Algorithm Design and Analysis

Week10: Divide and Conquer, Merge Sort, Counting Inversion

Adisak Supeesun

17 February 2022

Divide and Conquer (Intro.)

Merge Sort

Counting Inversion

Divide and Conquer (แบ่งแยก แล้วเอาชนะ)

การทำงานของอัลกอริทึมแบ่งเป็น 3 ขั้นตอน ได้แก่

1. Divide: แบ่งปัญหาที่ต้องการจะแก้ ออกเป็นปัญหาย่อยๆ

   → แบ่ง input ออกเป็นส่วนย่อยๆ

2. Conquer: แก้ปัญหาย่อยแบบ recursive

   → หาคำตอบของปัญหาสำหรับ input แต่ละส่วนย่อย

3. Combine: นำคำตอบของปัญหาย่อยมาสร้างเป็นคำตอบของปัญหาตั้งต้น

Note

เทคนิค divide and conquer ใช้กับปัญหาที่ซับซ้อนมาก แก้ด้วยอัลกอริทึมแบบง่าย เช่น แม้แต่อัลกอริทึม bruteforce ได้ในเวลา polynomial อยู่แล้ว

Sorting Problem

ให้จำนวนเต็ม n ตัว ต้องการเรียงจำนวนเหล่านี้จากน้อยไปมาก

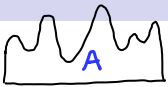
- Selection sort
- Insertion Sort
- Bubble Sort

```
for i = 1, ... , n
    for j = 1, ... , n
    ⋮
```

$O(n^2)$

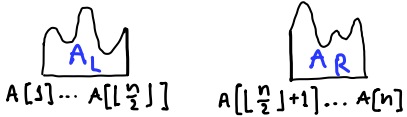
Merge Sort

Input: array A of n integers



$A[1, 2, \ldots, n]$

1. Divide: แบ่งข้อมูลใน A ออกเป็น 2 ส่วนเท่าๆกัน เก็บไว้ในอาร์เรย์ A_L และ A_R



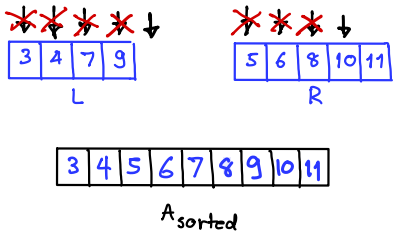
2. Conquer: เรียงข้อมูลใน A_L และ A_R จากน้อยไปมาก โดยการ recursive

3. Combine: นำข้อมูลที่เรียงแล้วทั้ง 2 ชุดมา merge กัน

Merge

การนำข้อมูล 2 แถวที่เรียงแล้วมารวมกัน ให้เป็นข้อมูล 1 แถวที่เรียงจากน้อยไปมาก

- เปรียบเทียบข้อมูล 2 ตัว ที่ pointer ชี้อยู่
- นำตัวที่น้อยกว่าไปใส่ A sorted
- ขยับ pointer ของตำแหน่งที่ถูกนำข้อมูลมาใส่ใน A sorted ไปทางขวา 1 ตำแหน่ง
- ทำไปเรื่อยๆ จนข้อมูลฝั่งใดฝั่งหนึ่งหมด ตามนั้นให้เอาข้อมูลของฝั่งที่เหลือไปใส่ A sorted



เวลาในการทำงาน $O(n)$

(เมื่อ n คือจำนวนข้อมูลสองแถวรวมกัน)

Mergesort($A[1, 2, \ldots, n]$)

```
if n == 1
    return A                                   } O(1)

divide   { A_L ← A[1, 2, ..., ⌊n/2⌋]
         { A_R ← A[⌊n/2⌋ + 1, ..., n]          } O(n)

conquer  { L ← Mergesort(A_L)   — สมมติว่าใช้เวลา T_L
         { R ← Mergesort(A_R)   — สมมติว่าใช้เวลา T_R

combine  { A_sorted ← Merge(L, R)   — O(n)
           return A_sorted
```

Running time of Merge Sort

ให้ $T(n)$ เป็นเวลาที่ merge sort ใช้ในการเรียงข้อมูลข้อมูล n ตัว

$$T(n) = \begin{cases} O(n) + T_L + T_R + O(n) & \text{if } n > 1 \\ O(1) & \text{if } n = 1 \end{cases}$$

