فصل دوم

روش‌های تحلیل الگوریتمهای بازگشتی

مقدمه

در فصل قبل الگوریتمهای ترتیبی را مورد تحلیل و بررسی قرار دادیم و زمان اجرای این الگوریتمها را محاسبه کردیم. در این فصل قصد داریم الگوریتمهای بازگشتی را تحلیل کنیم. همانطور که قبلاً اشاره کردیم الگوریتمهای بازگشتی، الگوریتمهایی هستند که داخل خود چند بار فراخوانی می‌شوند و بعد از تعدادی فراخوانی (با توجه به اندازه ورودی) به مقدار ثابت می‌رسند، سپس شروع به محاسبه مقدار تابع می‌نمایند. بدست آوردن پیچیدگی زمانی، چنین الگوریتمهایی، نیاز به تکنیک‌های خاصی دارند که در این فصل مورد بررسی قرار می‌دهیم.

۱-۲ الگوریتمهای بازگشتی (recursive algorithm)

الگوریتمی را بازگشتی می‌نامند که برای محاسبه مقدار تابع نیاز به فراخوانی خود به تعداد لازم باشد. از خصوصیات الگوریتمهای بازگشتی می‌توان به سادگی پیاده سازی و سادگی درک الگوریتم و غیره اشاره کرد.

در بسیاری از موارد با توجه به خصوصیات الگوریتمهای بازگشتی ممکن است برای بکارگیری در مسائل نسبت به الگوریتمهای ترتیبی ترجیح داده شوند ولی همیشه استفاده از آنها مفید نیست. در بعضی از مواقع ممکن است حافظه یا زمان اجرای زیادی

Scanned with CamScanner

را در مرحله اجرا هدر دهند. لذا غالباً بعد از تحلیل الگوریتمهای بازگشتی در مورد بهتر بودن آنها در مرحله اجرا تصمیم می‌گیرند.

بازگشت‌پذیری برای اولین بار در زبان برنامه‌سازی لیسپ پیاده‌سازی و امکان تعریف توابع بازگشتی برای برنامه‌سازان فراهم گردید. امروزه اکثر زبانهای برنامه‌سازی همه منظوره دارای امکان تعریف و فراخوانی توابع بازگشتی هستند.

الگوریتمهای بازگشتی شامل دو مرحله مهم هستند:

- عمل فراخوانی
- بازگشت از یک فراخوانی

با بکارگیری توابع بازگشتی دو مرحله بالا بترتیب انجام می‌گیرد. در مرحله فراخوانی اعمال زیر انجام می‌شود:

۱) کلیه متغیرهای محلی (Local Variable) و مقادیر آنها در پشته (Stack) سیستم قرار می‌گیرند.

۲) آدرس بازگشت به پشته منتقل می‌شود.

۳) عمل انتقال پارامترها (parameter passing) صورت می‌گیرد.

٤) کنترل برنامه (program counter) بعد از انجام مراحل بالا به ابتدای پردازه جدید اشاره می‌کند.

و در مراحل بازگشت عکس عملیات فوق، به‌صورت زیر انجام می‌شود:

۱) متغیرهای محلی از سرپشته حذف و در خود متغیرها قرار می‌گیرند.

۲) آدرس بازگشت از بالای پشته بدست می‌آید.

۳) آخرین اطلاعات از پشته حذف (pop) می‌شود.

٤) کنترل برنامه از آدرس بازگشت بند۲ ادامه می‌یابد.

همانطور که اشاره کردیم با بکارگیری الگوریتمهای بازگشتی اعمال فوق بترتیب انجام می‌شود. و همانطور که ملاحظه می‌کنید در بعضی از مواقع امکان استفاده از الگوریتمهای بازگشتی بدلیل اینکه حافظه زیادی را هدر می‌دهند، وجود ندارد. (در بعضی از مواقع نیز زمان زیادی را برای اجرا نیاز دارند).

Scanned with CamScanner

بنابراین در مسائلی که از الگوریتمهای بازگشتی استفاده می‌کنیم. تحلیل و بررسی دقیقی از میزان حافظه مصرفی و زمان اجرا نیازمندیم.

