۱- الگوریتم های بازگشتی

۱-۱- مقدمه:

توابع بازگ شتی توابعی ه ستند که داخل تعریف شان دوباره ازخود تابع ا ستفاده می شود.این توابع به علت اینکه خوانایی بیشتری نسبت به توابع دیگر دارند و کد کمتری برای نوشتن آن ها لازم است دربرنامه نویسی طرفداران زیادی دارند.

توابع بازگشتی مشهور عبارت اند از:تابع فاکتوریل،تابع فیبونانچی،توان، ضرب و...

۱-۲- مفهوم بازگشتی:

حل کردن م سائل از طریق بازگ شتی معمولا از چند بخش ت شکیل شده ا ست.وقتی که یک تابع بازگ شتی برای حل یک م سئله خوانده می شود یا برای حل بخش پایه فراخوانده شده ا ست یا توانایی حل ق سمتی ازم سئله رادارد وبرای حل قسمت دیگرباید خودش راصداکند.مثلابرای محاسبه ی !۳ به روش بازگشتی از تعریف می دانیم ۳ × !۲ = !۳. آن را به دو بخش مجزا که حل آن را می دانیم (×۳)وبخش !۲ تقسیم می کنیم. برای !۲ داریم !۱×۲=!۲ و دوباره آن را به دو بخش (×۲) و !۱ تقسیم می کنیم. !۱ حالت پایه است که آن را می شناسیم . پس نتیجه این شد که اگر تابع با بخش پایه صدا شد معمولا نتیجه ای را باز می گرداند و اگر با بخش پیچیده تری صداشد، معمولا جواب به طور نظری به دو بخش تقسیم می شود . یک بخش که تابع می داند چه طور آن را حل کند (×۳ در مثال بالا) و یک بخش که نمی داند (!۲ در مثال بالا) برای این که بتوانیم تابع را به صورت بازگ شتی تعریف کنیم بخ شی که حل آن را نمی دانیم باید قابل تبدیل به م سئله ای ساده تر یا کوچکتر از آن باشد. به خاطر این که این مسئله ی جدید شبیه به مسئله ی اولیه است تابع بازگشتی ، تابعی تازه از خودش را صدا می زند تا مسئله ی کوچک تر را حل کند. به این مرحله ((فراخوانی بازگشتی)) یا ((گام بازگشتی)) می گوییم.

این طرز فکر که مسئله رابه دو بخش کوچک تر تبدیل و آن ها را حل کنیم روش بازگشتی را جزو روش های تقسیم و غلبه قرار می دهد.

انواع بازگشت:

مفهوم بازگشتی در ریاضیات کاربرد داردکه نمونه های زیر مثال هایی از این نوع اند:

۱- دنباله های بازگشتی:

$$\begin{cases} S(n) = 2S(n-1) \\ S(0) = 1 \end{cases}$$

که این دنباله توان های ۲ را به ما می دهد:

اعداددنباله ی توانهای۲

n =0	n =1	n =2	n =3	n =4	…….	n =m
S =1	S =2	S =4	S =8	S =16	…….....	S = 2^m

$$\begin{cases} S(n) = 3S(n-1) + 2 \\ S(0) = 2 \end{cases}$$

که این دنباله اعدادی را به ما می دهد که در تقســـیم به ۳ با قیمانده ی یکســـانی دارند.(به پیمانه ی ۳ همنهشتند).

اعداد همنهشت به پیمانه ی ۳

n =0	n =1	n =2	n =3	n =4	…….	n =m
S =2	S =8	S =26	S =80	S =242	…….....	S =3^(m+1) - 1

۲- مجموعه های بازگشتی:

$$\begin{cases} \text{پدر و مادر قلی جزو اجداد او هستند} \\ \text{هر پدر و مادر جد مجموعه ی اجداد قلی هستند} \\ \text{این مجموعه شامل تمام اجداد قلی است} \end{cases}$$

۳- عملگرهای بازگشتی:

$$\begin{cases} M(1) = m \\ M(n) = M(n-1) + m \end{cases}$$

این عملگر ضرب را با استفاده از تعریف ضرب ($n \times m = (n-1) \times m + m$) و عملگر جمع می سازد.

$$\begin{cases} M(1) = m \\ M(n) = M(n-1) * m \end{cases}$$

این عملگر توان را با استفاده از تعریف توان ($m^n = m * m^{n-1}$) و عملگر ضرب می سازد.

