ساختمان داده ها در C++

مدرس :
مهندس شادی

گرد آورندگان
المیرا دربانی
حمید وحدت پسند

ساختمان داده

ساختمان داده ها شاخه ای از مهندسی نرم افزار است که به مطالعه انواع ساختارهای داده و نیز الگوریتم های مربوط به آنها می پردازد .

ساختار داده[1]

مجموعه ای است ساختمند و نامدار از داده های مرتبط به هم را ساختارداده گویند .

پیمایش

دستیابی به تمام عناصر یک ساختار داده به طوری که هر عنصر از آن تنها یک بار مورد دستیابی قرار گیرند عمل پیمایش نامیده می شوند .

الگوریتم[2]

الگوریتم مجموعه ای متناهی از دستورالعمل هاست که روش حل مسئله ای را بیان می کند و داری شرایط و مشخصات زیر است :

۱- ورودی : یک الگوریتم می تواند داری چندین ورودی باشد . همچنین می تواند هیچ ورودی نداشته باشد .
۲- خروجی : الگوریتم باید دارای حداقل یک نتیجه خروجی باشد .
۳- وضوح : هر دستورالعمل درالگوریتم باید بدون ابهام باشد .
۴- خاتمه پذیری : الگوریتم باید پس از طی مراحل محدودی خاتمه یابند .
۵- انجام پذیری : هر دستورالعمل در الگوریتم باید قابل انجام باشد .

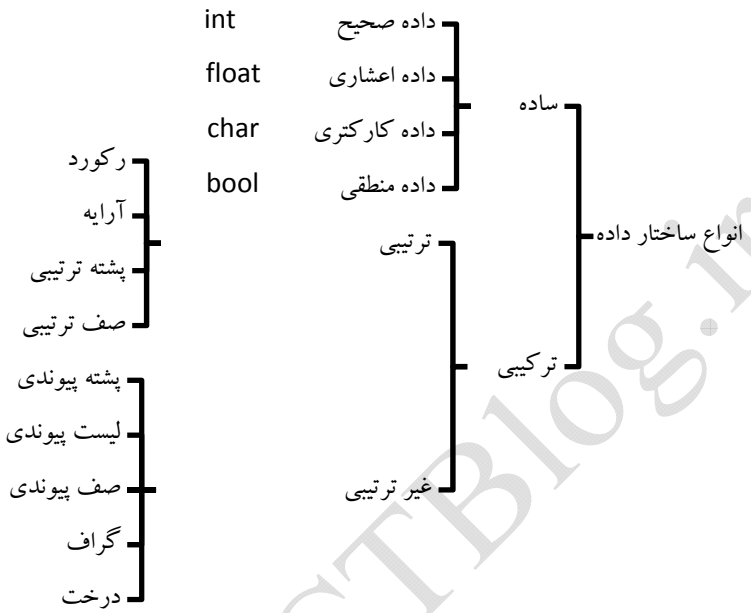
مراحل کلی تولید نرم افزار

به طور کلی فرآیند تولید یک محصول نرم افزاری شامل مراحل شش گانه زیر است :

۱- نیازمندیها
۲- تجزیه و تحلیل
۳- طراحی
۴- پیاده سازی
۵- تست و بازبینی
۶- نگهداری

1. Data structure
2. Algorithm

به این مراحل اصطلاحا چرخه زندگی سیستم[1] گفته میشودکه یک فرآیند تکاملی است .



ساختارهای داده ای ساده شامل داده های صحیح ، کاراکتری ، اعشاری و منطقی هستند . به عنوان مثال در زبان c++ به ترتیب انواع داده ها : int ، char ، float ، bool وجود دارد ساختارهای داده ترکیبی همانطور که از نامشان پیداست از ترکیب و سازماندهی ساختارهای ساده پدید می آید .

معیارهای انتخاب ساختار داده :معیارهای مهم در انتخاب ساختار داده را می توان به ترتیب اهمیت به صورت زیر بیان کرد :

۱- امکان پذیری : نوشتن الگوریتم های لازم بر روی ساختار داده با توجه به اهداف مسئله امکان پذیر باشد .

۲- سرعت اجرای الگوریتم : ساختاری انتخاب شود که الگوریتم های اعمال شده بر روی آن دارای سرعت اجرای مناسبی باشد .

۳- حافظه مصرفی : حافظه اشغال شده توسط داده های ساختار بهینه باشد .

1. System life cycle

۴- سازگاری با ماهیت مسئله : ساختاری انتخاب شوند که با منطق برنامه و ماهیت داده ها ی مسئله هم گونی داشته باشند . مثال : برنامه ای را در نظر بگیرید که هدف آن ذخیره اطلاعات تعداد نا مشخصی از دانشجویان و انجام عملیات اضافه ، حذف و جستجو باشند برای این برنامه ساختارهای داده مختلفی به شرح زیر قابل تعریف است :

۱- ساختار داده ساده که شرط اول (امکان پذیری) را دارا نیست .

۲- بکارگیری آرایه ای از رکورد های دانشجو .

۳- استفاده از لیست پیوندی (این ساختار تمام معیارهای لازم به عنوان یک ساختار داده مناسب را دارد .)

اعمال اصلی در ساختار داده

ساختار داده ابزاری است جهت نمایش و مدیریت داده ها ، یک جنبه مهم جهت تحقق مدیریت داده ها انجام عملیات بر روی داده های ساختار است .

اعمال اصلی در ساختارهای داده ای به شرح زیر است :

۱- ایجاد[1] ساختار

۲- درج[2] عنصر جدید

۳- حذف[3] عنصر

۴- به هنگام[4] سازی عنصر

۵- جستجوی[5] عنصر

۶- پیمایش[6] ساختار

۷- حذف ساختار

1. Create
2. Insert
3. Delete
4. Update
5. Search
6. Traversal

معیارهای سنجش کارایی الگوریتم

کارایی الگوریتم به دو عامل اصلی زیر بستگی دارد :

۱- پیچیدگی حافظه[1] : مقدار حافظه مورد نیاز جهت اجرای کامل الگوریتم

۲- پیچیدگی زمانی[2] : مدت زمان لازم جهت اجرای کامل الگوریتم

توجه

انتخاب ساختار داده ارتباط نزدیکی با پیچیدگی حافظه و پیچیدگی زمانی الگوریتم دارد .

زمان واقعی

برای بیان پیچیدگی زمانی الگوریتم دو روش وجود دارد :

یکی اینکه محاسبه پیچیدگی الگوریتم به صورت دقیق از لحظ زمان اجرای کامل برنامه مورد نظر بر روی کامپیوتر انجام شود این روش علاوه بر مشکل بودن به نوع ماشینی که الگوریتم برروی آن ایجاد شود بستگی دارد . مثال : پردازنده پنتیوم two ، tree ، four به حافظه ربط دارد به عوامل سخت افزاری نرم افزاری دیگری مانند حافظه نهان پردازنده ، میزان حافظه اصلی ، سرعت کامپایلر و موارد دیگر بستگی دارد.

در نتیجه معیار صحیحی برای بیان زمان اجرای الگوریتم وجود ندارد . مگر اینکه خود را محدود به عوامل سخت افزاری و نرم افزاری خاصی نمائیم . در اینگونه موارد معمولا از زمان سنج برای محاسبه زمان اجرا استفاده می شود . در اغلب زبانهای برنامه نویسی تابعی جهت دریافت زمان سیستم وجود دارد .

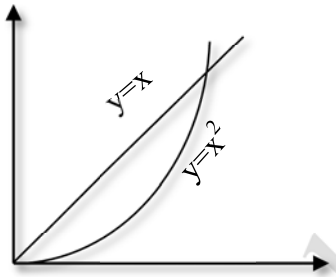
تابع زمانی روش دیگر این است که پیچیدگی الگوریتم را به صورت تابعی از ورودی بیان کنیم در این روش بدون اینکه نیاز به محاسبه دقیق زمان اجرای الگوریتم بر روی یک پلتفرم[3] سخت افزاری و نرم افزاری خاص باشد معیاری تئوریک بر پایه مفاهیم ریاضی برای مقایسه کارایی الگوریتم ها ارائه می شود که به مستقل از نوع کامپیوتری است که برای اجرای الگوریتم استفاده میشود .

1. Space complexity
2. Time complexity
3. Platform

تابع زمانی

تابعی است که زمان اجرای یک الگوریتم را بر حسب ورودی های آن بیان می کند .

تابع زمانی یک تابع ریاضی است که می تواند یک متغیره یا چند متغیره باشد تعداد متغیرهای تابع زمانی برابر با تعداد ورودی های الگوریتم است . این تابع رفتار زمان الگوریتم را بر حسب ورودی های آن نشان میدهد ، به عبارت دیگر نشان می دهد رشد زمان اجرای الگوریتم چه رابطه ای با ورودی های آن دارد .



تفاوت برنامه و الگوریتم

برنامه همیشه باید قابل اجرا باشد و الگوریتم در یک نقطه ای باید پایان داشته باشد .

مرحله

هر دستور از برنامه که انجام آن زمان ثابتی طول می کشد به عنوان یک مرحله از برنامه در نظر گرفته می شود .

رتبه[1] زمانی

رتبه زمانی ، نمادی برای بیان ماهیت تابع زمانی الگوریتم است .

مثال

الگوریتم زیر مجموع عناصر یک آرایه به طول n را محاسبه می کند می خواهیم با تعیین تعداد مراحل برنامه[2] تابع زمانی آن را معین می کنیم .

```
float sum ( float  a[ ] , int n )
{
    float s = 0 ;
    for ( int i = 0 ; i < n ; i++ )
    s = s + a[ i ] ;
    return s;
}
```

جواب

0 + 1+ n + 1 + n +1

1. Order
2. Program steps

روشن است که رتبه زمانی دقت تابع زمانی را ندارد . رتبه زمانی یک معیار تخمینی برای بیان رفتار تابع زمانی الگوریتم است و چگونگی رشد زمان الگوریتم را بر حسب ورودی های آن نشان می دهد با داشتن تابع زمانی یک الگوریتم به راحتی می توان رتبه زمانی آن را تعیین کرد ولی با داشتن رتبه زمانی نمی توان تابع زمانی رابه دست آورد .

مثال

تابع زمانی $T(n) = 2n+3$

رتبه زمانی $F(n) = n$

$Y = x$
$Y = n$
$T = n$

پیچیدگی یک الگوریتم از لحاظ فضا و زمان به دو قسمت بستگی دارد

۱- **قسمت ثابت** : به قسمتهایی از الگوریتم مربوط می شود که مستقل از اندازه ورودی الگوریتم است .

۲- **قسمت متغیر**: به قسمتهایی از الگوریتم مربوط می شود که اندازه آن بستگی به اندازه ورودی الگوریتم دارد .

تعریف (O بزرگ)

اگر g و f توان نامنفی باشند ، آنگاه می گوییم : f(n)=O(g(n)) اگر و فقط اگر داشته باشیم که در آن c , no مقادیر ثابتی هستند :

$\exists n > 0 , c > 0 , \forall n \geq no : f(n) \leq c.g(n)$

برای کران بالا است یعنی g(n) کران بالا برای f(n) است .

مثال

فرض کنید : f(n)=3n+2 باشد ، می توان نوشت : f(n)=O(n)

زیرا $3 \times 2 + 2n$ $\exists n \geq 2 : 3n + 2 < 4n$

که در آن $g(n) = n , \ c = 4 , \ no = 2$

بنابه گفته دکتر به مقدار n که دارای توان بیشتر است عدد ۱ اضافه می کنیم و برای مقدار o اعداد مابقی را با هم جمع می کنیم مقدار o به دست می آید .

مثال

فرض کنید : f(n)=10n+4n+2 باشد می توان نوشت :

$f(n) = o(n^2)$
$\forall n \geq 6 : 10n^2 + 4n + 2 \leq 11n^2$
$g(n) = n^2 \quad c = 11 \ , no = 6$

تعریف امگا Ω

اگر f , g توان نا منفی باشند آنگاه گوییم f(n)=Ω(g(n)) اگر و فقط اگر داشته باشیم که در آن no , c مقادیر ثابتی هستند . این نماد برای کران پایین است بر تابع f(n) بکار می رود یعنی g(n) کران پایین است برای g(n) بدیهی است که برای تابع f(n) چندین کران پایین می تواند وجود داشته باشد . بنابراین g(n) را بزرگترین کران پایین در نظر می گیریم :

$\exists \, no > 0 \ , c > 0 \, , \forall n \geq no \ , f(n) \geq c.g(n)$

مثال

فرض کنید f(n)=3n+2 آنگاه می توان نوشت : f(n)= Ω (n)

زیرا $\forall n \geq 1 : 3n + 2 \geq 3n$

که در آن $g(n) = n \, , \ c = 3 \, , \ no = 1$

مثال

فرض کنید f(n)=10n+4n+2 آنگاه داریم : f(n)=(2)

زیرا: n>=1 , 602+n>=2

که در آن :no=1 , c=1 , g(n)=2 است .

تعریف تتا Θ

اگر f , g توان نامنفی باشند آنگاه داریم f(n)=θ(g(n)) اگر و فقط اگر داشته باشیم که در آن no , c , c مقادیر ثابتی هستند :

$\exists \, no > 0 \ , c_1, c_2 > 0 \, , \forall n \geq no : c_1 \, . g(n) \leq f(n) \leq c_2 . g(n)$

این نماد به این معناست که تابع g(n) هم کران بالا و هم کران پایین برای تابع f(n) است . بنابراین نماد تتا از دو نماد O و اهم دقیق تر است .

مثال

اگر $f(n)=3n+2$ آنگاه $f(n)=\theta(n)$

زیرا $\forall n \geq 2, 3n \leq 3n+2 = 4$

که در آن $g(n)=n$, $no=2$, $c_1=3$, $c_2=4$

مثال

اگر $f(n) = 10n^2+4n+2$ آنگاه $f(n)=\theta(n)$

زیرا $\forall n \geq 6, n^2 \leq 10n^2+4n+2 \leq 11n^2$

که در آن: $g(n)=n^2$, $n_0=6$, $c_1=1$, $c_2=11$

برای محاسبه پیچیدگی یک الگوریتم می توان سه حالت زیر را بیان کرد :

۱- **بهترین حالت Ω**
۲- **حالت میانگین Θ**
۳- **بدترین حالت O**

برای اینکه معیار صحیحی برای مقایسه دو الگوریتم موجود در یک مسئله داشته باشیم ، منطقی تر این است که پیچیدگی را در بدترین حالت در نظر بگیریم به عبارت دیگر اغلب لازم است به دنبال یک کران بالا برای تابع زمانی الگوریتم باشیم به همین منظور معمولا پیچیدگی بیشتر الگوریتم ها بر حسب O بزرگ بیان می شود .

