به نام خدا

با سلام خدمت دانشجویان عزیز و آرزوی سلامتی برای همه عزیزان امیدوارم جزوه ساختمان داده که در زیر آمده است مورد استفاده شما عزیزان قرار بگیرید

توجه :ازبحث گراف فقط قسمت مربوط به درختها خوانده شود

آموزش ساختمان داده

مفاهيم مقدماتي

جهت دستيابي و پردازش سريع تر به داده ها معمولا آنها را طبق يک مدل رياضي يا منطقي سازماندهي مي كنند.

داده
نوع داده
نوع داده انتزاعي
ساختمان داده
عمليات روي ساختمان داده

داده

داده یک نمایش باینری از یک موجودیت منطقی قابل ذخیره سازی در حافظه کامپیوتر است .

داده ها درواقع مقادیری هستند که بعنوان ورودی به یک الگوریتم داده می شود تا پردازشی روی آنها انجام شود .

پردازش داده

هر نوع عملياتي، نظير محاسبات، مقايسه، جستجو، حذف يا تغيير داده كه توسط برنامه روي داده ها انجام مي پذيرد پردازش داده محسوب می شود.

نوع داده

داده ها نمایشی از اطلاعات در حافظه می باشند. نحوه نمایش يك داده خاص در حافظه توسط نوع داده (Data Type) آن مشخص مي شود. نوع داده مجموعه اي از مقادير و مجموعه اي از عملياتي كه روي اين مقادير اجرا مي شود را تعیین می کند .

نوع داده یک مفهوم ذهنی است که با مجموعه ای از خواص منطقی و اعمال مربوط به آن قابل تعریف است. وقتی نوع داده تعریف شد می توان آن را پیاده سازی کرد .

پیاده سازی می تواند سخت افزاری یا نرم افزاری باشد:

در پیاده سازی سخت افزاری مدارات لازم برای اجرای عملیات طراحی می شود .
در پیاده سازی نرم افزاری برنامه ای برای تفسیر رشته های بیتی و انجام عملیات موردنیاز با استفاده از دستورات سخت افزاری نوشته می شود .وقتی نوع داده پیاده سازی شد برنامه نویس می تواند از آنها برای حل مسائل استفاده کند .

انواع داده ای که به صورت سخت افزاری طراحی می شوند انواع داده ابتدايي (Primitive Data Type)نامیده می شوند. هر كامپيوتري داراي مجموعه اي از انواع داده ابتدايي است .

يك زبان برنامه نویسی سطح بالا امکانات خوبي را برای تعریف نوع متغير در اختيار برنامه نويس قرار مي دهد. تعريف نوع داده متغير مشخص مي كند محتويات حافظه چگونه بايد تفسير شود و كامپايلر را قادر مي سازد كه عملكرد متناسب با متغيررا تشخيص دهد .هر زبان برنامه نویسی مجموعه ای از انواع داده را تعریف می کند

نوع داده انتزاعي

جدا از جنبه های سخت افزاری، اگر مفهوم نوع داده بر اساس آنچه برنامه نویس می خواهد به صورت نرم افزاری پیاده سازی شود نوع داده انتزاعی (Abstract Data Type)نامیده می شود .اگر مفهوم نوع داده از توانائیهای سخت افزار جدا شود، تعداد نامحدودی از انواع نوع داده را می توان درنظر گرفت .

ساختمان داده

مجموعه مقادیر و عملیات روی آنها تشکیل یک ساختار ریاضی را می دهند. نوع داده انتزاعی به مدل ریاضی که یک نوع داده را تعریف می کند اشاره دارد .مدل منطقي يا رياضي سازماندهي داده ها به يك صورت خاص را ساختمان داده مي نامند. ساختمان داده مشخصات عناصر، ارتباط بین آنها و عملیاتی است که روی آنها انجام می شود را تعیین می کند .

انواع ساختمان داده

.۱ساختمان داده خطي : يك ساختمان داده را خطي مي گويند هرگاه عناصر آن تشكيل يك دنباله را دهند، به بيان ديگر يك ليست خطي باشند .
براي نمايش ليست خطي دو روش اساسي وجود دارد:
رابطه خطي بين عناصربه وسيله خانه هاي متوالي حافظه نمايش داده مي شود(آرايه.(
رابطه خطي بين عناصر به وسيله اشاره گرها نمايش داده مي شود(لیست پیوندی.(
۲ساختمان داده غير خطي مانند درخت ها و گراف ها

عمليات روي ساختمان داده

داده هايي كه در ساختمان داده ها ظاهر مي شوند به وسيله عمليات مشخصي پردازش مي شوند. در واقع ساختمان داده خاصي كه برنامه نويس براي يك مسئله انتخاب مي كند بستگي زيادي به ميزان عمليات خاصي دارد كه در آن مسئله انجام مي شود. برخي از اين عمليات كه زياد مورد استفاده قرار مي گيرند عبارتند از :

.۱پيمايش : دقیقا یک بار دسترسي به كليه داده های ساختمان داده
.۲جستجو : يافتن يك داده يا مجموعه ای از داده ها با شرايط خاصي درون ساختمان داده
.۳اضافه : افزودن يك داده جديد به ساختمان داده.
.٤حذف : حذف يك داده از ساختمان داده.
.٥مرتب سازي : قرار دادن داده ها در كنار هم با يك نظم معين.
.٦ادغام : تركيب داده هاي دو ساختمان داده مرتب و بدست آوردن یک ساختمان داده مرتب ديگر

.۷اتصال : پيوند دو ساختمان داده به يكديگر

الگوريتم

الگوریتم های مختلفی که روی هر ساختار داده ای پیاده سازی می شوند، توسط دو معیار پیچیدگی حافظه ای و زمان مورد ارزیابی قرار می گیرند .

تعريف
نحوه بيان الگوريتم
ارزیابی کارائی الگوریتم ها
پیچیدگی حافظه
پيچيدگي زماني

تعريف

به طور خلاصه مجموعه ای از دستوالعمل ها برای حل یک مسئله را الگوریتم می گویند. تعریف دقیق تر الگوریم به صورت زیر است :

یک الگوریتم مجموعه ای از متناهی از دستورات برای حل یک مسئله خاص توسط انسان یا ماشین است، که ترتیب انجام عملیات در آن مشخص شده و عملیات در زمان معینی خاتمه پیدا می کند. هر دستورالعمل در الگوریتم باید مختصر، دقیق، و صریح باشد .

یک الگوریتم پنج خاصیت زیر را باید دارا باشد:

.۱متناهی بودن. یک الگوریتم باید همیشه بعد از تعدادی گام به پایان برسد.
.۲صراحت. فعلی که در هر قدم الگوریتم انجام می گیرد باید مختصر، صریح و غیر مبهم باشد.
.۳ورودی. مقادیری هستند که ابتدا، قبل از شروع، به الگوریتم داده می شوند .
.٤خروجی. مقادیری هستند توسط الگوریم تولید می شود و رابطه مشخصی با

ورودی ها دارند.
.٥کارائی. دستورات الگوریتم درحد کفایت باید ساده و دقیق باشند تا یک انسان مانند یک روبات بتواند آنها را با استفاده از قلم و کاغذ بدون احتیاج به فکر کردن در زمان معینی انجام بدهد.

الگوریتم ها قرن ها برای حل مسائلی که بشر با آنها روبرو بوده استفاده می شده اند. تقریبا کلیه برنامه های کامپیوتر، بجز برنامه های کاربردی هوش مصنوعی، دربرگیرنده الگوریتم هستند. مشهورترین الگوریتم در تاریخ، الگوریتم اقلیدسی، مربوط به زمان یونان باستان است که برای محاسبه بزرگترین مقسوم علیه مشترک دو عدد صحیح به کار می رفته است و هنوز در دنیای ریاضی کاربرد دارد .

خلق الگوریتم های زیبا، ساده و با کمترین مراحل، یکی از چالش های برنامه نویسی است

نحوه بیان الگوریتم

الگوریتم های می توانند با نمادهای مختلفی بیان شوند:

•زبان طبیعی. استفاده از عبارات زبان طبیعی برای بیان الگوریتمی ممکن است باعث طولانی و مبهم شدن آن بشود و برای الگوریتم های پیچیده و فنی بندرت استفاده می شود.
•زبان های برنامه نویسی .در ابتدا بیان الگوریتم با زبان های برنامه نویسی موجود طرفدار داشت .
Pseudo Code و فلوچارت. راه های ساخت یافته ای برای نمایش الگوریتم، که درحین استقلال از زبان برنامه نویسی خاصی، از ابهام پرهیز می کنند.

امروزه الگوریتم ها معمولا با استفاده از Pseudo Code در یک زبان برنامه نویسی که معمولا پیاده سازی نشده بیان می شوند. در طی این درس از کدهای ساختگی که در ادامه شرح داده می شوند برای بیان الگوریتم ها استفاده می شود .

```
variable := value
```

اختصاص مقداری به یک متغیر را نشان می دهد.

```
if (condition) then
  statements۱

else
  statements۲

end if
```

برای بیان تصمیم گیری، اگر شرط برقرار باشد عبارت ۱ انجام می گیرد و اگر برقرار نباشد عبارت ۲

```
while (condition)
  statement
end while
```

برای نمایش حلقه تکرار، اگر شرط برقرار باشد دستورات تکرار می شوند.

```
repeat until (condition)
  statement
end loop
```

برای نمایش حلقه تکرار، تا وقتی شرط برقرار نشده باشد دستورات تکرار می شوند. درصورت برقرار بودن شرط از حلقه خارج می شود .

```
for (counter:=value۱ to value۲ )
  statement
end for
```

برای نمایش تکرار عبارتی به تعداد معینی، شمارنده از مقدار۱ شروع شده در هربار تکرار یک واحد به آن اضافه می شود تا به مقدار۲ برسد. برای شمارش معکوس به جای to از down to قرار می گیرد .

انتهای الگوریتم توسط کلمه end مشخص می شود.

مثال. الگوریتم زیر به متغیر x مقادیر ۱ تا ۱۰ بجز عدد ۵ را اختصاص می دهد .

```
x:=٠
while (x < ١٠)
  if x = ٤ then
    x := x + ٢
  else
    x := x + ١
  end if
end while
end
```

ارزیابی کارائی الگوریتم ها

الگوریتم های مختلفی برای حل یک مسئله ممکن است طراحی شده باشند. برای انتخاب بهترین الگوریتم باید معیاری جهت مقایسه کارائی الگوریتم ها داشته باشیم. ارزیابی در دو مرحله انجام می شود؛ آنالیز کارائی و اندازه گیری کارائی است .

آنالیز کارائی یک تخمین اولیه است با دو معیار پیچیدگی فضائی (space complexity)و پیچیدگی زمانی (time complexity) سنجیده می شود که رفتار الگوریتم را در زمان اجرا با مجموعه ای از ورودی های منتخب توصیف می کنند .

بعد از پیاده سازی الگوریتم با یک زبان برنامه نویسی، آمار حقیقی درباره زمان و حافظه مصرف شده توسط الگوریتم در حین اجرا جمع آوری می شود .

پیچیدگی حافظه

پیچیدگی حاظه ای میزان فضائی از حافظه است که برنامه برای اجرای کامل به آن نیاز دارد. فضای مورد نیاز در هربرنامه مجموع قسمت های زیر است :

- بخش ثابت فضا که معمولا شامل فضای دستورالعمل، فضای متغیرهای با اندازه ثابت و فضای لازم برای ذخیره ورودی و خروجی های برنامه است.
- بخش متغیر فضا شامل فضای پشته و فضای موردنیاز برای مقادیر متغیرهائی که اندازه آنها بستگی به مسئله و مشخصات ورودی دارد .

در تحلیل فضای لازم روی تخمین بخش متغیر تاکید نداریم زیرا برای هرمسئله ابتدا باید مشخصات موردی را تعیین کنیم که کار دشواری است

پیچیدگی زمانی

زمان اجرا مقدار زمانی از کامپیوتر است که برنامه برای اجرای کامل مصرف می کند.
برای محاسبه پیچیدگی زمان الگوریتم ابتدا تعداد قدم های الگوریتم به صورت تابعی از اندازه مسئله مشخص می شود، برای انجام این کار تعداد تکرارعملیات اصلی الگوریتم محاسبه می شود و به صورت تابع) $f(n)$که n تعداد ورودی هاست) بیان می شود.
سپس تابع $g(n)$، که مرتبه بزرگی تابع $f(n)$را وقتی اندازه ورودی به اندازه کافی بزرگ است نشان می دهد، بدست می آید. در نهایت پیچیدگی الگوریتم برای نشان دادن رفتار الگوریتم با ورودی های مختلف با استفاده از نمادها **O**، **Θ**و **Ω**بیان می شود .

تعریف Big-O حدبالا

تابع $f(n)$را نظر بگیرید که برای کلیه $n \geq 0$است، می گوئیم $f(n) = O(g(n))$اگر ثابت های مثبت n_0 و c وجود داشته باشند به طوریکه از یک n_0 به بعد همیشه $f(n) \leq cg(n)$برقرار باشد .

این نماد حدبالائی برای تابع $f(n)$می دهد و وقتی بکار می رود که رفتار الگوریتم بدترین حالت و بیشترین زمان اجرا را برای مقادیر معین ورودی دارد

تعریف Big-Ω حدپائین

تابع $f(n)$را نظر بگیرید که برای کلیه $n \geq 0$ است ، می گوئیم $f(n) = \Omega (g(n))$اگر ثابت های مثبت n_0 و c وجود داشته باشند به طوریکه از یک n_0 به بعد همیشه $f(n) \geq cg(n)$برقرار باشد .

این نماد حد پائینی برای تابع $f(n)$می دهد و وقتی بکار می رود که رفتار الگوریتم بهترین حالت و کمترین زمان اجرا را برای مقادیر معین ورودی دارد

تعریف Big-Θ حدمتوسط

تابع $f(n)$را نظر بگیرید که برای کلیه $n \geq 0$ است، می گوئیم $f(n) = \Theta(g(n))$اگر ثابت های مثبتn_0 ، c_1و c_2 وجود داشته باشند به طوریکه از یک n_0 به بعد همیشه $c_1g(n) \leq f(n) \leq c_2g(n)$برقرار باشد .

این نماد حدمتوسطی برای تابع $f(n)$می دهد و زمان اجرای الگوریتم را به صورت میانگینی از تعداد عملیات انجام شده با کلیه نمونه ورودی های مسئله نشان می دهد .

قضیه .اگر $f(n)=a_m n^m+a_{m-1}n^{m-1}+\ldots+a_1 n+a_0$ در اینصورت $f(n)=O(n^m)$ است

مثال. الگوریتم مرتب سازی حبابی را درنظر بگیرید .

```
for (i:=1 to n-1)
  for (j:=1 to n-1)
    if aj>aj+1 then exchange(aj,aj+1)
```

با درنظر گرفتن عمل مقایسه بعنوان عملگر اصلی، دستور If در الگوریتم فوق $(n-1)^2$ بار تکرار می شود. بنابراین $f(n)=(n-1)^2=n^2-2n+1$ و طبق قضیه $g(n)=n^2$ است. بنابراین پیچیدگی الگوریتم فوق برابر با $O(n2)$ می باشد

نکته. اگر زمان الگوریتم وابسته به ورودی نباشد با نماد $O(1)$ نشان داده می شود.

نکته. باید به اندازه کافی الگوریتم را درک کرده باشیم تا بهترین و بدترین رفتار را تولید و محاسبه کنیم. چون برآورد رفتار آماری ورودی ها امری دشوار است، در اکثر موارد به بدترین حالت قناعت می کنیم.

نکته. اگر الگوریتم شامل بخش های مختلفی باشد که هر قسمت پیچیدگی متفاوتی دارد، مرتبه بزرگی هر قسمت را پیدا کرده و بزرگترین مرتبه را بعنوان پیچیدگی کل الگوریتم درنظرمی گیریم.

غالبا پیچیدگی $g(n)$ یکی از توابع زیر است) n :پیچیدگی خطی(،) $\log n$ لگاریتمی(،) n^a چندجمله ای (و a^n که) $a\geq 2$ نمائی .(

در زیر مرتبه اجرائی چند تابع به ترتیب صعودی نوشته شده است.

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

آرایه

يکي از پرکاربردترین ساختمان های داده آرایه است که اغلب برای پیاده سازی داده های انتزاعی خطی بکار می رود

تعريف آرايه
آرايه هاي يک بعدی
آرايه هاي دو بعدی
محاسبه فضا و آدرس
آرايه های پويا
الگوريتم های درج و حذف

تعريف آرايه

آرايه (array) ليست متناهی از عناصر داده‌ای هم نوع است

محل يک عنصر درون آرايه توسط انديس (Index) معين می شود. عنصر a_i در مکان iام آرايه قرار مي گيرد به اين ترتيب مي توان به صورت تصادفي مقدار آرايه را بازيابي کرد. البته با روش ترتيبي هم مي توان به مقادير عناصر ليست دسترسي پيدا کرد .

به عناصر آرايه ممكن است به كمك مجموعه اي از انديس ها مراجعه شود .

فرم کلي تعريف آرايه در زبان برنامه نويسی Pascal به صورت زير است:

ArrayName : ARRAY [IndexType۱, IndexType۲,…, IndexTypen] OF Type;

IndexTypeمجموعه انديس را تعيين می کند و می تواند از هر نوع اسکالری باشد. اگر دستور کامپايلر {$R+} فعال نباشد کنترلی روی انديس های غير مجاز نيست. اگر فعال باشد در صورت استفاده از انديس غيرمجاز پيغام خطای Range Check Errorصادر می شود .

در زبان برنامه نويسی C آرايه به صورت کلی زير تعريف می شود:

Type ArrayName[Size۱] [Size۲] …[Sizen];

Sizeتعداد عناصر يک بعد آرايه را تعيين می کند. در زبان C کليه انديس ها عددی هستند و از صفر شروع می شوند (۰-based indexing)و کنترلی روی محدوده انديس ها وجود ندارد .

دراکثر زبان ها به آرایه به صورت ایستا حافظه اختصاص می دهند و اندازه آرایه در طول اجرای برنامه ثابت است. مگر اینکه حافظه پویا صریحا توسط برنامه نویس استفاده شود .

آرایه هاي یک بعدی

آرایه یک بعدی مجموعه متناهي از زوج ها به صورت ‹ اندیس،مقدار› است. بدین معنی که، به ازاي يك اندیس یک مقدار مربوط به آن وجود دارد .

براي تعریف آرایه يك بعدي یک مجموعه اندیس تعریف می شود

مثال .(C)آرایه num با ۲۰ عنصر صحیح تعریف شده است. عناصر آرایه در خانه های [۱۹]num[۰], num[۱], num[۲],..., numذخیره مي شوند .

int num [۲۰] ;

مثال .(C)آرایه کاراکتری num با ٤ عنصر تعریف و مقداردهی شده است.

```
Char num[٤]={'d','a','t','a'};
یا
Char num[٤]={"data"}
```

آرایه هاي دو بعدی

يك آرایه دو بعدی مجموعه اي با m×n عنصر داده اي است که هر عنصر آن با يك جفت اندیس مشخص مي شود .

آرایه دو بعدي را مي توان به جدولي تشبیه کرد که داراي m سطر و n ستون است. هر سطر شامل عناصري است که اندیس هاي اول آنها برابر است و هر ستون شامل عناصري هستند که اندیس هاي دوم آنها برابر هستند .

آرایه هاي دوبعدي به عنوان ماتریس به کار مي روند .

در تعریف آرایه دو بعدی دو مجموعه اندیس معین می شود. اندیس اول تعداد سطرها و اندیس آرایه تعداد ستون ها را مشخص می کند .

