دانشکده‌ی علوم ریاضی

ساختمان داده ۱۳ آبان ۹۲

جلسه‌ی ۶: اثبات و کاربرد قضیه اصلی

مدرّس: دکتر شهرام خزائی *نگارنده: امیر قوام، مهدی یوسفی، بهراد معینی*

۱ مروری بر قضیه اصلی

برای حل روابط بازگشتی روش‌های مختلفی وجود دارد. یکی از روش های مهم در این زمینه قضیه اصلی است که برای حالت‌های خاصی از رابطه بازگشتی $T(n) = aT(\frac{n}{b}) + f(n)$ روشی را ارایه می‌کند.
وقتی می‌خواهیم از روش تقسیم و حل برای حل رابطه بازگشتی $T(n)$ استفاده کنیم $T(n)$ را به a زیر مساله که هرکدام به طول $\dfrac{n}{b}$ است تقسیم می‌کنیم؛ حل هرکدام از قسمت‌ها $T(\dfrac{n}{b})$ زمان می‌برد پس در کل $aT(\dfrac{n}{b})$ طول می‌کشد.
پس از آن که عمل تقسیم انجام شد و زیرمساله‌ها حل شدند، با ترکیب زیرمساله‌ها برای حل مساله کلی به یک هزینه دیگر ($f(n)$) نیازمندیم.
پس هزینه کل برای n برابر $aT(\frac{n}{b}) + f(n)$ است.

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

که در آن "$a \geqslant 1$" برابر تعداد زیرمساله‌ها، "$b > 1 : \dfrac{n}{b}$" برابر اندازه زیرمساله‌ها، و "$f(n)$" برابر هزینه است.
بنابر قضیه اصلی بعضی از معادلات به این شکل را حل می‌کنیم.
طبق قضیه اصلی داریم:

حالت ۱:

$$f(n) = O(n^{\log_b a - \epsilon}) \implies T(n) = \Theta(n^{\log_b a})$$

حالت ۲:

$$f(n) = \Theta(n^{\log_b a}) \implies T(n) = \Theta(n^{\log_b a} \log n)$$

حالت ۳:

$$f(n) = \Omega(n^{\log_b a + \epsilon}) \Longrightarrow T(n) = \Theta(f(n))$$

به مثال های زیر دقت کنید.

مثال ۱ $T(n) = 9T(\dfrac{n}{3}) + n^{1/9}$

به دلیل آن که $n^{1/9} = O(n^{\log_3 9 - \epsilon})$ پس در اولین شرط قضیه اصلی صدق می‌کند و طبق آن $T(n) = \Theta(n^2)$

مثال ۲ $T(n) = 9T(\frac{n}{3}) + n^{1/9} \log n$

به دلیل آن که $n^{1/9} \log n = O(n^{\log_3 9 - \epsilon})$ پس در اولین شرط قضیه اصلی صدق می‌کند و طبق آن $T(n) = \theta(n^2)$

مثال ۳ $T(n) = 9T(\frac{n}{3}) + \frac{n^2}{\log n}$

با کمی دقت در می‌یابیم که هیچ‌کدام از شروط قضیه اصلی برقرار نیست پس این مساله را نمی‌توان با قضیه اصلی حل کرد.

مثال ۴ $T(n) = 9T(\frac{n}{3}) + n^2$

به دلیل آن که $n^2 = \theta(n^{\log_3 9})$ پس در دومین شرط قضیه اصلی صدق می‌کند و طبق آن $T(n) = \theta(n^2 \log n)$

مثال ۵ $T(n) = 9T(\frac{n}{3}) + n^2 \log n$

با کمی دقت در می‌یابیم که هیچ‌کدام از شروط قضیه اصلی برقرار نیست پس این مساله را نمی‌توان با قضیه اصلی حل کرد.