โดยที่ $T_L = T(\lfloor \frac{n}{2} \rfloor)$ และ $T_R = T(\lceil \frac{n}{2} \rceil)$

สมมติว่า n เป็นจำนวนเต็มของ 2 จะได้ว่า

$$T(\lfloor \tfrac{n}{2} \rfloor) = T(\lceil \tfrac{n}{2} \rceil) = T(\tfrac{n}{2})$$

$$\therefore T(n) = \begin{cases} 2T(\frac{n}{2}) + cn & \text{if } n > 1 \\ c & \text{if } n = 1 \end{cases}$$

"Recurrence"

Solving Divide and Conquer Recurrences

จาก $T(n) = \underbrace{T(\frac{n}{2})+T(\frac{n}{2})}_{\text{conquer (recursive)}} + \underbrace{cn}_{\text{divide+combine}}$

ทราบรูปแบบปิดของ $T(n)$ เราต้องรู้ค่า $T(\frac{n}{2})$ และทราบค่า $T(\frac{n}{2})$

$T(\frac{n}{2}) \longrightarrow T(\frac{n}{4})$

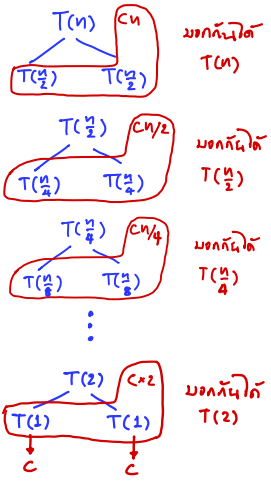
$T(\frac{n}{4}) \longrightarrow T(\frac{n}{8})$

$\vdots$

$T(4) \longrightarrow T(2)$

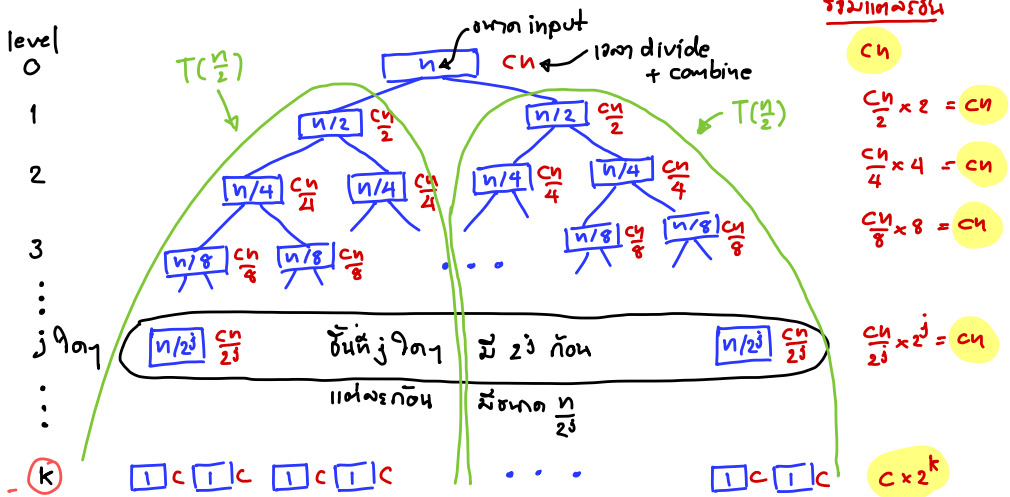
$T(2) \longrightarrow T(1) \leftarrow c$ (base case)

ไม่รู้ ต้องหาต่อ



** เมื่อรู้ $T(1)=c$ จะสามารถแทนค่ากลับเพื่อหา $T(2), T(4), \ldots, T(n)$ **

การแก้ recurrence โดยใช้ recursion tree



รวมเวลา $T(n) = \underbrace{[cn + cn + \ldots + cn]}_{\text{ขั้นที่ 0 ถึง } k-1} + \underbrace{c \times 2^k}_{\text{ขั้นสุดท้าย}} = \underbrace{cnk + c(2^k)}_{\text{ตัวแทน } k}$

เนื่องจากขั้นที่ j ใดๆ มีขนาด input เป็น $\frac{n}{2^j}$
เราจะได้ว่าถ้าเราเข้าขั้นที่ k (ขั้นสุดท้าย)