مثال .(Pascal)آرایه Table از نوع اعداد حقیقي با ٥ سطر و ٤ ستون تعریف شده است.

Table : Array[١..٥,٣..٦] of Real;
يا
Table : Array[١..٥] of Array [٣..٦] of Real;

مثال .(C)آرايه A از نوع اعداد صحيح با ٣ سطر و ٤ ستون

int A[٣][٤];

مثال .(C)آرایه دوبعدی num را می توان به دو صورت مقداردهی اولیه کرد.

```
int num [٤][٣]={{١٥,٦,١٣},{٩,١٧,٢},{٤,٥,٤},{١٠,١١,١٢}} ;
يا
int num [٤][٣]={١٥,٩,١٣,٩,١٧,٢,٤,٥,٤,١٠,١١,١٢};
```

آرايه هاي چندبعدي

آرايهn بعدی مجموعه اي از m١×m٢×…×mn عنصر داده اي است كه هر عنصر توسط nانديس نظير i١,i٢,…,in مشخص مي شوند. آرايه هاي چند بعدي در حافظه به صورت دنباله اي از خانه هاي پشت سر هم ذخيره مي شوند .

محاسبه فضا و آدرس

هر متغيري كه تعريف مي شود مقداري فضای حافظه را به خود اختصاص مي دهد به نوع داده متغير بستگي دارد. براي بدست آوردن ميزان فضاي اشغال شده بوسيله يك آرايه كافي است كه تعداد عناصر آرايه در تعداد بایت های نوع آرايه ضرب شود. تعداد عناصر آرايه را طول يا اندازه آرايه مي گويند .

در زبان Pascal اندازه آرایه به صورت زیر محاسبه می شود :

A[L١..U١, L٢..U٢, ….Ln..Un]

$$\prod_{i=1}^{n} U_i - L_i + 1$$

در زبان C اندازه آرایه به صورت زیر محاسبه می شود:

A[M١][M٢]…. [Mn]

$$\prod_{i=1}^{n} M_i$$

نوع های داده در زبان های برنامه نویسی C و

مثال .(Pascal)آرایه زیر ٢٤ عنصر کاراکتری دارد.

A: Array [١..٢,١..٣,١..٢,١..٢] of Char;
(٢-١+١)(٣-١+١)(٢-١+١)(٢-١+١) = ٢٤.

مثال .(C)میزان فضاي اشغال شده توسط آرایه num به صورت زیر محاسبه می شود:

int num [١٠][٥] ;
١٠×٥×٢ = ١٠٠

نمایش آرایه ها

حافظه کامپیوتر را می توان به صورت یک آرایه یک بعدی در نظر گرفت که آدرس ها اندیس های آن هستند. بنابراین در واقع برای نمایش آرایه در حافظه، یک آرایه n بعدی در یک آرایه یک بعدی جای داده می شود. برای این کار باید اندیس های آرایه n بعدی تبدیل به آدرس حافظه شود. دو روش برای این تبدیل وجود دارد :

- روش سطری(row-major order)
- روش ستونی (column-major order)

روش سطری

در روش سطری عناصر یک سطر پشت سر هم در حافظه قرار می گیرند در نتیجه اندیس های سمت راست سریع تر حرکت می کنند .

برای بدست آوردن آدرس هر عنصر درون حافظه کامپیوتر فقط آدرس اولین عنصر آرایه لازم است. این آدرس را آدرس شروع یا α می نامیم. با استفاده از آن آدرس محل بقیه عناصر آرایه در حافظه بدست می آید .

آرایه a[٢][٣]را درنظر بگیرید که از آدرس α حافظه شروع شده است.

آدرس	مقدار
$\alpha + 0 \times \text{sizeof(T)}$	a[0] [0]
$\alpha + 1 \times \text{sizeof(T)}$	a[0] [1]
$\alpha + 2 \times \text{sizeof(T)}$	a[0] [2]
$\alpha + 3 \times \text{sizeof(T)}$	a[1] [0]
$\alpha + 4 \times \text{sizeof(T)}$	a[1] [1]
$\alpha + 5 \times \text{sizeof(T)}$	a[1] [2]

آدرس اولین عنصر سطر اول برابر با $\alpha + ۰\times \text{sizeof}(t) = \alpha$. چون در هر سطر ۳ عنصر وجود دارد بنابراین آدرس عنصر اول سطر دوم برابربا $\alpha + ۳\times \text{sizeof}(t)$ می شود .

می توان یک فرمول کلی برای آرایه n بعدی بدست آورد. اگر آرایه $A[\delta_{۱}] [\delta_{۲}] [\ldots] [\delta_n]$ از آدرس α حافظه شروع شده باشد. آدرس خانه $A[i_{۱}][i_{۲}]\ldots[i_n]$ این آرایه به صورت زیر محاسبه می شود W)تعداد بایت‌هاي نوع آرایه است :(

$$\alpha + W [(i_{۱})\delta_{۲}\delta_{۳}\ldots \delta_n + (i_{۲})\delta_{۳}\ldots\delta_n + \ldots + (i_n)]$$

اگر آرایه توسط زبان Pascal به صورت $A[L_{۱}..U_{۱}, L_{۲}..U_{۲}, \ldots,L_n..U_n]$ تعریف شده باشد فرمول به شکل زیر در می آید :

$$\alpha + W [(i_{۱}-L_{۱})\delta_{۲}\delta_{۳}\ldots \delta_n + (i_{۲}-L_{۲})\delta_{۳}\ldots\delta_n + \ldots + (i_n-L_n)]$$

که $\delta_i=U_i-L_i$

مثال ۱. آدرس خانه های آرایه یک بعدی به طریق زیر محاسبه می شود.

آدرس $A[i] = \alpha + W(i-L)$

مثال ۲. آرایه دو بعدی A را در نظر بگیرید. اگر آرایه از آدرس ۱۰۰ ذخیره شده باشد، آدرس خانه $A[۱۲][٤]$ به طریق زیر محاسبه می شود.

A : Array [۱۰..۱۵][۳..۵] of integer;
$\delta_{۱}=۱۵-۱۰+۱=٦$, $\delta_{۲}=۵-۳+۱=۳$

آدرس $A[i_{۱},i_{۲}] = \alpha + W [(i_{۱}-L_{۱})\delta_{۲}+(i_{۲}-L_{۲})]$

آدرس $A[12,4] = 100 + 2\ [\ (12-10)3+(4-3)\ ]$
$=114$

مثال ۳. اگر آرایه سه بعدی num از آدرس ۲۰۰۰ به بعد ذخیره شده باشد، آدرس خانه $A[3][2][1]$ به طریق زیر محاسبه می شود:

float A[۴][۵][۳] ;
$\delta_1=4\ ,\delta_2=5,\ \delta_3=3$

آدرس $A[i_1][i_2][i_3] = \alpha + W\ [\ (i_1)\delta_2\delta_3+(i_2)\delta_3+(i_3)]$
آدرس $A[3][2][1] = 2000 + 4\ [\ (3)5\times 3+(2)3+(1)]$
$=2208$

روش ستونی

در روش ستونی عناصر یک ستون پشت سر هم در حافظه قرار می گیرند در نتیجه اندیس های سمت چپ سریع تر حرکت می کنند .

آدرس خانه $A[i_1][i_2]\ldots[i_n]$ در این روش توسط فرمول زیر محاسبه می شود:

$$\alpha + W\ [\ (i_1) + \delta_1(i_2)+ \delta_1\delta_2(i_3) + \ldots + \delta_1\delta_2\delta_3\ldots\delta_{n-1}(i_n)]$$

اگر آرایه توسط زبان Pascal تعریف شده باشد فرمول به شکل زیر در می آید :

$$\alpha + W\ [\ (i_1-L_1) + \delta_1(i_2-L_2)+ \delta_1\delta_2(i_3-L_3) + \ldots + \delta_1\delta_2\delta_3\ldots\delta_{n-1}(i_n-L_n)]$$
$$\delta_i=U_i-L_i$$

مثال. درمثال ۲ اگر آرایه به صورت ستونی ذخیره شده باشد:

$A[i_1,i_2] = \alpha + W[\ (i_1-L_1) + \delta_1(i_2-L_2)]$
آدرس $A[12,4] = 100 + 2\ [\ (12-10) +6(4-3)]$
$=116$

مثال. درمثال ۳ اگر آرایه به صورت ستونی ذخیره شده باشد:

آدرس $A[i۱][i۲][i۳] = \alpha + W[(i۱)+ \delta۱\ (i۲) + \delta۱\delta۲\ (i۳)]$
آدرس $A[۳][۲][۱] = ۲۰۰۰ + ٤\ [\ (۳)\ +٤(۲)+٤\times٥(۱)]$
$=۲۱۲٤$

آرایه های پویا

آرایه پویا (dynamic array) آرایه است اندازه اش در زمان اجرا با عمل درج یا حذف عناصردر آن تغيير می کند .

در زبان برنامه نويسی Visual Basic توسط دستور REDIM و در زبان C با تعريف حافظه پويا می توان آرايه پويا ايجاد كرد. در بعضی زبان ها مانند Perlكليه آرایه ها به صورت پويا هستند .

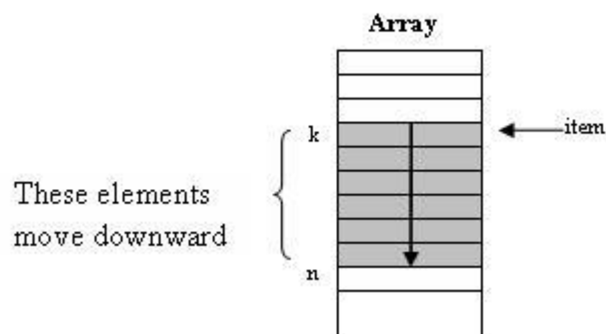
<u>برنامه مقداردهي عناصر آرايه يک بعدی پويا در زبانC</u>

<u>برنامه مقداردهي عناصر آرايه دو بعدی پويا در زبانC</u>

الگوریتم های درج و حذف

درج عنصری در آرایه

برای درج عنصری در آرایه تعدادی از عناصر باید به سمت پائين منتقل شوند تا عنصر جديد در محل مورد نظر قرار گيرد. اگر بخواهيم عنصر جديد در مكان k ام آرايه درج شود كليه عناصر از kبه بعد بايد شيفت داده شوند، سپس عنصر در مكانK ام ذخيره شود .



در كل n-k+۱ عنصر بايد جابجا شوند. اگر عنصر جديد در محل آخرين عنصر درج شود تنها عنصر آخر آرايه جابجا می شود. بدترين حالت زمانی اتفاق می افتد كه بخواهيم عنصر جديد را درمكان اول آرايه درج كنيم در اين حالت تعداد جابجائيها برابر است با n می شود .
به طور متوسط نياز به ۲/(n+۱)جابجائي است.

با هربار عمل درج یک واحد به n تعداد عناصر آرایه اضافه می شود n .تعداد عناصری که در آرایه درج شده اند را نشان می دهد و ربطی به طول آرایه ندارد .

الگوریتم زیر عنصر item را در مکان k ام آرایه A با n عنصر درج می کند .

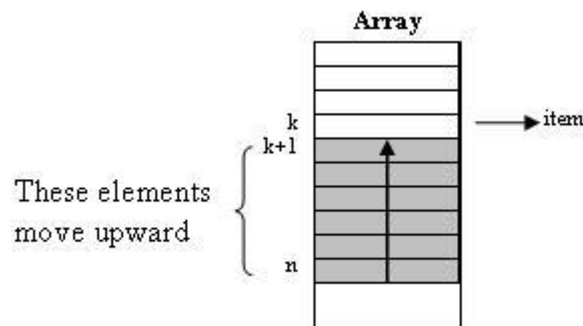
```
for (i:= n down to k )

  A[j+۱] := A[j]
end for
A[k] := item
n := n +۱
end
```

پيچيدگي الگوريتم فوق O(n)است

حذف عنصری از آرایه

وقتی عنصری از آرایه حذف می شود عناصر بعدی باید به سمت بالا منتقل شوند تا محل عنصر حذف شده پر شود. برای حذف عنصری که در مکان k ام قرار دارد، کلیه عناصر از k+۱به بعد باید به سمت بالا شیفت داده شوند .



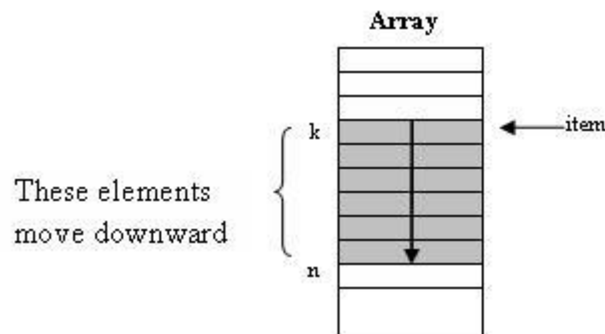
در کل n-k عنصر باید جابجا شوند. کمترین جابجائی وقتی است که عنصر آخر حذف شود که هیچ عنصری جابجا نمی شود. در بدترین حالت تعداد جابجائي‌ها برابر با n-۱ است وقتی که اولین عنصر آرایه حذف می شود.
به طور متوسط نیاز به ۲/(n-۱)جابجائي است.
با هربار عمل درج یک واحد به n تعداد عناصر آرایه اضافه می شود n .تعداد عناصری که در آرایه درج شده اند را نشان می دهد و ربطی به طول آرایه ندارد .

الگوریتم زیر عنصر kام آرایه A را حذف و در item ذخیره می کند .

```
item := A[k]
for (j := k to n – ۱)
   A[j] := A[j + ۱]
end for
n := n –۱
end
```

پیچیدگي الگوریتم فوق O(n)است.

زیربرنامه درج عنصری در آرایه به زبانC
زیربرنامه حذف عنصری از آرایه به زبانC
برنامه درج و حذف عنصری در آرایه به زبانC++



در کل n-k+۱ عنصر باید جابجا شوند. اگر عنصر جدید در محل آخرین عنصر درج شود تنها عنصر آخر آرایه جابجا می شود. بدترین حالت زمانی اتفاق می افتد که بخواهیم عنصر جدید را درمکان اول آرایه درج کنیم در این حالت تعداد جابجائي‌ها برابر است با n می شود .
به طور متوسط نیاز به (n+۱)/۲جابجائي است.
با هربار عمل درج یک واحد به n تعداد عناصر آرایه اضافه می شود .n تعداد عناصری که در آرایه درج شده اند را نشان می دهد و ربطی به طول آرایه ندارد .

الگوریتم زیر عنصر item را در مکان k ام آرایه A با n عنصر درج می کند .

```
for (i:= n down to k )
a
end for
A[k] := item
n := n +۱
end
```

پيچيدگي الگوريتم فوق O(n) است.

همانطور که مشاهده می شود عملیات درج و حذف در آرایه به طور متوسط منجر به انتقال نصف عناصر آرایه می شود. بنابراین در مواردی که مجموعه عناصر داده ای به طور مکرر درحال اضافه و حذف هستند آرایه خطي روش کارآمدي براي ذخيره داده ها نیست.

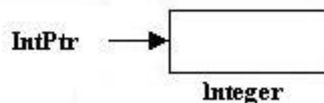
حافظه پویا و اشاره گرها

برای پیاده سازی ساختارهای پویا از اشاره گرها استفاده می شود.

اعلان متغیر اشاره گر
مقداردهی اولیه اشاره گر
عملیات اشاره گرها
تخصیص حافظه پویا
اشاره گر به آرایه
اشاره گر به رکورد

یک اشاره گر (Pointer)نوع خاصی از متغیر است که حاوی آدرسی از حافظه می باشد . درحالیکه یک متغیر استاندارد بایت هائی از حافظه را تعیین می کند که برای ذخیره نوع خاصی از داده کنار گذاشته شده است، متغیر اشاره گر به بخش هایی از حافظه که توسط متغیر دیگری اشغال شده اشاره می کند .

اشاره گرها امکان استفاده از حافظه آزاد را فراهم می کنند. علاوه براین، به طور پویا در طی اجرای برنامه می توان آنها را ایجاد یا حذف کرد .



از اشاره گرها برای ایجاد ساختمان داده های دیگر نظیر لیست های پیوندی، پشته، صف و درخت های دودوئی استفاده می شود

اعلان متغیر اشاره گر

هر متغیر اشاره گر به نوع خاصی از داده اشاره می کند. در برنامه باید به کامپایلر اعلان شود که نوع داده ای که اشاره گر به آن اشاره می کند چیست .

در زبان پاسکال، علامت (^) قبل از نوع داده قرار می گیرد تا متغیری را به عنوان اشاره گر معرفی کند .

در زبانC ، علامت (*) برای نشان دادن اشاره گر بودن متغیری اضافه می شود.
علامت ستاره را می توان بلافاصله بعد از نوع داده یا قبل از نام متغیر قرار داد .

ل .(Pascal)متغیر اشاره گر Pxبه داده صحیحی اشاره می کند.

Px : ^Integer;

مثال .(C)اشاره گر k به یک داده صحیح اشاره می کند.

int *k; یاint* k;

علامت ستاره به معنی داده ای است که k به آن اشاره می کند.

مقداردهی اولیه اشاره گر

نکته مهم هنگام کار با اشاره گرها مقدار دهی اولیه آنهاست. اگر یک اشاره گر را فقط اعلان کنید و بدون مقداردهی از آن استفاده کنید به محل نامعینی از حافظه اشاره کند و این می تواند مشکلاتی را روی سیستم تولید کند .

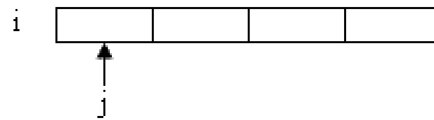
یک روش کلی مقداردهی اشاره گر با مقدارصفر یا تهی (ثابت NULL در زبان C و ثابت nil در زبان پاسکال) است .

int *x=NULL; یاint *x=۰;

راه معمول دیگر نسبت دادن آدرس یک متغیر استاندارد، توسط عملگر آدرس، به یک متغیر اشاره گر است .(C) .به برنامه زیر دقت کنید.

```
#include <iostream.h>
void main()
{
int i;
int* j;
j = &i;
i = ۱۰;
cout << "i is " i;
cout << "\n j is " << j << "\n";
}
```

آدرس متغیر i توسط عملگر &بدست آمده و به اشاره گر j نسبت داده می شود، بنابراین jبه متغیر i اشاره می کند .i یک متغیر صحیح است و ٤ بایت حافظه را اشغال می کند، jبه اولین بایت از این ٤ بایت اشاره می کند .



وقتی مقدار متغیر i چاپ می شود عدد ١٠ نمایش داده می شود. با چاپ متغیر اشاره گر jیک عدد طولانی تر نشان داده می شود که آدرسی در حافظه است .

نکته. نوع اشاره گر و متغیری که به آن اشاره می کند باید یکسان باشد

عملیات اشاره گرها

عملیات جمع و تفریق را می توان روی متغیرهای اشاره گر انجام داد. ضرب و تقسیم را روی یک اشاره گر نمی توان استفاده کرد. نکته مهمی که باید به آن توجه کرد این است که چون اشاره گر آدرسی در حافظه است وقتی عملیاتی که روی آن انجام می گیرد رفتار متفاوتی دارد. برای مثال عمل جمع اشاره گر را به تعداد بایت های نوع داده آن حرکت می دهد

.(C)چون a اشاره گری به یک عدد صحیح است و نوع صحیح ٤ بایت دارد با عمل افزایش ٤ واحد به a اضافه می شود. یعنی به ٤ بایت بعدی حافظه اشاره می کند و دیگر به همان ٤ بایت قبلی اشاره نمی کند .

```
int *a;
a++;
```

مثال .(C)اشاره گر p هشت بایت به جلو حرکت می کند و دیگر به متغیر a اشاره نمی کند.

```
int a;
int *p;
p=&a;
p=p+٢;
```

تخصیص حافظه پویا

اشاره گرها زمانی نقش مهمی را در برنامه بازی می کنند که بخواهیم یک تکه از حافظه پویا (Heap) را در حین اجرای برنامه به داده ای اختصاص دهیم .ناحیه Heapفضای آزاد حافظه در دسترس است که به صورت پویا استفاده می شود، یعنی در حین اجرای برنامه در صورت نیاز اختصاص داده می شود و هنگامی که دیگر به آن احتیاج نباشد آزاد می شود .

