ساختمان داده ها

۱ تحلیل الگوریتم و ساختمان داده های پایه

۱.۱ الگوریتم و تحلیل برنامه ها

تعریف الگوریتم:

مجموعه محدودی از دستورالعملها که با دنبال کردن آنها هدف خاصی دنبال می‌شود و دارای خصوصیات زیر است:

۱) ورودی : هیچ یا چندین کمیت ورودی از محیط خارجی
۲) خروجی : وجود حداقل یک کمیت به عنوان خروجی
۳) قطعیت : خالی بودن از هر گونه ابهام در مورد هر دستورالعمل
۴) محدودیت : خاتمه یافتن پس از طی مراحل محدود
۵) کارآیی : انجام پذیر بودن هر دستورالعمل (قابلیت اجرا به صورت دستی با قلم و کاغذ)

نکته:

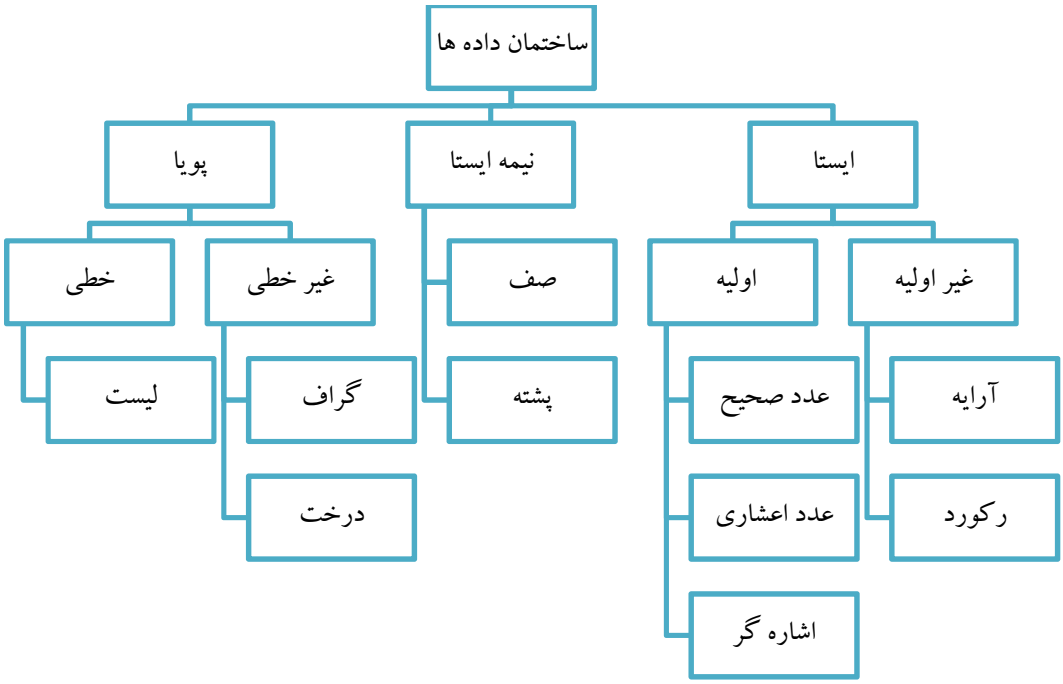
تفاوت برنامه و برنامه در این است که الگوریتم باید پایان پذیر باشد (خصوصیت ۴) حال آنکه برنامه لزوما پایان پذیر نمی‌باشد.

مثال :

سیسستم عامل برنامه‌ای است که هیچگاه پایان نمی‌پذیرد و همواره در یک سیکل انتظار قرار دارد تا برنامه بعدی وارد شود.

ساختمان داده:

ساختارهایی که جهت دریافت داده های خام به شکل مناسب توسط کامپیوتر و پیاده سازی و اجرای الگوریتم های مختلف روی آنها مورد استفاده قرار می‌گیرند، ساختمان داده نامیده می‌شوند. شکل ۱ تقسیم بندی ساختمان داده های مختلف را نشان می‌دهد.



شکل ۱

نکته: ساختار ساختمان داده های ایستا در طول حیاتشان تغییر نمی‌کند ولی در نوع پویا تغییرات مجاز و نامحدود است.

تحلیل برنامه ها:

عوامل زیادی در ارزیابی که برنامه موثرند. در تحلیل الگوریتم ها، با در نظرگرفتن مقدار حافظه مصرفی و زمان محاسباتی (زمان اجرا) الگوریتم های مختلف با یکدیگر مقایسه می‌شوند. اگر فرض شوند همه الگوریتم ها بر روی یک ماشین اجرا شوند و سرعت اجرای کلیه دستورالعملها برابر واحد (یک) باشد، زمان اجرای الگوریتم ها به عنوان تابعی از تعداد ورودی های مسئله محاسبه شده و ملاک مقایسه چند الگوریتم ارایه شده قرار می‌گیرد. سرعت اجرا مرتبه اجرا نیز نامیده می‌شود.