(کنکور ۷۹)

تعداد کل مراحل برنامه زیر را مقایسه کنید ؟

float sum (float list [ ] , int n)	0
{	0
float s = 0 ;	1
int i ;	0
for (I = 0 ; I < n ; i++)	n+1
s= s + list [i];	n
return s ;	1
}	0
	$t(n)$ یا $F(n)=2n+3$

مثال ۳ صفحه ٤٦

دستور اصلی ;x=x+1 تکه برنامه زیر چند بار اجرا می شود ؟ کل گامهای برنامه چقدر است ؟

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

$(m+1)+m(n+1)+nm$
$= m+1+mn+m+nm$
$= 2m+2nm+1$
فرض $\Rightarrow$ $m=n$ $2n+2n^2+1$ $\Rightarrow$ $2n^2+2n+1$

مثال

تعداد اجرا شدن دستور اصلی x=x+1 چیست ؟ در تکه برنامه زیر چیست ؟

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

تعداد اجرا شدن دستور اصلی	شرط i>1	i
۱	درست	۱۶
۱	درست	۸
۱	درست	۴
۱	درست	۲
–	نادرست	۱

پس در حالت کلی دستور اصلی . تعداد $[log_2^n]$

نکته

در ساختمان داده منظور از لگاریتم ، لگاریتم در مبنای ۲ است .

نکته

مرتبه اجرای حلقه زیر $O(1)$ می باشد چراکه حلقه به مقدار معین و محدودی اجرا میشود.

```
for ( i=0 ; i <= 13 ; i++ )
cout << "OK" ;
```

مثال

مرتبه اجرایی چیست ؟

فرض کنید n = 32 است ، پس برای عمل اصلی x = x+1 پنج بار انجام می شود $log_2 32$ پس در حالت کلی مرتبه اجرای الگوریتم فوق $O[log_k^n]$ خواهد بود همین ترتیب اگر شمارنده با دستور (i = i oliv k) ، (i = i * k) از یک تا n تغییر کند بازهم مرحله اجرای آن $O[log_k^n]$ است .

i	Y
32	1
16	2
8	3
4	4
2	5
1	-

مثال

```
x = 0 ;
i = n ;
while ( i > 1 )
{
x = x + 1 ;
i = I /2 ;
}
```

k

مثال

مرتبه زمانی الگوریتم زیر چیست ؟

```
proc ( var x : integer ; n integer )
var i , j , k : integer ;
{
        for i : to n do ;
        for j : 1 to i do ;
        for k : 1 to j do ;
            ( x = x+1 )
}
```

۱. $O(n)$

۲. $O(n^2)$

۳. $O(n^3)$ ✓

۴. $O(n \log n)$

نکته :

سه حلقه تودرتو چه به هم وابسته باشند و چه از هم مستقل باشند دارای مرتبه اجرائی ($O(n^3)$ هستند .

مثال

کدام یک از مقادبر زیر ئرست است ؟

1. $5n2 - 6n = O(n3)$
2. $5n2 - 6n = \theta(n3)$ ✓
3. $5n2 - 6n = \Omega(n3)$
4. $5n2 - 6n = \theta(n)$

واحد 10 = F (n) یا T(n) تابع زمانی

چون $F(n) \le y(n) \Rightarrow O(y(n))$

توضیح :

گزینه یک صحیح است برای این که در معیار ارزیابی الگوریتم با نام O میدانیم که تابع زمانی f(n) یا t(n) به ازای مقادیر مشخص c و n_0 و قضیه مربوطه کوچکتر از تابع g(n) است و گفتیم g(n) کوچکترین کران بالا برای f(n) است یعنی تابع زمانی الگوریتم ما زمان اجرایش کوچکتر از زمان اجرای تابع g(n) است یا طبق قضیه میدانیم $g(n) = on^2$ است و لذا چون $f(n) < g(n)$ یعنی n^2 است در حالت کلی f(n) میتواند کوچکتر از هر تابعی که بزرگتر از g(n) است تابع n^3 یعنی on^3 بزرگتر از on^2 است .

الگوریتم های برگشتی[1] :

الگوریتمی را بازگشتی گویند هرگاه حداقل دارای یک دستورالعمل باشد که خودش را فراخوانی کند .

دو ویژگی الگوریتم بازگشتی :

۱. زیر برنامه خودش ، خودش صدا بزند . (فراخوانی کند)
۲. یک شرط جهت انجام فراخوانی ها وجود دارد .

نکته

روش غیر بازگشتی را روش پویا[2] نیز میگویند .

1. Recursive
2. Dynamic

مثال

برنامه ای بنویسید که عددی را خوانده و با کمک تابع غیر بازگشتی و تابع بازگشتی n فاکتوریل را محاسبه و آن را چاپ کند ؟

```
main ()
{
        int x ;
        cout << fact 1( x ) ;
        cout << fact ( x ) ;
}
//---------- //تابع غیر بازگشتی ----------
{
        int  i , f = 1 ;
        for ( i = 1 ,  i <= n ; i++ )
        f = f  * i ;
        return (f);
}
//---------- //تابع بازگشتی ----------
int fact ( int n )
{
        if ( n <= 1 )
        return 1 ;
        else
        return n * fact ( n-1 ) ;
}
```

نکته

روش بازگشتی با روش معمولی سادگی برنامه نویسی۱ آن است و عیب آن مصرف زیاد حافظه است و نیز زمان زیادی است که برای فراخوانی ها صرف می شود .

نکته

اگر زیر برنامه ای مرتباً خودش را صدا بزند و شرطی برای اتمام فراخوانی ها وجود نداشته باشد پس از مدتی پیام خطای stack over flow را می بینید در زمان اجرا صادر و اجرا برنامه متوقف می شود.

مثال : صفحه ۵۷

اگر m واحد زمان اجرای ماژور A شود و N تعداد داده های ورودی باشد تابع پیچیدگی زمانی الگوریتم زیر کدام است ؟

۱. $mn\ log_5$ ✓ ۲. mn^2 ۳. $n\ log_5^m$ ۴. $mn\ log_2^n$

```
i := N
while I > 1
for j := to N
module A
end for
I := I / 5
End while
```

مثال صفحه ۵۸

در برنامه زیر تعداد دفعات تکرار دستورالعمل شماره ۳ برابر است با ؟

۱. $\frac{n(n+1)}{2}$ ✓ ۲. n^2 ۳. $\frac{n^2}{2}$ ۴. $\frac{n(n-1)}{2}$

```
1. for ( k = 0 ; k <= n-1 ; k++ )
2. for ( i = 1 ; I <= n-k ; i++ )
3. a [ i ] [ i + k ] = k ;
```

تعداد مراحل یا تعداد تکرار دستورالعمل پیچیدگی زمانی یا تابع زمانی

n	k	I
3	0	1 - 2 – 3
	1	1 – 2
		1

مجموع تکرار دستور شماره n $(1 + 2 + 3 + \dots + n) = \frac{12}{2} = 6 \quad \frac{3(3+1)}{2}$

مثال

کدام جریان بعضی از دستورات را هیچ ، یک و یا بیش از یک بار اجرا می کند ؟

۱. Repeative ✓ ۲. Sequential
۳. Conditional ۴. Conditional , Repeative

مسئله بریجهای هانوی

شروط

۱. در هر انتقال فقط و فقط یک دیسک جابجا شود .

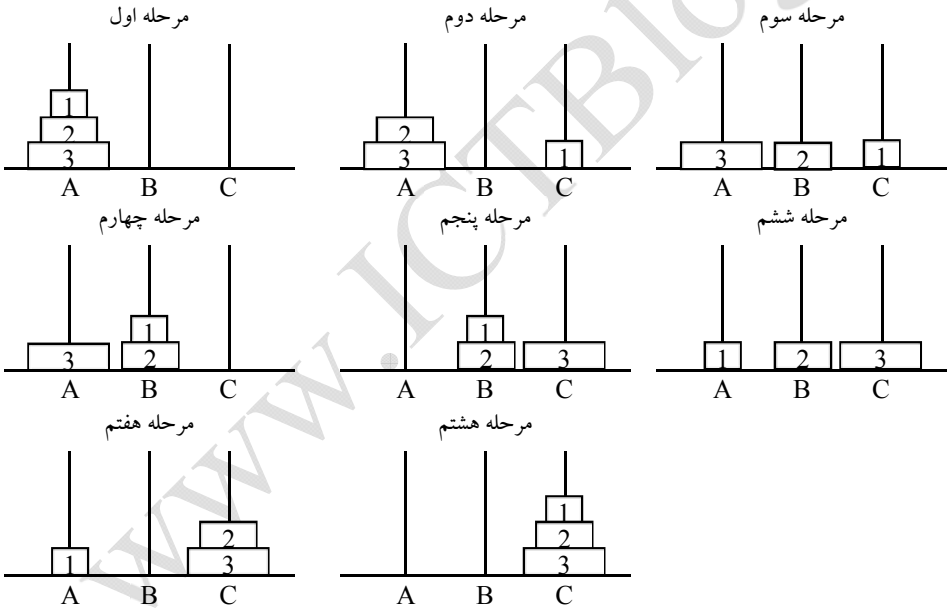
۲. در هیچ مرحله ای (انتقال) نمی تواند یک دیسک بزرگتر روی دیسک کوچک تر قرارداد .

نکته

تعداد کل مراحل برای برای n دیسک برابر است با 2^n-1 بار

مثال :

برج های هانوی را برای ۳ دیسک پیاده سازی کنید ؟



نحوه نوشتن جملات بازگشتی برای حالت کلی n دیسک :

۱. ابتدا n-1 دیسک را از مبدأ (A) به میله کمکی (B) انتقال بده.

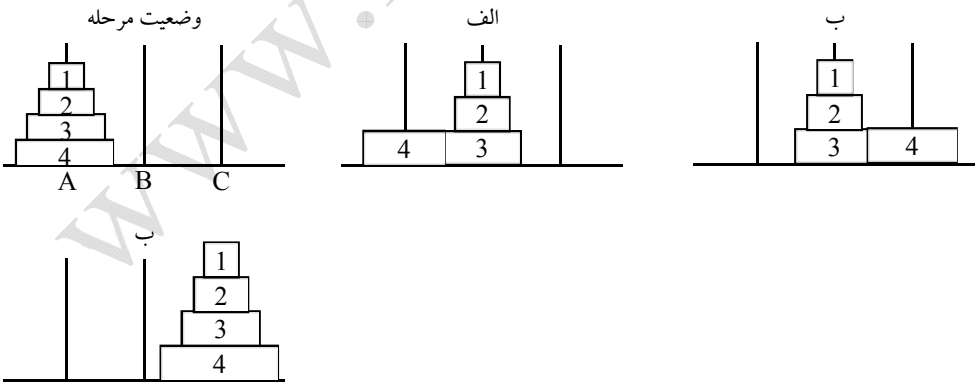
۲. تنها دیسک باقی مانده در میله A (که بزرگترین دیسک است) را به میله مقصد منتقل کن

۳. n-1 دیسک موجود در میله کمکی B را به میله C انتقال بده.

۴. با انجام مراحل ۱ تا ۳ مسئله حالت n ام تبدیل به مسئله n-1 ام می شود بدین ترتیب با تکرار این مراحل مرتباً مسئله کوچک می شود تا هنگامی که به حالت n-1 برشد ، برای این حالت خاص نیز مسئله به راحتی با انتقال آن دیسک از میله مبدأ به مقصد حل می شود .

```
Void Tower ( int n , char A , char B , char C )
{
        if ( n == 1 ) printf ( " move a disk form %c to %c \n " , A , C ) ;
        else {
        Tower ( n -1 , A , C , B ) ; // الف

        printf ( " move a disk form %c to %c \n " , A , C ) ; // ب

        Tower( n -1 ) , B , A , C ) ; //ج

}

}
```

در واقع اجرای دستورات الف و ب و ج معادل شکل های زیر هستند : (مثلاً برای n = 4)



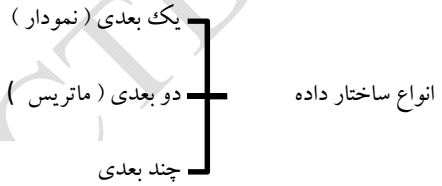
نوع داده انتزاعی (ADT[1])

یک مدل ریاضی است که عملیات بر روی آن مدل تعریف شده اند هر نوع داده تشکیل از چند مقدار و مجموعه ای از عملیات بر روی آنها است . مجموعه مقادیر و عملیات یک نوع داده یک ساختار ریاضی را تشکیل می دهد که ممکن است به کمک یک ساختمان داده سخت افزاری یا نرم افزاری پیاده سازی شود .

آرایه[2]

آرایه ساده ترین ساختار داده ای در زبان C++ است و آرا یه یک بعدی مجموعه مرتب و محدودی از عناصر هم گن است .

[1 , 2 , 3 , 4 , 5]



اعلان آرایه در C++ :

نوع Name [بعد] ;

مثال : int A [10]

نوع Name [capacity بعد] = { مقادیر اولیه } ;

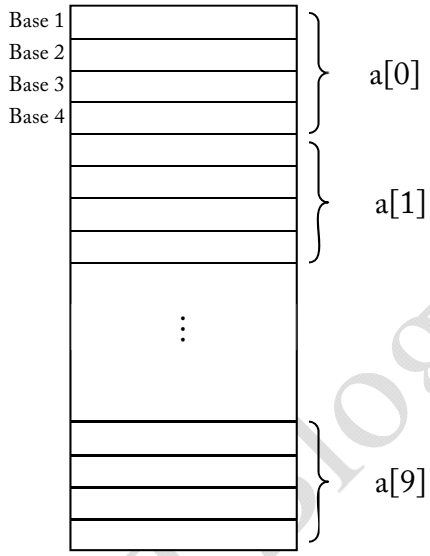
مثال : int B [10] = { 1,2,3,4,5,6,7,8,9,10 } ;

اندیس	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
B	۱	۲	۳	۴	۵	۶	۷	۸	۹	۱۰

1. Abstract Data Type (ADT)
2. Array

پیاده سازی آرایه یک بعدی

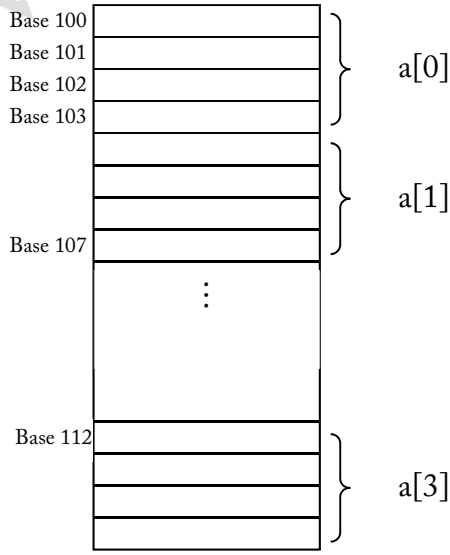
int a[10] ;



مثال

فرض کنید آرایه x به صورت int x [10] تعریف شده باشد base (x) = 100 باشد اگر مقادیر صحیح چهار بایت از فضای حافظه را اشغال کند، آدرس x [3] چیست ؟

Base (x) + i * size
= 100 + 3 +4
= 100 + 12 = 112



جستوجو در آرایه ← خطی / دودوئی

کلید جستوجوی key نام خاص یک دانشجو در لیست دانشجویان

جستوجوی خطی[1]

در روش جستوجوی خطی آرایه را از ابتدا تا رسیدن به عنصر مورد نظر (کلید جستوجو) جستوجو میکنیم . عبارت دیگر ابتدا مقدار key (کلید جستو جو) را با A_0 مقایسه می کنیم . طول آرایه هشت می باشد یعنی هشت عنصر قرار می گیرد .

A	a	b	c	d	e	f	j	h

در صورتی که key = A. باشد آنگاه عنصر مورد نظر پیدا شده است .