در زبان پاسکال توابع new و dispose برای تخصیص و بازپس گیری حافظه هنگام کار با حافظه پویا استفاده می شوند

مثال.(Pascal)

```
Var ptr:^Integer;
Begin
New(Ptr); {allocate memory to an Integer data}
{ Use ptr }
... Dispose(Ptr); {deallocate memory from the data}
End.
```

در زبان C++ دو عملگر new و delete برای اختصاص حافظه پویا بکار می روند .

دستور new تعداد بایت های معینی از حافظه پویا را به داده اختصاص می دهد. مقدار فضای مورد نیاز با توجه به نوع داده اشاره گر واگذار می شود .

اگر دستور new موفق باشد اشاره گری به فضای اختصاص داده شده بر می گرداند و اگر ناموفق باشد مقدار NULL را برمی گرداند. بعد از آن می توان از متغیر اشاره گر برای دسترسی به داده درصورت نیاز استفاده کرد .

وقتی فضای حافظه پویا دیگر مورد نیاز نباشد باید به حافظه آزاد برگردانده شود. دستور deleteداده را از بین می برد و باعث می شود فضای حافظه آزاد شود تا برای مورد دیگری مجددا مورد استفاده قرار گیرد .

مثال .(C)برای اختصاص فضا به یک داده صحیح دستورات زیر نوشته می شود:

```
int *IntPtr;
IntPtr=new int;
if (IntPtr!= NULL)
```

* IntPtr =۵۵; اگرچه بطور ایده ال فضای پویای اختصاص یافته به برنامه بعد از اتمام برنامه باید آزاد شود اما همیشه این اتفاق نمی افتد. یک برنامه که کلیه فضائی را

که گرفته آزاد نمی کند دچار memory leaks است. نتیجه آن این خواهد بود که بعد از اجرای برنامه حافظه آزاد کمتری نسبت به قبل از اجرا خواهید داشت تا زمانی که کامپیوتر را راه اندازی مجددا کنید. راه حل اصلی، پس گرفتن حافظه ای تخصیص داده شده در انتهای برنامه است

اشاره گر به آرایه

معمولا یک اشاره گر به آرایه در برنامه زیاد مورد استفاده قرار می گیرد .اشاره گر به آرایه به اولین بایت آن اشاره می کند. هر عملگر محاسباتی که روی آن انجام شود اشاره گر را به جلو و عقب آرایه حرکت می دهد .

```
#include <iostream.h>
void main ()
{
int i[٥];
int *p;
p = i;
for ( int j = ٠; j<٥ ; j++,p++ ) {
   *p = j;
    cout << i[j];
   cout << "\n";
}
}
```

اشاره گر به رکورد

اشاره گر به رکورد یا ساختمان اجازه دسترسی به آن را مانند هر متغیر دیگری می دهد. تنها تفاوت در اینستکه چون رکورد یک نوع داده ترکیبی است هنگام دسترسی فیلد موردنظر در آن باید تعیین شود

مثال .(C)ساختمان account به صورت زیر تعریف شده است:

```
typdef struct account {
   float balance;
};
account *ptracout;
```

برای مقداردهی فیلد balance به صورت زیر باید عمل کرد:

```
ptraccount->balance=۲۰۰۰;
```

مثال Ptr .(Pascal)اشاره گری به نوع رکورد CustRec است .

```
Type
Ptr=^CustRec;
CustRec = Record
  Code:Integer;
  Name:String[۲۵];
  Address : String[۵۰];
End;
```

برای دسترسی به فیلد Name دستور زیر نوشته می شود:

```
Ptr^.Name := "Sara";
```

ليست پيوندي

لیست پیوندی ساختاری است که ترتیب خطی عناصر داده ای در آن توسط اشاره گرها تعیین می شود.

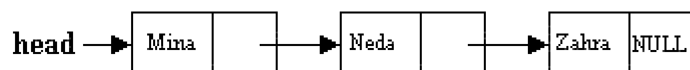
ليست پيوندي يک طرفه
ليست پيوندي حلقوي
ليست پيوندي دوطرفه
مقايسه ليست پيوندي و آرايه

لیست پیوندی یک طرفه

یک لیست پیوندی یک طرفه (Singly-linked list) دنباله ای از عناصر داده ای به نام گره (node)است که ترتیب خطی آنها توسط اشاره گرها تعیین می گردد .

عناصر لیست تنها می توانند به ترتیب از ابتدای لیست تا انتها مورد دسترسی قرار بگیرند. هر گره آدرس گره بعدی را شامل می شود که به این صورت امکان پیمایش از یک گره به گره بعدی فراهم می شود .

برای رسم لیست پیوندی گره ها به صورت مستطیل هائی پشت سرهم رسم می شوند که توسط فلش هائی بهم متصل شده اند .



مقدار ثابت NULL برای علامتگذاری انتهای لیست در اشاره گر آخرین گره ذخیره می شود .

لیست توسط یک اشاره گر Head که آدرس اولین گره لیست را در خود ذخیره می کند قابل دسترس است .بقیه عناصر توسط جستجوی خطی بدست می آیند .

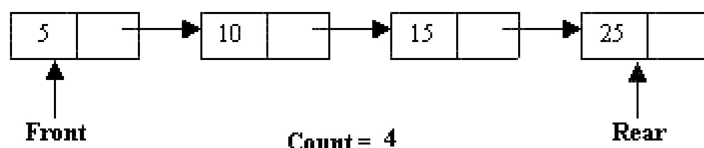
<u>پیاده سازي لیست پیوندي یک طرفه به زبان C++</u>

پیاده سازی لیست پیوندی یک طرفه

برای پیاده سازی لیست پیوندی ابتدا باید نوع داده یک گره و متغیرهای موردنیاز تعریف شوند که در زبان C می تواند به صورت زیر نوشته شود :

```
typedef int ItemType;
typedef struct Node {
  ItemType Info;
  Node * Next;
};
typedef Node * NodePtr;
NodePtr Front, Rear;
int Count;
```

در تعریف فوق ItemType نوع داده عناصر لیست را معین می کند که در مثال int درنظر گرفته شده است. ساختمان Node برای تعریف هر گره لیست است که دارای دو فیلد Info و Next است که به ترتیب عنصر داده ای گره و اشاره گر به گره بعدی را ذخیره می کنند. اشاره گر Front برای اشاره به ابتدای لیست در نظر گرفته شده است. گاهی دسترسی سریع به انتهای لیست موردنظر است، به همین دلیل ممکن است اشاره گر Rear را برای اشاره به انتهای لیست اضافه کنیم .متغیر Count تعداد گره های لیست را ذخیره می کند تا هروقت که احتیاج است بدانیم چه تعداد عنصر در لیست وجود دارد از آن استفاده کنیم .



در ابتدای برنامه متغیرهای Front و Rear باید برابر با NULL شوند تا یک لیست تهی ایجاد شود .

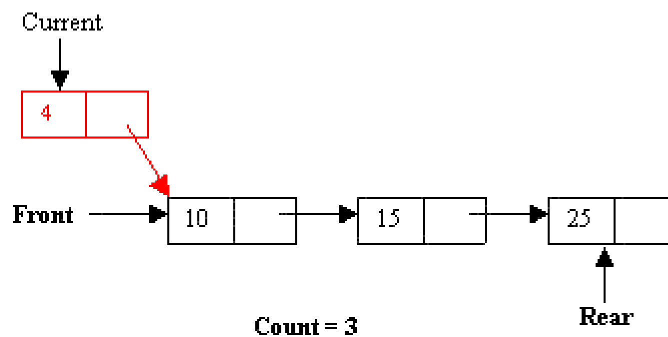
```
void CreateEmptyList(void)
{
  Front = NULL;
  Rear = Front;
  Count = ۰;
}
```

برای پیاده سازی یک لیست پیوندی، عملیات اصلی شامل موارد زیر هستند:

- درج : یک گره جدید را به ابتدا، انتها یا میان لیست اضافه می شود.
- حذف : یک گره از ابتدا، انتها یا میان لیست حذف می شود.
- جستجو : یک گره که شامل مقدار خاصی است در لیست جستجو می شود.

درج یک گره در ابتدای لیست

یک گره جدید می تواند در هر جائی از لیست اضافه شود؛ ابتدا، انتها یا میان لیست.
حالت زیر را درنظر بگیرید که می خواهیم گره جدید Current را به ابتدای یک لیست موجود اضافه کنیم .



اشاره گر Current به گره جدید اشاره می کند. فیلد Next این گره باید مقدار Front مقداردهی شود تا به اولین گره لیست اشاره کند. با اضافه شدن گره جدید به ابتدای لیست اشاره گر Front باید تغییر کند و به گره ابتدا که اکنون گره Current است اشاره کند . متغیر Count در انتها یک واحد اضافه می شود .

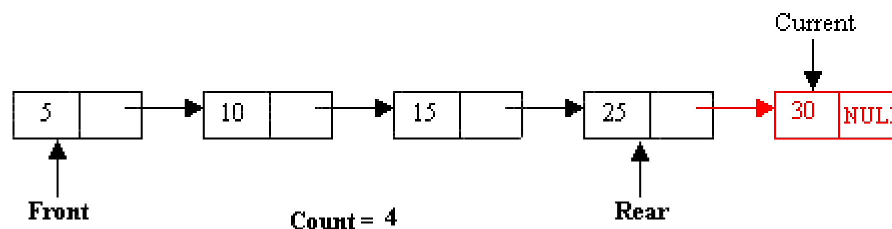
```
void InsertFirst( ItemType Item)
{
  NodePtr Current;
  Current = new Node;
```

```
  if (current == NULL)
    {
    cerr << "Memory allocation error!" << endl;
    exit(١);
    }
  current->Next = Front;
  current->Info = item;
  Front = current;
  if (Count == ٠)
    Rear = current;
  Count++;
}
```

در حالتی که گره جدید به ابتدای یک لیست خالی اضافه می شود(یعنی وقتی Count=٠ است) باید اشاره گر Rear هم تنظیم شود تا به گره جدید که اولین و آخرین گره لیست محسوب می شود اشاره کند .

درج یک گره در انتهای لیست

برای اضافه کردن یک گره در انتهای لیست گره Current باید بعد از آخرین گره لیست که آدرس آن در اشاره گر Rear نگهداری می شود درج شود .



فیلد Next آخرین گره به گره جدید باید اشاره کند . بعد از اضافه شدن گره در انتهای لیست اشاره گر Rear برابر با آدرس گره آخر می شود .

```
void InsertLast(ItemType & Item)
{
  NodePtr Current;
  current = new Node;
  if (current == NULL)
    {
    cerr << "Memory allocation error!" << endl;
    exit(١);
```

```
  }
  current->Next = NULL;
  current->Info = item;
  if (Count == ۰)
    Front = current;
  else
    Rear->Next = current;
  Rear = current;
  Count++;
}
```

در حالتی که گره جدید به انتهای یک لیست خالی اضافه می شود(یعنی وقتی Count=۰است) باید اشاره گر Front هم تنظیم شود تا به گره جدید که اولین و آخرین گره لیست محسوب می شود اشاره کند .

وقتی لیست مرتب است گره جدید باید در محلی اضافه شود که ترتیب گره های لیست حفظ شود. بنابراین ابتدا باید محل درج گره جدید در لیست پیدا شود سپس اشاره گرها تنظیم شوند .

حذف یک گره از ابتدای لیست

عمل حذف از درج ساده تر است. البته احتمال یک پیغام خطا وقتی که لیست تهی است وجود دارد. در ساده ترین حالت حذف از ابتدای لیست صورت می گیرد .آدرس اولین گره در متغیر اشاره گر Current ذخیره می شود. مقدار فیلد Info این گره در متغیر Itemنگهداری می شود. سپس فیلد Next گره اول برای اشاره به گره دوم لیست تنظیم می شود. حافظه گره ای که از لیست حذف شده است توسط دستور free آزاد می شود. در انتها از متغیر Count یک واحد کم می شود .

یک حالت استثنا وجود دارد وقتی که لیست تنها شامل یک گره است و با حذف آن Rearباید برابر با NULL شود .

```
void RemoveFirst(ItemType &Item)
{
  NodePtr Current;
  if (Count == ۰) {
    cerr << "ERROR: there is no item to remove in the list!" <<
endl;
    exit(۱);
  }
```

```
  Current = Front;
  Item = Current->Info;
  Front = Current->Next;
  Count--;
  if (Count == ۰)
     Rear = Front;
  delete Current;
}
```

جستجو در لیست

تا وقتی که لیست تهی نیست می توانیم داده معینی را در لیست جستجو کنیم .جستجو از ابتدای لیست آغاز می شود. اشاره گر Current برای جلو رفتن روی گره های لیست استفاده می شود و در ابتدا آدرس اولین گره در آن قرار می گیرد .در صورت وجود گره مورد نظر در لیست آدرس آن که در Current است برگردانده می شود .

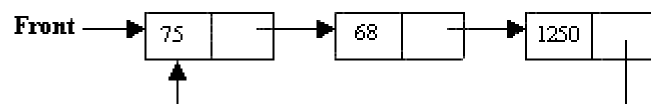
لیست پیوندی حلقوی

در مثال قبل احتیاج به یک گره آغازین و یک گره پایانی بود. ابتدای لیست توسط یک اشاره گر خاص (در مثال قبل Front) و انتهای لیست توسط گره ای که اشاره گر آن برابر با NULL است مشخص می شود. اگر مسئله به عملیاتی نیاز دارد که روی یک گره لیست پیوندی انجام می شود که مهم نیست گره اول یا آخرتوسط یک لیست پیوندی حلقوی حل می شود .

لیست حلقوی (circular list)لیست پیوندی است که آخرین گره آن به اولین گره لیست اشاره می کند. لیست حلقوی اشاره گر تهی ندارد و فیلد Next آخرین گره با آدرس گره اول مقداردهی می شود .

```
Rear->Next = Front;
```

عملیات درج، حذف و جستجو مانند لیست پیوندی است.



یکی از کاربردهای لیست حلقوی در اشتراک زمانی است که سیستم عامل لیستی از کاربران را دارد و به طور تناوبی به هر کاربر برش کوچکی از وقت پردازنده را

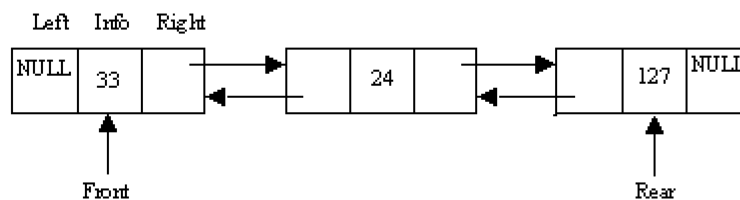
اختصاص می دهد. هر بار یک کاربر را انتخاب می کند و مقداری از وقت پردازنده را می دهد سپس به کاربر بعدی منتقل می شود

لیست پیوندی دوطرفه

در لیست یک طرفه فقط می توانیم در یک جهت پیمایش کنیم. گاهی پیمایش روی لیست از هردوطرف مورد نیاز است. بنابراین در هر گره به دو فیلد اشاره گر نیاز است؛ برای اشاره به گره بعدی و گره قبلی که اغلب اشاره گرهای Left و Rightنامیده می شوند. لیستی که شامل این نوع گره باشد را لیست پیوندی دوطرفه (doubly-linked list)می نامند. شکل زیر ساختار گره را نشان می دهد .

Left	Info	right

اگر لیست به صورت افقی ترسیم شود اشاره گرهای Right برای پیمایش لیست از چپ به راست (از ابتدا به انتها) استفاده می شوند. توسط اشاره گرهای Left می توان در صورت نیاز به گره قبلی برگشت. بنابراین امکان پیمایش لیست در هردو جهت وجود دارد .



همین طور که در شکل دیده می شود اشاره گر Left اولین گره و اشاره گر Right آخرین گره لیست NULL هستند که نشان دهنده ابتدای هر جهت می باشند

<u>پياده سازي ليست پيوندي دو طرفه به زبان C++</u>

مقایسه لیست پیوندی و آرایه

مزیـ اصلی آرایه نسبت به لیست پیوندی این است که آرایه امکان دسترسی تصادفی را می دهد و می توان به هر عنصر مستقیما توسط اندیس آن مراجعه کرد .به همین دلیل در یک آرایه مرتب می توانید از جستجوی باینری به جای جستجوی خطی استفاده کنید. اما با وجودیکه آرایه امکان دسترسی سریعتر به عناصر را می دهد در

عملیات درج و حذف ضعیف است و عملیات درج و حذف ممکن است باعث شیفت دادن عناصر زیاد دیگری شود. درحالیکه درج و حذف در لیست راحت تر است و درحقیقت تنها با تغییر اشاره گرها صورت می پذیرد. بنابراین احتمالا زمانی که عملیات درج و حذف زیاد انجام می شود روش بهتری است. تفاوت دیگر میزان فضای موردنیاز دو روش است. آرایه یک ساختمان داده ایستا است. هنگام تعریف، اگر اندازه آرایه را کوچک بگیریم از قدرت برنامه کاسته می شود بنابراین ناگزیریم بیشترین فضای ممکن را درنظر بگیریم که در نتیجه آرایه خیلی بزرگ تعریف می شود و حافظه زیادی مصرف می شود. درحالیکه لیست پیوندی ساختمان داده پویا است یعنی می تواند به راحتی رشد کند یا تحلیل برود یا تغییر کند .البته فضائی از حافظه برای ذخیره باید اشاره گرها صرف شود .

کلا لیست های پیوندی اغلب برای نمایش اطلاعاتی که ویژگی های زیر را دارند بکار می روند :

- تعداد کلی عناصر داده ای از قبل شناخته شده نیست.
- ممکن است عملیات اضافه و حذف زیاد انجام شوند.
- داده ها در یک طریق مرتب یا متوالی ذخیره شوند .
- با داده ذخیره شده به طور وسیع کار شود نه موردی.

صف

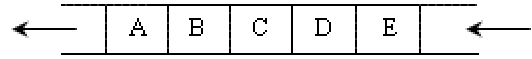
صف ساختمان داده ای است که کلیه عملیات اضافه از یک سر آن و کلیه عملیات حذف از انتهای دیگر آن انجام می پذیرد. صف در نرم افزارهایی صف انتظار را برای دسترسی به منبعی برقرار می کنند کاربرد دارد .

تعریف
نمایش صف
صف حلقوی
کاربردهای صف

تعریف

صف لیست مرتبی است که عناصر در انتهای آن (Rear) اضافه و از ابتدای آن (Front)حذف می شوند. به عبارت دیگر طول صف از انتهای آن افزایش و از ابتدای آن کاهش می یابد .

اولین عنصری که وارد صف می شود اولین عنصری است که از صف خارج می شود . بنابراین عناصر به همان ترتیبی که به صف اضافه می شوند از آن حذف می شوند .به همین دلیل به صف لیست FIFO (first in, first out)نیز گفته می‌شود .



نمایش صف

پیاده‌سازی صف با آرایه

صف را می توان توسط یک آرایه یک بعدی پیاده سازی کرد .به دو متغیر Front و Rearبرای مشخص کردن ابتدا و انتهای صف نیاز است .