مثال ۱ : در کدهای زیر

```
x=x+1;
```

کد ۱

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

کد۲

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

کد۳

در کد ۱ دستور x=x+1 یکبار،در کد ۲، دستور مربوطه n بار و در کد ۳ دستور مربوطه n^2 بار اجرا می‌شوند که سرعت اجرا مرتبه جرا نیز نامیده می‌شود.

۱.۲ زیر برنامه های بازگشتی

بعضی از مسائل طبیعت بازگشتی دارند. به عنوان نمونه اگر بخواهیم پاسخ !5 را محاسبه کنیم با توجه به فرمول n!=n x (n-1)! می‌توانیم بگوییم اگر !4 را بدانیم کافی است آنرا در ۵ ضرب کنیم پس مسئله !5 تبدیل می‌شود به !4 و الی آخر.

کد برنامه شماره ۱، نحوه محاسبه فاکتوریل n را به روش بازگشتی نشان می‌دهد.(برنامه را در کارگاه تایپ و اجرا کنید)

```
#include "stdafx.h"
#include <iostream>
using namespace std;
int fact(int);
int _tmain(int argc, _TCHAR* argv[])
{
	int a,b;
	cout<<"\n\tEnter n:";
	cin>>a;
	b=fact(a);
	cout<<"\n\t "<<a<<"! ="<<b;
	cin.ignore();
	cin.get();
	return 0;
}
//////////////////////////////////////////////////
int fact(int n){
int a;
	if (n<=1){
		a=1;
	return(a);
	}//end of the if (n<=1)
	if(n>1){
	a=n*fact(n-1);
	}//end of the if (n>1)

}//end of the fact
```

کد برنامه شماره1 :

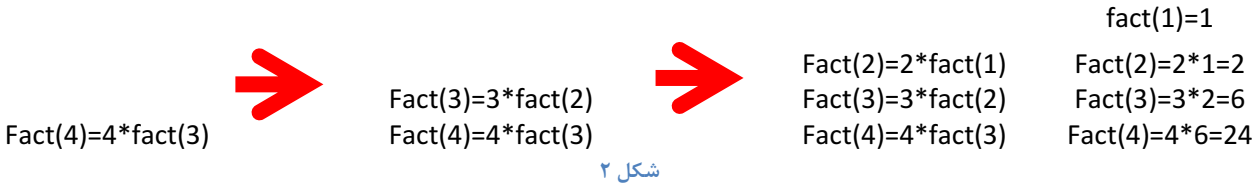
ویژگی های اصلی برنامه های بازگشتی :

۱) تابع خودش را صدا می‌کند البته اغلب با آرگومان کوچکتر.

۲) یک شرط جهت اتمام فراخوانی ها وجود دارد.

نحوه اجرای برنامه های بازگشتی

کامپیوتر دارای حافظه‌ای به نام پشه یا stack است، عملیاتی که بعدا باید توسط کامپیوتر انجام گیرد در این حافظه ذخیره می‌شود. این حافظه به صورت LIFO[1] می‌باشدیعنی آخرین اطلاعات وارد شده ابتدا خارج می‌شوند. مانند برداشتن بشقابهایی که روی هم چیده شده‌اند. پشته در هنگام اجرای برنامه ها اطلاعات آنها را نگهداری می‌کند و در هنگام اجرای برنامه های بازگشتی کاربرد ویژه‌ای دارد. در شکل ۲ مراحل فراخوانی fact(4) نمایش داده می‌شود.

