم سازی ذخیره می شوند. بنابراین جدول درهم سازی می تواند پر شود که در آن صورت هیچ درج دیگری نمی تواند صورت بگیرد. یعنی ضریب بار هیچگاه بیشتر از 1 نمی شود. مظور از ضریب بار،

$$\alpha = \frac{n}{m} \text{ می باشد.}$$

تعداد مورد انتظار بررسی ها در یک جستجوی موفق حداقل $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$ است. ($\alpha < 1$)

(با فرض اینکه همه کلیدهای جدول، احتمال جستجو شدن برابر دارند)

تعداد مورد انتظار بررسی ها در یک جستجوی ناموفق حداقل $\frac{1}{1-\alpha}$ است. ($\alpha < 1$)

درج یک عنصر در یک جدول درهم سازی آدرس دهی باز، در حالت میانگین به حداقل $\frac{1}{1-\alpha}$ بررسی نیاز دارد.

درهم سازی پویا (dynamic hashing)

در روش های درهم سازی مانند باز یا زنجیره ای، تعداد عناصر حداکثر برابر اندازه جدول یا تقریباً برابر آن است. اما در بیشتر کاربردها، تعداد عناصر از قبل مشخص نمی باشند و با اعمال درج و حذف در هر لحظه، تغییر می کند. در روش درهم سازی پویا اندازه جدول درهم سازی بر حسب نیاز زیاد و کم می شود. اعمالی که بر روی این جدول درهم سازی انجام می شود عبارتند از:

درج ساده	جدول جای خالی دارد و عنصر مانند قبیل در جدول درج می شود.
حذف ساده	حذفی که موجب تغییر در اندازه جدول نمی شود.
درج با گسترش	قبل از درج، جدول پر است و اندازه جدول دو برابر می شود و عناصر موجود به جدول جدید منتقل شود.
حذف و فشرده سازی	بعد از حذف، نسبت تعداد عناصر به اندازه جدول کم می شود. بنابراین اندازه جدول را نصف کرده و همه عناصر موجود به جدول جدید منتقل می شود.

در درهم سازی پویا اگر فقط درج داشته باشیم، فرض می کنیم که اندازه جدول همیشه توانی از ۲ است. اگر اندازه جدول در ابتدا صفر باشد، جدولی با یک درایه ایجاد می شود و عنصر مورد نظر را در آن قرار می دهیم. در غیر اینصورت، اگر تعداد عناصر موجود در جدول برابر اندازه جدول باشد، درج با گسترش انجام می شود. یعنی جدولی جدید به اندازه دو برابر جدول موجود ایجاد شده و همه عناصر موجود در جدول با تابع درهم سازی جدید در جدول جدید درج شده و جدول قبلی آزاد می شود و جدول جدید تغییر نام داده و پارامترهای آن تنظیم می شود.

مثال

مثالی از نحوه گسترش جدول برای درج های متوالی.

i	1	2	3	4	5	6	7	8	9	10	11	...
اندازه جدول بعد از عمل i ام	1	2	4	4	8	8	8	8	16	16	16	16
تعداد گسترش ها	1	2	3	3	3	4	4	4	5	5	5	5

تعداد کل گسترش ها برای n درج، برابر $\lceil \lg n \rceil + 1$ است.

هزینه درج i ام برابر است با :

الف- برابر یک است، اگر گسترش نداشته باشیم.

ب- برابر $n_{i-1} + 1$ است، اگر گسترش داشته باشیم.

(چون قبل از درج آخر باید n_{i-1} عنصر موجود در جدول قبلی به جدول جدید منتقل شوند).

(n_i : تعداد عناصر جدول بعد از عمل i ام)

بنابراین مجموع هزینه n عمل درج برابر است با:

$$n + \sum_{k=0}^{\lceil \lg n \rceil} 2^k = n + 2^{\lceil \lg n \rceil + 1} - 1 = n + 2 \times 2^{\lceil \lg n \rceil} - 1 \approx n + 2n - 1 \leq 3n$$

یعنی هزینه سرشکن شده هر درج حداکثر ۳ واحد یا $O(1)$ است.