مثال ۶ $T(n) = 9T(\frac{n}{3}) + n^{2/1}$

به دلیل آن که $n^{2/1} = \Omega(n^{\log_3 9 + \epsilon})$ پس در سومین شرط قضیه اصلی صدق می‌کند و طبق آن $T(n) = \theta(n^{2/1})$

۱.۱ صحت قضیه اصلی

نشان‌دادن درست بودن این قضیه در حالت کلی کار ساده‌ای نیست اما در ادامه صحت این قضیه را برای یک حالت خاص بررسی می‌کنیم.
فرض می‌کنیم $f(n)$ تابعی از مرتبه چندجمله‌ای باشد یعنی $f(n) = \Theta(n^d)$ آنگاه :

$$T(n) = aT(\frac{n}{b}) + n^d$$

پس داریم :

$\log_b a > d \longrightarrow T(n) = \Theta(n^{\log_b a})$
$\log_b a = d \longrightarrow T(n) = \Theta(n^{\log_b a})$
$\log_b a < d \longrightarrow T(n) = \Theta(n^d)$

می‌خواهیم مجموع هزینه‌ها را به دست بیاوریم پس درخت بازگشت آن را رسم می‌کنیم. برای هر سطح در درخت هزینه‌های زیر را داریم :

هزینه‌ی رأس‌ها در سطح ۱: cn^d

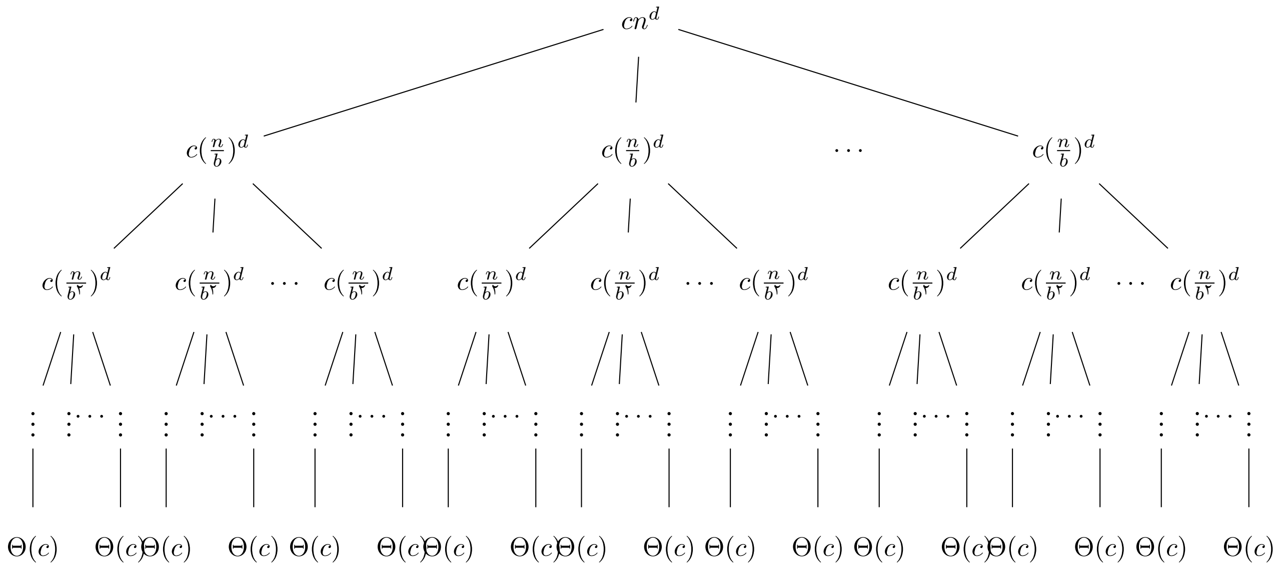
هزینه‌ی رأس‌ها در سطح ۲: $ac(\frac{n}{b})^d$

$\vdots$

هزینه‌ی رأس‌ها در سطح i: $a^{i-1}c(\frac{n}{b^{i-1}})^d$

برای محاسبه هزینه کل ، هزینه‌های هر سطح را با هم جمع می‌زنیم.

مقدار رأس ریشه در درخت برابر n^d است. به جز راس‌های برگ ، هر رأس در این درخت a فرزند دارد که مقدار هر کدام $(\frac{1}{b})^d$ برابر مقدار پدرش است؛ یعنی به عنوان مثال فرزندان ریشه مقداری برابر $(\frac{n}{b})^d$ دارند. و هزینه‌ی هر برگ c است.



با محاسبه جمع مقادیر سطوح مختلف درخت به مجموع کل می‌رسیم:

$$T(n) \leq \sum_{i=0}^{\log_b^n - 1} Ca^i(\frac{n}{b^i})^d + \Theta(n^{\log_b^a})$$

که این مقدار هزینه‌ی متناظر با ریشه $T(n)$ است را تعیین می‌کند.