۴-الگوریتم های بازگشتی:

در این فصـــل به الگوریتم‌های بازگشـــتی و تحلیل آنها می‌پردازیم. بســـیاری از مســـائل را می‌توان به کمک الگوریتم‌های بازگشتی حل کرد. هر الگوریتم بازگشتی معمولا از ۳ مرحله تشکیل شده است:

۱.تقسیم کردن مسئله اصلی به زیر مسئله‌های کوچک تر از همان مسئله.

۲.حل کردن زیر مسئله‌های کوچک تر به صورت بازگشتی.

۳.ترکیب کردن جواب‌های به د ست آمده از حل زیر م سئله‌های کوچک تر برای به د ست آوردن جواب م سئله اصلی.

به ۳ مرحله بالا که در کنار هم به حل مســـئله‌های بازگشـــتی منجر میشـــوند، تقســـیم و غلبه یا تقســـیم و حلِ(Divide and conquer)می‌گویند.

به عنوان مثال فرض کنید داریم:

$$\begin{cases} F(0) = F(1) = 1 \\ F(n) = F(n-1) + F(n-2) \end{cases}$$

در این جا حالت پایه می شود F(0) و F(1) که جواب را می دانیم.

F(n-1) و F(n-2) بخش هایی از الگوریتم هستند که حل آن ها را نمی دانیم.

بخشی که حل آن را می دانیم این است که با استفاده ازجمع F(n-1) و F(n-2) می توانیم F(n) را بسازیم.

در کل می‌توان ۲ مرحله برای این الگوریتم‌ها در نظر گرفت:

۱)هنگامی که زیر مسئله‌ها به اندازه‌ای کوچک شده باشند که جواب آنها بدیهی است و نیازی به بازگشت نیست به این مرحله، مرحله پایانی یا حالت بدیهی می‌گوییم .که معمولا در ابتدای یک الگوریتم بازگشـــتی به عنوان شرط خاتمه چک میشوند.

۲)هنگامی که زیر مسئله‌ها به اندازه کافی بزرگ باشند که به صورت بازگشتی حل شوند، به این مرحله، مرحله بازگشت(Recursion Step) می‌گوییم.این مرحله باید به گونه ای باشد که زیر مسئله را به سمت شرط خاتمه پیش ببرد.

```
int Fact (int n) {
    (   if (n == 0)
return 1;   ) ==========> شرط خاتمه
    (   return n*Fact (n-1);  )=========>   مرحله بازگشتی
}
```

تحلیل الگوریتم‌های بازگشـــتی برای به دســـت آوردن زمان اجرا یا حافظه مصـــرفی آنها معمولاً به یک رابطه بازگشتی منجر میشود. رابطه بازگشتی، یک معادله یا نا معادله است که یک تابع را بر حسب مقدارهای خود آن تابع به ازای ورودی‌های کوچکتر بیان می‌کند. مثلا بدترین زمان اجرای الگوریتم مرتب سازی ادغامی را به کمک رابطه بازگشتی زیر بیان می کنیم:

$$T(n) = \begin{cases} \theta(1), & if\ n = 1 \\ 2\,T\left(\frac{n}{2}\right) + \theta(n), & if\ n > 1 \end{cases}$$

که در نهایت به $T(n) = \theta(n \lg n)$ منجر خواهد شد.

نکته : مسائلی که به صورت روابط ریاضی و محاسباتی هستند یا میتوان آنها را به زیر مسئله هایی کوچکتر از همان مسئله تبدیل کرد را میتوان با استفاده از روش بازگشتی حل کرد.