۲-۲ محاسبه مقادیر الگوریتم بازگشتی

همانطور که در بالا اشاره کریم برای محاسبه مقادیر الگوریتمهای بازگشتی دو عمل فراخوانی و بازگشت از فراخوانی را نیاز داریم. که در بعضی مواقع ممکن است محاسبه مقدار الگوریتم بازگشتی مشکل به نظر برسد. بنابراین ترجیح دادیم که در این بخش مثال‌هایی را برای روشن شدن مطلب ارائه دهیم.

۲-۲-۱ روش بازگشتی محاسبه فاکتوریل

مساله محاسبه فاکتوریل یک عدد صحیح ساده‌ترین مثال، برای بیان الگوریتمهای بازگشتی می‌باشد. همان‌طور که می‌دانیم فاکتوریل یک عدد صحیح n، به‌صورت بازگشتی زیر قابل تعریف است:

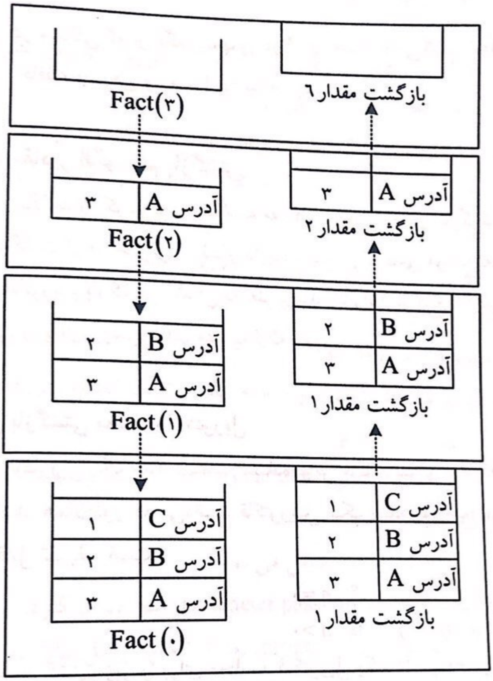
$$n! = \begin{cases} ۱ & \text{if} \quad n = ۰ \\ n \times (n-۱)! & \text{if} \quad n > ۰ \end{cases}$$

حال تابع بازگشتی زیر را برای محاسبه فاکتوریل یک عدد به‌صورت بازگشتی ارائه می‌دهیم:

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

حال دو مرحله اصلی در محاسبه الگوریتمهای بازگشتی را در مثال بالا بررسی می‌کنیم.

همانطور که قبلاً اشاره کریم در مرحله فراخوانی، مقادیر متغیرها در پشته قرار می‌گیرند یا اصطلاحاً در پشته push می‌شوند. بنابراین برای n=3 شکل زیر را خواهیم داشت:



شکل ۱-۲: مراحل محاسبه الگوریتمهای بازگشتی برای فاکتوریل

در الگوریتم بالا نخست fact(۳) فراخوانی می‌شود. بازای n=3 تابع دوباره فراخوانی می‌شود بنابراین مقادیر فراخوانی اول در پشته سیستم ذخیره می‌شود و عمل فراخوانی دوباره ادامه می‌یابد تا اینکه n=0 شود. در اینصورت برای محاسبه عملیات لازم در توابع فراخوانی شده، مقدار یک بازگشت داده می‌شود. بازای هر مرحله بازگشت یک عمل حذف از بالای پشته انجام می‌گیرد و در عین حال عملیات لازم برای بازگشت بعدی انجام می‌گیرد. تا زمانیکه پشته خالی نشده باشد عمل بازگشت ادامه می‌یابد.

Scanned with CamScanner

٢-٢-٢ روش بازگشتی محاسبه سری فیبوناچی

سری فیبوناچی یکی از مسائلی است که ذاتاً به‌صورت غیربازگشتی نیز ارائـه مـی‌شـود. ولی بیان آن به‌صورت بازگشتی به نظر ساده می‌رسد.