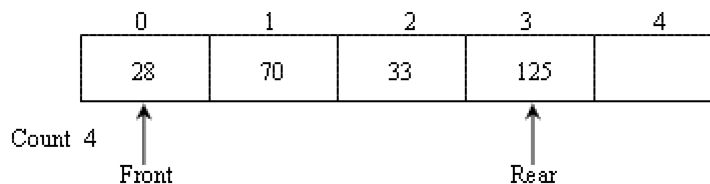
هر گاه عنصری به صف اضافه شود Rear يك گام به جلو حركت می كند و هر گاه كه عنصری را از صف حذف می شود Front يك واحد افزایش می یابد .

چون اندازه آرایه از قبل تعریف می شود، هنگام اضافه كردن عنصری به صف ابتدا باید اطمینان حاصل كرد كه هنوز ظرفیت پذیرش داده را دارد. اگر Rear برابر با ظرفیت كل آرایه شود صف پر درنظر گرفته می شود .

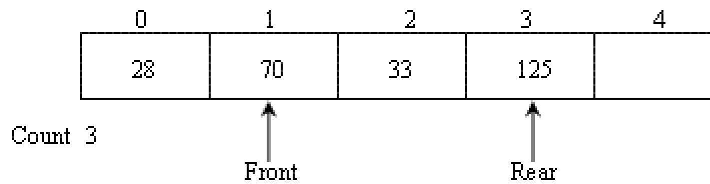
اگر ابتدا و انتهای صف برابر بودند (Front=Rear) یعنی صف خالی است. عمل حذف روی صف خالی انجام نمی گیرد .

طول صف یا تعداد عناصر موجود در صف برابر با Rear-Front+۱ است

مثال. در شكل زیر صف به ترتیب شامل عناصر ۲۸، ۷۰، ۳۳ و ۱۲۵ است. توجه كنید كه Front برابر با صفر است كه اندیس اولین داده صف یعنی عدد ۲۸ است. و Rearبرابر با ۳ است كه اندیس آخرین داده صف یعنی عدد ۱۲۵ است. طول صف در این مثال برابر با ٤ است .(Count=٤)

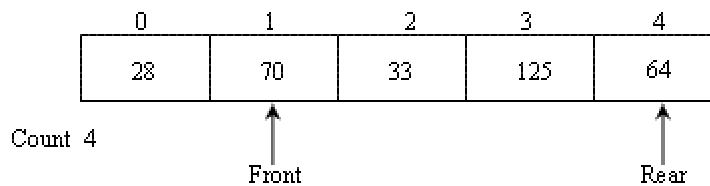


حذف از صف همیشه ازابتدا (Front) صورت می گیرد. بنابراین ۲۸ از صف حذف می شود و صف به صورت زیر درمی آید .



صف اکنون از اندیس ۱ تا اندیس ۳ می باشد، یعنی شامل اعداد ۷۰، ۳۳ و ۱۲۵ است. نیازی به پاک کردن عدد ۲۸ از صف نیست زیرا Front مشخص کننده اولین عنصر صف یعنی ۷۰ است .

اگر داده ٦٤ را به صف اضافه کنیم در انتهای صف اضافه می شود .



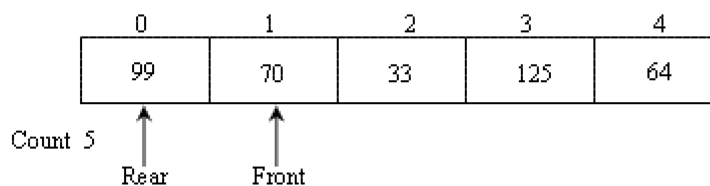
اکنون صف پر به نظر می رسد و اگر بخواهیم عددی را اضافه کنیم درصف جائی وجود ندارد. زیرا Rear= MaxSize . MaxSize حداکثر ظرفیت آرایه است که هنگام تعریف آرایه معین می شود

صف حلقوی

در مثال فوق هنگام اضافه کرن عدد ۹۹، تعداد عناصر صف کمتر از طول آرایه است ولی صف پر در نظر گرفته می‌شود. برای حل این مشکل یك روش انتقال عناصر صف به سمت ابتدای آن است تا فضای خالی همیشه در انتهای صف قرار گیرد. که روش مناسبی نیست. روش معمول استفاده از آرایه حلقوی است. یعنی دو انتهای آرایه متصل درنظر گرفته می شود .وقتی انتهای آرایه پر می شود عنصر بعدی در ابتدای آرایه، در صورت بلااستفاده بودن، درج می شود. این روش را صف حلقوی می نامند

<u>پیاده‌سازی صف حلقوی به زبانC++</u>

مثال. با فرض حلقوی بودن صف عنصر جدید ۹۹ در اندیس ۰ اضافه می شود.



وقتی Front=(Rear+۱)mod MaxSize باشد صف پر درنظر گرفته می شود.

در صف حلقوی اگر Front<Rear است طول صف برابر Rear-Front+۱ است.
درغیر اینصورت برابر با MaxSize-Front+Rear+۱_است

پیاده‌سازی صف با لیست پیوندی

در یک لیست پیوندی اگر درج در انتها و حذف از ابتدای آن انجام گیرد یک صف اجرا شده است. مزیت پیاده سازی صف توسط لیست پیوندی در این است که طول صف تنها محدود به حافظه دردسترس است

<u>پیاده‌سازی صف با لیست پیوندی به زبان C</u>

کاربردهای صف

صف معمولا در سیستم های عامل و نرم افزارهایی که صف انتظار برای دسترسی به منبعی را برقرار می کنند استفاده می شوند. سیستم عامل ممکن است صفی از پروسس هایی که منتظر اجرا روی CPU هستند یا صفی از کارهای که منتظر چاپ هستند را داشته باشد

پشته

پشته لیست خطی است که تنها از یک انتهای آن می توان به عناصر دسترسی پیدا کرد.
پشته برای نگهداری موقت داده ها در بسیاری از نرم افزارها کاربرد دارد .

تعریف
نمایش پشته
پشته چندگانه
کاربردهای پشته

تعریف

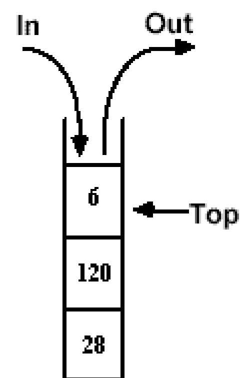
پشته لیست خطی است که عملیات حذف و اضافه تنها از يك طرف آن به نام Top صورت می گیرد. در هر لحظه فقط عنصر بالائی پشته قابل دسترس است .

آخرین عنصری که به پشته اضافه می شود اولین عنصری است که از آن حذف می شود. یعنی عناصر عکس ترتیب ورود از پشته خارج می شود به همین دلیل پشته یک ساختمان داده LIFO (last in, first out)محسوب می شود .

می توان پشته را ساختمان داده FILO (first in, last out)هم نامید زیرا اولین داده ای که در پشته ذخیره می شود آخرین داده ای است که خارج می شود .

عمل اضافه کردن عنصر جدیدی در پشته push و عمل حذف عنصری از پشته pop نامیده می شود. عملیات دیگری نظیر خواندن عنصر بالای پشته یا بررسی خالی بودن پشته نیز ممکن است در برنامه های مختلف به کاربیاید .

مثال. در زیر پشته ای از اعداد صحیح نشان داده شده است. عدد ۳٤ به بالای آن اضافه شده است



نمایش پشته

پیاده‌سازی پشته با آرایه

ساده‌ترین روش پیاده‌سازی پشته استفاده از یك آرایه یك بعدی و یک متغیر Top است .

```
const int MaxSize = ١٠٠;
ItemType Stack[MaxSize];
int Top;
```

Topاندیس عنصر بالای پشته Stack را نگه می دارد .Top=٠ دلالت بر خالی بودن پشته دارد .

برای Push کردن یک داده جدید به پشته، Topیکی اضافه می شود و داده جدید در آرایه در اندیس Top درج می کند .

```
void Push( const ItemType &Item)
{
if (Top == MaxSize){
  cerr << "ERROR: Cannot insert -- Stack is full" << endl;
```

```
  exit(١);
  }
else{
  Top++;
  Stack[top] = Item;
  }
}
```

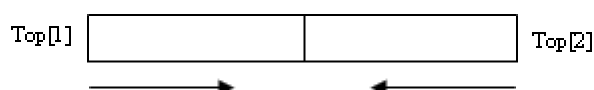
برای Pop کردن از پشته، داده ای که در اندیس Top قرار دارد برگردانده می شود و Topیک واحد کم می شود .

```
void Pop(ItemType & Item)
{
if (IsEmpty()){
  cerr << "ERROR: Cannot remove -- Stack is empty" << endl;
  exit(١);
  }
else{
  Item = Stack[Top];
  top--;
  }
}
```

<u>پیاده سازی پشته توسط آرایه در C++</u>

پشته چندگانه

اگر نیاز به دو پشته در برنامه باشد از يك آرايه يك بعدی استفاده می شود که [١]Top ابتدای پشته اول و [٢]Topابتدای پشته دوم است. پشته ها به سمت همدیگر رشد می‌کنند .

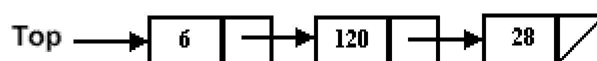


وقتی به n پشته احتیاج داریم. آرایه به n قسمت تقسیم می شود که هر قسمت به یک پشته اختصاص داده می شود. برای هر پشته [b[iپایین ترین مکان پشته و [t[i بالای

پشته i را نشان می دهد. اگر t[i]=b[i]باشد یعنی پشته i خالی است و اگر t[i]=b[i+۱]
یعنی پشته i ام پر شده است

پیاده سازی پشته با لیست پیوندی

چون پشته دنباله ای از عناصر داده ای است می توان آنرا در لیست پیوندی ذخیره کرد. اعمال درج و حذف از ابتدای لیست صورت می گیرند Top .آدرس اولین عنصر لیست را نگه می دارد .



پیاده سازی پشته توسط لیست پیوندی در C++

کاربردهای پشته

پشته در حل مسائل مختلفی کاربرد دارد که از جمله می توان به موارد زیر اشاره کرد :

- معکوس کردن ترتیب عناصر لیست
- تبدیل و ارزیابی عبارات
- ذخیره آدرس‌های برگشتي و متغیرهای محلی در توابع
- مسیر پرپیچ و خم
- برج هانوی

معکوس کردن ترتیب عناصر لیست

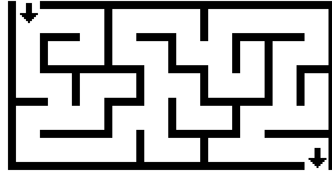
چون عناصر عکس ترتیب ورود از پشته خارج می شود پشته روش مناسبی برای معکوس کردن ترتیب عناصر یک لیست است. برای این کار کافی است ابتدا کلیه عناصر به ترتیب در پشته Push شوند سپس یکی یکی از پشته حذف می شوند. لیست خروجی حاصل عکس لیست اولیه است .

مثال. الگوریتم زیر ترتیب عناصر آرایه List با n عنصر را عکس می کند.

```
for (i:=۱ to n)
Push(List[i])
end for
for (i:=۱ to n)
Pop(List[i])
end for
```

مسیر پرپیچ و خم

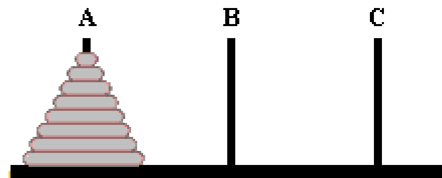
ماز (maze) مسیر پرپیچ و خمی است که حل کننده مسئله باید راهی را به بیرون پیدا کند. مسیر و موانع در ماز از قبل تعریف شده است. یکی از ساده ترین روش ها برای پیاده سازی ماز استفاده از کامپیوتر است؛ توسط يك آرايه دوبعدی n×m که صفر بودن سلولی در آن به معني بازبودن مسیر و یک بودن به معنی مانع است .



روش حل ماز به صورت سعی و خطا است. یک مسیر تصادفی انتخاب می شود اگر راه اشتباهی بود و در مسیر بسته گیرکردیم به عقب برمی گردیم و مسیر دیگری که قبلا طی نشده را امتحان می کنیم. برای برگشت به موقعیت قبلی نیاز است قبل از رفتن به موقعیت جدید موقعیت فعلی در پشته ذخیره شود .

برج هانوی

معمای برج هانوی (Tower of Hanoi) توسط ریاضیدان فرانسوی Edouard Lucasدر سال ۱۸۸۳ اختراع شد. سه میلهA ، Bو C را درنظربگیرید که روی میله Aتعداد n دیسک با اندازه های مختلف از بزرگ به کوچک روی هم قرار داده شده اند. هدف بازی برج هانوی این است که دیسک ها با استفاده از میله کمکی B به میله C منتقل شوند .



قوانین بازی به صورت زیر است:

- •درهربار تنها یک دیسک را می توان انتقال داد
- •تنها دیسک بالائی هر میله را می توان به میله دیگر منتقل کرد.
- •درهیچ زمانی نمی توان دیسک بزرگ تر را روی دیسک کوچک تر قرار داد

مثال. اگر تعداد دیسک ها ۳ باشد. ترتیب جابه جائی دیسک ها به صورت زیر می شود.

A->C, A->B, C->B, A->C, B->A, B->C, A->C

پیاده سازی با توابع بازگشتی

```
Tower ( n,A,B,C)
If (n=٠)
  exit
else
  Tower(n-١,A,C,B)
  A->C
  Tower(n-١,B,A,C)
end if
```

ارزيابی عبارات

عبارت

يك عبارت (expressions) از عملگرها(operators)، عملوندها (operamds)و پرانتز ساخته شده است. يك عبارت به شكل مي‌تواند نمايش داده شود :

- ميانوندی : (infix) عملگر بين عملوندهايش قرار مي‌گيرد(A+B)
- پسوندی : (postfix) عملگر بعد عملوندهايش قرار مي‌گيرد(AB+)
- پيشوندی : (prefix) عملگر قبل عملوندهايش قرار مي‌گيرد(AB+)

مثال.

Infix	PostFix	Prefix
١٦ / ٢	١٦ ٢ /	/ ١٦ ٢
٥ * (١٤ + ٢)	٥ ١٤ ٢ + *	* ٥ + ١٤ ٢
٥ * ١٤ + ٢	٥ ١٤ * ٢ +	+ * ٥ ١٤ ٢
(٤ + ٥) * (٢ - ٦)	٤ ٥ + ٢ ٦ - *	* + ٤ ٥ - ٢ ٦

روش معمول برای نوشتن یک عبارت روش میانوندی است. در یک عبارت میانوندی ترتیب اجرای ارزیابی عملگرها با توجه به الویت های قراردادی و پرانتزگذاری تعیین می شود. کامپایلر برای محاسبه یک عبارت میانوندی آنرا از شکل میانوندی به پسوندی تغییر می دهد، زیرا روش پسوندی نیازی به پرانتز گذاری ندارد و الویت هم مطرح نیست. عبارت میانوندی از چپ به راست پویش می شود، عملوندها در پشته قرار می گیرند و برای ارزیابی عملگر به تعداد لازم عملوند از پشته برداشته می شود و نتیجه

مجدد در پشته ذخیره می شود .به این ترتیب ارزشیابی عبارت پسوندی ساده تر از عبارت میانوندی انجام می گیرد

تبدیل عبارت میانوندی به پسوندی

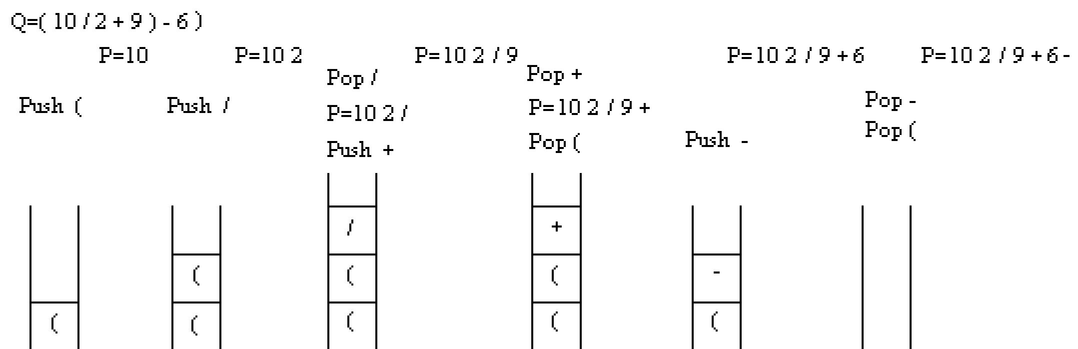
الگوریتم زیر عبارت میانوندی Q را به عبارت پسوندی Pتبدیل می کند .الگوریتم از پشته برای نگهداری موقت عملگرها و پرانتزهای باز استفاده می کند. در ابتدا یک پرانتز باز در پشته و یک پرانتز بسته در انتهای عبارت Q اضافه می شود. الگوریتم تا خالی شدن پشته تکرار می شود. عبارت Q از چپ به راست پیمایش می شود. هر عنصر عبارت اگر عملوند بود به P و اگر عملگر یا پرانتز باز بود به پشته اضافه می شود. اگر در عبارت به پرانتز بسته برسیم از پشته تا پرانتز باز را برداشنه و به P اضافه می کنیم. پرانتز باز هم از پشته برداشته می شود. وقتی به عملگری می رسیم، اگر الویت آن از الویت عملگر بالای پشته کمتری یا مساوی بود، عملگر بالای پشته را برداشته و به P اضافه می کنیم. این عمل را این قدر انجام مي دهيم تا زماني که عملگر جدید اولویت بیشتری از عملگر بالای پشته داشته باشد. سپس آنرا به پشته اضافه می کنیم .

الگوریتم تبدیل عبارت میانوندی به پسوندی

```
i:=j:=۰;
Q:=Q+')'; P=''; Push('(')
while (not IsEmpty())
  if (Q[i] is an Operand)
    P[j]=P[j]+Q[i]
    j:=j+۱;
  else if (Q[i] = '(' )
    push('(')
  else if (Q[i] = ')' )
    while ( Stack[top]<>'(' )
     Pop(x)
     P[j]=P[j]+x
     j:=j+۱;
    end while
    Pop(x)
  else if (Q[i] is an Operator)
    while (Precedence (Q[i]) <= Precedence (Stack[Top]))
     Pop(x)
     P[j]=P[j]+x
```

```
    j:=j+١
   end while
   Push(Q[i])
  end if
  i:=i+١
end while
```

مثال. مراحل تبدیل عبارت میانوندی Q به عبارت پسوندی توسط پشته.



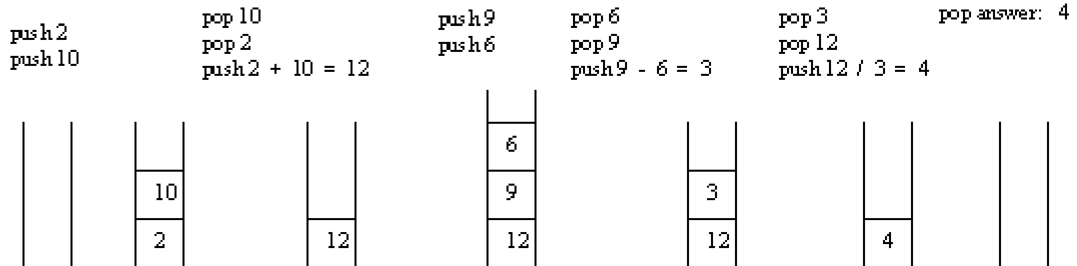
ارزیابی یك عبارت پسوندی

برای ارزیابی یک عبارت پسوندی، از چپ به راست عبارت پویش مي‌شود عملوندها در پشته اضافه می شوند. اگر در عبارت به عملگر برسیم دو عنصر از پشته برداشته عملگر روی آنها انجام شده نتیجه مجدد در پشته اضافه می شود .در انتها نتیجه عبارت در بالای پشته است .