مثال ۱)

```
int sum (int n) {
        if (n== 0)
                return 0;
        return sum(n-1) + n;
}
```

$$T(n) = \begin{cases} \theta(1), & if\ n = 0 \\ T(n-1) + \theta(1), & if\ n > 0 \end{cases}$$

مثال ۲)

```
void print_rev() {
        char ch;
        if( (ch = getchar()) !=”\n” ) {
                print_rev();
                print (ch);
        }
}
```

$$T(n) = \begin{cases} \theta(1), & if\ n = 0 \\ T(n-1) + \theta(1), & if\ n > 0 \end{cases}$$

این شبه کد یک رشته را بعنوان ورودی میگیرد و آن را برعکس چاپ میکند

در این فصل ۵ روش را برای حل رابطه‌های بازگشتی معرفی خواهیم کرد که معمولاً برای به دست آوردن مرتبه تابع‌های مورد نظر کافی هستند:

۱.حدس جواب و اثبات با استقرا

۲.تکرار و جایگزینی

۳.درخت بازگشت

۴.قضیه اصلی

۵.روشهای خلاقانه ی دیگر

در عمل، برای حل روابط بازگشتی از برخی جزئیات چشم پوشی می‌کنیم:

۱.معمولاً از سقف و کف در روابط صرف نظر می‌کنیم. این‌ها در اکثر موارد در جواب تاثیری ندارند، پس ابتدا رابطه را بدون در نظر گرفتن آنها حل می‌کنیم و بعداً تعیین می‌کنیم که آیا در جواب تغییری ایجاد کرده اند یا خیر. برای این کار تجربه و تعدادی قضیه به ما کمک میکنند. مثلا هنگام تحلیل زمان اجرای مرتب سازی ادغامی در واقع به رابطه $T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lceil \frac{n}{2} \right\rceil\right) + \theta(n)$ میرسیم اما ما با صرف نظر کردن از سقف و کف به رابطه‌ی $T(n) = 2T\left(\frac{n}{2}\right) + \theta(n)$ میرسیم.

۲.نمونه دیگری از جزئیاتی که از آنها چ شم پو شی می‌کنیم، شرایط مرزی ه ستند. از آنجا که زمان اجرای یک الگوریتم برای ورودی های ثا بت، ثا بت اســـت میتوانیم فرض کنیم که برای n های به ا ندازه کافی کوچک $T(n) = \theta(1)$. مثلا رابطه الگوریتم مرتب ســازی ادغامی را به صــورت $T(n) = 2\,T\left(\frac{n}{2}\right) + \theta(n)$ بیان می‌کنیم بدون این که مقدار دقیق T(n) را برای n های کوچک به صراحت مشخص کنیم. این

کار به آن دلیل است که اگر چه تغییر در مقدار $T(1)$ حل دقیق مسئله را تغییر میدهد، اما جواب نهایی حداکثر به اندازه یک عامل ثابت تغییر می‌کند و در نتیجه مرتبه رشد آن بدون تغییر خواهد ماند.

۳.گاهی هم برای راحتی کار از نماد گذاری‌های مجانبی همچون O و $\boldsymbol{\theta}$ در رابطه‌ها صرف نظر می‌کنیم. مثلا به جای $\boldsymbol{\theta(n)}$، $\boldsymbol{n}$می‌گذاریم و به حل رابطه بازگشتی می‌پردازیم. در عمل این چشم پوشی هم مرتبه رشد تابع را تغییر نخواهد داد و حل رابطه را آسان تر خواهد کرد.

۱-۴ حدس و استقرا

این روش از ۲ مرحله تشکیل شده:

۱.حدس جواب (به صورت دقیق یا با نماد گذاری مجانبی)

۲.استفاده از استقرای ریاضی برای اثبات این که جواب درست است.

این روش روشی قدرتمند است، اما به وضوح باید بتوانیم شکل جواب را حدس بزنیم تا از این روش استفاده کنیم. از روش حدس و استقرا میتوان برای پیدا کردن حدود بالا و پایین برای جواب بازگشتی‌ها استفاده کرد. معمولاً برای اثبات حدس خود از تعاریف نماد گذاری‌های مجانبی$\boldsymbol{(O, \theta, \Omega)}$استفاده می‌کنیم .برای آشنا شدن با این روش، مثال زیر را دنبال کنید.