مثال

یک جدول درهم سازی را در نظر بگیرید که اندازه‌ی آن در ابتدا ۱ است و آن درایه هم خالی است. درهم سازی هم به صورت بسته (در مقابل زنجیره‌ای) است و با یک تابع درهم سازی ساده و وارسی خطی انجام می‌شود. برای درج یک عنصر x ، اگر جدول جا داشته باشد، x یک راست درج می‌شود و گرنه اندازه جدول را دو برابر می‌کنیم و عناصر فعلی را با هزینه‌ای برابر تعداد شان، به جدول جدید منتقل می‌کنیم و سپس x در جدول جدید درج می‌شود. اگر ۱۲ عنصر در این جدول درج شوند، هزینه کل در مجموع چه مقدار خواهد بود؟ (توجه کنید که هزینه در اینجا تعداد نوشتن در جدول است و بقیه هزینه‌ها را ثابت فرض می‌کنیم).

پاسخ:

در ابتدا چون اندازه جدول ۱ است، عنصر اول را درج می‌کنیم. برای درج عنصر دوم، چون جدول جا ندارد، اندازه جدول دو برابر شده و سپس عنصر جدید را درج کرده و عنصر موجود در جدول قبل را به جدول جدید منتقل می‌کنیم. برای درج عنصر سوم، چون جدول جا ندارد، اندازه جدول دو برابر شده و دو عنصر قبلی با هزینه ۲ به جدول جدید منتقل شده و عنصر جدید نیز درج می‌شود. برای درج عنصر چهارم، نیازی به دو برابر کردن اندازه جدول نیست، چون یک جای خالی در جدول موجود است. برای درج عنصر پنجم، چون جدول جا ندارد، اندازه جدول دو برابر شده و چهار عنصر موجود در جدول قبل به جدول جدید منتقل شده و عنصر پنجم درج می‌شود. به همین ترتیب ادامه داده و جدول زیر را پر می‌کنیم:

i	1	2	3	4	5	6	7	8	9	10	11	12
اندازه جدول	1	2	4	4	8	8	8	8	16	16	16	16
هزینه		1	2		4				8			

$$(1+2+4+8)+12 = (2^4 - 1) + 12$$

بنابراین هزینه کل نوشتن n عدد برابر $\sum_{k=0}^{\lceil \lg n \rceil} 2^k$ می‌باشد.

اگر بخواهیم عمل حذف را نیز کنار درج انجام دهیم، باید عمل فشرده سازی جدول را که با یک حذف انجام می‌شود، تعریف کنیم. در این حالت، عنصر مورد نظر را حذف می‌کنیم، سپس اگر تعداد عناصر موجود نسبت به اندازه جدول از حدی کمتر شد، اندازه جدول را نصف می‌کنیم و همه عناصر را به جدول جدید منتقل می‌کنیم. عمل فشرده سازی را وقتی انجام می‌دهیم که عمل i ام حذف و ضرب بار نیز $\alpha_{i-1} = \frac{n_{i-1}}{s_{i-1}}$ باشد.

(s_i اندازه جدول بعد از عمل i ام) (n_i تعداد عناصر جدول بعد از عمل i ام)

کنکور ارشد

(مہندسی IT - دولتی، ۸۹)

۱- فرض کنید که یک جدول درهم سازی به اندازه ۸۰ از روش آدرس دهی خطی باز استفاده می کند. ابتدا جدول خالی بوده است و تنها عملیات اضافه کردن و جستجو روی جدول انجام شده است. در حال حاضر وضعیت درایه های ۴۵ تا ۵۶ جدول به صورت زیر است. اعداد بالای آرایه، اندیس درایه ها و اعداد پایین آرایه خروجیتابع درهم سازی است. اگر یکبار دیگر از جدول درهم سازی خالی شروع کنیم و همان عملیات را به غیر از اضافه کردن کلید ۶ دوباره انجام دهیم، در درایه ۵۰ چه کلیدی قرار می گیرد؟

45	46	47	48	49	50	51	52	53	54	55	56
a	b	c	d	e	f	G	h	i	j		
46	46	46	47	46	51	47	46	48	49		

حل: جواب گزینه ۲ است.