الگوریتم ارزیابی عبارات پسوندی

```
i:=٠;
P:=P+')';
while ( P[i}<>')' )
  if (P[i] is an Operand)
    Push(P[i])
  Else if (P[i] is an Operator ⊕)
    Pop(B)
    Pop(A)
    Push(A⊕B)
  end if
  i:=i+١
end while
```

مثال. مراحل ارزیابی عبارت پسوندی P توسط پشته.



ارزیابی پرانتزهای يك عبارت

از پشته می توان برای بررسی میزان بودن پرانتزهای باز و بسته یک عبارت میانوندی نیز استفاده کرد. عبارت میانوندی از چپ به راست پویش می شود .پرانتز بازها در پشته اضافه می شوند. وقتی به پرانتز بسته رسیدیم از پشته یک پرانتز باز را بر می داریم، دراین زمان اگر پشته خالی باشد به این معنی است که تعداد پرانتزهای بسته بیشتر از پرانتز باز بوده است. در انتهای عبارت اگر پشته خالی باشد پرانتزها میزان بوده اند اگر پرانتز بازی در پشته باقی مانده باشد یعنی تعداد پرانتز بازها از پرانتز بسته ها بیشتر بوده است .

```
Balance:=True
i:=٠
while (Balance = True and i<=Length(Q))
  if (Q[i] = '(' )
    Push('(')
  else if (Q[i] = ')' )
    if (IsEmpty() )
      Balance = false
    else
      Pop(x);
    end if
  i:=i+١
end while
if (IsEmty() )
  Expression is Balanced
end if
```

تبدیل عبارات

الگوریتم تبدیل عبارت میانوندی به پسوندی

۱. عبارت کامل پرانتزگذاری می شود
۲. هر عملگر به سمت راست پرانتز باز خود منتقل می شود
۳. پرانتزها حذف می شوند

تبدیل عبارت میانوندی به پیشوندی

۱. عبارت کامل پرانتزگذاری می شود
۲. هر عملگر به سمت چپ پرانتز بسته خود منتقل می شود
۳. پرانتزها حذف می شوند

تبدیل عبارت پسوندی به پیشوندی

۱. ابتدا عبارت پسوندی را تبدیل به میانوندی می کنیم
۲. عبارت میانوندی حاصل را به پیشوندی تبدیل می کنیم

نکته. در تبدیل عبارات ترتیب عملوندها تغییر نمی‌کند

بازگشتی

تابع بازگشتی تابعی است که در بدنه اش دستوری دارد که خودش را فراخوانی می کند. توابع بازگشتی برای نگهداری حالت قبلی خود از پشته مکرر استفاده می کنند .

Call stack
عملکردهایcall stack
ساختارcall stack
بازگشتی
بازگشتی در مقایسه با غیر بازگشتی
سرریزی پشته
نمونه هائی از توابع بازگشتی

تابع یا زیربرنامه بخش جداگانه ای از کد برنامه است که می تواند توسط یک نام صدا زده شود. به هر زیربرنامه ممکن است پارامترهایی ارسال شود و هر تابع می تواند یک

مقدار را برگرداند. وقتی تابعی فراخوانی می شود مقدار پارامترهای تابع در جائی باید ذخیره شود تا تابع بتواند به آنها دسترسی پیدا کند .

Call stack

اکثر کامپایلرها برای فراخوانی و برگشت از زیربرنامه call stack را پیاده سازی می کنند .Call stack یا run-time stack یک پشته است که اطلاعاتی درباره زیربرنامه فعال یک برنامه را نگهداری می کند. زیربرنامه فعال زیربرنامه ای است که فراخوانی شده است اما هنوز اجرایش تمام نشده است .

وقتی زیربرنامه ای فراخوانی می شود، قبل از اینکه کنترل اجرای برنامه به آدرس زیربرنامه پرش کند آدرس دستورالعمل بعدی (دستورالعملی که درحافظه بعد از دستور فراخوانی قرار دارد) درجائی باید ذخیره شود که هنگام برگشت از زیربرنامه از آن استفاده می شود. این آدرس را آدرس برگشتی (return addresses) می نامند .

معماری که بر اساس پشته است آدرس برگشتی را به عنوان نقطه برگشت در پشته اضافه می شود. هر بار که زیربرنامه ای فراخوانی می شود آدرس برگشتی در پشته pushمی شود. هنگام برگشت از زیربرنامه آدرس برگشتی از پشته pop شده و کنترل برنامه به آن آدرس پرش می کند و اجرای برنامه از بعد از دستور فراخوانی ادامه پیدا می کند .

به دلیل استفاده از پشته یک زیربرنامه می تواند خودش یا زیربرنامه های دیگر را صدا بزند.

در زبان های سطح بالا call stack معمولا از برنامه نویس مخفی است .درمقابل در زبان اسمبلی نیاز است خود برنامه نویس با پشته درگیر شود .

عملکردهایcall stack

هدف اصلی یک call stack نگهداشتن آدرس برگشتی هر زیربرنامه فعال است .اما بسته به زبان، سیستم عامل و محیط سخت افزاری ممکن است عملکردهای اضافی دیگری هم داشته باشد نظیر :

ذخیره آدرس های برگشتی

برای هر برنامه یک پشته درنظرگرفته می شود. وقتی زیربرنامه ای در برنامه فراخوانی می شود آدرس دستورالعمل بعد از عبارت فراخوانی (آدرس برگشتی) در

پشته قرار می گیرد. زیربرنامه می تواند به صورت بازگشتی باشد هر بار که زیربرنامه خودش را صدا می زند آدرس برگشتی در پشته ذخیره می شود .

ذخیره متغیرهای محلی

متغیرهائی که درون زیربرنامه تعریف می شوند متغیرهای محلی نامیده می شوند. متغیرهای محلی تنها درون زیربرنامه فعال شناخته شده هستند و بعد از اتمام زیربرنامه مقدار آنها در حافظه باقی نمی ماند. اغلب مناسب است که فضائی در پشته به آنها اختصاص داده شود که سریع تر از تخصیص فضای آزاد heap به آنها است. هر زیربرنامه فعال فضای جداگانه خودش را در پشته برای داده های محلی دارد .

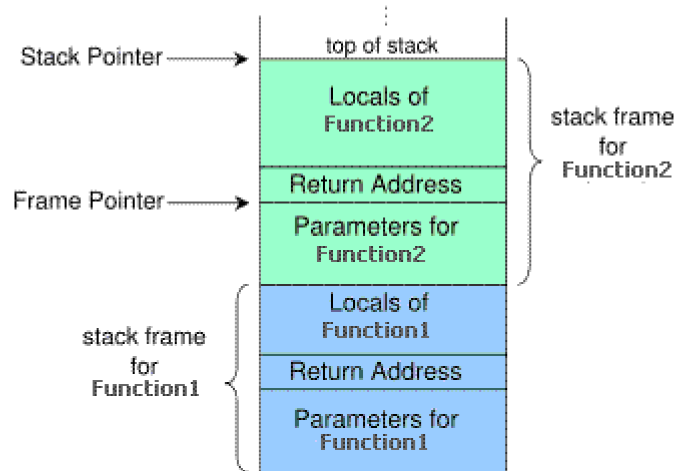
ارسال پارامتر

مقادیر پارامترهای موردنیاز زیربرنامه ها هنگام فراخوانی به آنها داده می شوند. معمولا فضای از call stack برای ذخیره مقدار این پارامترها اختصاص داده می شود. هر فراخوانی به زیربرنامه مقادیر مختلفی از پارامترها را خواهد داشت و فضای جداگانه ای در پشته به آنها داده می شود

. ساختارcall stack

یک call stack از stack frame ها یا activation record ها تشکیل شده است. فریم پشته اطلاعات زیربرنامه را نگه می دارد. هر فریم پشته مربوط به یک فراخوانی زیربرنامه ای است که هنوز تمام نشده است.

مثال. فرض کنید تابع function۲ اکنون در حال اجرا است و توسط function۱ فراخوانی شده است. وضعیت پشته می تواند به شکل زیر باشد



فریمی که در بالای پشته است مربوط به زیربرنامه ای است که اکنون در حال اجرا است. هر فریم ممکن است دربرگیرنده متغیرهای محلی، آدرس برگشتی و مقدار پارامترهای زیربرنامه باشد. فریم های پشته هم اندازه نبوده و زیربرنامه های مختلف فریم های متفاوتی دارند .

پشته توسط ثبات stack pointer دسترسی می شود که بالای پشته را مشخص می کند .

بازگشتی

بازگشتی اجازه بیان راه حل یک مسئله را به طور مختصر و مفید می دهد .مسئله ای که به صورت بازگشتی حل می شود باید بتواند به مسائل کوچک تر تقسیم بشود و حل مسائل کوچک به همان روش مسئله بزرگ قابل انجام باشد. مسئله کوچک تر به مسئله کوچک تری شکسته می شود تا سرانجام یه کوچک ترین اندازه مسئله برسد که base caseنامیده می شود که می تواند بدون استفاده از بازگشتی حل شود

یک مثال متعارف الگوریتم بازشگتی ضابطه تابع فاکتوریل f(n) است :

$$f(n) = \begin{cases} 1 & \text{if } n = 0 \\ n \times f(n-1) & \text{if } n > 0 \end{cases}$$

مقدار تابع برای f(۳) به صورت زیر محاسبه می شود:

f(۳) = ۳ * f(۳ – ۱)
= ۳ * f(۲)
= ۳ * ۲ * f(۲ – ۱)

= ۳ * ۲ * f(۱)
= ۳ * ۲ * ۱ * f(۱ – ۱)
= ۳ * ۲ * ۱ * f(۰)
= ۳ * ۲ * ۱ * ۱
= ٦

یک الگوریتم بازگشتی توسط تابع بازگشتی پیاده سازی می شود. تابع بازگشتی (recursive function)تابعی است که خودش را فراخوانی می کند .

تابع بازگشتی مشابه توابع دیگر کار می کند. برای هر فراخوانی بازگشتی یک فریم جدید از تابع در پشته ایجاد می شود .

مثال .(C)تابع بازگشتی محاسبه فاکتوریل یک عدد صحیح.

```
int Factorial (int x)
{
 if (x<= ۱)
   return ۱;
   return x * Factorial (x-۱);
}
```

همان تابع به صورت غیر بازگشتی به صورت زیر نوشته می شود. توجه کنید که به د و متغیر موقت نیاز دارد .

```
int Factorial (int x)
{
 int i, temp;
 for ( i=۱; i<=x; i++)
   temp*=i;
 return temp;
}
```

مثال .(C)تابع دیگر که زیبائی بازگشتی را بیشتر نشان می دهد الگوریتم اقلیدسی برای محاسبه بزرگترین مقسوم علیه مشترک (greatest common divisor) دو عدد صحیح است. الگوریتم بازگشتی آن در زبان C به صورت زیر است :

```
int GCD (int x, int y)
{
 if (y == ۰)
```

```
    return x;
  else
    return GCD (y, x % y);
}
```

الگوریتم غیر بازگشتی آن به صورت زیر است. مشاهده می شود که الگوریتم غیر بازگشتی احتیاج به یک متغیر موقت دارد و حتی با دانستن الگوریتم اقلیدسی درک فرآیند تابع مشکل است .

```
int GCD(int x, int y)
{
  while (y != ۰) {
    int r = x % y;
    x = y;
    y = r;
  return x;
}
```

هر تابع بازگشتی می تواند توسط یک پشته به تابع غیر بازگشتی تبدیل شود .علت اینکه توابع بازگشتی ممکن به سختی ردیابی شوند احتمالا نداشتن دانش کافی درباره نحوه کار پشته است

بازگشتی در مقایسه با غیر بازگشتی

برای اینکه بازگشتی موفق باشد مسئله نیاز است یک زیرساختار بازگشتی داشته باشد. راه حل بعضی از مسائل به طور ذاتی بازگشتی است چون احتیاج به نگداری حالت قبلی دارند. الگوریتم پیمایش درخت(tree traversal)، تابع اکرمن (Ackermann)و الگوریتم های تقسیم و غلبه مانند مرتب سازی سریع (Quicksort)همگی به صورت بازگشتی هستند. همه این الگوریتم ها می توانند به صورت غیربازگشتی با کمک پشته هم پیاده شوند اما نیاز به پشته مزیت راه حل غیر بازگشتی را از بین می برد .

تابع غیر بازگشتی احتمالا در عمل کمی سریعتر از نسخه بازگشتی آن اجرا می شود چون تابع غیر بازگشتی سربار فراخوانی تابع (function-call) را به اندازه تابع بازگشتی ندارد و این سربار در بعضی زبان ها نسبتا بالا است .

یک دلیل دیگر برای ترجیح غیربازگشتی به بازگشتی این است که فضای پشته قابل دسترس کمتر از فضای قابل دسترس در حافظه آزاد heap است. و الگوریتم های بازگشتی تمایل به فضای پشته بیشتری نسبت به غیر بازگشتی دارند .

سرریزی پشته

وقتی اشاره گر پشته به انتهای پشته می رسد پشته سرریز می شود. دلیل معمول سرریزی پشته فراخوانی مکرر یا تعداد زیاد متغیرهای محلی توابع بازگشتی است. اگر یک تابع بی نهایت بار خودش را صدا بزند در هر فراخوانی یک فریم پشته اضافه می شود و در یک نقطه پشته دیگر جا ندارد و سرریز می شود و خطای stack overflowرخ می دهد .

نمونه هائی از توابع بازگشتی

مثال .(C)تابع بازگشتي ضرب.

```
int Mul (int a , int b)
{
  if (b == ١)
    return a;
  else
    return a+Mul (a,b-١);
}
```

مثال .(Pascal)تابع بازگشتی فیبوناچی.

```
Function Fibonacci ( n: Integer ):Integer;
Begin
  If (n=١) or (n=٢) Then Fib:=١
  Else Fibonacci:= Fibonacci (n-٢)+ Fibonacci (n-١);
End;
```

مثال .(Pascal)تابع بازگشتي اكرمن (Ackermann's function) تابعی است که مقدار آن به سرعت رشد می کند.

```
Function Ackerman ( a,b: Integer ):Integer;
Begin
  If (a<٠) and (b<٠) Then Ackerman:=٠
  Else If a=٠ Then Ackerman:=b+١
  Else If b=٠ Then Ackerman:= Ackerman (b-١,١)
  Else Ackerman:= Ackerman (a-١,Ack(a,b-١));
End;
```

گراف

تعاریف گراف

یک گراف شامل دو مجموعه است؛ مجموعه غیر تهی از گره ها یا رئوس (vertex) و مجموعه ای از یال ها (edge) که راس ها را به هم متصل می کنند .

مثال. می توان شهر های یک کشور را رئوس و جاده های بین آن ها را یال های یک گراف تصور کرد .

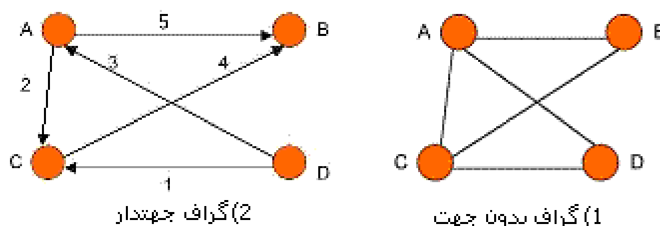
به هر راس یا هر یال گراف نامی اختصاص داده می شود .

یک گراف تهی (null graph) گرافی است که تنها شامل راس است و مجموعه یال های آن تهی است یعنی یالی ندارد .

جهت

یک گراف می تواند به دو شکل جهتدار (directed)یا غیرجهتدار (undirected) باشد .

یک گراف جهتدار گرافی است که جهت هر یال در آن تعیین شده است. در گراف جهتدار ترتیب رئوس در هر یال اهمیت دارد و یال ها با پیکان هائی از راس ابتدا به راس انتها رسم می شوند. در گراف غیرجهتدار می توان در هر دو جهت بین راس ها حرکت کرد و ترتیب راس های یال اهمیت ندارد .



1) گراف بدون جهت 2)گراف جهتدار

وزن

یال های گراف می توانند وزن دار (weighted)یا بدون وزن (unweighted) باشند. گرافی که یال های آن وزن باشد گراف وزن‌دار نامیده می شود. وزن می تواند نشان دهنده هزینه، مسافت، زمان یا هر مشخصه دیگری از یال باشد .

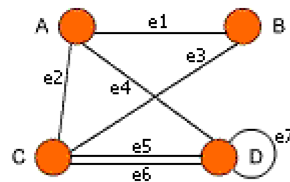
مجاورت

هر یال بوسیله یک جفت راس مشخص می شود. دو راسی که توسط یک یال به هم متصل می شوند را رئوس مجاور (adjacent) و یال را یک لبه تلاقی (incident) دو آن رو راس می نامند .

حلقه

یک حلقه (loop) یالی است که یک راس را به خودش متصل می کند. به عبارت دیگر راس ابتدا و انتهایش یکسان باشد .

مثال. در شکل زیر یال e۷ یک حلقه روی راس D است



یال های موازی

یال های موازی (parallel edges) یا چندگانه یال‌هایی هستند که رئوس یکسان را بهم مرتبط می‌کنند .

گرافی که دارای یال های موازی باشد را گراف چندگانه (multigraph) می نامند.

مثال. در گراف چندگانه شکل قبل یال های e۵ و e٦ که رئوس C و D را به هم متصل می کنند موازی هستند

گراف ساده

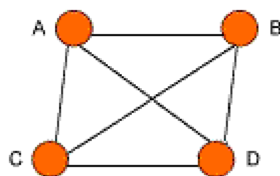
گراف بدون یال موازی و حلقه را گراف ساده (simple graph) می‌نامند. گراف جهتدار را وقتی ساده می گویند که یال موازی نداشته باشد .

حداکثر تعداد یال‌ها در يك گراف جهتدار با n راس برابر است با .n×(n-۱)

حداکثر تعداد یال‌ها در يك گراف غيرجهتدار با n راس برابر است با .n×(n-۱)/۲

گراف کامل

یک گراف کامل (complete graph) گراف ساده ای است که هر جفت راس آن مجاور باشند یعنی هر از راس به کلیه راس های دیگر یالی وجود داشته باشد .



درجه

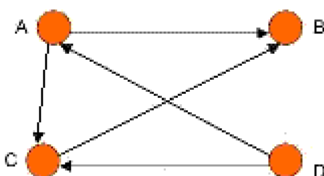
درجه (degree) هر راس توسط تعداد یال های متلاقی با راس مشخص می شود .

در گراف جهتدار درجه ورودی (indegree) یک راس تعداد یال هائی است که به آن راس وارد شده اند و درجه خروجی (outdegree) یک راس تعداد یآل هائی است که از آن راس خارج شده اند .

راس منبع راسی است که درجه خروجی آن مثبت و درجه ورودی آن صفر باشد.

راس چاه راسی که که درجه ورودی آن مثبت و درجه خروجی آن صفر باشد.

مثال. در گراف زیر درجه خروجی راس A دو و درجه ورودی آن یک است.. راس D منبع و راس B چاه است .



درجه گراف برابر درجه ماکزیمم رئوس گراف است.

گرافی که کلیه راس های آن از یک درجه باشد گراف منتظم (regular graph) نامیده می شود. گراف مکعب گراف منتظم درجه ۳ است .