در غیر این صورت مقدار key را با A_1 مقایسه میکنیم اگر A_1=key باشد آنگاه عنصر مورد جستوجو پیدا شده است در غیر این صورت key را با $A_1,A_2,\ldots,A_n$ مقایسه میکنیم تا به مقدار مورد جستوجو برسیم .

نحوه نوشتن برنامه جستوجوی خطی

جستوجوی ترتیبی در آرایه های به کار می رود که مرتب نیستند یعنی نه صعودی یا نه نزولی .

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

1. Linear search

جستوجوی دودوئی[1]

جستوجوی دودوئی تنها برای آرایه های مرتب قابل استفاده است . فرض کنید $a=\{a_0,a_1,\ldots,a_n\}$ آرایه ای به طول n باشد آرایه ای از مرتب می گوئیم که در این آرایه را مثل $a=\{a_0,a_1,\ldots,a_n\}$ مرتب می کنیم . $a_0\le a_1\le\ldots\le a_{n-1}$

نکته

الگوریتم جستوجوی خطی مبنای بسیاری از اعمال رایج برروی آرایه است به عنوان مثال در اعمالی مانند کپی برداری از آرایه و بررسی تساوی در آرایه از روشی مشابه جستوجوی خطی استفاده می شود .

فرایند جستوجو دودوئی به شرح زیر می باشد .

۱. عنصر وسط را پیدا کند .
۲. کلید جستوجو را با عنصر وسط لیست مقایسه کند (سه حالت زیر پیش می آید)

۱. مقدار جستو جو با عنصر وسط برابر است . در این صورت عنصر مورد نظر پیدا شده و الگوریتم خاتمه می یابد .

۲. مقدار جستوجو بزرگتر از عنصر وسط است . در این صورت با توجه به ترتیب صعودی عناصر لیست می توان عمل جستوجو را فقط برای نیمه راست ادامه داد .

۳. مقدار جستوجو کوچکتر از عنصر وسط است . در این حالت می توان عکل جستوجو را فقط برای نیمه چپ ادامه داد.

۳. همین روش را برای نیمه چپ یا راست تکرار کنید.

1. Binary search

الگوریتم برنامه

```
int Binsearch ( int a[] , cons int n , int key )
{
        int mid ;
        left = 0 ; right = n-1 ;
        while ( left <= right )
        {
        mid = ( left + right ) / 2 ;
        if ( key == a[ mid ] ) return mid ;
        else if ( key < a[ mid ] ) right = mid-1 ;
        else left = mid+1 ;
        }
return -1 ;
}
```

یافتن بزرگترین و کوچکترین عنصر آرایه :

روش کار با این روش : برای یافتن کوچکترین ابتدا اولین عنصر آرایه را به عنوان کوچکترین عنصر در نظر می گیریم و آن را در متغیری مانند min ذخیره میکنیم سپس عناصر بعدی را با متغیر min مقایسه میکنیم هر عنصری که کوچکتر از min باشد آنرا در متغیر min جایگزین میکنیم واضح است که در پایان کوچکترین عنصر در متغیر min قرار خواهد داشت . الگوریتم برنامه به شرح زیر می باشد .

```
int findmin ( int a [] , cons tint n )
{
        int min = a[0]
        for ( int i=1 ; i<n ; i++ )
        if ( a [i] < min )
        min = a [i] ;
        return min ;
}
```

درج و حذف عناصر آرایه :

فرض کنید آرایه ای به طول k عنصر باشد و شامل n عنصر معتبر $\{ a_0 , a_1 , a_2 , \dots , a_n \}$ باشد برای انجام عملیات درج و حذف در این آرایه باید نکات زیر را در نظر بگیریم .

۱. شرط پر بودن آرایه این است که $n = k$ باشد که در ابن صورت نمی توان عمل درج دیگری در آن انجام داد .

۲. شرط خالی بودن آرایه این است که $n= 0$ باشد که در این صورت عمل حزف امکان پذیر نیست .

۳. برای درج یک عنصر ئر مکان pos از آرایه در صورت پر بودن آرایه باید تمام عناصری که در مکان pos تا n-1 قرار دارد یک مکان به راست انتقال یابد ، سپس عنصر جدید در مکان pos از آرایه قرار گیرد . در این عمل ، اندیس تمام عناصری که در مکان pos تا n-1 قرار دارند ، یکی اضافه می شود . همچنین یک واحد به متغییر n اضافه می شود .

۴. برای حذف عنصری از آرایه که در مکان pos+1 تا k-1 قرار دارد یک مکان به چپ انتقال یابد در این عمل اندیس تمام عناصری که در مکان pos+1 تا k-1 قرار دارند کم می شود . همچنین یک واحد از متغیر n کم می شود .

برای انجام عملیات درج و حذف در آرایه توابع چهارگانه زیر نیاز است :

۱. تست پر بودن آرایه : این تابع در صورت پر بودن آرایه مقدار ۱ و ئر غیر این صورت مقدار ۰ بر می گرداند .

```
int IsFull () ;
{
        if ( k == n ) return 1 ;
        else return 0 ;
}
```

۲. تست خالی بودن آرایه : این تابع در صورت خالی بودن آرایه مقدار ۱ و در غیر این صورت مقدار ۰ را بر می گرداند .

```
int IsEmpty()
{
        if ( n == 0 ) return 1;
        else return 0 ;
}
```

۳. درج عمصر در آرایه : این تابع عنصر x را در موقعیت pos از آرایه درج می کند .

```
void Insert  ( int pos , int x )
{
        if ( IsFull() ) cuot << " Array is full. "  ;
        else if ( pos < k-1 || pos , 0 )
        return ; // In Valid pos
        else if ( pos > n-1 ) { a [n] = x ; n++ }
        else
{
        for ( int i = n-1 ; i >= pos  ; i -- )
        a [i+1] = a [i] ; // Shift Right
        a [pos] = x ; // Insert x
} // End else
}
```

۴. حذف عنصر از آرایه : این تابع عنصری که در مکان pos از آرایه قرار دارد حذف می کند و عنصر حذفی را در متغیر x قرار می دهد .

```
viod Delet ( int pos , int &x )
{
        if ( IsFull() ) cout << " Array is empety " ;
        else if ( pos >= n-1 || pos < 0 ) return ;
        else
{
         x =a [pos];
        for ( int i = pos + 1 ; i <= n-1 ; i ++ )
        a [i+1] = a [i] ; // shift left
        n-- ;
} // end else
}
```

تبدیل آدرس منطقی به آدرس فیزیکی

آرایه یک بعدی

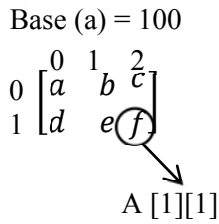
A [1] آدرس عنصر: base (a) + i × size

A [2] آدرس عنصر : ۱۰۰ + ۲ × ۲ = ۱۰۴

آرایه دو بعدی

A [i][j] base (a) + i × size × n + size

۱۰۰ + ۱× ۲ × ۲ × ۳ + ۱ = ۱۰۸



آرایه دو بعدی

آرایه دو بعدی ساختار داده ایاست که عناصر را با دو بعد شناسائی (بعد سطر و بعد ستون) می نماید و به این شکل تعریف می شود .

نوع نام آرایه [m] [n]

int B [3] [4] ;

در زبانهای برنامه نویسی عناصر آرایسه های دو بعدی و بالاتر به صورت سطری یا به صورت ستونی در حافظه ذخیره می شود در زبان C++ به صورت سطری ذخیره می شود .

ماتریس

آرایه دوبعدی است که در شاخه های مختلف علوم ریاضی ، فیزیک ، اقتصاد ، مدیریت و مهندسی کامپیوتر کاربرد دارد .

ماتریس اسپارس[1]

در بسیاری از کاربرد های علمی مانند حل معاملات خطی معادلات دیفرانسیل و مسائل برنامه ریزی خطی با ماتریس های بزرگی سر و کار داریم که تعداد زیادی ار عناصرشان صفر است اینگونه ماتریس اسپارس یا خلوت می نامند .

برای رسم ماتریس نمایش اسپارس به ابعاد ۳ (n+1) که تعداد n تعداد عناصر n_0 است .

$$\begin{bmatrix} 100 & 100 & 5 \\ 0 & 0 & 1 \\ 81 & 0 & 3 \\ 90 & 50 & 2 \end{bmatrix} \quad \begin{matrix} 0 \\ 81 \\ 90 \end{matrix} \quad \begin{matrix} 0 & 50 \end{matrix} \quad \begin{bmatrix} 1 & 0 & 0 & 0 \\ . & . & . & . \\ . & . & . & . \\ 3 & 0 & 0 & 0 \\ . & . & . & . \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

نکته

در ماتریس تذانه هادی سطرها را در ستون می نویسیم .

نمایش ماتریس ترانه هادی

۱۰۰	۷۰	۳
۰	۰	۲
۲	۲	۳

⇒

۷۰	۱۰۰	۳
۰	۰	۲
۲	۲	۳

1. Spase

تعیین ترانه هاده ماتنریس اسپارس از روی جدول ایندکس (نمایش ماتریس اسپارس)

روش الگوریتم

۱. با توجه به ماتریس نمایش در سطر اول (سطر عنوان) جای تعداد سطر ها و ستون ها را عوض می کنیم و در ماتریس ترانه هاده می نویسیم .

۲. در ماتریس نمایش در ستون وسطی به دنبال اعداد صفر گشته و در صورت پیدا کردن آنرا در ستون اول ماتریس ترانه هاده می نویسیم . سپس در ماتریس نمایش در ستون ئسطی به دنبال اعداد یک می گردیم و آن را در ستون ماتریس ترانه هاده می نویسیم .

۳. برای هر کدام از اعداد قرار داده شده در ستون اول ترانه هاده ، اعداد متناظر در ستون اول و سوم ماتریس نمایش را در ستون دوم و سوم ماتریس ترانه هاده می نویسیم

$$\begin{matrix} 0 & 3 \\ 6 & 0 \\ 0 & 0 \end{matrix}\left(\begin{matrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{matrix}\right)$$

۳	۵	۳
۰	۱	۳
۱	۰	٦
۳	۳	۲

در معکوس ماتریس به ستون نگاه می کنیم و از صفر پیدا کرده و معکوس آن را می نویسیم .

ماتریس های مثلثی

$$\begin{bmatrix} * & 0 & 0 & 0 \\ * & * & 0 & 0 \\ * & * & * & 0 \\ * & * & * & * \end{bmatrix}$$

پایین مثلثی

$$\begin{matrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 5 & 6 & 7 \end{matrix}$$

برای نمایش ماتریس های مثلثی دو روش وجود دارد :

نحوه ذخیه سازی ماتریس ها مثلثی (پایین و بالا مثلثی) به دو شکل صورت می گیرد :

۱. استفاده از آرایه دو بعدی

۲. استفاده از آرایه یک بعدی و یافتن فرمولی جهت آدرس دهی

روش اول

از لحاظ مصرف حافظه بهینه نیست زیرا تقریباً نیمی از عناصر ماتریس صفر هستند که نیازی به ذخیره سازی آنها نیست و لی در این روش برای آنها حافظه اختصاص می یابد . در روش اول تعداد کل عتاصر برابر است با n^2 (فرض که ماتریس n*n باشد) حداکثر تعداد عناصر صفر .

حداکثر تعداد عناصرنا صفر = $1 + 2 + 3 + \dots + n = \frac{n\,(n+1)}{2} = \frac{3\,(3+1)}{2} = 6$

حداقل تعداد عناصر صفر = $n^2 - \frac{n\,(n+1)}{2} = \frac{n\,(n+1)}{2}$

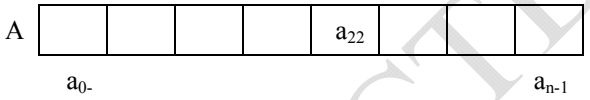
روش دوم

استفاده از آرایه یک بعدی و یافتن فرمولی برای آدرس دهی آرایه .

مثال :

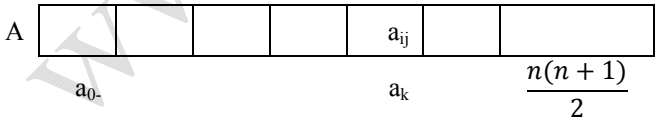
درصد استفاده ثاقعی از حافظه در این روش چقدر است ؟

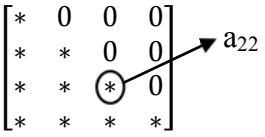
$$\frac{\frac{n(n+1)}{2}}{n^2} \times 100 = \frac{n+1}{2} \times 100 = \frac{11}{10} \times 100 = 55\%$$



در روش دوم برای صرفه جوئی در مصرف حافظه تنها عناصر غیر صفر ماتریس[1] مثلثی ذخیره می شوند این عناصر به ترتیب سطری[2] در یک آرایه یک بعدی قرار می گیرند بنابراین برای عناصر صفر موجود در بالای قطر اصلی حافظه تخصیص نمی یابد .

در این روش باید فرمولی جهت آدرس دهی عناصر ماتریس تعیین شود به عبارن دیگر با آدرس دهی دو بعدی به فضای یک بعدی برده شود .





i سطر

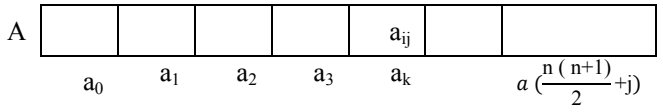
j ستون

k آدرس

1. Matrix
2. Row major order

برای این منظور کافی است تعداد عناصر ذخیره شده قبل از a_{ij} را به دست آورد . با توجه به اینکه تعداد عناصر قبل از عنصر aij برابر مجموع عناصر سطر های قبلی و تعداد عناصر قبلی در سطر i ام است .

$$k = (1 + 2 + \ldots + i) + j \quad \Rightarrow k = \frac{i (i+1)}{2} + j$$



$$\begin{bmatrix} * & 0 & 0 & 0 \\ * & * & 0 & 0 \\ * & * & * & 0 \\ * & * & * & * \end{bmatrix}$$

جند جمله ای

$$p(x) = a_n x^n + a_{n-1}\, x_{n-1} + \ldots + a_1 n_1 + a_0$$

دو روش برالی نمایش چند جمله ای[1] ها وجود دارد :

۱. آرایه یک بعدی
۲. آرایه دو بعدی

در روش اول

آرایه یک بعدی به طول n+1 تعریف می کنیم و ضریب هر جمله را از چند جمله ای به شکل x^i ها در اندیس i ام از این آرایه ذخیره می کنیم .

float p [n+1] ;
$p(x) = 5x^4 + x^2 + 2\times x^0$
float p[5];

2	0	1	0	5
0	1			9

1. Polynomial

در روش دوم

آرایه دو بعدی شامل دو سطر و n سطون تعریف می کنتیم (n تعداد جملات عنصر صفر است) سپس در سطر اول توان های جملات عنصر صفر را به ترتیب نزولی توان ها ذخیره می کنیم . پس از آن در سطر دوم ضرایب متناظر با توان های ذخیره شده را ذخیره می نمائیم .

