Lab 06: Transform & Conquer Algorithms

1 ตุลาคม 2567

Presorted Uniqueness

ในการแก้ปัญหาหลายๆอย่างนั้น สามารถทำได้หลายวิธี และรูปแบบการแก้ปัญหาแตกต่างกัน แต่มีวิธีการหนึ่งที่เปลี่ยนรูปแบบข้อมูลเพื่อให้สามารถแก้ปัญหาได้ดีขึ้น คือ Transform and Conquer ปัญหาของการตรวจสอบจำนวนซ้ำ หรือ Uniqueness หากแก้ปัญหาด้วยการวนเพื่อหาจำนวนซ้ำไปเรื่อยๆ วิธีปกติคือ $O(n^2)$ ซึ่งมีวิธีการที่สามารถแก้ปัญหานี้ได้อย่างเร็วขึ้นคือการเรียงลำดับก่อน จากนั้นวนเพื่อนับจำนวนซ้ำจากลำดับที่ถูกเรียง ซึ่งวิธีนี้หากใช้ sorting algorithm ที่มีประสิทธิภาพ จะสามารถลด complexity เหลือเพียง $O(n \log n)$ ได้ ซึ่ง algorithm นี้เรียกว่า PresortElementUniqueness

Algorithm 1 PresortElenentUniqueness(A)

Require: A, an arbitrary array

```
Sort the array A                          ▷ You can sort array by any method
for i from 0 to Length(A) -2 do
    if A[i] = A[i + 1] then
        Return False                      ▷ Duplicates found, not all elements are unique
    end if
end for
Return True                               ▷ No duplicates found
```

โดยในโจทย์ข้อนี้จะให้นักศึกษาได้ประยุกต์ใช้ `PresortElementUniqueness(A)` ในการลำดับของตัวเลขทั้งหมดไม่มีจำนวนซ้ำ โดยขั้นตอนของการ เรียงลำดับนั้นสามารถใช้ algorithm ไหนก็ได้แต่ต้องทำให้ได้ algorithm ที่มีขนาดน้อยกว่า $O(n^2)$ และ ห้ามใช้ function สำเร็จรูปในการเรียงลำดับทุกกรณีรวมถึง `merge()` แต่ยกเว้น `swap()`

งานของนักศึกษา

ให้เขียนโปรแกรมเพื่อรับค่าลำดับจากผู้ใช้ จากนั้นแสดงตัวเลขในลำดับที่ลบจำนวนซ้ำออกไป เช่น 3 4 4 5 จะได้ 3 4 5 โดยที่ 4 ที่ซ้ำอีกตัวจะถูกลบออกจากลำดับ

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	จำนวนของลำดับที่รับเข้าทั้งหมด (n)
บรรทัดที่ 2	ลำดับตัวเลข n_i

ข้อมูลส่งออก (Output)

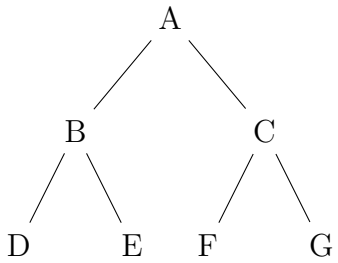
บรรทัดที่ 1	ตัวเลขในลำดับที่ไม่เกิดการซ้ำกันที่เรียงจากน้อยไปมาก

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
10 1 1 1 2 2 2 2 3 4 5	1 2 3 4 5
5 5 5 5 4 3	3 4 5
6 6 6 6 6 6 6	6

Heap Sort

Binary Heap เป็น Data Structure ประเภทหนึ่งที่ประยุกต์หลักการของ **Complete Binary Tree** เพื่อเก็บข้อมูลต่าง ๆ *(หวังว่าเรายังจำเรื่อง Binary Tree จากวิชา CPE112 ได้นะ)*



โดย Binary Heap จะมี 2 ประเภท นั่นคือ

- Min Heap
- Max Heap

Min Heap คือการที่โหนดแต่ละโหนดมีเงื่อนไขที่ว่า Parent Node ต้องมีค่าน้อยกว่าหรือเท่ากับ Child Node ของมันเสมอ ($A \leq B$ และ $A \leq C$) ส่วน Max Heap คือการที่โหนดแต่ละโหนดมีเงื่อนไขว่า Parent Node ต้องมีค่ามากกว่าหรือเท่ากับ Child Node ของมันเสมอ ($A \geq B$ และ $A \geq C$)

หากเราสังเกตจากนิยามด้านบนแล้ว ค่าที่มากที่สุดใน Max Heap จะต้องเป็น Root ของต้นไม้ กล่าวคือ A จะต้องมีค่ามากที่สุด

ในทำนองเดียวกัน ค่าที่น้อยที่สุดใน Min Heap ก็จะต้องเป็น Root ของต้นไม้เหมือนกัน หากว่าต้นไม้ด้านบนเป็น Min Heap จะได้ว่า A คือค่าที่น้อยที่สุดใน Heap นั่นเอง