۲ تحلیل زمانی چند الگوریتم

در حالت کلی روش مشخصی برای بهینه‌سازی الگوریتم‌ها وجود ندارد و بهینه‌سازی الگوریتم ، به خلاقیت نیازمند است تا بتوان الگوریتم‌های با مرتبه زمانی کمتر ارایه داد. با بررسی چند مثال روش‌هایی در این زمینه را معرفی می‌کنیم .

۱.۲ مساله حاصل‌ضرب دو عدد

در ابتدا الگوریتم متداول برای حاصل‌ضرب دو عدد را در نظر می‌گیریم. در این الگوریتم هر کدام از n درایه از x در n درایه y ضرب می شود . واضح است که زمان اجرای این الگوریتم از مرتبه‌ی $\Theta(n^۲)$ است .

همان طور که گفتیم در این بخش برای بهینه کردن مسأله ،الگوریتم‌های دیگری معرفی کنیم. برای به دست آوردن حاصل‌ضرب ، از روش تقسیم و حل استفاده می‌کنیم. مطابق زیر هر عدد n رقمی را به دو بخش $\frac{n}{۲}$ رقمی تقسیم می‌کنیم:

$$\boxed{a \mid b} = ۱۰^{\frac{n}{۲}} \times a + b = x$$

$$\boxed{c \mid d} = ۱۰^{\frac{n}{۲}} \times c + d = y$$

حاصل‌ضرب x و y را می‌توان به صورت زیر بازنویسی کرد:

$$x \cdot y = ۱۰^n a \cdot c + ۱۰^{\frac{n}{۲}}(a \cdot d + b \cdot c) + b \cdot d$$

Algorithm 1 : MULTIPLY

```
function MULTIPLY(x,y)
    n ← size of x
    aL ← left half digits of x
    aR ← right half digits of x
    bL ← left half digits of y
    bR ← right half digits of y
    [if number of digits of x or y is odd then add a 0 in begining of it(x or y)]
    p1 ← MULTIPLY(aL, bL)
    p2 ← MULTIPLY(aL, bR)
    p3 ← MULTIPLY(aR, bL)
    p4 ← MULTIPLY(aR, bR)
    return 10^n p1 + 10^(n/2) (p2 + p3) + p4
```

می‌توان ابتدا هر کدام از حاصل‌ضرب‌های سمت راست تساوی بالا را به صورت بازگشتی می‌توان انجام داد و سپس با استفاده از نتایج به دست آمده حاصل‌ضرب مورد نظر را به دست آورد.
حال اگر تغییر متغیرهای زیر را در نظر بگیرید:

$$p_۱ = a \cdot c \quad , \quad p_۲ = b \cdot d \quad , \quad p_۳ = (a+b)(c+d)$$

حاصل‌ضرب $x \cdot y$ را به صورت زیر می‌توان بازنویسی کرد:

$$۱۰^n p_۱ + ۱۰^{\frac{n}{۲}}(p_۳ - p_۲ - p_۱) + p_۲$$

با نوشتن معادله بازگشتی روشن می‌شود که این روش نسبت به روش قبل زمان اجرای کمتری دارد. در مرحله‌ی تقسیم مسأله به سه زیر مسأله، هر کدام به اندازه‌ی $T(\frac{n}{۲})$ تقسیم می‌کنیم و هزینه‌ی مرحله‌ی ادغام $\Theta(n)$ است، پس معادله بازگشتی آن به صورت زیر است:

$$T(n) = ۳T(\frac{n}{۲}) + \Theta(n) \Rightarrow T(n) = \Theta(n^{\log_۲^۳})$$

Algorithm 2 : MULTIPLY

```
function MULTIPLY(x,y)
    n ← size of x
    if n = 1 then
        return x.y
    aR ← right half digits of x
    aL ← left half digits of x
    bR ← right half digits of y
    bL ← left half digits of y
    [if number of digits of x or y is odd then add a 0 in begining of it(x or y)]
    p1 ← MULTIPLY(aL, bL)
    p2 ← MULTIPLY(aR, bR)
    p3 ← MULTIPLY(aL + bL, aR + bR)
    [until this line is same as privious algorithm]
    return 10^n p1 + 10^(n/2) (p3 − p2 − p1) + p2
```