به‌صورت زیر می‌توان رابطه بازگشتی سری را نمایش داد:

$$\mathrm{Fib}(n)=\begin{cases} ٠ & \text{if} \quad n=١ \\ ١ & \text{if} \quad n=١ \\ \mathrm{fib}(n-١)+\mathrm{fib}(n-٢) & \text{if} \quad n>٢ \end{cases}$$

در حالت کلی جملات سری عبارتند از:

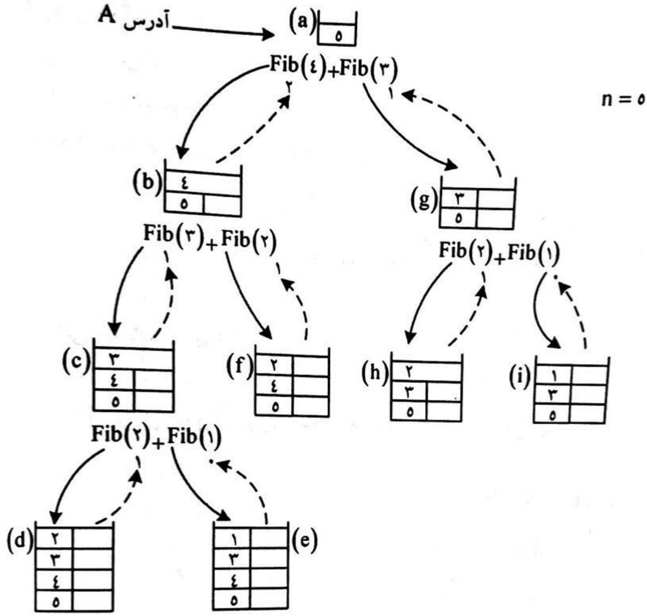
٠ ١ ١ ٢ ٣ ٥ ٨ ١٣ ...

تابع بازگشتی زیر را برای تولید جملات سری فیبوناچی بکار می‌بریم:

```
int    fib ( int    n )
{
      if (  n = = 1 )
           return ( 0 ) ;
      else if ( n = = 2 )
           return (1) ;
      else
           return ( fib ( n – 1 )  +  fib ( n – 2 ) ) ;
}
```

مراحل الگوریتمهای بازگشتی، برای الگوریتم بازگشتی بالا به ازای n=٥ در شکل (٢-٢) نمایش داده شده است.

Scanned with CamScanner



شکل ۲-۲: مراحل محاسبه الگوریتمهای بازگشتی برای سری فیبوناچی

در مرحله اول اجراء، فراخوانی تابع شروع می‌شود که با فلش توپردر شکل نشان داده است، بعد از انجام هر مرحله کامل از فراخوانی مرحله بازگشت شروع می‌شود که با فلشهای منقطع در شکل مشخص شده است. ترتیب فراخوانیها با حروف A تا I در شکل مشخص می باشد.

در نهایت تابع مقدار ۳ را به‌عنوان خروجی برمی‌گرداند.

۲-۲-۳ روش بازگشتی محاسبه برج هانوی

یکی دیگر از مسائل کلاسیک که حل آن به روش بازگشتی قدرت این روش را نشان می‌دهد. مسأله‌ای بنام برج هانوی است. در این مسأله سه محور ثابت (میله) به نامهای

Scanned with CamScanner

A، B و C داریم که در ابتدای کار هشت دیسک (Disk) با اندازه‌های متفاوت و از بزرگ به کوچک حول محور A رویهم انباشته شده‌اند (به شکل توجه کنید).

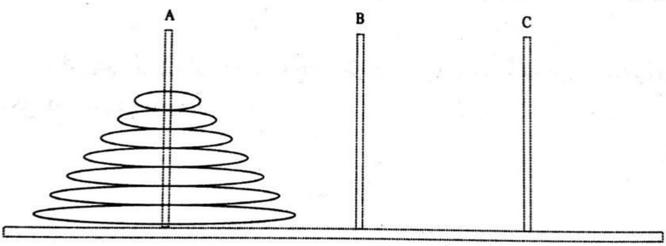
در این مسأله هدف انتقال تمام دیسک‌های روی میله A به میله دیگر مثلاً C می‌باشد، به‌طوریکه قواعد زیر رعایت شود:

۱) هر بار بالاترین دیسک باید حرکت داده شود.