$p(x) = 5x4 + x2 +2$

float p[2][3]

4	2	0

توان

رشته[1]

دنباله ای متهاهی از کاراکتر ها را رشته نامیده به عبارت دیگر آرایه ای از کاارکتر ها را رشته گویند .رشته حالت خاصی آرایه است تعریف و عملیات برئی آن نیز مشابه با آرایه انجام می شود در C++ انتهای رشته با کاراکتر خاصی بنام '\0' ختم می شود . برای تعریف رشته ای به طول n به صورت زیر عمل می شود

char نام رشته [n];

مثال

رشته ای به طول ۸ کاراکتر و با نام str ایجاد و مقدار Iran را در آن ذخیره کنید .

char str[8];
char str[8]="Iran";

I	r	a	n	'\0'	?	?	?
0	1	2	3	4	5	6	7

روش های مختلف تطبیق الگو های رشته (الگو های مختلف تطبیق رشته) نام برد و الگوریتم هر یک را بنویسید ؟

s = " abbccbbabcbac"
fat = "abc"

1. String

پشته[1]

ساختاری است ترکیبی که در آن درج و حذف عناصر تنها از یک سمت آن انجام می شود . سمتی از پشته که درج و حذف عناصر از آن سمت ثورت می گیرد و بالای پشته[2] نامیده می شود . اگر $s=\{a_0 , a_1 , \dots , a_{n-1}\}$ یک پشته باشد a_0 عنصر پائینی و a_{n-1} عنصر بالایی است و a_i در بالای عنصر a_{i-1} قرار می گیرد .

مثال

پارکینگ یک طرفه

a	b	c	d

پیاده سازی پشته

A. ساختار داده ک که شامل موارد زیر است .

۱. مقدار ثابت Maxsize جهت تغیر حداکثر طول پشته .

۲. متغیر Top ، با مقدار اولیه "-1" جهت مشخص کردن اندیس عنصر بالای پشته

۳. آرایه ای به طول Maxsize که آن را Stack می نامیم . به عنوان

مثال

آرایه از نوع صحیح .

```
cons int Maxsize = 10;
int top = -1 ;
int Stack[maxzise];
```

B. توابع چهار گانه جخت انجام اعمال درج و حذف در پشته :

۱. تابع IsFull : جهت بررسی پر بودن پشته

شرط پر بودن پشته این است که متغر Top برابر باشد Maxsize-1

```
Top=maxsize-1;

int IsFull(){
        if(Top=Maxsize-1)
        return 1 ;
}
```

1. Stack
2. Top of stack

۲. تابع IsEmpty جهت بررسی خالی بودن پشته :

شرط خالی بودن پشته این است که Top برابر -۱ باشد.

```
Top = -1

int IsEmpty(){
        if(Top=-1)
        return 1 ;
        else if
        return 0 ;
}
```

۳. تابع Push برای درج عنصر x در پشته است .

ابتدا یک واحد به متغیر Top اضافه می شود سپس مقدار x را در موقعیت جدید

```
void Push(int x)
{
        if (Isfull())
        cout<<"stack us full. " ;
        else
        Stack [++Top]=x;
}
```

۴. تابع Pop : جهت انجام عمل حذف از پشته :

الف) ابتدا باید عنصر های پشته یعنی Stack[Top] را در متغیر مرجع x قرار داد .

ب) سپس باید Top را یکی کاهش داد .

```
void Pop(int &s)
{
        if(IsEmpty())
        cout << "Stack is Empty " ;
        else {
        x = Stack[Top--];
        return x ;
}
```

متغیر ورودی &x ، متغیر نوع مرجع نام دارد و مانند ارسال از ارجائی پارامتر ها به تابع است x_0 یک متغیر صحیح بوده و &X آدرس متغیر x در حافظه است .

کاربرد های پشته

۱. فراخوانی توابع در برنامه
۲. ارزیابی عبارتمحاسباتی و منطقی
۳. تبدیل مبنای عدد
۴. تشخیص تقارن در پشته ها

روش های مختلف نمایش عبارات محاسباتی و ریاضی :

۱. روش میانوندی یا infix : در این روش هر عملگر بین عملوندهای خود قرار می گیرد .مثل a+b
۲. روش پسوندی یا postfix : در این روش عر عملگر بعد از عملوند های خود قرار می گیرد مثلاً *ab نمایش پسوندی برای عبارت a*b است .
۳. روش پیشوندی یا prefix : در این روش عملگر قبل از عملوند های خود قرار می گیرد به عنوان مثال ab* نمایش پیشوندی برای عبارت a*b است

مثال

عبارت میانوندی a*b/c را به صورت پسوندی و پیشوندی نمایش دهید .

postfix : ab*c/
prefix : /*abc

نشانه های عبارت میانوندی را از چپ به راست مورد بررسی قرار می دهیم :

نشانه ها عبارت اند از : عملگرها و عملوندها ، پارانتز باز و بسته و یا نشانه پایان رشته .

a|b-c+(d*e)/f '/0'

با توجه به نوع نشانه دیده شده حالات زیر وجود دارد:

۱. اگر نشانه عملوند است مستقیم به خروجی فرستاده شود .
۲. اگر نشانه پارانتز بسته است عناصر پشته را تا رسیدن به اولین پارانتز به خروجی می فرستیم .
۳. اگر نشانه پارانتز باز است بلا فاصله آن را در بالای پشته درج کنیم
۴. اگر نشانه عمگر است عملگر جدید را با عملگر بالای پشته (در صورت وجود) مقایسه میکنیم که دو حالت زیر پیش خواهد آمد :

الف) اگر پشته خالی باشد یا اولویت عملکر جدید بیشتر باشد عملگر در پشته درج شود .

ب) اگر عملگر جدید دارای اولویت برابر یا پایین باشد عناصر پشته را تا رسیدن به پارانتز باز و یا عملگری که اولویت آن از اولویت آن از اولویت عملگر جدید پائین تر است خارج کنیم و به خروجی ارسال کنیم سپس عملگر جدید را در پشته درج نمائیم .

مثال

تبدیل عبارت میانوندی

a*(b+c)/d

نشانه	پشته	خروجی
a		a
*	*	a
(	*(	a
b	*(	a b
+	*(+	a b
c	*(+	a b c

)	*(	abc+
/	/	abc=*
d	/	abc+*d

الگوریتم ارزیابی یک عبارت پسوندی : منظوراز ارزیابی یک عبارت محاسبه حامل آن به ازای مقادیر عملوندهای آن است.

۱. نشانه های عبارت را ازچپ به راست مورد برسی قرار دهیم.

۲. با توجه به نوع نشانه سه حالت وجود دارد:

الف) اگر نشانه عملوند است آن را در بالای پشته درج کنیم.

ب)اگر نشانه یک عملگر است دراین صورت باتوجه به اینکه عملگر دو عملوندی یا تک عملوندی باشد.به ترکیب دو یا یک نشانه از پشته خارج شده به عملگر مورد نظر بر روی آنها اعمال شود نتیجه در پشته درج می شود.

ج) نشانه کاراکتر پایان رشته است . معنی آن این است که ارزیابی عبارت به پایان رسیده است . دراین حالت پشته باید تنها دارای یک مقدار باشد که جواب مسئله است . با Pop نمودن این مقدار باید پشته خالی شود.

نحوه ارزیابی عبارت پسوندی : -ab/c-dex+acx باتوجه به الگوریتم بیان شده محاسبه کنید:

راهنمایی : فرض کنید هر بار که مقداری را محاسبه می کنیم آن را در متغیرT ذخیره می کنیم.

نکته

در موقع پوش کردن باید عنصر مشخص باشد ، x مقدار اصلی آن آدرس آن چیست ولی در پاپ کردن لازم نیست آخرین عنصر خودش ظاهر می شود.

نشانه	نوع	عملگر	پشته
a	عملوند	push (a)	a
b	عملوند	push (b)	ab
/	عملگر	Pop $=x_2=b$	
		Pop$=x_1=a$	
		$T_1=a/b$	
		Push$=(T_1)$	T_1

صف[1]

ساختاری است ترکیبی که در آن درج عناصراز آنها وحذف آنها از ابتدا انجام می شود.ابتدا انتهای صف به ترتیب با دو متغیر flaot و rear نشان داده می شود. اگر $Q = \{a_0,a_1,\ldots,a_{n-1}\}$ صفی از عنصر باشد آنگاه a را عنصرابتدایی و a_{n-1} را عنصر انتهایی صف می نامند . بنابراین در این صف عنصر a_1 بعد از عنصر a_1-1 قراردارد .

از تعریف چنین استباط می شود که در صف اولین عنصری که وارد می شود . اولین عنصری است که خارج می شود به همین علت ، صف را می توان به عنوان یک لیست FIFO تلقی کرد .

بر خلاف پشته صف ساختاری دو طرفه است ، ابتدا و انتهای صف با دو متغیر front و rear نشان داده می شود . موقعیت frant بر یک موقعیت عقب تر از یک موقعیت اولین عنصر و rear دقیقاً به موقعیت آخرین عنصر صف اشاره می کند .

شرط خالی بودن صف (N=0) عبارت است از front = rear = -1 باشد که شرط اولیه می باشد .

نکته

با هر عمل درج عنصر در صف یک واحد به متغیر rear اضافه می شود و با هر عمل حذف عنصر یک واحد به متغیر front اضافه می شود .

1. Queue

مقایسه پشته و صف

۱. هر دو ساختار های ترکیبی داشته و حالت ویژه ای از آرایه هستند .

۲. پشته از حالت LIFO[1] ولی صف از حالت FIFO[2] می باشد .

۳. پشته لیستی یک طرفه ولی صف لیستی دو طرفه می باشد .

پیاده سازی صف

a) تعریف ساختار داده که شامل موارد زیر است :

۱. مقدار ثابت Maxsize جهت تعیین حداکثر طول صف .

۲. دو متغیر front و rear با مقدار اولیه ۱- جهت مشخص نمودن اندیس ابتدائی و انتهائی صف

۳. آرایه به طول Maxsize به نام Queue و از نوع صحیح

```
cons tint Maxsize = 10;
int front = rear = -1 ;
Queue[Maxsize];
```

b) توابع چهار گانه زیر جهت انجام اعمال درج و حذف عصر در صف .

۱. تابع IsFull جهت بررسی پر بودن صف

```
int IsFull(){
        if ( rear == Maxsize -1 )
        return 1 ;
        else return 0;
```

۲. تابع IsEmpty جهت بررسی خالی بودن صف .

```
        int IsEmpty( ){
        if  ( front  ==  rear )
        return 1 ;
        else return 0;
}
```

۳. تابع Add جهت در عنصر x در انتهای صف .

```
void Add ( cons tint x ){
        if ( IsFull( ) )
        cout <<  " Queue is Full." ;
        else Queue[ ++rear ] = x;
}
```

---

1. Last In Firest Out (LIFO)
2. First In First Out (FIFO)

۴. تابع Del جهت انجام عمل حذف از ابتدای صف .

```
void Del (int &x)
{
        if ( IsEmpty() )
        cout<<” Queue is Empty”;
        else
        x = Queue[++front];
}
```

اول یک واحد اضافه بعد عنصر واقع در آن به x منتقل می شود . زیرا متقیر Front به یک کوقیعت عقب تر از اولین عنصر اشاره می کند .

پیچیدگی زمانی الگوریتم های درج و حذف در صف دارای زمان ثابت O_1 است چون برای این اعمال از دستورات شرطی و جایگزینی استفاده می شود که زمان ثابتی طول میکشد .

صف حلقوی [1]

صف حلقوی بسطی است که در آن موقسغت های ابتدائی و انتهایی بر هم منطبق هستند .

نکته

در سطح حلقوی اگر rear = Maxsize -1 باشد ولی در امتداد صف تعدادی خانه خالی وجود داشته باشد انگاه عنصر جدید در این خانه ها می تواند درج شود .

یکی از معایب صف خطی است که با انجام عملیات حذف نقطه ابتدائی صف به تدریج و سمت راست تغییر مکان می یابد . به طوری که به تدریج مکانهای ابتدای صف بلا استفاده باقی مانده و نمی تواند عنصر جدیدی در آن درج کرد . چون طبق ماهیت صف عناصر جدید نمی تواند در ابتدای صف درج گردد.

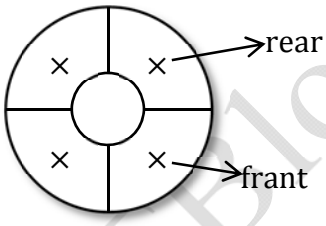
1. Circular Queue

نکات زیر در سطح حلقوی قابل توجه است .

۱. مانند سطح خطی front همیشه به یک موقعیت عقب تر از عنصر اشاره می کند .

۲. مانند صف خطی شرط خالی بودن صف این است که front برابر rear باشد .

۳. در یک صف حلقوی به طول maxsize که با حرف n هم نشان می دهند .حداکثر تعداد عناصر مجاز برابر maxsize-1 است زیرا در غیر این صورت در حالتی که صف پر است تعداد عناصر برابر maxsize و در نتیجه رابطه front=rear است . به عبارت دیگر نمی توان تمایز بین حالات پر بودن و خالی بودن صف قائل شد که این مسئله باعث بروز مشکل در پیاده سازی توابع add و delete می گردد .



۴. در صف حلقوی مقادیر اولیه برای متغیرهای front و rear صفر است .

۵. در صف حلقوی بر خلاف صف خطی شرط پر بودن صف این است که front=(rear+1)%maxsize باشد .

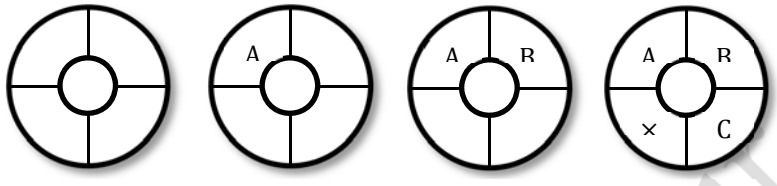
۶. در صورتی که صف پر نباشد برای عمل درج باید به متغیر rear یک واحد اضافه کرده و سپس باقیمانده تقسیم آن را بر maxsize محاسبه نمائیم حاصل محاسبه موقعیت عنصر جدید را جهت عمل درج نشان می دهد .

۷. در صورتی که صف خالی نباشد برای عمل حذف باید به متغیر front یک واحد اضافه کرد و پس از آن باقیمانده تقسیم آن را بر maxsize محاسبه نمود . حاصل محاسبه ، موقعیت عنصری که باید حذف شود نشان می دهد .

۸. هر دو عمل درج و حذف در جهت عقربه های ساعت انجام می شود .

مثال

عملیات درج و حذف در یک صف حلقوی به طول حداکثر سه عنصر را نمایش می دهد دقت کنید که همیشه باید یک خانه از صف خالی باقی بماند .