۲.۲ مساله حاصل‌ضرب دو ماتریس

اگر با الگوریتم متداول یک ماتریس $n \times n$ مانند x را در یک ماتریس $n \times n$ مانند y ضرب کنیم هزینه ضرب از مرتبه $\Theta(n^3)$ است .
برای این کار به روش تقسیم، هر ماتریس $n \times n$ را به چهار زیرماتریس به شکل زیر تقسیم می‌کنیم:

$$X_{(n\times n)} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \quad , \quad Y_{(n\times n)} = \begin{bmatrix} E & F \\ G & H \end{bmatrix} \quad \Rightarrow \quad XY_{(n\times n)} = \begin{bmatrix} AE+BG & AF+BH \\ CE+DG & CF+PH \end{bmatrix}$$

با استفاده از تغییر متغیرهای زیر می‌توانیم الگوریتم را بازنویسی کنیم:

$$T(n) = 8T(\tfrac{n}{2}) + \theta(n^2) \Longrightarrow T(n) = \theta(n^3)$$

الگوریتم حاصل‌ضرب دو ماتریس به همان روش عادی یعنی استفاده از دو حلقه‌ی **for** را در نظر می‌گیریم و می‌دانیم که هزینه‌ی آن $\Theta(n^3)$ است، اما این روش مرتبه را کاهش نمی‌دهد پس به دلیل آن که می‌خواهیم الگوریتم را بهینه‌تر کنیم ، برای این کار از چند متغیر جدید استفاده می‌کنیم.

$$p_1 = (A)(F-H) \ \ , \ \ p_2 = (A+B)(H) \ \ , \ \ p_3 = (C+D)(E) \ \ , \ \ p_4 = (D)(G-E)$$
$$p_5 = (A+D)(E+H) \ \ , \ \ p_6 = (B-D)(G+H) \ \ , \ \ p_7 = (A-C)+(E+F)$$
$$XY_{(n\times n)} = \begin{bmatrix} P_4+P_5-P_2+P_6 & P_1+P_2 \\ P_3+P_4 & P_1+P_5-P_3-P_7 \end{bmatrix}$$

معادله‌ی بازگشتی به صورت زیر می‌شود:

$$T(n) = 7T(\frac{n}{2}) + \Theta(n^2) \quad \Rightarrow \quad T(n) = \Theta(n^{\log_2^7})$$

واضح است که جواب این معادله از مرتبه زمانی $T(n) = \Theta(n^{\log_2^7})$ است.
می‌خواهیم در یک آرایه مرتب ، عدد خاصی را بیابیم. اولین روش آن است که تمام آرایه را بپیماییم که از مرتبه n است. حال می‌خواهیم روشی را معرفی کنیم که بتواند با زمان کمتری این کار را انجام دهد.

۳.۲ جست‌وجوی دودویی

ورودی: آرایه‌ی مرتب $A = \langle a_۱, a_۲, ..., a_n \rangle$ و کلید key
خروجی: اندیس i، به طوری که $A[i] = key$. درصورتی که هیچ iای یافت نشد ۱− برگرداند.
می‌خواهیم در یک آرایه مرتب ، کلید خاصی را بیابیم. اولین روش آن است که تمام آرایه را بپیماییم که از مرتبه n است. در الگوریتمی که در ادامه توضیح داده شده است این کار را با مرتبه زمانی کمتری انجام می دهد. از روش تقسیم استفاده می‌کنیم یعنی آرایه را به دو قسمت تقسیم می‌کنیم و هرکدام را جداگانه در جست‌وجوی دودویی قرار می‌دهیم تاجایی که به جواب برسیم.

Algorithm 3 : Recursive Binary Search

```
function RecursiveBinarySearch(A, key, low, high)
    if high < low then
        return −1
    else mid ← ⌊(low+high)/2⌋
    if key < A[mid] then
        return RecursiveBinarySearch(A, key, low, mid − 1)
    else if key > A[mid] then
        return RecursiveBinarySearch(A, key, mid + 1, high)
    return mid
```

در الگوریتم بعدی جست‌وجوی دودویی به صورت غیربازگشتی ارایه شده است.

Algorithm 4 : Binary Search

```
function BinarySearch(A, n, key)
    low ← 1
    high ← n
    while low ≤ high do
        mid ← ⌊(low+high)/2⌋
        if key < A[mid] then
            high ← mid − 1
        else if key > A[mid] then
            low ← mid + 1
        else
            return mid
    return −1
```