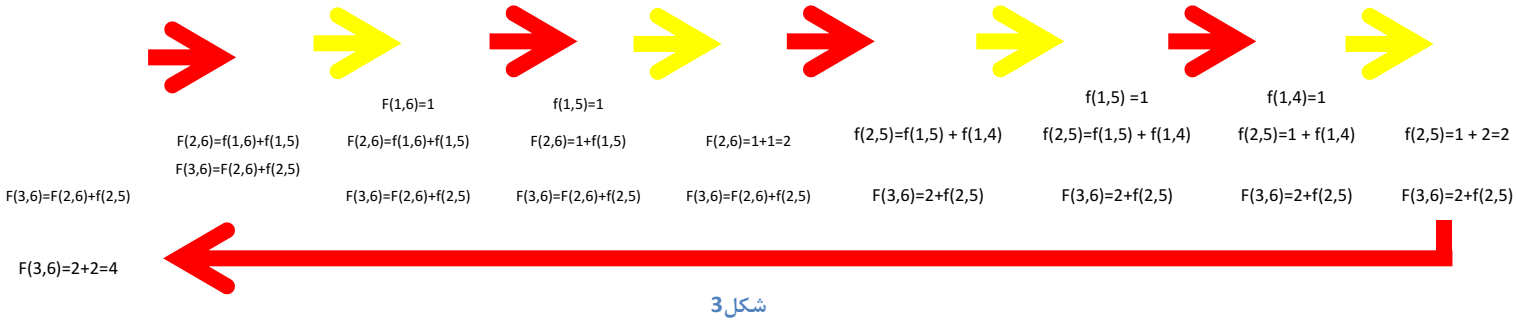


شکل ۲

تمرین ۱

برنامه زیر را در نظر بگیرید و خروجی آنرا برای F(3,6) با رسم شکل پشته مشخص نمایید(برنامه را در کارگاه تایپ و اجرا کنید).

```
int  f(int m, int n){
if (m==1 !! n==0 !!m==n)
return (1);
else
return ( f(m-1,n)+f(m-1,n-1));
}
```



شکل3

[1] LIFO: Last In First Out

همانطور که در شکل۳ مشاهده می‌شود با طولانی شدن برنامه شکل پشته بسیار بزرگ می‌شود و ممکن است امکان ترسیم آن وجود نداشته باشد، در چنین مواقعی بهتر است از درخت برای نمایش پشته و فراخوانی های مربوطه استفاده نمود شکل ۴.

شکل4

نکته :

توابع می‌توانند به طور مستقیم یا غیر مستقیم خودشان را فراخوانی کنند (شکل ۵ الف). در روش مستقیم یکی از دستورات تابع خودش را فراخوانی می‌کند. در روش غیر مستقیم، تابعی مثل f1() تابع f2() را فراخوانی می‌کند و f2() تابع f1() را فراخوانی می‌کند(شکل ۵ ب).

int g(int a){
 	
……
if (a!=1)
return (g*(g/2))
..
….
…
 	
}
 	
الف
	int f1(int a){
	…
	….
	b=f2(c);
	…
	}
	
	int f2(int b){
	…
	…
	d=f1(f);
	….
	
	}
	
	ب

شکل 5

۱.۳ تحلیل پیچیدگی زمانی

در ارزیابی یک الگوریتم دو فاکتور همی که باید مورد توجه قرار گیرد یکی حافظه مصرفی و دیگری زمان مصرفی الگوریتم است. یعنی الگوریتمی بهتر است که فضا و زمان کمتری بخواهد.

نکته: در این درس مسئله زمان مهمتر می‌باشد.

بازدهی الگوریتم معمولا بر اساس زمان تحلیل می‌شود. در این تحلیل ها تک تک دستورات شمرده نمی‌شوند زیرا تعداد دستورات به نوع زبان برنامه نویسی و نحوه نوشتن برنامه ها بستگی دارد در عوض به میزانی که مستقل از سخت افزار کامپیوتر و زبان برنامه سازی باشد نیاز است. به طور کلی زمان اجرای یک الگوریتم با افزایش اندازه ورودی (n) زیاد می‌شود و زمان اجرا با تعداد دفعاتی که عملیات اصلی انجام می‌شود تناسب دارد. بنابراین بازدهی الگوریتم را با تعیین دفعاتی که یک عمل اصلی انجام می‌شود، به عنوان تابعی از ورودی تحلیل می‌شود.

مثال ۲ : تابع زیر جمع عناصر یک آرایه را در زبان c را محاسبه می‌کند(این برنامه را در کارگاه اجرا کنید).

```
double sum(double list[],int n){
int i;
double s=0;
for(i=0;i<n;i++){
s=s+list[i];
}//end of the for
return(s);
}//end of the sum()
```

در این برنامه اندازه ورودی n یا تعداد عناصر آرایه می‌باشد و عمل اصلی s=s+list[i] می‌باشد که n بار انجام می‌شود.

نکته:

بعد از تعیین اندازه ورودی یک دستورو یا گروهی از دستورات باید انتخاب شوند که کل کار انجام شده توسط الگوریتم تقریبا متناسب با تعداد دفعاتی باشد که این دستور یا این گروه از دستورات اجرا می‌شوند. این دستور یا گروه دستورات را عمل اصلی می‌نامند

مثال ۳ تعداد کل مراحل برنامه مثال را حساب کنید.

```
double sum(double list[],int n){   0
int i;                              0
double s=0;                         1
for(i=0;i<n;i++){                   n+1
s=s+list[i];                        n
```

```
}//end of the for
return(s);                    1
}//end of the sum()          ________
                              2n+3
```

تعداد کل دستورات اجرا شده 2n+3 می‌باشد. چون عمل اصلی n بار اجرا می‌شود زمان اجرا را T(n)=n بیان می-کنیم.