مثال ۱) رابطه بازگشتی زیر را حل کنید. (تحلیل مرتب سازی ادغامی)

$$\boldsymbol{T(n) = \begin{cases} c_1, & n = 2 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + c_2 n, & n > 2 \end{cases}}$$

حل : با دقت در رابطه متوجه می شویم که مسئله در هر مرحله به ۲ زیر مسئله به اندازه نصف مسئله تقسیم میشود. پس بعد از حدود $\log(n)$ بار به زیرمسئله های با اندازه ۱ خواهیم رسید. با توجه به این که زیر مسئله‌ها را با ضریبی از n با هم ترکیب می‌کنیم، می‌توان حدس زد که جواب $\boldsymbol{\theta(nlgn)}$است. حال با استفاده از استقرا به اثبات ادعای خود می‌پردازیم. با توجه به تعریف $\boldsymbol{\theta}$ کافیست ثابت کنیم برای مقادیر بزرگ n عددی مانند$\boldsymbol{a > 0}$وجود دارد که$\boldsymbol{T(n) \leq a * nlg\, n}$و همچنین عددی مانند b وجود دارد که$\boldsymbol{T(n) \geq b * n \lg n}$.

برای قسمت اول داریم:

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + c_2 n \leq 2T\left(\frac{n}{2}\right) + c_2 n$$

پایه استقرا $n = 2 : T(2) \leq 2a \Rightarrow a \geq \frac{c_1}{2} > 0$:

فرض استقرا $\forall k < n , k \geq 2 : T(k) \leq ck * lg\ k$:

گام استقرا : باید اثبات کنیم $T(n) \leq a * nlg\ n$:

$$T(n) \leq 2a\left(\frac{n}{2}\right)\lg\left(\frac{n}{2}\right) + c_2 n = an * \lg\left(\frac{n}{2}\right) + c_2 n = an * lgn - an + c_2 n$$

که کافیست a را بزرگ تر از c_2 انتخاب کنیم تا حکم ثابت شود.

اثبات قسمت دوم نیز مشابه قسمت اول است و از آنجا b به دست میاید. در واقع ثابت کردیم که :

$$\&T(n) = \Omega(nlgn) \Rightarrow T(n) = \theta(nlgn) T(n) = O(nlgn)$$

مثال ۲) مرتبه رابطه بازگشتی زیر را به دست آورید.

$$T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 1$$

حل : به نظر می‌رسد جواب $O(n)$ است. پس باید اثبات کنیم $T(n) \leq cn$.اگر همانند روش مثال قبل از استقرا استفاده کنیم، خواهیم داشت:

$$T(n) \leq c\lfloor n/2 \rfloor + c\lceil n/2 \rceil + 1 = cn + 1 > cn$$

که ا شتباه ا ست. اما مقداری که در سمت چپ ا ضافه آماده، یک مقدار ثابت یعنی عدد ۱ ا ست و به n بستگی ندارد. در این موارد یک راهکار وجود دارد، این که فرض استقرا را تغییر دهیم و یک مقدار ثابت به آن اضافه کنیم. یعنی فرض کنیم:

$$.T(n) \leq cn + b$$

در این صورت با استقرا خواهیم داشت:

$$T(n) \leq 2c\left(\frac{n}{2}\right) + 2b + 1 = cn + 2b + 1 \ \leq cn + b$$

که برای این که نامساوی آخر برقرار باشد کافیست b را کوچک تر یا مساوی ۱- انتخاب کنیم. پس:

$$T(n) = O\ (n)$$

۱-۱-۴ تغییر متغیر

گاهی اوقات رابطه بازگشتی پیچیده تر است و حدس جواب سخت به نظر می‌رسد. در این موارد گاهی تغییر متغیر به کمک ما میاید. مثال زیر موضوع را روشن تر می‌کند.

مثال ۱) $T(n) = 2T\left(\sqrt{n}\right) + lgn$.

حل :با تغییر متغیر زیر رابطه را حل می‌کنیم:

$$m = lgn \ \Rightarrow T(2^m) = 2T\left(2^{m/2}\right) + m$$

با فرض $S(m) = T(2^m)$ خواهیم داشت :

$$S(m) = 2S\left(\frac{m}{2}\right) + m$$

که با توجه به مثال‌های قبل داریم

$$S(m) = O(mlgm) \ \Rightarrow T(n) = O(lgnlglgn)$$

مثال ۲) $T(n) = \sqrt{n}T\left(\sqrt{n}\right) + nlgn$.