۲) دیسک بزرگتر بر روی دیسک کوچکتر قرار نگیرد.

۳) در هر بار حرکت فقط یک دیسک را می‌توان انتقال داد.

این معما را می‌توان تعمیم داد و تعداد دیسک‌ها را به جای هشت تا، nتا در نظر گرفت. چنانچه n را برابر ۲ بگیریم و معما را حل کنیم شناخت بهتری راجع به مسأله پیدا خواهیم کرد. برای حل مسأله در این حالت ابتدا دیسک بالا را از محور A به محور B منتقل می کنیم، در مرحله بعد دیسک دیگر حول محور A به محور C منتقل می‌گردد و در نهایت دیسک محور B را به مجور C منتقل می‌کنیم.



شکل ۳-۲: وضعیت اولیه مسأله برج هانوی

حال مسأله را به n=۳ تعمیم می‌دهیم.

اگر به طریقی بتوانیم دو دیسک بالا از سه دیسک محور A را به محور B منتقل کنیم آنگاه دیسک آخر را می‌توان به محور C منتقل کرد و سپس دو دیسک موجود حول محور B را به محور C منتقل کرد. مراحل انجام کار به‌صورت زیر می‌باشد:

الف) دو دیسک بالا از سه دیسک محور A به محور B منتقل شود.

ب) آخرین دیسک محور A به محور C منتقل شود.

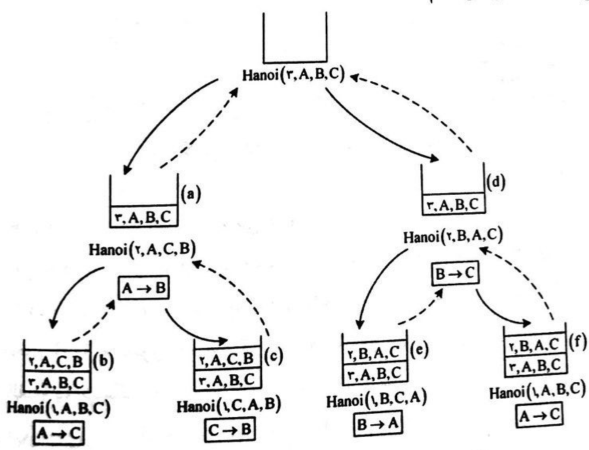
ج) دو دیسک حول محور B به محور C منتقل شود.

Scanned with CamScanner

این روند را می‌توان ادامه داد و مسأله برج هانوی برای n دیسک را حـل کـرد. در واقـع شاهکار روش بازگشتی در این است که حـل یـک مـسأله بزرگتـر را منـوط بـه حـل مـسأله کوچکتر می‌کند و مسأله کوچکتر را به مسائل کوچکتر، تا بـالاخره مـسأله بـسیار کوچـک بـه روش ساده حل شود و از حـل آن بترتیـب عکـس مـسائل بزرگتـر حـل مـی‌گـردد. در زیـر الگوریتم Hanoi نمونه‌ای از هنر طراحی الگوریتمهای بازگشتی را به نمایش مـی‌گـذارد. مقـدار n (تعداد دیسک ها) و محورهای A، B و C به‌عنوان ورودی الگوریتم زیر می‌باشند:

```
void  Hanoi ( int  n , peg  A ,  peg  B ,  pag  C )
{
        // دیسک‌های محور A به محور B منتقل شود
      if ( n = = 1 )
            move top Disk on A to C ;
      else{
            Hanoi (n-1, A, C, B) ;
            Move top Disk on A to C ;
            Hanoi (n-1, B, A, C)  ;
  }
```

الگوریتم بالا را برای n=۳ با توجه به مراحل اجرای یـک الگـوریتم بازگـشتی در شکل ٤-۲ نمایش می‌دهیم.



شکل ٤-۲: مراحل اجرای الگوریتم بازگشتی برج هانوی

Scanned with CamScanner

در شکل بالا فلش‌های توپر مرحله فراخوانی تابع و فلش‌های با خطوط منقطع مرحله بازگشت را نمایش می‌دهند.