$\frac{n}{2^k} = 1$ (ขั้นที่ k ขนาด input เป็น 1)

$2^k = n$

$\log_2 2^k = \log_2 n$ (จาก $\log_b a^c = c \log_b a$)

$k \log_2 2 = \log_2 n$ (จาก $\log_b b = 1$)

$k = \log_2 n$

แทน $k = \log_2 n$ ใน $T(n)$ จะได้

$T(n) = cn\log_2 n + c(2^{\log_2 n})$ (จาก $b^{\log_b a} = a$)

$= cn\log_2 n + cn$

$= O(n \log n)$

Counting Inversion

ให้ $A = a_1, a_2, \ldots, a_n$ เป็นลำดับของจำนวนเต็ม n ตัว

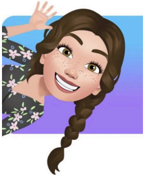
ต้องการหาจำนวน inversion ใน A

นิยาม (inversion) สำหรับลำดับ $a_1, a_2, \ldots, a_n$ เราจะกล่าวว่า a_i และ a_j เป็น "inversion" ถ้า $i < j$ แต่ $a_i > a_j$

Ex จงหาจำนวน inversion ของลำดับ 4, 1, 5, 9, 2

inversion = 4

Application: Collaborative Filtering

A	B	C
		
1. Action	3. Comedy	4. Documentary
2. Sci-fi	5. Drama	2. Sci-Fi
3. Comedy	2. Sci-fi	1. Action
4. Documentary	1. Action	5. Drama
5. Drama	4. Documentary	3. Comedy

inversion ใน rank ของ B = 6

inversion ใน rank ของ C = 5

∴ A คล้าย B มากกว่า C

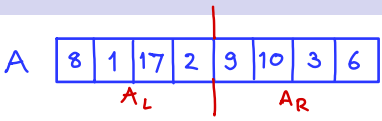
นับจำนวน inversion ใน rank ของ B และ rank ของ C (จำนวนความคู่ของภาพยนตร์ในลำดับความชอบของ B และ C ที่ไม่สอดคล้องกับ A) เพื่อวัดว่าใครคล้าย A มากกว่ากัน

Designing Algorithm

เราสามารถ bruteforce เพื่อนับจำนวน inversion ได้
โดยพิจารณาทุกคู่ของสมาชิกใน A และนับเฉพาะคู่ที่เป็น inversion
เวลาในการทำงาน = $\#\text{คู่ของสมาชิกใน } A \times O(1) = O(n^2)$

$\#\text{คู่ของสมาชิกใน } A = \binom{n}{2} = O(n^2)$

Designing Algorithm (divide & conquer)



$O(n)$ 1. divide: แบ่ง A เป็น 2 ส่วน — $A_L \leftarrow A[1,\ldots,\lfloor \frac{n}{2} \rfloor]$, $A_R \leftarrow A[\lceil \frac{n}{2} \rceil + 1,\ldots,n]$

$T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil)$ 2. conquer: นับจำนวน inversion ใน A_L และ A_R โดย recursive

สมมติได้คำตอบ k_L และ k_R

~~$O(n^2)$~~ พิจารณาทุกคู่ข้ามฝาก

3. combine: นำคำตอบที่ได้ มาสรุปคำตอบของปัญหาทั้งหมด

$O(n)$ ใช้วิธี merge & count (ดูสไลด์ 3 หน้าถัดไป)

$k_L + k_R +$ #inversion ข้ามฝาก (k_{LR})

ถ้าเราใช้เวลา $O(n^2)$ ใน combine, เวลารวมของอัลกอริทึมไม่ดีกว่า brute-force

ให้ $T(n)$ เป็นเวลาที่อัลกอริทึมใช้ในการหาจำนวน inversion ในลำดับของข้อมูล n ตัว

พิจารณาทุกคู่ที่เป็นไปได้

$$T(n) = \begin{cases} T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + cn + \cancel{cn^2} \; cn & \text{if } n > 1 \\ c & \text{if } n = 1 \end{cases}$$