حل :ابتدا طرفین رابطه را بر n تقسیم میکنیم:

$$\frac{T(n)}{n} = \frac{T\left(\sqrt{n}\right)}{\sqrt{n}} + lgn$$

با تغییر متغییر زیر رابطه را حل میکنیم:

$$S(n) = \frac{T(n)}{n} \qquad \text{====>} \qquad S(n) = S\left(\sqrt{n}\right) + lgn$$

با فرض $n = 2^m$ و $H(m) = S(2^m)$ خواهیم داشت:

$$\boldsymbol{H(m) = H\left(\frac{m}{2}\right) + m}$$

با توجه به حدس واستقرا H(m) = O(m)

از آنجا که:

$$\boldsymbol{\frac{T(n)}{n} = S(n) = S(2^m) = H(m) = O(m)}$$ و m=logn

داریم:

$$\boldsymbol{T(n) = O(nm) \quad ====> \quad T(n) = O(nlgn)}$$

۴-۲ تکرار و جایگزینی

در این روش در رابطه بازگشتی برای n که بر حسب $\boldsymbol{n_1, n_2, n_3, \dots, n_k}$ بیان شده، $\boldsymbol{n_1, n_2, n_3, \dots, n_k}$ را با استفاده از خود رابطه بازگشتی جایگزین می‌کنیم و بسط میدهیم. این روند را آنقدر تکرار می‌کنیم تا به جواب نهایی برسیم. از این روش می‌توان برای به دست آوردن جواب دقیق رابطه‌های بازگشتی استفاده کرد. برای این که با این روش آشنا شوید، مثال زیر را دنبال کنید:

مثال 1) رابطه بازگشتی $\boldsymbol{T(n) = 2T(n-1) + 1}$ را حل کنید$\boldsymbol{(T(0) = 1)}$.

حل :

$$\boldsymbol{T(n) = 2T(n-1) + 1 = 2\,[2T(n-2) + 1] + 1 = \; 2^2T(n-2) + 2 + 1}$$

$$\boldsymbol{= \; 2^2[2T(n-3) + 1] + 2 + 1 = 2^3T(n-3) + \; 2^2 + \; 2^1 + \; 2^0 = \cdots}$$

$$\boldsymbol{= 2^nT(0) + \; 2^{n-1} + \; 2^{n-2} + \cdots + \; 2^1 + \; 2^0 = \; 2^{n+1} - 1 \quad (T(0) = 1)}$$

در نتیجه$\boldsymbol{T(n) = \; \theta\,(2^n)}$.

مثال 2) رابطه بازگشتی $\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + n}$ را حل کنید$\boldsymbol{(T(1) = 1)}$.

حل :

$$T(n) = 2T\left(\frac{n}{2}\right) + n = 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n = 2^2T\left(\frac{n}{2^2}\right) + 2n = \cdots$$

$$= 2^iT\left(n/2^i\right) + in = \cdots = 2^{\lg n}T(1) + nlgn = n + nlgn = \theta(nlg\, n)$$

۳-۴ درخت بازگشت

درخت بازگشت روشی است که اکثرا برای حل و مخصوصاً حدس جواب روابط بازگشتی، بسیار به کار می آید. در این درخت هر گره نشان دهنده هزینه یک زیر مسئله در جایی از مجموعه فراخوانی‌های تابع است.مجموع اعداد موجود در هر سطح از درخت برابر هزینه کلی آن سطح است و برای به دست آوردن هزینه کل، هزینه سطرها را با هم جمع می‌کنیم. جوابی که از این جمع به دست می آید یا دقیق است، یا با تقریب خوبی می‌توان از آن برای حدس در روش حدس و استقرا استفاده کرد. از آنجایی که معمولاً جوابی را که توسط درخت بازگشت به دست می‌آوریم (که یک حدس است) بعداً توسط روش‌های دیگر تایید می‌کنیم، هنگام پر کردن اعداد هر سطح و هنگام ساختن درخت میتوانیم کمی بی دقتی را هم تحمل کنیم.