۳-۲ حل تعدادی مثال برای محاسبه الگوریتمهای بازگشتی

در این بخش قصد داریم با ارائه مثال‌هایی، طریقه محاسبه مقدار، برای الگوریتمهای بازگشتی را نشان دهیم. برای اینکه روش محاسبه مقدار تابع بازگشتی، مسئله بسیار مهمی است که در فصول بعد اهمیت آن را خواهیم دید.

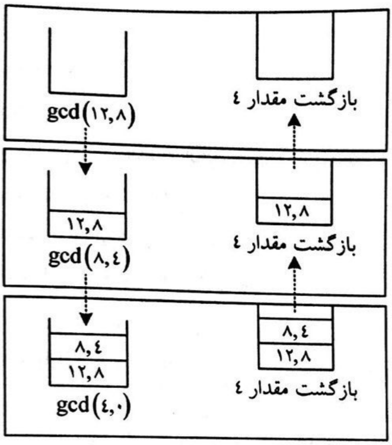
مثال ۱-۲: تابع بازگشتی برای محاسبه بزرگترین مقسوم‌علیه مشترک به روش اقلیدسی بنویسید.

روش اقلیدسی محاسبه بزرگترین مقسوم‌علیه مشترک دو عدد صحیح نامنفی یکی از الگوریتمهای بسیار قدیمی است که بر اساس تفکر بازگشتی بنا شده است. فرض کنید a و b دو عدد صحیح غیرمنفی باشند و $a \leq b$. چنانچه $b = ۰$ باشد، آنگاه بزرگترین مقسوم‌علیه مشترک a و b طبق تعریف همان a خواهد بود. در غیر این صورت، بزرگترین مقسوم‌علیه مشترک b و باقیمانده a بر b خواهد بود. تابع این الگوریتم به‌صورت زیر می‌باشد:

```
int  gcd ( int a , int b )
{
    if ( b = = 0 )
       return a ;
    else
       return (gcd ( b , a%b ) ) ;
}
```

حال الگوریتم بالا را بازای a =۱۲ و b=۸ برای محاسبه بزرگترین مقسوم‌علیه مشترک دو عدد در شکل نمایش می‌دهیم.

Scanned with CamScanner



شکل ۵-۲: مراحل اجرای الگوریتم gcd

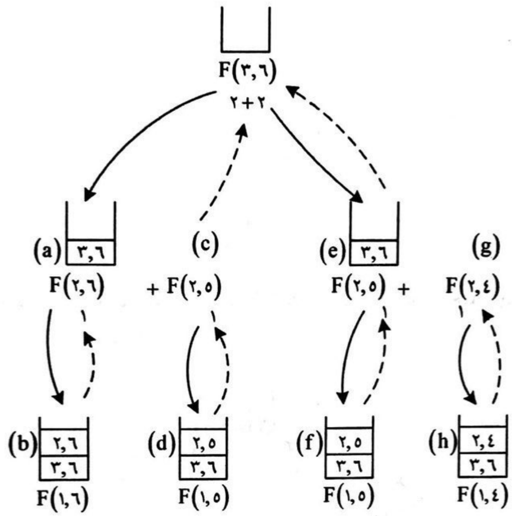
شکل بالا مراحل اجرای الگوریتم بازگشتی، برای محاسبه بزرگترین مقسوم‌علیه مشترک دو عدد a و b با مقادیر بالا را نمایش می‌دهد و در نهایت مقدار ٤ را به‌عنوان خروجی برمی‌گرداند.

مثال ۲-۲: خروجی تابع زیر را به ازای F(۳،٦) محاسبه نمائید:

```
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 ) ) ;
}
```

مراحل اجرای الگوریتم بالا را به ازای مقادیر داده شده، در شکل زیر نمایش می‌دهیم:

Scanned with CamScanner



شکل ٦-٢: مراحل اجرای الگوریتم F

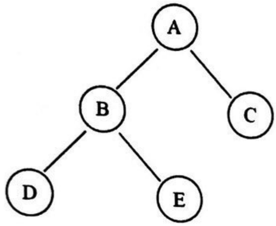
شکل بالا مراحل محاسبه مقدار تابع بازگشتی را در دو مرحله فراخوانی و بازگشت نشان می‌دهد. و در نهایت مقدار ٤ را به‌عنوان خروجی نمایش می‌دهد.