دريك گراف مجموع درجات كليه رئوس همواره عددی زوج است

مسیر

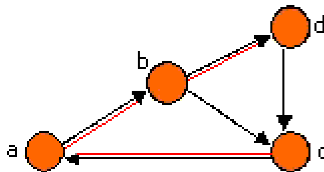
یک مسیر (path)در گراف یک گذر از راس های متوالی در امتداد یک سری از یال ها است. راس انتهای یک یال راس ابتدای یال بعدی در توالی محسوب می شود .

طول مسیر تعداد یال های مسیر است که در طول مسیر طی می شود. يك مسير با طول nدارای n+۱ راس و n یال است. در یگ گراف وزن دار طول مسیر برابر مجموع وزن‌های یال‌های مسیر است .

دوراس را متصل (reachable) می‌گویند اگر مسیری بین آنها وجود داشته باشد .

يك مسير (simple path) ساده مسیری است که همه رئوس آن بجز احتمالا راس شروع و پایان تکراری نباشد .

مثال. در شکل زیر یک مسیر نشان داده شده است که از راس C آغاز و به راس D ختم می شود .



دور

یک دور (cycle) مسیر ساده ای است که راس شروع و پایانی آن یکی باشد .

گراف ساده جهتداری که دارای دور نیست را غیر مدور (acyclic) می نامند.

یک دور در گراف ساده بدون جهت حداقل شامل سه یال متفاوت است که هیچ راسی در آن تکراری نیست بجز راس شروع و پایان .

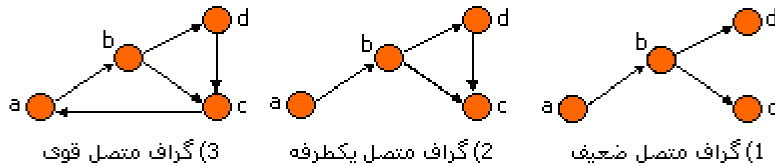
اتصال

یک گراف غیرجهتدار متصل یا همبند (connected) گفته می شود اگر مسیری بین هر جفت راس آن وجود داشته باشد. یعنی هر دو راس آن متصل باشند .

در گراف جهتدار چون جهت باید درنظر گرفته شود اتصال پیچیده تر است. ممکن است راس a به b متصل باشد ولی مسیری از راس b به a وجود نداشته باشد .

در گراف جهتدار ساده سه حالت برای اتصال وجود دارد:

- متصل ضعیف. (weakly connected) یک گراف متصل ضعیف گرافی است که اگر جهت گراف ندیده گرفته شود متصل است.
- متصل یکطرفه. (unilaterally connected) یک گراف متصل یکطرفه گرافی است که حداقل یک راس آن به هر راس دیگری متصل باشد.
- متصل قوی. (strongly connected) یک گراف متصل قوی گرافی است که هر جفت راس متصل باشد.



یک گراف ساده متصل بدون دور را درخت (tree) می‌نامند. درخت گرافی است که فقط يك مسير بين هر دو راس آن وجود دارد .

یک درخت با يك n راس است دارای n-۱ یال باشد

نمایش گراف

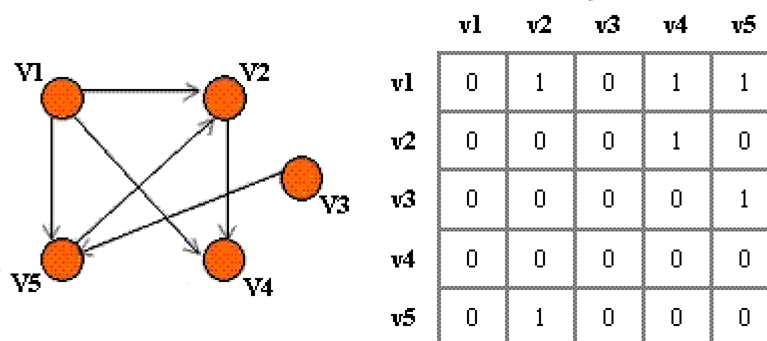
راه های متعددی برای نمایش گراف در کامپیوتر وجود دارد دو ساختمان داده پایه ای که برای نمایش گراف استفاده می شوند ماتریس مجاورت و لیست مجاورتی هستند .

ماتریس مجاورت

ماتریس مجاورت (adjacency matrix) گراف G با n راس (که رئوس آن به ترتیب از v۱ تا vn نامگذاری شده است) یک ماتریس بیتی n×nبا نام A است که در آن :

درایه aij برابر با ۱ است اگر یالی از vi به vj وجود داشته باشد
درایه aij برابر با ۰ است اگر یالی از vi به vj وجود نداشته باشد

مثال. گراف جهتدار زیر را با پنج راس درنظر بگیرید. ماتریس مجاورت آن در سمت راست نشان داد شده است.



	v1	v2	v3	v4	v5
v1	0	1	0	1	1
v2	0	0	0	1	0
v3	0	0	0	0	1
v4	0	0	0	0	0
v5	0	1	0	0	0

همانطور که در شکل دیده می شود اگر یعنی دو راس مجاور باشند در موقعیت متناظر در ماتریس باید ۱ باشد. برای مثال از رئوس مجاور با v۱ راس های v٤ ، v۲و v۵ هستند بنابراین در سطر اول در ستون های دوم، چهارم و پنجم باید ۱ قرار بگیرد .

ماتریس مجاورت برای یك گراف بدون جهت متقارن است.

در یک گراف غیر جهتدار درجه راس vi برابر با مجموع عناصر سطر iام در ماتریس مجاورت است. و در یك گراف جهتدار درجه خروجی راس vi برابر مجموع عناصر سطر i ام و درجه ورودی آن برابر مجموع عناصر ستون iام در ماتریس مجاورت است .

فضای مورد نیاز در روش ماتریس مجاورت برای نمایش یك گراف با مجموعه رئوس Vبرابر $O(|V|^۲)$ است. اگر تعداد یال‌های گراف کم باشد می‌توان آنرا به صورت ماتریس اسپارس نمایش داد .

قضیه. اگر A ماتریس مجاورتی گراف G باشد، درایه aij در ماتریس A^k تعداد مسیرهای با طول k از راس vi به vj را نشان می دهد .

مثال. با ضرب ماتریس مجاورت مثال قبل در خودش ماتریس $A^۲$ به صورت زیر بدست می آید. عدد ۱ در سطر i و ستون j ماتریس نشان دهنده وجود مسیری با طول ۲ از راس vi به vj در گراف است .

	v1	v2	v3	v4	v5
v1	0	1	0	1	0
v2	0	0	0	0	0
v3	0	1	0	0	0
v4	0	0	0	0	0
v5	0	0	0	1	0

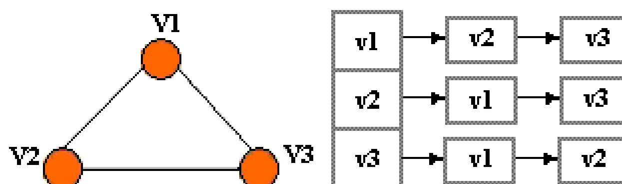
با توجه به ماتریس فوق می توان گفت که از راس v۱ با دو حرکت می توانیم به راس v۲بروریم. اگر گراف را بررسی کنیم می شویم که از v۱ به v۵ و سپس به v۲می توانیم برویم. یعنی مسیری با طول ۲ از راس v۱ به v۲ وجود دارد .

مزیت اصلی نمایش ماتریس در این است که محاسبه مسیر ها و دورها بسادگی توسط عملیات ماتریسی قابل انجام است. مجاورت بین دو راس با پیچیدگی زمان O(۱)تعیین می شود و اجازه رسم حلقه در گراف را می دهد. اشکال آن این است که از جنبه ظاهری گراف دور است و خواصی که بسادگی در شکل گراف نمایان است توسط ماتریس به سختی قابل رویت است. علاوه بر این ماتریس همجواری یال‌های موازی را نشان نمی‌دهد .

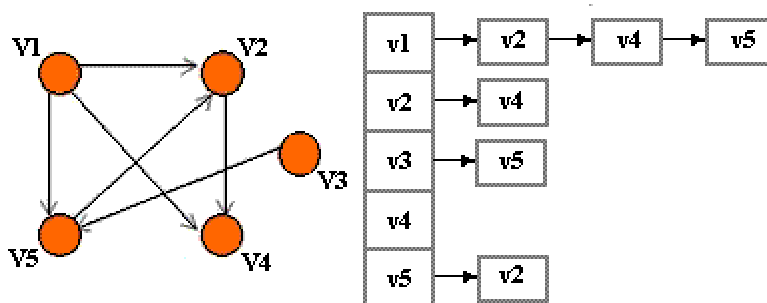
لیست مجاورت

لیست مجاورتی (adjacency list) فرم دیگر نمایش گراف در کامپیوتر است .این ساختمان داده شامل لیستی از کلیه رئوس گراف است. برای هر راس یك لیست پیوندی وجود دارد که گره های آن رئوس مجاور راس را دربر می گیرند. به عبارت دیگر لیست i حاوی رئوسی است که مجاور راس vi است .

مثال. گراف غیرجهتدار زیر را درنظر بگیرید. لیست مجاورتی آن در سمت راست آمده است:



مثال. گراف جهتدار زیر را درنظر بگیرید. لیست مجاورتی آن در سمت راست آمده است:



درجه هر راس در یك گراف غیر جهتدار با شمارش تعداد گره‌های لیست پیوندی مربوط به راس درلیست مجاورتی تعیین می‌شود. در یك گراف جهتدار درجه خروجی هر راس با شمارش تعداد گره‌های لیست پیوندی مربوط به آن بدست می آید .

اگر تعداد یال‌ها در گراف کم باشد این روش بهتر از ماتریس مجاورتی است .فضای مورد نیاز برای نمایش یك گراف با مجموعه رئوس V و مجموعه یال های E به روش لیست مجاورت برابر $O(|E|+|V|)$ می‌باشد .

لیست مجاورتی به روشنی طبیعت مجاورتی رئوس گراف را نشان می دهد و اغلب زمانی استفاده می شود که گراف دارای تعداد یال های نسبتا متعادلی باشد .

لیست مجاورتی معکوس

لیست مجاورتی معکوس مشابه لیست مجاورتی ساخته می شود با این تفاوت که در لیست پیوندی i رئوسی اضافه می شود که از آنها به راس vi یالی وارد شده باشد. تعداد گره های لیست پیوندی i ام درجه ورودی راس vi را نشان می دهد .

برای گراف غیرجهتدار لیست مجاورتی و لیست مجاورتی معکوس مشابه می شود.

پیمایش گراف

هدف از پیمایش این است که کلیه رئوسی که از طریق یک راس قابل دسترس هستند را بدست آوریم. دو روش معروف برای پیمایش وجود دارد؛ جستجوی اول عمق (Deep First Search)و جستجوی اول سطح . (Breadth First Search)

جستجوی اول عمق

در جستجوی اول عمق پیمایش از یک راس آغاز می شود. هر راس که پردازش یا اصطلاحا ملاقات می شود یکی از رئوس مجاور آن که قبلا ملاقات نشده است انتخاب می شود و پیمایش با ملاقات راس مجاور ادامه پیدا می‌کند. اگر راس مجاوری وجود نداشت که قبلا ملاقات نشده باشد يك سطح به عقب برمی گردد .

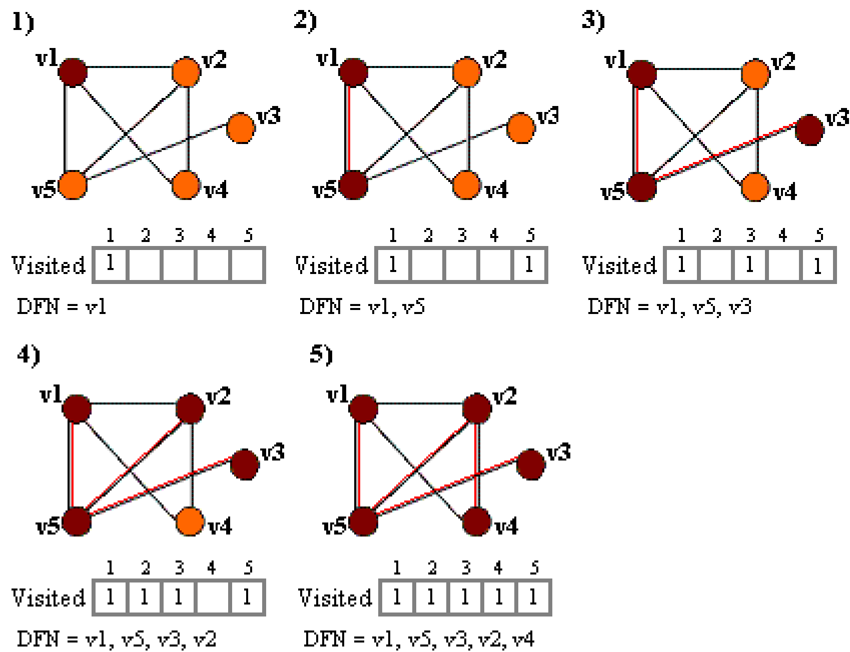
الگوریتم بازگشتی جستجوی اول عمق به صورت زیر است. آرایه یک بعدی Visited تعیین می کند آیا راسی قبلا ملاقات شده است یا خیر. اگر راس vi ملاقات شود Visited[i]برابر با یک می شود .

```
DFS (int v)
{
  int w
  Visited[v]:=١
  For (each vertex w adjacent to v)
    If (not visited[w]) then
      DFS(w)
    End if
  End For
}
```

ترتیب ملاقات رئوس را DFN می‌نامند .

برای گراف G با n راس و m یال ، مرتبه اجرائی الگوریتم وقتی گراف توسط ماتریس مجاورتی نمایش داده شده باشد $O(n^2)$و اگر از لیست مجاورتی استفاده شود O(m) است .

مثال. مراحل اجرای DFS(v١) برای یک گراف در شکل زیر نشان داده شده است .



جستجوی اول سطح

در جستجوی اول سطح پیمایش از یک راس آغاز می شود. آن راس و کلیه رئوس مجاورش ملاقات می شود سپس پیمایش از راس مجاور ادامه پیدا می کند .

الگوریتم جستجوی اول سطح به صورت زیر است. آرایه Visited برای تعیین رئوس ملاقات شده بکار می رود. از یک صف برای نگهداشتن رئوس مجاور استفاده می شود. هر بار که راسی ملاقات می شود کلیه رئوس مجاور آن در صف اضافه می شود. پیمایش از راسی که از صف برداشته می شود ادامه پیدا می کند .

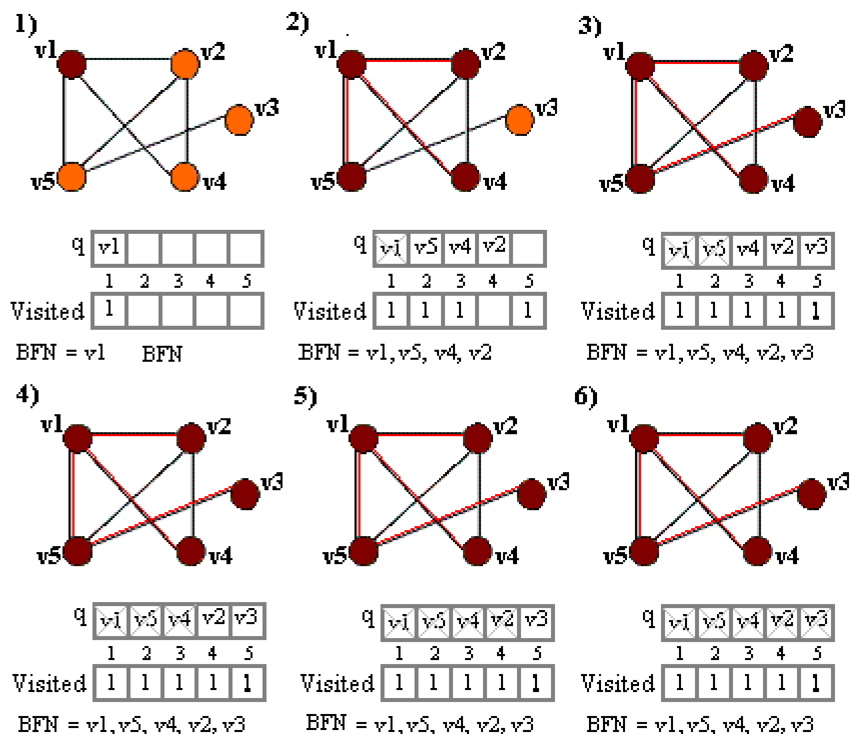
```
BFS (int v)
{   int w
  Queue q

  Visited[v]:=١
  CreateQueue(q)
  AddQueue(q, v)
  While (not EmptyQueue(q))
    DeleteQueue(q,v)
    For (all vertex w adjacent to v)
      If (not visited[w]) then
        AddQueue(q,w)
        Visited[w]:=١
      End if
```

```
  End For
 End while
}
```

برای گراف G با n راس و m یال ، مرتبه اجرائی الگوریتم وقتی گراف توسط ماتریس مجاورتی نمایش داده شده باشد $O(n^2)$ و اگر از لیست مجاورتی استفاده شود $O(m)$ است .

مثال. مراحل اجرای BFS(v۱) برای یک گراف در شکل زیر نشان داده شده است .



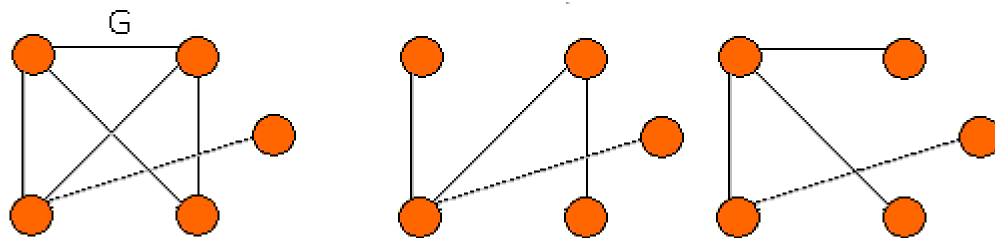
درخت پوشای حداقل

همانطور که قبلا تعریف شد درخت یک گراف متصل بدون است. یک درخت پوشا (spanning tree)زیرگرافی از گراف G است که شامل کلیه رئوس گراف G باشد و یک درخت باشد. بنابراین اگر گراف G دارای n گره باشد درخت پوشای آن دارای n-۱یال است .

پیمایش‌های DFS و BFS هر کدام یک درخت پوشا تولید می‌کنند .

تعداد درخت‌های پوشای يك گراف کامل با n گره برابر $2^{n-1}-1$ است.

مثال. دو درخت های پوشا که از پیمایش های DFS و BFS گراف G بدست آمده در شکل زیر نشان داده شده اند .



درخت پوشای حداقل (Minimum Spanning Tree) گراف وزن دارG ، درخت پوشائی است که مجموع وزن های آن حداقل باشد.

برای بدست آوردن درخت پوشای حداقل دو الگوریتم الگوریتم کروسکال (Kruskal) و الگوریتم پریم (Prim) را بررسی می کنیم .

الگوریتم کروسکال

گراف G با n راس را در نظر بگیرید. الگوریتم کروسکال به صورت زیر عمل می کند:

۱. تمام یال ها را به طور صعودی بر حسب وزن مرتب کنید.
۲. درخت T را متشکل از گره های G بدون یال را ایجاد کنید.
۳. عملیات زیر را n-۱ بارتکرار کنید:
٤. یک یال با حداقل وزن را به درخت T اضافه کنید به طوریکه حلقه ایجاد نشود.

گاهی چند یال دارای یک وزن هستند، در این حالت ترتیب یال هایی که انتخاب می شوند مهم نیست. درخت های پوشای حداقل مختلفی ممکن است حاصل شود اما مجموع وزن آنها همیشه یکسان و حداقل می شود .