مثال ۱) فرض کنید نوع جدیدی از الگوریتم مرتب سازی ادغامی را برای مرتب کردن یک آرایه n تایی از اعداد به کار می‌بریم. تنها تفاوت روش جدید با روش قبلی این است که هنگام ۲ قسمت کردن آرایه و تقسیم آن به ۲ زیر مسئله کوچکتر، به جای آن که آرایه را به ۲ آرایه مساوی با اندازه $n/2$ تقسیم کنیم به ۲ آرایه با اندازه‌های $n/3$ و $2n/3$ تقسیم می‌کنیم، پس از حل مسئله برای هر یک از آرایه‌های جدید، آن‌ها را با هم ادغام می‌کنیم. مرتبه الگوریتم جدید را به دست آورید.

حل : رابطه بازگشتی که این الگوریتم به دست می دهد، به صورت زیر است:

$$T(n) = \begin{cases} 1, & n = 1 \\ T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n, & n > 1 \end{cases}$$

دقت کنید که برای به دست آوردن بازگشتی بالا، از نشان گذاری‌های مجانبی و سقف و کف صرف نظر کردیم.

همانطور که در شکل زیر می‌بینید، در اولین سطح، محاسبه $T(n)$ منجر به محاسبه $T\left(\frac{n}{3}\right)$ و $T\left(\frac{2n}{3}\right)$ وتولید مقدار ثابت n میشود. در سطح دوم نیز محاسبه $T\left(\frac{n}{3}\right)$ و $T\left(\frac{2n}{3}\right)$ به ترتیب منجر به محاسبه $T\left(\frac{n}{3^2}\right)$ و $T\left(\frac{2n}{3^2}\right)$ و $T\left(\frac{2n}{3^2}\right)$ و $T\left(\frac{2^2n}{3^2}\right)$ و همچنین تولید ثابت‌های $n/3$ و $2n/3$ می شوند که مجموع آنها برابر با n است. اگر همین روند را ادامه دهیم، درخت بازگشت کامل خواهد شد. البته در این مسئله درخت حاصل یک

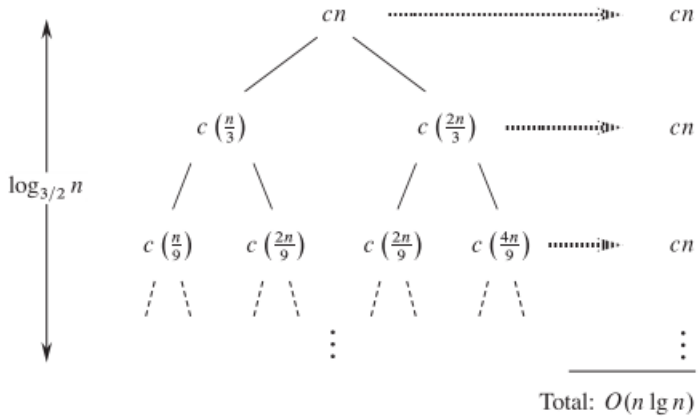
درخت نامتوازن است که عمق سمت چپ آن کمتر است و در عمق $\log_3 n$ به برگ $T(1)$ می‌رسد. در حالی که عمق سمت راست $\log_{3/2} n$ است. با کمی دقت در ساختار مسئله در می یابیم که مقدار ثابتی که از سطح iام بدست میاید اگر $i \leq \log_3 n$ برابر با n و در غیر این صورت کوچک تر از n خواهد بود. چون می‌خواهیم مرتبه جواب را بدست آوریم، میتوانیم فرض کنیم مجموع همه سطح‌ها برابر n است. بنابراین:

$$T(n) \leq n \log_{\frac{3}{2}} n = (n \log_2 n) \log_{3/2} 2 = c_1 nlgn$$

$$T(n) \geq n \log_3 n = (n \log_2 n) \log_3 2 = c_2 nlgn$$

و در نتیجه $T(n) = \theta(nlgn)$. حال با استفاده از روش حدس و استقرا میتوانیم از جواب خود مطمئن شویم.

دقت کنید که در اینجا کمکی که درخت بازگشت به ما کرد حدس زدن شکل جواب بود. البته با توجه به قسمت‌های قبل شما این جواب را دیده بودید، اما هنگام رویارویی با یک مسئله جدید، شاید فقط به کمک درخت بازگشت بتوان یک حدس خوب زد.