با وارد کردن کلیدهای قیل از e یعنی d, c, b, a به جدول خالی، خواهیم داشت:

45	46	47	48	49	50	51	52	53	54	55	56
	a	b	c	d							

حال با توجه به صورت تست، از e صرف نظر می کنیم. کلید f را در خانه با اندیس 51 قرار می دهیم چون خروجی

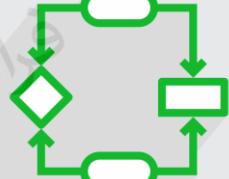
تابع درهم ساز برای آن برابر 51 است:

45	46	47	48	49	50	51	52	53	54	55	56
	a	b	c	d		f					

کلید g را طبق خروجی تابع درهم ساز باید در خانه با اندیس 47 درج کرد، ولی این خانه قبل از توسعه b پر شده است، بنابراین با توجه به روش آدرس دهی خطی باز، از آن خانه به سمت جلو حرکت کرده و g را در اولین خانه خالی یعنی 50 قرار می دهیم. بنابراین در درایه 50، کلید g قرار می گیرد.

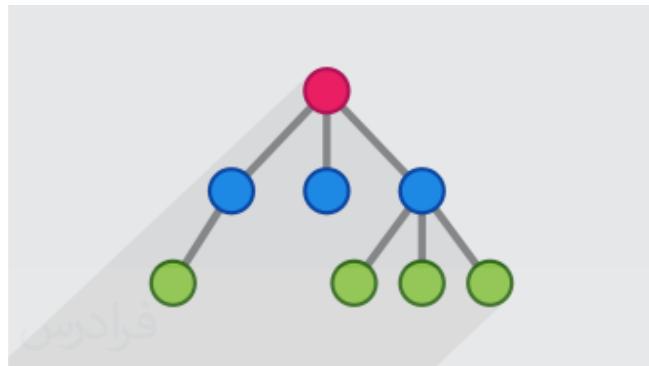


دسته بندی موضوعی آموزش‌های فرادرس، در ادامه آمده است:

 <p>مهندسی برق الکترونیک و روباتیک</p> <p>مهندسي برق الکترونيک و روباتيك - کلیک (+)</p>	 <p>هوش مصنوعی</p> <p>هوش مصنوعی - کلیک (+)</p>	 <p>آموزش‌های دانشگاهی و تخصصی</p> <p>آموزش‌های دانشگاهی و تخصصی - کلیک (+)</p>	 <p>برنامه‌نویسی</p> <p>برنامه‌نویسی - کلیک (+)</p>
 <p>مهارت‌های دانشگاهی</p> <p>مهارت های دانشگاهی - کلیک (+)</p>	 <p>مباحث مشترک</p> <p>مباحث مشترک - کلیک (+)</p>	 <p>دروس دانشگاهی</p> <p>دروس دانشگاهی - کلیک (+)</p>	 <p>مهندسي نرم‌افزار</p> <p>مهندسي نرم‌افزار - کلیک (+)</p>
 <p>آموزش‌های عمومی</p> <p>آموزش‌های عمومی - کلیک (+)</p>	 <p>طراحی و توسعه وب</p> <p>طراحی و توسعه وب - کلیک (+)</p>	 <p>نرم‌افزارهای عمومی</p> <p>نرم‌افزارهای عمومی - کلیک (+)</p>	 <p>نرم‌افزارهای تخصصی</p> <p>نرم‌افزارهای تخصصی - کلیک (+)</p>

منبع مطالعاتی تکمیلی مرتبط با این کتاب

آموزش ویدئویی ساختمان داده‌ها



ساختمان داده‌ها، یکی از دروس مهم و پایه‌ای دانشگاهی است که پیش نیاز دروس مختلف رشته کامپیوتر است و به عنوان مبحثی که نکات فراوانی دارد در کنکور کارشناسی ارشد کامپیوتر و دکترای هوش مصنوعی و نرم افزار از دروس با ضرایب بالا می‌باشد.

آموزش ساختمان داده‌ها، توسط مهندس فرشید شیرافکن، یکی از بهترین مدرسین مسلط به مباحث ساختمان داده‌ها، ارائه شده است.

مدرس: مهندس فرشید شیرافکن

مدت زمان: ۱۰ ساعت

faradars.org/fvds9402

جهت مشاهده آموزش ویدئویی این آموزش [کلیک](#) کنید.