پیچیدگی زمانی الگوریتم O(mn) می شود. که m تعداد یال ها و n تعداد رئوس گراف Gاست .

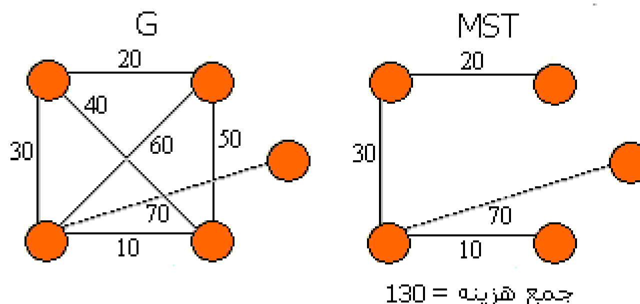
الگوریتم پریم

گراف G با n راس را در نظر بگیرید. الگوریتم پریم به صورت زیر عمل می کند:

۱. درخت تهی T را ایجاد کنید.
۲. راس v از گراف را انتخاب کرده و به درخت اضافه کنید.
۳. عملیات زیر را تکرار کنید تا کلیه راس های گراف به درخت T اضافه شوند:
۳. یالی که با حداقل وزن به رئوس T متصل است را پیدا کنید. یال و راس متصل به آن را به درخت T اضافه کنید به طوریکه حلقه ایجاد نشود.

پیچیدگی زمانی الگوریتم O(mn)می شود. که m تعداد یال ها و n تعداد رئوس گراف Gاست .

مثال



کاربردهای گراف

گراف کاربردهای متعددی در مسائل مختلف دارد از جمله می توان کاربردهای زیر را نام برد:

- پیداکردن کوتاهترین مسیر بین دو نقطه
- شبکه AOV برای کنترل پروژه و فعالیت ها و ارتباط بین آنها
- مسیربحرانی

درخت

داده های سلسله مراتبی توسط ساختمان داده درخت نمایش داده می شوند .انواع مختلفی از درخت ها وجود دارند که پرکاربردترین آنها درخت های دودویی هستند. درخت های BSTو heap نوعی درخت دودویی هستند که خواص ویژه ای دارند و در الگوریتم های مختلف برای حل مسائل استفاده می شوند .

اصطلاحات
درخت دودوئی
انواع درخت دودوئی
نمایش درخت دودوئی
پیمایش درخت دودوئی
درخت جستجوی دودوئی
درختHeap

اصطلاحات

ساختمان داده درخت برای نمایش داده‌های سلسله مراتبی به کار می رود و چون شبیه درخت رسم می شود ساختار درختی نامگذاری شده است. البته ساختار درختی در مقایسه با درخت واقعی معمولا به صورت وارونه رسم می شود، یعنی ریشه درخت در بالا و برگ های آن در پائین قرار می گیرند .

به طور كلي يک درخت مجموعه ای از گره هاست که از طریق پیوندهایی با هم در رابطه هستند. هر گره دارای داده مرتبط و مجموعه ای از گره های دیگر است .

در نظریه گراف یک درخت یک گراف متصل بدون دور است .

گره

داده ها دردرخت در ساختاری به نام گره (node) قرار دارند. هر گره حاوی اطلاعات و پیوند هایی به دیگر گره های درخت است .

شاخه

خطوطی که گره ها را در درخت به هم متصل می کنند شاخه (branche) نامیده می شوند .

والد و فرزند

گره ای که بلافاصله زیر یک گره قرار می گیرد فرزند (children) آن گره محسوب می شود. یک گره والد گره دیگر (parent) است اگر بلافاصله بالاتر از آن نزدیک تر به ریشه قرار داشته باشد .

گره ای که کلیه گره های سطوح پایین را به هم متصل می کند جد (ancestor) نامیده می شود .

ریشه

هر درخت گره خاصی به نام ریشه (root) دارد که کلیه گره های دیگر درخت در پایین آن قرار دارند. گره ریشه والدی ندارد. هر درخت تنها شامل یک گره ریشه است .

گره های همزاد

گره های همزاد (Sibling) گره هایی هستند که والد یکسانی دارند. به عبارت دیگر فرزندان يك گره با هم همزاد هستند .

درجه گره

تعداد فرزندان يك گره درجه (degree) آن گره ناميده ميشود.

درجه درخت

درجه درخت برابر ماكزيمم درجه گره‌ها در درخت است.

برگ

گره های بدون فرزند گره های پایانی (end-nodes) یا برگ (leaf) نامیده می شوند. درجه گره های برگ صفر است .

سطح

مجموعه گره هایی طول مسیر آنها تا ریشه یکسان است را سطح درخت (level) می نامند. اگر ريشه را در سطح يك فرض كنيم برحسب اينكه يك گره نسبت به ريشه در چه رديفی باشد شماره سطح می گيرد .

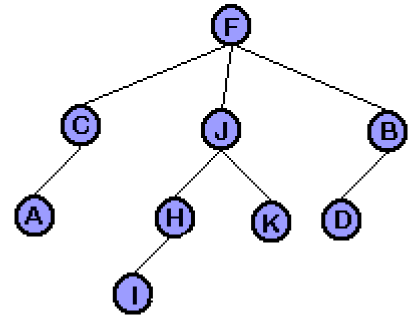
ارتفاع درخت

ارتفاع (height) درخت برابر با بيشترين سطح گره‌ها در درخت يا سطح دورترين برگ است. ارتفاع درختی که تنها گره ریشه را دارد صفر است .

هر درخت خواص زیر را نمایش می دهند:

- دقیقا یک ریشه دارد.
- همه گره ها بجز ریشه دقیقا یک والد دارند.
- تنها یک مسیر از بین هردو گره وجود دارد.
- دور وجود ندارد یعنی مسیری وجود ندارد که از یک گره شروع شود و به خود آن ختم شود.
- درختی که دارای n گره است n-۱ شاخه دارد

مثال. در درخت زير گره F ريشه است و گره هایC ، Jو B فرزندان ريشه هستند. گره هایA ، I، Kو D برگ های درخت هستند. درجه درخت ۳ و ارتفاع آن ٤ است.

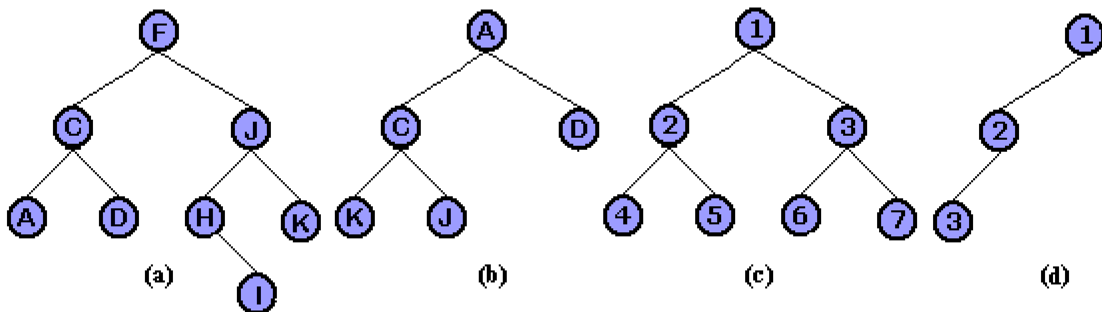


درخت دودوئی

احتمالا پرکاربردترین ترین نوع درختی، درخت دودویی (binary tree) است .درخت های دودویی یک پیاده سازی خاص از یک درخت m-ary است که در آن m=۲ است یا به عبارت ساده تر هر گره حداکثر دو فرزند دارد که فرزند چپ و فرزند راست)یا زیردرخت چپ و زیر درخت راست) نامیده می شوند .

ترتیت قرار گرفتن گره ها در درخت دودویی کاملا اختیاری نیست. درحقیقت نحوه درج گره های جدید در درخت دودویی ساختار و کاربرد آنرا تعیین می کند

مثال. درخت های زیر همگی دودویی هستند.



انواع درخت دودوئی

درخت دودوئی پر

یک درخت دودويي پر (full binary tree)درختی است که هر گره آن صفر یا دو فرزند دارد .

در درخت دودویی پر کلیه برگ ها در یک سطح هستند .

یک درخت دودويي پر به ارتفاع h دارای ۲^{h}-۱ گره است .

عمق يك درخت دودوئي پر با n گره $\log_{۲}^{n}$+۱ است.

تعداد گره‌های سطح i ام يك درخت دودوئی پر 2^{i-1} است

<u>مثال. درخت (c) در شکل فوق یک نمونه درخت دودویی پر است</u>

درخت دودوئی کامل

یک درخت دودویی کامل (complete binary tree) درختی است که همه سطوح آن به جز احتمالا آخرین سطح حداکثر گره ها را دارند و در سطح آخر گره ها از سمت چپ ظاهر می شوند. یعنی اگر ارتفاع درخت h باشد تعداد کل گره ها تا سطح h-۱ برابر با 2^h-1 است و در سطح آخر اگر گره هایی وجود دارند باید از چپ به راست اضافه شوند .

<u>مثال. درخت (b) در شکل فوق یک نمونه درخت دودویی کامل است.</u>

درخت دودویی میزان

درخت دودویی میزان (balanced binary tree) درختی است که کلیه برگ ها حداکثر یک سطح با هم تفاوت دارند

<u>مثال. در شکل قبل کلیه درخت ها به جز درخت (d) میزان هستند .</u>

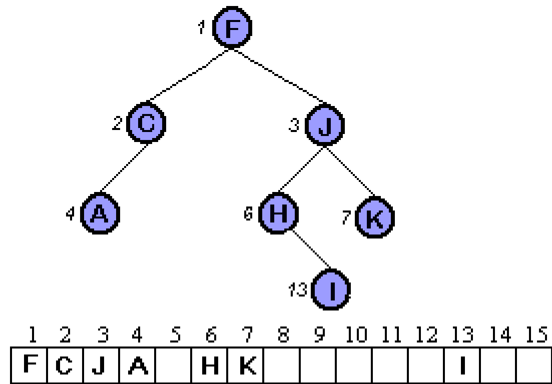
نمایش درخت دودوئی

نمایش توسط آرایه

يك درخت دودوئی کامل با n گره را می توان در يك آرايه يك بعدی ذخیره کرد. برای این کار گره های درخت شماره گذاری می شوند. شماره گذاری به ترتیب از بالا به پایین و از چپ به راست انجام می شود و به هر گره شماره ای تعلق می گیرد. سپس گره با شماره i در خانه i ام آرایه قرار می گیرد .

وقتی درخت دودویی در آرایه نمایش داده می شود برای هر گره با اندیس: i

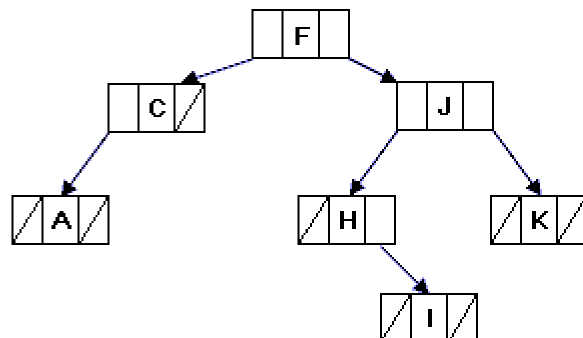
- اگر $i \neq 1$ باشد، والد i در $i/2$ است و اگر $i=1$ باشد i ریشه است.
- اگر $2i \le n$ باشد فرزند چپ i در $2i$ است و اگر $2i+1 \le n$ باشد، فرزند راست i در $2i+1$ است



نمایش توسط لیست پیوندی

یک درخت دودویی را می توان توسط لیست پیوندی نمایش داد. به این صورت که برای هر گره درخت يك گره در لیست در نظر گرفته می شود. هر گره يك اشارهگر به فرزند چپ و يك اشارهگر به فرزند راست دارد .

```
typedef struct node *TreePionter;
typedef struct node {
   char Data;
   Treepointer Left,Right;};
```



پیمایش درخت دودوئی

اغلب می خواهیم کلیه گره های درخت را بررسی کنیم. چند روش برای پیمایش وجود دارد که در آنها گره ها می توانند پردازش یا ملاقات شوند. هر روش وقتی روی یک درخت دودویی اجرا می شود ویژگی های مفیدی را در اختیار می گذارد .

سه روش معمول پیمایش درخت های دودویی روش هایpreorder ، postorderو inorderاست که در آنها هر گره و فرزندانش به طور بازگشتی ملاقات می شوند .هر سه پیمایش از ریشه درخت شروع می شوند. تفاوت آنها در ترتیب ملاقات گره و

فرزندانش است. در روش preorder ابتدا گره ملاقات شود سپس فرزندانش. در روش postorderابتدا فرزندان سپس خود گره ملاقات می شود. در روش inorder گره مابین فرزندان چپ و راست خود ملاقات می شود .

پیمایشPreorder

در روش preorder محتوای گره ریشه قبل از فرزند چپ و راست ملاقات می شود. الگوریتم بازگشتی پیمایش preorder به صورت زیر است :

۱. پردازش ریشه
۲. پیمایش زیردرخت چپ به روشpreorder
۳. پیمایش زیردرخت راست به روشpreorder

تابع زیر با توجه به الگوریتم فوق پیاده سازی شده است:

```
void Preorder(TreePointer T){
If (T!=NULL) {
   Visit(T->Data);
   Preorder(T->Left);
   Preorder(T->Right);
}}
```

پیمایش از گره جاری T شروع می شود سپس فرزند چپ و بعد فرزند راست آن ملاقات می شود Visit. برای نمایش یا پردازش مقدار گره است و بستگی به هدف از پیمایش می تواند بسادگی دستور نمایش محتوای گره باشد. تابع با فراخوانی گره ریشه آغاز می شود

مثال. ترتیب ملاقات گره ها به روش.Preorder

پیمایشInorder

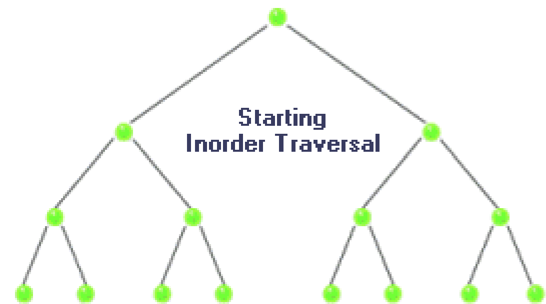
پیمایش inorder با ملاقات فرزند چپ گره جاری شروع شده سپس خود گره و بعد فرزند راست را ملاقات می کند. الگوریتم آن به صورت زیر است :

۱.پیمایش زیردرخت چپ به روش inorder
۲.پردازش ریشه
۳.پیمایش زیردرخت راست به روش inorder

تابع زیر مشابه تابع Preorder است با این تفاوت که ترتیب ملاقات گره تغییر کرده است:

```
void Inorder(TreePointer T){
If (T!=NULL) {
Inorder(T->Left);
Visit(T->Data);
Inorder(T->Right);
}}
```

مثال. ترتیب ملاقات گره ها به روش Inorder.



پیمایش Postorder

پیمایش postorder ابتدا محتوای زیردرخت چپ و سپس زیردرخت راست و در نهایت گره ریشه را ملاقات می کند. الگوریتم بازگشتی پیمایش postorder به صورت زیر است :

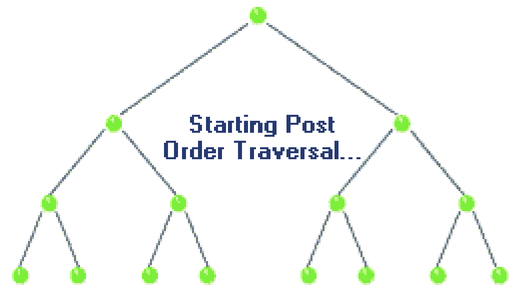
۱.پیمایش زیردرخت چپ به روش Postorder
۲.پیمایش زیردرخت راست به روش Postorder
۳.پردازش ریشه

تابع زیر با توجه به الگوریتم فوق پیاده سازی شده است:

```
void Postorder(TreePointer T){
If (T!=NULL) {
```

```
  Postorder(T->Left);
  Postorder(T->Right);
  Visit(T->Data);
}}
```

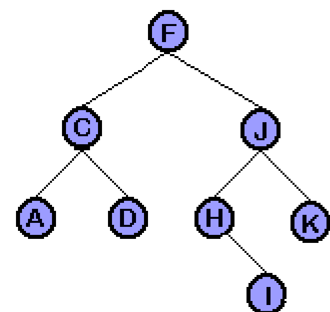
مثال. ترتيب ملاقات گره ها به روش.Postorder



پيمايش اول سطح

پيمايش اول سطح (breadth-first order) مشابه جستجوی اول سطح در گراف است و ابتدا سعی می کند نزدیک ترین گره به ریشه را ملاقات کند. پيمايش از سطح اول درخت آغاز شده، هر بار يك سطح از چپ به راست به طور کامل پيمايش می شود .

مثال. پيمايش های فوق روی درخت زير انجام شده است:



Pre Order : FCADJHIK
In Order : ACDFHIJK
Post Order : ADCIHKJF
Breadth First Order : FCJADHKI

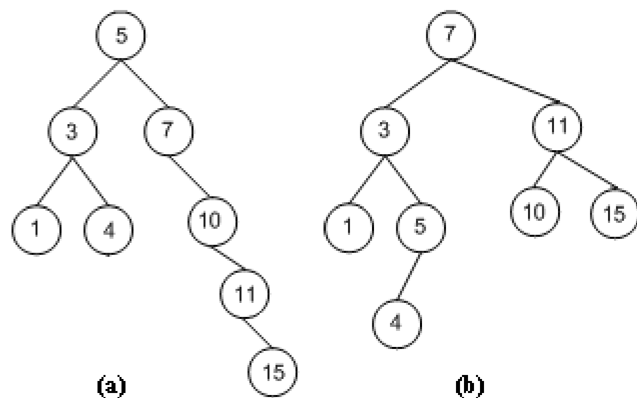
درخت جستجوی دودوئی

وقتي عمليات جستجو، حذف و اضافه مدنظر باشد درخت جستجوی دودوئی از تمام ساختارهای دیگر مناسب‌تر است .

یک درخت جستجوی دودویی (binary search tree) یا BST نوع خاصی از درخت دودویی است که اگر تهی نباشد خواص زیر را دارا است :

- •هر گره يك مقدار منحصر بفرد دارد.
- •کلیه مقادیر فرزندان زیردرخت چپ هر گره از مقدار خود گره کوچکتر هستند.
- •کلیه مقادیر فرزندان زیردرخت راست هر گره از مقدار خود گره بزرگتر هستند

مثال. درخت های زیر هر یک BST هستند .



جستجو درBST

جستجوی یک مقدار در BST از گره ریشه شروع می شود. آرگومان جستجو با مقدار گره مقایسه می شود. اگر یکسان باشند داده مورد نظر پیدا شده است در غیراینصورت اگر داده از مقدار گره کوچکتر باشد جستجو از زیردرخت چپ و اگر بزرگتر باشد جستجو از زیردرخت راست گره ادامه پیدا می کند .