مثال ٣-٢: یک درخت دودوئی را در نظر گرفته سپس خروجی تابع بازگشتی زیر را محاسبه کنید:

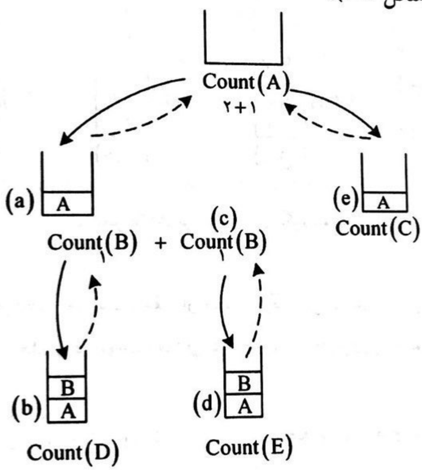
```
int  Count ( Node  * tree )
{
    if ( tree = = Null )
       return 0 ;
    else if (  (tree → left = = Null ) && ( tree → right  = = Null ) )
       return 1 ;
    else
       return (Count (tree → left) + Count (tree → right)) ;
}
```


Scanned with CamScanner

درخت دودونی زیر را در نظر بگیرید:



درخت بالا را به‌عنوان ورودی تابع بازگشتی در نظر گرفته، مراحل اجرای آن را نمایش می‌دهیم (شکل ۷-۲).



شکل ۷-۲: مراحل اجرای الگوریتم Count

با توجه به درودی الگوریتم بالا از اجرا، تعداد گره‌های برگ یک درخت دودوئی را به‌عنوان خروجی برمی‌گرداند.

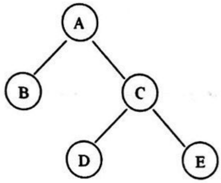
مثال ٤-۲: تابع بازگشتی را زیر بر روی درخت دودوئی T چه عملی انجام می‌دهد:

```
int   Func ( Node * tree )
```

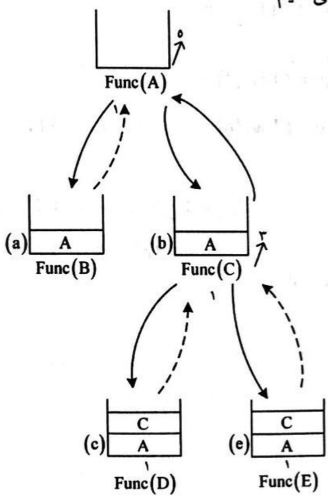

Scanned with CamScanner

```
{
    if ( tree! = = Null )
      if(  ( tree → right = = Null ) && ( tree → left = = Null ) )
          return ( 1 ) ;
     else
          return ( Func (tree → left ) + Func (tree → right ) + 1) ;
}
```

برای درک عمل تابع Func درخت دودوئی زیر را در نظر می‌گیریم:



حال درخت بالا را به‌عنوان ورودی تابع Func در نظر گرفته مراحل اجرای آن را در شکل ۲-۸ نشان می‌دهیم:



شکل ۸-۲: مراحل اجرای الگوریتم Func

Scanned with CamScanner

با توجه به شکل و مراحل اجرای الگوریتمهای بازگشتی تابع Func تعداد کل گره‌های یک درخت دودوئی را می‌شمارد.

۲-۴ خلاصه فصل

- الگوریتمی که برای محاسبه مقدار، خود را به اندازه لازم فراخوانی کند الگوریتم بازگشتی نامیده می‌شود.
- در بسیاری از مسائل که ذاتاً بازگشتی هستند، به‌کارگیری الگوریتمهای بازگشتی ضروری بنظر می‌رسد.
- الگوریتمهای بازگشتی برای محاسبه مقدار از دو مرحله: فراخوانی و بازگشت استفاده می‌کند.