شکل ۱- درخت بازگشت برای عبارت T(n) = T(n/3) + T(2n/3) + cn

۴-۴ قضیه اصلی

قضیه اصلی یک راه حل مشخص برای رابطه‌های بازگشتی به صورت $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ (که در آن‌ها a، b و f هر کدام شرایط خاصی دارند که در ادامه می آید) ارائه میدهد. این رابطه میتواند زمان اجرای الگوریتمی را بیان کند که مسئله‌ای با اندازه n را به a زیر مسئله با اندازه های n/b تقسیم می‌کند.پس از حل شدن زیر مسئله‌ها با همین الگوریتم برای ترکیب آنها نیاز به زمانی برابر با $f(n)$ داریم.

قضیه اصلی به صورت زیر بیان میشود:

فرض کنید $a \geq 1$و $b > 1$ و $\varepsilon > 0$ و $f(n)$ تابعی‌است که به صورت مجانبی مثبت است. رابطه بازگشتی

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

برای مقادیر مثبت n تعریف شده است. در این صورت:

الف) اگر $f(n) = O\left(n^{\log_b a - \varepsilon}\right)$ (یعنی آهنگ ر شد تابع $f(n)$ از تابع $n^{\log_b a}$ به صورت چند جمله‌ای کمتر باشد) در این صورت: $T(n) = \theta\left(n^{\log_b a}\right)$.

ب) اگر $f(n) = \theta\left(n^{\log_b a}\right)$ ، در این صورت: $T(n) = \theta\left(n^{\log_b a}\, lgn\right)$.

ج) اگر $f(n) = \Omega\left(n^{\log_b a + \varepsilon}\right)$ ،در این صورت: $T(n) = \theta\left(f(n)\right)$.

در واقع آهنگ ر شد دو تابع $f(n)$ و $g(n) = n^{\log_b a}$ را با هم مقایسه می‌کنیم و آن تابعی که درجه ر شد بیشتری دارد، به نوعی جواب را تعیین می‌کند.در مثال‌های زیر کاربرد این قضیه نشان داده شده است.

مثال 1) $T(n) = 9T\left(\frac{n}{3}\right) + n$.

حل :

$$a = 9, b = 3; \quad g(n) = n^{\log_3 9} = n^2; \quad f(n) = n = O\left(n^{2-0.5}\right) \Rightarrow T(n) = \theta\left(n^2\right)$$

مثال 2) $T(n) = T\left(\frac{2n}{3}\right) + 1$.

حل :

$$a = 1, b = 3/2; \quad g(n) = n^{\log_{\frac{3}{2}} 1} = 1; \quad f(n) = 1 = O\left(n^0\right) \Rightarrow T(n) = \theta\left(lgn\right)$$

مثال ۳) $T(n) = 3T\left(\frac{n}{4}\right) + nlgn$.

حل :

$$a = 3, b = 4; \quad g(n) = n^{\log_4 3}; \quad f(n) = nlgn = \Omega\left(n^{\log_4 3 + 0.1}\right) \Rightarrow T(n) = \theta(nlgn)$$

البته ذکر این نکته ضروری است که ۳ حالت فوق، همه ی حالت‌ها را در بر نمی گیرند و توابعی پیدا میشوند که هیچ یک از این حالت‌ها برای آنها صادق نیست. در این موارد نمی‌توان از قضیه اصلی استفاده کرد و باید از روش‌های دیگر حل رابطه بازگشتی به جواب رسید.

به عنوان نمونه قضیه اصلی در رابطه $T(n) = 2T\left(\frac{n}{2}\right) + nlgn$ قابل استفاده نیست. در این عبارت داریم:

$$a = 2, b = 2; \quad g(n) = n^{\log_2 2} = n; \quad f(n) = nlgn$$

شاید در ابتدا این طور به نظر برسد که این رابطه حالت سوم قضیه اصلی است اماچون باید f(n) / g(n) یک عبارت چندجمله ای باشد تا رابطه در حالت سوم قرار گیرد و در این جا حاصل f(n) / g(n) برابر با lgn است که این عبارت کوچکتر از هر عبارت به صورت n^{ϵ} که $\epsilon > 0$ است(به صورت حدی)، در نتیجه رابطه بازگشتی در حالتی بین حالت دوم و سوم قضیه اصلی قرار می گیرد و نمی توان از قضیه اصلی برای حل این رابطه استفاده کرد.

مثال ۴) $T(n) = 2T\left(\frac{n}{2} + 17\right) + n$.

