طراحی و تحلیل الگوریتم

استاد:

دکتر زاهد رحمتی

تدریسیاران:

داریوش کاظمی

اشکان ودادی

ترم دوم ۱۴۰۰

جلسه دوم

تحلیل پیچیدگی – Matching – تقسیم و حل

(بخش ۴-۳ کتاب CLRS)

01 Substitution method

02 Iterating the recurrence (Brute Force)

03 Recursion Tree

04 **Master Theorem**

حل مسائل بازگشتی

1. Substitution method
2. Iterating the recurrence (Brute Force)
3. Recursion Tree
4. **Master Theorem**

توابع زیر را بر حسب پیچیدگی زمانی، به صورت صعودی مرتب کنید.

$$2^{2^n}, \lg(n!)\,, 10^{logn}, n^{\frac{1}{lgn}}, n!, e^n, 4^n, \lg(n!)\,, nlgn$$

توابع زیر را بر حسب پیچیدگی زمانی، به صورت صعودی مرتب کنید.

$$2^{2^n}, \lg(n!)\,, 10^{logn}, n^{\frac{1}{lgn}}, n!, e^n, 4^n, \lg(n!)\,, nlgn$$

جواب:

$$n^{\frac{1}{lgn}} < 10^{logn} < \lg(n!) \approx nlgn < e^n < 4^n \; < n! < 2^{2^n}$$

case	solution	conditions
1	$T(n) = \Theta(n^{\log_b a})$	$f(n) = O(n^{\log_b a - \varepsilon})$ for some constant $\varepsilon > 0$
2	$T(n) = \Theta(n^{\log_b a} \log^{k+1} n)$	$f(n) = \Theta(n^{\log_b a} \log^k n)$ for some constant $k \geq 0$
3	$T(n) = \Theta(f(n))$	$f(n) = \Omega(n^{\log_b a + \varepsilon})$ for some constant $\varepsilon > 0$ and $af(n/b) < cf(n)$ for some constant $0 < c < 1$

The Master Theorem takes on a simpler form when $f(n)$ is a polynomial, such that the recurrence has the from $T(n) = aT(n/b) + \Theta(n^c)$ for some constant $c \geq 0$.

Let $a \geq 1$ and $b > 1$ be constants, and (for $n \geq 0$)

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + c \cdot n^d$$

Case1: If $d < \log_b a$, then $T(n) = \Theta(n^{\log_b a})$

Case2: If $d = \log_b a$, then $T(n) = \Theta(n^d \log n)$

Case3: If $d > \log_b a$, then $T(n) = \Theta(n^d)$

For merge sort ($a = 2$, $b = 2$, $d = 1$), we use Case2, $T(n) = \Theta(n \log n)$.

The theorem is as follows:

Given $T(n) = aT(n/b)+f(n)$, where $a \geq 1$ and $b > 1$ and $f(n) = \Theta(n^k \log^p n)$

Consider $log_b(a)$ & k

Case 1: If $log_b(a) > k \Rightarrow T(n) = \Theta(n^{log_b(a)})$

Case 2: If $log_b(a) = k$ &

2.1: If $p > -1 \Rightarrow T(n) = \Theta(n^k \log^{p+1} n)$

2.2: If $p = -1 \Rightarrow T(n) = \Theta(n^k \log \log n)$

2.3: If $p < -1 \Rightarrow T(n) = \Theta(n^k)$

Case 3: If $log_b a < k$ &

3.1: If $p > 0 \Rightarrow T(n) = \Theta(n^k \log^p n)$

3.2: If $p \leq 0 \Rightarrow T(n) = \Theta(n^k)$

Examples (Case 1):

Example 10.1:

T(n) = 2T(n/2) + 1

Here, a = 2, b = 2,

f(n) = Θ (1) = Θ ($n^0 \log^0 n$), .:. k = 0 & p = 0

Now, $\boldsymbol{log_b\,(a) = log_2\,(2) = 1 > k}$

.:., Case 1 is satisfied

T(n) = Θ(n^1)

Examples (Case 2):

Example 10.2:

T(n) = 2T(n/2) + n / log n

Here, a = 2, b = 2,

*f(n) = Θ (n **$\log^{-1}$ n**), .:. k = 1 & p = -1*

Now, $\boldsymbol{log_b\,(a) = log_2\,(2) = 1 = k \;\&\; p = -1}$

.:., Case 2.3 is satisfied

T(n) = Θ(n^k log log n) *= Θ (n log log n)*

Example 10.3:

$T(n) = 2T(n/2) + n / \log^2 n$

Here, $a = 2$, $b = 2$,

$f(n) = \Theta(n \log^{-2} n)$, $\therefore k = 1$ & $p = -2$

Now, $\log_b(a) = \log_2(2) = 1 = k$ & $p < -1$

$\therefore$, Case 2.2 is satisfied

$T(n) = \Theta(n^k) = \Theta(n)$

Examples (Case 3):

Example 10.4:

$T(n) = 2T(n/2) + n^2 \log n$

Here, $a = 2$, $b = 2$,

$f(n) = \Theta(n^2 \log^1 n)$, $\therefore k = 2$ & $p = 1$

Now, $\log_b(a) = \log_2(2) = 1 < k$ & $p > 0$

$\therefore$, Case 3.1 is satisfied

$T(n) = \Theta(n^k \log^p n) = \Theta(n^2 \log n)$

Example 10.5:

$\boldsymbol{T(n) = 4T(n/2) + n^3}$

Here, $a = 4$, $b = 2$,

$f(n) = \Theta\ (n^3) = \Theta\ (n^3 \log^0 n)$, $\therefore$ $k = 3$ & $p = 0$

Now, $\boldsymbol{log_b\ (a) = log_2\ (4) = 2 < k\ \ \&\ p = 0}$

$\therefore$, Case 3.2 is satisfied

$\boldsymbol{T(n) = \Theta(n^k)} = \Theta(n^3)$

The Master Theorem for decreasing functions

$$T(n) = aT(n-b) + f(n) \quad where \;\; a \geq 1\,, b > 0 \quad and \;\; f(n) \;\; is\; asymptotically\; positive$$

The theorem is as follows:

If $T(n) = a\,T(n-b) + f(n)$, where $a \geq 1$, $b > 0$, & $f(n) = O(n^k)$, and $k \geq 0$

Case 1: ***if $a = 1$,***
$T(n) = O(n * f(n))$ or $O(n^{k+1})$

E.g. *1) $T(n) = T(n-1) + 1$* *$O(n)$*
2) $T(n) = T(n-1) + n$ *$O(n^2)$*
3) $T(n) = T(n-1) + \log n$ *$O(n \log n)$*

Case 2: ***if $a > 1$,***
$T(n) = O(a^{n/b} * f(n))$ or $O(a^{n/b} * n^k)$

E.g. *1) $T(n) = 2T(n-1) + 1$* *$O(2^n)$*
2) $T(n) = 3T(n-1) + 1$ *$O(3^n)$*
3) $T(n) = 2T(n-1) + n$ *$O(n\,2^n)$*

Case 3: ***if $a < 1$,***
$T(n) = O(f(n))$ or $O(n^k)$

پیچیدگی زمانی را به کمک Master Theorem بدست آورید.

1. $T(n) = 2^n T(n/2) + n^n$
2. $T(n) = \sqrt{2}T\left(\frac{n}{2}\right) + logn$
3. $T(n) = 0.5T\left(\frac{n}{2}\right) + \frac{1}{n}$
4. $T(n) = 64T\left(\frac{n}{8}\right) - n^2 logn$
5. $T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{2}$
6. $T(n) = T(n/2) + n(2 - cosn)$
7. $T(n) = 2T\left(\frac{n}{4}\right) + n^{0.51}$

پیچیدگی زمانی را به کمک Master Theorem بدست آورید.

1. $T(n) = 2^n T(n/2) + n^n$ → a is not constant!
2. $T(n) = \sqrt{2}T\left(\frac{n}{2}\right) + \log n$ → $T(n) = \Theta(\sqrt{n})$ $(Case\ 1)$
3. $T(n) = 0.5T\left(\frac{n}{2}\right) + \frac{1}{n}$ → a<1!
4. $T(n) = 64T(n/8) - n^2 \log n$ → f(x) is not positive!
5. $T(n) = 3T\left(\frac{n}{3}\right) + \frac{n}{2}$ → $T(n) = \Theta(n \log n)$ $(Case\ 2)$
6. $T(n) = T(n/2) + n(2 - \cos n)$ → We are in Case 3, but the regularity condition is violated. (Consider n = 2πk, where k is odd and arbitrarily large. For any such choice of n, you can show that c ≥ 3/2, thereby violating the regularity condition.)
7. $T(n) = 2T\left(\frac{n}{4}\right) + n^{0.51}$ → $T(n) = \Theta(n^{0.51})$ $(Case\ 3)$