โดยหากเราได้รับ Array ที่มีสมาชิกทั้งหมด n ตัว หากเราต้องการแปลง Array ให้กลายเป็น Max Heap เราสามารถใช้ขั้นตอนวิธีการดังนี้

Algorithm 2 MaxHeapBottomUp($A[1 \dots n]$)

Require: $A[1 \dots n]$, an arbitrary array (which index **starts at 1**)

```
for i ← ⌊n/2⌋ to 1 do
    k ← i
    v ← A[k]
    heap ← False
    while not heap & (2 × k) ≤ n do
        j ← (2 × k)
        if j < n then                ▷ This mean there are two children of this node
            if A[j] < A[j + 1] then
                j ← j + 1
            end if
        end if
        if v ≥ A[j] then
            heap ← True
        else
            A[k] ← A[j]
            k ← j
        end if
    end while
    A[k] ← v
end for
```

ยกตัวอย่างเช่น เรามีสมาชิกใน Array ทั้งหมด 10 ตัว ดังนี้ $\{52, 60, 45, 23, 1, 13, 70, 48, 90, 84\}$ เมื่อเรานำ Array ไปทำ Heapify ให้ได้ Max Heap เราจะได้ผลการทำ Heapify ดังนี้ $\{90, 84, 70, 60, 52, 13, 45, 48, 23, 1\}$

จากเงื่อนไขด้านบนแล้ว เราสามารถเรียงลำดับ Array จากน้อยไปมากหรือมากไปน้อยได้ โดยการประยุกต์ใช้ Heap เข้าช่วย ซึ่งจะได้วิธีการ Sort แบบใหม่ เรียกว่า **Heap Sort** นั่นเอง โดยขั้นตอนของการทำ Heap Sort จะมีขั้นตอนดังนี้

1. เริ่มจากให้ตัวแปร i มีค่าเป็น n เมื่อ n คือจำนวนสมาชิกใน Array
2. ทำการ **Heapify** Array ของเรา โดยให้ขอบเขตการมองเห็นสมาชิก Array คือ 1 ถึง i (*สังเกตว่าเราจะได้สมาชิกที่มีค่ามากที่สุดเป็นตัวแรกแล้ว ก็คือ* `arr[0]`)
3. ทำการ**สลับ** `arr[0]` และ `arr[i]` (*สังเกตว่าเราจะทำให้ตัวที่มีค่ามากที่สุด ณ ตอนนี้ สลับไปอยู่ท้ายสุด*)
4. ลดค่า i ทีละหนึ่ง
5. ทำซ้ำข้อ 2 - 4 เมื่อค่า i ยังมากกว่า 0

ท้ายที่สุดแล้วเราก็จะได้ Array ที่เรียงจากน้อยไปมากแล้วนั่นเอง

งานของนักศึกษา

จงนำสิ่งที่ได้เรียนรู้มาใน Lab Sheet มาปรับให้เล็กน้อย โดยให้นักศึกษาทำการ Heap Sort Array จาก**มากไปน้อย** โดยใช้การ Heapify ให้กลายเป็น Min Heap *(เปลี่ยน Pseudocode ด้านบน เพียงเล็กน้อยเท่านั้น)*

ข้อมูลนำเข้า (Input)

บรรทัดที่ 1	จำนวนของข้อมูล (n) โดยที่ $1 \leq n \leq 1,000,000$
บรรทัดที่ 2	ลำดับตัวเลข A_i

ข้อมูลส่งออก (Output)

บรรทัดที่ 1	ผลจากการทำ Min Heapify **ครั้งแรก**
บรรทัดที่ 2	ลำดับของเลขที่เรียงจาก**มากไปน้อย**

ตัวอย่างข้อมูลนำเข้า ส่งออก (Examples of Input & Output)

Input	Output
10 52 60 45 23 1 -13 70 48 90 84	-13 1 45 23 60 52 70 48 90 84 90 84 70 60 52 48 45 23 1 -13
5 5 4 3 2 1	1 2 3 5 4 5 4 3 2 1
1 60	60 60

อธิบายตัวอย่าง

ในตัวอย่างแรก เมื่อข้อมูลนำเข้าเป็น `52 60 45 23 1 -13 70 48 90 84` ในขั้นตอนแรกของ Heap Sort คือการนำ Array ไปทำ Heapify ผลลัพธ์ในบรรทัดที่ 1 คือผลที่ได้ของ Array จากการทำ Min Heapify ครั้งแรกสุด ซึ่งจะได้ `-13 1 45 23 60 52 70 48 90 84` และบรรทัดที่ 2 คือ Array ที่เกิดจากการเรียงจากมากไปน้อยโดยการใช้ Heap Sort ก็คือ `90 84 70 60 52 48 45 23 1 -13`