ใช้วิธี merge & count

ต่างที่คูณ n^2 ~~n^2~~ n

สมมติว่า n เป็นกำลังของ 2 (เพื่อความง่ายต่อการแก้ recurrence)

$$T(n) = \begin{cases} 2T(\frac{n}{2}) + \cancel{cn^2} \; cn & \text{if } n > 1 \\ c & \text{if } n = 1 \end{cases}$$

recurrence เดียวกับเวลาการทำงานของ merge sort

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

ข้อสังเกต

ขั้นการรวมแบบ conquer เพื่อนับจำนวน inversion ใน A_L และ A_R มีการสลับลำดับของตัวเองใน A_L และ A_R ที่เรียงจากน้อยไปมากกลับมาด้วย

สมมติว่าลำดับดังกล่าว คือ L และ R ตามลำดับ

- ถ้าตัว i ของ L หรือตัวที่ j ของ R จะได้ว่า
  ตัวที่ i ของ L ไม่มีทางเป็น inversion ด้วยกับตัวที่ j ถึงตัวสุดท้องของฝั่ง R
- ถ้าตัวที่ i ของ L มากกว่าตัวที่ j ของ R จะได้ว่า
  ตัวที่ i จนถึงตัวสุดท้องของ L เป็น inversion ด้วยกับ ตัวที่ j ของ R

Merge_and_Count

// input: L (A_L ที่เรียงแล้ว), R (A_R ที่เรียงแล้ว)

// output: k_{LR} (จำนวน inversion ข้ามฝาก) และ Asorted (A ที่เรียงแล้ว)

A_{sorted} | 1 | 2 | 3 | 6 | 8 | 9 | 10 | 17 |

k_{LR} = #inversion ข้ามฝาก = 0

1<3 : 1 กับ 3 ไม่ใช่ inversion และ 1 กับเลขที่อยู่หลัง 3 ใน R ไม่ใช่ inversion
เอา 1 ไปต่อท้าย Asorted

2<3 : 2 กับ 3 ไม่ใช่ inversion และ 2 กับเลขที่อยู่หลัง 3 ใน R ไม่ใช่ inversion
เอา 2 ไปต่อท้าย Asorted

8>3 : 8 กับ 3 เป็น inversion และตัวเลขหลัง 8 ใน L กับ 3 เป็น inversion

$k_{LR} \leftarrow k_{LR} + 2$, เอา 3 ไปต่อท้าย Asorted

8>6: 8 กับ 6 เป็น inversion และตัวเลขหลัง 8 ใน L กับ 6 เป็น inversion

$k_{LR} \leftarrow k_{LR} + 2$, เอา 6 ไปต่อท้าย A_{sorted}

8<9: 8 กับ 9 ไม่ใช่ inversion และ 8 กับเลขที่อยู่หลัง 9 ใน R ไม่ใช่ inversion

เอา 8 ไปต่อท้าย A_{sorted}

17>9: 17 กับ 9 เป็น inversion และตัวเลขหลัง 17 ใน L กับ 9 เป็น inversion

$k_{LR} \leftarrow k_{LR} + 1$, เอา 9 ไปต่อท้าย A_{sorted}

17>10: 17 กับ 10 เป็น inversion และตัวเลขหลัง 17 ใน L กับ 10 เป็น inversion

$k_{LR} \leftarrow k_{LR} + 1$, เอา 10 ไปต่อท้าย A_{sorted}

เมื่อตัวเลขฝั่ง R หมดแล้ว, นำตัวเลขที่เหลือของฝั่ง L ไปต่อท้าย A_{sorted}

Merge_and_Count(L,R) ใช้เวลา = (#สมาชิกใน L + #สมาชิกใน R) × O(1)

Sort_and_Count(A[1,2,...,n]) //output: (A_{sorted}, # inversions in A)

if n == 1
return (A, 0) } $O(1)$

divide { $A_L \leftarrow A[1, \ldots, \lfloor \frac{n}{2} \rfloor]$
$A_R \leftarrow A[\lfloor \frac{n}{2} \rfloor + 1, \ldots n]$ } $O(n)$

conquer { $(L, k_L) \leftarrow$ Sort_and_Count(A_L) — $T(\lfloor \frac{n}{2} \rfloor)$
$(R, k_R) \leftarrow$ Sort_and_Count(A_R) — $T(\lceil \frac{n}{2} \rceil)$

combine { $(A_{sorted}, k_{LR}) \leftarrow$ Merge_and_Count(L, R) — $O(n)$

return (A_{sorted}, $k_L + k_R + k_{LR}$)