نکته: هنگام محاسبه تعداد دفعاتی که یک دستور درون حلقه ها اجرا می‌شود از چهار فرمول زیر استفاده می-شود:

ضریب k عدد ثابتی است $\sum_{i=1}^{n} kf(i) = k\sum_{i=1}^{n} f(i)$ $\sum_{i=1}^{n} 1 = n$

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2} \qquad \sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

مثال ۴ در برنامه زیر x=x+1 چند بار اجرا می‌شود؟

```
for j=:1 to n do
        for i:=1 to j do
                x:=x+1
```

جواب :

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

۱.۴ مرتبه اجرایی الگوریتم

تعریف :

f(n)=O(g(n)) می‌باشد اگر و فقط ارزیابی به ازای مقادیر ثابتی از C و n_0،f(n) برای مقادیر بزرگتر یا مساوی n_0 کمتر یا مساوی Cg(n) باشد یعنی:

$$f(n)=O(g(n)) \Leftrightarrow \exists\, C, n_0 : \forall\, n \geq n_0\ f(n) \leq Cg(n)$$

در این فرمول f و g توابع غیر منفی می‌باشند. در این حال می‌گوییم مرتبه اجرایی تابع f(n)، تابع g(n) می‌باشد.

مثال ۵

تابع f(n)=3n+3=O(n) می‌باشد چرا که اگر C=4 و n_0=2 فرض کنیم داریم :

برای n>=2 3n+3<=4n

توجه:

رابطه فوق برای n=1 صادق نیست ولی طبق تعریف کافی است n_0 ای وجود داشته باشد (حداقل یک n_0) که از آن به بعد f(n)<=Cg(n) باشد. در این مثال می‌توانم n_0 را برابر ۳ یا ۴ یا بیشتر نیز در نظر بگیریم. همچنین C را می‌توان ۵ یا ۶ درنظر گرفت.

مثال ۶

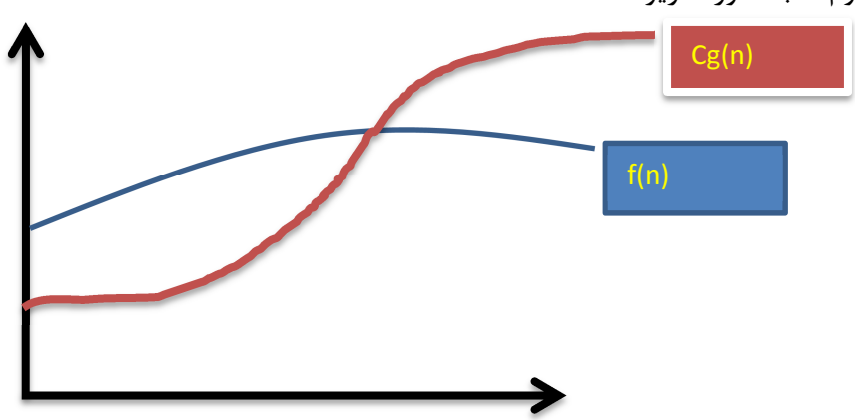
100n+6=O(n) می‌باشد چرا که اگر C=101 و n_0=10 فرض کنیم، داریم:

برای n>=10 : 100n+6<=101n

تذکر ۱:

گاهی اوغات عبارت f(n)=O(g(n)) را به صورت $f(n) \in O(g(n))$ نیز نمایش می‌دهند.

نمایش نموداری مفهوم O به صورت زیر است.



یعنی از n_0 به بعد عبارت Cg(n) همواره بزرگتر از f(n) می‌باشد.

مثال ۷

تابع $n^2+10n=O(n^2)$ می‌باشد چرا که برای $n_0=10$ و $C=2$ داریم :

$n^2+10n <=2n^2$

یعنی شکل نمودار n^2+10n سرانجام در زیر تابع $2n^2$ قرار می‌گیرد.

قضیه :

اگر $f(n)=a_m n^m+...+a_1 n+a_0$ باشد آنگاه $f(n)=O(n^m)$ خواهد بود.

مثال ۸

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

مثال ۹

مرتبه اجرایی برنامه زیر چیست؟

x:=0;	i	x
i:=n;	32	1
while (i>1) do begin	16	2
x:=x+1;	4	3
i:= i div 2;	2	4
end;	1	5

برای n=32 عمل اصلی x:=x+1 ۵ بار انجام می‌شود. $5=\log_2 32$

پس در حالت کلی مرتبه اجرایی الگوریتم برابر O(log2) می‌باشد.

نکته: در درس ساختمان داده های منظور از logn همان $\log_2 n$ می‌باشد.

نکته: در جدول زیر مرتبه اجرایی چند تابع به ترتیب صعودی از چپ به راست نوشته شده است:

نام تابع	ثابت	لگاریتمی	خطی	------	مرتبه ۲	توانی	فاکتوریل
مرتبه اجرا	O(1)	O(logn)	O(n)	O(nlogn)	$O(n^2)$	$O(2^n)$	O(n!)