به طور خلاصه الگوریتم جستجوی BST به صورت زیر بیان می شود Item .داده موردنظر است که در BST جستجو می شود. الگوریتم مقدار تهی یا گره ای که دنبالش هستیم را بر می گرداند .

```
Search ( TreePointer T, DataType Item )
Begin
  If (T = null) Then return null {tree is empty}
  Else If (Item = T.Data ) Then return T {item found}
  Else If (Item < T. Data and T.Left !=NULL ) Then
Data)ِ Search(T.Left,
  Else If (Item > T.Data and T.Right !=NULL ) Then Search
(T.Right, Data)
```

```
  End If
End
```

مثال. فرض کنید می خواهیم عدد ۱۰ را در درخت شکل (b) مثال قبل جستجو کنیم. جستجو از ریشه شروع می شود. عدد ۷ در ریشه است که کمتر از ۱۰ است .بنابراین اگر ۱۰ وجود داشته باشد باید در زیردرخت راست ریشه باشد .پس جستجو را از گره ۱۱ ادامه می دهیم. در حالت ۱۰ از عدد ۱۱ کمتر است بنابراین اگر ۱۰ در درخت باشد باید در زیردرخت چپ گره ۱۱ باشد. به سمت چپ گره ۱۱ حرکت می کنیم که گره ۱۰ است و گره ای که دنبالش می گشتیم را پیدا کردیم .

ممکن است که اگر گره ای که به دنبالش هستیم در درخت موجود نباشد. برای نمونه اگر بدنبال عدد ۹ هستیم ابتدا به همان طریق بالا عمل می کنیم. وقتی به گره ۱۰ می رسیم باید جستجو را از زیردرخت چپ ادامه دهیم ولی گره ۱۰ فرزند چپ ندارد بنابراین ۹ در درخت وجود ندارد

با دقت در الگوریتم جستجو می توان مشاهده کرد که درهرمرحله تعداد گره هایی که باید بررسی شوند نصف می شوند. بنابراین چنین زمان اجرای الگوریتم $\log_2 n$ می شود. البته زمان جستجو به توپولوژی درخت بستگی دارد و در بهترین حالت O(log n) است، در بدترین حالت O(n) می شود .

اگر یک BST به روش InOrder پیمایش شود مقادیر گره‌های درخت به صورت مرتب بدست می آید. زمان اجرای پیمایش روی درخت O(n) است.

درج درBST

درج یک گره جدید در BST در دو مرحله انجام می گیرد. ابتدا داده جدید در BST طبق الگوریتمی که ذکر شد جستجو می شود سپس در محل خاتمه جستجو گره جدید اضافه می شود .

با فرض اینکه داده تکراری در درخت مجاز نیست، هنگام درج گره جدید دو شرط باید مدنظر باشد :

- درج در یک درخت خالی. در این حالت گره ای که درج می شود ریشه درنظر گرفته می شود .
- درج در یک درخت غیر خالی. درخت باید جستجو شود همانطور که در بالا ذکر شد تا محل درج گره تعیین شود. گره جدید ابتدا با ریشه درخت مقایسه می شود .اگر مقدار گره جدید کمتر از ریشه باشد و زیردرخت چپ خالی باشد گره جدید در محل فرزند چپ ریشه درج می شود. درغیراینصورت جستجو از زیردرخت چپ ادامه پیدا می کند. اگر مقدار گره جدید بزرگتر از ریشه باشد و زیردرخت راست خالی باشد گره جدید در محل

فرزند راست ریشه اضافه می شود. درغیر اینصورت فرآیند جستجو از زیردرخت راست ادامه پیدا می کند .

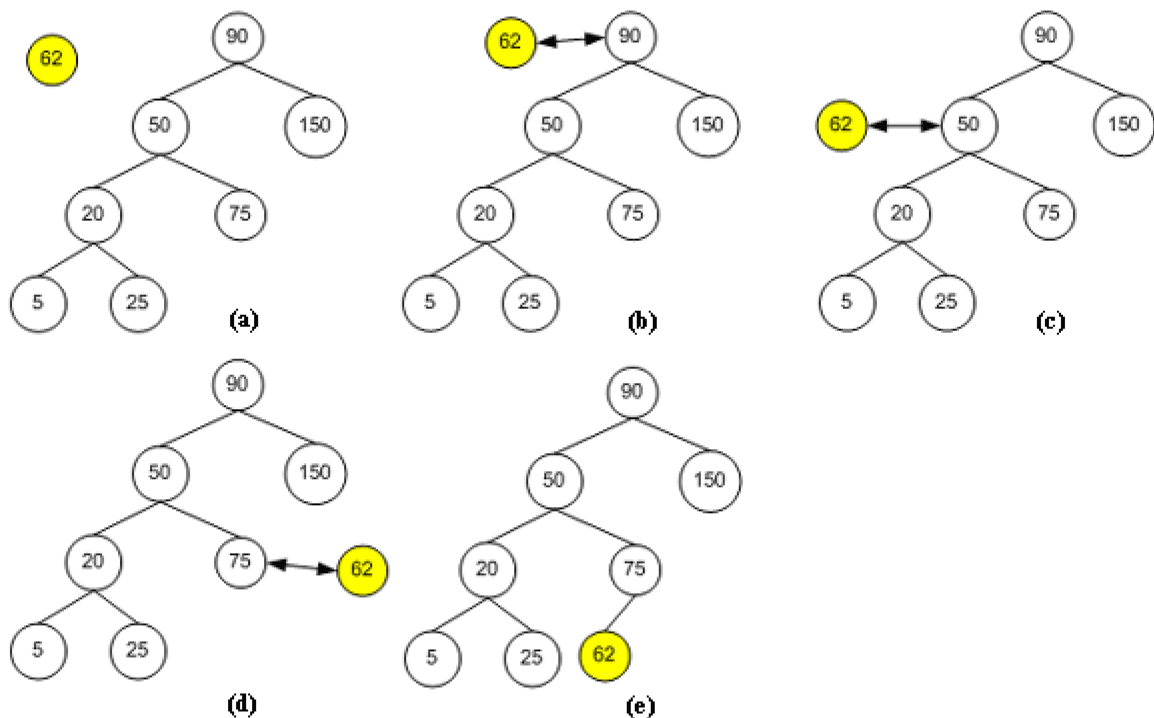
اگر زیربرنامه جستجو متوجه شود که گره جدید قبلا در BST وجود دارد پایان می یابد و دوباره آنرا درج نمی کند .

گره جدید به نحوی باید به درخت اضافه شود که BST خواص خود را حفظ کند.

گره جدید همیشه به صورت یک برگ اضافه می شود. بنابراین ترتیب درج روی توپولوژی درخت تاثیر می گذارد .

زمان اجرای الگوریتم درج مشابه الگوریتم جستجو است .

مثال. مراحلی که باید طی شود تا گره ٦٢ به BST اضافه شود در شکل های زیر نشان داده شده است .

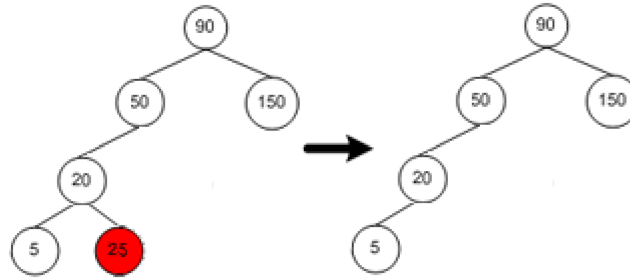


حذف از BST

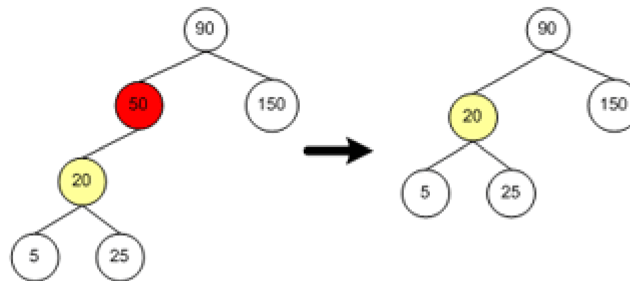
حذف یک گره از BST نسبتا دشوارتر از درج است. زیرا وقتی گره ای حذف می شود که دارای فرزند است باید گره دیگری انتخاب شود تا جایگزین گره حذف شده شود. اگر این انتخاب درست انجام نشود خواص BST نقض می شود .

گره ای که باید حذف شود ابتدا در BST جستجو می شود سپس به نحوی جایگزین می شود خواص درخت حفظ شود. چند حالت برای جایگزین کرده گره وجود دارد :

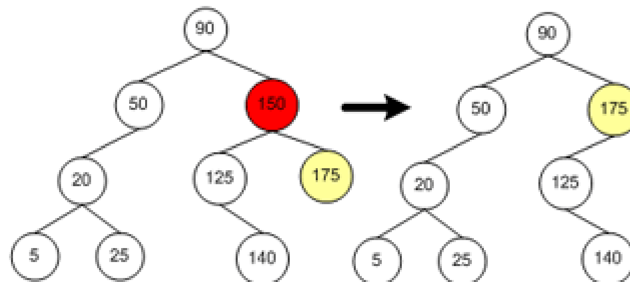
حالت ۱. گره که باید حذف شود برگ باشد و فرزندی نداشته باشد. در این حالت حذف به سادگی انجام می پذیرد و کافی است اشاره گر والد برابر تهی شود .برای مثال در شکل گره ۲۵ که حذف می شود فرزندی ندارد .



حالت ۲. گره ای که باید حذف شود تنها دارای یک فرزند چپ است که می تواند جایگزین آن شود. برای مثال فرض کنید بخواهیم گره ۵۰ را حذف کنیم. چون ۵۰ فرزند راستی ندارد ۲۰ با آن جایگزین می شود .

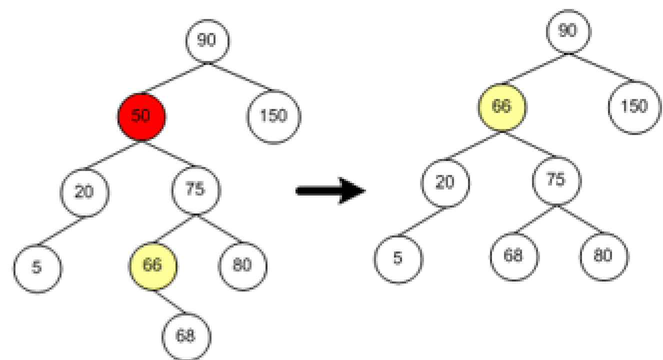


حالت ۳. فرزند راست گره ای که باید حذف شود فرزند چپی ندارد بنابراین فرزند راست گره جایگزین آن می شود. برای مثال اگر بخواهیم گره ۱۵۰ را حذف کنیم چون فرزند راست آن فرزند چپی ندارد فرزند راستش یعنی ۱۷۵ جایگزین می شود .



حالت ٤. فرزند راست گره ای که باید حذف شود فرزند چپ دارد. در این حالت چپ ترین فرزند راست گره جایگزین آن می شود. یعنی کوچکترین مقدار زیر درخت راست

گره .برای مثال اگر گره ۵۰ را در شکل زیر حذف کنیم چپ ترین فرزند آن یعنی ٦٦ را انتخاب کرده و جایگزین می کنیم .



در هر BST کوچکترین مقدار در سمت چپ ترین گره و بزرگترین مقدار در سمت راست ترین گره وجود دارد .

اولین مرحله در حذف گره پیدا کردن محل گره ای است که می خواهیم حذف کنیم که توسط الگوریتم جستجو که قبلا توضیح داده شد انجام می گیرد. بنابراین زمان حذف همان زمان O(log n)در حالت متوسط و O(n)در بدترین حالت است .

معایبBST

باوجودیکه درخت های جستجوی دودويي به طور ایده ال زمان زیرخطی برای درج، جستجو و حذف می دهند، زمان اجرا بستگی به توپولوزی BST دارد. توپولوژی درخت وابسته به ترتیبی است که داده در BST درج می شود. اگر داده ورودی مرتب باشد توپولوژی BST یک درخت نامیزان طولانی و باریک می شود که بدترین زمان را در اجرای الگوریتم ها می دهد و معمولا در دنیای واقعی داده ها بطور طبیعی مرتب یا نزدیک به مرتب هستند .

درختHeap

heapیک ساختمان داده درختی که خواص ویژه ای را داراست .

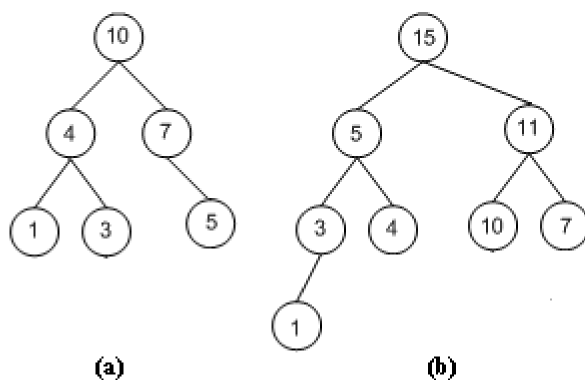
heapرا می توان به صورت زیر تعریف کرد:

- یک max meap يك درخت دودوئي كامل است كه بزرگ هم باشد. در max heapبزرگترین مقدار همیشه در ریشه قرار می گیرد.
- یک min meap يك درخت دودوئي كامل است كه كوچك هم باشد. در min heap كوچكترين مقدار در ریشه قرار می گیرد.

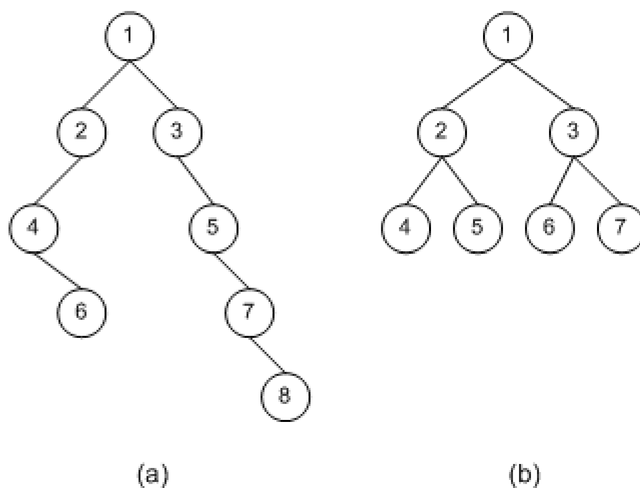
يك درخت بزرگ (max tree) درختی است که مقادیر کلید هر گره بزرگتر از مقادیر فرزندانش باشد. یعنی اگر B فرزند A است آنگاه $key(A) \geq key(B)$ است .

يك درخت كوچك (min tree) درختی است که مقادیر کلید هر گره کوچکتر از مقادیر فرزندانش باشد. یعنی اگر B فرزند A است آنگاه $key(A) \leq key(B)$ است

مثال. در شکل زیر درخت (b) یک max heap است. درخت (a) یک درخت بزرگ است ولی max heap نیست چون یک درخت دودویی کامل نیست .



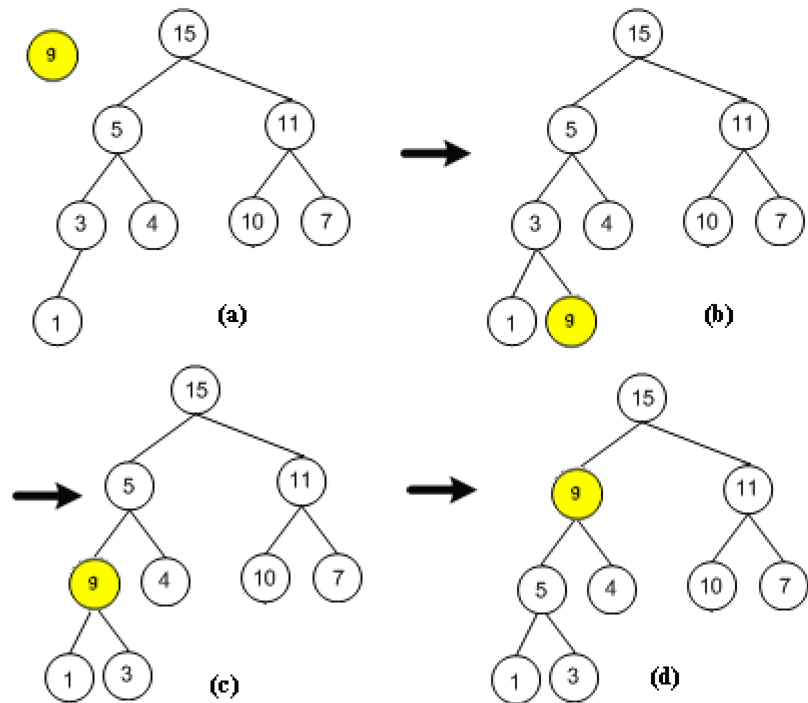
مثال. در شکل زیر درخت (b) یک min heap است. درخت (a) یک درخت کوچک است ولی min heap نیست چون یک درخت دودویی کامل نیست



درج در Heap

یک گره جدید در heap در محلی اضافه می شود که درخت دودوئی کامل باقی بماند. پس از درج درخت تنظیم می شود تا خواص heap حفظ شود. برای مثال اگر max heap باشد و اگر مقدار کلید گره جدید از والدش بیشتر باشد جای آن با والد عوض می شود. این عمل ممکن است تا ریشه ادامه پیدا کند

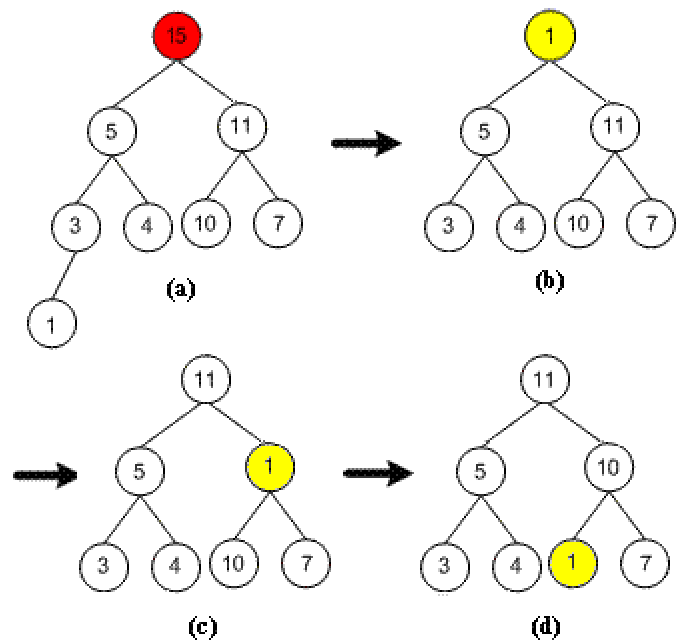
مثال. در درخت زیر گره جدید ۹ به max heap اضافه شده است. پس از درج heapتنظیم می شود و جای ۹ ابتدا با گره ۳ و سپس با ۵ جابه جا می شود



حذف ازHeap

در Heap حذف همیشه از ریشه صورت می گیرد. بعد از حذف ریشه، آخرین گره درخت جایگزین ریشه می شود. سپس درخت تنظیم می شود تا خواص heap را حفظ کند

مثال. در max heap شکل زیر گره ۱۵ که در ریشه قرار دارد حذف می شود. گره ۱که سمت چپ ترین گره درسطح آخر درخت است جایگزین ریشه می شود. سپس درخت تنظیم می شود تا به صورت max heap در بیاید. ابتدا گره ۱ با ۱۱ و سپس با ۱۰ جا به جا می شود



زمان اجرای عملیات حذف و درج در heap O(log n)است.

کاربردهای Heap

در maxh heap بزرگترین مقدار همیشه در ریشه است . در min heap کوچکترین مقدار همیشه در ریشه قرار می گیرد. به خاطر همین ویژگی heap برای پیاده سازی صف‌های الویت (priority queues) بسیار مناسب است. در صف الويت هر عنصر با الويت دلخواه را مي‌توان اضافه کرد اما عنصر ماکزیمم یا مینیمم همیشه ابتدا از صف حذف می شود .

Heapها در الگوریتم مرتب سازی heapsort هم استفاده می شوند .

http://www.hpkclasses.ir/Courses/DataStructure/ds۰۸۰۰.html