۲-۵ تمرینات

۱. در تابع زیر List را یک آرایه n عنصری در نظر بگیرید:

```
int     S( int   List [ ] ,  int   n )
{
        if (n = = 1)
           return ( List [1] ) ;
        else
           return ( List [n]  +  S ( List , n – 1 ) ) ;
}
```

خروجی تابع بالا را تعیین کنید. در حالت کلی تابع بالا چه عملی انجام می‌دهد؟

۲. تابع Func را به‌صورت زیر در نظر بگیرید:

```
int   Func ( int   n  )
{
     if (  n = = 1 )
        return (1) ;
    else
       return ( n + func ( n – 1 ) ) ;
}
```

خروجی تابع بالا را به ازای n=۸ محاسبه کرده، سپس در حالت کلی عملی که تابع Func روی n انجام می‌دهد را بیان کنید؟

Scanned with CamScanner

۳. در تابع زیر List یک آرایه n عنصری از اعداد صحیح می‌باشد:

```
int  M ( int   List [ ] ,  int   n )
{
    if (n = = 1)
       return (List [1] ) ;
    else
      return (List [n]  , M ( List , n – 1 ) ) ;
}
```

خروجی تابع M را برای ۱۰ عدد صحیح محاسبه نموده، سپس عملی که تابع M در حالت کلی روی عناصر آرایه List انجام می‌دهد را بیان کنید؟

٤. تابع test یک درخت دودوئی را دریافت می‌کند:

```
int   test ( Node  * tree  )
{
     if (tree = = Null)
         return 0 ;
   else
       return (1+max ( test( tree → left ) + test ( tree → right) )) ;
```

خروجی تابع test را در حالت کلی پیدا کرده سپس مراحل محاسبه تابع بازگشتی را برای آن بنویسید.

۲-٦ تمرینات تکمیلی

۵. الگوریتم بازگشتی بنویسید تا کلیه ترکیبات ارقام ۱ تا n را تولید نماید.
توجه کنید کلیه ترکیبات ارقام از ۱ تا n-۱ را داشته باشیم.

٦. الگوریتم بازگشتی برای یافتن بیشترین مقدار در یک آرایه از اعداد صحیح بنویسید.
سپس مراحل محاسبه مقدار تابع بازگشتی را با ارائه مثالی بحث نمائید.

۷. الگوریتم بازگشتی طراحی کنید تا یک ماتریس n×n را دریافت کرده سپس دترمینان ماتریس را به‌عنوان خروجی برمی‌گرداند.

۸. یک درخت دودوئی را درنظر گرفته، تابع بازگشتی بنویسید که تعداد گرههای غیربرگ درخت را بشمارد.

۹. الگوریتم بازگشتی برای محاسبه تعداد گرههای دو فرزندی یک درخت دودوئی بنویسید.

Scanned with CamScanner

۱۰. صفحه‌ای به ابعاد 3*n موجود است. می‌خواهیم این صفحه را با موزائیک‌های 1*2 فرش کنیم. رابطه بازگشتی برای حل این مسئله ارائه دهید.

۱۱. صفحه‌ای به ابعاد 2*n موجود است. می‌خواهیم این صفحه را با موزائیک‌های 1*2 فرش کنیم. به چند طریق می‌توان اینکار را انجام داد (رابطه‌ای بازگشتی بیابید).

۱۲. روی محیط دایره‌ای چند کارت کوچک سیاه و سفید قرار داده ایم. دو نفر به نوبت این عمل را انجام می‌دهند. اولی همه کارتهای سیاهی که در مجاورت یک سفید باشند، برمی‌دارد و دومی روی کارتهای سفید مجاور سیاه همین کار را انجام می‌دهد. اگر ٤٠ کارت وجود داشته باشد، آیا ممکن است پس از انجام ۲ حرکت فقط یک کارت باقی بماند؟ اگر کارتها ۱۰۰۰ باشد، حداقل چند حرکت لازم است؟ اگر n کارت دور دایره باشد، حداقل چند حرکت لازم است؟

۱۳. مسئله برج‌های هانوی را در نظر گرفته، رابطه بازگشتی برای حل آن بنویسید.

Scanned with CamScanner