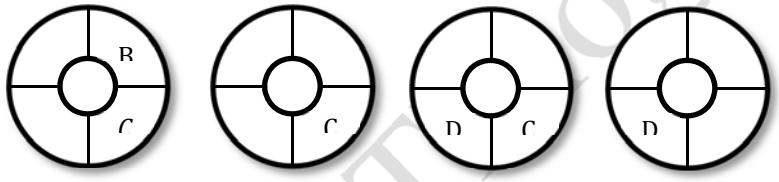


صف حلقوی خالی
f=r=0

درج A
f=0 , r=1

درج B
f=0 , r=2

درج C
f=0 , r=3



حذف A
f=1 , r=3

حذف B
f=2 , r=3

درج D
f=2 , r=0

حذف C
f=3 , r=0

توابع درج و حذف در سطح حلقوی

```
Void add (const int x)
{
    Int k =(rear+1)% maxsize;
    If (front==k)
    Cout<<"Queue is full";
    Els{rear =k ; Queue[rear]=x;}}
```

تابع delete

```
Void delete (int & x)
{
    If (front==rear)
    Cout<<"Queue is empty";
    Else{front++ ; front%=maxsize ;
    X=Queue[front] ; }}
```

کاربردهای صف

۱- در سیستم عامل جهت پردازش فرآیندها

۲- ذخیره سازی کلیدهای فشرده شده در حین اجرای برنامه ها

۳- در مسایل شبیه سازی جهان واقع

۴- در شبکه های کامپیوتری برای استفاده بهینه از منابع شبکه مثل پرینتر

یادآوری از c++

رکورد[1]

رکورد ساختمانی است ترتیبی از عناصر که لزوما هم نوع نیستند به هر عنصر از رکورد یک فیلد گفته می شود .

در C++ رکورد را با کلمه کلیدی struct تعریف می شود .

مثال

تعریف رکورد دانشجو با فیلدهای شماره دانشجویی ، نام و معدل ؟

```
Struct
Student{
Int stno;
Char name [40];
Float arr ; } ;
Student s ;
```

برای دستیابی به هر فیلد از رکورد باید ابتدا نام متغیر رکورد و سپس نام فیلد را ذکر کنیم بین این باید علامت نقطه قرار دهیم :

نام فیلد . نام متغیر فیلد

```
S.stno
S. name
S. avr
```


1. Record

اشاره گر[1]

اشاره گر متغیری است که می تواند آدرس ناحیه ای از حافظه را نگهداری نماید به عبارت دیگر می تواند آدرس متغیر دیگری در حافظه را نگهداری نماید .و به صورت زیر تعریف می شود :

< نوع > * < نام اشاره گر > ;

```
Int   *   p   ;

Float   *   q   ;

Student   *   r   ;
```

نکته

اگر p یک اشاره گر باشد و p* محتوای خانه ای از حافظه را که این اشاره گر به آن اشاره می کند نتیجه میدهند و بر عکس اگر x یک متغیر باشد &x آدرس آن را در حافظه نتیجه میدهد .

مثال :

```
Int  x=5 , y ;
Int  *p  ;
P = &x ;       x آدرس
Y = *p ;       5
Cout<<y;    (5)
```

نکته

اگر p یک اشاره گری به یک رکورد باشد از نماد (فلش) برای دستیابی به هر فیلد از آن رکورد استفاده می شود . اشاره گری که به جایی اشاره نمی کند Null یا تهی نامیده می شود .

dدستورات حافظه پویا[2] : پویا = استاتیک

برای کار با حافظه پویا در C++ از توابع New و Delete استفاده می کنیم .

تابع New

این تابع برای تخصیص حافظه به اندازه معین به کار میرود . تابع New پس از تخصیص حافظه آدرس ابتدای حافظه تخصیص داده شده را برمیگرداند . خروجی تابع در صورت عدم موفقیت مقدار Null خواهد بود.

1. Pointer
2. Dynamic memory alocation

تابع Delete

از این تابع جهت آزادسازی حافظه ای که قبلا تشخیص داده شده است استفاده می شود .

مثال :

```
Int  *p ;
  P = new  int ;
.
.
.
Delete  p ;
```

مثال

```
Student  *s  ;
S = new student ;
.
.
.
Delete  s ;
```

مثال

```
Int  *a ;
A = new int [10] ;
.
.
.
Delete [  ] a ;
```

لیست پیوندی : ساختارها

آرایه

پشته

صف

ترتیبی + منطقی و فیزیکی

معایب ساختارهای ترتیبی

۱- نیاز به حجم انتقال زیادی از داده ها در عملیات درج و حذف دارند .

۲- اغلب تخصیص حافظه در آنها ایستا[1] است و باعث هدر رفتن حافظه می گردد.

لیست پیوندی[2]

لیست پیوندی ساختاری است غیر ترتیبی از عناصر که ترتیب منطقی آنه توسط اشاره گرها تامین می شوند .

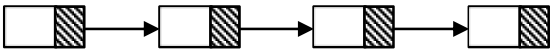
گره[3]

به هر عنصر در لیست پیوندی اصطلاحا گره گفته می شود .هر گره از لیست پیوندی رکوردی با دو بخش داده و آدرس است . فیلدهای آدرس در واقع همان اشاره گرهایی هستند که برای ایجاد ارتباط منطقی بین گره های لیست مورد استفاده قرار می گیرد .

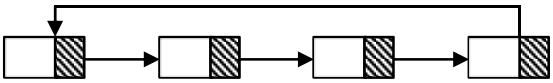
10	20	30	40
p	p+1	p+2	p+3

نمایش ترتیبی لیست پیوندی

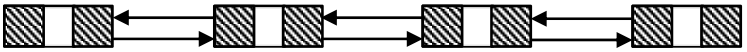
1. Static
2. Linked list
3. Node



لیست تک پیوندی[1] (یک طرفه)



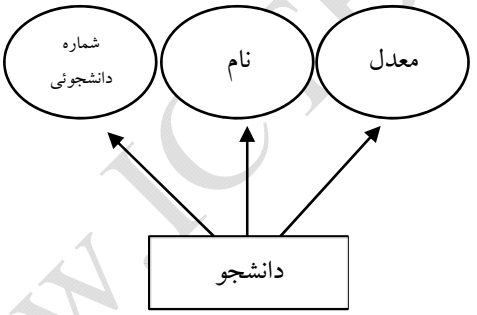
لیست حلقه ای[2]



لیست دوپیوندی[3] (دو طرفه)

مثال

برای ذخیره سازی اطلاعات دانشجویان در لیست یک طرفه حلقوی و دوطرفه ساختارهای زیر تعریف می شوند : فرض کنید هر دانشجو دارای فیلدهای شماره دانشجویی ، نام و معدل می باشد :



شماره دانشجویی	نام	معدل
۱۰۰	رضا شیری	20
۱۰۱	حمید وحدت	20
۱۰۲	فرهاد کاظمیان	21

1. Single linked list
2. Circular linked list
3. Doubly linked list

مثال

برای لیست تک پیوندی و حلقوی :

```
Struct  student {
Int  stno ;
Char  name [40] ;
Float  avr ;
Student  *naxt ;
};
```

برای لیست دو طرفه :

```
Struct  student{
Int  stno ;
Char  name [40] ;
Float  avr ;
Student  *next , *prev ;
};
```

توجه

در لیست پیوندی همیشه باید آدرس اولین گره از لیست را نگهداری نمدو . چون در غیر این صورت لیست در حافظه گم می شود و قابل دستیابی نیست . معمولا آدرس اولین گره از لیست را به صورت سراسری در متغیری به نام first یا head نگهداری می کند واضح است که با داشتن آدرس اولین گره از لیست پیوندی بقیه گره ها نیز قابل دستیابی هستند به اشاره گرهایی که به صورت یک فیلد در گره های لیست پیوندی ذخیره نمی شوند ، اشاره گر خارجی می گویند . بنابراین اشاره گر first یک اشاره گر خارجی است .

```
# include <iostream . h>
# include<conio . h>
# include<stdlib . h>
Int is full();
Int is empty();
Void push(int x);
Void pop(int &x);
Const int maxsize=8;
Int top= -1;
Void main ( )
{
Int x , I ;
Char option ;
```

```
While (1)
{
Clrscr( );
Cout <<"\n\t1)push" ;
Cout <<"\n\t2)pop" ;
Cout <<"\n\t3)dis play" ;
Cout <<"\n\t4)exit" ;
Cout <<"\n\t ) enter your option" ;
Option = getche( );
Switch(option){
Case '1' : cout<<"\n enter x" : ;
Cin>>x ;
Push(x);
Break ;
Case '2' : if(! Is empty (1))
{
Pop(x);
Cout<<"\n"<<x<<"is deleted" ;
}
Getch( );
Break ;
Case '3' : cout<<"\n" ;
For(i=top ; i>=0 ; I--)
Cout<<stack[i]<<"  " ;
Getch( );
Break;
Case '4' exit (1);
} // end switch
}// end main
Int is full( )
{
If (top ==maxsize)
Return 1;
Else
Return 0 :}
```

```
Int is empty( )
{
If (top= = -1)
Return 1;
Else
Return 0 ;}
Void push(int x)
{
If(is full( ) )
Cout<<"\n stack is full" ;
Else
Stack[+ + top] = x ;}
```

```
Void pop(int &x)
```

```
{
If(is empty)
Cout<<"\n stack is empty" ;
Else
X = stack[top - -] ;}
}
```

مثال

```
#include<iostream.h>
Void f1(int num [5]);
Void f2(int num[ ]);
Void f3(int *num);
Main( )
{
Int count[5]={1,2,3,4,5}
F1(count);
F2(count);
F3(count);
Return 0;
}
// sized array
Void f1(int num[5])
{
Int I ;
For(i=0 ; i<5 ; i+ +)cout<<num[i]<<"  " ; }
//nusized avray
Void f2(int num[ ])
{
Int I ;
For (i=0 ; i<5 ; i+ +)cout<<num[i]<<"  " ; }
//pointer
Void f3(int *num)
{
int I ;
For(i=0 ; i<5 ; i+ +)cout<<num[i]<<"  " ; }
```

اصول طراحی و برنامه نویسی شی گرا [1]

دو روش برای طراحی یک برنامه وجود دارد :

۱- روش ساخت یافته

۲- روش شی گرا

1. Object oriented

در هر دو روشاز نقطه نظر تقسیم و غلبه[1] برای تجزیه مسئله و حل آن استفاده می شود . در روش تقسیم و غلبه یک مسئله پیچیده به تعدادی زیر مسئله ساده تر تقسیم می شود . سپس هر یک از این زیر مسئله ها به طور جداگانه حل می شود . تفاوت این دو روش در نحوه تجزیه مسئله است .
بنابراین شباهت هر دو روش در تجزیه مسئله اصلی به مسائل کوچکتر است .

نکته

در روش ساخت یافته از روش تجزیه تابعی و در روش شی گرا از روش تجزیه شی استفاده می شود . در روش تجزیه تابعی مسئله بصورت سلسله مراتبی به تعدادی زیر برنامه مستقل ماژور[2] (در روش ساخت یافته بکار می رود) تقسیم می شود . از خصوصیات اصلی چنین روشی مجزا بودن الگوریتم های مورد استفاده در طراحی زیر برنامه ها از ساختارهایداده برنامه است . که این خود باعث کاهش انسجام و انعطاف پذیری برنامه می شود .اما روش شی گرا به نرم افزار به دید یک سیستم[3] می نگرد (سیستم مجموعه ای از اجزای مختلف است که با هم تعامل دارند و در کل (سیستم) هدفی را دنبال می کند) مثال : اتومبیل ، بدن انسان

مثال برای روش ساخت یافته

مطلوبست محاسبه متوسط سه نمره ، هر نمره وزن خاصی دارد داده ها روی فایل ورودی به نام score file ذخیره شده اند در هر خط این فایل یک نمره و وزن نظیر آن داده شده است .

آماده سازی فایل برای خواندن ورود اطلاعات ، پرینت اطلاعات یا خرج ، یافتن وزن متوسط

(مرتبه صفر) ماژور اصلی main modul

Prepar file for reading
Get data
Print data
Find weighted average
Print weighted average

(مرتبه یک) فایل را برای خواندن آماده کنید Prepar file for reading

Open score file

1. Divide and conquer
2. Module
3. System

گرفتن داده get data

Read test 1 , weight 1 from score file . . . Read test 3 , weight 1 from score file

چاپ داده print data

Print heading Print test 1 , weight 1 . . .

پیدا کردن میانگین وزن find weighed aver

Set ave=test 1 *weight1+ Test 2 *weight 2+test 3 *weigh 3

چاپ میانگین وزن print weighted average

Print blank line Print "weighted avearg="ave

چاپ عناوین print heading

Print "test score weight

که شامل مجموعه ای از اشیاء است که با یکدیگر ارتباط متقابل و تعریفشده ای دارد در این روش ابتدا سیستم به اشیاء[1] تجزیه شده و پس از آن در صورت لزوم از تجزیه تابعی نیز (روش ساخت یافته) استفاده می شود . روش شی گرا باعث افزایش انعطاف پذیری سیستم ، انسجام برنامه ومدل سازی طبیعی آن می گردد .

تعریف شی یا object

ساختاری متشکل از داده ها یا روال ها را یک شی می نامند .

1. Object

مثال شی گرایی

برنامه مدیریتی امور آموزشی در دانشگاه :

اشیاء محیط دانشگاه : ۱- دانشجو ۲- درس ۳- معلم ۴ – حساب بانکی ۵- خوابگاه

تعریف برنامه نویسی شی گرا

به زبانی برنامه نویسی شی گرا گفته می شود که :

۱- از اشیاء پشتیبانی کند .

۲- هر شی به یک کلاس[1] تعلق داشته باشد .

۳- وراثت[2] را پشتیبانی کند

نکته

زبان c شی گرا نیست ولی c++ که بهبود یافته آن است یک زبان برنامه نویسی شی گرا است .

class

Class یعنی طبقه بندی . گفتیم شی شامل داده و کلاس است . کلاس قالب کلی آن شی است . مثلا : قالب شیرینی همان کلاس است مواد مختلفی که استفاده می شود شی است پس کلاس طبقه بندی خاصی از اشیاء است . مثلا : می گوئیم کلاس موجود زنده : هر موجود زنده دارای خصوصیاتی است . موجود زنده نفس می کشد ، رشد می کند . وقتی از این کلاس ، کلاسهای دیگری بوجود می آید . مثل : نباتات ، جانداران دارای مشخصاتی که هر موجود زنده دارا است ذرا دارد در مورد جانداران حیوانات و انسانها را می توانیم نام بریم . در مورد نباتات گیاهان را و از طبقه جانداران طبقه حیوانات شامل می شود که دارای مشخصات ویژه است .

جانداران علاوه بر آنکه رشد می کنند و نفس می کشند راه هم میروند . این طبقه حیوانات داای سایر مشخصات هستند. مثلا: چهارپا هستند . پس ویژگی هایی است که مربوط به طبقه خاص خود است و در طبقات دیگر نیست هر چقدر به طبقات بالا برویم طبقات دارای مشخصات کلی است .از طبقه انسانها می توانیم طبقه مردان یا کلاس زنها را نام ببریم . طبقه مردان شامل ویژگی های خاص خود و ویژگی های انسانها هستند .

1. Class
2. Inheritance

پس در زبان برنامه نویسی شی گرا از این سیستم استفاده می کنند کلاس بندی می کنند . هر شی که در محیط جهان واقع به دیدگاه سیستم نگاه می کنیم و می گوئیم در آن سیستم آبجکت ها هستند که دارای ایفای نقش هستند می آیند این اشیاء را طبقه بندی می کنند یعنی اول برای آنها کلاس درست می کنند این کلاس دارای یک سری مشخصات کلی است . از این کلاس ویژگی خاص آن را پیدا می کنیم . هر طبقه دارای ویژگی مخصوص خودش است پس بحث وراثت از اینجا نشعت می گیرد .

class

ساختاغری است یک نوع تجرد داده را نمایش می دهد .

```
Class circle
{ public:
Circle (int r); //constract
Void setradius(int r);
Void display area(void);
~ circle( ); //destructor
Private:
Float calculate area(void);
Int radius ;
Int color;
} ;
```

نکاتی در تعریف class

۱- هر class باید دارای تابع سازنده[1] (هم نام با نام کلاس) و تابع مخرب باشد .و حتما بعد از public تعریف می شود .