حل : اضافه کردن مقدار ثابت ۱۷ به رابطه $T(n) = 2T\left(\frac{n}{2}\right) + n$ تاثیری در مرتبه پاسخ ندارد زیرا به ازای nهای بسیار بزرگ تقریبا $\frac{n}{2} + 17 = \frac{n}{2}$ است لذا داریم : $T(n) = 2T\left(\frac{n}{2}\right) + n$

پاسخ رابطه فوق طبق قضیه اصلی برابر است با : O(nlgn)

۴-۵ روشهای خلاقانه

تنها تعداد کمی از مسائلی که ما با آنها مواجه میشویم با استفاده از روشهای بالا قابل حل است و بسیار دیگر برای حلشان باید از روشهای خلاقانه و ایده های نو استفاده کرد.

مثال ۱)

$$\binom{n}{m} = \frac{n!}{m!\,(n-m)!} \begin{cases} 1, & if\ m = 0\ \text{or}\ m = n \\ \binom{n-1}{m} + \binom{n-1}{m-1}, \end{cases}$$

باتوجه به رابطه فوق پیچیدگی زمانی شبه کد زیر را بدست آورید.

```
Int comb( int n , int m ){
    If( m == n || m == 0 )
        Return 1;
    Return comb(m , n-1) + comb(m-1 , n-1 );
}
```

حاصل تابع فوق برابر است با مجموع تعداد یک هایی که در طول اجرا کد return می شوند.بنابراین پیچیدگی زمانی این شبه کد بصورت زیر است :

$$T(n, m) = \begin{cases} 0, & if\ m = 0\ \text{or}\ m = n \\ 1 + T(n-1, m) + T(n-1, m-1), \end{cases}$$

تغییر تابع زیر را درنظر میگیریم:

$$T(n, m) = H(m, n) - 1$$

پس داریم:

$$H(n,m) - 1 = \begin{cases} 0, & if\ m = 0\ or\ m = n \\ H(n-1,m) + T(n-1,m-1) - 1, & O.W \end{cases}$$

$$H(n,m) = \begin{cases} 1, & if\ m = 0\ or\ m = n \\ H(n-1,m) + T(n-1,m-1), & O.W \end{cases}$$

که این رابطه همان رابطه ی ذکر شده در صورت سوال است پس :

$$H(n,m) = \binom{n}{m} = T(n,m) + 1$$

در نتیجه :

$$T(n,m) = \binom{n}{m} - 1 = \begin{cases} 0, & if\ m = 0\ or\ m = n \\ \binom{n-1}{m} + \binom{n-1}{m-1} - 1, & O.W \end{cases}$$

مثال ۲)پیچیدگی زمانی شبه کد زیر را بدست آورید.

فرض کنید ftabآرایه یک بعدی است که مقادیر آن در ابتدا -۱ است.

```
Fib(n){
if ftab[n]>=0 then return ftab[n]
if n = 0 or n = 1
Then ftab[n] = n; return n;
ftab[n]=Fib( n -1 ) + Fib( n-2 )
return ftab[n];
}
```

حل : این یک الگوریتم پویاست که در آن هر درایه یه آرایه ftab فقط یک بار وزمانی که مقدار فعلی آن منفی با شد محا سبه می‌شود. بنابراین هیچ زیر مسئله ای بیش از یک بار فراخوانی نمی‌شود پس مرتبه الگوریتم خطی است.O(n)

چند رابطه سودمند

$$\sum_{i=1}^{n} a_i = \frac{a_1}{1-c}$$

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

$$H_n = \sum_{n=1}^{\infty} \frac{1}{n} = ln n + ^{\gamma} + O\left(\frac{1}{n}\right)$$

e^x = 1 + x + x^2/2! + x^3/3! + … 1+x< e^x <1 + x + x^2

تقریب استرلینگ:

$$n! = \sqrt{2\pi n}\left[\frac{n}{e}\right]^n (1 + O(\frac{1}{n}))$$

نتیجه :

N! = O (2^n)

log (n!) = ɵ (nlgn)

$f^{j}(n) = f(f(f(\ldots(n)))$

j بار

$\lg^{*} n = \min(j>=0 ; \log^{j} n <=1)$