Stable Matching

	1st	2nd	3rd	4th
Ryan	L	S	Z	D
Josh	S	L	D	Z
Blake	S	D	Z	L
Connor	L	S	Z	D

	1st	2nd	3rd	4th
Lisa	R	B	J	C
Sarah	R	B	C	J
Zoey	C	J	R	B
Daniela	R	J	C	B

Stable Matching

	1st	2nd	3rd	4th
Ryan	L	S	Z	D
Josh	S	L	D	Z
Blake	S	D	Z	L
Connor	L	S	Z	D

	1st	2nd	3rd	4th
Lisa	R	B	J	C
Sarah	R	B	C	J
Zoey	C	J	R	B
Daniela	R	J	C	B

تعیین آشپز

فرض کنید c آشپز برای پختن غذا در مراسمی استخدام شده‌اند. این آشپزان بنا دارند f نوع غذای متفاوتی که توسط کارفرما مشخص شده را بپزند. هر کدام از این آشپزها هم ترجیهاتی برای پخت این غذاها دارند (یعنی پختن یه سری غذاها را بیشتر ترجیح میدهند و یک لیست از ترجیهاتشان دارند). فرض کنید برای هر نوع غذا، لیستی وجود دارد که به ترتیب، مهارت و امتیاز آشپز برای پختن آن غذا را نشان می‌دهد که این لیست توسط کارفرما تهیه شده. همچنین فرض کنید تعداد آشپزان از تعداد انواع غذاها بیشتر باشند ($f<c$). یک آشپز یا غذایی نمی‌پزد، یا حداکثر یک نوع غذا را می‌پزد. با شرط اینکه هر غذا توسط یک مفر پخته شود الگوریتمی ارائه دهید که به کمک لیست‌ها، غذاها به بهترین نحو ممکن پخته شوند.

تعیین آشپز

فرض کنید c آشپز برای پختن غذا در مراسمی استخدام شده‌اند. این آشپزان بنا دارند f نوع غذای متفاوتی که توسط کارفرما مشخص شده را بپزند. هر کدام از این آشپزها هم ترجیهاتی برای پخت این غذاها دارند (یعنی پختن یه سری غذاها را بیشتر ترجیح میدهند و یک لیست از ترجیهاتشان دارند). فرض کنید برای هر نوع غذا، لیستی وجود دارد که به ترتیب، مهارت و امتیاز آشپز برای پختن آن غذا را نشان می‌دهد که این لیست توسط کارفرما تهیه شده. همچنین فرض کنید تعداد آشپزان از تعداد انواع غذاها بیشتر باشند (f<c). یک آشپز یا غذایی نمی‌پزد، یا حداکثر یک نوع غذا را می‌پزد. با شرط اینکه هر غذا توسط یک مفر پخته شود الگوریتمی ارائه دهید که به کمک لیست‌ها، غذاها به بهترین نحو ممکن پخته شوند.

جواب: c-f غذای فرضی اضافه می‌کنیم. لیست‌های آنها را به گونه‌ای می‌سازیم که مهارت و امتیاز آشپزان در آنها، از همه‌ی غذاهای دیگر کمتر باشد. با استفاده از الگوریتم Gale-Shapely، میایم یه stable matching پیدا می‌کنیم. اشخاصی که با غذاهای فرضی جفت شده‌اند، غذایی نمی‌پزند و بقیه آشپزان، غذاهایی که به آن‌ها match شده‌اند را می‌پزند.

زوج مرتب ویژه!

زوج مرتبی از اعداد (x,y) را ویژه می‌گوییم هرگاه به شکل روبرو باشند: $x + y = k \, , x < y$.

که k عددی ثابت و از پیش تعیین شده است.

فرض کنید آرایه‌ای طولانی از اعداد صحیح متمایز داریم. با فرض اینکه k ورودی شما بوده و برای شما مشخص شود، الگوریتمی ارائه دهید که بدون حافظه خارجی، تعداد زوج مرتب‌های ویژه آرایه را بشمارد.

زوج مرتب ویژه!

زوج مرتبی از اعداد (x,y) را ویژه می‌گوییم هرگاه به شکل روبرو باشند: $x+y=k\ ,\ x<y$.

که k عددی ثابت و از پیش تعیین شده است.

فرض کنید آرایه‌ای طولانی از اعداد صحیح متمایز داریم. با فرض اینکه k ورودی شما بوده و برای شما مشخص شود، الگوریتمی ارائه دهید که بدون حافظه خارجی، تعداد زوج مرتب‌های ویژه آرایه را بشمارد.

جواب: آرایه را به کمک Merge Sort مرتب می‌کنیم، سپس به ازای هر عضو آرایه در صورت نیاز، از binary search استفاده برای پیدا کردن مکملش استفاده می‌کنیم.

Merge Sort: $O(nlgn)$
n times Binary Search(Worse case scenario): $n \times \lg n \rightarrow O(nlgn)$

خسته نباشید!

داریوش کاظمی – اشکان ودادی