۲- نام تابع سازنده هم نام class است .

۳- نام تابع مخرب با نام تابع سازنده یکسان است اما در ابتدای آن یک علامت ~ دارد .

۴- برای تابع سازنده و مخرب نباید هیچ مقدار بازگشتی تعریف شود .

۵- تابع مخرب[2] هیچ مقدار ورودی نمی گیرد .

۶- هر class فقط شامل پیش تعریف توابع است .

```
Class stack
{
Public:
Stack( );
//-----
Stack : : empty( )private:
Stack : : pop and test( )
```

1. Consteuctor
2. Destructor

درخت [1]

درخت مجموعه ای متناهی از یک یا چند گره است . به طوری که :

۱- دارای گره خاصی بنام ریشه[2] است .

۲- گره ریشه شامل TK , … , T2 , T1 زیر درخت[3] است . (subtvr) . که هر کدام از این زیر درخت ها خود یک درخت هستند .

درخت ساختار داده بازگشتی[4] است . این گونه ساختارها دارای تعریف بازگشتی هستند .

اصطلاحات و مفاهیم مطرح در درخت

۱- ریشه[5] : گره موجود در بالاترین سطح از درخت که بقیه گره ها از آن منشعب شده اند .

۲- درجه گره[6] : تعداد زیر درختان هر گره را درجه آن گره می نامند .

۳- سطح[7] : درخت به عنوان ساختاری سلسله مراتبی دارای سطوحی است . گره ریشه در سطح ۵ درخت قرار دارد .

۴- عمق یا ارتفاع[8] : تعداد سطوح یک درخت را ارتفاع یا عمق درخت می نامند .

۵- گروه فرزند(child): به گره هایی در یک سطح پایین تر از یک گره به طور مستقیم به آن متصل اند ، گره فرزند آن گره گفته می شود .

مثال

گره های b و c وB فرزندان گره A و گره های E وf فرزندان گره B هستند . گره های فرزند همیشه در یک سطح از درخت قرار دارند ولی هر دو گره که در یک سطح از درخت قرار دارند . لزوما فرزندان یک گره نیستند . به عنوان مثال : گره های G,F در یک سطح قرار دارند ولی فرزندان یک گره نیستند .

۶- گره پدر (parent): به گروهی در یک سطح بالاتر از تعدادی گره هم سطح ، که به طور مستقیم به آنها متصل است . گره پدر گفته می شود . به عنوان مثال :گره a پدری برای گره های b , c , d است .

---

1. Tree
2. Root
3. SubTree
4. Recursive data structurenv
5. Root
6. Dogree
7. Level
8. Depth

همین طور که B پدر برای گره های E و F است .

توجه

گره ریشه پدر نام دارد .

۱. گره های همزاد (Sibling) : به گره هایی که از یک سطح ، که فرزندان یک پدر هستند گره های همزاد گفته می شود .

مثال

گره های B , C , D همزادند ، همین طور گره های E , F همزاد یکدیگرند .ام دو گره F , G همزاد نیستند . می توان گفت تمام گره های همزاد در یک سطح قرار دارند ولی تمام گره هایی که در یک سطح قرار دارند لزوماً همزاد نیستند .

۲. درجه دخت : بزرگترین درجه گره های یک درخت درجه آن درخت گفته می شود .

مثال

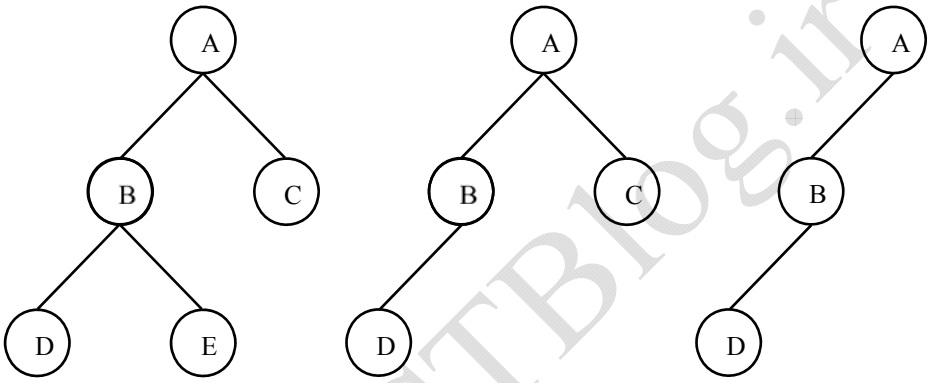
داده شده در شکل از درجه ۳ است به درختی که از درجه k باشد اصطلاحاً درخت k تائی یا درخت عمومی[1] گفته میشود . در حالت خاص به درختی که از درجه ۲ باشد درخت دو دوئی گفته می شود .

۳. گره برگ یا پایانی (Leaf) : به گره هایی که درجه آن صفر است گره برگ یا گره پایانی گفته می شود به عبارت دیگر گره پایانی گرهی است که فرزندی نداشته باشد . اگر گرهی در آخرین سطح درخت قرار داشته باشد گره پایانی است ولی هر گره پایانی لزوماً در آخرین سطح قرار ندارد . گره هایJ, I , M , G , F , L , K در شکل پایانی هستند .

1. General Tree

درخت دودوئی[1]

درخت دودوئی خالت ویژهای ار درختان k تائی است که در آن k = 2 است به عبارت دیگر حداکثر درجه گره ها در یک درخت دودوئی برابر دو است .بنابراین هر گره حداکثر دارای ۲ فرزند است که آنها را فرزند چپ[2] و فرزند راست[3] می نامیم . درخت دودوئی را می توان به طور دقیق تر با تعریف بازگستی زیر بیان کرد .



االف ب ج

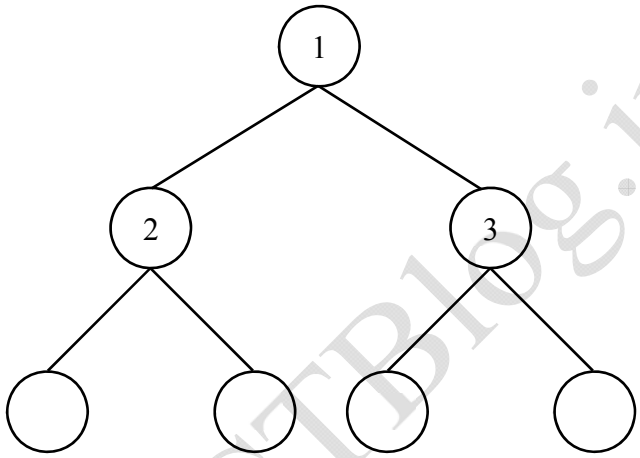
درخت دودوئی

مجموعه متناهی از گره هایست که میتواند تهی بوده . یا شامل یک رشته و دو زیر درخت چپ و راست باشد . به طوری که زیر درخت های چپ و راست خود درخت های دو دویی اند . همانند شکل ج

1. Binary Tree
2. Left child
3. Right child

قضیه

اگر A درخت دودویی نا تهی باشد به طوری که n_0 تعداد گره های پایانی و n_2 تعداد گره های درجه ۲ آن باشد . آنگاه رابطه $n_0 = n_2 + 1$ برقرار است .



درخت دودوئی پر[1]

درخت دودوئی پر درختی است که در آن تمام گره ها به جز گره های سطح آخر از درجه ۲ هستند بنابراین در درخت پر گرهی با درجه ۱ وجود ندارد .

شماره گذاری دیخت ها را توضیح دهید :

۱. گره ریشه دارای شماره یک است (شماره گزاری می تواند از صفر نیز شروع شود)
۲. گره های موجود در سطح دیگر به ترتیب از چپ به راست شماره گزاری می شود .
۳. در هر سطح شروع شماره گزاری از ادامه آخرین شماره در سطح قبلی انجام می گیرد .

قضیه

اگر T درخت دودوئی پر به عمق d باشد آنگاه :

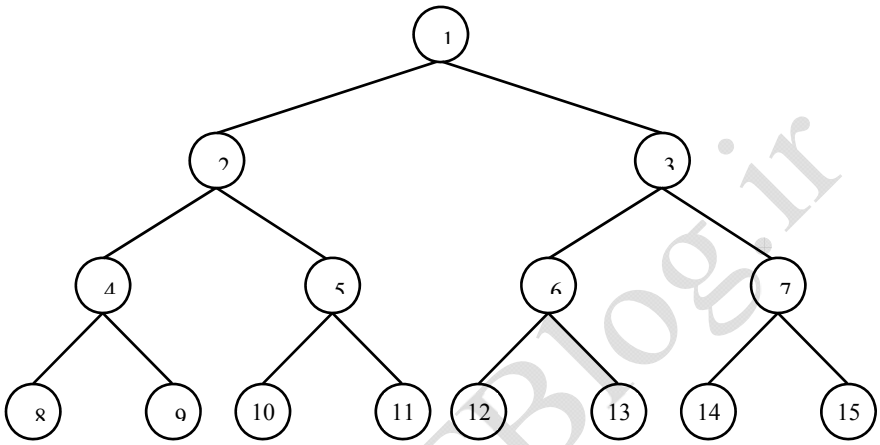
۱. تعداد گره ها در سطح i ام از درخت برابر 2^i . $0 \le i \le d$

1. Completed Tree

۲. تعداد کل گره های درخت برابر 2d-1 است .

مثال

تعداد گره های درخت پر در سطح دو برابر $2^2=4$ و تعداد کل گره ها برابر $2^4-1=15$ است .



نکته

اگر i شماره گرهی از درخت پر باشد . شماره گره پدر ، فرزند راست و فرزند چپ از روابط زیر بدست می آید .

1. parent (i) = [i/2]
2. lchild (i) = 2^i
3. rchild (i) = $2^i + 1$

درخت دودوئی T با n گره و عمق d کامل است اگر و فقط اگر گره های آن مطابق با گره های شماره گذاری شده از 1 تا n در یک درخت دودوئی پر به عمق d باشد .

توجه : هر درخت پر کامل است ولی هر درخت کامل لوزماً پر نیست .

نکته

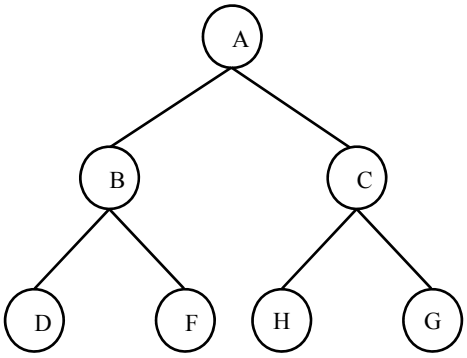
۱. حداکثر تعداد گره های درخت کامل به عمق d برابر $2^d - 1$ است .

۲. حداقل تعداد گره های درخت کامل به عمق d برابر 2^{d-1} است .

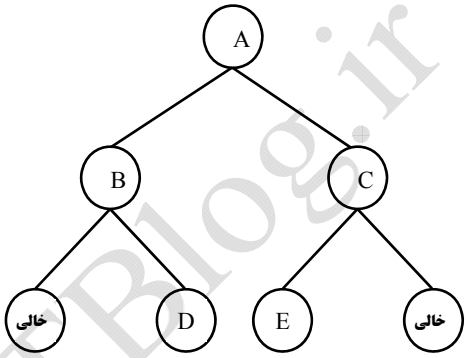
روش های نمایش درخت دودوئی را نام ببرید ؟ روش ترکیبی – روش غیر ترکیبی

روش ترکیبی با استفاده از آرایه را توضیح دهید :

۱. درخت را بر اساس درخت دودوئی پر شماره گذاری نمائید (آخرین شماره را n فرض کنید)
۲. آرایه ای به طول n +1 از نوعی یکسان با ساختاره گره های درخت تعریف نمائید
۳. مقدار داده هر گره از درخت را در مکانی از آرایه که برابر شماره این گره است ذخیره نمائید .



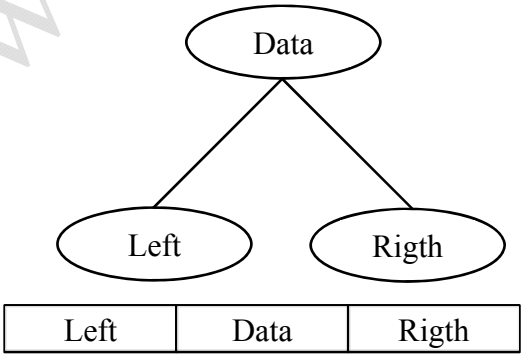
کامل

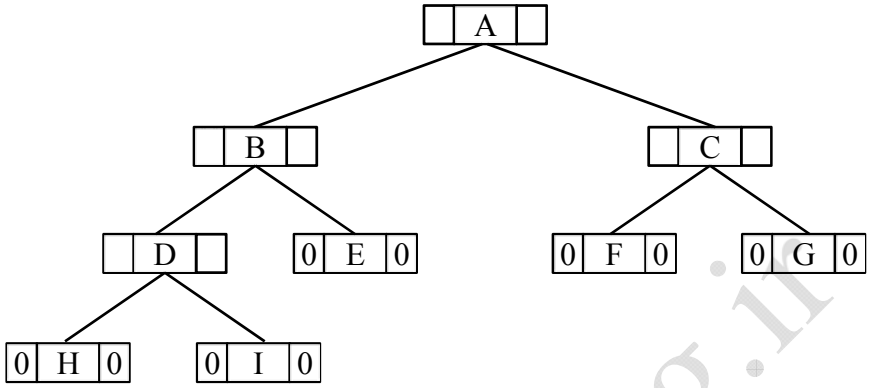


غیر کامل

نمایش پیوندی درخت دودوئی را توضیح دهید :

در این روش از لیست پیوندی با ساختار گره زیر استفاده می شود که در آن سه فیلد left , data , right تعریف میشود که به ترتیب آدرس فرزند راست مقدار داده ای گره و آدرس فرزند چپ را در خود نگداری می کند .





پیمایش درخت دودوئی

یکی از اعمال بنیادی که بر روی درخت ها انجام می شود پیمایش درخت است . حدف از پیمایش درخت دست یابی به تمام گره های آن است به طوری که هر گره تنها یکبار ملاقات می شود . پیمایش در درخت به این علت حائز اهمیت است که مبنائی برای بسیاری اعمال دیگر بر روی درخت دودوئی است .

روش های مختلف برای پیمایش درخت : پیمایش عمقی و پیمایش سطحی

۱. پیمایش عمقی :

در این روش در هنگام پیمایش حرکت در عمق درخت مقدم بر حرکت در سطح درخت است .

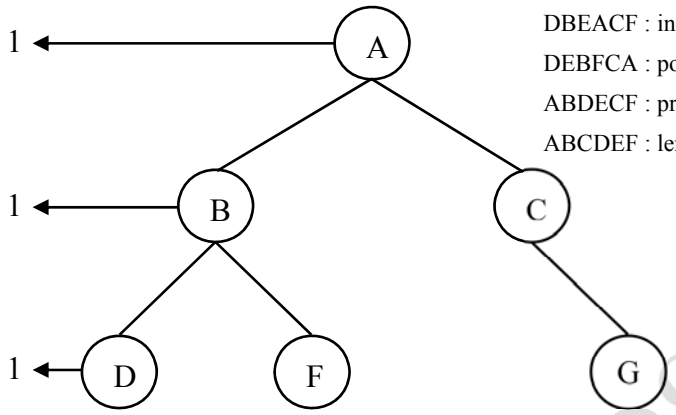
inorder (LDR) – postorder (LRD) – preorder (DLR)

۲. پیمایش سطحی :

این روش گره های درخت را سطوح به سطوح مورد پیمایش قرار می دهد .

توجه :

پیمایش عمقی جهت پیاده سازی به پشته نیاز دارد و در حالی که پیمایش سطحی به صف نیاز دارد .



میان ترتیب (LDR) : inorder DBEACF
پس ترتیب (LDR) : postorder DEBFCA
پیش ترتیب (DLR) : preorder : ABDECF
پیمایش سطحی : lerelorder ABCDEF

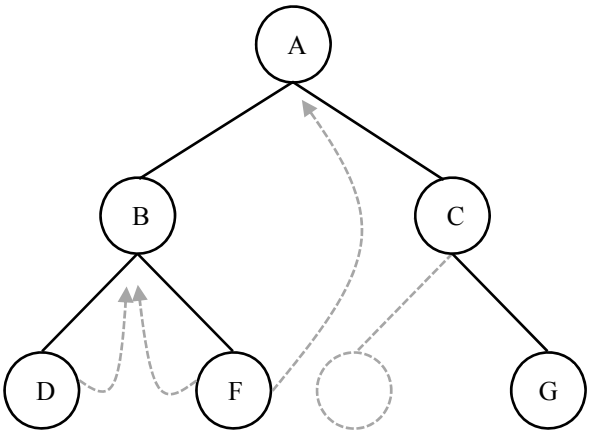
درخت دودوئی نخی[1]

اگر د تعداد گره های درجه صفر و n_1 تعداد گره های درجه یک و n_2 تعدا گره های درجه دو باشد n تعداد کل گره ها است . به ازای هر گره با درجه ۱ ، یک فیلد اتصال و به اعضای هر گره با درجه 0 دو فیلد اتصال در نمایش پیوندی برابر تهی است بنابر این تعداد کل فیلهای اتصال تهی در درخت دودوئی برابر است با $n_1 + 2n_0$. از قضیه یک میدانیم $n_2 = n_0 + 1$ بنابراین :

$$= n_1 + n_0 + n_2 + 1$$
$$= n + 1$$

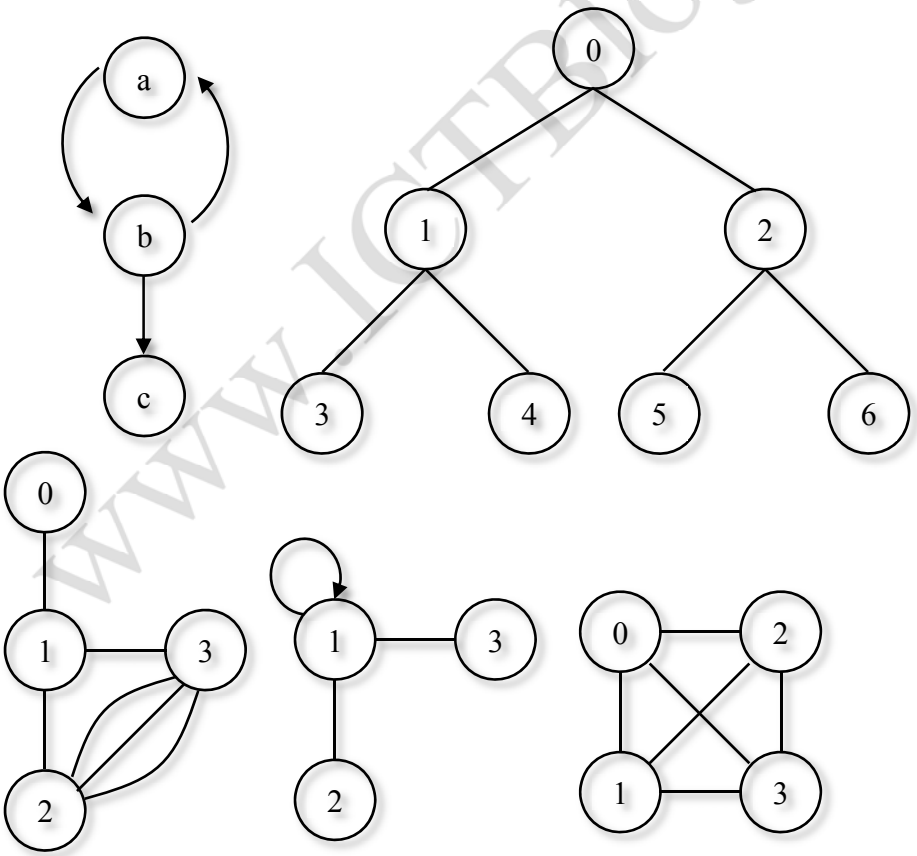
اگر به گونه ای از این فیلد های تهی جهت ایجاد ارتقا با دیگر گره های درخت استفاده می شود ، در این صورت چنین اتصالاتی را نخ و درخت دودوئی حاصل را درخت دودوئی نخی می نامند . یک روش معقول جهت به کارگیری این فیلد ها ایجاد ترتیب منطقی برای گره های درخت بر مبنای یک پیمایش خاص است .

1. Threaded Binary Tree



پیمایش LDR

گراف[1]



1. Graph

گراف $G = V, E$

گراف از دو مجموعه متناهی E , V تشکیل می شود V مجموعه غیر متناهی از راس[1] ها (گره ها) و E مجموعه ای از زوج راس های (U,V) است که یال[2] نامیده میشود . $U,V \in V$

درجه راس

تعداد یال های گذرنده در یک راس از گراف را درجه آن راس می نامند و با $d(V)$ نشان می دهند و همچنین $d(U)$ نشان می دهند .

اگر E تعداد یال های گراف باشد رابطه زیر همواره برقرار است :

تعداد گره ها در درخت $n = B + 1$

$$\frac{1}{2}\sum_{v \in V} d(V) = e$$

دو راس مجاور[3]

دو راس از گراف که توسط یالی به هم متصل شده باشند راس های مجاور نامیده می شوند .

دو یال مجاور :

دو یال واقع بر یک راس را دو یال مجاور نامند .

حلقه :

یالی را که دو سر آن بر یک راس متصل باشد حلقه می نامند .

یال چند گانه :

اگر در گراف بین دو راس چند یال مشابه وجود داشته باشد یال را یال چند گانه و آن گراف را گراف چند گانه می نامند .

---

1. Vertex
2. Edye
3. Multiple Graph

گراف ساده[1]

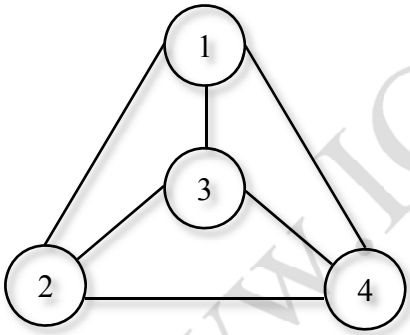
گرافی است که هیچ حلقه ای نداشته باشد و بین دو راس از آن بیش از یک یال وجود نداشته باشد .

گراف کامل[2]

گراف کامل گراف ساده ای است (حلقه ندارد – یال چندگانه ندارد) که بین هر دو راس آن یالی وجود داشته باشد . گراف کامل با N راس را k_n نمایش می دهند . تعداد یال های گراف کامل برابر است با $k_n = \frac{n}{2}(n-1)$

گراف K منتظم

گراف سادهای است که درجه تمام راس های آن برابر K باشد .تعدا یال های گراف K منتظم برابر است :



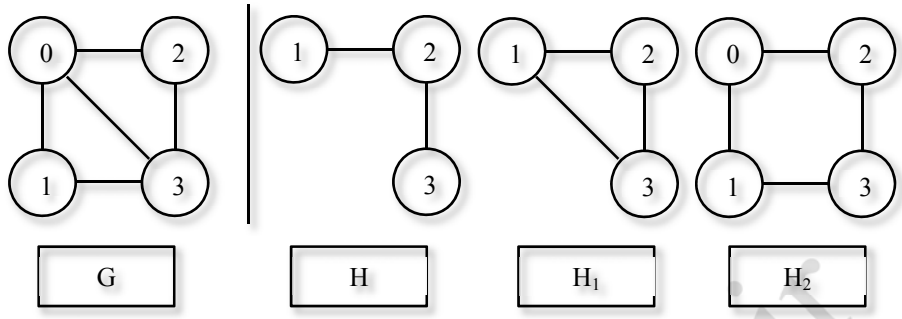
منتظم $k = \frac{nk}{2} \Rightarrow \frac{4 \times 3}{2} = \frac{12}{2} = 6$

گراف H را زیر گراف[3] G گویند هر گاه ، دو شرط زیر برقرار باشد:

1. V (H) ⊆ V(G)
2. E (H) ⊆ E(G)

1. Simple Graph
2. Complete Graph
3. Sub Graph

مثال



$E(G) = \{ (0,1) , (0,2) , (0,3) , (1,2) , (2,3)\}$
$E(H_1) = \{ (0,1) , (1,2)\}$

زیر گراف فراگیر[1]

گراف H زیر گراف فراگیر G گویند هرگاه H زیر گراف بوده و $V(H) = V(G)$ باشد به عبارت دیگر زیر گراف فراگیر زیر گرافی از G است که تمام راس های آن را پوشش می دهد .

مسیر[2]

یک مسیر از راس U به راس V در گراف G دنباله ای است از راس های $u, w_1 ,w_2 ,...,w_k , v$ است به طوری که (u,w_1) , (w_1,w_2) , (w_k,v) یال های متمایز در E(G) بوده و هیچ راسی به جزء احتمالاً u , v تکراری نباشند .

دور یا سیکل[3]

در گراف G مسیری است که در آن رأس های ابتدا و انتها یکی باشند

گراف همبند

گراف بدون جهت G را همبند گویند هرگاه بین دو راس آن مسیری وجود داشته باشد .

1. Spanning Sub Graph
2. Path
3. Cycle

همبند قوی[1]

گراف جهتدار را همبند قوی گویند هرگاه بین هر دو رأس آن مسیر جهت داری وجود داشته باشد .

همبند ضعیف[2]

اگر در گراف جهت دار[3] از جهت یال ها صرف نظر شود گراف غیر جهت داری حاصل می شود که در صورت همبند بودن آن گراف جهت دار اولیه را همبند ضعیف می نامند .

درخت

درخت گراف همبند بدون دور است ، میتوان ثابت کرد که در درخت هر دو رأس با مسیر یکتائی به هم متصل اند .

درخت فراگیر :

اگر G یک گراف با n رأس باشد زیر گراف فراگیری از آن که یک درخت باشد درخت فراگیر گویند .

فرمول کیلی

در حاللت خاصی که G گراف کامل است تعداد درخت های فراگیر آن برابر n^{n-2} است که به فرمول کیلی معروف است .

درخت پوشا[4]

به درخت فراگیر یک گراف اصتلاحاً یک درخت پوشا گویند .

گراف وزن دار[5]

به گرافی که یال های آن دارای وزن باشد گراف وزن دار گفته می شود .

4. Strongly connected
5. Weakly connected
6. Directed
1. Spanning Tree
2. Weighted Graph

کاربردهای گراف :

۱. تحلیل مدار الکتریکی
۲. یافتن کوتاه ترین مسیر
۳. طراحی پروژه ها
۴. نمایش ترکیبات شیمیائی
۵. ژنتیک
۶. زبان شناسی

ADT گراف

شامل عناصر داده ای و عملیات اساسی می باشد .

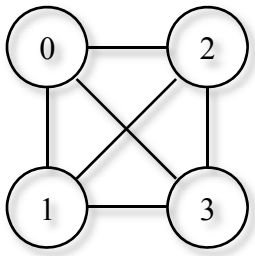
- نوع داده انتزاع گراف
- تصور ذهنی از گراف
- تصور ذهنی از ساختار داده گراف
- گراف دارای ساختار است

عناصر داده ای

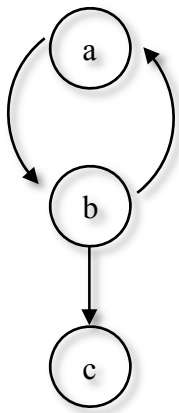
مجموعه ای از رأس ها و یال ها که در آن هر یال توسط دو رأس مشخص می شود .

عملیات اساسی

۱. ایجاد گراف : یک گراف تهی ایجاد می نمائید .
۲. درج رأس : رأس جدیدی به گراف اضافه می کند .
۳. درج یال : یال جدید بین دو رأس موجود در گراف اضافه می کند .
۴. حذف رأس : رأسی از گراف را همراه با تمام یال های مربوط به آن حذف می کند .
۵. حذف یال : یالی را حذف می کند .
۶. برسی تهی بودن : تابعی منطقی که در صورت تهی بودن مجموعه رأس های گراف مقدار True و در غیر این صورت False نتیجه می دهد .
۷. تابع مجاورت : لیستی از رئوس مجاور با رأس مشخصی از گراف نتیجه می دهد .



$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array}\begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$



$$\begin{array}{c} \\ a \\ b \\ c \end{array}\begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \end{array}$$

روش های نمایش گراف :

۱. ماتریس مجاورت
۲. لست مجاورت[1]

تعریف کلاس بر اساس ماترس مجاورت :

```
template <int Maxsize>
class Graph
{
        int n ;
        int A[Maxzise][Maxsize];
};
```

1. Adjacency List

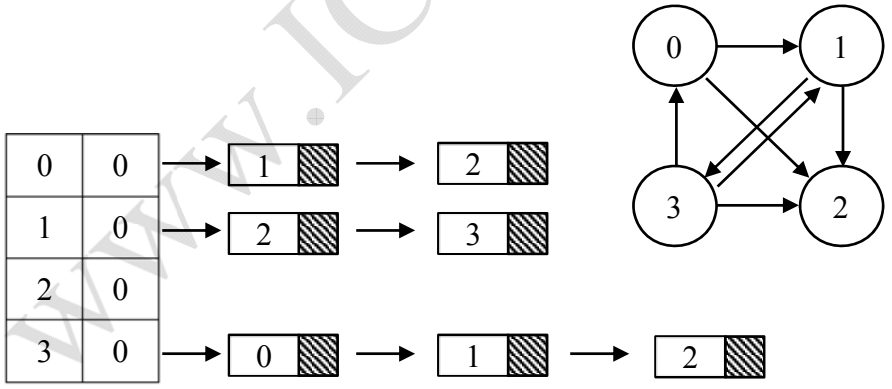
ماترس مجاورت[1]

ساده ترین روش برای نمایش گراف استفاده از ماتریس مجاورت است . اگر گرافی با n رأس باشد . ماتریس مجاورت آن ، آرایه دو بعدی n*m به نام A است . کع در آن اگر یال (i,j) در E(G) موجود باشد آنگاه $A_{ij}=1$ در غیر این صورت $A_{ij}=0$ خواهد بود .

$$a_{ij} \begin{cases} 1 & (i,j) \in E(G) \\ 0 & (i,j) \notin E(G) \end{cases}$$

لیست مجاورت :

در روش نمایش لیست مجاورت هر سطر از ماتریس مجاورت به صورت یک لیست یک طرفه نمایش داده می شود که هر گره از این لیست شامل رو فیلد Data و Next است . با توجه به این که در گرافی با n رأس ماتریس مجاورت دارای n سطر است بنابراین در روش لیست مجاورت n لیست یک طرفه وجود خواهد داشت که لازم است دارای آدرس لبتدائی هر یک از آنها نگهداری شود . برای این منظور می توان آرایه ای از اشاره گرهابه طول n استفاده نمود .



	0	1	2	3
0	0	1	1	0
1	0	0	1	1
2	0	0	0	0
3	1	1	1	0

2. Adjacency Matrix

تعریف کلاس گراف در روش نمایش مجاورت :

```
class Graph
{
        public ;
        Graph ( cons tint vetices = 0 ) : n ( vertices )
        {head = new list <int> * [n] ; };
{
        private :
        list <int> * head ;
        int n ; // number of vertices
};
```

پیمایش گراف

۱. پیمایش عمقی[1] (DFS)
۲. پیمایش سطحی[2] (BFS)

پیمایش عمقی

در روش پیمایش عمقی ابتدا رأس v ملاقات می شود پس از آن رأس های w_1 ملاقات می شود و رأس هاw_2 ... w_k منتظر می مانند تا پیمایش عمقی به صورت بازگشتی و با شروع از رأس w_1 خاتمه یابد . پس از آن پیمایش عمقی به صورت بازگشتی و با شروع از رأس w2 از سر گرفته می شود . این روند به همین ترتیب برای رأس های w3 تا wk ادامه می یابد .

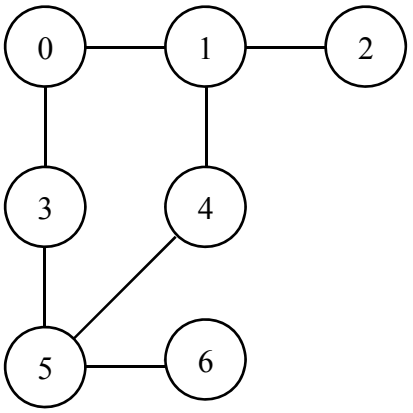
1. Deph First search (DFS)
2. Breadth First Search (BFS)

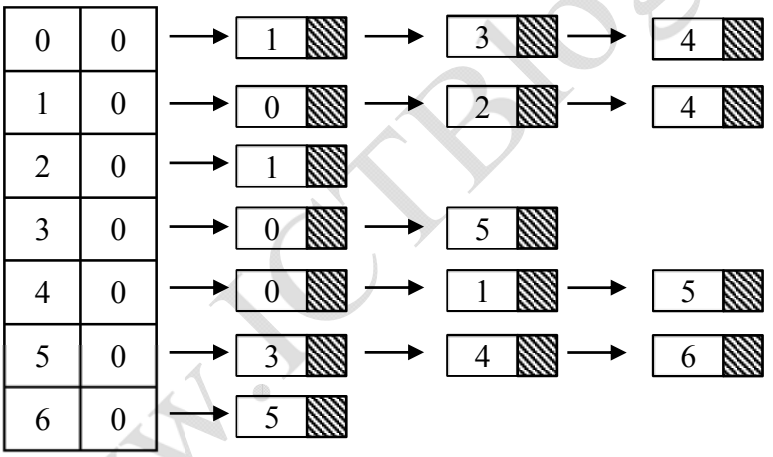
نکته

در روش عمقی در صورتی که گراف G دارای سیکل باشد الگوریتم مجداً به گره هائی که قبلاً ملاقات شده اند برخورد خواهد داشت که این مسأله باعث می شود الگوریتم در حلقه تکرار نا متناهی قرار بگیرد و هیچ گاه خلتمه نیابد . برای حل این مشکل کافی است از یک آرایه منطقی به طول n با نام visited استفاده می کنیم که n تعداد رأس های گراف است و به تعداد عناصر آن مقدار 0 نسبت می دهیم . به این معنا که در ابتدا هیچ کدام از رأس های گراف ملاقات نشده اند .سپس در هنگام پیمایش هر گاه رأسی مانند i ملاقات شود مقدار عنصر visited[i] را برابر 1 قرار می دهیم .

```
void DFS( const int v)
{
        visited [v] = 1 ;
        for ( each vertex w adjacent to v )
        if ( ! visited [w] )
        DFS (w) ;
}
```

مثال





DFS : 0 , 1 , 2 , 4 , 5 , 3 , 6
BFS : 0 , 1 , 3 , 4 , 2 , 5 , 6

مرتب سازی [1]

دسته بندی الگوریتم های مرتب سازی :

۱. از لحاظ مکان قرار گیری عناصر مرتب سازی

الف) مرتب سازی داخلی [2] (داخل حافظه Ram)

ب) مرتب سازی خارجی [3] (بر روی Hard disk)

1. Sort
2. Internal Sorting
3. External Sorting

۲. از لحاظ ایده اصلی در روش مرتب سازی :

الف) مرتب سازی تعویضی : (مثل مرتب سازی حبابی)

در این روش ایده اصلی در مرتب سازی تعویضی تعویض احتمالی عناصر پس از مقایسه کلید آنها است .

ب) مرتب سازی انتخابی :

در این گونه روش ها در هر گذر از الگوریتم عنصری از لیست انتخاب می شود و جای مناسب خود قرار می گیرید.

ج) **مرتب سازی درجی**[1]

در این روش از درج عناصر در موقعیت مناسب از لیست در مرتب سازی استفاده می شود .

۳. از لحاظ رتبه زمانی

الف) مرتب سازی $O(n^2)$ مثل مرتب سازی حبابی (الگوریتم دارای دو حلقه تو در تو)

ب) مرتب سازی از مرتبه ($O(n \log n)$ این روش بر اساس ایده تقسیم غلبه می کند در نتیجه در اغلب موارد دارای کارائی مناسبی است .

۴. از لحاظ مصرف حافظه اضافی جخت مرتب سازی

الف) مرتب سازی در جا[2]

ب) مرتب سازی برون از جا[3]

معیار های مهم در مرتب سازی

۱. سرعت الگوریتم : بحث مربوط به کارائی الگوریتم ، پیچیدگی ، تابع زمانی و ...)

۲. رفتار طبیعی الگوریتم :

به عنوان مثال باید انجام الگوریتم بر روی یک لیست مرتب بسیار سریع تر از انجام آن در یک لیست نا مرتب باشد .

۳. حفظ ترتیب عناصر با کلید مساوی

الگوریتم مرتب سازی خوب الگوریتمی است که از تعویض های غیر ضروری پرهیز نماید

الگوریتمی که دارای چنین ویژگی باشد متعادل یا پایدار

۴. حافظه مصرفی در مرتب سازی

---

1. Insertion Sort
2. In-Place
3. Out-place

انواع الگوریتم های مرتب سازی

۱. مرتب سازی حبابی[1]

(تعویضی ، متعادل ، In-Place)

این روش از مرتب سازی های تعویضی است و شامل چندین گذر[2] و در هر گذر لیست از ابتدا به سمت انتها پیمایش می شود و هر عنصر با عنصر بعدی مقایسه می شود ، در صورتی که عنصر جاری از عنصر بعدی بزرگتر باشد جای دو عنصر در لیست عوض می شود . واضح است که در پایان اولین گذر بزرگترین عنصر در انتهای لیست قرار خواهد گرفت به طور کلی در گذر i ام تمام عناصر موجود از مکان n-i به بعد در موقعیت مناسب خود قرار می گیرند .

مثال

```
void bubble sort ( int a[] , cons tint n )
{
        int i , j temp ;
        for ( i=1 ; i <= n-1 ; i ++ )
        for ( j=1 ; i < n-1-1 ; ; j ++)
        if ( a[j] > a[j+1] )
        {
        temp = a[j];
        a[j] = a[j+1];
        a[j+1] = temp ;
        }
}
```


1. Bubble Sort
2. Pass

مثال

گذر	a[0]	a[1]	a[2]	a[3]	توضیح
اول (i = 1)	8	5	3	4	جابجائی انجام می شود
	5	8	3	4	جابجائی انجام می شود
	5	3	8	4	جابجائی انجام می شود
	5	3	4	8	پایان گذر اول
دوم (i = 2)	5	3	4	8	جابجائی انجام می شود
	3	5	4	8	جابجائی انجام می شود
	3	4	5	8	پایان گذر دوم
سوم (i = 3)	3	4	5	8	جابجائی انجام نمی شود
	3	4	5	8	پایان گذر سوم

توجه

در روش حبابی در گذر اول n-1 مقایسه و در گذر دوم n-2 مقایسه و در نهایت در گذر n-1 یک مقایسه انجام می شود این موضوع نشان می دهد در گذر های بعدی به تدریج سرعت الگوریتم بیشتر می شود .

تحلیل الگوریتم Bubble Sort

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

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

$$\frac{1}{2} n(n-1) = O(n^2)$$

مرتب سازی انتخابی[1]

در مرتب سازی انتخابی ایده اصلی انتخاب یک عنصر از لیست و قرار دادن آن در مکان صحیح در لیست مرتب نهائی است . برای این کار از عمل جابجائی استفاده می شود . منظور از مکان صحیح موقعیت نهائی این عنصر در لیست مرتب شده است ساده ترین روش مرتب سازی انتخابی یافتن بزرگترین یا کوچکترین عنصر از لیست و قرار دادن ان در انتها یا ابتدای لیست نا مرتب باقی مانده است . ئاضح است که چنین روشی به n-1گذر جهت مرتب سازی کامل لیست نیاز دارد در صورتی که در هر گذر انتخاب بزرگترین عنصر از لیست نا مرتب باقی مانده مدنظر باشد می توان الگوریتم زیر را برای این منظور به کار برد .

```
void Selechtion Sort ( int a[] , cons tint n )
{
        int i , j , max , max index ;     محل آخرین عنصر در لیست ، تعداد  n-1 گذر لازم است
        for ( i = n-1 ; i >= 1 ; i -- )
        {
        max = a[0] ; maxindex=0;
        for ( j=1;j<=i;j++) // findundex
        if(a[j]>max)
        {
         max=a[j]; maxindex = j ;  اگر شرط برقرار باشد
        }
        a[maxindex]=a[i];  //a[i],maxتعویض
        a[i] = max;
        }
}
```

1. Selection

مثال

مرتب سازی آرایه a={8,1,6,5,3} به روش انتخابی :

توضیح	a[4]	a[3]	a[2]	a[1]	a[0]	گذر
تعویض 8 با 3	3	5	6	1	8	اول (i=1)
تعویض 6 با 5	8	5	6	1	3	دوم (i=2)
تعویض 5 با خودش	8	6	5	1	3	سوم (i=3)
تعویض 3 با 1	8	6	5	1	3	چهارم (i=4)
پایان مرتب سازی	8	6	5	3	1	

تابع زمانی

$$T(n) = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = O\,(n^2)$$

مرتب سازی درجی[1] : (درجی ساده)

این روش از این ایده برای مرتب سازی لیست n عنصری $a = \{ a_0,a_1,\ldots,a_{n-1}\}$ استفاده می کند . ایده از این قرار است که به این صورت که ابتدا مجموعه یک عنصری $\{a_0\}$ را در نظر گرفته و عنصر $\{a_1\}$ به گونه ای در آن درج می کند که یک لیست مرتب دو عنصری ایجاد شود سپس عنصر $\{a_2\}$ را در لیست دو عنصری ایجاد شده طوری ژرج می کند که یک لیست مرتب سه عنصری به دست آید . این روند ادامه می یابد تا لیست مرتبی شامل n عنصر ایجاد شود بنابراین مرتب سازی درجی در n-1 مرحله انجام می شود ، با توجه به این که در هر گذر طول لیست مرتب یک واحد افزایش می یابد الگوریتم در گذر های بعدی به تدریج کند تر می شود .

1. Inservtion Sort

```
void Insertion Sort ( int a[] , cons int n )
{
        int i,j,x ;
        for(i=1;i,=n-1;i++)
        {
        x=a[i];
        for(j=i-1; (j>=0)&&(x<a[j]);j--)
        a[j+1]=a[j];
        a[j+1] = x ;
        }
}
```

مرتبه زمانی الگوریتم درجی ساده :

$$T(n) = \frac{n(n-1)}{2} = O(n^2)$$

نکته :

با توجه به همه معیار های ارزیابی همه الگوریتم ها (رفتار طبیعی ، سرعت الگوریتم ،حافظه مصرفی متعادل یا پایدار بودن) می توان گفت مرتب سازی درجی ساده از هر دو روش مرتب سازی حبابی و انتخابی بهتر است .

مرتب سازی سریع[1]

یکی ار معروف ترین و بهترین روش های مرتب سازی است که توسط شخصی بنام هوار (C.A.R.Hoar) ایده اصلی در این روش شکستن لیست اولیه به لیست کوچکتر جهت تسریع عمل مرتب سازی است و از این لحاظ از نوع روش های مبتنی بر تقسیم بر غلبه است به منظور تقسیم لیست اولیه به دو قسمت کوچکتر ، ابتدا یکی از عناصر لیست به عنوان عنصر محور انتخاب می شود سپس عناصر لیست اولیه به گونه ای جابجا می شود که تمام عناصر بزرگتر از عنصر محور در سمت راست و تمام عناصر کوچکتر از عنصر محور در سمت چپ عنصر محور قرار گیرند در این حالت عنصر محور در مکان نهائی خود قرار می گیرد سپس هر دو قسمت از لیست به همین روش مرتب می شود . بنابراین طراحی این الگوریتم به روش بازگشتی مناسب تر و قابل فهم تر است . معمولاً عنصر ابتدائی یا عنصر وسط لیست برای این منظور انتخاب می شود . می توان یکی از عناصر لیست را به طور تصادفی نیز انتخاب کرد . در الگوریتم زیر فرض بر این است که هر بار عنصر ابتدائی لیست به عنوان محور انتخاب می شود . همچنین برای سادگی بیان از تابعی به عنوان Swap استفاده شده است که وظیفه آن تعویض دو عنصر از لیست است ، در این الگوریتم برای

1. Quick Sort

مرتب سازی لیستی به طول n اولین فراخوانی بـه صـورت Quick Sort (a , 0 , n-1) انجـام مـی شـود .

```
void Quick Sort( int a[] , int left , int right )
{
        int i , j , Pivot ;
        if ( left < right )
                                        i=left ; j=right ;
        pivot = a[ left ] ;
        do {
                                                do i++ ;
                                        while (a[i]<pivote);
                                                do i -- ;
                                        while ( a[j]<pivote);
                                        if ( i<j) swap ( a,i,j)
        }
        swap ( a,left,j );
        Quick Sort ( a,left,j-1);
        Quick Sort  ( a , j+1 , rught);
}
}
```


web : www.ictblog.ir
E-mail : iceman200063x@yahoo.com